Stock Brokerage Account

CMPE226 Team4
Rohan Deshmukh(015214329)
Omar Laymoun(007568951)

Database Project Topic

- App for stock market transaction system that allows users to buy and sell stocks.
- Users are allowed to post a buy or sell information in the database.
- When a buy matches a sell price, the name of stocks, the system will create trade.
- The system also allows users to cancel a buy or sell, search for stocks.
- Which also helps users with Financial Planning and Advice, Retirement Plans, Wealth Management Services, Trading and Brokerage services.

Database technologies and tools

Database engine	Mysql (mysql database hosted on digital ocean)
DB application technologies	mysql.connector
Frameworks	PySimpleGUI
Languages	Python, Mysql
DB access technology	mysql.connector
Libraries	Requests, mysql.connector, PySimpleGUI

List of functionalities of each role

Admin:
1. Admin can login through the admin sign in page.
2. Admin can view daily trades that are completed, or pending.
3. Admin gives access to users based on their user types. **(not implemented seemed irrelevant to application)**
4. Admin put stocks for sale for different companies.
5. Admin approves the cash transfer from external account to customer account in Miniworld and vice versa. **(not implemented)**

Customer:
1. Customers can login through the customer sign in page.
2. Customers can add funds to their account balance through transfer.
3. Customers can search for stock portfolios of different companies.
4. Customers can always view their current stock portfolios.
5. Customers can place orders which do not expire, or sell the shares for the market price and cancel orders for particular stocks using funds from their account balance.

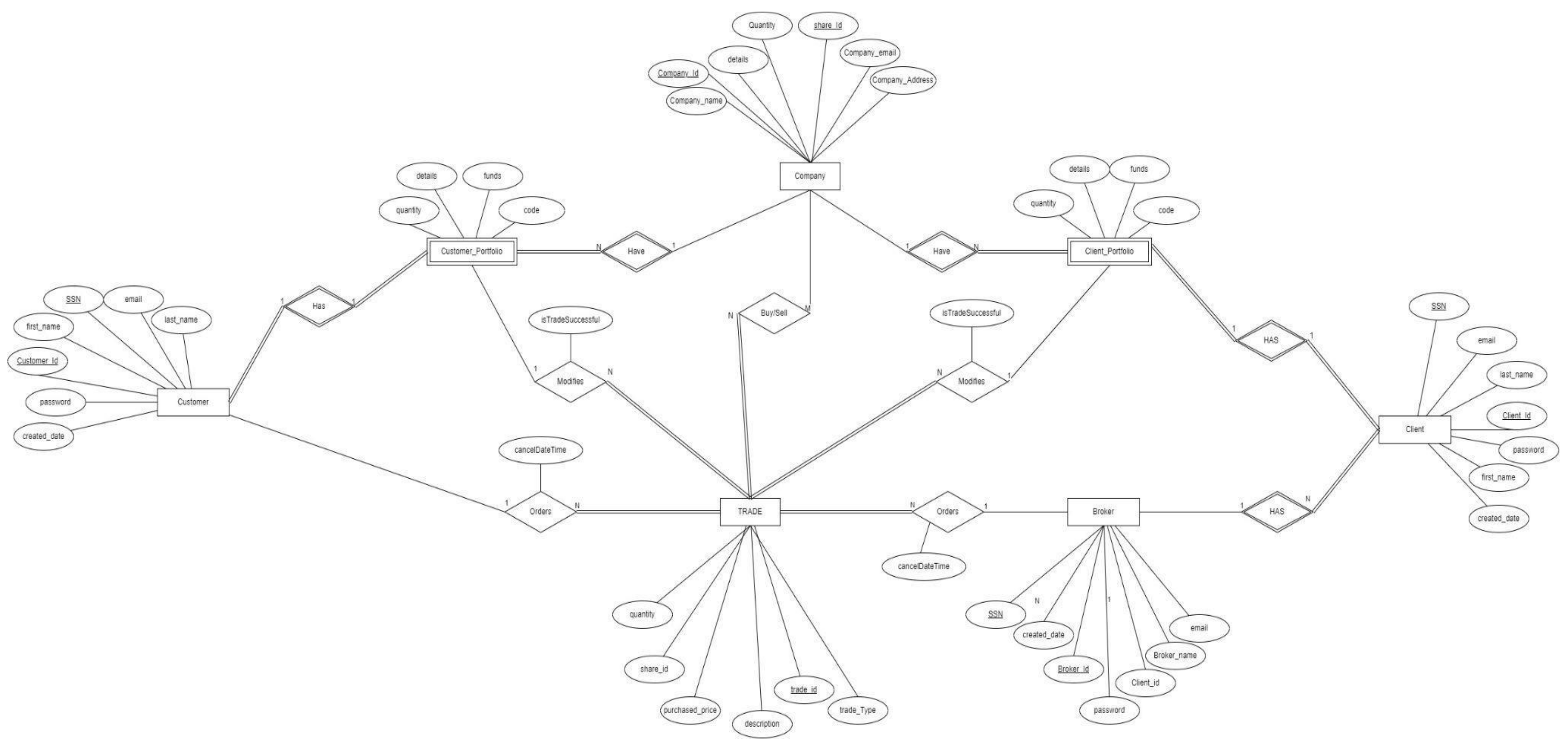
Broker:
1. Brokers can login through the broker sign in page.
2. Brokers can search for stock portfolios of different companies.
3. Brokers can always view their clients current stock portfolios.
4. Brokers help place orders which do not expire or sell the shares for the market price and cancel orders for clients.

Member’s contributions

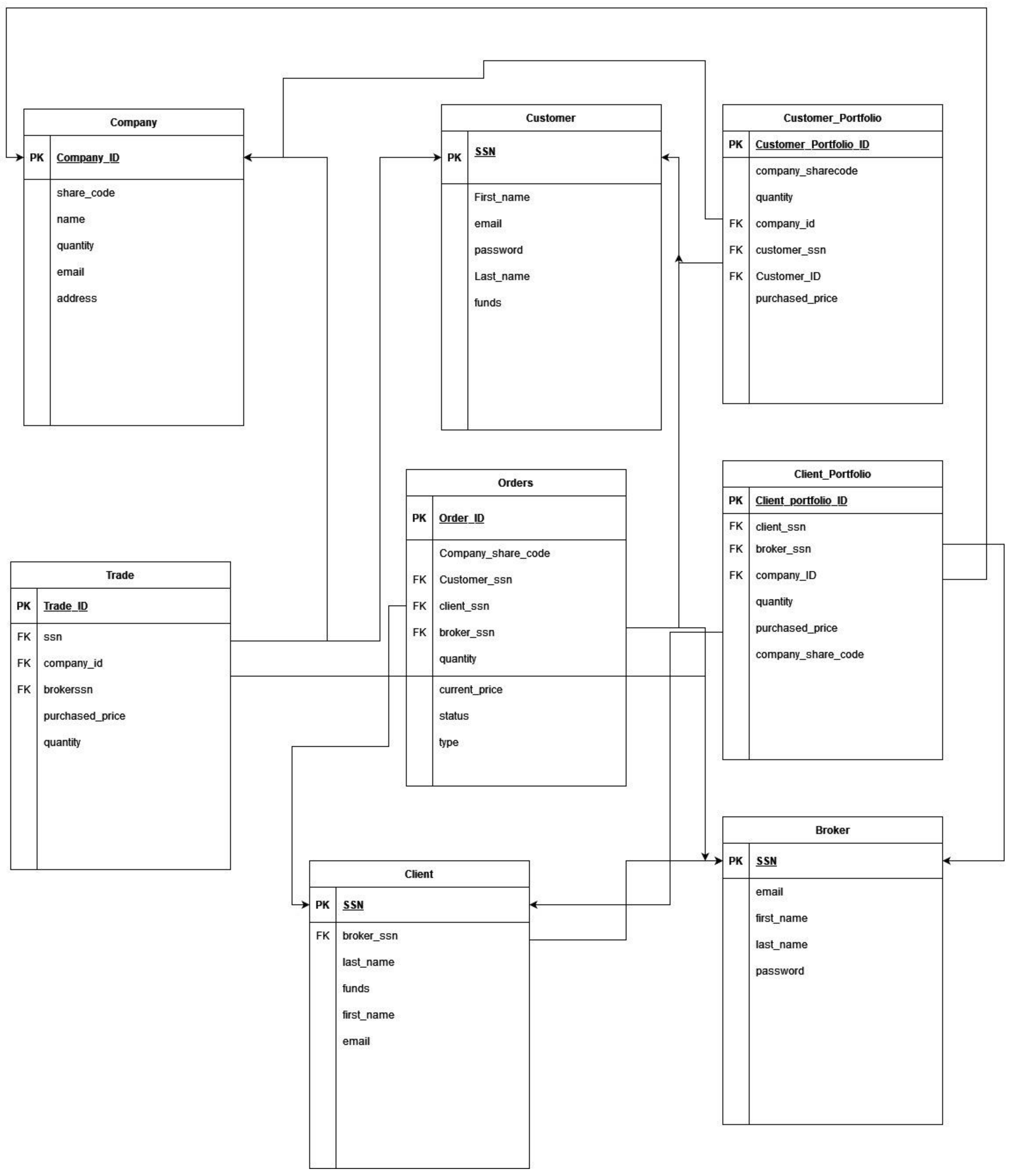
1. Proposal: Both worked on proposal.
2. ERD/Schema: ERD by Rohan and Schema Diagram by Omar.
3. Report, slides: Report was done by Rohan and Omar. Slides were done by Omar.
4. Implementation (break down to components), testing (break down to components)
    - Database design and implementation - Rohan
    - Omar took the leadership role as he had more experience working with GUIS and system design
    - Intial decision of tools and technologies - Omar
    - Creation of customer account by Omar
    - Login for customer and password hashing by Omar
    - Git repository maintained by Omar
    - Architecture and high level design by Omar
    - Creation of broker account by Omar
    - Login for broker and password hashing by Omar
    - Creation of admin account by Omar
    - Login for admin and password hashing by Omar
    - Detailed implementation of Buy stock and Sell Stock and view portfolio implemented by Rohan
    - Component for customer from sign in to every other features implemented by Rohan
    - Component for broker from sign in to every other features implemented by Rohan
    - Component for admin from sign in to every other features implemented by Rohan
    - Generation of log files done by Rohan
    - Stored Procedures and View by Rohan
    - Implementation Buy and Sell logic by Rohan
    - Marketstack.com API integration by Rohan
    - Database hosted on digital ocean mysql server by Rohan
    - Searching for current price of the share price by Rohan

Final design of database portion

1. **ER Diagram**:
   Link : https://drive.google.com/file/d/1XN-Zr3v5osivKNTRa7Qs0WM6eb5lKtoF/view?usp=sharing



2. **Schema diagram**:



3. **Specification of each DB object (table, column, view, stored procedure, etc.) and its meaning/purpose, in tabular or list format:**

Object	Specification
Broker	Broker table contains information about broker which has clients.
Client	Client do not have access to system but trades placed of their behalf by Broker.
Client Portfolio	Client Portfolio can be viewed by Broker every client that he has.
Company	Company gets added by Admin whose shares are available for clients and customers.

Customer	Customer can view their customer portfolio and can place trades.
placedorders	Placed orders are trades that are not yet completed.
portfoliocustomers	Portfolio customers can be viewed by Broker every client that he has.
trade	Every share transaction that are getting completed comes under trade.

4. **User login password: hashed or encrypted? How is this done? In which column of which table?**
   Hashing Algorithm : MD5, defined in RFC 1321, is a hash algorithm to turn inputs into a fixed 128-bit (16 bytes) length of the hash value.
   Encrypted using : Python Library - hashlib

Important SQL queries used in code:

Login functionality:
SELECT email,password,ssn from customer where email = %s and password = %s

Sample: 'deshmukhcr7@gmail.com' , '321654987', '654654654'

Cancel outstanding order:
SELECT * FROM placedorders where customerssn = %s and status = %s

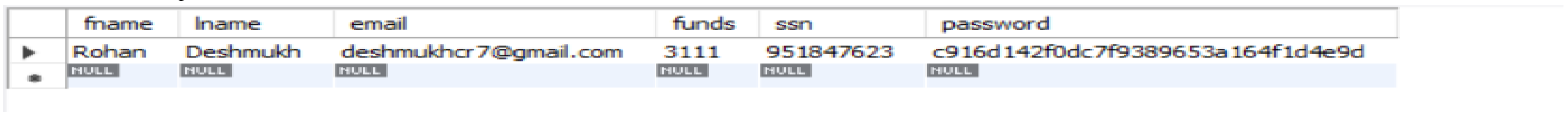
UPDATE Project2.placedorders SET status = 'cancelled' WHERE id = "+values['-OrderId-']+"

Display portfolio:
SELECT ssn,funds FROM client where Id = %s

SELECT companysharecode,sum(quantity) FROM clientportfolio where clientssn = %s GROUP BY companysharecode

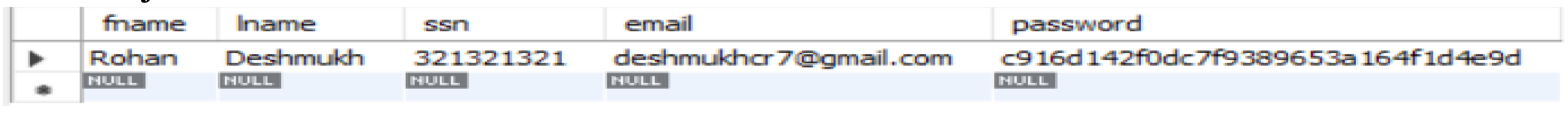
Create account functionality:
INSERT INTO customer (fname, lname, email, funds, ssn, password) VALUES(%s, %s, %s, %s, %s, %s)

Customer Object :



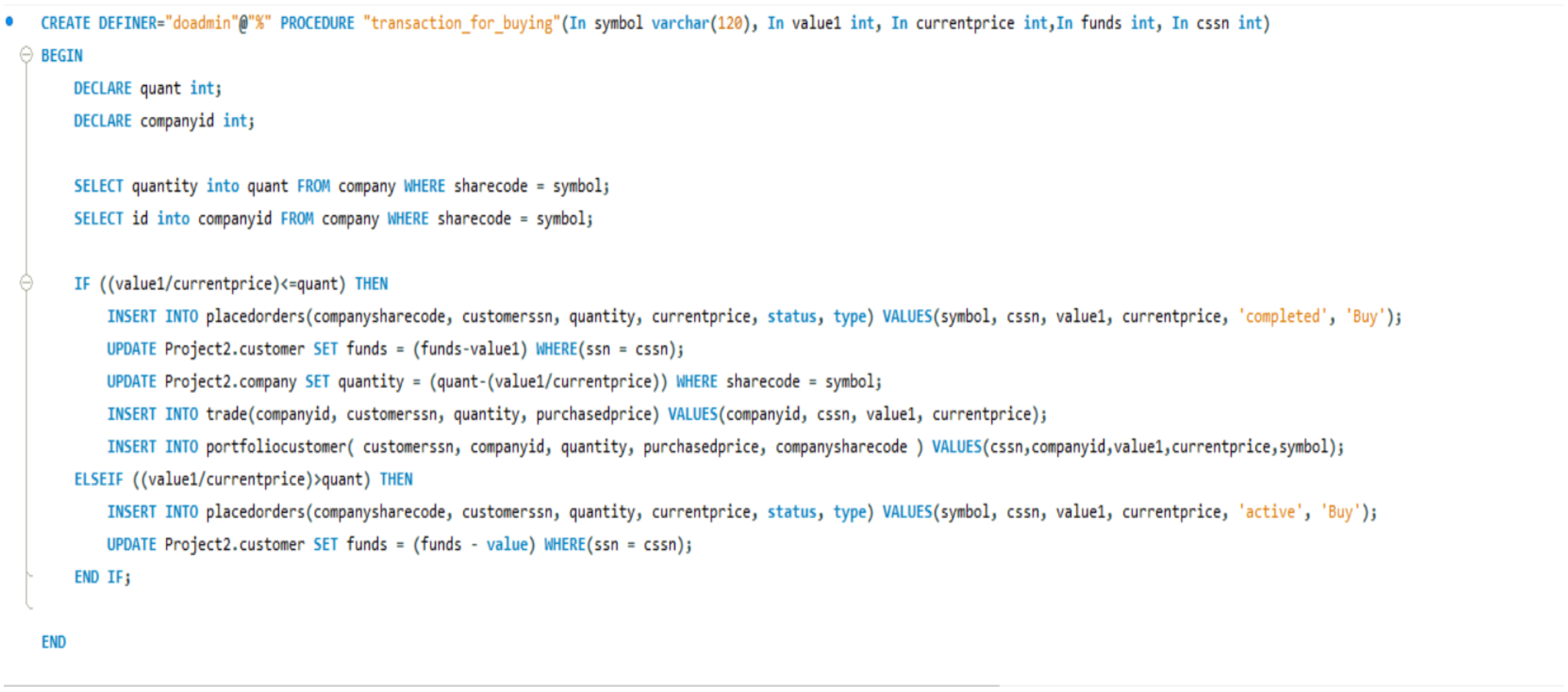
	fname	lname	email	funds	ssn	password
▶	Rohan	Deshmukh	deshmukhcr7@gmail.com	3111	951847623	c916d142f0dc7f9389653a164f1d4e9d
*	NULL	NULL	NULL	NULL	NULL	NULL

Broker Object:

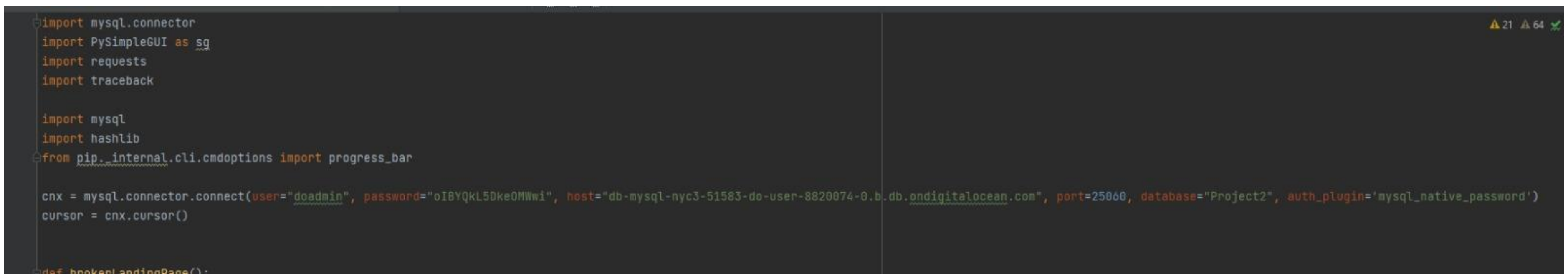


	fname	lname	ssn	email	password
▶	Rohan	Deshmukh	321321321	deshmukhcr7@gmail.com	c916d142f0dc7f9389653a164f1d4e9d
*	NULL	NULL	NULL	NULL	NULL

5. Any explicit multi-SQL-statement DB transactions that modify data and are initiated from DB server side (i.e., initiated from stored procedure)
   Stored Procedure : Transaction for buying
   a. **Place Order**
   b. **Reduce funds from customer**
   c. **Reduce no of shares from company's available shares from our mini world.**
   d. **Create record in trade.**
   e. **Create record in customer portfolio.**



```sql
CREATE DEFINER="doadmin"@"%" PROCEDURE "transaction_for_buying"(In symbol varchar(120), In value1 int, In currentprice int,In funds int, In cssn int)
BEGIN
    DECLARE quant int;
    DECLARE companyid int;

    SELECT quantity into quant FROM company WHERE sharecode = symbol;
    SELECT id into companyid FROM company WHERE sharecode = symbol;

    IF ((value1/currentprice)<=quant) THEN
        INSERT INTO placedorders(companysharecode, customerssn, quantity, currentprice, status, type) VALUES(symbol, cssn, value1, currentprice, 'completed', 'Buy');
        UPDATE Project2.customer SET funds = (funds-value1) WHERE(ssn = cssn);
        UPDATE Project2.company SET quantity = (quant-(value1/currentprice)) WHERE sharecode = symbol;
        INSERT INTO trade(companyid, customerssn, quantity, purchasedprice) VALUES(companyid, cssn, value1, currentprice);
        INSERT INTO portfoliocustomer( customerssn, companyid, quantity, purchasedprice, companysharecode ) VALUES(cssn,companyid,value1,currentprice,symbol);
    ELSEIF ((value1/currentprice)>quant) THEN
        INSERT INTO placedorders(companysharecode, customerssn, quantity, currentprice, status, type) VALUES(symbol, cssn, value1, currentprice, 'active', 'Buy');
        UPDATE Project2.customer SET funds = (funds - value) WHERE(ssn = cssn);
    END IF;

END
```

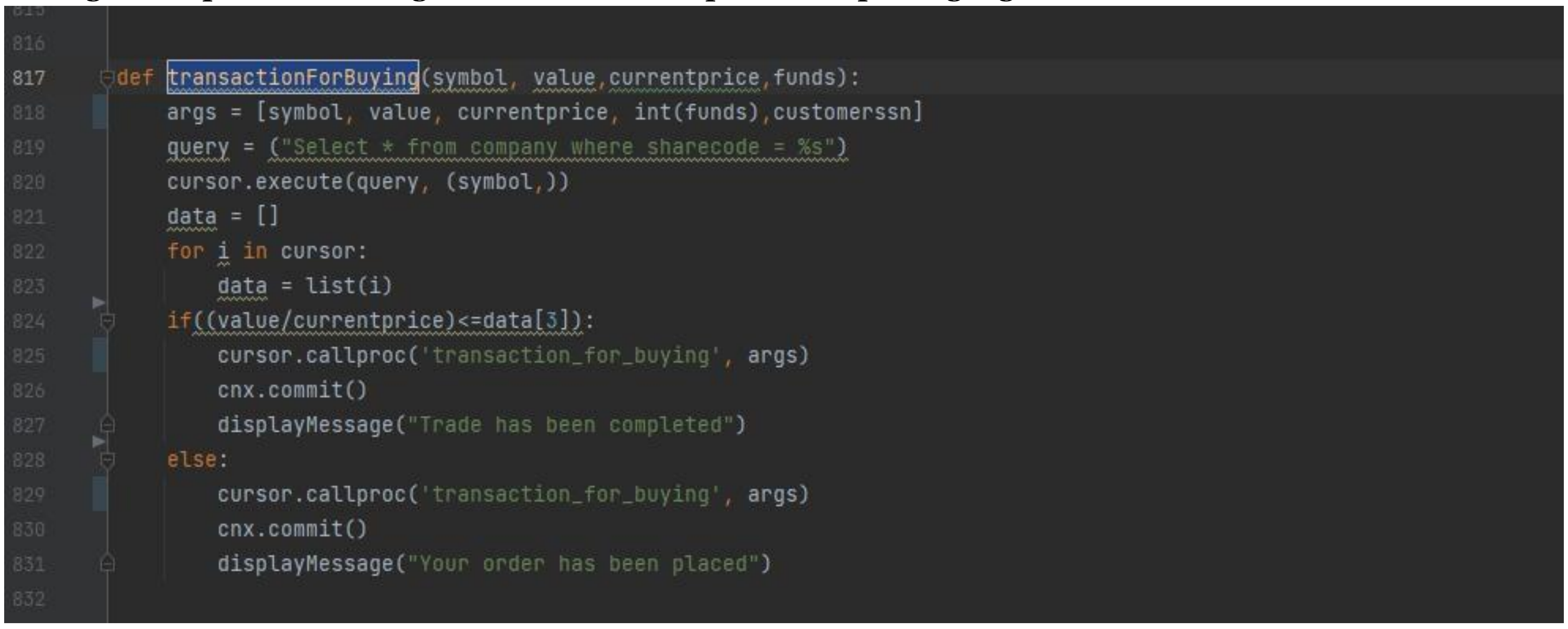


```python
import mysql.connector
import PySimpleGUI as sg
import requests
import traceback

import mysql
import hashlib
from pip._internal.cli.cmdoptions import progress_bar

cnx = mysql.connector.connect(user="doadmin", password="oIBYQkL5DkeOMWwi", host="db-mysql-nyc3-51583-do-user-8820074-0.b.db.ondigitalocean.com", port=25060, database="Project2", auth_plugin='mysql_native_password')
cursor = cnx.cursor()
```

6. Stored procedures and views: describe functionality for each and show code snippet
   a. Screenshots – show code snippet that invokes the stored procedure(s) you defined
      i. Transaction_for_buying

Connection By mysql.connector

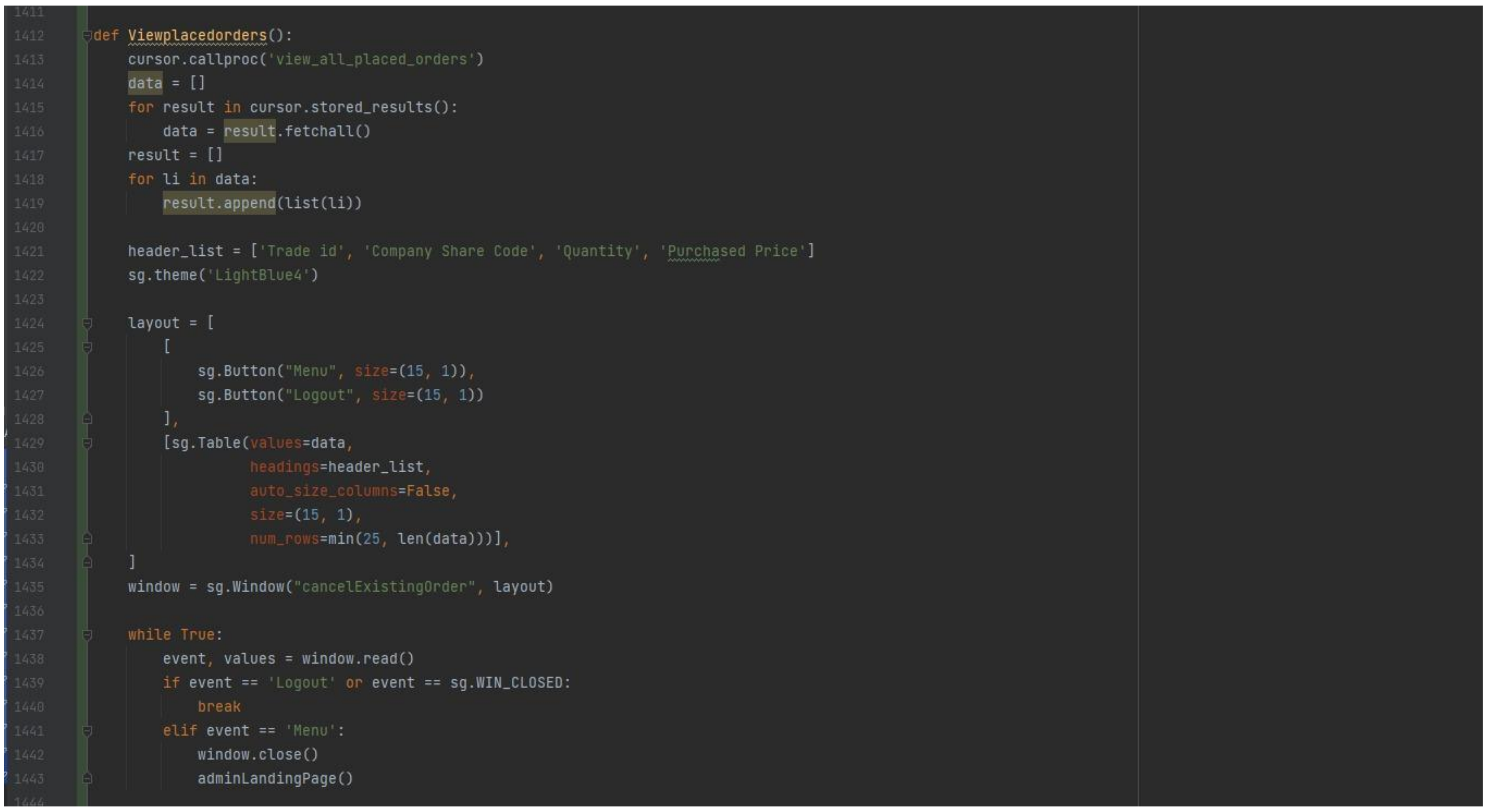
Executing using connection variable cursor.
Calling stored procedure using function cursor.callproc() and passing arguments to the cursor and commit.



```python
def transactionForBuying(symbol, value,currentprice,funds):
    args = [symbol, value, currentprice, int(funds),customerssn]
    query = ("Select * from company where sharecode = %s")
    cursor.execute(query, (symbol,))
    data = []
    for i in cursor:
        data = list(i)
    if((value/currentprice)<=data[3]):
        cursor.callproc('transaction_for_buying', args)
        cnx.commit()
        displayMessage("Trade has been completed")
    else:
        cursor.callproc('transaction_for_buying', args)
        cnx.commit()
        displayMessage("Your order has been placed")
```

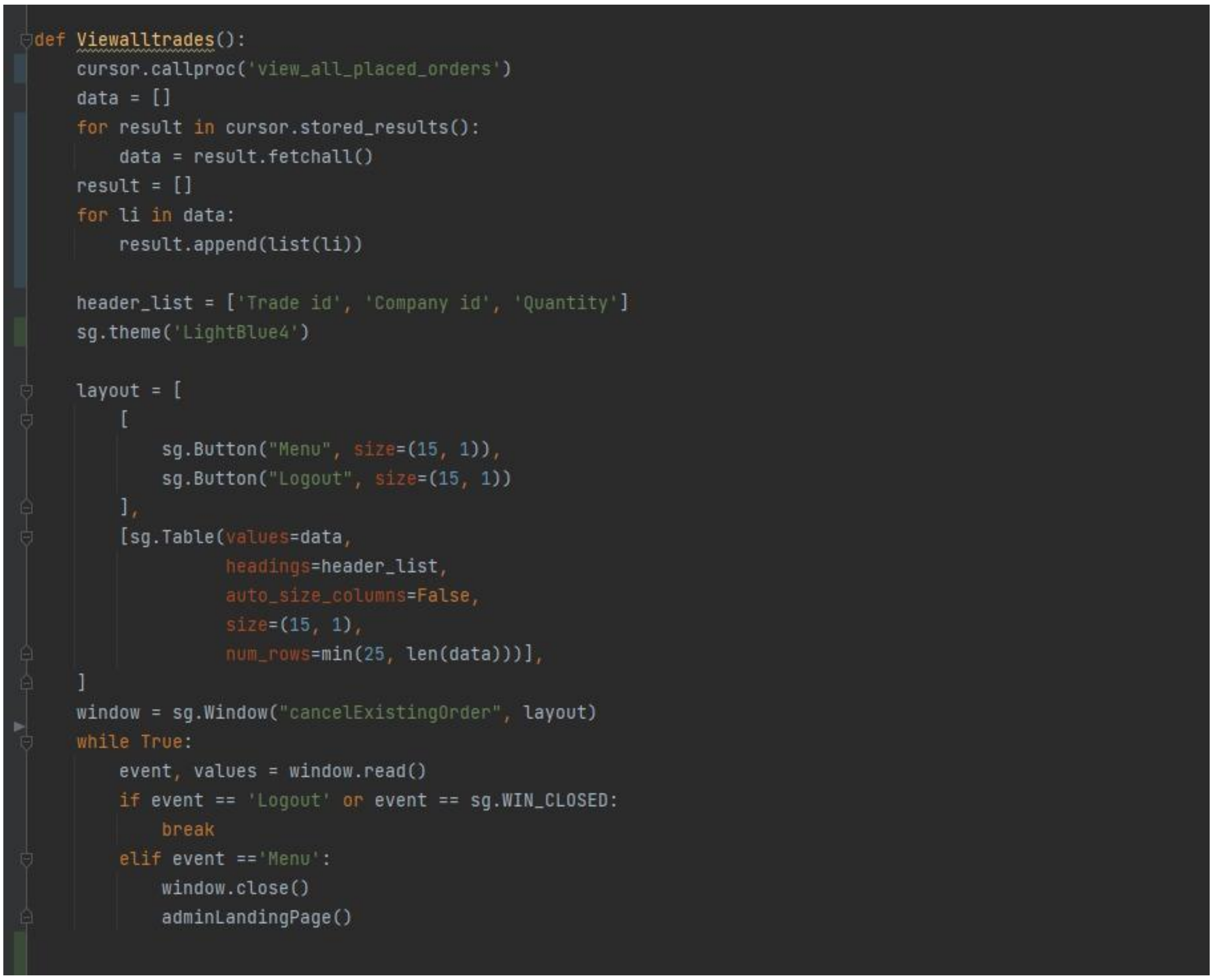
      ii. view_all_placed_orders

Using PYSIMPLE GUI TO DISPLAY THE RESULT OF THE STORED PROCEDURE



```python
def Viewplacedorders():
    cursor.callproc('view_all_placed_orders')
    data = []
    for result in cursor.stored_results():
        data = result.fetchall()
    result = []
    for li in data:
        result.append(list(li))

    header_list = ['Trade id', 'Company Share Code', 'Quantity', 'Purchased Price']
    sg.theme('LightBlue4')

    layout = [
        [
            sg.Button("Menu", size=(15, 1)),
            sg.Button("Logout", size=(15, 1))
        ],
        [sg.Table(values=data,
                  headings=header_list,
                  auto_size_columns=False,
                  size=(15, 1),
                  num_rows=min(25, len(data)))],
    ]
    window = sg.Window("cancelExistingOrder", layout)

    while True:
        event, values = window.read()
        if event == 'Logout' or event == sg.WIN_CLOSED:
            break
        elif event == 'Menu':
            window.close()
            adminLandingPage()
```

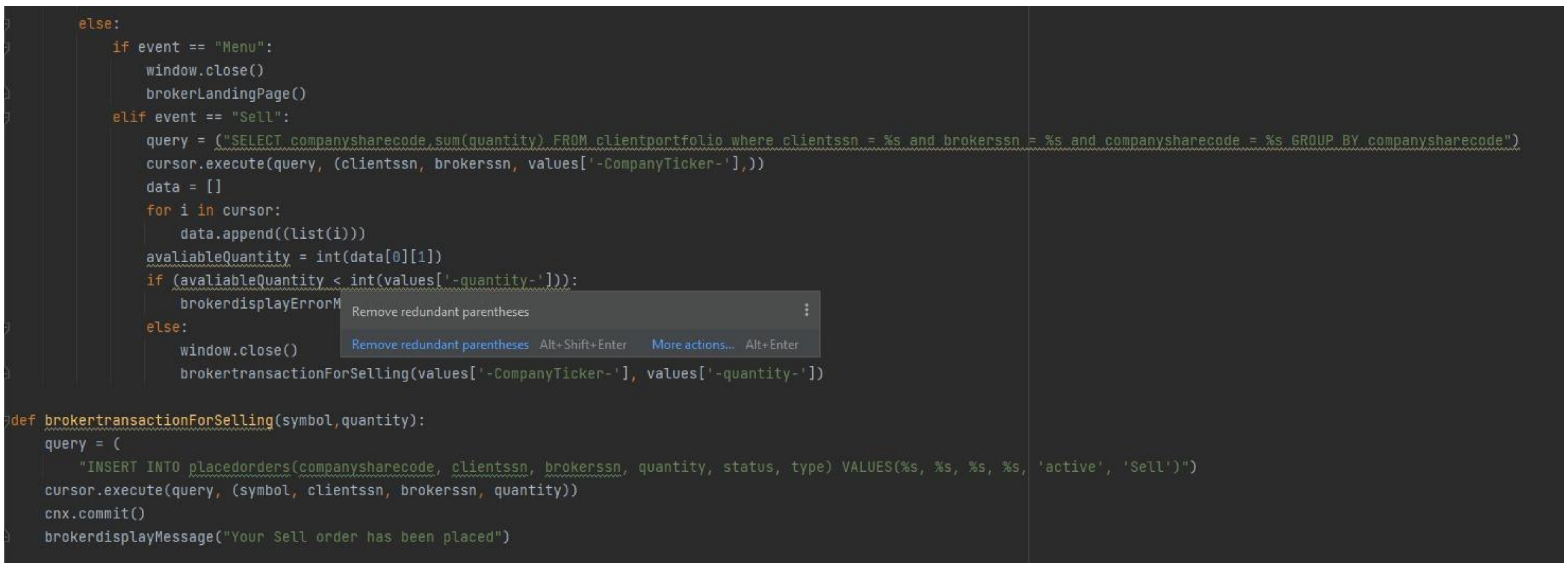
iii. view_all_trades



```python
def Viewalltrades():
    cursor.callproc('view_all_placed_orders')
    data = []
    for result in cursor.stored_results():
        data = result.fetchall()
    result = []
    for li in data:
        result.append(list(li))

    header_list = ['Trade id', 'Company id', 'Quantity']
    sg.theme('LightBlue4')

    layout = [
        [
            sg.Button("Menu", size=(15, 1)),
            sg.Button("Logout", size=(15, 1))
        ],
        [sg.Table(values=data,
                  headings=header_list,
                  auto_size_columns=False,
                  size=(15, 1),
                  num_rows=min(25, len(data)))],
    ]
    window = sg.Window("cancelExistingOrder", layout)
    while True:
        event, values = window.read()
        if event == 'Logout' or event == sg.WIN_CLOSED:
            break
        elif event =='Menu':
            window.close()
            adminLandingPage()
```

Final design of DB apps portion

1. Any specific functionality involving accessing more than one table (e.g., read t1 and then update t2) ▪ any explicit multi-SQL-statement DB transactions that modify data and are initiated from DB apps side are used to implement such functionality • If so, show code snippet of multi-SQL-statement DB transactions.



```
        else:
            if event == "Menu":
                window.close()
                brokerLandingPage()
            elif event == "Sell":
                query = ("SELECT companysharecode,sum(quantity) FROM clientportfolio where clientssn = %s and brokerssn = %s and companysharecode = %s GROUP BY companysharecode")
                cursor.execute(query, (clientssn, brokerssn, values['-CompanyTicker-'],))
                data = []
                for i in cursor:
                    data.append((list(i)))
                avaliableQuantity = int(data[0][1])
                if (avaliableQuantity < int(values['-quantity-'])):
                    brokerdisplayErrorM
                else:
                    window.close()
                    brokertransactionForSelling(values['-CompanyTicker-'], values['-quantity-'])

def brokertransactionForSelling(symbol,quantity):
    query = (
        "INSERT INTO placedorders(companysharecode, clientssn, brokerssn, quantity, status, type) VALUES(%s, %s, %s, %s, 'active', 'Sell')")
    cursor.execute(query, (symbol, clientssn, brokerssn, quantity))
    cnx.commit()
    brokerdisplayMessage("Your Sell order has been placed")
```

Remove redundant parentheses
Remove redundant parentheses Alt+Shift+Enter
More actions... Alt+Enter

Any major design decisions, trade-offs (and why)

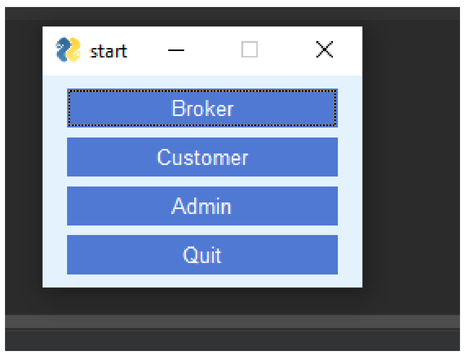
1. We did not create separated placed orders objects for customer and client instead of the we created only one table which will keep track of all placed orders.
2. We created primary key in customer portfolio and made entity a strong entity with the number of trades that gets completed. Id of customer portfolio is auto incremented and a primary key.

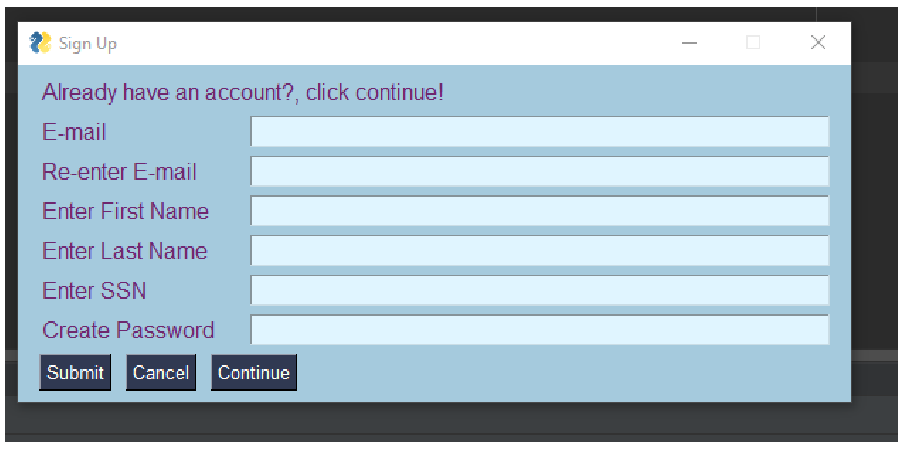
Any major modifications from proposal, ERD and why

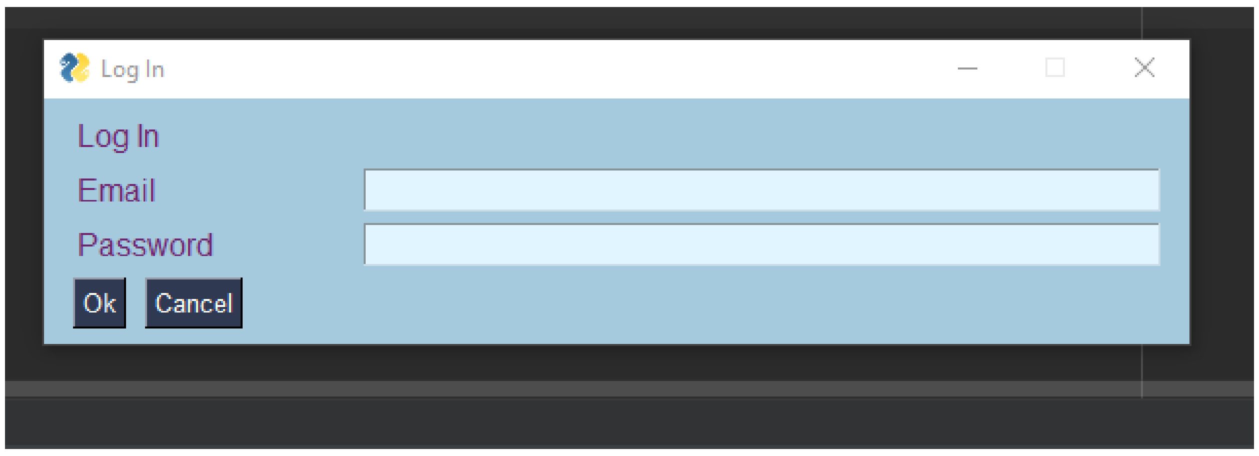
1. Keeping track of placed order in one single table made application easier to access data and implement functionality on top of it.

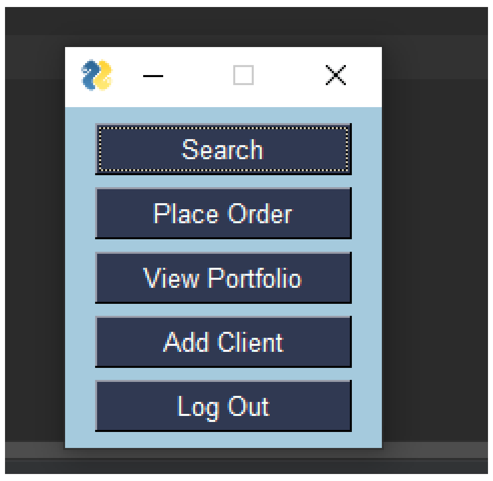
Functionality test cases and test plan execution

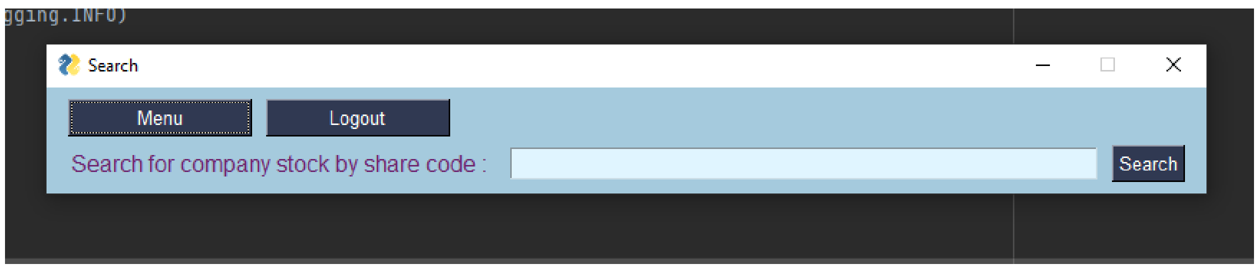
Broker

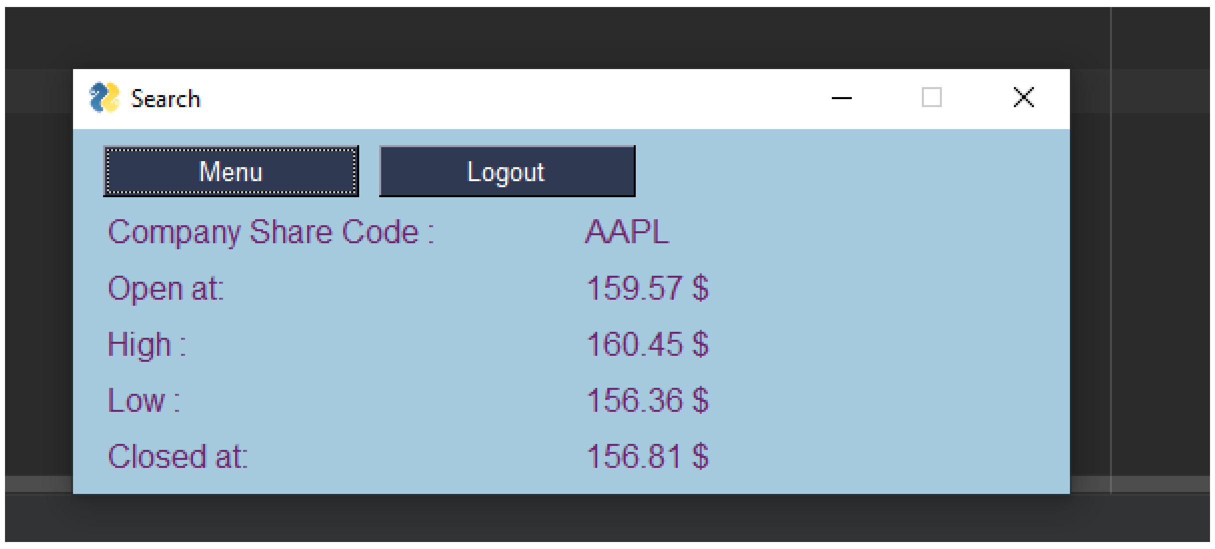




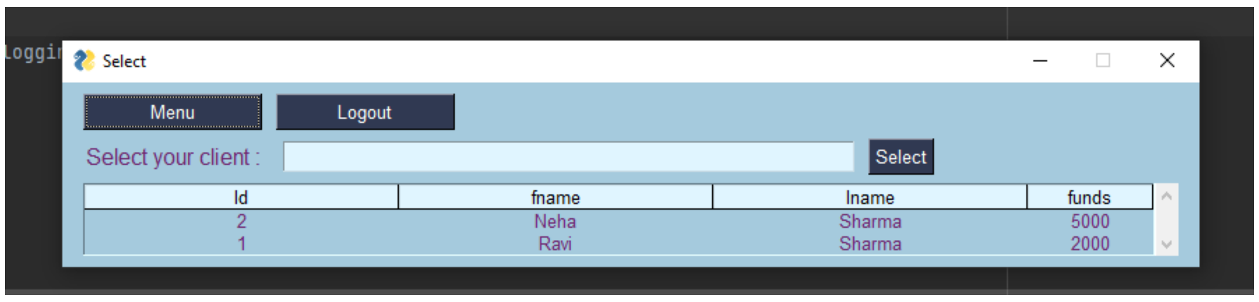






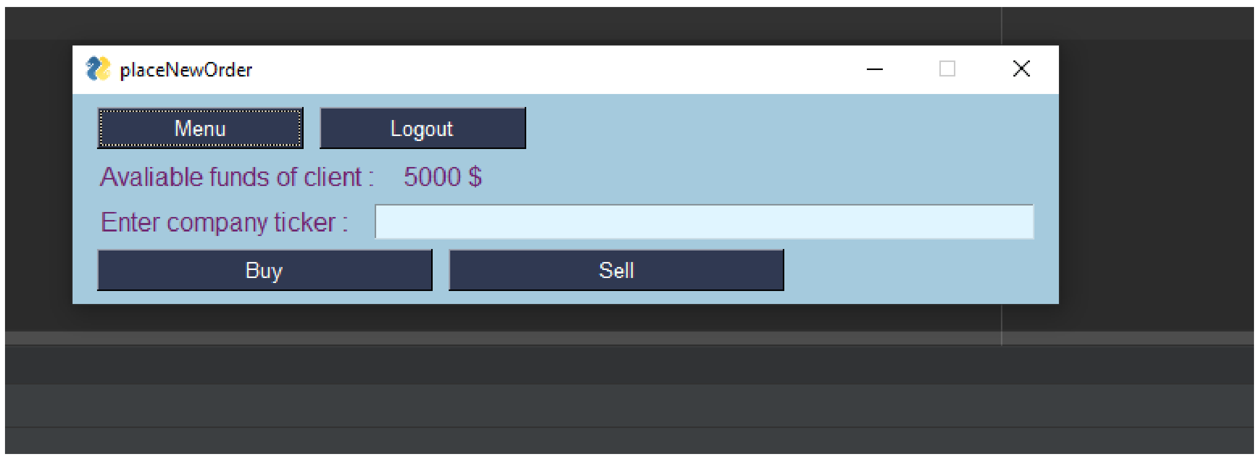


Company Share Code :	AAPL
Open at:	159.57 $
High :	160.45 $
Low :	156.36 $
Closed at:	156.81 $



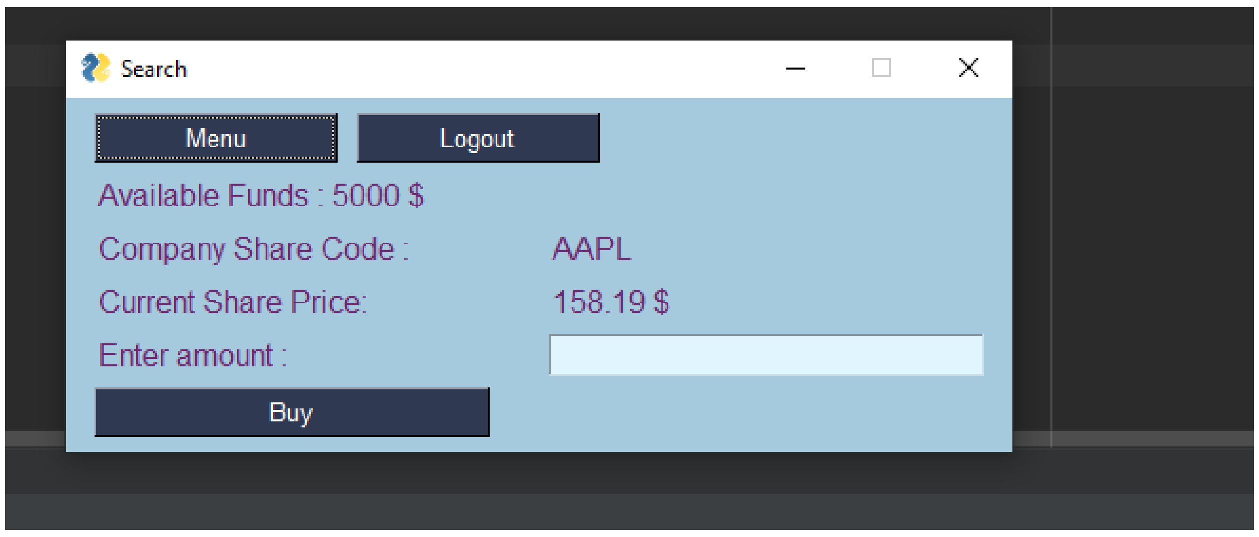
Select your client :

Id	fname	lname	funds
2	Neha	Sharma	5000
1	Ravi	Sharma	2000



Avaliable funds of client : 5000 $

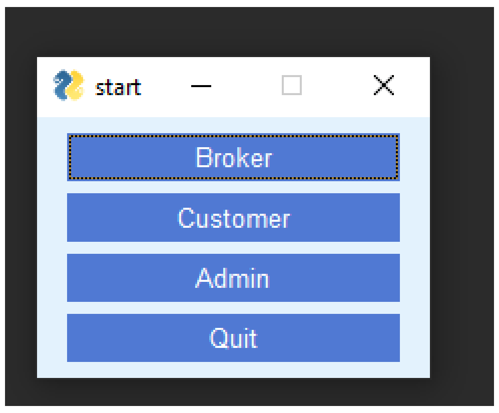
Enter company ticker :

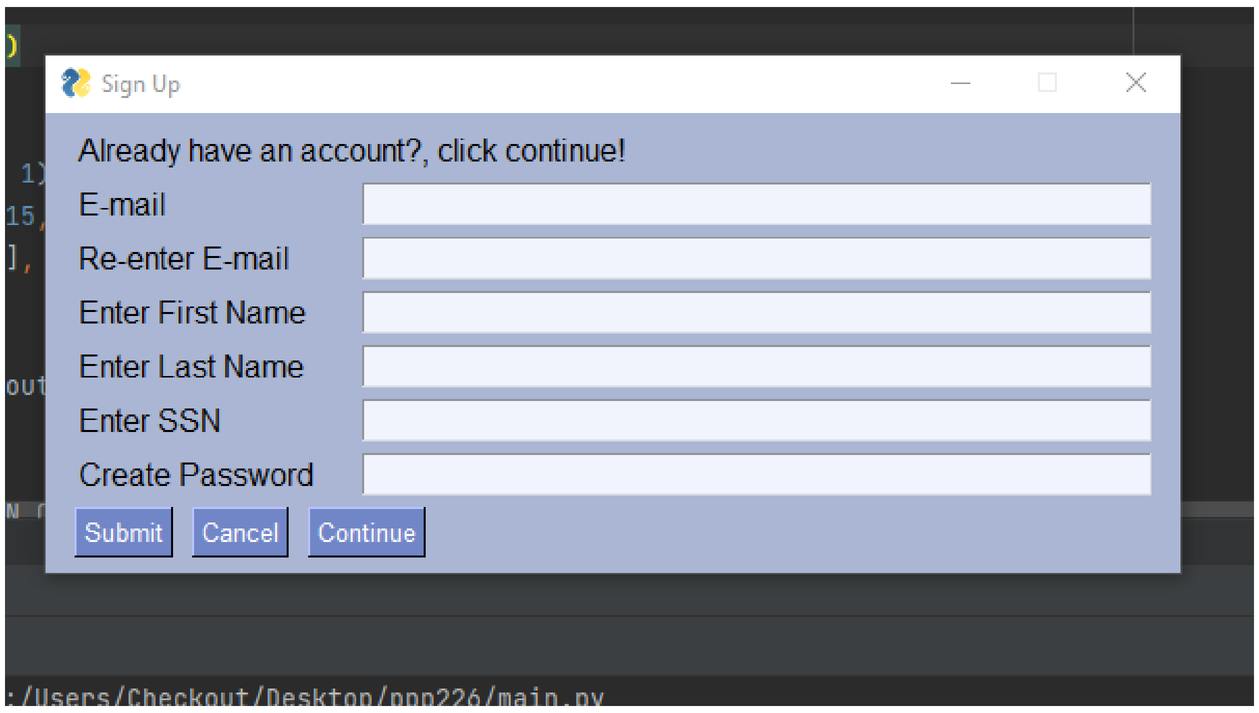


Available Funds : 5000 $

Company Share Code :	AAPL
Current Share Price:	158.19 $
Enter amount :	

Customer




Sign Up

Already have an account?, click continue!

E-mail

Re-enter E-mail

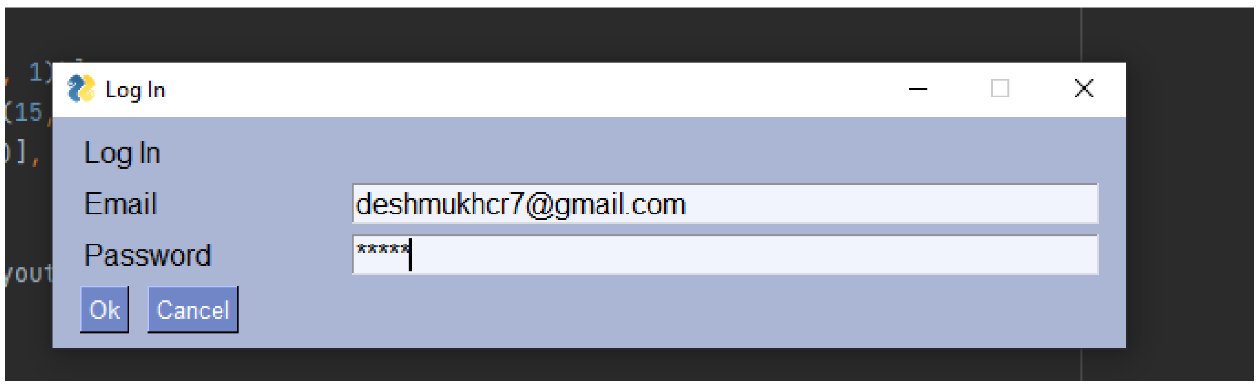
Enter First Name

Enter Last Name

Enter SSN

Create Password

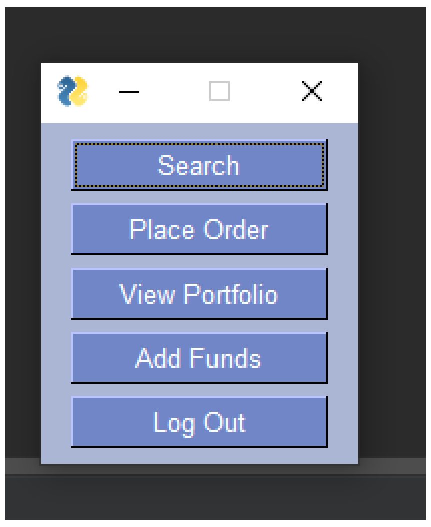
```
:/Users/Checkout/Desktop/ppp226/main.py
```

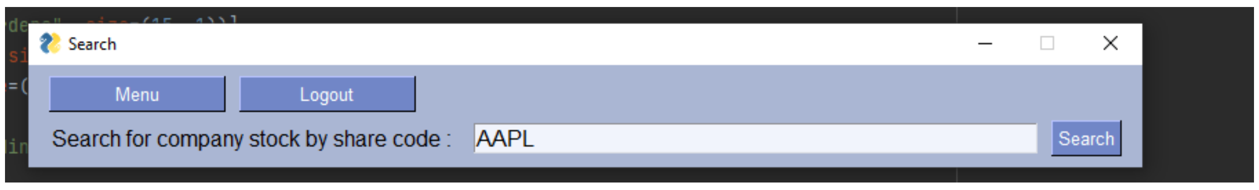

Log In

Log In

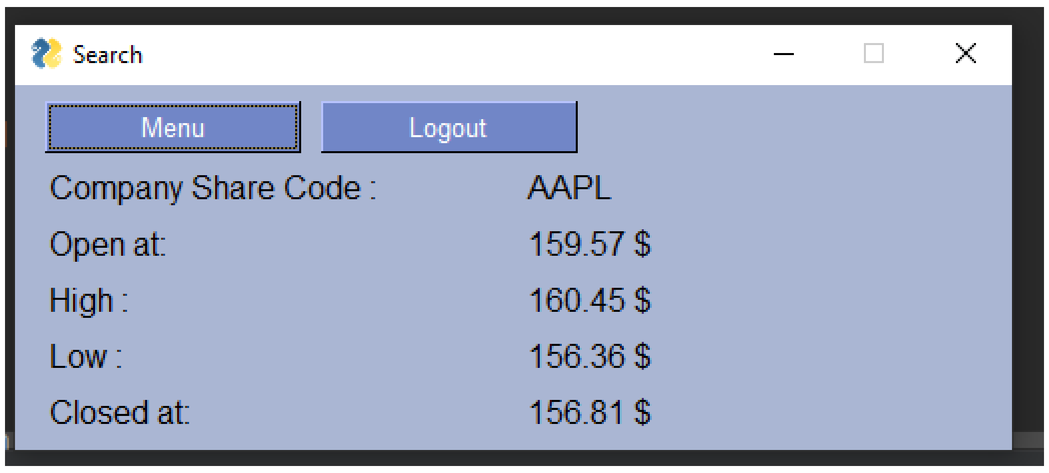
Email: deshmukhcr7@gmail.com

Password: *****

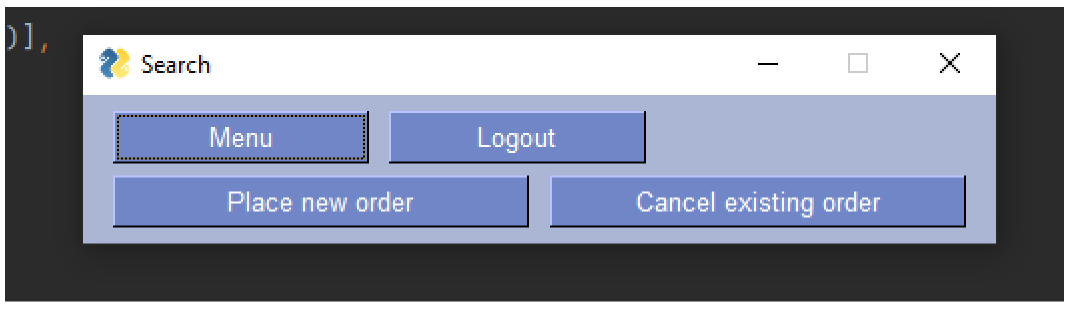


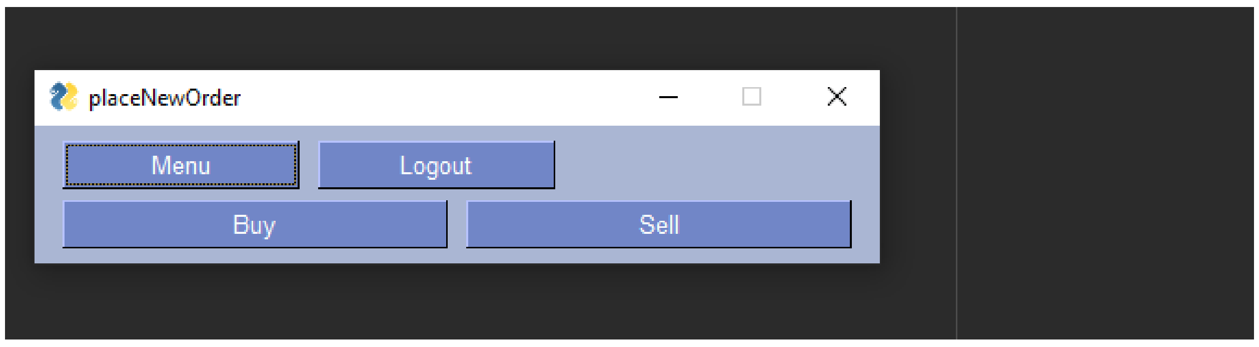

Search

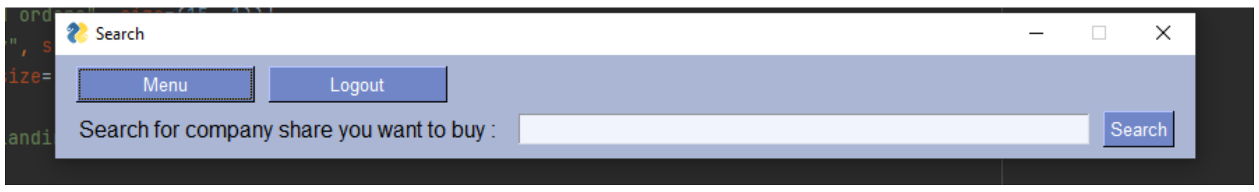
Search for company stock by share code : AAPL

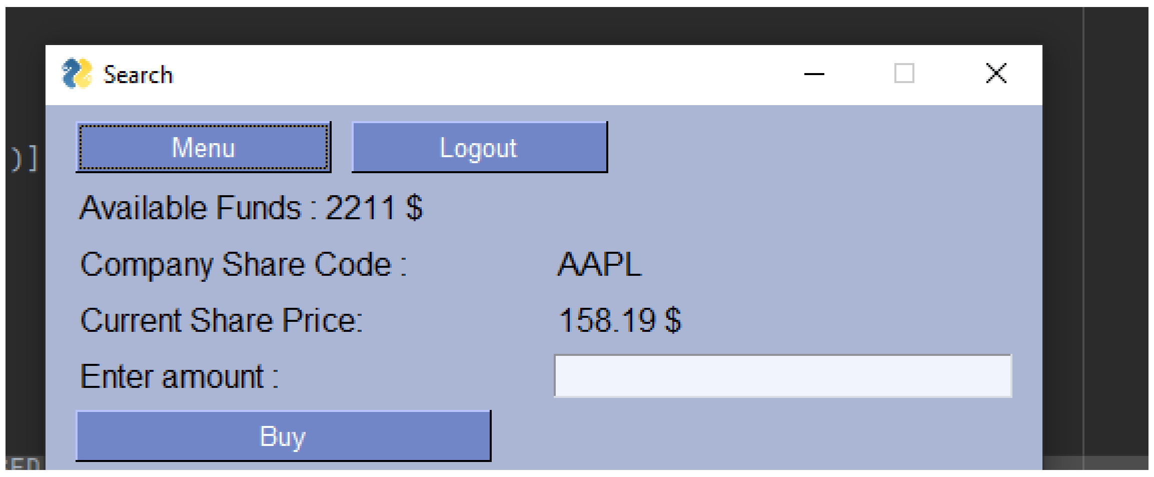

Search

Company Share Code :	AAPL
Open at:	159.57 $
High :	160.45 $
Low :	156.36 $
Closed at:	156.81 $


Search







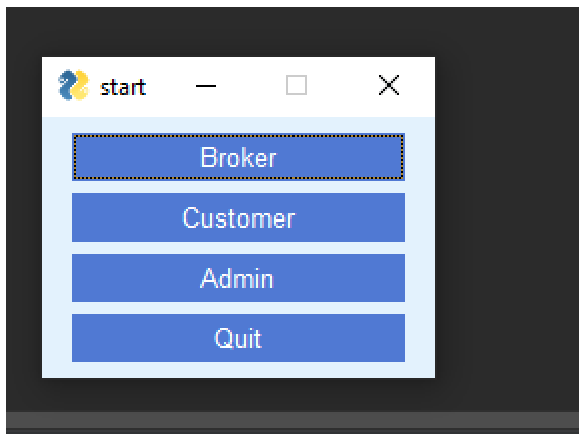
Available Funds : 2211 $

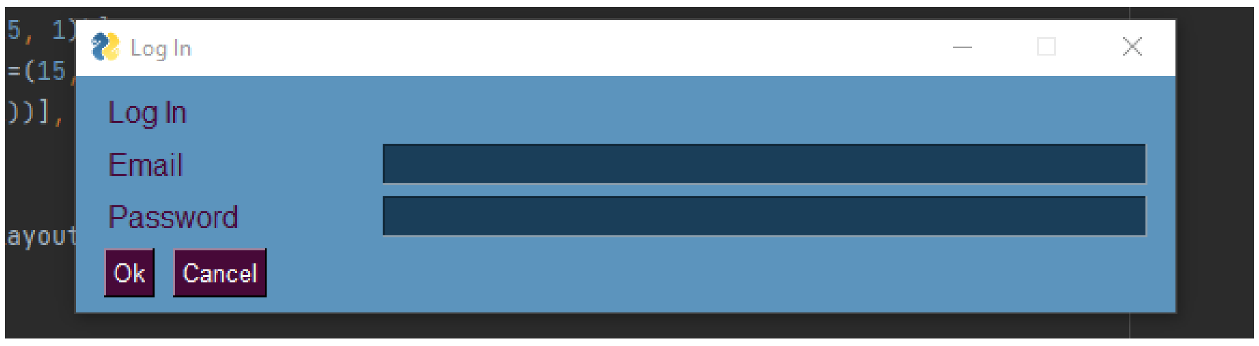
Company Share Code : AAPL

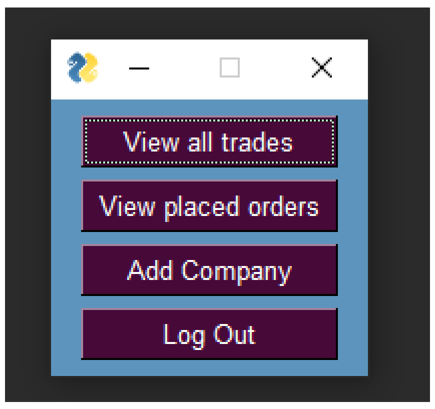
Current Share Price: 158.19 $

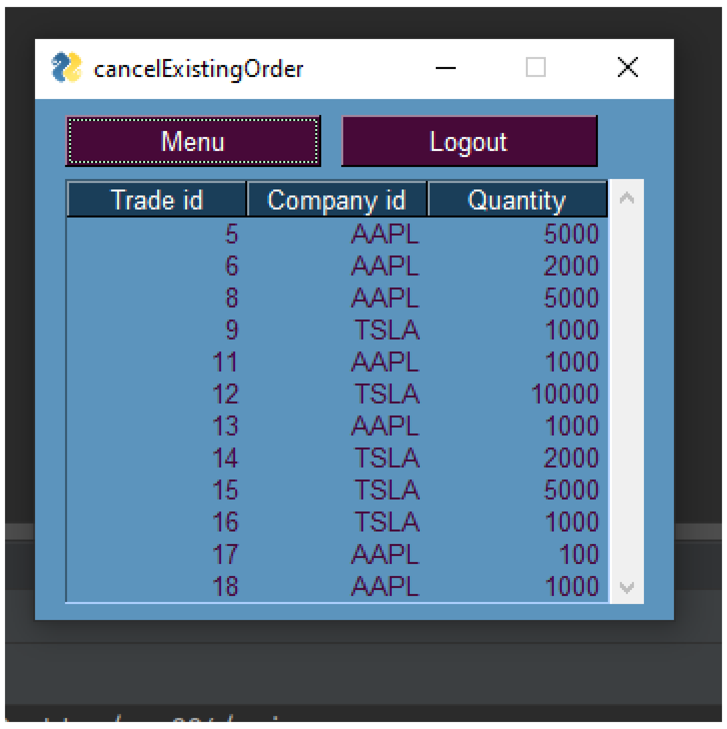
Enter amount :

Admin

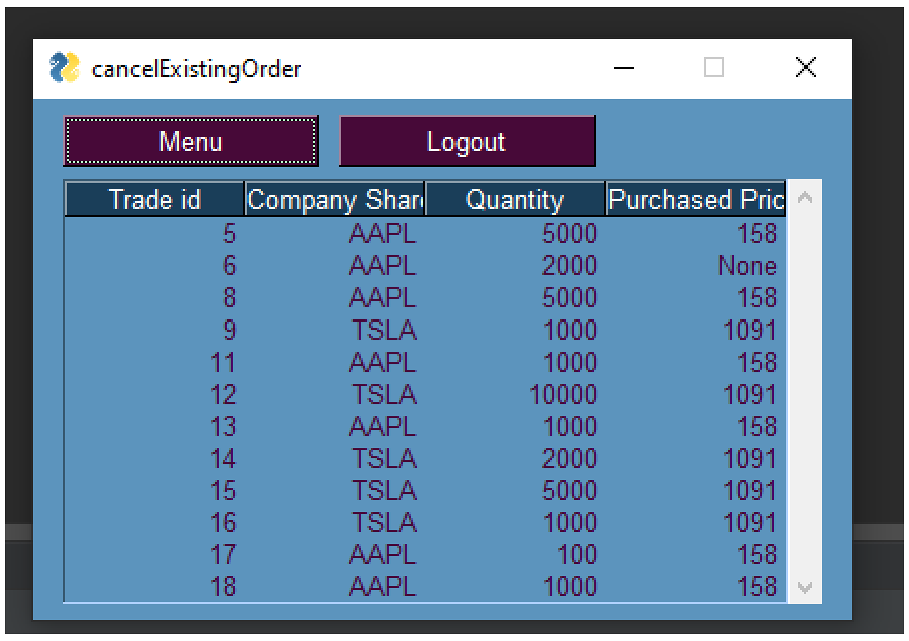




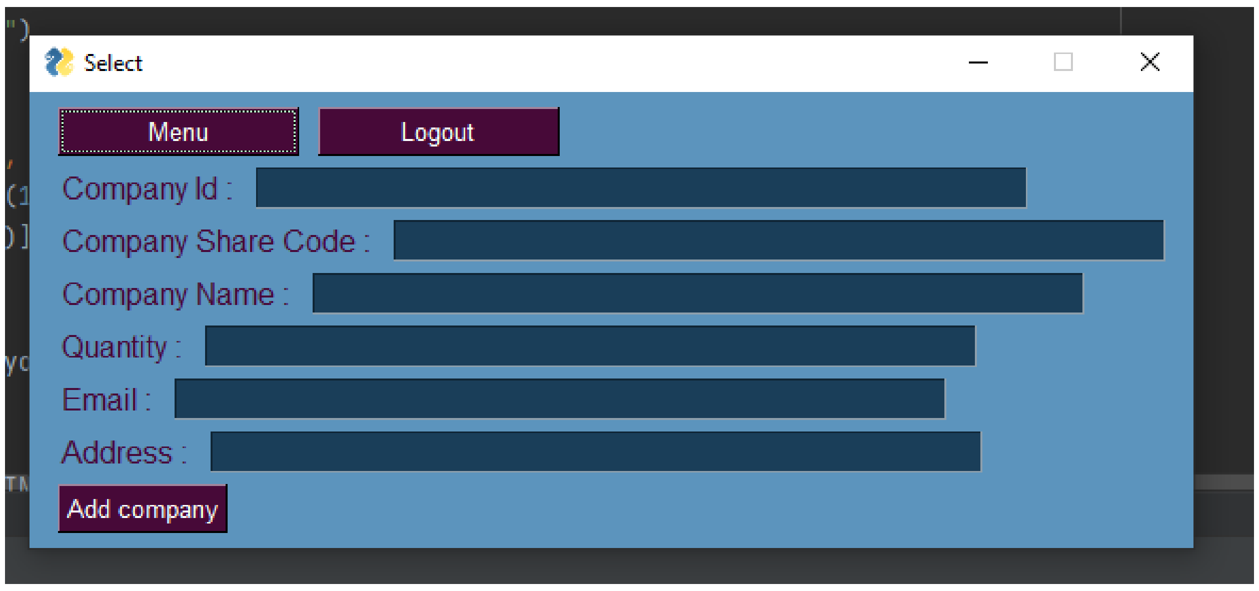




Trade id	Company id	Quantity
5	AAPL	5000
6	AAPL	2000
8	AAPL	5000
9	TSLA	1000
11	AAPL	1000
12	TSLA	10000
13	AAPL	1000
14	TSLA	2000
15	TSLA	5000
16	TSLA	1000
17	AAPL	100
18	AAPL	1000



Trade id	Company Shar	Quantity	Purchased Pric
5	AAPL	5000	158
6	AAPL	2000	None
8	AAPL	5000	158
9	TSLA	1000	1091
11	AAPL	1000	158
12	TSLA	10000	1091
13	AAPL	1000	158
14	TSLA	2000	1091
15	TSLA	5000	1091
16	TSLA	1000	1091
17	AAPL	100	158
18	AAPL	1000	158



Company Id :	
Company Share Code :	
Company Name :	
Quantity :	
Email :	
Address :	

Project postmortem

1. Implementing a FIFO queue to handle the trades is not completed and part of future work.
2. Instead of implementing we have introduced one new primary key Id auto incremental which when sorted on application can be used to first complete transactions in first come first serve.
3. Selling of trades puts order into active state which can be directly be used by broker to complete the trade with another buyer.