Tampere University Unit of Computing Sciences
COMP.SE.200-2022-2023-1 Software Testing

An E-commerce Store

Test Report

https://github.com/joonas175/COMP.SE.200-2022-2023-1

150321684, Joonas Salojärvi
H262941, Tomi Jokkeenhaara

Contents

Definitions, acronyms and abbrevations

CI	Continuous Integration
Pipeline	A long-running job consisting of multiple stages

1 Introduction

This document is the testing report for a utility library for an e-commerce store.

The purpose of this document is to explain the testing methods used and the findings from them. The production readiness of the whole library and the e-commerce application is estimated in this document.

Testing methodology is explained in chapter 2, along with tools used. Chapter 3 goes through the findings, as well the estimates for production readiness and tips on what to consider going forward.

2 Tests and CI pipeline

2.1 Tests

For every function selected for testing, multiple unit tests have been written. The unit tests cover typical use cases, with both positive and negative test cases.

Functions selected for testing: *defaultTo, filter, get, isArrayLike, isBoolean, isEmpty, keys, map, reduce, words*.

The minimum requirement for features to be considered passed, was that the functions associated passed the positive test cases with expected input. In addition, obvious negative test cases were tested. Goal for coverage was approx. 90% of statements.

Unit tests use Jest testing library [1]. The tests are automatically run on each commit. Coverage from the tests is reported to Coveralls [2]. The project is hosted publicly on GitHub, and all the tests and workflows can be viewed on the repository.

2.2 CI

GitHub Actions [3] was used for continuous integration (CI) pipeline. The pipeline runs on every pull request and commit. It uses npm to install dependencies and run Jest. The coverage report from Jest is sent to Coveralls, which can be used to view coverage reports.

2.3 Running tests locally

The tests can be run locally using a CLI and following these steps:

1. Navigate to the root of the repository
2. Run ‘npm install’ (only required for the first time)
3. Run ‘npm test’

The test results and a coverage report are displayed in output of the console. The test report is also saved in lcov-format in /coverage folder, which is automatically generated.

2.4 Coverage

The tests cover 92.1% of the statements in /src.

File		Statements		Branches		Functions		Lines	
src		92.1%	70/76	68.36%	67/98	95%	19/20	92%	69/75
src/.internal		87.34%	138/158	59.55%	81/136	95%	19/20	87.34%	138/158

3 Findings and conclusions

3.1 Bugs and issues

Bugs are reported to the project's repository issues page [4]. Bug report template is in use, and it is described in appendix A.

3.1.1 filter.js

The *filter* function should filter an array based on the given predicate function. It should return only values, where the predicate function returns true. It doesn't work as expected when there are no matches. The function should return array with a length of 0, but it returns an array with length 1. The one element in the array is an empty array. The empty array may cause problems later, as it probably differs from the objects we are trying to filter.

3.1.2 words.js

The *words* function should handle non-alphanumeric ascii-characters separated by spaces as their own words, but it ignores these characters completely. For example, the string 'first & second' should be split into three strings, '*first*', '*&*' and '*second*'. Instead, an array with only two strings, '*first*' and '*second*' is returned.

3.2 Estimate of library quality

The tested library is of intermediate quality. All of the provided functions are well documented, but some bugs were found even with limited unit testing. This suggests that many more bugs with unknown severity likely exist in the library and more comprehensive testing and improvements based on the results should be done before using the library in production.

As an additional note, some functions provided in the library, such as *reduce, map,* or *keys* might not be worthwhile to offer as part of an external library, because modern JavaScript already offers these functionalities.

3.3 Suitability of library for E-Commerce

Because the tested library is not reliable enough to be used in production, it should not be used in the development of the E-Commerce application. Even though the E-Commerce application is also not ready for production, it is unknown at what paces each of these might reach production readiness and relying on the library would only introduce unnecessary risks to the development of the E-Commerce application. Not only could it introduce bugs, the implementation of the library and its functions might change at any point, which might then lead to new bugs introduced to the E-Commerce application.

Therefore, a more mature library with more comprehensive testing should be identified and used in the development of the E-Commerce application instead.

3.4 Test coverage

The library was tested extensively for the function that were selected. It can't be considered fully tested, as a lot of functions were left out testing.

Unit tests for utility functions don't give a good estimate on how the actual application runs. More testing needs to be done. For the e-commerce application, E2E testing should be implemented, as well as system testing. System testing can be done manually, or as an automated process with a library like Robot Framework [5].

4 References

[1] https://www.npmjs.com/package/jest (Test framework)
[2] https://www.npmjs.com/package/coveralls (Test coverage service integration)
[3] https://github.com/features/actions (GitHub Actions)
[4] https://github.com/joonas175/COMP.SE.200-2022-2023-1/issues (Issues)
[5] https://robotframework.org/ (Robot Framework)

Appendix A

GitHub Issue template

Description:
*** short description of the bug ***

Severity estimate:
[ ] Severe
[ ] Moderate
[ ] Minor

*** to be corrected by a dev ***

Expected behaviour:
*** what is the expected behaviour and result of the action ***

Actual behaviour:
*** what actually happens ***

Steps to reproduce (optional):
*** how to reproduce error, if there are some additional steps required besides the actions mentioned in expected behaviour ***

Environment:
*** short description of the environment where the bug was encountered, in this case usually a browser ***