SW Engineering CSC648-848 Spring 2025

Project/application title and name: Gator Market

Section 04 Team 1 Milestone 2

Name	Role	Email
Dev Modi	Team Lead, Backend, Frontend, database	dmodi@sfsu.edu
Yash Pachori	Backend Lead with Database	ypachori@sfsu.edu
Kyle Yuen	Frontend Lead	kyuen4@sfsu.edu
Hsueh-Ta Lu	Scrum Master	hlu@sfsu.edu
Daniel	Tech Lead and Github	domstead@sfsu.edu

Revision History

Date	**Notes**
10 March 2025	Initial publication of M1 document
24 March 2025	Revised for Milestone 2 Part 1 Submission

Topics for the milestone

Executive Summary

In an era of digital connectivity and on-demand convenience, San Francisco State University students face a persistent challenge: finding a safe, efficient, and reliable way to buy and sell personal items. Whether it's textbooks, dorm furniture, electronics, or other essentials, students often rely on platforms like Craigslist or Facebook Marketplace. However, these platforms present significant risks, including scams, unverified buyers and sellers, and transactions that require meeting strangers in unsecured locations. To address these issues, we introduce "Gator Market", an exclusive SFSU student marketplace designed to create a secure, seamless, and community-driven buying and selling experience tailored specifically for the university.

At the core of "Gator Market" lies an unwavering commitment to trust and security. By requiring "SFSU email" verification, we ensure that only verified students can participate, fostering a safe and student-only environment where users can confidently list and purchase items. Additionally, our built-in in-app messaging system allows buyers and sellers to communicate securely without sharing personal phone numbers or relying on third-party messaging services, reducing the likelihood of fraud and ensuring peace of mind for both parties.

The "Gator Market" offers AI-powered pricing suggestions, helping students set competitive prices and ensuring fair market value. It's group buying feature allows students to save money on bulk purchases like textbooks and electronics. Unlike generic marketplaces, the platform focuses on SFSU, eliminating irrelevant listings and fostering a tight-knit community. With an intuitive interface, users can easily create listings, filter results, and receive notifications for items matching their interests, providing a seamless experience for all.

Behind this initiative is a team of passionate SFSU students, each bringing expertise in software development, UX design, and business strategy. Being students ourselves, we recognize the daily hurdles involved in campus transactions, and we are committed to creating a marketplace that's built by students, for students. Our varied backgrounds enable us to infuse innovative ideas into the project and continually refine the platform to adapt to the ever-changing needs of our peers. With the right backing, "Gator Market" has the potential to become the go-to hub for SFSU transactions, transforming how students buy and sell while promoting a safer, smarter, and more budget-friendly campus economy.

List of Main Data Items and Entities

1. User

Description: Represents a person interacting with the platform, which can be classified into different user types:

- **Registered User:** A student who has signed up and can buy, sell, and interact with other users.
- **Admin User:** A platform moderator with additional privileges to manage content and enforce rules.
- **Anonymous User:** A visitor who can browse products but must register to purchase or interact with sellers.

Attributes:

- **User ID** (unique identifier)
- **Name**
- **Email** (used for login and notifications)
- **Role** (e.g., buyer, seller, admin, anonymous user)
- **Registration date** (only applicable to registered users)
- **Profile picture** (optional)
- **Contact details** (only visible to admin or during transactions)

2. Product

Description: An item listed for sale by a registered user.

Attributes:

- **Product ID** (unique identifier)
- **Seller ID** (reference to **Registered User**)
- **Name** (title of the product)
- **Description** (detailed information about the product)
- **Category** (e.g., electronics, furniture, clothing, etc.)
- **Price**
- **Condition** (new, used, etc.)
- **Image(s)**

- **Date listed**
- **Status** (available, sold, reserved)

3. Category

Description: Represents the classification of products.

Attributes:

- **Category ID** (unique identifier)
- **Name** (e.g., electronics, furniture, clothing)
- **Parent category** (for subcategories)

4. Review

Description: A rating or feedback given by buyers and sellers after an interaction.

Attributes:

- **Review ID** (unique identifier)
- **Reviewer ID** (reference to **Registered User**)
- **Reviewee ID** (reference to **Registered User**)
- **Product ID** (optional reference)
- **Rating** (1-5 stars)
- **Comment**
- **Date of review**

5. Wishlist

Description: A collection of products a user is interested in purchasing later.

Attributes:

- **Wishlist ID** (unique identifier)
- **User ID** (reference to **Registered User**)
- **Product ID(s)** (reference to **Product**)

- **Date added**

6. Message

Description: A message exchanged between users for product inquiries.

Attributes:

- **Message ID** (unique identifier)
- **Sender ID** (reference to **Registered User**)
- **Receiver ID** (reference to **Registered User**)
- **Message content**
- **Timestamp**
- **Status** (read, unread)

7. Admin Actions

Description: Actions taken by an **Admin User** to manage platform content.

Attributes:

- **Admin ID** (reference to **Admin User**)
- **Action type** (delete user, block product, etc.)
- **Target entity** (User, Product, etc.)
- **Action description**
- **Timestamp**

8. Listing Report

Description: A report submitted by a user regarding a suspicious, inappropriate, or fraudulent listing.

Attributes:

- **Report ID** (unique identifier)
- **Reporter ID** (reference to **Registered User**)
- **Product ID** (reference to **Product**)
- **Reason** (e.g., scam, misleading info, inappropriate content)
- **Additional comments**
- **Date reported**
- **Status** (pending, reviewed, resolved)

Data Glossary (Google Analytics data not included)

ACCOUNT:

Field Name	Description
user_id	Unique identifier for each user, auto-incremented, serves as the primary key.
username	A unique username chosen by the user for identification and login purposes.
password_hash	The user's password, stored as a hashed value for security (e.g., using bcrypt).
first_name	The user's first name, for personalization and identification.
last_name	The user's last name, for personalization and identification.
email	The user's university email (e.g., ending in @sfsu .edu), unique, used for login and verification.
verification_status	Indicates if the user is a verified current SF State student (‘permanent’, 'annual', 'not verified').

Field Name	Description
phone_number	An optional contact phone number for verification purposes. (can be NULL)
profile_picture_url	An optional URL to the user's profile picture (can be NULL).
date_joined	Timestamp recording when the user created their account, defaults to current time.
last_login	Timestamp of the user's most recent login, for tracking activity (can be NULL).
user_role	The user's role on the platform ('user', 'moderator', 'admin'), defaults to 'user'.
account_status	The current status of the user's account ('active', 'inactive/banned', 'deleted'), defaults to 'active'.

POST:

Field Name	Description
post_id	Unique identifier for each post, auto-incremented, serves as the primary key.
user_id	Foreign key referencing the Users table, indicating the user who created the post.
title	The title of the post, limited to 100 characters for brevity and clarity.
description	A detailed description of the item being sold or sought, allowing flexibility in length.
category_id	Foreign key referencing the Categories table, specifying the item's category.

price	The price of the item, with two decimal places (e.g., supports up to 99999999.99).
condition	The condition of the item, restricted to predefined options for consistency.
location	Uses geocode to work with google maps API.
created_at	Timestamp of when the post was created, defaults to the current time for sorting purposes.
status	The current status of the post, managing its lifecycle (defaults to 'active').
approval_status	Indicates if the post has been reviewed by moderators (defaults to 'pending').

Functional Requirements - Prioritized

Priority 1 - Must Do (P1)

Unregistered User

- Unregistered users shall be able to view item listings.
- Unregistered users shall be able to use search and filtering functionalities (category, keyword, price, condition).

Registered User

- Registered users shall be able to register using an SFSU email (must be verified).
- Registered users shall be able to log in, log out, and manage account settings.
- Registered users shall be able to create, edit, and delete product listings.
- Registered users shall be able to upload one or more product images.
- Registered users shall be able to message other users securely within the app.
- Registered users shall be able to view and respond to messages.
- Registered users shall receive email or in-app notifications for new listings based on their interests.
- Registered users shall be able to report listings or users for inappropriate content.
- Registered users shall be able to rate and review buyers/sellers after a transaction.

Priority 2 - Good to Have (P2)

Registered User

- Registered users shall be able to enable group buying features for similar items (e.g., textbooks).
- Registered users shall be able to add multiple images per product listing.
- Registered users shall be able to filter search results by campus building or nearby location.
- Registered users shall be able to archive old listings for personal sales history.

Admin

- Admins shall be able to approve or reject listings before they are published.
- Admins shall be able to review and act on user reports.
- Admins shall be able to manage user accounts (ban/unban/delete).

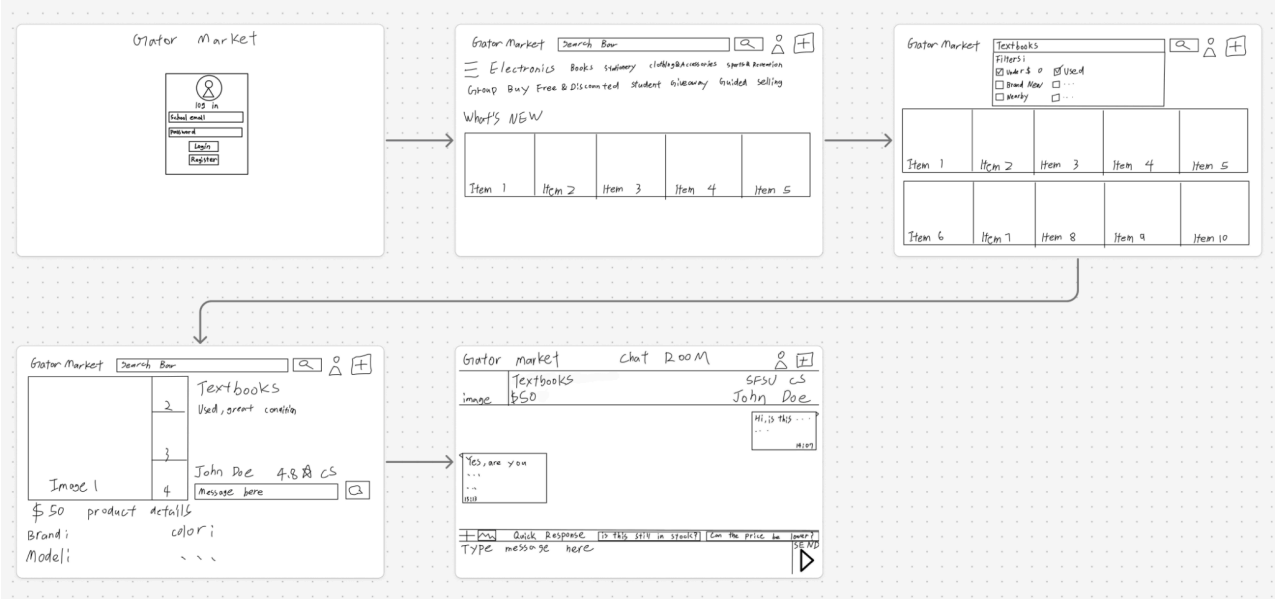
- Admins shall be able to access audit logs for moderation transparency.

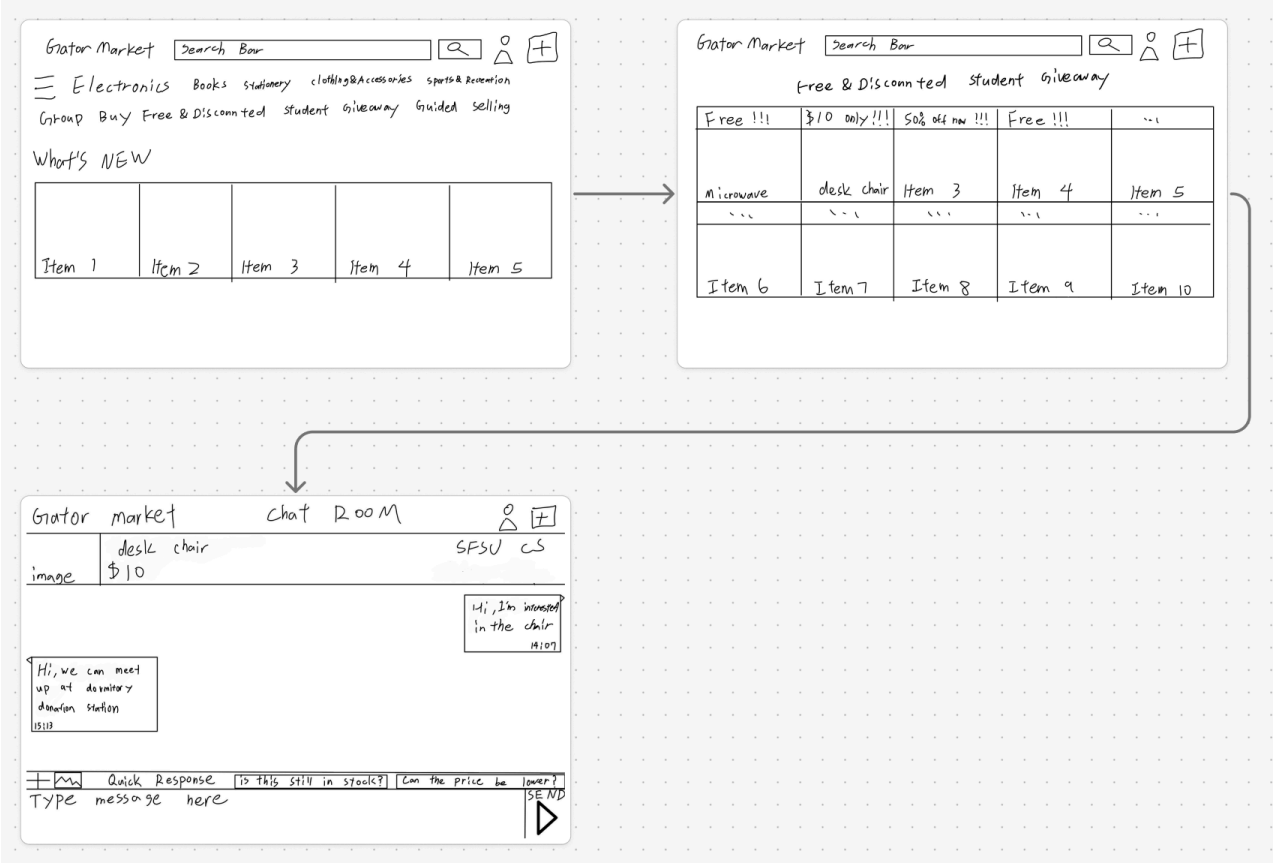
Priority 3 - Future Development (P3)

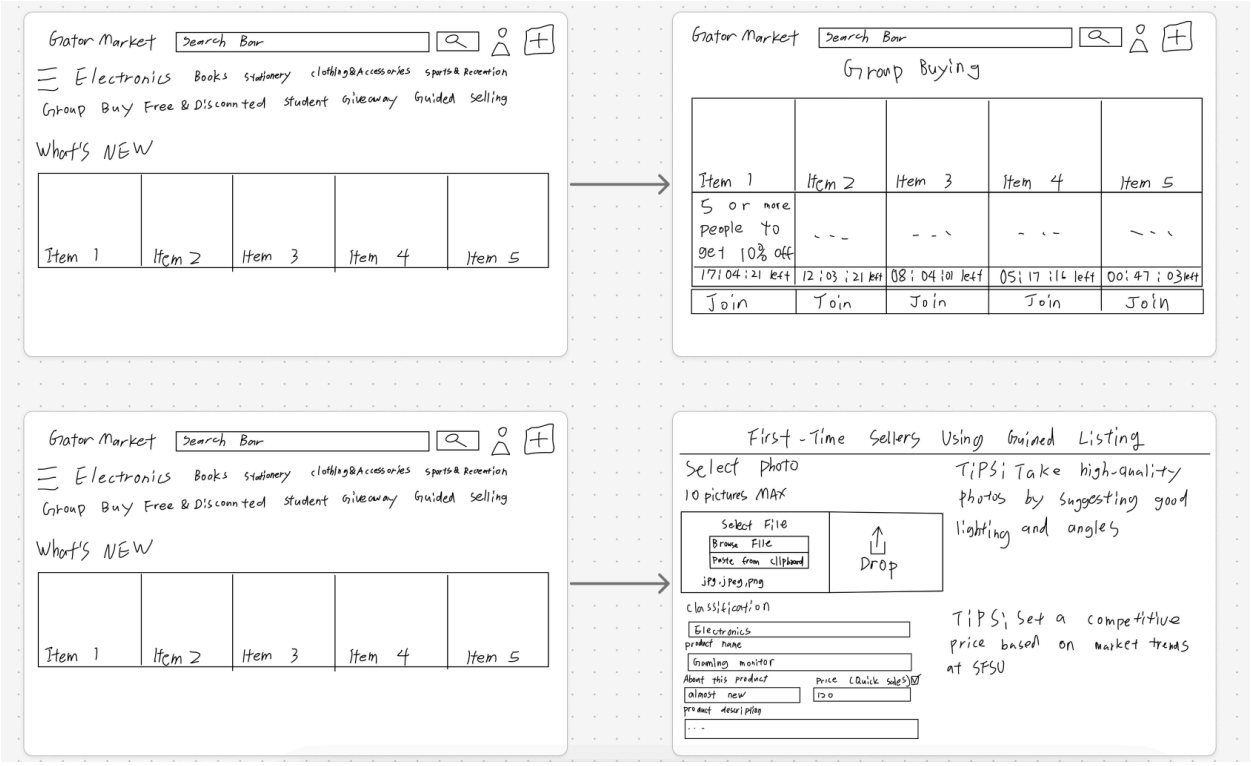
Registered User

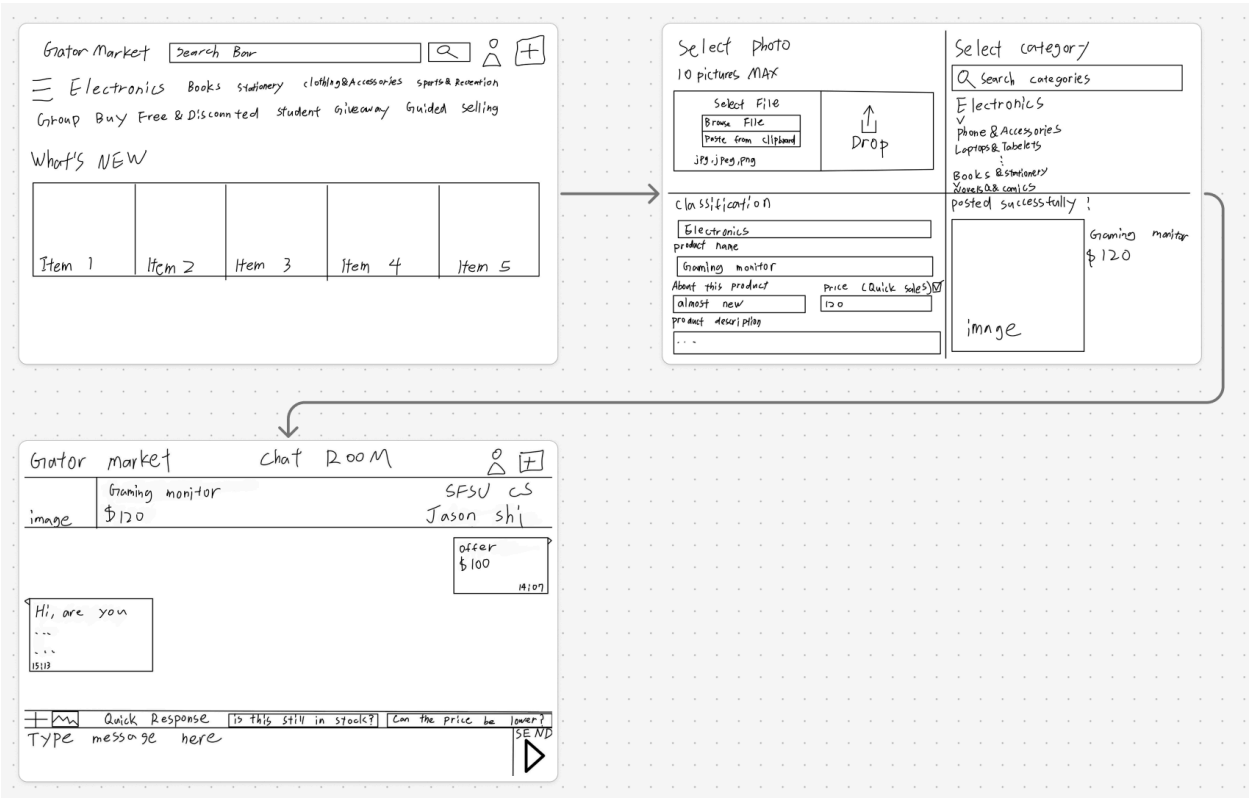
- Registered users shall be able to schedule meet-up time and location with sellers within the chat interface.
- Registered users shall be able to flag a product as "Urgent Sale" or "Free Giveaway."
- Registered users shall be able to bookmark sellers or listings for quick access.
- Registered users shall be able to enable a toggle for "Hide Sold Items" in the product feed.

UI Storyboards for Main Use Cases






Gator Market
Search Bar
Electronics
Books
Stationery
clothing&Accessories
sports& Recreation
Group Buy
Free & Discounted
Student
Giveaway
Guided selling
What's NEW
Item 1
Item 2
Item 3
Item 4
Item 5
Group Buying
5 or more people to get 10% off
17:04:21 left
12:03:21 left
08:04:01 left
05:17:16 left
00:47:03 left
Join
First-Time Sellers Using Guided Listing
Select photo
10 pictures MAX
Select File
Browse File
Paste from clipboard
jpg.jpeg.png
Drop
Classification
Electronics
product name
Gaming monitor
About this product
almost new
Price (Quick sales)
120
product description
TIPS: Take high-quality photos by suggesting good lighting and angles
TIPS: Set a competitive price based on market trends at SFSU


Gator Market
Search Bar
Electronics
Books
Stationery
clothing&Accessories
sports& Recreation
Group Buy
Free & Discounted
Student
Giveaway
Guided selling
What's NEW
Item 1
Item 2
Item 3
Item 4
Item 5
Select photo
10 pictures MAX
Select File
Browse File
Paste from clipboard
jpg.jpeg.png
Drop
Classification
Electronics
product name
Gaming monitor
About this product
almost new
Price (Quick sales)
120
product description
Select category
Search categories
Electronics
Phone & Accessories
Laptops & Tablets
Books & Stationery
Video Games & comics
posted successfully!
Gaming monitor
$120
image
Gator market
Chat ROOM
image
Gaming monitor
$120
SFSU CS
Jason shi
offer
$100
14:07
Hi, are you
15:13
Quick Response
Is this still in stock?
Can the price be lower?
Type message here
SEND

High-Level Architecture and Database Organization

The following provides an overview of the system components and how they interact with each other, including frontend, backend, database, and media storage.

Interaction Flow

1. The user shall interact with the frontend.
2. The frontend shall send API requests to the backend.
3. The backend shall process data and fetch/save data to the MySQL database.
4. If an image is uploaded, the backend shall generate an S3 pre-signed URL for the image upload.
5. The backend shall store the image URL in the MySQL database

Content for High-Level Architecture

1. Architectural Overview

The Product Catalog System shall follow a Model-View-Controller (MVC) architecture to ensure modularity and maintainability. The system shall consist of:

- A frontend UI for users to browse and search products.
- A backend server handling business logic and database operations.
- A relational database storing product, user, and message information.
- Media storage for product images.

This architecture shall allow seamless interaction between users and the product catalog while maintaining data integrity and security.

2. System Components

Frontend (Client-Side)

- Implemented using React and Tailwind CSS.
- Shall provide product listings, search functionality, and UI components for messaging and account management.

Backend (Server-Side)

- Built with Python and Flask.

- Shall handle API requests, process business logic, manage user accounts, and interact with the database.

Database (Data Storage)

- MySQL shall be used for structured data storage.
- Tables shall include:
    - **Users Table:** Stores user account information.
    - **Product Listings Table:** Stores product details.
    - **Categories Table:** Names of the categories of products sold.
    - **Messages Table:** Stores user-to-user messages.

Media Storage

- Product images shall be stored in a cloud-based storage like AWS S3.
- Backend shall generate pre-signed URLs to allow secure image uploads.
- Ensure no payments are required for S3 usage.

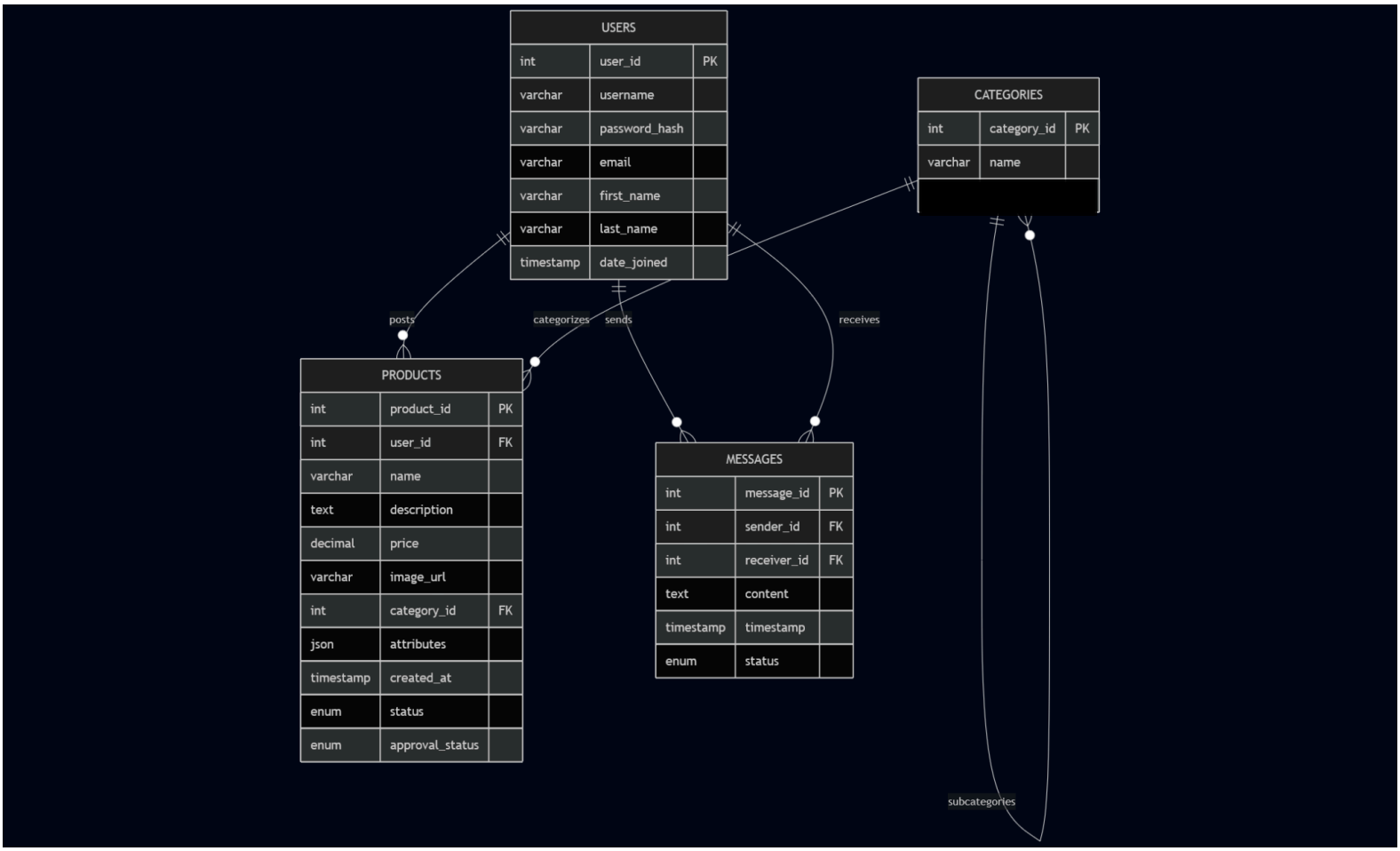
Search and Filtering

- SQL-based **%LIKE** queries shall be used for search.
- Filtering options shall be based on:
    - Categories
    - Price ranges
    - Product condition
    - Keywords

3. Technology Stack

Component	Technology
Frontend	React + Tailwind CSS
Backend	Python + Flask
Database	MySQL
Media Storage	AWS S3

Database Organization (Schema Overview)



The product catalog database shall consist of four core tables: **users, products, categories, and messages.**

All product-related data shall now be stored in a **single unified `products` table** with a `category_id` to support scalability and avoid redundancy.

Messages between users (buyers and sellers) shall be stored in a dedicated `messages` table.

In our initial design, we had separate tables for each product category — computers, phones, tablets, books, and clothes. While this approach organizes data by category, it introduces

significant redundancy since each table shares many of the same fields (e.g., name, description, price, image_url). This structure also makes querying across all products unnecessarily complex and inefficient.

To address this, we have merged all product categories into a single, unified products table. This redesign improves database normalization, simplifies query writing, and enhances scalability for adding new categories in the future.

The new products table includes all shared fields as well as optional fields that are specific to certain categories (e.g., author for books, size and color for clothes). Each product is associated with a category_id that references a centralized categories table. Additionally, each product is linked to a user_id that references the users table to identify the seller.

Products Table

Field	Type	Description
product_id	INT, PK, Auto Increment	Unique product ID
user_id	INT, FK to users	Seller (User)
name	VARCHAR(100)	Product name
description	TEXT	Product description
price	DECIMAL(10,2)	Product price
condition	ENUM('new', 'used', 'refurbished')	Item condition
image_url	VARCHAR(255)	Link to product image

category_id	INT, FK to categories	Product category
created_at	TIMESTAMP	Listing date
status	ENUM('active', 'sold', 'deleted')	Product listing status
approval_status	ENUM('pending', 'approved', 'rejected')	Moderation status

Categories Table

Field	Type	Description
category_id	INT, PK, Auto Increment	Unique category ID
name	VARCHAR(100)	Category name (e.g., electronics, books)

Users Table

Field	Type	Description
user_id	INT, PK, Auto Increment	Unique user ID

username	VARCHAR(50)	Unique username for login
password_hash	VARCHAR(255)	Hashed password
email	VARCHAR(100)	Unique SFSU email
first_name	VARCHAR(50)	First name
last_name	VARCHAR(50)	Last name
verification_status	ENUM('permanent', 'annual', 'not verified')	SFSU verification type
phone_number	VARCHAR(15)	Optional contact number
profile_picture_url	VARCHAR(255)	Optional profile picture URL
date_joined	TIMESTAMP DEFAULT CURRENT_TIMESTAMP	Registration date
last_login	TIMESTAMP NULL	Most recent login timestamp
user_role	ENUM('user', 'moderator', 'admin') DEFAULT 'user'	User privilege level

account_status	ENUM('active', 'inactive/banned', 'deleted') DEFAULT 'active'	Account state

Messages Table

Field	Type	Description
message_id	INT, PK, Auto Increment	Unique message ID
sender_id	INT, FK to users	Sender's user ID
receiver_id	INT, FK to users	Receiver's user ID
product_id	INT, FK to products	Associated product (optional)
message_text	TEXT	Message content
sent_at	TIMESTAMP	Timestamp when the message was sent

Relationships

One-to-One:

- Each user has a unique verification status and profile.

One-to-Many:

- One user can post multiple products.
- One user can send and receive multiple messages.
- One category can have multiple products.

Many-to-Many:

- Users and messages (direct communication between any two users).
- Future support for user roles and wishlists.

5. Data Flow and Interaction

- User Request: User accesses the platform to browse or post products.
- API Call to Backend: The frontend sends API requests to fetch, post, or update product information.
- Backend Processes Request: The backend handles business logic and interacts with the database.
- Database Query Execution: The database returns product, user, or message data as requested.
- Response to Frontend: The backend returns the data as a JSON response.
- Frontend Displays Data: The frontend dynamically displays listings, messages, or user information.

6. Deployment Strategy

- Frontend: Hosted on AWS S3/CloudFront.

- Backend: Deployed on AWS EC2.
- Database: Hosted on MySQL service (AWS RDS or equivalent).
- Media Storage: Product images stored in AWS S3 bucket.
  → Backend shall generate pre-signed URLs to allow secure and free image uploads. No payment required for storage access.

[User] → [React Frontend] → [Node.js Backend]

↓

[AWS S3 - Store Images]

↓

[MySQL - Save image URL in products table]

Product images uploaded through the frontend are sent to AWS S3 using a backend-generated pre-signed URL. Once uploaded, the backend stores the S3 URL in the product's image_url field in the database. This enables secure and efficient image access.

Identification of Key Risks

1. Skills Risk

Issue: The team consists of members with different skill sets and levels. This may include the technical aspects, such as the knowledge of the technology stack used for the project, as well as soft skills such as communication and coordination.

Solution: Conducted weekly knowledge-sharing sessions and assigned small tasks for skill-building.

2. Schedule Risk

Issue: Overlapping deadlines with other coursework.

Solution: Set internal milestone deadlines 3-4 days ahead of official submission dates.

Issue: The team has scheduled team meetings during and between class times. It may still be insufficient to establish a robust understanding for all team members. Even then, it is not guaranteed that all members can attend all meetings, as everyone has their own life.

Solution: The team can reduce it by prioritizing necessary tasks, planning ahead for contingencies, creating checkpoints to assess progression, and dynamically assigning members based on the current state.

3. Technical Risk

Issue: Deployment issues on AWS due to limited experience with Docker.

Solution: Backend team will set up a staging environment for early testing.

4. Teamwork Risk

Issue: Coordination challenges between frontend and backend teams.

Solution: Implement weekly stand-ups with clear task assignments. The team can organize the individual tasks chronologically to reduce dependency and distribute them according to the team member's skill and workload level.

5. Legal/Content Risk

Issue: Ensuring product images are owned by users or use copyright free images.

Issue: The team must be aware of copyright infringement, licensing requirements, intellectual property, privacy, data protection, or any other regulations.

Issue: The team shall ensure that the project does not contain harmful or inappropriate content within its scope of usage.

Solution: The team can ensure the project complies with laws and regulations, implements necessary standards, and reduces liability.

Project Management Strategy

1. **Task Management:**

- Trello is used as our central project management board. Each task card includes:
    - Clear descriptions and subtasks
    - Assigned members
    - Due dates
    - Labels for task categories (frontend, backend, documentation, etc.)
- Tasks are tracked through "To Do," "In Progress," and "Completed" columns.
- The Scrum Master and Team Lead collaboratively update and review progress on Trello before every meeting.
- We have separate boards for Milestones, Bug Tracking, and Feature Requests to avoid clutter and improve focus.

2. Communication & Collaboration:

- Daily communication is maintained via Discord, where we maintain organized channels for specific workstreams like #frontend, #backend, #m1, #m2-part1 and #m2-part2.
- Google Docs is used for collaborative drafting and notes during team sessions.

3. Workflow and Meeting Cadence:

- Weekly sprints run from Wednesday to the following Tuesday. Sprint planning is conducted during Wednesday meetings.
- On Wednesdays, tasks are assigned or reprioritized based on sprint progress.
- By Friday, team members are expected to reach mid-sprint deliverables, verified via Trello status updates.
- Weekend work is self-paced and documented on Trello.
- Monday/Tuesday is reserved for internal testing, documentation, and refinement.

4. Documentation & Version Control:

- Google Docs is used for all milestone planning, ensuring centralized collaborative editing.
- GitHub is our version control hub. Team members create branches per feature/task, and Pull Requests are reviewed by Tech Lead or assigned reviewer.

5. Milestone Tracking and Quality Assurance:

- Each milestone is broken into mini-goals, with Trello labels for better tracking.

- The Scrum Master checks on the status of major goals twice per sprint.
- Before each milestone deadline, the team conducts a QA pass to ensure that documentation, database, and frontend/backend functionality are in sync.

6. Conflict Resolution & Syncing Teams:

- When conflicts or confusion arise, a dedicated zoom call room is used for screen-sharing and real-time collaboration.
- If needed, temporary working groups are formed to resolve integration or logic issues.

7. Learning & Upskilling Support:

- We regularly encourage team members to explore new tools (e.g., Docker, Tailwind) with small guided tasks.
- Pair programming is used for difficult or unfamiliar components.
- Team members share useful resources and tutorials on Discord.

To sum it up, our team used Trello to manage tasks. Everyone was previously split into front-end and back-end groups based on their familiarity with the respective technologies. This made task assignment easier to break up for Trello. Every week after every meeting respective tasks would be assigned on Trello and everyone would be given proper deadlines. Every meeting (Wednesday and Friday) we set a goal to get work done by the weekend. On Friday, we check with each other (previously on github) whether our tasks have been completed. Now we can track them in Trello. On Wednesdays, we talk about how our sprint is going and make sure we are all on track. We are in constant communication on Discord to resolve any issues between sections of work. If there are any conflicts, the concerned members will join a call room and talk about their issues as well as sharing screens to better visualize and explain the issues.

Use of GenAI Tools in Milestone 2

GenAI Tools Used

For Milestone 2, our team utilized various Generative AI tools to streamline different aspects of the project. The primary tool used was:

ChatGPT-4.0 – Used for brainstorming, refining Database Organisation, structuring database schema, Risk Identification planning, and code prototyping.

Tasks Where ChatGPT Was Used and Effectiveness Ratings

Task	Rating	Description of Use
Expanding Functional Priority Requirements	HIGH	AI provided structured requirements with user role breakdowns. We refined them for accuracy.
Database Schema Optimization	HIGH	Suggested efficient table structures and foreign key relationships, which we validated against project needs.
UI Ideation	MEDIUM	AI-generated suggestions, but made changes to customize and fit project requirements.
Risk Identification & Mitigation	MEDIUM	Provided general risk factors, which were adapted for our specific challenges.
Code Boilerplate for API Endpoints	HIGH	Generated Flask API templates, reducing development time.
Search & Filter Logic Recommendations	MEDIUM	Suggested SQL %LIKE% queries and indexing strategies, which we refined.

Key Examples and Prompts Used

Functional Priority Detailing Prompt

Prompt: "List detailed functional Priority requirements for a university marketplace, categorized by user roles. Include key actions for registered users, admins, and guest users."

AI Response(Edited):

- **Registered Users:** Can create/edit/delete listings, contact sellers via in-app messaging, verify SFSU email, receive notifications for matching listings.
- **Admins:** Can moderate content, approve/reject listings, manage user roles and permissions.
- **Guest Users:** Can browse listings but cannot contact sellers without registration.

Database Schema Prompt

Prompt: "Suggest a relational database schema for an online student marketplace with users, product listings, messaging, and reviews. Include foreign key relationships."

AI Response (Edited):

- **Tables Suggested:** Users, Products, Messages, Reviews, Categories, Admin_Actions.
- **Relationships:** One-to-many (Users → Products), Many-to-Many (Users ↔ Messages).

Search & Filter Mechanism Prompt

Prompt:"Suggest an efficient search and filtering strategy for a student marketplace, considering SQL and indexing best practices."

AI Response(Edited):

- Use SQL %LIKE% queries for flexible keyword searching.
- Implement database indexing on frequently searched fields (product name, category).
- Allow multi-filtering (e.g., price range, condition, category).

Final Thoughts on AI Use

Overall, GenAI tools were highly beneficial in structuring content, brainstorming ideas, and speeding up certain development tasks. However, human oversight was necessary to:

- Refine functional requirements for precision.
- Adjust database schema based on actual project constraints.

- Customize UI layouts to align with our user experience goals.
- Validate search and filtering logic for efficiency.

For future milestones, we plan to leverage AI more for:

- Code debugging & optimization
- Further UI/UX enhancements

Team Lead Checklist

So far, all team members are fully engaged and attending team sessions when required – Done

Team found a time slot to meet outside of the class – Done

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 – Done

Team reviewed class slides on requirements and use cases before drafting Milestone 1 – Done

Team reviewed non-functional requirements from “How to start...” document and developed Milestone 1 consistently – Done

Team lead checked Milestone 1 document for quality, completeness, formatting and compliance with instructions before the submission – Done

Team lead ensured that all team members read the final M1 and agree/understand it before submission – Done

Team shared and discussed experience with GenAI tools among themselves – Done

GitHub organized as discussed in class (e.g. master branch, development branch, folder for milestone documents, etc.) – Done