# **Maven, Spring and Spring Boot**

**Please share your output screenshots in the assignment document along with the github link for each question. Provide an explanation wherever possible as part of your response :-)**

| 1. **Install maven 3.6 or above. Execute mvn -v in the local terminal/command prompt and share the screenshot**      1. **What is the difference between maven central repository and local repository?**  * Local Repository is a folder on the developer's machine (~/.m2/repository) where Maven stores downloaded libraries to avoid redownloading. * Maven Central Repository is a public online repository (<https://repo.maven.apache.org>) from where Maven downloads dependencies if not available locally. * Maven always checks the local repo first, then the central repo if needed.  1. **Maven commands**    1. **To build the maven project:** mvn clean install    2. **To run the maven tests:** mvn test 2. **Please locate the maven settings.xml file and local maven repository in your machine and share the screenshot**        1. **The basic principle behind Dependency Injection(DI) is that the objects define their dependencies .What are the different ways in which an object can define its dependency ?**   Dependency Injection (DI) allows objects to define their dependencies externally, improving flexibility and testability.  Spring supports three types of dependency injection:   1. Constructor Injection – Inject dependencies through constructors (recommended). 2. Setter Injection – Inject using setter methods, good for optional dependencies. 3. Field Injection – Inject directly into fields using **@Autowired**, though not recommended for production code.   Constructor injection is preferred because it makes the dependency requirements explicit and supports immutability.   1. **What is the difference between the @Autowired and @Inject annotation?**   **@Autowired** is a Spring-specific annotation used to inject dependencies automatically.  **@Inject** is a JSR-330 standard annotation from javax.inject, supported by Spring for compatibility.   * **@Autowired** provides additional Spring features like required=false, @Qualifier, and is more flexible for Spring apps. * **@Inject** is useful when you want to write framework-independent or portable code.   Both achieve dependency injection, but **@Autowired** is more common in Spring applications.   1. **Explain the use of @Respository, @Component, @Service and @Controller annotations with an example for each.**   Spring provides stereotype annotations like @Component, @Service, @Repository, and @Controller to define application layers:   * @Component – Generic Spring-managed bean (utility/helper classes). * @Repository – Indicates DAO classes; adds exception translation for database interactions. * @Service – Marks business logic layer; holds service logic. * @Controller – Used in Spring MVC to handle web requests and return views or JSON/XML.   These annotations help Spring auto-detect and register beans during component scanning. **@Component Example** @Component  public class EmailUtility {  public void sendEmail(String to) {  System.out.println("Sending email to " + to);  }  }  **@Repository Example**  @Repository  public class UserRepository {  public User findById(Long id) {  // Simulate DB fetch  return new User(id, "Anushka");  }  }  **@Service Example**  @Service  public class UserService {  @Autowired  private UserRepository userRepository;  public User getUser(Long id) {  return userRepository.findById(id);  }  }  **@Controller Example**  @Controller  public class UserController {  @Autowired  private UserService userService;  @GetMapping("/user/{id}")  @ResponseBody  public User getUser(@PathVariable Long id) {  return userService.getUser(id);  }  }   1. **Fix the code and explain why?**   **The following code tries to inject a property from application.properties, but the appName field is always null. Identify and fix the issue.**  **@Component**  **public class AppNamePrinter {**  **@Value("app.name")**  **private String appName;**    **public void printAppName() {  System.out.println("Application Name: " + appName);  }  }**  The line *@Value("app.name")* is incorrect because @Value expects ${} syntax to read from application.properties. Writing "app.name" injects the literal string app.name, not the value from the properties file.  **Corrected code:**  @Component  public class AppNamePrinter {  @Value("${app.name}") // Use ${} to fetch from application.properties  private String appName;  public void printAppName() {  System.out.println("Application Name: " + appName);  }  }   1. **What does the @SpringBootApplication annotation do?**  It is a convenience annotation that combines three core Spring annotations: @SpringBootApplication  // is equivalent to:  @Configuration: Marks the class as a source of bean definitions (like beans.xml)  @EnableAutoConfiguration: Tells Spring Boot to automatically configure the application based on dependencies on the classpath  @ComponentScan: Tells Spring to scan the package (and sub-packages) for components like @Component, @Service, @Repository, @Controller   1. **What is the maven command to start the SpringBootApplication?**   To run a Spring Boot application using Maven, use the following command:  *mvn spring-boot:run*   1. **Implement EmployeeCRUD using Spring and JDBC with the below Employee class. In the branch feature-spring, create a folder Employee-Spring. Push the solution to the branch and share the link.**   **class Employee{**  **private int id;**  **private String name;**  **private String department;**  **}**  **Github link:** <https://github.com/ANUSHKA1509/rg-assignments/tree/feature-spring/EmployeeSpringJDBC>  Results:    In MySQL Database: |
| --- |
|  |

1. **Implement EmployeeCRUD using SpringBoot and Spring Data JPA with the below Employee class. In the branch feature-spring, create a folder Employee-SpringBoot-JPA. Push the solution to the branch and share the link.**

**class Employee{**

**private int id;**

**private String name;**

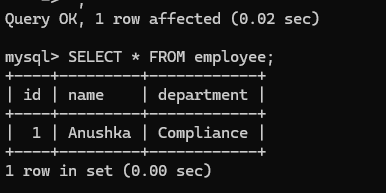
**private String department;**

**}**

**Github link:** <https://github.com/ANUSHKA1509/rg-assignments/tree/feature-spring/Employee-SpringBoot-JPA>

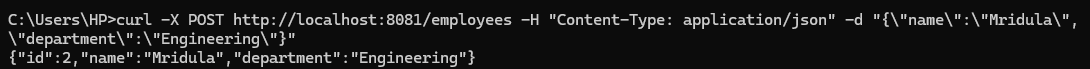
Results:

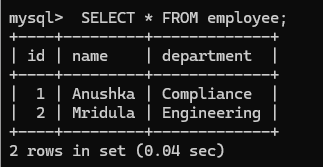
Before adding an employee:

****

****

Adding an employee using curl:





In this question, we are building and testing a simple Spring Boot CRUD application using Spring Boot + JPA (Hibernate). The focus is on verifying that the Employee REST API works end-to-end. Specifically, we test:

* Whether the application starts correctly using Maven (mvn clean install and mvn spring-boot:run),
* Whether the @RestController exposes the correct endpoints,
* Whether HTTP requests (like POST to add an employee and GET to retrieve them) are properly handled,
* Whether the application can parse JSON payloads, save data using JPA, and return appropriate responses.

This confirms that Spring Boot, REST, and JPA integrations are set up and working as expected.

1. **Follow the demo in the pre-work link** [**https://www.youtube.com/watch?v=hr2XTbKSdAQ&t=18s**](https://www.youtube.com/watch?v=hr2XTbKSdAQ&t=18s) **and create a Spring Batch application that processes customer data. In the branch feature-spring, create a folder Customer-SpringBatch. Push the solution to the branch and share the link.**

**Github link:** <https://github.com/ANUSHKA1509/rg-assignments/tree/feature-spring/Customer-SpringBatch>

This task involves creating a Spring Batch application that reads customer data from a CSV file, processes it (e.g., capitalizes names), and writes it to a database.