RAG Chatbot (Retrieval-Augmented Generation)

A modern, full-stack chatbot that allows users to upload PDF documents and ask questions about their content using Retrieval-Augmented Generation (RAG) with Google Gemini and ChromaDB. The project supports both local and AWS/LocalStack deployments, and features a beautiful frontend for seamless user interaction.

Features

- **PDF Upload:** Upload PDF files and extract their text for question answering.
- **RAG Pipeline:** Uses Google Gemini LLM and ChromaDB for vector storage and retrieval.
- **Chat Memory:** Remembers previous questions and answers for context-aware conversations.
- **AWS/LocalStack Integration:** Optionally uploads PDFs to S3 and stores metadata/metrics in DynamoDB.
- **Modern Frontend:** Responsive, user-friendly interface for uploading, chatting, and viewing history.
- **Rate Limiting:** Prevents abuse with per-IP request limits.

Project Structure

```
CHATBOT_RAG-MODEL/
  chatbot_rag/
    app/
      main.py            # FastAPI backend
      rag_pipeline.py    # RAG logic and vector store
      utils.py           # PDF text extraction
      ...
    aws_service/
      dynamo_handler.py  # DynamoDB logic
      s3_handler.py      # S3 upload logic
      ...
    frontend/
      index.html         # Main frontend UI
      style.css          # Styles (if separated)
    ...
  README.md              # This file
```

Setup Instructions

1. **Clone the Repository**

```
git clone <repo-url>
cd CHATBOT_RAG-MODEL
```

2. **Install Python Dependencies**

```
pip install -r chatbot_rag/requirements.txt
```

3. **Environment Variables**

Create a `.env` file in `chatbot_rag/app/` with your Google API key and (optionally) AWS/LocalStack credentials:

```
GOOGLE_API_KEY=your_google_api_key
AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=test
AWS_SECRET_ACCESS_KEY=test
ENDPOINT_URL=http://localhost:4566
S3_BUCKET_NAME=pdf-storage-bucket
```

4. **Start LocalStack (for AWS emulation)**

```
localstack start -d
```

5. **Create S3 Bucket and DynamoDB Tables**

```
awslocal s3 mb s3://pdf-storage-bucket
awslocal dynamodb create-table \
  --table-name PDF_Metadata \
  --attribute-definitions AttributeName=filename,AttributeType=S AttributeName=user_id,AttributeType=S \
  --key-schema AttributeName=filename,KeyType=HASH AttributeName=user_id,KeyType=RANGE \
  --billing-mode PAY_PER_REQUEST
awslocal dynamodb create-table \
  --table-name LLMMetrics \
  --attribute-definitions AttributeName=query_id,AttributeType=S AttributeName=timestamp,AttributeType=S \
  --key-schema AttributeName=query_id,KeyType=HASH AttributeName=timestamp,KeyType=RANGE \
  --billing-mode PAY_PER_REQUEST
```

6. **Run the Backend**

```
cd chatbot_rag/app
uvicorn main:app --reload
```

7. **Open the Frontend**

Open `chatbot_rag/frontend/index.html` in your browser.

Usage

1. **Upload a PDF** using the sidebar.
2. **Wait for processing** (status will be shown).

3. **Ask questions** about the PDF in the chat area.
4. **View chat history** and previous questions.

AWS/LocalStack Integration

- **S3:** Stores uploaded PDFs.
- **DynamoDB:** Stores PDF metadata and LLM metrics.
- **LocalStack:** Used for local AWS emulation (no real AWS costs).

Architecture

graph TD subgraph User Interface A["Browser (frontend/index.html)"] end subgraph "Backend (FastAPI)" B["main.py API"] C["RAG Pipeline"] D["Google Gemini LLM"] E["Vector Store (ChromaDB)"] end subgraph "AWS Services (via LocalStack)" F["S3 for PDF Storage"] G["DynamoDB for Metadata"] end A -- "HTTP Requests" --> B B -- "Processes PDF" --> C B -- "Asks Question" --> C C -- "Sends prompts" --> D C -- "Stores/Retrieves data" --> E B -- "Uploads to" --> F B -- "Writes to" --> G

API Documentation

`POST /upload-pdf/`

Uploads and processes a PDF file to make it available for question answering.

- **Method:** `POST`
- **Request:** `multipart/form-data`
  - `file`: The PDF file to upload.
- **Response:** `200 OK`

```
{
  "message": "PDF 'filename.pdf' processed successfully",
  "filename": "filename.pdf",
  "text_length": 12345,
  "available_pdfs": ["filename.pdf"],
  "status": "ready_for_questions"
}
```

`POST /ask`

Asks a question about the most recently uploaded PDF.

- **Method:** `POST`
- **Request:** `application/x-www-form-urlencoded`
  - `question`: The question to ask.
- **Response:** `200 OK`

```
{
  "answer": "The answer to your question is...",
  "question": "What is the main topic?",
  "pdf_name": "filename.pdf",
  "response_time": 1.23,
  "sources": ["...source text snippet 1...", "...source text snippet 2..."]
}
```

GET /status

Checks the current status of the RAG system.

- **Method:** `GET`
- **Response:** `200 OK`

```
{
  "pdf_loaded": true,
  "current_pdf": "filename.pdf",
  "status": "ready"
}
```

GET /health

A simple health check endpoint.

- **Method:** `GET`
- **Response:** `200 OK`

```
{
  "status": "healthy",
  "service": "RAG Chatbot API"
}
```

Troubleshooting

- **PDF Metadata Not Stored:**
  - Ensure DynamoDB table schema matches the code (`filename` and `user_id` as keys).
  - Check logs for errors about table initialization or AWS integration.
- **Frontend Reloads:**
  - Make sure `event.preventDefault()` is used in the upload button handler.
- **No Google API Key:**
  - Add your key to the `.env` file.
- **LocalStack Not Running:**
  - Start LocalStack before running the backend.

Contributing

Pull requests and issues are welcome! Please open an issue for bugs or feature requests.
