E-COMMERCE APPLICATION TEST PLAN

COMP.SE.200 SOFTWARE TESTING - TUNI

Tatu Kankare, 151182440
Heidi Seppi, H252889

Definitions, acronyms and abbrevations

AI	Artificial intelligence
E-commerce	Electronic commerce
UML	Unified modeling language
Sequence diagram	Interaction diagram showing how and when the needed actions of the process are happening

Introduction

The purpose of this document is to plan out how an E-commerce application defined in the course assignment will be tested. Main idea of this E-commerce application is to sell food products from different producers. Users can browse products using different search criteria, add products to the shopping cart and checkout when they are ready to purchase the selected products. On the other hand, producers can add their own products to the application for users to purchase.

First, the most common end-to-end scenarios are described and visualized using sequence diagrams and after that the selected tools for testing are described. After the scenario and testing tool chapters, the document focuses on describing the planned tests and lists the 10 chosen source files from the utility library for which the unit tests will be implemented later. Lastly, the way to report and document tests will be discussed about. Also, the use and benefits of AI tools are mentioned at the end of this documentation.

Scenarios

End-to-end scenarios define application workflows from the beginning to the end. We have defined four most common end-to-end scenarios for the described E-commerce store. These end-to-end scenarios are described below with the help of UML sequence diagrams. The assignment description didn't mention any use of databases but for this case it is assumed that at least the products are stored in a database.

Scenario 1: User browses and searches for different products by category, price, product content and producer information.

This scenario describes the situation when the user wants to browse the products and search for certain products from the E-commerce store. The user can browse different products using category, price, product content and producer information to find suitable products for them.

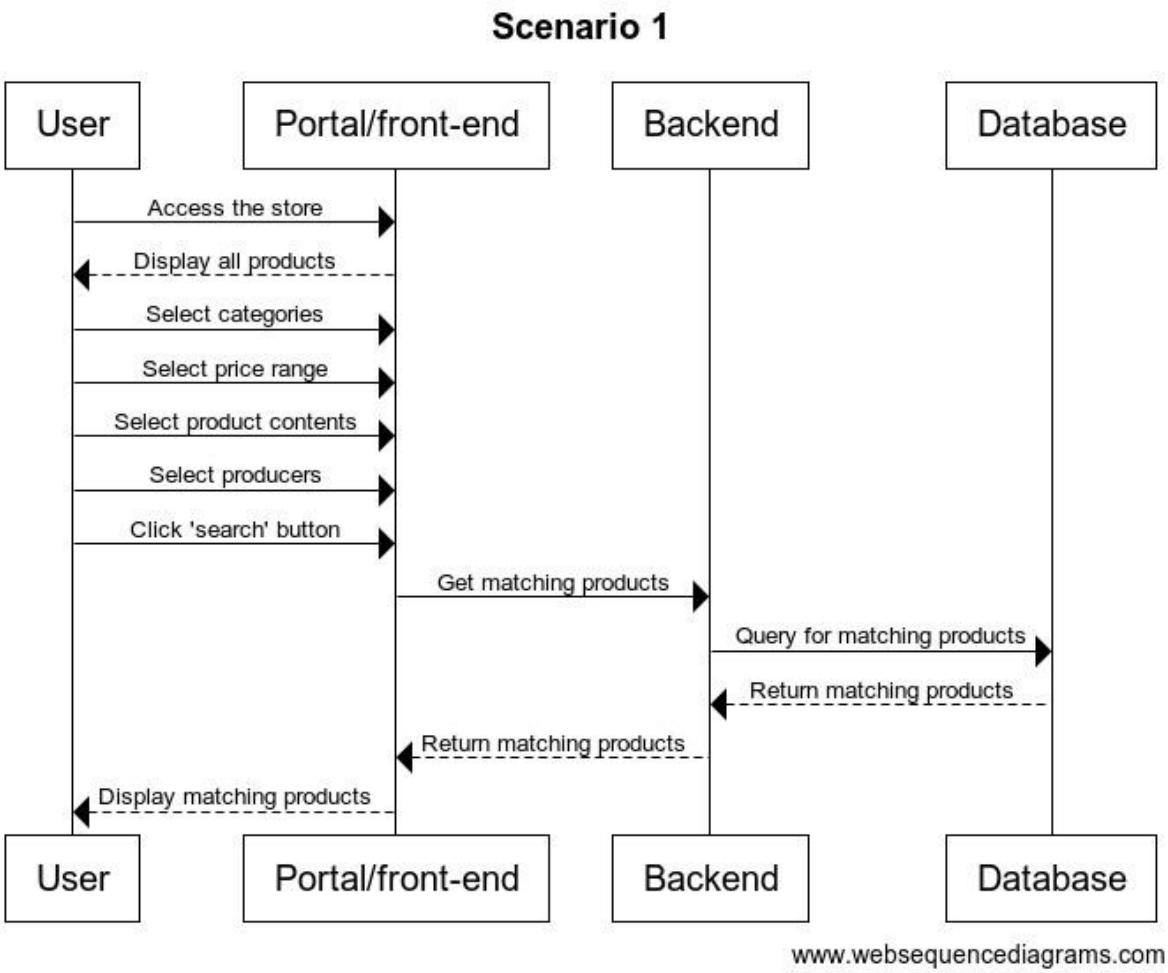

Figure 1: Sequence diagram for the scenario when user is browsing different products on the store.

Figure 1 sequence diagram shows the main actions when user wants to browse for different E-commerce store products. The user can at first see all the products and if they want to, search for certain kinds of products by selecting category, price range, product content and/or producer. Products are stored in the database so the backend will make a query to find matching products and return the results to the frontend. After the search the store displays the products to the user which match the selection criteria. The user can make new searches with different search criteria.

Scenario 2: User adds or removes products from the shopping cart and views the total price.

This scenario is about the user adding selected products to their shopping cart and being able to see the total price of their shopping cart. The user is also able to delete products from their shopping cart and the total price is updated automatically. In the E-commerce store assignment description, there were no direct mentions of being able to delete products from the shopping cart, but this functionality was considered here because removing items from the shopping cart is usually one of the basic

functionalities in online shops. Figure 2 displays the sequence diagram for this scenario.

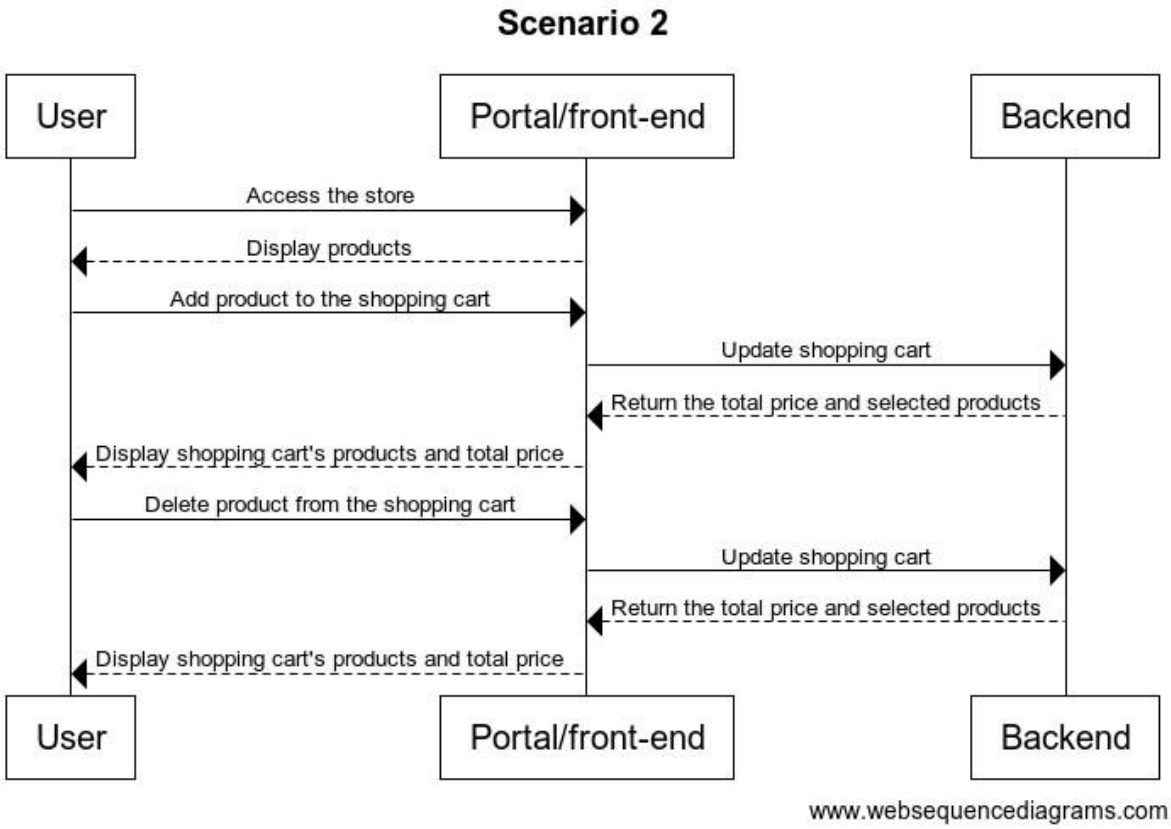

Figure 2: Scenario 2 of adding and removing products from the shopping cart.

Scenario 3: User purchases the products which have been added to the shopping cart.

Scenario 3 describes a situation where the user adds products to the shopping cart and wants to buy the selected products. Figure 3 shows the sequence diagram for this scenario.

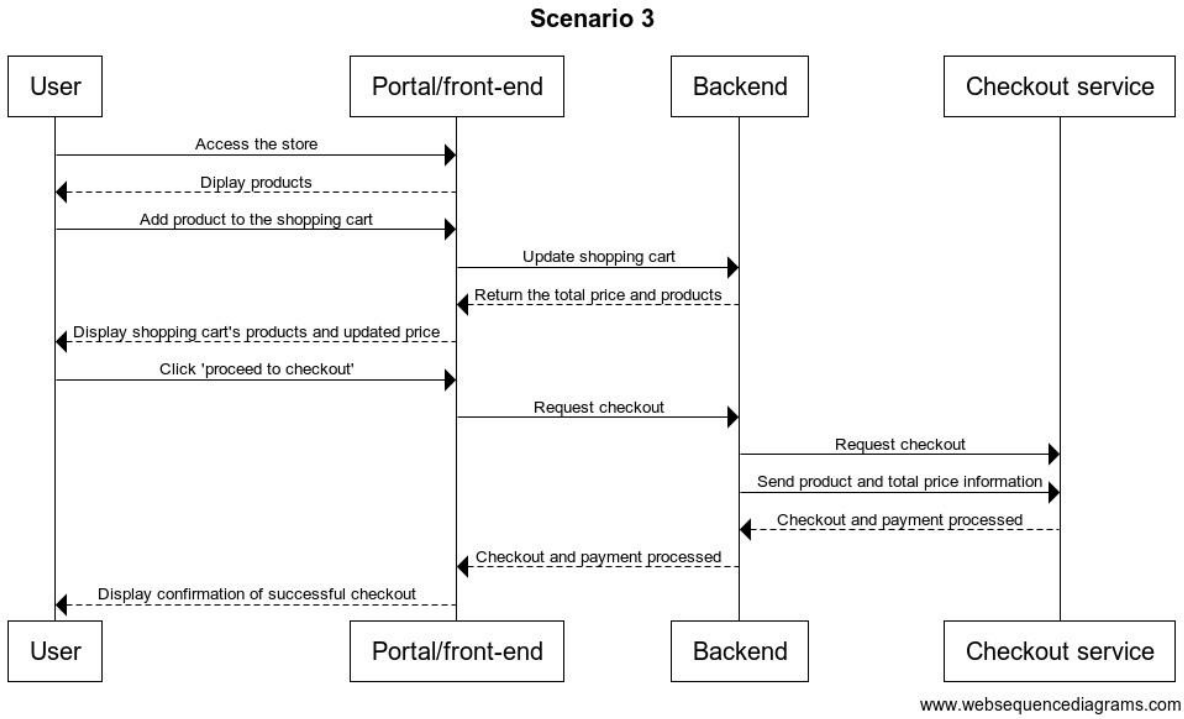

Figure 3: Scenario 3 of user doing checkout after adding products to the shopping cart.

At first user adds the needed products to the shopping cart and when they are ready to purchase the products, they can click ‘proceed to checkout’ button to proceed with the ordering. This begins the checkout process, and the third-party checkout service handles the checkout and payment methods. After the payment has been accepted and the ordering has been completed in the third-party service, the user gets confirmation of successful checkout. In this scenario it is assumed that the checkout service handles the ordering and the payment of the products completely.

Scenario 4: A food producer adds products for sale using the portal or front-end application.

A food producer needs to be able to add their products to the store and the scenario 4 shown in figure 4 describes this situation. In this case all the products are assumed to be saved to a database in which the new added product will be saved.

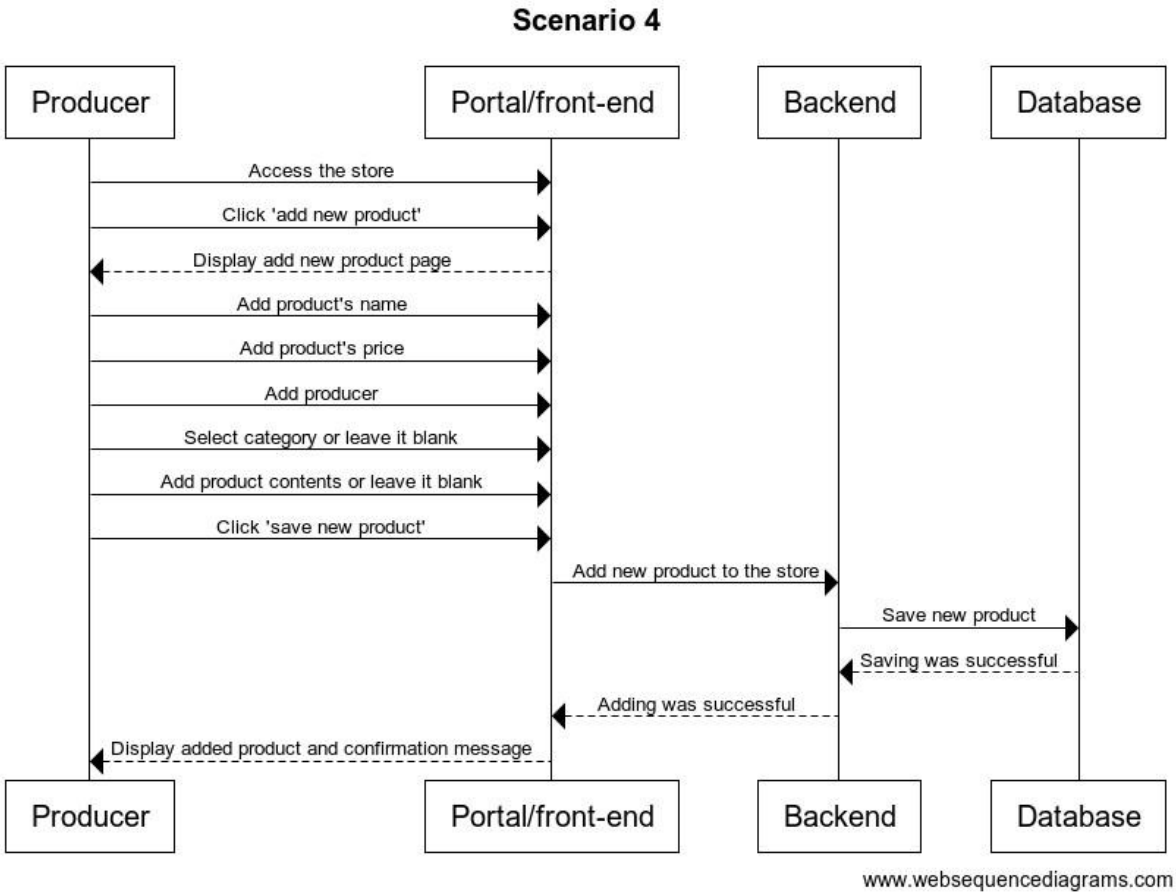

Figure 4: Scenario 4 of producer adding a new product to the store.

Producer can start adding a new product by clicking the button ‘add new product’ which then opens a page where all the product information for the new product can be filled. Product needs to have at least a name, a price and a producer. Category and product content is optional to fill in. After all the required information has been filled, producer can save the product. Product will be saved to the product database, and the producer will get a confirmation message when the product has been added successfully to the site.

Tools

While making the test plan and visualizing the end-to-end scenarios, WebSequenceDiagrams [1] drawing tool was used to create sequence diagrams. After testing this tool out, it seemed to be the most efficient way to draw the diagrams for the different scenarios.

The assignment definition requires the unit testing environment to include Github [2], Github Actions [3] and Coveralls [4].

Mocha unit testing framework [5] will be used for the unit tests. Chai assertion library [6] may also be used. They have been found to be an effective way of testing JavaScript in other projects.

Tests

Unit tests will be performed on the library [7] files specified in table 1 using appropriate test automation methods. Limited integration testing may be performed, where the selected components work together in a manner making integration possible.

Table 1: Files of the Library to be Unit tested.

Priority	File	Rationale for testing
High	chunk.js	Displaying products in a grid.
High	slice.js	Dependency of chunk.js
High	toInteger.js	Dependency of chunk.js
High	words.js	Search matching
High	filter.js	Search filtering
High	reduce.js	Calculating total price
Medium	get.js	Generic utility
Medium	defaultTo.js	Generic utility
Medium	eq.js	Generic utility
Medium	isEmpty.js	Generic utility

Other functions in the library were considered low priority due to their high specificity of use and lack of obvious use cases and have been excluded from unit testing due to time constraints (only 10 source files can be tested). Unit tests for the selected files try to find errors from the main actions mentioned in the end-to-end scenarios. For example, creating unit tests for chunk.js is important so that it is possible to find errors happening while displaying the store products.

System testing will be performed manually: testing through the described end-to-end scenarios 1,2,3 and 4. Errors from both users and third-party services will be included in some of the test runs. System testing ensures that the recognized end-to-end scenarios work as a whole and the different E-commerce parts, such as front-end,

back-end and third-party services, work together successfully so that the core requirements for the E-commerce store are being met.

Usability testing of the front-end interfaces both for users and producers will be performed to ensure they meet accessibility guidelines and can be operated without a difficulty if a person is unfamiliar with this particular web store.

Extensive performance testing will not be necessary. Due to the small scope of the store and serving small local businesses, large amounts of users are not expected. Sufficient overprovisioning of resources should minimize the possibility of significant performance issues not found during other testing.

If the E-commerce store application hasn't been released to the customer and is not yet in use, some form of customer acceptance testing should be performed to make sure that the customer requirements have been understood in a correct way. Acceptance testing helps to make sure that the customer is satisfied with the final application.

Content left out of testing

The given E-commerce store assignment description didn't mention any login and registration system for users so this won't be thoroughly tested since the documentation didn't describe clearly if the application has this feature and what kind of login and registration system would be needed. Although some form of access control system will likely be required for the producers to prevent misuse. In this case it will be assumed to be provided as a third-party service, but its interface may be tested.

The checkout service is specifically provided as a third-party service and as such will not be tested directly, however the interface of the checkout service may be tested.

The assignment description didn't mention if any databases are being used so it is assumed that at least the products and their information is stored in some database. Tests won't directly focus on database usage, but system testing makes sure that all the end-to-end scenarios work as intended.

Test reporting and classification

Test results for tests other than unit tests will be documented according to a template in Appendix A. Unit test results may also be documented if the defects found by them are not fixed immediately.

Found defects will be categorized based on their likelihood of occurrence (1-5, 1=unusual or incorrect use, 5=every time the system is used) and severity (1-5, 1=cosmetic, 5=clients lose money or are harmed).

Tests will be considered passed once no more defects are found during the specified test cases. Adequate test coverage is dependent on the quality of these tests.

AI and testing

It is possible to use AI, for example ChatGPT [8], to find different end-to-end scenarios for the E-commerce store application using the given store description. AI might give new ideas of what the E-commerce store could do and how the different features could work but it is not recommended to blindly trust all the information what AI gives since it might not always suit the exact situation perfectly. AI could also help to find new areas to be tested from the application which weren't taken into consideration originally. For example, in this case AI could give reasons why performance testing should be considered to be more important than it was originally considered to be. Although it is good to note that AI might hallucinate so it can give false information so it should only work as a tool to help to give new ways to think how the testing could be done.

During this assignment AI was not used because the main idea of the assignment is to learn yourself how to plan the testing for the E-commerce store application. One of the best ways to learn is to think and consider different solutions yourself and not just trust AI to give you direct answers how the testing should be done. Once the person has more experience of testing applications, AI can help by giving new different ideas of how to test components.

References

[1] WebSequenceDiagrams (https://www.websequencediagrams.com/)

[2] GitHub (https://github.com/)

[3] GitHub Actions (https://github.com/features/actions)

[4] Coveralls (https://www.npmjs.com/package/coveralls)

[5] Mocha (https://www.npmjs.com/package/mocha)

[6] Chai (https://www.npmjs.com/package/chai)

[7] Assignment Library (https://github.com/otula/COMP.SE.200-2024-2025-1)

[8] ChatGPT (https://chatgpt.com/)

Appendix

Appendix A: test_documentation_template.docx

Template can be found attached to the submission zip file. A snip image of the test documentation template is shown below.

Test Date: *YYYY-MM-DD*

Tester: (*If applicable)*

Test Type: *System, performance etc.*

Test Environment: *System, browser, test framework etc. (If applicable)*

Test Description: *What was tested, how it was tested*

Defects found: *Description of found defects and how to reproduce them*

Defect	How to reproduce	Severity	Likelihood of occurrence