Test-Concept

- Unit-Tests
- REST-Tests
- Integration-Tests
- Acceptance-Tests
- Performance-Tests
- System-Integration-Tests

General Concepts

The following test concept uses terms like “double” and “mock” (maybe in inflected form like “mocking” or “mocked”), “whitebox-test” and “blackbox-tests” and “test-fixture”. Please look up their definition in the glossary

Where our APIs should be designed in a way that it’s possible, using a mocking library like *Mockito* often leads to shorter test code.

Most important for a test is, to clearly express what it actually is testing. For this, it might help to wrap test setup and assertions into test fixture

Kinds of Tests

Depending on the concrete aspects which we want to test, we are using different kinds of tests as described as follows.

Unit-Tests In this project a *Unit* for *UnitTests* can be a single method (function), a class or even a group of classes which express a common concept.

The unit are technically whitebox-tests and count into test-code-coverage. But the whitebox-knowledge should only be used for the test-fixture.

Unit-Test in this project are implemented with *JUnit Jupiter*, *Mockito* and *AssertJ*.

Unit-Tests do not use any external systems, not even a database. They just test the unit, not any dependencies or proper integration with dependencies.

Such tests usually run very fast and should test all branches.

These Tests are always named `...UnitTest` and can automatically run in the build-process.

REST-Tests At the level of REST-Controllers, *Spring’s* `WebMvcTest`, a special kind of Unit-Test, are utilized to replace simple unit tests. Such tests issue REST-requests through a mocked REST-Layer and therefore use the controllers similar to a real client. Otherwise, the implementation technologies are like those of Unit-Tests.

Being unit-tests, also REST-tests are whitebox-tests and count into test-code-coverage.

Like other Unit-Tests, REST-Test do not use any external systems, not even a database. They just test the REST-related parts of the unit, e.g. URL-Mappings, HTTP-Headers and proper JSON encoding of request and response data. Other dependencies and integrations with such are not tested on this level.

Such tests usually run very fast, but should focus on REST-specific issues, leaving branch-testing to pure Unit-Tests.

These Tests are always named `...RestTest` and can automatically run in the build-process.

Integration-Tests Integration-Tests in this context mean integration with support systems like databases or messaging-systems, but not integration with external systems.

Integration-tests, are blackbox-tests and do not count into test-code-coverage.

Such tests are implemented with *JUnit Jupiter* through some sort of `@SpringBootTest`, e.g. `DataJpaTest` and usually utilize *Testcontainers* and *Docker* to wrap the supporting system, e.g. the *PostgreSQL* database. *Mockito* can also be used for this kind of tests, to separate multiple integrations.

Integration-Tests are relatively slow and therefore should focus on the integration. Java-internal issues should be tested through Unit-Tests.

These Tests are always named `...IntegrationTest` and can automatically run in the build-process.

DataJpaTest / Database-Integration-Tests In this project, a major part of the program logic is coded in the database as stored procedures, functions and triggers.

This program logic is tested through *integration tests* using `DataJpaTest` because pure unit tests in the database are not only cumbersome but also easily lead to large test gaps.

Acceptance-Tests We define Acceptance-Tests as test which describe user-stories, respectively high-level business requirements. Acceptance-Tests run on a fully integrated and deployed system with deployed doubles for external systems.

Acceptance-tests, are blackbox-tests and do not count into test-code-coverage.

TODO.test: Complete the Acceptance-Tests test concept.

Scenario-Tests Our Scenario-tests are induced by business use-cases. They test from the REST API all the way down to the database.

Most scenario-tests are positive tests, they test if business scenarios do work. But few might be negative tests, which test if specific forbidden data gets rejected.

Our scenario tests also generate test-reports which contain the REST-API calls needed for each scenario. These reports can be used as examples for the API usage from a business perspective.

There is an extra document regarding scenario-test, see Scenario-Tests README.

Performance-Tests Performance-critical scenarios have to be identified and a special performance-test has to be implemented.

The implementation-technologie depends on the scenario.

Performance-tests, are blackbox-tests and do not count into test-code-coverage.

Such tests usually are very slow and should not be automatically run in the build-pipeline but manually, after critical areas have been changed.

System-Integration-Tests We define System-Integration-Tests as test in which this system is deployed in a production-like environment to test integration with external systems.

System-Integration-tests, are blackbox-tests and do not count into test-code-coverage.

TODO.test: Complete the System-Integration-Tests test concept.