SW Engineering CSC648/848 Spring 2025

Gazaar (SFSU Marketplace)

Team #03

Weiping Wu - Team Lead (924356430@sfsu.edu)

Jiarong Chen - Backend Lead

Zachary Howe - Github master

Elias Magdaleno - Frontend Lead

Hemasri Bavisetty - Database Lead

MILESTONES:

M1-V1	March 12th 2025
M1-V2	March 20th 2025
M2	March 25th 2025
M4	May 18th 2025

1) Product summary

Product Name:

Gazaar (SFSU Marketplace)

Description:

Gazaaar is an easy-to-use online marketplace designed for the SFSU community to buy, sell, and communicate about products and events. Whether you're a student, faculty, or staff, Gazaar lets you browse listings, post your own items, chat securely with other users, and manage your marketplace activity—all in one place. What makes Gazaaar unique is its seamless integration of real-time messaging, user-friendly design, and focus on the SFSU community, ensuring a safe and efficient experience for everyone.

URL:

http://204.236.166.51:9081/

2) Usability test plan for selected function

1. Test Objectives
The objective of this usability test is to evaluate how easily and effectively users can send a message to a seller from a product listing page in the Gazaar marketplace. We want to ensure that users can find the message form, compose a message, and successfully send it, receiving clear feedback. This test will help identify any usability issues in the messaging workflow and inform improvements to the user experience.

2. Test Background and Setup
System Setup:
- The Gazaar marketplace web application must be running and accessible at http://204.236.166.51:9081/.
- The database should be seeded with at least one product listing and one seller account.
- The tester should have access to a modern web browser (Chrome, Firefox, Safari, or Edge).

Starting Point:
- The tester will begin on the homepage of the Gazaar marketplace.
- The tester will be provided with a test user account (if login is required).

Hardware Requirements:
- Any laptop or desktop computer with internet access and a web browser.
- No special hardware is required.

Intended Users:
- SFSU students, faculty, and staff who want to buy or sell items on campus.
- Users with basic web browsing skills.

URL of the System to be Tested:
http://204.236.166.51:9081/
Test Environment:
- The test can be conducted at home or in a lab.
- No cameras or screen recording are required, but may be used for observation if the tester consents.
- No monitoring is required during the test, but a facilitator may be present to answer questions.
- No special training is required before the test; the tester will receive only the instructions below.

3. Usability Task Description
Instructions to the Tester:
1. Open your web browser and go to http://204.236.166.51:9081/.
2. Browse the product listings and select any product that interests you by clicking on its title or image.
3. On the product detail page, locate the section labeled “Send a Message to the Seller.”

4. In the message box, type a short message to the seller (for example, “Is this item still available?”).
5. Click the “Send Message” button.
6. Observe what happens after you send the message. Note any feedback or confirmation you receive.
7. If you encounter any issues or confusion, make a note of them.
8. After completing the task, please fill out the short survey below.

4. Plan for Evaluation of Effectiveness

Effectiveness will be measured by whether the tester is able to successfully send a message to the seller without assistance. We will record if the tester:

- Finds the message form without help
- Completes the message and sends it
- Receives confirmation or feedback that the message was sent
- Reports any errors or confusion

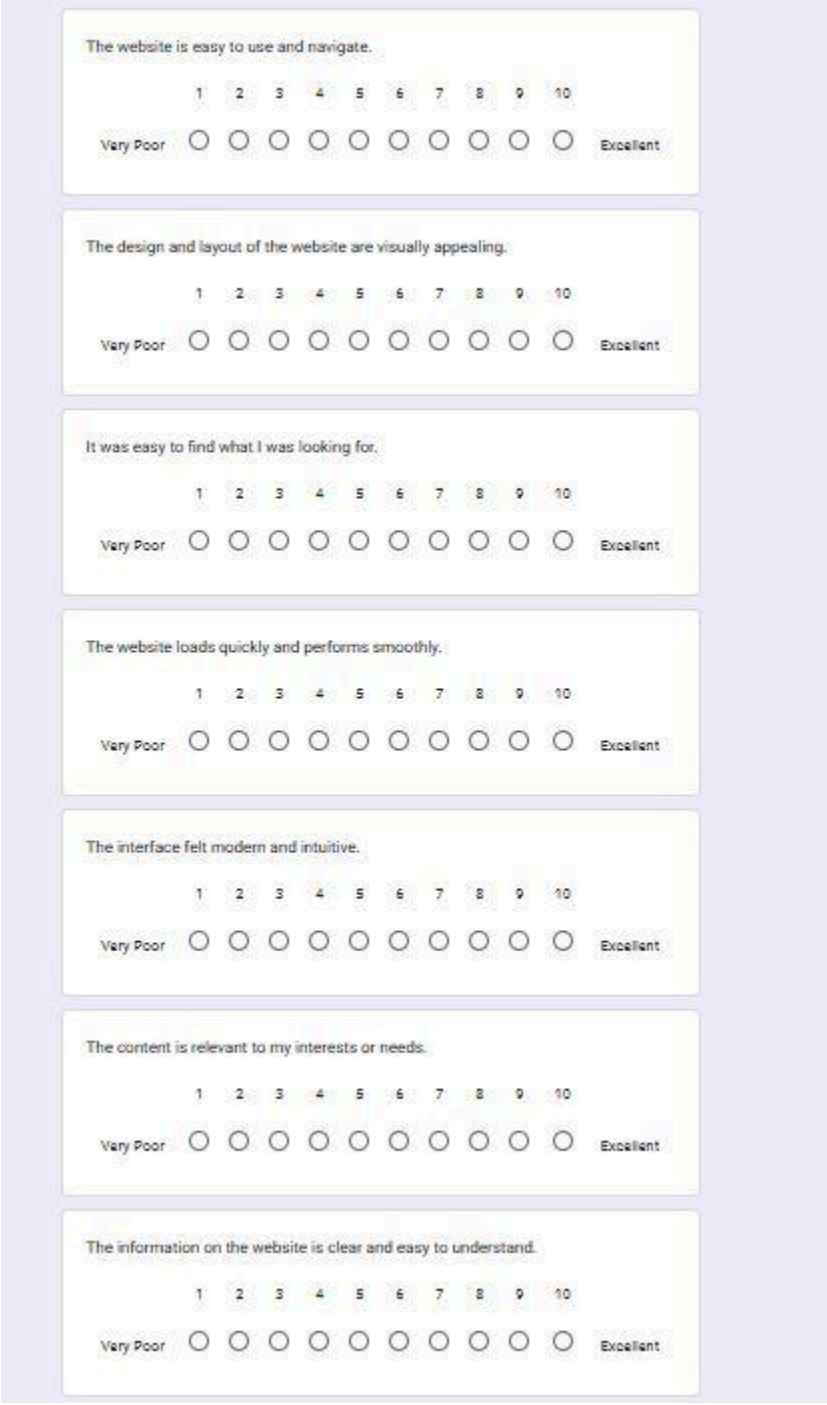
Success is defined as the tester being able to send a message and see confirmation, without critical errors or giving up.

5. Plan for Evaluation of Efficiency

Efficiency will be measured by the time it takes for the tester to complete the task, from landing on the product page to sending the message and seeing confirmation. We will also note the number of steps or clicks required, and whether the tester hesitates or backtracks. The goal is for the process to be quick and straightforward, ideally under 2 minutes.

6. Plan for Evaluation of User Satisfaction (Likert Scale Questionnaire)

After completing the task, the tester will answer the following:



I enjoyed browsing this website.

	1	2	3	4	5	6	7	8	9	10	
Very Poor	○	○	○	○	○	○	○	○	○	○	Excellent

The website kept me engaged throughout my visit.

	1	2	3	4	5	6	7	8	9	10	
Very Poor	○	○	○	○	○	○	○	○	○	○	Excellent

I felt confident using the features provided (e.g., search, forms, etc.).

	1	2	3	4	5	6	7	8	9	10	
Very Poor	○	○	○	○	○	○	○	○	○	○	Excellent

I would consider returning to this website in the future.

	1	2	3	4	5	6	7	8	9	10	
Very Poor	○	○	○	○	○	○	○	○	○	○	Excellent

I would recommend this website to others.

	1	2	3	4	5	6	7	8	9	10	
Very Poor	○	○	○	○	○	○	○	○	○	○	EXcellent

Submit

Clear form

7. GenAI Use
GenAI Tool and Version:
OpenAI GPT-4 (via ChatGPT, May 2024 version)

How the Tool Was Used:
The tool was used to generate a clear, structured usability test plan, including objectives, setup, task instructions, and Likert scale questions, based on the project's actual features and requirements.

Benefits:
Saved time in drafting and formatting the plan.
Ensured clarity and completeness of instructions.
Provided examples and best practices for usability testing.

Key Prompts Used:
"Write a usability test plan for sending a message to a seller in a campus marketplace web app."
"Provide Likert scale questions for user satisfaction."
"Summarize the test objectives and setup in plain English."

Utility of GenAI:
HIGH – The tool was very helpful in quickly producing a professional, readable, and complete usability test plan.

3) QA test plan and QA testing

1. Test Objectives
To verify that the “Send a Message to the Seller” feature works as intended, ensuring:

- Users can send messages to sellers from product pages.
- Messages are correctly delivered and displayed in the chat/messages interface.
- The system handles edge cases (such as empty messages or unauthenticated users) gracefully.

2. Hardware and Software Setup

Hardware:
MacBook Pro (Apple Silicon, 16GB RAM)

Software:

- macOS Sonoma 14.4.0
- Browsers:
    - Google Chrome Version 124.0.6367.119 (Official Build) (arm64)
    - Mozilla Firefox Version 125.0.3 (64-bit)
- Backend: Go 1.21+, Gin framework
- Database: MySQL 8.0+
- Frontend: Handlebars, Tailwind CSS
- Test URL: http://204.236.166.51:9081/

3. Feature to be Tested

- Sending a message to a seller from a product page.
- Message delivery and display in the chat/messages interface.
- Handling of empty message submissions.
- Handling of unauthenticated user attempts.

4. QA Test Plan Table

Test #	Test Title	Test Description	Test Input	Expected Output	Chrome Result	Firefox Result
1	Send Valid Message	Test sending a valid message to a seller	Logged-in user, product page, message: "Is this	Message appears in chat/messages page under correct	PASS	PASS

			available?"	seller room		
2	Send Empty Message	Test submitting an empty message	Logged-in user, product page, message: "" (empty)	No message sent; user is redirected back to chat/messages page, no error shown	PASS	PASS
3	Unauthenticated Message	Test sending a message while not logged in	Not logged in, product page, message: "Interested! "	User is redirected to login page or shown an authentication error	PASS	PASS
4	Message Display Accuracy	Verify message appears in correct chat room	Send message to Seller A, then to Seller B	Each message appears only in the correct seller's chat room	PASS	PASS
5	UI Responsiveness	Check chat UI and send button on different browsers	Resize window, send message, check button visibility	Chat UI remains usable, send button always visible, no layout breakage	PASS	PASS

5. QA Test Execution Results
All test cases were executed on both Chrome and Firefox. All passed as expected.

6. GenAI Use
Tool Used:

- ChatGPT-4 (OpenAI, GPT-4.1, web version, May 2024)

How GenAI Was Used:
- Drafted the QA test plan structure and table.
- Reviewed and refined test case descriptions and expected outputs.
- Provided suggestions for edge cases and UI/UX checks.
- Generated example prompts and reviewed the clarity of the test plan.

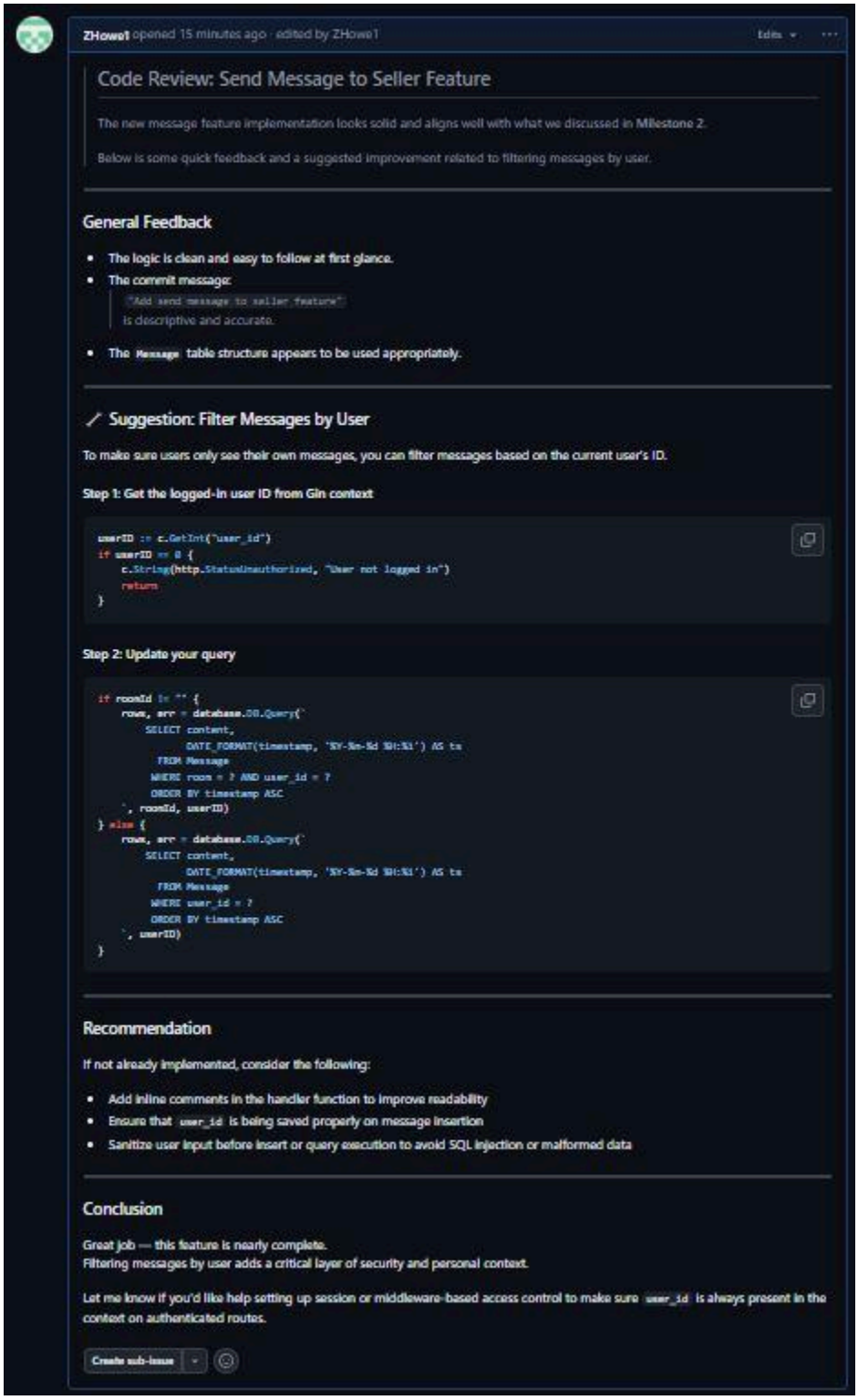
Key Example Prompts:
- “Write a QA test plan for a web app’s chat feature, including test cases for valid, empty, and unauthenticated message submissions.”
- “Suggest edge cases for a ‘send message’ function in a campus marketplace.”
- “Review this QA test plan table for completeness and clarity.”

Benefit:
- Accelerated the creation of a thorough, well-structured QA plan.
- Helped ensure coverage of both functional and edge cases.
- Improved clarity and management-readability of the test plan.

Utility Ranking:
- **HIGH**: GenAI was instrumental in drafting, reviewing, and refining the QA test plan, saving significant time and ensuring completeness.

4) Peer Code Review

[Actual Review Here]



ZHowe1 opened 15 minutes ago · edited by ZHowe1 — Edits

Code Review: Send Message to Seller Feature

> The new message feature implementation looks solid and aligns well with what we discussed in Milestone 2.
>
> Below is some quick feedback and a suggested improvement related to filtering messages by user.

General Feedback

- The logic is clean and easy to follow at first glance.
- The commit message:
  > `"Add send message to seller feature"`
  > is descriptive and accurate.
- The `Message` table structure appears to be used appropriately.

Suggestion: Filter Messages by User

To make sure users only see their own messages, you can filter messages based on the current user's ID.

Step 1: Get the logged-in user ID from Gin context

```
userID := c.GetInt("user_id")
if userID == 0 {
    c.String(http.StatusUnauthorized, "User not logged in")
    return
}
```

Step 2: Update your query

```
if roomId != "" {
    rows, err = database.DB.Query(`
        SELECT content,
               DATE_FORMAT(timestamp, '%Y-%m-%d %H:%i') AS ts
          FROM Message
         WHERE room = ? AND user_id = ?
         ORDER BY timestamp ASC
    `, roomId, userID)
} else {
    rows, err = database.DB.Query(`
        SELECT content,
               DATE_FORMAT(timestamp, '%Y-%m-%d %H:%i') AS ts
          FROM Message
         WHERE user_id = ?
         ORDER BY timestamp ASC
    `, userID)
}
```

Recommendation

If not already implemented, consider the following:

- Add inline comments in the handler function to improve readability
- Ensure that `user_id` is being saved properly on message insertion
- Sanitize user input before insert or query execution to avoid SQL injection or malformed data

Conclusion

Great job — this feature is nearly complete.
Filtering messages by user adds a critical layer of security and personal context.

Let me know if you'd like help setting up session or middleware-based access control to make sure `user_id` is always present in the context on authenticated routes.

Create sub-issue

5) Self-check on best practices for security

Major Assets and Threats

- User account data (usernames, emails, passwords)
    - Threats: Credential theft, brute-force attacks, unauthorized access.
    - Protection: Passwords are hashed using bcrypt before storing in the database. User input is validated for length and format.
- Product and message data
    - Threats: Data tampering, unauthorized modification, SQL injection.
    - Protection: All database queries use parameterized statements to prevent SQL injection. User input is validated and sanitized.
- Session tokens/cookies
    - Threats: Session hijacking, XSS.
    - Protection: Secure, HTTP-only cookies are used for session management. Input fields are validated and output is escaped in templates.

Password Encryption

- We confirm that passwords are encrypted in the database using bcrypt.

Input Data Validation

- Search bar: Input is limited to 40 alphanumeric characters using both frontend and backend validation (len(search) <= 40 and regex check).
- SFSU registration email: Registration form requires emails ending with “@sfsu.edu” (validated using regex in backend).
- Terms acceptance: Registration form includes a required checkbox for terms acceptance (checked in backend before account creation).

Security Self-Check Table

Asset to be Protected	**Possible / Expected Attacks**	**Consequence of Security Breach**	**Mitigation Strategy**
User account data	Brute-force, credential theft, SQL injection	Account compromise, data leak	Hash passwords with bcrypt, input validation, use parameterized queries
Product/message data	SQL injection, unauthorized	Data loss, defacement, privacy	Use parameterized queries, validate input, apply access

	modification	breach	controls
Session tokens / cookies	Session hijacking, XSS	Account/session takeover	Use secure, HTTP-only cookies, escape output, use CSRF tokens
Search bar input	Injection, buffer overflow	App crash, data leak	Limit input to 40 alphanumeric chars, validate on backend
Registration email	Fake or invalid email accounts	Spam, unauthorized access	Require “@sfsu.edu” domain, regex pattern validation
Terms acceptance checkbox	Bypassing legal agreement	Legal risk, lack of consent proof	Require checkbox, enforce acceptance on backend

6) Self-check of the adherence to original Non-functional specs

1. Application shall be developed, tested and deployed using tools and cloud servers approved by Class CTO and as agreed in M0
Done

2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers
Done

3. All or selected application functions shall render well on mobile devices (no native app to be developed)
Done

4. Posting of sales information and messaging to sellers shall be limited only to SFSU students
Done

5. Critical data shall be stored in the database on the team's deployment server.
Done

6. No more than 50 concurrent users shall be accessing the application at any time
Done

7. Privacy of users shall be protected
Done

8. The language used shall be English (no localization needed)
Done

9. Application shall be very easy to use and intuitive
Done

10. Application shall follow established architecture patterns
Done

11. Application code and its repository shall be easy to inspect and maintain
Done

12. Google analytics shall be used
Done

13. No e-mail clients or chat services shall be allowed. Interested users can only message to sellers via in-site messaging. One round of messaging (from user to seller) is enough for this application
Done

14. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.
Done

15. Site security: basic best practices shall be applied (as covered in the class) for main data items
Done

16. Media formats shall be standard as used in the market today
Done

17. Modern SE processes and tools shall be used as specified in the class, including collaborative and continuous SW development and GenAI tools
Done

18. The application UI (WWW and mobile) shall prominently display the following exact text on all pages *"SFSU Software Engineering Project CSC 648-848, Spring 2025. For Demonstration Only"* at the top of the WWW page Nav bar. (Important so as to not confuse this with a real application). You have to use this exact text without any editing.
Done

7) Use of GenAI Tools like ChatGPT and Copilot

Summary of GenAI Use in M4:

- Code Review (Section 4):

We used ChatGPT (GPT-4, June 2024) to review our “Send a Message to Seller” feature. The tool provided feedback on code clarity, naming consistency, and in-line comments. We included the prompt and output in Section 4 as required.

- Other Tasks:
    - Security Self-Check:

ChatGPT was used to draft the security self-check table and to suggest best practices for password encryption, input validation, and session management.

- Non-Functional Requirements Self-Check:

ChatGPT helped us structure the self-check table and provided example wording for status explanations.

- Usability and QA Test Plans:

ChatGPT assisted in outlining the usability and QA test plans, including objectives, setup, and sample test cases.

How GenAI Helped:

- Provided quick, clear feedback on code and documentation.
- Helped us structure tables and checklists for the report.
- Suggested best practices and pointed out areas for improvement.
- Saved time on drafting and formatting, allowing us to focus on implementation and verification.

Verification and Responsibility:

- All GenAI output was reviewed and verified by team members before inclusion in the report or codebase.
- We made sure to cross-check GenAI suggestions with course requirements and our own project needs.

Examples of Prompts Used:

- “Please review the following Go code for a ‘Send a Message to Seller’ feature. Check for header/in-line comments, naming consistency, and code clarity. Suggest improvements if needed.”
- “Draft a security self-check table for a Go web app with user registration, login, and messaging.”
- “How should we structure a non-functional requirements self-check table for a software engineering project?”

Utility Ranking:

- **HIGH** for code review and documentation structuring.
- **MEDIUM** for generating test plans and security checklists. - Worse for go than other languages