Part A: System Architecture Design

Chosen Architecture Pattern: MVC (Model–View–Controller)

The system is designed using the MVC Architecture Pattern, which separates the data (Model), user interface (View), and application logic (Controller). This improves modularity, maintainability, and scalability.

Architecture Description

1. Model (Data Layer)

- **Handles data storage and retrieval.**
- **Defines database schemas for courses, students, and enrollment records.**
- **Performs data validation and ensures data integrity.**

2. View (Presentation Layer)

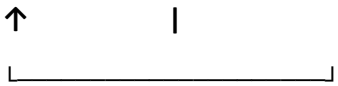
- **Handles the user interface.**
- **Displays data in forms, tables, or dashboards.**
- **Sends user inputs to the Controller.**

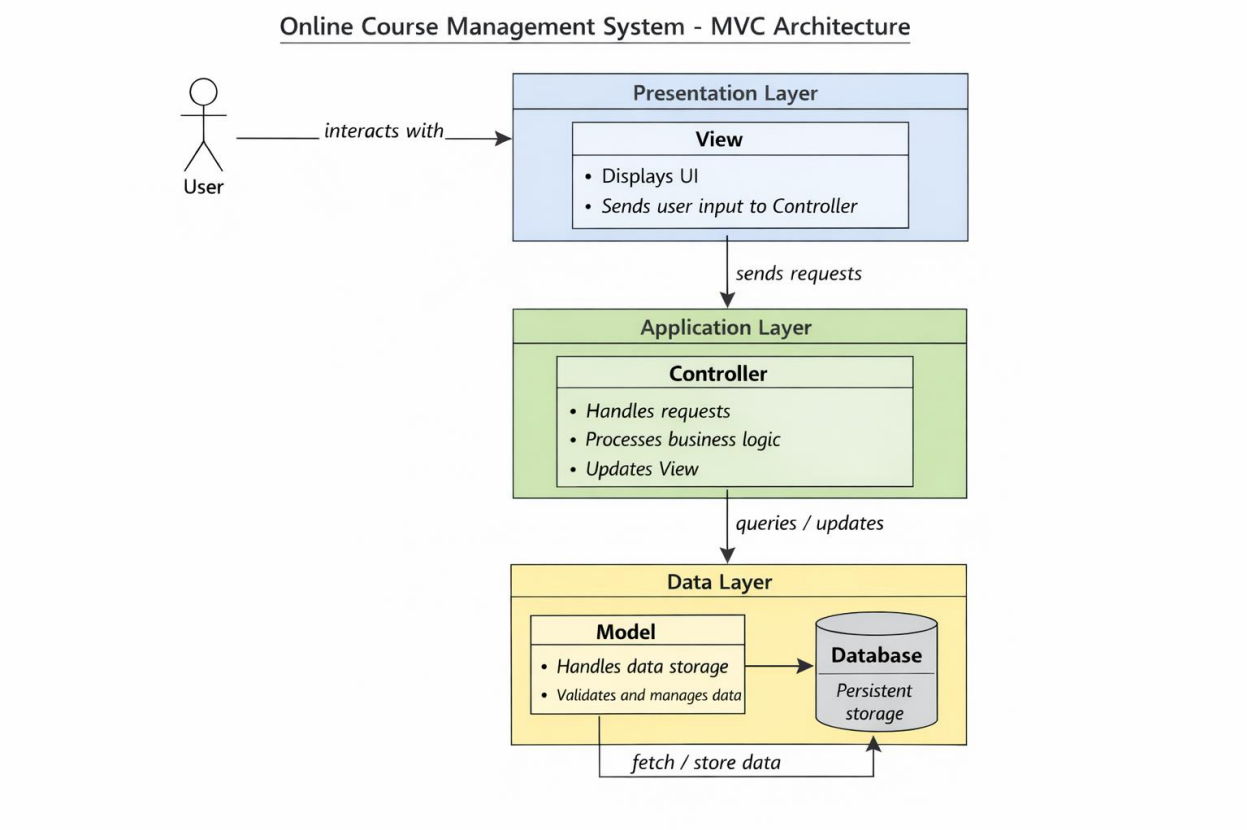
3. Controller (Application Layer)

- **Handles application logic.**
- **Receives user requests from the View.**
- **Calls the Model to access or modify data.**
- **Updates the View with the results.**

MVC Architecture Diagram (Draw.io)

User → View → Controller → Model → Database





Flow Explanation

1. **User interacts with the View (e.g., views available courses).**
2. **View sends request to Controller.**
3. **Controller processes request and queries the Model.**
4. **Model retrieves or updates data in the Database.**
5. **Controller sends data to View to display results to the user.**

Part B: Base Folder Structure

Project Name Example: OnlineCourseManagementSystem

Folder Structure:

OnlineCourseManagementSystem/

|— controllers/

│ └— courseController.py

│ └— studentController.py

├— models/

│ └— courseModel.py

│ └— studentModel.py

├— views/

│ └— courseView.html

│ └— studentView.html

├— services/

│ └— courseService.py

│ └— studentService.py

├— config/

│ └— database.py

├— static/

│ └— style.css

├— templates/

│ └— index.html

├— app.py

└— requirements.txt

Folder Description

- **controllers – Handles user requests and interacts with Models and Services.**
- **models – Defines database structure (courses, students) and handles persistence.**
- **views – Displays data and receives input from users.**
- **services – Handles business logic (e.g., enrollment rules, course progress).**
- **config – Stores database connection settings.**
- **static/templates – Contains CSS, JS, and HTML templates.**
- **app.py – Main entry point to run the server and handle routing.**
- **requirements.txt – Lists dependencies required for the project.**

Dependency Injection Implementation

```
class CourseController:
    def __init__(self, course_service):
        self.course_service = course_service
```

Benefits:

- **Easier testing with mock data.**
- **Loose coupling between Controller and Service.**
- **Modular and maintainable code.**
- **Follows the Dependency Inversion Principle.**

Monolithic Architecture Used

This project uses Monolithic Architecture because:

1. **Single Deployable Unit**
    - **All components (Controllers, Services, Models, Views) are packaged and deployed together.**
    - **Only one server (app.py) runs the entire system.**
2. **Low Latency Communication**
    - **Controller, Service, and Model communicate within the same memory space.**
    - **No network calls between components are required.**
3. **Simplicity for Academic Projects**
    - **Ideal for demonstrating MVC concepts without the complexity of microservices.**
    - **Easier to develop, test, and debug.**
4. **Centralized Configuration and Maintenance**
    - **Database settings, routes, and logic are all in one codebase.**
    - **Updating one component does not require deploying multiple services.**