name soon:

a Python-based Trading Platform

by Trevor Rowland(Github: dBCooper2)

Two years ago, I made a program in Python that connected to TD Ameritrade’s API through Alex Golec’s tda-api project that viewed my stocks in TD. After taking some business classes and wanting to revisit programming, I have decided to improve the program’s original design and add some features. This project mainly focuses on building a stable platform to add more tasks eventually. Additionally, I seek to integrate what I’ve been learning and what is expected of me in my career. I am building this with practice in mind in Python, SQL, Microsoft Excel, and Tableau or Microsoft PowerBI. This paper is meant to structure the project and provide a list of deliverables in a more organized manner than GitHub issues.

# Abstract

pythonTrader seeks to be a one-stop-shop for finance updates, offering access to the TD Ameritrade and Charles Schwab Trading Platform, SEC financial reports from publicly-traded companies, and the ability to locally store data from those platforms using Docker and SQL, as well as process it using programs like Tableau, Excel, and other Python programs. Additionally, the program will use a GUI with Flask to provide users with a simple frontend.

# Deliverables

* Research
  + Structure of a Python GUI Application(directories, best practices for files/variables/etc)
  + Relational Databases
  + Python Libraries(tda-api, numpy, pandas, matplotlib, sqlalchemy, flask, jinja, PyQT5)
  + APIs
    - TD Ameritrade and Schwab APIs
    - Accessing SEC Data
* Python-Trader: Updated from portfolio\_viewing\_script
  + Redone classes to add Orders and Stock classes, with the security class being detached from a Position and purely stating the security or company’s information.
  + Update the program to handle requests using tda-api
  + Update the program to handle the tda-api Streaming Client
  + Update the old App in the TDAmeritrade with new Project details
  + Update the program when the Schwab/TD Ameritrade Merger is complete
* Flask Frontend
  + Preliminary Sketches
  + GUI: Main Menu, Portfolio Viewer, Trading Deck, Experiments Page, Database Settings, User Settings, Program Settings, Newsfeed, Persistent Text-Entry Box
  + Shell/CLI-Like Interface
* SQL Database
  + Determine Structure
    - Identify what Data are tables and what are columns
    - Convert the databases into one relational database
    - Connect Database to the Python Project
    - Establish functions to move, update and copy data from the database to the program
  + SQLite in Python
    - Connect Database to Flask
    - Display Position data in a Jinja Template
    - Display Security data in a Jinja Template
    - Display Order data in a Jinja Template
    - Add values to the Positions, Securities or Orders databases through an element in the Flask project.
* Excel/CSV Functionality
  + Export Stock Info as an Excel Worksheet or CSV row
  + Export Position Info as an Excel Worksheet or CSV row
  + Export Trade Info as an Excel Worksheet or CSV row
* Tableau Functionality
  + Export Stock info from the API as a dashboard
  + Export Stock info from the Database as a dashboard
  + Repeat these for Orders, Positions and Balances
* Trading
  + Make Trade using a Program to execute a trade
  + Make Trade using the Live Streaming Client
* Experiments – COMING SOON
* Settings Page – COMING SOON

## Useful Links(Update from Github)

* Github:
  + Current project: <https://github.com/dBCooper2/pythonTrading>
  + Old project: <https://github.com/dBCooper2/portfolio_viewing_script>
  + tda-api: <https://github.com/alexgolec/tda-api>
* tda-api read-the-docs: <https://tda-api.readthedocs.io/en/latest/>
* Flask FullStackPython Guide: <https://www.fullstackpython.com/flask.html>
* RealPython Jinja Guide: <https://realpython.com/primer-on-jinja-templating/>
* Google Data Source Python Library: <https://developers.google.com/chart/interactive/docs/dev/gviz_api_lib#tojsonresponse-example>