| **Exercise 1: Basic Unit Test for a Service Method** **Task:**  Write a unit test for a **Spring service method** that adds two numbers.  **Output:**    **Exercise 2: Testing a Spring Service Using a Mock Repository**  **Scenario:** You have a UserService that fetches user info from a UserRepository.  You’ll write a test to mock the repository and verify the service logic works.  **Output:**    **Exercise 3: Testing a REST Controller with MockMvc**  Task: Test a controller endpoint that returns a user.  **Output:**    **Exercise 4: Integration Test with Spring Boot**  Task: Write an integration test that tests the full flow from controller to database.  **Output:**    **Exercise 5: Test Controller POST Endpoint**  Task: Test a POST endpoint that creates a user.  Controller: @PostMapping public ResponseEntity createUser(@RequestBody User user)  { return ResponseEntity.ok(userService.saveUser(user)); }  **Output:**    **Exercise 6: Test Service Exception Handling**  Task: Test how a service handles a missing user.  **Output:**    **Exercise 7: Test Custom Repository Query**  Task: Add and test a custom query method.  **Output:**    **Exercise 8: Test Controller Exception Handling**  Task:  Add and test a @ControllerAdvice for handling exceptions. Exception Handler: @ControllerAdvice public class GlobalExceptionHandler { @ExceptionHandler(NoSuchElementException.class) public ResponseEntity handleNotFound(NoSuchElementException ex) { return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("User not found"); } }  **Output:**    **Exercise 9: Parameterized Test with JUnit**  Task: Use @ParameterizedTest to test multiple inputs.  **Output:** |
| --- |