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](#_Toc122001957)

[2. Definition of done 2](#_Toc122001958)

[3. tests & CI Pipeline 3](#_Toc122001959)

[3.1 Description of the tests 3](#_Toc122001960)

[3.1.1 Updated tests 4](#_Toc122001961)

[3.2 Running the tests locally 6](#_Toc122001962)

[3.3 CI pipeline and coveralls 6](#_Toc122001963)

[4. findings and conclusion 8](#_Toc122001964)

[REFERENCES 9](#_Toc122001965)

List of Symbols and abbreviations

AUT Application Under Test, E-Commerce Store

npm Node Package Manager

nvm Node version manager

# 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. Additionally, some things were missing from the plan and are described here for the sake of clarity.

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.

# 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 as 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.

# 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. Third, the used CI pipeline and overalls integration is explained. Finally, a list of bugs is reported.

## Description of the tests

Unit tests were made using Jest JavaScript Testing Framework. Jest was selected because it is easy to use and there is little or no need for configuration. It is also easy to generate 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. |

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 filename is the utility library file under testing. The individual tests are pretty straightforward in of themselves. 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 are made to the tests since the test plan. Regarding the utility library file words.js, there were a few misunderstandings of how the function should work. A comparison of the original test and updated tests 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 | - |

## Running the tests locally

The tests can by run locally from a terminal by cloning the GitHub repository and then navigating to the project root folder. First install required dependencies by running the command:

$ npm install

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

$ npm test

You can run the tests with 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

## CI pipeline and coveralls

Node.js starter workflow template (By GitHub Actions) was used as the base for the continuous integration and testing workflow for this project. The workflow will do a clean installation of node dependencies, cache/restore them and run tests across different versions of node. Build of the source code was obviously 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 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 picture below shows GitHub Actions workflow running the tests on push event.

Graphical user interface, text

Description automatically generated

Figure 1. An example of GitHub Actions running.

From the Coveralls report you can view the test coverage information (Coveralls report, 2022).

# findings and conclusion

# REFERENCES

1. Jest. (2022). <https://jestjs.io/>
2. Node.js. (2022). <https://nodejs.org>
3. nvm. (2022). <https://github.com/nvm-sh/nvm>
4. Testing Library. (2022). <https://testing-library.com/docs/react-testing-library/intro/>
5. VMware. (2022). <https://www.vmware.com/products/workstation-player.html>