# Stock Broking Systems

**PROJECT REPORT**

**(2021 Regulation)**

***Submitted by***

**Pranav Prabhat Sinha (RA2211047010105)**

**Ananyashree (RA2211047010117)**

**II Year/ IIIrd Semester**

**Academic Year: 2023 -2024**

***Under the Guidance of***

**Dr. SUMATHY G**

**Assistant Professor, Department of Computational Intelligence**

***In partial satisfaction of the requirements for the degree of***

**BACHELOR OF TECHNOLOGY**

**In**

**ARTIFICIAL INTELLIGENCE**



**SCHOOL OF COMPUTING**

**COLLEGE OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR - 603203**

# NOV / DEC 2023

A close-up of a document

Description automatically generated

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **CONTENTS** | **PAGE NO** |
|  | Abstract | **1** |
| 1 | Problem Statement | **2** |
| 2 | Objective | **3** |
| 3 | Modules of Project | **4** |
| 4 | UML Diagrams | **5-13** |
|  | 4.1. Use case Diagram | **5** |
|  | 4.2. Class Diagram | **6** |
|  | 4.3. Sequence Diagram | **7** |
|  | 4.4. Collaboration Diagram | **8** |
|  | 4.5. State Chart Diagram | **9** |
|  | 4.6. Activity Diagram | **10** |
|  | 4.7. Package Diagram | **11** |
|  | 4.8. Component Diagram | **12** |
|  | 4.9. Deployment Diagram | **13** |
| 5 | Code/Output Screenshots | **14-24** |
| 6 | Conclusion | **25** |
| 7 | References | **26** |

**Abstract**

# The stock broking industry plays a pivotal role in financial markets by facilitating the buying and selling of securities. In an era characterized by rapid technological advancements and increasing investor demands, there is a growing need for an efficient Stock Broking System that can simplify and enhance the investment process. This abstract provides an overview of such a system designed to address these challenges.

# The proposed Stock Broking System leverages cutting-edge technology to streamline the entire stock trading process, from account creation and order placement to portfolio management and reporting. Key features of the system include:

# ****User-Friendly Interface****

# ****Real-time Market Data****

# ****Robust Security Measures****

# ****Customized Investment Strategies****

# ****Trading Platform Integration****

# ****Reporting and Analytics****

# In summary, the Stock Broking System described here represents a comprehensive solution for investors, offering efficiency, security, and accessibility in today's dynamic financial markets. By providing an all-in-one platform that caters to both novice and experienced investors, this system aims to enhance the investment experience and simplify the complexities of stock trading

# 

**1.Problem Statement**

Design and develop stockbroking systems that allow investors to trade securities, manage their portfolios, retrieve business information, and interact with financial markets. The system should meet the needs of both new and experienced traders by providing an intuitive user interface, powerful security measures, real-time market information and reliable business.

**2. Objective:**

The objective of designing and developing a Stock Brokerage System as outlined in the problem statement is to create a comprehensive platform that facilitates efficient and secure trading of securities while providing users with tools to manage their investments and make informed decisions. The system aims to achieve several key objectives:

1.Efficient Trading Platform

2.Portfolio Management

3.Security and Compliance

4.Reliable Transaction Tracking

5.Scalability and Performance

6.Automation and Efficiency

7.Customer Support

8.Long-Term Financial Goals

# 3. Modules of Project:

Functional requirements define the specific features, capabilities, and behaviours that a system must have in order to meet the needs of its users and achieve its objectives. In the context of a Stock Brokerage System, here are some functional requirements:

1.User Registration and Authentication

2.Account Management

3.Portfolio Creation and Management

4.Trading Functionality

5.Real-Time Market Data

6.Order Management

7.Portfolio Performance Tracking

8.Asset Allocation Tools

9.Transaction History

10.Alerts and Notifications

11.Educational Resources

12.Customer Support

13.Compliance and Reporting

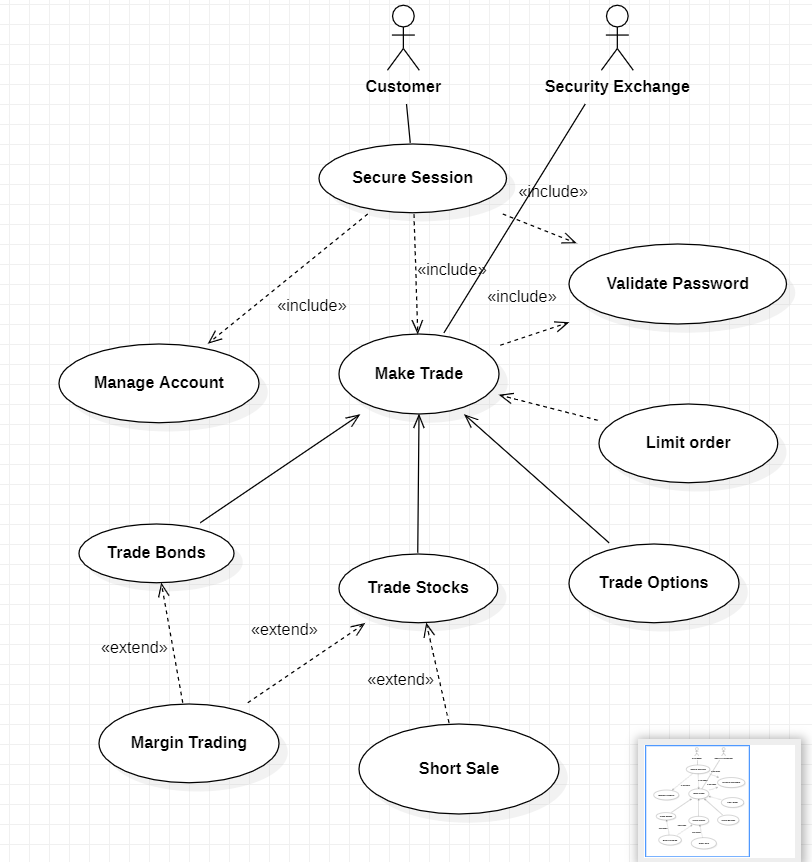
14.Scalability and Performance

15.Admin Control

These functional requirements provide a foundation for the features and capabilities that the Stock Brokerage System needs to fulfil in order to meet user needs and achieve its objectives. The specific requirements may vary based on the scope and goals of the system.

**4. UML DIAGRAMS:**

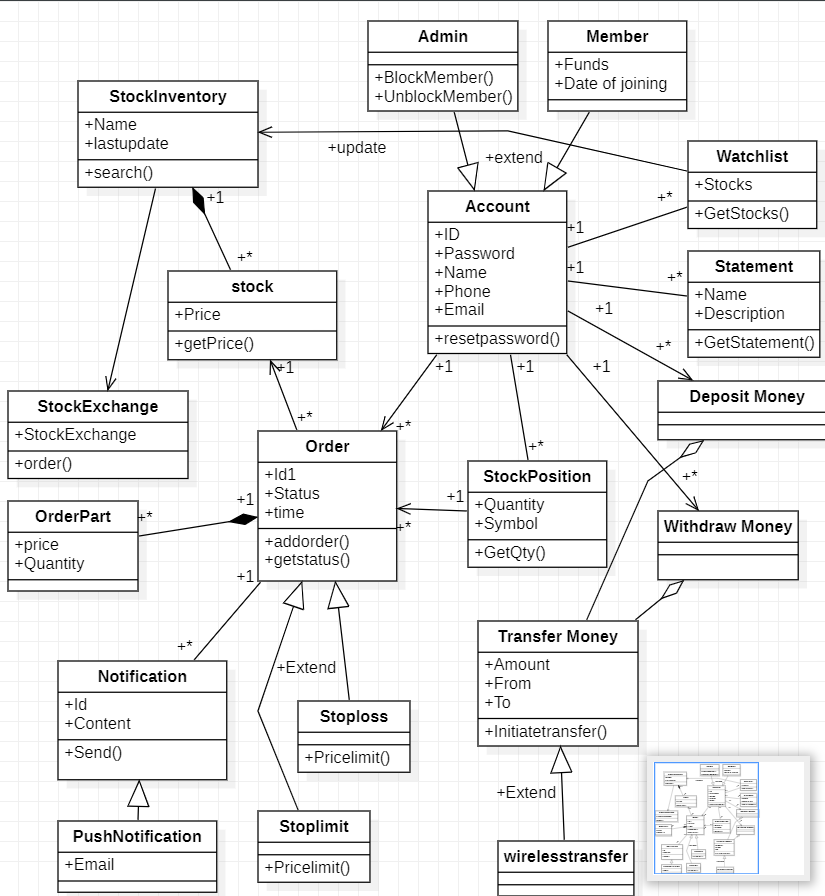
**4.1) Use Case Diagram:**

****

**Fig. 5.1**

A Use Case Diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts thehigh-level functionality of a system and also tells how the user handles a system.

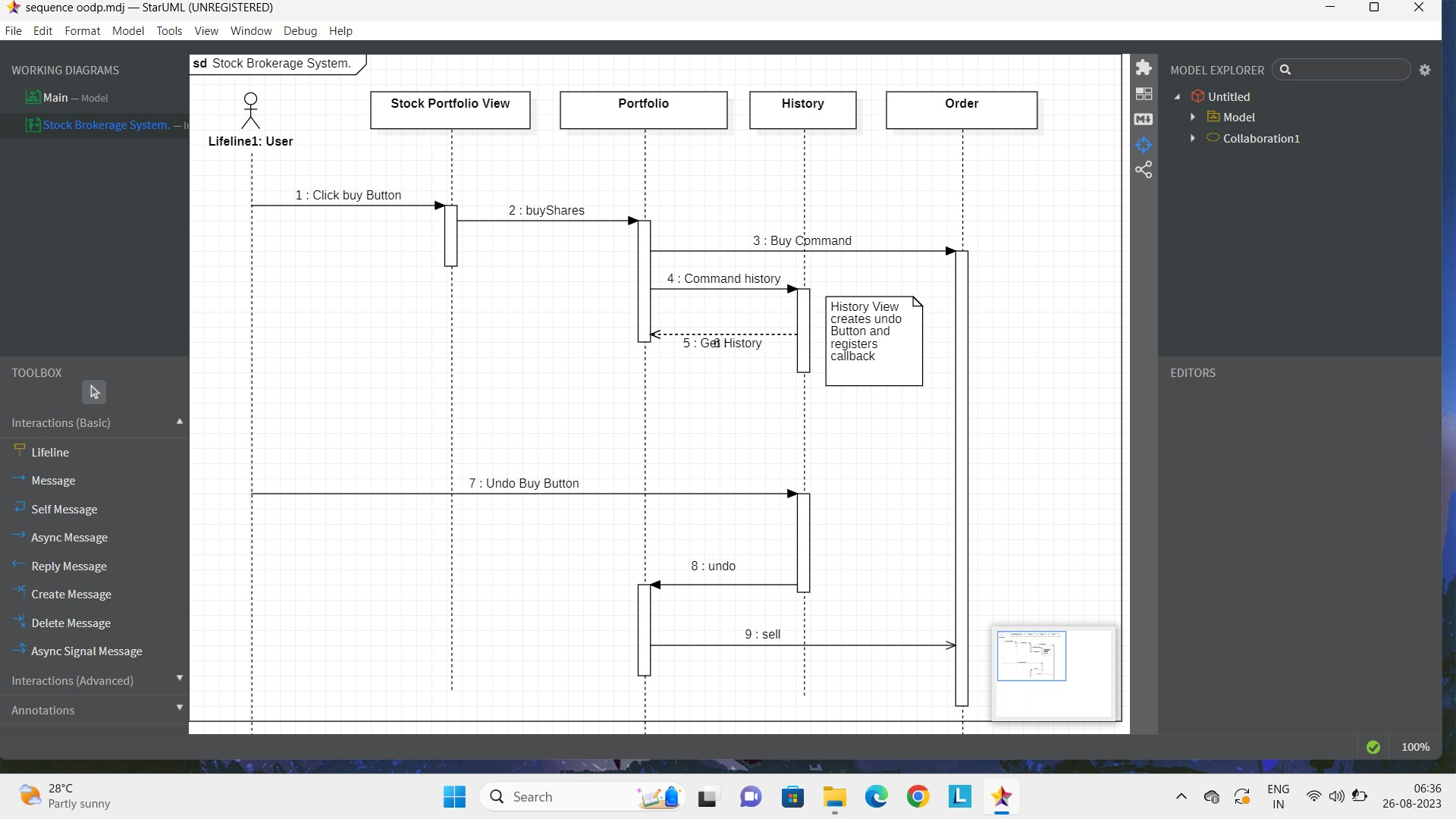
# 4.2) Class Diagram:



**Fig. 5.2**

Class diagram describes the attributes and operations of a class and also theconstraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

# 4.3) Sequence Diagram:

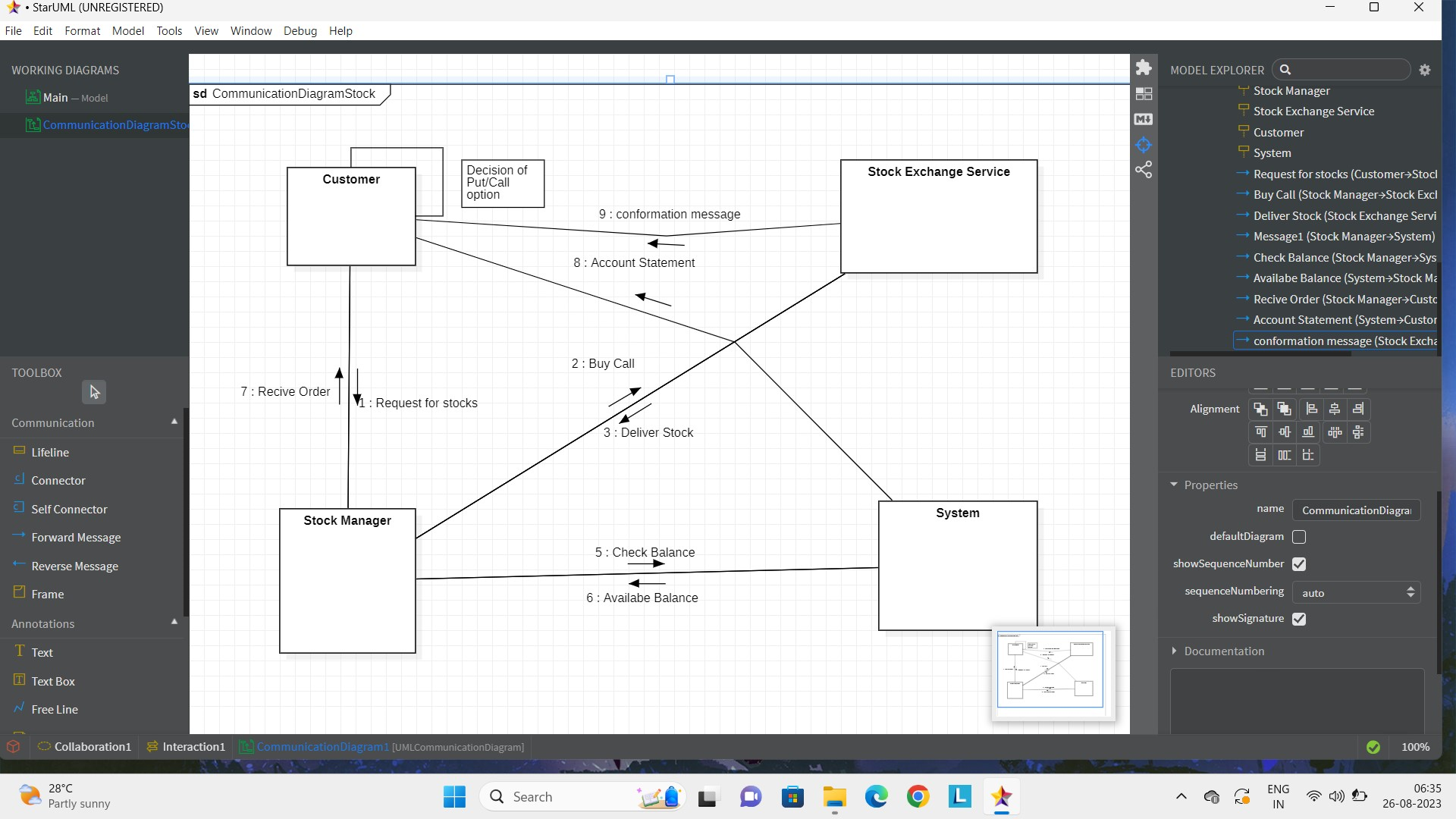


**Fig. 5.3**

A Sequence Diagram or system sequence diagram (SSD) shows process interactions arranged in time sequence in the field of software engineering. It depicts the processes involved and the sequence of messages exchanged between the processes needed to carry out the functionality.

Sequence diagramsare typically associated with use case realizations in the 4+1 architectural view model of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

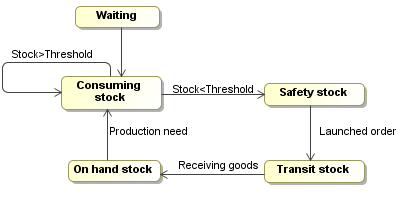
# 4.4) Collaboration Diagram:



**Fig. 5.4**

A collaboration diagram, also known as a communication diagram, is an illustration of the relationships and interactions among software [objects](https://www.techtarget.com/searchapparchitecture/definition/object) in the Unified Modeling Language ([UML](https://www.techtarget.com/searchsoftwarequality/definition/Unified-Modeling-Language)). These diagrams can be used to portray thedynamic behavior of a particular [use case](https://www.techtarget.com/searchsoftwarequality/definition/use-case) and define the role of each object.

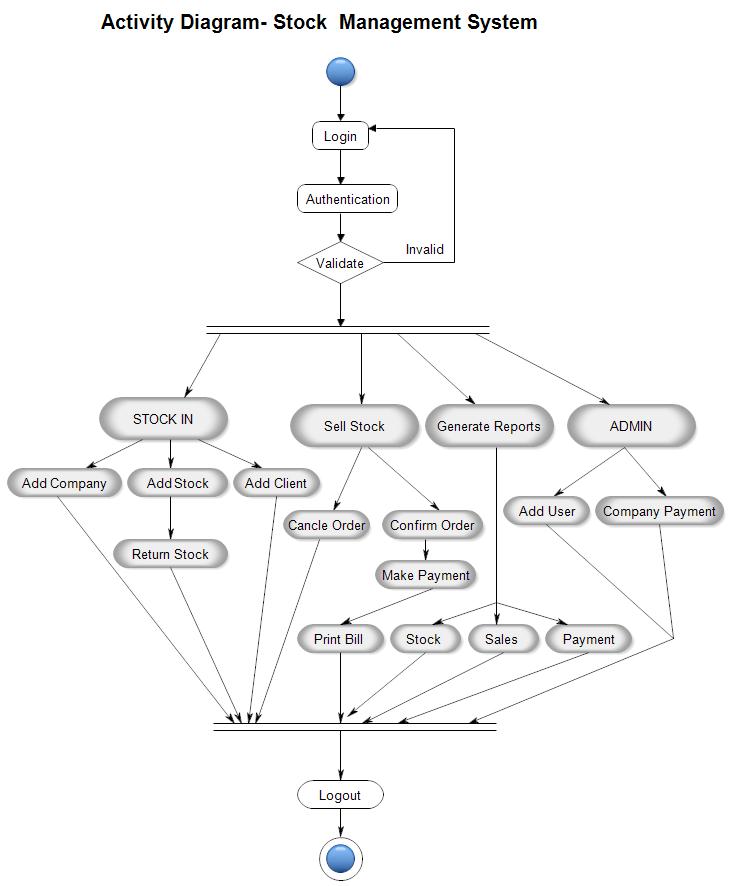
# 4.5) State Chart Diagram:



**Fig. 5.5**

State Chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changeswhen some event is triggered. The most important purpose of State chart diagram is to model lifetime of an object from creation to termination. State chart diagrams are also used for forward and reverse engineering of a system.

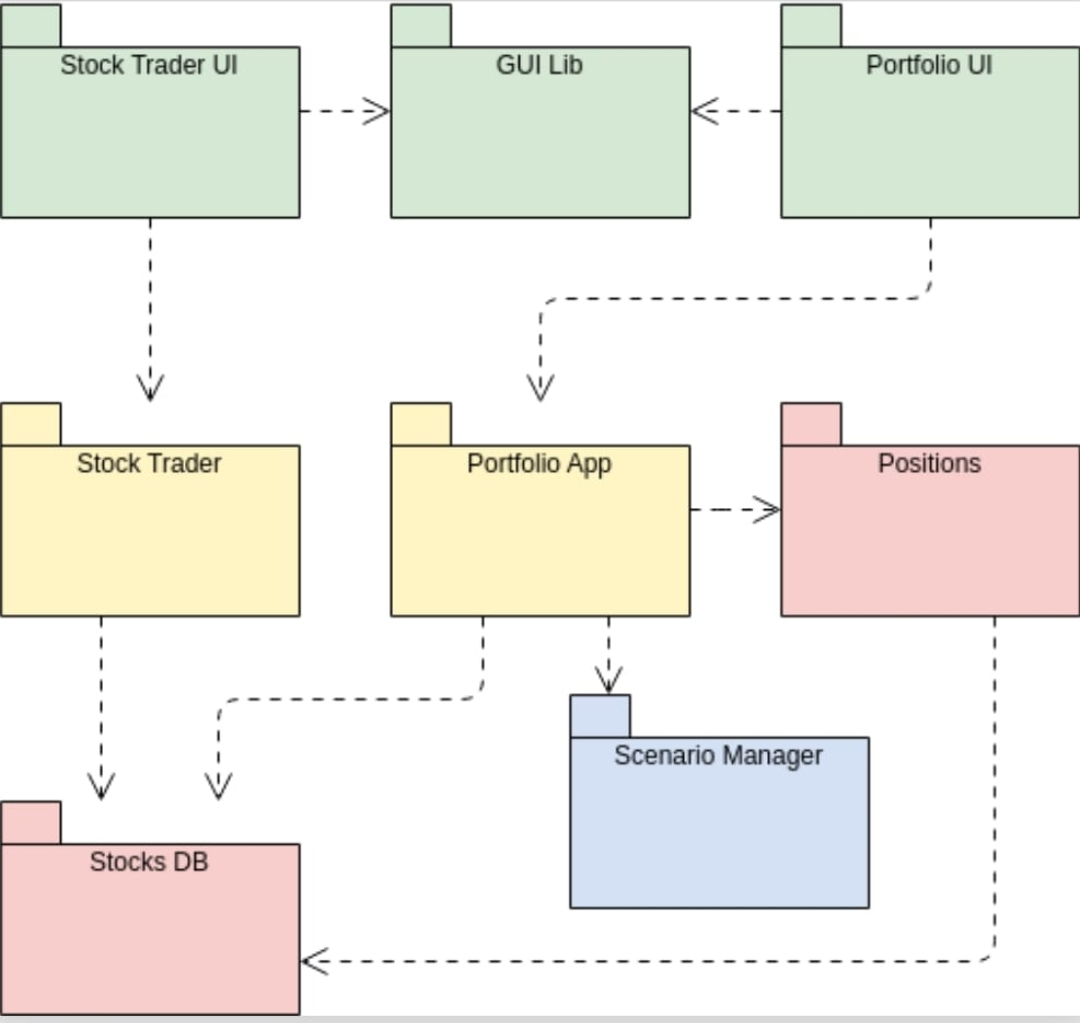
# Activity Diagram:



**Fig. 5.6**

Activity diagrams are graphical representations of workflows of stepwise activities and action with support for choice, iteration, and concurrency. In theUnified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagramsprimarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores. The filled circle in the above diagram denotes the starting position of the diagram. Then comes the idle condition.

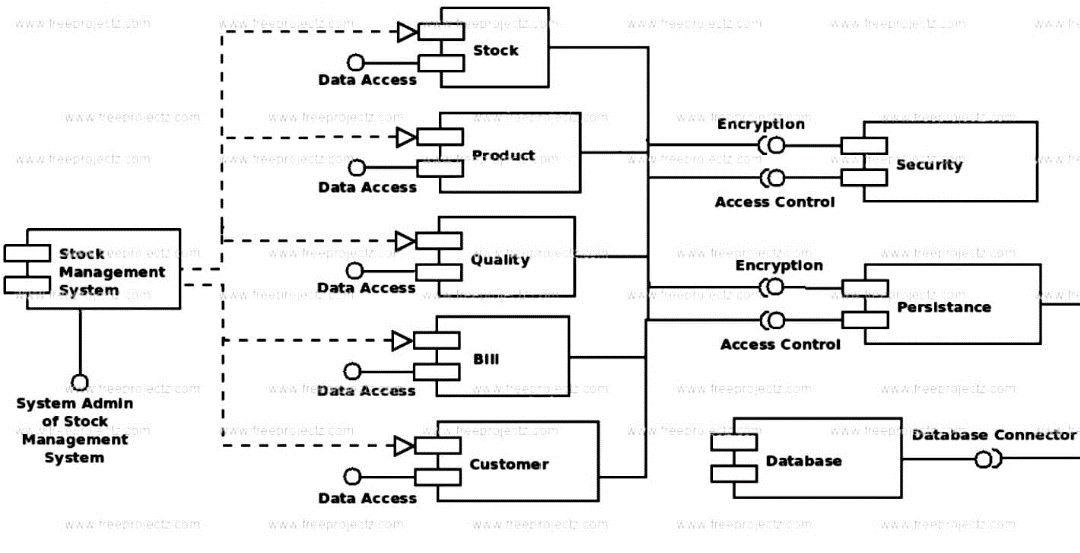
# Package Diagram:



**Fig. 5.7**

Package diagrams are structural diagrams used to show the organization and arrangement of various model elements in the form of packages. A package isa grouping of related [UML elements,](https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language) such as diagrams, documents, classes, oreven other packages. Each element is nested within the package, which is depicted as a file folder within the diagram, then arranged hierarchically within the diagram.

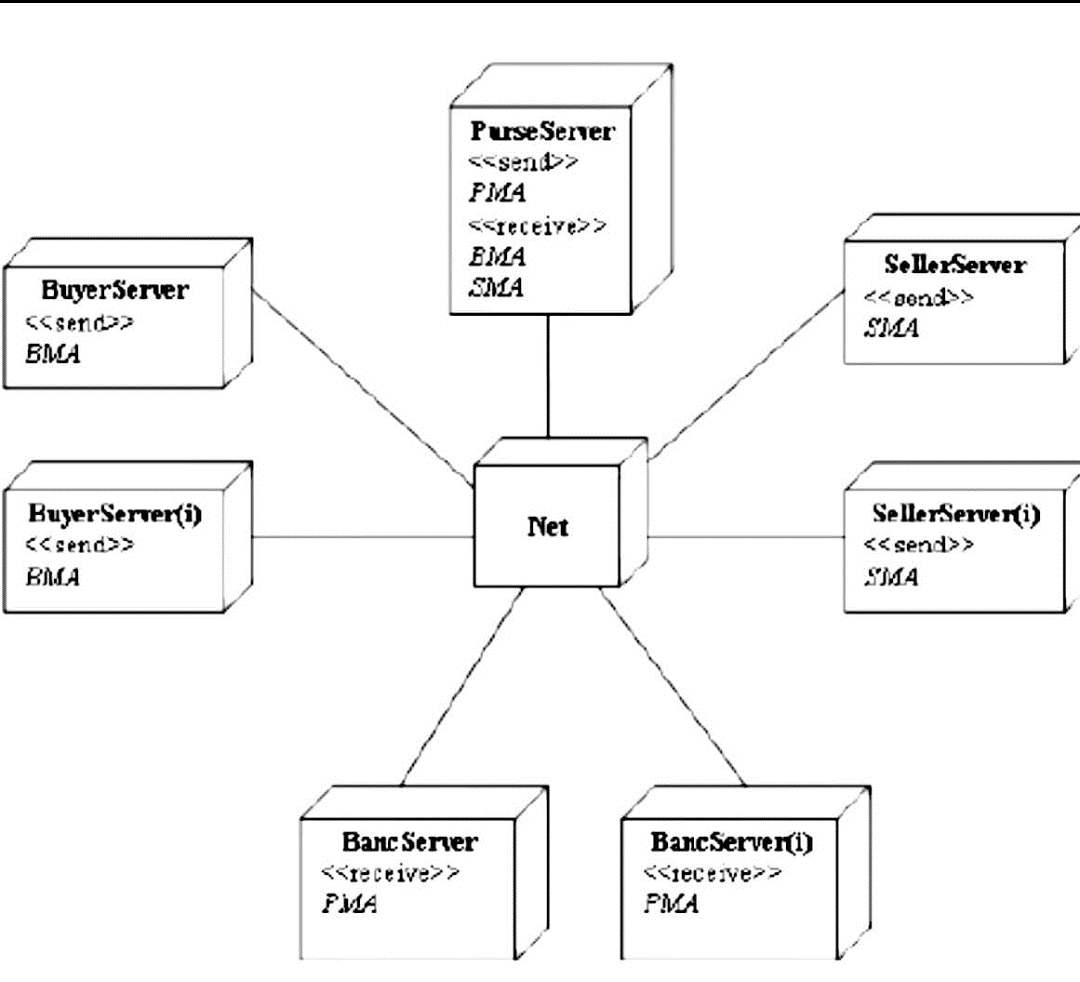
# Component Diagram:



**Fig. 5.8**

Component diagrams are used in modeling the physical aspects of object- oriented systems that are used for visualizing, specifying, and documenting component- based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.

# Deployment Diagram:



**Fig. 5.9**

A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them.

Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware. Deployment diagrams help model the hardware topology of a system compared to other UML diagram types which mostly outline the logical components of a system.

**5. CODE / OUTPUT SCREENSHOTS**

#include <bits/stdc++.h>

using namespace std;

class Account {

int dogecoin, balance;

int bitcoin;

int deposit, withdraw;

int total\_equity = 100;

int predict;

int dogecoin\_value;

int bitcoin\_value;

int crypto\_invest;

int crypto\_return;

// Vector declared

vector<pair<string, int> > transactions;

public:

// Deposit Cash

bool Deposit(int money)

{

deposit += money;

balance += money;

transactions.push\_back({ "Deposit:", money });

}

// Get account Details

void Get\_account\_information()

{

cout << "Money Details:\n";

cout << "Bank Balance:" << balance << endl;

cout << "Dogecoin:" << dogecoin << endl;

cout << "Bitcoin:" << bitcoin << endl;

}

// Withdraw Cash

bool Withdraw(int money)

{

if (money > balance) {

return false;

}

withdraw += money;

balance -= money;

transactions.push\_back({ "Withdraw:", money });

return true;

}

// Buy Crypto function

bool buy\_crypto()

{

int option;

cout << "Want to purchase dogecoin press 1 else "

"for bitcoin press 2\n";

cin >> option;

// Checking equity

if (total\_equity != 0) {

srand(time(NULL));

int luck = 251;

if (luck % 251 == 0) {

if (option == 1) {

dogecoin += 1;

balance -= dogecoin\_value;

crypto\_invest

+= (dogecoin)\*dogecoin\_value;

}

else {

bitcoin += 1;

balance -= bitcoin\_value;

crypto\_invest += bitcoin\_value;

}

}

else {

return false;

}

}

else {

return false;

}

return true;

}

// Selling crypto function

bool sell\_crypto()

{

int option;

cout << "Want to sell dogecoin press 1 else for "

"bitcoin press 2\n";

cin >> option;

if (option == 2) {

if (bitcoin == 0)

return false;

crypto\_return += bitcoin\_value;

balance += bitcoin\_value;

transactions.push\_back(

{ "Bitcoin Sold:", bitcoin\_value });

bitcoin -= 1;

}

else {

if (dogecoin == 0)

return false;

crypto\_return += dogecoin\_value;

balance += dogecoin\_value;

transactions.push\_back(

{ "Dogecoin Sold:", dogecoin\_value });

dogecoin -= 1;

}

return true;

}

// Checking All Transactions

void History()

{

cout << "Displaying All transactions\n";

for (auto it : transactions) {

cout << it.first << " " << it.second << endl;

}

char temp;

cout << "Do you want to clear all transactions:";

cin >> temp;

int no = transactions.size();

// Clearing All transactions

if (temp == 'Y') {

transactions.clear();

cout << "Total transactions cleared:" << no

<< endl;

}

else {

cout << "Total transaction:" << no << endl;

}

}

Account()

{

crypto\_invest = 0;

crypto\_return = 0;

total\_equity = 100;

balance = 50000;

dogecoin = 0;

bitcoin = 0;

withdraw = 0;

deposit = 0;

dogecoin\_value = 100;

bitcoin\_value = 500;

}

};

int main()

{

Account person;

int amount, choice;

bool check;

while (1) {

cout << " "

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n";

cout << endl;

cout << "Press 1 if want to have your Account Info "

<< endl;

cout << "Press 2 if want to Deposit your money "

<< endl;

cout << "Press 3 if want to withdraw your money "

<< endl;

cout << "Press 4 if want to know your history "

<< endl;

cout << "Press 5 if want to know your Buy Crypto "

<< endl;

cout << "Press 6 if want to know your Sell Crypto "

<< endl;

cout << "Else press any invalid key for exit \n"

<< endl;

cout << " "

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \n";

cin >> choice;

int ans;

switch (choice) {

case 1:

person.Get\_account\_information();

break;

case 2:

cout << "Enter amount to deposit : ";

cin >> amount;

ans = person.Deposit(amount);

if (ans)

cout << "Successfully deposited money"

<< endl;

else

cout << "Failed\n";

break;

case 3:

cout << "Enter amount to withdrawn : ";

cin >> amount;

person.Withdraw(amount);

if (ans)

cout << "Successfully withdrawn Amount"

<< endl;

else

cout << "Not Enough Balance\n";

break;

case 4:

person.History();

break;

case 5:

ans = person.buy\_crypto();

if (ans)

cout << "Successful Transaction" << endl;

else

cout << "Better Luck next time\n";

break;

case 6:

ans = person.sell\_crypto();

if (ans)

cout << "Successful Transaction" << endl;

else

cout << "Not Enough Cryptocoins\n";

break;

default:

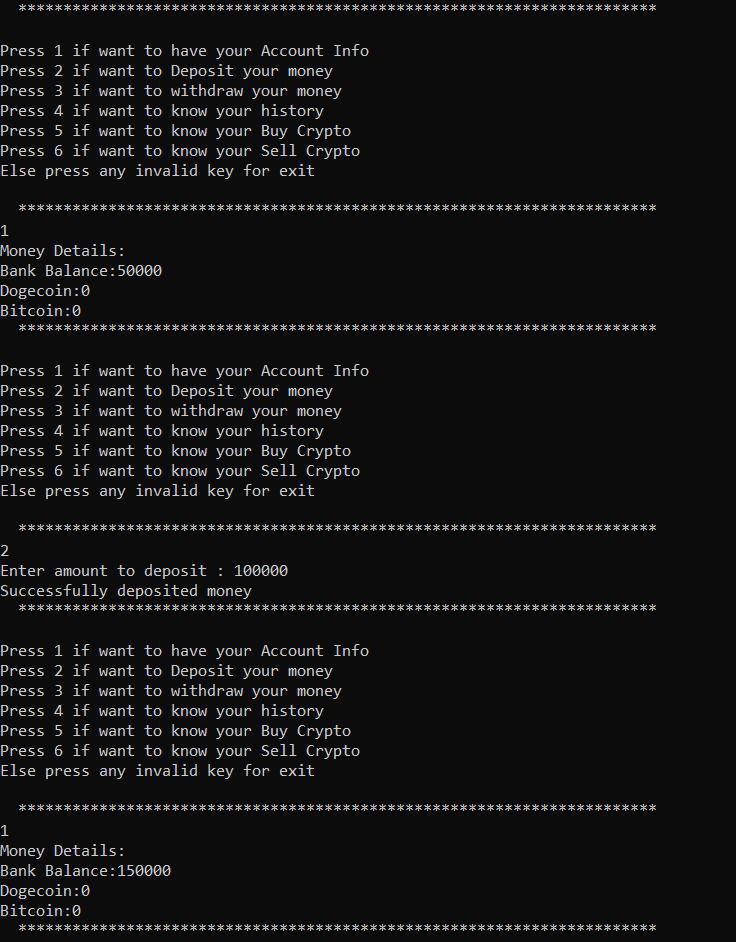
exit(0);

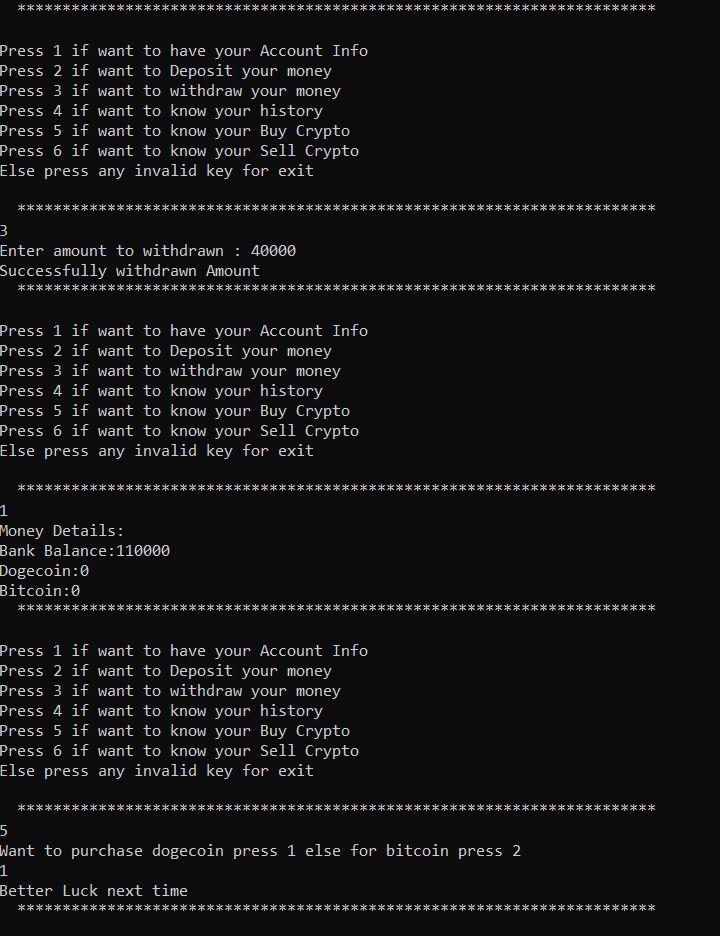
break;

}

}

}





# 

# Lightbox

# 6. CONCLUSION

This project is developed successfully and the performance is found to be satisfactory. This project is designed to meet the requirements of a Stock Brokerage System. It has been developedin C++ keeping in mind the specifications of the system. The user will be able to Buy and Sell Crypto as well as keep the track of expense with their account.

We have designed the project to provide the user with easy deposit and withdrawal of money with great trading experience. In this project, the user is provided with a program that can be used for Crypto Trading.

Working on this project was really a learning experience and we have come a long way in building our concepts. The Stock Broking System developed by us is purely based on C++, OOPS with help of UML Diagrams. During the course of this assignment we have gone through many obstacles which made us to research and though increased our knowledge. Now we are clear with all these concepts and fundamentals which will be helping us in the future.

# 7. REFERENCES

[1] kaur, l., kaur, n., ummat, a., kaur, j., & kaur, n. (2016). research paper on object oriented software engineering. international journal of computer science and technology, 36-38.

[2] Kak, Avinash C. Programming with Objects, A Comparative Presentation of Object-Oriented Programming with C++ and Java, John Wiley, 2003. ISBN 0-471-26852-6.

[3] Lafore, Robert, Object-Oriented Programming in C++, Fourth Edition, Sams Publishing, 2002. ISBN 0-672- 32308-7.

* [**https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/**](https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/)
* **https://**[**www.stackoverflow.com/**](http://www.stackoverflow.com/)
* **https://**[**www.google.com/**](http://www.google.com/)
* [**https://www.staruml.io/**](https://www.staruml.io/)
* [**https://www.moneycontrol.com/stocksmarketsindia/**](https://www.moneycontrol.com/stocksmarketsindia/)
* [**https://crypto.com/**](https://crypto.com/)