🔁 General Integration Testing

✅ 1. What are the key differences between unit testing and integration testing in Spring Boot?

Answer:

- **Unit Testing**: Isolates one class/component; uses mocks
- **Integration Testing**: Tests multiple layers (e.g., controller + service + repo); uses real beans, DB, configs

✅ 2. How do you test APIs with real HTTP calls in Spring Boot?

Answer:
Use `TestRestTemplate` or `WebTestClient` (for reactive) in combination with `@SpringBootTest(webEnvironment = RANDOM_PORT)`.

✅ 3. What annotations do you use for integration tests in Spring Boot?

Answer:

- `@SpringBootTest`
- `@AutoConfigureMockMvc`
- `@Testcontainers` (if using Testcontainers)
- `@Sql`, `@DirtiesContext` for DB setup/teardown

✅ 4. How do you test a secured REST API with OAuth2 or JWT in integration tests?

Answer:
Generate/access a valid token in the test, and send it via the `Authorization` header.

✅ 5. What is the purpose of `@Sql` and `@SqlGroup` in tests?

Answer:
They execute SQL scripts **before or after** tests to set up or clean up data.

```
java
```

Copy Edit

```
@Sql(scripts = "/setup.sql", executionPhase = BEFORE_TEST_METHOD)
```

🧪 Mocking & Data Setup

✅ 6. How do you initialize test data in integration tests?

- Use `@Sql` or `TestEntityManager`
- Use repository directly inside test method
- Use `@BeforeEach` to insert via repo

✅ 7. How do you isolate each test method from DB changes of others?

- Use H2 (in-memory DB)
- Use `@Transactional` at test level
- Use `@DirtiesContext` or `@TestPropertySource` to load isolated context

✅ 8. How do you use `MockMvc` for integration testing?

Answer:
It performs API calls and validates status, headers, and body:

java Copy Edit

```
mockMvc.perform(get("/api")) .andExpect(status().isOk())
.andExpect(jsonPath("$.id").value(1));
```

🧱 Database & Repository Testing

✅ 9. How do you test your repository layer with a real database?

Answer:
Use `@DataJpaTest` with in-memory H2, and `TestEntityManager` for insert/query.

✅ 10. How can you simulate DB constraints and assert failure (e.g., unique constraints)?

Insert duplicate record and assert an exception like `DataIntegrityViolationException`.

✅ 11. How do you test lazy-loaded associations in integration tests?

Use `@Transactional` test method so JPA session remains open during assertion.

📎 Microservices & External Services

✅ 12. How do you mock external REST API in integration tests?

- Use **WireMock** server
- Stub endpoints and assert the request payload and response
- Avoid using real API endpoints

✅ 13. What is Spring Cloud Contract? How does it help integration testing?

Answer:
It's used for **consumer-driven contract testing**. It ensures both consumer and provider agree on a contract. It generates stubs that can be reused in integration tests.

✅ 14. How to simulate service unavailability or timeout in integration tests?

- Use WireMock's `withFixedDelay()`
- Simulate HTTP 500/504 status

📊 Test Environment & Performance

✅ 15. How to set different configuration or DB for integration tests?

Use:

- `@TestPropertySource`
- `application-test.yml` + `@ActiveProfiles("test")`

✅ 16. How do you measure performance or latency in an integration test?

Use:

- `System.nanoTime()` to track duration
- `Stopwatch` from Guava or Spring's `StopWatch`

✅ 17. How do you handle time-sensitive operations in integration testing?

- Mock `Clock` or use a custom time provider
- Freeze time with `TimeMachine` / `Mockito`

🧭 Advanced Tools & CI/CD Integration

✅ 18. What is the role of Testcontainers in integration testing?

Answer:
Testcontainers allows real **Dockerized** infrastructure (like PostgreSQL, Kafka, Redis) to run during test execution. Ensures realistic environment.

✅ 19. How do you run integration tests in CI/CD pipelines?

- Use Maven/Gradle build steps (`mvn verify`)
- Start up DB or services using Testcontainers or GitHub Actions services

- Generate test reports with JaCoCo and publish

✅ 20. What is the difference between end-to-end (E2E) and integration testing?

- **Integration Test**: Tests multiple components inside the same app
- **E2E Test**: Covers entire workflow across multiple services/systems (UI, Backend, DB, API)