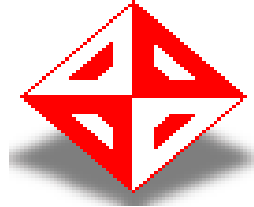




MIDDLE EAST TECHNICAL UNIVERSITY



COMPUTER ENGINEERING

ALGORITHMIC

TRADING

SOFTWARE DESIGN DESCRIPTION

Prepared by *Money Fellas*:

ALİ ŞAVKAR 1819879

GÖKÇER YAPAR 1819648

MURAT TARIMER 1819556

ÖMER YAVUZ 1819655

04.01.2015

TABLE OF CONTENTS

1. OVERVIEW	- 1 -
1.1. SCOPE.....	- 1 -
1.2. PURPOSE.....	- 1 -
1.3. INTENDED AUDIENCE.....	- 1 -
1.4. REFERENCES.....	- 1 -
2. DEFINITIONS	- 3 -
3. CONCEPTUAL MODEL FOR SOFTWARE DESIGN DESCRIPTIONS	- 4 -
3.1. SOFTWARE DESIGN IN CONTEXT.....	- 4 -
3.2. SOFTWARE DESIGN DESCRIPTIONS WITHIN THE LIFE CYCLE.....	- 5 -
3.2.1. INFLUENCES ON SDD PREPARATION.....	- 5 -
3.2.2. <i>INFLUENCES ON SOFTWARE LIFE CYCLE PRODUCTS</i>	- 5 -
3.2.3. <i>DESIGN VERIFICATION AND DESIGN ROLE IN VALIDATION</i>	- 5 -
4. DESIGN DESCRIPTION INFORMATION CONTENT	- 6 -
4.1. INTRODUCTION.....	- 6 -
4.2. SDD IDENTIFICATION.....	- 6 -
4.3. DESIGN STAKEHOLDERS AND THEIR CONCERNS.....	- 6 -
4.4. DESIGN VIEWS.....	- 7 -
4.5. DESIGN VIEWPOINTS.....	- 7 -
4.6. DESIGN ELEMENTS.....	- 8 -
4.6.1. <i>DESIGN ENTITIES</i>	- 8 -
4.6.1.1. SERVER SYSTEM.....	- 9 -
4.6.1.2. CLIENT SYSTEM.....	- 9 -
4.6.1.3. ALGORITHMS SERVER SYSTEM.....	- 9 -
4.6.1.4. COMMUNICATION COMPONENT.....	- 9 -
4.6.1.5. FIX8 FRAMEWORK.....	- 10 -
4.6.1.6. MONGODB DATABASE.....	- 10 -
4.6.1.7. MYSQL DATABASE.....	- 10 -
4.6.1.8. JAVA FX SOFTWARE PLATFORM.....	- 10 -
4.6.1.9. C++ PROGRAMMING LANGUAGE.....	- 10 -
4.6.1.10. APACHE SUBVERSION.....	- 11 -
4.6.2. <i>DESIGN ATTRIBUTES</i>	- 11 -
4.6.3. <i>DESIGN RELATIONSHIPS</i>	- 11 -
4.6.4. <i>DESIGN CONSTRAINTS</i>	- 12 -
4.7. DESIGN OVERLAYS.....	- 12 -
4.8. DESIGN RATIONALE.....	- 12 -

4.9. DESIGN LANGUAGES	- 12 -
5. DESIGN VIEWPOINT.....	- 13 -
5.1. INTRODUCTION.....	- 13 -
5.2. CONTEXT VIEWPOINT.....	- 13 -
5.2.1. DESIGN CONCERNS.....	- 13 -
5.2.2. DESIGN ELEMENTS	- 14 -
5.2.2.1. LOGIN SCREEN USE CASE DIAGRAM	- 14 -
5.2.2.1.1. ENTER USERNAME USE CASE DIAGRAM	- 15 -
5.2.2.1.2. ENTER PASSWORD USE CASE DIAGRAM	- 15 -
5.2.2.1.3. FORGOT YOUR PASSWORD USE CASE DIAGRAM	- 15 -
5.2.2.1.4. LOGIN USE CASE DIAGRAM	- 16 -
5.2.2.2. MAIN SCREEN USE CASE DIAGRAM	- 16 -
5.2.2.2.1. SELECT ACTIVE TRADE USE CASE DIAGRAM	- 17 -
5.2.2.2.2. STOP/RESUME TRADE USE CASE DIAGRAM	- 17 -
5.2.2.2.3. SIMULATE USE CASE DIAGRAM.....	- 17 -
5.2.2.2.4. SHOW TRADE LOG USE CASE DIAGRAM.....	- 18 -
5.2.2.2.5. SHOW TRADE DETAILS USE CASE DIAGRAM	- 18 -
5.2.2.2.6. SHOW PORTFOLIO USE CASE DIAGRAM.....	- 18 -
5.2.2.2.7. SHOW STOCK EXCHANGE MARKET DATE USE CASE DIAGRAM.....	- 19 -
5.2.2.2.8. SELL/BUY TRADE USE CASE DIAGRAM	- 19 -
5.2.2.2.9. OPTION MENU USE CASE DIAGRAM	- 19 -
5.2.2.2.10. CREATE NEW TRADE USE CASE DIAGRAM	- 20 -
5.2.2.3. NEW TRADE SCREEN USE CASE DIAGRAM	- 20 -
5.2.2.3.1. SELECT TRADE KIND USE CASE DIAGRAM	- 21 -
5.2.2.3.2. ENTER EXPIRATION DATE USE CASE DIAGRAM	- 21 -
5.2.2.3.3. ENTER SIZE USE CASE DIAGRAM	- 21 -
5.2.2.3.4. SELECT STRATEGY USE CASE DIAGRAM.....	- 22 -
5.2.2.3.5. START TRADE USE CASE DIAGRAM.....	- 22 -
5.2.2.3.6. CANCEL TRADE USE CASE DIAGRAM	- 22 -
5.2.2.4. LOG SCREEN USE CASE DIAGRAM	- 23 -
5.2.3. EXAMPLE UML LANGUAGE.....	- 24 -
5.2.4. DESIGN CONSTRAINTS	- 24 -
5.2.5. DESIGN RELATIONSHIPS	- 25 -
5.3. COMPOSITION VIEWPOINT	- 26 -
5.3.1. DESIGN CONCERNS.....	- 26 -
5.3.2. DESIGN ELEMENTS	- 26 -
5.3.3. EXAMPLE UML LANGUAGE.....	- 29 -
5.4. LOGICAL VIEWPOINT	- 30 -
5.4.1. DESIGN CONCERNS.....	- 30 -
5.4.2. DESIGN ELEMENTS	- 30 -

5.4.2.1. Users NAMESPACE	- 30 -
5.4.2.1.1. Account	- 30 -
5.4.2.2. Trades NAMESPACE	- 31 -
5.4.2.2.1. Trade	- 31 -
5.4.2.2.2. TradeOrder	- 32 -
5.4.2.2.3. MarketData	- 32 -
5.4.2.3. GUI PACKAGE	- 33 -
5.4.2.3.1. DisplayRunner	- 33 -
5.4.2.3.2. MainController	- 34 -
5.4.2.3.3. LoginController.....	- 35 -
5.4.2.3.4. OpenPosition.....	- 36 -
5.4.2.4. Database NAMESPACE	- 37 -
5.4.2.4.1. DBController.....	- 37 -
5.4.2.4.2. FIXDATARepositoryControllerDB	- 37 -
5.4.2.4.3. AccountControllerDB	- 38 -
5.4.2.4.4. TradeLogControllerDB.....	- 38 -
5.4.2.5. Strategy NAMESPACE.....	- 39 -
5.4.2.5.1. AlgoController	- 39 -
5.4.2.6. Communication NAMESPACE	- 39 -
5.4.2.6.1. SocketCommunication	- 39 -
5.4.2.6.2. FIX	- 40 -
5.4.2.7. CppComponent NAMESPACE.....	- 40 -
5.4.2.7.1. CppServer.....	- 40 -
5.4.2.7.2. CppClient.....	- 41 -
5.4.2.8. JavaComponent PACKAGE	- 42 -
5.4.2.8.1. JavaClient	- 42 -
5.4.2.8.2. JavaServer	- 43 -
5.4.3. EXAMPLE UML LANGUAGE.....	- 44 -
5.5. INTERFACE VIEWPOINT	- 45 -
5.5.1 DESIGN CONCERNS.....	- 45 -
5.5.2. DESIGN ELEMENTS	- 45 -
5.5.3. EXAMPLE UML LANGUAGE.....	- 47 -
5.6. INTERACTION VIEWPOINT.....	- 48 -
5.6.1. DESIGN CONCERNS.....	- 48 -
5.6.2. DESIGN ELEMENTS	- 49 -
5.6.2.1. LOGIN/LOGOUT SEQUENCE DIAGRAM	- 49 -
5.6.2.2. SCREEN REPLACING SEQUENCE DIAGRAM	- 50 -
5.6.2.3. RECEIVING PORTFOLIO SEQUENCE DIAGRAM	- 51 -
5.6.2.4. OPENING NEW TRADE DIALOG SEQUENCE DIAGRAM	- 52 -
5.6.2.5. C++ TRADE CREATION AND FIX COMMUNICATION SEQUENCE DIAGRAM	- 53 -
5.6.2.6. ALGORITHM CHOICE SEQUENCE DIAGRAM	- 55 -

5.6.2.7. DISPLAY DETAILED INFORMATION ABOUT A TRADE SEQUENCE DIAGRAM	- 56 -
5.6.3. EXAMPLE UML LANGUAGES.....	- 56 -
5.7. STATE DYNAMICS VIEWPOINT	- 57 -
5.7.1. DESIGN CONCERNS.....	- 57 -
5.7.2. DESIGN ELEMENTS	- 57 -
5.7.3. EXAMPLE LANGUAGES	- 59 -
5.8. RESOURCE VIEWPOINT	- 59 -
6. PLANNING	- 60 -
6.1. TEAM STRUCTURE	- 60 -
6.2. ESTIMATION (BASIC SCHEDULE).....	- 60 -
6.3. PROCESS MODEL	- 61 -
7. CONCLUSION	- 61 -

TABLES

TABLE 1 - ABBREVIATIONS, ACRONYMS OR DEFINITIONS 3
TABLE 2 - EXPLANATIONS ABOUT MAIN STATES OF AlgorithmicTrader 58

FIGURES

FIGURE 1 - LOGIN SCREEN USE CASE DIAGRAM	14
FIGURE 2 - ENTER USERNAME USE CASE DIAGRAM	15
FIGURE 3 - ENTER PASSWORD USE CASE DIAGRAM	15
FIGURE 4 - FORGOT YOUR PASSWORD USE CASE DIAGRAM	15
FIGURE 5 - LOGIN USE CASE DIAGRAM	16
FIGURE 6 - MAIN SCREEN USE CASE DIAGRAM	16
FIGURE 7 - SELECT ACTIVE TRADE USE CASE DIAGRAM	17
FIGURE 8 - STOP/RESUME TRADE USE CASE DIAGRAM	17
FIGURE 9 - SIMULATE USE CASE DIAGRAM	17
FIGURE 10 - SHOW TRADE LOG USE CASE DIAGRAM	18
FIGURE 11 - SHOW TRADE DETAILS USE CASE DIAGRAM	18
FIGURE 12 - SHOW PORTFOLIO USE CASE DIAGRAM	16
FIGURE 13 - SHOW STOCK EXCHANGE MARKET DATE USE CASE DIAGRAM	19
FIGURE 14 - SELL/BUY TRADE USE CASE DIAGRAM	19
FIGURE 15 - OPTION MENU USE CASE DIAGRAM	19
FIGURE 16 - CREATE NEW TRADE USE CASE DIAGRAM	20
FIGURE 17 - NEW TRADE SCREEN USE CASE DIAGRAM	20
FIGURE 18 - SELECT TRADE KIND USE CASE DIAGRAM	21
FIGURE 19 - ENTER EXPIRATION DATE USE CASE DIAGRAM	21
FIGURE 20 - ENTER SIZE USE CASE DIAGRAM	21
FIGURE 21 - SELECT STRATEGY USE CASE DIAGRAM	22
FIGURE 22 - START TRADE USE CASE DIAGRAM	22
FIGURE 23 - CANCEL TRADE USE CASE DIAGRAM	22
FIGURE 24 - LOG SCREEN USE CASE DIAGRAM	23
FIGURE 25 - COMPLETE USE CASE DIAGRAM OF ALGORITHMICTRADER	24
FIGURE 26 - COMPOSITION DIAGRAM OF ALGORITHMICTRADER	29
FIGURE 27 - CLASS DIAGRAM OF ALGORITHMICTRADER	44
FIGURE 28 - LOGIN SCREEN OF ALGORITHMICTRADER	46
FIGURE 29 - NEW TRADE SCREEN OF ALGORITHMICTRADER	46
FIGURE 30 - MAIN SCREEN OF ALGORITHMICTRADER	47
FIGURE 31 - LOGIN/LOGOUT SEQUENCE DIAGRAM	49
FIGURE 32 - SCREEN REPLACING SEQUENCE DIAGRAM	50
FIGURE 33 - RECEIVING PORTFOLIO SEQUENCE DIAGRAM	51
FIGURE 34 - OPENING NEW TRADE DIALOG SEQUENCE DIAGRAM	52

FIGURE 35 - C++ TRADE CREATION AND FIX COMMUNICATION SEQUENCE
DIAGRAM 53

FIGURE 36 - ALGORITHM CHOICE SEQUENCE DIAGRAM 55

FIGURE 37 - DISPLAY DETAILED INFORMATION ABOUT A TRADE SEQUENCE
DIAGRAM 56

FIGURE 38 - STATE DIAGRAM OF THE SYSTEM 59

FIGURE 39 - AGILE SOFTWARE DEVELOPMENT METHOD REPRESENTATION 61

1. OVERVIEW

This software design document presents detailed information about implementation of the software. This detailed information are given in next chapters of this document with the aid of exhaustive explanations, class diagrams, sequence diagrams, use case diagrams etc.

1.1. SCOPE

This document is a guideline for implementation of the software. It gives extensive explanations and solutions related to the project. By using UML diagrams such as use case and class diagrams, fundamental structure of the software are presented clearly. Therefore, this document serves clear comprehension of how the software will be implemented.

The contents mentioned in this document do not cover completely functional software since there might be certain presumes about process. Thus, there may be some modifications about these assumptions during implementation phase of the project.

1.2. PURPOSE

This software design description document aims to explain the goals, characteristics and interfaces of the software. Also it describes constraints under which the system should perform, how the software will operate and what is expected to do from the system. The functional and non-functional requirements mentioned in SRS constitute a basis for implementation phase. This document describes how these requirements will be realized through the software.

1.3. INTENDED AUDIENCE

The intended audience for this document is both stakeholders and the developers of the project.

1.4. REFERENCES

The resources listed below are references used in requirement analysis:

IEEE Standard Documents:

- [1] IEEE Standard for Information Technology – Systems Design – Software Design Descriptions – IEEE Std. 1016 – 2009.

- [2] StarUML 5.0 User Guide. (2005). Retrieved from [http://staruml.sourceforge.net/docs/user-guide\(en\)/toc.html](http://staruml.sourceforge.net/docs/user-guide(en)/toc.html)
- [3] Hull, J. (2009). *Options, futures, and other derivatives; seventh edition (7th edition)*. Upper Saddle River, N.J.: Prentice Hall.
- [4] Using Genetic Algorithms To Forecast Financial Markets. (n.d.). Retrieved November 30, 2014, from <http://www.investopedia.com/articles/financial-theory/11/using-genetic-algorithms-forecast-financial-markets.asp>
- [5] Binary option. (2014, November 29). Retrieved November 30, 2014, from http://en.wikipedia.org/wiki/Binary_option
- [6] RELEASE: Fraudadv_binaryoptions. (n.d.). Retrieved November 30, 2014, from http://www.cftc.gov/PressRoom/PressReleases/fraudadv_binaryoptions
- [7] FIX Trading Community. (n.d.). Retrieved November 30, 2014, from <http://www.fixtradingcommunity.org/>
- [8] Bond Option Definition | Investopedia. (n.d.). Retrieved November 30, 2014, from <http://www.investopedia.com/terms/b/bondoption.asp>
- [9] Agile and Scalable. (n.d.). Retrieved January 3, 2015, from <http://www.mongodb.org/>
- [10] QuickFIX. (n.d.). Retrieved January 3, 2015, from <http://www.quickfixengine.org/>
- [11] High performance FIX protocol apps with fix8 (n.d.). Retrieved January 3, 2015, from <http://www.fix8.org/>
- [12] QuantLib: A free/open-source library for quantitative finance. (n.d.). Retrieved January 3, 2015 from <http://quantlib.org/index.shtml>
- [13] JavaFX 2 Certified System Configurations. (n.d.). Retrieved January 3, 2015, from <http://www.oracle.com/technetwork/java/javafx/downloads/supportedconfigurations-1506746.html>
- [14] JavaFX. (n.d.). Retrieved January 3, 2015, from <http://en.wikipedia.org/wiki/JavaFX>
- [15] AlgoTrades - Algorithmic Trading Strategies - Algo Trading - Futures Trading System - Trading Algorithms - Automated Trading Systems - Quantitative Trading Strategies. (n.d.). Retrieved January 3, 2015, from <http://www.algotrades.net/>
- [16] Apache Subversion. (n.d.). Retrieved January 3, 2015, from http://en.wikipedia.org/wiki/Apache_Subversion

2. DEFINITIONS

<u>ABBREVIATION, ACRONYM OR DEFINITION</u>	<u>EXPLANATION</u>
Asset	An economic resource
BIST	The sole exchange entity of Turkey combining the former Istanbul Stock Exchange (ISE) (Istanbul Menkul Kıymetler Borsası, IMKB), the Istanbul Gold Exchange (Istanbul Altın Borsası, İAB) and the Derivatives Exchange of Turkey (Vadeli İşlem Opsiyon Borsası, VOB) under one umbrella.
Database	A collection of related data
Database Query	A piece of code (a query) that is sent to a database in order to get information back from the database.
DBMS	A software package/system to facilitate the creation and maintenance of a computerized database
Financial transaction	An agreement, communication, or movement carried out between a buyer and a seller to exchange an asset for payment.
FIX Protocol	An electronic communications protocol for international real-time exchange of information related to the securities transactions and markets.
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
Message Persister	A part of software that sends messages repetitively
Portfolio	A financial term denoting a collection of investments held by an investment company, hedge fund, financial institution or individual.
Security	A tradable financial asset of any kind
SSL	Secure Socket Layer
Stock	A type of security that signifies ownership in a corporation and represents a claim on part of the corporation's assets and earnings.

Trade	An exchange of a security (stocks, bonds, commodities, currencies, derivatives or any valuable financial instrument) for "cash"
Option	A contract that gives the buyer the right, but not the obligation, to buy or sell an underlying asset at a specific price on or before a certain date
UML	Unified Modelling Language

TABLE 1: ABBREVIATIONS, ACRONYMS OR DEFINITIONS**3. CONCEPTUAL MODEL FOR SOFTWARE DESIGN DESCRIPTIONS**

The conceptual model contains main terms and concepts of the project used during preparation of the SDD context. Also it includes the stakeholders who use them, and how these terms and concepts are used.

3.1. SOFTWARE DESIGN IN CONTEXT

The aim and intended use of the final product is to trade with BIST by using related algorithm which is chosen according to trading option. In order to realize this goal, object oriented paradigm in a modular fashion is aimed to be used during implementation of this project. Software structure is designed by considering modularity property in all aspects and that can be observed in class diagram clearly. Especially, following two issues are constructed based upon modularity principle in design of the software. Firstly, algorithms can be enlarged in terms of content and number in the software; therefore, new trading choices are served to the users of the product. Secondly, interface can be changed by developers easily according to users' requirements since JavaFX based structure of the system supports modularity principle. Besides, adaptability principle is guarded throughout design. The final software product is aimed to run on Ubuntu, Windows and MacOS. This cross-platforms supporting property is the most explicit characteristics of adaptability principle used in design of the product. In addition to these properties, FIX protocol, JavaFX, MongoDB and several required libraries will be used in development process.

3.2. SOFTWARE DESIGN DESCRIPTIONS WITHIN THE LIFE CYCLE

3.2.1. INFLUENCES ON SDD PREPARATION

Software requirements specification document of this project is fundamental for preparation of this software design description document.

3.2.2 INFLUENCES ON SOFTWARE LIFE CYCLE PRODUCTS

This project is a client-server application. All of the necessary calculations and other performance required background operations will be performed on C++ part and users will be informed via JavaFX interface. Java and C++ parts act as server-client applications. Likewise, same connection logic can be considered to exist between C++ part of the software and BIST server via network and FIX protocol. C++ part of the project is the most critical section due to performance requirements; therefore, great attention should be paid during implementation of this part. Because of this structure of the software, both C++ and Java(GUI) parts of the software should be developed in parallel to control compatibility of these parts in lifecycles.

The simplest accessible trading algorithm will be implemented at first step since financial algorithms are very deep research area in algorithmic trading field. Therefore, parts related to algorithms will not be focused on initially. All in all, we aim to implement C++ and Java parts, then related connections will be established and finally, implementation of several algorithms will be performed.

3.2.3. DESIGN VERIFICATION AND DESIGN ROLE IN VALIDATION

When user starts the application, he/she is waited for entering username and password. Software checks them to match the ones in database. These are required for successful login. User can create new trade when he/she is in main screen. To achieve this, the user is expected to fulfill necessary fields in Java interface and then these are sent to C++ part via socket connection. Therefore, request of the new trade can be prepared after performing of suitable algorithm then this prepared trade order is sent to BIST in server side of the application. After this, BIST is expected to return a verification message about sent trade order. Upon this, information about current status of the trade and related shares of that trade is sent to Java interface to be displayed on screen. These steps define successful execution of buy/sell trade order according to user's trading choice.

Each action performed in execution of the application is recorded to database MongoDB as logs. As a result of this, security of our project is guaranteed.

4. DESIGN DESCRIPTION INFORMATION CONTENT

4.1. INTRODUCTION

In this section of the SDD, it is intended to give necessary information about how the design of the software will be explained in the following sections. SDD identification information, identified design stakeholders, identified design concerns will be stated. Selected design viewpoints, each with type definitions of its allowed design elements and design languages, design view, design overlays and design rationale will be explained as well.

4.2. SDD IDENTIFICATION

This is the initial Software Design Description for the project Algorithmic Trading. This SDD is completed at the date of 04.01.2015. Note that this SDD is not a final document for this project, some modifications can be carried out during implementation of the software. The organization name for this SDD MoneyFellas which consists of Ali Şavkar, Gökçer Yapar, Murat Tarimer and Ömer Yavuz. Doctor Selim Temizer and Assistant Mehmet Çelik are supervisors of our project. Design information about Algorithmic Trading constitutes scope of this document. UML diagrams will be used primarily to explain the design viewpoints.

This Software Design Description document is prepared according to STD; IEEE 1016-2009.

4.3. DESIGN STAKEHOLDERS AND THEIR CONCERNS

Doctor Selim Temizer, Assistant Mehmet Çelik and other CENG 490 staff are the design stakeholders of this SDD document. Professor Ahmet Coşar is one of the instructors of the CENG 490 and he is our official supervisor. Assistant Mehmet Çelik is one the teaching assistants of CENG 490 course at Middle East Technical University and he is the one who is responsible for our team. Each week, a meeting is arranged by participating of Assistant Mehmet Çelik and our team. Thus, important feedbacks can be able to get from him about development process. He can be considered as main contact between CENG 490 staff and our team MoneyFellas.

4.4. DESIGN VIEWS

The software will be implemented by using object oriented paradigm which enables to form modular structure; thus, the stakeholders can add new properties or can remove any unwanted ones from product. Since object oriented paradigm is used as design pattern, any update can be integrated without too much effort. User will see login screen first, after successful login attempt, they will see main screen of AlgorithmicTrader. When user initiates a trade, trade order will be sent from Java part to C++ part of the software. Then C++ prepares FIX message data according to the trade order and sends it to BIST. C++ parts communicate with BIST via FIX protocol; also, it sends data to Java interface to draw graphics. By the way, each trade action and related shares values in time will be recorded to database to ensure safety.

Product context is specified and restricted to limitations which are mentioned in SRS document in this project. Diagrams explain and support logical view of the software in following sections of the document. Therefore, relationships between modules or classes of the software can be understood clearly. Besides, possible future problems and how the information is stored and shared among the users are shown in dependency and information viewpoints. Lastly, actions' flow and transitions of states are shown in state dynamic views.

As a result, context, dependency, logical, patterns use, interface, state dynamics and interaction views are used as design views in this SDD document. Note that these design views correspond to design viewpoints in the following section 4.5. Each design view governed by a design viewpoint.

4.5. DESIGN VIEWPOINTS

Context viewpoint describes the relationships, dependencies, and interactions between our software product and its environment which consists of user, BIST and other external entities with which it interacts. Besides it shows what AlgorithmicTrader software does and cannot do and where the boundaries are between our software and the outside world.

Dependency viewpoint provides overall picture of the design subject in order to assess the effect of the requirements or design changes in the project. Interconnection of classes or modules and sharing of resources are main concerns in this viewpoint. UML composition diagram is used to explain this viewpoint in following parts of the document.

Logical viewpoint is used to address development and reuse of adequate abstractions and their implementations in the project. Static structure which are classes, interfaces and their implementations are main concerns. Also reuse of types and implementations can be said as main concern. UML class diagram is used to explain this viewpoint in following parts of the document.

Patterns use viewpoint addresses design ideas as collaboration patterns involving abstracted roles and connectors in the project. Reuse of patterns is the main concern. UML composite structure diagram is used to explain this viewpoint in following parts of the document.

Interface viewpoint provides information about the means to know how to correctly use the services provided by the design subject for designers, programmers and testers in the project. Definitions and access for services are the main concerns. UML component diagram is used to explain this viewpoint in following parts of the document.

State dynamics viewpoint explains reactive systems and systems whose internal behavior is of interest in our system. Dynamic state transformation is the main concern. UML state machine diagram is used to explain this viewpoint in following parts of the document.

Interaction viewpoint defines strategies for interaction among modules, classes and other entities in the system. Communication of objects and messaging are the main concerns. UML sequence diagram is used to explain this viewpoint in following parts of the document.

4.6. DESIGN ELEMENTS

A design element is any item occurring in a design view. A design element may be any of the following subcases: design entity, design relationship, design attribute, or design constraint.

The type of each design element shall be introduced within exactly one design viewpoint definition. A design element may be used in one or more design views.

4.6.1. DESIGN ENTITIES

Design entities capture key elements of a software design.

4.6.1.1. SERVER SYSTEM

This system constitutes the most important part of the backside of our product. Almost all necessary calculations and connections between other servers are handled in this part. For example, processes like user and admin connection, getting information from packages and distributing them to related parts are handling in this part. Therefore, this part of the product can be thought like a bridge which link all packages of the program to each other.

4.6.1.2. CLIENT SYSTEM

Client side of the software is designed for cross platforms which are Linux, Windows and Mac OS. This part is actually server side interface of our software implemented by JavaFX. By using this system, user can supply required information which are necessary to execute desired processes to server side. In conclusion, users are capable of managing related server side parts thanks to this client interface system without knowing anything in detail about what is happening at server side.

4.6.1.3. ALGORITHMS SERVER SYSTEM

This system is the subsystem of program server system mentioned in 4.6.1.1. Main logic of this part is manipulation of stored stock exchange market data to make estimation about next sell/buy order. Thanks to algorithms, our product tries to make most accurate decision according to user choices in this part of the software. As a result of this, one of the purpose of our product which is making best investment according to user choices is achieved.

4.6.1.4. COMMUNICATION COMPONENT

This component is used to create connections and to manage these created connections during execution life cycle of our software. Therefore, some components which require to send/receive information can use related parts of this communication component. Note that socket communication between Java and C++ parts is still analyzed about where to locate these methods. Probably, it will be decided during implementation phase.

4.6.1.5. FIX8 FRAMEWORK

Fix8 is a FIX framework implemented in C++, provides client/server session and connection classes having SSL support for the standard FIX field sorts which are FIX asynchronous message persister, printer, asynchronous logger, and XML configuration classes. Fix8 can easily be enlarged and customized, so these open source libraries shall be used from developer team of the software while implementation of the FIX communication part of our product.^[11]

4.6.1.6. MONGODB DATABASE

One of the popular data storing system whose type NoSQL is MongoDB. This database is used while storing all stock exchange data within a specific time period. Then, these stored datum will be used to make estimation about future prices of shares using algorithms. There are two main properties which should be attached importance to. Firstly, there is no data manipulation processes over saved datum during execution of algorithms. Secondly, speed of datum analysis is vital in our project. Since MongoDB is the most appropriate database management system which meets above properties, it is chosen as data storing system for this part of software. ^[9]

4.6.1.7. MYSQL DATABASE

MySQL is an open source relational database management system(RDBMS) based on structured query language(SQL). This data storage system will be used to store relational user datum such as personnel information or active trades' information of a user.

4.6.1.8. JAVAFX SOFTWARE PLATFORM

JavaFX can be considered as a software platform that enable to create and deliver vast GUI applications that can run across a wide variety of devices thanks to Java Runtime Environment base. Developer team of our product shall use JavaFX for all GUI parts of the software. ^[14]

4.6.1.9. C++ PROGRAMMING LANGUAGE

C++ is one of the most famous general-purpose programming languages in software world. It enables to manipulate the low-level memory management facilities and it has generic programming, imperative and object-oriented features.

Because of these features, developer team of the software shall use C++ for back-end implementation.

4.6.1.10. APACHE SUBVERSION

Apache Subversion is distributed under the Apache License freely. This software provides to create software with versions and to construct revision control system to check source code. In order to maintain current and historical versions of files and directories, Subversion can be used by software developers. Also, developer can recover older versions their committed data thanks to Subversion. Therefore, developers of the software should use Apache Subversion to maintain software development in a well-structured way. ^[16]

4.6.2. DESIGN ATTRIBUTES

Information about design attributes can be achieved from above section 4.6.1. Design Entities. If the explanations about design attributes were given in this section, this would be repetition. Therefore there will be no description in this part, any necessary information about the attributes can be attained from above section 4.6.1. Design Entities.

4.6.3. DESIGN RELATIONSHIPS

As described above part 4.6.1. Design Entities, main systems that are related to our software are Server System, Client System, Algorithms Server System, Communication Component, Fix8 Framework, MongoDB Database, MySQL Database, JavaFX Software Platform, C++ Programming Language and Apache Subversion.

Server System has relationships with Client System, Communication System and the systems including any database.

Client system is interface of our software, it has relationships with Server System in backside.

Algorithms Server System can be considered as a subsystem of Server System. It has relationships with Server System and MongoDB Database.

Communication Component helps to access to BIST. It has relationships with Fix8 Framework, Server System.

Other less important relationships between entities or components of our project will be presented in Logical Viewpoint part.

4.6.4. DESIGN CONSTRAINTS

All design entities described in this design document should be implemented during implementation phase. Relationships mentioned in above section between entities should be preserved during implementation phase as well. Besides, object oriented paradigm should be followed totally in any phase of the development process. It should be attached great importance to all object oriented notions especially reusability, encapsulation and modular structure.

The classes should be defined well in design phase and they should be coded neatly in implementation phase; moreover, developers should be careful about neat comments during implementation.

4.7. DESIGN OVERLAYS

There is no design overlay that is used for presenting additional information with respect to an already defined design view.

4.8. DESIGN RATIONALE

Some significant features like sustainability, maintainability, performance, reliability and security affect the design choices of the project. These decisions can be updated when necessary based upon requirements of stakeholders or users. When implementation phase of the project, it will be attached great importance to comments in methods and fields. Therefore, different developers can understand and modify the any code part when necessary with the aid of these comments.

Because of the decisions mentioned above paragraph, we choose object-oriented approach to design the project. All systems and sub-systems will be organized and explained in this design document. Components should become independent from others that means any operations should not affect others directly.

4.9. DESIGN LANGUAGES

Unified Modeling Language (UML) is used to create this document and design of the software. StarUML is selected as the modelling tool.

5. DESIGN VIEWPOINT

5.1. INTRODUCTION

In this part, seven main design viewpoints of the Algorithmic Trading project will be explained briefly. The viewpoints are explained, with order, as follows:

- Context viewpoint
- Composition viewpoint
- Logical viewpoint
- Interface viewpoint
- Interaction viewpoint
- State dynamics viewpoint
- Resource viewpoint

Besides, each viewpoint will be supported by specific UML diagrams to provide understandable content about working processes of the system.

5.2. CONTEXT VIEWPOINT

AlgorithmicTrader software system's context viewpoint provides information about user functionalities of our application. There are four major screens controlled by the user which are Login Screen, Main Screen, New Trade Screen and Logs Screen. Each of them consists of subsections.

5.2.1. DESIGN CONCERNS

There are four main service categories related to our software product. Each of these shows parallelism with a screen of the application. Therefore, these main service categories will be analyzed in terms of Login Screen, Main Screen, New Trade Screen and Log Screen categories.

Login Screen can be considered as initial screen of the application. In order to use core services of our product, username and password fields should be filled correctly in this screen. These username and password fields are tried to be matched with ones in database. If the authentication is successful, then this screen is shut down and Main Screen is opened.

Main Screen have almost all core services of our application. All main and important task can be reached by using this screen. Creation of new trades, checking status of any active trades and controlling assets of user's assets can be said as few examples of these main services.

New Trade Screen can be reached by clicking New Trade button in main screen. When clicked this button, New Trade dialog box is opened. User can fulfil the fields in this dialog box and he/she can start a new trade according to his/her choices.

Log Screen can be reached by clicking Log button in Main Screen. All successful or unsuccessful transactions and operations related to selected trade can be displayed in this window. Normally, all details is not shown on screen but some expert users and developers may want to check each action carried by the software. Therefore, we will present this service. Note that logs will be received from database.

5.2.2. DESIGN ELEMENTS

5.2.2.1. LOGIN SCREEN USE CASE DIAGRAM

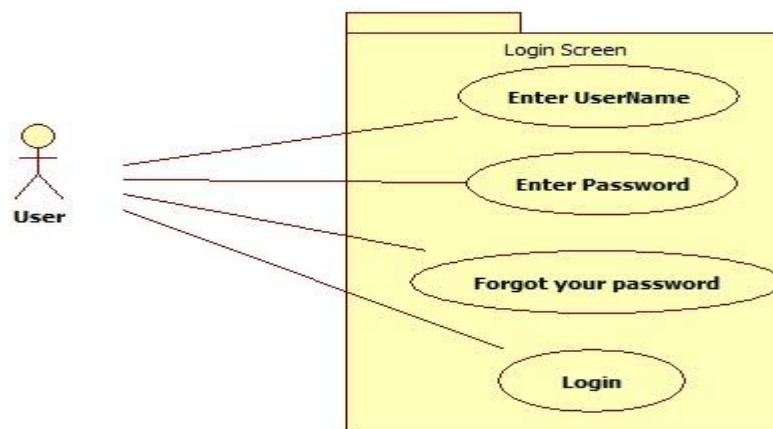
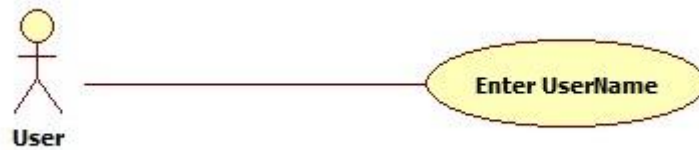


Figure 1: LOGIN SCREEN USE CASE DIAGRAM

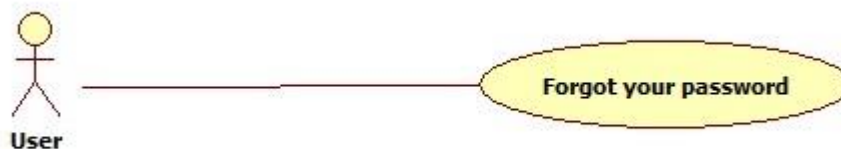
Login Screen has four user interaction sections. They can be seen above use case diagram.

5.2.2.1.1. ENTER USERNAME USE CASE DIAGRAM**Figure 2:** ENTER USERNAME USE CASE DIAGRAM

After application is started, user can write his/her username in the 'Username' field in Login Screen.

5.2.2.1.2. ENTER PASSWORD USE CASE DIAGRAM**Figure 3:** ENTER PASSWORD USE CASE DIAGRAM

After application is started, user can write his/her password in the 'Password' field in Login Screen.

5.2.2.1.3. FORGOT YOUR PASSWORD USE CASE DIAGRAM**Figure 4:** FORGOT YOUR PASSWORD USE CASE DIAGRAM

After application started, if user does not remember his/her password, user can reset own password by using this field.

5.2.2.1.4. LOGIN USE CASE DIAGRAM

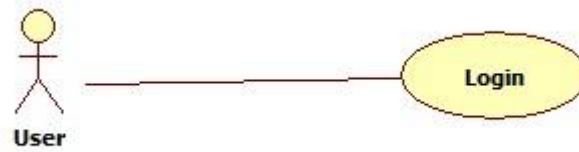


Figure 5: LOGIN USE CASE DIAGRAM

After application is started, user can login to the system successfully after filling Username and Password fields correctly by clicking 'Login' button in Login Screen.

5.2.2.2. MAIN SCREEN USE CASE DIAGRAM

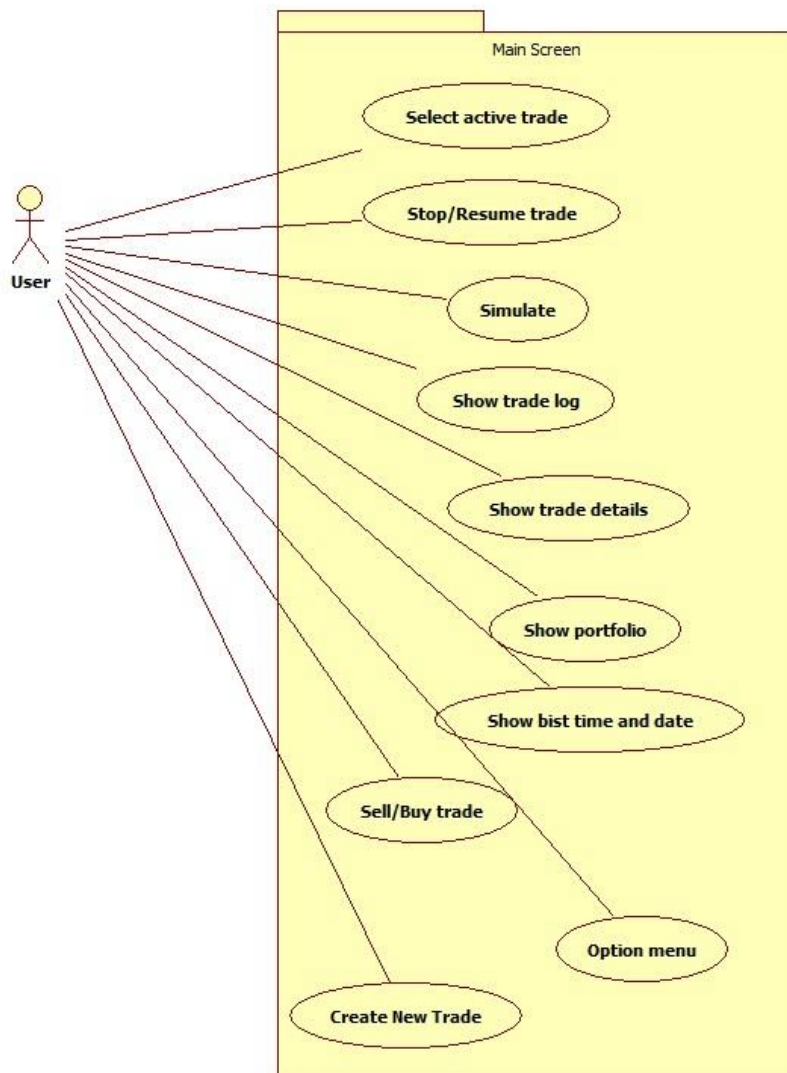
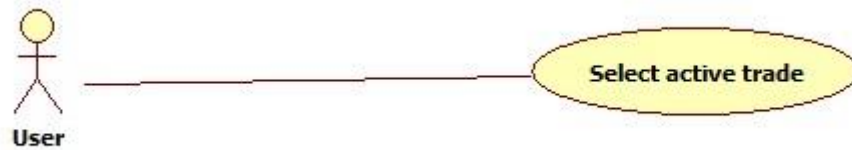
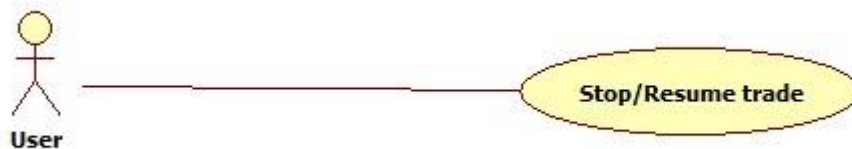


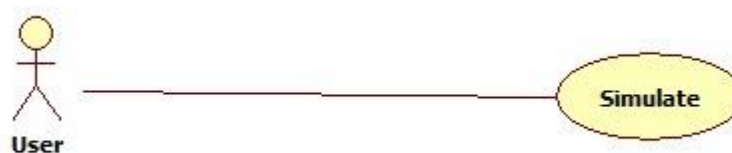
Figure 6: MAIN SCREEN USE CASE DIAGRAM

5.2.2.2.1. SELECT ACTIVE TRADE USE CASE DIAGRAM**Figure 7:** SELECT ACTIVE TRADE USE CASE DIAGRAM

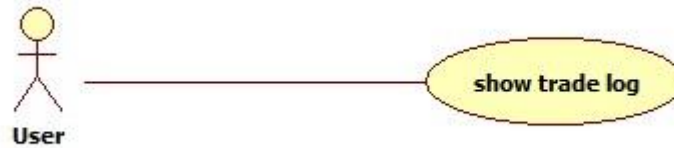
User can select an open position from the open position table in order to make some operations such as showing details of an open position element or simulating it in Main Screen.

5.2.2.2.2. STOP/RESUME TRADE USE CASE DIAGRAM**Figure 8:** STOP/RESUME TRADE USE CASE DIAGRAM

User can stop or resume any active trade in the open position table in Main Screen.

5.2.2.2.3 SIMULATE USE CASE DIAGRAM**Figure 9:** SIMULATE USE CASE DIAGRAM

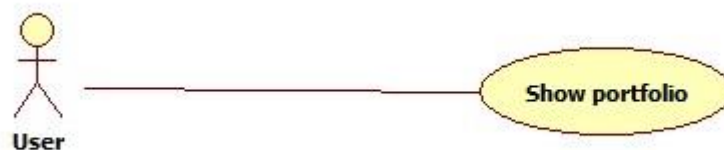
Current active trades information will be displayed on the screen as graphics in Main Screen, more information can be achieved from Interface Viewpoint section.

5.2.2.2.4. SHOW TRADE LOG USE CASE DIAGRAM**Figure 10:** SHOW TRADE LOG USE CASE DIAGRAM

In Main Screen, user can check his/her all trade information in detail such that whether sell trade order is accepted from stock exchange server or not. Therefore, user will be aware of when any failure exists in the system.

5.2.2.2.5. SHOW TRADE DETAILS USE CASE DIAGRAM**Figure 11:** SHOW TRADE DETAILS USE CASE DIAGRAM

When user clicks twice on an open position, details of it such as expiration date, size and type or creation date will be shown to the user in Main Screen.

5.2.2.2.6. SHOW PORTFOLIO USE CASE DIAGRAM**Figure 12:** SHOW PORTFOLIO USE CASE DIAGRAM

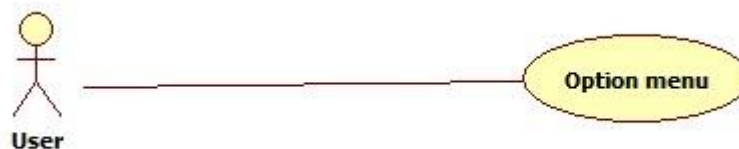
Thanks to this part, user will be informed about his/her general profit/loss amount information in Main Screen. This information will be shown on the screen.

5.2.2.2.7. SHOW STOCK EXCHANGE MARKET DATE USE CASE DIAGRAM**Figure 13:** SHOW STOCK EXCHANGE MARKET DATE USE CASE DIAGRAM

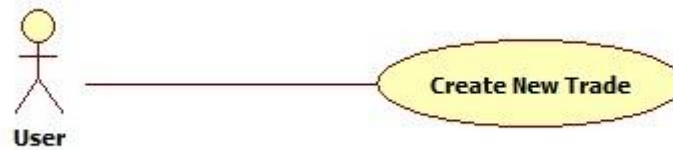
This part will be used to show exact detail date for related stock exchange market in Main Screen. This stock exchange is BIST for our project.

5.2.2.2.8. SELL/BUY TRADE USE CASE DIAGRAM**Figure 14:** SELL/BUY TRADE USE CASE DIAGRAM

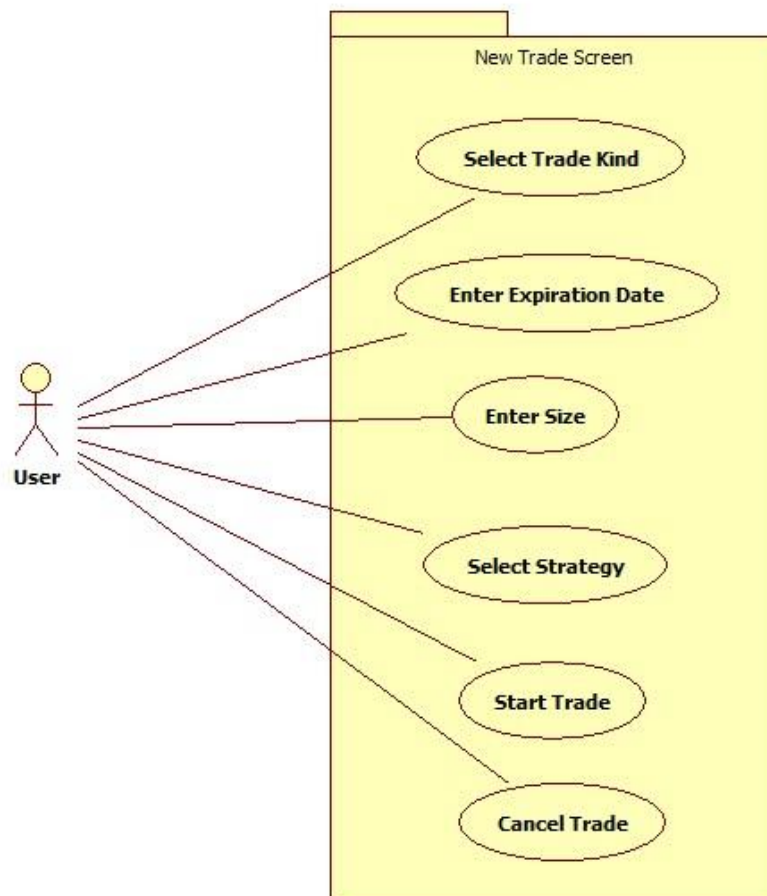
User can operate his/her assets directly with choosing sell/buy trade order and its amount in Main Screen.

5.2.2.2.9. OPTION MENU USE CASE DIAGRAM**Figure 15:** OPTION MENU USE CASE DIAGRAM

This menu includes lots of setting options about the software such as updating general user information in Main Screen. More information can be achieved from Interface Viewpoint section of this document.

5.2.2.2.10. CREATE NEW TRADE USE CASE DIAGRAM**Figure 16:** CREATE NEW TRADE USE CASE DIAGRAM

User can trigger new trade screen via pressing create new trade button to create new trade in Main Screen. New Trade Screen is opened in front of Main Screen.

5.2.2.3. NEW TRADE SCREEN USE CASE DIAGRAM**Figure 17:** NEW TRADE SCREEN USE CASE DIAGRAM

This screen is opened as dialog box in front of Main Screen. It lets user fulfil required fields to be able to start a new trade. Note that New Trade Screen and new trade dialog box is used interchangeably instead of each other in this SDD document.

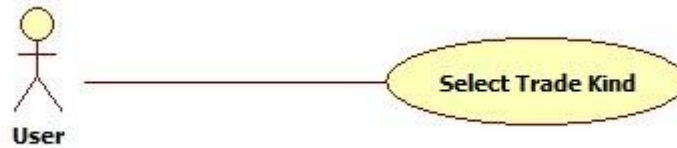
5.2.2.3.1. SELECT TRADE KIND USE CASE DIAGRAM

Figure 18: SELECT TRADE KIND USE CASE DIAGRAM

User can determine the type of trade to be created in new trade dialog box.

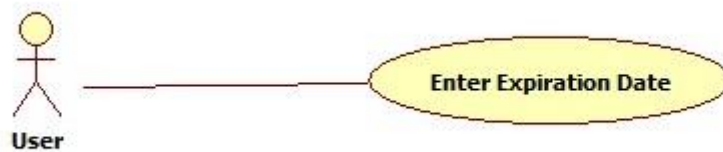
5.2.2.3.2. ENTER EXPIRATION DATE USE CASE DIAGRAM

Figure 19: ENTER EXPIRATION DATE USE CASE DIAGRAM

User can determine the expiration date of trade to be created by filling 'Expiration Date' field in new trade dialog box.

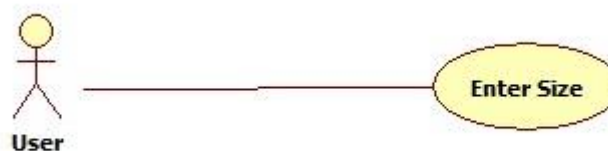
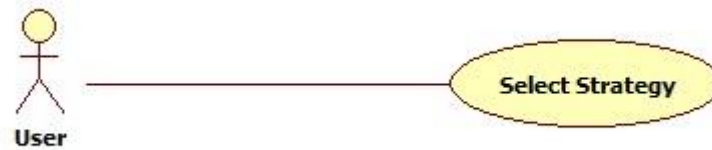
5.2.2.3.3. ENTER SIZE USE CASE DIAGRAM

Figure 20: ENTER SIZE USE CASE DIAGRAM

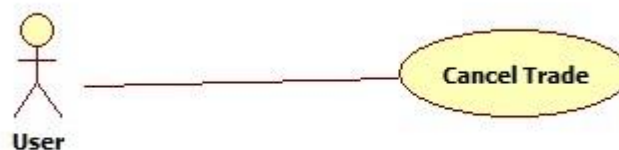
User can determine the size of trade to be created by filling 'Size' field in new trade dialog box.

5.2.2.3.4. SELECT STRATEGY USE CASE DIAGRAM**Figure 21:** SELECT STRATEGY USE CASE DIAGRAM

User can determine the strategy of trade to be created in new trade dialog box. In backside of the application, this strategy choice is evaluated and it is decided to use which algorithm/algorithms to be used.

5.2.2.3.5. START TRADE USE CASE DIAGRAM**Figure 22:** START TRADE USE CASE DIAGRAM

In New Trade Screen, user can start trade by clicking start button after fulfilling all required fields.

5.2.2.3.6. CANCEL TRADE USE CASE DIAGRAM**Figure 23:** CANCEL TRADE USE CASE DIAGRAM

In New Trade Screen, user can decide to give up starting trade by clicking cancel button in any time. When clicked this cancel button, new trade dialog box is shut down.

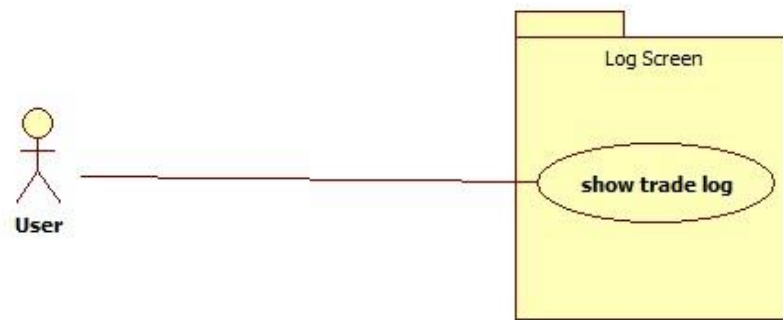
5.2.2.4. LOG SCREEN USE CASE DIAGRAM

Figure 24: LOG SCREEN USE CASE DIAGRAM

In this screen, user can display all logs of selected trade, these logs are received form database.

5.2.3. EXAMPLE UML LANGUAGE

Complete UML use case diagram of our system can be shown as follows. Note that services that belong to a screen can only be reached while the screen is displayed to the user.

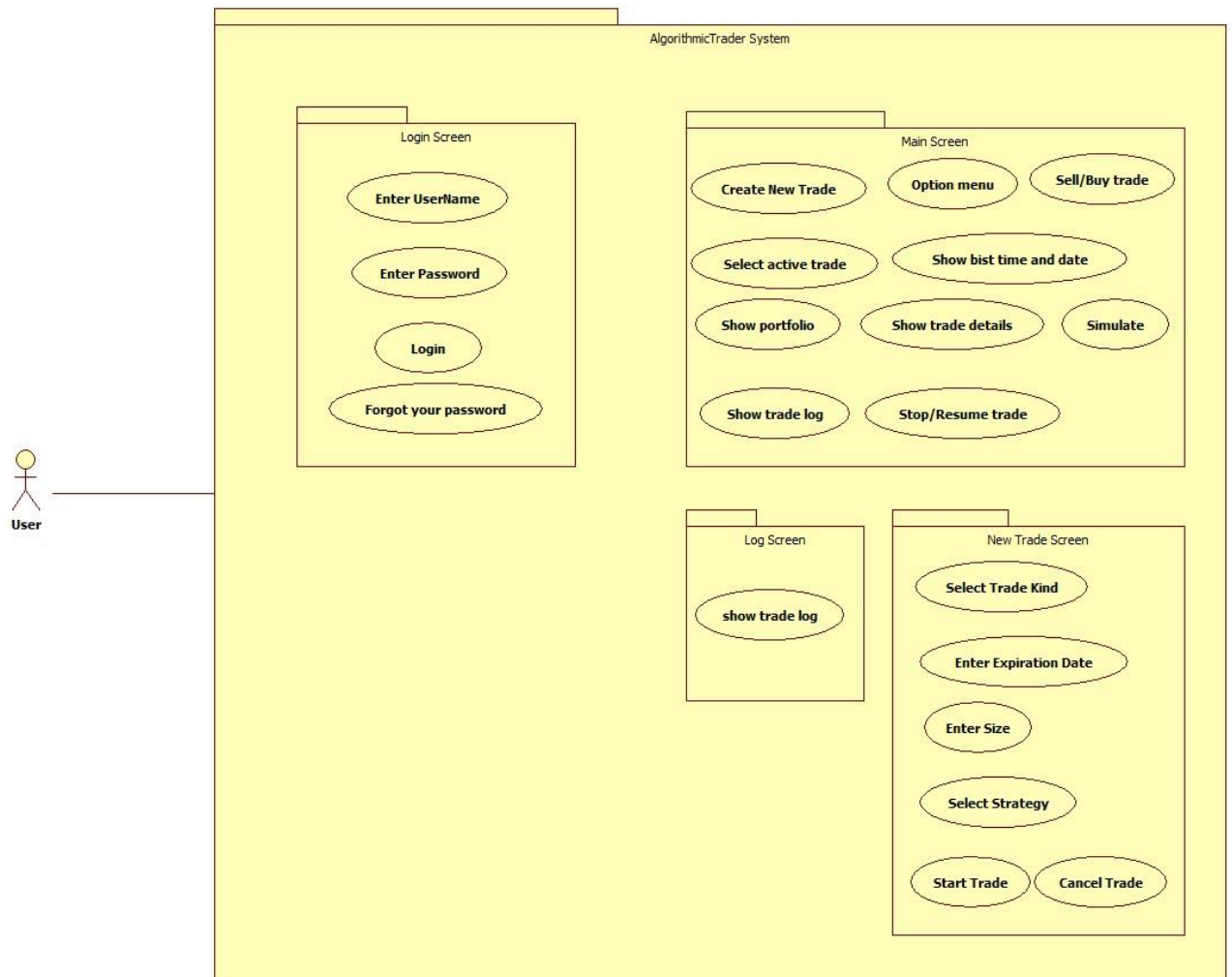


Figure 25: COMPLETE USE CASE DIAGRAM OF AlgorithmicTrader

5.2.4. DESIGN CONSTRAINTS

Quality is important with respect to three aspects for our software. One of them is fast issue. Since a share can be purchased by other one/ones, if there is a buy order for any share, that order is completed as much as possible. Besides, since our algorithms use huge amount of datum to make a decision that is sell/buy order, they should be implemented in most efficient way. Other two issues are security and safety. Safety is important because of in case of any failure, huge amount loss can be discussed. Also, security is important because datum that have high privacy level are processed continuously. Therefore, great importance should

be attached to this security issue and any information should not be achieved by any third part of software or people.

5.2.5. DESIGN RELATIONSHIPS

In Login Screen, username and password are expected from user and they are checked with ones in database. If there is match, Login Screen is closed and Main Screen is opened.

In Main Screen, there are many relationships between modules and classes in our software. Recording logs to database, getting necessary information to draw graphics from database, getting required datum to be processed by algorithms, communication with BIST via FIX protocol, communication with C++ and JavaFX parts of our software can be said as most important relationships in the application during execution.

In New Trade Screen, necessary fields are filled by user and this information is sent to C++ server part of the software via socket communication.

In Log Screen, detailed information are received from database and it is shown in JavaFX interface to the user.

5.3. COMPOSITION VIEWPOINT

5.3.1. DESIGN CONCERNS

This viewpoint helps to clarify some of the concerns such that cost, staffing and schedule for the development effort. Therefore, main and sub packages will be described in this part.

There will be tree main packages which are Server System, Client system and Algorithms Server System. Through the design of our software, Server System shall be implemented first because other main packages like Algorithms Server and Client System are strictly depended on the Server System package. After some of the critical part of Server System are implemented, other main packages which are mentioned in previous sentence will be ready to implement. In addition to main packages, there are sub packages and components to complete our system design sufficiently and they are described in below component diagram.

5.3.2. DESIGN ELEMENTS

Design Entities:

Main design entities of our software are Server System, Client System and Algorithms Server System. Server System includes subcomponents which are Communication Component, Trades Component and Database Component. Also, each component consists of other sub components. One of the subcomponent is Communication Component. This component has two subcomponent named as C++ component and Java component. In addition, these two subcomponent has c++ server-client part and Java server-client part respectively. Moreover, Communication Component uses some library and framework like fix data library and FIX8 framework. The other subcomponent is Trades Component and it has three subcomponent which are name as TradeOrder, Trade and MarketData. The last component is Database Component and it consists of just one subcomponent named as DBController. However this subcomponent has three subparts which are FIXDataRepositoryControllerDB, AccountControllerDB and TradeLogControllerDB. Client System includes three essential subcomponent which are DisplayRunner, Display and OpenPosition. Also, Display component has subparts which are named

as LoginController and MainController. These subparts uses a platform which is called as JavaFxSoftware Platform to design graphical user interface of our product.

Algorithms Server System has just one main subcomponent named as AlgoController and also this component has two subpart which are named as BinaryOption and BondOption.

Design Relationships:

In design of our software, almost all component has relation between each other so in this part these relations will be explained in detail. Communication Component which is explained in previous part has relation with almost all components because it is constructed as a bridge between each component. Hence, in design of software class diagram, association relation is used between Communication Component and others. Moreover, subparts of this component which are named as SocketCommunication and FIX have association relation with other connected components implicitly. Trades Component have three subparts as mentioned previous section. We can call these subparts as subclass of related component. One of the class of this component is "Trade" and this class is connected to "TradeOrder" class with aggregation relation. This relation type is chosen because "Trade" class has an element from "TradeOrder" class. On the other hand, one of the subclass of this component is "MarketData". This class is connected to "Trade" class using association relation since "Trade" class uses an element from this class. In addition to this, "TradeOrder" class has a aggregation relation with FIX class of Communication Component because fix connection is required to send created trade order. Other than these inner relations, "Trade" class is connected with Database, AlgorithmsServerSystem and DisplayRunner with association as outer relation. In Database Component, DBController is designed as parent class. Also, this parent class has two subclass which are FIXDataRepositoryControllerDB and AccountControllerDB. Therefore, these parts connected to each other with generalization to apply well-designed object oriented implementation. In addition to this, association relationship is used between this component and other related components to make it accessible. DisplayRunner class which is belong to Client System has composition relation with Display class. Also, Display class is designed as a parent class of other two subclasses which are LoginController and MainController. Therefore, these two

class are connected to Display class with generalization relation. In addition to this, one of the class of this component named as "OpenPositions" class has aggregation relationship with MainController class. This relation type is chosen because displaying content of main screen is strictly depend on this class. Furthermore, MainController class has association with Communication Component because it uses an object from that class. AlgoController is a class of the Algorithm Server System and this part is a parent class of other subclass which are BinaryOption and BondOption. Therefore, there is generalization relationship between this class and other two class. Also, this part has two association between Trade and FIXDataRepositoryDB classes because these two related class use some part of AlgoController class during life cycle of the software.

Design Attributes:

Detailed description about attributes of our software is given in part 4.6.1. Therefore, same information will not be explained again in this part to avoid repetition.

5.3.3. EXAMPLE UML LANGUAGE

UML composition diagram can be shown as follows:

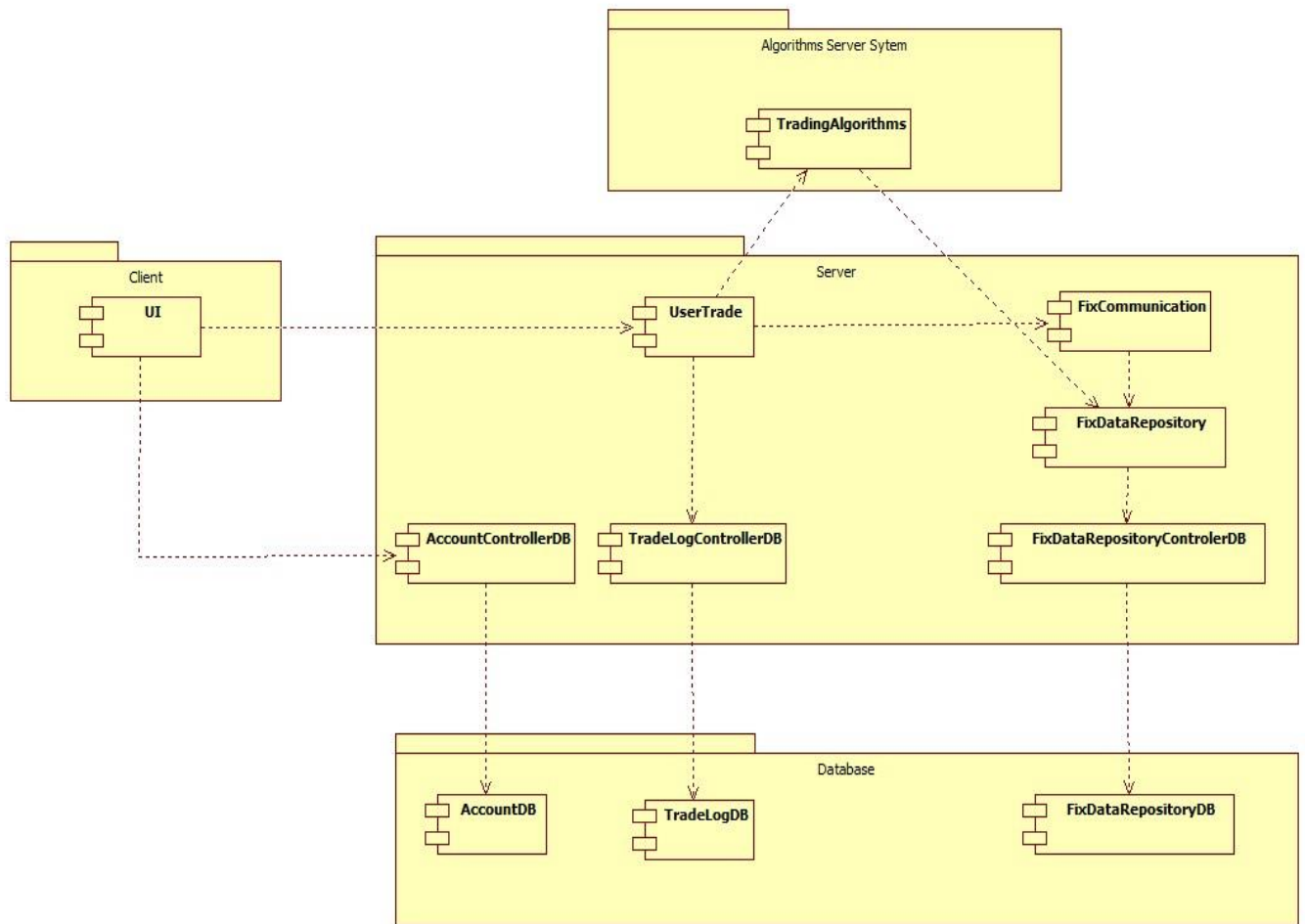


Figure 26: COMPOSITION DIAGRAM OF AlgorithmicTrader

5.4. LOGICAL VIEWPOINT

The purpose of the Logical viewpoint is to elaborate existing and designed types and their implementations as classes and interfaces with their structural static relationships.

5.4.1. DESIGN CONCERNS

The purpose of the Logical viewpoint is to describe the static structure in which compile time entities, associations and inheritance among them in full detail. The design view related to the logical viewpoint depends on class diagram. The classes and interfaces with their structural static relationships are represented in below.

5.4.2. DESIGN ELEMENTS**5.4.2.1. Users NAMESPACE**

Purpose of this namespace is to keep all user implementations together.

5.4.2.1.1. Account

The responsibility of this class is to hold and manipulate all information related to users.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
name	String	private	The name of the user.
surname	String	private	The surname of the user.
username	String	private	The username of the user.
password	String	private	The password of the user.
email	String	private	The email address of the user.
userID	Integer	private	The ID of the user kept in relational database.

5.4.2.2. Trades NAMESPACE

Purpose of this namespace is to keep all implementations related to trade and stock market data together.

5.4.2.2.1. Trade

Trade class stores information of all attributes of trades. This object is constructed when user creates a new trade and starts to run immediately. Users will reach this information on the main screen and interact with trade when he/she wants.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
ID	Integer	private	The ID of the trade.
tradeKind	String	private	This field represents which stock market trading area will be interested in.
expirationDate	Date	private	This field represents how long trade will be active.
size	Double	private	The amount of money that will be invested for the trade.
maxProfitBound	Integer	private	This field defines the maximum profit amount.
maxLossBound	Integer	private	This field defines the maximum loss amount.
strategy	String	private	The strategy will be used during trading.
filledTradeInfo()	void	public	This method analyzes the FIX data and fill related fields Trade object.
runTransaction()	void	private	This is a thread method that includes all operations related to trading such as analyzing stock market data, making decision and submitting orders to stock market.

5.4.2.2.2. TradeOrder

This class is responsible for creating FIX message to be sent to the stock exchange according to the decision made after execution of algorithm. Also, it has a method which receives FIX message to be sent, verifies the message. If message verification is successful, it is sent to stock market through methods of FIX class instance.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
orderMessage	String	private	The FIX message that created according to trade information.
createMessage()	String	public	Create FIX message.
evaluateMessage()	void	public	Get a FIX message as a parameter and extract meaning from data.

5.4.2.2.3. MarketData

This class consists of stock market information which is updated frequently. To update itself, a FIX class instance from Communication namespace is created. It gets market data through this FIX class instance by sending appropriate FIX request. Also, all status of responses taken from stock exchange is dumped to database as logs.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
currentValue	Double	private	Current price of the security.
date	Date	private	Current date of the security.
tradeKind	Integer	private	This field represents which stock market trading area will be interested in.

5.4.2.3. GUI PACKAGE

Purpose of this package is to keep all user interface implementations together.

5.4.2.3.1. DisplayRunner

This class can be considered as brain class of the software. It calls methods to draw LoginScreen, MainScreen and to update graphics taking place in MainScreen. Also it has methods to provide user to sign in. Note that main method exists in this class; therefore, the software will be started by this class.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
user	Account	private	The class which holds user information.
dbControllerObj	DBController	private static	This field operates all transaction between DisplayRunner and database.
newTrade()	void	private	This method is used to open new trade dialog when user clicks new trade button.
loadMainScreen()	void	public	After successful login, this method is called to display main screen.
loadLoginScreen()	void	public	After application starts, this method is called to display login screen.
replaceSceneContent()	void	public	This method loads related fxml file to scene content.

5.4.2.3.2. MainController

This class manages all processes which are performed in main screen such as drawing charts, graphs or bringing required information from back-end with creating communication etc.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
openPositionTable	TableView	private	The OpenPosition list that user have in stock market.
lineChart	LineChart	private	This field simulates the flow of the active trades in deliberate amount of time.
dateLabel	Label	private	This field is a non-editable text control. It is used to display date of the stock market.
timeLabel	Label	private	This field is a non-editable text control. It is used to display time of the stock market.
dateTimer	Timeline	private	This method will periodically get stock market date and display it.
chartTimer	Timeline	private	This method will periodically add asset value to lineChart.
tableTimer	Timeline	private	This method will periodically update open position table by updating their price and profit margin.
initialize()	void	public	This method runs in the beginning of the main screen

			and initialize components such as chart, table, date and time.
initializeDate()	void	public	Set a timer for 1 second to update date.
initializeTable()	void	public	Set a timer for 3 second to update table and add already created trades to table.
initializeChart()	void	public	Set a timer for 5 minute to update chart.
addDataChart()	void	public	Add new asset information to lineChart during simulation.
initializeCommunication()	void	public	Create communication between server and client.
showNewTradeDialog()	void	public	When user wants to add new trade, a dialog will appear in the main screen by calling this method.

5.4.2.3.3. LoginController

This class manages all preprocesses such as login and forget my password. These processes is performed in this part thanks to creating connection with back-end implicitly.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
username	String	private	A field which represents the user name.
password	String	private	A field which represents the user password.

initialize()	void	public	A method that initializes default values of required fields.
forgotPassword()	void	private	Enable user to reset password.
processLogin()	void	private	This method makes required authentication process.

5.4.2.3.4. OpenPosition

This class will be used while holding information about trades which are owned from current user to display in the main screen.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
ID	Integer	private	The ID of the open position.
size	Double	private	The surname of the user.
tradeKind	Integer	private	This field represents which stock market trading area will be interested in.
expirationDate	Date	private	This field represents how long trade will be active.
currentValue	Double	private	Current price of the security.
profitRation	Double	private	This field refers to a measure of profitability. It is calculated by finding the net profit as a percentage of the revenue.

5.4.2.4. Database NAMESPACE

Purpose of this namespace is to keep all database implementations together.

5.4.2.4.1. DBController

The task of this class is to connect and transfer data between database and server. This connection is established once when the main screen is opened and it is closed when the application is closed.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
connection	Connection	private	An object that links application to database.
openConnection()	void	public	Establishes a database connection.
closeConnection()	void	public	Finilize database connection.

5.4.2.4.2. FIXDATAREpositoryControllerDB

The responsibility of this class is to record stock market date to database and provide the system to reach all market data during execution trading algorithms.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
addInfo()	bool	public	Add stock market data to no-sql database.
getRelatedData()	void	public	Select desired stock market data from database.

5.4.2.4.3. AccountControllerDB

This class is responsible to record trades which are created by users and hold all information of the user. It verifies the user's password and username is recorded in database. This class provides the system to reach easily all trades information of related user.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
addTrade()	void	public	Add new trade information to current user profile in database.
verifyUser()	void	public	Checks user password and username from database.

5.4.2.4.4. TradeLogControllerDB

The responsibility of this class to save all transactions' status information to database as logs. Note that each log record shall be saved to database even in case of failure.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
saveLog()	void	public	Insert any kind of information like transaction details, errors or warnings to database.

5.4.2.5. Strategy NAMESPACE

Purpose of this namespace is to keep all algorithms implementations together.

5.4.2.5.1. AlgoController

This is an abstract class. There are various algorithms which inherit from this abstract class. According to chosen trade's strategy, it is determined which algorithm should be run in chooseBestAlgorithm method of this class.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
chooseBestAlgorithm()	Integer	public	According to chosen trade criteria, proper algorithms will be selected to make estimation.

5.4.2.6. Communication NAMESPACE

Purpose of this namespace is to keep all communication implementations together.

5.4.2.6.1. SocketCommunication

This class will create a socket connection between client side and server. Then with sending necessary messages, it will receive stock market data and show that to user at simulation. It will also send new trade information when user creates a trade.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
run()	void	public	Establish a connection between server and related port.
sendTradeInfo()	void	public	Send trade information to connected port.
addData()	void	public	Send received data to main controller.

5.4.2.6.2. FIX

This class contains the message to be sent or get from the stock exchange. It has two methods to accomplish to send/receive messages to/from BIST.

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
fixMessage	String	private	Holds a FIX message which is used to sell/ buy order with stock market server.
sendMessage()	void	public	Submit created FIX message to stock exchange system.
receiveMessage()	String	public	Receive FIX message from stock exchange system.

5.4.2.7. CppComponent NAMESPACE

Purpose of this namespace is to keep all socket communications implementations in C++ together.

5.4.2.7.1. CppServer

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
m_pBuffer	Double	private	This field is used as a buffer
m_pBuffer2	Double	private	This field is used as a buffer
m_addrRemote	struct sockaddr_in	private	This field holds connector's address information.
m_addrMe	struct sockaddr_in	private	This field holds this server address information.
connect()	bool	public	Accept a new connection between this object and the port listening on.
close()	bool	public	Shut down the socket.
sendBytes()	bool	public	Send bytes to the listened port.

recvBytes()	bool	public	Receive bytes from the listened port.
sendAck()	bool	public	Send short acknowledge to the server to know they are ready for more
receiveAck()	bool	public	Receive short acknowledge from the server to know they are ready for more

5.4.2.7.2. CppClient

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
m_pBuffer	Double	private	This field is used as a buffer
m_pBuffer2	Double	private	This field is used as a buffer
m_addrRemote	struct sockaddr_in	private	This field holds connector's address information.
connect()	bool	public	Accept a new connection between this object and the port listening on.
close()	bool	public	Shut down the socket.
sendBytes()	bool	public	Send bytes to the listened port.
recvBytes()	Integer	public	Receive bytes from the listened port.
sendAck()	bool	public	Send short acknowledge to the server to know they are ready for more

receiveAck()	bool	public	Receive short acknowledge from the server to know they are ready for more
--------------	------	--------	---

5.4.2.8. JavaComponent PACKAGE

Purpose of this package is to keep all socket communications implementations in java together.

5.4.2.8.1. JavaClient

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
data	byte	private	This field is used to store received data from server
buffer	byte	private	This field is used to temporarily store data
input	BufferedInputStream	private	This field is used to read data from the listened port.
output	BufferedOutputStream	private	This field is used to send data to the listened port.
sendBytes()	bool	public	Send bytes to the listened port.
recvBytes()	Integer	public	Receive bytes from the listened port.
sendAck()	bool	public	Send short acknowledge to the server to know they are ready for more
receiveAck()	bool	public	Receive short acknowledge from the server to know they are ready for more

5.4.2.8.2. JavaServer

<u>Name</u>	<u>Type</u>	<u>Visibility</u>	<u>Definition</u>
data	byte	private	This field is used to store data that will be sent to client
buffer	byte	private	This field is used to temporarily store data
input	BufferedInputStream	private	This field is used to read data from the listened port.
output	BufferedOutputStream	private	This field is used to send data to the listened port.
connect()	void	public	Accept a new connection between this object and the port listening on.
sendBytes()	void	public	Send bytes to the listened port.
recvBytes()	void	public	Receive bytes from the listened port.
sendAck()	void	public	Send short acknowledge to the server to know they are ready for more
receiveAck()	void	public	Receive short acknowledge from the server to know they are ready for more

5.4.3. EXAMPLE UML LANGUAGE

Class diagram of the project can be shown as follows:



Figure 27: CLASS DIAGRAM OF AlgorithmicTrader

5.5. INTERFACE VIEWPOINT

This interface viewpoint provides information designers, programmers, and testers the means to know how to correctly use the services provided by AlgorithmicTrader. This description includes the details of external and internal interfaces not provided in the SRS of this project.

5.5.1 DESIGN CONCERNS

- One of the operation system of Windows, Linux or Mac-OS has to be installed on computer.
- Client side of application is constructed with JavaFx platform so JVM (1.6 or later) is mandatory.
- Data transfer between server and client side is performed using sockets.
- TCP protocol is used while using operating system sockets so TCP drivers must be installed.
- C++ and Java parts have to send their data as bytes to transfer datum without corrupted.
- The application server and stock exchange market server communicate over Internet.
- Using cabled Internet connection which has high bandwidth is recommended for users.
- Because of the FIX protocol usage, host computer has to be protected strictly from attackers.
- Using multiple core computers with fastest processors is recommended for users.
- At least 1 GB memory size must be free to run program properly.

5.5.2. DESIGN ELEMENTS

External Interfaces:

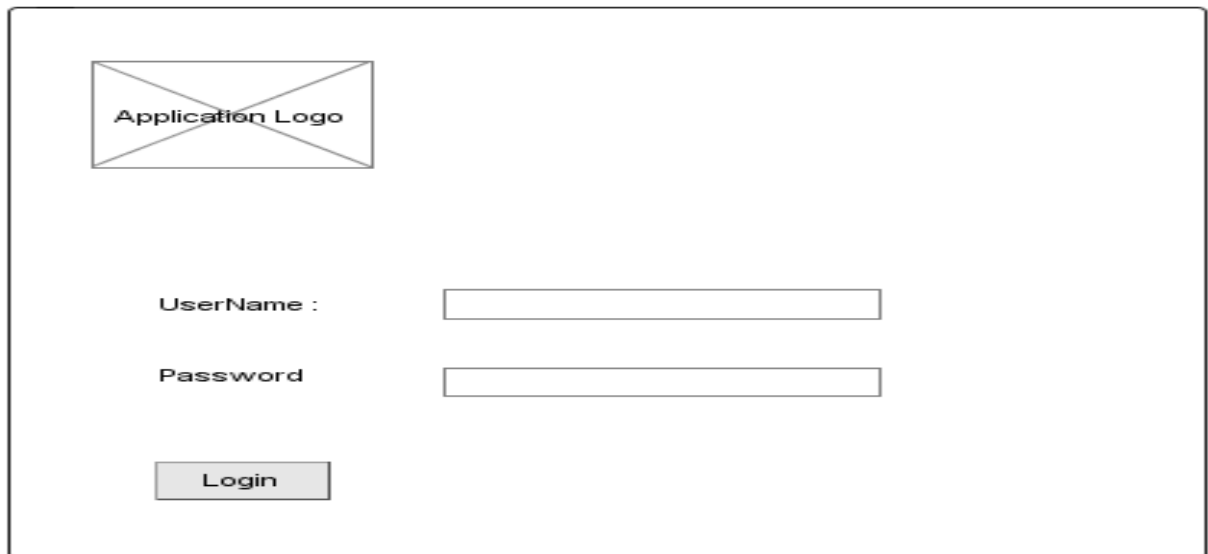
- FIX protocol via application and BIST
- Sockets via JavaFX and C++ parts

Internal Interfaces:

- JVM Run-time platform interface: for running app server which is developed in java

User Interfaces:

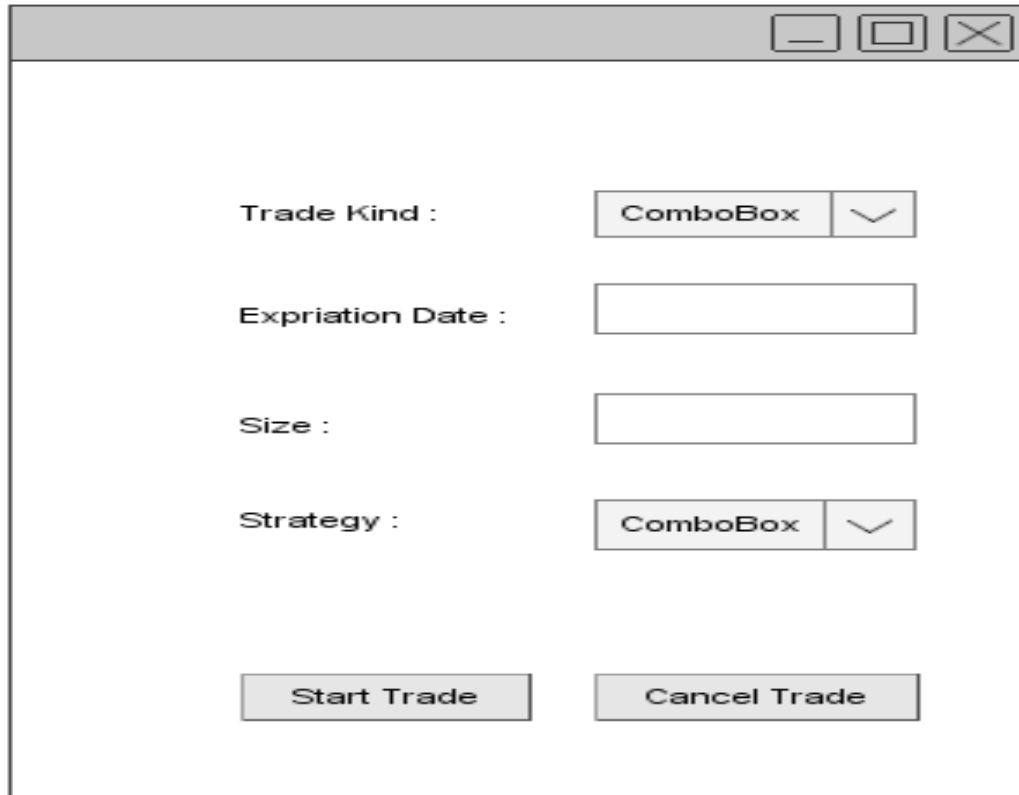
- **Login Screen**



The login screen features a rectangular box containing an application logo placeholder at the top left, labeled "Application Logo". Below the logo, there are two input fields: "UserName :" followed by a text box, and "Password" followed by a text box. At the bottom left of the box is a "Login" button.

Figure 28: Login Screen OF AlgorithmicTrader

- **New Trade Screen (New Trade Dialog Box)**



The New Trade Screen is a dialog box with a standard window title bar (minimize, maximize, close buttons). It contains four input fields: "Trade Kind :" with a "ComboBox" and a dropdown arrow, "Expiration Date :" with a text box, "Size :" with a text box, and "Strategy :" with a "ComboBox" and a dropdown arrow. At the bottom, there are two buttons: "Start Trade" and "Cancel Trade".

Figure 29: New Trade Screen OF AlgorithmicTrader

- **Main Screen**

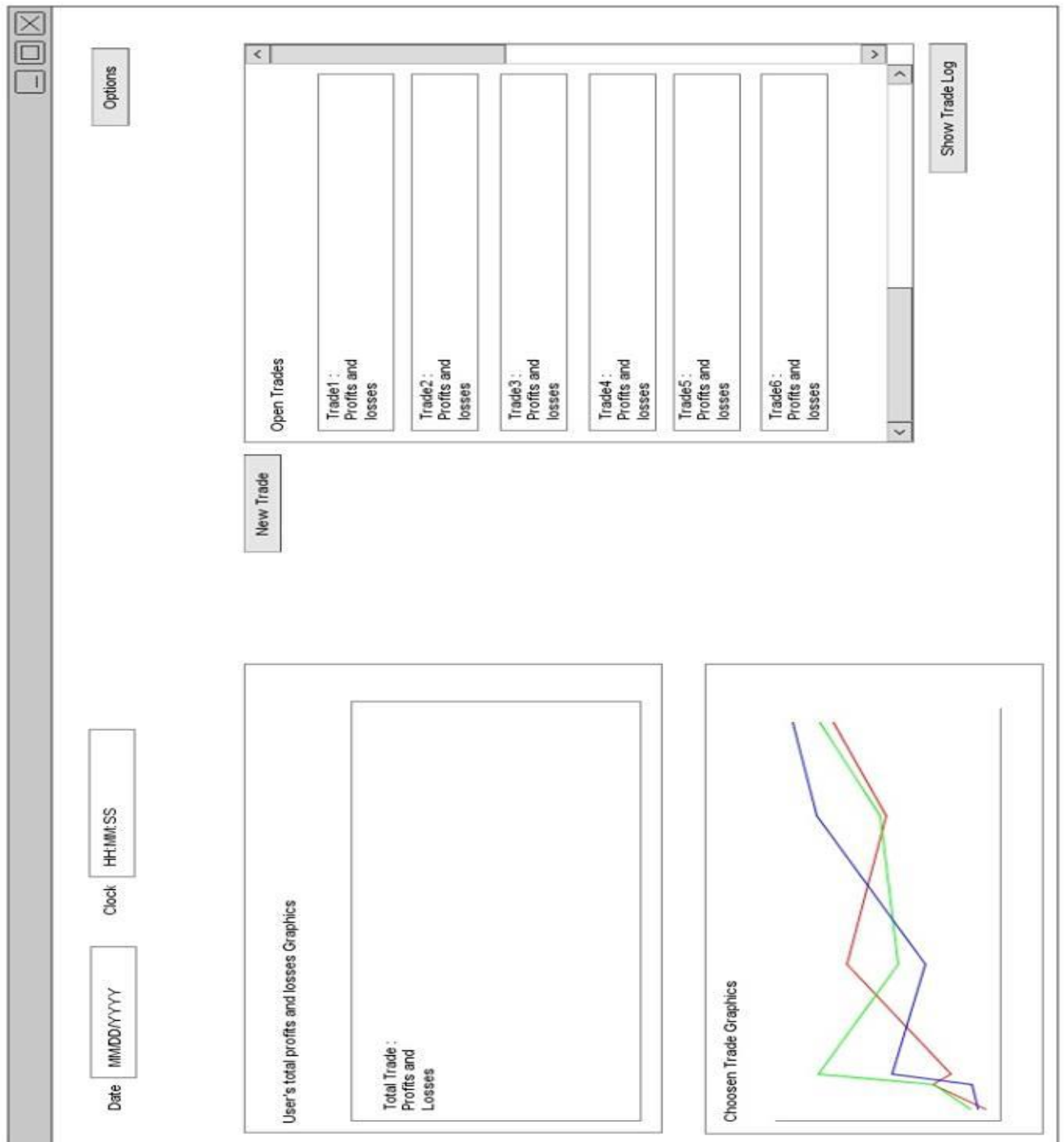


Figure 30: Main Screen OF AlgorithmicTrader

5.5.3. EXAMPLE UML LANGUAGE

Any required diagram was presented in previous section. Therefore, they will not be stated in this part again in order to avoid repetition.

5.6. INTERACTION VIEWPOINT

5.6.1. DESIGN CONCERNS

AlgorithmicTrader is a real-time multi-threaded application. Its working principle is based upon client-server architecture type. All shared methods and variables should be thread-safe to protect from data corruption due to multi-threaded property. This is true for not only communication between application and BIST but also between main C++ parts and JavaFX interface. Communication between application and BIST will be provided via FIX protocol. This part of the application will be implemented by C++. Also, communication between C++ and JavaFX will be carried out via sockets. Both JavaFX interface and C++ parts can send/receive data to/from each other.

Model view controller architecture pattern is used in JavaFX interface of the application. Model section is constituted by all information about any kind of trade. View part is shown to the user via JavaFX screens. And last part that is controller section can be analyzed from Logical Viewpoint part. MainController and LoginController classes are implemented for this purpose. We are trying to implement this architecture in most correct way. Furthermore, we aim to decouple classes as much as possible.

5.6.2. DESIGN ELEMENTS

Note that detailed information about interfaces can be achieved from Interface Viewpoint part of this SDD document in any case of confusion in this part.

5.6.2.1. LOGIN/LOGOUT SEQUENCE DIAGRAM

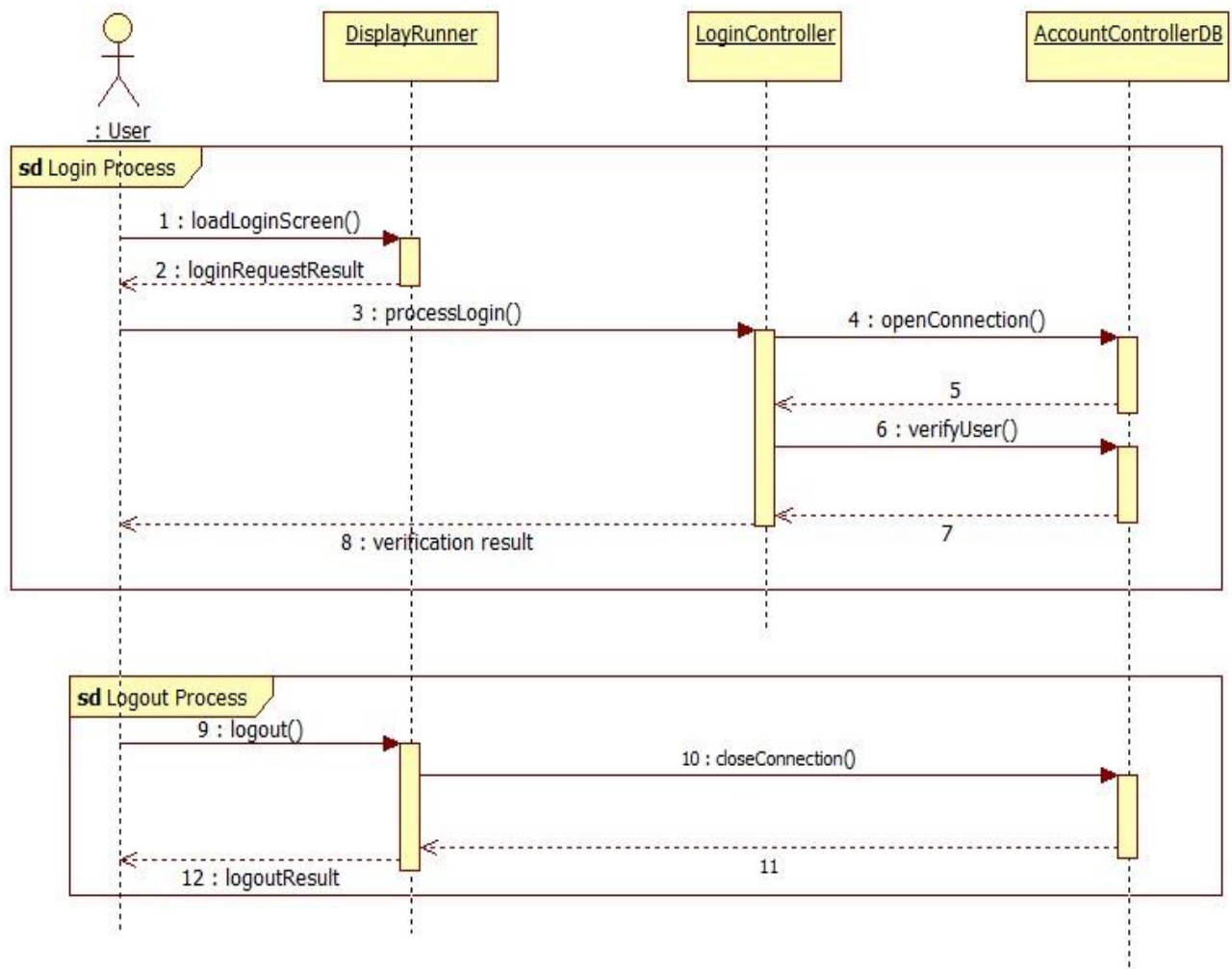


Figure 31: LOGIN/LOGOUT SEQUENCE DIAGRAM

In this login/logout sequence diagram, all actions taken by application are shown above. Firstly, user starts application and he/she sees Login Screen. He/she is expected to fill username and password fields. After the user enters username and password, when he/she clicks login button, these fields are taken and matched with the ones in the database. If there is matching record in database, this means login authentication is successful.

When user wants to log out from system, he/she can click logout button. Therefore, connection that exists for AccountControllerDB is closed. Successful logout operation is carried out. Note that login and logout parts will be referenced in next some sequence diagrams. Therefore, a frame is used in above diagram.

5.6.2.2. SCREEN REPLACING SEQUENCE DIAGRAM

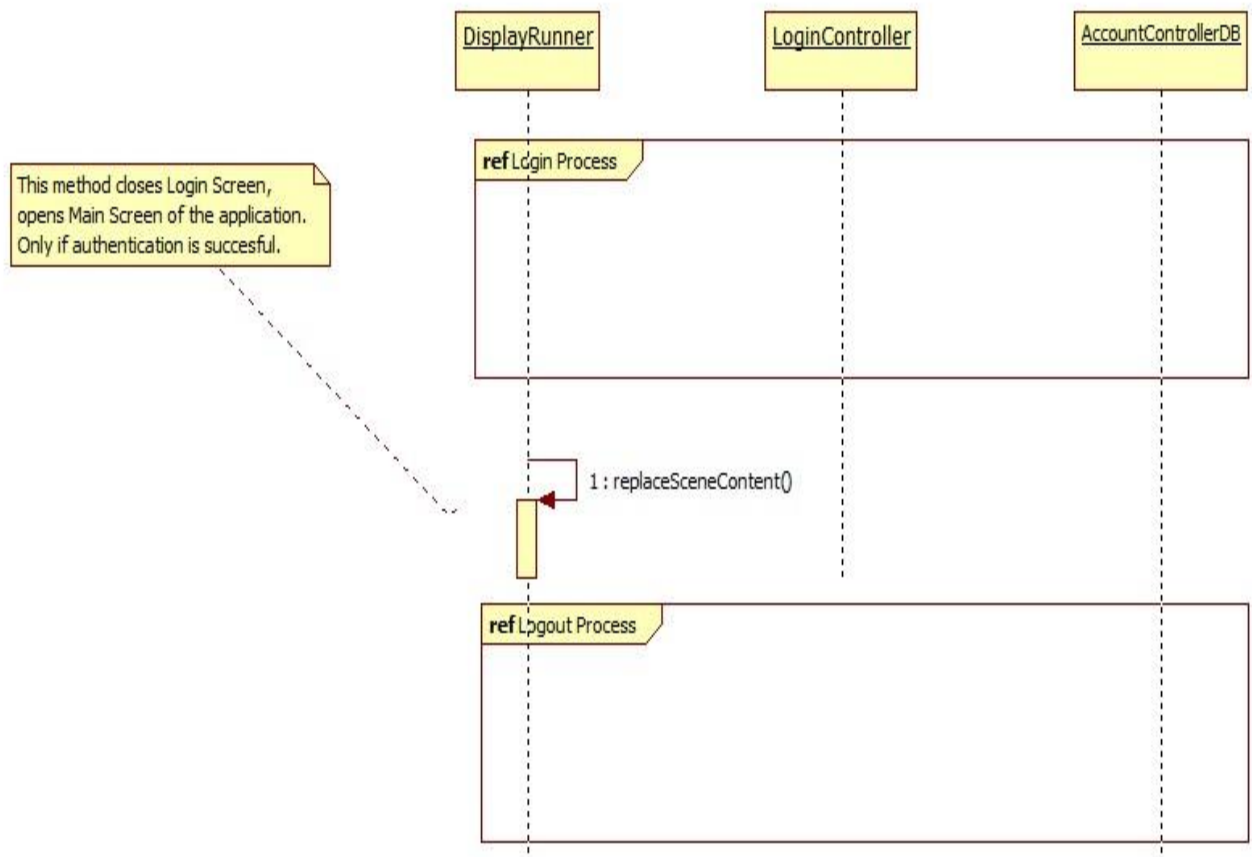
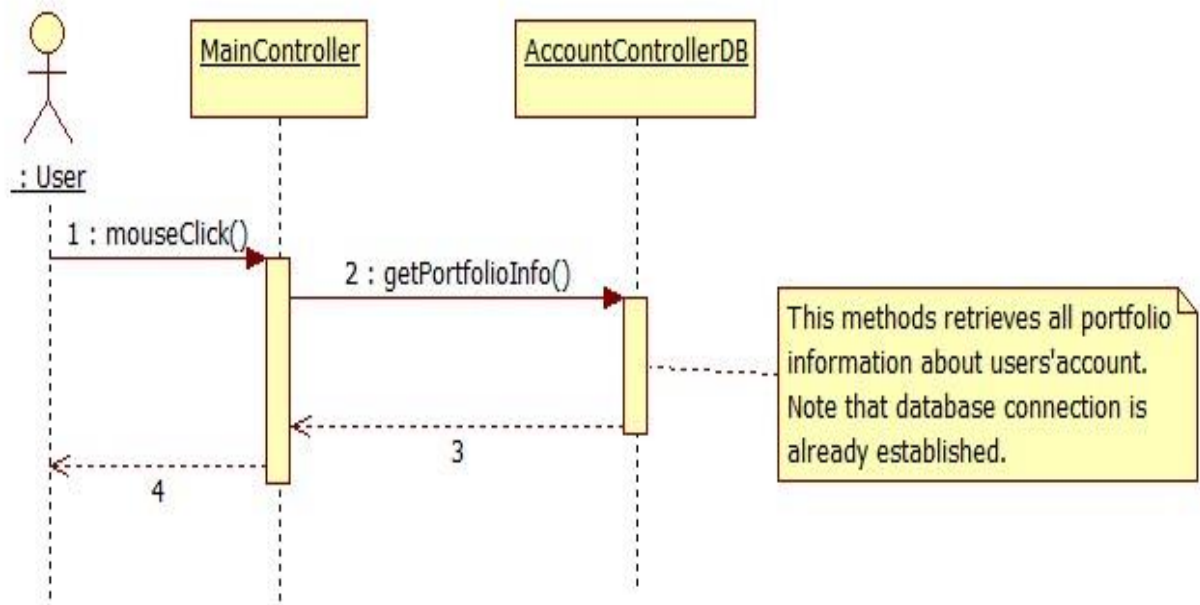
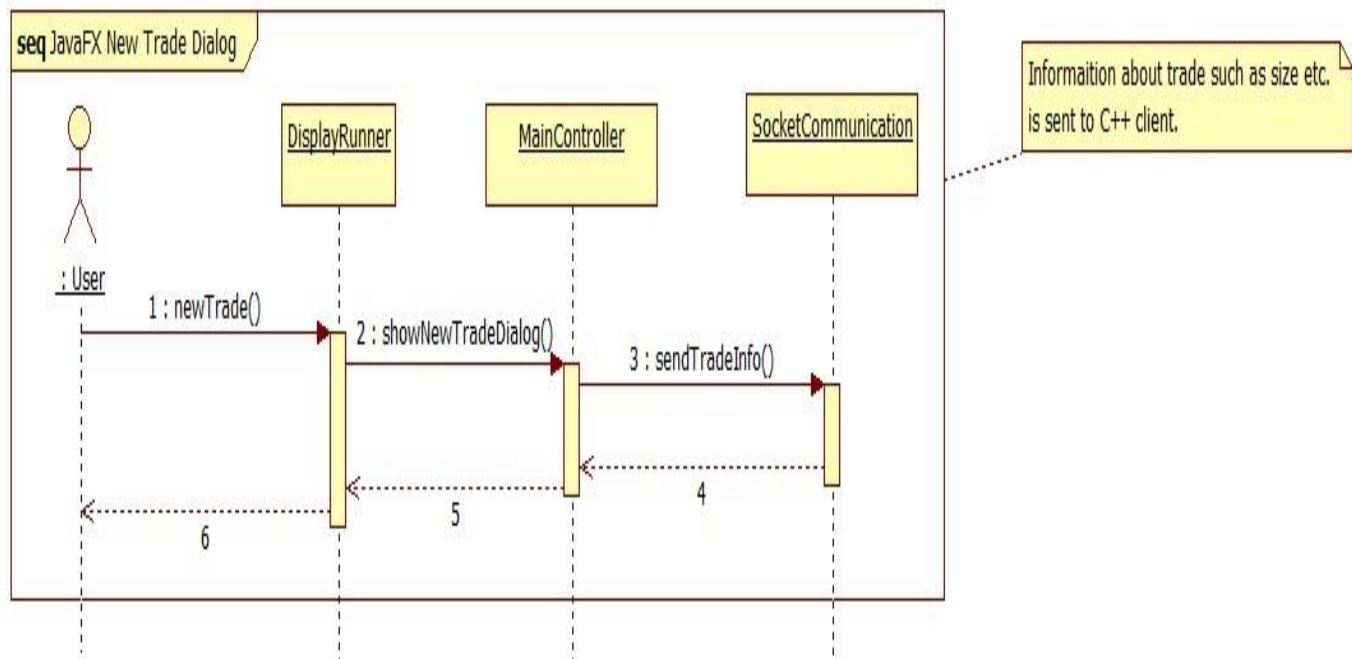


Figure 32: SCREEN REPLACING SEQUENCE DIAGRAM

In above sequence diagram, it is shown when and how will the Login Screen will be replaced by Main Screen. Previous sequence diagram explains login process in detail. When successful login authentication is carried out, Login Screen will be closed and Main Screen will be opened. This will be carried on by calling replaceSceneContent function. This function will be called from DisplayRunner class. Note that Login Process and Logout Process is referenced from previous sequence diagram. If they were drawn again, this would be repetition.

5.6.2.3. RECEIVING PORTFOLIO SEQUENCE DIAGRAM**Figure 33:** RECEIVING PORTFOLIO SEQUENCE DIAGRAM

In this sequence diagram, it shown how a user can display its portfolio. When Main Screen is opened, there will be a button that is used for receiving portfolio from database. When user clicks this button, all portfolio belonging to user will be retrieved from database and it will be displayed on top left part of the main screen. More information about interface can be achieved from Interface Viewpoint part of this SDD document. Note that database connection is already established when Main Screen is opened; therefore, fast response are returned to the user.

5.6.2.4. OPENING NEW TRADE DIALOG SEQUENCE DIAGRAM**Figure 34:** OPENING NEW TRADE DIALOG SEQUENCE DIAGRAM

In this sequence diagram, it is shown how and when New Trade Dialog Box will be shown to the user. In Main Screen, there will be a button that help to display New Trade Screen. When user clicks this button New Trade Dialog Box is shown in front of Main Screen. User is expected to fill the necessary fields in this opened window to be able to start new trade. After filling necessary fields, user clicks start button, and all information received from user, is sent to C++ part from this JavaFX part of the software via sockets. Thus, new trade is started, all other operations are performed on backside of the application and just necessary information is sent to JavaFX interface from C++ part to be able to inform user about what is the current status. Note that New Trade Screen and New Trade Dialog Box is used interchangeably instead of each other in this SDD document.

5.6.2.5. C++ TRADE CREATION AND FIX COMMUNICATION SEQUENCE DIAGRAM

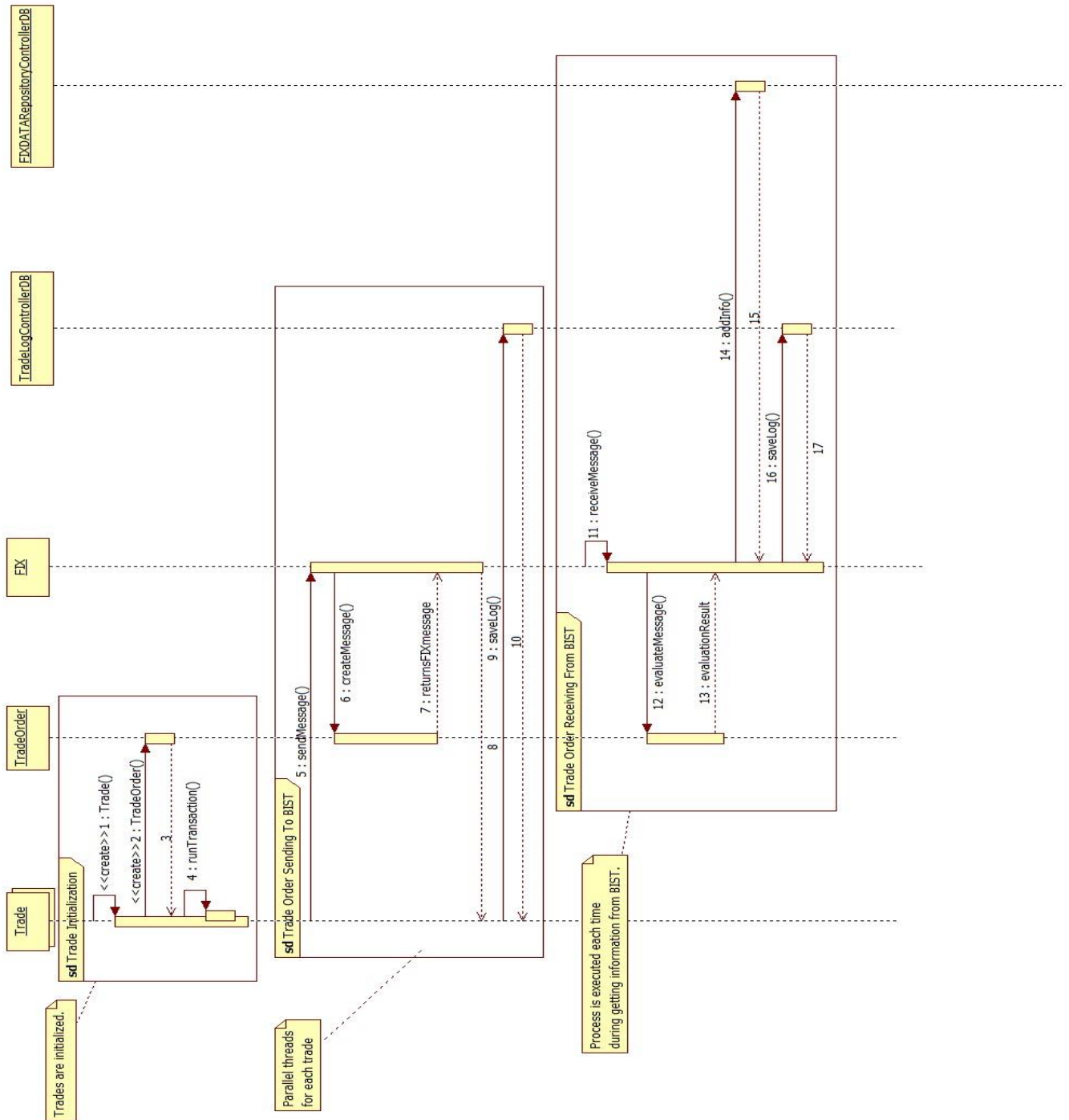


Figure 35: C++ TRADE CREATION AND FIX COMMUNICATION SEQUENCE DIAGRAM

In this sequence diagram, trade initialization is depicted in detail in Trade Initialization part above. In this part firstly, a Trade class object is constructed according to data that is received from JavaFX part via socket. Then a TradeOrder

class object is constructed and this objects belongs to firstly constructed trade class. After these operations, Trade class is started. Note that Trade class runTransaction method is a function pointer that is used to create thread.

In second part Trade Order Sending To BIST part above diagram, send message method is called from FIX class to be able to send FIX message to BIST. Inside of this method, eligible FIX message is constructed from information that is get from algorithms result. This message is sent to BIST. By the way, this action is recorded to database to be able to provide safety and security properties.

In last part Trade Order Receiving From BIST, receiveMessage method is used to catch coming data from BIST. After taking of BIST message, this message needs to be evaluated in order to be used in processes. Note that this coming message has FIX data format. After evaluation of this message, two actions are taken by the application. Firstly, coming information is sent to FIXDATARepositoryControllerDB; that means data is saved to database since algorithms will use them during execution, also they will be shown in user interface. By the way, this receiving data action is also recorded to TradeLogControllerDB database with saveLog method.

Above explained receiving and sending FIX messages processes are repeated many times. Time interval is not determined, this interval can change according to result time of an algorithm.

Note that some parts of above diagram will be referenced in next sequence diagram.

5.6.2.6. ALGORITHM CHOICE SEQUENCE DIAGRAM

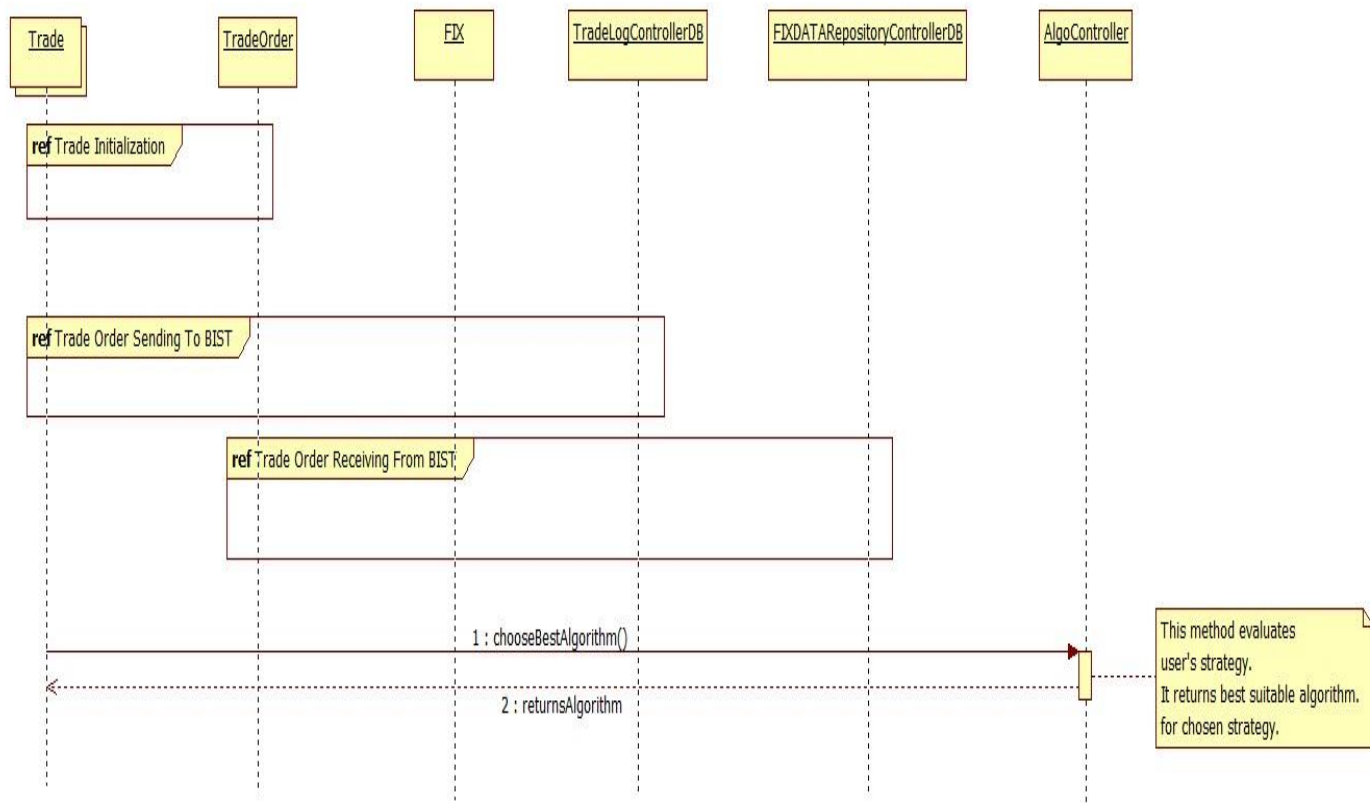


Figure 36: ALGORITHM CHOICE SEQUENCE DIAGRAM

In this sequence diagram, it is explained how will be chosen which algorithm to be executed according to user’s trade strategy. When a trade is created by a user, a trade strategy is expected to be chosen. According to this strategy, a decision is made in chooseBestAlgorithm method of AlgoController class. Upon making this decision, all trade process is shaped by this algorithm. That is all sell/buy orders are returned by this algorithm.

Note that some parts in above sequence diagram is taken by previous sequence diagram by giving reference. It is avoided unnecessary repetition.

5.6.2.7. DISPLAY DETAILED INFORMATION ABOUT A TRADE SEQUENCE DIAGRAM

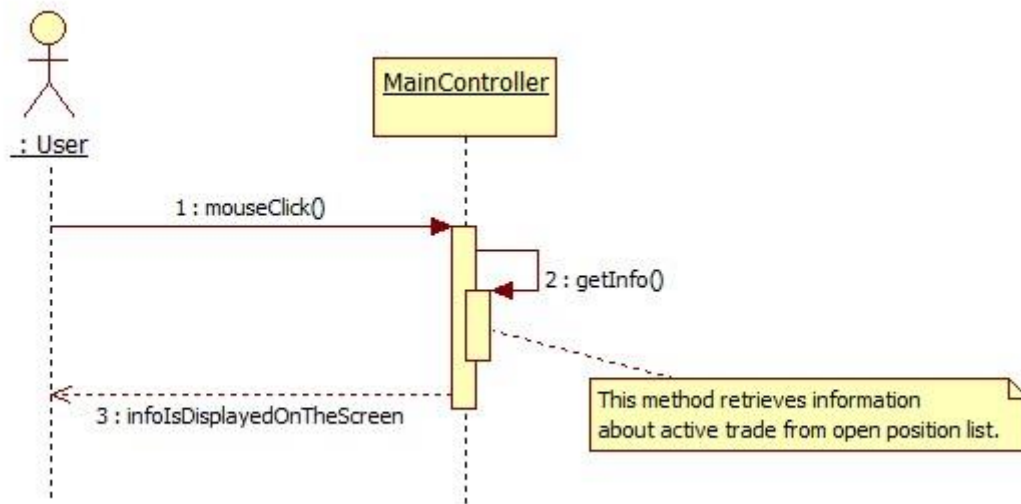


Figure 37: DISPLAY DETAILED INFORMATION ABOUT A TRADE SEQUENCE DIAGRAM

In this sequence diagram, it is depicted how to get detailed information about an active trade. In Main Screen, there is a list of active trades. When clicked one of those, detailed information about that trade is shown on the screen. This is done by a method named `getInfo` in `MainController` class. This method retrieves this detailed information from database. Note that the connection already exists for `FIXDATAREpositoryControllerDB` database in Main Screen.

5.6.3. EXAMPLE UML LANGUAGES

UML sequence diagrams that help to explain this viewpoint are shown above part 5.6.2. DESIGN ELEMENTS. Therefore, they will not be mentioned again here in order to avoid repetition.

5.7. STATE DYNAMICS VIEWPOINT

AlgorithmicTrader is a real-time multi-thread software. Since its real-time property, dynamic properties of the product are tried to be explained in detail as much as possible.

5.7.1. DESIGN CONCERNS

One of the most important issues for AlgorithmicTrader is performance. Therefore, number of states and transitions should be minimized also reaction time to events should be minimum as well. Safety and security principles are significant important for AlgorithmicTrader. More states and transition number means providing safety and security properties that much hard. These properties also encourage minimum state and transition number. Besides, logs that will be recorded to database should not be missed in any mode, state or transition. These are main concerns that should be attached to great importance during design and implementation phases of the project.

5.7.2. DESIGN ELEMENTS

Major events and states can be described as following table.

<u>STATE</u>	<u>EXPLANATION</u>
Login	The user will login to the system by entering username and password.
MainScreen	The system will show MainScreen which includes clock, date, user portfolio, open trades and profit/loss charts to user.
NewTrade	The user will create new trade. This screen is opened in front of Main Screen. Trades works as a parallel way.
Logout	The user will exit from the application.
Make trading decisions	The system will make trading decisions such as sell/buy securities after analyzing the stock market data according to trading strategy(algorithm).
Get market data	The system will get stock market data in order to analyze the securities for making trading decisions. Besides, this data will be sent to user interface as well.

Select securities to buy/sell	After the analysis of market data, the system will select securities that will be sold or bought.
Create buy/sell orders	The system will create trade order to buy or sell selecting securities in a format that can be able to send to BIST.
Send trade order to stock market	The system sends a message to the stock market via FIX protocol according to user's trading strategy in result of algorithm return value.
Analyze securities against trading strategy	The system analyzes the market data based upon trading choices and makes a decision to be followed as next move with the aid of an algorithm that is determined based upon user's trading strategy.
Add open trade	The system will add created trade to open trade list in MainScreen after confirmed order coming from BIST.
Specify trade type	The user will select trade type according to own trade choice in New Trade Screen.
Trade value	The user will specify quantity of securities which will sell or buy in New Trade Screen.
Expiration date	The user will determine deadline of the trade in New Trade Screen.
Strategy	The user will choose a strategy owing to his/her decision in New Trade Screen. This strategy is important since trade algorithm will be determined based upon this choice.
Calculate profit/loss	The system will calculate total profits or losses of user from all trades. Also, they will be shown on user interface.
Show chart	The system will show profit/loss chart of selected trade in Open Trade List in MainScreen.
List open trades	The system will show all trades of user in MainScreen.
Select open trade	The user will select one of his/her own active trades to be able different operations on it such as display details etc.

TABLE 2: EXPLANATIONS ABOUT MAIN STATES OF AlgorithmicTrader

5.7.3. EXAMPLE LANGUAGES

Overall behavior of the system can be depicted as following diagram:

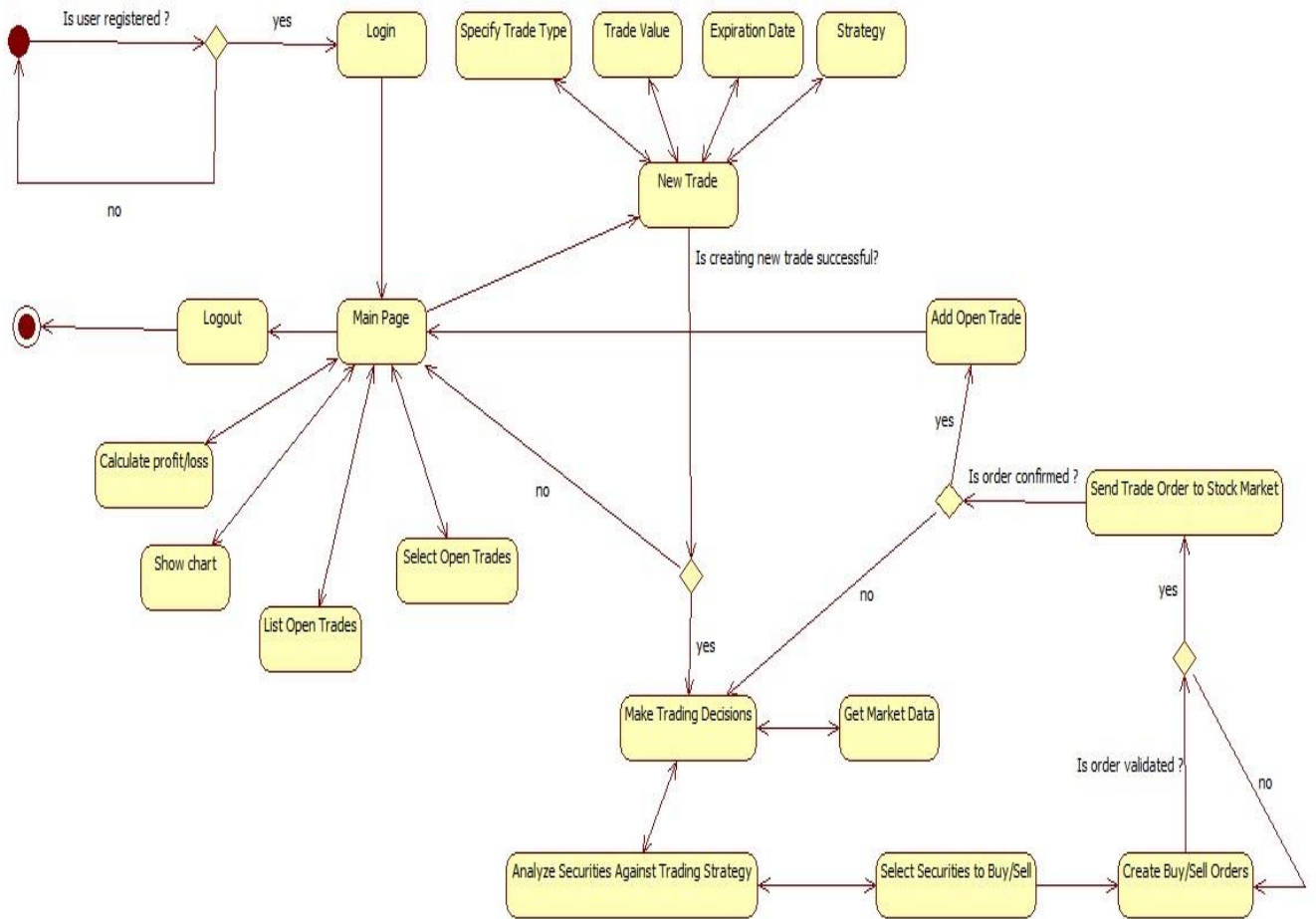


FIGURE 38: STATE DIAGRAM OF THE SYSTEM

5.8. RESOURCE VIEWPOINT

Resources that are used in this project are explained in section 4.6.1. Design Entities. They can be analyzed from that part. They will not be mentioned again because it is avoided from repetition in this SDD document.

6. PLANNING

6.1. TEAM STRUCTURE

As stated in SRS documented of the project, there shall be four developers in the team. All team members have main focus points but they will work on every stage of development, if it is necessary.

There will be two different stages for system design, which are developing prototype and delivering end product. For prototype phase of development, three developers shall work on server side and one developer shall work on client side. Initially the software prototype will be available only as desktop application if this step is achieved successfully web version can be developed. For delivering end product phase, two developers shall work on server side, one developer shall work on client side and three developers shall work on developing required algorithms for software.

6.2. ESTIMATION (BASIC SCHEDULE)

This is the weekly basic schedule of the project starting from the first SRS Document. We are following the schedule regularly; therefore, there will not be any changes than the one existing in SRS document of the project. Detailed week by week project schedule can be shown as follows:

0 - 1 Week : Team management and project selection.

1 - 3 Week : Searching required information from related internet source pages and source books to understand system execution and design.

3 - 5 Week : Finishing search and starting to design of the system for server side.

5 - 6 Week : Finishing system design for client side and starting implementation of server side.

6 - 7 Week : Preparation and writing of software requirements specification.

8 - 11 Week : Finishing both client and server side demos and starting to test the system. Also starting to bug fixing in same time with testing.

11 - 12 Week : Preparation and writing of software design descriptions.

12 - 13 Week : Finishing all current bug fixes.

13 - 14 Week : Presentation of the application and taking feedback and starting to add/remove.

14 - 15 Week : Finishing demos. Also starting to improve C++ server and search proper algorithms.

15 - 19 Week : Finishing algorithm searches and starting to implement them on system.

19 - 22 Week : Finishing implementation and starting to bug fix of analyzed data results from algorithms.

22 - 24 Week : Finishing bugs on algorithms working logic and starting to testing server and client sides.

24 - 25 Week : Beginning to bug fixes of server and client side. Continue to test process of them and fix new bugs.

26 - 27 Week : Fixing all bugs.

28. Week : Delivering the project

6.3. PROCESS MODEL

We are following agile software development methodology. In the milestones we will get feedback about what we did week by week and we will try to stand by our planning as long as possible.

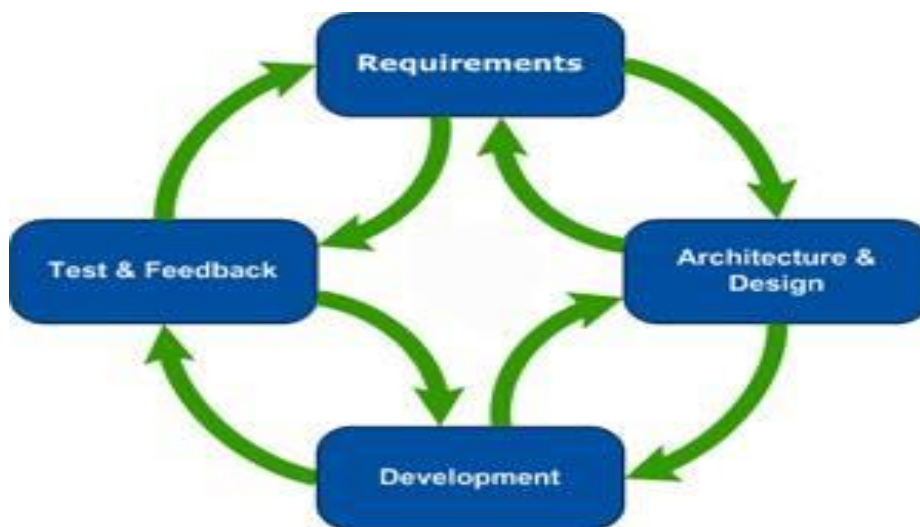


FIGURE 39: AGILE SOFTWARE DEVELOPMENT METHOD REPRESENTATION

7. CONCLUSION

As mentioned SRS document of the project, algorithmic trading system architectures are complicated because of the strict quality requirements of the system. Also, the wide range of regulatory and compliance requirements which manage automated trading makes the design of this software harder. Because of these complexities, careful attention should be paid to the design and

implementation of the system architecture. This software design description document is constructed by overemphasizing these mentioned issues. Implementation details are presented using viewpoints modules and data design by considering above mentioned critical subjects.