TEST REPORT FOR THE E-COMMERCE STORE

COMP.SE.200-2022-2023-1
Software Testing
Petri Kreus (K421552)
Olli Sund (150162212)

CONTENTS

1. INTRODUCTION 1
2. DEFINITION OF DONE 2
3. TESTS & CI PIPELINE 3
    3.1 Description of the tests 3
    3.2 Running the tests locally 6
    3.3 CI pipeline and coveralls 6
4. FINDINGS AND CONCLUSION 9
REFERENCES 12

LIST OF ABBREVIATIONS

CI	Continuous Integration

1. INTRODUCTION

This document contains a report of the testing process for the utility library of the front-end of the E-Commerce Store application. The aim of this document is to give a thorough description of what was tested, how the tests were run and what were the results.

The report begins with a description of the target of the testing and a list of issue categorization in Section 2. Next, section 3 gives an overview of the tests and the GitHub Actions workflow, in addition to instructions of how to run the tests. Finally, section 4 provides our findings and conclusion of the testing process.

2. DEFINITION OF DONE

The target of the testing was to test the most critical parts of the utility library. The files chosen for the tests are essential for the front end using the utility library, which is why all tests must pass, until the testing can be deemed passed as a whole.

The following severity rating of the bugs is utilized, listed from the least critical to the most critical:

- minor
- medium
- major.

3. TESTS & CI PIPELINE

This section describes the testing process. First, the files under testing are listed. Second, a guide is provided for running the tests and creating a coverage report locally. Finally, the used Continuous Integration (CI) pipeline and Coveralls integration is explained.

3.1 Description of the tests

Unit tests were made using the *Jest* JavaScript Testing Framework [6]. Jest was selected because it is easy to use and there is a little or no need for configuration. It also enables easy generation of code coverage reports with the `--coverage` flag. The ten selected utility function files from the provided library were the following:

Source File	Rationale for selection/Example use case
add.js	Basic math function, may be used in multiple situations.
at.js	May be used in many situations when getting data from an object.
defaultTo.js	May be used when displaying and saving information.
divide.js	Basic math function. May be used when displaying prices and in other calculations. Has a critical error in syntax.
filter.js	May be used when displaying product lists.
isDate.js	May be used when displaying dates in product pages. Working with dates is a well-known challenge in JavaScript development.
isEmpty.js	May be used when checking validity of form fields.

map.js	May be used when converting array data into React components.
reduce.js	May be used when displaying prices in cart.
words.js	May be used in parsing search parameters in product search.

Table 1. *Selected source files with their selection rationale and relation to scenarios.*

The tests consist of ten test suites corresponding to each of the selected files. Total amount of unit tests written was 25. Tests can be found from the `tests/` folder in the project root. Tests are named using the convention: *`filename`*`.test.js`, where the file-name is the utility library file under testing. The individual tests were written as described in the test plan. The functions are given predefined inputs and the function outputs are compared to the expected output. If output matches the expected output the test passes. If the function returns something unexpected or there are errors, the test fails.

A few changes were made to the test cases since the test plan. Regarding the utility library file words.js, there were a few misunderstandings of how the function should work. The test cases were updated to match the new understanding. A comparison of the original test description and updated test descriptions can be viewed below.

Original test case for words.js

ID	TC021
Name	Input string is divided into individual search words
File	words.js
Type	functional test, positive test
Purpose	To test that the input string is correctly divided into words
Preconditions	Words.js is imported
Inputs	String: “one two three”, Delimiter pattern: ” “ (empty space)
Expected Results	[“one”, “two, “three”]
After-conditions	-

Updated test cases for words.js

ID	TC021
Name	Returns an array of individual words, without matcher parameter
File	words.js
Type	functional test, positive test

Purpose	To test that the input string is correctly divided into words
Preconditions	Words.js is imported
Inputs	String: “one two three”
Expected Results	[“one”, “two, “three”]
After-conditions	-

ID	TC022
Name	Returns an array containing the first found match, based on given matcher string
File	words.js
Type	functional test, positive test
Purpose	To test that the function returns correct array with string-based pattern
Preconditions	Words.js is imported
Inputs	String: “one two three”, pattern: “two”
Expected Results	[“two]
After-conditions	-

ID	TC023
Name	Returns an array of individual words, based on given global regexp
File	words.js
Type	functional test, positive test
Purpose	To test that the function returns correct array with given RegExp pattern
Preconditions	Words.js is imported
Inputs	String: “one, two, & three”, pattern: /[^,]+/g
Expected Results	[“one”, “two”, “&”, “three”]
After-conditions	-

ID	TC024
Name	Returns an empty array if no matches are found
File	words.js
Type	functional test, positive test
Purpose	To test that the function returns an empty array when no matches are found
Preconditions	Words.js is imported
Inputs	String: “one two three”, pattern: “four”
Expected Results	[] (empty array)
After-conditions	-

3.2 Running the tests locally

The tests can by run locally from a terminal by cloning the testing project's GitHub repository [7] and then navigating to the project root folder. First, required dependencies should be installed by running the command:

```
$   npm install
```

You can run the unit tests without a coverage report in the terminal with the command:

```
$   npm test
```

You can run the tests and create a coverage report using the command:

```
$   npm run test:coverage
```

When locally running tests with coverage, the coverage report can be found from a folder named "coverage" from the project root folder. Path to the summary report is the following:

```
.\coverage\lcov-report\index.html
```

3.3 CI pipeline and coveralls

In addition to Jest, a Node.js dependency called "coveralls" was added to the testing project as a development dependency. With coveralls, test coverage report created by the Jest can be easily sent to the Coveralls app servers [3].

For the CI and testing workflow, a Node.js starter workflow template was used as the base for this project [5]. The workflow will do a clean installation of node dependencies, cache/restore them, and run tests across different versions of node. Build process of the source code was omitted since the actual web store application is not implemented in the scope of this assignment.

The workflow will be triggered on push events and pull requests to the main branch of the repository. The jobs to run on Linux, using the GitHub-hosted ubuntu-latest runners.

The starter workflow includes a matrix strategy that builds and tests the code with different Node.js versions. Our workflow uses Node versions 16.x and 18.x. The 'x' matches the latest minor and patch release available for a version. The jobs of the workflow are run on each of the specified Node versions separately. We had issues with Node version 14 when running our tests and not enough time to debug and fix the issue, so Node version 14 was dropped from the workflow.

The workflow was then supplemented with a Coveralls GitHub Action integration [2]. With it, the coverage report is sent to the Coveralls app every time the GitHub actions run.

The workflow runs the following commands:

```
$   npm ci
```

which does a clean install of the applications dependencies and

```
$   npm run test:coverage
```

which runs the Jest unit tests with coverage and coveralls. Then the workflow posts the test suite's LCOV coverage data to coveralls.io. The test coverage report of the project can be viewed from the Coveralls app [4].

The following picture shows the GitHub Actions workflow running the tests on push event.

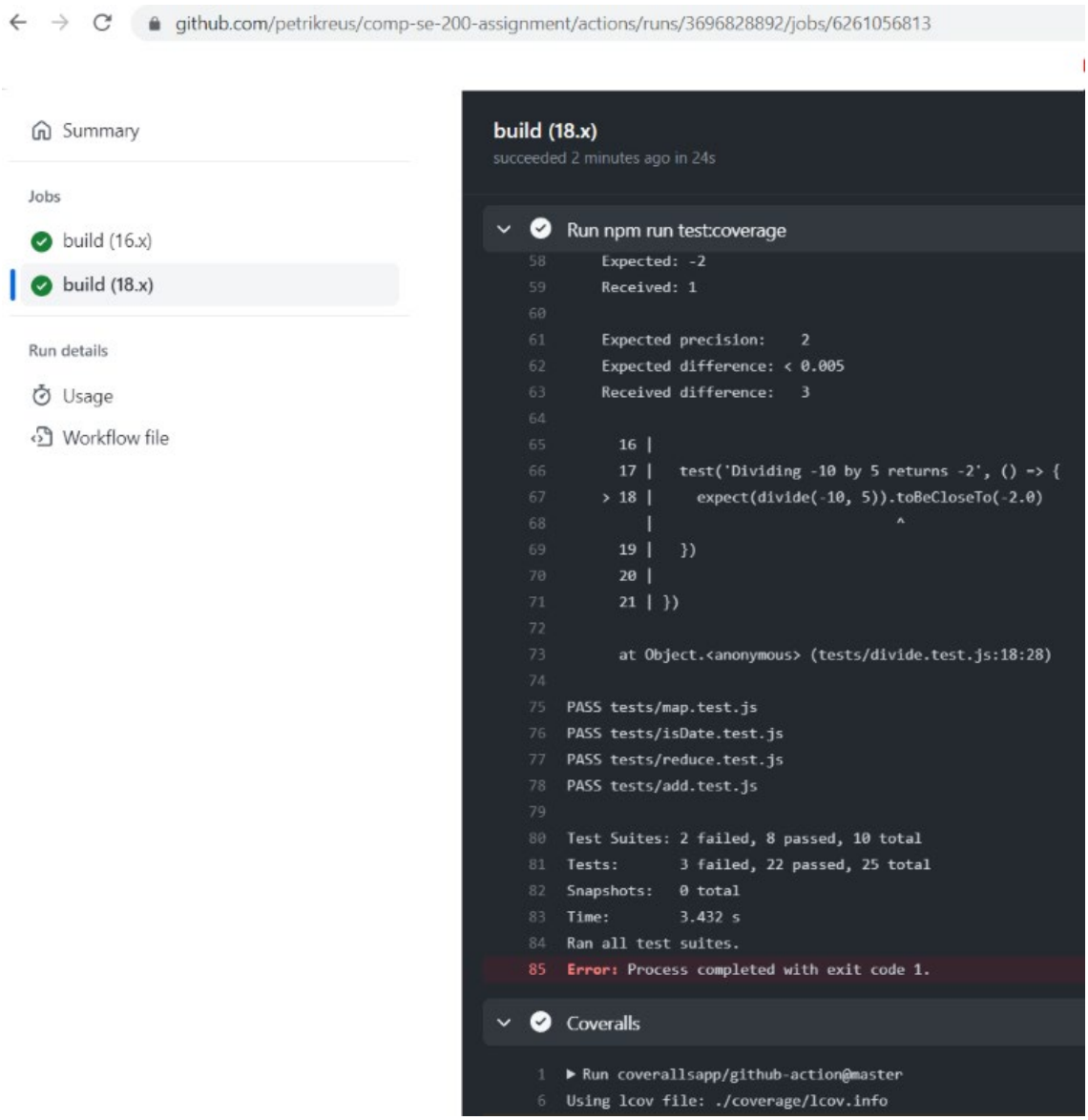

Figure 1. An example of GitHub Actions running.

4. FINDINGS AND CONCLUSION

The designed test cases returned three errors. Two tests in file divide.js are related to the same programming error and therefore are combined into one bug report. The third error is more of a design error and was found in the words.js file. The following tables describe the bugs found.

Bug report	
ID **BR-001**	
Title	Programming error in helper function divide. Division uses wrong variable as the dividend.
Related Test ID(s)	TC001, TC004
Description	In row number 16, where division is calculated, clearly the wrong variable (divisor) is used as the dividend. This causes the function to always divide itself with itself and return 1 with integer inputs.
Reported by	kgolsu
Reported Date	14.12.2022
Name of the Program	COMP.SE.200-2022-2023-1 Utils function library
Testing configuration	Node v18.12.0, Jest 29.1.2, Coveralls 3.1.1, Windows 11
Report type	Programming error
Failure repeatable	Yes
Seriousness	Major
How to repeat	Locally: npm test
Recommended fix	Change divisor / divisor to dividend / divisor on line 16 of divide.js
Assigned to	otula
Status	New
Comments	Function is used to show price information to the user and that is why Seriousness is determined to be major. Fix should be straightforward to implement.

Bug report	
ID **BR-002**	
Title	Possible design error in the file words.js, when words() is used with a string as a second parameter
Related Test ID(s)	TC022
Description	When the words function is given a second parameter as a string, the function returns an unexpected array.

Reported by	petrikreus
Reported Date	15.12.2022
Name of the Program	COMP.SE.200-2022-2023-1 Utils function library
Testing configuration	Node v16.17.0, Jest 29.1.2, Coveralls 3.1.1, Ubuntu 20.04.5 LTS (WSL2)
Report type	Design error
Failure repeatable	Yes
Seriousness	Minor
How to repeat	Locally: npm test
Recommended fix	1. Force the usage with RegExp by throwing an error when a string or number is used as a second parameter. OR 2. Check if the second parameter is a string. If it is, escape special characters to avoid unexpected results and return first index of the result array: string.match(pattern)[0]. See: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/match#a_non-regexp_as_the_parameter
Assigned to	otula
Status	New
Comments	The main use case of the function is expected to be without the second parameter, or the second parameter being a RegExp. That is the reason for minor seriousness.

The tests show that two files out of the total of ten files under testing have bugs in them. One bug which is major and one which is minor. Based on these findings, the testing did not pass our criteria, which was a 100% pass rate. The proposed fixes are estimated to require only minor effort, however. The major bug is most likely due to a programming error and the minor bug is more of a conceptual design error which should be defined unambiguously.

The application and the utility library are not fully tested. Our tests covered only a fraction of the utility library. Moreover, this testing did not include any integration, system, or acceptance tests. Since bugs were already found at the unit level suggests that additional higher-level tests would most likely reveal additional problems.

In conclusion, the overall quality of the utility library is difficult to estimate since the tests covered only a fraction of the whole library. However, based on our findings, our best

estimation is that the library has bugs in 20% of its code base, and 10% of the code base may include major bugs. Building on this estimation, we posit that found bugs should be fixed and further testing should be considered before using the utility library in production.

REFERENCES

[1] Coveralls.io. (2022). Retrieved from https://coveralls.io

[2] Coveralls GitHub Action. (2022). Coveralls GitHub Action. *GitHub Marketplace*. Retrieved from https://github.com/marketplace/actions/coveralls-github-action

[3] Coveralls. (2022). node-coveralls. *npm*. Retrieved from https://www.npmjs.com/package/coveralls

[4] Coveralls Report. (2022). Retrieved from https://coveralls.io/github/petrikreus/comp-se-200-assignment?branch=main

[5] GitHub. (2022). Building and testing Node.js. *GitHub Docs*. Retrieved from https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-nodejs

[6] Jest. (2022). https://jestjs.io/

[7] Kreus, P. & Sund, O. (2022). Comp-se-200-assignment. *GitHub*. Retrieved from https://github.com/petrikreus/comp-se-200-assignment?branch=main

[8] Node.js. (2022). https://nodejs.org