This document specifies how testing is going to be carried out in the project. The document contains a test plan where the testing tools and testing strategy is presented. The document also contains test suites where the test cases are presented.

Overall testing goal

The overall goal with the testing is to achieve 100 percent functionality coverage. This is to make sure that the system fulfills the customer's requirements and works as expected by the end-delivery. Our goal is to at least carry out one test on every part of the system, whether it being automated, explorative, negative or other.

Automated and manual testing should be done on a weekly basis to catch bugs early.

Test plan

Testing tools

For the automatic tests the following tools will be utilized. When an automated test is done; take a print screen and paste it into the test report.

React.js front-end

- Test framework **Jest**.
- Code coverage and test reports. In the terminal: **npm test**. (Same as jest --coverage -u)

ASP.NET backend

- Test framework **nUnit, Postman**.

Stress tests

- Load testing tool **jMeter**.

Testing strategy

Level	Approach	Type	Run test
Unit	Automated testning, negative testning	Automated	In the terminal: npm test
	Automated testning, negative		In the terminal: npm

Integration	testning	Automated	test
System	Explorative testning, user tests	Manual	Manual
Acceptance	Test cases	Manual	Manual

Levels

Testing will be carried out on different levels; unit, integration, system and acceptance.

- On **unit level** a smaller part of the source code is tested. A unit is a function or a method.
- On **integration level** already tested units are tested together to see if the parts work together.
- On **system level** the whole application is tested in its entirety.
- On **acceptance level** the software is verified against the customers requirements.

Approach

Testing will be carried out in different ways; automated, negative, explorative, user tests and test cases.

- **Automated testing:** In the automated tests the test code is written with the testing tools listed above. This will mainly be done to test that the functions work as expected.
- **Negative testing:** Like the automated tests, the test code for negative testing is written with the testing tools listed above. In these tests we will use improper in-data to get error messages. By doing this we can then see that the right error message is shown at the right time. For example an expected 404 should not be a 403.
- **Explorative testing:** In explorative testing the tester gets to use the system and keep a log over the steps taken throughout the system. If and when something unexpected happens this will be logged by the tester so that the developer knows where there might be faulty code or bugs hiding.
- **User tests:** The user tests let the final user test the system. Problems and comments are documented and used to evaluate the usability.
- **Test cases:** The test cases are written based on the requirements in the product backlog. They are used to test that the system has the expected functionality.

Code coverage

The parts of the application that might not be covered by the automated tests are presented by the code coverage tool Istanbul, that comes with the Jest framework, and the test reports generated from the tool.

Test suites

A test suite is a collection of test cases that tests the same requirement from different angles based on the scenarios that might take place. The test suites will here on after be listed based on the type of test it is; automated or manual. The test suites can be found in their respective subdocument.

6.1. Automated testing (https://github.com/jimmybengtsson/grupp03-redriver/wiki/6.1-Automated-testing)
6.2. Manual testing (https://github.com/jimmybengtsson/grupp03-redriver/wiki/6.2-Manual-testing)