# Homework: MVC Concepts Basics

This document defines the homework assignments from the ["Web Development Basics" Course @ Software University](https://softuni.bg/courses/web-development-basics/). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems.

## Self-Made MVC Framework for Web Applications

Design and Implement a **Web application MVC framework** with **Apache, PHP and MySQL**. Your framework should support as minimum the following functionality:

* Separate **views** from **controllers** and data **models**.
  + Use separate folders, files and classes.
  + **Views** should hold your HTML rendering logic.
  + **Controllers** should hold the presentation logic and the business logic (optionally).
  + **Models** should implement your data access logic (CRUD operations and queries).
  + You are allowed to use **procedural style** or **object-oriented style** for structuring your framework.
* Implement a **front-controller** to serve as entry point for all requests.
  + Create .htaccess configuration file to redirect all HTTP requests to the front-controller.
  + The front-controller should redirect the requests to the requested **controller** and **action**.
  + Use the classical **URL format** for your requests: [http://mysite/**controller**/**action**/**parameters**](http://mysite/controller/action/parameters).
  + Use **default** controller and action when no controller or no action is specified.
* Implement a **configuration file** to keep the application constants and settings.
  + Keep in it the database connection settings, the default controller, action and layout, etc.
* Implement a simple **layout system** (e.g. reusable header and footer).
* Implement a BaseController class and derive your controllers from it.
  + Implement a method to **render a view**. It should render the layout + the view inside it. By default, the view name may be the same as the action name.
  + Implement a functionality for **redirecting** to another controller / URL.
* Keep your **views** in separate folder.
  + Recommended structure for the views: views/{controller-name}/{view-name}.php.
* Implement a simple **data access layer**.
  + It may have **base class** holding the database connection + one class per database entity.
  + The data access layer should provide CRUD operations over the database entities + queries.
* Implement **access control** functionality.
  + Use the **user session** to keep the currently logged-in user.
  + Implement UserController with register, login and logout actions and corresponding views.
  + Keep the passwords encrypted by password\_hash() PHP function.
  + Implement methods for checking the current user in the BaseContoller, e.g. isLoggedIn(), authorize(), …

## TODO List

Using your **MVC Framework** Implement a simple **TODO list** in PHP. It should support user **registration**, user **login**, **list** TODO items, **add** TODO item, **delete** TODO item and **logout**. Follow the steps below to simplify your work:

* Design a MySQL database to keep the user accounts and TODO items:
  + Create table users(id, username, passwordHash) to keep the registered users.
    - Make the column "username" unique (disallow duplicates).
  + Create table todos(id, user\_id, todo\_item) to keep the users' TODO items.
* Create a data access functionality (MVC model) to hold your data access logic:
  + createUser(username, password)
  + isUserValid(username, password)
  + getTodoItems(user\_id)
  + addTodoItem(user\_id, todo\_text)
  + deleteTodoItem(user\_id, todo\_id)
* You may already have part of the above functionality from your MVC framework, so do not repeat it.
* Create a controller UsersController and action register(). It should map to URL /users/register.
  + It should **register a new user** in the database (by username and password).
  + In case of **GET** request, render the register view to display the user registration form.
  + In case of **POST** request, register the user in the database.
    - In case of success, redirect to /todos and show info notification "User registered".
    - Show an error message in case of problem with the user registration and render the user registration form again.
  + Keep the users and their password hashes in the database.
  + Use the [password\_hash()](http://php.net/password_hash) PHP function to hash the users' passwords.
  + Check for duplicated usernames when creating a new user.
* Create an action login() in the UsersController. It should map to URL /users/login.
  + It should login a user by username and password.
  + After successful login, save the username and user\_id in $\_SESSION and redirect to /todos.
  + In case of error, display an error notification message and render the login form again.
* Create a controller TodosController. It should map to URL /todos.
  + This controller should implement the TODO list functionality for logged-in users.
* Implement "**List TODOs**" functionality (action index). It should map to URL /todos.
  + First, check the logged-in user in $\_SESSION. Redirect to /users/login if no user is logged in.
    - To avoid repeating logic, this check + redirect could be implemented in a method authorize() in the BaseController.
  + Display all TODO items of the current user.
    - Using your MVC models, execute a MySQL **SELECT** to take the items from database.
    - In your view iterate over the items and render them as HTML list (<li>*item*</li>).
    - Right after each TODO item, display a **delete link**: /todos/delete/*<id>*.
* Implement **"Add TODO"** functionality. It should map to URL /todos/add.
  + If the action is **GET**, render the add view to show the "Add TODO" HTML form.
    - The form should **POST** its data (TODO text) to /todos/add action.
  + If the action is **POST**, save the new TODO in the database.
    - First, check the logged-in user. Redirect to /users/login if no user is logged in.
    - Get the current user\_id from the session. Get the todo\_text from the request POST parameters. Invoke addTodoItem(user\_id, todo\_text) from your data access layer.
    - Redirect the redirect to /todos.
    - In case of error, display an error notification message and redirect to /todos.
* Create an action delete to delete a TODO item by id. It should map to URL /todos/delete/*<id>*.
  + First, check the logged-in user. Redirect to /users/login if no user is logged in.
  + Get the current user\_id from the session. Get the TODO id from the request URI parameters. Invoke deleteTodoItem(user\_id, todo\_id) from your data access layer.
  + Redirect the browser to /todos.
  + In case of error, display an error notification message and redirect to /todos.
* Create an action logout in the UsersController. It should map to URL /users/logout.
  + The logout action should to logout the current user and redirect to /.