# FULL STACK ASSIGNMENT ANSWERS

**Q1. What frontend framework did you use and why?**

I used **React** for the frontend. The main reason for choosing React is that it makes building user interfaces much simpler through its component-based structure. Each part of the UI (like the upload form, file list, and buttons) can be created as separate components, which keeps the code organised and easier to maintain.

React also handles UI updates efficiently. For example, when I upload or delete a document, the page updates automatically without needing a full reload. This creates a smoother experience for the user. Another reason I preferred React is that it has a large community, plenty of ready-made libraries, and excellent support for making API calls, which helped me integrate the frontend with my Express backend very easily.

Overall, React helped me build a clean, responsive, and interactive interface in less time compared to writing everything manually with plain JavaScript.

**Q2. What backend framework did you choose and why?**

I chose **Express.js** for the backend. The main reason is that Express is very lightweight and straightforward, which makes it easy to set up APIs quickly. Since this project required file uploads, listing documents, downloading, and deleting, Express gave me full control over defining each endpoint clearly.

Express also has great middleware support. I was able to use libraries like **multer** for handling PDF uploads without any difficulty. Its folder structure is simple, so managing routes and database logic felt clean and organised. Another advantage is that Express works naturally with JavaScript, so I did not have to switch languages between the frontend and backend.

Overall, Express was a good choice because it is fast to work with, beginner-friendly, and flexible enough to handle all the requirements of this assignment.

**Q3. What database did you choose and why?**

I used **SQLite3** for this project. I chose it because it's very lightweight and doesn't require any complex setup or server configuration. Since the assignment is meant to run locally, SQLite is a perfect fit — it stores all the data in a single file and works out of the box.

For this project, I only needed to save basic metadata like the filename, file size, and upload date. SQLite handles this kind of small, straightforward data very well. It's also easy to integrate with Express, and the queries are simple to write.

Overall, SQLite3 was the most practical choice because it is simple, fast to set up, and more than enough for the scale of this assignment.

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

If the system had to support around 1,000 users instead of a single local user, I would make a few important changes to handle higher traffic and larger amounts of data. First, I would move from SQLite to a more robust database like **PostgreSQL or MySQL**, since they handle multiple concurrent connections much better.

I would also avoid storing files locally on the server. Instead, I'd use a cloud storage service like **AWS S3**, which is more reliable for storing user documents and scales easily as the number of uploads grows.

For user security, I'd add proper authentication using **JWT (JSON Web Tokens)**. JWT would allow each user to log in and securely access only their own documents without needing to maintain server-side sessions.

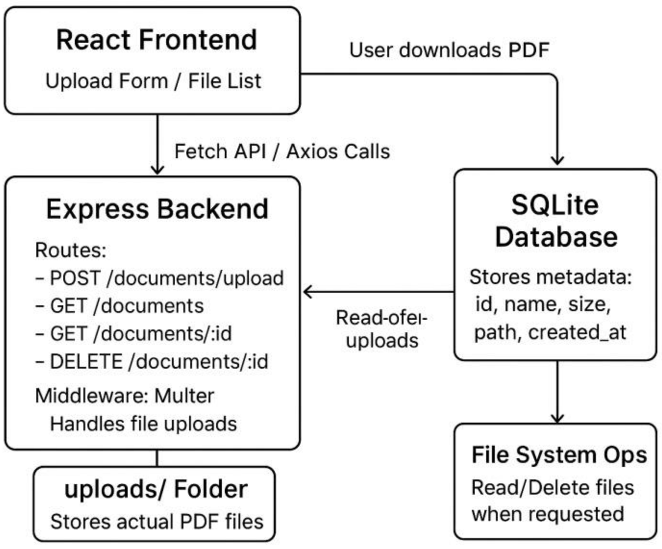
On the backend, I'd also consider running multiple Express instances with a load balancer and using caching tools like Redis to improve performance.

## 2. Architecture Overview

**Flow Between Frontend, Backend, Database, and File Storage**

I kept the architecture straightforward since this is a small full-stack application. Here is the overall flow:

1. **React Frontend**
   - The user uploads a PDF or interacts with the document list.
   - React sends requests to the Express backend using fetch/axios.
2. **Express Backend**
   - Receives upload, download, list, and delete requests.
   - Uses **Multer** to handle PDF uploads.
   - Stores each file inside the uploads/ folder.
   - Saves metadata (filename, size, path, date) into the SQLite database.
3. **SQLite3 Database**
   - Stores the document details.
   - Helps backend return the list of documents or find a document by ID.
4. **File Storage (uploads folder)**
   - Holds the actual PDF files that users upload.
   - Backend uses the file path to download or delete the file.



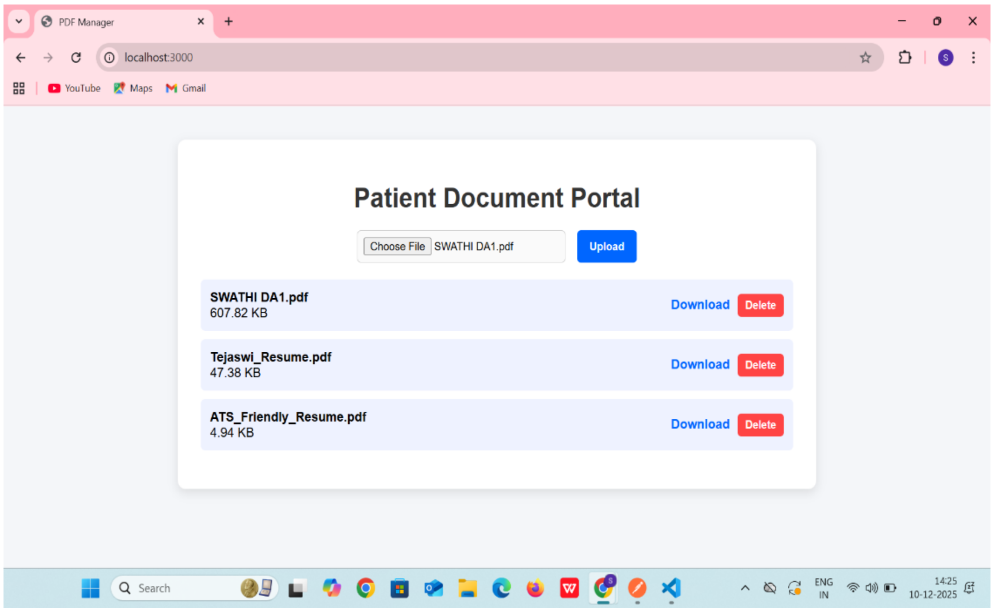
**3. API Specification**

For each of the following endpoints, provide:

● URL and HTTP method

● Sample request & response

● Brief description



SWATHI DA1.pdf
607.82 KB

Tejaswi_Resume.pdf
47.38 KB

ATS_Friendly_Resume.pdf
4.94 KB

**API Specification**

This API allows users to **upload**, **retrieve**, **view**, and **delete** documents stored on the server.

**1.GET /documents**

**Description**

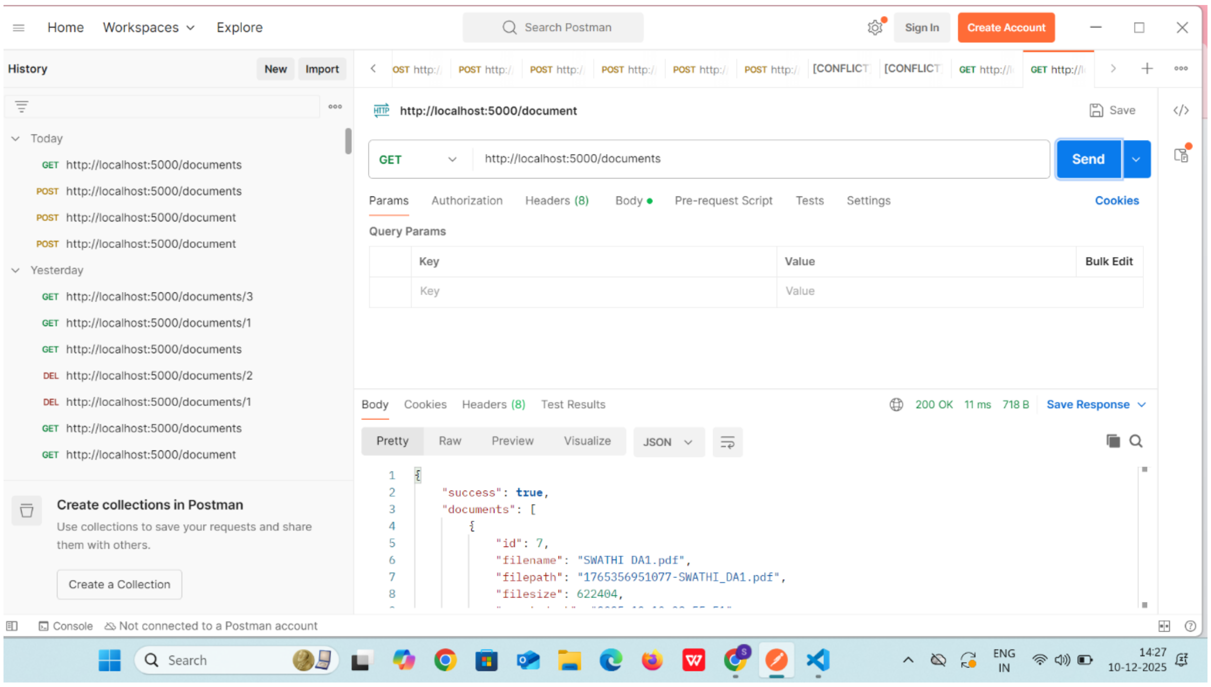
Returns a list of all uploaded documents with metadata such as ID, filename, size, and creation time.

**Sample Request**

GET http://localhost:5000/documents

**Sample Response**

```
{
  "success": true,
  "documents": [
    {
      "id": 8,
      "filename": "Modern_OnePage_Resume.pdf",
      "filepath": "1765357146674-Modern_OnePage_Resume.pdf",
      "filesize": 5440,
      "created_at": "2025-12-10 08:59:06"
    },
    {
      "id": 7,
      "filename": "SWATHI DA1.pdf",
      "filepath": "1765356951077-SWATHI_DA1.pdf",
      "filesize": 622404,
      "created_at": "2025-12-10 08:55:51"
    }
  ]
}
```



http://localhost:5000/document

GET http://localhost:5000/documents

Query Params

| Key | Value |
|---|---|
| Key | Value |

History — Today:

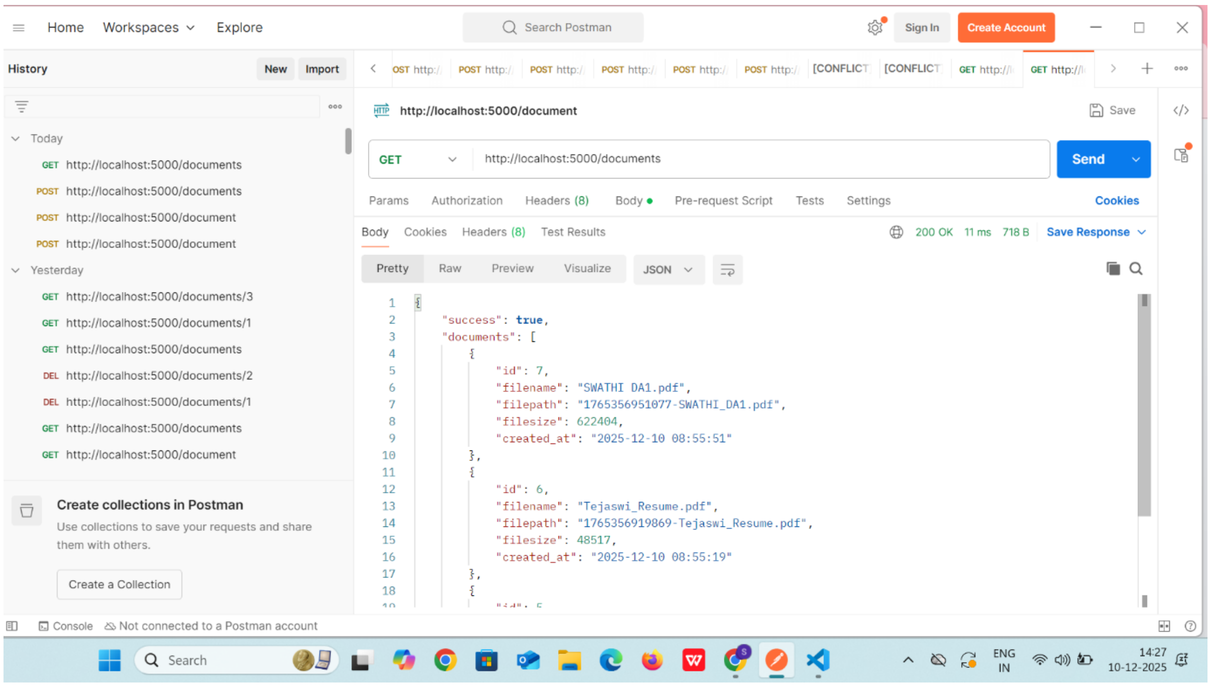
```
GET  http://localhost:5000/documents
POST http://localhost:5000/documents
POST http://localhost:5000/document
POST http://localhost:5000/document
```

Yesterday:

```
GET  http://localhost:5000/documents/3
GET  http://localhost:5000/documents/1
GET  http://localhost:5000/documents
DEL  http://localhost:5000/documents/2
DEL  http://localhost:5000/documents/1
GET  http://localhost:5000/documents
GET  http://localhost:5000/document
```

Body  Cookies  Headers (8)  Test Results    200 OK  11 ms  718 B

```
{
    "success": true,
    "documents": [
        {
            "id": 7,
            "filename": "SWATHI DA1.pdf",
            "filepath": "1765356951077-SWATHI_DA1.pdf",
            "filesize": 622404,
```



GET http://localhost:5000/documents

Body  Cookies  Headers (8)  Test Results    200 OK  11 ms  718 B

```
{
    "success": true,
    "documents": [
        {
            "id": 7,
            "filename": "SWATHI DA1.pdf",
            "filepath": "1765356951077-SWATHI_DA1.pdf",
            "filesize": 622404,
            "created_at": "2025-12-10 08:55:51"
        },
        {
            "id": 6,
            "filename": "Tejaswi_Resume.pdf",
            "filepath": "1765356919869-Tejaswi_Resume.pdf",
            "filesize": 48517,
            "created_at": "2025-12-10 08:55:19"
        },
        {
```

## 2. Get Document by ID

### GET /documents/{id}

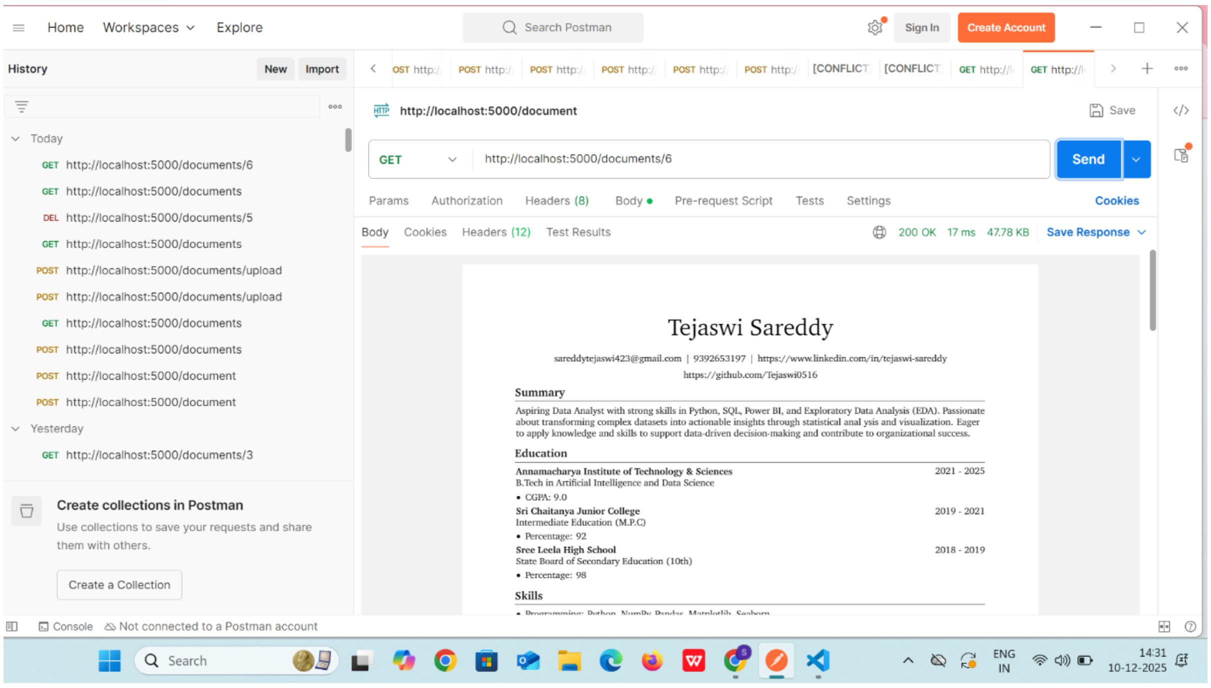
### Description

Fetches the file with the specified ID and returns it as a downloadable file.

### Sample Request

GET http://localhost:5000/documents/6

**Sample Response:**



Today

GET http://localhost:5000/documents/6
GET http://localhost:5000/documents
DEL http://localhost:5000/documents/5
GET http://localhost:5000/documents
POST http://localhost:5000/documents/upload
POST http://localhost:5000/documents/upload
GET http://localhost:5000/documents
POST http://localhost:5000/documents
POST http://localhost:5000/document
POST http://localhost:5000/document

Yesterday

GET http://localhost:5000/documents/3

http://localhost:5000/document

GET http://localhost:5000/documents/6

200 OK 17 ms 47.78 KB

Tejaswi Sareddy

sareddytejaswi423@gmail.com | 9392653197 | https://www.linkedin.com/in/tejaswi-sareddy
https://github.com/Tejaswi0516

**Summary**

Aspiring Data Analyst with strong skills in Python, SQL, Power BI, and Exploratory Data Analysis (EDA). Passionate about transforming complex datasets into actionable insights through statistical anal ysis and visualization. Eager to apply knowledge and skills to support data-driven decision-making and contribute to organizational success.

**Education**

**Annamacharya Institute of Technology & Sciences** 2021 - 2025
B.Tech in Artificial Intelligence and Data Science
- CGPA: 9.0

**Sri Chaitanya Junior College** 2019 - 2021
Intermediate Education (M.P.C)
- Percentage: 92

**Sree Leela High School** 2018 - 2019
State Board of Secondary Education (10th)
- Percentage: 98

**Skills**

- Programming: Python, NumPy, Pandas, Matplotlib, Seaborn

14:31
10-12-2025

# 3. Upload a Document

## POST /documents/upload

## Description

Uploads a document (PDF) to the server using multipart/form-data.

## Sample Request

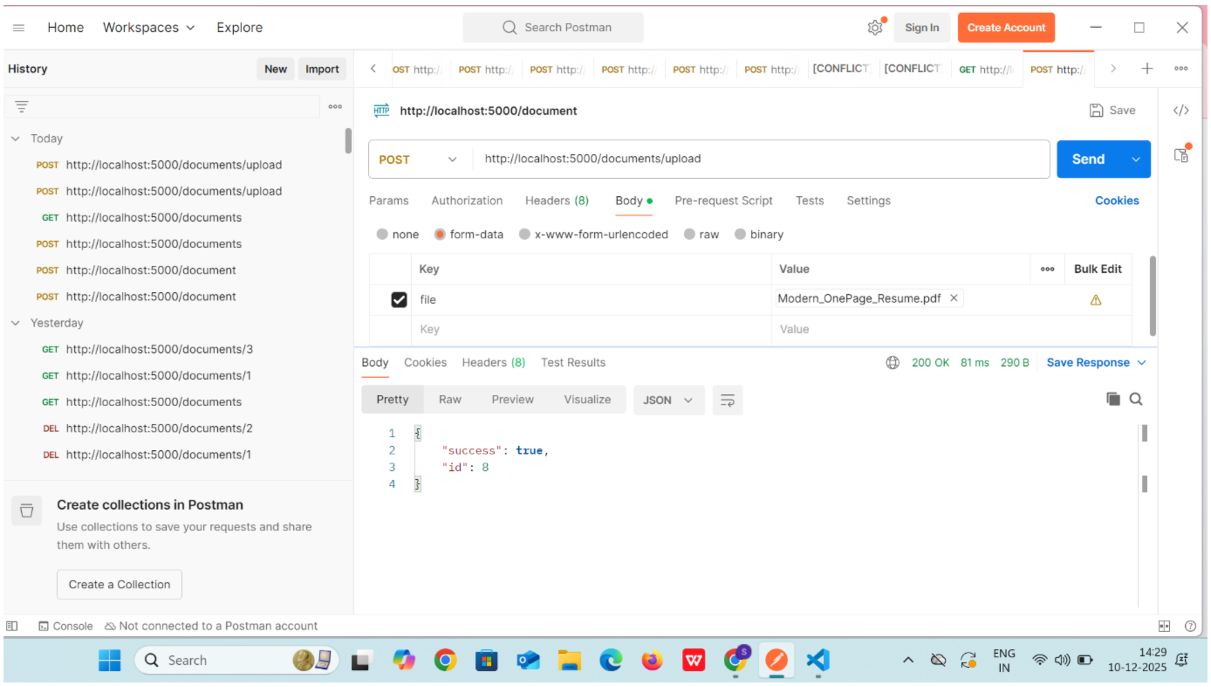
```
POST http://localhost:5000/documents/upload
```

### Body (form-data):

| Key | Type | Value |
|---|---|---|
| file | File | Modern_OnePage_Resume.pdf |

## Sample Response

```
{
  "success": true,
  "id": 8
}
```


History

Today
POST http://localhost:5000/documents/upload
POST http://localhost:5000/documents/upload
GET http://localhost:5000/documents
POST http://localhost:5000/documents
POST http://localhost:5000/document
POST http://localhost:5000/document
Yesterday
GET http://localhost:5000/documents/3
GET http://localhost:5000/documents/1
GET http://localhost:5000/documents
DEL http://localhost:5000/documents/2
DEL http://localhost:5000/documents/1

http://localhost:5000/document

POST http://localhost:5000/documents/upload

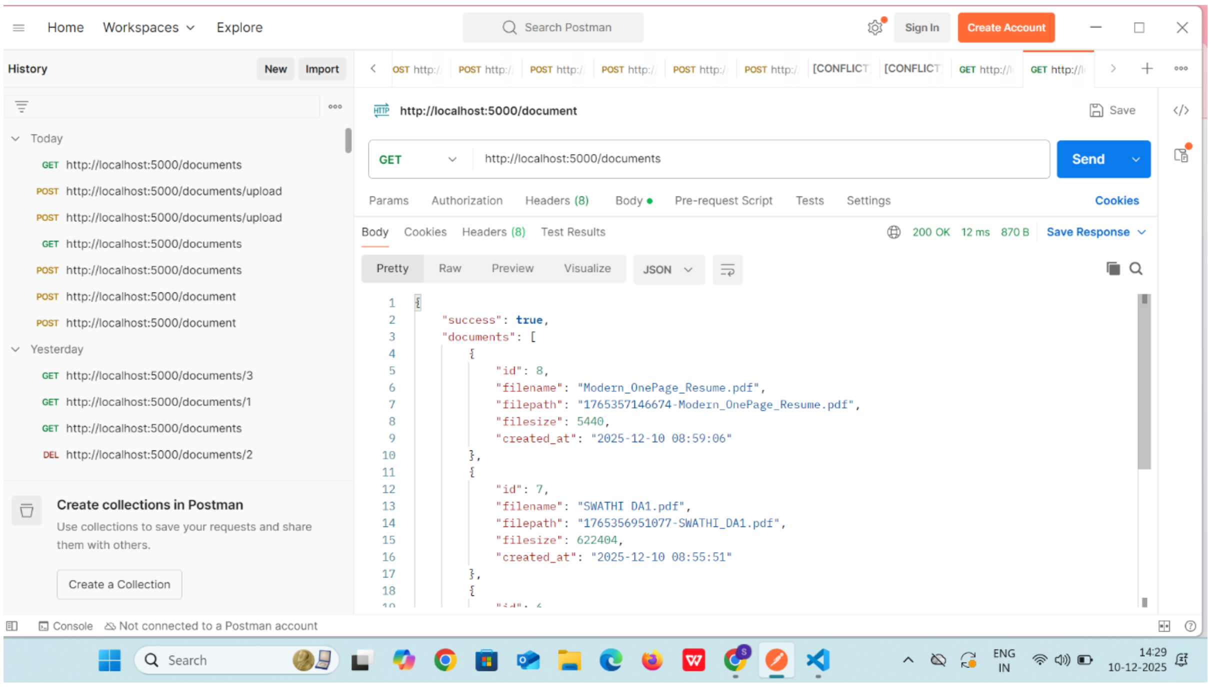
Body: form-data

| Key | Value |
|---|---|
| file | Modern_OnePage_Resume.pdf |

200 OK 81 ms 290 B

```
{
    "success": true,
    "id": 8
}
```

**Modern OnePage_Resume.pdf uploaded Successfully**


History

Today
GET http://localhost:5000/documents
POST http://localhost:5000/documents/upload
POST http://localhost:5000/documents/upload
GET http://localhost:5000/documents
POST http://localhost:5000/documents
POST http://localhost:5000/document
POST http://localhost:5000/document
Yesterday
GET http://localhost:5000/documents/3
GET http://localhost:5000/documents/1
GET http://localhost:5000/documents
DEL http://localhost:5000/documents/2

http://localhost:5000/document

GET http://localhost:5000/documents

200 OK 12 ms 870 B

```
{
    "success": true,
    "documents": [
        {
            "id": 8,
            "filename": "Modern_OnePage_Resume.pdf",
            "filepath": "1765357146674-Modern_OnePage_Resume.pdf",
            "filesize": 5440,
            "created_at": "2025-12-10 08:59:06"
        },
        {
            "id": 7,
            "filename": "SWATHI DA1.pdf",
            "filepath": "1765356951077-SWATHI_DA1.pdf",
            "filesize": 622404,
            "created_at": "2025-12-10 08:55:51"
        },
        {
```

## 4. Delete a Document

**DELETE /documents/{id}**

**Description**

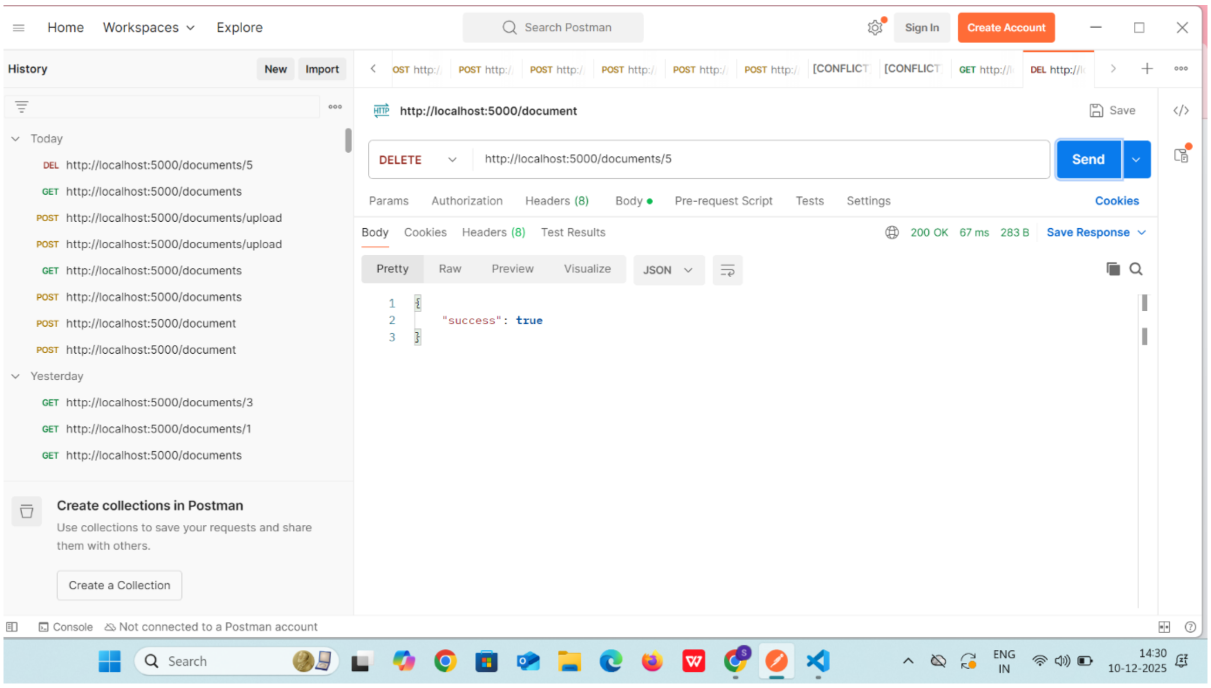
**Deletes a document by ID from the server's storage and database.**

**Sample Request**

**DELETE http://localhost:5000/documents/5**

**Sample Response**

```
{
  "success": true
}
```



History

Today
- DEL http://localhost:5000/documents/5
- GET http://localhost:5000/documents
- POST http://localhost:5000/documents/upload
- POST http://localhost:5000/documents/upload
- GET http://localhost:5000/documents
- POST http://localhost:5000/documents
- POST http://localhost:5000/document
- POST http://localhost:5000/document

Yesterday
- GET http://localhost:5000/documents/3
- GET http://localhost:5000/documents/1
- GET http://localhost:5000/documents

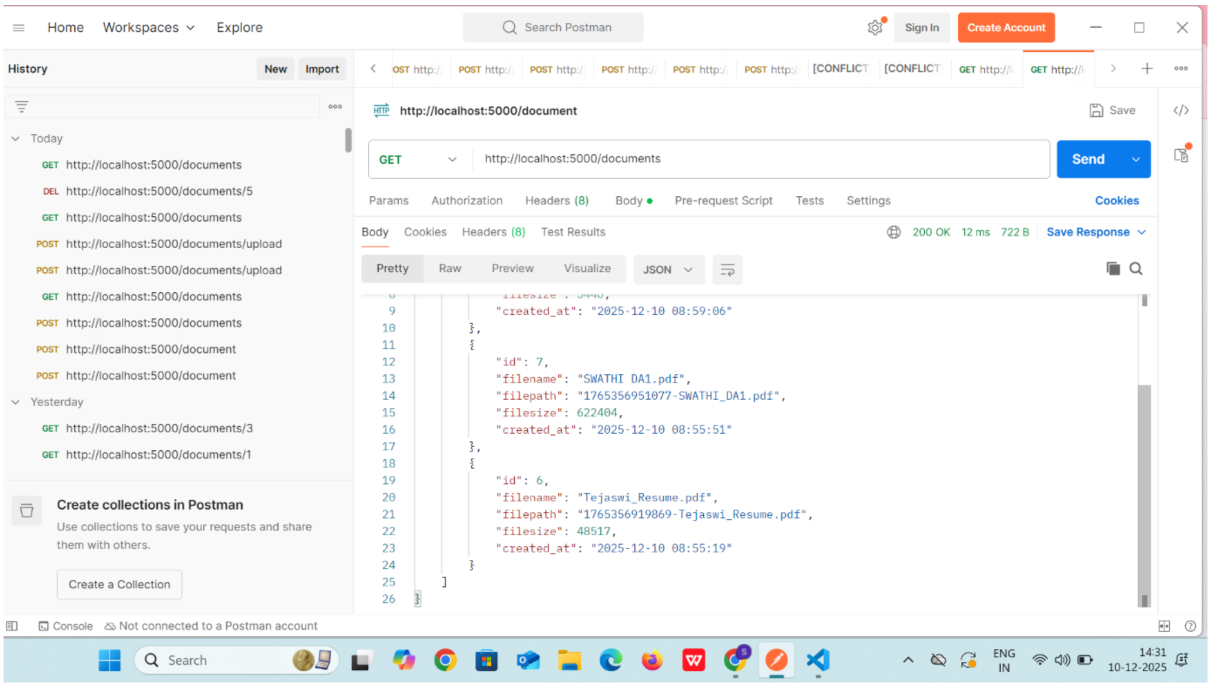
http://localhost:5000/document

DELETE http://localhost:5000/documents/5

200 OK 67 ms 283 B

```
{
    "success": true
}
```

**Deleted Successfully**



History

Today
- GET http://localhost:5000/documents
- DEL http://localhost:5000/documents/5
- GET http://localhost:5000/documents
- POST http://localhost:5000/documents/upload
- POST http://localhost:5000/documents/upload
- GET http://localhost:5000/documents
- POST http://localhost:5000/documents
- POST http://localhost:5000/document
- POST http://localhost:5000/document

Yesterday
- GET http://localhost:5000/documents/3
- GET http://localhost:5000/documents/1

http://localhost:5000/document

GET http://localhost:5000/documents

200 OK 12 ms 722 B

```
            "created_at": "2025-12-10 08:59:06"
        },
        {
            "id": 7,
            "filename": "SWATHI DA1.pdf",
            "filepath": "1765356951077-SWATHI_DA1.pdf",
            "filesize": 622404,
            "created_at": "2025-12-10 08:55:51"
        },
        {
            "id": 6,
            "filename": "Tejaswi_Resume.pdf",
            "filepath": "1765356919869-Tejaswi_Resume.pdf",
            "filesize": 48517,
            "created_at": "2025-12-10 08:55:19"
        }
    ]
}
```

**Create Document Metadata**

**POST /document**

**Description**

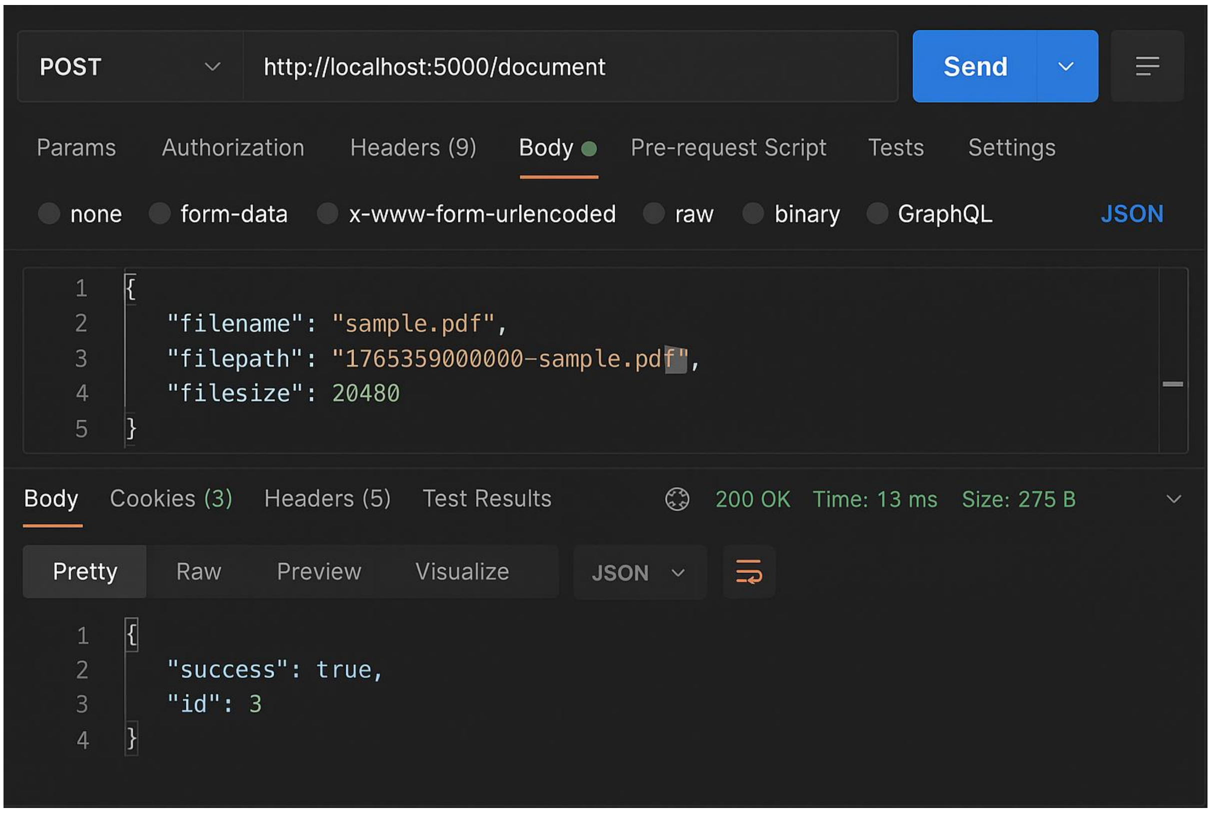
**Adds a document entry in the database (may be legacy or test endpoint).**

**Sample Request**

```
{
  "filename": "sample.pdf",
  "filepath": "12345-sample.pdf",
  "filesize": 10240
}
```

**Sample Response**

```
{
  "success": true,
  "id": 3
}
```



```
{
    "filename": "sample.pdf",
    "filepath": "1765359000000-sample.pdf",
    "filesize": 20480
}
```

```
{
    "success": true,
    "id": 3
}
```

## Data Flow Description

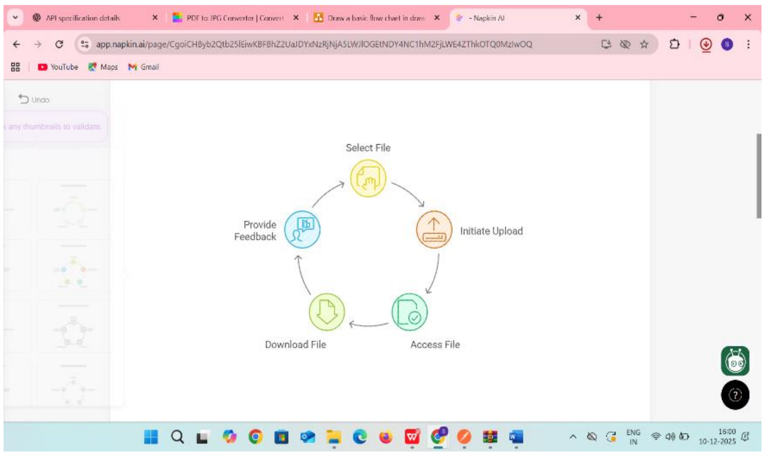
**1. When a File is Uploaded**

**Step-by-step flow:**

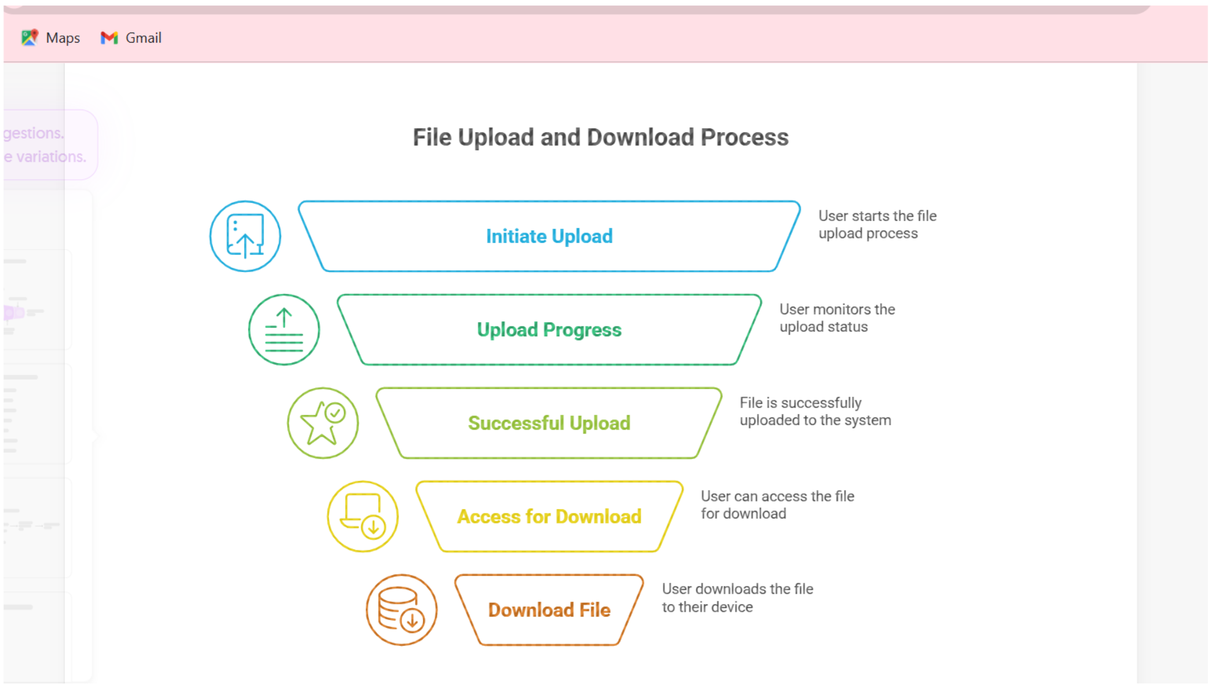
1. **User selects a file**
   - **The user chooses a file from their device (example: PDF, image, Word file).**
2. **Client (Browser/Postman/UI) sends request**

    - The file is sent to the server using a POST /documents/upload request.
    - The request uses multipart/form-data, which allows files to be uploaded.

3. Server receives the file
    - The server takes the uploaded file from the request.
    - A unique filename is generated to prevent duplicates.
4. Server stores the file
    - The file is saved in a physical folder (e.g., /uploads/).
5. Server creates metadata
    - After saving, the server collects information such as:
        - original file name
        - stored file name
        - file size
        - file type
        - upload time
        - file path
6. Server stores metadata in the database
    - A new entry is created in the Documents table.
    - The database returns a document ID.
7. Server sends response back to user
    - Response includes:
        - status (success)
        - document ID
        - original file name



8. **User sees upload success message**


Maps
Gmail
gestions.
e variations.
File Upload and Download Process
Initiate Upload
User starts the file upload process
Upload Progress
User monitors the upload status
Successful Upload
File is successfully uploaded to the system
Access for Download
User can access the file for download
Download File
User downloads the file to their device

---

**2. When a File is Downloaded**

**Step-by-step flow:**

1. **User requests download**
   - **The user enters the document ID or clicks a download button.**
   - **Client sends GET /documents/{id} to the server.**
2. **Server receives the download request**
   - **The server extracts the requested document ID.**
3. **Server checks metadata in the database**
   - **It looks up:**
     - **stored filename**
     - **file path**
     - **file type**
     - **whether the file exists**
4. **Server fetches file from file system**
   - **Reads the actual file from /uploads/ using the stored path.**
5. **Server prepares the file for download**

- **Sets headers such as:**
    - **Content-Type: application/pdf (or based on file)**
    - **Content-Disposition: attachment; filename="original_name"**

6. **Server sends the actual file bytes**
    - **The file is streamed back to the client.**
7. **User receives the file**
    - **The browser saves it or opens it, depending on the file type.**

---

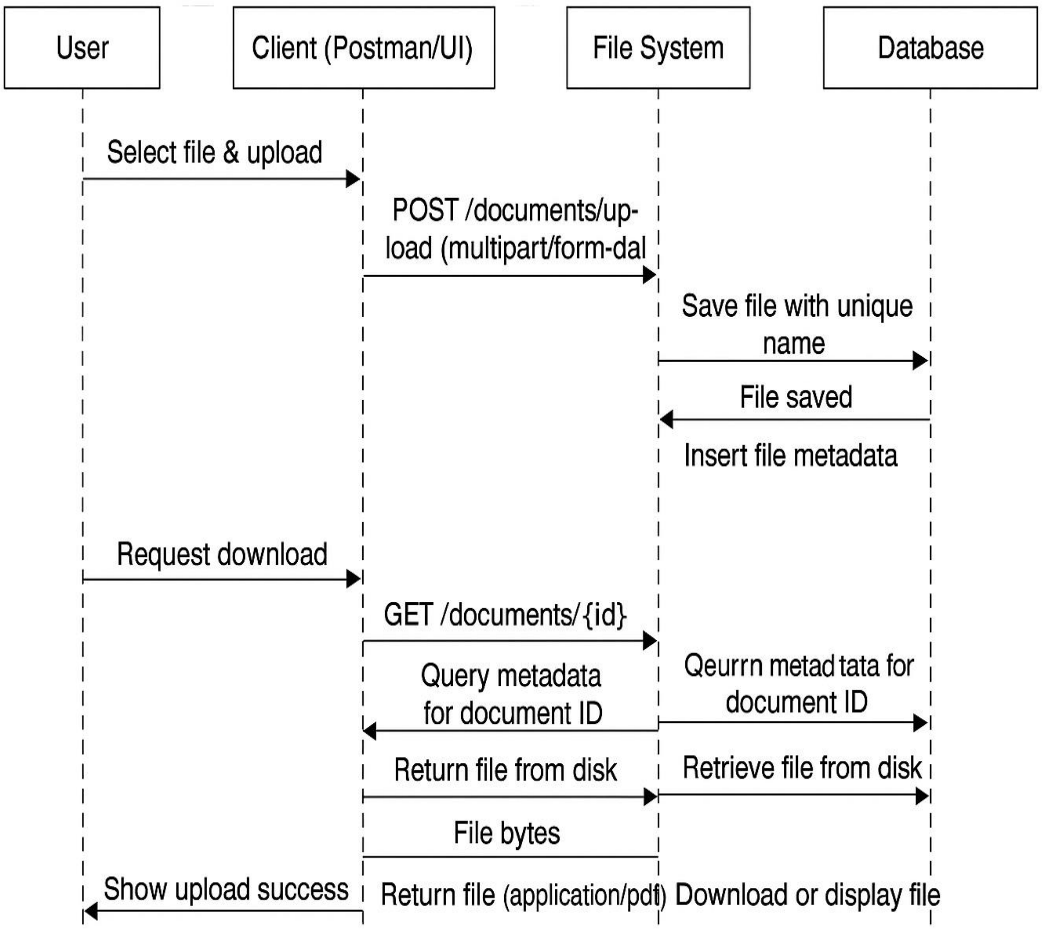
**✓ In summary**

**Upload Flow**

**User → Client → Server → File System → Database → Upload Response**

**Download Flow**

**User → Client → Server → Database → File System → File Response**



## Assumptions

While making this project, I made a few basic assumptions so that the system works smoothly:

1. **File Size**
   I assumed that the files uploaded will not be very large. Something like normal PDF or image sizes (a few MBs).
2. **File Types**
   I assumed users will mostly upload common file types like PDF, JPG, PNG, DOCX, etc.
3. **Storage**
   I assumed the server has enough space available to store the uploaded files. I also assumed the upload folder already exists and has the right permissions.
4. **Authentication**
   For this project, I assumed that login or authentication is not required. Anyone can upload and download using the API.
5. **Database Availability**
   I assumed the database is always connected and working, so saving and reading metadata will not fail.
6. **Multiple Users**
   I assumed that even if many users upload or download at the same time, the server can handle it.
7. **Error Handling**
   I assumed that users will upload proper files and use valid IDs. Only basic validation is added.
8. **Network Stability**
   I assumed the network between the client, server, and database remains stable and requests will not randomly fail.
9. **File Retention**
   I assumed that files will stay on the server unless someone manually deletes them.