Démarrer avec Zend Framework

Par Rob Allen, www.akrabat.com
Traduit par Bernard Opic
Version 1.7.6

Ce tutoriel propose une introduction à l'utilisation de Zend Framework en créant une application de base de données simple basée sur le paradigme Modèle-Vue-Contrôleur.

Note : Ce tutoriel a été testé sur les versions 1.10.1 à 1.11.4 de Zend Framework. Il a de grandes chances de fonctionner sur les versions à venir dans la série 1.x, mais ne fonctionnera pas avec les versions antérieures à la 1.10.1.

Exigences

Zend Framework à les exigences suivantes :

- PHP 5.2.4 (ou ultérieur) ;
- Un serveur Web prenant en charge `mod_rewrite` ou une fonctionnalité similaire.

Suppositions du tutoriel

J'ai supposé que vous utilisez PHP 5.2.4 ou ultérieur avec le serveur Web Apache. ***L'extension `mod_rewrite` doit être installée et configurée*** sur votre installation d'Apache.

Vous devez également vous assurer que Apache est configuré pour prendre en charge les fichiers `.htaccess`. Cela se fait généralement en remplaçant le paramètre :

```
AllowOverride None
```

par

```
AllowOverride All
```

dans votre fichier `httpd.conf`. Pour plus de précisions, reportez-vous à la documentation de votre distribution. Vous ne pourrez pas naviguer vers une autre page que la page d'accueil de ce tutoriel si vous n'avez pas configuré correctement l'utilisation de `mod_rewrite` et `.htaccess`.

Obtention du framework

Zend Framework peut être téléchargé au format `.zip` ou `.tar.gz` à l'adresse http://framework.zend.com/download/latest. Regardez au bas de la page pour trouver les liens directs. La version *Minimal* et celle dont vous avez besoin.

Installation de Zend_Tool

Zend Framework est livré avec un nouvel utilitaire en ligne de commande. Nous commencerons par l'installer.

Zend_Tool pour Windows

- Créez un nouveau répertoire `ZendFrameworkCli` dans `Program Files` ;
- Double-cliquez sur le fichier d'archive `ZendFramework-1.10.6-minimal.zip` que vous avez téléchargé ;

- Copiez les répertoires `bin` et `library` de la fenêtre du répertoire `ZendFramework-1.10.6-minimal.zip` vers le répertoire `C:\Program Files\ZendFrameworkCli`. Ce répertoire doit maintenant contenir deux sous-répertoires : `bin` et `library`.
- Ajoutez le répertoire `bin` à vos chemins d'accès :
  - Allez dans la section *Système* du *Panneau de contrôle* ;
  - Choisissez *Avancé* et appuyez sur le bouton *Variables d'environnement* ;
  - Repérez la variable *Path* dans la liste *Variables système* 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 au début est important !) ;
  - Redémarrez.

`Zend_Tool` pour OS X (Linux est similaire)

- Extrayez le fichier d'archive `ZendFramework-1.10.6-minimal.zip` que vous avez téléchargé dans votre répertoire `Downloads` en double-cliquant dessus ;
- Copiez dans `/usr/local/ZendFrameworkCli` en ouvrant un terminal et en saisissant : `sudo cp -r ~/Downloads/ZendFramework-1.10.6-minimal.zip /usr/local/ZendFrameworkCli`
- Éditez votre profil `bash` pour y ajouter un alias :
  - À partir du terminal, saisissez : `open ~/.bash_profile`
  - Ajoutez `alias zf=/usr/local/ZendFrameworkCli/bin/zf.sh` à la fin du fichier ;
  - Sauvez et quittez TextEdit ;
  - Quittez le terminal.

Test de `Zend_Tool`

Vous pouvez tester votre installation de l'interface `Zend_Tool` en ligne de commande en ouvrant un terminal et en saisissant :

```
zf show version
```

Si tout a fonctionné, vous devriez voir :

```
Zend Framework Version 1.10.0
```

Sinon, vérifiez que vous avez bien configuré les chemins d'accès et que le répertoire `bin` existe dans `ZendFrameworkCli`. Lorsque l'utilitaire `zf` fonctionne, `zf --help` liste les commandes disponibles.

Note : Si votre distribution PHP inclut Zend Framework, veuillez vérifier qu'elle n'utilise pas la version 1.9 car ce tutoriel ne fonctionnera pas. Au moment de la rédaction, c'est ce que fait la distribution XXAMP.

L'application du tutoriel

Maintenant que toutes les pièces sont en place pour que nous puissions construire une application Zend Framework, étudions l'application que nous allons réaliser. Nous allons développer un système d'inventaire très simple qui affichera notre collection de CD. La page principale listera notre collection et nous permettra d'ajouter, de modifier et de supprimer des CD. Comme pour n'importe quel projet informatique, il est utile d'avoir un petit pré-planning. Nous allons avoir besoin de quatre pages dans notre site Web :

Page d'accueil	Elle affichera la liste des albums et fournira des liens pour les modifier et les supprimer. Un lien pour l'ajout de nouveaux albums sera également fourni.
Ajouter un nouvel album	Cette page fournira un formulaire d'ajout d'un nouvel album.
Modifier un album	Cette page fournira un formulaire pour la modification d'un album.

Supprimer un album	Cette page confirmera que nous voulons supprimer un album et le supprimera.

Nous aurons aussi besoin d'enregistrer nos données dans une base de données. Nous n'aurons besoin que d'une table contenant ces champs :

Nom du champ	Type	Null autorisé ?	Notes
id	integer	Non	Clé primaire, incrémentation automatique
artist	varchar(100)	Non	
title	varchar(100)	Non	

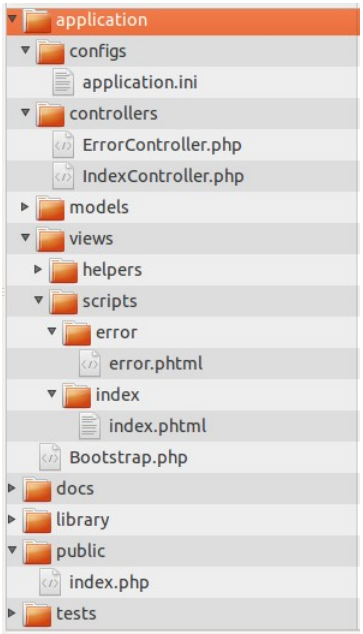
Commençons notre application

Commençons la construction de notre application. Lorsque cela sera possible, nous utiliserons l'utilitaire `zf` en ligne de commande car il permet d'économiser du temps et des efforts. Le premier travail est de créer les fichiers et les répertoires constituant le squelette du projet.

Ouvrez un terminal ou une ligne de commande et placez-vous dans le répertoire racine de votre site Web en utilisant la commande `cd`. Assurez-vous d'avoir les permissions pour créer des fichiers dans ce répertoire et que le serveur Web y a les permissions en lecture. Saisissez :

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

L'utilitaire `zf` créera un répertoire `zf-tutorial` et y placera la structure de répertoires recommandée. Cette structure suppose que vous ayez le contrôle complet de votre configuration Apache, de sorte que vous puissiez placer la plupart des fichiers en-dehors du répertoire racine du site Web. Vous devriez voir les fichiers et répertoires suivants :



(Il y a aussi un fichier `.htaccess` caché dans le répertoire `public/`).

Le répertoire `application/` contient le code source de ce site Web. Comme vous pouvez le voir, nous avons des répertoires séparés pour les fichiers de modèles, de vues et de contrôleurs de notre application. Le répertoire `public/` est la racine du site Web exposée au public, ce qui signifie que l'URL pour accéder à

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

Note : Sur un site Web en production, vous devrez créer un hôte virtuel pour ce site Web et positionner le répertoire racine des documents directement sur le répertoire `public/`. Vous pourriez par exemple créer un hôte virtuel appelé `zf-tutorial.localhost` qui ressemble à quelque chose comme ceci :

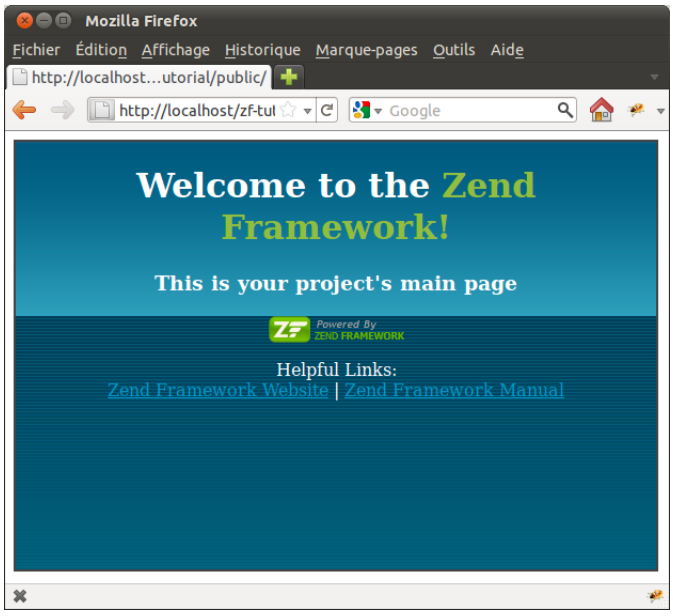
```
<VirtualHost *:80>
       ServerName zf-tutorial.localhost
       DocumentRoot /var/www/html/zf-tutorial/public
       <Directory "/var/www/html/zf-tutorial/public">
              AllowOverride All
       </Directory>
</VirtualHost>
```

Le site serait alors accessible à l'adresse http://zf-tutorial.localhost/ (pensez à mettre à jour votre fichier `c:\windows\system32\drivers\etc\hosts` afin que `zf-tutorial.localhost` corresponde à l'adresse `127.0.0.1`). Nous ne procéderons pas comme cela dans ce tutoriel car il est aussi simple d'utiliser un sous-répertoire pour le test.

Les fichiers images, JavaScript et CSS pris en charge sont stockés dans des répertoires séparés sous le répertoire `public/`. Les fichiers de Zend Framework téléchargés seront placés dans le répertoire `library/`. Si nous avions besoin d'utiliser d'autres librairies, elles pourraient être placées là.

Copiez le répertoire `library/Zend/` du fichier d'archive téléchargé (`ZendFramework-1.10.6-minimal.zip`) vers votre répertoire `zf-tutorial/library/`, de sorte qu'il contienne un sous répertoire appelé `Zend/`.

Vous pouvez tester que tout va bien en naviguant vers http://localhost/zf-tutorial/public/. Vous devriez voir quelque chose comme ceci :



Informations sur l'amorçage

Le contrôleur de Zend Framework utilise le modèle de conception du Contrôleur Frontal (ou Front Controler) et dirige toutes les requêtes via un unique fichier `index.php`. Cela garantit que l'environnement est correctement configuré pour l'exécution de l'application (cette phase s'appelle l'amorçage -- ou bootstraping).

Cela fonctionne grâce à un fichier `.htaccess` qui est généré pour nous par `Zend_Tool` dans le répertoire `zf-tutorial/public/` afin de rediriger toutes les requêtes vers le fichier `public/index.php`, également généré par `Zend_Tool`.

Le fichier `index.php` est le point d'entrée de notre application et sert à créer une instance de `Zend_Application` pour initialiser notre application et ensuite l'exécuter. Ce fichier définit également deux constantes : `APPLICATION_PATH` et `APPLICATION_ENV` qui précisent le chemin d'accès au répertoire `application/` et l'environnement de l'application. La valeur par défaut est fixée à `production` dans `index.php`, mais vous devrez la fixer à `development` dans le fichier `.htaccess` en y ajoutant cette ligne :

```
SetEnv APPLICATION_ENV development
```

Le composant `Zend_Application` est utilisé pour lancer l'application et il est configuré pour utiliser les directives qui se trouvent dans le fichier `application/configs/application.ini`. Ce fichier est également généré automatiquement pour nous.

Une classe `Bootstrap` qui étend `Zend_Application_Boostrap_Bootstrap` est fournie dans `application/Bootstrap.php` et peut être utilisée pour exécuter tout code spécifique au lancement.

Le fichier `application.ini` qui se trouve dans le répertoire `application/configs/` est chargé par le composant `Zend_Config_Ini`. `Zend_Config_Ini` implémente le concept d'héritage entre sections grâce à des deux-points placés dans un 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` indique la section qui est chargée. Évidemment, pendant le développement, la section `development` est préférable et lorsque l'on passe sur le serveur de production, la section `production` devrait être utilisée. Nous placerons toutes les modifications apportées au fichier `application.ini` dans la section `production` de sorte que toutes les configurations en bénéficient.

Modification du fichier application.ini

La première modification que nous devons faire est d'indiquer notre fuseau horaire aux fonctions de date et heure de PHP. Éditez `application/configs/application.ini` et ajoutez :

```
phpSettings.data.timezone = "Europe/London"
```

après toutes les autres valeurs de `phpSettings` dans la section `[production]`. Évidemment, vous devrez probablement utiliser votre propre fuseau horaire. Nous sommes maintenant en situation d'ajouter le code spécifique à notre application.

Code spécifique de l'application

Avant d'installer nos fichiers, il est important de comprendre comment Zend Framework a besoin que les pages soient organisées. Chaque page de l'application est appelée une action et les actions sont regroupées dans des contrôleurs. Pour une URL au format http://localhost/zf-tutorial/public/news/view, le contrôleur est `News` et l'action est `view`. Ceci pour permettre le regroupement d'actions liées. Par exemple, un contrôleur `News` pourrait avoir les actions `list`, `archived` et `view`. Le système MVC de Zend Framework permet aussi d'utiliser des modules pour regrouper des contrôleurs, mais cette application n'est pas assez conséquente pour avoir à s'en préoccuper !

Par défaut, le contrôleur de Zend Framework réserve une action spéciale appelé `index` comme action par défaut. Ainsi, dans des cas comme http://localhost/zf-tutorial/public/news/ c'est l'action `index` du contrôleur

`News` sera exécutée. Il y a également un nom de contrôleur par défaut, qui est aussi appelé `index`, ce qui fait que l'URL http://localhost/zf-tutorial/public/ déclenchera l'exécution de l'action `index` du contrôleur `Index`.

Puisqu'il s'agit d'un tutoriel simple, nous n'allons pas être concerné par des choses *compliquées* comme l'identification d'utilisateurs ! Cela peut attendre un autre tutoriel (ou vous pouvez vous reporter à *Zend Framework in Action* !).

Comme nous avons quatre pages qui concernent toutes des albums, nous les regrouperons sous forme de quatre actions dans un seul contrôleur. Nous utiliserons le contrôleur par défaut et les quatre actions seront :

Page	Contrôleur	Action
Page d'accueil	Index	index
Ajouter un nouvel album	Index	add
Modifier un album	Index	edit
Supprimer un album	Index	delete

Au fur et à mesure que le site se compliquera, d'autres contrôleurs seront nécessaires et vous pourrez aussi regrouper des contrôleurs dans des modules si nécessaire.

Mise en place du contrôleur

Nous sommes maintenant prêt à mettre en place notre contrôleur. Dans Zend Framework, le contrôleur est une classe qui doit être nommée `{Nom du contrôleur}Controller`. Notez que `{Nom du contrôleur}` doit commencer par une lettre majuscule. Cette classe doit se trouver dans un fichier nommé `{Nom du contrôleur}Controller.php` placé dans le répertoire `application/controllers/`. Chaque action est une fonction publique de la classe du contrôleur qui doit être nommée `{Nom de l'action}Action`. Ici `{Nom de l'action}` commence par une lettre minuscule et doit être tout en minuscule. Les noms de contrôleurs et d'actions mélangeant minuscules et majuscules ne sont pas autorisés, ils ont des règles spéciales que vous devez comprendre avant de les utiliser. Reportez-vous d'abord à la documentation !

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

L'ajout d'actions supplémentaires à un contrôleur se fait en utilisant la commande `action create` de l'utilitaire `zf` en ligne de commande. Ouvrez un terminal ou une ligne de commande et changez de répertoire pour vous placer dans votre répertoire `zf-tutorial/`. Ensuite saisissez ces trois commandes :

```
zf create action add Index
zf create action edit Index
zf create action delete Index
```

Ces commandes créent trois nouvelles méthodes : `addAction`, `editAction` et `deleteAction` dans `IndexController` ainsi que les fichiers de script de vue appropriés dont nous aurons besoin plus tard. Nous avons maintenant les quatre actions que nous voulons utiliser :

URL	Méthode d'action
http://localhost/zf-tutorial/public/	`IndexController::indexAction()`
http://localhost/zf-tutorial/public/index/add	`IndexController::addAction()`
http://localhost/zf-tutorial/public/index/edit	`IndexController::editAction()`
http://localhost/zf-tutorial/public/index/delete	`IndexController::deleteAction()`

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

```
View script for controller Index and script/action name add
```

Note : Si vous recevez une erreur 404, c'est que vous n'avez pas configuré Apache avec le module `mod_rewrite` ou que vous n'avez pas paramétré `AllowOverride` correctement dans vos fichiers de configuration d'Apache pour que le fichier `.htaccess` placé dans le répertoire `public/` soit utilisé.

La base de données

Maintenant que nous avons le squelette de notre application avec des fonctions d'action de contrôleur et des fichiers de vue, il est temps de passer à la partie modèle de notre application. Souvenez-vous que le modèle est la partie qui traite de la finalité de l'application (les *règles métier*) et, dans notre cas, cela concerne la base de données. Nous utiliserons la classe `Zend_Db_Table` de `Zend_Framework` qui sert à trouver, insérer, mettre à jour et supprimer des lignes dans une table de base de données.

Configuration de la base de données

Pour utiliser `Zend_Db-Table`, nous devons indiquer quelle base de données utiliser ainsi qu'un nom et mot de passe d'utilisateur. Comme nous préférons ne pas coder en dur ces informations dans notre application nous utiliserons un fichier de configuration pour les enregistrer. Le composant `Zend_Application` expose une ressource pour la configuration de la base de données, ce qui fait qu'il ne nous reste qu'à placer les informations appropriées dans le fichier `configs/application.ini` et il fera le reste.

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

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

Vous devrez bien sur utiliser votre nom d'utilisateur et mot de passe, et votre base de données, pas la mienne ! La connexion à la base de données sera établie automatiquement pour nous et l'adaptateur par défaut de `Zend_Db_Table` sera créé. Vous pouvez découvrir les autres ressources disponibles ici : http://framework.zend.com/manual/fr/zend.application.available-resources.html.

Créer la table de la base de données

Comme prévu dans le planning initial, nous allons utiliser une base de données pour les données de notre album. Je vais utiliser MySQL et donc les instructions SQL pour créer la table sont :

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

Exécutez ces instructions dans un client MySQL comme phpMyAdmin ou le client standard de MySQL en ligne de commande.

Insérer des données de test

Nous insérerons des lignes dans la table afin de pouvoir visualiser la résultat de la fonction de recherche de la page d'accueil. Je vais prendre quelques uns des premiers CD parmi les meilleurs ventes de Amazon au Royaume-Uni. Exécutez l'instruction suivante dans votre client MySQL :

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

Nous avons maintenant des données dans notre base de données et nous pouvons écrire un modèle associé très simple.

Le modèle

Zend Framework ne fourni pas de classe `Zend_Model` car le modèle est votre logique métier et c'est à vous de décider comment vous souhaitez qu'il fonctionne. Il y a beaucoup de composants que vous pouvez utiliser pour cela en fonction de vos besoins. Une approche est d'avoir des classes de modèle qui représentent chaque entité de votre application et ensuite d'utiliser des mappeurs d'objets qui chargent et enregistrent les entités dans la base de données. Cette approche est documentée dans *Démarrez rapidement avec 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`. Zend Framework fournit `Zend_Db_Table` qui implémente le modèle de conception Passerelle vers Table de Données (ou Table Data Gateway) pour permettre de s'interfacer avec les données se trouvant dans une table de base de données. Sachez cependant que le modèle Passerelle vers Table de Données peut avoir ses limites dans des systèmes plus conséquents. Il peut également être tentant de placer le code d'accès à la base de données dans des méthodes d'action de contrôleur comme celles qui sont exposées par `Zend_Db_Table`.

`Zend_Db_Table_Abstract` est une classe abstraite, à partir de laquelle nous dérivons notre classe qui est spécifique à la gestion d'albums. Le nom que nous donnons à notre classe n'a pas d'importance, mais il est logique de le nommer en fonction de la table de la base de données. Notre projet a un chargeur automatique (ou autoloader) par défaut instancié par `Zend_Application` qui fait correspondre les classes ressources d'un module avec le répertoire où il est défini. Nous utilisons le préfixe `Application_` pour les principaux répertoires de `application/`.

Le chargeur automatique fait correspondre les ressources aux répertoires en utilisant cette table de correspondance :

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 nommons en fonction de la table de base de données albums et que nous utiliserons `Zend_Db_Table`, notre classe s'apellera `Application_Model_DbTable_Albums` et sera enregistrée dans `application/models/DbTable/Albums.php`.

Pour indiquer à `Zend_Db_Table` le nom de la table qu'il gérera, nous devons déclarer une propriété protégée `$_nom` au nom de la table. De plus, `Zend_Db_Table` suppose que votre table a une clé primaire nommée `id` qui est incrémentée automatiquement par la base de données. Le nom de ce champ peut aussi être modifié si nécessaire.

Nous pouvons utiliser l'utilitaire `zf` en ligne de commande pour faire une partie du travail, alors exécutez la commande suivante à partir de la ligne de commande :

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

L'utilitaire en ligne de commande a maintenant créé le fichier `Albums.php` dans le répertoire `application/models/DbTable/`. Ce fichier contient une classe nommée `Application_Model_DbTable_Albums` dans laquelle est indiqué le nom de la table de base de données avec laquelle cette classe communiquera.

Nous avons maintenant besoin d'ajouter des fonctionnalités alors éditez `application/models/DbTable/Albums.php` et ajoutez les méthodes `getAlbum()`, `addAlbum()` et `deleteAlbum()` afin qu'il ressemble ensuite à ceci :

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

```
<?php

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

	public function getAlbum($id)
	{
		$id = (int)$id;
		$row = $this->fetchRow('id = ' . $id);
		if (!$row) {
			throw new Exception("Could not find row $id");
		}
		return $row->toArray();
	}

	public function addAlbum($artist, $title)
	{
		$data = array(
			'artist' => $artist,
			'title' => $title,
		);
		$this->insert($data);
	}

	public function updateAlbum($id, $artist, $title)
	{
		$data = array(
			'artist' => $artist,
```

```
                'title' => $title,
            );
            $this->update($data, 'id = '. (int)$id);
        }

        public function deleteAlbum($id)
        {
            $this->delete('id =' . (int)$id);
        }
}
```

Nous avons créé quatre méthodes utilitaires que notre application utilisera pour s'interfacer à la table de la base de données. `getAlbum()` récupère un seule ligne sous forme d'un tableau, `addAlbum()` créé une nouvelle ligne dans la base de données, `updateAlbum()` met à jour la ligne d'un album et `deleteAlbum()` supprime complètement la ligne. Le code de chacune de ces méthodes s'explique de lui-même. Bien que cela ne soit pas nécessaire dans ce tutoriel, vous pouvez également indiquer à `Zend_Db_Table` la présence de tables liées et il récupérera aussi les données liées.

Nous devons alimenter les contrôleurs en données provenant du modèle et faire les scripts de vue pour les afficher, cependant, avant de pouvoir faire cela, nous devons comprendre comment le système de vue de Zend Framework fonctionne.

Gabarits et vues

Le composant vue de Zend Framework s'appelle, sans surprise, `Zend_View`. Le composant vue nous permettra de séparer le code qui affiche la page en fonction du code des fonctions d'action.

L'utilisation basique de `Zend_View` est :

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

On peut voir très facilement que si nous avions mis ce code directement dans chacune de nos quatre fonctions d'action nous aurions répété du code *structuré* très ennuyeux qui n'a aucun intérêt pour l'action. Nous préférons faire l'initialisation de la vue quelque part ailleurs et accéder ensuite à notre objet vue déjà initialisé dans chaque fonction d'action.

Zend Framework fourni un utilitaire d'action appelé `ViewRenderer`. Il prend en charge l'initialisation de la propriété vue (`$this->view`) du contrôleur à utiliser et procédera au rendu d'un script de vue une fois que l'action aura été envoyée.

Pour faire le rendu, `ViewRenderer` prépare l'objet `Zend_View` pour qu'il cherche dans `views/scripts/{Nom du contrôleur}` les scripts de vue à interpréter et produit (au moins, par défaut) le rendu du script nommé en fonction de l'action avec l'extension `phtml`. Autrement dit, le script de vue est `views/scripts/{Nom du contrôleur}/{Nom de l'action}.pthml` et le résultat du rendu est ajouté à la suite du contenu de l'objet `Response`. L'objet `Response` est utilisé pour réunir toutes les en-têtes HTTP, le contenu et les exceptions générées en résultat de l'utilisation du système MVC. À la fin du processus de répartition, le contrôleur frontal envoie automatiquement les en-têtes suivies du contenu.

Ceci est entièrement mis en place pour nous par `Zend_Tool` lorsque nous créons le projet et que nous ajoutons les contrôleurs et les actions en utilisant les commandes `zf create controller` et `zf create action`.

Code HTML commun : Gabarits

Il devient rapidement évident qu'il y aura beaucoup de code HTML commun dans nos vues, au moins pour les sections d'en-tête et de pied de page et peut-être aussi pour une barre latérale ou deux. C'est un problème très courant et le composant `Zend_Layout` est conçu pour résoudre ce problème. `Zend_Layout` nous permet de déplacer tout le code commun à l'en-tête, au pied de page et autre dans un script de gabarit de vue qui inclut le code spécifique à la vue pour que l'action soit exécutée.

L'emplacement par défaut pour mettre nos gabarits est `application/layouts/` et `Zend_Application` expose une ressource qui configurera `Zend_Layout` pour nous. Nous utilisons `Zend_Tool` pour créer le fichier du script de gabarit de la vue et mettre à jour `application.ini` de manière appropriée. À nouveau, à partir du terminal ou de la ligne de commande, saisissez ce qui suit en étant placé dans votre répertoire `zf-tutorial`.

```
zf enable layout
```

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

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

```
<?php
$this->headMeta()->appendHttpEquiv('Content-Type', 'text/html;charset=utf-8');
$this->headTitle()->setSeparator(' - ');
$this->headTitle('Zend Framework Tutorial');
echo $this->doctype(); ?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<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 gabarit contient du code HTML *englobant* assez standard. Puisque c'est un fichier PHP normal, nous pouvons utiliser du code PHP à l'intérieur. Une variable `$this` désigne l'instance de l'objet vue qui a été créé pendant l'amorçage. Nous pouvons utiliser `$this` pour récupérer ce qui a été assigné à la vue et aussi pour appeler des méthodes. Les méthodes (connues sous le nom d'utilitaires de vue – ou view helpers) renvoient des chaînes de caractères que nous pouvons alors afficher.

Tout d'abord nous configurons des utilitaires de vue pour la section d'en-tête de la page Web et nous affichons le `doctype` approprié. Avec la balise `<body>`, nous créons un `div` contenant un `<h1>` où est placé le titre. Pour faire afficher le script de vue de l'action courante, nous faisons un `echo` de la zone de contenu en utilisant l'utilitaire de vue `layout()` : `echo $this->layout()->content;` qui fait le travail pour nous. Cela veut dire que les scripts de vue de l'action sont traités avant le script de gabarit de la vue.

Nous avons besoin de préciser le `doctype` de la page Web avant de faire le rendu des scripts de vue car les scripts de vue d'action sont rendus plus tôt et peuvent avoir besoin de savoir quel `doctype` est en place. Ceci est particulièrement vrai pour `Zend_Form`.

Pour préciser le `doctype` nous ajoutons une autre ligne à notre `application.ini`, dans la section `[production]` :

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

L'utilitaire de vue `doctype()` renverra le `doctype` approprié et des composants comme `Zend_Form` généreront du HTML compatible.

Mise en forme

Bien que ce ne soit *juste* qu'un tutoriel, nous aurons besoin d'un fichier CSS pour donner à notre application un aspect plus présentable ! Cela pose un léger problème dans la mesure où nous ne savons pas comment référencer un fichier CSS puisque l'URL ne pointe pas sur le bon répertoire racine. Heureusement, il existe un utilitaire de vue appelé `baseUrl()`. Cet utilitaire collecte les informations dont nous avons besoin à partir de l'objet requête et nous fournit le morceau d'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 à nouveau un utilitaire de vue, `headLink()`.

zf-tutorial/application/layouts/scripts/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 que des fichiers CSS, plus spécifiques, puissent être ajoutés dans les scripts de vue du contrôleur qui seront rendus dans la section `<head>` après `site.css`.

Enfin, nous avons besoin de styles CSS, ce qui fait que nous créons un répertoire `css` dans `public/` et y plaçons `site.css` avec ce code :

zf-tutorial/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 {
	margin-left: 100px;
}
```

Cela devrait lui donner un meilleur aspect, mais comme vous pourriez le dire, je ne suis pas un designer !

Nous pouvons maintenant nettoyer les quatre scripts d'action qui ont été générés pour nous prêt à être complétés, alors allez-y et videz les fichiers `index.phtml`, `add.phtml`, `edit.phtml` et `delete.phtml` qui, comme vous vous en souvenez sans doute, sont dans le répertoire `application/views/scripts/index/`.

Listage des albums

Maintenant que nous avons paramétré la configuration, les informations en base de données et les squelettes de nos vues, nous pouvons travailler sur l'application et afficher des albums. Cela se passe dans la classe `IndexController` et nous commençons par lister les albums qui se trouvent dans la table avec la fonction `indexAction()` :

zf-tutorial/application/controllers/IndexController.php

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

Nous créons une instance du modèle basé sur Passerelle vers Table de Données. La fonction `fetchAll()` renvoie un `Zend_Db_Table_Rowset` qui nous permettra d'itérer sur les lignes récupérées dans le fichier du script de vue d'action.

Nous pouvons maintenant compléter le script de vue, `index.phtml` :

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

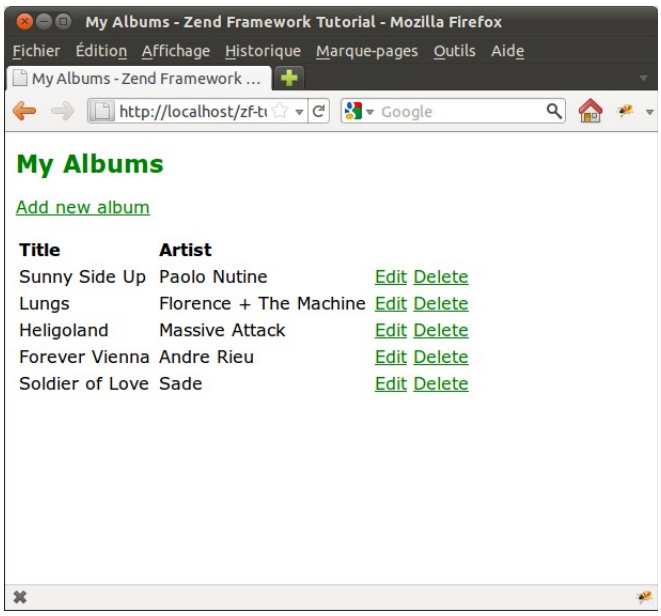
```
<?php
$this->title = "My Albums";
$this->headTitle($this->title);
?>
<p>
	<a href="<?php echo $this->url(array('controller'=>'index', 'action'=>'add'));?
>">Add new album</a>
</p>
<table>
	<tr>
		<th>Title</th>
		<th>Artist</th>
		<th> </th>
	</tr>
	<?php foreach($this->albums as $album) : ?>
```

```
        <tr>
                <td><?php echo $this->escape($album->title);?></td>
                <td><?php echo $this->escape($album->artist);?></td>
                <td>
                        <a href="<?php echo $this->url(array('controller'=>'index',
'action'=>'edit', 'id'=>$album->id));?>">Edit</a>
                        <a href="<?php echo $this->url(array('controller'=>'index',
'action'=>'delete', 'id'=>$album->id));?>">Delete</a>
                </td>
        </tr>
        <?php endforeach; ?>
</table>
```

La première chose que nous faisons est d'indiquer le titre de la page (utilisé dans le gabarit) et aussi le titre de la section `<head>` qui s'affichera dans la barre de titre du navigateur en utilisant l'utilitaire de vue `headTitle()`. Nous créons ensuite un lien pour ajouter un nouvel album. L'utilitaire de vue `url()` est fourni par le framework et créé des liens incluant l'URL de base correcte. Nous lui passons simplement un tableau contenant les paramètres dont nous avons besoin et il fait le reste.

Nous créons ensuite un tableau HTML pour afficher chaque titre et artiste d'album, et nous fournissons des liens pour la modification et la suppression de l'enregistrement. On utilise une boucle standard `foreach` pour itérer sur la liste d'albums, dans sa forme alternative avec deux-points et `endforeach;` car elle est plus facile à comprendre que d'essayer de retrouver des marqueurs. L'utilitaire de vue `url()` est à nouveau utilisé pour créer les liens modifier et supprimer.

Si vous ouvrez http://localhost/zf-tutorial/public/ vous devriez maintenant voir une jolie liste d'albums, quelque chose comme ceci :



Ajout de nouveaux albums

Nous pouvons maintenant coder la fonctionnalité pour ajouter de nouveaux albums. Il y a deux parties à réaliser :

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

Pour faire cela nous utilisons `Zend_Form`. Le composant `Zend_Form` nous permet de créer un formulaire et de valider la saisie. Pour définir notre formulaire nous créons une nouvelle classe `Application_Form_Album` qui étend `Zend_Form`. Comme c'est une ressource d'application, la classe est placée dans le fichier `Album.php` dans le répertoire `application/forms/`. Nous commençons par utiliser le script en `zf` ligne de commande pour créer le fichier correct :

```
zf create form album
```

Cela créé le fichier `Album.php` dans `application/forms/` et inclut une méthode `init()` où nous pouvons configurer le formulaire et ajouter les éléments dont nous avons besoin. Éditez `application/forms/Album.php`, supprimez le commentaire dans la méthode `init()` et ajoutez ce qui suit :

zf-tutorial/application/forms/Album.php

```
<?php

class Application_Form_Album extends Zend_Form
{

       public function init()
       {
              $this->setName('album');

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

              $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 la méthode `init()` de `Application_Form_Album`, nous créons quatre éléments de formulaire pour les champs `id`, `artist`, `title` et le bouton de soumission. Pour chaque élément nous précisons divers attributs, incluant le libellé à afficher. Pour l'`id`, nous voulons garantir que ce sera uniquement un entier pour prévenir des problèmes d'injection SQL. Le filtre `Int` le fera pour nous.

Pour les éléments suivants, nous ajoutons deux filtres, `StripTags` et `StringTrim` pour supprimer le HTML indésirable et les espaces inutiles. Nous les paramétrons aussi comme requis et nous ajoutons un validateur `NotEmpty` pour garantir que l'utilisateur saisira bien l'information requise. (Le validateur `NotEmpty` n'est pas techniquement nécessaire car il sera ajouté automatiquement par le système puisque `setRequired()` a été positionné à `true` ; il est là pour montrer comment ajouter un validateur.)

Nous avons maintenant besoin de faire afficher le formulaire et de traiter sa soumission. Cela est fait dans la

méthode `addAction()` de `IndexController` :

zf-tutorial/application/controllers/IndexController.php

```
...
public function addAction()
    {
       $form = new Application_Form_Album();
       $form->submit->setLabel('Add');
       $this->view->form = $form;

       if ($this->getRequest()->isPost()) {
              $formData = $this->getRequest()->getPost();
              if ($form->isValid($formData)) {
                     $artist = $form->getValue('artist');
                     $title = $form->getValue('title');
                     $albums = new Application_Model_DbTable_Albums();
                     $albums->addAlbum($artist, $title);

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

Examinons cela un peu plus en détails :

```
       $form = new Application_Form_Album();
       $form->submit->setLabel('Add');
       $this->view->form = $form;
```

Nous instancions notre `Application_Form_Album`, fixons le libellé du bouton de soumission à *Add* et nous l'assignons à la vue pour le rendu.

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

Si la méthode `isPost()` de l'objet `Request` est `true`, c'est que le formulaire a été soumis et nous pouvons donc récupérer les données du formulaire dans la requête en utilisant `getPost()` et vérifier si elles sont valides grâce à la fonction membre `isValid()`.

```
                     $artist = $form->getValue('artist');
                     $title = $form->getValue('title');
                     $albums = new Application_Model_DbTable_Albums();
                     $albums->addAlbum($artist, $title);
```

Si le formulaire est valide, nous instancions alors la classe du modèle `Application_Model_DbTable_Albums` et nous utilisons la méthode `addAlbum()` que nous avons créé précédemment pour créer un nouvel enregistrement dans la base de données.

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

Après avoir sauvegardé la ligne correspondant au nouvel album, nous effectuons une redirection en utilisant l'utilitaire d'action `Redirector` pour revenir à l'action `index` (e.g. Nous revenons à la page d'accueil).

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

Si les données du formulaire ne sont pas valides, nous remplissons le formulaire avec les données que

l'utilisateur a saisi et nous le réaffichons.

Nous devons maintenant faire le rendu du formulaire dans le script du vue `add.phtml` :

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

```
<?php
$this->title = "Add new album";
$this->headTitle($this->title);
echo $this->form ;
?>
```

Comme vous pouvez le voir, faire le rendu d'un formulaire est très simple – nous faisons juste un `echo`, car le formulaire sait comment s'afficher lui-même. Vous devriez maintenant pouvoir utiliser le lien *Add new album* sur la page d'accueil pour ajouter un nouvel album.

Modification d'un album

Modifier un album revient quasiment à la même chose que d'en créer un, ce qui fait que le code est très proche :

zf-tutorial/application/controllers/IndexController.php

```
...
public function editAction()
{
        $form = new Application_Form_Album();
        $form->submit->setLabel('Save');
        $this->view->form = $form;

        if ($this->getRequest()->isPost()) {
                $formData = $this->getRequest()->getPost();
                if ($form->isValid($formData)) {
                        $id = (int)$form->getValue('id');
                        $artist = $form->getValue('artist');
                        $title = $form->getValue('title');
                        $albums = new Application_Model_DbTable_Albums();
                        $albums->updateAlbum($id, $artist, $title);

                        $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->getAlbum($id));
                }
        }
}
...
```

Observons les différences par rapport à l'ajout d'un album. Tout d'abord, lors de l'affichage du formulaire à l'utilisateur nous devons récupérer l'artiste et le titre de l'album dans la base de données et remplir 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->getAlbum($id));
                }
```

Notez que ceci est fait si la requête n'est pas un POST, puisqu'un POST implique que nous ayons rempli le

formulaire et souhaitions le traiter. Pour l'affichage initial du formulaire, nous récupérons l'`id` dans la requête en utilisant la méthode `_getParam()`. Nous utilisons ensuite le modèle pour récupérer la ligne dans la base de données et remplir le formulaire directement avec les données de la ligne. (Vous savez maintenant pourquoi la méthode `getAlbum()` du modèle renvoyait un tableau !)

Après la validation du formulaire, nous devons sauvegarder les données dans la bonne ligne de la base de données. Cela est fait par la méthode `updateAlbum()` de notre modèle :

```
$id = (int)$form->getValue('id');
$artist = $form->getValue('artist');
$title = $form->getValue('title');
$albums = new Application_Model_DbTable_Albums();
$albums->updateAlbum($id, $artist, $title);
```

Le modèle de vue est le même que pour `add.phtml` :

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

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

Vous devriez maintenant pouvoir éditer des albums.

Suppression d'un album

Pour compléter notre application, nous devons ajouter la suppression. Nous avons un lien *Delete* à côté de chaque album dans notre liste et l'approche naïve serait de faire une suppression lorsqu'il est cliqué. Ce serait une erreur. En nous remémorant la spécification HTML, nous nous souvenons que l'on ne doit pas réaliser d'action irréversible en utilisant GET et que nous devons plutôt utiliser POST.

Nous devons montrer un formulaire de confirmation lorsque l'utilisateur clique sur *Delete* et s'il clique sur *Yes*, nous ferons une suppression. Comme le formulaire est trivial, nous le coderons directement dans notre vue (Après tout, `Zend_Form` est optionnel !).

Commençons avec le code d'action dans `IndexController::deleteAction()` :

zf-tutorial/application/controllers/IndexController.php

```
...
public function deleteAction()
{
       if ($this->getRequest()->isPost()) {
              $del = $this->getRequest()->getPost('del');
              if ($del == 'Yes') {
                     $id = $this->getRequest()->getPost('id');
                     $albums = new Application_Model_DbTable_Albums();
                     $albums->deleteAlbum($id);
              }
              $this->_helper->redirector('index');
       } else {
              $id = $this->_getParam('id', 0);
              $albums = new Application_Model_DbTable_Albums();
              $this->view->album = $albums->getAlbum($id);
       }
}
...
```

Comme pour l'ajout et la modification, nous utilisons la méthode `isPost()` de `Request` pour déterminer si

nous devons afficher le formulaire de confirmation ou si nous devons faire une suppression. Nous utilisons le modèle `Application_Model_DbTable_Albums` pour supprimer effectivement la ligne en utilisant la méthode `deleteAlbum()`. Si la requête n'est pas un POST, alors nous recherchons un paramètre `id`, nous récupérons le bon enregistrement de la base de données et nous l'assignons à la vue.

Le script de vue est un simple formulaire :

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

```
<?php
$this->title = "Delete album";
$this->headTitle($this->title);
?>
<p>
       Are you sure that you want to delete
       '<?php echo $this->escape($this->album['title']); ?>' by
       '<?php echo $this->escape($this->album['artist']); ?>'      '?
</p>
<form action="<?php echo $this->url(array('action'=>'delete')); ?>"
       method="post">
       <div>
              <input type="hidden" name="id" value="<?php echo $this->album['id']; ?>" />
              <input type="submit" name="del" value="Yes" />
              <input type="submit" name="del" value="No" />
       </div>
</form>
```

Dans ce script, nous affichons un message de confirmation à l'utilisateur puis un formulaire avec des boutons *Yes* et *No*. Dans l'action, nous vérifions que la valeur est *Yes* lorsque nous faisons la suppression.

C'est tout – vous avez maintenant une application totalement fonctionnelle.

Conclusion

Ceci conclut notre brève étude du développement d'une application MVC simple, mais totalement fonctionnelle, utilisant Zend Framework. J'espère que vous l'avez trouvé intéressante et enrichissante. Si vous trouvez quelque chose de faux, merci de m'envoyer un courriel à rob@akrabat.com !

Ce tutoriel a porté sur les bases de l'utilisation du framework ; il y a bien d'autres composants à explorer ! J'ai également laissé de côté beaucoup d'explications. Mon site Web http://akrabat.com contient plusieurs articles sur Zend Framework et vous devriez aussi lire le manuel à l'adresse http://framework.zend.com/manual.

Enfin, si vous préférez un document imprimé, j'ai écrit le livre *Zend Framework in Action* qui est disponible à la vente. Plus de détails sont disponibles à l'adresse http://www.zendframeworkinaction.com. Allez le voir :-)