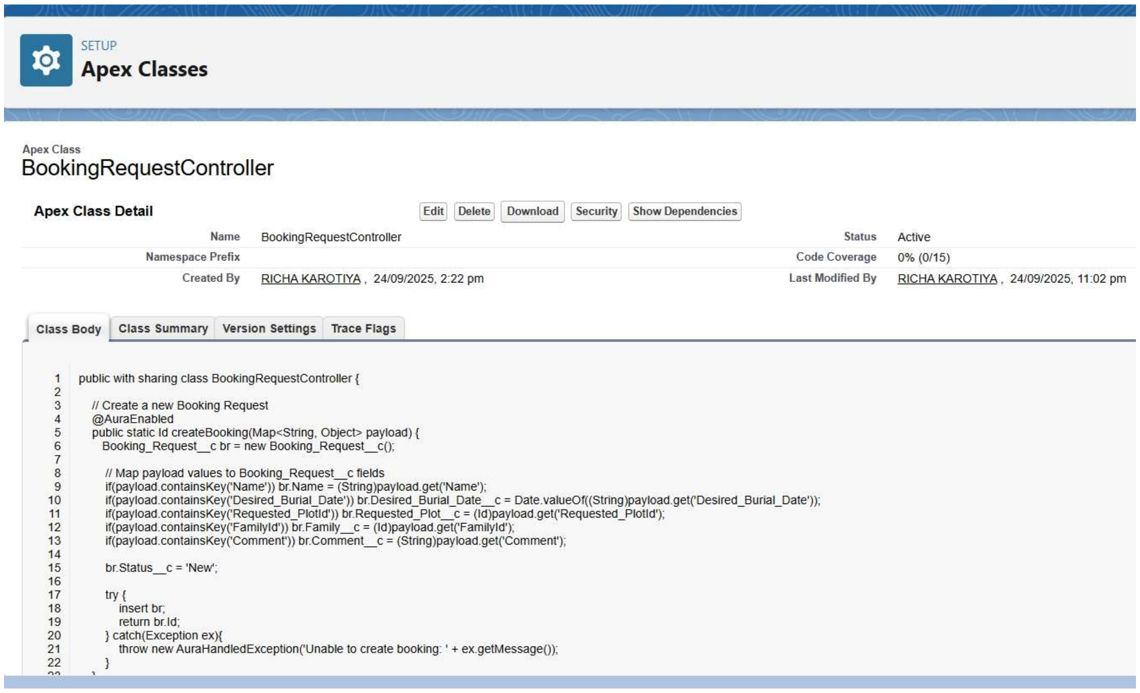
PHASE 5: Apex Programming

Goal: It is to enhance the Burial Booking System with custom logic and automation using Apex. It uses apex classes apex triggers etc.

1. Apex Classes

· Booking Request Controller - The BookingRequestController class is a custom Apex class designed to manage all backend logic related to Booking Requests in the system. It acts as a central controller to handle creation, validation, status updates, and notifications for booking requests submitted via the Flow or Experience Cloud portal.

· It basically handles the new booking request records by the families.

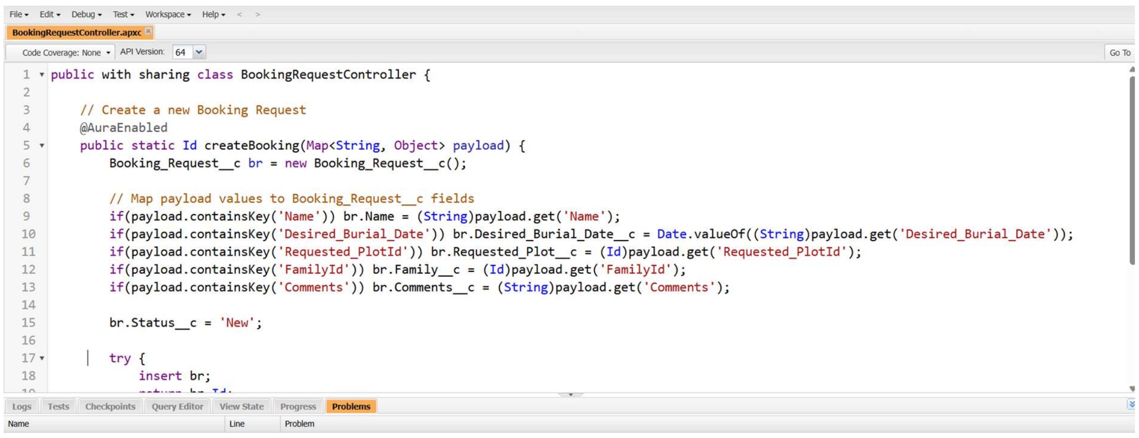


Apex Class
BookingRequestController

Apex Class Detail

Name	BookingRequestController	Status	Active
Namespace Prefix		Code Coverage	0% (0/15)
Created By	RICHA KAROTIYA , 24/09/2025, 2:22 pm	Last Modified By	RICHA KAROTIYA , 24/09/2025, 11:02 pm

```
public with sharing class BookingRequestController {

  // Create a new Booking Request
  @AuraEnabled
  public static Id createBooking(Map<String, Object> payload) {
    Booking_Request__c br = new Booking_Request__c();

    // Map payload values to Booking_Request__c fields
    if(payload.containsKey('Name')) br.Name = (String)payload.get('Name');
    if(payload.containsKey('Desired_Burial_Date')) br.Desired_Burial_Date__c = Date.valueOf((String)payload.get('Desired_Burial_Date'));
    if(payload.containsKey('Requested_PlotId')) br.Requested_Plot__c = (Id)payload.get('Requested_PlotId');
    if(payload.containsKey('FamilyId')) br.Family__c = (Id)payload.get('FamilyId');
    if(payload.containsKey('Comment')) br.Comment__c = (String)payload.get('Comment');

    br.Status__c = 'New';

    try {
      insert br;
      return br.Id;
    } catch(Exception ex){
      throw new AuraHandledException('Unable to create booking: ' + ex.getMessage());
    }
```



```
public with sharing class BookingRequestController {

    // Create a new Booking Request
    @AuraEnabled
    public static Id createBooking(Map<String, Object> payload) {
        Booking_Request__c br = new Booking_Request__c();

        // Map payload values to Booking_Request__c fields
        if(payload.containsKey('Name')) br.Name = (String)payload.get('Name');
        if(payload.containsKey('Desired_Burial_Date')) br.Desired_Burial_Date__c = Date.valueOf((String)payload.get('Desired_Burial_Date'));
        if(payload.containsKey('Requested_PlotId')) br.Requested_Plot__c = (Id)payload.get('Requested_PlotId');
        if(payload.containsKey('FamilyId')) br.Family__c = (Id)payload.get('FamilyId');
        if(payload.containsKey('Comments')) br.Comments__c = (String)payload.get('Comments');

        br.Status__c = 'New';

        try {
            insert br;
```

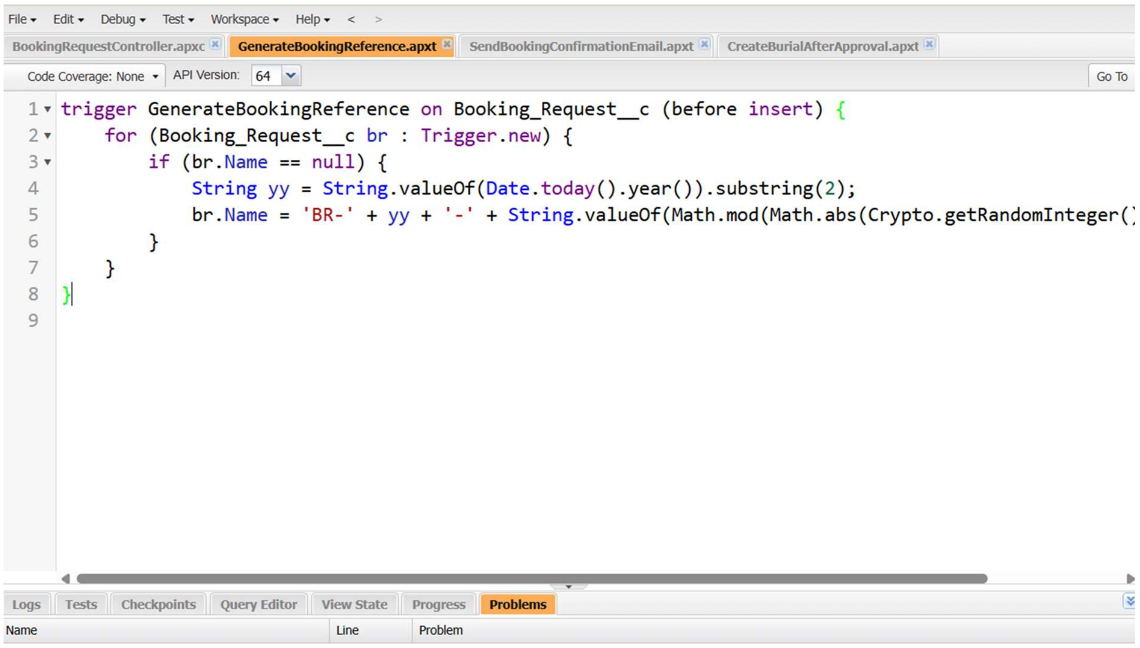
1. Apex Triggers

- Generate Booking Reference Trigger :

The GenerateBookingReference trigger automatically generates a unique booking reference number for each Booking_Request__c record before it is inserted. This ensures that every booking has a standardized, unique identifier even if the Name field is left blank.

Trigger Type:

Before Insert: Executes before the record is saved to the database, allowing the Name field to be populated automatically.


File Edit Debug Test Workspace Help < >
BookingRequestController.apxc
GenerateBookingReference.apxt
SendBookingConfirmationEmail.apxt
CreateBurialAfterApproval.apxt
Code Coverage: None
API Version: 64
Go To
Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem

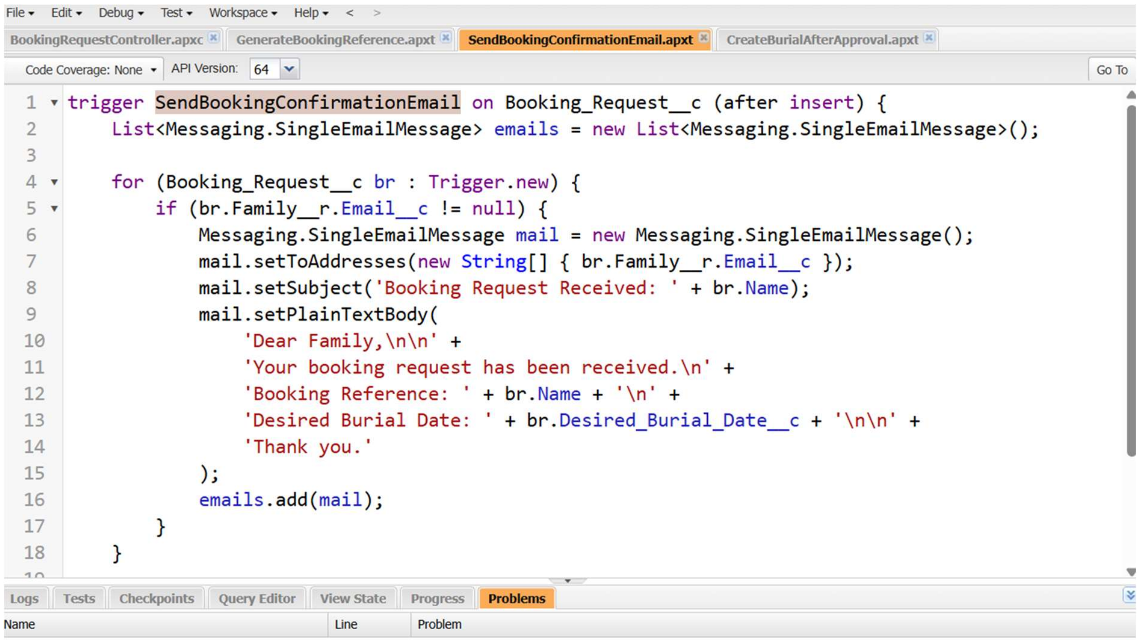
```
trigger GenerateBookingReference on Booking_Request__c (before insert) {
    for (Booking_Request__c br : Trigger.new) {
        if (br.Name == null) {
            String yy = String.valueOf(Date.today().year()).substring(2);
            br.Name = 'BR-' + yy + '-' + String.valueOf(Math.mod(Math.abs(Crypto.getRandomInteger(
        }
    }
}
```

- Send Booking Confirmation Email Trigger

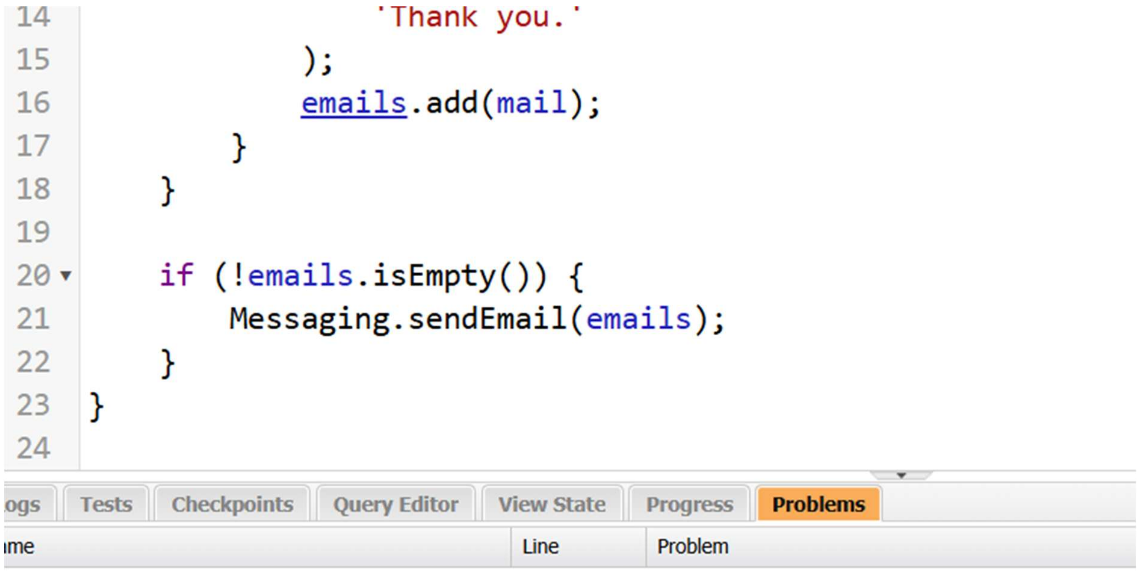
Purpose: The SendBookingConfirmationEmail trigger automatically sends a confirmation email to the family or user whenever a new booking request is created. This ensures that users receive immediate acknowledgment of their booking submission

Trigger Type:

After Insert: Executes after the record is saved to the database, so the booking ID and other fields are available for the email.


File Edit Debug Test Workspace Help
BookingRequestController.apxc
GenerateBookingReference.apxt
SendBookingConfirmationEmail.apxt
CreateBurialAfterApproval.apxt
Code Coverage: None
API Version: 64
Go To
Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem

```
trigger SendBookingConfirmationEmail on Booking_Request__c (after insert) {
    List<Messaging.SingleEmailMessage> emails = new List<Messaging.SingleEmailMessage>();

    for (Booking_Request__c br : Trigger.new) {
        if (br.Family__r.Email__c != null) {
            Messaging.SingleEmailMessage mail = new Messaging.SingleEmailMessage();
            mail.setToAddresses(new String[] { br.Family__r.Email__c });
            mail.setSubject('Booking Request Received: ' + br.Name);
            mail.setPlainTextBody(
                'Dear Family,\n\n' +
                'Your booking request has been received.\n' +
                'Booking Reference: ' + br.Name + '\n' +
                'Desired Burial Date: ' + br.Desired_Burial_Date__c + '\n\n' +
                'Thank you.'
            );
            emails.add(mail);
        }
    }
```


Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem

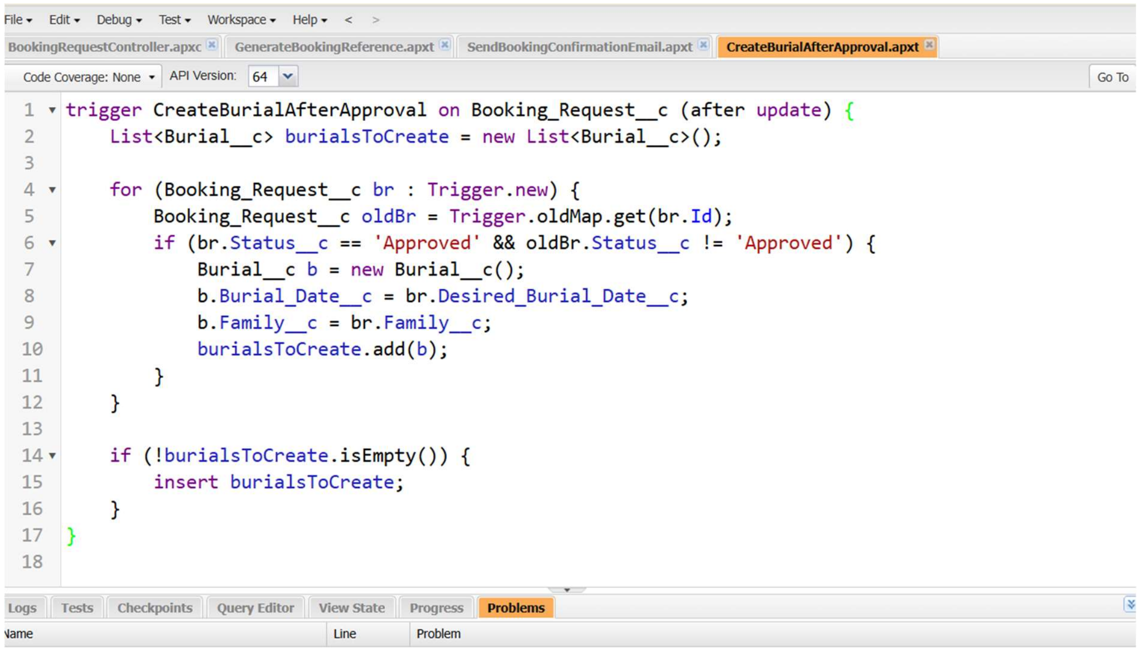
```
                'Thank you.'
            );
            emails.add(mail);
        }
    }

    if (!emails.isEmpty()) {
        Messaging.sendEmail(emails);
    }
}
```

- Create Burial After Approval Trigger

The CreateBurialAfterApproval trigger automatically creates a related Burial record when a booking request is approved. This ensures that only approved bookings result in actionable burial records, maintaining data integrity and workflow automation.

Trigger Type:

After Update: Executes after a booking request record is updated, because the trigger needs to check the updated Status field.


File Edit Debug Test Workspace Help
BookingRequestController.apxc
GenerateBookingReference.apxt
SendBookingConfirmationEmail.apxt
CreateBurialAfterApproval.apxt
Code Coverage: None
API Version: 64
Go To
Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem

```
trigger CreateBurialAfterApproval on Booking_Request__c (after update) {
    List<Burial__c> burialsToCreate = new List<Burial__c>();

    for (Booking_Request__c br : Trigger.new) {
        Booking_Request__c oldBr = Trigger.oldMap.get(br.Id);
        if (br.Status__c == 'Approved' && oldBr.Status__c != 'Approved') {
            Burial__c b = new Burial__c();
            b.Burial_Date__c = br.Desired_Burial_Date__c;
            b.Family__c = br.Family__c;
            burialsToCreate.add(b);
        }
    }

    if (!burialsToCreate.isEmpty()) {
        insert burialsToCreate;
    }
}
```

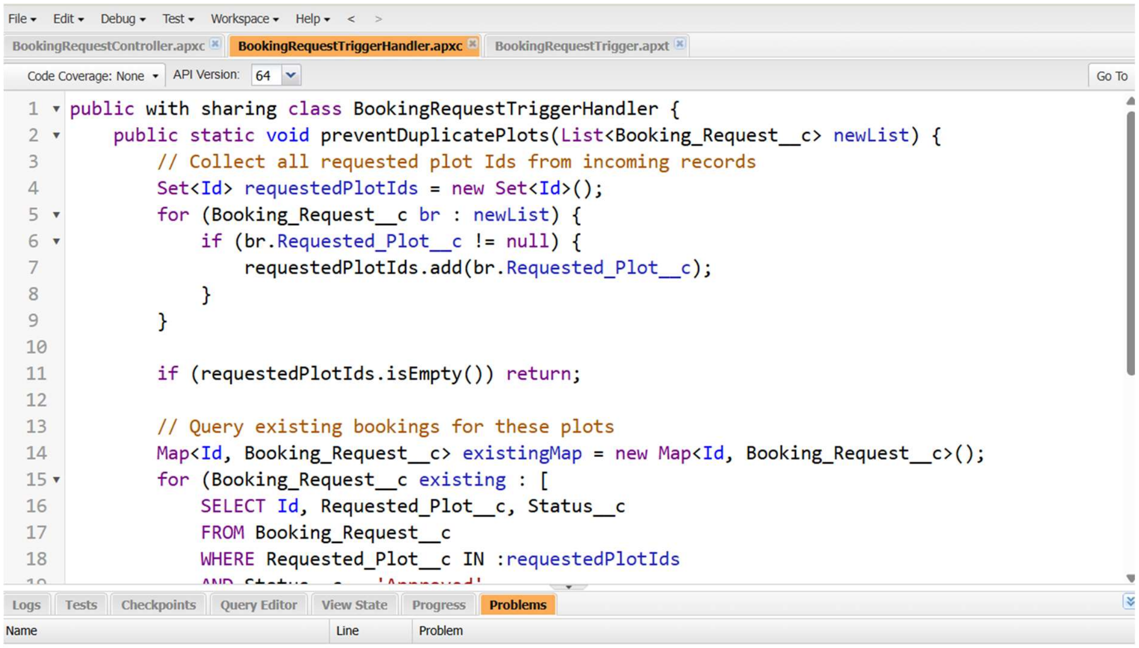
2. SOQL & SOSL

- SOQL is mainly used in triggers, controllers, and batch jobs to fetch booking or burial records.
- SOSL is helpful for search functionality in Experience Cloud, where users or admins might search bookings/families using names, references, or emails.

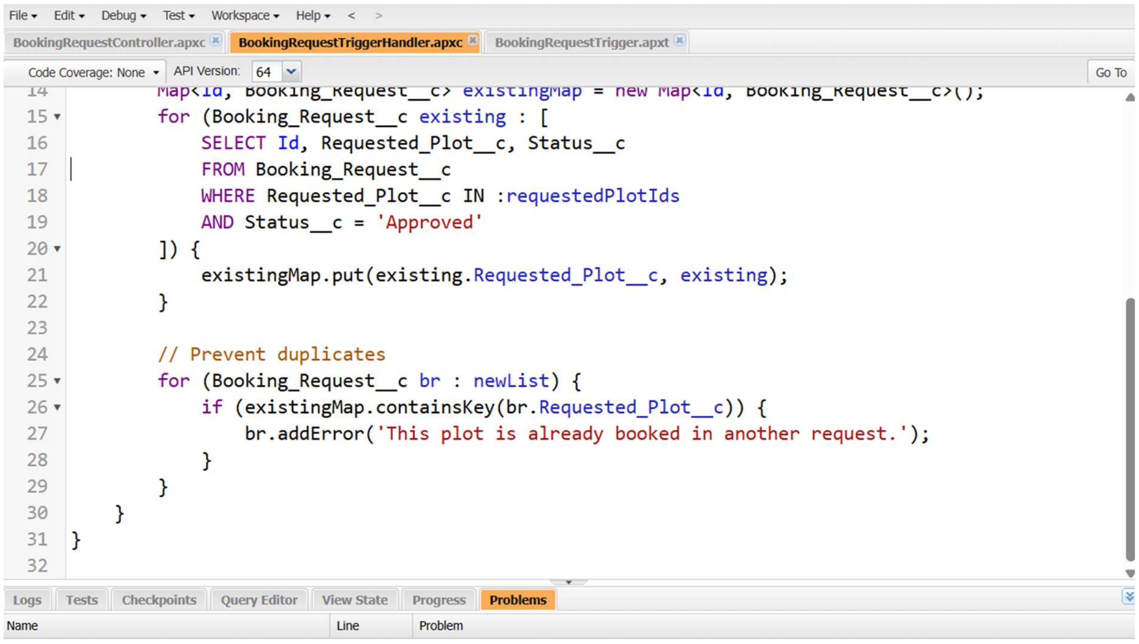
3. Collections: List, Set, Map

    - List – It is an ordered collection that allows duplicates. It store multiple booking requests retrieved using SOQL.
    - Set – An unordered collection of unique values (no duplicates). It store unique Family IDs from booking requests.
    - Map – A collection of key–value pairs, where each key is unique. Link Booking IDs to their corresponding Booking records. Also it quickly fetches a record by ID without looping.

BookingRequestTriggerHandler.cls (Apex Class)

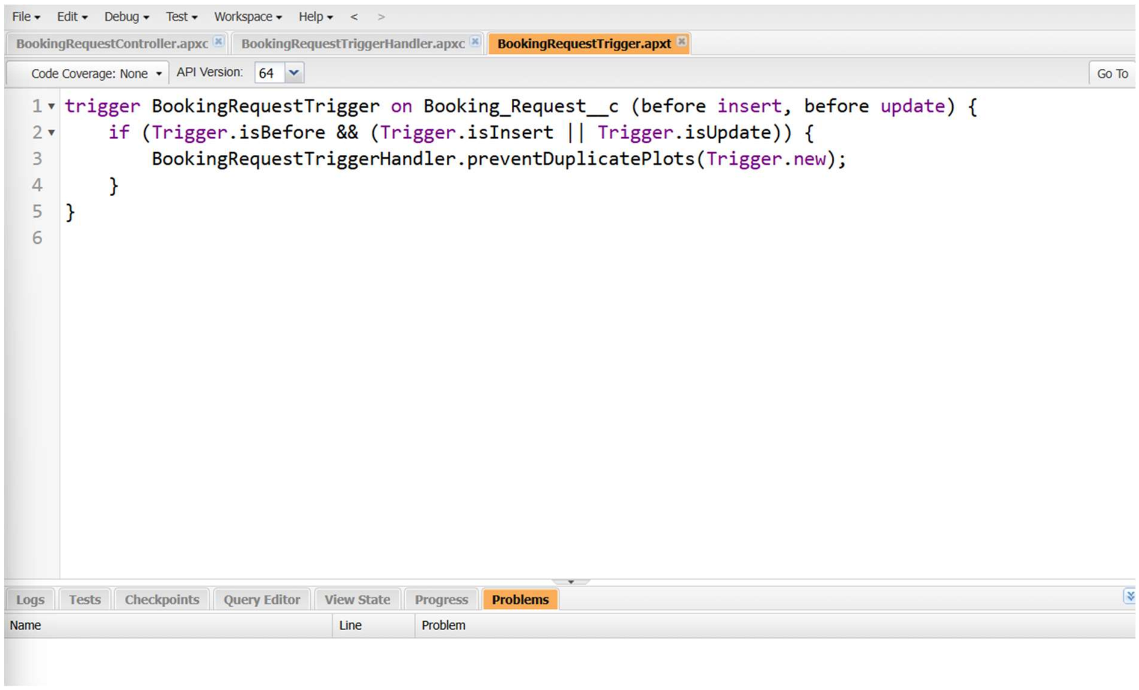

File Edit Debug Test Workspace Help < >
BookingRequestController.apxc
BookingRequestTriggerHandler.apxc
BookingRequestTrigger.apxt
Code Coverage: None
API Version: 64
Go To
Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem

```
public with sharing class BookingRequestTriggerHandler {
    public static void preventDuplicatePlots(List<Booking_Request__c> newList) {
        // Collect all requested plot Ids from incoming records
        Set<Id> requestedPlotIds = new Set<Id>();
        for (Booking_Request__c br : newList) {
            if (br.Requested_Plot__c != null) {
                requestedPlotIds.add(br.Requested_Plot__c);
            }
        }

        if (requestedPlotIds.isEmpty()) return;

        // Query existing bookings for these plots
        Map<Id, Booking_Request__c> existingMap = new Map<Id, Booking_Request__c>();
        for (Booking_Request__c existing : [
            SELECT Id, Requested_Plot__c, Status__c
            FROM Booking_Request__c
            WHERE Requested_Plot__c IN :requestedPlotIds
```


File Edit Debug Test Workspace Help < >
BookingRequestController.apxc
BookingRequestTriggerHandler.apxc
BookingRequestTrigger.apxt
Code Coverage: None
API Version: 64
Go To
Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem

```
        Map<Id, Booking_Request__c> existingMap = new Map<Id, Booking_Request__c>();
        for (Booking_Request__c existing : [
            SELECT Id, Requested_Plot__c, Status__c
            FROM Booking_Request__c
            WHERE Requested_Plot__c IN :requestedPlotIds
            AND Status__c = 'Approved'
        ]) {
            existingMap.put(existing.Requested_Plot__c, existing);
        }

        // Prevent duplicates
        for (Booking_Request__c br : newList) {
            if (existingMap.containsKey(br.Requested_Plot__c)) {
                br.addError('This plot is already booked in another request.');
            }
        }
    }
}
```

BookingRequestTrigger.trigger (Apex Trigger)


File Edit Debug Test Workspace Help < >
BookingRequestController.apxc
BookingRequestTriggerHandler.apxc
BookingRequestTrigger.apxt
Code Coverage: None
API Version: 64
Go To
Logs Tests Checkpoints Query Editor View State Progress Problems
Name Line Problem

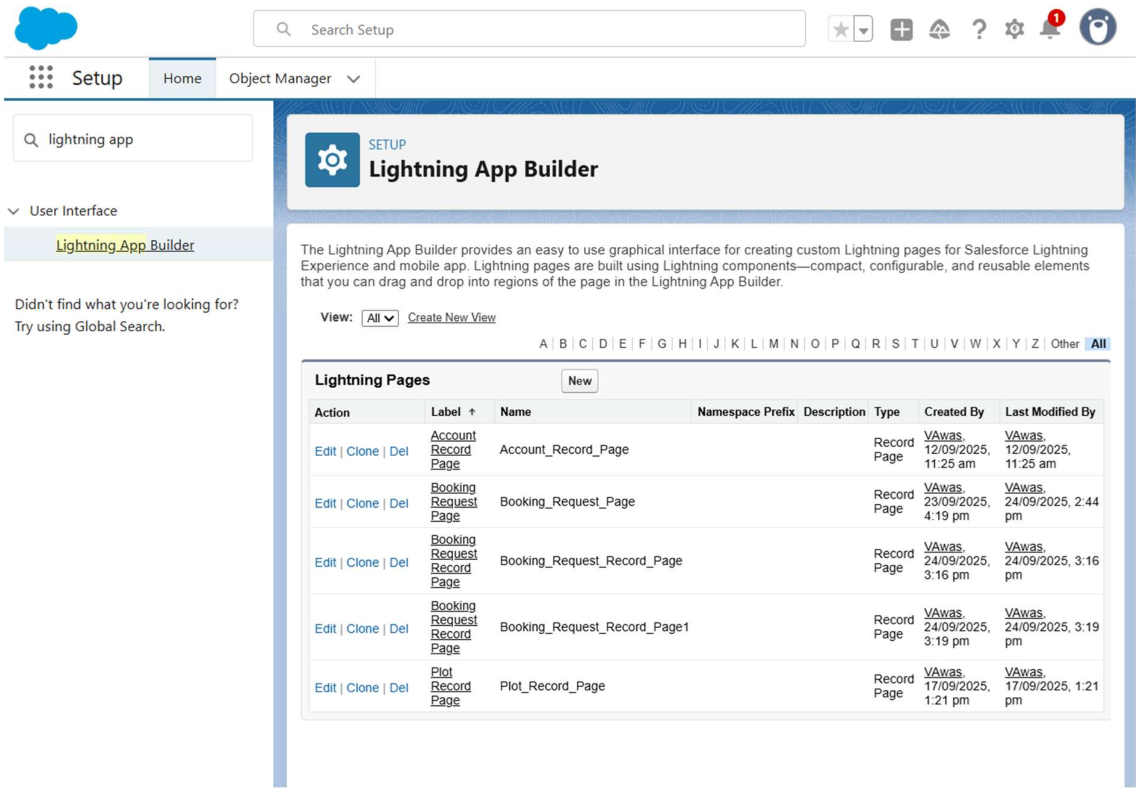
```
trigger BookingRequestTrigger on Booking_Request__c (before insert, before update) {
    if (Trigger.isBefore && (Trigger.isInsert || Trigger.isUpdate)) {
        BookingRequestTriggerHandler.preventDuplicatePlots(Trigger.new);
    }
}
```

PHASE 6: User Interface Development

Goal: The goal of this phase is to design and implement an intuitive, user-friendly interface for the Booking & Family Portal system. This includes using Lightning App Builder, Lightning Web Components (LWC), Flows, and record pages to enable families and admins to submit, view, and manage booking requests efficiently.

1. Lightning App Builder

The Lightning App Builder was used in our project to design and customize pages for the Family Portal and internal users. It allowed us to drag and drop components, such as Flows for booking requests, record detail pages, and related lists, without writing code. This provided a user-friendly interface for families to submit bookings and for admins to manage approvals, ensuring a smooth and intuitive experience



The Lightning App Builder provides an easy to use graphical interface for creating custom Lightning pages for Salesforce Lightning Experience and mobile app. Lightning pages are built using Lightning components—compact, configurable, and reusable elements that you can drag and drop into regions of the page in the Lightning App Builder.

View: All | Create New View

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | Other | All

Lightning Pages — New

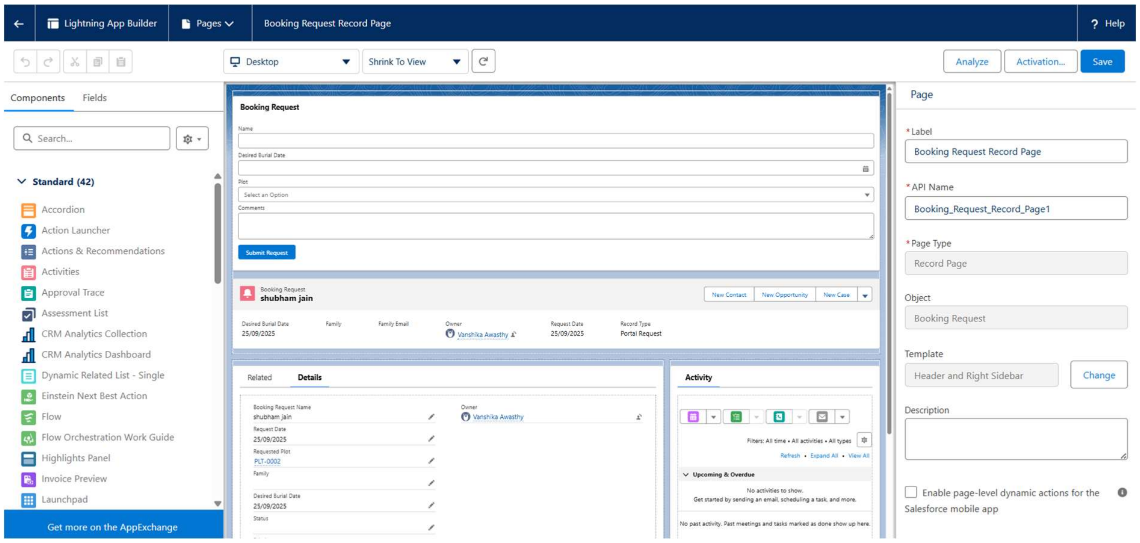
Action	Label ↑	Name	Namespace Prefix	Description	Type	Created By	Last Modified By
Edit | Clone | Del	Account Record Page	Account_Record_Page			Record Page	VAwas, 12/09/2025, 11:25 am	VAwas, 12/09/2025, 11:25 am
Edit | Clone | Del	Booking Request Page	Booking_Request_Page			Record Page	VAwas, 23/09/2025, 4:19 pm	VAwas, 24/09/2025, 2:44 pm
Edit | Clone | Del	Booking Request Record Page	Booking_Request_Record_Page			Record Page	VAwas, 24/09/2025, 3:16 pm	VAwas, 24/09/2025, 3:16 pm
Edit | Clone | Del	Booking Request Record Page	Booking_Request_Record_Page1			Record Page	VAwas, 24/09/2025, 3:19 pm	VAwas, 24/09/2025, 3:19 pm
Edit | Clone | Del	Plot Record Page	Plot_Record_Page			Record Page	VAwas, 17/09/2025, 1:21 pm	VAwas, 17/09/2025, 1:21 pm

.

2. Record Pages

- Booking Request Record Page

The Booking Request Record Page was customized using Lightning App Builder to display all key details of a booking request in one place.



Booking Request Record Page

Booking Request

Name

Desired Burial Date

Plot: Select an Option

Comments

Submit Request

Booking Request
shubham jain

Desired Burial Date	Family	Family Email	Owner	Request Date	Record Type
25/09/2025			Vanshika Awasthy	25/09/2025	Portal Request

Booking Request Name: shubham jain
Owner: Vanshika Awasthy
Request Date: 25/09/2025
Requested Plot: PLT-0002
Family:
Desired Burial Date: 25/09/2025
Status:

Filters: All time • All activities • All types

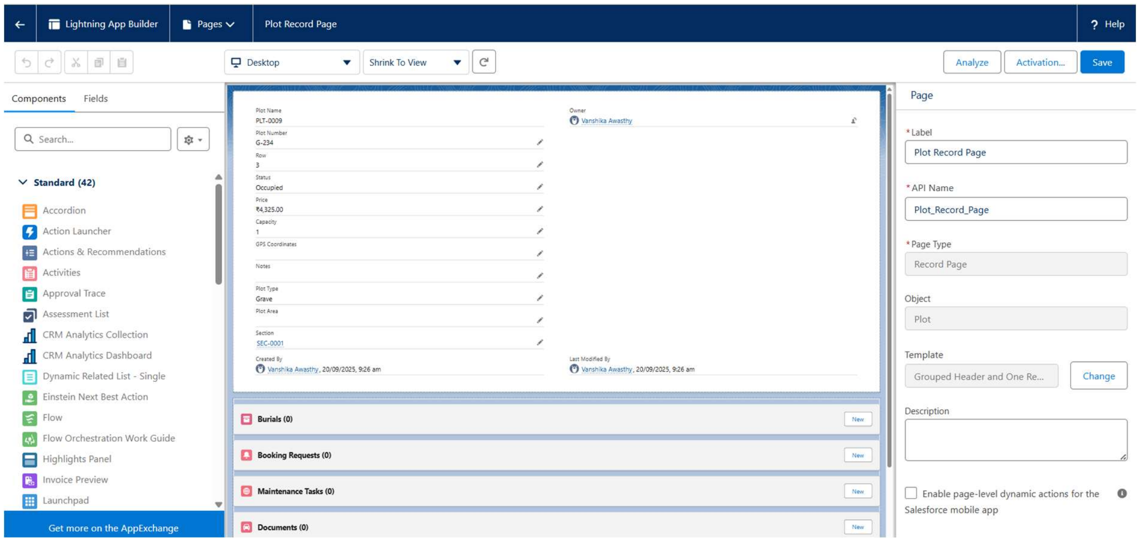
No activities to show.
Get started by sending an email, scheduling a task, and more.

No past activity. Past meetings and tasks marked as done show up here.

*Label: Booking Request Record Page
*API Name: Booking_Request_Record_Page1
*Page Type: Record Page
Object: Booking Request
Template: Header and Right Sidebar
Description:
Enable page-level dynamic actions for the Salesforce mobile app

- Plot Record Page

The Plot Record Page was designed in Lightning App Builder to display all information related to burial plots, including Plot Number, Location, Availability Status, and linked Burial records.



Plot Record Page

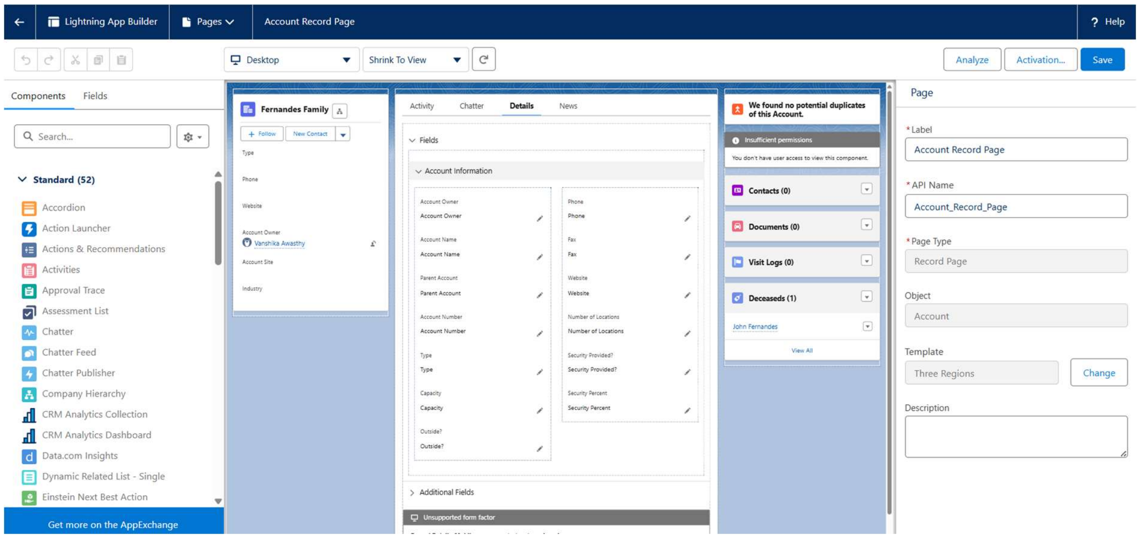
Plot Name: PLT-0009
Owner: Vanshika Awasthy
Plot Number: G-234
Row: 3
Status: Occupied
Price: ₹4,325.00
Capacity: 1
GPS Coordinates:
Notes:
Plot Type: Grave
Plot Area:
Section: SEC-0001
Created By: Vanshika Awasthy, 20/09/2025, 9:26 am
Last Modified By: Vanshika Awasthy, 20/09/2025, 9:26 am

Burials (0)
Booking Requests (0)
Maintenance Tasks (0)
Documents (0)

*Label: Plot Record Page
*API Name: Plot_Record_Page
*Page Type: Record Page
Object: Plot
Template: Grouped Header and One Re...
Description:
Enable page-level dynamic actions for the Salesforce mobile app

- Account Record Page

The Account Record Page was customized to act as the central hub for managing families and their related records.



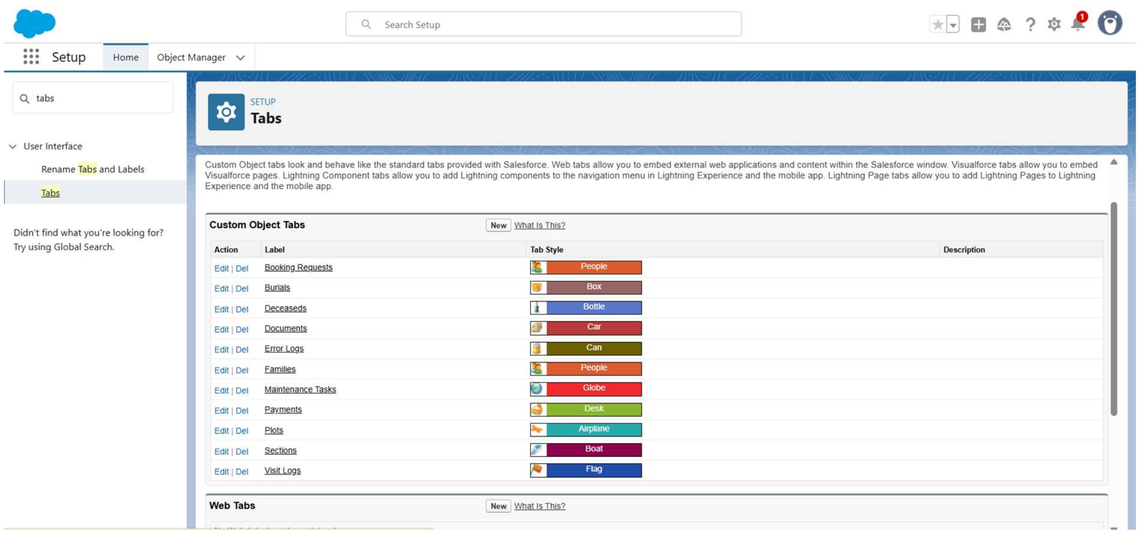
Page

*Label	Account Record Page
*API Name	Account_Record_Page
*Page Type	Record Page
Object	Account
Template	Three Regions
Description	

3. Tabs

Tabs are navigation items that allow users to access specific objects, records, or functionality quickly. Each tab acts like a shortcut that opens a particular type of data or application page. Tabs can represent standard objects (like Accounts, Contacts), custom objects (like Booking Requests, Burials, Plots), or even Visualforce/Aura/LWC pages.

In our project, tabs were created for Booking Requests, Burials, Deceased, Documents, Error Logs, Families, Maintenance Task, Payments, Plots, Section and Visit Logs.



Tabs

Custom Object tabs look and behave like the standard tabs provided with Salesforce. Web tabs allow you to embed external web applications and content within the Salesforce window. Visualforce tabs allow you to embed Visualforce pages. Lightning Component tabs allow you to add Lightning components to the navigation menu in Lightning Experience and the mobile app. Lightning Page tabs allow you to add Lightning Pages to Lightning Experience and the mobile app.

Custom Object Tabs New What Is This?

Action	Label	Tab Style	Description
Edit | Del	Booking Requests	People	
Edit | Del	Burials	Box	
Edit | Del	Deceaseds	Bottle	
Edit | Del	Documents	Car	
Edit | Del	Error Logs	Can	
Edit | Del	Families	People	
Edit | Del	Maintenance Tasks	Globe	
Edit | Del	Payments	Desk	
Edit | Del	Plots	Airplane	
Edit | Del	Sections	Boat	
Edit | Del	Visit Logs	Flag	

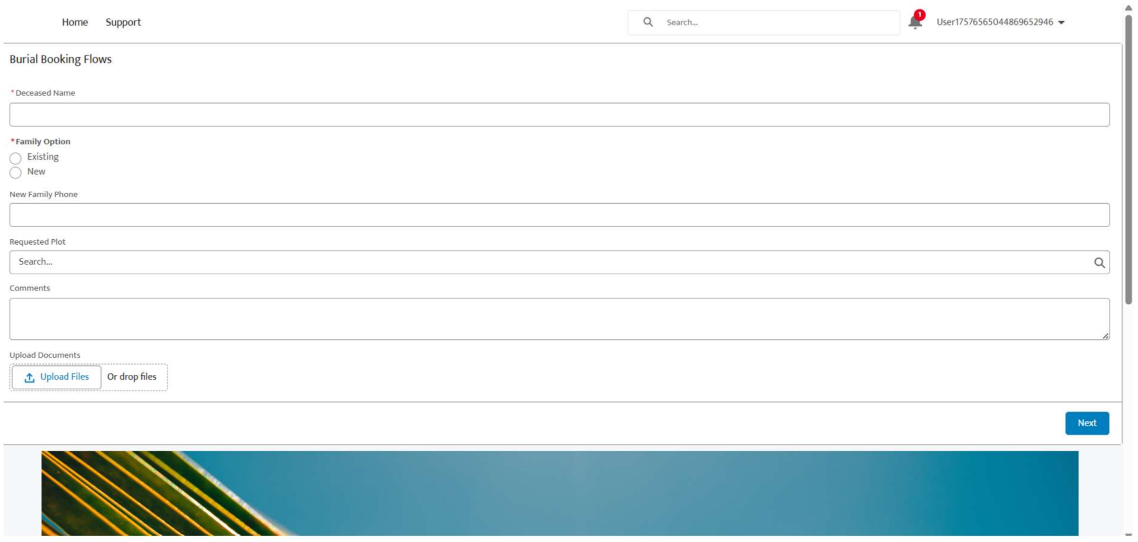
Web Tabs New What Is This?

4. Lightning Web Components (LWC)

Lightning Web Components (LWC) is Salesforce’s modern framework for building fast, reusable, and responsive web components using standard web technologies like HTML, JavaScript, and CSS

In our project, an LWC was developed for the Booking Request Form. This component allowed families to enter booking details in a dynamic and user-friendly interface. It improved the booking experience by making the form interactive, responsive, and mobile-friendly, compared to traditional page layouts or Visualforce pages.

Booking Request Family Portal Form



Burial Booking Flows

*Deceased Name

*Family Option
Existing
New

New Family Phone

Requested Plot
Search...

Comments

Upload Documents

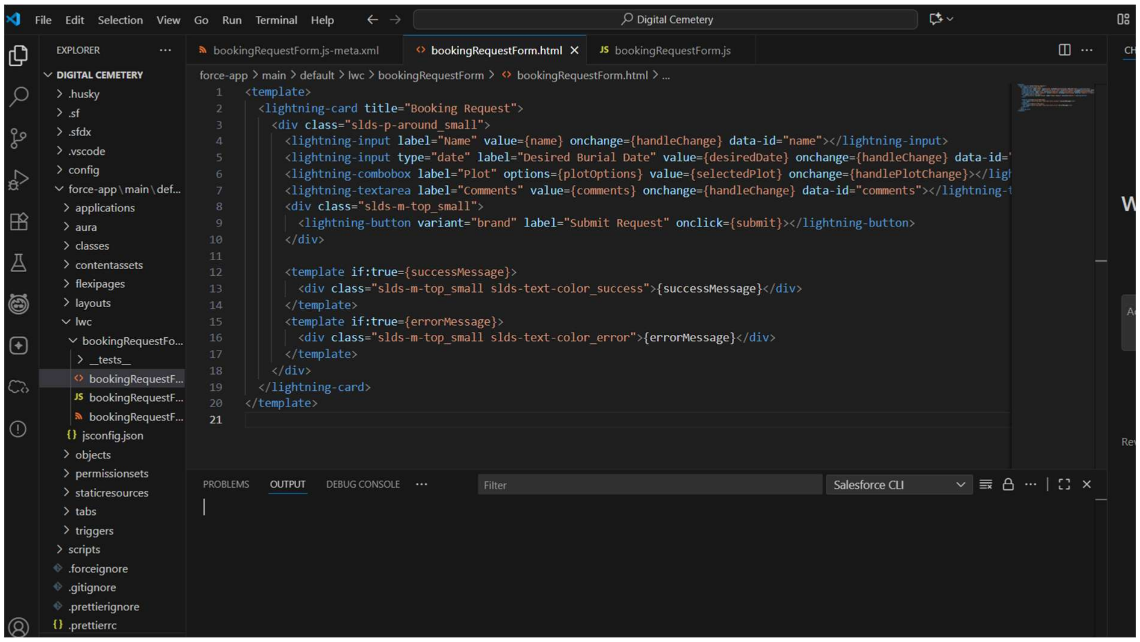
Lightning Web Component (LWC) Files:

- bookingRequestForm.html (Template / UI)

Defines the form layout: Name, Desired Burial Date, Plot selection, Comments, Submit button.

Displays success and error messages dynamically.

Uses Salesforce Lightning base components for consistent UI and accessibility.



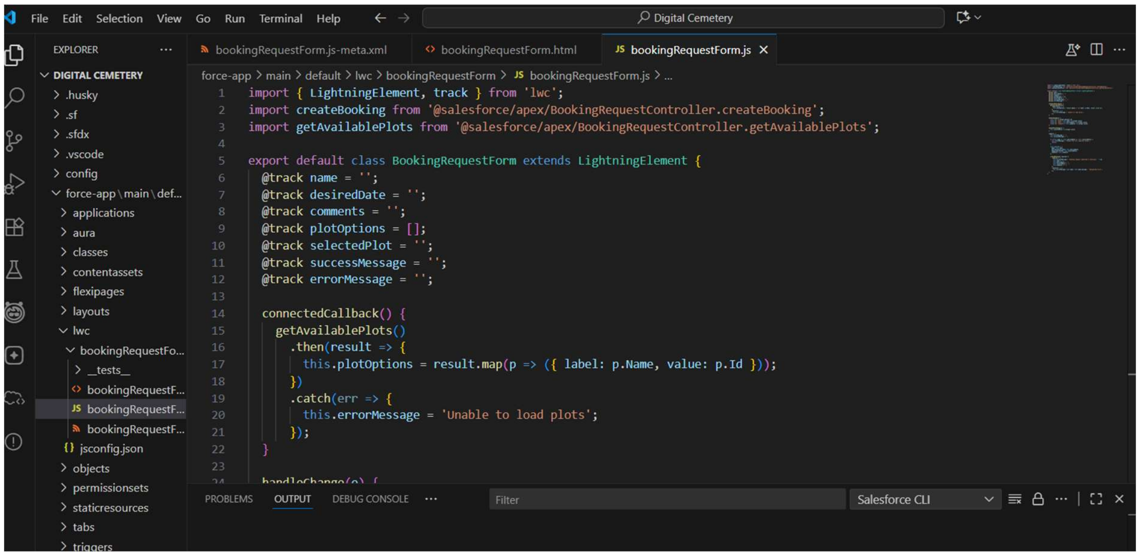
```html
<template>
  <lightning-card title="Booking Request">
    <div class="slds-p-around_small">
      <lightning-input label="Name" value={name} onchange={handleChange} data-id="name"></lightning-input>
      <lightning-input type="date" label="Desired Burial Date" value={desiredDate} onchange={handleChange} data-id='
      <lightning-combobox label="Plot" options={plotOptions} value={selectedPlot} onchange={handlePlotChange}></ligh
      <lightning-textarea label="Comments" value={comments} onchange={handleChange} data-id="comments"></lightning-t
      <div class="slds-m-top_small">
        <lightning-button variant="brand" label="Submit Request" onclick={submit}></lightning-button>
      </div>

      <template if:true={successMessage}>
        <div class="slds-m-top_small slds-text-color_success">{successMessage}</div>
      </template>
      <template if:true={errorMessage}>
        <div class="slds-m-top_small slds-text-color_error">{errorMessage}</div>
      </template>
    </div>
  </lightning-card>
</template>
```

- bookingRequestForm.js (Controller / Logic)

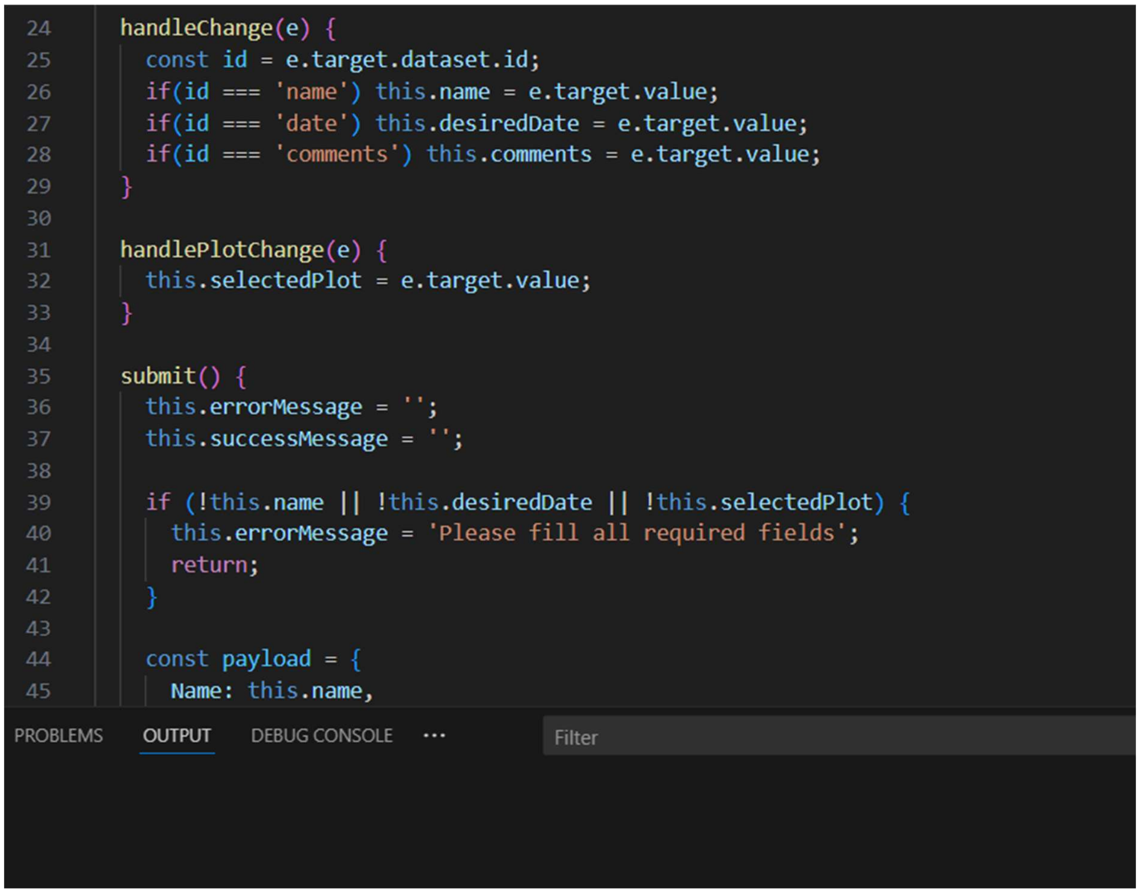
Handles component behavior : tracks field values, responds to user input, validates form.

Calls Apex (BookingRequestController) to create Booking Request records.

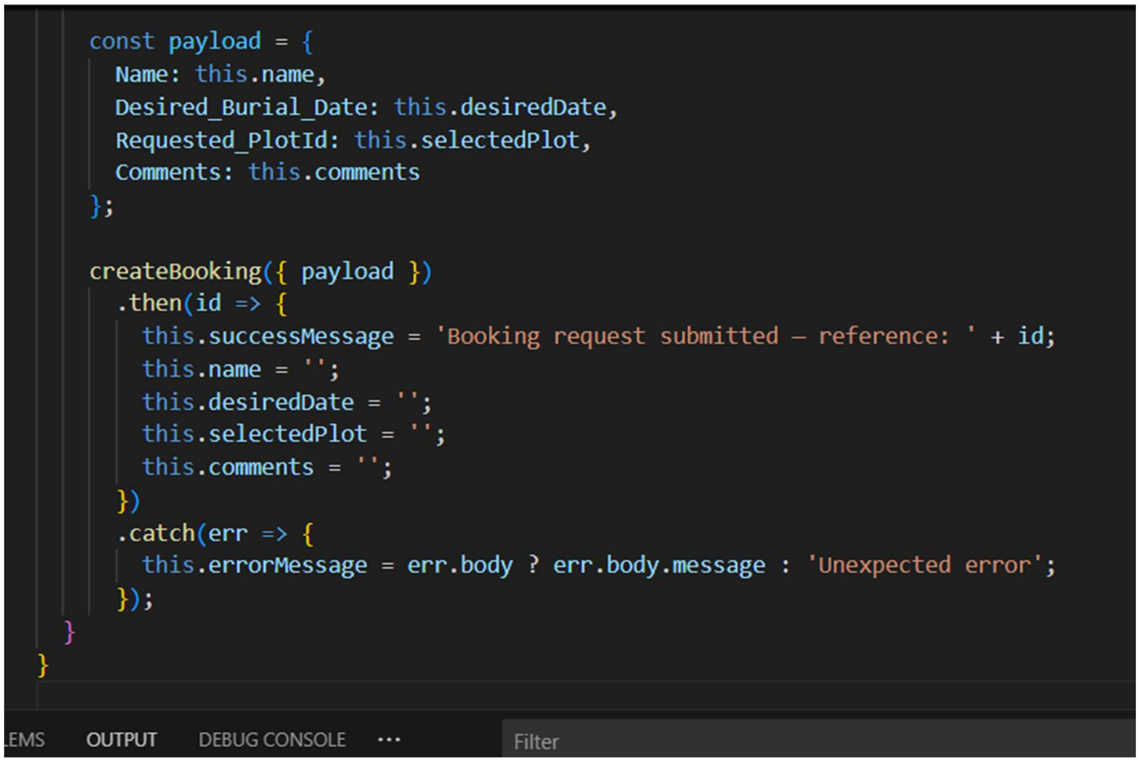
Updates UI with success or error messages.



```javascript
import { LightningElement, track } from 'lwc';
import createBooking from '@salesforce/apex/BookingRequestController.createBooking';
import getAvailablePlots from '@salesforce/apex/BookingRequestController.getAvailablePlots';

export default class BookingRequestForm extends LightningElement {
  @track name = '';
  @track desiredDate = '';
  @track comments = '';
  @track plotOptions = [];
  @track selectedPlot = '';
  @track successMessage = '';
  @track errorMessage = '';

  connectedCallback() {
    getAvailablePlots()
      .then(result => {
        this.plotOptions = result.map(p => ({ label: p.Name, value: p.Id }));
      })
      .catch(err => {
        this.errorMessage = 'Unable to load plots';
      });
  }

```



```javascript
  handleChange(e) {
    const id = e.target.dataset.id;
    if(id === 'name') this.name = e.target.value;
    if(id === 'date') this.desiredDate = e.target.value;
    if(id === 'comments') this.comments = e.target.value;
  }

  handlePlotChange(e) {
    this.selectedPlot = e.target.value;
  }

  submit() {
    this.errorMessage = '';
    this.successMessage = '';

    if (!this.name || !this.desiredDate || !this.selectedPlot) {
      this.errorMessage = 'Please fill all required fields';
      return;
    }

    const payload = {
      Name: this.name,
```



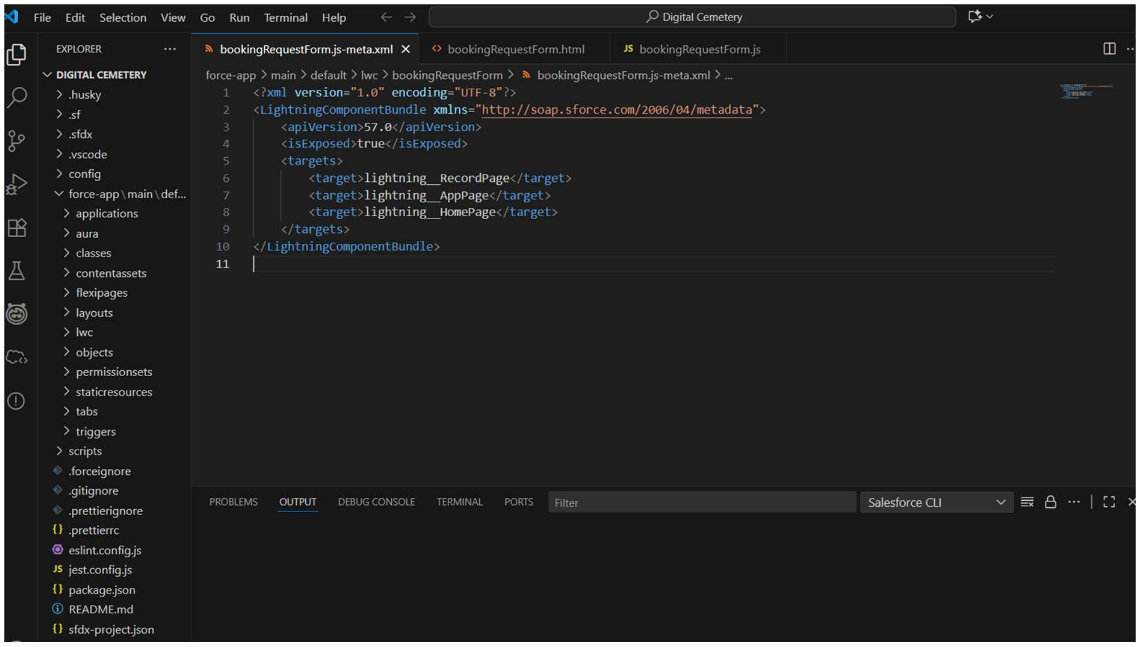
```
    const payload = {
      Name: this.name,
      Desired_Burial_Date: this.desiredDate,
      Requested_PlotId: this.selectedPlot,
      Comments: this.comments
    };

    createBooking({ payload })
      .then(id => {
        this.successMessage = 'Booking request submitted – reference: ' + id;
        this.name = '';
        this.desiredDate = '';
        this.selectedPlot = '';
        this.comments = '';
      })
      .catch(err => {
        this.errorMessage = err.body ? err.body.message : 'Unexpected error';
      });
  }
}
```

- bookingRequestForm.js-meta.xml (Metadata / Visibility)

Makes the LWC available in Lightning App Builder.

Defines page targets: Record Page, App Page, Home Page.

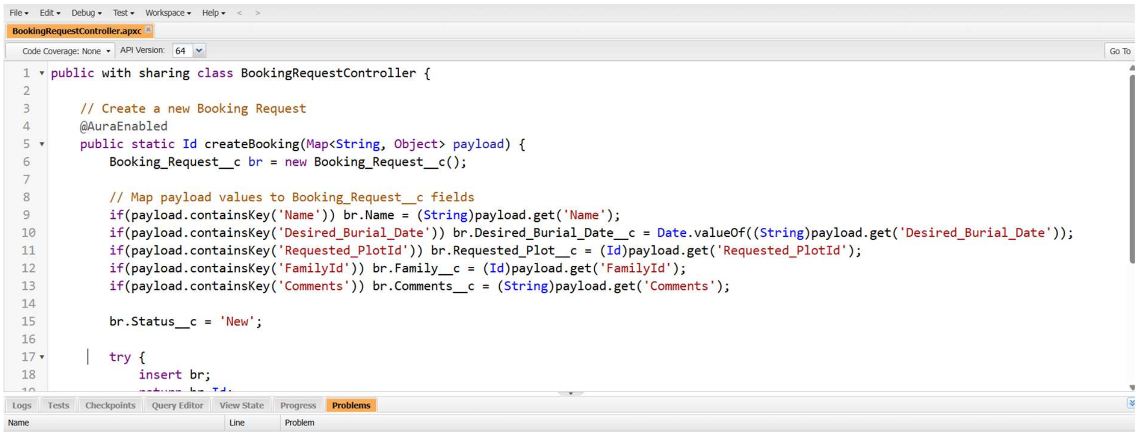
Controls where and how admins can place the component.



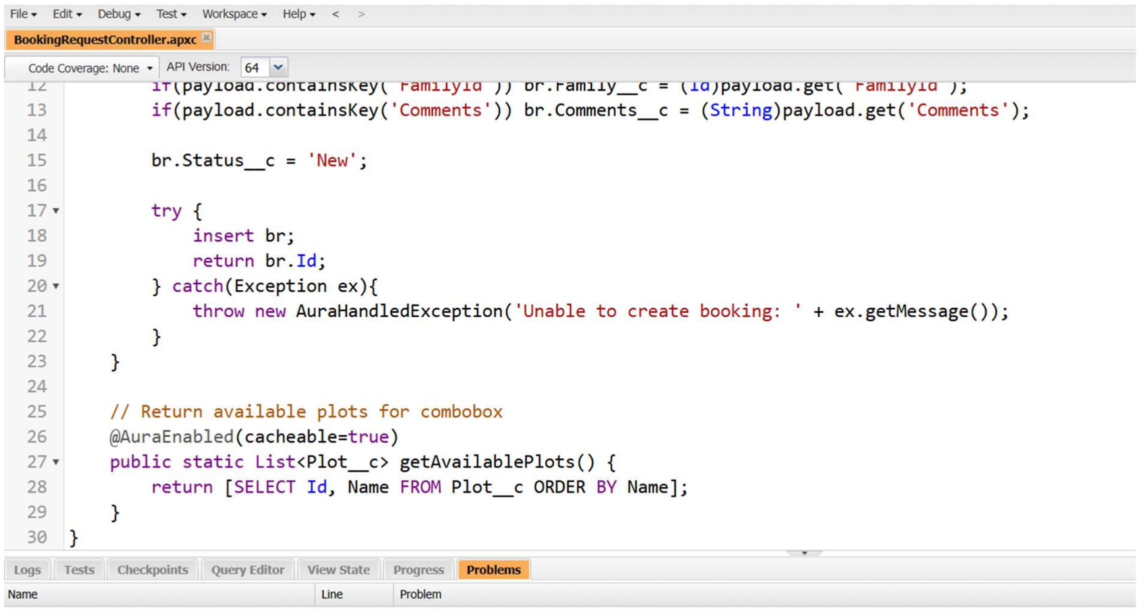
```xml
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>57.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
        <target>lightning__AppPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>

```

5. Apex with LWC

Booking Request Controller Class



```apex
public with sharing class BookingRequestController {

    // Create a new Booking Request
    @AuraEnabled
    public static Id createBooking(Map<String, Object> payload) {
        Booking_Request__c br = new Booking_Request__c();

        // Map payload values to Booking_Request__c fields
        if(payload.containsKey('Name')) br.Name = (String)payload.get('Name');
        if(payload.containsKey('Desired_Burial_Date')) br.Desired_Burial_Date__c = Date.valueOf((String)payload.get('Desired_Burial_Date'));
        if(payload.containsKey('Requested_PlotId')) br.Requested_Plot__c = (Id)payload.get('Requested_PlotId');
        if(payload.containsKey('FamilyId')) br.Family__c = (Id)payload.get('FamilyId');
        if(payload.containsKey('Comments')) br.Comments__c = (String)payload.get('Comments');

        br.Status__c = 'New';

        try {
            insert br;
```



```
        if(payload.containsKey('FamilyId')) br.Family__c = (Id)payload.get('FamilyId');
        if(payload.containsKey('Comments')) br.Comments__c = (String)payload.get('Comments');

        br.Status__c = 'New';

        try {
            insert br;
            return br.Id;
        } catch(Exception ex){
            throw new AuraHandledException('Unable to create booking: ' + ex.getMessage());
        }
    }

    // Return available plots for combobox
    @AuraEnabled(cacheable=true)
    public static List<Plot__c> getAvailablePlots() {
        return [SELECT Id, Name FROM Plot__c ORDER BY Name];
    }
}
```

PHASE 7 : Integration & External Access

In the project Digital Cemetery , features such as Named Credentials, External Services, REST/SOAP APIs, Callouts, Platform Events, Change Data Capture, and Salesforce Connect were not implemented. The Booking & Family Portal system is entirely self-contained within Salesforce, with all processes handled using Apex, Flows, LWC, and Experience Cloud components. These integration features can be considered for future enhancements if external system connectivity is required.