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

An E-commerce Store

Test Plan

150321684, Joonas Salojärvi
H262941, Tomi Jokkeenhaara

Contents

1 Introduction 3
2 Scenarios 4
  2.1 Search items with a valid product name 4
  2.2 Search items with an invalid product name 4
  2.3 Adding items to the shopping cart 4
  2.4 Removing items from the shopping cart 5
3 Tools 6
  3.1 Considered tools 6
    3.1.1 c8 6
    3.1.2 chai 6
    3.1.3 coveralls 6
    3.1.4 GitHub Actions 6
    3.1.5 jest 6
    3.1.6 jest-chain 6
    3.1.7 jest-extended 7
    3.1.8 mocha 7
    3.1.9 mocha-lcov-reporter 7
    3.1.10 mochawesome 7
    3.1.11 jest-html-reporter 7
    3.1.12 websequencediagrams 7
  3.2 Selected tools 7
4 Tests 9
  4.1 Unit testing 9
    4.1.1 defaultTo.js 9
    4.1.2 filter.js 9
    4.1.3 get.js 9
    4.1.4 isArrayLike.js 9
    4.1.5 isBoolean.js 9
    4.1.6 isEmpty.js 9
    4.1.7 keys.js 10
    4.1.8 map.js 10
    4.1.9 reduce.js 10
    4.1.10 words.js 10
  4.2 Integration testing 10
  4.3 System testing 10
  4.4 Usability testing 10
  4.5 Performance testing 10
  4.6 Reporting 11
  4.7 Issues 11
5 References 12
Appendix A 13
  GitHub Issue template 13

1 Introduction

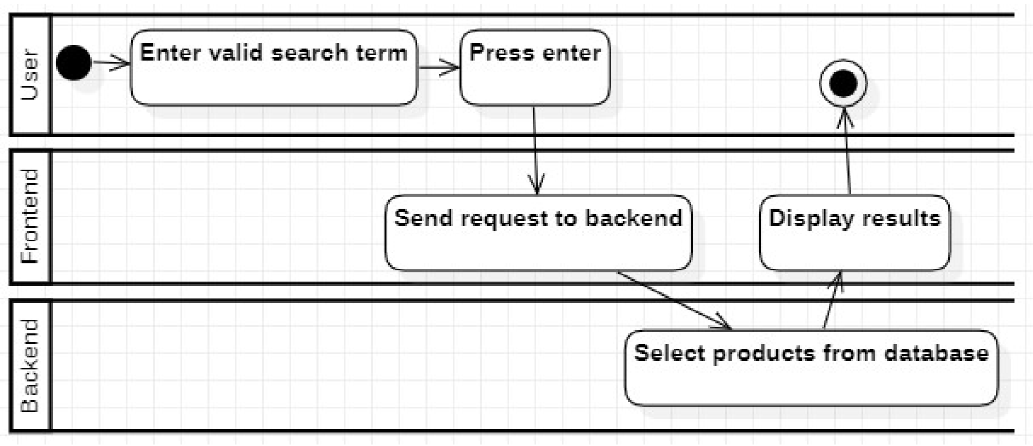
This document contains the test plan for an e-commerce system based on four end-to-end scenarios that were identified as key parts of the system. The scenarios are presented in chapter 2. The tools considered and chosen for the implementation of the test plan are introduced in chapter 3. Chapter 4 details the different kinds of testing planned for the system.

The purpose of this document is to explain the most important use case scenarios of the e-commerce system and the plan to test the system based on those scenarios.

2 Scenarios

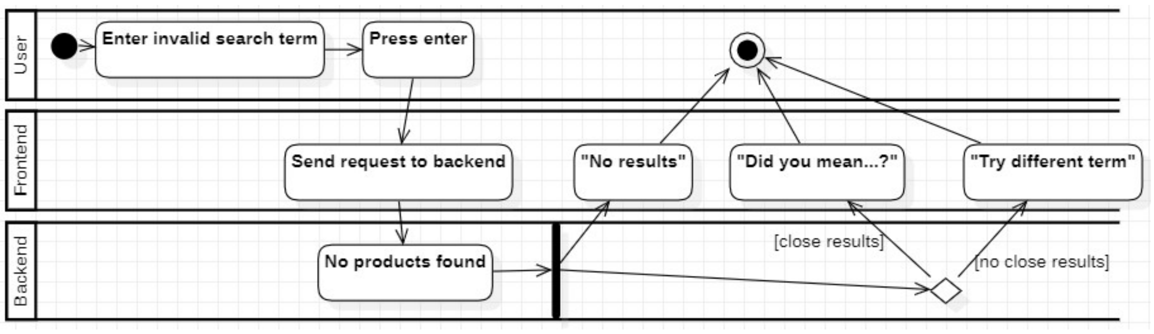
2.1 Search items with a valid product name

User enters search term, which equals to an exact product name, in the search box and presses enter. A list of products is shown to user matching the search term. The best match is shown on top.



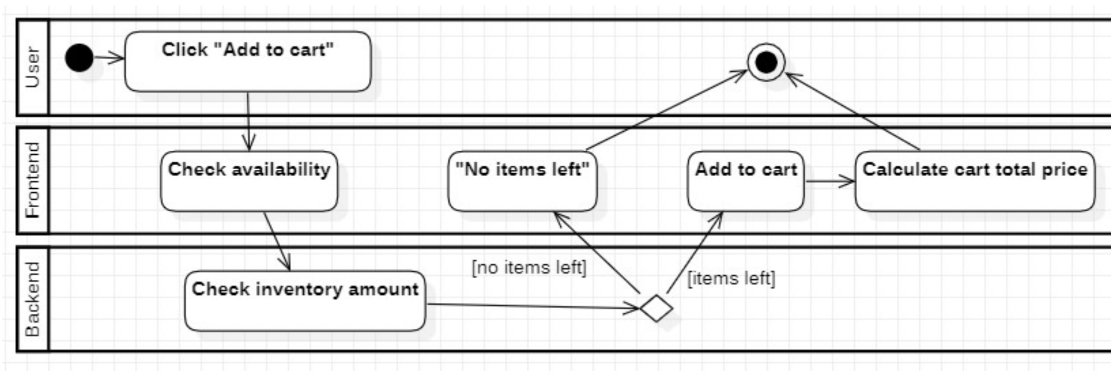
2.2 Search items with an invalid product name

User enters search term with an invalid product name in the search box and presses enter. No exact matches are found, and no products are shown to user. User is displayed a “Did you mean: ***” text under the search bar, if the search term is close to a product name.



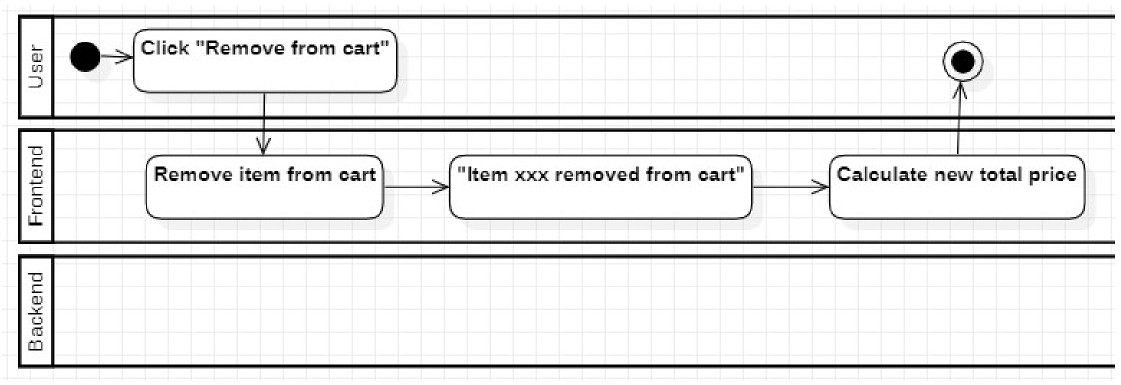
2.3 Adding items to the shopping cart

User clicks “Add to shopping cart” button. A notification of the added product is shown to the user. The product is listed in the shopping cart and the total price of the shopping cart is updated.



2.4 Removing items from the shopping cart

User clicks “Remove from shopping cart” button. A notification of the removed product is shown to the user. The product is removed from the shopping cart list and the total price of the shopping cart is updated.



3 Tools

3.1 Considered tools

Below are some tools that were studied and considered to be used in the implementation of this test plan.

3.1.1 c8

A tool for outputting test coverage metrics [1]. Can be configured for example to include or exclude certain directories or file extensions from the analysis.

Studied the tool by reading the documentation.

3.1.2 chai

Assertion library [2]. Simplifies testing by allowing to make many kinds of assertions against code. Allows writing tests that very closely resembles natural language and has a large selection of plugins available.

Studied the tool by reading the documentation.

3.1.3 coveralls

Coverage reporting tool [3]. Supported by most well-known CI services, like Travis CI, Jenkins, Gitlab CI and GitHub Actions CI.

Studied the tool by reading the documentation.

3.1.4 GitHub Actions

CI tool that is easy to use with GitHub [4]. Allows defining custom workflows that trigger based on events such as pushing commits or creating issues.

Tool was already familiar from previous experience.

3.1.5 jest

JavaScript testing tool [5]. Allows running only certain tests during development and re-running tests when changes to relevant files are made, and aids UI testing with snapshot testing. Has many plugins available to extend functionalities.

Tool was already familiar from previous experience.

3.1.6 jest-chain

Jest-plugin that reduces code duplication and makes multiple assertions on the same object in test cases lighter to read. [6]

Studied the tool by reading the documentation.

3.1.7 jest-extended

Jest-plugin that adds many additional matchers, allowing to write many new types of assertions, possibly simplifying the testing of many features. [7]

Studied the tool by reading the documentation.

3.1.8 mocha

JavaScript test framework that allows running tests on Node.js and in browser [8]. Allows using any assertion library alongside it to simplify testing. Runs tests serially to accurately map uncaught exceptions to the correct tests. Has many plugins available to extend functionalities.

Studied the tool by reading the documentation.

3.1.9 mocha-lcov-reporter

Coverage reporting tool to be used with mocha [9]. Uses the lcov-format.

Studied the tool by reading the documentation.

3.1.10 mochawesome

Report generating tool to be used with mocha [10]. Generates a HTML/CSS report to visualize results of test runs.

Studied the tool by reading the documentation.

3.1.11 jest-html-reporter

Jest-plugin that creates a html-report of a test run. [11]

Studied the tool by reading the documentation.

3.1.12 websequencediagrams

A browser-based tool for generating sequence diagrams with syntax that is close to natural language [12]. Not actually used to implement any tests, just creating documentation.

Studied the tool by reading the documentation.

3.2 Selected tools

After studying the tools, the following tools were chosen to be used when implementing the test plan:

GitHub for storing the project repository
GitHub Actions for CI
Jest for running the unit tests to be implemented
Jest-html-reporter to visualize test results

Coveralls for coverage reporting

GitHub, GitHub Actions and Coveralls were chosen mostly because the course instructions required it, but based on the tool studies they are also valid options.

Jest was chosen for the test runner based on familiarity and good experiences. Mochawesome was considered for test result reporting, but since it is designed to work with mocha and not jest, a jest-based alternative for the same role was found.

4 Tests

4.1 Unit testing

Unit testing is used to test single functions. It is the developer's responsibility to create the unit tests for the features they develop. Unit tests should be run by the developer before committing changes. They are run also as part of the CI pipeline in GitHub.

Minimum requirement for unit tests to be considered “passed” is that the function passes positive test case with expected input. For most cases, it is expected that the function testing includes obvious negative cases, handling errors in a controlled manner.

Code coverage requirement for unit tests is 10 functions / library, as defined by course instructions. Per function, all outcomes should be tested. All functions provided in the library were evaluated and the ones that were most likely to be used the most were selected to be unit tested. Functions selected for testing are specified below.

4.1.1 defaultTo.js

Used for display purposes, to show descriptive text in place of null or undefined.

4.1.2 filter.js

Search functionality. User can quickly filter results returned from backend.

4.1.3 get.js

Get nested value for product for display purposes. E.g., product listing.

4.1.4 isArrayLike.js

Confirms that an object is iterable, before trying to iterate it. E.g., product listing.

4.1.5 isBoolean.js

For product info page. If product info has boolean values, they are replaced by “Yes” / “No” for display.

4.1.6 isEmpty.js

Checks whether search results are empty or not. Used to display “No results” text to user instead of an empty page.

4.1.7 keys.js

Used to translate object keys. E.g., for product object, key productWeight could be translated to “Product Weight” for display on the product info.

4.1.8 map.js

General purpose function, used for many features. Example use case: cart price calculation by mapping products to just product prices before calculating.

4.1.9 reduce.js

General purpose function. Example use case: calculating cart overall price.

4.1.10 words.js

Used to split single search string to multiple search terms for more accurate search results.

4.2 Integration testing

Integration testing is done in the frontend with Jest. Utility library is tested as a part of the frontend features. Integration testing is done with positive test cases, as the frontend functions shouldn’t allow the user to give invalid input. Integration tests need to pass for the end-to-end scenarios be possible to test.

4.3 System testing

System testing involves a test environment with functioning frontend and backend. It is performed by developers in an explorative manner. Later, it can be automated with Robot Framework Browser Library, where considered necessary. Test cases equal to scenarios described in chapter 2.

4.4 Usability testing

Since actual implementations of the frontend or backend are not available for testing, testing the usability of the system does not make sense. Usability should be tested periodically throughout the development of the system, especially when introducing new features to the frontend.

4.5 Performance testing

The performance of the backend should be tested by simulating different load patterns to find out how the system reacts to a large number of users. Performance should be tested at least with a continuous stream of high amount of requests to find the maximum amount of handled requests in a minute for example and to see if any problems arise when that amount is exceeded. Another load pattern to test with is to periodically send very high spikes of requests, for example 10 or 100 times more than the backend can handle.

4.6 Reporting

Jest test results are exported to a readable HTML document. Reporter library used is jest-html-reporter [12].

GitHub Actions gives info on unit test status on pull requests.

Coveralls reports the coverage of the unit tests.

4.7 Issues

Bugs found are categorised in 3 separate categories. Issues are created on GitHub issues, using the template in appendix A.

Type	Description
Minor	A bug that affects a non-crucial feature, or a bug that can be bypassed or otherwise has no effect on the usability of the store
Moderate	A single feature is completely unusable
Severe	Disrupts the whole store

5 References

[1] https://www.npmjs.com/package/c8 (Test coverage)
[2] https://www.npmjs.com/package/chai (Assertion library)
[3] https://www.npmjs.com/package/coveralls (Test coverage service integration)
[4] https://github.com/features/actions (GitHub Actions)
[5] https://www.npmjs.com/package/jest (Test framework)
[6] https://www.npmjs.com/package/jest-chain (Utility library to make Jest tests chainable)
[7] https://www.npmjs.com/package/jest-extended (Extension for Jest assertions)
[8] https://www.npmjs.com/package/mocha (Test framework)
[9] https://www.npmjs.com/package/mocha-lcov-reporter (Test reporter)
[10] https://www.npmjs.com/package/mochawesome (Test reporter)
[11] https://www.npmjs.com/package/jest-html-reporter (Test reported)
[12] https://www.websequencediagrams.com/ (Tool for creating simple UML se-quence diagrams)

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 ***