**Market Emulator – Client Application – LLD for Milestone 3**

**Introduction:**

The project supposed to serve users who wants to perform trading via network, like selling stocks of commodities and buying stocks of commodities. The application adjusted for the frequent changes in the market and enables the user to know about the changes. The application supports manual marketing (when the user performed deals) , and automatically marketing, via the AMA (auto market agent) , that performs deals with an algorithm that considering the status of the market (bid and asked price for each commodity) , the user's funds and the number of each commodity. The marketing is done by communication with sever that gets valid requests, and performed the wanted commands.

In the next lines, we will focus in the functionality of every layer and class in the application:

**Presentation Layer**

**Presentaion Class**

A class with one method ( MainLoop() ) that responsible for starting the main loop of the communication module which runs the methods of AsciSilverTongue Class.

**AsciiSilverTongue Class**

This class responsible for display of the menu for the client with all the possible methods that a user can perform.

Methods:

public AsciiSilverTongue()- Default Constructor

public void PrintWelcome()- Just a title for designing the home page.

public void PrintGoodbye() – Print a message for exiting from the system.

public void PrintMenu()- Show a menu for the visitors with the possible using system options.

public static void printServerData(Object obj)- Prints information from the server.

public string ReadLine()-Returns the input of the user with the wanted request.

**Parser Class**

This class has one method (parse(String str)) that responsible for ensuring that the client entered valid comment and sent the input to the Socket Class in Business Layer, else- the method prints an error message.

**Business Layer**

**Socket Class**

The class responsible for checking typing of request and its parameters. If the input didn't pass the check- error message is printed, else- the input is converted to understandable data for the server.

Methods:

public Socket()- a constructor of the class.

public static void printNoValidCommandError() – prints an exception when the user types a wrong input.

public static int generalStringToInt(String str,int errorVal, string errorMsg) –convert an input from string to int.

public static int idStringToInt(String str,int errorVal, string errorMsg)-Responsible for converting input of id for requests from string to int.

public static void sell(String str) – This method gets an input of the required commodity for selling, and the method commits sell activity.

public static void buy(String str) – This method gets an input of the required commodity for purchasing, and the method buy the wanted product.

public static void cancel(String str) – This method gets an id, and tries to perform cancel activity

public void cancelAll()- This method asks from the server user's request and cancel requests in pending.

public void runAutoMarketAgent()- This method starts the auto market agent.

public static void findInfo(String str)- The method interprets the input and redirect to the suit query. If there is an incorrect input, error message will be sent.

public void allHistory()- The method asked for information of all buy and sell requests and their status.

public void allMarketRequest()- The method asked for information of all the commodities in the application (id, asked price, bid price).

public void userRequestsInfo()- The method asked for information of all the active requests.

public static bool userInfo() – The method returns details about the current client.

**Data Access Layer**

**Statistic Class**

This class uses DataBaseSocket class that prevents tables with information from history server, and analyzes it to helpful information. The GUI and Auto market agent use this information to display it to the user, and making finance decisions respectively.

public static double CalcAvgCommPriceByLastNTrades(int commID, int n)- gets an ID of commodity and calculates the average price of the last n trades of this commodity.

public static double CalcAvgCommPrice(Transaction[] transactions)- gets an array of transactions and returns the average price of those transactions.

public static int GetLeastTradedComm(int numOfTrades)- gets number of trades and returns tha least traded commodity id in the last "numOfTrades" trades.

public static int GetLeastTradedComm(int[,] marketShare)- gets tables that describes commodityID and amount of traded units per commodity. The method returns the least traded commodity id.

public static int GetMostTradedComm(int numOfTrades, int k)-

?????? What is K??!

public static int GetMostTradedComm(int[,] marketShare)- gets tables that describes commodityID and amount of traded units per commodity. The method returns the most traded commodity id.

public static int GetKthTradedComm(int numOfTrades, int k)- returns the k-th traded commodity in the last "numOfTrades" trades. If k is out of array's bound the method will return -1.

public static int GetKthTradedComm(int[,] marketShare, int k)- gets tables that describes commodityID and amount of traded units per commodity. The method returns the k-th traded commodity id.

**DatabaseSocket Class**

This class extracts information from the server and prevents it to some classes in the project, like Statistics class for analyzing and GUI for display it to the user.

Some methods contains the parameter "justOurs". If this parameter is true, the output will refer only to our user's data, else- all relevant and available data will be returned.

public static DataTable GetAllHistory(bool justOurs)- returns data-table object with all trades that performed since history DB has created.

public static DataTable GetHistoryOfLastNTrades(int n,bool justOurs)- return the last n trades in data-table object.

public static DataTable GetHistoryOfLastDay(bool justOurs)- returns the trades that performed in the last day.

public static DataTable GetHistoryOfLastHour(bool justOurs)- returns the trades that performed in the last hour.

public static int[,] getMarketShare(int n)- returns the market share of the last n trades.

public static int[,] getHistoryBetweenTwoDates(DateTime start,DateTime end, bool justOurs)- returns all requests between 2 dates when justOurs=false, else the method returns only our history.

public static Transaction[] getOurLastHistory()- returns array of transactions with our user's last 50 transactions.

public static Transaction[] getPriceOfCommFromStartDate(int commodityID, DateTime start)-returns array of transaction that displays the

|  |
| --- |
| oscillation of specific commodity's price from start date (which are given as parameters) to the last transaction. |

public static Transaction[] getPriceOfCommByLastNTrades(int commID, int n)- returns array of transaction that displays the

|  |
| --- |
| oscillation of specific commodity's price in the last n transactions (which are given as parameters) to the last performed transaction. |

private static int[,] Sanitize(int[,] matrix)- a private method which gets a table of integers that represents the market share and sanitize commodities that were not traded.

**Transaction**

This object represents a transaction, with all relevant details. This object supposed to help the AutoMarketAgent class extracting details in more comfortable way. The transaction object contains timestamp, commodity id, price and amount fields, that represent the relevant data to the AMA.

Transaction's constructors:

public Transaction(DateTime timestamp,int commodityID,int price,int amount)

public Transaction(DateTime timestamp,int price)

public Transaction(int commodityID,int amount,int price)

Transaction's getters:

public DateTime getTimestamp()

public int getPrice()

public int getAmount()

public int getCommodityID()

Transaction's ToString:

public override string ToString()

**MarketClient Class**

This class contains all the methods that responsible for communication with the server

public MarketClient()-initializes new object which contains details of client, and information about his authentication.

private void printError(string error)- prints an error.

private string SendRequest<T>(T data)-

private T2 SendRequest<T1, T2>(T1 data) where T2 : class- This method sends request to the server with Object with type of T1, and receives from the server an object with type of T2 with the relevant information.

public int SendBuyRequest(int price, int commodity, int amount)- Sending a buy request and returns the answer from the server.

public int SendSellRequest(int price, int commodity, int amount)- Sending a buy request and returns the answer from the server.

public IMarketItemQuery SendQueryBuySellRequest(int id)- This method asks from the server to get information about deal number id (The deal can be sell or buy request).

public IMarketUserData SendQueryUserRequest()- The method asks from the server information about current user.

public IMarketCommodityOffer SendQueryMarketRequest(int commodity)- The method asks from the server information about received commodity.

public bool SendCancelBuySellRequest(int id)- The method sends a request to the server for cancelling deal number id.

public QueryUserRequest[] sendQueryUserRequestsRequest()- The method asks from the server to send information about active requests.

public Commodity[] sendQueryAllMarketRequest()- The method asked from the server to get an array of the details of all the commodities in the application (bid price, ask price, id).

* This class implements IMarketClient interface.

The next classes represent all the kinds of the possible queries:

#### **SellRequest Class**

The class has constructor that initialize a sell request with given commodity-id, amount and price

#### **QueryUserRequest Class**

The class has 2 methods: toString method that responsible for showing details about request <id>, and constructor that initialize request with kind of "queryUser".

#### **QueryMarketRequest Class**

The class has constructor that gets id of commodity from and creates an object which describes a market query.

#### **QueryBuySellRequest Class**

The constructor gets id initializes an object which describes a buysell request <id>.

#### **CancelBuySellRequest Class**

The class has constructor that initializes a new object of cancel request with a given id of the order.

#### **BuyRequest Class**

The class has constructor that initialize a buy request with given commodity-id, amount and price

#### **MarketCommodityOffer Class**

public override string ToString()- The method prints information about the commodity's offers which received from the server.

* This class implements IMarketCommodityOffer interface.

#### **QueryUserRequest Class**

#### This class has a constructor that initializes a new object which describes the type of the commodity.

#### **QueryBuySellRequest Class**

#### The class has constructor that initializes a new object which describes details about an order. This object is supposed to be sent to the server.

**MarketUserData Class**

public override string ToString()- This method initializes a new object with received data from the server.

* This class implements IMarketUserData interface.

**MarketItemQuery Class**

public override string ToString()- The method prints received data from the server about the items.

* This class implements IMarketItemQuery interface.

**QueryUserRequestsRequest Class**

#### The class has constructor that initializes a new object which describes query with type of "queryUserRequests" .

**QueryAllMarketRequest Class**

#### The class has constructor that initializes a new object which describes query with type of "queryAllMarket" .

#### **RequestBase Class**

#### The class holds the data that necessary for every communication with the server, including username, token, url, and the private key.

#### **Commodity Class**

#### This class holds id of request and the information of the commodity which given from the server, using MarketCommodityOffer class and the amount the user has.

**Request Timer Class**This class handles the last 20 requests the client sent to the server, and makes sure the server is not spammed by limiting the client to max 20 requests per 10 seconds. Every request sent to the server is saved by the class for the last 10 seconds and afterwards expires and can be replaced by a new request.

#### The next 2 classes describe data structure for saving the history of sell and buy requests of user, and their status:

**History Items**

HistoryItem supposed to describe details about previous sell and buy requests. The saved details are the date when the request was performed, type of the request (sell/buy), information, status (pending, completed, cancelled) , and the id of the request. The class has also constructor that initialize historyItem with given date, type, information and id.

**HistoryTable**

This class describes a list of requests. We implemented it by LinkedList of HistoryItems.

In this class we can print the history, add a new event to the end of the list, and update a status of request (If a request cancelled, or accepted).

**Log**

**Logger Class**

This class is static because we want that one instance will write to a text file, and will document errors in the text file (if there are) , and messages about processes in the application. The class has 3 methods. The first write error messages to a text file, and shows where the error occurred, when it happened and the reason. The rest 2 methods document info and debug messages in the log file. The use of this class throughout the program is done by using those methods in some critic points in the code.

**AutoMarketAgent**

**Terminology**

Ask to Bid Ratio - The ratio between the commodity Ask price to it’s Bid price - ask/bid.

**Functions**

public void autoPilot() - this is the main function of the AMA which is main trading loop which calls the main trade functions in a call order which is based on the current state of the user.

private void updateCommodities()- the method updates our commodities.

private void updateUserData()- the methods updates user datal, commodities , funds, and his active requests.

private void upateAmount()- updates number of each commodity.

private void trade()-

private void TradeComm(Commodity comm,int n)

private void TradeMostTradedComm(int[,] marketShare)- gets a table of commodities and the amount of traded price and trades the most traded.

private int getTotalAmountOfComms()-

private double getAvgCommAmount()- calculates the average amount of traded commodities.

private bool isRequestSuccessful(int id)- the method searches for trade id. If the id found, the trade has succeeded, so the method returns true, else-false.

private Commodity getLowestAskComm()- the method goes through the list and returns the commodity with the low ask price.

private Commodity getHighestAskComm()-the method goes through the list and returns the commodity with the high ask price.

private Commodity getLowestBidComm()-the method goes through the list and returns the commodity with the low bid price.

private Commodity getLowestAskToBidRatioComm()-the method goes through the list and returns the commodity with the low ask price.

private Commodity getHighestAskToBidRatioComm()-the method goes through the list and returns the commodity with the high ratio of ask to bid price.

private Commodity[] getSortedBy(Func<Commodity, double> selector)

private Commodity[] getSortedByDescending(Func<Commodity, double> selector)

private Commodity[] getAskLowerThanBid()-the method goes through the list and returns the commodity with the

private Commodity[] getBidLowerThanAsk()-the method goes through the list and returns the commodities with bid price lower then ask price.

private Commodity[] getBidLowerEqualToAsk()--the method goes through the list and returns the commodities with bid price equals to ask price.

private void printMatrix(int[,] matrix)

#### private double getAvgAskToBid() - Calculates the average Ask to Bid ratio.

private void raiseCommAvg() - Raises the commodity average by buying ideal commodities which their amount is below the average.

private int checkAmountToBuy(Commodity comm) - calculates the amount of a given commodity type which the AMA needs to buy

private int checkAmountToSell(Commodity comm) - calulates the amount of a given commodity type which the AMA needs to sell

private bool isRequestSuccessful(int id) - checks if the trade with the given id was successful

private void wait() - checks if there is an available request slot and if not waits until the oldest request expires.

private void wait(int n) - checks if there is n available request slots and if not, waits until the n oldest requests expire.

private void waitAll() - checks if all the request slots are available, if not waits until they become available.

public void n00bTrade() - goes over all the commodities in the market and buys every commodity with an ask price lower the the bid price - then sells it in the bid price in order to earn funds equal to the difference between the two (ask - bid).

private void sell() - sells a certain amount from each commodity type

private void buy() - buys all the commodities with an ask price lower than the bid price, then buys the commodity with the best ratio greater than 1. (This is the ratio which is closest to 1 buy not smaller.)

private void buy(int budget) - this function does the same as the buy function above over and over again until running out of the given budget.

private void buyAskLowerThanBid() - buys all the commodity type with an Ask price lower than the Bid price.

private void sellAskLowerThanBid() - sell the commodity types with an Ask price lower than the Bid price.

**GUI (Graphical User Interface)**

**Terminology**

User Control - A section in a window (a “sub-window”) that can be changed without changing to a new window. We used user control for UX.

AMA - Short for Automatic Market Agent.

**Functions**

**Main Window (Startup Window)**

This window asks whether to open the program with the GUI or console (“Legacy”).

ButtonEnter\_Click - called when the GUI is picked. Closes the current window and opens a new window which will contain the entire GUI part of the program.

ButtonOpenConsole\_Click - called when the Legacy program (the console) is picked. Closes the current window and opens the console with Kernel32.dll’s AllocConsole function (and frees it later with FreeConsole).

**AlgoTratingWindow (GUI program window)**

InitAma - Initialises the Auto Market Agent’s necessary settings.

ExitProgram - Exit the program.

AlgoTradingWindow\_OnClosing - called when the Windows ‘X’ button (exit) is clicked. Asks the user if he’s sure he’d like to exit. If so - it calls ExitProgram.

buttonInformation\_Click,

buttonBuy\_Click,

buttonSell\_Click,

buttonSearch\_Click,

buttonSessionHistory\_Click,

buttonActiveRequests\_Click,

buttonMarketStatus\_Click - Called when navigation buttons in the Menu are clicked. These methods change the window’s content user control to the corresponding user control relevant to the asked activity.

buttonAMA\_Click - Called when the AMA button in the Menu is called. Turns the AMA on if it’s off, and off if it’s on.

ToggleAMA - Handles turning the AMA on and off on a different thread.

**UserControlActiveRequests**

GenerateColumns - Generate the base columns for the Data Grid.

PopulateGrid - Populates the Data Grid with the relevant data.

CancelRequest - Sends a cancel request when a user clicked the “Cancel” button.

**UserControlBuySell**

buttonBuySell\_Click - Checks if the user’s input in the buy/sell form is legal. If so, the function shoots a buy/sell request to the server. (Choosing whether to buy or sell using a radio button input from the user).

**UserControlInfo**

PresentData - Called on User Control load. Populates the user control with information about his requests, commodities, funds, etc.

**UserControlMarketStatus**

GenerateColumns - Generate the base columns for the Data Grid.

PopulateGrid - Populates the Data Grid with the relevant data.

**UserControlQuery**

comboBoxType\_SelectionChanged - Called when the comboBox selection is changed. Changes the input form according to the selection.

BuySell - If the user selected to search buy or sell requests, query the search term and present the output to the user.

Commodity - If the user selected to search commodity, query the search term and present the output to the user.

buttonSearch\_Click - Check the user input and determine which category to search.

**UserControlSell**

buttonSell\_Click - Checks if the user’s input in the sell form is legal. If so, the function shoots a sell request to the server.

**UserControlSessionHistory**

PopulateGrid - Populates the Data Grid with the session’s history.

**UserControlWelcome**

This user control has no functions. It only welcomes the user upon entering the program.

**UserControlAMAWorking**

This user control has no functions. It holds the user as the AMA is working.

**UserControlStatistics**

Holds tabs that contain other User Controls, the statistics tabs. Within this User Control, the user can easily switch and view other statistics.

**UserControlStatisticsTabCommodityRates**

Shows the user statistics about rates (The ratio between ask and bid for every commodity).

private void PopulateChart(Transaction[] data): Fills the charts/graphs with the relevant information given the user’s wills.

private void comboBoxRange\_SelectionChanged(object sender, SelectionChangedEventArgs e): Called upon changing the comboBox value, changes the information options.

**UserControlStatisticsTabMarketShare**

Shows the user statistics about market shares (What percentage of the market trades belonged to every commodity).

private void PopulateChart(Transaction[] data): Fills the charts/graphs with the relevant information given the user’s wills.

private void comboBoxRange\_SelectionChanged(object sender, SelectionChangedEventArgs e): Called upon changing the comboBox value, changes the information options.

**UserControlStatisticsTabHighlights**

Shows the user statistics about highlighted events – the largest trade, etc.

private void PopulateChart(): Fetches information and presents the data to the user.

**Test**

This project supposed to check our code and make insure that it stable for all inputs and situations. Test's project is project of one class which contains some methods which check all above.

TestsToProject Class-Methods

public static void TestStatistics()

public static void TestGetMostTradedComm()

public static void TestGetLeastMostTradedComm()

public static void TestCalcAvgPrice()

public static void TestCalcAvgCommPriceByLastNTrades()

**Recently Changes for This Milestone**

GUI and AMA support direct using of "history" Database, by using 2 helpful classes that created in the third milestone- DatabaseSocket, and Statistics that prevent an actual information for the AMA. This information helps the AMA to upgrade his decisions. The information is also actual to the GUI, because variety of statistics and information will be shown to the user. In order to support all above, another functions were added to the classes.

In addition, changes added in the communication Mechanism, according to the instructions, which insure us a more safe sending information to the server.