Authors

Name: Beaudlaire Jeancharles

BUID: U70445938

Name: Jiahe Zhang

BUID: U82392079

Name: Jiahao He

BUID: U03417870

Stock Market Analysis App

Final Project Documentation

Overview

The **Stock Market Analysis App** is an Android-based application built using modern tools such as **Jetpack Compose**, **Hilt**, **Room**, **Retrofit**, and OpenAI's **GPT-3.5 API**. It provides users with real-time stock data visualization, advanced trading insights, watchlist management, and offline caching.

The app is designed to address the needs of stock enthusiasts, day traders, and beginners by combining a clean user interface with powerful analytical tools. With features like real-time notifications, personalized watchlists, and AI-driven insights, users can easily track trends, make informed decisions, and improve their investment strategies. By integrating offline support, the app ensures accessibility anytime, anywhere, further enhancing user experience.

Market and Users

Market Background and User Painpoints

The stock market analysis sector attracts a variety of users with different levels of trading expertise. Key user pain points that impact user experience include:

- **No Data Showing Offline**: Users lack the ability to access critical trading and analytical data when disconnected from the internet.
- **Complicated UI**: Many existing platforms overwhelm users with complex designs, making navigation and analysis difficult, particularly for beginners.
- **Insufficient Operation Analysis**: There is a lack of comprehensive tools for profit and position tracking, analysis of trading behavior, and identifying trends.

The market demands a simple, accessible, and intuitive solution for stock analysis and operations, one that combines powerful insights with user-friendly design.

Target Users

Stock Enthusiasts: Stock enthusiasts rely on clear data to analyze trends and make investment decisions. Our app provides intuitive visualizations, such as *Trading Trends* and *Category Preferences*, enabling users to identify market patterns and active sectors easily. Simplified charts ensure efficient analysis without overwhelming complexity.

Day Traders: Day traders need fast, actionable insights to optimize their trades. The app delivers key metrics, including active trading times and profit-loss trends, helping users quickly adjust strategies. Offline access ensures continuous performance tracking, even without connectivity.

Beginners in the Stock Market: For beginners, the app simplifies learning with clear charts and easy-to-understand metrics, such as *Transaction Analysis* and *Most Traded Stocks*. This approach reduces complexity, helping new investors build confidence and knowledge step by step.

User Stories

- **Stock Enthusiast**: "As a stock enthusiast, I want to easily track trading volumes and active sectors so that I can identify trends and optimize my investments."
- **Day Trader**: "As a day trader, I need to see the most active buying/selling times and my trading profitability to make faster and more informed trading decisions."
- **Beginner Investor**: "As a beginner, I want an easy-to-understand dashboard with clear metrics and trends so I can learn and gradually grow my confidence in trading."

App Design

Tech Stack and Dependencies To achieve a seamless and robust application, we chose modern technologies that align with our goals of performance, modularity, and usability:

- **Jetpack Compose**: Used for building the user interface due to its declarative and composable nature. This allowed us to create reusable UI components while improving the readability and maintainability of the codebase. Compose simplifies the process of updating UI elements dynamically based on state changes.
- **Hilt (Dependency Injection)**: Hilt was selected for dependency injection to ensure clean, modular, and testable code. By managing dependencies efficiently, Hilt allows us to decouple components, making the project scalable and easy to maintain.
- **Room Database**: For offline data caching and persistent storage, Room was integrated. It provides an abstraction layer over SQLite, ensuring safe and efficient local data handling. This enables users to access previously fetched stock data even without an internet connection.

- **Retrofit**: Retrofit was used for managing API calls due to its simplicity and reliability in handling HTTP requests. It integrates seamlessly with RESTful APIs and handles JSON parsing efficiently. Retrofit ensures fast data retrieval, which is critical for real-time stock data analysis.
- **ViewModel and State Management**: Android ViewModel was used to manage UI-related data lifecycle-aware components. This ensures that data survives configuration changes, like screen rotations, providing a smooth user experience. Jetpack Compose's built-in state management complements this setup by updating UI components reactively.
- **LiveData and Flow**: For asynchronous programming and data observation, LiveData and Kotlin Flow were implemented. These ensure efficient data synchronization between the database, API, and the UI layer.
- **Navigation Component**: The Jetpack Navigation Component was used to manage navigation between screens, ensuring a consistent and smooth transition within the app. This made handling user flows intuitive and streamlined.

Architectural Design: The project follows a clean and modular architectural design divided into four main layers:

- **Data Layer**: Handles all data-related operations, including network requests (via Retrofit), database management (Room), and data parsing. This layer provides the app with stock data in a structured and efficient manner.
- **Domain Layer**: Contains core business logic, including use-case classes and domain models that act as intermediaries between the Data and Presentation layers. This ensures separation of concerns and keeps the app modular.
- **Dependency Injection (DI) Layer**: Uses Hilt to manage dependencies across the app. DI ensures that each class gets the dependencies it needs, reducing tight coupling and improving testability.
- **Presentation Layer**: Consists of Jetpack Compose-based UI components, ViewModels, and state management for rendering data and handling user interactions efficiently. The presentation layer ensures a responsive and visually appealing user interface.

API Integration and Data Flow: To fetch and display real-time data, the app uses the following flow:

1. **Alpha Vantage API (Data Source)**: Provides accurate and real-time stock market data, including intraday, weekly, and monthly price information. It serves as the primary data source for market trends and stock prices.
2. **Global Financial News Feed API**: Aggregates financial news articles and updates related to the stock market, allowing users to stay informed about global economic trends and events.

3. **OpenAI GPT-3.5 API**: The GPT API powers AI-driven insights by analyzing stock trends and user data to generate actionable recommendations. This feature makes the app stand out by providing intelligent predictions and simplifying decision-making for users.
4. **Firebase Authentication**: Manages secure sign-in and sign-out operations, ensuring robust user authentication using Google Authentication as the primary sign-in method.
5. **Firebase Realtime Database**: Allows seamless data synchronization and cloud-based storage for user-specific data, such as watchlists and saved preferences.
6. **Retrofit** communicates with RESTful APIs to fetch stock data.
7. Data is parsed using specialized parsers (*IntradayInfoParser*,
8. The parsed data is stored in the **Room Database** for offline access and persistence. *WeeklyInfoParser*, etc.) to structure it for use within the app.
9. The parsed data is stored in the **Room Database** for offline access and persistence.
10. **ViewModels** observe data changes and provide it to Jetpack Compose UI components.
11. The user interface dynamically updates to reflect the latest stock information using Jetpack Compose's state management capabilities.

This design ensures a smooth user experience by providing real-time data updates, offline accessibility, and a clean modular architecture that is easy to extend and maintain.

—

—

Features

1. Authentication (Login)

- **AuthRepository**: Handles user authentication logic, including sign-in, and sign-out operations using Firebase.
- **AuthState**: Represents the state of user authentication such as success, error, or loading.
- **AuthViewModel**: Exposes authentication state and interacts with the repository for managing workflows.
- **GoogleAuthUiClient**: Manages Google One Tap Sign-In and Firebase authentication.
- **loginAndSignupScreen.kt**: Implements the login UI with input fields and Google Sign-In integration.

- **SignInResult.kt**: A data class containing sign-in results, such as user data and error details.

2. Main Screen (Main_Screen)

- **BottomNavigationBar.kt**: Implements the bottom navigation bar for navigating between app screens.
- **FloatingTitle.kt**: Displays floating animated titles to enhance user interface aesthetics.
- **HomepageScreen.kt**: The main dashboard screen displaying stock market data, charts, and user watchlists.
- **HomeViewModel**: Manages stock data loading, user data, and watchlist integration for the home screen.

3. Trading Analysis (trading_analysis)

- **ChartCard.kt**: Displays stock charts with detailed price and trend visualizations.
- **TradingAnalysisScreen.kt**: Displays detailed trading insights, charts, and trend data.
- **TradingAnalysisState.kt**: Represents the loading, success, and error states for the trading analysis screen.
- **TradingAnalysisViewModel**: Fetches and manages trading data, generating insights for the analysis screen.

4. Watchlist Management (watch_list)

- **WatchListScreen.kt**: Displays the user's personalized watchlist with options to add or remove stocks.
- **WatchListViewModel**: Manages all watchlist operations, such as adding, removing, and retrieving favorite stocks.

5. Data Parsing and Management

- **CompanyListingsParser**: Parses raw CSV data for stock company listings, including names, symbols, and exchanges.
- **IntradayInfoParser**: Extracts intraday stock data for real-time price analysis and chart generation.
- **WeeklyInfoParser**: Processes weekly stock data to analyze mid-term performance trends.
- **MonthlyInfoParser**: Parses monthly data to analyze long-term trends and stock performance history.

- **NasdaqScreenerParser**: Extracts Nasdaq stock listings for efficient filtering and management.
- **TradingDataParser**: Converts raw trading data into structured models for trend analysis and visualization.

6. Repository and Dependency Injection

- **Repository Layer**: Acts as a bridge between the remote APIs, database operations, and app components, consolidating data efficiently.
- **AppModule**: Provides global dependencies for Room, Retrofit, and other core components using Hilt.
- **RepositoryModule**: Supplies repository instances for seamless data access and management.

—

Future Enhancements

- Integrate **Real-Time Notifications** to alert users about stock price changes and significant market updates, ensuring users never miss key opportunities.
- Add **Portfolio Management** for users to simulate, track, and analyze their investments over time, offering tools for profit tracking, trend comparison, and risk analysis.
- Enhance **Predictive Analytics** using GPT-3.5 and machine learning models to deliver trend forecasts, personalized recommendations, and AI-driven trading insights.
- Expand **Stock Screening Filters** to include sector-specific searches, real-time financial indicators, and customizable filters for identifying investment opportunities.
- Develop a **Social Trading Platform** where users can follow expert traders, view their strategies, and learn from their trades in a collaborative environment.
- Introduce **Educational Tools** such as in-app tutorials, trading glossaries, and video guides to help beginners gain foundational knowledge and confidence.
- Launch **Cross-Platform Support** for iOS and web, ensuring seamless accessibility across all devices and expanding the app's market reach.

—

Challenges and Learnings

Challenges Faced

Throughout the development of the Stock Market Analysis App, our team encountered several challenges:

- **Integrating Multiple APIs**: Combining data from multiple APIs, such as Alpha Vantage for stock prices, GPT-3.5 for insights, and Firebase for authentication, was challenging. Ensuring seamless integration and synchronization required extensive debugging and proper error handling.
- **State Management with Jetpack Compose**: Managing complex UI states, especially for features like real-time stock updates, trading analysis, and watchlists, was initially difficult. Learning how to effectively use Jetpack Compose's state management capabilities was crucial for ensuring a smooth user experience.
- **Offline Caching with Room**: Implementing offline data caching for stock information and user preferences required careful planning. Ensuring data consistency and avoiding redundancy during synchronization between the local database and APIs proved to be time-consuming.
- **Dependency Management with Hilt**: While Hilt simplified dependency injection, learning its implementation and configuring dependencies correctly for each layer of the app initially posed a learning curve.
- **Handling Real-Time Updates**: Providing near real-time stock updates while optimizing performance was a key challenge. We had to strike a balance between frequent API calls and app responsiveness.
- **UI/UX Optimization**: Designing a user-friendly interface that caters to beginners, enthusiasts, and professional traders required iterative feedback and multiple design adjustments.

What We Learned

Through overcoming these challenges, we gained valuable experience and insights:

- **API Integration**: We learned how to work with RESTful APIs effectively, manage network requests using Retrofit, and handle errors gracefully to deliver a robust experience.
- **State Management Best Practices**: By mastering Jetpack Compose and ViewModel, we now have a deep understanding of how to manage UI states efficiently and optimize performance.

- **Offline Data Handling**: Implementing Room for offline storage taught us the importance of data synchronization, query optimization, and maintaining database integrity.
- **Dependency Injection**: Working with Hilt strengthened our knowledge of modular design and clean architecture, enabling us to write testable and maintainable code.
- **Real-Time Updates and Performance Optimization**: We learned techniques to optimize real-time data retrieval and display without overwhelming the system or API limits.
- **User-Centered Design**: Designing intuitive UI components and improving accessibility reinforced the importance of aligning development with user needs.
- **Team Collaboration**: Effective communication, code reviews, and agile project management were critical in ensuring smooth development and successful delivery of the project.

Overall, this project has deepened our understanding of modern Android development practices and helped us acquire skills that will be invaluable for future projects.

—

Conclusion

The **Stock Market Analysis App** incorporates essential features such as real-time stock data visualization, trading insights, and user watchlist management, supported by a solid architectural foundation. While the app integrates most of the core functionalities aligned with our product goals and has been a valuable learning experience, it remains a work in progress. Several areas, such as predictive analytics, portfolio simulation, and real-time notifications, still need further development to fully meet our expectations. Future iterations will focus on refining these features and enhancing overall usability to provide a more robust platform for traders.