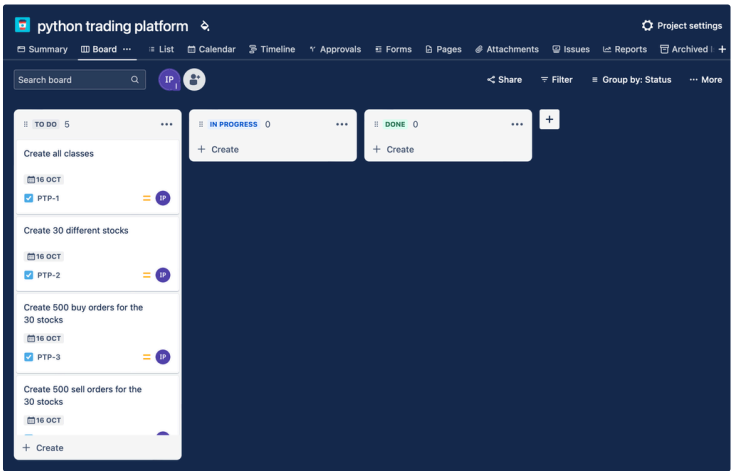
Python script for an electronic trading platform

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.

```
def menu():#This will print so user can see what the options are
    print("Izzy's Electronic Trading platform:")
    print("1) Register ")
    print("2) Login")
    print("3) View buy and sell orders for all stocks")
    print("4) Add a new order (buy/sell)")
    print("5) Cancel an order")
    print("6) Replace an order")
    print("7) View the status of all orders/ View your trade history")
    print("8) Logout")

while True:
    menu()
    option = int(input("Please enter your choice from the menu: "))

    if option == 8:
        pass

    elif option == 1:
        pass

    elif option == 2:
        pass

    elif option == 3:
        pass

    elif option == 4:
        pass

    elif option == 5:
        pass

    elif option == 6:
        pass

    elif option == 7:
        pass

    else:
        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.

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

```
        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.)

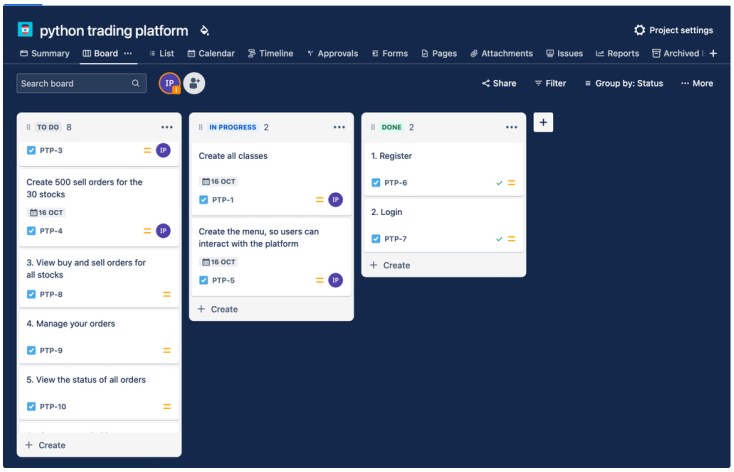
```
import csv
import getpass
import sys

base_dir = '/Users/izzy/Downloads/python_assessment'
User_File = os.path.join(base_dir, 'users.csv')
file1 = os.path.join(base_dir, 'buy_orders.csv')
file2 = os.path.join(base_dir, 'sell_orders.csv')
merged_file = os.path.join(base_dir, 'trade_history.csv')

class UserAuth:
    def register_user(self, username, password):
        # Store username and password
        with open(User_File, 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([username, password])  # Store plain password

        print(f"User {username} registered successfully.")

    def login_user(self, username, password):
        with open(User_File, 'r') as file:
            reader = csv.reader(file)
            for row in reader:
                if row[0] == username and row[1] == password:
                    print(f"User {username} logged in successfully.")
                    return True  # Login successful
        print("Login failed. Invalid credentials.")
        return False  # Login failed

user_info = UserAuth()
#this is an instance, so the class can be called on in menu

```

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.

```
logged_in = False

while True:
    menu()
    option = int(input("Please enter your choice from the menu: "))

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

    elif option == 1:
        print(" Please create a username and password ")

        username = input(" Please create a username : ")

        password = getpass.getpass("Please create a password: ")

        user_info.register_user(username, password)
        #this line of code links the information back to the correct class and also the correct function inside the class.

        print(f"User {username} registered successfully.")

        print("( Username and password created please login. )")

    elif option == 2:
        print(" Please enter your username and password ")
        username = input("Please enter your username: ")
        password = getpass.getpass("Please enter your password: ")
        success = user_info.login_user(username, password)
        #calls to read file and see if the login information exists.

        if success:
            logged_in = True  # Set login state to True(lets you use the rest of the menu.)
            print(f"Welcome back, {username}!")
        else:
            print("Login attempt failed.")

```

```
    elif logged_in:
        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.

```
class Stock:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def stock_info(self):
        return "Name :" +str(self.name)+ " Stock Price : " +str(self.price)+ " Stock Quantity : "+ str(self.quantity)


class BuyStock(Stock):
    def __init__(self, order_id, name, price, quantity):
        super().__init__(name, price, quantity)
        self.order_id = len(buy_list) + 1
        self.action = ' buy '
        self.status = ' pending '

    def stock_info(self):
        return " Order ID: " +str(self.order_id)+" Status: " + self.status + " Action: " +str(self.action) + super().stock_info()

class SellStock(Stock):
    def __init__(self, order_id, name, price, quantity):
        super().__init__(name, price, quantity)
        self.order_id = len(sell_list) + 1
        self.action = ' sell '
        self.status = ' pending '

    def sell_info(self):
        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.

```
#Predefined list of stocks
stock_names = ['AAPL', 'GOOG', 'AMZN', 'MSFT', 'TSLA', 'NFLX', 'FB', 'NVDA', 'BABA', 'DIS', 'JPM', 'V', 'PG', 'BMW', 'MERC', 'PEP', 'COST', 'STBX', 'HD', 'JNJ',
'UNI', 'KFC', 'ESYJ', 'BA', 'INTC', 'CSCO', 'ADBE', 'PYPL', 'MCD', 'ORCL']

stock_list = []

for i in range(30):
    name = stock_names[i]
    price = round(random.uniform(0, 500), 2)  # Random price between 0 and 500
    quantity = round(random.uniform(0, 50), 0)  # Random quantity between 0 and 50
    stock_list.append(Stock(name, price, quantity))  # Append stock to the list

# Display Stock Information
print("Stock List:")
  for stock in stock_list:p
    print(stock.stock_info())
```

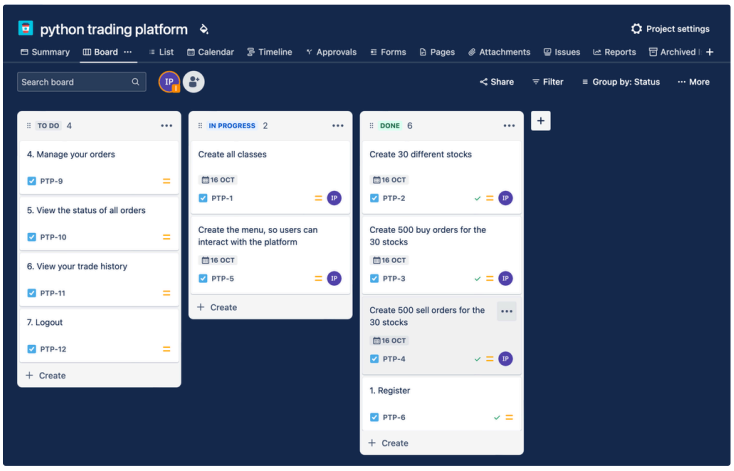
```
#Display Buy Information
print("Buy List:")
  for buy_stock in buy_list:
    print(buy_stock.stock_info())

#Display Sell Information
print("Sell List:")
for sell_stock in sell_list:
  print(sell_stock.sell_info())

#need to define, so the files can be created if they do not exist.
file1 = 'buy_orders.csv'
file2 = 'sell_orders.csv'
merged_file = 'merged_file.csv'
buy_list = []
sell_list = []

# Check if the CSV file buy_list exists
if os.path.exists(file1):
    print("CSV file already exists. Skipping data input...")
else:
    print("Generating new stock data and writing to CSV...")

    for stock in stock_list:
        for i in range(10):  # Generate 10 buy and 10 sell orders for each stock
            buy_price = round(random.uniform(0, 500), 2)
            buy_quantity = round(random.uniform(0, 100), 0)
            buy_list.append(BuyStock(i+1, stock.name, buy_price, buy_quantity))  # Add buy order with unique ID
    with open('buy_orders.csv', 'w', newline='') as csvfile:
        write = csv.writer(csvfile)
        write.writerow(['Order ID','Status', 'Action', 'Stock Name', 'Buy Price', 'Buy Quantity'])  # Wrote a header for each column
        for buy_stock in buy_list:
            write.writerow([buy_stock.order_id,buy_stock.status, buy_stock.action, buy_stock.name, buy_stock.price, buy_stock.quantity])

# Check if the CSV file sell_list exists
if os.path.exists(file2):
    print("CSV file already exists. Skipping data input...")
else:
    print("Generating new stock data and writing to CSV...")
    for stock in stock_list:
        for i in range(10):
            sell_price = round(random.uniform(0, 500), 2)
            sell_quantity = round(random.uniform(0, 100), 0)
            sell_list.append(SellStock(i+1, stock.name, sell_price, sell_quantity))  # Add sell order with unique ID
    with open('sell_orders.csv', 'w', newline='') as csvfile:
        write = csv.writer(csvfile)
        write.writerow(['Order ID', 'Status', 'Action', 'Stock Name', 'Sell Price', 'Sell Quantity'])
        for sell_stock in sell_list:
            write.writerow([sell_stock.order_id, sell_stock.status, sell_stock.action, sell_stock.name, sell_stock.price, sell_stock.quantity])

#creating a trade_history file
if os.path.exists(merged_file):
    print("CSV file already exists. Skipping data input...")
else:
    print("Generating new stock data and writing to CSV...")

    with open(file1, 'r') as f1:
        reader1 = csv.reader(f1)
        header1 = next(reader1)
        data1 = list(reader1)

    # Read data from second file
    with open(file2, 'r') as f2:
        reader2 = csv.reader(f2)
        header2 = next(reader2)
        data2 = list(reader2)
    # List of CSV file names to merge
    file_names = ['buy_orders.csv', 'sell_orders.csv']

    # Output file name
    trade_history = 'trade_history.csv'

    with open(trade_history, 'w', newline='') as f_out:
        writer = csv.writer(f_out)
        writer.writerow(['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price', 'Buy Quantity'] + ['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price',
'Buy Quantity'])  # Merge headers
        for row1, row2 in zip(data1, data2):
            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.

```
if option == 3:
```

```
            print("Choose to view either buy or sell orders for all stocks.")

            buy_or_sell =  input(" Input choice : ").lower() #ensures the input option is in lower case to match to the class

            if buy_or_sell == 'buy':
                with open('buy_orders.csv', 'r') as csvfile:
                    csv_reader = csv.reader(csvfile)
                    for line in csv_reader:
                        print(line)


            elif buy_or_sell == 'sell':
                with open('sell_orders.csv', 'r') as csvfile:
                    csv_reader = csv.reader(csvfile)
                    for line in csv_reader:
                        print(line)
            else:
                print("Invalid choice. Please select buy or sell.")

```

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.)

```
class OrderManager:
    def add_order(self, order_type, stock_name, price, quantity):
        if order_type == 'buy':
            order_id = round(random.randint(0, 500000), 0) # Unique ID for the new order
            new_order = BuyStock(order_id, stock_name, price, quantity)
            print(f"Added Buy Order: {new_order.stock_info()}")
            self.save_buy_order(new_order)  # Save to CSV
        elif order_type == 'sell':
            order_id = round(random.randint(0, 500000), 0)  # Unique ID for the new order
            new_order = SellStock(order_id, stock_name, price, quantity)
            print(f"Added Sell Order: {new_order.sell_info()}")
            self.save_sell_order(new_order)  # Save to CSV

    def save_buy_order(self, order):

        with open('buy_orders.csv', 'a', newline='') as csvfile:  # Append mode
            writer = csv.writer(csvfile)
            writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity])  # Append new order

        with open('trade_history.csv', 'a', newline='') as csvfile:  # Append mode
            writer = csv.writer(csvfile)
            writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity])  # Append new order

    def save_sell_order(self, order):
        with open('sell_orders.csv', 'a', newline='') as csvfile:  # Append mode
            writer = csv.writer(csvfile)
            writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity])  # Append new order

        with open('trade_history.csv', 'a', newline='') as csvfile:  # Append mode
            writer = csv.writer(csvfile)
            writer.writerow([order.order_id, order.status, order.action.strip(), order.name, order.price, order.quantity])  # Append new order

order_manager = OrderManager() #this is the instance for the order manager class, this calls on the class when the class is needed.

```

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.

```
elif option == 4:
            manage_order = input(" Please enter if you would like to place a buy or sell order : ").lower()
            if manage_order == 'buy':
                stock_name = input("Stock name: ").upper() #the upper saves the information as uppercases, so it matches the format of the created stock
                price = float(input("Price: ")) #this allows a user to input a price which includes decimals
                quantity = int(input("Quantity: ")) #this only allows users to input a whole number, if the user puts in an integer python will say error
                status = 'pending'
                order_manager.add_order(manage_order, stock_name, price, quantity)
                print(" Order added ")

            if manage_order == 'sell':
                stock_name = input("Stock name: ").upper()
                while stock_name not in stock_list:
                    print("Invalid stock name. Please enter a valid stock name from the list:")
                    break
                price = float(input("Price: "))
                quantity = int(input("Quantity: "))
                status = 'pending'
                order_manager.add_order(manage_order, stock_name, price, quantity)
                print(" Order added ")

            else:
                print("Invalid choice. Please select buy or sell.")

```

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 appened 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.)

```
class OrderManager:
    def cancel_order(self, order_id, order_type, stock_name, price, quantity):
        if order_type == 'buy':
            self.cancel_buy_order(order_id, stock_name, price, quantity) #saves to buy csv file
        elif order_type == 'sell':
            self.cancel_sell_order(order_id, stock_name, price, quantity) #saves to sell csv file
        else:
            print("Error: Please select either 'buy' or 'sell'.")

    def cancel_buy_order(self, order_id, stock_name, price, quantity):
        buy_cancel_file = []
        order_found = False

        with open('buy_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)  # Read the header
            buy_cancel_file.append(header)  # Keep the header for rewriting later and puts the file to the list above

            for row in reader: #loops through each row in the file and match to the conditions below
                if row[0] == str(order_id) and row[3] == stock_name and row[4] == str(price) and row[5] == str(quantity):
                    row[1] = 'Cancelled' #this rewrites the status of the order to be cancelled
                    print(f"Buy Order {order_id} for stock {stock_name} has been cancelled.")
                    order_found = True  # Mark that we found and updated the order

                buy_cancel_file.append(row)  # Add the updated row (or original if not updated)

        if order_found:
            # Write back to the file
            with open('buy_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                writer = csv.writer(csvfile)
                writer.writerows(buy_cancel_file)  # Write all rows back to the file

                # Write to trade history as well
            with open('trade_history.csv', 'a', newline='') as csvfile:  # Append mode
                writer = csv.writer(csvfile)
                writer.writerow([order_id, 'Cancelled', 'buy', stock_name])  # Append new order
        else:
            print("Error: 'buy_orders.csv' order in file not found.")

    def cancel_sell_order(self, order_id, stock_name, price, quantity):
        sell_cancel_file = []
        order_found = False

        with open('sell_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)  # Read the header
            sell_cancel_file.append(header)  # Keep the header for rewriting later

            for row in reader:
```

```
                if row[0] == str(order_id) and row[3] == stock_name and row[4] == str(price) and row[5] == str(quantity):
                    row[1] = 'Cancelled'
                    print(f"Sell Order {order_id} for stock {stock_name} has been cancelled.")
                    order_found = True  # Mark that we found and updated the order

                sell_cancel_file.append(row)  # Add the updated row (or original if not updated)

        if order_found:
        # Write back to the file
            with open('sell_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                writer = csv.writer(csvfile)
                writer.writerows(sell_cancel_file)  # Write all rows back to the file

                # Write to trade history as well
            with open('trade_history.csv', 'a', newline='') as csvfile:  # Append mode
                writer = csv.writer(csvfile)
                writer.writerow([order_id, 'Cancelled', 'sell', stock_name])  # Append new order
        else:
            print("Error: 'sell_orders.csv' order in file not found.")

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.

```
elif option == 5:
            print("Please enter your order details to cancel an order")
            order_id = int(input("Order ID to cancel: "))
            stock_name = input("Stock name to cancel: ").upper()
            order_type = input("Order type (buy/sell): ").lower()
            price = float(input("Order price: "))
            quantity = int(input("Order quantity: "))
            status = 'cancelled'
            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
            print("Order successfully cancelled.")

```

The last section of managing an order, was to append an already existing order. I put this into a seperate 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 quanitity 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.

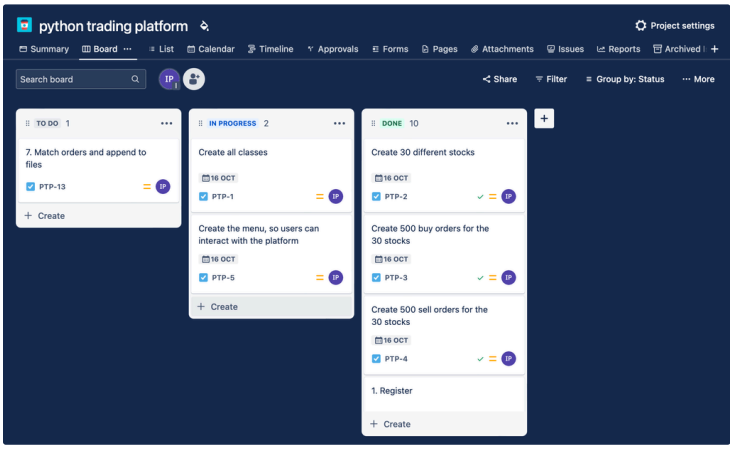
```
class ReplaceOrder:
    def replace_order(self, order_id, stock_name, order_type, new_price, new_quantity):
        if order_type == 'buy':
            self._replace_buy_order(order_id, stock_name, new_price, new_quantity)
        elif order_type == 'sell':
            self._replace_sell_order(order_id, stock_name, new_price, new_quantity)
        else:
            print("Error, please select either buy or sell.")

    def _replace_buy_order(self, order_id, stock_name, new_price, new_quantity):
        buy_file = []
        order_found = False

        with open('buy_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)  # Read the header
            buy_file.append(header)  # Keep the header for rewriting later

            for row in reader:
                if row[0] == str(order_id) and row[3] == stock_name:
                    row[4] = str(new_price)  # Update the price
                    row[5] = str(new_quantity)  # Update the quantity
                    row[1] = 'Pending (Replaced Order)'
                    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]}.")
                    order_found = True  # Mark that we found and updated the order

                buy_file.append(row)  # Add the updated row (or original if not updated)

        if order_found:
            # Write back to the file
            with open('buy_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                writer = csv.writer(csvfile)
                writer.writerows(buy_file)  # Write all rows back to the file

            # Write to trade history as well
            with open('trade_history.csv', 'a', newline='') as csvfile:  # This adds a new row so all the hsitory of users is stored
                writer = csv.writer(csvfile)
```

```
                writer.writerow([order_id, 'Pending (Replaced Order)', 'buy', stock_name, new_price, new_quantity])  # Append new order

        else:
            print(f"Buy Order {order_id} not found for stock {stock_name}.")

    def _replace_sell_order(self, order_id, stock_name, new_price, new_quantity):
        sell_file = []
        order_found = False

        with open('sell_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)  # Read the header
            sell_file.append(header)  # Keep the header for rewriting later

            for row in reader:
                if row[0] == str(order_id) and row[3] == stock_name:
                    row[4] = str(new_price)  # Update the price
                    row[5] = str(new_quantity)  # Update the quantity
                    row[1] = 'Pending (Replaced Order)'
                    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]}.")
                    order_found = True  # Mark that we found and updated the order

                sell_file.append(row)  # Add the updated row (or original if not updated)

        if order_found:
            # Write back to the file
            with open('sell_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                writer = csv.writer(csvfile)
                writer.writerows(sell_file)  # Write all rows back to the file

            # Write to trade history as well
            with open('trade_history.csv', 'a', newline='') as csvfile:  # Append mode
                writer = csv.writer(csvfile)
                writer.writerow([order_id, 'Pending (Replaced Order)', 'sell', stock_name, new_price, new_quantity])  # Append new order

        else:
            print(f"Sell Order {order_id} not found for stock {stock_name}.")


replace_order = ReplaceOrder() #this instant calls on the class

```

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 appened 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.

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

```

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,

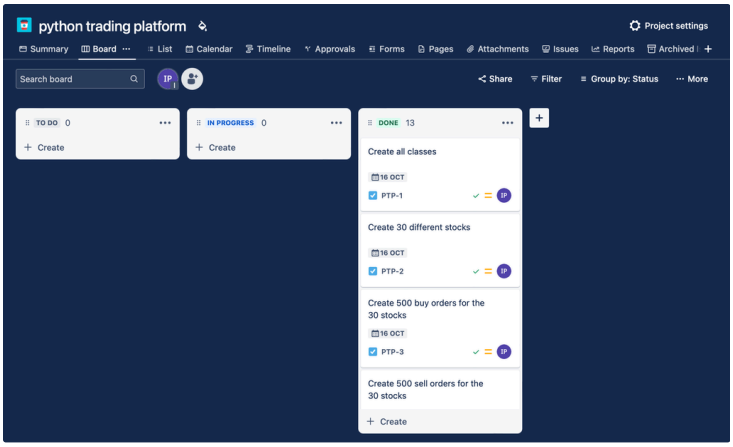
then matches the stock name in the buy list to the stock name in the sell list. If these match then the second condititon matches the buy price to the sell price, if the buy price equals the sell price or the buy price is higher then the sell price, the stock is sold and the status is changed to ‘fully 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 avaliable in the trade history, this means eventually when continually running when viewing the buy and sell orders only pending orders will appear.

```
class MatchOrders:
    def match_order(self):
        buy_list = []
        sell_list = []

        with open('buy_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)
            for row in reader:
                if row[1].strip() == 'pending' or row[1].strip() == 'Pending (Replaced Order)':
                    buy_list.append(row)  # Collect all pending buy orders

        # Read pending sell orders
        with open('sell_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)
            for row in reader:
                if row[1].strip() == 'pending' or row[1].strip() == 'Pending (Replaced Order)':
                    sell_list.append(row)  # Collect all pending sell orders

        for buylist in buy_list:
            for selllist in sell_list:
                if buylist[3] == selllist[3] and float(buylist[4]) >= float(selllist[4]):
                    min_quantity = min(float(buylist[5]),float(selllist[5]))

                    buylist[5] = str(float(buylist[5]) - min_quantity)
                    selllist[5] = str(float(selllist[5]) - min_quantity)

                    # Update statuses
                    if float(buylist[5]) == 0:
                        buylist[1] = "Fully filled order"
                    if float(selllist[5]) == 0:
                        selllist[1] = "Fully filled order"

                    with open('buy_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                        writer = csv.writer(csvfile)
                        writer.writerows(buy_list)  # Write all rows back to the file

                    with open('sell_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                       writer = csv.writer(csvfile)
                       writer.writerows(sell_list)

                    #Write to trade history as well
                    with open('trade_history.csv', 'a', newline='') as csvfile:
                        writer = csv.writer(csvfile)
                        writer.writerow([buylist[0], buylist[1], buylist[2], buylist[3], buylist[4], buylist[5], selllist[0], selllist[1], selllist[2],
selllist[3], selllist[4], selllist[5]])

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.

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

```

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.

```python
import csv
import random
import getpass
import os

base_dir = '/Users/izzy/Downloads/python_assessment'
User_File = os.path.join(base_dir, 'users.csv')
file1 = os.path.join(base_dir, 'buy_orders.csv')
file2 = os.path.join(base_dir, 'sell_orders.csv')
merged_file = os.path.join(base_dir, 'trade_history.csv')

class UserAuth:
    def register_user(self, username, password):
        # Store username and password
        with open(User_File, 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow([username, password])  # Store plain password

        print(f"User {username} registered successfully.")

    def login_user(self, username, password):
        with open(User_File, 'r') as file:
            reader = csv.reader(file)
            for row in reader:
                if row[0] == username and row[1] == password:
                    print(f"User {username} logged in successfully.")
                    return True  # Login successful
        print("Login failed. Invalid credentials.")
        return False  # Login failed

user_info = UserAuth()

class Stock:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def stock_info(self):
        return "Name :" +str(self.name)+ " Stock Price : " +str(self.price)+ " Stock Quantity : "+ str(self.quantity)


class BuyStock(Stock):
    def __init__(self, order_id, name, price, quantity):
        super().__init__(name, price, quantity)
        self.order_id = round(random.randint(0, 500000), 0)
        self.action = ' buy '
        self.status = ' pending '

    def stock_info(self):
        return " Order ID: " +str(self.order_id)+" Status: " + self.status + " Action: " +str(self.action) + super().stock_info()

class SellStock(Stock):
    def __init__(self, order_id, name, price, quantity):
        super().__init__(name, price, quantity)
        self.order_id = round(random.randint(0, 500000), 0)
        self.action = ' sell '
        self.status = ' pending '

    def sell_info(self):
        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'
```

```
                    print(f"Sell Order {order_id} for stock {stock_name} has been cancelled.")
                    order_found = True  # Mark that we found and updated the order

                sell_cancel_file.append(row)  # Add the updated row (or original if not updated)

        if order_found:
        # Write back to the file
            with open('sell_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                writer = csv.writer(csvfile)
                writer.writerows(sell_cancel_file)  # Write all rows back to the file

                # Write to trade history as well
            with open('trade_history.csv', 'a', newline='') as csvfile:  # Append mode
                writer = csv.writer(csvfile)
                writer.writerow([order_id, 'Cancelled', 'sell', stock_name])  # Append new order
        else:
            print("Error: 'sell_orders.csv' order in file not found.")

order_manager = OrderManager()

class ReplaceOrder:
    def replace_order(self, order_id, stock_name, order_type, new_price, new_quantity):
        if order_type == 'buy':
            self._replace_buy_order(order_id, stock_name, new_price, new_quantity)
        elif order_type == 'sell':
            self._replace_sell_order(order_id, stock_name, new_price, new_quantity)
        else:
            print("Error, please select either buy or sell.")

    def _replace_buy_order(self, order_id, stock_name, new_price, new_quantity):
        buy_file = []
        order_found = False

        with open('buy_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)  # Read the header
            buy_file.append(header)  # Keep the header for rewriting later

            for row in reader:
                if row[0] == str(order_id) and row[3] == stock_name:
                    row[4] = str(new_price)  # Update the price
                    row[5] = str(new_quantity)  # Update the quantity
                    row[1] = 'Pending (Replaced Order)'
                    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]}.")
                    order_found = True  # Mark that we found and updated the order

                buy_file.append(row)  # Add the updated row (or original if not updated)

        if order_found:
            # Write back to the file
            with open('buy_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                writer = csv.writer(csvfile)
                writer.writerows(buy_file)  # Write all rows back to the file

            # Write to trade history as well
            with open('trade_history.csv', 'a', newline='') as csvfile:  # This adds a new row so all the hsitory of users is stored
                writer = csv.writer(csvfile)
                writer.writerow([order_id, 'Pending (Replaced Order)', 'buy', stock_name, new_price, new_quantity])  # Append new order

        else:
            print(f"Buy Order {order_id} not found for stock {stock_name}.")

    def _replace_sell_order(self, order_id, stock_name, new_price, new_quantity):
        sell_file = []
        order_found = False

        with open('sell_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)  # Read the header
            sell_file.append(header)  # Keep the header for rewriting later

            for row in reader:
                if row[0] == str(order_id) and row[3] == stock_name:
                    row[4] = str(new_price)  # Update the price
                    row[5] = str(new_quantity)  # Update the quantity
                    row[1] = 'Pending (Replaced Order)'
                    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]}.")
                    order_found = True  # Mark that we found and updated the order

                sell_file.append(row)  # Add the updated row (or original if not updated)

```

```
        if order_found:
            # Write back to the file
            with open('sell_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                writer = csv.writer(csvfile)
                writer.writerows(sell_file)  # Write all rows back to the file

            # Write to trade history as well
            with open('trade_history.csv', 'a', newline='') as csvfile:  # Append mode
                writer = csv.writer(csvfile)
                writer.writerow([order_id, 'Pending (Replaced Order)', 'sell', stock_name, new_price, new_quantity])  # Append new order

        else:
            print(f"Sell Order {order_id} not found for stock {stock_name}.")


replace_order = ReplaceOrder()

class MatchOrders:
    def match_order(self):
        buy_list = []
        sell_list = []

        with open('buy_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)
            for row in reader:
                if row[1].strip() == 'pending' or row[1].strip() == 'Pending (Replaced Order)':
                    buy_list.append(row)  # Collect all pending buy orders

        # Read pending sell orders
        with open('sell_orders.csv', 'r') as file:
            reader = csv.reader(file)
            header = next(reader)
            for row in reader:
                if row[1].strip() == 'pending' or row[1].strip() == 'Pending (Replaced Order)':
                    sell_list.append(row)  # Collect all pending sell orders

        for buylist in buy_list:
            for selllist in sell_list:
                if buylist[3] == selllist[3] and float(buylist[4]) >= float(selllist[4]):
                    min_quantity = min(float(buylist[5]),float(selllist[5]))

                    buylist[5] = str(float(buylist[5]) - min_quantity)
                    selllist[5] = str(float(selllist[5]) - min_quantity)

                    # Update statuses
                    if float(buylist[5]) == 0:
                        buylist[1] = "Fully filled order"
                    if float(selllist[5]) == 0:
                        selllist[1] = "Fully filled order"

                    with open('buy_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                        writer = csv.writer(csvfile)
                        writer.writerows(buy_list)  # Write all rows back to the file

                    with open('sell_orders.csv', 'w', newline='') as csvfile:  # Overwrite mode
                       writer = csv.writer(csvfile)
                       writer.writerows(sell_list)

                    #Write to trade history as well
                    with open('trade_history.csv', 'a', newline='') as csvfile:
                        writer = csv.writer(csvfile)
                        writer.writerow([buylist[0], buylist[1], buylist[2], buylist[3], buylist[4], buylist[5], selllist[0], selllist[1], selllist[2],
selllist[3], selllist[4], selllist[5]])

matcher = MatchOrders()

#Predefined list of stocks
stock_names = ['AAPL', 'GOOG', 'AMZN', 'MSFT', 'TSLA', 'NFLX', 'FB', 'NVDA', 'BABA', 'DIS', 'JPM', 'V', 'PG', 'BMW', 'MERC', 'PEP', 'COST', 'STBX', 'HD', 'JNJ',
'UNI', 'KFC', 'ESYJ', 'BA', 'INTC', 'CSCO', 'ADBE', 'PYPL', 'MCD', 'ORCL']

stock_list = []

for i in range(30):
    name = stock_names[i]
    price = round(random.uniform(0, 500), 2)  # Random price between 0 and 500
    quantity = round(random.uniform(0, 50), 0)  # Random quantity between 0 and 50
    stock_list.append(Stock(name, price, quantity))  # Append stock to the list

# Display Stock Info
#print("Stock List:")
#for stock in stock_list:
    #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     # List of CSV file names to merge
374     file_names = ['buy_orders.csv', 'sell_orders.csv']
375
376     # Output file name
377     trade_history = 'trade_history.csv'
378
379     with open(trade_history, 'w', newline='') as f_out:
380         writer = csv.writer(f_out)
381         writer.writerow(['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price', 'Buy Quantity'] + ['Order ID', 'Status', 'Action', 'Stock Name', 'Buy Price',
    'Buy Quantity'])  # Merge headers
382         for row1, row2 in zip(data1, data2):
383             writer.writerow(row1 + row2)  # Merge rows
384
385 print("Buy and Sell lists have been written to trade_history.csv.")
386
387 def menu():
388     print("Izzy's Electronic Trading platform:")
```

```
    print("1) Register ")
    print("2) Login")
    print("3) View buy and sell orders for all stocks")
    print("4) Add a new order (buy/sell)")
    print("5) Cancel an order")
    print("6) Replace an order")
    print("7) View the status of all orders/ View your trade history")
    print("8) Logout")

logged_in = False

while True:
    menu()
    option = int(input("Please enter your choice from the menu: "))

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

    elif option == 1:
        print(" Please create a username and password ")

        username = input(" Please create a username : ")

        password = getpass.getpass("Please create a password: ")

        user_info.register_user(username, password)

        print(f"User {username} registered successfully.")

        print("( Username and password created please login. )")

    elif option == 2:
        print(" Please enter your username and password ")
        username = input("Please enter your username: ")
        password = getpass.getpass("Please enter your password: ")

        success = user_info.login_user(username, password)  # Only call it once

        if success:
            logged_in = True  # Set login state to True
            print(f"Welcome back, {username}!")
        else:
            print("Login attempt failed.")

    elif logged_in:
        if option == 3:
            print("Choose to view either buy or sell orders for all stocks.")

            buy_or_sell =  input(" Input choice : ").lower()

            if buy_or_sell == 'buy':
                with open('buy_orders.csv', 'r') as csvfile:
                    csv_reader = csv.reader(csvfile)
                    for line in csv_reader:
                        print(line)


            elif buy_or_sell == 'sell':
                with open('sell_orders.csv', 'r') as csvfile:
                    csv_reader = csv.reader(csvfile)
                    for line in csv_reader:
                        print(line)
            else:
                print("Invalid choice. Please select buy or sell.")

        elif option == 4:
            manage_order = input(" Please enter if you would like to place a buy or sell order : ").lower()
            if manage_order == 'buy':
                stock_name = input("Stock name: ").upper()
                price = float(input("Price: "))
                quantity = int(input("Quantity: "))
                status = 'pending'
                order_manager.add_order(manage_order, stock_name, price, quantity)
                print(" Order added ")

            if manage_order == 'sell':
                stock_name = input("Stock name: ").upper()
                while stock_name not in stock_list:
                    print("Invalid stock name. Please enter a valid stock name from the list:")
                    break
                price = float(input("Price: "))
```

```
                quantity = int(input("Quantity: "))
                status = 'pending'
                order_manager.add_order(manage_order, stock_name, price, quantity)
                print(" Order added ")

            else:
                print("Invalid choice. Please select buy or sell.")

        elif option == 5:
            print("Please enter your order details to cancel an order")
            order_id = int(input("Order ID to cancel: "))
            stock_name = input("Stock name to cancel: ").upper()
            order_type = input("Order type (buy/sell): ").lower()
            price = float(input("Order price: "))
            quantity = int(input("Order quantity: "))
            status = 'cancelled'
            order_manager.cancel_order(order_id, order_type, stock_name, price, quantity)
            print("Please enter your order details to cancel an order:")

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

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

    else:
        print(" Please log in or register to access options. ")

```