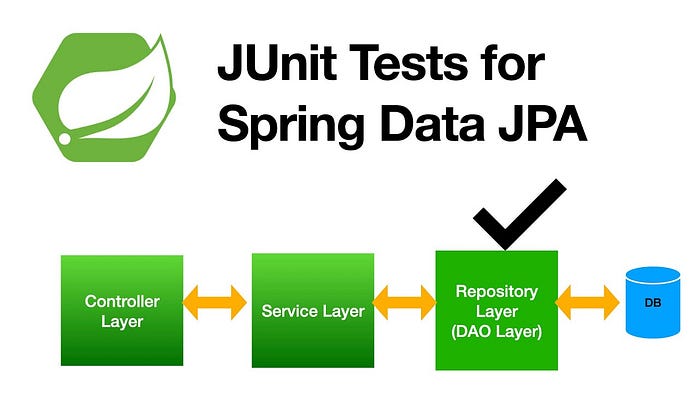
# Spring Boot Unit Testing for Repository Layer

https://mohosinmiah1610.medium.com/spring-boot-unit-testing-repository-layer-492bd004d417



**@DataJpaTest @BeforeEach**

**Spring Boot — Unit Testing Repository Layer**

In this blog describe how to to perform unit test for repository layer. Well organize coding example is given is the below **GitHub**repository.Please check out the repository.

**[GitHub - MohosinMiah/Spring-Boot--Unit-Testing-Repository-Layer: Spring Boot - Unit Testing…](https://github.com/MohosinMiah/Spring-Boot--Unit-Testing-Repository-Layer?source=post_page-----492bd004d417--------------------------------" \t "_blank)**

[Spring Boot - Unit Testing Repository Layer. Contribute to MohosinMiah/Spring-Boot--Unit-Testing-Repository-Layer…](https://github.com/MohosinMiah/Spring-Boot--Unit-Testing-Repository-Layer?source=post_page-----492bd004d417--------------------------------" \t "_blank)

[github.com](https://github.com/MohosinMiah/Spring-Boot--Unit-Testing-Repository-Layer?source=post_page-----492bd004d417--------------------------------" \t "_blank)

**Test Case Structure**

When writing test cases, it is beneficial to follow the “**given-when-then**” structure to provide clear and concise test descriptions. Here’s an example:

@Test  
public void given\_when\_then() {  
 // Given: Setup object or precondition  
   
 // When: Action or behavior that we are going to test  
   
 // Then: Verify the output or expected result  
}

**Annotation Details**

[**@DataJpaTest**](http://twitter.com/DataJpaTest)

The [**@DataJpaTest**](http://twitter.com/DataJpaTest) annotation in Spring Boot is used for integration testing of JPA repositories. It provides a convenient way to test the persistence layer of your application by automatically configuring an in-memory database, creating a TestEntityManager, and setting up a transactional test environment. It focuses on loading the repository layer only.

[**@BeforeEach**](http://twitter.com/BeforeEach)

The [**@BeforeEach**](http://twitter.com/BeforeEach) annotation in JUnit 5 is used to signal that a method should be executed before each test method in a test class. It allows you to set up the necessary preconditions or perform any common initialization steps before running each individual test.

**For Repository**

Create a **folder**repository in main -> java and inside create folder repository and EmployeeRepository.javaclass .

**EmployeeRepository.java**

@Repository  
public interface EmployeeRepository extends JpaRepository<Employee,Long>{  
 Optional<Employee> findByEmail(String email);  
  
 // Define custom query using JPQL with index params  
 @Query("select e from Employee e where e.firstName = ?1 and e.lastName = ?2")  
 Employee findByJPQL(String firstName, String lastName);  
   
 // Define custom query using JPQL with name params  
 @Query("select e from Employee e where e.firstName = :firstName and e.lastName = :lastName")  
 // Employee findByJPQLNameParams(String firstName, String lastName); // This is also works fine because params name and parameter name is sanme  
 Employee findByJPQLNameParams(@Param("firstName") String firstName, @Param("lastName") String lastName);  
  
 // Define custom query native sql query with index param  
 @Query(value= "select \* from employees e where e.first\_name = ?1 and e.last\_name = ?2", nativeQuery = true )  
 Employee findByNativeSQL(String firstName, String lastName);  
  
  
 // Define custom query native sql query with name param  
 @Query(value= "select \* from employees e where e.first\_name =:firstName and e.last\_name = :lastName", nativeQuery = true )  
 Employee findByNativeSQLNameParam(@Param("firstName") String firstName, @Param("lastName") String lastName);  
}

**For Test class**

Create a **folder**repository and inside create EmployeeRepositoryTests.javaclass .

**EmployeeRepositoryTests.java**

@DataJpaTest  
public class EmployeeRepositoryTests {  
   
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 private Employee employee;  
  
  
 @BeforeEach  
 public void setupTestData(){  
 // Given : Setup object or precondition  
 employee = Employee.builder()  
 .firstName("MOHOSIN")  
 .lastName("MIAH")  
 .email("mohosinmiah1610@gmail.com")  
 .departmentCode("CSE")  
 .build();  
 }  
  
 // JUnit Test for save employee operation  
 @Test  
 @DisplayName("JUnit test for save employee operation")  
 public void givenEmployeeObject\_whenSave\_thenReturnSaveEmployee(){  
  
  
 // When : Action of behavious that we are going to test  
 Employee saveEmployee = employeeRepository.save(employee);  
  
 // Then : Verify the output  
  
 assertThat(saveEmployee).isNotNull();  
 assertThat(saveEmployee.getId()).isGreaterThan(0);  
  
 }  
  
 // JUnit test for get Employee List operation  
 @Test  
 @DisplayName("JUnit test for get Employee List")  
 public void givenEmployeeList\_whenFindAll\_thenEmployeeList(){  
  
 // Given : Setup object or precondition  
 Employee employeeOne = Employee.builder()  
 .firstName("MOHOSIN One")  
 .lastName("MIAH One")  
 .email("mohosinmiah1610@gmail.com")  
 .departmentCode("CSE")  
 .build();  
  
 Employee employeeTwo = Employee.builder()  
 .firstName("MOHOSIN Two")  
 .lastName("MIAH Two")  
 .email("mohosinmiah1610@gmail.com")  
 .departmentCode("CSE")  
 .build();  
  
 employeeRepository.save(employeeOne);  
 employeeRepository.save(employeeTwo);  
  
 // When : Action of behavious that we are going to test  
 List<Employee> employees = employeeRepository.findAll();  
  
 // Then : Verify the output  
 assertThat(employees).isNotEmpty();  
 assertThat(employees.size()).isEqualTo(2);  
 }  
  
 // JUnit test for get Employee By Id operation  
   
 @Test  
 @DisplayName("JUnit test for get Employee By Id")  
 public void givenEmployeeObject\_whenFindById\_thenReturnEmployeeObject()  
 {  
 // Given : Setup object or precondition  
 employeeRepository.save(employee);  
  
 // When : Action of behavious that we are going to test  
 Employee getEmployee = employeeRepository.findById(employee.getId()).get();  
  
 // Then : Verify the output  
 assertThat(getEmployee).isNotNull();  
 }  
  
  
 // JUnit test for get employee by email operation  
 @Test  
 @DisplayName("JUnit test for get employee by email operation")  
 public void givenEmployeeEmail\_whenFindByEmail\_thenEmployeeObject() {  
  
 // Given: Setup object or precondition  
  
 employeeRepository.save(employee);  
  
 // When: Action or behavior that we are going to test  
 Employee getEmployee = employeeRepository.findByEmail("mohosinmiah1610@gmail.com").get();  
  
 // Then: Verify the output or expected result  
 assertThat(getEmployee).isNotNull();  
 assertThat(getEmployee.getEmail()).isEqualTo("mohosinmiah1610@gmail.com");  
 }  
  
  
 // JUnit test for get employee update operation  
 @Test  
 @DisplayName("JUnit test for get employee update operation")  
 public void givenEmployeeObject\_whenUpdate\_thenEmployeeObject() {  
  
 // Given: Setup object or precondition  
  
 employeeRepository.save(employee);  
  
 // When: Action or behavior that we are going to test  
 Employee getEmployee = employeeRepository.findById(employee.getId()).get();  
  
 getEmployee.setFirstName("MOHOSINE UPDATE");  
 getEmployee.setLastName("Last Name");  
 getEmployee.setEmail("update@gmail.com");  
 getEmployee.setDepartmentCode("EEE");  
  
 Employee updatedEmployee = employeeRepository.save(getEmployee);  
  
 // Then: Verify the output or expected result  
 assertThat(updatedEmployee).isNotNull();  
 assertThat(updatedEmployee.getEmail()).isEqualTo("update@gmail.com");  
 }  
  
  
 // JUnit test for delete employee operation  
 @Test  
 @DisplayName("JUnit test for delete employee operation")  
 public void givenEmployeeObject\_whenDelete\_thenRemoveEmployee() {  
  
 // Given: Setup object or precondition  
  
 employeeRepository.save(employee);  
  
 // When: Action or behavior that we are going to test  
 employeeRepository.deleteById(employee.getId());  
 Optional<Employee> deleteEmployee = employeeRepository.findById(employee.getId());  
  
 // Then: Verify the output or expected result  
 assertThat(deleteEmployee).isEmpty();  
 }  
  
  
 // JUnit test for custom query using JPQL with index param   
 @Test  
 @DisplayName("JUnit test for custom query using JPQL with index")  
 public void givenFirstNameAndLastName\_whenFindByJPQL\_thenReturnEmployeeObject() {  
  
 // Given: Setup object or precondition  
  
 employeeRepository.save(employee);  
  
 // When: Action or behavior that we are going to test  
 Employee getEmployee = employeeRepository.findByJPQL(employee.getFirstName(), employee.getLastName());  
  
 // Then: Verify the output or expected result  
 assertThat(getEmployee).isNotNull();  
 }  
  
 // JUnit test for custom query using JPQL with name param   
 @Test  
 @DisplayName("JUnit test for custom query using JPQL with name param ")  
 public void givenFirstNameAndLastName\_whenFindByJPQLNameParams\_thenReturnEmployeeObject() {  
  
 // Given: Setup object or precondition  
  
 employeeRepository.save(employee);  
  
 // When: Action or behavior that we are going to test  
 Employee getEmployee = employeeRepository.findByJPQLNameParams(employee.getFirstName(), employee.getLastName());  
   
 // Then: Verify the output or expected result  
 assertThat(getEmployee).isNotNull();  
 }  
  
  
 // JUnit test for custom query using findByNativeSQL with name index   
 @Test  
 @DisplayName("JUnit test for custom query using findByNativeSQL with name param ")  
 public void givenFirstNameAndLastName\_whenFindByfindByNativeSQL\_thenReturnEmployeeObject() {  
  
 // Given: Setup object or precondition  
  
 employeeRepository.save(employee);  
  
 // When: Action or behavior that we are going to test  
 Employee getEmployee = employeeRepository.findByJPQLNameParams(employee.getFirstName(), employee.getLastName());  
   
 // Then: Verify the output or expected result  
 assertThat(getEmployee).isNotNull();  
 }  
  
  
 // JUnit test for custom query using findByNativeSQLNameParam with name param   
 @Test  
 @DisplayName("JUnit test for custom query using findByNativeSQLNameParam with name param ")  
 public void givenFirstNameAndLastName\_whenfindByNativeSQLNameParam\_thenReturnEmployeeObject() {  
  
 // Given: Setup object or precondition  
  
 employeeRepository.save(employee);  
  
 // When: Action or behavior that we are going to test  
 Employee getEmployee = employeeRepository.findByJPQLNameParams(employee.getFirstName(), employee.getLastName());  
   
 // Then: Verify the output or expected result  
 assertThat(getEmployee).isNotNull();  
 }  
}

**Code Summary**

This repository contains unit tests for the `**EmployeeRepository**` interface in a Spring Boot application. The `**EmployeeRepository**` interface extends the `**JpaRepository**` interface, providing CRUD operations for the `**Employee**` entity. Additionally, it defines custom query methods using JPQL and native SQL queries.

**Test Environment**

The `**EmployeeRepositoryTests**` class is annotated with `[**@DataJpaTest**](http://twitter.com/DataJpaTest)`, indicating that it is a JPA repository test and will be executed within a Spring context. This allows the tests to interact with the underlying database.

**Test Setup**

The `**setupTestData**` method is annotated with `[**@BeforeEach**](http://twitter.com/BeforeEach)` and is executed before each test method. It sets up the test data by creating an `**Employee**` object with sample values.

**Test Cases**

The test methods in the `**EmployeeRepositoryTests**` class cover various scenarios to test the repository operations. Each test method follows the **Given-When-Then** pattern to structure the test cases.

1. `**givenEmployeeObject\_whenSave\_thenReturnSaveEmployee**`: This test case verifies the save operation by saving an `Employee` object and asserting that it is not null and has a valid ID.

2. `**givenEmployeeList\_whenFindAll\_thenEmployeeList**`: This test case tests the find all operation by saving multiple `Employee` objects and asserting that the list of employees is not empty and has the expected size.

3. `**givenEmployeeObject\_whenFindById\_thenReturnEmployeeObject**`: This test case tests the find by ID operation by saving an `**Employee**` object and then retrieving it using the `**findById**` method. It asserts that the retrieved employee is not null.

4. `**givenEmployeeEmail\_whenFindByEmail\_thenEmployeeObject**`: This test case tests the find by email operation by saving an `**Employee**` object and then retrieving it using the `**findByEmail**` method. It asserts that the retrieved employee is not null and has the expected email.

5. `**givenEmployeeObject\_whenUpdate\_thenEmployeeObject**`: This test case tests the update operation by saving an `**Employee**` object, updating its properties, and asserting that the updated employee has the expected values.

6. `**givenEmployeeObject\_whenDelete\_thenRemoveEmployee**`: This test case tests the delete operation by saving an `**Employee**` object, deleting it by ID, and asserting that the deleted employee is not present in the repository.

Additional test methods demonstrate the usage of custom queries using JPQL and native SQL.

**Assertions**

Each test method uses assertions from the `**assertj-core**` library to verify the output or expected result. The assertions ensure that the expected conditions are met during the test execution.

**Running the Tests**

To run the tests, execute the test.

Github Repository: <https://github.com/MohosinMiah/Spring-Boot--Unit-Testing-Repository-Layer>