

**COMSATS University Islamabad,**

**Abbottabad Campus**

# *Software Design and Architecture*

# *Submitted By: Saud Ur Rahman*

*Registration N0: Fa22-Bse-048*

## Submitted To: Mukhtiar Zamin

*Date: 07/ December / 2024*

Java Implementation

MVC-Labs/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ ├── Model.java

│ │ │ ├── View.java

│ │ │ ├── Controller.java

│ │ │ ├── Student.java

│ │ │ ├── Main.java

#### **Spring Boot Implementation**

MVC-Labs/

├── spring-boot-mvc/

│ ├── src/

│ │ ├── main/

│ │ │ ├── java/

│ │ │ │ └── com/

│ │ │ │ └── example/

│ │ │ │ └── mvclabs/

│ │ │ │ ├── controller/

│ │ │ │ │ └── StudentController.java

│ │ │ │ ├── model/

│ │ │ │ │ └── Student.java

│ │ │ │ └── MvcLabsApplication.java

│ │ ├── resources/

│ │ │ ├── templates/

│ │ │ │ └── student.html

│ │ │ ├── application.properties

│ ├── pom.xml

**MVC MODEL**

**Introduction to the Technology Stack**

The **MVC (Model-View-Controller)** pattern is a widely used architectural design pattern in software development. It's primarily used to separate the concerns of data, user interface, and user interaction, which makes it easier to maintain and scale applications.

* **Model**: Represents the data and business logic of the application. It manages the data and responds to requests from the controller to modify the state or retrieve data.
* **View**: Represents the UI (user interface) elements that display data. It gets information from the controller and reflects changes in the application state.
* **Controller**: Acts as an intermediary between the Model and View. It takes user input from the View, processes it (possibly updating the Model), and updates the View accordingly.

The **MVC** pattern is commonly used in Java for creating applications with graphical user interfaces, such as desktop apps, web apps, and mobile apps.

**Pros and Cons of MVC**

**Pros:**

1. **Separation of Concerns**:
   * The primary benefit of MVC is the separation of concerns, which allows developers to manage data (Model), user interfaces (View), and input logic (Controller) independently. This promotes modularity and makes it easier to scale and maintain the codebase.
2. **Improved Maintenance**:
   * Since each component is separated, changes in one part (for example, changes to the user interface) don't affect other parts (such as the business logic).
3. **Reusability**:
   * The Model, View, and Controller can be reused across different parts of the application. For example, you can have multiple views of the same data without modifying the model.
4. **Flexibility**:
   * It is easy to change the UI (View) without having to rework the business logic (Model), and vice versa.
5. **Multiple Views**:
   * Multiple views can interact with the same model, which is particularly useful for applications that require different representations of the same data.

**Cons:**

1. **Complexity**:
   * The MVC pattern can introduce additional complexity into the application, especially for smaller projects where simpler architectures might suffice.
2. **Increased Number of Classes**:
   * The MVC structure requires at least three classes (Model, View, Controller) for each component, which can make the codebase larger and harder to manage if not done properly.
3. **Tight Coupling Between Components**:
   * In some implementations, the View and Controller may end up being tightly coupled, which can undermine the benefits of separation of concerns.
4. **Learning Curve**:
   * For beginners, understanding how to implement and use MVC correctly may have a steeper learning curve.

Output

