TEST PLAN

Abracadabra Semester 3, DB03

Version 0.8 | 11-01-2021

Inhoudsopgave

Introduction

This document will list each test that should be performed to guarantee the useability of the product. Each test should meet the demands set in the definition of done. Furthermore, this document outlays which tests are going to be used and what they cover. Finally, each test method is bound to risks these risks are also described on a per test basis.

Test matrix

Test name	Through the user interface	Full infrastructure stack	Entire User flow
Performance tests	**X**	**X**	
Unit Tests			**X**
End to End test	**X**	**X**	**X**
Acceptance testing	**X**		**X**
Monkey testing	**X**		

Test approach

Unit tests

Unit tests will be used to ensure the functioning of small pieces of logic, the main purpose of the unit tests is to narrow down possible issues to a unit level. The unit tests should be written in such a way that they could be automated with a CI workflow. The unit tests should be used in cohesion with the end-to-end tests and narrow down errors when an end-to-end test fails.

Definition of done: As defined by the definition of done unit tests should be written on important features that could use extra narrowing down of issues to track down possible issues as fast as possible.

Risks:

- If the test is written incorrectly it might still pass even when the code doesn't do as expected.

Performance tests

Performance tests will used to ensure that the product is useable, and functions as expected. Performance tests should be performed in as well front-end as back-end. Performance tests should show that the product is in a useable state and can handle a production situation without problem.

Definition of done: As defined by the definition of done the performance of the product should meet at least an above 85% score from google lighthouse in the front-end and response times from the back end should be kept under 1000ms.

Risks:

- If some parts of the code are badly optimized, it could affect other parts of the code to also not pass the definition of done.

End-to-end testing

End to end tests will be used to ensure that each feature of the project works from top to bottom. The end-to-end test will cover each major feature and should reach the thresholds of the definition of done. The end-to-end test are automated with a CI workflow, the test gets built in the front-end and perform actions the same way a user would perform them. The end-to-end test makes use of the full stack of the project.

Definition of done: As defined by the definition of done there should be an 80% coverage, this means that at least 80% of features should have an end-to-end test making sure they function.

Risks:

- If the code doesn't pass on a unit test level it will also never pass the end-to-end tests.
- If no API connection could be established all tests will fail automatically.

Acceptance testing

Acceptance tests will be used to ensure the user experience is as expected and that the feature is implemented correctly. The acceptance tests should be performed each time a feature is finished in development and before a pull request is made.

Definition of done: As defined by the definition of done all features should pass the acceptance test.

Risks:

- A feature is deemed accepted but in reality, it wasn't acceptable for production yet.

Monkey tests

Monkey tests will be used to ensure that the front-end has no caveats and doesn't show unexpected behavior. Monkey tests should be performed at random and with no prior knowledge of the application.

Definition of done: As defined by the definition of done each page should at least be covered once by a monkey test.

Risks:

- The test carried out during monkey testing is so random that it is either not possible or very difficult to recreate any bug.

Code coverage

Jest has a build in function that shows how much of the code is covered with tests. We can get information about the statements, branches, functions and lines.

Branches: Has each branch of each control structure been executed? For example, given a statement that could be true and false, have both been tested?
Function: Has each function been executed?
Lines: Has each executable line in the file been executed?

We want every section to have a specific percentage of code coverage.
Statements: 80%
Branches: 80%
Function: 80%
Lines: 80%

When you look in the auto generated file of the code coverage, you can open a html file. In this file, the percentages can be seen.
The file works with different colors in each file, so you can see what needs to be improved in the testing.
Pink: Statement not covered.
Orange: Function not covered.
Yellow: Branches not covered.

For example, on the sideline (1x): so not all branches are tested.

```
 9      function Subject({ subjectName, response }) {
10  1x    if (response === 404 || response == "failure" || response === 400) {
11          return <DefaultErrorPage statusCode={404} />;
12        }
13
14  1x    return (
15          <>
16            <Navbar subjectTitle={subjectName} />
17            <FilterButtons subjectTitle={subjectName} />
18            <QuestionBody question={response} subject={subjectName} />
19          </>
20        );
21      }
22
23      export default Subject;
```