**CS673 Software Engineering** 

**Team 1 - Chit Chat**

**Software Test Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Deasia Little | QA Leader | *Deasia Little* | September 21, 2025 |
| Masih Vahida | Team Leader | Masih Vahida | September 21, 2025 |
| Robin Roeoesli | Requirement Leader | *Robin Roeoesli* | September 21, 2025 |
| Jordyn Lipsey | Configuration Leader | *Jordyn Lipsey* | September 21, 2025 |
| Ardit Briskaj | Design and Implementation Leader | *Ardit Briksaj* | September 21, 2025 |
| All | Security Leader |  | September 21, 2025 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1** | **Deasia Little** | **September 21, 2025** | **Initial Draft** |
| **2** | **Deasia Little** | **October 5, 2025** | **Updates for Iteration 2** |

[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, when to test, testing techniques used, and testing result. You may have the following tests

This Software Test Document defines the testing strategy, scope, metrics, and quality assurance approach for the *Chit Chat* web application throughout its development lifecycle.

The testing approach follows a progressive validation model, beginning with manual validation and advancing toward automated end-to-end testing.

#### **Scope of Testing**

* **Integration Testing:**Validate interactions between the frontend (React/Vite) and backend (NestJS/PostgreSQL) layers.
* **System Testing:**End-to-end testing of complete user workflows, including login, messaging, and chat persistence.
* **Acceptance Testing:**Automated tests aligned with user stories and acceptance criteria.
* **Regression Testing:**Automatically executed in the CI pipeline on each pull request and merge to ensure stability after code changes.

**Note:** Unit testing was not included due to time constraints and project priorities. The focus remained on functional integration and system testing that simulate real user interaction.

# Manual Testing Report

In this section, you will give a detailed description of each manual test case performed and the result. If this is a previous You shall list what are existing tests developed in the previous semester and what are new tests developed currently.

Here is a sample template that can be used for each test case. For system tests or acceptance tests, you may also include some screenshots.

* Test case ID, name
* New or old:
* Test items: (what do you test )
* Test priority (high/medium/low)
* Dependencies (to other test case/requirement if any):
* Preconditions: (if any)
* input data:
* Test steps:
* Postconditions:
* Expected output:
* Actual output:
* Pass or Fail:
* Bug id/link: (this should link to your github issue id)
* Additional notes:

(You can use an additional spreadsheet for this section as well)

Manual testing was performed throughout all development phases to confirm functionality before automation.

Focus areas included:

* User registration and authentication
* Message sending and receiving
* Help dialog visibility and responsiveness
* Enter-key message submission behavior
* Chat history persistence

Manual testing results were recorded using a shared test case spreadsheet. All tests passed prior to automation.

# 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.

Automation was implemented using Playwright to ensure consistency and reproducibility of tests across environments.

| Component | Details |
| --- | --- |
| Framework | Playwright |
| Configuration File | tests/e2e/playwright.config.ts |
| Execution Path | tests/e2e |
| Focus Areas | Authentication, chat persistence, help dialog, message sending, routing |
| Reports | tests/e2e/playwright-report/index.html and summary.html |
| CI/CD Integration | Automated tests run in GitHub Actions; results uploaded as build artifacts |

The Playwright framework was also integrated with GitHub Actions for continuous testing feedback and defect tracking.

# Testing Metrics

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

Code quality and complexity were continuously measured using custom scripts housed in the scripts/ directory.

| Metric | Script | Description |
| --- | --- | --- |
| Lines of Code (LOC) | scripts/count-loc.sh | Measures total LOC for source and test code |
| Function Count | scripts/count-function.sh | Counts functions to gauge modularity |
| Cyclomatic Complexity | scripts/eslint-report.sh | Assesses code complexity via ESLint |
| E2E Coverage | tests/e2e/playwright-report  tests/e2e/[auth-login.spec.ts](http://auth-login.spec.ts)  tests/e2e/[autoscroll.spec.ts](http://autoscroll.spec.ts)  tests/e2e/[chat-guest.spec.ts](http://chat-guest.spec.ts)  tests/e2e/[chat-history.spec.ts](http://chat-history.spec.ts)  tests/e2e/[enter-key.spec.ts](http://enter-key.spec.ts)  tests/e2e/[hello.spec.ts](http://hello.spec.ts)  tests/e2e/[help-community.spec.ts](http://help-community.spec.ts)  tests/e2e/[helpers.ts](http://helpers.ts)  tests/e2e/[sidebar.spec.ts](http://sidebar.spec.ts)  tests/e2e/[smoke.spec.ts](http://smoke.spec.ts)  tests/e2e/typing-indicator.spec.ts | Index.html and summary.html tracks Playwright pass/fail ratio |

All metrics reports are automatically uploaded as build artifacts during CI runs.

**Defect Management**

Defects are tracked directly in GitHub Issues, both manually and through automation triggered by failed Playwright tests.

Common categories of defects observed:

* UI element changes breaking automated tests
* Redirects to login on page reload
* Weak password validation and missing username checks
* Typing indicator visibility inconsistencies

Defects are triaged and assigned to the responsible developer based on the related feature or module.

**Continuous Integration (CI/CD)**

Testing and quality checks are integrated into the CI/CD pipeline:

* Metrics scripts and Playwright tests run automatically on each push or pull request.
* Artifacts and HTML reports are uploaded for instructor review.
* Future versions will include automatic defect linking via GitHub CLI scripts.

# References

* + Project Repo: <https://github.com/BUMETCS673/cs673olf25project-cs673olf25_team1>

# Glossary

* + E2E (End-to-end): Tests that run the app in a browser and verify user-visible flows
  + Smoke Test: A small set of test that check critical paths
  + Cyclomatic complexity: A metric indicating the number of independent paths through code. A high number means the code could be harder to test and maintain.
  + Pass Rate: Percentage of tests that passed in a run.