**Quality assurance**

Student: Jorn Kersten

Student number: 483331

Date: 11-01-2023

Semester: S-DB-IPS3-S3-DB03

Table of contents

[What is software testing 3](#_Toc124358521)

[Front-end 4](#_Toc124358522)

[Unit and integration tests 4](#_Toc124358523)

[Static code analysis 4](#_Toc124358524)

[Performance tests 5](#_Toc124358525)

[Load test 6](#_Toc124358526)

[Back-end 7](#_Toc124358527)

[Unit tests 7](#_Toc124358528)

[Integration tests 7](#_Toc124358529)

[Security tests 8](#_Toc124358530)

# What is software testing

There are several types of software testing that are commonly used to evaluate the functionality and quality of software applications. Some of the main types of software testing include:

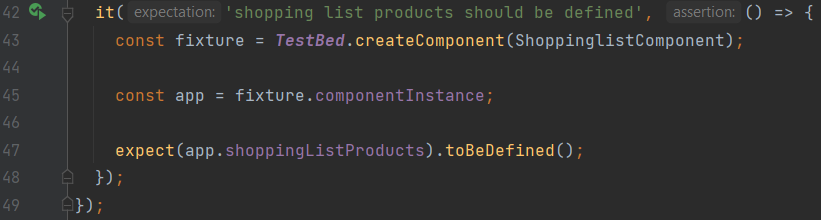
Unit testing: small, isolated tests that focus on individual components or functions of an application. They are typically written by developers and used to ensure that each component is working correctly before the application is integrated.

1. Integration testing: tests that are used to ensure that different components of an application work together correctly. They are typically run after unit tests have been successfully completed, and help to ensure that the application is functioning as intended before it is released to users.
2. Functional testing: tests that are used to ensure that an application meets the requirements of its users. They check that the application can perform the tasks that it is intended to perform and that it behaves as expected in different scenarios.
3. Acceptance testing: testing that is performed by the end user or a representative of the end user to ensure that the application meets the needs of the user. It is performed before the application is released to the general public and it validates if the requirements of the user are met.
4. Non-functional testing: testing that is focused on the performance and security of the application, rather than its functionality. Non-functional tests include stress tests, which check how well the application can handle high levels of usage and large amounts of data, as well as security tests, which check for vulnerabilities in the application.
5. Performance testing: tests that check the application's performance under different workloads, including load testing and stress testing, where the application's performance is measured under normal and peak conditions.
6. Regression testing: test that checks if any recent changes to the codebase did not break any previous functionalities
7. Compatibility testing: testing to see if the application behaves as expected across different platforms and devices.
8. Usability testing: checking how user-friendly the application is
9. Exploratory testing: testing where the tester has the freedom to investigate the application and explore its functionality in an unstructured way.

# Front-end

## Unit and integration tests

For testing my front-end application, I mainly used integration/unit testing. This involves checking whether all components work with each other. You can see an example of this test in the code snippet below.



## Static code analysis

It is possible to try to check all the code yourself for errors, but often you miss things. For this, something like static code analysis is often used; I used sonarcloud. Sonarcloud checks code for bugs, security flaws, etc.

Afbeelding met tekst

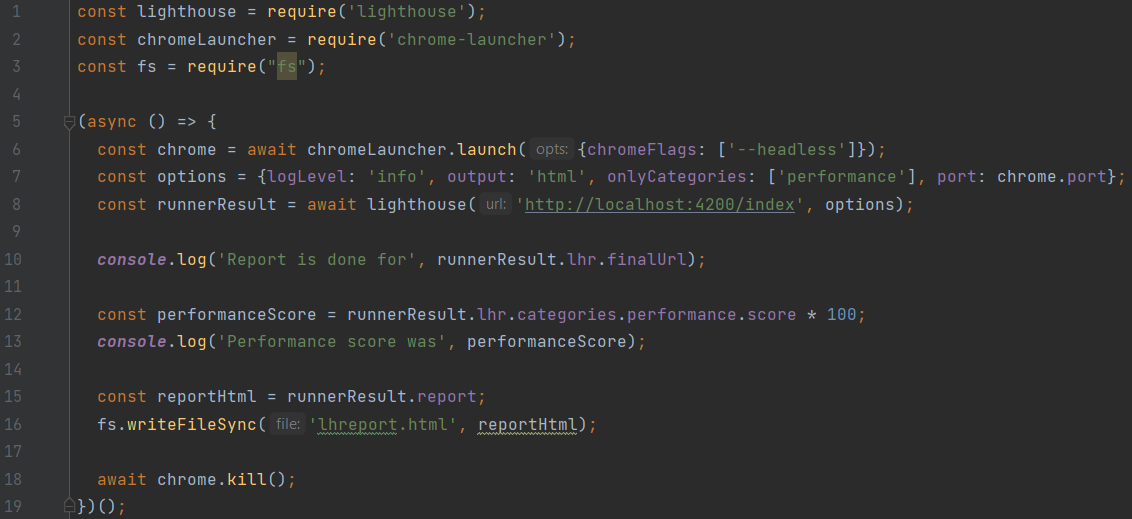
Automatisch gegenereerde beschrijving

The above test are found in the [CI file](https://github.com/Jorn-Kersten/DB-03-Frontend/blob/main/.github/workflows/CI.yml) of my front end.

## Performance tests

To test whether the website actually loads quickly, I applied performance testing. By installing an npm package from Google called Lighthouse, you can easily test the performance of your application.

The code below makes sure there is a headless Chrome browser running. Lighthouse will navigate to the index page of my application, then it will wait until the page is fully loaded and dumps everything in a lhreport.html file.



After calling the CI file, the CD is called. At the end of this page, lighthouse tests are called, the reports of which are then put into a file and pushed along to git. An example of this file is shown below.

Afbeelding met tekst, monitor, schermafbeelding, scherm

Automatisch gegenereerde beschrijving

## Load test

To see if the programme continues to work under much a heavy load, I apply stress tests with Artillery. Artillery is a package for npm that makes it very easy to write stress tests.

The code snippet below contains a .yml file that calls a particular link, in this case the homepage. It then visits this page in two stages. A warmup phase; this one is under normal load. A ramp up phase; this one is more or less a hefty load. If one of the tests fails, the whole load test fails.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Final result:

Afbeelding met tafel

Automatisch gegenereerde beschrijvingThese tests are found in the [CD file](https://github.com/Jorn-Kersten/DB-03-Frontend/blob/main/.github/workflows/CD.yml) of my front end.

# Back-end

## Unit tests

The back-end is very important to test well, you want to have the right data and good data from the endpoints. Testing this is done with mock data that is created in an h2 database when the test is started.

In Quarkus I've made use of the @QuarkusTest dependency to define that specific function as a test.

In the example below, I checked whether the mock product I want to create is actually returned by the response of the function being called.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

## Integration tests

Since keycloak is embedded in the Quarkus backend service we need to make sure it is well integrated with the written code.

In the example below the function updates the user given by the @TestSecurity. If the String username was a different user, we would not be allowed to update that user since the @TestSecurity has ‘admin’ as given user. This also has something to do with security tests.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

## Security tests

To make sure your application is secure you need to perform security tests. In Quarkus, together with keycloak we can check whether a user is allowed to access an endpoint or not.

In the test below, a product of the user 'admin' is being updated. Before this can happen, the user 'admin' must be logged in, this can be seen in the @TestSecurity. Since the user logged in is also 'admin', the response code 200 is 'Ok'.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

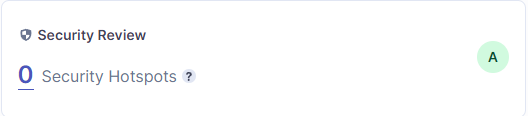
In the second example, the function testGetUserByIncorrectName tries to retrieve the profile of 'admin' as user 'Jorn'. This is not allowed within the application, so the response code 401 aka 'Unauthorised' comes up.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

[Backend tests](https://github.com/Jorn-Kersten/DB-03-Backend/tree/main/src/test)

SonarCloud also provides lots of functionality to check if there is a security flaw. For example, someone left a token or a password in the code and committed this to the repository. SonarCloud will recognize this and your workflow job will fail.



All the above processes are run in the CI pipeline from GitHub.

[Workflow files](https://github.com/Jorn-Kersten/DB-03-Backend/tree/main/.github/workflows)