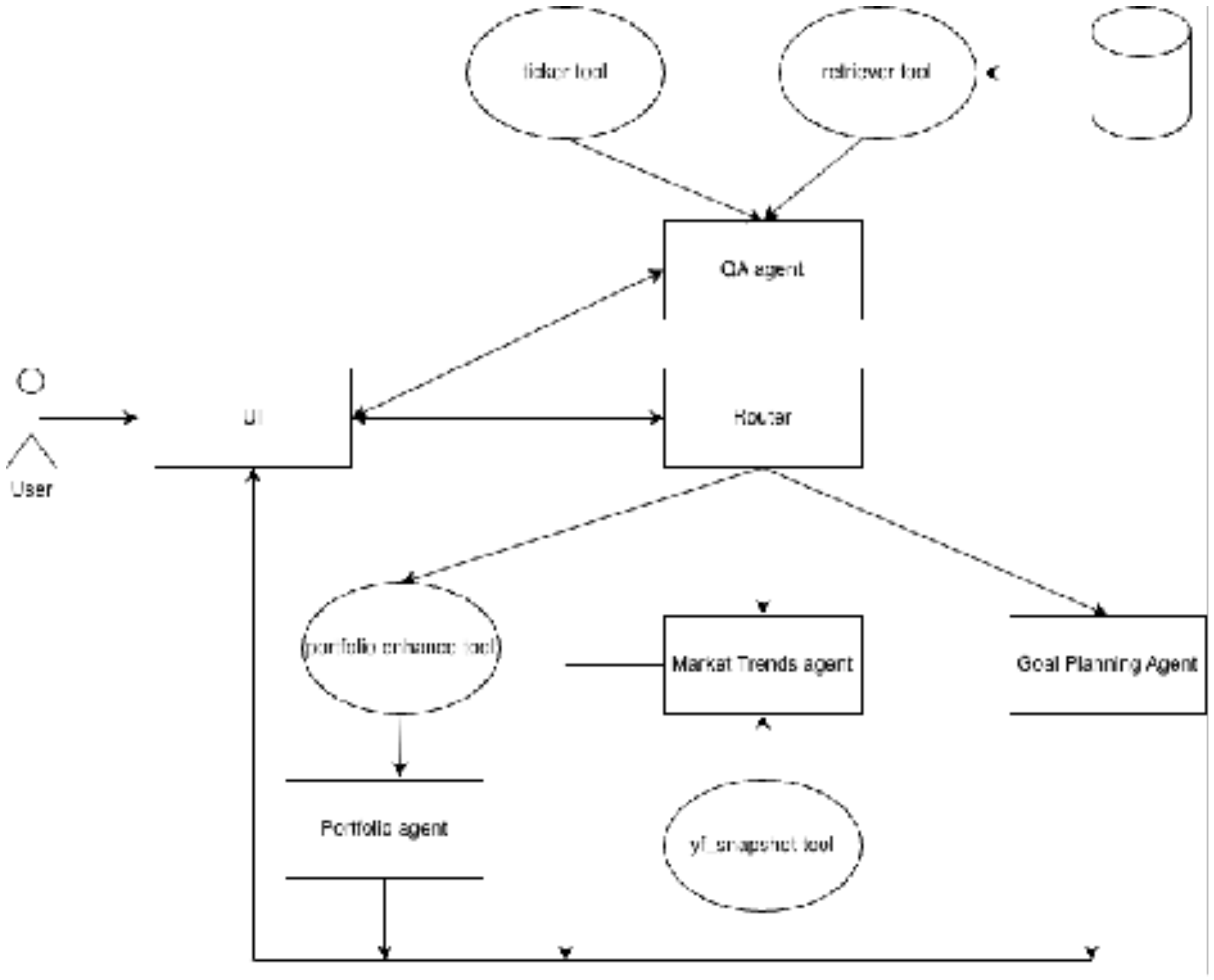
Thursday, November 13, 2025

Finance AI Assistant

- Dev Jyoti Behera

Architecture



Components:

What's not show in the above diagram is the LLM. The LLM used is **gemini-2.0-flash**.

1. UI (Streamlit based)
2. Agents:
    1. QA agent: for QA related to finance topics
    2. Market Trends agent: for getting market trends for a stock ticker
    3. Portfolio insights agent: given a JSON portfolio with tickers and amount for each ticker, returns portfolio evaluation.
    4. Goal Planning agent: given a user's financial details and a financial goal details, evaluates the plan and makes recommendations.
3. Tools used:

    1. retriever: for getting documents from ChromaDB vector store.

    2. Ticker: get basic stock data for ticker

    3. yf_snapshot: get stock data with history and news from yfinance.

    4. Portfolio enhance: computes total portfolio value, allocation weights, and enhances ticker with related info.

Agents

There are 4 agents built using Langchain/Langgraph.

3 of them (goals planning, market trends and portfolio agent) are part of a multi-agent system shown above. The correspond to goal_planning.py, market_trends.py, portfolio_insights.py.

The 4th one is the QA agent which uses RAG to retrieve contextual information from a vector store in ChromaDB. This is qa_agent_test.py.

Earlier the QA agent was part of the multi-agent graph. However, I faced issues with multi-turn chat which I attributed (correctly or incorrectly) to the single messages variable being used. Perhaps with a better understanding of placeholders in Langchain this could have been solved, but, in the interest of time, I went ahead and made this agent separate.

Since each agent has a separate concern, there is no inter-agent communication.

The separation between portfolio_enhance tool and portfolio agent was necessary because the portfolio agent outputs a structured json object. With a structured json object, I faced issue with binding tools. I read online that bind_tools doesn't work well with it, therefore this design.

The router (workflow.py) routes using a state variable called "context" that is passed from the UI. The other agents also use tools (yfinance based ticker info extraction).

The QA agent use the retrieval tool to follow RAG and retrieve documents from a vector store in ChromaDB. ChromaDB was chosen in the interest of time.

The other tool that's show for the QA agent allows the QA agent to show stock prices, given a ticker symbol.

Tools

- Retriever (retrieval_tool.py):
  - Loads from ChromaDB
  - 5 max documents per query which can be adjusted.
  - Documents are not segmented by topic, however source is present for source attribution in the UI.
- get_ticker_info (fin_tools.py)
  - Returns basic info about a ticker
- yf_snapshot (fin_tools.py)
  - Returns news, stock info, etc about a ticker.
- enhance_portfolio_data (fin_tools.py)
  - Given a basic portfolio, enhances it with stock ticker sector, weightage and total portfolio value.

RAG / KB

- Focussed on preparing a KB with 50 basic articles.
- Since the number of articles was low and time was limited, I decided to forgo scraping in favor of using the document loaders provided in Langchain.
- Manually created a list of URLs from Investopedia and Bogleheads wiki. The urls are in src/rag/urls.py
- Bogleheads wiki blocked my IP. So, I manually downloaded the files (The urls are in src/kb/boglehead) and used a specific loader to load them into ChromaDB.
- For the investopedia urls, I gave a 10s sleep before every url call with WebBaseLoader.
- One issue was that the Gemini embeddings timed out frequently. This was resolved by batching the documents. 1 URL at a time worked best obviously.
- The vector store itself was created using src/rag/vector_store.py

Errors / Performance Improvements

- Frequently, got 429 Resource Exhausted errors. I think this was because of the large contexts in some cases, with the yfinance data containing a lot of information. I saw it mostly with market_trends.py. Tried to filter the tool output from yfinance to separate what was needed for the LLM context and what was needed for the UI (stock price history, news links)
- Limited vector store retrieval to 5 documents and significantly decreased the UI latency.
- Caching: implemented caching of yfinance requests as well as agent outputs. Caches are TTL caches, mostly set to 1 hour expiry. Keys are based on the user inputs, using string joining and hashing.
- Exponential backoff on retries: With the gemini model, there was automatic exponential backoff based retry logic.