Patient Document Portal - Design Document

1. Tech Stack Choices

Q1. Frontend Framework

- **Choice:** Vanilla HTML, CSS, JavaScript
- **Reason:**
    - Lightweight and simple for a single-user portal.
    - No complex state management required.
    - Easy to integrate with REST APIs.

Q2. Backend Framework

- **Choice:** Spring Boot (Java)
- **Reason:**
    - Provides built-in support for REST APIs.
    - Easy database integration with Spring Data JPA.
    - Handles file uploads efficiently.
    - Robust for scaling if needed in future.

Q3. Database

- **Choice:** MySQL
- **Reason:**
    - Reliable relational database for storing metadata.
    - Supports unique IDs and indexing for faster queries.
    - Can scale to multiple users easily.

Q4. Supporting 1,000 users

- Use a proper authentication system (JWT or Spring Security).
- Store files in cloud storage (AWS S3, GCP Storage) instead of local uploads/.
- Use a connection pool for database (HikariCP).
- Enable pagination for document lists.
- Introduce caching for frequently accessed metadata.

2. Architecture Overview

Flow Diagram (simplified):

[User Browser] <--HTTP--> [Frontend: HTML/JS]

|

v

[Backend: Spring Boot REST API]

|

+--> /documents/upload -> saves file to uploads/ + metadata to DB

+--> /documents -> fetch metadata from DB

+--> /documents/:id -> download or delete file

|

v

[Database: MySQL]

[File Storage: local uploads/ folder]

Bullet Points:

- User interacts with **frontend** for uploading, viewing, downloading, deleting files.
- **Frontend** sends HTTP requests to **backend REST API**.
- Backend stores **files locally** (uploads/) and **metadata in database**.
- Backend returns JSON responses to frontend for display.

3. API Specification

Endpoint	**Method**	**Description**	**Sample Request**	**Sample Response**
/documents/upload	POST	Upload a PDF	Form-data: file	{ "id": 1, "originalFilename": "report.pdf", "storedFilename": "169999_report.pdf", "filePath": "uploads/169999_report.pdf", "fileSize": 124567, "createdAt": "2025-12-09T12:00:00" }
/documents	GET	List all uploaded files	GET request	[{ "id":1, "originalFilename":"report.pdf", "fileSize":124567, "createdAt":"2025-12-09T12:00:00" }, ...]
/documents/:id	GET	Download a file by ID	GET /documents/1	Returns PDF file as attachment
/documents/:id	DELETE	Delete a file by ID	DELETE /documents/1	"Document deleted successfully" or "Document ID 1 not found in database"

4. Data Flow Description

Upload File:

1. User selects PDF on frontend.
2. Frontend sends POST request /documents/upload with multipart/form-data.
3. Backend validates file type.
4. Backend saves file to uploads/ folder.
5. Metadata (filename, path, size, createdAt) saved in DB.
6. Backend returns success/failure message.

Download File:

1. User clicks “View” link or enters ID and clicks “View”.
2. Frontend sends GET request /documents/:id.
3. Backend retrieves file path from DB.
4. Backend sends file as attachment to browser.

Delete File:

1. User enters ID or clicks “Delete” in table.
2. Frontend sends DELETE request /documents/:id.
3. Backend checks if ID exists in DB.
4. If exists: delete file from uploads/ and remove DB record. Return success message.
5. If not: return message "ID not found in database".

5. Assumptions

- Only **PDF files** are allowed.
- Maximum file size: **12MB**.
- No authentication (single-user system).
- Database IDs are **unique**.
- Concurrency is minimal (local single-user portal).
- Local storage is sufficient for small-scale usage.