Class-32: Integration Testing in Spring Boot

 by Pial Kanti Samadder

What is Integration Testing?

Definition

Testing how **multiple layers of the application work together**, not individually.

Key Concept

Integration Testing validates:

```
Client → Controller → Service → Repository → Database
```

Why Integration Testing Is Needed

- Unit tests can’t catch issues between layers
- Ensures API endpoints work end-to-end
- Ensures validation, exceptions, filters, and DB operations work as expected
- Matches real-world behavior closer than unit tests

Unit vs Integration Testing

Unit Testing	Integration Testing
Tests one component	Tests multiple components together
Uses mocks	Uses real components
Very fast	Slower
Ideal for Service layer	Ideal for API + DB flow
Low confidence	High confidence

@SpringBootTest

Loads the ENTIRE Spring Application Context

- Starts the full backend
- Loads all beans: Controller, Service, Repo, Security, Filters

Perfect For:

- End-to-end API testing
- Testing DB interactions
- Testing authentication/security filters
- Testing custom exception handlers

@SpringBootTest Example

```
@SpringBootTest
@AutoConfigureMockMvc
class UserApiTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void should_ReturnUser_When_IdIsValid() throws Exception {
        mockMvc.perform(get("/api/users/5"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(5));
    }
}
```

When NOT to Use @SpringBootTest

- When you only want to test a Controller method
- When the Service layer logic is not needed
- When DB is not needed
- When the test is slow

Use **@WebMvcTest** instead.

@WebMvcTest

Loads ONLY the Web Layer

- Loads controller + controller-related components
- Does not load:
  - Services
  - Repositories
  - DB
  - Security (unless asked)

Perfect For:

- REST API endpoint tests
- Request/response testing
- JSON validation
- Input validation errors

@WebMvcTest Example

```
@WebMvcTest(UserController.class)
class UserControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserService userService;

    @Test
    void should_Return200_When_RequestIsValid() throws Exception {
        mockMvc.perform(get("/api/users/1"))
                .andExpect(status().isOk());
    }
}
```

What is MockMvc?

MockMvc lets us:

- Call a controller without running the server
- Test HTTP methods (GET, POST, PUT, DELETE)
- Assert:
  - status code
  - JSON content
  - headers
  - validation errors

Benefits:

- Very fast
- Perfect for controller testing

@DataJpaTest

For Testing Repository Layer

- Loads Repository + Entity + Hibernate
- Automatically configures H2
- Rolls back after every test

Perfect For:

- Custom JPQL queries
- Custom SQL queries
- Entity relationships (OneToMany, ManyToMany)

@DataJpaTest Example

```
@DataJpaTest
class ProductRepositoryTest {

    @Autowired
    private ProductRepository repository;

    @Test
    void should_FindProductByName() {
        repository.save(new Product("Laptop"));

        Product result = repository.findByName("Laptop");

        assertNotNull(result);
    }
}
```

Why We Use H2 in Tests

- Fast in-memory DB
- No installation needed
- All tables auto-created
- Automatically cleared after tests
- Matches real SQL behavior

Used by:

- @DataJpaTest
- @SpringBootTest (when test profile is active)

Transaction Rollback in Tests

What Happens Automatically:

- Each test runs inside a transaction
- At the end of the test → rollback
- Ensures DB is clean for next test

Benefits:

- Tests don’t depend on each other
- No manual delete needed

Rollback Example

```
@DataJpaTest
class OrderRepositoryTest {

    @Test
    void should_Rollback_AfterTest() {
        repository.save(new Order("Laptop", 1));

        assertEquals(1, repository.count());
    }
}
// After test → count = 0
```

Common Testing Scenarios

For Controllers

- GET /api/users/{id} → Should return 200
- POST /api/products → Should validate JSON
- 404 on invalid ID
- 400 on validation error

For Repositories

- Custom query
- Pagination
- Sorting

For Full End-to-End

- Register user → Login → Get profile