

I gang med Zend Framework

Af Rob Allen, www.akrobat.com

Dokumentrevision 1.3.4

Copyright © 2006, 2007

Denne tutorial er ment som en yderst grundlæggende introduktion til brugen af Zend Framework i forbindelse med en enkel databasedrevet applikation.

NOTE: Denne tutorial er testet på Zend Framework, version 0.9, 0.9.1 og 0.9.2.* Der er høj sandsynlighed for at den også virker med senere versioner, men den virker bestemt ikke med versioner før 0.9.

* oa: Oversætteren har testet tutorialen på version 0.9.3 uden problemer.

ADVARSEL vedrørende version 0.9: Hvis du bruger Zend Framework version 0.9, er du nødt til at redigere i filen `library/Zend/Db/Table/Row/Abstract.php` og tilføje "<" i begyndelsen af filens første linje, så den starter med "<?php" i stedet for "?php".

Model-View-Controller arkitektur

Den traditionelle måde, hvorpå en PHP applikation normalt bygges ligner eksemplet herunder:

```
<?php
include "common-libs.php";
include "config.php";
mysql_connect($hostname, $username, $password);
mysql_select_db($database);
?>

<?php include "header.php"; ?>
<h1>Home Page</h1>

<?php
$sql = "SELECT * FROM news";
$result = mysql_query($sql);
?>
<table>
<?php
while ($row = mysql_fetch_assoc($result)) {
?>
<tr>
    <td><?php echo $row['date_created']; ?></td>
    <td><?php echo $row['title']; ?></td>
</tr>
<?php
}
?>
</table>
<?php include "footer.php"; ?>
```

Vedligeholdelse af denne type applikation umuliggøres i takt med klientens voksende antal ønsker, og efterhånden som ændringer hackes ind flere og flere steder i den eksisterende kode.

Vedligeholdelsen kan forbedres ved at splitte koden op i tre bestemte og sædvanligvis adskilte dele:

Model	Applikationens modeldel tager vare på detaljer i forbindelse med data, der skal vises. I eksempelkoden ovenfor er det konceptet "news". Modeldelen er
--------------	---

	hovedsaglig beskæftiget med applikationens forretningslogik, ofte i forbindelse med databaserelateret funktionalitet.
View	Viewdelen består af applikationsbidder, der er gearet mod visning til brugeren. Dette sker sædvanligvis i html.
Controller	Controlleren binder data og views sammen for at sikre at de rigtige data vises på de rigtige sider.

Zend Framework bruger Model-View-Controller (MVC) arkitekturen. Derved adskilles applikationens forskellige dele og gør udvikling og vedligeholdelse nemmere.

Krav

Før Zend Framework kan bruges skal php 5.1.4 eller højere være installeret, og serveren skal understøtte mod_rewrite funktionalitet. Denne tutorial er rettet mod Apache serveren.

Hvordan får jeg fat i Zend Framework?

Zend Framework kan hentes på: <http://framework.zend.com/download/stable> i både zip og tar format. I skrivende stund er version 0.9.2 den nyeste (nu version 0.9.3, o.a.). Du bør i det mindste benytte version 0.9.1, hvis denne tutorial skal virke.

Mappestruktur

Brug af Zend Framework betinger ikke en bestemt mappestruktur, men manualen anbefaler en almindelig brugt struktur. Denne struktur antager, at vi har komplet kontrol over vores Apachekonfiguration. Ikke desto mindre kan livet gøres lettere, så vi benytter en modifikation.

Begynd med at lave en mappe i web-serverens rodmappe og kald den zf-tutorial. Det betyder, at den url, der giver adgang til applikationen er `http://localhost/zf-tutorial`.

Lav mappestrukturen til applikationens filer som vist nedenfor:

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

Som det ses har vi adskilte mapper til model-, view- og controllerfilerne i vores applikation. Understøttende fotos, scripts og CSS-filer opbevares i separate mapper i publicmappen. Det hentede Zend Framework placeres i librarymappen. Har vi brug for andre biblioteker til vores applikation, kan de også placeres her.

Pak den hentede fil, f.eks. `ZendFramework-0.9.3-Beta.zip`, ud i en midlertidig mappe. Alle filerne befinder sig i undermappen, `ZendFramework-0.9.3-Beta`. Kopier indholdet fra `ZendFramework-0.9.3-Beta` over i `zf-tutorial/library`.

`zf-tutorial/library` bør nu indeholder undermappen `Zend`.

Bootstrapping

Zend Frameworks controller, `Zend_Controller`, er designet til at understøtte websider med rene urler. For at opnå dette, går alle henvendelser til sitet via en enkelt fil, `index.php`, som i daglig tale kaldes bootstrapper. Dette udstyrer os med et centralt adgangspunkt til alle applikationens sider, hvilket sikrer at miljøet til afvikling af applikationen kan sættes rigtigt op. Dertil benytter vi os af en `.htaccess`-fil i `zf-tutorials` rodmappe:

zf-tutorial/.htaccess

```
RewriteEngine on
RewriteRule .* index.php

php_flag magic_quotes_gpc off
php_flag register_globals off
```

`RewriteRule` er meget enkel og kan fortolkes: "benyt `index.php` til enhver url-request".

Vi sætter også et par PHP ini indstillinger for sikkerhed og helbred. Disse bør allerede være indstillet korrekt, men vi ønsker at være helt sikre. Bemærk venligst at `php_flag` indstillinger i `.htaccess` kun virker, hvis du benytter `mod_php`. Hvis du benytter CGI/FastCGI, er du nødt til at sikre dig, at din `php.ini` er korrekt.

Henvendelser om fotos, JavaScript filer eller CSS filer bør ikke rutes til vor bootstrapper. Ved at opbevare alle disse filer i `public` undermappen kan vi nemt med en `.htaccess` fil i `zf-tutorial/public` konfigurere Apache til at servere disse direkte:

zf-tutorial/public/.htaccess

```
RewriteEngine off
```

Selvom det ikke er strengt nødvendigt med vores nuværende rewrite regler, kan vi føje endnu et par `.htaccess` filer til for at sikre, at applikationen og bibliotekernes mapper er beskyttede:

zf-tutorial/application/.htaccess

```
deny from all
```

zf-tutorial/library/.htaccess

```
deny from all
```

Bemærk: For at muliggøre Apaches brug af `.htaccess` filer skal konfigurationsdirektivet `AllowOverride` være sat til `All` i `httpd.conf` filen. Idéen om brug af flere `.htaccess` filer er fra Jayson Minards artikel: "[Blueprint for PHP Applications: Bootstrapping \(Part2\)](#)". Jeg anbefaler læsning af begge artikler.

Bootstrapfilen: index.php

`zf-tutorial/index.php` er vor bootstrapfil, og vi starter med følgende kode:

zf-tutorial/index.php

```
<?php
error_reporting(E_ALL|E_STRICT);
date_default_timezone_set('Europe/London');

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

Zend_Loader::loadClass('Zend_Controller_Front');

// setup controller
$frontController = Zend_Controller_Front::getInstance();
```

```

$frontController->throwExceptions(true);
$frontController->setControllerDirectory('./application/controllers');

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

```

Bemærk at vi ikke afslutter filen med `?>`, da det ikke er nødvendigt, og hvis vi undlader det, forebygges nogle fejl, der kan være meget svære at finde, når vi videresender via funktionen `header()`, og der er ekstra blanke tegn efter endelsen `?>`.

Lad os gennemgå filen.

```

error_reporting(E_ALL|E_STRICT);
date_default_timezone_set('Europe/London');

```

Disse linjer sikrer at de fejl, vi laver, vises (forudsat `php.ini` har `display_errors` sat til on). Vi sætter også den nuværende tidszone, som PHP 5.1+ ønsker det. Vælg den tidszone, der passer til din situation.

```

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

```

Zend Framework er designet sådan, at dets filer skal være i includestien. Vi placerer også vores modelmappe i includestien så vi nemt kan lade vores modelklasser senere. Til at begynde med inkluderer vi `Zend/Loader.php` for at give adgang til klassen `Zend_Loader`, som har de statiske funktioner, der skal til, for at vi kan lade andre klasser fra Zend Framework

```

Zend_Loader::loadClass('Zend_Controller_Front');

```

`Zend_Loader::loadClass` loader den nævnte klasse. Dette opnås ved at konvertere underscores i klassenavne til stiseparatører og derefter tilføje endelsen `.php`. Sådan vil klassen `Zend_Controller_Front` blive loadet fra filen `Zend/Controller/Front.php`. Hvis vi følger samme navnekonvention for vore egne biblioteksklasser, kan vi bruge `Zend_Loader::loadClass()` til at lade dem også. Den første klasse, vi har brug for, er front controlleren.

Frontcontrolleren bruger en routerklasse til mapning af en ønsket url til den PHP funktion, der viser siden. For at routeren skal kunne virke, skal den kunne finde ud af, hvilken del af urlen, der er stien til vores `index.php` så den kan se på URI elementerne derefter. Dette gøres med objektet `Request`. Dette objekt gør et godt stykke arbejde med autotetektering af den korrekte grundurl, men hvis det ikke virker for din opsætning, kan du forbigå (override) det ved at bruge funktionen `$frontController->setBaseUrl()`.

Vi er nødt til at konfigurere frontcontrolleren så den ved, i hvilken mappe den finder vore controllere.

```

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

```

Da dette er en tutorial, og vi benytter et testsystem, har jeg besluttet at instruere frontcontrolleren om at kaste alle undtagelser. Som standard fanger front controlleren undtagelser og opbevarer dem i "Response"-objektets egenskab `_exceptions`. Responseobjektet indeholder al information om responsen til en url-request. Det indbefatter HTTP header, sideindhold og undtagelser. Frontcontrolleren vil automatisk sende headere og vise sideindhold lige før den gør sit arbejde færdigt.

Dette kan være temmelig forvirrende når man stifter bekendtskab med Zend Framework for første gang, så det er lettere bare at genkaste undtagelserne, så de er let synlige. Selvfølgelig viser man ikke fejl til brugerne på en produktionsserver!

Endelig kommer vi til sagens kerne og vi kan køre applikationen:

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

Åbner du din browser på adressen <http://localhost/zf-tutorial> for at teste, bør du se en fatal fejl i stil med:

Fatal error: Uncaught exception 'Zend_Controller_Dispatcher_Exception' med meddelelsen 'invalid controller specified (index)' in ...

Dette fortæller os at vi endnu ikke har lavet vores applikation, og før vi gør det, må vi hellere tale om, hvad vi skal til at bygge, så lad os gøre det, som det næste.

Websitet

Vi skal bygge et meget enkelt lagersystem til visning af vores cd samling. Hovedsiden vil vise vores samling og tillade os at tilføje, ændre og slette cd'er. Vi opbevarer vores samling i en tabel som denne:

Feltnavn	Type	Null?	Noter
id	Integer	No	Primary key, Autoincrement
artist	Varchar(100)	No	
title	Varchar(100)	No	

Nødvendige sider

Følgende sider er nødvendige:

Hjemmeside	Denne vil vise en liste af albummer og links til brug for slet og rediger. Der vil også være et link, så nye album kan tilføjes.
Tilføj nyt album	Denne side har en form til tilføjelse af nye albummer
Rediger album	Denne side har en form til redigering af et album
Slet album	Denne side vil konfirmere sletning og derefter slette.

Sidernes organisering

Før vi laver vore filer, er det vigtig at forstå, hvordan frameworket forventer, at siderne er organiseret. Hver side i en applikation er kendt som en "action" og actions er grupperet i controllere. Det vil sige, at for et format som `http://localhost/zf-tutorial/news/view` er controlleren `news` og action er `view`. Dette tillader gruppering af relaterede actions. F.eks. kan en `news` controller have actions som `current`, `archived` og `view`. Zend Frameworks MVC system supporterer også moduler, der grupperer controllere sammen, men denne applikation er ikke stor nok til, at vi benytter os af dem!

Zend Frameworks controller reserverer en bestemt action ved navn `index` som standard action. Det betyder, at for en url som `http://localhost/zf-tutorial/news/view`, vil `index` action i `news` controlleren blive eksekveret. Zend Frameworks controller reserverer også et standard controllernavn, hvis et sådant ikke modtages. Ikke alt for overraskende er det også `index`. Sådant vil urlen `http://localhost/zf-tutorial` bevirke at `index` action i `index` controlleren eksekveres.

Da dette er en grundlæggende tutorial vil vi ikke bekymre os om "kompliserede" ting som at logge ind. Det kan vente til en senere tutorial ...

Da vi har fire sider, der alle handler om albummer, vil vi gruppere dem i en enkelt controller som fire actions. Vi bruger standard controlleren og de fire actions er:

<i>Side</i>	<i>Controller</i>	<i>Action</i>
Hjemmeside	Index	Index
Tilføj nyt album	Index	Add
Rediger album	Index	Edit
Slet album	Index	Delete

Dejlig enkelt!

Controllerens opsætning

Vi er nu klar til at lave vores controller. I Zend Framework er controlleren en klasse, der skal kaldes på denne måde: `{Controllernavn}Controller`. Bemærk at `{Controllernavn}` skal begynde med stort. Klassen skal være til stede i en fil med navnet `{Controllernavn}Controller.php` i den specificerede controllers mappe. Enhver action er en public funktion i controllerklassen, som skal navngives `{actionnavn}Action`. I dette tilfælde skal `{actionnavn}` begynde med småt.

Vores controllerklasse hedder `IndexController` og er defineret i `zf-tutorial/application/controllers/IndexController.php`:

zf-tutorial/application/controllers/IndexController.php

```
<?php

class IndexController extends Zend_Controller_Action
{
    function indexAction()
    {
        echo "<p>in IndexController::indexAction()</p>";
    }

    function addAction()
    {
        echo "<p>in IndexController::addAction()</p>";
    }

    function editAction()
    {
        echo "<p>in IndexController::editAction()</p>";
    }

    function deleteAction()
    {
        echo "<p>in IndexController::deleteAction()</p>";
    }
}
```

Begyndelsesvis har vi lavet det sådan, at de forskellige actions udskriver deres eget navn. Test dette ved at navigere til de følgende URLer:

<i>URL</i>	<i>Vist tekst</i>
http://localhost/zf_tutorial/	in IndexController::indexAction()
http://localhost/zf_tutorial/index/add	in IndexController::addAction()
http://localhost/zf_tutorial/index/edit	in IndexController::editAction()
http://localhost/zf_tutorial/index/delete	in IndexController::deleteAction()

Nu har vi en fungerende router og de korrekte actions eksekveres for hver side i vores applikation. Hvis dette ikke virker, kan du kigge i afsnittet *Fejlfinding* i slutningen af denne tutorial og måske der finde brugbare informationer.

Det er på tide at bygge viewdelen.

Opsætning af view

Zend Frameworks viewkomponent hedder ikke alt for overraskende, `Zend_View`. Viewkomponenten tillader os at adskille koden, der viser en side fra koden i actionfunktionerne.

Standardbrug af `Zend_View` er almindeligvis:

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

Det ses tydeligt, at hvis vi skulle lægge dette skelet direkte ind i hver af vores actionfunktioner, ville vi gentage kode, der er uinteressant for actionfunktionen. Vi vil hellere initialisere viewet et andet sted og derefter tilgå det allerede initialiserede viewobjekt fra hver actionfunktion.

Zend Framework designerne forudså denne type problemer og byggede løsningen ind i `Zend_Controller_Action` for os. Der er to hjælpefunktioner: `initView()` og `render()`. `initView()` laves for os a objektet `Zend_View`, som giver det til egenskaben `$view`, klar til at vi kan fodre det med data. Det sætter også objektet `Zend_View` til at kigge i `views/scripts/{controllernavn}` for det view-script, der skal renderes. Rendering udføres af `render()`, som (i det mindste i standardindstillingen) vil renderer scriptet `{action_navn}.phtml` og hæfte det til responseobjektets krop. Responseobjektet bruges til at samle alle headere, bodyindhold og undtagelser genereret under brug af MVC systemet. Front controlleren sender da automatisk headere fulgt af bodyindhold i slutningen af dispatch.

For at kunne integrere viewet i vores applikation, er vi nødt til at initialisere det i funktionen `init()` og sikre os, at vi kalder `render()` i hver action. Vi er også nødt til lave nogle viewfiler med testkode til visning.

Forandringerne i `IndexController` følger (forandringer er fremhævet med fed). Som det ses, tilføjer vi en ny funktion, `init()`, som kaldes automatisk af constructoren i `Zend_Controller_Action`. Dette sikrer, at viewet er initialiseret korrekt fra begyndelsen, og at vi kan have tillid til, at det er klar til brug i actionfunktionerne.

zf-tutorial/application/controllers/IndexController.php

```
<?php

class IndexController extends Zend_Controller_Action
{
    function init()
    {
        $this->initView();
    }

    function indexAction()
    {
        $this->view->title = "My Albums";
        $this->render();
    }

    function addAction()
    {
        $this->view->title = "Add New Album";
        $this->render();
    }

    function editAction()
    {
        $this->view->title = "Edit Album";
        $this->render();
    }

    function deleteAction()
    {
        $this->view->title = "Delete Album";
        $this->render();
    }
}
```

I hver funktion tildeler vi en titelvariabel til viewegenskaben og kalder derefter render() for at vise viewskabelonen. Men bemærk at selve visningen ikke sker endnu – det gøres af front controlleren til sidst i dispatchprocessen.

Vi er nu nødt til at føje fire viewfiler til vores applikation. Disse filer kendes som skabeloner, og metoden render() forventer, at hver skabelonfil er navngivet efter sin action og har endelsen .phtml (for at vise, at det er en skabelonfil). Filen skal være i en undermappe, der er navngivet efter controlleren, så de fire filer er:

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

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

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

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

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

```
<html>
```

```

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

```

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

```

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

```

Afprøvning af hver controller/action bør udskrive den tilhørende titel med fed skrift.

Fælles HTML kode

Det bliver meget hurtigt tydeligt, at der er megen fælles HTML kode i de fire views. Vi vil udfaktore den HTMLkode, der er fælles, til to filer: header.phtml og footer.phtml, i scriptmappen. Vi kan da bruge dem til at holde "fælles" HTML og nøjes med at referere til dem fra hver viewskabelon.

De nye filer er:

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

```

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

```

(Bemærk at vi har ændret koden til gyldig HTML!)

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

```

</div>
</body>
</html>

```

Nu er vi nødt til igen at ændre de fire views:

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

```

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

```

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

```

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

```

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

```

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

```

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

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

Formatering – styles

Selv om dette er en tutorial, har vi brug for en CSS-fil, så vores applikation bliver præsentabel. Dette skaber et mindre problem, fordi vi faktisk ikke ved, hvordan vi skal referere til denne fil, når urlen ikke peger på den rigtige mappe. Det kan løses ved at benytte `getBaseUrl()` funktionen, som er en del af “requesten”, og give den videre til det rette view. Dette giver os den del af urlen, vi ikke kender.

Lav om på `IndexController::init()` så den ser ud som følger:

zf-tutorial/application/controllers/IndexController.php

```
...
function init()
{
    $this->initView();
    $this->view->baseUrl = $this->_request->getBaseUrl();
}
...
```

Derefter ændrer vi `<head>` delen i `header.phtml`, så den indeholder et link til CSS-filen:

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

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

Til slut laver vi nogle CSS-styles:

zf-tutorial/public/styles/site.css

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

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

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

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

#formbutton {
```

```
        margin-left: 100px;
    }

    a {
        color: #800000;
    }
}
```

Dette burde forskønne vores sider en smule!

Databasen

Nu da vi har adskilt kontrol af applikationen fra visning, er det på tide at se på applikationens modeldel. Husk på, at modellen er den del, der har at gøre med applikationens kerneformål (de såkaldte forretningsregler) og som derfor, i vores tilfælde, har at gøre med databasen. Vi vil bruge Zend Frameworks klasse, `Zend_Db_Table`, som bruges til at finde, indsætte, opdatere og slette rækker i en databasetabel.

Konfiguration

Før vi kan bruge `Zend_Db_Table` er vi nødt til at definere, hvilken database, vi bruger, samt brugernavn og password. Da vi foretrækker ikke at kode dette direkte ind i vores applikation, laver vi en konfigurationsfil, der indeholder denne information.

Zend Framework stiller `Zend_Config` til rådighed, som sætter os i stand til at benytte en smidig objektorienteret adgang til konfigurationsfiler. Konfigurationsfilen kan enten være en INI eller en XML-fil. Vi vil benytte en INI-fil:

zf-tutorial/application/config.ini

```
[general]
db.adapter = PDO_MYSQL
db.config.host = localhost
db.config.username = rob
db.config.password = 123456
db.config.dbname = zftest
```

Du er selvfølgelig nødt til at bruge dit eget brugernavn, password og databasenavn, ikke mine!

At benytte `Zend_Config` er meget nemt:

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

Bemærk at `Zend_Config_Ini` i dette tilfælde bruger en enkelt sektion fra INI-filen og ikke alle (selvom det er muligt at benytte alle sektioner, hvis du ønsker det). Det understøtter en notation i sektionsnavnet som tillader os at loade yderligere sektioner. `Zend_Config_Ini` behandler også "punktum" i parameteret som hierarkisk separator og tillader derved gruppering af relaterede konfigurationsparametere. I vores `config.ini` vil `host`, `brugernavn`, `password` og `databasenavn` blive grupperet under `$config->db_config`.

Vi vil nu loade vores konfigurationsfil i bootstrapperen (`index.php`):

Relevant del i zf-tutorial/index.php

```
...
Zend_Loader::loadClass('Zend_Controller_Front');
Zend_Loader::loadClass('Zend_Config_Ini');
Zend_Loader::loadClass('Zend_Registry');

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

// setup controller
...

```

Forandringerne er fremhævet med fed. Vi loader de klasser, vi skal bruge (Zend_Config_Ini og Zend_Registry) og loader derefter sektionen 'general' fra application/config.ini ind i vort objekt, \$config.

Note: I denne tutorial, behøver vi faktisk ikke at opbevare \$config i registeret, men det er en god vane, da du i 'virkelige' applikationer ofte har mere end databasekonfigurationsinformation i INI-filen. Vær også opmærksom på, at registeret opfører sig globalt og skaber afhængigheder imellem objekter, hvor der ingen afhængighed er, hvis du ikke er forsigtig.

Opsætning af Zend_Db_Table

For at kunne benytte Zend_Db_Table, er vi nødt til at give den databasekonfigurationsinformationerne, vi lige har loadet. Til den brug skaber vi en instans af Zend_Db og registrerer den med den statiske funktion Zend_Db_Table::setDefaultAdapter(). Det gør vi også i bootstrapperen (tilføjelser er fremhævet med fed):

Relevant del i zf-tutorial/index.php

```
...
Zend_Loader::loadClass('Zend_Controller_Front');
Zend_Loader::loadClass('Zend_Config_Ini');
Zend_Loader::loadClass('Zend_Registry');
Zend_Loader::loadClass('Zend_Db');
Zend_Loader::loadClass('Zend_Db_Table');

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

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

// setup controller
...
```

Lav tabellen

Jeg vil benytte MySQL og erklæringen af tabellen er som følger:

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

Kør denne tabelerklæring i en MySQL klient som f.eks. phpMyAdmin eller fra MySQLs standard prompt.

Indsæt testalbummer

Vi vil også indsætte et par rækker i tabellen, så vi kan teste hjemmesidens afhentningsfunktionalitet. Jeg vil bruge de første to cd'er fra "Hot 100" på Amazon.co.uk:

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

Modeldelen

`Zend_Db_Table` er en abstrakt klasse, så vi er nødt til at bruge nedarvning fra den til den klasse, der er specifik for applikationens håndtering af albummer. Det er ligegyldigt, hvad vi kalder klassen, men det er fornuftigt at give den samme navn, som databasetabellen. Derfor kalder vi vores klasse for `Album`, og vores tabel for `album`. For at fortælle `Zend_Db_Table` navnet på den tabel, vi vil bruge, har vi sat egenskaben `$_name` til navnet på vores tabel. `Zend_Db_Table` går også ud fra at du har en primærnøgle ved navn `id`, som autooptælles af databasen. Primærnøglen navn kan også ændres, hvis det er et krav.

Vi vil opbevare klassen `Album` i modelmappen:

zf-tutorial/application/models/Album.php

```
<?php

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

Ikke særlig kompliceret, vel? Heldigvis for os, er vore behov meget enkle og `Zend_Db_Table` giver os al den funktionalitet, vi har brug for. Men skulle du få brug for specifik funktionalitet for at kunne varetage din model, så er dette klassen hvori du definerer funktionaliteten. Almindeligvis ville de ekstra funktioner, du gerne vil tilbyde, have forbindelse med søgning, så du kan sætte klassen i stand til at vise præcis de data, du ønsker. Du kan også fortælle `Zend_Db_Table` om relaterede tabeller, og den kan også hente relateret data.

Albumoversigt

Nu da vi har sat databasens konfiguration op, kan vi beskæftige os med applikationens kerne og vise nogle albummer. Dette gøres i klassen `IndexController`.

Enhver action i `IndexController` vil tydeligvis manipulere tabellen `album` i databasen, og der er derfor fornuft i at lade klassen `album`, når controlleren instantieres. Dette gøres fra `init()` funktionen:

zf-tutorial/application/controllers/IndexController.php

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

Note: Dette er et eksempel på, hvordan vi bruger `Zend_Loader::loadClass()` til at lade vore egne klasser, og det virker, fordi vi har defineret modelmappen i `php-include-stien` i `index.php`.

Vi vil vise en oversigt over vore albummer via en tabel i `indexAction()`:

zf-tutorial/application/controllers/IndexController.php

```
...
function indexAction()
{
    $this->view->title = "My Albums";
    $album = new Album();
    $this->view->albums = $album->fetchAll();
    $this->render();
}
...
```

Funktionen `Zend_Db_Table::fetchAll()` returnerer et `Zend_Db_Table_Rowset`, som tillader os at iterere over de returnerede rækker i viewskabelonen:

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

```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<p><a href="<?php echo $this->baseUri; ?>/index/add">Add new album</a></p>
<table>
<tr>
    <th>Title</th>
    <th>Artist</th>
    <th>&nbsp;</th>
</tr>

<?php foreach($this->albums as $album) : ?>
<tr>
    <td><?php echo $this->escape($album->title); ?></td>
    <td><?php echo $this->escape($album->artist); ?></td>
    <td>
        <a href="<?php echo $this->baseUri; ?>/index/edit/id/<?php
            echo $album->id; ?>">Edit</a>
        <a href="<?php echo $this->baseUri; ?>/index/delete/id/<?php
            echo $album->id; ?>">Delete</a>
    </td>
</tr>
<?php endforeach; ?>
</table>
<?php echo $this->render('footer.phtml'); ?>
```

<http://localhost/zf-tutorial/> (eller hvor du nu følger med fra!) bør nu vise en cool lille liste med (to) albummer.

Tilføj nye albummer

Vi kan nu kode funktionaliteten for tilføjelse af nye albummer. Der er to bidder i denne del:

- Vis en form hvori en bruger kan udfylde detaljer
- Behandl den sendte form og gem i databasen

Det gøres fra `addAction()`:

zf-tutorial/application/controllers/IndexController.php

```
...
function addAction()
{
    $this->view->title = "Add New Album";

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

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

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

```

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

// set up an "empty" album
$this->view->album = new stdClass();
$this->view->album->id = null;
$this->view->album->artist = '';
$this->view->album->title = '';

// additional view fields required by form
$this->view->action = 'add';
$this->view->buttonText = 'Add';

$this->render();
}
...

```

Bemærk hvordan vi checker `$_SERVER['REQUEST_METHOD']` variabelen for at se, om formen er blevet sendt. Hvis den er, henter vi kunstner og titel fra det sendte array ved at benytte klassen `Zend_Filter_StripTags` for at sikre, at ingen html er tilladt. I det vi antager, at felterne er udfyldt, bruger vi vores modelklasse, `Album()`, til at indsætte den nye række i databasetabellen album.

Efter vi har tilføjet det nye album sender vi brugeren tilbage til applikationens start med metoden `_redirect()`.

Endelig gør vi viewet klar til den form, vi vil bruge i skabelonen. Med tanke fremad er det klart, at den form edit action skal bruge, vil være mæge til denne. Vi vil derfor benytte en fælles skabelonfil (`_form.phtml`) som kaldes fra både `add.phtml` og `edit.phtml`.

Skabelonerne til tilføjelse af et album følger her:

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

```

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

```

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

```

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

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

```

Dette er temmelig enkel kode. Da vi ønsker at benytte `_form.phtml` til action edit, har vi brugt variabelen `$this->action` i stedet for at kode actionvariablen direkte ind i scriptet.

Redigere et album

Redigering af et album er næste mæge til tilføjelse af et, så koden ligner meget:

zf-tutorial/application/controllers/IndexController.php

```
...
function editAction()
{
    $this->view->title = "Edit Album";
    $album = new Album();

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

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

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

                $this->_redirect('/');
                return;
            } else {
                $this->view->album = $album->fetchRow('id='.$id);
            }
        }
    } else {
        // album id should be $params['id']
        $id = (int)$this->_request->getParam('id', 0);
        if ($id > 0) {
            $this->view->album = $album->fetchRow('id='.$id);
        }
    }

    // additional view fields required by form
    $this->view->action = 'edit';
    $this->view->buttonText = 'Update';

    $this->render();
}
...
```

Bemærk at vi ikke er i "post"-mode, vi henter id parameteret fra requestens params egenskab ved hjælp af `getParam()`.

Skabelonen er som følger:

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

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

Refaktorering!

Det har sikkert ikke undgået opmærksomheden, at addAction() og editAction() ligner hinanden meget, og at add- og editskabelonerne er ens. Refaktorering er på sin plads!

Det overlader jeg som en øvelse til dig, kære læser...

Slet et album

Til afrunding af applikationen har vi brug for at kunne slette. Vi har et slettelink i forbindelse med hvert album på vores oversigtsside og den naive tilgang ville være at slette det pågældende album, når der klikkes på linket. Det ville være ukorrekt. Ihukommende HTTP-specifikationerne, bør vi erindre, at vi ikke bør foretage en handling, der ikke kan fortrydes, ved hjælp af GET, men i stedet bruge POST. Googles seneste betaaccelerator bragte denne forståelse hjem hos mange mennesker.

Vi skal vise en form til bekræftelse, når brugeren klikker på slettelinket og kun hvis brugeren derefter klikker "ja", sletter vi.

Koden ligner stort set actionerne add og edit.

zf-tutorial/application/controllers/IndexController.php

```
...
function deleteAction()
{
    $this->view->title = "Delete Album";

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

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

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

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

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

Vi benytter igen det samme trick, hvor vi checker requestmetoden for at finde ud af, om vi skal vise formen til bekræftelse, eller om vi skal foretage en sletning via Album() klassen. Ganske som i insert og update foretages den faktiske sletning via et kald til Zend_Db_Table::delete().

Bemærk at vi returnerer straks efter opsætning af responsekroppen. Det gør vi, fordi vi så kan returnere til oversigten over albummer ved funktionens afslutning. Hvis nogen af vores helbredschecks fejler, kan vi derfor returnere til oversigten uden at kalde `_redirect` adskillige gange inden funktionen.

Skabelonen er en enkel form:

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

```
<?php echo $this->render('header.phtml'); ?>
<h1><?php echo $this->escape($this->title); ?></h1>
<?php if ($this->album) :?>
<form action="<?php echo $this->baseUrl ?>/index/delete" method="post">
<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>
<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>
<?php else: ?>
<p>Cannot find album.</p>
<?php endif;?>
<?php echo $this->render('footer.phtml'); ?>
```

Fejlfinding

Hvis du har problemer med at få andre actions end `index/index` til at virke er det mest sandsynligt, at routeren er ude af stand til at bestemme, hvilken undermappe din hjemmeside befinder sig i. Mine undersøgelser indtil nu viser, at dette sædvanligvis sker, når urlen til din hjemmeside afviger fra `web-roots` mappesti.

Hvis standardkoden ikke virker for dig, bør du sætte `$baseUrl` til den, for din server, korrekte værdi.

zf-tutorial/index.php

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

Erstat `'/mysubdir/zf-tutorial/'` med den korrekte URLsti til `index.php`. Hvis din URL til `index.php` f.eks. er `http://localhost/~ralle/zf_tutorial/index.php`, så vil den korrekte værdi til `$baseUrl` være: `'/~ralle/zf_tutorial/'`.

Konklusion

Dette afslutter vores hurtige kig på bygning af en enkel men fuldt fungerende MVC-applikation, der benytter Zend Framework. Jeg håber, du har fundet det interessant og informativt. Hvis du finder fejl, send mig da venligst en e-mail på rob@akrabat.com! *

* Finder du fejl i oversættelsen, send da venligst en e-mail til: s_benning@yahoo.com.

Denne tutorial har kun set på den mest grundlæggende brug af frameworket. Der er mange flere klasser at udforske. Du bør tage et kig i [manualen](http://framework.zend.com/manual) (<http://framework.zend.com/manual>) og også undersøge [wiki](http://framework.zend.com/wiki) (<http://framework.zend.com/wiki>) for dybere indsigt. Er du

interessert i udvikling af frameworket, så er det værd at browse development [wiki](http://framework.zend.com/developer)
(<http://framework.zend.com/developer>).