Test report

Comp.SE.200 Software Testing

TOMMI ILMONEN 253543, VILLE SAARINEN 252987
GITHUB: HTTPS://GITHUB.COM/VIKIDI/OHJTEST-SHALOM

Table of Contents

Definitions and acronyms

CD	Continuous Deployment
CI	Continuous Integration
CLI	Command Line Interface
npm	Node Package Manager, registry for JavaScript third party packages

1. Introduction

This document is the test report for the provided utility library [1]. The main purpose of this report is to show the quality of the library, ie. how did the library perform and what errors or unwanted functionality was found. Also, the other essential goal is to show what parts of the library was tested and how. The report's findings are constructed based on the part 1 of the assignment, which was the test plan for this library [2].

Like it was stated in the test planning phase [2], a project (or in this case a library) cannot be tested entirely nor can it be proven to be error free. However, since the library is compact and straightforward, a big portion of it was tested and covered (Over 90%), which is seen in this report. To clarify, big portion means in this case that every function of the library was tested, but not necessarily every branch or code line of a certain function.

Like stated above, the most important aspect of this document is to show the current quality of the given library. This was done by implementing tests having high quality and wide coverage (both positive and negative testing) to the contents of the library. The test suites were constructed by the test plan documentation [2], so that quality and coverage of tests are as high as possible. To document the findings of missing or unwanted functionality, errors and missing documentation are clearly and unambiguously labeled and categorized according to the test plan.

According to the test plan the target of testing, the library, consists of simply helper functions. Therefore according to the plan, only unit testing was done because there is nothing to be integrated and therefore integration testing is somewhat obsolete.

NOTE: The test plan was updated based on assistant feedback before making this report and referring test plan in this document means the updated plan version [2].

2. Test cases

Test plan [2] describes that all functions will be tested with unit tests. All of these unit tests were carried out in the tests. The tests composed of positive tests, negative tests, and manual documentation tests. Documentation was tested with glass box testing, mainly for missing or incorrect documentation compared to the functionality that resulted from the unit tests.

The library contains 43 functions, therefor there is 43 test suites. These include total of 338 test cases.

3. Findings and conclusions

This section describes issues found from the testing and concludes the quality of the tests and the library itself.

3.1. Issues found

The original test report created by Jest can be found from projects AWS S3 bucket [3]. Below is listed all functions that contained issues. Functions that passed the tests fully will not be included in this listing. After the function name will be given test case results for that function.

add (10 passed)

Label: Warning. Missing documentation. Exceptions are not documented.

Label: Warning. Missing documentation. Handling of incorrect values not documented.

at (6 passed)

Label: Warning. Missing documentation. Erroneous input situations are not documented.

camelCase (3 failed, 7 skipped)

Label: Critical. Function inserts always space character (“ “) as first in the resulting string.

Label: Warning. First letter is capitalized if the string begins with numbers. Not documented if this is intended behavior.

Label: Error. Incorrectly handled. Null input returns null as string.

capitalize (5 passed, 2 failed)

Label: Error. Spaces do not get removed and therefor word will not be capitalized.

Label: Error. Incorrectly handled. Returns null as string with null input.

castArray (5 passed, 2 failed)

Label: Error. If array is passed as parameter, function returns it inside another array.

Label: Error. Incorrectly handled. If no parameters are passed, function returns array with undefined inside.

ceil (11 passed, 1 failed)

Label: Error. Incorrectly handled. Null value does not return NaN as is the case with undefined input.

chunk (3 passed, 2 failed, 4 skipped)

Label: Critical. Function add some what randomly undefined to the final array.

Label: Error. Not handled. If chunk size is null function returns empty array.

clamp (5 passed, 6 failed)

Label: Warning. Missing documentation. Functioning in case of incorrect bound values is not documented.

Label: Critical. Function always clamps to the lower bound.

Label: Error. Not handled. Null and undefined values are not handled.

compact (3 failed)

Label: Critical. Expected values have always first entry removed.

countBy (3 failed)

Label: Critical. Received values have too low counts by one.

defaultTo (7 passed, 1 failed)

Label: Warning. Missing documentation. Documentation missing for return values with no, null or undefined arguments.

Label: Critical. Given arguments defaultTo(NaN, 10) the function should return first one that is not NaN, ie. 10. However, NaN is returned.

defaultToAny (7 passed, 1 failed)

Label: Warning. Missing documentation. Documentation missing for return values with no arguments.

Label: Critical. Given arguments defaultToAny(null, undefined, NaN, 1) the function should return first one that is not one of null, undefined or NaN. However, NaN is returned.

divide (4 passed, 4 failed)

Label: Critical. Does not work at all, returns 1 for every single division.

drop (11 passed)

Label: Warning. Not documented. Strings do work as well but are not documented.

endsWith (8 passed)

Label: Warning. Not documented. Thrown exceptions are not documented for invalid arguments.

Label: Warning. Not documented. Arrays seem to work as well, but are not documented

eq (4 passed, 1 failed)

Label: Error. Incorrect documentation. By documentation eq('a', Object('a')) should be false, but it translates to true.

filter (3 passed, 3 failed)

Label: Warning. Missing documentation. Thrown exceptions not documented.

Label: Error. Filtering all elements returns array inside of array i.e. [[]] instead of [].

Label: Error. Filtering all elements of null array returns array inside of array i.e. [[]] instead of [].

get (6 passed, 1 failed)

Label: Critical. Getting the object itself returns undefined.

map (7 passed)

Label: Warning. Missing documentation. Return values for unsupported types not documented, even though they are handled in some cases.

Label: Warning. Missing documentation. Thrown exceptions not documented.

memorize (5 passed)

Label: Warning. Missing documentation. Undocumented exception thrown.

reduce (10 passed)

Label: Warning. Missing documentation. Documentation missing for return values of empty or null arrays/objects.

slice (10 passed)

Label: Warning. Missing documentation. No documentation on wrong argument type return values.

toFinite (10 passed)

Label: Warning. Missing documentation. No documentation on NaN parameter return values, but one can assume it is 0.

Label: Warning. Missing documentation. No documentation on arrays, though they do work.

toInteger (9 passed)

Label: Warning. Missing documentation. No documentation on NaN parameter return values, but one can assume it is 0.

Label: Warning. Missing documentation. No documentation on arrays, though they do work.

toNumber (12 passed)

Label: Warning. Missing documentation. No documentation on arrays, even though they do work.

toString (6 passed, 4 failed)

Label: Error. Incorrect documentation. Functionality does not match documentation: By documentation, arguments of null or undefined should return empty string. Instead, 'null' and 'undefined' returned.

Label: Critical. Objects do not work and documentation what should be returned.

upperFirst (8 passed)

Label: Warning. Missing documentation. Handling of incorrect values (null, undefined) not documented.

words (9 passed)

Label: Warning. Missing documentation. Exceptions not documented.

Label: Warning. Missing documentation. Handling of incorrect values (null, undefined) not documented.

3.2. Quality of tests

Quality of the tests can be viewed from coveralls [4]. The quality mainly composes of testing coverage. The tests cover 190 branches of 224 branches in the library, which equals 84,82 % coverage on branches. On the other hand, tests cover 195 relevant lines of code from total of 201 lines in the library. This equals to line coverage of 97,01 % coverage. In total the tests have combined coverage of 90,59 %.

Most of the functions have over 95 % testing coverage, which was one condition for functions tests to be considered passed. Another requirement is that all non-equivalent test cases are created. Here is listed all functions, which test coverage is not 95 % or above: *chunk, eq, isBuffer, isDate, isEmpty, isTypedArray, memoize, slice* and *toNumber*. This results in 9 functions out of 43 (20,93 %) which do not have testing coverage good enough to be considered passing. This means that testing is sufficient enough on 79,07 % of the functions to be able to be considered passed.

3.3. Quality of the library

Travis CI shows that build is not passed. This happens if even one test fails. This does not automatically mean that library is broken, but neither does possible passing mean that the library would be error free.

There was total of 15 functions out of 43 that did not have any issues in the testing. These are *difference, every, isArgument, isArrayLike, isArrayLikeObject, isBoolean, isBuffer, isDate, isEmpty, isLength, isObject, isObjectLike, isSymbol, isTypedArray* and *keys*. For the function to be considered passed functionality vise it needs to have zero documentation related issues

and all tests need to pass. This means that the library has functionality vise passing rate of 34,88 % for functions.

Total amount of different label issues is:

- Critical: 10
- Error: 12
- Warning: 24

This results in total of 46 issues. Notable is that 21,74 % of the issues are critical. This means that out of 43 functions 10 is unusable (23,26 %).

Documentation vise there is total of 26 issues; 24 is about missing documentation and 2 is about incorrect documentation. This means that the library also needs a lot more investing to documentation.

These results combined with the requirements for testing coverage in last chapter will provide us with full requirements for passed function. Fully passed functions therefor are *difference, every, isArgument, isArrayLike, isArrayLikeObject, isBoolean, isLength, isObject, isObjectLike, isSymbol* and *keys.* This means that 11 out of 43 functions are considered passed. This results in total passing rate of 25,58 % for functions.

Test plan has requirements for passed tests:

1. all functions pass their tests
2. overall test coverage is at least 95%
3. there are no documentation related issues on the library itself

This means that the tests are not considered to be passed, and there for the library cannot be considered passed.

The tests need some improvement (passing rate 79,07 %) but the main issue is in the library (passing rate 34,88 %, 46 issues). This strengthens even more the fact that the library is not considered passed.

4. References

[1] COMP.SE.200-2020-assignment, library under testing, GitHub repository, website. Available (referenced 7.12.2020): https://github.com/tgcslearningtech/COMP.SE.200-2020-assignment/tree/master/src

[2] Test plan, Testing document, web document. Available (referenced 7.12.2020): https://github.com/vikidi/OhjTest-Shalom/blob/main/Documents/test_plan.pdf

[3] Raw test report, Jest report. Available (referenced 7.12.2020): https://ohj-test-shalom.s3.amazonaws.com/vikidi/OhjTest-Shalom/19/19.1/coverage/jest-result.txt

[4] Coverage analysis, Coveralls. Website. Available (referenced 7.12.2020): https://coveralls.io/github/vikidi/OhjTest-Shalom