

## General Requirements

### Video Assessment

- Please walk us through your **approach to the task**, clearly explaining your **thought process**.
- Highlight any **challenges** you encountered and discuss **alternative approaches** you considered.
- Explain how you addressed these challenges and the **reasoning behind your decisions**.
- Please **film yourself from start to finish**.

### Video Guidelines

- Preferred upload method: **Google Drive link**
- Audio must be **clearly understandable**
- Video length should be **under 60 minutes**

## Development Environment & Tools

- **IDE:** Please use an **Agentic IDE** such as **Windsurf** or **Cursor**. We want to see how you leverage AI to generate high-quality code efficiently.
- **LLM:** Use either a **local LLM instance** or the **Google Gemini Free Tier API**.
- **Node.js:** The entire system must be built using **Node.js**.
- **Docker:** The **Weaviate vector database** must be containerized and run in **Docker**.

### Additional Libraries

- **Weaviate JS Client** – for interacting with the Weaviate vector database
- **LangGraph** – for building the agent hierarchy
- **LangChain** – for LLM communication and abstraction

## Part 1: Weaviate Vector Database Setup (Multi-Tenancy)

### Requirements

- Set up a **Weaviate vector database** using **Docker**.

- Create a **multi-tenant schema** with the following fields:
  - fileId (string – not vectorized or index-searchable): Identifier for each file
  - question (text): The question being asked
  - answer (text): The answer to the question

## Data Insertion

- Using the **Weaviate JavaScript client**, insert **at least three fictional entries** into the vector database.
- You **do not need to provide vectors** for the entries.

## Part 2: LangGraph Hierarchical Agent Setup

### Agent Architecture

Set up a **LangGraph agent hierarchy** consisting of:

- **Delegating Agent**
  - Determines the flow based on the user's query
  - Decides whether to:
    - Call the **Chart.js tool**
    - Call the **RAG agent**
    - Or provide a direct response to the user
- **Chart.js Tool**
  - A mocked tool that generates a **Chart.js configuration** based on input data
- **RAG Agent**
  - A retrieval-augmented generation agent that:
    - Queries the Weaviate vector database
    - Retrieves relevant chunks (fileIds)
    - Returns both the **answer** and **references**

### Delegating Agent Requirements

- The delegating agent receives a **user query** and decides whether:
  - To interact with the **Chart.js tool** and return a chart configuration
  - To query the **RAG agent** for relevant information
  - Or to answer the user **directly**
- The agent must be capable of calling:
  - Both the **Chart.js tool** and **RAG agent**
  - Either **simultaneously or sequentially**, depending on the user request

## Tool Behavior Details

### Chart.js Tool

- Mocks generating a Chart.js configuration
- Can return a **fixed mock configuration** for simplicity

### RAG Agent

- Uses the **Weaviate vector database** to answer questions
- Fetches relevant chunks and returns:
  - The answer
  - The corresponding fileIds
- If the embedding model is unavailable, the agent should use the **fetchObjects API** to retrieve data

## Delegating Agent Response Format

Each response from the delegating agent must always include:

- The **answer text**
- **All references used**, provided in a **separate object**
- The **fileIds**, if the response is based on RAG or database retrieval
- The **Chart.js configuration**, if the response involves chart creation

## Integration & Multi-Tool Handling

- The delegating agent must correctly manage:
  - **Parallel or sequential execution** of multiple tools
- If a request requires **both charting and data retrieval**, the agent should:
  - Call the **Chart.js tool** and **RAG agent**
  - Combine and return both outputs in the **final response**