Salesforce Real Estate Management Project - Detailed Documentation

Phase 1 — Problem Understanding & Industry Analysis

This phase is all about laying a solid foundation for your project. Before writing a single line of code or creating an object, you need to understand the "why" behind what you're building. This ensures your solution directly addresses a business need.

Goal: Define the scope, stakeholders, and the core workflows you'll automate.

Key Deliverable: A comprehensive requirements document that serves as the single source of truth for the project.

1. Gather Requirements: User Stories

Think from the perspective of each user.

Example User Stories:

- As an Agent, I want to see all available properties, so I can easily find options for my clients.
- As a Customer, I want to submit a booking inquiry online, so I don't have to call someone during business hours.
- As a Manager, I want to see a weekly summary of new bookings, so I can track team performance.
- As an Admin, I want a way to prevent new bookings from being created without a related property, so our data stays clean.

2. Identify Stakeholders

→ Admin: Maintains the Salesforce org, understands architecture and deployment.
→ Agent: Needs a simple, fast interface to manage properties and bookings.
→ Manager: Needs reports and dashboards to track key metrics.
→ Customer: Needs seamless experience for inquiries and bookings.

3. Map Main Process Flow

Start: Property Listed (Available)

Step 1: Customer Enquiry → Booking (Inquiry)

Step 2: Agent Reviews → Confirmed / Rejected

Step 3: Property Updates → Status Booked

End: Close

4. Define Success Criteria & KPIs

- Booking Conversion Rate

- Average Booking Amount

- Time to Close

Phase 2 — Org Setup & Configuration

Goal: Prepare a clean Salesforce org and a robust security model.

Key Deliverable: A documented security matrix outlining roles, profiles, permission sets, and sharing rules.

🛠️ 1. Provision Developer Org

- **Sign Up**: Go to Salesforce Developer Edition and create a free org.
- **Verify Email**: Confirm your email and log in.
- **Explore Setup**: Use the gear icon → Setup to access configuration tools.
- **Enable Dev Hub (Optional)**: If you're using scratch orgs or Salesforce Functions, enable Dev Hub from Setup.

👤 2. Build Profiles & Permission Sets

◇ Profiles

- **Purpose**: Control baseline access to objects, fields, tabs, apps, and record types.
- **Configuration**:
  - Go to Setup → Profiles.
  - Clone standard profiles to create custom ones.
  - Set object-level permissions (Read, Create, Edit, Delete).
  - Assign page layouts, record types, and tab visibility.

◇ Permission Sets

- **Purpose**: Additive access beyond the profile, allowing flexibility.
- **Configuration**:
  - Setup → Permission Sets → New.
  - Assign object permissions, field-level security, Apex class access, etc.

- Assign to users without changing their profile.

🏢 3. Define Role Hierarchy & Sharing Settings

◇ Role Hierarchy

- **Purpose**: Controls record visibility upward in the hierarchy.
- **Configuration**:
  - Setup → Roles → Define hierarchy.
  - Assign users to roles to enable visibility to subordinates' records.

◇ Organization-Wide Defaults (OWD)

- **Purpose**: Baseline record access for all users.
- **Configuration**:
  - Setup → Sharing Settings.
  - Set OWD for each object:
    - Bookings: Private
    - Customers: Private

◇ Sharing Rules

- **Purpose**: Extend access beyond role hierarchy.
- **Configuration**:
  - Setup → Sharing Settings → Create Sharing Rules.
  - Criteria-based or owner-based rules.
  - Share roles, public groups, or territories.

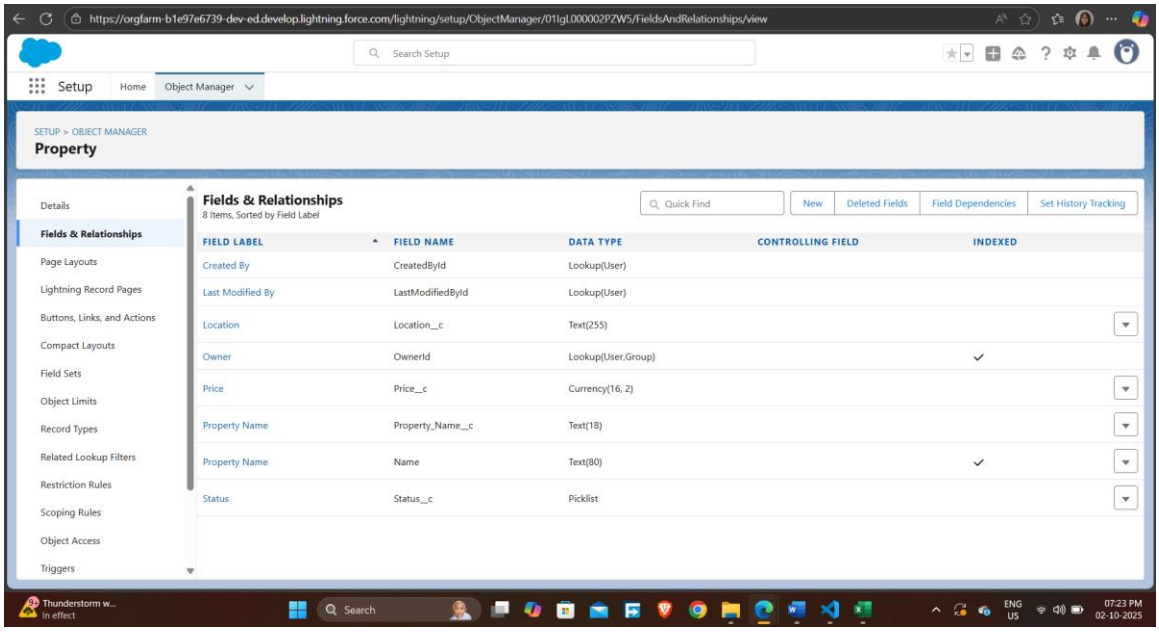
Phase 3 — Data Modeling & Relationships in Salesforce Lightning Experience

🧱 Step 1: Create Custom Objects

Go to **Setup → Object Manager → Create → Custom Object** for each:

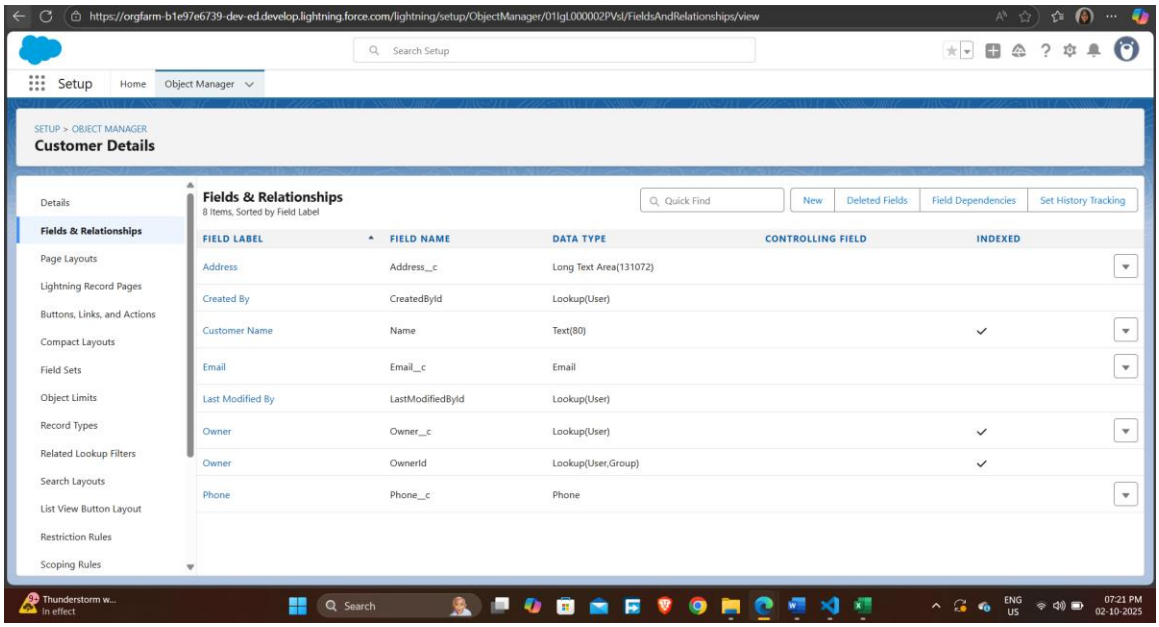
◇ Property__c

- Label: Property
- Plural Label: Properties
- Object Name: Property
- Record Name: Property Name (Text)



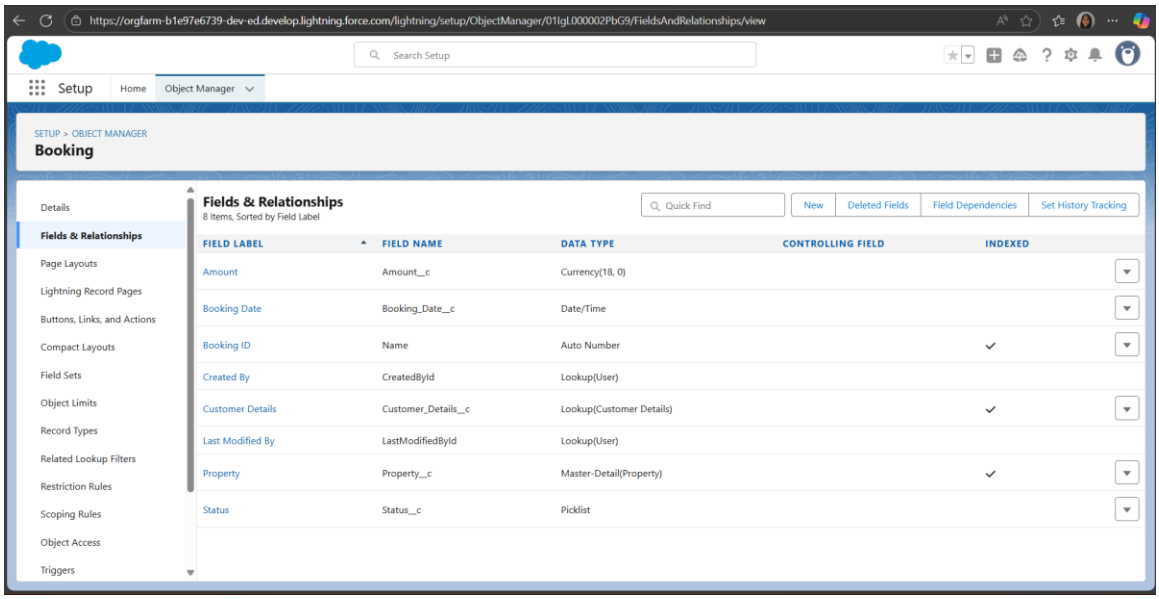
◇ Customer__c

- Label: Customer
- Plural Label: Customers
- Object Name: Customer
- Record Name: Customer Name (Text)



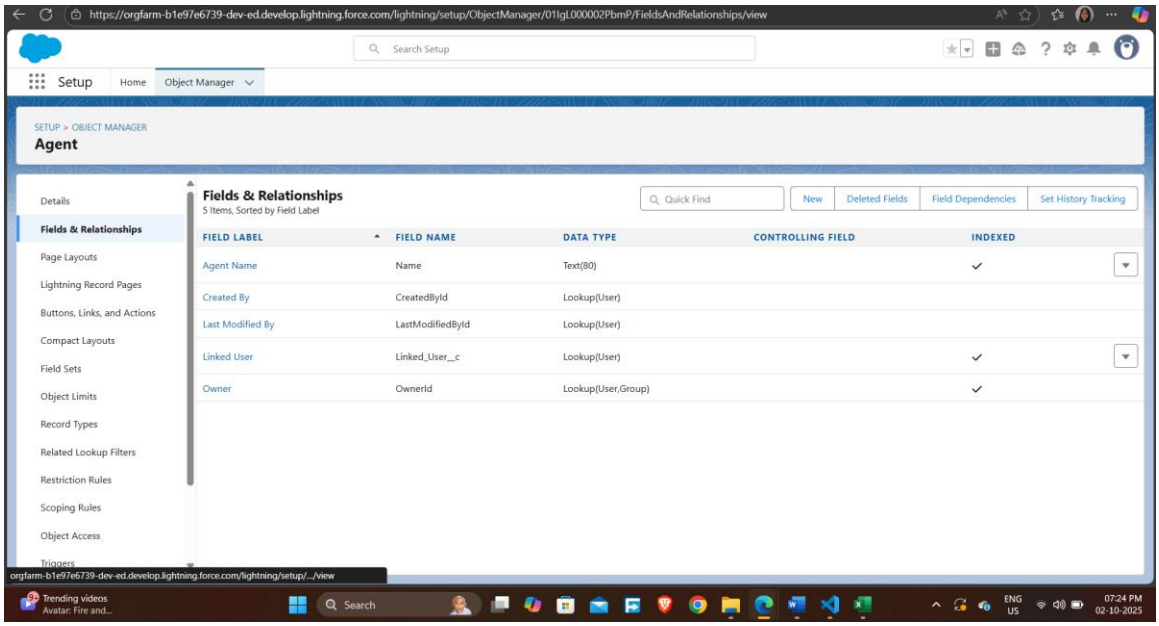
◇ Booking__c

- Label: Booking
- Plural Label: Bookings
- Object Name: Booking
- Record Name: Booking ID (Auto Number or Text)



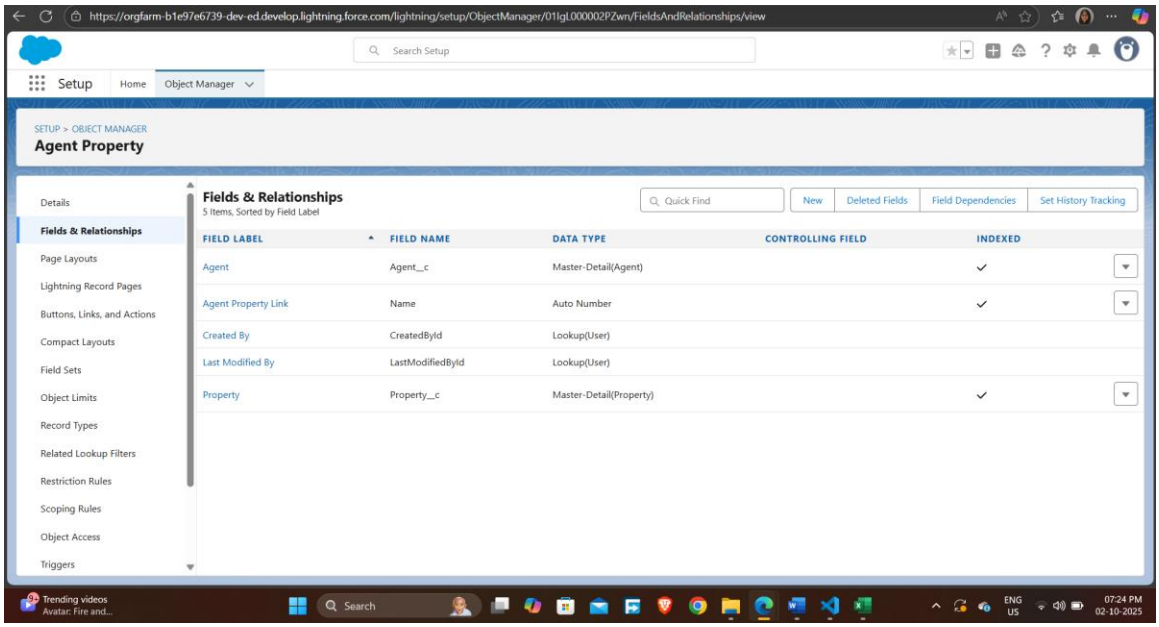
◇ Agent__c

- Label: Agent
- Plural Label: Agents
- Object Name: Agent
- Record Name: Agent Name (Text)



◇ Agent_Property__c (junction object)

- Label: Agent Property
- Plural Label: Agent Properties
- Object Name: Agent_Property
- Record Name: Agent Property ID (Auto Number)



🧩 Step 2: Add Custom Fields

Go to each object → Fields & Relationships → New Field:

◇ Property__c

- Location (Text)
- Price (Currency)
- Status (Picklist: Available, Sold, Under Contract)

◇ Booking__c

- Date (Date)
- Amount (Currency)
- Status (Picklist: Confirmed, Pending, Cancelled)

🔗 Step 3: Define Relationships

◇ Booking__c → Property__c (Master-Detail)

- On Booking__c, create a Master-Detail field to Property__c.

- This ensures Bookings are dependent on Property records.

◇ Booking__c → Customer__c (Lookup)

- On Booking__c, create a Lookup field to Customer__c.
- Allows linking a booking to a customer.

◇ Agent_Property__c → Agent__c + Property__c (Many-to-Many)

- On Agent_Property__c, create two Master-Detail fields:
  - Agent__c (Master-Detail)
  - Property__c (Master-Detail)
- This junction object enables many-to-many relationships between Agents and Properties.

📊 Step 4: Create Roll-Up Summary Fields

Go to **Property__c → Fields & Relationships → New Field → Roll-Up Summary**:

◇ Total_Bookings__c

- Summarize: COUNT of Booking__c records
- Filter: Status = Confirmed (optional)

◇ Total_Revenue__c

- Summarize: SUM of Booking__c.Amount
- Filter: Status = Confirmed (optional)

📐 Step 5: Schema Diagram (Optional)

Use Schema Builder:

- Setup → Schema Builder
- Drag and drop all custom objects
- Visualize relationships and field structure

Phase 4 — Process Automation (Admin)

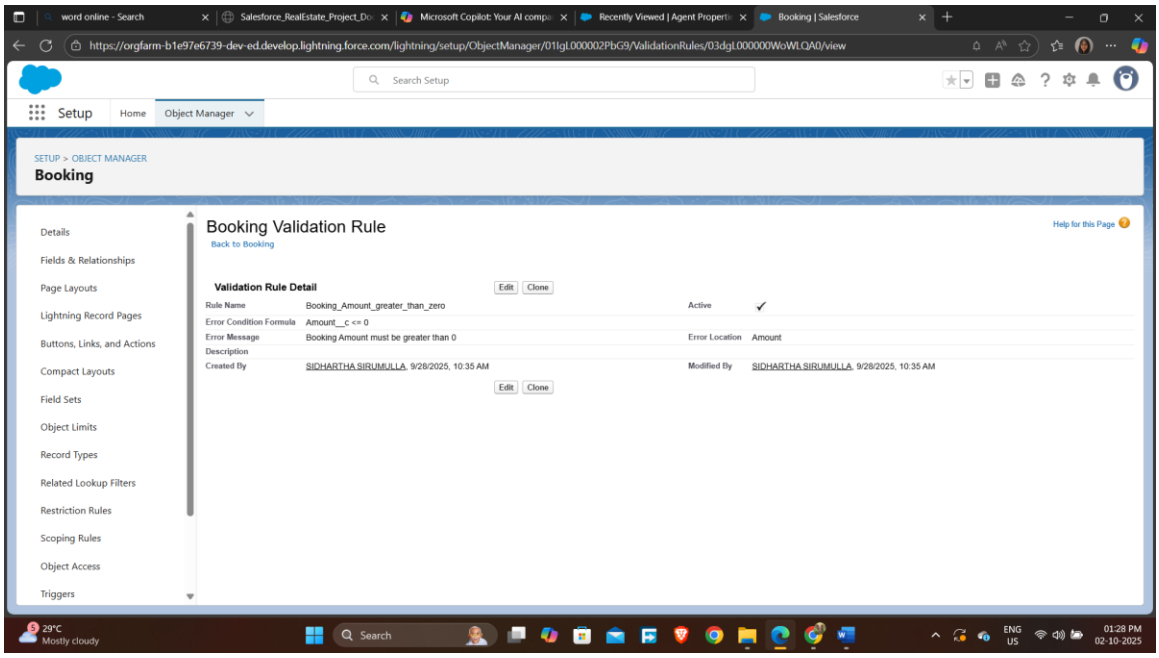
☑ 1. Validation Rules — Enforce Data Integrity

Validation Rules ensure users enter correct and complete data.

◇ Examples:

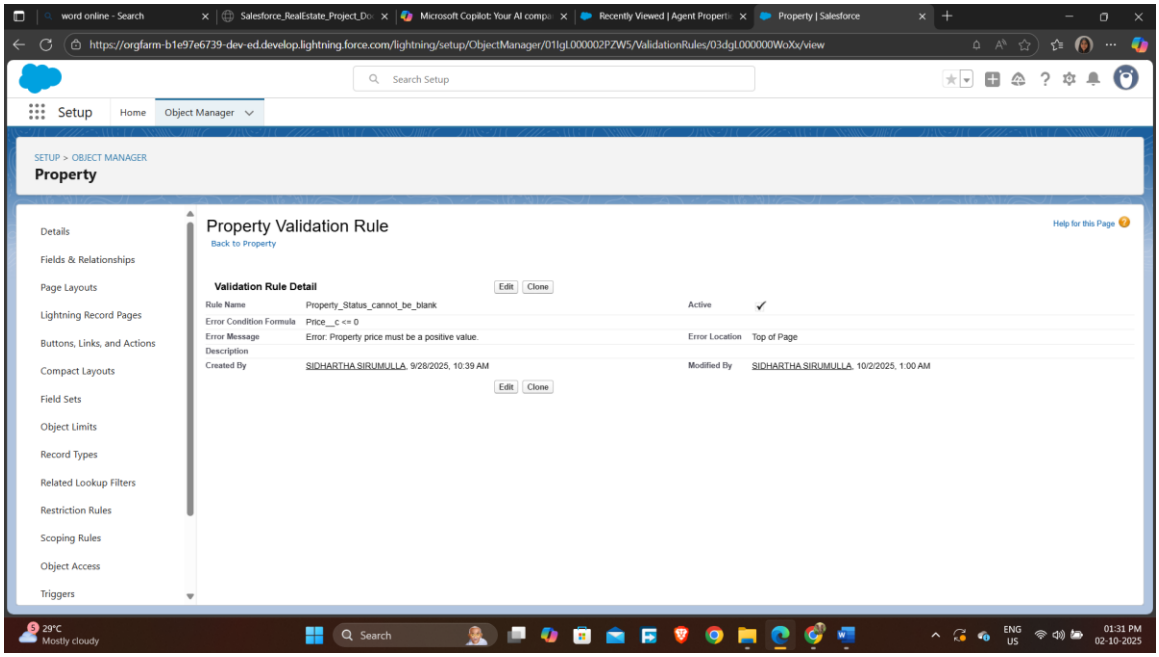
- **Booking__c.Amount > 0**

```
Amount__c <= 0
Error: Booking amount must be greater than zero.
```



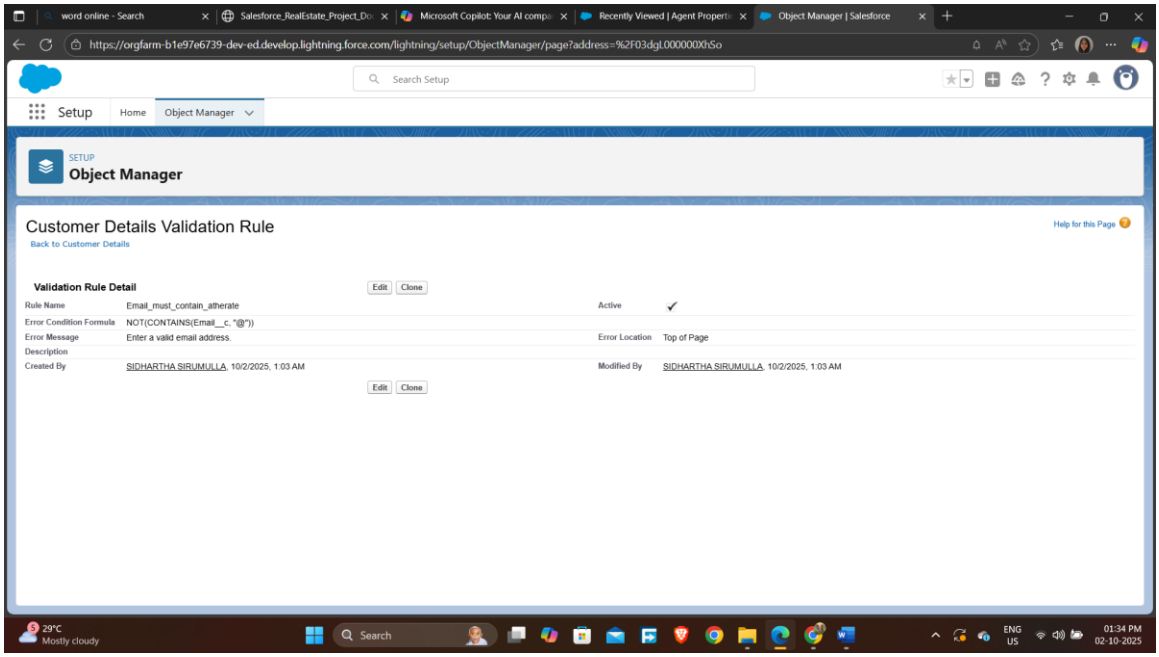
- **Property__c.Price must be positive**

```
Price__c <= 0
Error: Property price must be a positive value.
```



- **Customer__c.Email must contain “@”**

```
NOT(CONTAINS(Email__c, "@"))
Error: Enter a valid email address.
```



🔧 Setup:

- Go to Setup → Object Manager → [Object] → Validation Rules → New
- Define rule name, formula, and error message
- Activate the rule

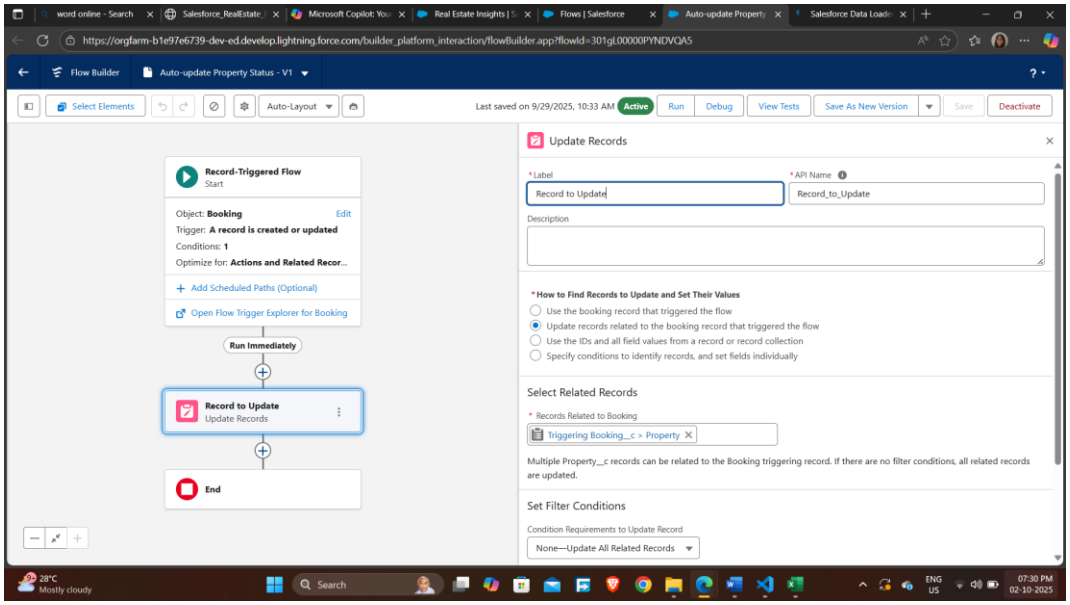
🔁 2. Record-Triggered Flow — Auto-Update Property Status

Goal: When a Booking is confirmed, update the related Property's status to "Booked".

🔧 Setup:

- Go to Setup → Flows → New Flow → Record-Triggered Flow
- Object: Booking__c
- Trigger: When record is updated
- Condition: Status__c = "Confirmed"
- Action:

    - Get related Property__c via Booking__c.Property__c
    - Update Property__c.Status__c to “Booked”
- Save and Activate

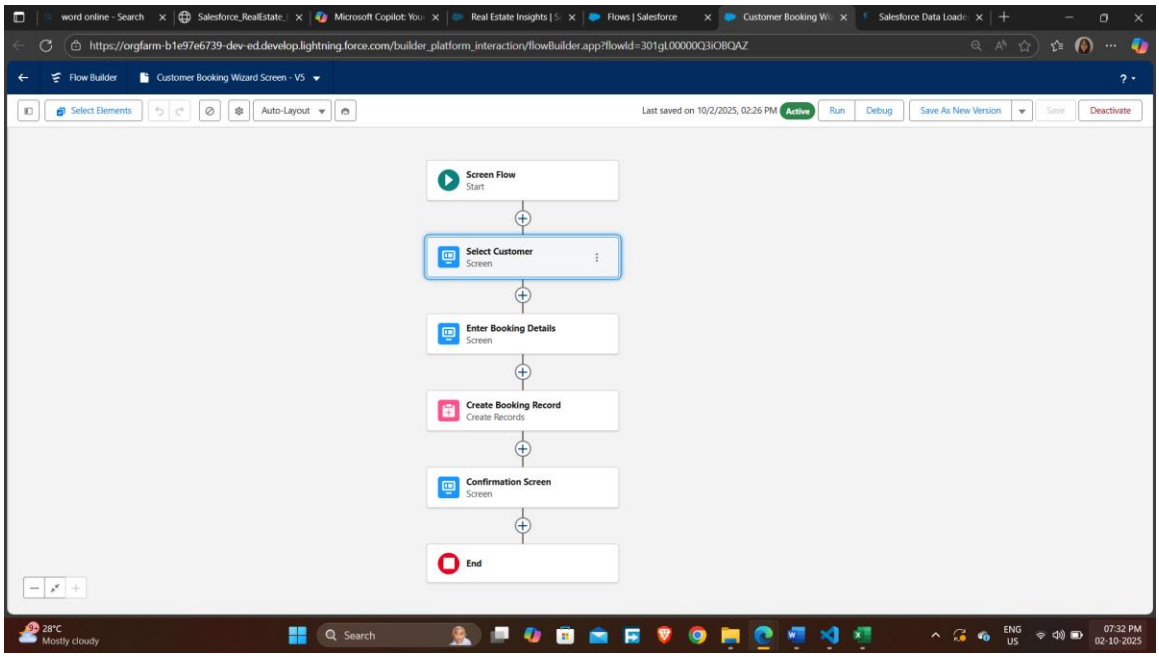


3. Screen Flow — Customer Booking Wizard

Goal: Guide users through booking creation with multiple steps.

Setup:

- Go to Setup → Flows → New Flow → Screen Flow
- Step 1: Select Customer (Lookup)
- Step 2: Choose Property (Lookup)
- Step 3: Enter Booking Details (Date, Amount, Status)
- Step 4: Confirmation screen
- Add navigation buttons (Next, Previous)
- Save and Activate
- Add to Lightning Page or Utility Bar for access

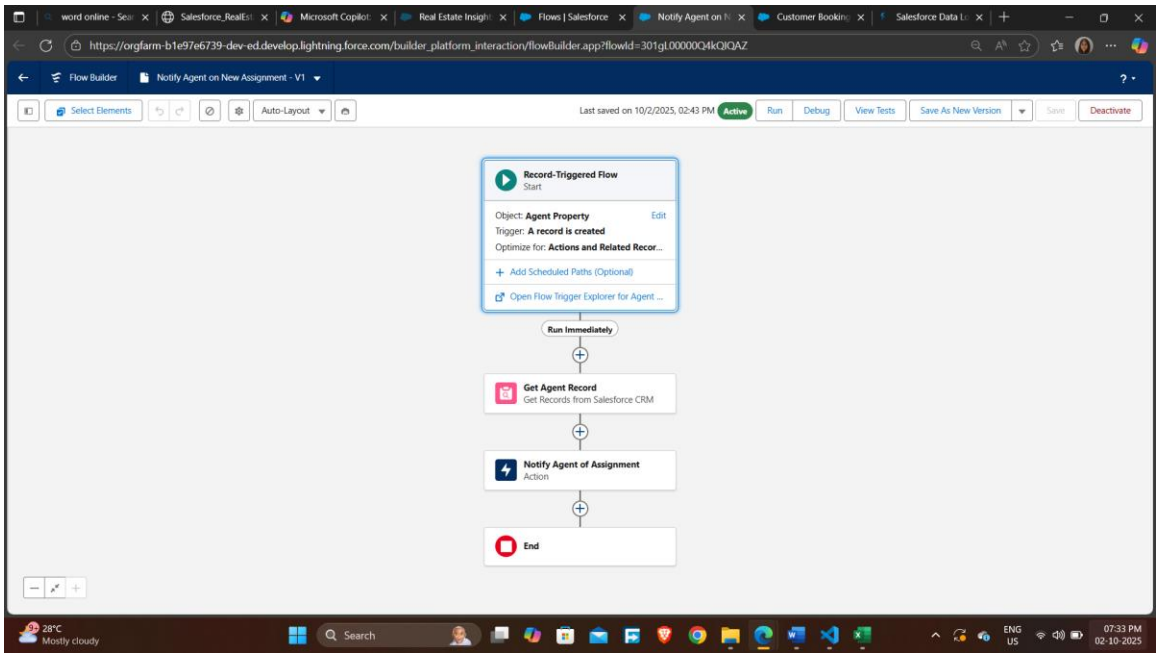


📣 4. Auto-Launched Flow — Notify Agent on Assignment

Goal: When an Agent_Property__c record is created, notify the assigned Agent.

🔧 Setup:

- Go to Setup → Flows → New Flow → Auto-Launched Flow
- Trigger: Record-Triggered Flow on Agent_Property__c (when created)
- Get Agent__c record
- Action: Send Email Alert or Post to Chatter
    - Use Email Template or custom message
- Save and Activate



Phase 5 — Apex Programming (Developer)

⚙ Apex Trigger: Booking Date Validation

Goal: Prevent creation of Booking__c records with a past date.

◇ Trigger: BookingTrigger

```
trigger BookingTrigger on Booking__c (before insert) {
    if (Trigger.isBefore && Trigger.isInsert) {
        BookingHandler.validateBookingDates(Trigger.new);
    }
}
```

🧠 Apex Handler Class: BookingHandler

Purpose: Encapsulate logic to check if booking date is in the future.

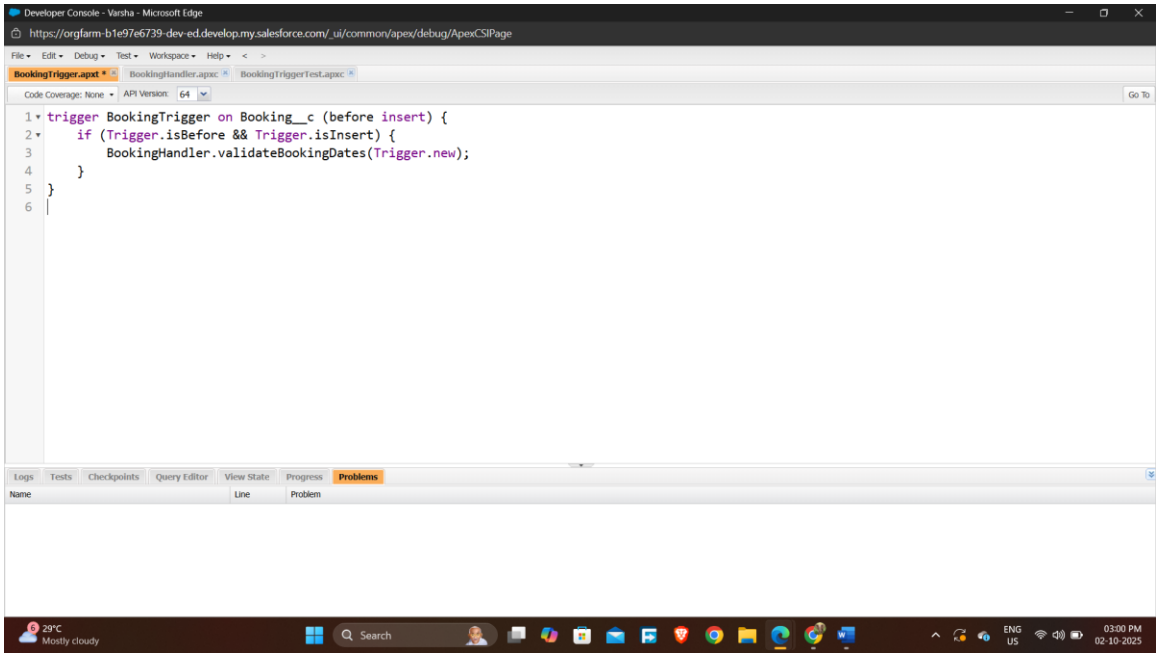
```
public class BookingHandler {
    public static void validateBookingDates(List<Booking__c> bookings)
{
        for (Booking__c booking : bookings) {
            if (booking.Date__c != null && booking.Date__c <
Date.today()) {
                booking.addError('Booking date must be in the
future.');
            }
        }
    }
}
```

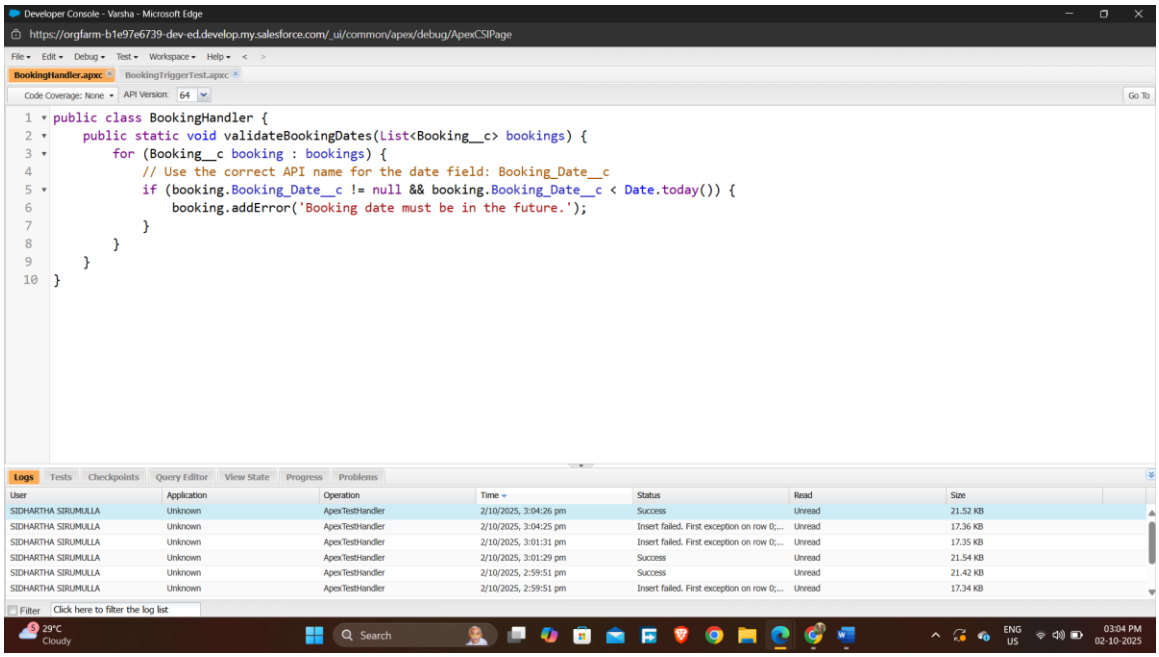
🧪 Apex Test Class: BookingTriggerTest

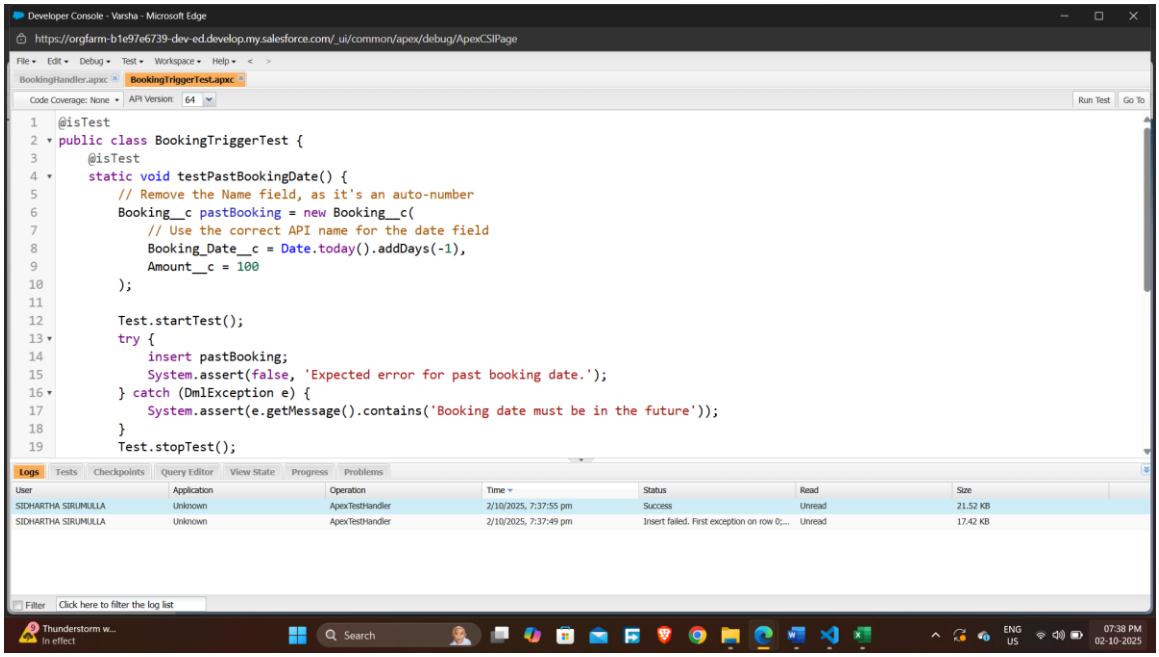
Purpose: Ensure trigger prevents past-dated bookings and allows valid ones.

```
@isTest
public class BookingTriggerTest {
    @isTest
    static void testPastBookingDate() {
        Booking__c pastBooking = new Booking__c(
            Name = 'Test Past Booking',
            Date__c = Date.today().addDays(-1),
            Amount__c = 100
        );
        Test.startTest();
        try {
            insert pastBooking;
            System.assert(false, 'Expected error for past booking
date.');
        } catch (DmlException e) {
            System.assert(e.getMessage().contains('Booking date must be
in the future'));
        }
```

```
        Test.stopTest();
    }

    @isTest
    static void testValidBookingDate() {
        Booking__c futureBooking = new Booking__c(
            Name = 'Test Future Booking',
            Date__c = Date.today().addDays(1),
            Amount__c = 200
        );
        Test.startTest();
        insert futureBooking;
        Test.stopTest();
        System.assertNotEquals(null, futureBooking.Id);
    }
}
```




Developer Console - Varsha - Microsoft Edge
https://orgfarm-b1e97e6739-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
BookingHandler.apxc
BookingTriggerTest.apxc
public class BookingHandler {
public static void validateBookingDates(List<Booking__c> bookings) {
for (Booking__c booking : bookings) {
// Use the correct API name for the date field: Booking_Date__c
if (booking.Booking_Date__c != null && booking.Booking_Date__c < Date.today()) {
booking.addError('Booking date must be in the future.');
}
}
}
}
Logs
Tests
Checkpoints
Query Editor
View State
Progress
Problems
User
Application
Operation
Time
Status
Read
Size
SIDHARTHA SIRUMULLA
Unknown
ApexTestHandler
2/10/2025, 3:04:26 pm
Success
Unread
21.52 KB
2/10/2025, 3:04:25 pm
Insert failed. First exception on row 0;...
17.36 KB
2/10/2025, 3:01:31 pm
17.35 KB
2/10/2025, 3:01:29 pm
21.54 KB
2/10/2025, 2:59:51 pm
21.42 KB
17.34 KB
Filter
Click here to filter the log list
29°C
Cloudy
Search
ENG
US
03:04 PM
02-10-2025


Developer Console - Varsha - Microsoft Edge
https://orgfarm-b1e97e6739-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
BookingHandler.apxc
BookingTriggerTest.apxc
Run Test
Go To
@isTest
public class BookingTriggerTest {
@isTest
static void testPastBookingDate() {
// Remove the Name field, as it's an auto-number
Booking__c pastBooking = new Booking__c(
// Use the correct API name for the date field
Booking_Date__c = Date.today().addDays(-1),
Amount__c = 100
);
Test.startTest();
try {
insert pastBooking;
System.assert(false, 'Expected error for past booking date.');
} catch (DmlException e) {
System.assert(e.getMessage().contains('Booking date must be in the future'));
}
Test.stopTest();
Logs
Tests
Checkpoints
Query Editor
View State
Progress
Problems
User
Application
Operation
Time
Status
Read
Size
SIDHARTHA SIRUMULLA
Unknown
ApexTestHandler
2/10/2025, 7:37:55 pm
Success
Unread
21.52 KB
2/10/2025, 7:37:49 pm
Insert failed. First exception on row 0;...
17.42 KB
Filter
Click here to filter the log list
Thunderstorm w...
in effect
Search
ENG
US
07:38 PM
02-10-2025

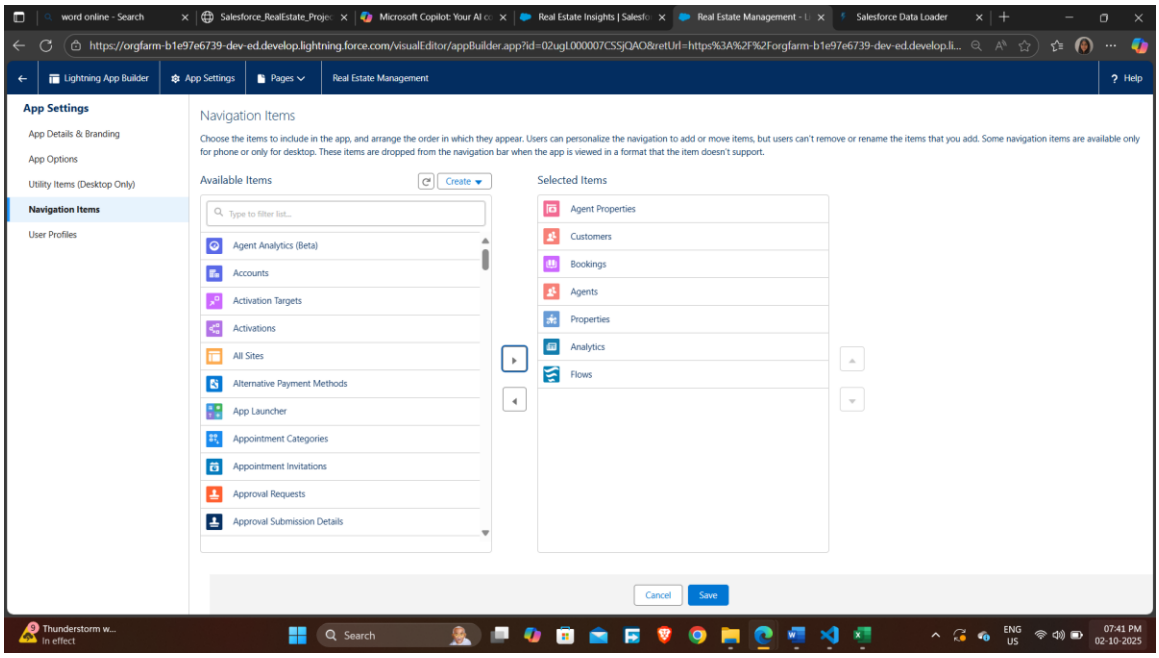
Phase 6 — User Interface Development

🏢 Step 1: Create Lightning App — Real Estate Management

Purpose: Centralize access to custom objects like Property, Booking, Customer, Agent.

🔧 Setup:

1. Go to **Setup → App Manager → New Lightning App**
2. **App Name**: Real Estate Management
3. **Navigation Style**: Standard (tab-based)
4. **Add Tabs**:
    a. Property__c
    b. Booking__c
    c. Customer__c
    d. Agent__c
5. Assign to relevant profiles
6. Save and Activate

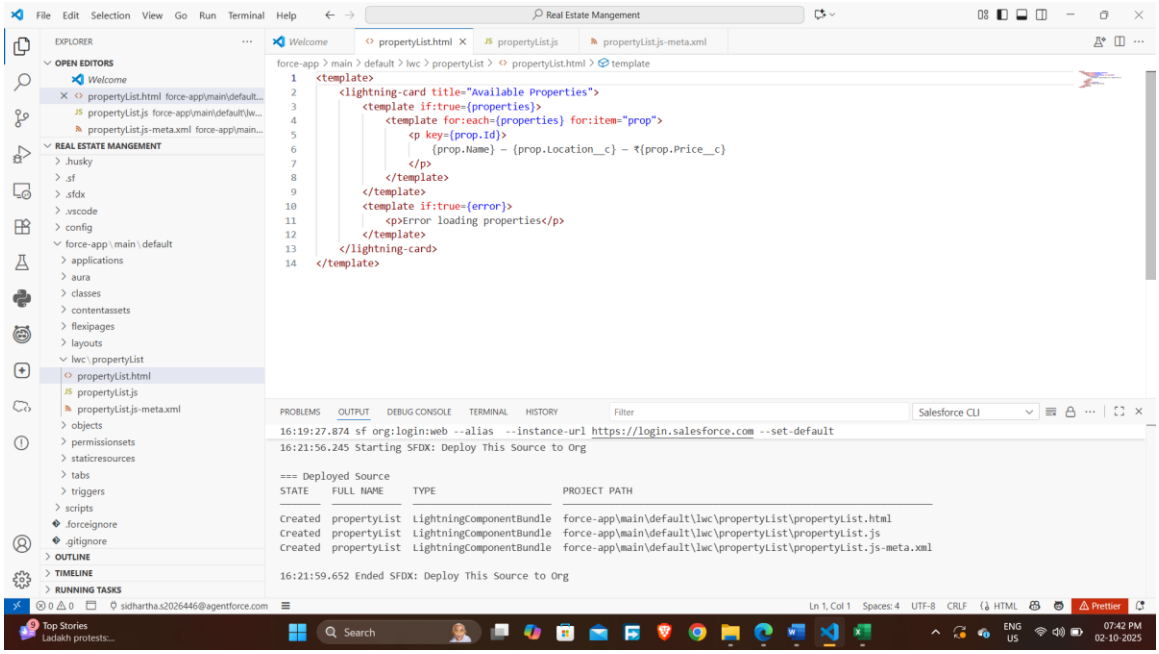


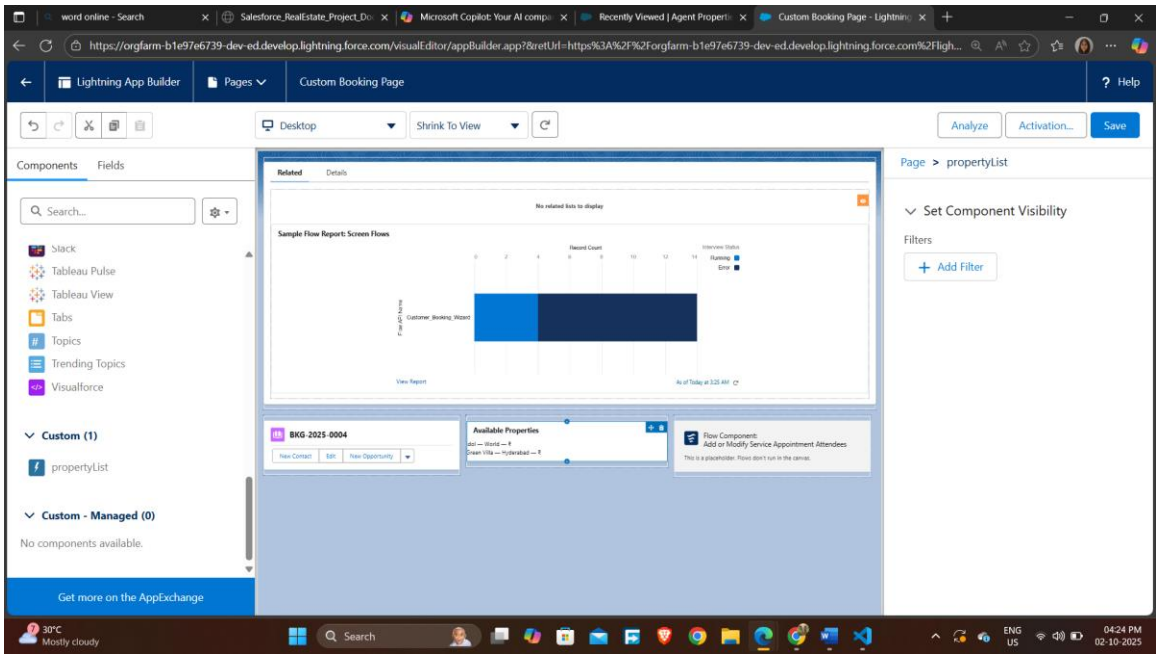
🎨 Step 2: Customize Record Pages with App Builder

Purpose: Enhance user experience with dynamic layouts and components.

🔧 Setup:

1. Go to **Setup → Object Manager → [Object] → Lightning Record Pages**
2. Click **New → Record Page**
3. Use **App Builder** to:
    a. Add components (e.g., Related Lists, Highlights Panel, Custom LWCs)
    b. Configure visibility rules
    c. Embed screen flows or dashboards
4. Save and Activate for desired app/profile





Step 3: LWC Example — `propertyList`

Goal: Display available properties using Apex service.

◇ Apex Class: PropertyService

```
public with sharing class PropertyService {
    @AuraEnabled(cacheable=true)
    public static List<Property__c> getAvailableProperties() {
        return [SELECT Id, Name, Location__c, Price__c, Status__c
                FROM Property__c
                WHERE Status__c = 'Available'];
    }
}
```

◇ LWC: propertyList

Files:

- propertyList.html
- propertyList.js
- propertyList.js-meta.xml

propertyList.html

```
<template>
    <lightning-card title="Available Properties">
        <template if:true={properties}>
            <template for:each={properties} for:item="prop">
                <p key={prop.Id}>
                    {prop.Name} – {prop.Location__c} – ₹{prop.Price__c}
                </p>
            </template>
        </template>
        <template if:true={error}>
            <p>Error loading properties</p>
        </template>
    </lightning-card>
</template>
```

propertyList.js

```
import { LightningElement, wire } from 'lwc';
import getAvailableProperties from
```

```
'@salesforce/apex/PropertyService.getAvailableProperties';

export default class PropertyList extends LightningElement {
    properties;
    error;

    @wire(getAvailableProperties)
    wiredProperties({ error, data }) {
        if (data) {
            this.properties = data;
            this.error = undefined;
        } else if (error) {
            this.error = error;
            this.properties = undefined;
        }
    }
}
```

propertyList.js-meta.xml

```
<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>
```

Phase 7 — Integration & External Access

🔐 Step 1: Create Named Credential for External API

Purpose: Securely store endpoint and authentication for external services.

🔧 Setup:

1. Go to **Setup → Named Credentials → New Named Credential**
2. **Fields to configure**:
    a. **Label**: PaymentService
    b. **Name**: PaymentService
    c. **URL**: `https://api.paymentprovider.com` (replace with actual endpoint)
    d. **Identity Type**: Named Principal (or per-user if needed)
    e. **Authentication Protocol**: Choose based on API (e.g., OAuth 2.0, Password Authentication)
    f. **Generate Authorization Header**: Checked (if required)
    g. **Custom Headers**: Add if API requires specific headers
3. Save and test connection (ensure endpoint is reachable)

☸️ Step 2: Apex Wrapper for Payment Service Using HTTP Callout

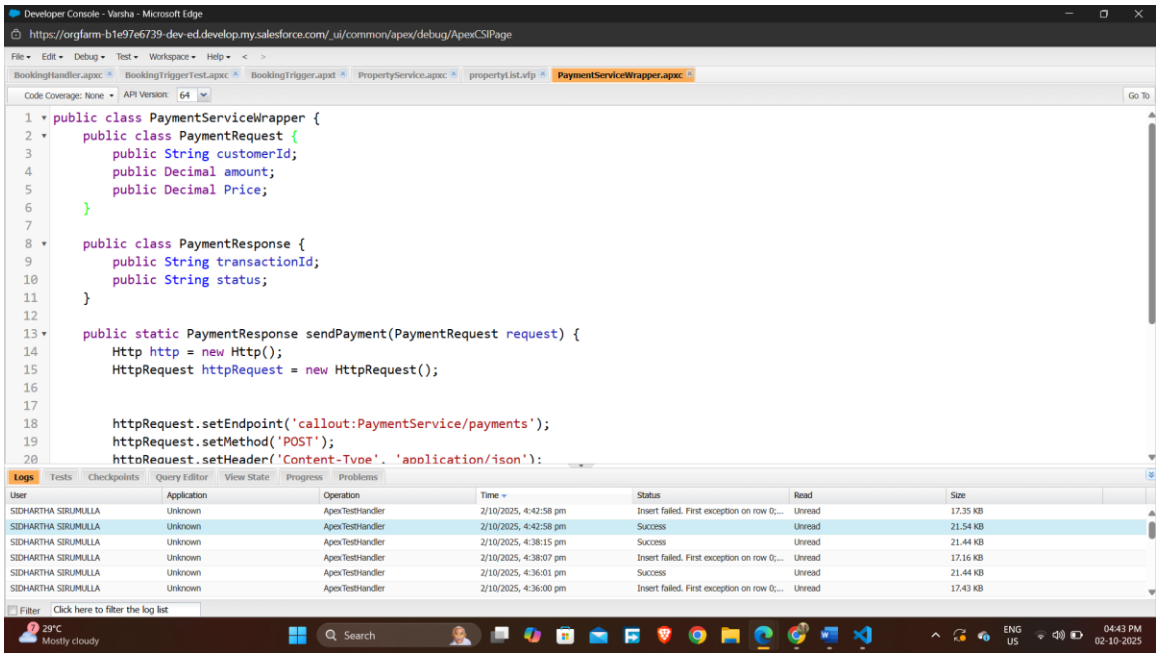
Goal: Send payment request to external API using Named Credential.

◇ Apex Class: `PaymentServiceWrapper`

```
public class PaymentServiceWrapper {
    public class PaymentRequest {
        public String customerId;
        public Decimal amount;
        public String currency;
    }

    public class PaymentResponse {
        public String transactionId;
        public String status;
    }

    public static PaymentResponse sendPayment(PaymentRequest request) {
        Http http = new Http();
        HttpRequest httpRequest = new HttpRequest();

        httpRequest.setEndpoint('callout:PaymentService/payments');
        httpRequest.setMethod('POST');
        httpRequest.setHeader('Content-Type', 'application/json');
        httpRequest.setBody(JSON.serialize(request));

        HttpResponse response = http.send(httpRequest);

        if (response.getStatusCode() == 200) {
            return (PaymentResponse)
JSON.deserialize(response.getBody(), PaymentResponse.class);
        } else {
            throw new CalloutException('Payment API failed: ' +
response.getBody());
        }
    }
}
```

◇ Notes:

- Replace `/payments` with the actual resource path.
- Use `callout:PaymentService` to reference the Named Credential.
- Handle errors gracefully with try-catch or custom exceptions.



🩹 Optional: Test Class for Callout

Use `HttpCalloutMock` to simulate responses in test context.

```
@isTest
global class PaymentServiceMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest req) {
        HttpResponse res = new HttpResponse();
        res.setHeader('Content-Type', 'application/json');
        res.setBody('{"transactionId":"TX123","status":"Success"}');
        res.setStatusCode(200);
        return res;
    }
}
 @isTest
private class PaymentServiceWrapperTest {
    @isTest
    static void testSendPayment() {
        Test.setMock(HttpCalloutMock.class, new PaymentServiceMock());

        PaymentServiceWrapper.PaymentRequest req = new
PaymentServiceWrapper.PaymentRequest();
        req.customerId = 'C001';
        req.amount = 500.00;
```

```
        req.currency = 'INR';

        Test.startTest();
        PaymentServiceWrapper.PaymentResponse res =
PaymentServiceWrapper.sendPayment(req);
        Test.stopTest();

        System.assertEquals('Success', res.status);
    }
}
```

Phase 8 — Data Management & Deployment

📥 Step 1: Prepare Import Templates (Data Loader / SFDX)

◇ CSV Templates

Create structured CSV files for each object with required fields and relationships:

Object	Required Fields Example
Property__c	Name, Location__c, Price__c, Status__c
Customer__c	Name, Email__c, Phone__c
Booking__c	Name, Date__c, Amount__c, Status__c, Property__c ID, Customer__c ID
Agent__c	Name, Email__c, Phone__c
Agent_Property__c	Agent__c ID, Property__c ID

◇ Import Tools

- **Data Loader**: Use for bulk import via CSV.
  - Download from Setup → Data Loader
  - Map fields carefully
  - Use external IDs or Salesforce IDs for relationships
- **SFDX Data Import**:
  - Use `data import` plugin or `force:data:tree:import` for hierarchical data
  - Example command: `sfdx force:data:tree:import -f data/Property__c.json`

🚀 Step 2: Deployment via SFDX (Salesforce DX)

◇ Setup GitHub Repo

- Create a GitHub repository for your Salesforce project
- Organize folders:
```
├── force-app/
│   ├── main/
│   │   ├── default/
│   │   │   ├── objects/
│   │   │   ├── classes/
│   │   │   ├── triggers/
```

```
│   │   │   ├── lwc/
│   │   │   ├── flows/
├── data/
├── scripts/
├── sfdx-project.json
```

◇ SFDX Workflow

1. **Pull Metadata from Sandbox/Dev Org**

```
sfdx force:source:retrieve -u DevOrgAlias -p force-app
```

2. **Commit to GitHub**

```
git add .
git commit -m "Initial metadata commit"
git push origin main
```

3. **Deploy to Target Org**

```
sfdx force:source:deploy -u TargetOrgAlias -p force-app
```

4. **Run Test**

```
sfdx force:apex:test:run -u TargetOrgAlias --resultformat human
```

Phase 9 — Reporting & Dashboards + Security Review

📊 1. Create Custom Report Types

Purpose: Enable reporting on custom objects and relationships.

🔧 Setup:

- Go to **Setup → Report Types → New Custom Report Type**
- Define:

    - **Primary Object**: e.g., Booking__c
    - **Related Object**: e.g., Property__c (via Master-Detail)
    - **Deployment Status**: Deployed
- Examples:
    - **Bookings with Properties**
    - **Agent_Property with Agent and Property**
    - **Bookings with Customers**

📈 2. Build Reports

Use **Report Builder** to create and customize reports:

🔹 Bookings by Status

- Report Type: Bookings
- Group by: Status__c
- Add filters: Date range, Property__c
- Add chart: Donut or bar chart for visual summary

🔹 Revenue by Property

- Report Type: Bookings with Property
- Group by: Property__c.Name
- Summarize: SUM of Amount__c
- Add chart: Column chart for revenue comparison

🔹 Top Agents

- Report Type: Agent_Property with Agent
- Group by: Agent__c.Name
- Add row count or custom metric (e.g., total properties assigned)
- Add chart: Horizontal bar chart

📊 3. Build Dashboard — Real Estate Insights

Purpose: Visualize KPIs and trends in one place.

🔧 Setup:

- Go to **Dashboards → New Dashboard**
- Name: Real Estate Insights
- Folder: Shared or Private
- Add Components:
  - **Bookings by Status** (Donut chart)
  - **Revenue by Property** (Column chart)
  - **Top Agents** (Bar chart)
  - **Total Bookings** (Metric)
  - **Average Booking Amount** (Gauge)
- Set filters:
  - Date range
  - Property__c
  - Agent__c

🔐 4. Security Review

Ensure data access and org security are properly configured:

◇ Field-Level Security (FLS)

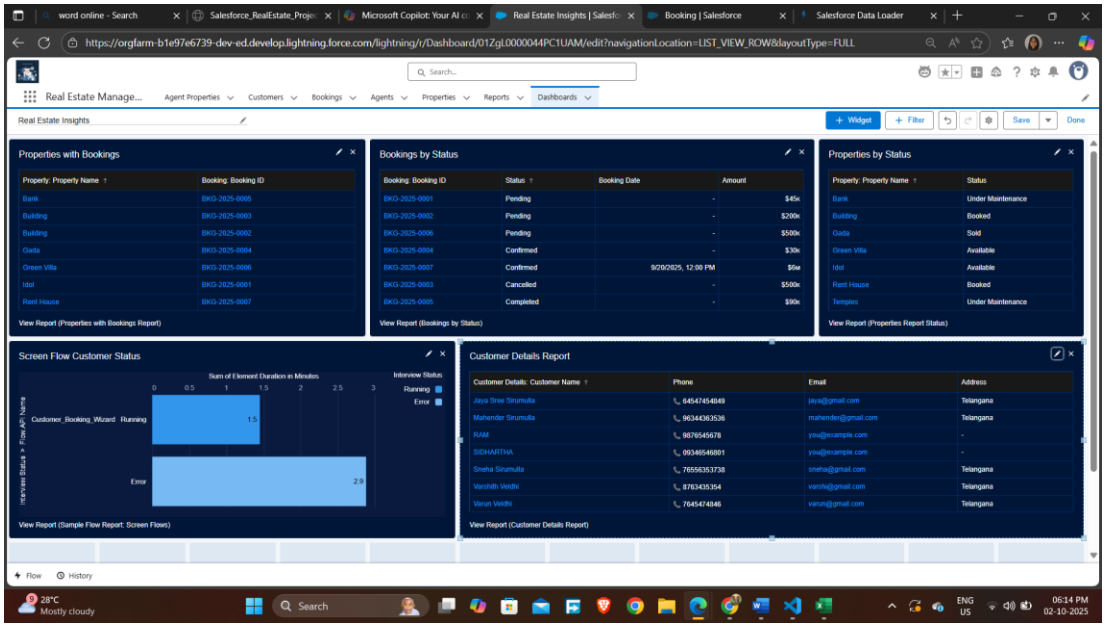
- Go to **Object Manager → [Object] → Fields & Relationships**
- Click each field → Set field-level visibility per profile
- Ensure sensitive fields (e.g., Amount__c, Email__c) are restricted as needed

◇ Session Settings

- Setup → Session Settings
- Recommended:
  - Enable IP address locking
  - Set session timeout (e.g., 30 minutes)
  - Require secure connections (HTTPS)

◇ Login IP Ranges

- Setup → Profiles → [Profile Name] → Login IP Ranges
- Define trusted IP ranges for internal users
- Prevent access from unknown networks



Phase 10 — Final Presentation & Handoff

Goal: Demonstrate solution & provide docs.

Key Deliverable: Presentation deck, demo script, handoff documentation.

Steps:

1. Prepare pitch deck (Problem, Solution, Architecture, Demo, Metrics).

2. Provide documentation (Admin Guide, Deployment Guide, Test Cases, GitHub Repo).

-term success.