Frontend and Backend Take-home assignment

App Name: LinkVault

Objective

Design and implement a **full stack web application** that allows users to upload **text or files** and share them securely with others using a **generated link**. Access to the uploaded content must be **restricted strictly to users who possess the link**, similar to link-based access in systems like Google Drive, Pastebin and TinyURL.

Problem Statement

You are required to build a **Pastebin-like web application** where users can:

1. Upload **plain text** or **any type of file** (either one for a single share)
2. Receive a **unique, shareable URL** after upload.
3. Share this URL with others who can:
    - View and copy the text, or
    - Download the uploaded file.
4. Ensure that the **content is not accessible without the exact link**.
5. Automatically **expire and delete content** after a specified duration.

No authentication or login system is required for the base implementation.

Functional Requirements

1. Upload Functionality

- Users should be able to:
    - Upload plain text, **or**
    - Upload a file (any format).
- Only one of the two (text or file) is required per upload.

2. Link Generation

- Upon successful upload, the server must generate a **unique, hard-to-guess URL**.
- This URL should be returned to the user and displayed on the frontend.
- The URL must be shareable and reusable until the content expires.

3. Access Control

- Content must be **accessible only via the generated link**.
- There should be **no public listing**, search, or browsing of uploaded content.

- Attempting to access content without a valid link should result in:
    - A `403 (Bad Request)` response.

4. Expiry Handling

- Users may optionally specify an **expiry date and time** during upload.
- If no expiry is specified:
    - The content must **expire 10 minutes after upload by default**.
- After expiry:
    - The content should no longer be accessible.
    - The system should handle expired links gracefully.

5. Content Retrieval

- For text uploads:
    - Display the text in a readable format.
    - Provide a **copy-to-clipboard** option.
- For file uploads:
    - Provide a **download option**.

Non-Functional Requirements

- The application should be:
    - Secure against direct URL guessing.
    - Efficient in handling uploads and downloads.
    - Cleanly structured with separation of concerns (frontend, backend, database).
- API responses should follow proper HTTP status codes.
- Input validation must be implemented on both frontend and backend.
- The UI of the application must be good (easy-to-use and clean)

API Expectations (High-Level)

Students are expected to design RESTful APIs, for example:

- Upload endpoint
- Content retrieval endpoint via unique ID
- Optional cleanup mechanism for expired content

Exact API design decisions are left to the student.

Additional Features (Optional / Bonus / To make this a Resume-ready project)

Students may implement all or any of the following:

- Password-protected links
- One-time view links
- Maximum download/view count
- Manual delete option
- Authentication and user accounts
- File size limits and type validation
- Background job for automatic deletion of expired content
- User-based access control after authentication is implemented

---

Tech Stack Constraints

Students must use the following stack:

Frontend

- **React + Vite**
- **Tailwind CSS**

Backend

- **Node.js**
- **Express.js**

Database

- Any database of the student’s choice
  (e.g., MongoDB, PostgreSQL, MySQL, SQLite, etc.)

Hint: Use an Object Store like **Firebase Storage** bucket and obtain the link to the uploaded files and store that in the DB. (Maybe this can be asynchronously using a cron job or synchronously (student’s choice)).

Deliverables

- Source code (frontend and backend folders clearly separate from each other)
- Database schema or model definitions
- README file including (strictly must not be AI generated):
  - Setup instructions
  - API overview
  - Design decisions

    - Assumptions and limitations
- Data flow diagram from the user upload to DB storage (The High-Level architecture of the application on Pen and Paper or using Online tools like Whimsical)
- **All the above deliverables are to be uploaded to GitHub and the repository link has to be submitted using the Google Form provided on Moodle.**
- **Do not add node_modules folder to git. Use the following .gitignore files in case your folder doesn't have them pre-generated by npm.**

https://github.com/github/gitignore/blob/main/Node.gitignore