

## DESIGN DOCUMENT

### 1. Tech Stack Choices

#### Q1. Frontend Framework:

No framework was used. The frontend uses plain HTML, CSS, and Vanilla JavaScript.

Reason: Lightweight, minimal complexity, ideal for a small local demo without build tools.

#### Q2. Backend Framework:

Express.js was chosen due to its simplicity, extensive middleware ecosystem (e.g., Multer), and suitability for REST APIs.

#### Q3. Database Choice:

SQLite was chosen.

Reason: Single-file, no installation, perfect for local development and internship assignments.

#### Q4. Scaling to 1,000 users:

- Move from SQLite to PostgreSQL
- Add file storage in AWS S3 or similar
- Use NGINX + PM2 for load balancing
- Add authentication and rate limiting
- Use a cloud deployment platform

### 2. Architecture Overview

Flow:

Frontend (HTML/JS)    Backend Express API    SQLite for metadata

Local filesystem (uploads/) for actual PDFs.

### 3. API Specification

#### 1. POST /documents/upload

Description: Upload a PDF.

Sample Response:

```
{  
  "file": {  
    "id": 1,  
    "name": "report.pdf",  
    "size": 20344  
  }  
}
```

#### 2. GET /documents

Description: List all stored documents.

Sample Response:

```
{  
  "files": [  
    { "id": 1, "name": "report.pdf", "size": 20344 }  
  ]  
}
```

#### 3. GET /documents/:id

Description: Download a file. Returns binary PDF.

#### 4. DELETE /documents/:id

Description: Delete a file and its metadata.

#### 4. Data Flow Description

Upload steps:

1. User selects a PDF in the frontend.
2. Frontend sends multipart/form-data to backend.
3. Multer stores file in uploads/ folder.
4. Backend stores metadata in SQLite.
5. Frontend refreshes list.

Download steps:

1. Frontend hits /documents/:id/download.
2. Backend fetches file path from DB.
3. File is streamed to browser.

#### 5. Assumptions

- Only PDFs allowed
- Max file size: 20MB
- Single user scenario, no authentication
- SQLite suitable for local use only
- No concurrent uploads expected