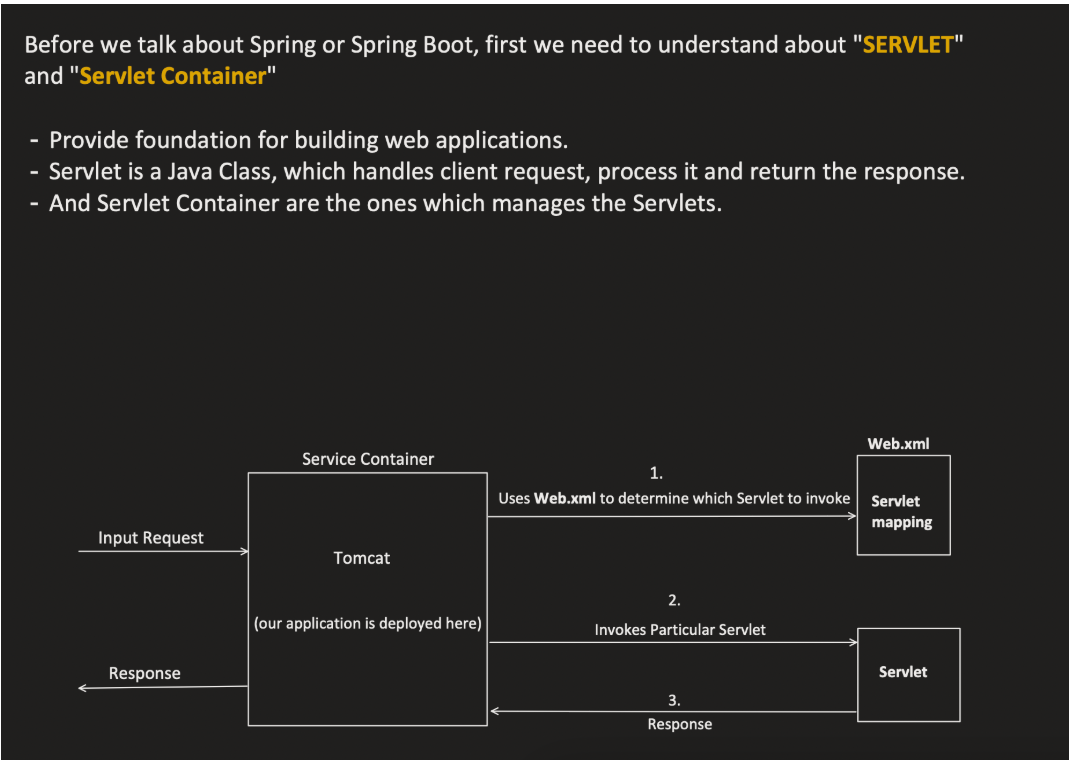
Udemy resource Url - [Notebook](https://notebook.zohopublic.in/public/notes/bietvaf4a031eeb7245f19e651a16232e077a)

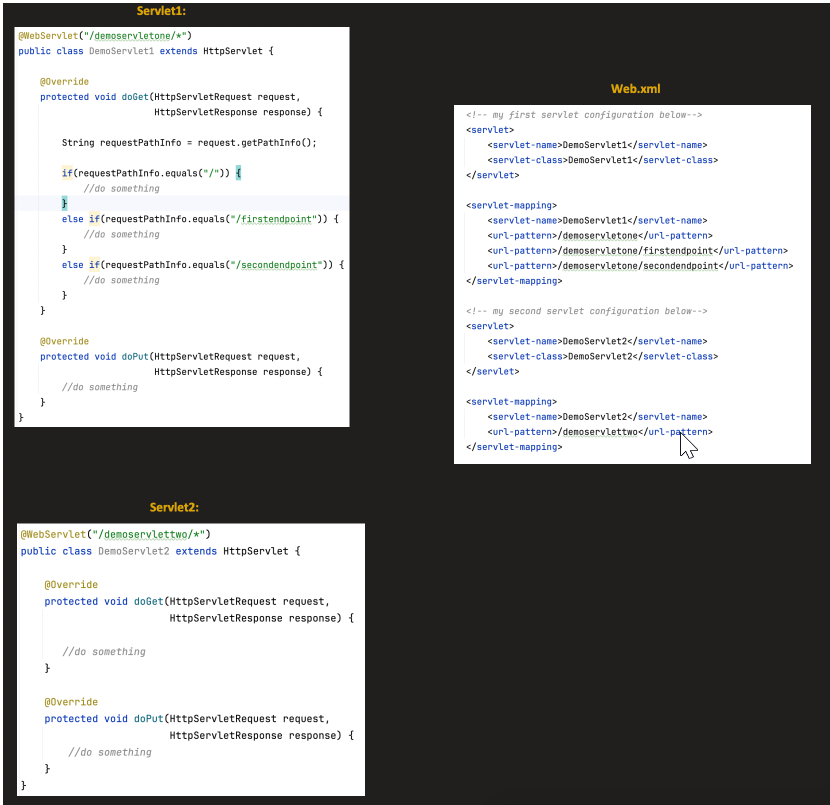
[Notebook](https://notebook.zohopublic.in/public/notes/bietv1d00e57bcbca4be89f155f6039985778)

# Spring Boot : Introduction

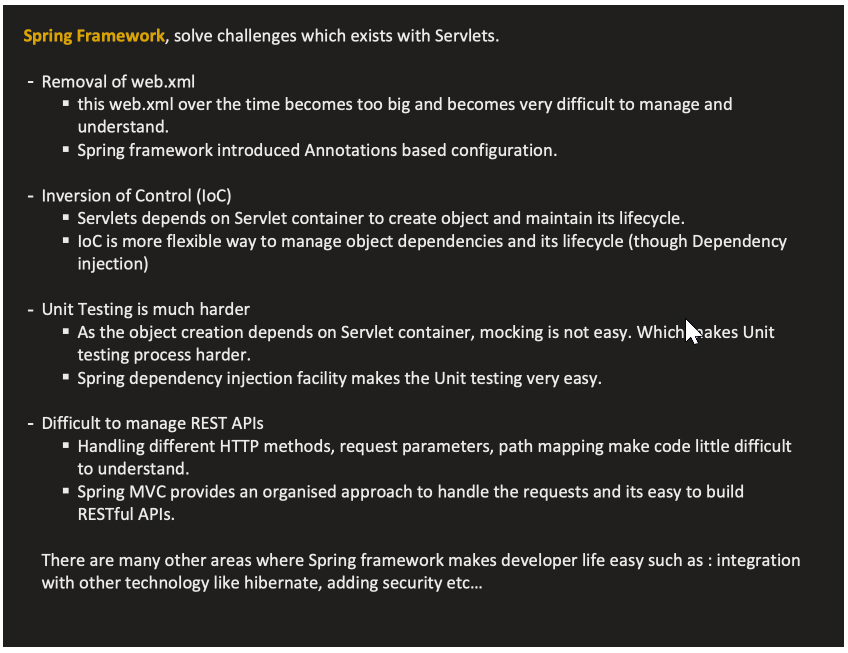


Servelet mapping Is needs to be done to call the endpoint.

Service Container- servlet container is the server where servelet will run



Spring is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides support to various frameworks such as [Struts](https://www.tpointtech.com/struts-2-tutorial), [Hibernate](https://www.tpointtech.com/hibernate-tutorial), Tapestry, [EJB](https://www.tpointtech.com/ejb-tutorial), [JSF](https://www.tpointtech.com/jsf-tutorial), etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.



Advantages of Spring Framework

There are many advantages of Spring Framework. They are as follows:

1) Predefined Templates

Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

Let's take the example of JdbcTemplate, you don't need to write the code for exception handling, creating connection, creating statement, committing transaction, closing connection etc. You need to write the code of executing query only. Thus, it save a lot of JDBC code.

2) Loose Coupling

The Spring applications are loosely coupled because of dependency injection.

3) Easy to test

The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.

4) Lightweight

Spring framework is lightweight because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface. That is why it is said non-invasive.

5) Fast Development

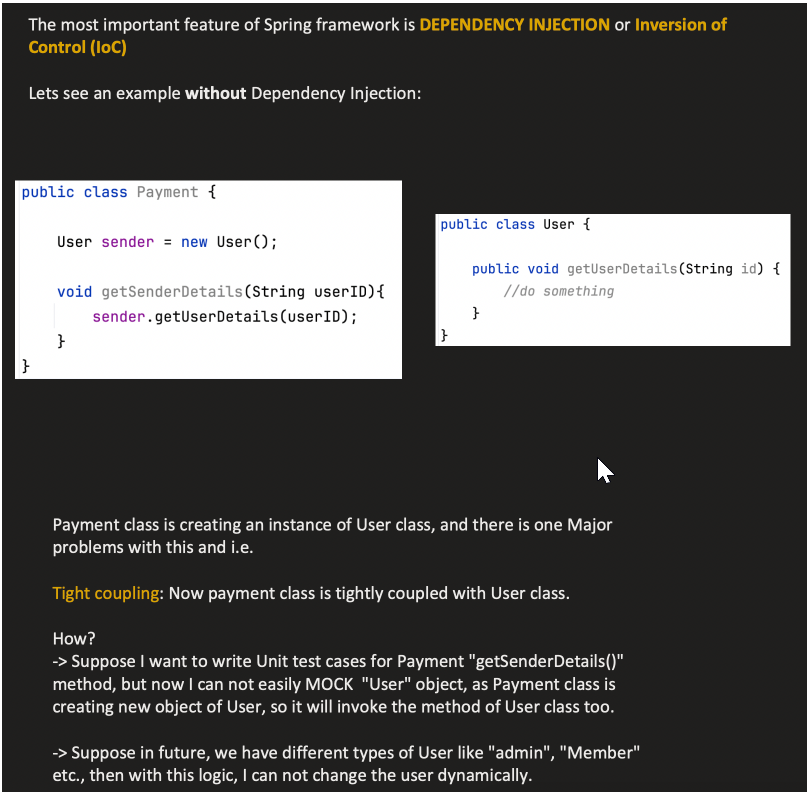
The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

6) Powerful abstraction

It provides powerful abstraction to JavaEE specifications such as [JMS](https://www.tpointtech.com/jms-tutorial), [JDBC](https://www.tpointtech.com/java-jdbc), JPA and JTA.

7) Declarative support

It provides declarative support for caching, validation, transactions and formatting.



A screenshot of a computer screen

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

## ✅ What is a DispatcherServlet?

The DispatcherServlet is the **front controller** in the Spring WebMVC architecture.  
It acts as the **central point** for handling **all HTTP requests**.

**Responsibilities:**

1. Receives the HTTP request from the client.
2. Delegates request handling to appropriate:
   * HandlerMapping (to find the controller),
   * HandlerAdapter (to invoke the controller method),
   * ViewResolver (to render the final response).
3. Sends the generated response back to the client.

So it’s like a **traffic cop** that routes the requests to the right controllers and assembles the response.

**🔧 Spring Boot WebMVC Flow Internally (Simplified):**

Client (Browser/Postman)

↓

Embedded Tomcat (or Jetty)

↓

DispatcherServlet (Front Controller)

↓

HandlerMapping

↓

Controller (@RestController/@Controller)

↓

HandlerAdapter

↓

Business Logic → Service → Repository

↓

ViewResolver (for @Controller with View) or HTTP Response (for @RestController)

↓

Response to Client

**📦 How Spring Boot Auto-Configures DispatcherServlet:**

Spring Boot **automatically configures** the DispatcherServlet when:

* You have spring-boot-starter-web in your dependencies.
* A class is annotated with @SpringBootApplication.
* @EnableAutoConfiguration is triggered (automatically included in @SpringBootApplication).

Spring Boot:

* Registers a DispatcherServlet bean.
* Maps it to URL pattern "/" by default.
* Sets up all necessary beans (RequestMappingHandlerMapping, ViewResolver, MessageConverters, etc.)

**🧠 Example:**

@RestController

public class HelloController {

@GetMapping("/hello")

public String hello() {

return "Hello, World!";

}

}

1. User sends GET /hello
2. DispatcherServlet receives it.
3. Uses HandlerMapping to find HelloController.hello().
4. Calls it using HandlerAdapter.
5. Returns response "Hello, World!"
6. DispatcherServlet sends response to client.

**✅ Summary (Interview-Style Points):**

| **Concept** | **Explanation** |
| --- | --- |
| DispatcherServlet | Central controller in Spring MVC that routes requests |
| Role | Front controller, handles routing, exception handling, response rendering |
| Auto-configuration | Spring Boot auto-registers it and its dependencies with spring-boot-starter-web |
| Controller types | @Controller for views, @RestController for REST APIs |
| ViewResolvers | Used for resolving view templates (like Thymeleaf, JSP) |
| Embedded Server | Tomcat/Jetty/Undertow handles incoming requests and passes them to DispatcherServlet |

A screenshot of a computer screen

AI-generated content may be incorrect.

EnableWebMvc 🡪 resolve all the dependency which spring MVC require.

@componentScan 🡪 tell spring start the scanning from the given path.

In Spring WebMVC we need to create dispatcher servlet. With servlet mappings.

# Spring Boot

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications.

It is a Spring module that provides the **RAD (*Rapid Application Development*)** feature to the Spring Framework. It is used to create a stand-alone Spring-based application that you can just run because it needs minimal Spring configuration.

Advantages of Spring Boot

* It creates **stand-alone** Spring applications that can be started using Java **-jar**.
* It tests web applications easily with the help of different **Embedded** HTTP servers such as **Tomcat, Jetty,** etc. We don't need to deploy WAR files.
* It provides opinionated '**starter**' POMs to simplify our Maven configuration.
* It provides **production-ready** features such as **metrics, health checks,** and **externalized configuration**.
* There is no requirement for **XML** configuration.
* It offers a **CLI** tool for developing and testing the Spring Boot application.
* It offers the number of **plug-ins**.
* It also minimizes writing multiple **boilerplate codes** (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.
* It **increases productivity** and reduces development time.

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen with text and images

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

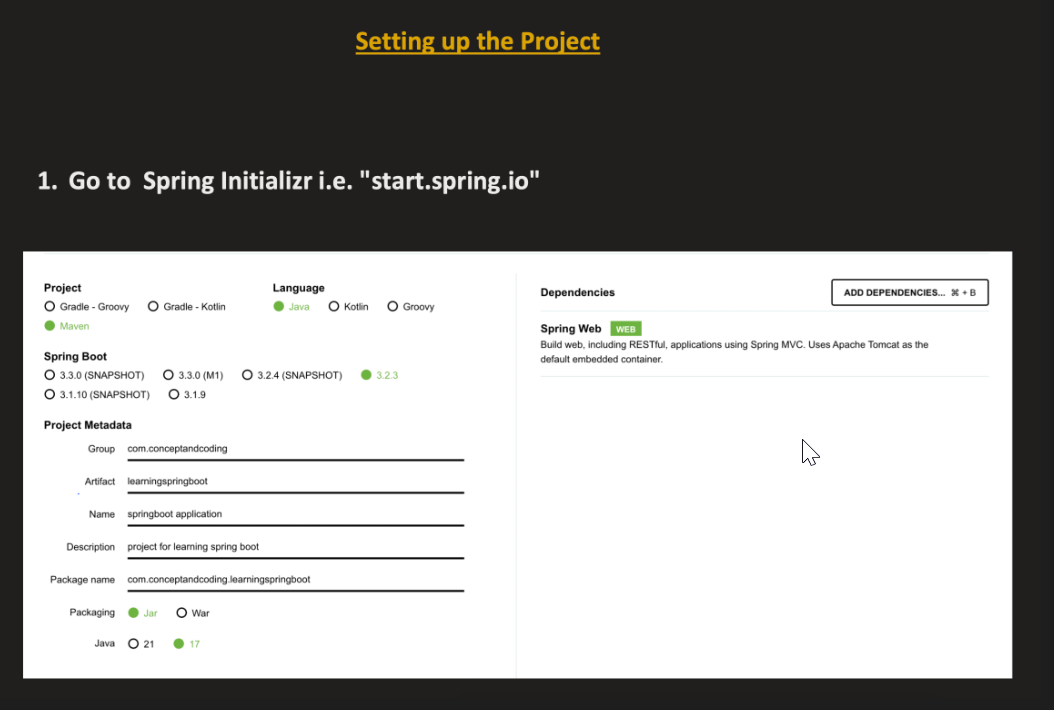
Convention means default configurations



A screenshot of a computer

AI-generated content may be incorrect.

# Project Structure and layered architecture



Packaging 🡪

Jar 🡪 Java Archive 🡪 when we need standalone java applications/microservices. A application which needs a server and can start it directly.

War 🡪 web archive 🡪 a application which needs complete bundle(html, css, jsp pages).

A diagram of a layering system

AI-generated content may be incorrect.

Controller Layer 🡪 classes which host the api. @controller, @restController

Service Layer 🡪 classes with business logic. @service

Repository Layer 🡪 work to connect with the DB. Interact with the DB. @repository

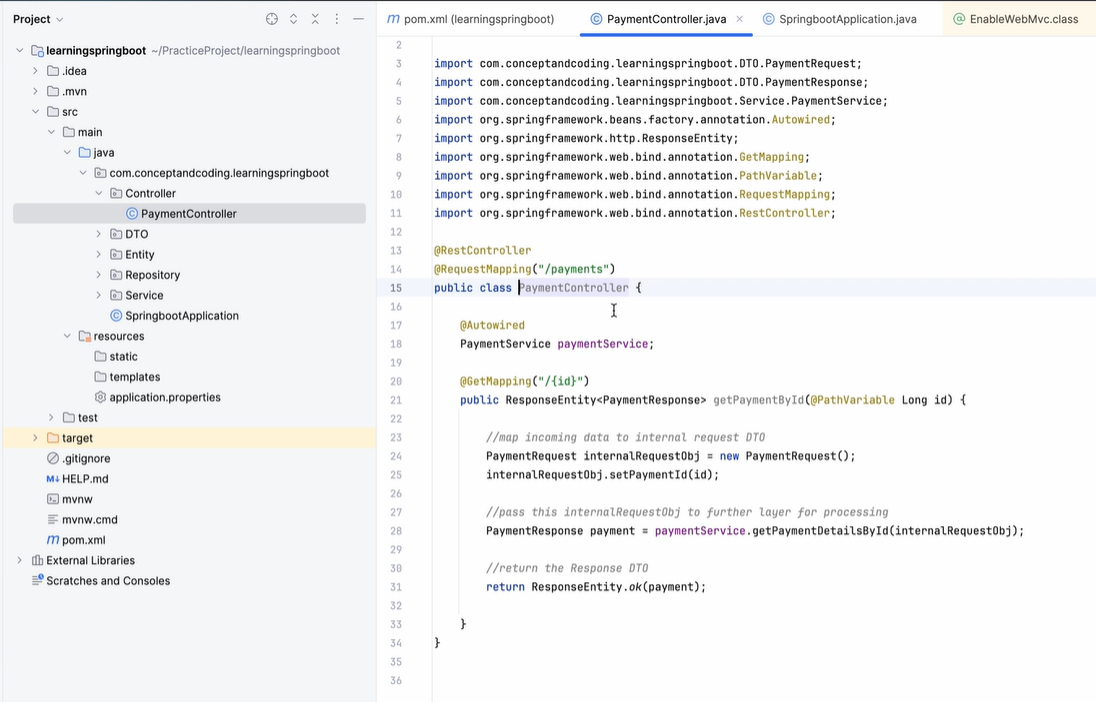
Other /Additional packages🡪

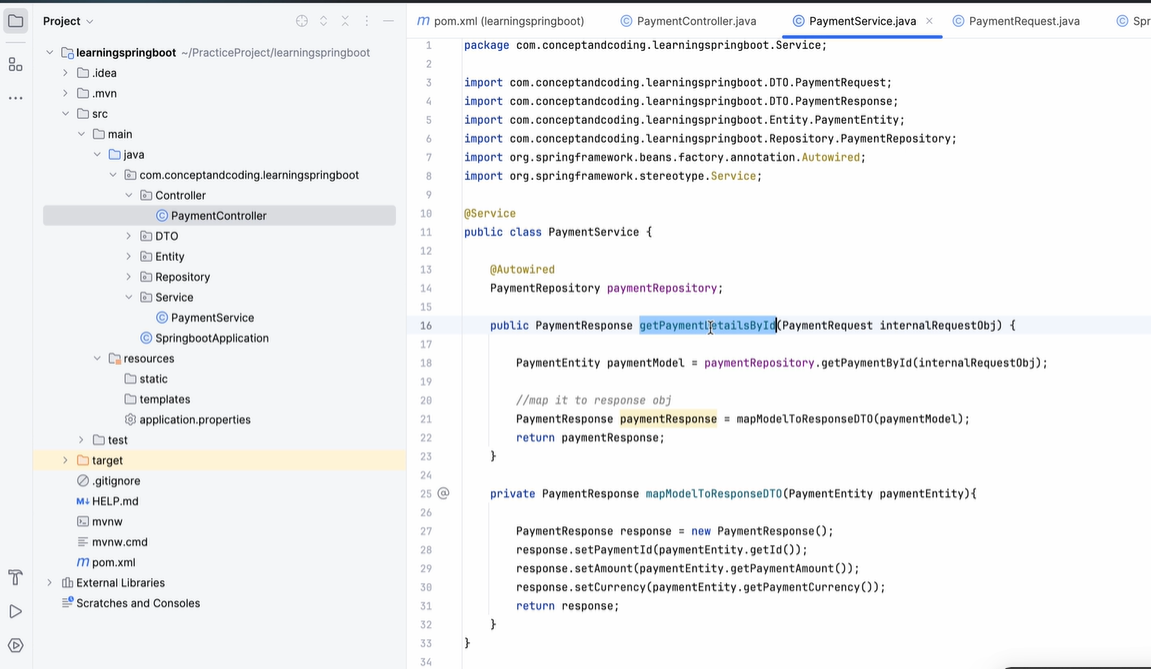
DTO 🡪 Data Transfer Objects 🡪 client is sending some object for ex from post mapping. We should map those objects to request DTO or to response Object. So that if any changes happen to client object does not affect the service layer. This mapping should be done by the controller. So that if any schema gets changes, then only controller should be changes. Same the response will affect the service layer. Service layer map entity object to DTO response.

Utility 🡪 common method among 2-4 classes.

Entity 🡪 which have the existence in DB. Service layer create a object of it and send it to repository layer to save it to DB. Same in case of fetching the result.

Configuration 🡪 all the configuration classes. Without doing any code changes we can change the configuration.





A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# Maven and its Lifecycle