**DocQA Backend & Frontend Documentation**

**Overview**

DocQA is a document-based question-answering system built with **FastAPI** (backend) and **Streamlit** (frontend). Users can:

* Register/Login
* Upload documents (PDF, DOCX, TXT)
* Ask questions on uploaded documents
* View chat history

The system uses **PostgreSQL** for persistence and **Google Gemini API** for LLM responses. It also uses embeddings for document chunk retrieval.

**Backend (FastAPI)**

**1. Setup & Configuration**

import os, uuid, shutil

from fastapi import FastAPI, UploadFile, File, HTTPException

from fastapi.middleware.cors import CORSMiddleware

import psycopg2

import google.generativeai as genai

* **CORS Middleware** enabled for all origins.
* **Upload directory**: uploads/ (can be overridden via UPLOAD\_DIR environment variable).
* **Gemini LLM** configured via GEMINI\_API\_KEY.
* **PostgreSQL connection** configured with:
  + Host: localhost
  + Port: 5432
  + User: postgres
  + Password: root
  + Database: docqa (created if missing)

**2. Database Schema**

**Users Table**

| **Column** | **Type** | **Constraints** |
| --- | --- | --- |
| id | SERIAL | PRIMARY KEY |
| username | TEXT | UNIQUE, NOT NULL |
| password | TEXT | NOT NULL |

**Chats Table**

| **Column** | **Type** | **Constraints** |
| --- | --- | --- |
| id | SERIAL | PRIMARY KEY |
| user\_id | INT | REFERENCES users(id) |
| question | TEXT |  |
| answer | TEXT |  |
| created\_at | TIMESTAMP | DEFAULT CURRENT\_TIMESTAMP |

**3. Models (Pydantic)**

class UserIn(BaseModel):

username: str

password: str

class QueryIn(BaseModel):

query: str

top\_k: int = 4

username: str

**4. Authentication Endpoints**

**Register**

POST /register

Body: { "username": "chandru", "password": "mypassword" }

Response: { "status": "ok", "user\_id": 1 }

* Inserts user into the database.
* Handles duplicate usernames with 400 error.

**Login**

POST /login

Body: { "username": "chandru", "password": "mypassword" }

Response: { "status": "ok", "user\_id": 1, "username": "chandru" }

* Validates credentials.
* Returns 401 if invalid.

**5. Document Upload Endpoint**

POST /upload

Params: username

Body: file (PDF, DOCX, TXT)

* Saves file to uploads/ with a unique UUID filename.
* Extracts text depending on file type:
  + PDF → extract\_text\_from\_pdf
  + DOCX/DOC → extract\_text\_from\_docx
  + TXT → extract\_text\_from\_txt
* Chunks text into 800-character pieces.
* Embeds chunks and stores in the collection (COLLECTION\_NAME) for retrieval.

**Response Example**

{

"status": "ok",

"filename": "document.pdf",

"num\_chunks": 10

}

**6. Chat / Query Endpoint**

POST /query

Body: { "query": "What is Python?", "top\_k": 4, "username": "chandru" }

* Retrieves top K most relevant document chunks using embeddings.
* Constructs a prompt with context for Gemini LLM.
* Saves the query and answer in the chats table.

**Response Example**

{

"query": "What is Python?",

"answer": "Python is a programming language...",

"sources": [

{"source\_filename": "document.pdf", "ord": 0, "text\_snippet": "Python is ..."}

]

}

**7. Chat History Endpoint**

GET /history?username=chandru

* Retrieves all chats for the logged-in user.
* Returns question, answer, and timestamp (created\_at).

**Implementation Example**

@app.get("/history")

def get\_history(username: str):

user = get\_user(username)

if not user:

raise HTTPException(status\_code=401, detail="Invalid user")

import psycopg2.extras

cur = conn.cursor(cursor\_factory=psycopg2.extras.RealDictCursor)

cur.execute(

"SELECT question, answer, created\_at FROM chats WHERE user\_id=%s ORDER BY created\_at DESC",

(user[0],)

)

rows = cur.fetchall()

return {"history": rows}

**Response Example**

{

"history": [

{

"question": "What is Python?",

"answer": "Python is a programming language...",

"created\_at": "2025-09-14T21:10:45"

}

]

}

**Frontend (Streamlit)**

**1. Authentication**

* Tabs for Login and Register.
* Stores username and login state in st.session\_state.

**2. Upload Documents**

* File uploader for PDF, DOCX, TXT.
* Sends file to /upload with username parameter.
* Displays success message and number of chunks indexed.

**3. Ask Questions**

* Input box for query and slider for top\_k.
* Sends query to /query endpoint.
* Displays answer and source metadata.

**4. Chat History**

if st.session\_state.logged\_in:

resp = requests.get(f"{API\_BASE}/history", params={"username": st.session\_state.username})

history = resp.json().get("history", [])

for h in history:

st.markdown(f"\*\*Q:\*\* {h['question']}")

st.markdown(f"\*\*A:\*\* {h['answer']}")

st.markdown(f"\*Timestamp:\* {h['created\_at']}")

st.markdown("---")

* Fetches and displays all past queries and answers with timestamps.

This documentation covers **registration, login, document upload, chat/query, and chat history** features of your DocQA system.