Assignment #3
CSCI 201 Spring 2021
5.0% of Course Grade

Title
SalStocks v2.0

Topics Covered
Networking
Multi-Threading
Concurrency Issues
API Querying

Introduction
After a stressful and sweaty pitch, the Council of SAL is satisfied with your trade-scheduling prototype demo application and has unanimously decided to give you the green light to fully implement your application. This involves pulling real data from Tiingo, issuing trades from SalStocks trading platform to traders in real time over the network, and chaining trades for traders, much like some other stock trade application whose name shall not be mentioned. A successful implementation will gain you 5 points and possibly a good word from the director.

Tiingo API
In addition to reading in data from two hard-coded csv files, you will use the Tiingo API to retrieve company stock costs. In other words, the company stock cost should be pulled from this API, and you will only have to read from the csv files for the schedule of trades and initial trader balances. You can learn more about the Tiingo API here: https://api.tiingo.com/documentation/general/overview . The data will be returned in a JSON format.

Please keep in mind that there are hourly, daily and monthly limits, so *do not wait to start your assignment until the last day*. You will also need to generate an API Key, so start the assignment early to allow enough time for testing.

Assignment
In this assignment, you will create two different programs - a server and a client. You will implement a networked delivery system where clients (which represent traders) receive trades from the server (which represents the trading platform), and the traders will complete the trades of the appropriate company stocks.

There will be some concurrency issues since traders will have to wait on other traders before starting the trade. There will be many ways to design out the program, so it would be wise to spend some time designing the program before you begin coding.

Similar to the previous assignment, here is a sample schedule.csv file:

```
0, AAPL, 2, 2021-02-19
0, AAPL, -1, 2021-02-19
0, AMD, 5, 2021-02-19
```

```
2, MSFT, -1, 2021-02-19
2, MSFT, -2, 2021-02-19
9, MSFT, -1, 2021-02-19
9, MSFT, 1, 2021-02-19
9, TSLA, 2, 2021-02-19
12, TSLA, 1, 2021-02-19
12, TSLA, -3, 2021-02-19
```

But there is an additional field specifying the date in the format YYYY-MM-DD. You will use this date in the Historical Price Information endpoint shown below to get the stock price details for that date like this:

```
https://api.tiingo.com/tiingo/daily/AMD/prices?startDate=2021-02-19&endDate=2021-02-19&token=6af808f3212a2076bf8aefb97835022531450323
```

Note: you will need to replace the token with your own.

You will get a response something like this:


```
[
    {
        "date":"2021-02-19T00:00:00.000Z",
        "close":89.58,
        "high":90.4163,
        "low":88.69,
        "open":89.75,
        "volume":29548273,
        "adjClose":89.58,
        "adjHigh":90.4163,
        "adjLow":88.69,
        "adjOpen":89.75,
        "adjVolume":29548273,
        "divCash":0.0,
        "splitFactor":1.0
    }
]
```


You will use the price corresponding to the “close” key above as the price for 1 stock of the specific company. Here the price for 1 stock of AMD is $89.58.

The documentation for this *Tiingo API* is listed in section *2.1.2, End-Of-day Endpoint* at:

https://api.tiingo.com/documentation/end-of-day

Aside from the schedule.csv file, you will need to read in a second file called traders.csv, in CSV format. Below is a layout of the second file, which contains information about the traders:

```
1, 1000
2, 2000
```

On each line of the CSV file, you will have the following fields:

- The first field indicates the serial number of the trader.
- The second field indicates the initial starting balance of the trader.

Server Functionality

When your server first runs, prompt the user for the name of the schedule file. Then it will prompt the user for the name of the traders file. The number of traders dispatched will be equal to the number of records in the traders.csv file. For example, if there are 2 records corresponding to 2 traders in the traders.csv file that means the server will not send out any orders until 2 clients (aka traders) have connected to the server. Afterwards, the server should begin listening on port 3456 for client connections. Every time a client connects to the server, verify that the connection was made by printing an output from the server, similar to the sample output below.

Client Functionality

The client will begin by welcoming the user to the program and prompt the user for the server hostname and port. If a valid connection is made, the program should let the user know how many more traders are needed before the trades are started. For example, in the case when there are 2 records in the traders.csv file, if there is currently only 1 connection, the client should print a message saying that 1 more trader is needed before the trades can be started.

Once the trades are initiated, the client will be responsible for completing the trade of the appropriate company stocks. The client should also be handling the returned data from the API calls to determine the stock prices of each company's stock.

Program Execution

Unlike the previous homework assignment, *traders can take up and be responsible for more than one trade at a time*. Additionally, traders can take up and be responsible for trades of multiple company stocks at a time.

A trader should take up as many trades as possible at the moment of initiation, provided that the sum obtained on totalling the money required for executing the purchase trades is less than or equal to the current balance of the trader. This means that if there are 5 trades that all have the same timestamp and all are purchase trades of value $100 each, a trader having a current remaining balance of $400 would be able to pick up and be responsible for only 4 of those 5 trades. As long as there are available traders, trades should be assigned promptly. If there are no available traders, the trade request will remain in the queue until a trader completes their trade. Once a trader completes, the trader should pick up as many of the queued trade requests (with respect to the current time) as are permissible under that trader’s balance limit.

Points to keep in mind:

1. The execution of a purchase trade should result in the reduction of the current balance of the trader who executed that trade, by an amount equal to the cost of that purchase trade.
2. **The execution of a selling trade will not result in any changes to the current balance of the trader.** You could consider the initial balance of each trader as a limit that they should not cross for total stock purchases.
3. If at a point in time there are multiple free traders, the assignment of trades should be done in the ascending order of the serial numbers of the traders.
4. The program should sleep for 1 second for each trade that is being executed.
5. The trades should be assigned in the order they are present in the schedule.csv file.
6. Consider the situation where the following trades are not executed because none of the traders have enough balance:
   a. `12, TSLA, 1, 2021-02-19`
   b. `12, MSFT, 10, 2021-02-19`

   At the end you need to print them as follows:
   `Incomplete trades: (12, TSLA, 1, 2021-02-19),(12, MSFT, 10, 2021-02-19)`
   If there are no incomplete trades, then print: `Incomplete trades:NONE`
7. At the end also print the total profit earned by each client(trader) which is the sum of all the sales that the trader did.

Please follow the sample execution below for the proper output format.

NOTE: It is not a requirement for the timestamps to start from all 0s, but you could do it as a learning exercise. But no points will be deducted for this. The time stamp format should be correct.

Sample Execution

Below is a sample execution of the program. The timestamp has been bolded to make it easier to read for this sample output.

Server	Client 1	Client 2
What is the path of the schedule file? **missing.csv** That file does not exist. What is the path of the schedule file? **schedule.csv** The schedule file has been properly read. What is the path of the traders file? **missing.csv** That file does not exist. What is the path of the traders file? **traders.csv** The traders file has been properly read. Listening on port 3456. Waiting for traders...		
Connection from	Welcome to SalStocks v2.0! Enter the server hostname: **localhost** Enter the server port: **3456**	

127.0.0.1 Waiting for 1 more trader(s)...	1 more trader is needed before the service can begin. Waiting...	
		Welcome to SalStocks v2.0! Enter the server hostname: **localhost** Enter the server port: **3456**
Connection from 127.0.0.1 Starting service.	All traders have arrived! Starting service.	All traders have arrived! Starting service.
	[00:00:000] Assigned purchase of 2 stocks of APPL. Total cost estimate= $129.87*2= $259.74. **[00:00:000]** Assigned sale of 1 stock of APPL. Total gain estimate= $129.87*1= $129.87. **[00:00:000]** Assigned purchase of 5 stocks of AMD. Total cost estimate= $89.58*5= $447.9. **[00:00:000]** Starting purchase of 2 stocks of APPL. Total cost= $129.87*2= $259.74. **[00:01.000]** Finished purchase of 2 stocks of APPL. Remaining balance= $1000-$259.74= $740.26. **[00:01.000]** Starting sale of 1 stock of APPL. Total gain= $129.87*1= $129.87. **[00:02.000]** Finished sale of 1 stock of APPL. Profit earned till now= $0+$129.87= $129.87. **[00:02.000]** Starting	**[00:02:000]** Assigned sale of 1 stock of MSFT. Total gain= $240.97*1= $240.97. **[00:02:000]** Assigned

	purchase of 5 stocks of AMD. Total cost= $89.58*5= $447.9. **[00:03.000]** Finished purchase of 5 stocks of AMD. Remaining balance= $740.26-$447.9= $292.36. **[00:09:000]** Assigned sale of 1 stock of MSFT. Total gain estimate= $240.97*1= $240.97. **[00:09:000]** Assigned purchase of 1 stock of MSFT. Total cost estimate= $240.97*1= $240.97. **[00:09:000]** Starting sale of 1 stock of MSFT. Total gain= $240.97*1= $240.97. **[00:10:000]** Finished sale of 1 stock of MSFT. Profit earned till now= $129.87+$240.97= $370.84. **[00:10:000]** Starting purchase of 1 stock of MSFT. Total cost= $240.97*1= $240.97. **[00:11:000]** Finished purchase of 1 stock of MSFT. Remaining balance= $292.36-$240.97= $51.39. **[00:12:000]** Assigned sale of 3 stocks of	sale of 2 stocks of MSFT. Total gain= $240.97*2= $481.94. **[00:02:000]**Starting sale of 1 stock of MSFT. Total gain= $240.97*1= $240.97. **[00:03:000]**Finished sale of 1 stock of MSFT. Profit earned till now= $0+$240.97= $240.97. **[00:03:000]**Starting sale of 2 stocks of MSFT. Total gain= $240.97*2= $481.94. **[00:04:000]**Finished signed sale of 2 stocks of MSFT. Profit earned till now= $240.97+$481.94= $722.91. **[00:09:000]** Assigned purchase of 2 stocks of TSLA. Total cost estimate= $781.3*2= $1562.6. **[00:09:000]** Starting purchase of 2 stocks of TSLA. Total cost= $781.3*2= $1562.6. **[00:10:000]** Finished purchase of 2 stocks of TSLA. Remaining balance= $2000-$1562.6= $437.4.

	TSLA. Total gain estimate= $781.3*3= $2343.9. Profit earned till now= $370.84+$2343.9= $2714.74. **[00:12:000]** Starting sale of 1 stock of TSLA. Total gain= $781.3*3= $2343.9. **[00:13:000]** Finished sale of 1 stock of TSLA.	
Incomplete trades: (12, TSLA, 1, 2021-02-19)	**[00:13:010]**Incomplete trades: (12, TSLA, 1, 2021-02-19) Total Profit Earned: $2714.74.	**[00:13:010]**Incomplete trades: (12, TSLA, 1, 2021-02-19) Total Profit Earned: $722.91.
Processing complete!	**[00:13:010]** Processing complete!!	**[00:13:010]** Processing complete!!

Grading Criteria

The way you go about implementing the solution is not specifically graded, but the output must match exactly what you see in the execution above.

Networking (1.0%)

0.2% The first client can connect to the server
0.3% Only the number of clients, which is determined by the number of records in traders.csv, can connect to the server.
0.5% Server output is correct

Data I/O (1.0%)

0.3% The schedule file and traders file are read appropriately
0.2% Data is parsed from the Tiingo API
0.5% Client output is correct

Program Execution (3.0%)

1.0% The order of deliveries is correct
1.0% The timing of deliveries is correct
1.0% Orders are delivered as expected with no exceptions, crashing, deadlock, starvation, or freezing.