AGORA

SFSU Community Market Place

Software Engineering CSC 648
Spring 2025

Team 02

Team Members
Nathan Delos Reyes - Team Lead - ndelosreyes@sfsu.edu
Xiaoxuan Wang - Github Master
Jose Ramirez- Front-end Assistant
Ranbir Atkar - Front-end Lead
Jeshwanth - Back-end Lead

Milestone	Date Submitted
M4	5/19

TABLE Of CONTENTS

Product Summary

Agora is a community marketplace platform developed specifically for the SFSU campus, enabling students, faculty, and staff to buy, sell, and trade items and share services or skills. It supports transactions involving textbooks, dorm essentials, electronics, and student-led services such as tutoring or tech support. Designed to foster a secure, user-friendly, and community-focused environment, Agora integrates search filters, direct messaging, reviews, and content moderation, while preserving user privacy. What makes Agora unique is its dual support for both product listings and skill-sharing posts, tailored entirely to the needs of the SFSU campus community.

Major Committed Functions

Unregistered Users

- Can create a new account using SFSU email and password
- Can view listings with photos and descriptions
- Can view skill-sharing posts
- Can use keywords to search for items or services

Registered Users

- Can buy listed products
- Can create listings to sell products
- Can offer services or skills
- Can upload images and describe their products
- Can message sellers directly within Agora
- Can search for products using keywords
- Can Scroll and browse listings and services
- Can view skills shared by other users
- Can rate sellers after a transaction
- Can post tutoring or skill-sharing offers

Admins

- Can Approve newly created posts before they go live

Website URL: https://team02sfsu.org/

Usability Test Plan for Search Function

1. Test objectives:
   Evaluate whether the users can discover, execute, and complete the "Search" workflow without assistance; verify the interface supports efficient item discovery; and capture subjective satisfaction with clarity, speed, and overall ease of use. Success criteria map to ISO usability metrics of effectiveness, efficiency, and satisfaction.
2. Test background and setup:
   System setup & starting point: The production-like build is deployed to https://team02sfsu.org/. The database contains a few real test listings across all categories. Testers begin on the home page and are not logged in.

   Hardware / software the tester needs: Laptop or desktop; The browser of their choice; stable internet; optional external mouse.

   Intended user: SFSU students and faculty with no or moderate web experience (We can put the people/personas here) If possible 1 to 4 participants to balance novice vs experienced marketplace users.

   URL of the system: https://team02sfsu.org/

   Test environment: Remote, participant's home. Session captured via Zoom screen-share with webcam on; facilitator observes silently and only intervenes if participant is stuck. No prior training, just a brief preamble ("We're testing the site, not you.").

3. Usability Task description
   Instructions read / shown to the tester
   " You need to find an item provided by the facilitator."
   1. Using only the search bar, type a query you think will locate the item."
   2. Browse the search results and open one listing you believe fits your needs."
   3. Tell the facilitator when you think you have completed the task."
4. Plan for evaluation of Effectiveness
   Metric: task-completion rate and critical-error count.
   Method: log three events (a) search submitted, (b) results page viewed, (c) listing detail opened. A run is *effective* if all three occur, in order, within 5 min and the opened listing's title mentions "Product picked by facilitator" or equivalent.
   User/Tester 1: Completed all three events within 2 min.
   User/Tester 2: Completed all three events within 1 min.
   User/Tester 3: Completed all three events within 50 seconds
   Plan for Evaluation of efficiency
   Metric: time-on-task plus interaction cost.

Method: automatically timestamp from first click inside the search bar to the moment the participant says “done.”
Tester 1: 1:45
Tester 2: 50 seconds
Tester 3: 11 seconds

5. Plan for Evaluation of user satisfaction (Likert scale questionnaire)
   Google form:
   https://docs.google.com/forms/d/e/1FAIpQLSfP61gVhcaxj0iPAWYVlFYDdRuKja9ny7lILBHFPf_DkZZl1A/viewform?usp=dialog

#	Statement	Scale(1 2 3 4 5)
1.	“It was easy to locate the search function.”	
2.	“The search results matched what I was looking for.”	
3.	“I am satisfied with how quickly I found a suitable listing.”	
4.	“The top results seemed highly relevant to my query.”	
5.	“The information shown for each listing was sufficient for deciding which one to open.”	
6.	“Search results appeared quickly enough for me.”	

   Survey Answer:
   Question 1: 5, 3, 4
   Question 2: 2, 4, 4
   Question 3: 4, 4, 4
   Question 4: 3, 3, 4
   Question 5: 4, 2, 4
   Question 5: 5, 5, 4
   Comments said:
   Just adding more listings to appear.
   Having the search button closer or next to the search bar would be better placement.
   Also the description for the items does not show completely, it's cut off.

6. GenAI use
   What GenAI tool and version you used
   I used ChatGPT (GPT-3.5) and Claude 3 Sonnet (3.7).

   Explain briefly how you used the tool and what benefit it offered.
   For milestone 4, I used GenAI tools primarily to create a structured outline for the usability test plan of our project's search function. I compared outputs from both ChatGPT and Claude, selecting the one from ChatGPT because it provided a clearer and more organized framework. Although ChatGPT’s initial structure was strong, I made further adjustments to address minor inaccuracies and improve clarity. The benefit of using these tools was a significant time saving in drafting and organizing the test plan.

Provide brief examples of key examples and prompts
Make a good usability test plan structure for my team's search function. Using the provided info double check the structure is a standard for Usability Testing. An example output would be the statements used for the Likert scale questionnaire; at least the first three were produced by AI.

RANK utility of GenAI here as LOW or MEDIUM or HIGH
MEDIUM

Quality Assurance Test Plan – Search Function

Test Objectives: The QA testing will evaluate the Search functionality of the student marketplace platform, which allows users to look up items and services such as electronics, books, accessories, and furniture. The goal is to ensure the feature correctly filters and displays listings based on user queries, functioning smoothly across browsers.

Hardware and Software Setup:
Hardware:

- Laptop: Dell Laptop XPS Windows 11
- Minimum 4GB RAM, 2GHz Processor

Software:

- Web Browsers: Google Chrome (v124+), Mozilla Firefox (v125+)
- Backend Server Running via python or flask run
- Localhost URL (or live if deployed):
  http://localhost:5000

Test Background Setup: The student marketplace website allows users to search for listings such as books, electronics, accessories, and furniture. The Search function plays a key role in helping users quickly locate relevant items or services. This QA testing ensures the search feature functions correctly, returns accurate results, and provides a consistent experience across browsers.

Task Description: The QA test will simulate user behavior by entering various search terms into the search bar. This includes full and partial keywords, case variations, invalid terms, and special characters. The tester will check if the results are relevant, error messages are shown when appropriate, and the interface remains stable and responsive.

Plan for Evaluation of Effectiveness:

Effectiveness will be evaluated by:

- The accuracy of returned search results.
- Handling of edge cases like empty input, special characters, and incorrect queries.
- Case insensitivity and partial matching behavior.
- The visibility and clarity of "no results found" messages.

The search function will be considered effective if all outputs meet expectations across test cases and browsers.

Plan for Evaluation of Efficiency:

Efficiency will be measured by:

- The time it takes for search results to load (ideally <2 seconds).
- The number of steps or interactions required to complete a search.
- Whether results update smoothly and allow quick follow-up searches.
- Cross-browser consistency in performance (Chrome and Firefox).

Plant for Evaluation of User Satisfaction:

User satisfaction will be evaluated based on:

- Ease of locating and using the search bar.
- Confidence in the accuracy of the search results.
- Clarity of design and messaging (especially for failed searches).
- Overall perceived usefulness and user-friendliness of the feature.

Feedback may be collected from users or testers through informal surveys or observation to assess how satisfying and intuitive the search experience was.

Feature to Test

Search Function – Located in the site's header or homepage, it allows users to search for listed items and services by keyword (e.g., "laptop", "book", "chair").

Test #	Test Title	Test Description	Test Input	Expected Output	Test Results (Chrome / Firefox)
1	Valid Keyword Search	Ensure correct results display for valid search	laptop	Listings with "laptop" in title/description appear	PASS / PASS
2	Case Insensitive Search	Verify search works regardless of case sensitivity	BoOk	Listings with "book" appear	PASS / PASS
3	No Result Found	Search for non-existent term	spaceship	Message: "No results found"	PASS / PASS
4	Special Characters Input	Input special characters to test stability	!@#%&*	"No results found" or input ignored gracefully	PASS / PASS
5	Empty Search	Click search without input	*(empty)*	No action or show all listings	PASS / PASS

Results Summary

All test cases were executed in both Chrome and Firefox. The search feature passed all tests, showing reliable behavior across edge cases and browsers. The system appropriately handled invalid input and displayed meaningful feedback to the user.

GenAI Use

For this QA test plan, I used ChatGPT (GPT-4, May 2024 version) to assist with formatting the document into a clear, professional layout. Specifically, ChatGPT helped me structure the QA test table, standardize the wording of test descriptions and expected outputs, and ensure the final presentation met academic and usability expectations.

I provided the test criteria and context for the search function, and ChatGPT formatted the inputs into a QA plan table consistent with class guidelines. No functional test ideas or analysis were generated by the tool—those were based entirely on our class lectures and my personal observations during testing.

Example Prompt Used:

> "Help me format this QA test into a clear table plan based on these inputs I did for my site.

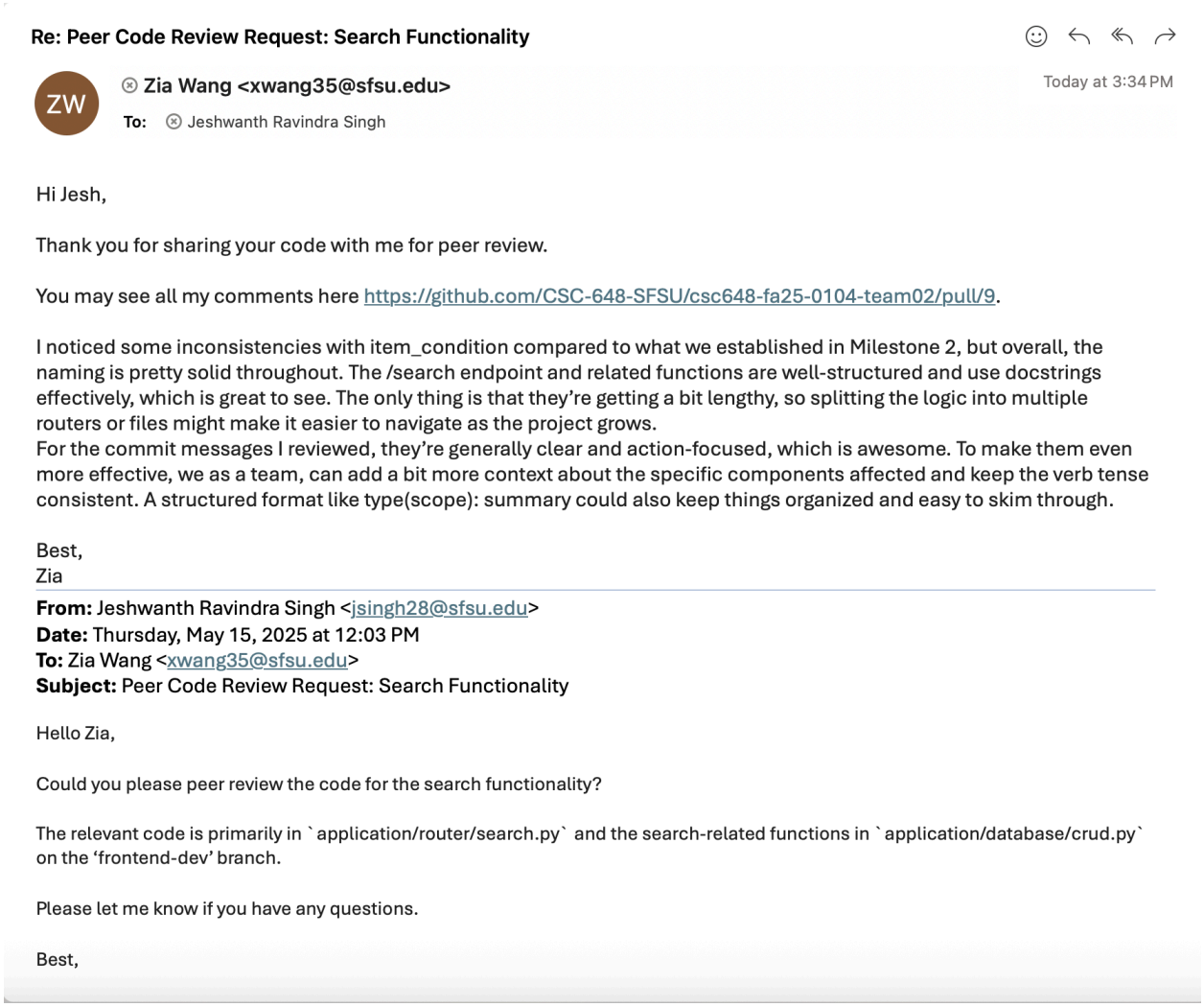
Utility of GenAI Use: MEDIUM

The tool's utility was limited to document presentation and clarity.

Peer Code Review

One team member reviewed the portion of code related to our search function and the commit messages in the last pull request through the GitHub Review feature.

1. Communications:



Re: Peer Code Review Request: Search Functionality

ZW **Zia Wang <xwang35@sfsu.edu>** Today at 3:34 PM

To: Jeshwanth Ravindra Singh

Hi Jesh,

Thank you for sharing your code with me for peer review.

You may see all my comments here https://github.com/CSC-648-SFSU/csc648-fa25-0104-team02/pull/9.

I noticed some inconsistencies with item_condition compared to what we established in Milestone 2, but overall, the naming is pretty solid throughout. The /search endpoint and related functions are well-structured and use docstrings effectively, which is great to see. The only thing is that they're getting a bit lengthy, so splitting the logic into multiple routers or files might make it easier to navigate as the project grows.
For the commit messages I reviewed, they're generally clear and action-focused, which is awesome. To make them even more effective, we as a team, can add a bit more context about the specific components affected and keep the verb tense consistent. A structured format like type(scope): summary could also keep things organized and easy to skim through.

Best,
Zia

From: Jeshwanth Ravindra Singh <jsingh28@sfsu.edu>
Date: Thursday, May 15, 2025 at 12:03 PM
To: Zia Wang <xwang35@sfsu.edu>
Subject: Peer Code Review Request: Search Functionality

Hello Zia,

Could you please peer review the code for the search functionality?

The relevant code is primarily in `application/router/search.py` and the search-related functions in `application/database/crud.py` on the 'frontend-dev' branch.

Please let me know if you have any questions.

Best,

2. Screenshots of Reviews on GitHub

Zia-wang commented 2 hours ago

Review of commit messages:

The commit messages show a good effort to maintain consistency in capitalization and generally stick to a straightforward, action-oriented style. It's good to keep messages concise and effective for quick browsing. However, there are a few areas that could use some refinement.

1. Focusing on the scope of the changes can help provide more context. For instance, instead of just stating what was added or fixed, briefly mentioning which specific component or feature was affected can make it easier for others to track the changes.
2. Maintaining consistent verb tense and spelling will enhance clarity and professionalism (e.g., correcting refistration to registration and chnages to changes).
3. Structuring each commit with a consistent format, such as type(scope): summary, can also be helpful.

Zia-wang reviewed 17 hours ago View reviewed changes

`application/router/search.py`

Zia-wang 17 hours ago

Review for the `/search` endpoint:

1. The use of `item_condition` is inconsistent with the `condition` established in the DATA section of Milestone 2. Other naming is consistent throughout.
2. Good use of docstrings to explain the functions.
3. Good practice of error handling.

Overall, the `search.py` is well-structured but a bit lengthy making it less readable. To improve readability and as the project grows, the team can split it into multiple routers.

Reply...

Resolve conversation

`application/database/crud.py`

Zia-wang 17 hours ago

Review for the `get_listings` and `get_categories` function:

1. Use of `item_condition` with `search.py` but inconsistent with the naming established in Milestone 2.
2. Consistent naming practices in functions related to the /categories endpoint.
3. Good practice of delegating the logic to the `crud` layer, keeping endpoints clean and focused on request/response handling.

The code for the search function is well-structured with mostly consistent naming practices. The minor fix could be to separate the logic into more files to improve readability.

Reply...

3. GenAI Interaction and Output

Reviews of the *`get_listings`* function in *`application/database/crud.py`* using Copilot code review feature in Visual Studio Code:

1. The parameter status is used to filter listings by their status, but the comment mentions it was "changed from available_only." For clarity and consistency, consider updating the comment to reflect the current usage of status without referencing the old parameter name.
Suggested change:

```
-     status: Optional[str] = None  # Changed from available_only, None means all statuses
+     status: Optional[str] = None  # Filter listings by their status (e.g., "available", "sold"). None means all statuses.
```

2. The case statement in the query prioritizes matches in the title, which is a useful feature. However, this can be computationally expensive for large datasets. Consider indexing the title column or evaluating whether this prioritization is necessary for all queries.
No change found to suggest.

3. The comment "Add skill sharing filter" is redundant since the parameter is_skill_sharing is self-explanatory. Consider removing or rephrasing the comment to avoid redundancy.
Suggested change:

```
-     is_skill_sharing: Optional[bool] = None, # Add skill sharing filter
+     is_skill_sharing: Optional[bool] = None,  # Filter listings by skill sharing type if specified
```

Reviews of the `search_listings` function in `application/router/search.py` using the Copilot code review feature in Visual Studio Code:
1. The endpoint lacks a clear description of the expected behavior for edge cases, such as when no results are found. Consider documenting whether an empty result set is returned or an error is raised.
Suggested change:

```
+     - If no results are found, an empty result set will be returned.
```

2. The q parameter is optional but lacks a default value in the function signature. Explicitly setting q: Optional[str] = None would improve consistency with other parameters.
Suggested change:

```
-     q: Optional[str] = Query(None, description="Search query for title, description, or keywords"),
+     q: Optional[str] = None,
```

3. The category_id parameter uses 0 to represent "all categories," but this logic is not enforced in the crud.get_listings call. Ensure that category_id=category_id if category_id and category_id > 0 else None is consistently applied.

Suggested change:
```
-     category_id: Optional[int] = Query(None, description="Filter by category ID. 0 means all categories."),
+     category_id: Optional[int] = Query(None, description="Filter by category ID. Use 0 to represent all categories."),
-             category_id=category_id if category_id and category_id > 0 else None,
+              category_id=None if category_id == 0 else category_id,
```

4. The default page_size of 20 might not be optimal for all use cases. Consider making this configurable via environment variables or application settings.
Suggested change:
```
-     page_size: int = Query(20, ge=1, le=100, description="Number of results per page."),
+     page_size: int = Query(
+         int(os.getenv("DEFAULT_PAGE_SIZE", 20)),
+         ge=1,
+         le=100,
+         description="Number of results per page."
+     ),
```

5. The logging statement is verbose and concatenates multiple variables. Using f-strings is good, but consider breaking it into multiple lines or simplifying the message for better readability.
Suggested change:
```
-     logging.info(
-         f"Searching listings with q='{q}', category_id={category_id}, status='{status}', "
-         f"min_price={min_price}, max_price={max_price}, item_condition='{item_condition}', "
-         f"is_skill_sharing={is_skill_sharing}, page={page}, page_size={page_size}"
-     )
+     logging.info("Searching listings with the following criteria:")
+     logging.info(f"Query: '{q}', Category ID: {category_id}, Status: '{status}'")
+     logging.info(f"Price Range: {min_price} - {max_price}, Condition: '{item_condition}'")
+     logging.info(f"Skill Sharing: {is_skill_sharing}, Page: {page}, Page Size: {page_size}")
```

6. The crud.count_search_results call duplicates much of the filtering logic from crud.get_listings. This could lead to performance issues if the database query is complex. Consider optimizing by combining these queries if possible.
Suggested change:
```
-         # Get results and total count
-         results = crud.get_listings(
-             db,
-             skip=skip,
-             limit=page_size,
```

```
-            search=q,
-            category_id=category_id if category_id and category_id > 0 else None,
-            min_price=min_price,
-            max_price=max_price,
-            item_condition=item_condition,
-            is_skill_sharing=is_skill_sharing,
-            status=status # Pass status parameter
-        )
-
-        total_count = crud.count_search_results(
-            db,
+        # Get results and total count in a single query
+        results, total_count = crud.get_listings_with_count(
+            db,
+            skip=skip,
+            limit=page_size,
```

7. The exception handling block raises a generic HTTPException with a 500 status code. This could expose sensitive information in production logs. Ensure that sensitive details are sanitized before logging.
Suggested change:

```
-        logging.error(f"Error in search_listings: {e}", exc_info=True)
-        raise HTTPException(status_code=500, detail="Internal server error during search")
+        sanitized_message = "An unexpected error occurred during the search operation."
+        logging.error(f"{sanitized_message} Details: {str(e)}", exc_info=True)
+        raise HTTPException(status_code=500, detail=sanitized_message)
```

8. The log message "Found {total_count} results for search criteria" does not specify whether the count includes pagination or the total available results. Clarify this in the message.
Suggested change:

```
-        logging.info(f"Found {total_count} results for search criteria.")
+        logging.info(f"Found {total_count} total results for search criteria (before pagination).")
```

9. The SearchResults response model is returned without validation of the results structure. Ensure that the results list conforms to the expected schema to avoid runtime errors.
Suggested change:

```
-        return SearchResults(total=total_count, results=results)
+        # Validate each result against the ListingSchema
+        validated_results = [ListingSchema(**result.__dict__) for result in results]
+        return SearchResults(total=total_count, results=validated_results)
```

4. GenAI Use

GenAI Tool and Version:

I utilized Copilot version 1.322.0 for code review. The tool provides suggestions and comments for selected code sections, allowing for targeted analysis.

Usage and Benefits:

I focused on reviewing functions related to the search functionality by executing the Copilot code review command. The tool systematically identified potential issues and provided suggestions for each one, allowing me to review and decide whether to apply or discard each recommendation. This process was particularly useful in reinforcing good commenting practices, optimizing code structure, and identifying potential security vulnerabilities.

Additionally, Copilot provided inline comments that guided me in refining existing comments to ensure clarity and consistency. This feature is beneficial as developers often overlook the importance of clear comments, especially when they already understand the code. The tool's ability to prompt for more descriptive comments serves as a valuable reminder to maintain good documentation practices.

Key Examples and Prompts:

Improving Comments for Clarity:

Original: status: Optional[str] = None # Changed from available_only, None means all statuses

Suggested Change: status: Optional[str] = None # Filter listings by their status (e.g., "available", "sold"). None means all statuses.

Benefit: The suggested comment is more descriptive and avoids referencing outdated parameter names, enhancing readability.

Exception Handling and Security Concerns:

Original:

```
logging.error(f"Error in search_listings: {e}", exc_info=True)
raise HTTPException(status_code=500, detail="Internal server error during search")
```

Suggested Change:

sanitized_message = "An unexpected error occurred during the search operation."
logging.error(f"{sanitized_message} Details: {str(e)}", exc_info=True)
raise HTTPException(status_code=500, detail=sanitized_message)

Benefit: This adjustment reduces the risk of exposing sensitive information in production logs, promoting better security practices.

Schema Validation for Consistent Structure:

Original:
return SearchResults(total=total_count, results=results)

Suggested Change:
validated_results = [ListingSchema(**result.__dict__) for result in results]
return SearchResults(total=total_count, results=validated_results)

Benefit: Implementing schema validation ensures the response structure conforms to expected data models, reducing the likelihood of runtime errors.

Utility Ranking and Target Audience:

Based on my experience, I would rank the utility of Copilot for code review as Medium to High. It is particularly effective for catching common issues, promoting consistent commenting practices, and identifying potential security risks. However, while it provides valuable guidance, it is not a substitute for in-depth manual code review and developer insight.

Security Checklist

This section summarizes our security posture: the key assets we protect, the threats we anticipate, and our mitigation strategies. It also confirms specific practices around password handling and input validation.

Asset to Protect	Possible Attacks	Consequences of Breach	Mitigation Strategy
User Accounts (usernames, emails, passwords, roles)	• Credential stuffing • Phishing • Account takeover	• Identity theft • Unauthorized access to listings & messaging • Reputation loss	• Hash passwords with bcrypt • JWT-based sessions • HTTPS everywhere • Rate-limit login attempts • Periodic security audits
User-Generated Content (titles, descriptions, images)	• Spam & fake listings • Inappropriate content • XSS	• Scams & fraud • Platform reputation damage	• Pydantic schemas for strong typing & sanitization • Filename normalization (make_safe_filename) • Admin moderation workflow
Messaging / Communication	• Spam & harassment • Phishing • MITM attacks	• User abuse • Loss of trust • Data leaks	• HTTPS for all endpoints • Admin controls (future: spam filters, reporting tools) • Monitor suspicious patterns

Uploaded Images	• Malware embedding • Directory traversal via filename	• Server compromise	• Sanitize filenames • Restrict extensions & content-type • Store under the per-listing directories
Admin Interfaces	• Unauthorized access • Privilege escalation	• Complete platform takeover • Data destruction or leak	• Role-based access control • Dedicated "get_current_admin_user" dependency • Secure admin authentication flows

Confirmation of Key Practices

1. Password Encryption
    a. What we do: All passwords are hashed with bcrypt via the passlib library before storage.
    b. How to verify: See application/security.py → get_password_hash and verify_password.
2. Input Data Validation
    a. Search Bar
    b. Requirement: Up to 40 alphanumeric characters.
    c. Planned implementation:

```
from fastapi import Query
q: Optional[str] = Query(
    None,
    description="Search query",
    max_length=40,
    regex="^[A-Za-z0-9 ]*$"
)
```

3. SFSU Email Registration
    a. Requirement: Must end in @sfsu.edu.
    b. Implementation approach:

```
from pydantic import validator
```

```
class UserCreate(BaseModel):
    email: str
    # …

    @validator('email')
    def must_be_sfsu_email(cls, v):
        if not v.endswith('@sfsu.edu'):
            raise ValueError('Email must be an sfsu.edu address')
        return v
```

4. Terms & Conditions Acceptance
    a. Requirement: The Checkbox must be checked.
    b. Planned schema update:

```
class UserCreate(BaseModel):
    terms_accepted: bool
    # …

    @validator('terms_accepted')
    def terms_must_be_true(cls, v):
        if not v:
            raise ValueError('You must accept the terms and conditions')
        return v
```

5. General Pydantic Validation
    a. All request bodies use Pydantic models in application/schemas.py for type enforcement and required-field checks.
    b. Uploads use UploadFile and are further validated by our filename-sanitizer to prevent path traversal.

Non-functional Checklist

1. The application shall be developed, tested, and deployed using tools and cloud servers approved by Class CTO and as agreed in M0: DONE
2. The 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. The privacy of users shall be protected.: DONE
8. The language used shall be English (no localization needed): DONE
9. The application shall be very easy to use and intuitive: DONE
10. Application shall follow established architectural patterns: DONE
11. Application code and its repository shall be easy to inspect and maintain: ON TRACK
12. Google Analytics shall be used: 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: ON TRACK
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: ON TRACK
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: ON TRACK
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 to not confuse this with a real application). : DONE

Use of GenAI tools like ChatGPT and copilot

Jose Ramirez: For milestone 4 the way I used GenAI was to structure the way the usability test plan for our search function. To have a better structure I entered a prompt to both claude and chatgpt and used the better of the two. While the chatgpt was better in structuring the test plan I still had to make some adjustments as some parts did not make sense. Following the prompt I used "Make me a good usability test plan structure for my team's search function". I also gave it some info that was located in the slides. Overall I give the use of GenAI a 6.5 out of 10.

Copilot was used for peer code review. A more detailed GenAI usage report is in the code review section.