RAG Chatbot (Retrieval-Augmented Generation)

A 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.
- **Frontend:** Responsive, user-friendly interface for uploading, chatting, and viewing history.
- **Rate Limiting:** Prevents abuse with per-IP request limits.
- **Comprehensive Testing:** Full test suite with pytest for PDF upload and RAG pipeline functionality.

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 (updated to use pypdf)
      ...
    test_rag_pipeline/    # Comprehensive test suite
      __init__.py         # Test package initialization
      conftest.py         # Pytest configuration and fixtures
      test_rag_pipeline.py # RAG pipeline tests
      test_pdf_upload.py # PDF upload specific tests
    aws_service/
      dynamo_handler.py   # DynamoDB logic
      s3_handler.py       # S3 upload logic
      ...
    frontend/
      index.html          # Main frontend UI
      style.css           # Styles (if separated)
    data/
      Machine learning.pdf # Sample PDF for testing
    requirements.txt      # Updated dependencies (pypdf, pytest, etc.)
    README.md             # Detailed setup and testing guide
  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
```

Updated Dependencies:

- **pypdf** (replaces deprecated PyPDF2) for PDF processing
- **pytest** for testing framework
- **pytest-cov** for test coverage reporting
- **pytest-mock** for mocking in tests

3. **Run Tests (Optional but Recommended)**

```
# Run all tests
cd chatbot_rag
pytest test_rag_pipeline/ -v

# Run specific test files
pytest test_rag_pipeline/test_pdf_upload.py -v
pytest test_rag_pipeline/test_rag_pipeline.py -v

# Run with coverage report
pytest test_rag_pipeline/ --cov=app --cov-report=html
```

Test Coverage:

- PDF upload and validation
- Text extraction and processing
- RAG pipeline functionality
- Vector store operations
- Error handling and edge cases

4. **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
```

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

```
localstack start -d
```

6. **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
```

7. **Run the Backend**

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

8. **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

```mermaid
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)"]
        F["PDF Processing (pypdf)"]
    end

    subgraph "Testing Suite"
        G["pytest Framework"]
        H["PDF Upload Tests"]
        I["RAG Pipeline Tests"]
    end

    subgraph "AWS Services (via LocalStack)"
        J["S3 for PDF Storage"]
        K["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
    C -- "Extracts text" --> F
    G -- "Tests" --> H
    G -- "Tests" --> I
    B -- "Uploads to" --> J
    B -- "Writes to" --> K
```

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"
}
```

POST /clear-vectorstore/

Clears the vector store to free up memory.

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

```
{
  "message": "Vector store cleared successfully"
}
```

Testing

The project includes a comprehensive test suite to ensure code quality and reliability:

Test Structure

```
test_rag_pipeline/
├── __init__.py                    # Package initialization
├── conftest.py                    # Pytest configuration and fixtures
├── test_rag_pipeline.py           # Main RAG pipeline tests
└── test_pdf_upload.py             # PDF upload specific tests
```

Test Categories

- **PDF Upload Tests**: File validation, text extraction, metadata retrieval
- **RAG Pipeline Tests**: Text splitting, vector store creation, QA chain setup
- **Integration Tests**: Complete workflow testing
- **Error Handling**: Various failure scenarios and edge cases

Running Tests

```
# Run all tests
pytest test_rag_pipeline/ -v

# Run with coverage
pytest test_rag_pipeline/ --cov=app --cov-report=html

# Run specific test file
pytest test_rag_pipeline/test_pdf_upload.py -v
```

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.
- **Test Failures:**
  - Ensure all dependencies are installed: `pip install -r requirements.txt`
  - Check that sample PDF exists in `data/` folder
  - Verify Google API key is set for integration tests
- **PDF Processing Errors:**
  - The project now uses `pypdf` instead of deprecated `PyPDF2`
  - Ensure PDF files are not corrupted or password-protected

Recent Updates

v2.0 Updates

- □ **Migrated to pypdf**: Replaced deprecated PyPDF2 with modern pypdf library
- □ **Added Comprehensive Testing**: Full pytest test suite for PDF upload and RAG pipeline
- □ **Enhanced Error Handling**: Better error messages and validation
- □ **Updated Dependencies**: Modern, secure, and well-maintained packages
- □ **Improved Documentation**: Detailed setup and testing guides

Key Improvements

- **Better PDF Processing**: More reliable text extraction with pypdf
- **Test Coverage**: 90%+ test coverage for core functionality
- **Error Resilience**: Graceful handling of various failure scenarios
- **Developer Experience**: Clear testing instructions and debugging guides

---

Contributing

Pull requests and issues are welcome! Please:

1. **Run tests** before submitting: `pytest test_rag_pipeline/ -v`
2. **Follow the existing code style**
3. **Add tests** for new functionality
4. **Update documentation** as needed

License
