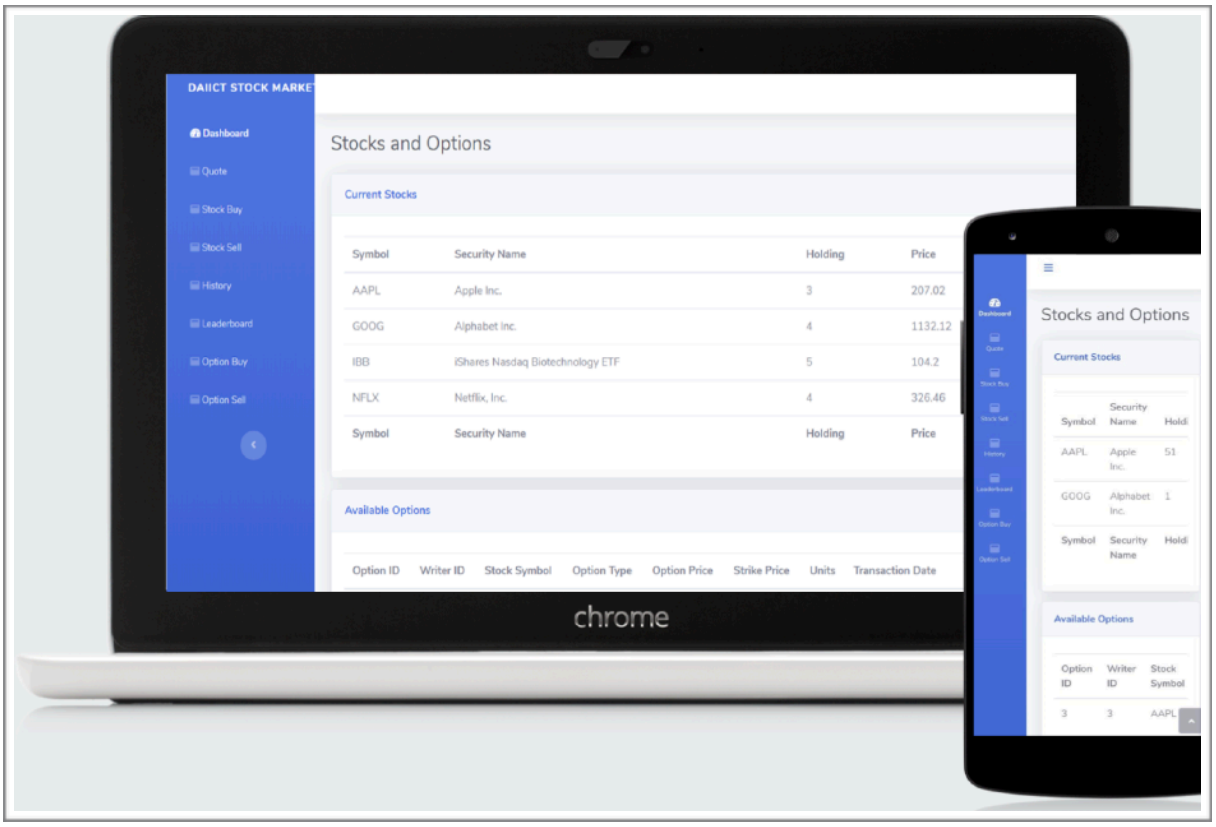
Mini BTP Report

Educational Trading Terminal



Stocks and Options

Current Stocks

Symbol	Security Name	Holding	Price
AAPL	Apple Inc.	3	207.02
GOOG	Alphabet Inc.	4	1132.12
IBB	iShares Nasdaq Biotechnology ETF	5	104.2
NFLX	Netflix, Inc.	4	326.46
Symbol	Security Name	Holding	Price

Available Options

Option ID	Writer ID	Stock Symbol	Option Type	Option Price	Strike Price	Units	Transaction Date

Vidhin Parmar

201601003

November 2019

Abstract

Design and develop a tool that can be used by students to attain a hands-on experience on real life trading and apply/understand concepts of Computational Finance. The Trading terminal is a Web-based application compatible for Mobile and Computer users wherein each new account is given an initial amount of 10,000 dollars of virtual money to buy/sell Stocks and Options and compete and compare progress amongst colleagues.

Introduction

The project got its motivation from the mock stock and options trading session done in the class of Computational Finance which were conducted offline on pen and paper and needed a lot of manual computation. Our work consisted of the following:

- Creating a web application and hosting it on the internet.
- Creating and managing a database of all users for several features.
- Adding desired features by building logic and coding it in Python.
- Creating a common pool for buying and selling Stocks.
- Fetching live data for Stocks from official Stock exchanges.
- Implementing an order book strategy for transaction of stocks among colleagues.
- Creating a common pool for buying, selling and writing Options.
- Extending the order book strategy to work on Options.
- Implementation of leader-board, history of transaction for all users.
- Implementing a Compound interest model that makes the money grow as it sits in a bank.
- Handling login, logout and cash updates for each user to maintain leaderboard.
- Making a user friendly and appealing Front-end for the application.
  My personal work includes developing the frontend, building the leaderboard, the transactional history pages and building a system for buying and selling of options. This was done by using an order book to replicate the way option trading occurs in stock exchanges.

Background

Python and flask form the application's core. HTML along with Bootstrap 4 was used to develop the application frontend. Heroku to host the application on the web. Github has been used for collaboration and version control. PostgreSQL for database management.

USP for the app is ease of understanding and dynamics for trading. Getting rid of complex UI and overwhelming information that a beginner has no use of and makes the topic more confusing and hard to navigate. We have aimed to keep it simple yet provide the closest experience of Computational finance to students without the money risk.

Project

I shall begin by explaining the features implemented under my part along with the various use cases and importance of those features.

Flask is a versatile frameworks which allows to build your webpage using python. You can provide URL's and HTTP methods using flask methods and templating was done using jinja2. We used the library of psycopg2 as our PostgreSQL adapter so that we can use SQL commands and fetch data from within our application.

Options Quote

The Quote feature lets the user perform several tasks. First it asks the user which stock option does he/she want the quote for. When the user enters the the stock symbol a comprehensive info page opens up which display the stocks current market price, the companies' full name, Intraday TimeSeries chart for the stock price since last 5 days, option order book for that stock, the current holdings the user has for that stock option, and the option to add a buy query or a sell query for that stock optoin in the order book either as a market order or as a limit order.

A Market order is a transaction that happens at the current market price and does not have to wait in the order book. A limit order gives the user the chance to determine the desired price at which he/she wants to buy/sell. That request waits in the order book for someone to add an opposite query at a similar price.

So there are two options on the information page. For buying that stock option, the user needs to enter the price at which he/she agrees to buy that stock option, the expiry date, the strike price and the quantity that he want to buy. This requests gets added to the option order book instantaneously.

Also for selling the stock option, the user needs to enter the price at which he/she agrees to sell that stock option and the quantity that he wishes to sell. This requests gets added to the order book instantaneously. Here is the code snippet for the buy stocks option function:

```
@app.route("/option_buy/", methods=["POST"])
@login_required
def option_buy():
    refresh()
    stock_symbol = request.args.get('stock_symbol', None)
    c.execute("SELECT cash FROM users WHERE id = %s",
[session["user_id"]])
    current_cash = float(c.fetchall()[0][0])
    option_price = float(request.form.get("option_price"))
    shares = int(request.form.get("shares"))
    option_type = request.form.get("option_type")
    strike_price = float(request.form.get("strike_price"))
    expiry_date = request.form.get("expiry_date_buy")
    transaction_cost = option_price * shares
    if transaction_cost <= current_cash:
        c.execute("INSERT INTO option_chain(writer_id, holder_id,
stock_symbol, option_type, option_price, strike_price, shares,
transaction_date,expiry_date, buy_sell) VALUES(%s, %s, %s, %s, %s, %s,
%s, %s, %s, 'buy')",
            [session["user_id"], session["user_id"], stock_symbol,
option_type, option_price, strike_price,
             shares,
             time.strftime("%c"), expiry_date])
        db.commit()
        print("Transaction sent.")
        c.execute("SELECT last_value FROM option_chain_option_id_seq;")
        order_id = c.fetchall()[0][0]
        option_chain_execute(order_id)
    else:
        return apology("ERROR", "INSUFFICIENT FUNDS")
    return redirect(url_for("option_quote",stock_symbol=stock_symbol))
```

Similarly, here is the code for the sell stock option function,

```
@app.route("/option_sell/", methods=["POST"])
@login_required
def option_sell():
    refresh()
    stock_symbol = request.args.get('stock_symbol', None)
    c.execute("SELECT username FROM users WHERE id = %s",
[session["user_id"]])
    c.execute("SELECT * FROM option_transaction WHERE holder_id=%s AND
is_available='Yes'", [session["user_id"]])
    transactions = c.fetchall()

    option_id = request.form.get("option_id")
    option_price = request.form.get("option_price")
    if not option_id:
        strike_price = request.form.get("strike_price")
        option_type = request.form.get("option_type")
        shares = request.form.get("shares")
        expiry_date = request.form.get("expiry_date_sell")
        c.execute("INSERT INTO option_chain(writer_id, holder_id,
stock_symbol, option_type, option_price, strike_price, shares,
transaction_date,expiry_date,buy_sell) VALUES(%s, %s, %s, %s, %s, %s,
%s, %s, %s,'sell')", [session["user_id"], session["user_id"],
stock_symbol, option_type, option_price, strike_price, shares,
time.strftime("%c"), expiry_date])
        db.commit()
        c.execute("SELECT last_value FROM option_chain_option_id_seq;")
        order_id = c.fetchall()[0][0]
        option_chain_execute(order_id)
    elif int(option_id) in [row[0] for row in transactions]:
        writer_id, holder_id, stock_symbol, option_type, strike_price, 
shares, expiry_date = c.execute("SELECT writer_id, holder_id, 
stock_symbol, option_type, strike_price, shares, expiry_date FROM 
option_transaction WHERE option_id = %s", [option_id]).fetchall()[0]
        c.execute("UPDATE option_transaction SET is_available = 'No' 
WHERE holder_id = %s and option_id = %s",[holder_id, option_id])
        c.execute("UPDATE option_chain SET option_price= %s, shares = 
%s where option_id=%s", [option_price, shares, option_id])
        db.commit()
        print("Transaction sent.")
        option_chain_execute(option_id)
    else:
        return apology("ERROR", "No such option")
    return redirect(url_for("option_quote", stock_symbol=stock_symbol))
```

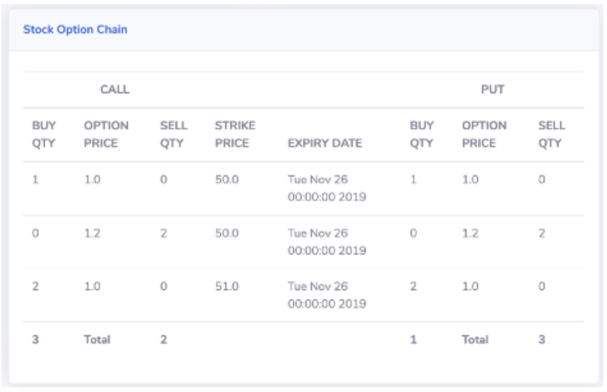
This code depicts have each error case has been handled and if all the requirements of the inputs passes then the query is pushed to the option order book using the command option_chain_execute(order_id).

Option Order Book Implementation and strategy

This is the most challenging part of the project as it required me to first understand the concept of order books fully and after certain discussions with Professor Mulherkar about how useful this will be for the understanding of stock markets and several hours spent on websites of NYSE, NSE etc that I gained a proper understanding of this concept. It is a sophisticated algorithm that stock markets use to assign and execute transaction within splits of seconds of such high volume with extreme precision. Not many applications on the market let you trade amongst your selected own environment so they do not need to develop their own order book and can use the market provided book. We created this application for students of DAIICT so we needed to implement our own order book.

What is an order book? An order book is an electronic list of buy and sell orders for a specific security or financial instrument organized by price level. An order book lists the number of shares being bid or offered at each price point, or market depth.

As shown in the image, the option price, strike price and expiry date is ordered. on either sides of the option price column is the Bid/Ask or Buy/sell columns which tell us the quantity of options that are available to be bought or sold at that particular price and expiry date. So when a user enter a request that he wants to buy a 15 shares at a price of say 44.5 dollars we need to first find the index in the ordered arrangement of prices where the entry shall go and if there is a sell query for the same price or lower price already available on the table, if yes then the max number of shares get traded and if some remain they are still left in the order book. The reverse is the case for the sell order for an option. The code is



Stock Option Chain

CALL					PUT		
BUY QTY	OPTION PRICE	SELL QTY	STRIKE PRICE	EXPIRY DATE	BUY QTY	OPTION PRICE	SELL QTY
1	1.0	0	50.0	Tue Nov 26 00:00:00 2019	1	1.0	0
0	1.2	2	50.0	Tue Nov 26 00:00:00 2019	0	1.2	2
2	1.0	0	51.0	Tue Nov 26 00:00:00 2019	2	1.0	0
3	Total	2			1	Total	3

explained step by step below

```
        c.execute(
            "SELECT option_price, sum(shares), buy_sell, option_type, 
expiry_date, strike_price FROM option_chain where stock_symbol=%s and 
shares>0 group by option_price, buy_sell, option_type, expiry_date, 
strike_price ORDER BY expiry_date, strike_price, option_price", 
[stock['symbol']])
        orders = c.fetchall()
```

Fetches all option orders from the options table

```
        exp_str = sorted(list(set([(item[4], item[5]) for item in 
orders])))
```

Forms a set of tuples of expiry date and strike price from the given options.

```
        option_order_book = []
        total_call_buy = 0
        total_call_sell = 0
        total_put_buy = 0
        total_put_sell = 0
        for row in orders:
            if row[3] == 'CALL':
                if row[2] == 'buy':
                    total_call_buy += row[1]
                else:
                    total_call_sell += row[1]
            if row[3] == 'PUT':
                if row[2] == 'buy':
                    total_put_buy += row[1]
                else:
                    total_put_sell += row[1]
```

Gets the value of total buy and sell quantities for PUT and CALL options in the order book.

```
        for item in exp_str:
            call_list = []
            put_list = []
            for row in orders:
                if (row[4], row[5]) == item and row[3] == 'CALL':
                    call_list.append(list(row))
                elif (row[4], row[5]) == item and row[3] == 'PUT':
                    put_list.append(list(row))
            call_list_order_price = set([item[0] for item in 
call_list])
            put_list_order_price = set([item[0] for item in put_list])
```

Forms a set of tuples of order prices for call and put orders in the option order book.

```
            new_call_list = []
            new_put_list = []
            for x in call_list_order_price:
                buy_qty = 0
                sell_qty = 0
                for row in call_list:
                    if row[0] == x:
                        if row[2] == 'buy':
                            buy_qty += row[1]
                        else:
                            sell_qty += row[1]
                new_call_list.append([buy_qty, x, sell_qty])
```

Forms the call side of the row in option book

```
            for x in put_list_order_price:
                buy_qty = 0
                sell_qty = 0
                for row in call_list:
                    if row[0] == x:
                        if row[2] == 'buy':
                            buy_qty += row[1]
                        else:
                            sell_qty += row[1]
                new_put_list.append([buy_qty, x, sell_qty])
```

Forms the call side of the row in option book

```
            call_put = list(itertools.zip_longest(new_call_list, 
new_put_list))
```

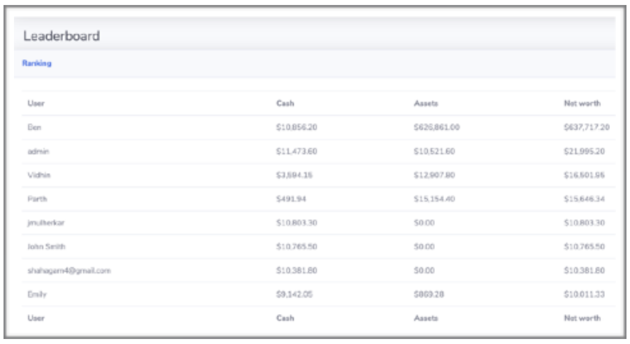
These two sides are then zipped together and the resulting row is added into the option order book

```
            for row in call_put:
                if row[0] is None:
                    option_order_book.append(['-', '-', '-', item[1], 
item[0], row[1][0], row[1][1], row[1][2]])
                elif row[1] is None:
                    option_order_book.append([row[0][0], row[0][1], 
row[0][2], item[1], item[0], '-', '-', '-'])
                else:
                    option_order_book.append([row[0][0], row[0][1], 
row[0][2], item[1], item[0], row[1][0], row[1][1], row[1][2]])
        return render_template("option_quoted.html", stock=stock, 
url='data:image/png;base64,{}'.format(graph_url), username=username, 
option_order_book=option_order_book, total_put_buy=total_put_buy, 
total_put_sell=total_put_sell, total_call_buy=total_call_buy, 
total_call_sell=total_call_sell, available=available, 
stock_symbol=stock["symbol"], option_dates=option_dates)
```

And now, our option order book is ready.

Leaderboard

Users are sorted on basis of their net worth and displayed on this page. Net worth = Cash + value of assets. This update occurs dynamically.



Leaderboard

Ranking

User	Cash	Assets	Net worth
Ben	$10,856.20	$626,861.00	$637,717.20
admin	$11,473.60	$10,521.60	$21,995.20
Vidhin	$3,694.16	$12,907.80	$16,601.96
Parth	$491.94	$15,154.40	$15,646.34
jmulherkar	$10,803.30	$0.00	$10,803.30
John Smith	$10,765.50	$0.00	$10,765.50
shahagam4@gmail.com	$10,381.80	$0.00	$10,381.80
Emily	$9,142.05	$869.28	$10,011.33
User	Cash	Assets	Net worth

History Of Transactions

Each transaction is stored either in the transactions table or in the option transactions table. These tables are used to display the history of transactions for the logged in user.

Summary

The website is up and ready, hosted on http://tradingterminal.herokuapp.com. The corner cases in terms of buying and selling of Stocks and options are covered and the database updation also occurs without errors or loss of Data. In cases of erroneous inputs it display what the error is so the user can make ammends.

References

- Investopedia options basic tutorial
- Jinja2 Documentation
- NSE India
- Heroku postgreSQL Documentation
- Github Yfinance Documentation
- Chicago Board Options Exchange Strategy Tool for trading