Seminarium 3

Applikationer för internet, grundkurs, ID1354

Maher Jabbar | maherj@kth.se Okt 28, 2016

Innehåll

1 Introduction

The tasks of this seminar is to choose between the MVC architecture without a framework or a PHP framework. It will be used for the website has been created in order to provide a more structured code that is easy to understand when you read the code. It would also have a low coupling, high cohesion and good encapsulation because it is the most sought when using the MVC, and a simple object-oriented design concept.
The next task is to make the website more secure by implementing three of the following security sections from nine lecture. The reason for it is that you want to avoid special outside attacks that try to destroy one's website.

2 Literature study

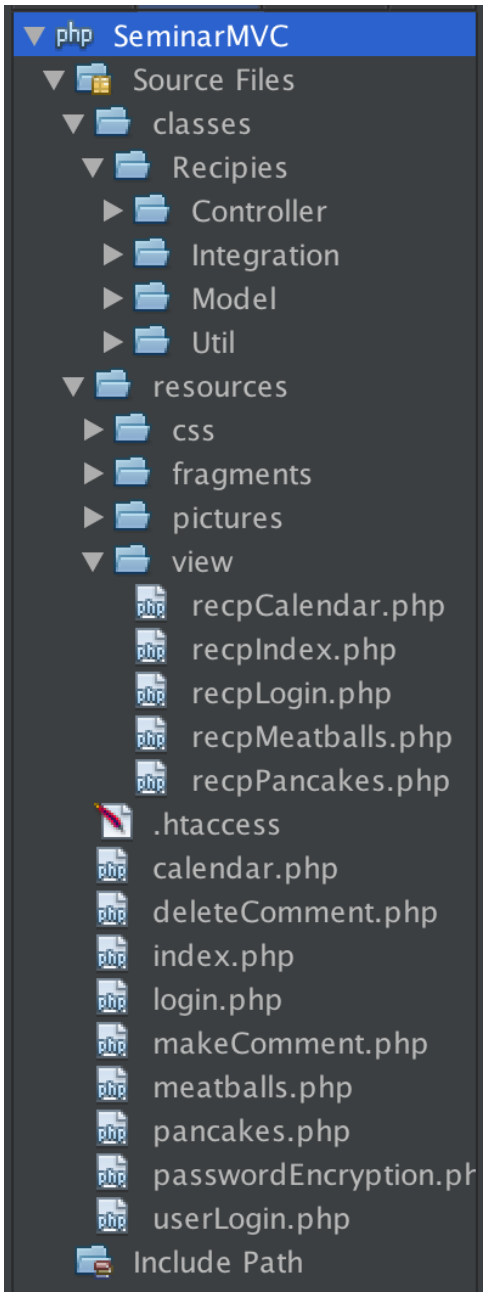
It was a while ago when i used a MVC architecture so i have got to check out a bit back on the old notes from object-oriented design course, which went through a lot of MVC and they also took the help of assistants from the guide that every good and the lectures that had useful info about MVC.

3 Metod

I've been using NetBeans to solve this task. The first thing I did was to create a new project and divide the program into different layers Model, View, Controller, util and integration. When this was done began to put the code I had from previous seminar in the new project, to ensure that all user actions in the view led to a method call in the controller, this led to a method call in a lower layer. When this is done i started to implement three different collateral. The duties i selected were input filtering, password encryption and cross site scripting. The first part input filtering solved by checking that all user input is only contained signs that produces output with command ctype_print (). The second part was password encryption was first made by encrypting passwords when a user was registered and the password was stored, and then decrypt it when the user logged in and the password was read from the file where it was saved. The last part cross site scripting was solved by making sure not to put the data into an attribute name, attribute value, tag, CSS, HTML comment.

4 Results

To implement the MVC program was to divide the different parts. Besides parts Model, View and controller I added folders integration and util.



Figur 1. How everything is divided

This shows how the program is divided into the different layers. All code in the view placed in a separate directory from the rest of the classes. When a user performs an action in the view it will lead to a method call in the controller layer, which will lead to a method call in a layer under the controller. The following example shows that the MVC pattern is followed.

```
<html>
    <head>
        <link rel="stylesheet" type="text/css" href="resources/css/foodcss.css">
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php include 'resources/fragments/menuLogin.php'; ?>
        <?php include 'resources/fragments/activeuser.php'; ?>
        <h1 class="reglog">Please enter nickname and password</h1>
        <div class="form">
            <form action="userLogin.php" method="post">
                <label for="name">Nick name</label>
                <input type="text" name="nameLogin"><br>
                <label for="password">Password</label>
                <input type="password" name="passwordLogin"><br>
                <input type="submit" value="Log in">
            </form>
        </div>
    </body>
</html>
```

Figure 2. How a user logs in

Figure 2 shows the HTML code of the page where a user can log in which is located in the folder view in the folder Resources.

```php
<?php

namespace Recipies\View;

use \Recipies\Util\Util;
use Recipies\Controller\Controller;
use Recipies\Controller\SessionManager;

require_once ('classes/Recipies/Util/Util.php');
Util::initRequest();

$name = $_POST['nameLogin'];
$password = $_POST['passwordLogin'];
$controller = SessionManager::getController();
$controller->loginUser($name, $password);
SessionManager::storeController($controller);

if($_SESSION['name']){
    require_once ('index.php');
}else{
    require_once('login.php');
}
```

Figur 3. userLogin.php

```php
<?php
namespace Recipies\Util;


final class Util {

    private function __construct() {

    }

    public static function initRequest() {
    spl_autoload_register(function ($class) {
        require_once 'classes/' . \str_replace('\\', '/', $class) . '.php';
    });

        @session_start();
    }

}
```

Figur 4. util.php

In the code when a user fills their name and password in the form shown in Figure 3, it will be executed. Then also first method initRequest class util run as shown in Figure 4. This method frees us from the include and require_once statments later. Once that's done, you get a controller object from the Session Manager. After username and password are sent from a form sent with method loginUser in the controller.

```php
use Recipies\Integration\UserHandler;
use Recipies\Integration\CommentHandler;


class Controller {

    private $uhandler;
    private $chandler;

    public function __construct(){
        $this->uhandler = new UserHandler();
        $this->chandler = new CommentHandler();
    }

    public function loginUser($name, $password){
       $this->uhandler->login($name, $password);
    }
```

Figure 5. The class Controller

Figure 5 shows the code from the controller, the controller contains objects of a user handler, which handles user registrations and logins, which is located in the integration layer. When the method loginUser user called in the controller then the controller calls the login in its userHandler and sends the name and password to the userHandler.

```php
public function login($name, $password) {
    if (ctype_print($name) AND ctype_print($password) AND ( !empty($name)) AND ( !empty($password))) {

        $users = explode(";\n", file_get_contents($this->file));
        $user = new User($name, $password);
        for ($i = 1; $i < count($users); $i++) {
            $compUser = unserialize($users[$i]);
            if ($compUser instanceof $user) {
                if (password_verify($password, $compUser->getPassword())) {
                    if ($compUser->getUsername() == $name) {
                        $_SESSION['name'] = $name;
                    }
                }
            }
        }
    } else {
        header('Location: login.php');
    }
}
```

Figur 6. koden i userHandler.php

```php
public function login($name, $password) {
    if (ctype_print($name) AND ctype_print($password) AND ( !empty($name)) AND ( !empty($password))) {

        $users = explode(";\n", file_get_contents($this->file));
        $user = new User($name, $password);

public function addComment($name, $comment, $dish) {
    if (ctype_print($comment) AND ( !empty($comment))) {
        $comment = htmlentities($comment, ENT_QUOTES);
        $newcomment = new Comment($name, $comment, $dish);
        file_put_contents($this->file, serialize($newcomment) . ";\n", FILE_APPEND);
    }
}

 public function addUser($name, $password) {
    if (ctype_print($name) AND ctype_print($password) AND ( !empty($name)) AND ( !empty($password))) {
        $name = htmlentities($name, ENT_QUOTES);
        $password = password_hash($password, PASSWORD_DEFAULT);
        $user = new User($name, $password);
        file_put_contents($this->file, serialize($user) . ";\n", FILE_APPEND);
```

Figur 7. Input filtering

There are three situations when a user to provide input to the web page when the user signs up when the user logs in and when the user writes a comment. To implement input filter as the method is ctype_print ($ input) and! Empty ($ input) to validate the input, and that it contains no controll characters.

```php
public function addUser($name, $password) {
    if (ctype_print($name) AND ctype_print($password) AND ( !empty($name)) AND ( !empty($password))) {
        $name = htmlentities($name, ENT_QUOTES);
        $password = password_hash($password, PASSWORD_DEFAULT);
        $user = new User($name, $password);
        file_put_contents($this->file, serialize($user) . ";\n", FILE_APPEND);
    } else {
        header('Location: Register.php');
    }
}
 public function login($name, $password) {
    if (ctype_print($name) AND ctype_print($password) AND ( !empty($name)) AND ( !empty($password))) {

        $users = explode(";\n", file_get_contents($this->file));
        $user = new User($name, $password);
        for ($i = 1; $i < count($users); $i++) {
            $compUser = unserialize($users[$i]);
            if ($compUser instanceof $user) {
                if (password_verify($password, $compUser->getPassword())) {
                    if ($compUser->getUsername() == $name) {
                        $_SESSION['name'] = $name;
                    }
                }
            }
        }
    } else {
        header('Location: login.php');
    }
}
```

Figur 8. Password Encryption

Password encryption was performed using the password hash function () and password_verify shown in the code. When a user registers the password is encrypted with the password hash () function shown in the figure , that specified by the user password that will authenticate against the password that is stored on the server that is decrypted.

```php
public function addComment($name, $comment, $dish) {
    if (ctype_print($comment) AND ( !empty($comment))) {
        $comment = htmlentities($comment, ENT_QUOTES);
        $newcomment = new Comment($name, $comment, $dish);
        file_put_contents($this->file, serialize($newcomment) . ";\n", FILE_APPEND);
    }
}
```

Figur 9. Cross site scripting

To prevent cross site scripting we made sure not to put any data to attribute names, attribute values, tags, CSS, HTML comments or ¡script¿ elements. Above of all input given from a user who can later be added to an HTML element, the function htmlentities () was run to convert HTML special characters like the ¡& to entities. Input from a user that can later be added to an element and comes from when the user writes his name when the user registers or logs in and when the user writes a comment that htmlentities () function is shown in Figure 8.

5 Discussion

this was a while since I used a MVC pattern where I am, as I said had to check on the old notes, etc. to get started with the first task. When it came to the security of the website, where it is not as complicated as I thought in the beginning because there were lot of info from the lectures, on the Internet and that the assistants had a good handle on how they could be used in the code to avoid security threats.
I feel I have a better grip on the object-oriented design PHP, using the MVC pattern for PHP and as well as how to avoid basic security of a website. Now that I know the collateral available and which can be used.

6 Comments about the course

I spent about 30 hours to keep up with the MVC and write the report, it was pretty tough but eventually succeeded. I think this course is well structured with a space between each seminar.