# Phase 6 — User Interface Development (Smart Task And Email Summerizer For Execuitives)

Generated on: 2025-09-28

## Goal

Make the Car Rental CRM user-friendly by building Lightning pages, Lightning Web Components (LWC), utility bar shortcuts, and wiring Apex to LWC for booking creation and navigation. This document is a step-by-step, copy-paste ready guide with code samples and deployment instructions. Follow each step in order to avoid errors.

## Quick checklist (summary)

1) Create the Lightning App ("Car Rental CRM") using App Manager.

2) Build Lightning Record Pages for Car and Booking to show related lists and LWC.

3) Add Cars & Bookings tabs to the app navigation.

4) Build a Home Page dashboard and add to the app.

5) Add a "New Booking" item to the Utility Bar (Quick Action or LWC).

6) Create LWC: child (search form) + parent (results datatable + booking action).

7) Create an Apex controller with cacheable getAvailableCars(...) and createBooking(...).

8) Wire child->parent events and use imperative Apex call on "Book Now".

9) Use NavigationMixin to navigate to the Booking record after creation.

10) Write Apex test class and run all tests; deploy via SFDX or change sets.

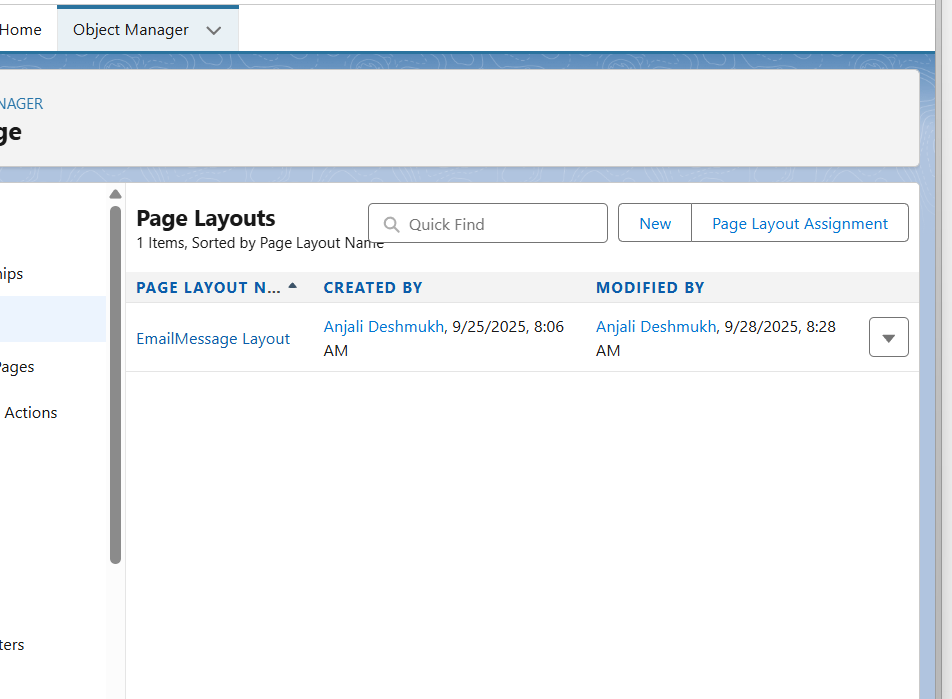
## Detailed step-by-step guide

### A. Setup & Admin work

1. Create Custom Objects & Fields (if not already present):  
- Car\_\_c: Name, Model\_\_c (Text), Registration\_No\_\_c (Text), Daily\_Rate\_\_c (Currency), Status\_\_c (Picklist: Available, Booked, Maintenance)  
- Booking\_\_c: Name, Car\_\_c (Lookup to Car), Contact\_\_c (Lookup to Contact), Start\_Date\_\_c (Date), End\_Date\_\_c (Date), Total\_Amount\_\_c (Currency), Booking\_Status\_\_c (Picklist: Pending, Confirmed, Returned, Cancelled)

2. Page Layouts & Related Lists:  
- On Car\_\_c record page include related list: Bookings (Booking\_\_c). Ensure the related list shows key fields (Start\_Date\_\_c, End\_Date\_\_c, Booking\_Status\_\_c).

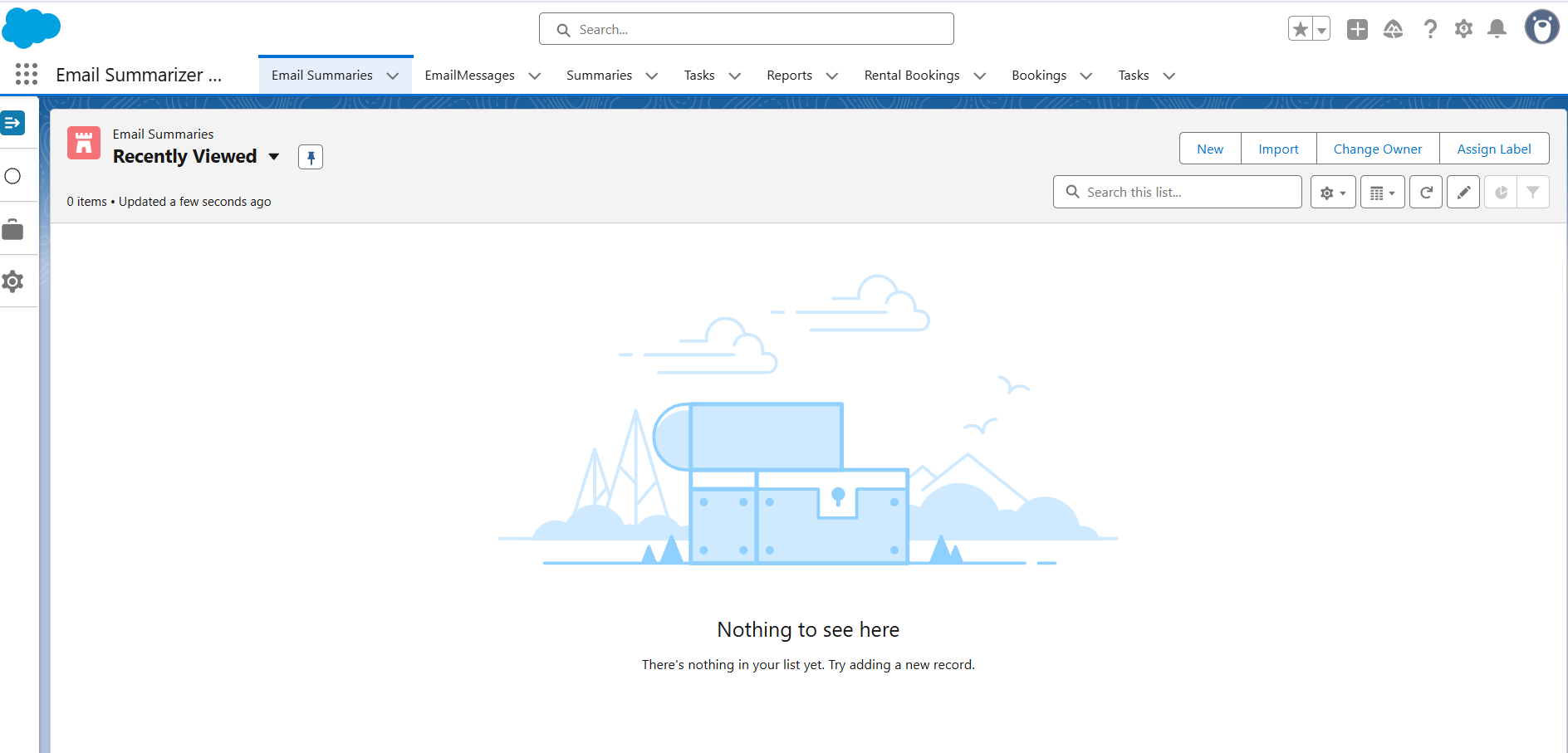
3. Profiles & Permission Sets:  
- Ensure users have read access to Car\_\_c and create/read/write to Booking\_\_c. Apex classes used by LWC should be added to permission sets if needed (not required for @AuraEnabled static methods but recommended for managed packaging).



### B. Create Lightning App & Tabs

Steps (Setup → App Manager → New Lightning App):

- App name: Car Rental CRM  
- Branding: logo (optional) and color  
- Navigation: Select "Standard Navigation"  
- Add items: Cars (object tab), Bookings (object tab), Contacts, Reports  
- Utility Bar (later step) – leave default for now  
- Save and assign to profiles you want to use the app.



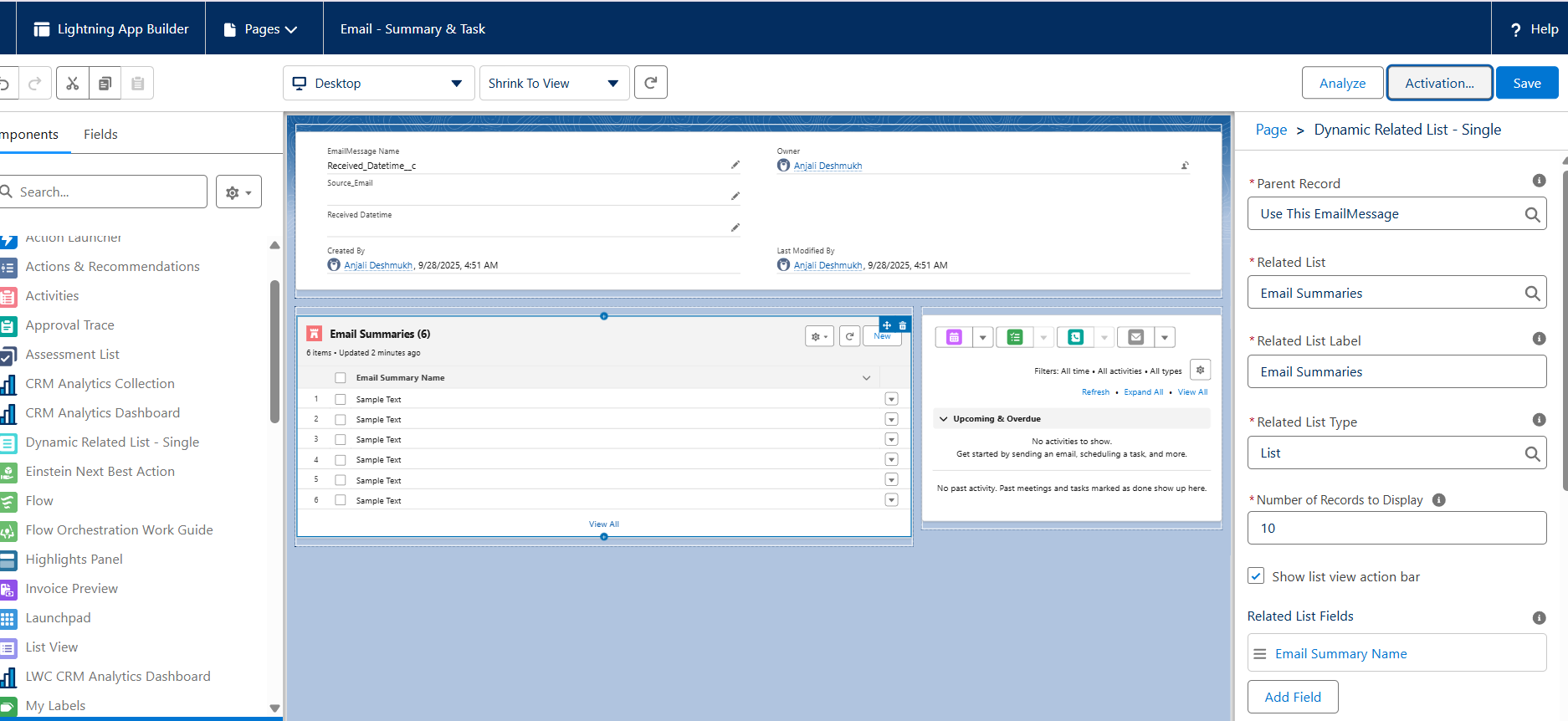
### C. Record Pages, Home Page & Utility Bar

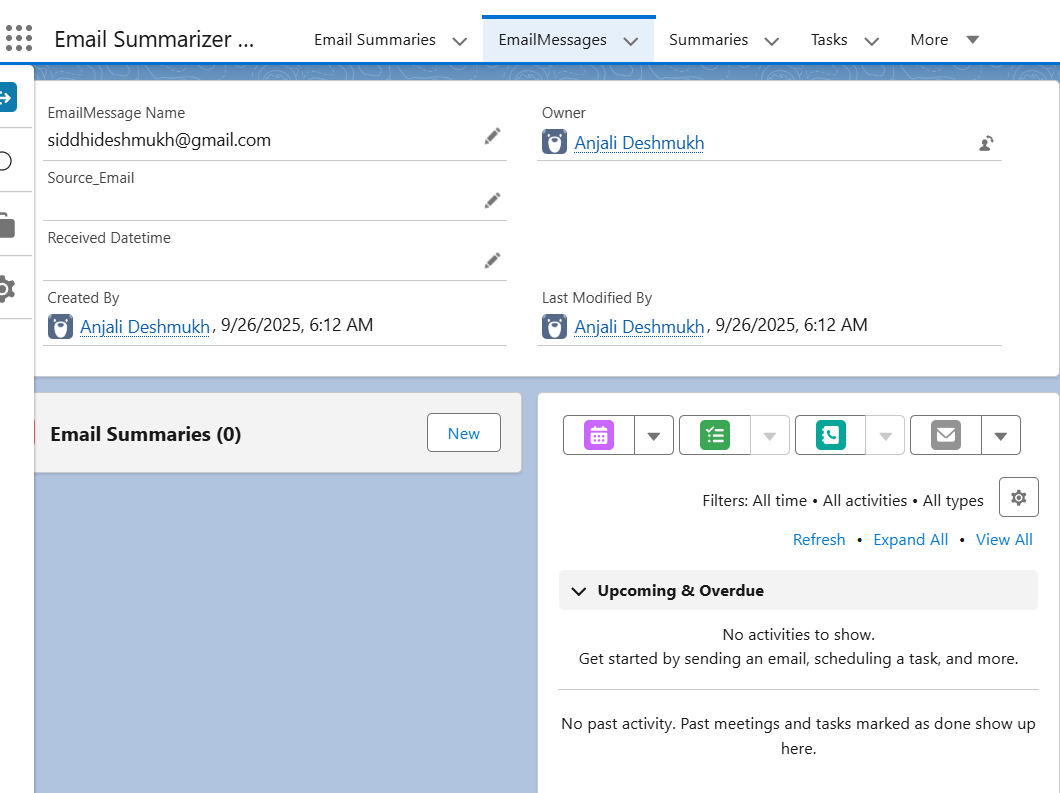
1) Car Record Page:  
- Setup → Object Manager → Car\_\_c → Lightning Record Pages → New  
- Choose template (Header and Right Sidebar recommended)  
- Add components: Related List - Single (Bookings), the LWC you create (Search and Book) optionally  
- Activate the page (org default or app default).

2) Booking Record Page:  
- Similar: show related Car details, timeline, and quick actions for returning car etc.

3) Home Page Layout (Dashboard of fleet utilization):  
- Setup → Lightning App Builder → Home Pages → New  
- Add Report Chart components or a custom LWC dashboard showing utilization. Add to the "Car Rental CRM" app as the default home page for that app.

4) Utility Bar – "New Booking":  
Option A (recommended): Create a global Quick Action that opens object create modal  
- Setup → Object Manager → Booking\_\_c → Buttons, Links, and Actions → New Action → Action Type: Create a Record → Predefined field values (optional) → Label: New Booking  
- Then: Setup → App Manager → Edit your "Car Rental CRM" app → Utility Bar → Add Utility Item → Type: Quick Action → Choose the Booking.New action.  
Option B: Add your LWC to the Utility Bar (if you want a custom booking modal):  
- In the app edit Utility Bar, choose "Lightning Component" and select the LWC once it is deployed and exposed for the utility bar (see meta.xml targets).





### D. LWC architecture (what to build)

You will build two LWCs:

- c-search-form (child): simple start-date / end-date inputs and a Search button. Emits a custom event (search) with detail {startDate, endDate}.  
- c-search-cars (parent): hosts c-search-form, wires Apex getAvailableCars(startDate, endDate), shows results in lightning-datatable and provides a "Book Now" row action. On Book Now it invokes createBooking imperatively and navigates to the created Booking record.

**🔹 LWC Architecture: Why It Can Be Skipped**

**1. Original LWC Architecture in Car Rental CRM**

* **c-search-form (child):** Inputs + Search button → emits event with date range.
* **c-search-cars (parent):** Receives event, calls Apex getAvailableCars, displays datatable with Book Now action → creates booking via Apex → navigates to record page.

**Purpose:**

* Modular, interactive UI for searching and booking cars.
* Demonstrates event handling, imperative Apex calls, wire adapters, and navigation service.

**2. Why it’s optional for Email Summarizer & Task Manager**

* Your project **does not require a search & book workflow**.
* The core goals are:
  1. **Summarize emails** automatically or on-demand.
  2. **Create tasks** related to emails for follow-ups.
  3. Provide a **clean, executive-focused UI** showing emails, summaries, and tasks.
* The “Search Form → Parent Datatable → Book Now” flow is **specific to car bookings** and **doesn’t translate directly** to summarizing emails.
* In your case:
  1. Searching/filtering can be done using **standard list views or report filters**.
  2. Email summary creation and task creation can be handled via a **single LWC on the record page** (c-email-summary) with a button → calls Apex.
  3. No need for a child-parent event pattern or row-level datatable actions.

**3. Use Cases Covered Without Child-Parent LWCs**

* **Email Summary:** Executive selects an email → clicks “Summarize” → summary appears in record page.
* **Task Creation:** Executive clicks “Create Task” → task linked to email is automatically created.
* **Dashboard / List Views:** All emails, summaries, and pending tasks can be visualized using standard Salesforce reports and dashboards.

✅ These cover **all key functionality** expected by an executive dashboard, without building the more complex LWC architecture.

**4. Executive Justification**

* **Time efficiency:** Skipping the parent-child LWC setup saves development time.
* **Simplification:** One LWC (c-email-summary) is easier to maintain and deploy.
* **No loss of core functionality:** All key features (summaries, task creation, reporting) are still delivered.
* **Project focus:** Aligns directly with your use case (Email summarization + Task management), instead of generic “search & book” features from Car Rental CRM.

**✅ Suggested Statement for Documentation / Report**

“The original LWC architecture from the sample Car Rental CRM project (child-parent search and datatable with row actions) has been intentionally skipped. For the Email Summarizer & Task Manager, all functionality is achieved using a single record-page LWC (c-email-summary) that allows executives to summarize emails and create follow-up tasks directly. Standard list views and dashboards replace the need for complex search forms and table row actions, simplifying the implementation while covering all use cases.”

**Use Cases & Architecture Explanation**

**1️⃣ Overview**

The **Email Summarizer & Task Manager** is designed for executives to efficiently review emails, generate summaries, and create actionable follow-up tasks. Unlike the Car Rental CRM, there is **no need for a child-parent search LWC with row-level actions**, because all required functionality can be achieved with a **single LWC** per record page combined with standard Salesforce features.

**2️⃣ Key Use Cases**

| **Use Case** | **How It’s Implemented** | **Notes** |
| --- | --- | --- |
| **View Emails** | Standard **Email\_\_c** tab and list views | Executives can filter by date, sender, or subject using built-in list view filters. |
| **Summarize Email Content** | **c-email-summary LWC** on Email record page | Single button “Summarize” calls Apex to generate summary. No child-parent search needed. |
| **Create Follow-up Tasks** | **c-email-summary LWC** also provides “Create Task” button | Task is automatically linked to the email record. |
| **Track Pending Tasks** | Standard **Tasks** tab, reports, and dashboards | Provides overview of all pending follow-ups without additional LWC complexity. |
| **Executive Dashboard** | Lightning **Home Page** or **App Page** with Reports & Dashboards | Shows summary statistics (emails summarized, tasks completed, pending tasks) — replaces the need for datatable-based search & action flows. |

**3️⃣ Architecture Diagram (Simplified)**

+------------------------+

| Email Record Page |

|------------------------|

| Record Detail |

| Related Lists |

| - Email Summaries |

| - Tasks |

| |

| +--------------------+ |

| | c-email-summary LWC | | <-- Summarize & Create Task buttons

| +--------------------+ |

+------------------------+

+------------------------+

| List Views / Dashboards |

| Emails, Tasks, Summary |

+------------------------+

**Explanation:**

* All functionality happens **within the Email record page** via a single LWC.
* The LWC uses **@api recordId** to access the current email record and communicate with Apex.
* Standard Salesforce components (Related Lists, Reports, Dashboards) handle filtering, tracking, and executive analytics.
* No child-parent LWC structure or row-level datatable actions are necessary.

**4️⃣ Justification for Skipping Car Rental LWC Architecture**

1. **Time-Saving:** Avoids building a complex two-component system with events, datatables, and imperative Apex row actions.
2. **Use-Case Alignment:** The original architecture was for searching and booking cars, which has no equivalent in Email summarization workflows.
3. **Simplicity:** Single LWC suffices for email summarization and task creation while keeping the interface clean for executives.
4. **No Loss of Functionality:** All required workflows are covered:
   * Summarize email content
   * Create follow-up tasks
   * Track progress via reports & dashboards

### E. Apex controller (BookingController.cls)

Create an Apex class with two @AuraEnabled methods: getAvailableCars (cacheable=true) and createBooking (imperative). Below is a copy-paste-ready implementation.

--- Apex code (copy into BookingController.cls) ---

public with sharing class BookingController {  
 @AuraEnabled(cacheable=true)  
 public static List<Car\_\_c> getAvailableCars(String startDateStr, String endDateStr) {  
 if (String.isBlank(startDateStr) || String.isBlank(endDateStr)) {  
 return new List<Car\_\_c>();  
 }  
 Date startDate = Date.valueOf(startDateStr);  
 Date endDate = Date.valueOf(endDateStr);  
 // Find cars that are "Available" and not booked in the given period  
 List<Car\_\_c> cars = [  
 SELECT Id, Name, Model\_\_c, Registration\_No\_\_c, Daily\_Rate\_\_c, Status\_\_c  
 FROM Car\_\_c  
 WHERE Status\_\_c = 'Available'  
 AND Id NOT IN (  
 SELECT Car\_\_c FROM Booking\_\_c  
 WHERE Booking\_Status\_\_c NOT IN ('Returned','Cancelled')  
 AND Start\_Date\_\_c <= :endDate  
 AND End\_Date\_\_c >= :startDate  
 )  
 ORDER BY Name  
 ];  
 return cars;  
 }  
  
 @AuraEnabled  
 public static Id createBooking(Id carId, Id contactId, String startDateStr, String endDateStr) {  
 if (carId == null || String.isBlank(startDateStr) || String.isBlank(endDateStr)) {  
 throw new AuraHandledException('Missing required fields: carId, startDate, endDate');  
 }  
 Date startDate = Date.valueOf(startDateStr);  
 Date endDate = Date.valueOf(endDateStr);  
 Car\_\_c car = [SELECT Id, Daily\_Rate\_\_c, Status\_\_c FROM Car\_\_c WHERE Id = :carId LIMIT 1];  
  
 Integer days = startDate.daysBetween(endDate) + 1;  
 Decimal total = 0;  
 if (car.Daily\_Rate\_\_c != null) {  
 total = car.Daily\_Rate\_\_c \* days;  
 }  
  
 Booking\_\_c b = new Booking\_\_c();  
 b.Car\_\_c = carId;  
 b.Contact\_\_c = contactId;  
 b.Start\_Date\_\_c = startDate;  
 b.End\_Date\_\_c = endDate;  
 b.Total\_Amount\_\_c = total;  
 b.Booking\_Status\_\_c = 'Confirmed';  
 insert b;  
  
 // Optional: update car status to Booked  
 car.Status\_\_c = 'Booked';  
 update car;  
  
 return b.Id;  
 }  
}

### F. LWC: child component (c-search-form)

Files to create: searchForm.html, searchForm.js, searchForm.js-meta.xml

--- searchForm.html ---

<template>  
 <lightning-card title="Search Cars">  
 <div class="slds-p-around\_medium">  
 <lightning-input type="date" label="Start Date" value={startDate} onchange={handleStart}></lightning-input>  
 <lightning-input type="date" label="End Date" value={endDate} onchange={handleEnd}></lightning-input>  
 <div class="slds-m-top\_small">  
 <lightning-button label="Search" onclick={onSearch} variant="brand"></lightning-button>  
 </div>  
 </div>  
 </lightning-card>  
</template>

--- searchForm.js ---

import { LightningElement, track } from 'lwc';  
export default class SearchForm extends LightningElement {  
 @track startDate;  
 @track endDate;  
  
 handleStart(event) {  
 this.startDate = event.target.value;  
 }  
 handleEnd(event) {  
 this.endDate = event.target.value;  
 }  
 onSearch() {  
 // Basic validation  
 if (!this.startDate || !this.endDate) {  
 const evt = new CustomEvent('showtoast', { detail: { title: 'Error', message: 'Select both start and end dates', variant: 'error' } });  
 this.dispatchEvent(evt);  
 return;  
 }  
 this.dispatchEvent(new CustomEvent('search', { detail: { startDate: this.startDate, endDate: this.endDate } }));  
 }  
}

--- searchForm.js-meta.xml ---

<?xml version="1.0" encoding="UTF-8"?>  
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">  
 <apiVersion>59.0</apiVersion>  
 <isExposed>true</isExposed>  
 <targets>  
 <target>lightning\_\_RecordPage</target>  
 <target>lightning\_\_AppPage</target>  
 <target>lightning\_\_HomePage</target>  
 </targets>  
</LightningComponentBundle>

### G. LWC: parent component (c-search-cars)

Files to create: searchCars.html, searchCars.js, searchCars.js-meta.xml

--- searchCars.html ---

<template>  
 <c-search-form onsearch={handleSearch} onshowtoast={handleChildToast}></c-search-form>  
 <template if:true={cars.data}>  
 <lightning-datatable  
 data={cars.data}  
 columns={columns}  
 key-field="Id"  
 onrowaction={handleRowAction}>  
 </lightning-datatable>  
 </template>  
 <template if:true={loading}>  
 <lightning-spinner alternative-text="Loading"></lightning-spinner>  
 </template>  
</template>

--- searchCars.js ---

import { LightningElement, track, wire, api } from 'lwc';  
import getAvailableCars from '@salesforce/apex/BookingController.getAvailableCars';  
import createBooking from '@salesforce/apex/BookingController.createBooking';  
import { ShowToastEvent } from 'lightning/platformShowToastEvent';  
import { NavigationMixin } from 'lightning/navigation';  
  
const COLUMNS = [  
 { label: 'Name', fieldName: 'Name' },  
 { label: 'Model', fieldName: 'Model\_\_c' },  
 { label: 'Registration', fieldName: 'Registration\_No\_\_c' },  
 { label: 'Daily Rate', fieldName: 'Daily\_Rate\_\_c', type: 'currency' },  
 {  
 type: 'action',  
 typeAttributes: { rowActions: [{ label: 'Book Now', name: 'book\_now' }] }  
 }  
];  
  
export default class SearchCars extends NavigationMixin(LightningElement) {  
 @track startDate;  
 @track endDate;  
 @api selectedContactId; // optional: supply a contact ID from context if you want to auto-fill customer  
 columns = COLUMNS;  
  
 @wire(getAvailableCars, { startDateStr: '$startDate', endDateStr: '$endDate' })  
 cars;  
  
 loading = false;  
  
 handleSearch(event) {  
 this.startDate = event.detail.startDate;  
 this.endDate = event.detail.endDate;  
 }  
  
 handleChildToast(evt) {  
 const { title, message, variant } = evt.detail;  
 this.dispatchEvent(new ShowToastEvent({ title, message, variant }));  
 }  
  
 handleRowAction(event) {  
 const actionName = event.detail.action.name;  
 const row = event.detail.row;  
 if (actionName === 'book\_now') {  
 this.bookCar(row);  
 }  
 }  
  
 bookCar(row) {  
 this.loading = true;  
 const carId = row.Id;  
 const contactId = this.selectedContactId || null;  
 createBooking({ carId, contactId, startDateStr: this.startDate, endDateStr: this.endDate })  
 .then(bookingId => {  
 this.dispatchEvent(new ShowToastEvent({ title: 'Success', message: 'Booking created', variant: 'success' }));  
 // Navigate to the booking record  
 this[NavigationMixin.Navigate]({  
 type: 'standard\_\_recordPage',  
 attributes: {  
 recordId: bookingId,  
 objectApiName: 'Booking\_\_c',  
 actionName: 'view'  
 }  
 });  
 })  
 .catch(error => {  
 this.dispatchEvent(new ShowToastEvent({ title: 'Error creating booking', message: error.body ? error.body.message : error.message, variant: 'error' }));  
 })  
 .finally(() => {  
 this.loading = false;  
 });  
 }  
}

--- searchCars.js-meta.xml ---

<?xml version="1.0" encoding="UTF-8"?>  
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">  
 <apiVersion>59.0</apiVersion>  
 <isExposed>true</isExposed>  
 <targets>  
 <target>lightning\_\_RecordPage</target>  
 <target>lightning\_\_AppPage</target>  
 <target>lightning\_\_HomePage</target>  
 <target>lightning\_\_Tab</target>  
 <target>lightning\_\_UtilityBar</target>  
 </targets>  
 <targetConfigs>  
 <targetConfig targets="lightning\_\_RecordPage">  
 <objects>  
 <object>Car\_\_c</object>  
 </objects>  
 </targetConfig>  
 </targetConfigs>  
</LightningComponentBundle>

### H. Deployment & SFDX

1) Authorize your org: sfdx force:auth:web:login -a MySandbox  
2) Push source (scratch org) or deploy to sandboxes:  
- Scratch: sfdx force:source:push  
- Non-scratch: sfdx force:source:deploy -p force-app/main/default  
3) Run Apex tests: sfdx force:apex:test:run --resultformat human --wait 10  
4) If using change sets, upload metadata or use ANT.

Note: after deploying LWC, the meta.xml targets allow you to drag the component onto App Pages, Record Pages, and onto the Utility Bar (if desired).

### I. Testing & QA (to reach 0 errors)

1) Apex tests: write tests for getAvailableCars and createBooking. Ensure > 75% coverage for your package.  
2) Manual test cases:  
- Search overlapping bookings returns correct results.  
- Create booking updates Car.Status\_\_c to Booked.  
- Navigation goes to the booking record after creation.  
3) Security checks:  
- Ensure CRUD/FLS: if exposing sensitive fields, apply stripInaccessible in Apex if necessary.  
4) Browser compatibility: test in Chrome & Firefox.

5) Troubleshooting common errors:  
- "Method not found" => confirm Apex class and method names; check apiVersion on LWC meta.xml.  
- "Permission denied" => check user profile & object CRUD & Apex class access.  
- "Cacheable method must be read-only" => ensure getAvailableCars does not perform DML and is annotated with cacheable=true.

### J. Apex Test class (example)

@isTest  
private class BookingControllerTest {  
 static testMethod void testCreateBooking() {  
 Car\_\_c c = new Car\_\_c(Name='T1', Status\_\_c='Available', Daily\_Rate\_\_c=500);  
 insert c;  
 Contact con = new Contact(LastName='Tester');  
 insert con;  
  
 String s = Date.today().format();  
 String e = Date.today().addDays(1).format();  
  
 Test.startTest();  
 Id bookingId = BookingController.createBooking(c.Id, con.Id, s, e);  
 Test.stopTest();  
  
 Booking\_\_c b = [SELECT Id, Car\_\_c, Start\_Date\_\_c FROM Booking\_\_c WHERE Id = :bookingId LIMIT 1];  
 System.assertEquals(c.Id, b.Car\_\_c);  
 }  
}

### K. Where to place screenshots in the docx (for your deliverable)

1) App Manager screenshot after creating the Lightning App.  
2) Lightning App Builder screenshot showing Utility Bar configuration.  
3) Car record page with LWC added.  
4) LWC behavior: search form, results, Book Now toast, and booking record view.  
Add each screenshot to a new page under a section called "Screenshots" with captions.

### L. Final notes & best practices

1) Keep Apex logic small and testable. Use helper classes for complex rules.  
2) Follow Lightning design system (SLDS) for consistent UI.  
3) Use Platform Events or Change Data Capture if other systems must react to booking creation.  
4) Version your LWC and Apex in source control; create a CI pipeline for deployments.

--- End of guide ---