Phase 6: User Interface Development

Airline Management System

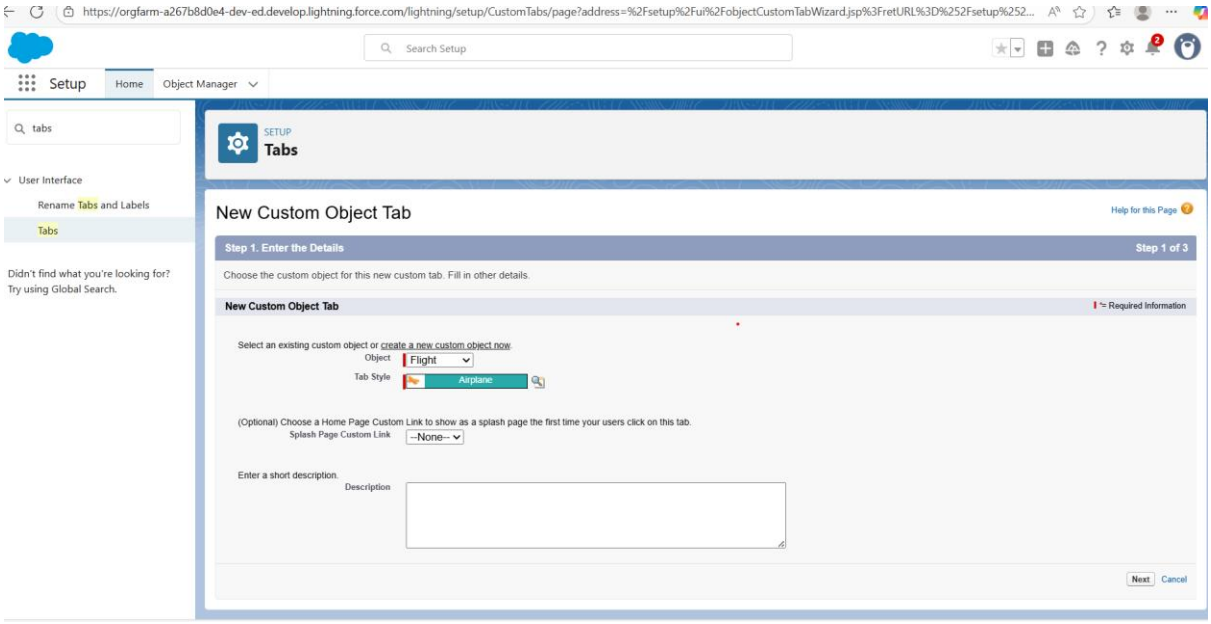
Salesforce-Based Passenger & Operations Management

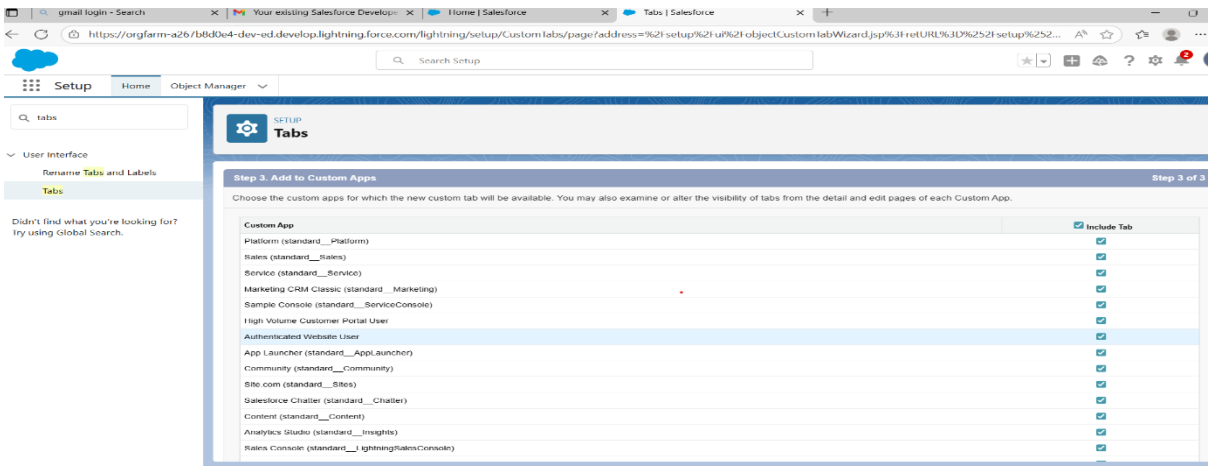
Step 1: Create a Lightning App

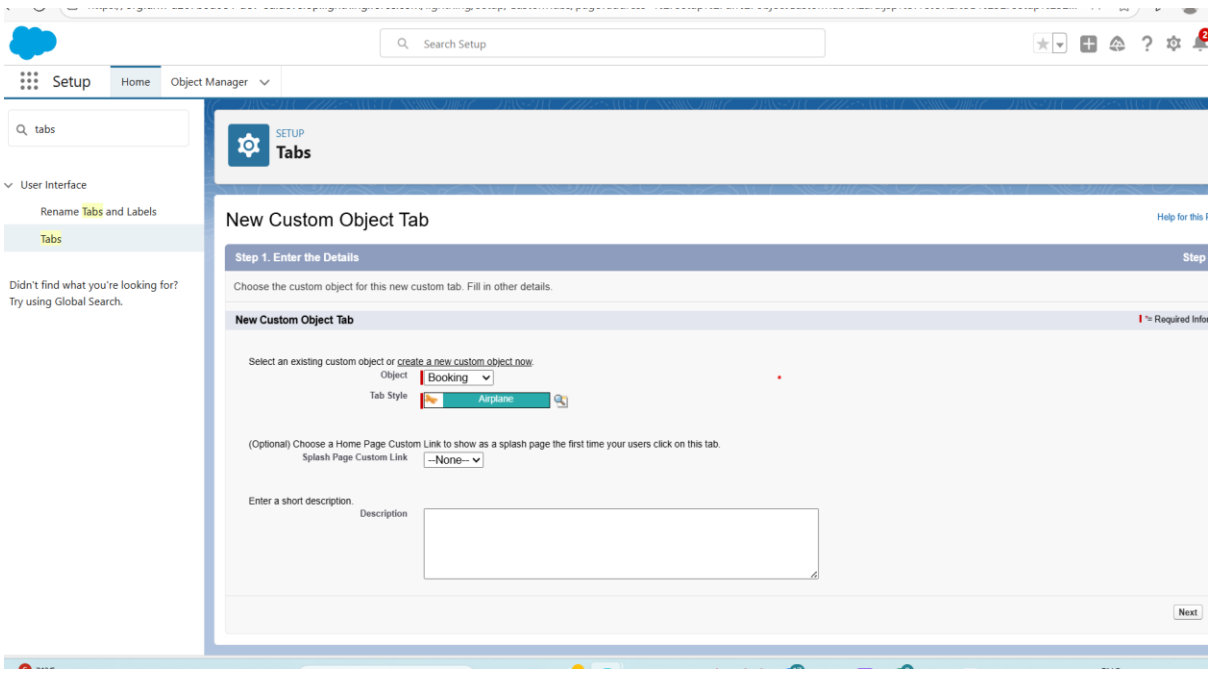
- Setup → **App Manager** → **New Lightning App**
- Name it Airline Console → choose navigation style → Save.

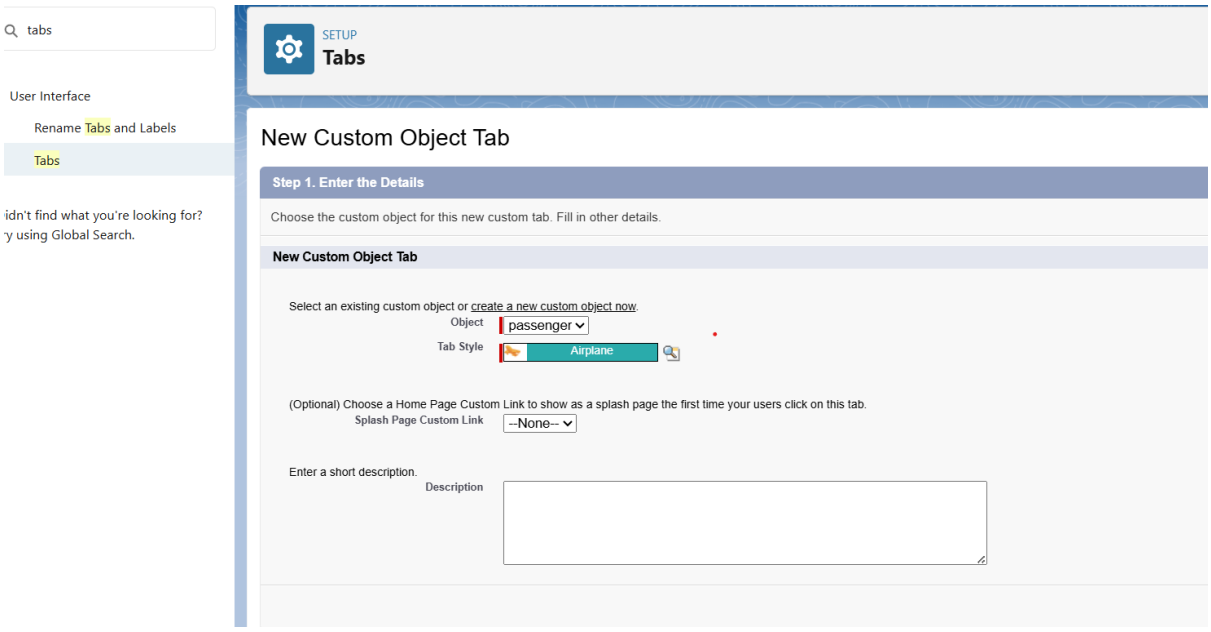
Step 2: Create Object Tabs

- Setup → **Tabs** → **New** → **Custom Object Tab**
- Create tabs for Flight__c, Booking__c, Passenger__c.
- Add these tabs to your Airline Console app.



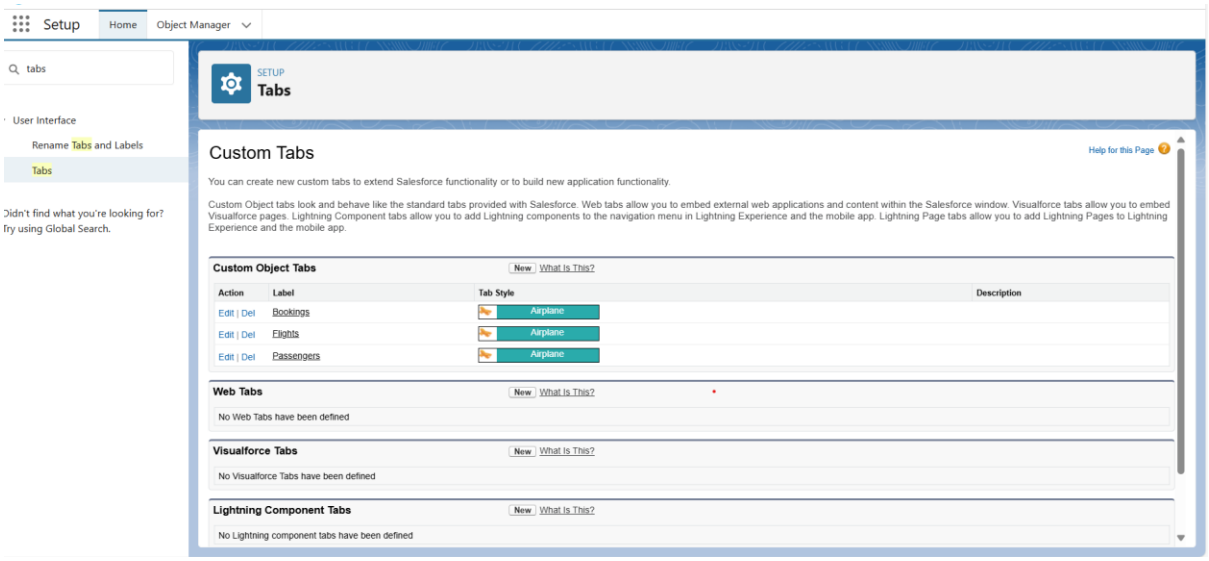






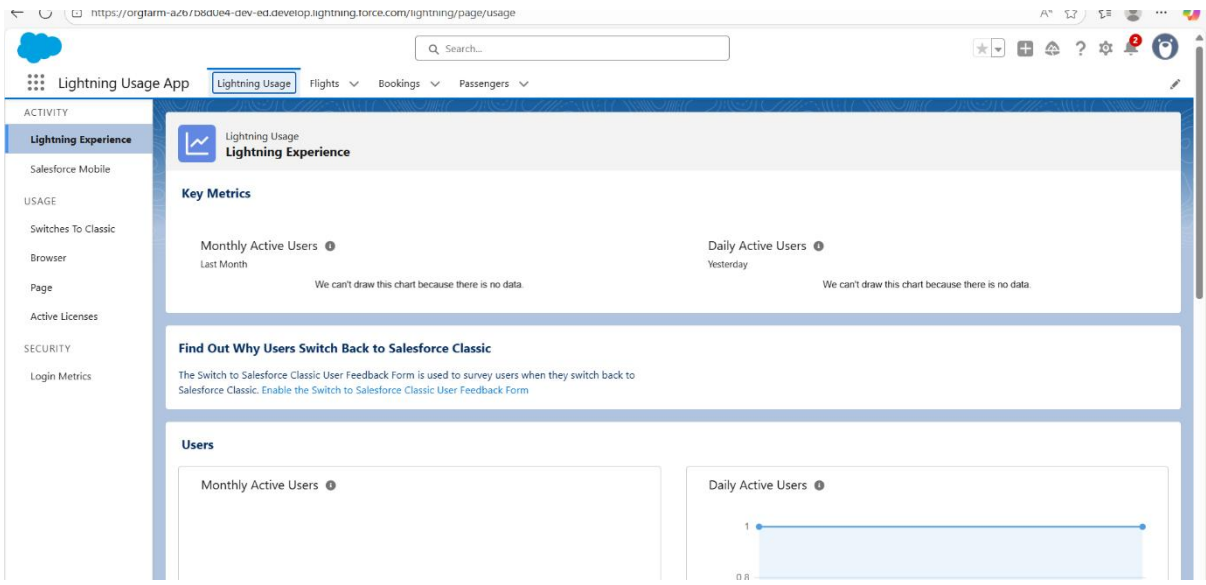
Step 3: Build Lightning Record Pages

- Setup → **Lightning App Builder → New Page → Record Page**.
- Select Flight__c → design layout (Record Details + Related Lists).
- Save (don't forget to **Activate** later).



Step 4: Customize Home Page & Utility Bar

- Setup → Lightning App Builder → New → Home Page.
- Add components (Reports, Dashboard, News).
- In App Manager → Edit App → Utility Bar → add *Notes*, *Recent Items*, or custom LWC.



Step 5: Create LWC Component

- In VS Code (SFDX Project): sfdx force:lightning:component:create --type lwc --componentname flightCard --outputdir force-app/main/default/lwc

- Files created: flightCard.html, flightCard.js, flightCard.js-meta.xml.

Step 6: Write LWC Code

- flightCard.html

```
<template>
  <lightning-card title={flightName}>
    <p>Status: {status}</p>
    <lightning-button label="Mark Completed" onclick={handleMarkCompleted}></lightning-button>
  </lightning-card>
</template>
```

- **flightCard.js**

```
import { LightningElement, api, wire } from 'lwc';
import getFlight from '@salesforce/apex/FlightController.getFlight';
import updateFlightStatus from '@salesforce/apex/FlightController.updateFlightStatus';

export default class FlightCard extends LightningElement {
  @api recordId;
  flight;

  @wire(getFlight, { flightId: '$recordId' }) wiredFlight({ data }) {
    if (data) this.flight = data;
  }

  get flightName() { return this.flight?.Name; }
  get status() { return this.flight?.Status__c; }

  handleMarkCompleted() {
    updateFlightStatus({ flightId: this.recordId, status: 'Completed' });
  }
}
```

- **FlightCard.js-meta.xml**

```xml
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>60.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

Step 7: Create Apex Controller

- **Setup → Apex Classes → New.**

```apex
@isTest
private class FlightControllerTest {
  @isTest static void testUpdate() {
    Flight__c f = new Flight__c(Name='TestFlight', Status__c='Scheduled');
    insert f;
    Test.startTest();
    FlightController.updateFlightStatus(f.Id, 'Completed');
    Test.stopTest();
    f = [SELECT Status__c FROM Flight__c WHERE Id=:f.Id];
    System.assertEquals('Completed', f.Status__c);
  }
}
```

Step 8: Deploy LWC & Apex

- In VS Code: right-click component → SFDX: Deploy Source to Org.
- Deploy Apex class too.

Step 9: Add LWC to Lightning Page

- Setup → **Lightning App Builder** → Open Flight_Record_Page_Custom.
- Drag flightCard component onto the page.
- **Save & Activate** → assign to App, Record Type, Profile.

Step 10: Test in Salesforce App

- Open a Flight__c record.
- Verify component displays flight info and button updates status.