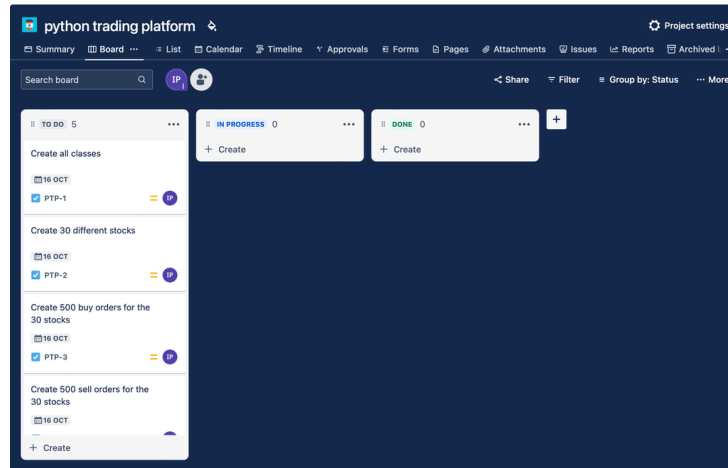


Python script for an electronic trading platform

Before the trading platform was coded, the first step was to create a list of what needed to be done. This broke down the assignment into smaller sections, which made the project easier to manage. This was recorded in Jira (image below.)



My approach to the code was to create a menu for a user to interact with the trading platform. This helped me break down the task into nine sections, so each section could be done separately, to ensure each section of code was running correctly when a person would interact with the programme.

The code below was for creating a function to define the menu, the second part of the menu function was to create a function to interact with the menu, and the first input option lets you pick which option you want from the menu. The if and elif options code what happens when you choose an option. The else code showed if you picked an integer that was outside of one to seven.

```
1 def menu():#This will print so user can see what the options are
2     print("Izzy's Electronic Trading platform:")
3     print("1) Register ")
4     print("2) Login")
5     print("3) View buy and sell orders for all stocks")
6     print("4) Add a new order (buy/sell)")
7     print("5) Cancel an order")
8     print("6) Replace an order")
9     print("7) View the status of all orders/ View your trade history")
10    print("8) Logout")
11
12 while True:
13     menu()
14     option = int(input("Please enter your choice from the menu: "))
15
16     if option == 8:
17         pass
18
19     elif option == 1:
20         pass
21
22     elif option == 2:
23         pass
24
25     elif option == 3:
26         pass
27
28     elif option == 4:
29         pass
30
31     elif option == 5:
32         pass
33
34     elif option == 6:
35         pass
36
37     elif option == 7:
38         pass
39
40     else:
41         print(" Please log in or register to access options. ")
```

After creating the menu, I chose to code the logout feature (option eight). This option breaks the loop created in the input function and leaves the menu. Break is the word used to end the loop, when typing in 8 to the input it prints "Logged out. Thank you for using Izzy's Electronic Trading platform." The code below shows this. The login_in = False will log the user out of the platform.

```
1 if option == 8:
2     print("Logged out. Thank you for using Izzy's Electronic Trading platform.")
3     logged_in = False
```

```
4         break
```

The next stage was to code the user class and at the same time code this feature into the menu. The code below shows the user class created, this was split into two functions (register_user and login_user). The register_user function writes a new row of code into the file called users.csv. The 'a' in the with open line appends a new row to the already existing file. The login_user is called on when the user tries to log into the trading platform. The function def login_user reads the existing file (users.csv) to see if the information that the user input is in the file. If the information exists the user will log into the platform. The return true, links to the menu function, as this allows the user to choose any option in the function. The base_dir finds the correct files when coding in the terminal, so that python can find the files in the directory. I firstly created a folder in downloads called python_assessment, with the python file text script and the files (once created could be found in the folder.)

```
1 import csv
2 import getpass
3 import sys
4
5 base_dir = '/Users/izzy/Downloads/python_assessment'
6 User_File = os.path.join(base_dir, 'users.csv')
7 file1 = os.path.join(base_dir, 'buy_orders.csv')
8 file2 = os.path.join(base_dir, 'sell_orders.csv')
9 merged_file = os.path.join(base_dir, 'trade_history.csv')
10
11 class UserAuth:
12     def register_user(self, username, password):
13         # Store username and password
14         with open(User_File, 'a', newline='') as file:
15             writer = csv.writer(file)
16             writer.writerow([username, password]) # Store plain password
17
18         print(f"User {username} registered successfully.")
19
20     def login_user(self, username, password):
21         with open(User_File, 'r') as file:
22             reader = csv.reader(file)
23             for row in reader:
24                 if row[0] == username and row[1] == password:
25                     print(f"User {username} logged in successfully.")
26                     return True # Login successful
27             print("Login failed. Invalid credentials.")
28             return False # Login failed
29
30 user_info = UserAuth()
31 #this is an instance, so the class can be called on in menu
32
```

The code below shows the updated menu for registering and logging in, the logged_in = False, makes sure the user is required to register or login before using the platform, otherwise the users will see " Please log in or register to access options. " if trying to input options three to seven. The code for the password includes a getpass, this makes sure the user cannot see the password being typed or created and will not be seen when printed in python.

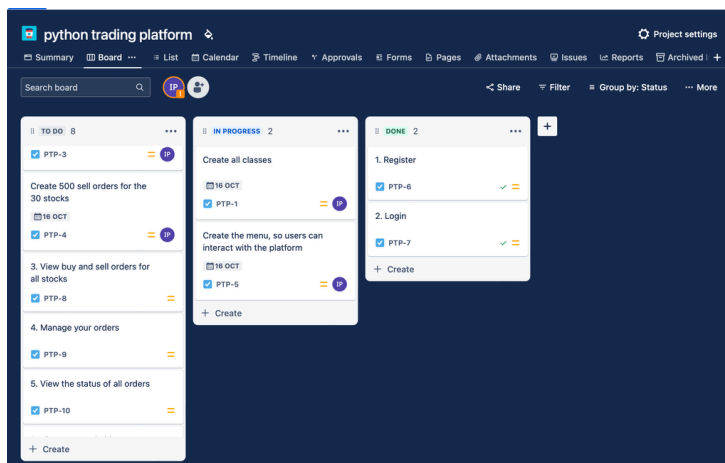
```
1 logged_in = False
2
3 while True:
4     menu()
5     option = int(input("Please enter your choice from the menu: "))
6
7     if option == 8:
8         print("Logged out. Thank you for using Izzy's Electronic Trading platform.")
9         logged_in = False
10        break
11
12    elif option == 1:
13        print(" Please create a username and password ")
14
15        username = input(" Please create a username : ")
16
17        password = getpass.getpass("Please create a password: ")
18
19        user_info.register_user(username, password)
20        #this line of code links the information back to the correct class and also the correct function inside the class.
21
22        print(f"User {username} registered successfully.")
23
24        print("( Username and password created please login. )")
25
26    elif option == 2:
27        print(" Please enter your username and password ")
28        username = input("Please enter your username: ")
29        password = getpass.getpass("Please enter your password: ")
30        success = user_info.login_user(username, password)
31        #calls to read file and see if the login information exists.
32
33        if success:
34            logged_in = True # Set login state to True(lets you use the rest of the menu.)
35            print(f"Welcome back, {username}!")
36        else:
37            print("Login attempt failed.")
38
```

```

39     elif logged_in:
40         if option == 3:

```

The below image shows my progress which was recorded in JIRA.



The next step was to create three classes for stock, buy_orders and sell_orders. These classes are below. The stock class is the parent class and has two child classes for buy and sell stock. This means the buy/sell classes have access to the stock class.

```

1 class Stock:
2     def __init__(self, name, price, quantity):
3         self.name = name
4         self.price = price
5         self.quantity = quantity
6
7     def stock_info(self):
8         return "Name : " + str(self.name) + " Stock Price : " + str(self.price) + " Stock Quantity : " + str(self.quantity)
9
10
11 class BuyStock(Stock):
12     def __init__(self, order_id, name, price, quantity):
13         super().__init__(name, price, quantity)
14         self.order_id = len(buy_list) + 1
15         self.action = 'buy '
16         self.status = 'pending '
17
18     def stock_info(self):
19         return " Order ID: " + str(self.order_id) + " Status: " + self.status + " Action: " + str(self.action) + super().stock_info()
20
21 class SellStock(Stock):
22     def __init__(self, order_id, name, price, quantity):
23         super().__init__(name, price, quantity)
24         self.order_id = len(sell_list) + 1
25         self.action = 'sell '
26         self.status = 'pending '
27
28     def sell_info(self):
29         return " Order ID: " + str(self.order_id) + " Status: " + self.status + " Action: " + str(self.action) + super().stock_info()

```

After creating the classes: the stock list, buy list and sell list was created and generated into csv files. Generating the data straight into files led me to run an error, as the write function ('w') would replace all my data with new generated data, each time the programme was ran. To overcome this error, I imported os, and put the files in a if function, so the file would only be created once and all the user data wouldn't be overwritten. The loops in the buy and sell list, create 10 buy and sell orders for the predefined list of stocks I created. The loop randomly generates a price and quantity 10 times for each of my thirty stocks. The files created first run through the classes, this ensures a random ordering and a status of pending is added to each buy and sell order, which is then written into the file for the buy and sell list.

```

1 #Predefined list of stocks
2 stock_names = ['AAPL', 'GOOG', 'AMZN', 'MSFT', 'TSLA', 'NFLX', 'FB', 'NVDA', 'BABA', 'DIS', 'JPM', 'V', 'PG', 'BMW', 'MERC', 'PEP', 'COST', 'STBX', 'HD', 'JNJ',
3               'UNI', 'KFC', 'ESYJ', 'BA', 'INTC', 'CSCO', 'ADBE', 'PYPL', 'MCD', 'ORCL']
4
5 stock_list = []
6
7 for i in range(30):
8     name = stock_names[i]
9     price = round(random.uniform(0, 500), 2) # Random price between 0 and 500
10    quantity = round(random.uniform(0, 50), 0) # Random quantity between 0 and 50
11    stock_list.append(Stock(name, price, quantity)) # Append stock to the list
12
13 # Display Stock Information
14 print("Stock List:")
15 for stock in stock_list:
16     print(stock.stock_info())

```

```

17 #Display Buy Information
18 print("Buy List:")
19 for buy_stock in buy_list:
20     print(buy_stock.stock_info())
21
22 #Display Sell Information
23 print("Sell List:")
24 for sell_stock in sell_list:
25     print(sell_stock.sell_info())
26
27 #need to define, so the files can be created if they do not exist.
28 file1 = 'buy_orders.csv'
29 file2 = 'sell_orders.csv'
30 merged_file = 'merged_file.csv'
31 buy_list = []
32 sell_list = []
33
34 # Check if the CSV file buy_list exists
35 if os.path.exists(file1):
36     print("CSV file already exists. Skipping data input...")
37 else:
38     print("Generating new stock data and writing to CSV...")
39
40 for stock in stock_list:
41     for i in range(10): # Generate 10 buy and 10 sell orders for each stock
42         buy_price = round(random.uniform(0, 500), 2)
43         buy_quantity = round(random.uniform(0, 100), 0)
44         buy_list.append(BuyStock(i+1, stock.name, buy_price, buy_quantity)) # Add buy order with unique ID
45     with open('buy_orders.csv', 'w', newline='') as csvfile:
46         write = csv.writer(csvfile)
47         write.writerow(['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price', 'Buy Quantity']) # Wrote a header for each column
48     for buy_stock in buy_list:
49         write.writerow([buy_stock.order_id, buy_stock.status, buy_stock.action, buy_stock.name, buy_stock.price, buy_stock.quantity])
50
51 # Check if the CSV file sell_list exists
52 if os.path.exists(file2):
53     print("CSV file already exists. Skipping data input...")
54 else:
55     print("Generating new stock data and writing to CSV...")
56
57 for stock in stock_list:
58     for i in range(10):
59         sell_price = round(random.uniform(0, 500), 2)
60         sell_quantity = round(random.uniform(0, 100), 0)
61         sell_list.append(SellStock(i+1, stock.name, sell_price, sell_quantity)) # Add sell order with unique ID
62     with open('sell_orders.csv', 'w', newline='') as csvfile:
63         write = csv.writer(csvfile)
64         write.writerow(['Order ID', 'Status', 'Action', 'Stock Name', 'Sell Price', 'Sell Quantity'])
65     for sell_stock in sell_list:
66         write.writerow([sell_stock.order_id, sell_stock.status, sell_stock.action, sell_stock.name, sell_stock.price, sell_stock.quantity])
67
68 #creating a trade_history file
69 if os.path.exists(merged_file):
70     print("CSV file already exists. Skipping data input...")
71 else:
72     print("Generating new stock data and writing to CSV...")
73
74 with open(file1, 'r') as f1:
75     reader1 = csv.reader(f1)
76     header1 = next(reader1)
77     data1 = list(reader1)
78
79 # Read data from second file
80 with open(file2, 'r') as f2:
81     reader2 = csv.reader(f2)
82     header2 = next(reader2)
83     data2 = list(reader2)
84
85 # List of CSV file names to merge
86 file_names = ['buy_orders.csv', 'sell_orders.csv']
87
88 # Output file name
89 trade_history = 'trade_history.csv'
90
91 with open(trade_history, 'w', newline='') as f_out:
92     writer = csv.writer(f_out)
93     writer.writerow(['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price', 'Buy Quantity'] + ['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price',
94     'Buy Quantity']) # Merge headers
95     for row1, row2 in zip(data1, data2):
96         writer.writerow(row1 + row2) # Merge rows

```

Will the files added, the menu was updated with option 3) to view buy or sell orders. The input functions lets a user choose to either view the buy or sell csv file. This then links to the of function, if buy was chosen then the with function will read in the buy orders file and print all the content. This is the same for the sell function.

```

1 if option == 3:

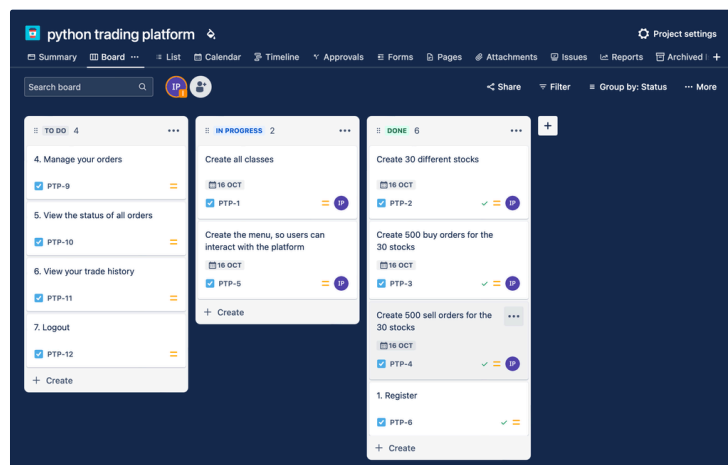
```

```

2         print("Choose to view either buy or sell orders for all stocks.")
3
4         buy_or_sell = input(" Input choice : ").lower() #ensures the input option is in lower case to match to the class
5
6         if buy_or_sell == 'buy':
7             with open('buy_orders.csv', 'r') as csvfile:
8                 csv_reader = csv.reader(csvfile)
9                 for line in csv_reader:
10                     print(line)
11
12
13         elif buy_or_sell == 'sell':
14             with open('sell_orders.csv', 'r') as csvfile:
15                 csv_reader = csv.reader(csvfile)
16                 for line in csv_reader:
17                     print(line)
18         else:
19             print("Invalid choice. Please select buy or sell.")
20

```

The image below shows the progress in JIRA, with six tasks completed.



After this, the next stage of the code was to add/cancel/append an order in the csv files. I first started by coding the add orders. I created a new class called order manager to store the functions for adding new orders and also later on to add cancelled orders. I firstly created a function to add an order, with a if function if the user selected if they wanted to buy or sell a stock. Once a choice is made, the function will add a random order, then run through the buy or sell class to give it a status of 'pending'. The print function will show the user that the order has been successful, and will save to the csv file of buy or sell file. When saving to the csv file, 'a' will append to the file and add a new line to the existing file. (This will also append a new line to the trade history.)

```

1 class OrderManager:
2     def add_order(self, order_type, stock_name, price, quantity):
3         if order_type == 'buy':
4             order_id = round(random.randint(0, 500000), 0) # Unique ID for the new order
5             new_order = BuyStock(order_id, stock_name, price, quantity)
6             print(f"Added Buy Order: {new_order.stock_info()}")
7             self.save_buy_order(new_order) # Save to CSV
8         elif order_type == 'sell':
9             order_id = round(random.randint(0, 500000), 0) # Unique ID for the new order
10            new_order = SellStock(order_id, stock_name, price, quantity)
11            print(f"Added Sell Order: {new_order.sell_info()}")
12            self.save_sell_order(new_order) # Save to CSV
13
14    def save_buy_order(self, order):
15
16        with open('buy_orders.csv', 'a', newline='') as csvfile: # Append mode
17            writer = csv.writer(csvfile)
18            writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity]) # Append new order
19
20        with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
21            writer = csv.writer(csvfile)
22            writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity]) # Append new order
23
24    def save_sell_order(self, order):
25        with open('sell_orders.csv', 'a', newline='') as csvfile: # Append mode
26            writer = csv.writer(csvfile)
27            writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity]) # Append new order
28
29        with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
30            writer = csv.writer(csvfile)
31            writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity]) # Append new order
32
33    order_manager = OrderManager() #this is the instance for the order manager class, this calls on the class when the class is needed.
34

```

The next step, was to code link this new class created into the menu. If option 4 was chosen from the menu the code below, asks the user to input in they would like to buy or sell stock, then asks the user to input the stock name, stock price and quantity. The code would then call on the order manager class which will append this information into the correct files.

```
1 elif option == 4:
2     manage_order = input(" Please enter if you would like to place a buy or sell order : ").lower()
3     if manage_order == 'buy':
4         stock_name = input("Stock name: ").upper() #the upper saves the information as uppercases, so it matches the format of the created stock
5         price = float(input("Price: ")) #this allows a user to input a price which includes decimals
6         quantity = int(input("Quantity: ")) #this only allows users to input a whole number, if the user puts in an integer python will say error
7         status = 'pending'
8         order_manager.add_order(manage_order, stock_name, price, quantity)
9         print(" Order added ")
10
11     if manage_order == 'sell':
12         stock_name = input("Stock name: ").upper()
13         while stock_name not in stock_list:
14             print("Invalid stock name. Please enter a valid stock name from the list:")
15             break
16         price = float(input("Price: "))
17         quantity = int(input("Quantity: "))
18         status = 'pending'
19         order_manager.add_order(manage_order, stock_name, price, quantity)
20         print(" Order added ")
21
22     else:
23         print("Invalid choice. Please select buy or sell.")
24
```

After adding a new order, the next step was to cancel and order this was done in the order manager class. The def_cancel_order function is to code whether the user inputs if they would like to cancel a buy or sell order. This code is important as the stocks are stored in separate files so separate codes are needed. This then automatically links to the def_cancel_order (for either buy or sell.) The next step of the code def_cancel_buy_order opens the existing buy order csv and puts the file into the buy_cancel_file list. When the user selects an order to cancel the next step reads through the file to match the order id, stock name, price and quantity. If the order is found the status is updated to cancelled and appended to the list. This append list is then re-written into the buy orders csv file but appened in the trade history file. This is done as the trade history file acts like an order book to keep all information. (This process is the same for cancelling a sell order.)

```
1 class OrderManager:
2     def cancel_order(self, order_id, order_type, stock_name, price, quantity):
3         if order_type == 'buy':
4             self.cancel_buy_order(order_id, stock_name, price, quantity) #saves to buy csv file
5         elif order_type == 'sell':
6             self.cancel_sell_order(order_id, stock_name, price, quantity) #saves to sell csv file
7         else:
8             print("Error: Please select either 'buy' or 'sell'.")
9
10    def cancel_buy_order(self, order_id, stock_name, price, quantity):
11        buy_cancel_file = []
12        order_found = False
13
14        with open('buy_orders.csv', 'r') as file:
15            reader = csv.reader(file)
16            header = next(reader) # Read the header
17            buy_cancel_file.append(header) # Keep the header for rewriting later and puts the file to the list above
18
19            for row in reader: #loops through each row in the file and match to the conditions below
20                if row[0] == str(order_id) and row[3] == stock_name and row[4] == str(price) and row[5] == str(quantity):
21                    row[1] = 'Cancelled' #this rewrites the status of the order to be cancelled
22                    print(f"Buy Order {order_id} for stock {stock_name} has been cancelled.")
23                    order_found = True # Mark that we found and updated the order
24
25                    buy_cancel_file.append(row) # Add the updated row (or original if not updated)
26
27        if order_found:
28            # Write back to the file
29            with open('buy_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
30                writer = csv.writer(csvfile)
31                writer.writerows(buy_cancel_file) # Write all rows back to the file
32
33            # Write to trade history as well
34            with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
35                writer = csv.writer(csvfile)
36                writer.writerow([order_id, 'Cancelled', 'buy', stock_name]) # Append new order
37        else:
38            print("Error: 'buy_orders.csv' order in file not found.")
39
40    def cancel_sell_order(self, order_id, stock_name, price, quantity):
41        sell_cancel_file = []
42        order_found = False
43
44        with open('sell_orders.csv', 'r') as file:
45            reader = csv.reader(file)
46            header = next(reader) # Read the header
47            sell_cancel_file.append(header) # Keep the header for rewriting later
48
49            for row in reader:
```

```

50         if row[0] == str(order_id) and row[3] == stock_name and row[4] == str(price) and row[5] == str(quantity):
51             row[1] = 'Cancelled'
52             print(f"Sell Order {order_id} for stock {stock_name} has been cancelled.")
53             order_found = True # Mark that we found and updated the order
54
55             sell_cancel_file.append(row) # Add the updated row (or original if not updated)
56
57         if order_found:
58             # Write back to the file
59             with open('sell_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
60                 writer = csv.writer(csvfile)
61                 writer.writerows(sell_cancel_file) # Write all rows back to the file
62
63             # Write to trade history as well
64             with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
65                 writer = csv.writer(csvfile)
66                 writer.writerow([order_id, 'Cancelled', 'sell', stock_name]) # Append new order
67         else:
68             print("Error: 'sell_orders.csv' order in file not found.")
69
70     order_manager = OrderManager()

```

After the cancelled order was created in the class, the next stage was to code this into the menu. This lets the user interact and input each criteria that is needed to cancel an order. The code below makes a user put in the order id, then the stock name, then the order type, then the price of the order and finally the quantity. The order will only be cancelled if all the information is correct, this stops the wrong order accidentally being cancelled as the file is overwritten each time a order is cancelled.

```

1  elif option == 5:
2      print("Please enter your order details to cancel an order")
3      order_id = int(input("Order ID to cancel: "))
4      stock_name = input("Stock name to cancel: ").upper()
5      order_type = input("Order type (buy/sell): ").lower()
6      price = float(input("Order price: "))
7      quantity = int(input("Order quantity: "))
8      status = 'cancelled'
9      order_manager.cancel_order(order_id, order_type, stock_name, price, quantity)#this code calls on the class order manager and then calls on the function
to cancel an order
10     print("Order successfully cancelled.")
11

```

The last section of managing an order, was to append an already existing order. I put this into a separate class called replaced order, this could be put into the order manager class but due to the length of the order manager class, but I created a new class just in case errors ran to pin point where the code could go wrong. This is the same principle as the cancel order, the first function will lead to either appending a buy or sell order. The code then automatically takes you to the next function def replace_buy_order or replace_sell_order; opens the existing buy order or sell order csv and puts the file into a new list. The user inputs the stock name and order is, this is then found in the file and a new price and quantity is appened to the existing row. The status then changes to 'pending (replaced order)', this is then written back into the buy or sell file, and the appened row is overwritten. This is also appended to the trade history file as a new line to keep the old data intact.

```

1  class ReplaceOrder:
2      def replace_order(self, order_id, stock_name, order_type, new_price, new_quantity):
3          if order_type == 'buy':
4              self._replace_buy_order(order_id, stock_name, new_price, new_quantity)
5          elif order_type == 'sell':
6              self._replace_sell_order(order_id, stock_name, new_price, new_quantity)
7          else:
8              print("Error, please select either buy or sell.")
9
10     def _replace_buy_order(self, order_id, stock_name, new_price, new_quantity):
11         buy_file = []
12         order_found = False
13
14         with open('buy_orders.csv', 'r') as file:
15             reader = csv.reader(file)
16             header = next(reader) # Read the header
17             buy_file.append(header) # Keep the header for rewriting later
18
19         for row in reader:
20             if row[0] == str(order_id) and row[3] == stock_name:
21                 row[4] = str(new_price) # Update the price
22                 row[5] = str(new_quantity) # Update the quantity
23                 row[1] = 'Pending (Replaced Order)'
24                 print(f"Buy Order {order_id} for stock {stock_name} has been replaced with new price: {new_price}, quantity: {new_quantity}, and status:
{row[2]}".)
25                 order_found = True # Mark that we found and updated the order
26
27                 buy_file.append(row) # Add the updated row (or original if not updated)
28
29         if order_found:
30             # Write back to the file
31             with open('buy_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
32                 writer = csv.writer(csvfile)
33                 writer.writerows(buy_file) # Write all rows back to the file
34
35             # Write to trade history as well
36             with open('trade_history.csv', 'a', newline='') as csvfile: # This adds a new row so all the hstory of users is stored
37                 writer = csv.writer(csvfile)

```

```

38         writer.writerow([order_id, 'Pending (Replaced Order)', 'buy', stock_name, new_price, new_quantity]) # Append new order
39
40     else:
41         print(f"Buy Order {order_id} not found for stock {stock_name}.")
42
43     def _replace_sell_order(self, order_id, stock_name, new_price, new_quantity):
44         sell_file = []
45         order_found = False
46
47         with open('sell_orders.csv', 'r') as file:
48             reader = csv.reader(file)
49             header = next(reader) # Read the header
50             sell_file.append(header) # Keep the header for rewriting later
51
52             for row in reader:
53                 if row[0] == str(order_id) and row[3] == stock_name:
54                     row[4] = str(new_price) # Update the price
55                     row[5] = str(new_quantity) # Update the quantity
56                     row[1] = 'Pending (Replaced Order)'
57                     print(f"Sell Order {order_id} for stock {stock_name} has been replaced with new price: {new_price}, quantity: {new_quantity}, and status:
{row[2]}".)
58                     order_found = True # Mark that we found and updated the order
59
60                     sell_file.append(row) # Add the updated row (or original if not updated)
61
62             if order_found:
63                 # Write back to the file
64                 with open('sell_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
65                     writer = csv.writer(csvfile)
66                     writer.writerows(sell_file) # Write all rows back to the file
67
68                 # Write to trade history as well
69                 with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
70                     writer = csv.writer(csvfile)
71                     writer.writerow([order_id, 'Pending (Replaced Order)', 'sell', stock_name, new_price, new_quantity]) # Append new order
72
73             else:
74                 print(f"Sell Order {order_id} not found for stock {stock_name}.")
75
76
77     replace_order = ReplaceOrder() #this instantly calls on the class
78

```

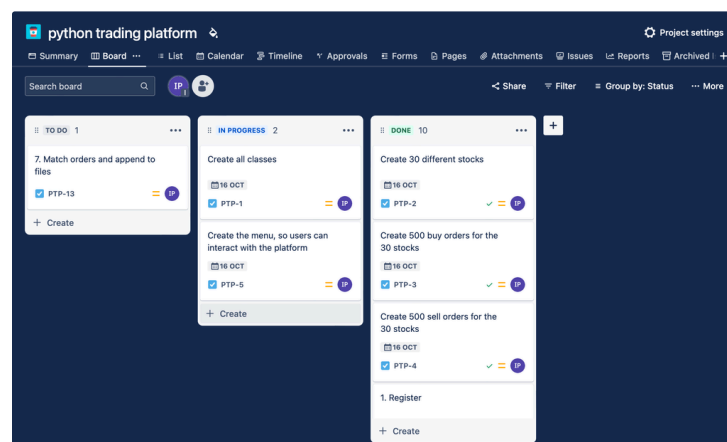
This was then linked into the menu, for a user to replace an order. This lets the user interact and input each criteria that is needed to append an order. The code below makes a user put in the order id, then the stock name, then the order type. This matches to the correct order and then the user updates the price of the stock and the quantity of the order. The order will only be appended if the order id and the stock name information is correct, this stops the wrong order accidentally being overwritten each time a order is appended.

```

1 elif option == 6:
2     print("Please enter your order details to replace an order")
3     order_id = int(input("Order ID to replace: "))
4     stock_name = input("Stock name to replace: ").upper()
5     order_type = input("Order type (buy or sell): ").lower()
6     new_price = float(input("New price: "))
7     new_quantity = int(input("New quantity: "))
8     status = "pending (replaced order)"
9     replace_order.replace_order(order_id, stock_name, order_type, new_price, new_quantity)
10

```

The below image shows my progress in JIRA, to show the manage orders has been complete and the final task was to match the buy and sell lists, so orders will update to a status of 'fully filled order'.



In the last stage of the project, the final class was created called match orders. The buy and sell list is read into python and added to a new list, one for buying stock and one for selling stock. The loop inside reading the file into the list, only reads in the pending and pending(appended order). (This means cancelled orders will be ignored.) Then a loop is made, which loops through the buy and sell list,

then matches the stock name in the buy list to the stock name in the sell list. If these match then the second condition matches the buy price to the sell price, if the buy price equals the sell price or the buy price is higher than the sell price, the stock is sold and the status is changed to 'fully filled order'. Another condition is linked to the quantity of the stock. This is done with the minimum function, so if the buy quantity is higher than the sell quantity, all the sell stock will be sold. The order will only be fully filled if the quantity is zero. This is then appended to the buy and sell list, with either 'pending' if the order is only partially filled and 'fully filled order' if the order is completed. This is then written back into the buy and sell files, the 'w' overrides the file, which means the more the match order is run, the smaller the list gets as fully filled orders and cancelled orders are removed from the file. However, these are all still available in the trade history, this means eventually when continually running when viewing the buy and sell orders only pending orders will appear.

```

1  class MatchOrders:
2      def match_order(self):
3          buy_list = []
4          sell_list = []
5
6          with open('buy_orders.csv', 'r') as file:
7              reader = csv.reader(file)
8              header = next(reader)
9              for row in reader:
10                 if row[1].strip() == 'pending' or row[1].strip() == 'Pending (Replaced Order)':
11                     buy_list.append(row) # Collect all pending buy orders
12
13          # Read pending sell orders
14          with open('sell_orders.csv', 'r') as file:
15              reader = csv.reader(file)
16              header = next(reader)
17              for row in reader:
18                 if row[1].strip() == 'pending' or row[1].strip() == 'Pending (Replaced Order)':
19                     sell_list.append(row) # Collect all pending sell orders
20
21          for buylist in buy_list:
22              for selllist in sell_list:
23                 if buylist[3] == selllist[3] and float(buylist[4]) >= float(selllist[4]):
24                     min_quantity = min(float(buylist[5]), float(selllist[5]))
25
26                     buylist[5] = str(float(buylist[5]) - min_quantity)
27                     selllist[5] = str(float(selllist[5]) - min_quantity)
28
29                     # Update statuses
30                     if float(buylist[5]) == 0:
31                         buylist[1] = "Fully filled order"
32                     if float(selllist[5]) == 0:
33                         selllist[1] = "Fully filled order"
34
35                     with open('buy_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
36                         writer = csv.writer(csvfile)
37                         writer.writerows(buy_list) # Write all rows back to the file
38
39                     with open('sell_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
40                         writer = csv.writer(csvfile)
41                         writer.writerows(sell_list)
42
43                     #Write to trade history as well
44                     with open('trade_history.csv', 'a', newline='') as csvfile:
45                         writer = csv.writer(csvfile)
46                         writer.writerow([buylist[0], buylist[1], buylist[2], buylist[3], buylist[4], buylist[5], selllist[0], selllist[1], selllist[2],
47                                         selllist[3], selllist[4], selllist[5]])
48
49          matcher = MatchOrders()

```

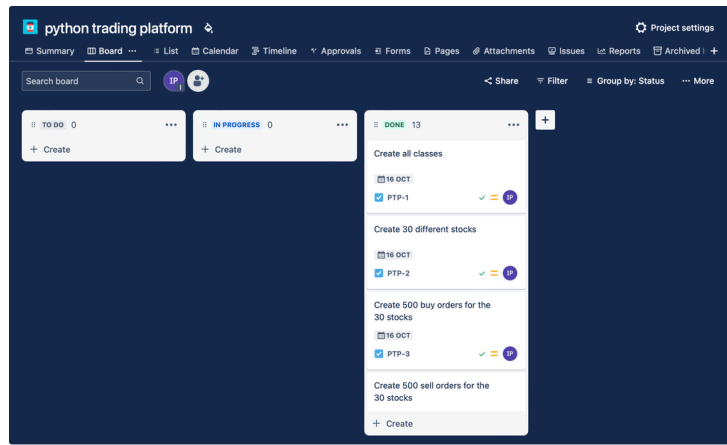
This was then linked to the menu, as option seven is to view the trade history. When a user views the trade history, the class of match orders is called, so everytime this option is typed in the matching or orders is triggered and the files change and update. When viewing the trade history, the trade history will show each individual stock and which order id it is matched to. So for example if a BMW buy stock had a price of 234 and a quantity of 4 and the BMW sell stock had a price of 200 and a quantity of 4, both will be matched, with a status of 'fully filled order', this will be added to the trade history as 4 new rows.

```

1  elif option == 7:
2      print("Viewing your trade history/ Viewing the status of all orders...")
3      matcher.match_order()
4      with open('trade_history.csv', 'r') as csvfile:
5          csv_reader = csv.reader(csvfile)
6          for line in csv_reader:
7              print(line)
8

```

The image below shows the trading platform has been complete in JIRA.



The below code is the python trading platform, which is shown above as one big chunk.

```

1 import csv
2 import random
3 import getpass
4 import os
5
6 base_dir = '/Users/izzy/Downloads/python_assessment'
7 User_File = os.path.join(base_dir, 'users.csv')
8 file1 = os.path.join(base_dir, 'buy_orders.csv')
9 file2 = os.path.join(base_dir, 'sell_orders.csv')
10 merged_file = os.path.join(base_dir, 'trade_history.csv')
11
12 class UserAuth:
13     def register_user(self, username, password):
14         # Store username and password
15         with open(User_File, 'a', newline='') as file:
16             writer = csv.writer(file)
17             writer.writerow([username, password]) # Store plain password
18
19         print(f"User {username} registered successfully.")
20
21     def login_user(self, username, password):
22         with open(User_File, 'r') as file:
23             reader = csv.reader(file)
24             for row in reader:
25                 if row[0] == username and row[1] == password:
26                     print(f"User {username} logged in successfully.")
27                     return True # Login successful
28             print("Login failed. Invalid credentials.")
29             return False # Login failed
30
31 user_info = UserAuth()
32
33 class Stock:
34     def __init__(self, name, price, quantity):
35         self.name = name
36         self.price = price
37         self.quantity = quantity
38
39     def stock_info(self):
40         return "Name : " + str(self.name) + " Stock Price : " + str(self.price) + " Stock Quantity : " + str(self.quantity)
41
42 class BuyStock(Stock):
43     def __init__(self, order_id, name, price, quantity):
44         super().__init__(name, price, quantity)
45         self.order_id = round(random.randint(0, 500000), 0)
46         self.action = 'buy'
47         self.status = 'pending'
48
49     def stock_info(self):
50         return "Order ID: " + str(self.order_id) + " Status: " + self.status + " Action: " + str(self.action) + super().stock_info()
51
52 class SellStock(Stock):
53     def __init__(self, order_id, name, price, quantity):
54         super().__init__(name, price, quantity)
55         self.order_id = round(random.randint(0, 500000), 0)
56         self.action = 'sell'
57         self.status = 'pending'
58
59     def sell_info(self):
60         return "Order ID: " + str(self.order_id) + " Status: " + self.status + " Action: " + str(self.action) + super().stock_info()

```

```

62
63 class OrderManager:
64     def add_order(self, order_type, stock_name, price, quantity):
65         if order_type == 'buy':
66             order_id = round(random.randint(0, 500000), 0) # Unique ID for the new order
67             new_order = BuyStock(order_id, stock_name, price, quantity)
68             print(f"Added Buy Order: {new_order.stock_info()}")
69             self.save_buy_order(new_order) # Save to CSV
70         elif order_type == 'sell':
71             order_id = round(random.randint(0, 500000), 0) # Unique ID for the new order
72             new_order = SellStock(order_id, stock_name, price, quantity)
73             print(f"Added Sell Order: {new_order.sell_info()}")
74             self.save_sell_order(new_order) # Save to CSV
75
76     def save_buy_order(self, order):
77
78         with open('buy_orders.csv', 'a', newline='') as csvfile: # Append mode
79             writer = csv.writer(csvfile)
80             writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity]) # Append new order
81
82         with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
83             writer = csv.writer(csvfile)
84             writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity]) # Append new order
85
86     def save_sell_order(self, order):
87         with open('sell_orders.csv', 'a', newline='') as csvfile: # Append mode
88             writer = csv.writer(csvfile)
89             writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity]) # Append new order
90
91         with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
92             writer = csv.writer(csvfile)
93             writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity]) # Append new order
94
95     def cancel_order(self, order_id, order_type, stock_name, price, quantity):
96         if order_type == 'buy':
97             self.cancel_buy_order(order_id, stock_name, price, quantity)
98         elif order_type == 'sell':
99             self.cancel_sell_order(order_id, stock_name, price, quantity)
100         else:
101             print("Error: Please select either 'buy' or 'sell'.")
102
103     def cancel_buy_order(self, order_id, stock_name, price, quantity):
104         buy_cancel_file = []
105         order_found = False
106
107         with open('buy_orders.csv', 'r') as file:
108             reader = csv.reader(file)
109             header = next(reader) # Read the header
110             buy_cancel_file.append(header) # Keep the header for rewriting later
111
112         for row in reader:
113             if row[0] == str(order_id) and row[3] == stock_name and row[4] == str(price) and row[5] == str(quantity):
114                 row[1] = 'Cancelled'
115                 print(f"Buy Order {order_id} for stock {stock_name} has been cancelled.")
116                 order_found = True # Mark that we found and updated the order
117
118             buy_cancel_file.append(row) # Add the updated row (or original if not updated)
119
120         if order_found:
121             # Write back to the file
122             with open('buy_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
123                 writer = csv.writer(csvfile)
124                 writer.writerows(buy_cancel_file) # Write all rows back to the file
125
126             # Write to trade history as well
127             with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
128                 writer = csv.writer(csvfile)
129                 writer.writerow([order_id, 'Cancelled', 'buy', stock_name]) # Append new order
130         else:
131             print("Error: 'buy_orders.csv' order in file not found.")
132
133     def cancel_sell_order(self, order_id, stock_name, price, quantity):
134         sell_cancel_file = []
135         order_found = False
136
137         with open('sell_orders.csv', 'r') as file:
138             reader = csv.reader(file)
139             header = next(reader) # Read the header
140             sell_cancel_file.append(header) # Keep the header for rewriting later
141
142         for row in reader:
143             if row[0] == str(order_id) and row[3] == stock_name and row[4] == str(price) and row[5] == str(quantity):
144                 row[1] = 'Cancelled'

```

```

145         print(f"Sell Order {order_id} for stock {stock_name} has been cancelled.")
146         order_found = True # Mark that we found and updated the order
147
148         sell_cancel_file.append(row) # Add the updated row (or original if not updated)
149
150     if order_found:
151         # Write back to the file
152         with open('sell_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
153             writer = csv.writer(csvfile)
154             writer.writerows(sell_cancel_file) # Write all rows back to the file
155
156         # Write to trade history as well
157         with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
158             writer = csv.writer(csvfile)
159             writer.writerow([order_id, 'Cancelled', 'sell', stock_name]) # Append new order
160     else:
161         print("Error: 'sell_orders.csv' order in file not found.")
162
163 order_manager = OrderManager()
164
165 class ReplaceOrder:
166     def replace_order(self, order_id, stock_name, order_type, new_price, new_quantity):
167         if order_type == 'buy':
168             self._replace_buy_order(order_id, stock_name, new_price, new_quantity)
169         elif order_type == 'sell':
170             self._replace_sell_order(order_id, stock_name, new_price, new_quantity)
171         else:
172             print("Error, please select either buy or sell.")
173
174     def _replace_buy_order(self, order_id, stock_name, new_price, new_quantity):
175         buy_file = []
176         order_found = False
177
178         with open('buy_orders.csv', 'r') as file:
179             reader = csv.reader(file)
180             header = next(reader) # Read the header
181             buy_file.append(header) # Keep the header for rewriting later
182
183             for row in reader:
184                 if row[0] == str(order_id) and row[3] == stock_name:
185                     row[4] = str(new_price) # Update the price
186                     row[5] = str(new_quantity) # Update the quantity
187                     row[1] = 'Pending (Replaced Order)'
188                     print(f"Buy Order {order_id} for stock {stock_name} has been replaced with new price: {new_price}, quantity: {new_quantity}, and status:
189 {row[2]}.")
190
191                     order_found = True # Mark that we found and updated the order
192
193                     buy_file.append(row) # Add the updated row (or original if not updated)
194
195             if order_found:
196                 # Write back to the file
197                 with open('buy_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
198                     writer = csv.writer(csvfile)
199                     writer.writerows(buy_file) # Write all rows back to the file
200
201                 # Write to trade history as well
202                 with open('trade_history.csv', 'a', newline='') as csvfile: # This adds a new row so all the history of users is stored
203                     writer = csv.writer(csvfile)
204                     writer.writerow([order_id, 'Pending (Replaced Order)', 'buy', stock_name, new_price, new_quantity]) # Append new order
205             else:
206                 print(f"Buy Order {order_id} not found for stock {stock_name}.")
207
208     def _replace_sell_order(self, order_id, stock_name, new_price, new_quantity):
209         sell_file = []
210         order_found = False
211
212         with open('sell_orders.csv', 'r') as file:
213             reader = csv.reader(file)
214             header = next(reader) # Read the header
215             sell_file.append(header) # Keep the header for rewriting later
216
217             for row in reader:
218                 if row[0] == str(order_id) and row[3] == stock_name:
219                     row[4] = str(new_price) # Update the price
220                     row[5] = str(new_quantity) # Update the quantity
221                     row[1] = 'Pending (Replaced Order)'
222                     print(f"Sell Order {order_id} for stock {stock_name} has been replaced with new price: {new_price}, quantity: {new_quantity}, and status:
223 {row[2]}.")
224
225                     order_found = True # Mark that we found and updated the order
226
227                     sell_file.append(row) # Add the updated row (or original if not updated)

```

```

226     if order_found:
227         # Write back to the file
228         with open('sell_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
229             writer = csv.writer(csvfile)
230             writer.writerows(sell_file) # Write all rows back to the file
231
232         # Write to trade history as well
233         with open('trade_history.csv', 'a', newline='') as csvfile: # Append mode
234             writer = csv.writer(csvfile)
235             writer.writerow([order_id, 'Pending (Replaced Order)', 'sell', stock_name, new_price, new_quantity]) # Append new order
236
237     else:
238         print(f"Sell Order {order_id} not found for stock {stock_name}.")
239
240
241 replace_order = ReplaceOrder()
242
243 class MatchOrders:
244     def match_order(self):
245         buy_list = []
246         sell_list = []
247
248         with open('buy_orders.csv', 'r') as file:
249             reader = csv.reader(file)
250             header = next(reader)
251             for row in reader:
252                 if row[1].strip() == 'pending' or row[1].strip() == 'Pending (Replaced Order)':
253                     buy_list.append(row) # Collect all pending buy orders
254
255         # Read pending sell orders
256         with open('sell_orders.csv', 'r') as file:
257             reader = csv.reader(file)
258             header = next(reader)
259             for row in reader:
260                 if row[1].strip() == 'pending' or row[1].strip() == 'Pending (Replaced Order)':
261                     sell_list.append(row) # Collect all pending sell orders
262
263         for buylist in buy_list:
264             for selllist in sell_list:
265                 if buylist[3] == selllist[3] and float(buylist[4]) >= float(selllist[4]):
266                     min_quantity = min(float(buylist[5]), float(selllist[5]))
267
268                     buylist[5] = str(float(buylist[5]) - min_quantity)
269                     selllist[5] = str(float(selllist[5]) - min_quantity)
270
271                 # Update statuses
272                 if float(buylist[5]) == 0:
273                     buylist[1] = "Fully filled order"
274                 if float(selllist[5]) == 0:
275                     selllist[1] = "Fully filled order"
276
277                 with open('buy_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
278                     writer = csv.writer(csvfile)
279                     writer.writerows(buy_list) # Write all rows back to the file
280
281                 with open('sell_orders.csv', 'w', newline='') as csvfile: # Overwrite mode
282                     writer = csv.writer(csvfile)
283                     writer.writerows(sell_list)
284
285                 # Write to trade history as well
286                 with open('trade_history.csv', 'a', newline='') as csvfile:
287                     writer = csv.writer(csvfile)
288                     writer.writerow([buylist[0], buylist[1], buylist[2], buylist[3], buylist[4], buylist[5], selllist[0], selllist[1], selllist[2],
289                                     selllist[3], selllist[4], selllist[5]])
290
291 matcher = MatchOrders()
292
293 # Predefined list of stocks
294 stock_names = ['AAPL', 'GOOG', 'AMZN', 'MSFT', 'TSLA', 'NFLX', 'FB', 'NVDA', 'BABA', 'DIS', 'JPM', 'V', 'PG', 'BMW', 'MERC', 'PEP', 'COST', 'STBX', 'HD', 'JNJ',
295                'UNI', 'KFC', 'ESYJ', 'BA', 'INTC', 'CSCO', 'ADBE', 'PYPL', 'MCD', 'ORCL']
296
297 stock_list = []
298
299 for i in range(30):
300     name = stock_names[i]
301     price = round(random.uniform(0, 500), 2) # Random price between 0 and 500
302     quantity = round(random.uniform(0, 50), 0) # Random quantity between 0 and 50
303     stock_list.append(Stock(name, price, quantity)) # Append stock to the list
304
305 # Display Stock Info
306 # print("Stock List:")
307 # for stock in stock_list:
308     # print(stock.stock_info())

```

```

307
308 # Display Buy and Sell Info
309 #print("Buy List:")
310 #for buy_stock in buy_list:
311     #print(buy_stock.stock_info())
312
313 #print("Sell List:")
314 #for sell_stock in sell_list:
315     #print(sell_stock.sell_info())
316
317
318 file1 = 'buy_orders.csv'
319 file2 = 'sell_orders.csv'
320 merged_file = 'trade_history.csv'
321 buy_list = []
322 sell_list = []
323
324 # Check if the CSV file buy_already exists
325 if os.path.exists(file1):
326     print("CSV file already exists. Skipping data input...")
327 else:
328     print("Generating new stock data and writing to CSV...")
329
330     for stock in stock_list:
331         for i in range(10): # Generate 10 buy and 10 sell orders for each stock
332             buy_price = round(random.uniform(0, 500), 2)
333             buy_quantity = round(random.uniform(0, 100), 0)
334             buy_list.append(BuyStock(i+1, stock.name, buy_price, buy_quantity)) # Add buy order with unique ID
335         with open('buy_orders.csv', 'w', newline='') as csvfile:
336             write = csv.writer(csvfile)
337             write.writerow(['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price', 'Buy Quantity']) # Wrote a header for each column
338         for buy_stock in buy_list:
339             write.writerow([buy_stock.order_id, buy_stock.status, buy_stock.action, buy_stock.name, buy_stock.price, buy_stock.quantity])
340
341 # Store sell_list in a CSV file
342 if os.path.exists(file2):
343     print("CSV file already exists. Skipping data input...")
344 else:
345     print("Generating new stock data and writing to CSV...")
346     for stock in stock_list:
347         for i in range(10):
348             sell_price = round(random.uniform(0, 500), 2)
349             sell_quantity = round(random.uniform(0, 100), 0)
350             sell_list.append(SellStock(i+1, stock.name, sell_price, sell_quantity)) # Add sell order with unique ID
351         with open('sell_orders.csv', 'w', newline='') as csvfile:
352             write = csv.writer(csvfile)
353             write.writerow(['Order ID', 'Status', 'Action', 'Stock Name', 'Sell Price', 'Sell Quantity'])
354         for sell_stock in sell_list:
355             write.writerow([sell_stock.order_id, sell_stock.status, sell_stock.action, sell_stock.name, sell_stock.price, sell_stock.quantity])
356
357 #creating a trade_history file
358 if os.path.exists(merged_file):
359     print("CSV file already exists. Skipping data input...")
360 else:
361     print("Generating new stock data and writing to CSV...")
362
363     with open(file1, 'r') as f1:
364         reader1 = csv.reader(f1)
365         header1 = next(reader1)
366         data1 = list(reader1)
367
368     # Read data from second file
369     with open(file2, 'r') as f2:
370         reader2 = csv.reader(f2)
371         header2 = next(reader2)
372         data2 = list(reader2)
373
374     # List of CSV file names to merge
375     file_names = ['buy_orders.csv', 'sell_orders.csv']
376
377     # Output file name
378     trade_history = 'trade_history.csv'
379
380     with open(trade_history, 'w', newline='') as f_out:
381         writer = csv.writer(f_out)
382         writer.writerow(['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price', 'Buy Quantity'] + ['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price',
383         'Buy Quantity']) # Merge headers
384         for row1, row2 in zip(data1, data2):
385             writer.writerow(row1 + row2) # Merge rows
386
387 print("Buy and Sell lists have been written to trade_history.csv.")
388
389 def menu():
390     print("Izzy's Electronic Trading platform:")

```

```

389 print("1) Register ")
390 print("2) Login")
391 print("3) View buy and sell orders for all stocks")
392 print("4) Add a new order (buy/sell)")
393 print("5) Cancel an order")
394 print("6) Replace an order")
395 print("7) View the status of all orders/ View your trade history")
396 print("8) Logout")
397
398 logged_in = False
399
400 while True:
401     menu()
402     option = int(input("Please enter your choice from the menu: "))
403
404     if option == 8:
405         print("Logged out. Thank you for using Izzy's Electronic Trading platform.")
406         logged_in = False
407         break
408
409     elif option == 1:
410         print(" Please create a username and password ")
411
412         username = input(" Please create a username : ")
413
414         password = getpass.getpass("Please create a password: ")
415
416         user_info.register_user(username, password)
417
418         print(f"User {username} registered successfully.")
419
420         print("( Username and password created please login. )")
421
422     elif option == 2:
423         print(" Please enter your username and password ")
424         username = input("Please enter your username: ")
425         password = getpass.getpass("Please enter your password: ")
426
427         success = user_info.login_user(username, password) # Only call it once
428
429         if success:
430             logged_in = True # Set login state to True
431             print(f"Welcome back, {username}!")
432         else:
433             print("Login attempt failed.")
434
435     elif logged_in:
436         if option == 3:
437             print("Choose to view either buy or sell orders for all stocks.")
438
439             buy_or_sell = input(" Input choice : ").lower()
440
441             if buy_or_sell == 'buy':
442                 with open('buy_orders.csv', 'r') as csvfile:
443                     csv_reader = csv.reader(csvfile)
444                     for line in csv_reader:
445                         print(line)
446
447             elif buy_or_sell == 'sell':
448                 with open('sell_orders.csv', 'r') as csvfile:
449                     csv_reader = csv.reader(csvfile)
450                     for line in csv_reader:
451                         print(line)
452             else:
453                 print("Invalid choice. Please select buy or sell.")
454
455         elif option == 4:
456             manage_order = input(" Please enter if you would like to place a buy or sell order : ").lower()
457             if manage_order == 'buy':
458                 stock_name = input("Stock name: ").upper()
459                 price = float(input("Price: "))
460                 quantity = int(input("Quantity: "))
461                 status = 'pending'
462                 order_manager.add_order(manage_order, stock_name, price, quantity)
463                 print(" Order added ")
464
465             if manage_order == 'sell':
466                 stock_name = input("Stock name: ").upper()
467                 while stock_name not in stock_list:
468                     print("Invalid stock name. Please enter a valid stock name from the list:")
469                     break
470                 price = float(input("Price: "))

```

```

472         quantity = int(input("Quantity: "))
473         status = 'pending'
474         order_manager.add_order(manage_order, stock_name, price, quantity)
475         print(" Order added ")
476
477     else:
478         print("Invalid choice. Please select buy or sell.")
479
480     elif option == 5:
481         print("Please enter your order details to cancel an order")
482         order_id = int(input("Order ID to cancel: "))
483         stock_name = input("Stock name to cancel: ").upper()
484         order_type = input("Order type (buy/sell): ").lower()
485         price = float(input("Order price: "))
486         quantity = int(input("Order quantity: "))
487         status = 'cancelled'
488         order_manager.cancel_order(order_id, order_type, stock_name, price, quantity)
489         print("Please enter your order details to cancel an order:")
490
491     elif option == 6:
492         print("Please enter your order details to replace an order")
493         order_id = int(input("Order ID to replace: "))
494         stock_name = input("Stock name to replace: ").upper()
495         order_type = input("Order type (buy or sell): ").lower()
496         new_price = float(input("New price: "))
497         new_quantity = int(input("New quantity: "))
498         status = "pending (replaced order)"
499         replace_order.replace_order(order_id, stock_name, order_type, new_price, new_quantity)
500
501     elif option == 7:
502         print("Viewing your trade history/ Viewing the status of all orders...")
503         matcher.match_order()
504         with open('trade_history.csv', 'r') as csvfile:
505             csv_reader = csv.reader(csvfile)
506             for line in csv_reader:
507                 print(line)
508
509     else:
510         print(" Please log in or register to access options. ")
511

```