**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](#_Toc124509641)

[Front-end 3](#_Toc124509642)

[Unit and integration tests 3](#_Toc124509643)

[Static code analysis 3](#_Toc124509644)

[Performance tests 4](#_Toc124509645)

[Load test 5](#_Toc124509646)

[Back-end 6](#_Toc124509647)

[Unit tests 6](#_Toc124509648)

[Integration tests 6](#_Toc124509649)

[Security tests 7](#_Toc124509650)

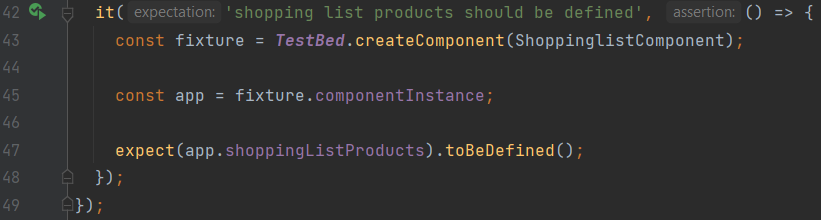
# What is software testing

Because I first wanted to know what kind of tests I could all use and what I need which tests for, I first researched what kind of test there are in software. I recorded this in a study that you will find in the file [Software tests research](Research%20documents/Software%20tests%20research.docx).

# 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.

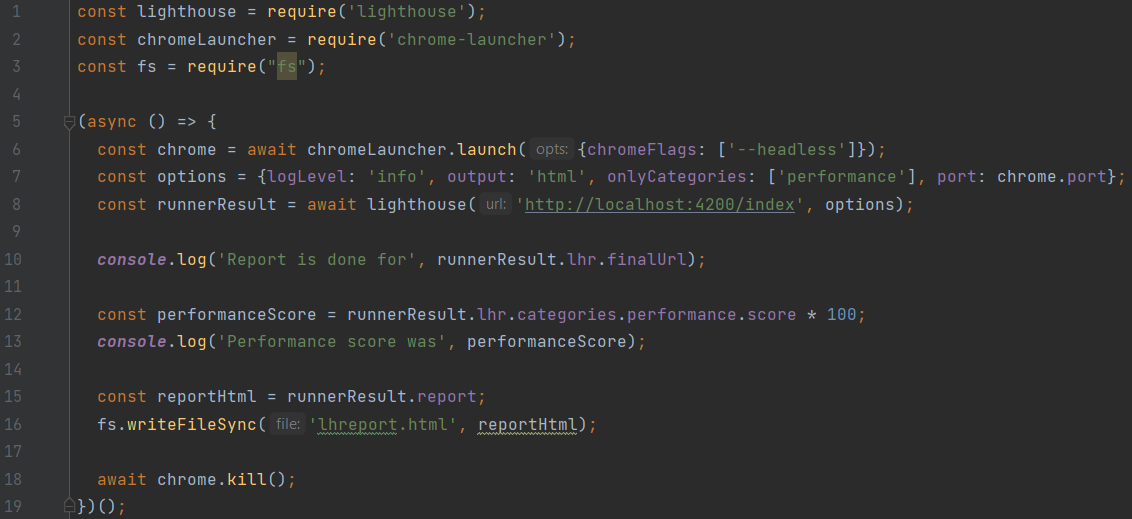
[View Sonarcloud front-end](https://sonarcloud.io/project/overview?id=Jorn-Kersten_DB-03-Frontend)

[View Sonarcloud back-end](https://sonarcloud.io/project/overview?id=Jorn-Kersten_DB-03-Backend)

## 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

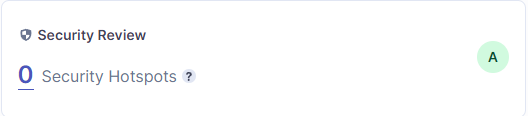
In the second example, the function testGetUserByIncorrectName tries to retrieve the profile of 'admin' as user 'Jorn'. This is not allowed within the application, so the response code 401 aka 'Unauthorised' comes up.

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)