

Mock Investor: A Virtual Stock Trading Android App

Final Project Report

Parth Kheni

Boston University

Course: ENG EC327
Section: Section C7
Instructor: Professor Ed Solovey
Date: 05/05/2025

Repository: https://github.com/parthnkheni/personal_finance_app

Contents

1 Abstract	1
2 Introduction	1
2.1 Project Goals	1
3 System Overview	1
3.1 Key Features	1
3.2 Target Platform and Tools	1
4 Requirements	2
4.1 Functional Requirements	2
4.2 Non-Functional Requirements	2
5 Design and Architecture	2
5.1 High-Level Architecture	2
5.2 Data Model (Conceptual)	3
5.3 User Interface (Screens)	3
6 Implementation	3
6.1 Portfolio Management	3
6.2 Stock Search and Market Data	3
6.3 Price Alerts and Notifications	4
7 Testing and Validation	4
7.1 Test Strategy	4
7.2 Representative Test Cases	4
8 Setup and Usage Guide	4
8.1 Environment Setup (Android Studio)	4
8.2 How to Use the App	5
9 Limitations	5
10 Future Work	5
11 Conclusion	5

1 Abstract

Mock Investor is an Android application that simulates real-time stock trading to help users learn personal finance concepts through hands-on experimentation. Users can build and manage a mock portfolio of U.S. stocks, view portfolio value updates, browse and search stock listings via an integrated stock data API, and configure price alerts that trigger Android notifications when thresholds are crossed. This report documents the problem motivation, requirements, high-level system design, implementation approach, and testing considerations.

2 Introduction

Learning how markets behave is difficult when the only exposure is reading static examples. Mock Investor addresses this by providing a safe environment where users can build a portfolio and observe how price changes impact total value over time. The app focuses on core learning interactions: selecting tickers, specifying share quantities, monitoring values, and setting alerts for market movements.

2.1 Project Goals

- Provide an intuitive mobile interface for creating and viewing a mock stock portfolio.
- Fetch and display up-to-date stock price and listing information using a third-party API.
- Persist user portfolio data locally so it remains available across app sessions.
- Allow users to configure price alerts and receive system notifications when triggered.

3 System Overview

3.1 Key Features

The project implements three core feature groups:

1. **Portfolio Management:** add/remove stocks, set quantities, compute per-position value (price \times quantity), and compute total portfolio value.
2. **Search and Browse:** search by name or ticker, browse U.S. stocks from an integrated API, and filter results.
3. **Price Alerts:** set above/below thresholds and deliver Android system notifications when thresholds are crossed.

3.2 Target Platform and Tools

- Platform: Android
- Development environment: Android Studio
- Primary language(s): Java with some Kotlin (as indicated by repository language breakdown)
- Build system: Gradle

4 Requirements

4.1 Functional Requirements

- FR1: The user can add a U.S. stock to a portfolio by specifying a ticker and quantity.
- FR2: The app displays the current price and computed position value for each portfolio holding.
- FR3: The app calculates and displays the total portfolio value as the sum of all positions.
- FR4: The user can remove a stock from the portfolio.
- FR5: The user can search stock symbols by ticker or company name.
- FR6: The user can browse a list of available stocks and apply basic filters.
- FR7: The user can create price alerts (above/below a threshold).
- FR8: The app generates a system notification when an alert condition is met.
- FR9: The portfolio persists locally across application restarts.

4.2 Non-Functional Requirements

- NFR1: Responsiveness—UI should remain usable during network requests (use async operations).
- NFR2: Reliability—gracefully handle API failures and show meaningful error states.
- NFR3: Usability—simple workflows for adding stocks and setting alerts.
- NFR4: Maintainability—separate UI logic from data/network logic to support future extensions.

5 Design and Architecture

5.1 High-Level Architecture

At a high level, the app can be viewed as four interacting components: (1) UI screens for user interaction, (2) a portfolio domain layer, (3) a data layer for API calls, and (4) local persistence + alert/notification handling.

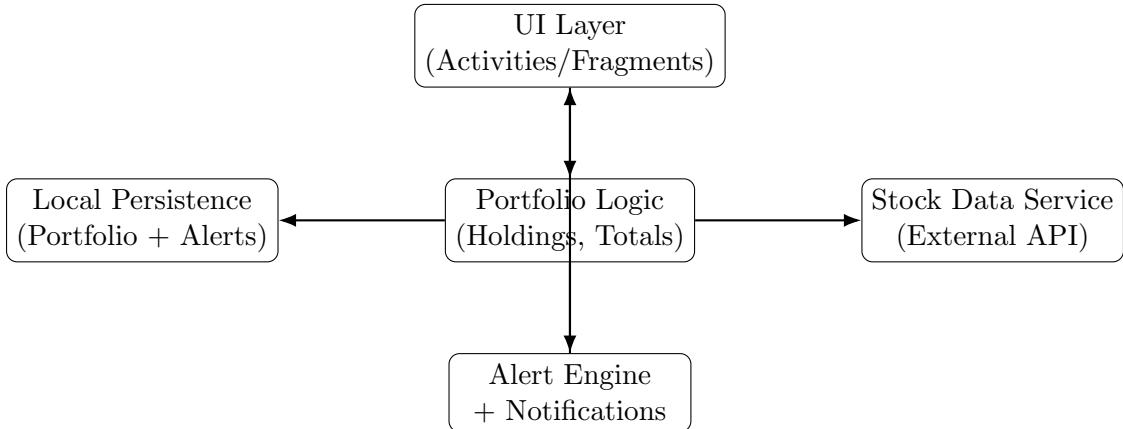


Figure 1: Conceptual system architecture for Mock Investor.

5.2 Data Model (Conceptual)

A minimal data model is sufficient for the core functionality:

- **Holding:** symbol, quantity, lastPrice, positionValue
- **Portfolio:** list of holdings + totalValue
- **Alert:** symbol, threshold, direction (above/below), enabled/disabled

5.3 User Interface (Screens)

Typical screens include:

- Portfolio screen (holdings list + total value)
- Add/Edit holding flow
- Search/Browse stocks screen (filter + results list)
- Alerts configuration screen

6 Implementation

6.1 Portfolio Management

The portfolio feature centers on CRUD operations: create holdings (add symbol + quantity), read-/display holdings with current pricing, update quantities, and delete holdings. The app recalculates:

$$positionValue = price \times quantity, \quad totalValue = \sum_i positionValue_i$$

To ensure continuity across sessions, portfolio state is saved locally (implementation may use Android local storage mechanisms such as a preferences store or a local database).

6.2 Stock Search and Market Data

The application integrates with a stock listings/quotes API to:

- Search symbols by ticker or company name,
- Browse a larger list of available U.S. stocks,
- Retrieve current pricing used in holdings valuation.

Network operations are performed asynchronously to avoid blocking the UI thread, and error cases (no connectivity, rate limits, server errors) are handled with user-friendly feedback.

6.3 Price Alerts and Notifications

Price alerts require three steps:

1. Store the alert configuration (symbol, direction, threshold).
2. Periodically or event-driven check the latest price against the threshold.
3. Trigger an Android system notification when the threshold condition becomes true.

Notifications are designed to be concise (symbol + threshold crossing message) and actionable (tapping can route the user back into the app).

7 Testing and Validation

7.1 Test Strategy

Testing focuses on correctness, stability, and UX behavior:

- **Unit tests:** portfolio math (position value, totals), alert trigger conditions.
- **Integration tests:** API response parsing, handling empty/invalid results.
- **UI tests (manual):** common user flows (add holding, remove holding, set alert).

7.2 Representative Test Cases

- TC1: Add AAPL with quantity 10; verify holding appears and total updates.
- TC2: Remove a holding; verify total decreases accordingly and holding disappears.
- TC3: Search “Apple”; verify results contain expected tickers (when available from API).
- TC4: Set alert “AAPL above 200”; simulate price crossing and confirm notification fires once.
- TC5: Restart app; confirm portfolio persists and loads correctly.

8 Setup and Usage Guide

8.1 Environment Setup (Android Studio)

1. Install Java (latest JDK) and Android Studio.
2. Clone the repository:

```
git clone https://github.com/partnkheni/personal_finance_app
```

3. Ensure `local.properties` is configured with your Android SDK path. Example:

```
sdk.dir=C:\\\\Users\\\\YourWindowsUserNameHere\\\\AppData\\\\Local\\\\Android\\\\
SdK
```

4. Sync Gradle files. If needed, enable AndroidX in `gradle.properties`.
5. Build and run on an emulator or physical Android device.

8.2 How to Use the App

1. Add stocks to your portfolio by selecting a symbol and entering a quantity.
2. View each position's value and the total portfolio value on the main portfolio screen.
3. Use search/browse to discover tickers and explore available listings.
4. Configure alerts (above/below threshold) for symbols you care about.
5. When the market price crosses your threshold, the app sends a notification.

9 Limitations

- The app is intended for learning and simulation; it does not execute real trades.
- Behavior depends on the availability and limitations (e.g., rate limits) of the external stock data API.
- Alert timing/accuracy depends on how frequently prices are refreshed and checked.

10 Future Work

- Add richer analytics (gain/loss charts, historical performance, sector breakdown).
- Add watchlists separate from holdings, and paper-trading transaction history.
- Improve alert logic (cooldowns, multiple alerts per symbol, alert history).
- Enhance persistence and offline mode with more robust caching.
- Expand filtering/search UX and add accessibility improvements.

11 Conclusion

Mock Investor delivers a practical learning-focused mobile experience for exploring portfolio construction, valuation, and simple alert-based monitoring of market movements. By combining portfolio management, searchable listings, and notification-based alerts, the project demonstrates core Android app development concepts (UI design, networking, persistence, and notifications) in a cohesive personal finance application.