Seminar 3, Improving the PHP Web Application

Internet Applications, ID1354

1 Goal

- Learn to use object-oriented design for the PHP server.
- Learn to use a PHP framework and/or the MVC pattern for the PHP server.
- Learn to mitigate some basic security threats in a web application.
- Learn to improve performance with caching and persistent connections.
- Learn to call a database from the PHP server.

2 Grading

The grading is as follows:

0 points The mandatory tasks are accepted and you have passed the seminar.

1 point The mandatory tasks and one higher grade task are accepted. You have passed the seminar and have also gained one point to improve the final course grade, see course plan for details on final grade.

2 points The mandatory tasks and both higher grade tasks are accepted. You have passed the seminar and have also gained two points to improve the final course grade, see course plan for details on final grade.

To pass the LAB1 sub course you must pass all four seminars. If you fail this seminar you have to report it again at the end of the course, at the fifth seminar. You can also report higher grade tasks at the fifth seminar.

3 Auto-Generated Code and Copying

All HTML, CSS and PHP code must be well designed and you must be able to explain and motivate every single part. You are *not* allowed to copy entire files or classes from the sample chat application, even if you understand it and/or change it.

However, you are allowed to write code very similar to the chat application. You are also allowed to copy HTML and CSS from any web site and to use any web development tool, you do not have to write HTML and CSS by hand. In particular, you are encouraged to get inspiration from (or use) free design templates.

4 Mandatory Tasks

Tasks one and two must be solved and reported at the seminar.

Task 1, MVC Architecture and Object-Oriented Design

You are free to choose whether to implement task 1a or task 1b. *Only one of those tasks shall be implemented.* The report must show that the specified requirements are met. To show this, you shall:

- Provide, and explain, a class diagram illustrating your application.
- Explain important parts of your code, and also include links to that code in your Git repository. Make sure your repository is public.

When not using a framework (task 1a), it easily happens to lower cohesion by placing tasks in the wrong layer. To avoid this, check that the following rules are followed. Similar mistakes can be made also when using a framework (task 1b), check that you follow all rules that apply even if you are using a framework.

- There shall not be anything related to HTTP or HTML outside the view. This means it shall be possible to change to a view that is not web-based, without having to change anything outside the view layer.
- Nothing that is displayed on the screen shall be generated outside the view layer. This means it is a mistake if the view displays a string that is created in another layer. As an example, if the user tries to register and the model finds that the username is taken, it is not the task of the model to return a string saying 'please try another username', which is then just displayed by the view.
- There shall not be anything related to databases outside the integration layer. This means it shall be possible to change between storing data in a text file or database, without having to change anything outside the integration layer.
- There shall not be any logic in the integration layer. Typically, integration contains only methods that insert, read, update or delete data in the data storage, without bothering about the meaning of the data. It is for example not appropriate to check that username and password match in the integration layer. That is logic and belongs in the model.

Task 1a, MVC Architecture Without Framework

Rewrite the Tasty Recipes web site to make it follow the MVC architectural pattern and basic object-oriented design concepts. The following requirements must be met.

- The application must be divided into the MVC layers. Add also other layers if required, for example a layer to interact with the datastore. All layers/namespaces, *including* `View`, must have the roles specified by the MVC pattern.
- All PHP code in `Controller`, `Model` and other layers/namespaces *except* `View` must be object-oriented. This means no code outside classes; no static methods unless there is a very good reason; a reasonable attempt to give classes high cohesion, low coupling and good encapsulation.
- Duplicated code must be avoided.

Task 1b, Use a PHP Framework

Use a PHP framework for the Tasty Recipes web site. You are free to choose whether to use the id1354-fw framework or some other framework. However, the framework you choose must make the web application follow the MVC architectural pattern. You must also follow basic object-oriented design concepts. The following requirements must be met.

- The framework must be used, as specified in the documentation, to split the application into layers. All layers/namespaces, *including* `View`, must have the roles specified by the MVC pattern.
- All PHP code in `Controller`, `Model` and other layers/namespaces, also *as much as possible* of `View` must be object-oriented. This means no code outside classes; no static methods unless there is a very good reason; a reasonable attempt to give classes high cohesion, low coupling and good encapsulation.
- Duplicated code must be avoided.

Task 2, Security

Make the Tasty Recipes web site more secure, by implementing *any three* of the following security sections covered at the lectures. You do not have to use HTTPS, but the report must show where HTTPS should be used.

- File System Security, To set the username of the Apache server might be complicated. It is allowed to skip this if facing problems.
- Input Filtering
- Database security, This may be chosen only if you solve optional task 2, *use a database*.

- Password Encryption
- Cross Site Scripting
- Impersonation

The report must meet the following requirements.

- It must be clear that all content in the three chosen security sections has been considered.
- Explain important parts of your code, and also include links to that code in your Git repository. Make sure your repository is public. Also, where appropriate, include links to configuration files, which you shall also upload to the Git repository.

5 Optional Tasks

Optional Task 1, Improve Performance

Implement performance improvements as described in the sections *Caching* and *Persistent Connections* on lecture nine.

- The report must show that all content in the two sections has been considered.
- The report must explain important parts of your code, and also include links to that code in your Git repository. Make sure your repository is public. Also, where appropriate, include links to configuration files, which you shall also upload to the Git repository.

Optional Task 2, Use a Database

Use a database to store comments and user data persistently on the server. All data must be in the database, do not store any data in plain files. There are no requirements on database design.

- The report must show how data is written to and read from the database.
- The report must include important parts of your code, and also include links to that code in your Git repository. Make sure your repository is public. Also, where appropriate, include links to configuration files, which you shall also upload to the Git repository.