# Pierwsze kroki z Zend Framework

# (tytuł oryginału: Getting Started With Zend Framework)

Autor: Rob Allen, www.akrabat.com
Tłumaczenie: Radosław Benkel, @singlespl

Wersja dokumentu 1.7.5


Celem kursu jest nauka podstaw tworzenia aplikacji za pomocą Zend Framework'a poprzez utworzenie prostej aplikacji bazodanowej z wykorzystaniem paradygmatu Model-View-Controller.

Uwaga: Informacje w tym kursie przetestowane zostały na wersji **1.10.1** Zend Framework'a. Jest bardzo duża szansa, że opisane tutaj metody zadziałają z nowszą wersją. Nie będą one jednak działać z wersjami poniżej 1.10.1

## Wymagania

Zend Framework posiada następujące wymagania:

PHP w wersji 5.2.4 (bądź wyższej)
Serwer WWW obsługujący mod_rewrite bądź podobną funkcjonalność.

## Założenia kursu

Założyłem, że używasz PHP w wersji przynajmniej 5.2.4 wraz z serwerem Apache. Twoja instalacja Apache **musi posiadać zainstalowane i skonfigurowane rozszerzenie mod_rewrite.**

**Dodatkowo, upewnij się, że Apache jest skonfigurowany tak, aby mógł obsługiwać pliki .htaccess.** Zazwyczaj dokonuje się tego dokonując zmiany dyrektywy konfiguracyjnej:

```
AllowOverride None
```

na

```
AllowOverride All
```

w pliku `httpd.conf`. Więcej szczegółów znajdziesz w dokumentacji. Jeśli nie skonfigurowałeś prawidłowo mod_rewrite oraz .htaccess, nie będziesz w stanie przejść do innej strony aplikacji niż domowa.

## Pobranie frameworka

Zend Framework dostępny jest pod adresem http://framework.zend.com/download/latest w formacie .zip bądź tar.gz. Na końcu strony znajdziesz linki do pobrania – wybierz wersję "Minimal".

## Konfiguracja Zend_Tool

Zend Framework jest dostarczany wraz z narzędziem linii poleceń – `ZendTool`. Zaczniemy od jego konfiguracji.

### Zend_Tool dla Windows

Utwórz nowy folder w `Program Files` nazywając go `ZendFrameworkCli`
Kliknij dwukrotnie na wcześniej pobrany plik - `ZendFramework-[wersja]-minimal.zip`.
Skopiuj foldery `bin` oraz `library` z okna otwartego archiwum `ZendFramework-[wersja]-minimal.zip` do lokalizacji `C:\Program Files\ZendFrameworkCli`. Folder ten powinien mieć teraz dwa podfoldery: `bin` oraz `library`.
Dodaj odpowiednią ścieżkę do folderu `bin` do **Path** w zmiennych środowiskowych systemu:

Otwórz Właściwości system z poziomu Panelu Sterowania oraz wybierz zakładkę "Zaawansoawne". Tam kliknij na przycisk "Zmienne środowiskowe".
W sekcji "Zmienne systemowe" znajdź zmienną `Path` i kliknij na niej dwukrotnie.
Dodaj ścieżkę `;C:\Program Files\ZendFrameworkCli\bin` na końcu pola i naciśnij OK. (średnik na początku jest istotny!)
Zrestartuj computer (w przypadku Windows 7 nie jest to konieczne, wystarczy zamknać okno - przyp. tłum.).

### Zend_Tool dla OS X (oraz Linux)

Rozpakuj archiwum `ZendFramework-[wersja]-minimal.zip`
Skopiuj pliki do folderu `/usr/local/ZendFrameworkCli` korzystając z Terminala/konsoli wydając następujące polecenia:

```
sudo cp -r ~/Downloads/ZendFramework-[wersja]-minimal
/usr/local/ZendFrameworkCli
```

Następnie dokonaj edycji profilu basha:
Z poziomu Terminala/konsoli, wpisz: `open ~/.bash_profile`
Dodaj linijkę `alias zf=/usr/local/ZendFrameworkCli/bin/zf.sh` na końcu pliku
Zapisz plik.
Zamknij terminal

### Testowanie Zend_Tool

Możesz sprawdzić, czy prawidłowo skonfigurowałeś narzędzie Zend_Tool poprzez otworzenie terminala/ /wiersza poleceń i wpisanie następującego polecenia:

```
zf show version
```

Powinieneś zobaczyć podobny tekst:

```
Zend Framework Version: 1.x.x
```

Jeśli nie, sprawdź ustawione wcześniej ścieżki dostępu oraz czy folder bin znajduje się w folderze ZendFrameworkCli. Jeśli narzędzie `zf` działa prawidłowo, wydanie polecenia `zf --help` wyświetli listę wszystkich dostępnych poleceń.

Uwaga: Jeśli Twoja dystrybucja PHP zawiera Zend Framework, upewnij się, że nie jest on w wersji 1.9, ponieważ przykłady podane w kursie nie bedą działać. W czasie pisania tego kursu, xampp postępował w ten sposób.

## Aplikacja

Teraz, kiedy posiadamy wszystko co potrzebne do stworzenia aplikacji, sprecyzujmy, jak wspomniana aplikacja ma działać, Zbudujemy bardzo prosty system do katalogowania naszej kolekcji CD. Główna strona zawierać będzie listę naszych płyt(albumów) i pozwoli nam na dodawanie, edycję oraz usuwanie pozycji. Zaczniemy od szkicu aplikacji. Nasza aplikacja wymagać będzie 4 stron:

| | |
|---|---|
| Strona domowa | Będzie zawierała listę płyt wprowadzonych do systemu wraz z odnośnikami do kasowania oraz edycji. Dodatkowo, znajdzie się tam odnośnik do dodawania nowej pozycji. |
| Dodaj nowy album | Ta strona zawierać będzie formularz dodawania nowego albumu. |
| Edytuj album | Ta strona zawierać będzie formularz do edycji danych albumu. |
| Skasuj album | Ta strona będzie służyć do potwierdzenia chęci usunięcia albumu oraz obsłuży jego usuwanie. |

Informacje o albumach będziemy trzymać w bazie danych. Potrzebować będziemy tylko jednej tabeli z następującymi polami:

| Nazwa pola | Typ | Null? | Komentarz |
|---|---|---|---|
| Id | Integer | No | Primary key, auto increment |
| artist | varchar(100) | No | |
| title | varchar(100) | No | |

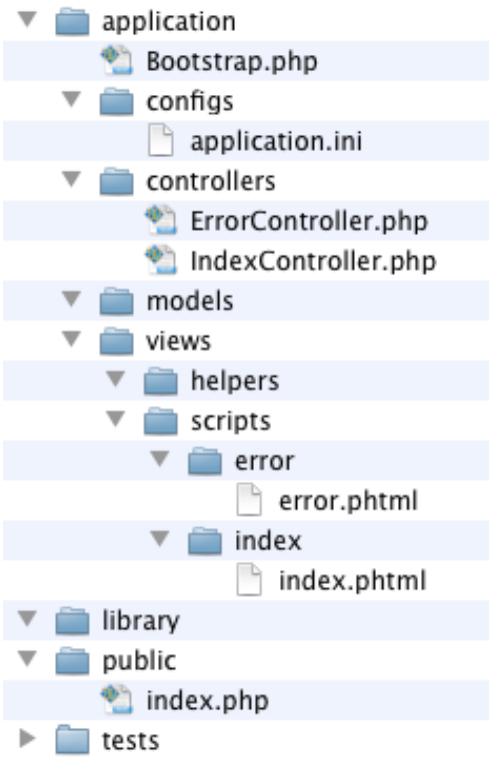
## Tworzenie szkieletu aplikacji

Zacznijmy tworzenie naszej aplikacji. Gdzie tylko będzie to możliwe, będziemy wykorzystywać narzędzie `zf`, ponieważ oszczędzi nam pracy i wysiłku. Pierwszym zadaniem będzie stworzenie szkieletu plików i folderów projektu.

W tym celu otwórz okno terminal/wiersza poleceń i przejdź do folderu głównego swojego serwera WWW. Upewnij się, że masz prawa do zapisu w tym folderze, a proces serwera prawa do odczytu. Następnie wykonaj polecenie:

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

Narzędzie ZF stworzny folder `zf-tutorial` wraz z rekomendowaną strukturą projektu. Taki układ bazuje na przechowywaniu większości plików i folderów poza folderem publicznym serwera. Zakłada on jednak, że posiadasz pełną kontrolę nad instalacją Apache'a. Struktura folderów powinna wyglądać następująco:




(W folderze `public/` powinien znajdować się jeszcze ukryty plik `.htaccess`).

Folder `application/` zawiera kod źródłowy aplikacji. Jak widać, posiadamy osobne foldery dla plików modeli, widoków oraz kontrolerów. Folder `public/` jest folderem publicznym – znaczy to, że link do aplikacji będzie wyglądał następująco: `http://localhost/zf-tutorial/public/`. Dzięki temu większość plików aplikacji nie jest dostępna z poziomu przeglądarki.

Uwaga:

W przypadku wersji produkcyjnej strony, powinieneś utworzyć tzw. “virtual hosta” i ustawić scieżkę dokumentów (`DirectoryRoot`) na folder `public` directory. Dla przykładu, możesz utworzyć virtual hosta `zf-tutorial.localhost` umieszając poniższy kod w pliku httpd.conf:

```
<VirtualHost *:80>
    ServerName zf-tutorial.localhost
    DocumentRoot [ścieżka główna serwera]/zf-tutorial/public
    <Directory "[ścieżka główna serwera]/zf-tutorial/public">
        AllowOverride All
    </Directory>
</VirtualHost>
```

Strona będzie wtedy dostępna pod adresem `http://zf-tutorial.localhost/` (upewnij się, że zmodyfikowałeś plik `/etc/hosts` bądź `c\windows\system32\drivers\etc\hosts` tak aby `zf-tutorial.localhost` wskazywał na IP 127.0.0.1). Nie będziemy tego robić w ramach kursu, ponieważ do testowania obecne podejście jest wystarczające.

Obrazki, pliki z kodem JavaScript oraz arkusze stylów CSS przechowywane są w specjalnych katalogach w ramach folderu `public/`. Pobrane pliki Zend Framework’a przechowywane będą w folderze `library/` - dodatkowe biblioteki także mogą zostać umieszczone w tym miejscu.

Skopiuj zawartość folderu `library/Zend/` z pobranego archiwum do folderu `zf-tutorial/library/`, taka by folder `zf-tutorial/library/` zawierał folder `Zend/`.

Następnie możesz przetestować aplikację wpisując w przeglądarce adres http://localhost/zf-tutorial/public. Powinieneś zobaczyć coś takiego:

### Konfigurowanie aplikacji

Kontrolery Zend Framework'a używają wzorca Front Controller i oraz traktują plik `index.php` jako jedyne "wejścia" aplikacji. Plik `index.php` zapewnia, że środowisko aplikacji będzie prawidłowo skonfigurowane - proces ten nazywany jest "bootstrapping". Przekierowanie wszystkich wywołań do `public/index.php` (utworzonego automatycznie) zapewnia plik `.htaccess` (stworzony rrzez narzędzie ZF) znajdujący się w folderze `zf-tutorial/public`.

Plik `index.php` jest punktem wejściowym naszej aplikacji i używany jest do stworzenia instancji klasy `Zend_Application`, która to inicjalizuje naszą aplikację i uruchamia ją. Plik ten defniuje także dwie stałe: `APPLICATION_PATH` oraz `APPLICATION_ENV` które określają odpowiednio ścieżkę do folderu `application/` oraz środowisko/tryb działania aplikacji. Domyślnie jest ono ustawione na `production`, lecz powinieneś zmienić je na `development` dodając w pliku `.htaccess` następującą linijkę:

```
SetEnv APPLICATION_ENV development
```

Komponent `Zend_Application` używany jest do uruchomienia aplikacji na bazie konfiguracji zawartej w pliku `application/configs/application.ini`. Ten plik także generowany jest automatycznie. Klasa "startowa" (tzw. "Bootstrap"), dziedzicząca po `Zend_Application_Bootstrap_Bootstrap` znajduje się w pliku `application/Bootstrap.php`. Plik ten może być używany do wykonania niestandardowego kodu wymaganego do startu aplikacji.

Plik `application.ini`, znajdujący się w folderze `application/configs` ładowany jest poprzez komponent `Zend_Config_Ini`. `Zend_Config_Ini` pozwala na wykorzystanie dziedziczenia w ramach plików konfiguracyjnych – defniuje się je oddzielając nazwy sekcji pliku `application.ini` dwukropkiem. Dla przykładu:

```
[staging : production]
```

Oznacza to, że sekcja `staging` dziedziczy wszystkie ustawienia z sekcji `production`. Stała `APPLICATION_ENV` określa, która sekcja zostanie załadowana. Oczywiście podczas tworzenia aplikacji najlepszym rozwiązaniem będzie korzystanie z trybu `development`, jednakże w przypadku gotowej aplikacji powinno wykorzystywać się tryb `production`. Wszystkich zmian pliku `application.ini` dokonywać będziemy w ramach sekcji `production`, dzięki czemu niezależnie od wybranego trybu działania ładowane bedą wszystkie dyrektywy konfiguracyjne (dzięki mechanizmowi dziedziczenia).

### Edytowanie pliku application.ini

Pierwszą zmianą, jakiej musimy dokonać jest zmiana strefy czasowej dla prawidłowej obsługi dat i czasu w PHP. Dokonaj edycji pliku `application/configs/application.ini` dodając linijkę:

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

za wszystkimi wartościami typu `phpSettings` w sekcji `[production]`. Oczywiście, powinieneś użyć strefy czasowej odpowiedniej do Twojego miejsca zamieszkania. Teraz możemy dodać własny kod aplikacji.

## Własny kod aplikacji

Zanim zaczniemy pisać aplikacje, ważne jest, aby zrozumieć jakiej organizacji stron oczekuje od nas Zend Framework. Każda strona aplikcji rozumiana jest jako **akcja,** akcje natomiast pogrupowane są w **kontrolery.** Dla URLa wyglądającego w ten sposób: `http://localhost/public/zf-tutorial/news/view`, kontrolerem będzie `News` a akcją `view`. Korzystanie z kontrolerów pozwala na grupowanie powiązanych ze sobą akcji. Dla przykładu, kontroler `News` może posiadać akcje `list`, `archived` and `view`. MVC w Zend Frameworku posiada także wsparcie dla modułów – służą one grupowaniu kontrolerów – jednak w przypadku naszej aplikacji nie ma potrzeby korzystania z tej funkcjonalności.

Domyślną akcją kontrolerów Zend Frameworka jest akcja `index`. Znaczy to, że w przypadku wywołania adresu `http://localhost/zf-tutorial/public/news/` zostanie wywołana akcja `index` zdenifiowana w kontrolerze `News`. Dodatkowo zdefuniowany jest domyślny kontroler, którzy także nazywa

się `index` – znaczy to, że wywołanie adresu `http://localhost/zf-tutorial/public/` spowoduje uruchomienie akcji `index` w ramach kontrolera `Index`.

Biorąc pod uwagę, że kurs ten jest bardzo prosty, nie będziemy zawracać sobie głowy takimi "skomplikowanymi" rzeczami jak logowanie użytkownika do systemu. To może poczekać do następnego kursu (możesz także poczytać na ten temat książce *Zend Framework in Action*!)

Z racji, że posiadamy cztery strony związane z płytami/albumami, umieścimy je w ramach jednego kontrolera. Użyjemy kontrolera domyślnego i będą to następujące akcje:

| Strona | Kontoler | Akcja? |
|---|---|---|
| Strona główna | Index | index |
| Dodaj nowy album | Index | add |
| Edytuj album | Index | edit |
| Skasuj album | Index | delete |

W miarę, jak strona będzie stawała się bardziej skomplikowana, potrzebne będą dodatkowe kontrolery bądź moduły.

## Tworzenie kontrolera

Jesteśmy gotowi do tworzenia kontrolera. W Zend Frameworku, kontroler reprezentowany jest przez klasę, która musi posiadać nazwę pasującą do wzorca: `{Nazwa kontrolera}Controller`. Zwróć uwagę, aby `{Nazwa kontrolera}` zaczynała się od wielkiej litery. Klasa ta musi być zdefniowana w pliku `{Nazwa kontrolera}Controller.php` zapisanym w folderze `application/controllers`. Każda akcja reprezentowana jest przez metodę publiczną zdefiniowaną w ramach kontrolera i nazywającą się `{nazwa akcji}Action`. W tym przypadku `{nazwa akcji}` zaczyna się od małej litery - tak samo cała nazwa powinna być pisana małymi literami. Używanie równocześnie wielkich i małych liter w nazwach kontrolerów oraz akcji jest dopuszczalne, lecz obowiązują specjalne reguły z tym związane – tak więc najpierw przeczytaj dokumentację!

Nazwa klasy naszego kontrolera to `IndexController` – zdefiniowana ona jest w pliku `application/controllers/IndexController.php` – został on automatycznie utworzony przez narzędzie Zend_Tool. Klasa ta zawiera także definicję jednej akcji - `indexAction()`. Musimy tylko dodać nasze akcje.

Akcje można dodawać korzystając z narzędzia `zf` z opcją `create action`. Otwórz terminal/wiersz poleceń i przejdź do katalogu `zf-tutorial/`. Wpisz następujące komendy:

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

Utworzą one następujące metody: `addAction`, `editAction` oraz `deleteAction` w ramach kontrolera `IndexController` oraz pliku widoków, które będą nam potrzebne później. Posiadamy teraz wszystkie akcje, których chcemy użyć.

Adresy dla każdej z akcji przedstawiają się następująco:

| Strona | Kontoler |
|---|---|
| http://localhost/zf-tutorial/public/ | `IndexController::indexAction()` |
| http://localhost/zf-tutorial/public/index/add | `Controller::addAction()` |
| http://localhost/zf-tutorial/public/index/edit | `IndexController::editAction()` |
| http://localhost/zf-tutorial/public/index/delete | `IndexController::deleteAction()` |

Możesz przetestować, czy wspomniane adresy działają – powinieneś komunikaty w tym stylu:

View script for controller **index** and script/action name **add**

**Note:** Jeśli otrzymujesz błąd 404, nie skonfigurowałeś prawidłowo Apache'a wraz z modułem mod_rewrite bądź nie ustawiłeś prawidłowo dyrektywy AllowOverride w konfiguracji Apache'a, przez co plik `.htaccess` w folderze `public/` nie jest interpretowany.

# Baza danych

Teraz, kiedy posiadamy szkielet aplikacji ze zdefniowanym kontrolerem, akcjami oraz widokami, nadszedł czas aby przyjrzeć się modelom naszej aplkacji. Pamiętaj, że model jest tą częścią aplikacji, odpowiada za reguły biznesowe aplikacji i w naszym wypadku reprezentuje bazę danych. Będziemy wykorzystywać klasę `Zend_Db_Table`, która używana jest do wyszukiwania, dodawania, edycji oraz usuwania rekordów tabeli.

## Konfiguracja bazy danych

Aby móc korzystać z klas `Zend_Db_Table`, musimy przekazać jej informację z której bazy danych mają korzystać (razem z nazwą użytkownika oraz hasłem). Z racji, że nie chcemy zakodować tej informacji "na sztywno" w kodzie aplikacji (tzw. "hard-coding" – przyp.tłum.), skorzystamy z pliku konfiguracyjnego. Komponent `Zend_Application` posiada plugin służący do automatycznej konfiguracji bazy danych, tak więc wystarczy dodać odpowiednie dyrektywy konfiguracyjne do pliku `configs/application.ini`, a plugin zajmie się resztą.

Otwórz plik `application/configs/application.ini` i dodaj poniższe linie w sekcji `[production]`:

```
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
```

Oczywiste jest, że powinieneś użyć swojej nazwy użytkownika, hasła oraz nazwy bazy danych – nie moich! (baza musi być wcześniej utworzona - przyp. tłum.). Połączenie z bazą danych zostanie utworzone automatycznie i przypisane do klasy `Zend_Db_Table`. Na temat innych pluginów `Zend_Applicaion` możesz poczytać tutaj: http://framework.zend.com/manual/en/zend.application.available-resources.html.

## Stwórz tabelę bazy danych

Jak wspomnieliśmy na początku, do przechowywania naszych danych wykorzystamy bazę danych. Używać będę MySQL, dlatego kod SQL tworzący tabelę wygląda następująco:

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

Uruchom ten kod korzystając z klienta MySQL, np. phpMyAdmin bądź klienta mysql obsługiwanego z linii poleceń (komenda `mysql`).

## Dodaj testowe dane

Dodamy także kilka rekordów do tabeli, abyśmy mogli zaobserwować działanie strony głównej naszej aplikacji. Ja wybiorę listę pierszych pięciu bestsellerów ze sklepu Amazon UK. Wykonaj poniższe zapytanie za pomocą swojego klienta 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');
```

Teraz, kiedy posiadamy prosty zestaw danych w naszej bazie, możemy zabrać się za pisanie prostego modelu.

## Model

Zend Framework nie udostępnia klasy `Zend_Model`, ponieważ model odpowiada za Twoją logikę biznesową i to od Ciebie zależy jak ją zaimplementujesz. Isnieje wiele różnych komponentów, które możesz wykorzystać do tego celu. Jednym z podejść jest posiadanie klasy modelu dla każdej encji Twojej aplikacji, a następnie używać tzw. 'obiektu mapującego' który pobiera i zapisuje dane do bazy. Podejście to opisane jest w tym dokumencie: http://framework.zend.com/manual/en/learning.quickstart.create-model.html.

Na potrzebny tego kursu, utworzymy model dziedziczący po klazie `Zend_Db_Table` oraz korzystający z klas `Zend_Db_Table_Row`. Zend Framework udostępnia component `Zend_Db_Table`, który implementuje wzorzec projektowy Table Data Gateway. Wzorzec ten udostępnia interfejs do komunikacji z tabelą w bazie danych. Miej jednak na uwadze, że Table Data Gateway może być ograniczeniem w przypadku większych aplikacji. Istnieje także pokusa, aby umieszczać kod związany z obsługą bazy danych w ramach akcji kontrolerów.

Klasa `Zend_Db_Table_Abstract` jest klasa abstrakcyjną, po której dziedziczyć będzie nasza klasa służąca do zarządzania naszymi albumami. Nazwa klasy nie ma znaczenia, aczkolwiek wskazane może być nazwanie jej podobnie jak tabeli w naszej bazie. Nasz projekt posiada domyślny "autoloader" zainicjowany przez `Zend_Application`, który mapuje nasze klasy na odpowiadające im pliki w obrębie naszego modułu. Znaczy to, że dla klas umieszczonych w ramach głównego folderu `application/` będziemy używać prefiksu `Application_` w nazwach klas.

Autolader mapuje klasy na w następujący sposób:

| Prefiks | Folder |
|---|---|
| `Form` | `forms` |
| `Model` | `models` |
| `Model_DbTable` | `models/DbTable` |
| `Model_Mapper` | `models/mappers` |
| `Plugin` | `plugins` |
| `Service` | `services` |
| `View_Filter` | `view/filters` |
| `View_Helper` | `view/helpers` |

Z racji, że nazywamy modele zgodnie z nazwami tabel w bazie danych oraz korzystamy z `Zend_Db_Table`, nasza klasa nazywać się będzie `Application_Model_DbTable_Albums` i zapisana będzie w pliku `applications/models/DbTable/Albums.php`.

Aby "dać znać" klasom `Zend_Db_Table` do której tabeli mają się odnosić, musimy przypisać do chronionego atrybutu obiektu (`$_name`) nazwę tableli. `Zend_Db_Table` zakłada także, że tablela posiada kolumnę (z kluczem podstawowym) o nazwie `id`, której wartość zwiększanana jest automatycznie przez bazę danych przy dodawaniu kolejnych rekordów ("autoincrement" w MySQL, sekwencje w przypadku PostgreSQL – przyp. tłum.). Nazwa kolumny z kluczem podstawowym może być zmieniona, jeśli zajdzie taka potrzeba.

Możemy skorzystać z narzędzia ZF w celu oszczędzenia nam pracy. Uruchom poniższą komendę:

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

Narzędzie ZF utworzyło za nas plik `Albums.php` w folderze `application/models/DbTable`. Plik ten zawiera definicję klasy `Application_Model_DbTable_Albums` łącznie z ustawionym atrybutem `$_name` wskazującym na odpowiadającą tabelę w bazie danych.

Teraz musimy dodać trochę własnego kodu. W tym celu, edytuj plik `application/models/DbTable/Albums.php` i dodaj metody `getAlbum()`, `addAlbum()`, `updateAlbum()` and `deleteAlbum()` tak, aby plik wyglądał następująco:

**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);
    }
}
```

Stworzylliśmy cztery metody pomocznicze, z których będzie korzystać nasza aplikacja. Metoda `getAlbum()` zwróci jeden rekord w postacji tablicy, `addAlbum()` tworzy nowy rekord w bazie danych, `updateAlbum()` uaktualnia informacje o albumie a `deleteAlbum()` usuwa album. Kodu dla każdej z tych akcji nie trzeba raczej wyjaśniać. Możesz także skonfigurować `Zend_Db_Table` tak, aby móc pobierać dane z powiąznych tabel, aczkolwiek nie jest to częścią tego kursu.

Naszym zadaniem będzie pobranie danych z bazy danych w kontrolerze i przypisanie ich do widoku, jednakże, zanim będziemy mogli tego dokonać, musimy zrozumieć jak działa system widoków w Zend Frameworku.

## Szablony widoku (layouts) i widoki (views)

Komponent widoku Zend Frameworka nazywa się `Zend_View` – co nie jest zaskakujące. Komponent widoku pozwoli nam na rozdzielenie kodu odpowiedzialnego za wyświetlanie strony od kodu w akcjach kontrolerów.

Podstawowy sposób użycia `Zend_View` wygląda tak:

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

Łatwo można zauważyć, że korzystając z powyższego kodu w każdej z naszych akcji, skończymy na pisaniu w kółko tego samego, strukturalnego kodu, który nie ma nic wspólnego z wykonywaną akcją. Lepszym rozwiązaniem byłoby zainicjowanie gdzieś obiektu widoku i korzystanie z zainicjowanego już obiektu. Zend Framework udostępnia ActionHelper (po polsku "pomocnik akcji") o nazwie `ViewRenderer`. Jego zadaniem jest zainicjowanie atrybutu widoku w ramach kontrolera (`$this->view`) oraz automatyczne wyrenderowanie widoku pod koniec wykonywania akcji.

W celach renederingu, `ViewRenderer` konfiguruje obiekt `Zend_View` tak, aby szukał on plików widoku w folderze `views/scripts/{nazwa kontrolera}` i domyślnie wybierał plik o takiej samej nazwie jak nazwa wykonywanej akcji, z dodanym rozszerzeniem `phtml`. Tak więc renderowanym plikiem widoku będzie `views/scripts/{nazwa kontrolera}/{nazwa akcji}.phtml` a wygenerowana treść zostanie przypisana do obiektu Response. Obiekt Response używany jest w celu łączenia wszystkich nagłówków HTTP, zawartości odpowiedzi oraz wyjątków będących wynikiem używania MVC. Front Controller pod koniec przetwarzania żądania automatycznie wysyła nagłówki, a wraz z nimi treść odpowiedzi.

Wymienione wyżej czynności są wykonywane za nas automatycznie podczas tworzenia projektu czy też dodawania kontrolerów oraz akcji za pomocą `Zend_Tool` .

## Wspólny kod HTML: Szablony widoków (layouts)

Szybko staje się oczywiste, że w naszych widokach będzie sporo identycznego kodu HTML – m.in. dla nagłówka i stopki, możliwe także, że dodatkowo dla paneli bocznych, Jest to często spotykany problem, dlatego ZendFramework zawiera komponent `Zend_Layout`. `Zend_Layout` pozwala nam na przeniesienie powtarzającego się kodu (nagłówek, stopka itd) do pliku szablonu widoku, w ramach którego wstawiana jest zawartość wyrenderowanego widoku przypisanego do wykonywananej akcji.

Domyślnym miejscem do zapisywania szablonów widoku jest folder `application/layouts/`. Istnieje także plugin do `Zend_Application` który skonfiguruje `Zend_Layout` za nas. Skorzystamy z `Zend_Tool` do stworzenia szablonu widoku i modyfikacji pliku konfiguracji. Z poziomu terminala/wiersza poleceń wykonaj następująca komendę w folderze `zf-tutorial`:

```
zf enable layout
```

`Zend_Tool` stworzył folder `application/layouts/scripts` oraz szablon widoku `layout.phtml` w tym folderze. Dokonał także modyfikacji pliku `application.ini` dodając linijkę `resources.layout.layoutPath = APPLICATION_PATH "/layouts/scripts/"` na końcu sekcji `[production]`.

Na samym końcu procesu przetwarzania żądania, kiedy akcje kontrolerów zostały już wykonane, `Zend_Layout` dokona renderingu naszego szablonu widoku. `Zend_Tool` udostępnia bardzo prosty szablon który wyświetla tylko zawartość pliku widoku wykonywanej akcji. Dodamy do niego kod wymagany w naszej aplikacji. Otwórz plik `layouts.phtml` i zamień znajdujący się tam kod na poniższy:

**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>
```

Plik szablonu zawiera “zewnętrzny” kod HTML, który wygląda dość standardowo. Tak samo jak w ramach normalnych plików PHP, w ramach widoków i szablonów możemy korzystać ze składni PHP. Dostępna zmienna `$this` odpowiada obiektowi widoku, do którego mamy dostęp z poziomu kontrolera: `$this->view`. Możemy używać tego obiektu do pobierania danych oraz w celu wywoływania metod. Metody te (znane jako helpery widoku – view helpers) zwracają najczęściej ciąg tekstowy, tak więc możliwe jest użycie instrukcji `echo`.

Na początku konfigurujemy kilka helperów widoku dla sekcji `head` strony, a następnie wyświetlamy prawidłowy doctype. W ramach taga `<body>`, stworzymy diva zawierającego tag `<h1>` z tytułem strony. Aby zagnieździć widok wykonywanej właśnie akcji w ramach szablonu widoku, skorzystamy z helpera `layout()`: `echo $this->layout()->content;`. Znaczy to, że najpierw zostanie wykonany skrypt widoku, a następnie szablon.

Zanim zaczniemy, musimy ustawić doctype. Z racji że widoki renederowane są wcześniej, warto ustawić odpowiedni doctype na samym początku żądania, tak aby niektóre klasy mogły z tego korzystać - przykładem takiej klasy jest `Zend_Form`.

Aby ustawić doctype dodamy jeszcze jedną linię w pliku `application.ini`, w sekcji `[production]`:

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

Dzięki temu helper `doctype()` wygeneruje za nas odpowiedni ciąg znaków, a komponenty takie jak `Zend_Form` będą w stanie wygeneorwać kod odpowiedni dla wybranego standardu.

## Stylowanie

Mimo tego, że dokument ten jest tylko wprowadzeniem, będziemy potrzebować pliku CSS, aby nasza aplikacja wyglądała choć trochę reprezentatywnie. Niestety, wykorzystanie plików CSS powoduje mały problem polegający na tym, że nie wiemy jaką ścieżkę podać do pliku CSS, ponieważ aplikacja nie znajduje się w głównym folderze serwera (tzn, że uruchamiamy ją z poziomu `http://localhost/zf-tutorial/public` – przyp.tłum.).Jednakże, istnieje helper o nazwie `baseUrl()`. Jego zadaniem jest pobranie informacji z obiektu żadania i utworzenie odpowiedniego przedrostka adresu (w naszym wypadku zwróci on ciąg `zf-tutorial/public`).

Możemy teraz dodać odnośnik do pliku CSS w ramach sekcji `<head>` pliku `application/layouts/scripts/layout.phtml` ponownie wykorzystując do tego helper widoku - w tym wupadku będzie to `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>
...
```

Poprzez wykorzystanie metody `prependStylesheet()` helpera `headLink()`, zezwalamy widokom na dołączanie specyficznych dla siebie plików CSS, które zostaną podlinkowane w sekcji `<head>` po pliku głównym - `site.css`.

W pliku CSS potrzebujemy kilku podstawowych defnicji, tak więc utwórzmy folder `css` w lokalizacji `public/` oraz plik `site.css` z poniższym kodem:

**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;
}
```

Teraz strona powinna wyglądać troszkę ładniej, aczkolwiek jak widzisz – nie jestem grafikiem!

Możemy teraz śmiało wykasować zawartość plików `index.phtml`, `add.phtml`, `edit.phtml` oraz `delete.phtml`, które jak pewnie pamiętasz, znajdziesz w folderze `pplication/views/scripts/index`.

## Lista albumów

Teraz, kiedy mamy przygotowaną konfigurację, schemat bazy danych oraz podstawowe widoki, możemy zabrać się za pisanie kodu aplikacji. Zaczniemy od klasy `IndexController` i akcji `indexAction()` – przygotujemy kod, który pobierze listę albumów z bazy danych i wyświetli ją w postaci tabeli.

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

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

Najpierw tworzymy nowy obiekt modelu. Metoda. `fetchAll()` zwraca obiekt klasy `Zend_Db_Table_Rowset`, który pozwoli nam iterować po znalezionych rekordach w ramach pliku widoku.

Możemy teraz zapisać poniższy kod w pliku `index.phtml`:

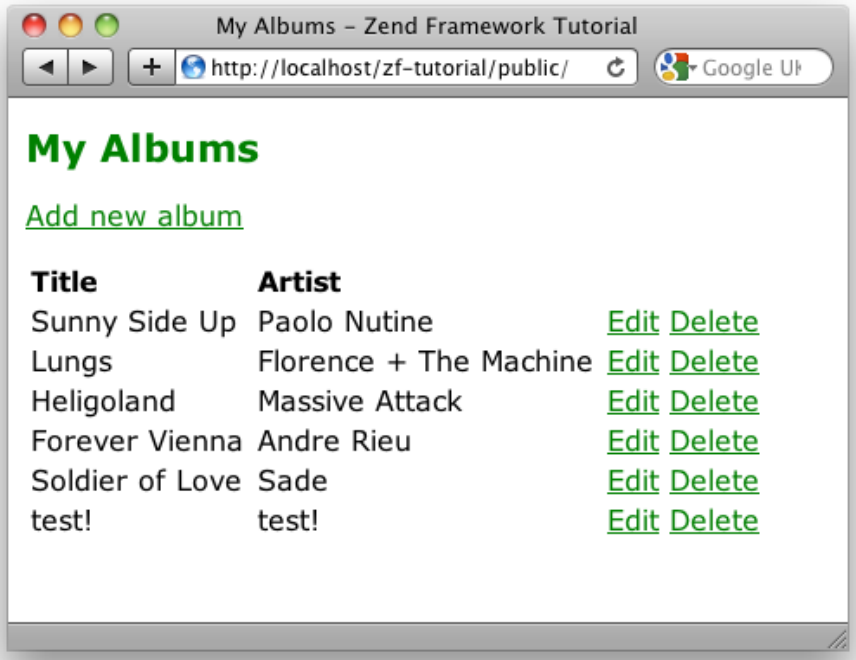
**zf-tutorial/application/views/scripts/index/index.phtml**

```
<?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>
```

Pierwszymi rzeczami, jakie zrobimy będzie ustawienie tytułu strony (wykorzystana będzie w layoucie) oraz ustawienie tytułu dla sekcji `<head>`, używając helpera `headTitle()` – wyświetlony zostanie on na pasku przeglądarki. Następnie tworzymy odnośnik do dodania nowego albumu. Helper widoku o nazwie `url()` jest częścią frameworka i odpowiada za stworzenia prawidłowego odnośnika wraz z adresem bazowym. Wystarczy, że przekażemy tablicę parametrów z nazwą kontrolera i akcji, a resztą zajmie się helper.

Następnie tworzymy tabelę zawierającą tytuł i wykonawcę każdego albumu oraz odnośniki prowadzących do edycji oraz usuwania rekordów. Wykorzystujemy standardową pętlę `foreach:` do iteracji po liście albumów. Warty odnotowania jest fakt, że używamy alternatywnej formy zapisu instrukcji sterujących – zakończonych dwukropkiem oraz średnikiem w przypadku `endforeach;` - rozwiązanie takie pozwala łatwiej zorientować się w kodzie. Ponownie wykorzystujemy helper `url()` do utworzenia odnośników.

Jeśli przejdziesz pod adres http://localhost/zf-tutorial/public/ (bądź ten ustawiony przez Ciebie!) powinieneś zobaczyć listę albumów, wyglądającą mniej więcej tak:




## Dodawanie nowych albumów

Możemy teraz dopisać kod odpowiedzialny za funkcjonalność dodawania nowego albumu. Proces ten będzie składał się z dwóch części:

Wyświetlenie użytkownikowi formularza dodawania albumu
Przetworzenie danych z formularza i zapisanie ich w bazie danych

Do tego celu wykorzystamy klasę `Zend_Form`. Komponent `Zend_Form` umożliwia utworzenie formularza oraz walidację danych wejściowych. Tworzymy nową klasę `Form_Album` (dziedziczącą po `Zend_Form`) w celu zdefiniowania naszego formularza. Z racji, że jest to komponent naszej aplikacji, definicja klasy przechowywana jest w pliku `Album.php` w folderze `forms`. Zaczniemy od wykorzystania komendy `zf` do stworzenia odpowiedniego pliku:

```
zf create form Album
```

Polecenie to utworzy plik `Album.php` w folderze `application/forms`. Klasa zawiera także metodę `init()`, w której to możemy dokonać konfiguracji formularza oraz dodać potrzebene elementy. Dokonaj edycji pliku `application/forms/Album.php` i wklej tam następujący kod:

**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));
    }
}
```

W ramach metody `init()` klasy `Application_Form_Album` utworzyliśmy cztery elementy formularza, odpowiednio dla identyfikatora albumu, nazwy wykonawcy, tytułu płyty oraz przycisku wysyłania formularza. Każdemu z nich przypisaliśmy różne atrybuty, łącznie z etykietą, która zostanie wyświetlona obok niego. W przypadku identyfikatora chcemy, aby był on tylko liczbą całkowitą (przeciwdziałanie potencjalnym atakom SQL Injection). Dlatego dodajemy filtr `Int`, który wykona to zadanie za nas.

Dla elementów tekstowych, dodajemy dwa filtry: `StripTags` oraz. `StringTrim`. Pierwszy usuwa niechciane znaczniki HTML, drugi natomiast niepotrzebne białe znaki. Definiujemy je także jako pola wymagane oraz dodajemy walidatory `NotEmpty` aby upewnić się, że użytkownik wpisze dane których od niego wymagamy (walidator `NotEmpty` praktycznie nie jest wymagany, ponieważ zostaje on automatycznie dodany w momencie zdefiniowania pola jako wymagane – linijka ta została jednak dodana, aby pokazać sposób w jaki dodaje się walidatory).

Musimy teraz wyświetlić formularz oraz obsłużyć przesłane dane. Dokonamy tego w ramach metody `addAction()` kontrolera `IndexController`:

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

```
    ...
    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);
            }
        }
    }
    ...
```

Przeanalizujmy kod krok po krku:

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

Inicjujemy obiekt klasy `Form_Album`, ustawiamy nazwę przycisku wysyłającego formularz na “Add” oraz przypisujemy do zmiennej widoku.

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

Jeśli metoda obiektu żądania, `isPost()` zwraca wartość `true`, znaczy to że formularz został wysłany, dlatego też możemy pobrać przesłane dane za pomocą metody `getPost()` i sprawdzić ich poprawność za pomocą metody `isValid()` formularza.

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

Jeśli przesłane dane są poprawne, inicjuemy obiekt klasy modelu - `Application_Model_DbTable_Albums` i wywołujemy zdefniowaną wcześniej metodę `addAlbum()` w celu zapisania danych w bazie.

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

Kiedy już zapisaliśmy nowy rekord w bazie, przekierowujemy żądanie do akcji `index`, wykorzystując do tego celu helper `Redirector` (chcemy wrócić do strony głównej).

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

Jeśli dane formularza nie przeszły procesu walidacji, wypełniamy pola formularza przesłanymi danymi i ponownie wyświetlamy go użytkownikowi.

Musimy teraz wyświetlić formularz w ramach pliku widoku `add.phtml`:

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

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

Jak widać, wyświetlenie formularza jest bardzo proste – wystarczy skorzystać z instrukcji echo. Teraz powiniśmy mieć możliwość skorzystania z odnośnika “Add new album” na stronie głównej i pomyślnego dodania nowego albumu do listy.

## Edycja albumu

Edycja albumu jest bardzo podobna do jego dodawania, dlatego kod wygląda podobnie:

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

```
    ...
    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));
            }
        }
    }
    ...
```

Przyjrzyjmy się różnicom między tym kodem, a kodem dodawania albumu. Pierwsze co robimy przed wyświetleniem formularza użytkownikowi to pobranie danych albumu oraz przypisanie ich do pól formularza.. Odpowiada za to poniższy fragment kodu:

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

Zwróć uwagę, że kod ten wywoływany jest w momencie, kiedy żądanie nie jest przesyłane metodą POST, ponieważ w przypadku przesłania go metodą POST, uruchamiany jest kod opowiedzialny za przetworzenie formularza. W celu wyświetlenia danych albumu, pobieramy najpierw z parameter id z żądania za pomocą metody `_getParam()`. Następnie wykorzystujemy model aby pobrać rekord z bazy danych oraz wypełnić formularz tymi danymi. (to jest właśnie powód, dla którego metoda `getAlbum()` zwraca tablicę).

Po walidacji danych formularza, uaktualniamy dane w bazie, wykorzystując do tego celu metodę `updateAlbum()`:

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

Plik widoku jest identyczny jak w przypadku add.phtml:

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

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

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

Powinieneś teraz mieć możliwość edycji albumów.

## Usuwanie albumu

W celu ukończenia naszej aplikacji, musimy dodać także funkcjonalność usuwania albumu. Posiadamy przecież odnośnik “Delete” przy każdym z albumów, jednak kasowanie albumu bezpośrednio to zbyt proste podejście. Byłoby to po prostu złe. Mając na uwadze specyfikację HTTP, pamiętamy, że nie powinniśmy używać metody GET w przypadku wywoływania nieodwracalnych akcji – wskazane jest użycie zamiast tego metody POST.

Powinniśmy wyświetlić formularz potwierdzający chęć usunięcia albumu i usunać go dopiero wtedy, gdy użytkownik kliknie przycisk “Yes”. Z racji tego, że formularz będzie bardzo prosty, zakodujemy go bezpośrednio w naszym widoku (w końcu korzystanie z `Zend_Form` jest opcjonalne).

Zacznijmy od kodu akcji `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);
        }
    }
    ...
```

Tak samo jak w przypadku dodawnia i edycji, wykorzystamy metodę obiektu żądania `isPost()` w celu określenia, czy powinnśmy wyświetlić formularz potwierdzający usunięcie czy też rzeczywiście usunąć rekord. Skorzystamy z modelu `Application_Model_DbTable_Albums` oraz metody `deleteAlbum()` w celu usunęcia albumu. Jeśli żądanie nie zostało przesłane metodą POST, pobierzemy odpowiedni rekord i przypiszemy go do obiektu widoku korzystając z parametru id.

Plik widoku zawiera prosty formularz:

**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>
```

Wyświetlamy wiadomość potwierdzającą chęć usunięcia albumu, a następnie formularz z przyciskami “Yes” oraz “No”. W ramach akcji sprawdzamy, czy uzytkownik wcisnał przycisk “Yes”.

To wszystko – posiadasz teraz w pełni działającą aplikację.

## Podsumowanie

Dotarliśmy do końca tego krótkiego kursu mającego na celu stworzenie prostej, lecz w pełni funkcjonalnej aplikacji bazującej na wzorcu MVC za pomocą Zend Frameworka.Mam nadzieję, że materiał ten był dla Ciebie interesujący. Jeśli znalazłeś jakieś  błędy, napisz proszę do mnie: rob@akrabat.com!

Ten kurs objął swoim zakresem tylko podstawowe zagadnia – jest jeszcze wiele do odkrycia! Dodatkowo, pominąłem wiele wyjaśnień. Na mojej stronie, http://akrabat.com znajdziesz wiele artykułów na temat Zend Framewora. No i zdecydowanie powinieneś przeczytać manual dostępny na stronie: http://framework.zend.com/manual

Jeśli jednak wolisz słowo drukowane, napisałem książkę nazwaną Zend Framework in Action, którą jest dostępna w sprzedaży. Więcej informacji znajdziesz na stronie http://www.zendframeworkinaction.com. Sprawdź ☺

*Od tłumacza: W miarę możliwości starałem się przetłumaczyć dokument słowo w słowo, zachowując styl oryginału. Nie wnikałem w to, czy MVC w ZF w rzeczywistości jest prawdziwym MVC, czy też Zendowy widok naprawdę jest widokiem wg definicji tego wzorca. To samo tyczy się innych wzorców wspomnianych w dokumencie. Takie rzeczy jak nazwy przycisków, czy odnośników nie były tłumaczone celowo.*