IMPLEMENTING MVC USING PHP7, HTML5 AND CSS3

BARASA MICHAEL MURUNGA

michael.barasa@strathmore.edu

STRATHMORE UNIVERSITY

TABLE OF CONTENTS

Implementing MVC using PHP, HTML and CSS

Introduction

MVC (Model-View-Controller) is a design pattern that separates an application into three components: **the Model, the View,** and **the Controller**

- **The Model**: Represents the data and business logic of our application. In PHP, you can create a class that contains all the functions that interact with the database or handle business logic.
- **The View**: Represents the presentation logic of the application. In HTML, one can create a class that displays the data from the Model and use CSS to style the View.
- **The Controller**: Acts as the mediator between the Model and the View. In PHP, you can create a class that receives requests from the user and sends them to the appropriate Model function. The Controller then receives data from the Model and sends it to the appropriate View.

Sample Implementation

1. Create a PHP file for the Model class. This file should contain all the functions that interact with the database or handle business logic.
2. Create a HTML file for the View class. This file should contain all user interface that displays the data from the Model.
3. Create a CSS file to style the View.
4. Create a PHP file for the Controller class. This file should receive requests from the user via the View class and send them to the appropriate Model function. The Controller then receives the data from the Model and sends it to the appropriate View.
5. Create config.py file that contains any configuration settings that are needed for the application, such as database credentials and API keys.

Sample File Structure

```
.
└── app
    ├── config.php
```

```
├── controller
│   ├── UserController.php
│   ├── login.php
│   ├── logout.php
├── model
│   └── UserModel.php
└── view
    ├── add.php
    ├── dashboard.php
    ├── index.php
    ├── login.php
    └── style.css
```

Listing 1-1.app/model/UserModel.php: Handle all database queries for user management system.

```
<?php
class UserModel
{
        private $pdo;

        function __construct($pdo)
        {
                $this->pdo = $pdo;
        }

        function getUsers()
        {
                $stmt = $this->pdo->query('SELECT * FROM users');

                return $stmt->fetchAll(PDO::FETCH_ASSOC);
        }

        function addUser($name, $email)
        {
                $stmt = $this->pdo->prepare(
```

```
                    'INSERT INTO users (name, email) VALUES (:name, :email)');
                $stmt->execute(['name'=>$name, 'email'=>$email]);

                return $this->pdo->lastInsertId();
        }

        function deleteUser($id)
        {
                $stmt = $this->pdo->prepare('DELETE FROM users WHERE id = :id');
                $stmt->execute(['id'=>$id]);
        }
}
?>
```

This code defines a class called **UserModel** in PHP, which interacts with a database to perform basic CRUD (create, read, update, delete) operations on a table called "users".

The constructor takes a PDO object as an argument and assigns it to the private property **$pdo**. The PDO object is a PHP extension that provides a lightweight interface for interacting with databases.

The **getUsers()** method fetches all rows from the "users" table using a SELECT query and returns them as an array of associative arrays, where each subarray represents a single row with column names as keys and column values as values.

The **addUser()** method inserts a new row into the "users" table using an INSERT query with named parameters to prevent SQL injection attacks. It takes two arguments- the name and email of the user to be added- and returns the auto-incremented ID of the newly added row using the lastInsertId() method of the PDO object.

The **deleteUser()** method deletes a row from the "users" table with the specified ID using a DELETE query with a named parameter. It takes a single argument- the ID of the user to be deleted- and does not return anything.

Listing 1-2.app/view/index.php: Display a list of all users and provide links to add or delete users.

```
<!DOCTYPE html>
```

```
<html>
	<head>
		<title>User Management System</title>
		<link rel = "stylesheet" type = "text/css" href = "style.css">
	</head>
	<body>
		<h1>User Management System</h1>
		<table>
			<thead>
				<tr>
					<th>ID</th>
					<th>Name</th>
					<th>Email</th>
					<th>Action</th>
				</tr>
			</thead>
			<tbody>
				<?php foreach ($users as $user): ?>
				<tr>
					<td><?php echo $user['id']></td>
					<td><?php echo $user['name']></td>
					<td><?php echo $user['email']></td>
					<td>
						<a href = "index.php? action =
delete&id=<?php echo $user['id'];?>">
							Delete
						</a>
					</td>
				</tr>
				<?php endforeach; ?>
			</tbody>
		</table>
		<p>
		<a href = "index.php?action=add">
			Add User
		</a>
		</p>
	</body>
</html>
```

This code generates a web page for a user management system. The page includes a table that displays a list of users and allows the user to delete an existing user or add a new user.

In the HTML code, the **<!DOCTYPE html> declaration** specifies the document type, and the <html> element is the root element of the document. The <head> element contains metadata such as the title of the page and a link to an external stylesheet. The <body> element contains the visible content of the web page.

The **<h1> element** displays the title of the page, "User Management System". The <table> element creates a table that contains the user information, including the ID, name, email, and a "Delete" button for each user. The table is constructed using the HTML <thead> and <tbody> elements.

The PHP code is embedded within the HTML using the **<?php ?> tags**. The foreach loop iterates over an array of users and generates a new row in the table for each user. The echo statements display the user's ID, name, and email in their respective table cells.

The "Delete" button in the last column of each row is an HTML <a> element that includes a link to the same page with a query string that specifies the user ID to be deleted. When the link is clicked, the user is redirected to the same page with the action parameter set to "delete" and the id parameter set to the ID of the user to be deleted.

Finally, the "Add User" link at the bottom of the page is an HTML <a> element that provides a way for the user to add a new user to the system. When the link is clicked, the user is redirected to the same page with the action parameter set to "add".

Listing 1-3.app/view/add.php: Display a form that allows users to add a new user.

```
<!DOCTYPE html>
<html>
	<head>
		<title>User Management System</title>
		<link rel = "stylesheet" type = "text/css" style = "style.css">
	</head>
	<body>
		<h1>Add User</h1>
		<form method = "post">
			<label for = "name">Name:</label>
			<input type = "text" name = "name" required>
			<br>
```

```
                    <label for = "name">Email:</label>
                    <input type = "text" name = "name" required>
                    <br>
                    <button type = "submit">Submit</button>
                </form>
        </body>
</html>
```

This code is creating an HTML form for adding a new user to the user management system. The form uses the HTTP POST method to submit the data to the server.

The PHP code within the HTML is used to generate the form and specify the required input fields. The **"name" attribute** of the input fields is used to identify the input when it is submitted to the server.

The PHP code is generating a label for each input field using the HTML "label" tag. The "for" attribute of the "label" tag is set to match the "name" attribute of the corresponding input field, so that clicking on the label will focus the input field.

Finally, a **"Submit" button** is added to the form to allow the user to submit the form data. When the user clicks the "Submit" button, the form data will be submitted to the server using the HTTP POST method.

Listing 1-4.app/view/style.css: Apply styling to the view.

```
table
{
        border-collapse:collapse;
        width:100%;
}

th, td
{
        border:1px solid #ddd;
        padding:8px;
        text-align:left;
}

th
```

```
{
        background-color:#f2f2f2;
}

a
{
        color:blue;
        text-decoration:underline;
}
```

This is a block of CSS code defining the style for a table and its contents.

table: This selector applies the styles to all tables in the HTML document. The border-collapse property is used to collapse the table borders together so that they share a single border. The width property sets the table width to 100%.

th, td: These selectors apply the styles to all table headers (th) and table data cells (td). The border property sets a 1px solid border around the cells with a light gray color (#ddd). The padding property adds space between the cell content and the cell border. The text-align property sets the alignment of the cell content to the left.

th: This selector applies additional styles to table headers (th). The background-color property sets the background color of the header row to a light gray (#f2f2f2).

a: This selector applies styles to all hyperlinks (a) in the HTML document. The color property sets the link color to blue. The text-decoration property adds an underline to the link text.

Listing 1-5.app/view/login.php: Enable users to login into the system.

```
<!DOCTYPE html>
<html>
        <head>
                <title>User Management System</title>
                <link rel = "stylesheet" type = "text/css" href = "style.css">
        </head>
        <body>
                <h1>Login</h1>
                <form method = "post">
                        <label for = "email">Email:</label>
```

```
                        <input type = "email" name = "email" required>
                        <br>
                        <label for "password">Password</label>
                        <input type = "password" name = "password" required>
                        <br>
                        <button type = "submit">Login</button>
                </form>
        </body>
</html>
```

Listing 1-6.app/controller/login.php

```
<?php
session_start();

if (isset($_POST['email']) && ($_POST['password']))
{
        $email = $_POST['email'];
        $password = $_POST['password'];

        // connect to the database
        $db = new PDO('mysql:host = hostname', dbname = 'mydatabase', 'username',
                'password');

        // query the database for the user with the given email
        $stmt = $db->prepare(
                'SELECT id, password FROM users WHERE email = :email');
        $stmt->execute(array(':email=>$email'));
        $user = $stmt->fetch(PDO::FETCH_ASSOC);

        // Verify the users password
        if ($user && password_verify($password, $user['password']))
        {
                $_SESSION['user_id'] = $user['id'];
                header('Location:index.php');
                exit;
        }
        else
        {
                $error = "Invalid email or password";
        }
}
include ('../view/login.php');
```

```
?>
```

session_start() starts a new or resumes an existing session.

if (isset($_POST['email']) && ($_POST['password'])) checks whether the email and password inputs are set in the POST request.

$email = $_POST['email']; and $password = $_POST['password']; store the email and password inputs in variables.

$db = new PDO('mysql:host = hostname', dbname = 'mydatabase', 'username', 'password'); connects to the MySQL database using PDO (PHP Data Objects).

$stmt = $db->prepare('SELECT id, password FROM users WHERE email = :email'); prepares a statement to select the user with the given email from the users table.

$stmt->execute(array(':email=>$email')); executes the prepared statement with the email input as a parameter.

$user = $stmt->fetch(PDO::FETCH_ASSOC); fetches the first row from the result set as an associative array and stores it in the $user variable.

if ($user && password_verify($password, $user['password'])) checks whether a user was found and whether the input password matches the hashed password in the database.

password_verify() is a PHP function used to verify a password hash against a given password.

$_SESSION['user_id'] = $user['id']; sets the user_id session variable to the ID of the authenticated user.

header('Location:index.php'); redirects the user to the index page.

else {$error = "Invalid email or password";} sets an error message if the login was unsuccessful.

include ('../view/login.php'); includes the login page HTML, along with any error message that was set.

Listing 1-7.app/view/dashboard.php: Logged in user.

```
<!DOCTYPE html>
<html>
        <head>
                <title>User Profile</title>
                <link rel = "stylesheet" type = "text/css" href = "style.css">
        </head>
        <body>
```

```
                <h1>User Profile</h1>
                <?php if (isset($_SESSION['user_id'])):?>
                <p>
                Welcome, User!
                <a href = "controller/logout.php">
                        Logout
                </a>
                </p>
                <ul>
                        <?php foreach ($users as $user):?>
                        <li>
                                <?php echo $user['name'];?>
                        </li>
                        <?php endforeach; ?>
                </ul>
                <?php else: ?>
                <p>
                You must be logged in to view this page.
                <a href = "controller/login.php">
                        Login
                </a>
                </p>
                <?php endif; ?>
        </body>
</html>
```

This code shows a user profile page that displays the user's name and allows them to logout if they are logged in. If the user is not logged in, it displays a message and a link to the login page.

The code starts with an HTML structure, including a title and a link to a CSS file. The body of the page contains a heading, and then it checks if the user is logged in by checking if the session variable user_id is set using the PHP isset() function.

If the user is logged in, it displays a welcome message along with a logout link. It also displays an unordered list of user names using a PHP foreach loop to iterate over an array of users.

If the user is not logged in, it displays a message indicating that the user needs to be logged in to view the page, along with a link to the login page.

Listing 1-8.app/controller/logout.php: Destroy session and redirect user to login page.

```php
<?php
session_start();
session_destroy();
header('Location:login.php');
exit;
?>
```

This is a PHP script that performs the following tasks:

1. Starts a new or resumes an existing session by calling **session_start()** function.
2. Destroys all data associated with the current session by calling **session_destroy()** function.
3. Redirects the user to the login page by sending an HTTP redirect header with the Location field set to login.php using the **header()** function.
4. Exits the script immediately using the **exit function** to ensure that no further code is executed after the redirect.