Assignment T1

Part 1

Nothing has changed. We are still Team Zappa, consisting of Zachary Brown (zb2292), Anne Zepecki (amz2145), Jiaxuan (Percy) Pan (jp4131), and Arun Ram-Mohan (amr2356), and we are still using Python and Windows. Our team's github repository is still at https://github.com/zachbrown12/ZAPPA_4156 .

Part 2

Our service will provide an opportunity for users to simulate stock trading in a competitive game environment. It will present a way to virtually buy and sell stocks at real market prices, in order to grow a theoretical portfolio over time and compare results against those achieved by other users. In order to do this, we will be integrating with a third party API that will allow us to fetch stock prices in real time.

The service will provide functionality that will:

- Allow users to login / initialize an account in the game
- Allow users to buy a stock at current price, deducting the cost from their cash balance
- Allow users to sell a stock at current price, adding the proceeds to their cash balance
- Allow users to fetch their current stock portfolio
- Allow users to fetch their current cash balance
- Allow users to check how the total value of their account ranks against the other players in the game

Its users will be players who wish to try stock trading online without actual financial risk, in order to learn the skills involved through practice, or players who already have trading experience and wish to measure their skills against others. A player will use the functionality to buy stocks from the virtual cash stockpile associated with the player's account, and sell stocks from the player's portfolio for virtual cash, all based on the current prices of stocks at the time the transactions are entered.

Our service will store the following persistent data:

1) User account and information. The information of the account includes username, password, the user’s current portfolio, and the user’s cash balance
2) Game information. It contains the time remaining in the game, users, groups, etc…
3) User trading log. It includes every time the user trades stock. This is used to calculate the user portfolios.

The following data is accumulated by raw data above and stored in our system:
1) Historical values of user portfolios. It can be calculated by portfolio and real-time stock data (from third-party API).
2) The leaderboard data. Each user’s rank on the leaderboard is based on the value of their cash balance plus the current combined value of all their stock holdings.

Part 3

We intend to test that our service will do what it is supposed to and provides the intended functionality by providing multiple layers of testing. First, we will do manual testing using Postman to ensure that the APIs that we have implemented have the expected behavior. We will write test cases in Postman to hit all the expected behavior (including error handling) via manual testing. We will also do manual blackbox testing where we interact with the service via the command line in the same manner that a user of our service would. We will also be implementing unit tests for all the functionality that we have implemented in our service (including helper functions e.g. database functionality, etc). We will be able to evaluate the level of unit test coverage using the Python coverage tool as we used in the individual course assignments. By implementing tests on several levels, we hope to achieve high confidence in our service.

In order to test that our service does not behave badly if it is used in unintended ways we will implement unit tests for each flow. To ensure we cover as many possible scenarios we will use a ‘testing checklist’ and build tests around those. For each flow/method we will implement tests which cover valid, invalid, and unexpected scenarios. For example, if a user is trading stocks in our app it may be a valid scenario to buy 10 stocks. It may be invalid to buy -45 stocks (which we should handle with an error). It may be unexpected to buy 999,999,999 stocks (which we may handle through a message to the user). If we make a list of testing scenarios, and test both manually/through a library like Python unittest we should catch most invalid inputs.

In order to test that our service handles data the way it’s supposed to we should include test scenarios which cover ‘expected’ results. For example we should create a few portfolios which invest in various stocks and track them ‘offline’ (outside of our app). If we compare the ‘offline’ results to the results from our service, we should expect those results to be the same. We should ensure to test with multiple users across multiple stocks to ensure that multiple clients/users will not impact each other in unexpected ways. We may use tools like DB Browser for SQLite while testing our application to ensure that data is Inserted/Updated/Deleted as expected. We will also seed data to the database with testing data to ensure the service acts as expected with a variety of users/data.