FILE SHARING WEBSITE

Student Name: Rubesh Raman

Reg: 12115752

Summer Training Report File

School of Computer Science and Engineering

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

TABLE OF CONTENT

- Acknowledgement
- Abstract
- Introduction
- Objective
- Modularization Details
- Software Engineering Paradigm Applied
- System Requirement
- Flow Chart
- Screen Shots
- Future scope of Improvements
- Code

ACKNOWLEDGEMENT

I would like to sincerely express my heartfelt gratitude to my mentor and teacher **Mr. Satyam Kumar Jha** and institute **Fifth Force** who have played a significant role in the completion of this report. Their support, guidance, and assistance have been invaluable, and I am truly grateful for their contributions.

First and foremost, I would like to extend my deepest appreciation to my teacher, **Mr. Satyam Kumar Jha**. His expertise, guidance, and continuous encouragement throughout this **MERN Course** have been instrumental in shaping the direction and enhancing the quality of this project. I am indebted to them for their unwavering support.

I would also like to acknowledge the support received from **Fifth Force**. Their provision of necessary resources, including access to relevant materials and equipment that has been crucial in learning and ensuring the accurate content in MERN Stack.

Rubesh Raman

ABSTRACT

This project presents a MERN (MongoDB, Express.js, React.js, Node.js) stack application that enables users to upload files and securely store them in a cloud-based MongoDB database. The system utilizes Node.js and Express.js for the server-side implementation, while React.js is employed for the client-side development.

The primary functionality of the application revolves around allowing users to upload files of various formats. Upon file upload, the server securely saves the file in a MongoDB database hosted in the cloud.

To enhance the user experience, React.js is utilized for building the application's frontend. Users are presented with a simple Interface where they can easily upload their desired files.

Once a file is successfully uploaded, the system generates a unique download link, which is provided to the user for easy access to the recently file. Users can securely store and retrieve their files from anywhere, as long as they have access to the internet. The combination of MongoDB, Express.js, React.js, and Node.js provides a robust and efficient solution for managing and sharing files.

KEYWORDS: MERN stack, fie upload, MongoDB, Express.js, Node.js, React.js, cloud-based storage, user interface, file retrieval.uploaded

INTRODUCTION

The rapid growth of cloud-based technologies and web applications has revolutionized the way we store and share information. This project presents a MERN (MongoDB, Express.js, React.js, Node.js) stack application that allows users to upload files, which are then securely saved in a cloud-based MongoDB database.

The primary objective of this project is to provide users with a seamless and efficient file upload and retrieval experience. Using the power of Node.js and Express.js on the server-side, users can effortlessly upload their desired files through a user-friendly interface. Once the file upload is complete, the application generates a unique download link, enabling users to access and download the recently uploaded file.

To enhance the frontend development and user experience, React.js is utilized for building the application's interface. This JavaScript library allows for the creation of dynamic and responsive user interfaces, ensuring a smooth and intuitive file upload process.

The choice of MongoDB as the backend database offers numerous advantages, including scalability, flexibility, and seamless integration with cloud services. By using a cloud-based MongoDB database, the project ensures reliable and secure storage of uploaded files, accessible from anywhere with an internet connection.

In the end, this MERN stack project combines the power of MongoDB, Express.js, React.js, and Node.js to create a robust and user-friendly application for file upload and retrieval. By utilizing cloud storage and a modern web development stack, the project provides a convenient solution for users to upload, store, and download files, catering to the evolving needs of modern data management.

OBJECTIVE

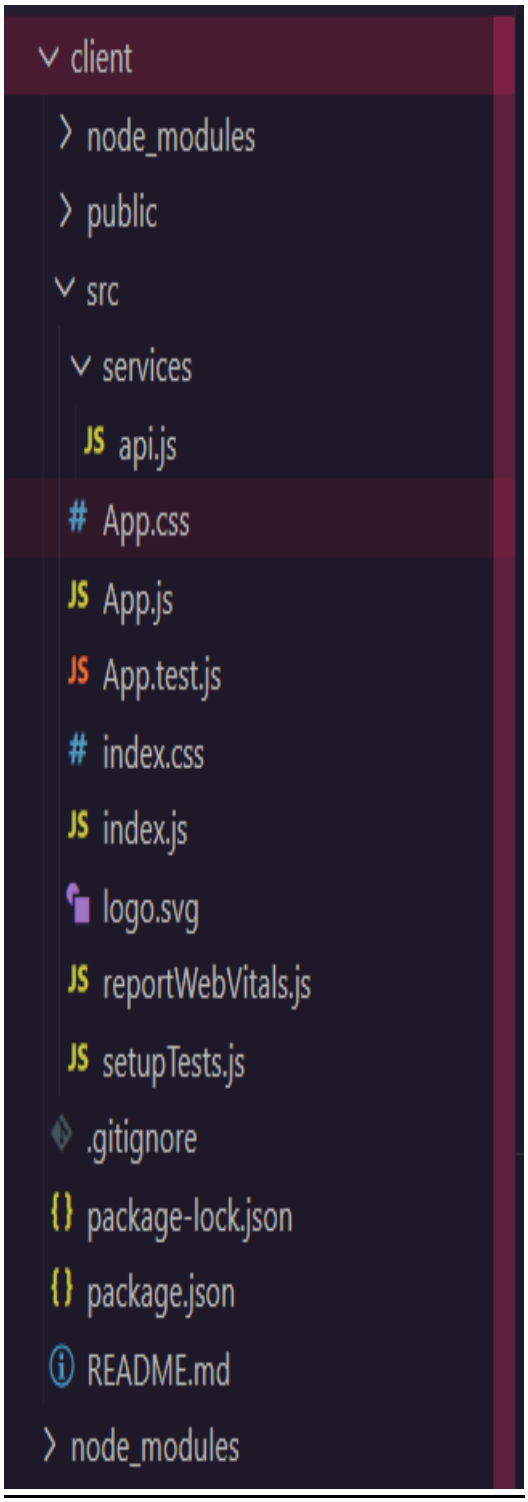
- Develop a web application using the MERN stack to enable users to upload/download files.
- Implement a backend system using Node.js and Express.js
- Handle file upload requests and save the uploaded files in a cloud-based MongoDB database.
- Create a user-friendly interface using React.js for seamless file upload functionality.
- Generate a unique download link for each uploaded file, allowing users to easily access and download their files.
- Implement efficient file retrieval mechanisms to enable users to quickly retrieve their recently uploaded files.
- Optimize the performance of the application to handle file uploads of various sizes and formats.
- Implement error handling and validation for integrity and security of the uploaded files.
- Conduct testing to ensure the stability, reliability, and usability of the application.

MODULARIZATION DETAILS

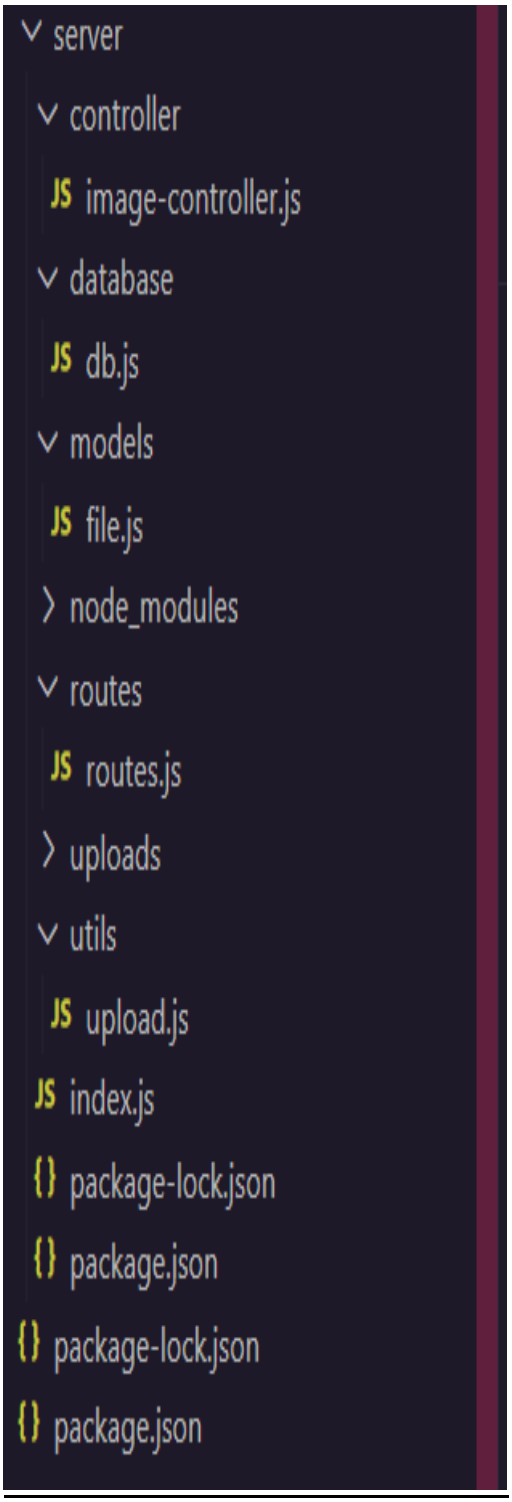
1. **CLIENT :** The **CLIENT** file plays a crucial role in the overall architecture. The Client file refers to the frontend part of the application, developed using React.js. In the Client file, developers can define various components such as file upload forms, download buttons, progress bars, and other UI elements. They can also incorporate external libraries and frameworks to enhance the functionality and aesthetics of the application.

2. **SERVER :** the **SERVER** file plays a crucial role in the overall architecture. The Server file refers to the backend part of the application, developed using Node.js and Express.js. Inside the Server file, developers define routes and endpoints using Express.js to handle incoming requests from the client-side. These routes specify the necessary operations, such as saving uploaded files to a cloud-based MongoDB database and generating unique download links.

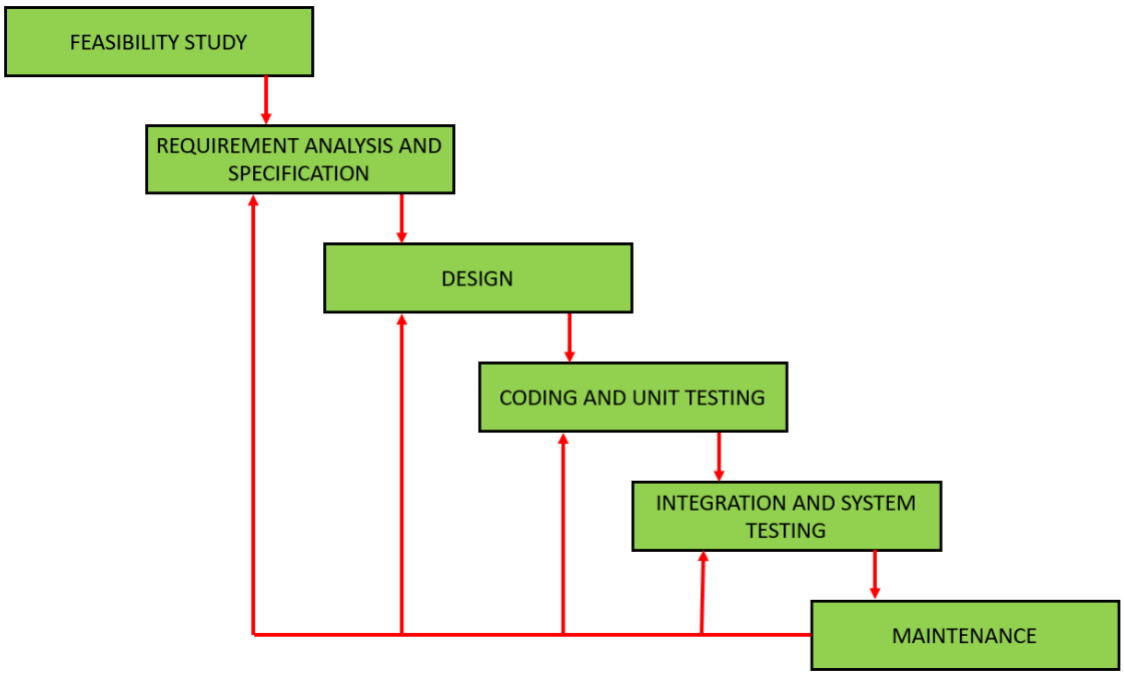
CLIENT



SERVER



SOFTWARE ENGINEERING PARADIGM APPLIED



SYSTEM REQUIREMENTS

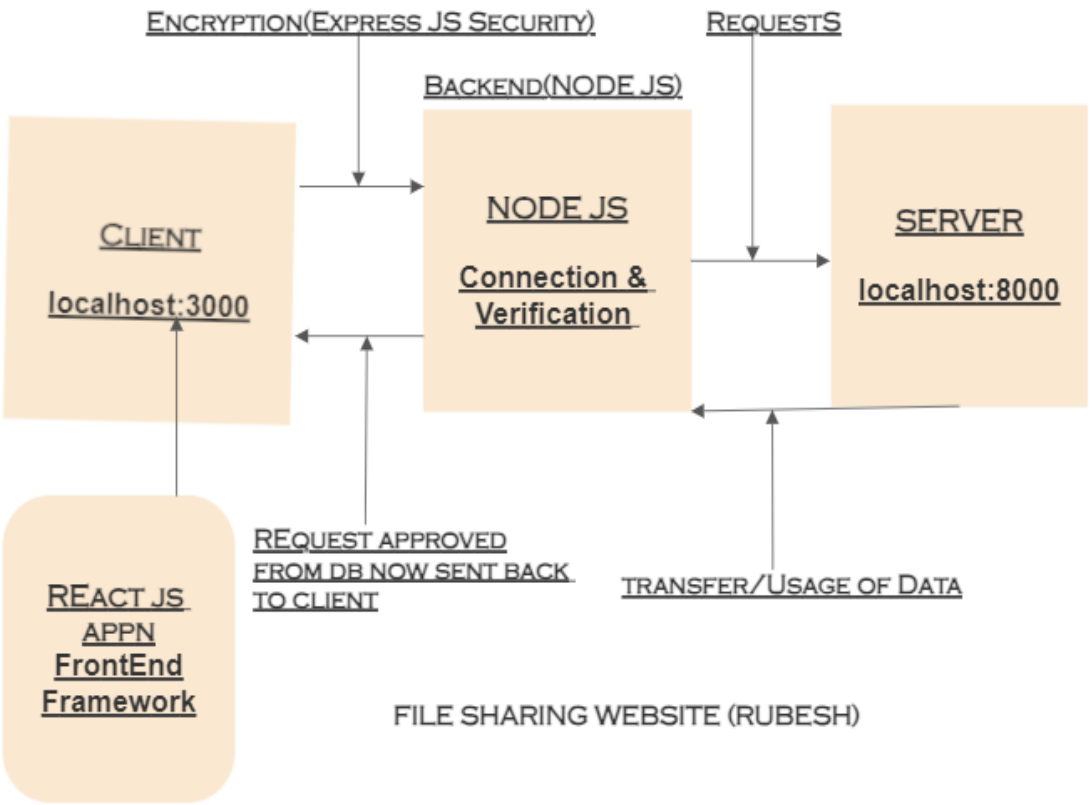
HARDWARE

- Processor: Intel Core i3 or equivalent.
- RAM: 4/8GB or higher.
- Display: Minimum resolution of 1280x720 pixels.

SOFTWARE

- MongoDB
- Express.JS
- ReactJS
- NodeJS
- VS Code

FLOWCHART



FILE SHARING WEBSITE (RUBESH)

SCREENSHOTS

App.js

client > src > JS App.js > App

```jsx
import { useRef, useState, useEffect } from 'react';
import logo from './logo.svg';

import './App.css';
import { uploadFile } from './services/api';



function App() {
  const [file, setFile]= useState('');
  const [result , setResult]= useState('');
const fileInputRef =useRef();

  const logo='https://images.unsplash.com/photo-1504711331083-9c895941bf81?ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxzZWFyY2h8OXx8ZmlsZXxlbnwwfH

  useEffect(()=>{
    const getImage=async()=>{
      if(file){
        const data= new FormData();
        data.append("name",file.name);
        data.append("file", file);

      let response= await uploadFile(data);
      setResult(response.path);
      }
    }
  getImage();
  }, [file])
  const onUploadClick = ()=>{
    fileInputRef.current.click();
  }
  console.log(file);
  return (
   <div className='container'>
    <img src={logo} alt="banner" />
    <div className='wrapper'>
      <h1 >File Sharing :) </h1>
      <p>Upload your file and get a Link to share :D </p>

      <button onClick={()=>onUploadClick()}>Upload File</button>
      <input type="file"
      ref={fileInputRef}
      style={{display: 'none'}}
      onChange={(e)=>setFile(e.target.files[0])}
      />
      <a href={result} target="_blank">{result}</a>
      </div>
   </div>

  );
}

export default App;
```

index.js

```
server > JS index.js > ...
import express from 'express';
import router from './routes/routes.js';
import cors from 'cors';
import DBConnection from './database/db.js';

const app=express();


app.use(cors());

app.use('/',router);

const PORT = 8000;
DBConnection();

app.listen(PORT,()=>console.log(`Rubesh your Server is Working Fine on PORT ${PORT}`));
```

db.js

```
server > database > JS db.js > ...
    try{
        await mongoose.connect(MONGODB_URL, {useNewUrlParser: true});
        console.log('Database Connected Successfully ... All THANKS to Rubesh ')

    } catch (error) {
        console.error('Error while connecting DataBase.... Ask Admin Rubesh to Resolve', error.message);
    }
}

export default DBConnection;

// https://cloud.mongodb.com/v2/64b050423d6d4110e8539ac1#/metrics/replicaSet/64b056508d93fa503fd42c53/explorer/test/files/find
```

routes.js

```
server > routes > JS routes.js > ...
import express from 'express';

import { uploadImage, downloadImage } from '../controller/image-controller.js';
import upload from '../utils/upload.js';
const router = express.Router();


router.post('/upload', upload.single('file') ,uploadImage);
router.get('/file/:fileId', downloadImage);

export default router;
```

image-controller.js

server > controller > JS image-controller.js > downloadImage

```
21
22  export const downloadImage = async(request,response)=>{
23      try{
24        const file =  await File.findById(request.params.fileId);
25        file.downloadContent++;
26
27        await file.save();
28
29        response.download(file.path,file.name);
30
31      }catch(error){
32              console.error(error.message);
33              return response.status(500).json({error: error.message})
34      }
35  }
36
```

file.js

server > models > JS file.js > fileSchema

```
1   import mongoose from "mongoose";
2
3   const fileSchema = new mongoose.Schema({
4       path: {
5           type: String,
6           required: true
7       },
8       name: {
9           type: String,
10          required: true
11      },
12      downloadCount: {
13          type: Number,
14          required: true,
15          default: 0
16      },
17  })
18
19  const File = mongoose.model('file', fileSchema);
20
21  export default File;
```

axios.js

client > src > services > JS api.js > ...

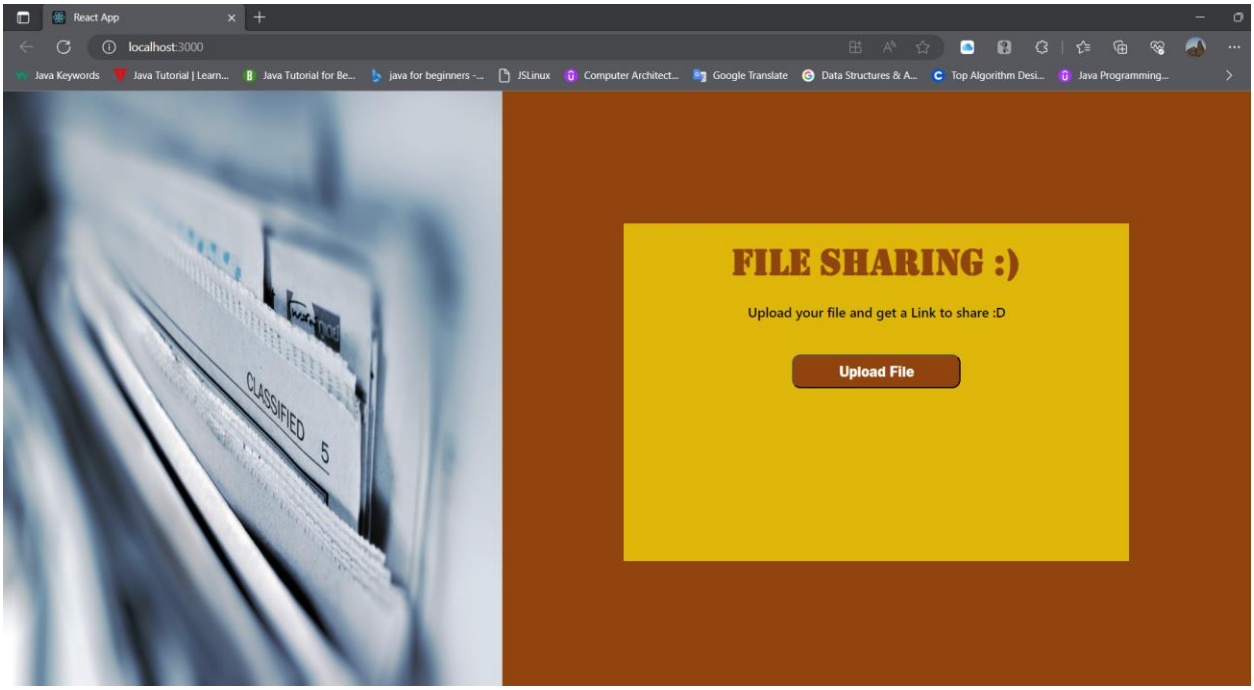
```
1   import axios from 'axios';
2
3   const API_URI = 'http://localhost:8000';
4
5   export const uploadFile = async (data) => {
6       try {
7           const response = await axios.post(`${API_URI}/upload`, data);
8           return response.data;
9       } catch (error) {
10          console.log('Error while calling the API ', error.message);
11      }
12  }
13
```

npm start (server)

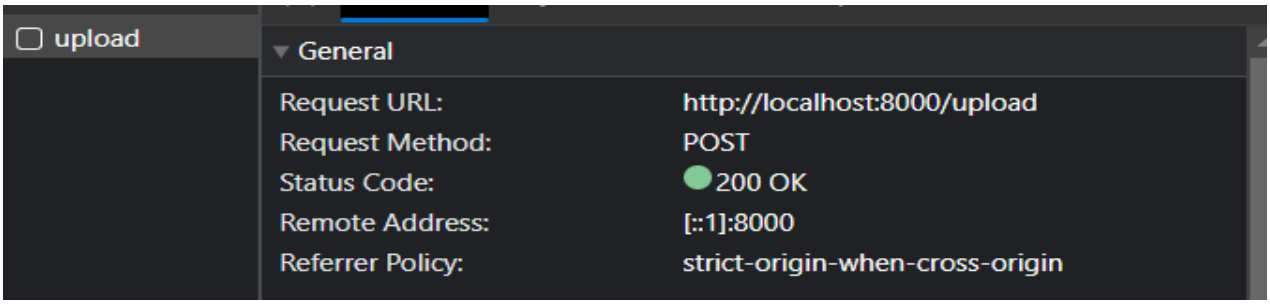
```
PS C:\C C++\Summer Training 2023\Project 3 file sharing\server> npm start

> server@1.0.0 start
> nodemon index.js

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Rubesh your Server is Working Fine on PORT 8000
Database Connected Successfully ... All THANKS to Rubesh
{
  path: 'uploads\\380cc8e675a08f739c42d57706567d10',
  name: 'bb.png',
  downloadCount: 0,
  _id: new ObjectId("64b1162c8f95fad9350bafe0"),
```

npm start(client)

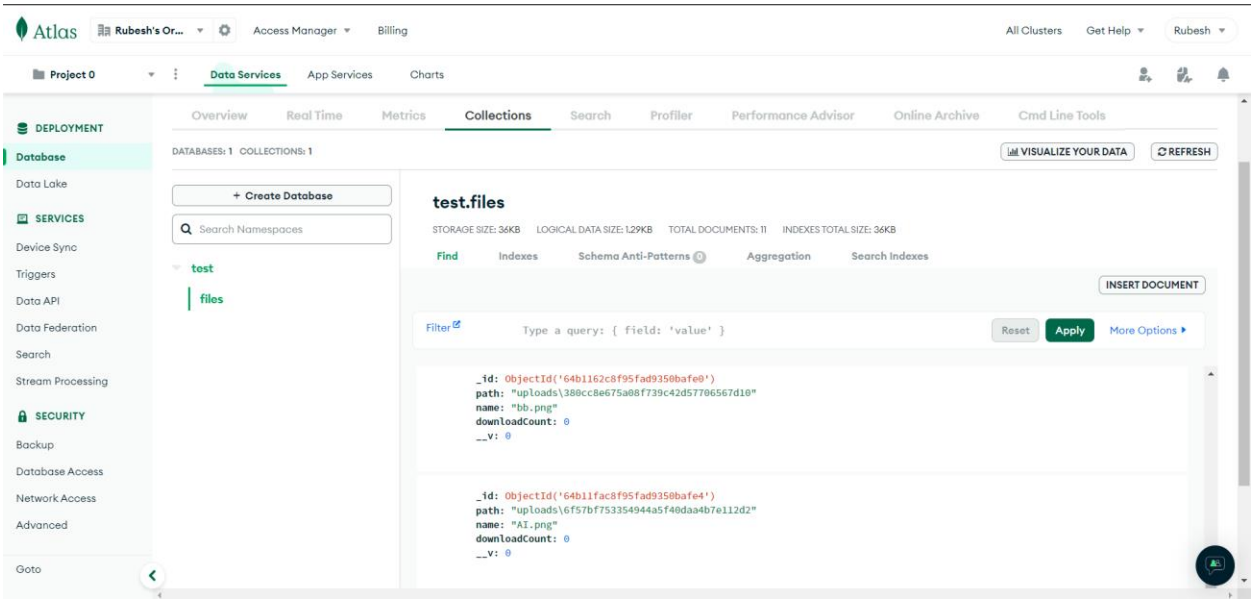
localhost:3000



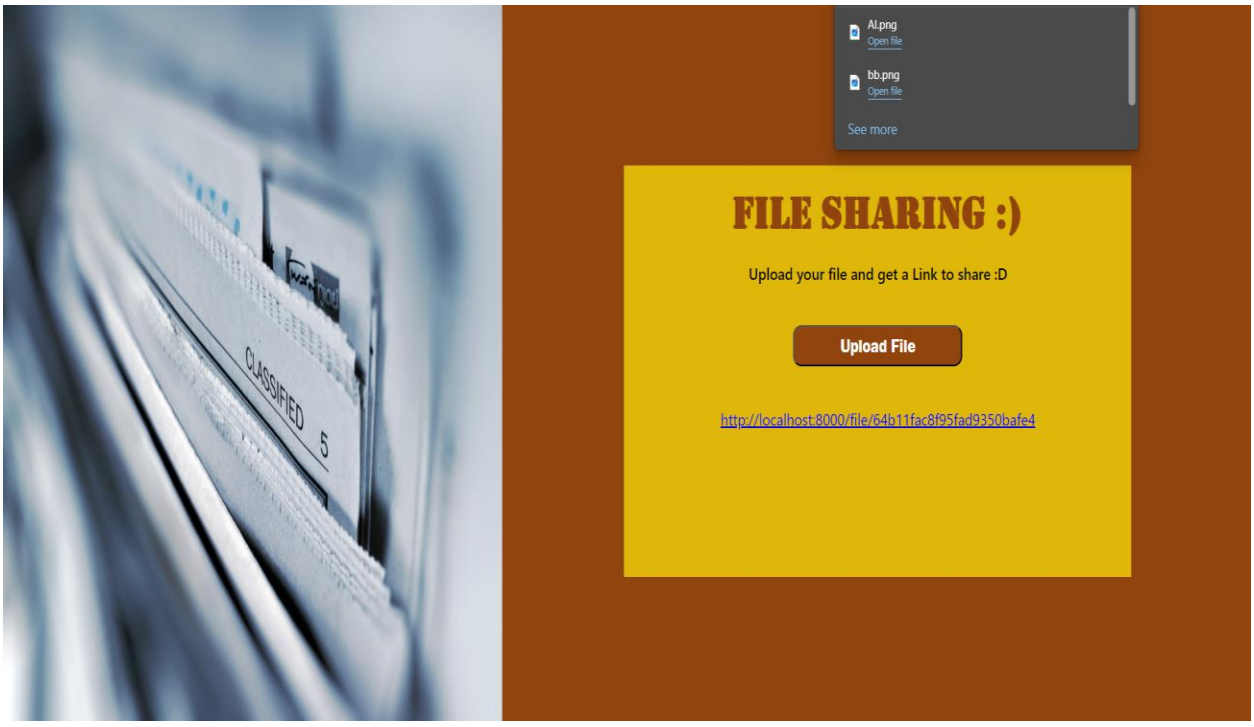
Frontend to Backend (Upload)



File saved in cloud.mongodb.com



Unique link for Downloading the recently uploaded file



FUTURE SCOPE FOR IMPROVEMENT

- **Enhanced File Management**: Implement advanced file management features such as file categorization, search functionality, and folder organization to provide users with better control and accessibility over their uploaded files.

- **User Authentication and Access Control**: Incorporate user authentication mechanisms to ensure secure access and file management. Implement user roles and permissions to control who can upload, download, or modify files within the application.

- **File Versioning and Revision History**: Introduce file versioning capabilities, allowing users to upload and manage multiple versions of a file. Maintain a revision history to track changes, enabling users to revert to previous versions if needed.

- **Integration with Cloud Storage Providers**: Offer integration options with popular cloud storage providers (e.g., Google Drive, Dropbox) to allow users to directly upload files from their existing accounts. This would provide users with flexibility and convenience in managing their files across platforms.

CODE

Source Code: https://github.com/Apex-Overlord-5/SummerTrainingRubesh