**Online Notes Sharing Platform**

This platform is used to share the notes over the internet between different users. Notes that the user wants for the particular subject can be found in the website.

**Objectives:**

* To provide a platform for students to deals with borrowing and transferring the notes over the internet.
* Used for students interested in borrowing and giving their own notes for the free of cost.
* Provide the digital platform for students where they can request notes as per required and another students can supply that notes.
* To ensure that the notes circulation is occurred within the students.

**Features:**

* User registration and login system.
* User profile system.
* Note sharing with other users.
* Note creation, editing and deletion.
* Note search.
* Note download.
* User rating system for notes.

**Technology:**

The platform can be developed using modern web technologies like HTML, CSS and JavaScript for the frontend, Node JS for backend, and Mongo DB for the database.

**System users:**

* Administrator
* User

**Administrator feature in online notes sharing platform:**

* Manage user accounts, including creating new accounts, editing existing accounts and deactivating accounts if necessary.
* Manage the content that is shared on the platform. This may include creating and editing notes, organizing notes into categories or collection.
* Manage whole website control.

**User features:**

* User should be able to create and manage their own accounts, including updating their profile information and resetting their password if necessary.
* User should be able to create and share their own notes on the platform.
* User should be able to search for notes.

Signup/front-end

  const navigate = useNavigate();

    const [formData, setFormData] = useState({

        name: "",

        email: "",

        password: "",

        confirmPassword: ""

    });

    const handleChange = (event) => {

        const { name, value } = event.target;

        setFormData((prevData) => ({ ...prevData, [name]: value }));

    };

    const handleSubmit = async (event) => {

        event.preventDefault();

        if (!formData.name || !formData.email || !formData.password || !formData.confirmPassword) {

            alert("Please fill in all required fields.");

            return;

        }

        if (formData.password !== formData.confirmPassword) {

            alert("Passwords do not match.");

            return;

        }

        try {

            const response = await fetch("http://localhost:5000/signup", {

                method: "POST",

                headers: {

                    "Content-Type": "application/json"

                },

                body: JSON.stringify({

                    ...formData,

                    isAdmin: false // Assuming the initial value for isAdmin is false

                })

            });

            const data = await response.json();

            console.log("Response:", data);

            if (data.message === 'User created successfully') {

            navigate("/login");

            alert("User created successfully");

            } else {

                alert(data.message);

            }

        } catch (error) {

            console.error(error);

        }

    };

Signup/server

const express = require('express');

const app = express.Router();

const User = require('../models/user');

app.post('/', async (req, res) => {

  const { name, email, password, confirmPassword, isAdmin } = req.body; // Add isAdmin to destructured object

  if (password !== confirmPassword) {

    return res.status(400).json({ message: 'Passwords do not match' });

  }

  try {

    // Check if user already exists

    const existingUser = await User.findOne({ email });

    if (existingUser) {

      return res.status(400).json({ message: 'User already registered' });

    }

    // Create a new user

    const newUser = new User({ name, email, password, isAdmin }); // Include isAdmin

    await newUser.save();

    res.status(201).json({ message: 'User created successfully' });

  } catch (error) {

    res.status(500).json({ message: 'Server error' });

  }

});

module.exports = app;

login/frond-end

 const navigate = useNavigate();

    const [formData, setFormData] = useState({

        email: "",

        password: ""

    });

    const handleChange = (event) => {

        const { name, value } = event.target;

        setFormData((prevData) => ({ ...prevData, [name]: value }));

    };

    const handleSubmit = async (event) => {

        event.preventDefault();

        if (!formData.email || !formData.password) {

            alert("Please fill in both email and password fields.");

            return;

        }

        try {

            const response = await fetch("http://localhost:5000/login", {

                method: "POST",

                headers: {

                    "Content-Type": "application/json"

                },

                body: JSON.stringify(formData)

            });

            const data = await response.json();

            console.log("Response:", data);

            if (data.success) {

                if (data.isAdmin) {

                    navigate("/admin");

                } else {

                    navigate("/user");

                }

            }

            alert(data.message);

        } catch (error) {

            console.error(error);

        }

    };

Login/server

const express = require('express');

const app = express.Router();

const User = require('../models/user');

const jwt = require('jsonwebtoken');

app.post('/', async (req, res) => {

  const { email, password } = req.body;

  try {

    // Check if user exists

    const user = await User.findOne({ email });

    if (!user) {

      return res.status(401).json({ message: 'Invalid email or password' });

    }

    // Check if password is correct

    if (user.password !== password) {

      return res.status(401).json({ message: 'Invalid email or password' });

    }

    // Generate JWT token

    const token = jwt.sign({ email: user.email }, 'secretkey'); // Add token expiration time

    res.status(200).json({

      success: true,

      message: 'Login successful',

      token,

      isAdmin: user.isAdmin // Send the isAdmin property

    });

  } catch (error) {

    res.status(500).json({ success: false, message: 'Server error' });

  }

});

app.get('/', async (req, res) => {

  try {

    const email = req.query.email;

    const user = await User.findOne({ email });

    if (!user) {

      return res.status(404).json({ message: 'User not found' });

    }

    res.status(200).json({ name: user.name });

  } catch (error) {

    res.status(500).json({ message: 'Server error' });

  }

});

module.exports = app;

backend/index.js

const express = require('express');

const cors = require("cors");

require('./models/db');

const app = express();

app.use(express.urlencoded({extended: true}));

app.use(express.json());

app.use(cors());

// routers

app.use('/signup', require('./routes/signup'))

app.use('/login', require('./routes/login'))

app.listen(5000, () => {

    console.log(`App running on port 5000`)

});

Upload/frondend:

const [file, setFile] = useState(null);

  const [formData, setFormData] = useState({

    branch: '',

    subject: '',

    fileType: '',

    description: '',

  });

  const handleFileChange = (event) => {

    setFile(event.target.files[0]);

  };

  const handleSubmit = async (event) => {

    event.preventDefault();

    if (formData.branch && formData.subject && formData.fileType && formData.description && file) {

      const data = new FormData();

      data.append('notes-file', file);

      data.append('branch', formData.branch);

      data.append('subject', formData.subject);

      data.append('fileType', formData.fileType);

      data.append('description', formData.description);

      try {

        const response = await axios.post('http://localhost:5000/upload', data);

        console.log(response.data);

        // Show success alert

        alert('File uploaded successfully!');

        // Clear form fields or perform other actions as needed

        setFormData({

          branch: '',

          subject: '',

          fileType: '',

          description: '',

        });

        setFile(null);

      } catch (error) {

        console.error('Error uploading file:', error);

        // Show error alert

        alert('Error uploading file. Please try again.');

      }

    } else {

      alert('Please fill in all the fields before submitting.');

    }

  };

Upload/server:

const express = require('express');

const multer = require('multer');

const path = require('path');

const router = express.Router();

const Note = require('../models/Note');

const storage = multer.diskStorage({

  destination: function (req, file, cb) {

    cb(null, 'uploads/');

  },

  filename: function (req, file, cb) {

    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() \* 1E9);

    cb(null, file.fieldname + '-' + uniqueSuffix + path.extname(file.originalname));

  },

});

const upload = multer({ storage });

router.post('/', upload.single('notes-file'), async (req, res) => {

  const { branch, subject, fileType, description } = req.body;

  const filePath = req.file.path; // Store the path to the uploaded file in the database

  try {

    const newNote = new Note({

      branch,

      subject,

      fileType,

      description,

      filePath,

    });

    await newNote.save();

    res.status(201).json({ message: 'File uploaded successfully!' });

  } catch (error) {

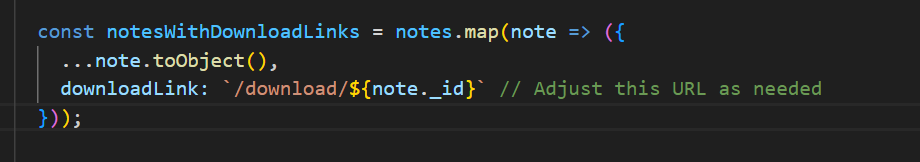
    console.error('Error uploading file:', error);

    res.status(500).json({ error: 'An error occurred while uploading the file.' });

  }

});

module.exports = router;



delete button/frontend

<td>

<button onClick={() => handleDelete(note.\_id)}>Delete</button>

</td>

  const handleDelete = (noteId) => {

    fetch(`http://localhost:5000/pendingnotes/${noteId}`, {

      method: 'DELETE',

    })

      .then((response) => {

        if (!response.ok) {

          throw new Error("Network response was not ok");

        }

        // Remove the deleted note from the state

        setNotes((prevNotes) => prevNotes.filter((note) => note.\_id !== noteId));

      })

      .catch((error) => console.error("Error deleting note:", error));

  };

Delete/backend

// Endpoint to delete a specific note by ID

router.delete('/:noteId', async (req, res) => {

  try {

    const noteId = req.params.noteId;

    const note = await Note.findById(noteId);

    if (!note) {

      return res.status(404).json({ error: 'Note not found' });

    }

    // Delete the note from the database

    await Note.findByIdAndDelete(noteId);

    // Optionally, you can also delete the corresponding file from your storage

    // Make sure to handle this part according to your file storage mechanism

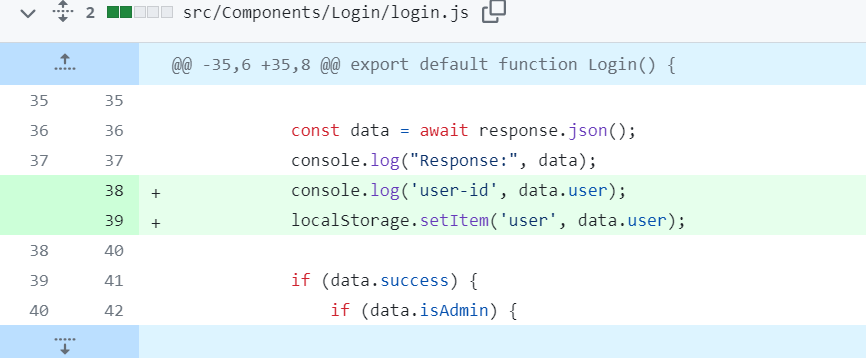
    res.json({ message: 'Note deleted successfully' });

  } catch (error) {

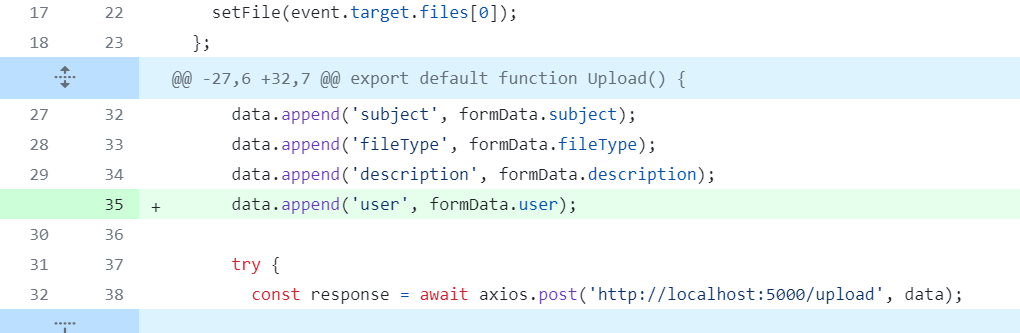
    res.status(500).json({ error: 'Internal server error' });

  }

});

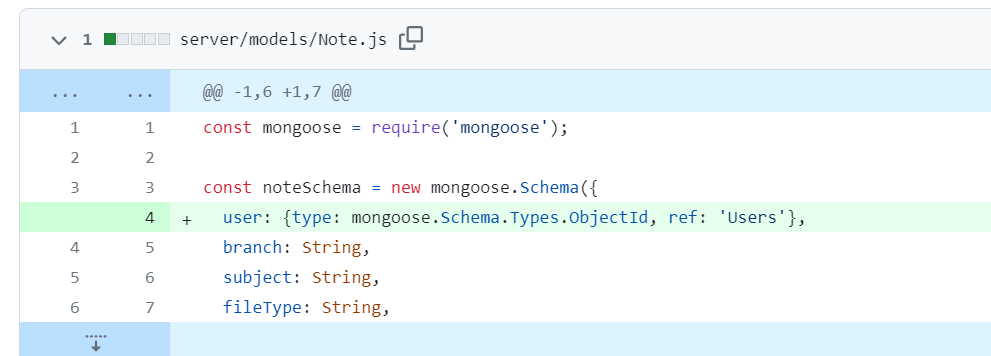


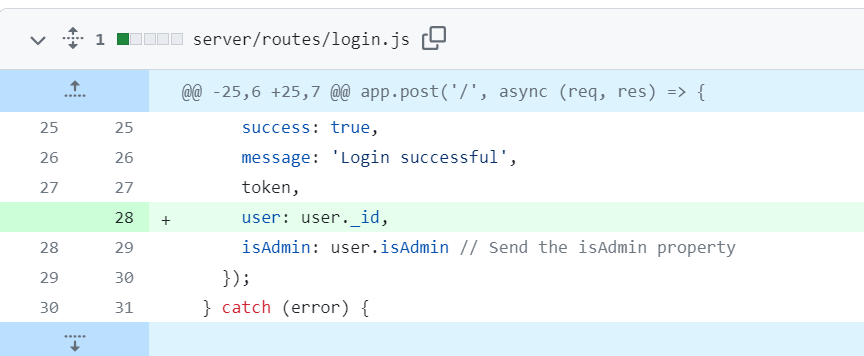


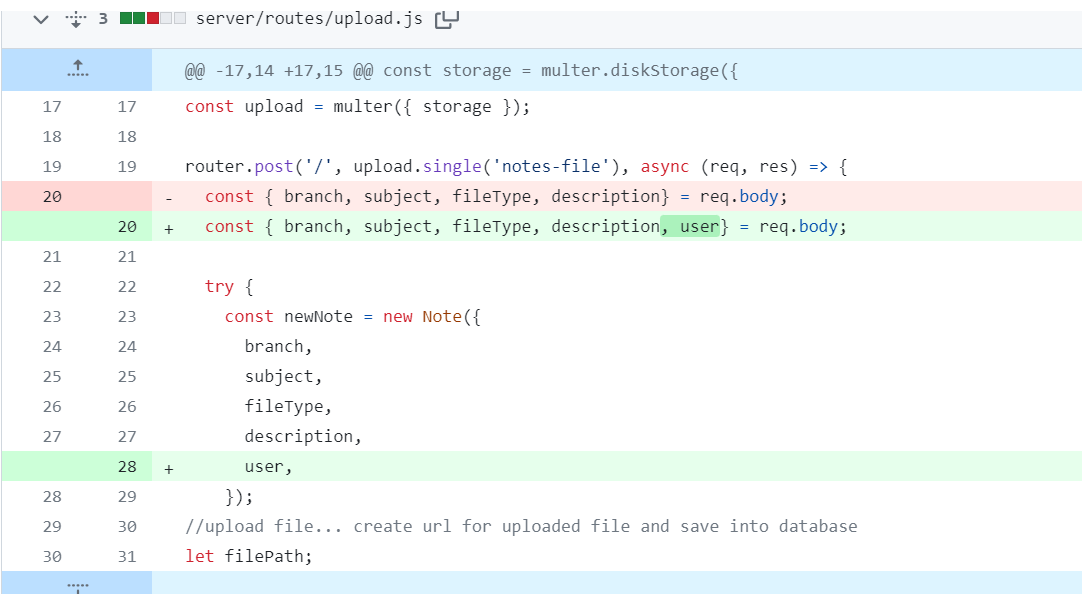


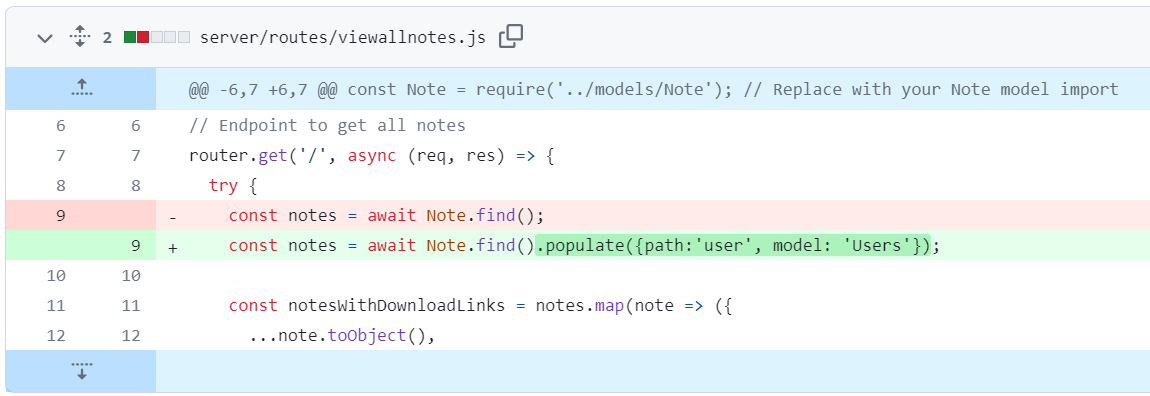


Backend









View user/frontend

import React, { useEffect, useState } from "react";

import AdminNavbar from "../Navbar/adminnavbar";

import { MdDeleteOutline } from "react-icons/md";

export default function ViewUsers() {

  const [users, setUsers] = useState([]);

  useEffect(() => {

    // list of users from backend API

    fetch("http://localhost:5000/viewUsers", {

      method: "GET",

      headers: {

        "Content-Type": "application/json",

      },

    })

      .then((response) => response.json())

      .then((data) => {

        setUsers(data);

      })

      .catch((error) => {

        console.error("Error fetching users:", error);

      });

  }, []);

  // Delete User

  const handleDeleteUser = async (userId) => {

    if (window.confirm("Are you sure to delete this user?")) {

      try {

        //DELETE request to delete the user by ID

        const response = await fetch(

          `http://localhost:5000/viewUsers/${userId}`,

          {

            method: "DELETE",

            headers: {

              "Content-Type": "application/json",

            },

          }

        );

        if (response.status === 200) {

          //refetch the user data from the server or remove the user from the local state.

        } else {

          // if the user deletion fails

          const data = await response.json();

          alert(data.message);

        }

      } catch (error) {

        console.error("Error deleting user:", error);

      }

    }

  };

  return (

    <>

      <AdminNavbar />

      <main className="view-users padding">

        <div className="container1">

          <h2 className="content-heading">View Users</h2>

          <table className="table" id="viewUsers">

            <thead>

              <tr>

                <th className="text-center">S.No</th>

                <th className="">Full Name</th>

                <th className="">Email ID</th>

                {/\* <th className="">Contact</th> \*/}

                {/\* <th className="">Branch</th> \*/}

                <th className="">Role</th>

                <th className="">Action</th>

              </tr>

            </thead>

            <tbody>

              {users.map((user, index) => (

                <tr key={index}>

                  <td className="text-center">{index + 1}</td>

                  <td>{user.name}</td>

                  <td>{user.email}</td>

                  {/\* <td>{user.contact}</td> \*/}

                  {/\* <td>{user.branch}</td> \*/}

                  <td>{user.role}</td>

                  {/\* <td>

                    <button onClick={() => handleDeleteUser(user.\_id)}>

                      Delete

                    </button>

                  </td> \*/}

                  <td>

                    <MdDeleteOutline

                      onClick={() => handleDeleteUser(user.\_id)}

                      className="delete-icon" style={{cursor: 'pointer'}}

                    />

                  </td>

                </tr>

              ))}

            </tbody>

          </table>

        </div>

      </main>

    </>

  );

}