Description of the project

It's essential for an institutional investment fund to balance the trade offs between the fund's leverage and liquidity. Fund managers often face the question to find the optimal trade offs in various market scenarios.

This project is built to help tackle the problem by enabling fund managers to simulate trades and observe subsequent impacts to the fund's leverage and liquidity position. The project has the ability to digest the hypothetical trade's impact and alerts the user for any risk limit breaches, which can make it to serve as a risk mitigation tool for traders and portfolio managers before executing real trades that could potentially damage the fund's leverage and liquidity profile.

The project also allows user to create and share their own trade scenarios with others to enhance collaboration among the fund managers.

Some fundamental features of the project are listed below:

- User registration, login, logout
- Allow user to input hypothetical trading activities
- Displaying dashboard charts on liquidity and leverage positions (vs. metric limits) with the impact after hypothetical trading activities
- Enable users to save as private and share scenario as a post with other users to communicate hypothetical trade impact
- Enable users to edit scenarios (i.e. making additional or remove hypothetical trades)
- Alerting background animation effects as fund's leverage and liquidity positions are breached as hypothetical trades been executed
- Enable user to drop a csv file as an input for hypothetical trades
- Mobile responsive and enable user to toggle between day and night color scheme with collapsable navigation menu bars

What's contained in each file created and why made certain design decisions

The "fund" app folder is the main component of the project. Similar to other class projects, many Django files are created under this folder:

- "admin.py" – enables admin user to modify the database
- "models.py" – creates all models required for storing and pulling relevant information
- "urls.py" – hosts all the urls and direct requests to "views.py" accordingly
- "views.py" – includes all the functions from the server side to create and update objects or simply render a particular html file
- "migration" folder – has all the migrations from Django models
- "static" folder with "fund" folder as a sub-folder
    - static css and javascript files are stored under the "fund" folder as "static" is referenced within html templates
    - project icon "balance-scale-solid" is also saved under there so that it's easy to be called in html "head" section
    - couple csv sample trade files are save under here as well which the staff can use as an example for trade entry via dropzone. The file ending with "correct" is an example for correct trade entry and the file ending with "false" is an example of

trades with amounts not netting off which would be rejected with a failed message after uploading.

- "templates" folder with "fund" folder as a sub-folder stores all the html templates as they are referred in "views.py" file
    - "layout.html" is the layout page for "login.html" and "register.html" to allow users to register and login without loading contents for the dashboard page
    - "main_page.html" is the layout page for "scenarios.html", "saved.html" and "shared.html" pages which display the dashboards and scenario posts

There's a "csvs" folder that stores all uploaded csv files. It's created to enable dropzone capability for uploading bulk trades rather than manually entering trades one after another.

Other additional information the staff should know about the project

Once user has registered or logged in, the user will be able to start creating new trades and saving them as new scenarios and sharing them with other colleagues.

Under the "Scenarios" menu page, the main dashboard is displayed but will have no trades and no impacts on fund leverage and liquidity initially. The charts will start reflecting impacts as soon as new trades are entered (either via manual input or bulk trade upload via dropzone csv). When manually entering the trades, user can request for extra trade slots to be submitted as a single trade set. Animated alerts will show up if leverage or liquidity ratios breach their limits (ceiling limit for leverage ratio and floor limit for liquidity ratio)

There are some form control checks in the front-end via javascript as well as some in the back-end in "views.py". The main checks include not allowing users to enter a buy transaction when a debt/liability is selected or vise versa since debts can only be issued or sold upfront (i.e. receive proceeds before eventually need to pay back). Another form check is to ensure buy and sell amounts net off within a trade set (can be multiple trades in one trade set) since rebalancing buys need to be funded from sells transactions. One other important check is to ensure user can only edit (i.e. adding or removing trades, publish or unpublish, changing post description etc.) their own scenario posts.

The "saved" menu page displays all saved scenario posts from the user and the user can revisit or edit the trade by clicking on the "see Details & Edit" anchor.

The "shared" menu page displays all shared scenario posts from all users where the user can like and view the scenario post in detail by clicking on the "see Details & Edit" anchor. However, the user can only save as a new scenario that belongs to him/herself before editing the scenario so that the original scenario post wouldn't be modified by another user.

Please also note that there is no need for "requirements.txt" file as no additional installation other than Django is required to run the project.