# AlumniConnect\eslint.config.js

import js from '@eslint/js'  
import globals from 'globals'  
import react from 'eslint-plugin-react'  
import reactHooks from 'eslint-plugin-react-hooks'  
import reactRefresh from 'eslint-plugin-react-refresh'  
  
export default [  
 { ignores: ['dist'] },  
 {  
 files: ['\*\*/\*.{js,jsx}'],  
 languageOptions: {  
 ecmaVersion: 2020,  
 globals: globals.browser,  
 parserOptions: {  
 ecmaVersion: 'latest',  
 ecmaFeatures: { jsx: true },  
 sourceType: 'module',  
 },  
 },  
 settings: { react: { version: '18.3' } },  
 plugins: {  
 react,  
 'react-hooks': reactHooks,  
 'react-refresh': reactRefresh,  
 },  
 rules: {  
 ...js.configs.recommended.rules,  
 ...react.configs.recommended.rules,  
 ...react.configs['jsx-runtime'].rules,  
 ...reactHooks.configs.recommended.rules,  
 'react/jsx-no-target-blank': 'off',  
 'react-refresh/only-export-components': [  
 'warn',  
 { allowConstantExport: true },  
 ],  
 },  
 },  
]

# AlumniConnect\index.html

<!doctype html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <link rel="icon" type="image" href="/collegeLogo.jpg" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>Alumni Connect</title>  
 <!-- <link rel="stylesheet" href="/src/css/Header.css"> -->  
 </head>  
 <body>  
 <div id="root"></div>  
 <script type="module" src="/src/main.jsx"></script>  
 </body>  
</html>

# AlumniConnect\src\index.css

\* {  
 margin: 0;  
 padding: 0;  
}  
body{  
 overflow-y: hidden;  
}  
a{  
 text-decoration: none;  
 color: Black;  
}

# AlumniConnect\src\main.jsx

import { StrictMode } from 'react'  
import { createRoot } from 'react-dom/client'  
import './index.css'  
import App from './App.jsx'  
import UserState from './context/user/UserState';  
  
createRoot(document.getElementById('root')).render(  
 // <StrictMode>  
 <UserState>  
 <App />  
 </UserState>  
 // </StrictMode>,  
)

# AlumniConnect\src\App.css

/\*//////////////////////////// main-body ////////////////////////////\*/  
.appCantainer{  
 display: flex;  
 justify-content: flex-start;  
}  
.mainBody{  
 padding: 0px;  
 margin: 0px;  
 width: 100%;  
 overflow:hidden;  
}  
  
.blur {  
 filter: blur(3px); /\* Adjust blur value as needed \*/  
 transition: filter 0.3s ease; /\* Smooth transition \*/  
}

# AlumniConnect\src\App.jsx

import { useState, useEffect, useContext } from 'react'  
import { BrowserRouter, Routes, Route } from "react-router";  
import UserContext from "./context/user/UserContext"  
import './App.css'  
import Header from './components/Header'  
import { Navbar } from './components/allcomponents'  
import MainBody from './components/MainBody'  
import Login from './components/Login'  
import ForgotPass from './components/ForgotPass';  
import VarifyContact from './components/VerifyContact';  
  
function App() {  
 // const [user, setUser] = useState(null)  
 const context = useContext(UserContext)  
 // const { user, setUser, fetchUserData } = context  
 const [mobileView, setMobileView] = useState(window.innerWidth < 750)  
 const [tabView, setTabView] = useState(window.innerWidth < 1075)  
 const [hideMenu, setHideMenu] = useState(window.innerWidth < 1075)  
  
 const handleResize = () => {  
 setMobileView(window.innerWidth < 750);  
 setTabView(window.innerWidth < 1075)  
 setHideMenu(window.innerWidth < 1075)  
 };  
  
 const toggleHideMenu = () => {  
 // const nav = document.querySelector('.nav');  
 // const mainBody = document.querySelector('.main\_body');  
 if (hideMenu) {  
 // Show the nav above the main\_body  
 setHideMenu(false)  
 return true  
 }  
 else {  
 // Hide the nav if it's already visible  
 setHideMenu(true)  
 return false  
 }  
 }  
  
 useEffect(() => {  
 // Add event listener for window resize  
 window.addEventListener('resize', handleResize);  
 // Cleanup event listener on component unmount  
 return () => {  
 window.removeEventListener('resize', handleResize);  
 };  
 }, []);  
  
 return (  
 < BrowserRouter >  
 <Routes>  
 {/\* Default Route to Redirect to /forms \*/}  
 {/\* <Route path="/" element={<Navigate to={`${user === 'student' ? '/forms' : user === 'teacher' ? '/dashboard' : '/login'}`} replace />} /> \*/}  
 {/\* {user ? <Route path="/" element={<Navigate to='/login' replace />} : } \*/}  
 <Route path="/" element={  
 <>  
 <Header mobileView={false} tabView={false} />  
 <Login />  
 </>  
 } />  
  
 <Route path={`/\*`} element={  
 <div className='appCantainer'>  
 <Navbar hideMenu={hideMenu} toggleHideMenu={toggleHideMenu} tabView={tabView} />  
 <div className='mainBody'>  
 <Header mobileView={mobileView} hideMenu={hideMenu} toggleHideMenu={toggleHideMenu} tabView={tabView} />  
 <MainBody hideMenu={hideMenu} tabView={tabView} />  
 </div>  
 </div>  
 } />  
 <Route path="/forgotpass" element={  
 <>  
 <Header mobileView={false} tabView={false} />  
 <ForgotPass />  
 </>  
 } />  
 <Route path="/varifycontact" element={  
 <>  
 <Header mobileView={false} tabView={false} />  
 <VarifyContact />  
 </>  
 } />  
  
 </Routes>  
 </BrowserRouter >  
 );  
}  
  
export default App

# AlumniConnect\src\components\AddBulkStudent.jsx

import React from 'react';  
  
const AddBulkStudent = ({ onFileUpload }) => {  
 return (  
 <div className="bulk-upload-form">  
 <h2>Bulk Upload Students</h2>  
 <div className="file-upload-container">  
 <input  
 type="file"  
 id="jsonFile"  
 accept=".json"  
 onChange={onFileUpload}  
 className="file-input"  
 />  
 <label htmlFor="jsonFile" className="file-label">  
 Choose JSON File  
 </label>  
 <p className="file-help">Upload a JSON file containing an array of student objects</p>  
 <div className="json-format-help">  
 <h3>Expected JSON Format:</h3>  
 <pre>  
 {`[  
 {  
 "firstName": "John",  
 "fatherName": "Doe",  
 "lastName": "Smith",  
 "motherName": "Jane",  
 "grandfatherName": "James",  
 "prn": "1234567890",  
 "moodleid": "12345678",  
 "mobileno": "1234567890",  
 "email": "john@example.com",  
 "address": "123 Main St",  
 "placeOfBirth": "Mumbai",  
 "dateOfBirth": "2000-01-01",  
 "branch": "COMPUTER SCIENCE",  
 "program": "B.Tech",  
 "semester": "8",  
 "yearOfJoining": "2021",  
 "dateOfAdmission": "2021-08-01",  
 "lastCollegeAttended": "Previous College",  
 "religion": "Hindu",  
 "caste": "General",  
 "category": "General",  
 "nationality": "Indian"  
 }  
]`}  
 </pre>  
 </div>  
 </div>  
 </div>  
 );  
};  
  
export default AddBulkStudent;

# AlumniConnect\src\components\AddNotice.jsx

import React, {useContext} from 'react'  
import { useForm } from 'react-hook-form';  
import UserContext from "../context/user/UserContext";  
  
  
const AddNotice = () => {  
 const { register, handleSubmit, watch, formState: { errors } } = useForm();  
 const { host, user, token } = useContext(UserContext);  
  
  
 const onSubmit = async (formData)=>{  
 try {  
 const response = await fetch('http://localhost:5000/api/notification/notifyall', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 },  
 body: JSON.stringify({  
 title: formData.noticeTitle,  
 message: formData.notice  
 })  
 });  
  
 const json = await response.json();  
   
 if (json.success) {  
 alert("Notice added successfully!");  
 // Reset form fields  
 formData.noticeTitle = '';  
 formData.notice = '';  
 } else {  
 alert("Error adding notice: " + json.error);  
 }  
 } catch (error) {  
 console.error("Error:", error);  
 alert("An error occurred while adding the notice");  
 }  
 }  
  
 return (  
 <div className="form-container">  
 <div>  
 <h1 className="form-header">Add Notice</h1>  
 </div>  
 <br />  
 <div className='addNoticeContainer'>  
 <form className='addnotice' onSubmit={handleSubmit(onSubmit)}>  
 <input className="noticeTitle" type="text" placeholder="Add Notice Title" {...register("noticeTitle", { minLength: { value: 5, message: 'Notice title should be atleast 5 characters' } })} />  
 <hr />  
 <br />  
 <textarea className="noticeContaint" rows="4" placeholder="Add Notice" {...register("notice", { minLength: { value: 10, message: 'Notice should be atleast 10 characters' } })} />  
 <div className="form-input">  
 <button type="submit" className="submit-button">Add Notice</button>  
 </div>  
 </form>  
 </div>  
 </div>  
 )  
}  
  
export default AddNotice

# AlumniConnect\src\components\AddSingleStudent.jsx

import React from 'react';  
import { useForm } from 'react-hook-form';  
  
const AddSingleStudent = ({ onSubmit, loading }) => {  
 const {  
 register,  
 handleSubmit,  
 formState: { errors }  
 } = useForm({  
 defaultValues: {  
 firstName: '',  
 fatherName: '',  
 lastName: '',  
 motherName: '',  
 grandfatherName: '',  
 prn: '',  
 moodleid: '',  
 mobileno: '',  
 email: '',  
 address: '',  
 placeOfBirth: '',  
 dateOfBirth: '',  
 branch: '',  
 program: '',  
 semester: '',  
 yearOfJoining: '',  
 dateOfAdmission: '',  
 lastCollegeAttended: '',  
 religion: '',  
 caste: '',  
 category: '',  
 nationality: '',  
 password: 'password'  
 }  
 });  
  
 return (  
 <div className="single-student-form">  
 <h2>Add Single Student</h2>  
 <form onSubmit={handleSubmit(onSubmit)}>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="First Name"  
 {...register("firstName", { required: "First name is required" })}  
 />  
 {errors.firstName && <span className="error-text">{errors.firstName.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Father's Name"  
 {...register("fatherName", { required: "Father's name is required" })}  
 />  
 {errors.fatherName && <span className="error-text">{errors.fatherName.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Last Name"  
 {...register("lastName", { required: "Last name is required" })}  
 />  
 {errors.lastName && <span className="error-text">{errors.lastName.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Mother's Name"  
 {...register("motherName", { required: "Mother's name is required" })}  
 />  
 {errors.motherName && <span className="error-text">{errors.motherName.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Grandfather's Name"  
 {...register("grandfatherName", { required: "Grandfather's name is required" })}  
 />  
 {errors.grandfatherName && <span className="error-text">{errors.grandfatherName.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="PRN"  
 {...register("prn", {   
 required: "PRN is required",  
 pattern: {  
 value: /^\d{10,15}$/,  
 message: "PRN must be 10-15 digits"  
 }  
 })}  
 />  
 {errors.prn && <span className="error-text">{errors.prn.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Moodle ID"  
 {...register("moodleid", {   
 required: "Moodle ID is required",  
 pattern: {  
 value: /^\d{8,10}$/,  
 message: "Moodle ID must be 8-10 digits"  
 }  
 })}  
 />  
 {errors.moodleid && <span className="error-text">{errors.moodleid.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Mobile Number"  
 {...register("mobileno", {   
 required: "Mobile number is required",  
 pattern: {  
 value: /^\d{10}$/,  
 message: "Mobile number must be 10 digits"  
 }  
 })}  
 />  
 {errors.mobileno && <span className="error-text">{errors.mobileno.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="email"  
 placeholder="Email"  
 {...register("email", {   
 required: "Email is required",  
 pattern: {  
 value: /^[A-Z0-9.\_%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i,  
 message: "Invalid email address"  
 }  
 })}  
 />  
 {errors.email && <span className="error-text">{errors.email.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Address"  
 {...register("address", { required: "Address is required" })}  
 />  
 {errors.address && <span className="error-text">{errors.address.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Place of Birth"  
 {...register("placeOfBirth", { required: "Place of birth is required" })}  
 />  
 {errors.placeOfBirth && <span className="error-text">{errors.placeOfBirth.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="date"  
 {...register("dateOfBirth", { required: "Date of birth is required" })}  
 />  
 {errors.dateOfBirth && <span className="error-text">{errors.dateOfBirth.message}</span>}  
 </div>  
 <div className="form-group">  
 <select  
 {...register("branch", { required: "Branch is required" })}  
 >  
 <option value="">Select Branch</option>  
 <option value="COMPUTER SCIENCE">Computer Science</option>  
 <option value="INFORMATION TECHNOLOGY">Information Technology</option>  
 <option value="CSE-DATA SCIENCE">CSE-Data Science</option>  
 <option value="CSE-AIML">CSE-AIML</option>  
 <option value="MECHANICAL">Mechanical</option>  
 <option value="CIVIL">Civil</option>  
 </select>  
 {errors.branch && <span className="error-text">{errors.branch.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Program"  
 {...register("program", { required: "Program is required" })}  
 />  
 {errors.program && <span className="error-text">{errors.program.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Semester"  
 {...register("semester", {   
 required: "Semester is required",  
 pattern: {  
 value: /^[1-8]$/,  
 message: "Semester must be between 1 and 8"  
 }  
 })}  
 />  
 {errors.semester && <span className="error-text">{errors.semester.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Year of Joining"  
 {...register("yearOfJoining", { required: "Year of joining is required" })}  
 />  
 {errors.yearOfJoining && <span className="error-text">{errors.yearOfJoining.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="date"  
 {...register("dateOfAdmission", { required: "Date of admission is required" })}  
 />  
 {errors.dateOfAdmission && <span className="error-text">{errors.dateOfAdmission.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Last College Attended"  
 {...register("lastCollegeAttended", { required: "Last college attended is required" })}  
 />  
 {errors.lastCollegeAttended && <span className="error-text">{errors.lastCollegeAttended.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Religion"  
 {...register("religion", { required: "Religion is required" })}  
 />  
 {errors.religion && <span className="error-text">{errors.religion.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Caste"  
 {...register("caste", { required: "Caste is required" })}  
 />  
 {errors.caste && <span className="error-text">{errors.caste.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Category"  
 {...register("category", { required: "Category is required" })}  
 />  
 {errors.category && <span className="error-text">{errors.category.message}</span>}  
 </div>  
 <div className="form-group">  
 <input  
 type="text"  
 placeholder="Nationality"  
 {...register("nationality", { required: "Nationality is required" })}  
 />  
 {errors.nationality && <span className="error-text">{errors.nationality.message}</span>}  
 </div>  
 <button type="submit" className="submit-btn" disabled={loading}>  
 {loading ? 'Adding...' : 'Add Student'}  
 </button>  
 </form>  
 </div>  
 );  
};  
  
export default AddSingleStudent;

# AlumniConnect\src\components\AddStudent.jsx

import React, { useState, useContext, useEffect } from 'react';  
import { useNavigate } from 'react-router-dom';  
import UserContext from '../context/user/UserContext';  
import AddSingleStudent from './AddSingleStudent';  
import AddBulkStudent from './AddBulkStudent';  
import '../css/AddStudent.css';  
  
const AddStudent = () => {  
 const context = useContext(UserContext);  
 const { host, token, user } = context;  
 const navigate = useNavigate();  
 const [loading, setLoading] = useState(false);  
 const [error, setError] = useState('');  
 const [success, setSuccess] = useState('');  
 const [activeTab, setActiveTab] = useState('single');  
  
 const handleSingleSubmit = async (formData) => {  
 try {  
 setLoading(true);  
 setError('');  
 setSuccess('');  
  
 const response = await fetch(`${host}auth/student/createuser`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 },  
 body: JSON.stringify(formData)  
 });  
  
 const data = await response.json();  
  
 if (!response.ok) {  
 throw new Error(data.error || 'Failed to create student');  
 }  
 alert('Student added successfully!');  
 setSuccess('Student added successfully!');  
 } catch (err) {  
 setError(err.message);  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 const handleBulkUpload = async (event) => {  
 try {  
 setLoading(true);  
 setError('');  
 setSuccess('');  
  
 const file = event.target.files[0];  
 if (!file) return;  
  
 const reader = new FileReader();  
 reader.onload = async (e) => {  
 try {  
 const students = JSON.parse(e.target.result);  
 if (!Array.isArray(students)) {  
 throw new Error('File must contain an array of student objects');  
 }  
  
 const response = await fetch(`${host}auth/student/bulk-create`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 },  
 body: JSON.stringify(students)  
 });  
  
 const data = await response.json();  
  
 if (!response.ok) {  
 throw new Error(data.error || 'Failed to create students');  
 }  
  
 setSuccess(`Successfully added ${students.length} students!`);  
 } catch (err) {  
 setError(err.message);  
 } finally {  
 setLoading(false);  
 }  
 };  
  
 reader.readAsText(file);  
 } catch (err) {  
 setError('Error reading file');  
 setLoading(false);  
 }  
 };  
  
 useEffect(() => {  
 if (!user || user !== 'teacher') {  
 navigate('/');  
 }  
 }, [user, navigate]);  
  
 return (  
 <div className="form-container">  
 <div className='checkastatusHeader'>  
 <h1 className="form-header">Add Student</h1>  
 <div className="tab-buttons">  
 <button  
 className={`tab-button ${activeTab === 'single' ? 'active' : ''}`}  
 onClick={() => setActiveTab('single')}  
 >  
 Add Single Student  
 </button>  
 <button  
 className={`tab-button ${activeTab === 'bulk' ? 'active' : ''}`}  
 onClick={() => setActiveTab('bulk')}  
 >  
 Bulk Upload  
 </button>  
 </div>  
 </div>  
 <br />  
 <hr />  
 <br />  
  
 {error && <div className="error-message">{error}</div>}  
 {success && <div className="success-message">{success}</div>}  
  
 {activeTab === 'single' ? (  
 <AddSingleStudent onSubmit={handleSingleSubmit} loading={loading} />  
 ) : (  
 <AddBulkStudent onFileUpload={handleBulkUpload} />  
 )}  
 </div>  
 );  
};  
  
export default AddStudent;

# AlumniConnect\src\components\Application.jsx

import React, { useState, useContext, useEffect } from 'react'  
import '../css/Application.css'  
import UserContext from "../context/user/UserContext"  
  
const getApplicationStates = (formData) => {  
 return [  
 {   
 id: 1,   
 name: "Submitted",   
 status: formData.submitted ? "completed" : "pending",  
 date: formData.submittedDate,  
 remarks: formData.submittedRemarks || "Document submitted by student"  
 },  
 {   
 id: 2,   
 name: "Admin Verification",   
 status: formData.admin ? "completed" : "pending",  
 date: formData.adminDate,  
 remarks: formData.adminRemarks || "Verification pending at Admin",  
 checks: [  
 {  
 name: "Fees Paid",   
 status: formData.adminFeesPaid ? "✓" : "✗",  
 showUpload: !formData.adminFeesPaid,  
 uploadType: "receipt",  
 uploadedFile: formData.feesReciept  
 },  
 {  
 name: "Document Verified",   
 status: formData.documentVerified ? "✓" : "✗",  
 showUpload: !formData.documentVerified,  
 uploadType: "document",  
 uploadedFile: formData.varificationDoc  
 }  
 ]  
 },  
 {   
 id: 3,   
 name: "Exam Cell Verification",   
 status: formData.examCell ? "completed" : "pending",  
 date: formData.examCellDate,  
 remarks: formData.examCellRemarks || "Exam cell verification pending",  
 checks: [  
 !formData.examCellFeesPaid && {  
 name: "Fees Paid",   
 status: "✗",  
 showUpload: !formData.examCellFeesPaid,  
 uploadType: "receipt",  
 uploadedFile: formData.feesReciept  
 }  
 ].filter(Boolean)  
 },  
 {   
 id: 4,   
 name: "Library Verification",   
 status: formData.library ? "completed" : "pending",  
 date: formData.libraryDate,  
 remarks: formData.libraryRemarks || "Library verification pending",  
 checks: [  
 !formData.libraryFeesPaid && {  
 name: "Fees Paid",   
 status: "✗",  
 showUpload: !formData.libraryFeesPaid || !formData.library,  
 uploadType: "receipt",  
 uploadedFile: formData.feesReciept  
 }  
 ].filter(Boolean)  
 },  
 {   
 id: 5,   
 name: "Branch Verification",   
 status: formData.atbranch ? "completed" : "pending",  
 date: formData.branchDate,  
 remarks: formData.branchRemarks || "Branch verification pending",  
 checks: [  
 !formData.uploadDocument && {  
 name: "Upload Document",   
 status: "✗",  
 showUpload: formData.library,  
 uploadType: "document",  
 uploadedFile: formData.varificationDoc  
 }  
 ].filter(Boolean)  
 },  
 {   
 id: 6,   
 name: "Ready to Collect",   
 status: formData.collected ? "completed" : "pending",  
 date: formData.collectionDate,  
 remarks: formData.collectionRemarks || "Document ready for collection"  
 }  
 ];  
};  
  
const Application = (props) => {  
 const [isExpanded, setIsExpanded] = useState(false);  
 const context = useContext(UserContext)  
 const { host, token } = context  
 const [applicationStates, setApplicationStates] = useState([]);  
 const [lcData, setLcData] = useState(null);  
  
 useEffect(() => {  
 const fetchLcData = async () => {  
 try {  
 const response = await fetch(`${host}forms/lc/${props.formData.docId}`, {  
 headers: {  
 'auth-token': token  
 }  
 });  
 const data = await response.json();  
 if (data.success) {  
 setLcData(data.doc);  
 }  
 } catch (error) {  
 console.error('Error fetching LC data:', error);  
 }  
 };  
  
 if (props.formData.docId) {  
 fetchLcData();  
 }  
 }, [props.formData.docId, host, token]);  
  
 useEffect(() => {  
 if (lcData) {  
 const states = getApplicationStates({  
 ...props.formData,  
 feesReciept: lcData.feesReciept,  
 varificationDoc: lcData.varificationDoc  
 });  
 setApplicationStates(states);  
 }  
 }, [props.formData, lcData]);  
  
 const toggleExpand = () => {  
 setIsExpanded(!isExpanded);  
 };  
  
 const handleClick = (e) => {  
 e.preventDefault();  
 toggleExpand();  
 };  
  
 const handleCancel = async (e) => {  
 e.preventDefault();  
 if (window.confirm('Are you sure you want to cancel this form?')) {  
 try {  
 const response = await fetch(`${host}forms/cancel/${props.formData.\_id}`, {  
 method: "PUT",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 }  
 });  
   
 if (!response.ok) {  
 throw new Error('Failed to cancel form');  
 }  
   
 alert('Form cancelled successfully');  
 window.location.reload();  
 } catch (error) {  
 console.error("Error cancelling form:", error);  
 alert('Failed to cancel form');  
 }  
 }  
 };  
  
 const handleFileUpload = async (e, stepId, checkName, uploadType) => {  
 e.preventDefault();  
 const file = e.target.files[0];  
 if (!file) return;  
  
 if (file.type !== 'application/pdf') {  
 alert('Please upload a PDF file');  
 return;  
 }  
  
 try {  
 const formData = new FormData();  
 formData.append(uploadType === 'document' ? 'verification-document' : 'fees-reciept', file);  
  
 const response = await fetch(`${host}files/upload-${uploadType === 'document' ? 'verification-document' : 'fees-reciept'}/${props.formData.\_id}`, {  
 method: 'POST',  
 headers: {  
 'auth-token': token,  
 'location': uploadType === 'document' ? 'verification-document' : 'fees-reciept'  
 },  
 body: formData  
 });  
  
 const data = await response.json();  
 if (data.success) {  
 // Refresh the LC data to get updated file names  
 const lcResponse = await fetch(`${host}forms/lc/${props.formData.docId}`, {  
 headers: {  
 'auth-token': token  
 }  
 });  
 const lcData = await lcResponse.json();  
 if (lcData.success) {  
 setLcData(lcData.doc);  
 }  
 alert(`${checkName} uploaded successfully!`);  
 } else {  
 alert(`Failed to upload ${checkName}: ${data.error}`);  
 }  
 } catch (error) {  
 console.error(`Error uploading ${checkName}:`, error);  
 alert(`Error uploading ${checkName}`);  
 }  
 };  
  
 const handleFileClick = async (e, fileType, formId) => {  
 e.preventDefault();  
 try {  
 const response = await fetch(`${host}files/get-${fileType}/${formId}`, {  
 headers: {  
 'auth-token': token  
 }  
 });  
   
 if (response.ok) {  
 const blob = await response.blob();  
 const url = window.URL.createObjectURL(blob);  
 window.open(url, '\_blank');  
 } else {  
 const errorData = await response.json();  
 alert(`Error: ${errorData.error}`);  
 }  
 } catch (error) {  
 console.error('Error opening file:', error);  
 alert('Error opening file');  
 }  
 };  
  
 const formatDate = (dateString) => {  
 if (!dateString) return '';  
 const date = new Date(dateString);  
 return date.toLocaleDateString('en-GB', {  
 day: '2-digit',  
 month: '2-digit',  
 year: 'numeric'  
 });  
 };  
  
 return (  
 <div className="appln-wrapper">  
 <div className={`appln ${isExpanded ? 'expanded' : ''}`} onClick={handleClick}>  
 <div className="app\_name">  
 <span>{props.fname}</span>  
 </div>  
 <div className="status">{props.fstatus}</div>  
 <div className="date">{props.fdate}</div>  
 </div>  
 <div className={`tracking-container ${isExpanded ? 'expanded' : ''}`}>  
 <div className="tracking-timeline">  
 {applicationStates.map((state, index) => (  
 <div key={state.id} className="tracking-step">  
 <div className={`step-icon ${state.status}`}>  
 {state.status === 'completed' ? '✓' : '○'}  
 </div>  
 <div className="step-content">  
 <div className="step-title">{state.name}</div>  
 <div className="step-status">{state.status}</div>  
 {state.date && (  
 <div className="step-date">Date: {formatDate(state.date)}</div>  
 )}  
 <div className="step-remarks">  
 <div className="remarks-label">Remarks:</div>  
 <div className="remarks-text">{state.remarks}</div>  
 </div>  
 {state.checks && state.checks.length > 0 && (  
 <div className="step-checks">  
 {state.checks.map((check, checkIndex) => (  
 <div key={checkIndex} className="check-item">  
 <span className={`check-status ${check.status === '✓' ? 'completed' : 'pending'}`}>  
 {check.status}  
 </span>  
 <span className="check-name">{check.name}</span>  
 {check.showUpload && (  
 <div className="receipt-upload">  
 {check.uploadedFile && (  
 <span   
 className="uploaded-file"  
 onClick={(e) => handleFileClick(e, check.uploadType === 'document' ? 'verification-document' : 'fees-reciept', props.formData.\_id)}  
 >  
 {check.uploadedFile}  
 </span>  
 )}  
 <input  
 type="file"  
 accept=".pdf"  
 onChange={(e) => handleFileUpload(e, state.id, check.name, check.uploadType)}  
 className="receipt-input"  
 id={`${state.id}-${checkIndex}-upload`}  
 />  
 <label   
 htmlFor={`${state.id}-${checkIndex}-upload`}   
 className={`upload-label ${check.uploadType === 'document' ? 'document-upload' : ''}`}  
 >  
 Upload {check.uploadType === 'document' ? 'Document' : 'Receipt'}  
 </label>  
 </div>  
 )}  
 </div>  
 ))}  
 </div>  
 )}  
 </div>  
 {index < applicationStates.length - 1 && (  
 <div className={`step-connector ${state.status}`}></div>  
 )}  
 </div>  
 ))}  
 {!props.formData.documentVerified && props.formData.status !== 'Cancelled' && (  
 <div className="cancel-form-section">  
 <button   
 className="cancel-button"  
 onClick={handleCancel}  
 >  
 Cancel Form  
 </button>  
 </div>  
 )}  
 </div>  
 </div>  
 </div>  
 )  
}  
  
export default Application

# AlumniConnect\src\components\CheckStatus.jsx

import React, { useContext } from 'react'  
import UserContext from "../context/user/UserContext"  
import StudentCheckStatus from './StudentCheckStatus'  
import TeacherCheckStatus from './TeacherCheckStatus'  
  
const CheckStatus = () => {  
 const context = useContext(UserContext)  
 const { user } = context  
  
 // Check if user ID starts with 't' for teacher or 's' for student  
 const isTeacher = user && user.startsWith('t')  
 const isStudent = user && user.startsWith('s')  
  
 return (  
 <>  
 {isStudent ? (  
 <StudentCheckStatus />  
 ) : isTeacher ? (  
 <TeacherCheckStatus />  
 ) : (  
 <div>Access Denied</div>  
 )}  
 </>  
 )  
}  
  
export default CheckStatus

# AlumniConnect\src\components\Convocation.jsx

import React, { useState } from 'react';  
// import '../css/Lor.css';  
  
const Convocation = () => {  
 // Generate an array of years (for Year of Admission/Leaving)  
 const currentYear = new Date().getFullYear();  
 const startYear = 2000; // Adjust as needed for your institution  
 const years = [];  
 for (let year = startYear; year <= currentYear; year++) {  
 years.push(year);  
 }  
  
 return (  
 <div className="form-container">  
 <h1 className="form-header">Convocation Form</h1>  
 <br />  
 <hr />  
 <form className="ConvocationForm">  
 {/\* Student Personal Information \*/}  
 <div className="form-section">  
 <br />  
 <div className="fieldContainer">  
 <div className="innerfieldContainer">  
 <div className="form-input name">  
 <label htmlFor="name">Name of the Student</label>  
 <div className="nameContainer">  
 <input type="text" id="FirstName" name="FirstName" placeholder="First Name" required />  
 <input type="text" id="MiddleName" name="MiddleName" placeholder="Middle Name" required />  
 <input type="text" id="LastName" name="LastName" placeholder="Last Name" required />  
 <input type="text" id="MotherName" name="MotherName" placeholder="Mother's Name" required />  
 </div>  
 </div>  
 </div>  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="prn">PRN number</label>  
 <input type="text" id="prn" name="prn" placeholder="Enter your PRN" className="no-arrows" required />  
 </div>  
 <div className="form-input">  
 <label htmlFor="moodleid">Moodle ID</label>  
 <input type="text" id="moodleid" name="moodleid" placeholder="Enter your Moodle ID" className="no-arrows" required />  
 </div>  
 <div className="form-input">  
 <label htmlFor="dob">Date of Birth</label>  
 <input type="date" id="dob" name="dob" style={{ display: 'block' }} className="no-arrows" required />  
 </div>  
 </div>  
 </div>  
 {/\* Academic Details \*/}  
 <div className="form-section">  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="Program">Program</label>  
 <select id="Program" name="Program" required>  
 <option value="">Select Program</option>  
 <option value="BE">BE</option>  
 </select>  
 </div>  
 <div className="form-input">  
 <label htmlFor="Course">Course</label>  
 <select id="course" name="course" required>  
 <option value="">Select Course</option>  
 <option value="Computer Science">COMPUTER</option>  
 <option value="Information Technology">INFORMATION TECHNOLOGY</option>  
 <option value="Information Technology">CSE-DATA SCIENCE</option>  
 <option value="Information Technology">CSE-AIML</option>  
 <option value="Mechanical Engineering">MECHANICAL</option>  
 <option value="Civil Engineering">CIVIL</option>  
 </select>  
 </div>  
 </div>  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="year">Date of Admission</label>  
 <select type="number" id="year" name="admissionYear" required >  
 <option key="selectyear" value="SelectYear">Select Year</option>{years.map((year) => (  
 <option key={year} value={year}>{year}</option>))}  
 </select>  
 </div>  
 <div className="form-input">  
 <label htmlFor="yearJoined">Year of Joining</label>  
 <select id="yearJoined" name="yearJoined" required>  
 <option value="">Select Year</option>  
 <option value="First Year">First Year</option>  
 <option value="Direct Second Year">Direct Second Year</option>  
 </select>  
 </div>  
 </div>  
 <div className="form-input">  
 <label htmlFor="year">Year of Convovation</label>  
 <select type="number" id="year" name="convocationYear" required >  
 <option key="selectyear" value="SelectYear">Select Year</option>{years.map((year) => (  
 <option key={year} value={year}>{year}</option>))}  
 </select>  
 </div>  
 </div>  
 </div >  
 <hr />  
 <br />  
 {/\* Form Submission \*/}  
 <div className="form-input button">  
 <button type="submit" className="submit-button">Submit Request</button>  
 </div>  
 </form >  
 </div >  
 );  
};  
  
export default Convocation;

# AlumniConnect\src\components\Dashboard.jsx

import React, { useState, useEffect, useContext } from 'react';

import {

    Box,

    Card,

    CardContent,

    Typography,

    Grid,

    CircularProgress,

    Paper,

    Table,

    TableBody,

    TableCell,

    TableContainer,

    TableHead,

    TableRow,

    Alert,

} from '@mui/material';

import { useNavigate } from 'react-router-dom';

import UserContext from '../context/user/UserContext';

import {

    BarChart,

    Bar,

    XAxis,

    YAxis,

    CartesianGrid,

    Tooltip,

    Legend,

    ResponsiveContainer,

    PieChart,

    Pie,

    Cell

} from 'recharts';

const COLORS = ['#0088FE', '#00C49F', '#FFBB28', '#FF8042', '#8884D8', '#82CA9D'];

const Dashboard = () => {

    const [stats, setStats] = useState([]);

    const [loading, setLoading] = useState(true);

    const [error, setError] = useState(null);

    const navigate = useNavigate();

    const { host, token } = useContext(UserContext);

    useEffect(() => {

        const fetchStats = async () => {

            try {

                if (!token) {

                    navigate('/login');

                    return;

                }

                const response = await fetch(`${host}dashboard/stats`, {

                    headers: {

                        'auth-token': token,

                        'Content-Type': 'application/json'

                    }

                });

                if (response.status === 401) {

                    navigate('/login');

                    return;

                }

                if (response.status === 403) {

                    setError('Access denied. Teachers only.');

                    setLoading(false);

                    return;

                }

                if (response.status === 404) {

                    setError('Teacher not found');

                    setLoading(false);

                    return;

                }

                if (!response.ok) {

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

                }

                const data = await response.json();

                console.log('API Response:', data);

                setStats(data);

                setLoading(false);

            } catch (err) {

                console.error('Error fetching stats:', err);

                setError(err.message);

                setLoading(false);

            }

        };

        fetchStats();

    }, [navigate, host, token]);

    const getTotalStats = () => {

        return stats.reduce((*acc*, *curr*) => ({

            total: acc.total + curr.total,

            submitted: acc.submitted + curr.submitted,

            adminVerified: acc.adminVerified + curr.adminVerified,

            examCellVerified: acc.examCellVerified + curr.examCellVerified,

            libraryVerified: acc.libraryVerified + curr.libraryVerified,

            branchVerified: acc.branchVerified + curr.branchVerified,

            collected: acc.collected + curr.collected,

        }), {

            total: 0,

            submitted: 0,

            adminVerified: 0,

            examCellVerified: 0,

            libraryVerified: 0,

            branchVerified: 0,

            collected: 0

        });

    };

    const getStatusData = () => {

        const totalStats = getTotalStats();

        return [

            { name: 'Submitted', value: totalStats.submitted },

            { name: 'Admin Verified', value: totalStats.adminVerified },

            { name: 'Exam Cell Verified', value: totalStats.examCellVerified },

            { name: 'Library Verified', value: totalStats.libraryVerified },

            { name: 'Branch Verified', value: totalStats.branchVerified },

            { name: 'Collected', value: totalStats.collected }

        ].filter(*item* => item.value > 0);

    };

    if (loading) {

        return (

            <Box *display*="flex" *justifyContent*="center" *alignItems*="center" *minHeight*="80vh">

                <CircularProgress />

            </Box>

        );

    }

    if (error) {

        return (

            <Box *p*={3}>

                <Alert *severity*="error">{error}</Alert>

            </Box>

        );

    }

    const totalStats = getTotalStats();

    const statusData = getStatusData();

    return (

        <Box *p*={3} *className*="form-container">

            <Typography *variant*="h4" *gutterBottom*>

                Document Statistics Dashboard

            </Typography>

            {*/\* Summary Cards \*/*}

            <Grid *container* *spacing*={3} *sx*={{ mb: 4 }}>

                <Grid *item* *xs*={12} *sm*={6} *md*={3}>

                    <Card *sx*={{ bgcolor: '#f5f5f5' }}>

                        <CardContent>

                            <Typography *color*="textSecondary" *gutterBottom*>

                                Total Documents

                            </Typography>

                            <Typography *variant*="h4">

                                {totalStats.total}

                            </Typography>

                        </CardContent>

                    </Card>

                </Grid>

                <Grid *item* *xs*={12} *sm*={6} *md*={3}>

                    <Card *sx*={{ bgcolor: '#e3f2fd' }}>

                        <CardContent>

                            <Typography *color*="textSecondary" *gutterBottom*>

                                Submitted

                            </Typography>

                            <Typography *variant*="h4">

                                {totalStats.submitted}

                            </Typography>

                        </CardContent>

                    </Card>

                </Grid>

                <Grid *item* *xs*={12} *sm*={6} *md*={3}>

                    <Card *sx*={{ bgcolor: '#e8f5e9' }}>

                        <CardContent>

                            <Typography *color*="textSecondary" *gutterBottom*>

                                Admin Verified

                            </Typography>

                            <Typography *variant*="h4">

                                {totalStats.adminVerified}

                            </Typography>

                        </CardContent>

                    </Card>

                </Grid>

                <Grid *item* *xs*={12} *sm*={6} *md*={3}>

                    <Card *sx*={{ bgcolor: '#fff3e0' }}>

                        <CardContent>

                            <Typography *color*="textSecondary" *gutterBottom*>

                                Exam Cell Verified

                            </Typography>

                            <Typography *variant*="h4">

                                {totalStats.examCellVerified}

                            </Typography>

                        </CardContent>

                    </Card>

                </Grid>

                <Grid *item* *xs*={12} *sm*={6} *md*={3}>

                    <Card *sx*={{ bgcolor: '#fce4ec' }}>

                        <CardContent>

                            <Typography *color*="textSecondary" *gutterBottom*>

                                Library Verified

                            </Typography>

                            <Typography *variant*="h4">

                                {totalStats.libraryVerified}

                            </Typography>

                        </CardContent>

                    </Card>

                </Grid>

                <Grid *item* *xs*={12} *sm*={6} *md*={3}>

                    <Card *sx*={{ bgcolor: '#f3e5f5' }}>

                        <CardContent>

                            <Typography *color*="textSecondary" *gutterBottom*>

                                Branch Verified

                            </Typography>

                            <Typography *variant*="h4">

                                {totalStats.branchVerified}

                            </Typography>

                        </CardContent>

                    </Card>

                </Grid>

                <Grid *item* *xs*={12} *sm*={6} *md*={3}>

                    <Card *sx*={{ bgcolor: '#e0f7fa' }}>

                        <CardContent>

                            <Typography *color*="textSecondary" *gutterBottom*>

                                Collected

                            </Typography>

                            <Typography *variant*="h4">

                                {totalStats.collected}

                            </Typography>

                        </CardContent>

                    </Card>

                </Grid>

            </Grid>

            {*/\* Charts \*/*}

            <Grid *container* *spacing*={3} *sx*={{ mb: 4 }}>

                <Grid *item* *xs*={12} *md*={8}>

                    <Paper *sx*={{ p: 2 }}>

                        <Typography *variant*="h6" *gutterBottom*>

                            Department-wise Distribution

                        </Typography>

                        <Box *height*={400} *sx*={{ width: '100%', maxWidth: '800px', margin: '0 auto' }}>

                            <ResponsiveContainer *width*="100%" *height*="100%">

                                <BarChart

*data*={stats}

*margin*={{ top: 20, right: 30, left: 20, bottom: 5 }}

*barSize*={30}

                                >

                                    <CartesianGrid *strokeDasharray*="3 3" />

                                    <XAxis *dataKey*="\_id" />

                                    <YAxis />

                                    <Tooltip />

                                    <Legend />

                                    <Bar *dataKey*="total" *fill*="#8884d8" *name*="Total Documents" />

                                    <Bar *dataKey*="adminVerified" *fill*="#82ca9d" *name*="Admin Verified" />

                                </BarChart>

                            </ResponsiveContainer>

                        </Box>

                    </Paper>

                </Grid>

                <Grid *item* *xs*={12} *md*={4}>

                    <Paper *sx*={{ p: 2 }}>

                        <Typography *variant*="h6" *gutterBottom*>

                            Document Status Distribution

                        </Typography>

                        <Box *height*={400} *sx*={{ width: '100%', maxWidth: '900px', margin: '0 auto', padding: '20px' }}>

                            <ResponsiveContainer *width*="100%" *height*="100%">

                                <PieChart *margin*={{ top: 20, right: 20, left: 20, bottom: 20 }}>

                                    <Pie

*data*={statusData}

*cx*="50%"

*cy*="50%"

*labelLine*={false}

*label*={({ *name*, *percent* }) => {

                                            if (percent > 0.05) {

                                                return `${name}: ${(percent \* 100).toFixed(0)}%`;

                                            }

                                            return null;

                                        }}

*outerRadius*={120}

*fill*="#8884d8"

*dataKey*="value"

                                    >

                                        {statusData.map((*entry*, *index*) => (

                                            <Cell *key*={`cell-${index}`} *fill*={COLORS[index % COLORS.length]} />

                                        ))}

                                    </Pie>

                                    <Tooltip

*formatter*={(*value*, *name*) => [`${value} documents`, name]}

                                    />

                                    <Legend *layout*="vertical" *align*="right" *verticalAlign*="middle" />

                                </PieChart>

                            </ResponsiveContainer>

                        </Box>

                    </Paper>

                </Grid>

            </Grid>

            {*/\* Department-wise Breakdown \*/*}

            <Typography *variant*="h5" *gutterBottom*>

                Department-wise Statistics

            </Typography>

            <TableContainer *component*={Paper}>

                <Table>

                    <TableHead>

                        <TableRow>

                            <TableCell>Department</TableCell>

                            <TableCell *align*="right">Total</TableCell>

                            <TableCell *align*="right">Submitted</TableCell>

                            <TableCell *align*="right">Admin Verified</TableCell>

                            <TableCell *align*="right">Exam Cell Verified</TableCell>

                            <TableCell *align*="right">Library Verified</TableCell>

                            <TableCell *align*="right">Branch Verified</TableCell>

                            <TableCell *align*="right">Collected</TableCell>

                        </TableRow>

                    </TableHead>

                    <TableBody>

                        {stats.map((*department*) => (

                            <TableRow *key*={*department*.\_id}>

                                <TableCell *component*="th" *scope*="row">

                                    {*department*.\_id}

                                </TableCell>

                                <TableCell *align*="right">{*department*.total}</TableCell>

                                <TableCell *align*="right">{*department*.submitted}</TableCell>

                                <TableCell *align*="right">{*department*.adminVerified}</TableCell>

                                <TableCell *align*="right">{*department*.examCellVerified}</TableCell>

                                <TableCell *align*="right">{*department*.libraryVerified}</TableCell>

                                <TableCell *align*="right">{*department*.branchVerified}</TableCell>

                                <TableCell *align*="right">{*department*.collected}</TableCell>

                            </TableRow>

                        ))}

                    </TableBody>

                </Table>

            </TableContainer>

        </Box>

    );

};

export default Dashboard;

# AlumniConnect\src\components\EditProfile.jsx

import React, { useState, useEffect, useContext } from "react";  
import { useForm } from "react-hook-form";  
import UserContext from "../context/user/UserContext";  
import "../css/lor.css";  
import studentLogo from "../assets/Navbar/studentLogo.png";  
  
const EditProfile = () => {  
 const context = useContext(UserContext);  
 const { host, user, token, userProfile, setUserProfile, userData, loadProfilePicture, fetchUserData } = context;  
  
 const [otpSent, setOtpSent] = useState({ mobile: false, email: false });  
 const [verified, setVerified] = useState({ mobile: false, email: false });  
 // const [imagePreview, setImagePreview] = useState(studentLogo);  
 const [uploading, setUploading] = useState(false);  
  
 useEffect(() => {  
 if(!userData){  
 fetchUserData(user, token);  
 }  
 if(userData?.profilePicture) {  
 // Load the profile picture from the server  
 loadProfilePicture();  
 // setImagePreview(userProfile)  
 }  
 }, [userData, user, token, host]);  
  
 const { register, handleSubmit, watch, setValue, formState: { errors } } = useForm({  
 defaultValues: {  
 firstName: userData?.firstName || "",  
 middleName: userData?.fatherName || "",  
 lastName: userData?.lastName || "",  
 mobileno: userData?.mobileno || "",  
 email: userData?.email || "",  
 Address: userData?.address || ""  
 }  
 });  
  
 useEffect(() => {  
 setValue("firstName", userData?.firstName || "");  
 setValue("middleName", userData?.fatherName || "");  
 setValue("lastName", userData?.lastName || "");  
 setValue("mobileno", userData?.mobileno || "");  
 setValue("email", userData?.email || "");  
 setValue("Address", userData?.address || "");  
 }, [userData, setValue]);  
  
 // const previewImage = (event) => {  
 // const file = event.target.files[0];  
 // if (file) {  
 // const reader = new FileReader();  
 // reader.onload = (e) => {  
 // setImagePreview(e.target.result);  
 // };  
 // reader.readAsDataURL(file);  
 // }  
 // };  
  
 const triggerFileInput = () => {  
 document.getElementById("imageUpload").click();  
 };  
  
 const handleImageUpload = async (event) => {  
 const file = event.target.files[0];  
 if (!file) return;  
  
 try {  
 setUploading(true);  
 const formData = new FormData();  
 formData.append('profile', file);  
  
 const response = await fetch(`${host}files/upload-profile/${user}`, {  
 method: 'POST',  
 headers: {  
 'auth-token': token,  
 'location': 'profiles'  
 },  
 body: formData  
 });  
  
 const data = await response.json();  
 if (data.success) {  
 alert('Profile picture uploaded successfully!');  
 // Create a preview URL for the uploaded image  
 const imageUrl = URL.createObjectURL(file);  
 setUserProfile(imageUrl)  
 // setImagePreview(imageUrl);  
 } else {  
 alert('Failed to upload profile picture: ' + data.error);  
 }  
 } catch (error) {  
 console.error('Error uploading profile picture:', error);  
 alert('Error uploading profile picture');  
 } finally {  
 setUploading(false);  
 }  
 };  
  
 const sendOtp = async (type, value) => {  
 try {  
 const response = await fetch(  
 `${host}auth/otp/${user}/send${type}otp`,  
 {  
 method: "POST",  
 headers: { "Content-Type": "application/json", "auth-token": token },  
 body: JSON.stringify({ [type === "mobile" ? "mobileno" : "email"]: value }),  
 }  
 );  
  
 const data = await response.json();  
 if (data.success) {  
 alert(`${type} OTP sent successfully!`);  
 setOtpSent((prev) => ({ ...prev, [type]: true }));  
 } else {  
 alert(`Failed to send OTP: ${data.error}`);  
 }  
 } catch {  
 alert("Error sending OTP.");  
 }  
 };  
  
 const verifyOtp = async (type, otp) => {  
  
 try {  
 const response = await fetch(`${host}auth/otp/verify${type}otp`, {  
 method: "POST",  
 headers: { "Content-Type": "application/json", "auth-token": token },  
 body: JSON.stringify({ [type === "mobile" ? "mobileno" : "email"]: watch(type === "mobile" ? "mobileno" : "email"), otp }),  
 });  
  
 const data = await response.json();  
 if (data.success) {  
 alert(`${type} OTP verified successfully!`);  
 setVerified((prev) => ({ ...prev, [type]: true }));  
 } else {  
 alert(`Failed to verify OTP: ${data.message}`);  
 }  
 } catch {  
 alert("Error verifying OTP.");  
 }  
 };  
  
 const onSubmit = async (formData) => {  
 const { mobileno, email } = formData;  
   
 if ((verified.mobile || mobileno === userData.mobileno) && (verified.email || email === userData.email)) {  
 try {  
 const response = await fetch(`${host}auth/${user}/updateuserinfo/`, {  
 method: "PUT",  
 headers: { "Content-Type": "application/json", "auth-token": token },  
 body: JSON.stringify(formData),  
 });  
   
 const data = await response.json();  
 if (data.success) {  
 alert("Updated Profile Successfully!");  
 } else {  
 alert("Failed to Update Information: " + (data.error || "Unknown error"));  
 }  
 } catch (error) {  
 console.error("Error updating data:", error);  
 alert("An error occurred while updating data.");  
 }  
 } else if (!verified.email && email !== userData.email) {  
 alert("Please verify your Email first.");  
 } else if (!verified.mobile && mobileno !== userData.mobileno) {  
 alert("Please verify your Mobile Number first.");  
 } else {  
 alert("Please verify both Mobile Number and Email before submitting.");  
 }  
 }  
   
  
  
  
 return (  
 <div className="form-container">  
 <h1 className="form-header">Profile Page</h1>  
 <br />  
 <hr />  
 <form onSubmit={handleSubmit(onSubmit)} className="ProfileForm">  
 <div className="image-upload-container">  
 <input   
 type="file"   
 id="imageUpload"   
 accept="image/\*"   
 onChange={handleImageUpload}   
 style={{ display: "none" }}   
 />  
 <div className="profile-image-wrapper">  
 <img   
 id="imagePreview"   
 src={userProfile}   
 alt="Profile"   
 className="profile-image"  
 onError={(e) => {  
 e.target.src = studentLogo;  
 }}  
 />  
 <button   
 type="button"   
 className="upload-button"   
 onClick={triggerFileInput}  
 disabled={uploading}  
 >  
 {uploading ? 'Uploading...' : 'Upload Image'}  
 </button>  
 </div>  
 </div>  
 <div className="form-input name">  
 <label htmlFor="name">Name of the Student</label>  
 <div className='nameContainer'>  
 <input type="text" {...register("firstName", { required: "First name is required", minLength: { value: 3, message: "Minimum 3 characters required" } })} placeholder="First Name" />  
 {errors.firstName && <p className="error">{errors.firstName.message}</p>}  
 <input type="text" {...register("middleName", { required: "Middle name is required", minLength: { value: 3, message: "Minimum 3 characters required" } })} placeholder="Middle Name" />  
 {errors.middleName && <p className="error">{errors.middleName.message}</p>}  
 <input type="text" {...register("lastName", { required: "Last name is required", minLength: { value: 3, message: "Minimum 3 characters required" } })} placeholder="Last Name" />  
 {errors.lastName && <p className="error">{errors.lastName.message}</p>}  
 </div>  
 </div>  
 <label htmlFor="mobileno">Mobile Number</label>  
 <div className="innerfieldContainer">  
 {/\* Mobile Number \*/}  
 <div className="passfild" style={{ marginTop: "0px", marginBottom: "20px" }}>  
 <input  
 className="input no-arrow"  
 type="number"  
 disabled={otpSent.mobile}  
 placeholder="Mobile No."  
 {...register("mobileno", { minLength: { value: 10, message: "Mobile number must be at least 10 digits" } })}  
 style={{ width: "160px" }}  
 />  
 <button type="button" className="otp-button" style={{ marginRight: "10px" }} onClick={() => sendOtp("mobile", watch("mobileno"))}>  
 Send OTP  
 </button>  
 </div>  
 {errors.mobileno && <span className="error">{errors.mobileno.message}</span>}  
  
 {/\* Mobile OTP \*/}  
 <div className="passfild" style={{ marginTop: "0px", marginBottom: "20px" }}>  
 <input  
 className="input no-arrow"  
 type="number"  
 disabled={verified.mobile}  
 placeholder="Enter OTP"  
 {...register("mobileotp", { minLength: { value: 6, message: "OTP must be at least 6 digits" } })}  
 style={{ width: "160px" }}  
 />  
 <button type="button" className="otp-button" onClick={() => verifyOtp("mobile", watch("mobileotp"))}>  
 Verify OTP  
 </button>  
 </div>  
 {errors.mobileotp && <span className="error">{errors.mobileotp.message}</span>}  
 </div>  
 <label htmlFor="email" style={{ marginTop: "20px" }}>email</label>  
 <div className="innerfieldContainer">  
 {/\* Email \*/}  
 <div className="passfild" style={{ marginTop: "0px", marginBottom: "20px" }}>  
 <input  
 className="input no-arrow"  
 type="email"  
 disabled={otpSent.email}  
 placeholder="Email ID"  
 {...register("email", { required: "Email is required", pattern: { value: /^[^\s@]+@[^\s@]+\.[^\s@]+$/, message: "Enter a valid email" } })}  
 style={{ width: "160px" }}  
 />  
 <button type="button" className="otp-button" style={{ marginRight: "10px" }} onClick={() => sendOtp("email", watch("email"))}>  
 Send OTP  
 </button>  
 </div>  
 {errors.email && <span className="error">{errors.email.message}</span>}  
  
 {/\* Email OTP \*/}  
 <div className="passfild" style={{ marginTop: "0px", marginBottom: "20px" }}>  
 <input  
 className="input no-arrow"  
 type="number"  
 disabled={verified.email}  
 placeholder="Enter OTP"  
 {...register("emailotp", { minLength: { value: 6, message: "OTP must be at least 6 digits" } })}  
 style={{ width: "160px" }}  
 />  
 <button type="button" className="otp-button" onClick={() => verifyOtp("email", watch("emailotp"))}>  
 Verify OTP  
 </button>  
 </div>  
 {errors.emailotp && <span className="error">{errors.emailotp.message}</span>}  
 </div>  
 <div className="form-input">  
 <label htmlFor="Address">Address</label>  
 <input type="text" {...register("Address", { required: "Address is required", minLength: { value: 3, message: "Minimum 3 characters required" } })} placeholder="Enter your address" />  
 {errors.Address && <p className="error">{errors.Address.message}</p>}  
 </div>  
 <div className="form-input">  
 <button type="submit" className="submit-button">Save</button>  
 </div>  
 </form>  
 </div >  
 );  
};  
  
export default EditProfile;

# AlumniConnect\src\components\ForgotPass.jsx

import React, { useState } from 'react';  
import { useNavigate } from 'react-router-dom'  
import { useForm } from "react-hook-form";  
import openEye from '../assets/Login/openeye.png'  
import closeEye from '../assets/Login/closeeye.png'  
  
const ForgotPass = () => {  
 const navigate = useNavigate();  
 const { register, handleSubmit, watch, formState: { errors } } = useForm();  
  
 const [eyeOpen, setEyeOpen] = useState(false)  
 const toggleEye = () => {  
 setEyeOpen(!eyeOpen)  
 }  
  
 const [otpSent, setOtpSent] = useState(false);  
 const [verified, setVerified] = useState(false);  
  
 // Simulate sending an OTP to the provided email and phone  
 const handleSendOtp = async (formData) => {  
 const { userType, moodleid } = formData;  
 // console.log("Sending OTP to:", userType, moodleid);  
  
 try {  
 const response = await fetch(  
 `http://localhost:5000/api/auth/otp/${userType}/sendotp`,  
 {  
 method: "POST",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify({ moodleid }),  
 }  
 );  
  
 const data = await response.json();  
 if (data.success) {  
 alert("OTP sent successfully!");  
 setOtpSent(true);  
 } else {  
 alert("Failed to send OTP: " + data.error);  
 }  
 } catch (error) {  
 // console.error("OTP Send Error:", error);  
 alert("Error sending OTP.");  
 }  
 };  
  
 const onSubmit = async (formData) => {  
 const { userType, moodleid, password } = formData;  
  
 try {  
 const response = await fetch(  
 `http://localhost:5000/api/auth/${userType}/resetpass/`,  
 {  
 method: "PUT",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify({ moodleid, password }),  
 }  
 );  
  
 const data = await response.json();  
 if (data.success) {  
 alert("Password reset successfully!");  
 navigate('/');  
 } else {  
 alert("Failed to reset Password: " + data.error);  
 }  
 } catch (error) {  
 // console.error("OTP Send Error:", error);  
 alert("Error reseting Password.");  
 }  
 };  
  
 // Simulate OTP verification  
 const handleVerifyOtp = async (formData) => {  
 // console.log('handleSendOtp clicked')  
 const { moodleid, otp } = formData;  
  
 try {  
 const response = await fetch(  
 `http://localhost:5000/api/auth/otp/verifyotp`,  
 {  
 method: "POST",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify({ moodleid, otp }),  
 }  
 );  
  
 const data = await response.json();  
 if (data.success) {  
 alert("otp verified successfully");  
 setVerified(true);  
 } else {  
 alert("Failed to verify OTP: " + data.message);  
 }  
 } catch (error) {  
 // console.error("Failed to varify OTP: ", error);  
 alert("Error verifing OTP.");  
 }  
 }  
  
 return (  
 <div className="forgetContainer">  
 <form onSubmit={handleSubmit(onSubmit)} className="form">  
 {/\* <div className="login\_txt">Login Your Account</div> \*/}  
 <h3 style={{ marginTop: '20px' }}>Please enter your details</h3>  
  
 {/\* select input \*/}  
 <div className="userType">  
 <select type="text" id="userType" disabled={otpSent} {...register("userType", { required: "User type is required" })} required >  
 <option value="">Select User</option>  
 <option value="student">Student</option>  
 <option value="teacher">Teacher</option>  
 </select>  
 </div>  
 {errors.userType && <span className="error">{errors.userType.message}</span>}  
  
 {/\* moodleid input \*/}  
 <div className='passfild'>  
 <input className="input no-arrow" type="number" disabled={otpSent} placeholder="MoodleId" {...register("moodleid", { minLength: { value: 8, message: 'moodleid should be atleast 8 digit' } })} style={{ width: '160px' }} />  
 <input className="otp-button" type="button" value="Send Otp" onClick={handleSubmit(handleSendOtp)} />  
 </div>  
 {errors.moodleid && <span className="error">{errors.moodleid.message}</span>}  
  
 {/\* Password input \*/}  
 <div className='passfild'>  
 <input className="input no-arrow" type='number' disabled={verified} placeholder="Enter OTP" {...register("otp", { minLength: { value: 6, message: 'Otp should be atleast 6 digits' } })} style={{ width: '160px' }} />  
 <input className="otp-button" type="button" value="Varify Otp" onClick={handleSubmit(handleVerifyOtp)} />  
 </div>  
 {errors.varifyOtp && <span className="error">{errors.varifyOtp.message}</span>}  
  
 {/\* Password input \*/}  
 <div className='passfild'>  
 <input className="input" type={eyeOpen ? 'text' : 'password'} placeholder="New Password" {...register("password", {  
 minLength: { value: 8, message: 'Password should be atleast 8 character' }, pattern: {  
 value: /^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$/,  
 message: "Must contain uppercase, lowercase, number, and special character",  
 }  
 })} />  
 <div className="eyeicon" onClick={toggleEye}>  
 <img className="image" src={eyeOpen ? openEye : closeEye} alt="collegeLogo" />  
 </div>  
 </div>  
 {errors.password && <span className="error">{errors.password.message}</span>}  
  
 {/\* confirm password input \*/}  
 <div className='passfild'>  
 <input className="input" type={eyeOpen ? 'text' : 'password'} placeholder="Re-enter Password" {...register("confirmPassword", { minLength: { value: 8, message: 'Password should be atleast 8 character', validate: (value) => value === password || "Passwords do not match", } })} />  
 <div className="eyeicon" onClick={toggleEye}>  
 <img className="image" src={eyeOpen ? openEye : closeEye} alt="collegeLogo" />  
 </div>  
 </div>  
 {errors.password && <span className="error">{errors.password.message}</span>}  
  
  
 <input className="Login submit-button" type="submit" value="Reset" />  
 </form>  
 </div>  
 );  
};  
  
export default ForgotPass;

# AlumniConnect\src\components\Forms.jsx

import React from 'react';  
import { Routes, Route ,Link } from "react-router-dom"; // Use react-router-dom  
import Lor from './Lor'  
import ViewLC from './ViewLC'  
import LC from './LC'  
import ViewConvocation from './ViewConvocation'  
import Convocation from './Convocation'  
import ViewMarksheet from './ViewMarksheet'  
import Marksheet from './Marksheet'  
import '../css/Forms.css';  
import convocationLogo from '../assets/Forms/Convo.png';  
import lcLogo from '../assets/Forms/lc.png';  
import lorLogo from '../assets/Forms/lor.png';  
import marksheetLogo from '../assets/Forms/Marksheet.png';  
  
const Forms = () => {  
 const documents = [  
 { to: '/forms/viewlc', img: lcLogo, label: 'LEAVING CERTIFICATE', alt: 'Leaving Certificate Document' },  
 { to: '/forms/lor', img: lorLogo, label: 'LETTER OF RECOMMENDATION', alt: 'Letter of Recommendation Document' },  
 { to: '/forms/ViewConvocation', img: convocationLogo, label: 'CONVOCATION', alt: 'Convocation Document' },  
 { to: '/forms/ViewMarksheet', img: marksheetLogo, label: 'MARKSHEET', alt: 'Marksheet Document' },  
 ];  
  
 return (  
 <Routes>  
 <Route path="/" element={  
 <div className="mainBodyContainer">  
 {documents.map((doc, index) => (  
 <Link key={index} to={doc.to} aria-label={doc.label}>  
 <div className="doc\_box">  
 <div className="doc\_img">  
 <img className="image" src={doc.img} alt={doc.alt} />  
 </div>  
 <div className="doc\_text">{doc.label}</div>  
 </div>  
 </Link>  
 ))}  
 </div>} />  
 <Route path="/lor" element={<Lor />} />  
 <Route path="/viewlc" element={<ViewLC />} />  
 <Route path="/lc" element={<LC />} />  
 <Route path="/viewconvocation" element={<ViewConvocation />} />  
 <Route path="/convocation" element={<Convocation />} />  
 <Route path="/viewmarksheet" element={<ViewMarksheet />} />  
 <Route path="/marksheet" element={<Marksheet />} />  
 </Routes>  
 );  
};  
  
export default Forms;

# AlumniConnect\src\components\Header.jsx

import React from 'react';  
import CollegeLogo from '../assets/Header/collegeLogo.png'  
import studentLogo from '../assets/Header/studentLogo.png'  
import '../css/Header.css'  
// import '../javascript/updateClgName'  
// import '../javascript/menuToggle'  
  
const Header = (props) => {  
  
 return (  
 <header className={`header ${props.hideMenu?'':(props.tabView?'blur':'')}`}>  
 {!props.hideMenu ? <div className="collegeLogo" onClick={()=>props.toggleHideMenu()}>  
 <img className="image" src={CollegeLogo} alt="College Logo" />  
 </div> :  
 <div className="menuIcon" onClick={()=>props.toggleHideMenu()}>  
 <img className="image" src={studentLogo} alt="Menu Icon" />  
 </div>}  
 {!props.mobileView ? <div className="clg\_name">A. P. Shah Institute of Technology</div> : <div className="clg\_name">APSIT</div>}  
 </header>  
 )  
}  
  
export default Header

# AlumniConnect\src\components\Lc.jsx

import React, { useState, useEffect, useContext } from 'react';  
import { useForm } from "react-hook-form";  
import UserContext from "../context/user/UserContext";  
import { useNavigate } from 'react-router-dom';  
// import '../css/Lor.css';  
  
const Lc = () => {  
 const navigate = useNavigate();  
  
 const context = useContext(UserContext);  
 const { host, user, token, userData, fetchUserData } = context;  
  
 useEffect(() => {  
 if(!userData){  
 fetchUserData(user, token)  
 }  
 }, []);  
  
 const { register, handleSubmit, setValue, formState: { errors } } = useForm({  
 defaultValues: {  
 firstName: userData?.firstName || "",  
 fatherName: userData?.fatherName || "",  
 lastName: userData?.lastName || "",  
 motherName: userData?.motherName || "",  
 grandfatherName: userData?.grandfatherName || "",  
 prn: userData?.prn || "",  
 moodleid: userData?.moodleid || "",  
 religion: userData?.religion || "",  
 caste: userData?.caste || "",  
 category: userData?.category || "",  
 nationality: userData?.nationality || "",  
 placeOfBirth: userData?.placeOfBirth || "",  
 dateOfBirth: userData?.dateOfBirth || "",  
 program: userData?.program || "",  
 branch: userData?.branch || "",  
 semester: userData?.semester || "",  
 lastCollegeAttended: userData?.lastCollegeAttended || "",  
 dateOfAdmission: userData?.dateOfAdmission || "",  
 yearOfJoining: userData?.yearOfJoining || "",  
 }  
 });  
  
 useEffect(() => {  
 if (userData) {  
 setValue("firstName", userData.firstName || "");  
 setValue("fatherName", userData.fatherName || "");  
 setValue("lastName", userData.lastName || "");  
 setValue("motherName", userData.motherName || "");  
 setValue("grandfatherName", userData.grandfatherName || "");  
 setValue("prn", userData.prn || "");  
 setValue("moodleid", userData.moodleid || "");  
 setValue("religion", userData.religion || "");  
 setValue("caste", userData.caste || "");  
 setValue("category", userData.category || "");  
 setValue("nationality", userData.nationality || "");  
 setValue("placeOfBirth", userData.placeOfBirth || "");  
 setValue("program", userData.program || "");  
 // Convert date format from dd-MM-yyyy to yyyy-MM-dd  
 if (userData.dateOfBirth) {  
 const [day, month, year] = userData.dateOfBirth.split('-');  
 setValue("dateOfBirth", `${year}-${month}-${day}`);  
 }  
   
 setValue("branch", userData.branch || "");  
 setValue("semester", userData.semester || "");  
 setValue("lastCollegeAttended", userData.lastCollegeAttended || "");  
   
 if (userData.dateOfAdmission) {  
 const [day, month, year] = userData.dateOfAdmission.split('-');  
 setValue("dateOfAdmission", `${year}-${month}-${day}`);  
 }  
   
 setValue("yearOfJoining", userData.yearOfJoining || "");  
 }  
 }, [userData, setValue]);  
  
 const onSubmit = async (formData) => {  
 try {  
 // Convert date format back to dd-MM-yyyy before sending to server  
 const formattedData = {  
 ...formData,  
 dateOfBirth: formData.dateOfBirth ? new Date(formData.dateOfBirth).toLocaleDateString('en-GB') : null,  
 dateOfAdmission: formData.dateOfAdmission ? new Date(formData.dateOfAdmission).toLocaleDateString('en-GB') : null  
 };  
 // console.log(formattedData);  
 const response = await fetch(`${host}forms/createlc`, {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 },  
 body: JSON.stringify(formattedData)  
 });  
 const data = await response.json();  
 if (data.success) {  
 alert("LC form submitted successfully!");  
 navigate('/checkstatus');  
 } else {  
 alert("Failed to submit LC form: " + data.error);  
 }  
 } catch (error) {  
 console.error("Error submitting LC form:", error);  
 alert("Error submitting LC form. Please try again.");  
 }  
 };  
  
 return (  
 <div className="form-container">  
 <h1 className="form-header">Leaving Certificate Form</h1>  
 <br />  
 <hr />  
 <form onSubmit={handleSubmit(onSubmit)} className="lcForm">  
 {/\* Student Personal Information \*/}  
 <div className="form-section">  
 <br />  
 <div className="fieldContainer">  
 <div className="innerfieldContainer">  
 <div className="form-input name">  
 <label htmlFor="firstName">First name</label>  
 <div className="nameContainer">  
 <input type="text" {...register("firstName", { required: "First name is required" })} placeholder="First Name" />  
 {errors.firstName && <p className="error">{errors.firstName.message}</p>}  
 </div>  
 </div>  
 <div className="form-input name">  
 <label htmlFor="fatherName">Father's name</label>  
 <div className="nameContainer">  
 <input type="text" {...register("fatherName", { required: "Father's name is required" })} placeholder="Father's Name" />  
 {errors.fatherName && <p className="error">{errors.fatherName.message}</p>}  
 </div>  
 </div>  
 <div className="form-input name">  
 <label htmlFor="lastName">Last name</label>  
 <div className="nameContainer">  
 <input type="text" {...register("lastName", { required: "Last name is required" })} placeholder="Last Name" />  
 {errors.lastName && <p className="error">{errors.lastName.message}</p>}  
 </div>  
 </div>  
 </div>  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="motherName">Mother's name</label>  
 <div className="nameContainer">  
 <input type="text" {...register("motherName", { required: "Mother's name is required" })} placeholder="Enter your mother's Name" />  
 {errors.motherName && <p className="error">{errors.motherName.message}</p>}  
 </div>  
 </div>  
 <div className="form-input">  
 <label htmlFor="grandFatherName">Grandfather's name</label>  
 <div className="nameContainer">  
 <input type="text" {...register("grandfatherName", { required: "Grandfather's name is required" })} placeholder="Enter your grandfather's Name" />  
 {errors.grandfatherName && <p className="error">{errors.grandfatherName.message}</p>}  
 </div>  
 </div>  
 <div className="form-input">  
 <label htmlFor="prn">PRN</label>  
 <input type="text" {...register("prn", { required: "PRN number is required" })} placeholder="Enter your PRN" className="no-arrows" />  
 {errors.prn && <p className="error">{errors.prn.message}</p>}  
 </div>  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="moodleid">Moodle ID</label>  
 <input type="text" {...register("moodleid", { required: "Moodle ID is required" })} placeholder="Enter your Moodle Id" className="no-arrows" />  
 {errors.moodleid && <p className="error">{errors.moodleid.message}</p>}  
 </div>  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="religion">Religion</label>  
 <select {...register("religion", { required: "Religion is required" })}>  
 <option value="">Select Religion</option>  
 <option value="HINDU">HINDU</option>  
 <option value="ISLAM">ISLAM</option>  
 <option value="CHRISTIAN">CHRISTIAN</option>  
 <option value="BUDDHIST">BUDDHIST</option>  
 <option value="JAIN">JAIN</option>  
 <option value="ADIVASI">ADIVASI</option>  
 </select>  
 {errors.religion && <p className="error">{errors.religion.message}</p>}  
 </div>  
 <div className="form-input name">  
 <label htmlFor="caste">Caste</label>  
 <div className="nameContainer">  
 <input type="text" {...register("caste", { required: "Caste is required" })} placeholder="Enter your caste" />  
 {errors.caste && <p className="error">{errors.caste.message}</p>}  
 </div>  
 </div>  
   
 <div className="form-input">  
 <label htmlFor="category">Category</label>  
 <select {...register("category", { required: "Category is required" })}>  
 <option value="">Select Category</option>  
 <option value="OBC">OBC</option>  
 <option value="SC">SC</option>  
 <option value="ST">ST</option>  
 <option value="NT">NT</option>  
 </select>  
 {errors.category && <p className="error">{errors.category.message}</p>}  
 </div>  
 <div className="form-input">  
 <label htmlFor="nationality">Nationality</label>  
 <select {...register("nationality", { required: "Nationality is required" })}>  
 <option value="">Select Nationality</option>  
 <option value="INDIAN">INDIAN</option>  
 <option value="AFGHAN">AFGHAN</option>  
 <option value="ALBANIAN">ALBANIAN</option>  
 <option value="ALGERIAN">ALGERIAN</option>  
 <option value="AMERICAN">AMERICAN</option>  
 <option value="ARGENTINIAN">ARGENTINIAN</option>  
 <option value="AUSTRALIAN">AUSTRALIAN</option>  
 <option value="BANGLADESHI">BANGLADESHI</option>  
 <option value="BRAZILIAN">BRAZILIAN</option>  
 <option value="BRITISH">BRITISH</option>  
 <option value="CANADIAN">CANADIAN</option>  
 <option value="CHINESE">CHINESE</option>  
 <option value="EGYPTIAN">EGYPTIAN</option>  
 <option value="FRENCH">FRENCH</option>  
 <option value="GERMAN">GERMAN</option>  
 <option value="INDONESIAN">INDONESIAN</option>  
 <option value="IRANIAN">IRANIAN</option>  
 <option value="IRAQI">IRAQI</option>  
 <option value="IRISH">IRISH</option>  
 <option value="ITALIAN">ITALIAN</option>  
 <option value="JAPANESE">JAPANESE</option>  
 <option value="KOREAN">KOREAN</option>  
 <option value="MALAYSIAN">MALAYSIAN</option>  
 <option value="MEXICAN">MEXICAN</option>  
 <option value="NEPALESE">NEPALESE</option>  
 <option value="PAKISTANI">PAKISTANI</option>  
 <option value="RUSSIAN">RUSSIAN</option>  
 <option value="SAUDI">SAUDI</option>  
 <option value="SINGAPOREAN">SINGAPOREAN</option>  
 <option value="SPANISH">SPANISH</option>  
 <option value="SRI LANKAN">SRI LANKAN</option>  
 <option value="THAI">THAI</option>  
 <option value="TURKISH">TURKISH</option>  
 <option value="VIETNAMESE">VIETNAMESE</option>  
 </select>  
 {errors.nationality && <p className="error">{errors.nationality.message}</p>}  
 </div>  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="placeOfBirth">Place of Birth</label>  
 <input type="text" {...register("placeOfBirth", { required: "Place of birth is required" })} placeholder="Enter your Birth Place" className="no-arrows" />  
 {errors.placeOfBirth && <p className="error">{errors.placeOfBirth.message}</p>}  
 </div>  
 <div className="form-input">  
 <label htmlFor="dateOfBirth">Date of birth</label>  
 <input type="date" {...register("dateOfBirth", { required: "Date of birth is required" })} style={{ display: 'block' }} className="no-arrows" />  
 {errors.dateOfBirth && <p className="error">{errors.dateOfBirth.message}</p>}  
 </div>  
 </div>  
 </div>  
  
 {/\* Academic Details \*/}  
 <div className="form-section">  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="branch">Course</label>  
 <select {...register("branch", { required: "Course is required" })}>  
 <option value="">Select Course</option>  
 <option value="COMPUTER SCIENCE">COMPUTER SCIENCE</option>  
 <option value="INFORMATION TECHNOLOGY">INFORMATION TECHNOLOGY</option>  
 <option value="CSE-DATA SCIENCE">CSE-DATA SCIENCE</option>  
 <option value="CSE-AIML">CSE-AIML</option>  
 <option value="MECHANICAL">MECHANICAL</option>  
 <option value="CIVIL">CIVIL</option>  
 </select>  
 {errors.branch && <p className="error">{errors.branch.message}</p>}  
 </div>  
 <div className="form-input">  
 <label htmlFor="semester">Semester</label>  
 <select {...register("semester", { required: "Semester is required" })}>  
 <option value="">Select Semester</option>  
 <option value="1">Sem 1</option>  
 <option value="2">Sem 2</option>  
 <option value="3">Sem 3</option>  
 <option value="4">Sem 4</option>  
 <option value="5">Sem 5</option>  
 <option value="6">Sem 6</option>  
 <option value="7">Sem 7</option>  
 <option value="8">Sem 8</option>  
 </select>  
 {errors.semester && <p className="error">{errors.semester.message}</p>}  
 </div>  
 </div>  
 <div className="form-input" style={{ Width: '98vh' }}>  
 <label htmlFor="lastCollegeAttended">Last college attended</label>  
 <input type="text" {...register("lastCollegeAttended", { required: "Last college attended is required" })} placeholder="Enter your previous college" />  
 {errors.lastCollegeAttended && <p className="error">{errors.lastCollegeAttended.message}</p>}  
 </div>  
 <div className="form-input">  
 <label htmlFor="dateOfAdmission">Date of admission</label>  
 <input type="date" {...register("dateOfAdmission", { required: "Date of admission is required" })} style={{ display: 'block' }} className="no-arrows" />  
 {errors.dateOfAdmission && <p className="error">{errors.dateOfAdmission.message}</p>}  
 </div>  
 <div className="form-input">  
 <label htmlFor="yearOfJoining">Semester</label>  
 <select {...register("yearOfJoining", { required: "Year Of Joining is required" })}>  
 <option value="">Select Year Of Joining</option>  
 <option value="FE">FE</option>  
 <option value="DSE">DSE</option>  
 </select>  
 {errors.yearOfJoining && <p className="error">{errors.yearOfJoining.message}</p>}  
 </div>  
  
 </div>  
 </div>  
 <hr />  
 <br />  
 <div className="innerfieldContainer instructions">  
 <h3>Note:</h3>  
 <ol>  
 Certified that the above information is in accordance with the College Register.  
 No change in any entry in this certificate shall be made except by the authority issuing it and any infringement of this requirement is liable to involve the imposition of a penalty and or legal action.  
 This certificate can be given only after clearance from various Labs. (No Dues.)  
 </ol>  
 </div>  
 <br />  
  
 {/\* Form Submission \*/}  
 <div className="form-input button">  
 <button type="submit" className="submit-button">Submit Request</button>  
 </div>  
 </form>  
 </div>  
 );  
};  
  
export default Lc;

# AlumniConnect\src\components\Login.jsx

import React, { useState, useEffect, useContext } from 'react'  
import { Link, useNavigate } from 'react-router-dom'  
import { useForm } from "react-hook-form";  
import cookies from "js-cookie"  
import UserContext from "../context/user/UserContext"  
import '../css/Login.css'  
import collegeLogo from '../assets/Login/collegeLogo.png'  
import loginPage from '../assets/Login/loginPage.png'  
import openEye from '../assets/Login/openeye.png'  
import closeEye from '../assets/Login/closeeye.png'  
import userLogo from '../assets/Login/user.png'  
  
const Login = () => {  
 const navigate = useNavigate();  
 const { register, handleSubmit, watch, formState: { errors } } = useForm();  
  
 const context = useContext(UserContext)  
 const {host, user, setUser, fetchFirstLogin } = context  
  
 // useEffect(() => {  
 // if (user === 'student') {  
 // console.log('student')  
 // navigate('/forms');  
 // } else if (user === 'teacher') {  
 // console.log('teacher')  
 // navigate('/dashboard');  
 // }  
 // else {  
 // navigate('/');  
 // }  
 // }, []);   
  
 useEffect(() => {  
 if(cookies.get('first-login')){  
 navigate('/varifycontact')  
 return   
 }  
 // else   
 if (user === 'student') {  
 // console.log('student')  
 navigate('/forms');  
 } else if (user === 'teacher') {  
 // console.log('teacher')  
 navigate('/dashboard');  
 }  
 else {  
 navigate('/');  
 }  
 }, [user]);  
  
 const onSubmit = async (formData) => {  
 try {  
 const { moodleid, password, userType } = formData;  
  
 if (!userType) {  
 setLoginError("User type is required.");  
 return;  
 }  
  
 const response = await fetch(`${host}auth/${userType}/login`, {  
 method: "Post",  
 headers: {  
 "Content-Type": "application/json",  
 },  
 body: JSON.stringify({ moodleid, password }),  
 });  
 const data = await response.json();  
  
 if (data.success) {  
 const firstLogin = await fetchFirstLogin(userType, data.token)  
 if(firstLogin){  
 // console.log(typeof(firstLogin))  
 cookies.set('auth-token', data.token, { expires: 1 / 288 })  
 cookies.set('user', userType, { expires: 1 / 288 })  
 cookies.set('first-login', true, { expires: 1 / 288 })  
 setUser(userType)  
 }else{  
 // console.log("Login Info:", data);  
 cookies.set('auth-token', data.token, { expires: 2 })  
 cookies.set('user', userType, { expires: 2 })  
 setUser(userType)  
 // alert("Passeord reset successfully!");  
 }  
 } else {  
 alert("Failed to Login: " + data.error);  
 }  
  
 } catch (error) {  
 // console.error("Error fetching login info:", error);  
 alert("Error fetching login info");  
 return null;  
 }  
 };  
  
 const [eyeOpen, setEyeOpen] = useState(false)  
 const toggleEye = () => {  
 setEyeOpen(!eyeOpen)  
 }  
 return (  
 <div>  
  
 <div className="loginContainer">  
 <div className="Logo\_container content-to-hide">  
 <img className="image content-to-hide" src={loginPage} alt="loginPage" />  
 </div>  
 <div className="Login\_window">  
 <div className="student\_Logo">  
 <img className="image" src={collegeLogo} alt="collegeLogo" />  
 </div>  
 <form onSubmit={handleSubmit(onSubmit)} className="form">  
 <div className="login\_txt">Login Your Account</div>  
 <div className="txt">Please enter your details</div>  
  
 {/\* moodleid input \*/}  
 <div className='passfild'>  
 <input className="input no-arrow" type="number" placeholder="moodleid" {...register("moodleid", { minLength: { value: 8, message: 'moodleid should be atleast 8 digit' } })} />  
 <div className="eyeicon">  
 <img className="image" src={userLogo} alt="collegeLogo" />  
 </div>  
 </div>  
 {errors.moodleid && <span className="error">{errors.moodleid.message}</span>}  
  
 {/\* Password input \*/}  
 <div className='passfild'>  
 <input className="input" type={eyeOpen ? 'text' : 'password'} placeholder="Password" {...register("password", { minLength: { value: 8, message: 'Password should be atleast 8 character' } })} />  
 <div className="eyeicon" onClick={toggleEye}>  
 <img className="image" src={eyeOpen ? openEye : closeEye} alt="collegeLogo" />  
 </div>  
 </div>  
 {errors.password && <span className="error">{errors.password.message}</span>}  
  
 {/\* select input \*/}  
 <div className="userType">  
 <select type="text" id="userType" {...register("userType", { required: "User type is required" })} required >  
 <option value="">Select User</option>  
 <option value="student">Student</option>  
 <option value="teacher">Teacher</option>  
 </select>  
 </div>  
 {errors.userType && <span className="error">{errors.userType.message}</span>}  
  
 {/\* checkbox input \*/}  
 <div id='checkbox'>  
 <label className="innerfieldContainer">  
 <input type="checkbox" {...register("rememberMe")} />  
 Remember me  
 </label>  
 <Link rel="stylesheet" to='/forgotpass'>Forgot Password?</Link>  
 {/\* <label className="checkBox innerfieldContainer">  
 <input type="checkbox" />  
 Show Password  
 </label> \*/}  
 </div>  
 {errors.exampleRequired && <span>This field is required</span>}  
 <input className="Login submit-button" type="submit" value="Login" />  
 </form>  
 </div>  
 </div>  
 </div>  
 )  
}  
  
export default Login;

# AlumniConnect\src\components\Lor.jsx

import React, { useState } from 'react';  
import '../css/Lor.css';  
  
const Lor = () => {  
 const teacherNamesIT = ["Select Staff Name", "Mr.Samir Nanivadekar", "Mr.Vishal Badgujar", "Mr.Mandar Ganjapurkar", "Mr.Ganesh Gourshete", "Mr.Sachin Kasare", "Ms.Rujata Chaudhari", "Ms.Saylee Lapalikar", "Ms.Sonal Jain", "Ms.Sonal Balpande", "Ms.Charul Singh", "Ms.Sonia Aneesh", "Ms.Rucha Kulkarni", "Ms.Jayshree Jha", "Ms.Roshna Sangle", "Ms.Urjashree Patil", "Ms.Shafaque Sayed", "Ms.Manjusha Kashilkar", "Ms.Shweta Mahajan", "Ms.Geetanjali Kalme", "Ms.Apeksha Mohite", "Ms.Randeep kaur Kahlon", "Ms.Sujata Oak", "Ms.Snehal Mali", "Ms.Anupama Singh"];  
  
 const teacherNamesCS = ["Select Staff Name", "Nidhi Patil", "Snehal Kulkarni", "Pallavi Salunkhe", "Janhavi Hirey", "Yash Sonar", "Shripad Dharmadhikari", "Sejal Chavhan", "Sanikya Yadkikar", "Shruti Patil", "Shreyas Patil", "Amit Sali", "Shivraj Shewale", "Pritesh Kachave"];  
  
 const teacherNamesCSEAI = ["Select Staff Name", "Divya Varkhede", "Saee Desale", "Pranali Bhamre", "Parth Shinde", "Sejal Kaklij"]  
 const teacherNamesCSEDS = ["Select Staff Name", "Sahil Desale", "Shruti Shirole", "Sarvesh Deore", "Prerana Bhavsar", "Dhanashri Yeole"]  
 const teacherNamesECE = ["Select Staff Name", "Mohit Deore", "Anand Yeolekar", "Khushi Deo", "Amruta Roy"]  
 const teacherNamesME = ["Select Staff Name", "praniv", "siddhi", "sarthak", "Vedant"]  
  
 const [teacherList, setTeacherList] = useState(['Select Staff Name']);  
  
 const handleBranchChange = (event) => {  
 const branch = event.target.value;  
  
 if (branch === 'IT') {  
 setTeacherList(teacherNamesIT);  
 }  
 else if (branch === 'CSE') {  
 setTeacherList(teacherNamesCS);  
 }  
 else if (branch === 'CSE-AI') {  
 setTeacherList(teacherNamesCSEAI);  
 }  
 else if (branch === 'CSE-DS') {  
 setTeacherList(teacherNamesCSEDS);  
 }  
 else if (branch === 'ECE') {  
 setTeacherList(teacherNamesECE);  
 }  
 else if (branch === 'ME') {  
 setTeacherList(teacherNamesME);  
 }  
 else {  
 setTeacherList(['Select Staff Name']);  
 }  
 };  
  
 const currentYear = new Date().getFullYear();  
 const startYear = 2016;  
 const years = [];  
  
 for (let year = startYear; year <= currentYear; year++) {  
 years.push(year);  
 }  
  
 return (  
 <div className="form-container">  
 <h1 className="form-header">LOR Application</h1>  
 <br />  
 <hr />  
 <form className="lorform">  
 <div className="form-input name">  
 <label htmlFor="name">Name of the Student</label>  
 <div className="nameContainer">  
 <input type="text" id="FirstName" name="FirstName" placeholder="First Name" required />  
 <input type="text" id="MiddleName" name="MiddleName" placeholder="Middle Name" required />  
 <input type="text" id="LastName" name="LastName" placeholder="Last Name" required />  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="studentId">Moodle ID</label>  
 <input type="number" id="studentId" name="studentId" placeholder="Enter your Moodle ID" className="no-arrows" required />  
 </div>  
 <div className="form-input">  
 <label htmlFor="yearJoined">Year of Joining the Institute</label>  
 <select id="yearJoined" name="yearJoined" required>  
 <option value="">Select Year</option>  
 <option value="First Year">First Year</option>  
 <option value="Direct Second Year">Direct Second Year</option>  
 </select>  
 </div>  
 </div>  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="year">Year of Admission</label>  
 <select type="number" id="year" name="admissionYear" required >  
 <option key="selectyear" value="SelectYear">Select Year</option>  
 {years.map((year) => (  
 <option key={year} value={year}>  
 {year}  
 </option>  
 ))}  
 </select>  
 </div>  
 <div className="form-input">  
 <label htmlFor="Course">Course</label>  
 <select id="course" name="course" required>  
 <option value="">Select Course</option>  
 <option value="Computer Science">COMPUTER</option>  
 <option value="Information Technology">INFORMATION TECHNOLOGY</option>  
 <option value="Information Technology">CSE-DATA SCIENCE</option>  
 <option value="Information Technology">CSE-AIML</option>  
 <option value="Mechanical Engineering">MECHANICAL</option>  
 <option value="Civil Engineering">CIVIL</option>  
 </select>  
 </div>  
 </div>  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="currentStatus">Currently Studying In</label>  
 <select id="currentStatus" name="currentStatus" required>  
 <option value="">Please Select</option>  
 <option value="PASSOUT">PASSOUT</option>  
 <option value="BE">BE</option>  
 <option value="TE">TE</option>  
 <option value="SE">SE</option>  
 <option value="FE">FE</option>  
 <option value="DROP">DROP</option>  
 </select>  
 </div>  
 <div className="form-input">  
 <label htmlFor="lorCopies">Number of LOR Copies Required</label>  
 <select type="number" id="lorCopies" name="lorCopies" required >  
 <option value="SelectCopies">Select number of copies</option>  
 <option value="1">1</option>  
 <option value="2">2</option>  
 <option value="3">3</option>  
 <option value="4">4</option>  
 <option value="5">5</option>  
 <option value="6">6</option>  
 <option value="7">7</option>  
 <option value="8">8</option>  
 <option value="9">9</option>  
 <option value="10">10</option>  
 </select>  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="form-input staffSelection">  
 <label htmlFor="staffName">LOR FROM STAFF<span className="form-required">\*</span></label>  
  
 <div className="innerfieldContainer">  
 <label className="innerfieldContainer checkBox">  
 <input type="checkbox" />  
 Principle  
 </label>  
 <label className="checkBox innerfieldContainer">  
 <input type="checkbox" />  
 Hod  
 </label>  
 </div>  
 {[...Array(4)].map((\_, i) => (  
 <select key={i} id={`staffName${i}`} name={`staffName${i}`} className="form-select" required>  
 {teacherList.map((teacher, index) => (  
 <option key={index} value={teacher}>{teacher}</option>  
 ))}  
 </select>  
 ))}  
 </div>  
 </div>  
 <hr />  
 <br />  
 <div className="form-input instructions">  
 <h3>PROCEDURE TO FOLLOW:</h3>  
 <ol style={{ marginLeft: '15px' }}>  
 <li>Rs.10/- per copy per letter head will be charged.</li>  
 <li>Submit Application form online only.</li>  
 <li>In response to your request, demand for payment to be paid will be made available in the admission portal.</li>  
 <li>Payment has to be done within 2 working days.</li>  
 <li>Pay the generated demand online.</li>  
 <li>LOR will be issued within 05 working days after payment is done.</li>  
 </ol>  
 </div>  
 </div>  
 <div className="form-input button">  
 <button type="submit" className="submit-button">Submit</button>  
 </div>  
 </form>  
 </div>  
 );  
}  
  
export default Lor;

# AlumniConnect\src\components\MainBody.jsx

import React, { useEffect, useContext } from "react";  
import { Routes, Route, useNavigate } from "react-router-dom";  
import UserContext from "../context/user/UserContext"  
import Forms from "./Forms";  
import CheckStatus from "./CheckStatus";  
import Profile from "./Profile";  
import Notice from "./Notice";  
import AddNotice from "./AddNotice";  
import AddStudent from "./AddStudent";  
import "../css/MainBody.css";  
  
const MainBody = ({ hideMenu, tabView }) => {  
 const navigate = useNavigate();  
  
 const context = useContext(UserContext)  
 const { user, setUser, userData, fetchFirstLogin } = context  
 useEffect(() => {  
 if (user==='student') {  
 navigate('/forms');  
 }else if(user==='teacher'){  
 navigate('/dashboard');  
 }  
 else{  
 navigate('/');  
 }  
 }, []);   
  
 return (  
 <div className={`main\_body ${hideMenu ? "" : tabView ? "blur" : ""}`}>  
 <Routes>  
 <Route path="/forms/\*" element={<Forms />} />  
 <Route path="/checkStatus/\*" element={<CheckStatus />} />  
 <Route path="/profile/\*" element={<Profile />} />  
 <Route path="/notice/\*" element={<Notice />} />  
 <Route path="/addnotice/\*" element={<AddNotice />} />  
 <Route path="/addstudent/\*" element={<AddStudent />} />  
  
 </Routes>  
 </div>  
 );  
};  
  
export default MainBody;

# AlumniConnect\src\components\Marksheet.jsx

import React, { useState } from 'react';  
// import '../css/Lor.css';  
  
const Marksheet = () => {  
 // Generate an array of years (for Year of Admission/Leaving)  
 const currentYear = new Date().getFullYear();  
 const startYear = 2000; // Adjust as needed for your institution  
 const years = [];  
 for (let year = startYear; year <= currentYear; year++) {  
 years.push(year);  
 }  
  
 return (  
 <div className="form-container">  
 <h1 className="form-header">Marksheet Form</h1>  
 <br />  
 <hr />  
 <form className="ConvocationForm">  
 {/\* Student Personal Information \*/}  
 <div className="form-section">  
 <br />  
 <div className="fieldContainer">  
 <div className="innerfieldContainer">  
 <div className="form-input name">  
 <label htmlFor="name">Name of the Student</label>  
 <div className="nameContainer">  
 <input type="text" id="FirstName" name="FirstName" placeholder="First Name" required />  
 <input type="text" id="MiddleName" name="MiddleName" placeholder="Middle Name" required />  
 <input type="text" id="LastName" name="LastName" placeholder="Last Name" required />  
 <input type="text" id="MotherName" name="MotherName" placeholder="Mother's Name" required />  
 </div>  
 </div>  
 </div>  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="prn">PRN number</label>  
 <input type="text" id="prn" name="prn" placeholder="Enter your PRN" className="no-arrows" required />  
 </div>  
 <div className="form-input">  
 <label htmlFor="moodleid">Moodle ID</label>  
 <input type="text" id="moodleid" name="moodleid" placeholder="Enter your Moodle ID" className="no-arrows" required />  
 </div>  
 </div>  
 </div>  
 {/\* Academic Details \*/}  
 <div className="form-section">  
 <div className="innerfieldContainer">  
 <div className="form-input">  
 <label htmlFor="Course">Course</label>  
 <select id="course" name="course" required>  
 <option value="">Select Course</option>  
 <option value="Computer Science">COMPUTER</option>  
 <option value="Information Technology">INFORMATION TECHNOLOGY</option>  
 <option value="Information Technology">CSE-DATA SCIENCE</option>  
 <option value="Information Technology">CSE-AIML</option>  
 <option value="Mechanical Engineering">MECHANICAL</option>  
 <option value="Civil Engineering">CIVIL</option>  
 </select>  
 </div>  
 </div>  
 </div>  
 </div >  
 <hr />  
 <br />  
 {/\* Form Submission \*/}  
 <div className="form-input button">  
 <button type="submit" className="submit-button">Submit Request</button>  
 </div>  
 </form >  
 </div >  
 );  
};  
  
export default Marksheet;

# AlumniConnect\src\components\Navbar.jsx

import React, {useContext, useEffect} from 'react';  
import { NavLink, useLocation, useNavigate } from "react-router-dom";  
import cookies from "js-cookie";  
import '../css/Navbar.css';  
import UserContext from "../context/user/UserContext"  
import studentLogo from '../assets/Navbar/StudentLogo.png';  
import profile from '../assets/Navbar/profile.png';  
import forms from '../assets/Navbar/Forms.png';  
import checkStatus from '../assets/Navbar/checkStatus.png';  
import notification from '../assets/Navbar/notification.png';  
import logout from '../assets/Navbar/Logout.png';  
import addStudent from '../assets/Navbar/Logout.png'; // Add this import  
  
const Navbar = ({ hideMenu, toggleHideMenu, tabView }) => {  
 const context = useContext(UserContext)  
 const { token, user, setUser, userProfile,loadProfilePicture, userData, fetchUserData, unreadCount } = context  
  
 const location = useLocation();  
 const navigate = useNavigate();  
   
 // Define navItems based on user type  
 const navItems = user === 'student'  
 ? [  
 { to: '/profile', img: profile, label: 'Profile', isActive: location.pathname.startsWith('/profile') },  
 { to: '/forms', img: forms, label: 'Forms', isActive: location.pathname.startsWith('/forms') },  
 { to: '/checkStatus', img: checkStatus, label: 'Check Status', isActive: location.pathname.startsWith('/checkStatus') },  
 { to: '/notice', img: notification, label: 'Notice', isActive: location.pathname.startsWith('/notice'), unreadCount },  
 ]  
 : [  
 { to: '/profile', img: profile, label: 'Profile', isActive: location.pathname.startsWith('/profile') },  
 { to: '/dashboard', img: forms, label: 'Dashboard', isActive: location.pathname.startsWith('/dashboard') },  
 { to: '/checkStatus', img: checkStatus, label: 'Check Status', isActive: location.pathname.startsWith('/checkStatus') },  
 { to: '/addnotice', img: notification, label: 'Add Notice', isActive: location.pathname.startsWith('/AddNotice') },  
 ...(userData && userData.department === 'admin' ? [{ to: '/addstudent', img: addStudent, label: 'Add Student', isActive: location.pathname.startsWith('/addstudent') }] : []),  
 ];  
  
 const handleLogout = () => {  
 setUser(null)  
 Object.keys(cookies.get()).forEach(cookie => cookies.remove(cookie));  
 navigate('/')  
 }  
  
 useEffect(() => {  
 if(!userProfile){  
 loadProfilePicture()  
 }  
 if(!userData){  
 fetchUserData(cookies.get("user"), cookies.get("auth-token"))  
 }  
 }, [])  
  
 return (  
 !hideMenu && (  
 <nav className={`nav ${hideMenu ? 'hideNav' : tabView ? 'showNav' : ''}`}>  
 {/\* Student Info Section \*/}  
 <div className="stu-info" onClick={toggleHideMenu}>  
 <div className="stu-img">  
 <img className="image" src={userProfile} alt="Student Logo" />  
 </div>  
 <div className="stu-name"> {userData && userData.firstName} </div>  
 </div>  
  
 {/\* Navigation Items \*/}  
 {navItems.map((item, index) => (  
 <NavLink key={index} to={item.to} end className={({ isActive }) => isActive || item.isActive ? 'nav\_container activeLink' : 'nav\_container'}>  
 <div className="img-container">  
 <img className="image" src={item.img} alt={`${item.label} Icon`} />  
 </div>  
 <div className="nav-item">{item.label}</div>  
 {item.unreadCount>0 && (  
 <div className="notification-count">{item.unreadCount}</div>  
 )}  
 </NavLink>  
 ))}  
 <button className='nav\_container' onClick={handleLogout}>  
 <div className="img-container">  
 <img className="image" src={logout} alt='Logout Icon' />  
 </div>  
 <div className="nav-item">Logout</div>  
 </button>  
 </nav>  
 )  
 );  
};  
  
export default Navbar;

# AlumniConnect\src\components\Notice.jsx

import React, { useState, useEffect, useContext } from 'react';  
import UserContext from '../context/user/UserContext';  
import '../css/Notice.css';  
  
function Notice() {  
 const { host, token, unreadCount, setUnreadCount } = useContext(UserContext);  
 const [notifications, setNotifications] = useState([]);  
 const [expandedId, setExpandedId] = useState(null);  
  
 useEffect(() => {  
 fetchNotifications();  
 }, []);  
  
 const fetchNotifications = async () => {  
 try {  
 const response = await fetch(`${host}notification/getnotifications`, {  
 method: 'GET',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 }  
 });  
 const data = await response.json();  
 if (data.success) {  
 setNotifications(data.notifications);  
 }  
 } catch (error) {  
 console.error('Error fetching notifications:', error);  
 }  
 };  
  
 const markAsRead = async (notificationId) => {  
 try {  
 const response = await fetch(`${host}notification/markread/${notificationId}`, {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 }  
 });  
   
 if (response.ok) {  
 // Update the notifications state locally  
 setNotifications(notifications.map(notif =>   
 notif.\_id === notificationId   
 ? { ...notif, isRead: true }  
 : notif  
 ));  
 setUnreadCount(unreadCount - 1);  
 }  
 } catch (error) {  
 console.error('Error marking notification as read:', error);  
 }  
 };  
  
 const handleCardClick = async (index, notificationId) => {  
 setExpandedId(expandedId === index ? null : index);  
   
 // If expanding and notification is unread, mark it as read  
 if (expandedId !== index && !notifications[index].isRead) {  
 await markAsRead(notificationId);  
 }  
 };  
  
 return (  
 <div className="form-container">  
 <div className='checkastatusHeader'>  
 <h1 className="form-header">Notice</h1>  
 <div className="notification-count">  
 {notifications.length} notification{notifications.length !== 1 ? 's' : ''}  
 {notifications.filter(n => !n.isRead).length > 0 &&  
 ` • ${notifications.filter(n => !n.isRead).length} unread`}  
 </div>  
 </div>  
 <br />  
 <hr />  
 <br />  
  
 <div className="notifications-container">  
 {notifications.length === 0 ? (  
 <div className="empty-status">  
 <p>No notifications available</p>  
 </div>  
 ) : (  
 notifications.map((notification, index) => (  
 <div  
 key={index}  
 className={`status-card ${expandedId === index ? 'expanded' : ''} ${!notification.isRead ? 'unread' : ''}`}  
 onClick={() => handleCardClick(index, notification.\_id)}  
 >  
 <div className={`status-card-header ${expandedId === index ? 'expanded' : ''}`}>  
 <div className="status-info">  
 <div className="status-title">  
 <span className="department">{notification.title}</span>  
 </div>  
 <div className="status-date">  
 {new Date(notification.timestamp).toLocaleDateString('en-US', {  
 day: 'numeric',  
 month: 'short',  
 year: 'numeric',  
 hour: '2-digit',  
 minute: '2-digit'  
 })}  
 {!notification.isRead && <span className="new-badge">New</span>}  
 </div>  
 </div>  
 </div>  
  
 {expandedId === index && (  
 <div className="status-card-content">  
 <div className="message-content">  
 {notification.message}  
 </div>  
 </div>  
 )}  
 </div>  
 ))  
 )}  
 </div>  
 </div>  
 );  
}  
  
export default Notice;

# AlumniConnect\src\components\NotificationModal.jsx

import React, { useState } from 'react';  
import '../css/NotificationModal.css';  
  
const NotificationModal = ({ isOpen, onClose, onConfirm, formId }) => {  
 const [message, setMessage] = useState('');  
 const [title, setTitle] = useState('');  
  
 if (!isOpen) return null;  
  
 const handleSubmit = () => {  
 if (!message.trim() || !title.trim()) {  
 alert('Please enter both title and message');  
 return;  
 }  
 onConfirm(formId, message, title);  
 setMessage('');  
 setTitle('');  
 onClose();  
 };  
  
 return (  
 <div className="modal-overlay" onClick={onClose}>  
 <div className="modal-content" onClick={e => e.stopPropagation()}>  
 <h2>Send Notification</h2>  
 <div className="modal-body">  
 <input  
 type="text"  
 value={title}  
 onChange={(e) => setTitle(e.target.value)}  
 placeholder="Enter notification title"  
 className="notification-title-input"  
 />  
 <textarea  
 value={message}  
 onChange={(e) => setMessage(e.target.value)}  
 placeholder="Enter your message here..."  
 className="notification-textarea"  
 />  
 </div>  
 <div className="modal-buttons">  
 <button onClick={handleSubmit} className="send-btn">Send</button>  
 <button onClick={onClose} className="cancel-btn">Cancel</button>  
 </div>  
 </div>  
 </div>  
 );  
};  
  
export default NotificationModal;

# AlumniConnect\src\components\Profile.jsx

import React, { useState } from "react";  
import { Routes, Route ,Link } from "react-router-dom";  
// import '../css/Profile.css'  
import ViewProfile from "./ViewProfile";  
import EditProfile from "./EditProfile";  
// import Edit from '../assets/Profile/Edit.png'  
  
  
const Profile = () => {  
 return (  
 <Routes>  
 <Route path="/" element={ <ViewProfile /> } />  
 <Route path="/editprofile" element={ <EditProfile />} />  
 </Routes>  
 );  
};  
  
export default Profile;

# AlumniConnect\src\components\StudentCheckStatus.jsx

import React, { useEffect, useState, useContext } from 'react'  
import '../css/CheckStatus.css'  
import UserContext from "../context/user/UserContext"  
import Application from './Application'  
  
const StudentCheckStatus = () => {  
 const context = useContext(UserContext)  
 const { host, user, token } = context  
  
 const [forms, setForms] = useState([])  
  
 const fetchForms = async (user, token) => {  
 try {  
 const response = await fetch(`${host}forms/${user}/getforms`, {  
 method: "Post",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 }  
 });  
 if (!response.ok) {  
 throw new Error('Failed to fetch user data');  
 }  
 const forms = await response.json();  
 // console.log(forms.forms)  
 setForms(forms.forms)  
 } catch (error) {  
 console.error("Error fetching user data:", error);  
 }  
 };  
  
 useEffect(() => {  
 fetchForms(user, token)  
 }, [])  
  
 return (  
 <div className="form-container">  
 <div className='checkastatusHeader'>  
 <h1 className="form-header">Application Status</h1>  
 {/\* <div className="filter">  
 <select className="status\_filter" name="status\_filter" id="status\_filter">  
 <option value="select\_filter">Select Filter</option>  
 <option value="completed">Completed</option>  
 <option value="in-progress">In-progress</option>  
 <option value="submitted">Submitted</option>  
 </select>  
 <button className="submit-button">Apply</button>  
 </div> \*/}  
 </div>  
 <br />  
 <hr />  
 <br />  
  
 <div className="check\_status">  
 <div className="form-list-header">  
 <div className="header-name">Form Name</div>  
 <div className="header-status">Status</div>  
 <div className="header-date">Submission Date</div>  
 </div>  
 {forms.length > 0 ? (  
 forms.map((form) => {  
 // Convert date format  
 const formattedDate = new Date(form.createdAt).toLocaleDateString('en-GB', {  
 day: '2-digit',  
 month: '2-digit',  
 year: 'numeric'  
 });  
   
 return (  
 <Application   
 key={form.\_id}   
 fname={form.docType}   
 fstatus={form.status}   
 fdate={formattedDate}  
 formData={form}  
 />  
 );  
 })  
 ) : (  
 <p>No forms found</p>  
 )}  
 </div>  
 </div>  
 )  
}  
  
export default StudentCheckStatus

# AlumniConnect\src\components\TeacherCheckStatus.jsx

import React, { useEffect, useState, useContext } from 'react'  
import '../css/CheckStatus.css'  
import UserContext from "../context/user/UserContext"  
import NotificationModal from './NotificationModal'  
  
const TeacherCheckStatus = () => {  
 const context = useContext(UserContext)  
 const { host, user, token, userData, fetchUserData } = context  
  
 const [forms, setForms] = useState([])  
 const [filter, setFilter] = useState('all')  
 const [loading, setLoading] = useState(true)  
 const [error, setError] = useState(null)  
 const [expandedForm, setExpandedForm] = useState(null)  
 const [docData, setDocData] = useState(null)  
 const [isModalOpen, setIsModalOpen] = useState(false)  
 const [selectedFormId, setSelectedFormId] = useState(null)  
 const [showConfirmation, setShowConfirmation] = useState(false)  
 const [confirmationMessage, setConfirmationMessage] = useState('')  
 const [dropdownOpen, setDropdownOpen] = useState(null)  
  
 useEffect(() => {  
 if (!userData && user && token) {  
 fetchUserData(user, token);  
 }  
 }, [userData, user, token]);  
  
 useEffect(() => {  
 if (userData && user && token) {  
 fetchForms(user, token);  
 }  
 }, [userData, user, token]);  
  
 // useEffect(() => {  
 // console.log('UserData:', userData);  
 // console.log('Department:', userData?.department);  
 // }, [userData]);  
  
 const fetchForms = async (user, token) => {  
 try {  
 setLoading(true)  
 setError(null)  
 const response = await fetch(`${host}forms/teacher/getforms`, {  
 method: "Post",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 }  
 });  
 if (!response.ok) {  
 throw new Error('Failed to fetch forms data');  
 }  
 const data = await response.json();  
 // console.log('API Response:', data);  
 if (data && data.forms) {  
 setForms(data.forms)  
 // console.log('Forms set:', data.forms);  
 } else {  
 setForms([])  
 }  
 } catch (error) {  
 console.error("Error fetching forms data:", error);  
 setError('Failed to load forms. Please try again later.')  
 setForms([])  
 } finally {  
 setLoading(false)  
 }  
 };  
  
 const fetchDocData = async (docId, docType) => {  
 try {  
 const response = await fetch(`${host}forms/${docType}/${docId}`, {  
 method: "Get",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 }  
 });  
 if (!response.ok) {  
 throw new Error('Failed to fetch document data');  
 }  
 const data = await response.json();  
 if (data.success && data.doc) {  
 setDocData(data.doc);  
 } else {  
 console.error('Failed to fetch document data:', data.error);  
 setDocData(null);  
 }  
 } catch (error) {  
 console.error("Error fetching document data:", error);  
 setLcData(null);  
 }  
 };  
  
 const handleFilterChange = (e) => {  
 setFilter(e.target.value);  
 };  
  
 const toggleExpand = async (formId, docId, docType) => {  
 if (expandedForm === formId) {  
 setExpandedForm(null);  
 setDocData(null);  
 } else {  
 setExpandedForm(formId);  
 if(docType === 'Leaving Certificate') {  
 await fetchDocData(docId, 'lc');  
 }else if(docType === 'Letter of Recommendation') {  
 await fetchDocData(docId, 'lor');  
 }  
 }  
 };   
  
 const filteredForms = forms.filter(form => {  
 if (filter === 'all') return true;  
  
 // For Admin Department  
 if (userData?.department === 'admin') {  
 if (filter === 'Pending') {  
 return !form.admin; // Pending admin verification  
 }  
 if (filter === 'Verified') {  
 return form.admin && (!form.examCell || !form.library || !form.atBranch); // Verified by admin but not complete  
 }  
 if (filter === 'Completed') {  
 return form.admin && form.examCell && form.library && form.atBranch; // All complete  
 }  
 }  
  
 // For Exam Cell  
 if (userData?.department === 'examCell') {  
 if (filter === 'Pending') {  
 return form.admin && !form.examCell; // Admin verified but pending exam cell  
 }  
 if (filter === 'Verified') {  
 return form.admin && form.examCell && (!form.library || !form.atBranch); // Exam cell verified but not complete  
 }  
 if (filter === 'Completed') {  
 return form.admin && form.examCell && form.library && form.atBranch; // All complete  
 }  
 }  
  
 // For Library  
 if (userData?.department === 'library') {  
 if (filter === 'Pending') {  
 return form.admin && form.examCell && !form.library; // Previous depts verified but pending library  
 }  
 if (filter === 'Verified') {  
 return form.admin && form.examCell && form.library && !form.atBranch; // Library verified but not at branch  
 }  
 if (filter === 'Completed') {  
 return form.admin && form.examCell && form.library && form.atBranch; // All complete  
 }  
 }  
  
 // For Branch  
 if (userData?.department !== 'admin' && userData?.department !== 'examCell' && userData?.department !== 'library') {  
 if (filter === 'Pending') {  
 return form.admin && form.examCell && form.library && !form.atBranch; // All previous verified but pending branch  
 }  
 if (filter === 'Verified' || filter === 'Completed') {  
 return form.admin && form.examCell && form.library && form.atBranch; // Branch verified  
 }  
 }  
  
 return false;  
 });  
  
 // useEffect(() => {  
 // console.log('Current department:', userData?.department);  
 // console.log('Current filter:', filter);  
 // console.log('All forms:', forms);  
 // console.log('Filtered forms:', filteredForms);  
 // }, [userData, filter, forms, filteredForms]);  
  
 const handleDropdownToggle = (e, formId) => {  
 e.stopPropagation();  
 setDropdownOpen(dropdownOpen === formId ? null : formId);  
 };  
  
 const updateStatus = async (action, formId) => {  
 try {  
 const response = await fetch(`${host}forms/updatestatus/${formId}`, {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token,  
 },  
 body: JSON.stringify({ action })  
 });  
 const data = await response.json();  
   
 if (data.success) {  
 setConfirmationMessage('Form status updated successfully');  
 setShowConfirmation(true);  
 fetchForms(user, token);  
 return true;  
 } else {  
 setConfirmationMessage(data.error || 'Failed to update form status');  
 setShowConfirmation(true);  
 return false;  
 }  
 } catch (error) {  
 console.error('Error updating form status:', error);  
 setConfirmationMessage('Failed to update form status');  
 setShowConfirmation(true);  
 }  
 };  
  
 const handleApprove = async (e, formId) => {  
 e.stopPropagation();  
 setDropdownOpen(null);  
 const res = await updateStatus('approve', formId);  
 const message = 'The form has been approved. please check the form details and update the status';  
 const title = 'Form approved by ' + userData?.department;  
 if(res.success) {  
 sendNotification(formId, message, title);  
 }  
 };  
  
 const handleFees = async (e, formId) => {  
 e.stopPropagation();  
 setDropdownOpen(null);  
 const message = 'you have not paid the fees. please pay the remaining fees on the portal and upload the receipt';  
 const title = 'Fees not paid';  
 const res = await updateStatus('fees', formId);  
 // if(res.success) {  
 // console.log("entered")  
 sendNotification(formId, message, title);  
 // }  
 // setSelectedFormId(formId);  
 // setIsModalOpen(true);  
 };  
  
 const handleDetails = async (e, formId) => {  
 e.stopPropagation();  
 setDropdownOpen(null);  
 const message = 'The form details are incorrect. please provide the correct details by cancelling the form and re-submitting it or provide the document proof to varify the provided details are correct';  
 const title = 'Incorrect form details';  
 const res = await updateStatus('details', formId);  
 if(res.success) {  
 sendNotification(formId, message, title);  
 }  
 // setSelectedFormId(formId);  
 // setIsModalOpen(true);  
 };  
  
 const getFormStatus = (form) => {  
 // For Admin Department  
 if (userData?.department === 'admin') {  
 if (!form.admin) {  
 return "Pending";  
 }  
 if (form.admin && (!form.examCell || !form.library || !form.atBranch)) {  
 return "Verified";  
 }  
 if (form.admin && form.examCell && form.library && form.atBranch) {  
 return "Completed";  
 }  
 }  
  
 // For Exam Cell  
 if (userData?.department === 'examCell') {  
 if (form.admin && !form.examCell) {  
 return "Pending";  
 }  
 if (form.admin && form.examCell && (!form.library || !form.atBranch)) {  
 return "Verified";  
 }  
 if (form.admin && form.examCell && form.library && form.atBranch) {  
 return "Completed";  
 }  
 }  
  
 // For Library  
 if (userData?.department === 'library') {  
 if (form.admin && form.examCell && !form.library) {  
 return "Pending";  
 }  
 if (form.admin && form.examCell && form.library && !form.atBranch) {  
 return "Verified";  
 }  
 if (form.admin && form.examCell && form.library && form.atBranch) {  
 return "Completed";  
 }  
 }  
  
 // For Branch  
 if (userData?.department !== 'admin' && userData?.department !== 'examCell' && userData?.department !== 'library') {  
 if (form.admin && form.examCell && form.library && !form.atBranch) {  
 return "Pending";  
 }  
 if (form.admin && form.examCell && form.library && form.atBranch) {  
 return "Verified";  
 }  
 }  
  
 return "Pending";  
 };  
  
 // Update the CSS classes accordingly  
 const getStatusClass = (status) => {  
 switch (status) {  
 case "Completed":  
 return "status-completed";  
 case "Verified":  
 return "status-verified";  
 case "Pending":  
 return "status-pending";  
 default:  
 return "";  
 }  
 };  
  
 const sendNotification = async (formId, message, title) => {  
 try {  
 const response = await fetch(`${host}notification/notify`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 },  
 body: JSON.stringify({  
 formId,  
 message,  
 title  
 })  
 });  
  
 const data = await response.json();  
   
 if (data.success) {  
 alert('Notification sent successfully');  
 } else {  
 alert(data.error || 'Failed to send notification');  
 }  
 } catch (error) {  
 console.error('Error sending notification:', error);  
 alert('Failed to send notification');  
 }  
 };  
  
 const handleFileClick = async (e, fileType, formId) => {  
 e.preventDefault();  
 try {  
 const response = await fetch(`${host}files/get-${fileType}/${formId}`, {  
 headers: {  
 'auth-token': token  
 }  
 });  
   
 if (response.ok) {  
 const blob = await response.blob();  
 const url = window.URL.createObjectURL(blob);  
 window.open(url, '\_blank');  
 } else {  
 const errorData = await response.json();  
 alert(`Error: ${errorData.error}`);  
 }  
 } catch (error) {  
 console.error('Error opening file:', error);  
 alert('Error opening file');  
 }  
 };  
  
 if (loading) {  
 return <div className="form-container">Loading...</div>  
 }  
  
 if (error) {  
 return <div className="form-container">{error}</div>  
 }  
  
 return (  
 <div className="form-container">  
 <div className='checkastatusHeader'>  
 <h1 className="form-header">Pending Verifications</h1>  
 <div className="filter">  
 <select   
 className="status\_filter"   
 name="status\_filter"   
 id="status\_filter"  
 value={filter}  
 onChange={handleFilterChange}  
 >  
 <option value="all">All Forms</option>  
 <option value="Pending">Pending</option>  
 <option value="Verified">Verified</option>  
 <option value="Completed">Completed</option>  
 </select>  
 </div>  
 </div>  
 <br />  
 <hr />  
 <br />  
  
 <div className="check\_status">  
 <div className="teacher-header">  
 <div className="teacher-header-name">Form ID</div>  
 <div className="teacher-header-status">Status</div>  
 <div className="teacher-header-actions">Actions</div>  
 <div className="teacher-header-date">Submission Date</div>  
 </div>  
 {filteredForms.length > 0 ? (  
 filteredForms.map((form) => {  
 const formattedDate = new Date(form.createdAt).toLocaleDateString('en-GB', {  
 day: '2-digit',  
 month: '2-digit',  
 year: 'numeric'  
 });  
   
 return (  
 <div key={form.\_id}>  
 <div className="teacher-appln" onClick={() => toggleExpand(form.\_id, form.docId, form.docType)}>  
 <div className="teacher-appln-name">{form.docId}</div>  
 <div className={`teacher-header-status ${getStatusClass(getFormStatus(form))}`}>  
 {getFormStatus(form)}  
 </div>  
 <div className="teacher-header-actions">  
 <div className="dropdown-container">  
 <button   
 className="dropdown-toggle"  
 onClick={(e) => handleDropdownToggle(e, form.\_id)}  
 >  
 Actions  
 </button>  
 {dropdownOpen === form.\_id && (  
 <div className="dropdown-menu">  
 <button   
 className="dropdown-item"  
 onClick={(e) => handleApprove(e, form.\_id)}  
 >  
 Approve  
 </button>  
 <button   
 className="dropdown-item"  
 onClick={(e) => handleFees(e, form.\_id)}  
 >  
 Fees not paid  
 </button>  
 {userData.department==='admin' && <button   
 className="dropdown-item"  
 onClick={(e) => handleDetails(e, form.\_id)}  
 >  
 Incorrect form details  
 </button>}  
 </div>  
 )}  
 </div>  
 </div>  
 <div className="teacher-appln-date">{formattedDate}</div>  
 </div>  
 {expandedForm === form.\_id && docData && (  
 <div className="tracking-container expanded">  
 <div className="lc-details">  
 <h2 className="form-type-header">{form.docType}</h2>  
 <div className="lc-section">  
 <h3>Personal Information</h3>  
 <p><strong>Name:</strong> {docData.firstName} {docData.fatherName} {docData.lastName}</p>  
 <p><strong>Mother's Name:</strong> {docData.motherName}</p>  
 <p><strong>Grandfather's Name:</strong> {docData.grandfatherName}</p>  
 <p><strong>Date of Birth:</strong> {new Date(docData.dateOfBirth).toLocaleDateString()}</p>  
 <p><strong>Place of Birth:</strong> {docData.placeOfBirth}</p>  
 </div>  
 <div className="lc-section">  
 <h3>Academic Information</h3>  
 <p><strong>PRN:</strong> {docData.prn}</p>  
 <p><strong>Moodle ID:</strong> {docData.moodleid}</p>  
 <p><strong>Branch:</strong> {docData.branch}</p>  
 <p><strong>Program:</strong> {docData.program}</p>  
 <p><strong>Semester:</strong> {docData.semester}</p>  
 </div>  
 <div className="lc-section">  
 <h3>Additional Information</h3>  
 <p><strong>Year of Joining:</strong> {docData.yearOfJoining}</p>  
 <p><strong>Date of Admission:</strong>{docData.dateOfAdmission}</p>  
 <p><strong>Last College Attended:</strong> {docData.lastCollegeAttended}</p>  
 <p><strong>Religion:</strong> {docData.religion}</p>  
 <p><strong>Caste:</strong> {docData.caste}</p>  
 <p><strong>Category:</strong> {docData.category}</p>  
 <p><strong>Nationality:</strong> {docData.nationality}</p>  
 </div>  
 <div className="lc-section">  
 <h3>Uploaded Documents</h3>  
 <div className="document-buttons">  
 {form.feesRecieptUploded && (  
 <button   
 className="check-button"  
 onClick={(e) => handleFileClick(e, 'fees-reciept', form.\_id)}  
 >  
 Check Fees Receipt  
 </button>  
 )}  
 {form.verificationDocUploaded && (  
 <button   
 className="check-button"  
 onClick={(e) => handleFileClick(e, 'verification-document', form.\_id)}  
 >  
 Check Verification Document  
 </button>  
 )}  
 </div>  
 </div>  
 </div>  
 </div>  
 )}  
 </div>  
 );  
 })  
 ) : (  
 <p>No forms found</p>  
 )}  
 </div>  
  
 <NotificationModal   
 isOpen={isModalOpen}  
 onClose={() => setIsModalOpen(false)}  
 onConfirm={sendNotification}  
 formId={selectedFormId}  
 />  
  
 {showConfirmation && (  
 <div className="confirmation-popup">  
 <div className="confirmation-content">  
 <p>{confirmationMessage}</p>  
 <button onClick={() => setShowConfirmation(false)}>OK</button>  
 </div>  
 </div>  
 )}  
 </div>  
 )  
}  
  
export default TeacherCheckStatus

# AlumniConnect\src\components\VerifyContact.jsx

import React, { useState, useEffect, useContext } from "react";  
import { useNavigate } from "react-router-dom";  
import { useForm } from "react-hook-form";  
import cookies from "js-cookie"  
import UserContext from "../context/user/UserContext";  
import openEye from "../assets/Login/openeye.png";  
import closeEye from "../assets/Login/closeeye.png";  
  
const VarifyContact = () => {  
 const navigate = useNavigate();  
 const { register, handleSubmit, watch, formState: { errors } } = useForm();  
 const { host, user, token } = useContext(UserContext);  
  
 const [eyeOpen, setEyeOpen] = useState(false);  
 const [otpSent, setOtpSent] = useState({ mobile: false, email: false });  
 const [verified, setVerified] = useState({ mobile: false, email: false });  
  
 const toggleEye = () => setEyeOpen(!eyeOpen);  
  
 const sendOtp = async (type, value) => {  
 try {  
 const response = await fetch(  
 `${host}auth/otp/${user}/send${type}otp`,  
 {  
 method: "POST",  
 headers: { "Content-Type": "application/json", "auth-token": token },  
 body: JSON.stringify({ [type === "mobile" ? "mobileno" : "email"]: value }),  
 }  
 );  
  
 const data = await response.json();  
 if (data.success) {  
 alert(`${type} OTP sent successfully!`);  
 setOtpSent((prev) => ({ ...prev, [type]: true }));  
 } else {  
 alert(`Failed to send OTP: ${data.error}`);  
 }  
 } catch {  
 alert("Error sending OTP.");  
 }  
 };  
  
 const verifyOtp = async (type, otp) => {  
  
 try {  
 const response = await fetch(`${host}auth/otp/verify${type}otp`, {  
 method: "POST",  
 headers: { "Content-Type": "application/json", "auth-token": token },  
 body: JSON.stringify({ [type === "mobile" ? "mobileno" : "email"]: watch(type === "mobile" ? "mobileno" : "email"), otp }),  
 });  
  
 const data = await response.json();  
 if (data.success) {  
 alert(`${type} OTP verified successfully!`);  
 setVerified((prev) => ({ ...prev, [type]: true }));  
 } else {  
 alert(`Failed to verify OTP: ${data.message}`);  
 }  
 } catch {  
 alert("Error verifying OTP.");  
 }  
 };  
  
 const onSubmit = async (formData) => {  
 if (verified.mobile && verified.email) {  
 const { mobileno, email, password } = formData  
 try {  
 const response = await fetch(`${host}auth/${user}/setpass/`, {  
 method: "PUT",  
 headers: { "Content-Type": "application/json", "auth-token": token },  
 body: JSON.stringify({ mobileno, email, password }),  
 });  
  
 const data = await response.json();  
 if (data.success) {  
 alert("Password reset successfully!");  
 cookies.remove('first-login')  
 navigate("/");  
 } else {  
 alert("Failed to reset password: " + data.message);  
 }  
 } catch {  
 alert("Error resetting password.");  
 }  
 }  
 else{  
 alert('First Verify the Mobile Number and Email')  
 }  
 };  
  
 useEffect(() => {  
 if (cookies.get('first-login')) {  
 navigate('/varifycontact')  
 return  
 }  
 // else   
 if (user === 'student') {  
 // console.log('student')  
 navigate('/forms');  
 } else if (user === 'teacher') {  
 // console.log('teacher')  
 navigate('/dashboard');  
 }  
 else {  
 navigate('/');  
 }  
 }, [user]);  
  
 return (  
 <div className="forgetContainer">  
 <form onSubmit={handleSubmit(onSubmit)} className="form">  
 <h3 style={{ marginTop: "20px" }}>Please enter your details</h3>  
  
 {/\* Mobile Number \*/}  
 <div className="passfild">  
 <input  
 className="input no-arrow"  
 type="number"  
 disabled={otpSent.mobile}  
 placeholder="Mobile No."  
 {...register("mobileno", { minLength: { value: 10, message: "Mobile number must be at least 10 digits" } })}  
 style={{ width: "160px" }}  
 />  
 <button type="button" className="otp-button" onClick={() => sendOtp("mobile", watch("mobileno"))}>  
 Send OTP  
 </button>  
 </div>  
 {errors.mobileno && <span className="error">{errors.mobileno.message}</span>}  
  
 {/\* Mobile OTP \*/}  
 <div className="passfild">  
 <input  
 className="input no-arrow"  
 type="number"  
 disabled={verified.mobile}  
 placeholder="Enter OTP"  
 {...register("mobileotp", { minLength: { value: 6, message: "OTP must be at least 6 digits" } })}  
 style={{ width: "160px" }}  
 />  
 <button type="button" className="otp-button" onClick={() => verifyOtp("mobile", watch("mobileotp"))}>  
 Verify OTP  
 </button>  
 </div>  
 {errors.mobileotp && <span className="error">{errors.mobileotp.message}</span>}  
  
 {/\* Email \*/}  
 <div className="passfild">  
 <input  
 className="input no-arrow"  
 type="email"  
 disabled={otpSent.email}  
 placeholder="Email ID"  
 {...register("email", { required: "Email is required", pattern: { value: /^[^\s@]+@[^\s@]+\.[^\s@]+$/, message: "Enter a valid email" } })}  
 style={{ width: "160px" }}  
 />  
 <button type="button" className="otp-button" onClick={() => sendOtp("email", watch("email"))}>  
 Send OTP  
 </button>  
 </div>  
 {errors.email && <span className="error">{errors.email.message}</span>}  
  
 {/\* Email OTP \*/}  
 <div className="passfild">  
 <input  
 className="input no-arrow"  
 type="number"  
 disabled={verified.email}  
 placeholder="Enter OTP"  
 {...register("emailotp", { minLength: { value: 6, message: "OTP must be at least 6 digits" } })}  
 style={{ width: "160px" }}  
 />  
 <button type="button" className="otp-button" onClick={() => verifyOtp("email", watch("emailotp"))}>  
 Verify OTP  
 </button>  
 </div>  
 {errors.emailotp && <span className="error">{errors.emailotp.message}</span>}  
  
 {/\* Password \*/}  
 <div className="passfild">  
 <input  
 className="input"  
 type={eyeOpen ? "text" : "password"}  
 placeholder="Enter new password"  
 {...register("password", {  
 minLength: { value: 8, message: "Password must be at least 8 characters" },  
 pattern: { value: /^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&]).{8,}$/, message: "Must contain lowercase, uppercase, number, and special character" },  
 })}  
 />  
 <img className="eyeicon" src={eyeOpen ? openEye : closeEye} onClick={toggleEye} alt="Toggle visibility" />  
 </div>  
 {errors.password && <span className="error">{errors.password.message}</span>}  
  
 {/\* confirm password input \*/}  
 <div className='passfild'>  
 <input className="input" type={eyeOpen ? 'text' : 'password'} placeholder="Re-enter Password" {...register("confirmPassword", { minLength: { value: 8, message: 'Password should be atleast 8 character', validate: (value) => value === password || "Passwords do not match", } })} />  
 <div className="eyeicon" onClick={toggleEye}>  
 <img className="image" src={eyeOpen ? openEye : closeEye} alt="collegeLogo" />  
 </div>  
 </div>  
 {errors.password && <span className="error">{errors.password.message}</span>}  
  
 <input className="Login submit-button" type="submit" value="Reset" />  
 </form>  
 </div>  
 );  
};  
  
export default VarifyContact;

# AlumniConnect\src\components\ViewConvocation.jsx

import React, { useContext, useEffect } from "react";  
import { Link, useNavigate } from "react-router-dom";  
import UserContext from "../context/user/UserContext"  
// import '../css/Lor.css';  
  
const ViewConvocation = () => {  
 const navigate = useNavigate();  
 const context = useContext(UserContext)  
 const { host, user, token, fetchUserData, userData } = context  
  
 useEffect(() => {  
 if (!userData) {  
 fetchUserData(user, token)  
 }  
 // const { moodleid, firstName, fatherName, lastName, email, mobileno, Address } = userData  
 }, [])  
  
 // const /userData = userData;  
 // console.log(/userData)  
  
 const submitConvocation = async () => {  
 try {  
 const ConvocationData = {  
 firstName: userData.firstName,  
 fatherName: userData.fatherName,  
 lastName: userData.lastName,  
 motherName: userData.motherName,  
 prn: userData.prn,  
 moodleid: userData.moodleid,  
 religion: userData.religion,  
 caste: userData.caste,  
 category: userData.category,  
 nationality: userData.nationality,  
 placeOfBirth: userData.placeOfBirth,  
 dateOfBirth: userData.dateOfBirth,  
 branch: userData.branch,  
 program: userData.program,  
 yearOfJoining: userData.yearOfJoining,  
 dateOfAdmission: userData.dateOfAdmission  
 };  
  
 const response = await fetch(`${host}forms/createlc`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 },  
 body: JSON.stringify(lcData)  
 });  
  
 const data = await response.json();  
  
 if (data.success) {  
 alert('Convocation form submitted successfully!');  
 // Redirect to check status page or home page  
 navigate('/checkstatus');  
 } else {  
 alert(data.error || 'Failed to submit Convocation form');  
 }  
 } catch (error) {  
 console.error('Error submitting LC form:', error);  
 alert('Failed to submit Convovation form. Please try again.');  
 }  
 };  
  
 return (  
 !userData ? (  
 <div>Loading...</div> // Display loading message if userData is not available  
 ) : (  
 <div className="form-container">  
 <h1>Student Convocation Preview</h1>  
 <br />  
 <hr />  
 <div style={{ maxWidth: "800px", margin: "auto", padding: "20px" }}>  
 <div style={{ padding: "10px", border: "1px solid #ccc", background: "#f9f9f9" }}>  
 <form className="viewlcform">  
 <div className="form-section">  
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Name:</strong> {userData.firstName} {userData.fatherName} {userData.lastName} </p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Mother's Name:</strong> {userData.motherName}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>PRN:</strong> {userData.prn}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Moodle Id:</strong> {userData.moodleid}</p>  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Date of Birth:</strong> {userData.dateOfBirth}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Program:</strong> {userData.program}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Course:</strong> {userData.branch}</p>  
 </div>  
 </div>  
  
 {/\* Add year of admission nd year of joining in preview \*/}  
  
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Date of Admission:</strong> {userData.dateOfAdmission}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Year of Joining:</strong> {userData.yearOfJoining}</p>  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Year of Convocation:</strong> {userData.yearofConvocation}</p>  
 </div>  
 </div>  
 <br />  
 <div className="innerfieldContainer" >  
 <div className="form-button">  
 <Link className='submit-button' onClick={submitConvocation}>Confirm</Link>  
 </div>  
 <div className="form-button">  
 <Link to={'/forms/convocation'} className='submit-button' >Edit</Link>  
 </div>  
 </div>  
 </div>  
 </form>  
 </div >  
 </div>  
 </div>  
 )  
 );  
};  
  
export default ViewConvocation;

# AlumniConnect\src\components\ViewLC.jsx

import React, { useContext, useEffect } from "react";  
import { Link, useNavigate } from "react-router-dom";  
import UserContext from "../context/user/UserContext"  
// import '../css/Lor.css';  
  
const ViewLC = () => {  
 const navigate = useNavigate();  
 const context = useContext(UserContext)  
 const { host, user, token, fetchUserData, userData } = context  
  
 useEffect(() => {  
 if(!userData){  
 fetchUserData(user, token)  
 }  
 // const { moodleid, firstName, fatherName, lastName, email, mobileno, Address } = userData  
 }, [])  
  
 // const /userData = userData;  
 // console.log(/userData)  
  
 const submitLC = async () => {  
 try {  
 const lcData = {  
 firstName: userData.firstName,  
 fatherName: userData.fatherName,  
 lastName: userData.lastName,  
 motherName: userData.motherName,  
 grandfatherName: userData.grandfatherName,  
 prn: userData.prn,  
 moodleid: userData.moodleid,  
 religion: userData.religion,  
 caste: userData.caste,  
 category: userData.category,  
 nationality: userData.nationality,  
 placeOfBirth: userData.placeOfBirth,  
 dateOfBirth: userData.dateOfBirth,  
 branch: userData.branch,  
 program: userData.program,  
 semester: userData.semester,  
 lastCollegeAttended: userData.lastCollegeAttended,  
 yearOfJoining: userData.yearOfJoining,  
 dateOfAdmission: userData.dateOfAdmission  
 };  
  
 const response = await fetch(`${host}forms/createlc`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 },  
 body: JSON.stringify(lcData)  
 });  
  
 const data = await response.json();  
   
 if (data.success) {  
 alert('LC form submitted successfully!');  
 // Redirect to check status page or home page  
 navigate('/checkstatus');  
 } else {  
 alert(data.error || 'Failed to submit LC form');  
 }  
 } catch (error) {  
 console.error('Error submitting LC form:', error);  
 alert('Failed to submit LC form. Please try again.');  
 }  
 };  
  
 return (  
 !userData ? (  
 <div>Loading...</div> // Display loading message if userData is not available  
 ) : (  
 <div className="form-container">  
 <h1>Student LC Preview</h1>  
 <br />  
 <hr />  
 <div style={{ maxWidth: "800px", margin: "auto", padding: "20px" }}>  
 <div style={{ padding: "10px", border: "1px solid #ccc", background: "#f9f9f9" }}>  
 <form className="viewlcform">  
 <div className="form-section">  
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Name:</strong> {userData.firstName} {userData.fatherName} {userData.lastName} </p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Father's Name:</strong> {userData.fatherName} {userData.grandfatherName} {userData.lastName} </p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Mother's Name:</strong> {userData.motherName}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>PRN:</strong> {userData.prn}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Moodle Id:</strong> {userData.moodleid}</p>  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Religion:</strong> {userData.religion}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Caste:</strong> {userData.caste}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Category:</strong> {userData.category}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Nationality:</strong> {userData.nationality}</p>  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Place of Birth:</strong> {userData.placeOfBirth}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Date of Birth:</strong> {userData.dateOfBirth}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Course:</strong> {userData.branch}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Semester:</strong> {userData.semester} </p>  
 {/\* <p><strong>Semester:</strong> {userData.sem}</p> \*/}  
 </div>  
 </div>  
  
 {/\* Add year of admission nd year of joining in preview \*/}  
   
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Last college attended:</strong> {userData.lastCollegeAttended}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Date of Admission:</strong> {userData.dateOfAdmission}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Year of Joining:</strong> {userData.yearOfJoining}</p>  
 </div>  
 </div>  
 <br />  
 <div className="innerfieldContainer" >  
 <div className="form-button">  
 <Link className='submit-button' onClick={submitLC}>Confirm</Link>  
 </div>  
 <div className="form-button">  
 <Link to={'/forms/lc'} className='submit-button' >Edit</Link>  
 </div>  
 </div>  
 </div>  
 </form>  
 </div >  
 </div>  
 </div>  
 )  
 );  
};  
  
export default ViewLC;

# AlumniConnect\src\components\ViewMarksheet.jsx

import React, { useContext, useEffect } from "react";  
import { Link, useNavigate } from "react-router-dom";  
import UserContext from "../context/user/UserContext"  
// import '../css/Lor.css';  
  
const ViewMarksheet = () => {  
 const navigate = useNavigate();  
 const context = useContext(UserContext)  
 const { host, user, token, fetchUserData, userData } = context  
  
 useEffect(() => {  
 if (!userData) {  
 fetchUserData(user, token)  
 }  
 // const { moodleid, firstName, fatherName, lastName, email, mobileno, Address } = userData  
 }, [])  
  
 // const /userData = userData;  
 // console.log(/userData)  
  
 const submitMarksheet = async () => {  
 try {  
 const ConvocationData = {  
 firstName: userData.firstName,  
 fatherName: userData.fatherName,  
 lastName: userData.lastName,  
 motherName: userData.motherName,  
 prn: userData.prn,  
 moodleid: userData.moodleid,  
 branch: userData.branch,  
 program: userData.program,  
 semester: userData.semester,  
 };  
  
 const response = await fetch(`${host}forms/createlc`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 },  
 body: JSON.stringify(lcData)  
 });  
  
 const data = await response.json();  
  
 if (data.success) {  
 alert('Marksheet information submitted successfully!');  
 // Redirect to check status page or home page  
 navigate('/checkstatus');  
 } else {  
 alert(data.error || 'Failed to submit Marksheet information');  
 }  
 } catch (error) {  
 console.error('Error submitting LC form:', error);  
 alert('Failed to submit Marksheet information. Please try again.');  
 }  
 };  
  
 return (  
 !userData ? (  
 <div>Loading...</div> // Display loading message if userData is not available  
 ) : (  
 <div className="form-container">  
 <h1>Student Marksheet Preview</h1>  
 <br />  
 <hr />  
 <div style={{ maxWidth: "800px", margin: "auto", padding: "20px" }}>  
 <div style={{ padding: "10px", border: "1px solid #ccc", background: "#f9f9f9" }}>  
 <form className="viewlcform">  
 <div className="form-section">  
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Name:</strong> {userData.firstName} {userData.fatherName} {userData.lastName} </p>  
 </div>  
  
 <div className="form-input name">  
 <p><strong>Mother's Name:</strong> {userData.motherName}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>PRN:</strong> {userData.prn}</p>  
 </div>  
 <div className="form-input name">  
 <p><strong>Moodle Id:</strong> {userData.moodleid}</p>  
 </div>  
 </div>  
 <div className="fieldContainer">  
 <div className="form-input name">  
 <p><strong>Course:</strong> {userData.branch}</p>  
 </div>  
 </div>  
  
 {/\* Add year of admission nd year of joining in preview \*/}  
 <br />  
 <div className="innerfieldContainer" >  
 <div className="form-button">  
 <Link className='submit-button' onClick={submitMarksheet}>Confirm</Link>  
 </div>  
 <div className="form-button">  
 <Link to={'/forms/marksheet'} className='submit-button' >Edit</Link>  
 </div>  
 </div>  
 </div>  
 </form>  
 </div >  
 </div>  
 </div>  
 )  
 );  
};  
  
export default ViewMarksheet;

# AlumniConnect\src\components\ViewProfile.jsx

import React, { useState, useContext, useEffect } from "react";  
import { Link } from "react-router-dom";  
// import '../css/Profile.css'  
import UserContext from "../context/user/UserContext"  
// import Edit from '../assets/Navbar/studentLogo.png'  
// import Edit from '../assets/Profile/Edit.png'  
  
  
const ViewProfile = () => {  
   
 const context = useContext(UserContext)  
 const { host, user, token, userProfile, loadProfilePicture,fetchUserData, userData } = context  
 // Function to handle image preview  
 const previewImage = (event) => {  
 const file = event.target.files[0];  
 if (file) {  
 const reader = new FileReader();  
 reader.onload = (e) => {  
 setImagePreview(e.target.result);  
 };  
 reader.readAsDataURL(file);  
 }  
 };  
  
 // Function to trigger file input click  
 const triggerFileInput = () => {  
 document.getElementById("imageUpload").click();  
 };  
  
 useEffect(() => {  
 if(!userProfile){  
 loadProfilePicture()  
 }  
 if(!userData){  
 fetchUserData(user, token)  
 }  
 // const { moodleid, firstName, fatherName, lastName, email, mobileno, Address } = userData  
 }, [])  
  
  
 return (  
 !userData ? (  
 <div>Loading...</div> // Display loading message if userData is not available  
 ) : (  
 <div className="form-container">  
 <div>  
 <h1 className="form-header">Profile Page</h1>  
 </div>  
 <br />  
 <hr />  
 <div style={{ maxWidth: "800px", margin: "auto", padding: "20px" }}>  
 <div style={{ padding: "10px", border: "1px solid #ccc", background: "#f9f9f9" }}>  
 <form method="POST" action="" className="ProfileForm">  
 <div className="image-upload-container">  
 <label htmlFor="imageUpload" style={{ cursor: "pointer" }}>  
 <img id="imagePreview" src={userProfile} alt="Upload" style={{ maxWidth: "150px", height: "auto", borderRadius: "50%" }} />  
 </label>  
 </div>  
 <br />  
 <div className="form-input">  
 <label htmlFor="name">Name of the Student: </label>  
 <div className='nameContainer'>  
 <label htmlFor="name">  
 {userData.firstName} {userData.fatherName} {userData.lastName}  
 </label>  
 </div>  
 </div>  
 <div className="form-input">  
 <label htmlFor="phone">Phone Number: </label>  
 <label htmlFor="mblno">{userData.mobileno || "N/A"}</label>  
 </div>  
 <div className="form-input">  
 <label htmlFor="email">Email: </label>  
 <label htmlFor="email">{userData.email || "N/A"}</label>  
 </div>  
 <div className="form-input">  
 <label htmlFor="Address">Address: </label>  
 <label htmlFor="address">{userData.address || "N/A"}</label>  
 </div>  
 <div className="form-input">  
 <label htmlFor="studentId">Moodle Id: </label>  
 <label htmlFor="m-id">{userData.moodleid || "N/A"}</label>  
 </div>  
 <div className="form-input">  
 <Link to={'/profile/editprofile'} end className='submit-button'>  
 Edit  
 </Link>  
 </div>  
 <br />  
 </form>  
 </div >  
 </div>  
 </div>  
 )  
 );  
  
};  
  
export default ViewProfile;

# AlumniConnect\src\context\user\UserContext.jsx

import { createContext } from "react";  
  
const UserContext = createContext()  
  
export default UserContext

# AlumniConnect\src\context\user\UserState.jsx

import UserContext from "./UserContext";  
import { useState, useEffect } from "react";  
import cookies from "js-cookie"  
import studentLogo from '../../assets/Navbar/studentLogo.png'  
  
const UserState = (props) => {  
 const host = 'http://localhost:5000/api/'  
 const [user, setUser] = useState(null)  
 const [token, setToken] = useState(null)  
 const [userProfile, setUserProfile] = useState(studentLogo)  
 const [userData, setUserData] = useState(null)  
 const [notifications, setNotifications] = useState([])  
 // const [notificationCount, setNotificationCount] = useState(0)  
 const [unreadCount, setUnreadCount] = useState(0)  
  
 const loadProfilePicture = async () => {  
 try {  
 const response = await fetch(`${host}files/profile/${user}`, {  
 headers: {  
 'auth-token': cookies.get("auth-token")  
 }  
 });  
 if (response.ok) {  
 const blob = await response.blob();  
 const imageUrl = URL.createObjectURL(blob);  
 // setImagePreview(imageUrl);  
 setUserProfile(imageUrl)  
 }  
 } catch (error) {  
 console.error('Error loading profile picture:', error);  
 }  
 };  
  
 const fetchUserData = async (user, token) => {  
 try {  
 const response = await fetch(`${host}auth/${user}/getuser`, {  
 method: "Post",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 }  
 });  
 if (!response.ok) {  
 throw new Error('Failed to fetch user data');  
 }  
 const data = await response.json();  
 // console.log(data.userData)  
 setUserData(data.userData)  
 } catch (error) {  
 console.error("Error fetching user data:", error);  
 }  
 };  
  
 const fetchFirstLogin = async (user, token) => {  
 try {  
 const response = await fetch(`${host}auth/${user}/getfirstlogin`, {  
 method: "Post",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 }  
 });  
 if (!response.ok) {  
 throw new Error('Failed to fetch user data');  
 }  
 const data = await response.json();  
 // console.log(data.firstLogin)  
 // console.log(typeof(data.firstLogin))  
 return data.firstLogin  
 } catch (error) {  
 console.error("Error fetching user data:", error);  
 }  
 };  
  
 const updateUserData = async () => {  
 try {  
 const response = await fetch(`${host}auth/${user}/updateuserdata`, {  
 method: "Post",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 }  
 });  
 if (!response.ok) {  
 throw new Error('Failed to fetch user data');  
 }  
 const data = await response.json();  
 // console.log(data.firstLogin)  
 // console.log(typeof(data.firstLogin))  
 return data.firstLogin  
 } catch (error) {  
 console.error("Error fetching user data:", error);  
 }  
 }  
  
 // Add these functions to your UserState component  
  
 const fetchNotifications = async () => {  
 try {  
 const response = await fetch(`${host}notification/getnotifications`, {  
 method: 'GET',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': cookies.get("auth-token")  
 }  
 });  
 const data = await response.json();  
 if (data.success) {  
 setNotifications(data.notifications);  
 setUnreadCount(data.unreadCount)  
 }  
 } catch (error) {  
 console.error('Error fetching notifications:', error);  
 }  
 };  
  
  
 useEffect(() => {  
 setUser(cookies.get("user"))  
 setToken(cookies.get("auth-token"))  
 fetchUserData(cookies.get("user"), cookies.get("auth-token"))  
 fetchNotifications(cookies.get("auth-token"))  
 loadProfilePicture()  
 }, [user, token]);  
  
 return (  
 <UserContext.Provider value={{ host, user, token,userProfile, userData, notifications, unreadCount, setUser,loadProfilePicture, setUserProfile, setUserData, setNotifications, setUnreadCount, fetchUserData, fetchFirstLogin, fetchNotifications }}>  
 {props.children}  
 </UserContext.Provider>  
 )  
}  
  
export default UserState

# AlumniConnect\src\css\AddStudent.css

.add-student-container {  
 max-width: 1200px;  
 margin: 2rem auto;  
 padding: 0 1rem;  
}  
  
.tab-buttons {  
 display: flex;  
 background-color: #f8f9fa;  
 border-radius: 8px;  
 box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);  
}  
  
.tab-button {  
 flex: 1;  
 padding: 0.5rem;  
 border: 2px solid #f7dfff;  
 margin-left: 10px;  
 border-radius: 6px;  
 background-color: #fff;  
 color: #333;  
 cursor: pointer;  
 font-size: 0.9rem;  
 font-weight: 500;  
 transition: all 0.3s ease;  
 box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);  
}  
  
.tab-button:hover {  
 background-color: #f7dfff;  
 transform: translateY(-1px);  
}  
  
.tab-button.active {  
 background-color: #f7dfff;  
 color: #333;  
 box-shadow: 0 2px 4px rgba(247, 223, 255, 0.2);  
}  
  
.error-message {  
 background-color: #ffebee;  
 color: #c62828;  
 padding: 1rem;  
 border-radius: 4px;  
 margin-bottom: 1rem;  
 border-left: 4px solid #c62828;  
}  
  
.success-message {  
 background-color: #e8f5e9;  
 color: #2e7d32;  
 padding: 1rem;  
 border-radius: 4px;  
 margin-bottom: 1rem;  
 border-left: 4px solid #2e7d32;  
}  
  
.single-student-form,  
.bulk-upload-form {  
 background-color: white;  
 padding: 2rem;  
 border-radius: 8px;  
 box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);  
}  
  
.form-group {  
 margin-bottom: 1.5rem;  
 position: relative;  
}  
  
.form-group input,  
.form-group select {  
 width: 100%;  
 padding: 0.75rem;  
 border: 1px solid #ddd;  
 border-radius: 4px;  
 font-size: 1rem;  
 transition: all 0.3s ease;  
}  
  
.form-group input:focus,  
.form-group select:focus {  
 outline: none;  
 border-color: #007bff;  
 box-shadow: 0 0 0 2px rgba(0, 123, 255, 0.25);  
}  
  
.form-group input.error,  
.form-group select.error {  
 border-color: #c62828;  
}  
  
.error-text {  
 color: #c62828;  
 font-size: 0.875rem;  
 margin-top: 0.25rem;  
}  
  
.submit-btn {  
 width: 100%;  
 padding: 1rem;  
 background-color: #f7dfff;  
 color: #333;  
 border: none;  
 border-radius: 4px;  
 font-size: 1rem;  
 font-weight: 500;  
 cursor: pointer;  
 transition: all 0.3s ease;  
}  
  
.submit-button:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.submit-btn:disabled {  
 background-color: #ccc;  
 cursor: not-allowed;  
 transform: none;  
}  
  
.file-upload-container {  
 text-align: center;  
 padding: 3rem 2rem;  
 border: 2px dashed #ddd;  
 border-radius: 8px;  
 margin-top: 1rem;  
 background-color: #f8f9fa;  
 transition: all 0.3s ease;  
}  
  
.file-upload-container:hover {  
 border-color: #007bff;  
 background-color: #f0f7ff;  
}  
  
.file-input {  
 display: none;  
}  
  
.file-label {  
 display: inline-block;  
 padding: 1rem 2rem;  
 background-color: #007bff;  
 color: white;  
 border-radius: 4px;  
 cursor: pointer;  
 font-weight: 500;  
 transition: all 0.3s ease;  
}  
  
.file-label:hover {  
 background-color: #0056b3;  
 transform: translateY(-1px);  
}  
  
.file-help {  
 margin-top: 1rem;  
 color: #666;  
 font-size: 0.9rem;  
}  
  
.json-format-help {  
 margin-top: 2rem;  
 text-align: left;  
 background-color: #f8f9fa;  
 padding: 1.5rem;  
 border-radius: 8px;  
 border: 1px solid #ddd;  
}  
  
.json-format-help h3 {  
 margin-bottom: 1rem;  
 color: #333;  
 font-size: 1.1rem;  
}  
  
.json-format-help pre {  
 background-color: #f1f1f1;  
 padding: 1rem;  
 border-radius: 4px;  
 overflow-x: auto;  
 font-size: 0.875rem;  
 border: 1px solid #ddd;  
}  
  
@media (max-width: 768px) {  
 .add-student-container {  
 margin: 1rem auto;  
 }  
  
 .tab-buttons {  
 flex-direction: column;  
 padding: 0.5rem;  
 }  
  
 .single-student-form,  
 .bulk-upload-form {  
 padding: 1rem;  
 }  
  
 .file-upload-container {  
 padding: 2rem 1rem;  
 }  
}

# AlumniConnect\src\css\Application.css

.appln-wrapper {  
 margin-top: 10px;  
 width: 100%;  
}  
  
.appln {  
 padding: 10px 0px;  
 border: 1px solid #e0e0e0;  
 border-radius: 8px;  
 background-color: white;  
 box-shadow: 0 2px 4px rgba(0,0,0,0.1);  
 transition: all 0.3s ease;  
 cursor: pointer;  
 user-select: none;  
 width: 100%;  
 position: relative;  
}  
  
.appln:hover {  
 background-color: #f8f9fa;  
}  
  
.app\_name {  
 display: flex;  
 justify-content: space-between;  
 align-items: center;  
 font-weight: 500;  
 /\* margin-bottom: 10px; \*/  
}  
  
.app\_name span {  
 color: #333;  
 /\* font-size: 1.1em; \*/  
}  
  
.status {  
 color: #333;  
 margin: 5px 0;  
}  
  
.date {  
 color: #333;  
 /\* font-size: 0.9em; \*/  
 /\* margin-bottom: 15px; \*/  
}  
  
.appln::after {  
 content: '';  
 position: absolute;  
 bottom: 0;  
 left: 0;  
 right: 0;  
 width: 100%;  
 height: 2px;  
 background-color: #e0e0e0;  
 opacity: 0;  
 transition: opacity 0.3s ease;  
 z-index: 2;  
}  
  
.appln.expanded::after {  
 opacity: 1;  
}  
  
.tracking-container {  
 max-height: 0;  
 overflow: hidden;  
 transition: all 0.3s ease-out;  
 background-color: #f8f9fa;  
 border-radius: 0 0 8px 8px;  
 border: 1px solid #e0e0e0;  
 border-top: none;  
 margin-top: -1px;  
 opacity: 0;  
 transform: translateY(-10px);  
 width: 100%;  
 padding: 0 15px;  
 box-sizing: border-box;  
}  
  
.tracking-container.expanded {  
 max-height: fit-content;  
 width: 100%;  
 /\* padding: 15px; \*/  
 opacity: 1;  
 transform: translateY(0);  
 box-sizing: border-box;  
}  
  
.tracking-timeline {  
 position: relative;  
 padding-top: 10px;  
 width: 100%;  
}  
  
.tracking-step {  
 display: flex;  
 align-items: flex-start;  
 margin-bottom: 20px;  
 position: relative;  
 width: 100%;  
}  
  
.step-icon {  
 width: 30px;  
 height: 30px;  
 border-radius: 50%;  
 background-color: #fff;  
 border: 2px solid #e0e0e0;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 font-size: 16px;  
 color: #666;  
 z-index: 1;  
 flex-shrink: 0;  
}  
  
.step-icon.completed {  
 background-color: #f7dfff;  
 border-color: #f7dfff;  
 color: #333;  
}  
  
.step-content {  
 margin-left: 15px;  
 flex-grow: 1;  
 width: calc(100% - 45px);  
}  
  
.step-title {  
 font-weight: 500;  
 color: #333;  
 margin-bottom: 5px;  
}  
  
.step-status {  
 font-size: 0.9em;  
 color: #666;  
 text-transform: capitalize;  
 margin-bottom: 5px;  
}  
  
.step-date {  
 font-size: 0.8em;  
 color: #666;  
 margin-bottom: 5px;  
}  
  
.step-remarks {  
 background-color: #f8f9fa;  
 padding: 10px;  
 border-radius: 4px;  
 margin-top: 5px;  
}  
  
.remarks-label {  
 font-weight: 500;  
 color: #333;  
 margin-bottom: 5px;  
}  
  
.remarks-text {  
 color: #666;  
 font-size: 0.9em;  
 line-height: 1.4;  
}  
  
.step-checks {  
 margin-top: 10px;  
 background-color: #fff;  
 border: 1px solid #e0e0e0;  
 border-radius: 4px;  
 padding: 10px;  
}  
  
.check-item {  
 display: flex;  
 align-items: center;  
 gap: 8px;  
 margin-bottom: 8px;  
 flex-wrap: wrap;  
}  
  
.check-status {  
 width: 20px;  
 height: 20px;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 margin-right: 10px;  
 font-size: 14px;  
}  
  
.check-status.completed {  
 color: #f7dfff;  
}  
  
.check-status.pending {  
 color: #dc3545;  
}  
  
.check-name {  
 color: #333;  
 font-size: 0.9em;  
}  
  
.step-connector {  
 position: absolute;  
 left: 15px;  
 top: 30px;  
 bottom: -20px;  
 width: 2px;  
 background-color: #e0e0e0;  
}  
  
.step-connector.completed {  
 background-color: #f7dfff;  
}  
  
.tracking-step:last-child .step-connector {  
 display: none;  
}  
  
.receipt-upload {  
 margin-top: 8px;  
 display: flex;  
 align-items: center;  
 gap: 8px;  
}  
  
.receipt-input {  
 display: none;  
}  
  
.upload-label {  
 background-color: #f7dfff;  
 color: #333;  
 padding: 6px 12px;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 12px;  
 transition: all 0.3s ease;  
}  
  
.upload-label:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.upload-label.document-upload {  
 background-color: #f7dfff;  
}  
  
.upload-label.document-upload:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.cancel-form-section {  
 margin-top: 20px;  
 padding: 5px;  
 border-radius: 6px;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 width: 100%;  
 box-sizing: border-box;  
}  
  
.cancel-button {  
 background-color: #f7dfff;  
 color: #333;  
 padding: 10px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 14px;  
 font-weight: 500;  
 transition: all 0.3s ease;  
 width: auto;  
 min-width: 120px;  
}  
  
.cancel-button:hover {  
 background-color: #b690c4;  
 color: white;  
}

# AlumniConnect\src\css\CheckStatus.css

/\* Root variables \*/  
:root {  
 --button-color: #f7dfff;  
 --button-hover: #b690c4;  
}  
  
/\* Universal box-sizing  
\*, \*::before, \*::after {  
 box-sizing: border-box;  
 margin: 0;  
 padding: 0;  
} \*/  
  
.form-container {  
 width: 100%;  
 padding: 20px;  
}  
  
.checkastatusHeader {  
 width: 100%;  
 display: flex;  
 justify-content: space-between;  
 align-items: center;  
 margin-bottom: 20px;  
}  
  
.form-header {  
 color: #1a237e;  
 font-size: 28px;  
 font-weight: 600;  
 margin: 0;  
}  
  
.filter {  
 display: flex;  
 align-items: center;  
}  
  
.status\_filter {  
 padding: 8px 16px;  
 font-size: 16px;  
 border: 1px solid #e0e0e0;  
 border-radius: 4px;  
 background-color: white;  
 color: #333;  
 cursor: pointer;  
 min-width: 150px;  
}  
  
.status\_filter:focus {  
 outline: none;  
 border-color: var(--button-color);  
}  
  
.check\_status {  
 width: 100%;  
 background-color: white;  
 border-radius: 8px;  
 box-shadow: 0 2px 4px rgba(0,0,0,0.05);  
  
}  
  
.appln {  
 width: 100%;  
 height: 60px;  
 background-color: #ffffff;  
 padding: 0 0px;  
 display: flex;  
 justify-content: space-between;  
 align-items: center;  
 flex-direction: row;  
 flex-wrap: nowrap;  
}  
  
.app\_name {  
 width: 55%;  
 height: 100%;  
 margin-left: 30px;  
 font-size: 25px;  
 font-weight: 700;  
 font-family: 'Inter', sans-serif;  
 line-height: 60px;  
 overflow: hidden;  
 text-align: start;  
 color: #0d016a;  
}  
  
.status {  
 width: 20%;  
 /\* height: 100%; \*/  
 font-size: 25px;  
 font-weight: 700;  
 line-height: 40px;  
 text-align: center;  
 border-left: 1px rgb(163, 163, 163) solid;  
 border-right: 1px rgb(163, 163, 163) solid;  
}  
  
.date {  
 width: 20%;  
 /\* height: 100%; \*/  
 font-size: 25px;  
 font-weight: 700;  
 line-height: 40px;  
 text-align: center;  
}  
  
.check\_status .appln:nth-child(even){  
 background-color: #f7dfff;  
}  
  
.form-list-header {  
 width: 100%;  
 height: fit-content;  
 /\* background-color: #ffffff; \*/  
 padding: 0 10px;  
 padding-bottom: 10px;  
 display: flex;  
 justify-content: start;  
 align-items: center;  
 flex-direction: row;  
 flex-wrap: nowrap;  
 /\* border: 1px solid #e0e0e0; \*/  
 /\* border-radius: 8px; \*/  
 /\* margin-bottom: 10px; \*/  
 font-weight: 700;  
 font-size: 25px;  
 line-height: 40px;  
 color: #0d016a;  
 font-family: 'Inter', sans-serif;  
 box-shadow: 0 2px 4px rgba(0,0,0,0.05);  
}  
  
/\* Teacher View Headers \*/  
.teacher-header {  
 width: 100%;  
 height: 50px;  
 display: flex;  
 align-items: center;  
 background-color: #f8f9fa;  
 border-bottom: 1px solid #e0e0e0;  
 font-size: 15px;  
 font-weight: 600;  
 color: #333;  
}  
  
.teacher-header-name {  
 width: 35%;  
 padding: 0 20px;  
 display: flex;  
 align-items: center;  
   
}  
  
.teacher-header-status {  
 width: 20%;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 padding: 0 10px;  
}  
  
.teacher-header-actions {  
 width: 20%;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
}  
  
.teacher-header-date {  
 width: 25%;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
}  
  
/\* Teacher View Application Items \*/  
.teacher-appln {  
 width: 100%;  
 height: 50px;  
 display: flex;  
 align-items: center;  
 background-color: white;  
 border-bottom: 1px solid #e0e0e0;  
 transition: background-color 0.2s ease;  
}  
  
.teacher-appln:hover {  
 background-color: #f5f5f5;  
}  
  
.teacher-appln-name {  
 width: 35%;  
 padding: 0 20px;  
 display: flex;  
 align-items: center;  
 font-size: 14px;  
 color: #333;  
 overflow: hidden;  
 text-overflow: ellipsis;  
 white-space: nowrap;  
   
}  
  
.teacher-appln-status {  
 width: 20%;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 padding: 0 10px;  
}  
  
.teacher-appln-actions {  
 width: 20%;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
}  
  
.teacher-appln-date {  
 width: 25%;  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 color: #666;  
 font-size: 14px;  
}  
  
.check\_status .teacher-appln:nth-child(even) {  
 background-color: #f8f9fa;  
}  
  
.header-name {  
 width: 55%;  
 height: 100%;  
 margin-left: 30px;  
}  
  
.header-status {  
 width: 20%;  
 height: 100%;  
 border-left: 1px rgb(163, 163, 163) solid;  
 border-right: 1px rgb(163, 163, 163) solid;  
 text-align: center;  
}  
  
.header-date {  
 width: 20%;  
 height: 100%;  
 text-align: center;  
 margin-left: 7px;  
}  
  
/\* Teacher View Buttons \*/  
.approve-btn, .notify-btn {  
 padding: 5px 15px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 14px;  
 font-weight: 500;  
 transition: all 0.3s ease;  
 background-color: #f7dfff;  
 color: #333;  
}  
  
.approve-btn:hover, .notify-btn:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.lc-details {  
 padding: 0px;  
 background-color: #ffffff;  
 border-radius: 8px;  
 width: 100%;  
}  
  
.lc-section {  
 margin-bottom: 12px;  
 padding: 8px;  
 background-color: #f8f9fa;  
 border-radius: 6px;  
 width: 100%;  
}  
  
.lc-section h3 {  
 color: #333;  
 margin-bottom: 8px;  
 font-size: 18px;  
 font-weight: 600;  
}  
  
.lc-section p {  
 margin: 0;  
 color: #666;  
 font-size: 14px;  
 display: flex;  
 justify-content: flex-start;  
 width: 100%;  
 padding: 2px 0;  
}  
  
.lc-section strong {  
 color: #333;  
 font-weight: 500;  
 margin-right: 0;  
 min-width: 180px;  
}  
  
.tracking-container.expanded {  
 width: 100%;  
 margin: 0px 0;  
 background-color: #ffffff;  
 border-radius: 8px;  
 box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);  
}  
  
/\* CheckStatus.css \*/  
.teacher-header-status {  
 flex: 1;  
 text-align: center;  
}  
  
.teacher-appln-status {  
 flex: 1;  
 text-align: center;  
 padding: 5px;  
 border-radius: 4px;  
 margin: 0 5px;  
}  
  
.dropdown-container {  
 position: relative;  
}  
  
.dropdown-toggle {  
 background-color: #f7dfff;  
 color: #333;  
 padding: 6px 16px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 14px;  
 min-width: 100px;  
 transition: all 0.3s ease;  
}  
  
.dropdown-toggle:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.dropdown-menu {  
 position: absolute;  
 top: 100%;  
 right: 0;  
 background-color: white;  
 min-width: 160px;  
 box-shadow: 0 2px 8px rgba(0,0,0,0.15);  
 border-radius: 4px;  
 margin-top: 4px;  
 z-index: 1000;  
}  
  
.dropdown-item {  
 padding: 8px 16px;  
 font-size: 14px;  
 color: #333;  
 background: none;  
 border: none;  
 width: 100%;  
 text-align: left;  
 cursor: pointer;  
 transition: all 0.3s ease;  
}  
  
.dropdown-item:hover {  
 background-color: #f7dfff;  
}  
  
.confirmation-popup {  
 position: fixed;  
 top: 0;  
 left: 0;  
 width: 100%;  
 height: 100%;  
 background-color: rgba(0, 0, 0, 0.5);  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 z-index: 2000;  
 animation: fadeIn 0.3s ease-in-out;  
}  
  
.confirmation-content {  
 background-color: white;  
 padding: 24px;  
 border-radius: 12px;  
 box-shadow: 0 8px 24px rgba(0, 0, 0, 0.2);  
 text-align: center;  
 max-width: 400px;  
 width: 90%;  
 animation: slideUp 0.3s ease-in-out;  
}  
  
@keyframes slideUp {  
 from {  
 transform: translateY(20px);  
 opacity: 0;  
 }  
 to {  
 transform: translateY(0);  
 opacity: 1;  
 }  
}  
  
.confirmation-content p {  
 margin-bottom: 24px;  
 font-size: 16px;  
 color: #333;  
 line-height: 1.5;  
}  
  
.confirmation-content button {  
 background-color: #f7dfff;  
 color: #333;  
 padding: 10px 24px;  
 border: none;  
 border-radius: 6px;  
 cursor: pointer;  
 font-size: 14px;  
 font-weight: 500;  
 transition: all 0.3s ease;  
 min-width: 100px;  
}  
  
.confirmation-content button:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.confirmation-content button:active {  
 transform: translateY(0);  
}  
  
.teacher-appln-actions {  
 width: 120px;  
 text-align: center;  
 padding: 0 8px;  
}  
  
/\* Status colors \*/  
.status-pending,  
.status-verified,  
.status-completed {  
 width: 100%;  
 padding: 6px 12px;  
 border-radius: 4px;  
 font-size: 13px;  
 font-weight: 500;  
 text-align: center;  
}  
  
.status-pending {  
 /\* background-color: #fff3cd; \*/  
 color: #856404;  
}  
  
.status-verified {  
 /\* background-color: #d4edda; \*/  
 color: #155724;  
}  
  
.status-completed {  
 /\* background-color: #cce5ff; \*/  
 color: #004085;  
}  
  
/\* Responsive adjustments \*/  
@media (max-width: 768px) {  
 .checkastatusHeader {  
 flex-direction: column;  
 align-items: flex-start;  
 gap: 16px;  
 }  
  
 .filter {  
 width: 100%;  
 }  
  
 .status\_filter {  
 width: 100%;  
 }  
  
 .teacher-header,  
 .teacher-appln {  
 padding: 10px;  
 }  
  
 .teacher-header-name,  
 .teacher-appln-name {  
 width: 30%;  
 padding: 0 10px;  
 }  
}  
  
.document-buttons {  
 display: flex;  
 gap: 10px;  
 margin-top: 10px;  
}  
  
.check-button {  
 background-color: #f7dfff;  
 color: #333;  
 padding: 8px 16px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 14px;  
 transition: all 0.3s ease;  
 white-space: nowrap;  
}  
  
.check-button:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.lc-section {  
 margin-bottom: 20px;  
}  
  
.lc-section h3 {  
 color: #333;  
 margin-bottom: 10px;  
 font-size: 16px;  
}

# AlumniConnect\src\css\Forms.css

/\*//////////////////////////// Dashboard ////////////////////////////\*/  
.mainBodyContainer {  
 padding: 10px;  
 max-width: 1200px;  
 width: 100%;  
 height: 100%;  
 /\* box-shadow: 0 0 5px 5px rgba(0, 0, 0, 0.1); \*/  
 display: flex;  
 justify-content: space-around;  
 flex-wrap: wrap;  
 align-items: center;  
 /\* border-radius: 25px; \*/  
 background-color: #fff;  
 box-sizing: border-box;   
 overflow-y: scroll;  
 z-index: 12;  
}  
.doc\_box{  
 margin:5px 10px;  
 padding: 20px;  
 border: 2px solid #efd0ff;  
 border-radius: 20px;  
 box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);  
 width: 200px;  
 height: 225px;  
 display: flex;  
 justify-content: space-around;  
 flex-wrap: wrap;  
}  
.doc\_box:hover{  
 box-shadow: 0 0 2px 2px #b690c4;  
}  
.doc\_img{  
 --length:150px;  
 width: var(--length);  
 height: var(--length);  
 margin-bottom: 15px;  
}  
.doc\_text{  
 --text\_height:fit-content;  
 height: var(--text\_height);  
 font-size: 20px;  
 font-weight: 700;  
 font-family:'hanuman', sans-serif;  
 line-height: var(--text\_height);  
 text-align: center;  
}  
  
@media (max-width : 350px) {  
 .mainBodyContainer{  
 min-width: fit-content;  
 background-color: transparent;  
 box-shadow: none;  
 }  
 .doc\_img{  
 display: none;  
 }  
 .doc\_text{  
 font-size: 15px;  
 font-weight: 600;  
 }  
 .doc\_box{  
 height: fit-content;  
 background:#f7e9f8 ;  
 }  
}

# AlumniConnect\src\css\Header.css

/\*//////////////////////////// header section ////////////////////////////\*/  
:root{  
 --header\_height:75px;  
 --img\_height:60px;  
}  
.header {  
 padding: 0px;  
 margin: 0px;  
 height: var(--header\_height);   
 width: 100%;  
 display: flex;  
 justify-content: center;  
 align-items: stretch;  
 background-color: #f7dfff;  
 box-sizing: border-box;  
 z-index: 10;  
 box-shadow: 0 0 5px 5px rgba(0, 0, 0, 0.1);  
}  
.collegeLogo {  
 margin: auto;  
 margin-left: 15px;  
 width: var(--img\_height);  
 height: var(--img\_height);  
}  
.menuIcon{  
 margin: auto;  
 margin-left: 15px;  
 width: var(--img\_height);  
 height: var(--img\_height);  
}  
.image{  
 width: 100%;  
 height: 100%;  
 object-fit: contain;  
}  
.clg\_name{  
 width: 100%;  
 height: var(--header\_height);  
 font-size: 30px ;  
 font-weight: 700;  
 font-family: 'Inter', sans-serif;  
 line-height: var(--header\_height);  
 overflow: hidden;  
 text-align: center;   
 letter-spacing: 5px;  
}

# AlumniConnect\src\css\LC.css

.form-input label {  
 text-transform: capitalize;  
}  
  
.form-input select option {  
 text-transform: capitalize;  
}

# AlumniConnect\src\css\Login.css

body{  
 background: #ffeaff;  
}  
.loginContainer{  
 align-self: center;  
 width: 900px;  
 height: fit-content;  
 position: absolute;  
 top: 50%;  
 left: 50%;  
 transform: translate(-50%, -50%);  
 box-shadow: 0 0 5px 5px rgba(0, 0, 0, 0.1);  
 display: flex;  
 flex: 1 1 100%;  
 align-items: stretch;  
 border-radius: 25px;  
 overflow: hidden;  
}  
.forgetContainer{  
 width: 405px;  
 background: #F4ECFF;  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 position: absolute;  
 top: 50%;  
 left: 50%;  
 transform: translate(-50%, -50%);  
 box-shadow: 0 0 5px 5px rgba(0, 0, 0, 0.1);  
 border-radius: 25px;  
}  
  
.Logo\_container{  
 width: 55%;  
 /\* height: 100%; \*/  
 background: #ffffff;  
}  
.image{  
 width: 100%;  
 height: 100%;  
 object-fit: contain;  
}  
.Login\_window{  
 width: 45%;  
 background: #F4ECFF;  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
}  
.student\_Logo{  
 margin-top: 20px;  
 width: 75px;  
 height: 75px;  
}  
.form{  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
}  
.login\_txt{  
 margin-top: 15px;  
 /\* margin-bottom: -15px; \*/  
 color: black;  
 opacity: 0.75;  
 font-size: 30px;  
 font-family: 'Inter', sans-serif;  
 font-weight: 700;  
 line-height: 45px;  
 word-wrap: break-word  
}  
.txt{  
 /\* margin-bottom: 30px; \*/  
 color: black;  
 opacity: 0.75;  
 font-size: 14px;  
 font-family: 'Inter', sans-serif;  
 font-weight: 700;  
 word-wrap: break-word  
}  
.input{  
 /\* margin-top: 30px; \*/  
 width: 250px;   
 height: 40px;   
 background: white;   
 border-radius: 5px;   
 border: 1px rgb(163, 163, 163) solid;  
 text-align: center;  
 font-size: large;  
 font-family: 'Inter', sans-serif;  
}  
#pass{  
 position: absolute;  
 left: 0px;  
 top: 0px;  
}  
.text{  
 color: black;  
 opacity: 0.75;  
 font-size: 12px;  
 font-family: 'Inter', sans-serif;  
 font-weight: 700;  
 word-wrap: break-word  
}  
.passfild{  
 position: relative;  
 margin-top :20px ;  
 height: 40px;  
}  
.eyeicon{  
 width: 23px;  
 height: 23px;  
 position: absolute;  
 right: 10px;  
 top: 50%;  
 transform: translateY(-50%);  
 background: none;  
 border: none;  
 cursor: pointer;  
 font-size: 18px;  
}  
.userType select{  
 width: 250px;  
 /\* margin-inline-end: 30px; \*/  
 /\* width: 75%; \*/  
 margin-top: 20px;  
 padding: 10px;  
 text-align: center;  
 font-size: large;  
 font-family: 'Inter', sans-serif;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 display: inline;  
}  
.userType select:focus, .input:focus{  
 border-color: #b690c4;  
 outline: none;  
 box-shadow: 0 0 5px #b690c4;  
 box-sizing: border-box  
}  
#checkbox{  
 margin-top: 20px;  
 margin-left: 0px;  
 width: 250px;  
 display: flex;  
 flex-direction: row;  
 justify-content: space-between;  
 align-items: center;  
 flex-wrap: nowrap;  
}  
.Login{  
 margin-top: 30px;  
 margin-bottom: 10px;  
 width: 200px;   
 height: 50px;   
 background: #e9bbf8;   
 border: 0px;  
 border-radius: 25px;  
}  
.otp-button{  
 /\* margin-top: 30px; \*/  
 /\* margin-bottom: 10px; \*/  
 margin-left: 5px;  
 width: 80px;   
 height: 40px;   
 background: #e9bbf8;   
 border: 0px;  
 border-radius: 5px;  
  
}  
.Login:hover{  
 box-shadow: 0 0 2px 2px #b690c4;  
}  
.otp-button:hover{  
 box-shadow: 0 0 2px 2px #b690c4;  
}  
  
input[type="number"].no-arrow::-webkit-inner-spin-button,  
input[type="number"].no-arrow::-webkit-outer-spin-button {  
 -webkit-appearance: none;  
 margin: 0;  
}  
  
/\* For Firefox \*/  
input[type="number"].no-arrows {  
 -moz-appearance: textfield;  
}  
  
@media (max-width: 900px) {   
 /\* body{  
 background:#ffffff;  
 } \*/  
 .loginContainer{  
 width: 405px;  
 height: 500px;  
 }  
 .content-to-hide {  
 display: none;  
 }  
 .Login\_window{  
 width: 405px;  
 }  
}  
  
/\* @media (max-width: 300px) {  
 .content-to-hide {  
 display: none;  
 }  
 .Login\_window{  
 width: 200px;  
 }  
} \*/

# AlumniConnect\src\css\Lor.css

/\*//////////////////////////// Lor ////////////////////////////\*/  
.form-container {  
 width: 100%;  
 height:100%;  
 background-color: #fff;  
 /\* margin: 20px auto; \*/  
 padding: 20px 20px;  
 /\* border-radius: 10px; \*/  
 box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);  
 font-family: 'Inter', sans-serif;  
 color: #0d016a;  
 overflow-y: scroll;  
 box-sizing: border-box;  
}  
  
.form-header {  
 font-size: 35px;  
 font-weight: 700;  
}  
  
.lorform{  
 width: 95%;  
 margin-top: 10px;  
 display: flex;  
 flex-direction:column;  
 align-items: start;  
}  
  
.form-input {  
 width: 300px;  
 margin-right: 10px;  
 margin-bottom: 15px;  
}  
  
.form-input label {  
 /\* display: block; \*/  
 margin-bottom: 5px;  
 font-weight: 600;  
}  
  
.form-input input,  
.form-input select {  
 margin-inline-end: 30px;  
 /\* width: 75%; \*/  
 margin-bottom: 10px;  
 padding: 10px;  
 font-size: 16px;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
 display: inline;  
}  
.form-input select{  
 width: 300px;  
}  
  
.form-input input:focus,  
.form-input select:focus {  
 border-color: #b690c4;  
 outline: none;  
 box-shadow: 0 0 5px #b690c4;  
 box-sizing: border-box  
}  
  
.submit-button {  
 width: 100%;  
 padding: 10px 30px;  
 background-color: #f7dfff;  
 color: #000;  
 font-size: 18px;  
 font-weight: bold;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 text-align: center;  
}  
  
.submit-button:hover {  
 background-color: #b690c4;  
}  
  
.nameContainer{  
 display: flex;  
 justify-content: flex-start;  
 flex-wrap: wrap;  
}  
  
.fieldContainer{  
 min-width: 500px;  
 display: flex;  
 flex-direction: column;  
 justify-content: flex-start;  
 flex-wrap: wrap;  
}  
  
.innerfieldContainer{  
 /\* min-width: 500px; \*/  
 display: flex;  
 flex-direction: row;  
 justify-content: flex-start;  
 flex-wrap: wrap;  
}  
  
.button{  
 margin: auto;  
}  
  
/\* .checkbox{  
 display: flex;  
 flex-direction: row;  
} \*/  
  
.name{  
 width: fit-content;  
}  
input[type="number"].no-arrows::-webkit-inner-spin-button,  
input[type="number"].no-arrows::-webkit-outer-spin-button {  
 -webkit-appearance: none;  
 margin: 0;  
}  
  
/\* For Firefox \*/  
input[type="number"].no-arrows {  
 -moz-appearance: textfield;  
}  
.no-arrows{  
 width: 300px;  
 box-sizing: border-box;  
}  
  
.instructions{  
 /\* margin-left: 15px; \*/  
 width:fit-content  
}  
.checkBox{  
 margin-top: 10px;  
 display:inline;  
 width: 300px;  
}  
  
.checkbox{  
 display:inline;  
 width: 10px;  
}  
.staffSelection{  
 width: fit-content;  
}  
  
.image-upload-container {  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 margin-bottom: 20px;  
}  
  
.profile-image-wrapper {  
 position: relative;  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 gap: 10px;  
}  
  
.profile-image {  
 width: 150px;  
 height: 150px;  
 border-radius: 50%;  
 object-fit: cover;  
 border: 3px solid #f7dfff;  
 box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);  
}  
  
.upload-button {  
 background-color: #f7dfff;  
 color: #333;  
 padding: 8px 20px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 14px;  
 font-weight: 500;  
 transition: all 0.3s ease;  
}  
  
.upload-button:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.upload-button:disabled {  
 background-color: #e0e0e0;  
 cursor: not-allowed;  
 color: #666;  
}  
  
#lastclg{  
 max-width: 575px;  
 display: block;  
 width: 90vw  
}  
  
.noticeTitle{  
 width:500px;  
 padding: 10px;  
 border-radius: 5px;  
 margin-bottom: 10px;  
 font-size: 24px;  
 border: 1px solid #ccc;  
 border-color: #b690c4;  
 box-shadow: 0 0 0px #b690c4;  
}  
  
.noticeContaint{  
 width:500px;  
 height:400px;  
 resize: none;  
 overflow-y: scroll;  
 border-radius: 5px;  
 margin-bottom: 10px;  
 padding: 10px;  
 font-size: 20px;  
 border: 1px solid #ccc;  
 border-color: #b690c4;  
 box-shadow: 0 0 0px #b690c4;  
 box-sizing: border-box;  
}  
  
.form-button{  
 margin-inline:auto;   
 margin-top : 10px;   
 margin-bottom : 10px;  
 box-sizing: border-box;  
}

# AlumniConnect\src\css\MainBody.css

/\*//////////////////////////// Main body ////////////////////////////\*/  
.main\_body {  
 width: 100%;  
 min-height: calc(100vh - 78px);  
 height: calc(100vh - 78px);  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 flex-direction: column;  
 /\* padding: 20px; \*/  
 box-sizing: border-box;  
 overflow: hidden;  
}

# AlumniConnect\src\css\Navbar.css

/\*//////////////////////////// navigation ////////////////////////////\*/  
.nav{  
 width: 22vw;  
 min-width:fit-content;  
 min-height: 100vh;  
 height: 100vh;  
 background: rgb(255, 255, 255);  
 border: 2px #F4ECFF solid;  
 z-index: 1;  
 overflow-x: hidden;  
 transition: 0.5s;  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 align-self: stretch;  
 padding: 0px;  
 box-shadow: 0 0 5px 5px rgba(0, 0, 0, 0.1);  
}  
.stu-info{  
 padding: 10px;  
 margin-bottom: 40px;  
 height: 75px;  
 width: 100%;  
 background-color: #f7dfff;  
 box-shadow: 0 0 5px 5px rgba(0, 0, 0, 0.1);  
 /\* border: 2px black solid; \*/  
 display: flex;  
 align-items: center;  
 justify-content: center;  
 box-sizing: border-box;  
}  
.stu-img{  
 min-width:60px;  
 height: 60px;  
 border-radius: 30px;  
 overflow: hidden;  
 margin: 0px 0px;  
 padding: 0px;  
}  
.image{  
 width: 100%;;  
 height: 100%;  
 object-fit: contain;  
}  
.stu-name{  
 max-width: 200px;  
 margin:0px 10px;  
 height: 80px;  
 font-size: 25px ;  
 font-weight: 700;  
 font-family: 'Inter', sans-serif;  
 line-height: 80px;  
 overflow: hidden;  
}  
.nav\_container{  
 margin-bottom: 25px;  
 margin-inline: 40px;  
 height: 60px;  
 width: 250px;  
 border: 4px solid #f7dfff;  
 border-radius: 35px;  
 background-color: #ffffff;   
 display: flex;  
 align-items: center;  
 justify-content: center;  
}  
.nav\_container:hover, .nav\_container:focus {  
 background-color: #f7dfff;   
}  
.activeLink{  
 background-color: #f7dfff;   
}  
.img-container{  
 height: 35px;  
}  
.nav-item{  
 margin-left: 10px;  
 font-size: 20px ;  
 font-weight: 500;  
 font-family: 'Inter', sans-serif;  
 line-height: 30px;  
 opacity: 75%;  
}  
  
.showNav{  
 display : block;  
 position : fixed;  
 top : 0px;  
 left : 0px;  
 z-index : 11;  
 overflow-y : visible;  
 height : -webkit-fill-available;  
}  
.showNav > .stu-info{  
 padding: 0px;  
 justify-content: flex-start;  
}  
.showNav > .stu-info > .stu-img{  
 margin-left: 15px;  
}  
  
.hideNav{  
 display : none;  
}

# AlumniConnect\src\css\Notice.css

.notice{  
 margin: 20px;  
 width: 300px;  
 height: 300px;  
 display: flex;  
 flex-direction: column;  
 flex-wrap: nowrap;  
 /\* border: 1px solid #000; \*/  
 border-radius: 5px;  
}  
.notice-title{  
 padding: 10px;  
   
}  
.notice-containt{  
 margin: 10px;  
}  
  
.status-card-header {  
 padding: 15px 20px;  
 margin: 10px 10px 0px 10px;  
 display: flex;  
 justify-content: space-between;  
 align-items: center;  
 background-color: #ffffff;  
 box-shadow: 0 2px 4px rgba(0,0,0,0.1);  
 border: 1px solid #e0e0e0;  
 border-radius: 8px;  
 transition: background-color 0.3s ease;  
}  
.status-card.expanded{  
 border-radius: 8px 8px 0 0;  
}  
.status-card-header.expanded{  
 border-radius: 8px 8px 0 0;  
}  
  
.status-card-header:hover {  
 background-color: #f7f9fc;  
}  
  
.status-info {  
 display: flex;  
 justify-content: space-between;  
 align-items: center;  
 flex-grow: 1;  
}  
  
.status-title {  
 display: flex;  
 align-items: center;  
 gap: 10px;  
 margin-bottom: 0;  
}  
  
.department {  
 font-weight: 600;  
 color: #333;  
}  
  
.status-date {  
 font-size: 0.85rem;  
 color: #666;  
 margin-right: 10px;  
}  
  
.new-badge {  
 background-color: #efd0ff;   
 color: #000000;  
 padding: 3px 10px;  
 border-radius: 12px;  
 font-size: 0.8rem;  
 font-weight: 500;  
 margin-left: 10px; /\* Space to the left of the badge \*/  
 align-self: center; /\* Center it vertically \*/  
 margin-right: 10px; /\* Space to the right of the badge \*/  
}  
  
.status-card-content {  
 padding: 15px 20px;  
 margin: 0px 10px;  
 background-color: #ffffff; /\* White background for the content box \*/  
 border: 1px solid #e0e0e0; /\* Light border around the box \*/  
 box-sizing: border-box;  
 border-radius: 0 0 8px 8px; /\* Rounded corners only on the bottom \*/  
 box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1); /\* Subtle shadow for depth \*/  
 max-width: 100%; /\* Ensure it does not exceed the container width \*/  
 overflow: auto; /\* Add scroll if content overflows \*/  
}  
  
.message-content {  
 font-size: 16px; /\* Font size for the message \*/  
 line-height: 1.5; /\* Line height for better readability \*/  
 box-sizing: border-box;  
 color: #333; /\* Text color \*/  
 margin: 0; /\* Remove default margin \*/  
 padding: 0px 0; /\* Padding for the message \*/  
 font-family: 'Inter', sans-serif; /\* Use the same font as other components \*/  
 font-weight: 500; /\* Medium font weight for better readability \*/  
 word-wrap: break-word; /\* Break long words to prevent overflow \*/  
}  
  
.notice-button {  
 background-color: #f7dfff;  
 color: #333;  
 padding: 8px 16px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 14px;  
 transition: all 0.3s ease;  
}  
  
.notice-button:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.action-button {  
 background-color: #f7dfff;  
 color: #333;  
 padding: 6px 12px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 13px;  
 transition: all 0.3s ease;  
}  
  
.action-button:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.notification-count{  
 background-color: #efd0ff;   
 color: #000000;  
 padding: 3px 10px;  
 border-radius: 12px;  
 font-size: 0.8rem;  
 font-weight: 500;  
 margin-left: 10px; /\* Space to the left of the badge \*/  
}

# AlumniConnect\src\css\NotificationModal.css

.modal-overlay {  
 position: fixed;  
 top: 0;  
 left: 0;  
 right: 0;  
 bottom: 0;  
 background: rgba(0, 0, 0, 0.5);  
 display: flex;  
 justify-content: center;  
 align-items: center;  
 z-index: 1000;  
}  
  
.modal-content {  
 background: white;  
 padding: 20px;  
 border-radius: 8px;  
 width: 400px;  
 box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);  
}  
  
.modal-content h2 {  
 margin: 0 0 20px 0;  
 text-align: center;  
 color: #333;  
}  
  
.modal-body {  
 margin-bottom: 20px;  
}  
  
.notification-title-input {  
 width: 100%;  
 padding: 10px;  
 margin-bottom: 10px; /\* Space between title and message \*/  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 font-size: 16px;  
 box-sizing: border-box;  
}  
  
.notification-textarea {  
 width: 100%;  
 height: 100px; /\* Adjust height as needed \*/  
 padding: 10px;  
 border: 1px solid #ccc;  
 border-radius: 4px;  
 font-size: 16px;  
 resize: none; /\* Prevent resizing \*/  
 box-sizing: border-box;  
}  
  
.modal-buttons {  
 display: flex;  
 justify-content: space-between;  
}  
  
.send-btn, .cancel-btn {  
 padding: 10px 15px;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 16px;  
 transition: all 0.3s ease;  
}  
  
.send-btn {  
 background-color: #f7dfff;  
 color: #333;  
}  
  
.send-btn:hover {  
 background-color: #b690c4;  
 color: white;  
}  
  
.cancel-btn {  
 background-color: #f7dfff;  
 color: #333;  
}  
  
.cancel-btn:hover {  
 background-color: #b690c4;  
 color: white;  
}

# AlumniConnect\src\css\Profile.css

.form-container {  
 width: 100%;  
 background-color: #fff;  
 padding: 20px;  
 box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);  
 font-family: 'Inter', sans-serif;  
 color: #0d016a;  
}  
  
.form-header {  
 font-size: 35px;  
 font-weight: 700;  
}  
  
.ProfileForm{  
 width: 95%;  
 margin-top: 10px;  
 display: flex;  
 flex-direction:column;  
 align-items: start;  
}  
  
.form-input {  
 margin-bottom: 15px;  
}  
  
.form-input label {  
 display: block;  
 margin-bottom: 5px;  
 font-weight: 600;  
}  
  
.form-input input,  
.form-input select {  
 width: 99%;  
 padding: 10px;  
 font-size: 16px;  
 border: 1px solid #ccc;  
 border-radius: 5px;  
}  
  
.form-input input:focus,  
.form-input select:focus {  
 border-color: #dc9efb;  
 outline: none;  
 box-shadow: 0 0 5px #dc9efb;  
}  
  
.submit-button {  
 width: fit-content;  
 padding: 10px 30px;  
 background-color: #f7dfff;  
 color: #000;  
 font-size: 18px;  
 font-weight: bold;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 text-align: center;  
 margin-inline: auto;  
}  
  
.submit-button:hover {  
 background-color: #b690c4;  
}  
  
.nameContainer{  
 display: flex;  
 flex-direction: row;  
 flex-wrap: nowrap;  
 align-items: flex-start;  
 justify-content: flex-start;  
}  
  
.image-upload-container{  
 display: flex;  
 flex-direction: column;  
 align-items: center;  
 justify-content: center;  
}  
  
.uploadButton{  
 font-size: 15px;  
 margin: 10px 0px;  
 background-color: #f7dfff;  
}  
.uploadButton:hover{  
 background: #b690c4;  
}  
  
input[type="number"].no-arrows::-webkit-inner-spin-button,  
input[type="number"].no-arrows::-webkit-outer-spin-button {  
 -webkit-appearance: none;  
 margin: 0;  
}  
  
/\* For Firefox \*/  
input[type="number"].no-arrows {  
 -moz-appearance: textfield;  
}

# AlumniConnect\src\data\sample\_students.json

[  
 {  
 "firstName": "John",  
 "fatherName": "Michael",  
 "lastName": "Doe",  
 "motherName": "Sarah",  
 "grandfatherName": "James",  
 "prn": "1234567684",  
 "moodleid": "12345678",  
 "mobileno": "9876543210",  
 "email": "john.doe@example.com",  
 "address": "123 Main Street, City",  
 "placeOfBirth": "Mumbai",  
 "dateOfBirth": "2000-01-15",  
 "branch": "COMPUTER SCIENCE",  
 "program": "B.Tech",  
 "semester": "8",  
 "yearOfJoining": "FE",  
 "dateOfAdmission": "2021-08-01",  
 "lastCollegeAttended": "Previous College",  
 "religion": "Hindu",  
 "caste": "General",  
 "category": "General",  
 "nationality": "Indian",  
 "password": "password123"  
 },  
 {  
 "firstName": "Priya",  
 "fatherName": "Rajesh",  
 "lastName": "Patel",  
 "motherName": "Meera",  
 "grandfatherName": "Krishna",  
 "prn": "2345678901",  
 "moodleid": "23456789",  
 "mobileno": "8765432109",  
 "email": "priya.patel@example.com",  
 "address": "456 Park Avenue, City",  
 "placeOfBirth": "Delhi",  
 "dateOfBirth": "2000-03-20",  
 "branch": "INFORMATION TECHNOLOGY",  
 "program": "B.Tech",  
 "semester": "8",  
 "yearOfJoining": "DSE",  
 "dateOfAdmission": "2021-08-01",  
 "lastCollegeAttended": "Previous College",  
 "religion": "Hindu",  
 "caste": "General",  
 "category": "General",  
 "nationality": "Indian",  
 "password": "password123"  
 },  
 {  
 "firstName": "Amit",  
 "fatherName": "Sunil",  
 "lastName": "Kumar",  
 "motherName": "Rita",  
 "grandfatherName": "Ram",  
 "prn": "3456789012",  
 "moodleid": "34567890",  
 "mobileno": "7654321098",  
 "email": "amit.kumar@example.com",  
 "address": "789 Lake View, City",  
 "placeOfBirth": "Bangalore",  
 "dateOfBirth": "2000-06-10",  
 "branch": "CSE-DATA SCIENCE",  
 "program": "B.Tech",  
 "semester": "8",  
 "yearOfJoining": "FE",  
 "dateOfAdmission": "2021-08-01",  
 "lastCollegeAttended": "Previous College",  
 "religion": "Hindu",  
 "caste": "General",  
 "category": "General",  
 "nationality": "Indian",  
 "password": "password123"  
 }  
]

# AlumniConnectAPI\.env

# Server Port  
  
PORT = 5000  
  
# MongoDb URI  
  
MONGO\_URI = mongodb://localhost:27017/alumniconnect  
  
  
  
# JWT Secret Key  
  
JWTSECRETKEY = 56593e3a139c4e5f8b5c1a1e474239e6  
  
  
  
# Email Config (Gmail)  
  
EMAIL\_USER=developermonkeydluffy@gmail.com  
  
EMAIL\_PASS=fghj zcec frhx hzht  
  
  
  
# Twilio Config  
  
TWILIO\_SID=your-twilio-sid  
  
TWILIO\_AUTH\_TOKEN=your-twilio-auth-token  
  
TWILIO\_PHONE\_NUMBER=your-twilio-phone-number

# AlumniConnectAPI\db.js

import mongoose from 'mongoose';  
import dotenv from "dotenv";  
  
dotenv.config();  
// Use environment variables for sensitive info (you can set it in a .env file)  
const mongoURI = process.env.MONGO\_URI;   
  
// Function to connect to MongoDB  
const connectToMongo = async () => {  
 await mongoose.connect(mongoURI)  
 .then(() => {  
 console.log('Connected to MongoDB');  
 })  
 .catch((error) => {  
 console.error('Error connecting to MongoDB:', error);  
 });  
};  
  
export default connectToMongo;

# AlumniConnectAPI\index.js

import connectToMongo from './db.js';  
import express, { json } from 'express';  
import cors from 'cors';  
import dotenv from "dotenv";  
import authRoutes from './routes/auth/userAuth.js';  
import lcRoutes from './routes/forms/lc.js';  
import otpRoutes from './routes/auth/otp.js';  
import notificationRoutes from './routes/notification/notification.js';  
import filesRoutes from './routes/files/files.js'   
  
dotenv.config();  
  
connectToMongo();  
  
const app = express()  
const port = process.env.PORT  
  
app.use(cors())  
app.use(json())  
app.use('/api/auth', authRoutes)  
app.use('/api/forms', lcRoutes)  
app.use('/api/notification', notificationRoutes)  
app.use('/api/auth/otp/',otpRoutes )  
app.use('/api/files',filesRoutes )  
  
app.listen(port, ()=>{  
 console.log(`CloudNotes Backend listining at URL http://localhost:${port}` )  
})

# AlumniConnectAPI\mailconfig.js

import nodemailer from "nodemailer";  
import dotenv from "dotenv";  
  
dotenv.config();  
  
const createTransporter = nodemailer.createTransport({  
 service: "gmail",  
 auth: {  
 user: process.env.EMAIL\_USER,  
 pass: process.env.EMAIL\_PASS,  
 },  
});  
  
export default createTransporter;

# AlumniConnectAPI\studRecord.json

[  
 {  
 "firstName": "Sneha",  
 "middleName": "Rajesh",  
 "lastName": "Verma",  
 "motherName": "Pooja Verma",  
 "grandfatherName": "Dinesh Verma",  
 "moodleid": 234567,  
 "mobileno": "9988776655",  
 "address": "456, Andheri West, Mumbai, India",  
 "dob": "2001-08-23",  
 "branch": "Information Technology",  
 "program": "BE",  
 "yearOfJoining": "DSE",  
 "yearOfAdmission": 2019,  
 "email": "sneha.verma@example.com"  
 },  
 {  
 "firstName": "Rohan",  
 "middleName": "Mahesh",  
 "lastName": "Patil",  
 "motherName": "Sangeeta Patil",  
 "grandfatherName": "Vikas Patil",  
 "moodleid": 345678,  
 "mobileno": "8765432109",  
 "address": "789, Pune, Maharashtra, India",  
 "dob": "2000-12-10",  
 "branch": "CSE-Data Science",  
 "program": "BE",  
 "yearOfJoining": "FE",  
 "yearOfAdmission": 2018,  
 "email": "rohan.patil@example.com"  
 },  
 {  
 "firstName": "Priya",  
 "middleName": "Suresh",  
 "lastName": "Deshmukh",  
 "motherName": "Anita Deshmukh",  
 "grandfatherName": "Shankar Deshmukh",  
 "moodleid": 456789,  
 "mobileno": "9123456780",  
 "address": "102, Nashik, Maharashtra, India",  
 "dob": "1999-06-05",  
 "branch": "CSE-AI & ML",  
 "program": "BE",  
 "yearOfJoining": "DSE",  
 "yearOfAdmission": 2017,  
 "email": "priya.deshmukh@example.com"  
 },  
 {  
 "firstName": "Vikas",  
 "middleName": "Ramesh",  
 "lastName": "Joshi",  
 "motherName": "Meena Joshi",  
 "grandfatherName": "Surendra Joshi",  
 "moodleid": 567890,  
 "mobileno": "9098765432",  
 "address": "12, Kandivali, Mumbai, India",  
 "dob": "2003-02-17",  
 "branch": "Electronics & Communication",  
 "program": "BE",  
 "yearOfJoining": "FE",  
 "yearOfAdmission": 2021,  
 "email": "vikas.joshi@example.com"  
 },  
 {  
 "firstName": "Kiran",  
 "middleName": "Arun",  
 "lastName": "Nair",  
 "motherName": "Lakshmi Nair",  
 "grandfatherName": "Vishnu Nair",  
 "moodleid": 678901,  
 "mobileno": "9012345678",  
 "address": "5, Thane, Maharashtra, India",  
 "dob": "2001-07-12",  
 "branch": "Mechanical Engineering",  
 "program": "BE",  
 "yearOfJoining": "DSE",  
 "yearOfAdmission": 2019,  
 "email": "kiran.nair@example.com"  
 },  
 {  
 "firstName": "Nikita",  
 "middleName": "Sudhir",  
 "lastName": "Chopra",  
 "motherName": "Neeta Chopra",  
 "grandfatherName": "Satish Chopra",  
 "moodleid": 789012,  
 "mobileno": "9123456789",  
 "address": "23, Kothrud, Pune, India",  
 "dob": "2000-03-21",  
 "branch": "Civil Engineering",  
 "program": "BE",  
 "yearOfJoining": "FE",  
 "yearOfAdmission": 2018,  
 "email": "nikita.chopra@example.com"  
 },  
 {  
 "firstName": "Manish",  
 "middleName": "Deepak",  
 "lastName": "Agarwal",  
 "motherName": "Sarita Agarwal",  
 "grandfatherName": "Govind Agarwal",  
 "moodleid": 890123,  
 "mobileno": "9234567890",  
 "address": "34, Powai, Mumbai, India",  
 "dob": "1999-09-15",  
 "branch": "Computer Science",  
 "program": "BE",  
 "yearOfJoining": "DSE",  
 "yearOfAdmission": 2017,  
 "email": "manish.agarwal@example.com"  
 },  
 {  
 "firstName": "Anushka",  
 "middleName": "Harish",  
 "lastName": "Shetty",  
 "motherName": "Kavita Shetty",  
 "grandfatherName": "Gopal Shetty",  
 "moodleid": 901234,  
 "mobileno": "9345678901",  
 "address": "56, Vashi, Navi Mumbai, India",  
 "dob": "2002-11-29",  
 "branch": "Information Technology",  
 "program": "BE",  
 "yearOfJoining": "FE",  
 "yearOfAdmission": 2020,  
 "email": "anushka.shetty@example.com"  
 },  
 {  
 "firstName": "Sagar",  
 "middleName": "Yogesh",  
 "lastName": "Pawar",  
 "motherName": "Madhuri Pawar",  
 "grandfatherName": "Suresh Pawar",  
 "moodleid": 987654,  
 "mobileno": "9456789012",  
 "address": "78, Borivali, Mumbai, India",  
 "dob": "2003-06-08",  
 "branch": "CSE-Data Science",  
 "program": "BE",  
 "yearOfJoining": "DSE",  
 "yearOfAdmission": 2021,  
 "email": "sagar.pawar@example.com"  
 }]

# AlumniConnectAPI\middleware\fetchuser.js

import jwt from 'jsonwebtoken';  
import dotenv from "dotenv";  
dotenv.config();  
  
const jwtSecretKey = process.env.JWTSECRETKEY;  
  
const fetchUser = (req, res, next) => {  
 // Get the user from jwt token and send the id in response  
 const token = req.header('auth-token');  
 if (!token) {  
 return res.status(401).json({ error: 'Please validate using a valid token' });  
 }  
   
 try {  
 const data = jwt.verify(token, jwtSecretKey); // Use the 'verify' method from the default import  
 // console.log(data)  
 req.userId = data.id; // Attach user info to the request object  
 // console.log(req.userId)  
 next(); // Only call next if the token is valid  
 } catch (error) {  
 return res.status(401).json({ error: 'Invalid token' }); // Send error if token verification fails  
 }  
};  
  
export default fetchUser;

# AlumniConnectAPI\models\convocation.js

import mongoose, { model } from 'mongoose';  
const { Schema } = mongoose;  
  
// Schema definition for the Convocation entity, this is to store student details related to convocation  
const convocationSchema = new Schema({  
 // First Name of the student  
 'First name': {  
 type: String,  
 required: true // This is mandatory since it’s part of their identity  
 },  
  
 // Middle Name of the student (in case they have one)  
 'Middle name': {  
 type: String,  
 required: true // This field is mandatory as well  
 },  
  
 // Last Name of the student  
 'Last name': {  
 type: String,  
 required: true // Also mandatory  
 },  
  
 // Roll Number or Moodle ID, used to uniquely identify the student in the system  
 'moodleid/rollno': {  
 type: Number,  
 required: true, // This is required because it’s the student’s unique identifier  
 unique: true // Ensures that no two students can have the same roll number  
 },  
  
 // PRN (Permanent Registration Number), another unique identifier for the student  
 // This is used by universities for registration purposes  
 'PRN NUMBER': {  
 type: Number,  
 required: true // This is a mandatory field for the convocation process  
 },  
  
 // Student's mobile number, for communication purposes  
 mobileno: {  
 type: Number,  
 required: true // Required to contact the student in case of any issues or updates  
 },  
  
 // Student's email address, likely needed for sending official communications  
 email: {  
 type: String,  
 required: true, // Also mandatory for email correspondence  
 },  
  
 // Date of Birth of the student (this helps to verify age and other eligibility criteria)  
 'date of birth': {  
 type: Date,  
 default: Date.now // If not provided, defaults to the current date (may need adjustment)  
 },  
  
 // Place of Birth for the student (useful for demographic and verification purposes)  
 'Place of Birth': {  
 type: String,  
 required: true // Needed to track where the student was born (for records)  
 },  
  
 // The date the student was admitted into the college (could be used for record-keeping)  
 'Date of Admission in this college': {  
 type: Date,  
 default: Date.now // Defaults to the current date if not provided (though it’s likely to be given)  
 },  
  
 // The program the student is enrolled in (e.g., B.Tech, M.Tech, MBA)  
 Program: {  
 type: String,  
 required: true // We need this to know the student’s course of study  
 },  
  
 // Year of Convocation, indicating when the student is expected to graduate and attend the ceremony  
 'Year of Convocation': {  
 type: Date,  
 default: Date.now // This might need to be set to a specific year, but the current date is set by default  
 },  
  
 // \*\*Additional Information for the Last College Attended\*\*   
  
 // Last College Attended (in case the student transferred from another institution)  
 'Last College Attended': {  
 type: String,  
 required: false // This is optional unless the student transferred from another college  
 },  
  
 // The course the student attended in the last college (e.g., B.Tech, M.Sc., etc.)  
 'Last College Course': {  
 type: String,  
 required: false // Optional, but useful if they transferred from another course  
 },  
  
 // The semester the student was in when they left the last college  
 'Last College Semester': {  
 type: String, // This could be a number (e.g., 4) or a string (e.g., "Semester 2")  
 required: false // Optional unless it applies to the student  
 },  
  
 // Date the student left the last college (useful for record tracking)  
 'Date of Leaving Last College': {  
 type: Date,  
 required: false // Optional unless the student transferred or discontinued from a previous institution  
 },  
  
 // Reason for leaving the last college (e.g., transferred, discontinued, etc.)  
 'Reason of Leaving Last College': {  
 type: String,  
 required: false // This is optional, but could be important for tracking reasons  
 },  
  
 // Remarks on conduct at the last college (important for the college records and LOR)  
 'Remark of Conduct in Last College': {  
 type: String,  
 required: false // This field can be left blank, but it's helpful for keeping records  
 },  
  
 // \*\*End of Additional Information\*\*   
  
});  
  
// Create a model called "convocations" from the schema definition  
const convocation = model('convocations', convocationSchema);  
  
// Export the model to be used elsewhere in the application  
export default convocation;

# AlumniConnectAPI\models\leavingcertificate.js

import mongoose, { model } from 'mongoose';  
const { Schema } = mongoose;  
  
// Schema definition for Leaving Certificate, storing student-related information when they leave the college  
const leavingcertificateSchema = new Schema({  
 userId:{  
 type: mongoose.Schema.ObjectId,  
 ref: 'students',  
 required:true  
 },  
 moodleid: {  
 type: Number,  
 required: true,  
 unique: true  
 },  
 feesReciept:{  
 type:String  
 },  
 varificationDoc:{  
 type:String  
 },  
 prn: {  
 type: Number,  
 required: true,  
 unique: true  
 },  
 firstName: {  
 type: String,  
 required: true  
 },  
 fatherName: {  
 type: String,  
 required: true  
 },  
 lastName: {  
 type: String,  
 required: true  
 },  
 motherName: {  
 type: String,  
 required: true  
 },  
 grandfatherName: {  
 type: String,  
 required: true  
 },  
 religion: {  
 type: String,  
 required: true,  
 },  
 caste: {  
 type: String,  
 required: true,  
 },  
 category: {  
 type: String,  
 required: true,  
 },  
 nationality: {  
 type: String,  
 required: true,  
 },  
 placeOfBirth: {  
 type: String,  
 required: true,  
 },  
 dateOfBirth: {  
 type: String,  
 required: true,  
 },  
 lastCollegeAttended: {  
 type: String,  
 required: true,  
 },  
 program: {  
 type: String,  
 required: true,  
 },  
 branch: {  
 type: String,  
 required: true,  
 },  
 semester:{  
 type: Number,  
 required: true,  
 },  
 yearOfJoining: {  
 type: String,  
 required: true  
 },  
 dateOfAdmission: {  
 type: String,  
 required: true  
 },  
 createdAt: {  
 type: Date,  
 default: Date.now  
 }  
});  
  
// Create a model called "leavingcertificates" from the schema definition  
const leavingcertificate = model('leavingcertificates', leavingcertificateSchema);  
  
// Export the model to be used elsewhere in the application  
export default leavingcertificate;

# AlumniConnectAPI\models\lor.js

import mongoose, { model } from 'mongoose';  
const { Schema } = mongoose;  
  
// Schema definition for the LOR (Letter of Recommendation) entity  
const lorSchema = new Schema({  
 // First Name of the student (as part of their personal details)  
 First name: {  
 type: String,  
 required: true // This field is mandatory  
 },  
   
 // Middle Name of the student (again part of their personal details)  
 Middle name: {  
 type: String,  
 required: true // This field is mandatory  
 },  
   
 // Last Name of the student  
 Last name: {  
 type: String,  
 required: true // This field is mandatory  
 },  
  
 // Roll Number or ID for the student - this field must be unique across the database  
 // This is a unique identifier for the student  
 'moodleid/rollno': {  
 type: Number,  
 required: true, // This is mandatory because it's how we uniquely identify the student  
 unique: true // Ensures no two students can have the same roll number  
 },  
  
 // The number of copies of the LOR requested by the student (why? Not sure but seems like an admin thing)  
 'Number of LOR Copies': {  
 type: Number,  
 required: true, // Makes sense to make this a required field  
 unique: true // Ensures no two records have the same number of copies requested  
 },  
  
 // The date the student was admitted to the college, defaults to the current date  
 // This could help track how long they've been studying  
 'Date of Admission in this college': {  
 type: Date,  
 default: Date.now // If no date is provided, it will automatically set to the current date  
 },  
  
 // The academic year the student is currently studying in (e.g., 2023-24)  
 'Academic Year in which studing': {  
 type: String,  
 required: true // We need this to categorize students by academic year  
 },  
  
 // The semester in which the student is currently enrolled (it could be a number or a string depending on input)  
 'Semester in which studing': {  
 type: String, // Could also be a number, but using string for flexibility  
 required: true // This will always be required  
 },  
  
 // The semester field again, seems like a duplicate or could be the same as the above  
 // Might need to refactor or remove this if unnecessary  
 semester: {  
 type: String, // Or could be a number (again flexibility here)  
 required: true // Mandatory field  
 },  
  
 // List of roles the student holds, such as 'HOD', 'Principal', etc.  
 roles: {  
 type: [String], // An array to hold multiple roles  
 required: true // This field will always have values, like leadership positions  
 },  
  
 // The branch the student is enrolled in (e.g., Computer Science, Electrical Engineering, etc.)  
 'Branch in which studying': {  
 type: String,  
 required: true // Required to know the student's field of study  
 },  
  
 // Staff names - Seems like a student is supposed to provide staff members who have handled them  
 // Probably for tracking who will sign the LORs?  
 'Staff Name 1': {  
 type: String,  
 required: true // This is important to link the LOR request to staff members  
 },  
 'Staff Name 2': {  
 type: String,  
 required: true  
 },  
 'Staff Name 3': {  
 type: String,  
 required: true  
 },  
 'Staff Name 4': {  
 type: String,  
 required: true  
 },  
});  
  
// Creating the model for LOR based on the schema  
const lor = model('lors', lorSchema)  
  
// Exporting the model for use in other parts of the application  
export default lor;

# AlumniConnectAPI\models\marksheet.js

import mongoose, { model } from 'mongoose';  
const { Schema } = mongoose;  
  
// Define the marksheet schema with various fields related to student records  
const marksheetSchema = new Schema({  
 // Student's first name  
 firstName: {  
 type: String,  
 required: true, // This field is required  
 },  
   
 // Student's middle name  
 middleName: {  
 type: String,  
 required: true, // This field is required  
 },  
   
 // Student's last name  
 lastName: {  
 type: String,  
 required: true, // This field is required  
 },  
   
 // Roll number or model ID, made unique for each student  
 rollNo: {  
 type: Number,  
 required: true, // This field is required  
 unique: true, // Ensures no duplicate roll numbers  
 },  
   
 // PRN (Personal Registration Number) - an identifier for the student  
 prnNumber: {  
 type: Number,  
 required: true, // This field is required  
 },  
   
 // Branch of study (e.g., Computer Science, Information Technology, Civil Engineering, Mechanical Engineering)  
 branch: {  
 type: String,  
 required: true, // This field is required  
 enum: ['Computer Science', 'Information Technology', 'Civil Engineering', 'Mechanical Engineering'], // Allow only the listed branches  
 },  
   
 // The semester the student is currently in (could be string or number depending on input)  
 semester: {  
 type: String, // Could also use Number if semester numbers are used (e.g., "1", "2")  
 required: true, // This field is required  
 },  
  
 // Semester-wise subjects for Computer Science, Information Technology, Civil Engineering, Mechanical Engineering  
   
 // Semester 1 - Common subjects across branches (for most engineering branches)  
 semester1: {  
 Applied\_Mathematics\_I: { type: Number, required: true },  
 Engineering\_Physics\_I: { type: Number, required: true },  
 Engineering\_Chemistry\_I: { type: Number, required: true },  
 Engineering\_Mechanics: { type: Number, required: true },  
 Basic\_Electrical\_Engineering: { type: Number, required: true },  
 Engineering\_Graphics: { type: Number, required: true },  
 Computer\_Programming: { type: Number, required: true },  
 Environmental\_Studies: { type: Number, required: true },  
 },  
  
 // Semester 2 - Subjects for Computer Science, IT, Civil, Mechanical  
 semester2: {  
 Applied\_Mathematics\_II: { type: Number, required: true },  
 Engineering\_Physics\_II: { type: Number, required: true },  
 Engineering\_Chemistry\_II: { type: Number, required: true },  
 Basic\_Electronics\_Engineering: { type: Number, required: true },  
 Engineering\_Mechanics\_II: { type: Number, required: true },  
 Engineering\_Graphics\_II: { type: Number, required: true },  
 Data\_Structures: { type: Number, required: true },  
 Professional\_Communication: { type: Number, required: true },  
 },  
  
 // Semester 3 - Subjects for Computer Science and IT  
 semester3\_CS\_IT: {  
 Applied\_Mathematics\_III: { type: Number, required: true },  
 Digital\_Logic\_Design: { type: Number, required: true },  
 Data\_Structures\_and\_Algorithms: { type: Number, required: true },  
 Database\_Management\_Systems: { type: Number, required: true },  
 Principles\_of\_Communication: { type: Number, required: true },  
 Digital\_Design\_Lab: { type: Number, required: true },  
 Data\_Structures\_Lab: { type: Number, required: true },  
 Java\_Programming\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 3 - Subjects for Civil Engineering  
 semester3\_Civil: {  
 Applied\_Mathematics\_III: { type: Number, required: true },  
 Building\_Materials: { type: Number, required: true },  
 Fluid\_Mechanics: { type: Number, required: true },  
 Surveying: { type: Number, required: true },  
 Structural\_Analysis: { type: Number, required: true },  
 Geotechnical\_Engineering: { type: Number, required: true },  
 Surveying\_Lab: { type: Number, required: true },  
 Fluid\_Mechanics\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 3 - Subjects for Mechanical Engineering  
 semester3\_Mechanical: {  
 Applied\_Mathematics\_III: { type: Number, required: true },  
 Strength\_of\_Materials: { type: Number, required: true },  
 Thermodynamics: { type: Number, required: true },  
 Fluid\_Mechanics\_Mechanical: { type: Number, required: true },  
 Material\_Science: { type: Number, required: true },  
 Mechanical\_Workshop\_Lab: { type: Number, required: true },  
 Engineering\_Materials\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 4 - Core subjects for Computer Science and IT  
 semester4\_CS\_IT: {  
 Applied\_Mathematics\_IV: { type: Number, required: true },  
 Computer\_Organization\_and\_Architecture: { type: Number, required: true },  
 Operating\_Systems: { type: Number, required: true },  
 Microprocessor\_and\_Interfacing: { type: Number, required: true },  
 Design\_and\_Analysis\_of\_Algorithms: { type: Number, required: true },  
 Digital\_Signal\_Processing: { type: Number, required: true },  
 Microprocessor\_Lab: { type: Number, required: true },  
 Operating\_Systems\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 4 - Core subjects for Civil Engineering  
 semester4\_Civil: {  
 Applied\_Mathematics\_IV: { type: Number, required: true },  
 Structural\_Analysis\_II: { type: Number, required: true },  
 Environmental\_Engineering: { type: Number, required: true },  
 Geotechnical\_Engineering\_II: { type: Number, required: true },  
 Concrete\_Technology: { type: Number, required: true },  
 Transportation\_Engineering: { type: Number, required: true },  
 Structural\_Lab: { type: Number, required: true },  
 Concrete\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 4 - Core subjects for Mechanical Engineering  
 semester4\_Mechanical: {  
 Applied\_Mathematics\_IV: { type: Number, required: true },  
 Kinematics\_of\_Machines: { type: Number, required: true },  
 Thermodynamics\_II: { type: Number, required: true },  
 Machine\_Design: { type: Number, required: true },  
 Heat\_Transfer: { type: Number, required: true },  
 Fluid\_Mechanics\_II: { type: Number, required: true },  
 Workshop\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 5 - Advanced subjects for Computer Science and IT  
 semester5\_CS\_IT: {  
 Applied\_Mathematics\_V: { type: Number, required: true },  
 Software\_Engineering: { type: Number, required: true },  
 Computer\_Networks: { type: Number, required: true },  
 Theory\_of\_Computation: { type: Number, required: true },  
 Web\_Technologies: { type: Number, required: true },  
 Software\_Engineering\_Lab: { type: Number, required: true },  
 Computer\_Networks\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 5 - Core subjects for Civil Engineering  
 semester5\_Civil: {  
 Applied\_Mathematics\_V: { type: Number, required: true },  
 Transportation\_Engineering\_II: { type: Number, required: true },  
 Structural\_Design\_I: { type: Number, required: true },  
 Fluid\_Mechanics\_II: { type: Number, required: true },  
 Building\_Construction: { type: Number, required: true },  
 Irrigation\_Engineering: { type: Number, required: true },  
 Surveying\_Lab: { type: Number, required: true },  
 Fluid\_Mechanics\_Lab\_II: { type: Number, required: true },  
 },  
  
 // Semester 5 - Core subjects for Mechanical Engineering  
 semester5\_Mechanical: {  
 Applied\_Mathematics\_V: { type: Number, required: true },  
 Machine\_Design\_II: { type: Number, required: true },  
 Fluid\_Mechanics\_III: { type: Number, required: true },  
 Thermal\_Engineering\_II: { type: Number, required: true },  
 Manufacturing\_Processes: { type: Number, required: true },  
 Mechatronics: { type: Number, required: true },  
 Manufacturing\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 6 - Advanced subjects for Computer Science and IT  
 semester6\_CS\_IT: {  
 Applied\_Mathematics\_VI: { type: Number, required: true },  
 Compiler\_Design: { type: Number, required: true },  
 Artificial\_Intelligence: { type: Number, required: true },  
 Information\_Security: { type: Number, required: true },  
 Cloud\_Computing: { type: Number, required: true },  
 Compiler\_Design\_Lab: { type: Number, required: true },  
 AI\_Lab: { type: Number, required: true },  
 Information\_Security\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 6 - Core subjects for Civil Engineering  
 semester6\_Civil: {  
 Applied\_Mathematics\_VI: { type: Number, required: true },  
 Structural\_Design\_II: { type: Number, required: true },  
 Hydraulics: { type: Number, required: true },  
 Environmental\_Management: { type: Number, required: true },  
 Construction\_Management: { type: Number, required: true },  
 Concrete\_Lab\_II: { type: Number, required: true },  
 Structural\_Design\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 6 - Core subjects for Mechanical Engineering  
 semester6\_Mechanical: {  
 Applied\_Mathematics\_VI: { type: Number, required: true },  
 Refrigeration\_and\_Air\_Conditioning: { type: Number, required: true },  
 Manufacturing\_Technology\_II: { type: Number, required: true },  
 Power\_Engineering: { type: Number, required: true },  
 Thermal\_Engineering\_III: { type: Number, required: true },  
 Fluid\_Mechanics\_III: { type: Number, required: true },  
 Heat\_Transfer\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 7 and 8: Further semesters will follow similar structure for all branches.  
 // You can continue this structure further as per the branch's detailed syllabus.  
// Semester 7 - Core subjects for Computer Science and IT  
semester7\_CS\_IT: {  
 Enterprise\_Network\_Design: { type: Number, required: true },  
 Infrastructure\_Security: { type: Number, required: true },  
 Artificial\_Intelligence\_II: { type: Number, required: true },  
 Department\_Level\_Optional\_Course\_III: { type: Number, required: true },  
 Institute\_Level\_Optional\_Course\_I: { type: Number, required: true },  
 Network\_Design\_Lab: { type: Number, required: true },  
 Advanced\_Security\_Lab: { type: Number, required: true },  
 Intelligence\_Systems\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 7 - Core subjects for Civil Engineering  
 semester7\_Civil: {  
 Advanced\_Structural\_Design: { type: Number, required: true },  
 Project\_Management\_in\_Construction: { type: Number, required: true },  
 Construction\_Quality\_Management: { type: Number, required: true },  
 Structural\_Design\_Lab\_III: { type: Number, required: true },  
 Geotechnical\_Engineering\_III: { type: Number, required: true },  
 Environmental\_Impact\_Assessment: { type: Number, required: true },  
 Surveying\_Lab\_III: { type: Number, required: true },  
 },  
  
 // Semester 7 - Core subjects for Mechanical Engineering  
 semester7\_Mechanical: {  
 Advanced\_Manufacturing\_Technology: { type: Number, required: true },  
 Industrial\_Engineering: { type: Number, required: true },  
 Automation\_in\_Manufacturing: { type: Number, required: true },  
 Advanced\_Thermal\_Engineering: { type: Number, required: true },  
 Robotics: { type: Number, required: true },  
 Mechatronics\_Lab: { type: Number, required: true },  
 Automation\_Lab: { type: Number, required: true },  
 },  
  
 // Semester 8 - Core subjects for Computer Science and IT  
 semester8\_CS\_IT: {  
 Big\_Data\_Analytics: { type: Number, required: true },  
 Internet\_of\_Things: { type: Number, required: true },  
 Mobile\_Application\_Development: { type: Number, required: true },  
 Department\_Level\_Optional\_Course\_IV: { type: Number, required: true },  
 Institute\_Level\_Optional\_Course\_II: { type: Number, required: true },  
 Project\_Work\_I: { type: Number, required: true },  
 Project\_Work\_II: { type: Number, required: true },  
 },  
  
 // Semester 8 - Core subjects for Civil Engineering  
 semester8\_Civil: {  
 Advanced\_Railway\_Engineering: { type: Number, required: true },  
 Wastewater\_Treatment: { type: Number, required: true },  
 Disaster\_Management\_in\_Buildings: { type: Number, required: true },  
 Advanced\_Structural\_Lab: { type: Number, required: true },  
 Building\_Information\_Modeling: { type: Number, required: true },  
 Construction\_Management\_Lab: { type: Number, required: true },  
 Project\_Work\_II: { type: Number, required: true },  
 },  
  
 // Semester 8 - Core subjects for Mechanical Engineering  
 semester8\_Mechanical: {  
 Advanced\_Manufacturing\_Processes: { type: Number, required: true },  
 Design\_Engineering: { type: Number, required: true },  
 Sustainable\_Energy\_Management: { type: Number, required: true },  
 Industrial\_Automation: { type: Number, required: true },  
 Project\_Work\_I: { type: Number, required: true },  
 Project\_Work\_II: { type: Number, required: true },  
 Robotics\_Lab: { type: Number, required: true },  
 },  
});  
  
// Create and export a model named 'marksheets' based on the schema  
const marksheet = model('marksheets', marksheetSchema);  
  
// Export the model for use in the application  
export default marksheet;

# AlumniConnectAPI\models\notification.js

import mongoose from 'mongoose';  
  
const notificationSchema = new mongoose.Schema({  
 userId: {  
 type: mongoose.Schema.ObjectId,  
 ref: 'studentsForms',  
 },  
 title: {  
 type: String,  
 required: true  
 },  
 message: {  
 type: String,  
 required: true  
 },  
 from: {  
 type: String,  
 required: true  
 },  
 timestamp: {  
 type: Date,  
 default: Date.now  
 },  
 isRead: {  
 type: Boolean,  
 default: false  
 }  
});  
  
const Notification = mongoose.model('Notification', notificationSchema);  
  
export default Notification;

# AlumniConnectAPI\models\otp.js

import mongoose from "mongoose";  
  
const otpSchema = new mongoose.Schema({  
 moodleid: {  
 type: Number,  
 required: true  
 },  
 email: {  
 type: String  
 },  
 mobileno: {  
 type: String  
 },  
 otp: {  
 type: String,  
 required: true  
 },  
  
 createdAt: {  
 type: Date, default:  
 Date.now,  
 expires: 300  
 }, // OTP expires in 5 mins  
});  
  
export default mongoose.model("Otp", otpSchema);

# AlumniConnectAPI\models\student.js

import mongoose, { model } from 'mongoose';  
const { Schema } = mongoose;  
  
const studentSchema = new Schema({  
 firstLogin: {  
 type: Boolean,  
 default: false  
 },  
 password: {  
 type: String,  
 required:true  
 },  
 moodleid: {  
 type: Number,  
 required: true,  
 unique: true  
 },  
 prn: {  
 type: Number,  
 required: true,  
 unique: true  
 },  
 profilePicture:{  
 type:String  
 },  
 firstName: {  
 type: String,  
 required: true  
 },  
 fatherName: {  
 type: String,  
 required: true  
 },  
 lastName: {  
 type: String,  
 required: true  
 },  
 motherName: {  
 type: String,  
 required: true  
 },  
 grandfatherName: {  
 type: String,  
 required: true  
 },  
 mobileno: {  
 type: Number,  
 required: true  
 },  
 email: {  
 type: String,  
 required: true,  
 },  
 address: {  
 type: String,  
 required: true,  
 },  
 religion: {  
 type: String,  
 required: true,  
 },  
 caste: {  
 type: String,  
 required: true,  
 },  
 category: {  
 type: String,  
 required: true,  
 },  
 nationality: {  
 type: String,  
 required: true,  
 },  
 placeOfBirth: {  
 type: String,  
 required: true,  
 },  
 dateOfBirth: {  
 type: String,  
 required: true,  
 },  
 lastCollegeAttended: {  
 type: String,  
 required: true,  
 },  
 program: {  
 type: String,  
 required: true,  
 },  
 branch: {  
 type: String,  
 required: true,  
 },  
 semester:{  
 type: Number,  
 required: true,  
 },  
 yearOfJoining: {  
 type: String,  
 required: true  
 },  
 dateOfAdmission: {  
 type: String,  
 required: true  
 },  
 date: {  
 type: Date,  
 default: Date.now  
 },  
});  
  
const Student = model('students', studentSchema)  
// User.createIndexes()  
export default Student

# AlumniConnectAPI\models\submittedForms.js

import mongoose, { model } from 'mongoose';  
const { Schema } = mongoose;  
  
// Schema definition for Leaving Certificate, storing student-related information when they leave the college  
const submittedFormsSchema = new Schema({  
 docType:{  
 type:String,  
 required:true  
 },  
 userId:{  
 type: mongoose.Schema.ObjectId,  
 ref: 'students'  
 },  
 docId:{  
 type: mongoose.Schema.ObjectId,  
 ref: 'leavingcertificate'  
 },  
 branch:{  
 type:String,  
 required:true  
 },  
 status:{  
 type:String,  
 default:"Submitted",  
 required:true  
 },  
 submitted: {  
 type: Boolean,  
 default: true,  
 required: true  
 },  
 submittedDate: {  
 type: Date,  
 default: Date.now  
 },  
 submittedRemarks: {  
 type: String,  
 default: "Document submitted by student"  
 },  
 admin: {  
 type: Boolean,  
 default: false,  
 required: true  
 },  
 adminDate: {  
 type: Date,  
 default: Date.now  
 },  
 adminRemarks: {  
 type: String,  
 default: "Verification pending at Admin"  
 },  
 adminFeesPaid: {  
 type: Boolean,  
 default: true  
 },  
 feesRecieptUploded:{  
 type:Boolean  
 },  
 varificationDocUploded:{  
 type:Boolean  
 },  
 documentVerified: {  
 type: Boolean,  
 default: true  
 },  
 examCell: {  
 type: Boolean,  
 default: false,  
 required: true  
 },  
 examCellDate: {  
 type: Date  
 },  
 examCellRemarks: {  
 type: String,  
 default: "Exam cell verification pending"  
 },  
 examCellFeesPaid: {  
 type: Boolean,  
 default: true  
 },  
 library: {  
 type: Boolean,  
 default: false,  
 required: true  
 },  
 libraryDate: {  
 type: Date  
 },  
 libraryRemarks: {  
 type: String,  
 default: "Library verification pending"  
 },  
 libraryFeesPaid: {  
 type: Boolean,  
 default: true  
 },  
 atbranch: {  
 type: Boolean,  
 default: false,  
 required: true  
 },  
 branchDate: {  
 type: Date  
 },  
 branchRemarks: {  
 type: String,  
 default: "Branch verification pending"  
 },  
 uploadDocument: {  
 type: Boolean,  
 default: false,  
 required: true  
 },  
 collected: {  
 type: Boolean,  
 default: false,  
 required: true  
 },  
 collectionDate: {  
 type: Date  
 },  
 collectionRemarks: {  
 type: String,  
 default: "Document ready for collection"  
 },  
 createdAt: {  
 type: Date,  
 default: Date.now  
 },  
 updatedAt: {  
 type: Date,  
 default: Date.now  
 }  
});  
  
// Update the updatedAt field before saving  
submittedFormsSchema.pre('save', function(next) {  
 this.updatedAt = Date.now();  
 next();  
});  
  
// Create a model called "submittedDocs" from the schema definition  
const submittedForms = model('submittedForms', submittedFormsSchema);  
  
// Export the model to be used elsewhere in the application  
export default submittedForms;

# AlumniConnectAPI\models\teacher.js

import mongoose, { model } from 'mongoose';  
const { Schema } = mongoose;  
  
const teacherSchema = new Schema({  
 firstLogin:{  
 type:Boolean,  
 default:false  
 },  
 moodleid:{  
 type: Number,  
 required: true,  
 unique: true  
 },  
 password:{  
 type: String,  
 required: true  
 },  
 firstName: {  
 type: String,  
 required: true  
 },  
 middleName: {  
 type: String,  
 required: true  
 },  
 lastName: {  
 type: String,  
 required: true  
 },  
 mobileno:{  
 type:Number,  
 required: true  
 },  
 email:{  
 type: String,  
 required: true,  
 },  
 address: {  
 type: String,  
 required: true,  
 },  
 department: {  
 type: String,  
 required: true,  
 },  
 post: {  
 type: String,  
 required: true,  
 },  
 date:{  
 type: Date,  
 default: Date.now  
 }  
});  
  
const Teacher=model('teachers', teacherSchema)  
// User.createIndexes()  
export default Teacher

# AlumniConnectAPI\routes\auth\otp.js

import express from "express";  
import { body, validationResult } from 'express-validator';  
import createTransporter from "../../mailconfig.js";  
import twilio from "twilio";  
import crypto from "crypto";  
import dotenv from "dotenv";  
import Otp from "../../models/Otp.js";  
import Student from '../../models/student.js';  
import Teacher from '../../models/teacher.js';  
import fetchUser from '../../middleware/fetchuser.js';  
// import createTransporter from "../../mailconfig.js";  
  
dotenv.config();  
const router = express.Router();  
  
// Twilio Config  
// const twilioClient = twilio(process.env.TWILIO\_SID, process.env.TWILIO\_AUTH\_TOKEN);  
  
// Nodemailer Transporter  
const transporter = createTransporter;  
  
// Generate 6-digit OTP  
const generateOTP = () => crypto.randomInt(100000, 999999).toString();  
  
// \*\*Generate & Send OTP (Email & SMS)\*\*  
router.post('/:user/sendotp',  
 [  
 body('moodleid', 'Enter a valid moodle id').trim().isNumeric().escape(),  
 ],  
 async (req, res) => {  
 try {  
 let success = false;  
 const errors = validationResult(req);  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 const { moodleid } = req.body;  
 const { user } = req.params  
  
 let contact  
  
 if (user === 'student') {  
 contact = await Student.findOne({ moodleid }).select('email mobileno')  
 } else if (user === 'teacher') {  
 contact = await Teacher.findOne({ moodleid }).select('email mobileno')  
 } else {  
 return res.status(400).json({ success, error: "Invalid user type." });  
 }  
  
 if (!contact) {  
 return res.status(404).json({ success, error: "User not found." });  
 }  
  
 const { email, mobileno } = contact;  
  
 const otp = generateOTP();  
 await Otp.create({ moodleid, email, mobileno, otp });  
  
 // Send OTP via Email  
 await transporter.sendMail({  
 // from: process.env.EMAIL\_USER,  
 from: '"APSIT " <developermonkeydluffy@gmail.com>',  
 to: email,  
 subject: "Your OTP Code",  
 text: `Your OTP is: ${otp}. It is valid for 5 minutes.`,  
 });  
  
 // // Send OTP via SMS  
 // await twilioClient.errors.create({  
 // body: `Your OTP is: ${otp}. It is valid for 5 minutes.`,  
 // from: process.env.TWILIO\_PHONE\_NUMBER,  
 // to: mobile,  
 // });  
  
 res.json({ success: true, error: "OTP sent successfully to email and mobile." });  
 } catch (error) {  
 console.error("OTP Error:", error);  
 res.status(500).json({ success: false, error: "Error sending OTP." });  
 }  
 });  
  
router.post('/:user/sendemailotp',  
 [  
 body('email', 'Enter a valid Email').isEmail().escape(),  
 ],  
 fetchUser,  
 async (req, res) => {  
 try {  
 let success = false;  
  
 const errors = validationResult(req);  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 const { email } = req.body  
 const userId = req.userId;  
 const { user } = req.params  
  
 let moodleid  
 if(user==='student'){  
 moodleid = await Student.findById(userId).select('moodleid');  
 } else if(user==='teacher'){  
 moodleid = await Teacher.findById(userId).select('moodleid');  
 } else{  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
 moodleid = moodleid.moodleid  
  
 const otp = generateOTP();  
 await Otp.create({ moodleid, email, otp });  
  
 // Send OTP via Email  
 await transporter.sendMail({  
 // from: process.env.EMAIL\_USER,  
 from: '"APSIT " <developermonkeydluffy@gmail.com>',  
 to: email,  
 subject: "Your OTP Code",  
 text: `Your OTP is: ${otp}. It is valid for 5 minutes.`,  
 });  
  
 res.json({ success: true, message: "OTP sent successfully to email." });  
 } catch (error) {  
 console.error("OTP Error:", error);  
 res.status(500).json({ success: false, error: "Error sending OTP." });  
 }  
 });  
  
router.post('/:user/sendmobileotp',  
 [  
 body('mobileno', 'Enter a valid mobile number').trim().isNumeric().escape()  
 ],  
 fetchUser,  
 async (req, res) => {  
 try {  
 let success = false;  
  
 const errors = validationResult(req);  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 const { mobileno } = req.body  
 const userId = req.userId;  
 const { user } = req.params  
  
 let moodleid  
 if(user==='student'){  
 moodleid = await Student.findById(userId).select('moodleid');  
 } else if(user==='teacher'){  
 moodleid = await Teacher.findById(userId).select('moodleid');  
 } else{  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
 moodleid = moodleid.moodleid  
  
 const otp = generateOTP();  
 await Otp.create({ moodleid, mobileno, otp });  
  
 // // Send OTP via SMS  
 // await twilioClient.errors.create({  
 // body: `Your OTP is: ${otp}. It is valid for 5 minutes.`,  
 // from: process.env.TWILIO\_PHONE\_NUMBER,  
 // to: mobile,  
 // });  
  
 res.json({ success: true, message: "OTP sent successfully to mobile." });  
 } catch (error) {  
 console.error("OTP Error:", error);  
 res.status(500).json({ success: false, error: "Error sending OTP." });  
 }  
 });  
  
  
  
// \*\*Verify OTP\*\*  
router.post("/verifyotp",  
 [  
 body('moodleid', 'Enter a valid moodle id').trim().isNumeric().escape(),  
 body('otp', 'Enter a valid otp').trim().isNumeric().escape(),  
 ],  
 fetchUser,  
 async (req, res) => {  
 try {  
 let success = false;  
 const errors = validationResult(req);  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 const { moodleid, otp } = req.body;  
  
 const storedOtp = await Otp.findOne({ moodleid, otp });  
 if (!storedOtp) return res.status(400).json({ success: false, error: "Invalid or expired OTP." });  
  
 // OTP is valid, delete it from the database  
 await Otp.deleteMany({ moodleid });  
 res.json({ success: true, error: "OTP verified successfully." });  
 } catch (error) {  
 console.error("OTP Verification Error:", error);  
 res.status(500).json({ success: false, error: "Error verifying OTP." });  
 }  
 });  
  
  
router.post("/verifymobileotp",  
 [  
 body('mobileno', 'Enter a valid mobile number').trim().isNumeric().escape(),  
 body('otp', 'Enter a valid otp').trim().isNumeric().escape(),  
 ],  
 fetchUser,  
 async (req, res) => {  
 try {  
 let success = false;  
 const errors = validationResult(req);  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 const { mobileno, otp } = req.body;  
  
 const storedOtp = await Otp.findOne({ mobileno, otp });  
 if (!storedOtp) return res.status(400).json({ success: false, error: "Invalid or expired OTP." });  
  
 // OTP is valid, delete it from the database  
 await Otp.deleteMany({ mobileno });  
 res.json({ success: true, error: "OTP verified successfully." });  
 } catch (error) {  
 console.error("OTP Verification Error:", error);  
 res.status(500).json({ success: false, error: "Error verifying OTP." });  
 }  
 });  
  
   
router.post("/verifyemailotp",  
 [  
 body('email', 'Enter a valid Email').isEmail().escape(),  
 body('otp', 'Enter a valid otp').trim().isNumeric().escape(),  
 ],  
 fetchUser,  
 async (req, res) => {  
 try {  
 let success = false;  
 const errors = validationResult(req);  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 const { email, otp } = req.body;  
  
 const storedOtp = await Otp.findOne({ email, otp });  
 if (!storedOtp) return res.status(400).json({ success: false, error: "Invalid or expired OTP." });  
  
 // OTP is valid, delete it from the database  
 await Otp.deleteMany({ email });  
 res.json({ success: true, error: "OTP verified successfully." });  
 } catch (error) {  
 console.error("OTP Verification Error:", error);  
 res.status(500).json({ success: false, error: "Error verifying OTP." });  
 }  
 });  
  
export default router;

# AlumniConnectAPI\routes\auth\userAuth.js

import express from 'express';  
import { body, validationResult } from 'express-validator';  
import bcrypt from 'bcryptjs';  
import jwt from 'jsonwebtoken';  
import dotenv from "dotenv";  
import Student from '../../models/student.js';  
import Teacher from '../../models/teacher.js';  
import fetchuser from '../../middleware/fetchuser.js';  
  
dotenv.config();  
  
const router = express.Router();  
const jwtSecretKey = process.env.JWTSECRETKEY;  
  
router.post('/:user/bulk-create',  
 async (req, res) => {  
 try {  
 const { user } = req.params;  
  
 // Check if request body is an array  
 if (!Array.isArray(req.body) || req.body.length === 0) {  
 return res.status(400).json({ success: false, error: "Invalid data format. Expecting a non-empty array." });  
 }  
  
 // Common validation rules for students  
 const studentValidations = req.body.map((student, index) => [  
 body(`${index}.password`, 'Enter a valid password').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.firstName`, 'Enter a valid First name').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.fatherName`, 'Enter a valid Middle name').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.lastName`, 'Enter a valid Last name').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.motherName`, 'Enter a valid Mother name').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.grandfatherName`, 'Enter a valid Grandfather name').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.prn`, 'Enter a valid PRN (10-15 digits)').trim().isInt().isLength({ min: 10, max: 15 }).escape(),  
 body(`${index}.moodleid`, 'Enter a valid Moodle ID (8-10 digits)').trim().isInt().isLength({ min: 8, max: 10 }).escape(),  
 body(`${index}.mobileno`, 'Enter a valid Mobile number (10 digits)').trim().isInt().isLength({ min: 10, max: 10 }).escape(),  
 body(`${index}.email`, 'Enter a valid Email').isEmail().escape(),  
 body(`${index}.address`, 'Enter a valid Address').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.placeOfBirth`, 'Enter a valid Place Of Birth').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.dateOfBirth`, 'Invalid date format. Use DD-MM-YYYY').trim().isLength({ min: 10 }).escape(),  
 body(`${index}.branch`, 'Enter a valid Branch').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.program`, 'Enter a valid Program').trim().isLength({ min: 2 }).escape(),  
 body(`${index}.yearOfJoining`, 'Enter a valid Year Of Joining (FE/DSE)').trim().isLength({ min: 2, max: 3 }).escape(),  
 body(`${index}.dateOfAdmission`, 'Invalid date format. Use DD-MM-YYYY').trim().isLength({ min: 10 }).escape(),  
 body(`${index}.lastCollegeAttended`, 'Enter a valid Last College').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.religion`, 'Enter a valid Religion').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.caste`, 'Enter a valid Caste').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.category`, 'Enter a valid Category').trim().isLength({ min: 2 }).escape(),  
 body(`${index}.nationality`, 'Enter a valid Nationality').trim().isLength({ min: 3 }).escape(),  
 ]).flat();  
  
 // Validation rules for teachers  
 const teacherValidations = req.body.map((teacher, index) => [  
 body(`${index}.password`, 'Enter a valid password').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.firstName`, 'Enter a valid First name').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.middleName`, 'Enter a valid Middle name').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.lastName`, 'Enter a valid Last name').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.moodleid`, 'Enter a valid Moodle ID').trim().isInt().isLength({ min: 8, max: 10 }).escape(),  
 body(`${index}.mobileno`, 'Enter a valid Mobile number (10 digits)').trim().isInt().isLength({ min: 10, max: 10 }).escape(),  
 body(`${index}.address`, 'Enter a valid Address').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.department`, 'Enter a valid Branch').trim().isLength({ min: 2 }).escape(),  
 body(`${index}.post`, 'Enter a valid Post').trim().isLength({ min: 3 }).escape(),  
 body(`${index}.email`, 'Enter a valid Email').isEmail().escape(),  
 ]).flat();  
   
  
 // Apply validations based on user type  
 if (user === 'student') {  
 await Promise.all(studentValidations.map(validation => validation.run(req)));  
 } else if (user === 'teacher') {  
 await Promise.all(teacherValidations.map(validation => validation.run(req)));  
 } else {  
 return res.status(400).json({ success: false, error: 'Select a valid user type (student or teacher).' });  
 }  
  
 // Check for validation errors  
 const errors = validationResult(req);  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success: false, errors: errors.array() });  
 }  
  
 // Insert data into the database  
 if (user === 'student') {  
 // Hash passwords for students  
 const students = await Promise.all(  
 req.body.map(async (student) => {  
 const hashedPassword = await bcrypt.hash(student.password, 10); // Hash password  
 return { ...student, password: hashedPassword };  
 })  
 );  
   
 await Student.insertMany(students);  
 res.status(201).json({ success: true, message: "Students added successfully!" });  
   
 } else if (user === 'teacher') {  
 // Hash passwords for teachers  
 const teachers = await Promise.all(  
 req.body.map(async (teacher) => {  
 const hashedPassword = await bcrypt.hash(teacher.password, 10); // Hash password  
 return { ...teacher, password: hashedPassword };  
 })  
 );  
   
 await Teacher.insertMany(teachers);  
 res.status(201).json({ success: true, message: "Teachers added successfully!" });  
 }  
  
 } catch (error) {  
 console.error("Error:", error);  
 res.status(500).json({ success: false, error: "Internal server error" });  
 }  
 }  
);  
  
// Endpoint 1: Creating a user (POST "/api/auth/student/createuser") - No login required  
router.post(  
 '/student/createuser',  
 [  
 // Validations for user data  
 body('password', 'Enter a valid password').trim().isLength({ min: 3 }).escape(),  
 body('firstName', 'Enter a valid First name').trim().isLength({ min: 3 }).escape(),  
 body('fatherName', 'Enter a valid middle name').trim().isLength({ min: 3 }).escape(),  
 body('lastName', 'Enter a valid Last name').trim().isLength({ min: 3 }).escape(),  
 body('motherName', 'Enter a valid Mother name').trim().isLength({ min: 3 }).escape(),  
 body('grandfatherName', 'Enter a valid Grandfather name').trim().isLength({ min: 3 }).escape(),  
 body('prn', 'Enter a valid PRN').trim().isInt().isLength({ min: 10, max: 15 }).escape(),  
 body('moodleid', 'Enter a valid moodle id').trim().isInt().isLength({ min: 8, max: 10 }).escape(),  
 body('mobileno', 'Enter a valid mobile number').trim().isInt().isLength({ min: 10, max: 10 }).escape(),  
 body('email', 'Enter a valid Email').isEmail().escape(),  
 body('address', 'Enter a valid Address').trim().isLength({ min: 3 }).escape(),  
 body('placeOfBirth', 'Enter a valid Place Of Birth').trim().isLength({ min: 3 }).escape(),  
 body('dateOfBirth').trim().notEmpty().withMessage('Date of birth is required'),  
 body('branch', 'Enter a valid Branch').trim().isLength({ min: 3 }).escape(),  
 body('program', 'Enter a valid Program').trim().isLength({ min: 2 }).escape(),  
 body('semester', 'Enter a valid semester').trim().isInt().isLength({ min: 1, max: 1 }).escape(),  
 body('yearOfJoining', 'Enter a valid Year Of Joining').trim().isLength({ min: 2, max: 3 }).escape(),  
 body('dateOfAdmission').trim().notEmpty().withMessage('Date Of Admission is required'),  
 body('lastCollegeAttended', 'Enter a valid Last College').trim().isLength({ min: 3 }).escape(),  
 body('religion', 'Enter a valid religion').trim().isLength({ min: 3 }).escape(),  
 body('caste', 'Enter a valid caste').trim().isLength({ min: 3 }).escape(),  
 body('category', 'Enter a valid category').trim().isLength({ min: 3 }).escape(),  
 body('nationality', 'Enter a valid nationality').trim().isLength({ min: 3 }).escape(),  
 ],  
 async (req, res) => {  
 let success = false;  
 const errors = validationResult(req);  
  
 // Check for validation errors  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 try {  
 // Check if email or username already exists in the database  
  
 const duplicateMoodleid = await Student.findOne({ moodleid: req.body.moodleid });  
 // console.log(duplicateMoodleid)  
 if (duplicateMoodleid) {  
 return res.status(400).json({ success, error: 'Student already exist' });  
 }  
  
 // Save user data to the database  
 const salt = await bcrypt.genSalt(10);  
 const hashedPassword = await bcrypt.hash(req.body.password, salt);  
 // console.log(hashedPassword)  
 // console.log(req.body)  
  
 const newStudent = {  
 ...req.body,  
 password: hashedPassword // Replace plain password with hashed one  
 };  
 await Student.create(newStudent);  
  
 success = true;  
 res.json({ success });  
 } catch (err) {  
 console.error(err.message);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
 }  
);  
  
router.post(  
 '/teacher/createuser',  
 [  
 // Validations for user data  
 body('password', 'Enter a valid password').trim().isLength({ min: 3 }).escape(),  
 body('firstName', 'Enter a valid First name').trim().isLength({ min: 3 }).escape(),  
 body('middleName', 'Enter a valid middle name').trim().isLength({ min: 3 }).escape(),  
 body('lastName', 'Enter a valid Last name').trim().isLength({ min: 3 }).escape(),  
 body('moodleid', 'Enter a valid moodle id').trim().isInt().isLength({ min: 8, max: 10 }).escape(),  
 body('mobileno', 'Enter a valid mobile number').trim().isInt().isLength({ min: 10, max: 10 }).escape(),  
 body('address', 'Enter a valid Address').trim().isLength({ min: 3 }).escape(),  
 body('department', 'Enter a valid Branch').trim().isLength({ min: 3 }).escape(),  
 body('post', 'Enter a valid Post').trim().isLength({ min: 3 }).escape(),  
 body('email', 'Enter a valid Email').isEmail().escape(),  
 ],  
 async (req, res) => {  
 let success = false;  
 const errors = validationResult(req);  
  
 // Check for validation errors  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 try {  
 // Check if email or username already exists in the database  
  
 const duplicateMoodleid = await Teacher.findOne({ moodleid: req.body.moodleid });  
 console.log(duplicateMoodleid)  
 if (duplicateMoodleid) {  
 return res.status(400).json({ success, error: 'Teacher already exist' });  
 }  
  
 // Save user data to the database  
 const salt = await bcrypt.genSalt(10);  
 const hashedPassword = await bcrypt.hash(req.body.password, salt);  
 console.log(hashedPassword)  
 // console.log(req.body)  
 await Teacher.create(req.body);  
  
 success = true;  
 res.json({ success });  
 } catch (err) {  
 console.error(err.message);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
 }  
);  
  
// Endpoint 2: Student login (POST "/api/auth/student/login") - No login required  
router.post(  
 '/:user/login',  
 [  
 body('moodleid', 'Enter a valid moodleid').trim().isNumeric().escape(),  
 body('password', 'Enter a valid password').trim().isLength({ min: 8 }).escape(),  
 ],  
 async (req, res) => {  
 let success = false;  
 const errors = validationResult(req);  
  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 const { moodleid, password } = req.body;  
 const { user } = req.params  
 try {  
 let storedUser  
 if (user === 'student') {  
 storedUser = await Student.findOne({ moodleid });  
 } else if (user === 'teacher') {  
 storedUser = await Teacher.findOne({ moodleid });  
 } else {  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
  
 if (!storedUser) {  
 return res.status(400).json({ success, error: 'Enter valid credentials' });  
 }  
  
 const isPasswordValid = await bcrypt.compare(password, storedUser.password);  
 if (!isPasswordValid) {  
 return res.status(400).json({ success, error: 'Enter valid credentials' });  
 }  
  
 const data = {  
 id: storedUser.id,  
 };  
  
 const token = jwt.sign(data, jwtSecretKey);  
 success = true;  
 res.json({ success, token });  
 } catch (err) {  
 console.error(err.message);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
 }  
);  
  
// Endpoint 3: Student forget password (POST "/api/auth/student/resetpass") - No login required  
router.put(  
 "/:user/resetpass",  
 [  
 body("moodleid", "Enter a valid Moodle ID").trim().isNumeric().escape(),  
 body("password")  
 .trim()  
 .isLength({ min: 8 })  
 .withMessage("Password must be at least 8 characters long")  
 .matches(/[A-Z]/)  
 .withMessage("Password must contain at least 1 uppercase letter")  
 .matches(/[a-z]/)  
 .withMessage("Password must contain at least 1 lowercase letter")  
 .matches(/\d/)  
 .withMessage("Password must contain at least 1 number")  
 .matches(/[@$!%\*?&]/)  
 .withMessage("Password must contain at least 1 special character (@, $, !, %, \*, ?, &)")  
 .escape(),  
 ],  
 async (req, res) => {  
 let success = false;  
 const errors = validationResult(req);  
  
 // Validate input  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 const { moodleid, password } = req.body;  
 const { user } = req.params  
 try {  
 // Check if student exists  
 let storedUser  
 if (user === 'student') {  
 storedUser = await Student.findOne({ moodleid });  
 } else if (user === 'teacher') {  
 storedUser = await Teacher.findOne({ moodleid });  
 } else {  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
  
 if (!storedUser) {  
 return res.status(400).json({ success, error: "Invalid Moodle ID" });  
 }  
  
 // Hash the new password before saving  
 const salt = await bcrypt.genSalt(10);  
 const hashedPassword = await bcrypt.hash(password, salt);  
  
 // Update password securely  
 if (user === 'student') {  
 await Student.findOneAndUpdate(  
 { moodleid },  
 { $set: { password: hashedPassword } },  
 { new: true }  
 );  
 } else if (user === 'teacher') {  
 await Teacher.findOneAndUpdate(  
 { moodleid },  
 { $set: { password: hashedPassword } },  
 { new: true }  
 );  
 } else {  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
  
 success = true;  
 res.json({ success, error: "Password updated successfully" });  
 } catch (err) {  
 console.error(err.message);  
 return res.status(500).json({ success, error: "Internal server error" });  
 }  
 }  
);  
  
// Endpoint 4: Student forget password (POST "/api/auth/student/resetpass") - No login required  
router.put(  
 "/:user/setpass",  
 [  
 body('mobileno', 'Enter a valid mobile number').trim().isNumeric().escape(),  
 body('email', 'Enter a valid Email').isEmail().escape(),  
 body("password")  
 .trim()  
 .isLength({ min: 8 })  
 .withMessage("Password must be at least 8 characters long")  
 .matches(/[A-Z]/)  
 .withMessage("Password must contain at least 1 uppercase letter")  
 .matches(/[a-z]/)  
 .withMessage("Password must contain at least 1 lowercase letter")  
 .matches(/\d/)  
 .withMessage("Password must contain at least 1 number")  
 .matches(/[@$!%\*?&]/)  
 .withMessage("Password must contain at least 1 special character (@, $, !, %, \*, ?, &)")  
 .escape(),  
 ],  
 fetchuser,  
 async (req, res) => {  
 let success = false;  
 const errors = validationResult(req);  
  
 // Validate input  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 const userId = req.userId;  
 const { mobileno, email, password } = req.body;  
 const { user } = req.params  
 try {  
 // Check if student exists  
 let storedUser  
 if (user === 'student') {  
 storedUser = await Student.findById(userId).select('moodleid');  
 } else if (user === 'teacher') {  
 storedUser = await Teacher.findById(userId).select('moodleid');  
 } else {  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
  
 if (!storedUser) {  
 return res.status(400).json({ success, error: "Invalid Moodle ID" });  
 }  
 const moodleid = storedUser.moodleid  
 // Hash the new password before saving  
 const salt = await bcrypt.genSalt(10);  
 const hashedPassword = await bcrypt.hash(password, salt);  
  
 // Update password securely  
 if (user === 'student') {  
 await Student.findOneAndUpdate(  
 { moodleid },  
 { $set: { password: hashedPassword, email: email, mobileno: mobileno, firstLogin: false } },  
 { new: true }  
 );  
 } else if (user === 'teacher') {  
 await Teacher.findOneAndUpdate(  
 { moodleid },  
 { $set: { password: hashedPassword, email: email, mobileno: mobileno, firstLogin: false } },  
 { new: true }  
 );  
 } else {  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
  
 success = true;  
 res.json({ success, error: "Password updated successfully" });  
 } catch (err) {  
 console.error(err.message);  
 return res.status(500).json({ success, error: "Internal server error" });  
 }  
 }  
);  
  
// Endpoint 5: Fetch user data (POST "/api/auth/student/getuser") - Login required  
  
router.post('/:user/getuser', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const { user } = req.params  
 const userId = req.userId;  
 // console.log(userId)  
 let userData  
 if (user === 'student') {  
 userData = await Student.findById(userId).select('-password');  
 } else if (user === 'teacher') {  
 userData = await Teacher.findById(userId).select('-password');  
 } else {  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
 // console.log(userData)  
 success = true;  
 res.json({ success, userData });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
// Endpoint 6: Fetch user first login (POST "/api/auth/student/getfirstlogin") - Login required  
router.post('/:user/getfirstlogin', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const { user } = req.params  
 const userId = req.userId;  
 let firstLogin  
  
 if (user === 'student') {  
 firstLogin = await Student.findById(userId).select('firstLogin');  
 } else if (user === 'teacher') {  
 firstLogin = await Teacher.findById(userId).select('firstLogin');  
 } else {  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
 success = true;  
 res.json(firstLogin);  
 } catch (err) {  
 console.error(err.message);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
// Endpoint 7: update user data user data (PUT "/api/auth/student/updateuserinfo") - Login required  
  
router.put('/:user/updateuserinfo', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const { user } = req.params  
 const userId = req.userId;  
 // console.log(userId)  
  
 let { firstName, middleName, lastName, mobileno, email, address } = req.body  
  
 const newData = {}  
 if(firstName){ newData.firstName = firstName }  
 if(lastName){ newData.lastName = lastName }  
 if(middleName){ newData.middleName = middleName }  
 if(mobileno){ newData.mobileno = mobileno }  
 if(email){ newData.email = email }  
 if(address){ newData.address = address }  
  
 let userData  
 if (user === 'student') {  
 let userData = await Student.findById(userId)  
 if(!userData){  
 return res.status(404).send('Student Not Found')  
 }  
 userData = await Student.findByIdAndUpdate(userId, {$set: newData}, {new:true})  
 } else if (user === 'teacher') {  
 let userData = await Teacher.findById(userId)  
 if(!userData){  
 return res.status(404).send('Teacher Not Found')  
 }  
 userData = await Teacher.findByIdAndUpdate(userId, {$set: newData}, {new:true})  
 } else {  
 return res.status(400).json({ success, error: 'Select user type' });  
 }  
  
 console.log(newData)  
  
 success = true;  
 res.json({ success, userData });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
export default router;

# AlumniConnectAPI\routes\files\files.js

import express from 'express';  
import multer from 'multer';  
import path from 'path';  
import fs from 'fs';  
import { fileURLToPath } from 'url';  
// import jwt from 'jsonwebtoken';  
import Student from '../../models/student.js';  
import Teacher from '../../models/teacher.js';  
import LC from '../../models/leavingcertificate.js'  
import SubmittedForms from '../../models/SubmittedForms.js'  
import fetchuser from '../../middleware/fetchuser.js';  
  
const router = express.Router();  
const \_\_filename = fileURLToPath(import.meta.url);  
const \_\_dirname = path.dirname(\_\_filename);  
  
// Configure multer for file storage  
const storage = multer.diskStorage({  
 destination: (req, file, cb) => {  
 const uploadDir = `uploads/${req.headers.location}`;  
 // Create directory if it doesn't exist  
 if (!fs.existsSync(uploadDir)) {  
 fs.mkdirSync(uploadDir, { recursive: true });  
 }  
 cb(null, uploadDir);  
 },  
 filename: async (req, file, cb) => {  
 try {  
 const userId = req.userId  
  
 // Generate filename with user ID  
 cb(null, `${req.headers.location}-${userId}${path.extname(file.originalname)}`);  
 } catch (error) {  
 cb(new Error('Invalid token or user not found'));  
 }  
 }  
});  
  
// Configure multer upload  
const uploadImage = multer({  
 storage,  
 limits: {  
 fileSize: 5 \* 1024 \* 1024 // 5MB limit  
 },  
 fileFilter: (req, file, cb) => {  
 // Accept only images  
 if (file.mimetype.startsWith('image/')) {  
 cb(null, true);  
 } else {  
 cb(new Error('Only image files are allowed!'));  
 }  
 }  
});  
  
const uploadPdf = multer({  
 storage,  
 limits: {  
 fileSize: 5 \* 1024 \* 1024 // 5MB limit  
 },  
 fileFilter: (req, file, cb) => {  
 // Accept only PDF files  
 if (file.mimetype === 'application/pdf') {  
 cb(null, true);  
 } else {  
 cb(new Error('Only PDF files are allowed!'));  
 }  
 }  
});  
  
  
  
// Upload profile picture  
router.post('/upload-profile/:user', fetchuser, uploadImage.single('profile'), async (req, res) => {  
 try {  
 const userId = req.userId  
 if (!req.file) {  
 return res.status(400).json({ success: false, error: 'No file uploaded' });  
 }  
  
 // Get user type from URL parameter  
 const userType = req.params.user;  
 let UserModel;  
  
 // Determine which model to use based on user type  
 if (userType === 'student') {  
 UserModel = Student;  
 } else if (userType === 'teacher') {  
 UserModel = Teacher;  
 } else {  
 return res.status(400).json({ success: false, error: 'Invalid user type' });  
 }  
  
 // Update user's profile picture path in database  
 await UserModel.findByIdAndUpdate(userId, {  
 profilePicture: req.file.filename  
 });  
  
 // Return the file path  
 res.json({  
 success: true,  
 filePath: req.file.path,  
 fileName: req.file.filename  
 });  
 } catch (error) {  
 console.error('Error uploading profile:', error);  
 res.status(500).json({ success: false, error: 'Error uploading profile' });  
 }  
});  
  
  
// Get profile picture  
router.get('/profile/:user', fetchuser, async(req, res) => {  
 try {  
 const userId = req.userId  
  
 const userType = req.params.user;  
 let UserModel;  
 if (userType === 'student') {  
 UserModel = Student;  
 } else if (userType === 'teacher') {  
 UserModel = Teacher;  
 } else {  
 return res.status(400).json({ success: false, error: 'Invalid user type' });  
 }  
  
 let profile = await UserModel.findById(userId).select('profilePicture');  
 // console.log(profile.profilePicture)  
  
 const filePath = path.join(\_\_dirname, '../../uploads/profiles', profile.profilePicture);  
 if (fs.existsSync(filePath)) {  
 res.sendFile(filePath);  
 } else {  
 res.status(404).json({ success: false, error: 'File not found' });  
 }  
 } catch (error) {  
 console.error('Error getting profile:', error);  
 res.status(500).json({ success: false, error: 'Error getting profile' });  
 }  
});  
  
// Upload Fees Reciept   
router.post('/upload-fees-reciept/:formId', fetchuser, uploadPdf.single('fees-reciept'), async (req, res) => {  
 try {  
 const formId = req.params.formId;  
 if (!req.file) {  
 return res.status(400).json({ success: false, error: 'No file uploaded' });  
 }  
   
 const form = await SubmittedForms.findById(formId);  
 if (!form) {  
 return res.status(404).json({ success: false, error: 'Form not found' });  
 }  
  
 console.log(form.docId)  
 const updatedLc = await LC.findByIdAndUpdate(form.docId,{  
 feesReciept : req.file.filename  
 })  
   
 if(updatedLc){  
 await SubmittedForms.findByIdAndUpdate(formId,{  
 feesRecieptUploded:true  
 })  
 // Return the file path  
 res.json({  
 success: true,  
 filePath: req.file.path,  
 fileName: req.file.filename  
 });  
 }  
 } catch (error) {  
 console.error('Error uploading profile:', error);  
 res.status(500).json({ success: false, error: 'Error uploading profile' });  
 }  
});  
  
// Get profile picture  
router.get('/get-fees-reciept/:formId', fetchuser, async(req, res) => {  
 try {  
 const userId = req.userId  
 const formId = req.params.formId;  
  
 const form = await SubmittedForms.findById(formId);  
 if (!form) {  
 return res.status(404).json({ success: false, error: 'Form not found' });  
 }  
   
 let doc = await LC.findById(form.docId).select('feesReciept');  
 console.log(doc.feesReciept)  
  
 const filePath = path.join(\_\_dirname, '../../uploads/fees-reciept', doc.feesReciept);  
 if (fs.existsSync(filePath)) {  
 res.sendFile(filePath);  
 } else {  
 res.status(404).json({ success: false, error: 'File not found' });  
 }  
 } catch (error) {  
 console.error('Error getting profile:', error);  
 res.status(500).json({ success: false, error: 'Error getting profile' });  
 }  
});  
  
// Upload Varification Document   
router.post('/upload-verification-document/:formId', fetchuser, uploadPdf.single('verification-document'), async (req, res) => {  
 try {  
 const formId = req.params.formId;  
 if (!req.file) {  
 return res.status(400).json({ success: false, error: 'No file uploaded' });  
 }  
   
 const form = await SubmittedForms.findById(formId);  
 if (!form) {  
 return res.status(404).json({ success: false, error: 'Form not found' });  
 }  
  
   
  
 const updatedLc = await LC.findByIdAndUpdate(form.docId,{  
 varificationDoc : req.file.filename  
 })  
  
 if(updatedLc){  
 await SubmittedForms.findByIdAndUpdate(formId,{  
 varificationDocUploded:true  
 })  
 // Return the file path  
 res.json({  
 success: true,  
 filePath: req.file.path,  
 fileName: req.file.filename  
 });  
 }  
 } catch (error) {  
 console.error('Error uploading profile:', error);  
 res.status(500).json({ success: false, error: 'Error uploading profile' });  
 }  
});  
  
// Get verification document  
router.get('/get-verification-document/:formId', fetchuser, async(req, res) => {  
 try {  
 const userId = req.userId  
 const formId = req.params.formId;  
  
 const form = await SubmittedForms.findById(formId);  
 if (!form) {  
 return res.status(404).json({ success: false, error: 'Form not found' });  
 }  
  
 let doc = await LC.findById(form.docId).select('varificationDoc');  
 // console.log(profile.profilePicture)  
  
 const filePath = path.join(\_\_dirname, '../../uploads/verification-document', doc.varificationDoc);  
 if (fs.existsSync(filePath)) {  
 res.sendFile(filePath);  
 } else {  
 res.status(404).json({ success: false, error: 'File not found' });  
 }  
 } catch (error) {  
 console.error('Error getting profile:', error);  
 res.status(500).json({ success: false, error: 'Error getting profile' });  
 }  
});  
export default router;

# AlumniConnectAPI\routes\forms\lc.js

import express from 'express';  
import { body, validationResult } from 'express-validator';  
import bcrypt from 'bcryptjs';  
import jwt from 'jsonwebtoken';  
import dotenv from "dotenv";  
import Student from '../../models/student.js';  
import Teacher from '../../models/teacher.js';  
import fetchuser from '../../middleware/fetchuser.js';  
import LC from '../../models/leavingcertificate.js'  
import SubmittedForms from '../../models/SubmittedForms.js'  
  
dotenv.config();  
  
const router = express.Router();  
const jwtSecretKey = process.env.JWTSECRETKEY;  
  
  
  
// Endpoint 1: Creating a user LC (POST "/api/forms/createlc") - No login required  
router.post(  
 '/createlc', fetchuser,  
 [  
 // Validations for user data  
 body('firstName', 'Enter a valid First name').trim().isLength({ min: 3 }).escape(),  
 body('fatherName', 'Enter a valid middle name').trim().isLength({ min: 3 }).escape(),  
 body('lastName', 'Enter a valid Last name').trim().isLength({ min: 3 }).escape(),  
 body('motherName', 'Enter a valid Mother name').trim().isLength({ min: 3 }).escape(),  
 body('grandfatherName', 'Enter a valid Grandfather name').trim().isLength({ min: 3 }).escape(),  
 body('prn', 'Enter a valid PRN').trim().isInt().isLength({ min: 10, max: 15 }).escape(),  
 body('moodleid', 'Enter a valid moodle id').trim().isInt().isLength({ min: 8, max: 10 }).escape(),  
 body('placeOfBirth', 'Enter a valid Place Of Birth').trim().isLength({ min: 3 }).escape(),  
 body('dateOfBirth').trim().notEmpty().withMessage('Date of birth is required'),  
 body('branch', 'Enter a valid Branch').trim().isLength({ min: 3 }).escape(),  
 body('semester', 'Enter a valid semester').trim().isInt().isLength({ min: 1, max: 1 }).escape(),  
 body('yearOfJoining', 'Enter a valid Year Of Joining').trim().isLength({ min: 2, max: 3 }).escape(),  
 body('dateOfAdmission', 'Invalid date format. Use DD-MM-YYYY').trim().isLength({ min: 10 }).escape(),  
 body('lastCollegeAttended', 'Enter a valid Last College').trim().isLength({ min: 3 }).escape(),  
 body('religion', 'Enter a valid religion').trim().isLength({ min: 3 }).escape(),  
 body('caste', 'Enter a valid caste').trim().isLength({ min: 3 }).escape(),  
 body('category', 'Enter a valid category').trim().isLength({ min: 2 }).escape(),  
 body('nationality', 'Enter a valid nationality').trim().isLength({ min: 3 }).escape(),  
 ],  
 async (req, res) => {  
 let success = false;  
 const errors = validationResult(req);  
  
 // Check for validation errors  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 try {  
 // Check if username already exists in the database  
 const userId = req.userId  
 const duplicateMoodleid = await LC.findOne({ userId: userId });  
 // console.log(duplicateMoodleid)  
 if (duplicateMoodleid) {  
 return res.status(400).json({ success, error: 'Lc already exist check progress of lc in check status window' });  
 }  
  
 // Save LC data to the database  
 const newLC = {  
 ...req.body,  
 userId:userId  
 }  
 const lc = await LC.create(newLC);  
 const lcId = lc.id  
 // console.log(lcId)  
  
 const newForm = {  
 docType:"Leaving Certificate",  
 docId:lcId,  
 userId:userId,  
 branch:req.body.branch  
 }  
  
 const doc = await SubmittedForms.create(newForm)  
  
 success = true;  
 res.json({ success });  
 } catch (err) {  
 console.error(err.message);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
 }  
);  
  
// Endpoint 4: Create Letter of Recommendation (POST "/api/forms/lor/create") - Login required  
router.post('/lor/create', fetchuser, [  
 body('branch', 'Branch is required').notEmpty().trim().escape(),  
 body('program', 'Program is required').notEmpty().trim().escape(),  
 body('semester', 'Semester is required').notEmpty().trim().escape(),  
 body('moodleid', 'Moodle ID is required').notEmpty().trim().escape(),  
 body('firstName', 'First name is required').notEmpty().trim().escape(),  
 body('lastName', 'Last name is required').notEmpty().trim().escape(),  
 body('reason', 'Reason is required').notEmpty().trim().escape(),  
 body('teacher1', 'Teacher 1 is required').notEmpty().trim().escape(),  
 body('teacher2', 'Teacher 2 is required').notEmpty().trim().escape(),  
 body('teacher3', 'Teacher 3 is required').notEmpty().trim().escape(),  
 body('teacher4', 'Teacher 4 is required').notEmpty().trim().escape(),  
], async (req, res) => {  
 let success = false;  
 const errors = validationResult(req);  
  
 // Check for validation errors  
 if (!errors.isEmpty()) {  
 return res.status(400).json({ success, errors: errors.array() });  
 }  
  
 try {  
 // Check if username already exists in the database  
 const userId = req.userId;  
 const duplicateMoodleid = await LOR.findOne({ userId: userId });  
 if (duplicateMoodleid) {  
 return res.status(400).json({ success, error: 'LOR already exists, check progress of LOR in check status window' });  
 }  
  
 // Save LOR data to the database  
 const newLOR = {  
 ...req.body,  
 userId: userId  
 };  
 const lor = await LOR.create(newLOR);  
 const lorId = lor.id;  
  
 const newForm = {  
 docType: "Letter of Recommendation",  
 docId: lorId,  
 userId: userId,  
 branch: req.body.branch,  
 teachers: [req.body.teacher1, req.body.teacher2, req.body.teacher3, req.body.teacher4], // Save the teachers as an array  
 principal: req.body.principal,  
 hod: req.body.hod  
 };  
  
 const doc = await SubmittedForms.create(newForm);  
  
 success = true;  
 res.json({ success });  
 } catch (err) {  
 console.error(err.message);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
  
  
// Endpoint 2: Fetch forms for student (POST "/api/forms/student/getuser") - Login required  
  
router.post('/student/getforms', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const { user } = req.params  
 const userId = req.userId;  
 // console.log(userId)  
 let forms = await SubmittedForms.find({userId:userId});  
 // console.log(userData)  
 success = true;  
 res.json({ success, forms });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
// Endpoint 3: Fetch forms for teachers (POST "/api/forms/teacher/getforms") - Login required  
router.post('/teacher/getforms', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const userId = req.userId;  
 console.log(userId);  
 let teacher = await Teacher.findById(userId).select('department');  
 console.log(teacher.department);  
   
 if(!teacher.department) {  
 return res.json({ success, forms: [], message: "No department assigned" });  
 }  
  
 let teacherForms;  
 if(teacher.department === 'admin') {  
 teacherForms = await SubmittedForms.find({ status: 'Submitted' });  
 } else if(teacher.department === 'examCell') {  
 teacherForms = await SubmittedForms.find({ admin: true });  
 } else if(teacher.department === 'library') {  
 teacherForms = await SubmittedForms.find({ admin: true, examCell: true });  
 } else {  
 teacherForms = await SubmittedForms.find({ admin: true, examCell: true, library: true, branch: teacher.department });  
 }  
  
 // console.log('Fetched forms:', teacherForms);  
 success = true;  
 res.json({ success, forms: teacherForms });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
// Endpoint 4: to fetch LC data by ID (GET "/api/forms/lc/:id")  
router.get('/:docType/:id', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 // console.log(req.params.docType)  
 const docType = req.params.docType;  
 const id = req.params.id;  
 let doc;  
 if(docType === 'lc') {  
 doc = await LC.findById(id);  
 }else if(docType === 'lor') {  
 doc = await LOR.findById(id);  
 }  
 if (!doc) {  
 return res.status(404).json({ success, error: 'Document not found' });  
 }  
 success = true;  
 res.json({ success, doc });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
// Endpoint5: to update form status (PUT "/api/forms/updatestatus/:formId")  
router.put('/updatestatus/:formId', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const formId = req.params.formId;  
 const userId = req.userId;  
  
 // Get teacher's department  
 let teacher = await Teacher.findById(userId).select('department');  
 if (!teacher || !teacher.department) {  
 return res.status(400).json({ success, error: 'No department assigned to teacher' });  
 }  
  
 // Find the form  
 const form = await SubmittedForms.findById(formId);  
 if (!form) {  
 return res.status(404).json({ success, error: 'Form not found' });  
 }  
  
 // Update based on department  
 let updateData = {};  
 switch (teacher.department) {  
 case 'admin':  
 console.log(req.body.action)  
 if(req.body.action === 'approve') {  
 updateData = { admin: true, adminRemarks: "Form approved by Admin", adminFeesPaid: true, documentVerified: true, feesRecieptUploded:false, varificationDocUploded:false, examCellDate: Date.now() };  
 }else if(req.body.action === 'fees') {  
 updateData = { admin: false, adminFeesPaid: false, adminRemarks: "Pay fees and upload the receipt", feesRecieptUploded:false };  
 }else if(req.body.action === 'details') {  
 console.log("details")  
 updateData = { admin: false, documentVerified: false, adminRemarks: "Incorrect form details, please provide the correct details by cancelling the form and re-submitting it or provide the document proof to varify the provided details are correct", varificationDocUploded:false};  
 }  
  
 break;  
 case 'examCell':  
 // Only allow if admin has approved  
 if (!form.admin) {  
 return res.status(400).json({ success, error: 'Admin approval pending' });  
 }  
 if(req.body.action === 'approve') {  
 updateData = { examCell: true, examCellRemarks: "Form approved by Exam Cell", examCellFeesPaid: true, feesRecieptUploded:false, feesRecieptUploded:false, varificationDocUploded:false, libraryDate: Date.now() };  
 }else if(req.body.action === 'fees') {  
 updateData = { examCell: false, examCellFeesPaid: false, examCellRemarks: "Pay fees and upload the receipt", varificationDocUploded:false };  
 }  
 break;  
 case 'library':  
 // Only allow if admin and examCell have approved  
 if (!form.admin || !form.examCell) {  
 return res.status(400).json({ success, error: 'Previous approvals pending' });  
 }  
 if(req.body.action === 'approve') {  
 updateData = { library: true, libraryRemarks: "Form approved by Library", libraryFeesPaid: true, feesRecieptUploded:false, feesRecieptUploded:false, varificationDocUploded:false, atBranchDate: Date.now() };  
 }else if(req.body.action === 'fees') {  
 updateData = { library: false, libraryFeesPaid: false, libraryRemarks: "Pay fees and upload the receipt", varificationDocUploded:false };  
 }  
 break;  
 default:  
 // For branch verification  
 // Only allow if admin, examCell and library have approved  
 if (!form.admin || !form.examCell || !form.library) {  
 return res.status(400).json({ success, error: 'Previous approvals pending' });  
 }  
 if(req.body.action === 'approve') {  
 updateData = { atBranch: true, atBranchRemarks: "Form approved by Branch", collected: true, collectionDate: Date.now() };  
 }else if(req.body.action === 'fees') {  
 updateData = { atBranch: false, atBranchFeesPaid: false, atBranchRemarks: "Pay fees and upload the receipt" };  
 }  
 //add branch level verification  
 updateData = { atBranch: true };  
 }  
  
 // Update the form status  
 const updatedForm = await SubmittedForms.findByIdAndUpdate(  
 formId,  
 { $set: updateData },  
 { new: true }  
 );  
  
 success = true;  
 res.json({ success, form: updatedForm });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
  
// Endpoint 6: to cancel form (PUT "/api/forms/cancel/:formId")  
router.put('/cancel/:formId', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const formId = req.params.formId;   
 const userId = req.userId;  
  
 const form = await SubmittedForms.findById(formId);  
 if (!form) {  
 return res.status(404).json({ success, error: 'Form not found' });  
 }   
  
 form.userId = form.userId;  
 // console.log(form.userId.toString(), userId.toString())  
 if (form.userId.toString() !== userId.toString()) {  
 return res.status(400).json({ success, error: 'Unauthorized to cancel this form' });  
 }  
 await SubmittedForms.deleteOne({ \_id: formId });  
 await LC.deleteOne({ \_id: form.docId });  
  
 success = true;  
 res.json({ success });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
export default router;

# AlumniConnectAPI\routes\notification\notification.js

import express from 'express';  
import { body, validationResult } from 'express-validator';  
import bcrypt from 'bcryptjs';  
import jwt from 'jsonwebtoken';  
import dotenv from "dotenv";  
import Student from '../../models/student.js';  
import Teacher from '../../models/teacher.js';  
import fetchuser from '../../middleware/fetchuser.js';  
import LC from '../../models/leavingcertificate.js'  
import SubmittedForms from '../../models/SubmittedForms.js'  
import Notification from '../../models/Notification.js';  
  
dotenv.config();  
const router = express.Router();  
  
// Add notification endpoint  
router.post('/notify', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const { formId, message, title } = req.body;  
   
 // Find the form  
 const form = await SubmittedForms.findById(formId);  
 if (!form) {  
 return res.status(404).json({ success, error: 'Form not found' });  
 }  
 // console.log(form.userId)  
  
 // Get teacher's department  
 const teacher = await Teacher.findById(req.userId);  
 if (!teacher) {  
 return res.status(404).json({ success, error: 'Teacher not found' });  
 }  
  
 const userId = form.userId  
 // Create notification  
 const notification = {  
 userId,  
 title,  
 message,  
 from: teacher.department,  
 timestamp: new Date(),  
 isRead: false  
 };  
  
 // Update student's notifications  
 await Notification.create(  
 notification  
 );  
  
 success = true;  
 res.json({ success, message: 'Notification sent successfully' });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
// send notification to everyone endpoint  
router.post('/notifyall', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const { message, title } = req.body;  
  
 // Get teacher's department  
 const teacher = await Teacher.findById(req.userId);  
 if (!teacher) {  
 return res.status(404).json({ success, error: 'Teacher not found' });  
 }  
  
 // Create notification  
 const notification = {  
 title,  
 message,  
 from: teacher.department,  
 timestamp: new Date(),  
 isRead: false  
 };  
  
 // Update student's notifications  
 await Notification.create(  
 notification  
 );  
  
 success = true;  
 res.json({ success, message: 'Notification sent successfully' });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
// Add endpoint to get notifications for a student  
router.get('/getnotifications', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const userId = req.userId;  
 // console.log("1. User ID:", userId);  
  
 // Find the student  
 const student = await Student.findById(userId);  
 // console.log("2. Found student:", student);  
   
 if (!student) {  
 // console.log("3. Student not found for ID:", userId);  
 return res.status(404).json({ success, error: 'Student not found' });  
 }  
  
 const notifications = await Notification.find({  
 $or: [  
 { userId: { $exists: false } }, // common for all users  
 { userId: userId } // specific to a user  
 ]  
 });  
   
 // console.log(notifications)  
  
 // Sort notifications by timestamp if they exist  
 if (notifications.length > 0) {  
 notifications.sort((a, b) => new Date(b.timestamp) - new Date(a.timestamp));  
 }  
  
 // Calculate unread notifications  
 const unreadCount = notifications.filter(n => !n.isRead).length;  
  
 success = true;  
 res.json({   
 success,   
 notifications,  
 count: notifications.length,  
 unreadCount  
 });  
   
 } catch (err) {  
 console.error("ERROR in getnotifications:", err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
// Add endpoint to mark notification as read  
router.put('/markread/:notificationId', fetchuser, async (req, res) => {  
 let success = false;  
 try {  
 const { notificationId } = req.params;  
 // const userId = req.userId;  
  
 // Find the student and update the specific notification  
 const notification = await Notification.findById(notificationId);  
   
 if (!notification) {  
 return res.status(404).json({ success, error: 'Notification not found' });  
 }  
  
 // // Find and update the specific notification  
 // const notification = student.notifications.id(notificationId);  
 // if (!notification) {  
 // return res.status(404).json({ success, error: 'Notification not found' });  
 // }  
  
 // Mark as read  
 await Notification.findOneAndUpdate(  
 { \_id: notificationId }, // <-- Filter (no $set here!)  
 { $set: { isRead: true } }, // <-- $set ONLY here  
 { new: true } // <-- Optional: returns updated document  
 );  
 // notification.isRead = true;  
 // await student.save();  
  
 success = true;  
 res.json({ success, message: 'Notification marked as read' });  
 } catch (err) {  
 console.error(err);  
 return res.status(500).json({ success, error: 'Internal server error' });  
 }  
});  
  
// Add endpoint to get unread notification count  
// router.get('/unread/count', fetchuser, async (req, res) => {  
// let success = false;  
// try {  
// const userId = req.userId;  
// const count = await Notification.countDocuments({   
// userId,   
// isRead: false   
// });  
  
// success = true;  
// res.json({ success, count });  
// } catch (err) {  
// console.error(err);  
// return res.status(500).json({ success, error: 'Internal server error' });  
// }  
// });  
  
export default router;

# AlumniConnectApp\index.js

/\*\*  
 \* @format  
 \*/  
  
import {AppRegistry} from 'react-native';  
import App from './src/App';  
import {name as appName} from './app.json';  
  
AppRegistry.registerComponent(appName, () => App);

# AlumniConnectApp\src\App.tsx

import { SafeAreaView, StyleSheet, View } from 'react-native'  
import React, { useEffect, useState } from 'react'  
import Header from './components/Header'  
import { UserProvider, useUser } from './context/user/UserContext'  
import { StatusBar } from 'react-native'  
import { NavigationContainer } from '@react-navigation/native'  
import { createNativeStackNavigator } from '@react-navigation/native-stack'  
import Login from './components/Login'  
import VerifyContact from './components/VerifyContact'  
import CheckStatus from './components/CheckStatus'  
import Profile from './components/Profile'  
import Notices from './components/Notices'  
  
export type RootStackParamList = {  
 Login: undefined  
 VerifyContact: undefined  
 CheckStatus: undefined  
 Profile: undefined  
 Notices: undefined  
}  
  
const Stack = createNativeStackNavigator<RootStackParamList>()  
  
const VerifyContactWrapper = () => {  
 const { host, token, user } = useUser();  
 return <VerifyContact host={host} token={token || ''} user={user || 'student'} />;  
}  
  
const ScreenLayout = ({ children, showNavbar = true }: { children: React.ReactNode, showNavbar?: boolean }) => {  
 return (  
 <View style={styles.container}>  
 <Header showNavbar={showNavbar} />  
 {children}  
 </View>  
 );  
};  
  
const NavigationWrapper = () => {  
 const { isLogin, isFirstLogin } = useUser();  
 const [initialized, setInitialized] = useState(false);  
  
 useEffect(() => {  
 // Wait for auth state to be initialized  
 setInitialized(true);  
 }, [isLogin, isFirstLogin]);  
  
 if (!initialized) {  
 return null; // Or a loading screen  
 }  
  
 return (  
 <Stack.Navigator   
 initialRouteName={isLogin ? (isFirstLogin ? "VerifyContact" : "CheckStatus") : "Login"}  
 screenOptions={{  
 headerShown: false,  
 }}  
 >  
 <Stack.Screen name="Login">  
 {() => (  
 <ScreenLayout showNavbar={false}>  
 <Login />  
 </ScreenLayout>  
 )}  
 </Stack.Screen>  
 <Stack.Screen name="VerifyContact">  
 {() => (  
 <ScreenLayout showNavbar={false}>  
 <VerifyContactWrapper />  
 </ScreenLayout>  
 )}  
 </Stack.Screen>  
 <Stack.Screen name="CheckStatus">  
 {() => (  
 <ScreenLayout>  
 <CheckStatus />  
 </ScreenLayout>  
 )}  
 </Stack.Screen>  
 <Stack.Screen name="Profile">  
 {() => (  
 <ScreenLayout>  
 <Profile />  
 </ScreenLayout>  
 )}  
 </Stack.Screen>  
 <Stack.Screen name="Notices">  
 {() => (  
 <ScreenLayout>  
 <Notices />  
 </ScreenLayout>  
 )}  
 </Stack.Screen>  
 </Stack.Navigator>  
 );  
};  
  
const App = () => {  
 return (  
 <UserProvider>  
 <NavigationContainer>  
 <SafeAreaView style={styles.safeArea}>  
 <StatusBar   
 barStyle="dark-content"  
 backgroundColor="#ffffff"  
 />  
 <NavigationWrapper />  
 </SafeAreaView>  
 </NavigationContainer>  
 </UserProvider>  
 )  
}  
  
export default App  
  
const styles = StyleSheet.create({  
 safeArea: {  
 flex: 1,  
 backgroundColor: '#ffffff',  
 },  
 container: {  
 flex: 1,  
 backgroundColor: '#ffffff',  
 },  
})

# AlumniConnectApp\src\components\Application.tsx

import { StyleSheet, Text, View, TouchableOpacity } from 'react-native';  
import React from 'react';  
import { Form } from '../types/form';  
  
interface ApplicationProps {  
 fname: string;  
 fstatus: string;  
 fdate: string;  
 formData: Form;  
}  
  
const Application: React.FC<ApplicationProps> = ({ fname, fstatus, fdate, formData }) => {  
 return (  
 <View style={styles.container}>  
 <View style={styles.header}>  
 <Text style={styles.title}>{fname}</Text>  
 <View style={styles.statusContainer}>  
 <Text style={[  
 styles.status,  
 fstatus === 'approved' && styles.statusApproved,  
 fstatus === 'rejected' && styles.statusRejected,  
 fstatus === 'pending' && styles.statusPending  
 ]}>  
 {fstatus}  
 </Text>  
 </View>  
 </View>  
   
 <View style={styles.details}>  
 <Text style={styles.label}>Submission Date:</Text>  
 <Text style={styles.value}>{fdate}</Text>  
 </View>  
  
 {formData.remarks && (  
 <View style={styles.details}>  
 <Text style={styles.label}>Remarks:</Text>  
 <Text style={styles.value}>{formData.remarks}</Text>  
 </View>  
 )}  
  
 {formData.documents && formData.documents.length > 0 && (  
 <View style={styles.documents}>  
 <Text style={styles.label}>Documents:</Text>  
 {formData.documents.map((doc, index) => (  
 <TouchableOpacity   
 key={index}   
 style={styles.documentItem}  
 onPress={() => {  
 // Handle document opening  
 console.log('Opening document:', doc.url);  
 }}  
 >  
 <Text style={styles.documentName}>{doc.name}</Text>  
 </TouchableOpacity>  
 ))}  
 </View>  
 )}  
 </View>  
 );  
};  
  
const styles = StyleSheet.create({  
 container: {  
 backgroundColor: '#fff',  
 borderRadius: 10,  
 padding: 15,  
 marginBottom: 15,  
 shadowColor: '#000',  
 shadowOffset: {  
 width: 0,  
 height: 2,  
 },  
 shadowOpacity: 0.1,  
 shadowRadius: 3,  
 elevation: 3,  
 },  
 header: {  
 flexDirection: 'row',  
 justifyContent: 'space-between',  
 alignItems: 'center',  
 marginBottom: 10,  
 },  
 title: {  
 fontSize: 18,  
 fontWeight: 'bold',  
 color: '#333',  
 },  
 statusContainer: {  
 paddingHorizontal: 10,  
 paddingVertical: 5,  
 borderRadius: 15,  
 },  
 status: {  
 fontSize: 14,  
 fontWeight: '600',  
 textTransform: 'capitalize',  
 },  
 statusApproved: {  
 color: '#28a745',  
 },  
 statusRejected: {  
 color: '#dc3545',  
 },  
 statusPending: {  
 color: '#ffc107',  
 },  
 details: {  
 marginBottom: 10,  
 },  
 label: {  
 fontSize: 14,  
 color: '#666',  
 marginBottom: 2,  
 },  
 value: {  
 fontSize: 16,  
 color: '#333',  
 },  
 documents: {  
 marginTop: 10,  
 },  
 documentItem: {  
 backgroundColor: '#f8f9fa',  
 padding: 10,  
 borderRadius: 5,  
 marginTop: 5,  
 },  
 documentName: {  
 fontSize: 14,  
 color: '#007bff',  
 },  
});  
  
export default Application;

# AlumniConnectApp\src\components\CheckStatus.tsx

import { StyleSheet, Text, View, ScrollView, TouchableOpacity, Dimensions, ActivityIndicator } from 'react-native'  
import React, { useState, useEffect } from 'react'  
import { useUser } from '../context/user/UserContext'  
  
// Define types for status history  
interface StatusHistoryItem {  
 status: string;  
 date: string;  
 isApproved?: boolean;  
}  
  
interface FormData {  
 \_id: string;  
 docType: string;  
 status: string;  
 createdAt: string;  
 updatedAt: string;  
 submitted: boolean;  
 submittedRemarks: string;  
 submittedDate: string;  
 admin: boolean;  
 adminRemarks: string;  
 adminDate: string;  
 adminFeesPaid: boolean;  
 documentVerified: boolean;  
 examCell: boolean;  
 examCellRemarks: string;  
 examCellDate: string;  
 examCellFeesPaid: boolean;  
 library: boolean;  
 libraryRemarks: string;  
 libraryDate: string;  
 libraryFeesPaid: boolean;  
 atbranch: boolean;  
 branchRemarks: string;  
 uploadDocument: boolean;  
 collected: boolean;  
 collectionRemarks: string;  
}  
  
interface StatusTimelineProps {  
 statusHistory: StatusHistoryItem[];  
 currentStatus: string;  
}  
  
const getStatusHistory = (form: FormData): StatusHistoryItem[] => {  
 const history: StatusHistoryItem[] = [];  
  
 if (form.submitted) {  
 history.push({  
 status: 'submitted',  
 date: form.submittedDate  
 });  
 }  
  
 if (form.admin) {  
 history.push({  
 status: 'admin',  
 date: form.adminDate,  
 isApproved: true  
 });  
 }  
  
 if (form.examCell) {  
 history.push({  
 status: 'examcell',  
 date: form.examCellDate,  
 isApproved: true  
 });  
 }  
  
 if (form.library) {  
 history.push({  
 status: 'library',  
 date: form.libraryDate,  
 isApproved: true  
 });  
 }  
  
 if (form.atbranch) {  
 history.push({  
 status: 'department',  
 date: form.updatedAt,  
 isApproved: true  
 });  
 }  
  
 if (form.collected) {  
 history.push({  
 status: 'collected',  
 date: form.updatedAt,  
 isApproved: true  
 });  
 }  
  
 return history;  
};  
  
const getCurrentStatus = (form: FormData): string => {  
 if (form.collected) return 'collected';  
 if (form.atbranch) return 'department';  
 if (form.library) return 'library';  
 if (form.examCell) return 'examcell';  
 if (form.admin) return 'admin';  
 if (form.submitted) return 'submitted';  
 return 'submitted';  
};  
  
const StatusTimeline: React.FC<StatusTimelineProps> = ({ statusHistory = [], currentStatus }) => {  
 const statusOrder = [  
 'submitted',  
 'admin',  
 'examcell',  
 'library',  
 'department',  
 'ready\_to\_receive',  
 'collected'  
 ]  
  
 const getStatusState = (status: string, currentStatus: string, index: number) => {  
 const currentIndex = statusOrder.indexOf(currentStatus)  
 const statusItem = statusHistory?.find((item: StatusHistoryItem) => item.status === status)  
   
 if (statusItem?.isApproved) return 'completed'  
 if (index < currentIndex) return 'completed'  
 if (index === currentIndex) return 'current'  
 return 'pending'  
 }  
  
 return (  
 <View style={styles.timelineContainer}>  
 {statusOrder.map((status, index) => {  
 const statusState = getStatusState(status, currentStatus, index)  
 const showLine = index < statusOrder.length - 1  
 const statusItem = statusHistory?.find((item: StatusHistoryItem) => item.status === status)  
  
 return (  
 <View key={status} style={styles.timelineItem}>  
 <View style={styles.timelineContent}>  
 <View style={[  
 styles.statusCircle,  
 statusState === 'completed' && styles.statusCircleCompleted,  
 statusState === 'current' && styles.statusCircleCurrent,  
 statusState === 'pending' && styles.statusCirclePending  
 ]}>  
 {statusState === 'completed' && (  
 <Text style={styles.statusIcon}>✓</Text>  
 )}  
 {statusState === 'current' && (  
 <Text style={styles.statusIcon}>○</Text>  
 )}  
 {statusState === 'pending' && (  
 <Text style={styles.statusIcon}>○</Text>  
 )}  
 </View>  
 {showLine && (  
 <View style={[  
 styles.timelineLine,  
 statusState === 'completed' && styles.timelineLineCompleted,  
 statusState === 'current' && styles.timelineLineCurrent,  
 statusState === 'pending' && styles.timelineLinePending  
 ]} />  
 )}  
 </View>  
 <View style={styles.statusInfo}>  
 <Text style={[  
 styles.statusLabel,  
 statusState === 'completed' && styles.statusLabelCompleted,  
 statusState === 'current' && styles.statusLabelCurrent,  
 statusState === 'pending' && styles.statusLabelPending  
 ]}>  
 {status.replace(/\_/g, ' ')}  
 </Text>  
 {statusItem && (  
 <Text style={styles.statusDate}>  
 {new Date(statusItem.date).toLocaleDateString('en-GB', {  
 day: '2-digit',  
 month: '2-digit',  
 year: 'numeric'  
 })}  
 </Text>  
 )}  
 </View>  
 </View>  
 )  
 })}  
 </View>  
 )  
}  
  
const CheckStatus = () => {  
 const [expandedForms, setExpandedForms] = useState<Set<string>>(new Set())  
 const [forms, setForms] = useState<FormData[]>([])  
 const [loading, setLoading] = useState(true)  
 const [error, setError] = useState<string | null>(null)  
 const { host, user, token } = useUser()  
 const screenWidth = Dimensions.get('window').width  
  
 const fetchForms = async () => {  
 if (!token) {  
 setError('Authentication token is missing')  
 setLoading(false)  
 return  
 }  
  
 try {  
 setLoading(true)  
 setError(null)  
 const response = await fetch(`${host}forms/${user}/getforms`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 }  
 })  
  
 if (!response.ok) {  
 throw new Error('Failed to fetch forms')  
 }  
  
 const data = await response.json()  
 if (data.success) {  
 setForms(data.forms)  
 } else {  
 throw new Error(data.error || 'Failed to fetch forms')  
 }  
 } catch (err) {  
 setError(err instanceof Error ? err.message : 'An error occurred')  
 console.error('Error fetching forms:', err)  
 } finally {  
 setLoading(false)  
 }  
 }  
  
 useEffect(() => {  
 fetchForms()  
 }, [user, token])  
  
 const toggleForm = (formId: string) => {  
 setExpandedForms(prev => {  
 const newSet = new Set(prev)  
 if (newSet.has(formId)) {  
 newSet.delete(formId)  
 } else {  
 newSet.add(formId)  
 }  
 return newSet  
 })  
 }  
  
 if (loading) {  
 return (  
 <View style={styles.loadingContainer}>  
 <ActivityIndicator size="large" color="#f7dfff" />  
 <Text style={styles.loadingText}>Loading your forms...</Text>  
 </View>  
 )  
 }  
  
 if (error) {  
 return (  
 <View style={styles.errorContainer}>  
 <Text style={styles.errorText}>{error}</Text>  
 <TouchableOpacity   
 style={styles.retryButton}  
 onPress={fetchForms}  
 >  
 <Text style={styles.retryButtonText}>Retry</Text>  
 </TouchableOpacity>  
 </View>  
 )  
 }  
  
 return (  
 <View style={styles.container}>  
 <View style={styles.header}>  
 <Text style={styles.headerText}>Check Status</Text>  
 </View>  
  
 <ScrollView   
 style={styles.scrollView}  
 showsVerticalScrollIndicator={false}  
 contentContainerStyle={styles.scrollContent}  
 >  
 {forms.length === 0 ? (  
 <View style={styles.emptyContainer}>  
 <Text style={styles.emptyText}>No forms found</Text>  
 </View>  
 ) : (  
 <View style={styles.formList}>  
 {forms.map((form) => {  
 const formattedDate = new Date(form.createdAt).toLocaleDateString('en-GB', {  
 day: '2-digit',  
 month: '2-digit',  
 year: 'numeric'  
 });  
 const isExpanded = expandedForms.has(form.\_id)  
 const statusHistory = getStatusHistory(form)  
 const currentStatus = getCurrentStatus(form)  
  
 return (  
 <View key={form.\_id} style={styles.formItem}>  
 <TouchableOpacity   
 style={styles.formHeader}  
 onPress={() => toggleForm(form.\_id)}  
 activeOpacity={0.7}  
 >  
 <View style={styles.formHeaderContent}>  
 <Text style={styles.formName} numberOfLines={1}>  
 {form.docType}  
 </Text>  
 <View style={styles.statusContainer}>  
 <Text style={[  
 styles.status,  
 form.status === 'approved' && styles.statusApproved,  
 form.status === 'rejected' && styles.statusRejected,  
 form.status === 'pending' && styles.statusPending  
 ]}>  
 {form.status}  
 </Text>  
 <Text style={styles.dateText}>{formattedDate}</Text>  
 </View>  
 </View>  
 </TouchableOpacity>  
  
 {isExpanded && (  
 <View style={styles.expandedContent}>  
 <StatusTimeline   
 statusHistory={statusHistory}  
 currentStatus={currentStatus}  
 />  
   
 <View style={styles.details}>  
 <Text style={styles.label}>Remarks:</Text>  
 {form.submitted && (  
 <Text style={styles.value}>Submitted: {form.submittedRemarks}</Text>  
 )}  
 {form.admin && (  
 <Text style={styles.value}>Admin: {form.adminRemarks}</Text>  
 )}  
 {form.examCell && (  
 <Text style={styles.value}>Exam Cell: {form.examCellRemarks}</Text>  
 )}  
 {form.library && (  
 <Text style={styles.value}>Library: {form.libraryRemarks}</Text>  
 )}  
 {form.atbranch && (  
 <Text style={styles.value}>Branch: {form.branchRemarks}</Text>  
 )}  
 {form.collected && (  
 <Text style={styles.value}>Collection: {form.collectionRemarks}</Text>  
 )}  
 </View>  
 </View>  
 )}  
 </View>  
 );  
 })}  
 </View>  
 )}  
 </ScrollView>  
 </View>  
 )  
}  
  
export default CheckStatus  
  
const styles = StyleSheet.create({  
 container: {  
 flex: 1,  
 backgroundColor: '#fff',  
 margin: 20,  
 borderColor: '#f7dfff',  
 borderWidth: 1,  
 borderRadius: 20,  
 shadowColor: '#000',  
 shadowOffset: {  
 width: 0,  
 height: 2,  
 },  
 shadowOpacity: 0.1,  
 shadowRadius: 3,  
 },  
 header: {  
 padding: 20,  
 backgroundColor: '#f7dfff',  
 borderTopLeftRadius: 20,  
 borderTopRightRadius: 20,  
 borderBottomWidth: 1,  
 borderBottomColor: '#ddd',  
 },  
 headerText: {  
 fontSize: 24,  
 fontWeight: 'bold',  
 color: '#333',  
 },  
 scrollView: {  
 flex: 1,  
 },  
 scrollContent: {  
 paddingBottom: 20,  
 },  
 formList: {  
 padding: 15,  
 },  
 formItem: {  
 backgroundColor: '#f9f9f9',  
 borderRadius: 10,  
 padding: 15,  
 marginBottom: 10,  
 borderWidth: 1,  
 borderColor: '#eee',  
 shadowColor: '#000',  
 shadowOffset: {  
 width: 0,  
 height: 2,  
 },  
 shadowOpacity: 0.1,  
 shadowRadius: 3,  
 elevation: 2,  
 },  
 formHeader: {  
 width: '100%',  
 },  
 formHeaderContent: {  
 flexDirection: 'row',  
 justifyContent: 'space-between',  
 alignItems: 'center',  
 },  
 formName: {  
 fontSize: 16,  
 fontWeight: 'bold',  
 color: '#333',  
 flex: 1,  
 marginRight: 10,  
 },  
 statusContainer: {  
 alignItems: 'flex-end',  
 minWidth: 100,  
 },  
 status: {  
 fontSize: 14,  
 fontWeight: '600',  
 textTransform: 'capitalize',  
 marginBottom: 4,  
 },  
 statusApproved: {  
 color: '#28a745',  
 },  
 statusRejected: {  
 color: '#dc3545',  
 },  
 statusPending: {  
 color: '#ffc107',  
 },  
 dateText: {  
 fontSize: 12,  
 color: '#666',  
 },  
 expandedContent: {  
 marginTop: 15,  
 paddingTop: 15,  
 borderTopWidth: 1,  
 borderTopColor: '#eee',  
 },  
 details: {  
 marginBottom: 10,  
 },  
 label: {  
 fontSize: 14,  
 color: '#666',  
 marginBottom: 2,  
 },  
 value: {  
 fontSize: 16,  
 color: '#333',  
 },  
 documents: {  
 marginTop: 10,  
 },  
 documentItem: {  
 backgroundColor: '#f8f9fa',  
 padding: 10,  
 borderRadius: 5,  
 marginTop: 5,  
 },  
 documentName: {  
 fontSize: 14,  
 color: '#007bff',  
 },  
 // Timeline styles  
 timelineContainer: {  
 marginBottom: 20,  
 },  
 timelineItem: {  
 flexDirection: 'row',  
 marginBottom: 15,  
 },  
 timelineContent: {  
 flexDirection: 'row',  
 alignItems: 'center',  
 marginRight: 10,  
 },  
 statusCircle: {  
 width: 24,  
 height: 24,  
 borderRadius: 12,  
 borderWidth: 2,  
 alignItems: 'center',  
 justifyContent: 'center',  
 backgroundColor: '#fff',  
 },  
 statusCircleCompleted: {  
 borderColor: '#28a745',  
 backgroundColor: '#28a745',  
 },  
 statusCircleCurrent: {  
 borderColor: '#007bff',  
 backgroundColor: '#fff',  
 },  
 statusCirclePending: {  
 borderColor: '#ddd',  
 backgroundColor: '#fff',  
 },  
 statusIcon: {  
 color: '#fff',  
 fontSize: 14,  
 fontWeight: 'bold',  
 },  
 timelineLine: {  
 width: 2,  
 height: 40,  
 backgroundColor: '#ddd',  
 marginLeft: 11,  
 },  
 timelineLineCompleted: {  
 backgroundColor: '#28a745',  
 },  
 timelineLineCurrent: {  
 backgroundColor: '#007bff',  
 },  
 timelineLinePending: {  
 backgroundColor: '#ddd',  
 },  
 statusInfo: {  
 flex: 1,  
 },  
 statusLabel: {  
 fontSize: 14,  
 color: '#666',  
 textTransform: 'capitalize',  
 },  
 statusLabelCompleted: {  
 color: '#28a745',  
 fontWeight: '600',  
 },  
 statusLabelCurrent: {  
 color: '#007bff',  
 fontWeight: '600',  
 },  
 statusLabelPending: {  
 color: '#666',  
 },  
 statusDate: {  
 fontSize: 12,  
 color: '#666',  
 marginTop: 2,  
 },  
 loadingContainer: {  
 flex: 1,  
 justifyContent: 'center',  
 alignItems: 'center',  
 backgroundColor: '#fff',  
 },  
 loadingText: {  
 marginTop: 10,  
 fontSize: 16,  
 color: '#666',  
 },  
 errorContainer: {  
 flex: 1,  
 justifyContent: 'center',  
 alignItems: 'center',  
 padding: 20,  
 backgroundColor: '#fff',  
 },  
 errorText: {  
 fontSize: 16,  
 color: '#dc3545',  
 textAlign: 'center',  
 marginBottom: 20,  
 },  
 retryButton: {  
 backgroundColor: '#f7dfff',  
 paddingHorizontal: 20,  
 paddingVertical: 10,  
 borderRadius: 5,  
 },  
 retryButtonText: {  
 color: '#333',  
 fontSize: 16,  
 fontWeight: '600',  
 },  
 emptyContainer: {  
 flex: 1,  
 justifyContent: 'center',  
 alignItems: 'center',  
 padding: 20,  
 },  
 emptyText: {  
 fontSize: 16,  
 color: '#666',  
 textAlign: 'center',  
 },  
})

# AlumniConnectApp\src\components\Header.tsx

import { StyleSheet, Text, View, TouchableOpacity, Image } from 'react-native'  
import React, { useState } from 'react'  
import { useUser } from '../context/user/UserContext'  
import Navbar from './Navbar'  
  
interface HeaderProps {  
 showNavbar?: boolean;  
}  
  
const Header = ({ showNavbar = true }: HeaderProps) => {  
 const { userProfile } = useUser()  
 const [isNavbarVisible, setIsNavbarVisible] = useState(false)  
  
 const toggleNavbar = () => {  
 setIsNavbarVisible(!isNavbarVisible)  
 }  
  
 return (  
 <View style={styles.container}>  
 <Text style={styles.title}>Alumni Connect</Text>  
 {showNavbar && (  
 <TouchableOpacity onPress={toggleNavbar}>  
 <View style={styles.profileImageContainer}>  
 {userProfile ? (  
 <Image  
 source={{ uri: userProfile }}  
 style={styles.profileImage}  
 />  
 ) : (  
 <View style={styles.defaultProfile}>  
 <Text style={styles.defaultProfileText}>U</Text>  
 </View>  
 )}  
 </View>  
 </TouchableOpacity>  
 )}  
 {isNavbarVisible && <Navbar onClose={toggleNavbar} />}  
 </View>  
 )  
}  
  
export default Header  
  
const styles = StyleSheet.create({  
 container: {  
 flexDirection: 'row',  
 justifyContent: 'space-between',  
 alignItems: 'center',  
 paddingHorizontal: 20,  
 paddingVertical: 10,  
 backgroundColor: '#EAB8E4',  
 height: 60,  
 },  
 title: {  
 fontSize: 24,  
 fontWeight: 'bold',  
 color: '#5B2E4D',  
 },  
 profileImageContainer: {  
 width: 40,  
 height: 40,  
 borderRadius: 20,  
 borderWidth: 2,  
 borderColor: '#5B2E4D',  
 overflow: 'hidden',  
 },  
 profileImage: {  
 width: '100%',  
 height: '100%',  
 },  
 defaultProfile: {  
 width: '100%',  
 height: '100%',  
 backgroundColor: '#5B2E4D',  
 justifyContent: 'center',  
 alignItems: 'center',  
 },  
 defaultProfileText: {  
 color: '#FFFFFF',  
 fontSize: 20,  
 fontWeight: 'bold',  
 },  
})

# AlumniConnectApp\src\components\Login.tsx

import React, { useState } from 'react';  
import { View, Text, TextInput, TouchableOpacity, StyleSheet, Alert } from 'react-native';  
import { useUser } from '../context/user/UserContext';  
import { useNavigation } from '@react-navigation/native';  
import { NativeStackNavigationProp } from '@react-navigation/native-stack';  
import type { RootStackParamList } from '../App';  
import AsyncStorage from '@react-native-async-storage/async-storage';  
  
const Login = () => {  
 const { host, setIsLogin, setUser, setToken, fetchUserData, fetchFirstLogin } = useUser();  
 const [moodleid, setMoodleid] = useState('');  
 const [password, setPassword] = useState('');  
 const [isLoading, setIsLoading] = useState(false);  
 const navigation = useNavigation<NativeStackNavigationProp<RootStackParamList>>();  
  
 const handleLogin = async () => {  
 if (!moodleid || !password) {  
 Alert.alert('Error', 'Please enter both Moodle ID and password');  
 return;  
 }  
  
 setIsLoading(true);  
 try {  
 const response = await fetch(`${host}auth/student/login`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 'Accept': 'application/json',  
 },  
 body: JSON.stringify({  
 moodleid: moodleid.trim(),  
 password: password.trim()  
 })  
 });  
  
 const responseText = await response.text();  
  
 let data;  
 try {  
 data = JSON.parse(responseText);  
 } catch (e) {  
 throw new Error('Invalid server response');  
 }  
  
 if (!response.ok) {  
 throw new Error(data.message || `HTTP error! status: ${response.status}`);  
 }  
   
 if (data.success) {  
 await AsyncStorage.setItem('auth-token', data.token);  
 await AsyncStorage.setItem('user', 'student');  
 await AsyncStorage.setItem('is-login', 'true');  
  
 setUser('student');  
 setToken(data.token);  
 await fetchUserData('student', data.token);  
 const isFirstLogin = await fetchFirstLogin('student', data.token);  
   
 if (typeof isFirstLogin !== 'boolean') {  
 throw new Error('Invalid first login status received');  
 }  
  
 await AsyncStorage.setItem('first-login', isFirstLogin.toString());  
 setIsLogin(true);  
  
 if (isFirstLogin === true) {  
 navigation.replace('VerifyContact');  
 } else if (isFirstLogin === false) {  
 navigation.replace('CheckStatus');  
 } else {  
 Alert.alert('Error', 'Invalid login status');  
 }  
 } else {  
 Alert.alert('Error', data.message || 'Invalid credentials');  
 }  
 } catch (error) {  
 Alert.alert(  
 'Login Error',  
 error instanceof Error ? error.message : 'Failed to connect to server. Please try again.'  
 );  
 } finally {  
 setIsLoading(false);  
 }  
 };  
  
 return (  
 <View style={styles.container}>  
 {/\* <View style={styles.iconContainer}>  
 <Image  
 source={require('../assets/apsit.png')} // Replace with your icon path  
 style={styles.icon}  
 />  
 </View> \*/}  
 <Text style={styles.header}>Login</Text>  
 <TextInput  
 style={[styles.input, { textAlign: 'center' }]}  
 placeholder="MOODLE ID"  
 placeholderTextColor="#5B2E4D"  
 value={moodleid}  
 onChangeText={setMoodleid}  
 autoCapitalize="none"  
 autoCorrect={false}  
 keyboardType="numeric"  
 />  
 <TextInput  
 style={[styles.input, { textAlign: 'center' }]}  
 placeholder="PASSWORD"  
 placeholderTextColor="#5B2E4D"  
 value={password}  
 onChangeText={setPassword}  
 secureTextEntry  
 autoCapitalize="none"  
 autoCorrect={false}  
 />  
 <TouchableOpacity   
 style={[styles.button, isLoading && styles.buttonDisabled]}   
 onPress={handleLogin}  
 disabled={isLoading}  
 >  
 <Text style={styles.buttonText}>  
 {isLoading ? 'LOGGING IN...' : 'LOGIN'}  
 </Text>  
 </TouchableOpacity>  
 </View>  
 );  
};  
  
const styles = StyleSheet.create({  
 container: {  
 marginHorizontal:20,  
 borderRadius:10,  
 marginTop:125,  
 marginBottom:125,  
 flex: 1,  
 justifyContent: 'center',  
 alignItems: 'center',  
 backgroundColor: '#EAB8E4',  
 padding: 10,  
 },  
 header: {  
 fontSize: 24,  
 fontWeight: 'bold',  
 color: '#5B2E4D',  
 marginBottom: 20,  
 },  
 input: {  
 width: '80%',  
 padding: 15,  
 borderRadius: 5,  
 backgroundColor: '#FFFFFF',  
 marginBottom: 15,  
 borderColor: '#D1C4E9',  
 borderWidth: 1,  
 color: 'black',  
 },  
 button: {  
 width: '40%',  
 padding: 15,  
 backgroundColor: '#F7DFFF',  
 borderRadius: 5,  
 alignItems: 'center',  
 marginTop: 30,  
 },  
 buttonDisabled: {  
 backgroundColor: '#D1C4E9',  
 },  
 buttonText: {  
 fontSize: 18,  
 fontWeight: 'bold',  
 color: '#5B2E4D',  
 },  
});  
  
export default Login;

# AlumniConnectApp\src\components\MainBody.tsx

import { StyleSheet, View, ScrollView, SafeAreaView } from 'react-native';  
import React, { useEffect } from 'react';  
import { Dimensions } from 'react-native';  
import Login from '../components/Login';   
import VerifyContact from '../components/VerifyContact';  
import CheckStatus from '../components/CheckStatus';  
import { useUser } from '../context/user/UserContext';  
import { useNavigation, useRoute } from '@react-navigation/native';  
import type { NativeStackNavigationProp } from '@react-navigation/native-stack';  
import type { RootStackParamList } from '../App';  
  
const { width, height } = Dimensions.get('window');  
  
type NavigationProp = NativeStackNavigationProp<RootStackParamList>;  
  
const MainBody = () => {  
 const { isLogin, isFirstLogin } = useUser();  
 const navigation = useNavigation<NavigationProp>();  
 const route = useRoute();  
  
 useEffect(() => {  
 if (!isLogin && route.name !== 'Login') {  
 navigation.replace('Login');  
 return;  
 }  
  
 if (isLogin) {  
 if (isFirstLogin === true && route.name !== 'VerifyContact') {  
 navigation.replace('VerifyContact');  
 } else if (isFirstLogin === false && route.name !== 'CheckStatus') {  
 navigation.replace('CheckStatus');  
 }  
 }  
 }, [isLogin, isFirstLogin, navigation, route.name]);  
   
 return null;  
};  
  
export default MainBody;  
  
const styles = StyleSheet.create({  
 safeArea: {  
 flex: 1,  
 backgroundColor: '#ffffff',  
 },  
 scrollContent: {  
 flexGrow: 1,  
 },  
 mainBody: {  
 backgroundColor: '#ffffff',  
 minHeight: height - 70,  
 width: width,  
 flex: 1,  
 padding: 15,  
 flexDirection: 'row',  
 alignItems: 'center',  
 justifyContent: 'center',  
 },  
});

# AlumniConnectApp\src\components\Navbar.tsx

import React, { useEffect, useRef } from 'react'  
import { Text, View, TouchableOpacity, Image, Animated, Dimensions, StyleSheet } from 'react-native'  
import { useNavigation } from '@react-navigation/native'  
import { NativeStackNavigationProp } from '@react-navigation/native-stack'  
import type { RootStackParamList } from '../App'  
import { useUser } from '../context/user/UserContext'  
  
const { width, height } = Dimensions.get('window');a  
  
type NavigationProp = NativeStackNavigationProp<RootStackParamList>  
  
const Navbar = ({ onClose }: { onClose: () => void }) => {  
 const navigation = useNavigation<NavigationProp>()  
 const { logout } = useUser()  
 const slideAnim = useRef(new Animated.Value(width \* 0.8)).current  
  
 useEffect(() => {  
 Animated.timing(slideAnim, {  
 toValue: 0,  
 duration: 300,  
 useNativeDriver: true,  
 }).start()  
 }, [])  
  
 const closeMenu = () => {  
 Animated.timing(slideAnim, {  
 toValue: width \* 0.8,  
 duration: 200,  
 useNativeDriver: true,  
 }).start(() => {  
 onClose()  
 })  
 }  
  
 const handleNavigation = (screen: keyof RootStackParamList) => {  
 Animated.timing(slideAnim, {  
 toValue: width \* 0.8,  
 duration: 200,  
 useNativeDriver: true,  
 }).start(() => {  
 onClose()  
 navigation.navigate(screen)  
 })  
 }  
  
 const handleLogout = async () => {  
 Animated.timing(slideAnim, {  
 toValue: width \* 0.8,  
 duration: 200,  
 useNativeDriver: true,  
 }).start(async () => {  
 await logout()  
 onClose()  
 navigation.replace('Login')  
 })  
 }  
  
 return (  
 <Animated.View  
 style={[  
 styles.container,  
 { transform: [{ translateX: slideAnim }] }  
 ]}  
 >  
 <TouchableOpacity style={styles.closeButton} onPress={closeMenu}>  
 <Text style={styles.closeButtonText}>×</Text>  
 </TouchableOpacity>  
 <View>  
 <TouchableOpacity onPress={() => handleNavigation('Profile')} style={[styles.menuItem, styles.firstMenuItem]}>  
 <Image source={require('../assets/profile.png')} style={styles.icon} />  
 <Text style={styles.menuText}>Profile</Text>  
 </TouchableOpacity>  
 <TouchableOpacity onPress={() => handleNavigation('CheckStatus')} style={styles.menuItem}>  
 <Image source={require('../assets/status.png')} style={styles.icon} />  
 <Text style={styles.menuText}>Check Status</Text>  
 </TouchableOpacity>  
 <TouchableOpacity onPress={() => handleNavigation('Notices')} style={styles.menuItem}>  
 <Image source={require('../assets/notice.png')} style={styles.icon} />  
 <Text style={styles.menuText}>Notices</Text>  
 </TouchableOpacity>  
 <TouchableOpacity onPress={handleLogout} style={styles.menuItem}>  
 <Image source={require('../assets/logout.png')} style={styles.icon} />  
 <Text style={styles.menuText}>Logout</Text>  
 </TouchableOpacity>  
 </View>  
 </Animated.View>  
 )  
}  
  
const styles = StyleSheet.create({  
 container: {  
 position: 'absolute',  
 top: 0,  
 right: 0,  
 width: width \* 0.8,  
 height: height,  
 backgroundColor: '#EAB8E4',  
 padding: 25,  
 shadowColor: '#000',  
 shadowOffset: { width: -2, height: 0 },  
 shadowOpacity: 0.25,  
 shadowRadius: 3.84,  
 elevation: 5,  
 zIndex: 10000,  
 },  
 closeButton: {  
 position: 'absolute',  
 top: 15,  
 right: 15,  
 width: 40,  
 height: 40,  
 borderRadius: 20,  
 backgroundColor: '#5B2E4D',  
 justifyContent: 'center',  
 alignItems: 'center',  
 zIndex: 10001,  
 },  
 closeButtonText: {  
 color: '#fff',  
 fontSize: 28,  
 fontWeight: 'bold',  
 lineHeight: 28,  
 },  
 menuItem: {  
 flexDirection: 'row',  
 alignItems: 'center',  
 paddingVertical: 20,  
 borderBottomWidth: 1,  
 borderBottomColor: '#D1A6D1',  
 minHeight: 50,  
 },  
 firstMenuItem: {  
 marginTop: 60,  
 },  
 icon: {  
 width: 30,  
 height: 30,  
 marginRight: 20,  
 tintColor: '#5B2E4D',  
 },  
 menuText: {  
 fontSize: 20,  
 color: '#5B2E4D',  
 fontWeight: '600',  
 },  
})  
  
export default Navbar

# AlumniConnectApp\src\components\Notices.tsx

import { StyleSheet, Text, View, ScrollView, ActivityIndicator, TouchableOpacity, Alert } from 'react-native'  
import React, { useEffect, useState } from 'react'  
import { useUser } from '../context/user/UserContext'  
  
const Notices = () => {  
 const { notifications, fetchNotifications, token, markNotificationAsRead } = useUser()  
 const [isLoading, setIsLoading] = useState(true)  
 const [error, setError] = useState<string | null>(null)  
 const [markingId, setMarkingId] = useState<string | null>(null)  
  
 useEffect(() => {  
 const loadNotices = async () => {  
 if (!token) {  
 setError('Not logged in');  
 setIsLoading(false);  
 return;  
 }  
  
 try {  
 setIsLoading(true);  
 setError(null);  
 await fetchNotifications(token);  
 } catch (err) {  
 setError('Failed to load notices');  
 console.error('Error loading notices:', err);  
 } finally {  
 setIsLoading(false);  
 }  
 };  
  
 loadNotices();  
 }, []);  
  
 const formatDate = (dateString: string) => {  
 try {  
 const date = new Date(dateString);  
 if (isNaN(date.getTime())) {  
 return 'Invalid Date';  
 }  
 return date.toLocaleDateString('en-US', {  
 year: 'numeric',  
 month: 'short',  
 day: 'numeric',  
 hour: '2-digit',  
 minute: '2-digit',  
 hour12: true  
 });  
 } catch (error) {  
 console.error('Error formatting date:', error);  
 return 'Invalid Date';  
 }  
 };  
  
 const handleNotificationPress = async (notificationId: string, isRead: boolean) => {  
 if (isRead || markingId === notificationId) return;  
  
 try {  
 setMarkingId(notificationId);  
 await markNotificationAsRead(notificationId);  
 } catch (error) {  
 Alert.alert(  
 'Error',  
 'Failed to mark notification as read. Please try again.'  
 );  
 console.error('Error marking notification as read:', error);  
 } finally {  
 setMarkingId(null);  
 }  
 };  
  
 if (isLoading) {  
 return (  
 <View style={styles.container}>  
 <Text style={styles.title}>Notices</Text>  
 <View style={styles.loadingContainer}>  
 <ActivityIndicator size="large" color="#5B2E4D" />  
 </View>  
 </View>  
 );  
 }  
  
 if (error) {  
 return (  
 <View style={styles.container}>  
 <Text style={styles.title}>Notices</Text>  
 <View style={styles.errorContainer}>  
 <Text style={styles.errorText}>{error}</Text>  
 </View>  
 </View>  
 );  
 }  
  
 return (  
 <View style={styles.container}>  
 <Text style={styles.title}>Notices</Text>  
 <ScrollView style={styles.noticesContainer}>  
 {notifications.length > 0 ? (  
 notifications.map((notice, index) => (  
 <TouchableOpacity  
 key={index}  
 style={[  
 styles.noticeItem,  
 !notice.isRead && styles.unreadNotice,  
 markingId === notice.\_id && styles.markingNotice  
 ]}  
 onPress={() => handleNotificationPress(notice.\_id, notice.isRead)}  
 activeOpacity={0.7}  
 disabled={markingId === notice.\_id}  
 >  
 <Text style={styles.noticeTitle}>{notice.title}</Text>  
 <Text style={styles.noticeMessage}>{notice.message}</Text>  
 <View style={styles.noticeFooter}>  
 <Text style={styles.noticeFrom}>From: {notice.from}</Text>  
 <Text style={styles.noticeDate}>{formatDate(notice.timestamp)}</Text>  
 </View>  
 {markingId === notice.\_id && (  
 <ActivityIndicator   
 size="small"   
 color="#5B2E4D"   
 style={styles.markingIndicator}  
 />  
 )}  
 </TouchableOpacity>  
 ))  
 ) : (  
 <Text style={styles.noNotices}>No notices available</Text>  
 )}  
 </ScrollView>  
 </View>  
 )  
}  
  
export default Notices  
  
const styles = StyleSheet.create({  
 container: {  
 flex: 1,  
 backgroundColor: '#ffffff',  
 padding: 20,  
 },  
 title: {  
 fontSize: 24,  
 fontWeight: 'bold',  
 color: '#5B2E4D',  
 marginBottom: 20,  
 },  
 loadingContainer: {  
 flex: 1,  
 justifyContent: 'center',  
 alignItems: 'center',  
 },  
 errorContainer: {  
 flex: 1,  
 justifyContent: 'center',  
 alignItems: 'center',  
 },  
 errorText: {  
 fontSize: 16,  
 color: 'red',  
 textAlign: 'center',  
 },  
 noticesContainer: {  
 flex: 1,  
 },  
 noticeItem: {  
 backgroundColor: '#F5F5F5',  
 borderRadius: 10,  
 padding: 15,  
 marginBottom: 15,  
 },  
 unreadNotice: {  
 backgroundColor: '#FFF0F5',  
 borderLeftWidth: 4,  
 borderLeftColor: '#5B2E4D',  
 },  
 markingNotice: {  
 opacity: 0.7,  
 },  
 markingIndicator: {  
 position: 'absolute',  
 right: 10,  
 top: 10,  
 },  
 noticeTitle: {  
 fontSize: 18,  
 fontWeight: 'bold',  
 color: '#5B2E4D',  
 marginBottom: 10,  
 },  
 noticeMessage: {  
 fontSize: 16,  
 color: '#666',  
 marginBottom: 10,  
 },  
 noticeFooter: {  
 flexDirection: 'row',  
 justifyContent: 'space-between',  
 alignItems: 'center',  
 },  
 noticeFrom: {  
 fontSize: 14,  
 color: '#666',  
 fontStyle: 'italic',  
 },  
 noticeDate: {  
 fontSize: 14,  
 color: '#999',  
 },  
 noNotices: {  
 fontSize: 16,  
 color: '#666',  
 textAlign: 'center',  
 marginTop: 20,  
 },  
})

# AlumniConnectApp\src\components\Profile.tsx

import { StyleSheet, Text, View, Image } from 'react-native'  
import React from 'react'  
import { useUser } from '../context/user/UserContext'  
  
const Profile = () => {  
 const { userData, userProfile } = useUser()  
  
 return (  
 <View style={styles.container}>  
 <View style={styles.profileHeader}>  
 <Image  
 source={userProfile ? { uri: userProfile } : require('../assets/profile.png')}  
 style={styles.profileImage}  
 />  
 <Text style={styles.name}>{userData?.firstName || 'User'}</Text>  
 <Text style={styles.email}>{userData?.email || 'No email provided'}</Text>  
 </View>  
  
 <View style={styles.infoContainer}>  
 <View style={styles.infoItem}>  
 <Text style={styles.infoLabel}>Moodle ID</Text>  
 <Text style={styles.infoValue}>{userData?.moodleid || 'Not available'}</Text>  
 </View>  
 <View style={styles.infoItem}>  
 <Text style={styles.infoLabel}>Mobile Number</Text>  
 <Text style={styles.infoValue}>{userData?.mobileno || 'Not available'}</Text>  
 </View>  
 </View>  
 </View>  
 )  
}  
  
export default Profile  
  
const styles = StyleSheet.create({  
 container: {  
 flex: 1,  
 backgroundColor: '#ffffff',  
 padding: 20,  
 },  
 profileHeader: {  
 alignItems: 'center',  
 marginBottom: 30,  
 },  
 profileImage: {  
 width: 120,  
 height: 120,  
 borderRadius: 60,  
 marginBottom: 15,  
 borderWidth: 3,  
 borderColor: '#EAB8E4',  
 },  
 name: {  
 fontSize: 24,  
 fontWeight: 'bold',  
 color: '#5B2E4D',  
 marginBottom: 5,  
 },  
 email: {  
 fontSize: 16,  
 color: '#666',  
 },  
 infoContainer: {  
 backgroundColor: '#F5F5F5',  
 borderRadius: 10,  
 padding: 20,  
 },  
 infoItem: {  
 marginBottom: 15,  
 },  
 infoLabel: {  
 fontSize: 14,  
 color: '#666',  
 marginBottom: 5,  
 },  
 infoValue: {  
 fontSize: 16,  
 color: '#5B2E4D',  
 fontWeight: '500',  
 },  
})

# AlumniConnectApp\src\components\VerifyContact.tsx

import React, { useState } from 'react';  
import { View, Text, TextInput, TouchableOpacity, StyleSheet, ScrollView, Alert } from 'react-native';  
import { useNavigation } from '@react-navigation/native';  
import AsyncStorage from '@react-native-async-storage/async-storage';  
import { NavigationProp } from '@react-navigation/native';  
  
interface VerifyContactProps {  
 host: string;  
 token: string;  
 user: string;  
}  
  
type RootStackParamList = {  
 CheckStatus: undefined;  
};  
  
const VerifyContact: React.FC<VerifyContactProps> = ({ host, token, user }) => {  
 const [isVisible, setIsVisible] = useState(false);  
 const [mobileNumber, setMobileNumber] = useState('');  
 const [email, setEmail] = useState('');  
 const [mobileOtp, setMobileOtp] = useState('');  
 const [emailOtp, setEmailOtp] = useState('');  
 const [password, setPassword] = useState('');  
 const [confirmPassword, setConfirmPassword] = useState('');  
 const [otpSent, setOtpSent] = useState({ mobile: false, email: false });  
 const [verified, setVerified] = useState({ mobile: false, email: false });  
   
 const navigation = useNavigation<NavigationProp<RootStackParamList>>();  
  
 const toggleVisibility = () => {  
 setIsVisible(!isVisible);  
 };   
  
 const sendOtp = async (type: 'mobile' | 'email', value: string) => {  
 try {  
 const response = await fetch(  
 `${host}auth/otp/student/send${type}otp`,  
 {  
 method: "POST",  
 headers: { "Content-Type": "application/json", "auth-token": token },  
 body: JSON.stringify({ [type === "mobile" ? "mobileno" : "email"]: value }),  
 }  
 );  
  
 const data = await response.json();  
 if (data.success) {  
 Alert.alert('Success', `${type} OTP sent successfully!`);  
 setOtpSent((prev) => ({ ...prev, [type]: true }));  
 } else {  
 Alert.alert('Error', `Failed to send OTP: ${data.error}`);  
 }  
 } catch (error) {  
 Alert.alert('Error', "Error sending OTP.");  
 }  
 };  
  
 const verifyOtp = async (type: 'mobile' | 'email', otp: string) => {  
 try {  
 if (!otp) {  
 Alert.alert('Error', `Please enter the ${type} OTP`);  
 return;  
 }  
  
 const contact = type === "mobile" ? mobileNumber : email;  
 if (!contact) {  
 Alert.alert('Error', `Please enter your ${type} first`);  
 return;  
 }  
  
 console.log('Verifying OTP with:', {  
 type,  
 contact,  
 otp,  
 url: `${host}auth/otp/verify${type}otp`  
 });  
  
 const response = await fetch(`${host}auth/otp/verify${type}otp`, {  
 method: "POST",  
 headers: {   
 "Content-Type": "application/json",   
 "auth-token": token   
 },  
 body: JSON.stringify({   
 [type === "mobile" ? "mobileno" : "email"]: contact,  
 otp: otp.trim()  
 }),  
 });  
  
 console.log('Response status:', response.status);  
 const responseText = await response.text();  
 console.log('Response text:', responseText);  
  
 let data;  
 try {  
 data = JSON.parse(responseText);  
 } catch (e) {  
 console.error('Failed to parse response:', e);  
 throw new Error('Invalid server response');  
 }  
  
 if (!response.ok) {  
 throw new Error(data.message || `HTTP error! status: ${response.status}`);  
 }  
  
 if (data.success) {  
 Alert.alert('Success', `${type} OTP verified successfully!`);  
 setVerified((prev) => ({ ...prev, [type]: true }));  
 } else {  
 Alert.alert('Error', `Failed to verify OTP: ${data.message || 'Invalid OTP'}`);  
 }  
 } catch (error) {  
 console.error('OTP verification error:', error);  
 Alert.alert(  
 'Error',  
 error instanceof Error ? error.message : 'Failed to verify OTP. Please try again.'  
 );  
 }  
 };  
  
 const handleSubmit = async () => {  
 if (!verified.mobile || !verified.email) {  
 Alert.alert('Error', 'Please verify both mobile number and email first');  
 return;  
 }  
  
 if (!password || !confirmPassword) {  
 Alert.alert('Error', 'Please enter both password fields');  
 return;  
 }  
  
 if (password !== confirmPassword) {  
 Alert.alert('Error', 'Passwords do not match');  
 return;  
 }  
  
 if (password.length < 6) {  
 Alert.alert('Error', 'Password must be at least 6 characters long');  
 return;  
 }  
  
 try {  
 console.log('Resetting password with:', {  
 mobileno: mobileNumber,  
 email,  
 url: `${host}auth/${user}/setpass/`  
 });  
  
 const response = await fetch(`${host}auth/${user}/setpass/`, {  
 method: "PUT",  
 headers: {   
 "Content-Type": "application/json",   
 "auth-token": token   
 },  
 body: JSON.stringify({   
 mobileno: mobileNumber,   
 email,   
 password: password.trim()  
 }),  
 });  
  
 console.log('Response status:', response.status);  
 const responseText = await response.text();  
 console.log('Response text:', responseText);  
  
 let data;  
 try {  
 data = JSON.parse(responseText);  
 } catch (e) {  
 console.error('Failed to parse response:', e);  
 throw new Error('Invalid server response');  
 }  
  
 if (!response.ok) {  
 throw new Error(data.message || `HTTP error! status: ${response.status}`);  
 }  
  
 if (data.success) {  
 Alert.alert('Success', 'Password reset successfully!', [  
 {  
 text: 'OK',  
 onPress: async () => {  
 try {  
 await AsyncStorage.removeItem('first-login');  
 navigation.navigate('CheckStatus');  
 } catch (error) {  
 console.error('Navigation error:', error);  
 Alert.alert('Error', 'Failed to navigate to next screen');  
 }  
 }  
 }  
 ]);  
 } else {  
 Alert.alert('Error', `Failed to reset password: ${data.message || 'Unknown error'}`);  
 }  
 } catch (error) {  
 console.error('Password reset error:', error);  
 Alert.alert(  
 'Error',  
 error instanceof Error ? error.message : 'Failed to reset password. Please try again.'  
 );  
 }  
 };  
  
 return (  
 <View style={styles.container}>  
 <ScrollView showsVerticalScrollIndicator={false} showsHorizontalScrollIndicator={false}>  
 {/\* Mobile Number Section \*/}  
 <View style={styles.fieldContainer}>  
 <Text style={styles.label}>Mobile no.</Text>  
 <View style={styles.inputContainer}>  
 <TextInput   
 style={styles.input}   
 placeholder="Enter Mobile Number"   
 placeholderTextColor="#5B2E4D"  
 value={mobileNumber}  
 onChangeText={setMobileNumber}  
 keyboardType="phone-pad"  
 />  
 <TouchableOpacity   
 style={styles.button}  
 onPress={() => sendOtp('mobile', mobileNumber)}  
 >  
 <Text style={styles.buttonText}>Send</Text>  
 </TouchableOpacity>  
 </View>  
 <View style={styles.inputContainer}>  
 <TextInput   
 style={styles.input}   
 placeholder="Enter Mobile OTP"   
 placeholderTextColor="#5B2E4D"  
 value={mobileOtp}  
 onChangeText={setMobileOtp}  
 keyboardType="number-pad"  
 />  
 <TouchableOpacity   
 style={styles.button}  
 onPress={() => verifyOtp('mobile', mobileOtp)}  
 >  
 <Text style={styles.buttonText}>Verify</Text>  
 </TouchableOpacity>  
 </View>  
 </View>  
  
 {/\* Email ID Section \*/}  
 <View style={styles.fieldContainer}>  
 <Text style={styles.label}>Email id</Text>  
 <View style={styles.inputContainer}>  
 <TextInput   
 style={styles.input}   
 placeholder="Enter Email ID"   
 placeholderTextColor="#5B2E4D"  
 value={email}  
 onChangeText={setEmail}  
 keyboardType="email-address"  
 />  
 <TouchableOpacity   
 style={styles.button}  
 onPress={() => sendOtp('email', email)}  
 >  
 <Text style={styles.buttonText}>Send</Text>  
 </TouchableOpacity>  
 </View>  
 <View style={styles.inputContainer}>  
 <TextInput   
 style={styles.input}   
 placeholder="Enter Email OTP"   
 placeholderTextColor="#5B2E4D"  
 value={emailOtp}  
 onChangeText={setEmailOtp}  
 keyboardType="number-pad"  
 />  
 <TouchableOpacity   
 style={styles.button}  
 onPress={() => verifyOtp('email', emailOtp)}  
 >  
 <Text style={styles.buttonText}>Verify</Text>  
 </TouchableOpacity>  
 </View>  
 </View>  
  
 {/\* Password Section \*/}  
 <View style={styles.fieldContainer}>  
 <Text style={styles.label}>Enter New Password</Text>  
 <View style={styles.inputContainer}>  
 <TextInput   
 style={styles.input}   
 placeholder="Enter New Password"   
 placeholderTextColor="#5B2E4D"   
 secureTextEntry={!isVisible}  
 value={password}  
 onChangeText={setPassword}  
 keyboardType="default"  
 autoCapitalize="none"  
 autoCorrect={false}  
 />  
 <TouchableOpacity style={styles.button} onPress={toggleVisibility}>  
 <Text style={styles.buttonText}>{isVisible ? 'Hide' : 'Show'}</Text>  
 </TouchableOpacity>  
 </View>  
 </View>  
  
 <View style={styles.fieldContainer}>  
 <Text style={styles.label}>Re-enter Password</Text>  
 <View style={styles.inputContainer}>  
 <TextInput   
 style={styles.input}   
 placeholder="Re-enter Password"   
 placeholderTextColor="#5B2E4D"   
 secureTextEntry={!isVisible}  
 value={confirmPassword}  
 onChangeText={setConfirmPassword}  
 keyboardType="default"  
 autoCapitalize="none"  
 autoCorrect={false}  
 />  
 <TouchableOpacity style={styles.button} onPress={toggleVisibility}>  
 <Text style={styles.buttonText}>{isVisible ? 'Hide' : 'Show'}</Text>  
 </TouchableOpacity>  
 </View>  
 </View>  
  
 {/\* Submit Button \*/}  
 <TouchableOpacity style={styles.submitButton} onPress={handleSubmit}>  
 <Text style={styles.submitButtonText}>Submit</Text>  
 </TouchableOpacity>  
 </ScrollView>  
 </View>  
 );  
};  
  
const styles = StyleSheet.create({  
 container: {  
 flex: 1,  
 backgroundColor: '#EAB8E4', // Light purple background  
 paddingHorizontal: 20,  
 justifyContent: 'center',  
 borderRadius: 10,  
 shadowColor: '#000',  
 shadowOffset: { width: 0, height: 2 },  
 shadowOpacity: 0.25,  
 shadowRadius: 3.84,  
 elevation: 10,  
 },  
 fieldContainer: {  
 marginTop: 20,  
 backgroundColor: '#F5F5F5', // Light gray background for input sections  
 borderRadius: 10,  
 padding: 15,  
 },  
 inputContainer: {  
 flexDirection: 'row',  
 alignItems: 'center',  
 justifyContent: 'flex-start',  
 padding: 5,  
 },  
 label: {  
 fontSize: 16,  
 marginBottom: 5,  
 },  
 input: {  
 height: 40,  
 width: '70%',  
 color: '#5B2E4D',  
 borderColor: '#ccc',  
 borderWidth: 1,  
 borderRadius: 5,  
 paddingHorizontal: 10,  
 // marginBottom: 10,  
 marginRight: 10,  
 },  
 button: {  
 height: 40,  
 width: '30%',  
 backgroundColor: '#D1A6D1', // Light purple for Send button  
 borderRadius: 5,  
 padding: 10,  
 alignItems: 'center',  
 },  
 buttonText: {  
 color: '#fff',  
 },  
 submitButton: {  
 backgroundColor: '#D1A6D1', // Light purple for Submit button  
 borderRadius: 5,  
 padding: 15,  
 alignItems: 'center',  
 marginVertical: 20,  
 },  
 submitButtonText: {  
 color: '#000', // Black text for Submit button  
 fontSize: 18,  
 },  
});  
  
export default VerifyContact;

# AlumniConnectApp\src\context\user\UserContext.tsx

import React, { createContext, useContext, useState, ReactNode } from 'react';  
import AsyncStorage from '@react-native-async-storage/async-storage';  
  
interface UserData {  
 id: string;  
 name: string;  
 email: string;  
 mobileno: string;  
 [key: string]: any;  
}  
  
interface Notification {  
 \_id: string;  
 title: string;  
 message: string;  
 isRead: boolean;  
 timestamp: string;  
 from: string;  
 [key: string]: any;  
}  
  
interface UserContextType {  
 host: string;  
 isLogin: boolean;  
 isFirstLogin: boolean;  
 user: string | null;  
 token: string | null;  
 userProfile: string;  
 userData: UserData | null;  
 notifications: Notification[];  
 unreadCount: number;  
 setIsLogin: (isLogin: boolean) => void;  
 setIsFirstLogin: (isFirstLogin: boolean) => void;  
 setUser: (user: string | null) => void;  
 setToken: (token: string | null) => void;  
 setUserProfile: (profile: string) => void;  
 setUserData: (data: UserData | null) => void;  
 setNotifications: (notifications: Notification[]) => void;  
 setUnreadCount: (unreadCount: number) => void;  
 loadProfilePicture: () => Promise<void>;  
 fetchUserData: (user: string, token: string) => Promise<void>;  
 fetchFirstLogin: (user: string, token: string) => Promise<boolean | undefined>;  
 fetchNotifications: (token: string) => Promise<Notification[]>;  
 markNotificationAsRead: (notificationId: string) => Promise<void>;  
 logout: () => Promise<void>;  
}  
  
const UserContext = createContext<UserContextType | undefined>(undefined);  
  
export const UserProvider: React.FC<{ children: ReactNode }> = ({ children }) => {  
 const [host] = useState<string>(() => {  
 return 'http://192.168.57.19:5000/api/';  
 });  
 const [isLogin, setIsLogin] = useState<boolean>(false);  
 const [isFirstLogin, setIsFirstLogin] = useState<boolean>(false);  
 const [user, setUser] = useState<string | null>(null);  
 const [token, setToken] = useState<string | null>(null);  
 const [userProfile, setUserProfile] = useState<string>('');  
 const [userData, setUserData] = useState<UserData | null>(null);  
 const [notifications, setNotifications] = useState<Notification[]>([]);  
 const [unreadCount, setUnreadCount] = useState<number>(0);  
  
 const loadProfilePicture = async () => {  
 if (!user || !token) return;  
   
 try {  
 const response = await fetch(`${host}files/profile/${user}`, {  
 headers: {  
 'auth-token': token  
 }  
 });  
 if (response.ok) {  
 const blob = await response.blob();  
 const imageUrl = URL.createObjectURL(blob);  
 setUserProfile(imageUrl);  
 }  
 } catch (error) {  
 console.error('Error loading profile picture:', error);  
 }  
 };  
  
 const fetchUserData = async (user: string, token: string) => {  
 try {  
 const response = await fetch(`${host}auth/student/getuser`, {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 }  
 });  
 if (!response.ok) {  
 throw new Error('Failed to fetch user data');  
 }  
 const data = await response.json();  
 if (data.success) {  
 setUserData(data.userData);  
 }  
 } catch (error) {  
 console.error("Error fetching user data:", error);  
 throw error;  
 }  
 };  
  
 const fetchFirstLogin = async (user: string, token: string) => {  
 try {  
 const response = await fetch(`${host}auth/${user}/getfirstlogin`, {  
 method: "POST",  
 headers: {  
 "Content-Type": "application/json",  
 "auth-token": token  
 }  
 });  
  
 const responseText = await response.text();  
  
 if (!response.ok) {  
 throw new Error(`Failed to fetch first login status: ${response.status}`);  
 }  
  
 let data;  
 try {  
 data = JSON.parse(responseText);  
 } catch (e) {  
 throw new Error('Invalid server response for first login');  
 }  
  
 if (typeof data.firstLogin === 'boolean') {  
 setIsFirstLogin(data.firstLogin);  
 await AsyncStorage.setItem('first-login', data.firstLogin.toString());  
 return data.firstLogin;  
 } else {  
 throw new Error('Invalid first login response format');  
 }  
 } catch (error) {  
 throw error;  
 }  
 };  
  
 const fetchNotifications = async (token: string) => {  
 try {  
 const response = await fetch(`${host}notification/getnotifications`, {  
 method: 'GET',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 }  
 });  
 const data = await response.json();  
 if (data.success) {  
 setNotifications(data.notifications || []);  
 setUnreadCount(data.unreadCount || 0);  
 }  
 return data.notifications || [];  
 } catch (error) {  
 console.error('Error fetching notifications:', error);  
 return [];  
 }  
 };  
  
 const markNotificationAsRead = async (notificationId: string) => {  
 if (!token) {  
 console.error('No token available for marking notification as read');  
 return;  
 }  
   
 try {  
 console.log('Marking notification as read:', notificationId);  
 const response = await fetch(`${host}notification/markread/${notificationId}`, {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 'auth-token': token  
 }  
 });  
  
 console.log('Response status:', response.status);  
 const responseText = await response.text();  
 console.log('Response text:', responseText);  
  
 let data;  
 try {  
 data = JSON.parse(responseText);  
 } catch (e) {  
 console.error('Failed to parse response:', e);  
 throw new Error('Invalid server response');  
 }  
  
 if (!response.ok) {  
 throw new Error(data.message || `HTTP error! status: ${response.status}`);  
 }  
  
 if (data.success) {  
 // Update local state  
 setNotifications(prevNotifications =>   
 prevNotifications.map(notification =>   
 notification.\_id === notificationId   
 ? { ...notification, isRead: true }  
 : notification  
 )  
 );  
 setUnreadCount(prevCount => Math.max(0, prevCount - 1));  
 } else {  
 throw new Error(data.message || 'Failed to mark notification as read');  
 }  
 } catch (error) {  
 console.error('Error marking notification as read:', error);  
 throw error;  
 }  
 };  
  
 const logout = async () => {  
 try {  
 await AsyncStorage.removeItem('auth-token');  
 await AsyncStorage.removeItem('user');  
 setIsLogin(false);  
 setUser(null);  
 setToken(null);  
 setUserData(null);  
 setUserProfile('');  
 setNotifications([]);  
 setUnreadCount(0);  
 } catch (error) {  
 console.error('Error during logout:', error);  
 }  
 };  
  
 React.useEffect(() => {  
 const initializeAuth = async () => {  
 try {  
 const [authToken, userCookie, isLoginCookie, firstLoginCookie] = await Promise.all([  
 AsyncStorage.getItem('auth-token'),  
 AsyncStorage.getItem('user'),  
 AsyncStorage.getItem('is-login'),  
 AsyncStorage.getItem('first-login')  
 ]);  
  
 if (authToken && userCookie && isLoginCookie === 'true') {  
 setIsLogin(true);  
 setUser(userCookie);  
 setToken(authToken);  
   
 // Fetch and set user profile  
 try {  
 const response = await fetch(`${host}files/profile/student`, {  
 headers: {  
 'auth-token': authToken  
 }  
 });  
 if (response.ok) {  
 const blob = await response.blob();  
 const imageUrl = URL.createObjectURL(blob);  
 setUserProfile(imageUrl);  
 }  
 } catch (error) {  
 // If profile fetch fails, continue with other initialization  
 setUserProfile('');  
 }  
   
 if (firstLoginCookie) {  
 const isFirstLogin = firstLoginCookie === 'true';  
 setIsFirstLogin(isFirstLogin);  
 } else {  
 const isFirstLogin = await fetchFirstLogin(userCookie, authToken);  
 setIsFirstLogin(isFirstLogin);  
 await AsyncStorage.setItem('first-login', isFirstLogin.toString());  
 }  
  
 await fetchUserData(userCookie, authToken);  
 await fetchNotifications(authToken);  
 } else {  
 setIsLogin(false);  
 setUser(null);  
 setToken(null);  
 setIsFirstLogin(false);  
 setUserProfile('');  
 setNotifications([]);  
 setUnreadCount(0);  
 }  
 } catch (error) {  
 setIsLogin(false);  
 setUser(null);  
 setToken(null);  
 setIsFirstLogin(false);  
 setUserProfile('');  
 setNotifications([]);  
 setUnreadCount(0);  
 }  
 };  
  
 initializeAuth();  
 }, []);  
  
 return (  
 <UserContext.Provider value={{  
 host,  
 isLogin,  
 isFirstLogin,  
 user,  
 token,  
 userProfile,  
 userData,  
 notifications,  
 unreadCount,  
 setIsLogin,  
 setIsFirstLogin,  
 setUser,  
 setToken,  
 setUserProfile,  
 setUserData,  
 setNotifications,  
 setUnreadCount,  
 loadProfilePicture,  
 fetchUserData,  
 fetchFirstLogin,  
 fetchNotifications,  
 markNotificationAsRead,  
 logout  
 }}>  
 {children}  
 </UserContext.Provider>  
 );  
};  
  
export const useUser = () => {  
 const context = useContext(UserContext);  
 if (context === undefined) {  
 throw new Error('useUser must be used within a UserProvider');  
 }  
 return context;  
};  
  
export default UserContext;

# AlumniConnectApp\src\types\form.ts

export interface Form {  
 \_id: string;  
 docType: string;  
 status: string;  
 createdAt: string;  
 updatedAt: string;  
 student: {  
 \_id: string;  
 firstName: string;  
 lastName: string;  
 };  
 teacher: {  
 \_id: string;  
 firstName: string;  
 lastName: string;  
 };  
 documents: Array<{  
 name: string;  
 url: string;  
 }>;  
 remarks?: string;  
}  
  
export interface FormFilters {  
 status?: string;  
 date?: string;  
}

# AlumniConnectApp\src\types\student.ts

export interface Student {  
 \_id: string;  
 firstLogin: boolean;  
 password: string;  
 moodleid: string;  
 prn: string;  
 firstName: string;  
 fatherName: string;  
 lastName: string;  
 motherName: string;  
 grandfatherName: string;  
 mobileno: string;  
 email: string;  
 address: string;  
 religion: string;  
 caste: string;  
 category: string;  
 nationality: string;  
 placeOfBirth: string;  
 dateOfBirth: string;  
 lastCollegeAttended: string;  
 program: string;  
 branch: string;  
 semester: string;  
 yearOfJoining: string;  
 dateOfAdmission: string;  
 notifications: Array<{  
 message: string;  
 title: string;  
 from: string;  
 timestamp: string;  
 isRead: boolean;  
 }>;  
}  
  
export interface StudentFormData {  
 firstName: string;  
 fatherName: string;  
 lastName: string;  
 motherName: string;  
 grandfatherName: string;  
 prn: string;  
 moodleid: string;  
 mobileno: string;  
 email: string;  
 address: string;  
 placeOfBirth: string;  
 dateOfBirth: string;  
 branch: string;  
 program: string;  
 semester: string;  
 yearOfJoining: string;  
 dateOfAdmission: string;  
 lastCollegeAttended: string;  
 religion: string;  
 caste: string;  
 category: string;  
 nationality: string;  
}