Backend Developer Assessment

Company: Edwid Tech PVT LTD
Role: Backend Developer
Assignment: Scalable RAG API for News Intelligence

1. Overview

This is an assessment for the **Backend Developer** role at **Edwid Tech PVT LTD**. You are required to architect and build a robust, production-ready REST API service that answers queries over a news corpus using a Retrieval-Augmented Generation (RAG) pipeline.

Unlike a full-stack assignment, the focus here is strictly on **API design, data modeling, code architecture, and system performance.**

2. Objectives

A. RAG Pipeline & Ingestion

- **Ingestion Engine:** Create a script or API endpoint to trigger the scraping/ingestion of ~50 news articles (from RSS feeds or a mock JSON file).
- **Vector Processing:** Generate embeddings using **Jina Embeddings** (or any open-source alternative).
  - Store embeddings in a **Vector Database** (Qdrant, Chroma, or Faiss).
- **LLM Integration:** Retrieve the top-k relevant passages for a user query and pass them to the **Gemini API** to generate a contextual answer.

B. API & System Architecture

- **Framework:** Node.js (Express).
- **Endpoints:**
  - `POST /ingest` - Triggers the document processing pipeline.
  - `POST /chat` - Accepts `sessionId` and `query`. Returns the streaming response or final text.
  - `GET /history/:sessionId` - Fetches past Q&A for the specific session.
  - `DELETE /history/:sessionId` - Clears the session data.
- **Database & Persistence:**
  - **Vector DB:** For semantic search.
  - **Redis:** Strictly for caching conversation context (short-term memory) to maintain the "chat" feel.
  - **SQL (Postgres/MySQL): Mandatory.** You must store structured logs of every interaction (*Timestamp, SessionID, UserQuery, LLMResponse, ResponseTime*) for analytics purposes.

C. Backend Engineering Standards

- **Documentation:** You must provide a **Postman Collection**.
- **Containerization:** The application (API + DBs) should be runnable via `docker-compose`.
- **Error Handling:** Implement standardized error responses (e.g., 400 vs 500, rigorous validation of input payloads).

3. Tech Stack Requirements

Please choose your specific tools within these categories and justify your choice in the README.

- **Runtime:** Node.js
- **Embeddings:** Jina AI / HuggingFace (can use any)
- **Vector DB:** Qdrant / ChromaDB / Pinecone (can use any)
- **LLM API:** Google Gemini (Free Tier)(can use any)
- **Backend Framework:** Node.js (Express)
- **Caching:** Redis
- **Primary Database:** PostgreSQL or MySQL
- **DevOps:** Docker & Docker Compose

4. Deliverables

Please share your work at **hr@edwidtech.com** with the following:

1. **Git Repositories:** One public monorepo with clean, modular code structure (e.g., MVC, Clean Architecture).
   - A comprehensive `README.md` explaining how to run the project locally.
2. **Docker Setup:** A `docker-compose.yml` file that spins up your Node API, Redis, and Vector DB (if self-hosted) with a single command.
3. **API Documentation:** A link to the Swagger UI or an exported Postman JSON file.
4. **Demo Video:** A video (mp4 or unlisted link) **under 5 minutes** showing: Starting the application via Docker. Sending queries via Postman/cURL and observing Gemini responses. Viewing the structured logs in the SQL database.
5. **Live Deployment (Optional):** A hosted, publicly accessible link (using Render/Railway/AWS) where we can test the API.

5. Evaluation Criteria

Area	Weight	Description
System Architecture & Code Quality	**40%**	Folder structure, separation of concerns (Services vs Controllers), and type safety.
RAG Implementation	**30%**	Accuracy of retrieval, handling of embeddings, and prompt engineering logic.
API Design & Documentation	**20%**	RESTful standards, input validation, and clarity of Swagger/Postman docs.
DevOps & Persistence	**10%**	Docker implementation and correct usage of SQL vs Redis.

6. Bonus Points (To stand out)

- **Rate Limiting:** Implement middleware to prevent API abuse.
- **Queueing:** Use a queue (like BullMQ) for the ingestion process so the API doesn't hang while scraping articles.

Good luck! We look forward to reviewing your architecture, code, and documentation.