CSC648-848
Spring 2025

RedJr: SFSU Marketplace

Ria Thakker (Team Lead, rthakker@sfsu.edu), Rohith Gannoju (Backend Lead), Danny Duenas (Frontend Lead), James Richards (GitHub Master), Eric/Po-Han Chen(Frontend Dev)

Milestone 4, 05/08/2025

History

Date Submitted	05/08/2025
Date Revised	

1. Product Summary

Product Name: REDJR Marketplace

Description:

In today's fast-paced university environment, students and faculty often struggle to find affordable and convenient ways to buy and sell essential items like textbooks, electronics, and furniture. While general online marketplaces exist, they are not tailored to students and often come with unrelated listings and scams. REDJR is a marketplace platform designed specifically for the SFSU community, offering a trusted space to buy and sell items safely. The platform features verified SFSU-only access, item listings, keyword search, and a messaging system for buyers and sellers. Unique tools like on-campus meetups, a dashboard to manage listings, and optional item rentals help REDJR stand out as a student-first solution.

Final P1 Functional Commitment:

- Unregistered users can browse and search for items on the home, listing, and about pages.
- Unregistered users can register using a verified SFSU email address.
- Registered users can log in and out.
- Registered users can post item listings with images and descriptions.
- Registered users can search for listings using keywords and category.
- Registered users can send offers to sellers via the messaging system.
- Registered users can message and respond to other registered users.
- Registered users can view a dashboard showing posted items, sold items, and saved-for-later items.
- Registered users who agree to a purchase can set up a sale with meeting location and time on campus.
- Registration form includes email validation (must contain "@sfsu.edu"), required field indicators, and terms agreement checkbox.
- A site administrator shall be able to approve all items uploaded to the site before they are available for public view.

Deployment URL: http://18.191.88.25/

2. Usability Test Plan

Selected Function: Contact Seller via Message System ("Make Offer")

1. Test Objectives: To evaluate if users can successfully contact sellers and understand the messaging flow. This tests clarity, usability, and task completion.

2. Test Background and Setup:

- **System setup:** REDJR Marketplace deployed on AWS (http://18.191.88.25/).
- **Starting point:** Logged-in homepage.
- **Hardware/software:** Laptop/Desktop with Chrome or Firefox.
- **Target users:** SFSU students, 18–30 years old.
- **Environment:** Remote, user tested from home, no camera required, screen sharing optional. No training prior to test.

3. Usability Task Description:

You are browsing REDJR and found a listing titled "Laptop - $1000". Please contact the seller and send a message offering to meet on campus. Confirm your message was sent.

4. Evaluation of Effectiveness:

Success is determined by whether the user locates the item, opens it, sends a message, and receives confirmation. We'll keep track of task completion.

5. Evaluation of Efficiency:

The time taken from finding the item to confirming the message will be tracked. A duration of under 90 seconds is deemed efficient.

6. Likert Questionnaire (User Satisfaction):

- The message feature was easy to find.(1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, 5 = Strongly Agree)
- I felt confident sending a message to the seller.(1 = Strongly Disagree → 5 = Strongly Agree)
- I would use this feature again in the future.(1 = Strongly Disagree → 5 = Strongly Agree)

7. GenAI Use: ChatGPT (GPT-4, May 2025)

Use: Drafted usability instructions, suggested Likert questions.
Prompt: "Generate a usability task flow for a marketplace message system."
Utility Rank: HIGH

3. QA Test Plan

1. Test Objectives: Validate that the core messaging feature (Make Offer) works across browsers.

2. HW and SW Setup:

Devices: Windows 11 laptop

Browsers: Chrome 124, Firefox 125

App URL:

3. Feature to be Tested: Message system (sending a message to the seller)

4. QA Test Plan Table:

Test #	Title	Description	Input	Expected Output	Chrome	Firefox
T01	Valid Message	Send a message	“Hi, is this still available?”	The message appears in the seller’s inbox	PASS	PASS
T02	Empty Message Block	Try sending a blank message	“”	Error: message cannot be empty	PASS	PASS
T03	Navigation Test	Navigate back to the item after sending	Click back after sending	Returns to the item listing page	PASS	PASS

5. Tested on Two Browsers: Results above (Chrome and Firefox)

6. GenAI Use: Tool: ChatGPT (GPT-4)

Use: Helped formulate test cases and expectations
Prompt: "Suggest 3 QA test cases for a marketplace message feature."
Utility Rank: MEDIUM

7. Developer Note:

During QA testing, we discovered a mismatch between the selected category in the search bar and the actual filtered results. For example, selecting "Electronics" showed books, and "Books" returned nothing. After investigation, it was found that the issue stemmed from mislabeled category values in the DEV database. Our team lead (Ria) created a new listing to confirm and submitted a pull request to fix the category ID mapping during item posting. This ensures consistent behavior across both the production and development databases.

4. Peer Code Review

Code Feature: Dashboard interface layout and frontend integration

Review Process:

Po-Han Chen (Eric) implemented the dashboard.html page, which displays selling listings, sold items, and saved-for-later items. This involved integrating dynamic sections with backend API responses and ensuring visual consistency with the rest of the site. James Richards reviewed the code for HTML structure, layout consistency, and route correctness.

He verified that:

- The navbar styling matched other pages (aligned with navbar-search.css)
- The dashboard sections (selling/sold/saved) rendered properly
- The saved items section displayed correctly, even if empty

GenAI (ChatGPT) was also used to assist in code review.

Prompt: "Review a Flask HTML template for a user dashboard with item cards and conditional sections. Suggest improvements for clarity and reusability."

ChatGPT Suggestions:

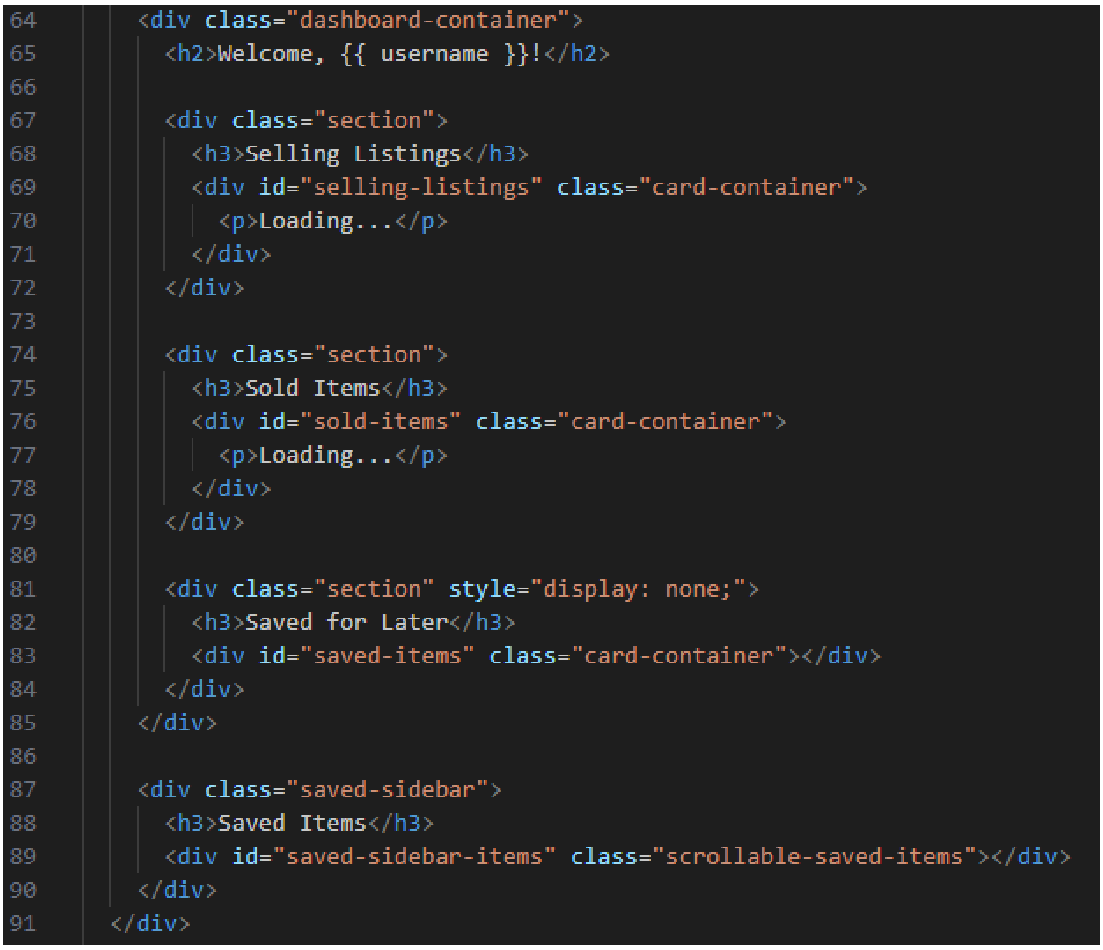
Use Jinja conditionals to handle empty item sections

Confirm use of consistent class names for CSS reuse

Suggested semantic HTML tags for readability

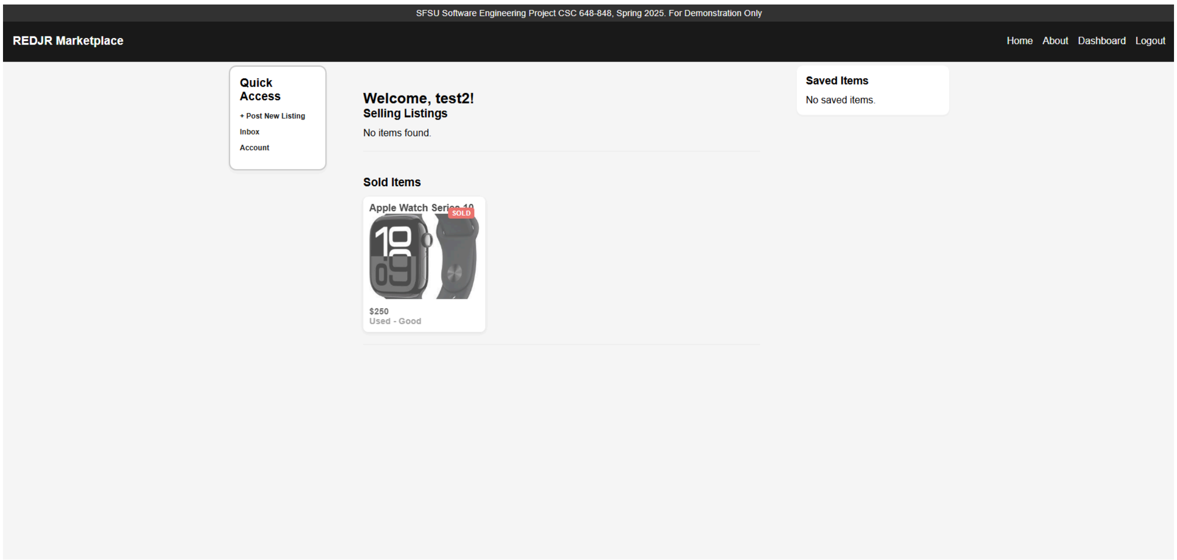
Submitted:

- Screenshot of dashboard HTML layout



```
    <div class="dashboard-container">
      <h2>Welcome, {{ username }}!</h2>

      <div class="section">
        <h3>Selling Listings</h3>
        <div id="selling-listings" class="card-container">
          <p>Loading...</p>
        </div>
      </div>

      <div class="section">
        <h3>Sold Items</h3>
        <div id="sold-items" class="card-container">
          <p>Loading...</p>
        </div>
      </div>

      <div class="section" style="display: none;">
        <h3>Saved for Later</h3>
        <div id="saved-items" class="card-container"></div>
      </div>
    </div>

    <div class="saved-sidebar">
      <h3>Saved Items</h3>
      <div id="saved-sidebar-items" class="scrollable-saved-items"></div>
    </div>
  </div>
```

- Screenshot of rendered dashboard



SFSU Software Engineering Project CSC 648-848, Spring 2025. For Demonstration Only

REDJR Marketplace

Welcome, test2!

Selling Listings

No items found.

Sold Items

Apple Watch Series 10 SOLD

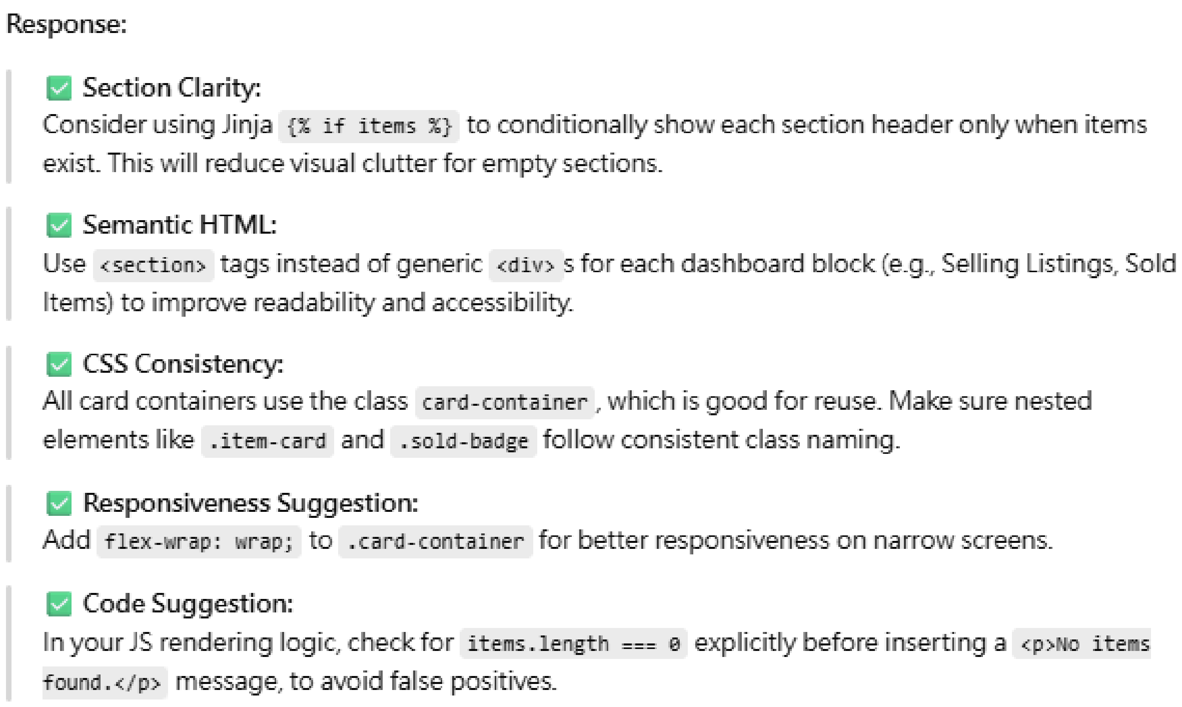
$250

Used - Good

Saved Items

No saved items.

- Screenshot of ChatGPT review output



Response:

> ✅ Section Clarity:
> Consider using Jinja `{% if items %}` to conditionally show each section header only when items exist. This will reduce visual clutter for empty sections.

> ✅ Semantic HTML:
> Use `<section>` tags instead of generic `<div>`s for each dashboard block (e.g., Selling Listings, Sold Items) to improve readability and accessibility.

> ✅ CSS Consistency:
> All card containers use the class `card-container`, which is good for reuse. Make sure nested elements like `.item-card` and `.sold-badge` follow consistent class naming.

> ✅ Responsiveness Suggestion:
> Add `flex-wrap: wrap;` to `.card-container` for better responsiveness on narrow screens.

> ✅ Code Suggestion:
> In your JS rendering logic, check for `items.length === 0` explicitly before inserting a `<p>No items found.</p>` message, to avoid false positives.

5. Security Self-Check

Asset	Threats	Consequences	Mitigation Strategy
User Passwords	DB leaks, weak hash	Account takeover	Stored as bcrypt hash
Listing Data	Unauthorized edit	Wrong info, spam	Access control per user
Message	Snooping, spam	Privacy breach	Only visible to involved users, message sanitization
Email	Spoofing, fake signups	Fake users	SFSU email domain required

Validations:

- Passwords encrypted in DB (bcypt)
- Input fields validated
- Email: must contain “@sfsu.edu”

6. Non-Functional Requirement Self-Check

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

    a. 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
    a. DONE
3. All or selected application functions shall render well on mobile devices (no native app to be developed)
    a. ON TRACK
4. Posting of sales information and messaging to sellers shall be limited only to SFSU students
    a. DONE
5. Critical data shall be stored in the database on the team's deployment server.
    a. DONE
6. No more than 50 concurrent users shall be accessing the application at any time
    a. ON TRACK
7. Privacy of users shall be protected
    a. ON TRACK
8. The language used shall be English (no localization needed)
    a. DONE
9. Application shall be very easy to use and intuitive
    a. ON TRACK
10. Application shall follow established architecture patterns
    a. DONE
11. Application code and its repository shall be easy to inspect and maintain
    a. DONE
12. . Google analytics shall be used
    a. ON TRACK
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
    a. DONE
14. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.
    a. DONE
15. Site security: basic best practices shall be applied (as covered in the class) for main data items
    a. DONE
16. Media formats shall be standard as used in the market today
    a. DONE
17. Modern SE processes and tools shall be used as specified in the class, including collaborative and continuous SW development and GenAI tools
    a. 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.

   a. DONE

7. GenAI Use Summary

Used GPT-4 for:

- Drafting the usability test plan, including task flow and Likert scale wording
- Suggesting QA test case titles, descriptions, and validation logic
- Reviewing dashboard layout code for structure clarity, not design
- Recommending improvements on conditional rendering and mobile responsiveness

Clarification:

The UI layout and structure (HTML/CSS) were designed and implemented by team members.

GenAI was only used to provide feedback or suggestions, which were then reviewed and selectively applied by the team.

Utility Evaluation: MEDIUM–HIGH

Final Note: All GenAI suggestions were human-reviewed and modified as needed.