RAG-Powered Chatbot for News Websites – Project Documentation

1. Overview

This project is a **Retrieval-Augmented Generation (RAG)** powered chatbot designed for news websites. Users can ask natural language questions, retrieve relevant information from a news corpus (~50 articles), and get responses generated via **Google Gemini API**.

Key features:

- Unique sessions per user.
- Chat history cached in **Redis** (temporary).
- Full transcripts optionally stored in **MySQL**.

2. Objectives

- Implement a RAG pipeline to answer questions from a news dataset.
- Provide a session-based chat interface.
- Cache responses and ensure efficient performance.
- Deploy both frontend and backend publicly.

3. Tech Stack

Component	Technology
Embeddings	**Jina Embeddings**

Vector DB	**Pinecone API**
LLM API	**Google Gemini**
Backend	**Node.js (Express) + Socket.io**
Cache & Sessions	**Redis**
Database	**MySQL** (transcripts), **PostgreSQL** (deployment)
Frontend	**React + SCSS**
Hosting	**Render** (frontend & backend)

4. System Architecture (High-Level Flow)

4.1 News Ingestion

- ~50 news articles ingested **manually via RSS feed** initially.
- Scheduled **daily updates via GitHub Actions**.
- Articles split into small text chunks.
- Chunks embedded using **Jina Embeddings**.
- Stored in **Pinecone Vector DB** with metadata.

4.2 User Interaction

- User opens chatbot → **new session ID generated**.
- Chat history stored in **Redis** for fast access.

4.3 Query Processing (RAG Pipeline)

1. User query embedded via Jina.
2. **Top-k matches retrieved from Pinecone**.
3. Relevant article chunks selected.
4. Augmented prompt built with retrieved context.
5. Prompt sent to **Google Gemini API**.

4.4 Response Handling

- Gemini generates final answer.

- Response returned to frontend.
- Interaction stored in **Redis (session cache)** and **MySQL (transcripts)**.

4.5 Frontend (React)

- Displays chat messages.
- Supports live responses via REST or **Socket.io streaming**.
- Reset session button clears Redis history.

5. Data Flow – Example

User Query: "What's happening in India's stock market today?"

Backend Steps:

1. Embed query → retrieve top 3 article chunks from Pinecone.
2. Send context + query → Gemini API.
3. Gemini Response: "The Indian stock market rose today, led by banking stocks, with Nifty gaining 2%."
4. Store response in Redis + MySQL.

Reset Session: Clears Redis history.

6. Session & Storage Design

Redis:

- Stores session chat history (fast access).
- TTL: 24 hours to auto-clear inactive sessions.

MySQL:

- Stores permanent transcripts.

7. Deployment Plan

Frontend & Backend:

- Both deployed on **Render** .

Databases:

- Redis & Pinecone: managed services or hosted.
- MySQL: managed or hosted service.

8. Caching & Performance

- Redis TTL: 24 hours.
- Cache warming: preload embeddings at startup.
- Streaming responses via **Socket.io** for better UX.

9. Deliverables

- Tech Stack documentation.
- Two public GitHub repos (Frontend & Backend).
- Demo video (chat, history, reset).
- Code walkthrough (README/video).
- Live deployment links.

10. Evaluation Criteria

- End-to-End correctness.
- Code quality.
- System design & caching efficiency.
- Frontend UX.
- Hosting & deployment readiness.