CIS 2368 FALL 2024 Final Project

Stock Brokerage System – Investors, Stocks, and Bonds

For your final project you will be building a full stack web application following a standard 3 tier architecture:

UI – REST API Services – Database

This application will help users to plan and manage a stock brokerage system.
Overview:
The user has the option to manage the investors, their individual stocks, and their individual bonds. Thus, you will need to create at least the following tables:
investor, stock, **bond**, **stocktransaction, bondtransaction**.
For all tables, the id should be the primary key, and autogenerated. Pick appropriate data types for each column.

The **investor** table should at minimum contain the following columns:
id, firstname, lastname

The **stock** table should have the following columns:
id, stockname, abbreviation, currentprice

The **bond** table should have the following columns:
id, bondname, abbreviation, currentprice

The **stocktransaction** table should have the following columns:
id, date, investorid, stockid, quantity
The **id** and **date** should be automatically generated. The **investorid** column is a **foreign key** to the **investor** table, indicating who performed the transaction. The **stockid** column is a **foreign key** to the **stock** table identifying the stock for the transaction. The **quantity**, if **positive**, shows a **buy**, if **negative**, shows a **sale**.

The **bondtransaction** table should have the following columns:
id, date, investorid, bondid, quantity
The **id** and **date** should be automatically generated. The **investorid** column is a **foreign key** to the **investor** table, indicating who performed the transaction. The **bondid** column is a **foreign key** to the **bond** table identifying the bond for the transaction. The **quantity**, if **positive**, shows a **buy**, if **negative**, shows a **sale**.

There is one important rule that must always be adhered to: a sale can **never** exceed the current possession of a stock or bond. You must check an investor's current possession of a certain stock, before they can sell it. There are multiple ways to approach this: you can create additional tables if you'd like, or calculate current possession with the help of registered transactions. It is up to you whether to handle this logic in the front end, or the back end.

The necessary user stories that need to be fulfilled are:

- User creates, updates, deletes, or selects an investor.
- User creates, updates, or deletes a bond.
- User creates, updates, or deletes a stock.
- User sees the investors current portfolio (stocks and bonds).
- User makes a transaction for the investor (either buy or sell a stock or bond)
- User cancels (deletes) a transaction (stock or bond) entirely.

Therefore, you will need **CRUD APIs** for all of your tables. Depending on your implementation, this includes single and/or multiple records at once.

For simplicity we assume every investor has unlimited money and we do not have to worry about having enough funds to make a purchase. You can also assume the user inputs only valid data, and you do not have to error-check user input (for instance inputting strings into a number field).

Note: When the user creates a **transaction**, the user should have to **pick an existing investor, and an existing stock/bond** (e.g. from a dropdown, or a list), and should not have to type it as an input (where errors could happen)

Requirements

This project will be developed in **2 Sprints**. The first sprint focuses on the Python backend development, all the logic/functionality, and the deployment of the REST API services with Flask. The second sprint focuses on the UI development in JavaScript using the EJS. Both **code submissions** will happen into the **same repository** on **GitHub Classroom**. There must be a folder **'frontend'** and a folder **'backend'** in that repo, with the corresponding code bases. **Do not forget to comment and document your references.**

Each Sprint for the project is due at **midnight** of the listed due date on the schedule.

No late submission will be accepted.

Part of the submission for Sprint 2 will be a 3-5-minute long **presentation video**, giving an overview over the functionality of your application features, and explaining the technology, frameworks, etc. that you have used. This video must be made **available through a link** (for instance, recorded with MS Teams and provided is the link to MS Stream). **Do NOT submit video files on CANVAS or in your repo! Commit the link in a text file in your repo.**

Details for Sprint 1 - Deadline October 12th, 2024

For Sprint 1 you will have to design and implement all the REST API services you will need so that your app can interact with your database. (**ALL CRUD operations for all the tables.)**
These REST Services need to be implemented using the **Python Flask Framework**. You will be testing your backend code with **Postman**, without the need to have the UI pages implemented.
On CANVAS, **submit the link** to your GitHub repo. If you work as a **team of two,** one person needs to submit the **GitHub classroom link on CANVAS**, and the other person **MUST** submit a note stating the **name, student ID, and section of their partner.**
You are allowed to make modifications and changes to your Sprint 1 code even after the submission deadline, but only the last commit before the deadline will be considered for grading.

Details for Sprint 2 - Deadline November 23rd 2024

You can choose to let the user manage all necessary CRUD operations on one page (SPA) or on individual pages (multi page application). You can use any of the templates introduced in class if you like.

NOTE: Under no circumstances should the user ever see the ids used as primary keys on any of the pages! (You can, however, introduce secondary Ids of your choice of you want).

Other Requirements

- Make sure the username and password for the database you setup are credentials you're willing to share. Do not use personal passwords for this project, which you might be using anywhere else.
- Make sure your project compiles and runs. If the project doesn't run, you will forfeit points.
- Submit early and often! **Only code that has been submitted to your repo can be graded.**

Unexpected circumstances can occur (power outages, etc) – make sure you have commits to fall back to if need be.

- Using GitHub Classroom to setup the remote repo for this assignment

Use the provided link to create your private GitHub repo:

https://classroom.github.com/a/L7vBKDkl

Please push your commits on a regular basis.

What to Turn in:

Commit your source code (**properly commented**) to your folders in your private repository in the GitHub classroom. **Do not zip the files together.** Your GitHub repository should show multiple meaningful commits illustrating how you worked through the project.
There will be **2 separate assignments** listed in CANVAS - one for each sprint, so that 2 separate grades can be assigned. Each assignment/sprint will be worth **50 points max**. Together, they are worth **100 points max** for the final project (not counting the extra credit).
For **both sprints,** on CANVAS simply submit the **same link** to your GitHub repository containing your project. When working in a team, submit a **project document 24h before due date for BOTH sprints**, outlining the work distribution among the team members. It should be **part of your repository** (no separate document submission on CANVAS). For **Sprint 2** just **update the project document**. Also remember, **both team members must make a CANVAS submission** as stated above, to be assigned a grade. **This applies to both Sprint 1 and Sprint 2!**
Additionally, you will need to submit the link to your **video presentation**. The video **MUST** be **accessible** for everyone **to stream/view** (verify the link before submitting). **Do NOT upload or commit a video file (.mp4, .avi, etc.)** The presentation video does not have to be narrated/showcased by both team members. However, both team members must be aware of each other's code. The video should **first showcase the functionality** of the project (without any technical details, 1-2 minutes), followed by a **short technical review** of the project, detailing the concepts, frameworks, techniques, etc used for this project (2-3 minutes). The **total length** of the video should be around **3-5 minutes**.
Do NOT submit anything via email!

Both team members will receive the **same grade for the project**.

Extra Credit

You can earn **extra credit** if you are able to deploy the Python Flask API and/or the web front-end to the cloud. **Do not attempt this until you have finished the other parts of the project.** There are many options of cloud deployment (AWS, MS Azure) and you are free to choose one that supports the technologies of this project (Python Flask backend + NodeJS/EJS based front-end). S**ubmit your public URL of the deployed project in a text file as part of your repository.**

Basic Grading rubric for the project

Item	Points in Sprint 1	Points in Sprint 2
Remote DB Tables setup	10	-
Code for API	35	-
Correct submission (GitHub commits, CANVAS link)	5	-
Code for UI	-	20
Code for API communication (both ways)	-	25
Correct submission (GitHub, CANVAS, including video)	-	5
Total	**50**	**50**
Extra credit for cloud deployment	**15 points**	**15 points**