TEST PLAN

Open-Source JavaScript Utility Library for an E-commerce application selling food products

COMP.SE.200-2021-2022-1 Software Testing

Olayinka Gold-292959

Ayodeji Ilori-050391549

Table of Contents

List of definitions, acronyms and abbreviations used in the document

Test case - a document with a set of test data, preconditions, expected results and post conditions to verify compliance against a specific requirement for the application

BDD – Behaviour Driven Development

TDD – Test Driven Development

UI – User Interface

Test Scenario – any functionality that can be tested, also called test condition or test possibility

Introduction

Test plan is a comprehensive document that lays out all the major activities associated with a particular testing project. It includes the scope of the project, goals, and objectives, testing cycle start/end dates, test environment, deliverables, risks to be handle. This test plan is aimed at all the testing activities of an open-source JavaScript utility library. The library is meant to be used in a front-end application built with React, React promotes functional programming and composition over inheritance. The application is an E-commerce store selling food products from various small producers.

In this plan, the items to be tested are identified, the features to be tested, type of testing to be done and resources required to complete testing as well as the risks involved with the plan.

1.0 Test Strategy

1.1 Scope of Testing

1.1.1 Features to be tested

In this, functions and features of the open-source library are requirements to be tested and their description.

Test Scenario

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

Table 1: Test Scenario

Function	Description
endsWith	Checks if `string` ends with the given target string.
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).
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.
add	Adds two numbers.
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`.
isSymbol	Checks if `value` is classified as a `Symbol` primitive or object.
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**).**
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('')`)

Table 2: Unit Testing

The functions above have been selected to be tested as they are vital parts of the application as it pertains to an e-commerce application. They are crucial to ensuring the application is at least bug free.

1.1.2 Features/Functions not to be tested

Some features are not to be tested as they are not included in the scope to be tested

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

1.2 Test Type

In this project, two (2) types of testing will be done

- Unit Testing: Testing is done at the source code level, interface, limit values, error handling, data structures, execution parts and loops. The blackbox system is used.

- Integration testing: Testing is done after units have been carried out.

1.3 Test Logistics

1.3.1 Who will test?

The project should be tested by members of the group

1.3.2 When to test

Testing will start when test items required for testing are ready, such as:

- Test environment where testing is to be done
- Test plan is done to show describe what will be tested and how
- Test cases are designed to narrow down things to be tested
- Library and other components are available for testing

2.0 Test Objective

Test objectives are to verify the functionality of the open-source JavaScript utility library, the project is aimed at testing operations such as user search product, user add product to cart, automatic update cart, producer add product, handle missing values, similar production descriptions, etc. to guarantee they behave exactly as it is intended in the business environment.

3.0 Test Criteria

3.1 Result Format

This contains detailed information about the test result based on the format provided

Description	Gives a more detailed information about what needs to be tested
Test case	If there are specific test case used, refer to it
Input	Inputs needed to reproduce the bug Good to provide to run test
Steps to reproduce	Shortest steps to reproduce the bug Give clear explanation so developer can follow and understand Provide logs if possible
Expected result/output	How it is intended to be Refer to user stories or specification
Real results/output	What exactly happened Outputs, error messages, logs and description of what went wrong
Other anomalies	Anything strange (certain behaviours) that needs notification
Severity of bug	Explain the bug severity encountered Include things likely responsible How easy it is to fix or if it affects other things
Test environment	Information of the environment test was carried out Versions involved
Analysis of the source of defect	Tester can give information about the cause of defect
Effects on testing	If the discovered bug prevents others tests from being carried out
Can the bug be reproduced	Reproducing the exact bug helps the developer to tackle the problem

Table 3: Result Format

3.2 Defect Classification

Bugs or issues must be classified for it to be handled accordingly

3.2.1 Software Defects by Severity

- Critical defects often affect functionality and prevents testing from going on, such defects need to be fixed
- High-severity defects affects the main functionality of the application and cause the application to behave differently from what was stated in the requirement
- Medium-severity defects are identified in case when a minor function does not behave in a way stated in the requirements
- Low-severity defects are mostly related to the applications user interface which might include size of text area or button size or colour

3.2.2 Software Defects by Priority

- Urgent defects need to be fixed within 24 hours after being reported. And the defect with critical severity falls into this category.
- High-priority defect are business critical that needs to be fixed with immediate effect before the next release to meet the criteria.
- Medium-priority defect are moderate level and needs to be fixed in the current release
- Low-priority defects shows that there is an issue but does not need to be fixed for the exit criteria

3.3 Suspension Criteria

If 40% of the test cases fails, then suspend all or portion of testing activities until they are fixed

3.4 Exit Criteria

The criteria or requirements which must be met to complete a specific task or test phase

- All tests planned have been run
- Verify if the level of requirement coverage has been met
- Pass rate is 80%, the pass rate is mandatory before project can be closed

4.0 Resource Planning

4.1 Test Environment

In this testing, the following test tools will be used

Tool	Description
Mocha	A simple, fun, and fun JavaScript test framework
Mocha-lcov-reporter	A reporter for Mocha to get test coverage data
Chai	BDD/TDD assertion library for node and the browser that can be delightfully paired with any JS testing framework
Coverall	Support for node JS that gives a great coverage
Github	Used for version and source control (https://github.com/)
Travis CI	a continuous integration service used to build and test software projects

Table 4: Testing Tools

References

Test scenario - https://www.guru99.com/test-scenario.html

Test case - https://www.tutorialspoint.com/software_testing_dictionary/test_case.htm

COMP.SE.200-2021-2022-1 Software Testing. (Lectures)- https://moodle.tuni.fi/course/view.php?id=18063&section=3#tabs-tree-start

COMP.SE.200-2020-2021-1 Software Testing. (Assignment) - https://moodle.tuni.fi/mod/assign/view.php?id=1207043

Mocha- https://www.npmjs.com/package/mocha

Mocha-lcov-reporter- https://www.npmjs.com/package/mocha-lcov-reporter

Chai- https://www.npmjs.com/package/chai

Coverall - https://www.npmjs.com/package/coveralls

Travis CI – https://www.travis-ci.com/