CS501 Final Report: DealTracker — An Intelligent Android Price Comparison Application

Overview

DealTracker is an intelligent Android shopping comparison application designed to help users make informed purchasing decisions. It provides a unified platform for discovering and evaluating products across multiple e-commerce services, enabling users to efficiently identify competitive deals without having to navigate different shopping websites manually.

In addition to basic comparison capabilities, DealTracker focuses on reducing user effort through automation and personalization. By combining a clean and intuitive user interface with API-driven data integration and user-centered design principles, the application streamlines the shopping workflow from discovery to decision-making. Its design emphasizes clarity, efficiency, and usability, ensuring that users can quickly access relevant information while minimizing unnecessary interactions. Overall, DealTracker aims to save users both time and money by simplifying the online shopping experience.

Core Features

1. Product Search and Discovery: Users can search for products using text or voice input. The app supports keyword-based and fuzzy search, enabling flexible and efficient product discovery across platforms.

2. Price Comparison and Analysis: DealTracker retrieves real-time price and availability data from multiple e-commerce platforms (such as Amazon, eBay, and Walmart) via external APIs. Search results are presented in a structured list, with filtering and sorting options based on price, rating, shipping availability, and stock status.

3. Price History Visualization: For each product, DealTracker displays a 30-day price history, allowing users to analyze pricing trends and make better-informed purchasing decisions.

4. Wishlist and Price Alerts: Users can add products to a wishlist and set a target price. DealTracker continuously monitors price changes and sends notifications when the product price drops below the user-defined threshold, ensuring that users do not miss favorable deals.

5. Personalized Recommendations: The app recommends products based on the user's recent browsing history. If no browsing data is available, DealTracker falls back to globally high-rated products to maintain a relevant discovery experience.

6. User Profile and Preferences: DealTracker includes user account management features such as browsing history, wishlist access, profile editing, and customizable settings including font size and dark mode, enhancing accessibility and personalization.

Architecture

1. View (UI Layer)

Responsible for rendering the UI and collecting user events using Jetpack Compose. It observes the UI state exposed by the ViewModel via StateFlow and sends user actions to the ViewModel through callback functions.

Examples: DealsScreen observes the state of DealsViewModel. When users perform searches or apply filters, events are passed to the ViewModel via callback functions. Then, UI automatically updates based on state changes (Loading, Success, Error)

2. ViewModel (State Management)

Handles UI logic, manages UI state through StateFlow, and acts as the bridge between the View and Repository. It contains no Android framework dependencies.

3. Domain Layer

Defines core business models and business logic using pure Kotlin code independent of any framework.

Components:

- Domain Models: Pure data classes such as Product, User, PlatformPrice
- Repository Interfaces: Define data operation interfaces without depending on specific implementations

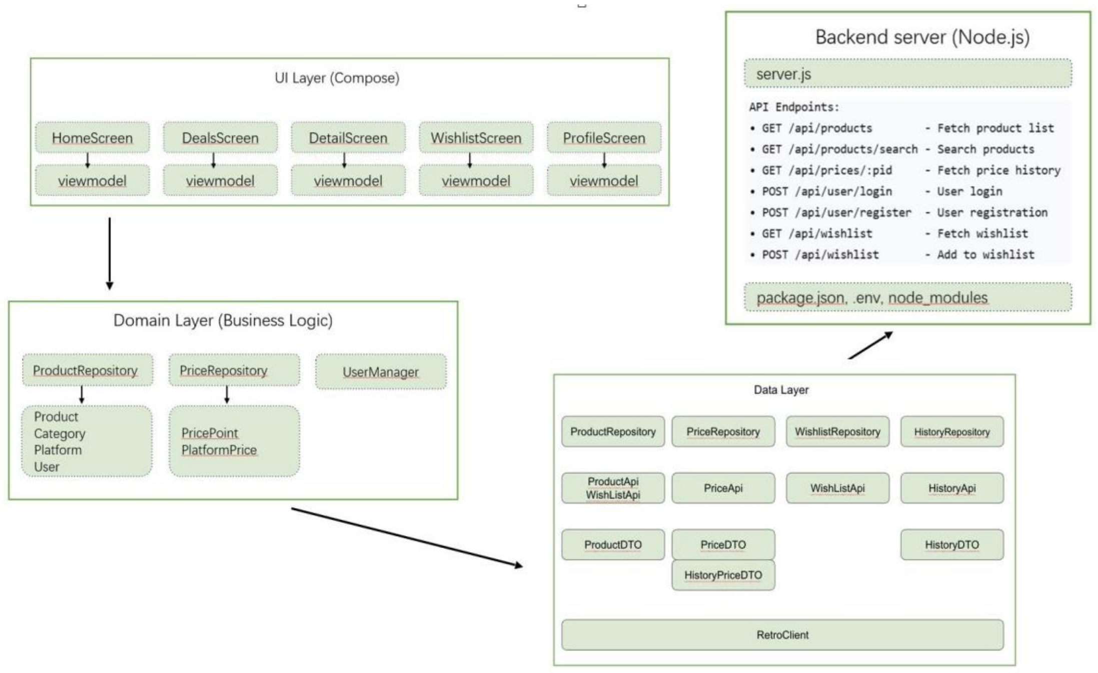
4. Data Layer

Contains Repository implementations, data sources, and network API interfaces. Responsible for data fetching, transformation, and caching.

Components:

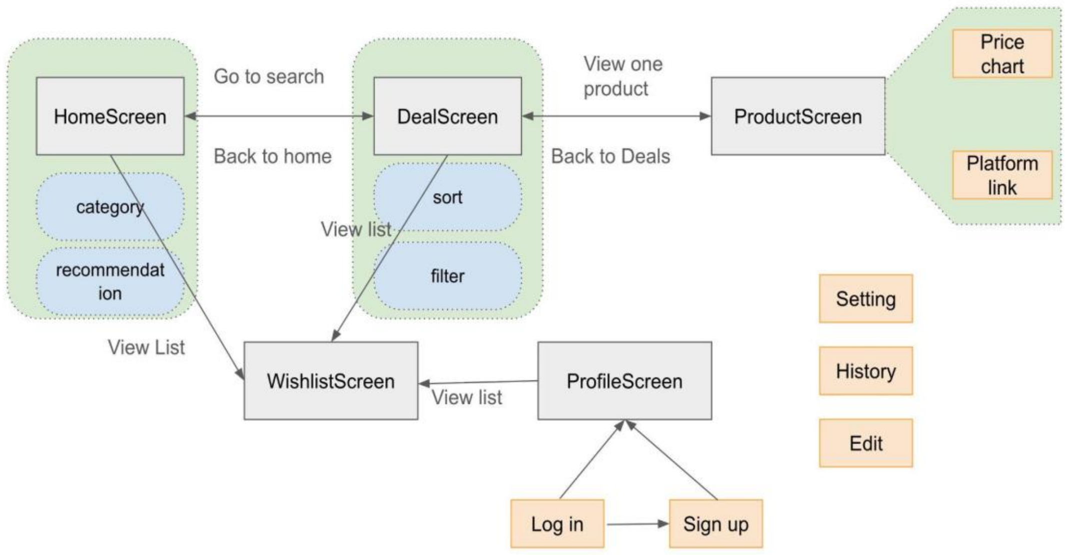
- **Remote Data Source**: API interfaces defined through Retrofit (DatabaseApiService, UserApi, WishListApi, etc.)
- **DTOs**: Data Transfer Objects (ProductDto, PriceDto, etc.)
- **Repository Implementation**: Implements interfaces defined in the domain layer, handles data fetching and error handling

Architecture Diagram



```
API Endpoints:
• GET /api/products         - Fetch product list
• GET /api/products/search  - Search products
• GET /api/prices/:pid      - Fetch price history
• POST /api/user/login      - User login
• POST /api/user/register   - User registration
• GET /api/wishlist         - Fetch wishlist
• POST /api/wishlist        - Add to wishlist
```

Navigation Flow Chart



Integration

1. External API Integration

The project achieves seamless integration with RapidAPI to fetch real-time product data from multiple e-commerce platforms.

RapidAPI Integration:

- **Real-Time Amazon Data API**: Fetches product information, prices, and availability from Amazon
- **eBay API**: Retrieves eBay product prices and stock status
- **Walmart API**: Obtains Walmart product data for price comparison

Backend Implementation:

- Node.js backend serves as an intermediary, calling RapidAPI endpoints to fetch multi-platform price data
- Automated daily price updates through cron jobs scheduled at 3:00 AM EST
- Data is processed and stored in a MySQL database for quick retrieval

Android Client Integration:

- Retrofit client connects to the Node.js backend API endpoints
- DTOs (ProductDto, PriceDto) map API responses to domain models
- Error handling for API quota limits (HTTP 429) and timeout scenarios

Multi-Platform Price Comparison:

- Aggregates prices from Amazon, eBay, and Walmart
- Calculates the lowest price across platforms
- Tracks price history for trend analysis

2. Sensor Integration

The application integrates the microphone as an input sensor to support voice-based search on mobile devices, improving user experience. Spoken input is converted to text through speech recognition and inserted into the search field for user review and confirmation.
Before initiating voice input, the system checks device support and microphone permission to ensure safe execution. If voice recognition fails, returns empty results, or is canceled, the application handles the situation gracefully by informing the user and falling back to manual text-based search, ensuring robustness and uninterrupted use.

3. Jetpack Compose Usage

The entire application is built using Jetpack Compose, adopting a declarative programming paradigm.

Key Features:

- All screens are Composable functions (HomeScreen, DealsScreen, etc.)
- Uses Material 3 Design components
- Manages local state through remember
- Handles side effects using LaunchedEffect

Custom Composables:

- Reusable UI components following Compose principles
- Stateless composables for better reusability and testing

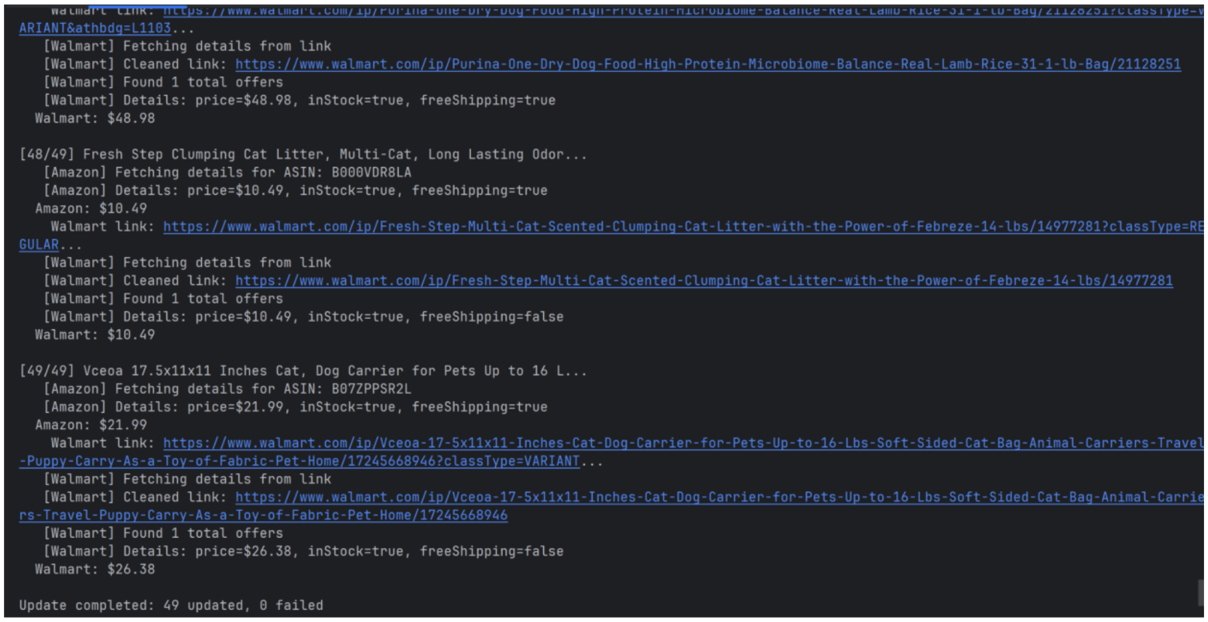
Advanced Compose Features: Canvas API for Data Visualization:

- ScrollablePriceChart component uses Compose's Canvas API to draw historical price charts
- Implements touch interactions using pointerInput modifier for gesture detection (scrolling, selecting data points)
- Custom drawing logic for price trends, grid lines, and interactive data visualization

Testing approach and results

Our application automatically updates all product prices and calculates the new lowest prices along with their corresponding platforms daily at 3:00 AM. To test this functionality, we developed dedicated API endpoints: update-all-prices and sync-lowest-prices, which were tested using Postman. Additionally, we implemented two testing interfaces: add-walmart-prices and sync-ebay-prices, to separately evaluate the system's capability to import data from the Walmart API and eBay API, respectively. To facilitate explicit problem identification, we incorporated comprehensive debugging logs into the codebase. By monitoring the backend console output, we were able to verify the correctness of parameter transmission across different components and perform effective debugging.

The following 2 figures demonstrate the backend output and Postman response results for the update-all-prices endpoints. As shown in the backend logs, we printed the API query parameters—including ASIN identifiers and URLs—as well as critical data returned from the APIs, such as price, free shipping status, and stock availability.

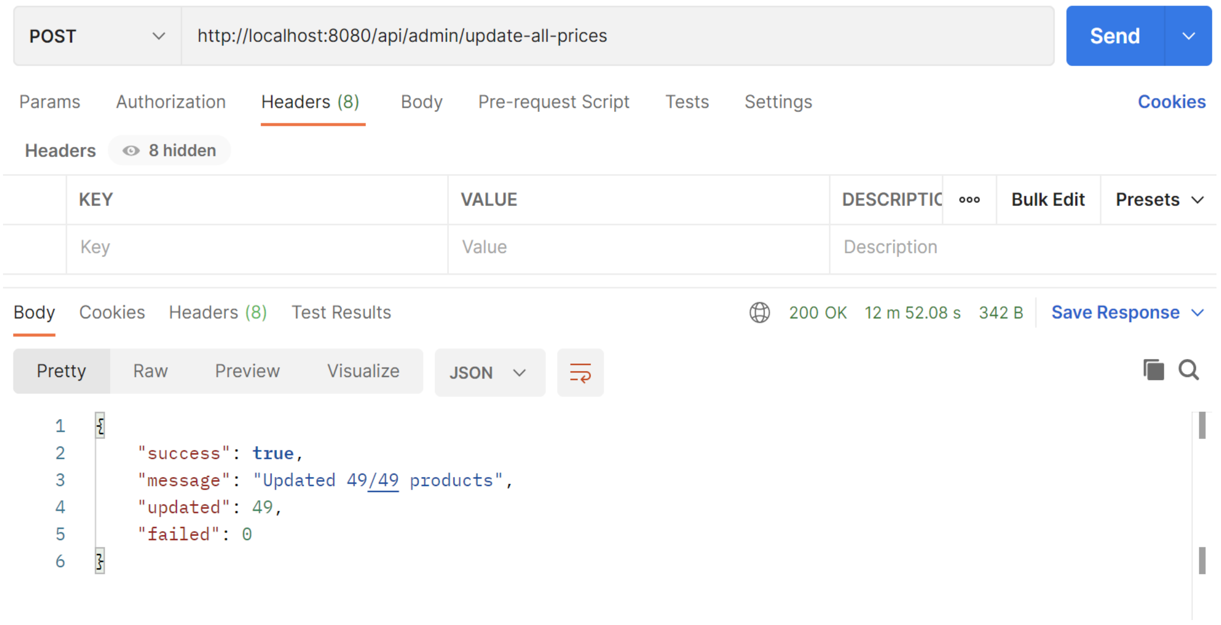


```
    Walmart link: https://www.walmart.com/ip/Purina-One-Dry-Dog-Food-High-Protein-Microbiome-Balance-Real-Lamb-Rice-31-1-lb-Bag/21128251?classType=V
ARIANT&athbdg=L1103...
   [Walmart] Fetching details from link
   [Walmart] Cleaned link: https://www.walmart.com/ip/Purina-One-Dry-Dog-Food-High-Protein-Microbiome-Balance-Real-Lamb-Rice-31-1-lb-Bag/21128251
   [Walmart] Found 1 total offers
   [Walmart] Details: price=$48.98, inStock=true, freeShipping=true
  Walmart: $48.98

[48/49] Fresh Step Clumping Cat Litter, Multi-Cat, Long Lasting Odor...
   [Amazon] Fetching details for ASIN: B000VDR8LA
   [Amazon] Details: price=$10.49, inStock=true, freeShipping=true
  Amazon: $10.49
    Walmart link: https://www.walmart.com/ip/Fresh-Step-Multi-Cat-Scented-Clumping-Cat-Litter-with-the-Power-of-Febreze-14-lbs/14977281?classType=RE
GULAR...
   [Walmart] Fetching details from link
   [Walmart] Cleaned link: https://www.walmart.com/ip/Fresh-Step-Multi-Cat-Scented-Clumping-Cat-Litter-with-the-Power-of-Febreze-14-lbs/14977281
   [Walmart] Found 1 total offers
   [Walmart] Details: price=$10.49, inStock=true, freeShipping=false
  Walmart: $10.49

[49/49] Vceoa 17.5x11x11 Inches Cat, Dog Carrier for Pets Up to 16 L...
   [Amazon] Fetching details for ASIN: B07ZPPSR2L
   [Amazon] Details: price=$21.99, inStock=true, freeShipping=true
  Amazon: $21.99
    Walmart link: https://www.walmart.com/ip/Vceoa-17-5x11x11-Inches-Cat-Dog-Carrier-for-Pets-Up-to-16-Lbs-Soft-Sided-Cat-Bag-Animal-Carriers-Travel
-Puppy-Carry-As-a-Toy-of-Fabric-Pet-Home/17245668946?classType=VARIANT...
   [Walmart] Fetching details from link
   [Walmart] Cleaned link: https://www.walmart.com/ip/Vceoa-17-5x11x11-Inches-Cat-Dog-Carrier-for-Pets-Up-to-16-Lbs-Soft-Sided-Cat-Bag-Animal-Carrie
rs-Travel-Puppy-Carry-As-a-Toy-of-Fabric-Pet-Home/17245668946
   [Walmart] Found 1 total offers
   [Walmart] Details: price=$26.38, inStock=true, freeShipping=false
  Walmart: $26.38

Update completed: 49 updated, 0 failed
```



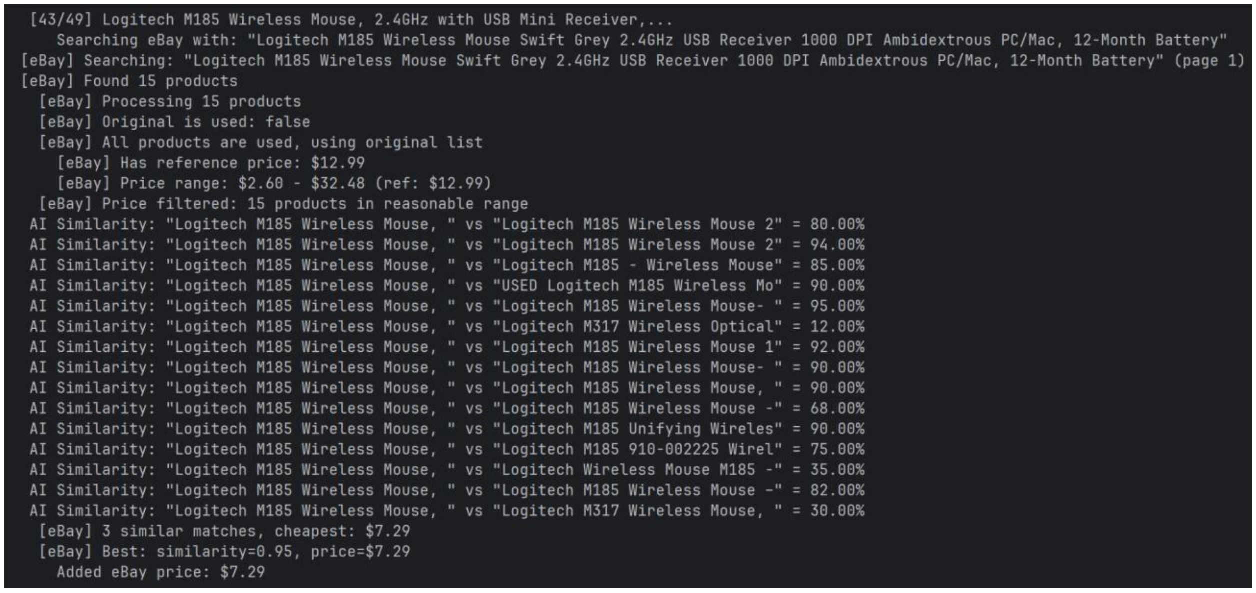
POST http://localhost:8080/api/admin/update-all-prices

KEY	VALUE	DESCRIPTIC
Key	Value	Description

200 OK 12 m 52.08 s 342 B

```json
{
    "success": true,
    "message": "Updated 49/49 products",
    "updated": 49,
    "failed": 0
}
```

The same methodology was applied to the sync-ebay-prices interface, which was designed to test the functionality of querying products from the database and inserting relevant information into the price table via eBay searches. Through detailed and clear backend logging, we were able to observe crucial code logic execution. Furthermore, this approach enabled us to validate the reliability of the AI-powered title matching similarity scores, confirming whether the most relevant products were successfully identified on eBay.



```
 [43/49] Logitech M185 Wireless Mouse, 2.4GHz with USB Mini Receiver,...
    Searching eBay with: "Logitech M185 Wireless Mouse Swift Grey 2.4GHz USB Receiver 1000 DPI Ambidextrous PC/Mac, 12-Month Battery"
[eBay] Searching: "Logitech M185 Wireless Mouse Swift Grey 2.4GHz USB Receiver 1000 DPI Ambidextrous PC/Mac, 12-Month Battery" (page 1)
[eBay] Found 15 products
  [eBay] Processing 15 products
  [eBay] Original is used: false
  [eBay] All products are used, using original list
    [eBay] Has reference price: $12.99
    [eBay] Price range: $2.60 - $32.48 (ref: $12.99)
  [eBay] Price filtered: 15 products in reasonable range
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 Wireless Mouse 2" = 80.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 Wireless Mouse 2" = 94.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 - Wireless Mouse" = 85.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "USED Logitech M185 Wireless Mo" = 90.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 Wireless Mouse- " = 95.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M317 Wireless Optical" = 12.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 Wireless Mouse 1" = 92.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 Wireless Mouse- " = 90.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 Wireless Mouse, " = 90.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 Wireless Mouse -" = 68.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 Unifying Wireles" = 90.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 910-002225 Wirel" = 75.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech Wireless Mouse M185 -" = 35.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M185 Wireless Mouse –" = 82.00%
 AI Similarity: "Logitech M185 Wireless Mouse, " vs "Logitech M317 Wireless Mouse, " = 30.00%
  [eBay] 3 similar matches, cheapest: $7.29
  [eBay] Best: similarity=0.95, price=$7.29
    Added eBay price: $7.29
```

Reflection on teamwork and Agile practices

DealTracker was developed by our three-member team through an iterative and collaborative process. Leyan Chen coordinated the project and focused on UI/UX design, sensor integration, and application testing. Yijia Chen was mainly responsible for the navigation structure, external API integration, product database design, and backend stability. Juling Fan handled user database design, data persistence, data visualization, and external integrations such as sharing and redirection to third-party platforms. While responsibilities were clearly divided, many features required close collaboration across different parts of our team.

Throughout development, we followed an Agile-style workflow that emphasized gradual progress and continuous refinement. The project evolved from implementing core functionality to adding more advanced features over multiple iterations. GitHub was used to manage the codebase, with separate branches created for major features or development stages. Before merging changes, we discussed implementation decisions and tested different approaches to ensure the application remained stable and consistent.

When merge conflicts or design disagreements occurred, we addressed them through open discussion rather than enforcing a single solution. By comparing different design choices and evaluating their trade-offs through testing, we were able to reach better decisions as a team. This experience highlighted the importance of clear communication, early planning, and incremental development, and helped us gain a more practical understanding of collaborative software development.

AI Reflection

We utilized two AI tools in our development process: ChatGPT and Claude. ChatGPT assisted us in retrospectively analyzing the project, summarizing technical highlights, core functionalities, and overall architecture, thereby facilitating report writing and presentation slide preparation. Furthermore, one of our application's critical features is cross-platform price comparison, which necessitated identifying identical or similar products across different platforms—a significant technical challenge. We leveraged OpenAI's API to perform natural language processing (NLP) for two primary purposes: first, to extract concise, search-optimized short titles from Amazon's lengthy product titles; and second, to use these abbreviated titles to query other platform APIs and determine which returned product titles best matched the original item. With minimal code implementation and simple prompts, we achieved remarkably effective results.

Claude provided substantial assistance in our code implementation; however, all core functionalities, implementation logic, and the MVVM architecture were designed independently by our team. To enhance development efficiency, we judiciously employed Claude to rapidly scaffold the project framework, while manually implementing detailed code logic ourselves. Additionally, we strategically utilized AI's rapid diagnostic capabilities during debugging. By incorporating comprehensive logging statements throughout our codebase and providing the backend output to the AI, we were able to quickly identify overlooked issues.

Throughout this semester's development process, we recognized that AI assistance presents both advantages and limitations. While it undeniably accelerated development efficiency in the initial stages, its utility became increasingly constrained as our application's functionality and logic grew more complex. The AI typically focused on isolated functions or individual file-level code correctness, failing to comprehend broader invocation logic and holistic functionality implementation. Consequently, manual code modification often proved more efficient than relying on AI suggestions, ultimately resulting in time inefficiency.

When incorporating AI-generated code, we maintained a rigorous review process rather than blindly copying and pasting. Our code review and refactoring methodology included the following systematic approaches:
Quality Assurance: We meticulously examined each line of code, verified the correctness of function and parameter invocations. We also conducted comprehensive testing of AI-generated code segments, validating that they produced expected outputs under various input conditions and edge cases.
Security Review: We scrutinized all data handling operations, particularly those involving user information and API credentials, to ensure proper input validation, secure data transmission, and adherence to privacy best practices. Any hardcoded sensitive information was replaced with environment variables or secure storage solutions.
Kotlin/Compose Idioms Adherence: We refactored AI-generated code to follow Kotlin best practices, including the use of nullable types, extension functions, and coroutines for asynchronous operations.