TEST REPORT: Open-Source JavaScript Utility Library for E-Commerce Application

COMP.SE.200-2020-2021-1 Software Testing

EVERGREEN PROSPER - 050542738
MOHSIN QADEER - 050364592
ST_GROUP GitHub Repo

Table of Content

List of definitions, acronyms and abbreviations used in the document.

Test Scenario - any functionality that can be tested. It is a collective set of test cases

Test Case - a set of actions executed to verify a particular feature or functionality of your software application.

LCOV - lcov is a graphical front-end for GCC's coverage testing tool gcov. Gcov is a source code coverage analysis and statement-by-statement profiling tool.

BDD - Behavior Driven Development

TDD - Test Driven Development

UI - User interface

Introduction

The Test Plan is designed to prescribe the scope, approach and resources of all testing activities of an open-source JavaScript utility library. The library is meant to be used in a front end application built with React. The application is an E-commerce store selling food products from various small producers

The plan identifies the items to be tested, the features to be tested, the types of testing to be performed and the resources required to complete testing, and the risks associated with the plan.

1. Test Strategy

1.1 Scope of Testing

1.1.1 Features to be tested

All the functions of the open-source library which are required for the E-commerce application should be tested based on the following requirement.

E-commerce application requirement

Feature	Applicable role	Description
Search product	User	Users can search products by category, price, producer and various other criteria.
Add product to cart	User	Products can be added to a shopping cart.
Auto update cart		Shopping cart automatically updates and shows the total price.
Add product	Producer	The food producers can add their products via a previously created portal
Handle missing values	Producer	The producers can leave some fields blank if they do not want to specify some attributes like category or calories and these missing values must be handled.
product descriptions look similar		making sure that the product descriptions look similar i.e. the first word of a sentence starts with an upper-case letter and that prices are shown with two decimal accuracy.

In the table below there are three colors mentioned, each of them define the scenario of the test cases that we have done.

Green :-

The test cases that are in the green color are those that passed successfully and they also passed the coverage test.

Red :-

The test cases that are mentioned in the test plan but weren't implemented because they were found unnecessary for the requirement of the application.

Yellow :-

Yellow are the test cases that are found with errors/bugs.

Unit Testing

Function	Description
add	Adds two numbers
at	Creates an array of values corresponding to 'paths' of 'object'
camelCase	Converts 'string' to camel case
capitalize	Converts the first character of string to upper case and the remaining to lower case.
ceil	Computes `number` rounded up to `precision`. (Round up: the smallest integer greater than or equal to a given number.)
clamp	Clamps `number` within the inclusive `lower` and `upper` bounds.
compact	Creates an array with all falsey values removed. The values `false`, `null`, `0`, `""`, `undefined`, and `NaN` are falsey.
countBy	Creates an object composed of keys generated from the results of running each element of `collection` thru `iteratee`. The corresponding value of each key is the number of times the key was returned by `iteratee`. The iteratee is invoked with one argument: (value).
defaultTo	Checks `value` to determine whether a default value should be returned in its place. The `defaultValue` is returned if `value` is `NaN`, `null`, or `undefined`.
defaultToAny	This method is like `defaultTo` except that it accepts multiple default values and returns the first one that is not `NaN`, `null`, or `undefined`.
difference	Creates an array of `array` values not included in the other given arrays using [`SameValueZero`](http://ecma-international.org/ecma-262/7.0/#sec-samevaluezero) for equality comparisons. The order and references of result values are determined by the first array.

divide	Divide two numbers.
drop	Creates a slice of `array` with `n` elements dropped from the beginning.
endsWith	Checks if `string` ends with the given target string.
eq	Performs a [`SameValueZero`](http://ecma-international.org/ecma-262/7.0/#sec-samevaluezero) comparison between two values to determine if they are equivalent.
every	Checks if `predicate` returns truthy for all elements of `array`. Iteration is stopped once `predicate` returns falsey. The predicate is invoked with three arguments: (value, index, array).
filter	Iterates over elements of `array`, returning an array of all elements `predicate` returns truthy for. The predicate is invoked with three arguments: (value, index, array).
get	Gets the value at `path` of `object`. If the resolved value is `undefined`, the `defaultValue` is returned in its place.
isArgument	Checks if ‘value’ is likely an ‘argument’ object
isArrayLike	Checks if `value` is array-like. A value is considered array-like if it's not a function and has a `value.length` that's an integer greater than or equal to `0` and less than or equal to `Number.MAX_SAFE_INTEGER`.
isArrayLikeObject	This method is like `isArrayLike` except that it also checks if `value` is an object.
isBoolean	Checks if `value` is classified as a boolean primitive or object.
isBuffer	Checks if ‘value’ is a buffer
isDate	Checks if `value` is classified as a `Date` object.
isEmpty	Checks if `value` is an empty object, collection, map, or set.
isLength	Checks if `value` is a valid array-like length.
isObject	Checks if `value` is the [language type](http://www.ecma-international.org/ecma-262/7.0/#sec-ecmascript-language-types) of `Object`. (e.g. arrays, functions, objects, regexes, `new Number(0)`, and `new String('')`)
isObjectLike	Checks if `value` is object-like. A value is object-like if it's not `null` and has a `typeof` result of "object".
isSymbol	Checks if `value` is classified as a `Symbol` primitive or object.
isTypedArray	Checks if `value` is classified as a typed array.
keys	Creates an array of the own enumerable property names of `object`.

map	Creates an array of values by running each element of `array` thru `iteratee`. The iteratee is invoked with three arguments: (value, index, array).
reduce	Reduces `collection` to a value which is the accumulated result of running each element in `collection` thru `iteratee`, where each successive invocation is supplied the return value of the previous. If `accumulator` is not given, the first element of `collection` is used as the initial value. The iteratee is invoked with four arguments: (accumulator, value, index|key, collection)
slice	Creates a slice of `array` from `start` up to, but not including, `end`.
toFinite	Converts `value` to a finite number.
toInteger	Converts `value` to an integer.
toNumber	Converts `value` to a number.
toString	Converts `value` to a string. An empty string is returned for `null` and `undefined` values. The sign of `-0` is preserve
upperFirst	Converts the first character of `string` to upper case.
words	Splits `string` into an array of its words.

1.1.2 Functions/Features not to be tested

These feature are not be tested because they are not included in the scope to be tested

- Checkout process: it is handled with a third-party solution.
- Files in the folder .internal which is not part of the testing process and should be excluded from any plans, test reports and coverage reports.
- memoize.js, chunk.js and castArray.js will also be excluded as they do not play any role in the functionalities required of the library

1.2 Test Type

For the project, 2 types of testing should be conducted.

- Unit Testing: Each function is tested using the black box system
- Integration Testing (Individual software modules are combined and tested as a group)

1.3 Test Logistics

1.3.1 Who will test?

The project should be tested by members of the group as intended by the course

1.3.2 When will the test occur?

The tester will start the test execution when all the following inputs are ready

- Library is available for testing
- Test case specification is created
- Test environment is built

2. Test Objective

The test objectives are to verify the functionality of the open-source JavaScript utility library, the project should focus on testing the operation such as Users search product, Users add product to cart, Auto update cart, Producer to add product, Handle missing values, and Product descriptions to look similar...etc. to guarantee all these operation can work normally in real business environment.

3. Test Criteria

3.1 Test Result Format

The test result should contain as much details as possible from the format provided

ID: Title – a compact but descriptive

Description	- More detailed than the title, explaining what happens.
Test case	- If specific test case was used, refer to it
Steps to reproduce	- Shortest possible steps to reproduce the bug - Explain clearly. Remember that the developer might not be using for example the UI as often as the testers do, so be precise.
Input	- Inputs needed to reproduce the bug - Good idea to provide even if they could be deduced from description. But if the developer can just copy/paste, it makes reproducing error easier.

Expected outputs / results	- What was supposed to happen - If necessary, refer to specifications / user stories.
Real results	- What actually happened. - Outputs, error messages, other descriptions of what went wrong - Can include references to screenshots, logs etc
Other anomalies	- If there was anything else weird that should be noted
Severity of the bug, consequences	- Hopefully your project has a way to rank bugs, use that. - Include also bit more explanation of the consequences. How this affects users or the business cases. Is there easy work around available etc…
Testing environment	- Necessary information of the testing environment - Program version
Can the bug be reproduced	
Analyses of the source of defect	- If the tester has a good idea of what causes the bug, that can be mentioned. - However, better to leave empty if the information is not useful
Effects on testing	- For example, if this is blocking large amount of other tests

3.2 Defects Classification

If bugs or issues are found, they must be classified as

3.2.1 Software defects by severity

- **Critical defects** usually block an entire system on module's functionality, and testing cannot proceed further without such a defect being fixed.
- **High-severity defects** affect key functionality of an application, and the app behaves in a way that is strongly different from the one stated in the requirements.
- **Medium-severity defects** are identified in case a minor function does not behave in a way stated in the requirements.

- **Low-severity defects** are primarily related to an application's UI and may include such an example as a slightly different size or color of a button.

3.2.2 Software defects by priority

- **Urgent defects** need to be fixed within 24 hours after being reported. Defects with a critical severity status fall in this category. However, low-severity defects may be classified as high-priority as well.
- **High-priority defects** are the errors that must be fixed in an upcoming release in order to meet the exit criteria.
- **Medium-priority defects** are the errors that may be fixed after an upcoming release or in the subsequent release.
- **Low-priority defects** are the errors that do not need to be fixed in order to meet the exit criteria but require fixing before an application becomes generally available.

3.3 Suspension Criteria

If 40% of test cases failed, suspend testing until all the failed cases are fixed.

3.4 Exit Criteria

Specifies the criteria that denote a successful completion of a test phase

- Run rate is mandatory to be 100% unless a clear reason is given.
- Pass rate is 80%, achieving the pass rate is mandatory.

4. Resource Planning

4.1 Test Environment

For the purpose of the testing, this tools will be used

Tool	Description
Mocha	Simple, flexible, fun JavaScript test framework for Node.js & The Browser

	(https://www.npmjs.com/package/mocha)
Mocha-lcov-reporter	The mocha-lcov-reporter is a LCOV reporter for mocha.(https://www.npmjs.com/package/mocha-lcov-reporter)
Chai	a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework. (https://www.npmjs.com/package/chai)
Coverall	a tool that measures test code coverage for JavaScript programs.(https://www.npmjs.com/package/coveralls)
Github	provides hosting for software development and version control using Git.(https://github.com)
Travis CI	a hosted continuous integration service used to build and test software projects hosted at GitHub and Bitbucket. (https://travis-ci.org)

"These tools are used on the basis of previous experience and I have good hands on experience with the tools. That is the reason I have selected these tools."

5. Findings and conclusions

Error Report

1. CamelCase.js

Description	- Converts `string` to camel case
Test case	- Should check if `value` is an empty object, collection, map, or set.
Steps to reproduce	- camelCase('Foo Bar') - function should return "fooBar" but returns " foobar" i.e. with a leading space.
Input	- string = 'Foo Bar'
Expected outputs / results	- function should return "fooBar"

Real results	- function returns “ fooBar” i.e. with a leading space
Severity of the bug, consequences	- **Medium-severity defects**
Testing environment	- nodejs, mocha, chai
Can the bug be reproduced	- Yes
Analyses of the source of defect	- The function adds an unnecessary space.
Effects on testing	- gives an error and fails

2. Clamp.js

Description	- Clamps `number` within the inclusive `lower` and `upper` bounds.
Test case	- Should clamp `number` within the inclusive `lower` and `upper` bounds.
Steps to reproduce	- clamp(-4, -5, 10); - function should return -4 but returns -5
Input	- number = -4, lower = -5, upper = 10
Expected outputs / results	- function should return -4
Real results	- function returns -5
Severity of the bug, consequences	- **High-severity defects**
Testing environment	- nodejs, mocha, chai
Can the bug be reproduced	- Yes
Analyses of the source of defect	- The function always returns -5. And compares the parameter number === number
Effects on testing	- gives an error and fails

3. divide.js

Description	- Divide two numbers.
Test case	- Should divide two numbers.
Steps to reproduce	- divide(10, 5); - function should return 2 but returns 1
Input	- dividend = 10, divisor = 2
Expected outputs / results	- function should return 2
Real results	- function returns 1
Severity of the bug, consequences	- **High-severity defects**
Testing environment	- nodejs, mocha, chai
Can the bug be reproduced	- Yes
Analyses of the source of defect	- The function always returns -5. Function performs divisor / divsor instead of dividend / dvisor.
Effects on testing	- gives an error and fails

Conclusions

- ***Overall quality of the test library was not so bad but there were some errors that we faced.***
- ***Yes it passed our test through it and the test cases are ready for production.***

- ***The estimated test coverage was 95%. Library was fully tested as far as we can do it. So, there shouldn't be any more testing required.***

References

Chai Core Team. (2018). Chai Assertion Library. Retrieved from https://www.chaijs.com

Merwin, Nick. (2019). Test coverage service integration. Retrieved from https://www.npmjs.com/package/coveralls

Hiller, Christopher. (2011). Mocha Test Framework. Retrieved from https://www.npmjs.com/package/mocha

Looman, Steven. (2016). LCOV reporter for Mocha. Retrieved from https://www.npmjs.com/package/mocha-lcov-reporter

GitHub, Inc. (2007). Github: Where the world builds software. Retrieved from https://github.com/

Travis CI community. (2011). Travis CI - Test and Deploy your code with Confidence. Retrieved from https://travis-ci.org/getting_started

COMP.SE.200-2020-2021-1 Software Testing. (Lectures). (2020). Assignment - Part 1 - Test plan. Retrieved from https://moodle.tuni.fi/mod/assign/view.php?id=665313

Rungta, Krishna. (2018). TEST PLAN: WHat is, How to Create (with Example). Retrieved from https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html