AI Document Assistant Setup Guide

Overview

This is a full-stack document assistant application that allows users to upload PDFs and interact with them using AI-powered chat, summarization, and analysis features.

Prerequisites

- Python 3.9+
- Node.js 18+
- OpenAI API key (or DeepSeek/DuoBao API key)

Backend Setup

1. Create a new directory for the backend:

```bash
mkdir doc-assistant-backend
cd doc-assistant-backend
```

2. Create a virtual environment:

```bash
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. Create `main.py` and `requirements.txt` files with the provided code.

4. Install dependencies:

```bash
pip install -r requirements.txt
```

5. Create a `.env` file:

```bash
OPENAI_API_KEY=your-api-key-here
```

6. Run the backend:

```bash
uvicorn main:app --reload
```

The backend will be available at `http://localhost:8000`

Frontend Setup

1. Create a new Next.js app:

```bash
npx create-next-app@latest doc-assistant-frontend --typescript --tailwind --app
cd doc-assistant-frontend
```

2. Install additional dependencies:

```bash
npm install lucide-react
```

3. Replace the contents of `app/page.tsx`, `app/layout.tsx`, and `app/globals.css` with the provided code.
4. Create additional configuration files:

tailwind.config.ts:

```typescript
import type { Config } from 'tailwindcss'

const config: Config = {
  content: [
    './pages/**/*.{js,ts,jsx,tsx,mdx}',
    './components/**/*.{js,ts,jsx,tsx,mdx}',
    './app/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
export default config
```

next.config.js:

```javascript
/** @type {import('next').NextConfig} */
const nextConfig = {}

module.exports = nextConfig
```

5. Run the frontend:

```bash
npm run dev
```

The frontend will be available at `http://localhost:3000`

Usage

1. **Upload Documents**: Click "Upload PDF" to upload a document.
2. **Chat**: Ask questions about the document in the chat interface.
3. **Summary**: Click the "Summary" tab to get an automatic summary and key points.
4. **Analysis**: Click the "Analysis" tab for detailed document analysis.

Features

- **RAG-based Chat**: Uses retrieval-augmented generation for accurate responses
- **Streaming Responses**: Real-time streaming of AI responses
- **Source Citations**: Shows which parts of the document were used for answers
- **Document Summarization**: Automatic summary generation with key points
- **Document Analysis**: In-depth analysis including main themes, audience, and recommendations
- **Multi-document Support**: Upload and switch between multiple documents

API Endpoints

- `POST /upload` - Upload a PDF document
- `POST /chat` - Chat with a document (streaming)
- `POST /summarize` - Generate document summary
- `POST /analyze` - Perform document analysis
- `GET /documents` - List all uploaded documents

Customization

Using DeepSeek/DuoBao API

Replace the OpenAI initialization in `main.py`:

```python
# For DeepSeek
llm = ChatOpenAI(
    temperature=0.7,
    model_name="deepseek-chat",
    openai_api_key="your-deepseek-api-key",
    openai_api_base="https://api.deepseek.com/v1"
)

# For DuoBao
llm = ChatOpenAI(
    temperature=0.7,
    model_name="duobao-model-name",
    openai_api_key="your-duobao-api-key",
    openai_api_base="https://api.duobao.com/v1"
)
```

Using PostgreSQL

To use PostgreSQL instead of local storage:

1. Install additional dependencies:

```bash
pip install psycopg2-binary sqlalchemy
```

2. Create a database connection and store document metadata in PostgreSQL
3. Use pgvector for vector storage instead of ChromaDB

Using Pinecone

To use Pinecone instead of ChromaDB:

1. Install Pinecone:

```bash
pip install pinecone-client
```

2. Replace ChromaDB initialization:

python

```python
from langchain.vectorstores import Pinecone
import pinecone

pinecone.init(api_key="your-pinecone-api-key", environment="your-environment")
vectorstore = Pinecone.from_documents(
    documents=chunks,
    embedding=embeddings,
    index_name="your-index-name"
)
```

Troubleshooting

1. **CORS Issues**: Make sure the backend CORS middleware includes your frontend URL
2. **Large Files**: For very large PDFs, consider implementing chunked upload
3. **Memory Issues**: For production, implement proper cleanup of vector stores
4. **Rate Limits**: Implement rate limiting and request queuing for API calls