**BAHRIA UNIVERSITY, LAHORE CAMPUS   
WEB COURSE PROJECT   
Submitted to: Ma’am Rida**

***Group Members:***

***Sahil, Nabiha, Momina, Awais***

***CODE with OUTPUT:***

***Frontend:***

**Home screen :   
Code:**   
  
import React, { useState } from "react";

import { Link } from "react-router-dom";

import jobVisual from './assets/undraw\_drag-and-drop\_v4po.png';

import "./Homepage.css";

export default function HomePage() {

   const [isMobileMenuOpen, setIsMobileMenuOpen] = useState(false);

  const toggleMenu = () => {

    setIsMobileMenuOpen(!isMobileMenuOpen);

  };

  return (

    <div className="homepage">

      <header className="navbar">

        <div className="navbar-left">

          <span className="logo">Trackflow</span>

        </div>

        <div className="mobile-menu-icon" onClick={toggleMenu}>

          &#9776;

        </div>

        <nav className={`navbar-right ${isMobileMenuOpen ? "open" : ""}`}>

          <Link to="/" onClick={toggleMenu}>Home</Link>

          <Link to="/AllJobs" onClick={toggleMenu}>All Jobs</Link>

          <Link to="/Auth" className="nav-button" onClick={toggleMenu}>Login/SignUP</Link>

          {/\* <Link to="/Auth" className="nav-button" onClick={toggleMenu}>Sign up</Link> \*/}

        </nav>

      </header>

      <div className="main-section">

        <div className="left-section">

          <h1>Track Your Dream Job Opportunities Effortlessly</h1>

          <p>Stay organized, stay ahead. Manage your job applications with ease and confidence.</p>

          <Link to="/AllJobs" onClick={toggleMenu}><button className="cta-button">Start Tracking Jobs Now</button></Link>

        </div>

        <div className="right-section">

          <img src={jobVisual} alt="Visual Job Tracking" className="hero-visual" />

        </div>

      </div>

      <section className="features">

        <h2>Key Features</h2>

        <div className="features-grid">

          <div className="feature-card">

            <h3>Job Posts</h3>

            <p>Employers post jobs that match your interests and career path.</p>

          </div>

          <div className="feature-card">

            <h3>Application Tracking</h3>

            <p>Track each application with clear statuses and updates.</p>

          </div>

          <div className="feature-card">

            <h3>Job Search</h3>

            <p>Search smart. Filter by location, company, and position easily.</p>

          </div>

        </div>

      </section>

      <section className="testimonials">

        <h2>What Users Are Saying</h2>

        <div className="testimonial-card">

          <p>"This app helped me stay on top of my job search. Highly recommended!"</p>

          <span>- Alex, Software Developer</span>

        </div>

      </section>

      <section className="blog-preview">

        <h2>Latest from Our Blog</h2>

        <div className="blog-post">

          <h3>5 Tips for Job Hunting in 2025</h3>

          <p>Discover smart strategies to improve your job search this year.</p>

        </div>

      </section>

      <footer className="footer">

        <p>&copy; 2025 Trackflow. All rights reserved.</p>

        <div className="footer-links">

          <a href="#">Privacy Policy</a>

          <a href="#">Terms of Service</a>

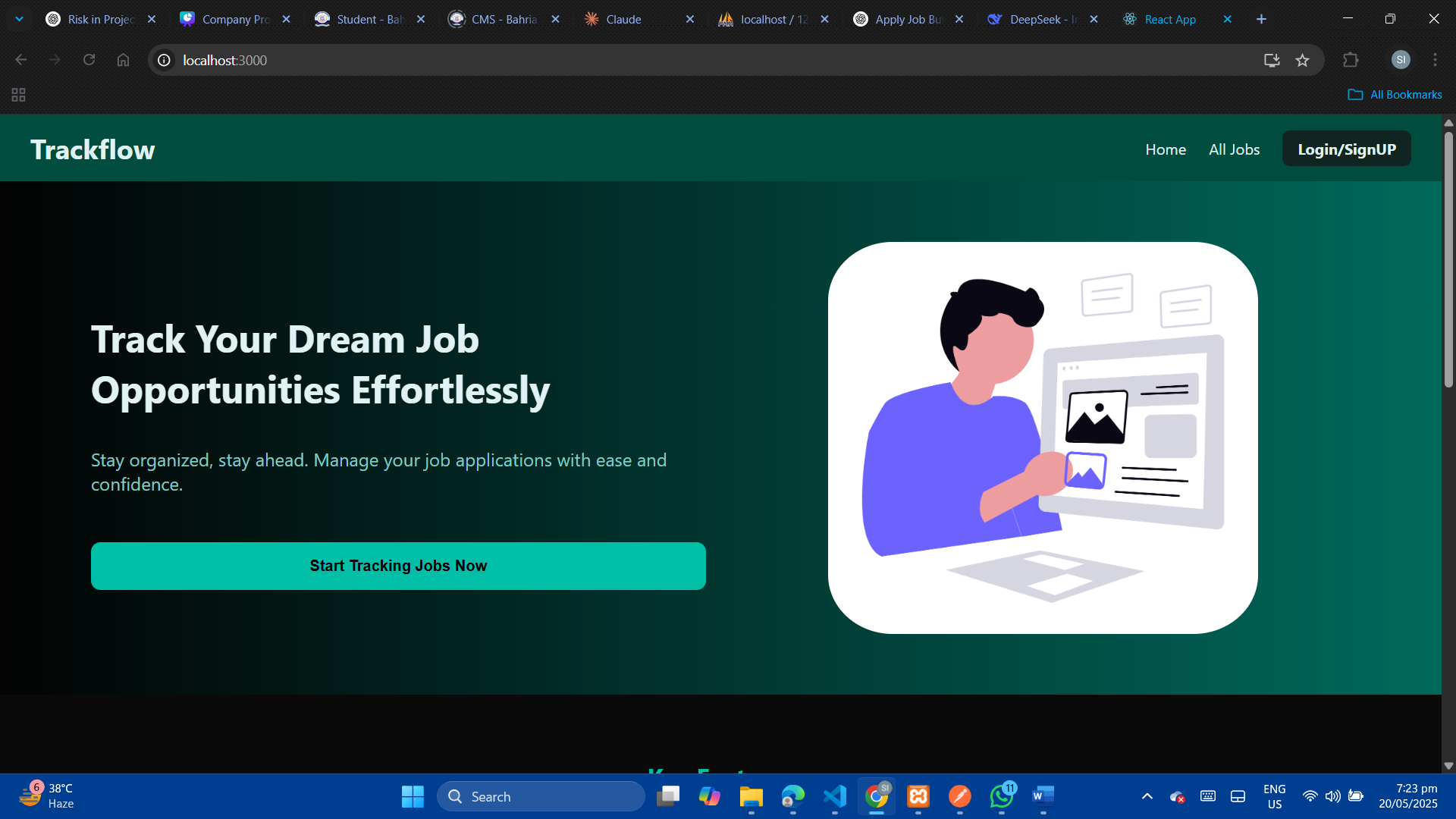
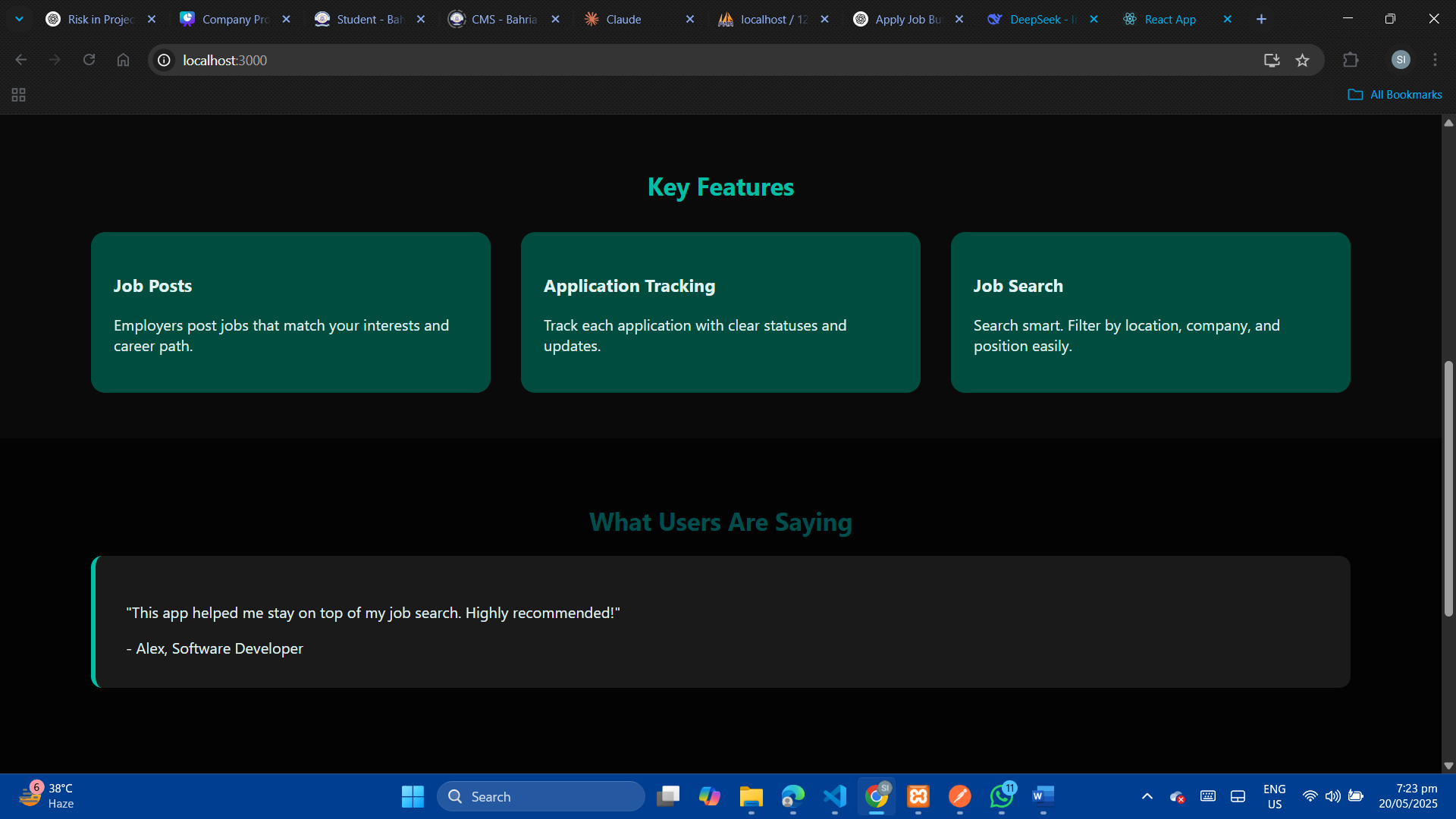
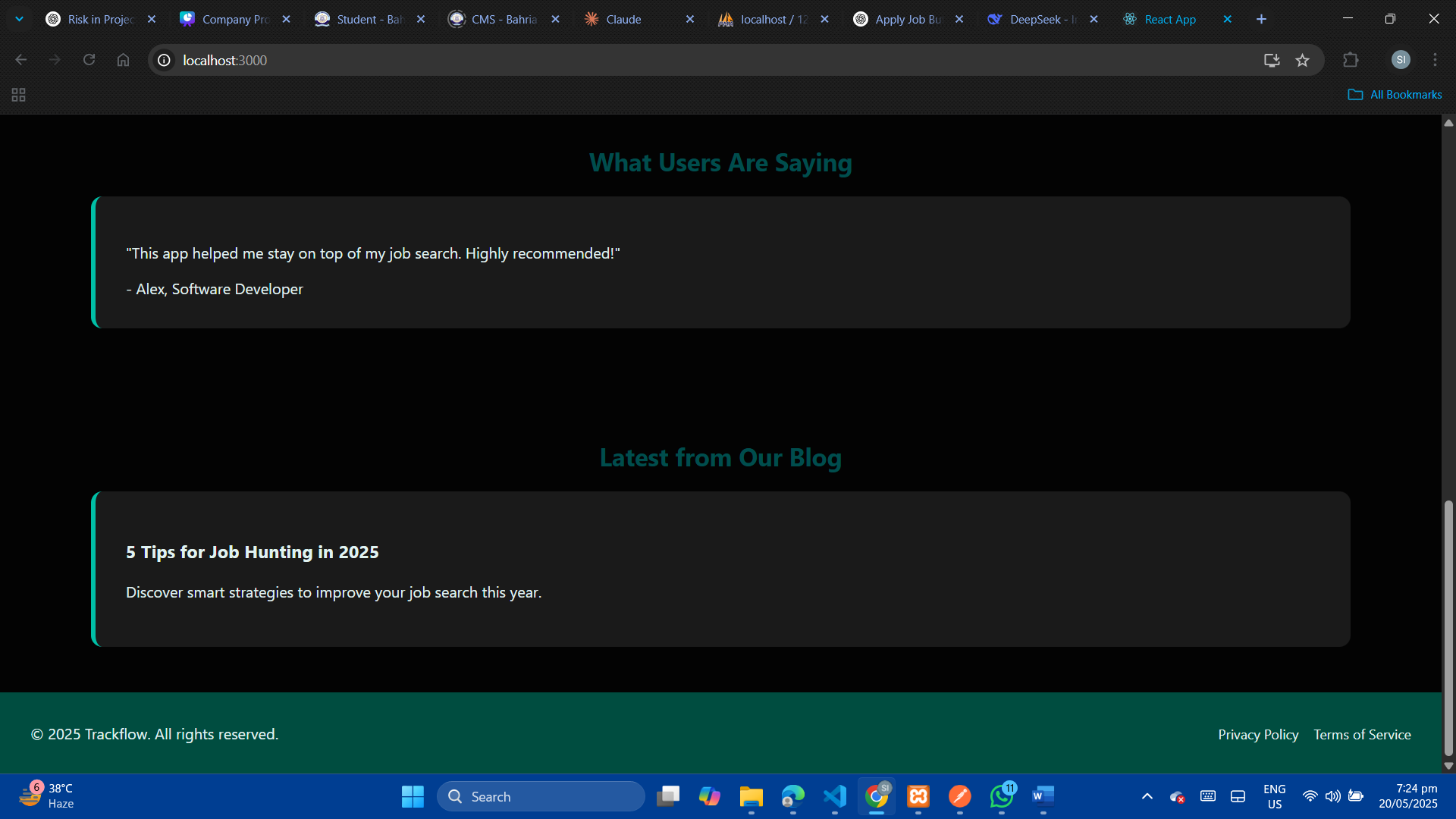
        </div>

      </footer>

    </div>

  );

}

**OutPut:**  
  
  
  
  
  
  
  
**Auth.jsx:**

Code:   
import React, { useState } from "react";

import axios from "axios";

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

import "./Auth.css";

const SignupLogin = () => {

  const [isLogin, setIsLogin] = useState(true);

  const [formData, setFormData] = useState({

    name: "",

    email: "",

    password: "",

    gender: "",

    role: "user",

  });

  const navigate = useNavigate();

  const toggleForm = () => {

    setIsLogin((prev) => !prev);

    setFormData({

      name: "",

      email: "",

      password: "",

      gender: "",

      role: "user",

    });

  };

  const handleChange = (e) => {

    const { name, value } = e.target;

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

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

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

      alert("Email and password are required.");

      return;

    }

    if (!isLogin && (!formData.name || !formData.gender || !formData.role)) {

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

      return;

    }

    try {

      let response;

      if (isLogin) {

        response = await axios.post("http://localhost:8000/api/login", {

          email: formData.email,

          password: formData.password,

        });

      } else {

        response = await axios.post("http://localhost:8000/api/create-user", formData);

      }

      const userData = response.data?.data;

      const token = response.data?.token;

      const userRole = userData?.role || formData.role;

      // Store user data in localStorage

      localStorage.setItem("email", userData?.email || formData.email);

      localStorage.setItem("role", userRole);

      localStorage.setItem("token", token);

      alert(response.data.message || (isLogin ? "Login successful!" : "Signup successful!"));

      // Navigate based on role

      if (userRole === "user") {

        navigate("/user-dashboard");

      } else if (userRole === "recruiter") {

        navigate("/recruiter-dashboard");

      } else {

        alert("Unknown role. Please contact support.");

      }

    } catch (error) {

      const msg =

        error?.response?.data?.message ||

        "Something went wrong. Please try again.";

      alert(msg);

      console.error("Auth error:", error);

    }

  };

  return (

    <div className="auth-page">

      <div className="auth-card">

        <h2>{isLogin ? "Welcome Back" : "Create Your Account"}</h2>

        <form onSubmit={handleSubmit} className="auth-form">

          {!isLogin && (

            <>

              <input

                type="text"

                name="name"

                placeholder="Full Name"

                value={formData.name}

                onChange={handleChange}

                required

              />

              <select

                name="gender"

                value={formData.gender}

                onChange={handleChange}

                required

              >

                <option value="">Select Gender</option>

                <option value="male">Male</option>

                <option value="female">Female</option>

              </select>

            </>

          )}

          <input

            type="email"

            name="email"

            placeholder="Email"

            value={formData.email}

            onChange={handleChange}

            required

          />

          <input

            type="password"

            name="password"

            placeholder="Password"

            value={formData.password}

            onChange={handleChange}

            required

          />

          {!isLogin && (

            <select

              name="role"

              value={formData.role}

              onChange={handleChange}

              required

            >

              <option value="user">User</option>

              <option value="recruiter">Recruiter</option>

            </select>

          )}

          <button type="submit" className="auth-button">

            {isLogin ? "Login" : "Sign Up"}

          </button>

        </form>

        <p className="toggle-text">

          {isLogin ? "Don't have an account?" : "Already have an account?"}{" "}

          <span onClick={toggleForm} className="toggle-link">

            {isLogin ? "Sign up" : "Login"}

          </span>

        </p>

      </div>

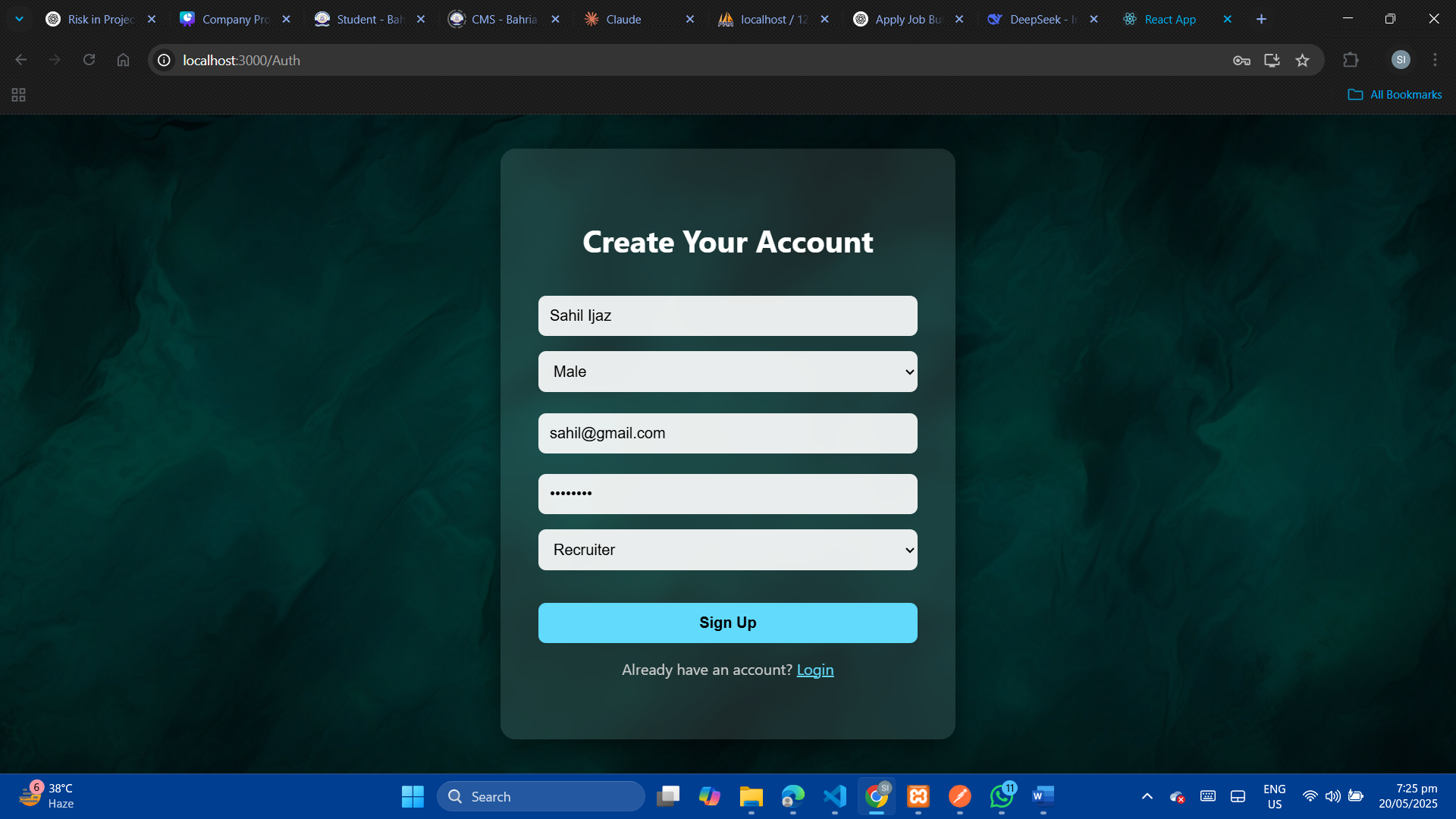
    </div>

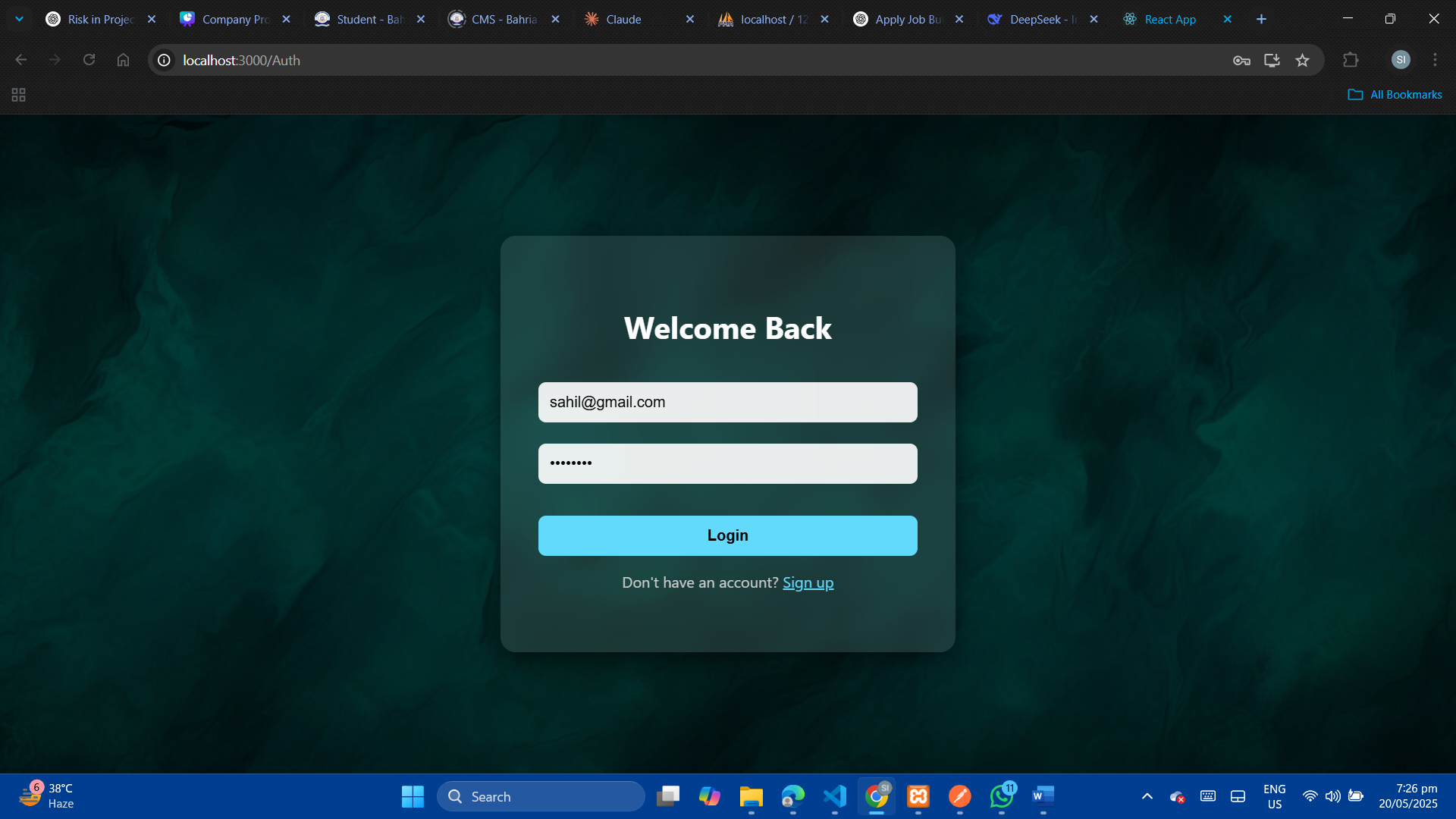
  );

};

export default SignupLogin;

**OutPut:**

**SignUp:**  


LogIn:   


All jobs page:   
cde:   
import React, { useState, useEffect } from "react";

import axios from "axios";

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

import "./AllJobsPage.css";

export default function AllJobs() {

  const [jobs, setJobs] = useState([]);

  const [filterStatus, setFilterStatus] = useState("All");

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

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

  const [token, setToken] = useState("");

  const navigate = useNavigate();

  // Load token from localStorage on mount

  useEffect(() => {

    const storedToken = localStorage.getItem("token");

    setToken(storedToken || "");

  }, []);

  // Fetch jobs whenever filterStatus changes

  useEffect(() => {

    const fetchJobs = async () => {

      try {

        setLoading(true);

        const statusQuery = filterStatus === "All" ? "" : `?status=${filterStatus}`;

        const response = await axios.get(`http://localhost:8000/api/all-jobs${statusQuery}`);

        setJobs(response.data.jobs);

        setError("");

      } catch (err) {

        setError("Failed to fetch jobs. Please try again later.");

      } finally {

        setLoading(false);

      }

    };

    fetchJobs();

  }, [filterStatus]);

  // When Apply Job button clicked

  const handleApplyClick = (jobId) => {

    if (!token) {

      alert("You are not logged in. Please login to apply.");

      return;

    }

    localStorage.setItem("jobId", jobId); // Save job id in localStorage

    navigate("/apply-job"); // Navigate to application form

  };

  // Filter jobs by status

  const filteredJobs =

    filterStatus === "All"

      ? jobs

      : jobs.filter((job) => job.status.toLowerCase() === filterStatus.toLowerCase());

  return (

    <div className="all-jobs-page">

      <h2 className="page-title">All Job Listings</h2>

      <div className="filter-container">

        <label htmlFor="statusFilter">Filter by status:</label>

        <select

          id="statusFilter"

          value={filterStatus}

          onChange={(e) => setFilterStatus(e.target.value)}

        >

          <option value="All">All</option>

          <option value="open">Opened</option>

          <option value="closed">Closed</option>

        </select>

      </div>

      {loading ? (

        <p>Loading jobs...</p>

      ) : error ? (

        <p className="error">{error}</p>

      ) : filteredJobs.length === 0 ? (

        <p className="no-jobs">No jobs found for this status.</p>

      ) : (

        <div className="job-list">

          {filteredJobs.map((job) => (

            <div key={job.id} className="job-card">

              <h3>{job.jobTitle}</h3>

              <p><strong>Company:</strong> {job.companyName}</p>

              <p><strong>Location:</strong> {job.location}</p>

              <p><strong>Salary:</strong> {job.salary}</p>

              <p><strong>Description:</strong> {job.description}</p>

              <p><strong>Requirements:</strong> {job.requirements}</p>

              <p><strong>Status:</strong> {job.status}</p>

              <p><strong>Posted on:</strong> {new Date(job.created\_at).toLocaleDateString()}</p>

              <button

                className="apply-button"

                onClick={() => handleApplyClick(job.id)}

              >

                Apply Job

              </button>

            </div>

          ))}

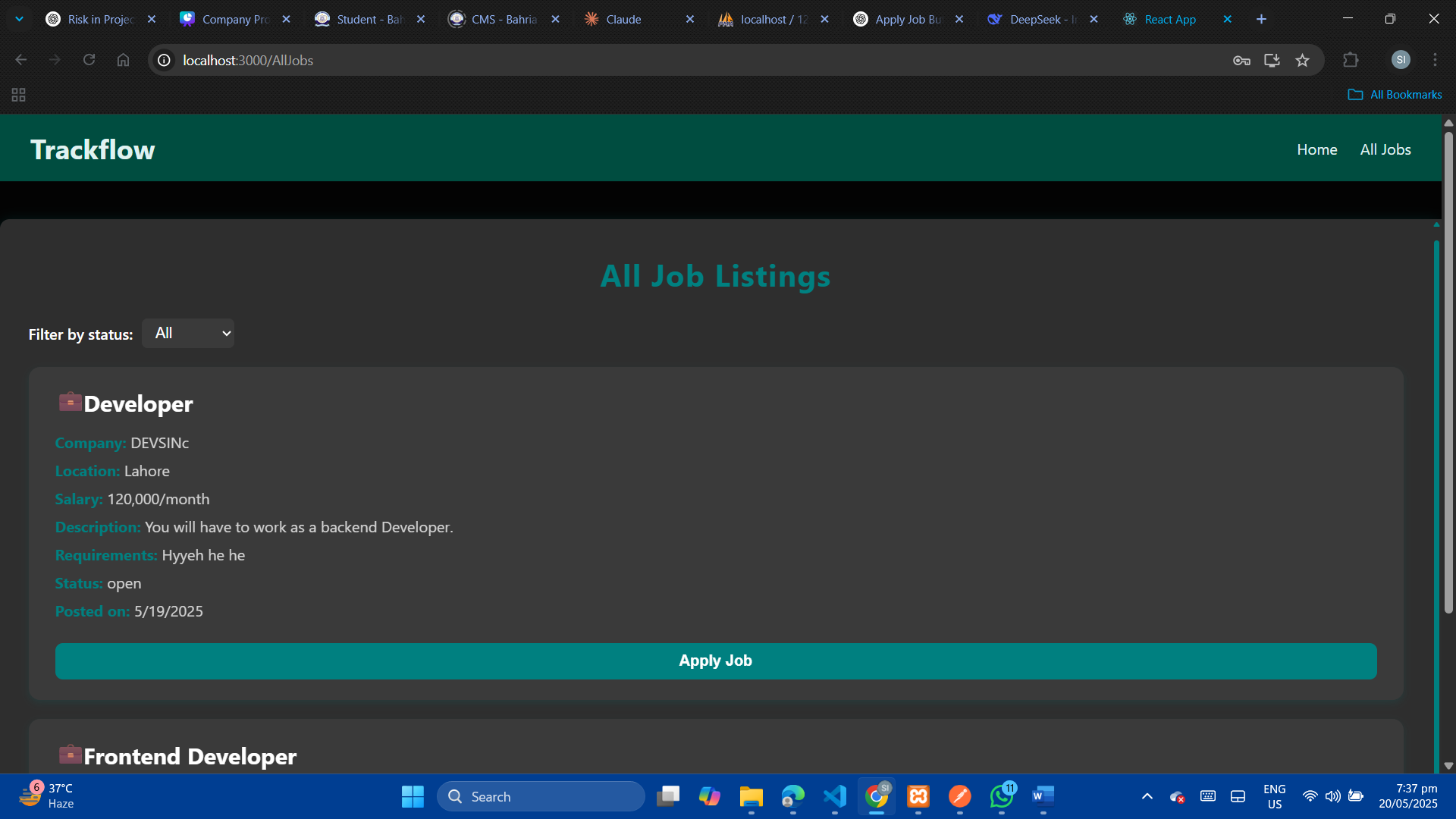
        </div>

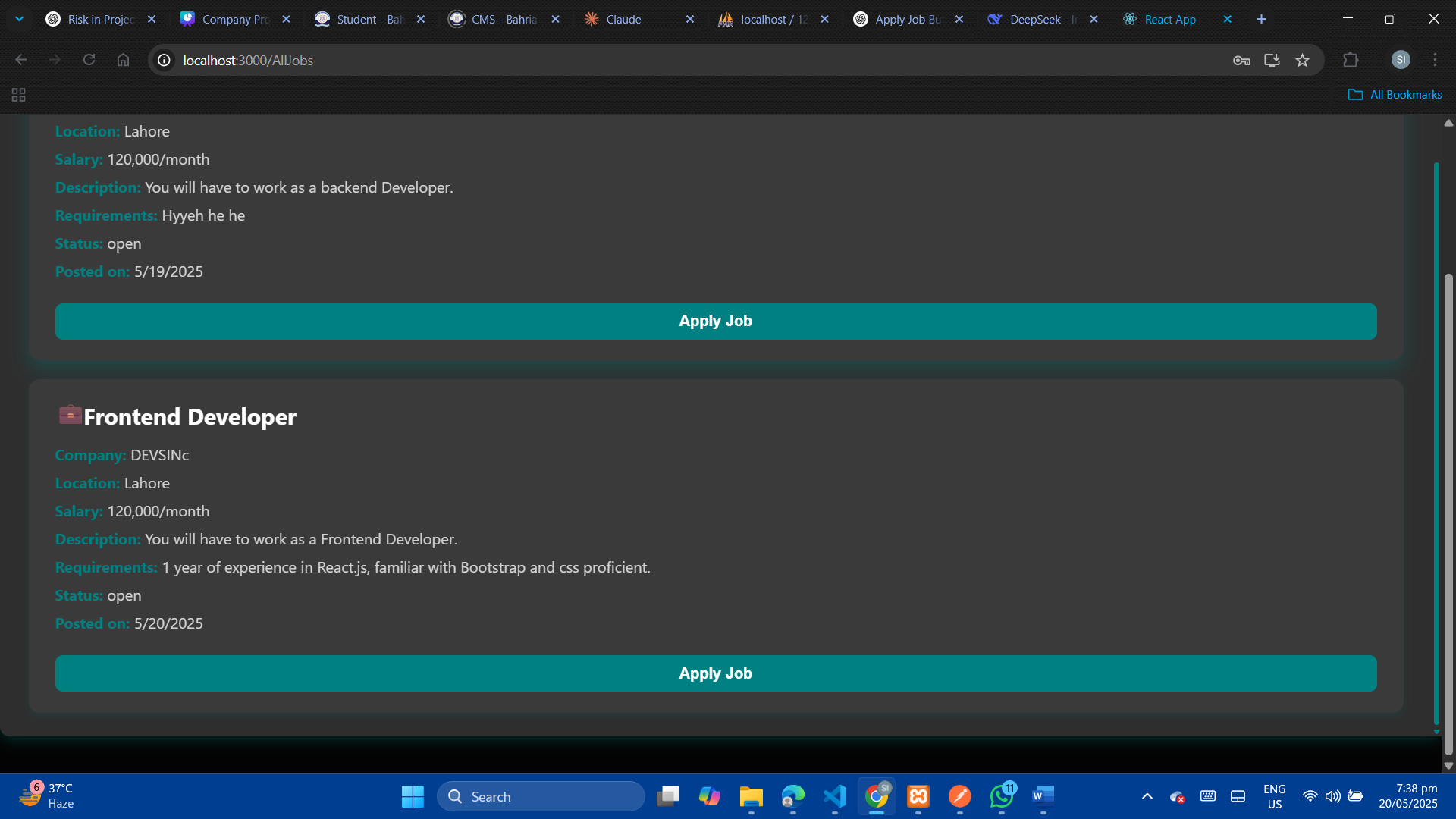
      )}

    </div>

  );

}

**Output:**  




**Apply job:   
Code:**   
import React, { useState, useEffect } from "react";

import axios from "axios";

import "./ApplyJob.css";

const JobApplicationForm = () => {

  const [jobId, setJobId] = useState("");

  const [formData, setFormData] = useState({

    name: "",

    email: "",

    contact: "",

    resume: "",

    experience: "",

    expectations: "",

  });

  useEffect(() => {

    const savedJobId = localStorage.getItem("jobId");

    if (savedJobId) {

      setJobId(savedJobId);

      console.log("Loaded jobId from localStorage:", savedJobId);

    }

  }, []);

  const handleChange = (e) => {

    setFormData((prev) => ({ ...prev, [e.target.name]: e.target.value }));

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    if (!jobId) {

      alert("No job selected. Please go back and select a job.");

      return;

    }

    const requestBody = {

      job\_id: jobId,

      name: formData.name,

      email: formData.email,

      contactNumber: formData.contact,

      resumeLink: formData.resume,

      experience: formData.experience,

      expectations: formData.expectations,

    };

    console.log("Submitting application for job ID:", jobId);

    console.log("Request Body:", JSON.stringify(requestBody, null, 2));

    try {

      const response = await axios.post(

        "http://localhost:8000/api/apply-job",

        requestBody,

        {

          headers: {

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

          },

        }

      );

      alert("Application submitted successfully!");

      console.log("Server response:", response.data);

      localStorage.removeItem("jobId");

      setFormData({

        name: "",

        email: "",

        contact: "",

        resume: "",

        experience: "",

        expectations: "",

      });

    } catch (error) {

      console.error(

        "Error submitting application:",

        error.response?.data || error.message

      );

      alert("Failed to submit application. Please try again later.");

    }

  };

  return (

    <div className="job-form-bg">

      <div className="form-container">

        <h2>Job Application Form</h2>

        {jobId ? (

          <p>

            Applying for Job ID: <strong>{jobId}</strong>

          </p>

        ) : (

          <p style={{ color: "red" }}>

            No job selected. Please go back and select a job.

          </p>

        )}

        <form onSubmit={handleSubmit}>

          <label>

            Name:

            <input

              type="text"

              name="name"

              required

              value={formData.name}

              onChange={handleChange}

            />

          </label>

          <label>

            Email:

            <input

              type="email"

              name="email"

              required

              value={formData.email}

              onChange={handleChange}

            />

          </label>

          <label>

            Contact Number:

            <input

              type="tel"

              name="contact"

              required

              value={formData.contact}

              onChange={handleChange}

            />

          </label>

          <label>

            Resume Link:

            <input

              type="url"

              name="resume"

              required

              value={formData.resume}

              onChange={handleChange}

            />

          </label>

          <label>

            Experience:

            <textarea

              name="experience"

              rows="4"

              required

              value={formData.experience}

              onChange={handleChange}

            />

          </label>

          <label>

            Expectations:

            <textarea

              name="expectations"

              rows="3"

              required

              value={formData.expectations}

              onChange={handleChange}

            />

          </label>

          <button type="submit" disabled={!jobId}>

            Apply Now

          </button>

        </form>

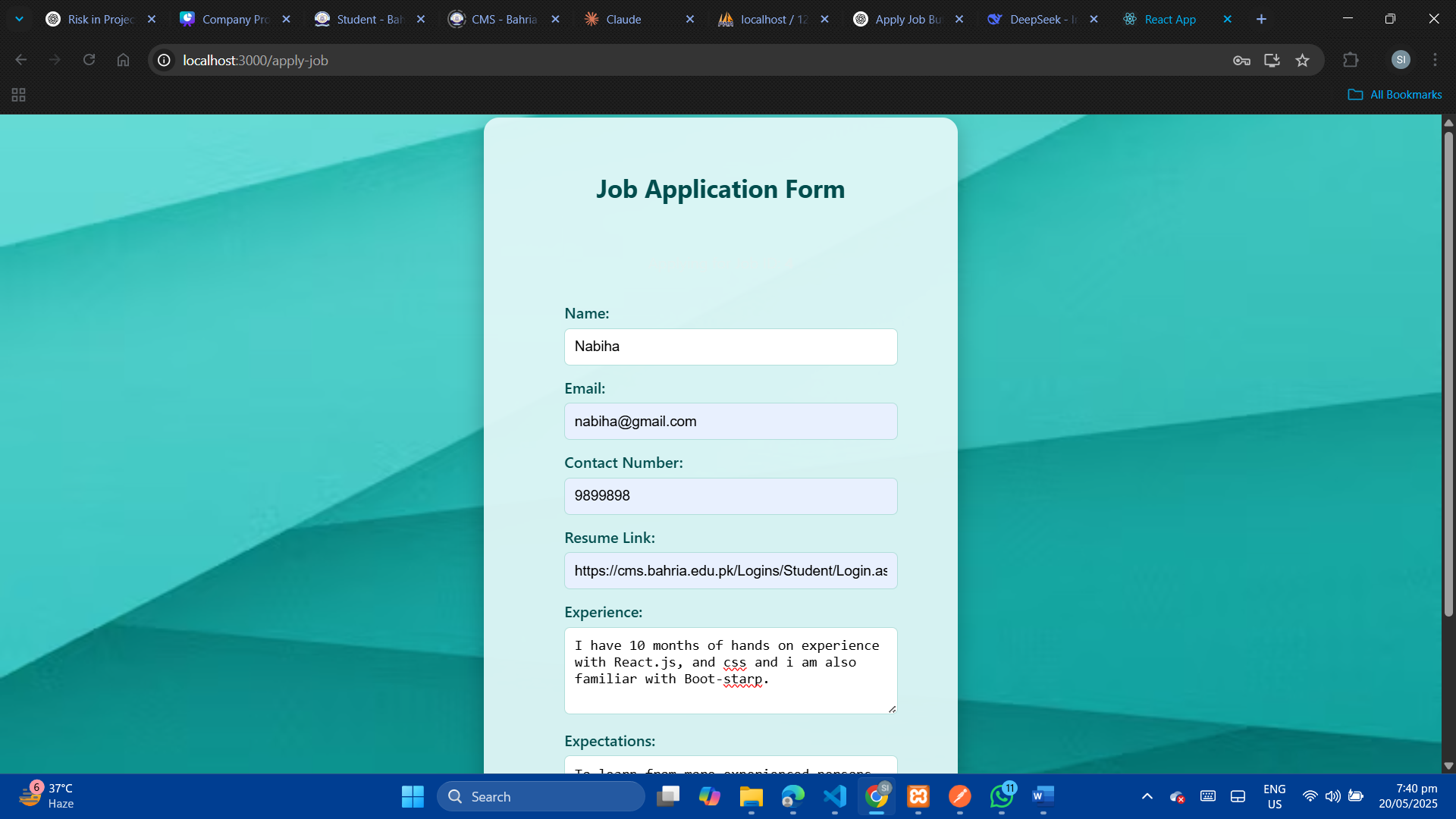
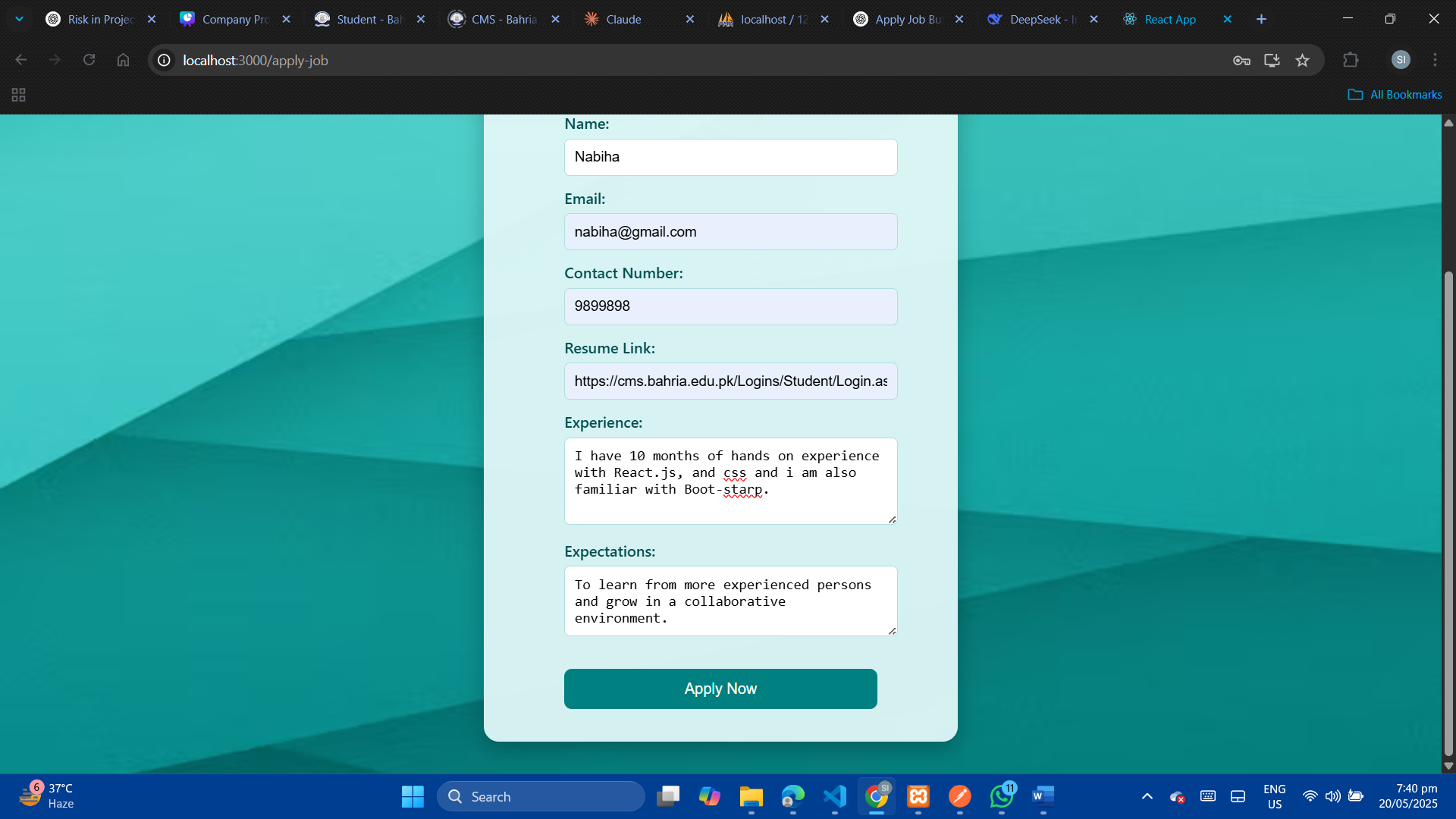
      </div>

    </div>

  );

};

export default JobApplicationForm;

**Output:**   
  


**User dashboard:   
Code:**   
import React, { useState, useEffect } from "react";

import axios from "axios";

import "./UserDashboard.css";

export default function UserDashboard() {

  const [email, setEmail] = useState("");

  const [appliedJobs, setAppliedJobs] = useState([]);

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

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

  useEffect(() => {

    const storedEmail = localStorage.getItem("email");

    if (storedEmail) {

      setEmail(storedEmail);

      fetchApplications(storedEmail);

    }

  }, []);

  const fetchApplications = async (emailToUse = email) => {

    if (!emailToUse) {

      setError("Email is missing. Please login again.");

      return;

    }

    setLoading(true);

    setError("");

    try {

      const response = await axios.post(

        "http://localhost:8000/api/applications-by-email",

        { email: emailToUse }

      );

      if (response.data && response.data.applications) {

        setAppliedJobs(response.data.applications);

      } else {

        setAppliedJobs([]);

      }

    } catch (err) {

      console.error(err);

      setError("Failed to load applications.");

    } finally {

      setLoading(false);

    }

  };

  return (

    <div className="user-dashboard">

      <h2 className="dashboard-title">Your Applications</h2>

      {loading && <p>Loading applications...</p>}

      {error && <p className="error">{error}</p>}

      <div className="application-list">

        {appliedJobs.length === 0 && !loading && <p>No applications found.</p>}

        {appliedJobs.map((job) => (

          <div key={job.id} className="application-card">

            <p><strong>Job Applied:</strong> {job.job\_title}</p>

            <p><strong>Applicant Name:</strong> {job.applicant\_name}</p>

            <p><strong>Applied on:</strong> {new Date(job.appliedAt).toLocaleString()}</p>

          </div>

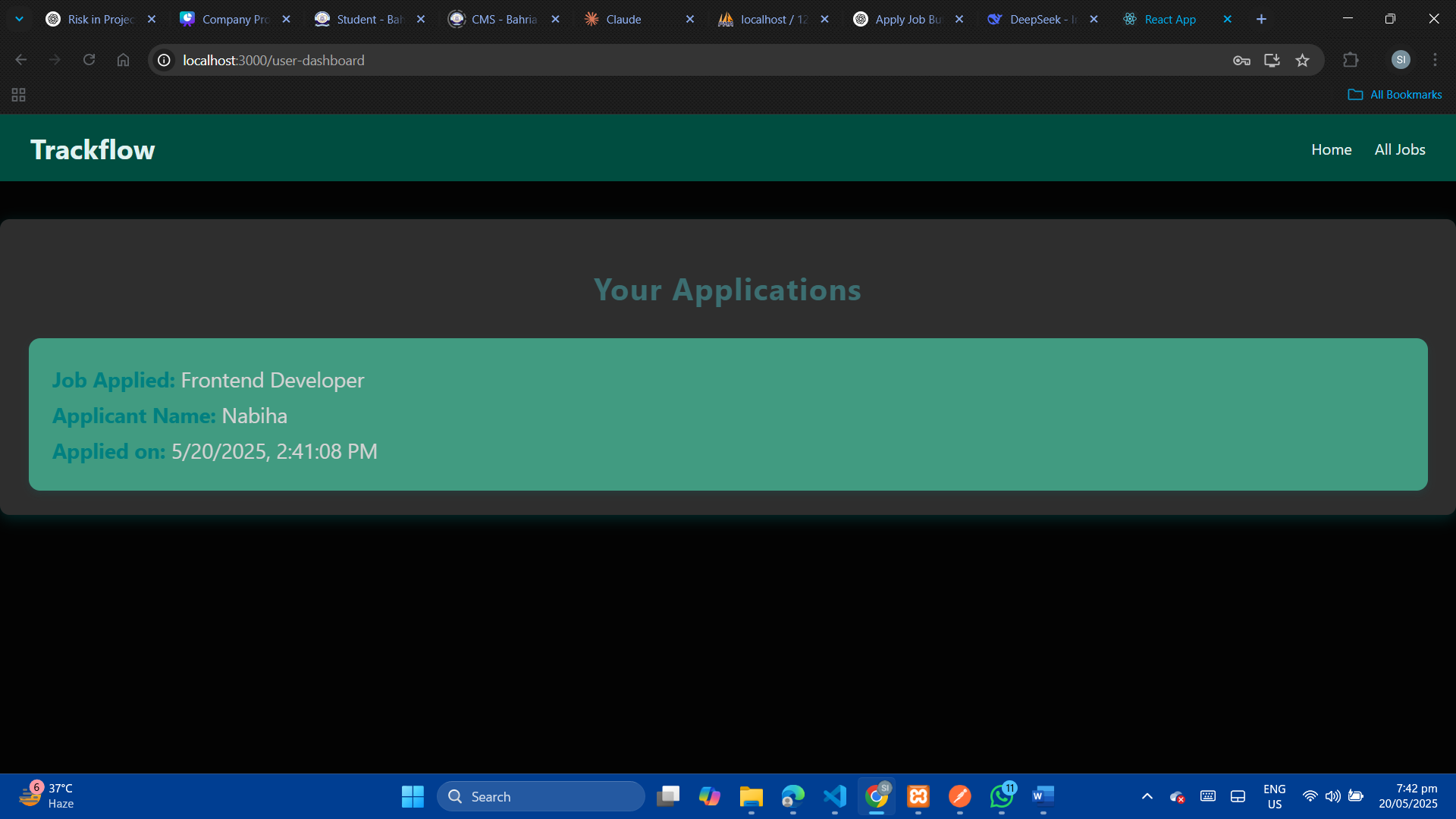
        ))}

      </div>

    </div>

  );

}

**Output:**   


**Recruiter dashbrd:   
Code:**   
import React, { useState, useEffect } from "react";

import axios from "axios";

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

import "./RecruiterDashboard.css";

export default function RecruiterDashboard() {

  const [jobs, setJobs] = useState([]);

  const [editId, setEditId] = useState(null);

  const [editData, setEditData] = useState({});

  const [email, setEmail] = useState("");

  const [activeTab, setActiveTab] = useState("All");

  const [fetchTrigger, setFetchTrigger] = useState(0);

  const navigate = useNavigate();

  useEffect(() => {

    const savedEmail = localStorage.getItem("email");

    const savedRole = localStorage.getItem("role");

    if (savedRole !== "recruiter") {

      navigate("/");

    }

    if (savedEmail) {

      setEmail(savedEmail);

    }

  }, [navigate]);

  useEffect(() => {

    if (!email) {

      setJobs([]);

      return;

    }

    const fetchJobs = async () => {

      try {

        const url = `http://localhost:8000/api/jobs-by-creator?email=${email}`;

        const response = await axios.get(url);

        const transformedJobs = response.data.jobs.map((job) => ({

          id: job.id,

          title: job.jobTitle,

          company: job.companyName,

          location: job.location,

          jobType: job.employmentType,

          description: job.description,

          status: job.status === "open" ? "Active" : "Closed",

          salary: job.salary || "",

          requirements: job.requirements || "",

        }));

        setJobs(transformedJobs);

      } catch (error) {

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

        setJobs([]);

      }

    };

    fetchJobs();

  }, [email, fetchTrigger]);

  const handleDelete = async (id) => {

    try {

      await axios.delete("http://localhost:8000/api/delete-job", {

        data: { job\_id: id, email },

      });

      setJobs(jobs.filter((job) => job.id !== id));

    } catch (error) {

      console.error("Failed to delete job:", error);

    }

  };

  const handleEdit = (job) => {

    setEditId(job.id);

    setEditData(job);

  };

  const handleChange = (e) => {

    setEditData({ ...editData, [e.target.name]: e.target.value });

  };

  const handleUpdate = async (e) => {

    e.preventDefault();

    try {

      const updatedJob = {

        creator\_id: 1,

        jobTitle: editData.title,

        employmentType: editData.jobType,

        location: editData.location,

        salary: editData.salary,

        companyName: editData.company,

        requirements: editData.requirements,

        description: editData.description,

        status: editData.status === "Active" ? "open" : "closed",

      };

      await axios.patch(

        `http://localhost:8000/api/update-job/${editId}`,

        updatedJob

      );

      setJobs(jobs.map((job) => (job.id === editId ? { ...editData } : job)));

      setEditId(null);

    } catch (error) {

      console.error("Failed to update job:", error);

    }

  };

  const filteredJobs =

    activeTab === "All" ? jobs : jobs.filter((job) => job.status === activeTab);

  return (

    <div className="recruiter-dashboard">

      <h2 className="dashboard-title">Recruiter Dashboard</h2>

      <div className="tabs">

        {["All", "Active", "Closed"].map((tab) => (

          <button

            key={tab}

            className={`tab-button ${activeTab === tab ? "active" : ""}`}

            onClick={() => setActiveTab(tab)}

          >

            {tab}

          </button>

        ))}

      </div>

      <div className="jobs-list">

        {filteredJobs.length === 0 ? (

          <p>No jobs found.</p>

        ) : (

          filteredJobs.map((job) => (

            <div key={job.id} className="job-card">

              {editId === job.id ? (

                <form onSubmit={handleUpdate} className="edit-form">

                  <input type="text" name="title" value={editData.title} onChange={handleChange} required />

                  <input type="text" name="company" value={editData.company} onChange={handleChange} required />

                  <input type="text" name="location" value={editData.location} onChange={handleChange} required />

                  <input type="text" name="jobType" value={editData.jobType} onChange={handleChange} required />

                  <input type="text" name="salary" value={editData.salary} onChange={handleChange} />

                  <textarea name="requirements" value={editData.requirements} onChange={handleChange} />

                  <textarea name="description" value={editData.description} onChange={handleChange} required />

                  <select name="status" value={editData.status} onChange={handleChange} required>

                    <option value="Active">Active</option>

                    <option value="Closed">Closed</option>

                  </select>

                  <button type="submit">Update</button>

                </form>

              ) : (

                <>

                  <h3>{job.title}</h3>

                  <p><strong>Company:</strong> {job.company}</p>

                  <p><strong>Location:</strong> {job.location}</p>

                  <p><strong>Type:</strong> {job.jobType}</p>

                  <p><strong>Salary:</strong> {job.salary}</p>

                  <p><strong>Description:</strong> {job.description}</p>

                  <p><strong>Status:</strong> {job.status}</p>

                  <button onClick={() => handleEdit(job)}>Edit</button>

                  <button onClick={() => handleDelete(job.id)}>Delete</button>

                </>

              )}

            </div>

          ))

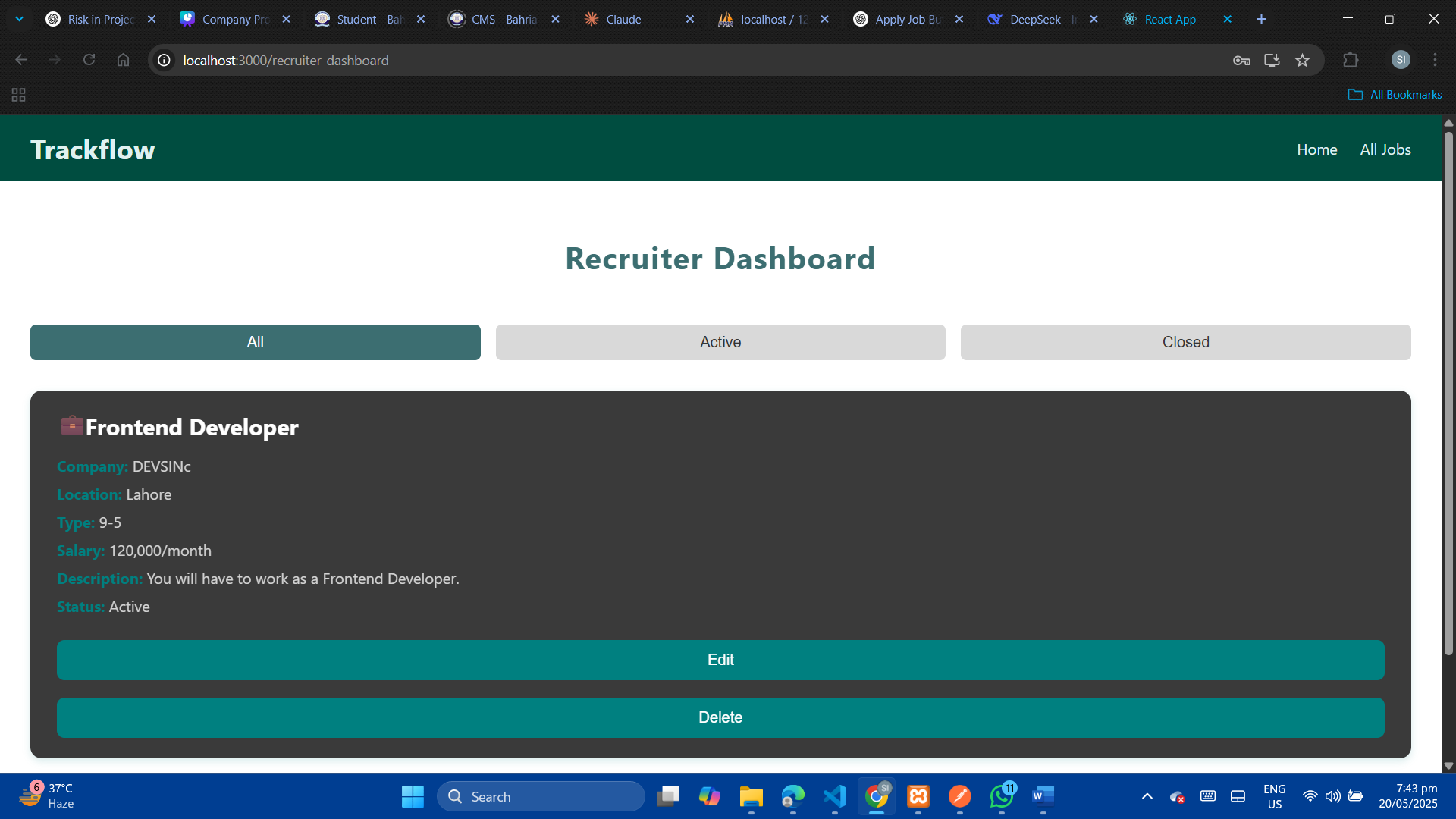
        )}

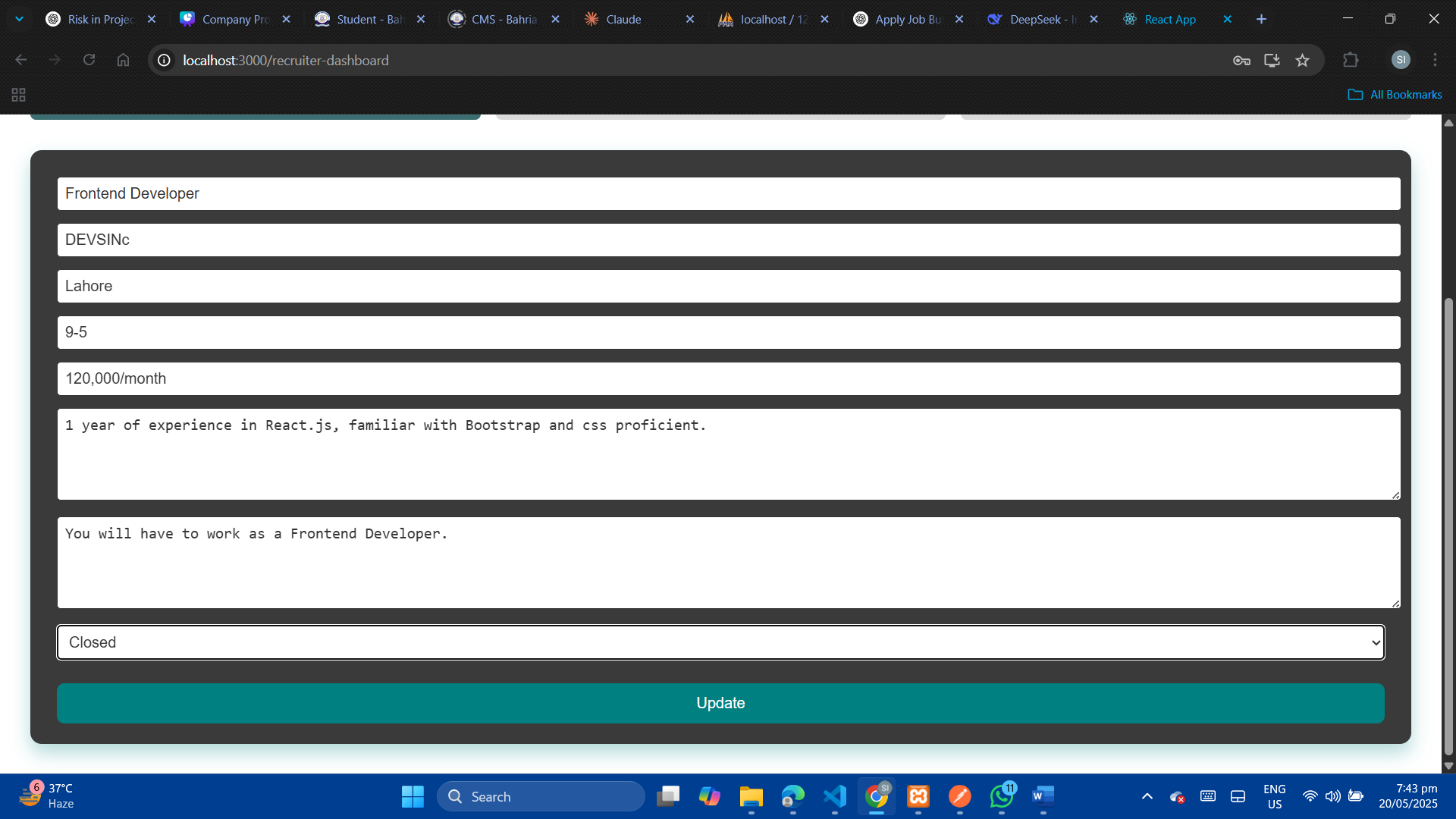
      </div>

    </div>

  );

}

**Output:**   


Job being updated by recruiter :   


**Create job post:   
Code:**  
import React, { useState } from 'react';

import axios from 'axios';

import './JobPostForm.css';

import {

  FaBuilding,

  FaMapMarkerAlt,

  FaRegFileAlt,

  FaClipboardList,

  FaBriefcase,

  FaSuitcase,

  FaEnvelope,

} from 'react-icons/fa';

export default function JobPostForm() {

  const [formData, setFormData] = useState({

    jobTitle: '',

    companyName: '',

    location: '',

    employmentType: '',

    salary: '',

    description: '',

    requirements: '',

    status: 'open', // default status

    email: '', // new field

  });

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

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

  const [success, setSuccess] = useState('');

  const handleChange = (e) => {

    const { name, value } = e.target;

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

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    setLoading(true);

    setError('');

    setSuccess('');

    try {

      const response = await axios.post('http://localhost:8000/api/create-job', formData);

      setSuccess('Your job has been posted successfully!');

      setFormData({

        jobTitle: '',

        companyName: '',

        location: '',

        employmentType: '',

        salary: '',

        description: '',

        requirements: '',

        status: 'open',

        email: '',

      });

    } catch (err) {

      setError('Failed to post job. Please try again later.');

    }

    setLoading(false);

  };

  return (

    <div className="job-form-page">

      <div className="job-form-container">

        <h2 className="form-title">Create a Job Post</h2>

        <form className="job-form" onSubmit={handleSubmit}>

          <div className="input-group">

            <FaBriefcase className="icon" />

            <input

              type="text"

              name="jobTitle"

              required

              placeholder=" "

              value={formData.jobTitle}

              onChange={handleChange}

            />

            <label>Job Title</label>

          </div>

          <div className="input-group">

            <FaBuilding className="icon" />

            <input

              type="text"

              name="companyName"

              required

              placeholder=" "

              value={formData.companyName}

              onChange={handleChange}

            />

            <label>Company Name</label>

          </div>

          <div className="input-group">

            <FaMapMarkerAlt className="icon" />

            <input

              type="text"

              name="location"

              required

              placeholder=" "

              value={formData.location}

              onChange={handleChange}

            />

            <label>Location</label>

          </div>

          <div className="input-group">

            <FaEnvelope className="icon" />

            <input

              type="email"

              name="email"

              required

              placeholder=" "

              value={formData.email}

              onChange={handleChange}

            />

            <label>Creator's Email</label>

          </div>

          <div className="input-group select-group">

            <FaSuitcase className="icon" />

            <select

              name="employmentType"

              required

              value={formData.employmentType}

              onChange={handleChange}

            >

              <option value="" disabled>

                Select Job Type

              </option>

              <option value="Full-Time">Full-Time</option>

              <option value="Part-Time">Part-Time</option>

              <option value="Remote">Remote</option>

              <option value="Contract">Contract</option>

              <option value="Internship">Internship</option>

            </select>

          </div>

          <div className="input-group">

            <FaBriefcase className="icon" />

            <input

              type="text"

              name="salary"

              required

              placeholder=" "

              value={formData.salary}

              onChange={handleChange}

            />

            <label>Salary</label>

          </div>

          <div className="input-group textarea-group">

            <FaRegFileAlt className="icon" />

            <textarea

              name="description"

              required

              placeholder=" "

              value={formData.description}

              onChange={handleChange}

            ></textarea>

            <label>Job Description</label>

          </div>

          <div className="input-group textarea-group">

            <FaClipboardList className="icon" />

            <textarea

              name="requirements"

              required

              placeholder=" "

              value={formData.requirements}

              onChange={handleChange}

            ></textarea>

            <label>Requirements</label>

          </div>

          <button type="submit" className="post-btn" disabled={loading}>

            {loading ? 'Posting...' : 'Post Job'}

          </button>

        </form>

        {error && <p className="error">{error}</p>}

        {success && <p className="success">{success}</p>}

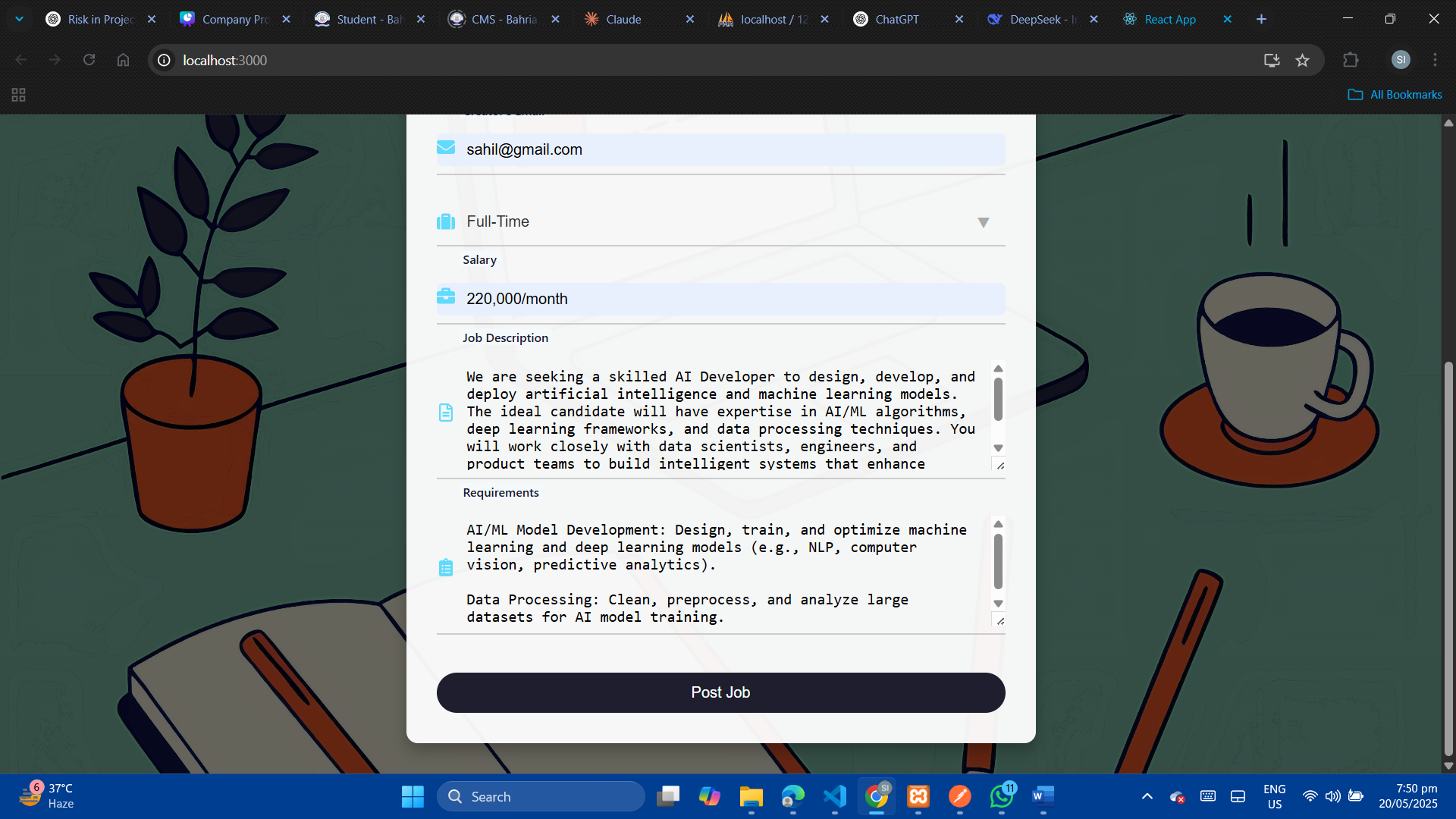
      </div>

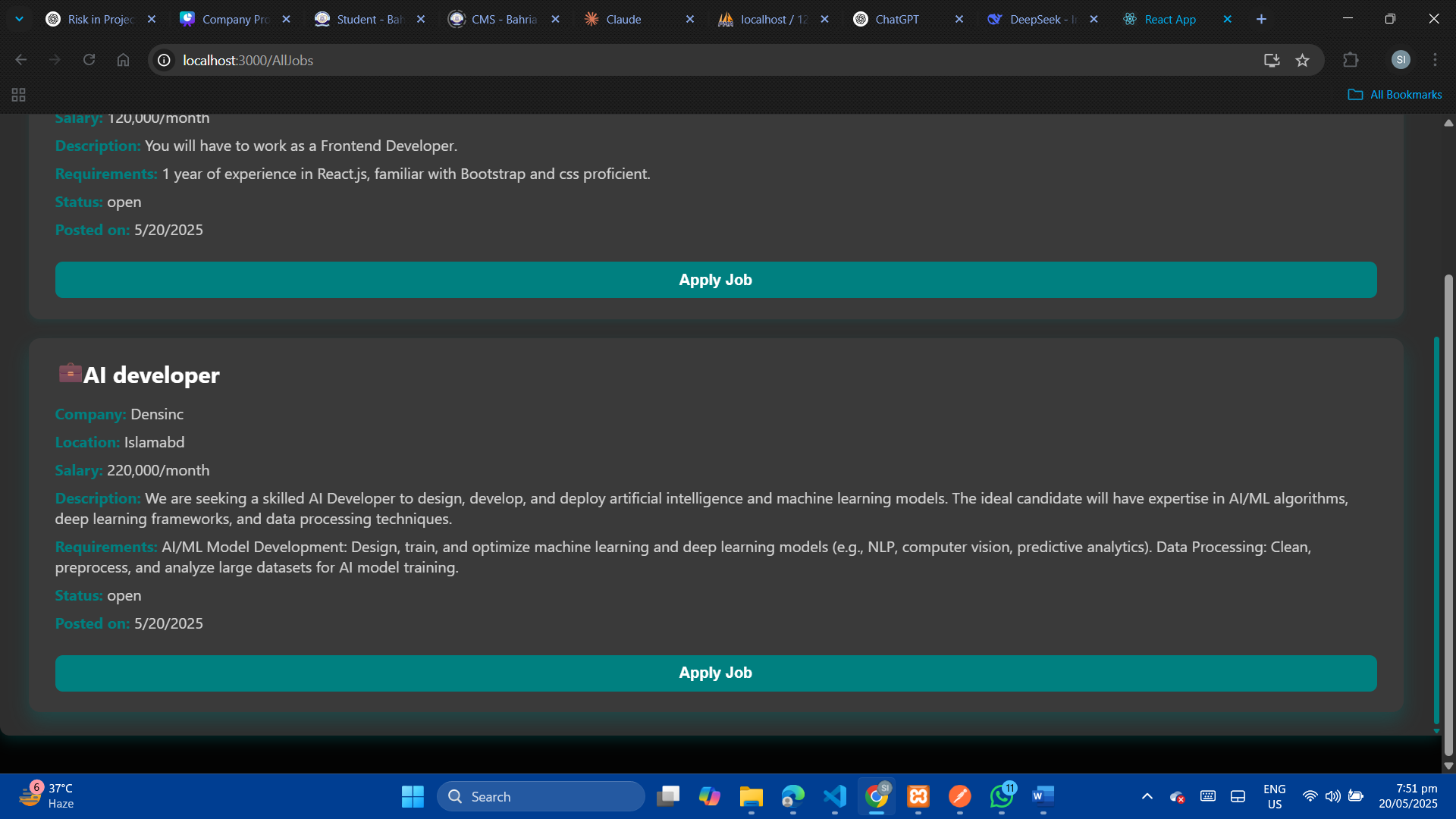
    </div>

  );

}

**Output:**   

Currently added job:   
  
  


**Backend:   
Models:**  
  
**user model:**  
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Factories\HasFactory;

class user extends Model

{

    //

    use HasFactory;

    protected $table = 'user';

    protected $fillable = [

        'name',

        'email',

        'gender',

        'role',

        'password'

    ];

}

**Jb model :**  
  
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use App\Models\User;

class table\_jobs\_uploading extends Model

{

    //

    use HasFactory;

    protected $table = 'table\_jobs\_uploading';

    protected $fillable = [

        'jobTitle',

        'employmentType',

        'location',

        'salary',

        'description',

        'creator',

        'status',

        'companyName',

        'requirements',

    ];

    public function creator()

    {

        return $this->belongsTo(User::class, 'creator');

    }

}

**applications model :**  
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use App\Models\table\_jobs\_uploading;

use App\Models\User;

class user\_application extends Model

{

    //

    use HasFactory;

    protected $table = 'user\_application';

    protected $fillable = [

        'applicant',

        'jobApplied',

        'appliedAt',

        'name',

        'experience',

        'contactNumber',

        'resumeLink',

        'expectations'

    ];

    public function user()

    {

        return $this->belongsTo(User::class, 'applicant');

    }

    // Relationship to Job model

    public function job()

    {

        return $this->belongsTo(table\_jobs\_uploading::class, 'jobApplied');

    }

}

**Schema:   
user schema;**   
<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration {

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

        Schema::create('user', function (Blueprint $table) {

            $table->id();

            $table->string('name');

            $table->string('email');

            $table->string('gender');

            $table->string('role');

            $table->string('password');

            $table->timestamps();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::dropIfExists('user');

    }

};

**job schema:**   
<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration {

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

        Schema::create('table\_jobs\_uploading', function (Blueprint $table) {

            $table->id();

            $table->string('jobTitle');

            $table->string('employmentType');

            $table->string('location');

            $table->string('salary');

            $table->text('description');

            $table->text('companyName');

            $table->text('requirements');

            $table->foreignId('creator')->constrained('user')->onDelete('cascade');

            $table->string('status')->default('open');

            $table->timestamps();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::dropIfExists('table\_jobs\_uploading');

    }

};

application schema;   
  
<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

return new class extends Migration {

    /\*\*

     \* Run the migrations.

     \*/

    public function up(): void

    {

        Schema::create('user\_application', function (Blueprint $table) {

            $table->id();

            $table->foreignId('applicant')->constrained('user')->onDelete('cascade');

            $table->foreignId('jobApplied')->constrained('table\_jobs\_uploading')->onDelete('cascade');

            $table->dateTime('appliedAt');

            $table->string('name');

            $table->string('experience');

            $table->string('contactNumber');

            $table->string('resumeLink');

            $table->text('expectations');

            $table->timestamps();

        });

    }

    /\*\*

     \* Reverse the migrations.

     \*/

    public function down(): void

    {

        Schema::dropIfExists('user\_application');

    }

};

**controllers:  
user controllers:**   
<?php

namespace App\Http\Controllers\UserControllers;

use Illuminate\Support\Facades\Validator;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use App\Models\User;

use Illuminate\Support\Facades\Log;

use Illuminate\Support\Facades\Hash;

class UserController extends Controller

{

    public function index()

    {

        $Users = User::all();

        if ($Users->count() > 0) {

            return response()->json([

                'status' => 200,

                'length' => $Users->count(),

                'users' => $Users

            ], 200);

        } else {

            return response()->json([

                'status' => 404,

                'users' => 'not found'

            ], 404);

        }

    }

    public function store(Request $request)

    {

        Log::info('Request Data:', $request->all());

        $validator = Validator::make($request->all(), [

            'name' => 'required|string|max:255',

            'email' => 'required|email|unique:user,email',

            'password' => 'required|string|min:6',

            'gender' => 'required|string',

            'role' => 'required|string'

        ]);

        if ($validator->fails()) {

            Log::error('Validation failed:', $validator->errors()->toArray());

            return response()->json([

                'errors' => $validator->errors()

            ], 400);

        }

        try {

            $user = User::create([

                'name' => $request->name,

                'email' => $request->email,

                'password' => bcrypt($request->password),  // Make sure to hash passwords!

                'gender' => $request->gender,

                'role' => $request->role,

            ]);

            // Generate random 9-digit number

            $randomNumber = mt\_rand(100000000, 999999999);

            Log::info('User created successfully', ['user' => $user, 'random\_number' => $randomNumber]);

            return response()->json([

                'message' => 'User created successfully!',

                'status' => 200,

                'data' => $user,

                'token' => $randomNumber

            ], 201);

        } catch (\Exception $e) {

            Log::error('Error creating user: ' . $e->getMessage());

            return response()->json([

                'message' => 'Error creating user',

                'error' => $e->getMessage()

            ], 500);

        }

    }

    // public function store(Request $request)

    // {

    //     Log::info('Request Data:', $request->all());

    //     $validator = Validator::make($request->all(), [

    //         'name' => 'required|string|max:255',

    //         'email' => 'required|email|unique:user,email',

    //         'password' => 'string',

    //         'gender' => 'string',

    //         'role' => 'string'

    //     ]);

    //     if ($validator->fails()) {

    //         Log::error('Validation failed:', $validator->errors()->toArray());

    //         return response()->json([

    //             'errors' => $validator->errors()

    //         ], 400);

    //     }

    //     try {

    //         $user = User::create([

    //             'name' => $request->name,

    //             'email' => $request->email,

    //             'password' => $request->password,

    //             'gender' => $request->gender,

    //             'role' => $request->role,

    //         ]);

    //         Log::info('User created successfully', ['user' => $user]);

    //         return response()->json([

    //             'message' => 'User created successfully!',

    //             "status" => 200,

    //             'data' => $user

    //         ], 201);

    //     } catch (\Exception $e) {

    //         Log::error('Error creating user: ' . $e->getMessage());

    //         return response()->json([

    //             'message' => 'Error creating user',

    //             'error' => $e->getMessage()

    //         ], 500);

    //     }

    // }

    // public function store(Request $request)

    // {

    //     Log::info('Request Data:', $request->all());

    //     $validator = Validator::make($request->all(), [

    //         'name' => 'required|string|max:255',

    //         'email' => 'required|email|unique:user,email',

    //         'password' => 'required|string|min:6',

    //         'gender' => 'required|string',

    //         'role' => 'required|string'

    //     ]);

    //     if ($validator->fails()) {

    //         Log::error('Validation failed:', $validator->errors()->toArray());

    //         return response()->json([

    //             'errors' => $validator->errors()

    //         ], 400);

    //     }

    //     try {

    //         $user = User::create([

    //             'name' => $request->name,

    //             'email' => $request->email,

    //             'password' => $request->password,

    //             'gender' => $request->gender,

    //             'role' => $request->role,

    //         ]);

    //         Log::info('User created successfully', ['user' => $user]);

    //         return response()->json([

    //             'message' => 'User created successfully!',

    //             'status' => 200,

    //             'data' => $user

    //         ], 201);

    //     } catch (\Exception $e) {

    //         Log::error('Error creating user: ' . $e->getMessage());

    //         return response()->json([

    //             'message' => 'Error creating user',

    //             'error' => $e->getMessage()

    //         ], 500);

    //     }

    // }

    public function update(Request $request, $id)

    {

        Log::info('Request Data for Update:', $request->all());

        $validator = Validator::make($request->all(), [

            'name' => 'nullable|string|max:255',

            'email' => 'nullable|email|unique:user,email,' . $id, // Ensure the email is unique but allow the current one to remain unchanged

            'password' => 'nullable|string|min:8', // Password should be string and has a min length of 8

            'gender' => 'nullable|string',

            'role' => 'nullable|string',

        ]);

        if ($validator->fails()) {

            Log::error('Validation failed:', $validator->errors()->toArray());

            return response()->json([

                'errors' => $validator->errors()

            ], 400);

        }

        try {

            $user = User::find($id);

            if (!$user) {

                // Return 404 if the user is not found

                return response()->json([

                    'message' => 'User not found',

                    'status' => 404

                ], 404);

            }

            // Update the user's data, only updating fields provided in the request

            $user->name = $request->name ?: $user->name;

            $user->email = $request->email ?: $user->email;

            $user->password = $request->password ? bcrypt($request->password) : $user->password;

            $user->gender = $request->gender ?: $user->gender;

            $user->role = $request->role ?: $user->role;

            // Save the updated user

            $user->save();

            Log::info('User updated successfully', ['user' => $user]);

            return response()->json([

                'message' => 'User updated successfully!',

                'status' => 200,

                'data' => $user

            ], 200);

        } catch (\Exception $e) {

            Log::error('Error updating user: ' . $e->getMessage());

            return response()->json([

                'message' => 'Error updating user',

                'error' => $e->getMessage()

            ], 500);

        }

    }

    public function destroy($id)

    {

        try {

            $user = User::find($id);

            if (!$user) {

                return response()->json([

                    'message' => 'User not found'

                ], 404);

            }

            $user->delete();

            Log::info('User deleted successfully', ['user\_id' => $id]);

            return response()->json([

                'message' => 'User deleted successfully',

                'status' => 200,

                'data' => null

            ], 200);

        } catch (\Exception $e) {

            // Log any errors

            Log::error('Error deleting user: ' . $e->getMessage());

            return response()->json([

                'message' => 'Error deleting user',

                'error' => $e->getMessage()

            ], 500);

        }

    }

    public function show($id)

    {

        try {

            $user = User::find($id);

            if (!$user) {

                return response()->json([

                    'message' => 'User not found'

                ], 404);

            }

            // Log the successful retrieval

            Log::info('User retrieved successfully', ['user\_id' => $id]);

            return response()->json([

                'message' => 'User retrieved successfully',

                'status' => 200,

                'data' => $user

            ], 200);

        } catch (\Exception $e) {

            // Log any errors

            Log::error('Error retrieving user: ' . $e->getMessage());

            return response()->json([

                'message' => 'Error retrieving user',

                'error' => $e->getMessage()

            ], 500);

        }

    }

    // public function login(Request $request)

    // {

    //     // Validate email and password inputs

    //     $credentials = $request->validate([

    //         'email' => 'required|email',

    //         'password' => 'required'

    //     ]);

    //     // Attempt to find the user

    //     $user = User::where('email', $credentials['email'])->first();

    //     $randomNumber = mt\_rand(100000000, 999999999);

    //     // If user not found or password mismatch, return error

    //     if (

    //         !$user

    //         //  || !Hash::check($credentials['password'], $user->password)

    //     ) {

    //         return response()->json([

    //             'message' => 'Invalid email or password.'

    //         ], 401);

    //     }

    //     // On success, return a JSON success message (you could also create a token here)

    //     return response()->json([

    //         'message' => 'Login successful.',

    //         'data' => $user,

    //         'token' => $randomNumber

    //     ], 200);

    // }

    public function login(Request $request)

    {

        // Validate email and password inputs

        $credentials = $request->validate([

            'email' => 'required|email',

            'password' => 'required'

        ]);

        // Attempt to find the user

        $user = User::where('email', $credentials['email'])->first();

        // If user not found or password mismatch, return error

        if (!$user || !Hash::check($credentials['password'], $user->password)) {

            return response()->json([

                'message' => 'Invalid email or password.'

            ], 401);

        }

        $randomNumber = mt\_rand(100000000, 999999999);

        // On success, return a JSON success message (you could also create a token here)

        return response()->json([

            'message' => 'Login successful.',

            'data' => $user,

            'token' => $randomNumber

        ], 200);

    }

    public function logout()

    {

        return response()->json([

            'message' => 'Logged out successfully.'

        ], 200);

    }

}

**job controllers:**   
<?php

namespace App\Http\Controllers\JobControllers;

use App\Http\Controllers\Controller;

use App\Models\table\_jobs\_uploading;

use Illuminate\Support\Facades\Log;

use Illuminate\Http\Request;

use App\Models\User;

// use App\Models\user\_application;

use App\Models\user\_application;

class JobController extends Controller

{

    public function index()

    {

        try {

            $jobs = table\_jobs\_uploading::all();

            if ($jobs->count() > 0) {

                return response()->json([

                    'status' => 200,

                    'length' => $jobs->count(),

                    'jobs' => $jobs

                ], 200);

            } else {

                return response()->json([

                    'status' => 404,

                    'jobs' => 'No jobs found'

                ], 404);

            }

        } catch (\Exception $e) {

            // Log the exact error for debugging

            \Log::error('Error fetching jobs: ' . $e->getMessage());

            // Return the error message in the response

            return response()->json([

                'status' => 500,

                'error' => $e->getMessage(),

            ], 500);

        }

    }

    public function getJobsByCreator(Request $request)

    {

        $email = $request->query('email');

        $status = $request->query('status'); // optional

        // Email is required

        if (!$email) {

            return response()->json([

                'status' => 400,

                'message' => 'Email is required as a query parameter.',

            ], 400);

        }

        // Find user by email

        $user = User::where('email', $email)->first();

        if (!$user) {

            return response()->json([

                'status' => 404,

                'message' => 'User not found.',

            ], 404);

        }

        // Build query

        $query = table\_jobs\_uploading::where('creator', $user->id);

        // If status is provided, filter by it

        if ($status) {

            $query->where('status', $status);

        }

        $jobs = $query->get();

        return response()->json([

            'status' => 200,

            'message' => 'Jobs retrieved successfully.',

            'jobs' => $jobs,

        ]);

    }

    // public function getJobsByCreator(Request $request)

    // {

    //     $email = $request->query('email');

    //     $status = $request->query('status');

    //     if (!$email || !$status) {

    //         return response()->json([

    //             'status' => 400,

    //             'message' => 'Email and status are required as query parameters.',

    //         ], 400);

    //     }

    //     $user = User::where('email', $email)->first();

    //     if (!$user) {

    //         return response()->json([

    //             'status' => 404,

    //             'message' => 'User not found.',

    //         ], 404);

    //     }

    //     $jobs = table\_jobs\_uploading::where('creator', $user->id)

    //         ->where('status', $status)

    //         ->get();

    //     return response()->json([

    //         'status' => 200,

    //         'message' => 'Jobs retrieved successfully.',

    //         'jobs' => $jobs,

    //     ]);

    // }

    public function store(Request $request)

    {

        try {

            $request->validate([

                // 'email' => 'required|email|exists:user,email',

                // 'jobTitle' => $request->jobTitle,

                // 'employmentType' => $request->employmentType,

                // 'location' => $request->location,

                // 'salary' => $request->salary,

                // 'description' => $request->description,

                // 'companyName' => $request->companyName,

                // 'requirements' => $request->requirements,

                // 'status' => $request->status,

                'email' => 'required|email|exists:user,email',

                'jobTitle' => 'required|string',

                'employmentType' => 'required|string',

                'location' => 'required|string',

                'salary' => 'required|string',

                'description' => 'required|string',

                'companyName' => 'required|string',

                'requirements' => 'required|string',

                'status' => 'required|string|in:open,closed'

            ]);

            $user = User::where('email', $request->email)->first();

            if (!$user || $user->role !== 'recruiter') {

                return response()->json([

                    'status' => 403,

                    'message' => 'Only recruiters can create jobs.',

                ], 403);

            }

            $job = table\_jobs\_uploading::create([

                'jobTitle' => $request->jobTitle,

                'employmentType' => $request->employmentType,

                'location' => $request->location,

                'salary' => $request->salary,

                'description' => $request->description,

                'creator' => $user->id,

                'status' => $request->status,

                'companyName' => $request->companyName,

                'requirements' => $request->requirements,

            ]);

            return response()->json([

                'status' => 200,

                'message' => 'Job created successfully',

                'data' => $job,

            ], 200);

        } catch (\Exception $e) {

            return response()->json([

                'status' => 500,

                'error' => $e->getMessage(),

            ], 500);

        }

    }

    // public function updateJob(Request $request, $jobId)

    // {

    //     $request->validate([

    //         'creator\_id' => 'required|integer|exists:user,id',

    //         'jobTitle' => 'nullable|string',

    //         'employmentType' => 'nullable|string',

    //         'location' => 'nullable|string',

    //         'salary' => 'nullable|string',

    //         'description' => 'nullable|string',

    //         'status' => 'nullable|string',

    //     ]);

    //     $user = User::find($request->creator\_id);

    //     if (!$user || $user->role !== 'recruiter') {

    //         return response()->json([

    //             'status' => 403,

    //             'message' => 'Only recruiters can update jobs.',

    //         ], 403);

    //     }

    //     $job = table\_jobs\_uploading::find($jobId);

    //     if (!$job) {

    //         return response()->json([

    //             'status' => 404,

    //             'message' => 'Job not found.',

    //         ], 404);

    //     }

    //     if ($job->creator !== $user->id) {

    //         return response()->json([

    //             'status' => 403,

    //             'message' => 'You are not the creator of this job.',

    //         ], 403);

    //     }

    //     $job->update($request->only([

    //         'jobTitle',

    //         'employmentType',

    //         'location',

    //         'salary',

    //         'description',

    //         'status',

    //     ]));

    //     return response()->json([

    //         'status' => 200,

    //         'message' => 'Job updated successfully.',

    //         'data' => $job,

    //     ], 200);

    // }

    public function allJobs(Request $request)

    {

        try {

            $status = $request->query('status');

            $query = table\_jobs\_uploading::query();

            if ($status && strtolower($status) !== 'all') {

                $query->where('status', $status);

            }

            $jobs = $query->get();

            return response()->json([

                'status' => 200,

                'length' => $jobs->count(),

                'jobs' => $jobs,

            ], 200);

        } catch (\Exception $e) {

            return response()->json([

                'status' => 500,

                'error' => $e->getMessage(),

            ], 500);

        }

    }

    public function updateJob(Request $request, $jobId)

    {

        $request->validate([

            'creator\_id' => 'required|integer|exists:user,id',

            'jobTitle' => 'nullable|string',

            'employmentType' => 'nullable|string',

            'location' => 'nullable|string',

            'salary' => 'nullable|string',

            'description' => 'nullable|string',

            'companyName' => 'nullable|string',

            'requirements' => 'nullable|string',

            'status' => 'nullable|string',

        ]);

        $user = User::find($request->creator\_id);

        if (!$user || $user->role !== 'recruiter') {

            return response()->json([

                'status' => 403,

                'message' => 'Only recruiters can update jobs.',

            ], 403);

        }

        $job = table\_jobs\_uploading::find($jobId);

        if (!$job) {

            return response()->json([

                'status' => 404,

                'message' => 'Job not found.',

            ], 404);

        }

        if ($job->creator !== $user->id) {

            return response()->json([

                'status' => 403,

                'message' => 'You are not the creator of this job.',

            ], 403);

        }

        $job->update($request->only([

            'jobTitle',

            'employmentType',

            'location',

            'salary',

            'description',

            'companyName',

            'requirements',

            'status',

        ]));

        return response()->json([

            'status' => 200,

            'message' => 'Job updated successfully.',

            'data' => $job,

        ], 200);

    }

    // public function applyForJob(Request $request)

    // {

    //     try {

    //         // Validate incoming request data

    //         $request->validate([

    //             'job\_id' => 'required|exists:postejob,id', // Validate job exists

    //             'email' => 'required|email|exists:user,email', // Validate email exists

    //         ]);

    //         // Fetch user by email

    //         $user = User::where('email', $request->email)->first();

    //         // Check if user is not found

    //         if (!$user) {

    //             return response()->json([

    //                 'status' => 404,

    //                 'message' => 'User not found.',

    //             ], 404);

    //         }

    //         // Fetch job by job\_id

    //         $job = table\_jobs\_uploading::find($request->job\_id);

    //         // Check if job is not found

    //         if (!$job) {

    //             return response()->json([

    //                 'status' => 404,

    //                 'message' => 'Job not found.',

    //             ], 404);

    //         }

    //         // Check if user has already applied for this job

    //         $existingApplication = user\_application::where('applicant', $user->id)

    //             ->where('jobApplied', $job->id)

    //             ->first();

    //         if ($existingApplication) {

    //             return response()->json([

    //                 'status' => 400,

    //                 'message' => 'You have already applied for this job.',

    //             ], 400);

    //         }

    //         // Create a new application entry

    //         $application = user\_application::create([

    //             'applicant' => $user->id,

    //             'jobApplied' => $job->id,

    //             'appliedAt' => now(), // or Carbon::now()

    //             'status' => 'pending',

    //         ]);

    //         return response()->json([

    //             'status' => 200,

    //             'message' => 'You have successfully applied for the job.',

    //             'application' => $application,

    //         ], 200);

    //     } catch (\Illuminate\Validation\ValidationException $e) {

    //         // Handle validation errors

    //         return response()->json([

    //             'status' => 422,

    //             'message' => 'Validation failed.',

    //             'errors' => $e->errors(),

    //         ], 422);

    //     } catch (\Exception $e) {

    //         // Catch any other exceptions

    //         return response()->json([

    //             'status' => 500,

    //             'message' => 'An error occurred.',

    //             'error' => $e->getMessage(),

    //         ], 500);

    //     }

    // }

    // Update with your actual Job model class

    // public function deleteJob(Request $request)

    // {

    //     try {

    //         // Validate input

    //         $request->validate([

    //             'job\_id' => 'required|exists:,id',

    //             'email' => 'required|email|exists:user,email',

    //         ]);

    //         // Find user by email

    //         $user = User::where('email', $request->email)->first();

    //         // Check if user is a recruiter

    //         if ($user->role !== 'recruiter') {

    //             return response()->json([

    //                 'status' => 403,

    //                 'message' => 'Only recruiters are allowed to delete jobs.',

    //             ], 403);

    //         }

    //         // Find the job

    //         $job = table\_jobs\_uploading::find($request->job\_id);

    //         // Check if the job was created by this user

    //         if ($job->creator != $user->id) {

    //             return response()->json([

    //                 'status' => 403,

    //                 'message' => 'You are not the creator of this job.',

    //             ], 403);

    //         }

    //         // Delete the job

    //         $job->delete();

    //         return response()->json([

    //             'status' => 200,

    //             'message' => 'Job deleted successfully.',

    //         ]);

    //     } catch (\Illuminate\Validation\ValidationException $e) {

    //         return response()->json([

    //             'status' => 422,

    //             'message' => 'Validation failed.',

    //             'errors' => $e->errors(),

    //         ], 422);

    //     } catch (\Exception $e) {

    //         return response()->json([

    //             'status' => 500,

    //             'message' => 'An error occurred.',

    //             'error' => $e->getMessage(),

    //         ], 500);

    //     }

    // }

    public function applyForJob(Request $request)

    {

        // Log all incoming request data

        \Log::info('Job Application Request Data:', $request->all());

        try {

            // Fetch user by email

            $user = User::where('email', $request->email)->first();

            if (!$user) {

                return response()->json([

                    'status' => 404,

                    'message' => 'User not found with provided email.',

                ], 404);

            }

            // Fetch job by job\_id

            $job = table\_jobs\_uploading::find($request->job\_id);

            if (!$job) {

                return response()->json([

                    'status' => 404,

                    'message' => 'Job not found with provided job\_id.',

                ], 404);

            }

            // Check if user has already applied for this job

            $existingApplication = user\_application::where('applicant', $user->id)

                ->where('jobApplied', $job->id)

                ->first();

            if ($existingApplication) {

                return response()->json([

                    'status' => 400,

                    'message' => 'You have already applied for this job.',

                ], 400);

            }

            // Create a new application entry

            $application = user\_application::create([

                'applicant' => $user->id,

                'jobApplied' => $job->id,

                'appliedAt' => now(),

                'name' => $request->name,

                'experience' => $request->experience,

                'contactNumber' => $request->contactNumber,

                'resumeLink' => $request->resumeLink,

                'expectations' => $request->expectations,

            ]);

            return response()->json([

                'status' => 200,

                'message' => 'You have successfully applied for the job.',

                'application' => $application,

            ], 200);

        } catch (\Exception $e) {

            // Log the error too, for debugging

            \Log::error('Error in applyForJob:', ['error' => $e->getMessage()]);

            return response()->json([

                'status' => 500,

                'message' => 'An unexpected error occurred.',

                'error' => $e->getMessage(),

            ], 500);

        }

    }

    public function deleteJob(Request $request)

    {

        try {

            Log::info('Delete Job Request:', $request->all());

            // Validate input: job\_id must exist in jobs, email must exist in users table

            $request->validate([

                'job\_id' => 'required|exists:table\_jobs\_uploading,id',

                'email' => 'required|email|exists:user,email',

            ]);

            // Find the job

            $job = table\_jobs\_uploading::find($request->job\_id);

            if (!$job) {

                return response()->json([

                    'status' => 404,

                    'message' => 'Job not found.',

                ], 404);

            }

            // Find user by email

            $user = User::where('email', $request->email)->first();

            if (!$user) {

                return response()->json([

                    'status' => 404,

                    'message' => 'User not found.',

                ], 404);

            }

            // Check if this user is the creator of the job

            if ($job->creator !== $user->id) {

                return response()->json([

                    'status' => 403,

                    'message' => 'You are not authorized to delete this job.',

                ], 403);

            }

            // Delete the job

            $job->delete();

            return response()->json([

                'status' => 200,

                'message' => 'Job deleted successfully.',

            ]);

        } catch (\Illuminate\Validation\ValidationException $e) {

            Log::error('Validation failed:', $e->errors());

            return response()->json([

                'status' => 422,

                'message' => 'Validation failed.',

                'errors' => $e->errors(),

            ], 422);

        } catch (\Exception $e) {

            Log::error('Delete job failed:', ['error' => $e->getMessage()]);

            return response()->json([

                'status' => 500,

                'message' => 'An error occurred while deleting the job.',

                'error' => $e->getMessage(), // only for debugging

            ], 500);

        }

    }

    public function getAllApplications()

    {

        try {

            // Fetch all job applications from the database

            $applications = user\_application::all();

            return response()->json([

                'status' => 200,

                'message' => 'Applications retrieved successfully.',

                'length' => $applications->count(),

                'applications' => $applications

            ], 200);

        } catch (\Throwable $e) {

            // Log the full error in Laravel logs (storage/logs/laravel.log)

            \Log::error('Error fetching applications', [

                'message' => $e->getMessage(),

                'file' => $e->getFile(),

                'line' => $e->getLine(),

                'trace' => $e->getTraceAsString()

            ]);

            // Return error in response body (visible in Postman)

            return response()->json([

                'status' => 500,

                'message' => 'An error occurred while fetching applications.',

                'error' => $e->getMessage(),

                'file' => $e->getFile(),

                'line' => $e->getLine()

            ], 500);

        }

    }

    // public function getApplicationsByEmail(Request $request)

    // {

    //     try {

    //         // Validate the email input

    //         $request->validate([

    //             'email' => 'required|email|exists:user,email', // Ensure email exists in the user table

    //         ]);

    //         // Fetch the user by email

    //         $user = User::where('email', $request->email)->first();

    //         // Start the query to fetch applications for this user

    //         $query = user\_application::where('applicant', $user->id);

    //         // Check if 'status' is provided in the query parameters

    //         if ($request->has('status') && $request->status) {

    //             // If a status is provided, filter applications by that status

    //             $query->where('status', $request->status);

    //         }

    //         // Fetch the applications based on the query

    //         $applications = $query->get();

    //         // Return the response

    //         return response()->json([

    //             'status' => 200,

    //             'message' => 'Applications retrieved successfully.',

    //             'applications' => $applications

    //         ], 200);

    //     } catch (\Throwable $e) {

    //         // Log the full error in Laravel logs (storage/logs/laravel.log)

    //         \Log::error('Error fetching applications by email', [

    //             'message' => $e->getMessage(),

    //             'file' => $e->getFile(),

    //             'line' => $e->getLine(),

    //             'trace' => $e->getTraceAsString()

    //         ]);

    //         // Return error in response body (visible in Postman)

    //         return response()->json([

    //             'status' => 500,

    //             'message' => 'An error occurred while fetching applications.',

    //             'error' => $e->getMessage(),

    //             'file' => $e->getFile(),

    //             'line' => $e->getLine()

    //

    // public function getApplicationsByEmail(Request $request)

    // {

    //     try {

    //         $request->validate([

    //             'email' => 'required|email|exists:user,email',

    //         ]);

    //         $user = User::where('email', $request->email)->first();

    //         $query = user\_application::where('applicant', $user->id);

    //         if ($request->has('status') && $request->status) {

    //             $query->where('status', $request->status);

    //         }

    //         // Eager load related job and creator

    //         $applications = $query->with(['job.creator', 'applicant'])->get();

    //         $formatted = $applications->map(function ($app) {

    //             return [

    //                 'id' => $app->id,

    //                 'job\_title' => $app->job->jobTitle ?? 'N/A',

    //                 'job\_creator' => $app->job->creator->name ?? 'N/A',

    //                 'applicant\_name' => $app->applicant->name ?? 'N/A',

    //                 'appliedAt' => $app->appliedAt,

    //                 'status' => $app->status,

    //                 'created\_at' => $app->created\_at,

    //                 'updated\_at' => $app->updated\_at,

    //             ];

    //         });

    //         return response()->json([

    //             'status' => 200,

    //             'message' => 'Applications retrieved successfully.',

    //             'applications' => $formatted,

    //         ]);

    //     } catch (\Throwable $e) {

    //         \Log::error('Error fetching applications by email', [

    //             'message' => $e->getMessage(),

    //             'file' => $e->getFile(),

    //             'line' => $e->getLine(),

    //         ]);

    //         return response()->json([

    //             'status' => 500,

    //             'message' => 'An error occurred while fetching applications.',

    //             'error' => $e->getMessage(),

    //             'file' => $e->getFile(),

    //             'line' => $e->getLine(),

    //         ]);

    //     }

    // }

    public function getApplicationsByEmail(Request $request)

    {

        try {

            $request->validate([

                'email' => 'required|email|exists:user,email',

            ]);

            $user = User::where('email', $request->email)->first();

            $query = user\_application::where('applicant', $user->id);

            if ($request->has('status') && $request->status) {

                $query->where('status', $request->status);

            }

            // Eager load related job and job.creator and user (applicant)

            $applications = $query->with(['job.creator', 'user'])->get();

            $formatted = $applications->map(function ($app) {

                return [

                    'id' => $app->id,

                    'job\_title' => $app->job->jobTitle ?? 'N/A',

                    'job\_creator' => $app->job->creator->name ?? 'N/A',

                    'applicant\_name' => $app->user->name ?? 'N/A',  // <-- use 'user' here

                    'appliedAt' => $app->appliedAt,

                    'status' => $app->status ?? 'N/A',

                    'created\_at' => $app->created\_at,

                    'updated\_at' => $app->updated\_at,

                ];

            });

            return response()->json([

                'status' => 200,

                'message' => 'Applications retrieved successfully.',

                'applications' => $formatted,

            ]);

        } catch (\Throwable $e) {

            \Log::error('Error fetching applications by email', [

                'message' => $e->getMessage(),

                'file' => $e->getFile(),

                'line' => $e->getLine(),

            ]);

            return response()->json([

                'status' => 500,

                'message' => 'An error occurred while fetching applications.',

                'error' => $e->getMessage(),

                'file' => $e->getFile(),

                'line' => $e->getLine(),

            ]);

        }

    }

}

**Routes:**   
  
<?php

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\UserControllers\UserController;

use App\Http\Controllers\JobControllers\JobController;

use App\Http\Controllers\JobControllers\applicationController;

Route::middleware('auth:sanctum')->get('/user', function (Request $request) {

    return $request->user();

});

// user-routes

Route::get('all-users', [UserController::class, 'index']);

Route::post('create-user', [UserController::class, 'store']);

Route::put('update-user/{id}', [UserController::class, 'update']);

Route::delete('delete-user/{id}', [UserController::class, 'destroy']);

Route::get('/one-user/{id}', [UserController::class, 'show']);

Route::post('login', [UserController::class, 'login']);

Route::get('/logout', [UserController::class, 'logout']);

//job-routes

Route::get('all-jobs', [JobController::class, 'allJobs']);

Route::post('create-job', [JobController::class, 'store']);

Route::patch('update-job/{jobId}', [JobController::class, 'updateJob']);

Route::post('apply-job', [JobController::class, 'applyForJob']);

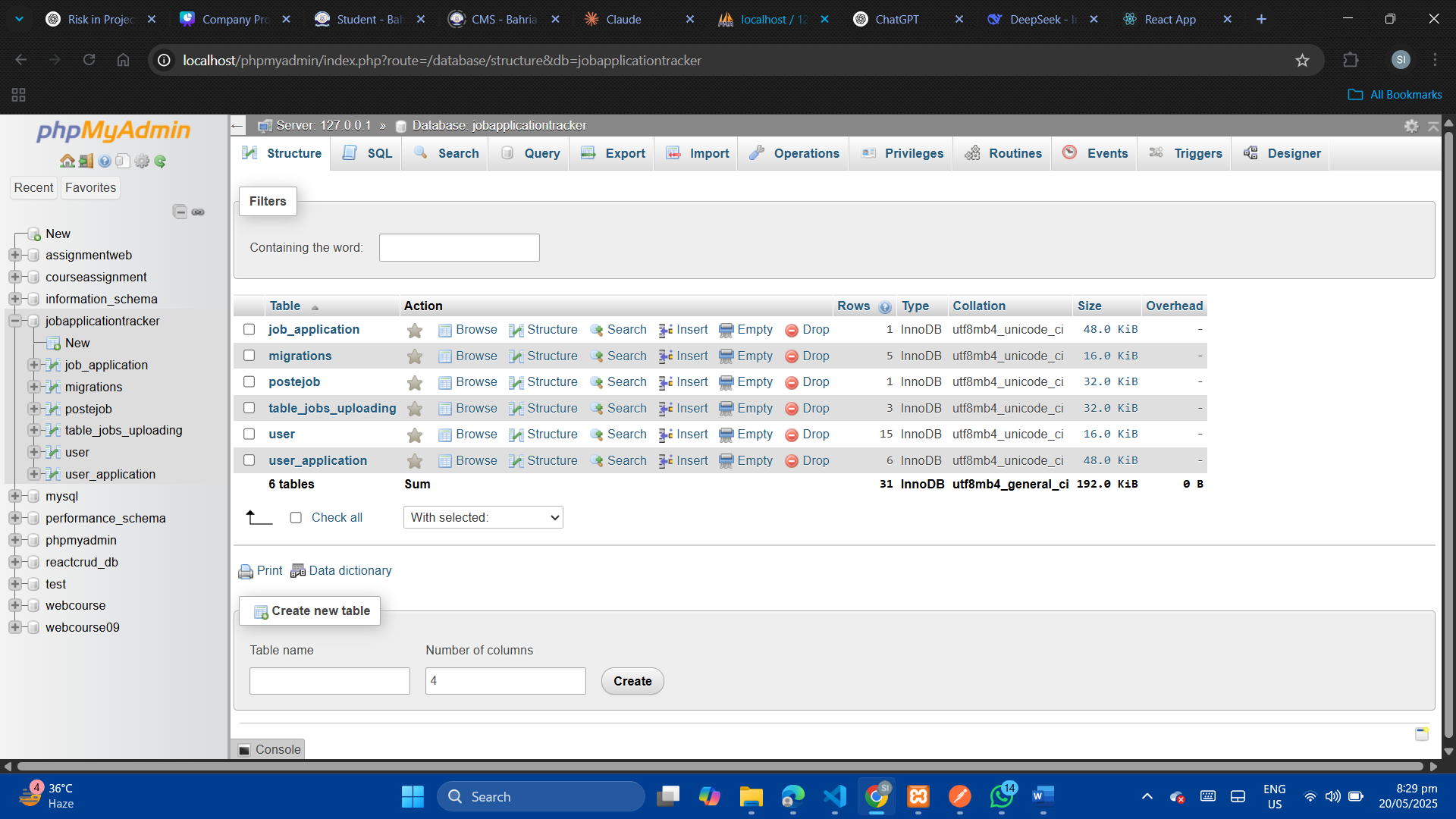
Route::delete('delete-job', [JobController::class, 'deleteJob']);

Route::get('/jobs-by-creator', [JobController::class, 'getJobsByCreator']);

//application-routes

Route::get('all-applications', [JobController::class, 'getAllApplications']);

Route::post('applications-by-email', [JobController::class, 'getApplicationsByEmail']);

**Database:**   
  


postman:   
