PDF Chat Application Architecture

High Level Design (HLD)

System Architecture

The PDF Chat application follows a modern client-server architecture with three main components:

1. **Frontend Layer (Client)**
    - Built with React + TypeScript + Vite
    - Responsive UI using TailwindCSS
    - Client-side routing with React Router
    - Real-time notifications using react-hot-toast
2. **Backend Layer (Server)**
    - FastAPI-based RESTful API
    - Asynchronous request handling
    - CORS middleware for security
    - Modular router-based architecture
3. **Data Layer**
    - PostgreSQL with pgvector extension for vector storage
    - File system storage for PDF documents
    - Cohere API integration for embeddings and chat

Low Level Design (LLD)

1. Frontend Components

Core Components

- components/
    - Navbar.tsx (Navigation and routing)
    - FeatureCard.tsx (Reusable feature display)
    - NavItem.tsx (Navigation items)

Route Structure

- routes/
    - Upload.tsx (PDF upload interface)

    - Chat.tsx (Main chat interface)
    - Synced.tsx (File management)
    - About.tsx (App information)
    - Feature.tsx (Features showcase)
    - Pricing.tsx (Pricing plans)
    - Contact.tsx (Contact form)

2. Backend Services

API Routes

- routes/
    - upload.py (File upload handling)
    - chat.py (Chat functionality)
    - files.py (File management)

Utility Services

- utils/
    - chunker.py (Text splitting)
    - storedata.py (Vector storage)
    - storemeta.py (Metadata management)
    - delete.py (Cleanup utilities)

3. Data Flow

1. **Document Processing Flow**

   ```
   Upload → PDF Extraction → Text Chunking → Vector Embedding → PostgreSQL Storage
   ```

2. **Chat Flow**

   ```
   Query → Embedding Generation → Vector Search → Context Retrieval → LLM Response
   ```

4. Key Features Implementation

Document Processing

- Uses PyMuPDF for PDF text extraction
- Implements RecursiveCharacterTextSplitter for text chunking (500 chars, 50 overlap)
- Stores vectors using pgvector with 1024-dimensional embeddings

Chat System

- Uses Cohere's embed-english-v3.0 model for embeddings
- Implements semantic search with cosine similarity
- Maintains conversation context using LangGraph

- Supports markdown formatting in responses

File Management

- Implements CRUD operations for PDF files
- Handles both file system and database storage
- Cascading deletes for file and vector cleanup

5. Database Schema

```
-- Files Table
CREATE TABLE files (
    file_id SERIAL PRIMARY KEY,
    file_location VARCHAR(255),
    file_name VARCHAR(255) UNIQUE,
    file_size INT,
    file_type VARCHAR(255),
    user_gmail VARCHAR(255)
);

-- Embeddings Table
CREATE TABLE embeddings (
    id SERIAL PRIMARY KEY,
    text TEXT,
    embedding vector(1024),
    file_id INT REFERENCES files(file_id) ON DELETE CASCADE
);
```

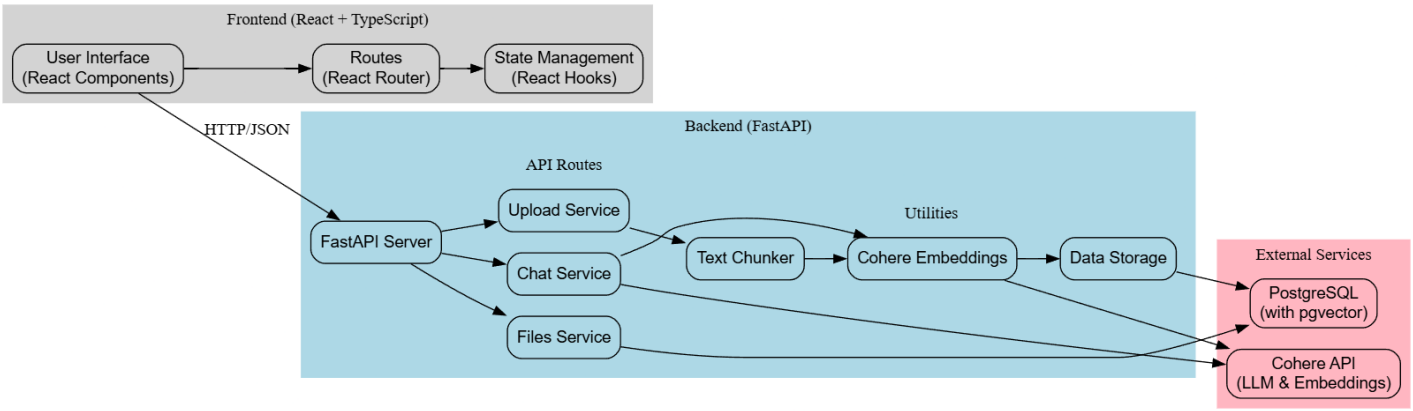
6. Security Measures

- CORS configuration for frontend-backend communication
- File type validation for uploads
- Size limitations on uploads (recommended 10-20 pages)
- Environment variable management for sensitive data

7. Performance Optimizations

- Async request handling
- Chunked file processing
- Vector similarity search optimization
- Client-side state management
- Responsive UI design

Architecture


Frontend (React + TypeScript)
User Interface (React Components)
Routes (React Router)
State Management (React Hooks)
HTTP/JSON
Backend (FastAPI)
API Routes
FastAPI Server
Upload Service
Chat Service
Files Service
Utilities
Text Chunker
Cohere Embeddings
Data Storage
External Services
PostgreSQL (with pgvector)
Cohere API (LLM & Embeddings)