COMP.SE.200-2022-2023

Test plan assignment, part 1

Kalle Puutio, 430584, kalle.puutio@tuni.fi
Kari Ylönen, 428178, kari.ylonen@tuni.fi

Introduction 3
  What is in the document? 3
  What is the purpose of the document? 3
  Definitions and abbreviations 3
Scenarios 4
  Scenario 1: Customer searches for a product 4
  Scenario 2: Customer orders products 5
  Scenario 2.1: Customer Cancels payment 6
  Scenario 2.2: Customer closes web page during ordering 6
  Scenario 2.3: Customer deletes items from shopping cart 6
  Scenario 3: Producer adds a product via portal 7
Tests 9
  Unit tests 9
  Integration tests 10
    How to test the defined requirements 10
  Usability tests 10
    How to test the font and contrast standards: 11
    How to test the user performance: 11
  Performance tests 11
    How to test database performance: 11
  Testing tools 11
Test reporting 13
  Test report templates 13
   Automated and integration tests 13
   Performance tests: 13
Testing environment 14

Introduction

What is in the document?

This document contains the test plan for an E-commerce web application. Included are the user scenarios upon which the test cases are based upon. Prioritization of unit tested source files is based on MoSCoW prioritization. Integration testing, usability testing, and performance testing plans are outlined and the test requirements are defined. Test reporting template is outlined.

What is the purpose of the document?

The purpose of this document is to define the test plan for an E-commerce web application and to be as the main reference document during the test plan implementation.

Definitions and abbreviations

Abbreviation/term	*Definition*
Checkout	The point where customer begins the payment process. Responsibility after this point is on the 3rd party payment system.
MoSCoW	Must have, Should have, Could have, Won't have. A prioritization method.
User	The individual using the application. Can be a customer or vendor.
Customer	The individual using the application, with the goal of finding and buying products.
Vendor	The individual using the application with goal of adding or removing products via a previously created portal.
Frontend	The part of the application that is visible to the user. Data is transmitted from frontend to backend.
Backend	A REST back-end, which provides the interfaces for executing the core functionality of the E-commerce store: - searching, - buying, - adding - removing products.
3rd party payment solution	The third-party solution that handles the checkout- and payment processes.

Scenarios

Scenario 1: Customer searches for a product

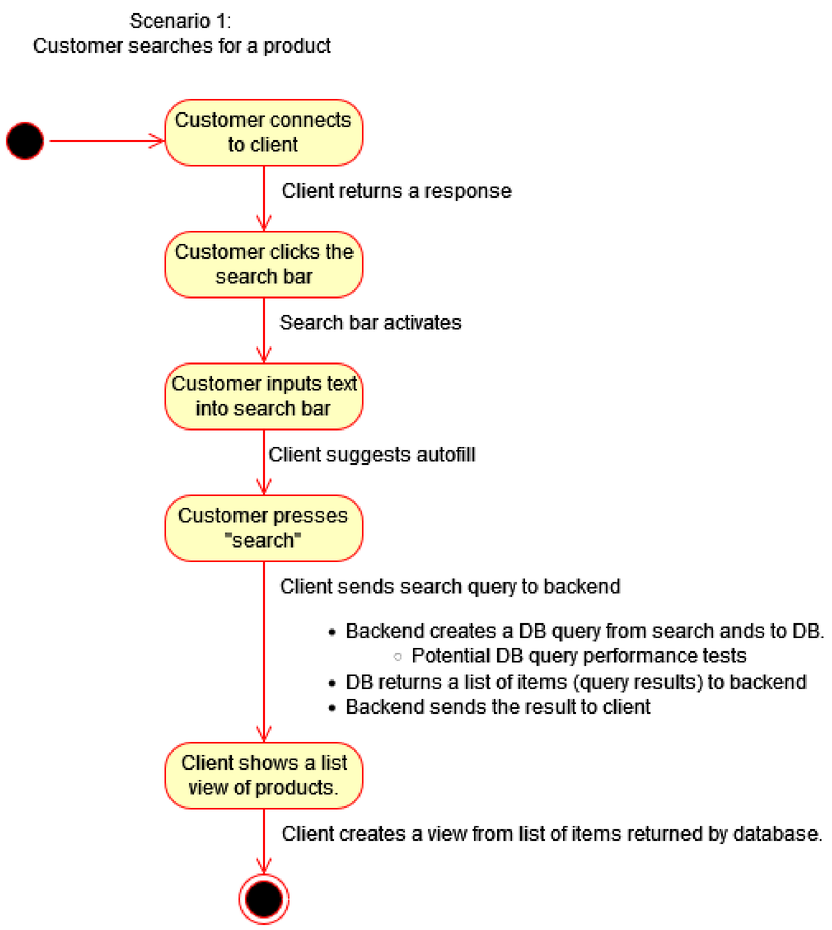

Scenario 1:
Customer searches for a product
Customer connects to client
Client returns a response
Customer clicks the search bar
Search bar activates
Customer inputs text into search bar
Client suggests autofill
Customer presses "search"
Client sends search query to backend
Backend creates a DB query from search ands to DB.
Potential DB query performance tests
DB returns a list of items (query results) to backend
Backend sends the result to client
Client shows a list view of products.
Client creates a view from list of items returned by database.

Figure 1: Scenario 1 sequence diagram.

1. Customer connects to client
   - Client returns response
2. Customer clicks the search bar
   - Search bar activates
3. Customer inputs text into search bar
   - Client suggests autofill
4. Customer presses "search"
   - Client sends search query to backend
     - Backend creates a DB query from search and sends to DB
     - DB returns a list of items (query results) to backend
     - Backend sends the results to client
5. Client shows a list view of products.
   - Client creates a view from list of items returned by DB

Scenario 2: Customer orders products

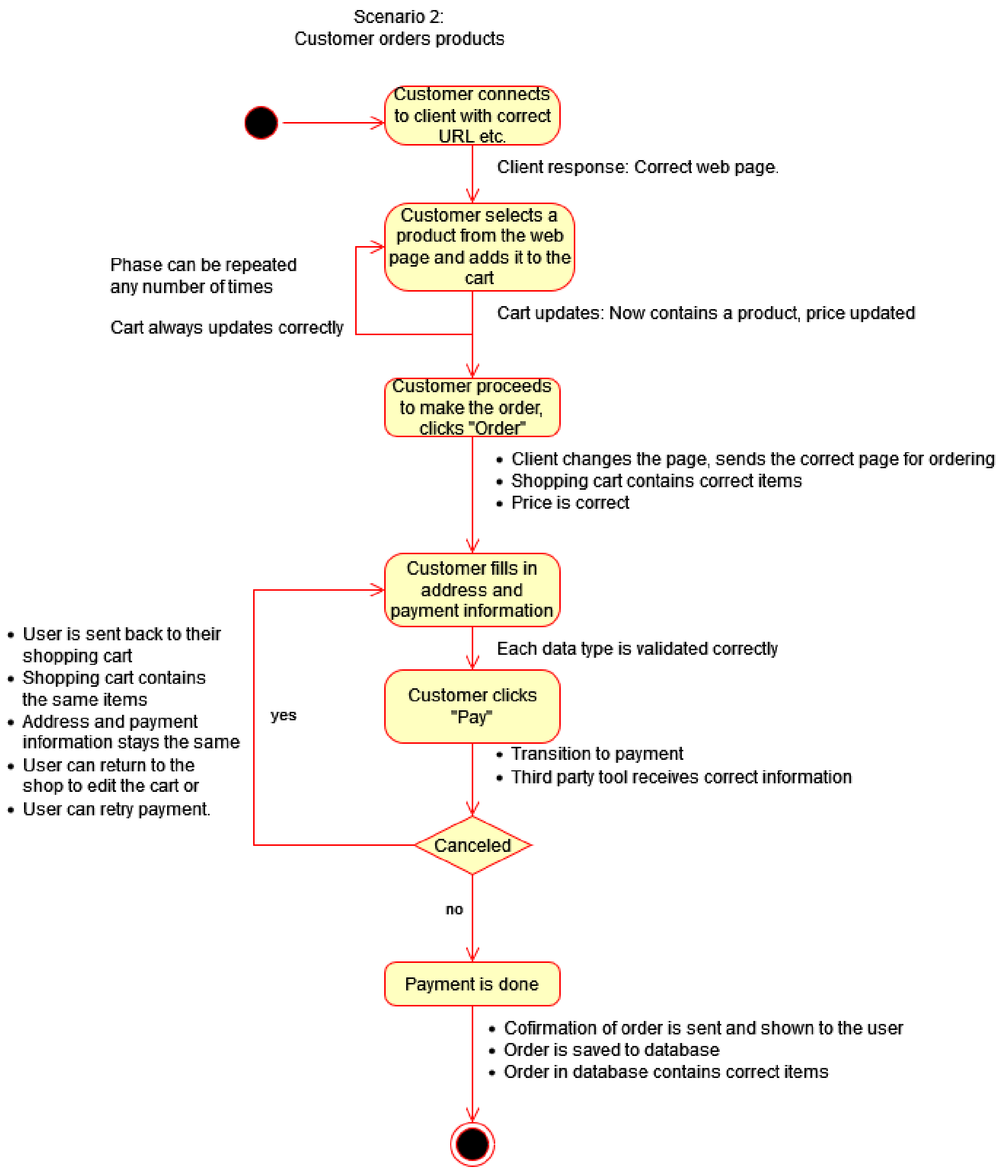

Scenario 2:
Customer orders products
Customer connects to client with correct URL etc.
Client response: Correct web page.
Customer selects a product from the web page and adds it to the cart
Phase can be repeated any number of times
Cart always updates correctly
Cart updates: Now contains a product, price updated
Customer proceeds to make the order, clicks "Order"
Client changes the page, sends the correct page for ordering
Shopping cart contains correct items
Price is correct
Customer fills in address and payment information
User is sent back to their shopping cart
Shopping cart contains the same items
Address and payment information stays the same
User can return to the shop to edit the cart or
User can retry payment.
yes
Each data type is validated correctly
Customer clicks "Pay"
Transition to payment
Third party tool receives correct information
Canceled
no
Payment is done
Cofirmation of order is sent and shown to the user
Order is saved to database
Order in database contains correct items

Figure 2 Scenario 2 sequence diagram.

1. Customer connects to client with correct URL etc.

   - Client response: Correct web page.

2. Customer selects a product from the web page and adds it to the cart

   - Cart updates: Now contains a product, price updated

3. Phase two can be repeated any number of times.

   - Cart always updates correctly

4. Customer proceeds to make the order, clicks "Order"

   - Client changes the page, sends the correct page for ordering

   - Shopping cart contains correct items

  - Price is correct

5. Customer fills in address and payment information

  - Each data type is validated correctly

6. Customer clicks "Pay"

  - Moves to payment

  - Third party tool receives correct information

7. Payment is done:

  - Confirmation of order is sent and shown to the customer.

  - Order is saved to database

  - Order in database contains correct items.

Scenario 2.1: Customer Cancels payment

Same as base scenario, except step 7 changes.

7. Payment is canceled:

  - Customer is sent back to their shopping cart

  - Shopping cart contains the same items

  - Address and payment information stays the same

  - Customer can return to the shop to edit the cart or

  - Customer can retry payment.

Scenario 2.2: Customer closes web page during ordering

At any point during the base scenario, the web page could be closed due a user or software error. On the web page:

- Cart information is saved

- Cart price is shown correctly

- Possible address information is saved on client.

- Payment information is saved on client.

Scenario 2.3: Customer deletes items from shopping cart

At any point during the base scenario, the customer may delete one or more products from their shopping cart. The customer cannot remove products from an empty shopping cart or during the third-party payment. The following is expected:

- The cart contains update correctly.

- The cart price updates correctly.

- If a payment is done after, the deleted items are also deleted from the payment and the order correctly.

Scenario 3: Producer adds a product via portal

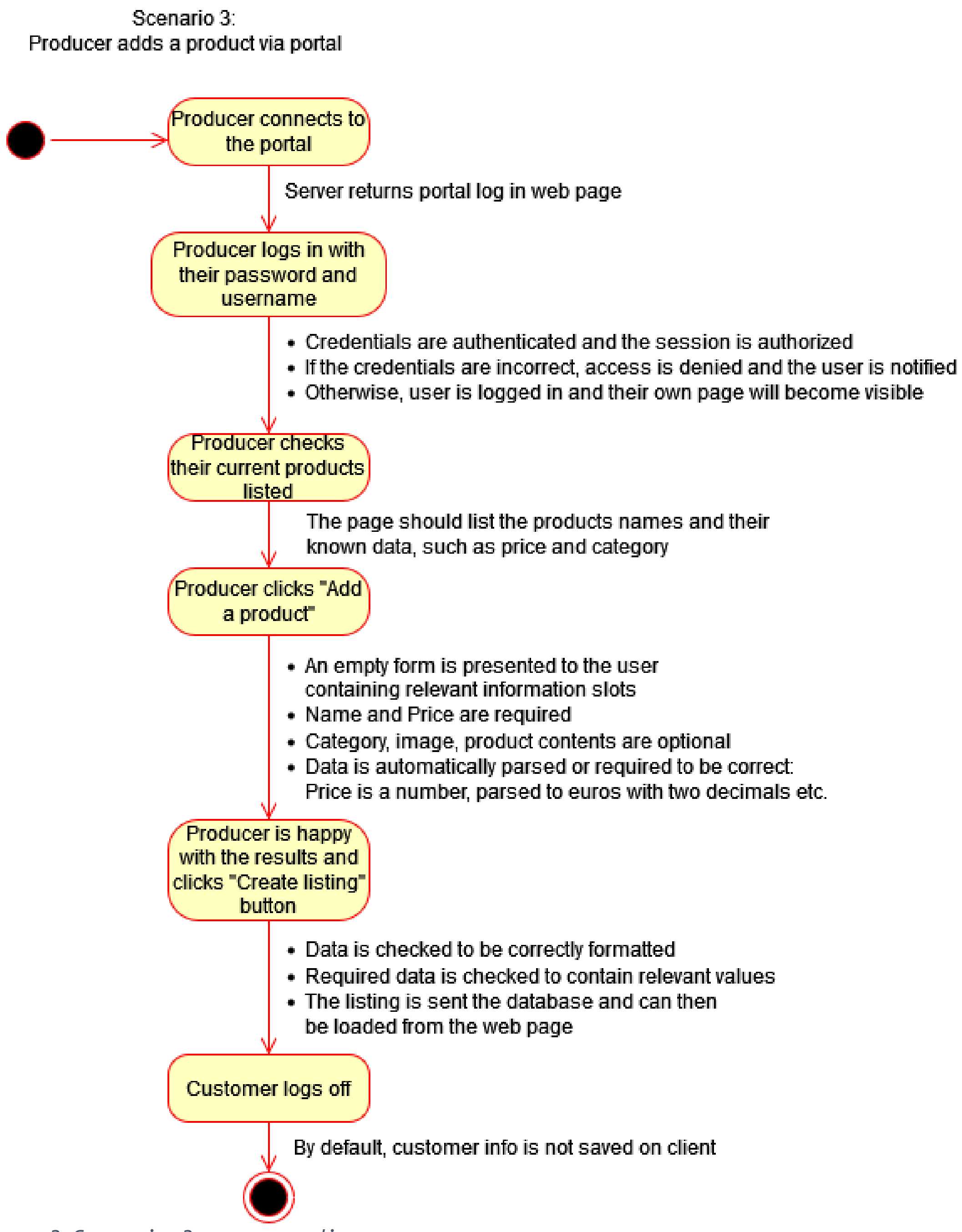

Scenario 3:
Producer adds a product via portal
Producer connects to the portal
Server returns portal log in web page
Producer logs in with their password and username
• Credentials are authenticated and the session is authorized
• If the credentials are incorrect, access is denied and the user is notified
• Otherwise, user is logged in and their own page will become visible
Producer checks their current products listed
The page should list the products names and their known data, such as price and category
Producer clicks "Add a product"
• An empty form is presented to the user containing relevant information slots
• Name and Price are required
• Category, image, product contents are optional
• Data is automatically parsed or required to be correct: Price is a number, parsed to euros with two decimals etc.
Producer is happy with the results and clicks "Create listing" button
• Data is checked to be correctly formatted
• Required data is checked to contain relevant values
• The listing is sent the database and can then be loaded from the web page
Customer logs off
By default, customer info is not saved on client

Figure 3 Scenario 3 sequence diagram

1. Producer connects to the portal

   - Server returns portal log in web page

2. Producer logs in with their password and username

   - Credentials are authenticated and the session is authorized

   - If the credentials are incorrect, access is denied and the customer is notified

- Otherwise, customer is logged in and their own page will become visible

3. Producer checks their current products listed

   - The page should list the products names and their known data, such as price and category

4. Producer clicks "Add a product"

   - An empty form is presented to the customer containing relevant information slots

   - Name and Price are required

   - Category, Image, product contents are optional

   - Data is automatically parsed or required to be correct: Price is a number, parsed to euros with two decimals,

5. Producer is happy with the result and clicks "Create listing" button

   - Data is checked to be correctly formatted

   - Required data is checked to contain relevant values

   - The listing is sent to a database and can then be loaded from the web page

6. Customer logs off.

   - By default, customer info is not saved on the client.

Tests

Unit tests

We use MoSCoW prioritization method to select 10 source files for testing. The 10 must have source files are considered the most relevant functions in the e-commerce application. Based on the user scenarios, these are the functions that would be used to provide the functionality under the hood.

The priority files include the functions used in both frontend and backend.

Priority	Files
Must have	words.js toNumber.js toString.js isArrayLikeObject.js ceil.js compact.js endsWith.js get.js isEmpty.js chunk.js
Should have	add.js clamp.js countby.js every.js filter.js isArrayLike.js isArrayLikeObject.js isBoolean.js isObject.js isObjectLike.js toInteger.js
Could have	at.js capitalize.js castArray.js defaultTo.js defautlToAny.js drop.js eq.js isDate.js isLength.js keys.js slice.js toFinite.js upperFirst.js
Won’t have	camelCase.js difference.js divide.js isArguments.js isBuffer.js isSymbol.js

	isTypedArray.js map.js memoize.js reduce.js

Integration tests

Each step of the previously defined use-case scenarios is tested via integration testing. Shopping cart and the stores products must remain consistent throughout the interaction of the user in multiple different edge cases.

More integration tests can be added as needed.

The following requirements must be met:

- Chrome, Safari, Edge, Firefox and Samsung Internet browsers must support every page and their states.
- Each transition in use-case scenarios must function without any authentication or shopping cart information loss.
    - The following are considered information losses during transitions:
        - Logout without user input.
        - Items being removed or added into the shopping cart without user input.
- The purchase information from frontend must transfer to backend.
- The purchase information from backend must transfer to 3rd party payment system.
- The frontend must be able to ping the backend.
- Backend must be able to ping database.

How to test the defined requirements

- For each test, use each of the defined browsers and log the results.
- Log a starting state for a transition and compare it with the post-transition state. Log either success or failure and reason for failure. Unexpected state changes are failures.

The shopping cart, the store selection and the payment methods are the most important pieces of the software and will be most thoroughly tested. All the tests are done from separate network than where the servers are located to ensure the client-server connections and communications.

Usability tests

As the product is a store for consumer, the usability of the store is one of the most, if not the most important single property of the software.

More usability tests can be added as needed.

Following standards must be met:

- Fonts and font sizes must be consistent within the application.
    - Use the same fonts.
    - The font sizes must match the page header levels uniformly.
- The contrasts must meet ISO-9241-3 standard.

- The customer must be able to find and buy a product with a maximum of 10 clicks and 4 page loads.

How to test the font and contrast standards:

- Automate font size checking to match the suggested header font size.
- Run webAIM contracts checker tool on the page (https://webaim.org/resources/contrastchecker/)

How to test the user performance:

- Calculate the clicks required for the customer to reach the 3rd party checkout button.

Performance tests

The goal of the performance tests is to maintain smooth user experience for the customer.

In the frontend, the target is to keep the page load duration under 3 seconds on a 100M internet connection.

In the backend, the target is to maintain database query time of maximum 5 seconds.

Following limits must be met:

- The server load must be tested via quadruple number of users expected to visit the site on peak times like Black Friday (10 000).
- Page load durations must not exceed three seconds with an internet connection of 100M in more than 5% of the cases.

The stores databases may grow massively when new users create accounts and the store gains increasing number of products. The size of the database grows over time.

How to test database performance:

- Server load could be tested alongside the usability tests previously defined or done separately as a different organized test.

Testing tools

Mocha – Mocha is a test framework for node.js. It helps creating a good, separated testing code with helpful mechanisms and functions. Beyond simple unit testing, this framework would be used together with Chai.

Chai – Chai is an Assertation library. It expands the programmer's options on creating good and easy to read tests for the software. Mocha and Chai both support test driven development and are good tools together. In the case of this assignment (just 10 unit tests), Chai will be enough.

GitHub – GitHub is a version control system and is used as the main repository control for the project.

GitHub Actions – GitHub Actions is the CI/CD pipeline of GitHub. It automates testing to be ran every time a new version is pushed to the repository. It also handles the reporting of the outcomes.

Coveralls – Coveralls is a test coverage tool. With it the coverage of tests can be calculated. Reaching high coverage is important for a well test automated software.

Mocha LCOV Reporter – To get Mocha framework work alongside coveralls, LCOV Reporter turns the data into Coveralls readable style, and allows automation of the coverage calculations. This way we can add the coveralls information automatically to our test reports.

Test reporting

Test report templates

Automated and integration tests

ID	Description	Type	Success	Category	Bug/issues
0	Sums positive integers	Automatic	Success		
1	Automatically creates a correct shopping cart	Integration	Fail	Critical	Shopping cart implementation is not finished yet
2	…		…	…	…

Overall result:
Successful tests: 1 / 2 (50%)
Automatic test coverage: (9 633 / 120 412) 8% of total function points
Comments: Shopping cart causes a lot of issues.

Performance tests:

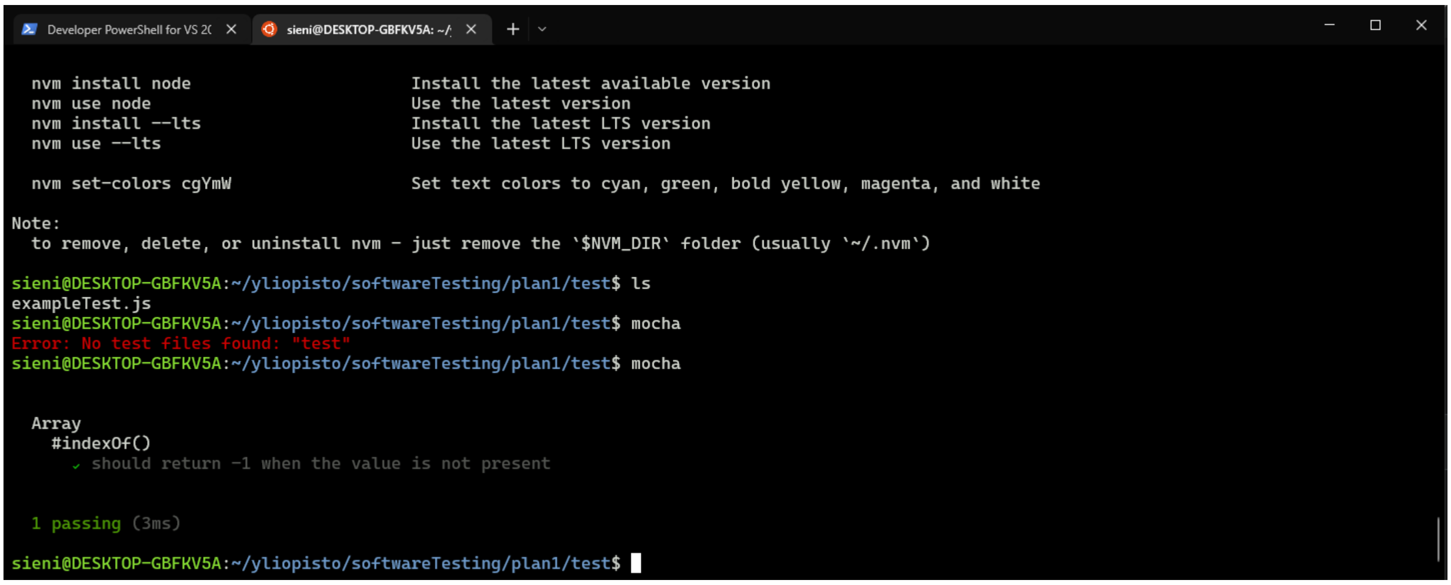
ID	**Environment**	**Outcome**	**Load time**
0	Mac, Chrome, 720p	Success	0.4s
1	Android, Samsung Internet, 564p	Success	2.8s
2	Windows, Edge, 1080p	Failure	12.6s

Success rate: 2 / 3 67%
Mean: 5.3s
Median: 2.8s

Testing environment

The test setup is Ubuntu 20.04. running on WSL2.

- Distributor ID: Ubuntu
- Description: Ubuntu 20.04.5 LTS
- Release: 20.04
- Codename: focal



```
  nvm install node                     Install the latest available version
  nvm use node                         Use the latest version
  nvm install --lts                    Install the latest LTS version
  nvm use --lts                        Use the latest LTS version

  nvm set-colors cgYmW                 Set text colors to cyan, green, bold yellow, magenta, and white

Note:
  to remove, delete, or uninstall nvm - just remove the `$NVM_DIR` folder (usually `~/.nvm`)

sieni@DESKTOP-GBFKV5A:~/yliopisto/softwareTesting/plan1/test$ ls
exampleTest.js
sieni@DESKTOP-GBFKV5A:~/yliopisto/softwareTesting/plan1/test$ mocha
Error: No test files found: "test"
sieni@DESKTOP-GBFKV5A:~/yliopisto/softwareTesting/plan1/test$ mocha


  Array
    #indexOf()
      ✓ should return -1 when the value is not present


  1 passing (3ms)

sieni@DESKTOP-GBFKV5A:~/yliopisto/softwareTesting/plan1/test$
```