Assignment: Build a RAG-Powered Chatbot for News Websites

1. Overview

This is an assignment for the role of Full Stack Developer at Voosh.

You are required to create a simple full-stack chatbot that answers queries over a news corpus using a Retrieval-Augmented Generation (RAG) pipeline. Every new user should be a new session so create a session identifier.

---

2. Objectives

1. **RAG Pipeline**
    - Ingest ~50 news articles (e.g. RSS feed or scraped HTML).
    - Embed with Jina Embeddings (free tier) or any other open source embeddings you like .
    - Store embeddings in a vector store of your choice (Qdrant, Chroma, faiss, etc.).
    - Retrieve top-k passages for each query, then call Gemini API for final answer.
2. **Back-End**
    - Build a REST API (Node.js – Express).
    - Use API / Socket based for chat. Also Implement to fetch session's history and clear session
    - **Storage:**
        - **Redis** for in-memory chat history (per session).
        - Persist final transcripts optionally in a SQL database (MySQL/Postgres)(Optional).
3. **Front-End**
    - React + SCSS with:
        1. **Chat screen**:
            - Displays past messages.
            - Input box for new messages.
            - Streaming bot responses (if possible) or typed-out reply.
            - button to reset the session
4. **Caching & Performance**
    - Cache session history and conversations in In-memory database
    - Show in your README how you'd configure TTLs or cache warming.

---

3. Tech Stack (choose & justify)

- **Embeddings**
- **Vector DB**
- **LLM API:** Google Gemini (free trial)
- **Backend:** Node.js (Express)
- **Cache & Sessions:** Redis (in-memory) or any other in-memory database you like
- **Database (optional):** MySQL or Postgres
- **Frontend:** React + SCSS

---

4. Deliverables

Please share your work at richa@voosh.in with-

1. **List of Tech Stack Used**
2. **Git Repositories**
   - Two public repos (frontend & backend) with full code and a clear `README.md` in each.
3. **Demo Video**
   - A video (mp4 or hosted via unlisted link) showing:
     - Starting the frontend.
     - Sending queries and observing Gemini responses.
     - Viewing and resetting chat history of session.
4. **Code Walkthrough**
   - Written or video explanation of the end-to-end flow, covering:
     - How embeddings are created, indexed, and stored.
     - How Redis caching & session history works.
     - How the frontend calls API/Socket and handles responses.
     - Any noteworthy design decisions and potential improvements.
5. **Live Deployment**
   - A hosted, publicly accessible link (using any free hosting service) where we can test the chatbot.

---

5. Evaluation Criteria

Area	Weight
End-to-End Correctness	35%
Code Quality	30%
System Design & Caching	20%
Frontend UX & Demo	5%
Hosting	10%

Good luck!

We look forward to reviewing your working demo, code repos, and detailed flow explanation and testing it.

Note: Feel free to leverage any existing code—from open-source libraries, GitHub repositories, or AI-generated snippets—provided you fully understand and can explain it.

Resources & References

- News ingestion example: https://github.com/fhamborg/news-please
- Reuters sitemaps: https://www.reuters.com/arc/outboundfeeds/sitemap-index/?outputType=xml
- Jina Embeddings: https://jina.ai/embeddings
- Google AI Studio API keys: https://aistudio.google.com/apikey
- Qdrant quickstart: https://qdrant.tech/documentation/quickstart/
- Pinecone quickstart: https://docs.pinecone.io/guides/get-started/quickstart
- Render.com hosting: https://render.com/
- Redis Python client: https://github.com/redis/redis-py
- For frontend development you can use - v0, bolt.new or any other LLM.