PDF-Q/A-Analyser

Architecture Overview

The application is a full-stack PDF Q&A platform enabling users to upload PDF files, extract their content, and ask questions about the content. It comprises the following key components:

Backend

1. **Framework**: FastAPI
    - Provides API endpoints to handle PDF uploads, extract content, and process Q&A requests.
2. **Database**: PostgreSQL
    - Stores metadata about uploaded files, such as file paths and unique identifiers.
3. **Libraries and Tools**:
    - **LangChain** : Used for chunking text and performing vector-based Q&A.
    - **FAISS**: Provides efficient similarity search and vector storage.
    - **Gemini Generative AI**: Processes user queries based on the extracted content using a generative language model.
    - **Pydantic**: Validates request payloads.
    - **SQLAlchemy**: Manages ORM for database interactions.
4. **File Storage**: Local storage for uploaded files.

Frontend

1. **Framework**: React.js
    - Provides an intuitive UI for uploading files and interacting with the system.
2. **Components**:
    - **File Upload**: Allows users to upload PDF documents.
    - **Question Input**: Enables users to input questions.
    - **Conversation Display**: Shows a history of questions and answers.
3. **Styling**: Custom CSS for responsive design and interactive UI elements.

Flow of Operations

1. **File Upload**:
    - **Frontend**: The user selects a PDF file and uploads it via the file upload component.

   - **Backend**:
     - The file is validated (must be a PDF) and stored in the UPLOAD_DIRECTORY.
     - File metadata (e.g., path and unique ID) is saved to the database.
     - The backend responds with a unique file_id for the uploaded document.
2. **Text Extraction and Chunking**:
   - The backend extracts text from the uploaded PDF using a utility function.
   - The extracted text is split into manageable chunks using CharacterTextSplitter, ensuring efficient processing during Q&A.
3. **Q&A Processing**:
   - **Frontend**: The user enters a question, which is sent to the backend along with the document ID.
   - **Backend**:
     - Retrieves the document from the database and processes the text using LangChain or Gemini Generative AI.
     - Generates a context-aware response based on the question and text content.
     - Returns the answer to the frontend.
4. **Conversation Management**:
   - **Frontend**:
     - Displays the question and its corresponding answer in a conversational format.
     - Allows users to view the history of their interactions.

Key Interactions Between Components

- **Frontend ↔ Backend**:
  - Communicates using RESTful API endpoints (/upload for file uploads and /ask for Q&A).
- **Backend ↔ Database**:
  - Uses SQLAlchemy for storing and retrieving file metadata.
- **Backend ↔ NLP Models**:
  - Integrates with FAISS and Gemini for text vectorization, similarity search, and generative responses.

Component Roles and Responsibilities

Component	Role
FastAPI	Handles API requests, validates input, and coordinates backend workflows.
React.js	Manages the user interface and provides an interactive experience for file uploads and Q&A.
SQLite/PostgreSQL	Stores document metadata and facilitates data persistence.
LangChain/FAISS	Processes document text, chunks it, and supports vector-based information retrieval.
Gemini Generative AI	Generates context-aware answers to user questions.

This documentation provides a clear overview of the application's architecture, facilitating better understanding for development, testing, and deployment.