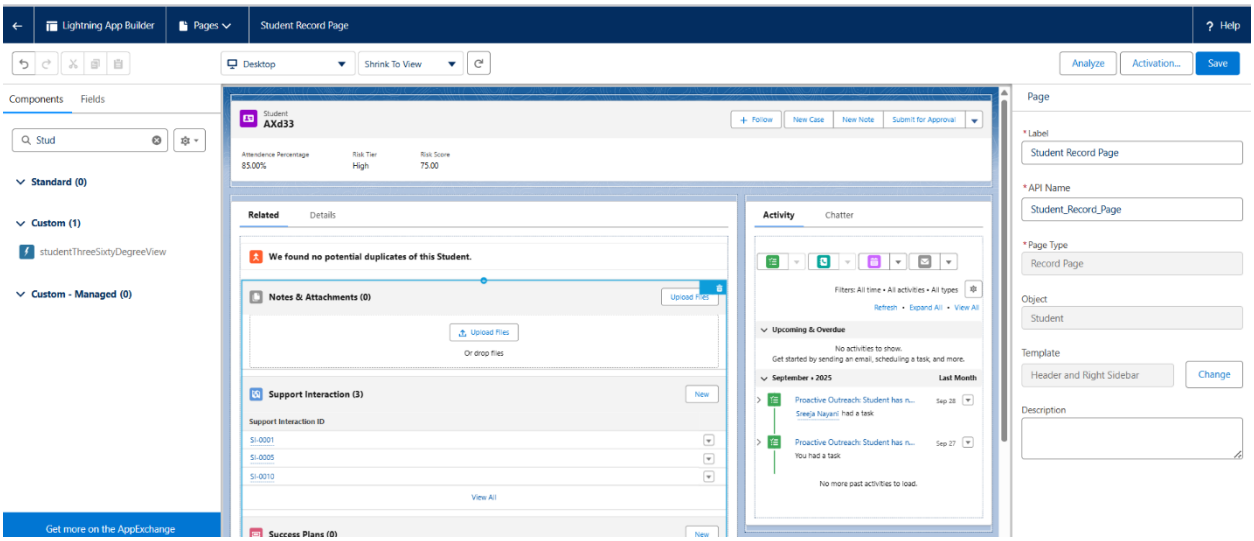
Phase 6: User Interface Development

In this phase we will build a custom "Student 360° View" Lightning Web Component (LWC) was developed to serve as a central dashboard on the Student record page, providing advisors with an at-a-glance summary of a student's status and recent activities.

6.1 Lightning App Builder

the Lightning App Builder was used to place components on the desired page. We used in previous Phases to arrange the components

App builder before LWC



6.2 Record Pages

The Student (Contact) object's Lightning Record Page was modified in this Phase. The standard "Highlights Panel" (controlled by the Compact Layout) was kept for key identifiers, and our new custom studentThreeSixtyView LWC was added prominently, demonstrating how standard and custom components can coexist to create a rich user experience.

6.3 Tabs

We created tabs to the objects so that they can be visible to the users according to their profiles

6.4 LWC (Lightning Web Components)

LWC (Lightning Web Components)

The core of this phase was the creation of the studentThreeSixtyView LWC. This involved a full-stack development process:

6.4.1 HTML Template (.html): The component's structure was built using HTML and standard lightning-* base components like lightning-card, lightning-layout, and lightning-badge. SLDS (Salesforce Lightning Design System) utility classes (slds-p-around_medium, slds-border_bottom, etc.) were used extensively for styling and layout, ensuring the component matched the look and feel of standard Salesforce UI.

Code:

```
<template>
  <lightning-card title="Student 360° View" icon-name="standard:user">
    <div class="slds-p-around_medium">
      <!-- Header Section -->
      <div class="slds-text-heading_large slds-p-bottom_small">
        {studentName}
      </div>
      <div class="slds-p-bottom_medium">
        <lightning-badge label={riskTier} class={riskBadgeClass}></lightning-badge>
      </div>

      <!-- Key Metrics Section -->
      <lightning-layout multiple-rows>
        <lightning-layout-item size="4" padding="around-small">
          <p class="slds-text-heading_small">Overall CGPA</p>
          <p class="slds-text-body_regular">{cgpa}</p>
        </lightning-layout-item>
        <lightning-layout-item size="4" padding="around-small">
          <p class="slds-text-heading_small">Current Backlogs</p>
          <p class="slds-text-body_regular">{backlogs}</p>
        </lightning-layout-item>
        <lightning-layout-item size="4" padding="around-small">
```

```
                <p class="slds-text-heading_small">Engagement Points</p>
                <p class="slds-text-body_regular">{engagementPoints}</p>
            </lightning-layout-item>
        </lightning-layout>
        <!-- Recent Activites Section -->
<div class="slds-p-top_large">

    <h3 class="slds-text-heading_large slds-p-bottom_medium">
        <strong>Recent Activities</strong>
    </h3>

    <div class="slds-box slds-m-bottom_large">
        <h4 class="slds-text-heading_medium slds-p-bottom_small slds-border_bottom">
            Recent Support Interactions
        </h4>

        <div class="slds-p-left_small slds-p-top_small">
            <ul>
                <template for:each={recentInteractions} for:item="interaction">
                    <li key={interaction.Id} class="activity-item">
                        <lightning-formatted-date-time
                            value={interaction.Interaction_Date__c}
                            year="numeric"
                            month="short"
                            day="2-digit">
                        </lightning-formatted-date-time>
                        : <strong>{interaction.Support_Type__c}</strong>
```

```
                <template if:true={interaction.Notes__c}>
                    <p class="activity-notes">{interaction.Notes__c}</p>
                </template>
            </li>
        </template>
    </ul>

    <template if:true={noInteractions}>
        <p class="slds-text-body_regular">No recent Support Interactions found.</p>
    </template>
  </div>
</div>


<div class="slds-box">
  <h4 class="slds-text-heading_medium slds-p-bottom_small slds-border_bottom">
    Recent Well-being Check-ins
  </h4>

  <div class="slds-p-left_small slds-p-top_small">
    <ul>
      <template for:each={recentCheckins} for:item="checkin">
        <li key={checkin.Id} class="activity-item">
          <lightning-formatted-date-time
            value={checkin.Date_of_check__c}
            year="numeric"
```

```
                    month="short"
                    day="2-digit">
                </lightning-formatted-date-time>
                : Overall Mood was <strong>{checkin.Overall_Mood__c}</strong>
              </li>
            </template>
          </ul>

          <template if:true={noCheckins}>
            <p class="slds-text-body_regular">No recent check-ins found.</p>
          </template>
        </div>
      </div>
    </div>
          <!-- Spinner for loading data -->
          <template if:true={isLoading}>
            <lightning-spinner alternative-text="Loading..." size="medium"></lightning-spinner>
          </template>
        </div>
      </lightning-card>
    </template>
```

6.4.2 A custom CSS file (.css) was also added to apply specific styles for spacing and visual separation.

Code:

```
li.activity-item {
  border-top: 1px solid #dddbda;
```

```
    padding-top: 0.75rem;
    margin-top: 0.75rem;
}
.activity-notes {
    /* Pushes the text to the right by 1.5rem (approx. 24px) to indent it. */
    padding-left: 1.5rem;
    font-style: italic;
      color: #444444;
}
```

6.4.3 JavaScript Controller (.js): This file contains all the client-side logic.

- **@api recordId:** A public property was used to receive the record ID from the Lightning Record Page, making the component context-aware.
- **Getters:** The component's logic is primarily built using JavaScript "getters" (get studentName(), get riskBadgeClass(), etc.). This pattern provides a clean way to process and format data from the Apex controller before it is rendered in the HTML, including conditional logic for the risk badge's color.

Code:

```
import { LightningElement, api, wire, track } from 'lwc';
// Import the Apex method we just created
import getStudentInfo from '@salesforce/apex/StudentViewController.getStudentInfo';

export default class StudentThreeSixtyView extends LightningElement {
    // This decorator makes the recordId property public and gets the ID from the page
    @api recordId;

    // The @wire decorator calls our Apex method.
    // It passes the student's recordId as a parameter.
    // The results are then provisioned into the studentData property.
```

```
@wire(getStudentInfo, { studentId: '$recordId' })
studentData;

// --- Getters to process the data returned from Apex ---

// This getter safely returns the student's name
get studentName() {
    return this.studentData.data ? this.studentData.data.Name : 'Loading...';
}

// This getter returns the risk tier
get riskTier() {
    return this.studentData.data ? this.studentData.data.Risk_Tier__c : '';
}

// This getter formats the CGPA
get cgpa() {
    return this.studentData.data ? this.studentData.data.Overall_CGPA__c : '...';
}

// Getter for backlogs
get backlogs() {
    return this.studentData.data ? this.studentData.data.Backlogs__c : '...';
}

// Getter for points
get engagementPoints() {
```

```
        return this.studentData.data ? this.studentData.data.Engagement_Points__c : '...';
    }

    // This getter returns the list of related Interactions
    get recentInteractions() {
    // First check if the Apex data and the relationship property both exist.
    // If they do, return the list. Otherwise, return an empty list [].
    if (this.studentData.data && this.studentData.data.Support_Interactions__r) {
        return this.studentData.data.Support_Interactions__r;
    }
    return []; // Always return an array
}

    // Getter to check if there are no activities, for showing the message
    get noInteractions() {
        return !this.isLoading && this.recentInteractions.length === 0;
    }
    // This  getter returns the list of related check-ins
   get recentCheckins() {
    // Use the correct relationship name you found in Step 1
    if (this.studentData.data && this.studentData.data.Well_being_Check_ins__r) {
        return this.studentData.data.Well_being_Check_ins__r;
    }
    return []; // Always return an array
}
    // Getter to check if there are no check-ins
    get noCheckins() {
```

```
    return !this.isLoading && this.recentCheckins.length === 0;
}

    // This getter tells the spinner when to show up
    get isLoading() {
        // The wire service has not returned data yet
        return !this.studentData.data && !this.studentData.error;
    }

    // This getter dynamically sets the color of our risk badge
    get riskBadgeClass() {
        const tier = this.riskTier;
        if (tier === 'High' || tier === 'Very High') {
            return 'slds-badge_lightest slds-theme_error';
        } else if (tier === 'Medium') {
            return 'slds-badge_lightest slds-theme_warning';
        }
        return 'slds-badge_lightest slds-theme_success';
    }
}
```

6.4.4 Metadata Configuration (.js-meta.xml): The component was exposed (isExposed>true</isExposed>) and configured to be available only on the lightning__RecordPage for the Contact object, ensuring it can only be placed where it is contextually relevant.

Code:

```
 <?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>64.0</apiVersion>
```

```
    <isExposed>True</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
    </targets>
    <targetConfigs>
        <targetConfig targets="lightning__RecordPage">
            <objects>
                <object>Contact</object>
            </objects>
        </targetConfig>
    </targetConfigs>
</LightningComponentBundle>
```

6.5 Apex with LWC

6.5.1 Apex Controller (StudentViewController.cls): A dedicated Apex class was created to act as the server-side controller for the LWC.

6.5.2 @AuraEnabled(cacheable=true): The getStudentInfo method was annotated with @AuraEnabled(cacheable=true). This is the critical annotation that securely exposes the Apex method to be called from the LWC and enables client-side caching for fast, efficient, read-only data retrieval.

6.5.3 Parent-Child Subqueries: The Apex method used an efficient SOQL query that included two parent-child subqueries. This allowed the component to fetch the student's own details, their five most recent Support Interactions, and their five most recent Well-being Check-ins in a single, performant database round-trip.

Apex Code:

```
public with sharing class StudentViewController {

    // The @AuraEnabled annotation is CRITICAL. It exposes this method to be called from a Lightning Component.
```

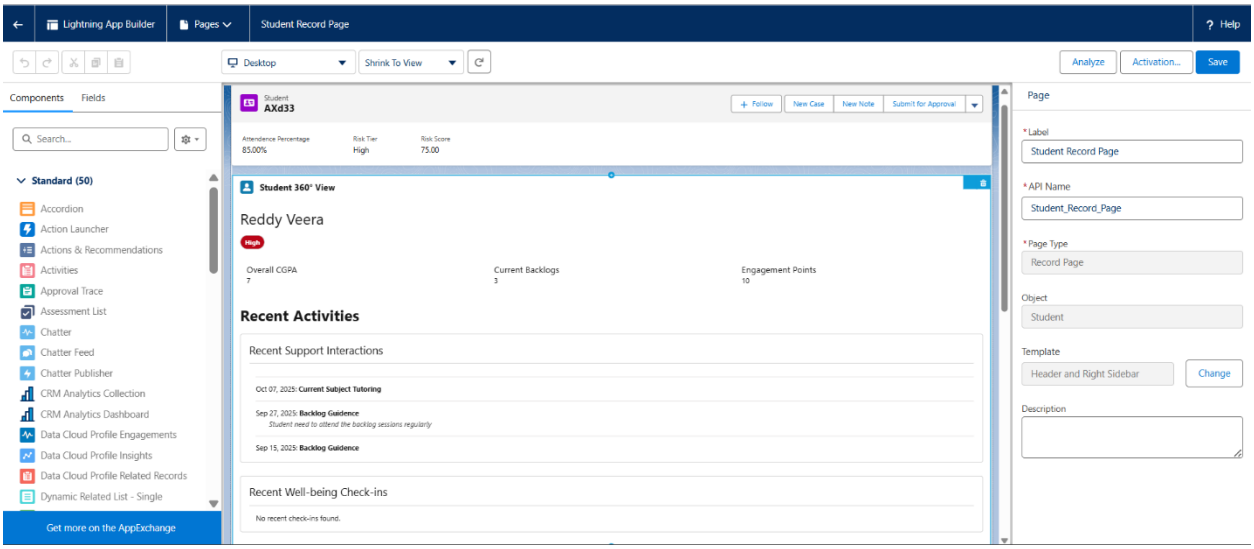
```
    // The (cacheable=true) is a best practice for read-only queries, making them fast and secure.
//@AuraEnabled(cacheable=true)
    @AuraEnabled(cacheable=true)
    public static Contact getStudentInfo(Id studentId) {
        // This single, powerful SOQL query gets the student's details AND their recent activities in one go.
        //"Parent-Child Subquery".
        return [SELECT
            Id,
            Name,
            Risk_Tier__c,
            Overall_CGPA__c,
            Backlogs__c,
            Engagement_Points__c,

            // Subquery 1: Get the 5 most recent Support Interactions
            (SELECT Id, Interaction_Date__c, Support_Type__c, Notes__c
             FROM Support_Interactions__r
             ORDER BY Interaction_Date__c DESC
             LIMIT 5),

            // Subquery 2: Get the 5 most recent Well-being Check-ins
            (SELECT Id, Date_of_check__c, Overall_Mood__c
             FROM Well_being_Check_ins__r
             ORDER BY Date_of_check__c DESC
             LIMIT 5)
```

```
        FROM Contact
        WHERE Id = :studentId
        LIMIT 1];
    }
}
```

6.6 Wire Adapters

- **@wire Adapter:** The LWC's JavaScript controller uses the @wire decorator to call the getStudentInfo Apex method. The wire service is a powerful, declarative feature of LWC. It automatically handles calling the Apex method, passing in the recordId, and provisioning the results to a property (studentData). It also automatically re-fetches the data if the recordId changes, making the component reactive.

Lightning Web Builder After LWC:



Final Student Record Page: