Software Requirement Specification (SRS)

Project: RAG-Powered Chatbot for News Websites
Prepared For: Voosh – Full Stack Developer Assignment
Prepared By: Praveen N

1. Introduction

1.1 Purpose

This project is a **Retrieval-Augmented Generation (RAG)** powered chatbot for news websites. It answers user queries by retrieving relevant content from a news corpus and generating responses using the **Google Gemini API**.

Each user has a unique session, with chat history cached in **Redis** and transcripts optionally stored in **MySQL**.

1.2 Scope

- Users interact with a web-based frontend (**React + SCSS**).
- Chatbot retrieves passages from a news dataset (~50 articles) stored in **Qdrant Vector DB**.
- Embeddings created with **Jina Embeddings**.
- Responses generated via **Google Gemini API**.
- Session history cached in Redis; transcripts optionally stored in MySQL.
- System deployed on **Render** (backend) and **Render** (frontend).

1.3 Acronyms

- **RAG** – Retrieval-Augmented Generation
- **LLM** – Large Language Model
- **Embedding** – Numeric vector representation of text

- **Vector DB** – Database optimized for similarity search on embeddings
- **Redis** – In-memory data store for sessions and caching
- **MySQL** – Relational database for permanent transcript storage

2. Overall Description

2.1 Product Perspective

The chatbot integrates multiple technologies:

- **Frontend (React)** → User-facing interface
- **Backend (Node.js + Express)** → API layer
- **Redis** → Session & caching layer
- **Pinecone**→ Vector DB for embeddings
- **MySQL** → Permanent transcript storage
- **Google Gemini API** → Natural language response generation

2.2 Product Features

Chat Interface:

- Send/receive messages
- Reset session button

RAG Pipeline:

- Embed news corpus using Jina
- Store in Pinecone
- Retrieve top-k passages per query
- Augment prompt for Gemini API
- Return natural language answer

Session Management:

- Unique session per user
- Session history in Redis with TTL
- Reset session clears Redis history

Transcript Storage:

- Store session and messages in MySQL (optional)

Deployment:

- Frontend → Render
- Backend → Render

2.3 User Characteristics

- **End Users:** News readers querying the chatbot
- **Developers/Reviewers:** Evaluators testing functionality
- Users assumed to have basic internet access

2.4 Constraints

- Must use Google Gemini API (free trial)
- Must ingest ~50 news articles
- Session handling via Redis
- Transcripts stored in MySQL
- Deployable on free-tier hosting

2.5 Assumptions & Dependencies

- News articles available via RSS .
- Internet required for Gemini API
- Redis, Pinecone, and MySQL accessible via hosted service

3. System Features

3.1 Session Management

- No user authentication
- Each user gets a new session ID
- Session stored in Redis with TTL (~24h)

3.2 Chat Functionality

- User sends query → embedded → relevant passages retrieved → answered by Gemini
- Responses optionally streamed via **Socket.io**

3.3 Session Reset

- Reset button clears Redis history
- Creates a fresh session

3.4 Transcript Management

- Session data and messages optionally stored in MySQL for permanent records

3.5 Admin Features (Future Scope)

- View past transcripts
- Monitor Redis usage

4. External Interface Requirements

4.1 User Interfaces

- Chat window to display messages
- Input box for queries
- Reset session button
- Responsive SCSS design

4.2 API Interfaces

- `POST /chat` → Send user message
- `GET /:sessionId/history` → Fetch session history
- `DELETE /:sessionId/reset` → Reset session

4.3 Hardware Interfaces

- Standard client (PC/Mobile) with internet browser

4.4 Software Interfaces

- Google Gemini API
- Jina Embeddings API
- Redis
- Pinecone API
- MySQL

5. Non-Functional Requirements

5.1 Performance

- Query response < 3 seconds (excluding Gemini latency)
- Redis lookup < 50ms

5.2 Reliability

- Redis sessions expire gracefully
- MySQL provides durable transcript storage

5.3 Scalability

- Supports multiple concurrent users (session-based)

5.4 Security

- No authentication required
- API keys stored securely in environment variables

5.5 Maintainability

- Frontend & backend in separate repositories

- Clear documentation in README

6. Storage & Caching

Redis:

- Stores session history for fast access
- TTL ~24 hours

MySQL:

- Permanent transcript storage (optional)