Spring Testing

Unit Testing the Persistence Layer

Using @DataJpaTest Slice

@DataJpaTest is tailored to test JPA components like repositories. It configures an in-memory database, sets up Spring Data JPA repositories, and scans for JPA entities. This makes it ideal for testing repository methods and their interactions with the database.

By default, ***@DataJpaTest*** runs each test within a transaction, which is rolled back after the test completes. This ensures tests do not affect each other and provides a clean state for each test.

Configuring Test Database

Use the AutoConfiureTestDatabase Annotation to configure the test database

```
@AutoConfigureTestDatabase(replace = AutoConfigureTestDatabase.Replace.NONE)
```

- **ANY**: Replace the DataSource bean whether it was auto-configured or manually defined.
- **NONE**: Don't replace the application default DataSource.
- **AUTO_CONFIGURED**: Only replace the DataSource if it was auto-configured.

Read more here:

TestContainer

Use Testcontainer to mock a real database, Use when mocking the repository or in Integration testing.

Step 1: Download the Docker Application from here and run it.

Step 2: Add the following dependency:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-testcontainers</artifactId>
    <scope>test</scope>
</dependency>
```

Running TestContainer

Step 3: Create the following Test Configuration:

```
@TestConfiguration(proxyBeanMethods = false)
    public class TestcontainersConfiguration {
        @Bean
        @ServiceConnection
        PostgreSQLContainer<?> postgresContainer() {
            return new PostgreSQLContainer<>(DockerImageName.parse("postgres:latest"));
        }
}
```

Step 4: Import the Configuration in your test file:

```
@DataJpaTest
@Import(TestcontainersConfiguration.class)
class EmployeeRepositoryTest {...}
```