**CS673 Software Engineering** 

**Scrumble Bug - Campus Exchange**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Yingtong Zhou | Team leader | [Yingtong Zhou](mailto:cutejiu@bu.edu) | Oct 17, 2024 |
| HungHsu(Allen) Chen | Configuration Leader | *HungHsu(Allen) Chen* | 10/17/2024 |
| Yuanbin Man | Requirement Leader | *Yuanbin Man* | 10/17/2024 |
| Ang Li | Design and Implementation Leader | *Ang LI* | 10/17/2024 |
| Yueyihan Qi | Security Leader | [Yueyihan Qi](mailto:qyyh@bu.edu) | 10/17/2024 |
| Srujana N | QA Leader | *Srujana N* | 10/17/2024 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1.0.0** | **Srujana N** | **11/07/24** | **All testing scenarios covered**  **(UAT to be done when product fully ready)** |
|  |  |  |  |

[Testing Summary](#_sm5odwyvuk3j)

[Manuel Tests Reports](#_pqso2mbjyzx4)

[Automated Testing Reports](#_mtfbusfb0eq3)

[Testing Metrics](#_rijyjeu2ojqa)

[References](#_15tmymhipvdv)

[Glossary](#_8n34lvocupub)

# Testing Summary

In this section, you will summarize what was tested, who is involved in testing, testing techniques used, and testing result. You may have the following tests

* + Unit Testing

**Purpose**: Focuses on verifying individual components (like functions, methods, or classes) to ensure they work correctly in isolation.

**Who**: Typically done by developers.

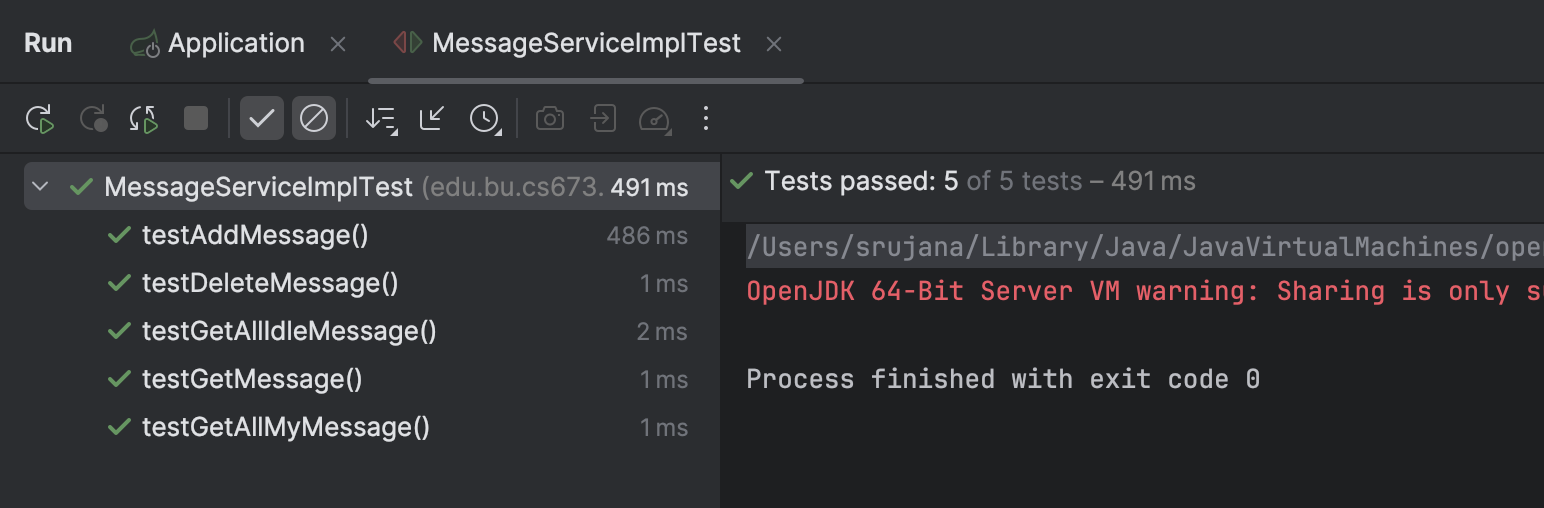
**Techniques**: Often uses frameworks like JUnit (for Java) or pytest (for Python) to automate tests for small units.

**Results**: Aims to catch bugs at the smallest level, ensuring that each piece of code behaves as expected independently.

MessageService Unit Testing - SRUJANA

The MessageServiceImplTest is a runnable unit test file that includes the following test methods, utilizing JUnit and Mockito:

* void testAddMessage()
* void testDeleteMessage()
* void testGetMessage()
* void testGetAllMyMessage()
* void testGetAllIdleMessage()



**Test Details:**

**testAddMessage()**:

* Verifies that a message can be added successfully. Mocks the insert method, checks if it returns true, and confirms insert was called once.

**testDeleteMessage()**:

* Verifies message deletion by ID. Mocks deleteByPrimaryKey, expects true, and ensures deleteByPrimaryKeywas called once.

**testGetMessage()**:

* Checks if a message can be retrieved by ID. Mocks selectByPrimaryKey, expects a non-null result matching the mocked message, and confirms selectByPrimaryKey was called once.

**testGetAllMyMessage()**:

* Verifies retrieval of all messages for a user. Mocks getMyMessage, along with user and idle item data retrieval, then checks if the returned list matches the expected messages.

**testGetAllIdleMessage()**:

* Checks retrieval of all messages for a specific idle item. Mocks getIdleMessage and user data retrieval, then confirms the returned list matches the expected messages.

**Result:** All unit test cases have passed for MessageService

ItemService Unit Testing - HungHsu(Allen)

The ItemServiceImplTest is a runnable unit test file that includes the following test methods, utilizing JUnit and Mockito:

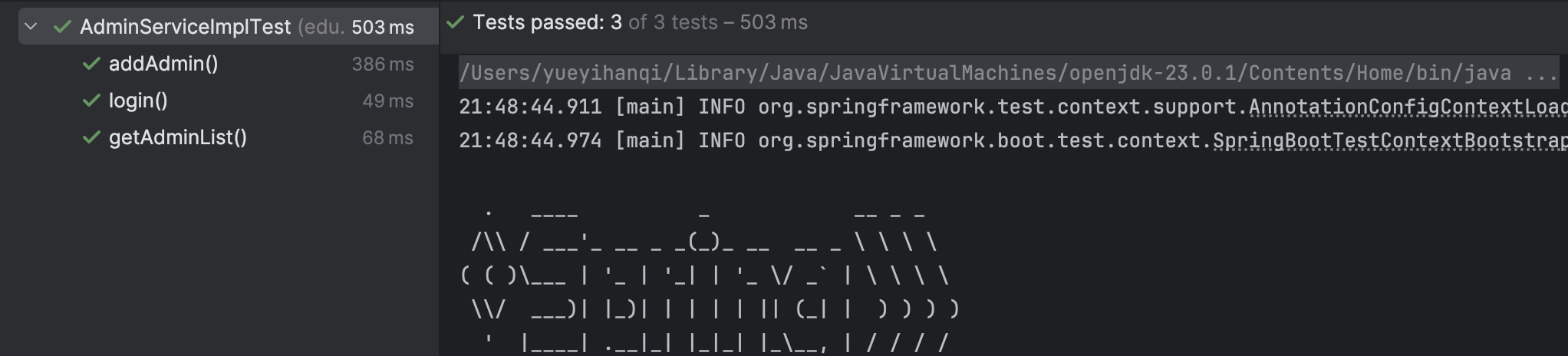
* void testAddNewItem()
* void testGetItem()
* void testGetAllItemByUser()
* void testUpdateItem()
* void testRemoveItem()
* void testSearchItemLabel()
* void testSearchItem()

**Result:** All unit test cases have passed for ItemService

### *AdminService Unit Testing - Yueyihan Qi*

The AdminServiceImplTest is a runnable unit test file that includes the following test methods, utilizing JUnit and Spring Boot Test:

* void addAdmin()
* void login()
* void getAdminList()



**Test Details:**

addAdmin(): Tests the creation of a new admin user

Verifies successful admin creation

Validates admin attributes (account number, password, name)

login(): Tests admin authentication

Verifies successful login with valid credentials

Validates returned admin object properties

getAdminList(): Tests retrieval of paginated admin list

Verifies list is not null and contains items

Validates pagination limits

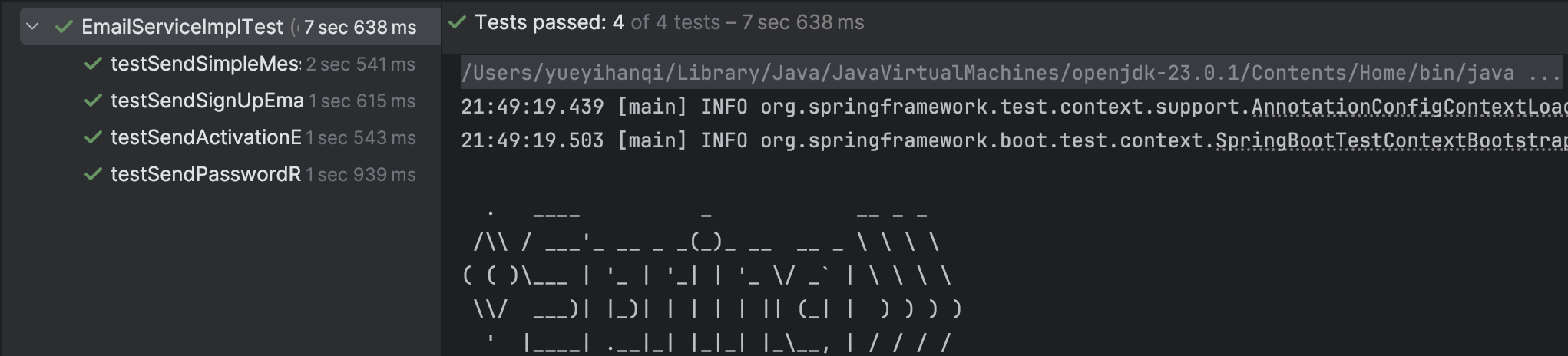
Confirms presence of test admin in the list

**Result:** All unit test cases have passed for AdminService

### *EmailService Unit Testing - Yueyihan Qi*

The EmailServiceImplTest is a runnable unit test file that includes the following test methods, utilizing JUnit and Spring Boot Test:

* void testSendSimpleMessage()
* void testSendSignUpEmail()
* void testSendActivationEmail()
* void testSendPasswordResetEmail()



Test Details:

testSendSimpleMessage(): Tests basic email sending functionality

Verifies successful sending of simple email messages

Tests with subject, content, and recipient

testSendSignUpEmail(): Tests sign-up email functionality

Verifies sending of registration confirmation emails

Includes activation token and registration link

testSendActivationEmail(): Tests account activation emails

Verifies sending of account activation emails

Includes activation token and activation link

testSendPasswordResetEmail(): Tests password reset functionality

Verifies sending of password reset emails

Includes reset token and reset link

**Result:** All unit test cases have passed for EmailService

* + Integration testing

**Purpose**: Verifies the interactions between different components or modules of an application to ensure they work together as expected.

**Who**: Often done by developers and sometimes by QA engineers.

**Techniques**: Uses techniques like stubbing, mocking, or using APIs to simulate interactions and check for data flow between modules.

**Results**: Identifies issues in the interfaces and integration points, ensuring that combined modules interact as intended.

Entire Application has been deployed and interaction between different modules working as expected - Integration testing successful

* + System Testing

**Purpose**: Tests the entire application as a whole, verifying that it meets the specified requirements and functions as intended.

**Who**: Primarily performed by QA engineers.

**Techniques**: Involves end-to-end testing, testing under real conditions, and using scenarios that mimic how a user might interact with the system.

**Results**: Ensures the entire system behaves as expected, with any failures indicating issues in the system’s integration or requirements fulfillment.

All functionality is working as expected overall. Only the admin flow test is pending.

* + Acceptance Testing

**Purpose**: Conducted to validate that the system meets business requirements and is ready for delivery to the customer or end-user.

**Who**: Usually performed by end-users, clients, or a QA team on behalf of stakeholders.

**Techniques**: User-driven testing, often with predefined acceptance criteria that the system must meet for approval.

**Results**: Determines if the application is ready for production; if accepted, it is approved for deployment, or it is revised if rejected.

Will be done in Final Presentation as this is User Acceptance Testing when the final product is deployed.

* + Regression Testing

**Purpose**: Ensures that recent code changes haven’t adversely affected the existing functionality.

**Who**: Typically done by QA engineers, often automated due to the repetitive nature of these tests.

**Techniques**: Automated testing tools (e.g., Selenium, JUnit) re-run previously executed tests on updated builds.

**Results**: Detects unintended bugs introduced by new changes, verifying that the software still functions as expected.

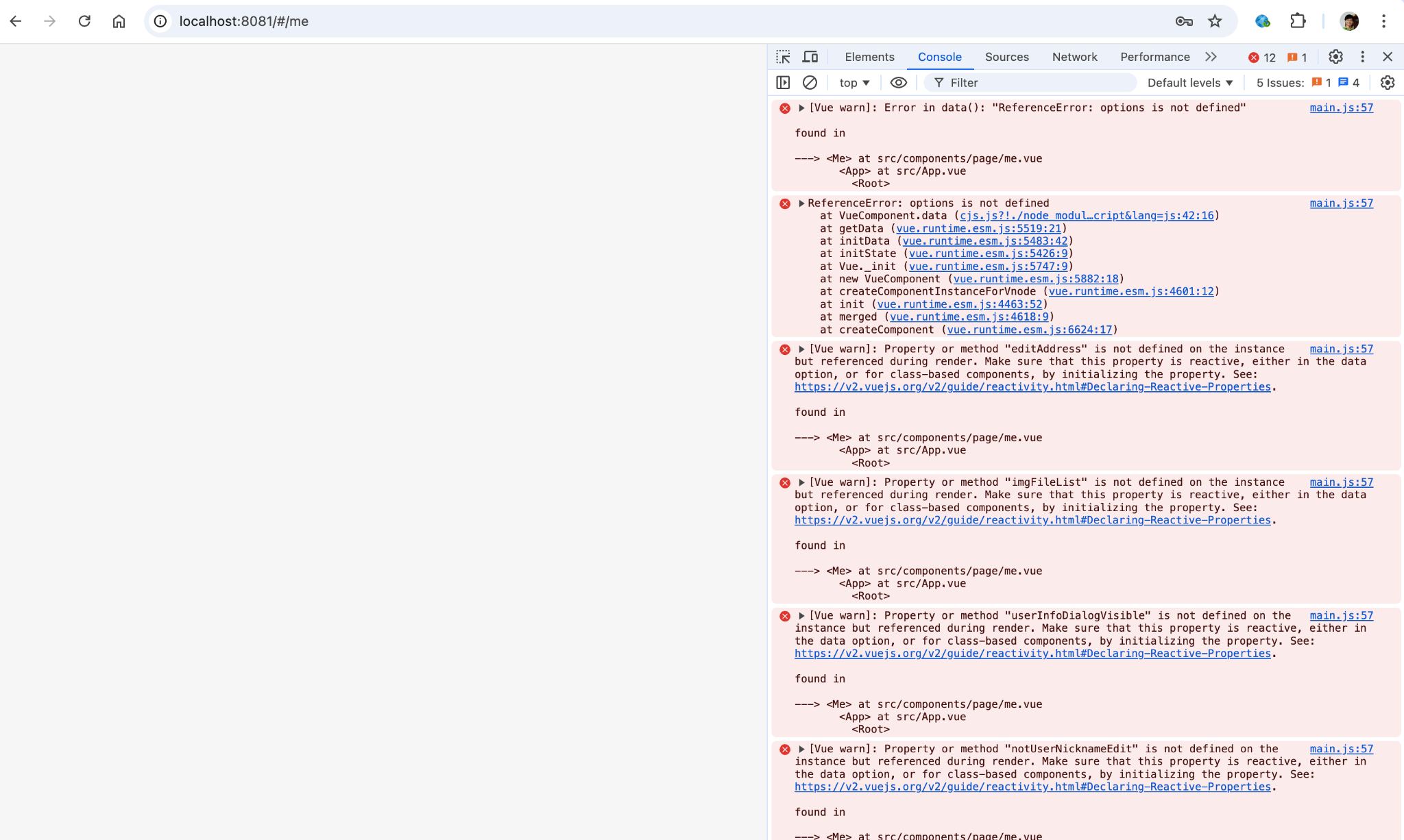
Regression testing has been implemented through continuous automation testing and unit testing.

# Manual Testing Report

### *Frontend Testing - Yingtong Zhou*

* Test case ID: #1.ReferenceError: options is not defined
* New or old: New
* Test items: User Profile Page
* Test priority (high/medium/low): high
* Dependencies (to other test case/requirement if any):
* Preconditions (if any): User has logged in to the application
* input data:
* Test steps: Navigate to the user profile page
* Postconditions:
* Expected output: It should display the user profile
* Actual output: Blank page with ReferenceError in the console. Several properties and methods are not defined. No api errors in the Network tab.
* Pass or Fail: Pass
* Bug id/link: (this should link to your github issue id) <https://bu-cs673f24a2team2.atlassian.net/jira/software/projects/SCRUM/boards/1/timeline?selectedIssue=SCRUM-108>
* Additional notes: Already solved

Bug screenshot:



**LINK FOR SPREADSHEET CONTAINING TEST CASES -** <https://docs.google.com/spreadsheets/d/1JZI21AtTpdGgYhOkqtyVDQNbj0Sd42BSK3ljJVku4qA/edit?usp=sharing>

# Automated Testing Report

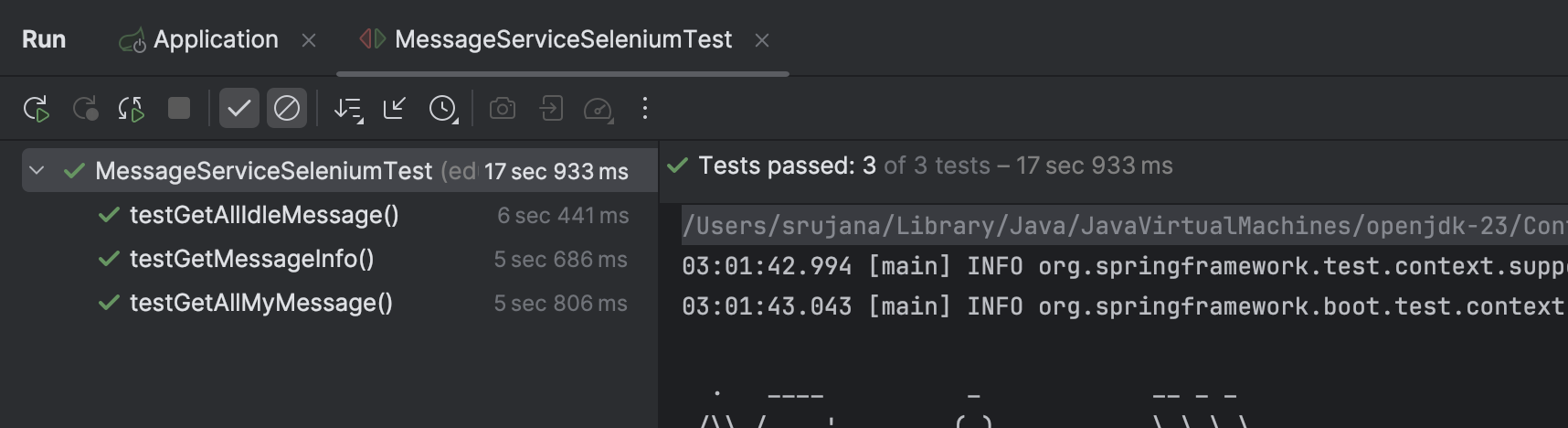
Describe briefly the automated testing you have done, including where the test code resides in your code repository, what test frameworks are used, and the screen shots or **generated testing report. (JUnit auto generated)**

### *MessageService Automated Testing - SRUJANA*

Automated testing for the MessageService has been implemented using **Selenium** within a Spring Boot application. The tests are executed in the Chrome web browser and include the following methods:

* testGetMessageInfo()
* testGetAllIdleMessage()
* testGetAllMyMessage()

These automated tests are designed to run predefined checks to ensure that the code functions correctly, even after the latest updates.



**Result:** All automated test cases implemented for GET APIs have passed for MessageService. Tests happen automatically with a delay(thread.sleep) of 5 seconds for observation purposes.

Path where test file resides in backend repository: src/test/java/edu/bu/cs673/secondhand/service/Selenium/MessageServiceSeleniumTest.java

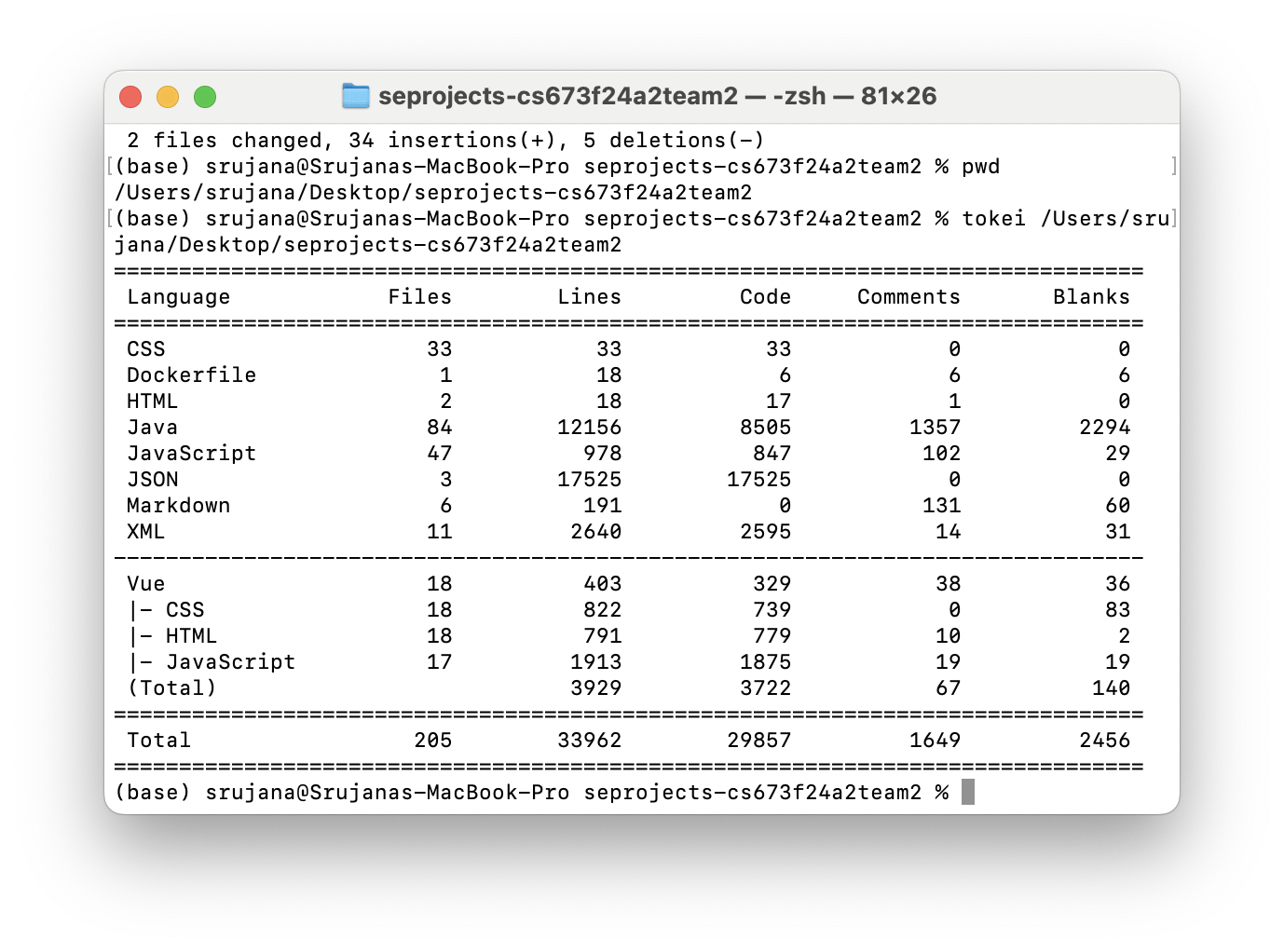
Command to run and generate test report - *mvn clean test*

After the tests run, the reports will be generated in the **target/surefire-reports** directory, typically as .txt and .xml files.

# Testing Metrics

In this section, you shall report any metrics used for the evaluation, e.g. # of test cases, test coverage, defects rate, etc.

| Metric Name | Description |
| --- | --- |
| Product Metric:  Size & Complexity | The number of lines of code (~ 33K) and different tools integration along with the number of files can show product size and complexity. - Tokei (img attached below) |
| Process Metric: Effort (in person hours) | Cost of person hours used (personal) ~ 200 Hrs |
| In-Process Metric:  Defect Repair rate | The rate at which the defects are fixed as soon as they are found out to minimize loss  25 defects in this sprint, defined a 3-day target to fix each defect. 20 of those defects were fixed within 3 days  Defect Repair Rate=(20/25)×100=80% |
| In-Process Metric: Defect arrival pattern | The pattern of defect arrivals offers deeper insights into varying levels of quality in the field.  Week\_1 - 7, Week\_2 - 5, Week\_3 - 3, Week\_4 - 10 |
| Product Metric:  Defect Density | the defects according to size of software in terms of lines of code or function point, etc. i.e., it measures code quality per unit. It is perfect for our second-hand trading market e-commerce website  For 50 defects (total) and 29,857 lines of code, the defect density is:   * **0.00167 defects per LOC** * **1.67 defects per 1,000 LOC** |



# References

1. XAMPPRocky. \*tokei\*. GitHub. [https://github.com/XAMPPRocky/tokei](https://github.com/XAMPPRocky/tokei).

2. Spring Framework Documentation. \*Testing Spring MVC with HtmlUnit and WebDriver\*. Spring. https://docs.spring.io/spring-framework/reference/testing/spring-mvc-test-framework/server-htmlunit/webdriver.html

3. Atlassian. \*Types of Software Testing\*. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

# Glossary

LOC - Lines of Code

Defect - A deviation from the expected quality of code