**JOB PORTAL MANAGEMENT SYSTEM**

**Course Code & Name:** CAUC514\_FSWD

**Division:** MCA Semester II

**Student IDs:**

* **24MCA001 Nitin Ambani**
* **24MCA056 Bhavi Patel**
* **24MCA081 Amit Prajapati**

**Case Study Topic:**  Job Portal Management System

**1. INTRODUCTION**

The Job Portal Management System is a web-based application designed to connect job seekers with recruiters. It provides a platform for job seekers to search and apply for jobs, and for recruiters to post job openings and manage applications. The system aims to streamline the job search and recruitment process, making it more efficient and user-friendly.

In today's competitive job market, both job seekers and recruiters face numerous challenges. Job seekers struggle to find relevant opportunities that match their skills and experience, while recruiters find it difficult to identify qualified candidates from a large pool of applicants. Our Job Portal Management System addresses these challenges by providing a centralized platform with advanced search capabilities, application tracking, and user-friendly interfaces.

**2. PROJECT OVERVIEW**

The Job Portal Management System is designed to address the challenges faced by both job seekers and recruiters in the job market. For job seekers, finding suitable job opportunities can be time-consuming and frustrating. For recruiters, identifying qualified candidates from a large pool of applicants can be challenging.

Our system provides a centralized platform where:

* Job seekers can create profiles, search for jobs, and apply for positions
* Recruiters can post job openings, review applications, and manage the hiring process
* Both parties can communicate and track the status of applications

The project implements a complete end-to-end solution with the following components:

* User authentication and authorization system
* Job posting and management system
* Job search and filtering capabilities
* Application submission and tracking
* User profile management
* Responsive design for various devices

**3. TECHNOLOGY STACK**

The Job Portal Management System is built using the MEAN stack:

* **MongoDB:** A NoSQL database used to store user profiles, job listings, and application data
* **Express.js:** A web application framework for Node.js used to build the RESTful API
* **HTML/CSS/JavaScript:** Used for the frontend user interface
* **Node.js:** A JavaScript runtime environment used for the backend server

Additional technologies and tools:

* **JWT (JSON Web Tokens):** For secure authentication
* **Mongoose:** An ODM (Object Data Modeling) library for MongoDB and Node.js
* **RESTful API:** For communication between frontend and backend
* **bcrypt:** For password hashing and security
* **Bootstrap:** For responsive design and UI components
* **Fetch API:** For making HTTP requests from the frontend

**4. SYSTEM ARCHITECTURE**

The Job Portal Management System follows a client-server architecture:

**1. Client-side (Frontend):**

* HTML/CSS/JavaScript for the user interface
* Responsive design for compatibility with various devices
* Form validation for user inputs
* Fetch API for communication with the backend

**2. Server-side (Backend):**

* Node.js with Express.js for handling HTTP requests
* RESTful API endpoints for CRUD operations
* JWT authentication for secure access
* Middleware for request validation and error handling

**3. Database:**

* MongoDB for storing all application data
* Mongoose for data modeling and validation
* Indexes for efficient queries

**5. DATABASE DESIGN**

The database consists of the following main collections:

**1. Users Collection:**

Stores user information (job seekers and recruiters)

Fields:

* \_id: ObjectId (unique identifier)
* name: String (user's full name)
* email: String (unique email address)
* password: String (hashed password)
* role: String (jobseeker or recruiter)
* skills: Array of Strings (for job seekers)
* experience: Number (years of experience for job seekers)
* company: String (company name for recruiters)
* bio: String (user biography)
* createdAt: Date (account creation timestamp)

**2. Jobs Collection:**

Stores job listings posted by recruiters

Fields:

* \_id: ObjectId (unique identifier)
* title: String (job title)
* description: String (detailed job description)
* company: String (company name)
* location: String (job location)
* jobType: String (full-time, part-time, contract, etc.)
* skills: Array of Strings (required skills)
* salary: String (salary information)
* postedBy: ObjectId (reference to user who posted the job)
* createdAt: Date (posting timestamp)
* updatedAt: Date (last update timestamp)

**3. Applications Collection:**

Stores job applications submitted by job seekers

Fields:

* \_id: ObjectId (unique identifier)
* job: ObjectId (reference to the job)
* applicant: ObjectId (reference to the applicant)
* status: String (pending, reviewed, rejected, shortlisted, accepted)
* coverLetter: String (applicant's cover letter)
* resume: String (link to resume file)
* notes: String (recruiter's notes)
* createdAt: Date (application timestamp)
* updatedAt: Date (last update timestamp)

**6. KEY FEATURES**

**1. User Authentication and Authorization:**

* User registration and login
* Role-based access control (job seeker, recruiter)
* Secure authentication using JWT
* Password hashing for security
* Session management

**2. Job Seeker Features:**

* Create and update profile
* Search for jobs by title, location, or job type
* Apply for jobs with cover letter and resume
* View application status
* Track application history
* Dashboard with recent applications and job recommendations

**3. Recruiter Features:**

* Create and manage company profile
* Post, edit, and delete job listings
* View and manage applications
* Update application status (pending, reviewed, shortlisted, rejected, accepted)
* Add notes to applications
* Dashboard with job statistics and recent applications

**7. IMPLEMENTATION DETAILS**

The implementation of the Job Portal Management System involved several key components:

**1. Backend Development:**

* Setting up Node.js and Express.js server
* Creating RESTful API endpoints for users, jobs, and applications
* Implementing JWT authentication
* Connecting to MongoDB using Mongoose
* Implementing data validation and error handling

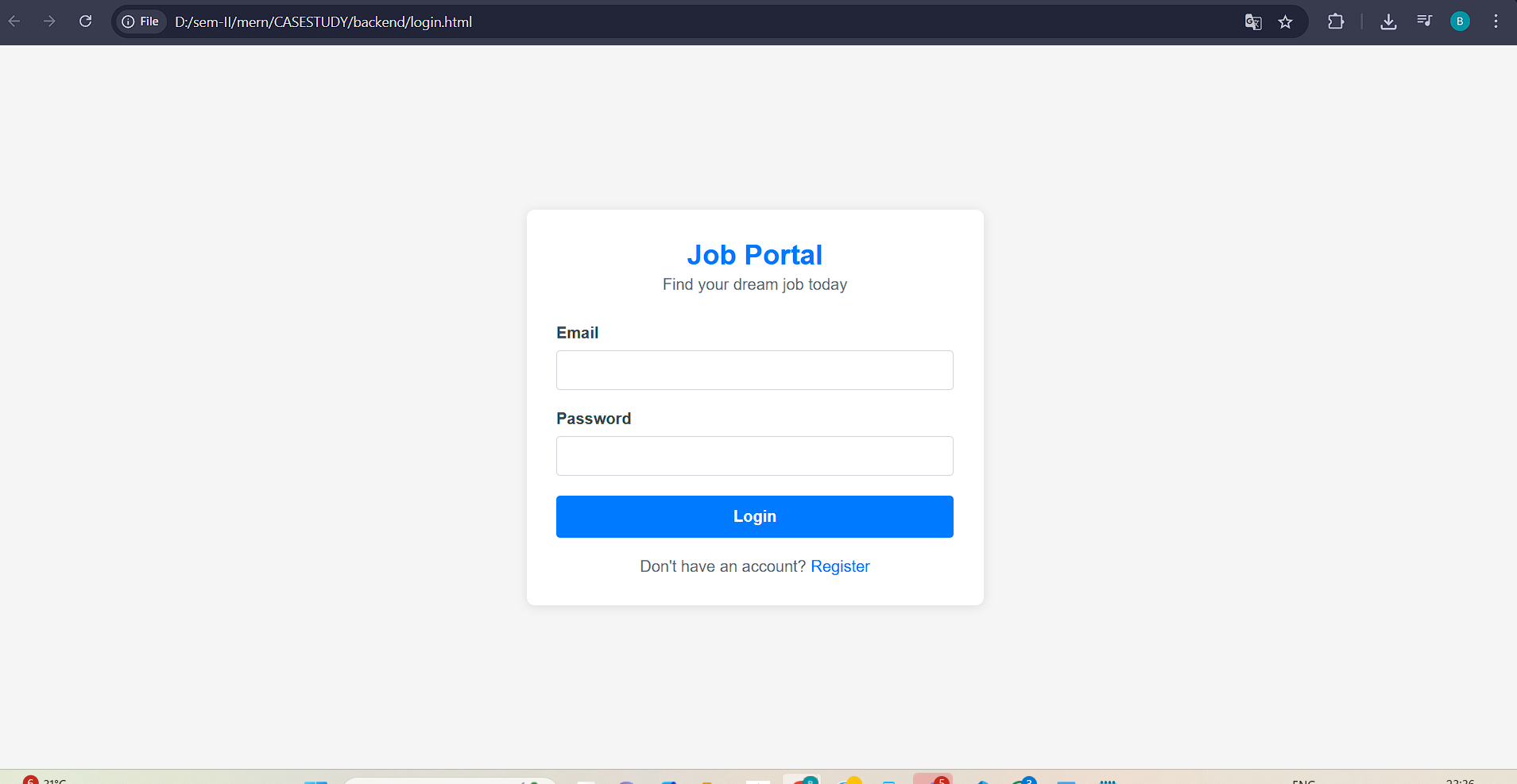
**2. Frontend Development:**

* Creating responsive HTML/CSS layouts
* Implementing client-side form validation
* Developing interactive UI components
* Integrating with backend API using fetch

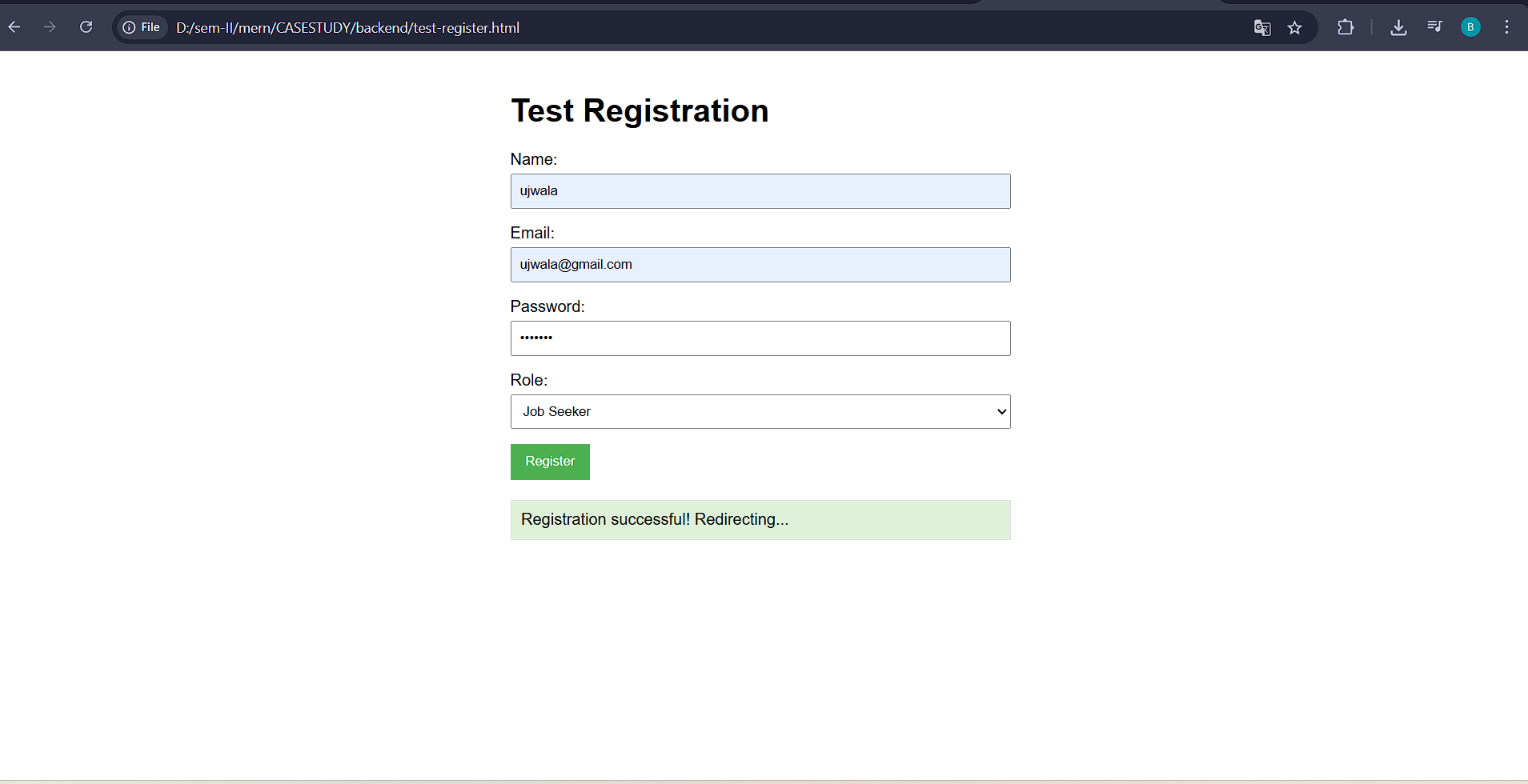
**8. SCREEN LAYOUTS**

The Job Portal Management System includes the following key screens:

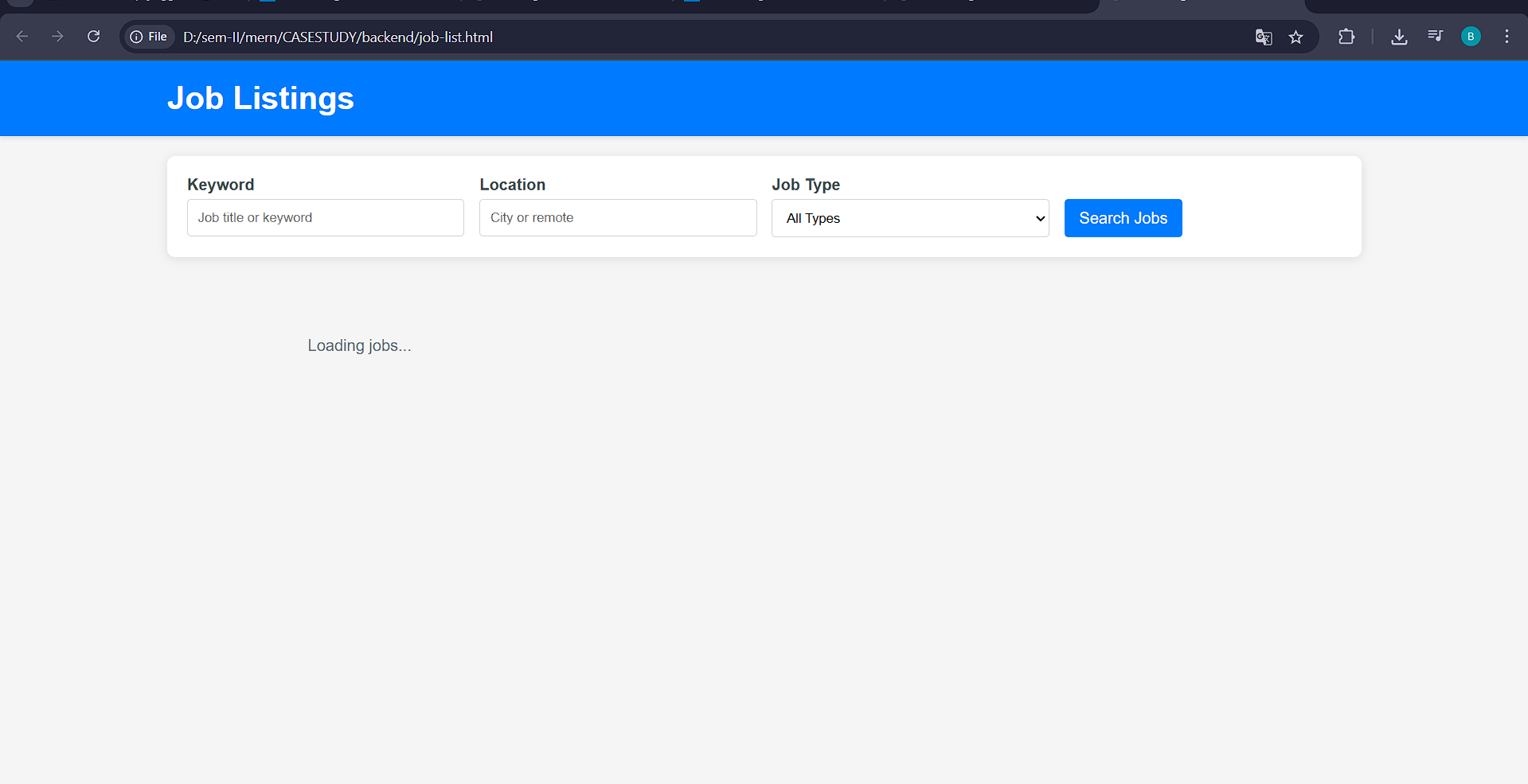
**1. Login Page**

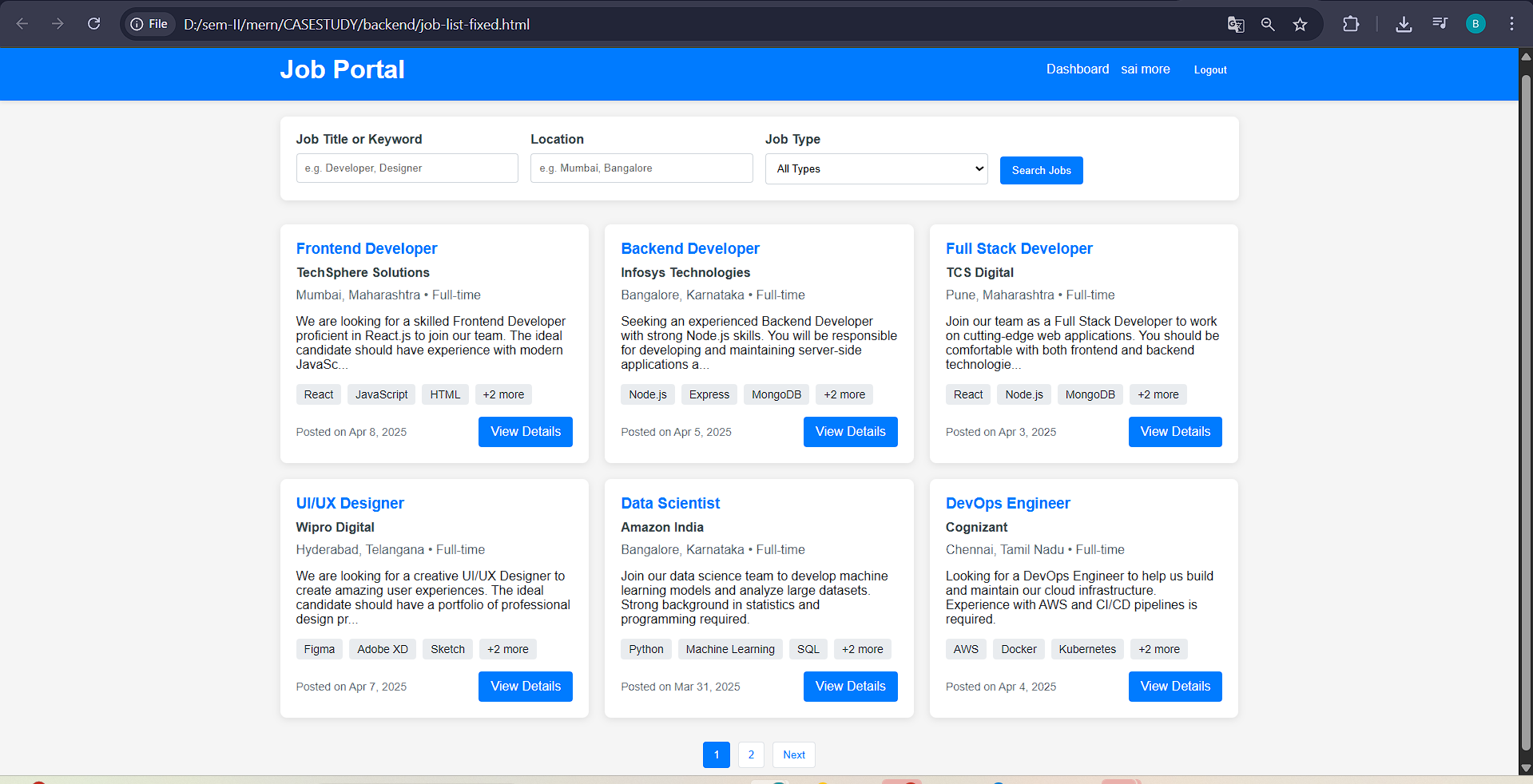
****

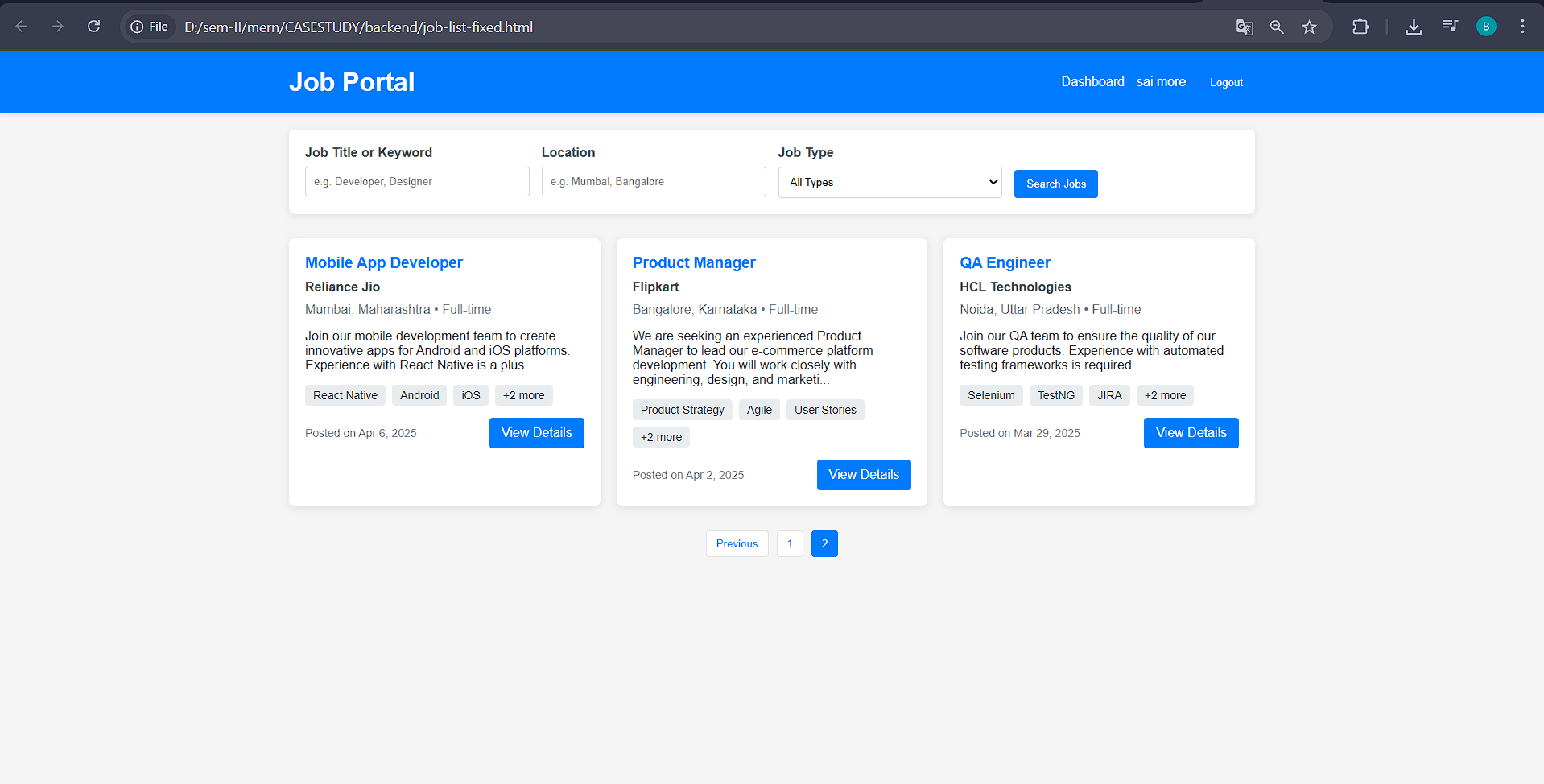
**2. Registration Page**



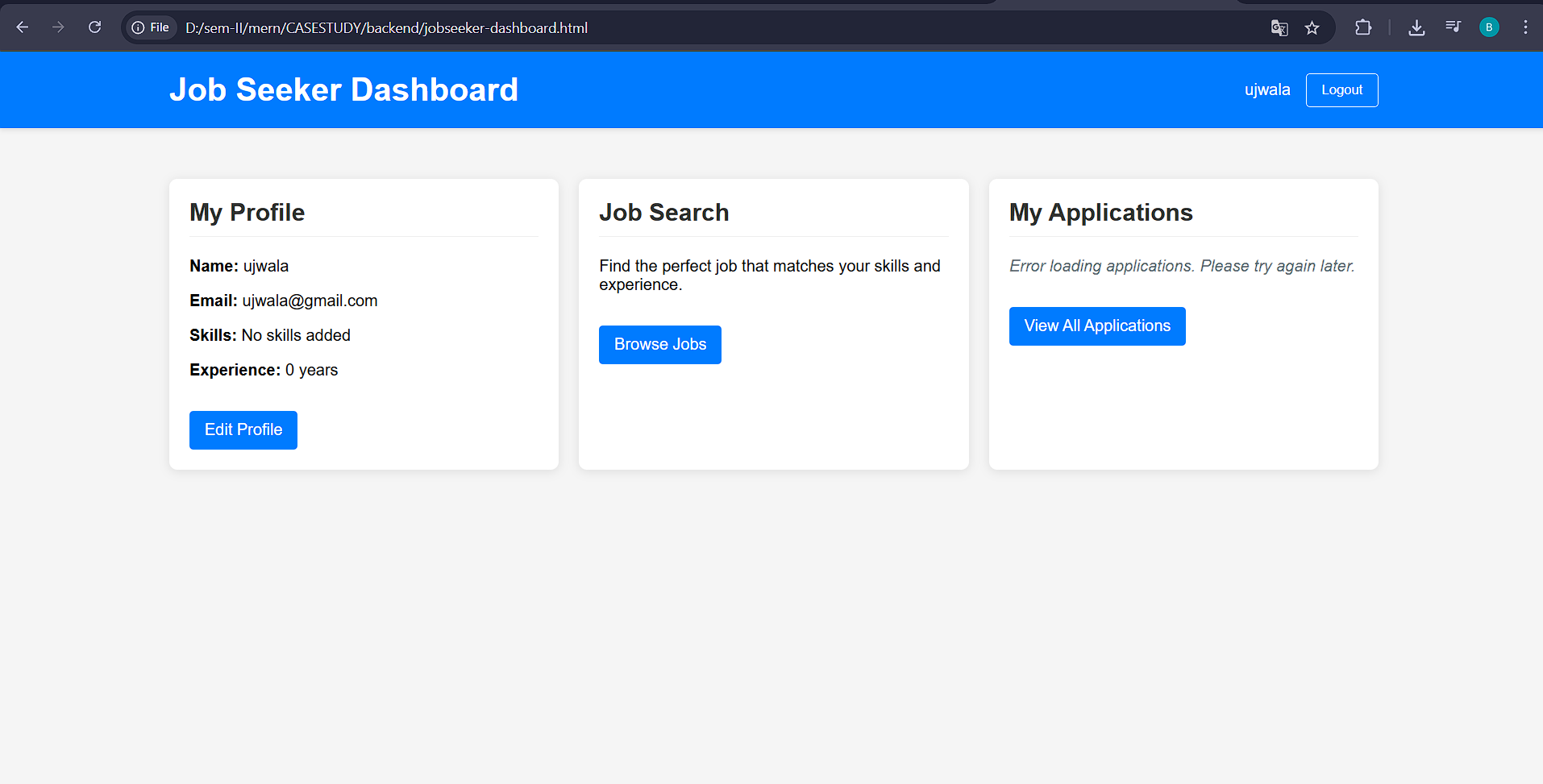
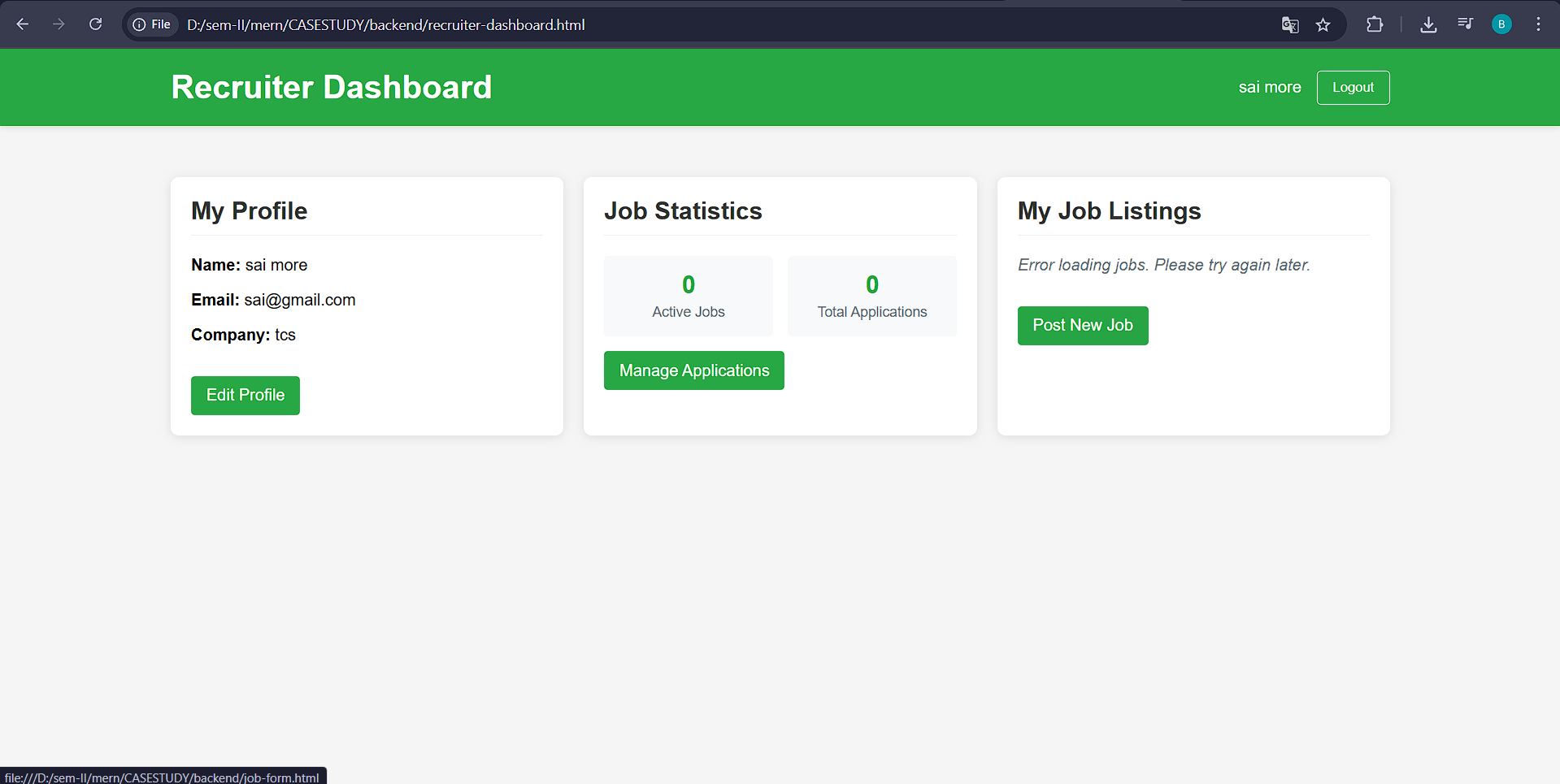
**3. Job List Page**

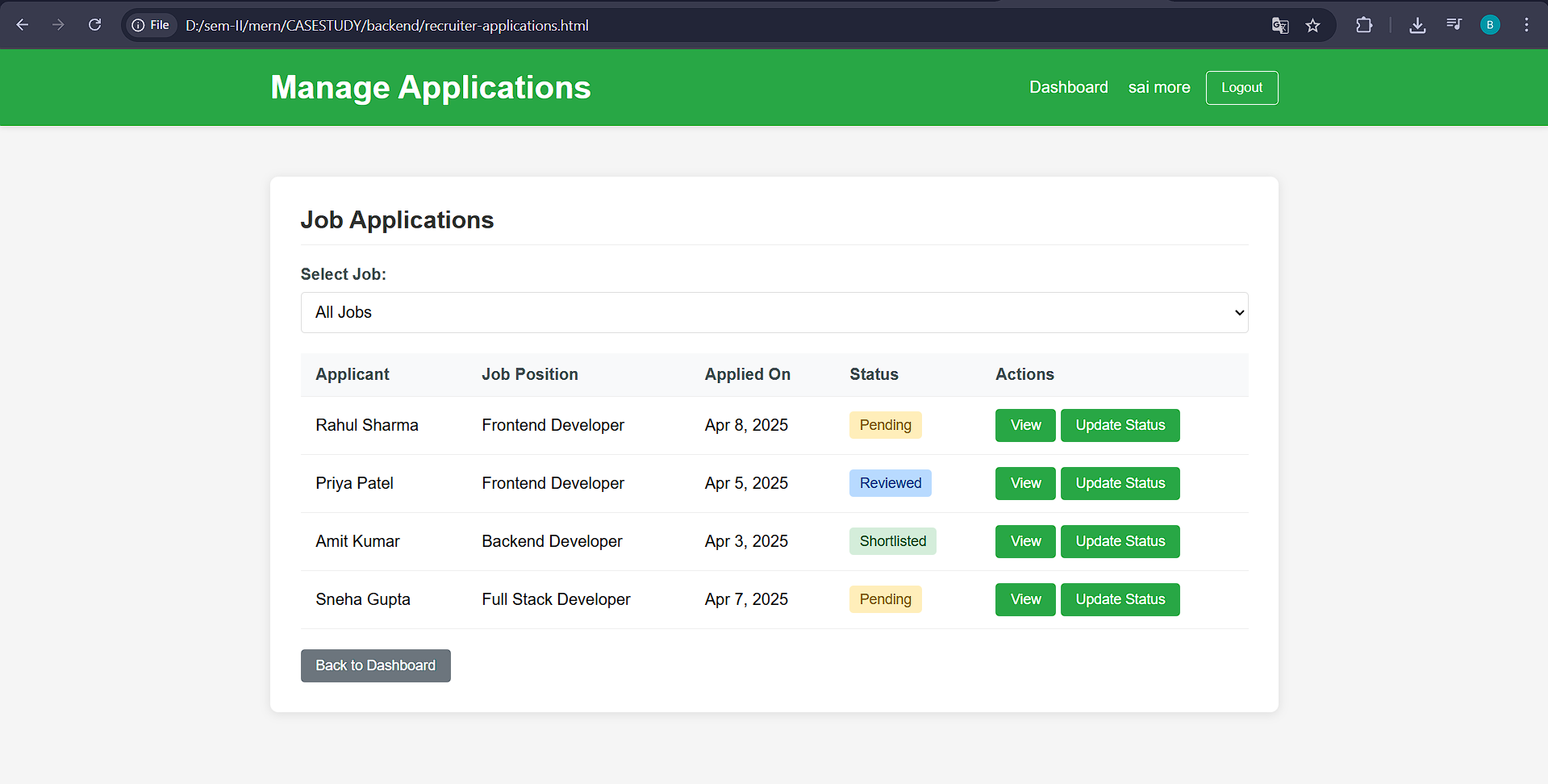


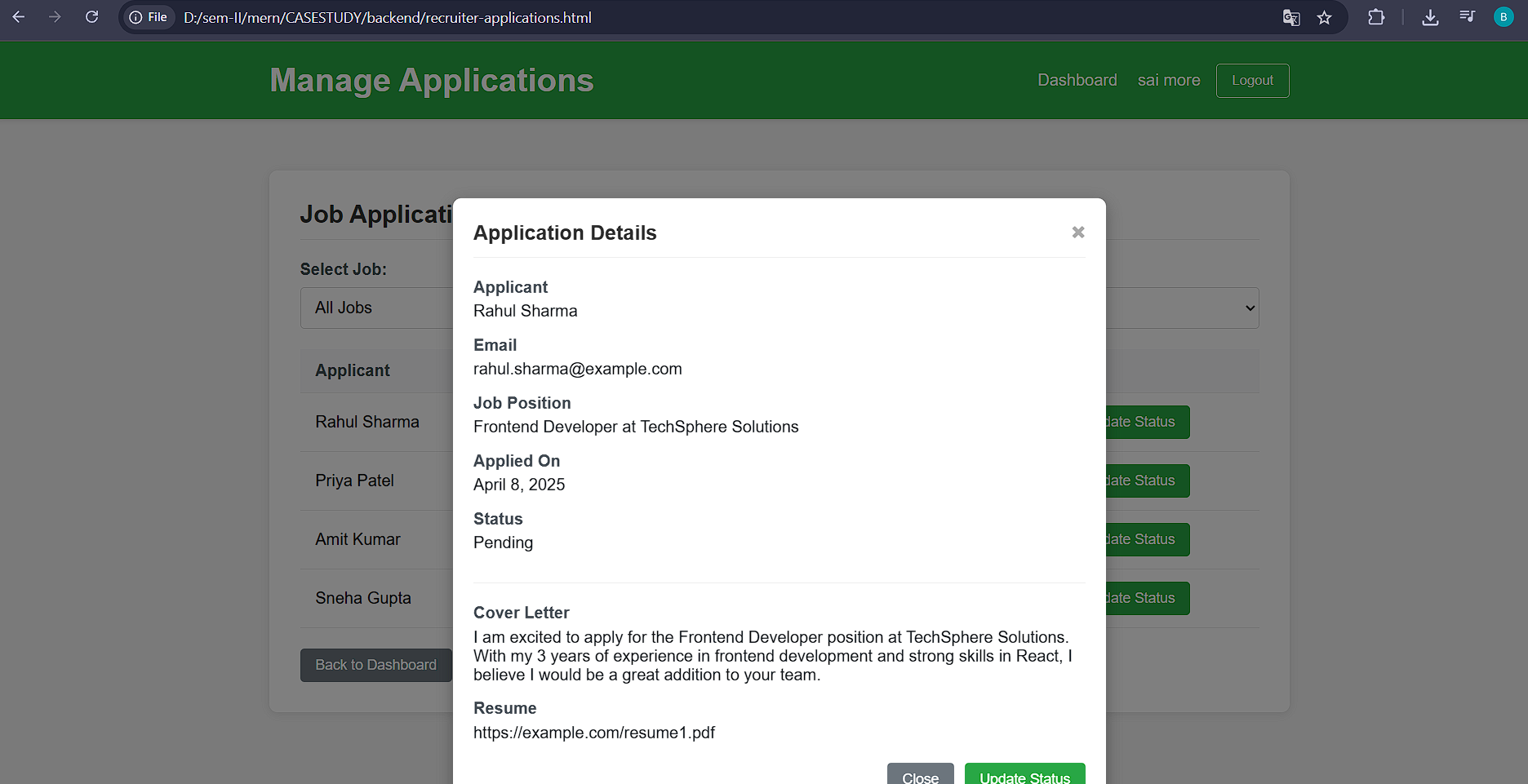
****

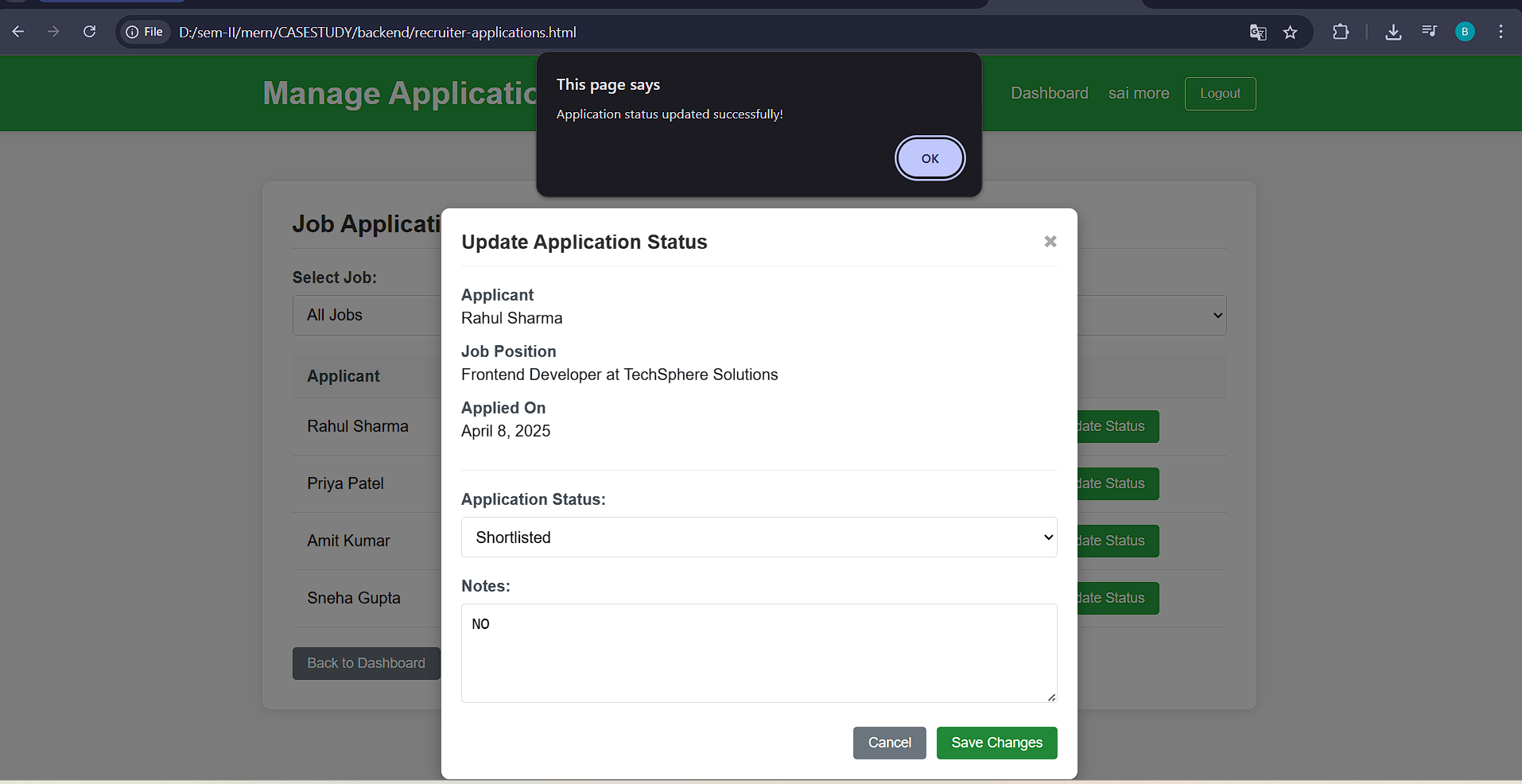
****

**4. Job Seeker Dashboard**

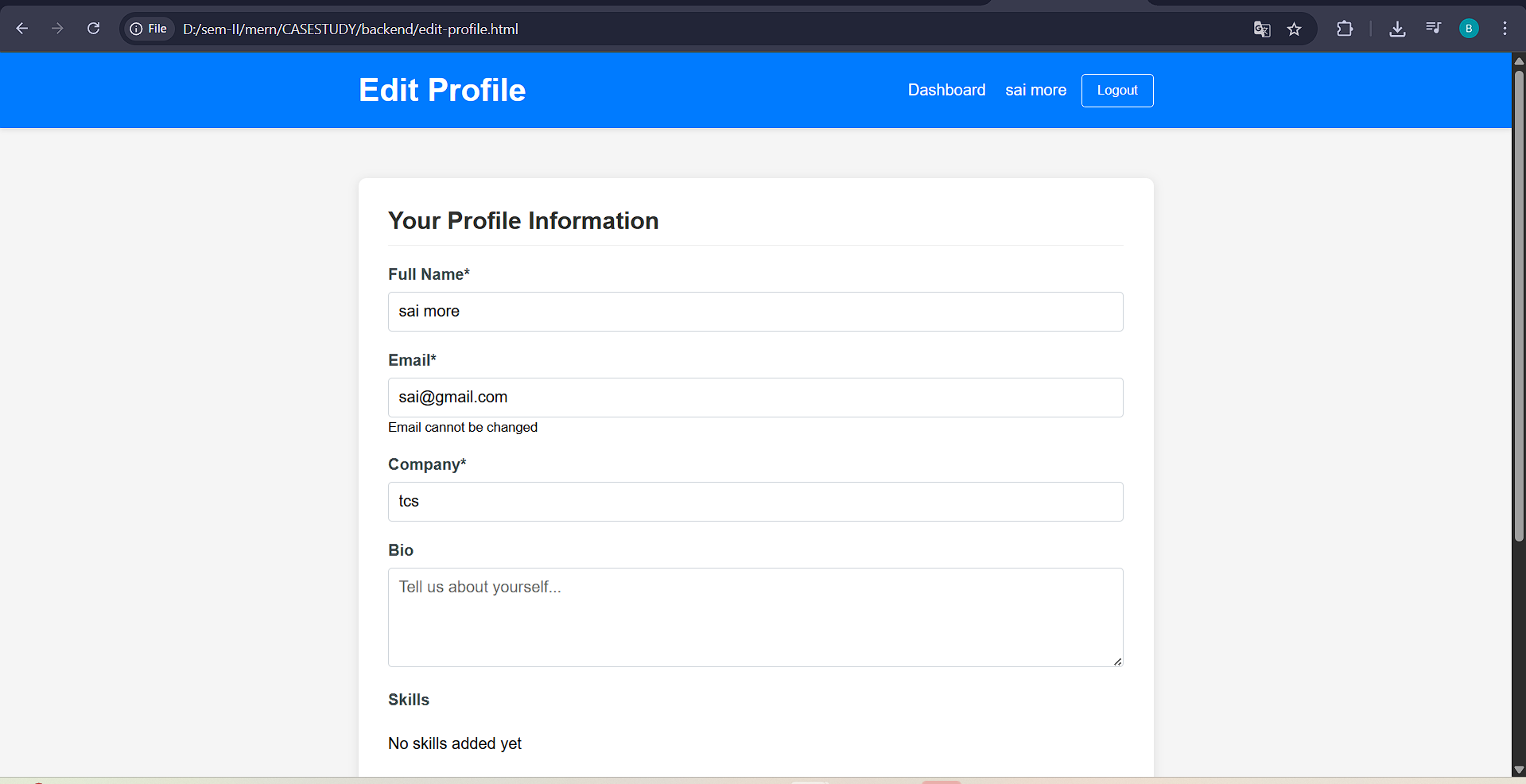
**5. Recruiter Dashboard** ****

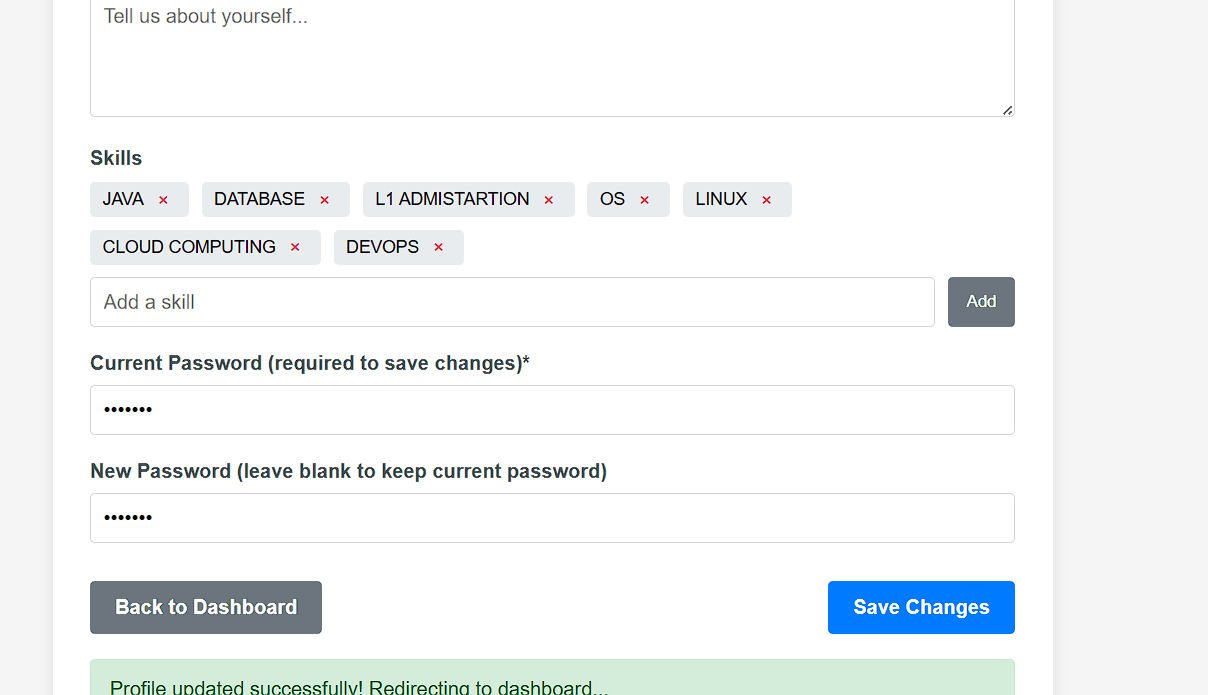
****

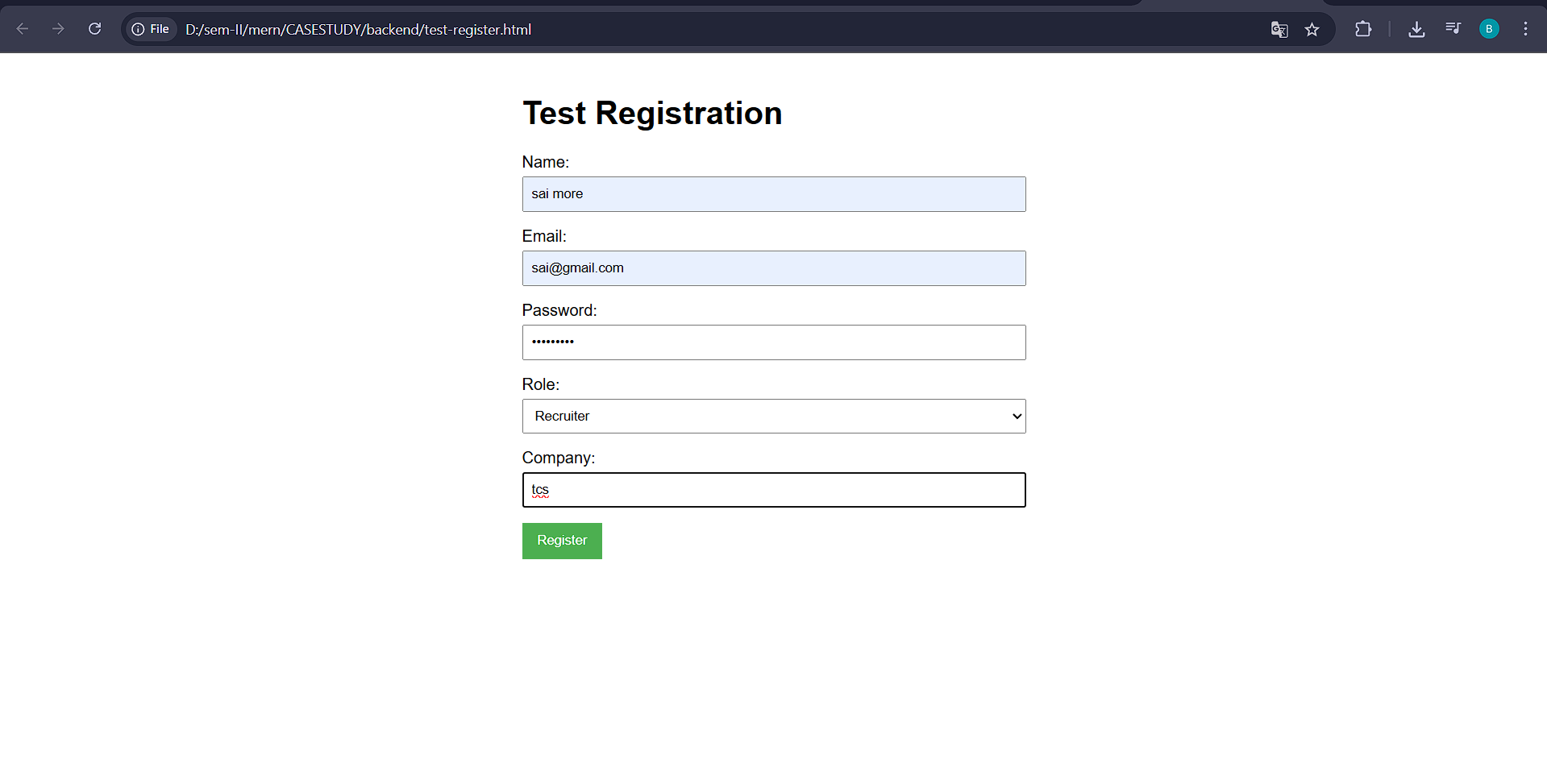
****

****

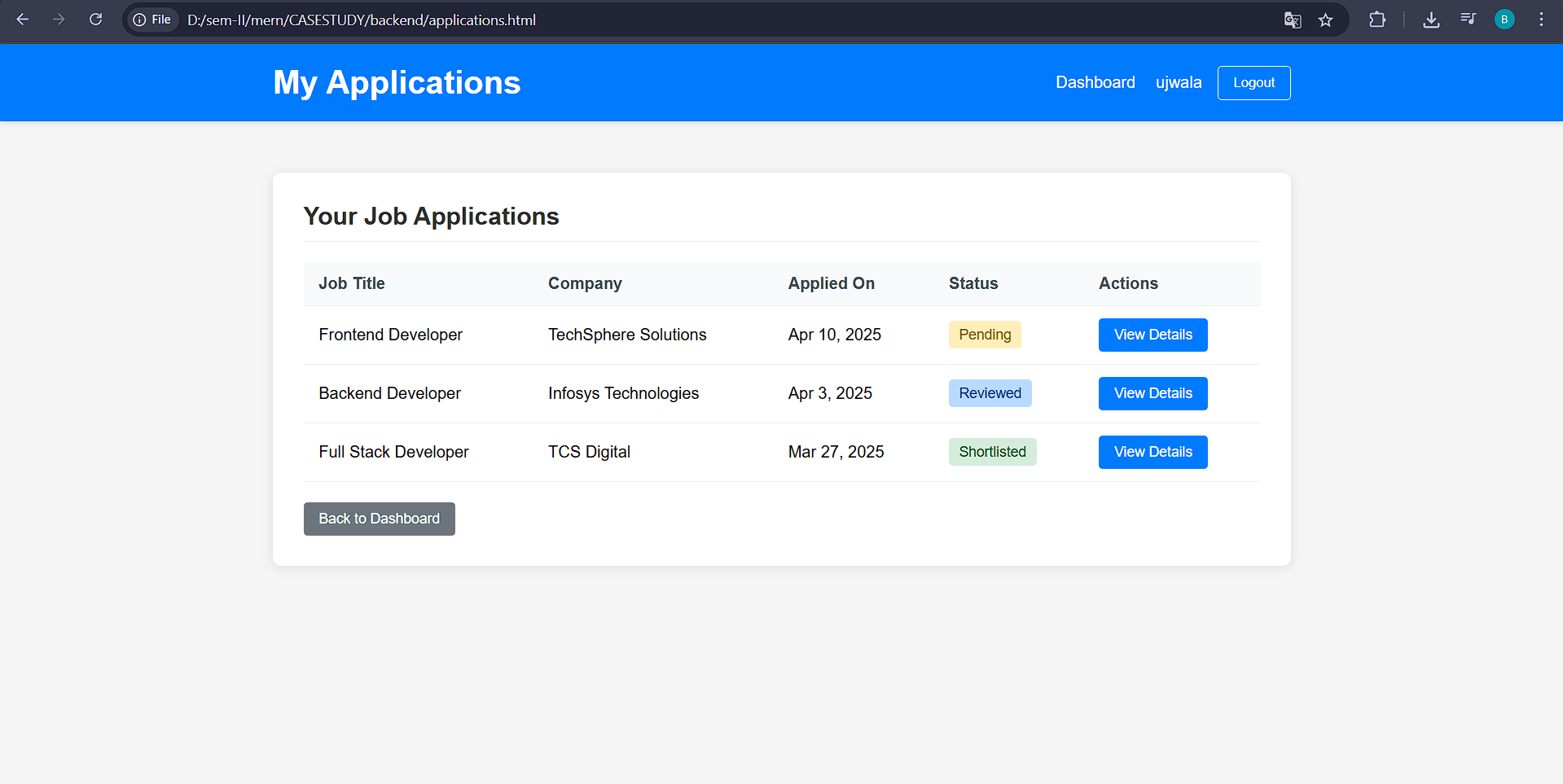
**EDIT PROFILE :**

****

****

**LOGIN:**

**6. Application Management**



**9. CODE SNIPPETS**

**1. User Authentication (Backend):**

// User login endpoint router.post('/login', async (req, res) => { try { const { email, password } = req.body; // Find user by email const user = await User.findOne({ email }); if (!user) { return res.status(400).json({ success: false, message: 'Invalid credentials' }); } // Check password const isMatch = await bcrypt.compare(password, user.password); if (!isMatch) { return res.status(400).json({ success: false, message: 'Invalid credentials' }); } // Generate JWT token const token = jwt.sign( { id: user.\_id, role: user.role }, process.env.JWT\_SECRET, { expiresIn: '30d' } ); res.json({ success: true, token, data: { \_id: user.\_id, name: user.name, email: user.email, role: user.role, company: user.company, skills: user.skills, experience: user.experience } }); } catch (error) { console.error(error); res.status(500).json({ success: false, message: 'Server error' }); } });

**2. Job Search (Frontend):**

// Function to search for jobs async function searchJobs() { const keyword = document.getElementById('keyword').value; const location = document.getElementById('location').value; const jobType = document.getElementById('jobType').value; try { let url = 'http://localhost:5000/api/jobs?'; if (keyword) url += `keyword=${encodeURIComponent(keyword)}&`; if (location) url += `location=${encodeURIComponent(location)}&`; if (jobType && jobType !== 'all') url += `jobType=${encodeURIComponent(jobType)}&`; const response = await fetch(url); const result = await response.json(); if (result.success) { displayJobs(result.data); setupPagination(result.pagination); } else { showError('Failed to fetch jobs'); } } catch (error) { console.error('Error:', error); showError('An error occurred while searching for jobs'); } }

**3. Application Status Update (Backend):**

// Update application status endpoint router.put('/applications/:id/status', auth, recruiterOnly, async (req, res) => { try { const { status, notes } = req.body; // Find application const application = await Application.findById(req.params.id); if (!application) { return res.status(404).json({ success: false, message: 'Application not found' }); } // Check if the job belongs to the recruiter const job = await Job.findById(application.job); if (!job || job.postedBy.toString() !== req.user.id) { return res.status(403).json({ success: false, message: 'Not authorized' }); } // Update application application.status = status; if (notes) application.notes = notes; await application.save(); res.json({ success: true, data: application }); } catch (error) { console.error(error); res.status(500).json({ success: false, message: 'Server error' }); } });

**4. Job Model (MongoDB Schema):**

const mongoose = require('mongoose'); const JobSchema = new mongoose.Schema({ title: { type: String, required: [true, 'Please add a title'], trim: true, maxlength: [100, 'Title cannot be more than 100 characters'] }, description: { type: String, required: [true, 'Please add a description'], maxlength: [5000, 'Description cannot be more than 5000 characters'] }, company: { type: String, required: [true, 'Please add a company name'], trim: true, maxlength: [100, 'Company name cannot be more than 100 characters'] }, location: { type: String, required: [true, 'Please add a location'], trim: true }, jobType: { type: String, required: [true, 'Please add a job type'], enum: ['Full-time', 'Part-time', 'Contract', 'Internship', 'Temporary'] }, skills: { type: [String], required: [true, 'Please add at least one skill'], validate: { validator: function(v) { return v && v.length > 0; }, message: 'Please add at least one skill' } }, salary: { type: String, trim: true }, postedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true } }, { timestamps: true }); // Create index for search JobSchema.index({ title: 'text', description: 'text', company: 'text', location: 'text' }); module.exports = mongoose.model('Job', JobSchema);

**10. CHALLENGES AND SOLUTIONS**

**1. Challenge: Implementing secure authentication**

**Solution:** Used JWT for token-based authentication and bcrypt for password hashing. We implemented middleware to verify tokens on protected routes and set up proper error handling for authentication failures.

**2. Challenge: Managing complex relationships between users, jobs, and applications**

**Solution:** Designed a well-structured MongoDB schema with proper references between collections. Used Mongoose populate method to retrieve related data efficiently.

**3. Challenge: Implementing an efficient search system**

**Solution:** Created text indexes in MongoDB for frequently searched fields like job title, description, and location. Implemented server-side filtering with query parameters for more specific searches.

**11. FUTURE ENHANCEMENTS**

* **Advanced Search Filters:** Salary range filter, experience level filter, remote/on-site filter
* **Recommendation System:** Job recommendations based on user skills and preferences
* **Messaging System:** Direct messaging between job seekers and recruiters
* **Analytics Dashboard:** Job performance metrics for recruiters, application success rate for job seekers
* **Integration with External Services:** LinkedIn profile import, resume parsing and analysis

**12. CONCLUSION**

The Job Portal Management System successfully addresses the needs of both job seekers and recruiters by providing a comprehensive platform for job posting, searching, and application management. The system's user-friendly interface, robust backend, and secure authentication make it a valuable tool for streamlining the recruitment process.

Through this project, we have gained valuable experience in full-stack web development, database design, and implementing secure authentication. We have also learned the importance of user experience design and responsive web development.

The system is scalable and can be extended with additional features in the future to meet evolving user needs and market demands.

**13. REFERENCES**

1. MongoDB Documentation: <https://docs.mongodb.com/>
2. Express.js Documentation: <https://expressjs.com/>
3. Node.js Documentation: <https://nodejs.org/en/docs/>
4. JWT Documentation: <https://jwt.io/>
5. Mongoose Documentation: <https://mongoosejs.com/docs/>
6. MDN Web Docs: <https://developer.mozilla.org/>
7. W3Schools: <https://www.w3schools.com/>