

Project Title

Campus Marketplace System: A Secure Second-Hand Trading Platform for University Students

1. Problem Statement

University students frequently buy and sell second-hand items such as textbooks, small furniture, and electronics. Most transactions are currently handled through fragmented channels. These methods make it difficult to search for items efficiently, compare prices, or build trust between buyers and sellers.

There is no campus-specific platform that supports an end-to-end trading workflow (listing, searching, communication, reservation, and completion) with basic security and monitoring. As a result, students may waste time browsing many messages, miss good deals, or feel uncertain about the reliability of the other party. A dedicated campus marketplace can make second-hand trading safer, more organized, and more convenient for students.

2. Project Objectives and Goals

The goal of this project is to build a Java Swing–based Campus Marketplace System that allows students to buy and sell used items inside the university ecosystem.

Core Objectives

- Provide a campus-only platform for posting, browsing, and managing second-hand item listings.
- Support secure in-system communication between buyers and sellers, instead of relying only on external tools.
- Allow administrators to review suspicious listings and manage user accounts to improve safety.

Advanced Objectives

- Implement a complete order workflow: buyer request → pending seller decision → reserved → completed or cancelled.
- Provide advanced search and filtering by category, price range, condition, and seller rating, with sorting options.

- Implement a simple content-based recommendation module that suggests similar items based on category and view/favorite history.
 - Provide data for administrators for analysis
-

3. User Roles and Use Cases

The system defines three main user roles.

Role 1: Seller

Use Case 1 – Post Item

The seller creates a new listing by entering title, description, price, category, condition, and uploading one or more photos.

Use Case 2 – Manage Listings

The seller views a list of their own items, and can edit details, change price, mark an item as reserved/sold, or delete the listing.

Use Case 3 – Respond to Inquiries

The seller receives messages and order requests from buyers and can reply in the in-app message panel.

Use Case 4 – Manage Orders

For each item, the seller can see order requests, accept or reject them, and confirm when a transaction is completed offline.

Use Case 5 – View Listing Statistics (Advanced)

The seller can see simple statistics such as views, favorites, and completed orders for each listing to help decide how to adjust price or description.

Use Case 6 – Managing Personal Reputation (Advanced)

The seller can view the list of Reviews that buyers have given you.

Role 2: Buyer

Use Case 1 – Browse & Search Items

The buyer browses all active listings and uses filters (category, price range, condition) and sorting (newest, lowest price, most popular).

Use Case 2 – View Details & Contact Seller

The buyer opens a detailed item page with description, photos, and seller rating, and sends inquiries to the seller via in-app messages.

Use Case 3 – Favorite Items

The buyer can bookmark interesting listings into a “Favorites” list to check later.

Use Case 4 – Place Order / Reservation

The buyer sends an order request for an item. The system records the request with status “REQUESTED” and notifies the seller.

Use Case 5 – Manage Orders

The buyer views all their orders with status (Requested, Pending Seller, Reserved, Completed, Cancelled) and can cancel unconfirmed orders or confirm completion.

Use Case 6 – Rate Seller and Item (Advanced)

After a completed trade, the buyer can rate the seller and leave a short review. Ratings contribute to the seller’s reputation score.

Role 3: Administrator

Use Case 1 – Review New Listings

The administrator checks newly created listings, approves normal items, and rejects or hides inappropriate content.

Use Case 2 – Manage User Accounts

The administrator can view user profiles and deactivate accounts that have repeated violations.

Use Case 3 – Handle Complaints

The administrator reviews complaints submitted by users, investigates the related items or accounts, and takes action (warning, banning, removing content).

Use Case 4 – Monitor Transactions

The administrator monitors open orders and can close long-inactive orders or investigate suspicious trading patterns.

Use Case 5 – View Analytics Dashboard (Advanced)

The administrator can access a statistics page that displays key system information, such as user activity, popular product categories, and transaction trends. This helps the administrator monitor platform health and make informed decisions about system management.

4. System Architecture and Design Approach

High-Level Architecture

We're organizing the code into three main layers to keep everything clean and organized:

1. Presentation Layer (UI - What users see)

- We'll use Java Swing with CardLayout to switch between different screens like login, product browsing, seller management, and the admin panel
- Each user role (Buyer, Seller, Admin) will only see the features they're allowed to use
- Main screens include: login/registration, marketplace view, my products page, order tracking, messaging, and admin controls

2. Business Layer (Service - The logic behind the scenes)

- UserService – handles user registration, login, and managing different user roles
- ListingService – lets sellers post products, edit them, and lets buyers search and filter items
- OrderService – manages the whole order process from request to completion, using the WorkRequest structure from class
- MessageService – enables buyers and sellers to communicate within the app
- RecommendationService – suggests similar products based on what users have looked at or favorited
- StatisticsService – aggregates system data and generates reports for administrators

3. Persistence Layer (DAO - How we save data)

- We'll create Data Access Objects (DAOs) like UserDao, ProductDao, OrderDao, and MessageDao to handle all file reading/writing
- We're using CSV files to store data because they're simple, easy to debug, and we can even open them in Excel to check if everything is working correctly
- Each entity will have its own CSV file (users.csv, products.csv, orders.csv, messages.csv, etc.)
- The DAO design means we can easily switch to JSON or a real database later without rewriting all our code

Our Design Approach

- We're following the MVC (Model-View-Controller) pattern we learned in class, which separates the interface, the data, and the business logic
- We want to keep our service methods simple and testable, so we can check if the core features work correctly even before finishing the UI
- Using CSV files keeps things straightforward for our first implementation - we can literally see our data in a spreadsheet while developing

- Order status updates and confirmation messages.
 - Admin dashboard containing summary tables and simple charts (e.g., listings per category, monthly transactions).
 - Persistent data files (CSV format) storing users, products, orders, and messages. (Updated to CSV based on your request.)
-

6. Implementation Plan and Timeline

Week 1 — System Design, Core Development, and Essential Features

Requirements & High-Level Architecture

- Finalize project scope, user roles (Buyer, Seller, Admin), and primary use cases. Create UML diagrams, including class diagram. Set up the base project structure.

Core Domain Model & Persistence

- Implement all core entity classes: User, Buyer, Seller, Admin, Product, Order, Message, Category, Review.
- Develop the DAO / PersistenceManager for data loading and saving using local CSV files.

Fundamental UI & Basic User Functions

- Build login and registration pages with role-based UI navigation.
 - Implement seller-side product management: create, edit, delete, update status (available/sold).
 - Implement buyer-side product browsing, basic search, and item detail view.
-

Week 2 — Advanced Features, Admin Tools, Testing, and Finalization

Order Workflow & Messaging

- Implement the full order lifecycle: requested → pending → reserved → completed / cancelled.
- Integrate order tasks with organizations and work queues (if required by course framework).
- Implement in-app messaging between buyers and sellers, linked to specific listings or orders.

Advanced Buyer Features & Admin Panel

- Add advanced filtering, sorting, and favorites list for buyers.
- Implement the rating and review system and compute seller reputation scores.
- Build the admin console: product moderation, user management, and complaint handling.
- Add a simple analytic.

Testing, Optimization & Deliverables

- Perform integration testing for all modules; fix major issues.
- Improve UI quality: input validation, error dialogs, clearer layouts.
Prepare demo data, final slides, user guide, and complete code documentation.

7. Assumptions and Limitations

- The system is restricted to campus users only (e.g., registration with a university email).
- The project does not implement real online payment. All money transactions happen offline; users only record the agreement in the system.
- Images are stored as local file paths rather than in a full image server.
- Data persistence is implemented using files (CSV) instead of a production-level database, due to course scope and time limits.
- The recommendation module uses simple heuristic rules based on categories and user interactions; it is not a full machine learning system.
- The application is a desktop Java Swing application and does not include mobile or web interfaces in this version.