Design Document

for

Maester Buy/Sell
The Share Buy/Sell indicator tool

Version 1.2

Prepared by
Jennifer Sunahara - 27590628
Inna Atanasova - 27876947
Charles Boudreau - 27717679
Claudiu Bacisor - 27735332
Amirali Shirkhodaei - 26255906
Jordan Senosiain - 26638538
Krasimir Kanev - 27848056

For ProfitsRUS

April 12th, 2017

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Version 1.0	15/03/2017	Initial version	1.0
Version 1.1	18/03/2017	Corrections after customer feedback on v.1.0	1.1
Version 1.2	11/04/2017	Changes after Beta tested software implementation	1.2

1. Introduction

The Maester Buy/Sell share buy/sell indicator tool is a software program that allows a user to choose a stock from the DOW 30 and get access to advice and charts that help visualize whether or not they wish to buy/sell their shares in this stock.

Our project group will be focused on the implementation of the Maester Buy/Sell share buy/sell indicator tool using the Java programming language as well as storing the user's data. This document follows a template provided by Professor Amin Ranj Bar to specify the system requirements and describe the system design. Based on the course of the Computer Science Department at Concordia University.

While this documentation mainly refers to the envisioned, finished product, where relevant, it will also mention the status of the version of the software that has been developed at the time of writing.

1.1 Purpose

The product whose software requirements are specified in this document goes by the name of Maester Buy/Sell. It is intended as a software application to help the customers of ProfitsRUS make a decision on buying/selling shares of a stock from the DOW 30. The complete tool will contain a user friendly graphical user interface, user account system, user data reporting system and the creation of charts with information and advice on buying/selling shares, based on simple user selections.

Let it be noted that the version 1.0 of Maester Buy/Sell implements only the calculation of a simple moving average, based on a subset of user criteria that will be implemented in the full version.

1.2 Document Conventions

This document uses the IEEE standard Times New Roman size 12 fonts [1]. Bolding will be used for emphasis on important terms. Every requirement statement will have its own priority.

1.3 Intended Audience and Reading Suggestions

This document's intended audience is any potential user with an interest in buying/selling stocks, and is designed to be user-friendly, regardless of technical knowledge. Readers are urged to check the attached glossary for any unfamiliar terms.

It is organized into 6 sections: 1- Introduction, 2 - Overall Description, 3 - System Features, 4 - External Interface Requirements, 5 - Other Nonfunctional Requirements, 6 - Other Requirements.

Developers may want to read sections 1.1 to 1.5, 2.1 to 2.6, 3.1 to 3.7, 4.1 to 4.4, and 5.1 to 5.4.
Project Managers may want to read sections 1.1 to 1.5, , 2.1 to 2.8, 3.1 to 3.7, and 5.1 to 5.4.
Marketing staff will be interested in sections 1.1 to 1.5, 2.1, 2.2 and 2.6.
Everyday Users will want to read sections 1.1 to 1.5, 2.1, 2.2, 2.4, 2.6, 4.1, 4.4, and 5.2 to 5.4.
Testers will want to read sections 1.1 to 1.5, 2.1 to 2.4, 2.6, 2.7, 4.1 to 4.4 and 5.1 to 5.4.
Documentation writers may be interested in sections 1.1 to 1.5, 2.1 to 2.6, 3.1 to 3.7, 4.1 to 4.4, and 5.1 to 5.4.

1.4 Project Scope

The Maester Buy/Sell share buy/sell indicator tool is a software program that is intended to be useful to clients of ProfitsRUS that are looking for advice and information on buying/selling stocks. It is also intended to track user search information for ProfitsRUS. The aforementioned company wants specifically desires a tool that is user-friendly to clients who have no pre-existing technical knowledge.

This SRS will contain documentation of the features of the finished product, such as create account, update account, choose stock, choose moving average range, choose historical data, update and request log, which have yet to be developed in this deliverable. Let it be noted that currently, the moving average calculator component allows the calculation of the moving averages, based on a subset of user criteria, which will be used to create the charts.

The deliverable products are:

The Maester Buy/Sell, version 1.0 (Moving Average calculator component): implemented in Java.

The Maester Buy/Sell Software Requirements Specification Document

1.5 References

[1]
Document style requirements for IEEE 802.11: http://www.ieee802.org/11/Rules/format-rules.html

[2]
Investopedia: Technical Analysis: http://www.investopedia.com/terms/t/technicalanalysis.asp

[3]
Web Service Glossary: https://www.w3.org/TR/ws-gloss/

[4]
Investopedia: Technical Analysis Tutorial*:* http://i.investopedia.com/inv/pdf/tutorials/technicalanalysis.pdf

2. Overall Description

2.1 Product Perspective

The **Maester Buy/Sell**, a technical analysis software under development for ProfitsRUS is a new, self-contained product designed to meet the customer's need for a simple tool to help guide them in investment decisions related to the DOW 30 stocks. The main function of this software is the generation of charts and a "buy" or "sell" recommendation based on moving averages. User choices of Stock, Moving Average range and Historical Data range are recorded for later analysis. The product also allows users to have an account and allows accounts with necessary privileges to receive a report on recorded application usage data.

2.2 Product Features

Account:
1. Create Account
2. Change Password

Chart Generation:
1. Choose Stock
2. Choose Moving Average Range
3. Choose Historical Data Range
4. Update Chart

Logging:
1. Generate Report
2. Select Start Date
3. Select End Date

2.3 User Classes and Characteristics

The users of the software can be split into two broad categories: **End Users** and **Administrators**. **End Users** are generally the customers the software is targeted to, and thus the favored class. These users may use the software for its intended purpose of receiving investment advice and the technical knowledge of such users can vary greatly. End Users may not use the "Request Log" option. **Administrators** are users who work under ProfitsRUS. In addition to access all the features that End Users have access to, Administrators may also have access to the Log Request function, which can be used to inform further development of the software.

2.4 Operating Environment

The software is a standalone application, not a web application. It will access the internet only to receive statistical data. It's also platform independent and runs on every platform where a Java Virtual Machine is available.

2.5 Design and Implementation Constraints

Maester Buy/Sell is platform independent and written in Java application. User's information and extracted stock values are saved in a database.

2.6 User Documentation

The application is designed with user-friendliness in mind. The lightweight and simplistic GUI attempts to hide application and network complexities. No training is required in regards to learning the software. A simple user manual will be included with the software in the second deliverable. This manual will include product overview, a list of commands in alphabetical order and contact information.

2.7 Assumptions and Dependencies

- The service is used on a desktop, laptop or tablet. A mobile application might be implemented later;
- Since the application will connect to Yahoo Finance to retrieve data, an internet connection with sufficient speed is required;
- A working Java Runtime Environment version 1.6 or above is necessary;
- Only US and Canadian stock markets are considered;
- Included libraries: MySQL Connector library, JFoenix library
- Supported language: English;
- Supported currency: USD;
- Any contract negotiation and legal concerns are ignored.

2.8 Budget

The cost estimation methods used are COCOMO and the bottom up cost estimation models. The COCOMO cost estimation model was considered a prime candidate due to it's repeatable and analyzable formula. This will play a crucial role throughout the project as a low cost and efficient model for adjusting costs as the project moves forward.Given our current estimates using a bottom up approach, the sum of the modules put together will sum to approximately 2400 lines of code.

The following is a breakdown of the evaluated development time and lines of code for each module:

Components	**Estimated lines of code**	**Development Time**
Data Collector	250	1.5 Man-Months
Chart Generator	500	2.5 Man-Months
User Accounts	150	3 Man-Weeks
User Logs (Reports)	100	2 Man-Weeks
Graphical User Interface	600	3 Man-Months
Communications Services (Database connection and communications)	200	3 Man-Weeks
MVC Architecture	600	3 Man-Months
Total	2400	12 Man-Months

Given a team of 7 programmers, the development time is more accurately 1.7 months.

Breakdown of employees and cost over time:

Workers	**Days of work** (7 h/day)	**$/hour**	**Cost**
Architect	7	50	2,450
Project Manager	51	50	17,850
Technical Lead	51	55	19,635
UI Designer	45	25	7,875
IT Admin	51	20	7,140
Engineer x2	14	40	7,840
Programmer x7	47	30	69,090
Total	--	--	131,880$

The COCOMO cost estimation was done using COCOMO II for its additional parameters.
Given an input of 2400 lines of code and using the Organic project type input, the estimated man-months is 12.012. The qualifying parameters of the project are the following:

Product Attributes

Required Software Reliability: High

Size of Application database: Normal

Complexity of the Product: Normal

Hardware Attributes: Normal

Memory Constraints: Normal

Volatility of the virtual machine environment: Low

Required Turnaround Time: Normal

Personal Attributes

Analyst Capabilities: Low

Applications Experience: Low

Software Engineer capability: Normal

Virtual Machine Experience: Normal

Programming Language Experience: Low

Project Attributes

Application of Software Engineering Methods: Normal

Use of Software Tools: Normal

Required development Schedule: High

Given the average salary of 2,586/day, the cost for the whole project within 51 days will be approximately 131,886$.

3. System Features

3.1 Project Features

The **Maester Buy/Sell** technical analysis software under development for ProfitsRUS is a new, self-contained product designed to meet the customer's need for a simple tool for analyzing and visualizing stock market data for facilitating investment decisions.
The main function of this software is the generation of charts and a "buy" or "sell" recommendation based on moving averages. User choices of Stock, Moving Average range and Historical Data range are recorded for later analysis. The product also allows users to have an account and allows accounts with necessary privileges to receive a report on recorded application usage data.

3.1.2. List of Actors

1. User as a primary actor;
2. Admistrator as a primary actor;
3. Yahoo Finance as a secondary actor;
4. User Repository as a secondary actor;

3.1.3. Actors Description

1. User: The main users of the application, the clients;
2. Administrator: An user with elevated System access privileges;
3. Yahoo Finance: A database from which our product will acquire the Stock Market Data requested by the User;
4. User Repository: A database to store User information;

3.1.4. Project Domain Model

Overview:

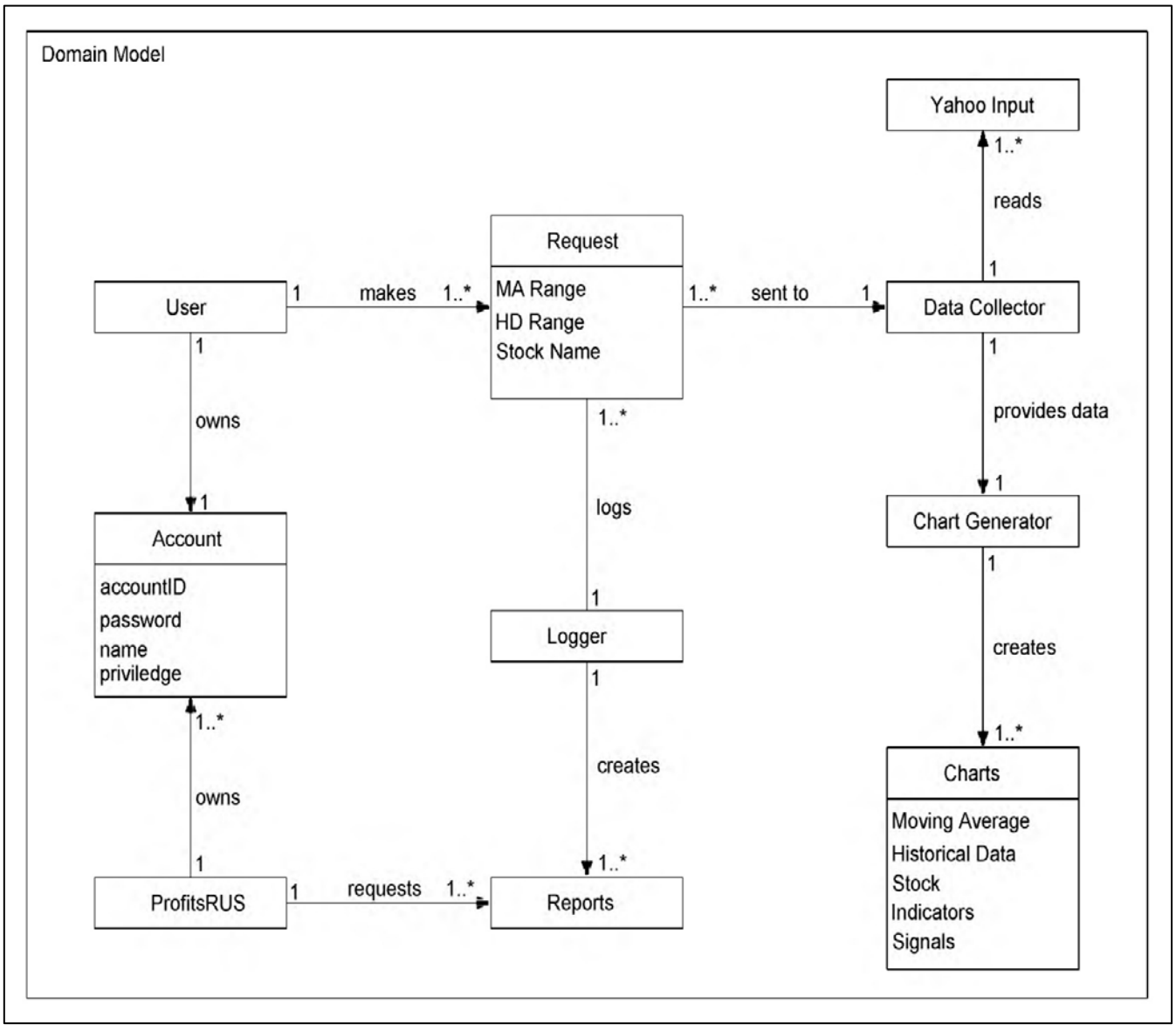
The domain model is a representation of meaningful real-world concepts or objects relevant to the domain that need to be modeled in software. It is a way to describe and model real world entities and the relationship between them. It can be used to solve problems related to that domain.

Domain Model Concepts Table:

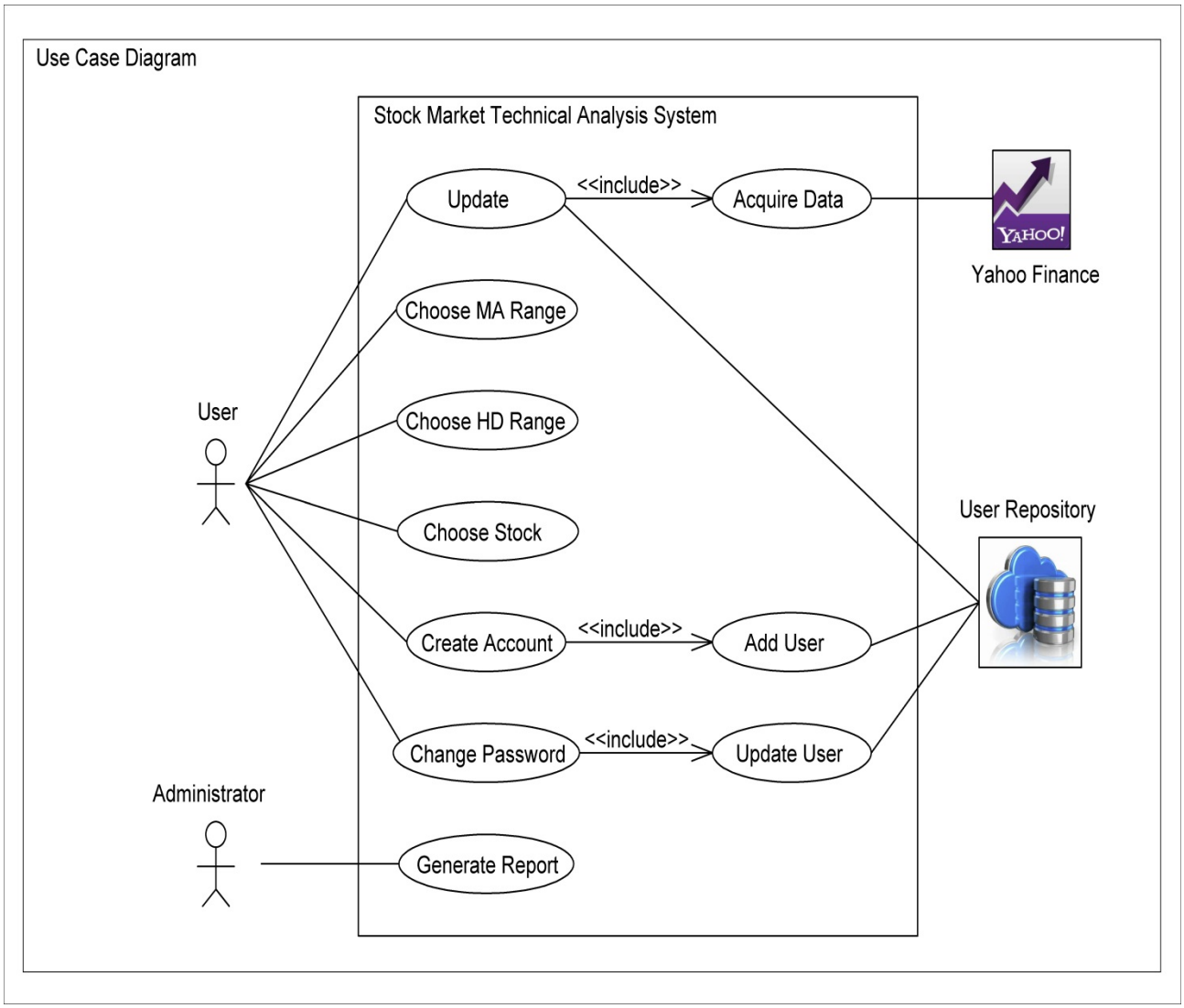
Concept Name	Description
ProfitsRUS	1. The company ProfitsRUS owns the accounts of the Users that are dealing with the Stock Market Data. The User could be a customer or/and employee of ProfitsRUS 2. ProfitsRUS could request Reports of the recorded application usage data
Account	1. Every user Account has an unique AccountID 2. The Account distinguishes Users by Name, Password and Privileges.
User	1. Every user owns an Account 2. Every user can make a Request for Stock Market Data, choosing a combination of 'MA Range'; 'HD Range'*; 'Stock Name'
Request	1. A Request for Stock Market Data is composed of the combination of user selections of the following options: 'MA Range'; 'HD Range'*; 'Stock Name' 2. Request schema for Log records is formed by ProfitsRUS. - Request contains application usage data based on that schema. - Request is submitted to the Logger
Yahoo Input	1. Main source of Stock Market Data 2. Will be accessed in real time on user demand
Data Collector	1. Receives and processes requests for Stock Market Data 2. Communicate with the Stock Market Data source 3. After receiving the data from Stock Market Data source, based on the user Request, feeds the necessary part of raw Data to the Chart Generator
Chart Generator	1. Creates Charts based on the Data provided by the Data Collector
Charts	1. Creates charts for Moving Averages, Historical Data and Stock. 2. Display Indicators and signals to facilitate "buy/sell" decisions.
Logger	1. Records application usage data. 2.Creates Reports on demand about application usage data
Reports	1. Contains data regarding application usage from Users. 2. Can only be requested by employees of ProfitRUS with certain authorization / privileges.

* 'MA Range' stands for 'Moving Average Range'
'HD Range' stands for ' Historical Data range'

Domain Model Diagram:



3.1.5. Use Case Diagram



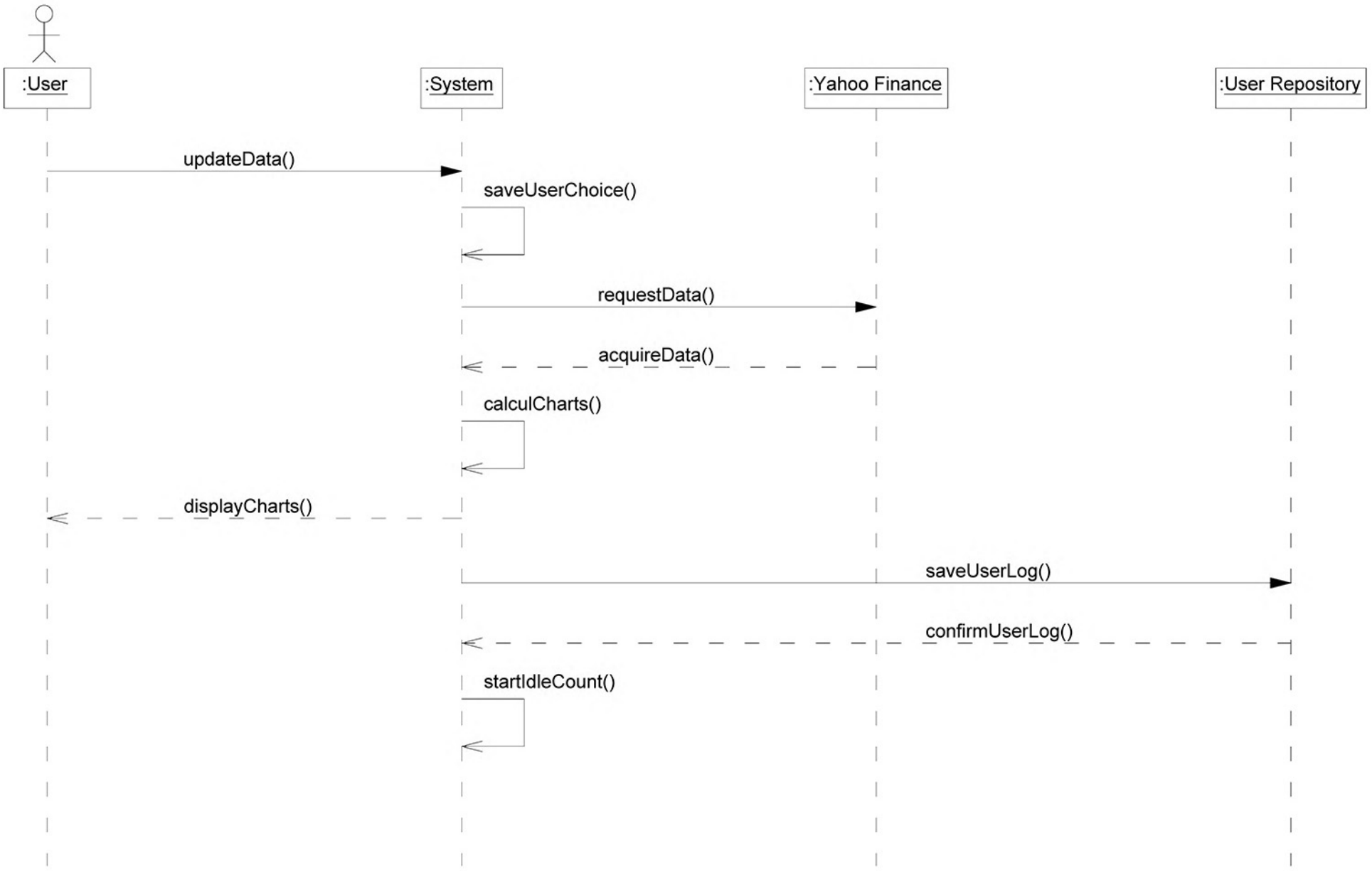
3.1.6. Use Cases

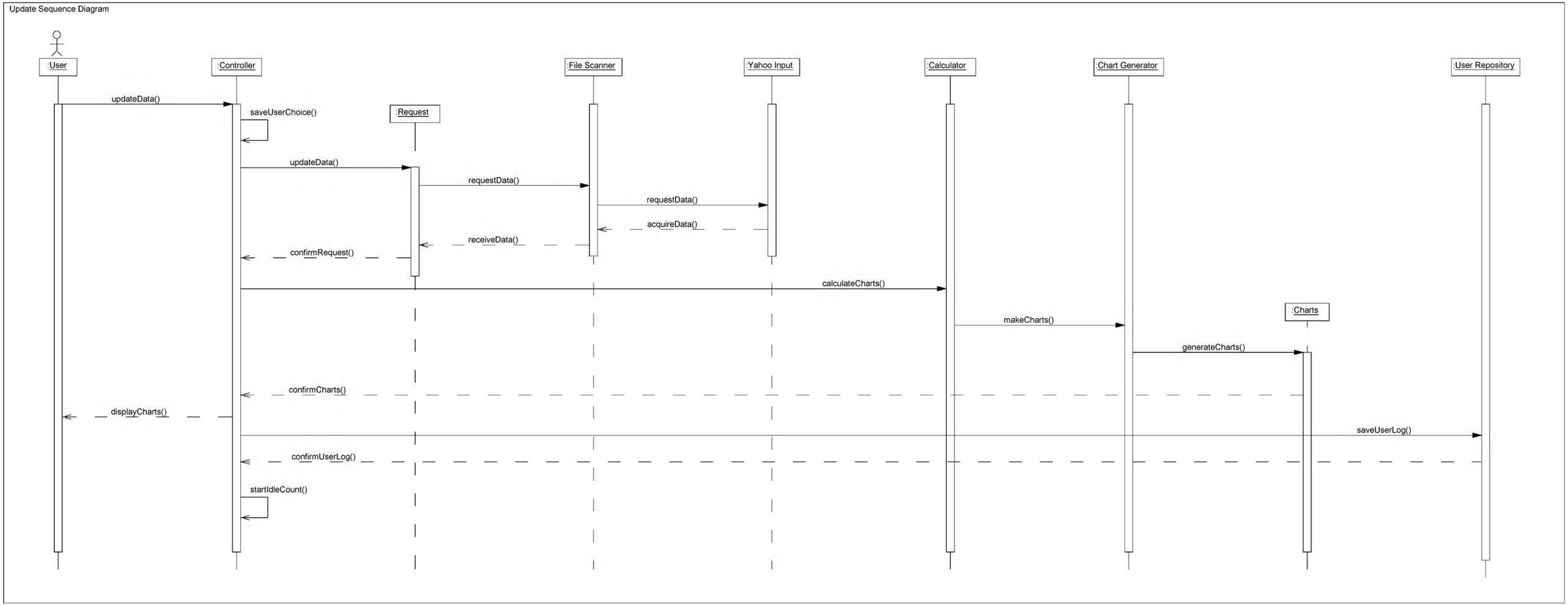
Use Case # 001 - Update Data
Use Case # 002 - Choose Moving Average Range
Use Case # 003 - Choose Historical Data range
Use Case # 004 - Choose Stock
Use Case # 005 - Create Account
Use Case # 006 - Change Password
Use Case # 007 - Request Report

3.2 Use Case 001 – Update Data

Number	001	
Name	Update Data	
Summary	Updates Stocks Data, Graphics and the User Log File in User Repository	
Priority	5	
Preconditions	User is logged to the system	
Postconditions	Updated Stocks Data, Graphics and the User Log File in User Repository	
Primary Actor(s)	User	
Secondary Actor(s)	Yahoo Finance, User Repository	
Trigger	User selects "Submit"	
Main Scenario	**Step**	**Action**
	1	User selects "Submit"
	2	System saves User's choice for "STOCK", "MA range", "HD range"
	3	System sends a request to read data from Yahoo Finance
	4	System acquires data from Yahoo Finance
	5	Based on the User choice, System uses acquired data to calculate charts
	6	System display updated charts
	7	System updates the User Log File in User Repository
Extensions	**Step**	**Branching Action**
	3a	System request to read data from Yahoo Finance is not granted
	3a-2	After unsuccessful attempt System display an error message: "Unable to acquire data from Yahoo Finance! Please try again later or ask Administrator for assistance"
	3a-3	System goes to Step# 08
Open Issues	0	

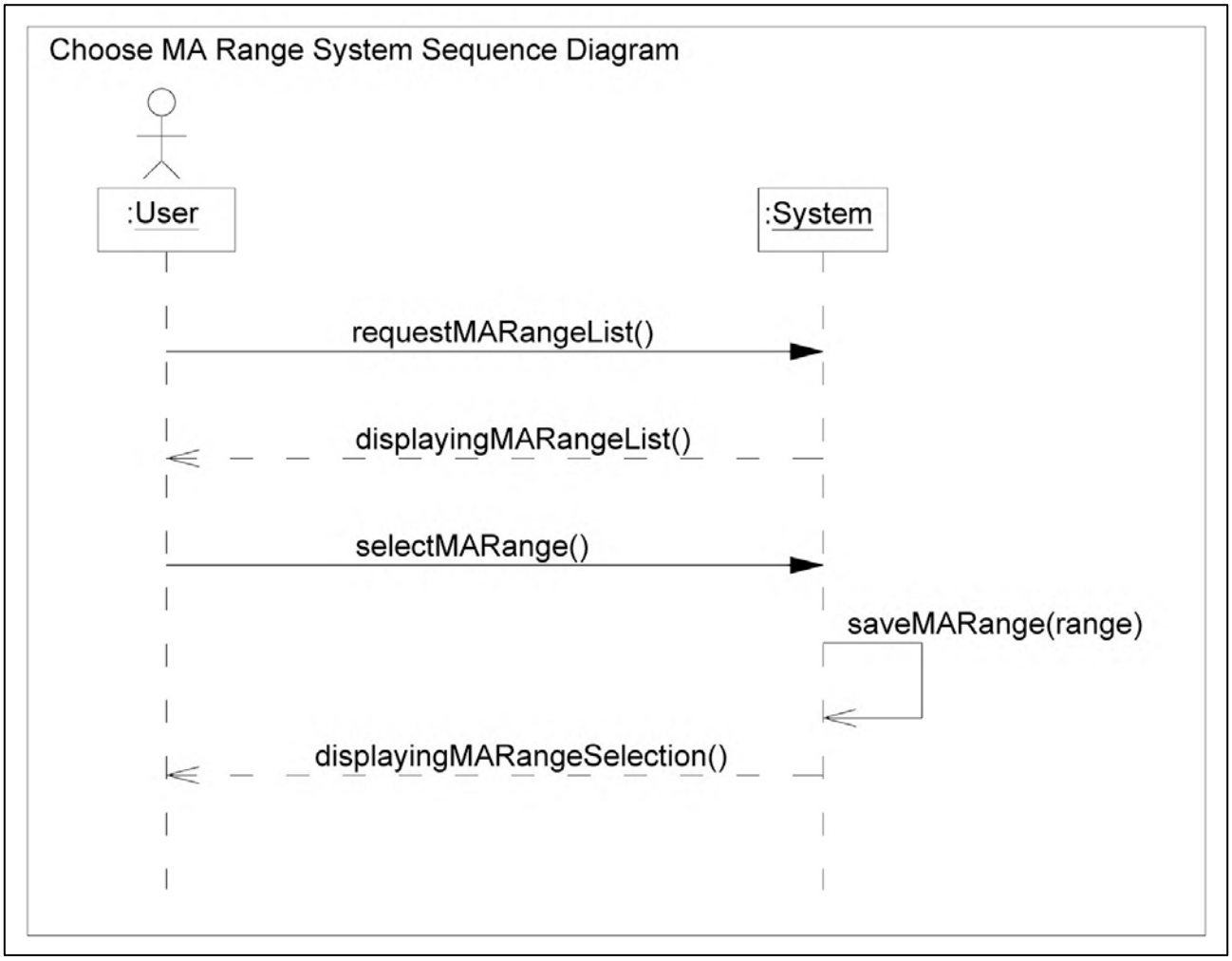
Update System Sequence Diagram

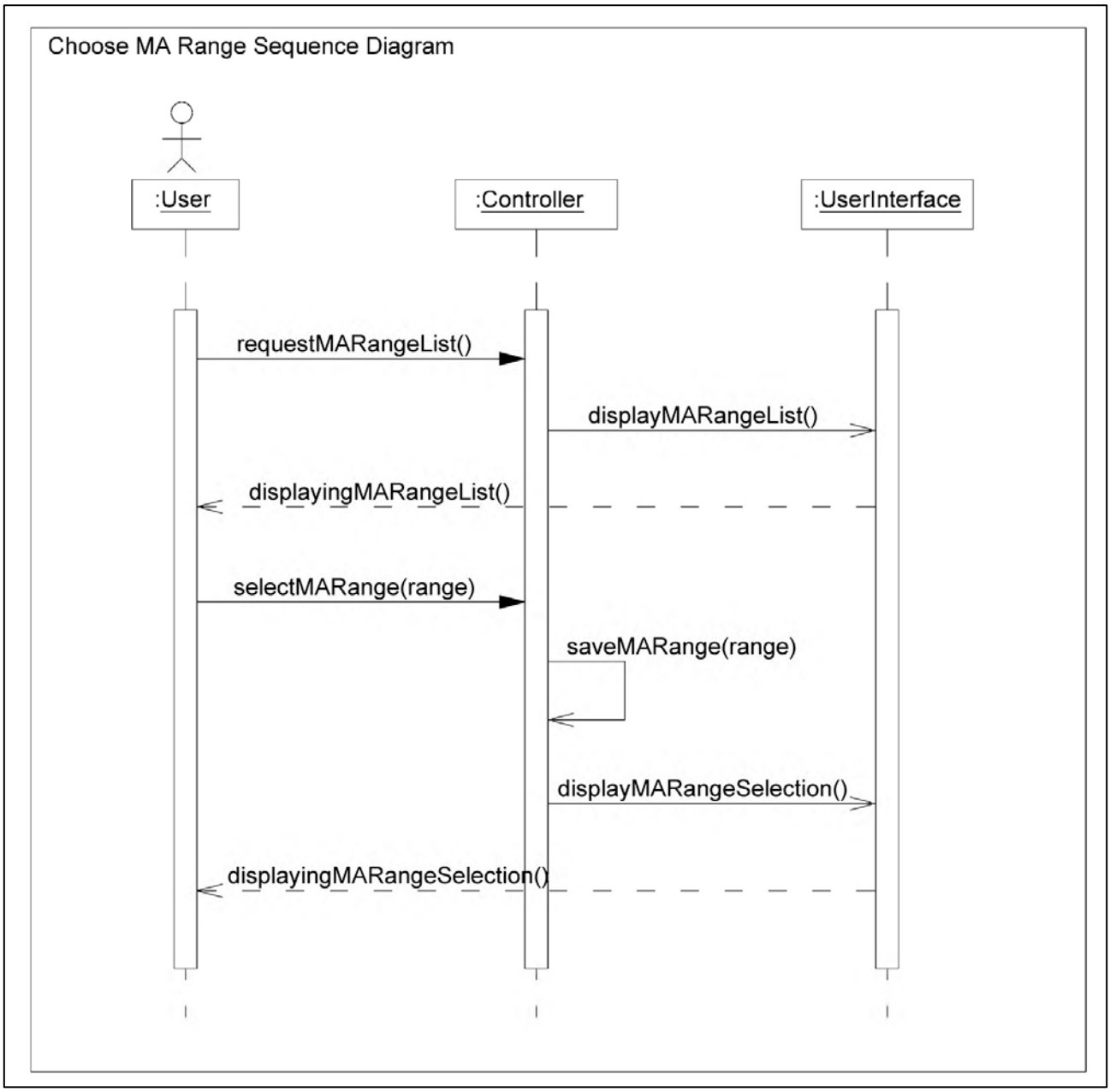



Update Sequence Diagram
:User
:Controller
:Request
:File Scanner
:Yahoo Input
:Calculator
:Chart Generator
:Charts
:User Repository
updateData()
saveUserChoice()
updateData()
requestData()
requestData()
acquireData()
receiveData()
confirmRequest()
calculateCharts()
makeCharts()
generateCharts()
confirmCharts()
displayCharts()
saveUserLog()
confirmUserLog()
startIdleCount()

3.3 Use Case 002 – Choose Moving Average Range

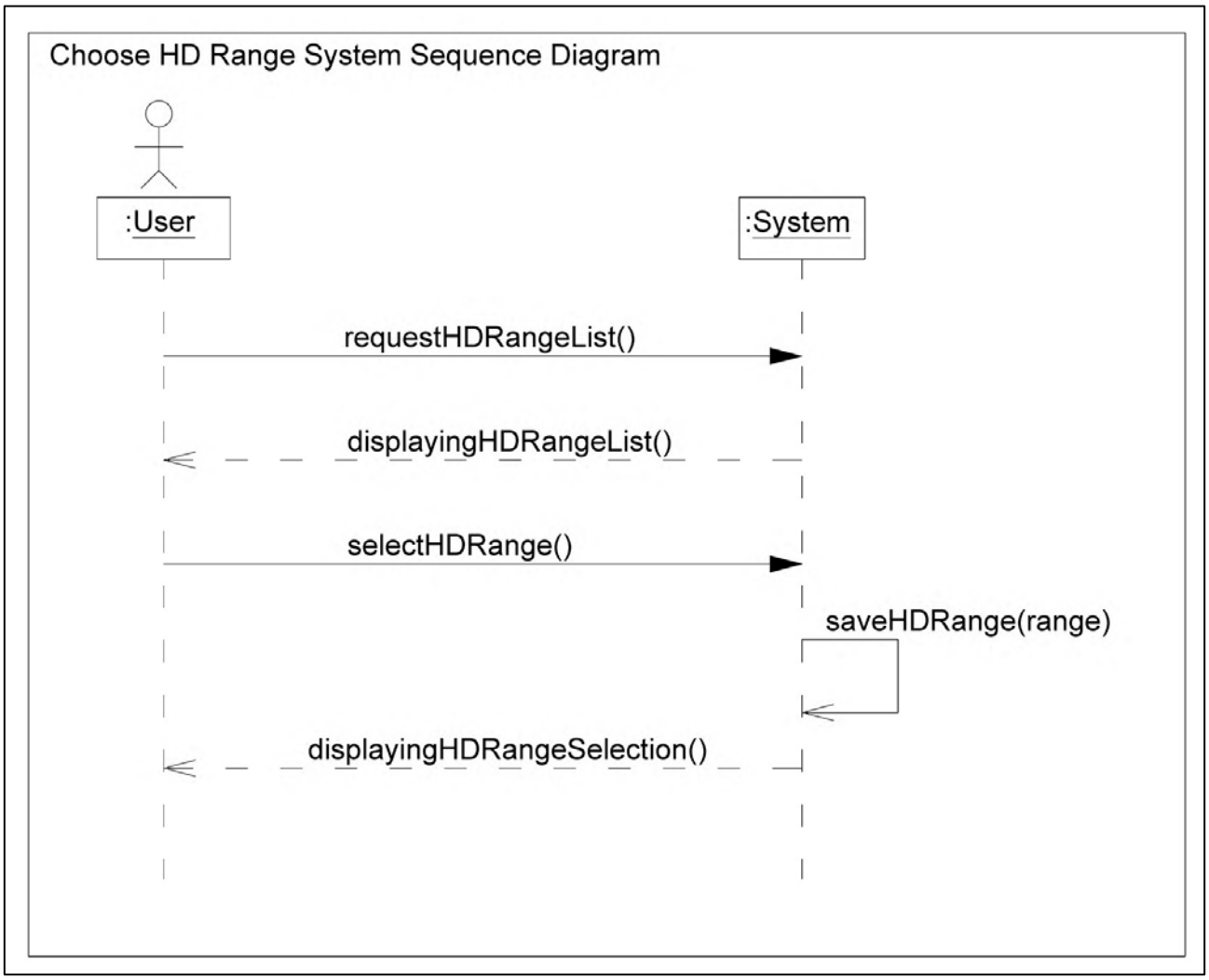
Number	002	
Name	Choose Moving Average Range	
Summary	User selects the moving average from a predefined range of 20, 50, 100, 200 days	
Priority	3	
Pre-Condition	User has logged in	
Post-Condition	Moving Average Range has been selected	
Primary Actor(s)	User	
Secondary Actor(s)	System	
Trigger	User has selected the “Moving Average Range” option	
Main Scenario	**Step**	**Action**
	1	User selects Moving Average Range option.
	2	System displays list of Moving Average Range options (20, 50, 100, 200 days).
	3	User selects range .
	4	System registers selection.
Extensions	**Step**	**Branching Action**
	3a	User moves on without selecting a moving average range option.
	3a-1	System cancels the operation.
Open Issues	0	

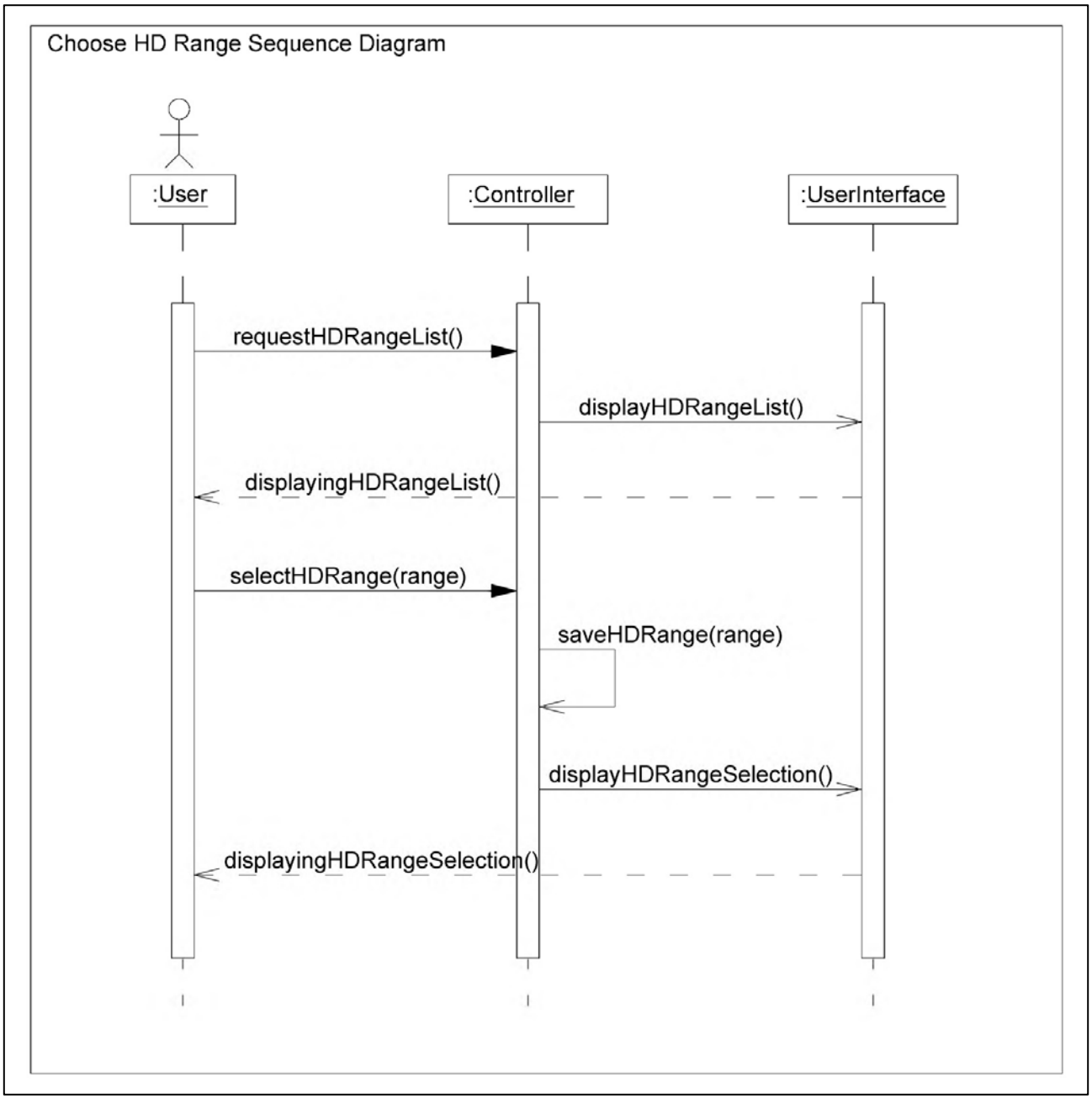

Choose MA Range System Sequence Diagram
:User
:System
requestMARangeList()
displayingMARangeList()
selectMARange()
saveMARange(range)
displayingMARangeSelection()


Choose MA Range Sequence Diagram
:User
:Controller
:UserInterface
requestMARangeList()
displayMARangeList()
displayingMARangeList()
selectMARange(range)
saveMARange(range)
displayMARangeSelection()
displayingMARangeSelection()

3.4 Use Case 003 – Choose Historical Data Range

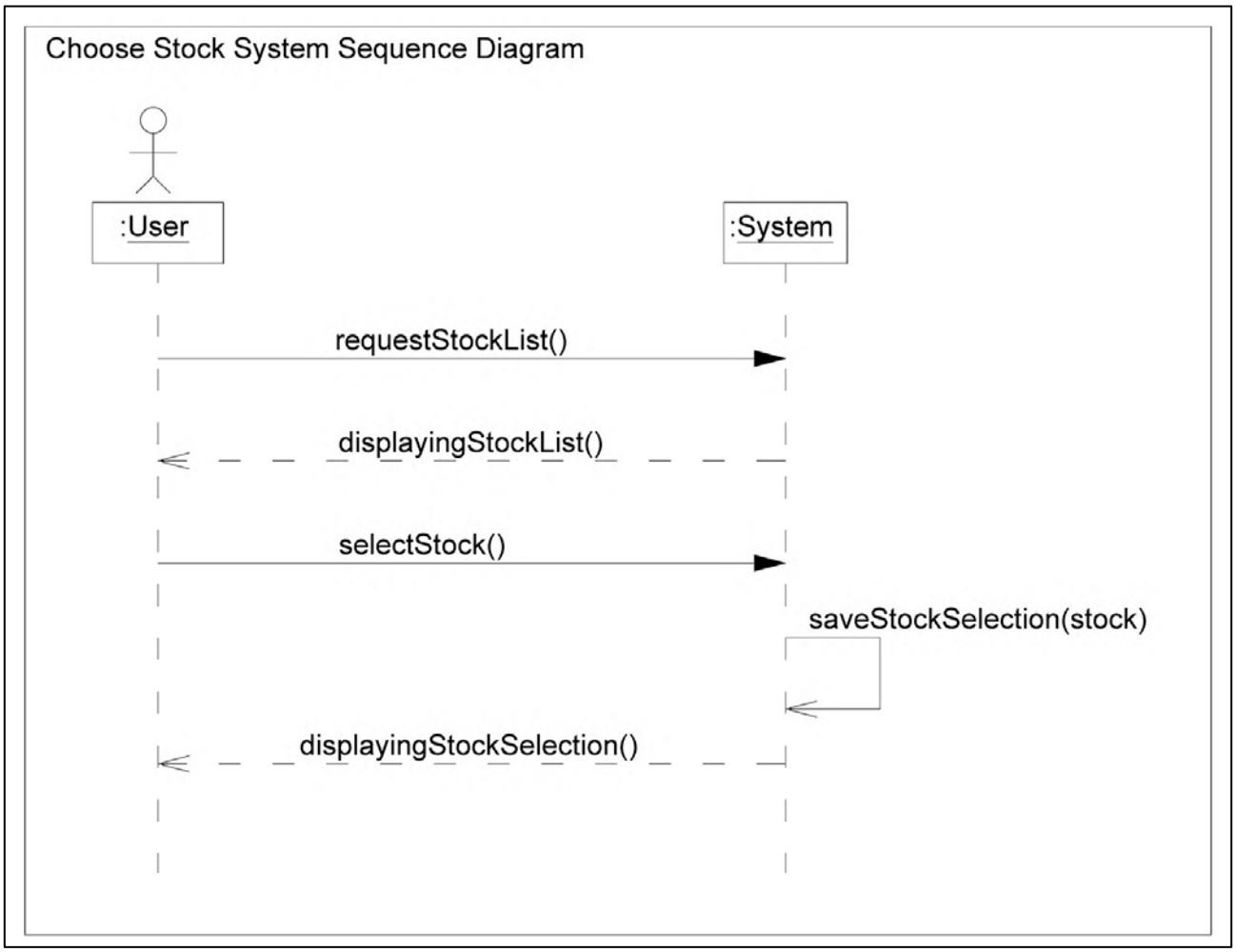
Number	003	
Name	Choose Historical Data Range	
Summary	User makes a change to the selected historical data range	
Priority	1	
Precondition	User has logged into their account.	
Postcondition	The selected historical data range has been changed.	
Primary Actor(s)	User	
Secondary Actor(s)	None	
Trigger	User has selected the change historical data range option.	
Main Scenario	**Step**	**Action**
	1	User selects change historical data range option.
	2	System display a list of historical data range options (all data, last 5 years, last 2 years, last year).
	3	User selects on their choice list.
	4	System registers the selection.
Extensions	**Step**	**Branching Action**
	3a	User moves on without selecting a historical data range option.
	3a-1	System cancels the operation.
Open Issues	0	

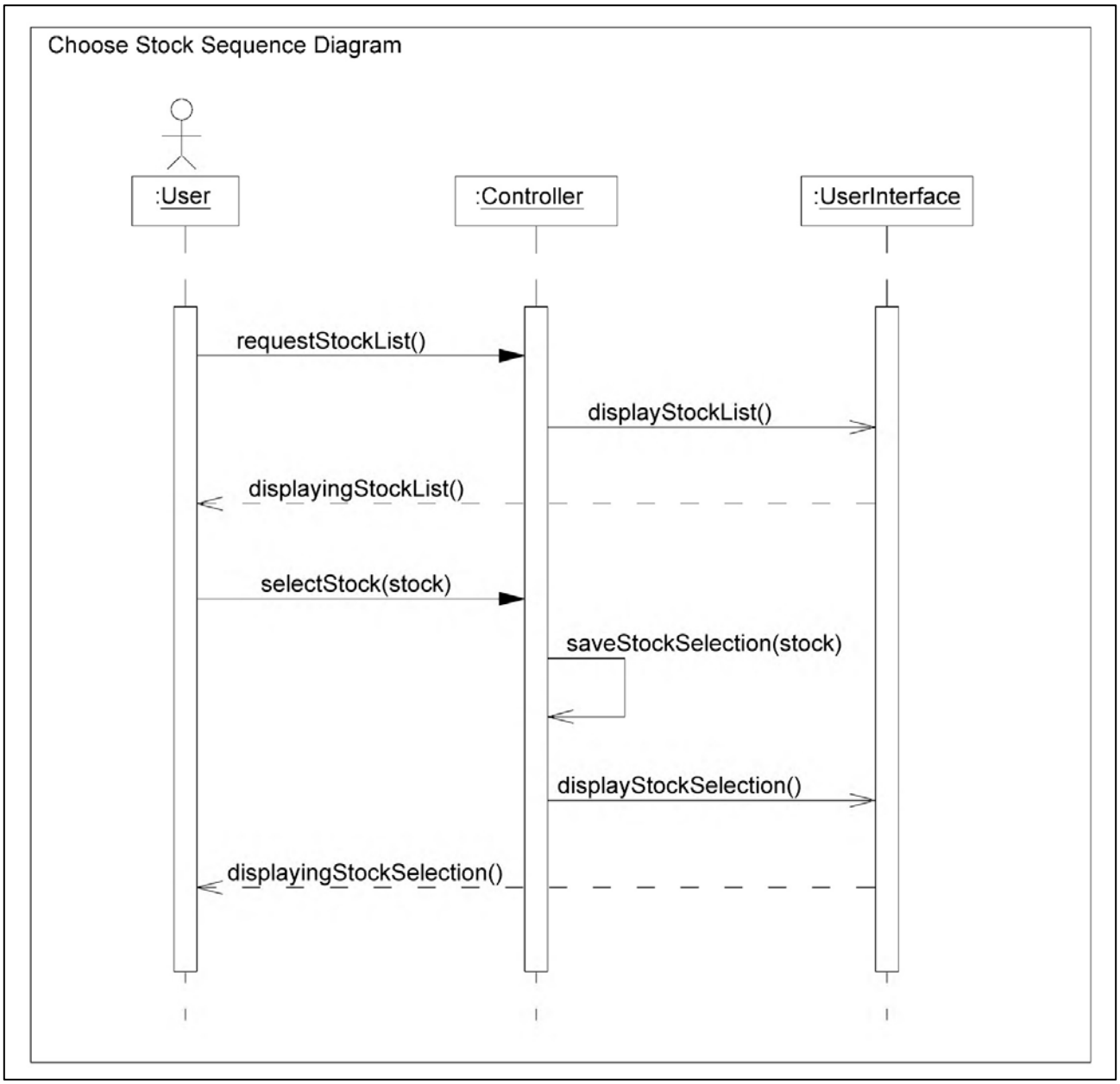

Choose HD Range System Sequence Diagram
:User
:System
requestHDRangeList()
displayingHDRangeList()
selectHDRange()
saveHDRange(range)
displayingHDRangeSelection()


Choose HD Range Sequence Diagram
:User
:Controller
:UserInterface
requestHDRangeList()
displayHDRangeList()
displayingHDRangeList()
selectHDRange(range)
saveHDRange(range)
displayHDRangeSelection()
displayingHDRangeSelection()

3.5 Use Case 004 – Choose Stock

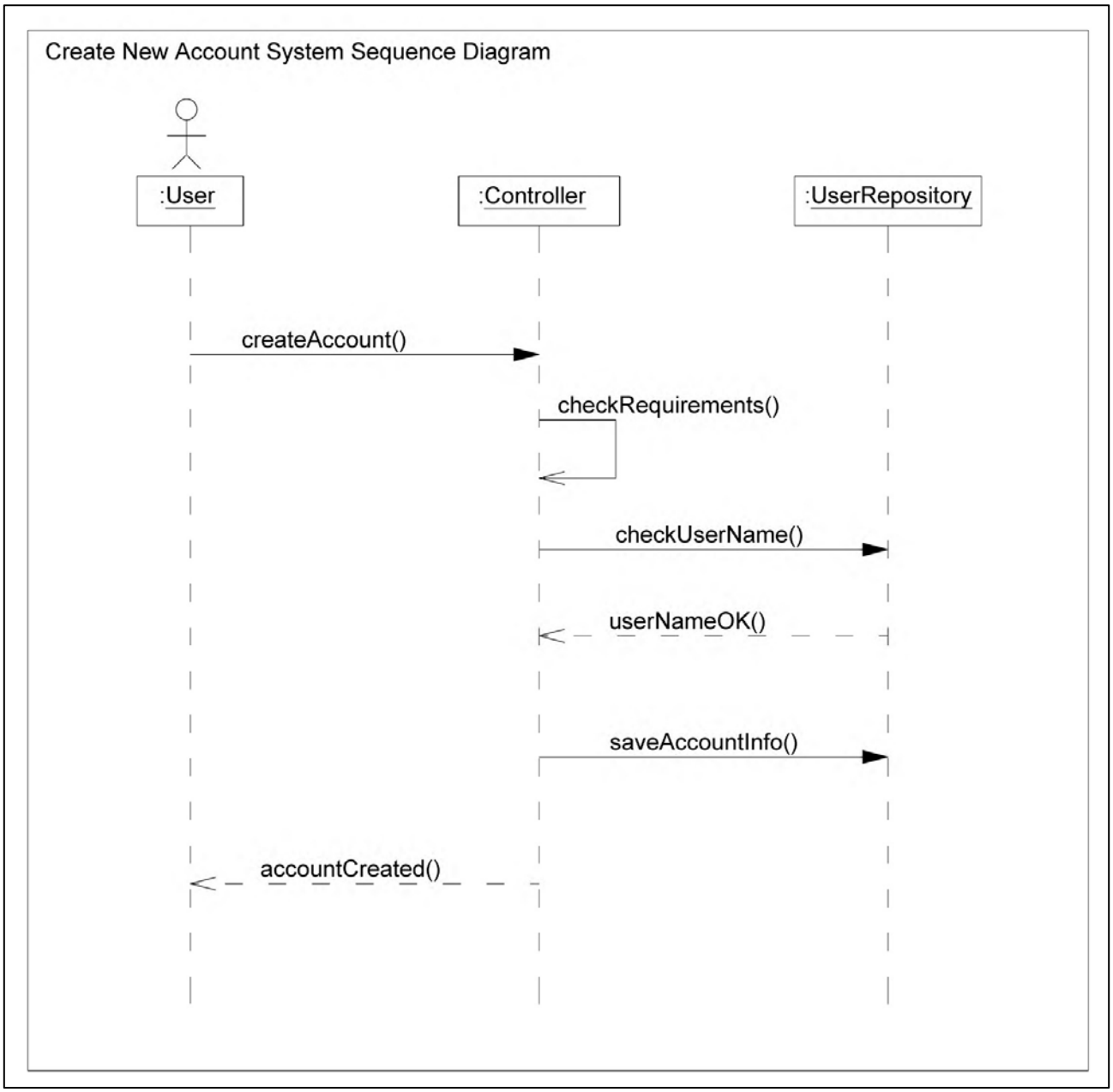
Number	004	
Name	Choose Stock	
Summary	User selects a stock from a list (DOW 30 components)	
Priority	1	
Precondition	User has logged into their account.	
Postcondition	The stock has been chosen and saved.	
Primary Actor(s)	User	
Secondary Actor(s)	None	
Trigger	User has selected choose stock option	
Main Scenario	**Step**	**Action**
	1	User selects the choose stock option.
	2	System displays a list of stocks (DOW 30 components).
	3	User selects their choice from the list.
	4	System registers the selection.
Extensions	**Step**	**Branching Action**
	3a	User moves on without selecting a stocks choice.
	3a-1	System cancels operation.
Open Issues	0	

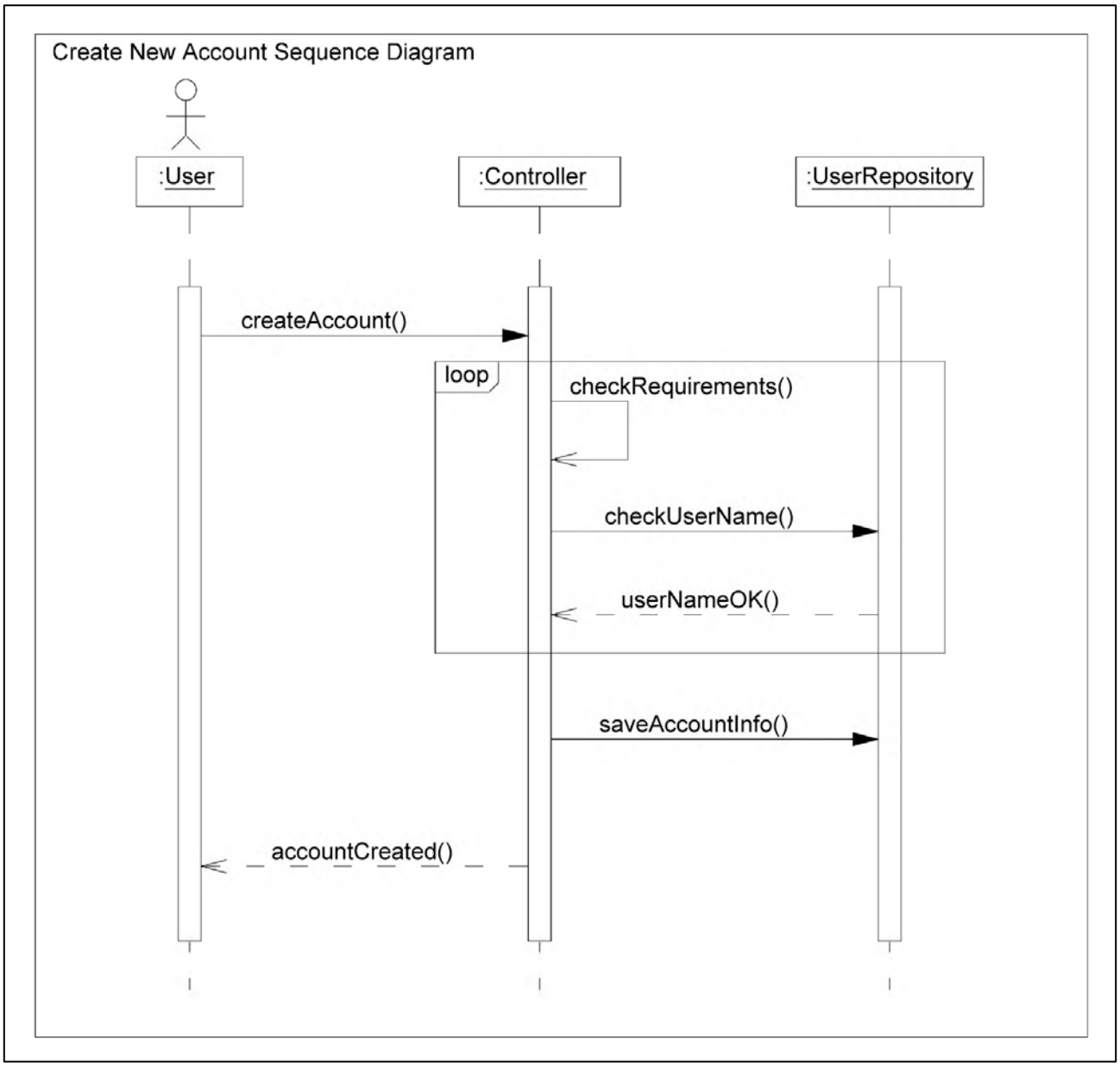

Choose Stock System Sequence Diagram
:User
:System
requestStockList()
displayingStockList()
selectStock()
saveStockSelection(stock)
displayingStockSelection()


Choose Stock Sequence Diagram
:User
:Controller
:UserInterface
requestStockList()
displayStockList()
displayingStockList()
selectStock(stock)
saveStockSelection(stock)
displayStockSelection()
displayingStockSelection()

3.6 Use Case 005 – Create Account

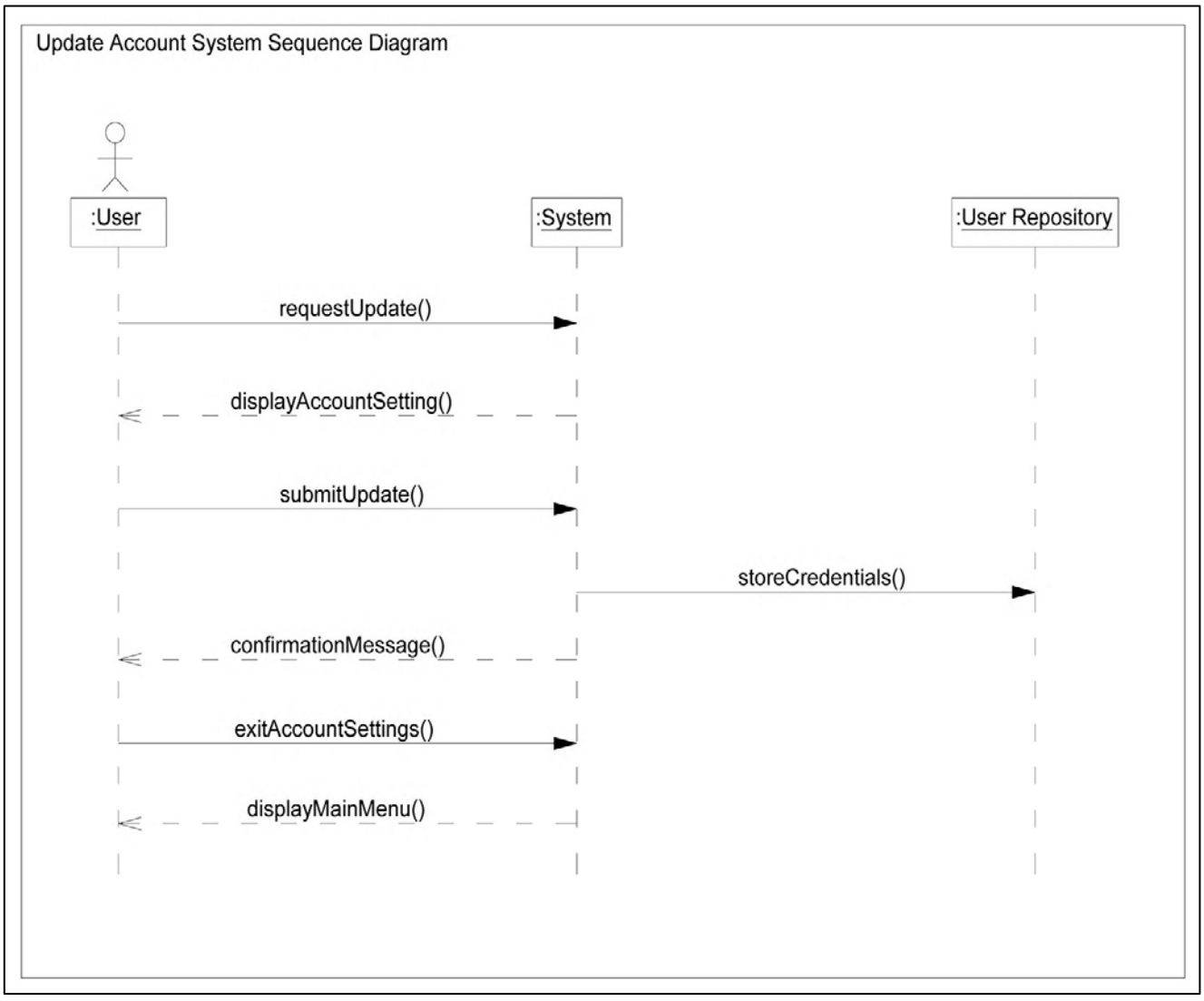
Number	005	
Name	Create Account	
Summary	Employees at ProfitsRUS and their clients will use the software to create a new account	
Priority	1	
Precondition	Software is available on user's computer and is connected to profitsRUS database	
Postcondition	A new account is created and user returns to login page	
Primary Actor(s)	User	
Secondary Actor(s)	User Data Repository	
Trigger	User has selected the "REGISTER NEW ACCOUNT" option	
Main Scenario	**Step**	**Action**
	1	User selects the "CREATE A NEW ACCOUNT" option.
	2	User enters his/her preferred User Name and Password
	3	System checks if user's User Name and Password matches the requirements (if there are any). System also checks if the User Name already exists or not.
	4	System returns the user to login page
Extensions	**Step**	**Branching Action**
	3a	System finds that user's User Name and Password does not match the requirements or User Name has already been taken.
	3a-1	System displays message stating that which of user's choices need to change
	3a-2	User changes his/her choices based on the requirements and availability of an user name
Open Issues	0	


Create New Account System Sequence Diagram
:User
:Controller
:UserRepository
createAccount()
checkRequirements()
checkUserName()
userNameOK()
saveAccountInfo()
accountCreated()

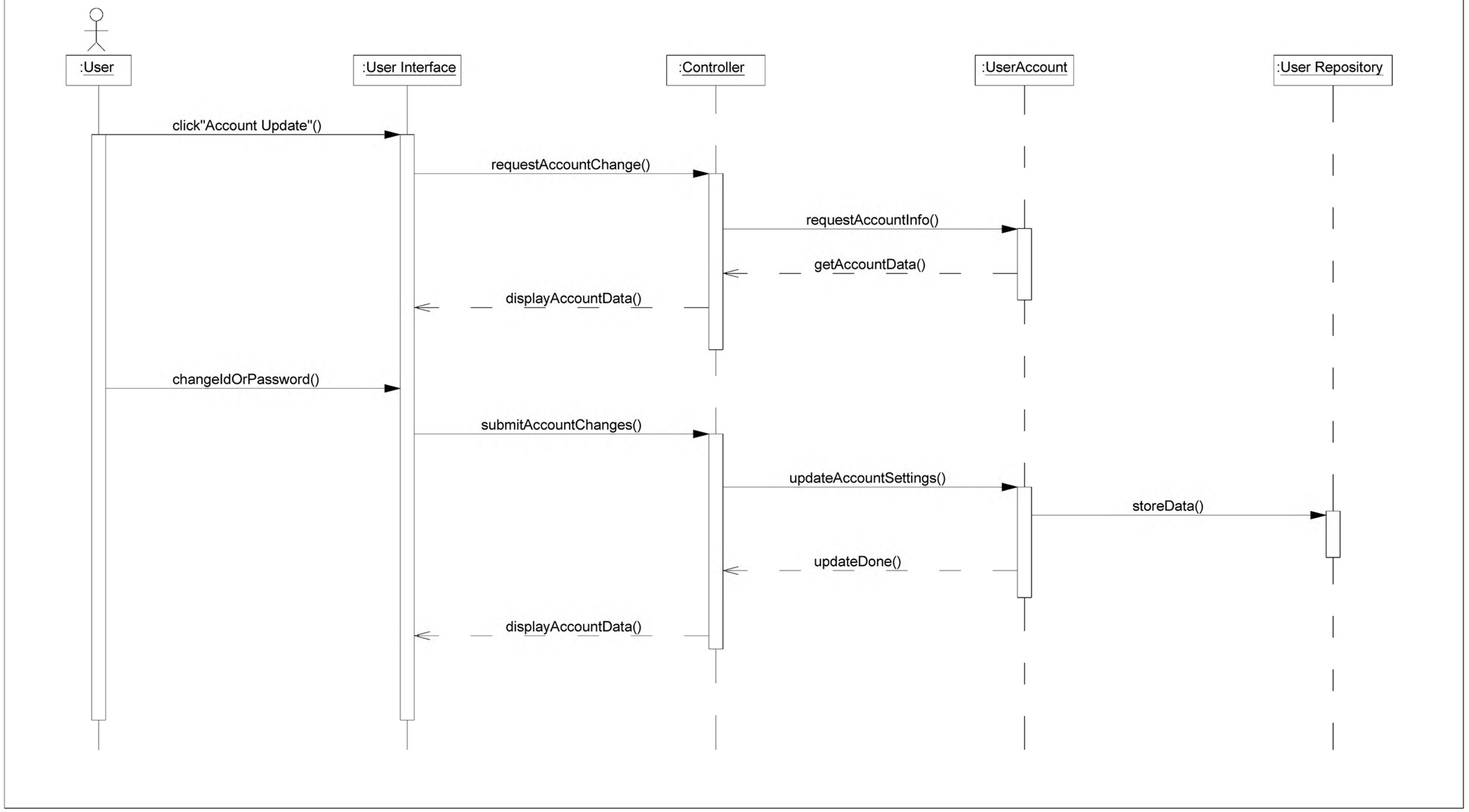

Create New Account Sequence Diagram
:User
:Controller
:UserRepository
createAccount()
loop
checkRequirements()
checkUserName()
userNameOK()
saveAccountInfo()
accountCreated()

3.7 Use Case 006 – Change Password

Number	006	
Name	Change Password	
Summary	Changes User Password	
Priority	2	
Pre-Condition	A User Account Exists	
Post-Condition	User Account is updated	
Primary Actor(s)	User	
Secondary Actor(s)	System	
Trigger	User selects update Account option	
Main Scenario	**Step**	**Action**
	1	User selects update Account
	2	System opens Account info Box
	3	User writes new Account parameters
	4	User selects save
	5	System Validates data is OK
Extensions	**Step**	Branching Action
	0	
Open Issues	0	


Update Account System Sequence Diagram
:User
:System
:User Repository
requestUpdate()
displayAccountSetting()
submitUpdate()
storeCredentials()
confirmationMessage()
exitAccountSettings()
displayMainMenu()

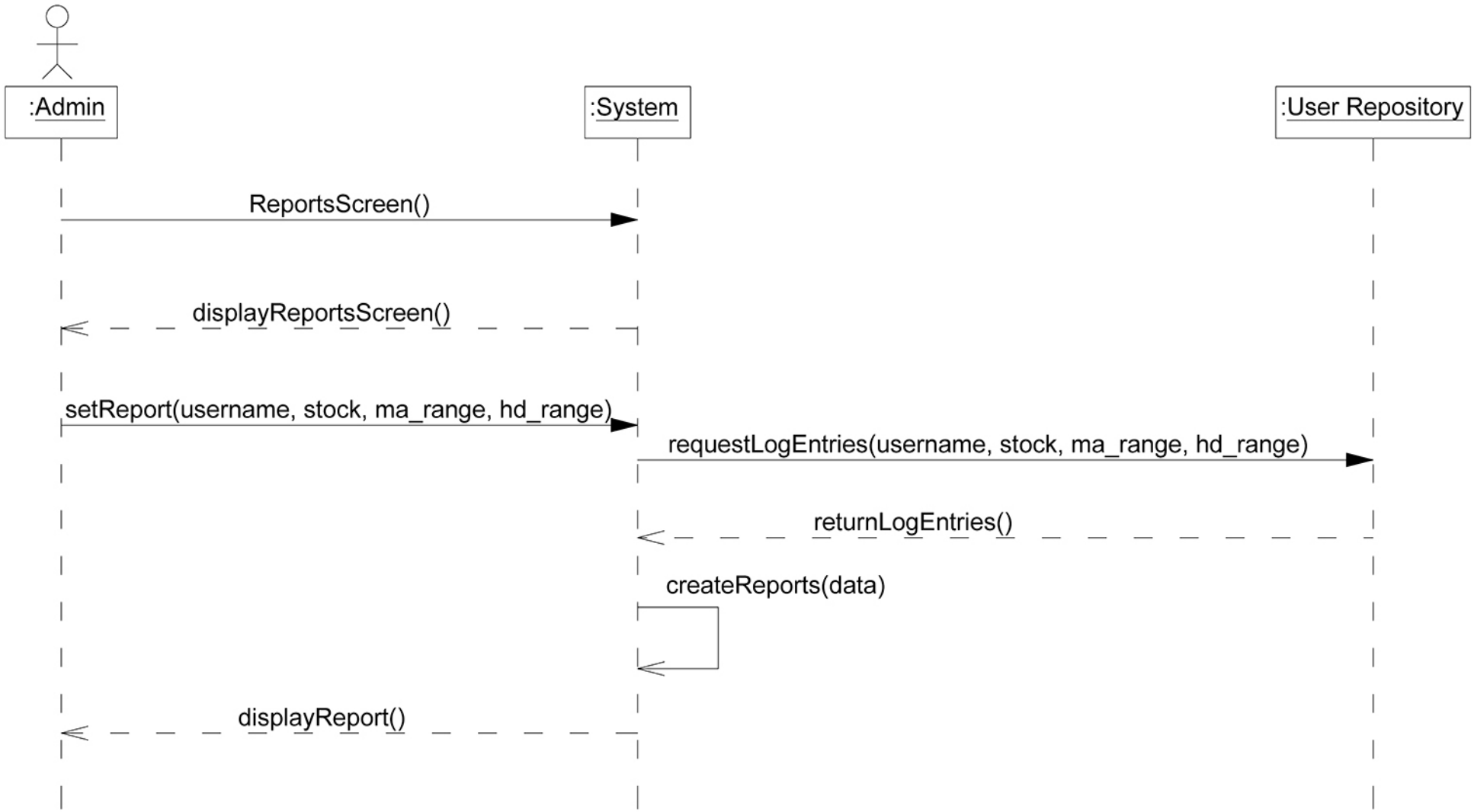
Update Account Sequence Diagram



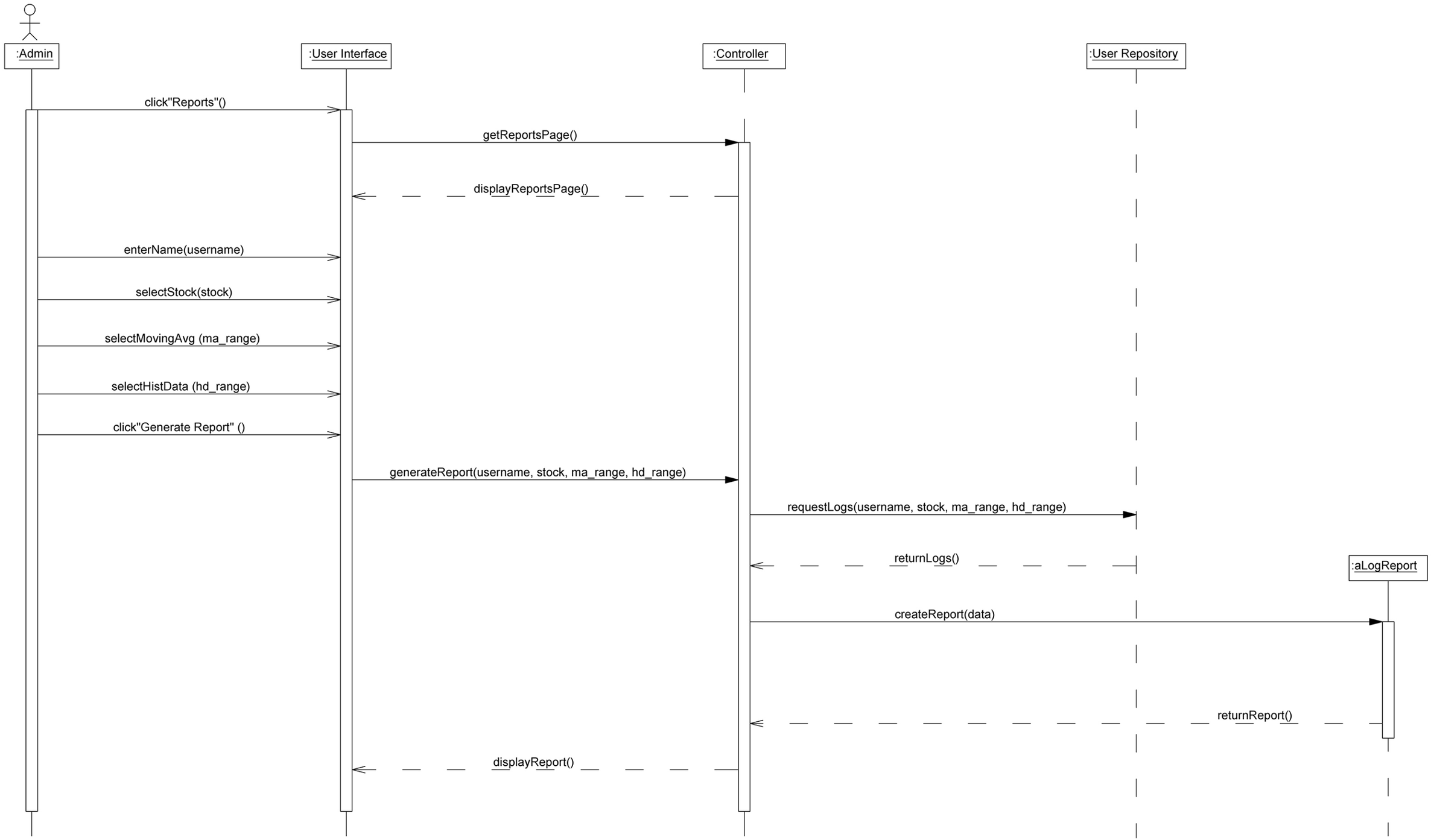
3.8 Use Case 007 – Generate Report

Number	007	
Name	Generate Report	
Summary	Administrator (user with Administrator privileges requests a report of application usage data from the System.	
Priority	1	
Precondition	User has logged into an account with necessary privileges to access usage data logs	
Postcondition	App usage data is displayed in a window.	
Primary Actor(s)	Administrator (user with necessary privileges)	
Secondary Actor(s)	User Data Repository	
Trigger	Administrator has selected the "Reports” option from Side Menu.	
Main Scenario	**Step**	**Action**
	1	Administrator selects the "Reports” option from Side Menu.
	2	System displays a window prompting the Administrator to select an option for the Report from particular Stock, MA Range, and HD Range options. Administrator may choose "All data" for all of those options (according to a scheme displayed on the window). System also displays a box for the Administrator to type a user name into and search for.
	3	Administrator choses Stock, MA Range, and HD Range for the Report, types in a username to search for (or leaves the box blank to broaden the search to any user name) and selects “Generate Report”.
	4	System sends a read request to User Data Repository.
	5	System retrieves data entries from User Data Repository logged within selected range.
	6	System uses selected entries to create a Report with requested parameters.
	7	System displays Report on a window, which the Administrator may close.
Extensions	**Step**	**Branching Action**
	2a	System finds that user does not have necessary privileges to request logs.
	2a-1	System displays message that user does not have the necessary privileges.
	5a	User Data Repository does not respond.
	5a-1	System re-attempts to contact the User Data Repository a certain number of times and proceeds to step 6 upon success
	5a-2	System informs Administrator that contact with User Data Repository could not be established and asks them to try again later or contact support.
Open Issues	0	

Generate Report System Sequence Diagram



Generate Report Sequence Diagram



4 Architectural Design

4.1 Rationale

The architecture chosen for 'The Share Buy/Sell indicator tool' (or simply the tool from now on), is the Model View Controller model (MVC). The MVC architecture is made up of 3 separate component sets, called the models, the views and the controllers.

The model component is the core of the tool where, based on the user choices, the stock data is transformed in charts and calculation for indicators 'BUY' and "SELL' are performed. Moreover, any request to change the tool state is also handled by the model. Whenever there are changes made to the model, the model updates the view.

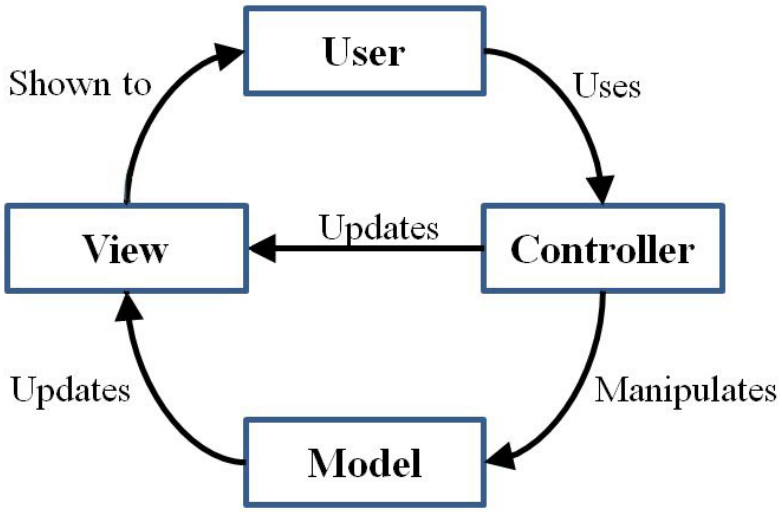
The view is the graphical user interface (GUI) of the 'Indicator Tool' and displays the data from the model. Whenever the model changes, the view responds to those changes. The view also gets updated by the controller, as the controller performs simple data validations on user input and updates the view in the form of a pop-up if any errors are detected. Different versions of the view can be developed in order to present the same data in different ways.

The controller is what the end user uses to interact with the Indicator Tool. All user choices are entered via the controller, which then passes them on to the model. The controller also interprets all mouse clicks or Indicator Tool events and it determines which part of the model needs to be manipulated. Basic input validations are also performed in the controller, which updates the view if an error is detected.

The MVC architecture allows the three components to be developed separately from one another and they can be done in parallel. This way either of the components can be updated without affecting the other components as long as their application program interface remains the same.

With a common API, the 3 models can be seamlessly integrated into one Indicator Tool and different versions of the models can be used in the Indicator Tool. This architecture allows for many different versions of views to be developed, which could be used to make variations of the Indicator Tool based on different themes.

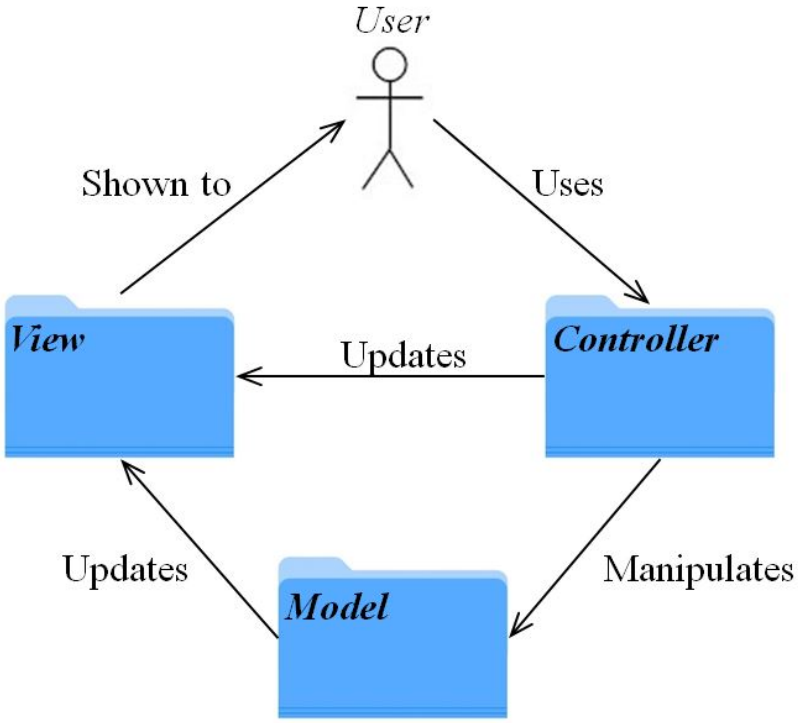
The main reason why we chose the MVC model for the Indicator Tool is that it allows the GUI to be separated from the core application. This flexibility allows the Indicator Tool's core to be developed by the implementation team and the GUI to be designed by the design team. Both teams could then work in parallel and easily collaborate on any changes that need to be made. By using this architecture, enhancements to the Indicator Tool's GUI could be made without having to modify any of the Indicator Tool's core functionality. Furthermore, several models could be developed in parallel with each model containing different Indicator Tool functionalities. Each model could be then thoroughly tested and debugged. Once the quality assurance on the model has been completed, its functionalities could be integrated into future models.



MVC Architecture

4.2 Software Architecture Diagram

The three components of the MVC model that are used in the Maester Buy/Sell Tool are shown below in the form of a high-level class diagram. The user interact directly with the view and the controller, while the model accepts requests from the controller and updates the view.



High-level Class Diagram of an MVC model

The tool controller accepts input from the user via the mouse and the keyboard through the use of Java FX events. Simple validations, like input checking, are done by the controller and will inform the user of any faulty data entered with a message box. Moreover, if a user tries to perform an illegal action, the controller will also inform the user via an error message on the GUI. Any valid action performed by the user will be accepted by the controller, which will then relay the information to the model.

The model, when instructed by the controller, will perform the task that was requested. All of the tool's data is stored in the model, which also contains the core functionalities of the tool. Whenever there is a change to the model’s data, the model updates the view to reflect the change.

The view consists of the GUI, which is the visual aspect of the tool. With the GUI, user can monitor the tool response it also provides visual cues to let the users know where they can click on the screen and where they can enter data. This input is handled by the controller, which then gets manipulated by the model and in turn, updates the view.

4.3 System topology

The Share Buy/Sell indicator tool is to be developed for a standalone environment and each installation of the product is to be run on a single computer. This will allow for easy distribution of the tool, as all the components of the product will be integrated into one executable. Moreover, the tool does not require any third party software to run.

5 Software Design

5.1 System Interface Diagrams

The only system level interface in the Maester's Share Buy/Sell indicator tool' is the user interface, as the tool does not employ any software or hardware interfaces. The user interface is the tool's GUI, which allows the user to interact with the tool. Through the tool's GUI, the user will be able to see the different states and charts produced from the tool and make decisions based on the view.

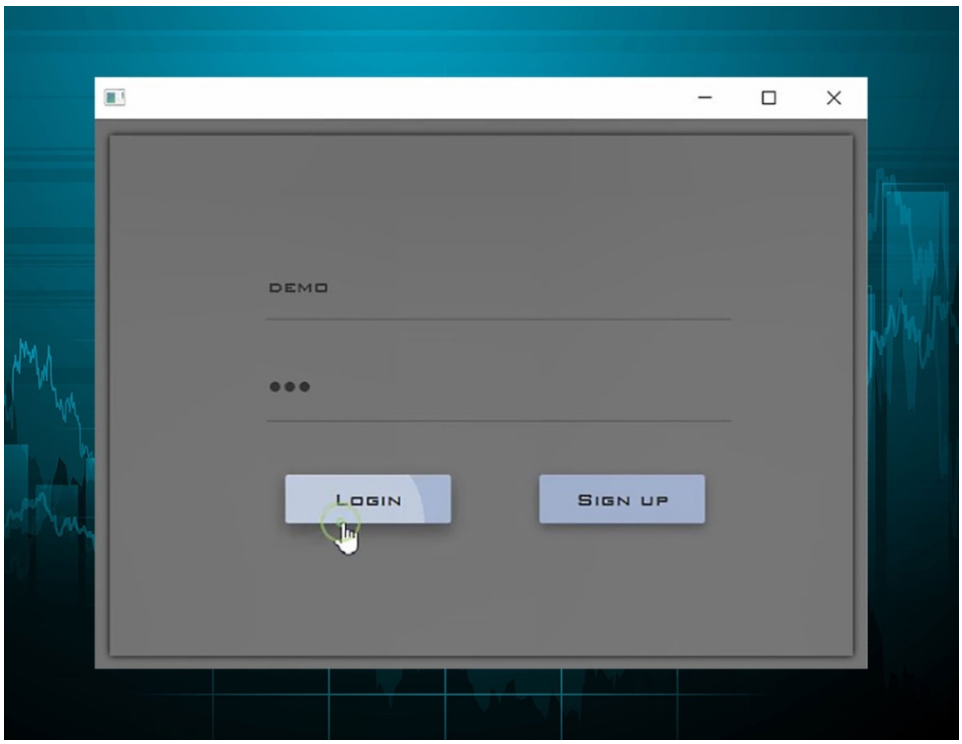
5.1.1 User Interface

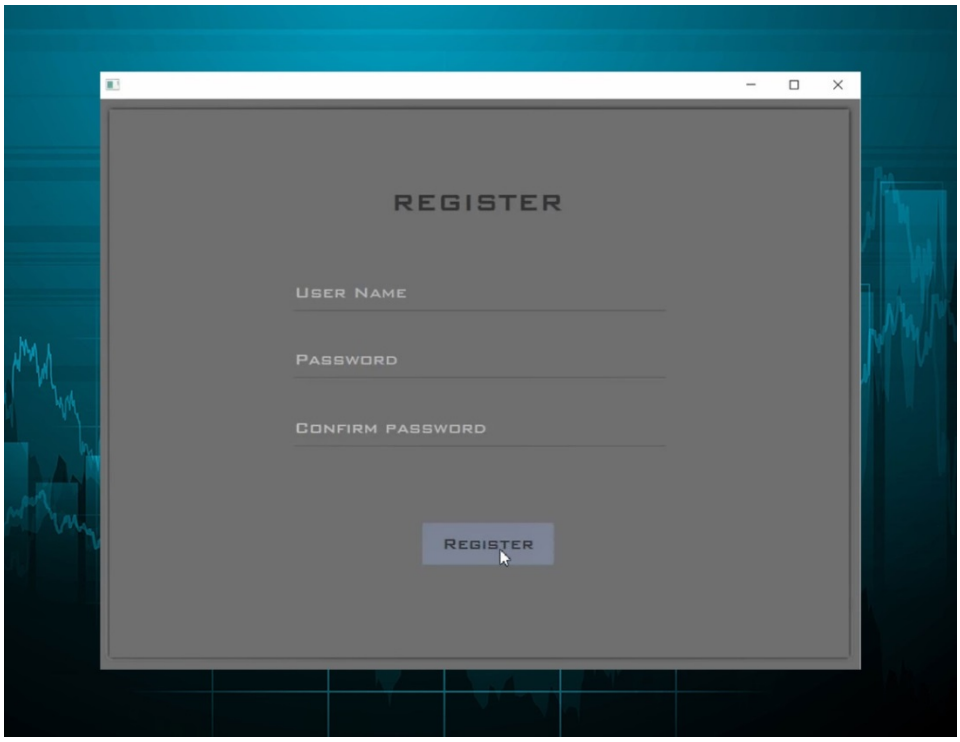
In the Indicator tool, the user interfaces are the links between the users and computer. In order to complete a task efficiently and with ease, an intuitive user interface must be developed. To achieve this, several points should be taken into consideration:

- Complete information: users should find all elements they need quickly to make a choice;
- Error avoidance: avoid utilization of ambiguous terms or concepts. If an error occurs, an error message should tell what kind of error has occurred, why it has occurred, and how to solve it;
- Usability: components should be explicit enough for users in order for them to intuitively know which actions they will perform.

The graphical user interfaces (GUI) that the user will interact with are described below.

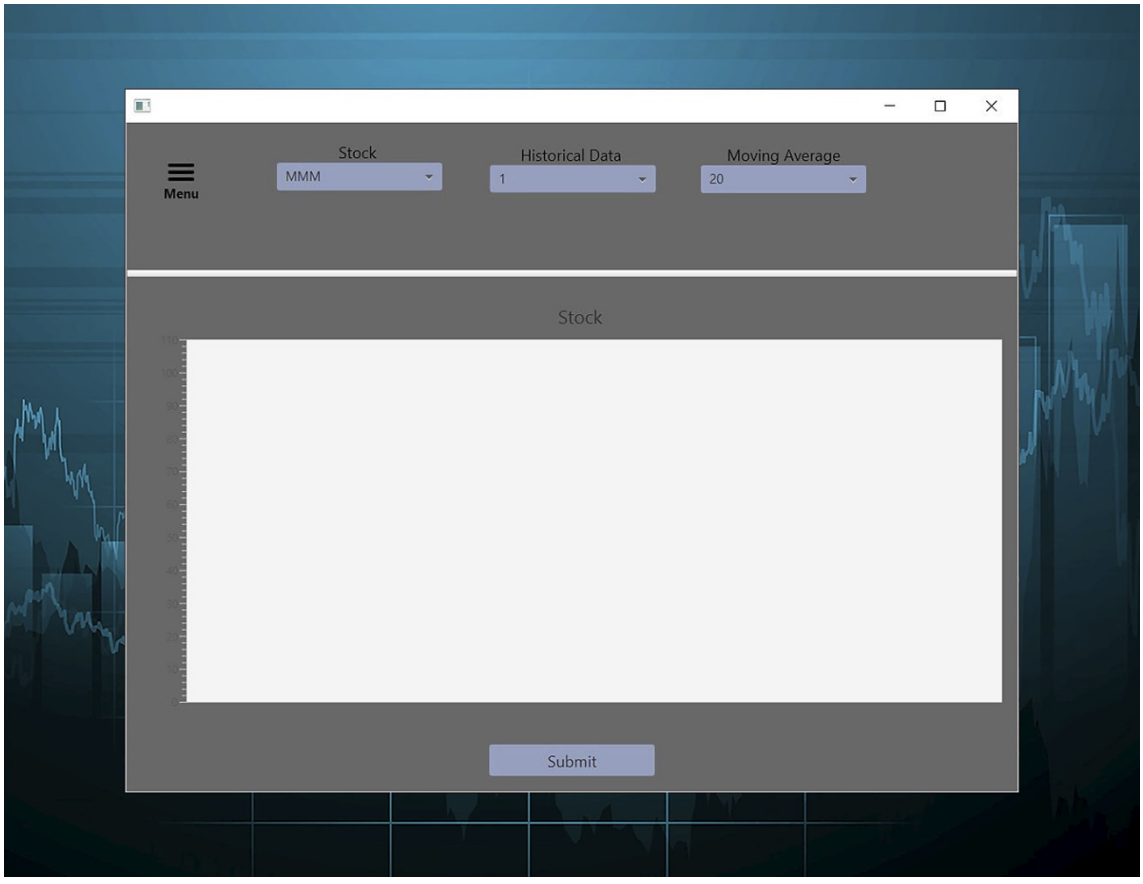
5.1.1.1. Login/Register Page: All users are required to either login to an existing account or sign up for a new one. To login, the user is required to enter a username and a password. New users are required to register in order to use the application. Registration is done by providing the following information: username, password and confirmation of the password. The user must click the Register button to finalize the process.



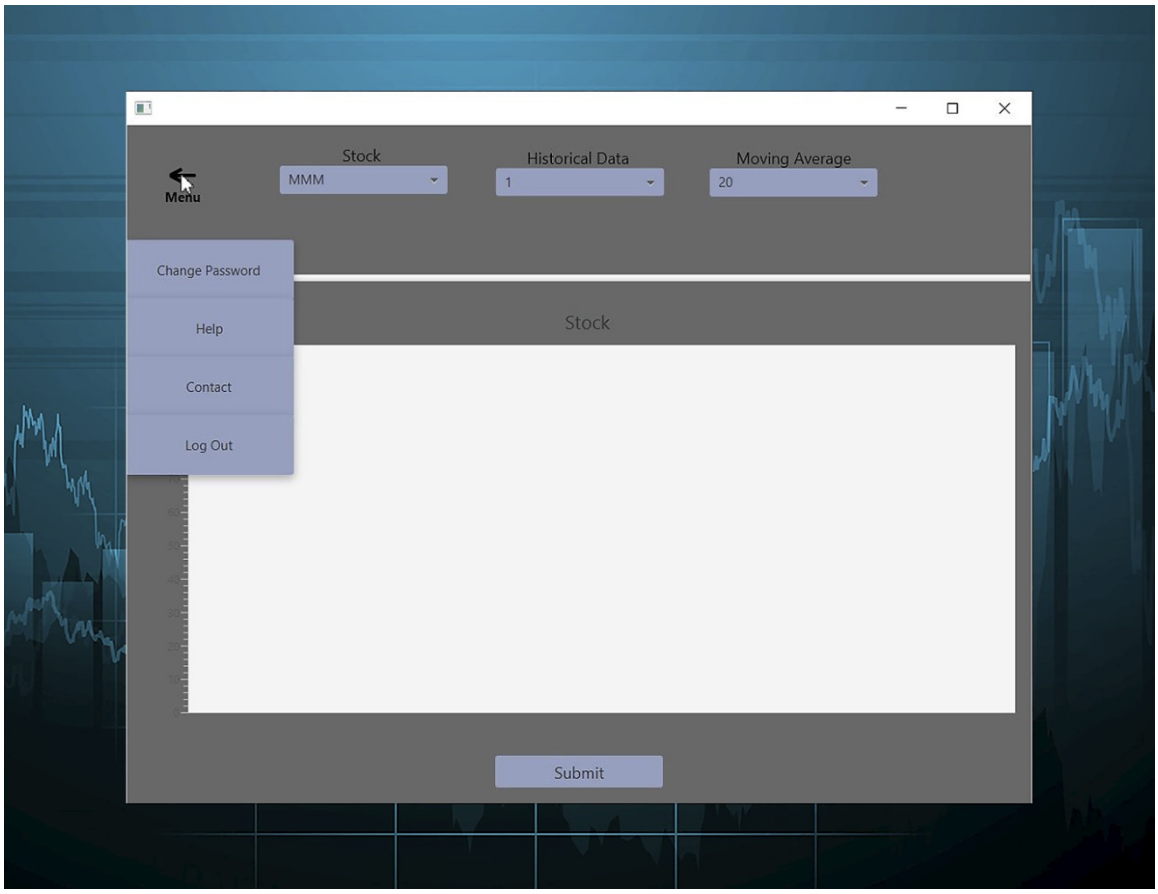


5.1.1.2 User's Main Page: Upon Logging-in, the user is sent to his/her home page. Here the main functionality of the tool is presented by the three drop-down menus:

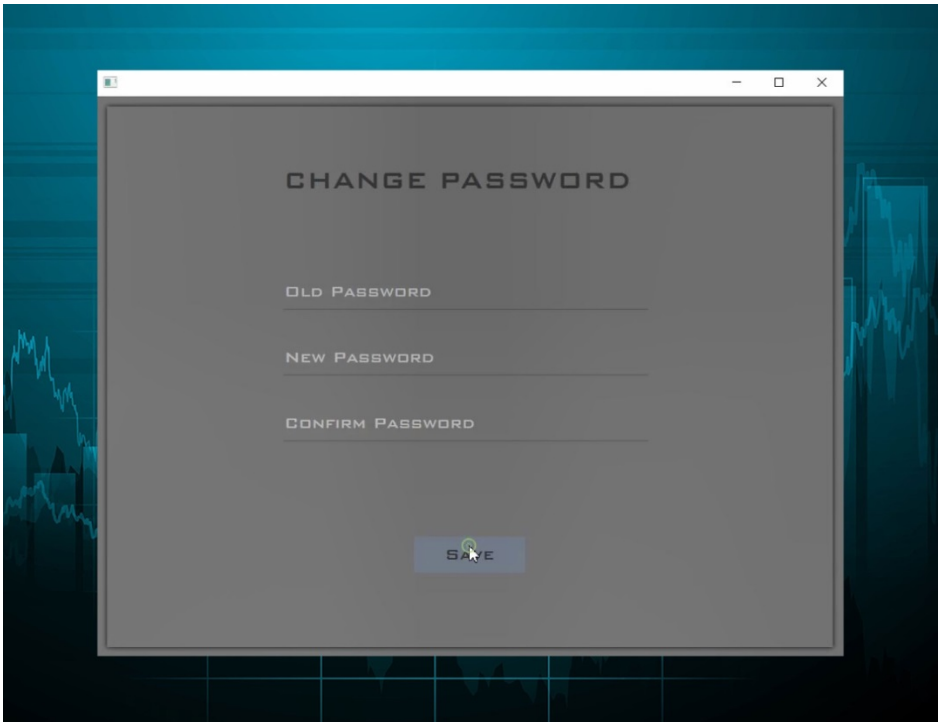
STOCK, HISTORICAL DATA, MOVING AVERAGE and the **SUBMIT** button



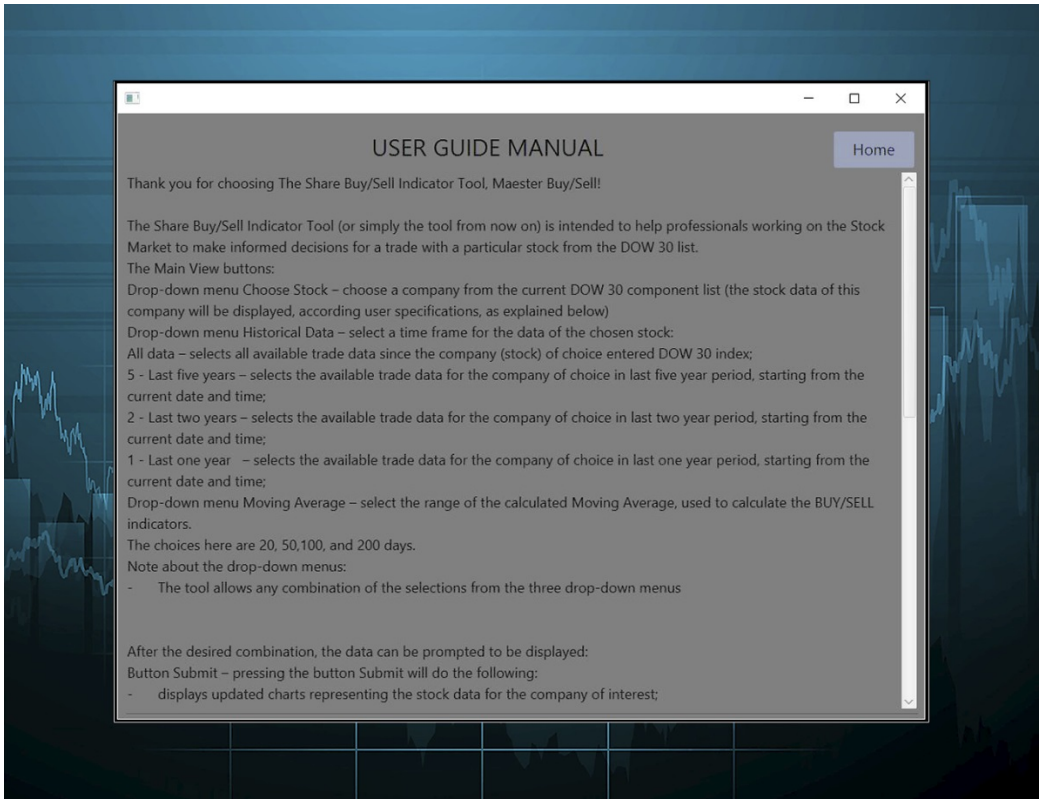
The **Menu** button brings up a **Side Menu** - the users is presented with three additional options: **Change Password, Help**, **Contact** and **Log Out**.



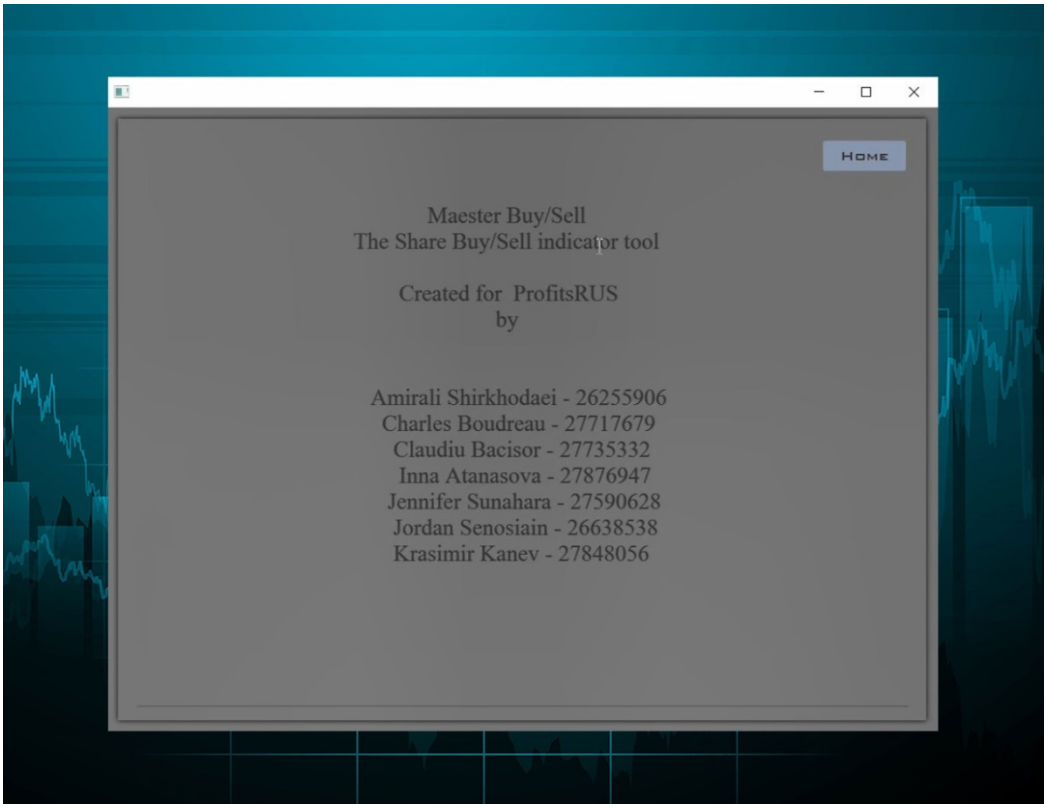
- User chooses **Change Password** from the **Side Menu:**



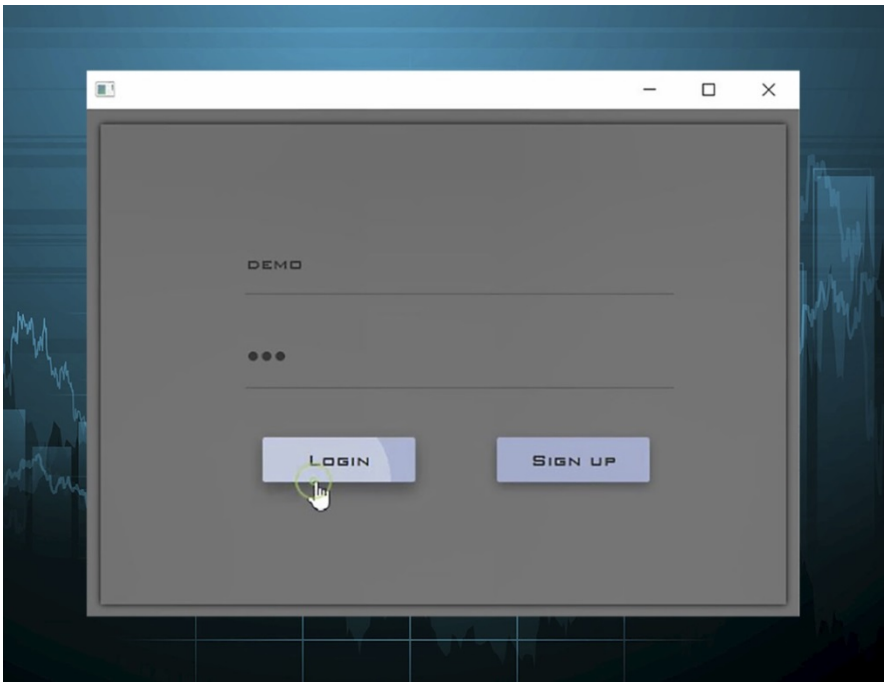
- User chooses **Help** from the **Side Menu**



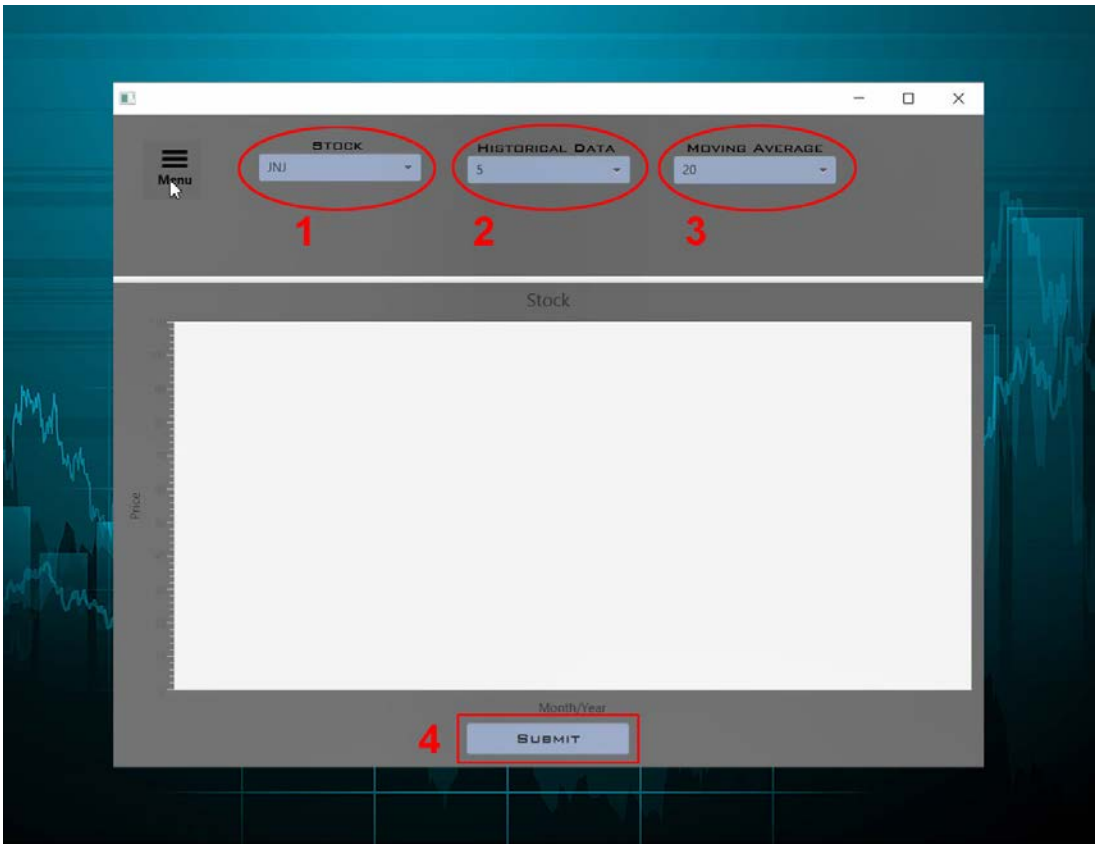
- User chooses **Contact** from the **Side Menu**



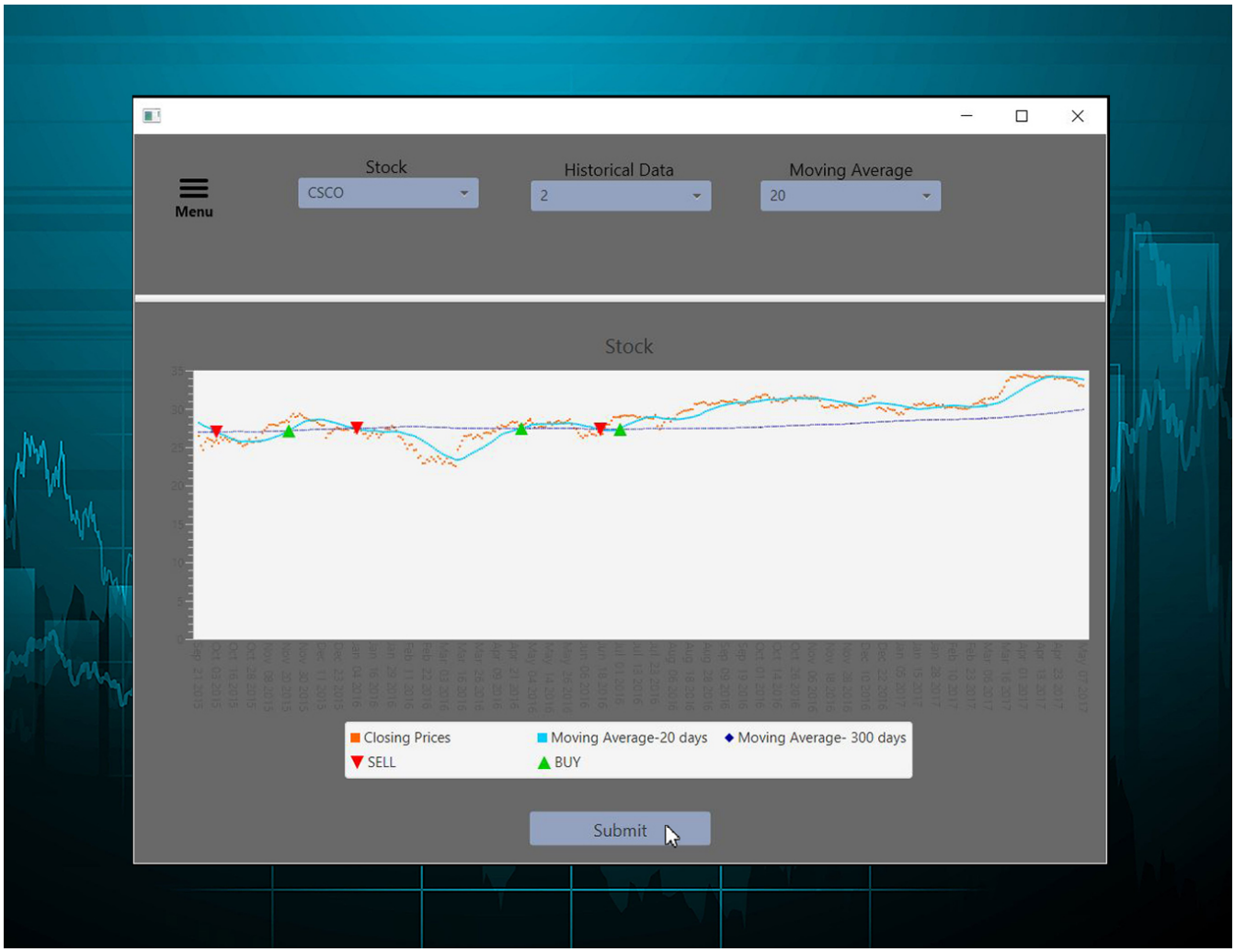
- User chooses **Log-Out** from the **Side Menu.** Session is closed, the Log-In screen is displayed:



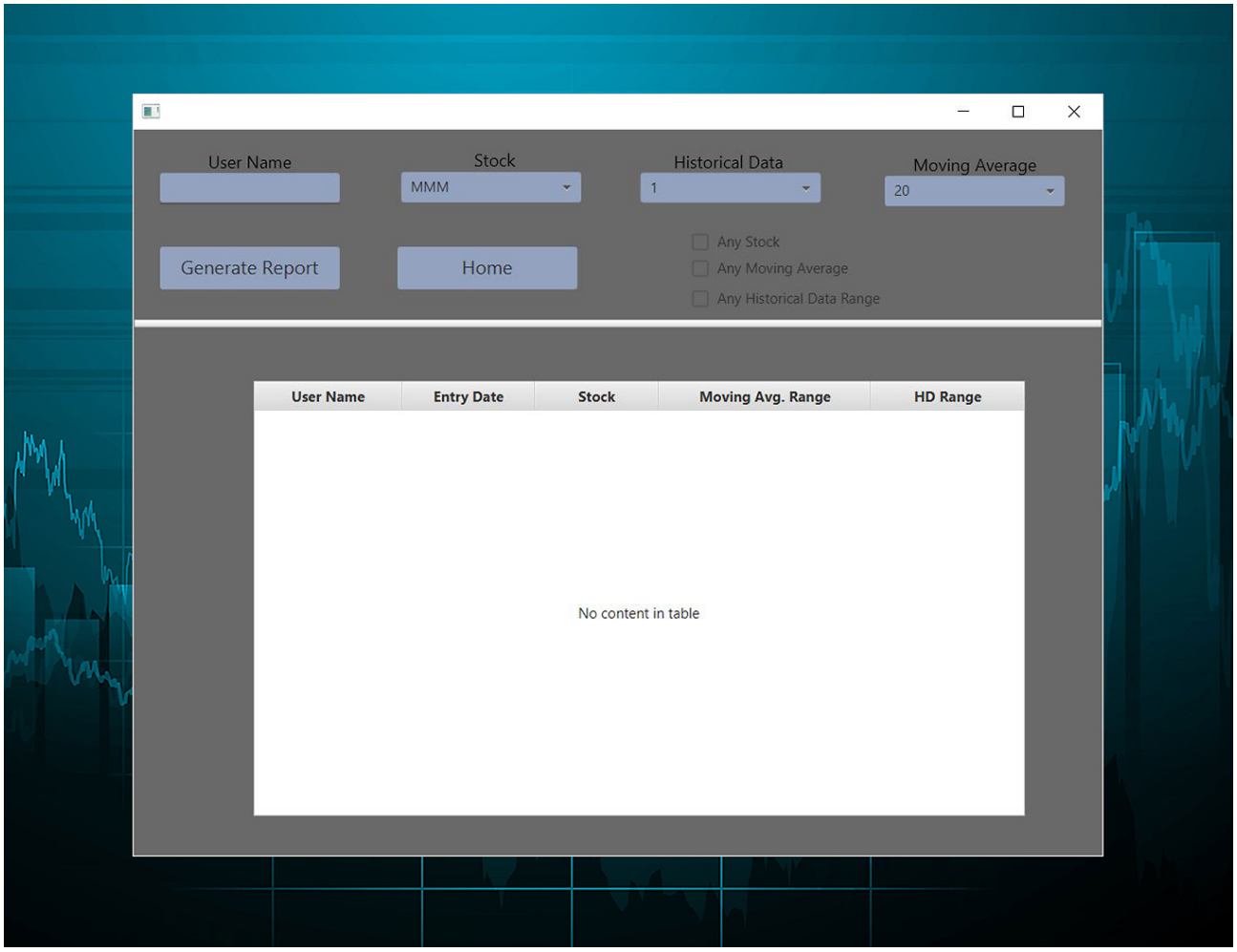
1. Choose **Stock** drop-down menu: user selects between the current DOW 30 stocks;
2. Historical Data drop-down menu: user chooses between, last 1, 2, 5 years and "All data";
3. Moving Average drop-down menu: user chooses between 20, 50,100, and 200 days.
(Default values as of version 1.2 release are 3M Company, last 1 year, 20 days)



4. Submit button: Once the user's selection is done, after pressing the **Submit** button, the page will display the graph of the stock's historical and prediction data.
The "Moving Average – 300 days" of the company of choice is always displayed along with the other charts as a long-term reference.
Buy/Sell signals are indicated in the charts.
A legend for the used colors will be displayed just above the **Submit** button



5.1.1.3 Administrative Option - Reports: The option will be visible in the **Side Menu** for all types of user accounts. Although the option will, be available only in the accounts of users with granted **administrative privileges**. Choosing Reports from the Side Menu will invoke the following screen:



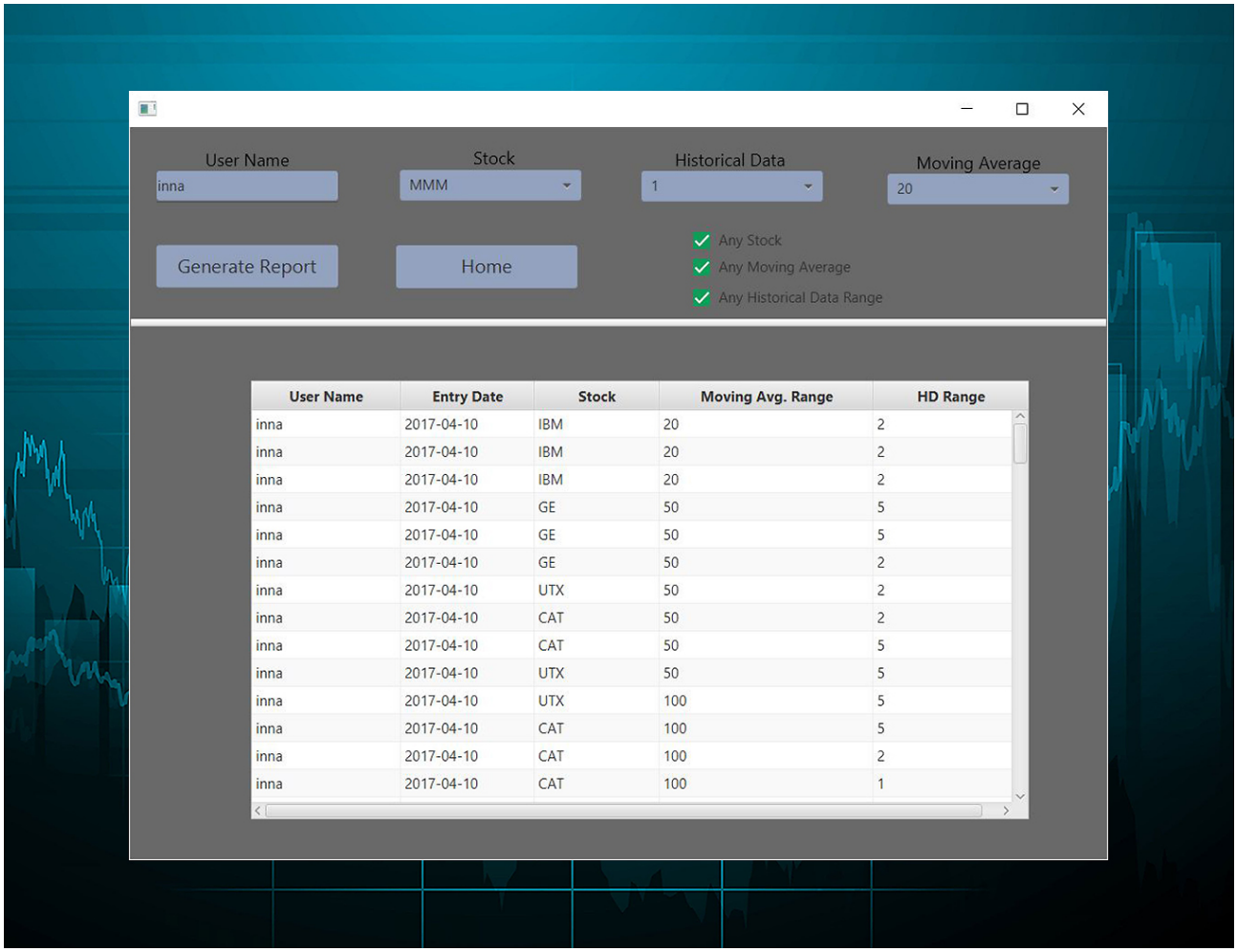
- Text field **User Name**: To enter the name of the user for which a report needs to be created;

- Drop-down menus **Stock, Historical Data, Moving Averages**: Selecting the limiting parameters for exactly what data the report will contain;

- Check boxes **Any Stock, Any Moving Average, Any Historical Data Range**: checking a box will include all the date for the parameter of choice into the report

- Any combination of all parameter selections is possible;

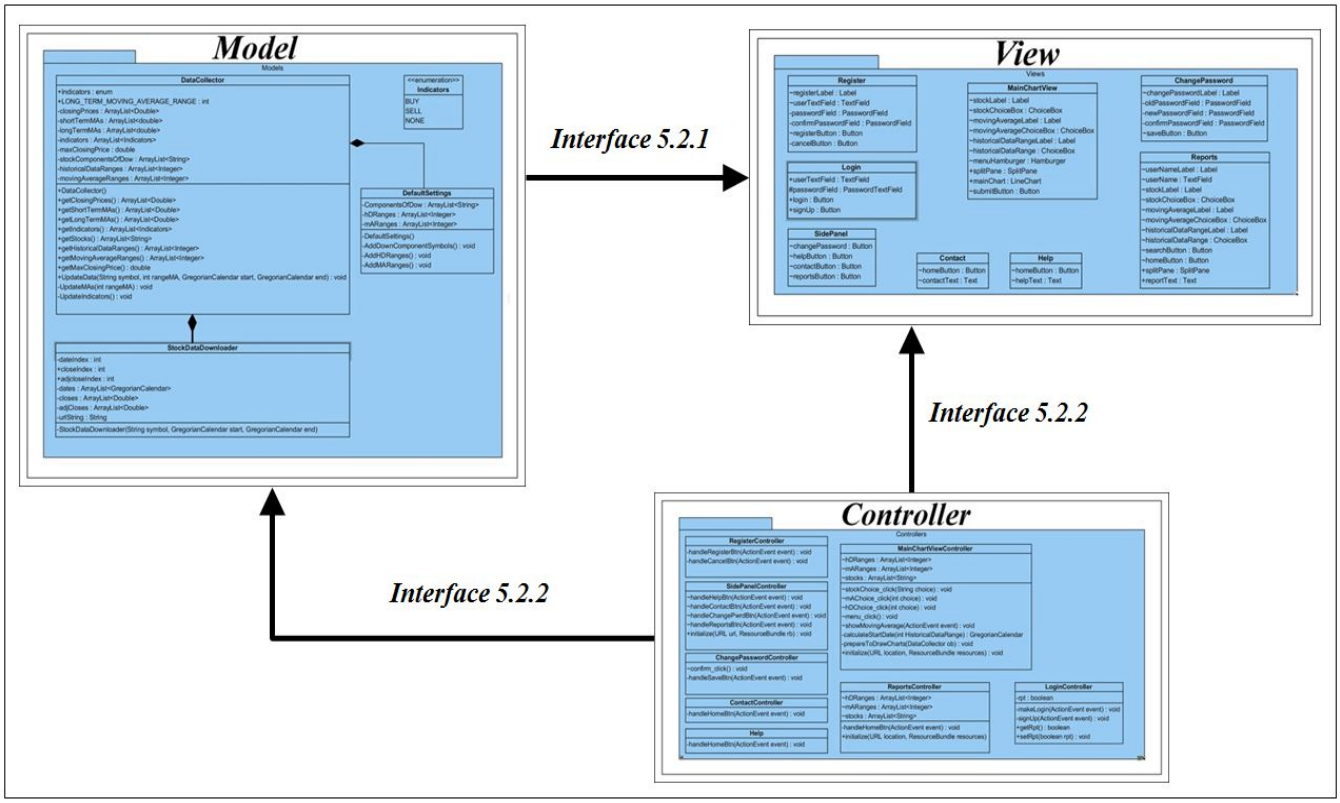
- Button **Generate Report**: after selecting the parameters of interest, pressing the button **Generate Report** will display a Report for the entered User Name:



User Name	Entry Date	Stock	Moving Avg. Range	HD Range
inna	2017-04-10	IBM	20	2
inna	2017-04-10	IBM	20	2
inna	2017-04-10	IBM	20	2
inna	2017-04-10	GE	50	5
inna	2017-04-10	GE	50	5
inna	2017-04-10	GE	50	2
inna	2017-04-10	UTX	50	2
inna	2017-04-10	CAT	50	2
inna	2017-04-10	CAT	50	5
inna	2017-04-10	UTX	50	5
inna	2017-04-10	UTX	100	5
inna	2017-04-10	CAT	100	5
inna	2017-04-10	CAT	100	2
inna	2017-04-10	CAT	100	1

Button **Home**: will take back the user to the **User's Main Page**

5.2 Module Interface Diagram



5.2.1 View Interface: The interface between the model and view is called by the model whenever there is a change in its internal data structures.

5.2.2 Model Interface: The interface between the controller and model is called whenever the controller receives input from the user.

5.2.3 Controller Interface: The controller accepts input from the user and performs simple validations on it.

5.3 Hardware Interfaces

All hardware interfaces are handled by the Java Virtual Machine.

5.4 Software Interfaces

The **Maester Buy/Sell** application is developed under Java 8 and requires a compatible Java Runtime Environment (JRE) to be installed on the running machine. The system will be connected to a database managed by SQL software.

Description of File Format: Input

The application accepts a .CSV file (Comma separated value). A CSV file consists of a series of records (Date, Open, High, Low, Close, Volume, Adj Close), separated by line breaks for a particular stock. The line break represents the start of a new row of data. Each Row consists of fields, separated by some other character or string, most commonly a literal comma (,) or tab. All records have an identical sequence of fields.

Description of File Format: Output

The display output is a graph that will display two moving averages from a selected stock, with:

- buy signals : small green arrow up should be shown in the graph at the crossing of two moving averages
- sell signals: small red arrow down should be shown in the graph at the crossing of two moving averages

5.5. Communications Interfaces

The application runs across multiple computers and subsystems. The software is a standalone application and will access the internet only to receive statistical data.

6. Internal Component Design

6.1 Component < Model >

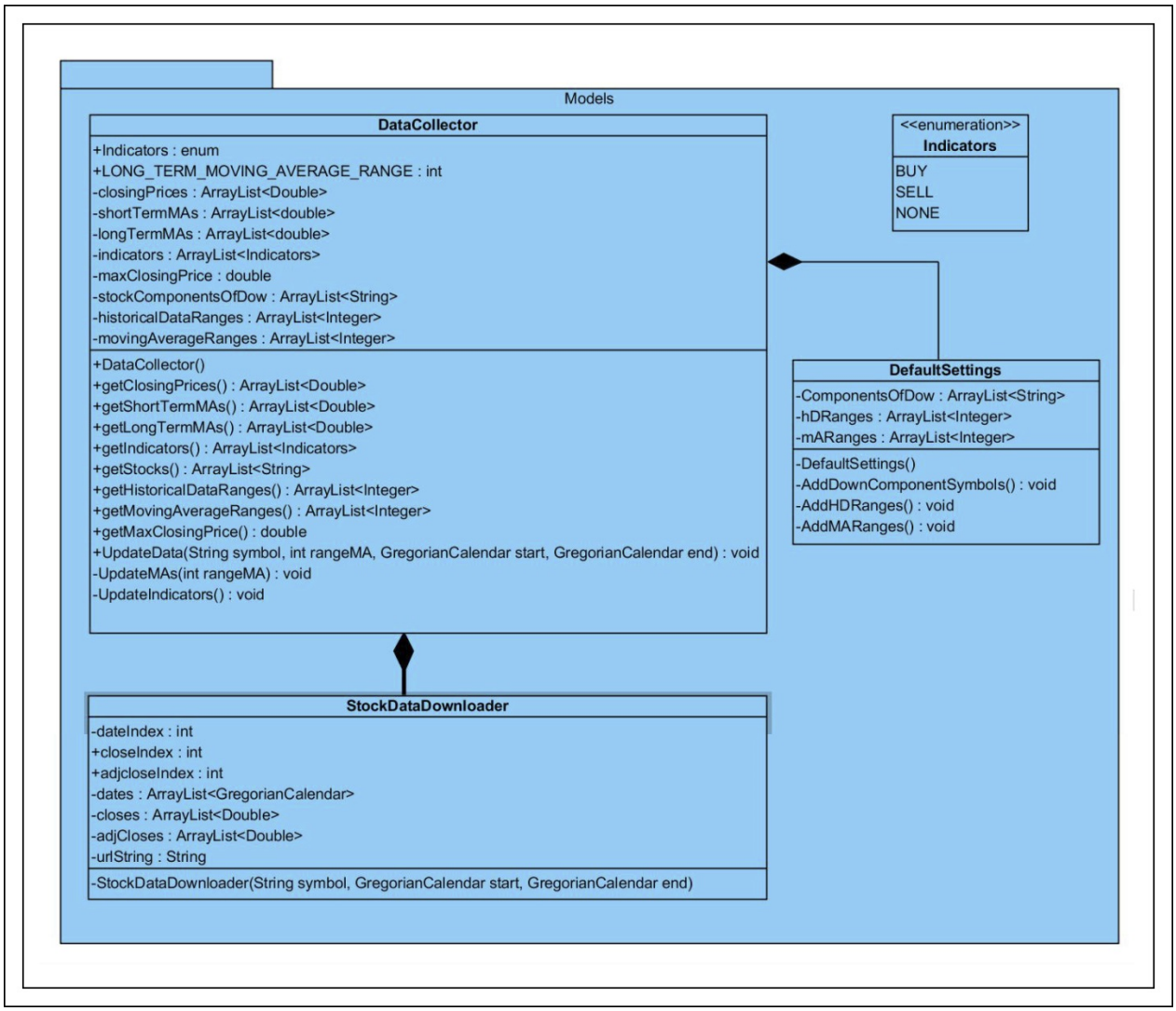
The most important component in the Maester's tool design is the model component. This component is used to represent the state and actions that occur in tool. This includes data collection, data processing, charts and indicator calculations, recording user states. To perform those tasks the model consists of the following classes:

- DataCollector
- StockDataDownloader
- DefaultSettings
- Indicators

Where the following compositions are formed:

- class DataColector **has** StockDataDownloader
- class DataColector **has** DefaultSettings

Those are strong aggregations, i.e. the aggregated classes could not function alone.



6.1.1 Class <DataCollector>

Class Name	DataCollector			
Inherits From	None			
Description	The DataCollector acquires stock data from Yahoo and saves it in an array, using it to calculate the moving averages.			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Public	enum	Indicators	Enumerated type for indicators: BUY, SELL, and NONE
	public	int	LONG_TERM_MOVING_AVERAGE_RANGE	Fixed value for long term moving average: 300
	private	ArrayList<Double>	closingPrices	List of stock closing prices
	private	ArrayList<Double>	shortTermMAs	List of short-term moving averages
	private	ArrayList<Double>	longTermMAs	List of Long-Term moving averages
	private	ArrayList<Indicators>	indicators	List of indicators
	private	Double	maxClosingPrice	Highest closing price recorded
	private	ArrayList<String>	stockComponentsOfDow	List of stocks whose prices can be consulted
	private	ArrayList<Integer>	historicalDataRanges	List of historical data ranges available for selection
	private	ArrayList<integer>	movingAverageRanges	List of short-term moving averages that can be used

Methods	**Visibility**	**Method Name**	**Description**
	public	DataCollector()	Constructor
	public	getClosingPrices()	Return list of closing prices
	public	getShortTermMAs()	Return list of calculated short-term moving averages
	public	getLongTermMAs()	Return list of calculated long-term moving averages
	public	getIndicators()	Return list of indicators
	public	getStocks()	Return list of available stocks
	public	getHistoricalDataRanges()	Return list of historical data range options
	public	getMovingAverageRanges()	Return list of moving average ranges
	public	getMaxClosingPrice()	Return highest closing price of current stock
	public	updateData(String symbol, int rangeMA, GregorianCalendar start, GregorianCalendar end)	Collects data based on chosen stock, moving average and historical data range and prepares is for display on chart
	Private	updateMAs(int rangeMA)	Updates short and long term moving averages based on currently held data
	Private	updateIndicators()	Updates indicators based on currently held data

6.1.2 Class <StockDataDownloader>

Class Name	StockDataDownloader			
Inherits From	None			
Description	Inner Class charged with acquiring the stock data from Yahoo Finance			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Private	Int	dateIndex	Index for date column
	Public	Int	closeIndex	Index for closing prices column
	Public	Int	adjcloseIndex	Index for adjusted closing prices column
	Private	ArrayList<GregorianCalendar>	dates	List of downloaded dates
	Private	ArrayList<Double>	closes	List of downloadedclosing prices
	Private	ArrayList<Double>	adjCloses	List of downloaded adjusted closing prices
	Private	String	urlString	String representing url of stock data to download
Methods	**Visibility**	**Method Name**		**Description**
	private	StockDataDownloader(String symbol, GregorialCalendar start, GregorianCalendar end)		Constructor

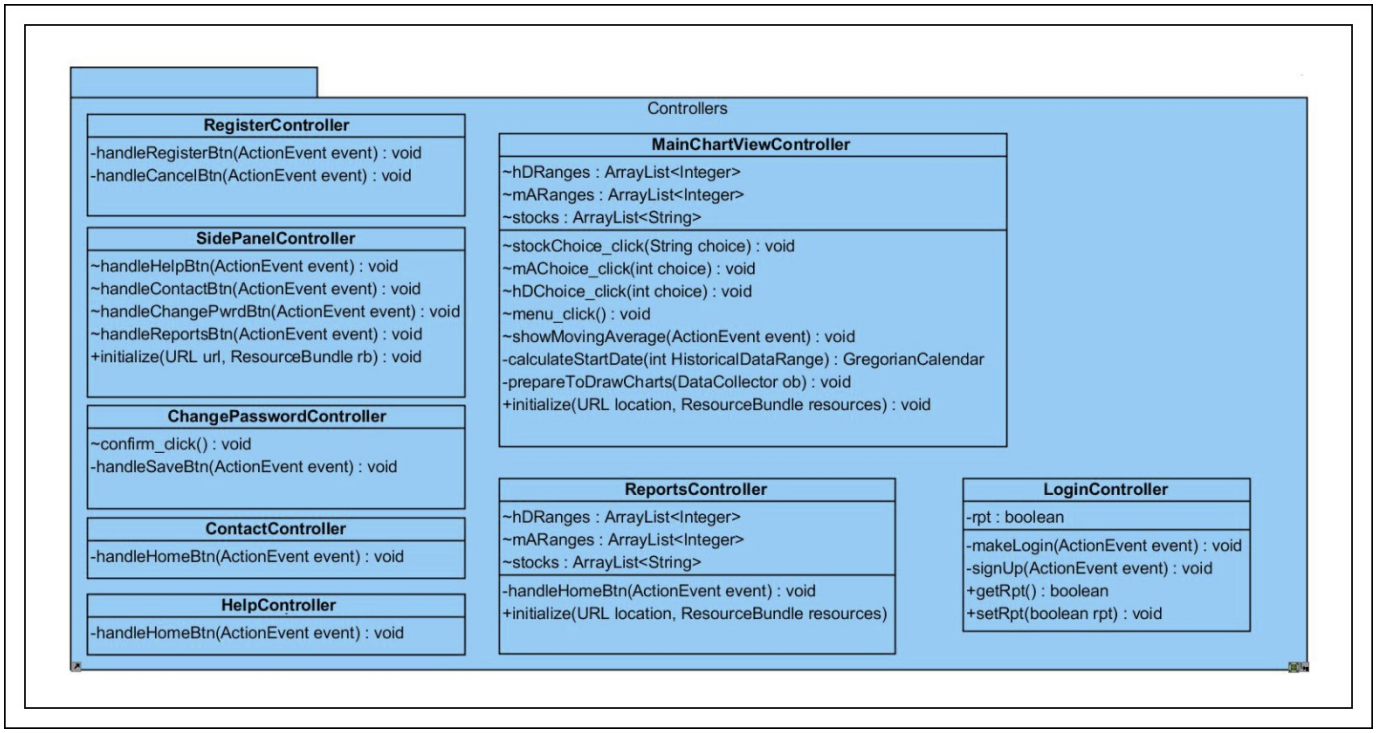
6.1.3 Class <DefaultSettings>

Class Name	DefaultSettings			
Inherits From	None			
Description	Inner Class containing the default values for available stocks, moving average ranges and historical data ranges			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Private	ArrayList<String>	ComponentsOfDow	List of stocks
	Private	ArrayList<Integer>	hDRanges	List of short-term moving average range options
	private	ArrayList<Integer>	mARanges	List of historical data range options
Methods	**Visibility**	**Method Name**	**Description**	
	Private	DefaultSettings()	Constructor	
	Private	AddDowComponentSymbols()	Add stocks that are part of the DOW 30 to the list of stocks	
	Private	AddHDRanges()	Add Historical Data range options to hDRanges	
	private	AddMARanges()	Add Moving Average Range options to mARanges	

6.2. Component < Controller >

The Component Controller is build up from several small controllers responsible for particular tasks. Each class here represents one of those small controllers:

- class RegisterController – handles user account registrations;
- class LoginController – controls the user's Log-in and Log-out;
- MainChartViewController – responsible for the primary tool's functionality on the main user view;
- SlidePanelControler – controls the Side Menu;
- ChangePasswordController – controls request for changing the User Password;
- ContactController – handles requests for displaying Contact Information;
- HelpController – handles requests for displaying the User Guide Manual;
- ReportsController – handles request for user reports



6.2.1 Class <RegisterController>

Class Name	RegisterController			
Inherits From	None			
Description	Handles the user registration window events			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	-	-	-	-
Methods	**Visibility**	**Method Name**	**Description**	
	Private	handleRegisterBtn (ActionEvent event)	Handles register button event. Submits new username and password to create an account.	
	private	handleCancelBtn (ActionEvent event)	Handles cancel button. Returns to the login window.	

6.2.2 Class <SidePanelController>

Class Name	SidePanelController			
Inherits From	None			
Description	Handles the side panel events			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
Methods	**Visibility**	**Method Name**	**Description**	
	Private	handleHelpBtn(ActionEvent event)	Handle help button. Displays usage instructions.	
	Private	handleContactBtn(ActionEvent event)	Handle contact button. Displays developer information.	
	Private	handleChangePwrdBtn (ActionEvent event)	Handle change password button. Brings up the change password window.	
	Private	handleReportstBtn(ActionEvent event)	Handle reports button. Brings user to the Reports page.	
	public	Initialize(URL url, ResourceBundle rb)		

6.2.3 Class <ChangePasswordController>

Class Name	SidePanelController			
Inherits From	None			
Description	Handles the change password window events			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
Methods	**Visibility**	**Method Name**	**Description**	
	Private	HandleSaveBtn(ActionEvent event)	Handle save button. Save user's new password.	
	private	HandleCancelBtn(ActionEvent event)	Handle cancel button. Return to the main chart.	

6.2.4 Class <ContactController>

Class Name	ContactController			
Inherits From	None			
Description	Handles the contact window events			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
Methods	**Visibility**	**Method Name**	**Description**	
	Private	HandleHomeBtn(ActionEvent event)	Handle home button. Return user to main chart page.	

6.2.5 Class <HelpController>

Class Name	SidePanelController			
Inherits From	None			
Description	Handles the help window events			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
Methods	**Visibility**	**Method Name**	**Description**	
	Private	HandleHomeBtn(ActionEvent event)	Handle home button. Return user to main chart page.	

6.2.6 Class <MainChartViewController>

Class Name	MainChartViewlController			
Inherits From	None			
Description	Handles the Main Chart window events			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Private	ArrayList<Integer>	hDRanges	List of historical data range options
	Private	ArrayList<Integer>	mARanges	List of moving average options
	private	ArrayList<String>	stocks	List of available stocks
Methods	**Visibility**	**Method Name**		**Description**
	private	showMovingAverage(ActionEvent event)		Draw chart and display moving averages and indicators.
	private	prepareToDrawCharts(DataCollector ob)		Collect data before drawing charts.

6.2.7 Class <ReportsController>

Class Name	ReportsController			
Inherits From	None			
Description	Handles the Reports window events			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Private	ArrayList<Integer>	hDRanges	List of historical data range options
	Private	ArrayList<Integer>	mARanges	List of moving average options
	private	ArrayList<String>	stocks	List of available stocks
Methods	**Visibility**	**Method Name**		**Description**
	private	handleHomeBtn(ActionEvent event)		Handle home button event. Return user to main chart page.
	public	initialize(URL location, ResourceBundle resources)		

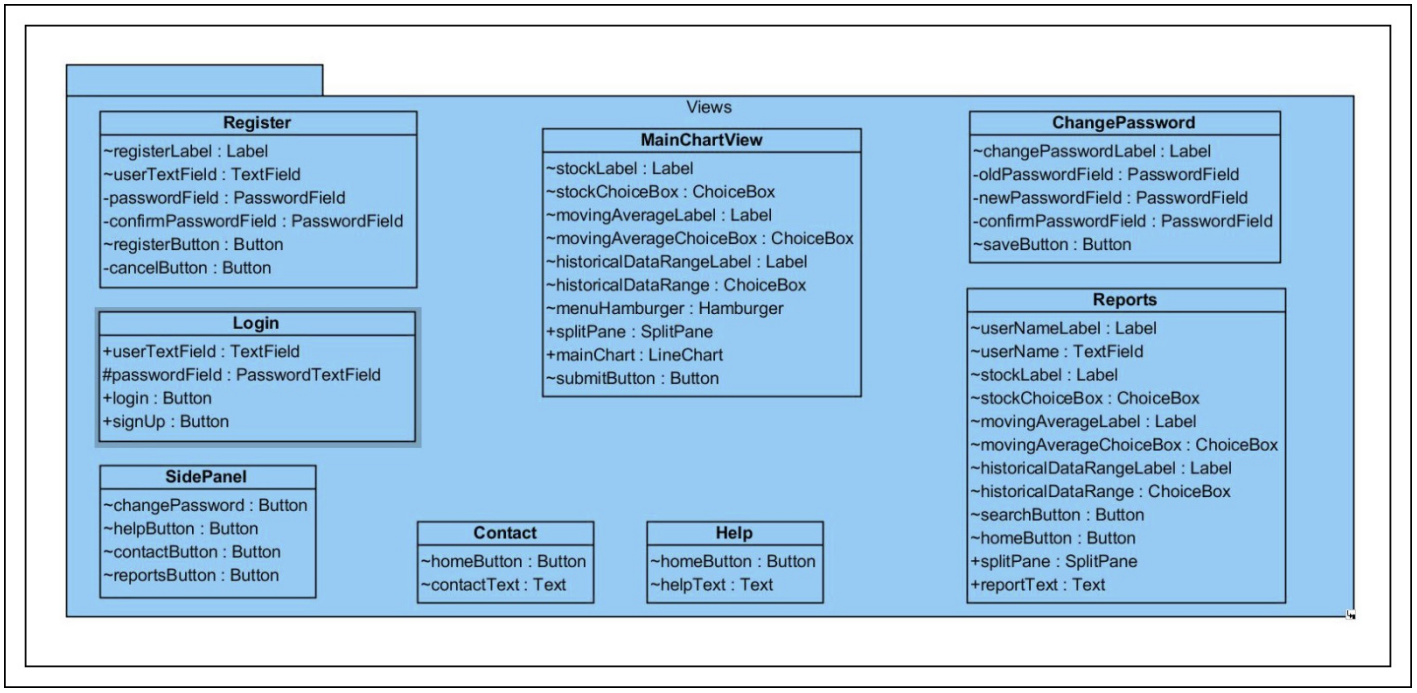
6.2.8 Class <LoginController>

Class Name	LoginController			
Inherits From	None			
Description	Handles the login window events			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Private	boolean	rpt	Allows, if true, or denies, if false, user access to the Reports page
Methods	**Visibility**	**Method Name**		**Description**
	Private	makeLogin(ActionEvent event)		Handles Login button. Allows access to charts if user's credentials are valid.
	private	signUp(ActionEvent event)		Handles sign up button. Brings user to the Register page.
	Public	getRpt()		Get Reports Access Boolean value.
	public	setRpt()		Set access to reports true or false.

6.3. Component < View >

The Component View is build up from several classes each of them responsible for particular part of the user view:

- class Register – handles user account registrations view;
- class Login – handles the user's Log-in and Log-out view;
- MainChartView – responsible for the view of the primary tool's functionality on the main user view;
- SlidePanel – controls the view of the Side Menu;
- ChangePassword – controls the view when changing the User Password;
- Contact – handles Contact Information view;
- Help – handles User Guide Manual view;
- Reports – handles the view for the user reports



6.3.1 Class <Register>

Class Name	Register			
Inherits From	None			
Description	Displays Account Registration window to users.			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Label	registerLabel	Text label.
	private	TextField	userTextField	Username text entry field
	private	PasswordField	passwordField	Password text entry field
	private	PasswordField	confirmPasswordField	Confirm password entry field
	private	Button	registerButton	Register button.
	private	Button	cancelButton	Cancel Button.
Methods	**Visibility**	**Method Name**		**Description**

6.3.2 Class <Login>

Class Name	Login			
Inherits From	None			
Description	Displays Login window to users.			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	TextField	userTextField	Text field for username entry
	Private	PasswordTextField	passwordField	Password field for password entry
	Private	Button	login	Login button
	private	Button	signup	Sign up Button
Methods	**Visibility**	**Method Name**		**Description**

6.3.3 Class <SidePanel>

Class Name	SidePanel			
Inherits From	None			
Description	Displays Side Panel to users.			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Button	changePassword	“Change Password” Button
	private	Button	helpButton	“Help” Button
	private	Button	contactButton	“Contact” Button
	private	Button	reportsButton	“Reports” Button
Methods	**Visibility**	**Method Name**		**Description**

6.3.4 Class <MainChartView>

Class Name	MainChartView			
Inherits From	None			
Description	Displays Main Chart window to users.			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Label	stockLabel	Label for Stock Choice Box
	private	ChoiceBox	stockChoiceBox	Choice Box for stocks
	private	Label	movingAverageLabel	Label for moving average range Choice Box
	private	ChoiceBox	movingAverageChoiceBox	Choice Box for moving average range
	private	Label	historicalDataRangeLabel	Label for historical data range Choice Box
	private	ChoiceBox	historicalDataRange	Choice Box for historical data range
	private	Hamburger	menuHamburger	Clicking opens and closes the side panel.
	Public	SplitPane	splitPane	Split the window into 2: options on top, chart on bottom.
	Public	LineChart	mainChart	Chart displaying stock information
	private	Button	submitButton	Submit Button..
Methods	**Visibility**	**Method Name**		**Description**

6.3.5 Class <Contact>

Class Name	Contact			
Inherits From	None			
Description	Displays Contact window to users.			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Private	Button	homeButton	“Home” Button
	private	Text	contactText	Text content of the contact page.
Methods	**Visibility**	**Method Name**		**Description**

6.3.6 Class <Help>

Class Name	Help			
Inherits From	None			
Description	Displays Help window to users.			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	Private	Button	homeButton	“Home” Button
	private	Text	helpText	Text content of the Help page.
Methods	**Visibility**	**Method Name**		**Description**

6.3.7 Class <ChangePassword>

Class Name	ChangePassword			
Inherits From	None			
Description	Displays Change Password window to users.			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Label	changePasswordLabel	Text Label
	private	PasswordField	oldPasswordField	Old password text entry field
	private	PasswordField	newPasswordField	New Password text entry field
	private	PasswordField	confirmPasswordField	Confirm password entry field
	private	Button	registerButton	Register button.
Methods	**Visibility**	**Method Name**		**Description**

6.3.8 Class <Reports>

Class Name	MainChartView			
Inherits From	None			
Description	Displays Main Chart window to users.			
Attributes	**Visibility**	**Data Type**	**Name**	**Description**
	private	Label	userNameLabel	Label for Username text field
	private	TextField	userName	Text Field for username search
	private	Label	stockLabel	Label for stock Choice Box
	private	ChoiceBox	stockChoiceBox	Choice Box for stock
	private	Label	movingAverageLabel	Label for moving average range Choice Box
	private	ChoiceBox	movingAverageChoiceBox	Choice Box for moving average range
	private	Label	historicalDataRangeLabel	Label for historical data range Choice Box
	private	ChoiceBox	historicalDataRange	Choice Box for historical data range
	private	Button	searchButton	“Search” Button
	private	homeButton	submitButton	“Home” Button
	Public	SplitPane	splitPane	Split Screen in two sections: top section contains choice boxes for options, bottom contains reports.
	public	Text	reportText	Text content for reports page
Methods	**Visibility**	**Method Name**		**Description**

7. Other Nonfunctional Requirements

7.1 Performance Requirements

The performance of browsing and displaying is measured by response time and is influenced by user's perception. The functions and features of the application are not computationally intensive. Displaying the graphs should not take more than a few seconds. The application does not require specific processor characteristics, RAM or disk space.

7.2 Safety Requirements

Maester Buy/Sell will not affect data stored outside of its servers nor any other applications installed on user's personal computer. The application cannot cause damage to the user's personal computer.

7.3 Security Requirements

The system will be storing user's information. This information must be kept private from the outside world. The system will implement authentication via a secure login screen. All users who attempt to access the system will be identified and authenticated. The access to the server and users accounts is restricted to authorized personnel. The system will detect attempted intrusions by unauthorized persons and client applications. The system will be able to provide full functionality from backup in case of attack or failure.

7.4 Software Quality Attributes

Usability: The usability is the first priority of the GUI. The application is easy to handle and navigates in the most expected way with no delays.

Correctness: The system will provide reliable information and calculations to the user.

Security: The system will prevent malicious or accidental actions outside of the designed usage, disclosure or loss of information, unauthorized modification of information.

8. Other Requirements

Appendix A: Glossary

Terms:

Authorization: the process of determining whether a subject is allowed to have the specified types of access to a particular resource. Usually, authorization is in the context of authentication. An authenticated user is authorized to perform different types of access.

Chart: a graphical representation of a series of prices over a set time frame.

Closing Price*:* the final price a particular stock closes at on a given trading day.

Database: a structured set of related data organized for convenient access.

DOW 30: a stock market index that shows how 30 large publicly owned companies based in the United States have traded during a standard trading session in the stock market.

Graphical User Interface: the space where interaction between users and computers occur.

Indicators: price based calculations that measure money flow, trends, volatility and momentum. They are used as a secondary measure to the actual price movements. Their purpose is to add additional information to the analysis of securities, to confirm price movement and the quality of chart patterns, and to form buy and sell signals.

Moving average: the average price of a security over a set amount of time.

Opening price: the price at which a stock first trades upon the opening of an exchange at a particular trading day.

Simple Moving Average: the most common method used to calculate the moving average of prices. It takes the sum of all of the past closing prices over the time period (range) and divides the result by the number of prices used in the calculation.

Technical analysis: focus on charts of price movement and various analytical tools to identify trends and forecast future price changes.

Trading day: the duration of time the stock market is open for buying/selling stocks.

Trendline: a charting technique that adds a line to a chart to represent the trend in the market/stock.

Abbreviations:

COCOMO: Constructive Cost Model
GUI: Graphical User Interface
KLOC: 1,000 Lines of Code
OS: Operating System
SRS: Software Requirements Specification
URL: Uniform Resource Locator (web address)