🔰 Easy Level (3+ Years Experience)

Focus: Basics of integration testing with Spring Boot

✅ 1. What is integration testing in Spring Boot?

Answer:
Integration testing verifies the interaction between multiple layers (Controller → Service → Repository), often involving Spring context and database.

✅ 2. How do you annotate a Spring Boot integration test?

Answer:
Use `@SpringBootTest` to load the full application context and `@AutoConfigureMockMvc` to test MVC endpoints.

java Copy Edit

```java
@SpringBootTest @AutoConfigureMockMvc public class UserControllerIT { }
```

✅ 3. How do you test a REST API endpoint using MockMvc?

java Copy Edit

```java
@Test void getUserById() throws Exception { mockMvc.perform(get("/users/1"))
.andExpect(status().isOk()) .andExpect(jsonPath("$.name").value("John")); }
```

✅ 4. Difference between `@WebMvcTest` and `@SpringBootTest`?

Answer:

- `@WebMvcTest` : Loads only controller layer + MVC-related beans
- `@SpringBootTest` : Loads full application context including services, repositories, etc.

✅ 5. How to test POST API with JSON body?

java Copy Edit

```
@Test void createUser() throws Exception { mockMvc.perform(post("/users") .content("
{\"name\":\"Alice\"}") .contentType(MediaType.APPLICATION_JSON))
.andExpect(status().isCreated()); }
```

🟡 Moderate Level (5+ Years Experience)

Focus: Real DB, testcontainers, error flows, embedded servers

✅ 6. How to use H2 database in integration tests?

Answer:
Spring Boot uses `H2` in memory automatically if you include the dependency and set:

properties Copy Edit

```
spring.datasource.url=jdbc:h2:mem:testdb spring.jpa.hibernate.ddl-auto=create
```

✅ 7. How to verify database state after POST/PUT in integration test?

java Copy Edit

```
@Test void createUserAndCheckDB() {
mockMvc.perform(post("/users").content(json).contentType(...)); User user =
userRepository.findByName("Alice"); assertNotNull(user); }
```

✅ 8. How to mock external API call in integration test?

Use `WireMock` to simulate API behavior.

java Copy Edit

```
WireMock.stubFor(get(urlEqualTo("/external"))
```

```
.willReturn(aResponse().withBody("ok").withStatus(200)));
```

✅ 9. How to test exception and error responses?

java Copy Edit

```
mockMvc.perform(get("/users/999")) .andExpect(status().isNotFound())
.andExpect(jsonPath("$.message").value("User not found"));
```

✅ 10. What is @TestRestTemplate? When to use it?

Answer:
Use `@TestRestTemplate` to send real HTTP requests in integration tests when you want to test full HTTP stack.

✅ 11. How to test REST APIs with security enabled (JWT or Basic Auth)?

Add headers/token in test request:

java Copy Edit

```
mockMvc.perform(get("/secure") .header("Authorization", "Bearer " + token))
.andExpect(status().isOk());
```

🔴 Difficult Level (7+ Years Experience)

Focus: Real-world scenarios, Testcontainers, microservices, performance

✅ 12. How to use Testcontainers to spin up real DB like PostgreSQL?

java Copy Edit

```
@Container static PostgreSQLContainer<?> postgres = new PostgreSQLContainer<>
```

```
("postgres:14");
```

Configure `spring.datasource.url` using the container in test setup.

✅ 13. How to test Kafka message publishing in integration tests?

Use `EmbeddedKafka`:

java Copy Edit

```
@EmbeddedKafka(partitions = 1, topics = "test-topic") class KafkaIntegrationTest { }
```

✅ 14. How to simulate network latency/failure in integration tests?

Use `WireMock`'s `withFixedDelay()` or return HTTP 500:

java Copy Edit

```
.withFixedDelay(3000) .willReturn(serverError())
```

✅ 15. What's contract testing? How is it different from integration testing?

Answer:
Contract testing ensures the consumer and provider agree on the API behavior (e.g., using **Spring Cloud Contract**). Integration tests verify real runtime interaction. Contract testing can run without starting both services.

✅ 16. How do you parallelize integration tests for faster execution?

Use:

- JUnit 5 parallel execution (`junit.jupiter.execution.parallel.enabled=true`)
- Separate DBs for parallel threads (e.g., via `Testcontainers`)

✅ 17. What is `@DirtiesContext`? When should you use it?

Answer:
It tells Spring to reload context after the test to avoid pollution from state changes. Use when one test might affect others.

java Copy Edit

```
@DirtiesContext(classMode = AFTER_EACH_TEST_METHOD)
```

✅ 18. How to integrate test coverage with CI/CD pipeline?

Use:

- **Jacoco** plugin for Maven/Gradle
- Export reports in `XML/HTML`
- Integrate with **SonarQube** or GitHub Actions

✅ 19. How do you test idempotency in REST APIs?

Call the same `POST/PUT` API multiple times with the same data and assert the result or resource state remains unchanged.

✅ 20. How to handle flaky tests in integration testing?

Strategies:

- Use `Awaitility` for async behavior
- Ensure test isolation
- Avoid shared state
- Mark unstable test and run separately (`@Tag("flaky")`)