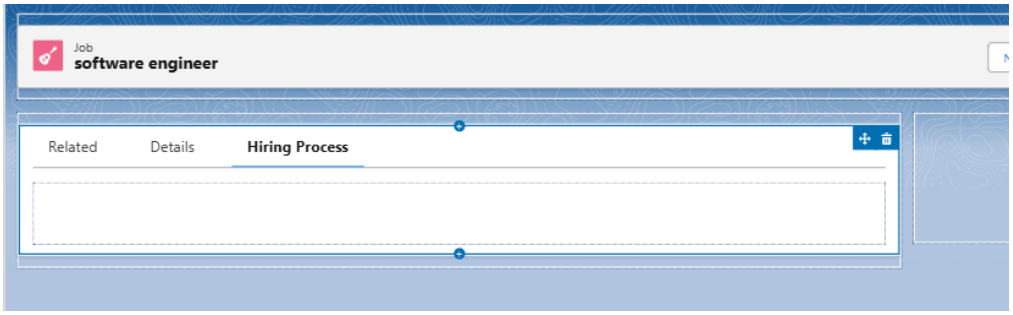
JOB RECRUITMENT & HIRING MANAGEMENT SYSTEM

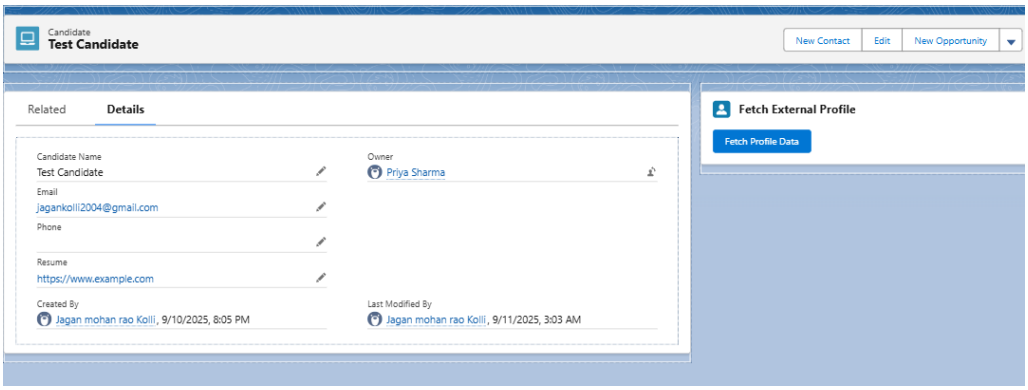
NAME: KOLLI JAGAN MOHAN RAO

Phase 6: User Interface Development

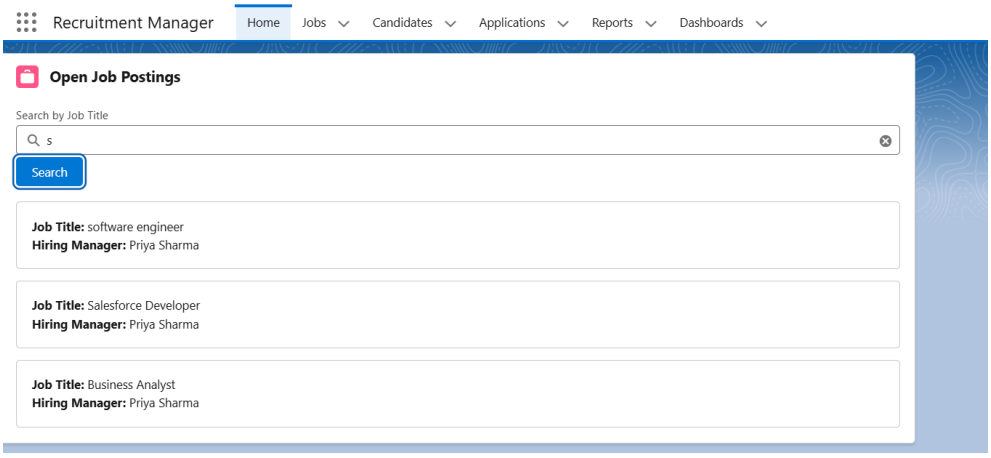
- **Lightning App Builder**:
    - This was the primary tool used for all declarative UI design. We used the Lightning App Builder's drag-and-drop canvas to create and modify our custom pages.

- **Record Pages**:
    - We significantly customized two key record pages:
        1. **Job Record Page**: We enhanced this page by removing the default single-column layout and replacing it with a **Tabs** component for better organization.
        2. **Candidate Record Page**: We modified this page by adding our custom candidateApiProfile LWC to the sidebar, demonstrating the ability to extend standard pages with custom components.





- **Home Page Layouts**:
    - We created a custom **Home Page** named "Recruitment Home Page" using the Lightning App Builder. This page was designed specifically for our application and featured our custom jobSearch LWC as the main component. We then assigned this custom page as the default Home Page for the "Recruitment Manager" app and the "System Administrator" profile.



- **LWC (Lightning Web Components)**:
    - We built two custom Lightning Web Components from scratch using VS Code:
        1. **jobSearch**: A component placed on the Home page that provides a UI for searching open job records.
        2. **candidateApiProfile**: A component placed on the Candidate record page that features a button to trigger an API callout.
    - For each LWC, we wrote the three core files: the **HTML** template for structure, the **JavaScript** controller for logic, and the **XML** metadata file to expose it to Salesforce.

jobSearch.js ×

force-app > main > default > lwc > jobSearch > jobSearch.js

```
1  import { LightningElement, track } from 'lwc';
2  // Import the Apex method
3  import findOpenJobs from '@salesforce/apex/jobSearchController.findOpenJobs';
4
5  export default class JobSearch extends LightningElement {
6      // Reactive properties to store our data and search term
7      @track jobs;
8      @track error;
9      searchKey = '';
10
11     // This function runs whenever the text in the search box changes
12     handleKeyChange(event) {
13         this.searchKey = event.target.value;
14     }
15
16     // This function runs when the 'Search' button is clicked
17     handleSearch() {
18         // Call our Apex method 'findOpenJobs' and pass the searchKey
19         findOpenJobs({ searchKey: this.searchKey })
20             .then(result => {
21                 // If the call is successful, store the results
```

jobSearch.html

force-app > main > default > lwc > jobSearch > jobSearch.html > template

```
<template>
    <lightning-card title="Open Job Postings" icon-name="standard:case">
        <div class="slds-m-around_medium">
            <lightning-input
                label="Search by Job Title"
                type="search"
                onchange={handleKeyChange}>
            </lightning-input>

            <lightning-button
                label="Search"
                variant="brand"
                onclick={handleSearch}
                class="slds-m-top_small">
            </lightning-button>

            <template if:true={jobs}>
                <div class="slds-m-top_medium">
                    <template for:each={jobs} for:item="job">
                        <div key={job.Id} class="slds-box slds-m-bottom_small">
                            <p><strong>Job Title:</strong> {job.Name}</p>
```

jobSearch.js-meta.xml

force-app > main > default > lwc > jobSearch > jobSearch.js-meta.xml > LightningComponentBundle

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>59.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

candidateApiProfile.html

orce-app > main > default > lwc > candidateApiProfile > candidateApiProfile.html > template

```
<template>
    <lightning-card title="Fetch External Profile" icon-name="standard:user">
        <div class="slds-m-around_medium">
            <lightning-button
                label="Fetch Profile Data"
                variant="brand"
                onclick={handleFetchProfile}>
            </lightning-button>

            <template if:true={profileData}>
                <div class="slds-box slds-m-top_medium">
                    <h2 class="slds-text-heading_small">API Response:</h2>
                    <pre>{profileData}</pre>
                </div>
            </template>

            <template if:true={error}>
                <div class="slds-m-top_medium">
                    <p>Error: {error}</p>
                </div>
            </template>
```

candidateApiProfile.js

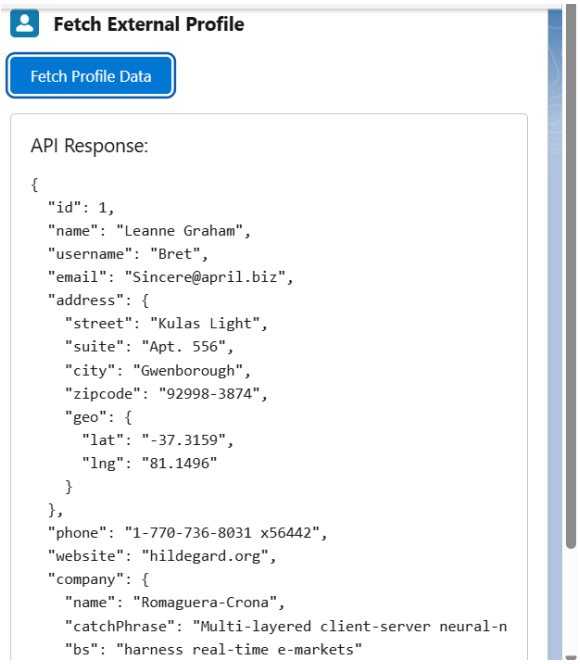
force-app > main > default > lwc > candidateApiProfile > candidateApiProfile.js > CandidateApiProfile

```
5   export default class CandidateApiProfile extends LightningElement {
14      handleFetchProfile() {
16          getProfileData({ candidateId: this.recordId })
17              .then(result => {
18                  // The result from Apex is a raw text (JSON) string.
19                  // This line formats it nicely with indentation.
20                  this.profileData = JSON.stringify(JSON.parse(result), null, 2);
21                  this.error = undefined;
22              })
23              .catch(error => {
24                  this.error = error.body.message;
25                  this.profileData = undefined;
26              });
27      }
28  }
```

candidateApiProfile.js-meta.xml

force-app > main > default > lwc > candidateApiProfile > candidateApiProfile.js-meta.xml > LightningComp

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
3       <apiVersion>59.0</apiVersion>
4       <isExposed>true</isExposed>
5       <targets>
6           <target>lightning__RecordPage</target>
7       </targets>
8       <targetConfigs>
9           <targetConfig targets="lightning__RecordPage">
10              <objects>
11                  <object>Candidate__c</object>
12              </objects>
13          </targetConfig>
14      </targetConfigs>
15  </LightningComponentBundle>
```



- **Apex with LWC**:
    - Both of our LWCs communicated with the Salesforce backend using **Apex**. We created two Apex controller classes (jobSearchController and CandidateProfileService) containing @AuraEnabled methods that our components could call to query data or perform actions.

- **Events in LWC**:
    - We used standard JavaScript events to make our components interactive. For example, in the jobSearch LWC, we used the **onchange** event on the input field to capture the user's search term and the **onclick** event on the button to trigger the search logic.

- **Wire Adapters**:
    - We initially used a **@wire** adapter in our jobSearch LWC to declaratively fetch the initial list of jobs from the findOpenJobs Apex method. This demonstrated the simplest way to get data from Salesforce.