**Algo-Trading client application LLD**

**Presentation layer (user gui):**

The presentation layer communicates with the user by using a gui, which allowing the following options for the user:

1. Buying\Selling- asks the user to enter the requirements for buying\selling a commodity: commodity number (int), price (int) and amount (int).

The function sends the request with the matching parameters to the server through MarketClient by using SendBuyRequest\SendSellRequest method.

1. Canceling- asks the user to enter a transaction ID (int) which he\she wish to cancel. The function sends the request to the server through MarketClient by using SendCancelBuySellRequest method.
2. Queries-

* Buy\Sell- asks the user to enter a transaction id number for returning information about this transaction. The function sends the request to the server through MarketClient by using SendQueryBuySellRequest.
* User- sends a request for user information through MarketClient by using SendQueryUserRequest, and returns it to the user.
* Market- asks the user to enter a commodity number and sends a request for stock information through MarketClient by using SendQueryUserRequest, and returns it to the user.
* allMarket- send a request through the MarketClient by using SendQueryAllMarketRequest() , and returns the ask and bid for each stock in the market.
* User requests- returns the status for all of the open requests of the user by sending a request via SendQueryUserRequestsRequest().
  + Export to pdf- export the user status to a PDF file by using an open source “itextsharp.dll”

1. Cancelling all requests- allows the user to cancel all his transactions (sell and buy). The function sends the server a request to get the user's info by using SendQueryUserRequest, and once it has it- the function goes through all the user's transactions and cancelling them by using the method SendCancelBuySellRequest.
2. History- shows the user all of the transactions history by request via the cancel log or by sending a query to the history data base. (the user chose between showing buying/selling/canceling history file.
3. Semi auto pilot- Operating a semi-auto pilot for buying/selling. the function asks the user to enter a commodity number (int), a min price (for selling) or max price (for buying) (int), and amount for buying (int) and calls the semi pilot class to operate the request.
4. Auto pilot- Operating an auto pilot that runs autonomously and buying and selling stocks, and showing the data of the transactions it made.

**Logic layer:**

The logic layer gets the parameters from the presentation layer, and sends them to the matching function in MarketClient project, using the methods in it.

**LogicLayer class funnctions:**

1. int SendBuyRequest(int price, int commodity, int amount)- sends the server the given parameters and returns the buying request's ID. If the request can't be executed, the function will write on the screen the reason, and will return -
2. int SendSellRequest(int price, int commodity, int amount)- send the server the given parameters and returns the selling request's ID. If the request can't be executed, the function will write on the screen the reason, and will return -1.
3. bool SendCancelBuySellRequest(int id)- sends the server a request to cancel transaction number "id". The function returns true if the transaction canceled successfully, and if it doesn't- writes on the screen the reason and returns false.
4. IMarketItemQuery SendQueryBuySellRequest(int id)- sends the server the given ID and returns an instance of IMarketItemQuery, which contains the data about transaction number "id" (commodity number, price and amount). If there is no such transaction, it will write on the screen a matching message and will return null.
5. IMarketUserData SendQueryUserRequest()- sends the server a request to get user's info and returns and instance of IMarketUserData, which contains the user's info (commodity number, price and amount). If it couldn't be happened, the function will write on the screen the reason, and will return null.
6. IMarketCommodityOffer SendQueryMarketRequest(int commodity)- sends the server the given commodity number and returns an instance of IMarketCommodityOffer, which contains the data about commodity number "commodity" (ask price and bid price). If there is no such commodity, it will write on the screen a matching message and will return null.
7. LinkedList <Commodities> SendQueryAllMarketRequest ()-

sends the server a request to get data for all of the possible commodities and returns LinkedList of the instance: commodities - which contains stock id number, and an MarketCommodityOffer that describe the stock id.

1. LinkedList <UserRequests> SendQueryUserRequestsRequest ()

sends the server a request to get data for all of the open transactions requests and returns LinkedList of the instance: userRequests - which contains transaction id number, and an MarketItemQuery that describe the transaction id.

1. IQueryable<item> getBuyHistory() – send query to the server for all buy history of user52
2. IQueryable<item> getSellHistory()- send query to the server for all sell history of user52
3. IQueryable<item> getBuyHistoryByDate(DateTime start, DateTime end) - send query to the server for all buy history of user52 that occuerd between the date of start and end
4. IQueryable<item> getSellHistoryByDate(DateTime start, DateTime end) - send query to the server for all sell history of user52 that occuerd between the date of start and end

All of the functions returns a matching "item" from MarketItems class to send to the server and fills it with the right parameters. The query functions create also an object, which is an instance of the interface they return.

history class functions (part of logic layer):

this class has some function (queries) that serve the auto pilot

1. IQueryable<float> getLastHourCommodityHistoryOrderedByDate(int commodity)- query to the history data base that return last hour commodity history ordered by date
2. float getTodaysRecommendedBuyPrice(int commodity)- query to the history data base that return today recommended buy price of commodity
3. float getTodaysRecommendedSellPrice(int commodity)- query to the history data base that return today recommended sell price of commodity

**Pilots class functions:**

The millstone 2 auto pilot:

1. Void runSemiPilot (int id , int price , int amount , bool requestKind) –

semi-auto pilot gets the data from the presentation layer and checking the market every second, if the price of the stock fits the user conditions it make the transaction. The semi auto pilot stops or when making all of the transactions or by a stop request of the presentation layer.

The millstone 2 auto pilot:

1. void runPilot ()-

the auto pilot gets the command to start from the presentation layer, and checking the stocks in the market by calling SendQueryAllMarketRequest(). After that running on the stocks and looking for a stock that it bid price is higher then the ask, then it buys the stock and sell it on place. The auto pilot keep with this process until it gets the command to stop from the presentation layer.

The millstone 3 auto pilot:

1. void runPilot()-

the new auto pilot gets the command to start from the presentation layer, and checks (by using methods from History class) the best prices (sell and buy) for each stock, picks the stock which will give the best profit and buys it. After 1 interval, the pilot sells all the commodities with the best sell price it calculates before.

The pilot connecting to the server via the LogicLayer class.