AGORA

SFSU Community Market Place

Software Engineering CSC 648
Spring 2025

Team 02

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

Milestone	Date Submitted
M2	3/25
Revised M2	3/26

Table of Contents

Executive Summary

Agora is a specialized buy-and-sell platform developed for the San Francisco State University (SFSU) community, including students, faculty, and staff. Recognizing the unique needs of our university community, Agora offers a secure, user-friendly marketplace designed to facilitate buying, selling, and trading university-related items such as textbooks, electronics, dorm necessities, and student services.

What makes Agora custom-built for SFSU is its focus on campus-specific needs, creating a community-driven platform tailored to support university life. Beyond buying and selling, Agora offers features such as skill sharing and listings for student-led services, including tutoring and tech help. The platform emphasizes ease of use, with powerful search and filtering functionalities that help students and faculty find what they need quickly and effortlessly, regardless of their technical skills.

Key functionalities include user messaging, enabling direct communication between buyers and sellers without requiring users to disclose their personal contact information. Product listings are thoughtfully designed, supporting multiple high-quality images and product descriptions to represent item conditions and features. Furthermore, a built-in rating system helps users establish and assess credibility, promoting accountability and transparency in every transaction.

Agora also prioritizes administrative efficiency by providing moderators with tools and analytics, empowering them to maintain platform integrity, manage disputes effectively, and address fraudulent or inappropriate activities. Automated flagging systems help identify suspicious listings, ensuring the marketplace remains secure, reliable, and compliant with community standards.

Our team is composed of SFSU Computer Science students committed to creating a meaningful and lasting solution that enhances campus life. As a student-led startup, we are connected to the real challenges and needs of our peers, and Agora reflects that firsthand experience. More than just a transactional platform, Agora is a solution designed to address and improve the buying and selling needs of the SFSU community. By investing in Agora, you are not only supporting a student-driven tech solution but also empowering students to build a stronger, safer, and more connected campus community.

Database:

Users - Stores all user accounts (buyer/seller/admin)
Listings - Products and services for sale
Categories - Hierarchical organization of listings
Listing_Images - Image paths for listing photos
Messages - Communication between users
Reviews - User feedback and ratings
Skill_Listings - Additional data for skill-sharing
Flagged_Content - Reported listings management

List of Main Data Items and Entities

1. Users

- user_id: Unique identifier for each user (Mandatory)
- email: User's email address, must be SFSU email (Mandatory)
- password_hash: Securely stored password (Mandatory)
- first_name: User's first name (Mandatory)
- last_name: User's last name (Mandatory)
- role: User's role in the system - buyer, seller, or admin (Mandatory)
- profile_image_path: File path to user's profile image (Optional)
- created_at: Timestamp when account was created (Mandatory)
- updated_at: Timestamp when account was last modified (Optional)
- is_verified: Boolean indicating if user's account is verified (Optional)

2. Categories

- category_id: Unique identifier for each category (Mandatory)
- name: Category name (Mandatory)
- parent_id: Reference to parent category for hierarchical structure (Optional)
- display_order: Integer for ordering categories in UI (Mandatory)
- is_active: Boolean indicating if category is active and visible (Mandatory)

3. Listings

- listing_id: Unique identifier for each listing (Mandatory)
- seller_id: Reference to the user who posted the listing (Mandatory)
- title: Title of the listing (Mandatory)
- description: Detailed description of the item (Mandatory)

- search_keywords: Additional keywords to improve search functionality (Optional)
- price: Decimal value representing the asking price (Mandatory)
- category_id: Reference to the listing category (Mandatory)
- condition: Item condition - new, like_new, good, fair, poor (Mandatory)
- status: Current listing status - available, pending, sold (Mandatory)
- is_skill_sharing: Boolean indicating if this is a skill sharing listing (Mandatory)
- created_at: Timestamp when listing was created (Mandatory)
- updated_at: Timestamp when listing was last modified (Mandatory)
- views_count: Integer tracking number of views (Optional)

4. Listing_Images

- image_id: Unique identifier for each image (Mandatory)
- listing_id: Reference to the associated listing (Mandatory)
- image_path: File path to the stored image (Mandatory)
- display_order: Order for displaying multiple images (Mandatory)
- is_primary: Boolean indicating if this is the main image (Mandatory)
- image_size: Size of image in bytes (Optional)
- width: Image width in pixels (Optional)
- height: Image height in pixels (Optional)
- created_at: Timestamp when image was uploaded (Mandatory)

5. Skill_Listings

- skill_id: Unique identifier for each skill listing (Mandatory)
- listing_id: Reference to the associated listing (Mandatory)
- skill_type: Type of skill being offered (Mandatory)
- experience_level: Provider's experience level - beginner, intermediate, advanced, expert (Mandatory)
- format: Delivery format - one_on_one, group, or video (Mandatory)

6. Messages

- message_id: Unique identifier for each message (Mandatory)
- sender_id: Reference to the user sending the message (Mandatory)
- receiver_id: Reference to the user receiving the message (Mandatory)
- listing_id: Reference to the listing being discussed (Mandatory)
- content: Text content of the message (Mandatory)
- created_at: Timestamp when message was sent (Mandatory)

7. Reviews

- review_id: Unique identifier for each review (Mandatory)

- reviewer_id: Reference to the user writing the review (Mandatory)
- reviewed_id: Reference to the user being reviewed (Mandatory)
- rating: Integer value from 1 to 5 (Mandatory)
- comment: Text feedback about the transaction (Optional)
- created_at: Timestamp when review was created (Mandatory)

8. Flagged_Content

- flag_id: Unique identifier for each flag (Mandatory)
- listing_id: Reference to the flagged listing (Mandatory)
- reporter_id: Reference to the user who reported the content (Mandatory)
- reason: Text explanation for the report (Mandatory)
- status: Flag status - pending, reviewed, resolved (Mandatory)
- created_at: Timestamp when flag was created (Mandatory)
- resolved_at: Timestamp when flag was resolved (Optional)
- admin_notes: Administrative notes about the resolution (Optional)

9. Search_Logs

- log_id: Unique identifier for each search log (Mandatory)
- user_id: Reference to the user who performed the search (Optional)
- search_query: Text of the search query (Mandatory)
- search_type: Type of search performed - category, text, parametric, advanced (Mandatory)
- filter_json: JSON data storing all filter parameters (Optional)
- results_count: Number of results returned (Mandatory)
- created_at: Timestamp when search was performed (Mandatory)

10. Image_Tags

- tag_id: Unique identifier for each tag (Mandatory)
- image_id: Reference to the associated image (Mandatory)
- tag: Text value of the tag (Mandatory)

11. listing_category

- listing_category_id: Unique identifier for each category assignment (Mandatory)
- listing_id: Reference to the listing (Mandatory)
- category_id: Reference to the category (Mandatory)
- is_primary: Boolean indicating if this is the primary category for the listing (Mandatory)
- display_order: Integer for ordering when a listing appears in multiple categories (Optional)
- created_at: Timestamp when the assignment was created (Mandatory)
- updated_at: Timestamp when the assignment was last modified (Optional)

Functional Requirements - Prioritized

Priority 1

Unregistered User

1. Unregistered shall register with email and passwords
2. Unregistered Users shall view posts.
3. Unregistered Users shall scroll through listings and view photos and descriptions.
4. Unregistered Users shall view skills and skill-sharing post descriptions.
5. Unregistered Users shall use keywords to search for products.

Registered Users

6. Registered Users shall buy products
7. Registered Users shall sell products
8. Registered Users shall sell services or skills
9. Registered Users shall view posts.
10. Registered Users shall scroll through listings and view photos and descriptions.
11. Registered Users shall view skills and skill-sharing post descriptions.
12. Registered Users shall upload images for listings.
13. Registered User shall describe products for sale.
14. Registered Users shall message sellers.
15. Registered Users shall use keywords to search for products.
16. Registered Buyer shall rate sellers after purchases.
17. Registered Users shall offer tutoring or skill tutorials.

Admin

18. Admin shall approve newly created posts

Priority 2

Registered Users

19. Registered Users shall bundle multiple products into one purchase.
20. Registered Buyer shall decide payment methods.
21. Registered Buyer shall set up meeting points for transactions.
22. Registered Users shall report fraudulent or inappropriate products.
23. Registered Users shall view other registered users and their ratings.

Priority 3

Admin

24. Admin shall track total sales.
25. Admin shall view registered users.
26. Admin shall remove flagged posts.

27. Admin shall remove flagged registered users.

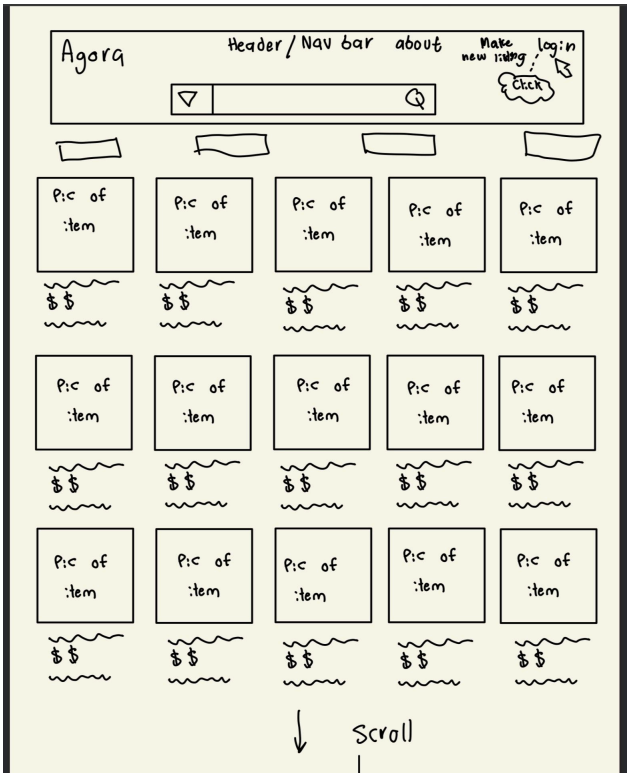
UI Storyboards

1. Seller interacting with Agora to post details about products they are selling

- Actors: Jacob (User), Agora (Software)

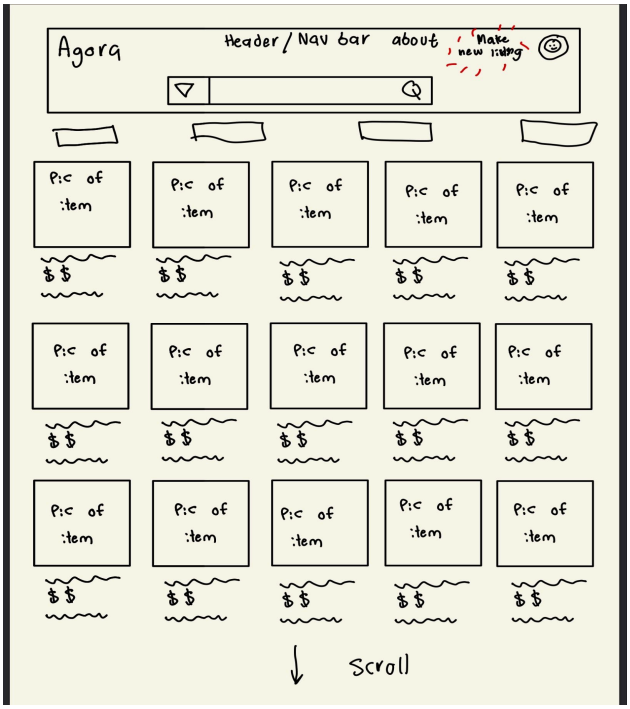
- Assumptions
    - Jacob is tech-savvy but doesn't want to spend too much time managing listings
    - Jacob wants to make some extra cash
    - Jacob is worried about meeting with strangers and getting scammed when selling

Jacob is a third-year business student at SFSU. He has old textbooks that have been collecting dust in his closet and doesn't want to simply throw them away. Jacob wants to sell his old textbooks and he knows other students need his old textbooks. He finds Agora and makes an account so he can start selling. He posts several of his old textbooks. Each post has several pictures and a short description. Jacob notices a stack of old textbooks gathering dust in his closet. Jacob hears about Agora from classmates. Jacob opens Agora's website.

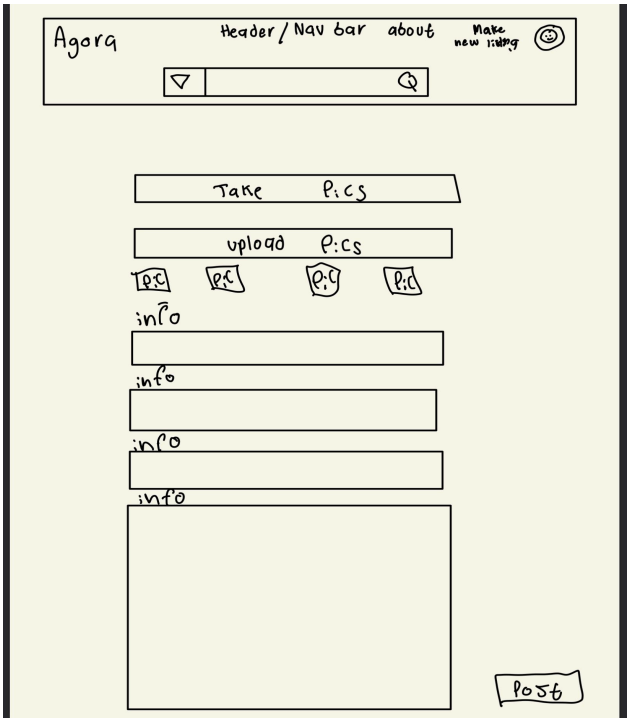


Jacob clicks on the login button to create an account.

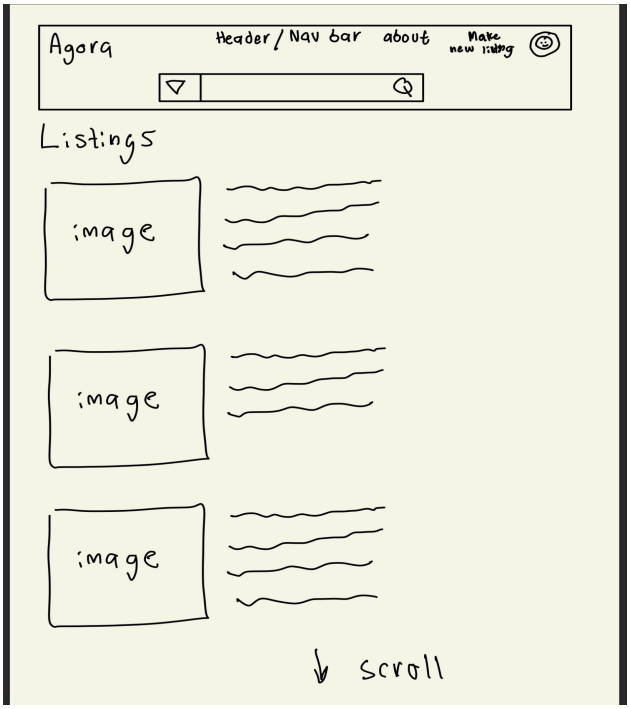
Jacob goes through the process of creating an account on Agora; his email, quickly verifies the account.



Jacob selects " Make New Listing" in Agora.



Once in the Listing page, Jacob starts to upload multiple clear pictures of each textbook from various angles. Jacob has to make sure the images are not too big. Jacob types in all the information needed for potential buyers (condition etc.). He then hits post for each one of his listings.



After Jacob is done listing the textbooks he goes to his listing page on Agora and sees that everything is correct. Jacob feels like Agora offers a simple way to post his items stress free.

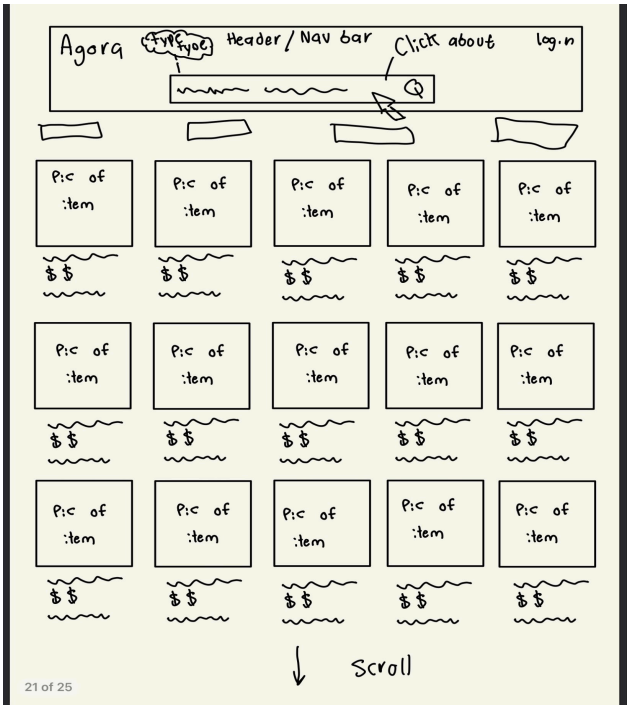
2. Buyer messaging seller about product details and meetup

- Actors: Molly (User), Agora (Software)

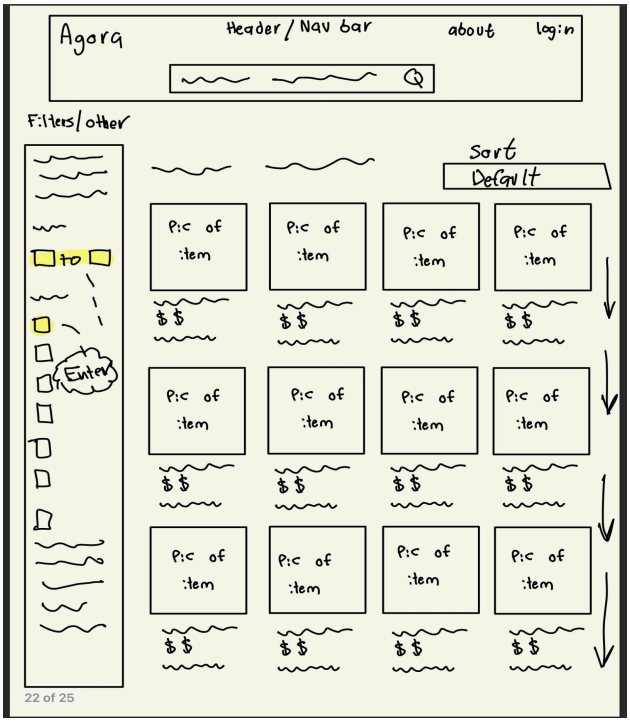
- Assumptions
    - Molly prefers verified sellers and clear product descriptions
    - Molly wants an easy method to contact sellers without needing to share personal details
    - Molly is not tech-savvy but can use simple websites

Molly is a part-time lecturer at SFSU who looks for affordable teaching materials and uses tech. Molly is able to use search filters to find the specific products she wants. However, the item she wants only has one picture, and the post is missing information, such as condition and meetup location. Molly can easily find the message button when she navigates to the post. With messaging, Molly can ask for more pictures and questions about the item from the seller. Their communication is smooth, so they meet on campus and have a successful transaction through Agora.

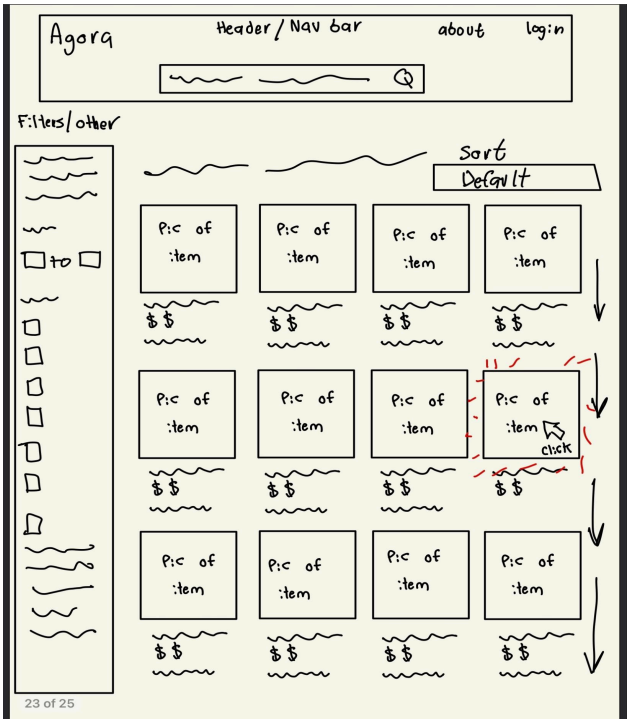
Molly realizes she needs affordable textbooks or teaching materials for her upcoming lectures.



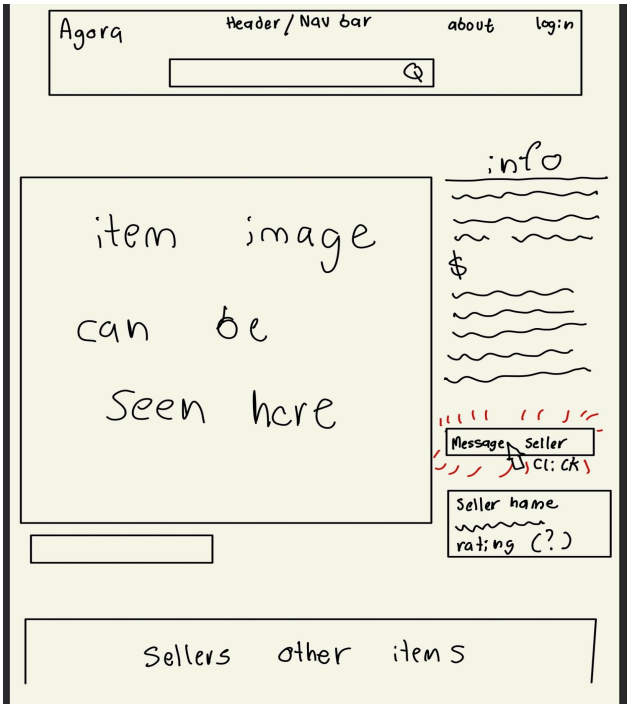
Molly accesses Agora's easy-to-use website and searches.



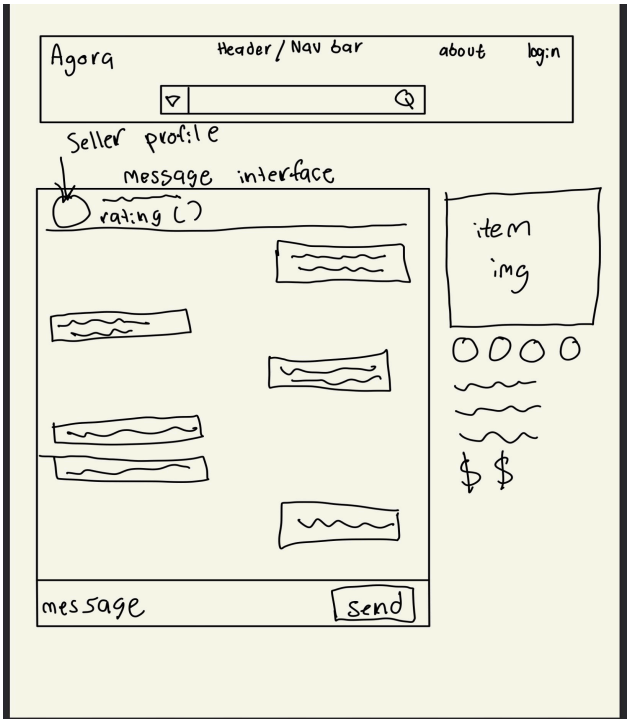
Molly accesses Agora's easy-to-use website and searches for a specific textbook or item using filters (price, location, item category).



Molly clicks on a promising listing.



Molly notices it only has one photo and missing details (condition, meetup location). Molly easily locates the "Message Seller" button on the listing page, clicking it to begin messaging directly through Agora, keeping personal information private.



Molly politely asks for additional details: extra photos of the product, clarification on the condition of the item, and preferred meetup locations. Seller quickly responds by uploading more clear pictures directly into the chat, provides details of item condition ("gently used,

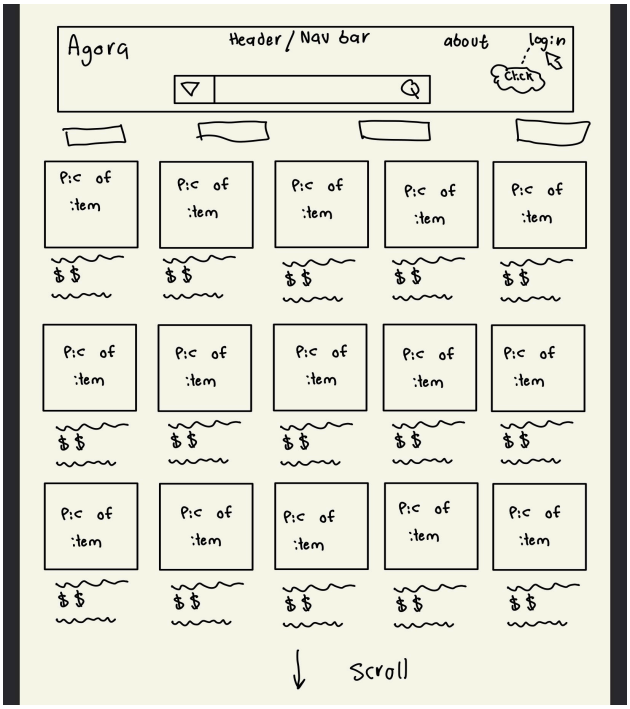
minimal highlights"), and suggests a convenient, public meetup spot (campus library entrance). Molly agrees on the offered details and meetup location, confirms a specific date and time for the transaction.

3. Skill sharing on Agora

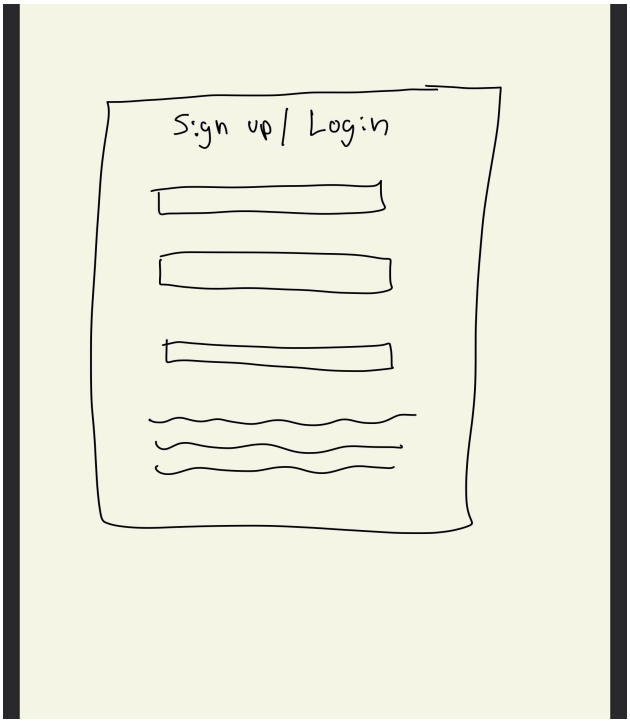
- Maria (Learner/Buyer), Agora (Software)

- Assumptions:
    - Maria is a senior CS student at SFSU
    - Maria (Learner/Buyer), Agora (Software)
    - Maria is tech-savvy and comfortable navigating online platforms.

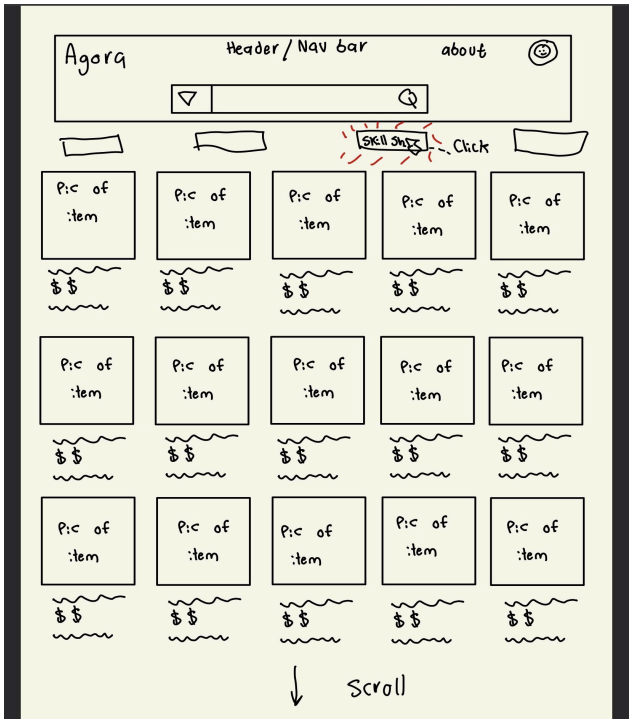
Maria, a senior Computer Science student at SFSU, has recently developed an interest in learning how to knit as a way to relax and take breaks from her studies. Wanting to learn, she decides to use Agora. Maria creates an account on Agora and navigates to the Skill Sharing section. Using Agora's search and filtering, she looks for knitting lessons or people offering one-on-one tutoring in knitting. After browsing several listings, reading descriptions, and reviewing ratings of available tutors, Maria finds a few students who offer knitting lessons. She reaches out to one of them through Agora’s messaging feature to inquire about availability, lesson format, and pricing. Maria feels confident that she will be able to connect with the right person to help her learn knitting.



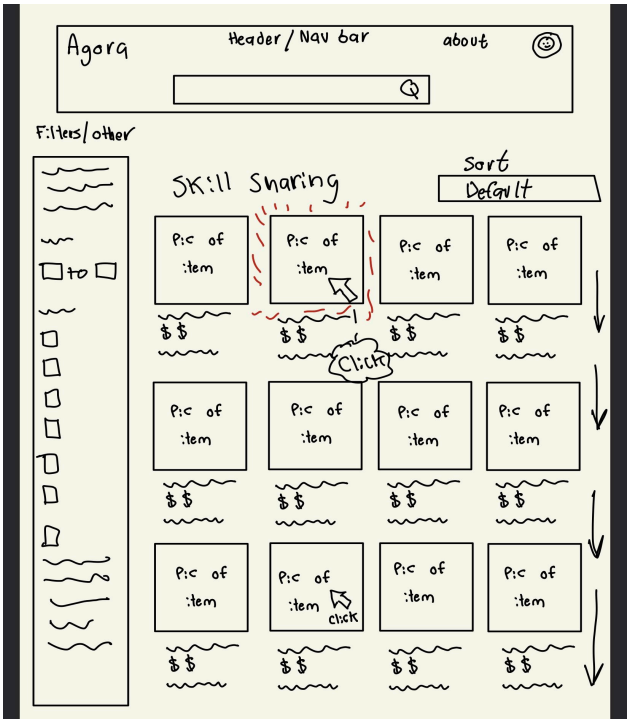
Maria, already comfortable using online platforms, opens the application Agora and looks for the login button to quickly create an account. Once found Maria clicks on it.



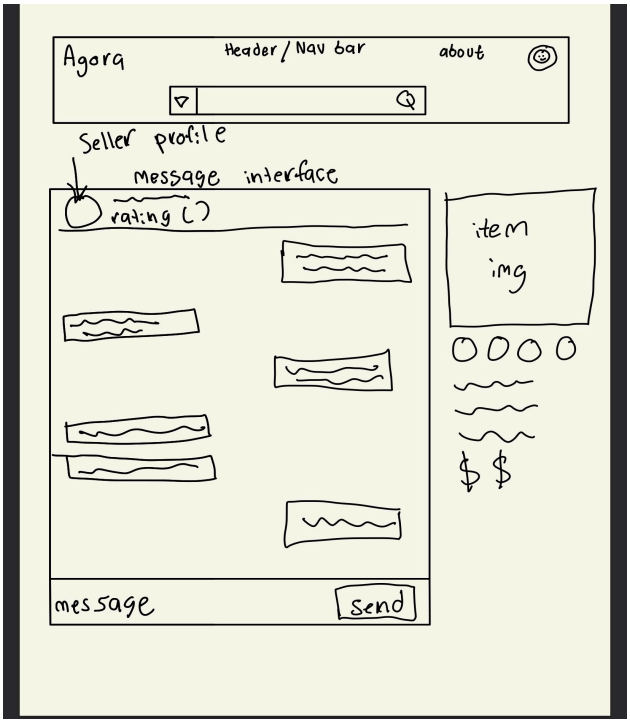
Maria inputs her SFSU information for creating an account. Once done inputting information Maria clicks enter and is taken back to the homepage as a user.



Maria clicks on the clearly-marked “Skill Sharing” section on Agora’s homepage.



Maria searches for “knitting lessons,” using filters. Browsing through detailed listings, Maria identifies a few promising tutors who are also SFSU students, with positive reviews and clear skill descriptions that resonate with her goals. Maria clicks on the one she likes.

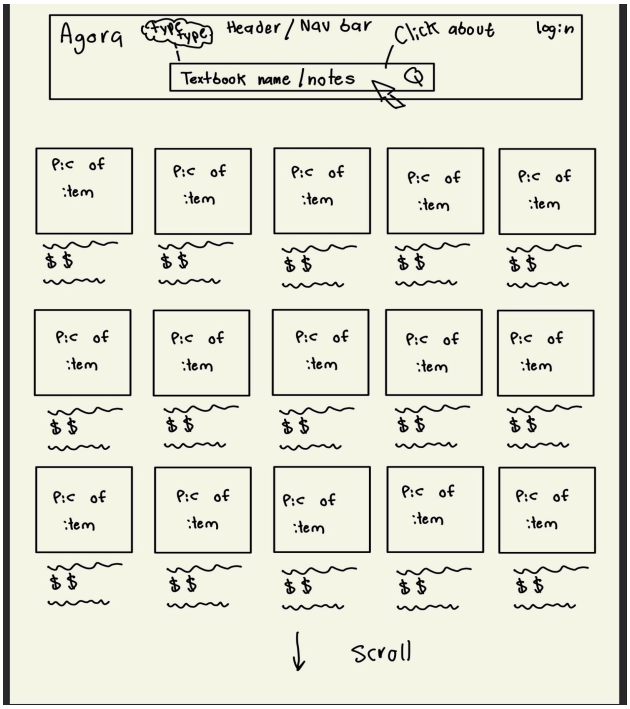


Maria messages one selected tutor, clearly asking about their availability, lesson format, cost per session, and other things. Tutor promptly replies with friendly, clear information sharing availability, pricing, and recommended session format. Maria agrees with the tutor on a convenient date, time, and location through Agora. Maria meets with the tutor, enjoys the experience, learns the basics of knitting, and establishes a positive relationship.

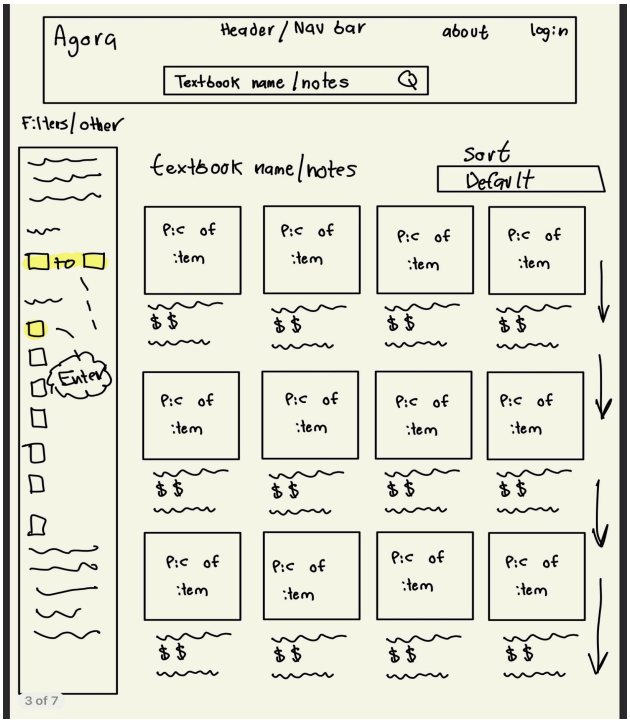
4. Buyer interacting with search features and sellers

- David (Buyer), Agora (Software)
- Assumptions:
    - David is tech-savvy but new to online marketplaces.
    - David prefers local purchases to avoid extra costs like shipping and taxes.
    - David is cautious about legitimacy and safety, verifying sellers before buying.
    - David is looking for affordable second-hand textbooks and notes.
    - David prefers a simple and secure buying process, including clear listings and easy communication.

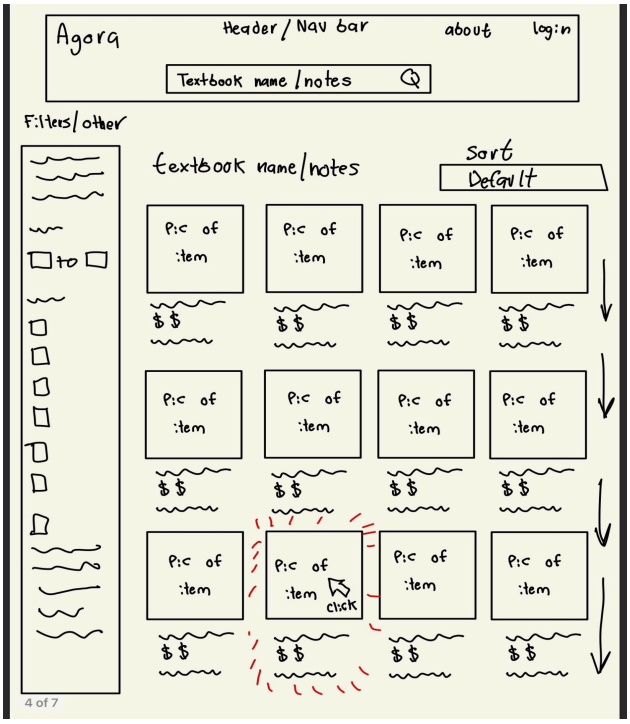
David, a first-year CS student at SFSU, is looking for affordable textbooks and class notes for the upcoming semester. New to online marketplaces, he's cautious about second-hand purchases and prefers local transactions to avoid extra costs. He searches on Agora, using filters to narrow results by price, condition, and location. While browsing, he finds a set of notes for a challenging class and a used textbook at a reasonable price. Wanting to ensure legitimacy, he checks seller ratings, reviews, and listing details before reaching out. Through Agora's messaging system, he asks the seller about the notes' completeness and arranges to meet on campus to inspect the textbook before finalizing the purchase. By using Agora's search, filtering, and communication tools, David feels more confident making a safe and budget-friendly purchase.



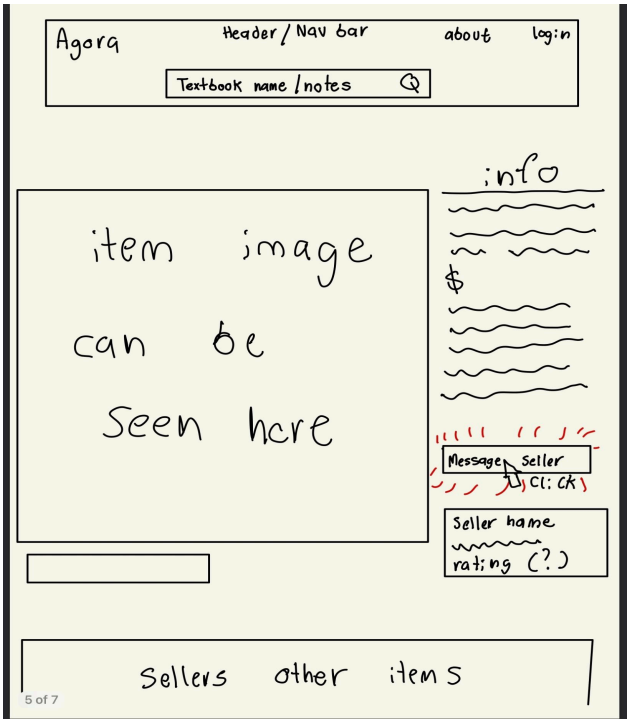
Davic visits the Agora application. He then enters into the search bar the book title/notes and hits the search button. The page refreshes and we enter the next page.



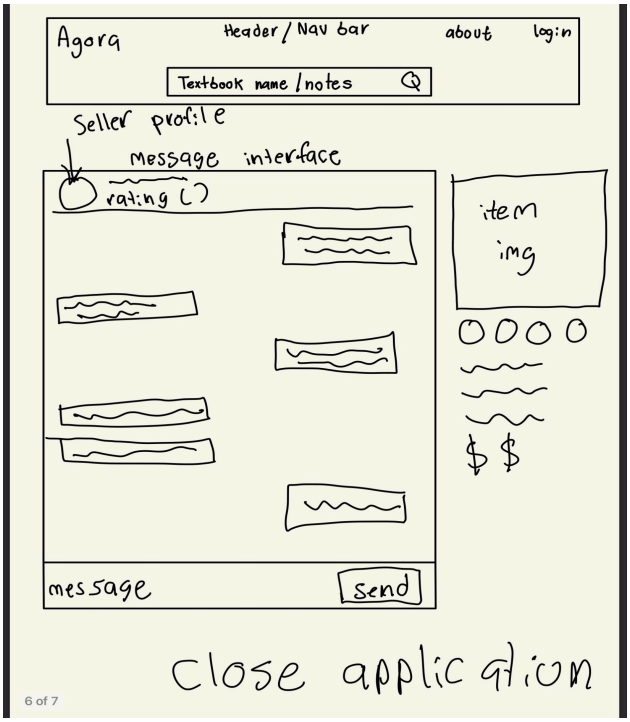
Here David applies the necessary filters that best fit his needs (price, condition, location). After selecting all the filters the page refreshes with new items/products.



With new items/products to look at, David carefully examines the listings. He scrolls down the page checking photos, price, detailed descriptions, seller ratings, and reviews from other students/staff. After a few minutes of scrolling and looking David finds an affordable used textbook and a promising set of class notes for a challenging class. David clicks on the listing and we are taken to the item/products page.



Here David looks at the item/product page and decides to reach out to the seller via Agora's messaging system. David clicks on the message seller button.



David politely asks about the notes' completeness, accuracy, and any marking or highlights. He also asks for confirmation about the textbook's condition. Seller promptly responds with clear answers about the notes. Both David and the seller coordinate a safe and convenient meetup location on the SFSU campus through Agora's messaging system. Here we see David and the seller carefully inspecting the textbook and notes and completing the transaction. By using Agora's search, filtering, and communication tools, David feels more confident making a safe and budget-friendly purchase.

High Level Architecture, Database Organization Summary

DB Organization

1. users: Stores user account information, credentials, and verification status
2. categories: Organizes listings into hierarchical categories for easier navigation
3. listings: Contains core product/service information including pricing and status
4. listing_images: Manages images associated with listings
5. listing_category: Junction table that creates many-to-many relationships between listings and categories, allowing a listing to appear in multiple categories
6. skill_listings: Stores additional information for skill-sharing listings
7. messages: Tracks communication between users about listings
8. reviews: Stores user feedback and ratings after transactions
9. flagged_content: Manages reported listings for administrative review

10. search_logs: Tracks search activity for analytics and improvements
11. image_tags: Associates descriptive tags with listing images for improved search

Media Storage

We have decided to implement a file system-based approach rather than using database BLOBs for Agora's media storage needs.

Our Implementation Approach:

1. Dual-Resolution Storage System:
    - Full-resolution images (limited to a few MB) for detailed product views
    - Thumbnail images (approximately 20KB) for search results and listing previews
    - Database will store relative paths to both versions, not the binary data itself
2. Image Processing:
    - Using Python Pillow (PIL) package as recommended by Professor Souza
    - Automatic thumbnail generation upon image upload
    - Basic image optimization (compression, resizing) to ensure reasonable file sizes
3. Serving Strategy:
    - Images will be served through FastAPI's static file serving capabilities
    - Alternatively, we'll set up Nginx to serve static files directly for better performance
    - Links to images will be used in HTML/templates rather than embedding binary data

Rationale for Choosing File System over BLOBs:

1. Performance: Database queries remain lighter and faster without the overhead of retrieving large binary objects
2. Simplicity: Easier to implement and manage for a student project with limited time
3. Scalability: File storage can be expanded independently from the database if needed
4. Browser Caching: Browsers can cache static files more effectively when served from a file system
5. Development Experience: Easier to debug and work with during development

Security Considerations:

1. Path Sanitization: Implement proper filename sanitization to prevent directory traversal attacks
2. Access Control: Configure web server permissions to prevent unauthorized direct access
3. Unique Filenames: Generate unique filenames to prevent overwriting and predictability

Search/Filter Architecture and Implementation

Agora will implement a robust yet straightforward search functionality using SQL with the '%LIKE%' operator for text-based searching. This approach provides an optimal balance between implementation simplicity and search effectiveness for our application scale.

Search Types and Implementation Complexity

1. Browse by Category

- Description: Navigation-based approach where users select from predefined categories
- Implementation: Simple SQL WHERE clauses against the category table
- Complexity: Easy to implement with standard SQL joins between listing and category tables

2. Free Text Entry Field

- Description: Standard search box allowing keyword entry across multiple fields
- Implementation: SQL '%LIKE%' operators searching across title, description, and keywords
- Complexity: Straightforward implementation with some attention needed for performance optimization

3. Parametric Search

- Description: Form-based search with fields for various attributes like price range, condition
- Implementation: Multiple WHERE clauses with optional parameters
- Complexity: Requires more complex query construction but still achievable with standard SQL

Implementation Strategy

Phase 1: Core Search Functionality (Current Milestone)

For the Agora vertical prototype, we will implement the core search components:

1. A category dropdown menu with at least three options (Electronics, Books, Furniture) plus a default "All Categories" option
2. A free text entry field for keyword search
3. Display of number of items found
4. Results showing title, price, description, and image

Search parameters will remain persistent after execution to facilitate iterative searching. Users can use either component individually or combine them.

Database Terms to be Searched

The following data items will be searched using '%LIKE%':

- listing.title
- listing.description
- listing.search_keywords

Phase 2: Enhanced Search Capabilities (Future Milestone)

In subsequent development phases, we will add:

- Parametric search with filters for price range, condition, and posting date
- Improved result sorting options (newest, price high/low, etc.)
- Saved searches functionality

Database Terms to be Searched

The following data items will be searched using '%LIKE%':

- listing.title
- listing.description
- listing.search_keywords

Database Coding and Organization

For our database implementation:

1. The above fields will be indexed to optimize search performance
2. For category filtering, we'll use standard SQL WHERE clauses against the category table

3. Combined searches will use AND conditions to join text search with category filters

Sample query structures:

Core Search - Category Browsing:

```
SELECT * FROM listing
JOIN listing_category ON listing.listing_id = listing_category.listing_id
JOIN category ON listing_category.category_id = category.category_id
WHERE category.name = 'selected_category';
```

Core Search - Text Search:

```
SELECT * FROM listing
WHERE listing.title LIKE '%search_term%'
  OR listing.description LIKE '%search_term%'
  OR listing.search_keywords LIKE '%search_term%';
```

Core Search - Combined:

```
SELECT * FROM listing
JOIN listing_category ON listing.listing_id = listing_category.listing_id
JOIN category ON listing_category.category_id = category.category_id
WHERE (listing.title LIKE '%search_term%'
  OR listing.description LIKE '%search_term%'
  OR listing.search_keywords LIKE '%search_term%')
AND category.name = 'selected_category';
```

Empty Search Handling (Show All):

```
SELECT * FROM listing

WHERE listing.status = 'available'

ORDER BY listing.created_at DESC

LIMIT 50;
```

When no search parameters are provided, the system will display all available listings to help users browse the complete inventory. This "empty search" functionality serves as the default browsing experience, with results ordered by most recent first and limited to a reasonable number per page to avoid overwhelming users.

Non-Trivial Algorithms

To improve search relevance, we implement a simple ranking algorithm that prioritizes matches in the title over matches in other fields:

```
SELECT listing.*
FROM listing
WHERE (listing.title LIKE '%search_term%'
   OR listing.description LIKE '%search_term%'
   OR listing.search_keywords LIKE '%search_term%')
ORDER BY
   CASE WHEN listing.title LIKE '%search_term%' THEN 1 ELSE 2 END,
   listing.created_at DESC;
```

This ensures that when a user searches for a term, listings with that term in the title appear before listings where the term only appears in the description or keywords.

We are not proposing any alternative search technologies or APIs at this time, as SQL with '%LIKE%' meets our current requirements for simplicity and functionality.

Key Risks

Legal/Content Risks

Risk: There is a potential risk of unintentionally using copyrighted images, videos, or code snippets without proper licensing, especially when sourcing media online or using GenAI tools. Additionally, using third-party APIs or logos without permission may violate terms of service.

Mitigation: We will only use royalty-free or team-generated media, and verify all third-party libraries have permissive licenses. GenAI outputs will be reviewed and rewritten as needed. We will avoid using real brand logos and ensure all APIs used comply with their terms of use.

Skills Risks

Risk: Team members have varying levels of experience with the selected frameworks (FastAPI, Svelte). Some backend team members are not familiar with SQLAlchemy ORM and advanced SQL queries needed for the search functionality.

Mitigation: We hold weekly team meetings where members share their knowledge and expertise on specific frameworks. During these sessions, we address questions. We also need to maintain a shared document with useful resources and tutorials for continuous learning.

Schedule Risks

Risk: The vertical prototype deadline coincides with midterms in other courses, creating a time crunch that may affect our ability to deliver a fully functioning search mechanism on time.

Mitigation: We'll prioritize core search functionality (category filtering and basic text search) to ensure a working prototype by the deadline. Enhanced features will be scheduled for implementation after midterms.

Technical Risks

Risk: Implementing effective search functionality with SQL LIKE queries may face performance issues with larger datasets, especially when searching across multiple fields simultaneously.

Mitigation: We will properly index all searchable fields and implement pagination to limit result sets. For the vertical prototype, we'll use a smaller test dataset to demonstrate functionality while researching optimization techniques for the full implementation.

Project Management

So far, the team has held weekly meetings to discuss progress, assign tasks, and confirm we are meeting the milestone requirements. Each meeting includes status updates for team members, any problems that came up, and follow up items for the next meeting. The meetings have helped us stay on track for the development and planning of milestone 2. As our team develops Agora, we plan on continuing our weekly meetings and increasing their frequency especially during critical phases. Additionally, we are going to be using Trello for a more organized way of administering tasks and setting deadlines.

AI Disclosure

1. ChatGPT(OpenAI):`MEDIUM Helpfulness
    - Assisted with writing documentation and explaining technical concepts
    - Consistent formatting in documentation

Team Lead Checklist

So far all team members are fully engaged and attending team sessions when required: ON TRACK

Team ready and able to use the chosen back and front end frameworks and those who need to learn are working on learning and practicing: ON TRACK

Team reviewed suggested resources before drafting Milestone 2:DONE

Team lead checked Milestone 2 document for quality, completeness, formatting and compliance with instructions before the submission: DONE

Team lead ensured that all team members read the final Milestone 2 document and agree/understand it before submission: DONE

Team shared and discussed experience with GenAI tools among themselves: DONE