13. Asynchronous Processing

Explanation:
Asynchronous processing (Batch Apex, Queueable, Scheduled, Future methods) allows operations to run in the background without blocking the main execution.

Scenario:

- Batch Apex: Closing inactive applications.
- Queueable Apex: Sending notifications for new job postings.
- Scheduled Apex: Interview reminders.
- Future Method: Background verification with external systems.

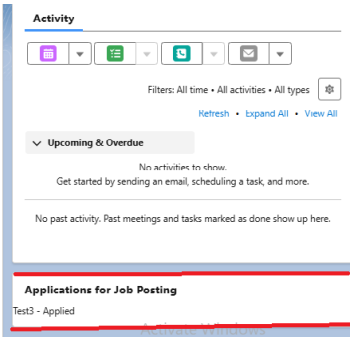
This ensured better performance and scalability of the system.

Phase 6: User Interface Development

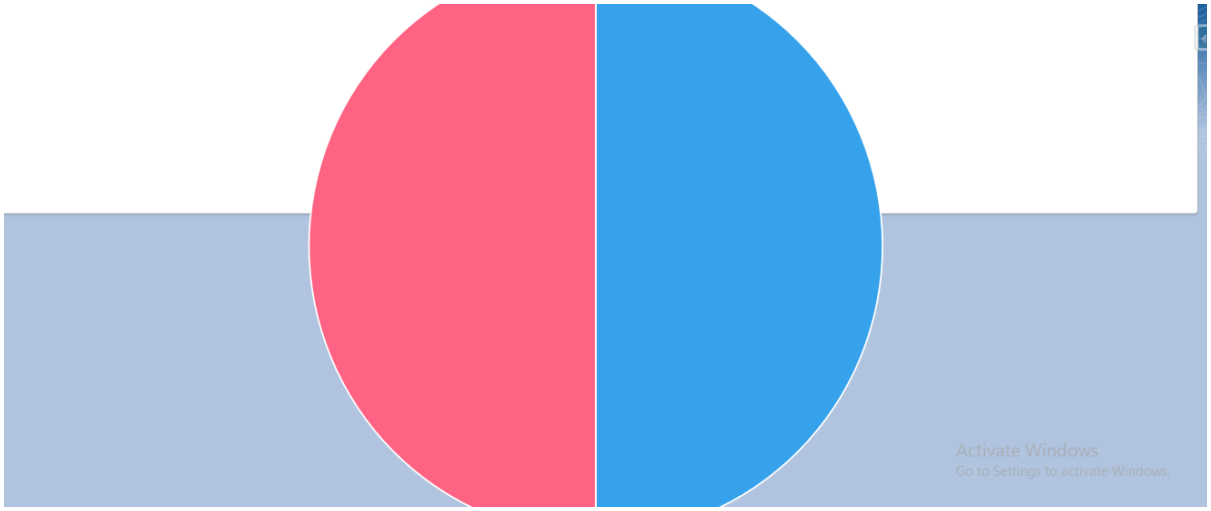
In this phase, the focus was on creating an intuitive and interactive interface for recruiters and HR managers to manage Job Postings, Applications, and Interviews efficiently. Salesforce Lightning Experience along with **Lightning Web Components (LWC)** was leveraged to enhance usability, display dynamic data, and integrate backend functionality.

1. Lightning App Builder & Record Pages

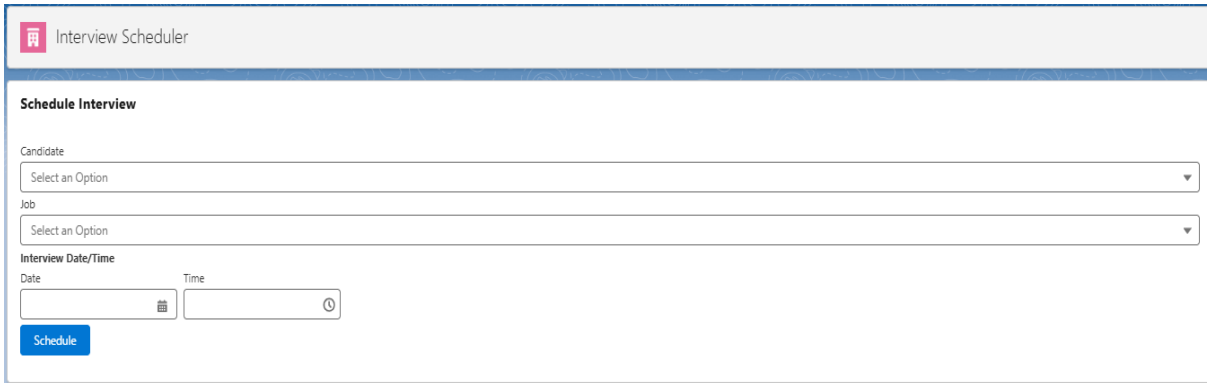
- **Job Posting Record Page**
  - Customized to display key details such as Job Title, Department, Location, Skills Required, and Application Deadline.
  - Integrated **Job Applications LWC**, which lists all Applications associated with the Job Posting dynamically.



  - Related lists of Applications and Interviews are included for complete visibility of job progress.
- **Application Record Page**
  - Displays candidate details, status, and associated Job Posting.
  - Integrated **Application Dashboard LWC**, summarizing candidate information and interview schedules.

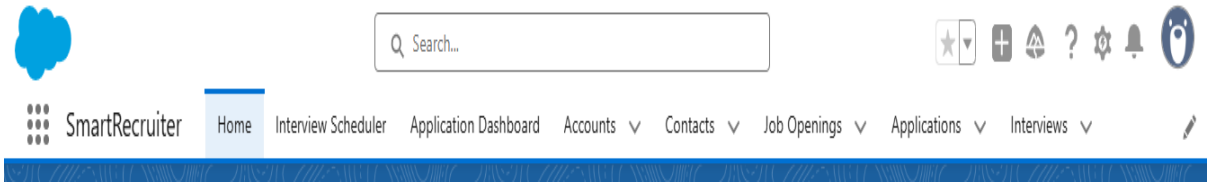


- **Interview Scheduler LWC** added to allow recruiters to schedule interviews directly from the Application record.



2. Tabs and Navigation

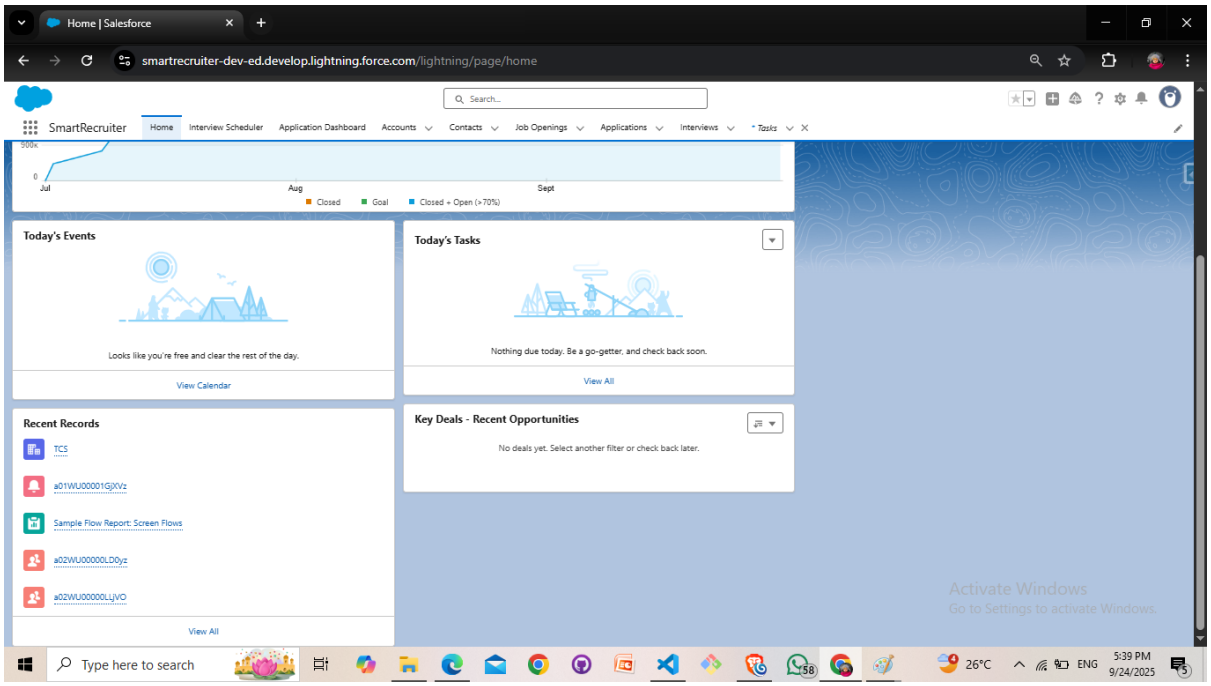
- Custom **tabs** created for Job Postings and Applications for easy access from the App Launcher.
- Custom Tabs for the object are present in the smart recruiter App in the app manager. As shown below



- Enables recruiters to navigate efficiently between Jobs, Applications, and Interviews.

3. Home Page Layouts & Utility Bar

- Home Page layouts designed to highlight recent applications, pending approvals, and upcoming interviews.



- Approval records and upcoming interviews I will be implementing using report and dashboards.
- Utility Bar can be configured to provide quick access to frequently used components like interview scheduling or application creation.

4. Lightning Web Components (LWC)

- **Job Applications LWC**
  - Displays all applications for a selected Job Posting.
  - Connected to Apex via **wire adapter**, providing real-time data display.
  - **Applications In Job Opening. HTML**

```
<template>
    <lightning-card title="Applications for Job Posting">
        <template if:true={applications}>
            <template for:each={applications} for:item="app">
                <p key={app.Id}>
                    {app.Contact__r.Name} - {app.Applicant_Status__c}
                </p>
            </template>
        </template>

        <template if:true={error}>
            <p class="slds-text-color_error">{error}</p>
        </template>
    </lightning-card>
</template>
```

  - **Applications In Job Opening. Js**

```
import { LightningElement, api, wire } from 'lwc';
import getApplicationsByJob from
'@salesforce/apex/AppicationInJobPosting.getApplicationsByJob';

export default class JobApplications extends LightningElement {
    @api recordId; // Job Posting Id from record page
    applications;
    error;

    @wire(getApplicationsByJob, { jobId: '$recordId' })
    wiredApplications({ error, data }) {
        if(data) {
            this.applications = data;
```

```
            this.error = undefined;
        } else if(error) {
            this.error = error.body ? error.body.message : error;
            this.applications = undefined;
        }
    }
}
```

- **Applications In Job Opening. Xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>61.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
    </targets>
</LightningComponentBundle>
```

- **Application Dashboard LWC**
  - Summarizes candidate information and associated interviews.
  - Provides visual insights into the recruitment pipeline.

- **Application Dashboard LWC . html**

```
<template>
    <lightning-card title="Applications Dashboard">
        <div class="chart-container">
            <canvas></canvas>
        </div>
```

```
    </lightning-card>
</template>
```

- **Application Dashboard LWC.js**

```
import { LightningElement, wire } from 'lwc';
import { loadScript } from 'lightning/platformResourceLoader';
import ChartJS from '@salesforce/resourceUrl/ChartJS';
import getApplicationsByStatus from
'@salesforce/apex/Application_Dashboard_Controller.getApplicationsByStatus';

export default class Application_Dashboard_Component extends LightningElement
{
    chart;
    chartData = {};
    chartInitialized = false;

    @wire(getApplicationsByStatus)
    wiredApplications({ error, data }) {
        if (data) {
            this.chartData = data;
            if (this.chartInitialized) {
                this.renderChart(); // re-render chart if Chart.js loaded
            }
        } else if (error) {
            console.error('Error fetching application data', error);
        }
    }

    renderedCallback() {
        if (this.chartInitialized) return;
        this.chartInitialized = true;

        loadScript(this, ChartJS)
            .then(() => {
                if (Object.keys(this.chartData).length > 0) {
                    this.renderChart();
                }
            })
            .catch(error => {
                console.error('Error loading ChartJS', error);
            });
    }
```

```
    renderChart() {
        const canvas = this.template.querySelector('canvas');
        if (!canvas) return;
        const ctx = canvas.getContext('2d');

        if (this.chart) {
            this.chart.destroy();
        }

        // eslint-disable-next-line no-undef
        // eslint-disable-next-line no-undef
this.chart = new Chart(ctx, {
    type: 'pie',
    data: {
        labels: Object.keys(this.chartData),
        datasets: [{
            data: Object.values(this.chartData),
            backgroundColor: ['#36A2EB', '#FF6384', '#FFCE56', '#4CAF50']
        }]
    },
    options: {
        responsive: true,       // makes it adjust to container
        maintainAspectRatio: true, // preserves aspect ratio
        legend: { position: 'bottom' }
    }
});

    }
}
```

- **Application Dashboard LWC.xml**

```
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>61.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

- **Application Dashboard LWC.css**

```css
.chart-container {
    width: 100%;
    max-width: 400px; /* chart won't exceed 400px */
    height: auto;
    margin: auto; /* center chart */
}

canvas {
    width: 100% !important;
    height: auto !important;
}
```

- **Interview Scheduler LWC**
    - Enables recruiters to schedule interviews for candidates directly from the Application record.
    - Collects interview date/time, mode (online/offline/phone), and assigns interviewers.
- **Interview Scheduler LWC.html**

```html
<template>
    <lightning-card title="Schedule Interview">
        <div class="slds-p-around_medium">
            <lightning-combobox
                label="Candidate"
                value={candidateId}
                options={candidateOptions}
                onchange={handleCandidateChange}>
            </lightning-combobox>

            <lightning-combobox
                label="Job"
                value={jobId}
                options={jobOptions}
                onchange={handleJobChange}>
```

```
            </lightning-combobox>

            <lightning-input
                type="datetime"
                label="Interview Date/Time"
                value={interviewDate}
                onchange={handleDateChange}>
            </lightning-input>

            <lightning-button
                variant="brand"
                label="Schedule"
                onclick={scheduleInterview}
                class="slds-m-top_medium">
            </lightning-button>
        </div>
    </lightning-card>
</template>
```

- **Interview Scheduler LWC.js**

```
import { LightningElement, track, wire } from 'lwc';
import getCandidates from '@salesforce/apex/schedule_Interview.getCandidates';
import getOpenJobs from '@salesforce/apex/schedule_Interview.getOpenJobs';
import scheduleInterviewApex from
'@salesforce/apex/schedule_Interview.scheduleInterview';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';

export default class InterviewScheduler extends LightningElement {
    @track candidateId;
    @track jobId;
    @track interviewDate;
    candidateOptions = [];
    jobOptions = [];

    @wire(getCandidates)
    wiredCandidates({ data }) {
        if (data) {
            this.candidateOptions = data.map(c => ({ label: c.Name, value:
c.Id }));
        }
    }

    @wire(getOpenJobs)
    wiredJobs({ data }) {
```

```
        if (data) {
            this.jobOptions = data.map(j => ({ label: j.Name, value: j.Id }));
        }
    }

    handleCandidateChange(event) {
        this.candidateId = event.detail.value;
    }
    handleJobChange(event) {
        this.jobId = event.detail.value;
    }
    handleDateChange(event) {
        this.interviewDate = event.detail.value;
    }

    scheduleInterview() {
        scheduleInterviewApex({ candidateId: this.candidateId, jobId:
this.jobId, interviewDate: this.interviewDate })
            .then(() => {
                this.dispatchEvent(new ShowToastEvent({
                    title: 'Success',
                    message: 'Interview Scheduled!',
                    variant: 'success'
                }));
                this.candidateId = this.jobId = this.interviewDate = null;
            })
            .catch(error => {
                console.error(error);
                this.dispatchEvent(new ShowToastEvent({
                    title: 'Error',
                    message: 'Failed to schedule interview',
                    variant: 'error'
                }));
            });
    }
}
```

- **Interview Scheduler LWC.xml**

```
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>61.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
```

```
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

5. Apex Integration with LWC

- LWCs connected to Apex controllers to fetch and display data dynamically.

Apex Class for Application Dashboard Controller

```
public with sharing class Application_Dashboard_Controller {
     @AuraEnabled(cacheable=true)
    public static Map<String, Integer> getApplicationsByStatus() {
        Map<String, Integer> statusCount = new Map<String, Integer>();

        // Query applications and group by status
        for (AggregateResult ar : [
            SELECT Applicant_Status__c status, COUNT(Id) cnt
            FROM Application__c
            GROUP BY Applicant_Status__c
        ]) {
            statusCount.put((String)ar.get('status'), (Integer)ar.get('cnt'));
        }

        return statusCount;
    }
}
```

Applications In job Opening -> apex class what fetch the data from the database

```
public with sharing class AppicationInJobPosting {
```

```
    @AuraEnabled(cacheable=true)
    public static List<Application__c> getApplicationsByJob(Id jobId) {
        return [
            SELECT Id, Name, Contact__r.Name,   Applicant_Status__c s
            FROM Application__c
            WHERE   Job__c = :jobId
            ORDER BY CreatedDate DESC
        ];
    }
}
```

- **Wire adapters** used to automatically update UI when data changes.

6. Events & Navigation Service

- LWCs communicate with each other using **custom events** where required, e.g., updating Application Dashboard after an interview is scheduled.
- **Navigation Service** used to redirect users to relevant record pages after actions such as interview creation.

Important Note - the code of all the lwc components in present in my github repo.