Stock Shark

THE SOFTWARE ARCHITECTURE & DESIGN

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

Content

Project Overview

- - Meet the team
- - The purpose of this project
- - Tools and Constraints
- - App Overview & Branding

Project Management & Planning

- - Management Strategies Employed
- - Managing Rules and Conduct

Core Components

- - The Architecture
- - Compound Components
  - - Interfaces
  - - Justification
  - - Advantages
  - - How StockShark solves the problem
- - Reusable Components: UI & Models
- - Technical Overview
  - - Implementation
  - - Architectural Styles Used
  - - Implemented Principles of SOA
- - Testing

Project Demo

Introduction

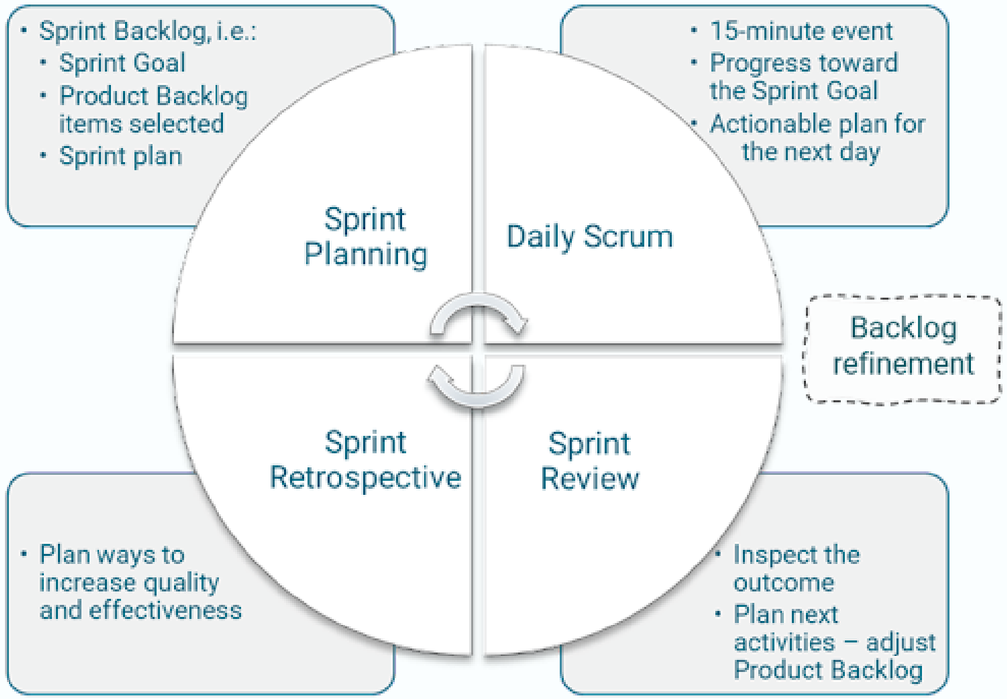
StockShark is a web application designed to provide users with quick and accessible stock data, powered by real time information fetched from the third-party Finance API. This project aims to deliver a simple tool that enables users to compare historical stock data and make informed decisions.

Our Aim:

Plan & Develop an architecture that will efficiently manage the operations of this application.

Tools Used:

Project Management



Agile methodology: is a project management framework that breaks projects down into several dynamic phases, commonly known as sprints.

GitHub

Advantages of Agile

- Feature Prioritisation - Focus on the most important features first for the maximum impact
- Iterative Improvements - Continuous updates based on user feedback
- Divide & Conquer - Work is broken into small, manageable tasks for efficiency
- Team Ownership - Teams collaborate and make decisions together
- Promotes strong coordination between product owners, developers, and other stakeholders.
- Strong Coordination - Ensures seamless communication between product owners, developers and stakeholders
- Incremental Releases - Delivers small, functional updates early for quick feedback and validation
- Built-in Quality Assurance - Ensures high standards and minimises errors through development
- Risk Reduction - Continous delivery helps detect and resolve issues early

The Code of Conduct

README MIT license

Code of Conduct

Team Guidelines

Respect and Inclusion

- Respect each team member's ideas and inputs.
- Foster a supportive and inclusive environment for collaboration.

Communication

- Communicate clearly and concisely.
- Keep the team updated on your task progress, blockers, or delays.

Accountability

- Provide feedback on your tasks during meetings.
- If no progress is made, state **why** and what you are doing to catch up.
- If you fall behind significantly without valid reason, it will be reported.

Meeting Etiquette

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

Disputes

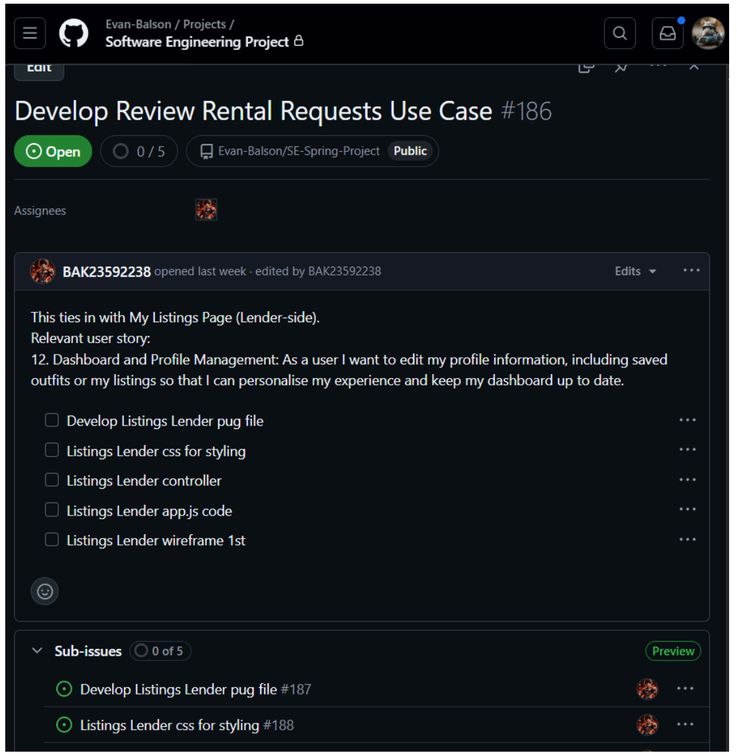
- If dissatisfied with a decision (e.g., on design features), provide proper reasoning.
- Disputes must be well-researched and presented to the team for discussion.

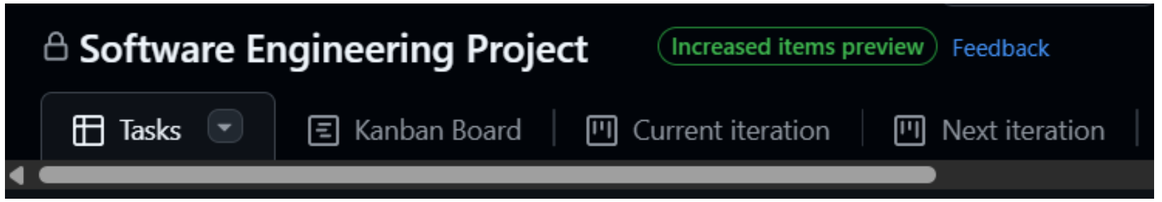
Feedback and Conflict Resolution

- Offer actionable, constructive feedback.
- Escalate unresolved conflicts to **Hala (Scrum Master)** or **Niwhar (Product Manager)**.

Github projects

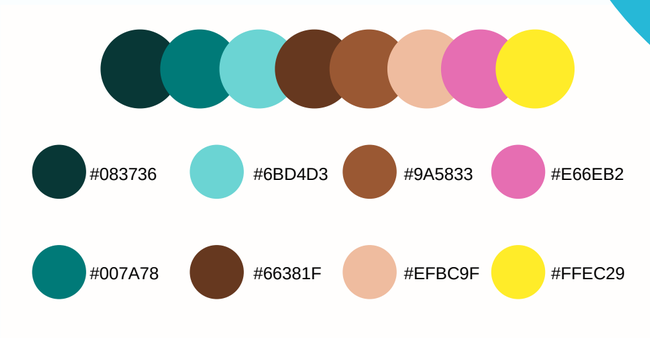
- Version Control: GitHub uses Git, a distributed version control system that tracks changes to your code. Git allows multiple developers to work on a project simultaneously without overwriting each other's changes.
- Collaboration: GitHub is designed for collaborative work. It allows multiple people to contribute to a project, review code, discuss issues, and merge changes efficiently.
- Project Management: GitHub provides tools for managing your projects, such as issues, pull requests, and project boards. These features help you keep track of tasks, bugs, and enhancements





Brand Development

recreate this palete



What Makes a successful brand?

1. Provides a solution to a specific problem
2. Easy to remember naming convention
3. Recognizable Logos icons and elements
4. Easy of use and accessibility
5. Meets consumer expectations
6. Consistency

What is branding & Why is it important?

- Branding is the process of creating a distinct identity for a business. It involves researching, developing, and applying features that consumers can associate with a brand's products or services.

LOGO

LOGO

Aa

Palantino Bold

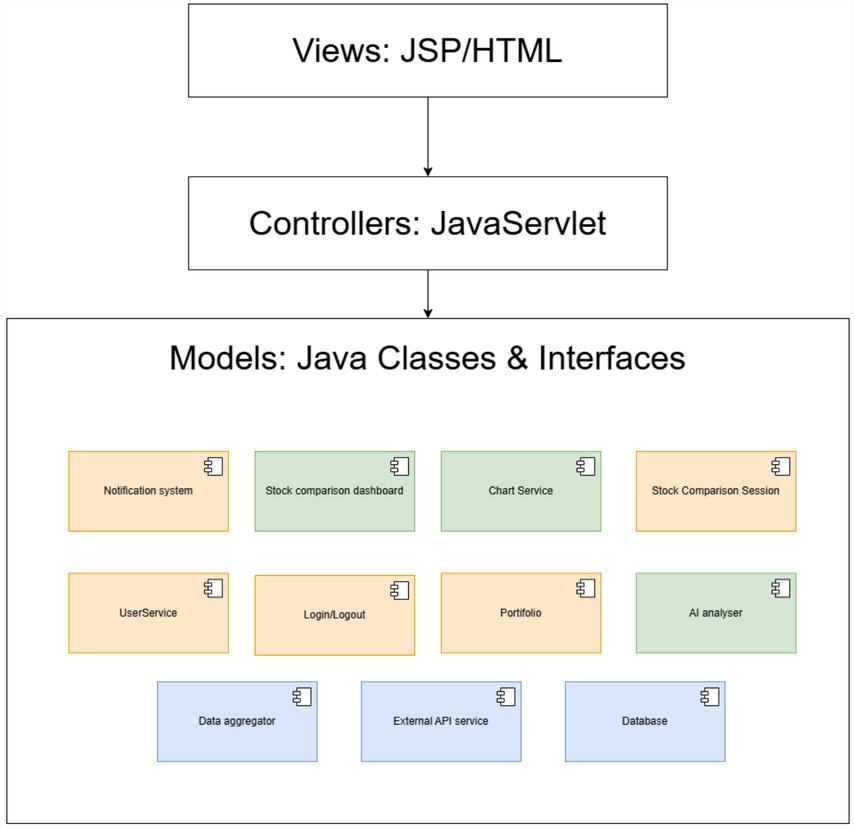
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0

Aa

Calibri (MS)

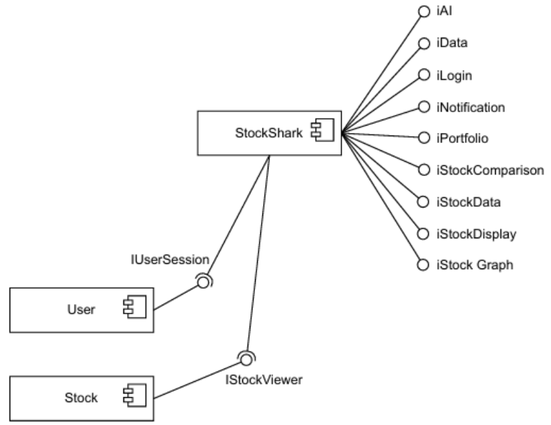
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0

The Architecture

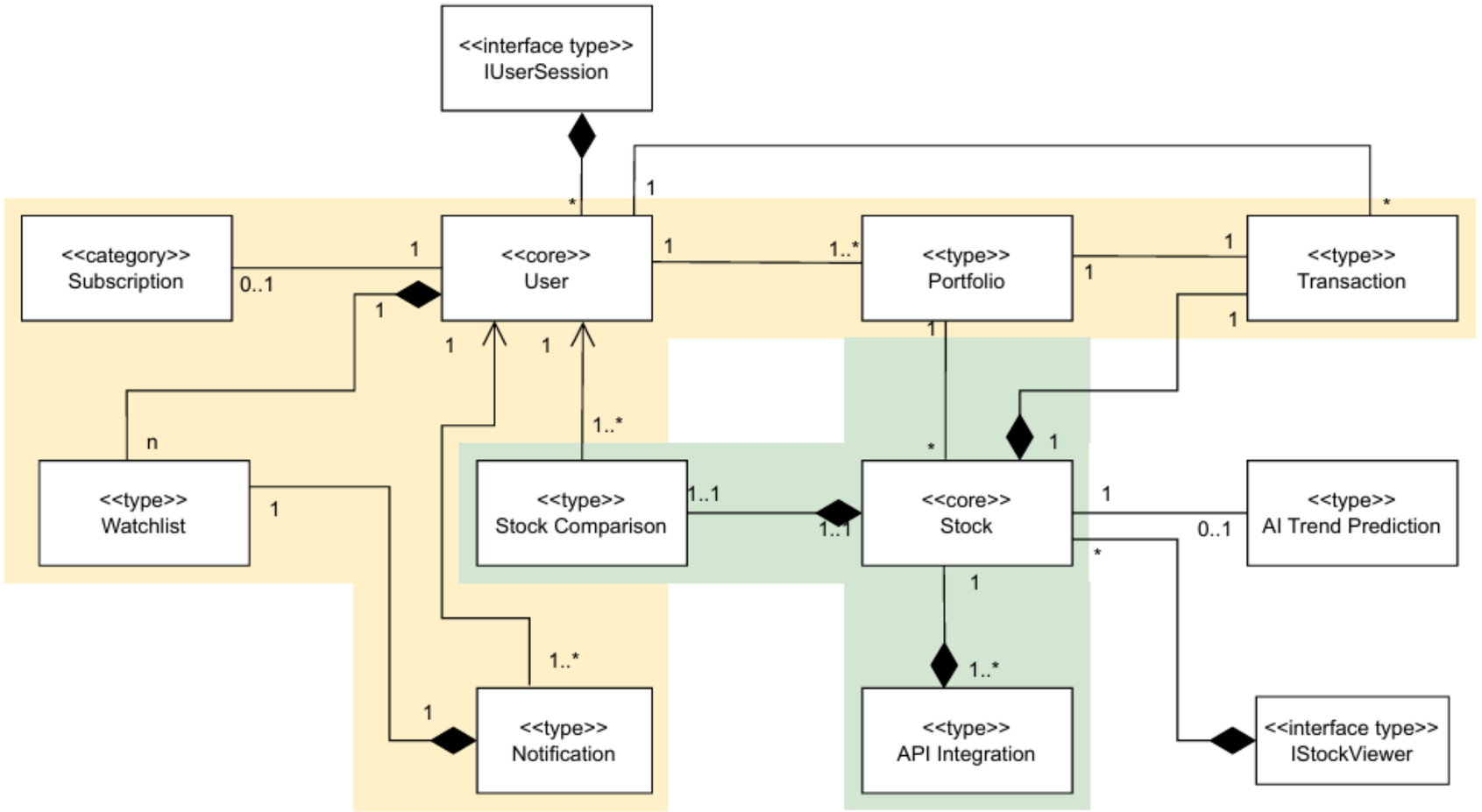


The architecture is divided into :

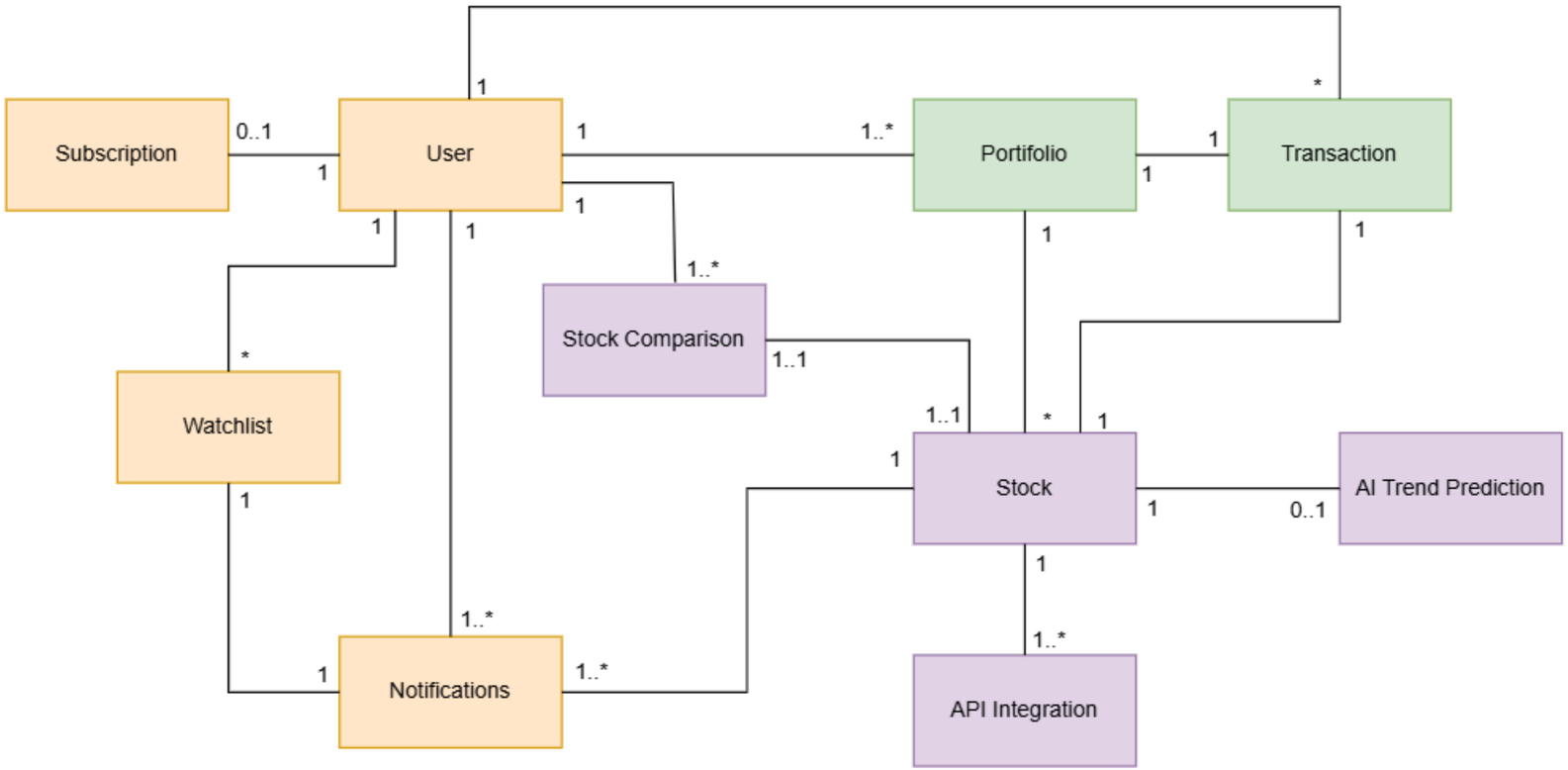
- Observer pattern:
  - The notification system listens for changes made in the database which periodically collects the latest updates from the Stocks API.
  - Then the other side responsible for the port - I notifications publishes the information to the user service
- Model view controller: this is employed on a much broader scope than the individual component connections. In order for the user to interact with the system. We have developed the UI using Java Service Package files and HTML.
  - Java itself is not a web oriented language. Therefore .jsp files are necessary inorder to inject our functions within html.
  - The controllers are responsible for managing the communication between the two.
- 5tiered architecture
  - This means our architecture also satisfies the five tiered architecture. Client side interface, business logic, and database server which is provided using Xammp.
  - Sub components employ 2 tiered architecture
  - Where We are required to source the stock data from Alpha vantage (our External source). We have developed a database client that can request data from this API.



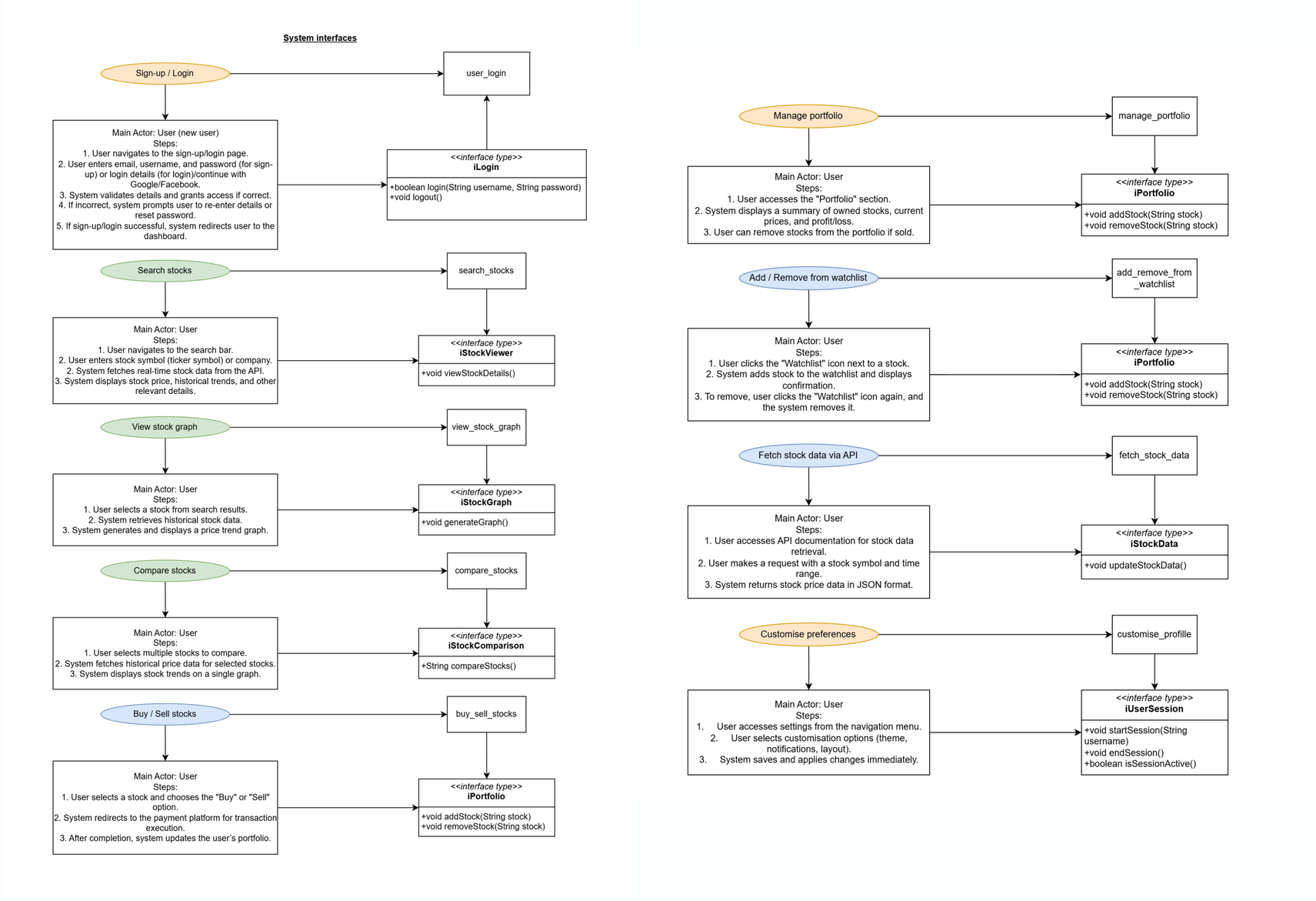
StockShark Components



Business Concept Model

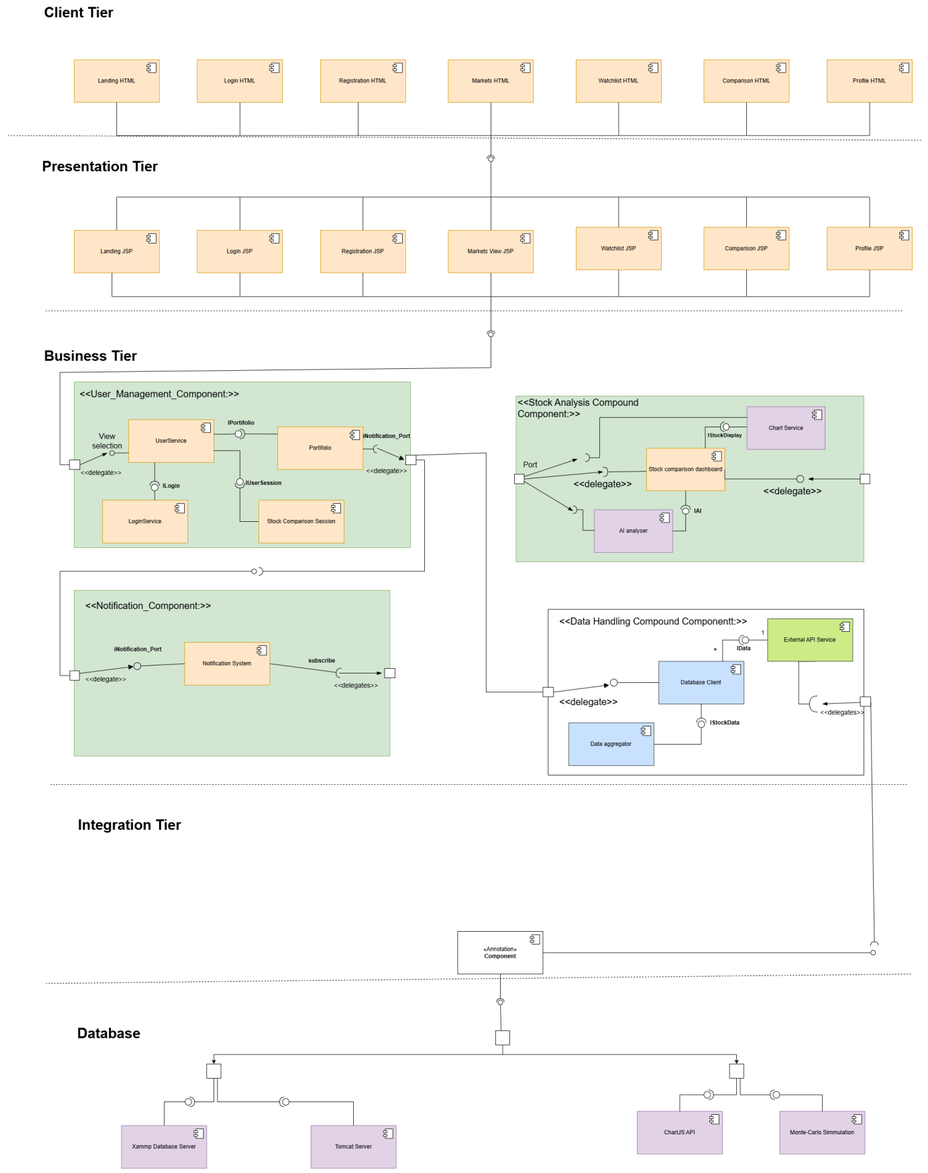


System Interfaces



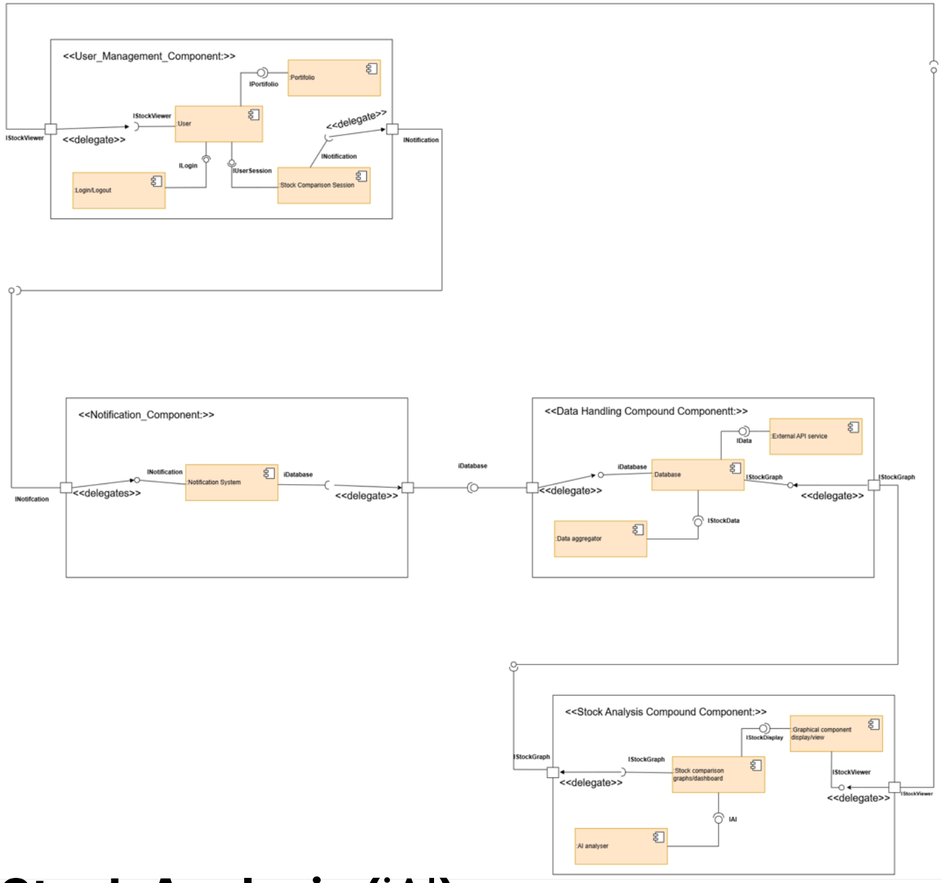
Compound Components

A Detailed look into the architecture and its components



CORE COMPONENTS

STOCKSHARK CONSISTS OF KEY COMPONENTS THAT WORK TOGETHER TO DELIVER STOCK MARKET ANALYSIS AND PORTFOLIO MANAGEMENT.



01 User Management (IUserSession, iLogin)

The ‘User’ component acts as the central hub, managing user sessions (IUserSession), interactions, and access to the application’s features.
This component is responsible for managing user interactions, handling login and logout operations through the ‘iLogin’ interface, and granting access to essential system features. User can manage their portfolios, compare stocks, and receive notifications, making this a critical part of the application

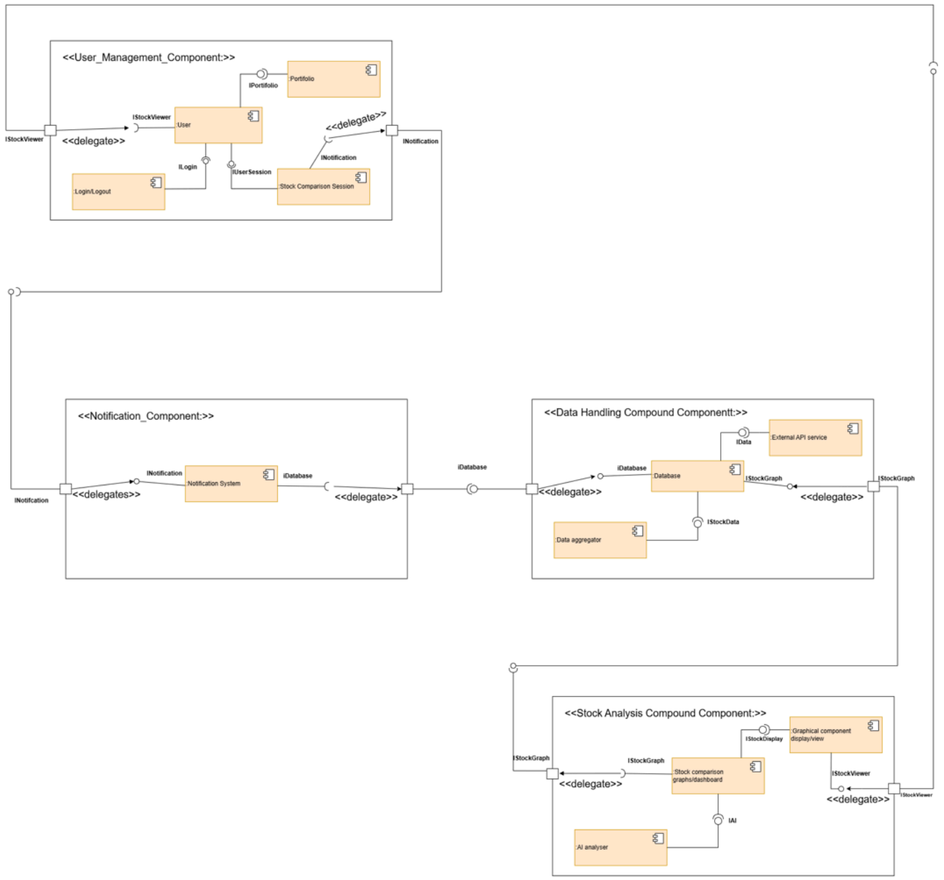
02 Stock Data Handling (iStockData, Database)

The ‘Stock’ component represents individual stock information, crucial for both portfolio management and analysis.
This class is used by the portfolio to store the stock information, and by the API Integration to get the stock data. Additionally, the ‘DataAggregator’ abstract class, which implements ‘iStockData’, ensures stock information is continuously fetched and updated.

03 Stock Analysis (iAI)

The system incorporates AI-powered stock analysis through the ‘AiAnalyser’ class, which implements the ‘iAI’ interface. This component is designed to analyze stock trends and predict potential market movements.

CORE COMPONENTS



04 Graphical Visualization (iStockDisplay)

StockShark includes a visualization module managed by the ‘GraphicalComponent’ class, which implements ‘iStockDisplay’. This component integrates with JavaFX ‘LineChart’ to display stock trends graphically. The ‘GraphicalComponent’ relies on ‘iStockGraph’ to generate stock performance graphs, allowing users to interact with stock data in an intuitive and visually engaging manner.

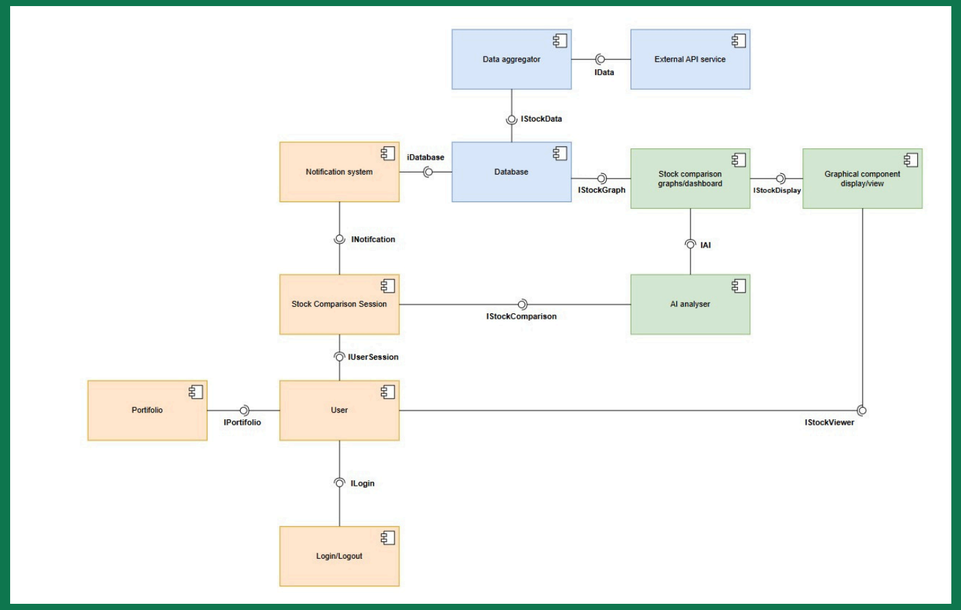
05 Stock Comparison (iStockComparison)

The ‘StockComparisonSession’ class, implementing ‘iStockComparison’, enables users to compare different stocks effectively. Additionally, the ‘NotificationSystem’ class, which implements ‘iNotification’, ensures users stay informed by sending real-time alerts on stock movements, portfolios changes, and market updates. These features enhance user engagement and help investors make timely decisions.

06 Notification (iNotification)

Notification sends real-time alerts on stock market changes and keeps the users updated on portfolio movements and significant events.

COMPOUND COMPONENTS



The StockShark system follows a modular and layered architecture, ensuring scalability, maintainability, and efficiency. Below is an overview of the key architectural components, their interfaces, and their justification for use.

Observer Pattern (Notification System):

The ‘NotificationSystem’ listens for stock updates and sends real-times alerts. Helps keep users informed on stock movements and portfolio changes.

MVC (Model-View-Controller):

The ‘GraphicalComponent’ acts as the Vies displaying stock trends. The ‘StockComparisonSession’ and ‘AIAnalyser’ acts as the Controller, handling user input and processing stock data. The ‘stock’, ‘Portfolio’, and ‘Database’ serve as the Model managing data.

Facade Pattern (User Component):

The ‘User’ component acts as a central hub, providing a unified interface to manage user sessions, portfolios, and stock comparisons. Hides the complexity of multiple interconnected components.

Factory Pattern (Data Aggregation & API Integration):

the ‘DataAggregator’ implements ‘iStockData’, ensuring stock data is fetched consistently from different sources.

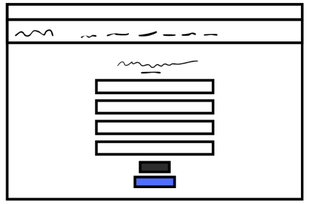
HOW GIVEN ARCHITECTURE PROVIDES A SOLUTION TO USER STORIES

- **User wants to manage their portfolio** → The ‘User’ component facilitates stock additions/removals, storing data via the ‘Database’
- **User wants AI-Driven stock predictions** → The ‘AIAnalyser’ component processes stock data and provides trend forecasts.
- **User wants real-time stock updates** → The ‘NotificationSystem’ alerts users of important stock movements.
- **User wants a visual representation of stock trends** → The ‘GraphicalComponent’ integrates with JavaFX ‘LineChart’ to display stock trends.

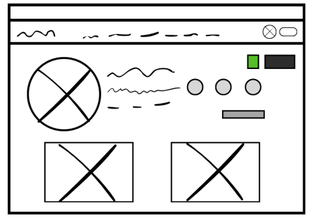
REUSEABLE UI COMPONENTS

Low-fidelity Wireframes

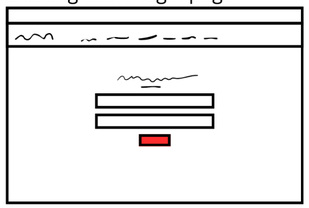
Create account page



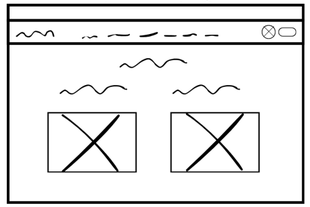
Profile page



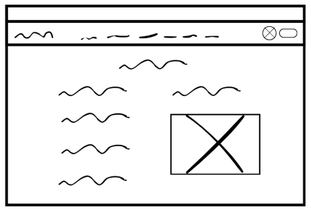
Existing users login page



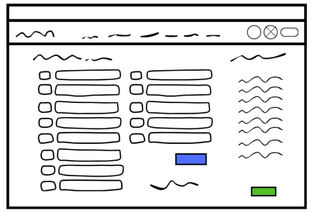
Stock comparison page



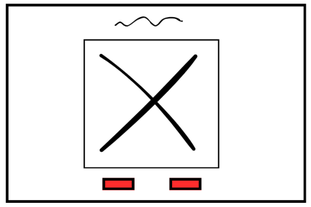
Watchlist



Stocklist



Landing page



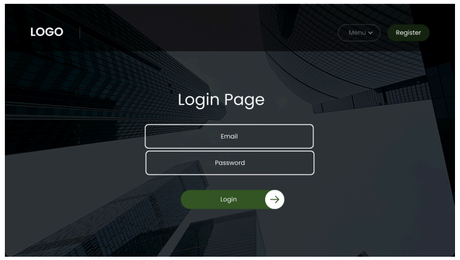
Why use reusable UI components?

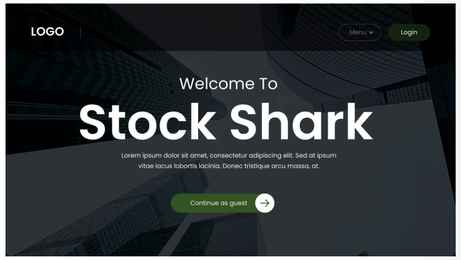
By using reusable components in our project, we saved time, reduced redundancy, and made the system easier to maintain and scale.

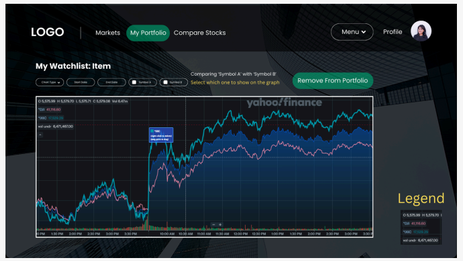
What are some examples in our system?

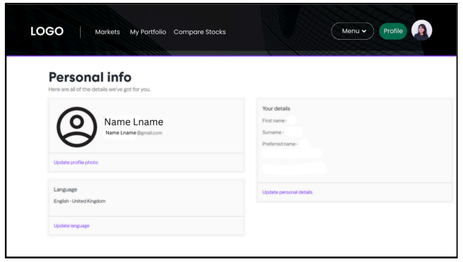
Some examples include our login/account creation forms with user input fields and graphs that we generated with the help of Chart.js tools.

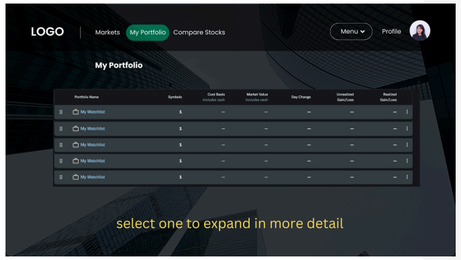
High-fidelity Wireframes

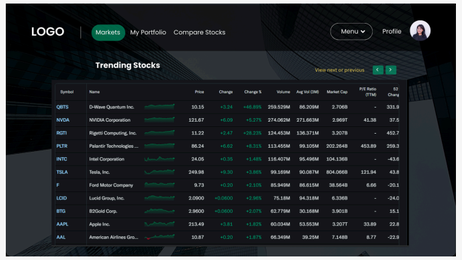












SOA

SERVICE ORIENTATED ARCHITECTURE

User → Login → Session Created → Data Requested → Graph Displayed

01 What are they?

Service-Oriented Architecture (SOA) is an architectural design principle where the system is divided into a set of independent services that communicate over a network. Each service handles a specific task and can be reused by multiple clients.

02 How did we implement these principles?

- User Authentication: Handles user login and session management.
- Stock Data Retrieval: Fetches stock data from external APIs.
- Data Display: Processes and presents the stock data on user-facing graphs.

03 How did this benefit our system?

SOA enhances modularity, scalability, and interoperability, allowing independent service updates and easy integration. It also improves maintainability by enabling isolated changes without disrupting the entire system.

MOST DOMINANT STYLES USED

01 Adapter

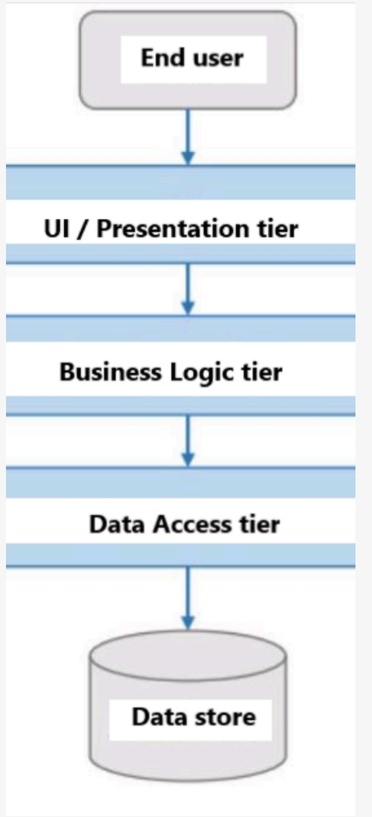
The Adapter pattern in our project is used to integrate external stock APIs. It converts the raw API data into a format that our system can easily process, allowing us to display stock prices and perform analysis without tightly coupling the system to the API.

02 Model View Controller

- Model: Contains the application's data and logic (e.g., UserService and LoginService).
- View: The JSP pages (like profile.jsp, register.jsp) are responsible for displaying the user interface.
- Controller: The Servlets (e.g., RegisterServlet, LoginServlet) act as the controller, processing user input, invoking the appropriate business logic, and selecting the view to render.

03 N-Tiered Architecture (5-Tier)

- Presentation Layer: The JSP files (e.g., profile.jsp, register.jsp) represent the user interface where users interact with the system.
- Business Logic Layer: The services (like UserService, LoginService) handle the core logic of the application, such as managing user registration, profile, and authentication.
- Data Access Layer: The data access layer handles interactions with the database, via SQL, to store and retrieve user data.



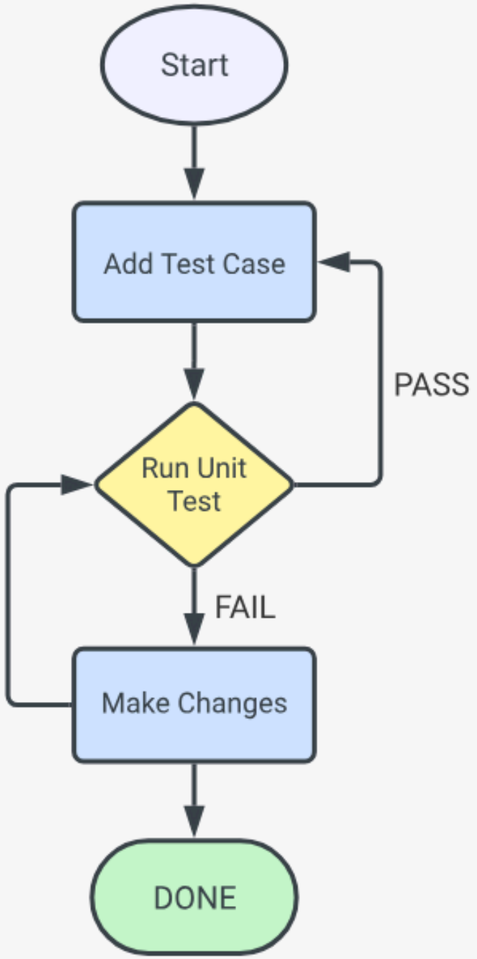
UNIT TESTING

01 What is Unit Testing?

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

02 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.



03 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

★★★★★

"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

What is User Testing?

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

Gaining Feedback:

We focused on key user flows such as registration, navigation, form submissions and viewing stock data.

What our next steps would be:

- Expand User Testing: Test with more users to gather broader feedback.
- Automate Testing: Integrate automated UI tests to ensure consistent performance across different user interactions.

Application Demo

Q&A

References

[1]. https://getbootstrap.com/docs/5.3/getting-started/introduction/
[2]. https://www.chartjs.org/docs/latest/getting-started/