

**COLLEGE CODE : 9222**

**COLLEGE NAME: Theni Kammavar Sangam College Of Technology DEPARTMENT: B.Tech(IT)**

**STUDENT NM-ID :712FF51E29E0A159F18453AF89FF7671**

**ROLL NO: 23IT005**

**DATE :17/10/2025**

**Completed the project named as Phase 5 TECHNOLOGY**

**PROJECT NAME : JOB APPLICATION TRACKER**

**SUBMITTED BY,**

**NAME : V.Balasankar**

**MOBILE NO:8508676825**



**Project Demonstration & Documentation**

**TITLE:IBM-NJ-JOB APPLICATION TRACKER**

**1. Final Demo Walkthrough**

Objective: Provide a clear and concise demonstration of how the app works, focusing on key features and user interactions.

**Content:**

* Introduction: Briefly describe the app's purpose and its key features (e.g., job tracking, resume uploads, reminders).
* Walkthrough: Show the user flow from start to finish:
* Sign Up/Log In: Show how users can sign up using Google, LinkedIn, or manually.
* Dashboard: Display the main dashboard with job application statuses (e.g., applied, interview, rejected).
* Adding Jobs: Demonstrate adding a new job, uploading resumes, and tagging it with relevant statuses.
* Tracking Job Status: Change the status of an application (from “applied” to “interviewing” or “rejected”) and show how it reflects visually.
* Reminders: Show the reminder system, which can send push notifications or emails when deadlines or interview dates are approaching.
* Notifications: Walk through any automated emails or SMS messages that are sent when statuses are updated.
* Analytics: If applicable, show how users can see their job application success rates or trends.
* Delivery: You can record a screen-share video for clarity, or do a live demo if presenting to a class, team, or stakeholders.

**2. Project Report**

* Objective: Summarize the entire project, from concept to execution.
* Content:
* Introduction: Brief description of the app and its main functionalities.

**Technology Stack:**

List all technologies used, including:

* Frontend: React.js, Vue.js, Tailwind CSS, Bootstrap, etc.
* Backend: Node.js, Django, Flask, etc
* Database: MongoDB, PostgreSQL, Firebase, etc.
* APIs: Third-party APIs integrated (e.g., LinkedIn, Google OAuth).
* Design Process: Include wireframes, UI/UX design considerations, and any prototyping tools used (Figma, Sketch, etc.).

**Implementation:**

* Walkthrough of the development phases:
* Feature development
* Code structure
* API development
* Challenges Faced: Mention any roadblocks and how you overcame them.

Future Improvements: Suggest features that could be added in future iterations.    
**Program:**<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Job Application Tracker</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 2rem;

background: #f9f9f9;

}

h1 {

text-align: center;

}

form {

background: #fff;

padding: 1rem;

border-radius: 8px;

max-width: 500px;

margin: 0 auto 2rem auto;

box-shadow: 0 2px 5px rgba(0,0,0,0.1);

}

label {

display: block;

margin-top: 1rem;

}

input[type="text"], select, textarea {

width: 100%;

padding: 0.5rem;

margin-top: 0.3rem;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

}

button {

margin-top: 1rem;

padding: 0.7rem 1.5rem;

background: #007bff;

border: none;

border-radius: 4px;

color: white;

cursor: pointer;

}

button:hover {

background: #0056b3;

}

.job-list {

max-width: 700px;

margin: 0 auto;

border-collapse: collapse;

width: 100%;

}

.job-list th, .job-list td {

border: 1px solid #ddd;

padding: 0.8rem;

text-align: left;

}

.job-list th {

background-color: #f2f2f2;

}

.status-applied {

color: green;

font-weight: bold;

}

.status-pending {

color: orange;

font-weight: bold;

}

.status-rejected {

color: red;

font-weight: bold;

}

</style>

</head>

<body>

<h1>Job Application Tracker</h1>

<form id="jobForm">

<label for="position">Position</label>

<input type="text" id="position" name="position" required />

<label for="company">Company</label>

<input type="text" id="company" name="company" required />

<label for="status">Status</label>

<select id="status" name="status" required>

<option value="Applied">Applied</option>

<option value="Pending">Pending</option>

<option value="Rejected">Rejected</option>

</select>

<label for="notes">Notes</label>

<textarea id="notes" name="notes" rows="3"></textarea>

<button type="submit">Add Job</button>

</form>

<table class="job-list" id="jobListTable">

<thead>

<tr>

<th>Position</th>

<th>Company</th>

<th>Status</th>

<th>Notes</th>

</tr>

</thead>

<tbody id="jobList">

</tbody>

</table>

<script>

const form = document.getElementById('jobForm');

const jobList = document.getElementById('jobList');

form.addEventListener('submit', function(e) {

e.preventDefault();

const position = form.position.value.trim();

const company = form.company.value.trim();

const status = form.status.value;

const notes = form.notes.value.trim();

if (!position || !company) {

alert("Position and Company are required!");

return;

}

const tr = document.createElement('tr');

const tdPosition = document.createElement('td');

tdPosition.textContent = position;

tr.appendChild(tdPosition);

const tdCompany = document.createElement('td');

tdCompany.textContent = company;

tr.appendChild(tdCompany);

const tdStatus = document.createElement('td');

tdStatus.textContent = status;

tdStatus.className = `status-${status.toLowerCase()}`;

tr.appendChild(tdStatus);

const tdNotes = document.createElement('td');

tdNotes.textContent = notes;

tr.appendChild(tdNotes);

jobList.appendChild(tr);

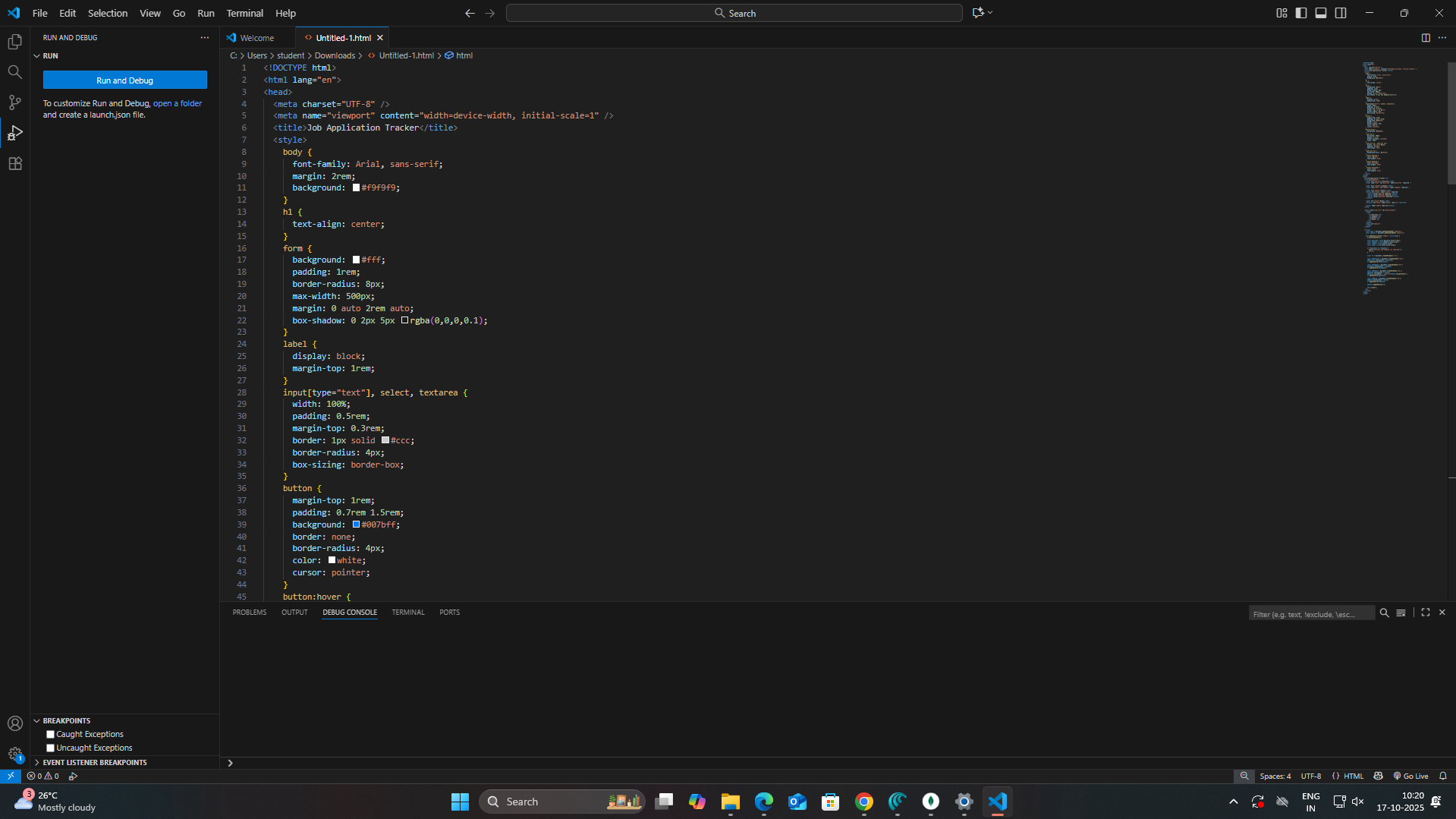
form.reset();

});

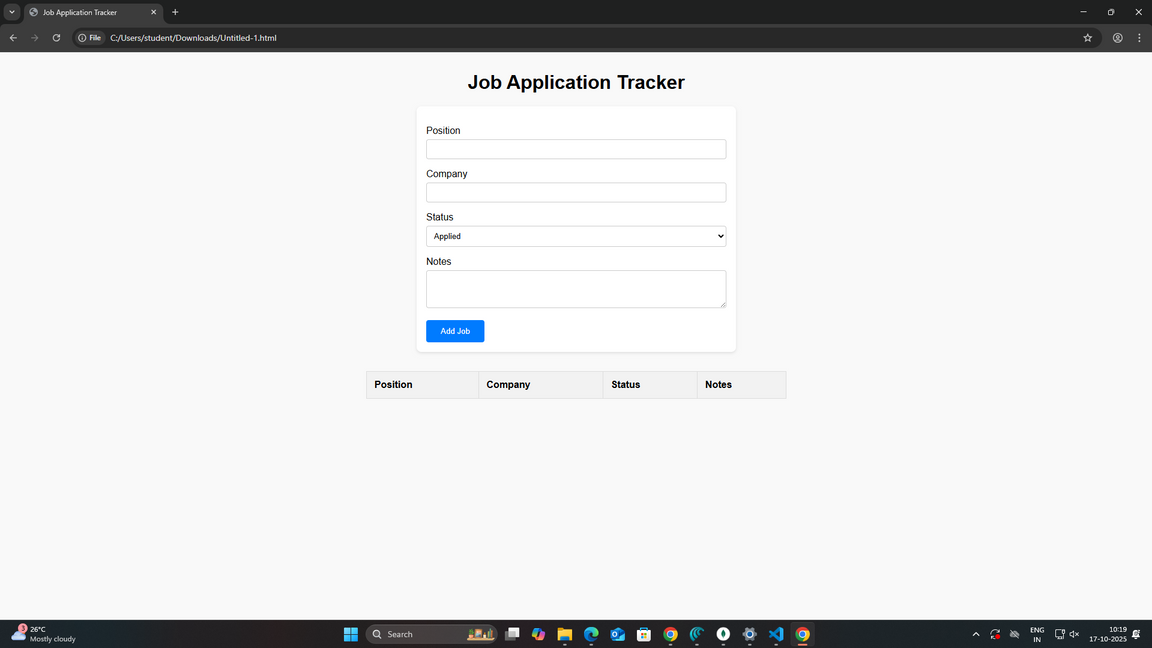
</script>

</body>

</html>**3. Screenshots / API Documentation**



**OUTPUT:**



**4. Challenges & Solutions**Objective: Demonstrate problem-solving skills and reflection on what was learned throughout the development.

Challenges:

API Integrations: Did you face difficulties while integrating with third-party APIs (e.g., LinkedIn, job boards)? How did you overcome authentication or data fetching issues?

User Authentication: Was setting up secure login (OAuth) tricky? How did you ensure data security?

State Management: Did you run into challenges while managing app state (especially with job statuses or user sessions)? Did you use Redux or Context API (for React)?

UI/UX Design: Were there any issues designing a smooth, responsive interface? Did you need to refactor components for better mobile compatibility?

Performance: Were there performance bottlenecks? Did you optimize for speed and data fetching (e.g., lazy loading, pagination)?

Solutions:

Provide detailed solutions and techniques you used to fix the problems (e.g., using Axios for API calls, state management libraries, implementing lazy loading for job listings, etc.).

**5. GitHub README & Setup Guide**

<https://github.com/balasankar766-sudo/project.git>

**6. Final Submission**