Spring MVC in Spring Boot

give me the deep and detailed explaination of Spring MVC in spring boot so that i can able to understand it well and do not have any type of doubt in this. so give me the deep and detailed explaination

Below is a **very deep, step-by-step, zero-assumption explanation of Spring MVC in Spring Boot**, written specially for **you as a backend-focused learner** so that **no conceptual gaps remain**.
I'll go **from basics → internal flow → annotations → request lifecycle → real-world structure → common confusions**.

🌱 What is Spring MVC?

Spring MVC is a **web framework** inside the **Spring ecosystem** that helps you build **web applications and REST APIs** by following the **MVC (Model–View–Controller) design pattern**.

It is part of the **Spring Framework**, and when we use it with **Spring Boot**, most configurations are done **automatically**.

📌 Key entities:

- Spring MVC
- Spring Boot
- Spring Framework

🧱 What does MVC mean?

MVC is a **separation of responsibilities**.

Layer	Responsibility
Model	Holds application data & business logic

Layer	Responsibility
View	Displays data (HTML, JSON, XML, etc.)
Controller	Handles requests and coordinates everything

In **Spring Boot REST APIs**, **View = JSON**, not HTML.

🎯 Why Spring MVC Exists (Problem It Solves)

Before Spring MVC:

- Developers manually handled:
  - HTTP requests
  - URL parsing
  - Parameter extraction
  - Response creation
- Code was **tightly coupled**, **hard to maintain**, **error-prone**

Spring MVC:
✅ Handles HTTP lifecycle
✅ Maps URLs to methods
✅ Converts JSON ↔ Java automatically
✅ Separates concerns
✅ Highly testable

🧠 High-Level Architecture of Spring MVC

```yaml
Client (Browser / Postman)
        |
        v
DispatcherServlet  ← Front Controller
        |
        v
Controller
```

```
        |
        v
Service Layer
        |
        v
Repository (DB)
        |
        v
Response (JSON / View)
```

The **heart of Spring MVC** is:

❤️ DispatcherServlet

DispatcherServlet

🚦 DispatcherServlet (MOST IMPORTANT CONCEPT)

What is it?

The **DispatcherServlet** is the **single entry point** for **all HTTP requests** in Spring MVC.

📌 Think of it as:

> "Traffic police for HTTP requests"

What does DispatcherServlet do?

1. Receives request
2. Finds correct controller method
3. Calls business logic
4. Returns response to client

Why is it called *Front Controller*?

Because **every request** passes through it first.

This follows the **Front Controller Design Pattern**.

🔄 Complete Request Flow (Very Important)

Let’s say the client hits:

```bash
GET /users/1
```

STEP-BY-STEP FLOW 👇

1️⃣ Client sends HTTP request

```bash
GET /users/1
```

2️⃣ DispatcherServlet receives request

It does **NOT** handle logic itself.

3️⃣ HandlerMapping finds Controller

Spring checks:

```java
@GetMapping("/users/{id}")
```

📌 Finds matching method.

4️⃣ DispatcherServlet calls Controller method

```java
@GetMapping("/users/{id}")
public User getUser(@PathVariable Long id) {
```

```
    return userService.getUserById(id);
}
```

5️⃣ Controller calls Service

Business logic is executed.

6️⃣ Data returned to DispatcherServlet

7️⃣ ResponseBody / ViewResolver

- REST → JSON returned
- MVC → HTML page returned

8️⃣ Response sent to client

🧩 Core Components of Spring MVC

1️⃣ Controller

📌 Handles HTTP requests.

```java
@RestController
@RequestMapping("/users")
public class UserController {
}
```

Annotations:

- `@Controller` → For MVC views
- `@RestController` → For REST APIs (JSON)

2️⃣ HandlerMapping

📌 Decides **which controller method** handles a request.

Spring Boot uses:

- `RequestMappingHandlerMapping`

3️⃣ HandlerAdapter

📌 Invokes the selected controller method.

4️⃣ ViewResolver (Optional)

Used when returning:

- JSP
- Thymeleaf
- HTML

Not needed for pure REST APIs.

5️⃣ Model

Holds data sent to View.

```java
model.addAttribute("user", user);
```

🧪 Spring MVC Annotations (Deep Explanation)

🔹 `@RequestMapping`

Base URL mapping.

```java
@RequestMapping("/api")
```

- `@GetMapping`, `@PostMapping`

Shortcut annotations.

```java
@GetMapping("/users")
@PostMapping("/users")
```

- `@PathVariable`

Extracts value from URL.

```java
@GetMapping("/users/{id}")
public User get(@PathVariable Long id)
```

- `@RequestParam`

Query parameter.

```bash
/users?age=25
```

```java
@RequestParam int age
```

- `@RequestBody`

Converts JSON → Java Object.

```java
@PostMapping("/users")
```

```
public User save(@RequestBody User user)
```

✔️ Uses **Jackson** internally.

🔹 `@ResponseBody`

Converts Java → JSON.

Automatically applied in `@RestController`.

🔁 Data Binding & Conversion (Hidden Power)

Spring MVC automatically:

- Converts JSON to POJO
- Converts String → int → long
- Validates input

Powered by:

- **HttpMessageConverters**
- **Jackson**

🧵 Thread Model (Very Important)

- Each HTTP request → **new thread**
- Controller beans → **singleton**
- Must be **stateless**

⚠ Never use instance variables in controllers.

🧩 Spring MVC vs Spring Boot MVC

Feature	Spring MVC	Spring Boot MVC
Configuration	Manual	Auto
DispatcherServlet	Manual	Auto

Feature	Spring MVC	Spring Boot MVC
JSON Conversion	Manual	Auto
Embedded Server	❌	✅

Spring Boot **wraps Spring MVC** and **simplifies it**.

🏗️ Recommended Project Structure

```arduino

controller/
service/
repository/
model/
dto/
exception/
config/
```

✔️ Clean
✔️ Scalable
✔️ Industry standard

❌ Common Confusions (Very Important)

❓ Is Spring MVC only for HTML?

❌ NO
✔️ Used heavily for REST APIs

❓ Is Spring MVC different from Spring Boot?

✔️ Spring MVC = Framework
✔️ Spring Boot = Auto-configuration layer

❓ Can I use Spring MVC without Spring Boot?

✔️ YES (but complex)

🧠 Mental Model (Remember This)

> **Spring MVC = Request Routing Engine**
> **Spring Boot = Configuration Simplifier**

🏁 Final Summary

Spring MVC:

- Implements MVC pattern
- Uses DispatcherServlet
- Handles HTTP lifecycle
- Maps URLs to methods
- Converts JSON ↔ Java
- Separates concerns cleanly

Spring Boot:

- Auto-configures Spring MVC
- Reduces boilerplate
- Enables rapid development

If you want, next I can explain:
✅ Spring MVC **with diagrams**
✅ **Complete code flow** of a real API
✅ **Exception handling in Spring MVC**
✅ **Spring MVC vs Spring WebFlux**

Just tell me 👍