RISHABH SOFTWARE

Spring Boot

29-APR-2025

RISHABH Software

2 DECADES OF SERVICE EXCELLENCE

Agenda

1. Introduction to Spring Boot
2. Decoding some terminologies
3. Why should we use frameworks?
4. Differences between Spring and Spring Boot
5. Why choose Spring Boot?
6. Basic Components of a Spring Boot Application
7. REST API Structure
    1. Annotations used for building a web application

Introduction to Spring Boot

1. Spring boot is a **framework** that simplifies creating **stand-alone**, **production-ready** Spring applications.
    1. **Framework**: It is a set of libraries. It ***provides a structure and many pre-built features*** so we ***don't have to start from scratch***.
    2. **Stand alone**: This means the application can run on its own without relying on other systems or setups. In Spring Boot, it ***includes*** everything it needs (like a ***web server***) so it can run independently, just by starting the application.
    3. **Production ready**: Production-ready means the application is prepared to be used by real users in a live environment. It includes features for stability, security, and monitoring, which are essential when deploying it to serve actual users.

2. **Key Features:**
    1. Auto-configuration (based on dependencies),
    2. Embedded servers,
    3. Starter dependencies (added via maven),
    4. and Executable JAR/WAR files.

Decoding some terminologies

1. **Auto-configuration:**
    1. Based on dependencies present in the classpath, spring boot auto-configures beans (objects) and property configurations.
    2. However, *auto-configuration backs away* from the default configuration *if it detects user-configuration*.
    3. This feature *automatically sets up many parts of the application* based on what you have included in your project.
    4. For example, if you add a database dependency, Spring Boot will automatically configure a connection to that database for you. This saves you from writing a lot of setup code, making development faster and easier.
2. **Starter dependencies:**
    1. These are *sets of dependencies* (tools and libraries) that we can *include in our project* with a single line.
    2. For instance, if you want to create a web application, you just add the “spring-boot-starter-web” starter, and it will bring in everything needed for building a web app (like a web server and libraries for handling web requests). This simplifies dependency management and helps avoid compatibility issues.

Why should we use frameworks?

DEV SANJEEV

Uses best readily available best frameworks like Spring, Angular etc. to build a web app

- Leverage Security, Logging etc. from frameworks
- Can easily scale his application
- App will work in an predictable manner
- Focus more on the business logic
- Less efforts and more results/revenue

DEV VICKY

Build his own code by himself to build a web app

- Need to build code for Security, Logging etc.
- Scaling his is not an option till he test everything
- App may not work in an predictable manner
- Focus more on the supporting components
- More efforts and less results/revenue

Differences Between Spring and Spring Boot

Spring	Spring Boot
Requires explicit configuration (XML or Java-based)	Auto-configures most dependencies and settings
Setup can be time-consuming	Bundled with embedded servers (like Tomcat) so setup is quick

Why choose Spring Boot?

1. Advantages

1. Quick setup and reduced boilerplate
2. Production-ready with embedded server options
3. Simplifies dependency management
4. Actuator support for monitoring (e.g., Spring Boot Actuator, DevTools)

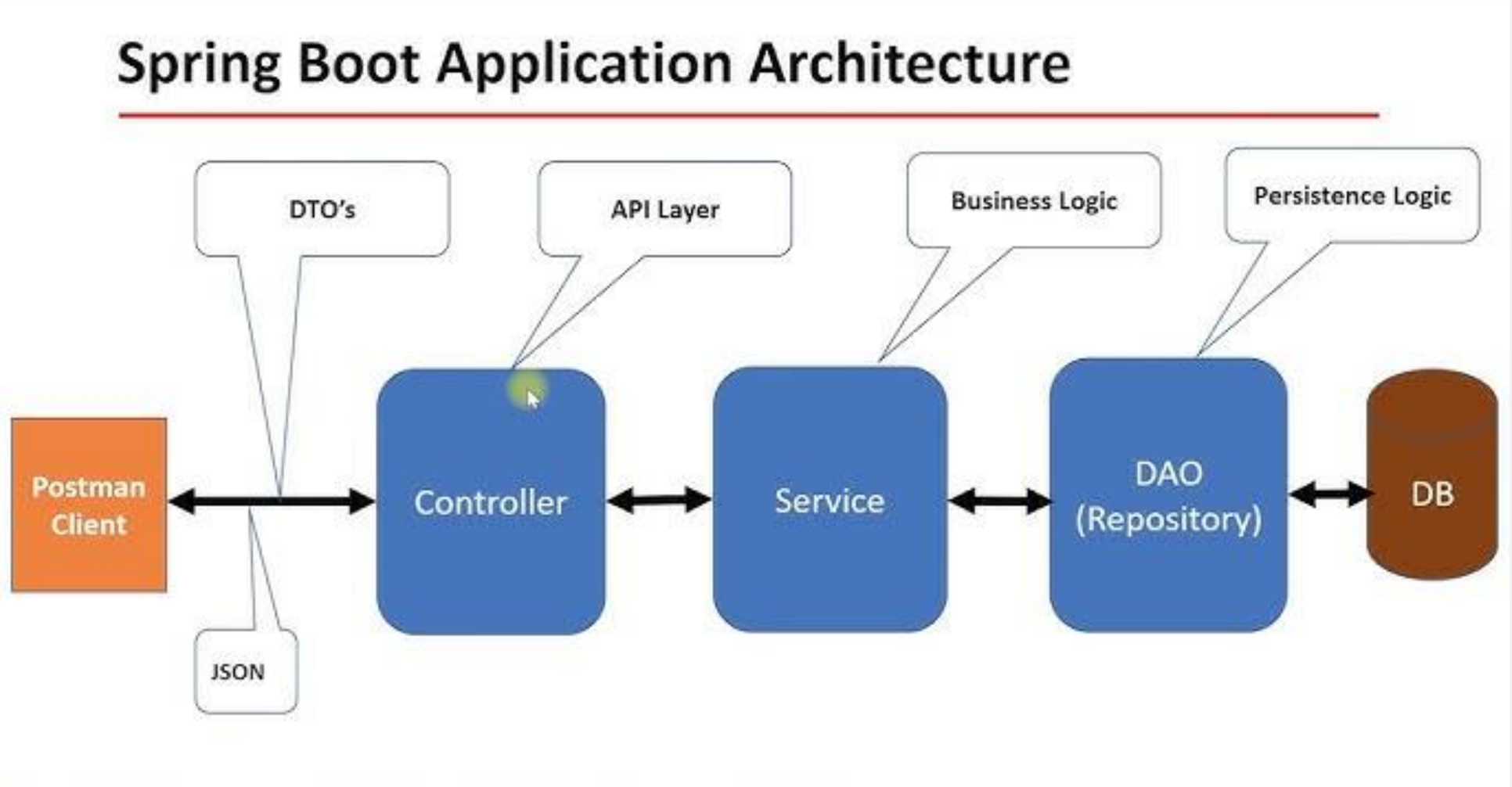
Basic Components of a Spring Boot Application

1. **Main Components:**
    1. **@SpringBootApplication:** Entry point, combines @EnableAutoConfiguration, @ComponentScan, and @Configuration
        1. This annotation ***marks the main class of a Spring Boot application*** and ***serves as the entry point*** when you run the application. It combines three key annotations:
            1. **@EnableAutoConfiguration:** Automatically sets up Spring Boot's default configurations based on what's available in the project.
            2. **@ComponentScan:** Scans the project for components (like services and controllers for beans(objects)) to include in the application context.
            3. **@Configuration:** Marks the class as a source of configuration settings for the application.
        2. Together, @SpringBootApplication makes starting a Spring Boot app simple and sets up everything needed to get it running.
    2. **Application Properties:** Central configuration for the application (in application.properties or application.yml)
    3. **Embedded Server:** Built-in support for Tomcat, Jetty, or Undertow

REST API Structure

1. **Overview of three main layers:**
    1. **Controller:** Handles HTTP requests and responses
    2. **Service:** Business logic layer
    3. **Repository:** Data access layer, interacts with the database
2. **What are annotations?**
    1. Annotations provide **metadata** about our code, allowing us to configure and customize our application.
    2. They are used to simplify and streamline the configuration and management of our Spring Boot application.
3. **Annotations used for building web application:**
    1. **Repository:**
        1. **@Repository:** This annotation marks a class as a "repository," meaning it's responsible for interacting with the database (like saving, updating, and retrieving data).
    2. **Service:**
        1. **@Service:** Used to mark a class as a "service" layer, where the main business logic of the application is handled (like processing data or making decisions).

REST API Structure (contd.)



REST API Structure (contd.)

3. **Annotations used:**
    3. **Controller:**
        1. **@RestController:** This annotation marks a class as a REST controller, which means it ***handles HTTP requests*** and ***sends back HTTP responses***, typically in JSON format.
        2. **@RequestMapping:** Used to ***map web requests to specific methods or classes***, so the application knows which method should handle a given URL.
        3. **@GetMapping:** Specifies that the method will handle HTTP GET requests, typically used for ***reading or retrieving data***.
        4. **@PostMapping:** Indicates that the method will handle HTTP POST requests, commonly used for ***creating new data*** or resources.
        5. **@PutMapping:** Specifies that the method will handle HTTP PUT requests, usually for ***updating existing data***.
        6. **@DeleteMapping:** Indicates that the method will handle HTTP DELETE requests, typically ***used to delete data***.
        7. **@PatchMapping:** Specifies that the method will handle HTTP PATCH requests, often used for ***partially updating data***.

REST API Structure (contd.)

4. Some important HTTP methods and codes:

https://www.javaguides.net/2021/01/rest-api-http-status-codes.html

Sr. No.	API Name	HTTP Method	Path	Status Code	Description
(1)	GET Users	GET	`/api/v1/users`	200 (OK)	All User resources are fetched.
(2)	POST User	POST	`/api/v1/users`	201 (Created)	A new User resource is created.
(3)	GET User	GET	`/api/v1/users/{id}`	200 (OK)	One User resource is fetched.
(4)	PUT User	PUT	`/api/v1/users/{id}`	200 (OK)	User resource is updated.
(5)	DELETE User	DELETE	`/api/v1/users/{id}`	204 (No Content)	User resource is deleted.

We are **TECHNOLOGY EXPERTS** that care

RISHABH SOFTWARE

Thank You

sales@rishabhsoft.com

www.rishabhsoft.com