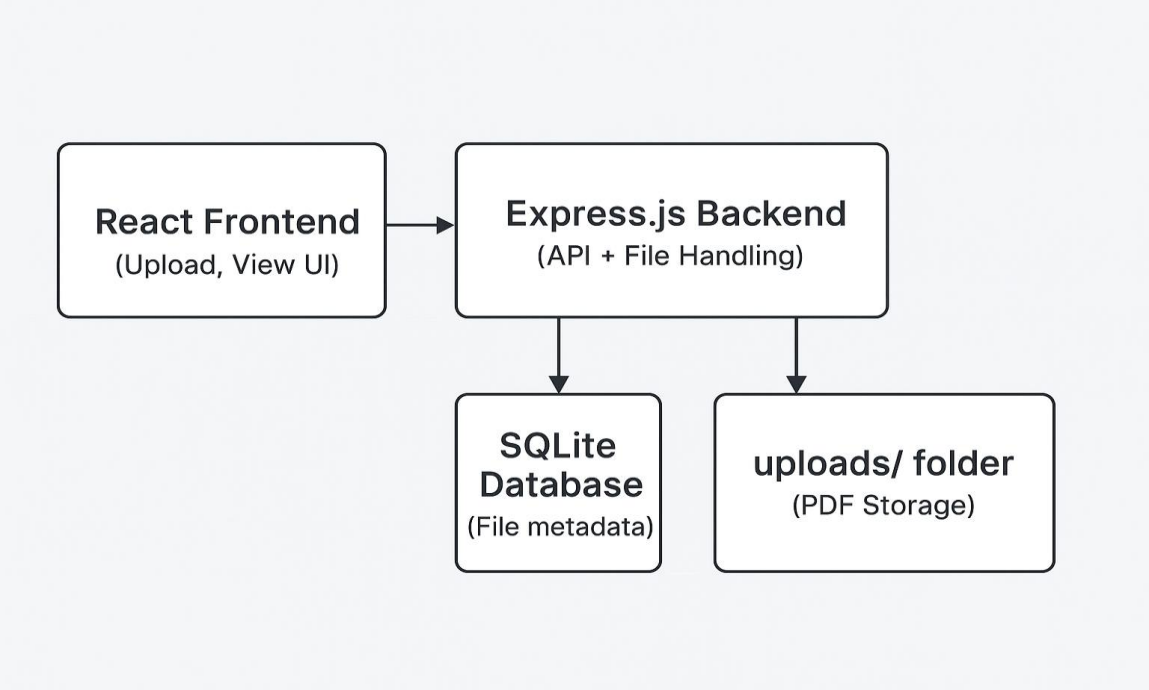
ASSESSMENT

Architecture

High-Level System Architecture



1. **Tech Stack Choices**

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

Chosen: React.js
Reason:

- Component-based architecture → easy to build reusable UI (upload form, file list, file card).
- Strong ecosystem and community support.
- Ideal for building responsive, interactive dashboards.
- Easy integration with REST API using Axios or Fetch.

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

Chosen: Node.js + Express.js
Reason:

- Simple, lightweight, and ideal for building REST APIs.
- Fast file handling using middleware like multer.
- JSON-native ecosystem → smooth communication with frontend.
- Minimal setup required for small-scale local applications.

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

Chosen: SQLite
Reason:

- Zero-configuration, file-based DB.
- Perfect for local development and small-scale apps.
- Allows quick CRUD operations without complex setup.
- Easy to migrate to PostgreSQL if app scales later.

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

To support 1,000 users:

Frontend

- Build optimization (React production build).
- CDN delivery for faster static asset loading.

Backend

- Move from single Express server → load-balanced Node cluster.
- Add request validation + rate limiting.
- Use S3 or cloud-based file storage instead of local folder.

Database

- Migrate from SQLite → PostgreSQL / MySQL.
- Add proper indexing (e.g., filename, created_at).

Storage

- Replace local uploads/ folder with AWS S3 bucket.

2. Architecture Overview

Q1. Draw or describe the flow between frontend, backend, database, and file storage.

React Frontend → Express Backend API → SQLite Database

↓

Local uploads/ folder

Q2. You can use a simple diagram or bullet points.

Described:

1. **User uploads a PDF through React UI.**

2. **React sends file → Express /documents/upload.**
3. **Express validates file, saves file in uploads/.**
4. **Metadata stored in the SQLite table documents.**
5. **Frontend fetches list using /documents.**
6. **For download/delete:**
    - **/documents/:id GET → returns file stream.**
    - **/documents/:id DELETE → removes entry + deletes file.**

3. API Specification

For each of the following endpoints, provide:

● URL and HTTP method

● Sample request & response

● Brief description

Answer:

Base URL **http://localhost:5000**

1. **Upload PDF**

Endpoint: `POST /documents/upload`

Description: Upload a PDF file.

Headers: `Content-Type: multipart/form-data`

Request: `file: <PDF file>`

Sample Response:

```
{
  "message": "File uploaded successfully",
  "document": {
    "id": 1,
    "filename": "report.pdf",
    "filesize": 204800,
    "created_at": "2025-12-11 10:00:00"
  }
}
```

2. **List All Documents**

Endpoint: `GET /documents`

Description: Returns metadata of all uploaded files.

Sample Response:

```
[
  {
    "id": 1,
    "filename": "report.pdf",
    "filesize": 204800,
    "created_at": "2025-12-11 10:00:00"
  }
]
```

3. Download File

Endpoint:

`GET /documents/:id`

Description: Sends back PDF file as download.

4. Delete File

Endpoint: DELETE /documents/:id

Description: Deletes PDF file + database entry.

Sample Response:

{

"message": "Document deleted successfully"

}

4. Data Flow Description

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

1. User selects PDF in frontend.
2. React uploads file → Express using multipart/form-data.
3. Express validates:
   - File exists
   - File is a PDF
4. File is saved in /uploads/.
5. File metadata inserted into SQLite:
   - filename
   - filepath
   - filesize
   - created_at
6. Response sent back to frontend.
7. Frontend shows success message.

File Download Flow:

1. Frontend calls GET /documents/:id.
2. Backend finds file path from DB.
3. If file exists → Express sends file using res.download().
4. Browser downloads PDF.

5. Assumptions

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

Answer:

- Only one user exists → no login/authorization needed.
- Only PDF files are allowed.
- Local storage is used (uploads/ folder).
- Concurrency is low since this is a local running app.
- No versioning of files; filenames may be stored uniquely.
- Delete operation removes both:
  - local file
  - database record