Griffin Fitzpatrick
7/16/2025

Balance Sheet Agent Write Up

The goal of this project was to create an automated program to generate a pro-forma balance sheet for any given company. The use of this is to generate with confidence a more up to date representation of a company's current financials, such as enterprise value, value per share, and shares outstanding. This program used an interlocking set of agents to produce this. In the simplest terms possible, this is how my program works:

1. Retrieve the most recently filed 10-Q or 10-K from the SEC Edgar database
2. Extract the balance sheet and expand it (i.e. uncondensed, unconsolidated balance sheet)
3. Retrieve all filings filed after the 10-Q/10-K's effective date
4. Parse these filings for any material events that affect the balance sheet
5. Reflect these events in the new pro-forma balance sheet

I will now describe the process this program runs through to achieve this for better comprehension of the workflow.

The main workflow is dictated in the script "orchestrator.py". This module coordinates every step from downloading the original filing from the Edgar Cache to applying post-filing updates. The "build_balance_sheet" coroutine begins by fetching the index page via SubmissionPage and deriving document URLS from the Edgar Cache. Next, all HTML files for the filing are uploaded to an OpenAI vector store. This enables later retrieval of specific filing text by the agents. The orchestrator then builds three specialized agents—assets, liabilities, and equity—each using a FileSearchTool connected to the vector store. These agents run concurrently to pull only their respective portion of the balance sheet.

After the sections are gathered, an expander agent examines each section for "Other" or roll-up lines and replaces them with their note-level components. In essence, it creates a "uncondensed, unconsolidated balance sheet". The assembler agent merges the three section tables and verifies the accounting equation. If assets to not equal liabilities plus equity, it flags this by adding an "Imbalance detected" line.

Subsequent filings are scanned for settled events. Two more agents handle this: "update_agent" extracts an ordered list of events that would affect the balance sheet (stock issuances, debt repayments, crypto investments, etc.), "accountant_agent" converts each into a balanced BalanceSheetDelta. The purpose of this is to handle the "balancing" component of the new updates on a case-by-case basis, ensuring that, independently, if one line item is updated, what equivalent line item needs to be adjusted to balance that specific update.

The deltas are then applied sequentially to the original balance sheet. The helper “apply_updates” verifies that each delta balances; any imbalance is recorded on bs.update_errors. The coroutine returns a tuple: the original FullBalanceSheet extracted from the filing, and the updated one with deltas applied. This is the main logic of the program.

Now we will dive into command-line and testing utilities. Build_balance_sheet.py exposes a CLI wrapper and a synchronous helper get_balance_sheet so callers don’t need to manage asyncio themselves. The scripts docstring gives many usage examples. Test.py provides a simple loop over sample filings. Pretty.py renders the original and updated balance sheets side by side using the tabulate library. The file settings.py loads the OpenAI API key from a shared keys.json and exposes a singleton client for all agents.

Overall, the project automates extraction of the latest balance sheet, expands summarized notes, and applies subsequent updates using a series of specialized agents. The orchestrator coordinates these steps, ensuring the output remains balanced and producing both the original and updated sheets for further analysis. The supporting modules provide command-line tools, pretty-printing helpers, and shared configuration so the workflow can run end-to-end with minimal manual intervention.