



UNIVERSITY OF SRI JAYEWARDENEPURA
Faculty of Technology
Department of Information and Communication
Technology

**ITS 4243 - Microservices and Cloud Computing
Assignment 01**

Part 1

1. What is Spring Boot and why is it used?

Spring Boot is an open-source framework developed by Pivotal (VMware) to simplify the process of building and deploying Spring-based Java applications.

It is built on top of the Spring Framework and helps developers create production-ready, stand-alone applications quickly and with minimal configuration.

Key Features:

- **Auto-Configuration:** Automatically configures application components based on dependencies.
- **Embedded Servers:** Supports built-in servers like Tomcat, Jetty, and Undertow, so there is no need to deploy WAR files.
- **Starter Dependencies:** Provides pre-configured starter packages like spring-boot-starter-web, spring-boot-starter-data-jpa, etc.
- **Actuator:** Gives production-ready features such as health checks and metrics.
- **Spring Initializr:** Offers an easy web-based project setup tool.

Why it is used:

- Reduces boilerplate configuration.
- Speeds up development and testing.
- Provides an opinionated setup (predefined configurations) for best practices.
- Makes deployment simple applications can run with a single command:

Command : java -jar appname.jar

2. Explain the difference between Spring Framework and Spring Boot

Feature	Spring Framework	Spring Boot
Setup	Requires complex XML or Java-based configuration.	Minimal configuration due to auto-configuration.
Dependency Management	Developer must manually define dependencies.	Uses pre-defined starter dependencies.
Server Setup	Needs an external server (Tomcat/Jetty).	Comes with an embedded server.
Deployment	Typically deployed as a WAR file.	Packaged as a self-contained JAR.
Focus	Provides flexibility and modularity.	Focuses on rapid development and ease of deployment.
Learning Curve	Steeper due to manual configurations.	Easier to start for beginners.

Example:

In Spring, developers had to configure `dispatcher-servlet.xml`, `applicationContext.xml`, etc. In Spring Boot, only annotations like `@SpringBootApplication` are needed.

3. What is Inversion of Control (IoC) and Dependency Injection (DI)?

Inversion of Control (IoC):

It is a design principle where the control of object creation and lifecycle management is given to a container (Spring IoC Container) rather than being handled by the developer. This helps in loose coupling between components.

Dependency Injection (DI):

A technique used to implement IoC. Instead of a class creating its own dependencies using new, the Spring container injects dependencies automatically.

Example:

```
@Component
public class StudentService {
    private final StudentRepository repo;

    @Autowired
    public StudentService(StudentRepository repo) {
        this.repo = repo;
    }
}
```

Here, Spring injects an instance of **StudentRepository** automatically into **StudentService**.

Benefits:

- Loose coupling.
- Easier testing and maintenance.
- Improved flexibility and scalability.

4. What is the purpose of application.properties / application.yml?

Both files are used for external configuration in a Spring Boot application. They contain key-value pairs that define application settings.

Examples:

application.properties

```
1 server.port=8081
2 spring.datasource.url=jdbc:mysql://localhost:3306/mydb
3 spring.datasource.username=root
4 spring.datasource.password=1234
5
```

application.yml

```
server:
| port: 8081
spring:
| datasource:
| | url: jdbc:mysql://localhost:3306/mydb
| | username: root
| | password: 1234
```

Purpose:

- Centralized configuration management.
- Allows environment-specific settings (dev, test, prod).
- Avoids hardcoding sensitive information in the source code.

5. Explain what a REST API is and list HTTP methods used.

REST (Representational State Transfer) is an architectural style for designing networked applications.

A REST API allows communication between client and server using standard HTTP methods.

Key Principles:

- Stateless communication (no session info stored on the server).
- Resource-based (each resource has a unique URL).
- Uses standard HTTP operations.

Common HTTP Methods:

Method	Purpose	Example
GET	Retrieve data	/api/users
POST	Create new data	/api/users
PUT	Update entire record	/api/users/1
PATCH	Partially update record	/api/users/1
DELETE	Delete data	/api/users/1

Example REST Controller:

```
@RestController
@RequestMapping("/api/users")
public class UserController {
    @GetMapping
    public List<User> getUsers() {
        return userService.getAllUsers();
    }
}
```

6. What is Spring Data JPA? What is an Entity and a Repository?

Spring Data JPA is a part of the Spring ecosystem that simplifies data access using the Java Persistence API (JPA).

It removes the need for boilerplate code for CRUD operations.

Entity:

A class annotated with `@Entity` that represents a table in a database.

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

Repository:

An interface that provides data access functionality.

```
@Repository  
public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
}
```

Spring automatically creates implementation for CRUD methods.

7. Difference between `@Component`, `@Service`, `@Repository`, `@Controller`, `@RestController`

Annotation	Layer	Description
<code>@Component</code>	General	Marks a generic bean managed by Spring.
<code>@Service</code>	Business	Used in the service layer for business logic.
<code>@Repository</code>	Data Access	Used in the DAO layer, integrates with database exceptions.
<code>@Controller</code>	Presentation	Handles web requests and returns views (HTML).
<code>@RestController</code>	Presentation (API)	Combines <code>@Controller</code> and <code>@ResponseBody</code> , returns JSON responses for REST APIs.

Example:

```
@Service  
public class ProductService { ... }  
  
@Repository  
public interface ProductRepository extends JpaRepository<Product, Long> { ... }  
  
@RestController  
public class ProductController { ... }  
|
```

8. What is `@Autowired`? When should we avoid it?

`@Autowired` is used to automatically inject dependencies into a class. Spring searches the IoC container for a suitable bean and injects it.

Example:

```
@Autowired  
private ProductService productService;  
|
```

Avoid when:

- Using field injection, as it makes testing difficult and violates immutability.
- Prefer constructor-based injection, which is safer and more testable:

```
public ProductController(ProductService productService) {  
    this.productService = productService;  
}  
|
```

9. Explain how Exception Handling works in Spring Boot (@ControllerAdvice)

Spring Boot provides a centralized exception-handling mechanism using the **@ControllerAdvice** annotation.

It allows developers to handle exceptions globally across all controllers.

Example:

```
1  @ControllerAdvice
2  public class GlobalExceptionHandler {
3
4      @ExceptionHandler(ResourceNotFoundException.class)
5      public ResponseEntity<String> handleNotFound(ResourceNotFoundException ex) {
6          return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT_FOUND);
7      }
8
9      @ExceptionHandler(Exception.class)
10     public ResponseEntity<String> handleGlobal(Exception ex) {
11         return new ResponseEntity<>("Something went wrong!", HttpStatus.INTERNAL_SERVER_ERROR);
12     }
13 }
14 |
```

Benefits:

- Centralized error handling.
- Cleaner controller code.
- Custom error messages and HTTP responses.

10. What is the role of Maven/Gradle in a Spring Boot project?

Maven and Gradle are build automation and dependency management tools used in Java projects.

Roles in Spring Boot:

1. Dependency Management: Automatically downloads and manages JAR files required for the project.

Example (Maven pom.xml):

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
|
```

2. Build Automation: Compiles, tests, packages, and deploys applications using simple commands:

mvn clean install

3. Project Structure: Maintains consistent folder and dependency structure.
4. Plugins: Integrates tools for testing, reporting, and packaging.
5. Gradle: Provides faster builds using a Groovy or Kotlin DSL (build.gradle).