1.Difference between spring and spring boot?

| **Feature** | **Spring Framework** | **Spring Boot** |
| --- | --- | --- |
| **Setup** | Manual (XML or Java config) | Auto-configured |
| **Project Start** | Slower, more code | Faster, minimal setup |
| **Embedded Server** | ❌ No | ✅ Yes (Tomcat, Jetty, etc.) |
| **Starter Dependencies** | ❌ Must add manually | ✅ Comes with predefined starters |
| **Main Goal** | Flexibility and modular design | Rapid development and simplicity |
| **Configuration** | Complex, verbose | Convention over configuration |
| **Deployment** | Typically WAR files | Runs as standalone JAR |
| **Ideal For** | Large, complex applications | Microservices, quick web/API apps |

**One-line Definitions for MCQs**

* **Spring:** A powerful, flexible Java framework for building applications with manual configuration.
* **Spring Boot:** A simplified, opinionated extension of Spring that enables fast development with minimal setup.

2. Advantage of spring boot

1. **Auto-configuration** – Reduces manual setup.
2. **Standalone apps** – No need for external server (uses embedded Tomcat/Jetty).
3. **Less boilerplate code** – Minimal XML/config files.
4. **Rapid development** – Quick setup using Spring Initializr & starters.
5. **Microservices support** – Easily build and deploy microservices.
6. **Spring Boot Actuator** – Built-in monitoring and health checks.
7. **Easy integration** – With DB, messaging, security, etc.
8. **Full Spring support** – Works with Spring MVC, Security, Data, etc.
9. **Strong community** – Great documentation and support.
10. **Testing support** – Built-in tools for unit and integration testing.

3.What is spring boot data JPA

**Spring Boot Data JPA** is a **part of Spring Data** that simplifies the use of **Java Persistence API (JPA)** for working with relational databases in Spring Boot applications

**📌 Key Features:**

1. **Simplifies database operations** (CRUD) without writing boilerplate code.
2. Uses **interfaces** like JpaRepository or CrudRepository.
3. Automatically implements methods like save(), findById(), delete(), etc.
4. Supports **custom query methods** using method names (e.g., findByName()).
5. Can use **JPQL or native SQL** with @Query annotation.

**🛠️ Example:**

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByUsername(String username);

}

No need to implement this interface — Spring Data JPA generates the logic at runtime.

**✅ Benefits:**

* Reduces boilerplate code
* Fast integration with databases
* Supports pagination and sorting
* Works well with Hibernate (default JPA provider)

4. Advantage of JPA over hibernate and jdbc

**✅ JPA vs JDBC (Advantages of JPA):**

1. 🔹 Less boilerplate code
2. 🔹 Object-oriented (Java classes → DB tables)
3. 🔹 No manual SQL/result mapping
4. 🔹 Easy transaction handling
5. 🔹 Supports relationships (@OneToMany, etc.)
6. 🔹 JPQL – object-based querying
7. 🔹 Portable across DBs and JPA providers

**✅ JPA vs Hibernate (Advantages of JPA):**

1. 🔹 JPA is a standard (vendor-independent)
2. 🔹 Multiple implementations (Hibernate, EclipseLink)
3. 🔹 Cleaner & simpler API
4. 🔹 Works well with Spring Data JPA
5. 🔹 Easier to maintain and test

5.What is web services

A **Web Service** is a **software application** that allows **communication between two devices** over a **network (like the Internet)** using **standard protocols**.

**🔹 Key Points:**

1. 📡 **Platform-independent communication**
2. 🔗 Uses **HTTP**, **XML/JSON**, **SOAP**, or **REST**
3. 🧩 Enables apps to **interact and share data**
4. 🔐 Supports **interoperability** across different systems
5. ⚙️ Can be **consumed** by web, mobile, or desktop clients

**🔧 Types of Web Services:**

* **SOAP (Simple Object Access Protocol)** – XML-based, strict, secure
* **REST (Representational State Transfer)** – Lightweight, uses HTTP methods (GET, POST, etc.)

6. Diff btn rest and soap

| **Feature** | **REST** | **SOAP** |
| --- | --- | --- |
| Protocol | Uses **HTTP** | Uses **SOAP protocol over HTTP/SMTP** |
| Format | **JSON, XML** (mostly JSON) | **XML only** |
| Speed | **Faster** (lightweight) | **Slower** (heavy with XML) |
| Simplicity | Simple to use | More complex |
| Flexibility | Supports multiple formats | Only XML |
| State | **Stateless** | **Can be stateful** |
| Security | Basic (HTTPS) | Advanced (WS-Security) |
| Use Case | Web/Mobile APIs | Enterprise-level, banking, security |

**✅ Summary:**

* **Use REST** for lightweight, fast, and web/mobile-friendly APIs.
* **Use SOAP** for secure, reliable, enterprise-level applications.

7. Advantage of rest api

* 🔹 **Simple & Lightweight** – Uses HTTP methods (GET, POST, etc.)
* 🔹 **Faster** – Supports JSON (lighter than XML)
* 🔹 **Platform Independent** – Works across web, mobile, and desktop
* 🔹 **Scalable** – Easy to scale with multiple clients
* 🔹 **Stateless** – Each request is independent
* 🔹 **Easy to Learn & Use** – URL-based, readable structure
* 🔹 **Supports Multiple Formats** – JSON, XML, plain text, etc.
* 🔹 **Better Performance** – Caching possible using HTTP headers

8.Explain layers architecture in springboot

**✅ Layered Architecture in Spring Boot (Short Notes)**

Spring Boot follows a **4-layered architecture**:

**1. Presentation Layer (Controller)**

* Handles **HTTP requests & responses**
* Uses @RestController, @RequestMapping, etc.
* Talks to **Service Layer**

🧩 Example: UserController.java

**2. Service Layer**

* Contains **business logic**
* Uses @Service
* Calls **Repository Layer**

🧩 Example: UserService.java

**3. Repository/Data Access Layer**

* Handles **database operations**
* Uses Spring Data JPA (@Repository, JpaRepository)
* Talks to **Database**

🧩 Example: UserRepository.java

**4. Model Layer**

* Defines **data objects/entities**
* Uses @Entity
* Mapped to database tables

🧩 Example: User.java

**🔁 Flow Summary:**

**Client → Controller → Service → Repository → DB**

9.Write all the validation in spring boot

Use these annotations in your **model/entity class** with @Valid in controller.

* **🔹 Field-Level Validations**

| **Annotation** | **Purpose** |
| --- | --- |
| @NotNull | Field must not be null |
| @NotEmpty | Field must not be empty (Strings, Collections) |
| @NotBlank | Field must not be blank (for Strings only) |
| @Size(min, max) | Check length of string/collection |
| @Min(value) | Minimum numeric value |
| @Max(value) | Maximum numeric value |
| @Email | Valid email format |
| @Pattern(regex) | Match custom regex |
| @Positive | Must be a positive number |
| @Negative | Must be a negative number |
| @Future | Date must be in the future |
| @Past | Date must be in the past |
| @Digits(int, fraction) | Number of digits before and after decimal |
| @AssertTrue / @AssertFalse | Must be true / false |

**🔹 Example Usage:**

* @Entity
* public class User {
* @NotNull
* private Long id;
* @NotBlank
* private String name;
* @Email
* private String email;
* @Size(min = 8, max = 20)
* private String password;
* @Min(18)
* @Max(60)
* private int age;
* }
* **🔹 In Controller:**
* @PostMapping("/register")
* public ResponseEntity<?> register(@Valid @RequestBody User user, BindingResult result) {
* if (result.hasErrors()) {
* return ResponseEntity.badRequest().body(result.getAllErrors());
* }
* return ResponseEntity.ok("User registered");
* }

10.How to handle the Globle exception handling in spring boot

**✅ Global Exception Handling in Spring Boot**

Spring Boot provides a clean way to handle exceptions globally using @ControllerAdvice.

**🔹 Steps to Handle Global Exceptions:**

1. **Create a Global Exception Handler Class**

import org.springframework.web.bind.annotation.\*;

import org.springframework.http.\*;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)

public ResponseEntity<String> handleNotFound(ResourceNotFoundException ex) {

return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT\_FOUND);

}

@ExceptionHandler(Exception.class)

public ResponseEntity<String> handleGeneral(Exception ex) {

return new ResponseEntity<>("Something went wrong!", HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

1. **Create a Custom Exception (Optional)**

public class ResourceNotFoundException extends RuntimeException {

public ResourceNotFoundException(String message) {

super(message);

}

}

1. **Use Custom Exception in Service/Controller**

if (user == null) {

throw new ResourceNotFoundException("User not found");

}

**✅ Benefits:**

* Centralized error handling
* Clean and reusable
* Returns proper HTTP status codes and messages

11. How to create a custom exception handling   
**✅ Custom Exception Handling in Spring Boot – Quick Steps**

**🔹 1. Create a Custom Exception Class**

public class UserNotFoundException extends RuntimeException {

public UserNotFoundException(String message) {

super(message);

}

}

**🔹 2. Throw the Exception in Service/Controller**

public User getUserById(Long id) {

return userRepository.findById(id)

.orElseThrow(() -> new UserNotFoundException("User not found with ID: " + id));

}

**🔹 3. Handle It in @ControllerAdvice Class**

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(UserNotFoundException.class)

public ResponseEntity<String> handleUserNotFound(UserNotFoundException ex) {

return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT\_FOUND);

}

}

**✅ Output Example:**

If the user is not found, API returns:

{

"status": 404,

"error": "User not found with ID: 5"

}

12. Explain why we use application property file   
**✅ Why We Use application.properties (or application.yml) in Spring Boot**

**🔹 Purpose:**

To **externalize configuration** so we can manage settings **without changing code**.

**🔹 Common Uses:**

1. **Database Configuration**

spring.datasource.url=jdbc:mysql://localhost:3306/dbname

spring.datasource.username=root

spring.datasource.password=pass

1. **Server Settings**

server.port=8081

1. **JPA Settings**

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

1. **Custom Values**

app.name=MySpringApp

Access in code with:

@Value("${app.name}")

private String appName;

**✅ Benefits:**

* 🔄 Easy to change without recompiling
* 🧪 Different profiles (dev, test, prod)
* 🔐 Centralized config (clean & manageable)
* 💡 Supports both .properties and .yml

13. What is the Pom . XML file   
**✅ What is pom.xml in Spring Boot (or Maven Projects)?**

pom.xml stands for **Project Object Model** and is the **core configuration file** for **Maven-based projects** like Spring Boot.

**🔹 Purpose of pom.xml:**

1. 📦 **Manages Dependencies**  
   Add libraries like Spring Boot, MySQL, Hibernate, etc.
2. ⚙️ **Build Configuration**  
   Specifies how the project is built, packaged, and run.
3. 🧩 **Plugin Management**  
   Used for tools like compiler, surefire (tests), Spring Boot plugin, etc.
4. 🔁 **Version Control**  
   Define version of Java, Spring Boot, and other dependencies.

**🔹 Example pom.xml:**

<project>

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.1.0</version>

</parent>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Add more dependencies here -->

</dependencies>

</project>

**✅ In Short:**

pom.xml tells Maven **what the project is, what it needs**, and **how to build it**.

14. What is maven ? Why we use ?

**Maven** is a **build automation and dependency management tool** used mainly for **Java projects**.

**✅ Why We Use Maven (Key Points):**

1. 📦 **Dependency Management**  
   Automatically downloads required libraries (like Spring Boot, MySQL).
2. ⚙️ **Build Automation**  
   Compiles code, runs tests, and packages the app (e.g., into a .jar).
3. 📁 **Standard Project Structure**  
   Follows a common directory layout (src/main/java, src/test/java, etc.).
4. 🔁 **Reusable Configuration**  
   Centralized settings in pom.xml — easy to maintain.
5. 📤 **Easy Integration**  
   Works with IDEs (Eclipse, IntelliJ), CI/CD tools, and plugins.
6. 🚀 **Simplifies Project Setup**  
   Quick project creation using archetypes.

**Maven** helps you **build, manage, and share** Java projects efficiently with minimal manual work.