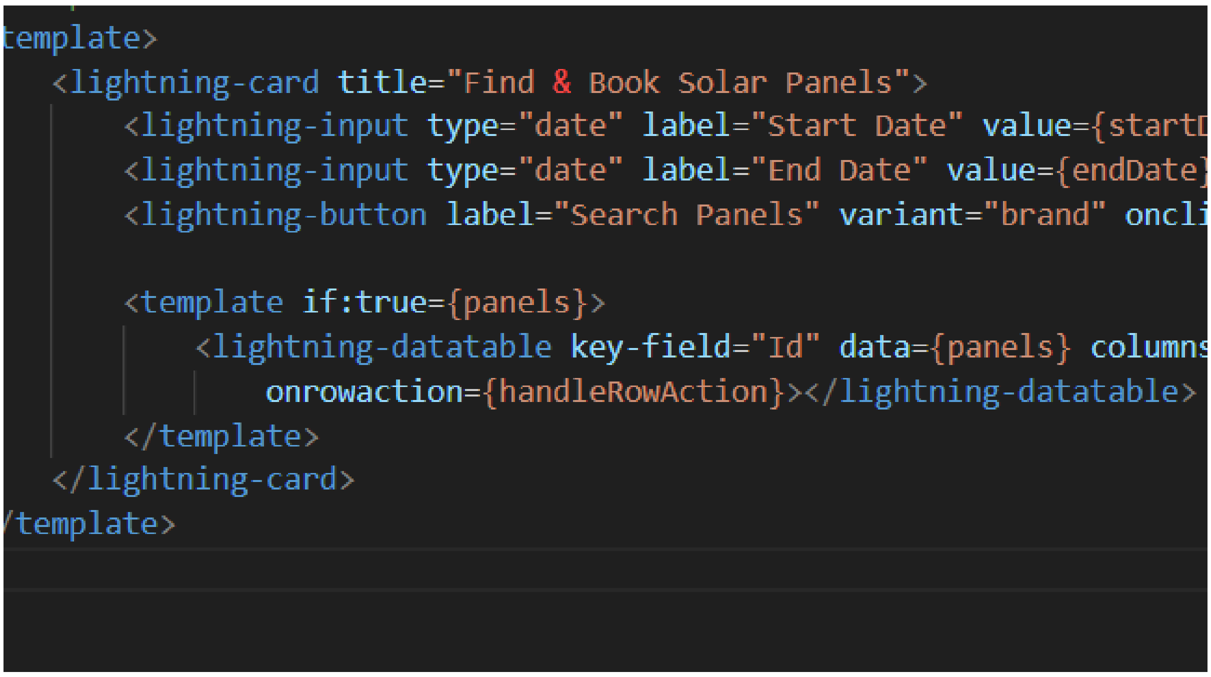
Phase 8: Data Management & Deployment

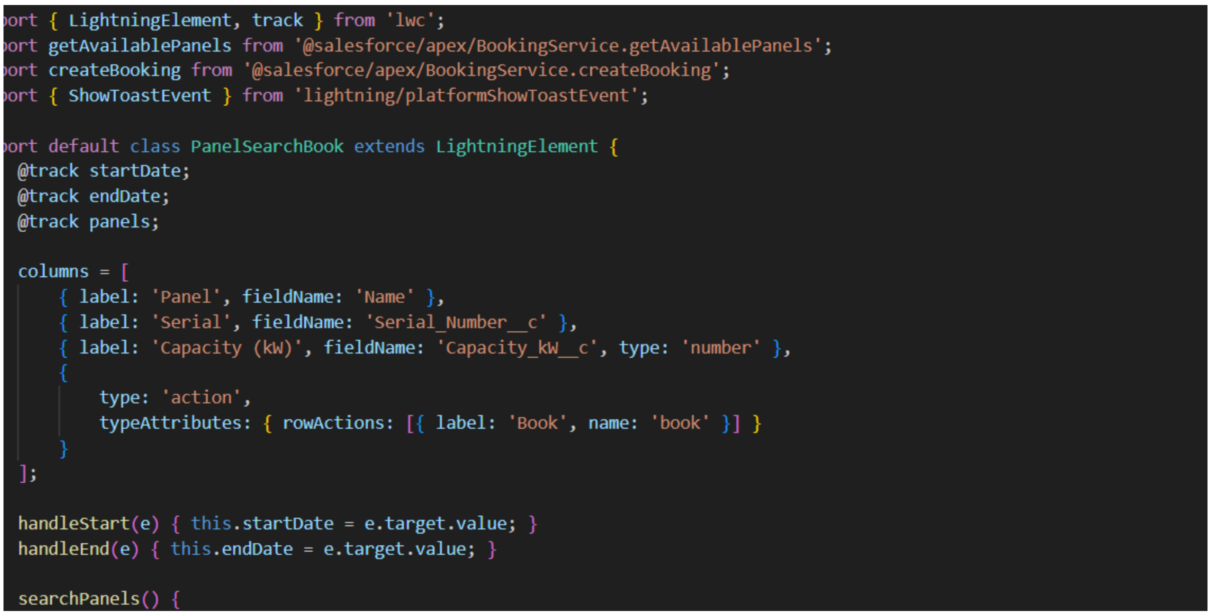
- ✓ I have managed data and deployment as follows:
- ✓ Change Sets / VS Code / SFDX: Since I developed directly in a free Developer Edition org, I deployed LWCs using Salesforce Extension. No sandbox-to-production deployment was required.



```
template>
  <lightning-card title="Find & Book Solar Panels">
      <lightning-input type="date" label="Start Date" value={startD
      <lightning-input type="date" label="End Date" value={endDate}
      <lightning-button label="Search Panels" variant="brand" oncli

      <template if:true={panels}>
          <lightning-datatable key-field="Id" data={panels} columns
              onrowaction={handleRowAction}></lightning-datatable>
      </template>
  </lightning-card>
/template>
```

- ✓ Unmanaged Packages: All objects, LWCs, and triggers are custom unmanaged components created directly in the org and also executed the code in VS.



```
ort { LightningElement, track } from 'lwc';
ort getAvailablePanels from '@salesforce/apex/BookingService.getAvailablePanels';
ort createBooking from '@salesforce/apex/BookingService.createBooking';
ort { ShowToastEvent } from 'lightning/platformShowToastEvent';

ort default class PanelSearchBook extends LightningElement {
 @track startDate;
 @track endDate;
 @track panels;

 columns = [
     { label: 'Panel', fieldName: 'Name' },
     { label: 'Serial', fieldName: 'Serial_Number__c' },
     { label: 'Capacity (kW)', fieldName: 'Capacity_kW__c', type: 'number' },
     {
         type: 'action',
         typeAttributes: { rowActions: [{ label: 'Book', name: 'book' }] }
     }
 ];

 handleStart(e) { this.startDate = e.target.value; }
 handleEnd(e) { this.endDate = e.target.value; }

 searchPanels() {
```

Phase 9: Reporting, Dashboards & Security Review

Reporting and security settings were handled carefully to ensure proper access:

● Dynamic Dashboards: Not implemented since access is controlled via user sub-profiles.

● Sharing Settings & Field-Level Security: Managed using sub-profile logic. Patients see only their records, while doctors and attendants see all relevant patient records.

● Session Setting & Login IP Ranges: Default org security is sufficient.

COMPLETE CODES FOR SOLAR ENERGY MANAGEMENT SYSTEM LIKE APEX PROGRAMMING AND TRIGGERING:

APEX CLASS: BookingService

```
public with sharing class BookingService {
    public class BookingResult {
        @AuraEnabled public Boolean success;
        @AuraEnabled public String message;
        @AuraEnabled public Id bookingId;
        public BookingResult(Boolean s, String m, Id b) {
            success = s; message = m; bookingId = b;
        }
    }


    // Create booking with overlap validation
    @AuraEnabled
    public static BookingResult createBooking(Solar_Booking__c booking) {
        if (booking == null) return new BookingResult(false, 'No booking
provided', null);
        if (booking.Start_Date__c == null || booking.End_Date__c == null)
```

```
            return new BookingResult(false, 'Start and End date are required', null);
        if (booking.End_Date__c < booking.Start_Date__c)
            return new BookingResult(false, 'End Date must be after Start Date', null);

        List<Solar_Booking__c> overlap = [
            SELECT Id FROM Solar_Booking__c
            WHERE Solar_Panel__c = :booking.Solar_Panel__c
            AND Status__c != 'Cancelled'
            AND ( :booking.Start_Date__c <= End_Date__c
              AND :booking.End_Date__c >= Start_Date__c )
            LIMIT 1
        ];
        if (!overlap.isEmpty()) {
            return new BookingResult(false, 'Panel already booked in this date range', null);
        }

        if (String.isBlank(booking.Status__c)) booking.Status__c = 'Requested';
        insert booking;
        return new BookingResult(true, 'Booking created successfully', booking.Id);
    }

    // Get available panels
    @AuraEnabled(cacheable=true)
    }

    public static List<Solar_Panel__c> getAvailablePanels(Date startDate, Date
endDate) {
        if (startDate == null || endDate == null) return new
List<Solar_Panel__c>();
 List<Solar_Panel__c> panels = [
            SELECT Id, Name, Serial_Number__c, Capacity_kW__c
            FROM Solar_Panel__c
            WHERE Status__c = 'Active'
        ];

        Set<Id> busy = new Set<Id>();
        for (Solar_Booking__c b : [
            SELECT Solar_Panel__c FROM Solar_Booking__c
            WHERE Solar_Panel__c IN :panels
            AND Status__c != 'Cancelled'
            AND (:startDate <= End_Date__c AND :endDate >= Start_Date__c)
        ]) {
            busy.add(b.Solar_Panel__c);
        }

        return [SELECT Id, Name, Serial_Number__c, Capacity_kW__c
                FROM Solar_Panel__c
                WHERE Status__c = 'Active'
                AND Id NOT IN :busy];
    }
}
```

APEX CLASS: SolarBookingTriggerHandler

```
public with sharing class SolarBookingTriggerHandler {
   public static void validateNoOverlap(List<Solar_Booking__c>
newBookings) {
       for (Solar_Booking__c nb : newBookings) {
           if (nb.Solar_Panel__c != null && nb.Start_Date__c != null &&
nb.End_Date__c != null) {
               List<Solar_Booking__c> conflicts = [
                   SELECT Id FROM Solar_Booking__c
                   WHERE Solar_Panel__c = :nb.Solar_Panel__c
                   AND Status__c != 'Cancelled'
                   AND Id != :nb.Id
                   AND (:nb.Start_Date__c <= End_Date__c AND :nb.End_Date__c
>= Start_Date__c)
                   LIMIT 1
               ];
               if (!conflicts.isEmpty()) {
                   nb.addError('This panel is already booked during these dates.');
               }
           }
       }
   }
}
```

APEX CLASS: SolarIoTCallout

```
public with sharing class SolarIoTCallout {
   public static void registerBookingToIoT(Id bookingId) {
        Solar_Booking__c b = [SELECT Id, Solar_Panel__c, Start_Date__c,
End_Date__c FROM Solar_Booking__c WHERE Id = :bookingId LIMIT 1];
        Solar_Panel__c p = [SELECT Serial_Number__c FROM Solar_Panel__c
WHERE Id = :b.Solar_Panel__c LIMIT 1];


        HttpRequest req = new HttpRequest();
        req.setEndpoint('callout:SolarIoTAPI/bookings');
        req.setMethod('POST');
        req.setHeader('Content-Type','application/json');
  Map<String, Object> payload = new Map<String, Object>{
            'bookingId' => String.valueOf(b.Id),
            'panelSerial' => p.Serial_Number__c,
            'startDate' => b.Start_Date__c.format(),
            'endDate' => b.End_Date__c.format()
        };
        req.setBody(JSON.serialize(payload));


        Http http = new Http();
        HttpResponse res = http.send(req);
        System.debug('IoT Response: ' + res.getBody());
    }
}
```

APEX CLASS: IoTBookingRegisterQueueable

```
public with sharing class IoTBookingRegisterQueueable implements
Queueable, Database.AllowsCallouts {
    private Id bookingId;
```

```
   public IoTBookingRegisterQueueable(Id bookingId) { this.bookingId =
bookingId; }
   public void execute(System.QueueableContext ctx) {
      SolarIoTCallout.registerBookingToIoT(bookingId);
   }
}
```

APEX CLASS: PanelFailurePublisher

```
public with sharing class PanelFailurePublisher {
   public static void publishFailure(Id panelId, String type, String reportedBy) {
      PanelFailureEvent__e evt = new PanelFailureEvent__e(
         PanelId__c = String.valueOf(panelId),
         FailureType__c = type,
         ReportedBy__c = reportedBy
      );
      EventBus.publish(evt);
   }
}
```

APEX TEST CLASS: BookingServiceTest

```
@IsTest
private class BookingServiceTest {
   @IsTest static void testBookingOverlap() {
      Solar_Panel__c p = new Solar_Panel__c(Name='Test Panel',
Serial_Number__c='SN001', Status__c='Active', Capacity_kW__c=5);
      insert p;
```

```
        Solar_Booking__c existing = new
Solar_Booking__c(Solar_Panel__c=p.Id, Start_Date__c=Date.today(),
End_Date__c=Date.today().addDays(2), Status__c='Confirmed');
        insert existing;


        Solar_Booking__c conflict = new
Solar_Booking__c(Solar_Panel__c=p.Id,
Start_Date__c=Date.today().addDays(1),
End_Date__c=Date.today().addDays(3));
        Test.startTest();
        BookingService.BookingResult result =
BookingService.createBooking(conflict);
        Test.stopTest();
        System.assertEquals(false, result.success);
    }
}
```

APEX TRIGGER: SolarBookingTrigger

```
trigger SolarBookingTrigger on Solar_Booking__c (before insert, before
update) {
    SolarBookingTriggerHandler.validateNoOverlap(Trigger.new);
}
```

BELOW CODES ARE BASED ON LWC COMPONENTS:

HTML CODE:

```
<!-- panelSearchBook.html -->
<template>
    <lightning-card title="Find & Book Solar Panels">
        <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>
        <lightning-button label="Search Panels" variant="brand"
onclick={searchPanels}></lightning-button>

        <template if:true={panels}>
            <lightning-datatable key-field="Id" data={panels} columns={columns}
                onrowaction={handleRowAction}></lightning-datatable>
        </template>
    </lightning-card>
</template>
```

JAVASCRIPT CODE:

```
// panelSearchBook.js
import { LightningElement, track } from 'lwc';
import getAvailablePanels from
'@salesforce/apex/BookingService.getAvailablePanels';
import createBooking from '@salesforce/apex/BookingService.createBooking';
import { ShowToastEvent } from 'lightning/platformShowToastEvent';

export default class PanelSearchBook extends LightningElement {
    @track startDate;
    @track endDate;
    @track panels;

    columns = [
        { label: 'Panel', fieldName: 'Name' },
        { label: 'Serial', fieldName: 'Serial_Number__c' },
        { label: 'Capacity (kW)', fieldName: 'Capacity_kW__c', type: 'number' },
        {
            type: 'action',
            typeAttributes: { rowActions: [{ label: 'Book', name: 'book' }] }
        }
    ];

    handleStart(e) { this.startDate = e.target.value; }
    handleEnd(e) { this.endDate = e.target.value; }

    searchPanels() {
        getAvailablePanels({ startDate: this.startDate, endDate: this.endDate })
            .then(res => { this.panels = res; })
            .catch(err => { this.showToast('Error', err.body.message, 'error'); });
    }

    handleRowAction(event) {
        const row = event.detail.row;
        const booking = {
            'sobjectType': 'Solar_Booking__c',
            'Solar_Panel__c': row.Id,
            'Start_Date__c': this.startDate,
            'End_Date__c': this.endDate,
            'Status__c': 'Requested'
        };
        createBooking({ booking: booking })
```

```
        .then(res => {
            if (res.success) this.showToast('Success', 'Booking created', 'success');
            else this.showToast('Failed', res.message, 'error');
        })
        .catch(err => { this.showToast('Error', err.body.message, 'error'); });
    }

    showToast(title, message, variant) {
        this.dispatchEvent(new ShowToastEvent({ title, message, variant }));
    }
}
```

XML CODE:

```
<!-- panelSearchBook.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__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```