COLLEGE CODE : 8203

COLLEGE : AVC COLLEGE OF ENGINEERING

DEPARTMENT : INFORMATION TECHNOLOGY

STUDENT NM-ID: 50186AA220AC61191B545EC1FD3CC1DF

ROLL NO : 23IT74

DATE :08/09/2025

Completed the project named as phase 1

TECHNOLOGY PROJECT NAME:File upload manager

SUBMITTED BY,

NAME:PERAMANI B

MOBILE:8610226879

Problem Statement

In today's digital applications, efficient and secure file handling is essential for user experience and data management. However, many systems lack a standardized solution for uploading, storing, retrieving, and managing files. Developers often face challenges such as:

- Handling large files without performance issues.
- Ensuring scalability and reliability in file storage.
- Providing a secure way to upload, access, and delete files.
- Managing metadata and file versions effectively.
- Integrating with both local storage and database-driven storage like MongoDB GridFS.

Without a dedicated and well-structured solution, applications risk inefficient storage utilization, poor performance, security vulnerabilities, and difficulties in file retrieval.

This project aims to develop a **File Upload Manager** using **Node.js, Express, Multer, and MongoDB/GridFS** that allows users to seamlessly upload, fetch, and delete files while providing file URLs or metadata in responses. The solution ensures secure storage, easy access, and robust API support, addressing the common challenges faced in file management within modern web applications.

Users & Stakeholders

Users

1. **End Users / Application Users** – Individuals who upload, fetch, and delete files (e.g., students, employees, customers).
2. **Application Developers** – Developers who integrate the File Upload Manager into their web or mobile applications.
3. **Administrators** – Users with elevated privileges to manage stored files, monitor usage, and ensure security compliance.

Stakeholders

1. **Project Owner / Client** – The individual or organization requesting the File Upload Manager to solve their file management challenges.
2. **Development Team** – Backend and frontend developers building the system using Node.js, Express, Multer, and MongoDB/GridFS.
3. **Database Administrators (DBAs)** – Responsible for maintaining and optimizing the MongoDB/GridFS storage.
4. **IT Security Team** – Ensures that file uploads are secure, scanned for vulnerabilities, and comply with security policies.

5. **End User Organizations** – Companies, educational institutions, or platforms that will integrate and rely on the File Upload Manager.
6. **System Administrators** – Manage servers, deployment, and availability of the application.

User Stories

End User:

- **As an End User,**
  I want to upload files (documents, images, videos, etc.) through the application
  so that I can store and access them later when needed.
- **As an End User,**
  I want to retrieve/download my previously uploaded files using a unique file URL
  so that I can easily share or reuse them.
- **As an End User,**
  I want to delete files I no longer need
  so that I can free up storage space and maintain an organized system.
- **As an End User,**
  I want to receive a confirmation message after uploading, fetching, or deleting a file
  so that I know whether my action was successful.

Application Developer:

- **As an Application Developer,**
  I want a simple and standardized API for uploading, retrieving, and deleting files
  so that I can integrate the File Upload Manager into different applications.
- **As an Application Developer,**
  I want to access file metadata (name, size, type, upload date)
  so that I can use this information in my application features.

Administrator:

- **As an Administrator,**
  I want to monitor file storage and usage statistics
  so that I can ensure efficient resource utilization.
- **As an Administrator,**
  I want to restrict file types and sizes during upload
  so that the system remains secure and optimized.
- **As an Administrator,**
  I want to remove unauthorized or harmful files from the system
  so that the application stays safe and compliant.

MVP [Minimum Viable Product] Features

1.File Upload

- Allow users to select and upload files via frontend.
- Support common file types (e.g., PDF, DOCX, JPG, PNG).
- Validate file size and type before upload.

2.File Storage

- Store uploaded files either locally (server folder) or in MongoDB GridFS.
- Generate a unique file ID or URL for each uploaded file.

3.File Retrieval (Download/Fetch)

- Provide an API endpoint to fetch files using file ID or URL.
- Return file metadata (name, size, type, upload date).

4.File Deletion

- Enable users to delete uploaded files via an API endpoint.
- Confirm deletion with a success/failure response.

5.API Responses & Metadata

- Ensure all API requests return proper responses (success/error).
- Provide file details (ID, name, size, type, upload date) in responses.

6.Basic Security & Validation

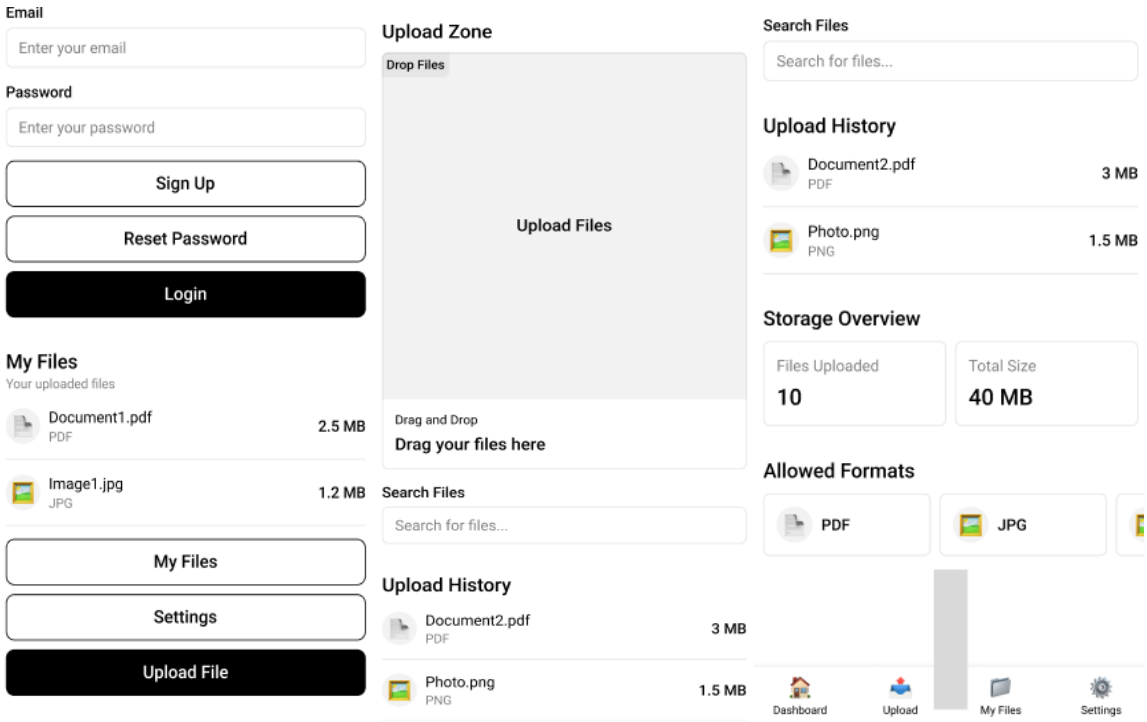
- Restrict invalid/malicious file types (e.g., .exe, .bat).
- Limit maximum file size to prevent server overload.

7.Basic User Interface (Frontend)

- Simple form for file upload (choose file → submit).
- List of uploaded files with options to download or delete.
- Display upload status messages (success/error).

Wireframe & API Endpoints

Mid Fidelity Wireframe:



Acceptance Criteria

1. File Upload

- User can select and upload a file via the frontend.
- System accepts only allowed file types (e.g., PDF, DOCX, JPG, PNG).
- System rejects disallowed file types (e.g., .exe, .bat) with an error message.
- Uploads exceeding the maximum size limit are rejected with a clear message.
- On successful upload, the system generates a unique file ID or URL.
- User receives confirmation (success/failure) for each upload.

2. File Storage

- Files are stored successfully in local storage or MongoDB GridFS.
- Metadata (file name, size, type, upload date) is saved with each file.
- Files remain retrievable after server restart.

3. File Retrieval (Download/Fetch)

- User can fetch a file using file ID or URL.
- Retrieved file opens/downloads without corruption.
- API returns correct file metadata (name, size, type, upload date).

- If file does not exist, system returns a “File not found” error.

4. File Deletion

- User can delete a file using file ID or URL.
- Deleted files are no longer retrievable.
- System confirms deletion with a success message.
- If a non-existent file is requested for deletion, an error message is shown.

5. API Responses & Metadata

- All API responses follow a consistent JSON format.
- Success responses include file ID/URL and metadata.
- Error responses include status code and descriptive message.

6. Basic Security & Validation

- System restricts unauthorized/malicious uploads.
- Maximum upload size is enforced (e.g., 10MB by default).
- Only authenticated users (if auth is included) can upload, fetch, or delete files.

7. User Interface (Frontend)

- Upload form allows file selection and submission.
- Uploaded files appear in a list with options to download or delete.
- System displays success/error messages for all actions.