

Débuter avec Zend Framework 1.10 (approche MVC)

par Rob Allen ([Auteur](#)) Guillaume Rossolini ([Traducteur](#)) Sylvain Jorge Do Marco ([Traducteur](#))


Date de publication : 22/11/2010


Dernière mise à jour : 22/11/2010

Cet article est la traduction de l'excellent tutoriel de Rob Allen :  **Getting started with the Zend Framework (Document Revision 1.7.5)**.

Commentez :

Ce cours est une introduction à l'utilisation du Zend Framework, en créant une application très simple utilisant une base de données basée sur le paradigme Modèle-Vue-Contrôleur.

 Ce tutoriel a été testé avec la versions 1.10.1 du Zend Framework. Il a de bonnes chances de fonctionner avec des versions plus récentes de la série 1.x, mais il ne fonctionnera pas avec des versions antérieures à la 1.10.1. Les traductions des tutoriels équivalents pour les versions antérieures du Zend Framework sont toujours disponibles : **Débuter avec Zend Framework 1.5 (approche MVC)** et **Débuter avec Zend Framework (versions 1.8 et 1.9) (approche MVC)**.

 Si vous obtenez des erreurs 404 en allant sur toute autre page que la page d'accueil, assurez-vous s'il vous plait que vous avez défini AllowOverride à All dans votre configuration d'Apache et que vous avez activé l'extension mod_rewrite.

Si vous le déployez sur un site "utilisateur" (par exemple `http://localhost/~rob`), alors vous avez besoin d'une ligne `RewriteBase` dans votre fichier `.htaccess` qui ressemble à ça :

```
RewriteBase /~rob/tutoriel-zf/public/.
```

I - Pré-requis.....	4
II - Pré-suppositions.....	4
III - Récupérer le framework.....	4
IV - Configurer Zend_Tool.....	4
IV-A - Zend_Tool pour Windows.....	4
IV-B - Zend_Tool pour OS X (similaire sous Linux).....	4
IV-C - Tester Zend_Tool.....	5
V - L'application du tutoriel.....	5
V-A - Créer le projet.....	5
V-B - Bootstrapping en tâche de fond.....	8
V-C - Editer le fichier application.ini.....	8
VI - Le code spécifique de l'application.....	8
VII - Mettre en place le contrôleur.....	9
VIII - La base de données.....	10
VIII-A - Configuration de la base de données.....	10
VIII-B - Création de la table de base de données.....	10
VIII-C - Insertion d'albums de test.....	11
IX - Le modèle.....	11
X - Mise en page et vues.....	13
X-A - Code HTML en commun : Mise en page.....	13
X-B - Mise en forme.....	15
XI - Lister les albums.....	16
XII - Ajouter de nouveaux albums.....	17
XIII - Modifier un album.....	19
XIV - Supprimer un album.....	21
XV - Conclusion.....	22

I - Pré-requis

Le Zend Framework a besoin des éléments suivants :

- PHP 5.2.4 (ou ultérieur) ;
- Un serveur Web supportant le `mod_rewrite` ou une fonctionnalité équivalente.

II - Pré-suppositions

J'ai supposé que vous utilisiez PHP 5.2.4 ou ultérieur avec le serveur web Apache. Votre installation d'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 l'instruction `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 l'utilisation de `mod_rewrite` et `.htaccess`.

III - Récupérer le framework

Le Zend Framework peut être téléchargé à l'adresse <http://framework.zend.com/download/latest> au format .zip ou .tar.gz. Regardez en bas de page pour des liens directs. Vous aurez besoin de la version "Minimal".

IV - Configurer Zend_Tool

Le Zend Framework est fourni avec un nouvel outil en ligne de commandes. Nous commençons par le configurer. (1)

IV-A - Zend_Tool pour Windows

- Créez un nouveau répertoire dans *Program Files* appelé *ZendFrameworkCli*
- Double-cliquez sur le fichier archive téléchargé, *ZendFramework-1.10.8-minimal.zip*
- Copiez les répertoires *bin* et *library* du dossier *ZendFramework-1.10.8-minimal.zip* vers le répertoire C:\Program Files\ZendFrameworkCli. Ce répertoire doit maintenant avoir deux sous-répertoires : *bin* et *library*.
- Ajoutez le répertoire *bin* à votre path :
 - Allez à la section Système du Panneau de Configuration.
 - Choisissez Avancé et appuyez sur le bouton Variables d'Environnement.
 - Dans la liste des "Variables système", trouvez la variable *Path* et double-cliquez dessus.
 - Ajoutez ;C:\Program Files\ZendFrameworkCli\bin à la fin de la zone de saisie et appuyez sur Ok (Le point-virgule en première position est important !) (2)
 - Redémarrez

IV-B - Zend_Tool pour OS X (similaire sous Linux)

- Décompressez l'archive téléchargée, *ZendFramework-1.10.8-minimal.zip* dans votre répertoire *Downloads* en double-cliquant dessus.
- Copiez-la vers */usr/local/ZendFrameworkCli* en ouvrant un Terminal et en y saisissant :
`sudo cp -r ~/Downloads/Zendframework-1.10.8-minimal /usr/local/ZendFrameworkCli`
- Modifiez votre profil bash pour ajouter un alias :
 - Depuis le Terminal, tapez : `open ~/.bash_profile`
 - Ajoutez `alias zf=/usr/local/ZendFrameworkCli/bin/zf.sh` à la fin du fichier
 - Sauvegardez et fermez TextEdit.
 - Quittez le Terminal

IV-C - Tester Zend_Tool


Vous pouvez tester votre installation de l'interface en ligne de commandes Zend_Tool en ouvrant un Terminal ou une Invite de Commande et en tapant :

```
zf show version
```

Si tout fonctionne, vous devriez voir :

```
Zend Framework Version : 1.10.8
```

Si ce n'est pas le cas, vérifiez que vous avez correctement défini votre path et que le répertoire *bin* existe bien dans le dossier *ZendFrameworkCli*. Une fois l'outil zf fonctionnel, `zf --help` vous montrera toutes les commandes disponibles.

 *Si votre distribution de PHP est livrée avec le Zend Framework, assurez-vous qu'elle n'utilise pas le ZF 1.9, puisque ce tutoriel ne fonctionnerait pas. Alors que j'écris ces lignes, c'est le cas de la distribution xxamp.*

V - L'application du tutoriel

Tous les éléments étant en place pour construire une application Zend Framework, étudions l'application que nous allons réaliser. Nous allons construire un système d'inventaire très simple pour exposer notre collection de CD. La page principale listera notre collection et nous permettra d'ajouter, modifier ou supprimer des CDs. Comme avec toute conception de logiciel, réaliser un petit pré-programme nous aidera. Nous aurons besoin de quatre pages dans notre site web :

Page d'accueil	Elle affichera la liste des albums avec des liens permettant de les modifier et de les supprimer. Un lien permettant d'ajouter de nouveaux albums sera aussi proposé.
Ajouter un nouvel album	Cette page proposera un formulaire pour ajouter un nouvel album.
Modifier l'album	Cette page proposera un formulaire pour modifier un album.
Supprimer l'album	Cette page confirmera que vous voulez supprimer un album et le supprimera.

Nous aurons également besoin de stocker nos données dans une base de données. Nous aurons juste besoin d'une table avec ces champs :

Nom de champs	Type	Null ?	Notes
id	integer	Non	Clé primaire, auto-incrémentée
artiste	varchar(100)	Non	
titre	varchar(100)	Non	

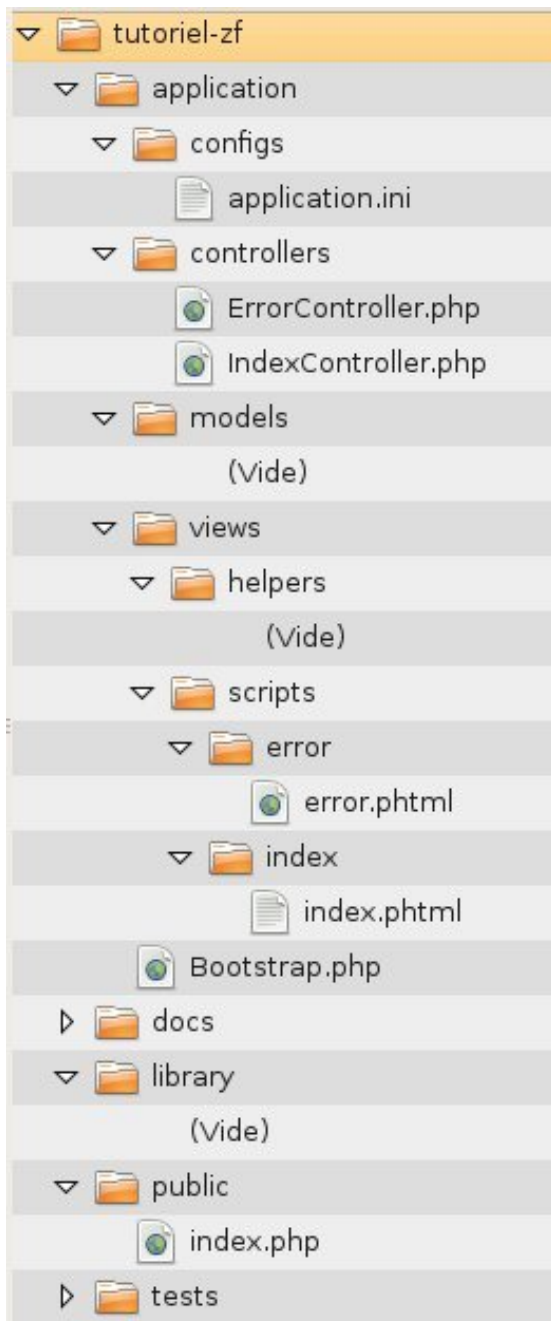
V-A - Créer le projet

Commençons à construire notre application. Quand ce sera possible, nous ferons appel à l'outil en ligne de commandes zf qui nous épargnera du temps et des efforts. Le premier travail consiste à créer les fichiers et répertoires squelette du projet.

Ouvrez un terminal ou une invite de commandes et allez dans le répertoire racine de votre serveur web en utilisant la commande `cd`. Assurez-vous d'avoir la permission de créer des fichiers dans ce répertoire et que le serveur web a le droit de lecture. Entrez :

```
zf create project tutorial-zf
```


L'outil du ZF créera un dossier nommé *tutorial-zf* et le remplira avec la structure de répertoires conseillée. Cette structure suppose que vous ayez le contrôle complet de la configuration d'Apache, de façon à pouvoir laisser la majorité des fichiers en dehors du répertoire web racine. Vous devriez voir les fichiers et répertoires suivants :



(il y a aussi un fichier caché *.htaccess* dans *public*)

Le répertoire *application/* est l'emplacement où réside le code source pour ce site. Comme vous pouvez le constater, nous avons des répertoires séparés pour les fichiers du modèle, de la vue et du contrôleur de notre application. le répertoire *public/* est la racine du site web accessible au public, ce qui signifie que l'URL permettant d'obtenir

l'application sera `http://localhost/tutoriel-zf/public`. De cette façon la majorité des fichiers de l'application ne sont pas directement accessibles par Apache et sont donc plus sécurisés.

 Sur un site de production, vous devriez créer un hôte virtuel pour le site web et définir la racine du site directement au répertoire public. Par exemple vous pourriez créer un hôte virtuel appelé `tutoriel-zf.localhost` qui ressemblerait à ça :

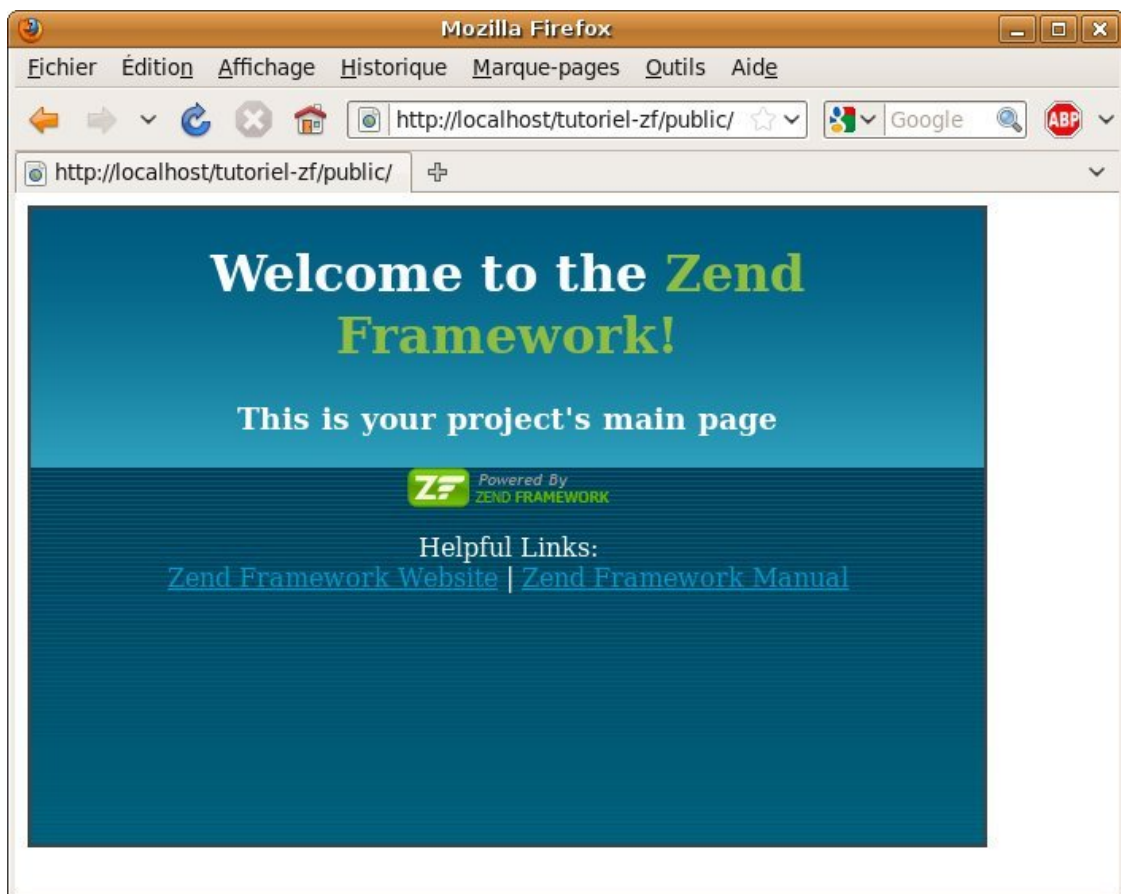
```
<VirtualHost *:80>
    ServerName tutoriel-zf.localhost
    DocumentRoot /var/www/html/tutoriel-zf/public
    <Directory "/var/www/html/tutoriel-zf/public">
        AllowOverride All
    </Directory>
</VirtualHost>
```

Le site serait alors accessible en utilisant `http://tutoriel-zf.localhost/` (assurez-vous d'avoir mis à jour votre fichier `/etc/hosts` ou `C:\Windows\system32\drivers\etc\hosts` de façon à rediriger `tutoriel-zf.localhost` vers `127.0.0.1`). Nous ne le ferons pas dans ce tutoriel car il est aussi simple d'utiliser un sous-répertoire pour tester.


Les fichiers images, JavaScript et CSS indépendants seront stockés dans des répertoires séparés dans le dossier `public/`. Les fichiers téléchargés du Zend Framework seront placés dans le répertoire `library/`. Si nous avons besoin d'utiliser d'autres bibliothèques, nous pourrons aussi les placer à cet endroit.

Copiez le répertoire `library/Zend/` du fichier archive téléchargé (`ZendFramework-1.10.8-minimal.zip`) dans votre `tutoriel-zf/library/`, de telle sorte que votre `tutoriel-zf/library/` contienne un sous-répertoire appelé `Zend/`.

Vous pouvez tester que tout est correct en allant sur <http://localhost/tutoriel-zf/public>. Vous devriez voir quelque chose comme ça :



V-B - Bootstrapping en tâche de fond

Le contrôleur du Zend Framework utilise le  **motif de conception** *Front Controller* et route toutes les requêtes vers un unique fichier *index.php*. Cela assure de correctement définir l'environnement pour exécuter l'application (connu sous le terme de bootstrapping). Nous y parvenons en utilisant un fichier *.htaccess* dans le répertoire *tutoriel-zf/public*, généré pour nous par Zend_Tool, qui redirige toutes les requêtes vers *public/index.php*, également créé par Zend_Tool.

Le fichier *index.php* est le point d'entrée de notre application et est utilisé pour créer une instance de *Zend_Application* pour initialiser notre application et enfin la lancer. Ce fichier définit aussi deux constantes : *APPLICATION_PATH* et *APPLICATION_ENV* qui définissent le chemin vers le répertoire *application/* et l'environnement ou mode de l'application. Il est par défaut défini à production dans *index.php*, mais vous pouvez le positionner à development dans le fichier *.htaccess* en ajoutant cette ligne :

```
SetEnv APPLICATION_ENV development
```

Le composant *Zend_Application* est utilisé pour démarrer l'application et est configuré pour utiliser les directives du fichier de configuration *application/configs/application.ini*. Ce fichier est aussi auto-généré pour nous. Une classe *Bootstrap* qui hérite de *Zend_Application_Bootstrap_Bootstrap* est fournie dans *application/Bootstrap.php* et peut être utilisée pour exécuter tout code spécifique requis au démarrage.

Le fichier *application.ini* situé dans le répertoire *application/configs* est chargé en utilisant le composant *Zend_Config_Ini*. Ce dernier comprend le concept d'héritage de sections quand elles sont décrites en utilisant les deux points dans le nom de section. Par exemple :

```
[staging : production]
```

Cela signifie que la section *staging* hérite de tous les paramètres de la section *production*. La constante *APPLICATION_ENV* définit la section chargée. Evidemment, en développement, la section *development* est la plus appropriée et pour le serveur de production, la section *production* doit être utilisée. Nous mettrons toutes les modifications que nous ferons dans le fichier *application.ini* dans la section *production* de façon à ce que toutes les configurations chargent les modifications que nous effectuons.

V-C - Editer le fichier application.ini

La première modification que nous avons besoin de réaliser est d'ajouter notre fuseau horaire pour les fonctionnalités de date et d'heure de PHP. Modifiez *application/configs/application.ini* et ajoutez :

```
phpSettings.date.timezone = "Europe/Paris"
```

après toutes les autres valeurs *phpSettings* de la section *[production]*. Bien sûr, vous devriez probablement utiliser votre propre fuseau horaire. Nous sommes maintenant prêts à ajouter le code spécifique à notre application.

VI - Le code spécifique de l'application

Avant de mettre en place nos fichiers, il est important de comprendre l'organisation des pages qu'attend le Zend Framework. Chaque page de l'application est connue comme une action et les actions sont regroupées dans des contrôleurs (controllers). Avec une URL au format *http://localhost/public/tutoriel-zf/actualites/voir*, le contrôleur est *Actualites* et l'action est *voir*. Cela permet de regrouper des actions apparentées. Par exemple, un contrôleur *Actualites* peut avoir les actions *lister*, *archives* et *voir*. Le système MVC du Zend Framework supporte aussi les modules pour regrouper les contrôleurs ensemble, mais cette application n'est pas suffisamment importante pour s'en préoccuper.

Par défaut, le contrôleur du Zend Framework utilise une action spéciale appelée `index` comme action par défaut. C'est pour des cas comme `http://localhost/tutoriel-zf/public/actualites/` pour lequel l'action `index` du contrôleur `Actualites` sera exécutée. Il y a aussi un nom de contrôleur par défaut, qui est également appelé `index` et ainsi l'URL `http://localhost/tutoriel-zf/public/` entraînera l'exécution de l'action `index` du contrôleur `index`.

Comme c'est un tutoriel simple, nous ne nous occuperons pas de choses "complexes" comme se connecter ! Cela attendra un tutoriel différent (ou vous pouvez en lire plus dans *Zend Framework in Action* !).

Puisque nous avons quatre pages, toutes s'appliquant aux albums, nous allons les regrouper en un seul contrôleur avec quatre actions. Nous utiliserons le contrôleur par défaut et les actions seront :

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

Au fur et à mesure qu'un site se complexifie, des contrôleurs supplémentaires sont nécessaires et vous pouvez même si besoin en grouper plusieurs dans des modules.

VII - Mettre en place le 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 `application/controllers`. Chaque action est une fonction publique dans la classe du contrôleur et doit être appelée `{nom de l'action}Action`. Dans ce cas, `{nom de l'action}` doit commencer par une lettre minuscule et de nouveau être complètement en minuscules. Les noms d'action et de contrôleur contenant des majuscules sont permis, mais il existe des règles spéciales que vous devez comprendre avant de les utiliser. Commencez par vérifier dans la documentation !

Notre classe contrôleur s'appelle `IndexController`, est définie dans `application/controllers/IndexController.php` et a été automatiquement créée pour nous par `Zend_Tool`. Elle contient aussi la première méthode d'action, `indexAction()`. Nous avons juste besoin d'ajouter nos actions supplémentaires.

Ajouter des actions supplémentaires se fait en utilisant la commande `create action` de l'outil en ligne de commandes `zf`. Ouvrez un terminal ou une invite de commandes et placez vous dans votre répertoire `tutoriel-zf/`. Entrez ensuite ces trois commandes :

- `zf create action ajouter Index`
- `zf create action modifier Index`
- `zf create action supprimer Index`


Ces commandes créent trois nouvelles méthodes : `ajouterAction`, `modifierAction` et `supprimerAction` dans `IndexController` et créent aussi les fichiers de script de vue appropriés dont nous aurons besoin plus tard. Nous avons maintenant mis en place les quatre actions que nous voulons utiliser.

Les URLs de chaque action sont :

URL	Méthode d'action
http://localhost/tutoriel-zf/public/	IndexController::indexAction()
http://localhost/tutoriel-zf/public/index/ajouter	IndexController::ajouterAction()
http://localhost/tutoriel-zf/public/index/modifier	IndexController::modifierAction()
http://localhost/tutoriel-zf/public/index/supprimer	IndexController::supprimerAction()

Vous pouvez tester ces trois nouvelles actions, et devriez voir un message comme celui-ci :

View script for controller **Index** and script/action name **ajouter**

 Si vous avez une erreur 404, c'est que vous n'avez pas activé le module `mod_rewrite` d'Apache ou que vous n'avez pas défini correctement la directive `AllowOverride` dans votre fichier de configuration Apache de façon à ce que le fichier `.htaccess` du répertoire `public/` soit véritablement utilisé.

VIII - La base de données

Maintenant que nous avons le squelette de l'application avec les méthodes d'action du contrôleur et les fichiers de vue prêts, il est temps de regarder le modèle de notre application. Rappelez-vous que le modèle est la partie qui s'occupe de l'objectif central de l'application (communément appelé la *logique applicative*) et, dans notre cas, s'occupe de la base de données. Nous nous servons de la classe du Zend Framework `Zend_Db_Table` qui est utilisée pour trouver, ajouter, modifier ou supprimer des enregistrements de table de base de données.

VIII-A - Configuration de la base de données

Pour utiliser `Zend_Db_Table`, nous avons besoin de lui indiquer quelle base de données utiliser, avec un utilisateur et un mot de passe. Puisque nous préférons ne pas coder en dur ces informations dans l'application, nous allons utiliser un fichier de configuration pour les gérer. Le composant `Zend_Application` est fourni avec une ressource de configuration de base de données, nous avons donc juste besoin de définir les informations appropriées dans le fichier `configs/application.ini` et il s'occupera du reste.

Ouvrez `application/configs/application.ini` et ajoutez ce qui suit à la fin de la section `[production]` (i.e. avant la section `[staging]`) :

```
resources.db.adapter = PDO_MYSQL
resources.db.params.host = localhost
resources.db.params.username = rob
resources.db.params.password = 123456
resources.db.params.dbname = tutoriel-zf
```

Vous devez évidemment utiliser votre nom d'utilisateur, votre mot de passe et votre nom de base de données, pas les miens ! La connexion à la base de données sera maintenant établie automatiquement pour nous et l'adaptateur par défaut pour `Zend_Db_Table` sera défini. Vous pouvez trouver des informations sur les autres plugins de ressource disponibles ici : <http://framework.zend.com/manual/fr/zend.application.available-resources.html>.

VIII-B - Création de la table de base de données

Comme indiqué dans le programme initial, nous utiliserons une base de données pour stocker les données de nos albums. J'utiliserai MySQL et donc la requête SQL pour créer la table est la suivante :

```
CREATE TABLE albums (  
    id int(11) NOT NULL auto_increment,  
    artiste varchar(100) NOT NULL,  
    titre varchar(100) NOT NULL,  
    PRIMARY KEY (id)  
);
```

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

VIII-C - Insertion d'albums de test

Nous allons également ajouter quelques enregistrements dans la table de manière à pouvoir tester la fonctionnalité de récupération de la page d'accueil. Je vais prendre quelques CDs "meilleures ventes" à partir d'Amazon UK. Exécutez la requête suivante dans votre client MySQL :

```
INSERT INTO albums (artiste, titre)  
VALUES  
( 'Paolo Nutine', 'Sunny Side Up'),  
( 'Florence + The Machine', 'Lungs'),  
( 'Massive Attack', 'Heligoland'),  
( 'Andre Rieu', 'Forever Vienna'),  
( 'Sade', 'Soldier of Love');
```

Nous avons maintenant quelques données dans la base et pouvons écrire un modèle très simple pour elles.

IX - Le modèle

Le Zend Framework ne fournit pas de classe `Zend_Model` puisque le modèle correspond à notre logique applicative et que c'est à nous de décider comment nous voulons que cela fonctionne. Vous pouvez pour cela utiliser de nombreux composants suivant vos besoins. Une approche est d'avoir des classes de modèle qui représentent chaque entité de votre application puis d'utiliser des objets de liaison qui chargent et sauvegardent les entités dans la base de données. Cette approche est documentée dans le guide de démarrage rapide du Zend Framework ici : <http://framework.zend.com/manual/fr/learning/quickstart.create-model.html>.

Pour ce tutoriel, nous allons créer un modèle qui étend `Zend_Db_Table` et utilise `Zend_Db_Table_Row`. Le Zend Framework fournit `Zend_Db_Table` qui implémente le motif de conception *Table Data Gateway* pour permettre de s'interfacer avec les données d'une table de base de données. Soyez toutefois avertis que le motif *Table Data Gateway* peut s'avérer limité dans des systèmes plus complexes. Il est également tentant de placer le code d'accès à la base de données dans les méthodes d'action du contrôleur tel qu'il est exposé par `Zend_Db_Table`.

`Zend_Db_Table_Abstract` est une classe abstraite, nous devons donc en hériter pour définir notre classe spécifique à la gestion des albums. Peu importe le nom de notre classe, mais il est logique de lui donner le nom de la table de la base de données. Notre projet a un auto-chargement par défaut instancié par `Zend_Application` qui associe les classes de ressource d'un module au répertoire dans lequel elles sont définies. Pour les répertoires racines situés dans *application/* nous utilisons le préfixe `Application_`.

L'auto-chargement associe les ressources aux répertoires selon ce tableau :

Préfixe	Répertoire
Form	forms
Model	models
Model_DbTable_	models/DbTable
Model_Mapper_	models/mappers
Plugin	plugins
Service	services
View_Filter	views/filters
View_Helper	views/helpers

Puisque nous utilisons une appellation fonction de la table de base de données, albums, et que nous utiliserons `Zend_Db_Table`, notre classe sera appelée `Application_Models_DbTable_Albums` et sera stockée dans *application/models/DbTable/Albums.php*.

Pour indiquer à `Zend_Db_Table` le nom de la table qu'il aura à gérer, nous devons définir la propriété protégée `$_name` avec le nom de la table. Ainsi, `Zend_Db_Table` supposera que notre table a une clé primaire appelée `id` auto-incrémentée par la base de données. Le nom de ce champ peut également être modifié si besoin.

Nous pouvons utiliser l'outil en ligne de commandes `zf` pour faire une partie du travail, donc exécutez la commande suivante en ligne de commandes :

```
zf create db-table Albums albums
```

L'outil en ligne de commandes vient de créer le fichier *Albums.php* dans le répertoire *application/models/DbTable*. On y trouve une classe appelée `Application_Model_DbTable_Albums` dans laquelle est défini le nom de la table de base de données avec laquelle cette classe communique.

Nous devons maintenant ajouter quelques fonctionnalités, donc éditez *application/models/DbTable/Albums.php* et ajoutez les méthodes `obtenirAlbum()`, `ajouterAlbum()`, `modifierAlbum()` et `supprimerAlbum()` de manière à obtenir le code suivant :

tutoriel-zf/application/models/DbTable/Albums.php

```
<?php

class Application_Model_DbTable_Albums extends Zend_Db_Table_Abstract
{
    protected $_name = 'albums';

    public function obtenirAlbum($id)
    {
        $id = (int)$id;
        $row = $this->fetchRow('id = ' . $id);
        if (!$row) {
            throw new Exception("Impossible de trouver l'enregistrement $id");
        }
        return $row->toArray();
    }

    public function ajouterAlbum($artiste, $titre)
    {
        $data = array(
            'artiste' => $artiste,
            'titre' => $titre,
        );
        $this->insert($data);
    }

    public function modifierAlbum($id, $artiste, $titre)
    {
        $data = array(
            'artiste' => $artiste,
            'titre' => $titre,
        );
    }
}
```

tutoriel-zf/application/models/DbTable/Albums.php

```
);  
$this->update($data, 'id = ' . (int)$id);  
}  
  
public function supprimerAlbum($id)  
{  
    $this->delete('id = ' . (int)$id);  
}  
}
```

Nous créons quatre méthodes d'aide que notre application utilisera pour s'interfacer avec la table de base de données. obtenirAlbum() récupère un enregistrement unique dans un tableau, ajouterAlbum() crée un nouvel enregistrement dans la base de données, modifierAlbum() met à jour un enregistrement d'album et supprimerAlbum() supprime complètement l'enregistrement. Le code pour chacune de ces méthodes est explicite. Même si ce n'est pas utile dans ce tutoriel, vous pouvez aussi indiquer à Zend_Db_Table les tables liées et il peut récupérer également des données liées.

Nous devons remplir les contrôleurs avec les données du modèle et demander aux scripts de vue de les afficher. Cependant, avant de le faire, nous devons comprendre le fonctionnement du système de vue du Zend Framework.

X - Mise en page et vues

Le composant de vue du Zend Framework est, sans surprise, appelé Zend_View. Le composant de vue nous permettra de séparer le code qui affiche une page du code des méthodes d'action.

L'utilisation basique de Zend_View est la suivante :

```
$view = new Zend_View();  
$view->setScriptPath('/chemin/vers/les/scripts');  
echo $view->render('script.php');
```

Vous voyez très facilement que si nous mettions ce squelette directement dans chacune de nos méthodes d'action nous répéterions du code "structurel" très ennuyeux qui n'a pas d'intérêt pour l'action. Nous devrions plutôt initialiser la vue ailleurs et ensuite accéder à notre objet de vue déjà initialisé dans chaque méthode d'action. Le Zend Framework nous fournit une aide d'action appelée ViewRenderer. Elle prend soin d'initialiser une propriété de vue dans le contrôleur (\$this->view) pour que nous l'utilisions et elle "rend" également un script de vue après traitement de l'action.

Pour le rendu, le ViewRenderer demande à l'objet Zend_View de rechercher les scripts de vue à rendre dans *views/scripts/{nom du contrôleur}* et rendra (au moins par défaut) le script nommé comme l'action avec l'extension phtml. Ainsi, le script de vue rendu est *views/scripts/{nom du contrôleur}/{nom de l'action}.phtml* et les contenus rendus sont ajoutés au corps de l'objet réponse. L'objet réponse est utilisé pour rassembler tous les en-têtes HTTP, le corps du contenu et les exceptions générées par l'utilisation du système MVC. Le contrôleur frontal envoie automatiquement les en-têtes suivis par le contenu du corps à la fin de la répartition (dispatch).

Tout est mis en place pour nous par Zend_Tool quand nous créons le projet et ajoutons des contrôleurs et des actions en utilisant les commandes `zf create controller` et `zf create action`.

X-A - Code HTML en commun : Mise en page

Il devient très vite évident que nous aurons beaucoup de code HTML en commun dans nos vues pour les sections d'en-tête et de pied de page, et peut-être aussi une ou deux barres latérales également. C'est un problème très classique, et le composant Zend_Layout est conçu pour y remédier. Zend_Layout nous permet de déplacer tout le code commun des en-têtes, pieds de page et autre vers un script de "mise en page" (layout) qui se charge alors d'inclure le code de vue spécifique de l'action en cours d'exécution.

L'emplacement par défaut pour stocker nos mises en page est *application/layouts/* et il existe une ressource de Zend_Application qui configurera Zend_Layout pour nous. Nous utilisons Zend_Tool pour créer le fichier de script de vue de mise en page et mettre à jour convenablement notre fichier *application.ini*. De nouveau, depuis un Terminal ou l'invite de commandes, entrez les commandes suivantes dans votre répertoire *tutoriel-zf* :

```
zf enable layout
```

Zend_Tool a maintenant créé le répertoire *application/layouts/scripts* et y a placé un script de vue appelé *layout.phtml*. Il a également mis à jour le fichier *application.ini* et ajouté la ligne `resources.layout.layoutPath = APPLICATION_PATH "/layouts/scripts/"` à la section [production].

À la fin du cycle de répartition (dispatch), une fois les méthodes d'action des contrôleurs terminées, Zend_Layout rendra notre mise en page. Zend_Tool fournit un fichier de mise en page très basique qui se contente d'afficher le contenu du script de vue de l'action. Nous le compléterons avec le HTML requis pour notre site. Ouvrez *layout.phtml* et remplacez son code par celui-ci :

tutoriel-zf/application/layouts/scripts/layout.phtml

```
<?php
$this->headMeta()->appendHttpEquiv('Content-Type', 'text/html; charset=utf-8');
$this->headTitle()->setSeparator(' - ');
$this->headTitle('Tutoriel Zend Framework');

echo $this->doctype(); ?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
    <?php echo $this->headMeta(); ?>
    <?php echo $this->headTitle(); ?>
</head>
<body>
    <div id="content">
        <h1><?php echo $this->escape($this->title); ?></h1>
        <?php echo $this->layout()->content; ?>
    </div>
</body>
</html>
```

Le fichier de mise en page contient le code HTML "externe" qui est tout ce qu'il y a de plus classique. Comme c'est un fichier PHP normal, nous pouvons utiliser du PHP à l'intérieur. Une variable disponible, `$this`, est une instance de l'objet de vue créé pendant le *bootstrapping*. Nous pouvons l'utiliser pour récupérer les données affectées à la vue et aussi pour appeler des méthodes. Les méthodes (connues comme des "aides de vue") retournent une chaîne de caractères que nous pouvons alors afficher.

Pour commencer, nous utilisons quelques aides de vue pour la section `<head>` de la page web puis affichons le doctype adéquat. Dans le `<body>`, nous créons une division (div) avec un `<h1>` contenant le titre. Pour rendre le script de vue de l'action en cours, nous affichons l'emplacement virtuel content en utilisant l'aide de vue `layout()` : `echo $this->layout()->content;` qui fait le travail pour nous. Cela implique que les scripts de vue de l'action soient exécutés avant le script de vue de mise en page.

Nous devons définir le doctype de la page web avant de rendre le moindre script de vue. Les scripts de vue d'action sont en effet rendus avant et peuvent avoir besoin de connaître le doctype en vigueur. C'est notamment vrai pour Zend_Form.

Pour définir le doctype nous ajoutons une nouvelle ligne à notre *application.ini*, dans la section [production] :

```
resources.view.doctype = "XHTML1_STRICT"
```

L'aide de vue doctype affichera maintenant le doctype correct et les composants comme Zend_Form généreront du HTML compatible.

X-B - Mise en forme

Bien qu'il s'agisse juste d'un tutoriel, nous aurons besoin d'un fichier CSS pour que notre application soit un minimum présentable ! Cela pose un problème mineur, car nous ne savons pas réellement comment référencer le fichier CSS puisque l'URL n'indique pas le bon répertoire racine. Heureusement, une aide de vue appelée `baseUrl()` est disponible. Cette aide récupère l'information dont nous avons besoin à partir de l'objet de requête et nous donne la partie de l'URL que nous ne connaissons pas.

Nous pouvons maintenant ajouter le fichier CSS à la section `<head>` du fichier `application/layouts/scripts/layout.phtml` et nous utilisons de nouveau une aide de vue, `headLink()` :

tutoriel-zf/application/layouts/layout.phtml

```
...
<head>
    <?php echo $this->HeadMeta(); ?>
    <?php echo $this->headTitle(); ?>
    <?php echo $this->headLink()->prependStylesheet($this->baseUrl().'/css/site.css'); ?>
</head>
...
```

En utilisant la méthode `prependStylesheet()` de `headLink()`, nous permettons à des fichiers CSS supplémentaires, plus spécifiques, d'être ajoutés dans les scripts de vue du contrôleur, et ils seront alors ajoutés à la section `<head>` après `site.css`.

Pour terminer, nous avons besoin de quelques styles CSS, donc créez un répertoire `css` dans `public/` et ajoutez `site.css` avec ce code :

tutoriel-zf/public/css/site.css

```
body,html {
    margin: 0 5px;
    font-family: Verdana,sans-serif;
}

h1 {
    font-size: 1.4em;
    color: #008000;
}

a {
    color: #008000;
}

/* Table */
th {
    text-align: left;
}

td, th {
    padding-right: 5px;
}

/* style form */
form dt {
    width: 100px;
    display: block;
    float: left;
    clear: left;
}

form dd {
    margin-left: 0;
    float: left;
}

form #submitButton {
```


tutoriel-zf/public/css/site.css

```
margin-left: 100px;
}
```

Cela devrait le rendre un peu plus joli, mais comme vous le remarquez, je ne suis pas un designer !

Nous pouvons maintenant nettoyer les quatre scripts d'action qui ont été auto-générés pour nous prêts à être remplis, donc allez-y et videz les fichiers *index.phtml*, *ajouter.phtml*, *modifier.phtml* et *supprimer.phtml* qui, comme vous vous souvenez sans doute, sont dans le répertoire *application/views/scripts/index*.

XI - Lister les albums

Maintenant que nous avons mis en place la configuration, les informations de la base de données et nos squelettes de vues, nous pouvons entrer dans le vif du sujet et afficher quelques albums. C'est fait dans la classe *IndexController* et nous commençons par lister les albums de la table dans la méthode *indexAction()* :

tutoriel-zf/application/controllers/IndexController.php

```
...
function indexAction()
{
    $albums = new Application_Model_DbTable_Albums();
    $this->view->albums = $albums->fetchAll();
}
...
```

Nousinstancions notre modèle basé sur le motif *Table Data Gateway*. La méthode *fetchAll()* retourne un *Zend_Db_Table_Rowset* qui nous permettra de parcourir les enregistrements retournés dans le script de vue de l'action.

Nous pouvons maintenant remplir le script de vue associé, *index.phtml*.

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

```
<?php
$this->title = 'Mes Albums';
$this->headTitle($this->title);
?>
<p><a href="<?php echo $this->url(array('controller'=>'index',
    'action'=>'ajouter'));">Ajouter de nouveaux albums</a></p>
<table>
    <tr>
        <th>Titre</th>
        <th>Artiste</th>
        <th>&nbsp;</th>
    </tr>
    <?php foreach($this->albums as $album) : ?>
        <tr>
            <td><?php echo $this->escape($album->titre); ?></td>
            <td><?php echo $this->escape($album->artiste); ?></td>
            <td>
                <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>
```

Nous commençons par définir le titre de la page (utilisé dans la mise en page) et nous définissons aussi le titre de la section *<head>* avec l'aide de vue *headTitle* qui l'affichera dans la barre de titre du navigateur. Nous créons ensuite un lien pour ajouter un nouvel album. L'aide de vue *url()* est fournie par le framework et crée des adresses en utilisant

la bonne URL de base. Nous lui passons simplement un tableau des paramètres dont nous avons besoin et elle s'occupe du reste dans la mesure du besoin.

Nous créons ensuite un tableau HTML pour afficher le titre de l'album, l'artiste et des liens permettant de modifier ou de supprimer l'enregistrement. Une boucle classique foreach: est utilisée pour parcourir la liste des albums, et nous utilisons la syntaxe alternative avec deux points et endforeach; car elle est plus simple à lire que d'essayer de faire correspondre des accolades. L'aide de vue url() est là aussi utilisée pour créer les liens de modification et de suppression.

Si vous ouvrez <http://localhost/tutoriel-zf/public> (ou toute adresse dérivée que vous avez choisie !) alors vous devriez maintenant voir une belle liste d'albums, quelque chose comme ça :



XII - Ajouter de nouveaux albums

Nous pouvons maintenant programmer une fonctionnalité pour ajouter de nouveaux albums.

Il y a deux étapes à cette partie :

- Afficher un formulaire pour que l'utilisateur fournisse des détails ;
- Traiter la soumission du formulaire et enregistrer en base de données.

Nous utilisons Zend_Form pour y parvenir. Ce composant nous permet de créer un formulaire et d'en valider la saisie. Nous créons une nouvelle classe Application_Form_Album qui étend Zend_Form pour définir notre formulaire. Comme c'est une ressource de l'application, la classe est stockée dans le fichier *Album.php* dans le répertoire *forms*. Nous commençons par utiliser le script en ligne de commandes zf pour créer le bon fichier :

```
zf create form Album
```

Cela crée le fichier *Album.php* dans *application/forms* et ajoute une méthode *init()* où nous pouvons paramétrer le formulaire et ajouter les éléments dont nous avons besoin. Editez *application/forms/Album.php*, supprimez le commentaire de la méthode *init()* et ajoutez ce qui suit :

tutoriel-zf/application/forms/Album.php

```
<?php

class Application_Form_Album extends Zend_Form
{
    public function init()
    {
        $this->setName('album');
    }
}
```

tutoriel-zf/application/forms/Album.php

```
$id = new Zend_Form_Element_Hidden('id');
$id->addFilter('Int');

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

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

$envoyer = new Zend_Form_Element_Submit('envoyer');
$envoyer->setAttrib('id', 'boutonenvoyer');

$this->addElements(array($id, $artiste, $titre, $envoyer));
}
```

Dans la méthode `init()` de `Application_Form_Album`, nous créons quatre éléments pour l'id, l'artiste, le titre et le bouton d'envoi. Chaque élément se voit affecter certains attributs, notamment le libellé à afficher. Pour l'élément id, nous voulons nous assurer que c'est uniquement un entier pour prévenir d'éventuels problèmes d'injection SQL. Le filtre `Int` le fera pour nous.

Pour les éléments textuels, nous ajoutons deux filtres `StripTags` et `StringTrim` pour supprimer le HTML non désiré et les espaces inutiles. Nous les rendons également obligatoires et leur ajoutons un validateur `NotEmpty` (non vide) pour nous assurer que l'utilisateur entre véritablement l'information dont nous avons besoin (le validateur `NotEmpty` n'est techniquement pas nécessaire puisqu'il sera ajouté automatiquement par le système en ayant défini `setRequired` à `true` ; il existe ici comme une démonstration sur la façon d'ajouter un validateur.).

Nous avons maintenant besoin d'afficher le formulaire puis de le traiter après soumission. Cela se fait dans l'action `ajouterAction()` du contrôleur `IndexController` :

tutoriel-zf/application/controllers/IndexController.php

```
...
function ajouterAction()
{
    $form = new Application_Form_Album();
    $form->envoyer->setLabel('Ajouter');
    $this->view->form = $form;

    if ($this->getRequest()->isPost()) {
        $formData = $this->getRequest()->getPost();
        if ($form->isValid($formData)) {
            $artiste = $form->getValue('artiste');
            $titre = $form->getValue('titre');
            $albums = new Application_Model_DbTable_Albums();
            $albums->ajouterAlbum($artiste, $titre);

            $this->_helper->redirector('index');
        } else {
            $form->populate($formData);
        }
    }
}
...
```

Examinons ça plus en détail :

```
$form = new Application_Form_Album();  
$form->envoyer->setLabel('Ajouter');  
$this->view->form = $form;
```

Nousinstancions notre `Application_Form_Album`, nous affectons au bouton d'envoi le libellé "Ajouter" puis nous assignons le formulaire à la vue pour affichage.

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

Si la méthode `isPost()` de l'objet de requête renvoie `true`, alors le formulaire a été envoyé. Nous récupérons alors les données de la requête avec la méthode `getPost()` et nous vérifions qu'elles sont valides avec la méthode membre `isValid()`.

```
$artiste = $form->getValue('artiste');  
$titre = $form->getValue('titre');  
$albums = new Application_Model_DbTable_Albums();  
$albums->ajouterAlbum($artiste, $titre);
```

Si le formulaire est valide, nousinstancions la classe modèle `Application_Model_DbTable_Albums` et nous utilisons la méthode `ajouterAlbum()` que nous avons créée plus tôt pour créer un nouvel enregistrement dans la base de données.

```
$this->_helper->redirector('index');
```

Après avoir sauvegardé le nouvel enregistrement d'album, nous redirigeons vers l'action `index` avec l'aide d'action `Redirector` (i.e. nous retournons à la page d'accueil).

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

Si les données du formulaire ne sont pas valides, nous le remplissons avec les données fournies et nous l'affichons à nouveau (3).

Nous avons maintenant besoin d'afficher le formulaire dans le script de vue `ajouter.phtml` :

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

```
<?php  
$this->title = "Ajouter un nouvel album";  
$this->headTitle($this->title);  
echo $this->form;  
?>
```

Comme vous pouvez le constater, afficher un formulaire est très simple - nous avons juste à utiliser `echo` dessus, puisqu'il sait comment s'afficher lui-même. Vous devriez maintenant pouvoir utiliser le lien "Ajouter de nouveaux albums" de la page d'accueil de l'application pour ajouter un nouvel enregistrement d'album.

XIII - Modifier un album

Modifier un album est presque identique à en ajouter un, le code est donc vraiment similaire :

tutoriel-zf/application/controllers/IndexController.php

```
...  
function modifierAction()  
{  
    $form = new Application_Form_Album();  
    $form->envoyer->setLabel('Sauvegarder');  
    $this->view->form = $form;
```

tutoriel-zf/application/controllers/IndexController.php

```
if ($this->getRequest()->isPost()) {
    $formData = $this->getRequest()->getPost();
    if ($form->isValid($formData)) {
        $id = $form->getValue('id');
        $artiste = $form->getValue('artiste');
        $titre = $form->getValue('titre');
        $albums = new Application_Model_DbTable_Albums();
        $albums->modifierAlbum($id, $artiste, $titre);

        $this->_helper->redirector('index');
    } else {
        $form->populate($formData);
    }
} else {
    $id = $this->_getParam('id', 0);
    if ($id > 0) {
        $albums = new Application_Model_DbTable_Albums();
        $form->populate($albums->obtenirAlbum($id));
    }
}
...
}
```

Regardons les différences avec l'ajout d'un album. Premièrement, lorsque nous affichons le formulaire, nous devons récupérer l'artiste et le titre de l'album à partir de la base de données puis renseigner les éléments du formulaire avec. C'est à la fin de la méthode :

```
$id = $this->_getParam('id', 0);
if ($id > 0) {
    $albums = new Application_Model_DbTable_Albums();
    $form->populate($albums->obtenirAlbum($id));
}
```

Notez que ce n'est exécuté que si la requête n'est pas de type POST, puisqu'un POST implique que le formulaire a été rempli et que nous voulons le traiter. Pour l'affichage initial du formulaire, nous récupérerons l'id de la requête en utilisant la méthode `_getParam()`. Nous utilisons ensuite le modèle pour récupérer l'enregistrement de la base de données et remplir le formulaire directement avec les données de l'enregistrement (maintenant vous savez pourquoi la méthode `obtenirAlbum()` du modèle retourne un tableau !).

Après validation du formulaire, nous devons sauvegarder les données sur le bon enregistrement de base de données. C'est fait en utilisant notre méthode `modifierAlbum()` du modèle :

```
$id = $form->getValue('id');
$artiste = $form->getValue('artiste');
$titre = $form->getValue('titre');
$albums = new Application_Model_DbTable_Albums();
$albums->modifierAlbum($id, $artiste, $titre);
```

Le script de vue est le même qu'*ajouter.phtml* :

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

```
<?php
$this->title = "Modifier un album";
$this->headTitle($this->title);

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

Vous devriez désormais être en mesure de modifier des albums.

XIV - 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 de nos spécifications HTTP : nous vous rappelons que vous ne devriez pas faire d'action irréversible en utilisant une requête GET mais plutôt POST.

Nous devons afficher un formulaire de confirmation quand l'utilisateur clique sur supprimer et, s'il clique sur "Oui", nous effectuerons la suppression. Puisque ce formulaire est trivial, nous le coderons directement dans notre vue (Zend_Form est facultatif, après tout !).

Commençons par le code de l'action dans `IndexController::supprimerAction()` :

tutoriel-zf/application/controllers/IndexController.php

```
...  
public function supprimerAction()  
{  
    if ($this->getRequest()->isPost()) {  
        $supprimer = $this->getRequest()->getPost('supprimer');  
        if ($supprimer == 'Oui') {  
            $id = $this->getRequest()->getPost('id');  
            $albums = new Application_Model_DbTable_Albums();  
            $albums->supprimerAlbum($id);  
        }  
  
        $this->_helper->redirector('index');  
    } else {  
        $id = $this->getParam('id', 0);  
        $albums = new Application_Model_DbTable_Albums();  
        $this->view->album = $albums->obtenirAlbum($id);  
    }  
}  
...
```

Comme pour l'ajout et la modification, nous utilisons la méthode `isPost()` de la requête pour savoir si nous devons afficher le formulaire ou bien effectuer la suppression. Nous utilisons le modèle `Application_Model_DbTable_Albums` pour réellement supprimer l'enregistrement en utilisant la méthode `supprimerAlbum()`. Si la requête n'est pas de type POST, alors nous recherchons un paramètre `id`, récupérons le bon enregistrement de base de données et l'assignons la vue.

Le script de vue est un simple formulaire :


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



```
<?php  
$this->title = "Supprimer un album";  
$this->headTitle($this->title);  
?>  
<p>Êtes-vous sûr de vouloir supprimer  
    '<?php echo $this->escape($this->album['titre']); ?>' de  
    '<?php echo $this->escape($this->album['artiste']); ?>' ?  
</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="supprimer" value="Oui" />  
    <input type="submit" name="supprimer" value="Non" />  
</div>  
</form>
```


Dans ce script, nous présentons un message de confirmation à l'utilisateur ainsi qu'un formulaire avec les boutons "Oui" et "Non". Dans l'action, nous avons vérifié spécifiquement la valeur "Oui" avant d'exécuter la suppression.

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

XV - Conclusion

Cela conclut notre rapide coup d'oeil à la création d'une application MVC simple, mais parfaitement fonctionnelle, avec le Zend Framework. J'espère que vous l'avez trouvé intéressant et instructif. Si vous trouvez une erreur, veuillez m'envoyer un e-mail à  rob@akrabat.com !

Ce tutoriel a mis l'accent sur les bases d'utilisation du framework ; il y a beaucoup plus de composants à explorer ! J'ai également laissé de côté beaucoup d'explications. Mon site web  <http://akrabat.com> contient de nombreux articles sur le Zend Framework, et vous devriez également  [lire le manuel](#) !

Pour terminer, si vous préférez les pages imprimées, j'ai écrit un livre appelé *Zend Framework in Action* qui est disponible à la vente. Plus de détails sont disponibles sur  <http://www.zendframeworkinaction.com>. Jetez-y un oeil :-)

- 1 : NdT : Pour plus d'informations sur Zend_Tool, vous pouvez vous référer au tutoriel **Créer une application basée sur Zend Framework avec Zend_Tool**.
- 2 : NdT : Le fichier php.exe doit également être dans le path de Windows, et le répertoire *library* du framework dans l'*include_path* de PHP.
- 3 : NdT : Cette étape est inutile avec les versions récentes du Zend Framework qui remplit automatiquement le formulaire lors de l'utilisation de sa méthode `isValid()`.