Software Architecture & Design - Coursework– Sprint 1

Team Members:

Evan Balson - BAL18466416
Maahia Rahman - RAH23614335
Hala Bakhtiar - BAK23592238
Mahbouba Rezaei - REZ23579670

Roehampton University

Course Number: CMP020N207S

Lecturer: Dr Shekoufeh Kolahdouz Rahimi

18/02/2025

StockShark Web Application Documentation

Executive Summary

StockShark is a comprehensive stock analysis application designed to provide real-time and historical stock data analysis, enabling users to make informed investment decisions.

Leveraging Yahoo Finance API for data retrieval and MySQL for persistent storage, StockShark aims to offer seamless access to offer a robust set of features that cater to both novice and experienced investors.

Core Functionalities: Overview

1. **Real-Time and Historical Data Retrieval:** StockShark uses the YFinance API to fetch real-time stock prices and historical data. Data is pre-processed for format consistency and accuracy before being stored in a MySQL database with entry protection mechanisms to prevent data duplication.

2. **Data Persistence**: Utilizing mysql-connector-python, StockShark ensures that all stock data are persistently stored, allowing for offline access and historical data analysis.
3. **Graphical Performance Comparison**: The application features a Java-based client interface that displays interactive charts and graphs for comparing the performance of multiple stocks. This is supported by Python backend services that manage data fetching, processing, and API handling.
4. **AI-Powered Trend Prediction**: StockShark incorporates an AI analyzer built on Python. This module will use machine learning models, to predict stock trends based on historical data. This feature aims to provide users with foresight into potential future stock movements.
5. **User Interface and Experience**: The Java client will ensure a seamless user experience with a clean and intuitive interface that includes functionalities such as a portfolio tracker, watchlist, and personalized stock alerts. The system supports user sessions, including secure login/logout functionalities.

Technical specifications:

1. **Concurrency Management**: Python's asyncio library is expected to be employed to handle concurrent API calls and database operations efficiently.
2. **Security and Compliance**: StockShark will implement robust security measures including encrypted data transmission, SQL injection prevention, and compliance with GDPR for handling user data.
3. **Notification System**: An integrated notification system alerts users about significant stock movements and system anomalies, improving user engagement and system reliability.

Future work:

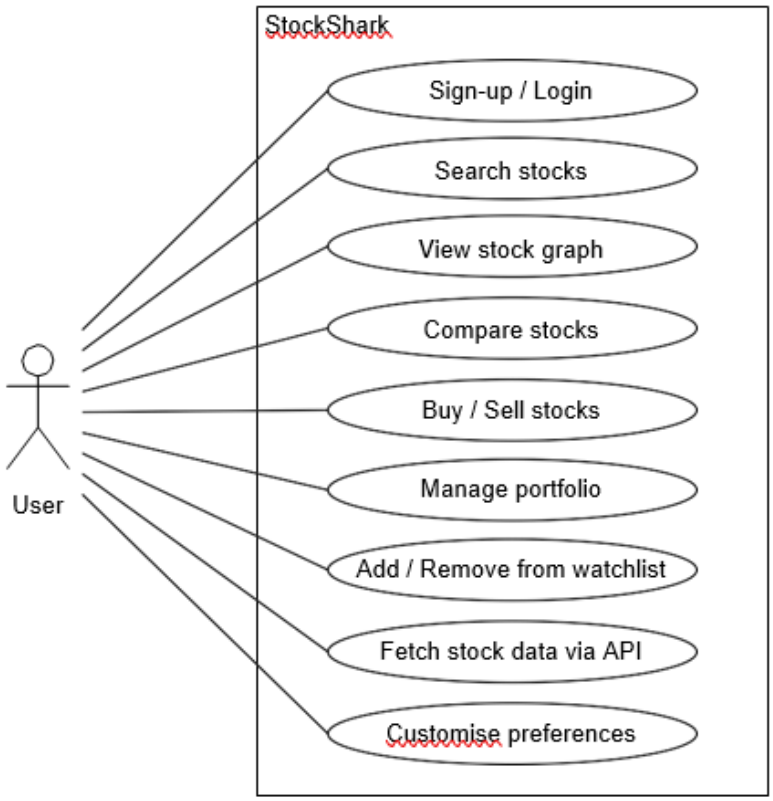
1. **Performance Optimization**: Implement caching mechanisms to improve data retrieval times and responsiveness of the application.
2. **Expanded AI Capabilities**: Future updates aim to incorporate more advanced AI-driven tools for market analysis and portfolio optimization.

Project Requirements

StockShark's core functionalities include daily stock price retrieval, persistent data storage, graphical performance comparison, and optional AI-based trend predictions. Data is fetched via Yahoo Finance API using YFinance, stored in MySQL, and visualized through Java components.

Features Collected From Yahoo Finance	**Features Collected From Google Finance**	**Features Collected From Bloomberg Terminal**
☑ **Features:** • Displays real-time and historical share prices. • Allows comparison between multiple stocks. • Provides graphical charting tools. • Offers financial news and AI-powered insights. 🔎 **Key Takeaways for Our Project:** • Implement a **simple UI** with **interactive charts**. • Support **historical data retrieval** for comparisons. • Ensure **data persistence for offline access**. • Consider **AI-based insights for trend prediction**.	☑ **Features:** • Tracks real-time market trends. • Allows personalized watchlists. • Displays stock market news and financial analysis. 🔎 **Key Takeaways for Our Project:** • Include a **stock search feature**. • Provide **side-by-side stock comparisons**. • Ensure a **clean and simple UI**.	☑ **Features:** • Professional-grade analytics and market data. • Offers advanced financial models and trend analysis. • Features **AI-powered forecasting tools**. 🔎 **Key Takeaways for Our Project:** • Consider implementing **basic AI-driven trend analysis**. • Offer **historical stock data visualization**.

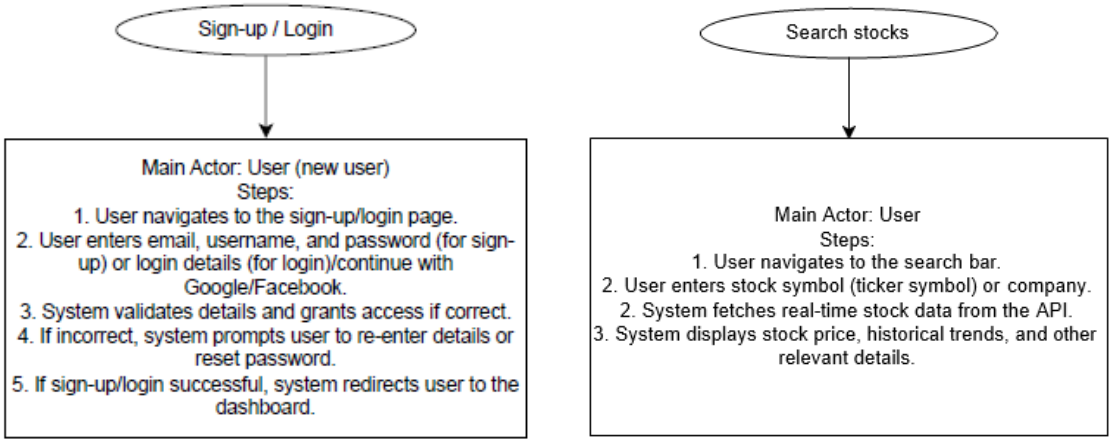
➢ Use Case Diagram

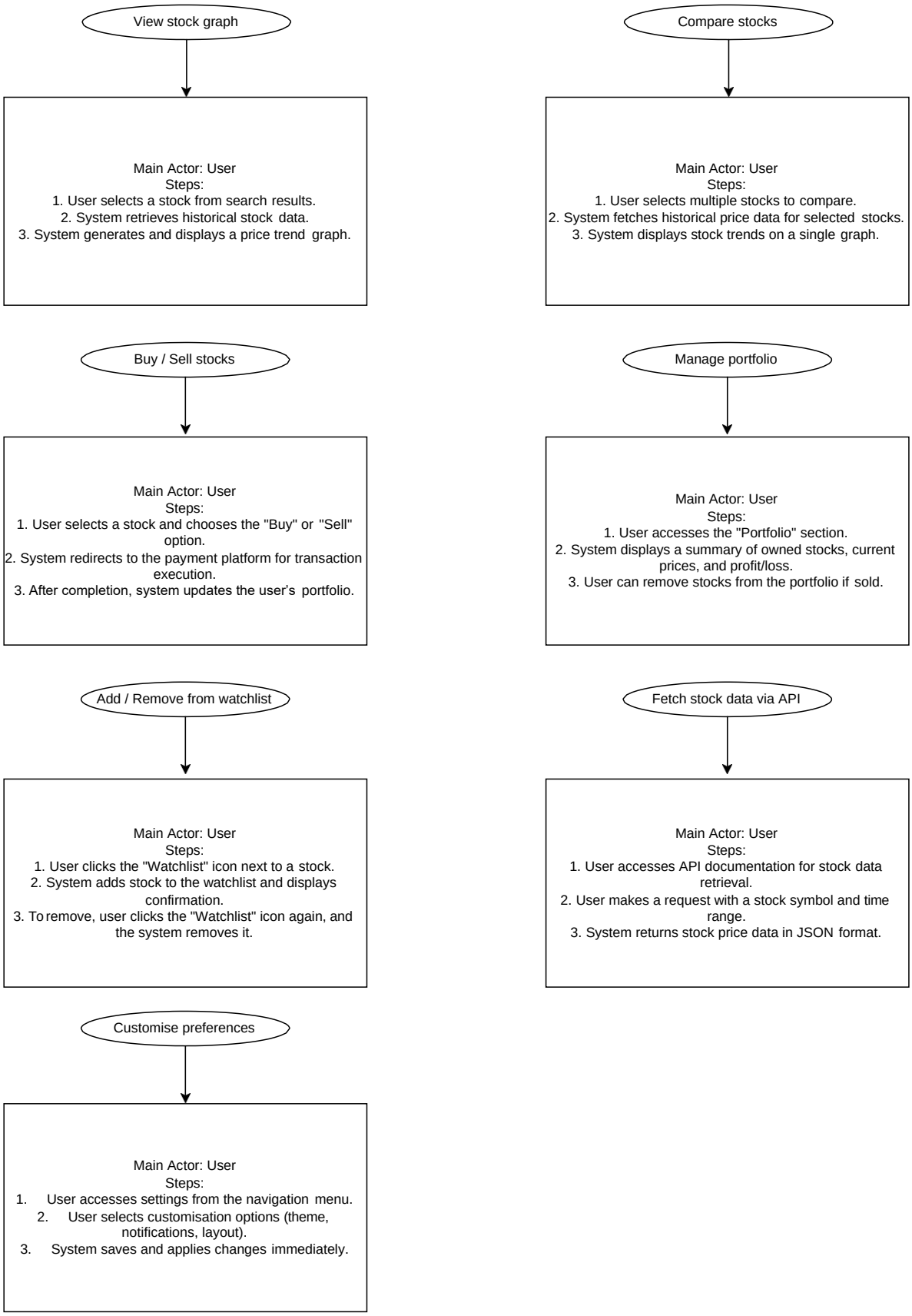


User Stories

- **Stock Price Retrieval:** As a user, I want to enter a stock symbol and date range to retrieve historical price data.
- **Data Persistence:** As a user, I want to save stock data for offline analysis.
- **Graphical Comparison:** As a user, I want to compare two stocks side by side.
- **AI Predictions:** As a user, I want AI-based trend predictions for future analysis.

➢ Scenario Details




View stock graph
Main Actor: User
Steps:
1. User selects a stock from search results.
2. System retrieves historical stock data.
3. System generates and displays a price trend graph.
Compare stocks
Main Actor: User
Steps:
1. User selects multiple stocks to compare.
2. System fetches historical price data for selected stocks.
3. System displays stock trends on a single graph.
Buy / Sell stocks
Main Actor: User
Steps:
1. User selects a stock and chooses the "Buy" or "Sell" option.
2. System redirects to the payment platform for transaction execution.
3. After completion, system updates the user's portfolio.
Manage portfolio
Main Actor: User
Steps:
1. User accesses the "Portfolio" section.
2. System displays a summary of owned stocks, current prices, and profit/loss.
3. User can remove stocks from the portfolio if sold.
Add / Remove from watchlist
Main Actor: User
Steps:
1. User clicks the "Watchlist" icon next to a stock.
2. System adds stock to the watchlist and displays confirmation.
3. To remove, user clicks the "Watchlist" icon again, and the system removes it.
Fetch stock data via API
Main Actor: User
Steps:
1. User accesses API documentation for stock data retrieval.
2. User makes a request with a stock symbol and time range.
3. System returns stock price data in JSON format.
Customise preferences
Main Actor: User
Steps:
1. User accesses settings from the navigation menu.
2. User selects customisation options (theme, notifications, layout).
3. System saves and applies changes immediately.

➢ Personas

Intermediate User - Financial Expert

Name: Aisha Malik
Occupation: Financial Analyst

"I don’t need fancy features; I just need accurate data and clear charts to make quick decisions."

User Information	
User Story	As a financial analyst, Aisha wants a straightforward way to track stock prices and compare trends so she can make informed investment recommendations for her clients.
User Scenario	Aisha is preparing a quarterly investment report. She needs to compare the stock performance of two competitors over the past year. She logs into the application, enters the stock symbols, selects the date range, and generates a comparison chart. After analysing the data, she exports the report to present to her team.

Demographic information	
Age	32
Gender	Female
Location	London, UK
Tech Skills:	Moderate (Excel, financial software, but limited coding experience)
Financial Knowledge:	High (stock market analysis, portfolio management)

Needs and motivations	
Motivations	Frustrations
• Making data-driven investment decisions efficiently • Saving time with quick and clear stock price comparison • Ensuring accurate financial reporting • Staying ahead of market trends	• Finds some financial tools too overly complex and expensive • Gets frustrated when data isn’t up to date or when there’s limited historical data • Prefers clear visualizations of data over raw numbers
Goals	Environment
• Easily track and compare stock prices • Export high quality reports • Access historical data in an easy-to-read format	• Works in an office with multiple screens • Uses a Windows laptop and sometimes a tablet • Often on the move and needs access to stock data via phone

Experienced Software Engineer – FinTech Developer

Name: James Carter
Occupation: Senior Software Engineer (FinTech)

"If your app doesn't have a solid API, I'm not interested."

User Information

User Story	As a software engineer, James wants an application with a well-documented API and customizable features so he can integrate stock data into his own financial tools.
User Scenario	James is developing a personal finance dashboard and needs real-time stock data to track, buy, and sell shares programmatically. He checks the application's documentation, writes a quick script to retrieve stock prices, and integrates the API into his system. Within minutes, his dashboard is fetching livestock data, executing automated trades based on predefined conditions, and analysing market trends.

Demographic information

Age	27
Gender	Male
Location	New York, USA
Tech Skills:	High (Java, API development, cloud computing)
Financial Knowledge:	Medium (understands stock markets but isn't a professional trader)

Needs and motivations

Motivations	Frustrations
• Wants to work with clean architecture and efficient APIs • Prefers open-source or extensible platforms • Aims to streamline stock trading through automation	• Hates dealing with poorly documented APIs • Finds hardcoded limitations restrictive • Prefers a fast and efficient backend over fancy UI
Goals	**Environment**
• Integrate stock data seamlessly into his financial software • Customize the application to suit his needs • Ensure the system is efficient and scalable • Automate stock buying and selling with minimal manual effort	• Works on a MacBook Pro with Linux virtual machines • Uses VS Code, Postman, and GitHub everyday • Frequently switches between local and cloud environments

Skilled User – Experience in Finance & Tech

Name: Omar Khan
Occupation: Investment Manager with IT Background

"If I can't access my data quickly, I lose money. Simple."

User Information

User Story	As an investment manager, Omar wants an application that provides automated stock alerts and customizable dashboards so he can efficiently track multiple investments.
User Scenario	Omar is managing his clients' portfolios. He sets automated alerts for specific stocks. When a company's stock drops by 5%, he gets notified instantly, reviews the comparison chart, and makes a decision before calling his clients.

Demographic information

Age	40
Gender	Male
Location	Dubai, UAE
Tech Skills:	Moderate (SQL, Python for finance, cloud-based tools)
Financial Knowledge:	High (portfolio management, risk assessment)

Needs and motivations

Motivations	Frustrations
• Keeping clients' investments profitable • Having real-time insights into stock market movements • Minimizing manual effort while maximizing accuracy	• Finds dashboards cluttered with unnecessary info • Gets frustrated with slow-loading financial tools • Hates missing out on critical market movements due to system lag and lack of alerts
Goals	**Environment**
• Monitor multiple stocks efficiently • Get real-time alerts for key stock movements • Use AI-based insights for smarter investment decisions	• Works in a corporate office but travels frequently • Uses a high-end Windows PC and a tablet for meetings • Checks stock data on his phone regularly

Architectural Design

StockShark incorporates a blend of architectural paradigms tailored to optimize maintainability, modularity and a clear separation of concerns. This section details each architectural choice and its application within the StockShark system.

Simple Architecture

Application: Simple Architecture is utilized in StockShark for straightforward and non-complex system components where high scalability and complex transactions are not primary concerns. This architecture is predominantly applied to user session management and authentication processes, such as the **Login/Logout** functionality. This choice ensures that these components are easy to implement and maintain, without the overhead of more complex architectural frameworks.

Clean Architecture

Application: The principles of Clean Architecture are foundational to StockShark, particularly in ensuring that all system dependencies are directed inwards. This approach is manifest in:

- **Core Entities** such as **User** and **Portfolio**, which lie at the heart of the system.
- **Use Cases/Business Rules Layer** with components like the **Stock Comparison** and **Notification System**, which encapsulate specific business logic independent of external interfaces.
- **Interface Adapters** such as the **Data Aggregator**, which act as intermediaries translating data between the database, external APIs, and internal business rules.

This layered approach not only supports modularity and independence in development and testing but also ensures that the system remains adaptable to changes in external APIs or database technologies.

Service-Oriented Architecture (SOA)

Application: StockShark leverages SOA to decompose the application into discrete services that can be independently developed, deployed, and maintained. Key components such as the **AI Analyzer**, **Data Aggregator**, and **Database** are developed as separate services, each responsible for distinct functionalities within the system. SOA facilitates the flexible integration of these services, promotes reusability, and enhances the system's scalability by allowing services to be scaled independently based on demand.

Model-View-ViewModel (MVVM)

Application: Within the UI layer, StockShark employs the MVVM architectural pattern, particularly effective in the **Graphical Component Display/View**. This pattern supports robust data binding between the UI elements and the underlying data models, simplifying the management of dynamic content such as real-time stock data visualization and user-specific configurations. MVVM aids in separating the presentation logic from business logic, thereby facilitating a more manageable and testable UI architecture.

Model-View-Controller (MVC)

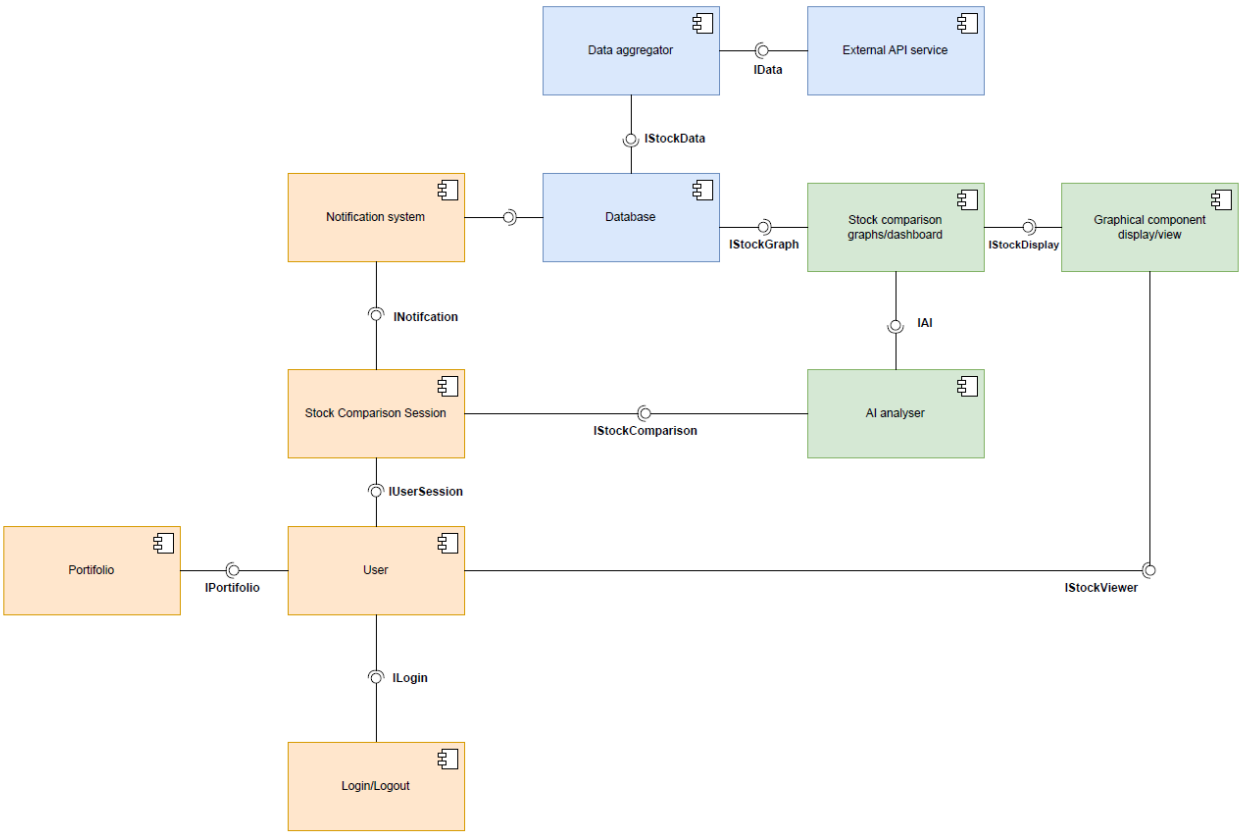
Application: MVC is used to structure the interaction and data flow between the **Stock Comparison Graphs/Dashboard** and the underlying models. In this setup:

- **Models** manage the data and logic of the application.
- **Views** are responsible for outputting representation to the user.
- **Controllers** handle input and convert it to commands for the model or view.

This separation helps manage complex data interactions and user interfaces, making the system more understandable and easier to debug.

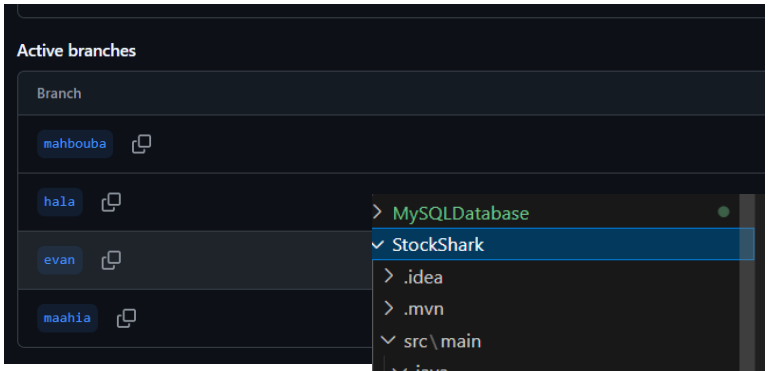
Component and Use Case Diagrams

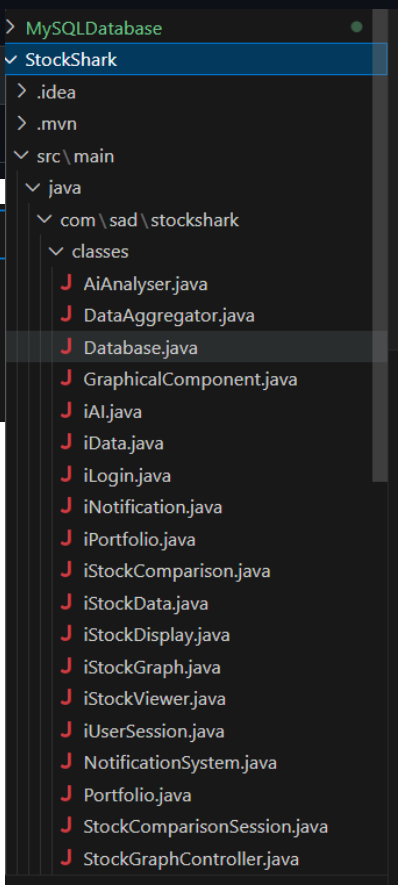
Component diagrams illustrate the structure of the system by showing its components and their relationships, which is crucial for understanding how different parts of StockShark interact, such as data retrieval, storage, and user interfaces. Use case diagrams map user interactions with the system, helping define user requirements and system functionality.

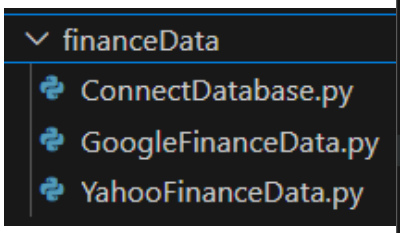


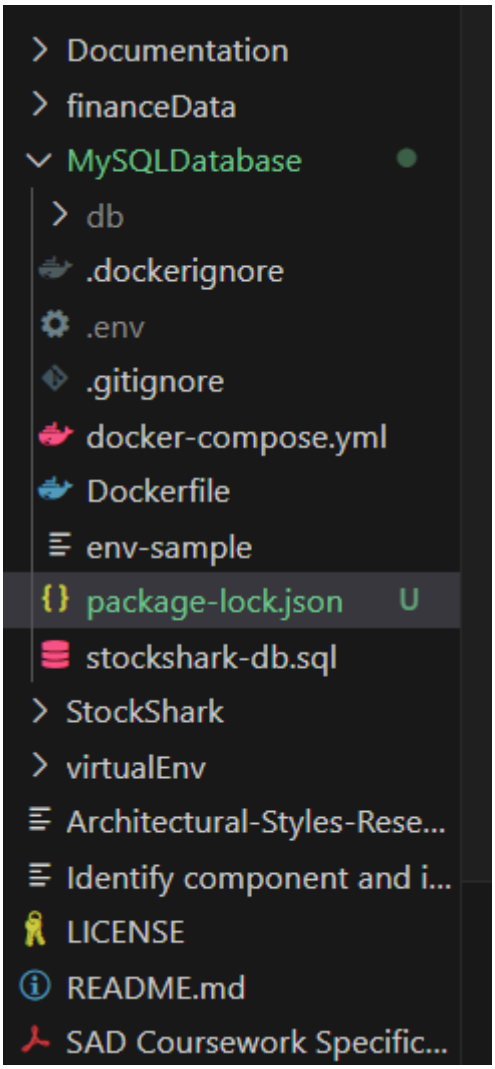
Implementation

We set up the MySQL database with necessary tables, installed required libraries (mysql-connector-python and yfinance), and developed Java classes for data retrieval, storage, and visualization. All code was committed to GitHub, with each team member contributing through individual branches.



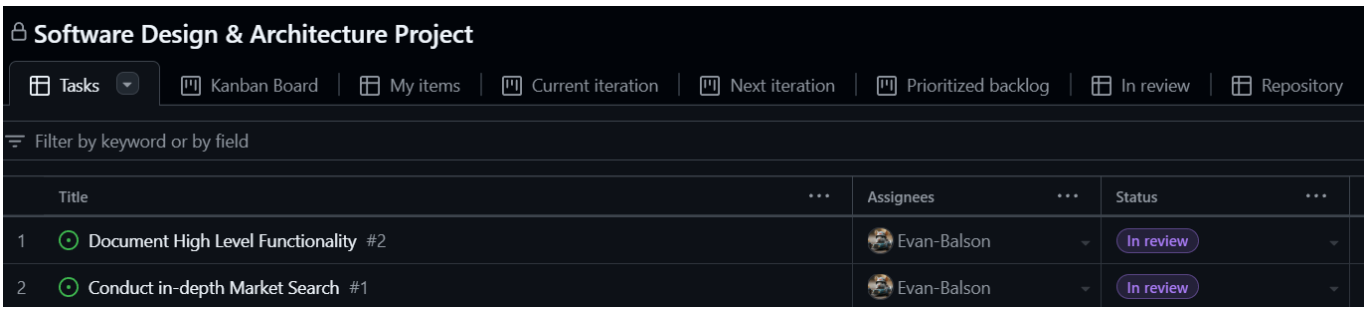






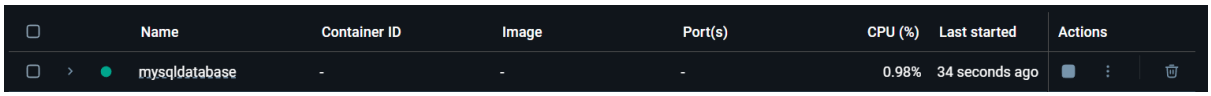
Team Management

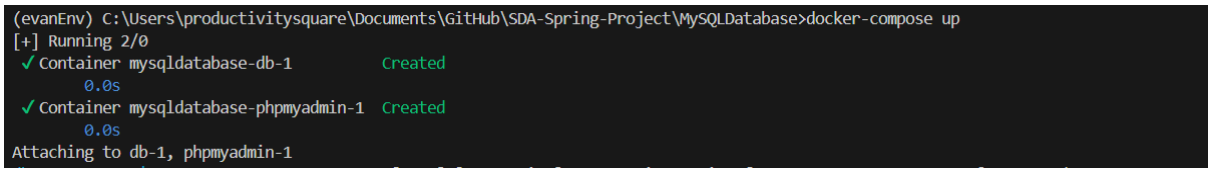
GitHub was used for version control and task tracking, ensuring seamless collaboration. Regular meetings facilitated progress reviews, and a code of conduct was established for efficient teamwork.



Challenges and Solutions

Setting up development environments for all team members was a challenge, as each person had different systems and configurations. We overcame this by implementing virtual environments within our coding IDEs, leading to a successful setup for MySQL, Java, and required libraries.





- Another challenge was ensuring collaborative development with controlled versions and commits, which led us to adopt GitHub for version control.
- Regular team meetings and task assignments helped resolve any technical issues.
- We also faced limitations with API access, as many direct APIs have been discontinued, forcing us to rely on third-party suppliers. This created challenges with limited request tokens, requiring us to plan for acquiring more tokens or switching suppliers. Our application must be flexible enough to handle these changes seamlessly.

Code of Conduct

Our team established a code of conduct emphasizing respect, collaboration, and clear communication. Each member agreed to adhere to deadlines, commit code regularly, and support each other throughout the project.

Code of Conduct

Team Guidelines

Respect and Inclusion

- Respect and value each team member's ideas and contributions.
- Foster an inclusive and collaborative environment.

Communication

- Communicate clearly and concisely.
- Provide updates on your task progress, blockers, or delays.

Accountability

- Actively participate in meetings and provide feedback on your tasks.
- If no progress is made, explain **why** and how you plan to get back on track.
- Falling behind significantly without a valid reason will be reported.

Meeting Etiquette

- Be punctual and prepared for meetings.
- **If you're late, treats for the team are on you! -> chocolates for Mahbouba please!**
- **Absences:**
  - Missing **2 meetings** will result in a warning.

Meeting Etiquette

- Be punctual and prepared for meetings.
- **If you're late, treats for the team are on you! -> chocolates for Mahbouba please!**
- **Absences:**
  - Missing **2 meetings** will result in a warning.
  - Missing **3 meetings** will be escalated and reported.

Disputes

- If dissatisfied with a decision (e.g., design features), provide proper reasoning.
- Present your argument to the team with supporting research for discussion.

Feedback and Conflict Resolution

- Provide constructive, actionable feedback.
- Escalate unresolved conflicts to **Mahbouba (Scrum Master)** or **Evan (Product Manager)**.

References

[1] GeeksforGeeks, “Architectural Design – Software Engineering,” Sep. 27, 2019. [Online]. Available: https://www.geeksforgeeks.org/architectural-design-software-engineering/. [Accessed: Feb. 7, 2025].

[2] GeeksforGeeks, “Types of Software Architecture Patterns,” Jun. 20, 2024. [Online]. Available: https://www.geeksforgeeks.org/types-of-software-architecture-patterns/. [Accessed: Feb. 7, 2025].

[3] Yahoo Finance API Documentation. [Online]. Available: https://finance.yahoo.com. [Accessed: Feb. 7, 2025].

[4] Google Finance API Documentation. [Online]. Available: https://www.searchapi.io/docs/google-finance. [Accessed: Feb. 7, 2025].

[5] R. C. Martin, "Clean Architecture: A Craftsman's Guide to Software Structure and Design," Pearson Education, 2017.

[6] Python Software Foundation, “yfinance: Yahoo Finance Library,” [Online]. Available: https://pypi.org/project/yfinance/. [Accessed: Feb. 7, 2025]. [7] Oracle, “MySQL Documentation,” [Online]. Available: https://dev.mysql.com/doc/. [Accessed: Feb. 7, 2025].

Environment Variables:

Required libraries:

- pip install mysql-connector-python
- pip install yfinance

GitHub | Working with multiple developers on same project