YANE STOCK

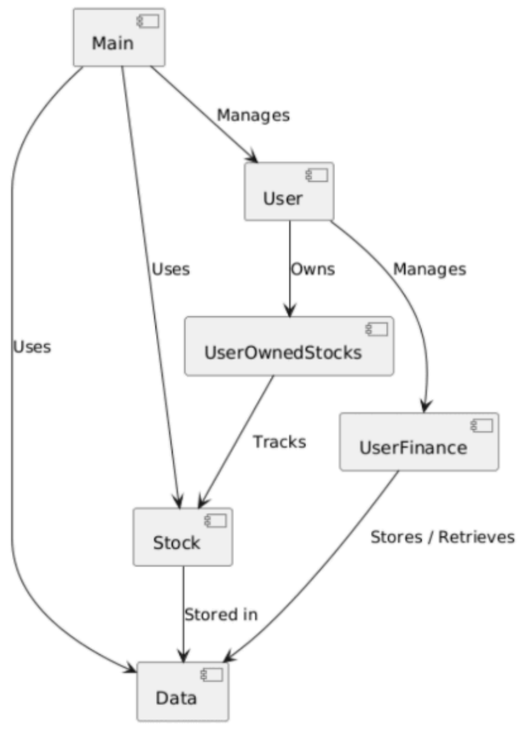
Sprint 3 Report

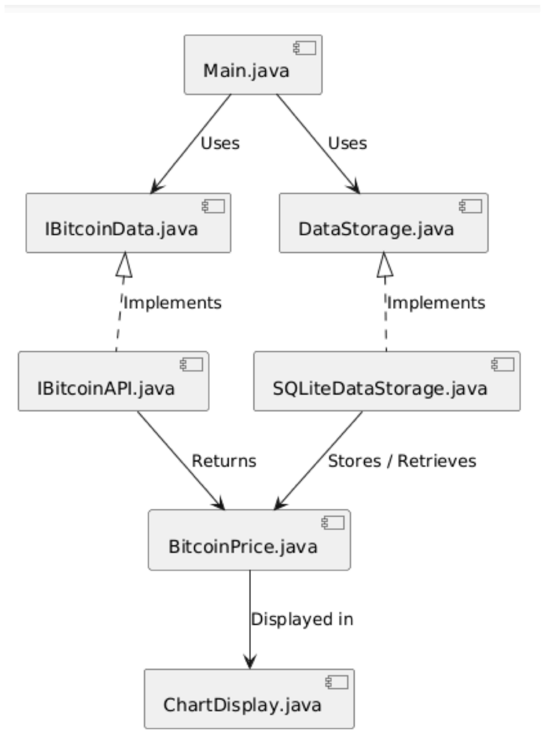
Enea Shahinaj (SHA23591192)
Adam Maayni (MAA23579597)
Yulian Kokalov (KOK23591755)
Niwhar Amin (AMI23589889)

Sprint 1 Goal:

The goal of sprint 1 is to create the conceptual base for the code of our web app, to understand the full context and requirements for our program, figure out all details and how it will be implemented step by step, even if not implemented precisely the same, as a big point is to adapt our program to best aid our actual functionality. Also to grasp a logical understanding of how our plan will have to be executed in terms of code later on and how/what it will do.

Sprint 1 Diagrams:





This was our initial diagram that we built. However, after the demo we had realised that the relationships were not satisfying the requirements so you will see this diagram further into the document seeing how we changed and implemented the new features to satisfy and approve for our Sprint 2 demo.

In the second UML diagram we broke down how the bitcoin data will be tracked, stored and displayed to the user.

Reflection upon diagram from sprint 1 feedback:

Both of these diagrams were not the correct way to approach the task we had at hand. While not incorrect they were not what was the task we were given as a group and therefore the initial code which was presented was also incorrect since it followed the structure of the diagrams. They helped us get an idea and picture a way to make the program and some appropriate links however overall gave the wrong structure on how to lay the foundations of our code.

Future implementations and ideas moving forward:

The next step to this program is ensuring that moving forward we fix the structure which we are meant to use in our code to create the program and ensure that our diagrams in sprint 2 and onwards are all related when necessary to each other and to the code itself. Ensuring we follow the structure of the diagrams in the code means that we have a clean design of our program and it is easy to follow and understand what is going on and how the program works.

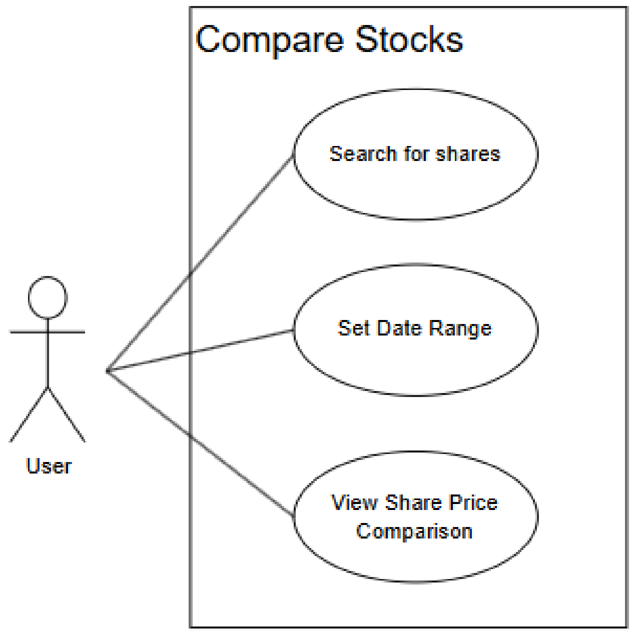
Sprint 2

Sprint 2 Goal:

Designing the diagrams that are in the sprint 2 requirements. These diagrams will serve as the foundation for a code skeleton that we will begin implementing in Sprint 3. Our aim is to closely follow our Kanban board, working on tasks according to their assigned days. This approach will help us stay on track and ensure timely completion of the project specifications.

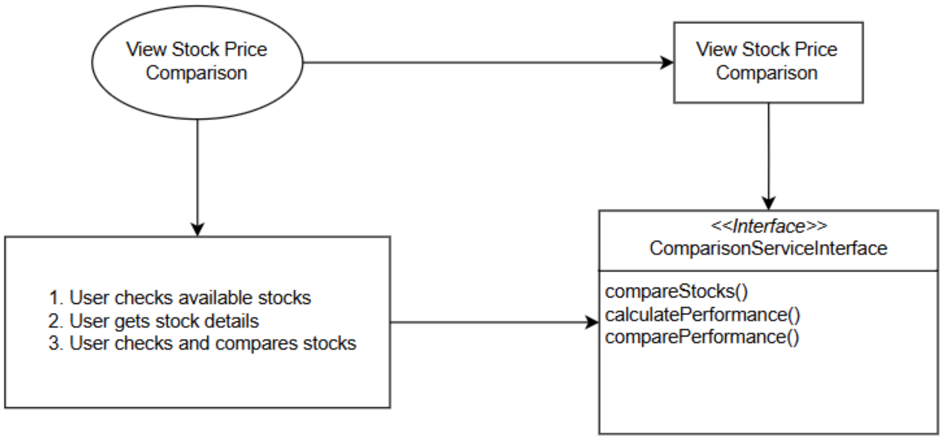
Sprint 2 Diagrams:

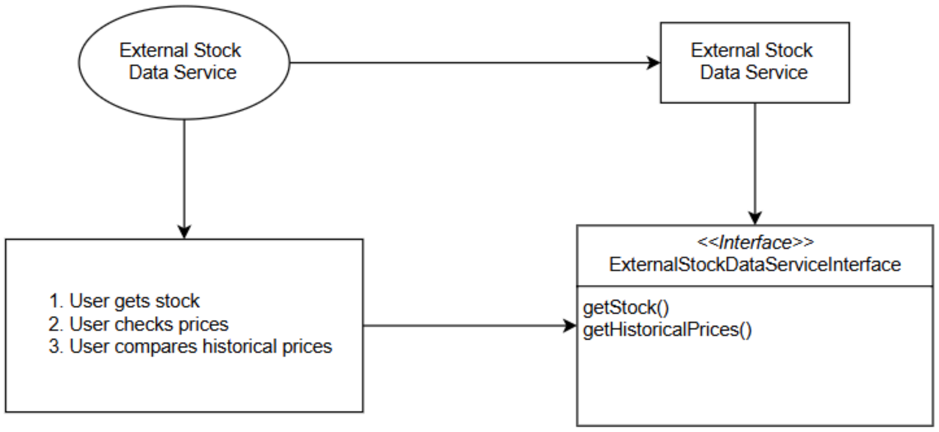
Use Case:

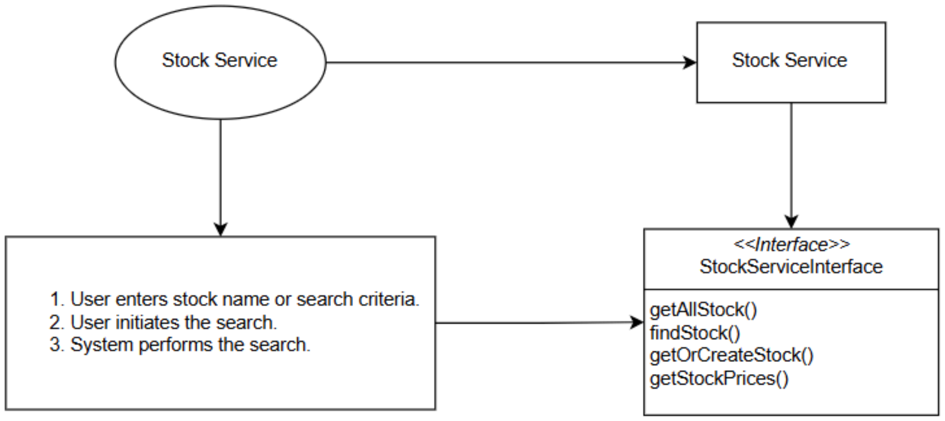


We decided to create a simple use case diagram which outlines the main components of what our program is designed to do to ensure that we complete the requirements of the coursework.

System Interface:

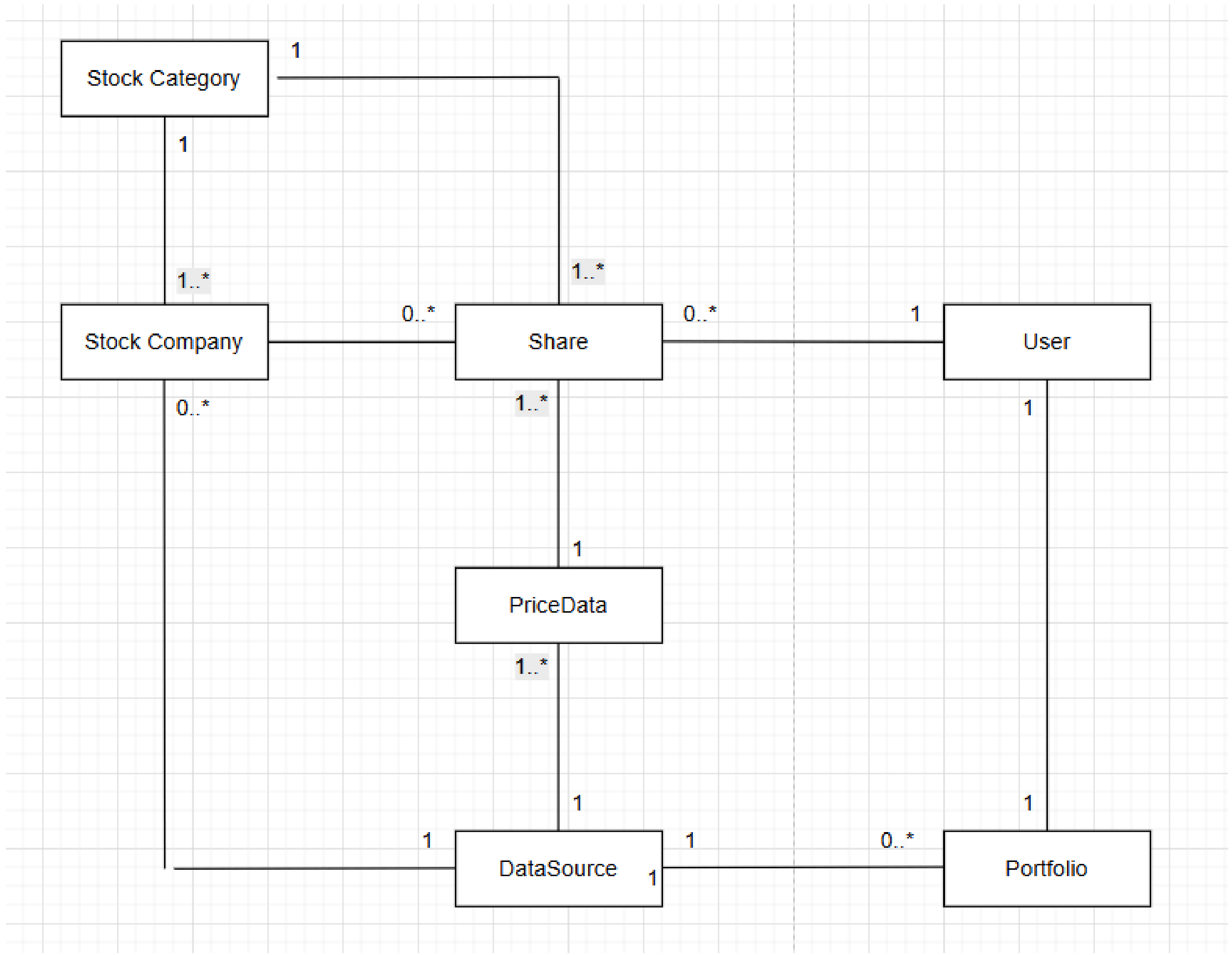

View Stock Price Comparison
View Stock Price Comparison
1. User checks available stocks
2. User gets stock details
3. User checks and compares stocks
<<Interface>>
ComparisonServiceInterface
compareStocks()
calculatePerformance()
comparePerformance()


External Stock Data Service
External Stock Data Service
1. User gets stock
2. User checks prices
3. User compares historical prices
<<Interface>>
ExternalStockDataServiceInterface
getStock()
getHistoricalPrices()



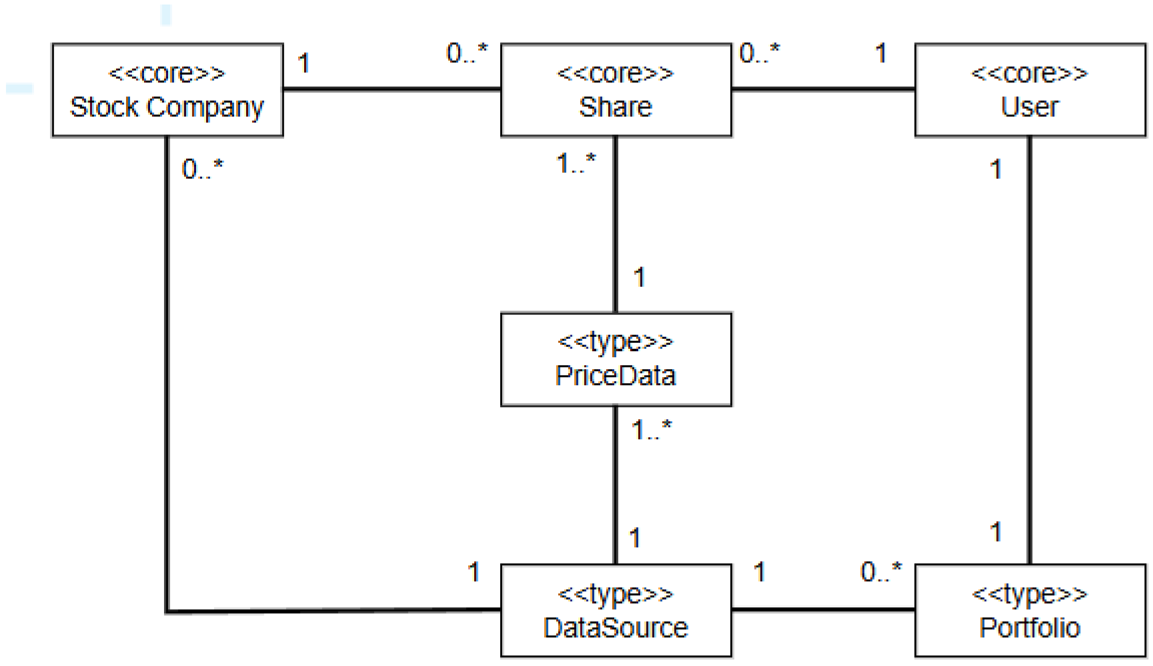
This diagram shows the steps the user follows for three actions: setting a date range, searching for shares, and exporting/saving data. For each action, the user interacts with an interface (ICompareRrice) to perform tasks like selecting a date range, searching for stocks, or exporting data, with the system handling the operations based on the user's inputs.

Business Concept Model:



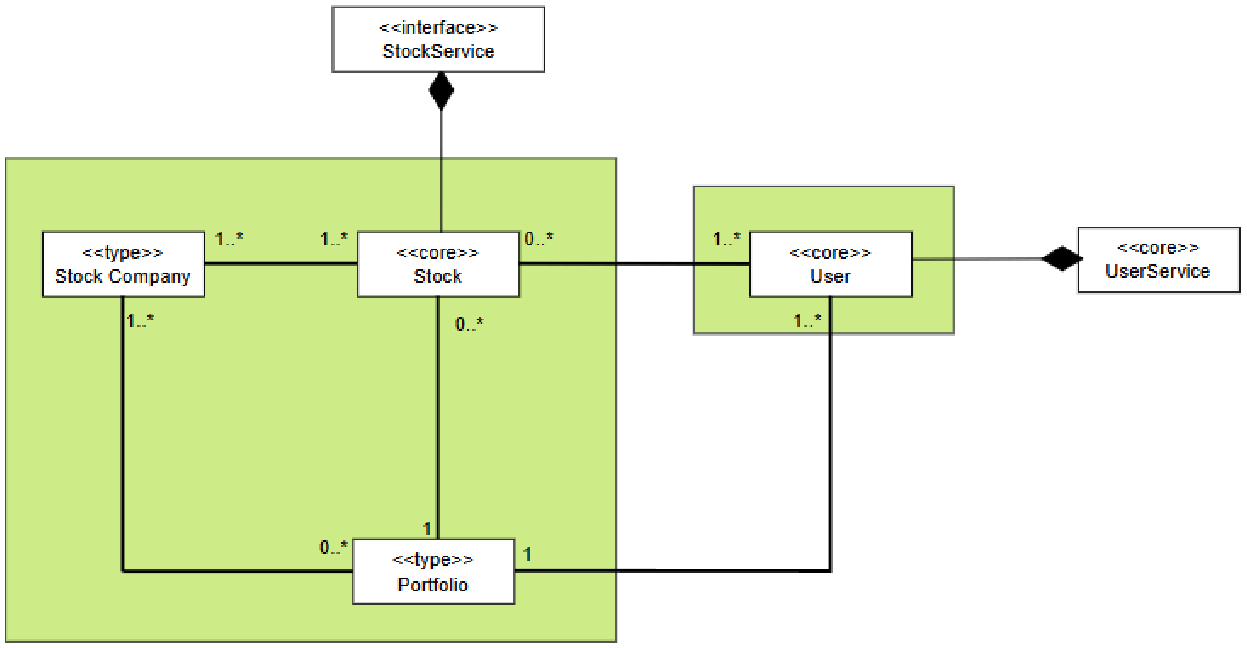
This is the way we designed our business concept model, we chose it since it describes in detail and logically how exactly the stock system would work together in the app along with the user, the share is linked to a company, which has a category, and the Share has a PriceData obtained from DataSource which is all linked to the user that has a portfolio, this was an easy setup both visually and structurally to best aid us going forward in both implementation into code as well as any diagrams we needed as its comprehensive and easy to follow.

Business Type Model:



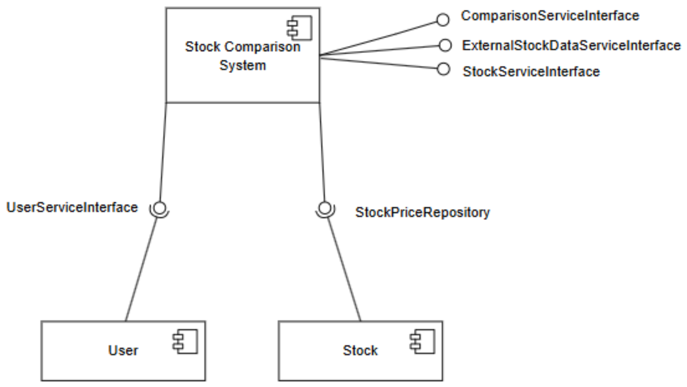
Business Type model: that we derived from the Business Concept Model, which includes a simplification of concepts that aren't unique or irrelevant, and also labels that define each component to describe what it is/the utility of it.

Business Interface:



Business Interface Model: We split the diagram into two components that we can use and extended them with the chosen interfaces that were used in our code.

Initial System Architecture:



Initial System Architecture: Shows the structure of the "Comparison System," which uses interfaces (IComparePrice, ISetDateRange, ISearchShares, IExport/SaveData) and is connected to two managers: Data Manager and User Manager, handling their respective operations for data and user management.

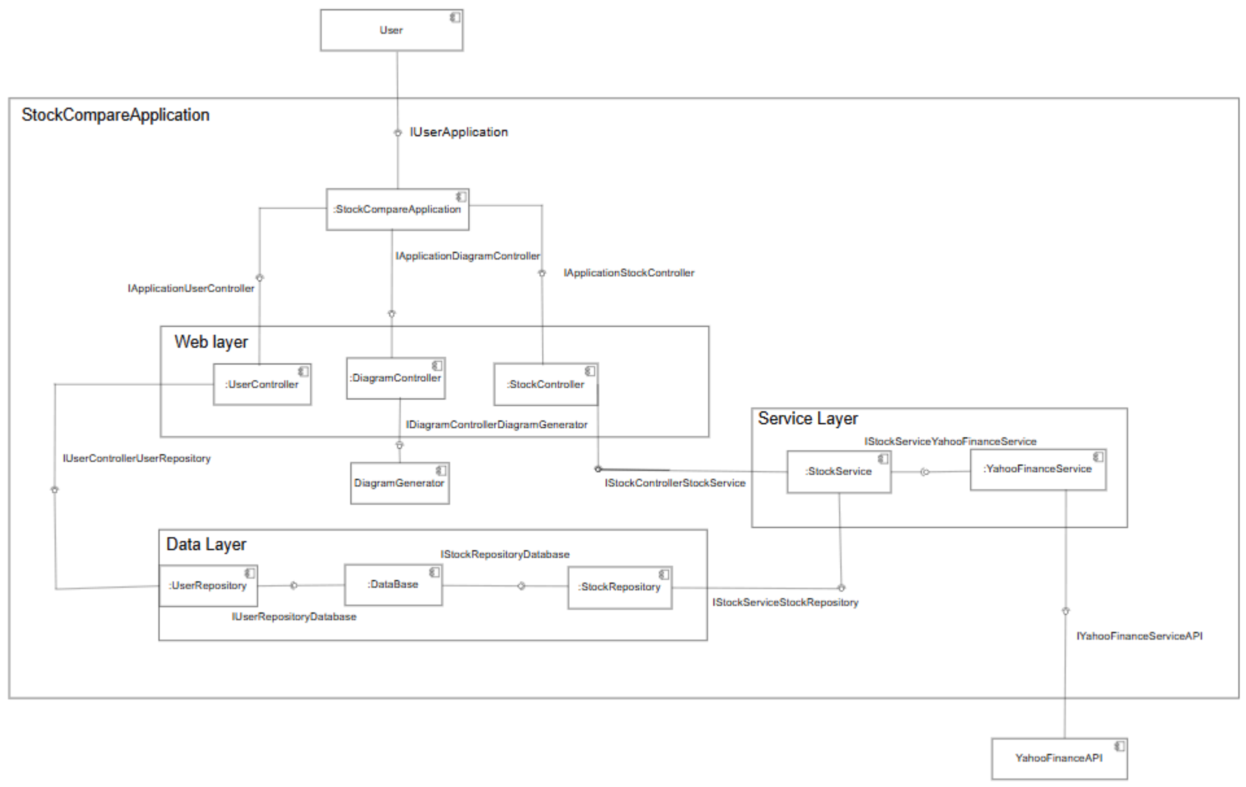
Sprint 3

Sprint 3 Goal:

For our final sprint, our main goal is to ensure that all pages of the website are fully developed and that a real-time stock API is successfully integrated. At this stage, we have all the necessary diagrams and resources prepared, so our focus will shift to implementing the specification, including the MVC architecture and other key requirements. As the Product Owner (Niwhar) and Scrum Master (Enea), we will oversee task assignments and ensure that the team remains on track, completing their work on time.

Sprint 3 Diagrams:

Compound Components:



The diagram shows the layered architecture of the StockCompareApplication, divided into Web, Service, and Data layers. The Web layer handles user input through controllers, the Service layer processes logic and fetches external data via the Yahoo Finance API, and the Data layer manages database interactions through repositories. This structure ensures modularity and maintainability.

This application follows a well-structured, maintainable architecture by combining layered architecture, Service-Oriented Architecture (SOA), and key design patterns. Responsibilities are clearly separated across presentation, controller, service, data access, and domain layers. Core business logic is centered around domain models like *Stock*, *StockPrice*, and *UserProfile*, with other components built around them in accordance with Clean Architecture and the dependency rule.

The use of the Repository pattern abstracts data access, and Dependency Injection (via Spring) improves testability. The system aligns with SOLID principles—ensuring single responsibility, easy extensibility, and proper abstraction through interfaces. Services and integrations are modular and loosely coupled, while controllers serve as interface adapters that route external requests to business logic. Cross-cutting concerns like logging and validation are handled consistently, resulting in a scalable, testable, and framework-independent design that supports long-term growth and maintainability.

The architecture centers around core domain models like Stock, StockPrice, and UserProfile, with all dependencies flowing inward—adhering to Clean Architecture principles. It also strongly follows the SOLID principles:

- Single Responsibility (SRP): Each class has one clear job—StockService manages stock operations, ComparisonService handles comparisons, UserService manages user-related logic, each repository deals with a specific entity, and each controller handles a distinct request type.

- Open/Closed (OCP): The system is open to extension but closed to modification. For instance, new data providers can be added via ExternalStockDataServiceInterface without altering existing logic, and service/repository interfaces allow for custom implementations.

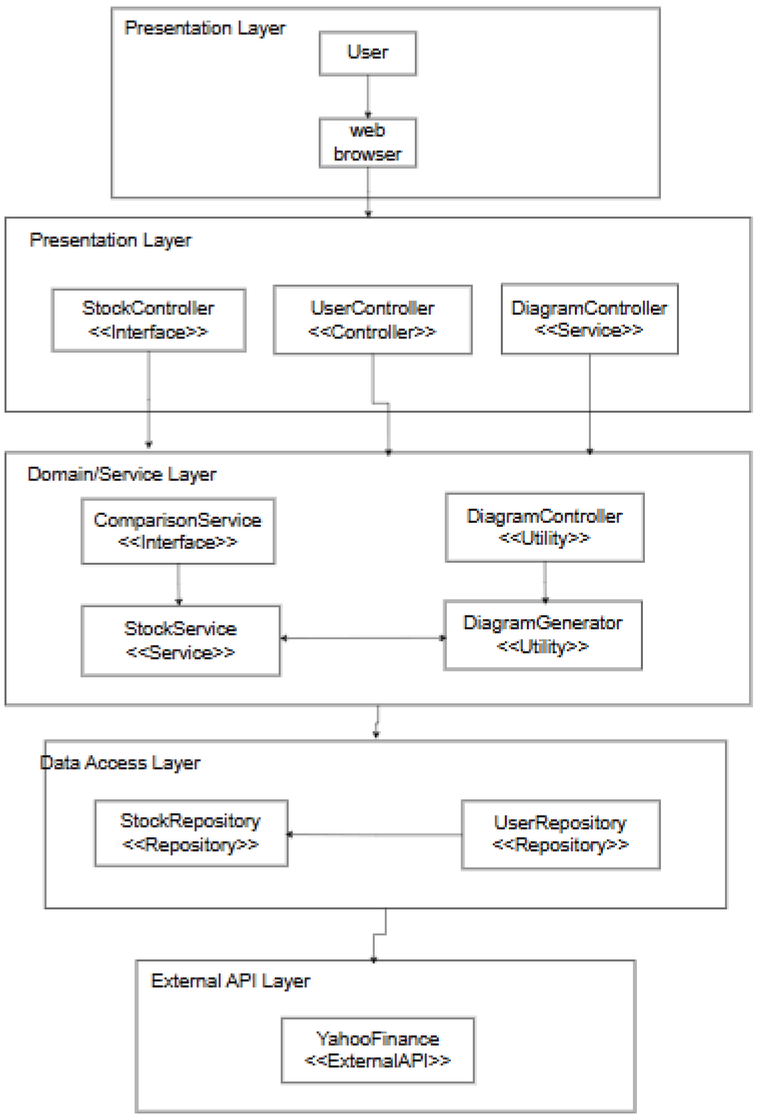
- Liskov Substitution (LSP): All implementations can be swapped seamlessly—e.g., AlphaVantageService and YahooFinanceService both work via the shared ExternalStockDataServiceInterface, and service classes can be used through their interfaces without breaking functionality.

- Interface Segregation (ISP): Interfaces are specific and focused—StockServiceInterface, UserServiceInterface, ComparisonServiceInterface, and ExternalStockDataServiceInterface each define only what's needed for their context.

- Dependency Inversion (DIP): High-level modules like services and controllers depend on interfaces, not concrete classes—keeping the system decoupled and flexible. Repositories also depend only on domain models, maintaining clean abstraction layers.

This results in a codebase that is modular, maintainable, testable, and easy to extend.

N-Tier:

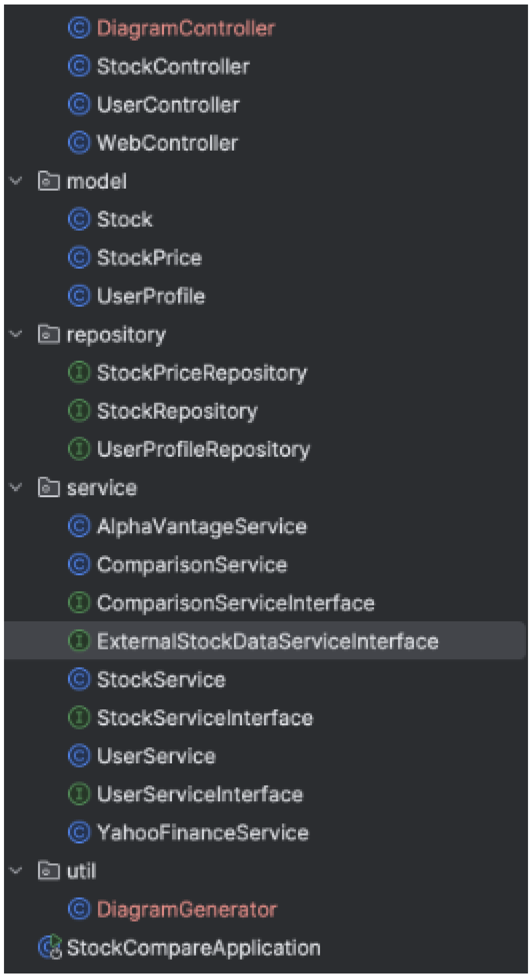

Presentation Layer
User
web browser
Presentation Layer
StockController <<Interface>>
UserController <<Controller>>
DiagramController <<Service>>
Domain/Service Layer
ComparisonService <<Interface>>
DiagramController <<Utility>>
StockService <<Service>>
DiagramGenerator <<Utility>>
Data Access Layer
StockRepository <<Repository>>
UserRepository <<Repository>>
External API Layer
YahooFinance <<ExternalAPI>>

1. Presentation Layer:
The user interacts with the system through a web browser.
2. Presentation Layer (Controllers):
Handles user requests via controllers like StockController, UserController, and DiagramController.
3. Domain/Service Layer:
Implements core logic with services (StockService, ComparisonService) and utility classes (DiagramController, DiagramGenerator).

4. Data Access Layer:
Handles database interactions using repositories (StockRepository, UserRepository).
5. External API Layer:
Fetches external stock data through services like YahooFinance.

Architectural Patterns:

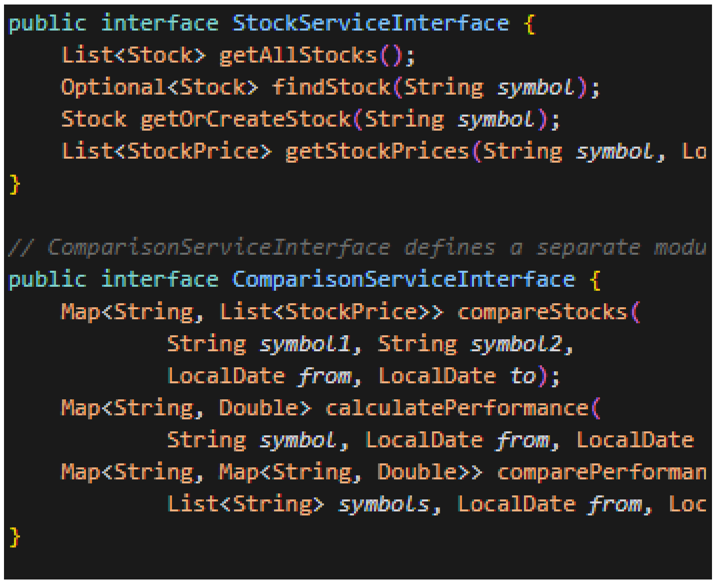
MVC:



```
    DiagramController
    StockController
    UserController
    WebController
  model
    Stock
    StockPrice
    UserProfile
  repository
    StockPriceRepository
    StockRepository
    UserProfileRepository
  service
    AlphaVantageService
    ComparisonService
    ComparisonServiceInterface
    ExternalStockDataServiceInterface
    StockService
    StockServiceInterface
    UserService
    UserServiceInterface
    YahooFinanceService
  util
    DiagramGenerator
  StockCompareApplication
```

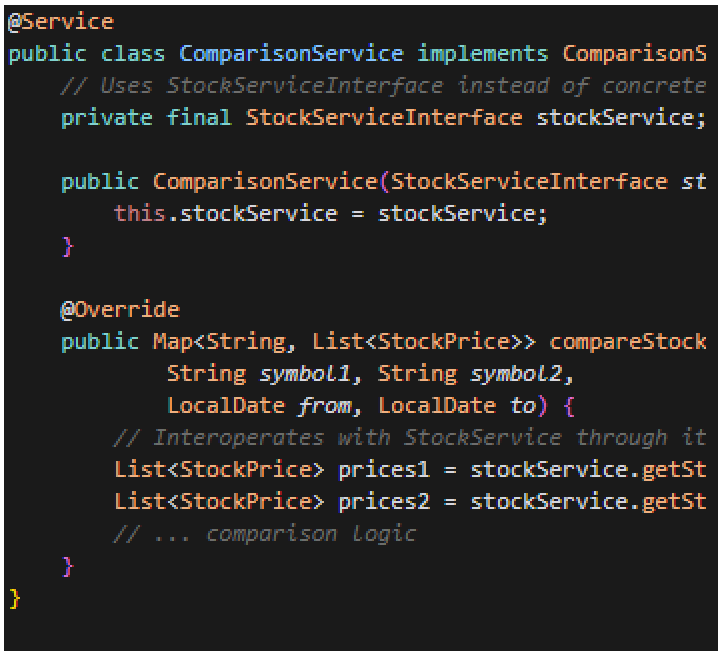
The project follows the Model-View-Controller (MVC) architecture to ensure a clear separation of concerns and maintainable code structure. The Model layer includes classes like Stock, StockPrice, and UserProfile, which represent the core data and business logic. The View is managed through controllers such as StockController and WebController, which handle incoming requests and return appropriate responses, typically to the frontend. The Controller layer interacts with services and repositories to process data and deliver it to the user. Services like StockService and ComparisonService encapsulate business logic, while repositories handle communication with the database. This structure promotes modularity, making it easier to develop, test, and scale the application.

Modularity:



```
public interface StockServiceInterface {
    List<Stock> getAllStocks();
    Optional<Stock> findStock(String symbol);
    Stock getOrCreateStock(String symbol);
    List<StockPrice> getStockPrices(String symbol, Lo
}

// ComparisonServiceInterface defines a separate modu
public interface ComparisonServiceInterface {
    Map<String, List<StockPrice>> compareStocks(
            String symbol1, String symbol2,
            LocalDate from, LocalDate to);
    Map<String, Double> calculatePerformance(
            String symbol, LocalDate from, LocalDate
    Map<String, Map<String, Double>> comparePerforman
            List<String> symbols, LocalDate from, Loc
}
```

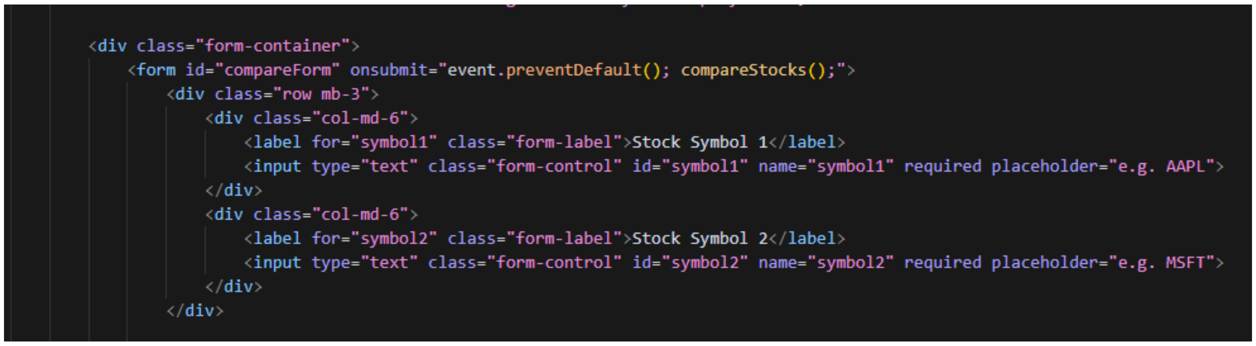
Interoperability:



```
@Service
public class ComparisonService implements ComparisonS
    // Uses StockServiceInterface instead of concrete
    private final StockServiceInterface stockService;

    public ComparisonService(StockServiceInterface st
        this.stockService = stockService;
    }

    @Override
    public Map<String, List<StockPrice>> compareStock
            String symbol1, String symbol2,
            LocalDate from, LocalDate to) {
        // Interoperates with StockService through it
        List<StockPrice> prices1 = stockService.getSt
        List<StockPrice> prices2 = stockService.getSt
        // ... comparison logic
    }
}
```

Presentation Layer:

compare.html



```
<div class="form-container">
    <form id="compareForm" onsubmit="event.preventDefault(); compareStocks();">
        <div class="row mb-3">
            <div class="col-md-6">
                <label for="symbol1" class="form-label">Stock Symbol 1</label>
                <input type="text" class="form-control" id="symbol1" name="symbol1" required placeholder="e.g. AAPL">
            </div>
            <div class="col-md-6">
                <label for="symbol2" class="form-label">Stock Symbol 2</label>
                <input type="text" class="form-control" id="symbol2" name="symbol2" required placeholder="e.g. MSFT">
            </div>
        </div>
```

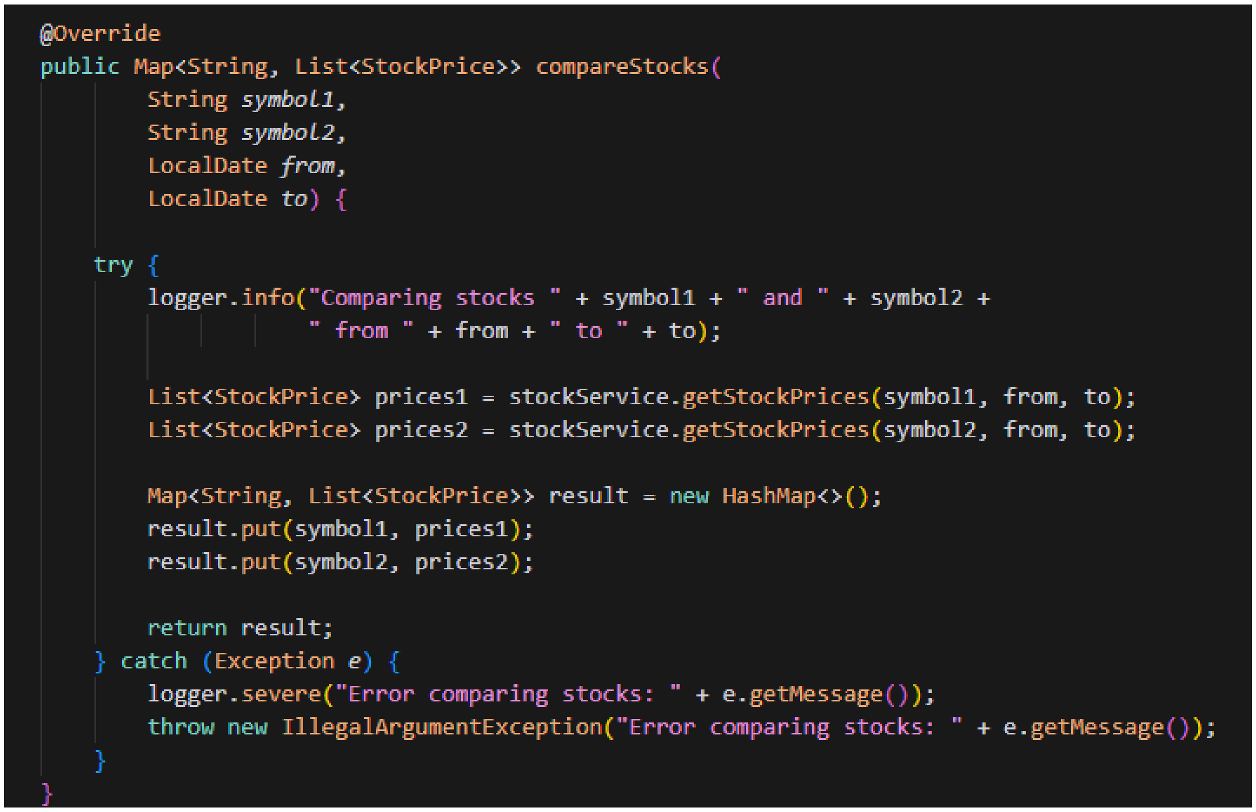
This HTML code creates a form with two input fields for users to enter stock ticker symbols (e.g., AAPL, MSFT). When submitted, the form prevents the default page reload and triggers a JavaScript function called compareStocks() to handle the comparison.

↳Also in WebController.java, StockController.java, UserController.java

↳ index.html, result.html

Business Logic Layer:

Example from ComparisonService.java:

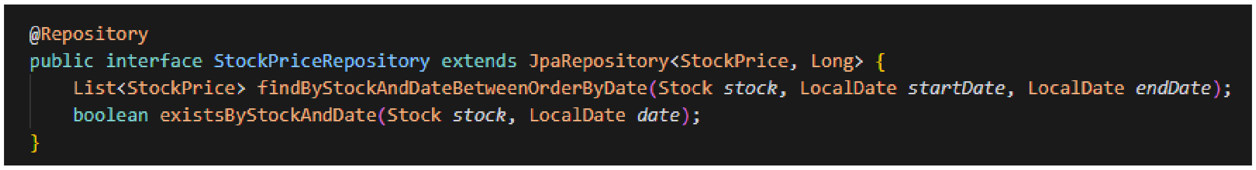


```java
@Override
public Map<String, List<StockPrice>> compareStocks(
        String symbol1,
        String symbol2,
        LocalDate from,
        LocalDate to) {

    try {
        logger.info("Comparing stocks " + symbol1 + " and " + symbol2 +
                    " from " + from + " to " + to);

        List<StockPrice> prices1 = stockService.getStockPrices(symbol1, from, to);
        List<StockPrice> prices2 = stockService.getStockPrices(symbol2, from, to);

        Map<String, List<StockPrice>> result = new HashMap<>();
        result.put(symbol1, prices1);
        result.put(symbol2, prices2);

        return result;
    } catch (Exception e) {
        logger.severe("Error comparing stocks: " + e.getMessage());
        throw new IllegalArgumentException("Error comparing stocks: " + e.getMessage());
    }
}
```

This method compares two stocks by retrieving their price data over a specified date range using a stock service and returns the results in a map. It logs the process and handles any exceptions by logging the error and throwing an IllegalArgumentException.

Data Access Layer:

Classes included: StockPriceRepository.java:

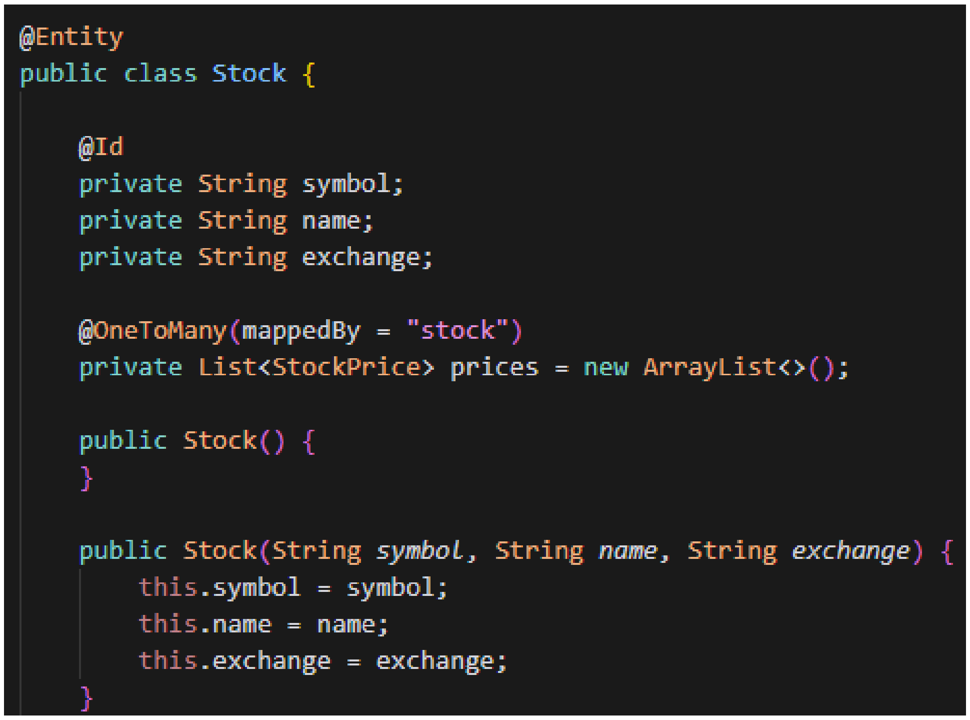


```java
@Repository
public interface StockPriceRepository extends JpaRepository<StockPrice, Long> {
    List<StockPrice> findByStockAndDateBetweenOrderByDate(Stock stock, LocalDate startDate, LocalDate endDate);
    boolean existsByStockAndDate(Stock stock, LocalDate date);
}
```

This code defines a Spring Data JPA repository interface StockPriceRepository for managing StockPrice entities. It includes two custom query methods: one to retrieve stock prices between two dates for a specific stock, and another to check if a stock price exists for a specific date.

Domain Layer:

Classes used: Stock.java

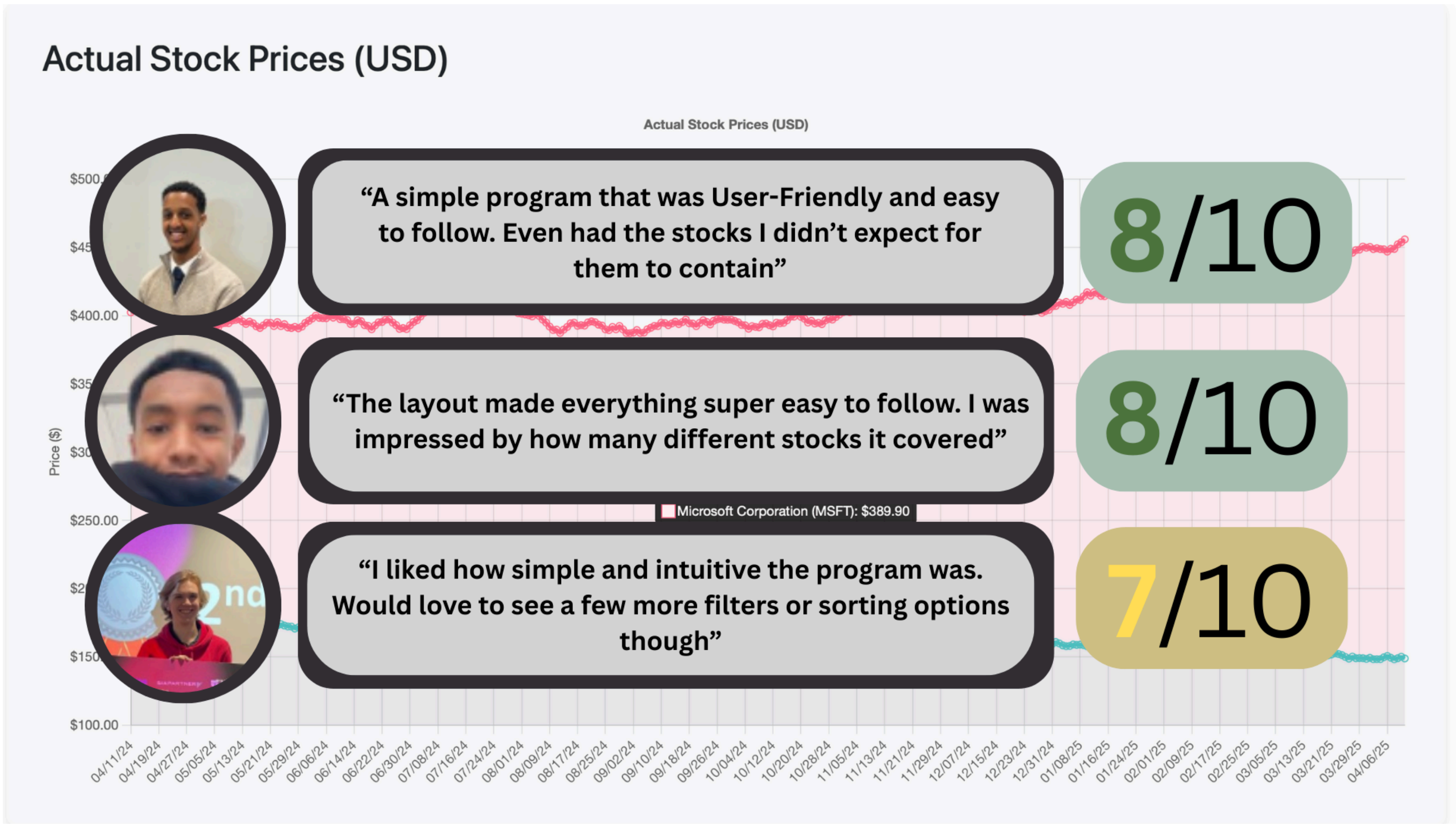


```java
@Entity
public class Stock {

    @Id
    private String symbol;
    private String name;
    private String exchange;

    @OneToMany(mappedBy = "stock")
    private List<StockPrice> prices = new ArrayList<>();

    public Stock() {
    }

    public Stock(String symbol, String name, String exchange) {
        this.symbol = symbol;
        this.name = name;
        this.exchange = exchange;
    }
```

This code defines a JPA entity class Stock representing a stock with fields for symbol, name, and exchange, where symbol is the primary key. It also establishes a one-to-many relationship with the StockPrice entity through the prices list, meaning one stock can have multiple price entries.

Test Cases:

We used 3 Students to test our program and this was the feedback we received.


Actual Stock Prices (USD)
Actual Stock Prices (USD)
Price ($)
Microsoft Corporation (MSFT): $389.90
8/10
8/10
7/10

"A simple program that was User-Friendly and easy to follow. Even had the stocks I didn't expect for them to contain"

"The layout made everything super easy to follow. I was impressed by how many different stocks it covered"

"I liked how simple and intuitive the program was. Would love to see a few more filters or sorting options though"

Meeting Minutes:

Sprint 1 meeting minutes:

Dates and Time	**03/02/2025 at 8:00pm** **05/02/2025 at 7:30pm** **10/02/2025 at 8:00pm** **13/02/2025 at 7:00pm** **15/02/2025 at 8:10pm** **18/02/2025 at 6:00pm**
Project Name	**YANE-STOCK**
Meeting Goal	**03/02/2025 at 8:00pm = understanding the project and what is asking us to do and come up with ideas** **05/02/2025 at 7:30= we discussed our ideas and we voted to choose our topic and done research** **10/02/2025 at 8:00pm = after everyone did their research, we assigned each other tasks fairly.** **13/02/2025 = short meeting to updated each other and made sure no one was struggling** **15/02/2025 = again another short meeting, this time everyone was close to finishing** **18/02/2025 = Finishing up on pdf and finalising the code**

Facilitator	Niwhar
Note taker	03/02/2025 at 8:00pm -Enea 05/02/2025 at 7:30pm - Yulian 10/02/2025 at 8:00pm - Adam 13/02/2025 at 7:00pm -Niwhar 15/02/2025 at 8:10pm -Yulian 18/02/2025 at 6:00pm - Enea
Attendees	03/02/2025 at 8:00pm = everyone was present, but Enea and Yulian were late 05/02/2025 at 7:30pm = everyone was present and on time 10/02/2025 at 8:00pm = everyone was present and on time 13/02/2025 7:00pm = everyone was present, but Adam had to leave early because of personal issues 15/02/2025 8:10pm = everyone was present and on time 18/02/2025 at 6:00pm = everyone was present and on time

Roundtable Updates (each group member to contribute)	**03/02/2025 at 8:00pm = we just went through the description and made sure everyone understood and gave ourselves the task to come up with ideas for our next meeting.** **05/02/2025 = after a few days everyone had interesting plans and ideas to share and we all voted which one sounded the most interesting, this really helped us getting started with this sprint, after we finalised our topic and our scopes for this project, we each done our research in the company we chose to be as ready as possible to start the work** **10/02/2025 at 8:00pm = we decided to wait 5 days to make our next meeting as we had another coursework to submit in between these 5 days, this meeting was all about assigning tasks fairly to everyone and made sure not to give a task to someone who clearly isn't well suited for it so we can more efficient as a group.** **13/02/2025 = this meeting was just about updating each other with where we got up to with our assigned tasks, thankfully no one was struggling and kept this meeting short sand brief.** **15/02/2025 = this meeting was also short and brief, we just updated each other on our progress, checking in if anyone needs help with anything.** **18/02/2025 = this meeting we finalised everything, made sure every task was done correctly and that we have met all the criteria's to ensure we the highest grade, after that was done we briefly discussed our plan for sprint 2 just so we can get an idea what we're dealing with.**

Discussion points	**03/02/2025 at 8:00pm = Understanding the project, setting name** **05/02/2025 at 7:30pm = Sharing our ideas** **10/02/2025 at 8:00pm = Task allocation** **13/02/2025 at 7:00pm = Update and research discussion** **15/02/2025 at 8:10pm = Looking at ideas regarding the code and how to approach using the diagrams** **18/02/2025 at 6:00pm = Finalisation of code and PDF, and double checking all work. Make all commits from group doc and ensure all commits successful.**
Actions (list tasks and assign a group member)	**Adam = he was in charge of making the groups code of conduct and to record what the group contributed and did during the meetings as well as record the day and time the meetings took place.** **Niwhar = he was in charge of making a github and setting tasks for everyone and made sure to allocate each member with a task he knew they could accomplish.** **Enea = in charge of setting up meetings, seeing that everyone is meeting the progress they are supposed to and doing tasks they were set, done planning behind the team's next steps towards their overall goal** **Yulian = in charge of documenting technical decisions in the group and describing choices made explaining the decisions taken towards our final scope. In charge of the initial code concept/structure planning (with group ideas)**

Sprint 2 Meeting Minutes:

Dates and Time	**10/03/2025 at 8:00pm** **13/03/2025 at 7:00pm** **15/03/2025 at 8:10pm** **18/03/2025 at 6:00pm**
Project Name	**YANE-STOCK**
Meeting Goal	**08/03/2025 at 7:00pm = Understanding Sprint 2 requirements based on feedback from Sprint 1, setting out updated project goals, and discussing the business concept model and use case diagrams.** **12/03/2025 at 6:30pm = Mid-sprint check-in to review progress and realign tasks. We reviewed system interfaces, business models, and discussed architecture implementation.** **17/03/2025 at 7:15pm = Quick catch-up to ensure all components were nearly finished, including any backend/frontend integration and preparing for final implementation.** **23/03/2025 at 5:30pm = Finalisation of documentation and code. Ensured GitHub was updated with clean commits, double-checked each member's contributions, and discussed Sprint 2 reflections.**
Facilitator	**Niwhar**

Note taker	08/03/2025 at 7:00pm – Enea 12/03/2025 at 6:30pm – Yulian 17/03/2025 at 7:15pm – Adam 23/03/2025 at 5:30pm – Niwhar
Attendees	08/03/2025 at 7:00pm = Everyone was present, Adam joined 10 minutes late due to other academic work. 12/03/2025 at 6:30pm = All present and actively engaged throughout. 17/03/2025 at 7:15pm = Everyone attended, Yulian experienced minor technical issues but was able to participate. 23/03/2025 at 5:30pm = All members were present and fully focused on wrapping up the sprint.
Roundtable Updates (each group member to contribute)	08/03/2025 at 7:00pm = We reviewed our Sprint 1 feedback and refined the business concept model and system use case diagram. Everyone contributed ideas on improving the project design and interface clarity. 12/03/2025 = Shared progress and made adjustments to the workload where needed. Discussed clean architecture principles and validated that system interfaces and data flows were in line with the concept model. 17/03/2025 = Everyone gave a quick update on their component status. Backend and frontend teams coordinated on integrating modules. Minor bugs were flagged for fixing before finalisation. 23/03/2025 = Each team member walked through their final tasks, confirming that all criteria were covered. Code was reviewed for consistency, documentation was completed, and the project was prepared for submission.

Discussion points	08/03/2025 at 7:00pm = Finalising business concept model, aligning it with the business type model, and re-evaluating the system's initial architecture. 12/03/2025 at 6:30pm = Verifying interface definitions, clean architecture layers, and updating our system design based on what has been implemented. 17/03/2025 at 7:15pm = Discussing integration strategy, test coverage, and final fixes. 23/03/2025 at 5:30pm = Ensuring all commits were done correctly, documentation aligned with code, and reflecting on what went well during Sprint 2.
Actions (list tasks and assign a group member)	Adam = Tasked with keeping records of each meeting including dates, and making sure logs reflected contributions for the assessment. Niwhar = Maintained the GitHub repository, created and assigned GitHub issues based on team strengths, and ensured team members followed the clean architecture approach. Enea = Responsible for organising meeting times, ensuring the group stayed on track with the sprint timeline, and coordinating the final plan for integration and documentation. Yulian = Handled documentation of all technical design decisions, maintained alignment between initial design and final codebase, and guided the planning for overall system architecture and component structure.

Sprint 3 Meeting Minutes:

Dates and Time	04/03/2025 at 4:00pm 08/03/2025 at 5:30pm 10/03/2025 at 8:00pm 15/03/2025 at 7:00pm 20/03/2025 at 8:10pm 18/02/2025 at 6:00pm
Project Name	YANE-STOCK
Meeting Goal	03/02/2025 at 8:00pm = understanding the project and what is asking us to do and come up with ideas 05/02/2025 at 7:30= we discussed our ideas and we voted to choose our topic and done research 10/02/2025 at 8:00pm = after everyone did their research, we assigned each other tasks fairly. 13/02/2025 = short meeting to updated each other and made sure no one was struggling 15/02/2025 = again another short meeting, this time everyone was close to finishing 18/02/2025 = Finishing up on pdf and finalising the code

Facilitator	**Niwhar**
Note taker	**03/02/2025 at 8:00pm -Enea** **05/02/2025 at 7:30pm - Yulian** **10/02/2025 at 8:00pm - Adam** **13/02/2025 at 7:00pm -Niwhar** **15/02/2025 at 8:10pm -Yulian** **18/02/2025 at 6:00pm - Enea**
Attendees	**18/02/2025 at 6:00pm – Held initial sprint planning to outline compound components and discuss reusable architecture.** **04/03/2025 at 4:00pm – Completed breakdown of architecture into compound components and planned domain-independent styles.** **08/03/2025 at 5:30pm – Implemented basic structure for N-tiered and MVC styles across key components.** **10/03/2025 at 8:00pm – Began implementing service modularity using SOA principles; connected two services via adapters.** **15/03/2025 at 7:00pm – Applied consistent reusable styles and completed the Layered architecture setup.** **20/03/2025 at 8:10pm – Conducted internal testing, logged test cases and documented initial results for the group report.**

Roundtable Updates (each group member to contribute)	**03/02/2025 at 8:00pm – Enea** **During this initial meeting, we kicked off the sprint planning by thoroughly discussing the project requirements and objectives. Our primary focus was on outlining the compound components we needed to create for the web application, ensuring that we were all aligned with the design and functionality goals. We also brainstormed ways to structure the architecture to make it reusable and scalable. By the end of the meeting, we had a clear understanding of the tasks ahead and the direction we needed to take for the sprint, along with a shared goal of ensuring the components would be flexible for future use.** **05/02/2025 at 7:30pm – Yulian** **In this session, we made significant progress by breaking down the overall architecture into individual compound components. Each component was considered with reusability in mind, ensuring that we could easily integrate them into different parts of the system as needed. We also delved into domain-independent styles, researching various architectural patterns we could apply, like MVC (Model-View-Controller) and N-tiered systems. This planning session laid a solid foundation for the next steps in the sprint, as everyone was clear on the individual components and the best ways to implement them.** **10/02/2025 at 8:00pm – Adam** **This meeting focused on implementing the basic structure for two key architectural styles: N-tiered architecture and MVC. The goal was to set up the foundation for the system, making sure that the data flow and component separation followed best practices. We worked on building out the backend and front-end components according to the N-tiered architecture, which separates the application into logical layers, and started implementing the MVC framework to organize the interaction between the user interface, business logic, and data. This was a critical step in creating a robust and scalable foundation for the rest of the project.** **13/02/2025 at 7:00pm – Niwhar** **In this session, we began focusing on service modularity, which is a core principle of Service-Oriented Architecture (SOA). The aim was to break down the application into discrete services that could communicate with each other efficiently. Niwhar led the effort to**

	implement adapters to connect two services and ensure smooth interoperability. We discussed how to structure the services to promote modularity and scalability, and how these services could be reused across different parts of the application. This meeting was crucial in understanding how to make the application more flexible and capable of handling future growth. **15/02/2025 at 8:10pm – Yulian** During this meeting, we focused on applying consistent, reusable styles across the components. Yulian led the effort to finalize the Layered architecture, which focuses on organizing the system into distinct layers, each with its own responsibility. This approach allows for better scalability and maintainability. We also reviewed the different styles that had been implemented so far, ensuring that the design was cohesive and aligned with the principles of reusability. This was a key step in ensuring that the architecture could grow and adapt as the project progresses. **18/02/2025 at 6:00pm – Enea** By this stage, we had completed much of the work and conducted our first round of internal testing. During this meeting, Enea led the effort to test the implemented system, checking whether the components functioned as expected and whether the integration between them was smooth. We identified and logged test cases to ensure that the system met the requirements, and any issues or bugs were noted for fixing. The results from the tests were documented thoroughly, and we discussed what still needed to be addressed for the next phase of the project. This meeting was important for evaluating our progress and ensuring that everything was on track for the final stages of the sprint.
Discussion points	- Discuss distribution of tasks - Review feedback from sprint 2 - Plan for integration and consistency of our diagrams - Discuss use of APIs, integrating them into the app - Talk

Actions (list tasks and assign a group member)	Adam = Niwhar = Ensured that MVC structure was followed in the code Enea = Yulian =