Spring Boot architecture explained

give me the deep and detailed explaination of Architecture 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-confusion explanation of Spring Boot Architecture**, written specially for **you as a backend-focused Java learner**, so that **every layer, flow, and concept is crystal clear**.

I'll explain **what exists**, **why it exists**, **how it connects**, and **what happens when a request comes in**.

🌱 What Do We Mean by "Spring Boot Architecture"?

Spring Boot Architecture is the **layered structure** that defines:

- How a Spring Boot application is **organized**
- How **HTTP requests flow**
- How **business logic, database access, and security** are separated
- How Spring Boot **reduces boilerplate** using auto-configuration

> Think of architecture as the **skeleton + nervous system** of your backend application.

🧠 Big Picture (Bird's Eye View)

A typical Spring Boot application looks like this:

```
scss

Client (Browser / Mobile / Postman)
        ↓
Controller Layer (REST API)
        ↓
```

```
Service Layer (Business Logic)
        ↓
Repository Layer (Database Access)
        ↓
Database (MySQL / PostgreSQL / MongoDB)
```

Spring Boot **automatically wires** all of this together.

🧩 Core Building Blocks of Spring Boot Architecture

Let's break it down one by one.

1️⃣ Application Entry Point (Main Class)

```java
@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }
}
```

What actually happens here?

When `SpringApplication.run()` executes:

1. Starts **Spring Application Context**
2. Scans all packages
3. Creates and manages **Beans**
4. Applies **Auto-Configuration**
5. Starts **embedded server**

📌 This is powered by the **Spring Framework**
Spring Framework

2️⃣ Embedded Web Server (Tomcat)

Spring Boot **does NOT need an external server**.

It embeds:

- Tomcat (default)
- Jetty
- Undertow

📌 Default is **Tomcat**

Apache Tomcat

Why this is powerful?

❌ Old Spring:

- Build WAR
- Deploy manually on server

✅ Spring Boot:

- Build JAR
- Run using `java -jar`

3️⃣ DispatcherServlet (Heart of MVC)

What is DispatcherServlet?

It is the **front controller**.

```
markdown


Every HTTP request
      ↓
DispatcherServlet
```

Responsibilities:

- Receives request
- Finds correct controller
- Calls business logic
- Returns response

📌 You NEVER create it manually
Spring Boot auto-configures it.

4️⃣ Controller Layer (API Layer)

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

    @GetMapping("/{id}")
    public User getUser(@PathVariable Long id) {
        return userService.getUser(id);
    }
}
```

Purpose:

- Handle HTTP requests
- Validate inputs
- Send responses

Key annotations:

- `@RestController`
- `@GetMapping`, `@PostMapping`
- `@RequestBody`, `@PathVariable`

📌 Controllers **DO NOT contain business logic**

5️⃣ Service Layer (Business Logic)

```java
@Service
public class UserService {

    public User getUser(Long id) {
        // business rules
    }
}
```

Why Service Layer exists?

Because:

- Business logic must be **independent**
- Controllers should stay thin
- Easy testing

📌 This is where:

- Validations
- Calculations
- Rules

  happen

6️⃣ Repository Layer (Persistence Layer)

```java
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```

What does it do?

- Talks to database
- Executes queries
- Maps rows to objects

Powered by:

- **JPA**
- **Hibernate**
- **Spring Data JPA**

Java Persistence API
Hibernate
Spring Data JPA

📌 You don't write SQL most of the time — Spring generates it.

7️⃣ Entity Layer (Domain Model)

```java
```

```
@Entity
public class User {

    @Id
    @GeneratedValue
    private Long id;
    private String name;
}
```

Purpose:

- Represents database table
- Maps Java object ↔ DB row

📌 This is **pure data**

No logic.

8️⃣ Dependency Injection (IoC Container)

Spring Boot uses **Inversion of Control (IoC)**.

Example:

```
java

@Service
public class UserService {

    private final UserRepository repo;

    public UserService(UserRepository repo) {
        this.repo = repo;
    }
}
```

What Spring does:

- Creates `UserRepository`
- Injects it into `UserService`
- Manages lifecycle

📌 You NEVER use `new` for Spring beans.

9️⃣ Auto-Configuration (Magic of Spring Boot)

Spring Boot **automatically configures**:

- DataSource
- JPA
- Security
- MVC
- Jackson
- Tomcat

Based on:

- Classpath
- Dependencies
- `application.properties`

📌 This is why Spring Boot is **fast to develop**

🔐 Security Layer (Optional but Common)

Handled by **Spring Security**

Spring Security

Where it sits:

```nginx
Request
 ↓
Security Filters
 ↓
Controller
```

Handles:

- Authentication
- Authorization
- JWT
- OAuth2

⚙️ Configuration Layer

`application.properties`

```properties
server.port=8080
spring.datasource.url=jdbc:mysql://localhost:3306/db
spring.jpa.hibernate.ddl-auto=update
```

Used for:

- DB config
- Server config
- Security config

🧱 Layered Architecture Summary

Layer	Responsibility
Controller	HTTP handling
Service	Business logic
Repository	Database
Entity	Data model
Config	Application setup
Security	Authentication

🔄 Complete Request Flow (Very Important)

Let's trace **one request**:

```pgsql
Client → /users/1
   ↓
Tomcat
```

```
↓
DispatcherServlet
↓
UserController
↓
UserService
↓
UserRepository
↓
Database
↑
Response JSON
```

📌 This flow NEVER changes.

🧠 Why Spring Boot Architecture Is So Powerful

✔️ Separation of concerns
✔️ Easy testing
✔️ Scalable
✔️ Maintainable
✔️ Industry standard

🎯 Final Mental Model (Remember This)

> **Spring Boot Architecture = Layers + Auto-Configuration + Dependency Injection**

If you understand:

- Layers
- Request flow
- Bean management

👉 **You fully understand Spring Boot architecture**

If you want next, I can explain:

- ❓ **How Spring Boot starts internally**

- ❓ **How DispatcherServlet works line by line**
- ❓ **Monolithic vs Microservice architecture in Spring Boot**
- ❓ **How Spring Boot differs from Spring MVC**

Just tell me 👍