**SEMESTER PROJECT REPORT**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

# **Trading System with AI Integration**

**Submitted by:**

M. Muntazar | 470861

Mehreen Raheel | 460502

Sana Khan Khitran | 464597

Usman Naeem | 481453

M. Hamdan Ishfaq | 455881

**School of Electrical Engineering and Computer Science**

Bachelor of Science in Data Science

**Fundamentals of Computer Programming**

Instructors: Nazia Perwaiz and Iram Tariq Bhatt

**Table of Contents**

[Trading System with AI Integration 1](file:///D:\OneDrive%20-%20National%20University%20of%20Sciences%20&%20Technology\Study\Semester%201\Fundamentals%20of%20Computer%20Programming\Project\Documents\Project%20Report.docx#_Toc154425538)

[ABSTRACT 4](#_Toc154425539)

[INTRODUCTION 5](#_Toc154425540)

[Objective 6](#_Toc154425541)

[Implementation 7](#_Toc154425542)

[**Helper Functions** 14](#_Toc154425543)

[**1.** **Symbols Box:** 16](#_Toc154425544)

[**2.** **Leverage:** 18](#_Toc154425545)

[**3.** **Select Symbol Message:** 19](#_Toc154425546)

[**4.** **Open Positions:** 20](#_Toc154425547)

[**5.** **Available Balance:** 21](#_Toc154425548)

[**6.** **Select Order Type:** 22](#_Toc154425549)

[**7.** **Order Type Box:** 23](#_Toc154425550)

[**8.** **Margin Slider:** 24](#_Toc154425551)

[**9.** **Price:** 25](#_Toc154425552)

[**10.** **BUY/SELL Buttons:** 26](#_Toc154425553)

[**11.** **Open Orders:** 27](#_Toc154425554)

[**12.** **Cancel Order:** 28](#_Toc154425555)

[**13.** **Next Order:** 29](#_Toc154425556)

[**14.** **Close Position:** 30](#_Toc154425557)

[**15.** **Next Position:** 31](#_Toc154425558)

[**16.** **Order Book:** 32](#_Toc154425559)

[**17.** **Market Price:** 33](#_Toc154425560)

[**18.** **Open Chart Button:** 34](#_Toc154425561)

[**19.** **Reload Button:** 35](#_Toc154425562)

[**20.** **On-Screen Logs:** 36](#_Toc154425563)

[**21.** **Status Bars:** 37](#_Toc154425564)

[**22.** **How are we making the text in the status bar disappear?** 37](#_Toc154425565)

[**23.** **AI Bot:** 38](#_Toc154425566)

[**Logic of the Machine Learning Model:** 39](#_Toc154425567)

[**Limitations:** 47](#_Toc154425568)

[**Plans for the Future:** 47](#_Toc154425569)

[**Features:** 48](#_Toc154425570)

[**Challenges:** 48](#_Toc154425571)

# **ABSTRACT**

This project report describes the development of a robust Trading System and an advanced Artificially Intelligent Trading Bot.

The purpose of the Trading System is to provide users with a dynamic Graphical User Interface via which they can execute transactions on different commodities using real money on a real exchange. It involves functions including an interactive order book, open positions and orders, and real-time balance tracking. Users can also define transaction volumes, margins, order types, leverage, and more when placing trades, among other choices. The system also incorporates Candlestick charts to enhance the user's trading experience, aspiring to create a competitive trading environment akin to prominent platforms like LBANK, TradingView, and Binance.

The second component of this project involves an Artificially Intelligent Trading Bot that leverages historical data and machine learning techniques. Using a predictive model to estimate future candlestick patterns, the bot assigns values to green and red candles depending on set parameters. Through the integration of established trading indicators and results, the bot seeks to develop an effective AI-powered trading bot that can identify and profit from market opportunities.

The main goal of the project is to provide an easy-to-use interface that will enable users to trade an extensive range of commodities. The development of an AI model also concentrates on identifying possible trading opportunities in the market. Key functions of this application include real time data access, account details display, real exchange integration for live trade placements, interactive charting for market pattern analysis, and a responsive interface for user-friendly navigation.

The Trading Application implementation serves as a practical example of applying fundamental concepts in a C++ environment as well as the integration of a Graphical User Interface (GUI), JSON data handling, and WinForms (Windows Forms) collectively enhances the user experience, data management, and overall performance of the application.

# **INTRODUCTION**

The **Trading System**, designed with a user-centric approach, offers a **Graphical User Interface** **(GUI)** enabling real-time trades on diverse commodities directly within a real exchange. This comprehensive platform provides users with detailed financial insights, multiple trade placement options, and insightful Candlestick charts, aspiring to rival established industry platforms. The **AI Trading Bot** adds to this by utilizing historical data and advanced algorithms. Main features including user settings and a responsive interface are included in the functionality, along with real-time market data and comprehensive account information. With the use of **C++ programming language**, the **WinForms (Windows Forms)**, **JSON parsing**, and **file handling**, the project seeks to offer a dependable, adaptable, and cutting-edge platform for effectively navigating financial markets.

# **Objective**

The lofty goal of our project is to enable Pakistan's financially impoverished and less educated citizens to take the first steps toward financial independence. Beyond mere democratization of access to financial markets, our mission extends to providing comprehensive educational resources within the platform. Through tutorials, guides, and simplified explanations, users can acquire fundamental knowledge about trading strategies, risk management, and market dynamics—eradicating the traditional barriers imposed by expensive courses. Our AI-powered trading platform is distinguished by its outstanding functionality, intuitive layout, and effectiveness. Our state-of-the-art technology guarantees smooth platform navigation even for users with no prior experience. The system's intuitive interface makes trading accessible to everyone, irrespective of their background in financial markets. With a minimal investment, anyone can leverage the power of our trading system to make profitable decisions. We utilized C++, WinForms, JSON parsing, and GUI not merely as a technical aspect but to ensure accessibility and ease of use for users with diverse technological backgrounds. By providing a responsive and stable environment, our platform aims to dismantle barriers and enable all users—regardless of socioeconomic background—to make a little but significant profit from their participation in the financial markets with confidence, as our AI system does the heavy lifting, allowing users to participate and prosper in the world of trading with ease.

# **Implementation**

## **Algorithm**

### General Structure:

### **Initialization:**

* + Setup necessary components, global variables, and libraries.
  + Initialize UI elements and event handlers.

### **API Interaction Functions:**

* + Define functions for handling API requests, response parsing, and error handling (e.g., handleGetRequests, writeToLog, writeToCSV, etc.).

### **Helper Functions:**

* + Define auxiliary functions for logging, timestamp generation, signature creation, rounding, array manipulation, etc.

### UI Components:

Initialization and Fetching Symbols:

### **Fetch Available Symbols:**

* + In the MyForm\_Shown event handler:
    - Use fetchAvailableSymbols to get a list of symbols from Binance.
    - Display symbols in the Symbols List Box.

### **Leverage and Symbol Selection:**

* + listBox1\_SelectedIndexChanged:
    - Enable leverage slider and AI bot checkbox.
    - Fetch allowed leverage based on selected symbol.

### **Orders, Balances, and Order Types:**

* + Update available balance and display open positions.
  + Enable order type selection (Limit/Market).

### **Buy/Sell Actions:**

* + Validate user inputs.
  + Place a trade using placeTrade for buying/selling.

### **Open Orders and Order Book:**

* + Display open orders and order book.
  + Enable cancel order functionality.

### **Close Position and AI Bot Integration:**

* + Close a position if requested.
  + Integrate AI bot: Fetch historical data, apply ML algorithm (ai function), execute trade actions based on the AI output.

### **Refresh and Chart Viewing:**

* + Periodically refresh data for positions, orders, market price, and order book.
  + Enable chart viewing for selected symbols.

### **Error Handling and Log Display:**

* + Handle errors elegantly and display them on the UI.
  + Log actions, errors, and responses for debugging.

### Algorithm for AI Bot Functionality:

### **AI Bot Initialization:**

* + Enable AI bot checkbox and time frame selection upon valid symbol selection.

### **Fetch Historical Data:**

* + Fetch historical data for the selected symbol.
  + Process data and send it to the AI function.

### **AI Logic (ai Function):**

* + Analyze historical data using ML algorithm.
  + Generate boolean indicators for trading actions: Open Long, Close Long, Open Short, Close Short.

### **Trade Execution Based on AI Output:**

* + Implement trade actions based on AI-generated signals.
  + Validate and execute trades for the selected symbol.

### Miscellaneous:

### **Reload and Cleaning Up:**

* + Provide a reload button for program restart.
  + Manage status bars and clean-up for displaying trade-related message

## **Usage**

### **If-else Statements:**

|  |
| --- |
| if (open\_orders[0] == "Press the Next Button or Reload. ") { //Showing error msg with appropriate color if occurs  onScreenLogs->Text+="Error: Couldn't Fetch Open Orders. Try Again.\n";  return;  }  else {  onScreenLogs->Text+="Open Orders Fetched Successfully.\n"; //Showing msg  } |

### **For-loop:**

|  |
| --- |
| for (int i = 0; i < historicalData.size(); i += 4) {  open.push\_back((historicalData[i]));  high.push\_back((historicalData[i + 1]));  low.push\_back((historicalData[i + 2])); //Splits the input data into 4 different vectors. Which will be used for calculations.  close.push\_back((historicalData[i + 3]));  } |

### **Nested Loop:**

|  |
| --- |
| for (int j = 0; j < close.size(); j++) {  for (int i = 3; i > j; i--) {  closeReplication.pop\_back();  }  } |

### **Functions:**

|  |
| --- |
| //Function for round to precision  double round\_to(double value, double precision)  {  return std::round(value / precision) \* precision; // Example: round\_to(0.123456, 0.0001) = 0.1235  } |

### **Arrays:**

|  |
| --- |
| int arrary\_length(bool arr[]) {  return sizeof(arr) / sizeof(arr[0]);  } |

### **Strings:**

|  |
| --- |
| //Function to fetch the realtime value of a symbol  std::string fetchRealTimeValue(const std::string& symbol) {  std::string readBuffer;  // Binance API endpoint for retrieving the real-time price of a symbol  std::string url = "https://api.binance.com/api/v3/ticker/price?symbol=" + symbol;  readBuffer = handleGetRequests(url);  try {  // Parse the JSON response to extract the real-time value  json json\_result = json\_result.parse(readBuffer);  // Check if the "price" field exists in the response  if (json\_result.contains("price") && json\_result["price"].is\_string()) {  std::string realTimeValue = json\_result["price"];  std::vector<std::string> result;  result.push\_back("Real Time Price Fetched Successfully. ");  writeToLog(result);  return realTimeValue;  }  else {  std::vector<std::string> error;  error.push\_back("Problem faced with Fetching Real Time Price");  writeToLog(error);  throw std::runtime\_error("Error Fetching Real Time Price. ");  }  }  catch (const std::exception&e) {  std::vector<std::string> error;  error.push\_back({ e.what() });  writeToLog(error);  return "Try Again After Reloading. ";  }  } |

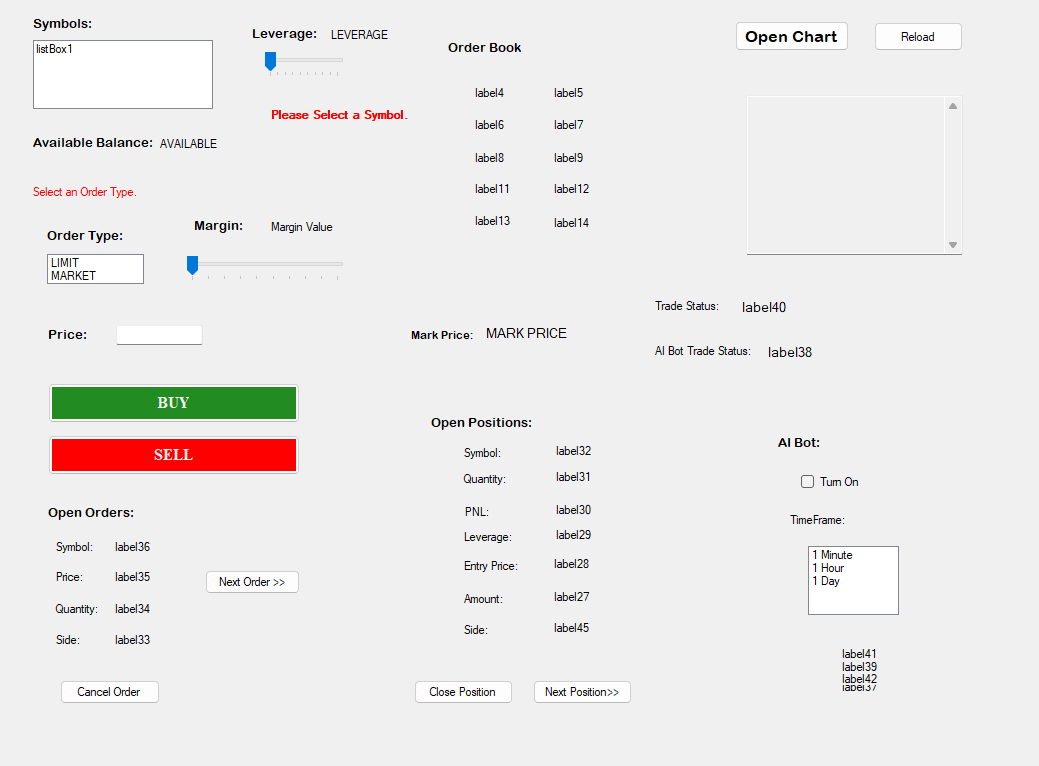
### **Use of Pointers to access arrays:**

|  |
| --- |
| double\* arr = new double[2000];  long long timestamp = json\_string[0][0];  current\_timestamp = std::to\_string(timestamp);  for (int i = 0; i < 2000; i+=4) {  \*(arr + i) = std::stod(json\_string[i][0].get<std::string>());  \*(arr + i+1) = std::stod(json\_string[i][1].get<std::string>());  \*(arr + i+2) = std::stod(json\_string[i][2].get<std::string>()); //Storing the required data in appropriate vectors  \*(arr + i+3) = std::stod(json\_string[i][3].get<std::string>());  } |

### **File Handling:**

|  |
| --- |
| // Function to write to the log file  void writeToLog(std::vector<std::string> Buffer) {  std::ofstream file; // Create an output file stream  file.open("Logs.txt", std::ios\_base::app); // Open the file in append mode  // Check if the file is open  if (!file.is\_open()) {  file.open("Logs.txt", std::ios\_base::app); // Open the file in append mode  if (!file.is\_open())  return;  }  for(const auto& data : Buffer) { // Write the data into the file  file << data << "\n";  }  file.close(); // Close the file  }  // Function to write to the CSV file  void writeToCSV(std::vector<std::string> rowData) {  std::fstream file; // Create an output file stream  file.open("Trades.csv", std::ios::in | std::ios::out | std::ios\_base::app); // Open the file in append mode  // Check if the file is open  if (!file.is\_open()) {  file.open("Trades.csv", std::ios\_base::app); // Open the file in append mode  if (!file.is\_open())  return;  }  file.seekg(0, std::ios::end); //Move to the end of file  // Check the file's size  if (file.tellg() == 0) {  // File is empty, write the header info  file << "Date/Time,Side,Symbol,Price,Amount,OrderType,Leverage,Status\n";  }  else {  // File has content, move the pointer back to the last newline  file.seekg(-1, std::ios::cur);  char ch;  file.get(ch); // Get the current byte data  while (ch != '\n') { // Check if the current byte is a newline  file.seekg(-2, std::ios::cur); // Move the pointer back by 2 bytes  file.get(ch); // Get the current byte data  }  }  // Write the data into the file  for (const auto& data : rowData) {  file << data << ",";  }  file << "\n";  // Close the file  file.close();  } |

This is the main window of our program. This UI has been created using Windows Forms application. WinForms provide us ease because we do not have to write all the code by ourselves. It is just drag and drop and it even creates event handlers for us automatically. We do have done some visual changes like change the colour of some things or change the formatting of a few things. But we did not need to write code for that, instead we only changed the colour from the toolbox provided by WinForms.



Before we dive into the actual implementation, I think it would be helpful to first understand the basic flow of execution of the program. When the user interacts with the UI, that interaction is first sent to the WinForms app, in which we have created many event handlers to interpret all sorts of inputs. Now these event handlers interpret the user’s actions and respond accordingly and if the user’s actions are meaningful, they will require some sort of interaction with the Binance exchange. For that we have created a separate file called API\_Interaction.cpp containing all the functions that the user may need to interact with the Binance API. But these functions have a lot of repetitive tasks that they have in common. So, we created another file called CRUD\_Requests\_helper\_functions.cpp. Now I know that this API is restful, but I have named the file so and I am not going to change it for simplicity. Here we are going to dive deep into the helper functions within the CRUD\_Requests\_helper\_functions.cpp file before we discuss the rest of the program.

## **Helper Functions**

writeToLog():

This function takes a string vector as parameter and returns nothing. It is used to log the contents of the string vector it received in parameter in a Logs.txt file.

If a file with the name exits, it will append that file. Otherwise, it will create a new file and write in that.

writeToCSV():

This function takes a string vector as parameter and returns nothing. It is used to save the trades that the user placed. The info of the trade is passed to it through the string vector it received in parameter. It stores that info in a Trades.csv file.

If a file with the name exits, it will append that file. Otherwise, it will create a new file and write in that.

readEnvFile():

Since we are dealing with API, we also have an API and a SECRET key that we need to interact and access data on the exchange. Although these are not needed when fetching trivial information from the exchange.

This takes the filename as a string in parameters and finds that file. The file must be in the same directory as the exe file. If it cannot find or extract the required info from the env file. The program will exit. Otherwise, it will store the keys values in a string vector and return that. Which we will then store in variables and use.

One limitation of this approach is that the file must have a very strict format as C++ does not have built-in support for .env files. The format should be something like:

API\_KEY=abcdefgh

SECRET\_KEY= abcdefgh

No blank space is acceptable in between or at the ends. Otherwise, it will mess up with the program.

generateTimeStamp():

This function uses the chrono library to construct a timestamp in milliseconds and calculates it with respect to the time since epoch.

createSignature():

This function takes a string message as a parameter and returns the hashed string.

This is needed because the Binance API doesn’t accept requests for sensitive information without the API and SECRET key being hashed in a particular way in the HMAC SHA265 digest. This has not been written by us but instead provided by Binance.

write\_callback():

This function takes ‘contents’ as a void pointer, size as size\_t, nmenb as size\_t and data as a string pointer in parameters. And returns the number of bytes received in size\_t datatype.

handleGetRequests():

This function takes only the URL as a string in parameters and returns the response of the request as a string.

It uses the curl library. It first initializes the curl object, then if the object has been initialized it sets the headers for the request including the API key. It then sets the URL for the request, the callback function to handle the response data, the readBuffer, the errorBuffer to store the errors, and sets the verbose mode. Then it performs the request and if the request is successful, it returns the readBuffer in which it stored the response. Otherwise, if at any point an error occurs it automatically throws an error, which we catch, log and return.

*One thing to note that we have already initialized the curl global object and have also prepared for its clean up in the MyForm.cpp file which contains the* main() *function. This is because the curl API docs tell us that the global initialization must be done only once in the program.*

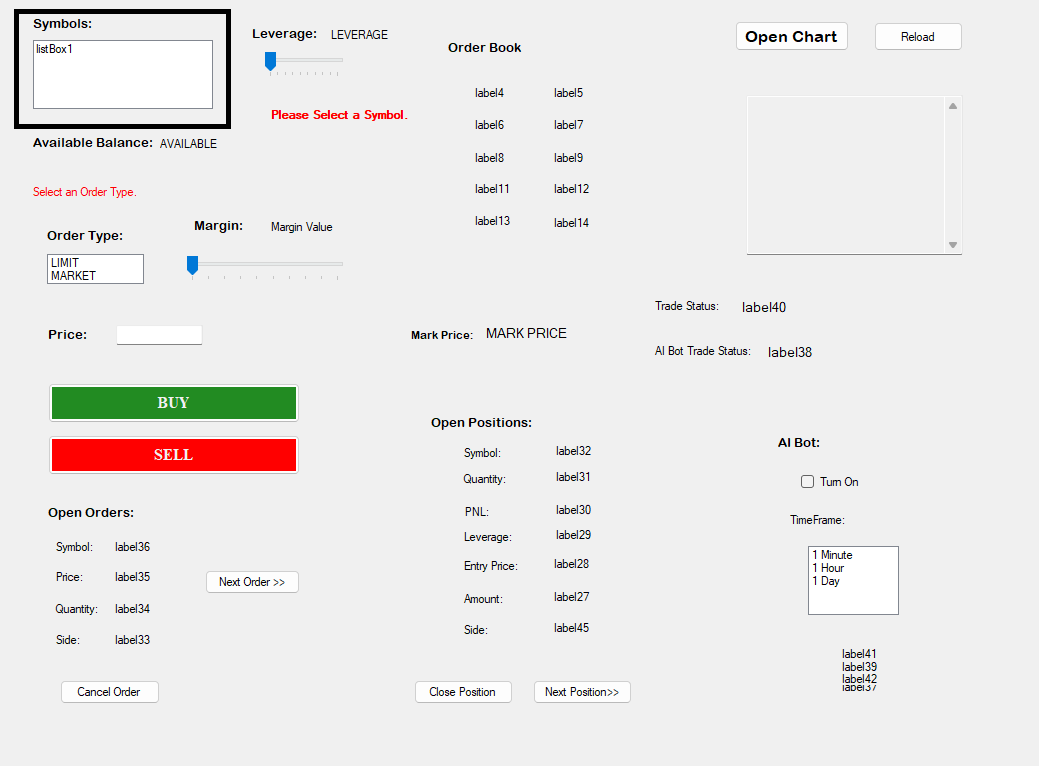
round\_to():

It takes two parameters, the value: double and the precision: double. It rounds the value off to the number of decimal places told in the precision and returns the rounded off number as a double.

array\_push\_before():

It takes two double vectors as params and returns a double vector which has the second vector’s elements before all the elements of the first vector. Essentially pushing one vector before another.

## **Symbols Box:**

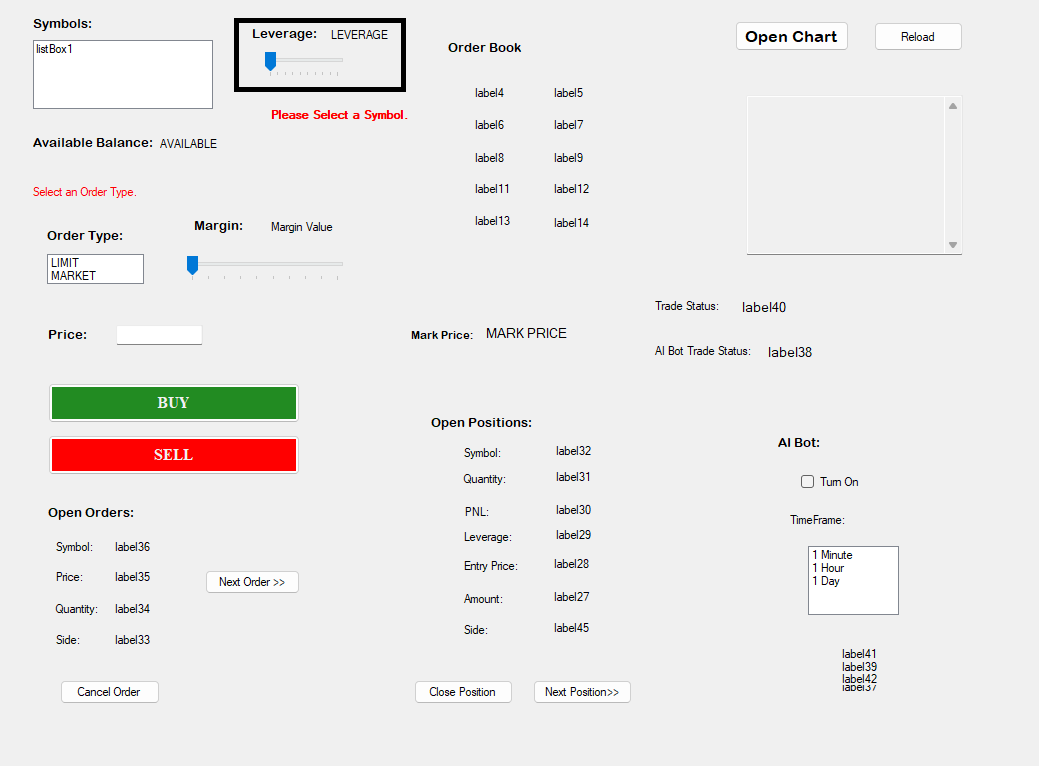


This is a List Box of WinForms. When the form is loaded, we make an API request to Binance’s API endpoint /fapi/v1/exchangeInfo which returns the list of available symbols for trading on Binance.

The Form has an event handler named MyForm\_Shown() which is invoked only when the form is shown. In this event handler we have put the logic for fetching and displaying the symbols. For the actual fetching of the symbols, we have created a separate function named fetchAvailableSymbols() in the API\_Interaction.cpp file. This function creates a string called readBuffer to store the response of the GET request. We then create the URL to which we want to send the request to use the globally defined BASE\_URL. Then we perform the get request by calling the handleGetRequest() function which as we have already discussed above sends the get request and stores the response in the readBuffer. Now the response we received is in JSON format and as we know that C++ has very poor built-in JSON handling. So, we are using a 3rd party library called nlohmann JSON. We first initialize ‘Json’ object (Here ‘Json’ is referring to a datatype provided by the nlohmann JSON library) by parsing the readBuffer. We then slowly start to extract and check what to do with the response. If the response contains our required data, we extract that data and return in a string vector. Otherwise, if the response doesn’t contain what we want. We just send an error back in a string vector.

The string vector is received by the WinForms app which then checks for error and displays result accordingly. And also, we have intentionally made the WinForms app incapable of interacting with the logs file so everything that we do in the ‘frontend’ side we log it on the on-screen logs.

## **Leverage:**

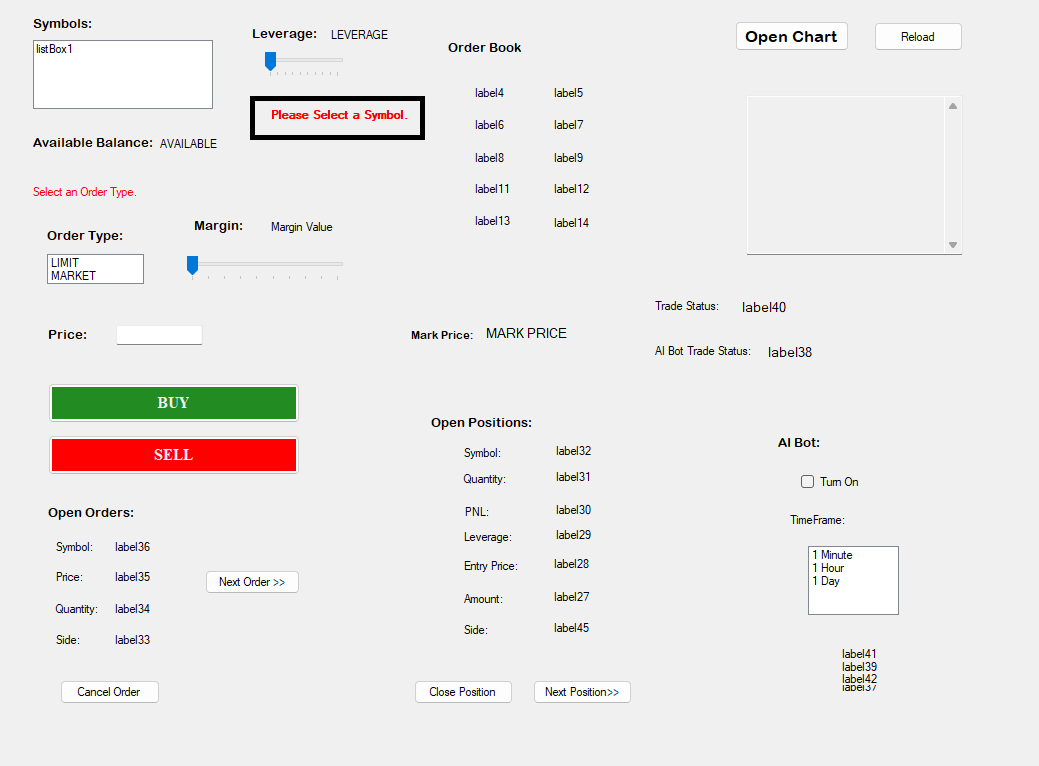


The leverage slider’s value is linked to a label so that the user is able to see the value that they are selecting. This slider will remain disabled as long as a symbol is not selected because the leverage actually varies for every symbol. In the symbols list box, we have created an event handler named listBox1\_SelectedIndexChanged() that is invoked every time something is selected from the symbols box. It has three functions:

1. To enable the leverage slider
2. To enable the AI bot checkbox. More on that later
3. Open Chart Button

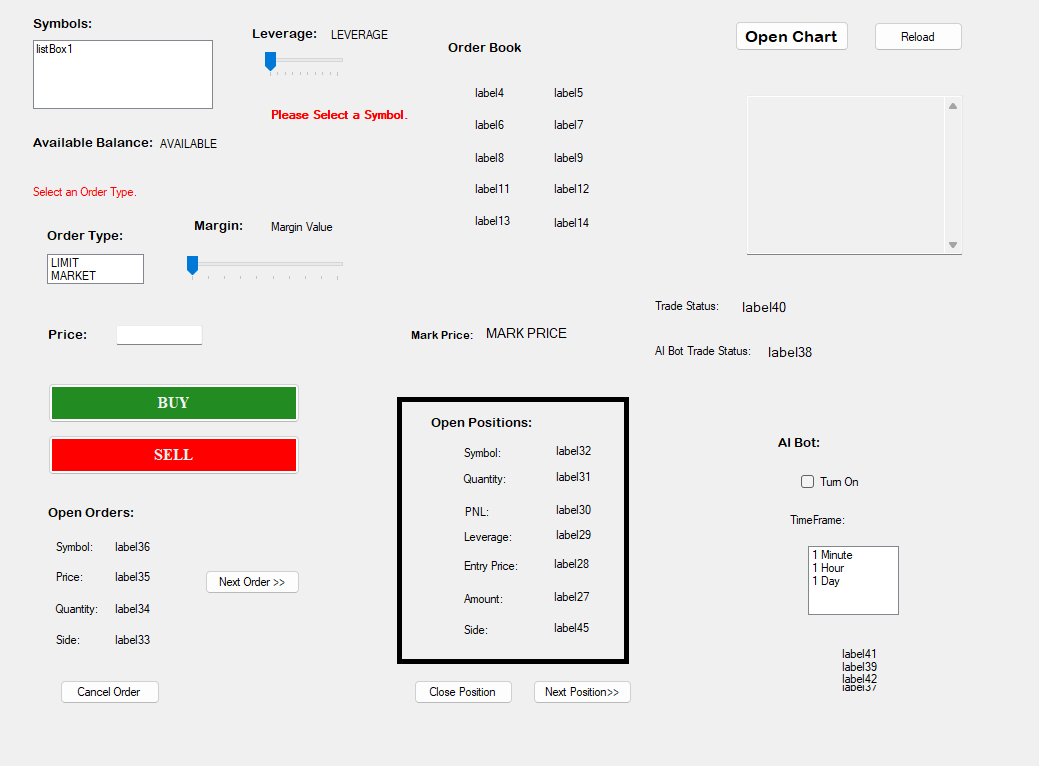
So, when some symbol is selected it calls the fetchAllowedLeverage() function with the name of the selected symbol. Then this function makes a request to /fapi/v1/leverageBracket endpoint with the name of the symbol and the current timestamp generated from the (generateTimeStamp()) function using the handleGetRequest() function that we have discussed earlier. The JSON response is parsed, and the needed info is extracted using loops and if-else statements and is returned. Otherwise, an error is returned. The event handler that initially made the request, receives the response and handles it accordingly. And if it was successful then the slider is automatically set to the maximum to make it easier for the user. All events are logged.

## **Select Symbol Message:**



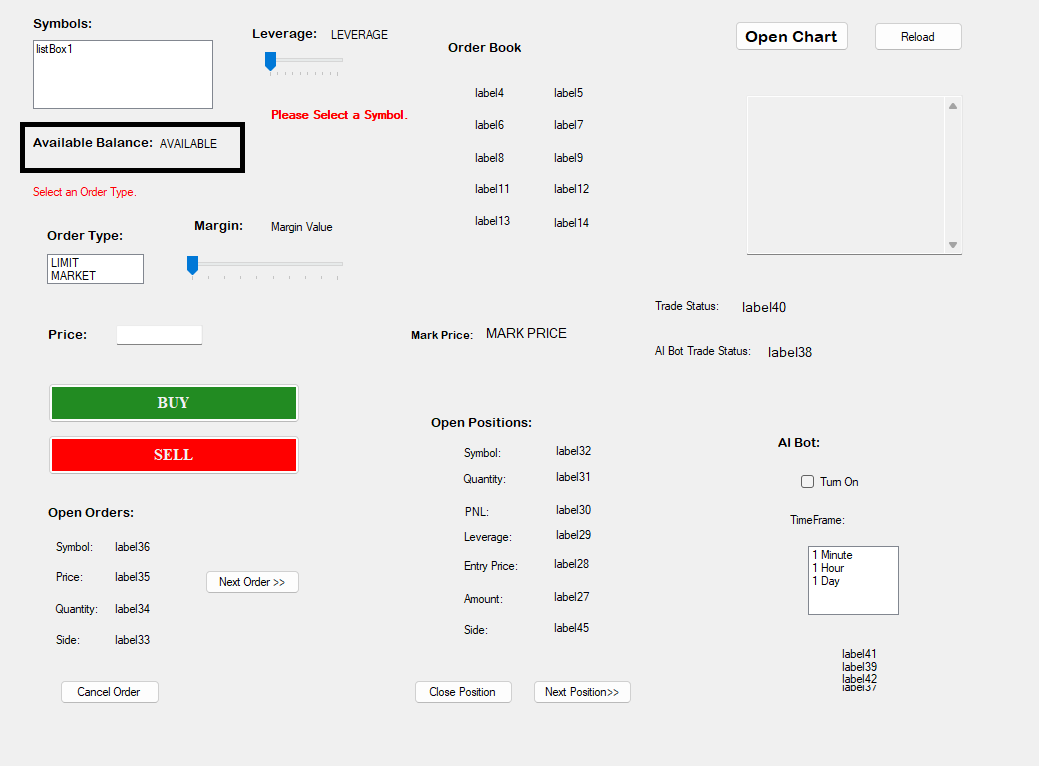
When you select any valid symbol, this message goes away. It is linked to listBox1\_SelectedIndexChanged() event handler as well.

## **Open Positions:**



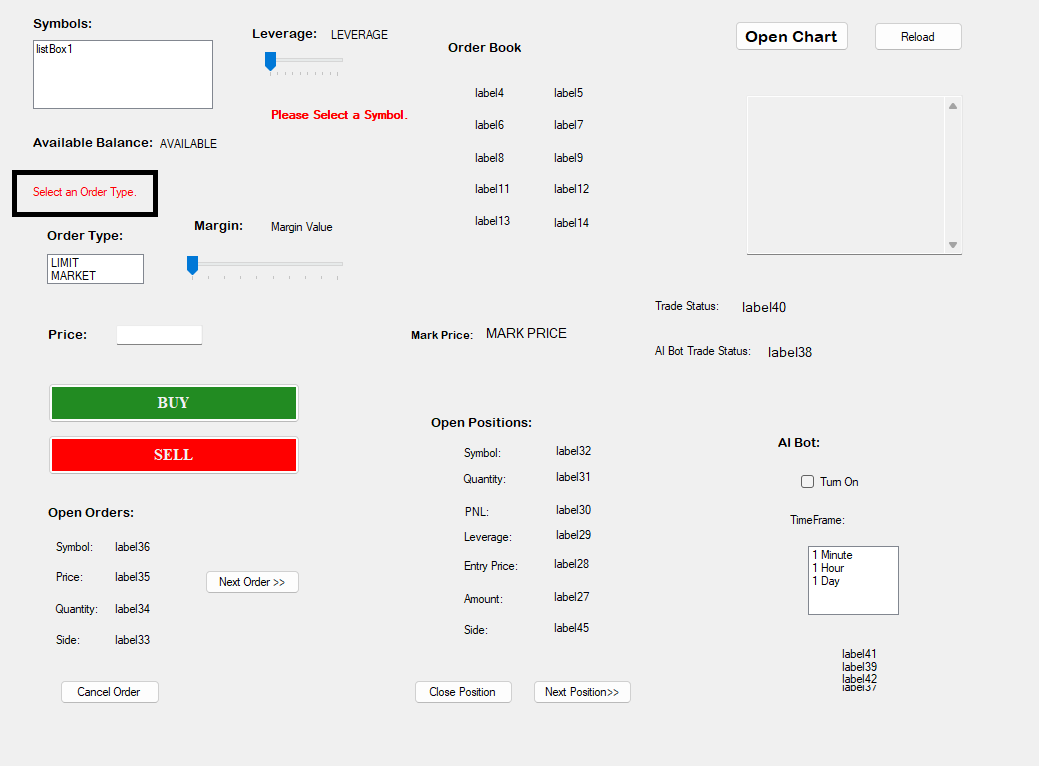
It displays the open positions. The function that contains its logic is named positionsInitializer () (because in addition to the open positions, it initializes many other things that is why it has such a general name) is called in the thread as well. So, it is also continuously updating after every 15 seconds. Whenever the positionsInitializer () is called it first calls the function named fetchOpenPositions(). Which fetches all the open positions of the user from the /fapi/v2/account endpoint with the current timestamp. This endpoint returns many things from which one useful thing other than the list of open positions is the available balance of the user. So, we first send a get request using the handleGetRequest() function, parse the JSON, use if to handle anomalies and for loop to extract info. We then send this this string vector if the operation was successful other an error message is sent and logged. The positionsInitializer () receives this vector and then depending upon a few flags and the parameter i: int. It displays the open positions or displays that there are no open positions if that is the case. This function also has the capability to close a position but more on that later. All events are logged.

## **Available Balance:**



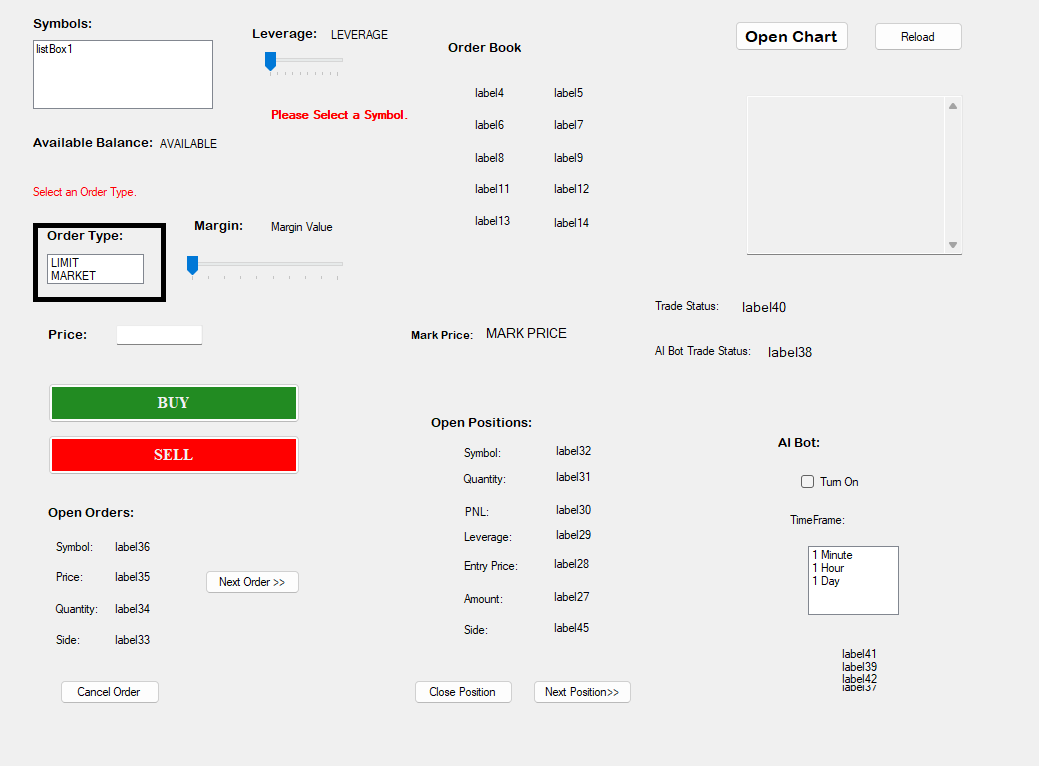
This displays the available balance of the user. This is received as part of the response that we receive for the open positions. So, we just extract it from there. As we have discussed it earlier.

## **Select Order Type:**



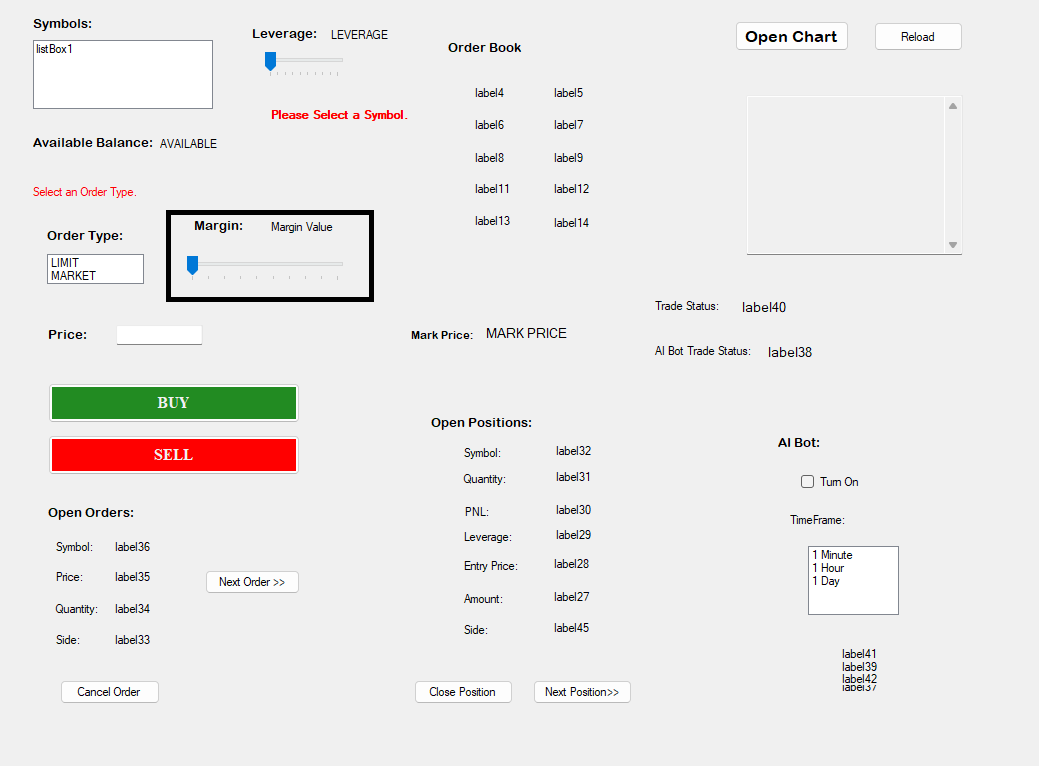
This message keeps on displaying until the user selects an order type. More on the implementation in the succeeding heading.

## **Order Type Box:**



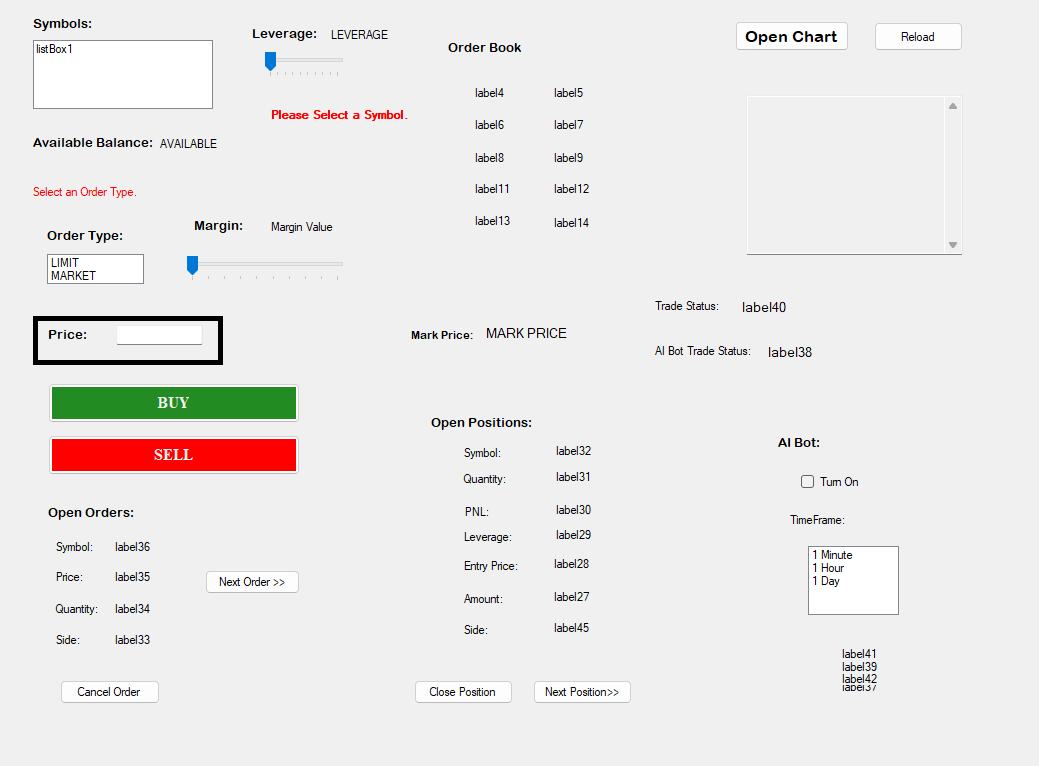
This box contains two options: LIMIT and MARKET. It has an event handler listBox2\_SelectedIndexChanged() like that of the symbols box. Whenever an order type is selected it is invoked and upon invocation it first clears the select order type message. Now because MARKET order does not need to interact with price, the price textbox remains disabled and is only enable if the selected order type is LIMIT.

## **Margin Slider:**



This displays the margin selected by the user and its max value is the available balance that is also being shown to the user above.

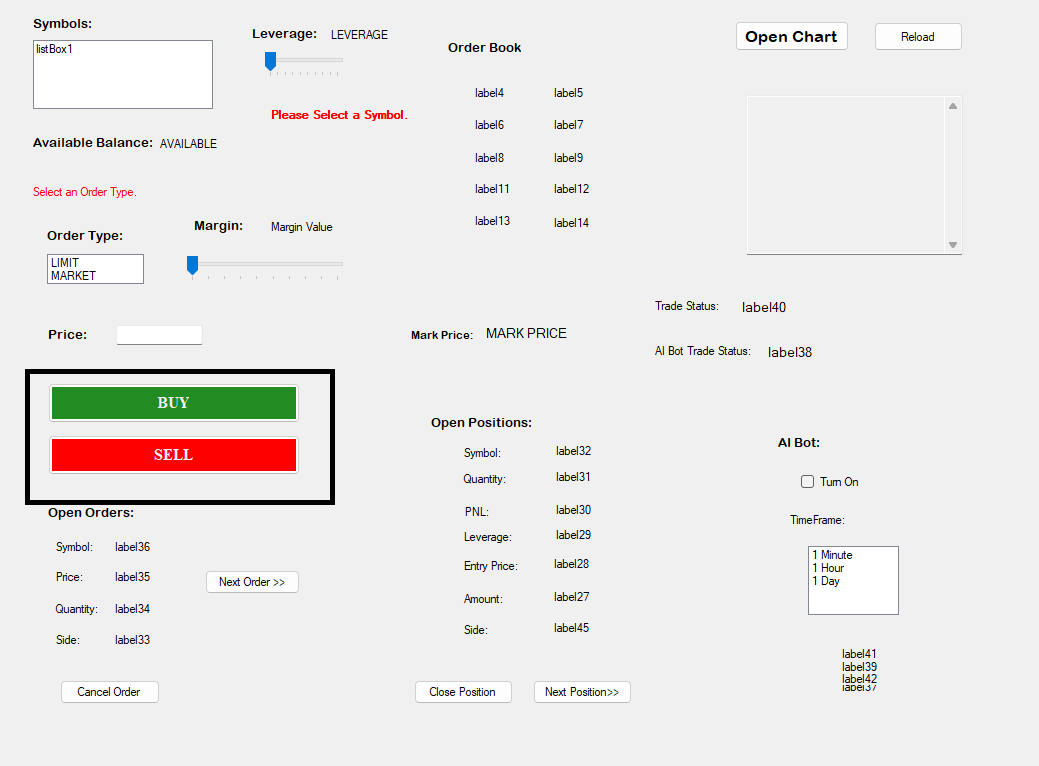
## **Price:**



This is a textbox that takes input from the user if the selected order type is LIMIT. It has an event handler textBox1\_KeyPress() that is invoked every time a key is pressed inside of it. And in it we have put an if-else statement that only allows the following things:

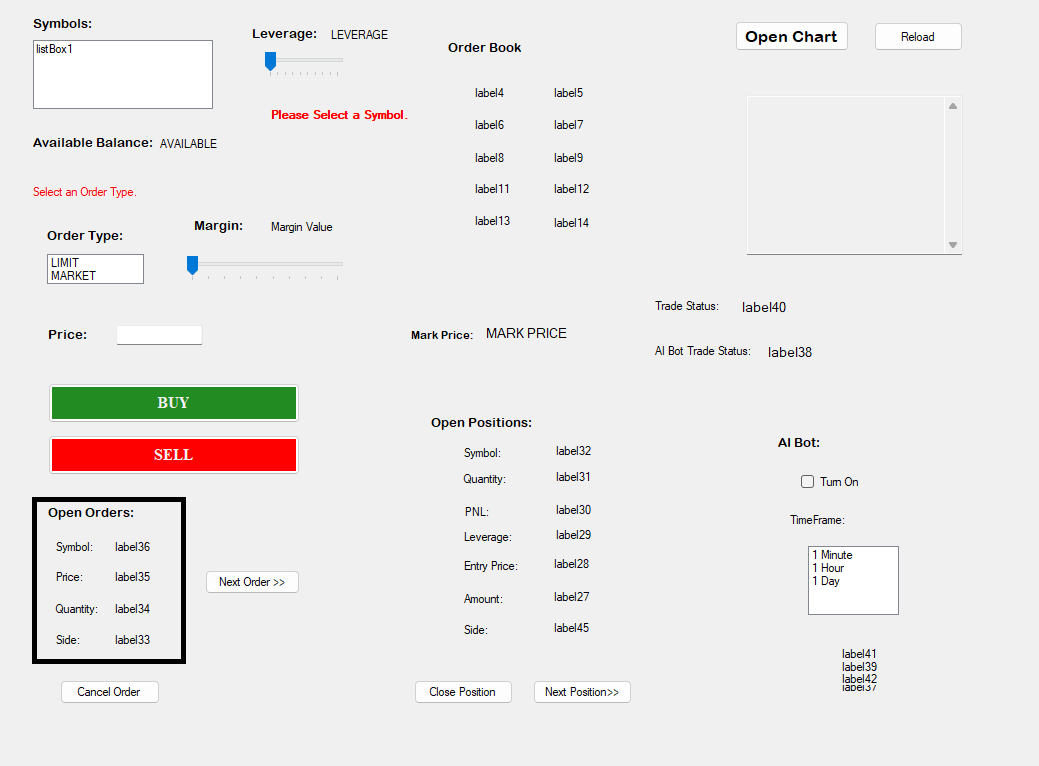
* 1. Numbers 0-9
  2. Decimal point ‘.’
  3. Backspace

## **BUY/SELL Buttons:**



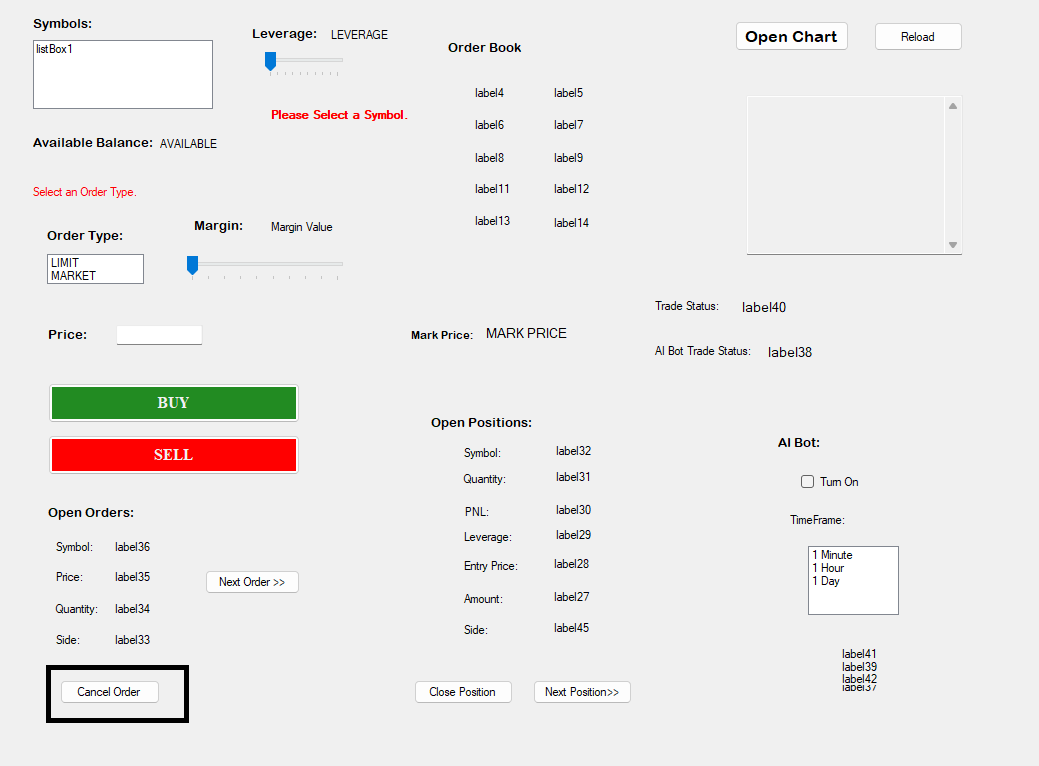
These buttons have event handlers defined to them that get invoked whenever these buttons are clicked. And they call the placeLongTrade() and the placeShortTrade() functions respectively. Both of have same working with the only difference being that one uses the value “BUY” while the other uses “SELL”. These functions first check if all the required and correct things have been selected, like the symbol, margin, order type etc. And if any of these are missing then a status message is shown at label40 in the trade status. If all the things are given and correct, then this function places the trade by calling the placeTrade(). This function takes the required things from parameters and sends the request to /fapi/v1/order because it is a post request. We cannot use the handleGetRequest() function here and instead we have to construct the curl object from scratch just like while sending a get request. Except that here we put it in POST Request ‘mode’ and set the required fields and also the errorBuffer for storing errors. Then it performs the request and extracts data from it and returns a success message back in form of string. Otherwise, if an error occurs it handles it elegantly and sends a predefined error back to the user while logging the detailed error msg. One important thing we need to mention is that to place an order it needs something called quantity. And the acceptable precision for the quantity varies from symbol to symbol. So, we created a function getSymbolPrecision() that gets any symbol’s precision and returns it which is then used to calculate the quantity of the trade. All of this is logged, and in the case where trade is successfully placed, it is also saved in the csv file with all the details. This function can also be used to close open trades/positions. But more on that in its section.

## **Open Orders:**



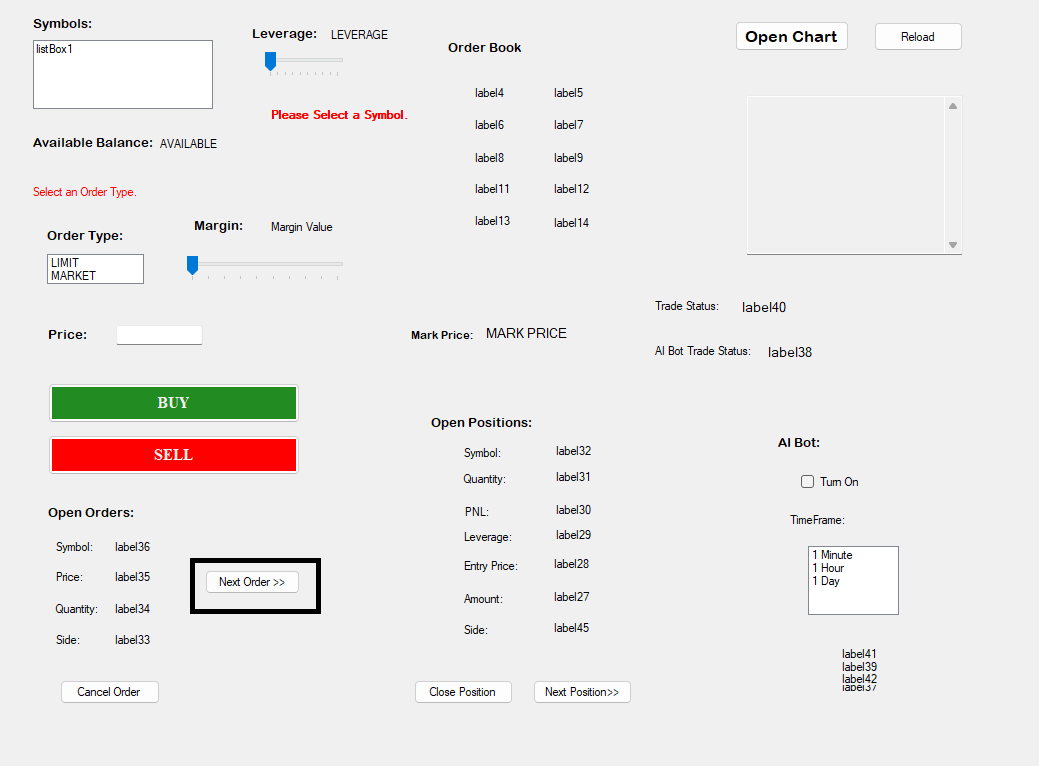
The open orders display the currently open orders or display “No Open Orders” if there are none. It is also connected to our thread, and so its function ordersInitializer() gets executed after every 15 seconds. This function first calls the fetchOpenOrders() function that makes a GET request to /fapi/v1/openOrders with the current timestamp using the handleGetRequests function. It parses the JSON response, extracts the open orders if any, and returns a string vector back. If any error occurs, then a predefined msg is returned while a detailed msg is logged upon failure or success. The returned string vector is then broken down into its components and displayed by the ordersInitializer() function. This function contains logic to cancel an order and also the logic to show the next orders, if any, from the open orders array.

## **Cancel Order:**



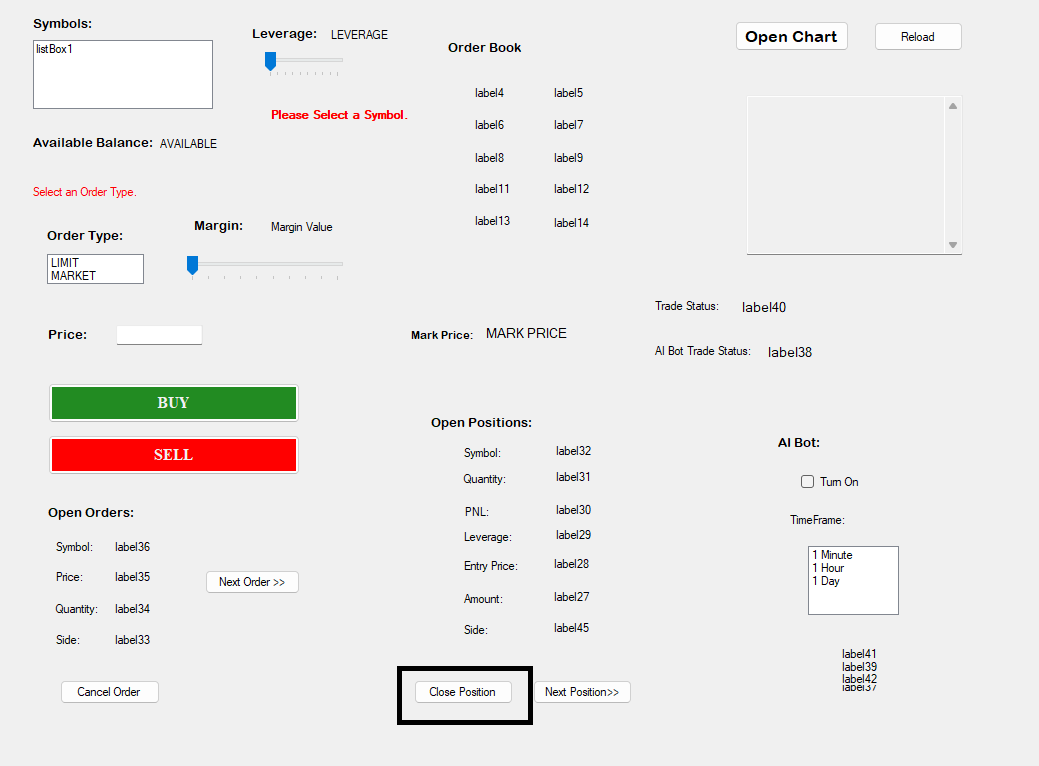
These buttons ‘click’ event handler has only two lines of code. It turns the cancelThisOrder flag to true and then calls the ordersInitializer() function with the previous value of the integer ordersStartIndex. The ordersInitializer() checks a few things and if there are any open orders on the index ordersStartIndex told by the button’s event handler then it calls the cancelOrder() function. This function takes the order’s symbol name and the order’s id and sends the request with the current timestamp to /fapi/v1/order. Since it is a post request, we have done here the same thing that we did in the placeTrade() function. We initialize the curl object and sets the correct headers and settings and then performs the request. The response is parsed and then checked. If the operation was successful, then it is stored in the Trades.csv file and in the logs file. A string response is returned as well to tell the frontend what to do. The frontend receives this string and performs the required operations such as showing the status etc. Then it sets the cancelThisOrder flag back to false and again fetches the open\_orders and then the rest of the code executes as explained in the Open Orders section. Some of these things are logged in the file and the rest are shown in both the on-screen logs and the Trade Status bar.

## **Next Order:**



This button makes a call to the ordersInitializer() function after incrementing 5 into the ordersStartIndex integer variable. This k tells the ordersInitializer() which order to display. If there is only one open order, then it is not changed. If there are no open orders, then nothing happens.

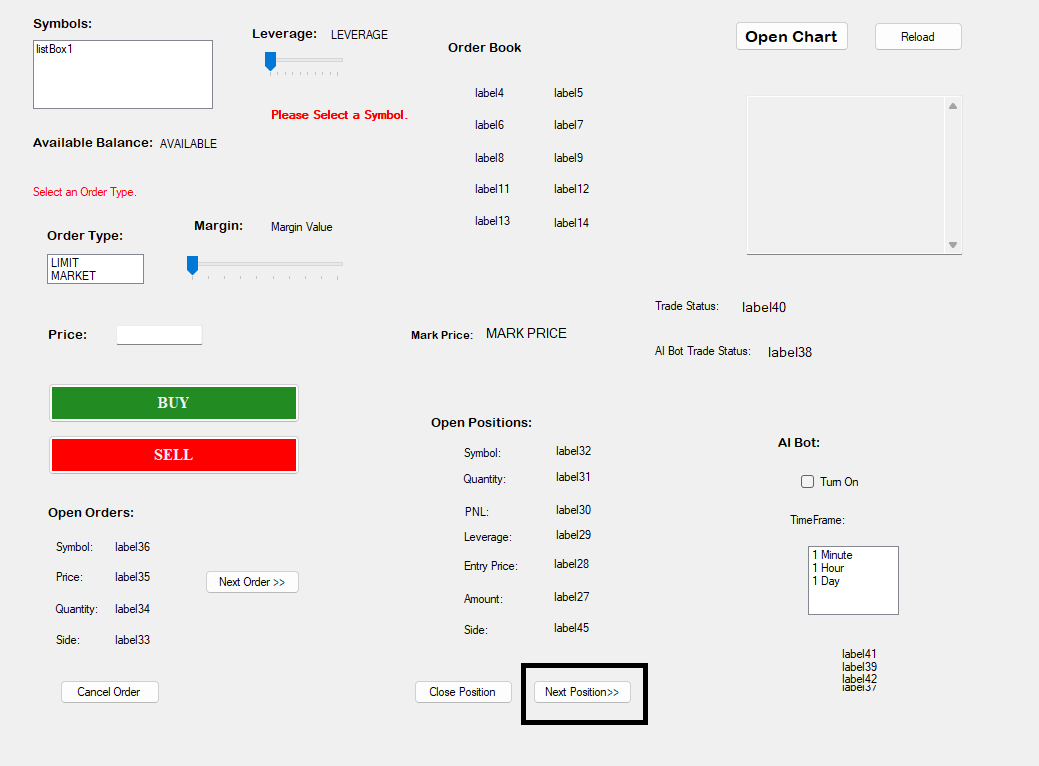
## **Close Position:**



Upon clicking this button, the closeThisPosition flag is set to true and this calls the positionsInitializer () function without changing the value of the integer positionsStartIndex. This causes the positionsInitializer () to extract the details of the selected position and then make a call to the closePosition() function which first fetches the current price of that symbol using the fetchRealTimeValue() function. Then after collecting all the needed values. It calls the placeTrade() function, which then places the same trade, which was open, with the same amount, in the opposite direction. Essentially closing the trade. This is logged both in trades.csv and logs.txt. The result of this trade being placed is returned to the closePosition() function which upon checking the responses category, sends a predefined msg back to the frontend depending upon the category. The frontend (initializer()) function receives this response and acts accordingly. All of these things are logged in many different ways.

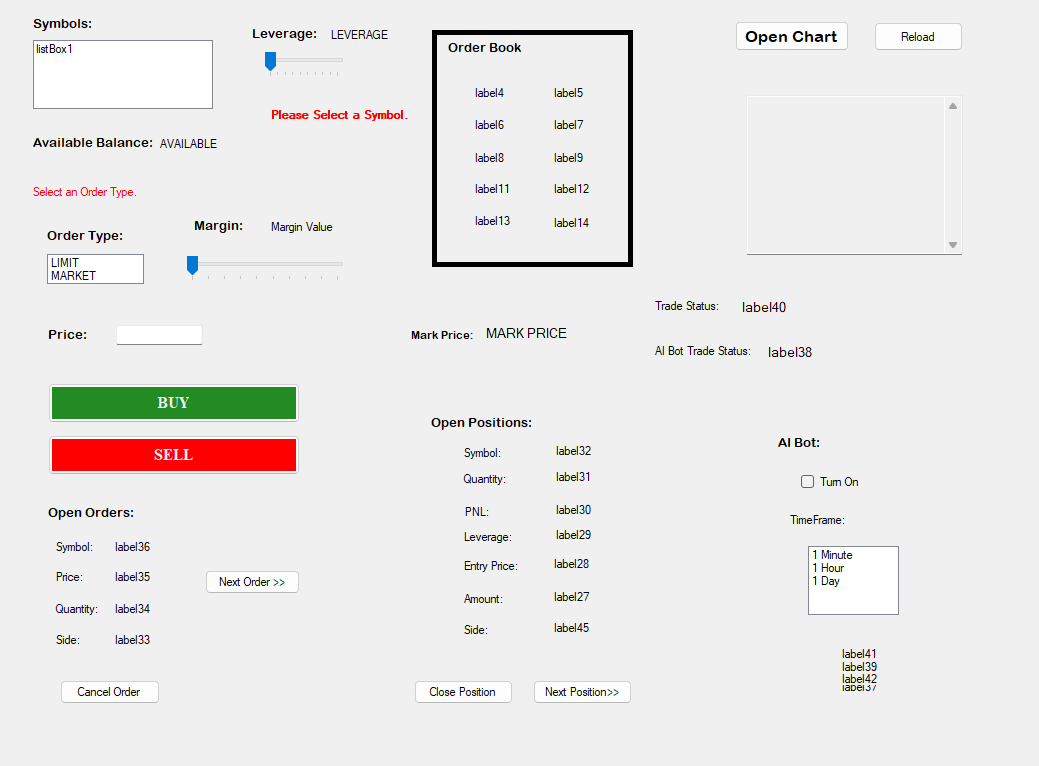
If no trade was open, then nothing happens.

## **Next Position:**



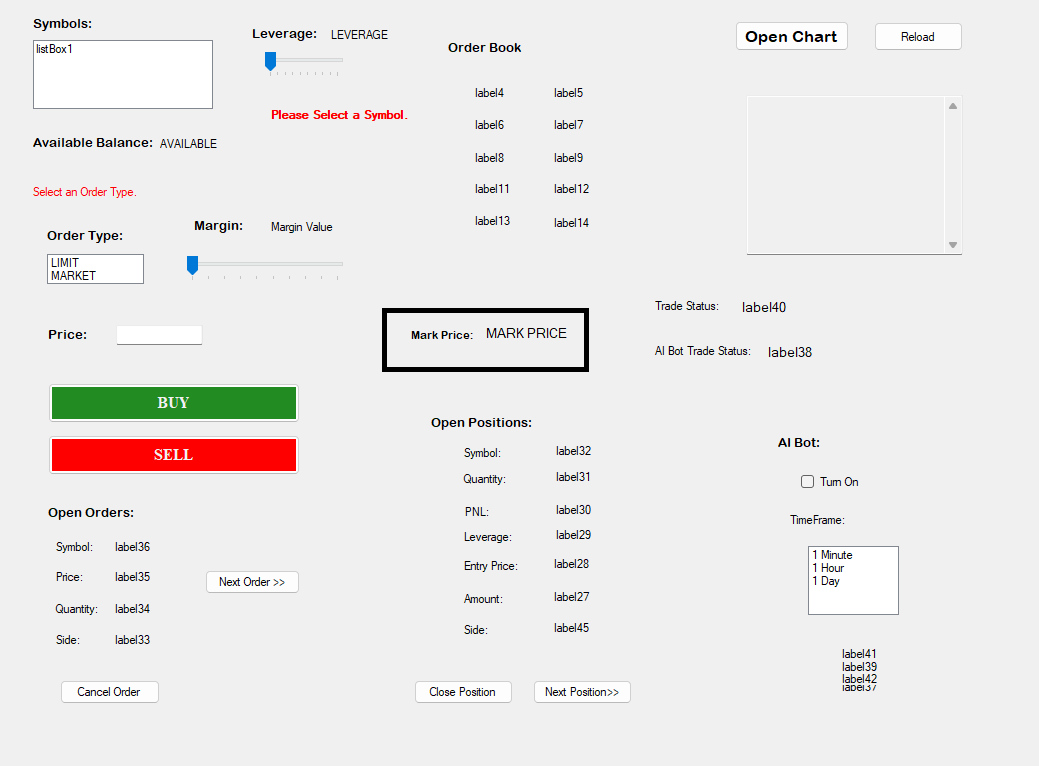
This button makes a call to the positionsInitializer() function after incrementing 6 into the integer positionsStartIndex variable. This positionsStartIndex tells the positionsInitializer() which order to display. If there is only one open position, then it is not changed. If there are no open orders, then nothing happens.

## **Order Book:**



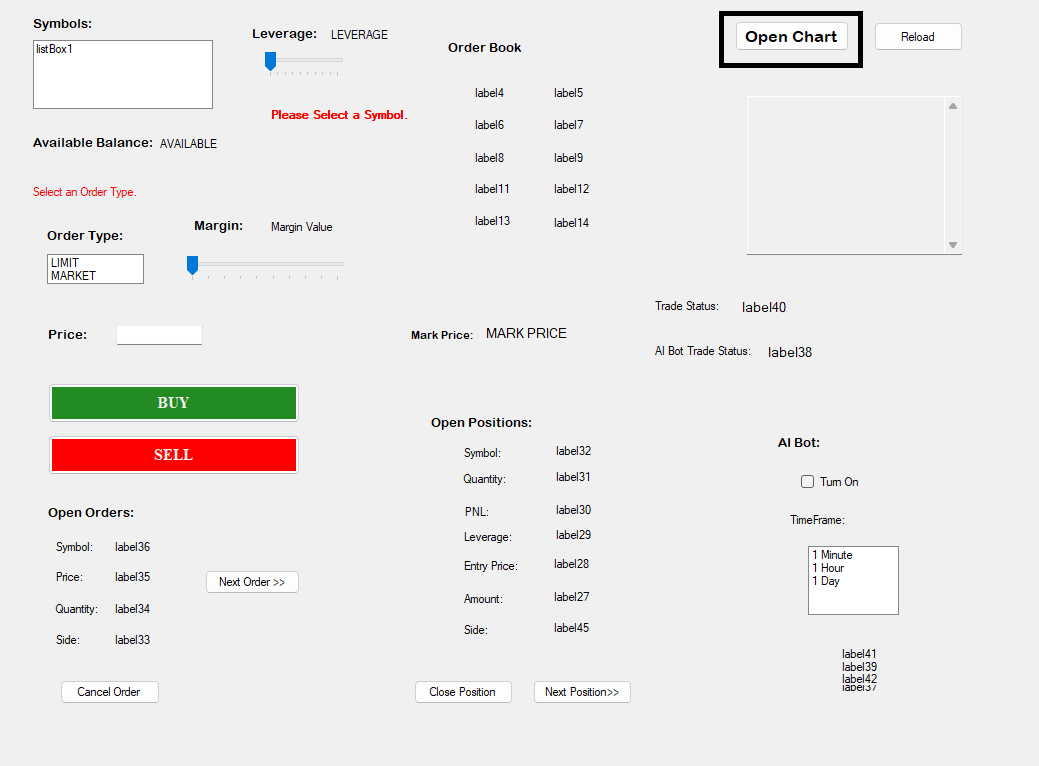
It basically displays the order book to the user. It runs only on the thread and is updated after every 15 seconds. It is updated only after a valid symbol has been selected. When a valid symbol is selected. The condition becomes true and the function updateTheOrderBookOrdersAndPositions() is invoked from the thread and fetchOrderBook() is called and the selected symbol is passed to it. This sends a request to /fapi/v1/depth with the symbol and a parameter called limit. Which basically defines how many values you want. So, we have selected the smallest which is 5. It sends this URL to handleGetRequest(), parses the response, stores it in a string vector and returns it. If there was an error and the Order Book couldn’t be fetched, then it stores a predefined error into the vector and returns it to the back. Now the frontend receives this vector, checks for the error, if it is not present then it displays the values of the vector onto the screen. All the actions are logged as well.

## **Market Price:**



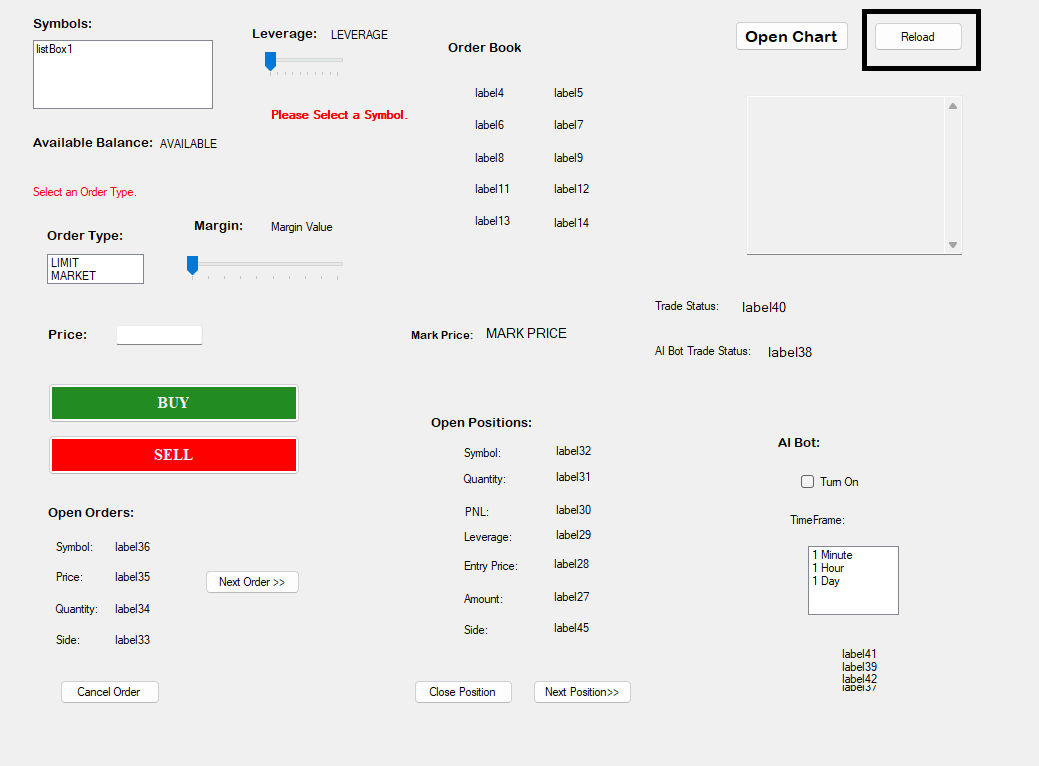
It’s working is done inside the thread as well. It is updated every 15 seconds only if a valid symbol is selected. We first create a variable outside of the thread and initialize it with -1. Then if the conditions are true, the function updateMarkPrice() is invoked from the custom thread and it calls the fetchRealTimeValue() function with the selected symbol. This function makes a request to the /api/v3/ticker/price/ with the selected symbol. Passes this URL to the handleGetRequest(). It parses the JSON and returns a string, which contains the symbol’s current price if the operation was successful otherwise an error msg. The frontend receives this string, and if it doesn’t contain the error message, displays the value and stores the same value in the variable if the variable that we defined earlier is equal to -1. If yes, then it stores the current value in it. And if it isn’t equal to -1, then it compares the current and previous values. If the current is greater than it is displayed in green otherwise, in red. All the actions are logged.

## **Open Chart Button:**



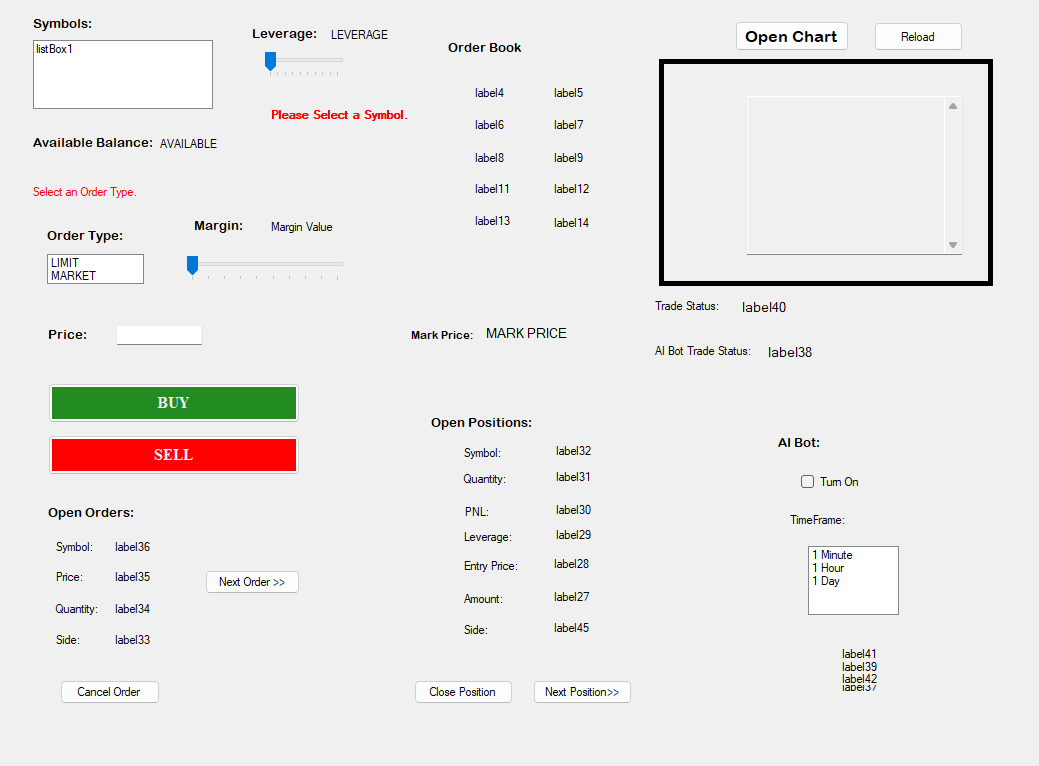
Clicking this button opens a chart of that symbol. It is only enabled when a valid symbol is selected. We are using the services of gocharting.com. User can view the chart of the symbol in any time frame that they want and also draw on it etc. It also offers a lot of basic indicators. It uses the Start() function defined in the process.

## **Reload Button:**



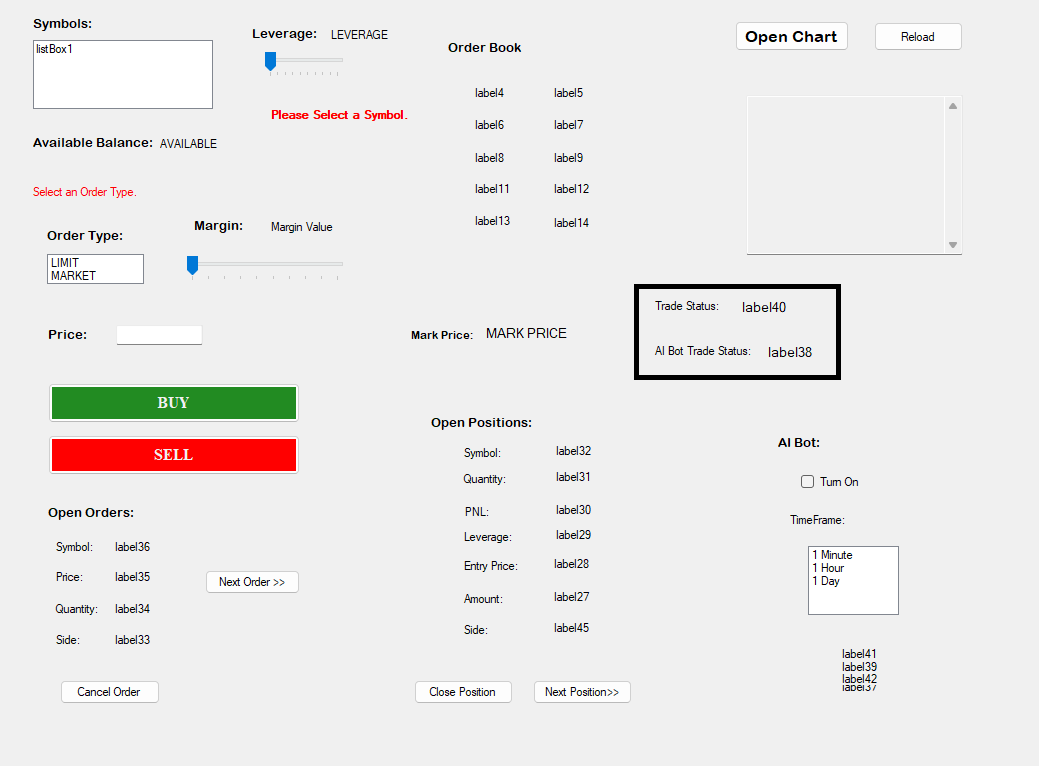
Restarts the whole program. It first creates a new instance of the MyForm object. It then closes the original form and open the new one.

## **On-Screen Logs:**



This is a textbox with multiline support enabled. Every time we want to log something we append it into its existing value. We also have an event handler defined on its TextChanged() event. Which is invoked every time text within it changes. In it we have two lines of code that basically tells it to scroll down. It basically changes its views starting length to the length of the text in it. And then tells it to ScrollToCaret().

## **Status Bars:**

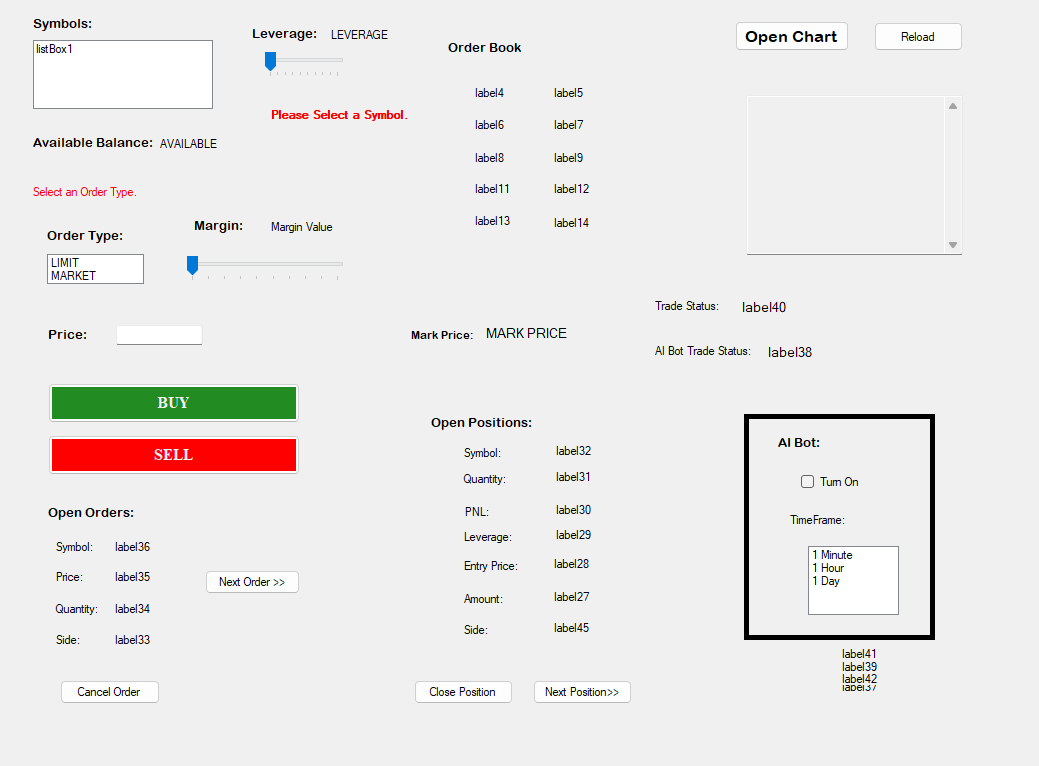


The “Trade Status” shows the status related to the user’s trades. The “AI Bot Trade Status” tells us details about the trades made by the AI. These are just regular labels.

## **How are we making the text in the status bar disappear?**

What we do is that when the updateMarkPrice() is called from our custom thread, we first empty these status bars by assigning them null strings. In this way, the text in the status bar will get cleaned up after at most 15seconds but it can also disappear without the user even being able to see it because in the rare case that the user’s trade’s message appears exactly before the execution of our custom thread, the text will vanish without the user seeing it.

## **AI Bot:**



The “Turn On” check box only gets enabled when a valid symbol is selected. And the TimeFrame ListBox gets enabled only when the checkbox is checked. When a timeframe is selected, a dialog box appears telling the user, that the setting up of the AI may take some time. If the user selects no, then nothing happens. But are the user selects yes then another message box appears telling the user that AI will take 20 seconds to set up.

Once that is done, the event handler makes a call to the fetchHistoricalData() function and passes it the selected symbol. Which then sends that symbol and a limit of 500 to /fapi/v1/klines. It then parses the JSON, checks it, retrieves the data, stores it in a double vector and sends it back to the frontend if everything went alright, otherwise it returns an empty vector. The frontend receives it and sends it to the ai() function. Which contains the logic of the Machine Learning Algorithm, more on the logic in its own section. The ai() function after processing, creates a bool vector and stores the values of 4 variables in it. They are:

1. Open Long Trade
2. Close Long Trade
3. Open Short Trade
4. Close Short Trade

If the vector of historical data passed to the ai() function was empty, then it automatically returns a vector containing all false. Otherwise, they contain their respective values, if any of these becomes true, then that respective action is performed for the selected symbol.

### **Logic of the Machine Learning Model:**

**Disclaimer:** Firstly, I would like to give ©jdehorty of TradingView credit for making this script. I have just converted it to C++. This was originally in PineScript, a language very similar to python, syntax-wise.

PineScript, on the other hand, is more limited in terms of data structures. It primarily revolves around time - series data, focusing on organizing and manipulating financial market data such as OHLC(open, high, low, close) prices, volume, and other trading related information. It provides built - in functions and structures specifically tailored for handling financial data, like series, plots, and simple array - like structures for storing historical market data. In PineScript, the series data structure is fundamental for handling time-series data, particularly in the context of financial markets. It represents a sequence of values over time, typically comprising historical price data, indicators, or any numerical information related to trading. The series data structure in PineScript is implicitly managed and is designed to handle time-specific data efficiently.

Comparatively, in C++, there isn't a direct equivalent data structure named series. However, C++ provides various data structures that can be used to represent time-series data, such as arrays, vectors, or custom-designed classes.

The primary difference lies in the implicit handling and functionality tailored specifically for financial time - series data that PineScript's series offers. In PineScript, the series data structure incorporates specific built-in functions and behaviors optimized for handling time-series data, making it easy to apply calculations, indicators, and visualize data within the TradingView platform seamlessly.

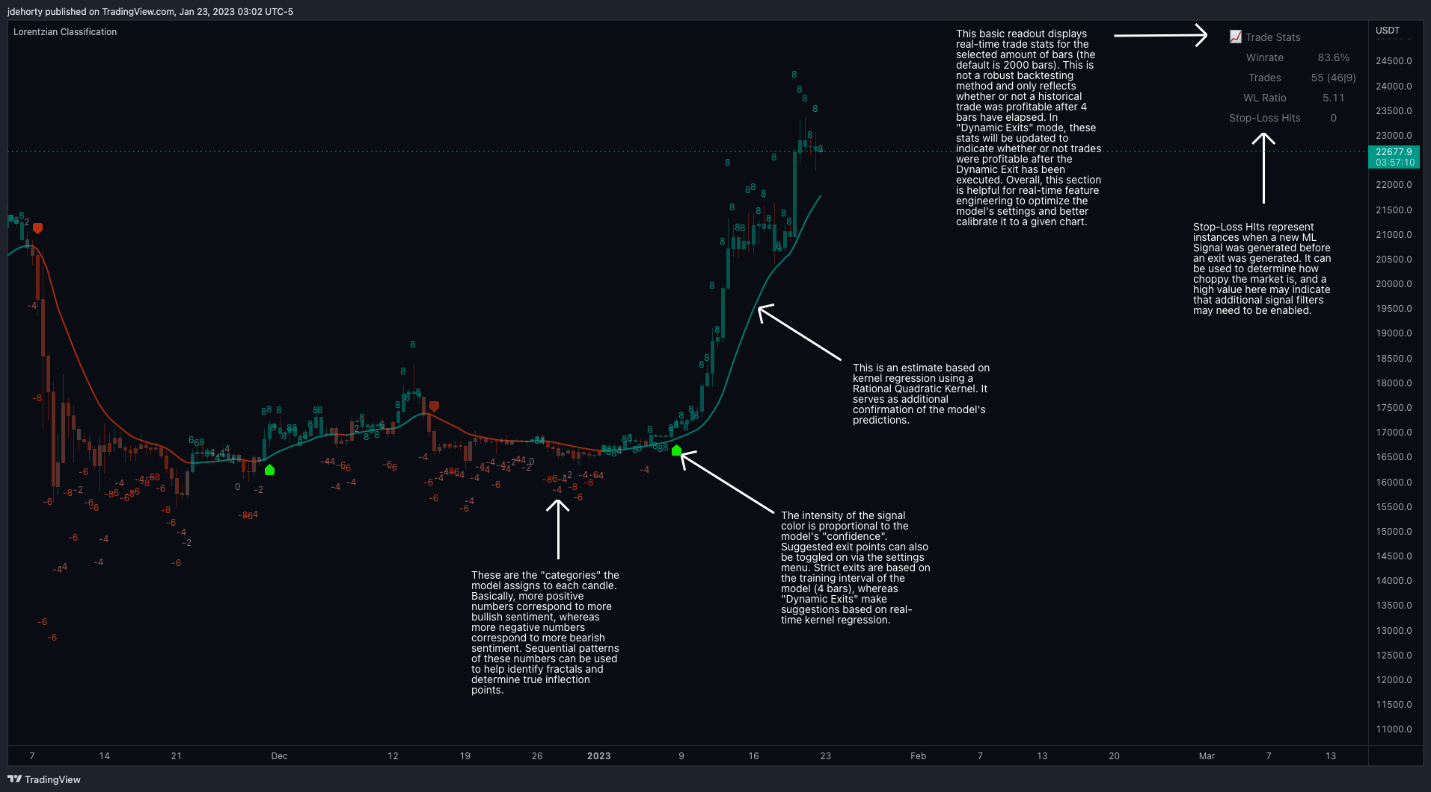
C++ lacks the specialized built - in features and syntax specifically designed for financial time - series data, unlike PineScript. However, C++ offers more flexibility and control over data structures. Developers can create custom classes or structures in C++ to mimic some functionalities of PineScript's series, but it would require more manual implementation and lack the seamless integration that PineScript offers within the TradingView platform for financial analysis.

█ OVERVIEW  
  
 A Lorentzian Distance Classifier (LDC) is a Machine Learning classification algorithm capable of categorizing historical data from a multi-dimensional feature space. This indicator demonstrates how Lorentzian Classification can also be used to predict the direction of future price movements when used as the distance metric for a novel implementation of an Approximate Nearest Neighbors (ANN) algorithm.  
  
█ BACKGROUND  
  
 In physics, Lorentzian space is perhaps best known for its role in describing the curvature of space-time in Einstein's theory of General Relativity (2). Interestingly, however, this abstract concept from theoretical physics also has tangible real-world applications in trading.  
  
 Recently, it was hypothesized that Lorentzian space was also well-suited for analyzing time-series data (4), (5). This hypothesis has been supported by several empirical studies that demonstrate that Lorentzian distance is more robust to outliers and noise than the more commonly used Euclidean distance (1), (3), (6). Furthermore, Lorentzian distance was also shown to outperform dozens of other highly regarded distance metrics, including Manhattan distance, Bhattacharyya similarity, and Cosine similarity (1), (3). Outside of Dynamic Time Warping based approaches, which are unfortunately too computationally intensive for PineScript at this time, the Lorentzian Distance metric consistently scores the highest mean accuracy over a wide variety of time series data sets (1).  
  
 Euclidean distance is commonly used as the default distance metric for NN-based search algorithms, but it may not always be the best choice when dealing with financial market data. This is because financial market data can be significantly impacted by proximity to major world events such as FOMC Meetings and Black Swan events. This event-based distortion of market data can be framed as similar to the gravitational warping caused by a massive object on the space-time continuum. For financial markets, the analogous continuum that experiences warping can be referred to as "price-time".  
  
Below is a side-by-side comparison of how neighborhoods of similar historical points appear in three-dimensional Euclidean Space and Lorentzian Space:

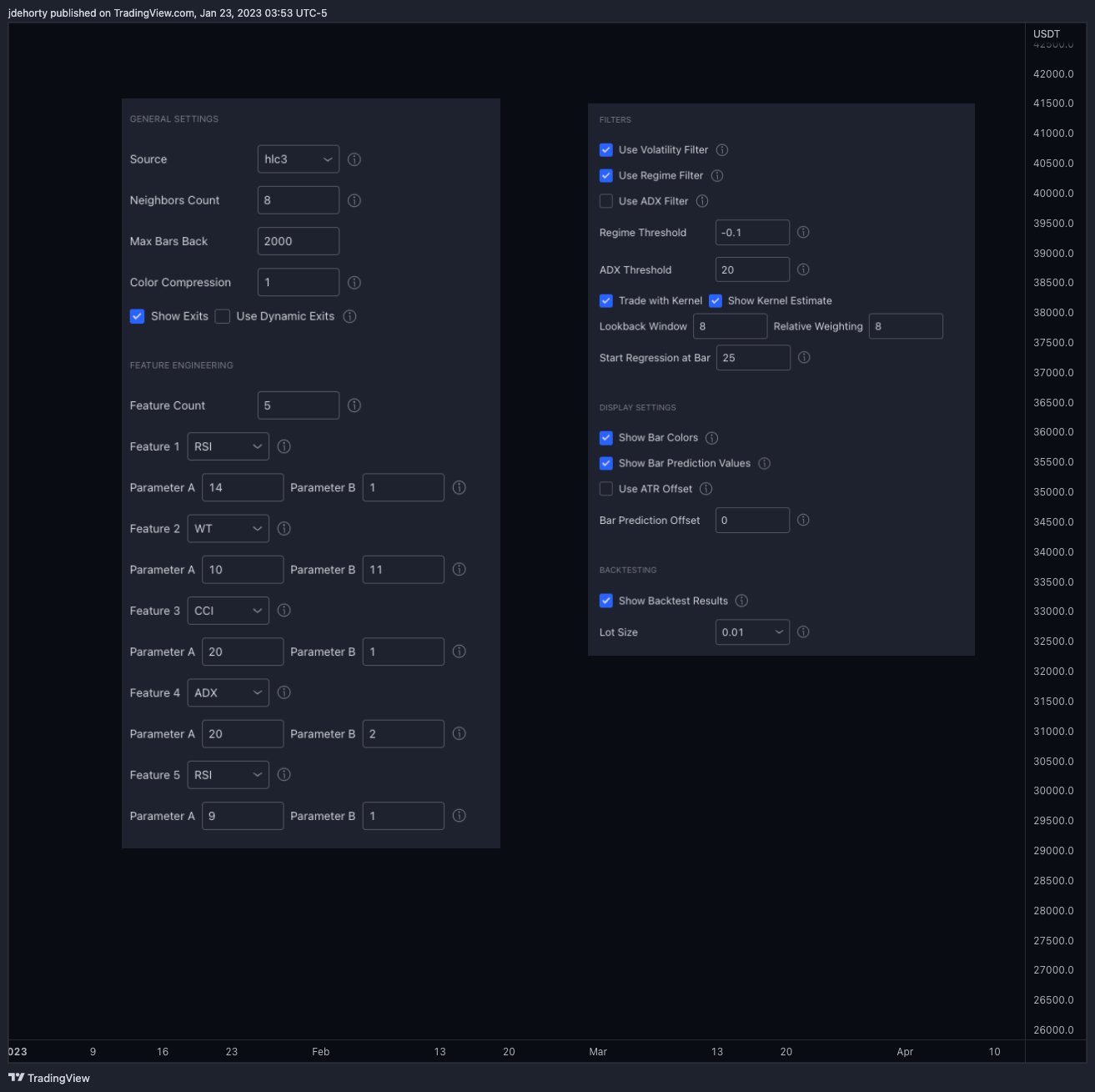


This figure demonstrates how Lorentzian space can better accommodate the warping of price-time since the Lorentzian distance function compresses the Euclidean neighborhood in such a way that the new neighborhood distribution in Lorentzian space tends to cluster around each of the major feature axes in addition to the origin itself. This means that, even though some nearest neighbors will be the same regardless of the distance metric used, Lorentzian space will also allow for the consideration of historical points that would otherwise never be considered with a Euclidean distance metric.  
  
 Intuitively, the advantage inherent in the Lorentzian distance metric makes sense. For example, it is logical that the price action that occurs in the hours after Chairman Powell finishes delivering a speech would resemble at least some of the previous times when he finished delivering a speech. This may be true regardless of other factors, such as whether or not the market was overbought or oversold at the time or if the macro conditions were more bullish or bearish overall. These historical reference points are extremely valuable for predictive models, yet the Euclidean distance metric would miss these neighbors entirely, often in favour of irrelevant data points from the day before the event. By using Lorentzian distance as a metric, the ML model is instead able to consider the warping of price-time caused by the event and, ultimately, transcend the temporal bias imposed on it by the time series.  
  
 For more information on the implementation details of the Approximate Nearest Neighbors (ANN) algorithm used in this indicator, please refer to the detailed comments in the source code.

█ HOW TO USE THIS ON TRADING VIEW  
  
Below is an explanatory breakdown of the different parts of this indicator as it appears in the interface:



Below is an explanation of the different settings for this indicator:



##### **General Settings:**

* Source - This has a default value of "hlc3" and is used to control the input data source.
* Neighbors Count - This has a default value of 8, a minimum value of 1, a maximum value of 100, and a step of 1. It is used to control the number of neighbors to consider.
* Max Bars Back - This has a default value of 2000.
* Feature Count - This has a default value of 5, a minimum value of 2, and a maximum value of 5. It controls the number of features to use for ML predictions.
* Colour Compression - This has a default value of 1, a minimum value of 1, and a maximum value of 10. It is used to control the compression factor for adjusting the intensity of the colour scale.
* Show Exits - This has a default value of false. It controls whether to show the exit threshold on the chart.
* Use Dynamic Exits - This has a default value of false. It is used to control whether to attempt to let profits ride by dynamically adjusting the exit threshold based on kernel regression.

**Feature Engineering Settings:**  
 Note: The Feature Engineering section is for fine-tuning the features used for ML predictions. The default values are optimized for the 4H to 12H timeframes for most charts, but they should also work reasonably well for other timeframes. By default, the model can support features that accept two parameters (Parameter A and Parameter B, respectively). Even though there are only 4 features provided by default, the same feature with different settings counts as two separate features. If the feature only accepts one parameter, then the second parameter will default to EMA-based smoothing with a default value of 1. These features represent the most effective combination I have encountered in my testing, but additional features may be added as additional options in the future.

* Feature 1 - This has a default value of "RSI" and options are: "RSI", "WT", "CCI", "ADX".
* Feature 2 - This has a default value of "WT" and options are: "RSI", "WT", "CCI", "ADX".
* Feature 3 - This has a default value of "CCI”, and options are: "RSI", "WT", "CCI", "ADX".
* Feature 4 - This has a default value of "ADX”, and options are: "RSI", "WT", "CCI", "ADX".
* Feature 5 - This has a default value of "RSI" and options are: "RSI", "WT", "CCI", "ADX".

**Filters Settings:**

* Use Volatility Filter - This has a default value of true. It is used to control whether to use the volatility filter.
* Use Regime Filter - This has a default value of true. It is used to control whether to use the trend detection filter.
* Use ADX Filter - This has a default value of false. It is used to control whether to use the ADX filter.
* Regime Threshold - This has a default value of -0.1, a minimum value of -10, a maximum value of 10, and a step of 0.1. It is used to control the Regime Detection filter for detecting Trending/Ranging markets.
* ADX Threshold - This has a default value of 20, a minimum value of 0, a maximum value of 100, and a step of 1. It is used to control the threshold for detecting Trending/Ranging markets.

**Kernel Regression Settings:**

* Trade with Kernel - This has a default value of true. It is used to control whether to trade with the kernel.
* Show Kernel Estimate - This has a default value of true. It is used to control whether to show the kernel estimate.
* Lookback Window - This has a default value of 8 and a minimum value of 3. It is used to control the number of bars used for the estimation. Recommended range: 3-50
* Relative Weighting - This has a default value of 8 and a step size of 0.25. It is used to control the relative weighting of time frames. Recommended range: 0.25-25
* Start Regression at Bar - This has a default value of 25. It is used to control the bar index on which to start regression. Recommended range: 0-25

**Back testing Settings:**

* Show Backrest Results - This has a default value of true. It is used to control whether to display the win rate of the given configuration.

**Note on "Use Worst Case Estimate" in the Trade Stats section:**  
 As I've stated several times in the comments and documentation, the purpose of the Trade Stats section is to provide real-time feedback during Feature Engineering, and it is NOT meant to be used as a substitute for proper back testing. This section was designed around the premise that this indicator is best used as a source of confluence to traditional TA, and thus, the calculations in this section by default attempt to estimate performance based on mid-bar entries (i.e. not necessarily waiting for the close of the bar as confirmation). On large timeframes, well-calculated mid-bar entries can prevent one from missing out on large moves, and thus, the default settings for the trade stats section are designed to reflect this intended use case. Despite this intended usage, there are some users who prefer to try to evaluate the indicator as a standalone strategy and try to use this section as a back test. Although this is not recommended, the "Use Worst Case Estimate" option has been added to allow these users to more closely match traditional back test results by waiting for the close of the bar as confirmation before entering/closing a trade. When this option is enabled, it is recommended to also enable additional filters such as EMA and SMA to help reduce the number of trades and thus, reduce the impact of the additional latency introduced by waiting for the close of the bar as confirmation.  
  
**Note on back test accuracy over time:**  
 Some users have observed that the accuracy for the back test results tend to increase over time. This behaviour is to be expected as the indicator will become more accurate as more historical data is added to the training set. More historical points of reference for the ML model means more options when selecting neighbors for the Approximate Nearest Neighbors algorithm. This effect should also be kept in mind when evaluating the back test results for this indicator.  
  
**Note on "repainting":** To be clear, once a bar has closed, this indicator will NOT repaint. This is true for both the ML predictions and the Kernel estimate.

Results of Testing the Trading Bot:



Detail of these results is also shown in “AI Trades.xlsx” file.

**Limitations:**  
We have basically created a separate thread that is used to fetch some data after every 15 seconds.

1. For trading we need actual real-time data. Using such (15 seconds) old data in trading is impractical so we need to somehow make it real-time.
2. After every 15 seconds when that thread runs, (and it takes about a second to complete) the program halts during that time because the UI thread of the WinForms application stops executing during that duration.
3. UI could use some designer level beautification.
4. It still lacks an integrated candle stick chart on with which the user can interact.
5. We had to “debuff “ the AI so that its processing time was cut down to 20s. Originally, we planned to provide it with 9000 candles as trading data but the estimated time for that was in days. So now we are providing it about 250 candles as training data. Part of the reason for taking so much time was our thread. But the main reason for it was that the AI function has a time complexity of (O) to the power n. It is an exponential function.
6. The other thing is that the cancel order and close position buttons will not necessarily close the specific position or order that you want. It is programmed to close the one that is being displayed but the problem is that that the close position or order function fetches the list of positions/orders again and there is a chance that if the data sent by Binance back this time would have the positions/orders in different order than before. Essentially, providing us with a chance that the wrong trade/order may be closed.
7. The text of the Trade Status vanishes as soon as our custom thread runs. That means that if you do something and a status msg appears and immediately after our custom thread runs. There will be no time for you to see it and the only chance for you to see it may be if you watch the on-screen logs or the logs.txt file.

**Plans for the Future:**

1. We plan on making it more robust, more efficient and more ‘real-timely’ updated to provide a dependable trading environment for the user.
2. We plan on including the following features:
   1. More Order Types
   2. Lower Margin Restriction. So that poor people can also trade
   3. Allow greater customization options including setting TP/SL, liquidation prices etc.
3. Including a built-in interactable candle stick chart to provide comfort to user.
4. Further GUI Beautification
5. Putting the AI on a remote computer and then accessing it to make trades, to further cut down on time and to increase efficiency.
6. Making sure the intended trade/order gets closed.
7. Separating the status msg vanishing and the custom thread logic.

**Features:**

1. Allows users to place limit and market orders with varying leverage and margin.
2. View charts of the respective symbol
3. Close open positions and orders
4. Displays mark price and updates critical information regularly.
5. Displays on-screen logs.
6. Displays colorful helpful messages.
7. Stores all of the trades whether closing/opening of positions/orders in a csv file.
8. Stores detailed messages for the developer in a txt logs file. These logs are stored after every network request whether it is successful or not.
9. Has an env file to store the API and SECRET keys of the user.
10. Has an AI with many different options and is capable of placing trades on its own?
11. The AI is trained on current data in real-time.

**Challenges:**

1. Using Multithreading in WinForms was a NIGHTMARE.
2. Making Candlestick Charts
3. Translating the AI
4. Error Handling for Network Requests
5. JSON Parsing