Application Architecture Overview

1 Application Architecture Overview

This document provides a brief overview of the application architecture, which consists of a FastAPI backend and a React frontend, along with a SQLite database . The architecture is designed to handle PDF uploads, text extraction, and question-answering functionalities using LangChain and AI21 API.

1.1 1. Frontend (React)

- **Components:** The frontend is built using React, which manages the user interface. It includes components for:
  - Uploading PDF files.
  - Asking questions related to the uploaded PDFs.
  - Displaying answers returned by the backend.
- **API Integration:** The frontend interacts with the backend using RESTful API calls to upload PDFs and ask questions. The API endpoints are defined in `api.js`.
- **State Management:** Local state management is handled within React components, ensuring a responsive user experience.

1.2 2. Backend (FastAPI)

- **Web Framework:** FastAPI is used to build the backend services. It handles incoming HTTP requests and responses, providing fast and efficient API endpoints.
- **CORS Middleware:** Configured to allow cross-origin requests, enabling the frontend to communicate with the backend.
- **Database Management:**
  - Uses SQLAlchemy for ORM (Object-Relational Mapping) to interact with the database.
  - Defines a model (`PDFDocument`) to store metadata and content from uploaded PDFs.
  - Handles PDF uploads and question answering using specific endpoints:
    - `/upload_pdf/`: Accepts a PDF file upload, processes the file to extract text, and stores it in the database.
    - `/ask_question/`: Accepts a PDF ID and a question, retrieves the corresponding PDF content, and generates an answer using LangChain and AI21.

1.3 3. Database

- **Database:** The application can connect to a SQLite. The database stores:
  - Metadata about the uploaded PDFs, including file names, extracted text content, and timestamps for creation and updates.
- **Schema Definition:** The schema is defined using SQLAlchemy's declarative base, ensuring structured storage and retrieval of PDF documents.

1.4 4. PDF Processing

- **Text Extraction:** The `pdf_processing.py` module uses PyMuPDF to extract text from uploaded PDFs.
- **Question Answering:** Utilizes LangChain to process the extracted text and provide answers to user questions. This involves creating embeddings and leveraging the AI21 API for natural language processing tasks.

1.5 5. Environment Configuration

- **Environment Variables:** The application uses environment variables to store sensitive data like database URLs and API keys, enhancing security and configurability.

2 Workflow Summary

1. A user uploads a PDF file through the React frontend.
2. The frontend sends the file to the FastAPI backend via the `/upload_pdf/` endpoint.
3. The backend processes the PDF, extracts the text, and stores it in the database.
4. The user can then ask questions about the PDF content through the frontend, which sends the question to the backend via the `/ask_question/` endpoint.
5. The backend retrieves the PDF content, processes it using LangChain, and returns the answer to the frontend.
6. The frontend displays the answer to the user.

3 Diagram Representation (Optional)

You can visualize this architecture with a simple diagram:

```
User Interface (React)
        |
        v
  [HTTP Requests]
        |
        v
   FastAPI Backend
        |
        v
   [Database Operations]
        |
        v
     SQLite
        |
        v
   [PDF Processing]
        |
        v
   LangChain + AI21 API
```

This architecture allows for efficient handling of PDF documents and supports advanced question-answering capabilities, providing a robust and user-friendly application.