

# 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.

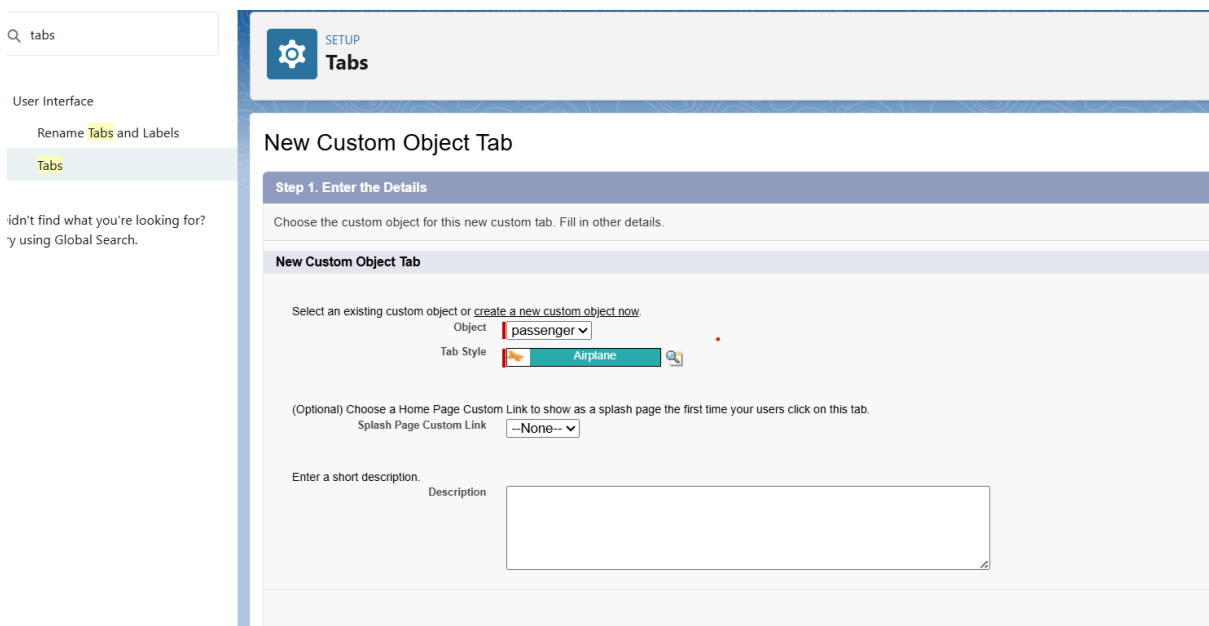
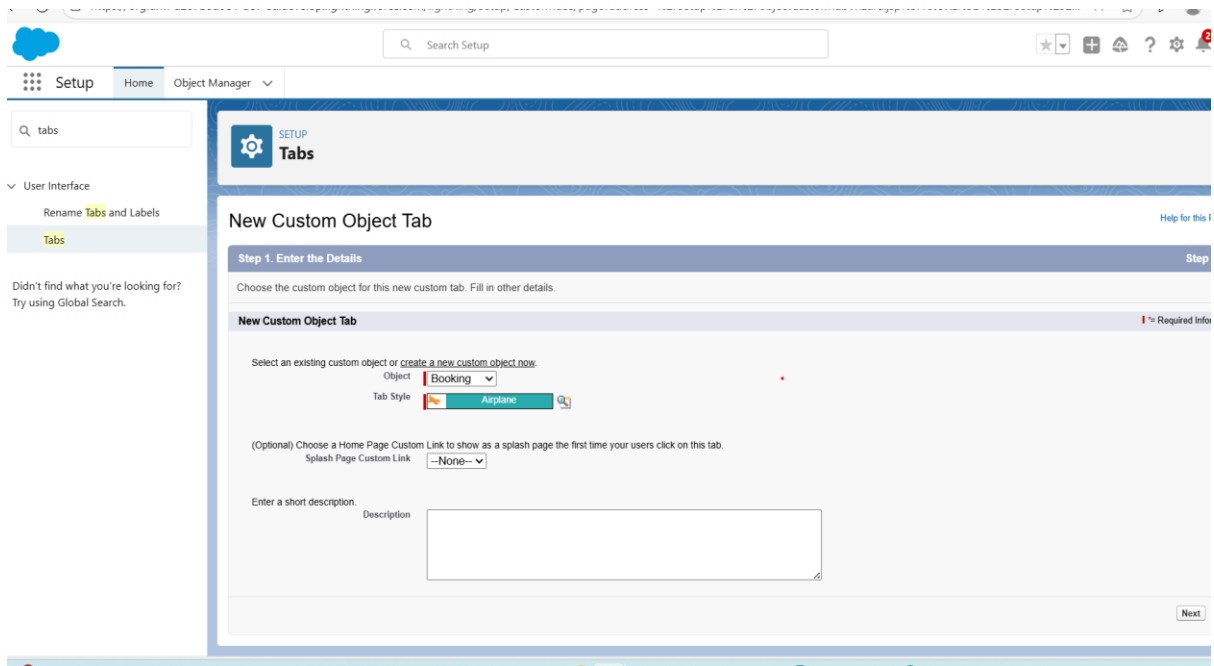
#### Step 2: Create Object Tabs

- Setup → **Tabs** → **New** → **Custom Object Tab**
- Create tabs for Flight\_\_c, Booking\_\_c, Passenger\_\_c.
- Add these tabs to your Airline Console app.

The screenshot shows the Salesforce Setup interface for creating a new custom object tab. The left sidebar shows the navigation menu with 'Setup' and 'Home' tabs. The main content area is titled 'New Custom Object Tab' and includes a 'Step 1. Enter the Details' section. The 'Object' dropdown is set to 'Flight'. The 'Tab Style' is set to 'Airplane'. The 'Splash Page Custom Link' is set to '--None--'. There is a text input field for 'Description'. The 'Next' and 'Cancel' buttons are at the bottom right.

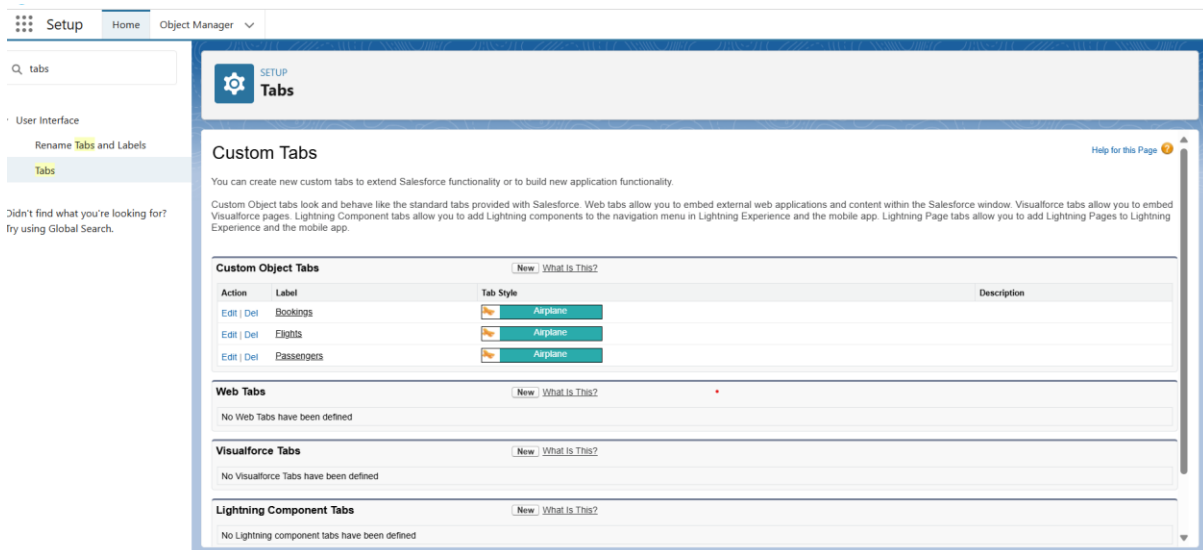
The screenshot shows the Salesforce Setup interface for adding the new custom tab to custom apps. The left sidebar shows the navigation menu with 'Setup' and 'Home' tabs. The main content area is titled 'Add to Custom Apps' and includes a table with columns 'Custom App' and 'Include Tab'. The table lists various Salesforce apps, and the 'Include Tab' column has checkboxes for each. The 'Authenticated Website User' app is highlighted.

Custom App	Include Tab
Platform (standard__Platform)	<input checked="" type="checkbox"/>
Rules (standard__Rules)	<input checked="" type="checkbox"/>
Service (standard__Service)	<input checked="" type="checkbox"/>
Marketing CRM Classic (standard__Marketing)	<input checked="" type="checkbox"/>
Sample Console (standard__ServiceConsole)	<input checked="" type="checkbox"/>
High Volume Customer Portal User	<input checked="" type="checkbox"/>
Authenticated Website User	<input checked="" type="checkbox"/>
App Launcher (standard__AppLauncher)	<input checked="" type="checkbox"/>
Community (standard__Community)	<input checked="" type="checkbox"/>
Site.com (standard__Site)	<input checked="" type="checkbox"/>
Salesforce Chatbot (standard__Chatbot)	<input checked="" type="checkbox"/>
Content (standard__Content)	<input checked="" type="checkbox"/>
Analytics Studio (standard__Insights)	<input checked="" type="checkbox"/>
Rules Console (standard__LightningRulesConsole)	<input checked="" type="checkbox"/>



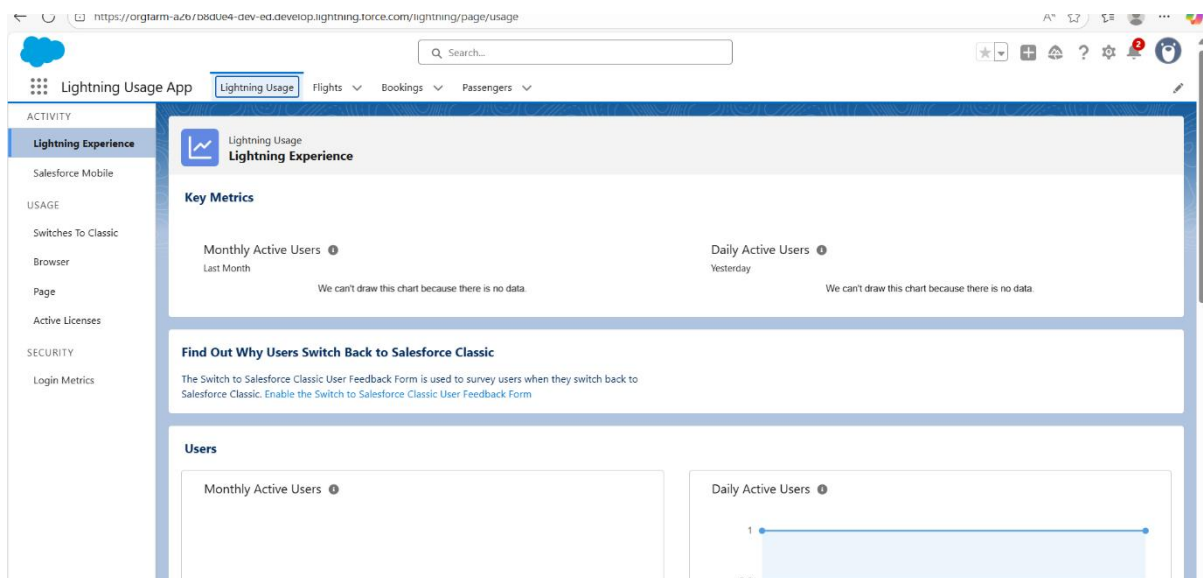
### Step 3: Build Lightning Record Pages

- 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

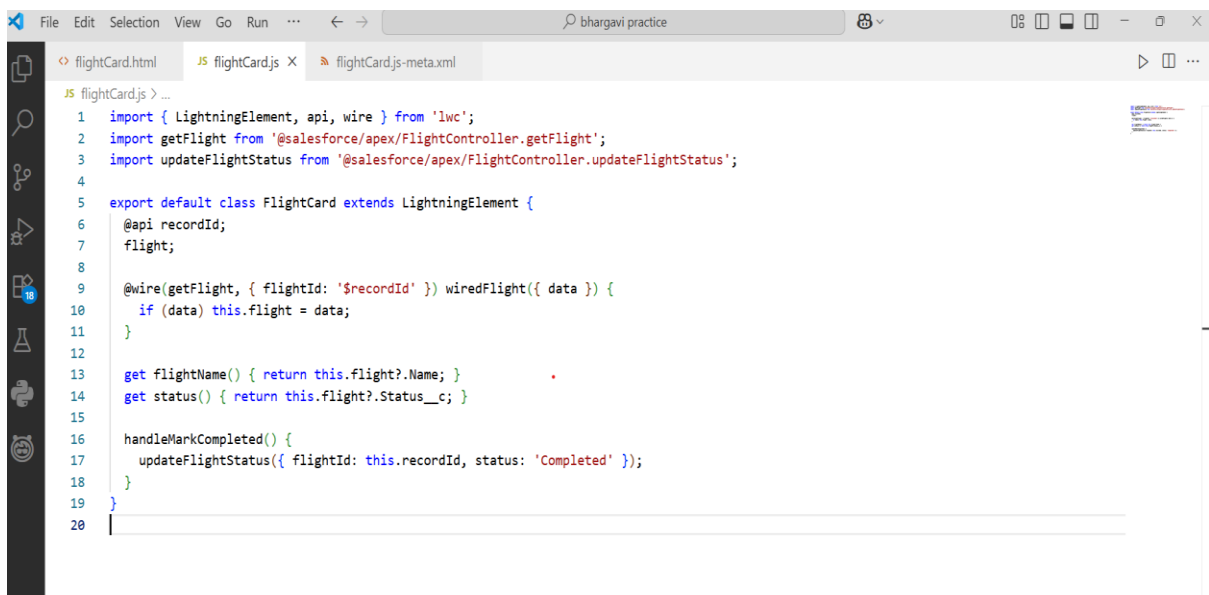


```

1 <template>
2   <lightning-card title={flightName}>
3     <p>Status: {status}</p>
4     <lightning-button label="Mark Completed" onclick={handleMarkCompleted}></lightning-button>
5   </lightning-card>
6 </template>
7

```

- flightCard.js

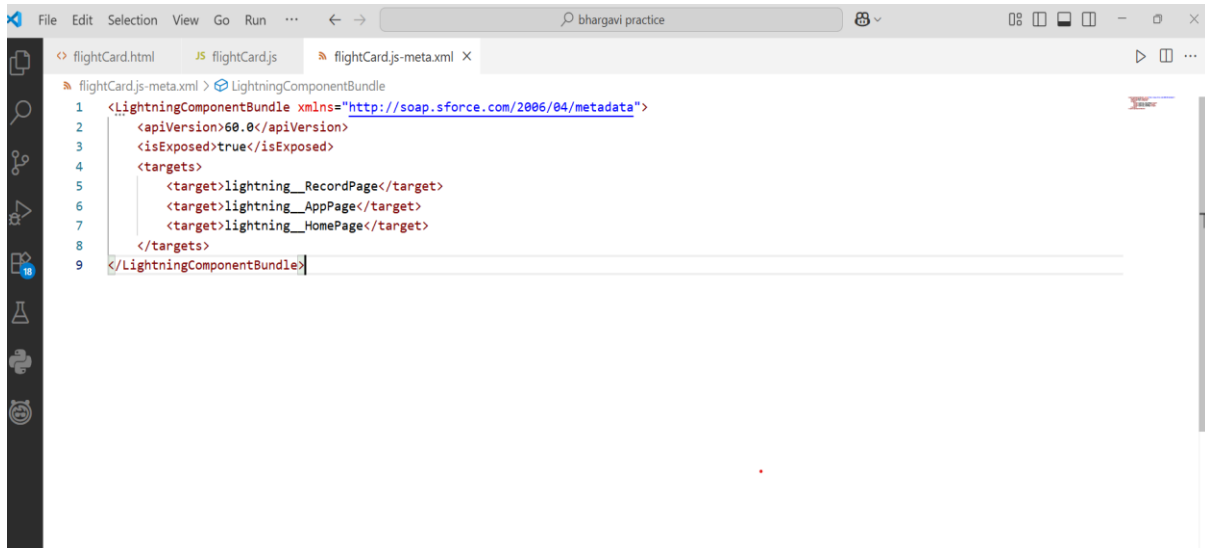


```

1 import { LightningElement, api, wire } from 'lwc';
2 import getFlight from '@salesforce/apex/FlightController.getFlight';
3 import updateFlightStatus from '@salesforce/apex/FlightController.updateFlightStatus';
4
5 export default class FlightCard extends LightningElement {
6   @api recordId;
7   flight;
8
9   @wire(getFlight, { flightId: '$recordId' }) wiredFlight({ data }) {
10    if (data) this.flight = data;
11  }
12
13  get flightName() { return this.flight?.Name; }
14  get status() { return this.flight?.Status__c; }
15
16  handleMarkCompleted() {
17    updateFlightStatus({ flightId: this.recordId, status: 'Completed' });
18  }
19 }
20

```

- FlightCard.js-meta.xml



## Step 7: Create Apex Controller

- Setup → Apex Classes → New.



## 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.