Project Title: Campus Marketing Application

Project Description: The goal of Campus Marketplace, a student-only, mobile-first platform, is to establish a secure, practical, and community-focused area where college students may purchase, sell, and peruse personal listings and used goods within their campus network. Campus Marketplace provides everything you need, whether you're looking for reasonably priced textbooks, an extra phone charger, or someone selling furnishings for your dorm room before you graduate.

The software, which was developed with security, ease of use, and student needs in mind, supports student-to-student transactions, encourages sustainable living, and helps cut down on waste. In addition to fostering a flourishing campus local economy, it encourages students to recycle unwanted goods, help their peers, and conserve money.

Configuration

All versions must be updated to the latest stable release.

- Android Studio: Meerkat 2025.x.x
- Java Version: 8.x ~ 17.x
- Minimum SDK: API 24 (Android 7.0 – Nougat)
- Target SDK: API 34+ (Android 14.0 and above) – Future Proofing

Basic General Flow

User log in -> Search items -> Select item -> Add item -> Purchase item -> Seller checks the order -> Deliver to buyer -> Buyer tracks the order -> Buyer pays for order -> Update item status

Requirements

1. Login System

- Roles:
  - Buyers and Sellers (users can act as both buyer and seller).
  - Separate roles for admins to manage users, transactions, and system settings.
- Authentication:

    - Users must register using their school email for verification, ensuring exclusivity to students.
    - Email verification to activate accounts.
    - Password hashing (e.g., bcrypt or SHA) to securely store passwords.

    - Users can manually log out.

2. Database Design

- Database Choices:
    - SQLite for local storage (for offline use).
    - Firebase/Firestore or MySQL for remote storage and scalability.

3. Logging

- Log all system activities, such as:
    - Login/logout attempts (successful and failed).
    - Product postings, edits, or deletions.
    - Purchase attempts or completions.
    - Errors, such as failed logins or invalid inputs.
- Logs can be exported to a text file or viewed in a console for debugging and demo purposes.

5. Concurrency / Simultaneous Actions

- Multiple Users Browsing Simultaneously:
    - Uses WebSocket or polling to show real-time updates when users are browsing products.
- Sellers Uploading While Buyers Browse:
    - Ensures that sellers can upload products without interrupting the buyers' browsing experience.
- Frontend Responsiveness:

    - The UI remains responsive even when product lists are updating or being filtered.

Entities

1. Customer

- Entity that browses, buys, and sells second-hand items.
- Each customer can post multiple products and can purchase multiple items.
- Customers are students with verified school email addresses.
- Can register on their own via the app.

2. Seller

- A user who lists products for sale on the platform.
- Can also be a buyer (dual-role possible).
- Can post, edit, and delete their own product listings.
- Responsible for marking products as sold or unavailable.

3. Buyer

- A user who browses and purchases available products.
- Can view product details, contact sellers, and finalize purchases.
- Can favorite items for later tracking or price drop alerts.

4. Administrator

- Manages overall app operations and user roles.
- Handles moderation tasks like removing inappropriate listings or banning users.
- Can view system logs, oversee transactions, and manage user disputes.
- Has access to analytics and app performance monitoring tools.

System Architecture Overview

Layer / Component	Technologies / Tools	Description / Responsibilities
Frontend UI	Android XML Layout, Java View Layer	Handles all visual elements: layouts, widgets, product listing, user input forms, navigation.
App Logic	Java 17, ViewModels	Processes user actions, handles validation, UI state, and links UI to service layer.
Service Layer	Custom Java Classes (e.g., `AuthService`, `OrderService`)	Manages business logic like login handling, order processing, and purchase flow.
Data Layer	SQLite (local), Firebase / Firestore / MySQL (remote), JSON	Stores and syncs structured data such as user info, product listings, transactions, and logs.
Session Management	SharedPreferences	Manages session persistence, e.g., storing tokens or user state across sessions.
Background Jobs	Android Services, Threads, WorkManager	Background tasks like price drop alerts, order tracking, product auto-refresh.
Network / APIs	Retrofit / HttpURLConnection	Handles RESTful API communication for authentication, listing sync, and remote data operations.
Push Notifications	Firebase Cloud Messaging (FCM), Polling logic	Real-time updates for new listings, delivery status, message notifications, or price changes.

Logging & Debugging	Logcat, File Export, Debug Console	Logs system events (logins, purchases, errors) for both user and admin viewing and debugging.
Testing	JUnit, Espresso	Unit and UI testing to ensure app reliability, performance, and correctness.