Patient Document Portal — Project Submission

Full-stack implementation using React + Express + SQLite

Contents:

Project Overview

This project implements a simple full-stack Patient Document Portal allowing a single user to upload, view, download, and delete PDF medical documents (prescriptions, test results, referral notes). Files are stored locally in an **uploads/** folder and metadata is saved in a SQLite database.

Deliverables Included

- **Frontend**: React application (Upload + Documents pages)
- **Backend**: Express server with endpoints for upload, list, download, delete; Multer for file handlingSQLite for metadata
- README with run steps and example API calls .

Tech Stack Choices

Frontend: React with react-router-dom for simple SPA routing. Chosen for developer productivity and ease of building interactive forms.

Backend: Node.js with Express. Chosen for minimal boilerplate and quick API development.

Database: SQLite (sqlite3). Chosen because the assignment requires a simple local DB, SQLite requires no separate server and is sufficient for single-user local testing.

File Storage: Local filesystem (uploads/). For this assignment local storage is adequate and simple to inspect.

If scaling to 1,000 users: Replace SQLite with PostgreSQL, move files to object storage (S3), add authentication (JWT), use a load balancer and multiple backend instances, and add pagination and background processing for large uploads.

Architecture Overview

Flow between components:

1. Frontend (React) provides Upload UI and Documents listing.
2. Frontend calls Backend REST API endpoints on **http://localhost:5000.**
3. Backend (Express) receives uploads (multipart/form-data), stores the file in uploads/ and writes metadata (filename, filepath, filesize, created_at) into SQLite database (db.sqlite).
4. Frontend lists documents by calling GET /documents which reads metadata from SQLite and returns JSON.
5. Download calls GET /documents/:id which sends the file with original filename.
6. Delete calls DELETE /documents/:id which removes the file from disk and its metadata from the database.

API Specification

1) POST /documents/upload - Description: Upload a PDF file (multipart/form-data, field name 'file'). - Request: form-data with 'file' -> PDF - Response (201): { id, filename, filesize, created_at, message }

2) GET /documents - Description: List all uploaded documents metadata. - Response (200): Array of { id, filename, filepath, filesize, created_at }

3) GET /documents/:id - Description: Download stored file for the given id. - Response: file download (Content-Disposition attachment)

4) DELETE /documents/:id - Description: Delete the file and its metadata. - Response (200): { message: 'Deleted successfully', id }

Data Flow (Upload & Download)

Upload flow (step-by-step):

1. User selects PDF in frontend Upload page.

2. Frontend validates MIME type and size client-side.

3. Frontend sends multipart/form-data POST to /documents/upload.

4.Backend multer middleware writes file to uploads/ with a unique filename. 5. Backend inserts metadata record into documents table in SQLite (id, filename, filepath, filesize, created_at). 6. Backend returns 201 with metadata including id. 7. Frontend refreshes the documents list.

Download flow:

1. User clicks Download on a document.

2. Frontend navigates to GET /documents/:id URL (anchor link with download attribute).

3. Backend locates file path from DB and sends file via res.download().

Assumptions & Limits

- Single-user application (no authentication) as required.
- Uploads limited to PDF files only. Both MIME-type and file extension are checked.
- File size limit set to 20 MB (configurable in backend multer limits).
- Files stored on local disk under backend/uploads/. Metadata in backend/db.sqlite.
- Concurrency: Basic local server; for heavy concurrent uploads a production architecture is needed.

How to Run Locally

Backend: 1. cd backend 2. npm install 3. npm start

Frontend: 1. cd frontend 2. npm install 3. npm start

Ensure backend is available at http://localhost:5000 (default). Frontend runs at http://localhost:3000.

Example API calls: - Upload (curl): curl -X POST -F "file=@/path/to/example_document.pdf" http://localhost:5000/documents/upload - List: curl http://localhost:5000/documents - Download: open http://localhost:5000/documents/1 - Delete: curl -X DELETE http://localhost:5000/documents/1

Files Created / Important Paths

backend/ - server.js - routes/documents.js - db.js - uploads/ (created at runtime) - db.sqlite (created at runtime)

frontend/ - src/components/UploadPage.js - src/components/DocumentsPage.js - src/api.js - src/App.js - src/index.css

Example PDF for testing: example_document.pdf (included in project root in this environment)