Algo-trading market emulator LLD

For terminology and other relevant information please refer to the server LLD document.

<https://goo.gl/h5MyZu>

The project id delegated to three sub-projects(layers) containing the following:

**Presentation layer:**

This layer contains the GUI Interface and it's Window Forms.

**GUI Buttons and Windows:**

**Main Menu Window:**

Buttons:

Buy Commodities – opens a new window for buying commodities.

Sell Commodities – opens a new window for selling commodities.

Cancel Request – opens a new window for cancelling a buy / sell request.

Help – opens a Help window.

History – opens the History window, shows the History when clicking "Show History" button.

Commodity Query – opens the Commodity Query window.

Buy / Sell Query – opens the Buy/Sell Query window.

User Query , All Market Query , User Requests Query – after pressing one of these buttons a message box will appear with the requested information.

AMA On / AMA Off – turns the AMA on or off

Market Statistics – opens a Statistics menu window

Generate PDF Report – creates an instance of PDFGenerator class with an adequate string, in order to save a report on a PDF file in Reports directory

**Buy / Sell Commodities Windows:**

In this windows the user is able to send a Buy / Sell request to the server.

There are 3 numeric input fields: Commodity ID, Amount , Price.

There are 2 Buttons: Cancel – Exits the window.

Apply – Takes the numbers in the input fields and sends a Buy/ Sell Request to the server(through the BL). After that a Message Box will appear and tell the user the request id if successful, or the error otherwise.

**Cancel Request Window:**

Has one numeric input field for the request ID, sends a cancel request when clicking the Apply button. Returns a Message Box with the response.

**Buy / Sell Query , Commodity Query Windows:**

These windows have one numeric field( commodity ID for commodity, Request ID for query buy/sell). Clicking the apply button will send a query request to the server and show a Message box with the requested information ( or an error that occurred otherwise).

**Statistics Menu:**

In this window the user is able to choose from 3 different statistics about the market's commodities – Max prices, Min Prices and Average prices.

The user chooses the Stat type and a start date and end date. Then, using the SQL server data and LiveCharts open source, a chart is generated and displayed , with the adequate data.

**Business layer:**

This class holds the History and AMA classes, and besides that this layer is used as the "Middle Man" between the GUI and the Data Access Layer. It receives input from the user for all the requests and sends the requests to the DAL. After that it receives the response from the DAL, processes it and returns an adequate response to the GUI (if there was an error with the request, it returns an error string).

**Classes**

**Request Agent Class** – this class operates as the "Middle Man" between the GUI and the DAL. All the functions in this class return a string back to the GUI.

**Functions -** the functions in this class are very simple; They simply take the input coming from the user about a certain request, send it to the DAL and send back a response string to the GUI. Each functions writes to the History log.

string buyCommodities(int price, int commodity, int amount)

string sellCommodities(int price, int commodity, int amount)

string cancelRequest(int id)

string QueryBuySell(int id)

string CommodityQuery(int id)

string UserQuery()

string AllMarketQuery()

string UserRequestsQuery()

**Autonomous Market Agent** – this class responsible to connect with the server automatically while the user decided to use it instead of using the GUI interface. It is using a timer which allowed us to make 20 action every 10 seconds. Any action such as apply to the server, get commodity list and buy/sell requests accordingly to the rules we decided to use when we make buy or sell request. The process is first to get the information if the stock is on ascent or descent. If it's on ascent, then we want to sell it. The next step is to calculate the relation between the current bid to the average of the stock in the 20 deals, and with the relation we get, we will sell amount of stocks accordingly to the number we get. So the main propose is while the relation is higher we will sell more stocks.

And if we know that the stock is on descent, we will try to buy the commodity. Here we will calculate the relation between the ask price to the average of the last 20 deals, and then we will compare this to some other numbers, and the main propose is while the relation is lower we will buy more stocks. But before we will divide our funds to 10, so each stock will have the same price to waste.

**Functions *–***

OnTimedEvent(Object source, ElapsedEventArgs e) – the main function who make all the communication with the server automatically.

Start () – we use this function from the GUI interface to start the automatic action.

Stop () – we use this function in the GUI interface to stop the automatic action

setAverage(int commodity) - the function get some commodity and by using the Sql server she calculate the average of the 20 last deals that was made with the current stock.

public Boolean isUpOrDown(int commodity) – the function get sone commodity and returns us Boolean variable which give us information whether in the last 20 deals from the last 5 days the stock is on ascent or descent.

**StatsManager** – this class is responsible to get the statistics from the SQL server. It creates an SQL connection using the SQLManager class in the DAL, then gets the requested stats by the MaxMinPrices function, or the AvgPrices Function.

**Functions –**

public Double[] MaxMinPrices(Boolean max, DateTime start, DateTime end) –

this function returns the Max/Min prices of all of the commodities

transactions between start and end times.

public Double[] AvgPrices(DateTime start, DateTime end) –

this function returns the Average prices of all of the commodities

transactions between start and end times.

**PDFGenerator –** this class simply creates a pdf from a given string and saves it in Reports folder in the project directory. It uses ITextSharp open source for it.

**Data Access layer:**

This class contains all the communication with the server. All the requests coming from the user through the BL are processed here, sent to the server and then this layer returns an adequate response back to the BL.

**Classes**

**Request Manager Class** – this class operates as the main class of the DAL. This class contains all the functions communicating with the server with the adequate inputs as given from the Presentation Layer. This class implements the IMarketClient interface as given in the MarketClient solution. Any of the function will use an unique nonce that will communicate with the server in decrypt way so no other request will use the same one.

**Functions**

Each of these functions creates an instance of SimpleHTTPClient, and use that instance in order to use the SendPostRequest function, which is used in order to translate the requests and send them to the server.Each function writes to the Log.

SendCancelBuySellRequest(int id) –

SendBuyRequest(int price, int commodity, int amount) -

SendSellRequest(int price, int commodity, int amount)

SendQueryBuySellRequest(int id)

SendQueryUserRequest()

SendQueryMarketRequest(int commodity)

SendAllMarketQuery()

SendUserRequestsQuery()

**Requests Classes**

These classes include no functions, only appropriate fields (according to the input needed of the function) for creating requests instances.

BuyRequest Class

SellRequest Class

CancelRequest Class

QueryMarketRequest Class

QuerySellBuyRequest Class

AllMarketRequest Class

UserRequestsQuery Class

**Query output Classes**

These classes include one function – ToString(), and appropriate fields (according to the output received from the server) . these Classes implement the interfaces as given in DataEntries folder in the MarketClient solution.

MarketCommodityOffer Class

MarketItemQuery

MarketUserData

AllCommodityOffer

MarketUserRequests