Q1. What frontend framework did you use and why?

- **React.js**
- React allows fast, modular UI development using reusable components and hooks (useState, useEffect).
- It provides easy integration with REST APIs and handles state updates efficiently.
- The axios library was used for handling HTTP requests.

Q2. What backend framework did you choose and why?

- **Express.js**
- Lightweight and minimalist Node.js web framework ideal for building RESTful APIs.
- Integrates well with middleware such as multer for file handling.
- Allows asynchronous handling and is fast to prototype with.

Q3. What database did you choose and why?

- **PostgreSQL**
- A powerful open-source relational database that supports complex queries and strong data integrity.
- It is well-suited for structured data like file metadata and is scalable for production use.
- Simple to set up and integrates smoothly with Node.js via the pg library.

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

- Add user authentication and associate files with specific users.
- Store uploaded files in cloud storage (e.g., AWS) instead of local file system.
- Use connection pooling for PostgreSQL to manage concurrent queries efficiently.
- Add pagination and search filters for the /documents endpoint.
- Implement rate-limiting to prevent misuse and improve performance.

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

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

Upload PDF Flow

- User selects a PDF and clicks the "Upload" button on the frontend.
- The React frontend (FE) sends a POST /documents/upload request with the file using FormData.
- The Express backend (BE) uses multer to store the file in the uploads/ directory.
- The backend inserts metadata (filename, path, file size, and timestamp) into the PostgreSQL DB (DB).
- The DB confirms that the data is inserted successfully.
- The backend sends a success response to the frontend.
- The frontend shows a message like "Upload successful" to the user.

View Uploaded PDFs Flow

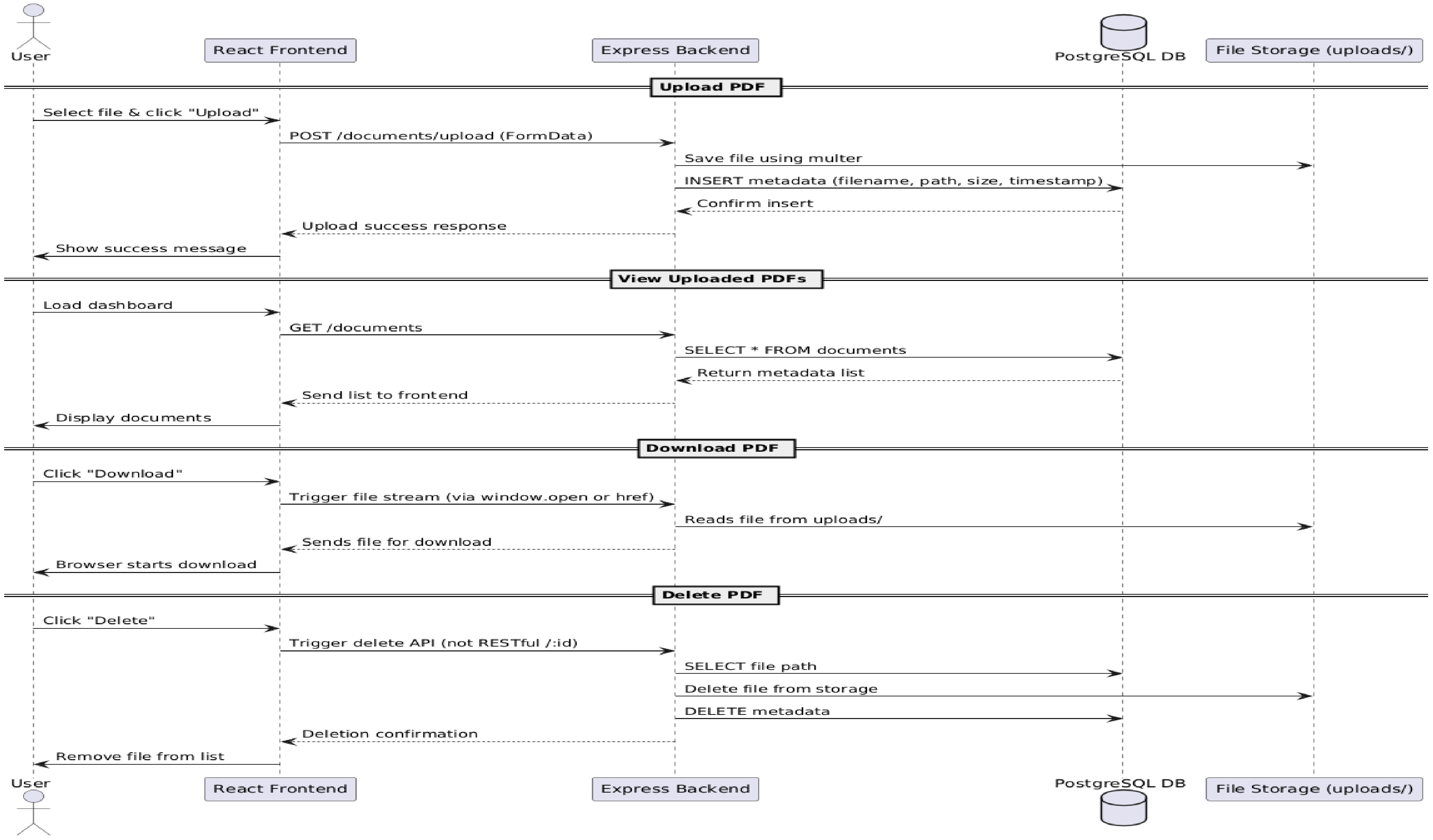
- User opens the dashboard or refreshes the page.
- The frontend makes a GET /documents request to fetch the list of uploaded files.
- The backend queries the database for all documents.
- The DB returns a list of document metadata (e.g., original filename, size, id).
- The backend sends the list back to the frontend.
- The frontend displays the documents in a list to the user.

Download PDF Flow

- User clicks a "Download" button next to a file.
- The frontend triggers a download using a file URL (e.g., via window.open() or <a href=...>).
- The backend reads the file from the local uploads/ folder.
- The backend sends the file as a response (with proper headers for download).
- The browser automatically starts the file download for the user.

Delete PDF Flow

- User clicks a "Delete" button next to a document.
- The frontend calls a delete API (not necessarily /documents/:id, possibly via query or body param).
- The backend first retrieves the file path from the database.
- Then it deletes the file from the local storage (uploads/).
- The backend removes the document record from the database.
- It sends a deletion confirmation response to the frontend.
- The frontend updates the UI and removes the deleted file from the list.


User
React Frontend
Express Backend
PostgreSQL DB
File Storage (uploads/)
Upload PDF
Select file & click "Upload"
POST /documents/upload (FormData)
Save file using multer
INSERT metadata (filename, path, size, timestamp)
Confirm insert
Upload success response
Show success message
View Uploaded PDFs
Load dashboard
GET /documents
SELECT * FROM documents
Return metadata list
Send list to frontend
Display documents
Download PDF
Click "Download"
Trigger file stream (via window.open or href)
Reads file from uploads/
Sends file for download
Browser starts download
Delete PDF
Click "Delete"
Trigger delete API (not RESTful /:id)
SELECT file path
Delete file from storage
DELETE metadata
Deletion confirmation
Remove file from list
User
React Frontend
Express Backend
PostgreSQL DB
File Storage (uploads/)