StockPulse: Real-time Investment Tracker

CSCI 621 Project Proposal

Albert Chen, Zunyang Ma

Fall 2023

1 Introduction

Our team has selected the project option to build a comprehensive stock price application designed to deliver live and historical stock data to users using the Yahoo Finance API, as inspired by CNBC's API and award-winning website [1]. Our stock price application aims to not only provide access to real-time market data but also historical records presented to users in a dynamic display that features an interactive line graph, as well as tabular data detailing summary statistics. Moreover, the application will empower users with personalized alert functionalities, tailored to their stock monitoring needs.

The application's architecture strives for an intuitive interface with optimized response times, enabling ease of use in making financial decisions. The vision for our application is to engineer a user-centric stock price application with capabilities to track and analyze trends for stocks such as Apple, Tesla, and Amazon.

2 Conceptual Model

Our conceptual model outlines the architecture of our stock price application and details the resulting end user experience. Upon initialization, the application displays a graphical representation of the stock price data and a table of associated statistics similar to those found on CNBC's website by leveraging the Jinja templating engine [2]. Additional features allow users to explore in-depth historical data or configure alerts based on their preferences. Yahoo Finance's API is targeted for real-time stock price information, while historical stock data is sourced from the the application's SQLite database, which stores data stored from previous requests and thus does not require additional API calls.

The business logic lives on the Flask server acting as the bridge between the user interface and the data sources. Its purpose is twofold, as it not only processes end user requests from the interface to fetch live data from Yahoo Finance's API but also retrieves data from the SQLite database as needed [2]. The SQLite database will also be updated periodically, thus triggering alerts to the user based on their indicated preferences. The inner-workings of the application are not privy to the end user who is the recipient of a seamless user experience. Due to robust request processing by the Flask server, users are enabled access to real-time and historical data critical to their financial needs.

Overall, the stock price application is designed with an intuitive interface for a strong user experience enabling access to real-time and historical stock data from Yahoo Finance. With visualizations and summary statistics of current stock prices presented to users upon application startup, users can also explore historical trends and set alerts for a personalized product experience. The key architectural components of our application from frontend to datastore include the Jinja templating engine for dynamic web pages, a Flask backend for handling requests and data retrieval, and a SQLite database to store historical data and user preferences. Each component in collaboration is designed to deliver a comprehensive and user-friendly stock market analysis application.

3 Logical Model

The logical model maps out the flow of data throughout the various components of our stock price application detailing the pathways triggered by user interface interactions leading to various forms of data processing within the application. From the frontend interface, user-driven requests are received for processing server-side and directed to the correct data source based on the type of request made by the user. Upon application startup, users are presented with current stock price information, which is data returned from the Yahoo Finance API. If the user decides to search for a specific stock, the Yahoo Finance API will similarly return the current stock price information for the specified stock replacing the initially populated data fields with new data.

Alternatively, if a user decides to search for historical stock data instead of current stock data, their request is processed by the Flask server and funneled through a secondary application pathway and communicates with a SQLite database. Once a user establishes alerts to track stocks, the Flask server processes this request similarly by communicating with the SQLite database to receive notifications. By separating the logic for request processing across two different pathways, the application's data delivery mechanism is optimized since the database only needs to be periodically updated. This decoupled approach to server design allows request processing to be optimized since only data from the appropriate data source needs to be retrieved, whether from the Yahoo Finance API or the local data store containing saved data. Once data is returned from either the Yahoo Finance API or the SQLite database, Jinja (template engine) renders the data to the end user.

The data flow with the stock price application is governed by the request-response cycle. Users interact with a Jinja-powered frontend to request stock information and their request is subsequently processed by the Flask backend to either fetch live data from Yahoo Finance's API or historical data from the SQLite database. The backend then serves the requested data, which the frontend displays with the help of Jinja (template engine). The decoupled API processing and database design ensure seamless access to stock prices, historical data, and custom alerts set by the user.

4 Physical Data Model

The physical model of our stock price application proceeds to describe the implementation details of our application. In our search to present key statistics sourced from the Yahoo Finance API such as "Open", "Prev Close", "52 Week High", "52 Week Low", and "YTD % Change ", we plan to implement the stock price application with the following technologies: the Jinja templating engine for the frontend, a Flask backend, and a SQLite database for lightweight data management [1]. A new route will be created on the Flask server to make a request to the Yahoo Finance API for each unique statistic requested, thus "52 Week High" will have its own route associated with a Jinja template for rendering on the user interface. Similarly, if a user wants to view historical data such as "YTD % Change" a route will be created that is directed to the SQLite database by querying the relevant table and returning this data. An ORM such as SQLAlchemy will be utilized to communicate with the SQLite database for either retrieving data or updating tables in the database [2].

The implementation of our stock price application is based on the following three components, Jinja Templates for displaying data, Flask server processing for implementing API calls and business logic, and SQLite database for storing historical data. Requests for real-time and historical data is managed by routes created on the Flask server that direct API calls to the Yahoo Finance API and queries to the SQLite Database, respectively. An additional library such as SQLAlchemy may be integrated to optimize database queries and table updates. Once data is returned from either the Yahoo Finance API or the SQLite database, the Flask routes associated Jinja template renders the data to the user.

5 Project Outline and Deliverables

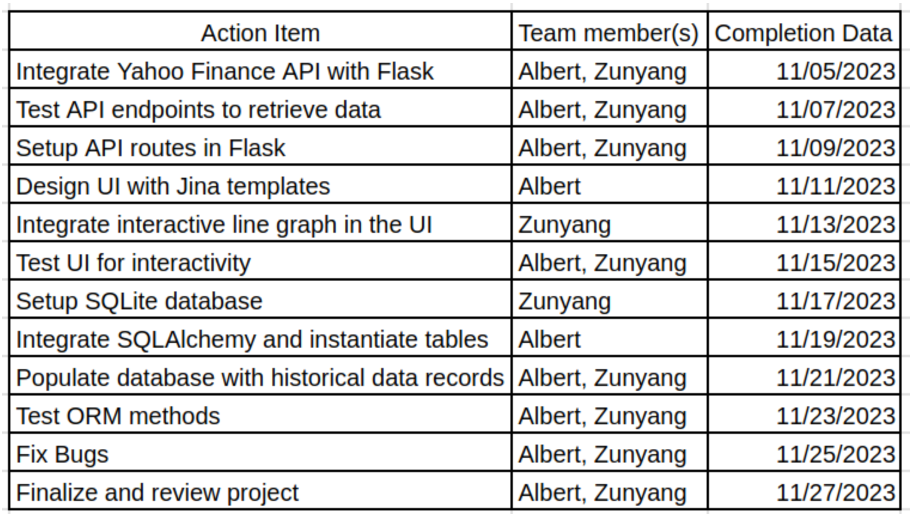
Our project will be to build a stock price application. This project will be split into three separate components, which include API integration and server connection, visualization and display of statistics, and database management with an Object Relational Mapping library (ORM). With regards to API integration and server connection, we will integrate the API to the Flask Server to obtain the statistics required to be displayed to the user for each individual Stock searched. This will be done primarily using Yahoo Finance's API documentation [3] to connect our Flask server to the API and obtain all the data specified in our project models.

Following the acquisition of the required data from the API, the second component of the project requires visually representing the data captured from the API. This will be done with Jinja templates that are associated with the

Flask routes per API call. The user interface of the application will allow a user to interact with the data by providing an interactive line graph. Additional data will be presented in a table outlining key statistics regrading the stock shown.

The final component of the project will be the database component that will be achieved with a SQLite database. The goal is to store historical data records in tables over a 52 week period. An ORM such as SQLAlchemy will be utilized to facilitate table creation and modification, as well as data retrieval for non-relational and relational data. Each component of the project will be achieved within a one week sprint, and the remaining time will be spent on testing the application's effectiveness and efficiency.

6 Project Timeline



Action Item	Team member(s)	Completion Data
Integrate Yahoo Finance API with Flask	Albert, Zunyang	11/05/2023
Test API endpoints to retrieve data	Albert, Zunyang	11/07/2023
Setup API routes in Flask	Albert, Zunyang	11/09/2023
Design UI with Jina templates	Albert	11/11/2023
Integrate interactive line graph in the UI	Zunyang	11/13/2023
Test UI for interactivity	Albert, Zunyang	11/15/2023
Setup SQLite database	Zunyang	11/17/2023
Integrate SQLAlchemy and instantiate tables	Albert	11/19/2023
Populate database with historical data records	Albert, Zunyang	11/21/2023
Test ORM methods	Albert, Zunyang	11/23/2023
Fix Bugs	Albert, Zunyang	11/25/2023
Finalize and review project	Albert, Zunyang	11/27/2023

Figure 1: Action Item Timeline

References

[1] "Check out apple's stock price (aapl) in real time." [Online]. Available: www.cnbc.com/quotes/AAPL

[2] M. Grinberg, "Flask-web-development,2nd-edition."

[3] "Api documentation - yahoofinance documentation." [Online]. Available: python-yahoofinance.readthedocs.io/en/latest/api.html