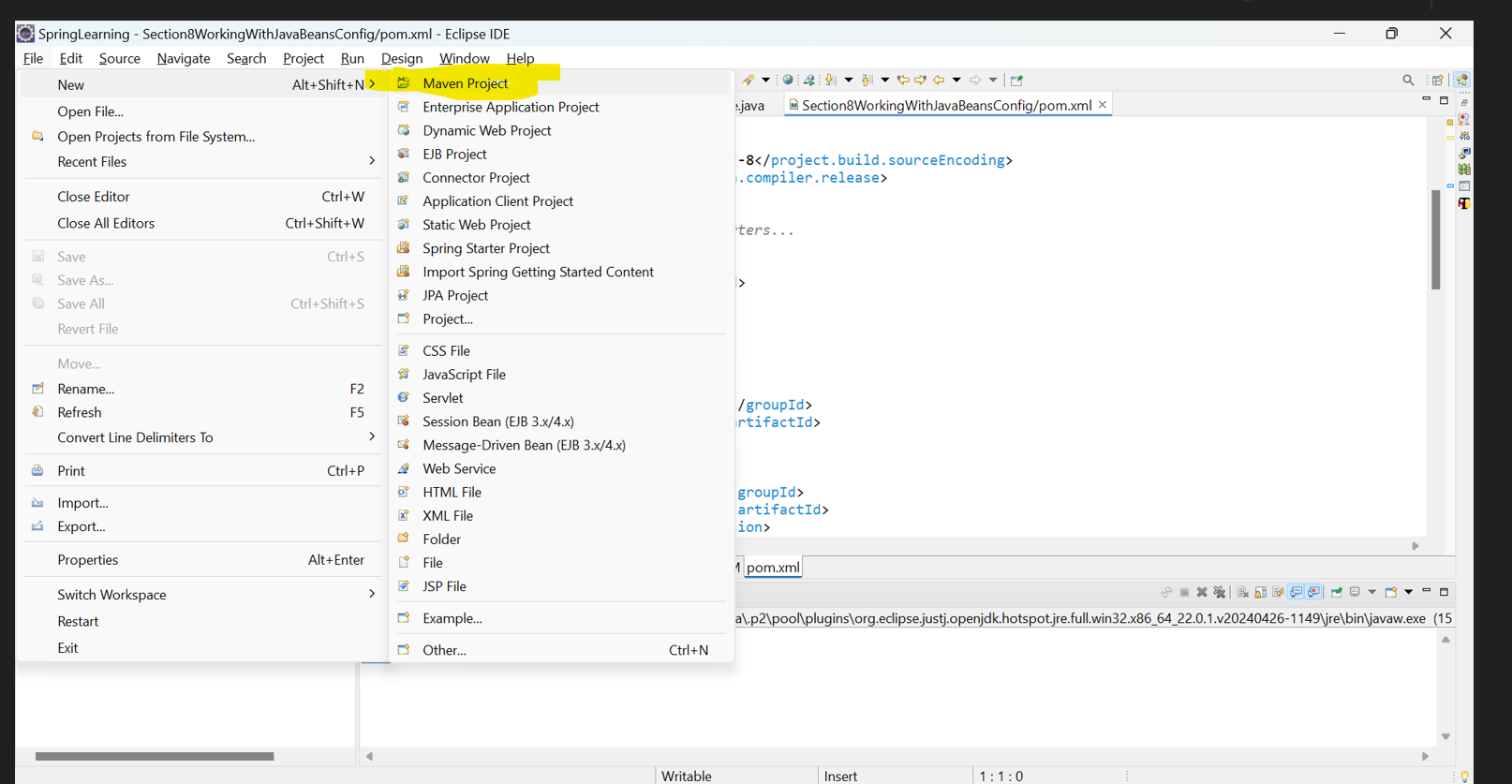
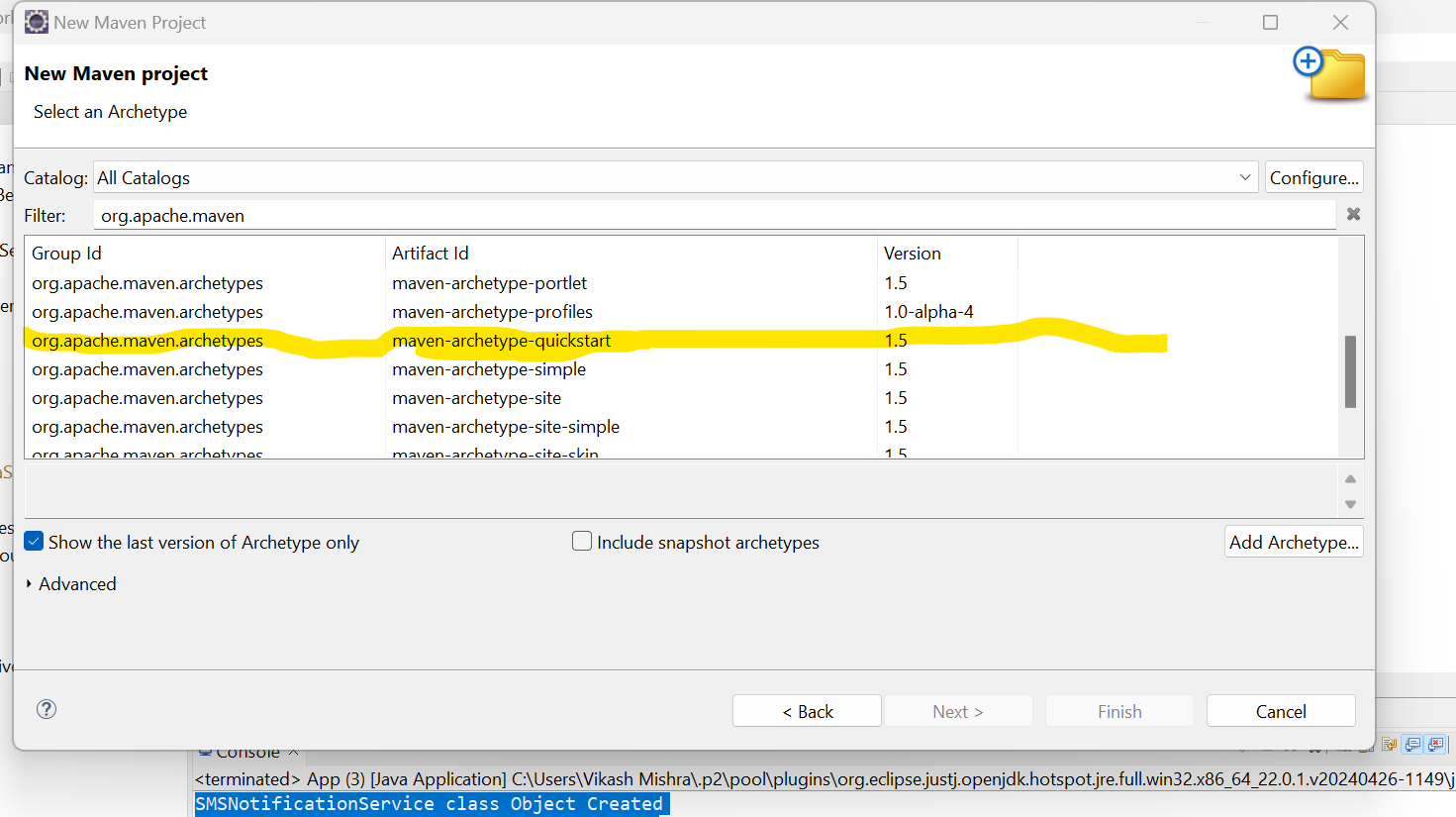
# **Spring Boot**

**Prerequisite**

**Project Creation**

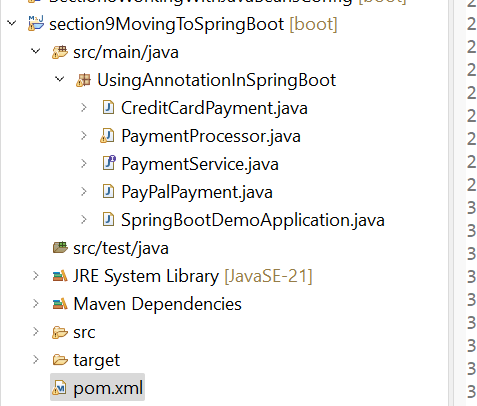
****

****

**POM.XML File**



**Project Structure**

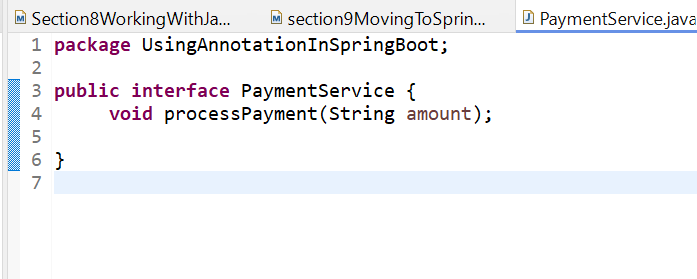
****

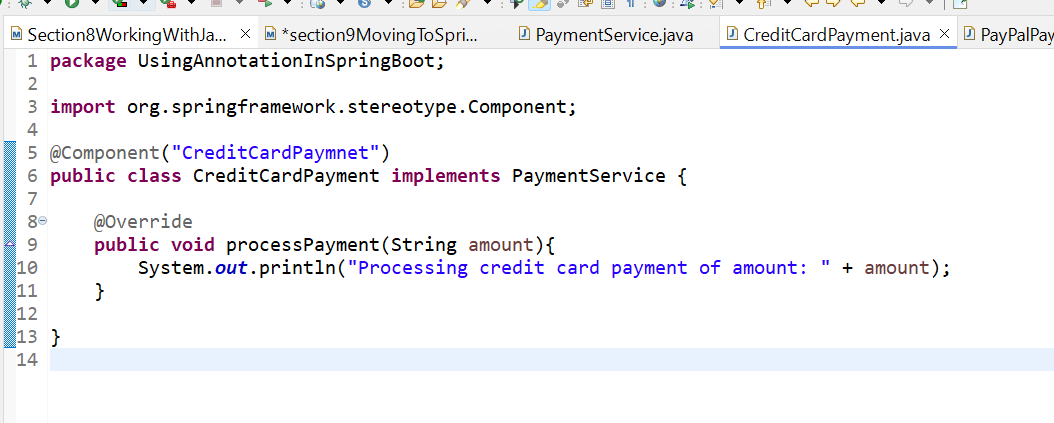
1. **Using Annotation in Spring Boot**

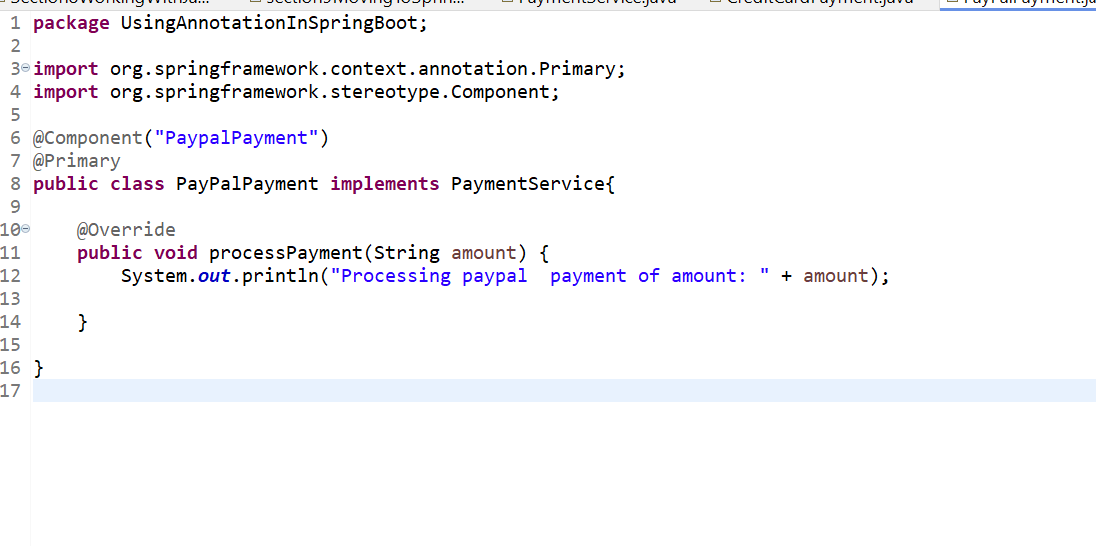
Here's the structured content for a payment application that mirrors the previous example while keeping the class names relevant to payments (CreditCardPayment and PayPalPayment). The main points cover the application setup and the required Spring annotations.

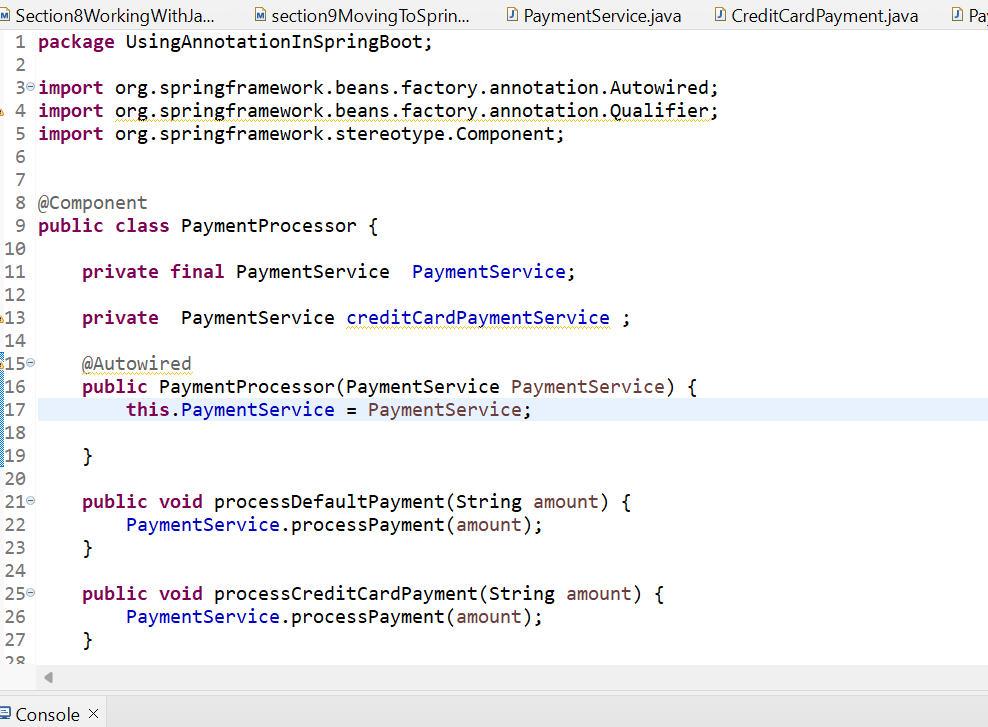
**Final Structure for Payment Application**

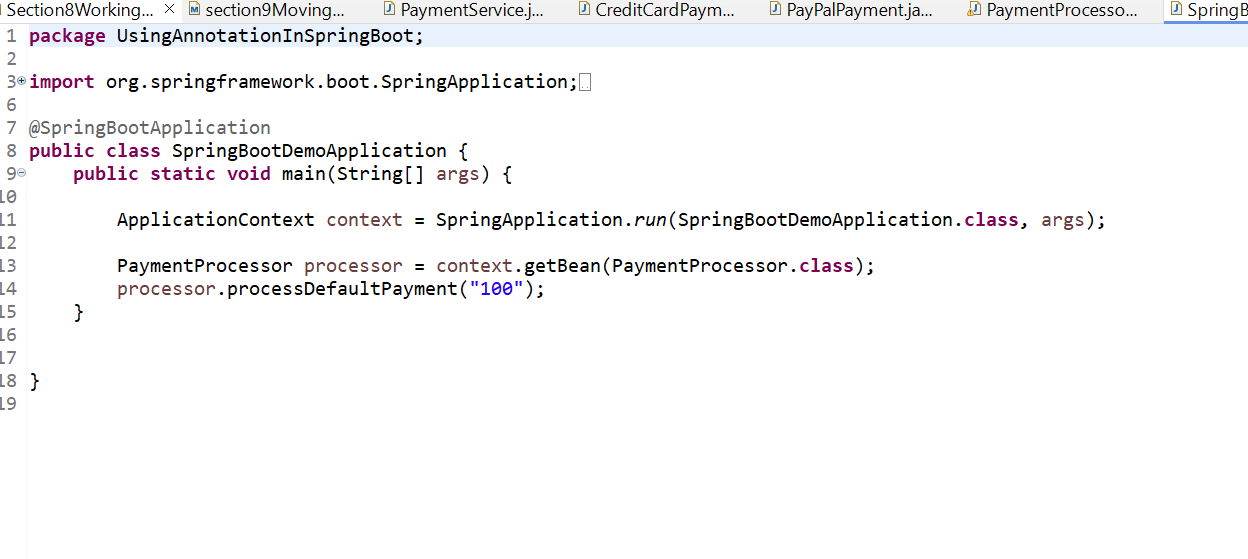
1. **SpringBootDemoApplication.java** (Main class)
2. **PaymentService.java** (Interface)
3. **CreditCardPayment.java** (Implementation of PaymentService)
4. **PayPalPayment.java** (Implementation of PaymentService)
5. **PaymentProcessor.java** (Class that uses both services)











**1.1 Explanation of Key Annotations and Concepts**

1. **@SpringBootApplication**: This annotation in the main class marks it as a Spring Boot application and enables component scanning.
2. **@Component**: This annotation is used on service classes to register them as Spring beans.
3. **@Primary**: This annotation on PayPalPayment indicates that it should be the default bean injected when no specific qualifier is used.
4. **@Autowired**: Automatically injects the dependency into the class. In PaymentProcessor, it's used to inject both the primary payment service and the credit card payment service.
5. **@Qualifier**: Specifies which bean to inject when multiple candidates are available. In this case, it specifies the CreditCardPayment implementation.

**1.2 Summary**

This setup for the payment application maintains the structural integrity and annotations while focusing on payment processing logic. The payment service implementations allow flexibility in handling various payment methods, demonstrating effective use of Spring’s dependency injection features.

# **Spring vs. Spring Boot vs. Spring MVC**

**2.1 Spring vs. Spring Boot**

**Spring:** Spring Framework is the most popular application development framework of Java. The main feature of the Spring Framework is **dependency Injection** or **Inversion of Control** (IoC). With the help of Spring Framework, we can develop a **loosely** coupled application. It is better to use if application type or characteristics are purely defined.

**Spring Boot:** Spring Boot is a module of Spring Framework. It allows us to build a stand-alone application with minimal or zero configurations. It is better to use if we want to develop a simple Spring-based application or RESTful services.

The primary comparison between Spring and Spring Boot are discussed below:

|  |  |
| --- | --- |
| **Spring** | **Spring Boot** |
| **Spring Framework** is a widely used Java EE framework for building applications. | **Spring Boot Framework** is widely used to develop **REST APIs**. |
| It aims to simplify Java EE development that makes developers more productive. | It aims to shorten the code length and provide the easiest way to develop **Web Applications**. |
| The primary feature of the Spring Framework is **dependency injection**. | The primary feature of Spring Boot is **Autoconfiguration**. It automatically configures the classes based on the requirement. |
| It helps to make things simpler by allowing us to develop **loosely coupled** applications. | It helps to create a **stand-alone** application with less configuration. |
| The developer writes a lot of code (**boilerplate code**) to do the minimal task. | It **reduces** boilerplate code. |
| To test the Spring project, we need to set up the sever explicitly. | Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc. |
| It does not provide support for an in-memory database. | It offers several plugins for working with an embedded and **in-memory** database such as **H2**. |
| Developers manually define dependencies for the Spring project in **pom.xml**. | Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care of downloading the dependencies **JARs** based on Spring Boot Requirement. |

* 1. **Spring MVC:**

Spring MVC is a Web MVC Framework for building web applications. It contains a lot of configuration files for various capabilities. It is an HTTP oriented web application development framework.

Spring Boot and Spring MVC exist for different purposes. The primary comparison between Spring Boot and Spring MVC are discussed below:

|  |  |
| --- | --- |
| **Spring Boot** | **Spring MVC** |
| **Spring Boot** is a module of Spring for packaging the Spring-based application with sensible defaults. | **Spring MVC** is a model view controller-based web framework under the Spring framework. |
| It provides default configurations to build **Spring-powered** framework. | It provides **ready to use** features for building a web application. |
| There is no need to build configuration manually. | It requires build configuration manually. |
| There is **no requirement** for a deployment descriptor. | A Deployment descriptor is **required**. |
| It avoids boilerplate code and wraps dependencies together in a single unit. | It specifies each dependency separately. |
| It **reduces** development time and increases productivity. | It takes **more** time to achieve the same. |

1. **Spring Boot Architecture**

Spring Boot is a module of the Spring Framework. It is used to create stand-alone, production-grade Spring Based Applications with minimum efforts. It is developed on top of the core Spring Framework.

Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above (hierarchical structure) it.

Before understanding the Spring Boot Architecture, we must know the different layers and classes present in it. There are four layers in Spring Boot are as follows:

* **Presentation Layer**
* **Business Layer**
* **Persistence Layer**
* **Database Layer**

****

**3.0.1 Presentation Layer**: The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of views i.e., frontend part.

**3.0.2 Business Layer:** The business layer handles all the **business logic**. It consists of service classes and uses services provided by data access layers. It also performs **authorization** and **validation**.

**3.0.3 Persistence Layer:** The persistence layer contains all the **storage logic** and translates business objects from and to database rows.

**3.0.4 Database Layer:** In the database layer, **CRUD** (create, retrieve, update, delete) operations are performed.

**3.1 Spring Boot Flow Architecture**

****

The diagram you provided illustrates the architecture of a typical Spring Boot application, emphasizing how different components interact with each other in a flow for handling HTTP requests. Let’s break it down step by step:

**3.1.1 Client**

* The client is typically the front end of your application, which could be a web browser, a mobile app, or any other service making HTTP requests.
* It sends an **HTTP request** to the application (e.g., GET, POST, PUT, DELETE).

**3.1.2 Controller**

* The controller acts as a bridge between the client and the service layer.
* It receives the incoming HTTP requests and maps them to specific handler methods.
* In Spring Boot, controllers are usually annotated with @RestController or @Controller, which allows them to handle RESTful web services or standard web requests.
* The controller processes the request, interacts with the service layer, and returns an appropriate HTTP response.

**3.1.3 Service Layer**

* The service layer contains the business logic of the application. It processes the data received from the controller and interacts with the repository layer.
* Services are typically annotated with @Service, indicating their role in the application.
* The service layer often uses **dependency injection** to manage dependencies. This means that the service layer can call repository methods without creating repository instances manually.

**3.1.4 Model**

* The model represents the data structure of the application, usually corresponding to the entities that interact with the database.
* This can include classes annotated with @Entity, which defines how the data is structured in the database.
* The model can also include any data transfer objects (DTOs) that the application uses to send data between layers.

**3.1.5 Repository Layer**

* The repository layer is responsible for interacting with the database. It contains the data access logic to perform CRUD (Create, Read, Update, Delete) operations.
* In Spring Boot, repositories typically extend interfaces such as JpaRepository or CrudRepository, which provide built-in methods for data manipulation.
* This layer abstracts the data access, making it easier to manage database interactions.

**3.1.6 Database**

* The database is where the application's data is stored.
* Spring Boot applications often use JPA (Java Persistence API) or Spring Data JPA for ORM (Object-Relational Mapping) to map Java objects to database tables.
* The database can be any relational database (like MySQL, PostgreSQL) or NoSQL database (like MongoDB), depending on the application's requirements.

**3.1.7 Flow of Execution**

* **Client sends an HTTP request** to the controller.
* **Controller receives the request** and delegates the processing to the service layer.
* **Service layer executes business logic** and may interact with the repository to fetch or save data.
* **Repository interacts with the database** to perform the required CRUD operation.
* **Service returns the result** back to the controller.
* **Controller constructs an HTTP response** and sends it back to the client.

**Conclusion**

This architecture promotes a clean separation of concerns, making the application easier to maintain, test, and extend. Each layer has its own responsibilities, and they interact with each other in a well-defined manner, allowing for better scalability and modularity. This pattern is widely used in modern web applications built with Spring Boot and helps in developing robust and maintainable applications.