Assignment 3

Applikationer för internet, ID1354

Max Körlinge, korlinge@kth.se

November 25, 2017

Contents

1 Introduction

This assignment concerns structuring the web application to follow the MVC pattern, and to implement three security measures from a list. Optional tasks were to use a database to store data, and to improve performance. I chose to use a database, but not improve performance. For the mandatory task 1 I chose to implement the MVC architecture without using a framework.

2 Literature Study

To complete the tasks I first studied the course lecture notes on MVC for a web application and web security.

3 Method

3.1 Task 1a

I started trying to "happy hack" my way through this assignment but quickly realized that I had to have a plan, so I started off by making a class diagram in Astah-community. I started by implementing all actions a user could take as a function in the Controller, and then added operations and other classes as I saw fit. After the MVC diagram was finished I coded each action one by one (for example first logging in a user, then registering a user, then writing a comment, etc), changing the class diagram when necessery. I use view, controller, model, and integration layers. The view layer has all pages, HTTP requests, and parts that are included in the views, such as header and footer. All the other layers are pure PHP classes that are called from the view (through the controller). To make objects persist I made the Controller destruct method serialize the controller in the session cookie.

3.2 Task 2

I chose to implement File Security. Apache2 works as a user who only deals with the webroot (www-data), and when working with the site I use a small bash script that copies the files into the web root, and then sets `chmod -R 755` on the directories, which means that only root user has access to changing the files, all other users, such as the apache user, can only read and execute files inside. I also denied HTTP access to all files containing PHP classes through the apache configuration.

I also chose to implement database security. I had already set up my mySQL database so that a special user was created who only had access to the database pertaining to the website. I then changed all the methods in the integration layer class `DatabaseRequest`, which handles database requests, to use parameterized requests using `prepare()` and `execute()`, instead of the `query()` method I was using before.

At last I chose to implement password encryption. Since I first restructured the program to use the MVC architecture, I could simply change the `loginUser()` and `registerNewUser()` methods in the `UserAccountHandler` class to use `password_hash` and `password_verify` PHP functions instead of storing the password as plain text.

3.3 Optional Task 1

I actually chose to use a database already in the last assignment. To set up the database quickly on different servers when I change environment, I wrote a sql file that does this if you source it from the mySQL command line. You can find it at this link. I then use SQL queries and the PDO library to fetch and insert data into the database.

4 Result

The git repository can be found here.

4.1 Task 1

All source code is inside the `src` directory, where the layers are: `controller`, `model`, `integration`, and `util`. Inside each map are the class files, one file per class. The structure is best described by the class diagram in Figure 4.1. There is only mixed PHP and HTML in the `view` layer, all other code is object oriented PHP. There are only three static methods: two in the `Util` class used by the `view` layer for redirections, and one, also used by the view, for getting the `Controller` from the `SESSION` cookie. All layers have the roles specified by the MVC pattern, and there are only dependencies ”downwards” through the layers, as can be seen in the class diagram.

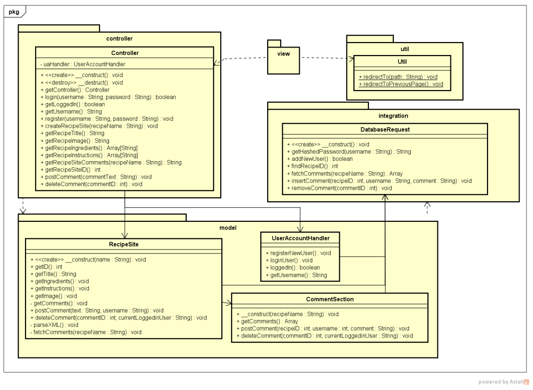
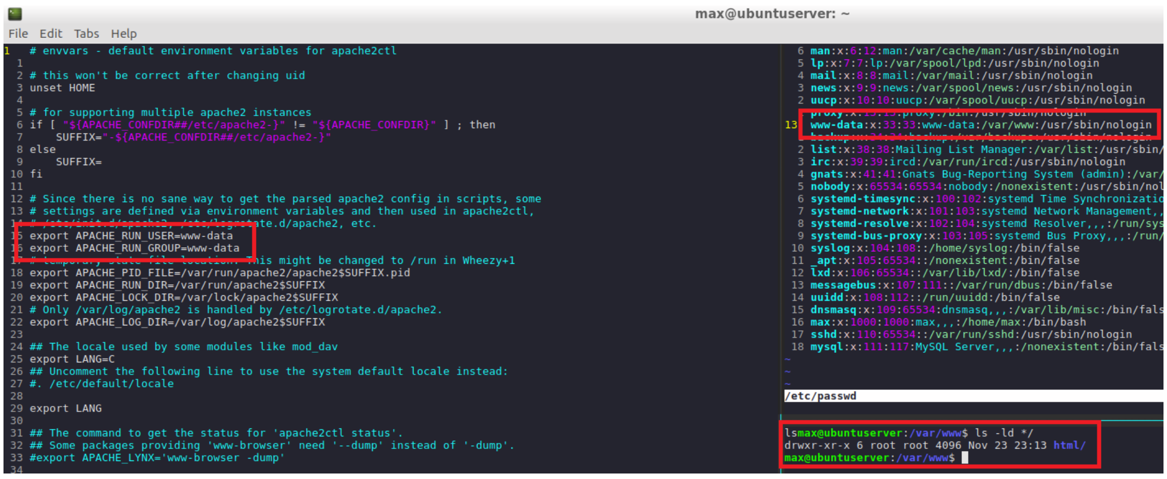

pkg
controller
Controller
- uaHandler : UserAccountHandler
+ <<create>> __construct() : void
+ <<destroy>> __destruct() : void
+ getController() : Controller
+ login(username : String, password : String) : boolean
+ getLoggedIn() : boolean
+ getUsername() : String
+ register(username : String, password : String) : void
+ createRecipeSite(recipeName : String) : void
+ getRecipeTitle() : String
+ getRecipeImage() : String
+ getRecipeIngredients() : Array[String]
+ getRecipeInstructions() : Array[String]
+ getRecipeSiteComments(recipeName : String) : String
+ getRecipeSiteID() : int
+ postComment(commentText : String) : void
+ deleteComment(commentID : int) : void
view
util
Util
+ redirectTo(path : String) : void
+ redirectToPreviousPage() : void
integration
DatabaseRequest
+ <<create>> __construct() : void
+ getHashedPassword(username : String) : String
+ addNewUser() : boolean
+ findRecipeID() : int
+ fetchComments(recipeName : String) : Array
+ insertComment(recipeID : int, username : String, comment : String) : void
+ removeComment(commentID : int) : void
model
RecipeSite
+ <<create>> __construct(name : String) : void
+ getID() : int
+ getTitle() : String
+ getIngredients() : void
+ getInstructions() : void
+ getImage() : void
- getComments() : void
+ postComment(text : String, username : String) : void
+ deleteComment(commentID : int, currentLoggedinUser : String) : void
- parseXML() : void
- fetchComments(recipeName : String) : void
UserAccountHandler
+ registerNewUser() : void
+ loginUser() : void
+ loggedIn() : boolean
+ getUsername() : String
CommentSection
+ __construct(recipeName : String) : void
+ getComments() : Array
+ postComment(recipeID : int, username : int, comment : String) : void
+ deleteComment(commentID : int, currentLoggedinUser : String) : void
powered by Astah

Figure 4.1: A class diagram describing the Tasty Recipes web application

4.2 Task 2

I do not use HTTPS but HTTPS should be used whenever HTTP requests are being made by an authenticated user such as logging in, registering, or posting or deleting comments.

The file security of the system is strong on the webserver, as seen in Figure 4.2 and Figure 4.3.



```
# envvars - default environment variables for apache2ctl

# this won't be correct after changing uid
unset HOME

# for supporting multiple apache2 instances
if [ "${APACHE_CONFDIR##/etc/apache2-}" != "${APACHE_CONFDIR}" ] ; then
    SUFFIX="-${APACHE_CONFDIR##/etc/apache2-}"
else
    SUFFIX=
fi

# Since there is no sane way to get the parsed apache2 config in scripts, some
# settings are defined via environment variables and then used in apache2ctl,
# /etc/init.d/apache2, /etc/logrotate.d/apache2, etc.
export APACHE_RUN_USER=www-data
export APACHE_RUN_GROUP=www-data
# temporary state file location. This might be changed to /run in Wheezy+1
export APACHE_PID_FILE=/var/run/apache2/apache2$SUFFIX.pid
export APACHE_RUN_DIR=/var/run/apache2$SUFFIX
export APACHE_LOCK_DIR=/var/lock/apache2$SUFFIX
# Only /var/log/apache2 is handled by /etc/logrotate.d/apache2.
export APACHE_LOG_DIR=/var/log/apache2$SUFFIX

## The locale used by some modules like mod_dav
export LANG=C
## Uncomment the following line to use the system default locale instead:
#. /etc/default/locale

export LANG

## The command to get the status for 'apache2ctl status'.
## Some packages providing 'www-browser' need '--dump' instead of '-dump'.
#export APACHE_LYNX='www-browser -dump'
```

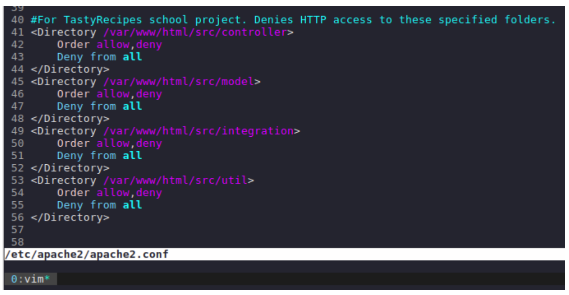
```
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nol
systemd-timesync:x:100:102:systemd Time Synchronizatio
systemd-network:x:101:103:systemd Network Management,,
systemd-resolve:x:102:104:systemd Resolver,,,:/run/sys
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/
syslog:x:104:108::/home/syslog:/bin/false
_apt:x:105:65534::/nonexistent:/bin/false
lxd:x:106:65534::/var/lib/lxd/:/bin/false
messagebus:x:107:111::/var/run/dbus:/bin/false
uuidd:x:108:112::/run/uuidd:/bin/false
dnsmasq:x:109:65534:dnsmasq,,,:/var/lib/misc:/bin/fals
max:x:1000:1000:max,,,:/home/max:/bin/bash
sshd:x:110:65534::/var/run/sshd:/usr/sbin/nologin
mysql:x:111:117:MySQL Server,,,:/nonexistent:/bin/fals
~
~
~
/etc/passwd
```

```
lsmax@ubuntuserver:/var/www$ ls -ld */
drwxr-xr-x 6 root root 4096 Nov 23 23:13 html/
max@ubuntuserver:/var/www$
```

Figure 4.2: Files showing apache username and web root file access.

Database security is maintained on the mySQL server using a unique user for the webapp, and by using parameterized queries in all requests to the database, as seen in this class file.

Usage of password encryption can be seen here and here.



```
#For TastyRecipes school project. Denies HTTP access to these specified folders.
<Directory /var/www/html/src/controller>
    Order allow,deny
    Deny from all
</Directory>
<Directory /var/www/html/src/model>
    Order allow,deny
    Deny from all
</Directory>
<Directory /var/www/html/src/integration>
    Order allow,deny
    Deny from all
</Directory>
<Directory /var/www/html/src/util>
    Order allow,deny
    Deny from all
</Directory>


/etc/apache2/apache2.conf

0:vim*
```

Figure 4.3: The apache2 config file restricting HTTP access to all layers except the view

4.3 Optional Task 2

Database requests are made using the PDO library and SQL queries. All requests are made using a new instance of the `DatabaseRequest` class. The easiest way to see how data is inserted and extracted is to see the code for that class, here.

5 Discussion

Implementing the MVC architecture from completely unstructured code was a long process and not very easy at first, the result is however a clear impementation of the MVC architecture. It helped tremendously to plan ahead using a class diagram. There are some requests that are passed down quite a long way, for example through the `Controller`, through a `RecipeSite` all the way to a `CommentSection`, but this is to reduce coupling on the `Controller` which now only has dependencies on two classes. All layers are well encapsulated, since all calls are made from the top and downwards. The `Util` class could have been left out but redirecting pages was so frequent in the view that I wanted to make a class for it. Since there were not many classes and files in this webapp I decided not to use a class autoloader. It makes it easier to keep track of what is actually imported, perhaps keeping a likeness to Java. Each import is required once instead.

The security implementations were straightforward following the instructions in the lecture notes, there is not much to discuss about that.

6 Comments About the Course

I spent more than 40 hours on this assignment. As predicted after the previous assignment, structuring the code was a lot of work, despite having done the Object Oriented Design course last semester. Implementing security was easy, especially using a linux web server.