Learning MVC: Concepts, Features, and PHP mini-Project

1. Introduction to MVC

Model-View-Controller (MVC) is an architectural design pattern that separates an application into three interconnected components:

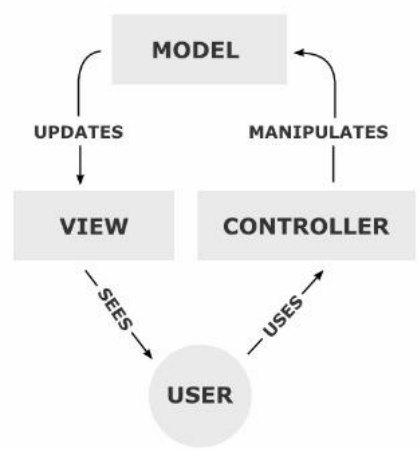
- **Model** : Handles data and business logic
- **View** : Presents data to the user (UI)
- **Controller** : Processes user input and coordinates model and view

☠ **Real-world analogy**: Think of MVC like a restaurant:

- **Model** = The kitchen (prepares food, follows recipes)
- **View** = The dining area (what customers see)
- **Controller** = The waitstaff (takes orders, delivers food, manages interactions)

---

2. MVC Architecture



☠ The diagram illustrates how these components interact:

1. The **User** interacts with the **Controller** (sends requests)
2. The **Controller** manipulates the **Model** (processes data)
3. The **Model** updates the **View** (prepares display)
4. The **View** presents information to the **User** (shows results)

---

3. Core Components Explained

☠ **Model**

- **Purpose**: Data management and business logic

- **Responsibilities**:
    - Stores and manages application data
    - Implements business rules and validation
    - Communicates with databases
    - Notifies the view of data changes

💣 **Important note**: Models don't know how their data will be displayed

☠️ **View**

- **Purpose**: User interface and presentation
- **Responsibilities**:
    - Renders data for the user
    - Implements visual elements (HTML, CSS)
    - Receives updates from the model
    - Sends user actions to the controller

💣 **Important note**: Views shouldn't contain business logic

☠️ **Controller**

- **Purpose**: Coordination and request handling
- **Responsibilities**:
    - Receives input from users
    - Processes requests
    - Interacts with the model to fetch/update data
    - Selects which view to display

💣 **Important note**: Controllers connect models and views

---

4. Advantages of MVC

- **Separation of concerns**: Each component has a distinct responsibility
- **Parallel development**: Teams can work simultaneously on different components
- **Code reusability**: Models and controllers can be reused across views
- **Easier maintenance**: Changes in one component don't necessarily affect others
- **Better organization**: Clear structure makes the codebase more navigable

- **Easier testing**: Components can be tested independently

---

5. Disadvantages of MVC

- **Learning curve**: More complex than simple architectures
- **Overhead for small projects**: Can be excessive for basic applications
- **Strict conventions**: Requires adherence to specific patterns
- **Navigation complexity**: Following the flow can be challenging for beginners
- **Increased initial development time**: Setting up the architecture takes time

---

6. Popular MVC Frameworks in PHP

Framework	Key Features	Best For
Laravel	Elegant syntax, robust ecosystem	Full-stack web applications
CodeIgniter	Lightweight, minimal configuration	Simpler projects with small footprint
Symfony	Component-based, enterprise-ready	Complex, large-scale applications
CakePHP	Convention over configuration, built-in tools	Rapid development
Yii	High performance, security-focused	Modern web applications
Zend	Enterprise-ready, modular	Enterprise applications

---

7. MVC In Practice: PHP Mini-Project Concept

Student Management System

Project Structure:

```
student_management/
├── index.php                    # Front controller and minimal router
├── indexWithSafetyChecks.php    # Same router with friendly safeguards
├── setup.sql                    # DB schema and optional seed data
├── styles.css                   # Minimal styling for the views
├── config/
│   └── database.php             # PDO connection (ERRMODE_EXCEPTION enabled)
├── controllers/
│   └── StudentController.php    # Orchestrates requests and selects views
├── models/
│   └── Student.php              # Data access layer (CRUD via PDO)
└── views/
    └── students/
        ├── list.php             # List all students
        ├── add.php              # Create form
        ├── edit.php             # Update form
        ├── view.php             # Details page
        └── delete.php           # Delete confirmation
```

student_management\config\database.php

```php
<?php
// Super simple PDO connection for beginners

$host = 'localhost';
$db   = 'student_management';
$user = 'root';
$pass = '';
$charset = 'utf8mb4';
/*
$charset = 'utf8mb4';
- Supports all Unicode characters, including emojis, symbols, and multilingual text.
- Prevents "incorrect string value" errors when storing characters outside the Basic Multilingual Plane.
- Always prefer utf8mb4 for new projects to ensure full compatibility and future-proofing.
*/

try {
    // Create the PDO instance (this will be used across the app)
    $database = new PDO("mysql:host=$host;dbname=$db;charset=$charset", $user, $pass);
    // Show errors as exceptions (helpful while learning)
    $database->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die('Database connection failed: ' . $e->getMessage());
}
```

student_management\controllers\StudentController.php

```php
<?php
class StudentController
{
    private $model;

    // Constructor to initialize the model
    public function __construct($database) // __construct is a special method that is automatically called when an object of the class is created. It is typically used to initialize properties or perform setup tasks.
    {
        $this->model = new Student($database);  // this used to initialize the model property with a new instance of the Student model
    }

    // List all students
    public function listAction()
    {
        // Get all students from model; getAllStudents() method called from model
        $students = $this->model->getAllStudents();
        // Loads the view file responsible for displaying the student list.
        include 'views/students/list.php';
        // The `include` statement imports and executes the specified file at runtime.
        // If the file is missing, PHP will show a warning but continue execution.
    }

    // View a specific student
    public function viewAction($id)
    {
        // Get specific student; getStudentById($id) method called from model
        $student = $this->model->getStudentById($id);
        // Loads the view file responsible for displaying the student details.
        include 'views/students/view.php';
    }

    // Add a new student
    public function addAction($unused = null) // $unused = null to avoid error if called with an id means it can be called with or without an argument
    {
        $error = null;
        // If form submitted, try to insert
        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
            $name = trim($_POST['name'] ?? ''); // The null coalescing operator (??) is used to check if a variable is set and not null. If it is not set or is null, it returns the value on its right side.
            $email = trim($_POST['email'] ?? '');

            if ($name === '' || $email === '') {
                $error = 'Name and Email are required.';
            } elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
                $error = 'Please enter a valid email address.';
            } else {
                try {
                    // Attempt to add the student; addStudent() method called from model
                    $ok = $this->model->addStudent(['name' => $name, 'email' => $email]); // $ok will be true if insertion was successful
                    if ($ok) {
                        // Redirect back to list
                        header('Location: index.php?controller=student&action=list'); // if the addition is successful, the user is redirected to the student list page
                        // ?controller=student&action=list is a query string used to pass parameters to the server
                        // It tells the application to use the StudentController and call the listAction method.
```

```
                    exit; // exit statement is used to stop further script execution.
                } else {
                    $error = 'Failed to add student.';
                }
            } catch (Exception $e) {
                $error = 'Error: ' . $e->getMessage();
            }
        }
    }

    // After processing the form submission (success or failure),
    // load the "add student" form view again.
    // - If adding was successful → the user is redirected before this point, so this won't run.
    // - If validation failed or no form was submitted yet → the form is shown again,
    //   with any error messages (from $error) passed to the view.
    include 'views/students/add.php';
}

// Edit an existing student
public function editAction($id)
{
    if (!$id) {
        header('Location: index.php?controller=student&action=list');
        exit;
    }

    $error = null;
    $student = $this->model->getStudentById($id);

    // Check if student exists
    if (!$student) {
        $error = 'Student not found.';
    }

    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $name = trim($_POST['name'] ?? '');
        $email = trim($_POST['email'] ?? '');

        if ($name === '' || $email === '') {
            $error = 'Name and Email are required.';
        } elseif (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $error = 'Please enter a valid email address.';
        } else {
            try {
                $ok = $this->model->updateStudent($id, ['name' => $name, 'email' => $email]); // $ok will be true if update was
successful
                if ($ok) {
                    header('Location: index.php?controller=student&action=list'); // if the update is successful, the user is
redirected to the student list page
                    exit; // exit from the editAction method to prevent further code execution
                } else {
                    $error = 'Failed to update student.';
                }
            } catch (Exception $e) {
                $error = 'Error: ' . $e->getMessage();
            }
        }
        // Refresh student data with previous values on validation failure.
        $student = ['id' => $id, 'name' => $name, 'email' => $email];
    }

    include 'views/students/edit.php';
}

// Delete a student
public function deleteAction($id)
{
    if (!$id) {
        header('Location: index.php?controller=student&action=list'); // if no id provided, redirect to list
        exit;
    }

    $error = null;
    $student = $this->model->getStudentById($id); // Fetch student to confirm deletion

    if (!$student) {
        $error = 'Student not found.';
    }

    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        try {
            $ok = $this->model->deleteStudent($id); // Attempt to delete the student; deleteStudent() method called from model
            if ($ok) {
                header('Location: index.php?controller=student&action=list'); // if deletion successful, redirect to list
                exit;
            } else {
                $error = 'Failed to delete student.';
            }
        } catch (Exception $e) {
            $error = 'Error: ' . $e->getMessage();
```

```
            }
        }

        include 'views/students/delete.php'; // Show delete confirmation form
    }
}
```

student_management\models\Student.php

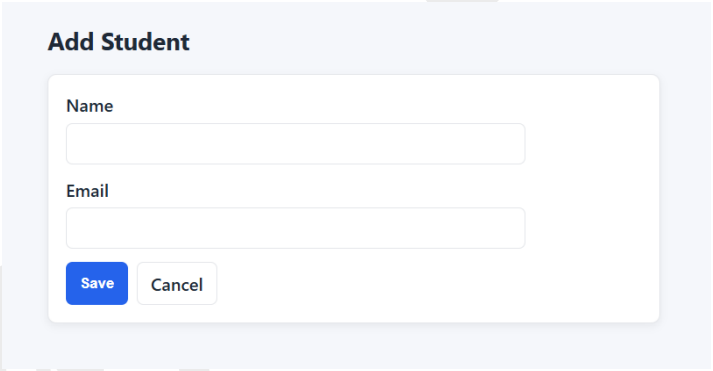
```
<?php
class Student
{
    private $db; // private used to restrict access to the property within the class only

    public function __construct($database) // __construct is a special method that is automatically called when an object of the class is created. It is typically used to initialize properties or perform setup tasks.
    {
        $this->db = $database;
    }

    // view all students
    public function getAllStudents()
    {
        // Query database and return all students
        $stmt = $this->db->query('SELECT id, name, email FROM students ORDER BY id ASC');
        // $this refers to the current object instance
        // ASC means ascending order
        return $stmt->fetchAll();
        // fetchAll() used to fetch all results from a PDO(PHP Data Objects) statement as an array
    }

    // view one student by id
    public function getStudentById($id)
    {
        // Fetch specific student data
        $stmt = $this->db->prepare('SELECT id, name, email FROM students WHERE id = :id'); // : is a named placeholder in a SQL query, used in prepared statements to bind values securely and prevent SQL injection attacks.
        // prepare() is used to prepare an SQL statement for execution
        $stmt->execute([':id' => $id]); // execute() is used to execute a prepared statement
        return $stmt->fetch(); // returning a single record based on id
    }

    // add a new student
    public function addStudent($data)
    {
        // Insert new student record (very basic example)
        $stmt = $this->db->prepare('INSERT INTO students (name, email) VALUES (:name, :email)');
        return $stmt->execute([
            ':name' => $data['name'] ?? '', // The null coalescing operator (??) is used to check if a variable is set and not null. If it is not set or is null, it returns the value on its right side.
            ':email' => $data['email'] ?? '',
        ]);
    }

    // update an existing student by id
    public function updateStudent($id, $data)
    {
        // Update an existing student record
        $stmt = $this->db->prepare('UPDATE students SET name = :name, email = :email WHERE id = :id');
        return $stmt->execute([
            ':name' => $data['name'] ?? '', // : is a named placeholder in a SQL query, used in prepared statements to bind values securely and prevent SQL injection attacks.
            ':email' => $data['email'] ?? '',
            ':id' => $id,
        ]);
    }

    // delete a student by id
    public function deleteStudent($id)
    {
        // Delete a student by id
        $stmt = $this->db->prepare('DELETE FROM students WHERE id = :id');
        return $stmt->execute([':id' => $id]);
    }
}
```

student_management\views\students\add.php

```
<!doctype html>
<html>

<head>
    <meta charset="utf-8">
    <title>Add Student</title>
```

```
    <link rel="stylesheet" href="/student_management/styles.css">
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <!-- this meta tag ensures proper rendering and touch zooming on mobile devices -->

</head>

<body>
    <div class="container">
        <h1>Add Student</h1>

        <?php if (!empty($error)): ?>
            <div class="message-error"><?php echo htmlspecialchars($error, ENT_QUOTES, 'UTF-8'); ?></div>
            <!--
            htmlspecialchars is used to prevent XSS attacks by escaping special characters.
            ENT_QUOTES ensures both single and double quotes are converted to HTML entities (e.g., ' → &#039;, " → &quot;).
            'UTF-8' specifies the character encoding to use:
                - Ensures multibyte characters (emojis, accented letters, non-Latin scripts) are handled correctly.
                - Prevents broken characters or misinterpretation.
            Always use 'UTF-8' if your app may handle international text or emojis.
            -->
        <?php endif; ?>

        <form class="form" method="post" action="index.php?controller=student&action=add">
            <div class="field">
                <label for="name">Name</label>
                <input type="text" id="name" name="name" required value="<?php echo isset($_POST['name']) ?
htmlspecialchars($_POST['name'], ENT_QUOTES, 'UTF-8') : ''; ?>">
            </div>
            <div class="field">
                <label for="email">Email</label>
                <input type="email" id="email" name="email" required value="<?php echo isset($_POST['email']) ?
htmlspecialchars($_POST['email'], ENT_QUOTES, 'UTF-8') : ''; ?>">
            </div>
            <div class="actions">
                <button class="btn btn-primary" type="submit">Save</button>
                <a class="btn btn-secondary" href="index.php?controller=student&action=list">Cancel</a>
            </div>
        </form>
    </div>
</body>

</html>
```

Output:

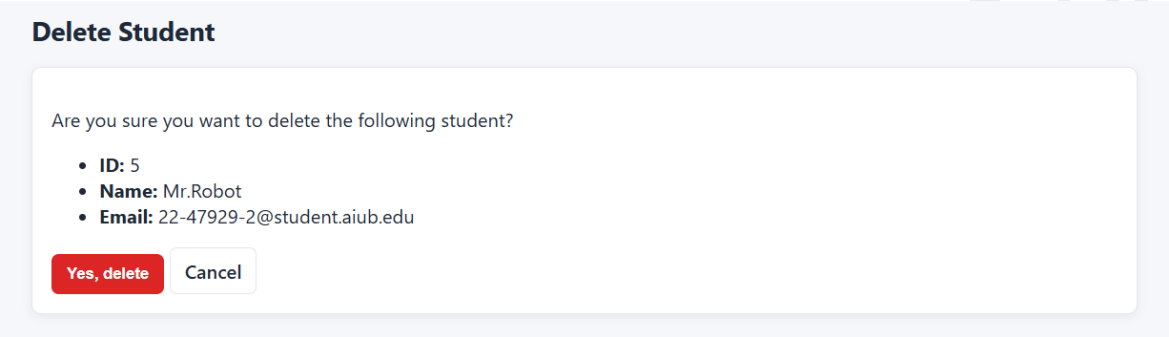


student_management\views\students\delete.php

```
<!doctype html>
<html>

<head>
    <meta charset="utf-8">
    <title>Delete Student</title>
    <link rel="stylesheet" href="/student_management/styles.css">
    <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>

<body>
    <div class="container">
        <h1>Delete Student</h1>

        <?php if (!empty($error)): ?>
            <div class="message-error"><?php echo htmlspecialchars($error, ENT_QUOTES, 'UTF-8'); ?></div>
        <?php endif; ?>

        <?php if (!empty($student)): ?>
            <div class="card">
                <p>Are you sure you want to delete the following student?</p>
                <ul>
                    <li><strong>ID:</strong> <?php echo htmlspecialchars($student['id'], ENT_QUOTES, 'UTF-8'); ?></li>
                    <li><strong>Name:</strong> <?php echo htmlspecialchars($student['name'], ENT_QUOTES, 'UTF-8'); ?></li>
```

```
                    <li><strong>Email:</strong> <?php echo htmlspecialchars($student['email'], ENT_QUOTES, 'UTF-8'); ?></li>
                </ul>
                <form method="post" action="index.php?controller=student&action=delete&id=<?php echo (int)$student['id']; ?>">
                    <button class="btn btn-danger" type="submit">Yes, delete</button>
                    <a class="btn btn-secondary" href="index.php?controller=student&action=list">Cancel</a>
                </form>
            </div>
        <?php else: ?>
            <div class="card">
                <p>Student not found.</p>
                <p><a class="btn btn-secondary" href="index.php?controller=student&action=list">Back to list</a></p>
            </div>
        <?php endif; ?>
    </div>
</body>

</html>
```

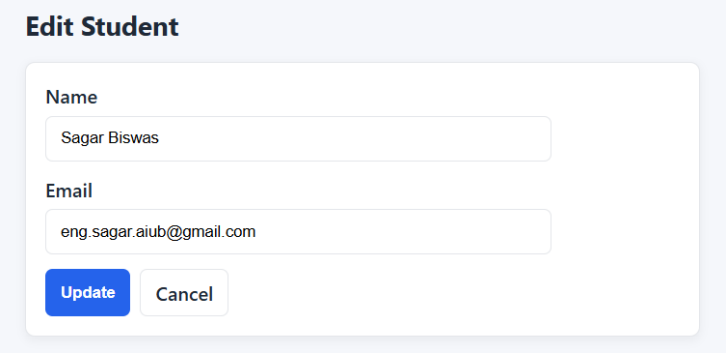
Output:



student_management\views\students\edit.php

```
<!doctype html>
<html>

<head>
    <meta charset="utf-8">
    <title>Edit Student</title>
    <link rel="stylesheet" href="/student_management/styles.css">
    <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>

<body>
    <div class="container">
        <h1>Edit Student</h1>

        <?php if (!empty($error)): ?>
            <div class="message-error"><?php echo htmlspecialchars($error, ENT_QUOTES, 'UTF-8'); ?></div>
        <?php endif; ?>

        <?php if (!empty($student)): ?>
            <form class="form" method="post" action="index.php?controller=student&action=edit&id=<?php echo (int)$student['id']; ?>">
                <div class="field">
                    <label for="name">Name</label>
                    <input type="text" id="name" name="name" value="<?php echo htmlspecialchars($student['name'] ?? '', ENT_QUOTES,
'UTF-8'); ?>" required>
                </div>
                <div class="field">
                    <label for="email">Email</label>
                    <input type="email" id="email" name="email" value="<?php echo htmlspecialchars($student['email'] ?? '', ENT_QUOTES,
'UTF-8'); ?>" required>
                </div>
                <div class="actions">
                    <button class="btn btn-primary" type="submit">Update</button>
                    <a class="btn btn-secondary" href="index.php?controller=student&action=list">Cancel</a>
                </div>
            </form>
        <?php else: ?>
            <div class="card">
                <p>Student not found.</p>
                <p><a class="btn btn-secondary" href="index.php?controller=student&action=list">Back to list</a></p>
            </div>
        <?php endif; ?>
    </div>
</body>

</html>
```

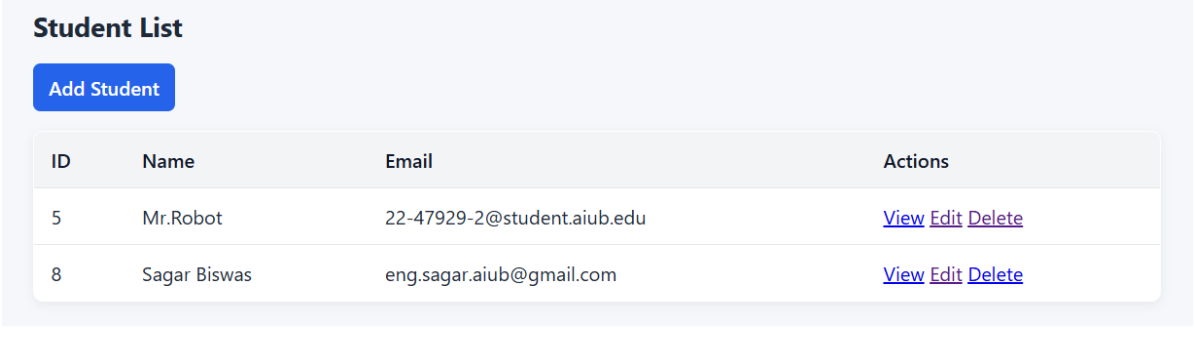
Output:



student_management\views\students\list.php

```php
<!doctype html>
<html>

<head>
    <meta charset="utf-8">
    <title>Students</title>
    <link rel="stylesheet" href="/student_management/styles.css">
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <style>
        body {
            visibility: visible
        }
    </style>
    <script>
        document.documentElement.style.visibility = 'visible';
    </script>
    <script>
        document.documentElement.style.visibility = 'visible';
    </script>
    <script>
        document.documentElement.style.visibility = 'visible';
    </script>
</head>

<body>
    <div class="container">
        <h1>Student List</h1>
        <div class="actions-bar">
            <a class="btn btn-primary" href="index.php?controller=student&action=add">Add Student</a>
        </div>
        <table class="table">
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Email</th>
                <th>Actions</th>
            </tr>
            <?php if (!empty($students) && is_array($students)): foreach ($students as $student): ?>
                    <tr>
                        <td><?php echo $student['id']; ?></td>
                        <td><?php echo $student['name']; ?></td>
                        <td><?php echo $student['email']; ?></td>
                        <td>
                            <a href="index.php?controller=student&action=view&id=<?php echo $student['id']; ?>">View</a>
                            <a href="index.php?controller=student&action=edit&id=<?php echo $student['id']; ?>">Edit</a>
                            <a href="index.php?controller=student&action=delete&id=<?php echo $student['id']; ?>">Delete</a>
                        </td>
                    </tr>
                <?php endforeach;
            else: ?>
                <tr>
                    <td colspan="4">No students found.</td>
                </tr>
            <?php endif; ?>
        </table>
    </div>
</body>

</html>
```

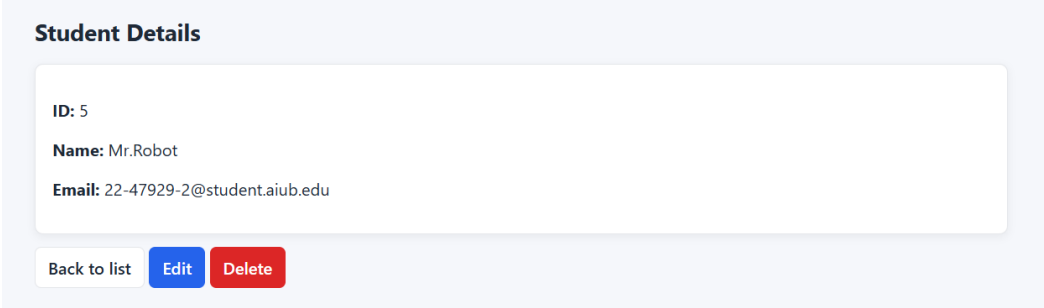
Output:



student_management\views\students\view.php

```php
<!doctype html>
<html>

<head>
    <meta charset="utf-8">
    <title>Student Details</title>
    <link rel="stylesheet" href="/student_management/styles.css">
    <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>

<body>
    <div class="container">
        <h1>Student Details</h1>
        <?php if (!empty($student)): ?>
            <div class="card">
                <p><strong>ID:</strong> <?php echo htmlspecialchars($student['id'], ENT_QUOTES, 'UTF-8'); ?></p>
                <p><strong>Name:</strong> <?php echo htmlspecialchars($student['name'], ENT_QUOTES, 'UTF-8'); ?></p>
                <p><strong>Email:</strong> <?php echo htmlspecialchars($student['email'], ENT_QUOTES, 'UTF-8'); ?></p>
            </div>
        <?php else: ?>
            <div class="card">
                <p>Student not found.</p>
            </div>
        <?php endif; ?>
        <p style="margin-top:12px;">
            <a class="btn btn-secondary" href="index.php?controller=student&action=list">Back to list</a>
            <a class="btn btn-primary" href="index.php?controller=student&action=edit&id=<?php echo isset($student['id']) ?
(int)$student['id'] : ''; ?>">Edit</a>
            <a class="btn btn-danger" href="index.php?controller=student&action=delete&id=<?php echo isset($student['id']) ?
(int)$student['id'] : ''; ?>">Delete</a>
        </p>
    </div>
</body>

</html>
```

Output:



student_management\index.php

```php
<?php
// Include configuration
require_once 'config/database.php';

// Get controller and action from URL
$controller = isset($_GET['controller']) ? $_GET['controller'] : 'student'; // if no controller is specified, default to 'student'
$action = isset($_GET['action']) ? $_GET['action'] : 'list'; // if no action is specified, default to 'list'
// how to set specified controller and action? by URL query parameters ?controller=student&action=add

/* Test default value:

        Only controller is set          : http://localhost/student_management/index.php?controller=student
        Only action is set              : http://localhost/student_management/index.php?action=add
        No controller or action is set  : http://localhost/student_management/index.php
*/

/* Example URLs to access different functionalities by id:

        Edit student with ID 5      : http://localhost/student_management/index.php?controller=student&action=edit&id=5
        View student with ID 3      : http://localhost/student_management/index.php?controller=student&action=view&id=5
        Delete student with ID 2    : http://localhost/student_management/index.php?controller=student&action=delete&id=5
*/

// Load appropriate controller -- It loads immediately when index.php executes.
// require_once is used to include and evaluate the specified file during the execution of the script
require_once "controllers/{$controller}Controller.php"; // e.g., controllers/StudentController.php
// Load model
// ucfirst() function is used to convert the first character of a string to uppercase
require_once "models/" . ucfirst($controller) . ".php"; // e.g., models/Student.php

/*
```

```
Purpose of require_once:
    require_once tells PHP: “Before running the rest of the script, go fetch this file and include its code here.”
So when you do:
        require_once "controllers/{$controller}Controller.php";
        require_once "models/" . ucfirst($controller) . ".php";
#). PHP literally copies the code from those files into index.php at runtime.
#). Without this, PHP would have no idea what StudentController or Student is, and you’d get a “class not found” error when you try to
do:
        $controllerObj = new $controllerName($database);



Why load them here?
Because in MVC:
    Controllers contain the methods (listAction, addAction, etc.).
    Models contain the database logic (getAllStudents, addStudent, etc.).
Your index.php is the entry point (the front controller).
It decides:
        1. Which controller class file is needed (StudentController).
        2. Which model class file is needed (Student).
        3. Loads them into memory with require_once.
        4. Creates the controller object and runs the chosen method.



Think of it like a toolbox:
    #). index.php is the worker.
    #). require_once is opening the toolbox.
    #). Controllers and Models are the actual tools.
    #). If you don’t open the toolbox, the worker (PHP) has no tools to do the job → error.
*/

// Initialize controller and call action
$controllerName = ucfirst($controller) . 'Controller'; // ucfirst() function is used to convert the first character of a string to
uppercase; e.g., StudentController
$controllerObj = new $controllerName($database); // used to create an instance of the controller class; e.g., new
StudentController($database) creates an instance of the StudentController class of the controller class.
$actionMethod = $action . 'Action'; // e.g., listAction, addAction, editAction, deleteAction, viewAction

// Call the action method on the controller object
// Pass id if available, otherwise null
$controllerObj->$actionMethod($_GET['id'] ?? null);
```

student_management\indexWithSafetyChecks.php

```
<?php
// Include configuration
require_once 'config/database.php';

// Get controller and action from URL (with defaults)
$controller = $_GET['controller'] ?? 'student'; // default = student
$action = $_GET['action'] ?? 'list';           // default = list

/*
================================================================================
 Purpose of this file: indexWithSafetyChecks.php
================================================================================
 Acts as the Front Controller for the Student Management project.
 It routes requests to the correct controller and action, while applying
 safety checks to avoid fatal PHP errors.

 Why safety checks?
 ------------------
 - Prevents raw "500 Internal Server Error" or "Class not found" messages.
 - Shows clear, custom error messages without exposing sensitive details.
 - Makes debugging easier for developers while still being user-friendly.

 Examples:
 ---------
 1) Missing controller file:
    URL: http://localhost/student_management/indexWithSafetyChecks.php?controller=foo
    Output: "Error: Controller file not found → controllers/fooController.php"

 2) Missing action method:
    URL: http://localhost/student_management/indexWithSafetyChecks.php?action=bar
    Output: "Error: Action method <b>barAction</b> not found in StudentController."

 Default behavior:
 -----------------
 - If no controller/action is provided in the URL,
   it loads the StudentController → listAction().
================================================================================
*/

// ---------------- SAFETY CHECKS ---------------- //

// Controller file path
$controllerFile = "controllers/{$controller}Controller.php";
// Model file path
```

```php
$modelFile = "models/" . ucfirst($controller) . ".php";

// Check controller file exists
if (!file_exists($controllerFile)) {
        die("Error: Controller file not found → {$controllerFile}");
}
require_once $controllerFile;

// Check model file exists
if (!file_exists($modelFile)) {
        die("Error: Model file not found → {$modelFile}");
}
require_once $modelFile;

// Controller class name
$controllerName = ucfirst($controller) . 'Controller';

// Check class exists
if (!class_exists($controllerName)) {
        die("Error: Controller class <b>{$controllerName}</b> not found.");
}

// Create controller object
$controllerObj = new $controllerName($database);

// Action method
$actionMethod = $action . 'Action';

// Check method exists
if (!method_exists($controllerObj, $actionMethod)) {
        die("Error: Action method <b>{$actionMethod}</b> not found in {$controllerName}.");
}

// ---------------- EXECUTE REQUEST ---------------- //
$controllerObj->$actionMethod($_GET['id'] ?? null);
```

student_management\setup.sql

```sql
-- Initialize database and schema for student_management

-- Create database if it doesn't exist
CREATE DATABASE IF NOT EXISTS `student_management` CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE `student_management`;

-- Create students table
CREATE TABLE IF NOT EXISTS `students` (
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(100) NOT NULL,
  `email` VARCHAR(150) NOT NULL UNIQUE,
  `created_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updated_at` TIMESTAMP NULL DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- Optional seed data
INSERT INTO `students` (`name`, `email`)
VALUES
  ('Alice Johnson', 'alice@example.com'),
  ('Bob Smith', 'bob@example.com')
ON DUPLICATE KEY UPDATE `name` = VALUES(`name`);
```

student_management\styles.css

```css
/* Basic app styles */
:root {
    --bg: #f5f7fb;
    --card: #ffffff;
    --text: #1f2937;
    --muted: #6b7280;
    --primary: #2563eb;
    --primary-600: #1d4ed8;
    --danger: #dc2626;
    --danger-600: #b91c1c;
    --border: #e5e7eb;
    --shadow: 0 2px 8px rgba(0, 0, 0, .06);
}

* {
    box-sizing: border-box;
}

body {
    margin: 0;
```

```
    font-family: system-ui, -apple-system, Segoe UI, Roboto, Ubuntu, Cantarell, Noto Sans, Helvetica, Arial, "Apple Color Emoji",
"Segoe UI Emoji";
    background: var(--bg);
    color: var(--text);
}

.container {
    max-width: 960px;
    margin: 32px auto;
    padding: 0 16px;
}

h1 {
    font-size: 22px;
    margin: 0 0 16px;
}

.actions-bar {
    display: flex;
    gap: 8px;
    align-items: center;
    margin: 12px 0 16px;
}

.table {
    width: 100%;
    border-collapse: collapse;
    background: var(--card);
    border: 1px solid var(--border);
    border-radius: 8px;
    overflow: hidden;
    box-shadow: var(--shadow);
}

.table th,
.table td {
    padding: 12px 14px;
    border-bottom: 1px solid var(--border);
    text-align: left;
}

.table th {
    background: #f3f4f6;
    font-weight: 600;
    color: #111827;
}

.table tr:last-child td {
    border-bottom: none;
}

.table tbody tr:hover {
    background: #f9fafb;
}

.btn {
    display: inline-block;
    padding: 8px 12px;
    border-radius: 6px;
    text-decoration: none;
    font-weight: 600;
    border: 1px solid transparent;
    cursor: pointer;
}

.btn-primary {
    background: var(--primary);
    color: #fff;
}

.btn-primary:hover {
    background: var(--primary-600);
}

.btn-secondary {
    background: #fff;
    color: var(--text);
    border-color: var(--border);
}

.btn-secondary:hover {
    background: #f9fafb;
}

.btn-danger {
    background: var(--danger);
    color: #fff;
}

.btn-danger:hover {
```

```
    background: var(--danger-600);
}

.form {
    background: var(--card);
    padding: 16px;
    border: 1px solid var(--border);
    border-radius: 8px;
    box-shadow: var(--shadow);
    max-width: 560px;
}

.form .field {
    margin-bottom: 12px;
}

.form label {
    display: block;
    font-weight: 600;
    margin-bottom: 6px;
}

.form input[type="text"],
.form input[type="email"] {
    width: 100%;
    max-width: 420px;
    padding: 10px 12px;
    border: 1px solid var(--border);
    border-radius: 6px;
}

.form .actions {
    margin-top: 12px;
    display: flex;
    gap: 8px;
}

.message-error {
    color: var(--danger-600);
    background: #fee2e2;
    border: 1px solid #fecaca;
    padding: 8px 10px;
    border-radius: 6px;
    margin: 8px 0 12px;
}

.card {
    background: var(--card);
    padding: 16px;
    border: 1px solid var(--border);
    border-radius: 8px;
    box-shadow: var(--shadow);
}
```

Project GITHUB link:

https://github.com/SagarBiswas-MultiHAT/Student-Management-MVC-Learning-Project

---------- X ----------