Current Project Structure (High-Level)

Front End

To load the project, the following directions include:

1. Go to extensions on a Chrome-Based browser and select developer mode
2. Click "Load unpacked" on the top left corner.
3. Navigate to the folder "FinGPT_For_RCOS/ChatBot-Fin/Extension-ChatBot-Fin/src", select and load it.

Essentially, google chrome extensions are essentially small web applications that can interact with web pages. When going into the developer mode and loading unpacked, you can test the extension program directly from the local file system without uploading to chrome web store.

High Level Understanding of the Components

- The project has a file called **"manifest.json"** under the src directory, which is essentially the **config file** for the extension.
    - This is what makes the extension front end interface pop up when on specified sites (meaning it always tracks what website you are on).
- The popup.js handles the user interface start up when you are on one of the specified sites for scraping.
- The remaining html/CSS files are used for the layout, buttons, aesthetics, and other functionality of the front end of the chrome extension.

Back End

- **The back end of the project uses Python's Django framework.**
    - This **processes the HTTP requests** from the front-end interface.
    - The request **typically contains data in JSON format** (obtained from python web scraping scripts for the data), which the **Python backend interprets, processes, and returns a response** using the OpenAI module and gpt-4o LLM.
    - **Redirection Step**: the views.py file contains functions that handle incoming requests from users (such as retrieving data or responding to user inputs) and return the appropriate response after feeding it into the LLM.
- **Why do we need a server and a backend?**
    - Servers and backends are required to use different APIs, for instance OpenAI in this project, making it possible to fetch, process, and return the data for the user's query.

- **Django is used to host the server and process requests** using the OpenAI API in python.
    - Django listens for HTTP requests from the user on the front end, waiting asynchronously.
- In simple steps (produced by the gpt-4o generic response)
    1. Receive the question via an HTTP request.
    2. Pass the question to the scraping script, which uses the **beautifulsoup** module (which fetches the relevant financial data).
    3. Call the gpt-4o API with both the question and the scraped data.
    4. Return the AI's customized response to the user.

How does this model differ from generic responses from ChatGPT?

- In this project, we are not only using gpt-4o's responses, but we are **also feeding it specific data scraped from Yahoo Finance**.
- In a generic LLM, such as ChatGPT, it cannot fetch live data or process external files unless they are provided as inputs during the conversation.
- This will give more accurate information about stocks in real time when using this chrome extension.