



Software Architecture & Design – Sprint 3

Team Members:

Evan Balson – BAL18466416

Maahia Rahman – RAH23614335

Hala Bakhtiar – BAK23592238

Mahboubia Rezaei – REZ23579670

Roehampton University

Course Number: CMP020N207S

Lecturer: Dr Shekoufeh Kollahdouz Rahimi



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Overview	4
Architecture & Design:	4
Presentation Layer:	4
Controller Layer:	5
Business Logic Layer:	5
Data Access Layer:	5
Integration Layer:	5
5 Tier Architecture Component Diagram:	6
How is this achieved?	17
The Database:	17
Key Tables:	18
Deployment:	18
Environment Details:	18
Building The Architecture	19
Business Concept Model	19
Use Case Model	20
System Interfaces	21
Business Type Model	24
Initial System Architecture	25
Unit Testing	26
User Testing	26
Project Management	28
Code-Related Deliverables	28
Implementation of Clean Architecture Principles	28
Component and Interface Implementation	29



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Team Management	29
Team Coordination.....	29
Process	30
GitHub Project Board:	30
Team Code of Conduct	31
Team's Meetings	32
Screenshots of the application.....	42
References:	52



Overview

StockShark is a web-based application that empowers users to view, compare, and manage stock portfolios (or watchlists). The application integrates with external market data APIs (e.g., Alpha Vantage) to retrieve real-time and historical stock data. It provides features such as:

- **User Registration and Authentication:** New users can register, log in, and manage their profile.
- **Market Overview and Stock Comparison:** Users can view market data, compare stocks over custom date ranges, and see graphical representations using Chart.js.
- **Watchlist/Portfolio Management:** Users can create and manage portfolios (watchlists) where they add their favorite or frequently traded stocks.
- **Data Persistence:** All user and portfolio data are persisted in a MySQL database using a robust schema with proper referential integrity.
- **Interactive, Modern UI:** A dark-themed interface with responsive design and interactive charts for a smooth user experience.

Architecture & Design:

The StockShark project follows a 5-tier architecture with the following layers:

Presentation Layer:

JSP Pages and HTML/CSS/JavaScript: User interface is rendered by JSPs that incorporate JSTL, custom EL expressions, and utilize CSS for styling. Client-side interactivity is provided via JavaScript and Chart.js for dynamic charting.



Controller Layer:

Servlets: Java servlets handle HTTP requests and responses. Notable servlets include LoginServlet, RegistrationServlet, CompareStocksServlet, PortfolioServlet, and WatchlistServlet. They are mapped using annotations (e.g., @WebServlet). They manage request/response cycles, encapsulate user interactions and forward data to the view fulfilling user stories such as viewing personalized market data and portfolios.

Business Logic Layer:

The business logic is organized into compound components like User_Management_Compound and Data_Handling_Component. This abstraction facilitates meeting complex user interactions (e.g., adding portfolios, updating charts) while keeping the controllers lean.

Data Access Layer:

Database Access Classes: Classes like PortfolioDAO encapsulate database access. They run SQL queries using JDBC with proper resource management (try-with-resources) and manage transactions. Instead of a DAO, our project uses a Database Client class to handle all persistence operations. This ensures that queries and updates such as retrieving portfolio details by joining portfolios with 'portfolio items' are centralized. This design was chosen to maintain flexibility and consistency with our overall application workflow.

Integration Layer:

The application integrates with external APIs (such as Alpha Vantage) to fetch real-time stock data. User stories requiring updated market information are supported through API calls managed via the Database Client. The API responses (JSON) are parsed and converted into domain objects (like StockData).



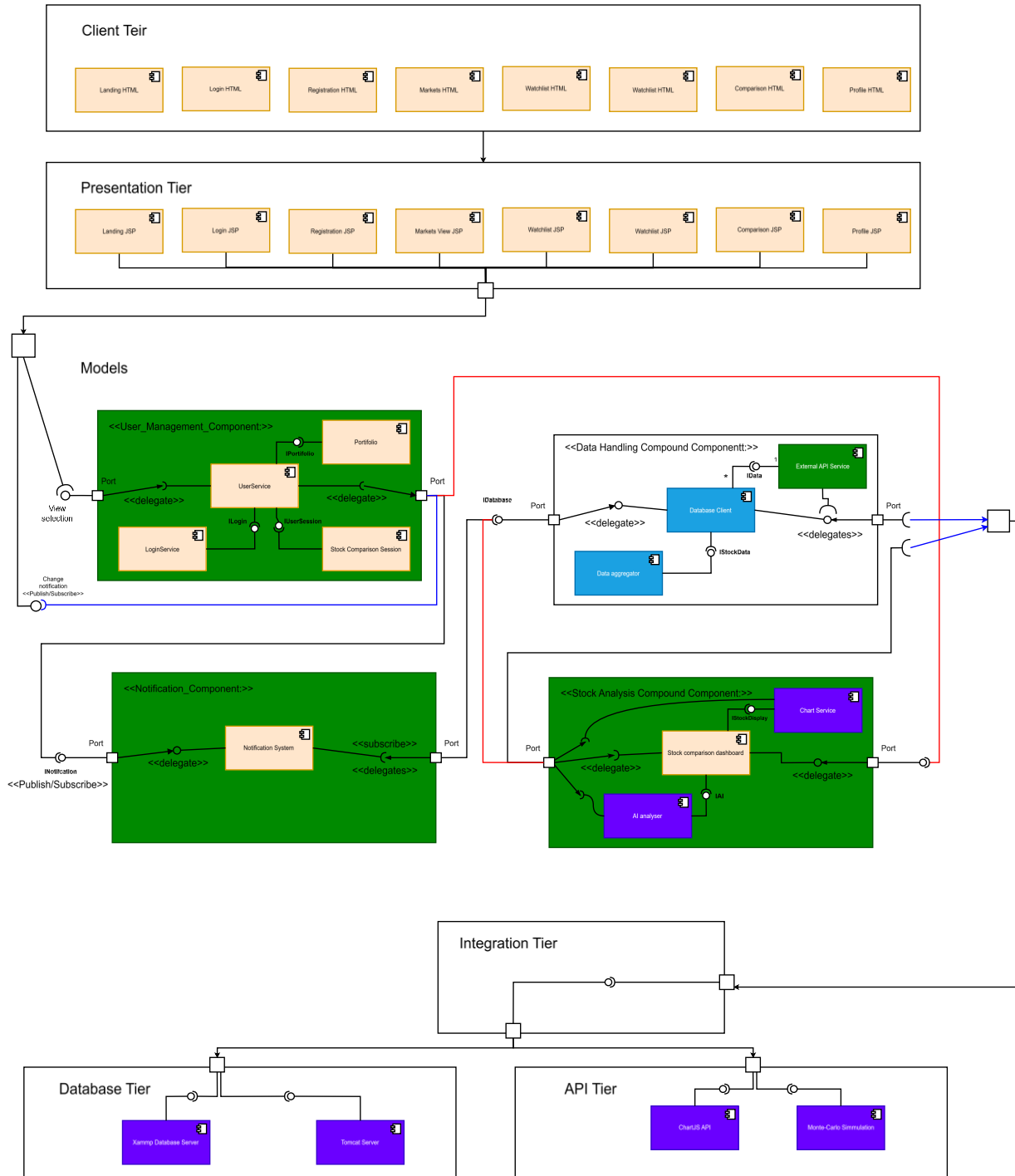
5 Tier Architecture Component Diagram:





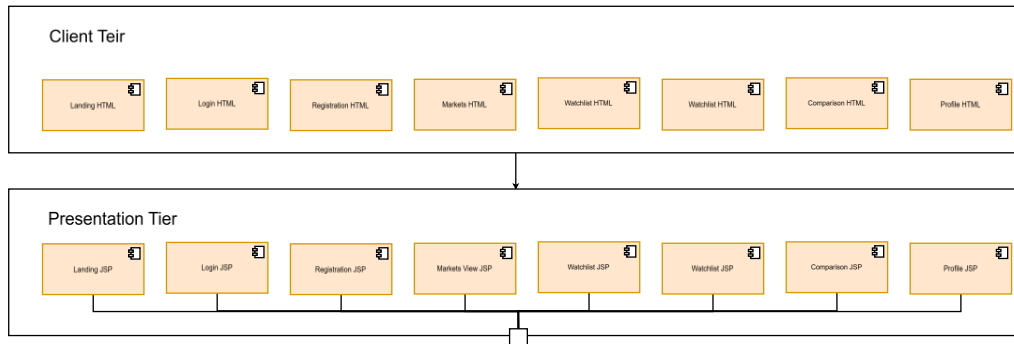
Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN





This solution uses the 5-tiered architecture.



Tier 1 – Client Tier

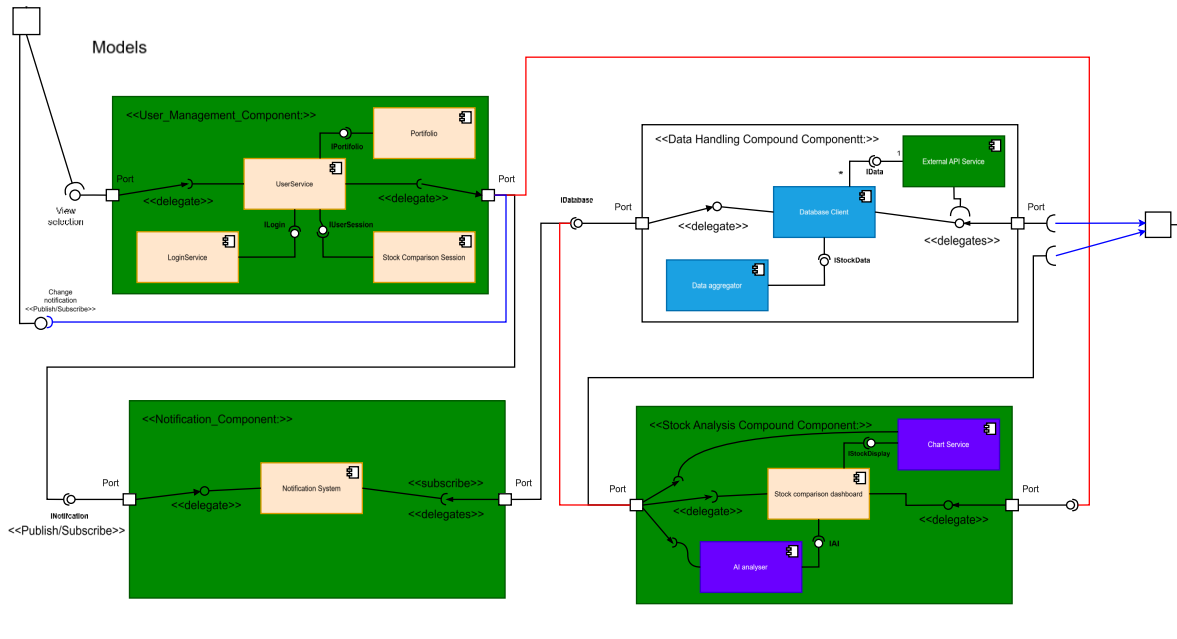
To provide a complete design, we include a client tier (the first tier), which contains the UI elements that users employ to interact with the rest of the system.

Tier 2 – Presentation Tier

The first tier then interacts with the presentation tier, which contains our servlets. These servlets decide which of the models should handle each user interaction. They do so by connecting to a dedicated port in the user components that serves as the main entry point to the **User_management_Component**. This component is responsible for handling user-facing requests such as login, portfolio actions, and starting stock-comparison sessions.

Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN



Tier 3 – Business Logic Tier/ Models

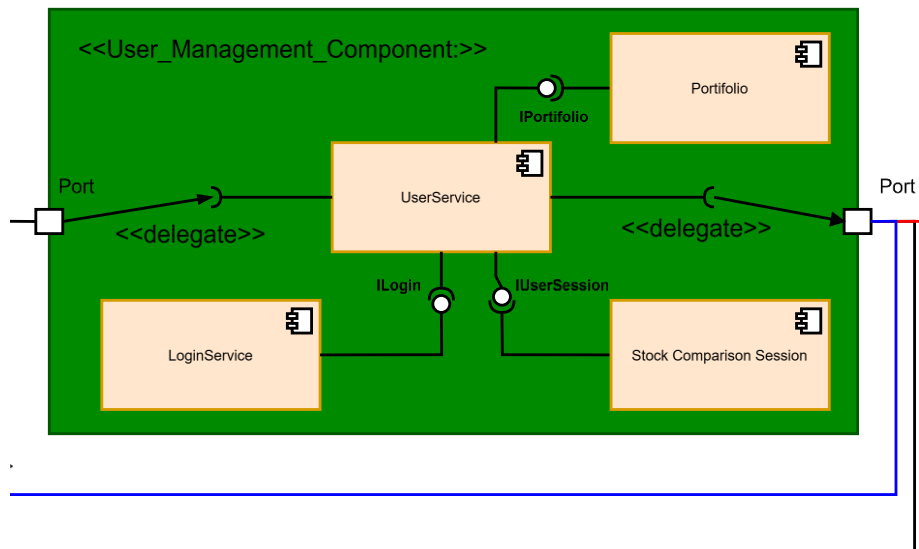
The business logic is encapsulated in four client Compound components:

- **User_Management_Component** – handles front-facing user requests that come from the servlets.
- **Data_Handling_Compound_Component** – manages all incoming and outgoing data packets and how they are manipulated.
- **Stock_Analysis_Compound_Component** – provides custom-coded charts and refined mathematical comparison calculations.
- **Notification_Compound_Component** – sends notifications to the user and listens for changes within the Data_Handling component



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN



This compound component has four sub-components:

- **Portfolio:** responsible for creating Portfolio objects for the user and contains methods for accessing and storing them.
- **LoginService:** responsible for providing methods that can be used to authenticate and log users in and out of sessions on the platform.
- **StockComparison:** responsible for initiating and terminating comparison sessions.
- **UserService:** responsible for hosting the ports that other external components can connect to and also to communicate with other internal components.

The UserService has three interfaces:

- **IPortfolio:** exposes functionality that can be used to manage a portfolio, such as adding, removing, and updating.
- **ILogin:** exposes functionality related to login, logout, and registration.
- **IUserSession:** exposes functionality that can be used to start or stop stock-comparison sessions.



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

<<Interface>>

IPortfolio

- + Map<String, List<String>> getPortfolio(String portfolioID);
- + Boolean updatePortfolio(String email, String symbolA, String SymbolB);
- + Boolean removePortfolio(String user, Portfolio portfolio);

getPortfolio: returns a dictionary whose key is the PortfolioID and whose value is a list of Stock Symbols for the specified portfolio.

Update/Remove Portfolio: returns **true** if insertion into the database succeeds, or **false** if it fails.

<<Interface>>

- + Boolean registerUser(String username,String email, String password, Data_Handling_Component dh);
- + Boolean loginUser(String email, String password, Data_Handling_Component dh);
- + Boolean logoutUser(String email, Data_Handling_Component dh);

Login/Logout/Register: returns **true** if insertion into the database succeeds, or **false** if it fails.

<<Interface>>

IUserSession

- + void startSession(List<String> stockSymbols);
- + void endSession();
- + boolean isSessionActive(boolean session);



Stock Shark

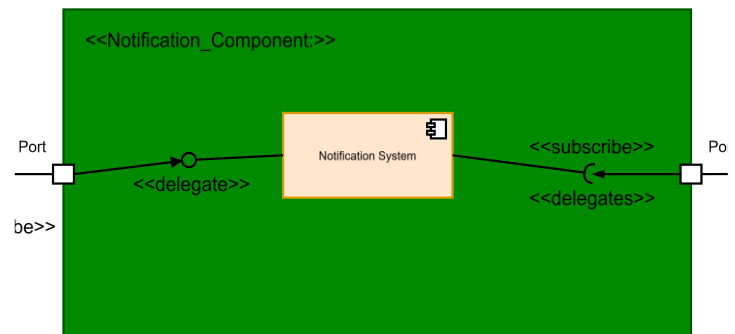
THE SOFTWARE ARCHITECTURE & DESIGN

IsSessionActive: returns **true** if the user is currently in an active comparison session, or **false** otherwise.

he **User_Management_Compound_Component** communicates directly with two other compound components:

- Notification_Compound_Component
- Stock_Analysis_Compound_Component.

The **Notification_Compound_Component** offers a single provided interface, **iNotification**, which exposes publish/subscribe functionality so users are informed of the latest system changes. When a user interacts, they can update the notification system, enabling it to track any additional data to which it is subscribed.



<<Interface>> INotification
+ void sendNotification(String message); + List<String> viewNotifications(String userID);

Send Notifications: collects and sends new symbols or portfolio updates to the system so that it will now listen for updates within the database.

View Notifications: returns a list of symbols reflecting the latest system changes that have occurred within the user's saved portfolio items.

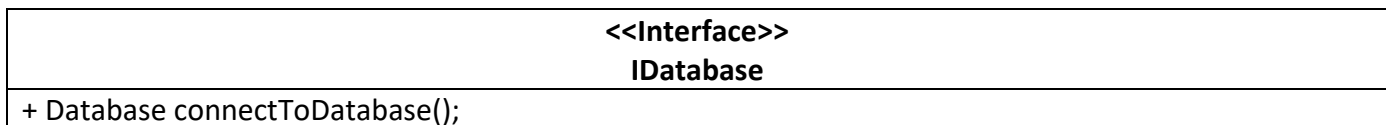


Stock Shark

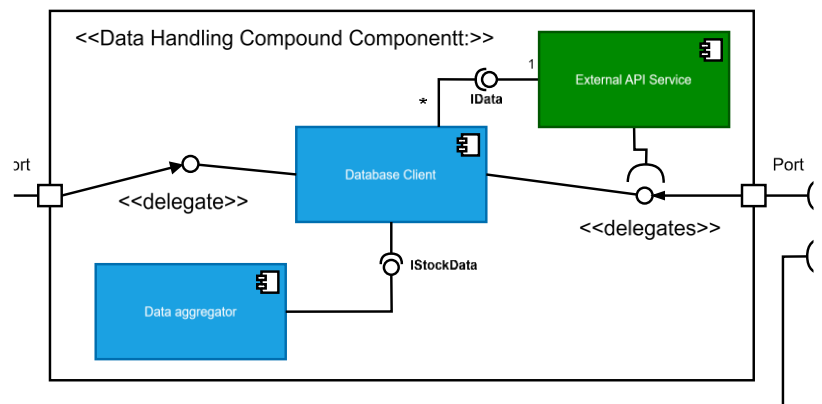
THE SOFTWARE ARCHITECTURE & DESIGN

This compound component contains a single sub-component, **Notification System**. It encapsulates the core functionality for updating notifications in — and retrieving them from — the database. The sub-component exposes two ports for external communication:

- **Notification_ComponentPort:** provides notification functionality to the User_Management_Compound_Component.
- **getDataHandlingComponentPort:** connects to the data-handling compound components that expose database-reading functionality through the required interface IDatabase.



Once the Data Handling Component Port is invoked, it exposes the functionality of the Data Handling Compound Component by connecting to the delegated port `getDataHandlingComponentPort` hosted within the `DatabaseClient` subcomponent.



The Database Client has two required interfaces:

- **iData:** interface provided by the External API Service module; exposes functionality used to communicate with APIs outside our system. It is defined as a one-to-many relationship because multiple methods for different API providers can be made available to our Database Client.
- **iStockData:** interface provided by the Data Aggregator; exposes functionality that translates data pulled from our External API Service into a form usable by our application.



<<Interface>> IData	
+ JSONObject	getStockData(String symbol) throws IOException, InterruptedException;
+ JSONObject	getStockDataByDateRange(String symbol, String startDate, String endDate) throws IOException, InterruptedException;
+ JSONObject	getTrendingMarketData(String symbol) throws IOException, InterruptedException;

getStockData: returns stock data in JSON format for a defined date range if a date is not specified.

getStockDataByDateRange: returns stock data from AlphaVantage in JSON format for a specified date range placed by the user.

getTrendingMarketData: returns data from AlphaVantage for a defined set list of symbols.

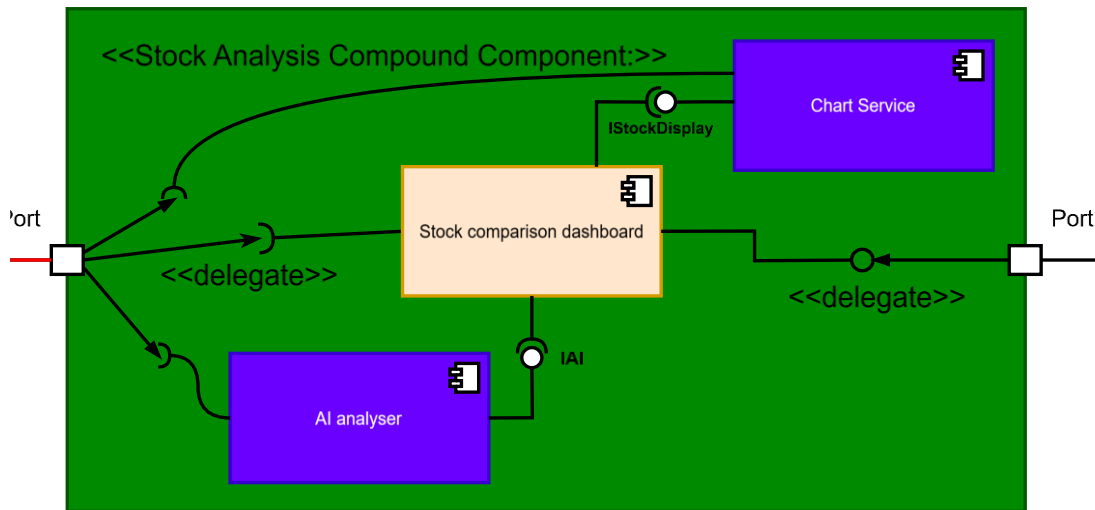
<<Interface>> iStockData	
+ List<StockData>	formatData_from_AlphaVantage(JSONObject jsonData);

formatData_from_AlphaVantage: receives the data from Alpha Vantage and returns a List of Stock objects that can be used by our system classes.

On the other hand, the UserService could take a different route to communicate with the database by using its required interface:

- **StockAnalysis:** exposes the functionality of the Stock Analysis Compound Component by connecting to the delegated port hosted within the Stock Comparison Dashboard.

Stock Shark THE SOFTWARE ARCHITECTURE & DESIGN



<<Interface>> iStockDisplay
<pre> public void GenerateAreaChart(List<String> stockData); public void GenerateAreaChartByDateRange(List<String> stockSymbols,String date1, String date2); </pre>

GenerateAreaChart: retrieves data from the database and formats it into an array of values suitable for plotting a graph.

GenerateAreaChartByDateRange: performs the same operation but filters the data so it returns values for a specific date.

The Stock Analysis Compound component had three sub components:

- **ChartService:** requires the Chart .js API and provides usable chart templates to the Stock Comparison Dashboard.
- **AI Analyser:** requires the OpenAI API and supplies summarized analysis and insights to the Stock Comparison Dashboard.



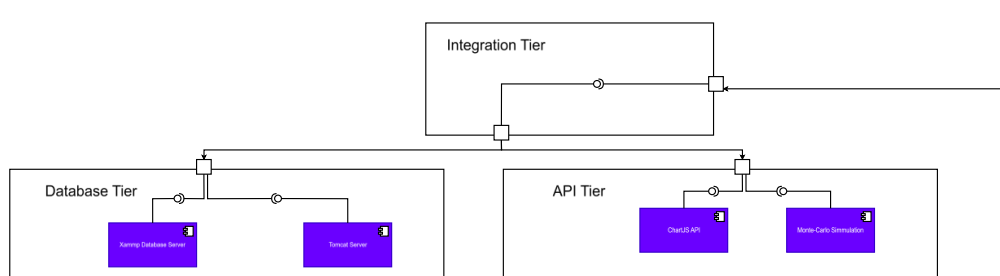
Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

- **StockComparison Dashboard:** encapsulates functionality for users to create diagram comparisons and has two interfaces:
 - **IStockDisplay** – already detailed.
 - **IAI** – this interface is not yet implemented.

The stock comparison dashboard has two delegated ports that is responsible for providing communication with the external components:

- **getDataHandlingComponentPort:** creates a required interface for connecting to the Data Handling Component.
- **getStockAnalysisComponentPort:** provides external components with access to the internal methods of the Stock Analysis Compound Component.



Tier 4 – Integration Tier: exposes a localhost port that allows the system to connect to the external services located in Tier 5.

Tier 5 – External Services Tier: contains all services external to the system, including:

- the **Database tier**, which consists of our Tomcat server and XAMPP, and
- the **API tier**, which consists of the AlphaVantage API and ChartJS API.



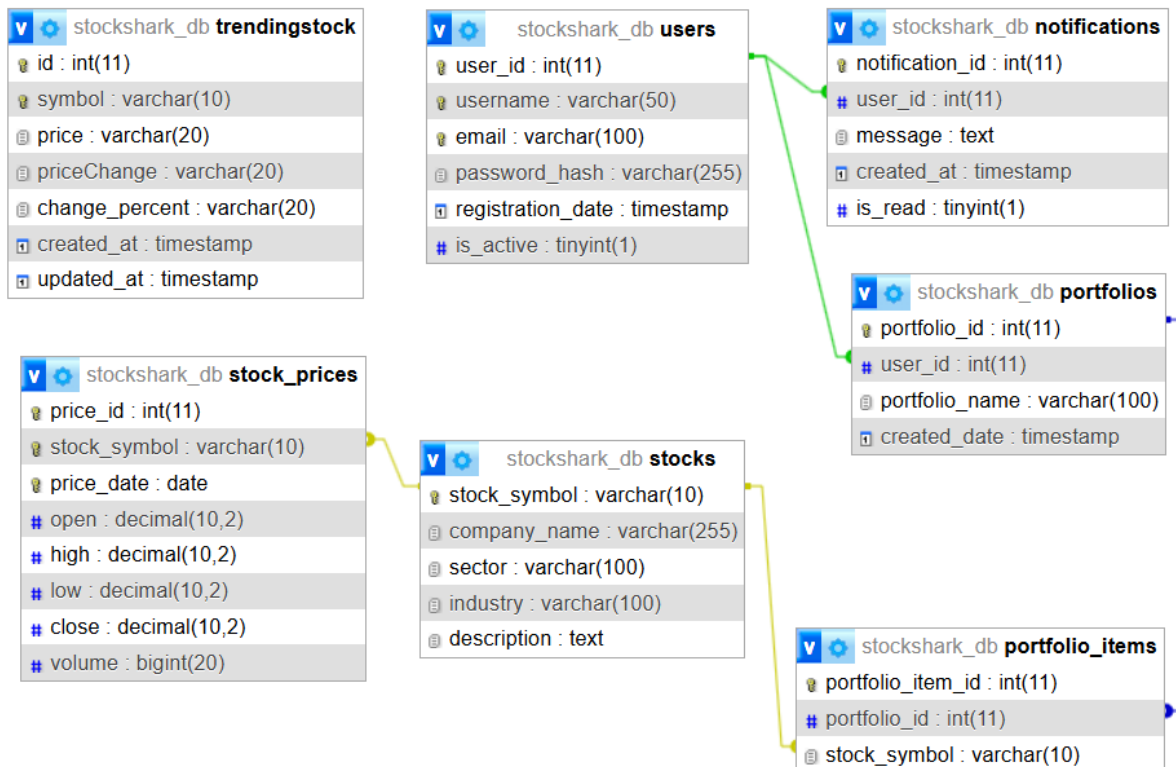
Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

How is this achieved?

The Database:

The database schema was designed to enforce data integrity and performance while supporting key functionalities like user management, stock data retrieval, and portfolio tracking. This design ensures that critical user data is stored securely and reliably, directly addressing user stories for privacy and accessibility.





Key Tables:

- **users:** Stores user credentials and profile details.
- **stocks:** Contains static information about stocks.
- **stock_prices:** Maintains historical (and potentially real-time) pricing information.
- **portfolios:** Saves user-defined portfolios or watchlists.
- **portfolio_items:** Associates specific stocks with a given portfolio.
- **notifications:** (Planned) Will store user notifications.

Deployment:

The deployment environment was selected to balance ease of development with production readiness. XAMPP is used as a local MySQL development server, and Apache Tomcat serves as the web/application server, ensuring broad compatibility and robust performance.

Environment Details:

- **Database Server:**
MySQL provided through XAMPP.
- **Application Server:**
Apache Tomcat.
- **Build Tools:**
Maven is used for building and packaging the application as a WAR file.
- **Configuration:**
Database configuration details (host, port, database name, user credentials) are managed centrally via the DatabaseConfig class.



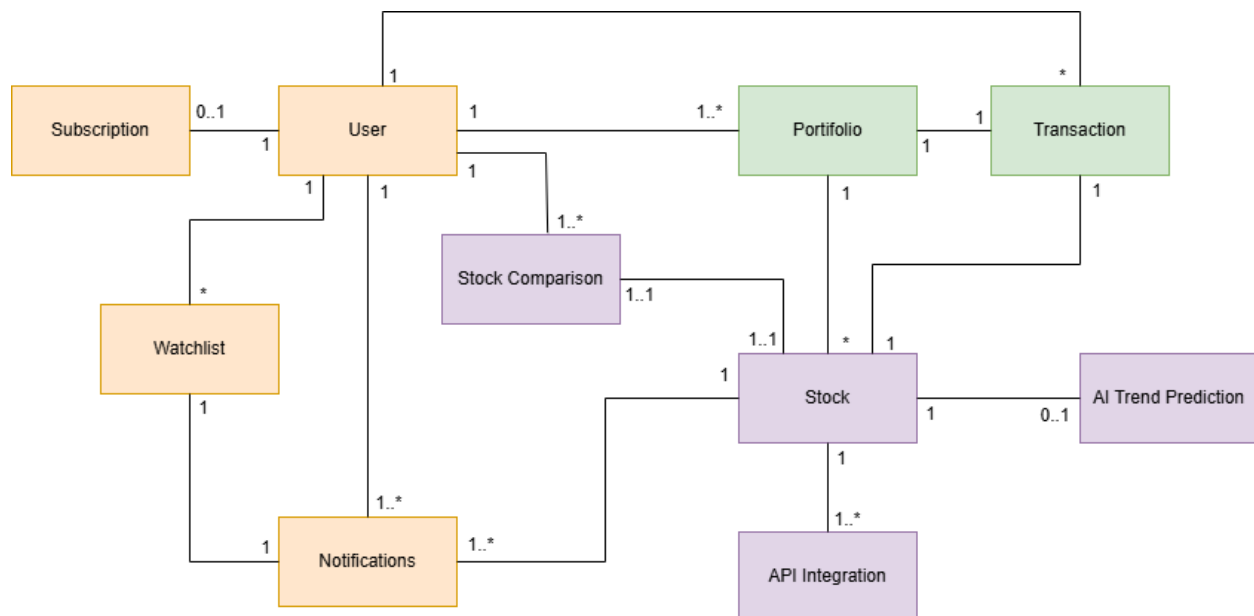
Building The Architecture

StockShark's architecture is built upon the core principles of Clean Architecture, which aims to create systems that are independent of frameworks, testable, and maintainable. This design follows the SOLID principles, making the system strong and easy to change.

- Single/Closed Principle (OCP): StockShark is designed to be open for extension but closed for modification. New features are added by introducing new code, rather than altering existing, stable code.
- Dependency Inversion Principle (DIP): High level modules, such as use cases that implement business logic, do not depend on low level modules, such as data access layers. Instead, low level modules depend on abstractions defined by high level modules.

By separating these layers, we create a system that's flexible, maintainable, and easy to understand. This makes it easier for our team to add new features, fix bugs, and adapt to changing requirements, ultimately providing you with a better StockShark experience.

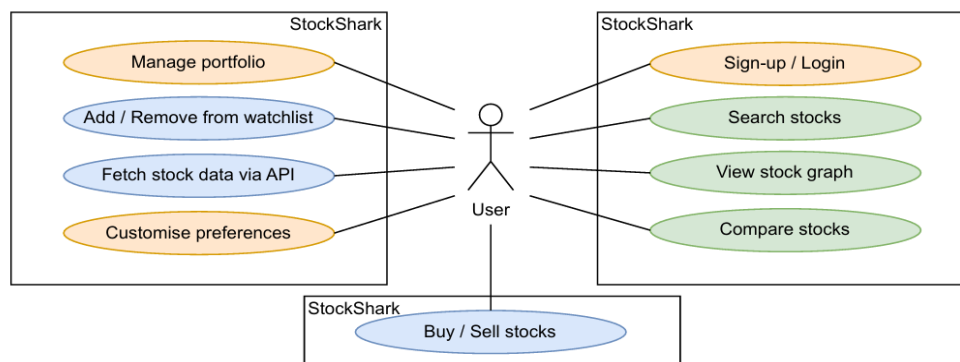
Business Concept Model





The Business Concept Model is structured to provide a user-centric, scalable, and modular system for managing stock investments and transactions. At its core, the User entity connects to essential functions like portfolios, transactions, subscriptions, watchlists, and notifications, ensuring seamless interaction with the platform. The Stock entity is deeply integrated with real-time market updates via API integration, AI trend predictions, and stock comparisons, allowing for data-driven decision-making. The model also enforces flexibility, allowing users to manage multiple portfolios and transactions while ensuring that each action is properly tracked. The relational structure supports future expansion, such as adding new AI models, additional investment tools, or automation, without disrupting existing functionalities. By balancing functionality and scalability, this design enables a robust financial system that caters to both beginner and advanced investors.

Use Case Model



The StockShark Use Case Diagram is structured to provide a clear and user-centric experience, dividing functionalities into three main categories: portfolio management, stock exploration, and trading transactions. Users can manage their investments by adding or removing stocks from their watchlist, customizing preferences, and fetching real-time stock data via API. The system also supports stock research through features like stock search, comparison, and historical graph visualization. Additionally, the buy/sell stocks functionality ensures seamless trading operations. The inclusion of sign-up/login enhances security and access control. This structured approach ensures scalability, allowing for future enhancements like AI-driven insights or automated trading while maintaining an intuitive and efficient user workflow.



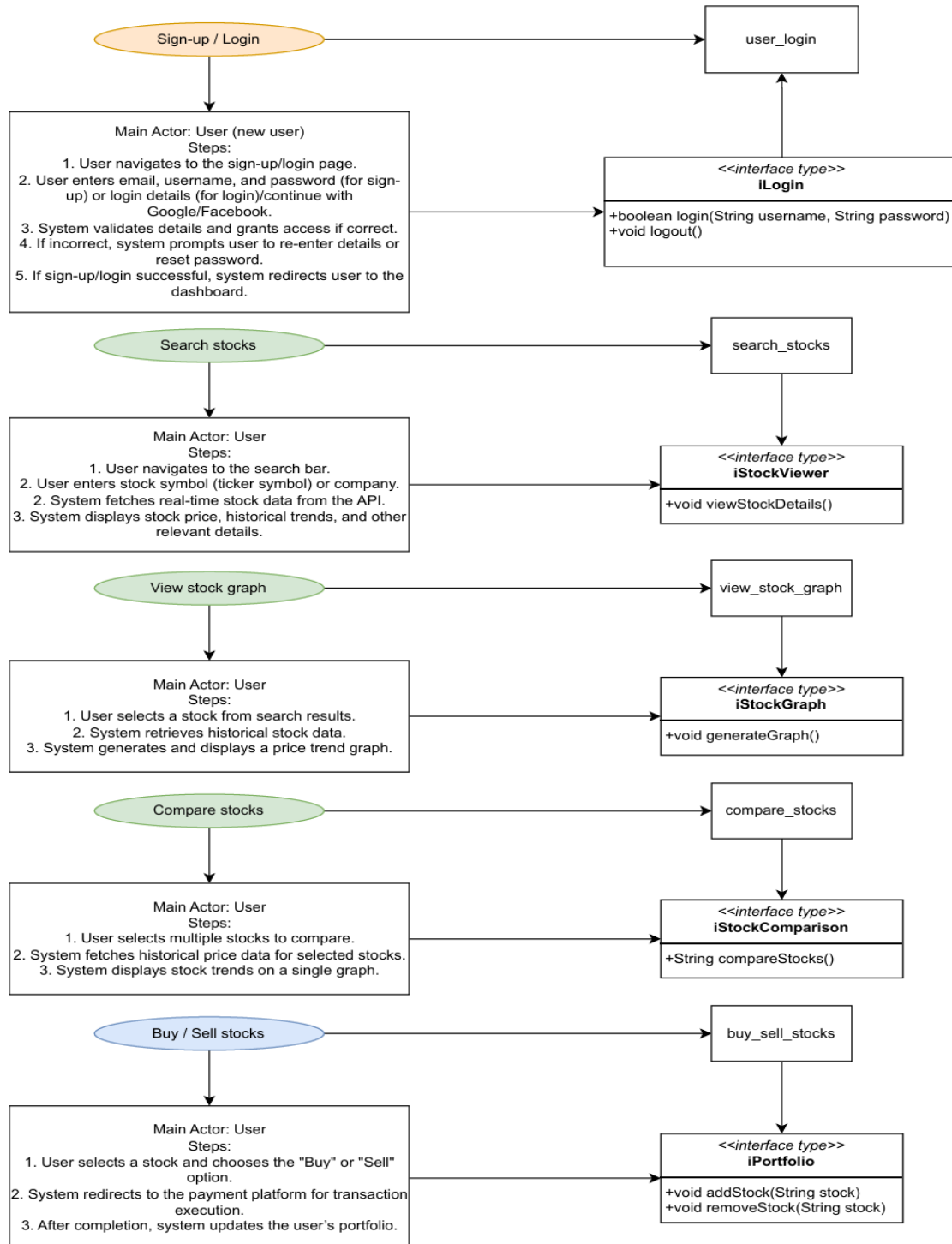
System Interfaces





Stock Shark

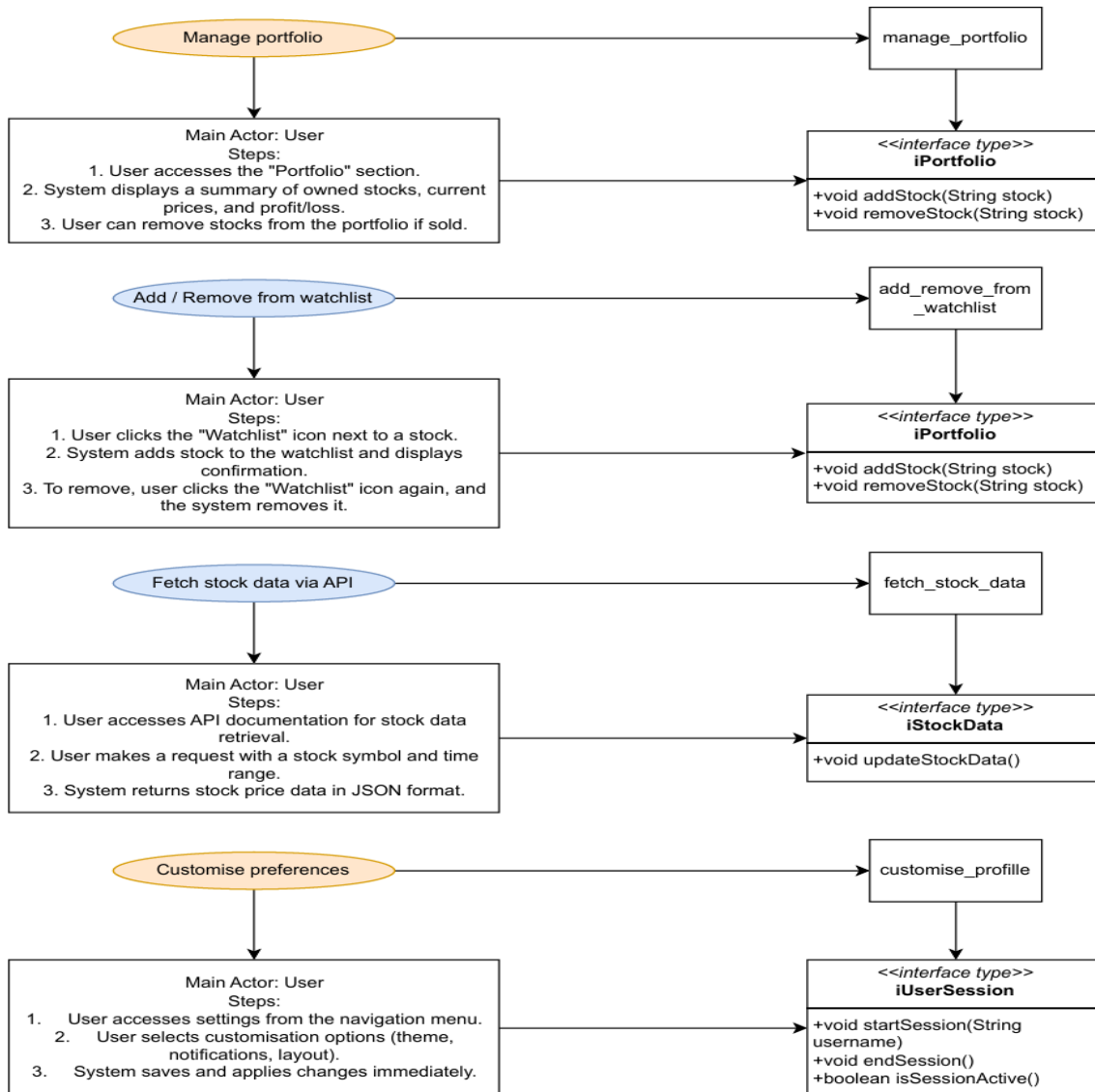
THE SOFTWARE ARCHITECTURE & DESIGN



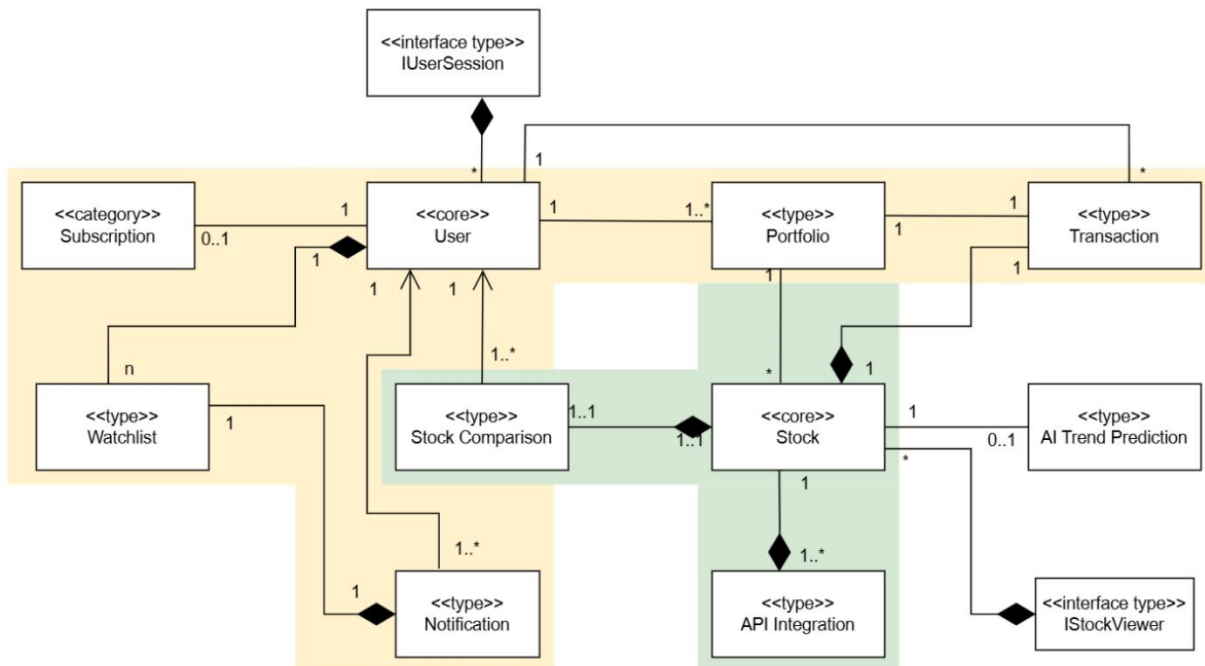


Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN



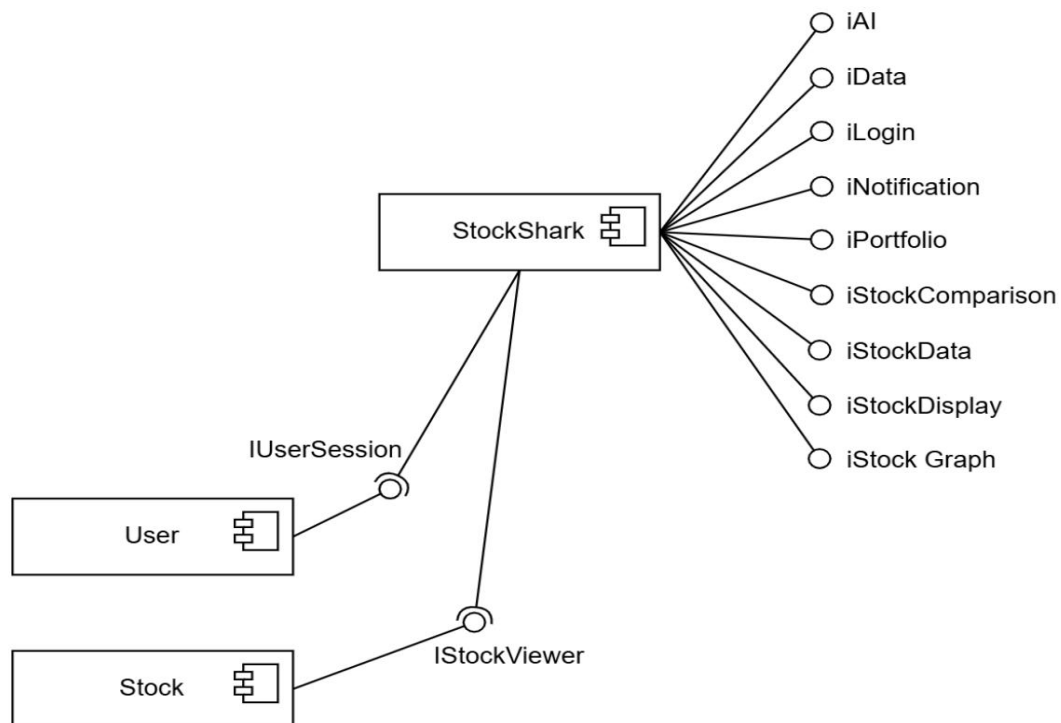
Business Type Model



The Business Type Model in the diagram is structured to represent a scalable and modular stock trading system with distinct core entities, functional types, and interfaces. At the center, the User entity acts as the primary connection point, interacting with subscriptions, portfolios, transactions, notifications, and watchlists. The model ensures that users can manage their investments efficiently while maintaining real-time stock tracking through API integrations. The Stock entity, another core component, connects to stock comparison, AI trend prediction, and API integration, ensuring data-driven decision-making. Additionally, interface types like IUserSession and IStockViewer support session management and stock visualization, enhancing the platform's usability. The composition relationships (diamonds) indicate ownership structures, ensuring that key elements like portfolios, watchlists, and notifications are user-specific, while stocks exist independently. This model is designed for flexibility and expansion, allowing future enhancements such as automated trading, advanced analytics, or AI-driven insights without disrupting the core architecture.



Initial System Architecture



The Initial System Architecture diagram represents the foundational structure of the StockShark platform, showcasing key system components and their interactions. At the core, StockShark serves as the central system, connecting various interfaces such as iAI, iData, iLogin, iNotification, iPortfolio, iStockComparison, iStockData, iStockDisplay, and iStockGraph, ensuring modularity and extendability. The User entity is linked through the IUserSession interface, allowing authentication and session management, ensuring secure access to the system. Meanwhile, the Stock entity interacts with StockShark via the IStockViewer interface, enabling real-time stock data retrieval and visualization. This architecture provides a clear separation of concerns, ensuring scalability, maintainability, and integration of advanced functionalities like AI-based trend analysis and stock comparison. The structured design allows for future enhancements, such as automated trading or additional analytics, while keeping the system flexible and user-friendly.



Unit Testing

What is Unit Testing?

Testing individual components (functions/methods) in isolation to ensure they behave as expected.

What did we do?

- **Written Functions:** Each function is a "unit" that can be tested. SQL for Testing Data: Used SQL queries to test data directly, bypassing the API with limited runs.
- **GitHub Pull Requests:** We worked on separate parts of the project and used pull requests for collaboration, code review, and merging.
- **Manual Testing:** Initially tested manually to verify functionality before committing changes.

Benefits we gained from this:

- **Catch Bugs Early:** Identify issues early without affecting the API.
- **Refactor Confidently:** Safely refactor code, knowing unit tests will ensure functionality remains intact.
- **Collaboration and Code Review:** Pull requests allowed for efficient collaboration and peer reviews.
- **Automated Checks:** SQL-based tests and unit tests ensure data integrity without hitting the API.

User Testing

What is User Testing?

Testing the application from an end-user perspective to ensure it works as intended and provides a smooth user experience

The core areas we focused on were key user flows such as registration, navigation, form submissions and viewing stock data. We tested our application on three different users from



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

three different backgrounds to gain a diverse set of end user feedback. Our results showed that all of them were happy to navigate across the pages as we kept our design simple, consistent and accessible to most. It also showed how registering and profile creation – core parts of user functionality, were easy to carry out. The data was also transferred correctly and readable as it was supposed to, and the graphs were displayed without any errors during these tests. Overall, the tests showed a general positive consensus, but in the future, we would like to gain broader feedback by carrying out many more user tests so that our results aren't limited. We would also like to integrate automated UI tests to ensure consistent performance across different user interactions.



"I was able to register myself with a StockShark account easily and my profile was automatically created with my details!"



Sumana A



"Easy to navigate through StockShark. Also, the stock data display was clear and accurate, allowing me to easily compare stock prices."



Murad N



"The UI was consistent and intuitive. Real-time updates were functional, and I could understand the data without confusion."



Abigail F



Project Management

Code-Related Deliverables

Implementation of Clean Architecture Principles

- **Single Responsibility Principle (SRP):**
Each class in the StockShark project is designed to have one responsibility. For example, servlets only manage HTTP requests/responses, while separate classes handle business logic and database interactions. This separation ensures that changes in one area (such as the API integration) do not impact others (e.g., user authentication).
- **Open/Closed Principle (OCP):**
The system is built to be extendable without modifying existing code. For instance, new features can be added by extending base classes or interfaces rather than altering core logic. This allows us to implement future enhancements such as advanced chart interactivity and AI-based predictions without impacting current functionality.
- **Liskov Substitution Principle (LSP):**
All subtypes and implementations are interchangeable with their base types. This means that classes implementing interfaces (like our Data_Handling_Component) can be substituted without altering the correctness of the program. This facilitates unit testing and future maintenance.
- **Interface Segregation Principle (ISP):**
Clients are not forced to depend on methods they do not use. Interfaces in the project (for example, those defining user management functions and data handling operations) are focused and minimal. This ensures that each component only implements what is necessary, enhancing readability and maintainability.
- **Dependency Inversion Principle (DIP):**
High-level modules do not depend on low-level modules; both depend on abstractions. The design uses abstract interfaces for key components (e.g., for database operations and API services), allowing us to change implementations without modifying the core business logic. This increases flexibility and supports unit testing by allowing dependency injection.



Component and Interface Implementation

- The **Database Client** (formerly DAO) encapsulates all database operations, ensuring that all SQL interactions are centralized and changeable without affecting business logic.
- **User Management and Data Handling Components** abstract complex operations (like authentication, portfolio management, and API integration) behind clean interfaces. These components satisfy the principles above while delivering modular and testable code.

Team Management

Team Coordination

- **Team Leader: Evan Balson**
 - **Responsibilities:** Directed Sprint 2 tasks, monitored progress, and ensured timely completion.
 - **Focus Areas:** System Interfaces, composite component diagrams, high-fidelity wireframes, and coding implementation for the Database, AI Analyzer, and API Service.
- **Scrum Master: Maahia Rahman**
 - **Responsibilities:** Scheduled weekly team meetings, managed the Business Concept Model, and created low-fidelity wireframes.
 - **Focus Areas:** Coding implementation for the Graphical Component and Stock Comparison Graph.
- **Mahboub:**
 - **Responsibilities:** Managed the Business Type Model and participated in the documentation process.
 - **Focus Areas:** Coding implementation for the Notification System and Stock Comparison Session.
- **Hala:**
 - **Responsibilities:** Oversaw Use Case development, Initial Architecture, and defined Business Interfaces.
 - **Focus Areas:** Coding implementation for Portfolio, Login/Logout, and User modules. Also contributed to project documentation.



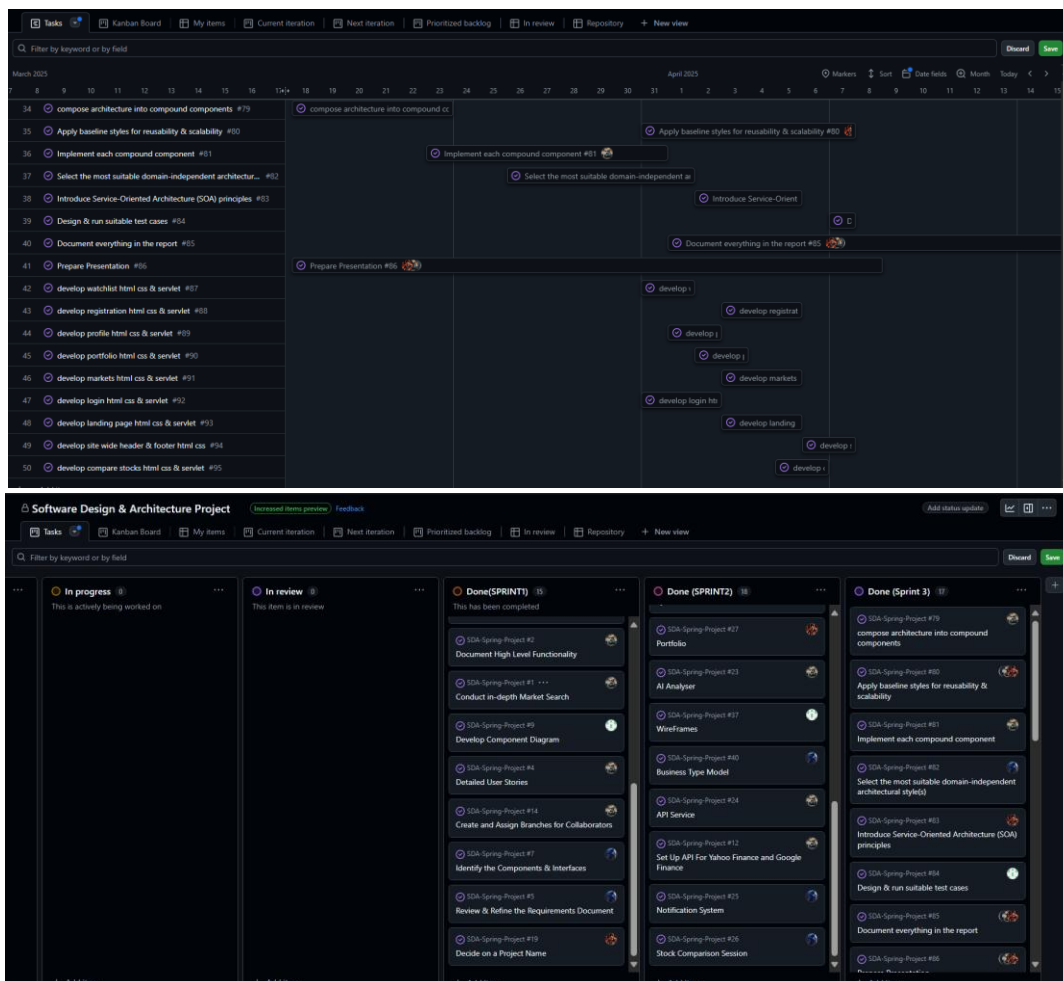
Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Process

- **Task Distribution:**
Tasks were divided among team members based on individual strengths and preferences to maximize efficiency and quality.
- **Code Review and Integration:**
To avoid disruptions in the main branch, the team used pull requests to review code changes before merging them. This process helped ensure that new implementations adhered to our architectural principles and were properly tested.

GitHub Project Board:





Team Code of Conduct

README MIT license

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.
 - 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)**.



Team's Meetings

Software Architecture & Design - 1
Date: 01 February 2025 Time: 19:30 - 20:30 (GMT)
Attendee: Hala, Mahbouba, Maahia, Evan
Discussion: <ul style="list-style-type: none">• Setting up a database to test API data retrieval.• Ensuring the app functions offline without constant internet access.• Retrieving stock data using Google/Yahoo APIs for analysis.• GitHub workflow: accessing the repository, cloning, and local file management.• Reviewing how the UI integrates with API data.• Validation and processing data to ensure usability.• Using user stories to guide feature planning and task assignments.• Technical demos using Visual Studio and terminal commands• Importance of clear team communication.
Decision: <ul style="list-style-type: none">• A database will be set up to support API testing.• The app must support offline functionality for better user experience.• Stock data will be fetched and analyzed via APIs• GitHub best practices followed: clone repository, manage files locally.• UI will be seamlessly connected with API data.• All API data must be validated and processed for user clarity.• User stories continue to guide features development and planning.• Team members to apply technical steps shown in Visual Studio and terminal.• Maintain strong communication and keep everyone aligned.



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Software Architecture & Design - 2

Date: 08 February 2025

Time: 19:00 - 20:00 (GMT)

Attendee: Mahbouba, Maahia, Hala

Discussion:

- The project deadline for sprint 1 is approaching.
- Personas have been completed, but one needs to better reflect buying/selling stock behavior.
- The files are based on the personas and contributed research on architectural, and design uploaded to GitHub.
- The component diagram needs improvement.
- Use case diagrams are still to be worked on.
- The lecture emphasized the importance of keeping the code simple.
- Some items under review on the Kanban board haven't been uploaded to GitHub.
- The team is considering a name for the project, currently using "StockShark".

Decision:

- One persona will be updated to show active stock trading behavior.
- Refine the component diagram to align with user personas.
- Work on the use case diagrams.
- Keep the code simple.
- Upload all under review Kanban items to GitHub.
- The working project name is "StockShark", capturing a sharp focused theme.

Software Architecture & Design - 3

Date: 15 February 2025

Time: 19:30 - 20:30 (GMT)

Attendee: Hala, Mahbouba, Maahia, Evan

Discussion:



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

- The team held a planning session to review and confirm task assignments for the current sprint.
- Responsibilities were distributed based on components such as backend services, user interface, system architecture, and business modeling.
- It was emphasized that members should regularly update progress on GitHub and the Kanban board, ensuring integration readiness.
- Coordination across tasks like the API, user modules, and graphical components is key for smooth implementation.
- All tasks are expected to align with the finalized Business Concept Model and Compound Component Diagram.

Result:

- Evan:
 - Develop and configure the Database.
 - Set up the API integrations for Yahoo Finance and Google Finacc to retrieve stock data.
- Maahia:
 - Complete and update the Business Concept Model.
 - Implement the Graphical Component, including UI visuals and controllers.
- Mahbouba:
 - Implement the Notification System for real time alerts.
 - Develop the Stock Comparison Session feature.
- Hala:
 - Desing the Initial Architecture and define relevant Business Interfaces.
 - Implement Login/Logout functionality.
 - Develop the User module, including user data handling and session control.



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Software Architecture & Design - 4

Date: 22 February 2025

Time: 19:00 - 19:30 (GMT)

Attendee: Hala, Mahbouba, Maahia, Evan

Discussion:

- Responsibilities for different system models and diagrams were assigned across the team.
- Implementation tasks were divided among team members to ensure parallel progress.
- Focus was placed on both backend (database, APIs, AI analysis) and frontend (UI components, graphs) development.

Decision:

- Mahbouba is responsible for the Business Type Model, and has implemented:
 - The Notification System.
 - The Stock Comparison Session
- Maahia is working on the Business Concept Model and Wireframes, and has implemented: Graphical Components (controllers and FXML files)
 - Stock Comparison Graphs
- Evan is assigned to System Interfaces, and has implemented:
 - The Database.
 - The AI Analyzer.
 - The API Service.
- Hala is handling the Initial Architecture and Business Interfaces, and has implemented:
 - Portfolio.
 - Login/ Logout System
 - User Management



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Software Architecture & Design - 5

Date: 01 March 2025

Time: 19:00 - 20:30 (GMT)

Attendee: Hala, Mahbouba, Maahia, Evan

Discussion:

- Workflow process for branching and making changes.
- The need for team communication before commits.
- Progress check on the Business Type Model design.
- Proper branch usage for development.
- Updating and expanding wireframes.
- Ongoing backend implementation tasks.

Decision:

- The team will work in a single branch and submit all changes via pull request.
- Members must message the group chat before committing for approval.
- Progress on the Business Type Model will be reviewed and tracked.
- Wireframes will be updated, with additional pages added as needed.
- Continue with the backend code implementation as planned.

Software Architecture & Design –6

Date: 08 March 2025

Time: 19:00 - 20:30 (GMT)

Attendee: Hala, Mahbouba, Maahia, Evan

Discussion:

- Using existing lab work as a base for implementation.
- Progress on the Business Concept Model.
- Research responsibilities for system architecture and interfaces.
- Communication of updates within the team



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

- Timeline for model completion and feedback.

Decision:

- The Business Concept Model will be completed and updated.
- Initial System Architecture and Business Interfaces will be researched.
- All implementation updates should be shared via WhatsApp.
- All models must be completed by Monday to receive feedback on Wednesday.

Software Architecture and Design – 7

Date: 15 March 2025

Time: 19:00 - 20:30 (GMT)

Attendee: Hala, Mahbouba, Maahia, Evan

Discussion:

- Tasks for completing Sprint 2 documentation.
- Reviewing the code implementation for required/ provided interfaces.
- Responsibility for uploading code implementations to GitHub.
- Uploading the Composite Component Diagram.
- Confirming deadline for code and Sprint 2 completion.

Decision:

- Hala and Mahbouba will work on the Sprint 2 documentation.
- Everyone will review the code for required/provided interfaces.
- Evan and Maahia will upload code implementations to GitHub.
- Mahbouba will maintain a backup of code implementation.
- Maahia will upload the Component Diagram.
- Maahia and Mahbouba will work on the Composite Component Diagram.
- Code implementation deadline is tonight; the Sprint 2 deadline is Tuesday. (18 March 2025)



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Software Architecture and Design – 8

Date: 25 March 2025

Time: 16:30 - 17:30 (GMT)

Attendee: Hala, Mahbouba, Maahia, Evan

Discussion:

- Final Sprint routine and responsibilities were outlined.
- Presentation Preparation:
 - Scheduling two rehearsal sessions (one online, one in-person).
 - Completing the PowerPoint using Canva (link shared via email)
 - Required content and structure of the final presentation were reviewed.
- Priority shift: Coding will begin only after diagrams and presentation elements are complete.
- Syntax and code implementation issues: identified and will be discussed in the next meeting
- Saturday meeting proposal:
 - Focus on beginning the code phase.
 - Cover technical details: folder structure, database access, JSON handling, HTML/JSP views, CSS, chart creation, and code-page assignments.
- Implement wireframes

Decision:

- Maahia will coordinate two presentation rehearsals for next week:
 - One online via Teams
 - One in –person in library study room.
- All team members must access and contribute to the Canva presentation (link sent via email)
- A full presentation draft is due by Friday; diagrams need to be updated using existing project PDFs.
- Do not begin coding assigned pages yet, focus first on finalizing presentation and diagrams.
- Saturday morning meeting scheduled to begin coding discussions and resolve syntax related issues.
- Coding will follow the structure of the Compound Components Diagram



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

- Only tested and validated code will be accepted into the final codebase.
- Initial Wireframes distribution:

Assigned to	Page
Evan	<ul style="list-style-type: none">• Site-wide layout• Home page• Login
Maahia	<ul style="list-style-type: none">• Registration• Profile
Hala	<ul style="list-style-type: none">• Trending Stocks/Markets• Compare Stocks
Mahbouba	<ul style="list-style-type: none">• My Portfolio/ Watchlist• My Watchlist/ Item

Software Architecture and Design – 9

Date: 29 March 2025

Time: 14:00- 16:00 (GMT)

Attendee: Hala, Mahbouba, Maahia, Evan

Discussion:

- The team discussed the correct method for inserting data into a database using JAVA JDBC.
- A step-by-step process was reviewed, including setting up a connection, preparing and executing an SQL statement, and properly managing exceptions and resources.
- Common syntax issues (e.g., incorrect placeholders, missing try-catch, or unclosed resources) were highlighted as areas to clarify in the meeting.
- Emphasis was placed on using try-with-resources to prevent memory leaks and ensure clean code structure.

Decision:

- The meeting was a live walkthrough of JDBC database insertion process:



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

- Establishing a connection using DriverManager.
- Preparing and executing a parameterized SQL insert statement using PreparedStatement.
- Setting correct data types and values for placeholders.
- Using try-with-resources to manage connections and statements safely.
- Handling SQL exceptions properly.
- All members are expected to review this process beforehand to ensure a smoother meeting
- A working example will be implemented and tested together to clarify syntax and improve consistency across the codebase.

Software Architecture and Design – Online Rehearsal

Date: 1 April 2025

Time: 21:15 – 23:00 (GMT)

Attendee: Hala, Mahbouba, Maahia, Evan

Discussion:

- Final presentation slides.
- The focus will be on covering key components: architecture, interfaces, implementation, and testing results.
- Everyone should review the project structure and contribute content relevant to their assigned modules.
- Members also discussed the importance of sharing implementation code with the team to ensure everyone has access to current progress and can provide feedback or integrate their own parts.

Decision:

- All members will start working on their slide content and contribute to the shared presentation file (Canva).
- Deadline for full slide content draft: 3 April 2025
- Team members will share their implementation code on GitHub.



Software Architecture and Design – In Person Rehearsal

Date: 4 April 2025

Time: 12:30 - 14:30 (GMT)

Attendee: Hala, Mahboub, Maahia, Evan

Discussion:

- The team reviewed the finalized presentation slides, ensuring all required topics are covered, including system architecture, interface design, implementation, testing, and results.
- Time management was emphasized, with the need to assign specific slides to each team member to ensure smooth delivery within the allocated time limit (15 minutes total)
- Finalizing all project diagrams, such as the Component Diagram, Business Concept Model, and Interface Design.

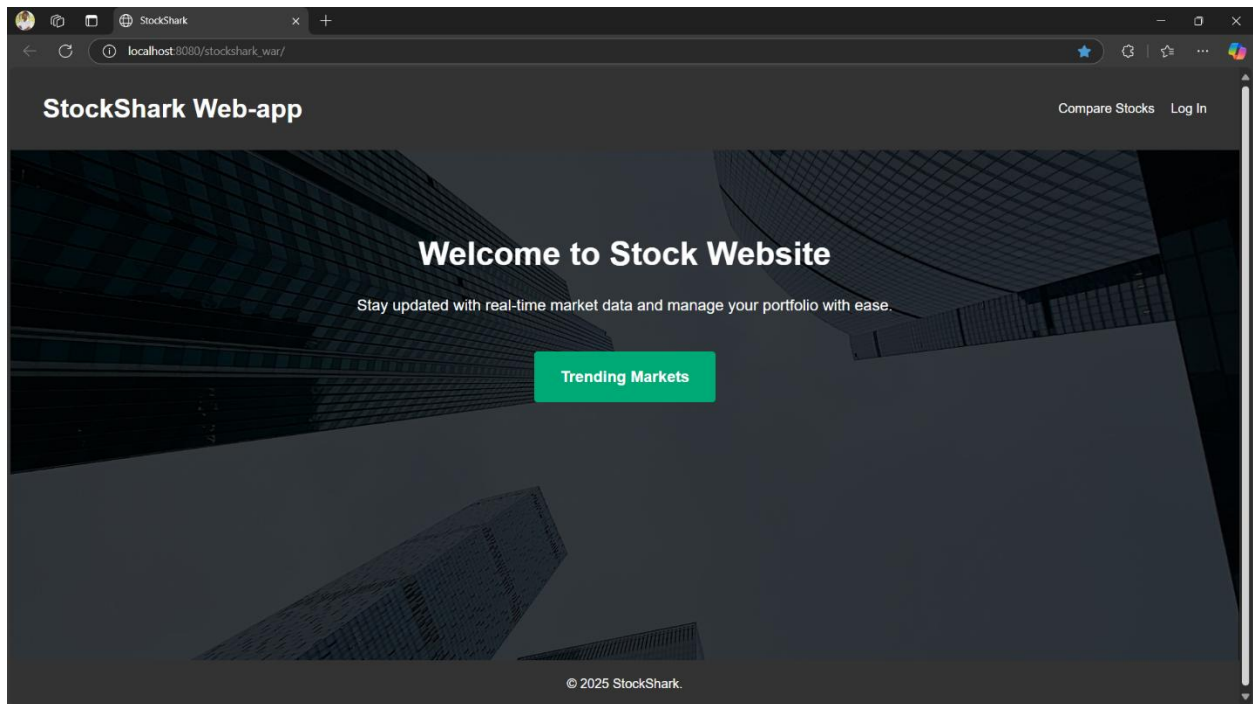
Decision:

Slide	Presenter	Topic
<ul style="list-style-type: none"> • 1-3 • 17-19 	Hala	<ul style="list-style-type: none"> • Introduction, Content Overview • Reusable UI Components, SOA Principles, Dominant Architecture Style
<ul style="list-style-type: none"> • 4-7 • 20-22 	Maahia	<ul style="list-style-type: none"> • Project Management, Agile Advantages, GitHub Projects, Brand Development • Unit Testing, User Testing
<ul style="list-style-type: none"> • 8-12 	Mahboub	<ul style="list-style-type: none"> • System Architecture, Core Components, Business Concept Model, Interfaces, Compound Components
<ul style="list-style-type: none"> • 13-16 	Evan	<ul style="list-style-type: none"> • Core Implementation, Design Patterns, How Architecture Solves User Stories
Final	All	Q&A



Screenshots of the application

Landing Page





Login Page – Includes password hashing

StockShark Web-app

Compare Stocks Log In

Login

Email:

Password:

[Login](#)

Don't have an account? [Register here.](#)

© 2025 StockShark



User Registration Page – includes validation checks and stores hashed password in the database for security.

StockShark Web-app

Compare Stocks Log In

Register

Username:

Email:

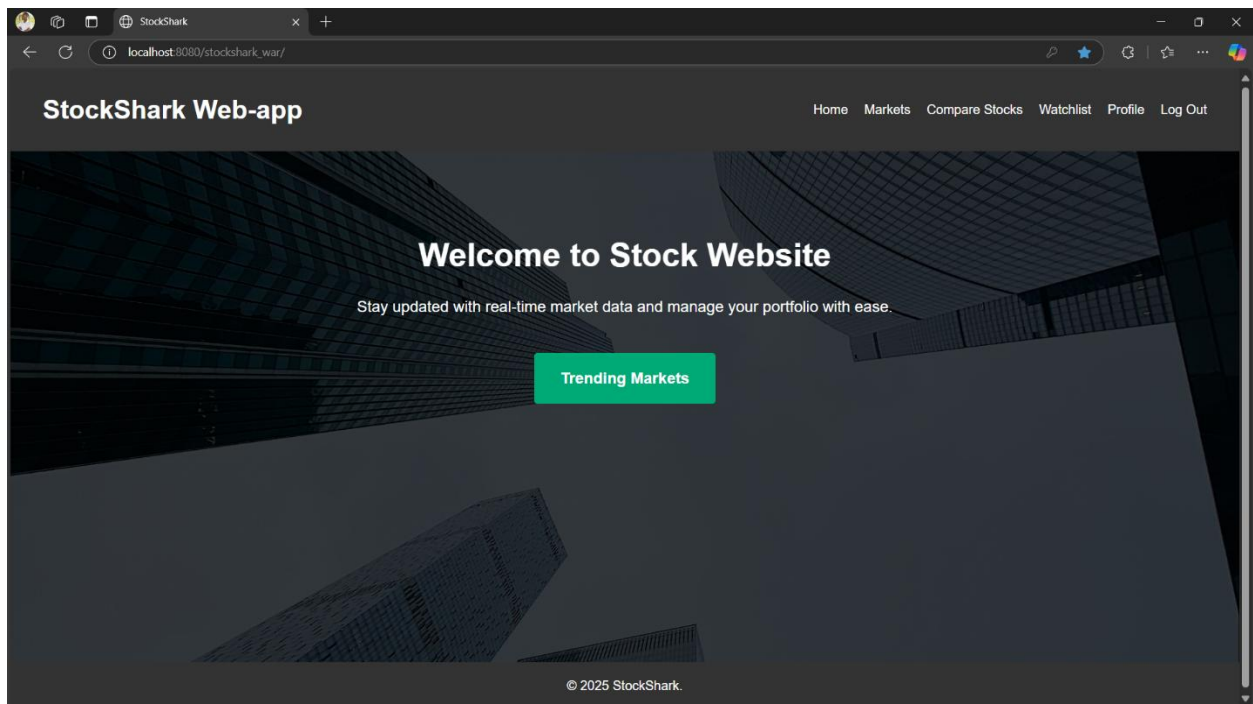
Password:

Confirm Password:

© 2025 StockShark



Landing Page (After successful login) – Additional menu options revealed





Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Trending Markets View:

StockShark Web-app [Compare Stocks](#) [Log In](#)

Market Overview

Symbol	Price	Change	Change Percent
AAPL	\$172.42	-9.04%	-4.9818%
TSLA	\$252.31	-0.09%	-0.0357%
GOOGL	\$157.14	4.32%	2.8269%
AMZN	\$184.87	3.65%	2.0141%
MSFT	\$388.45	7.10%	1.8618%
NFLX	\$918.29	-2.88%	-0.3126%

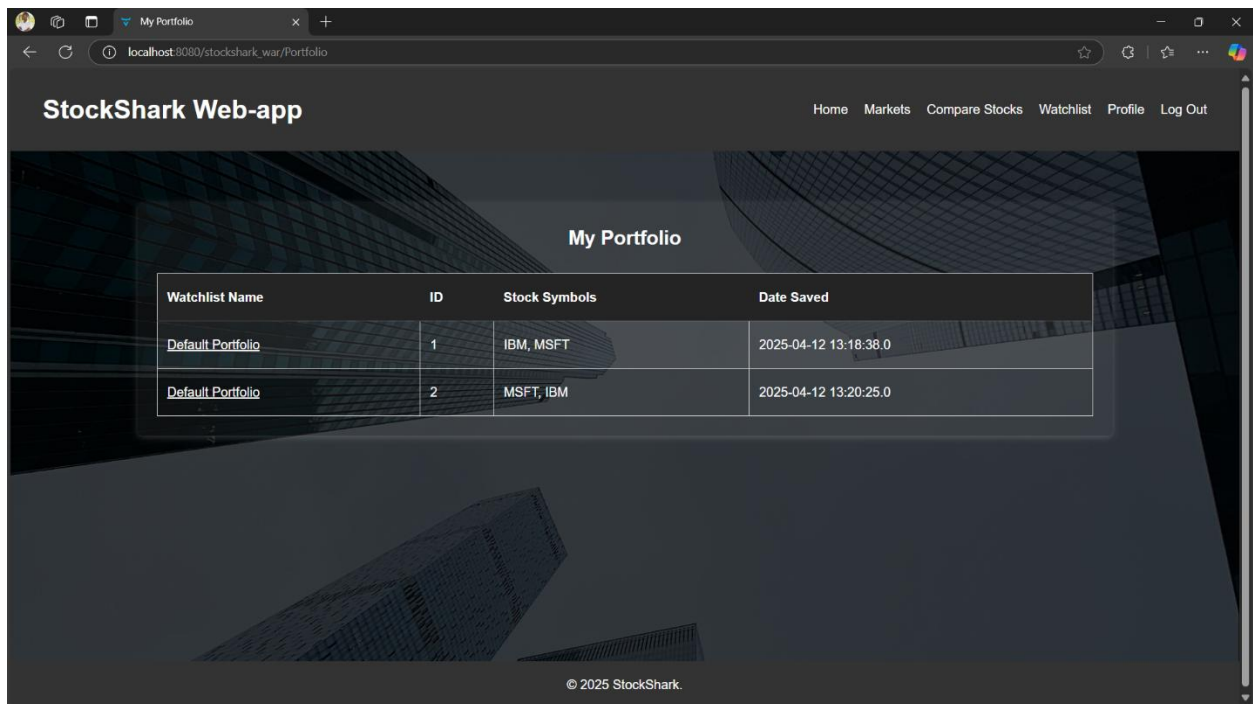
© 2025 StockShark.



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

My Portfolio View: List saved portfolios created by the user.



StockShark Web-app

Home Markets Compare Stocks Watchlist Profile Log Out

My Portfolio

Watchlist Name	ID	Stock Symbols	Date Saved
Default Portfolio	1	IBM, MSFT	2025-04-12 13:18:38.0
Default Portfolio	2	MSFT, IBM	2025-04-12 13:20:25.0

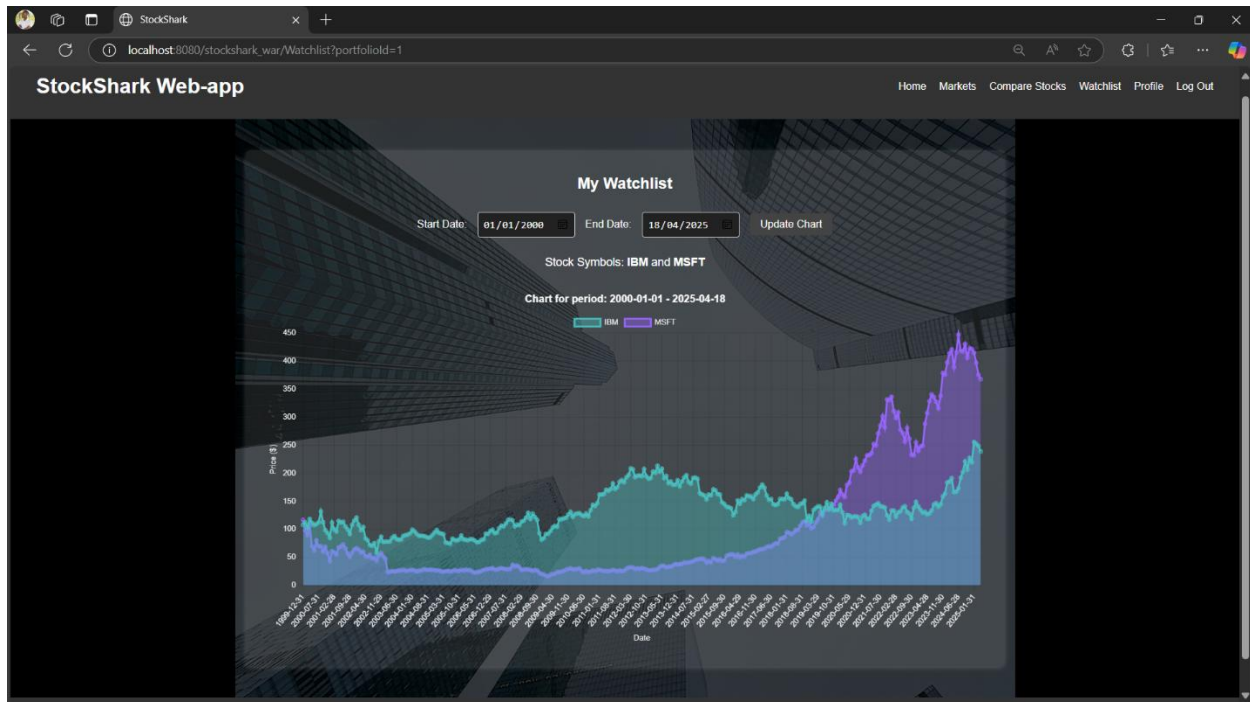
© 2025 StockShark.



Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Watchlist page: After selecting a portfolio item the user can view its details in this window. They can also update the search dates within each portfolio.

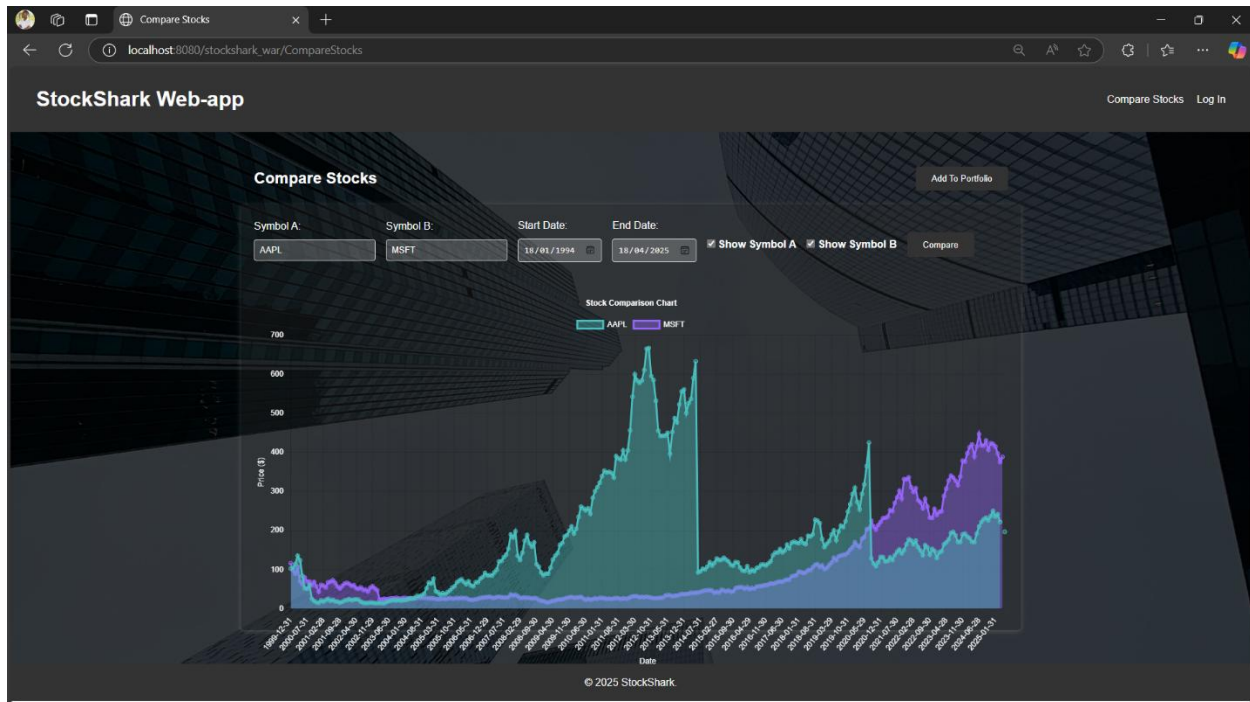




Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Compare Stocks: This page allows guest and registered users to search and compare any two stocks.

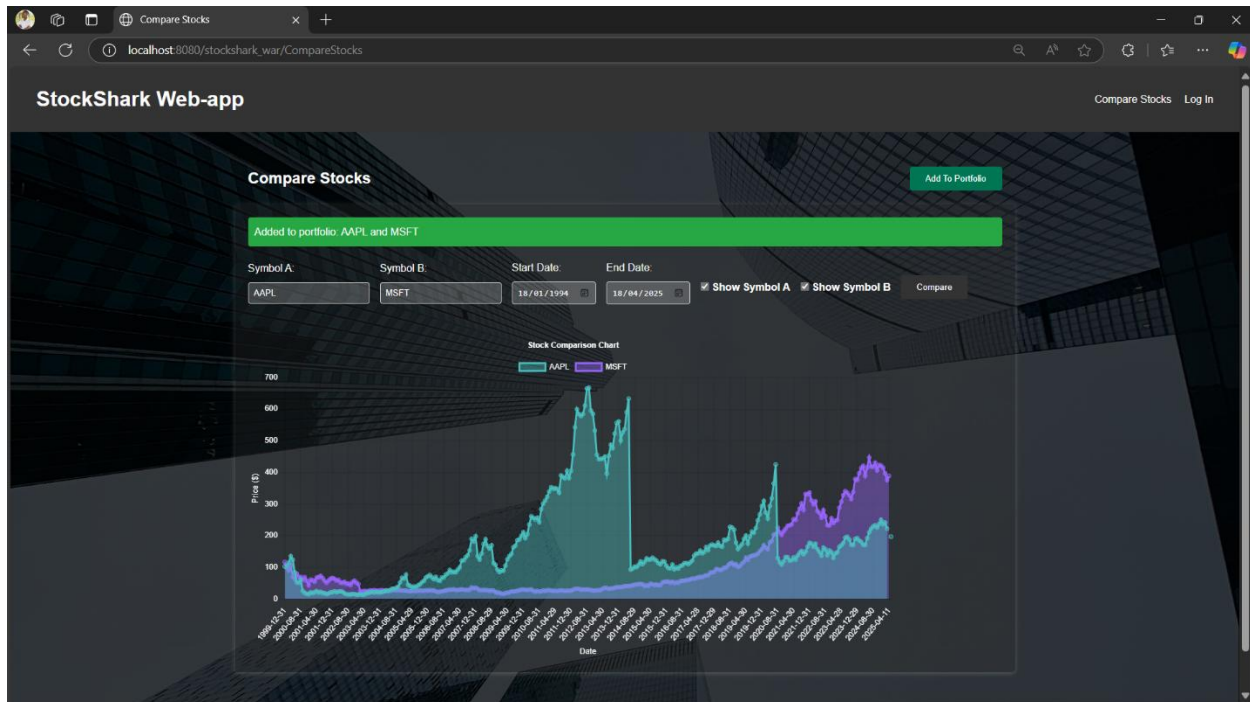




Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

Adding an item to Portfolio: The add to portfolio button is activated when a compare stock session has been initiated. Clicking it will save the active symbols into a portfolio.





Profile page: This page gives the user some account based features to manage their account and details.

User Profile

localhost:8080/stockshark_war/Profile

StockShark Web-app

Home Markets Compare Stocks Watchlist Profile Log Out

Account Settings

Update Email

New Email:

newemail@example.com

Update Email

Update Password

Current Password:

New Password:

Update Password

Delete Account

Warning: This action cannot be undone.

Delete Account

© 2025 StockShark



References:

- [1]. GeeksforGeeks, "Complete Guide to Clean Architecture", [Online], Available: [Complete Guide to Clean Architecture - GeeksforGeeks](#), [Accessed: March. 16, 2025]
- [2] Yahoo Finance, "Yahoo Finance," [Online]. Available: <https://finance.yahoo.com>. [Accessed: Apr. 16, 2025].
- [3] Alpha Vantage, "Alpha Vantage Documentation," [Online]. Available: <https://www.alphavantage.co/documentation/>. [Accessed: March. 16, 2025].
- [4] Chart.js, "Chart.js Documentation," [Online]. Available: <https://www.chartjs.org/docs/latest/>. [Accessed: March. 16, 2025].
- [5] Apache Tomcat, "Apache Tomcat Documentation," [Online]. Available: <https://tomcat.apache.org/tomcat-9.0-doc/>. [Accessed: March. 16, 2025].
- [6] XAMPP, "XAMPP Documentation," [Online]. Available: <https://www.apachefriends.org/index.html>. [Accessed: March. 16, 2025].