CSCI-3010 Project Proposal

Sofia Lange

October 6th, 2019

1 Introduction

The project idea that I have is to create a web application where a use can choose a stock market trading strategy and the app will keep track of the stock indicators for that strategy and notify the user by email when it is time to buy or sell. My plan is to start with one strategy that I am familiar with and implement that as an option, and possibly add more strategies on top of that if I have time.

2 Technology

2.1 Language

Python 3 - Completely Familiar
HTML - Moderately Familiar
CSS - Moderately Familiar
JavaScript - Moderately Familiar

2.2 Libraries

Flask - Moderately familiar
Pandas - Completely familiar
Numpy - Moderately familiar
Requests - Moderately familiar
SQLite - Not familiar
Docker - Not familiar
APScheduler - Not familiar
JSON - Moderately familiar
Flask-Mail - Not familiar

2.3 Other Technologies

Version Control - GitHub - Completely familiar
Deployment - Docker - Not familiar

Deployment (Backup Plan) - Google App Engine - Slightly familiar
Continuous Integration - Jenkins - Not familiar

3 Requirements

My project requires access to stock prices at closing time each day. If it cannot access these prices it will not be able to calculate whether the indicator has been triggered that day to say to buy or sell. I have found multiple free to use APIs which provide the information required so that if one fails it can move to another.

My project needs to be able to send emails to users based on criteria being met on a daily, monthly, or quarterly, etc. basis. If this doesn't work the next option would be to simply display a message in the app and the user would have to log in to check the status of their strategy and what it says they should do.

My project needs to be able to associate users with their chosen trading strategies. The alternative would be to simply add users to an email list which goes out to anyone who signs up for it instead of having to store users.

4 Resources

I plan to use this API: click here because it has the specific options I need and I can get the calculated numbers instead of having to implement the exponential moving average myself.
There is a limit of 5 requests per minute, or 500 per day. So, I will check to see how many requests I need to make each day and if it exceeds 5 I will wait a minute before running the next 5 requests, and so on. Within the scope of this project I won't need to make more than 500 requests per day.

5 Architecture

5.1 Design Patterns

1. **Factory** I will use the factory method for my Strategy object because this design pattern is useful if you plan to add in other derived classes as time goes on and I add more strategies to the app.
   I may move to another method as I begin implementation if I find this one doesn't work for my application, and will update by Homework 3 Submission Date
2. **Singleton** I will use the singleton pattern for my database connection

5.2 Objects

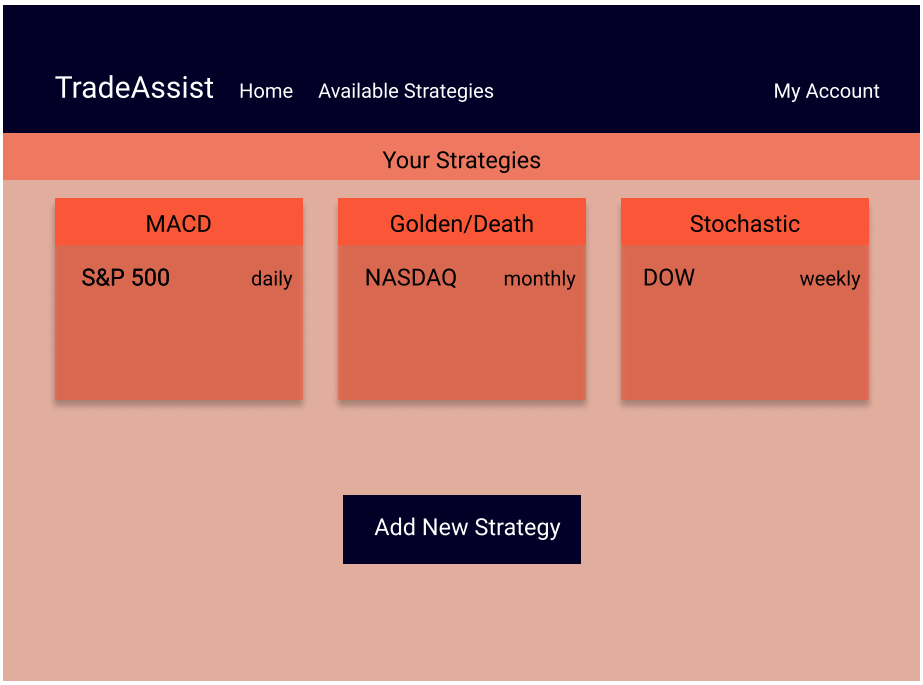
Object	Job	Interactions	Derived Classes
Strategy	Encapsulate the information of a given trading strategy	User, Mailer, Scheduler	All included trading strategies will inherit from base Strategy class
User	Contain info about user	Strategy, Mailer, Scheduler, DbConnection	No derived class
Mailer	Will create and send well formatted emails	User, Scheduler, Strategy	No derived class
Scheduler	Checks if buy/sell signals have happened based on date time and dispatches emails to users who should buy/sell	User, Mailer, DbConnection, Strategy	May have derived classes which perform different operations for weekly, monthly, quarterly checks
DbConnection	CRUD ops with database	Scheduler, User	No derived class

5.3 Data Exchange Overview

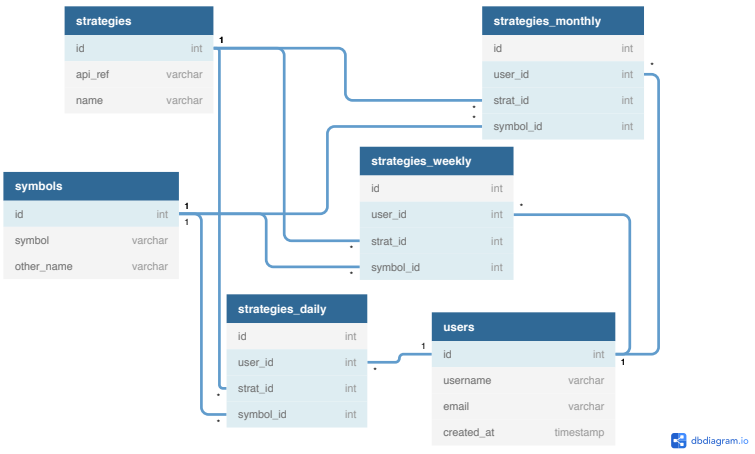
- The app will maintain a connection to the database through the DbConnection object
- The User object will talk to the DbConnection object in order to log in or sign up
- The User object will contain an array of Strategy objects
- The Strategy object will simply be an interface where it can calculate if there is a need to buy or sell for this particular strategy on a given date. I think I will have each strategy have its own functionality to make API calls based on the date and return if a buy / sell is needed.
- The Scheduler will need to interact with Strategy objects in order to determine if any buy sells are needed, every day. Then, based on these results it will get all the users who need to be notified from the DbConnection object, and finally will use the Mailer object to send emails to all those users
- The Mailer object will simply take in a user or list of users and create emails for them based on their strategies and will send these emails

6 UI Framework

Here is a link to the figma prototype : figma link



7 Database Configuration



8 Schedule

1. WEEK 1 (Oct 11)
   a) Basic Flask app setup, user should be able to see home page when they go to correct URL, set up git and CI
   b) Required knowledge - how to set up and initialize a Flask app, how to use Jenkins with Flask
   c) I will have to learn about using Jenkins with Flask

2. WEEK 2 (Oct 18) - Database setup, user should be able to log in and view which trading strategies they have chosen
   Also, the timing and stock they've chosen for this strategy
   HOMEWORK PLAN

   (a) Will turn in code along with screen shots or capture of functioning log in page which displays your information back to you as well as shots of interacting with the database

   (b) Required knowledge - SQlite database set up and connecting to it through a flask app, also processing user sign up data through web forms, checking if they exist already, signing them up if not

   (c) I will need to learn how to interact with a SQLite database. I know how to process web forms in Flask but I haven't used a database with Flask before so I will have to read up on that

3. WEEK 3 (Oct 25)
   a) Implement framework for making API calls and checking if certain indicators have been triggered for available strategies
   a.2) Later (Week 4), APScheduler will handle scheduling the API calls
   b) Required knowledge - making API calls in Flask and parsing JSON results to check if results match certain criteria
   c) I have knowledge of making API calls in Flask and parsing JSON results, so this week will mostly be focused on how to best structure the way it checks and makes different calls based on the available strategies

4. WEEK 4 (Nov 1)
   a) Set up and test scheduled tasks. The app should be making an API call every day after market close, and checking the results against all strategies.
   b) Required knowledge - scheduling tasks based on date and time in Flask, not based on requests
   c) I will have to learn how to schedule tasks, in my research it looks like APScheduler can be used to perform tasks based on date and time so I will have to learn how to use it to accomplish my specific goals

5. WEEK 5 (Nov 8) - Set up framework for sending emails based on user and which strategy they have selected. At this point the app will have the ability to send emails based on requests.
   HOMEWORK PLAN

(a) Will turn in code along with screen shots or capture of functioning scheduled tasks based on time, as well as demonstration of making a Get or Post and receiving an email from the app

(b) Required knowledge - How to send emails from a Flask app and format the content based on user data

(c) I will need to learn how to send emails from Flask, there is a plugin called Flask-Mail which looks simple enough to make use of

6. WEEK 6 (Nov 15)
   a) At this point I would like for the app to be able to send emails based on the scheduled tasks above to any users registered for the service if their chosen strategies indicator calls for a buy/sell
   b) Required knowledge - How to integrate multiple different functionalities together without breaking the others c) I will not need to learn anything as far as I can tell as long as I understand how the other foundations of my app work, it will be simple enough to integrate it all together

7. WEEK 7 (Nov 22) - The app will be able to execute all functionality at this point and will also have a user-friendly interface which makes sense and looks polished for signing up for the service and checking status of strategy
   HOMEWORK PLAN

   (a) Will turn in code along with screen capture and shots of using the app from signing up to receiving an email based on a strategy which was input by the user

   (b) Required knowledge - Mostly HTML and CSS will be used for this final week

   (c) I will need to probably learn some tricks for html and css that will make my app look better

8. WEEK 8 (Nov 29) - Planned break, may do catch up if above schedule is not met

9. WEEK 9 (TBA) - Will work on finishing touches to front end and practice demoing the app for presentation, should be completely finished by this point HOMEWORK PLAN

   (a) Will turn in code along with application demo and presentation slides

   (b) Required knowledge - N/A

   (c) N/A