Build a Wealth Management Agentic AI Chatbot with MS Fabric Project Overview

Overview

This project marks the second phase of the **Microsoft Fabric Project to Build a Financial Reporting Agent**, introducing **Agentic AI** to enhance financial data analysis, risk assessment, and investment planning. While the first phase of the project focused on centralizing wealth management data using Microsoft Fabric, this phase transforms the system into an intelligent financial assistant that can autonomously analyze data, generate investment strategies, and provide dynamic risk assessments. By incorporating Agentic AI, the system dynamically selects the best financial tools and executes complex multi-step workflows based on user intent.

Agentic AI represents the next step in AI evolution, where models are not just reactive but proactive and adaptive. Unlike traditional AI, which simply processes inputs and returns outputs, Agentic AI analyzes context, determines the optimal course of action, and executes tasks dynamically. The Agentic AI system operates by breaking down high-level financial queries into smaller, modular tasks, executed independently by specialized AI-powered tools. These tools include stock position analysis, portfolio risk assessment, investment strategy generation, and risk mitigation modeling, each of which functions as a standalone agent tool. The system employs reasoning to determine which tools to invoke, ensuring the most relevant computations are performed based on user queries.

This allows our **risk assessment agent** to intelligently plan investment strategies, mitigate risks, and refine recommendations based on the data, making financial decision-making more efficient and personalized. To enable contextual intelligence, the project implements **memory management** using chromadb vector stores, allowing the agent to retain past financial conversations and user preferences. This ensures that users receive personalized, long-term investment guidance rather than isolated, one-time responses.

Prerequisite Project: Kindly ensure the completion of the following projects before proceeding with this project:

- **LLM Project for building and fine-tuning a large language model**
- **Microsoft Fabric Project to Build a Financial Reporting Agent**

Prerequisites: Basics of LLMs, Vector Databases, Prompt Engineering, Python and Streamlit, Basics of Azure Cloud, MS Fabric

Note:

- Utilizing Azure services for this project may result in charges; it is essential to thoroughly review the Azure documentation to understand the pricing structure and potential costs associated with different resources and usage patterns.

- Utilizing Microsoft Fabric is chargeable, with costs depending on the services and resources used. However, a **free trial of 60 days** is available, allowing users to explore its features without initial charges.

Aim

The aim of this project is to build an **Agentic AI-powered Risk Assessment Agent** that dynamically analyzes financial data, evaluates investment risks, and provides personalized risk mitigation strategies. This system autonomously selects and executes financial analysis tools, enabling real-time, goal-driven decision-making for wealth management.

Data Description

The dataset includes comprehensive financial information crucial for wealth management:

1. **Wealth Assets Data**: Details of client portfolios, including asset classes such as stocks, bonds, real estate, and alternative investments.
2. **Financial Records**: Transaction histories, and investment portfolios for a complete financial overview.

Tech Stack

- ➔ **Language**: Python 3.10
- ➔ **Libraries**: pandas, numpy, Azure, langchain, Open AI, Flask
- ➔ **Cloud Platform Components**: Microsoft Fabric (OneLake, LakeHouse, Fabrics Datawarehouse)
- ➔ **Model**: Retrieval-Augmented Generation (RAG) with GPT
- ➔ **Cloud Platform**: Azure
- ➔ **API Framework**: Flask API with HTML UI

Approach

1. Defining the Goal and Task Decomposition

- Establish the core objective: Assess financial risks and provide mitigation strategies.
- Break down the goal into smaller, manageable tasks:
    - Analyze stock positions and portfolio distribution.
    - Identify and quantify risk exposure (e.g., market volatility, diversification gaps).
    - Generate tailored risk mitigation strategies.
    - Refine recommendations based on user feedback.

2. **Building the Agentic AI Framework**

- Develop an AI agent that autonomously selects and executes financial analysis tools.
- Use Chain of Thought (CoT) reasoning to dynamically determine the best approach for risk assessment.
- Implement task selection to optimize decision-making.

3. **Creating Modular Financial Analysis Tools**

- Stock Position Analysis Tool:
    - Retrieves user's portfolio composition from the database.
    - Assesses market volatility and stock performance.
- Risk Calculation Tool:
    - Calculates portfolio exposure, diversification levels, and historical risk trends.
    - Evaluates risk-to-reward ratios for different asset classes.
- Risk Mitigation Suggestion Tool:
    - Suggests portfolio reallocation, hedging strategies, or alternative investments.
    - Adapts recommendations based on user goals (e.g., conservative vs. aggressive investor).
- Feedback Loop & Refinement Tool:
    - Captures user responses to previous recommendations.
    - Refines future risk assessments based on historical interactions.

4. **Implementing Memory Management & Context Retention**

- Store user financial history and past recommendations using a vector database.
- Enable agent to recall previous interactions and maintain personalized, context-aware conversations.

5. **Developing the Decision-Making Logic**

- Implement Intent Detection to classify queries into:

    - Risk Analysis (e.g., "How risky is my portfolio?").
    - Investment Planning (e.g., "What are safe investments for me?").
    - Unknown Queries (fallback mechanism for unrelated questions).
- Use the agent to decide which tool(s) should dynamically execute.

6. Create a Flask-Based API

- Develop an API endpoint that accepts user queries and routes them to the AI agent.
- Pass user inputs, retrieve responses from AI tools, and store conversation history.
- Integrate a HTML frontend chatbot to interact with the risk assessment system in real time.

Modular code overview:

Once you unzip the modular_code.zip file, you can find the following:

```
├─ Agent
│  ├─ agent.py
│  ├─ intent.py
│  ├─ memory.py
│  ├─ tools.py
├─ app.py
├─ connection.py
├─ CreateDataWarehouse
│  ├─ CreatingTables.sql
│  └─ InsertToSQL.py
├─ FinancialGoals
│  └─ RAGToSQL
│     ├─ FabricsRAG.py
│     ├─ Helper
│     │  ├─ Credentials.py
│     │  ├─ FabricsConnection.py
│     │  ├─ VannaObject.py
│     ├─ InferenceRAG.py
│     ├─ TrainingRAG-Artifact
│     │  ├─ Documentation.txt
│     │  └─ Tables.json
│     ├─ training_summary.csv
│     ├─ TrainRAG.py
│     ├─ VisualizeRAG.py
├─ readme.md
├─ requirements.txt
└─ templates
   └─ index.html
```

Here is a brief information on the files:

- **`Agent/`**: Contains core chatbot logic, including `agent.py` for creating the AI agent, `intent.py` for intent detection, `memory.py` for user memory management, and `tools.py` for additional utilities.
- **`CreateDataWarehouse/`**: Manages data storage, with `CreatingTables.sql` defining schema and `InsertToSQL.py` handling data insertion.
- **`FinancialGoals/RAGToSQL/`**: Implements a Retrieval-Augmented Generation (RAG) pipeline, with `TrainRAG.py` for model training, `InferenceRAG.py` for retrieval, and `FabricsRAG.py` integrating with a database. The `Helper/` folder contains connection utilities.
- **`app.py` & `connection.py`**: `app.py` is the main backend server handling chatbot interactions, while `connection.py` manages database connections.
- It includes a `Readme.md` file for project documentation and a `requirements.txt` file listing all necessary dependencies.
  Kindly follow all the instructions for running the code from Readme.md file.

Project Takeaways

1. Understand how Agentic AI enables autonomous decision-making using task selection instead of predefined workflows
2. Use LangChain's agent framework to break down complex financial risk analysis tasks into modular, executable components
3. Implement LangChain tools and agentic execution chains to dynamically invoke the right financial analysis tool based on user input
4. Store and retrieve structured and unstructured financial data using Microsoft Fabric's OneLake, Lakehouse, and Data Warehouse components
5. Implement ChromaDB vector store to retain conversation history for personalized, context-aware recommendations
6. Build a Flask API that acts as the agent execution engine, dynamically processing queries and returning risk assessment results
7. Design custom Python functions for portfolio risk evaluation, asset diversification analysis

8. Learn how to configure LangChain's agent and tool structure to orchestrate multi-step AI-driven financial processes
9. Build an HTML chatbot UI that interacts with the Flask API for risk assessment queries
10. Use long-term memory modules in LangChain to retain user-specific financial preferences

Cost Breakdown (As of January 17, 2025)

The cost might have been updated, it is recommended to go through the following pricing pages to come up with updated guesstimates.

Open AI Pricing https://openai.com/api/pricing/
Azure App service Pricing - Pricing Page
Azure Blob Storage - https://azure.microsoft.com/en-us/pricing/details/storage/blobs/

Component	Description	Usage (Estimated for Experimentation)	Estimated Cost
API Calls	GPT-4o API for RAG-based financial insights and text-to-SQL queries using Vanna AI	100 queries (Avg. 2,000 tokens per query)- 200,000 tokens (input)- 100,000 tokens (output)- GPT-4o: $0.50 for input, $1.00 for output	$1.50
Azure SQL	Storing financial data for analysis and querying	50 GB of data storage- Pricing: $0.10 per GB	$5.00
Microsoft Fabric	Data consolidation and reporting using OneLake	100 GB data storage, light querying- Free for 60 days (64 CUs included)- Post free tier: $0.023 per GB for storage- Pay-as-you-go: $262.80/month for F2 SKU (2 CUs)	Free (during trial), ~$2.30 (after trial for storage), or $262.80/month for compute

Network Usage	Data transfer for retrieving and processing financial data	50 GB egress- Azure Data Egress: $0.09 per GB	$4.50

Total Estimated Cost for Experimentation Phase:

- **Using Free Tiers:** ~$11.00
- **Post Free Tiers:** ~$13.80 + $262.80/month for Microsoft Fabric compute

Recommendation:

- Leverage the Microsoft Fabric free trial for initial testing and evaluation.
- Monitor API token usage to optimize query execution costs.
- Consider long-term data storage needs and plan for potential Azure SQL scaling.
- Utilize Vanna AI's free tier strategically for efficient text-to-SQL conversions.