Testing Guidelines

All findings of tests (nightly tests, tests run from devs, testing by hand) should result in writing a new ticket describing the issue.
We do not want a abandoned confluence document where issues don't get noticed!

Used Testing Frameworks

Unit Tests

Unit Tests are writtten with the combination of https://mochajs.org/ and https://www.chaijs.com/. Mocha is the foundational testing framework (test runner), chai adds functions to check the test results.

Import everything you need into the <test-name>.spec.ts and run `npm test`. *You may need to add more testing functionalities like chai-http to test http requests.*

install chai and mocha

1. run `npm install chai mocha ts-node @types/chai @types/mocha --save-dev`
2. add `"test": "env TS_NODE_COMPILER_OPTIONS='{\"module\": \"commonjs\" }' mocha -r ts-node/register 'tests/**/*.spec.ts'"` to scripts in `package.json`
3. add `test/unit` folder to project
4. run tests with `npm test`

Source: https://dev.to/matteobruni/mocha-chai-with-typescript-37f

example test (this should not be used in our real codebase to cover the fact that you don't want to write real tests, yea looking at you @ Joneppard 😉):

```
import { assert } from 'chai';

describe('2 + 2', () => {
  it('is 4 (quick maths)', () => {
    assert.equal(2 + 2, 4);
  });
});
```

a list of all available assert functions can be found here.

Integration / E2E Tests

Integration and E2E Tests are written in https://www.cypress.io/. Cypress tests are written like in mocha and chai.

install cypress

1. run `npm install cypress --save-dev`
2. add `"e2e": "cypress open"` to scripts in `package.json`
3. run `npm run e2e` to open cypress app
4. put tests inside the `integration` folder

Levels of Testing

We use 3 different levels of testing in the project.

Unit Tests

- test a specific functionality
- are isolated a failed test does not lead to other tests failing because of it
- are used to check if a change breaks single functionalities
- give the reviewer the chance of testing before looking at the code (test therefore has to be meaningful)

Integration Tests

- tests how different functionalities work when combined

    - use the result of function call 1 to call function 2 …
- should run regularly (nightly / when PR is created) to ensure everything is running alright

End2End (E2E) Tests

- tests one specific user flow
- ensures that the software works as a whole

Test coverage

Testing the “happy path” is required in every case. Those tests describe if the expected input produces the expected output if the code is working as expected.

Error cases should also be tested: each error that is handled by the code should therefore be tested. If the code contains many different branches then they should be tested.

We aim for 45% code coverage.

Who is writing the test

Unit Tests

Every **developer** writes the units tests along writing the functionality itself. Not delivering a test along is only allowed if

- tests for the functionality exist yet
- its not possible to test (because of special reasons need to be stated in the PR)

Unit tests therefore are implicit required!

Integration & E2E Tests

Integration Tests and especially e2e Tests are written based on a extra ticket. Ticket is created when all storys epic is finished. The ticket specifies a collection of other feature tickets which functionalities should be included in the test.

Those tests can be written by every developer, **the responsible service lead is hotly 🔥 favored!**