Software Engineering Project:

Test Plan

**TEST DATA SET**

The core data components of the RSI Trading App are the User, Stock, and Simulator classes. Testing will be conducted using a default User that is hard coded into the User class and does not rely on the external user database. The user will have the essential data fields; name, password, email, and capital; as well as two stocks: one share of Apple and one share of Google.

USER:

1. Username: Test1
2. Password: $Password1
3. Email: email@email.com
4. Capital: 200000 ($2000)

The stock objects will be default stocks that do not rely on an API call for current prices. 14 days’ worth of price data will be hard coded into the default cases for calculating RSI. For comparison, the these stocks’ RSI, calculated from a 3rd party source, were recorded on the day of collecting the price data.

APPLE

1. Ticker = AAPL
2. Company = Apple Inc.
3. CurrentPrice = 148.69
4. HistoricPrices = 147.92, 150.81, 150.44, 151.28, 150.96, 151.49, 150.02, 148.96, 149.80, 152.57, 148.85, 149.32, 148.64, 148.69
5. NumPrices = 14
6. NumOwned = 1

GOOGLE

1. Ticker = GOOGL
2. Company = Apple Inc.
3. CurrentPrice = Alphabet Inc. Class A
4. HistoricPrices = 2984.97, 2932.52, 2934.96, 2992.91, 2987.76, 2981.52, 2981.24, 3014.18, 2999.05, 2941.57, 2935.14, 2934.35, 2856.12, 2935.11
5. NumPrices = 14
6. NumOwned = 1

The Simulator class heavily relies on actual stock data to be tested, so testing of the simulator will be done with Apple and Google stock, but after API code has been written, and the stocks can be given 100 historic prices from the Alpha Vantage API.

**UNIT TESTING**

**USER CLASS METHODS**

**Function: validateUsername()**

Data structure:

1. String username

Data Set:

1. “” : empty string
2. “uuuu”: less than the required 5 characters
3. “uuuuuuuuuuuuuuuuu”: longer than the required 16 characters
4. “test1” : exactly 5 characters
5. “test111111111111”: exactly 16 characters

Results:

1. Return -1
2. Return -2
3. Return -3
4. Return -4
5. Return 1

**Function: validatePassword()**

Data structure:

1. String password

Data Set:

1. “” : empty string
2. “ppp1”: less than the required 5 characters
3. “pppppppppppppppppppp1”: longer than the required 20 characters
4. “passs” : exactly 5 characters but missing a number
5. “passssssssssssssssss”: exactly 20 characters but missing a number
6. “password1” : missing a capital letter
7. “PASSWORD1: missing a lowercase number
8. “11111111” : missing a letter
9. “Password1”: missing a special character
10. “$Password1”: proper password

Results:

1. Return -1
2. Return -2
3. Return -3
4. Return -4
5. Return -4
6. Return -6
7. Return -5
8. Return -5
9. Return -7
10. Return 1

**Function: testEmail()**

Data structure:

1. String email address

Data Set:

1. “” : empty string
2. “email”: missing email provider
3. “email@”: missing email provider
4. “@gmail.com” : missing email address
5. “email@gmail.com”: valid format

Results:

1. Return -1
2. Return -3
3. Return -3
4. Return -3
5. Return 1

**Function: addStock()**

Data Structure:

1. Stock[] stocks
2. String ticker

Data Set:

1. Stocks = [], ticker = “APPL” : no stocks, add apple
2. Stocks = [APPL], ticker = “APPL” : has apple, add apple
3. Stocks = [GOOGL], ticker = “APPL” : has google, add apple

Result:

1. Stocks = [APPL]
2. Stocks = [APPL]
3. Stocks = [GOOGL, APPL]

**Function: removeTracking()**

Data Structure:

1. Stock[] stocks
2. String ticker

Data Set:

1. Stocks = [], ticker = “APPL” : no stocks, remove apple
2. Stocks = [APPL], ticker = “APPL” : has apple, remove apple
3. Stocks = [GOOGL], ticker = “APPL” : has google, remove apple
4. Stocks = [GOOGL, APPL], ticker = “APPL” : has google and apple, remove apple

Results:

1. Stocks = []
2. Stocks = []
3. Stocks = [GOOGL]
4. Stocks = [GOOGL]

**Function: Buy Stock**

Data Structure:

1. User {int capital, Stock[] stocks {string ticker, int price, int amount}
2. String ticker
3. Int amount

Data Set:

1. User {capital = 0, stocks { APPL, 50, 1}}; ticker = APPL, amount = 1 : user with no money tries to buy apple stock
2. User {capital = 49, stocks { APPL, 50, 1}}; ticker = APPL, amount = 1 : user with less than required money tries to buy apple stock
3. User {capital = 50, stocks { APPL, 50, 1}}; ticker = APPL, amount = 1 : user with the exact amount money tries to buy apple stock
4. User {capital = 51, stocks { APPL, 50, 1}}; ticker = APPL, amount = 1 : user with enough money tries to buy apple stock
5. User {capital = 99, stocks { APPL, 50, 2}}; ticker = APPL, amount = 1 : user with enough money to buy 1 stock, but not enough to buy 2
6. User {capital = 100, stocks { APPL, 50, 2}}; ticker = APPL, amount = 1 : user with the exact amount of money to buy 2 stocks
7. User {capital = 101, stocks { APPL, 50, 2}}; ticker = APPL, amount = 1 : user with enough money to buy 2 stocks
8. User {capital = 50, stocks { APPL, 50, 0}}; ticker = APPL, amount = 1 : user attempts to buy zero stocks

Results

1. User {capital = 0, stocks { APPL, 50, 1}}; notify user, nothing happens
2. User {capital = 49, stocks { APPL, 50, 1}}; notify user, nothing happens
3. User {capital = 0, stocks { APPL, 50, 2}}; deduct capital, increase stock amount
4. User {capital = 1, stocks { APPL, 50, 2}}; deduct capital, increase stock amount
5. User {capital = 99, stocks { APPL, 50, 1}}; notify user, nothing happens
6. User {capital = 0, stocks { APPL, 50, 3}}; deduct capital, increase stock amount by 2
7. User {capital = 1, stocks { APPL, 50, 3}}; deduct capital, increase stock amount by 2
8. User {capital = 50, stocks { APPL, 50, 0}}; nothing happens

**Function: Sell Stock**

Data Structure:

1. User {int capital, Stock[] stocks {string ticker, int price, int amount}
2. String ticker
3. Int amount

Data Set:

1. User {capital = 0, stocks { APPL, 50, 0}}; ticker = APPL, amount = 1 : user with no stocks tries to sell
2. User {capital = 0, stocks { APPL, 50, 1}}; ticker = APPL, amount = 1 : user with one stock tries to sell 1 stock
3. User {capital = 0, stocks { APPL, 50, 2}}; ticker = APPL, amount = 1 : user with two stocks tries to sell 1 stock
4. User {capital = 0, stocks { APPL, 50, 1}}; ticker = APPL, amount = 2 : user with 1 stock tries to sell two stocks
5. User {capital = 0, stocks { APPL, 50, 2}}; ticker = APPL, amount = 2 : user with 2 stocks tries to sell 2 stocks
6. User {capital = 0, stocks { APPL, 50, 3}}; ticker = APPL, amount = 2 : user with 3 stocks tries to sell 2 stocks
7. User {capital = 0, stocks { APPL, 50, 1}}; ticker = APPL, amount = 0 : user with 1 stock tries to sell 0 stocks

Results

1. User {capital = 0, stocks { APPL, 50, 0}}; notify user, nothing happens
2. User {capital = 50, stocks { APPL, 50, 0}}; increase capital, deduct stocks
3. User {capital = 50, stocks { APPL, 50, 1}}; increase capital, deduct stocks
4. User {capital = 0, stocks { APPL, 50, 1}}; notify user, nothing happens
5. User {capital = 100, stocks { APPL, 50, 0}}; increase capital, deduct stocks
6. User {capital = 100, stocks { APPL, 50, 1}}; increase capital, deduct stocks
7. User {capital = 0, stocks { APPL, 50, 1}}; nothing happens

**STOCK METHODS**

**Function: calculateAverageGain()**

Data Structure

1. Int numPeriods: should be at least 14, representing at least 2 weeks of data
2. double[numPeriods] appleClosingPrice

Data Set:

1. numPeriods= 13appleClosingPrice[147.92, 150.81, 150.44, 151.28, 150.96, 151.49, 150.02, 148.96, 149.80, 152.57, 148.85, 149.32, 148.64, 148.69] :   
    number of periods is too small
2. numPeriods= 14, appleClosingPrice[147.92, 150.81, 150.44, 151.28, 150.96, 151.49, 150.02, 148.96, 149.80, 152.57, 148.85, 149.32, 148.64, 148.69]  
    number of periods is 14, and the stock data is accurate to historical data
3. numPeriods= 14, appleClosingPrice[147.92, 151.81, 150.44, 152.28, 150.96, 152.49, 150.02, 149.96, 149.80, 153.57, 148.85, 150.32, 148.64, 149.69]  
    number of periods is 14, ½ of closing prices have been increased to produce higher gains
4. numPeriods= 14, appleClosingPrice[147.92, 149.81, 150.44, 150.28, 150.96, 150.49, 150.02, 147.96, 149.80, 151.57, 147.85, 149.32, 147.64, 148.69]  
    number of periods is 14, ½ closing prices have been reduced to produce lower gains

Results:

1. algo refuses to run because there is not enough data
2. number around .001 to .008
3. number higher than result 2
4. number lower than result 2

**Function: calculateAverageLoss()**

Data Structure: same data set as average gain

Data Set: same data set as average gain

Results:

1. algo does not run because there is not enough data
2. number around .001 to .008
3. number lower than result 2
4. number higher than result 2

**Function: calculateCurrentGain()**

Data Structure:

1. double currentClosingPrice
2. double previousClosingPrice

Data Sets:

1. current = 151.1, previous = 150.5: a day where there was a gain
2. current = 151.1, previous = 151.1: a day where there was no change
3. current = 151.1, previous = 151.5: a day where there was a loss

Results:

1. return 0.6
2. return 0
3. return 0

**Function: calculateCurrentLoss()**

Data Structure: same as calculateCurrentGain

Data Sets: same as calculateCurrentGain

Results:

1. return 0
2. return 0
3. return 0.4

**Function: calculateRSI()**

Data Structure:

1. Int numPeriods: should be at least 14, representing at least 2 weeks of data
2. Double averageGains
3. Double averageLosses
4. double actualRSI value: using historical data calculated by apple itself. (apple 46.06)(google 56.91)

Data Set:

1. numPeriods = 13, result 2 averageGain(APPL), result 2 average loss(APPL), actualRSI 46.06
2. numPeriods = 14, result 2 averageGain(APPL), result 2 average loss(APPL), actualRSI 46.06
3. numPeriods = 14, result 2 averageGain(GOOGL), result 2 average loss(GOOGL), actualRSI 56.91

Result:

1. Algo does not run because there is not enough data
2. Number around 45
3. Number around 55

**Function: getStockSuggestion():**

Data Structure:

1. Double RSI

Data Set:

1. RSI = .29
2. RSI = .31
3. RSI = .69
4. RSI = .71

Results:

1. “Buy”
2. “Hold”
3. “Hold”
4. “Sell”

**SYSTEM TESTING**

**Test Case 1: Logging in:**

Functions:

1. testUsername()
2. testPassword()
3. loginRequest()

Data Format:

String username, String password

Data Sets:

1. “”, “password1” : Empty username
2. “logintest1”, “” : Empty password
3. “logintest2”, “password2” Username that is not in the database
4. “logintest1”, “password2” : Username is in database, password does not match database entry
5. “logintest1”, “password1” : Username is in database, password matches

Results:

1. Enter =>Rejected by testUsername() =>Notify user of invalid entry
2. Enter => Rejected by testPassword() => Notify user of invalid entry
3. Enter =>Passes tests =>loginRequest() =>Call database => Return no result => notify user of invalid Username
4. Enter =>Passes tests =>loginRequest() =>Call database => Return password mismatch => notify user of invalid Password
5. Enter =>Passes tests =>loginRequest() =>Call database => Return user account => entry is allowed

**Test Case 2: Creating a new account:**

Functions:

1. testEmail()
2. testUsername()
3. testPassword()
4. newAccountRequest()

Data Format:

String username, String password, String email

Data Sets:

1. “newtest2”, “password2”, “” : empty email
2. “”, “password2”, “email@gmail.com” : empty username
3. “newtest2”, “”, “email@gmail.com” : empty password
4. “newtest2”, “p” , “email@gmail.com” : password too short
5. “newtest2”, “ppppppppppppppppppppp” , “email@gmail.com” : password too long
6. “logintest1”, “password1” , “email@gmail.com” : username already in use
7. “logintest1”, “password1” , “email” : invalid email entry
8. “newtest2”, “password2”, “email@gmail.com” : valid inputs

Results:

1. Enter =>testEmail() rejects email =>Notify user of invalid entry
2. Enter =>testUsername() rejects empty username =>Notify user of invalid entry
3. Enter =>testPassword() rejects empty password =>Notify user of invalid entry
4. Enter =>testPassword() rejects short password =>Notify user of invalid entry
5. Enter =>testPassword() rejects long password =>Notify user of invalid entry
6. Enter => Call database => Return user account =>notify user the username is already in use
7. Enter =>notify user of invalid email format
8. Enter => Call database =>no result =>Push user to database => entry is allowed

**Test Case 3: Updating the Stock information for the user’s tracked stocks with API**

Function:

updateStocks()

Data Format:

User Object { String username, String password, String email, Stock[] stocks}

Data Sets:

1. User Object{ Stock[] = []}: User with no stocks
2. User Object{ Stock[] = [Stock{APPL}]}: user with apple stock
3. User Object{ Stock[] = [Stock{APPL}, Stock{GOOGL}]}: user with apple and google stock

Results:

1. Enter => No stocks => no update
2. Enter =>updateStocks() =>API call using APPL ticker => get data => parse result into stock object
3. Enter =>updateStocks() => (Loop start) API call using tickers => get Data => parse result to stock object => loop until reach the end of stock list

**Test Case 4: Entering/Updating the homepage:**

Functions:

1. generateHomescreen()

Data Format:

User Object { String username, String password, String email, Stock[] stocks}

Data Sets:

1. User Object = null : null user
2. User Object{ Stock[] = []}: User with no stocks
3. User Object{ Stock[] = [Stock{APPL}]}: user with apple stock
4. User Object{ Stock[] = [Stock{APPL}, Stock{GOOGL}]}: user with apple and google stock

Results:

1. Enter => user = null => notify user / return to login screen
2. Enter => No stocks =>generate empty UI
3. Enter =>One stock => generate UI with 1 element
4. Enter =>two stocks => generate UI with 2 elements

**Test Case 5: Simulation**

Functions:

1. runSimulation()

Data Format:

Stock Object {with 100 days worth of closing price data, and timestamps}  
 Simulator: {BuyThreshold, SellThreshold, Buy Amount, Sell Amount}  
 \*Due to the nature of the simulation, it is not predictable whether the RSI trading strategy will guarantee a higher net worth after the simulation. Successful test will just show that the changing of the buy and sell thresholds should result in a different set of transactions where stocks are bought at a lower price when the buy threshold is lower, and stocks are soldat a higher price when the sell threshold is higher. It is also possible greater or fewer transactions will be made.

Data Sets:

1. Apple Stock with Simulator Buy Threshold 40, Sell Threshold 70
2. Apple Stock with Simulator Buy Threshold 30, Sell Threshold 70
3. Apple Stock with Simulator Buy Threshold 50, Sell Threshold 70
4. Apple Stock with Simulator Buy Threshold 40, Sell Threshold 60
5. Apple Stock with Simulator Buy Threshold 40, Sell Threshold 80

Results:

1. 2021-08-17: sold 1 shares at 150.19

2021-09-07: sold 1 shares at 156.69

2021-09-17: bought 1 shares at 146.06

2021-09-20: bought 1 shares at 142.94

2021-09-21: bought 1 shares at 143.43

2021-10-21: sold 1 shares at 149.48

2021-10-22: sold 1 shares at 148.69

2021-10-25: sold 1 shares at 148.64

1. 2021-08-17: sold 1 shares at 150.19

2021-09-07: sold 1 shares at 156.69

2021-09-20: bought 1 shares at 142.94

2021-09-21: bought 1 shares at 143.43

2021-09-24: bought 1 shares at 146.92

2021-10-21: sold 1 shares at 149.48

2021-10-22: sold 1 shares at 148.69

2021-10-25: sold 1 shares at 148.64

1. 2021-08-17: sold 1 shares at 150.19

2021-09-07: sold 1 shares at 156.69

2021-09-10: bought 1 shares at 148.97

2021-09-14: bought 1 shares at 148.12

2021-09-17: bought 1 shares at 146.06

2021-10-21: sold 1 shares at 149.48

2021-10-22: sold 1 shares at 148.69

2021-10-25: sold 1 shares at 148.64

2021-11-10: bought 1 shares at 147.92

2021-11-11: bought 1 shares at 147.87

2021-11-16: bought 1 shares at 151.0

2021-11-18: sold 1 shares at 157.87

2021-11-19: sold 1 shares at 160.55

2021-11-22: sold 1 shares at 161.02

1. 2021-08-17: sold 1 shares at 150.19

2021-08-30: sold 1 shares at 153.12

2021-09-17: bought 1 shares at 146.06

2021-09-20: bought 1 shares at 142.94

2021-09-21: bought 1 shares at 143.43

2021-10-18: sold 1 shares at 146.55

2021-10-19: sold 1 shares at 148.76

2021-10-20: sold 1 shares at 149.26

1. 2021-09-07: sold 1 shares at 156.69

2021-09-17: bought 1 shares at 146.06

2021-09-20: bought 1 shares at 142.94

2021-10-21: sold 1 shares at 149.48

2021-10-28: sold 1 shares at 152.57

2021-11-30: sold 1 shares at 165.3