High-Level Design (HLD) :

Overview

The PDF Question-Answering Application is a full-stack system enabling users to upload PDF documents, extract and process their contents, and interactively query the documents using AI. The application is divided into two primary layers: **Frontend** (React.js) and **Backend** (FastAPI), with seamless integration through RESTful APIs. Google's Gemini API powers the natural language processing capabilities.

System Components

1. Frontend

Responsibilities:

- User interaction and input handling
- Displaying uploaded documents and AI-generated answers
- Providing a responsive and intuitive interface

Technologies:

- React.js for component-based UI development
- Tailwind CSS for styling
- Axios for making HTTP requests
- Vite for optimized builds

Key Features:

- Document upload UI
- Interactive Q&A interface with real-time feedback
- Document management (view/delete)

---

2. Backend

Responsibilities:

- Handling file uploads and text extraction
- Managing documents and associated metadata
- Processing user queries and interacting with Google Generative AI API
- Ensuring security and performance optimization

Technologies:

- FastAPI for building RESTful APIs
- PyPDF for text extraction from PDF files
- Google Generative AI for NLP
- Python-multipart for file upload handling

Key Features:

- PDF processing and storage
- Integration with external NLP API
- Efficient handling of user queries

---

3. Database

Responsibilities:

- Store metadata of uploaded documents
- Track Q&A history for each document

Proposed Technologies:

- SQLite (for lightweight storage needs, extendable to PostgreSQL/MySQL)
- Fields: document_id, document_name, upload_date, file_path, qa_history

---

4. External Services

Google Generative AI (Gemini):

- Natural language processing for extracting answers from document text
- Ensures contextual and relevant responses

High-Level Architecture

```
+-------------------------+
|       Frontend          |
|  (React.js + Tailwind)  |
+-----------+-------------+
            |
            v
```

```
+------------------------+
|      Backend          |
|      (FastAPI)        |
+-----------+------------+
          |
          v
+------------------------+
|  Google Generative AI  |
|        (Gemini)       |
+------------------------+
          |
          v
+------------------------+
|      Database         |
|  (SQLite/PostgreSQL)    |
+------------------------+
```

Low-Level Design (LLD):

Backend Modules

1. File Handling Module

- **Functionality:**
  - Validate uploaded files (type and size)
  - Store files in the uploads/ directory
  - Extract text using PyPDF
- **Key Functions:**
  - validate_file_type(file)
  - store_file(file)
  - extract_text(file_path)

2. Question-Answering Module

- **Functionality:**
  - Parse user queries
  - Interact with Google Generative AI API
  - Return relevant answers
- **Key Functions:**
  - ask_question(document_id, question)
  - parse_response(api_response)

3. Document Management Module

- **Functionality:**
  - Handle CRUD operations for documents
  - Provide metadata for frontend rendering
- **Key Functions:**
  - get_all_documents()
  - get_document_details(document_id)
  - delete_document(document_id)

4. API Routing

- **Endpoints:**
  - POST /upload – Upload and process PDFs
  - POST /ask – Query document content
  - GET /documents – Fetch document list
  - GET /documents/{document_id} – Fetch document details
  - DELETE /documents/{document_id} – Delete a document

Frontend Components

1. App.jsx

- **Role:** Main entry point for the React app.
- **Sub-components:**
  - Header

    - DocumentUpload
    - QuestionAnswer
    - DocumentManagement

2. DocumentUpload Component

- **Role:** Handle PDF uploads
- **Logic:**
    - Validate file selection
    - Make API call to /upload
    - Show progress indicators

3. QuestionAnswer Component

- **Role:** Enable interactive Q&A
- **Logic:**
    - Capture user input
    - Make API call to /ask
    - Render AI responses

4. DocumentManagement Component

- **Role:** Display document list and handle deletions
- **Logic:**
    - Fetch data from /documents
    - Render document metadata
    - Delete document via /documents/{document_id}

Deployment

Backend

- Host using **AWS Lambda** or **Heroku** for scalability
- Use **Gunicorn** or **Uvicorn** for serving FastAPI

Frontend

- Deploy via **Vercel** or **Netlify** for optimized static hosting

Database

- Use **AWS RDS** or **Heroku PostgreSQL** for production

Conclusion

The design ensures modularity, scalability, and security while providing a seamless user experience. Each component can be independently developed, tested, and deployed, enabling robust development practices.