Assignment: Full Stack Developer Intern

This assignment is designed to help us evaluate your understanding of **web development fundamentals**, ability to build a basic full-stack application, and the thought process behind your choices. While we don't expect enterprise-grade production code, we do expect a **working solution**, a **clean structure**, and clear explanations of your decisions.

Please **read the problem carefully** and provide clear justifications wherever asked. This is **not just a coding test**, but also an assessment of your approach to solving real-world problems.

Problem Statement

A healthcare platform wants to build a patient portal where users (patients) can upload and manage their **medical documents** (PDFs). These could include prescriptions, test results, or referral notes.

Each user should be able to:

- Upload a PDF file.
- View all their uploaded documents.
- Download any of the documents.
- Delete a document when it's no longer needed.

Your job is to build a **simple full-stack application** that allows the above operations via a **clean web interface** and a **working backend API**. This application should run **locally**.

Core Requirements

1. **Frontend Application**
    - Form to upload a PDF file
    - List all uploaded files
    - Download and delete buttons for each file
    - Display messages on success/failure
2. **Backend API Service**
    - REST APIs to:
        - Upload a file (PDF only)
        - List all uploaded files

        - Download a specific file
        - Delete a file

    - Store uploaded files in a local `uploads/` folder
    - Save metadata to a database (e.g., filename, upload date, file size)

3. **Database**

    - Store file metadata in a table (e.g., id, filename, size, created_at)
    - Use SQLite, PostgreSQL, or similar
    - No need to implement user login — assume one user for simplicity

Part 1: Design Document (Mandatory)

Create a file called `design.md` or `design.pdf` in your repository. It should include:

1. Tech Stack Choices

Q1. What frontend framework did you use and why? (React, Vue, etc.)

Q2. What backend framework did you choose and why? (Express, Flask, Django, etc.)

Q3. What database did you choose and why? (SQLite vs PostgreSQL vs others)

Q4. If you were to support 1,000 users, what changes would you consider?

2. Architecture Overview

1. Draw or describe the flow between frontend, backend, database, and file storage.
2. You can use a simple diagram or bullet points.

3. API Specification

For each of the following endpoints, provide:

- URL and HTTP method
- Sample request & response
- Brief description

Required Endpoints:

Endpoint	Method	Description
`/documents/upload`	POST	Upload a PDF
`/documents`	GET	List all documents
`/documents/:id`	GET	Download a file
`/documents/:id`	DELETE	Delete a file

4. Data Flow Description

Q5. Describe the step-by-step process of what happens when a file is uploaded and when it is downloaded.

5. Assumptions

Q6. What assumptions did you make while building this? (e.g., file size limits, authentication, concurrency)

Part 2: Local Implementation

Tasks:

1. Frontend

- Upload PDF file (with validation)
- Show uploaded files in a list
- Allow download and deletion

2. Backend

- Store files locally in `uploads/`
- Store metadata in a database
- Handle all required endpoints

3. Database

- Use a simple table called `documents` with fields:
  - `id`, `filename`, `filepath`, `filesize`, `created_at`

Deliverables

Push your solution to a **public GitHub repository** and share the link.

Your repository **must include**:

1. `design.md` or `design.pdf` with:
   - Architecture
   - Stack choices
   - Answered questions
   - API spec
   - Assumptions

2. `frontend/` with:
   - Source code
   - A working UI

3. `backend/` with:
   - Source code
   - File handling & API logic

4. `README.md` with:
   - Project overview
   - How to run it locally
   - Example API calls (curl or Postman)