**TESTING IN SPRING BOOT**

**Dependency**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<version>2.5.0</version>

</dependency>

**2 types of testing**

* **Integration Testing**
* **Unit Testing**

**Note**:

Keep Integration tests separate from Unit tests as Integration tests are time-consuming and might need an actual database to execute.

**Integration Testing With @SpringBootTest**

Integration tests focus on integrating different layers of the application. [There is no mocking is involved and we can use @Autowire as we have ApplicationContext (Spring container)]

**Annotations**

*@RunWith(SpringRunner.class)*

*@SpringBootTest(*

*SpringBootTest.WebEnvironment.MOCK,*

*classes = Application.class)*

*@AutoConfigureMockMvc*

*@TestPropertySource(*

*locations = "classpath:application-integrationtest.properties")*

*public class EmployeeRestControllerIntegrationTest {*

*@Autowired*

*private MockMvc mvc;*

*@Autowired*

*private EmployeeRepository repository;*

*// write test cases here*

*}*

**@RunWith(SpringRunner.class)** tells JUnit to run using Spring's testing support. SpringRunner is an alias for the SpringJUnit4ClassRunner.

The integration tests need to start up a container to execute the test cases.

The **@SpringBootTest** annotation is used to bootstrap the entire container. It creates the ApplicationContext that will be utilized in our tests.

We can use the **webEnvironment attribute** of **@SpringBootTest** to configure our runtime environment; we're using **WebEnvironment.MOCK** here so that the container will operate in a mock servlet environment.

The **@TestPropertySource** annotation is used to configure the locations of properties files specific to our tests.

Integration Test Example:

*@Test*

*public void givenEmployees\_whenGetEmployees\_thenStatus200()*

*throws Exception {*

*createTestEmployee("bob");*

*mvc.perform(get("/api/employees")*

*.contentType(MediaType.APPLICATION\_JSON))*

*.andExpect(status().isOk())*

*.andExpect(content()*

*.contentTypeCompatibleWith(MediaType.APPLICATION\_JSON))*

*.andExpect(jsonPath("$[0].name", is("bob")));*

*}*

If you want to **Define Your Own Configuration with @TestConfiguration**

*@TestConfiguration*

*public class EmployeeServiceImplTestContextConfiguration {*

*@Bean*

*public EmployeeService employeeService() {*

*return new EmployeeService() {*

*// implement methods*

*};*

*}*

*}*

**MOCKING With @MockBean**

Our Service layer code is dependent on our Repository.

But while testing Service layer, we don’t need to care about implementation of persistence layer.

To achieve this, we can use @MockBean and Mockito methods and Assertions.