Youssef Taoudi - yousseft@kth.se
11 januari 2018

# Seminarie 3 - Rapport

## Internet Applications, ID1354

## Introduction

The task for this seminar was to improve the previous PHP application, this time using the MVC architecture, a database and some security mechanisms.

I worked on this task by myself.

## Literature

I skimmed through the lectures on the KTH-website and watched the id1354 framework tutorial on the KTH-website. The main things I studied was how the MVC-architecture was built in PHP and how to use the built in functions in the framework.

I used the id1354-framework for the MVC architecture, netbeans as coding software and firefox developer tools.

## Method

I started out by creating a view class in the View layer for each user interface in the views folder and from there I started expanding the application. The first functionality added was the login function, then the register function and lastly the comment functions. After all functionalities worked as expected, I implemented the security measures.

## Result

GITHUB: https://github.com/Taoudi/ID1354

I will divide this section into several sections, one for each piece of code I feel is important to the website. The methods used for most of the application are fairly similar.

**View:**

Figure 1 shows on of the View classes. Post variables are set using set functions and the values and are filtered if they are used. Page is filtered directly in the set function and comment is checked before it is used in the doExecute function. The doExecute function creates a controller object that is used to call methods in the model. Also, a session variable is gotten using the session->get method from the framework.

```php
namespace recept\View;

use Id1354fw\View\AbstractRequestHandler;
use recept\Controller\CommentController;

class deletecomment extends AbstractRequestHandler {
    private $page;
    private $comment;
    private $user;
        public function setComment($comment){
            $this->comment = $comment;
        }
           public function setPage($page){
                  if(ctype_alpha($page)){
                   $this->page = $page;
                }
                else{
                    $this->page = "startpage";
                }
        }

    protected function doExecute() {
        $this->contr = new CommentController();
        $this->user = $this->session->get('uname');
          if(ctype_print($this->comment) && ctype_print($this->user) ){
                $validcomment = $this->contr->deleteComment($this->user, $this->comment,$this->page);
                $this->addVariable('validcomment',!$validcomment);
                $this->addVariable('comments',$this->contr->getComments($this->page));
          }
        return $this->page;
    }

}
```

Figur 1

Kod för delete comment i View

**Controller:**

Figur 2 visar en bild på Controllern för kommentarhantering. construct metoden skapar en ny commentHandler som är en klass i model-lagret. Resten av dess funktioner är metodkallelser till modellen.

```php
*/
class CommentController {
    private $commentHandler;
    public function __construct(){
        $this->commentHandler = new CommentHandler();
    }

    public function getComments($page){
      $copy = $this->commentHandler->getComments($page);
     return $copy;
  }

  public function enterComment($username,$comment,$page){
      return $this->commentHandler->enterComment($username,$comment,$page);
  }

  public function deleteComment($username,$comment, $page){
       return $this->commentHandler->deleteComment($username,$comment,$page);
  }

}
```

Figur 2

Kod för CommentController i Controller lagret

**Model:**

I det här lagret sker all logik. Figur 3 visar hur modellen för kommentarer ser ut.

enterComment och deleteComment gör inte så mycket, de kallar bara en metod i integrationslagret som kör en SQL-query. enterComment har ingen logik, den anropar bara databasmetod. deleteComment avkodar html-entity så att den kan ta bort motsvarande kommentar (man måste avkoda två gånger av någon anledning). getComments samlar alla värden som en array och skickar tillbaka det till controllern.

```php
class CommentHandler {
    private $commentData;
    public function __construct(){
        $this->commentData = new commentDAO();
    }

    public function enterComment($user,$comment,$page){
        return $this->commentData->enterComment($user, $comment,$page);
    }

     public function getComments($page){
         $arr = array( 0=> array() , 1=> array());
         $result =  $this->commentData->getComments($page);

        if($result->num_rows>0){
            while($row=$result->fetch_assoc()){
                array_push($arr[0],htmlentities(htmlentities($row["user"], ENT_QUOTES), ENT_QUOTES));
                array_push($arr[1],htmlentities(htmlentities($row["comment"], ENT_QUOTES), ENT_QUOTES));
            }
        }
        else{
          array_push($arr, "no result");
        }
        return $arr;
    }

    public function deleteComment($user,$comment,$page){
        return $this->commentData->deleteComment(html_entity_decode($user), html_entity_decode($comment),$page);
    }
}
```

Figur 3

Kod för CommentHandler i model lagret

**Integration:**

Figur 4 visar hur man skapar en connection till databasen genom konstruktör. I getComments ser man till genom en switch att $page argumentet faktiskt är ett valid table name (borde möjligtvis ha gjorts i model) och query skrivs sen in och anropas i databasen.

```php
namespace recept\DAO;

class commentDAO {

    private $servername = "127.0.0.1";
    private $username = "root";
    private $password = "1234";
    private $dbname = "recept";
    private $conn;

    //create and check connection
    public function __construct() {
        $this->conn = new \mysqli($this->servername, $this->username, $this->password, $this->dbname) or die("connection failed");
    }

    // Returns result of query
    public function getComments($page) {
        switch ($page) {
            case "pannkakor":
                $query = "SELECT comment, user from pannkakor";
                break;
            case "kottbullar":
                $query = "SELECT comment, user from kottbullar";
                break;
            default :
                return false;
        }

        $result = $this->conn->query($query);
        return $result;
    }
```

Figur 4

Kod för commentDAO i integration lagret

```php
array_push($arr[0],htmlentities(htmlentities($row["user"], ENT_QUOTES), ENT_QUOTES));
array_push($arr[1],htmlentities(htmlentities($row["comment"], ENT_QUOTES), ENT_QUOTES));
```

Figur 6

Kodstycke för att ändra specialtecken

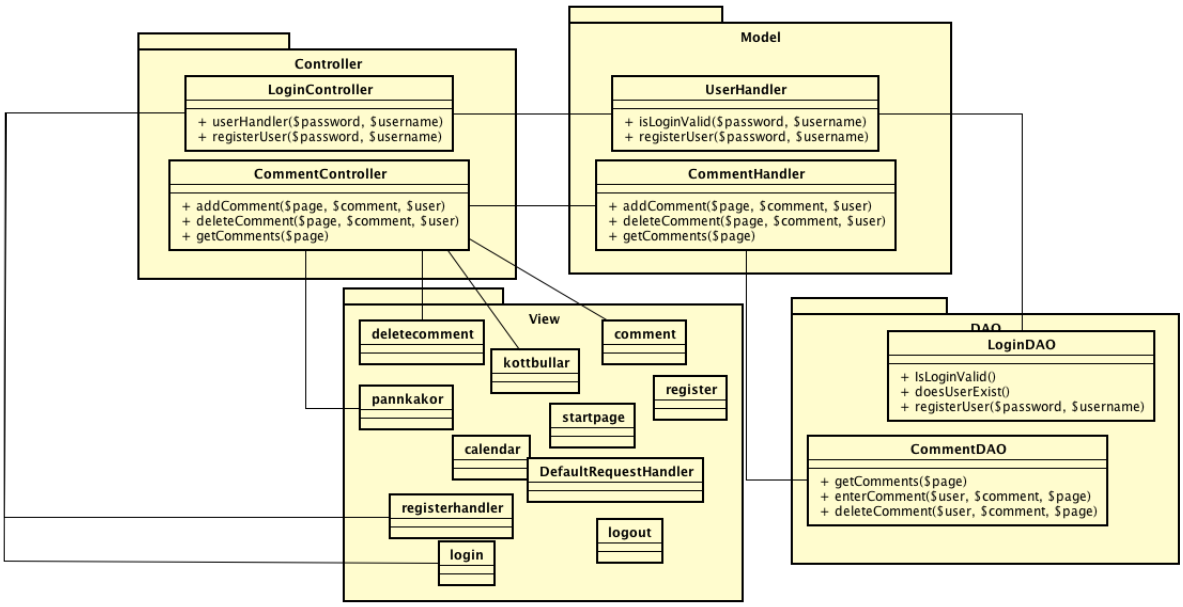
```php
pass = password_hash($password, PASSWORD_DEFAULT);
```

Figur 5

Kodstycke för hashing av lösenord

**Security:**

Security-wise, I decided to focus on handling and filtering input which is why I chose to implement password encryption, input filtering and XSS measures into the application. Figure one shows a piece of code where posted variables are checked before being used, using the ctype_function. Figur 5 visar hur en lösenord krypteras med hjälp av en inbyggt hashing metod i UserHandler klassen i modellen. Figur 6 visar ett kodstycke där metoden html-entities används för att ändra vissa specialtecken till entities för att undvika html-kod. Av någon anledning fungerar metoden endast om man använder den två gånger. Jag använde inte HTTPS i applikationen, men den bör användes då man skickar lösenordet mellan filer.




Figur 7

Klassdiagram

Klassdiagrammet i figur 7 visar alla namespaces och deras klasser, samt hur de är associerade. Klasser i view är bara associerade med klasser i controller. Klasser i controller är bara associerade med klasser i model och model är bara associerade med klasser i DAO (integration). Några saker jag märker nu när jag kollar på diagrammet är att flera metoder har långa parametrar vilket skadar inkapslingen (encapsulation). Jag skulle till exempel ha kunnat skapa en objekt UserDTO som kunde hållit information för använder namn och lösen ord och sedan passerat det i parametrar. En till sak är att vissa namn, framförallt LoginController och LoginDAO borde ha förbättrats eftersom utöver att logga in så registrerar de också användare vilket kan tyda på låg cohesion.

## Discussion

Uppgiften var att förbättra PHP strukturen på föregående uppgift med hjälp av MVC arkitekturen med hjälp av ett framework och genom objektorientering vilken jag anser själv anser har följts. Duplicerad kod har undvikts. Jag har dessutom implementerat tre säkerhetsmekanismer i form av lösernordkryptering, input filter samt skydd för cross site scripting. Jag har även använt en databas som jag sätter in, tar bort och hämtar värden ifrån genom SQL (figur 4).

## Comments about the course

Det tog runt 20-25 timmar att göra denna uppgift (möjligtvis mer).