17-780 Final Project: Python Robinhood Wrapper (Link)

Evangeline Liu (evangell), Sara Harvey-Browne (sharveyb), Chinedu Ojukwu (cio), John Paul Harriman (jharrima), Anthony Attipoe(aattipoe)

Introduction

Pyrh is an unofficial Python API used to trade within the Robinhood framework. Although the API allows users to access their Robinhood account, there were many flaws. The main flaws of the API were the following: poor documentation, broken methods, methods outside of scope, and multiple methods with similar functionality. These broke the following API Design Principles defined in class: D1, D2, D3, D4, 1.1, 1.2, 1.5, 2.1, 2.9, 4.1, 4.3.

The presence of these many flaws within the API made it difficult for users to perform simple actions. Some of these actions include finding the popularity of various instruments, getting information such as historical data and current buy/sell prices for doing analysis on an instrument, getting the basic quote information for instruments, and providing current market data.

Flaws & Solutions

Flaw 1: Poor Documentation

It was found that documentation for multiple methods was incredibly insufficient, where the intended use case and functionality were unclear. In addition, there was little to no client code, which also made it difficult to understand the API. To resolve this, we ensured that our redesign had full documentation and client code so the functionality was clear and our API is easy to use.

Flaw 2: Incorrect Return Types

There were multiple methods that returned inappropriate return types. Some methods returned HTTP response objects which violates the level of abstraction we are working at and exposes to clients that this API is built atop a web API. In some cases, documentation mentioned that some methods were supposed to return floats for monetary values; however, the methods were found to return a two-dimensional list with the string representation of these floats. We thought it would be more appropriate to represent money using a fixed point decimal instead of a floating-point format. To do this we considered two classes, the python-money and Decimal classes, which could both be used to represent money to the fourth digits place. This rounding was the Robinhood standard (which adheres to SEC Guidelines), and since Decimal is very well integrated with the existing python math functionality, we decided to choose the Decimal class.

Flaw 3: Broken Methods

Through functional testing, we found that several of the existing API methods were

broken. The method, last_updated_at, was documented to return the time when the stock's information was last updated, but the wrong dictionary key was used, resulting in crashing the program. In addition, get_quote_list failed to parse a list of ticker symbols, such as "AMZN,AAPL", formatted according to the original documentation. There were many methods in the original API that were dependent on get_quote_list, which also failed due to get_quote_lists' flawed implementation. Other methods had bugs in their functionality due to broken endpoints, incorrect dictionary keys, incorrect/old method names/parameters, and insufficient return type checks.

Flaw 4: Methods Outside of Scope

The API's desired functionality was to trade within the Robinhood framework, but there were a few methods that fell outside this scope. These included: get_url, get, post, and robinhood.to_json. With our redesign we remove these methods, so users weren't concerned with the API being written atop a web API.

Flaw 5: API Organization

One of the largest flaws with the API was the number of methods with similar functionality. We found that in addition to having similar functionality, the naming and lack of documentation made them very unclear. To resolve this, we decided to divide the API into multiple classes. With each class having limited and specific functionality we hoped to increase the accessibility of the API. The new design comprised of the following classes:

- Robinhood Session: Robinhood superclass
- User: Object used to access the current user's portfolio and orders
- Portfolio: User's portfolio comprising of the user's owned instruments
- Order: An order which a user can place
- Instrument: Object used to represent tradable instruments within the Robinhood platform

Assessment

While initially we picked the Robinhood API because of the design flaws (unwieldy and unorganized as well as inappropriate return types), we were somewhat surprised at how many of the original API methods threw exceptions when we actually attempted to use our wrapper methods in unit tests. From incorrect method signatures to incorrect dictionary keys and broken endpoints, we had to figure out ways to get around the errors if the errors were uninformative or it was unclear what the correct ways to access them were. If the issue was programmatically obvious we made the appropriate change, otherwise we often attempted to copy from a working version of the source code into the original broken API and then try to fix it from there.

A primary lesson learned from this project was that it pays to be good at searching for resources! We also learned that the most difficult part of projects like these can be 1) coming up with an idea or finding an appropriate idea and 2) refining the idea to a point where it is feasible

to implement it. Coding actually felt like the easiest part since we already had a direction for the project. As a team we learned how to speed up the debugging process by working together and discussing issues that we were facing.

Further Work

Currently, since our API is mainly wrapped around the original API is subject to the connectivity issues present when actually using the API. Sometimes when initializing a session, an HTTP connection pool error would occur and then a new session with new email verification would be needed in order to interface with the API. Future work could be using multiple threads or a similar retry logic to ensure the connection is fully established with the Robinhood API. This would result in the overall session experience being smoother for the user.

Future work that would give the API a more robust functionality, would be to add high-level market analysis and portfolio analysis to the API. This would include stock comparison, averages over time, forecasting, modeling, and other functionality that lies beyond reporting the state of the Robinhood ecosystem. This could give the user insights into what instruments they should buy or sell to bolster their portfolio. The API can act as a companion app to a Robinhood user. This could be done easily through multiple financial analysis APIs that exist in the wild. Combining our APIs real-time aspect with powerful analysis could prove to be highly beneficial to the user.

Interview

We interviewed someone with knowledge of sales and trading as well as programming to see how our API fared against their own use-cases. The main issues seemed to stem from how Robinhood's HTTP responses' values weren't always intuitive. For example, finding out how many tradable shares you have of a certain stock is difficult to figure out because instead of looking at the stock's "quantity" you needed to look at the "intraday quantity." We thought about creating wrappers over every return type, but that isn't designing for evolution if Robinhood decides to change its response backend.

One positive comment we got was in relation to our positions. The endpoint returns all positions you've ever owned whether or not you currently do. This is excess information in the eyes of the interviewee since users usually do not usually care about instruments they no longer own. We fixed this by defaulting to returning currently owned instruments and gave users the option to see what they previously owned by adding an optional argument. Another comment was in regards to naming convention. The term "equity" seemed to carry many meanings throughout the API where a similar name like "capital" would be clearer to the user. The main reason we were against changing something like this is because we wanted it to be contained within a Robinhood scope. More advanced traders would likely still be able to understand what equity meant, but it would help users that came from the Robinhood platform a lot more in terms of vocabulary. The rest of the functionality seemed standard and not confusing for a brokerage

API which was not the case for the original implementation.

Division of Work

- Evangeline Liu implemented the Instrument class and its unit tests. She found many of the bugs in the original source code when running the unit tests and successfully fixed some of them. She also contributed to the initial brainstorming to find possible APIs and helped write up the report.
- Sara Harvey-Browne implemented the User class and its unit tests. She designed the final presentation , wrote the README, collaborated on the final report, and added additional comments to provide clarity to the redesign code. In addition, she was part of the API redesign process.
- Anthony Attipoe worked on designing and implementing the new Order class. This consisted of all the supported versions of orders provided by the previous API such as stop limit orders, stop loss orders, market orders, etc. Simple tests and client code were written to the new API. He also contributed towards issue tracking as well as writing up the final report.
- Chinedu Ojukwu implemented the portfolio and the relevant tests regarding the portfolio. Handled the client-facing monetary return types, requirements gathering, issue tracking, and the final report. Also, contributed to API discovery and the initial API design.
- John Paul Harriman implemented the session aspect of the project to make it easier to use. Fixed issues within the source code so that the wrapping API wouldn't break. Did the initial testing of pyrh to see how the API functioned/didn't function. Compiled the documentation on readthedocs and wrote client code. Contributed to other classes throughout with input and bug fixes. Conducted the interview to gain insight.

Project Documentation

Github: https://github.com/anthonyattipoe/pyrh_redesign
Full Documentation: https://pyrh-redesign.readthedocs.io/en/latest/
Google Drive: https://drive.google.com/open?id=1bPPd9QQa8urj5bqNVhVCNJUVY2jyWq7K