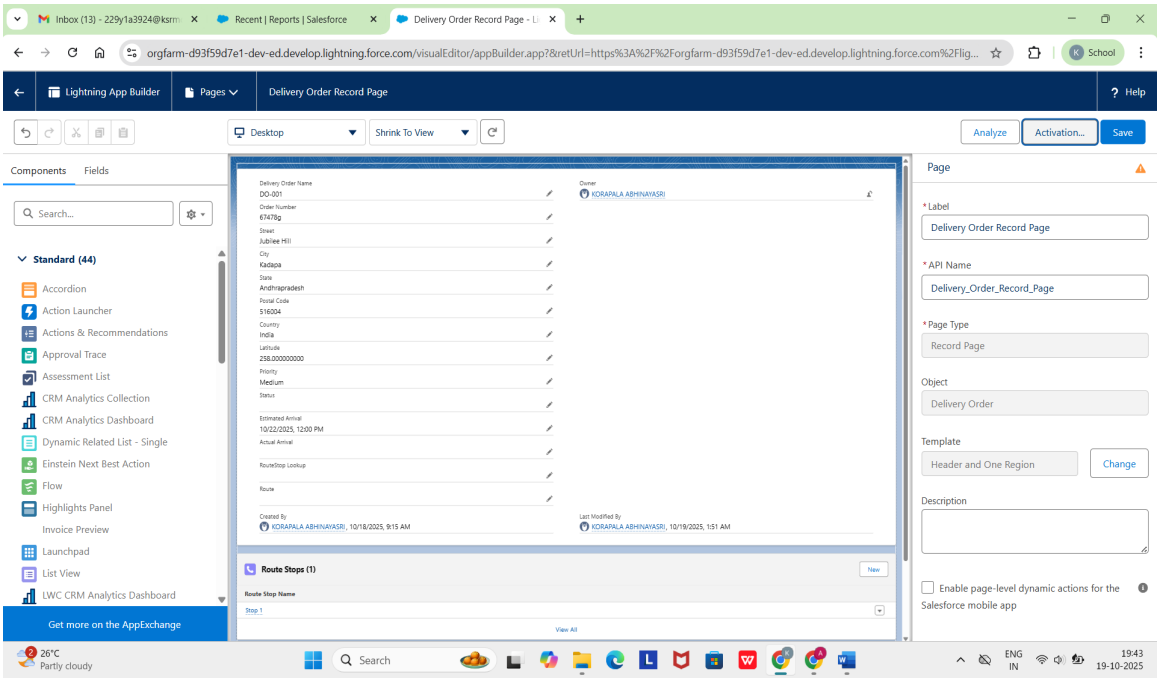
Dynamic Route Optimization — Phase 6: User Interface Development

In this phase, we focus on building the user interface for the Dynamic Route Optimization application using Salesforce's Lightning App Builder and Lightning Web Components (LWC). The goal is to create an interactive, modern interface that displays delivery orders, route details, and related information in real time. This phase combines Salesforce declarative tools like record pages and tabs with custom LWC and Apex backend logic.

Lightning App Builder and Record Pages

To begin, open Setup in your Salesforce Developer Org and search for "App Builder." Choose Lightning App Builder and click on "New → Record Page." Enter the label as "Delivery Order Record Page" and select the Delivery Order object. Choose the "Header and One Region" template and click Finish. On this page, you can customize what fields and components are visible. Add sections such as delivery details, route stop information, and estimated arrival time. Once configured, click Save and then Activation to make the page available for users.



Tabs, Home Page Layouts, and Utility Bar

Next, navigate to Setup → Tabs, and create tabs for your custom objects such as Delivery Orders, Routes, and Route Stops. These tabs allow users to quickly access each record type from the Salesforce app navigation bar. You can also customize the Home Page Layout to include key performance metrics, quick links, and charts related to delivery performance. Add a Utility Bar from the App Manager to show tools like Recent Items or Notes at the bottom of the app for quick access.

Lightning Web Component (LWC) Setup

After building the record page, you'll move to the programmatic side using Visual Studio Code (VS Code). Open VS Code and connect it to your Salesforce Org using the Salesforce CLI. Create a Lightning Web Component by pressing Ctrl + Shift + P, selecting SFDX: Create Lightning Web Component, and naming it deliveryOrderList. This component will display delivery orders dynamically. It retrieves data from the Apex backend using both wire adapters (for real-time data refresh) and imperative Apex calls (for user-triggered actions).

Example LWC JavaScript Code:

// deliveryOrderList.js import { LightningElement, wire } from 'lwc'; import getDeliveryOrders from '@salesforce/apex/RouteService.getDeliveryOrders'; export default class DeliveryOrderList extends LightningElement { deliveryOrders; error; @wire(getDeliveryOrders) wiredOrders({ data, error }) { if (data) { this.deliveryOrders = data; this.error = undefined; } else if (error) { this.error = error; this.deliveryOrders = undefined; } } }

Example LWC HTML Code:

{error}

Apex Integration with LWC

To support this component, create an Apex class named RouteService in your Salesforce Developer Console or VS Code. This class should include the following method to fetch delivery orders dynamically:

public with sharing class RouteService { @AuraEnabled(cacheable=true) public static List getDeliveryOrders() { return [SELECT Id, OrderNumber__c, City__c, Status__c FROM Delivery_Order__c LIMIT 10]; } }

Deploying and Adding LWC to Record Page

Once both the LWC and Apex class are ready, deploy them to your org using Ctrl + Shift + P → SFDX: Deploy Source to Org. After deployment, go back to Lightning App Builder, open the Delivery Order Record Page, and drag your new deliveryOrderList component from the Custom section onto the page layout. Click Save and Activate. Now, you can see a dynamic table showing all delivery orders directly on your record page.

Testing the User Interface

After publishing, open the Delivery Orders tab and select a record. You should now see your custom Lightning Web Component along with all standard fields. Verify that changes in the database reflect automatically in the LWC table. If there are issues such as missing data or access errors, check your Apex permissions and debug logs. This integration ensures that your users can interact with live delivery order data within an intuitive, responsive interface built using Lightning Web Components and Salesforce's Lightning framework.