

BLOGGA MED ZEND FRAMEWORK

Janimatti Ellonen <janimatti.ellonen@gmail.com>

2008

1	Förberedelser	5
1.1	Databasen.....	6
1.2	Nedladdning av Zend Framework.....	6
2	Inuti en ZF-applikation.....	8
3	Zend_Controller	10
4	Zend_View	12
4.1	Zend_View_Helper	12
5	Zend_Model.....	14
6	Zend_Controller_Route	15
7	Zend_Form och Zend_Validate	17
7.1	Zend_Validate.....	18
8	Zend_Db	19
9	Zend_Auth	20
10	Zend_Acl	22
11	Sen då?	24

X Inledning

Denna guide visar hur man skapar en blogg med Zend Framework.

Denna guide visar de grundläggande beståndsdelarna i en ZF-baserad webb-applikation och hur man använder dem. Efter att ha läst guiden, bör läsaren ha en bättre uppfattning om ZF och kunna skapa en egen ZF-applikation.

Zend Framework (ZF) är ett PHP 5 -ramverk som kan användas för att bygga webb-applikationer. ZF består av många olika moduler som kan i princip användas helt separat (med vissa undantag). Med det menas att man kan i sitt nuvarande program använda en eller flera ZF-moduler.

En ZF-baserad webbapplikation följer MVC-metodologin. Detta är tanken i alla fall och så är även fallet med den blogg vi skall skapa! MVC står för Model-View-Controller och är en etablerad teknik för att separera presentationslagret från affärslogiken.

För att guiden inte skall bli tjock som en bok så är bloggens funktionaliteter begränsade till följande:

- ny artikel
- uppdatera artikel
- radera artikel
- sökfunktion
- listning av artiklarna
- inloggning och utloggning av användare
- användar- och accesskontroll

Följande moduler kommer att tas upp:

- Zend_View
- Zend_Controller
- Zend_Form
- Zend_Validate
- Zend_Db
- Zend_Auth
- Zend_Acl

För att kunna följa efter, förstå och skapa bloggen, måste du ha följande installerade på din dator:

- PHP 5 (5.2.4 och nyare)
- PDO (samt PDO-drivrutin för MySQL)
- MySQL 5 (bör fungera även med version 4)
- en texteditor eller IDE (Eclipse, NetBeans, Zend Studio). Jag rekommenderar varmt antingen Zend Studio eller NetBeans då dessa har utmärkt stöd för PHP.
- Apache 2 (1.3 bör fungera också)

Hur du installerar dessa är upp till dig själv.

Därtill måste du ha kunskap om följande:

- PHP 5 och OOP
- MVC
- svn (ifall du vill ladda ned ZF med från en svn repository)
- konfigurering av *httpd.conf* samt *httpd-vhosts.conf*
- hur man skapar en MySql-databas.

Denna guide antar att du använder Apache som webbservar. Guiden använder `/usr/local/apache2/htdocs/` som webbserverns webbroot.

I filen "zfapp.zip" finns den kompletta bloggen.

Guiden är uppdelad i följande delar:

- Del 1 - Förberedelser
- Del 2 - Inuti en ZF-applikation
- Del 3 - Zend_Controller
- Del 4 - Zend_View
- Del 5 - Zend_Model
- Del 6 - Zend_Controller_Route
- Del 7 - Zend_Form och Zend_Validate
- Del 8 - Zend_Db
- Del 9 - Zend_Auth
- Del 10 - Zend_Acl
- Del 11 - Sen då?

1 FÖRBEREDELSE

Ladda ned 'zfapp.zip' och packa upp den. Placera den uppackade katalogen ("zfapp") i htdocs-katalogen (t.ex. `/usr/local/apache2/htdocs`).

Blogg-applikationens struktur består av tre huvudkataloger:

- application
- html
- library

Katalogen "applikation" innehåller applikationens logik.

Katalogen "html" innehåller den publika delen av applikationen, dvs css- och javascript-filerna samt bilderna. Dessutom innehåller den även `index.php` och `.htaccess` (mer om dessa lite senare).

Katalogen "library" innehåller själva Zend Framework. Normalt används "library"-katalogen för tredjeparts-bibliotek som används av applikationen.

Av dessa tre kataloger så skall endast "html" vara synlig för webbläsaren. Vi skall begränsa åtkomsten till enbart "html"-katalogen genom vhosts, som vi konfigurerar härnäst.

Öppna filen 'httpd-vhosts.conf' (på min dator: `/usr/local/apache2/conf/extra/httpd-vhosts.conf`).

Lägg till följande rad t.ex. i slutet av filen:

```
<VirtualHost _default_:9001>
    DocumentRoot /usr/local/apache2/htdocs/zfapp/html
</VirtualHost>
```

De väsentliga delarna i denna konfiguration är siffran 9001 samt värdet på `DocumentRoot`. Siffran (i detta fall 9001) betecknar den HTTP-port som vår applikation kommer att lyssna på. Du kan välja vad du vill, bara den är ledig och ditt system tillåter dig att använda det. T.ex. i många *nix och Linux så måste du ha administratörsrättigheter för att använda portarna 0-1024.

Värdet på `DocumentRoot` visar den verkliga webbrooten för vår applikation.

Denna konfigurering möjliggör användning av följande adress:

<http://localhost:9001/> -> `/usr/local/apache2/htdocs/zfapp/html/index.php` (om vi förmodar att 'index.php' används automatiskt ifall URLen slutar med en '/').

Nu är "application" samt "library" utanför applikationens webbroot!

Spara förändringarna. Vi måste göra en till konfigurering i httpd.conf (på min dator: `/usr/local/apache2/conf/httpd.conf`):

```
Listen 9001
```

Följande direktiv beordrar Apache att lyssna på den angivna porten.

Spara förändringarna och starta om Apache.

1.1 Databasen

Nu skall vi skapa databasen. Ladda ned filen "databas.sql". I den finns den kompletta databasstrukturen för vår blogg samt exempeldata (användare och kategorier). Lättast skapar du databasen genom att klistra in innehållet i phpmyadmin och köra det som en vanlig SQL-fråga. Du kan också göra detta med Mysql client. Hur du bäst vill.

Vi passar på att nu också konfigurera bloggen så att den kan använda databasen. Öppna filen application/Settings.php. Fälten för användarnamn och lösenord är tomma. Fyll i dem.

1.2 Nedladdning av Zend Framework

Vi måste ladda ned Zend Framework. Ladda ned ZF från:

<http://framework.zend.com/download/latest>

På sidan kan du välja mellan "Full" och "Minimal". Ladda ned "Minimal" om du inte bryr dig om demos och tester. Lite längre ned på sidan under rubriken "Latest release" hittar du passande länkar.

Packa upp innehållet och kopiera innehållet i "library" ("katalogen "Zend") till "library" i vår blogg.

Klart! Nu borde alla inställningar vara gjorda. Öppna <http://localhost:9001> i webbläsaren så borde du se följande om allt fungerar:

Main Search	Log in
---	------------------------

No blog entries.

Search

2 INUTI EN ZF-APPLIKATION

Nu börjar det roliga! Innan vi börjar med att studera själva koden så är det passande att först kika lite närmare på hur en ZF-applikation fungerar.

Vår applikation har en enda accesspunkt, index.html. All kommunikation med vår blogg sker genom denna fil. I index.html laddas in en bootstrap-fil (bootstrap.php) som:

- sätter väsentliga sökvägarna (`set_include_path(...)`). Vi måste berätta för vår blogg var t.ex. Zend Framework -biblioteket och andra klasserna kan hittas.
- skapar en ny instans av `Zend_Controller_Front` (front controller)
- laddar in diverse inställningar (sköts av klassen *Initializer*)
- processerar HTTP-förfrågan (`dispatch()`).

I klassen `Initializer` finns följande metoder:

- `initControllers()`
- `initDb()`
- `initPhpConfigs()`
- `initRoutes()`
- `initView()`
- `routeStartup()`

Vi måste även sätta sökvägarna till controller-katalogerna (se `initControllers()`):

```
$this->_front->addControllerDirectory($this->_root .  
'/application/blog/controllers', 'blog');  
  
$this->_front->addControllerDirectory($this->_root .  
'/application/login/controllers', 'login');
```

Metoden `initPhpConfigs()` skapar ett `Zend_config`-objekt av de inställningar som finns i filen `settings.php`. Vår blogg har inte så många inställningar så det enda som finns i `settings.php` är inställningarna för databasuppkopplingen. Här är det viktigt att `type`, `host`, `username` och `password` har korrekta värden. Variabeln `type` låter oss välja vilken sorts databas vi vill använda. I detta fall har vi valt `pdo_mysql` som bestämt säger oss att `MySQL` skall användas!

Metoden `initDb()` skapar ett `Zend_Db`-objekt som vi kan använda för att kommunicera med databasen. Den använder sig utav det `Zend_Config`-objekt som vi tidigare nämnde. Vi väljer också att ha detta `Zend_Db`-objekt som default genom att använda metoden `Zend_Db_Table_Abstract::setDefaultAdapter(...)`;

Senare i koden, när vi behöver en databasuppkoppling, kan vi mycket enkelt komma åt en genom att kalla `Zend_Db_Table::getDefaultAdapter()`. Nu slipper vi skapa en egen wrapper-klass, som gör samma sak åt oss!

Om du tycker att `Zend_Db_Table::getDefaultAdapter()` är tok för långt så kan du göra en wrapper-klass:

```
class Db
{
    public static function connection()
    {
        return Zend_Db_Table::getDefaultAdapter();
    }
}
```

Då kan du använda

```
Db::connection()
```

istället för

```
Zend_Db_Table::getDefaultAdapter()
```

3 ZEND_CONTROLLER

En controller är C:t i MVC. ZF använder två sorts controller-klasser:

- Front Controller (se *bootstrap.php*)
- Action Controller (page controller, se *blog/controllers/IndexController.php* samt *login/controllers/IndexController.php*)

Av dessa så är den senare mer intressant ur vår bloggs synvinkel. Det är den vi kommer att fokusera på.

Front controllerns uppgift är bl. a att förmedla HTTP-förfrågan till rätt Action Controller. Härefter när vi nämner ordet controller så menar vi en Action Controller.

Vår blogg kommer att tillämpa "*Thin Controller/Fat model*" regeln (<http://blog.astrumfutura.com/archives/353-An-Example-Zend-Framework-Blog-Application-Part-2-The-MVC-Application-Architecture.html>). Med "*Thin Controller/Fat model*" menas i princip att kontrollern innehåller så lite som möjligt, medan modellen innehåller affärslogiken och validering av data från t.ex. användaren.

Tanken är att kontrollern agerar som en förmedlare mellan modellen och presentationsskiktet. Modellen genererar ett resultat som genom kontrollern vidarebefordras till presentationsskiktet.

En controller innehåller metoder som åstadkommer (eller delegerar till en viss modell) det som användaren vill. Dessa metoder (controller actions) är oftast mappade till en viss variabel i den anropande adressen. Adressen <http://localhost:9001/blog/list> anger att vi skall använda oss av controller in modulen "blog" samt att vi skall anropa dess metod "`listAction()`".

Just ja, mappningen sker till så att namnen på dessa action-metoder skall sluta med "Action":

```
http://localhost:9001/blog/list -> public void listAction() { ... }  
http://localhost:9001/blog/search -> public void searchAction() { ... }  
OSV...
```

Låt oss ta ett exempel ur vår blogg:

```
class Blog_IndexController extends Zend_Controller_Action
{
    ...

    // http://localhost:9001/blog/list
    public function listAction()
    {
        $model = new BlogModel();

        $entries = $model->fetch(array() );

        if(count($entries) > 0)
            $this->view->entries = $entries;

        return $this->render('list');
    }
    ...
}
```

I ovanstående kodexempel se ni hur en sökning görs med metoden `fetch()` och som resulterar i något som lagras i variabeln `$entries`. Vår controller skall inte hantera visningen av resultatet utan det är vyns uppgift. Följande rad lagrar resultatet i vyn:

```
$this->view->entries = $entries;
```

Senare i vyn kan vi komma åt `$entries` genom:

```
$this->entries
```

Nu undrar ni kanske hur vi kommer åt GET- och POST-värden. I ZF bör `$_GET` och `$_POST` aldrig användas direkt. Dessa två skall accesseras genom wrapper-metoder:

```
// komma åt hela GET-tabellen
$get = $this->_request->getParams(); // eller $this->->getRequest()-
>getParams();

// komma åt en viss GET-variabel
$name = $this->_request->getParam('name'); // eller $this->-
>getRequest()->getParam('name');

// komma åt POST-tabellen
$post = $this->_request->getPost(); // eller $this->->getRequest()-
>getPost();

// komma åt en viss POST-variabel
$name = $this->_request->getPost('name'); // eller $this->->getRequest()-
>getPost('name');
```

Med metoden `$this->_request->isPost()` kan du kontrollera om det formuläret postades som GET eller POST.

4 ZEND_VIEW

Zend_view hanterar visningen och genereringen av vyerna. I kodexemplet ovan kan ni se följande rad, som returnerar den genererade vyn:

```
return $this->render('list');
```

ZF har som en standard att vy-filen mappar mot action-metodens namn:

```
public function listAction() -> 'list.phtml' -> $this->render('list');
public function searchAction() -> 'search.phtml' -> $this->render('search');
```

Vyfilerna har som standard filändelsen '.phtml'.

Vyfilerna innehåller mestadels HTML samt viss PHP-kod för att kunna visa data från t.ex. en modell. Vi vill dock ha så "rena" vyfiler som möjligt som innehåller så lite PHP-kod som möjligt. Detta gör det lättare för bl. a designers att göra ändringar vyerna utan större kunskap om PHP. Till detta kan vi använda Zend_View_Helper.

4.1 Zend_View_Helper

Vyfilerna har ofta viss funktionalitet som återkommer i flera vyer, t.ex. visning av datum. Dessutom vill vi visa om användaren är inloggad eller inte.

Normalt:

```
if($loggedIn)
{
    echo "Welcome xxxx, " . '<a href="#logout.php">Log out</a>;
}
else
{
    echo '<a href="login.php">Log in</a>
}
```

Med en view helper:

```
echo $this->loggedIn();
```

Vår blogg har faktiskt en sådan view helper:

```
class Zend_View_Helper_LoggedIn
{
    ...
    public function loggedIn()
    {
        $auth = Zend_Auth::getInstance();

        $str = '';

        if($auth->hasIdentity())
        {
            $username = $auth->getStorage()->read()->username;

            $str = 'Hello '. $username . ' (<a href="' . $this->_view->
            >baseUrl() . '/login/logout">Log out</a>)' ;
        }
        else
        {
            $str = '<a href="' . $this->_view->baseUrl() . '/login">Log
            in</a>';
        }

        return $str;
    }
    ...
}
```

ZF har flera inbyggda view helpers. Några av dessa har vi faktiskt använt, om än implicit. Titta bara i form-klasserna. Där skapar vi ju formulärinnehållet t.ex. på följande vis:

```
$username = $this->createElement('text', 'username');
```

Grundtanken är att en view helper inte skriver ut sitt innehåll själv med `echo` eller `print()`, utan returnerar innehållet istället.

Metoden `createElement(...)` använder faktiskt internt en view helper för att skapa textfältet åt oss. Då vår blogg använder `Zend_Form` för att generera formulärfälten så har vi inte behövt använda dessa view helpers direkt.

5 ZEND_MODEL

Det finns faktiskt ingen modul med detta namn. I ZF har detta lämnats åt användaren. Vår blogg är uppbyggd på det sättet att kontrollern anropar en viss modell, vars uppgift är att validera indata samt anropa en viss databas-klass som sedan hämtar/kommunicerar med själva databasen.

Exempel ur vår blogg:

```
Blog_IndexController <----> BlogModel <----> BlogDb <----> MySql databas.
```

6 ZEND_CONTROLLER_ROUTE

Vi använder Apaches *mod_rewrite* endast till för att dirigera trafiken till index.php. ZF innehåller egen funktionalitet för att skriva om adresserna.

Detta görs med hjälp av klassen *Zend_Controller_Router_Route*. Exempel kan ni hitta i metoden *initRoutes()* i klassen *Initializer*.

Vår blogg behöver tre enkla "regler". För det första så vill vi att om användaren besöker adressen <http://localhost:9001> så skall han/hon dirigeras om till en sida som visar alla inlägg.

Detta görs med följande regel:

```
$router->addRoute(
    'short', new Zend_Controller_Router_Route('/',
        array( 'module' => 'blog',
               'controller' => 'index',
               'action' => 'list' ) ) );
```

Observera att ovanstående regel aktiveras endast, om adressen inte innehåller GET-variabler.

Andra argumentet till *Zend_Controller_Router_Route* innehåller värden för de variabler som används då dessa inte har angivits. Vill vi tydligt ange dessa värden i adressen, skulle vi kunna göra det på följande sätt:

<http://localhost:9001/blog/index/list>

Vår blogg har endast två controller- classer, en i varje modul. Då båda dessa klasser heter *IndexController* så känns det onödigt att i varje URL ange "index". Därför har vi en passande regel för detta:

```
$router->addRoute(
    'default', new Zend_Controller_Router_Route('/:module/:action/*',
        array( 'controller' => 'index' ) ) );
```

Ovanstående regel förväntar sig att adressen innehåller namnet på modulen och action-metoden. Vi har placerat två behållare för dessa värden. Nämligen "module" och "action". Tidigare nämndes att det är onödigt att inkludera namnet på controller-klassen i adressen så här har vi lagt till "index" som default värde.

Om vi nu t.ex. vill navigera till vår bloggs söksida så kan vi ange följande adress:

<http://localhost:9001/blog/search>

Observera att ovanstående regel avslutas med en `""`. Det är en wildcard och antyder att regeln accepterar flera variabler.

Våg blogg behöver rätt så enkla regler men `Zend_Controller_Router_Route` har mer att ge.

Mer om `Zend_Controller_Router_Route` kan du läsa på sidan

<http://framework.zend.com/manual/en/zend.controller.router.html>.

7 ZEND_FORM OCH ZEND_VALIDATE

Zend_Form har två huvudsakliga funktioner: generering av färdiga formulär samt validering av formulärdata. Följande exempel tagen ur bloggen skapar ett inloggningsformulär:

```
class Form_LoginForm extends Zend_Form
{
    ...
    $this->setAction('/login');
    $this->setMethod('post');

    $username = $this->createElement('text', 'username');
    $username->setLabel('Username');

    $password = $this->createElement('password', 'password');
    $password->setLabel('Password');

    $submit = $this->createElement('submit', 'submit');
    $submit->setLabel('Login');

    $this->addElement($username);
    $this->addElement($password);
    $this->addElement($submit);
    ...
}
```

Ovanstående formulär kan sedan visas genom:

```
// genererar ett inloggningsformulär med textfält för användarnamn och lösenord samt en submit-knapp
echo $this->form->render();
```

Den andra viktiga funktionen är validering av formulärdata. Du kan lätt lägga till valideringsregler som bestämmer att användarnamnet måste vara 6-20 bokstäver långt och måste innehålla minst en siffra (varför just en siffra låter vi läsaren avgöra).

Valideringen görs t.ex. på följande sätt:

```
if($form->isValid($_POST) )
{
    ...
}
else
{
    ...
}
```

7.1 Zend_Validate

Zend_Validate hjälper dig med validering av formulärdata. Dessa nämndes redan ovan.

Låt oss återigen ta ett exempel från bloggen (se klassen `Form_CreateForm` (*blog/controllers/models/Form*)):

```
// skapar ett textfält för rubriken
$topic = $this->createElement('text', 'topic');
$topic->setLabel('Topic');

// rubrikens längd: 3 eller fler tecken
$topicLengthValidator = new Zend_Validate_StringLength(3);
// alternativt felmeddelande
$topicLengthValidator->setMessage('The topic is too short',
Zend_Validate_StringLength::TOO_SHORT);

// Vi kan ha flera validatorer till samma fält

$topicValidator = new Zend_Validate();

$topicValidator->addValidator($topicLengthValidator, false);
$topicValidator->addValidator($topicContentValidator, false);

$topic->addValidator($topicValidator);

Själva valideringen:

...
if($form->isValid($data) ) // $data är $_GET eller $_POST
{
    ...
}
```

Valideringsklasserna kan också användas utan `Zend_Form`. Detta får dock bli en extrauppgift för läsaren.

8 ZEND_DB

Zend_Db är ett gränssnitt för lättare kommunikering med en databas. Modulen innehåller bl. a funktioner för att lagra (INSERT), uppdatera (UPDATE), ta bort (DELETE) och hämtning av data (SELECT).

Följande INSERT:

```
INSERT INTO foo (age, name) VALUES (25, 'jme');
```

Kan med Zend_Db skrivas:

```
$db->insert('foo', array('age' => 25, 'name' => 'jme'));
```

Följande SELECT:

```
SELECT u.username FROM users u WHERE id = 1;
```

blir då:

```
$id = 1;
$select = $db->select()
    ->from(      array('u' => 'users'),
                array('username') )
    ->where('id = ?', $id);

$username = $db->fetchOne($select);
```

Ovanstående fråga kunde även ha skrivits lite lättare:

```
$username = $db->fetchOne('SELECT u.username FROM users u WHERE id = ?',
    $id);
```

Följande DELETE:

```
DELETE FROM users WHERE id = 1;

$db->delete('users', $db->quoteInto('id = ?', $id));
```

Det mer komplicerade sättet kommer till hands då vi måste göra flera joins bl. a. Sedan måste man komma ihåg att använder man ZFs sätt att skapa SQL-frågorna så är de mer kompatibla med andra RDBMS. Skriver du SQL-frågorna manuellt så finns det större risk för att du måste göra förändringar om du byter till en annan RDBMS.

9 ZEND_AUTH

Zend_Auth är en modul för att identifiera en användare och säkerställa att användaren är den han/hon utsäger sig för att vara. Zend_Auth innehåller flera "adaptrar" som möjliggör autentisering mot olika källor, t.ex:

- databas
- LDAP
- OpenId
- och fler...

Vi har valt att använda databasen som källa och guiden kommer att förklara Zend_Auth ur databasens synvinkel. Mer om Zend_Auth finns att läsa på sidan

<http://framework.zend.com/manual/en/zend.auth.html>.

När vi använder en databas som källa måste vi ange namnet på användartabellen samt fälten för användarnamnet och lösenordet:

```
$adapter = Zend_Db_Table::getDefaultAdapter();

$authAdapter = new Zend_Auth_Adapter_DbTable($adapter);
$authAdapter->setTableName('users');
$authAdapter->setIdentityColumn('username');
$authAdapter->setCredentialColumn('password');
```

Sedan måste vi ge adaptern användarnamnet och lösenordet från formulärfältet:

```
// notera att $_POST används för exemplets skull
$authAdapter->setIdentity($_POST['username']);
$authAdapter->setCredential($_POST['password']);
```

Nästa steg blir själva autentiseringen:

```
$auth = Zend_Auth::getInstance();
$result = $auth->authenticate($authAdapter);

if($result->isValid() )
{
    $data = $authAdapter->getResultRowObject(null, 'password'); // vi
vill inte lagra lösenordet
    $auth->getStorage()->write($data);

    echo "KTHXBYE";
}
else
{
    echo "FAIL";
}
```

För att se, om användaren är inloggad:

```
// i princip samma kod finns att hitta i vår view helper
Zend_View_Helper_LoggedIn
$auth = Zend_Auth::getInstance();

if($auth->hasIdentity() )
{
    $username = $auth->getStorage()->read()->username;

    echo "Welcome $username";
}
```

Notera att vår blogg inte hanterar lösenord på ett korrekt och säkert sätt. Lösenordet lagras som klartext i databasen. Lösenordet borde hashas och saltas men det får bli en hemläxa!

Autentisering av användaren skall även göras vid varje HTTP-förfrågan och inte bara då vi vill visa alternativ text i vyn. Mer om hur detta görs får ni veta när vi behandlar Zend_Acl.

10 ZEND_ACL

Där Zend_Auth användes för att identifiera användaren, används Zend_Acl till för att verifiera om användaren har access till en viss resurs/sida. I ZF talar vi om roller och resurser; en roll har access till vissa resurser.

Vår blogg behöver endast två roller:

- administratör (admin)
- besökare (guest)

En administratör kan skriva, uppdatera och ta bort inlägg medan besökaren kan endast visa, lista och söka efter inlägg (givetvis har administratören samma rättigheter som besökaren).

Vi har valt att skapa rollerna och resurserna i en controller. Dessa kunde även ha lagrats i en config-fil eller i databasen.

Rollerna skapar vi så här:

```
$acl = new Zend_Acl();

// notera att addRole() implementerar fluent interface
(http://en.wikipedia.org/wiki/Fluent_interface)
$acl->addRole(new Zend_Acl_Role('admin') )
      ->addRole(new Zend_Acl_Role('guest') );
```

Nästa steg är att definiera resurserna. Lyckligtvis så behöver vår blogg rätt så få resurser så det blir inte så mycket att skriva.

Vad är en "resurs"? Jo, i vårt fall så pekar resursen mot en viss action-metod i en viss controller.

```
$acl->add(new Zend_Acl_Resource('blog') );           // blogg controllern
$acl->add(new Zend_Acl_Resource('create'),           'blog' ); // metoden
createAction()
$acl->add(new Zend_Acl_Resource('delete'),           'blog' ); // metoden
deleteAction()
$acl->add(new Zend_Acl_Resource('index'),            'blog' ); // osv ...
$acl->add(new Zend_Acl_Resource('list'),             'blog' );
$acl->add(new Zend_Acl_Resource('search'),           'blog' );
$acl->add(new Zend_Acl_Resource('view'),            'blog' );

// vi vill att administratören skall ha full access
$acl->allow('admin', 'blog');

// begränsade rättigheter för besökaren
$acl->allow('guest', 'blog', 'index');
$acl->allow('guest', 'blog', 'list');
$acl->allow('guest', 'blog', 'search');
$acl->allow('guest', 'blog', 'view');
```

Zend_Acl är mer kraftfull och erbjuder bl. a roller som ärver rättigheterna av en roll. Då vår blogg är så enkel så är vi nöjda med ovanstående konfiguration!

Nu till det viktiga, nämligen validering av förfrågan.

Vi behöver veta två saker:

- namnet på modulen
- namnet på action

Dessa två får vi fram genom:

```
$module = $this->_request->getModuleName();  
$action = $this->_request->getActionName();
```

Själva valideringen:

```
$role = Zend_Auth::getInstance()->hasIdentity() ? 'admin' : 'guest';  
  
if($acl->isAllowed($role, $module, $action) )  
{  
    echo "ACCESS GRANTED";  
}  
else  
{  
    echo "ACCESS DENIED!";  
}
```

11 SEN DÅ?

Nu har vi gått igenom mycket av det goda i ZF. Vår blogg är inte perfekt ännu. Några förbättringar jag kommer på är:

- lagring av data i en **cache** så att vi kan t.ex. minska antalet databasanrop (databasanrop är dyrbara tidsmässigt och bör minimeras)
- Internationalisering (l18n). Vår blogg är just nu på engelska men med lite arbete så kan vi bygga in stöd för flera olika språk
- mer avancerad autentisering av användare
- en modul för hantering av användare och kategorier.

Jag har även använt Zend_Layout utan närmare förklaring. Med Zend_Layout blir det lättare att använda en viss HTML som bas för alla eller flera vyer. Om ni har varit observanta så har ni märkt att vyfilerna inte innehåller `<html>` eller `<body>` taggar utan dessa är definierade med Zend_layout i en layout-fil som alla vyer bakas in i. Vi behöver således definiera `<html>`, `<head>` och `<body>` EN GÅNG. Som en extra bonus slipper vi också inkludera vår meny i varje vyfil. Smart va?

Här är en lista med intressanta webbsidor angående ZF och diverse annat inom programmering och livet:

- <http://framework.zend.com/manual/en/>
- <http://akrabat.com/>
- <http://blog.astrumfutura.com/>
- <http://codeutopia.net/blog/>
- <http://weierophinney.net/matthew/>
- <http://www.dasprids.de/>

Hittar du fel (sakfel och grammatiska) eller om du har andra synpunkter och förslag så får du gärna berätta! Om du länkar till detta dokument så får du det men jag vill gärna veta.

Maila mig på janimatti.ellonen@gmail.com.