CODING SHUTTLE

Spring Testing

Unit Testing vs Integration Testing

Unit Test vs Integration Test

1. **Unit Tests**:

A unit test covers a single “unit”, where a unit commonly is a single class, but can also be a cluster of cohesive classes that is tested in combination.

- **Focus**: Individual components, such as methods in a service class.
- **Tools**: JUnit, Mockito
- **Example**: Testing a service method that performs a calculation or business logic

2. Integration Tests:

A test that covers multiple layers. This might cover the interaction between a business service and the persistence layer, for instance.

- **Focus**: Interaction between multiple components, such as repositories, services, and controllers.
- **Tools**: Spring Test, @SpringBootTest.
- **Example**: Testing the interaction between a service and a repository.

Unit Testing Example

```java
class EmployeeServiceTest {

    @Mock
    private EmployeeRepository employeeRepository;

    @InjectMocks
    private EmployeeServiceImpl employeeService;

    @Test
    void testGetEmployeeById() {
        Employee employee = new Employee(1L, "John Doe", "john.doe@example.com");
        when(employeeRepository.findById(1L)).thenReturn(Optional.of(employee));

        EmployeeDto result = employeeService.getEmployeeById(1L);

        assertThat(result.getName()).isEqualTo("John Doe");
        assertThat(result.getEmail()).isEqualTo("john.doe@example.com");
    }
}
```

Integration Testing Example

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
class EmployeeControllerIntegrationTest {

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    void testGetEmployeeById() {
        ResponseEntity<EmployeeDto> response = restTemplate.getForEntity("/employees/1", EmployeeDto.class);

        assertThat(response.getStatusCodeValue()).isEqualTo(200);
        assertThat(response.getBody().getName()).isEqualTo("John Doe");
        assertThat(response.getBody().getEmail()).isEqualTo("john.doe@example.com");
    }
}
```

Key Testing Annotations in Spring Boot

- **@SpringBootTest**: Used to create an application context and load the full application for integration tests. Useful in Integration Testing.
- **@DataJpaTest**: Used to test JPA repositories, configuring an in-memory database for the test. Useful in Unit Testing Service Layer and Persistence Layer.
- **@TestConfiguration**: Used to define extra beans or configurations for tests.
- **@WebMvcTest**: Used for testing Spring MVC controllers. It initializes only the web layer and not the entire context. Useful in Unit Testing Controller layer
- **@AutoConfigureTestDatabase**: Used to replace the actual database with an embedded database during tests.