Test plan for  
the E-commerce store

**COMP.SE.200-2022-2023-1**

Contents

[1. Introduction 1](#_Toc117854010)

[2. Design 2](#_Toc117854011)

[2.1 Scope 2](#_Toc117854012)

[2.2 Scenarios 2](#_Toc117854013)

[2.3 Selected Source Files 8](#_Toc117854014)

[3. Tools 10](#_Toc117854015)

[3.1 General environment 10](#_Toc117854016)

[3.2 Testing tools 10](#_Toc117854017)

[4. Tests 11](#_Toc117854018)

[4.1 Test Cases 11](#_Toc117854019)

[4.1.1 Test cases for divide.js 11](#_Toc117854020)

[REFERENCES 12](#_Toc117854021)

List of Symbols and abbreviations

npm Node Package Manager

# Introduction

This document describes a test plan and a test design for the utility library of the front-end of the E-Commerce Store application. The aim is to give a detailed description of the test strategy, test objectives and test deliverables for the project.

First, descriptions of end-to-end scenarios are presented. Second, tools for the testing are defined with the actual tests. Finally, a Virtual Machine setup for the testing environment is illustrated.

# Design

In this section, we describe the general test design and its rationale. First, we define a scope for the test design. Second, the most important end-to-end scenarios of the application are specified. Third, we present the main components that we identified based on the scenarios. Finally, a selection of source files to be tested are introduced.

## Scope

The scope of the testing is to test 10 source files of the utility library, which consists of 43 source files in total. The amount to be tested is limited to 10 due to the time constraints. The test strategies are limited to unit test and integration tests, mostly since we do not have access to the application or any of its parts aside from the utility library source files. That means that many testing strategies are left out of the scope, such as usability testing, system testing and acceptance testing.

Additionally, the specification states that only the top level of the utility library is to be tested, hence the files under the “.internal/” folder are left out of the testing scope. Due to the nature of the files under test, the testing concentrates only on functional tests.

To select the source files to be tested, first a few end-to-end scenarios are detected based on the provided specification of the product. Next, main components of the application are analysed from the scenarios. Finally, a prioritization method is used, utilizing the scenarios and components, to figure out the most important source files that need to be tested.

## Scenarios

The testing is designed upon four main end-to-end scenarios identified from the application description. The scenarios described here are limited to four, since based on our evaluation, those cover the most important features of the application. Two of them are related to customer functionality and two to producer functionality. Following present these scenarios in a form of sequence diagrams.

Figure 1 describes user logging in, searching a product, adding a product to cart, and finally making a purchase. Figure 2 describes a new user registering an account and logging in. Figure 3 describes a producer logging in, adding a new product, and removing a product. Figure 4 describes a new producer registering a new account, waiting for it to be approved, and finally logging in after an approval.

Diagram

Description automatically generated

1. US1: User logs in and makes a purchase.

Diagram

Description automatically generated

1. US2: New customer registers and logs in.

Diagram

Description automatically generated

1. US3: Producer adds and removes a product.

Diagram

Description automatically generated

1. US4: New producer registers and logs in.

Based on these scenarios, we recognized five main functional components of the application. These components are:

* Log in / Registration (Customer & Producer)
* Product list (Customer & Producer)
* Shopping cart (Customer)
* Search (Customer)
* Product management (Producer).

## Selected Source Files

With the scenarios and components identified, we first used the MoSCoW method to prioritize the source files. Due to the strict time limitations, the prioritization were conducted on a highly critical manner, and most of the files ended up under the Won’t test category from the start. In the end, we narrowed the source files down to the 10 Must test files, and all other files were moved under Won’t test category.

The source files included in the testing are display’d in the following table (Table 1).

|  |  |  |
| --- | --- | --- |
| **Source File** | **Rationale for selection/Example use case** | **Related Scenario(s)** |
| add.js | Basic math function, may be used in multiple situations. | S1, S3 |
| at.js | May be used in many situations when getting data from an object. | S1, S2, S3, S4 |
| defaultTo.js | May be used when displaying and saving information. | S1, S3 |
| divide.js | Basic math function. May be used when displaying prices and in other calculations. Has a critical error in syntax. | S1, S3 |
| filter.js | May be used when displaying product lists. | S1 |
| isDate.js | May be used when displaying dates in product pages. Working with dates is a well-known challenge in JavaScript development. | S1, S3 |
| isEmpty.js | May be used when checking validity of form fields. | S1, S2, S3, S4 |
| map.js | May be used when converting array data into React components. | S1, S3 |
| reduce.js | May be used when displaying prices in cart. | S1 |
| words.js | May be used in parsing search parameters in product search. | S1 |

1. Selected source files with their selection rationale and relation to scenarios.

# Tools

To get the most out the testing, the production environment and the testing environment should be as similar as possible. However, the specification of the E-Commerce application does not describe the environment, aside from that its front-end uses React with the utility library provided. Thus, we will give our best guess, based on our research and experience, to describe a working environment for the basis of our tool selection. The environment description will concentrate only on the front-end React application and tools relating to the functional tests, which are specified in the next section.

## General environment

As the front-end of the application is built with React, and the utility library has a “package.json” file, it will most likely run on Node.js. Node.js is an open-source JavaScript runtime. It is a widely used base for various JavaScript applications. Node.js provides a useful package manager, Node Package Manager (npm), although other package managers could be used as well, such as pnpm or Yarn. With the npm, installing dependencies to the project is effortless, and controlling the versions of the dependencies is relatively easy.

The utility library provided will supposedly be installed to the production application with the npm, or similar.

## Testing tools

We picked Jest [1], a JavaScript testing framework, for unit and integration testing. Since the testable file/function count is low, Jest will work well. It works without much additional configuration and is easy to use. If the testable amount was higher, a deeper consideration and comparison of various testing libraries should be conducted.

The Jest framework was tested by creating a simple test case for the divide function of the provided utility library.

If we had access to the React application, we suggest that React Testing Library was to be used. It is a light-weight solution for testing React components. It provides utility functions with react-dom and react-dom/test-utils. React testing library is not a test runner or framework. Even though the library isn't specific to any framework it is recommended that React testing library would be used with Jest.

# Tests

* unit tests
* integration tests

## Test Cases

The following are short descriptions of designed test cases for the selected functionality.

### Test cases for divide.js

Test cases are selected using a mix of Equivalence partition method and limit value analysis. The divide function should perform a simple mathematical division operation and the expected behavior of the operation with given inputs should be quite straightforward.

# REFERENCES

1. Jest. (2022). <https://jestjs.io/>