Algoma UNIVERSITY

Final Project

Total Marks: 35 marks

Course: COSC2956: Internet Tools

Topic: Secure File Hosting Web Application

Objective: The goal of this project is to design and implement a secure full-stack web application that allows users to register, log in, upload, download, and delete files. Each uploaded file must be marked as either public (visible to all users) or private (accessible only via a shareable link). All data must be dynamically managed using a database; no information should be hardcoded.

Functional Requirements: The application should have 5 functional requirements.

1. User Registration
2. File Upload
3. File listing and Access Control
4. File Deletion
5. Database Integration

1. User Registration and Authentication

The application must allow users to register with a unique email, username, and password.

- Passwords must be securely hashed before storage.
- If a user attempts to register using an email that already exists in the system, an error message must be displayed. (**Your login information should be stored in database**)
- During login, credentials should be verified, and a token should be issued upon successful authentication.
- Tokens must be stored securely in the local Storage of the browser, and all protected routes must verify the token before allowing access.
- This functionality should strictly be implemented in backend. Information should flow from frontend to backend.

2. File Upload

Authenticated users should be able to see a new HTML page, where they are able to upload files with a privacy option (public or private).

- Uploaded files should be stored in an /uploads directory.
- Metadata (filename, size, uploader, privacy, and timestamp) must be stored in the database.
- Files exceeding a defined size or unsupported formats should be rejected gracefully (Supported format is .pdf and .mp4, and max file size should be 20 MB).

3. File Listing and Access Control

Public files should appear on a Downloads page visible to all users. At this page, any user should be able to download the public file (irrespective of who the owner is). Private files must not be listed publicly but can be accessed through a unique shareable link.

Logged-in users should have a My Files dashboard showing their uploaded files with options to download or delete them (May be two buttons or whatever you like).

4. File Deletion

Users must be able to delete only their own files. Deleting a file removes its record from the database and deletes the actual file from the uploads directory. Unauthorized deletion attempts should return an appropriate error message. You should only be able to delete the file from MyFiles dashboard as listed in step 3.

5. Database Schema

Use MongoDB, PostgreSQL, or MySQL. Use whatever database you like. The database should individually store two pieces of information. One is **authentication information** (id, username, email, hashed password, created_at). Second is **file metadata** (id, filename, path, size, privacy, uploaded_by, uploaded_at). It's upto you how you design database, but all your front end should get the data from backend and your html pages should be dynamic.

6. Frontend Requirements

Frontend must be implemented using HTML, CSS, and JavaScript (You can also use react if you want). Required pages include Register, Login, Upload, My Files, and Downloads. All data should be dynamically fetched from backend APIs using fetch() or Axios, with no hardcoded content.

- In case of register/login, the information should go from register/login page to backend API.
- To upload file, the information should be stored in database in the backend through backend API.
- For MyFiles page, the information should come directly from database.
- Again for downloads, the information should come directly from the database in the backend.

7. Suggested API Endpoints

POST /api/register – Register a new user
POST /api/login – Log in and receive a token
POST /api/upload – Upload a file (auth required)
GET /api/public-files – Retrieve all public files

GET /api/my-files – Retrieve logged-in user’s files
GET /api/files/:id/download – Download file (permission check)
DELETE /api/files/:id – Delete file (owner only)

8. Security and Validation

All inputs must be validated on both frontend and backend. Tokens must be verified on all protected routes. File names must be sanitized, file types restricted, and size limits enforced. Sensitive information must never be exposed in API responses.

9. Deliverables

1. GitHub repository with folder structure (/backend, /frontend, /uploads etc.)
2. README.md with setup instructions and endpoint documentation
3. Demo video: register → login → upload → view → delete → logout
4. Make your code in such a way that if I must run it, I will simply run the server, and use it directly on my browser. Clearly write steps in readme what do I have to do to run this code on my laptop. If the steps are not clear or I won’t be able to run it on my laptop, marks will be deducted.

10. Marking Breakdown (Total = 35%)

- Authentication & Authorization – 10%
- File Upload & Access Control – 10%
- File Deletion – 5%
- Database Integration – 5%
- Frontend Integration – 5%