PROJECT REPORT

STOCK TRADING AND INVENTORY MANAGEMENT

OOAD MINI PROJECT

Submitted by:

ALAN S PAUL	**PES1UG20CS624**
B.R SINCHANA	**PES1UG20CS630**
BASAVAPRABHU G K	**PES1UG20CS631**
ADITYA S RAJ	**PES1UG20CS622**

Under the guidance of

Prof Bhargavi Mokashi
PES University

January - May 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Synopsis:

The objective of this project is to develop a web-based stock trading application using Java and Spring Boot MVC architecture that enables users to buy and sell stocks. The application will provide users with a seamless trading experience and enable them to make informed investment decisions. The application will also allow users to track their portfolio performance, view stock prices, and receive alerts on price changes.

The following are the use cases that will be implemented in the application:

1. User Registration and Login: The application will allow users to register and create an account. Registered users will be able to log in to the system using their credentials.

2. Stock Inventory Management: The application will maintain a database of all available stocks, their current prices, and the quantity of shares available for purchase. The stock price updates will not be in real-time but will be updated regularly based on scheduled batch jobs.

3. Buy Shares: Users will be able to search for stocks and buy shares. The application will verify that the user has enough funds to buy shares and that there are enough shares available to purchase. The application will deduct the appropriate amount from the user's account balance and update the user's portfolio with the new shares.

4. Sell Shares: Users will be able to sell shares. The application will update the user's account balance and portfolio with the sold shares.

The application will also have the following minor use cases:

1. Shares Quantity Backend Authentication: The application will verify that the user has enough shares to sell.

2. User Login Backend Authentication: The application will validate user credentials against the database to ensure that only registered users can access the system.

3. Server Session Management: The application will manage user sessions on the server side to ensure security and prevent unauthorized access.

4. Server Redirection and User Logout: The application will redirect users to the appropriate pages based on their actions, and enable users to log out of the system securely.

It is to be noted that the usage of sessions and redirections would make state and activity diagrams redundant for our demonstrated application.

Hence, relevant Use Case Diagrams and Class Modules have been shown.

Design Principles Identified

Single Responsibility Principle (SRP):

- By following SRP, the code becomes easier to read, understand, and maintain over time.
- SRP ensures that each class or module is focused on doing one thing well, which reduces the chances of bugs and makes it easier to test.

Don't Repeat Yourself (DRY):

- DRY promotes code reusability, making it easier to change code in one place without affecting other parts of the application.
- DRY helps to avoid inconsistencies in the code by ensuring that each piece of information has only one authoritative source.

Dependency Inversion Principle (DIP):

- DIP promotes loose coupling, which means that changes to one part of the system do not impact other parts of the system.
- DIP makes it easier to switch out implementations of a particular interface, making the code more modular and flexible.

Interface Segregation Principle (ISP):

- ISP ensures that clients only depend on the methods they need, which helps to avoid the problem of unnecessary dependencies.
- ISP allows for greater modularity and flexibility in the system, making it easier to change and maintain over time.

Design Patterns Identified

Model-View-Controller (MVC) Design Pattern:

- MVC separates concerns between the data, the presentation, and the business logic, making the code easier to maintain and test.
- MVC makes it easier to develop and maintain large applications by separating the code into distinct layers.

Singleton Pattern:

- Singleton pattern ensures that a class has only one instance throughout the lifetime of the application, which helps to conserve resources.
- Singleton pattern provides a global point of access to the object, making it easier to manage and maintain over time.

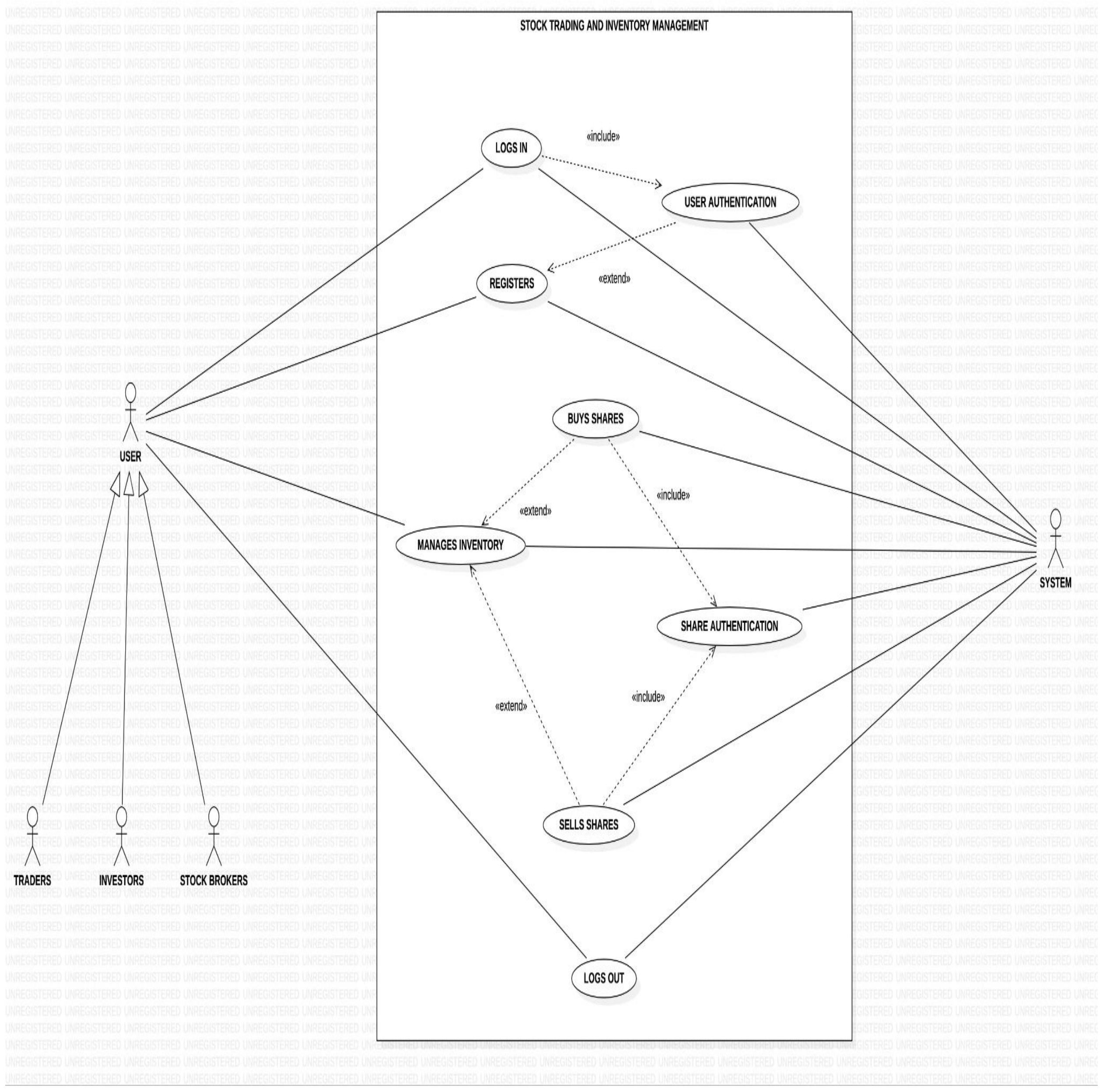
Data Access Object (DAO) Pattern:

- DAO pattern provides a layer of abstraction between the application and the database, which makes it easier to change the database or the data access layer without affecting the rest of the application.
- DAO pattern separates the data access logic from the business logic, making the code easier to test and maintain.

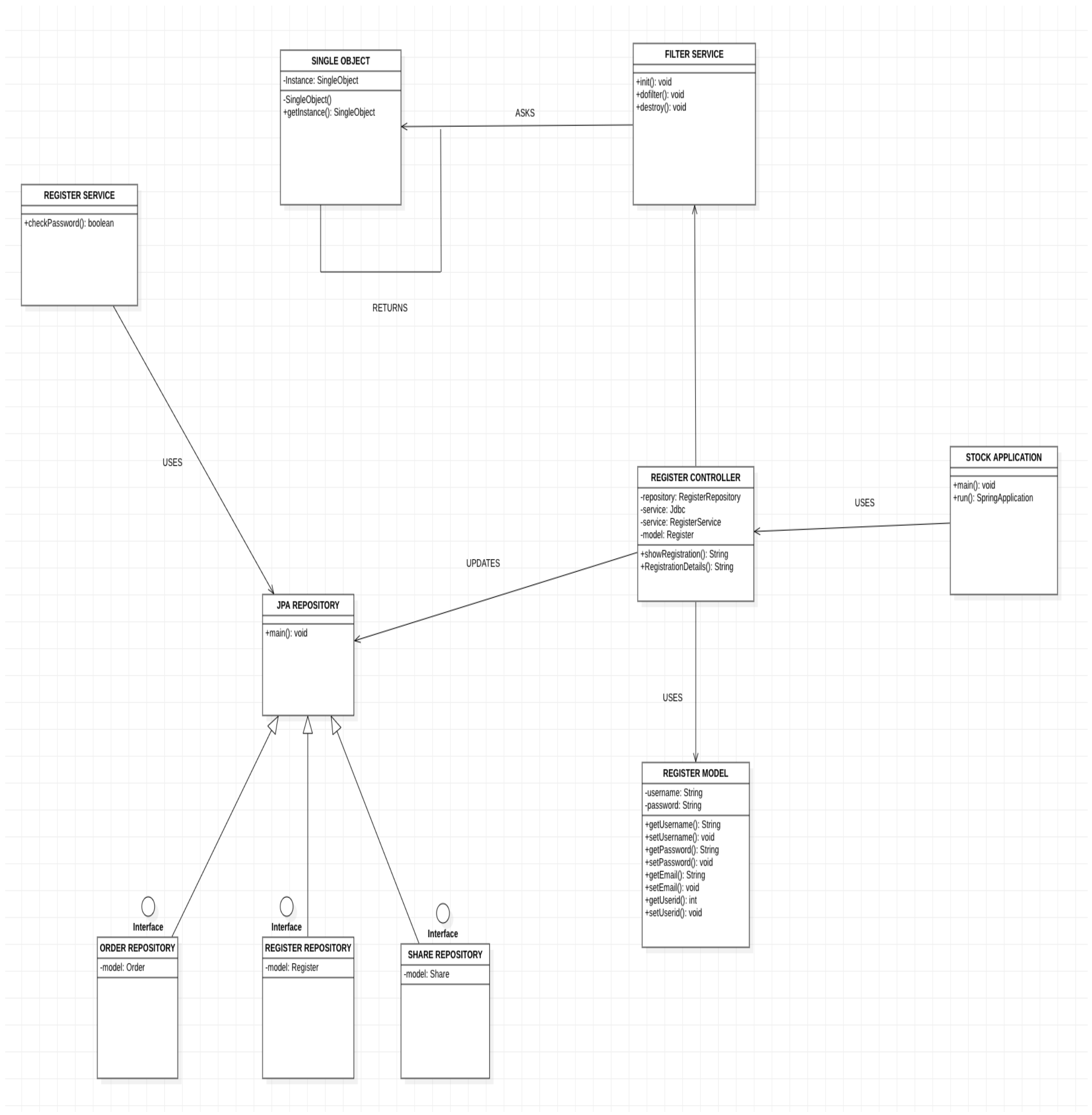
Chain of Responsibility Pattern:

- Chain of Responsibility pattern promotes loose coupling between objects, making it easier to add, remove or modify objects in the chain without affecting the rest of the system.
- Chain of Responsibility pattern allows for dynamic handling of requests, making it easier to change the behavior of the system at runtime.

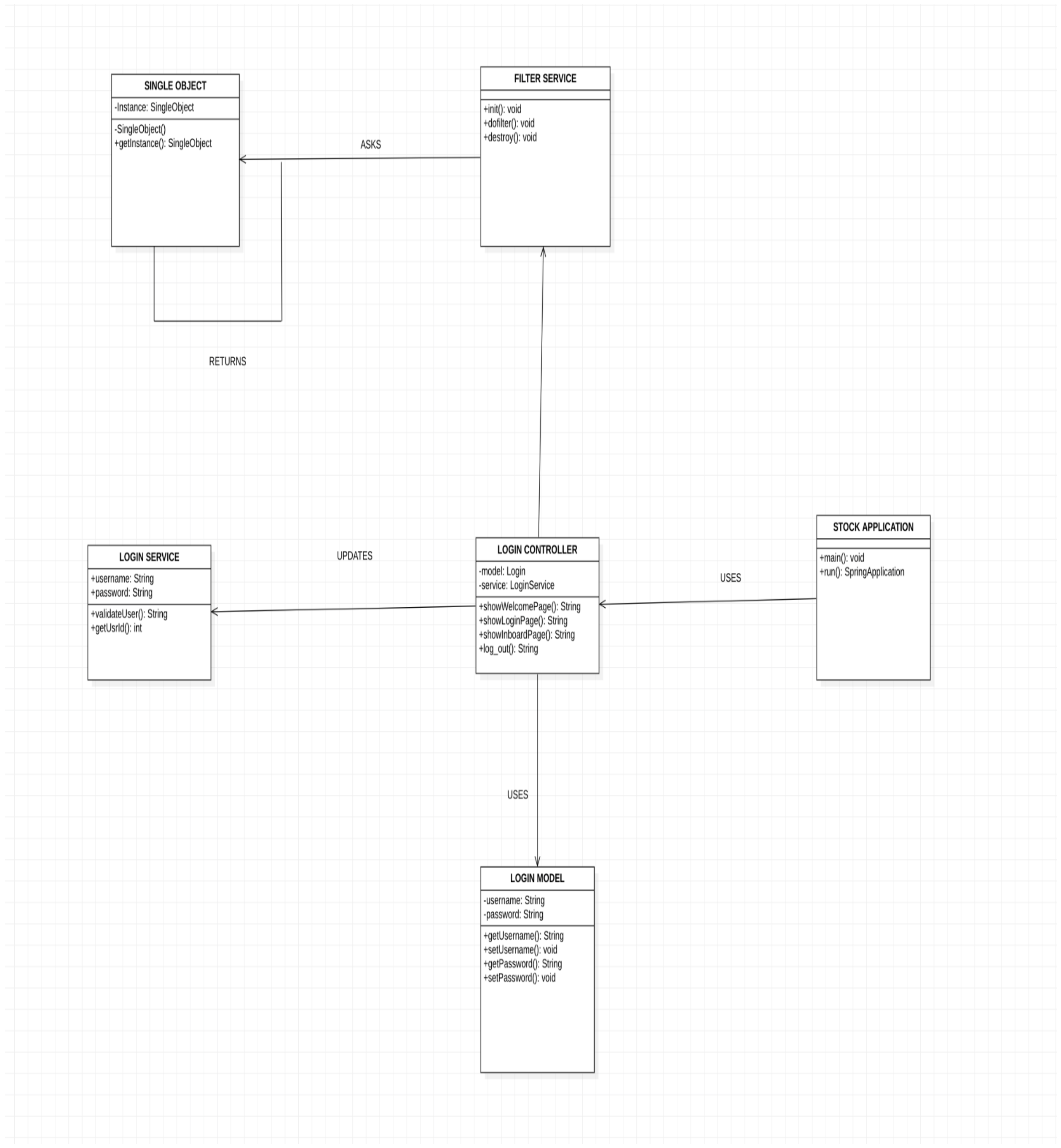
USE CASE DIAGRAM



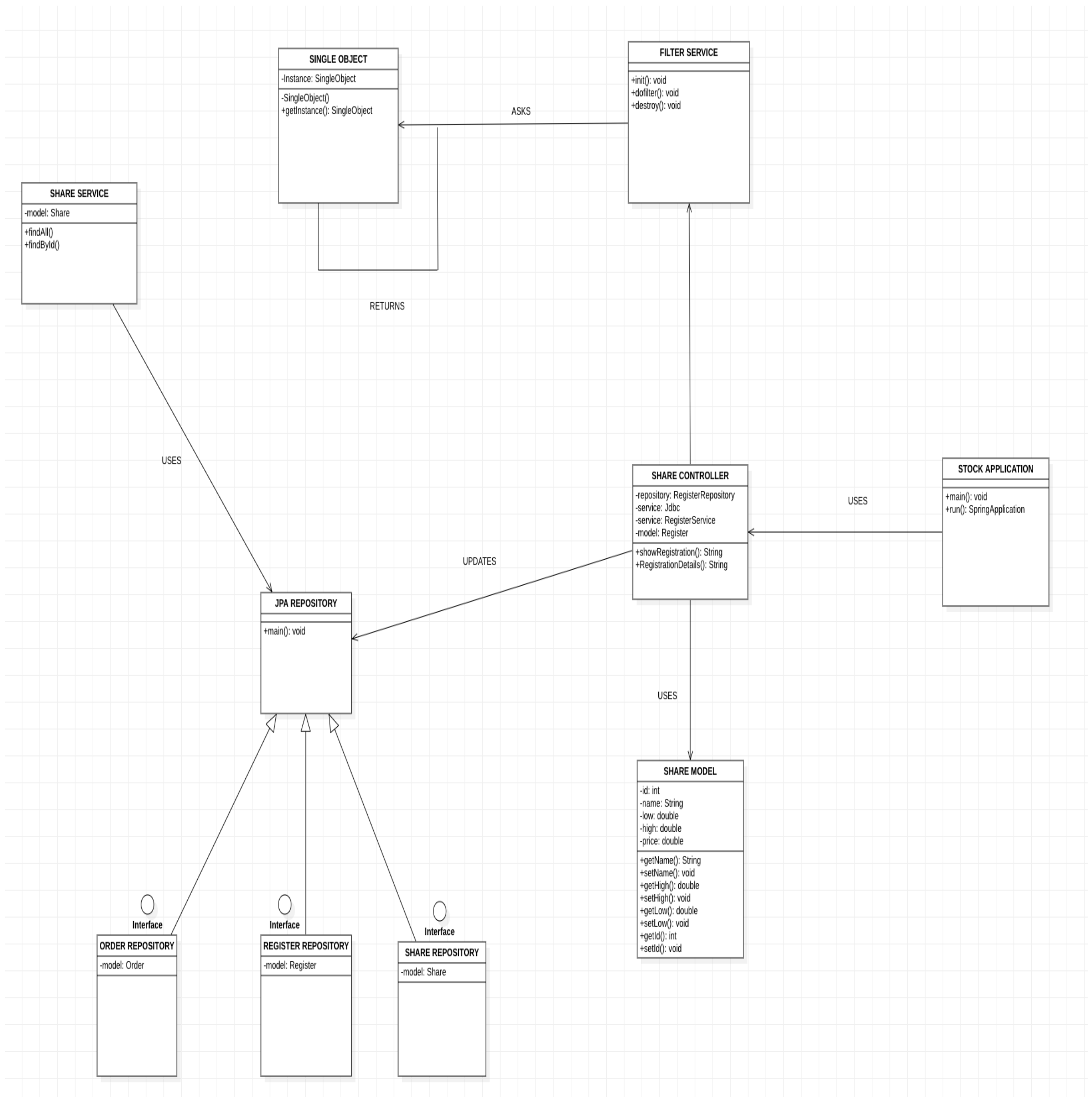
CLASS DIAGRAM FOR REGISTER



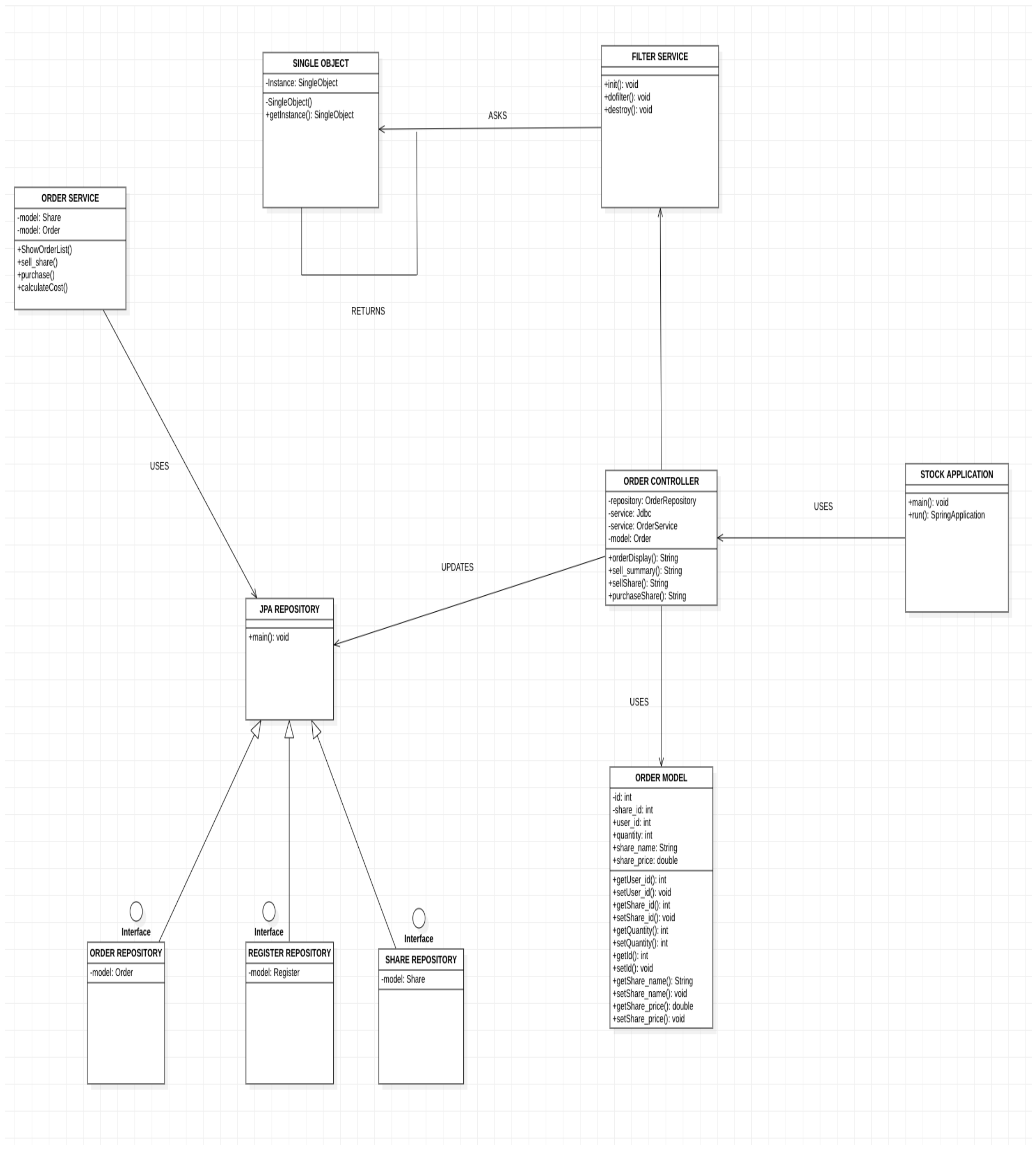
CLASS DIAGRAM FOR LOGIN



CLASS DIAGRAM FOR SHARE



<u>CLASS DIAGRAM FOR ORDER</u>


SINGLE OBJECT
-Instance: SingleObject
-SingleObject()
+getInstance(): SingleObject
FILTER SERVICE
+init(): void
+dofilter(): void
+destroy(): void
ASKS
RETURNS
ORDER SERVICE
-model: Share
-model: Order
+ShowOrderList()
+sell_share()
+purchase()
+calculateCost()
USES
ORDER CONTROLLER
-repository: OrderRepository
-service: Jdbc
-service: OrderService
-model: Order
+orderDisplay(): String
+sell_summary(): String
+sellShare(): String
+purchaseShare(): String
USES
STOCK APPLICATION
+main(): void
+run(): SpringApplication
UPDATES
JPA REPOSITORY
+main(): void
USES
ORDER MODEL
-id: int
-share_id: int
+user_id: int
+quantity: int
+share_name: String
+share_price: double
+getUser_id(): int
+setUser_id(): void
+getShare_id(): int
+setShare_id(): void
+getQuantity(): int
+setQuantity(): int
+getId(): int
+setId(): void
+getShare_name(): String
+setShare_name(): void
+getShare_price(): double
+setShare_price(): void
Interface
ORDER REPOSITORY
-model: Order
Interface
REGISTER REPOSITORY
-model: Register
Interface
SHARE REPOSITORY
-model: Share