An

Minor Project Report

On

Online File Storing Website

Submitted to the Department of Computer Science & Engineering, Shri Jagdishprasad Jhabarmal Tibrewala University

Partial fulfillment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

Supervisor
Arti Panwar
Designation
Assistant Professor

Submitted By:
Praveen Sharma
Roll No.: 22BC8006

Department of Computer Science & Engineering

Shri Jagdishprasad Jhabarmal Tibrewala University, Jhunjhunu (Rajasthan) - 333010

March 2025

Candidate's Declaration

I hereby declare that the work, which is being presented in the BCA-403, Industrial Project, entitled **"Online File Storing Website"** in partial fulfilment for the award of Degree of "**Bachelor of Computer Application**" in Department of Computer Science & Engineering submitted to the Shri Jagdishprasad Jhabarmal Tibrewala University is a record of my own work carried under the Guidance of Mrs. Arti Panwar Department of Computer Science & Engineering, Shri Jagdishprasad Jhabarmal Tibrewala University.

I have not submitted the matter presented in this Project Report any where for the award of any other Degree.

Praveen Sharma
Roll No.: 22BC8006

Mrs. Arti Panwar

Shri Jagdishprasad Jhabarmal Tibrewala University
Department of Computer Science & Engineering

Certificate

Date: 25 Nov 2023

This is to certify that the Industrial Project (BCA-403) work entitled "*Online File Storing Website* '' submitted by "*Praveen Sharma***"** (SJJTU Roll No. 22BC2006) to the Department of Computer Science and Engineering, Shri Jagdishprasad Jhabarmal Tibrewala University has been examined and evaluated.

The Project work has been prepared as per the regulations of Shri Jagdishprasad Jhabarmal Tibrewala University, Jhunjhunu and qualifies to be accepted in partial fulfillment of the requirement for the degree of BCA (Bachelor of Computer Applications).

Signature of the student

Supervisor/Guide
Mrs. Arti Panwar

Head of Department/Institute

INDEX

1. Introduction

1.1 Topic of the project

Our project is a simple and useful platform where users can create their own accounts and upload files. After uploading, these files can be seen by other users on the platform. If someone wants to download a file, they can easily do so. This helps users share information and stay connected. The main aim of this project is to make file sharing easy and to allow users to help each other by sharing useful documents and resources.

1.2 Problem Description

- To provide a secure platform for users to store their files digitally.
- To allow users to create accounts and manage their uploaded files.
- To enable other users to view and download shared files.
- To solve the problem of file loss and limited offline storage.
- To promote easy sharing and access of documents anytime, from anywhere.

1.3 Conclusion

In conclusion, the file storing platform provides a secure and efficient way for users to upload, manage, and share files, ensuring easy access and collaboration within a connected system.

2. System Study

2.1 System with limitations

The current system is not fully automated, and it relies on manual processes for uploading, storing, and managing files. This leads to inefficiencies, potential data loss, and difficulties in sharing files securely between users.

2.2 Significance of the Project

This platform allows users to securely store, manage, and share important files, fostering collaboration and knowledge sharing. It helps students and other users stay connected and support each other through easy access to shared resources.

- The main objective of the platform is to help users store, manage, and share important files securely, improving collaboration and knowledge sharing within their academic and professional networks.
- Connecting with peers through file sharing can open doors to new resources, while uploading and downloading files creates opportunities for learning and supporting each other.
- The platform promotes active engagement through file sharing and collaborative activities, enhancing users' ability to access and contribute to shared resources.

2.3 Beneficiaries of the System

- **Faculty**: Can securely upload and share educational materials, research, and assignments with students.
- **Current Students**: Can access and download study resources, collaborate on projects, and share important documents.
- **University**: Gains an organized system for storing and distributing academic materials, improving accessibility and efficiency.
- **Users**: Benefit from a centralized platform to store, manage, and share files with peers, fostering collaboration and easy access to resources.

2.4 Feasibility study

During this study, it was found that the organization has enough resources to implement the new system. There already exists infrastructure to execute the software (Hardware / Software / Server) in the concerned organization.

Technical feasibility includes 2 main aspects:

- Hardware feasibility
- Software feasibility

Hardware feasibility

To implement this project we need hardware configuration for server and client.

- System : intel i5 Processor.
- Hard Disk : 1 TB
- Display : Monitor , Mobile Phone
- Ram : 8 GB.

Software feasibility

This system is developed using Spring Boot, React, Tailwind CSS. All resources used for the development of the project are available. The system can be expended as required in future and modified with the change of acts and rules.

- Operating System : Windows 7 and higher, Mac , Linux .
- Coding Language : Spring Boot, React, Tailwind CSS.
- Tool : Visual Studio 2024
- Database : MySQL

3. System Analysis Requirement Specification

i. Functional Requirement.

1	**Login**
1.1	User can register with first name, last name, email and password.
1.2	User can login with email and password.
1.3	User can reset the password once they forgot.
2	**Upload and Manage File by User**
2.2	User can upload file with (Title, Description).
2.3	User can edit uploaded file (Title, Description).
2.4	User can delete uploaded file
3	**Manage personal info**
3.1	User can update their information.
3.2	User can view their information.
4	**User View, and Download Files**
4.1	User can view the all uploaded files of other users
4.2	User can Download the file of other users

ii. Non-Functional Requirement Description

1	**Performance**
1.1	The system must response to user request or user input not more than 2 seconds.
2	**Reliability issues**
2.1	For a single user, the system should crash no more than once per 10 hours.
2.2	If the systems crash, it should behave perfectly normal when reloaded again.
3	**Usability issues**
3.1	System should be able to process within 10 second.
4	**Security**
4.1	All the user must have own account to use this system.
4.2	The system must ensure every user have unique ID.
4.3	The system only allows member login once the email and the password are match.

iii. User Requirement

- Users should be able to register and log in securely.
- Users must be able to upload files with titles and descriptions.
- Users should be able to view and download files uploaded by others.
- The system should allow easy sharing and access to educational resources.

- A user-friendly dashboard should display all uploaded and accessible files.
- The platform should ensure data security and file access control.

iv. System Requirement

HARDWARE REQUIREMENTS:

- System : intel i5 Processor.
- Hard Disk : 1 TB
- Display : Monitor , Mobile Phone
- Ram : 8 GB.

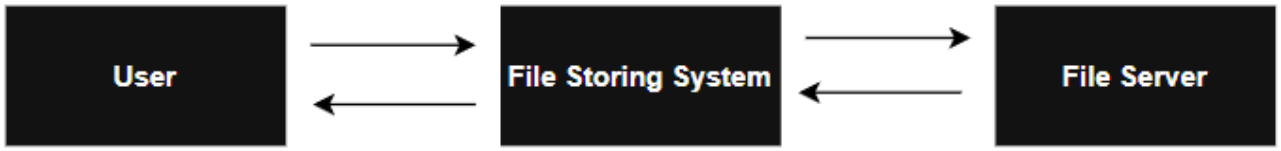
SOFTWARE REQUIREMENTS:

- Operating System : Windows 7 and higher, Mac , Linux .
- Coding Language : Spring Boot, React, Tailwind CSS.
- Tool : Visual Studio 2024
- Database : MySQL

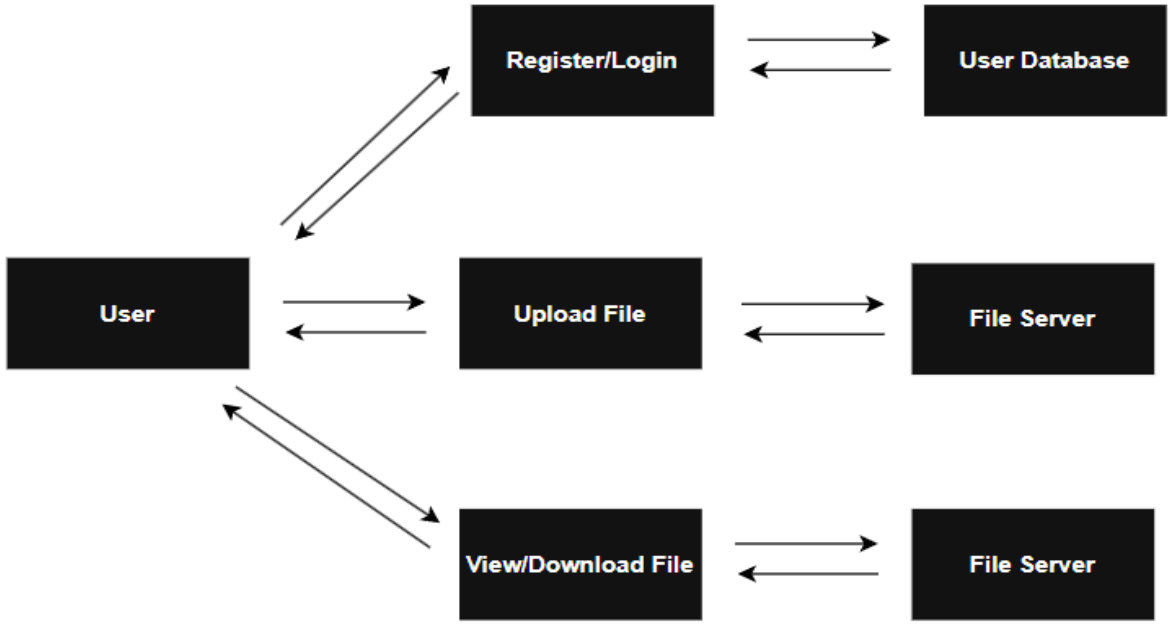
4. System Design

a) Data Flow Diagram

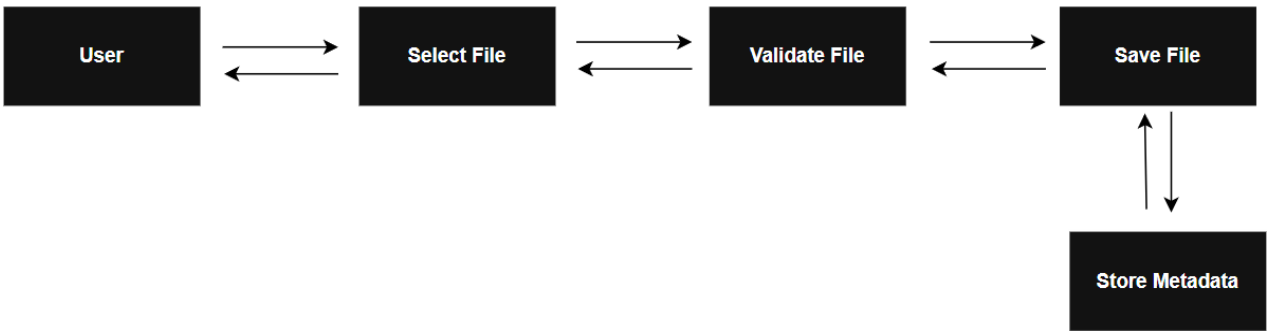
0 level Diagram



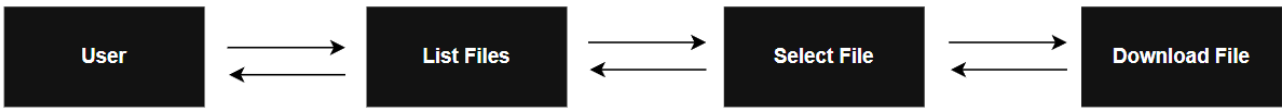
Level 1 Diagram



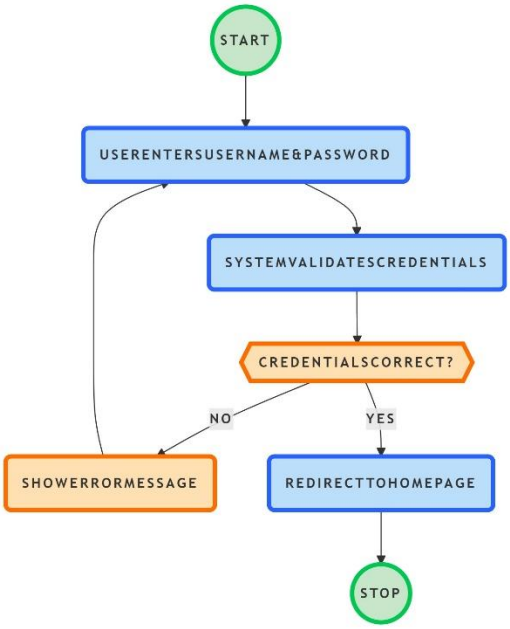
Level 2 Diagram



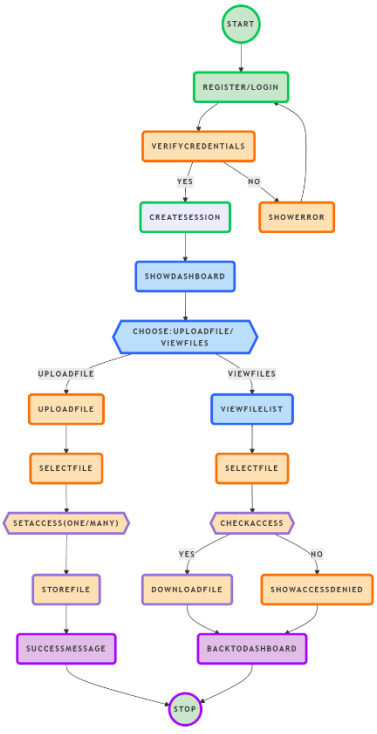
b) E-R Diagrams



c) Use Case Diagrams



e) Database Tables



e) Database Tables

a) User Table

Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	id	testdb	users	BIGINT	binary	20	1	0
2	email	testdb	users	VARCHAR	utf8mb4	255	28	0
3	first_name	testdb	users	VARCHAR	utf8mb4	255	7	0
4	gender	testdb	users	VARCHAR	utf8mb4	255	4	0
5	last_name	testdb	users	VARCHAR	utf8mb4	255	6	0
6	password	testdb	users	VARCHAR	utf8mb4	255	10	0

b) File Table

Field Types

#	Field	Schema	Table	Type	Character Set	Display Size	Precision	Scale
1	id	testdb	files	BIGINT	binary	20	2	0
2	created_date	testdb	files	DATETIME	binary	26	20	6
3	description	testdb	files	TEXT	utf8mb4	1073741823	116967	0
4	file_path	testdb	files	VARCHAR	utf8mb4	255	50	0
5	title	testdb	files	VARCHAR	utf8mb4	255	149	0
6	user_id	testdb	files	BIGINT	binary	20	1	0
7	file_url	testdb	files	VARCHAR	utf8mb4	255	78	0

f) Input output Forms

INPUT

- Registration
- Signup
- Login
- Profile Update
- Edit Events

OUTPUT

- Profile
- Events
- About the Website
- Contact Us

5. Development

a) Environment

HARDWARE REQUIREMENTS:

- Processer : The Software need I5 Processor to run or execute in the system, as it will provide the maximum utilization of OS and improve the speed or resource utilization.

- Hard Disk : The Software need 1 TB storage to store the data in the system, as it will provide the it will make data access and retrieve easy and faster.
- Display : mobile phone or computer screen.
- RAM : We need a minimum RAM “4 GB” for the seamless process of website functioning.

SOFTWARE REQUIREMENTS:

- Operating System : Windows 7 and higher, Mac , Linux , Andriod.
- Coding Language

  HTML , CSS : frontend and designing.

  JavaScript : To make web pages interactive.
- Tool : Visual Studio Code
- Database : Google Sheet

b) Coding Home page

Frontend Code

Home Page

```
import React, { useState, useEffect } from "react";
import { Helmet } from "react-helmet";
import { Link } from "react-router-dom";
import Cookies from "js-cookie"; // Import the js-cookie library

const Home = () => {
  const [files, setFiles] = useState([]);
  const [users, setUsers] = useState([]);
  const [loadingFiles, setLoadingFiles] = useState(true);
  const [loadingUsers, setLoadingUsers] = useState(true);
  const [userId, setUserId] = useState(null); // State for storing userId from cookies

  // Fetch files and users from API
  useEffect(() => {
    // Get userId from cookies
    const storedUserId = Cookies.get("userId"); // Assuming 'userId' is stored in a cookie
    setUserId(storedUserId);
    console.log("User ID from cookies: ", storedUserId); // For debugging purposes

    // Fetch files (latest 10)
    fetch("http://localhost:8080/files/all")
      .then((response) => response.json())
      .then((data) => {
        // Filter last 10 files (most recent)
        setFiles(data.reverse().slice(0, 5)); // Reverse and slice for the latest 10
        setLoadingFiles(false);
      })
      .catch((error) => {
```

```
      console.error("Error fetching files:", error);
      setLoadingFiles(false);
    });

  // Fetch users (latest 10)
  fetch("http://localhost:8080/users")
    .then((response) => response.json())
    .then((data) => {
      // Filter last 10 users (most recent)
      setUsers(data.reverse().slice(0, 5)); // Reverse and slice for the latest 10
      setLoadingUsers(false);
    })
    .catch((error) => {
      console.error("Error fetching users:", error);
      setLoadingUsers(false);
    });
 }, []); // Empty dependency array means this runs once on mount

 return (
  <div className="min-h-screen p-8">
    <Helmet>
      <title>Home - FileHub</title>
    </Helmet>

    <div className="max-w-7xl mx-auto text-center text-gray-600">
      <h1 className="text-4xl font-extrabold mb-4">Welcome to FileHub</h1>
      <p className="text-xl mb-8">
        Check out the latest uploads and new users on our platform.
      </p>

      <div className="w-full md:max-w-6xl mx-auto">
        {/* Last 10 Users */}
        <div className="mb-10">
          <h2 className="text-3xl font-semibold mb-6">New Users</h2>
          {loadingUsers ? (
            <p className="text-gray-300">Loading users...</p>
          ) : (
            <div className="grid grid-cols-2 md:grid-cols-3 lg:grid-cols-5 gap-6">
              {users.map((user, index) => (
                <div
                  key={index}
                  className="bg-white p-6 rounded-lg shadow-lg hover:shadow-xl transform hover:scale-105
transition duration-300 ease-in-out"
                >
                  <div className="flex flex-col items-center">
                    <Link to={`/users/${user.id}`}>
                      <div className="text-base text-nowrap font-semibold">
                        {user.firstName} {user.lastName}
                      </div>
                    </Link>
                  </div>
                </div>
```

```
    ))}
   </div>
  )}
</div>

{/* Last 10 Uploaded Files */}
<div className="mb-10">
  <h2 className="text-3xl font-semibold mb-6">Latest Uploaded Files</h2>
  {loadingFiles ? (
    <p className="text-gray-300">Loading files...</p>
  ) : (
    <div className="overflow-x-auto shadow-md rounded-lg">
      <table className="min-w-full text-sm text-left text-gray-500">
        <thead className="text-xs text-gray-700 uppercase bg-gray-50 border-b border-gray-300">
          <tr>
            <th className="px-6 py-3 whitespace-nowrap">S. No.</th>
            <th className="px-6 py-3">File Title</th>
            <th className="px-6 py-3 whitespace-nowrap">
              File Owner
            </th>
            <th className="px-6 py-3 whitespace-nowrap">
              Created Date
            </th>
            <th className="px-6 py-3 text-right">Action</th>
          </tr>
        </thead>
        <tbody>
          {files.map((file, index) => (
            <tr
              key={file.id}
              className="bg-white border-b border-gray-200 hover:bg-gray-50"
            >
              <td className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap">
                {index + 1}
              </td>
              <td className="px-6 py-4">{file.title}</td>
              <td className="px-6 py-4 whitespace-nowrap">
                {file.user.firstName} {file.user.lastName}
              </td>
              <td className="px-6 py-4 whitespace-nowrap">
                {file.createdDate
                  ? new Date(file.createdDate).toLocaleDateString(
                      "en-GB"
                    )
                  : "N/A"}
              </td>
              <td className="px-6 py-4 text-right">
                <Link
                  to={`/file/${file.id}/${file.title}`}
                  className="font-medium text-blue-600 hover:underline"
                >
                  View
```

```
                    </Link>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>
      )}
    </div>
  </div>
 </div>
);
};

export default Home;
```

Upload File

```
import React, {
  useState,
  useRef,
  useEffect,
  useCallback,
  useMemo,
} from "react";
import axios from "axios";
import JoditEditor from "jodit-react";
import Cookies from "js-cookie";
import { useDropzone } from "react-dropzone";
import { useNavigate } from "react-router-dom"; // Import useNavigate for navigation

export default function UploadFile() {
  const [file, setFile] = useState(null);
  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [showModal, setShowModal] = useState(false); // State to manage modal visibility
  const editor = useRef(null);

  // Retrieve userId from cookies
  const userId = Cookies.get("userId");
  const navigate = useNavigate(); // Used for navigation

  useEffect(() => {
    if (!userId) {
      alert("User is not logged in. Please login first.");
      window.location.href = "/login";
    }
  }, [userId]);

  // Handle file selection via drag and drop
  const onDrop = useCallback((acceptedFiles) => {
```

```
  setFile(acceptedFiles[0]); // Set the first file from dropzone
}, []);

const { getRootProps, getInputProps, isDragActive } = useDropzone({
  onDrop,
  accept: "*",
  multiple: false,
});

const handleUpload = async (e) => {
  e.preventDefault(); // Prevent page reload on button click

  // Validate if file, title, and description are filled
  if (!file || !title || !description) {
    alert("Please fill in all fields.");
    return;
  }

  try {
    const formData = new FormData();
    formData.append("file", file);
    formData.append("title", title);
    formData.append("description", description);

    const response = await axios.post(
      `http://localhost:8080/files/upload/${userId}`,
      formData,
      { headers: { "Content-Type": "multipart/form-data" } }
    );

    setShowModal(true); // Show the success modal when upload is successful
    console.log(response.data);

    // Navigate to the 'all-files' page after upload is successful
    navigate("/all-files"); // Redirect to the all-files page
  } catch (error) {
    console.error("Upload error:", error);
    alert("File upload failed!");
  }
};

// JoditEditor configuration to add images from local storage
const editorConfig = useMemo(
  () => ({
    readonly: false,
    toolbar: true,
    uploader: {
      insertImageAsBase64URI: true,
      url: (file) => {
        // Logic for uploading image locally to Jodit
        const reader = new FileReader();
        reader.onload = (e) => {
```

```
          const imgDataUrl = e.target.result;
          editor.current.selection.insertImage(imgDataUrl); // Insert image into the editor
        };
        reader.readAsDataURL(file);
      },
    },
  }),
  []
); // Memoize the configuration to avoid unnecessary updates

 // Handle closing the modal
 const closeModal = () => {
  setShowModal(false); // Close the modal
 };

 return (
  <div className="flex flex-col items-center justify-center min-h-screen bg-gray-100">
   <div className="bg-white shadow-2xl rounded-lg p-8 w-full max-w-4xl">
    <h2 className="text-3xl font-semibold mb-6 text-center text-gray-800">
     Upload File
    </h2>

    {/* Drag and Drop File Upload */}
    <div
     {...getRootProps()}
     className="border-2 border-dashed border-gray-300 p-8 mb-6 text-center cursor-pointer rounded-lg
hover:border-blue-500 transition-all duration-300"
    >
     <input {...getInputProps()} />
     {isDragActive ? (
      <p className="text-blue-500 font-medium">Drop the file here...</p>
     ) : (
      <p className="text-gray-600 font-medium">
       Drag & drop a file here, or click to select one
      </p>
     )}
    </div>


    {file && (
     <p className="text-green-600 text-center mb-4 font-medium">
      Selected file: {file.name}
     </p>
    )}

<button
     className="w-full mb-5 bg-blue-600 text-white py-3 rounded-lg font-medium hover:bg-blue-700
transition duration-300"
     onClick={handleUpload} // Prevent form submission
    >
     Upload File
    </button>
```

```
      <label className="block text-gray-800 font-medium mb-2">Title</label>
      <input
        type="text"
        className="w-full px-4 py-3 border border-gray-300 rounded-lg focus:ring-2 focus:ring-blue-500
mb-4"
        placeholder="Enter file title"
        value={title}
        onChange={(e) => setTitle(e.target.value)}
      />

      <label className="block text-gray-800 font-medium mb-2">
        Description
      </label>
      <JoditEditor
        ref={editor}
        value={description}
        onChange={(newContent) => setDescription(newContent)}
        config={editorConfig} // Apply configuration for image upload
        className="w-full mb-6 border-2 border-gray-300 rounded-lg"
        style={{ height: "600px", overflowY: "auto" }} // Fixed height and scroll
      />



      {/* Dialog Box for Success */}
      {showModal && (
        <div
          data-dialog-backdrop="dialog"
          data-dialog-backdrop-close="true"
          className="pointer-events-none fixed inset-0 z-[999] grid h-screen w-screen place-items-center bg-
black bg-opacity-60 opacity-100 backdrop-blur-sm transition-opacity duration-300"
        >
          <div
            data-dialog="dialog"
            className="relative m-4 p-4 w-2/5 min-w-[40%] max-w-[40%] rounded-lg bg-white shadow-sm"
          >
            <div className="flex shrink-0 items-center pb-4 text-xl font-medium text-slate-800">
              File uploaded successfully!
            </div>
            <div className="relative border-t border-slate-200 py-4 leading-normal text-slate-600 font-light">
              You can now view or manage your uploaded file.
            </div>
            <div className="flex shrink-0 flex-wrap items-center pt-4 justify-end">
              <button
                data-dialog-close="true"
                className="rounded-md border border-transparent py-2 px-4 text-center text-sm transition-all
text-slate-600 hover:bg-slate-100 focus:bg-slate-100 active:bg-slate-100 disabled:pointer-events-none
disabled:opacity-50 disabled:shadow-none"
                type="button"
                onClick={closeModal} // Close the modal
              >
```

```
              Done
            </button>
            <button
              data-dialog-close="true"
              className="rounded-md bg-green-600 py-2 px-4 border border-transparent text-center text-sm text-white transition-all shadow-md hover:shadow-lg focus:bg-green-700 focus:shadow-none active:bg-green-700 hover:bg-green-700 active:shadow-none disabled:pointer-events-none disabled:opacity-50 disabled:shadow-none ml-2"
              type="button"
              onClick={() => navigate("/all-files")} // Redirect after upload
            >
              View File
            </button>
          </div>
        </div>
      </div>
    )}
  </div>
 </div>
);
}
```

Login Page

```
import React, { useState } from "react";
import { Eye, EyeOff } from "lucide-react";
import { Link, useNavigate } from "react-router-dom";
import Cookies from "js-cookie";

export default function Login() {
 const [password, setPassword] = useState("");
 const [showPassword, setShowPassword] = useState(false);
 const [email, setEmail] = useState("");
 const [error, setError] = useState("");

 const navigate = useNavigate();

 const handleLogin = async (e) => {
  e.preventDefault(); // Prevent form submit default behavior

  setError("");
  try {
   const response = await fetch(
    `http://localhost:8080/users/login?email=${email}&password=${password}`,
    {
     method: "POST",
     headers: { "Content-Type": "application/json" },
     body: JSON.stringify({ email, password }),
    }
   );

   if (response.status === 401) {
    throw new Error("Invalid email or password");
```

```
    }

    if (!response.ok) {
      throw new Error("Something went wrong. Please try again.");
    }

    const data = await response.json();

    if (!data.userId) {
      throw new Error("User ID is missing in response.");
    }

    // Save userId in cookies
    Cookies.set("userId", data.userId, { expires: 1 });

    alert("Login Successful");
    navigate("/"); // Navigate to home page after login
  } catch (error) {
    console.error(error.message);
    setError(error.message);
  }
};

 return (
  <div className="flex items-center justify-center h-screen">
   <div className="w-full max-w-md bg-white p-8 rounded-lg shadow-xl">
    <h2 className="text-3xl font-bold text-center text-gray-800 mb-6">
     Sign in
    </h2>

    {error && (
     <p className="text-red-500 text-sm text-center mb-4">{error}</p>
    )}

    <form id="loginForm" onSubmit={handleLogin}>
     <div className="mb-6">
      <label className="block text-sm font-medium text-gray-600 mb-2">
       Email
      </label>
      <input
       type="email"
       name="email"
       className="w-full px-4 py-3 border border-gray-300 rounded-md focus:ring-2 focus:ring-blue-500
focus:outline-none"
       placeholder="Enter your email"
       value={email}
       onChange={(e) => setEmail(e.target.value)}
       autoComplete="email"
       required
      />
     </div>
```

```
        <div className="mb-6 relative">
          <label className="block text-sm font-medium text-gray-600 mb-2">
            Password
          </label>
          <div className="relative">
            <input
              type={showPassword ? "text" : "password"}
              name="password"
              className="w-full px-4 py-3 border border-gray-300 rounded-md focus:ring-2 focus:ring-blue-
500 focus:outline-none"
              placeholder="Enter your password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              autoComplete="current-password"
              required
            />
            <button
              type="button"
              className="absolute right-3 top-3 text-gray-500"
              onClick={() => setShowPassword(!showPassword)}
            >
              {showPassword ? <EyeOff size={20} /> : <Eye size={20} />}
            </button>
          </div>
        </div>

        <button
          type="submit"
          className="w-full py-3 bg-blue-600 text-white font-semibold rounded-md hover:bg-blue-700
transition duration-300"
        >
          Sign In
        </button>
      </form>

      <div className="mt-4 text-center">
        <p className="text-sm text-gray-600">
          Don't have an account?{" "}
          <Link to="/register" className="text-blue-500">
            Sign Up
          </Link>
        </p>
      </div>
    </div>
  </div>
 );
}
```

Register Page

```
import React, { useState } from "react";
import { Eye, EyeOff } from "lucide-react";
import { useNavigate } from "react-router-dom";
```

```
import Cookies from "js-cookie";

export default function Register() {
  const [firstName, setFirstName] = useState("");
  const [lastName, setLastName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [showPassword, setShowPassword] = useState(false);
  const [gender, setGender] = useState("");
  const [error, setError] = useState("");
  const navigate = useNavigate();

  const handleRegister = async () => {
    setError(""); // Reset the error message before attempting registration

    // Check if any field is empty
    if (!firstName || !lastName || !email || !password || !gender) {
      setError("All fields are required.");
      return; // Prevent registration if any field is empty
    }

    try {
      const response = await fetch("http://localhost:8080/users/register", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ firstName, lastName, email, password, gender }),
      });

      if (!response.ok) {
        // Check if the response indicates a user already exists (e.g., 409 status code)
        if (response.status === 409) {
          throw new Error("User already exists. Please try logging in.");
        }
        throw new Error("Registration failed. Please try again.");
      }

      const data = await response.json();
      console.log("Registration successful", data);

      // Save userId in cookies
      Cookies.set("userId", data.userId, { expires: 7 });

      alert("Registration Successful");
      navigate("/"); // Redirect after successful registration
    } catch (error) {
      setError(error.message); // Display the error message
    }
  };

  return (
    <div className="flex items-center justify-center h-screen">
      <div className="w-full max-w-lg bg-white p-8 rounded-xl shadow-2xl">
```

```
      <h2 className="text-3xl font-semibold text-center text-gray-800 mb-6">
        Create an Account
      </h2>

      {error && (
        <p className="text-red-500 text-sm text-center mb-4">{error}</p>
      )}

      <form>
        {/* First Name */}
        <div className="mb-4">
          <label className="block text-sm font-medium text-gray-600 mb-2">
            First Name
          </label>
          <input
            type="text"
            className="w-full px-4 py-3 border border-gray-300 rounded-md focus:outline-none focus:ring-2
focus:ring-green-500"
            placeholder="Enter your first name"
            value={firstName}
            onChange={(e) => setFirstName(e.target.value)}
            required
          />
        </div>

        {/* Last Name */}
        <div className="mb-4">
          <label className="block text-sm font-medium text-gray-600 mb-2">
            Last Name
          </label>
          <input
            type="text"
            className="w-full px-4 py-3 border border-gray-300 rounded-md focus:outline-none focus:ring-2
focus:ring-green-500"
            placeholder="Enter your last name"
            value={lastName}
            onChange={(e) => setLastName(e.target.value)}
            required
          />
        </div>

        {/* Email */}
        <div className="mb-4">
          <label className="block text-sm font-medium text-gray-600 mb-2">
            Email
          </label>
          <input
            type="email"
            className="w-full px-4 py-3 border border-gray-300 rounded-md focus:outline-none focus:ring-2
focus:ring-green-500"
            placeholder="Enter your email"
            value={email}
```

```
            onChange={(e) => setEmail(e.target.value)}
            required
          />
        </div>

        {/* Password */}
        <div className="mb-4 relative">
          <label className="block text-sm font-medium text-gray-600 mb-2">
            Password
          </label>
          <div className="relative">
            <input
              type={showPassword ? "text" : "password"}
              className="w-full px-4 py-3 border border-gray-300 rounded-md focus:outline-none focus:ring-2
focus:ring-green-500"
              placeholder="Enter your password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              required
            />
            <button
              type="button"
              className="absolute right-3 top-3 text-gray-500"
              onClick={() => setShowPassword(!showPassword)}
            >
              {showPassword ? <EyeOff size={20} /> : <Eye size={20} />}
            </button>
          </div>
        </div>

        {/* Gender */}
        <div className="mb-6">
          <label className="block text-sm font-medium text-gray-600 mb-2">
            Gender
          </label>
          <div className="relative">
            <select
              className="w-full px-4 py-3 pr-10 border border-gray-300 rounded-md bg-white focus:outline-
none focus:ring-2 focus:ring-green-500 text-gray-700 appearance-none"
              value={gender}
              onChange={(e) => setGender(e.target.value)}
              required
            >
              <option value="">Select Gender</option>
              <option value="male">Male</option>
              <option value="female">Female</option>
              <option value="other">Other</option>
            </select>
            <svg
              className="absolute right-3 top-3 text-gray-500"
              xmlns="http://www.w3.org/2000/svg"
              fill="none"
```

```
            viewBox="0 0 24 24"
            stroke="currentColor"
            width="20"
            height="20"
           >
            <path
             strokeLinecap="round"
             strokeLinejoin="round"
             strokeWidth="2"
             d="M19 9l-7 7-7-7"
            />
           </svg>
          </div>
         </div>

         {/* Register Button */}
         <button
          type="button"
          className="w-full py-3 bg-green-600 text-white font-semibold rounded-md hover:bg-green-700
transition duration-300"
          onClick={handleRegister}
         >
          Register
         </button>
        </form>

        <div className="mt-4 text-center">
         <p className="text-sm text-gray-600">
          Already have an account?{" "}
          <a href="/login" className="text-blue-500 hover:underline">
           Sign In
          </a>
         </p>
        </div>
      </div>
    </div>
  );
}
```

All Users

```
import React, { useEffect, useState } from "react";
import { Link, useNavigate } from "react-router-dom";

const AllUsers = () => {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const navigate = useNavigate();

  // Get the logged-in user's ID from localStorage and convert it to a number
  const loggedInUserId = parseInt(localStorage.getItem("userId"), 10); // Ensure it's a number
```

```
useEffect(() => {
  fetchUsers();
}, []);

// Fetch users from the backend
const fetchUsers = async () => {
  try {
    const response = await fetch("http://localhost:8080/users");
    if (!response.ok) {
      throw new Error("Failed to fetch users");
    }
    const data = await response.json();

    // Exclude the logged-in user from the list
    const filteredUsers = data.filter((user) => user.id !== loggedInUserId);
    setUsers(filteredUsers);
  } catch (error) {
    setError(error.message);
  } finally {
    setLoading(false);
  }
};

return (
  <div className="min-h-screen w-full bg-gray-100 flex flex-col items-center p-6">
    <div className="relative overflow-x-auto shadow-md sm:rounded-lg w-full bg-white p-4">
      <h2 className="text-2xl font-bold text-gray-700 text-center mb-6">
        All Users
      </h2>

      {loading ? (
        <p className="text-center text-gray-500">Loading users...</p>
      ) : error ? (
        <p className="text-center text-red-500">{error}</p>
      ) : users.length > 0 ? (
        <table className="w-full text-sm text-left text-gray-500 border border-gray-200">
          <thead className="text-xs text-gray-700 uppercase bg-gray-50 border-b border-gray-300">
            <tr>
              <th className="px-6 py-3 whitespace-nowrap">S. No.</th>
              <th className="px-6 py-3 whitespace-nowrap">User Name</th>
              <th className="px-6 py-3 whitespace-nowrap">Email</th>
              <th className="px-6 py-3">Gender</th>
              <th className="px-6 py-3 whitespace-nowrap">Action</th>
            </tr>
          </thead>
          <tbody>
            {users.map((user, index) => (
              <tr
                key={user.id}
                className="bg-white border-b border-gray-200 hover:bg-gray-50 cursor-pointer"
                onClick={() => navigate(`/users/${user.id}`)}
              >
```

```
              <td className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap">
                {index + 1}
              </td>

              <td className="px-6 py-4 whitespace-nowrap">{user.firstName + " " + user.lastName}</td>
              <td className="px-6 py-4 whitespace-nowrap">{user.email}</td><td className="px-6 py-4
whitespace-nowrap">{user.gender}</td>
              <td className="px-6 py-4 whitespace-nowrap"> <Link
                to={`/users/${user.id}`}
                className="font-medium text-blue-600  hover:underline"
              >
                View
              </Link></td>
            </tr>
          ))}
        </tbody>
      </table>
    ) : (
      <p className="text-center text-gray-500">No users found.</p>
    )}
  </div>
 </div>
);
};

export default AllUsers;
```

All Files

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { Link } from "react-router-dom";

const FileList = () => {
 const [files, setFiles] = useState([]);
 const [loading, setLoading] = useState(true);

 useEffect(() => {
  axios
   .get("http://localhost:8080/files/all")
   .then((response) => {
    setFiles(response.data);
    setLoading(false);
   })
   .catch((error) => {
    console.error("Error fetching files:", error);
    setLoading(false);
   });
 }, []);

 return (
  <div className="min-h-screen w-full bg-gray-100 flex flex-col items-center p-6">
   <div className="relative overflow-x-auto shadow-md sm:rounded-lg w-full bg-white p-4">
```

```
<h2 className="text-2xl font-bold text-gray-700 text-center mb-6">
  Uploaded Files
</h2>

{loading ? (
  <p className="text-center text-gray-500">Loading files...</p>
) : files.length > 0 ? (
  <table className="w-full text-sm text-left text-gray-500 border border-gray-200">
    <thead className="text-xs text-gray-700 uppercase bg-gray-50 border-b border-gray-300">
      <tr>
        <th className="px-6 py-3 whitespace-nowrap">S. No.</th>
        <th className="px-6 py-3">File Title</th>
        <th className="px-6 py-3 whitespace-nowrap">File Owner</th>
        <th className="px-6 py-3 whitespace-nowrap">Created Date</th>
        <th className="px-6 py-3 text-right">Action</th>
      </tr>
    </thead>
    <tbody className="w-full">
      {files.map((file, index) => (
        <tr
          key={file.id}
          className="bg-white border-b border-gray-200 hover:bg-gray-50"
        >
          <td className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap">
            {index + 1}
          </td>
          <td className="px-6 py-4 text-nowrap truncate w-1/3 max-w-xs">
            {file.title}
          </td>
          <td className="px-6 py-4 whitespace-nowrap">
            {file.user.firstName + " " + file.user.lastName}
          </td>
          <td className="px-6 py-4 whitespace-nowrap">
            {file.createdDate
              ? new Date(file.createdDate).toLocaleDateString("en-GB")
              : "N/A"}
          </td>
          <td className="px-6 py-4 text-right">
            <Link
              to={`/file/${file.id}/${file.title}`}
              className="font-medium text-blue-600 hover:underline"
            >
              View
            </Link>
          </td>
        </tr>
      ))}
    </tbody>
  </table>
) : (
  <p className="text-center text-gray-500">No files available</p>
)}
```

```
      </div>
    </div>
  );
};

export default FileList;
```

Edit File

```
import { useState, useEffect, useRef, useMemo } from "react";
import { useParams, useNavigate } from "react-router-dom";
import { useForm } from "react-hook-form";
import axios from "axios";
import JoditEditor from "jodit-react";
import Cookies from "js-cookie";

const EditFile = () => {
  const { fileId } = useParams(); // Get fileId from URL
  const navigate = useNavigate();
  const { register, handleSubmit, setValue } = useForm();
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");
  const [editorValue, setEditorValue] = useState(""); // Local state for editor value
  const editor = useRef(null);

  // Retrieve userId from cookies
  const userId = Cookies.get("userId");

  useEffect(() => {
    if (!userId) {
      setError("User not authenticated.");
      return;
    }

    axios
      .get(`http://localhost:8080/files/${fileId}`)
      .then((response) => {
        setValue("title", response.data.title); // Set form title
        setEditorValue(response.data.description || ""); // Set editor value
        setLoading(false);
      })
      .catch(() => {
        setError("Failed to load file details.");
        setLoading(false);
      });
  }, [fileId, setValue, userId]);

  // JoditEditor configuration to match UploadFile
  const editorConfig = useMemo(
    () => ({
      readonly: false,
      toolbar: true,
      uploader: {
```

```
      insertImageAsBase64URI: true,
      url: (file) => {
        const reader = new FileReader();
        reader.onload = (e) => {
          const imgDataUrl = e.target.result;
          editor.current.selection.insertImage(imgDataUrl);
        };
        reader.readAsDataURL(file);
      },
    },
  }),
  []
);

const handleEditorChange = (newContent) => {
  setEditorValue(newContent);
};

const onSubmit = async (data) => {
  try {
    data.description = editorValue; // Set the editor's content to the form data
    await axios.put(
      `http://localhost:8080/files/update/${fileId}/${userId}`,
      data
    );
    alert("File updated successfully!");
    navigate("/");
  } catch (err) {
    setError("Failed to update file.");
  }
};

return (
  <div className="max-w-5xl mx-auto mt-10 p-6 bg-white shadow-lg rounded-lg">
    <h2 className="text-2xl font-semibold text-gray-800 mb-4">Edit File</h2>

    {error && <p className="text-red-500">{error}</p>}
    {loading ? (
      <p>Loading...</p>
    ) : (
      <form onSubmit={handleSubmit(onSubmit)} className="space-y-4">
        <div>
          <label className="block font-medium">Title</label>
          <input
            {...register("title")}
            className="w-full p-2 border rounded-lg"
            required
          />
        </div>

        <div>
          <label className="block font-medium">Description</label>
```

```
          <JoditEditor
            ref={editor}
            value={editorValue}
            onChange={handleEditorChange}
            config={editorConfig}
            className="w-full mb-6 border-2 border-gray-300 rounded-lg"
            style={{ height: "600px", overflowY: "auto" }}
          />
        </div>

        <button
          type="submit"
          className="w-full bg-blue-600 text-white py-2 rounded-lg hover:bg-blue-700 transition"
        >
          Update File
        </button>
      </form>
    )}
  </div>
 );
};

export default EditFile;
```

Profile Edit

```
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { Eye, EyeOff } from "lucide-react";

const getCookie = (name) => {
  const cookieString = document.cookie
    .split("; ")
    .find((row) => row.startsWith(name + "="));
  return cookieString ? decodeURIComponent(cookieString.split("=")[1]) : null;
};

const ProfileEditPage = () => {
  const [user, setUser] = useState({
    firstName: "",
    lastName: "",
    email: "",
    password: "", // Added password field
  });

  const [showPassword, setShowPassword] = useState(false); // Moved here
  const [loading, setLoading] = useState(true);
  const navigate = useNavigate();

  useEffect(() => {
    const userId = getCookie("userId");

    if (!userId) {
```

```
    console.error("User ID not found in cookies");
    setLoading(false);
    return;
  }

  fetch(`http://localhost:8080/users/${userId}`)
    .then((res) => res.json())
    .then((data) => {
      setUser(data); // Populate user data with response
      setLoading(false);
    })
    .catch((err) => {
      console.error("Error fetching user:", err);
      setLoading(false);
    });
}, []);

const handleChange = (e) => {
  // Ensure the password field updates correctly
  setUser({ ...user, [e.target.name]: e.target.value });
};

const handleSubmit = async (e) => {
  e.preventDefault();
  const userId = getCookie("userId");

  try {
    const response = await fetch(`http://localhost:8080/users/${userId}`, {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(user),
    });

    if (response.ok) {
      alert("Profile updated successfully!");
      navigate("/profile");
    } else {
      alert("Failed to update profile.");
    }
  } catch (error) {
    console.error("Error updating profile:", error);
  }
};

return (
  <div className="flex items-center justify-center min-h-screen bg-gray-100">
    <div className="bg-white shadow-lg rounded-lg p-8 w-full max-w-md">
      <h2 className="text-2xl font-bold text-center text-gray-700 mb-6">
        Edit Profile
      </h2>

      {loading ? (
```

```
  <p className="text-center text-gray-500">Loading...</p>
) : (
  <form onSubmit={handleSubmit} className="space-y-4">
    <div>
      <label className="block text-gray-700 font-medium">First Name</label>
      <input
        type="text"
        name="firstName"
        value={user.firstName}
        onChange={handleChange}
        className="w-full p-3 border border-gray-300 rounded-lg focus:ring focus:ring-blue-300"
        required
      />
    </div>

    <div>
      <label className="block text-gray-700 font-medium">Last Name</label>
      <input
        type="text"
        name="lastName"
        value={user.lastName}
        onChange={handleChange}
        className="w-full p-3 border border-gray-300 rounded-lg focus:ring focus:ring-blue-300"
        required
      />
    </div>

    <div>
      <label className="block text-gray-700 font-medium">Email</label>
      <input
        type="email"
        name="email"
        value={user.email}
        onChange={handleChange}
        className="w-full p-3 border border-gray-300 rounded-lg focus:ring focus:ring-blue-300"
        required
      />
    </div>

    <div className="mb-4 relative">
      <label className="block text-gray-700 font-medium">Password</label>
      <div className="relative w-full">
        <input
          type={showPassword ? "text" : "password"}
          name="password"
          value={user.password}
          onChange={handleChange}
          className="w-full p-3 border border-gray-300 rounded-lg focus:ring focus:ring-blue-300"
        />
        <button
          type="button"
          className="absolute right-3 top-2.5 text-gray-500"
```

```
              onClick={() => setShowPassword(!showPassword)}
            >
              {showPassword ? <EyeOff size={20} /> : <Eye size={20} />}
            </button>
          </div>
          <p className="text-gray-500 text-sm">Leave blank to keep the current password.</p>
        </div>

        <button
          type="submit"
          className="w-full bg-blue-500 text-white py-3 rounded-lg font-medium hover:bg-blue-600
transition"
        >
          Save Changes
        </button>
      </form>
    )}
   </div>
  </div>
 );
};

export default ProfileEditPage;
```

Profile Page

```
import { useEffect, useState } from "react";
import { Link } from "react-router-dom";

const getCookie = (name) => {
  const cookieString = document.cookie
    .split("; ")
    .find((row) => row.startsWith(name + "="));
  return cookieString ? decodeURIComponent(cookieString.split("=")[1]) : null;
};

// Function to strip HTML tags from description
const stripHtml = (html) => {
  const doc = new DOMParser().parseFromString(html, "text/html");
  return doc.body.textContent || "";
};

const ProfilePage = () => {
  const [user, setUser] = useState(null);
  const [files, setFiles] = useState([]);

  useEffect(() => {
    const userId = getCookie("userId");

    if (!userId) {
      console.error("User ID not found in cookies");
      return;
    }
```

```
    // Fetch user details
    fetch(`http://localhost:8080/users/${userId}`)
      .then((res) => res.json())
      .then((data) => setUser(data))
      .catch((err) => console.error("Error fetching user:", err));

    // Fetch user files
    fetch(`http://localhost:8080/files/user/${userId}`)
      .then((res) => res.json())
      .then((data) => setFiles(Array.isArray(data) ? data : []))
      .catch((err) => console.error("Error fetching files:", err));
  }, []);

  const handleDelete = async (fileId) => {
    const userId = getCookie("userId");
    if (!userId) return;

    if (confirm("Are you sure you want to delete this file?")) {
      try {
        await fetch(`http://localhost:8080/files/delete/${fileId}/${userId}`, {
          method: "DELETE",
        });

        setFiles((prevFiles) => prevFiles.filter((file) => file.id !== fileId));
      } catch (error) {
        console.error("Error deleting file:", error);
      }
    }
  };

  return (
    <div className="container mx-auto p-4 max-w-5xl">
      {user ? (
  <div className="bg-white shadow-lg rounded-2xl p-6 mb-8 flex flex-col items-center text-center w-full
max-w-md mx-auto">
    {/* Profile Icon */}
    <div className="w-24 h-24 bg-blue-100 text-blue-500 flex items-center justify-center rounded-full text-
3xl font-bold mb-4">
      {user.firstName.charAt(0)}
      {user.lastName.charAt(0)}
    </div>

    {/* User Details */}
    <h2 className="text-2xl font-semibold text-gray-800">
      {user.firstName} {user.lastName}
    </h2>
    <p className="text-gray-500 text-sm capitalize">{user.gender}</p>
    <p className="text-gray-600 mt-1">{user.email}</p>

    {/* Edit Profile Button */}
    <Link
```

```
      to="/profile-edit"
      className="mt-4 bg-blue-600 hover:bg-blue-700 text-white font-medium px-6 py-2 rounded-lg
transition duration-300"
    >
      Edit Profile
    </Link>
  </div>
) : (
  <p className="text-gray-500 text-center">Loading user...</p>
)}


      <h3 className="text-xl font-semibold mb-4">Uploaded Files</h3>
      <div className="space-y-4">
        {files.length > 0 ? (
          files.map((file) => (
            <div
              key={file.id}
              className="bg-white shadow-lg rounded-lg p-6 mb-6 w-full   flex justify-between items-center"
            >
              <div className="h-20 w-[80%] ">
                <h4 className="font-bold">{file.title}</h4>
                <p className="text-gray-700 text-nowrap truncate  w-full   mt-3">
                  <strong>Description:</strong>{" "}
                  {file.description
                    ? stripHtml(file.description)
                    : "No description available."}
                </p>
              </div>
              <div className="space-x-2">
                <Link
                  to={`/edit/${file.id}/${getCookie("userId")}/${file.title}`}
                  className="bg-yellow-500 text-white px-3 py-1 rounded"
                >
                  Edit
                </Link>
                <button
                  className="bg-red-500 text-white px-3 py-1 rounded"
                  onClick={() => handleDelete(file.id)}
                >
                  Delete
                </button>
              </div>
            </div>
          ))
        ) : (
          <p>No files found.</p>
        )}
      </div>
    </div>
  );
};
```

export default ProfilePage;

Backend Code

Profile Page

a) File Controller

```
package com.example.filestoring.controller;

import com.example.filestoring.model.FileEntity;
import com.example.filestoring.service.FileService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;

@CrossOrigin(origins = "http://localhost:5173")
@RestController
@RequestMapping("/files")
public class FileController {

    @Autowired
    private FileService fileService;

    @PostMapping("/upload/{userId}")
    public FileEntity uploadFile(@PathVariable Long userId,
                                 @RequestParam("file") MultipartFile file,
                                 @RequestParam("title") String title,
                                 @RequestParam("description") String description) throws Exception {
        return fileService.uploadFile(userId, file, title, description);
    }

    @GetMapping("/all")
    public List<FileEntity> getAllFiles() {
        return fileService.getAllFiles();
    }

    @GetMapping("/user/{userId}")
    public List<FileEntity> getUserFiles(@PathVariable Long userId) {
        return fileService.getUserFiles(userId);
```

```
    }

    @GetMapping("/{fileId}")
    public ResponseEntity<FileEntity> getFileById(@PathVariable Long fileId) {
        FileEntity file = fileService.getFileById(fileId);
        return file != null ? ResponseEntity.ok(file) : ResponseEntity.notFound().build();
    }

    @PutMapping("/update/{fileId}/{userId}")
    public FileEntity updateFile(@PathVariable Long fileId,
                                 @PathVariable Long userId,
                                 @RequestBody FileEntity updatedFile) throws Exception {
        return fileService.updateFile(fileId, updatedFile.getTitle(),
updatedFile.getDescription(), userId);
    }

    @DeleteMapping("/delete/{fileId}/{userId}")
    public String deleteFile(@PathVariable Long fileId, @PathVariable Long userId) throws
Exception {
        fileService.deleteFile(fileId, userId);
        return "File deleted successfully!";
    }

    @GetMapping("/download/{fileId}")
    public ResponseEntity<Resource> downloadFile(@PathVariable Long fileId) throws
Exception {
        return ResponseEntity.ok()
                .body(fileService.downloadFile(fileId));
    }

    @GetMapping("/view/{fileName}")
    public ResponseEntity<Resource> viewFile(@PathVariable String fileName) throws
Exception {
        return fileService.viewFile(fileName);
    }
}
```

b) UserController

```
package com.example.filestoring.controller;

import com.example.filestoring.model.User;
import com.example.filestoring.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;



@CrossOrigin(origins = "http://localhost:5173") // Allow React frontend
@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService userService;

    // Home endpoint
    @GetMapping("/")
    public String home() {
        return "This is home";
    }

    @PostMapping("/login")
    public ResponseEntity<?> loginUser(@RequestBody Map<String, String> requestData) {
        String email = requestData.get("email");
        String password = requestData.get("password");

        User user = userService.findUserByEmail(email);

        if (user == null || !user.getPassword().equals(password)) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid credentials");
        }

        // ☑ Return only required fields
        Map<String, Object> response = new HashMap<>();
        response.put("userId", user.getId());  // Ensure ID is correctly sent
        response.put("email", user.getEmail());

        return ResponseEntity.ok(response);
    }



    // ☑ Fetch all users
    @GetMapping
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    @PostMapping("/register")
```

```
    public ResponseEntity<?> registerUser(@RequestBody User user) {
        User savedUser = userService.registerUser(user);

        if (savedUser == null) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("User registration failed");
        }

        // ☑ Return only necessary data
        Map<String, Object> response = new HashMap<>();
        response.put("userId", savedUser.getId());
        response.put("email", savedUser.getEmail());

        return ResponseEntity.ok(response);
    }


    // ☑ Find user by ID
    @GetMapping("/{id}")
    public User getUserById(@PathVariable Long id) throws Exception {
        return userService.findUserById(id);
    }

    // ☑ Find user by email
    @GetMapping("/email/{email}")
    public User getUserByEmail(@PathVariable String email) {
        return userService.findUserByEmail(email);
    }

    // ☑ Update user details
    @PutMapping("/{id}")
    public User updateUser(@RequestBody User user, @PathVariable Long id) throws Exception {
        return userService.updateUser(user, id);
    }

    // ☑ Search users by name or email
    @GetMapping("/search")
    public List<User> searchUsers(@RequestParam String query) {
        return userService.serachUser(query);
    }
}
```

c) File Entity

```
package com.example.filestoring.model;

import jakarta.persistence.*;
import java.util.Date;
import java.util.Objects;

@Entity
@Table(name = "files")
```

```
public class FileEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @Lob
    private String description;

    private String fileUrl;

    private String filePath;  // File path stored in DB

    @Temporal(TemporalType.TIMESTAMP) // Ensures correct DB mapping
    @Column(nullable = false, updatable = false) // Prevents modification after creation
    private Date createdDate;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    public FileEntity() {}

    public FileEntity(String title, String description, String filePath, String fileUrl, User user) {
        this.title = title;
        this.description = description;
        this.filePath = filePath;
        this.fileUrl = fileUrl;
        this.user = user;
    }

    @PrePersist
    protected void onCreate() {
        this.createdDate = new Date(); // Auto-set createdDate before persisting
    }

    // Getters and Setters

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }
```

```
public void setTitle(String title) {
    this.title = title;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public String getFilePath() {
    return filePath;
}

public void setFilePath(String filePath) {
    this.filePath = filePath;
}

public Date getCreatedDate() {
    return createdDate;
}

public void setCreatedDate(Date createdDate) {
    this.createdDate = createdDate;
}

public User getUser() {
    return user;
}

public void setUser(User user) {
    this.user = user;
}

public String getFileUrl() {
    return fileUrl;
}

public void setFileUrl(String fileUrl) {
    this.fileUrl = fileUrl;
}

// Override toString, hashCode, and equals for better debugging and comparison

@Override
public String toString() {
    return "FileEntity{" +
            "id=" + id +
            ", title='" + title + '\'' +
            ", description='" + description + '\'' +
```

```
                ", fileUrl='" + fileUrl + '\'' +
                ", filePath='" + filePath + '\'' +
                ", createdDate=" + createdDate +
                ", user=" + user +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        FileEntity that = (FileEntity) o;
        return Objects.equals(id, that.id);
    }

    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
}
```

C) User Enitity

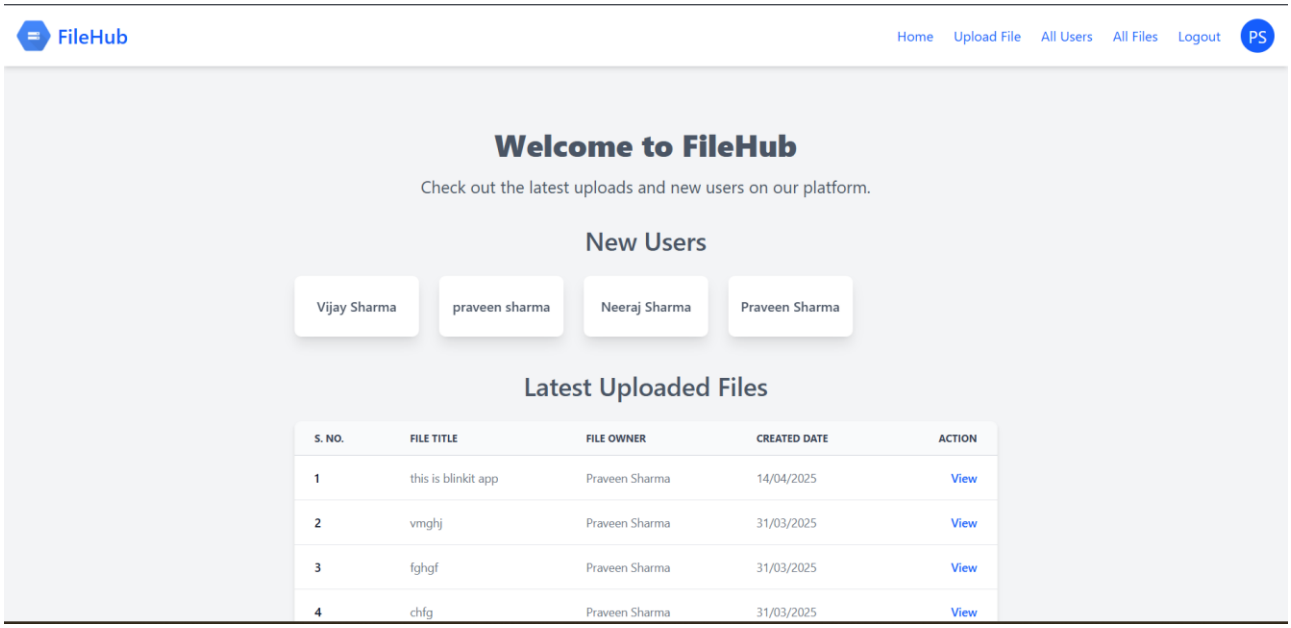
```
package com.example.filestoring.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity
@Table(name= "users")
public class User {

@Id @GeneratedValue(strategy=GenerationType.IDENTITY)
private Long id;

private String firstName;

private String lastName;

private String email;

private String password;

private String gender;



public User() {
```

```
		// TODO Auto-generated constructor stub
}


public User(Long id, String firstName, String lastName, String email, String password, String gender) {
		super();
		this.id = id;
		this.firstName = firstName;
		this.lastName = lastName;
		this.email = email;
		this.password = password;
		this.gender = gender;

}


public Long getId() {
		return id;
}


public void setId(Long id) {
		this.id = id;
}


public String getFirstName() {
		return firstName;
}


public void setFirstName(String firstName) {
		this.firstName = firstName;
}


public String getLastName() {
		return lastName;
}


public void setLastName(String lastName) {
		this.lastName = lastName;
}


public String getEmail() {
		return email;
}


public void setEmail(String email) {
```

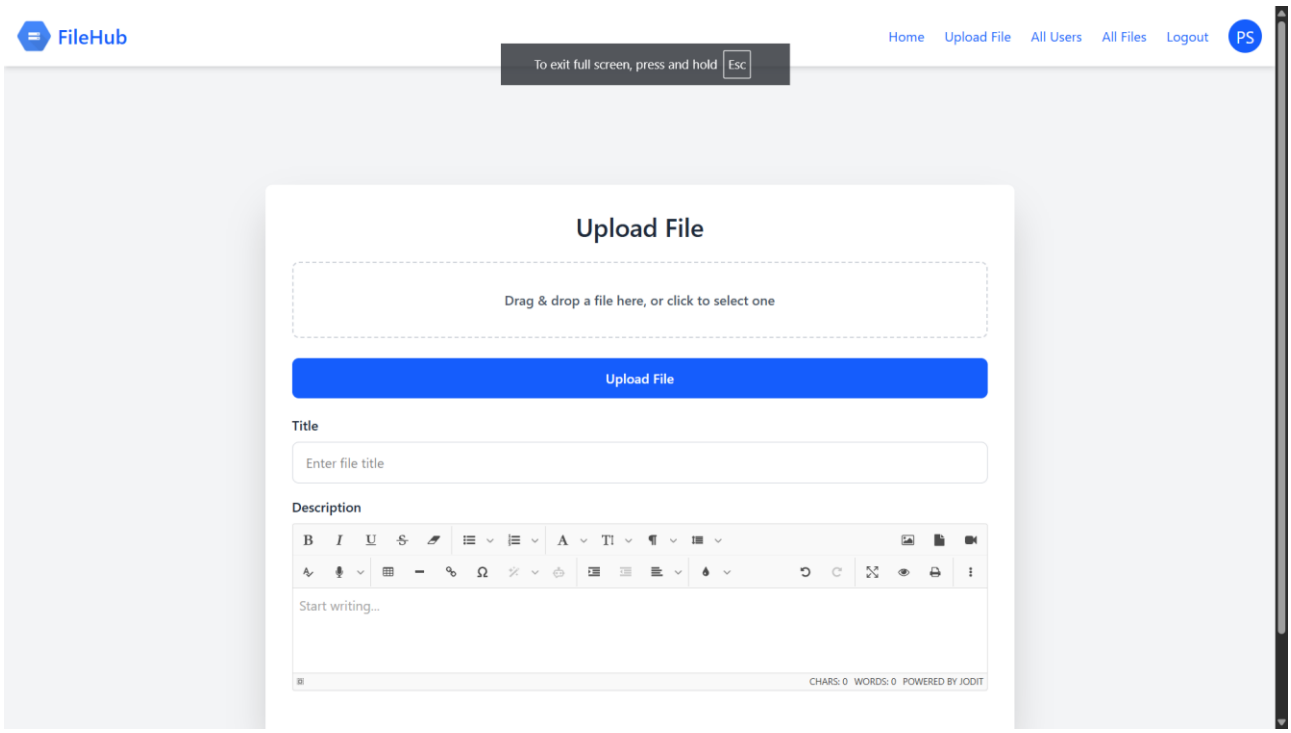
```
        this.email = email;
}


public String getPassword() {
        return password;
}


public void setPassword(String password) {
        this.password = password;
}


public String getGender() {
        return gender;
}


public void setGender(String gender) {
        this.gender = gender;
}


}
```

Screenshot

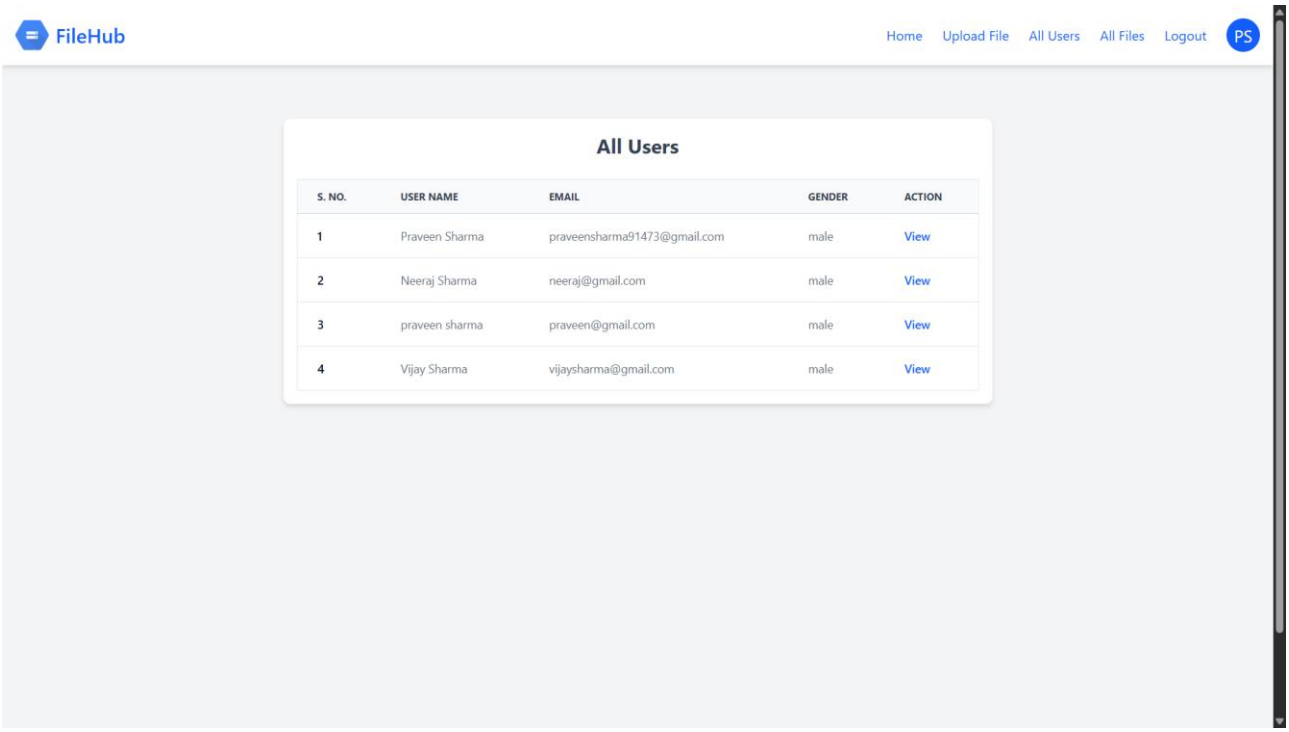
Home Page



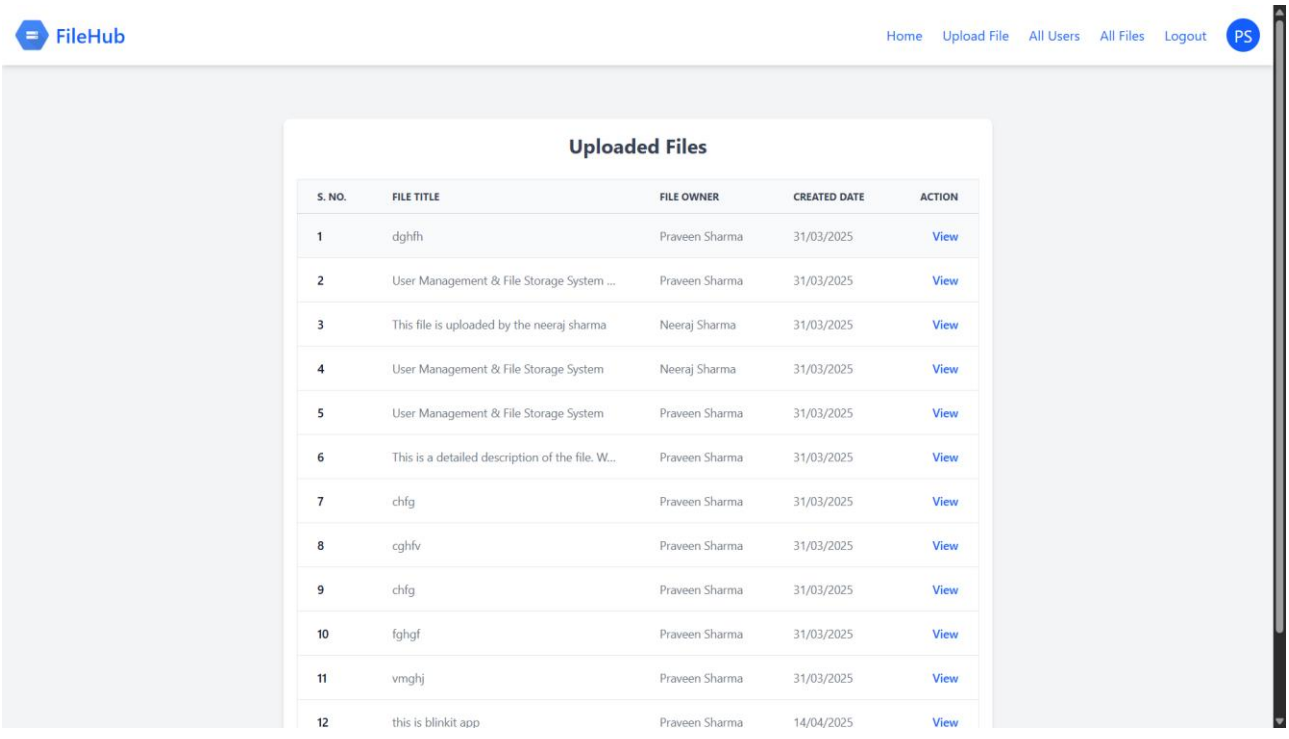
Upload File



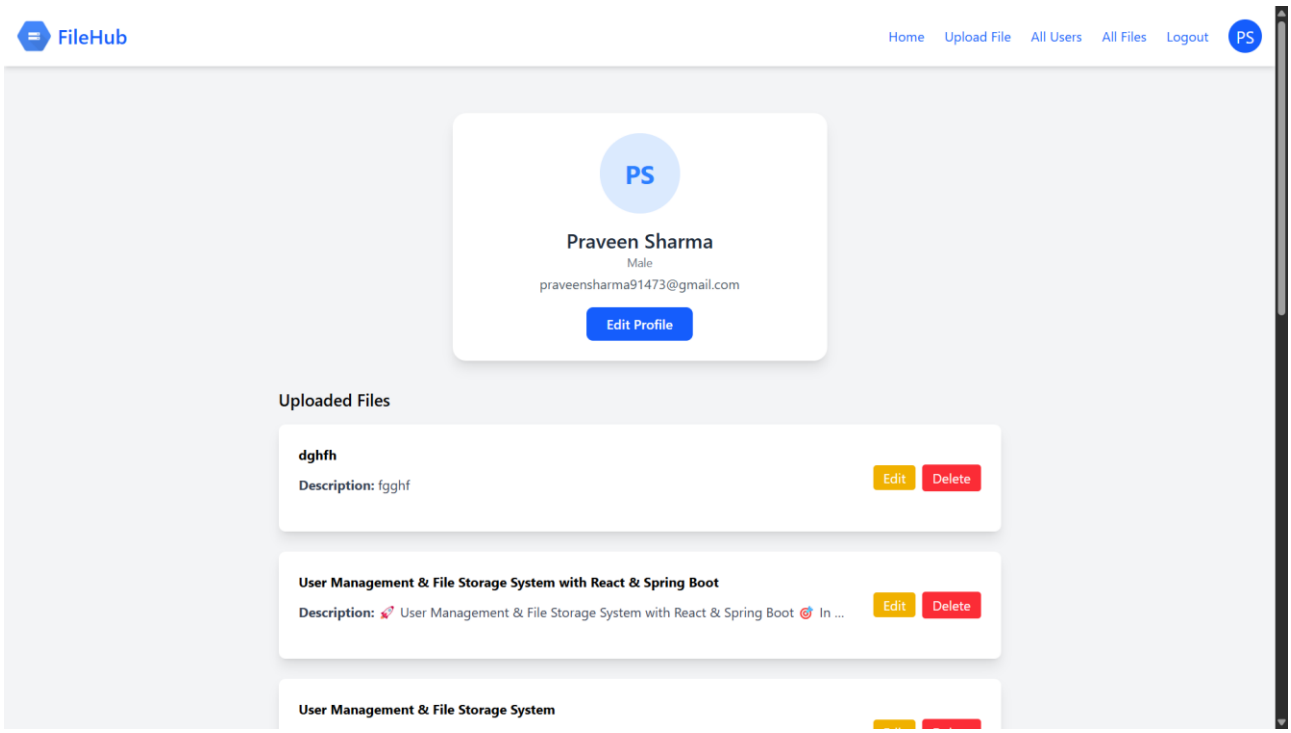
All Users



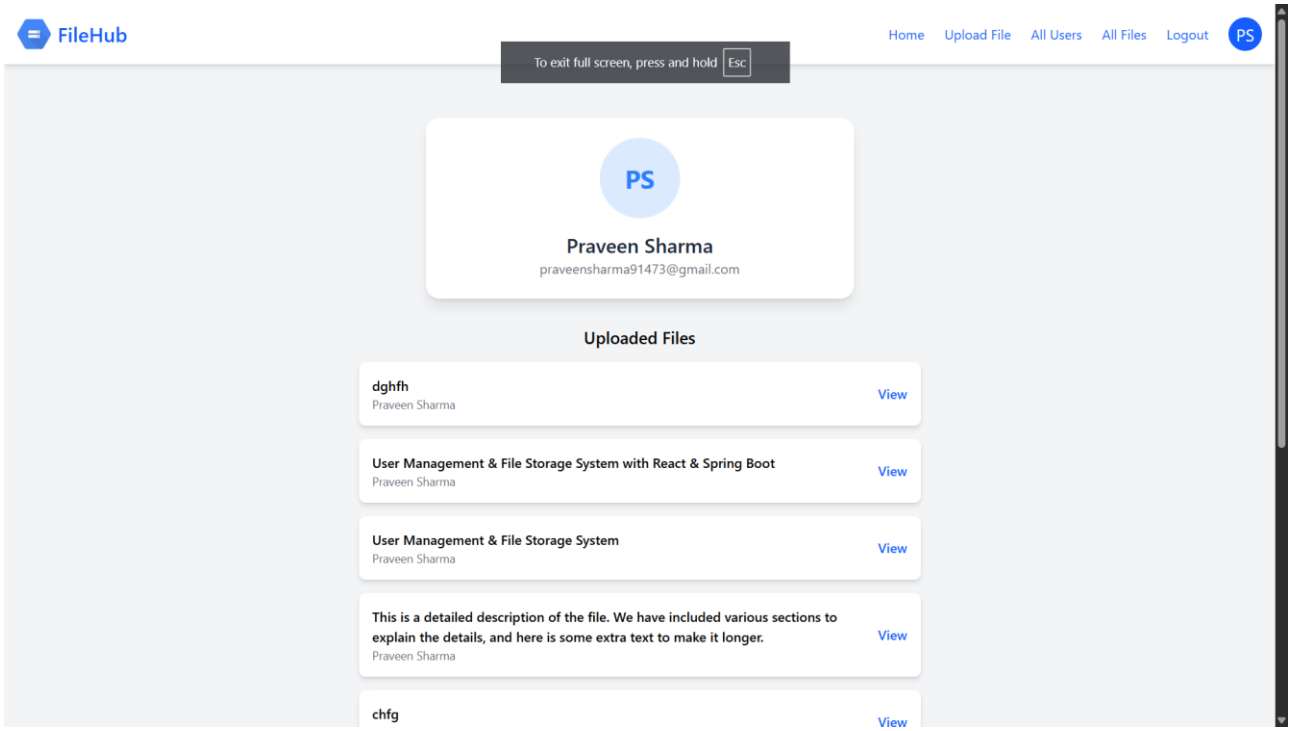
All Files



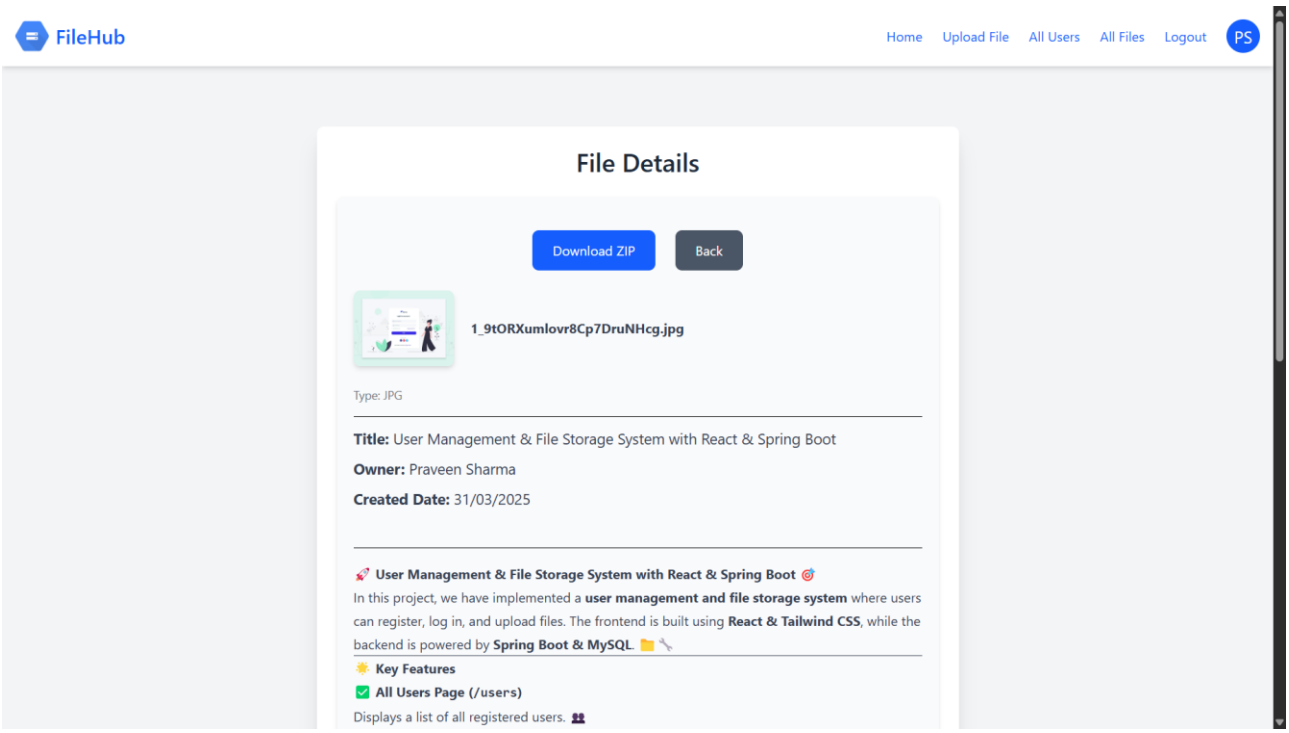
Profile Page



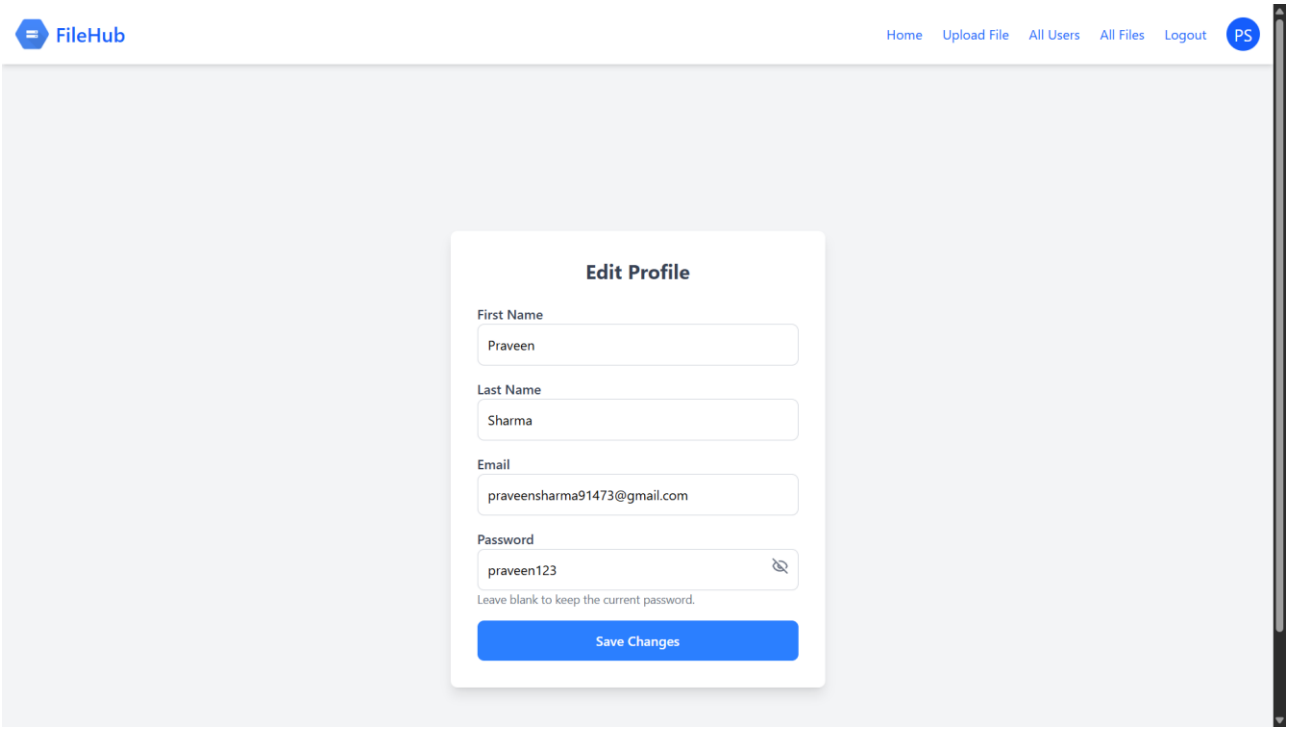
User Profile



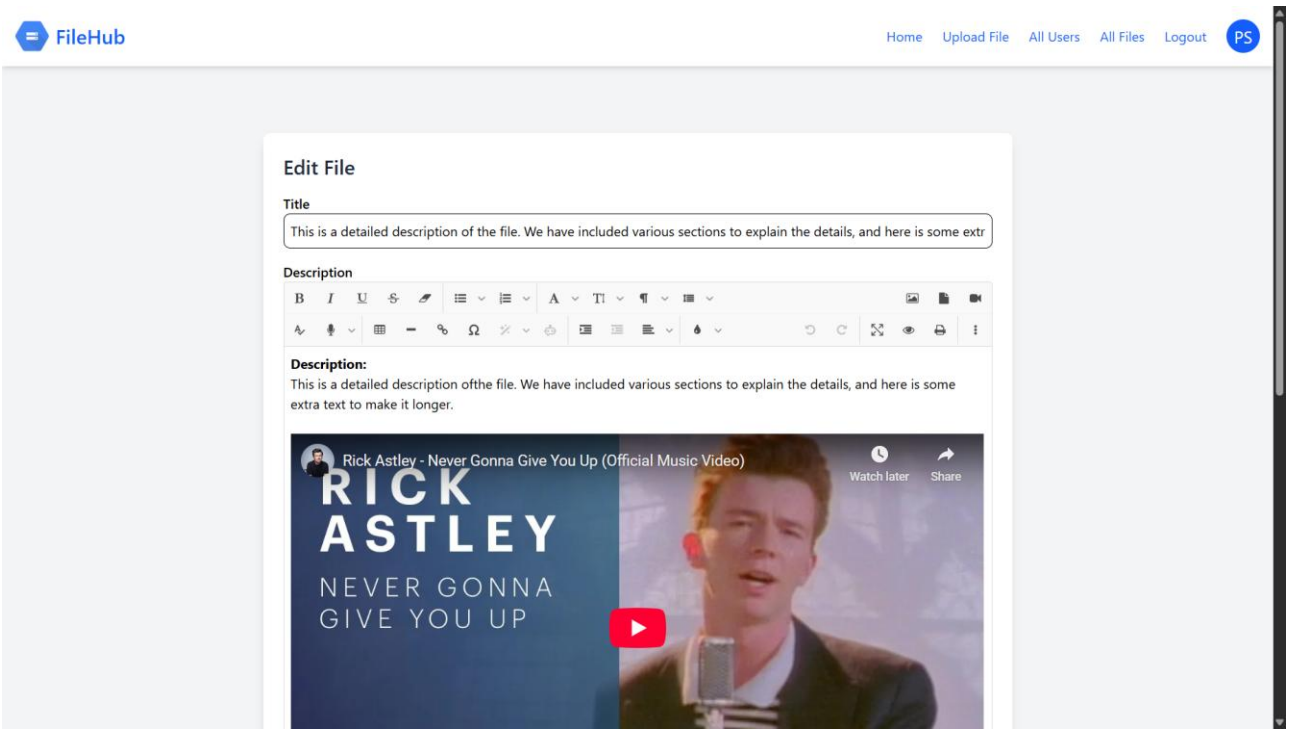
File View



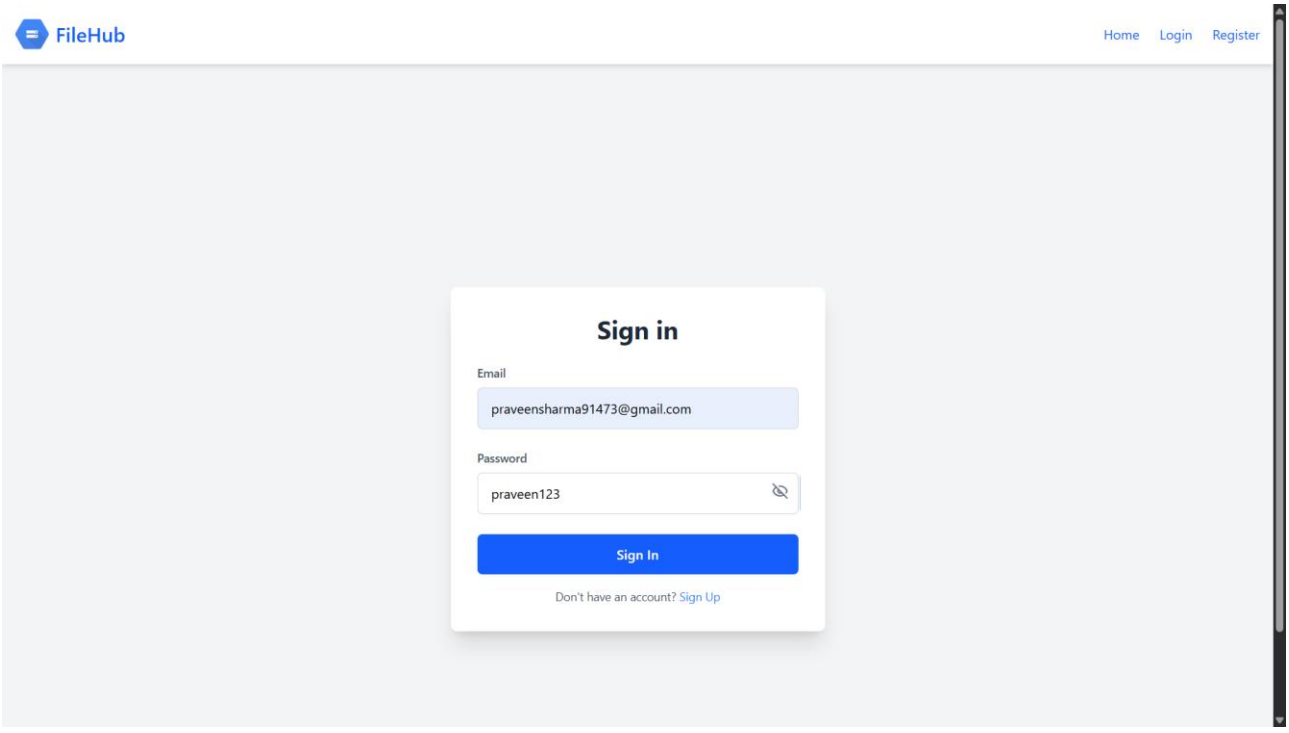
Edit File



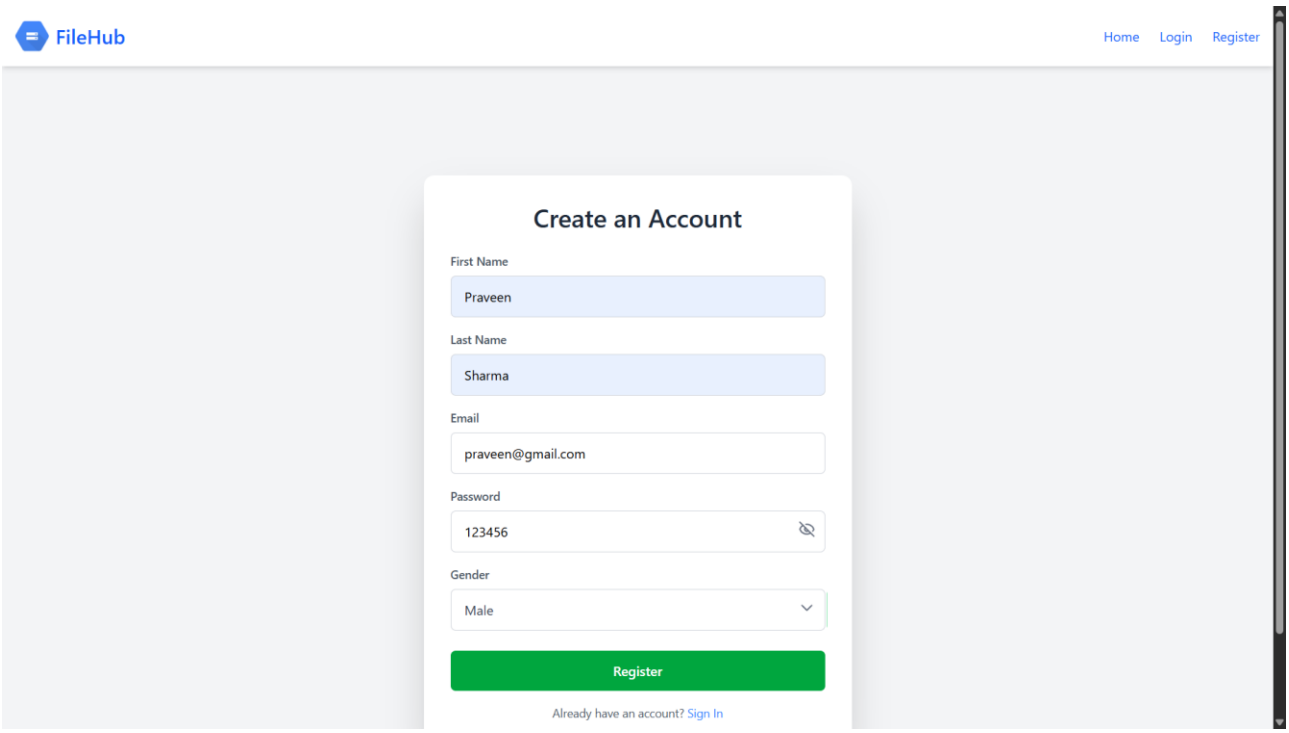
Edit File


FileHub
Home
Upload File
All Users
All Files
Logout
PS
Edit File
Title
This is a detailed description of the file. We have included various sections to explain the details, and here is some extr
Description
Description:
This is a detailed description ofthe file. We have included various sections to explain the details, and here is some extra text to make it longer.
Rick Astley - Never Gonna Give You Up (Official Music Video)
Watch later
Share
RICK ASTLEY
NEVER GONNA GIVE YOU UP

Login


FileHub
Home
Login
Register
Sign in
Email
praveensharma91473@gmail.com
Password
praveen123
Sign In
Don't have an account? Sign Up

Register Page



6. Testing

Manual: Manual testing is a software testing process in which test cases are executed manually without using any automated tool.

Testing Technique:

White box testing: White box testing techniques analyse the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing.

Approach:

Top to bottom: In testing, the top to bottom approach refers to carrying out testing from top to bottom, according to the control flow of the system. We test the higher-level modules first, and then the lower-level modules.

Test cases:

Sr. No.	Test id	Test case	Input valid	Input invalid	Output expected	Actual Output	Pass / Fail
Login							
1	Alma 01	Login id	Hemant5@Gmail.com		Login	Login	Pass
2	Alma 02	Login id		Kumawat.in	Login	Error	Fail
3	Alma 03	Login Password	Kjhf25465@8		Login	Login	Pass
4	Alma 04	Login Password		65	Login	Error	Fail
Registration							
5	Alma 05	Registration	21K/4043		verify	Successful	Pass
6	Alma 06	Registration		Hemant	verify	Not successful	Fail
Create Profile							
7	Alma 07	Profile name	sachin		update	Updated	Pass
8		Profile name		skjdf	update	Not updated	Fail
9	Alma 08	Mobile No.	8754123692		update	Updated	Pass
10	Alma 09	Mobile No.		54213475d	update	Not updated	Fail
11	Alma 10	Email id	Naresh5@gmail.com		update	Updated	Pass
12	Alma 11	Email id		Nareshkumar.com	update	Not updated	Fail
13	Alma 12	Passing year	2023		update	Updated	Pass
14	Alma 13	Passing year		8h899	update	Not updated	Fail
15	Alma 14	Branch	CSE		update	Updated	Pass
16	Alma 15	Branch		sdkfj	update	Not updated	Fail
17	Alma 16	Research field	computer		update	Updated	Pass
18	Alma 17	Research field		phd	update	Not updated	Fail
19	Alma 18	Work experience	5 year		update	Updated	Pass
20	Alma 19	Work experience		6	update	Not updated	Fail
21	Alma 20	Project during college	Ai project		update	Updated	Pass
22	Alma 21	Project during college		5 project	update	Not updated	Fail
23	Alma 22	Worked in	amozon		update	Updated	Pass
24	Alma 23	Worked in		45554	update	Not updated	Fail
25	Alma 24	Current position	manager		update	Updated	Pass
26	Alma 25	Current position		6kdk	update	Not updated	Fail
Generate Id Password							
28		Create id	Hemant6@gmail.com		Create id	Successful	Pass
29	Alma 26	Create id		Kjd78sad	Create id	Not successful	Fail
30	Alma 27	Create password	fjh@763k		Create password	Successful	Pass
31	Alma 28	Create password		564	Create password	Not successful	Fail
Verify the user							
32	Alma 29	Enrollment number	21T/2132		Verify	Successful	Pass

33	Alma 30	Enrollment number		85569/74F	Verify	Not successful	Fail
34	Alma 31	Name	Satish		Verify	Successful	Pass
35	Alma 32	Name		kumar	Verify	Not successful	Fail
36	Alma 33	Batch	2014		Verify	Successful	Pass
37	Alma 34	Batch		3045	Verify	Not successful	Fail

7. System Security

1. Access Controls:
    - **Checkpoint:** Regularly review and update user access permissions.
    - **Control:** Implement role-based access controls (RBAC) to restrict access based on job roles and responsibilities.
2. Encryption:
    - **Checkpoint:** Enable encryption for sensitive project files.
    - **Control:** Use strong encryption algorithms and manage encryption keys securely.
3. Backup and Recovery:
    - **Checkpoint:** Establish a regular backup schedule.
    - **Control:** Store backup copies in an offsite location to prevent data loss due to physical incidents.
4. Version Control:
    - **Checkpoint:** Regularly check and commit changes to a version control system.
    - **Control:** Implement a version control policy, restricting direct edits to master files without proper documentation.
5. Firewalls and Antivirus Software:
    - **Checkpoint:** Ensure firewalls and antivirus software are up-to-date.
    - **Control:** Schedule regular scans and updates for firewalls and antivirus definitions.

Control Points:

1. Audit Trail:
    - **Checkpoint:** Enable and review audit trails for file access and modifications. **Control:** Implement automated alerts for suspicious activities and conduct periodic manual audits.
2. Secure File Transfer**:**
    - **Checkpoint:** Use secure transfer methods for exchanging files.
    - **Control:** Monitor and log file transfers, and restrict access to secure transfer protocols.
3. Physical Security:
    - **Checkpoint:** Limit physical access to servers and storage.
    - **Control:** Implement physical security measures such as biometric access controls and surveillance.
4. Employee Training:
    - **Checkpoint:** Provide security training for employees.
    - **Control:** Conduct regular security awareness programs and phishing simulations.
5. Regular Security Audits:
    - **Checkpoint:** Perform regular security audits.
    - **Control:** Develop a comprehensive security audit plan and address any vulnerabilities or noncompliance promptly.
6. Legal and Compliance Considerations:
    - **Checkpoint:** Ensure compliance with data protection regulations.
    - **Control:** Regularly review and update policies to align with changing legal requirements.

7. Conclusion/Future Enhancement

Conclusion:

1. Project Objectives:

- Summarize the achievement of project objectives.
- Highlight any significant milestones or deliverables.

2. Successes and Challenges:
    - Discuss successful aspects of the project. Address challenges faced and how they were overcome.
3. User Feedback:
    - Include feedback from end-users, stakeholders, or clients. Highlight positive feedback and any suggested improvements.
4. Lessons Learned:
    - Reflect on lessons learned during the project. Identify what worked well and areas for improvement in future projects.
5. Compliance and Security:
    - Confirm compliance with relevant regulations. Discuss the effectiveness of security measures implemented.
6. Project Timeline and Budget:
    - Evaluate project completion within the scheduled timeline and budget. Note any deviations and the reasons behind them.
7. Collaboration and Communication:
    - Assess the effectiveness of team collaboration and communication.
    - Identify tools or strategies that were particularly helpful.

Future Enhancements:

1. Feature Enhancements:
    - Outline potential additional features or improvements to existing features.
    - Consider feedback received from users or stakeholders.
2. Scalability:
    - Discuss plans for scalability as user or data volumes increase.
    - Consider technologies or architectures that support scalability.
3. User Experience (UX) Improvements:
    - Explore opportunities to enhance the user interface and overall user experience. Consider incorporating user feedback for a more user-friendly design.
4. Integration with Other Systems:
    - Identify potential integrations with other systems or platforms. Enhance interoperability for a seamless user experience.
5. Automation Opportunities:
    - Explore automation possibilities to streamline processes.
    - Consider incorporating machine learning or artificial intelligence for intelligent automation.
6. Performance Optimization:
    - Address any performance bottlenecks and propose optimizations. Consider enhancements for faster processing or response times.
7. Data Analytics and Reporting:
    - Integrate advanced analytics or reporting features. Provide users with more insightful data visualization tools.
8. Mobile Compatibility:
    Evaluate the need for mobile compatibility and responsive design. Enhance accessibility for users on various devices.
9. Continuous Security Improvements:
    - Outline plans for ongoing security measures and improvements.
    - Stay updated on the latest security standards and technologies.
10. Training and Documentation:
    - Develop additional training materials and documentation.

- Ensure resources are available for onboarding new users or team members.

11. Community and User Engagement:
    - Foster a community around the project.
    - Encourage user engagement and feedback for continuous improvement.

8. Bibliography

Books:

Book title: Spring Boot, React, and Tailwind CSS
Author: Steven M. Schafer
Date of publication: 2010

Book title: HTML & CSS: Design and Build Web Sites
Author: Jon Duckett
Date of publication: 2011

Websites:

https://www.w3schools.com/springboot/default.asp

https://www.javatpoint.com/react-tutorial

https://www.tutorialspoint.com/tailwind