SPRING BOOT

What is Spring Boot?

Spring Boot is an **open-source Java-based framework** used to create **microservices** and **web applications** quickly with **minimal configuration**. It follows a **convention-over-configuration** approach and comes with **embedded servers**, **auto-configuration**, and **starter dependencies**.

Advantages:

Feature	Description
✅ Auto Configuration	Automatically configures beans based on classpath dependencies.
✅ Standalone	No need for deploying WAR files; runs as a JAR with embedded server.
✅ Embedded Servers	Supports Tomcat, Jetty, or Undertow internally.
✅ Microservice Ready	Ideal for creating REST APIs and microservices.
✅ Starter Templates	Comes with pre-defined dependencies via "starters" (like `spring-boot-starter-web`).
✅ Spring Boot CLI	Allows rapid prototyping with Groovy or Kotlin.
✅ Production Ready	Includes metrics, health checks, and externalized configuration.
✅ Reduced Development Time	Simplifies setup and development using annotations and templates.

Spring Boot Architecture:

Spring Boot architecture is layered and built around the core Spring Framework, but with several key enhancements:

SpringBoot Layers

Client (Browser/Postman)

↓

Controller Layer (REST Controller)

↓

Service Layer (Business Logic)

↓

Repository Layer (Data Access - JPA/Hibernate)

↓

Database (MySQL, PostgreSQL, etc.)

Process of Creating SpringBoot project

Step 1️⃣ : Project Metadata

- **Project**: Maven or Gradle (Choose Maven)
- **Language**: Java
- **Spring Boot Version**: (e.g., 3.2.0 or latest stable)
- **Group**: com.example
- **Artifact**: demo
- **Name**: demo
- **Package Name**: com.example.demo
- **Packaging**: Jar
- **Java Version**: Choose your installed JDK version (17 or 21)

Step 2️⃣ : Add Dependencies

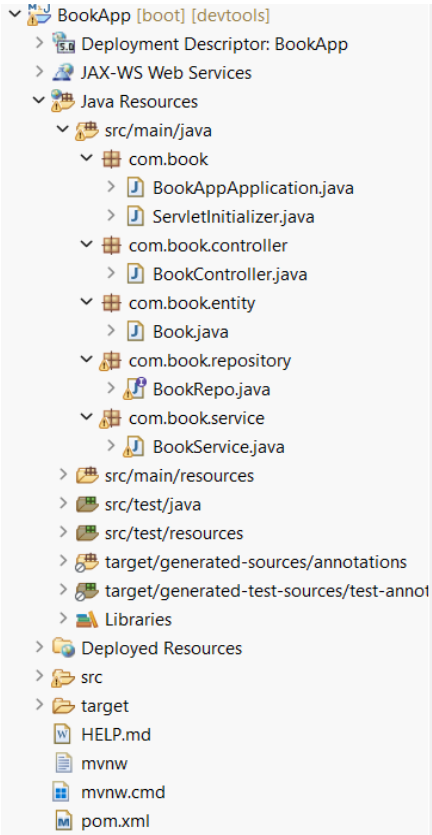
- ✅ Spring Web

- ✅ Spring Boot DevTools (for hot reload)
- ✅ Spring Data JPA (for database)
- ✅ MySQL Driver (or your preferred DB)

Step 3 : Click "Generate"

- This will download a .zip file with the Spring Boot project.
- Extract it and open in **IDE** (IntelliJ IDEA / Eclipse / STS / VS Code).
- ## Sample codes for the Project:

1.Create Packages for every class is inside the Base Package and required Classes(com.book).



- Com.book.controller
  - BookController.java
- Com.book.entity
  - Book.java
- Com.book.repository
  - BookRepo.java(interface)
- Com.book.service
  - BookService.java

2.Add Dependencies(POM.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.5.3</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.book</groupId>
	<artifactId>BookApp</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>war</packaging>
	<name>BookApp</name>
	<description>Demo project for Spring Boot</description>
	<url/>
	<licenses>
		<license/>
	</licenses>
	<developers>
		<developer/>
	</developers>
	<scm>
		<connection/>
		<developerConnection/>
		<tag/>
```

```
        <url/>
    </scm>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
```

```
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

3.Spring boot Properties:

```
spring.application.name=BookApp
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/bookapp
spring.datasource.username=root
spring.datasource.password=tap503
spring.jpa.hibernate.ddl-auto=update
```

4.Sample Codes

- ✓ BookController.java

```java
package com.book.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.book.entity.Book;
import com.book.service.BookService;

@RestController
public class BookController {

	@Autowired
	BookService bs;

	@GetMapping("/getAllbooks")
	public List<Book> getAllbooks()
	{
		return bs.getAllbooks();
	}

	@GetMapping("/getbook/{id}")
	public Book getBook(@PathVariable("id") int bid)
	{
		return bs.getBook(bid);
	}

	@PostMapping("/addbook")
	public String addBook(@RequestBody Book b)
	{
		return bs.addBook(b);
	}

	@PutMapping("/updatebook/{id}")
	public String updateBook(@PathVariable("id") int bid, @RequestBody
	Book b)
	{
		return bs.updateBook(bid,b);
	}

	@DeleteMapping("/deletebook/{id}")
	public String deletebook(@PathVariable("id") int bid)
	{
		return bs.deletebook(bid);
	}


}
```

✓ Book.java

```java
package com.book.entity;
```

```java
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="book")
public class Book {

	@Id
	@Column(name="id")
	private int id;

	@Column(name="name")
	private String name;

	@Column(name="author")
	private String author;

	public Book() {

	}

	public Book(int id, String name, String author) {
		super();
		this.id = id;
		this.name = name;
		this.author = author;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getAuthor() {
		return author;
	}

	public void setAuthor(String author) {
		this.author = author;
	}

	@Override
	public String toString() {
		return "Book [id=" + id + ", name=" + name + ", author=" +
author + "]";
```

```
        }



    }
```

✓ BookRepo.java(Interface)

```
package com.book.repository;
import org.springframework.data.repository.CrudRepository;
import com.book.entity.Book;
import java.util.List;
public interface BookRepo extends CrudRepository<Book,
Integer> {
        Book findById(int bid);
}
```

✓ BookService.java

```
package com.book.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import com.book.entity.Book;
import com.book.repository.BookRepo;

@Component
public class BookService {

        @Autowired
        BookRepo br;
        public List<Book> getAllbooks(){
                List<Book>allBooks = (List<Book>)br.findAll();
                return allBooks;
        }

        public Book getBook(int bid) {
                return br.findById(bid);
        }

        public String addBook(Book b)
        {
                if(br.save(b)!=null)
                {
                        return "Book is added";
                }
                else
                {
                        return"Book is Not Added";
```

```
            }
        }

        public String updateBook(int bid, Book b) {
            if(br.findById(bid) != null)
            {
                br.save(b);
                return "Book id UPDATED";
            }
            else
            {
                return "Book is not UPDATED";
            }
        }

        public String deletebook(int bid) {
            if(br.findById(bid)!=null) {
                br.deleteById(bid);
                return "Book id DELETED";
            }else {
                return "book bot DELETED";
            }

        }
    }
```

2. Why Spring Boot?

Before Spring Boot, you had to:

- Configure XML files for beans and dependencies.
- Manually set up Tomcat/Jetty servers.
- Handle complex dependencies in Spring projects.

Spring Boot solves this by:

- ✅ Auto-configuration → No need to write boilerplate configuration.
- ✅ Embedded Server → No external Tomcat needed; just run like a normal Java program.
- ✅ Starter Dependencies → Provides predefined dependencies for web, JPA, security, etc.
- ✅ Production Ready → Health checks, metrics, logging, security built-in.

3. Features of Spring Boot

1. **Spring Boot Starters** → Predefined dependencies (example: spring-boot-starter-web for web apps).
2. **Auto Configuration** → Automatically configures beans based on classpath.
3. **Embedded Servers** → Tomcat, Jetty, Undertow are included.
4. **Spring Boot CLI** → Command line tool to run Spring Boot apps.
5. **Spring Boot Actuator** → Provides monitoring & management endpoints.
6. **Microservices Ready** → Easy to build REST APIs.

4. Spring Boot Architecture

```
Client → Controller → Service → Repository → Database
```

- **Controller Layer** → Handles HTTP requests (@RestController).
- **Service Layer** → Business logic (@Service).
- **Repository Layer** → Database interaction (@Repository, JPA/Hibernate).

Spring Boot uses **Inversion of Control (IoC)** & **Dependency Injection (DI)** to manage objects.

5. Important Annotations

- @SpringBootApplication → Main class annotation (enables auto-config, component scan, config).
- @RestController → Handles REST APIs.
- @GetMapping, @PostMapping → For API methods.
- @Autowired → Dependency Injection.
- @Entity → Maps class to database table.
- @Repository → DAO layer.
- @Service → Business logic.

6. Example Spring Boot Project

Step 1: Main Class

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

Step 2: Controller

```
@RestController
public class HelloController {

    @GetMapping("/hello")
    public String sayHello() {
        return "Hello, Spring Boot!";
    }
}
```

Run it → Open browser → http://localhost:8080/hello

7. Spring Boot with Database (JPA Example)

Model class:

```
@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
}
```

Repository class:

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long> { }
```

Service class:

```
@Service
public class StudentService {
    @Autowired
    private StudentRepository repo;

    public List<Student> getAllStudents() {
        return repo.findAll();
    }
}
```

Controller class:

```java
@RestController
@RequestMapping("/students")
public class StudentController {
    @Autowired
    private StudentService service;

    @GetMapping
    public List<Student> getStudents() {
        return service.getAllStudents();
    }
}
```

8. Spring Boot Actuator

Add dependency:

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Endpoints:

- /actuator/health → Application health
- /actuator/info → Info

9. Advantages of Spring Boot

- Less code, less configuration.
- Easy integration with databases, security, cloud.
- Microservice ready.
- Faster development (production-ready).

10. Real-World Use Cases

- Building REST APIs
- Microservices Architecture

- E-commerce apps
- Banking & Finance apps
- Cloud-ready applications

✅ In simple words:

👉 **Spring** = Big tool (more power but setup heavy).

👉 **Spring Boot** = Pre-setup + Auto-configured Spring (faster & easier).

Spring Boot Dependencies

1. Core Dependencies:

```
<!-- Spring Boot Starter (core features) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
</dependency>

<!-- Spring Boot Starter Test (JUnit, Mockito for testing) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

spring-boot-starter

👉 Provides core Spring Boot functionality (logging, auto-configuration, YAML, JSON support).

📌 **Use When** → You want the basic features of Spring Boot in any project.

spring-boot-starter-test

👉 Includes **JUnit, Mockito, Hamcrest, AssertJ** → Used for unit & integration testing.

📌 **Use When** → Writing test cases for your APIs or services.

2. Web & REST APIs:

```
<!-- Web (REST APIs, Embedded Tomcat) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<!-- WebFlux (Reactive programming) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>
```

Spring-boot-starter-web

👉 Builds web apps & REST APIs. Includes:

- **Spring MVC**
- **Jackson** (for JSON serialization)
- **Embedded Tomcat** (default server)

📌 **Use When** → You're building REST APIs or web apps.

spring-boot-starter-webflux

👉 Reactive programming support (non-blocking APIs). Uses Netty server instead of Tomcat.

📌 **Use When** → You need high-performance, reactive, event-driven applications.

3. Database & JPA:

```xml
<!-- JPA + Hibernate -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<!-- JDBC (for raw SQL) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<!-- MySQL Driver -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
```

```xml
<!-- H2 Database (in-memory for testing) -->
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

spring-boot-starter-data-jpa

👉 Provides **Spring Data JPA** + Hibernate ORM.
📌 **Use When** → You need CRUD operations with MySQL/Postgres/Oracle DB using JPA.

spring-boot-starter-jdbc

👉 Provides direct **JDBC support** (without Hibernate).
📌 **Use When** → You want to execute **raw SQL queries**.

mysql-connector-j

👉 MySQL driver for database connectivity.
📌 **Use When** → Your DB is **MySQL**.

com.h2database:h2

👉 In-memory DB (auto resets after restart).
📌 **Use When** → Testing or learning Spring Boot without setting up MySQL.

4. Security:

```
<!-- Spring Security -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<!-- JWT (for token-based authentication) -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.11.5</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.11.5</version>
    <scope>runtime</scope>
</dependency>
```

spring-boot-starter-security

👉 Provides authentication, authorization, and password encryption.
📌 **Use When** → You need **login, role-based access, and API security**.

io.jsonwebtoken (jjwt)

👉 For **JWT (JSON Web Token)** implementation.
📌 **Use When** → Securing APIs with token-based authentication.

5. Thymeleaf & Frontend Integration:

```
<!-- Thymeleaf Template Engine -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

spring-boot-starter-thymeleaf

👉 Integrates **Thymeleaf template engine** for server-side HTML rendering.

📌 **Use When** → You want to build **web pages + dynamic HTML** (like JSP alternative).

6. Messaging & Queues:

```
<!-- Spring Boot + Kafka -->
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
</dependency>

<!-- Spring Boot + RabbitMQ -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

spring-kafka

👉 Kafka integration for messaging.

📌 **Use When** → Handling **event-driven applications**.

spring-boot-starter-amqp

👉 RabbitMQ integration.

📌 **Use When** → You need **message queues** for distributed systems.

7. Spring Boot Actuator (Monitoring):

```
<!-- Actuator (health checks, metrics, monitoring) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

spring-boot-starter-actuator

👉 Adds monitoring endpoints (/actuator/health, /info, /metrics).

📌 **Use When** → You want to monitor your app in **production**.

8. Cloud & Microservices:

```
<!-- Eureka Client (Service Discovery) -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>

<!-- Eureka Server -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>

<!-- API Gateway -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>

<!-- Feign Client (service-to-service calls) -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

spring-cloud-starter-netflix-eureka-server

👉 Eureka service registry (server).
📌 **Use When** → You're building **microservices** with service discovery.

spring-cloud-starter-netflix-eureka-client

👉 Registers microservice with Eureka server.
📌 **Use When** → Your app is a **microservice client**.

spring-cloud-starter-gateway

👉 API Gateway (request routing, filtering, load balancing).
📌 **Use When** → You're building **API Gateway** for microservices.

spring-cloud-starter-openfeign

👉 Feign client → makes REST calls between microservices easier.
📌 **Use When** → One microservice calls another (like **Order Service → Product Service**).

9. Dev Tools:

```
<!-- Spring Boot DevTools (auto restart, live reload) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
</dependency>
```

spring-boot-devtools

👉 Provides **auto restart & live reload** for faster development.
📌 **Use When** → You want hot reload while coding.

Quick Reference Table

Category	Dependency (Starter)
Core	spring-boot-starter
Web (REST)	spring-boot-starter-web
Reactive Web	spring-boot-starter-webflux
Database (JPA)	spring-boot-starter-data-jpa
JDBC	spring-boot-starter-jdbc
Security	spring-boot-starter-security
Template Engine	spring-boot-starter-thymeleaf
Messaging (Kafka)	spring-kafka
Messaging (RabbitMQ)	spring-boot-starter-amqp
Monitoring	spring-boot-starter-actuator
Microservices	spring-cloud-starter-* (Eureka, Feign, Gateway)
Testing	spring-boot-starter-test

Category	Dependency (Starter)
DevTools	spring-boot-devtools
spring-boot-starter-amqp	RabbitMQ messaging
spring-boot-starter-actuator	Monitoring & health endpoints
spring-cloud-starter-netflix-eureka-server	Eureka service registry
spring-cloud-starter-netflix-eureka-client	Eureka client (register microservices)
spring-cloud-starter-gateway	API Gateway for microservices
spring-cloud-starter-openfeign	Service-to-service calls
spring-boot-devtools	Auto-restart & live reload

❖ For your **backend projects (Student Mgmt, E-commerce, Food Delivery, etc.)**, you'll mainly use:

- spring-boot-starter-web (APIs)
- spring-boot-starter-data-jpa (DB)
- mysql-connector-j (DB driver)
- spring-boot-starter-test (Testing)
- spring-boot-starter-security + JWT (Security, if needed)
- spring-boot-starter-actuator (Monitoring, if needed)