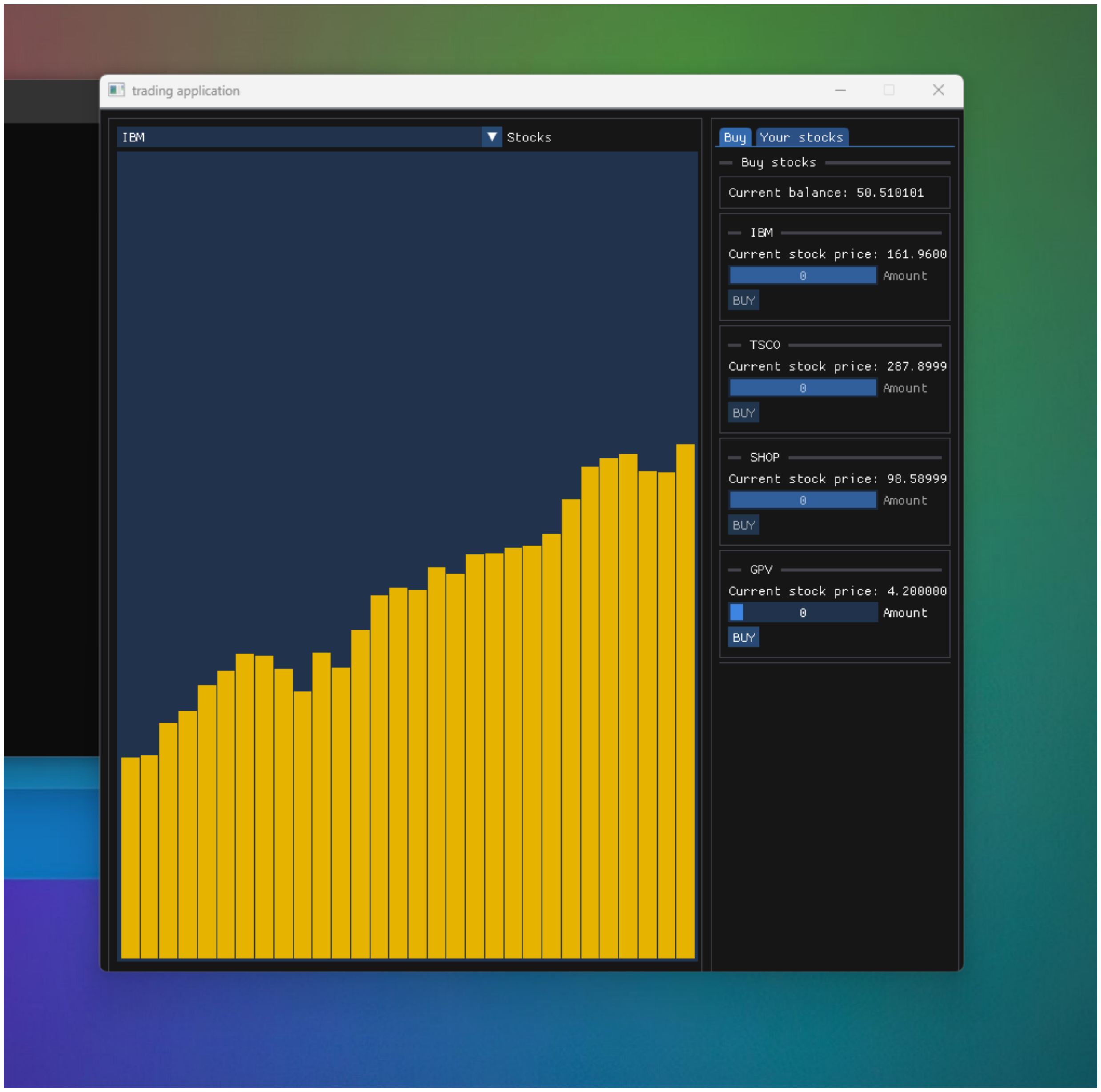
STOCK TRADING APPLICATION IN C++



Current balance: 50.510101

Current stock price: 161.9600
0

Current stock price: 287.8999
0

Current stock price: 98.58999
0

Current stock price: 4.200000
0

Version 1.0 • 12.12.2023

CONTENT

INTRODUCTION

“Trading application” is an application made in C++ programming language for trading stocks and tracking user’s assets. Its main purpose is to simulate a realistic stock trading experience where new data is received, processed, and displayed for users in histograms and menus, where revenue is calculated, and stocks can be bought or sold. The functionality of application can be divided in 3 parts:

1. Receive data from a website containing financial data about stocks.
2. Parse data into usable information for the application.
3. Using parsed information, generate the user interface.

User interface includes elements such as:

- Histograms.
- Dropdown menu for selecting different histograms.
- Buy menu.
    - Stock name
    - Current price
    - Sliders to control the number of stocks to buy.
    - Buy button.
- Sell menu.
    - Stock amount bought by user for each stock.
    - Sell button.
    - Total revenue of stocks user is holding.

USED LIBRARIES

Total of 4 libraries were used for the application:

- **cURL** – open-source library that allows to fetch data using URLs. For this application it is used to perform a HTTP GET request to a stock information website and retrieve the response data.
- **rapidJSON** – open-source library for parsing JSON data in C++. For this application it parses JSON data received from cURL and extracts specific stock data which is used to create Stock class objects.
- **SFML** – open-source graphics library for creating 2D graphics with functionality for window creation, graphics rendering, audio playback, and user input. For this application it is used in combination with Dear Imgui to create a window where UI elements are rendered.
- **Dear Imgui** – Open-source graphical user interface (GUI) library for C++. For this application it is responsible for most of the user interface layout, logic, and functionality.

USE OF CURL IN CODE

A total of 2 methods were created using cURL library: writeFunction and fetchData.

fetchData(): Goal of this method is to fetch data and store it in a string which is returned at the end. The method receives URL as parameter. A pointer to cURL variable is created. After that cURL is initialized and inside if statement URL is passed to cURL, writeFunction is called to handle data that is received during a HTTP request. After data is received it is stored in a response string which is returned at the end of function. The rest of code is cleaning up cURL handle. If statement also has error protection that prints out an error in console if it couldn't fetch data.

writeFunction(): This method is responsible for writing received data in a string and returning how many bytes were processed. It receives parameters such as pointer to the received data buffer, size of each data element, number of data elements received and pointer to where data will be stored.

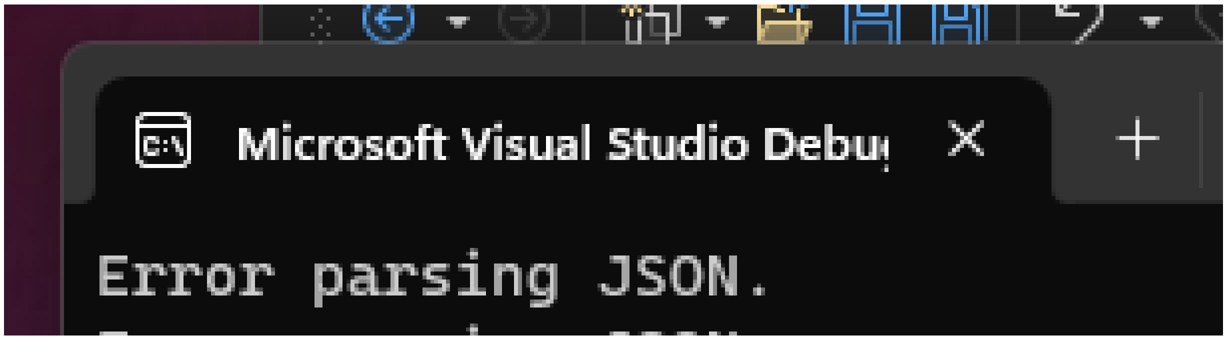
Example of cURL being used in code:

```
string IBM = fetchData(IBMurl);
```

USE OF RAPIDJSON IN CODE

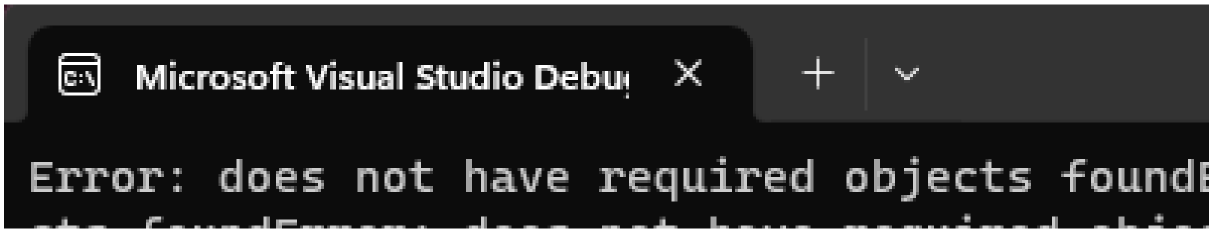
Only 1 method was created for rapidJSON but it is one of the most important ones in the code: parseData().

parseData(): The goal of the method is to parse string received from cURL and create a Stock object with values like date, price and name of the stock. 3 variables are passed to the function as arguments, first contains cURL response string, second is a vector array which contains Stock objects and third contains size of an array which is used to limit how much Stocks are created. A rapidJSON document is created, which then is used to parse received cURL response string. The method checks if there is a parse error in which case it prints out the error in the console and returns false.



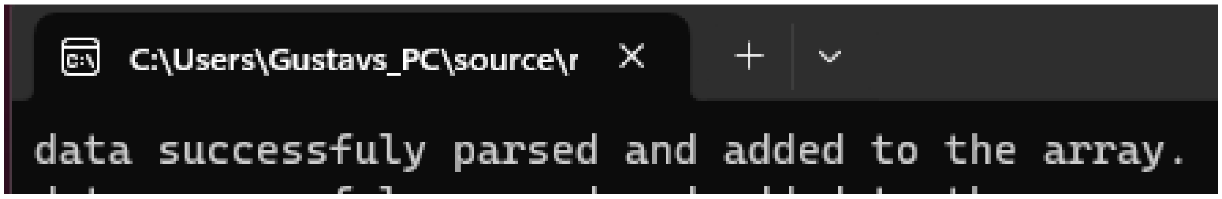
```
Error parsing JSON.
```

Next, the function checks if document contains needed members for the Stock class. If true it reads date, price, name, high and low from the document and creates a Stock class object which is added to the stock array. If false it prints out an error that there weren't the right objects found and returns false.



```
Error: does not have required objects found
```

If all is completed without any errors, message is printed in console that states all was parsed successfuly and function returns true.



```
data successfuly parsed and added to the array.
```

Examples of rapidJSON being used in code:

```
bool data1 = parseData(IBM, stocks, max);

if (!data1 || !data2 || !data3 || !data4) {
    return 0;
}
```

USE OF SFML IN CODE

No additional methods were created for SFML so here are basic elements of SFML that were used. Its main purpose was to create window where Imgui UI elements could be rendered.

RenderWindow: used to create a window where application is rendered. Size of window was fixed to 800x800 pixels, "trading application" name given and only close button enabled for its title bar.

```
sf::RenderWindow window(sf::VideoMode(800, 800), "trading application", sf::Style::Close);
```

Window got initialized by Imgui-SFML library for it to be able to use it.

```
ImGui::SFML::Init(window);
```

When window is open it is being updated every frame using Update() function. Once window gets a command to close, it stores necessary variables in a txt file and window is closed:

```
sf::Clock deltaClock;
 while (window.isOpen()) {
     //close window event
     sf::Event e;
     while (window.pollEvent(e)) {
         ImGui::SFML::ProcessEvent(window, e);
         if (e.type == sf::Event::Closed) {
             writeVariablesToFile(   balance,
                                     moneySpentIBM,
                                     moneySpentTSCO,
                                     moneySpentSHOP,
                                     moneySpentGPV,
                                     IBMAmount,
                                     TSCOAmount,
                                     SHOPAmount,
                                     GPVAmount
                                 );
             window.close();
         }
     }
     ImGui::SFML::Update(window, deltaClock.restart());
```

Before frame is rendered window is cleared of old elements, after which Imgui UI elements and SFML window elements are rendered.

```
     window.clear(sf::Color(129, 133, 137));
     ImGui::SFML::Render(window);
     window.display();
```

Once the window is closed Imgui-sfml library closes the window.

```
ImGui::SFML::Shutdown();
```

USE OF IMGUI IN CODE

Imgui is by far one of the most important libraries for this application since it revolves around visualizing data in histograms and menus. Here are the most important elements and functions used to achieve creation of user interface for this application.

IMGUI MAIN WINDOW

Default position of Imgui window is set to 0,0 coordinates (top left corner):

```
    ImGui::SetNextWindowPos(ImVec2(0, 0), ImGuiCond_FirstUseEver);
```

The main Imgui window is created using Begin() function, End() function shows end of the created window. This window is where rest of the Imgui elements are located in:

```
    ImGui::Begin("Trading Application", nullptr, ImGuiWindowFlags_NoTitleBar |
ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove);
    ...
    ImGui::End();
```

HISTOGRAM WINDOW

A child is created for Histogram, it is done so that both histogram and menu can be in a flex layout inside the main Imgui window:

```
    ImGui::BeginChild("Histogram", ImVec2(550, 0.0f), true);
    ...
    ImGui::EndChild();
```

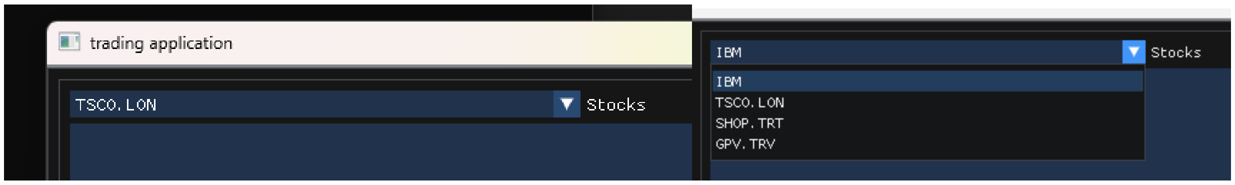
Inside histogram child to populate array used that is used for histogram values, on first open a loop is created which adds prices of default stock (in our case IBM) to the array:

```
int x = 0;
for (int i = stocks.size() - 1; i >= 0; i--) {
    if (stocks[i].getName() == items[item_current_idx]) {
        arr[x] = stocks[i].getPrice();
        x++;
    }
}
```

The initial component on the left side is a dropdown menu. When the user interacts with this menu by opening it, the program checks for any selection made by the user within the dropdown menu. If a selectable item is clicked, the program captures the index of the selected item. In addition to that, it utilizes this index to update the values of an array, which is then passed to a histogram. The updating of values is achieved through a for loop, where the selected item's information is applied to the array elements:

```
if (ImGui::BeginCombo("Stocks", combo_preview_value))
{
    for (int n = 0; n < IM_ARRAYSIZE(items); n++)
    {
        const bool is_selected = (item_current_idx == n);
        if (ImGui::Selectable(items[n], is_selected)) {
            item_current_idx = n;
            int x = 0;
            //Selects histogram data based on selected dropdown option
            int i = stocks.size();
            for (int i = stocks.size() - 1; i >= 0; i--) {
                if (stocks[i].getName() == items[item_current_idx]) {
                    arr[x] = stocks[i].getPrice();
                    x++;
                }
            }
        }

        //Set the initial focus when opening the combo (scrolling + keyboard navigation focus)
        if (is_selected) {
            ImGui::SetItemDefaultFocus();
        }
    }
    ImGui::EndCombo();
}
```

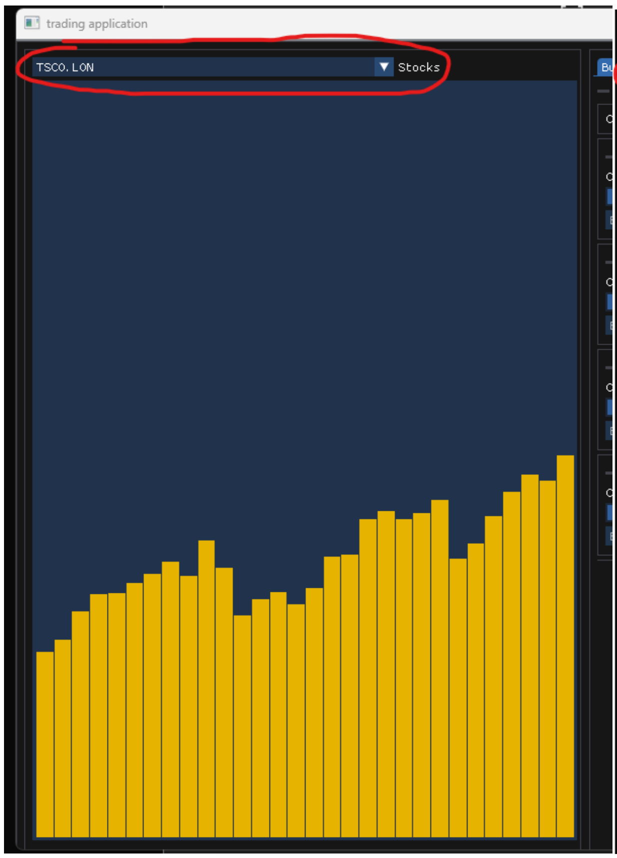
Here is an example of what dropdown menu should look like:

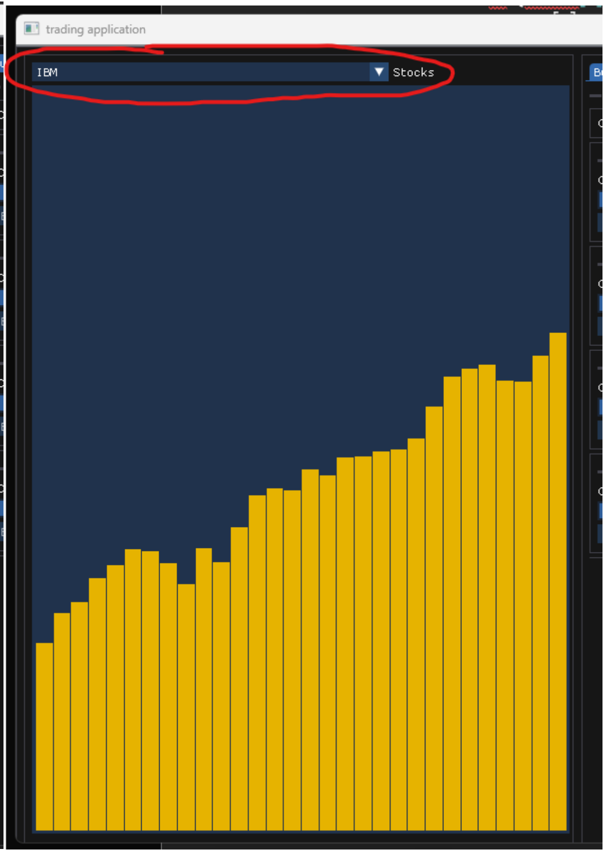


After dropdown menu, the histogram is created based on what is the active dropdown menu item and array that is generated for it. The dimensions of histogram are fixed to 540x750 pixels.

```
ImGui::PlotHistogram("", arr, IM_ARRAYSIZE(arr), 0, NULL, min_val[item_current_idx],
max_val[item_current_idx], ImVec2(540, 750));
```

Here is an example of what histogram looks like and how it changes based on selected item:





MENU WINDOW

SameLine() function is called to make a flex row layout between histogram and menu. Menu window itself is created using the same BeginChild function:

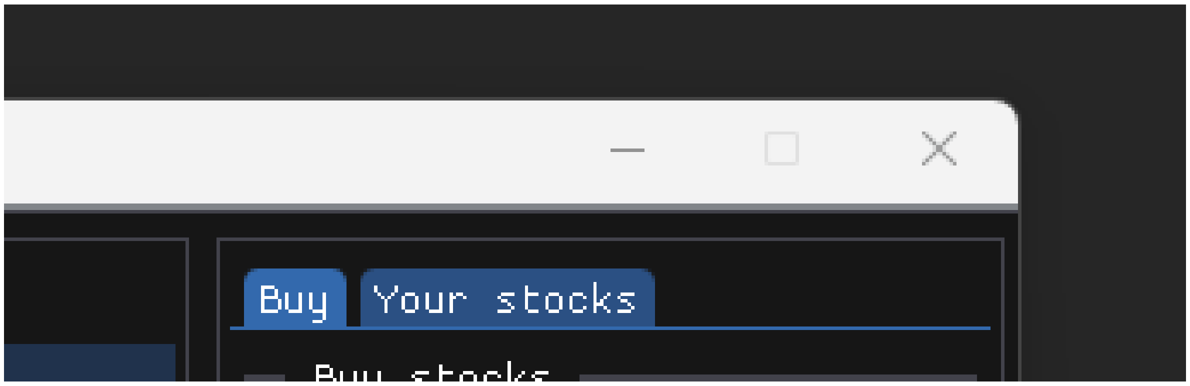
```
ImGui::SameLine();

ImGui::BeginChild("Menu", ImVec2(230, 0.0f), true);
```

```
...

ImGui::EndChild();
```

Inside menu child, two tabs are created “Buy” and “Your stocks” to organize the menu in different sections:

```
ImGui::BeginTabItem("Buy")

...

ImGui::EndTabItem();

ImGui::BeginTabItem("Your stocks")

...

ImGui::EndTabItem();
```

Here is an example of what they look like:

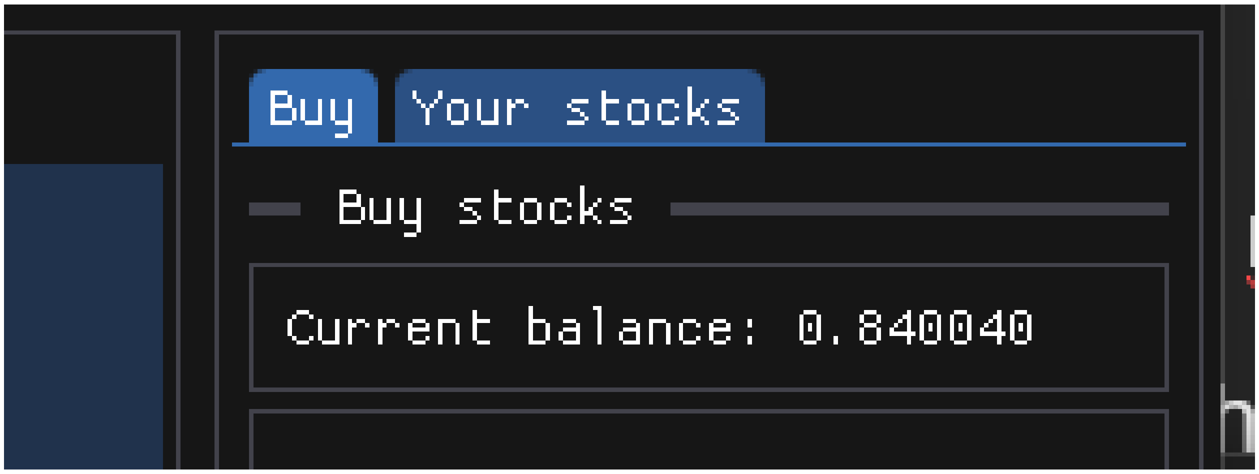


Inside “Buy” tab there are 2 methods being called:

renderBalance(): Goal of this function is to show current balance of the user. Height is set to 30 pixels so that no extra space is taken. It is put into a child because for next function sliders are used and they don’t work correctly while being together.

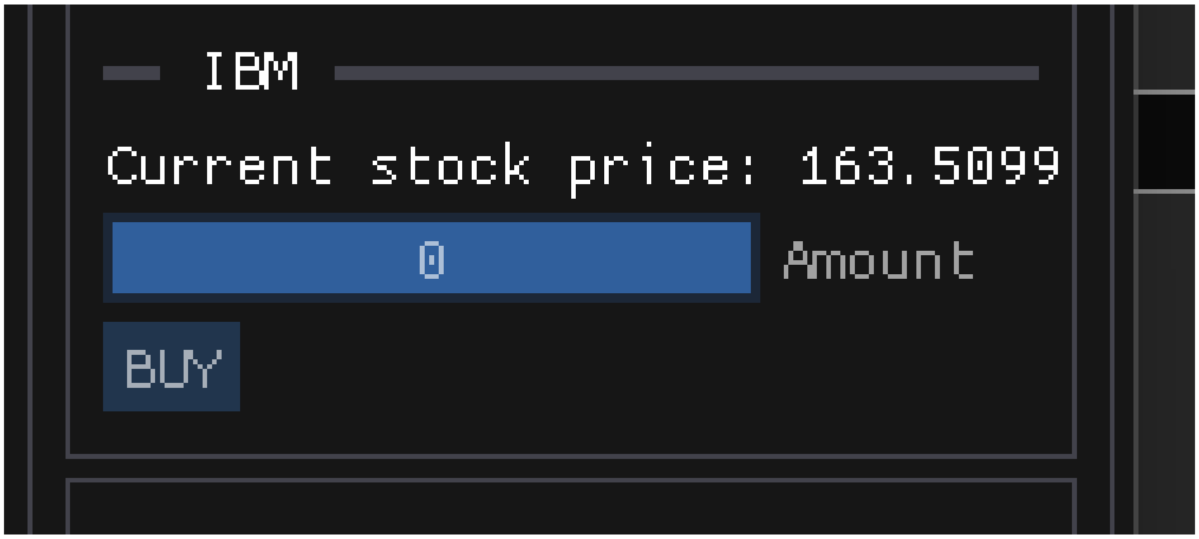
```
void renderBalance(float balance) {
   ImGui::SeparatorText("Buy stocks");
   ImGui::BeginChild("Balance", ImVec2(0.0f, 30), true);
   ImGui::Text("Current balance: %f", balance);
   ImGui::EndChild();
}
```

Here is an example of what renderBalance() method generates when called:



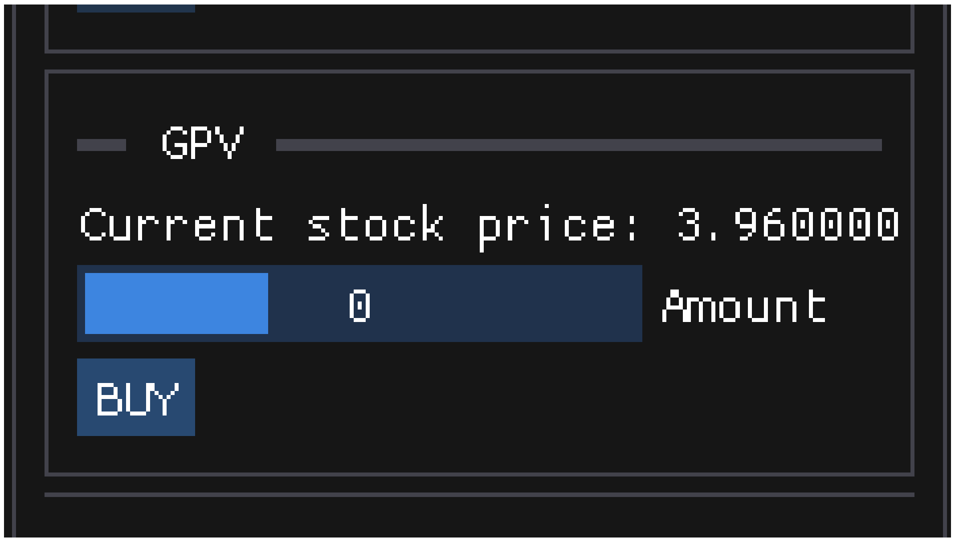
Current balance: 0.840040

renderBuyStockOption(): Goal of this function is to create a subsection in “Buy” tab for a stock which can be purchased. There are total of 6 values passed to the function: name of the stock, how much it costs, slider value (to track how much has been selected), user balance, amount (to track how many stocks have been bought) and variable that tracks how much money has been spent on that stock. First, a child is made so that this element is separated from other elements because sliders do not work correctly when paired with other sliders in the same element. Separator is added for a pretty heading above. After that the current stock price is read and shown. 2 state variables are made, one for tracking if button is pressed and other one to track if slider and button must be disabled. If the used balance is less than the current stock value, then slider and button are disabled.



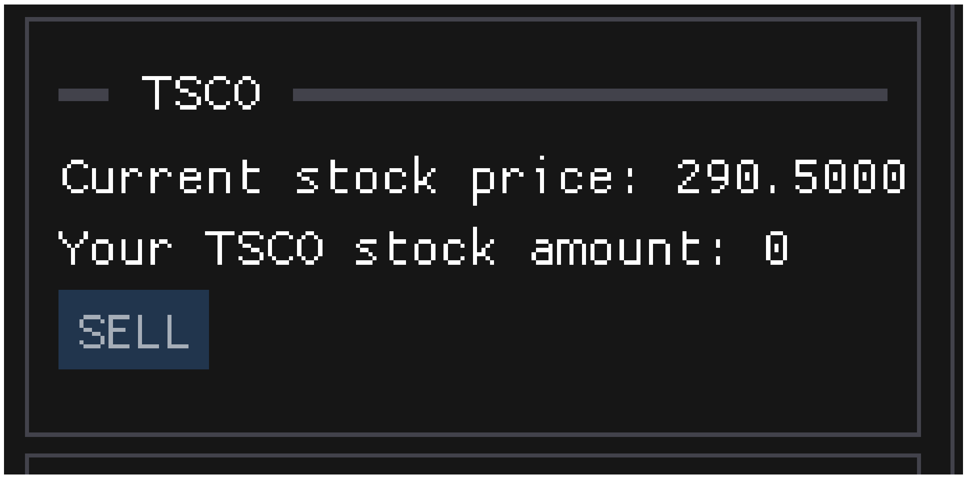
Current stock price: 163.5099

However, if the user has bigger balance than current stock price then slider and buy button are enabled. Slider max value changes based on user balance so that means user clicks on buy button and has selected value more than 0 it will subtract the value total stock value from his current balance, add it to money spent variable, as well as add slider value to the amount of stock he has.

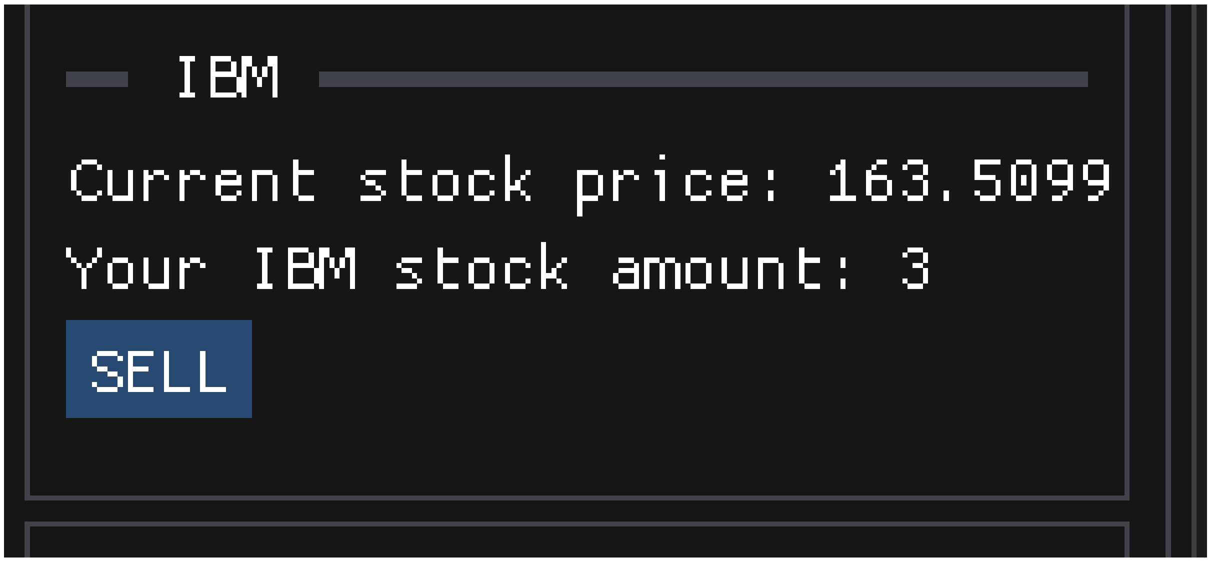


Inside “Your stocks” tab there is 1 method being called:

renderSellStockOption(): Goal of this method is to create a subsection in “Your stocks” tab where stocks bought by user can be sold. A total of 5 variables are passed to the method: name if the stock, how much it costs at this point, user balance, user stock amount (how many stocks of this has he bought) and how much total money was spent on this stock. First, a child is made so that this element is separate from other elements in the “Your stocks” tab. Next, the current stock price and amount of these stocks user owns is displayed. 2 state variables are created, one tracks if button has been pressed and other tracks whether to disable sell button. If statement checks if user has stocks and amount. If he has no stocks, then 0 is displayed as amount and sell button is disabled.



If user has these stocks then sell button is enabled and current amount of stocks he has is displayed. Once sell button is pressed it subtracts current stock price from money spent, stock amount is decreased by 1, and price of the stock is added to the user balance.

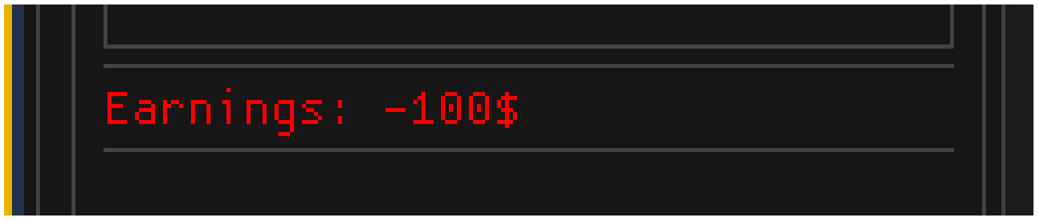


IBM

Current stock price: 163.5099

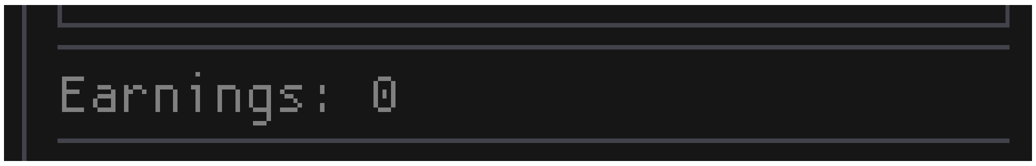
Your IBM stock amount: 3

Last element of the tab is the Earnings element. It is text which displays total earnings for the user. Earnings are calculated using calculateProfit() method which takes in consideration price of the stocks, how much money was spent on them and amount of stocks user owns. If user has earnings less than 0 then Earnings element is created with red color.



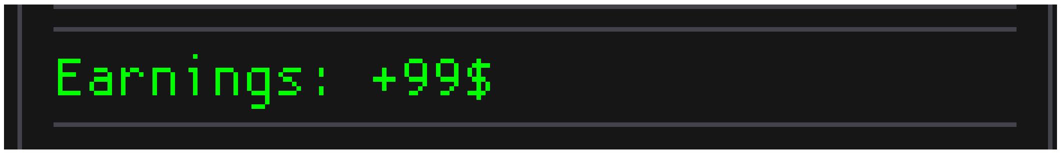
Earnings: -100$

if user has earnings at 0$ then Earnings element is created with grey color.



Earnings: 0

If user has earnings more than 0$ then Earnings element is created with color green.

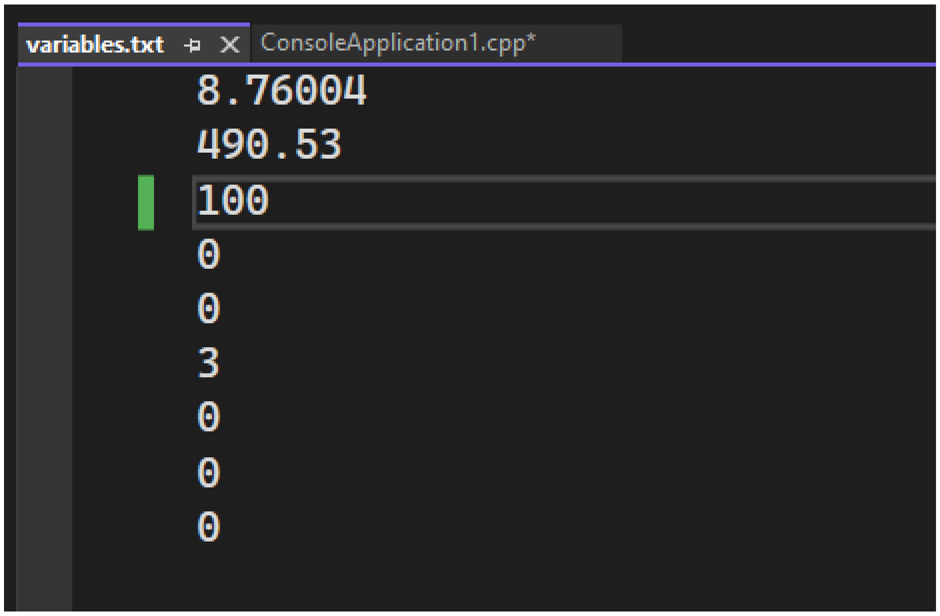


Earnings: +99$

CREATION AND STORING OF VARIABLES

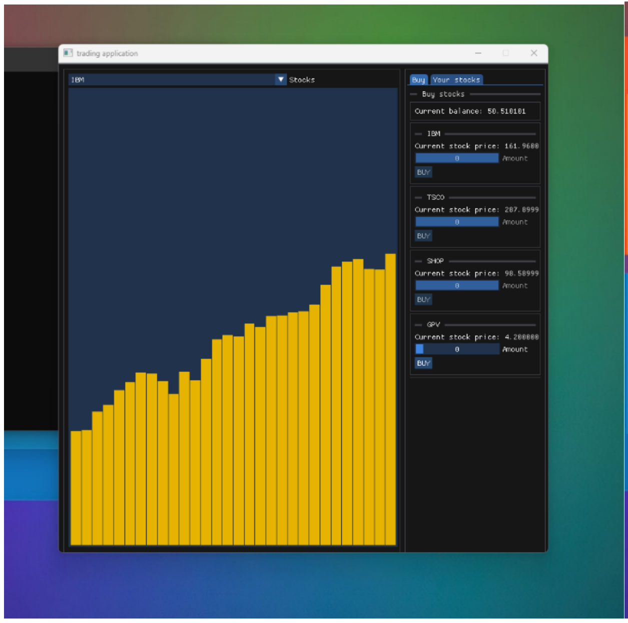
Variables are declared at start of main method. URL strings are declared, data is fetched and parsed into Stock class. Stock class consists of private attributes like, name, price and date. To get these values getPrice() and getName(), getDate() methods can be used. There is section for variables with comment “UI variables” above them. These are the default values of variables for the UI incase methods to read variables don’t work. This executes method readVariablesFromFile() which using fstream library reads a text file and replaces default variable value with previously used value. If there is no text file the application will continue with default values. However, when window is closed, it either generates variables.txt file or overwrites its values with the new session ones.

Here is an example of how txt file looks after closing window:



```
8.76004
490.53
100
0
0
3
0
0
0
```

FINISHED PRODUCT



Buy stocks

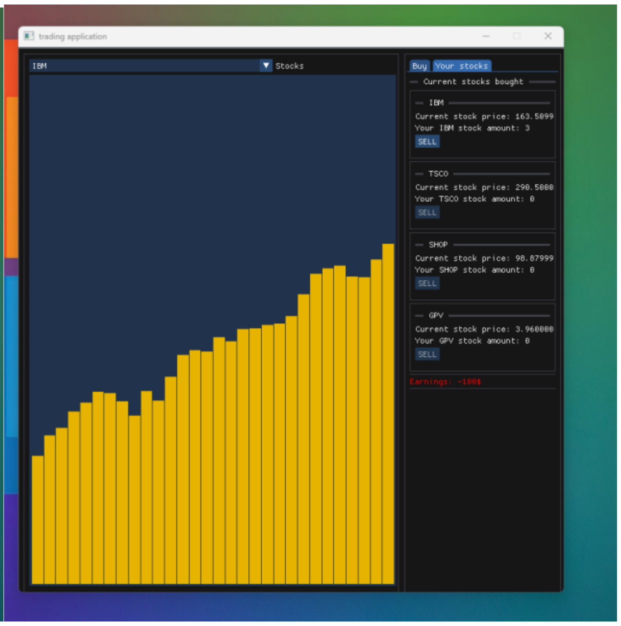
Current balance: 50.510101

IBM
Current stock price: 161.9600
0 Amount

TSCO
Current stock price: 287.8999
0 Amount

SHOP
Current stock price: 98.58999
0 Amount

GPV
Current stock price: 4.200000
0 Amount



Current stocks bought

IBM
Current stock price: 163.5099
Your IBM stock amount: 3

TSCO
Current stock price: 290.5000
Your TSCO stock amount: 0

SHOP
Current stock price: 98.87999
Your SHOP stock amount: 0

GPV
Current stock price: 3.960000
Your GPV stock amount: 0

Earnings: -100$

SOURCE CODE

```
#define CURL_STATICLIB
#include <iostream>
#include <string>
#include <vector>
#include <cmath>
#include <fstream>
#include "SFML/Graphics.hpp"
#include "rapidjson/document.h"
#include "curl/curl.h"
#include "imgui.h"
#include "imgui-SFML.h"
using namespace std;


//Stocks class
```

```
class Stock {
    string name;
    float price;
    string date;
public:
    Stock() { price = 0.0; date = "0000-00-00"; name = "N/A"; }
    Stock(float p, string d, string n) {
        name = n;
        price = p;
        date = d;
    }
    float getPrice() {
        return price;
    }
    string getName() {
        return name;
    }
    string getDate() {
        return date;
    }
};

//Method for parsing cURL response string
typedef vector<Stock> stockArr;
bool parseData(string& data, stockArr& arr, int max) {
    rapidjson::Document document;
    document.Parse<0>(data.c_str());
    if (document.HasParseError()) {
        cout << "Error parsing JSON." << endl;

        return 0;

    }

    //Response string has to contain these objects

    if (document.HasMember("Time Series (Daily)") && document.HasMember("Meta
Data")) {

        const rapidjson::Value& timeSeries = document["Time Series (Daily)"];

        const rapidjson::Value& metaData = document["Meta Data"];

        string name = metaData["2. Symbol"].GetString();

        int count = 0;

        //Iterates through all the correct objects, creates and stores Stock class object in
the passed array

        for (rapidjson::Value::ConstMemberIterator it = timeSeries.MemberBegin(); it !=
timeSeries.MemberEnd() && count < max; ++it, ++count) {

            const string& date = it->name.GetString();

            const rapidjson::Value& values = it->value;

            float price = stof(values["4. close"].GetString());

            float high = stof(values["2. high"].GetString());

            float low = stof(values["3. low"].GetString());

            arr.push_back(Stock(price, date, name));

        }


    }

    else {

        cout << "Error: does not have required objects found";

        return 0;

    }

    cout << "data successfuly parsed and added to the array." << endl;

    return 1;
```

```
}

//Used for current stock price
float getCurrentPrice(stockArr& arr, string name) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr[i].getName() == name) {
            return arr[i].getPrice();
        }
    }
    return 0;
}

//For cURL
static size_t writeFunction(void* ptr, size_t size, size_t nmemb, string* data) {
    data->append((char*)ptr, size * nmemb);
    return size * nmemb;
}
string fetchData(string& url) {
    string response_string;
    CURL* curl;
    curl = curl_easy_init();
    if (curl) {
        curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, writeFunction);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &response_string);
        curl_easy_perform(curl);
        curl_easy_cleanup(curl);
        curl = NULL;
    }
    else {
        cout << "ERROR fetching data.";
    }
    return response_string;
}


void renderBuyStockOption(const char* name, float& price, int& sliderValue, float&
balance, int& amount, float& moneySpent) {
    //It is a child so that sliders don't bug out
    ImGui::BeginChild(name, ImVec2(0.0f, 100), true);
    ImGui::SeparatorText(name);
    ImGui::Text("Current stock price: %f", price);
    bool disabled_button = false;
    bool pressed = false;
    //Does not allow user to buy stocks with insufficient funds
    if (balance < price) { disabled_button = true; }
    if (disabled_button) { ImGui::BeginDisabled(); }
    ImGui::SliderInt("Amount", &sliderValue, 0, floor(balance / price));
    if (ImGui::Button("BUY")) {
        pressed = true;
        if (sliderValue > 0) { balance -= price * sliderValue; moneySpent += price *
sliderValue; amount += sliderValue; sliderValue = 0; }
    }
    if (disabled_button) { ImGui::EndDisabled(); }
    if (pressed) { pressed = false; }
    ImGui::EndChild();
}
```

```
void renderSellStockOption(const char* name, float& price, float& balance, int& amount,
float& moneySpent) {
    //Child because layout might bug the proporions
    ImGui::BeginChild(name, ImVec2(0.0f, 100), true);
    ImGui::SeparatorText(name);
    ImGui::Text("Current stock price: %f", price);
    ImGui::Text("Your %s stock amount: %d", name, amount);
    bool disabled_button = false;
    bool pressed = false;
    if (amount <= 0) { disabled_button = true; }
    if (disabled_button) { ImGui::BeginDisabled(); }
    if (ImGui::Button("SELL")) {
        pressed = true;
        moneySpent -= moneySpent / amount;
        amount--;
        balance += price;
    }
    if (disabled_button) { ImGui::EndDisabled(); }
    if (pressed) { pressed = false; }
    ImGui::EndChild();
}


void renderBalance(float balance) {
    ImGui::SeparatorText("Buy stocks");
    ImGui::BeginChild("Balance", ImVec2(0.0f, 30), true);
    ImGui::Text("Current balance: %f", balance);
    ImGui::EndChild();
}
//Writes variables in txt file so that they can be used next time this application is opened
```

```
void writeVariablesToFile(float& balance,
                          float& moneySpentIBM,
                          float& moneySpentTSCO,
                          float& moneySpentSHOP,
                          float& moneySpentGPV,
                          int& IBMAmount,
                          int& TSCOAmount,
                          int& SHOPAmount,
                          int& GPVAmount) {
    std::ofstream outFile("variables.txt");
    if (outFile.is_open()) {
        outFile << balance << endl;
        outFile << moneySpentIBM << endl;
        outFile << moneySpentTSCO << endl;
        outFile << moneySpentSHOP << endl;
        outFile << moneySpentGPV << endl;
        outFile << IBMAmount << endl;
        outFile << TSCOAmount << endl;
        outFile << SHOPAmount << endl;
        outFile << GPVAmount << endl;
        outFile.close();
        cout << "Variables stored in the file." << endl;
    }
    else {
        cout << "Unable to open the file for writing." << endl;
    }
}
//Reads varaibles from txt file to not use default values but ones you were previously
using
```

```
void readVariablesFromFile( float& balance,
                            float& moneySpentIBM,
                            float&moneySpentTSCO,
                            float& moneySpentSHOP,
                            float& moneySpentGPV,
                            int& IBMAmount,
                            int& TSCOAmount,
                            int& SHOPAmount,
                            int& GPVAmount) {
    ifstream File;
    string label;
    File.open("variables.txt", ios::in);
    File >> balance >> moneySpentIBM >> moneySpentTSCO >> moneySpentSHOP
        >> moneySpentGPV >> IBMAmount >> TSCOAmount >> SHOPAmount >>
GPVAmount;
}
//Used for sell tab
float calculateProfit(  float& IBMPrice,
                        float& TSCOPrice,
                        float& SHOPPrice,
                        float& GPVPrice,
                        float& moneySpentIBM,
                        float& moneySpentTSCO,
                        float& moneySpentSHOP,
                        float& moneySpentGPV,
                        int& IBMAmount,
                        int& TSCOAmount,
                        int& SHOPAmount,
                        int& GPVAmount) {
```

```
    return (IBMPrice * IBMAmount + TSCOPrice * TSCOAmount + SHOPPrice *
SHOPAmount + GPVPrice * GPVAmount) - (moneySpentIBM + moneySpentTSCO +
moneySpentSHOP + moneySpentGPV);

}


int main() {

    //Stock URLs

    string IBMurl =
"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=IBM&out
putsize=full&apikey=demo";

    string TSCOurl =
"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY_ADJUSTED&sym
bol=TSCO.LON&outputsize=full&apikey=demo";

    string SHOPurl =
"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY_ADJUSTED&sym
bol=SHOP.TRT&outputsize=full&apikey=demo";

    string GPVurl =
"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY_ADJUSTED&sym
bol=GPV.TRV&outputsize=full&apikey=demo";

    //Response string generation

    string IBM = fetchData(IBMurl);

    string TSCO = fetchData(TSCOurl);

    string SHOP = fetchData(SHOPurl);

    string GPV = fetchData(GPVurl);

    const int max = 30; //Used for histogram and array that stores stock values

    float arr[max]; //Active stock prices from past (max) day

    stockArr stocks(max); //Stores Stock class objects

    //This is where cURL response strings are being parsed and turned into Stock class
objects

    bool data1 = parseData(IBM, stocks, max);

    bool data2 = parseData(TSCO, stocks, max);

    bool data3 = parseData(SHOP, stocks, max);
    bool data4 = parseData(GPV, stocks, max);
    //Does not continue if data parsed is bad
    if (!data1 || !data2 || !data3 || !data4) {
        return 0;
    }
    const char* items[] = { "IBM", "TSCO.LON", "SHOP.TRT", "GPV.TRV" }; //Stock
names
    //Current stock prices
    float IBMPrice = getCurrentPrice(stocks, items[0]);
    float TSCOPrice = getCurrentPrice(stocks, items[1]);
    float SHOPPrice = getCurrentPrice(stocks, items[2]);
    float GPVPrice = getCurrentPrice(stocks, items[3]);

    //UI variables
    float balance = 500;
    float moneySpentIBM = 0;
    float moneySpentTSCO = 0;
    float moneySpentSHOP = 0;
    float moneySpentGPV = 0;
    int IBMAmount = 0;
    int TSCOAmount = 0;
    int SHOPAmount = 0;
    int GPVAmount = 0;
    int sliderValue1 = 0;
    int sliderValue2 = 0;
    int sliderValue3 = 0;
    int sliderValue4 = 0;
    float max_val[] = { 180, 330, 130, 5 };
    float min_val[] = { 130, 250, 60, 2 };
    static int item_current_idx = 0;
    //Reads previous session variables that were stored in a txt file
    readVariablesFromFile(
        balance,
        moneySpentIBM,
        moneySpentTSCO,
        moneySpentSHOP,
        moneySpentGPV,
        IBMAmount,
        TSCOAmount,
        SHOPAmount,
        GPVAmount);
    int profit = calculateProfit(IBMPrice,
        TSCOPrice,
        SHOPPrice,
        GPVPrice,
        moneySpentIBM,
        moneySpentTSCO,
        moneySpentSHOP,
        moneySpentGPV,
        IBMAmount,
        TSCOAmount,
        SHOPAmount,
        GPVAmount);
    sf::RenderWindow window(sf::VideoMode(800, 800), "trading application",
sf::Style::Close); //Windows API window
    ImGui::SFML::Init(window);
    sf::Clock deltaClock;
    while (window.isOpen()) {
        //Close window event
        sf::Event e;
        while (window.pollEvent(e)) {
            ImGui::SFML::ProcessEvent(window, e);
            if (e.type == sf::Event::Closed) {
                writeVariablesToFile(   balance,
                                        moneySpentIBM,
                                        moneySpentTSCO,
                                        moneySpentSHOP,
                                        moneySpentGPV,
                                        IBMAmount,
                                        TSCOAmount,
                                        SHOPAmount,
                                        GPVAmount
                                    );
                window.close();
            }
        }
        ImGui::SFML::Update(window, deltaClock.restart()); //refreshes every frame


        //default parameters for the window
        ImGui::SetNextWindowPos(ImVec2(0, 0), ImGuiCond_FirstUseEver); //default
position


        //Here is where all the UI elements are
        ImGui::Begin("Trading Application", nullptr, ImGuiWindowFlags_NoTitleBar |
ImGuiWindowFlags_NoResize | ImGuiWindowFlags_NoMove);
        ImGui::BeginChild("Histogram", ImVec2(550, 0.0f), true);
        //Loads data for histogram when app is first launched
```

```
int x = 0;
for (int i = stocks.size() - 1; i >= 0; i--) {
    if (stocks[i].getName() == items[item_current_idx]) {
        arr[x] = stocks[i].getPrice();
        x++;
    }
}

//Dropdown menu above histogram
const char* combo_preview_value = items[item_current_idx];
if (ImGui::BeginCombo("Stocks", combo_preview_value))
{
    for (int n = 0; n < IM_ARRAYSIZE(items); n++)
    {
        const bool is_selected = (item_current_idx == n);
        if (ImGui::Selectable(items[n], is_selected)) {
            item_current_idx = n;
            int x = 0;
            //Selects histogram data based on selected dropdown option
            int i = stocks.size();
            for (int i = stocks.size() - 1; i >= 0; i--) {
                if (stocks[i].getName() == items[item_current_idx]) {
                    arr[x] = stocks[i].getPrice();
                    x++;
                }
            }
        }
```

```
            //Set the initial focus when opening the combo (scrolling + keyboard
navigation focus)
            if (is_selected) {
                ImGui::SetItemDefaultFocus();
            }
        }
        ImGui::EndCombo();
    }
    //Histogram
    ImGui::PlotHistogram("", arr, IM_ARRAYSIZE(arr), 0, NULL,
min_val[item_current_idx], max_val[item_current_idx], ImVec2(540, 750));
    ImGui::EndChild();
    ImGui::SameLine();
    ImGui::BeginChild("Menu", ImVec2(230, 0.0f), true);
    if (ImGui::BeginTabBar("MyTabBar", ImGuiTabBarFlags_None))
    {
        //BUY tab
        if (ImGui::BeginTabItem("Buy"))
        {
            renderBalance(balance);
            renderBuyStockOption("IBM", IBMPrice, sliderValue1, balance, IBMAmount,
moneySpentIBM);
            renderBuyStockOption("TSCO", TSCOPrice, sliderValue2, balance,
TSCOAmount, moneySpentTSCO);
            renderBuyStockOption("SHOP", SHOPPrice, sliderValue3, balance,
SHOPAmount, moneySpentSHOP);
            renderBuyStockOption("GPV", GPVPrice, sliderValue4, balance, GPVAmount,
moneySpentGPV);
            ImGui::EndTabItem();
        }
```

```
            //Your stocks tab
            if (ImGui::BeginTabItem("Your stocks"))
            {
                ImGui::SeparatorText("Current stocks bought");
                renderSellStockOption("IBM", IBMPrice, balance, IBMAmount,
moneySpentIBM);
                renderSellStockOption("TSCO", TSCOPrice, balance, TSCOAmount,
moneySpentTSCO);
                renderSellStockOption("SHOP", SHOPPrice, balance, SHOPAmount,
moneySpentSHOP);
                renderSellStockOption("GPV", GPVPrice, balance, GPVAmount,
moneySpentGPV);
                ImGui::Separator();
                if (profit < 0) {
                    //makes it red
                    ImGui::TextColored(ImVec4(1.0f, 0.0f, 0.0f, 1.0f), "Earnings: %d$", profit);
                }
                else if (profit == 0) {
                    //makes it grey
                    ImGui::TextColored(ImVec4(0.5f, 0.5f, 0.5f, 1.0f), "Earnings: %d", profit);
                }
                else if (profit > 0) {
                    //makes it green
                    ImGui::TextColored(ImVec4(0.0f, 1.0f, 0.0f, 1.0f), "Earnings: +%d$", profit);
                }
                ImGui::EndTabItem();
            }
            ImGui::EndTabBar();
        }
        ImGui::Separator();
        ImGui::EndChild();
        ImGui::End();

        window.clear(sf::Color(129, 133, 137)); //Clears old elements
        ImGui::SFML::Render(window); //This is where Imgui elements are rendered
        window.display(); //This is where SFML elements are rendered
    }
    ImGui::SFML::Shutdown(); //Dont forget to shutdown SFML
    return 0;
}
```