Programming Languages Assignment5

2021312738 소프트웨어학과 김서환

1. 초기화면

파이썬 프로그램 코드를 실행하면 실행한 순간부터 백그라운드 스레드가 실행되게 된다.
3개의 선택지가 주어지는데 1을 입력하면 사용자를 등록하는 register()함수를 실행하고, 2를 입력하면 등록되어있는 사용자 정보를 기반으로 login()함수를 실행하고, 3을 입력하면 “Goodbye!”를 출력하며 프로그램이 종료되게 된다. 그 외의 값을 입력하면 “Invalid selection. Try again.”을 출력하면서 재입력을 사용자에게 요구한다.

```
def main():
    threading.Thread(target=backgroundthreading, daemon=True).start()

    while True:
        print("=== Stock Trading Simulation ===")
        print("1) Register  2) Login  3) Exit")
        choice = input("Select: ")

        if choice == '1':
            register()
        elif choice == '2':
            user = login()
            if user:
                stock(user)
        elif choice == '3':
            print("Goodbye!")
            break
        else:
            print("Invalid selection. Try again.")

if __name__ == "__main__":
    main()
```

또한, users.json, market.json, transactions.json 이 3개의 파일을 읽거나 수정하는 작업을 도와주는 함수들을 따로 정의해뒀다. 함수들의 내용은 다음과 같다.

```
def marketread():
    try:
        with open("market.json", 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        return {}

def marketwrite(market):
    with open("market.json", 'w') as file:
        json.dump(market, file, indent = 4)

def userread():
    try:
        with open("users.json", 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        return {}

def userwrite(user):
    with open("users.json", 'w') as file:
        json.dump(user, file, indent = 4)

def transactionread():
    try:
        with open("transactions.json", 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        return {}

def transactionwrite(transaction):
    with open("transactions.json", 'w') as file:
        json.dump(transaction, file, indent = 4)
```

read함수에서 .json 파일이 초기에 없다면 빈 배열을 반환하도록 하여 내용(데이터)을 작성한

후 write함수안에 json.dump메서드(indent는 들여쓰기의 정도를 의미함)를 통해 데이터를 매개변수로 받아와 파일이 없다면 파일을 생성하고 파일이 존재한다면 기존 파일의 내용을 덮어씌우도록 구현했다.

1-1. Register

```python
def register():
    userinfo = userread()

    while True:
        username = input("Username: ")
        if username in userinfo:
            print("Username already exists. Please choose another.")
        else:
            break

    while True:
        password = input("Password: ")
        error = []
        if len(password) < 8:
            error.append("- Password must be at least 8 characters.")

        uppercaseexist = False
        for c in password:
            if c.isupper():
                uppercaseexist = True
                break
        if not uppercaseexist:
            error.append("- Password must include at least one uppercase letter.")

        specialstring = "!@#$%^&*()"
        specialcharexist = False
        for c in password:
            if c in specialstring:
                specialcharexist = True
                break
        if not specialcharexist:
            error.append("- Password must include at least one special character (!@#$%^&*()).")

        if error:
            for e in error:
                print(e)
        else:
            break
```

```python
while True:
    print("Select strategy:")
    print(" 1) RandomStrategy")
    print(" 2) MovingAverageStrategy")
    print(" 3) MomentumStrategy")
    choice = input("Choose (1/2/3): ")

    if choice == '1':
        strategy = 'RandomStrategy'
        break
    elif choice == '2':
        strategy = 'MovingAverageStrategy'
        break
    elif choice == '3':
        strategy = 'MomentumStrategy'
        break
    else:
        print("Invalid selection. Try again.")

userinfo[username] = {
    "password": password,
    "strategy": strategy,
    "balance": 10000.0,
    "auto": False,
    "portfolio": {}
}

userwrite(userinfo)
print(f"User '{username}' registered successfully.\n")
```

userinfo를 통해서 users.json의 파일을 읽어오고, input으로 username을 받고 이미 존재한다면 계속 다시 입력받도록 했고, Password를 입력받은 후에 password의 3개의 조건인 At least 8 characters, At least one uppercase letter, At least one special character from !@#$%^&*()를 위반하면 error에 추가하여 조건을 확인한 후에 error가 존재하면 존재하는 error를 모두 출력하도록 구현했다. error가 없다면 strategy를 선택하는 메뉴가 출력되도록 했고 strategy를 선택하면 userinfo에 username을 key로 지정하여 password, strategy, balance, auto, portfolio에 해당하는 값을 value로 입력하고 userwrite(userinfo)를 통해서 해당 user 정보를 users.json에 생성해줬다. register의 실행 결과는 다음과 같다.

```
=== Stock Trading Simulation ===
1) Register  2) Login  3) Exit
Select: 1
Username: seo
Password: seo12
- Password must be at least 8 characters.
- Password must include at least one uppercase letter.
- Password must include at least one special character (!@#$%^&*()).
Password: seohwan12
- Password must include at least one uppercase letter.
- Password must include at least one special character (!@#$%^&*()).
Password: Seohwan12
- Password must include at least one special character (!@#$%^&*()).
Password: Seo1234@
Select strategy:
 1) RandomStrategy
 2) MovingAverageStrategy
 3) MomentumStrategy
Choose (1/2/3): 1
User 'seo' registered successfully.
```

1-2. Login

```
def login():
    userinfo = userread()
    username = input("Username: ")
    password = input("Password: ")
    if (username not in userinfo) or (password != userinfo[username]["password"]):
        print("Invalid username or password.\n")
        return ""
    else:
        print(f"Welcom, {username}!\n")
        return username
```

로그인은 userinfo를 통해 데이터를 읽어와 username이 존재하지 않거나 password가 맞지 않다면 메인메뉴로 돌아가도록 했고, 제대로 입력했다면 해당 username을 반환하도록 구현했다. login의 실행 결과는 다음과 같다.

```
=== Stock Trading Simulation ===
1) Register  2) Login  3) Exit
Select: 2
Username: kim
Password: Kim1234@
Invalid username or password.

=== Stock Trading Simulation ===
1) Register  2) Login  3) Exit
Select: 2
Username: seo
Password: Seo1233@
Invalid username or password.

=== Stock Trading Simulation ===
1) Register  2) Login  3) Exit
Select: 2
Username: seo
Password: Seo1234@
Welcom, seo!
```

1-3. Exit

위의 메인 메뉴에서 알 수 있듯이 choice == '3'일 경우 "Goodbye!"를 출력하며 프로그램이 종료되게 된다. Exit의 실행 결과는 다음과 같다.

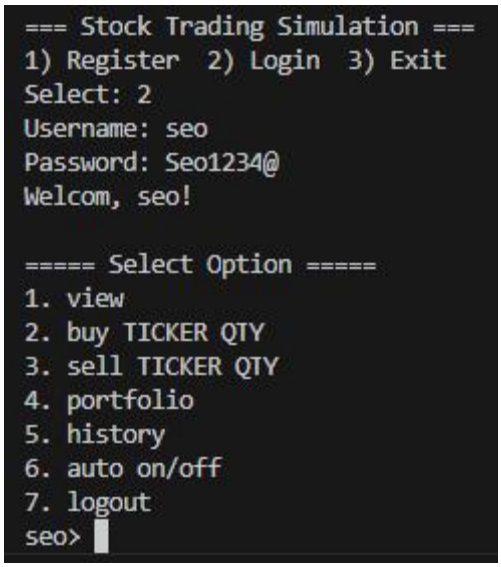
```
=== Stock Trading Simulation ===
1) Register  2) Login  3) Exit
Select: 3
Goodbye!
```

2. Main screen

login()을 통해서 받아온 username을 user변수에 저장하고 user변수가 비어있는 문자열이 아니라면(로그인 실패의 경우) stock(user)로 stock함수를 실행한다.

```python
def stock(user):
    while True:
        print("===== Select Option =====")
        print("1. view\n2. buy TICKER QTY\n3. sell TICKER QTY\n4. portfolio\n5. history\n6. auto on/off\n7. logout")
        choice = input(f"{user}> ")

        if choice == '1':
            view()
        elif choice == '2':
            buyticker(user)
        elif choice == '3':
            sellticker(user)
        elif choice == '4':
            portfolio(user)
        elif choice == '5':
            history(user)
        elif choice == '6':
            autoonoff(user)
        elif choice == '7':
            return
        else:
            print("Invalid Option. Try again.")
```

아래의 사진이 로그인 성공시 나오는 메인 메뉴 스크린이다.



2-1. View

```python
def view():
    market = marketread()
    last5price = {}
    last5volume = {}
    for ticker in market:
        last5price[ticker] = []
        last5volume[ticker] = []
        for i in range(min(4, len(market[ticker]["prices"])-1), -1, -1):
            price = market[ticker]["prices"][-i-1]
            volume = market[ticker]["volumes"][-i-1]
            last5price[ticker].append(price)
            last5volume[ticker].append(volume)
    for ticker in market:
        currentprice = market[ticker]["prices"][-1]
        currentvolume = market[ticker]["volumes"][-1]
        print(f"\n[{ticker}] ${currentprice} Vol:{currentvolume}")
        print(f"Last {len(last5price[ticker])} prices: {last5price[ticker]}")
        print(f"Last {len(last5volume[ticker])} volumes: {last5volume[ticker]}")
    print("\n")
```

1을 입력하면 view함수가 실행이 되는데, view는 3개의 ticker 각각에 대해 최근 5개의 가격,

최근 5개의 거래량, 현재 가격, 현재 거래량에 대해 출력하는 함수이다. view함수의 실행결과는 다음과 같다.

```
===== Select Option =====
1. view
2. buy TICKER QTY
3. sell TICKER QTY
4. portfolio
5. history
6. auto on/off
7. logout
seo> 1

[AAPL] $101.23 Vol:336645
Last 5 prices: [94.58, 96.76, 98.77, 101.68, 101.23]
Last 5 volumes: [448166, 488171, 410486, 40110, 336645]

[TSLA] $234.48 Vol:386503
Last 5 prices: [244.47, 248.2, 237.15, 236.47, 234.48]
Last 5 volumes: [193731, 61807, 338430, 484445, 386503]

[GOOG] $420.02 Vol:446807
Last 5 prices: [454.62, 432.14, 427.72, 440.42, 420.02]
Last 5 volumes: [169063, 40616, 235522, 222884, 446807]
```

2-2. Buyticker

```python
def buyticker(user):
    userinfo = userread()
    market = marketread()
    transactioninfo = transactionread()
    count = 0
    print("\n===== Buy Menu =====\n")
    print(f"Available Cash: ${round(userinfo[user]["balance"],2)}")
    print(f"Your Holdings: ")

    if not userinfo[user]["portfolio"]:
        userinfo[user]["portfolio"] = {"AAPL":[], "TSLA": [] , "GOOG":[]}
    for ticker in ["AAPL", "TSLA", "GOOG"]:
        if userinfo[user]["portfolio"][ticker]:
            portfolio = userinfo[user]["portfolio"][ticker]
            stocklist = []
            totalquantity = 0
            totalprice = 0
            for item in portfolio:
                if item["volume"] > 0:
                    stocklist.append({"price": item["price"], "volume":item["volume"]})
                else:
                    soldquantity = item["volume"]*(-1)
                    while soldquantity > 0 and stocklist:
                        if stocklist[0]["volume"] > soldquantity:
                            stocklist[0]["volume"] -= soldquantity
                            soldquantity = 0
                        else:
                            soldquantity -= stocklist[0]["volume"]
                            stocklist.pop(0)
            for item in stocklist:
                totalquantity += item["volume"]
                totalprice += (item["volume"] * item["price"])
            if totalquantity == 0:
                count += 1
                continue
            avgprice = round(totalprice/totalquantity,2)
            print(f" {ticker}: {totalquantity} shares @ avg ${avgprice}")
        else:
            count += 1
            continue

    if count == 3:
        print("  (No stocks owned)")
```

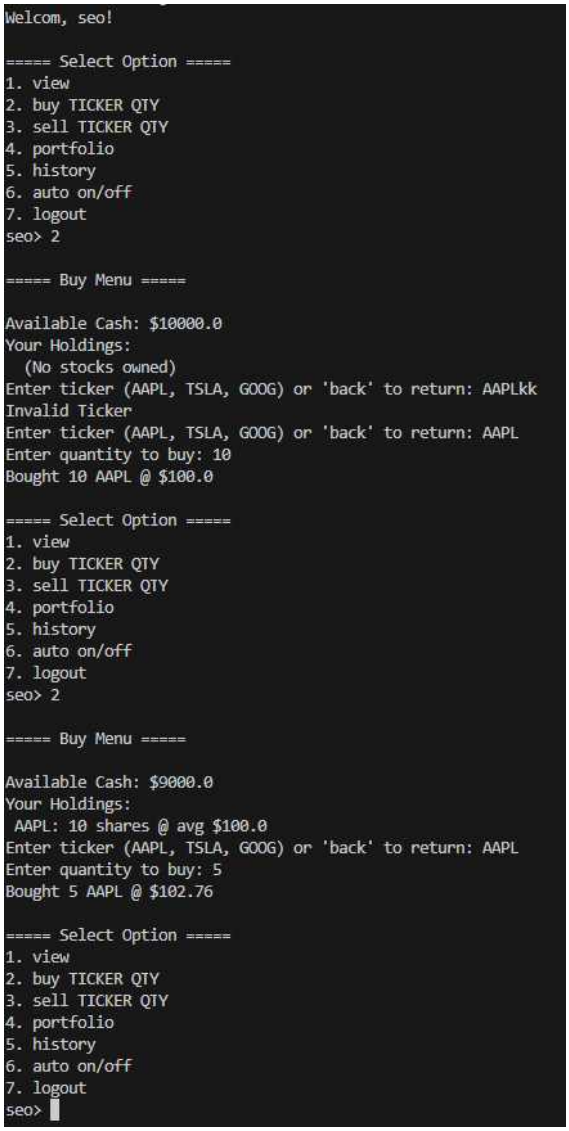
2를 입력하면 buyticker(user)함수가 실행이 되는데, userinfo 변수를 통해서 users.json의 데이터를 읽어오고, market 변수를 통해서 market.json의 데이터를 읽어오고, transactioninfo 변수를 통해서 transactions.json의 데이터를 읽어온다.

그 후 Buy Menu 출력문과 user 매개변수로 받아온 user의 이름을 key로 불러와 balance 현재 계좌에 있는 현금을 출력한다. 만일 해당 유저(user)가 거래한 이력(portfolio)가 없다면 빈 리스트로 각 ticker마다 값을 추가할 리스트를 생성해준다.
그 후 각 ticker 별로 해당 user가 보유한 주식 현황을 보여준다. 보여주는 정보는 해당 ticker의 거래량과 평균 매입가를 보여주며, 이를 위해 평균 매입가를 구해준다. 평균 매입가는 선입선출법으로 계산해주었으며, item["volume"] > 0의 조건을 통해서 매입한 거래 이력만 따로 stocklist에 저장하여 매입하여 보유한 주식의 리스트를 생성했다. item["volume"] < 0인 item(매도)의 거래량을 매입한 거래량에서 선입선출법으로 빼서 계산했다. 그 후 매입량 - 매도량을 계산한 값이 양수라면 남아있는 보유 주식이 있기 때문에 해당 주식들에 대해 총 수량, 총 가격을 구하여 평균 매입가를 계산했다.
item["volume"]은 Buy의 경우 양수, Sell의 경우 음수로 저장하여 totalquantity가 item["volume"]을 전부 더했을 때 0이라면 보유한 주식이 0이기 때문에 해당 ticker의 주식은 보여줄 필요가 없으므로 count 변수를 1만큼 증가시키고 다음 ticker에 대한 반복문을 돌 수 있도록 continue로 구현했다. count 변수가 3이라면 3개의 ticker에 대해 전부 보유 수량이 없다는 말이므로 "(No stocks owned)"을 출력하도록 했다.

```
buywantticker = input("Enter ticker (AAPL, TSLA, GOOG) or 'back' to return: ")
while buywantticker not in ["AAPL", "TSLA", "GOOG"]:
    if buywantticker == 'back':
        return
    print("Invalid Ticker")
    buywantticker = input("Enter ticker (AAPL, TSLA, GOOG) or 'back' to return: ")


buywantquantity = int(input("Enter quantity to buy: "))
while buywantquantity <= 0:
    print("Invalid Quantity")
    buywantquantity = int(input("Enter quantity to buy: "))

currentprice = market[buywantticker]["prices"][-1]
cost = currentprice * buywantquantity

if userinfo[user]["balance"] < cost:
    print("Insufficient balance.")
    return

userinfo[user]["balance"] -= cost

userinfo[user]["portfolio"][buywantticker].append({
    "price": currentprice,
    "volume": buywantquantity
})

print(f"Bought {buywantquantity} {buywantticker} @ ${currentprice}\n")

userwrite(userinfo)

currenttime = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

if not transactioninfo:
    transactioninfo[user] = []
else:
    if user not in transactioninfo:
        transactioninfo[user] = []
transaction = f"{currenttime} - BUY {buywantquantity} {buywantticker} @${currentprice}"
transactioninfo[user].append(transaction)

transactionwrite(transactioninfo)
```

그 후 buywantticker변수를 통해서 구매하길 원하는 ticker를 입력받아줬고, back을 입력하면 메인 메뉴로 리턴하게 되고, AAPL, TSLA, GOOG를 제외한 나머지 입력값들은 "Invalid

Ticker”를 출력하도록 구현했다. 그리고 구매하길 원하는 수량을 입력받아 0 이하의 값은 “Invalid Quantity”를 출력하도록 구현했고, 구매하길 원하는 ticker의 현재 금액을 기준으로 거래량과 곱한 값을 cost변수에 저장하여 해당 유저가 보유한 현금과 비교하여 구매할 수 없다면 “Insufficient balance.”를 출력하고 메인 메뉴로 반환했다. 구매할 수 있다면 유저의 balance에서 cost를 빼고 유저의 거래 이력에 가격과 거래량을 추가하여 userwrite를 통해서 userinfo를 users.json에 덮어씌워줬다. 또한, 현재 시간을 currenttime 변수로 받아서 transactions.json에서 해당 user의 정보가 아무것도 없다면 거래 내역을 입력해주기 위해 빈 리스트를 만들고, Transaction history에 Time of transaction, Action: BUY or SELL, Quantity, Stock ticker, Trade price을 담아야하기 때문에 “{currenttime} - BUY {buywantquantity} {buywantticker} @${currentprice}”의 내용의 문자열을 생성하여 transactions.json 파일에 데이터를 저장했다. Buyticker 함수의 실행 결과는 왼쪽 사진과 같다. 또한, transactions.json 파일의 저장 내용은 오른쪽 사진과 같다.



2-3. Selltickerr

```
    print("  (No stocks owned)")

sellwantticker = input("Enter ticker (AAPL, TSLA, GOOG) or 'back' to return: ")
if sellwantticker == 'back':
    return
totalquantityflag = 0
for item in userinfo[user]["portfolio"][sellwantticker]:
   availabletotalquantity += item["volume"]
if availabletotalquantity == 0:
    totalquantityflag = 1
else:
    totalquantityflag = 0

while sellwantticker not in ["AAPL", "TSLA", "GOOG"] or (sellwantticker in ["AAPL", "TSLA", "GOOG"] and totalquantityflag == 1):
    availabletotalquantity = 0
    if sellwantticker == 'back':
        return
    elif sellwantticker not in ["AAPL", "TSLA", "GOOG"]:
        print("Invalid Ticker")
        sellwantticker = input("Enter ticker (AAPL, TSLA, GOOG) or 'back' to return: ")
        continue
    else:
        for item in userinfo[user]["portfolio"][sellwantticker]:
            availabletotalquantity += item["volume"]
        if availabletotalquantity == 0:
            totalquantityflag = 1
            print("The holdings of this ticker are 0")
            sellwantticker = input("Enter ticker (AAPL, TSLA, GOOG) or 'back' to return: ")
        else:
            totalquantityflag = 0

sellwantquantity = int(input("Enter quantity to sell: "))
availabletotalquantity = 0
for item in userinfo[user]["portfolio"][sellwantticker]:
    availabletotalquantity += item["volume"]

while sellwantquantity <= 0 or availabletotalquantity < sellwantquantity:
    print("Invalid Quantity")
    sellwantquantity = int(input("Enter quantity to sell: "))

currentprice = market[sellwantticker]["prices"][-1]
proceeds = currentprice * sellwantquantity

userinfo[user]["balance"] += proceeds

userinfo[user]["portfolio"][sellwantticker].append({
    "price": currentprice,
    "volume": (-1)*sellwantquantity
})

print(f"Sold {sellwantquantity} {sellwantticker} @ ${currentprice}\n")

userwrite(userinfo)

currenttime = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

if not transactioninfo:
    transactioninfo[user] = []
else:
    if user not in transactioninfo:
        transactioninfo[user] = []
transaction = f"{currenttime} - SELL {sellwantquantity} {sellwantticker} @${currentprice}"
transactioninfo[user].append(transaction)

transactionwrite(transactioninfo)
```

3을 입력하면 Selltickerr함수가 실행되는데 이 함수는 Buyticker함수와 판매하길 원하는 ticker를 입력하는 부분 전까지는 내용이 똑같다. 판매하길 원하는 ticker를 입력받으면 availabletotalquantity 변수에 현재 user가 보유한 주식의 수량(매입량 - 매도량)을 계산하여 할당해줬고 만약 이 값이 0이라면 flag를 1로, 0이 아니라면 flag를 0으로 설정해줬다. 그 후 만약 입력받은 ticker가 "AAPL", "TSLA", "GOOG"가 아니거나 입력받은 ticker가 "AAPL", "TSLA", "GOOG"이지만, flag가 1(보유 주식 수량이 0)이라면 while문을 돌면서 유저가 다시 ticker를 입력하도록 했다. 다만, back을 입력할 경우 메인 메뉴로 돌아간다.

그 후 판매하길 원하는 수량을 입력받고 만약 해당 수량이 0 이하거나 내가 보유한 주식 수

량보다 크다면 while문을 돌면서 valid한 수량을 입력받을 수 있도록 구현했다. valid한 ticker와 valid한 수량을 입력받았다면 proceeds를 계산하여 판매 이익을 설정했고 유저의 현금 계좌에 판매 이익만큼을 더하고 거래 이력(portfolio)에 거래 내역을 더해줬다. 이때, 판매하길 원하는 수량에 -1을 곱해서 보유 주식 수량을 계산을 용이하게 만들어줬다. 또한, 현재 시간을 받아 Buyticker와 비슷한 방식으로 transaction 변수에 SELL로 지정하여 transactions.json 파일에 데이터를 저장해줬다. Sellticker 함수의 실행 결과는 왼쪽 사진과 같다. 또한, transactions.json 파일의 저장 내용은 오른쪽 사진과 같다.

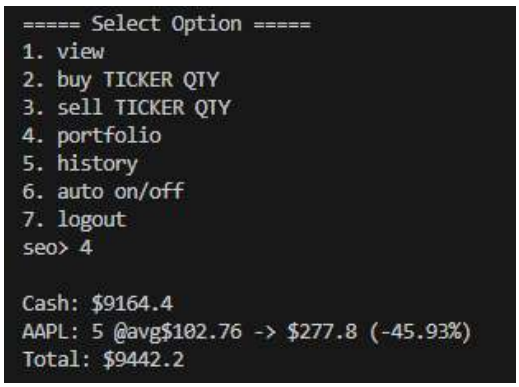
```
===== Select Option =====
1. view
2. buy TICKER QTY
3. sell TICKER QTY
4. portfolio
5. history
6. auto on/off
7. logout
seo> 3

===== Sell Menu =====


Available Cash: $8486.2
Your Holdings:
 AAPL: 15 shares @ avg $100.92
Enter ticker (AAPL, TSLA, GOOG) or 'back' to return: TSLA
The holdings of this ticker are 0
Enter ticker (AAPL, TSLA, GOOG) or 'back' to return: TSLAA
Invalid Ticker
Enter ticker (AAPL, TSLA, GOOG) or 'back' to return: AAPL
Enter quantity to sell: 20
Invalid Quantity
Enter quantity to sell: 10
Sold 10 AAPL @ $67.82
```

```
{
    "seo": [
        "2025-05-21 13:04:48 - BUY 10 AAPL @$100.0",
        "2025-05-21 13:05:03 - BUY 5 AAPL @$102.76",
        "2025-05-21 13:18:26 - SELL 10 AAPL @$67.82"
    ]
}
```

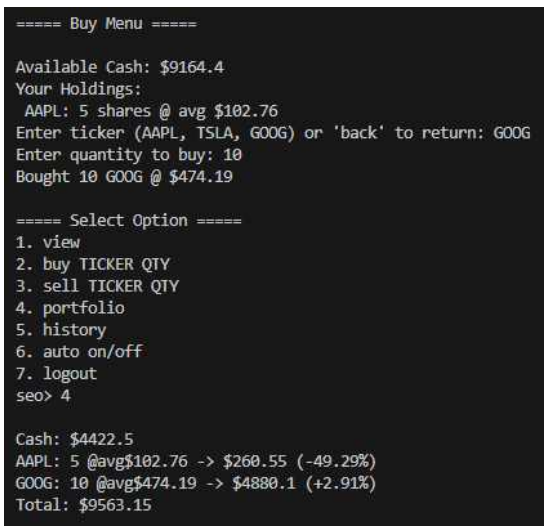
2-4. Portfolio

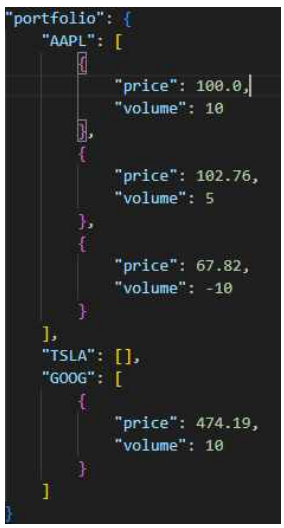
```
def portfolio(user):
    userinfo = userread()
    market = marketread()
    totalaccountvalue = 0
    if not userinfo[user]["portfolio"]:
        userinfo[user]["portfolio"] = {"AAPL":[], "TSLA": [] , "GOOG":[]}

    print(f"\nCash: ${round(userinfo[user]["balance"],2)}")
    totalaccountvalue += userinfo[user]['balance']

    for ticker in ["AAPL", "TSLA", "GOOG"]:
        if userinfo[user]["portfolio"][ticker]:
            portfolio = userinfo[user]["portfolio"][ticker]
            stocklist = []
            totalquantity = 0
            totalprice = 0
            for item in portfolio:
                if item["volume"] > 0:
                    stocklist.append({"price": item["price"], "volume":item["volume"]})
                else:
                    soldquantity = item["volume"]*(-1)
                    while soldquantity > 0 and stocklist:
                        if stocklist[0]["volume"] > soldquantity:
                            stocklist[0]["volume"] -= soldquantity
                            soldquantity = 0
                        else:
                            soldquantity -= stocklist[0]["volume"]
                            stocklist.pop(0)
            for item in stocklist:
                totalquantity += item["volume"]
                totalprice += (item["volume"] * item["price"])
            if totalquantity == 0:
                continue
            avgprice = round(totalprice/totalquantity,2)
            currenttotalprice = round(totalquantity * market[ticker]["prices"][-1],2)
            totalaccountvalue += currenttotalprice
            percentage = round((100 * (market[ticker]["prices"][-1] - avgprice))/avgprice, 2)
            if (market[ticker]["prices"][-1] - avgprice) >= 0:
                print(f"{ticker}: {totalquantity} @avg${avgprice} -> ${currenttotalprice} (+{percentage}%)")
            else:
                print(f"{ticker}: {totalquantity} @avg${avgprice} -> ${currenttotalprice} ({percentage}%)")
        else:
            continue

    print(f"Total: ${round(totalaccountvalue, 2)}\n")
```

4를 입력하면 portfolio 함수가 실행되는데 이 함수는 해당 유저가 가지고 있는 현금, 해당 유저가 가지고 있는 각 주식의 평균 매입가와 손익률, 해당 유저가 가지고 있는 총 계좌의 금액(현금 + 보유 주식의 가격)을 출력하는 함수이다. 총 계좌에 존재하는 금액을 구하기 위해 totalaccountvalue를 이용하여 모든 가격을 더해줄 것이다.
avgprice = round(totalprice/totalquantity,2) <-- 이 코드 line 전까진 Buyticker, Sellticker와 마찬가지로 평균 매입가를 구하기 위한 코드이고, 각 ticker마다 currenttotalprice를 통해 현재 가격 * 보유 주식 수량으로 보유 주식의 현재 가격을 구했고, 이를 totalaccountvalue에 더해줬다. percentage에는 (현재 가격 - 평균 매입가) * 100 / 평균 매입가으로 손익률을 계산해준 값을 할당했다. 현재 가격 - 평균 매입가가 양수라면 +%, 음수라면 -%가 나오도록 출력해주고, 마지막에 총 계좌에 존재하는 금액을 출력해줬다.
Portfolio 함수의 실행 결과는 왼쪽 사진과 같다. 또한, users.json 파일에 저장된 거래 이력은 오른쪽 사진과 같다. AAPL 보유 주식이 10+5-10 = 5이므로 AAPL 5개 주식만 출력된다.



GOOG 10주를 산 상황에서의 portfolio 함수 출력 결과이다.





2-5. History

```
def history(user):
    transactioninfo = transactionread()
    userinfo = userread()
    print("\n--- Transactions ---")
    if len(userinfo[user]["portfolio"]) == 0:
        print()
        return
    for i in range(len(transactioninfo[user])-1, -1, -1):
        if i == 0:
            print(f"{transactioninfo[user][i]}\n")
            continue
        print(transactioninfo[user][i])
```

5를 입력하면 history 함수가 실행되는데 이 함수는 해당 유저의 transaction을 모두 출력하는 함수이다. 출력 순서는 가장 최근 거래부터 순서대로 출력해야하며 거래 이력이 없다면 “--- Transactions ---”만 출력하고 메인메뉴로 돌아가게 된다. History 함수의 실행 결과는 다음 사진과 같다.

```
===== Select Option =====
1. view
2. buy TICKER QTY
3. sell TICKER QTY
4. portfolio
5. history
6. auto on/off
7. logout
seo> 5

--- Transactions ---
2025-05-21 13:35:40 - BUY 10 GOOG @$474.19
2025-05-21 13:18:26 - SELL 10 AAPL @$67.82
2025-05-21 13:05:03 - BUY 5 AAPL @$102.76
2025-05-21 13:04:48 - BUY 10 AAPL @$100.0
```

```
{
    "seo": [
        "2025-05-21 13:04:48 - BUY 10 AAPL @$100.0",
        "2025-05-21 13:05:03 - BUY 5 AAPL @$102.76",
        "2025-05-21 13:18:26 - SELL 10 AAPL @$67.82",
        "2025-05-21 13:35:40 - BUY 10 GOOG @$474.19"
    ]
}
```

pdf에서 명시되어 있듯이, History함수에서 출력되어야 하는 5가지 데이터가(Time of transaction, Action: BUY or SELL, Quantity, Stock ticker, Trade price) 잘 출력됨을 알 수 있다.

2-6. Auto on/off

```
def autoonoff(user):
    userinfo = userread()
    if userinfo[user]["auto"]:
        print("Auto-trade unenabled.")
        userinfo[user]["auto"] = False
    else:
        print("Auto-trade enabled.")
        userinfo[user]["auto"] = True
    userwrite(userinfo)
```

6을 입력하면 auto on/off 함수가 실행되는데 이 함수는 user의 정보를 userinfo로 받아와서 해당 user의 auto값이 True라면(auto모드 활성화되어 있음), False로 설정해서 비활성화시켜 주고 auto값이 False라면(auto모드 비활성화되어 있음), True로 설정해서 활성화시켜

주는 함수이다.

2-7. Logout
7을 입력하면 logout이 되고 초기 화면으로 돌아가게 된다.
1~7 외의 입력값을 받게 되면 "Invalid Option. Try again."를 출력하고 유저가 재입력하도록 구현했다.

3. Background thread

```
def backgroundthreading():
    while True:
        backgroundmarket()
        autotrade()
        time.sleep(10)
```

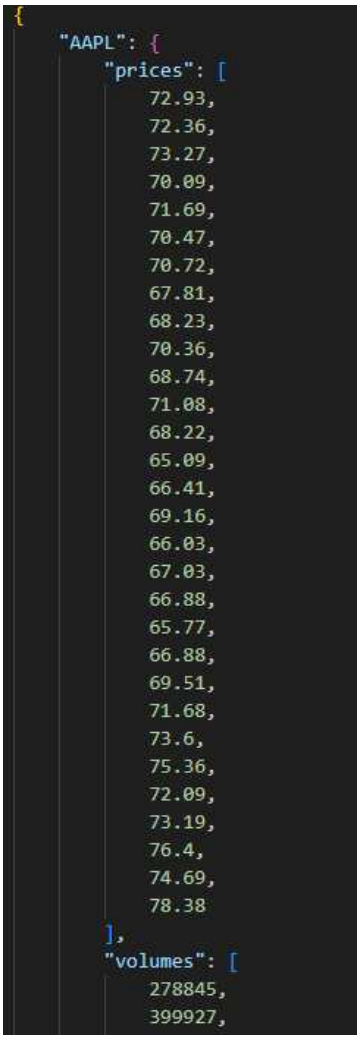
위의 main 함수 코드에서 threading.Thread(target=backgroundthreading, daemon=True).start()를 통해서 프로그램을 실행시키자마자 백그라운드에서 실행되며, while 문을 이용하여 10초가 지날때마다 backgroundmarket 함수와 autotrade 함수가 실행되도록 구현했다.

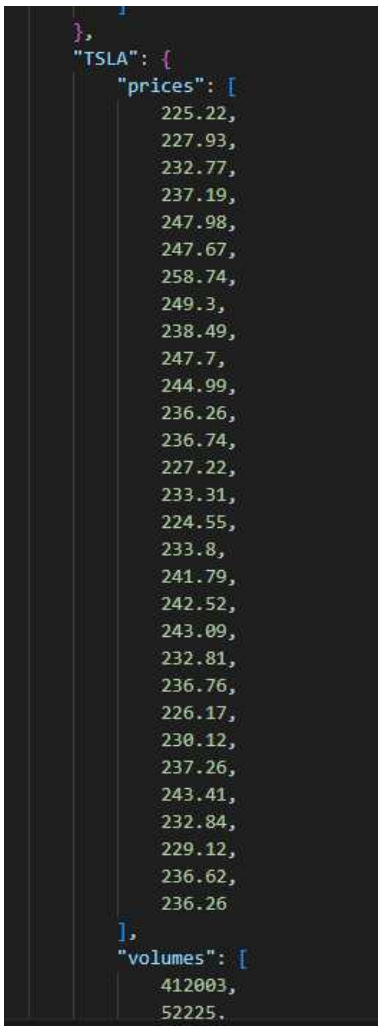
3-1. Backgroundmarket

```
def backgroundmarket():
    market = marketread()
    if not market:
        currenttime = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        market = {
            "AAPL": {"prices": [100.00], "volumes": [10000], "history": [{"time":currenttime, "price":100.00, "volume":10000}]},
            "TSLA": {"prices": [200.00], "volumes": [20000], "history": [{"time":currenttime, "price":200.00, "volume":20000}]},
            "GOOG": {"prices": [500.00], "volumes": [50000], "history": [{"time":currenttime, "price":500.00, "volume":50000}]}
        }
        marketwrite(market)
        time.sleep(10)
    for ticker in market:
        price = market[ticker]["prices"][-1]
        fluctuation = random.uniform(-0.05, 0.05)
        tempprice = price * (1 + fluctuation)
        while tempprice < 0:
            fluctuation = random.uniform(-0.05, 0.05)
            tempprice = price * (1 + fluctuation)
        currentprice = round(tempprice, 2)
        currentvolume = random.randint(10000, 500000)

        currenttime = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        market[ticker]["prices"].append(currentprice)
        market[ticker]["volumes"].append(currentvolume)
        market[ticker]["history"].append({
            "time": currenttime,
            "price": currentprice,
            "volume": currentvolume
        })

        if len(market[ticker]["prices"]) > 30:
            market[ticker]["prices"].pop(0)
            market[ticker]["volumes"].pop(0)
            market[ticker]["history"].pop(0)
    marketwrite(market)
```

백그라운드 마켓을 통해서 각 ticker 별 주식의 가격이 변동하게 된다. 처음에 marketread()를 통해서 market.json으로부터 데이터를 읽어와 market에 할당한다. market이 비어있다면(초기상태), 각 ticker별 주식의 초기 가격과 초기 거래량을 설정해준다. 그 후 marketwrite(market)을 통해서 데이터를 market.json에 저장한다. 그 후 10초 후에 market

에 존재하는 각 ticker(ticker는 market.json에서 딕셔너리 구조로 key로써 존재함)마다 +-0.05의 랜덤한 상승/하락률을 fluctuation으로 지정하여 price를 결정한다. 다만, price가 음수일 경우에는 음수가 아닐때까지 fluctuation을 반복적으로 랜덤하게 생성해내어 current price를 결정한다. 또한 거래량도 10000~500000 사이의 랜덤한 정수 값으로 지정했다. 그 후 현재 시간을 받아주고 market.json에서 받아온 market에 currentprice, currentvolume, currenttime를 추가하여 만들어낸 market의 prices, volumes, history 리스트를 marketwrite(market)를 통해서 market.json에 덮어쓰기해준다. 이 과정에서 너무 많은 prices, volumes, history가 저장되는 것을 방지하기 위해 30개만 출력하도록 구현했다. market.json에 저장되는 형식은 아래의 사진과 같다.





3-2. Autotrade

```python
def autotrade():
    userinfo = userread()
    market = marketread()
    transactioninfo = transactionread()
    currenttime = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    for user in userinfo:
        if userinfo[user]["auto"]:
            if userinfo[user]["strategy"] == 'RandomStrategy':
                for ticker in ["AAPL", "TSLA", "GOOG"]:
                    purchaseflag = random.randint(1, 2)
                    if purchaseflag == 1:
                        continue
                    else:
                        purchasequantity = random.randint(1, 5)
                        currentprice = market[ticker]["prices"][-1]
                        cost = currentprice * purchasequantity
                        if userinfo[user]["balance"] < cost:
                            continue
                        userinfo[user]["balance"] -= cost
                        if not userinfo[user]["portfolio"]:
                            userinfo[user]["portfolio"] = {"AAPL":[], "TSLA": [] , "GOOG":[]}
                        userinfo[user]["portfolio"][ticker].append({
                            "price": currentprice,
                            "volume": purchasequantity
                        })
                        userwrite(userinfo)
                        if not transactioninfo:
                            transactioninfo[user] = []
                        else:
                            if user not in transactioninfo:
                                transactioninfo[user] = []
                        transaction = f"{currenttime} - BUY {purchasequantity} {ticker} @${currentprice}"
                        transactioninfo[user].append(transaction)
                        transactionwrite(transactioninfo)
            elif userinfo[user]["strategy"] == 'MovingAverageStrategy':
                for ticker in ["AAPL", "TSLA", "GOOG"]:
                    shortterm = []
                    longterm = []
                    if len(market[ticker]["prices"]) < 7:
                        continue
                    for i in range(6, -1, -1):
                        price = market[ticker]["prices"][-i-1]
                        if i < 3:
                            shortterm.append(price)
                        longterm.append(price)
                    shortavg = round(sum(shortterm)/len(shortterm),2)
                    longavg = round(sum(longterm)/len(longterm),2)
                    currentprice = market[ticker]["prices"][-1]
                    if not userinfo[user]["portfolio"]:
                        userinfo[user]["portfolio"] = {"AAPL":[], "TSLA": [] , "GOOG":[]}
                    # print(f"ticker: {ticker} shortavg :{shortavg} longavg :{longavg}")
                    if shortavg > longavg:
                        cost = currentprice
                        if userinfo[user]["balance"] < cost:
                            continue
                        userinfo[user]["balance"] -= cost
                        userinfo[user]["portfolio"][ticker].append({
                            "price": currentprice,
                            "volume": 1
                        })
                        userwrite(userinfo)
                        if not transactioninfo:
                            transactioninfo[user] = []
                        else:
                            if user not in transactioninfo:
                                transactioninfo[user] = []
                        transaction = f"{currenttime} - BUY {1} {ticker} @${currentprice}"
                        transactioninfo[user].append(transaction)
                        transactionwrite(transactioninfo)
                    elif shortavg < longavg:
                        proceeds = currentprice
                        availabletotalquantity = 0
                        for item in userinfo[user]["portfolio"][ticker]:
                            availabletotalquantity += item["volume"]
                        if availabletotalquantity < 1:
                            continue
                        userinfo[user]["balance"] += proceeds
                        userinfo[user]["portfolio"][ticker].append({
                            "price": currentprice,
                            "volume": -1
                        })
                        userwrite(userinfo)
                        if not transactioninfo:
                            transactioninfo[user] = []
                        else:
                            if user not in transactioninfo:
                                transactioninfo[user] = []
                        transaction = f"{currenttime} - SELL {1} {ticker} @${currentprice}"
                        transactioninfo[user].append(transaction)
                        transactionwrite(transactioninfo)
```

```
        elif userinfo[user]["strategy"] == 'MomentumStrategy':
            for ticker in ["AAPL", "TSLA", "GOOG"]:
                if len(market[ticker]["prices"]) < 7:
                    continue
                currentprice = market[ticker]["prices"][-1]
                minuteagoprice = market[ticker]["prices"][-7]
                if not userinfo[user]["portfolio"]:
                    userinfo[user]["portfolio"] = {"AAPL":[], "TSLA": [] , "GOOG":[]}
                # print(f"ticker: {ticker} currentprice :{currentprice} minuteagoprice :{minuteagoprice}")
                if currentprice > minuteagoprice:
                    cost = currentprice
                    if userinfo[user]["balance"] < cost:
                        continue
                    userinfo[user]["balance"] -= cost
                    userinfo[user]["portfolio"][ticker].append({
                        "price": currentprice,
                        "volume": 1
                    })
                    userwrite(userinfo)
                    if not transactioninfo:
                        transactioninfo[user] = []
                    else:
                        if user not in transactioninfo:
                            transactioninfo[user] = []
                    transaction = f"{currenttime} - BUY {1} {ticker} @${currentprice}"
                    transactioninfo[user].append(transaction)
                    transactionwrite(transactioninfo)
                elif currentprice < minuteagoprice:
                    proceeds = currentprice
                    availabletotalquantity = 0
                    for item in userinfo[user]["portfolio"][ticker]:
                        availabletotalquantity += item["volume"]
                    if availabletotalquantity < 1:
                        continue
                    userinfo[user]["balance"] += proceeds
                    userinfo[user]["portfolio"][ticker].append({
                        "price": currentprice,
                        "volume": -1
                    })
                    userwrite(userinfo)
                    if not transactioninfo:
                        transactioninfo[user] = []
                    else:
                        if user not in transactioninfo:
                            transactioninfo[user] = []
                    transaction = f"{currenttime} - SELL {1} {ticker} @${currentprice}"
                    transactioninfo[user].append(transaction)
                    transactionwrite(transactioninfo)
        else:
            continue
```

autotrade함수는 auto모드가 활성화되면 백그라운드에서 각 strategy에 따라 자동적으로 주식을 매매해주는 함수이다.

for user in userinfo:

  if userinfo[user]["auto"]:

를 통해서 user 중에서 auto모드가 활성화되어있는 user에 한해서만 autotrade함수가 실행되게 된다. 등록된 모든 user가 auto모드가 비활성화되어 있다면, auto trade함수는 else문의 continue를 통해서 모든 user가 for문을 건너뛰게 되게 된다. 즉, userinfo에 등록된 유저 모두를 대상으로 for문을 돌면서 유저 중에 auto모드가 활성화된 유저에 한해서만 strategy에 따른 자동거래가 발생하게 되는 것이다.

<Automated Trading Strategies>

해당 유저의 auto모드가 활성화되었을 때 3가지 전략에 따라 다른 방식으로 매매가 진행된다.

1번째는 RandomStrategy이다. 각 ticker별로 매입할 확률이 50%이고, 매입량이 1 ~ 5중에 랜덤한 숫자로 지정되어 주식을 매입하는 자동거래 전략이다.

그래서 purchaseflag를 통해서 1, 2 둘 중에 하나의 정수를 받고, '1이라면 매입하지 않음/2라면 매입함'으로 설정하여 50%확률로 매입하는 과정을 표현했다. 또한, 1 ~ 5 중에 랜덤한 정수를 매입량으로 설정하였고, 현재 가격을 받아와 해당 유저의 현금 잔액과 비교하여 매입할 수 있다면 cost를 현금 잔액에서 빼주고, 만일 해당 유저(user)가 거래한 이력(portfolio)

가 없다면 빈 리스트로 각 ticker마다 값을 추가할 리스트를 생성해준다. 그 후 거래 이력을 작성하여 userwrite를 통해 덮어쓰기 해줬다. transaction도 마찬가지로 이력이 없다면 key를 해당 유저의 이름 , value를 빈 리스트로 해당 유저의 거래 내역을 기록할 수 있도록 설정해주고 transaction을 생성하여 transactionwrite를 통해 덮어쓰기 해줬다. 이 과정을 AAPL, TLSA, GOOG 3개의 ticker에 대해서 반복한다.

2번째는 MovingAverageStrategy이다. 각 ticker의 short-term average(최근 3개의 주식 가격의 평균)과 long-term average(최근 7개의 주식 가격의 평균)을 비교하여 short-term average > long-term average이라면 해당 ticker의 주식을 1주 구매하고, short-term average < long-term average이라면 해당 ticker의 주식을 1주 판매하는 자동거래 전략이다. 당연하게도 1주를 판매할 때 해당 주식을 보유하지 않고 있다면 판매하지 않도록 구현했다.
shortterm, longterm 2개의 빈 리스트를 만들어주고, market.json에 생성된 가격이 7개 이상이 될 때까지는 자동거래가 되지 않도록(그 아래의 코드가 실행되지 않도록) continue로 반복문을 넘겨줬다.

```
for i in range(6, -1, -1):
    price = market[ticker]["prices"][-i-1]
    if i >= 4:
        shortterm.append(price)
    longterm.append(price)
```

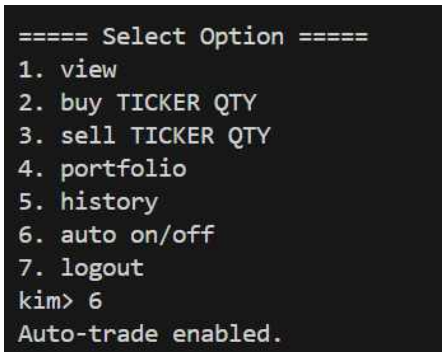
을 통해서 해당 ticker의 최근 7개의 주식 가격을 longterm 리스트에 넣고, 해당 ticker의 최근 7개의 주식 가격을 shortterm 리스트에 넣었다. 그 후 2개의 리스트의 평균을 계산해주고 short-term average > long-term average이라면 해당 ticker의 주식을 현재 가격으로 1주 구매하고 해당 유저의 현금잔액에서 cost를 빼주고 해당 유저의 portfolio에 거래 이력을 기록하고 userwrite를 통해서 users.json을 업데이트해주고, 1주 구매한 거래 내역을 transaction 변수에 할당하여 transactions.json파일에 해당 유저의 transaction(거래 내역)을 업데이트해줬다.
마찬가지로 short-term average < long-term average이라면 해당 ticker의 주식을 현재 가격으로 1주 판매한다. 판매하는 과정에서 availabletotalquantity 변수에 해당 유저가 보유한 주식의 수량을 할당해주고 1과 비교하여 1주를 판매할 수 있는지 확인한다. 만약 보유 주식 수량이 1주 미만이라면 continue를 통해서 매도가 발생하지 않도록 구현했다. 판매할 수 있다면 proceeds를 계산하여 해당 유저의 현금잔액에 판매한 주식의 현재 가격을 더해주고 그 후 과정은 구매과정과 똑같이 해당 유저의 portfolio에 거래 이력을 기록하고 userwrite를 통해서 users.json을 업데이트해주고, 1주 구매한 거래 내역을 transaction 변수에 할당하여 transactions.json파일에 해당 유저의 transaction(거래 내역)을 업데이트해줬다.

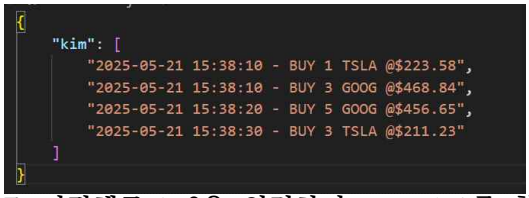
마지막 3번째는 MomentumStrategy이다. 각 ticker의 현재 가격과 60초 전의 가격(현재 가격에서 6번째 이전의 가격)을 비교하여 현재 가격이 더 크다면 해당 ticker의 주식을 1주 구매하고, 현재 가격이 더 작다면 해당 ticker의 주식을 1주 판매하는 자동거래 전략이다.
2번째 전략과 마찬가지로, market.json에 생성된 가격이 7개 이상이 될 때까지는 자동거래가 되지 않도록(그 아래의 코드가 실행되지 않도록) continue로 반복문을 넘겨줬다.

currentprice = market[ticker]["prices"][-1]을 통해서 현재 가격을 할당해줬고,
minuteagoprice = market[ticker]["prices"][-7]을 통해서 60초 전의 가격을 할당해줬다. 이 둘을 비교해서 currentprice > minuteagoprice 이라면 해당 ticker의 주식을 현재 가격으로 1주 구매하고 해당 유저의 현금잔액에서 cost를 빼주고 해당 유저의 portfolio에 거래 이력을 기록하고 userwrite를 통해서 users.json을 업데이트해주고, 1주 구매한 거래 내역을 transaction 변수에 할당하여 transactions.json파일에 해당 유저의 transaction(거래 내역)을 업데이트해줬다.
마찬가지로 currentprice < minuteagoprice 이라면 해당 ticker의 주식을 현재 가격으로 1주 판매한다. 판매하는 과정에서 availabletotalquantity 변수에 해당 유저가 보유한 주식의 수량을 할당해주고 1과 비교하여 1주를 판매할 수 있는지 확인한다. 만약 보유 주식 수량이 1주 미만이라면 continue를 통해서 매도가 발생하지 않도록 구현했다. 판매할 수 있다면 proceeds를 계산하여 해당 유저의 현금잔액에 판매한 주식의 현재 가격을 더해주고 그 후 과정은 구매과정과 똑같이 해당 유저의 portfolio에 거래 이력을 기록하고 userwrite를 통해서 users.json을 업데이트해주고, 1주 구매한 거래 내역을 transaction 변수에 할당하여 transactions.json파일에 해당 유저의 transaction(거래 내역)을 업데이트해줬다.

결론적으로, backgroundthreading함수 안에 있는 Backgroundmarket(), Autotrade()와 time.sleep(10)을 통해서 market.json에 있는 주식의 가격과 거래량을 10초마다 백그라운드에서 갱신해주고, auto모드가 활성화된 유저가 존재한다면 10초마다 백그라운드에서 해당 유저가 선택한 전략에 맞게 자동거래를 진행하도록 구현했다.

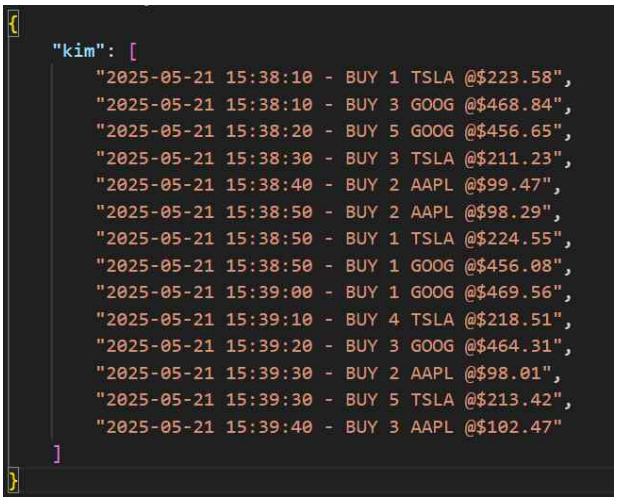
백그라운드에서 10초마다 주식의 가격과 거래량이 갱신되고, auto모드 활성화/비활성화에 따른 자동거래가 진행되는 것의 결과는 아래의 사진과 같다.



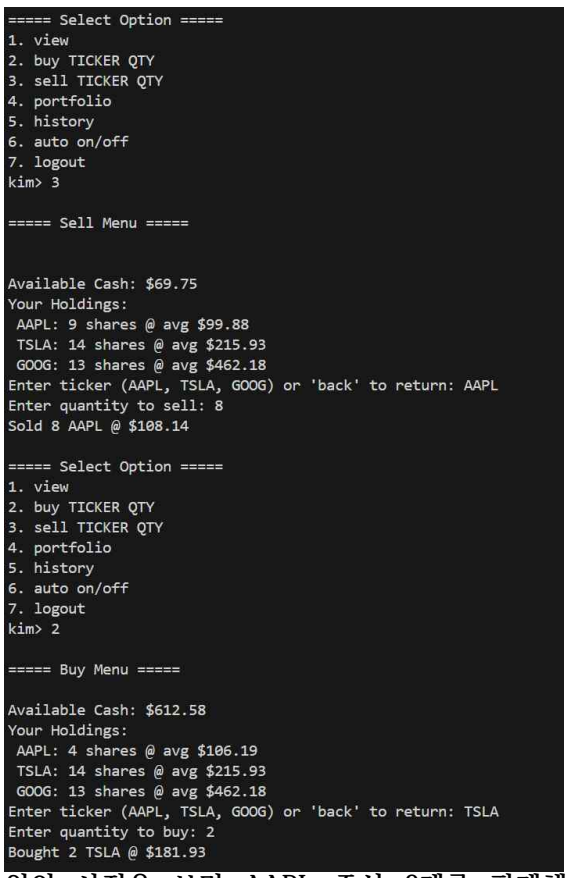


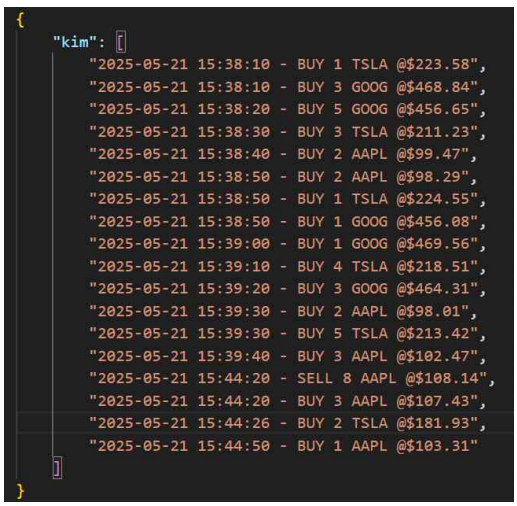
kim이라는 유저의 전략을 RandomStrategy로 지정해주고 6을 입력하여 auto모드를 활성화시켜서 오른쪽 위의 사진인 users.json을 보면 auto의 값이 true로 변경되고 오른쪽 아래의

사진인 transactions.json을 보면 10초마다 자동으로 1 ~ 5 사이의 랜덤한 개수의 주식을 매입하는 것을 알 수 있다. 보유한 현금 잔액이 부족하면 매입을 더 이상 하지 않는다.



보유한 현금이 69.75.. 정도라 더 이상 매입할 수 있는 주식이 없다.
또한, 자동거래를 활성화한 상태로 주식을 직접 사고 팔 수 있다.





위의 사진을 보면 AAPL 주식 8개를 판매해서 9-8=1 AAPL 1주를 보유해야하지만, Buy Menu에서 Your Holdings에는 AAPL 4주를 보유하고 있다고 나온다. 그리고 오른쪽 사진을 보면 transaction에 자동거래로 AAPL 3주를 구매한 것을 알 수 있다. 이렇듯 자동거래와 수동거래가 동시에 잘 작동함을 알 수 있다.

그렇다면 이번에는 MovingAverageStrategy에 대한 작동을 나타내보면,

```
===== Select Option =====
1. view
2. buy TICKER QTY
3. sell TICKER QTY
4. portfolio
5. history
6. auto on/off
7. logout
seo> 6
Auto-trade enabled.
===== Select Option =====
1. view
2. buy TICKER QTY
3. sell TICKER QTY
4. portfolio
5. history
6. auto on/off
7. logout
seo> ticker: AAPL shortavg :124.64 longavg :126.92
ticker: TSLA shortavg :158.41 longavg :162.64
ticker: GOOG shortavg :483.09 longavg :476.05
```

```
"seo": {
    "password": "Seo1234@",
    "strategy": "MovingAverageStrategy",
    "balance": 9020.06,
    "auto": true,
    "portfolio": {
        "AAPL": [
            {
```

seo라는 유저는 자동거래 전략이 MovingAverageStrategy이고, auto모드를 6을 입력함으로써 활성화시켜줬다.

ticker: AAPL shortavg :124.64 longavg :126.92

ticker: TSLA shortavg :158.41 longavg :162.64

ticker: GOOG shortavg :483.09 longavg :476.05

이렇게 출력한 것은 디버깅&계산 값이 제대로 출력되고 있는지 확인하기 위함이다. (제출할때는 삭제하겠습니다.)

AAPL, TSLA는 shortavg < longavg 이므로 1주를 판매해야 하고, GOOG는 shortavg > longavg 이므로 1주를 구매해야 한다.

```
"seo": [
    "2025-05-21 15:52:27 - BUY 10 AAPL @$123.21",
    "2025-05-21 15:52:33 - SELL 1 AAPL @$122.55",
    "2025-05-21 15:52:33 - BUY 1 GOOG @$470.9",
```

위의 사진은 transactions.json 파일의 스크린샷인데 원래 seo라는 유저는 10주의 AAPL을 구매했었고, AAPL은 보유 주식이 1주 이상이기 때문에 1주를 판매했다. 하지만 TSLA는 보유하고 있던 주식이 없기 때문에 판매가 발생하지 않음을 알 수 있다. 또한, GOOG를 1주 구매한 사실로 보아 정상적으로 자동거래가 이뤄지고 있음을 알 수 있다.

그리고 위에서 AAPL의 shortavg, longavg가 제대로 된 값이 반환하고 있는지 확인해보면 15:52:33의 시간에서의 AAPL의 가격은 122.55이다.

오른쪽 사진에서 shortavg를 확인해보면 122.55+123.21+128.15=373.91

373.91 / 3 = 124.64가 나오게 된다.

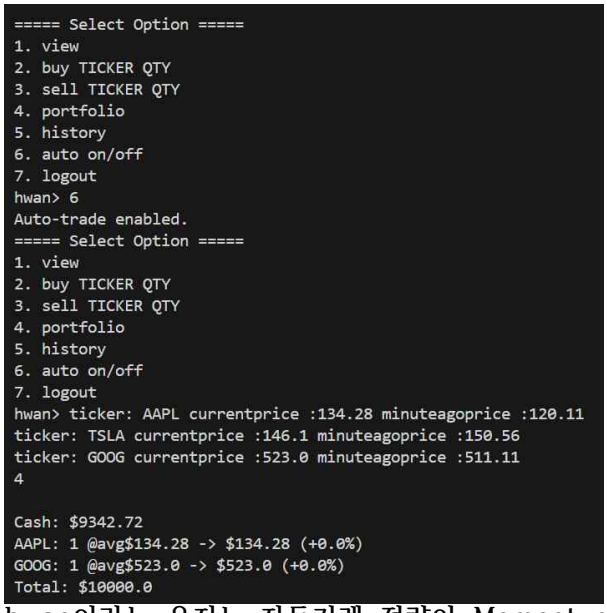
longavg를 확인해보면 7개의 가격을 모두 더한 값 = 888.41

888.41 / 3 = 126.92가 나오게 된다.

즉, 저 위에 AAPL shortavg :124.64 longavg :126.92가 제대로 출력되고 있음을 알 수 있다.

```
130.31,
126.62,
126.63,
130.94,
128.15,
123.21,
122.55,
```

그렇다면 이번에는 MomentumStrategy에 대한 작동을 나타내보면,



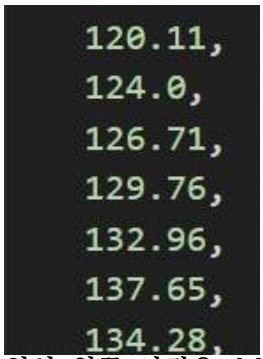
hwan이라는 유저는 자동거래 전략이 MomentumStrategy이고, auto모드를 6을 입력함으로써 활성화시켜줬다.

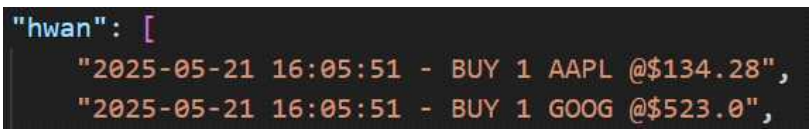
ticker: AAPL currentprice :134.28 minuteagoprice :120.11

ticker: TSLA currentprice :146.1 minuteagoprice :150.56

ticker: GOOG currentprice :523.0 minuteagoprice :511.11

이렇게 출력한 것은 디버깅&계산 값이 제대로 출력되고 있는지 확인하기 위함이다. (제출할때는 삭제하겠습니다.)

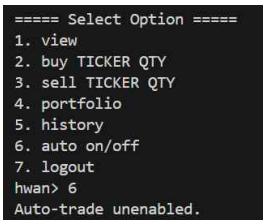




위의 왼쪽 사진은 AAPL의 가격인데, 현재 가격이 134.28이고 현재 가격의 6번째 이전의 가격이 120.11이므로 저 위에 AAPL currentprice :134.28 minuteagoprice :120.11가 제대로 출력되고 있음을 알 수 있다.

또한, TSLA는 currentprice < minuteagoprice 이므로 1주를 판매해야 하고, AAPL, GOOG는 currentprice > minuteagoprice 이므로 1주를 구매해야 한다. 위의 오른쪽 사진을 보면 transactions.json 파일의 스크린샷인데 AAPL, GOOG는 조건에 따라 1주 구매하였고 TSLA는 보유한 주식이 없기 때문에 판매하지 않음을 알 수 있다.

그리고 auto모드가 활성화된 유저에서 6을 또 입력하면 auto모드가 비활성화된다.



결론적으로, auto모드가 활성화된 유저에 대해서 3개의 전략 모두 auto모드가 조건에 맞게 잘 작동하고 있음을 알 수 있다.

제가 레포트를 작성하면서 시간이 좀 오래 걸리기도 했고 매입/매도하는 과정과 자동 거래가 진행되는 과정을 각각 스크린 샷을 찍다보니 최종적으로 제출한 users.json, market.json, transactions.json 파일의 내용이 레포트에서 사용한 스크린 샷과 일치하지 않는 내용이 있을 수도 있는 점 양해 부탁드립니다.