# Homework: Working with Forms in PHP

# Forms in PHP, OOP and MVC

This document defines the homework assignments for ["PHP Web Development Basic" Course @ Software University](https://softuni.bg/trainings/1746/php-web-developmentbasics-september-2017). Please submit ……………………………………………….???

|  |
| --- |
| **Download** the initial code **13.Working-with-forms-in-PHP-Initialcode.zip.**  **Download** the initial database SQL dump file **13. Working-with-forms-in-PHP -Initialsqldump.zip.**  **Import the** SQL into your MySQL . |

# Employee Application for the Web

In this exercise we will write tasks for an Employee application in a large organization. We will use the MVC principle again but our main task is to **handle form data with PHP**, sanitize it and validate it. As of previous exercises you are given the main structure of the code and a database with already filled in information.

# Structure of Code

|  |  |
| --- | --- |
| **Folder** | **Description** |
| \core | Your core classes and core methods. Two abstract core classes are **Controller** and **Model** |
| \controller | Your control functionality distributed in many classes: MainController, EmployeeController and etc. |
| \model | Your DB actions distributed in many classes like: EmployeesModel, AddressesModel and etc. |
| \view | HTML, CSS, Javascript. HTML forms, tables, menus and etc. Use subfolders to hold a hierarchy of a templates. |

This time you will use **multiple controllers** and **multiple models**. The framework we will create will have a simple routing system in which as **$\_GET** parameters are passed the **controller** and the .

# Problem 1. Create a Routing System

Let's begin by creating core functionality to the MVC framework. What we want is functionality that will take the data from the browser trough the GET HTTP method. It should look like this:

|  |
| --- |
| http://localhost/employee2/?controller=**controllerName**&action=**actionName** |

This way we **avoid putting everything in a method called main()** and we **dynamically create objects** for the functionality we need. This way **our coded is distributed** and only what is needed is loaded so it is **fast**.

# Constraints

You must validate the input from **$\_GET**. The controller name and action name can contain only upper or lower characters and no numbers. The underscore (\_) is also disallowed.

|  |  |
| --- | --- |
| **Valid** | **Not Valid** |
| controller=employee&action=addNew | controller=employee23&action=add\_new  Page not found error |

You can use the function matching a regular expression preg\_match(): <http://php.net/manual/en/function.preg-match.php>

# Dynamic Load of Methods

The index.php must load dynamically the exact method in the OOP hierarchy. If action is empty or not given at all then Controller must load the **main()** method which should always be implemented!

If the controller name is not given in the input URL then the MainController should be called!

|  |  |
| --- | --- |
| **Input** | **Call** |
| controller=employee&action=addNew | EmployeeController->addNew(); |
| controller=employee&action= | EmployeeController->main(); |
| controller=employee | EmployeeController->main(); |
|  | MainController->main(); |

Create a **"Page not found 404"** for all other cases.

# Hint

Your code in index.php can look like the screenshot on the left.

For now our code is put in index.php but further we can put it somewhere else. It can't be in the Controller class because it is abstract and needs a class to extend it.

There is a **bug in this code** as it will not display any "Page not found" page but reload to the default controller and action! **Improve the code!**

# Create a Method to Load HTML

Extend the functionality of the Controller class with a method loadView($filename) which will include the proper HTML file if it exists and throw an exception if it doesn't.

# Problem 2. View Employees

Create a **new EmployeeController class** that **extends the abstract Controller class** and **code a method which shows an HTML table of all employees**. Choose a proper name for your method. Do not show employee address data, but show department data. Create proper method in the EmployeeModel class.

**Create a menu** in the header of your HTML and put a link to the new functionality there.

# Problem 3. Add Project to Employee

Add a column to the employee table (problem 2) and call it " Actions ". Put a link in every employee row, called "Add project" which will lead you to an **HTML form** that should be submitted trough **POST**.

Ask the user to fill the following fields: Name, Description, Start date, End\_date. Name and Description should be validated as strings. Start date and end date are in format YYYY-MM-DD.

# Step 1

Take the data with PHP from the browser by submitting it to the proper controller/action. You should create a new method in EmployeeController to do this. You can name it addProject(). Make sure the data is valid.

# Step 2

Create a new model for projects and name it ProjectsModel class. Introduce a new method create() in it that takes the data from addProject() and inserts it into the database.

# Problem 4. View Projects Per Employee

In the Actions column in the list of employees (problem 2) add a new link called "Projects" which would load EmployeeController / viewProjects. What data do you have to pass to viewProjects() to view the projects of a particular employee?

# Step 1

Because the method is dynamically loaded you can get the passed arguments with the built-in func\_get\_args().

Code the viewProjects() method and connect it to a proper method in the proper model.

# Step 2

You should already have a **ProjectsModel class**. You can name the method read(). What do you have to pass to the model to get the proper data from the database?

# Step 3

Create an HTML table to display the output from read(). Put the HTML in a proper HTML file.

# Problem 5. View Employee Address

In the Actions column in the list of employees (problem 2) add a **new link called "Address"** which would load AddressController / viewAddresses in a **new tab**. What data do you have to pass to viewAddresses() to view all addresses of a particular employee? Create the proper method in the model. Use the same steps as in problem 4 to achieve the result.

Compose the proper URL and use: <a href="?controller=…&action=…" target="\_blank">Address</a>

# Problem 6. Update Employee Address\*

There should be a possibility to edit address data. In the output of problem 5 add a column called ' Actions ' and in every row add a link called ' Edit '. Create a proper **controller and method** and a proper **model and method** to be able to edit an address of an employee. What data would you need to edit an address?

# Hint

You will need to pass to the model that edits the data at least one ID. You will need to update information also for the town.

# Constraints

Study now the structure of the addresses and towns tables. Validate all input data as strings. Use filter\_var to do that.

# Problem 7. Develop the Controller class

Until now your routing functionality was in index.php. Encapsulate it in the abstract Controller class in a separate method called route().

Let MyController extend the abstract Controller class. Now change index.php to contain an instance of MyController and start the route() method which will dynamically load the proper controller / action combination.

# Problem 8. Methods for Input Data

Develop Further the Controller class. Create three methods with the following names: inputGet(), inputPost(), inputRequest() which take the submitted data from the browser using $\_GET, $\_POST and $\_REQUEST.

# Step 1. Sanitize All Input against Scripts and Tags

No strings like **<script>**, <p> or other should ever pass through the input by any means.

You can use sanitize filters to do that: <http://php.net/manual/en/filter.filters.sanitize.php>

To filter a whole array you can use the built-in function filter\_var\_array() or other filter functions: <http://php.net/manual/en/ref.filter.php>

# Step 2. Limit the Data Types

Add two arrays ($fields, $types) to every method like this inputGet($fields, $types).

If $fields is not empty it should contain the possible names of input data. Everything else should be deleted from the input.

|  |  |
| --- | --- |
| **Fields Example** | **Types example** |
| $fields = ['name', 'email', 'address\_id', 'some\_name']; | $  types = ['string', 'email', 'number\_int', 'not\_checked'] |

See what sanitization filters you need in your app and add them to the table of possible filters. If the type is 'not\_checked' then do not do filtering.

# Step 3. Change Your Previous Code

Now go back to what you have already coded and use inputGet() and inputPost() to handle data in the controllers instead of $\_GET and $\_POST.

# Problem 9. AJAX Version of Add Project to Employee\*\*\*

If you are familiar with JavaScript and jQuery rewrite Problem 3 to dynamically add projects for a particular employee and dynamically update the list of projects.