**Design for Online Trade Brokerage System**

Use case: brokerage application receives huge number of requests from different users on multiple channels. These requests are always come from multiple platforms such as mobile, website or over a call etc.

An Online Trade Brokerage System facilitates its users the trade (i.e. buying and selling) of shares online.

**System Requirements**

We will focus on the following set of requirements while designing the online trade brokerage system:

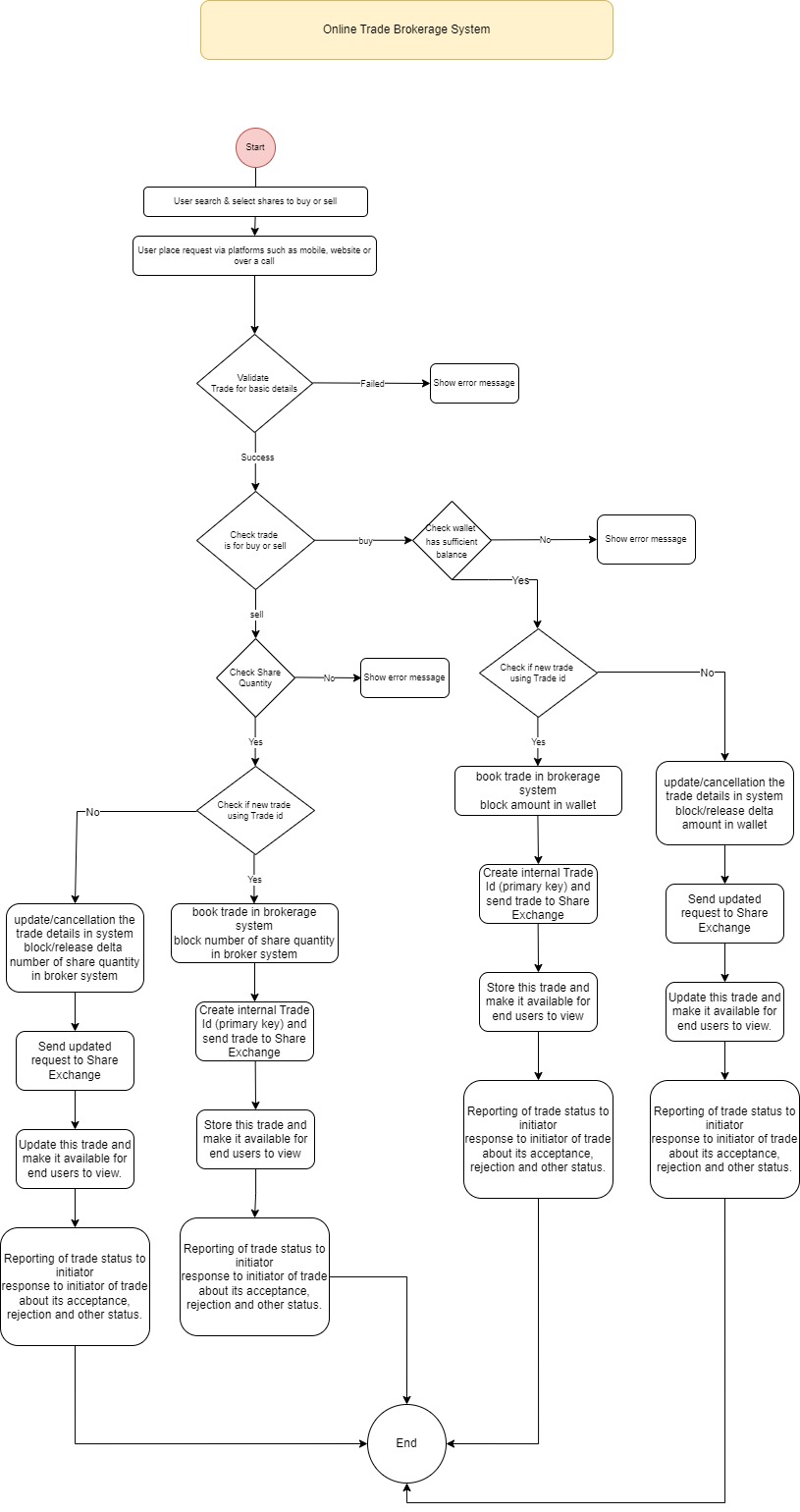
1. Authenticate and Authorise user of our system should be able to perform below actions:

buy and sell stocks, check wallet balance, check shares status, get confirmation of order,

1. Users should be able to place share trade orders of the from multiple platforms such as mobile, website or over a call etc.
2. The system should be able to generate reports, statements.
3. Users should be able to deposit and withdraw money either via check, wire, or electronic bank transfer.
4. The system should be able to send notifications whenever trade orders are executed.
5. High availability
6. Volume of trade requests such as 100K in an hour.
7. Deployment strategy
8. Database and why?
9. Resiliency
10. Monitoring system

## To implement this requirement, I will be using **Chain of Responsibility Pattern** from type Behavioural Design Pattern. Reason to use this pattern as below:

1. request is processed sequentially by a chain of handlers until one of them handles it. In brokerage system mentioned tasks are done in given sequential order.
2. In a microservices architecture, this pattern can be used to decouple services, enabling them to evolve independently while maintaining a well-defined flow.
3. A common use case is applying a series of filters or validations to a request before processing it. This pattern enhances reusability, maintainability, and modularity while allowing for easy addition or removal of handlers.
4. This pattern encourages loose coupling between sender and receiver, providing freedom in handling the request.
5. Each handler in the chain has a single responsibility, either handling the request or passing it to the next handler, which helps in maintaining a clean and modular design.
6. The chain can include a mechanism to handle requests that are not handled by any handler in the chain, providing a fallback or default behavior.

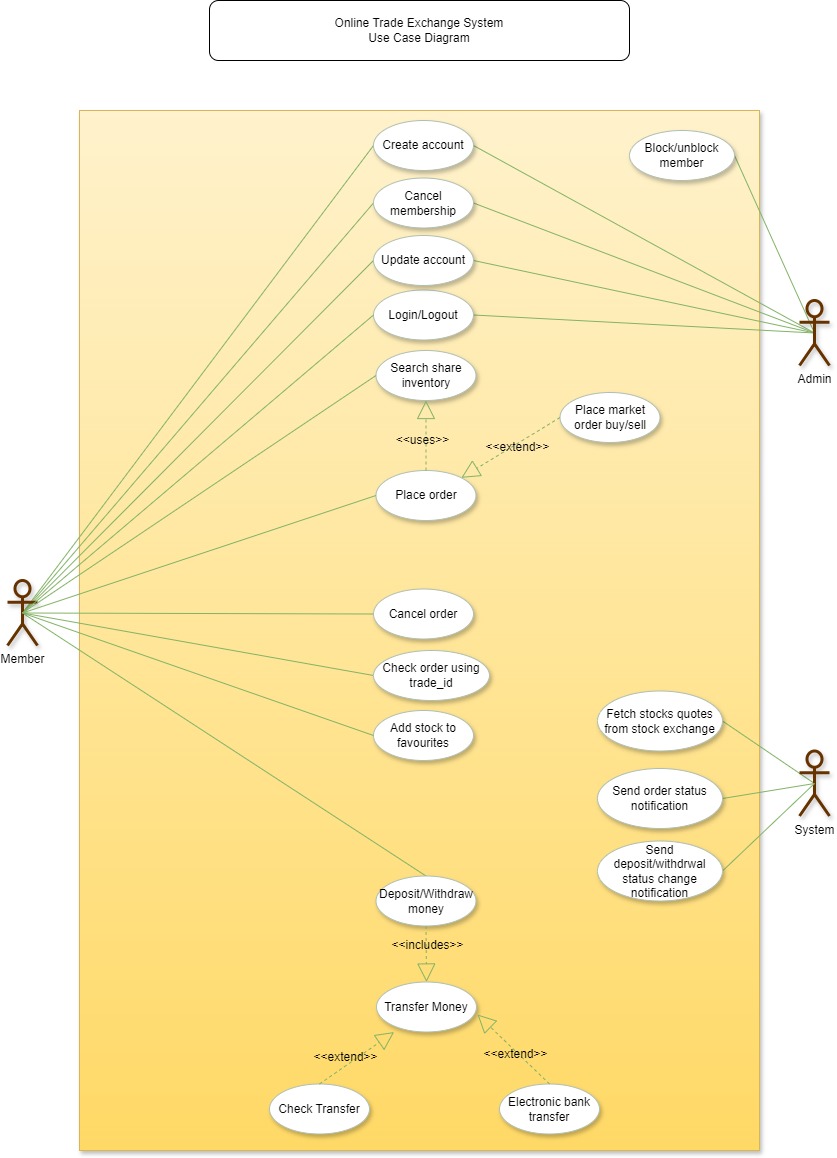
****

### Use Case Diagram

* **Admin:** Mainly responsible for administrative functions like blocking or unblocking members.
* **Member:** All members can search the stock inventory, as well as buy and sell stocks. Members can have multiple watchlists containing multiple stock quotes.
* **System:** Mainly responsible for sending notifications for stock orders and periodically fetching stock quotes from the stock exchange.

Here are the top use cases of the Stock Brokerage System:

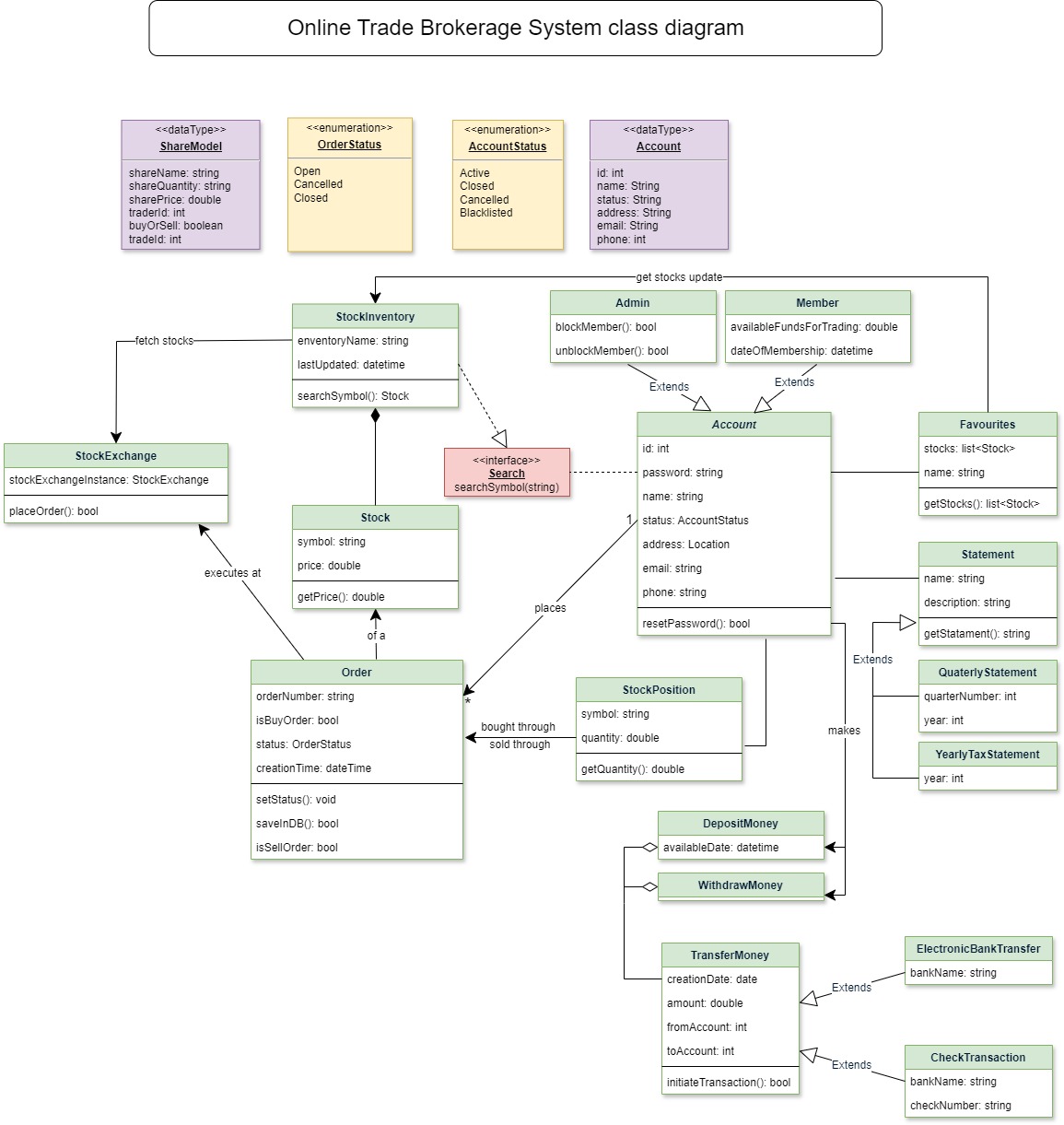
* **Register new account/Cancel membership:** To add a new member or cancel the membership of an existing member.
* **Add/Remove/Edit watchlist:** To add, remove or modify a watchlist.
* **Search stock inventory:** To search for stocks by their symbols.
* **Place order:** To place a buy or sell order on the stock exchange.
* **Cancel order:** Cancel an already placed order.
* **Deposit/Withdraw money:** Members can deposit or withdraw money via check, wire or electronic bank transfer.



### Class Diagram

Here are the main classes of our Online Stock Brokerage System:

* Share Model: Consist of details of Share Name, Share Quantity, Share Price, Trader Id, Buy Or Sell, Trade Id.
* **Account:** Consists of the member’s name, address, e-mail, phone, total funds, funds that are available for trading, etc. We’ll have two types of accounts in the system: one will be a general member, and the other will be an Admin.
* **Stock Exchange:** The stockbroker system will fetch all stocks and their current prices from the stock exchange.
* **Stock:** The basic building block of the system. Every stock will have a symbol, current trading price, etc.
* **Stock Inventory:** This class will fetch and maintain the latest stock prices from the Stock Exchange.
* **Favourites:** A watchlist will contain a list of stocks that the member is interested
* **Order:** Members can place stock trading orders whenever they would like to sell or buy share positions. Consist of order number, buy or sell order, status, time etc
* **Stock Position:** This class will contain all the stocks that the user holds.
* **Statement:** All members will have reports for quarterly updates and yearly tax statements.
* **Deposit Money & Withdraw Money:** Members will be able to move money through check, or electronic bank transfers.
* **Notification:** Will take care of sending notifications to members.



**Also as mentioned in requirements design should consider below points:**

* 1. High availability - High availability refers to the characteristic of a system or service being continuously operational and accessible for a high percentage of time, typically measured as a percentage of uptime. It involves implementing strategies to minimize downtime and ensure that the system remains accessible and functional even in the face of failures, errors, or maintenance activities.

To achieve this, we need to have system like docker and Kubernetes so it will have less down time, load will get balanced by using pods, maintenance of system will be easier, we can scale as per requirement. Containerization platforms like Docker, coupled with orchestration tools like Kubernetes, facilitate the deployment and management of applications in a highly available manner. Containers provide lightweight, isolated environments, while orchestration automates tasks such as scaling, load balancing, and self-healing.

* 1. Volume of trade requests such as 100K in an hour.

As mentioned above using Containerization platforms like Docker, coupled with orchestration tools like Kubernetes will ease to balance load and cater each request without failing.

* 1. Deployment strategy

To use load balancing, you need to scale your deployments. When you scale a deployment, you replicate its pods, creating more running instances of your applications. Scaling is one of the primary advantages of Kubernetes because you can replicate your application to accommodate more traffic, and then descale your deployments to free up resources when the traffic decreases.

* 1. Database

Its very difficult to judge any database because all database providers have their pros and cons.

I personally believe PostgreSQL will be good selection here.

PostgreSQL is a database management system that operates on the Linux operating system and interacts with objects in a relational fashion. It is named as such because it uses SQL to retrieve data stored in the database’s tables.

Alternatively, we have option like Oracle, Cassandra, PostgreSQL, MongoDB

* 1. Resiliency

Resiliency is the ability of application to recover from certain types of failures and remain functional.

Though The circuit breaker pattern has best resiliency but with other factors won’t be easy to achieve with this pattern.

But using Chain of Responsibility Design Pattern will be helpful to achieve high resiliency as its chain of actions so till the time previous action doesn't get satisfied, we don’t move ahead.

* 1. Monitoring system

Monitoring help to provide the insights and visibility needed to maintain and improve Java application and server performance.

Using different log input-based tools like Elastic search, Splunk, VMware log insight etc monitoring can be achieved.