

PropertyMatch- Real Estate CRM

Project Overview

PropertyMatch – Real Estate CRM is a Salesforce-based application that helps real estate companies manage properties, brokers, and customer interactions in one place. It allows users to search and filter properties by city, price, bedrooms, and bathrooms, and view details like images, broker information, and location on Google Maps. Administrators can create and assign properties to brokers, while automated email alerts keep brokers informed of new assignments. The system also includes reports and dashboards for insights, validation rules and duplicate checks for data quality, and security controls like profiles, roles, and sharing rules. Overall, PropertyMatch makes property management more efficient, transparent, and user-friendly.

Project Objectives

The objective of this Real Estate CRM is to provide a centralized platform for managing properties, brokers, and client interactions. The system enables users to filter and search properties by city, price, bedrooms, and bathrooms, improving the customer experience by offering quick and accurate results. By allowing administrators to create properties and assign brokers, and by automating broker notifications through email alerts, the project ensures faster communication and reduced manual follow-up. Integration with Google Maps adds transparency by showing exact property locations. Overall, this solution brings business value by improving customer management, streamlining the property search and assignment process, enhancing broker accountability, and ensuring better operational efficiency in the real estate workflow.

Phase 1: Problem Understanding & Industry Analysis

1. Requirement Gathering

- Buyers need property search by **city, price, bedrooms, bathrooms**.
- Display **property details, images, broker info, and map location**.
- Admins should **create properties and assign brokers**.
- **Email alerts** must notify brokers on assignment.
- Navigation: **Home Page, Property Records, Broker Records**.

2. Stakeholder Analysis

- **Primary:** Buyers (search/filter properties), Brokers (assigned properties, email alerts), Admins (manage properties/brokers).
- **Secondary:** Salesforce Platform (data & automation), Email Services (notifications).

3. Business Process Mapping

- *Current (Manual):* Scattered property info, no map visibility, no broker automation.
- *Proposed (Automated):* LWC-based search filters → Property Records → Broker assignment → Google OAuth 2.0 maps → Email alerts.

4. Industry-Specific Use Case Analysis

- Existing apps: generic property listings.
- Gaps: lack of Salesforce-native solution with **map integration + broker automation**.
- Solution: Custom Salesforce LWC app with filters, maps, and email alerts.

5. AppExchange Exploration

- Explored property apps.
- Found missing features (Google maps, broker email automation).
- Decision: **Custom build on Salesforce CRM.**

Phase 2: Org Setup & Configuration

1. Salesforce Edition & Org Setup

- Created a **Developer Org** for the Dreamhouse LWC project.
- Used it for building, testing, and deploying CRM components.

The screenshot shows a terminal window with the following content:

```
PS C:\Users\manog\PropertyMatch-lwc> sf org open
Opening org 00DgL000007pvbrUAA as user manogna.bandlamudi99804@agentforce.com
Waiting to resolve the Lightning Experience-enabled custom domain..... done
PS C:\Users\manog\PropertyMatch-lwc> sf org list
```

	Alias	Username	Org Id	Status
*		manogna.bandlamudi99804@agentforce.com	00DgL000007pvbrUAA	Connected

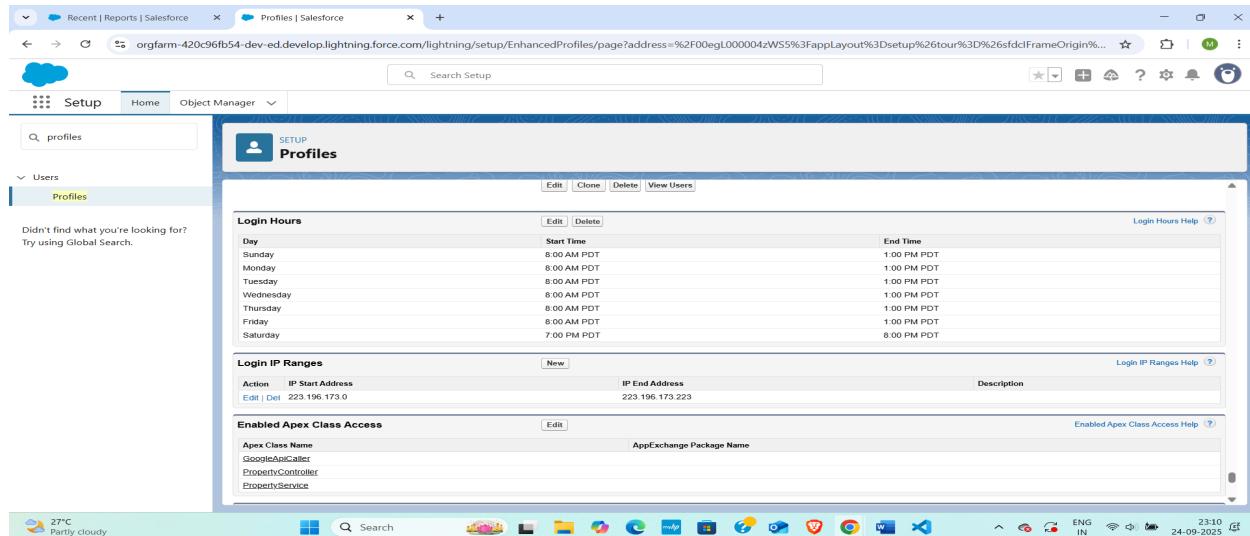
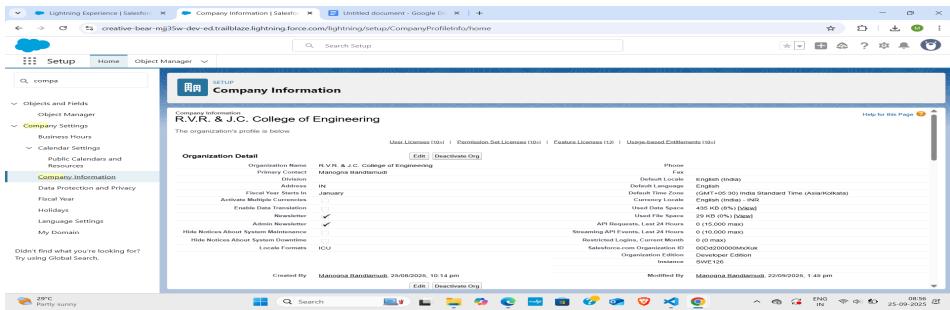
Legend: 🌱=Default DevHub, 🌸=Default Org Use --all to see expired and deleted scratch orgs

```
PS C:\Users\manog\PropertyMatch-lwc>
```

99804@agentforce.com Connect Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} XML Go Live

2. Company Profile Setup

- Navigation: **Setup → Company Information → Edit.**
- Updated organization details (Name, Default Time Zone).
- Purpose: Ensure correct org identity and time settings for automation.



3. Business Hours & Holidays

- Navigation: **Setup → Business Hours → New.**
- Configured business hours (09:00 – 18:00).
- Added holidays (Independence Day, Republic Day).
- Purpose: Time-based automations respect working days.

4. Role Hierarchy

- Your final roles will look like:
 - Property Agency Manager(RVRJCCE)
 - Property Agent
 - Buyer
 - Property Owner

Creating the Role Hierarchy

You can build on the existing role hierarchy shown on this page. To insert a new role, click **Add Role**.

Your Organization's Role Hierarchy

R.V.R. & J.C. College of Engineering

- + Add Role
- + CEO Edit | Del | Assign
- + Add Role
- + Property Agent Edit | Del | Assign
- + Add Role
- + Buyer Edit | Del | Assign
- + Add Role
- + Property Owner Edit | Del | Assign
- + Add Role

Help for this Page ⓘ Show in tree view ▾

5. User Setup & Licenses

- Navigation: **Setup** → **Users** → **New User**.
- Created different users:
 - **Admin** → manages properties and brokers.
 - **Broker** → receives property assignments and email alerts.
 - **Property Manager** → manages records and assignments.

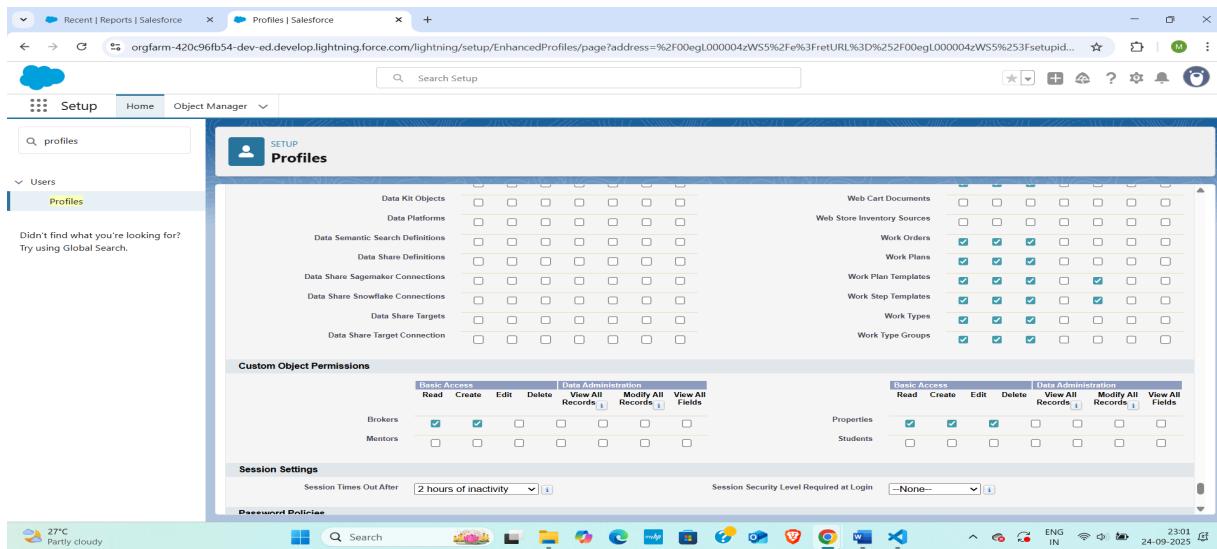
- Assigned profiles and licenses to each user.

Action	Full Name	Alias	Username	Role	Active	Profile
<input type="checkbox"/>	Bandimuthi, Manoona	mband	manoona.bandimuthi@creative-bear-mj35w.com	Manager	<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	Brown, Courtney	cbrow	cbrown@apchive2419.com	Buyer	<input checked="" type="checkbox"/>	Standard Platform User
<input type="checkbox"/>	Buyer, Buyer	Buyer	manoona.bandimuthi999@gmail.com	Buyer	<input checked="" type="checkbox"/>	Standard Platform User
<input type="checkbox"/>	Chatter Expert	Chatter	chaty.00d200000mxukear@m4k0p0x0kg@chatter.salesforce.com	Buyer	<input checked="" type="checkbox"/>	Chatter Free User
<input type="checkbox"/>	User_Interaction	inter	integration@00d200000mxukear.com	Buyer	<input checked="" type="checkbox"/>	Analytics Cloud Integration User
<input type="checkbox"/>	User_Security	sec	insightssecure@00d200000mxukear.com	Property Owner	<input checked="" type="checkbox"/>	Analytics Cloud Security User
<input type="checkbox"/>	user_test	testuser	testusers196@gmail.com	Property Owner	<input checked="" type="checkbox"/>	Standard User
<input type="checkbox"/>	Wheeler, Allison	awhee	awheeler@apchive05239.com	VP_North American Sales	<input checked="" type="checkbox"/>	Standard Platform User

6. Configure Profiles (Object Permissions)

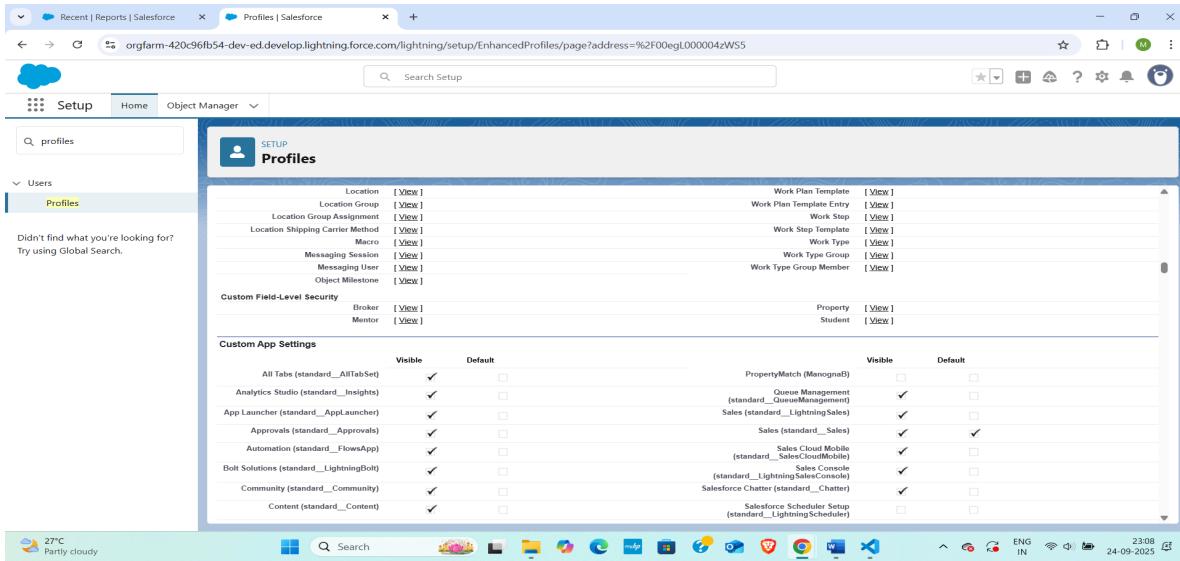
Quick Find → Profiles → open a profile (e.g., Standard User), Scroll to Object Settings → select object Property_c → click Edit. Grant permissions: Read, Create, Edit (as required), Click Save

- For System Administrator Profile: Full access enabled by default.
- For Standard User Profile:
 - Object Permissions (after creating objects): Read, Create, Edit on Property, Buyer, and Visit objects.
 - **Field-Level Security** to allow key fields visible/editable (after objects are created).



7. Login Access Policies

- Navigation: **Setup** → **Login Access Policies**.
- Enabled admin login as other users for testing.



8. Sandbox / Deployment

- Used **VS Code** with **SFDX** for source push and pull.
- Deployment steps:
 - Edited components in VS Code.
 - Command: `sfdx force:source:push` to deploy changes.

Phase 3: Data Modeling & Relationships

1. Custom Objects Created

- Navigation: **Setup** → **Object Manager** → **Create** → **Custom Object**
- Objects:
 - **Property__c** → Stores property details (city, state, price, bedrooms, bathrooms, latitude, longitude, broker, image).
 - **Broker__c** → Stores broker details (name, email, phone, assigned properties).

2. Custom Fields (Property__c)

- City__c (Text) – City name.
- State__c (Text) – State name.
- Price__c (Currency) – Property price.
- Bedrooms__c (Number) – No. of bedrooms.
- Bathrooms__c (Number) – No. of bathrooms.
- Latitude__c / Longitude__c (Number) – Coordinates for map display.
- Image_URL__c (Text) – Store property image link.
- Broker__c (Lookup → Broker__c) – Assigned broker.

Custom Fields (Broker__c)

- Email__c (Email) – Broker's email (used in email alerts).
- Phone__c (Phone) – Contact number.
- Assigned_Properties (Related List from Property__c).

3. Relationships

- **Lookup:** Property__c → Broker__c (each property linked to one broker).
- **Master-Detail:** Not required, as broker deletion should not delete properties.

Property

Fields & Relationships
29 Items, Sorted by Field Label

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Address	Address_c	Text(100)		
Asking Price	Price_c	Currency(8, 0)		
Assessed Value	Assessed_Value_c	Currency(18, 0)		
Baths	Baths_c	Number(2, 0)		
Beds	Beds_c	Number(2, 0)		
Broker	Broker_c	Lookup(Broker)		
City	City_c	Text(50)		
Created By	CreatedById	Lookup(User)		
Date Agreement	Date_Agreement_c	Date		

Broker

Fields & Relationships
11 Items, Sorted by Field Label

FIELD LABEL	FIELD NAME	DATA TYPE	CONTROLLING FIELD	INDEXED
Broker Id	Broker_Id_c	Number(18, 0) (External ID)		
Broker Name	Name	Text(80)		
Created By	CreatedById	Lookup(User)		
Email	Email_c	Email		
Last Modified By	LastModifiedById	Lookup(User)		
Mobile Phone	Mobile_Phone_c	Phone		
Owner	OwnerId	Lookup(User,Group)		
Phone	Phone_c	Phone		
Picture	Picture_IMG_c	Formula (Text)		
	Picture_c	Image (255)		

4. Page Layouts

- Navigation: **Setup → Object Manager → [Object] → Page Layouts**
- **Property Layout:** Show all fields + Related List (Broker).
- **Broker Layout:** Show contact info + Related List (Assigned Properties).

5. Record Types & Compact Layouts

- Created default Record Type for Property__c.
- Compact Layout: Display key fields (City, Price, Bedrooms, Bathrooms) in highlights panel.

The screenshot shows the Salesforce Object Manager interface. On the left, a sidebar lists various setup options like Details, Fields & Relationships, Page Layouts, and Compact Layouts. The Compact Layouts section is currently selected. In the main area, a new compact layout named 'Broker Compact' is being created. The layout detail page shows the label 'Broker Compact', API name 'Broker_Compact', and included fields: 'Broker Name', 'Title', 'Phone', 'Mobile Phone', and 'Email'. The layout was created by 'Bandiamudi Manogna' on 9/15/2025 at 6:44 AM and modified by the same user on the same date and time. The bottom of the screen shows the standard Salesforce navigation bar.

6. Schema Builder

- Navigation: **Setup → Schema Builder**.
- Verified relationship between **Property__c** and **Broker__c**.

The screenshot shows the Salesforce Schema Builder interface. The top navigation bar includes tabs for Home, Create Property - V2, Email Alerts, and SchemaBuilder/home. The main area displays two objects: 'Broker' and 'Property'. The 'Broker' object has fields like 'Broker ID', 'Broker Name', 'Email', 'Mobile Phone', 'Phone', and 'Title'. The 'Property' object has fields like 'Address', 'Assessed Value', 'Bedrooms', 'Beds', 'City', 'Comments', 'Days On Market', 'Last Modified By', 'Main Picture', 'Max Price', 'Min Price', 'Owner', 'Price Sold', 'Price Type', and 'Record Link'. A legend on the right side of the diagram indicates that blue lines represent 'Lookup Relationship', red lines represent 'Master-Detail Relationship', and orange lines represent 'Required Field'. The bottom of the screen shows the standard Salesforce navigation bar.

7. Add Sample Records

- **Properties:**
 - Property 1 → City: Hyderabad, Price: 50,00,000, Bedrooms: 3, Bathrooms: 2, Status: Available
 - Property 2 → City: Bangalore, Price: 75,00,000, Bedrooms: 4, Bathrooms: 3, Status: Available

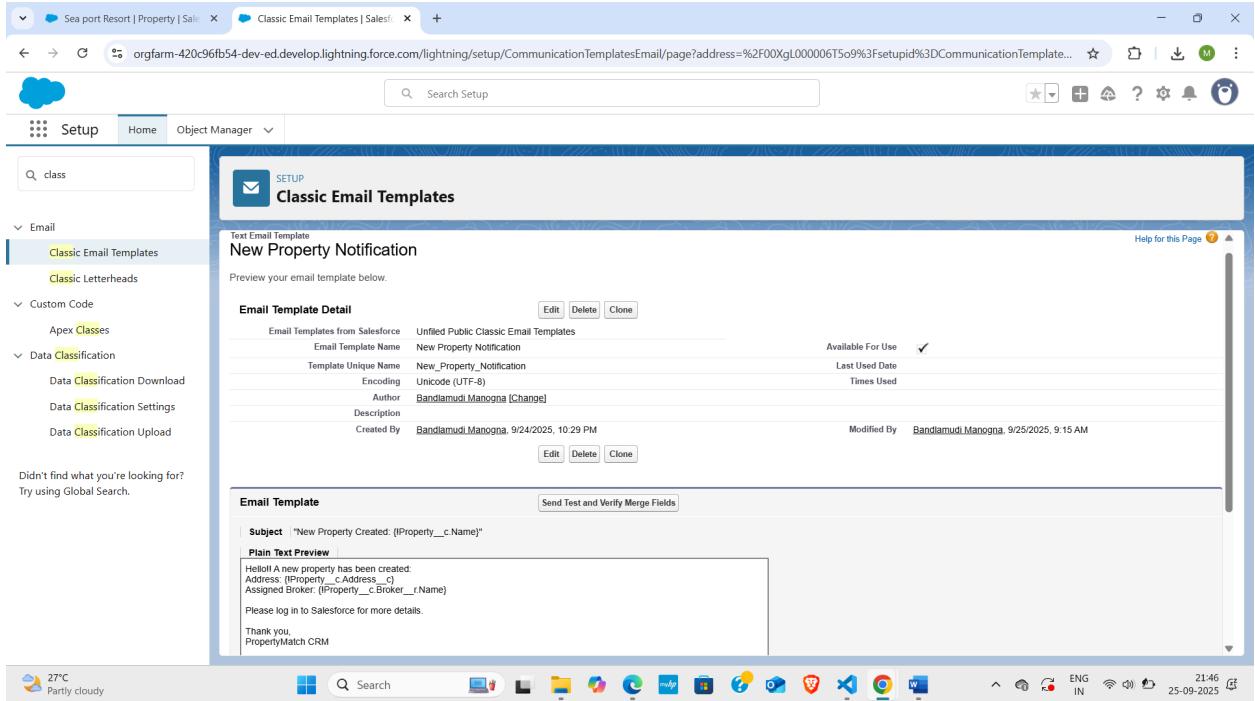
- **Brokers:**
 - Broker 1 → Name: charitha, Email: charitha@propertymatch.com, Phone: 9876543210
 - Broker 2 → Name: sampath, Email: sampath@propertymatch.com, Phone: 9876501234
- **Link Property to Broker:**
 - Property 1 (Hyderabad) → Assigned to Broker 1 (charitha)
 - Property 2 (Bangalore) → Assigned to Broker 2 (sampath)

Brokers		Recently Viewed ▾	New	Import	Change Owner	Assign Label
	Broker Name					
3 items • Updated a few seconds ago		▼				
1	<input type="checkbox"/> Charitha Bandlamudi					
2	<input type="checkbox"/> Sampath					
3	<input type="checkbox"/> Rajani					

Phase 4: Process Automation (Admin)

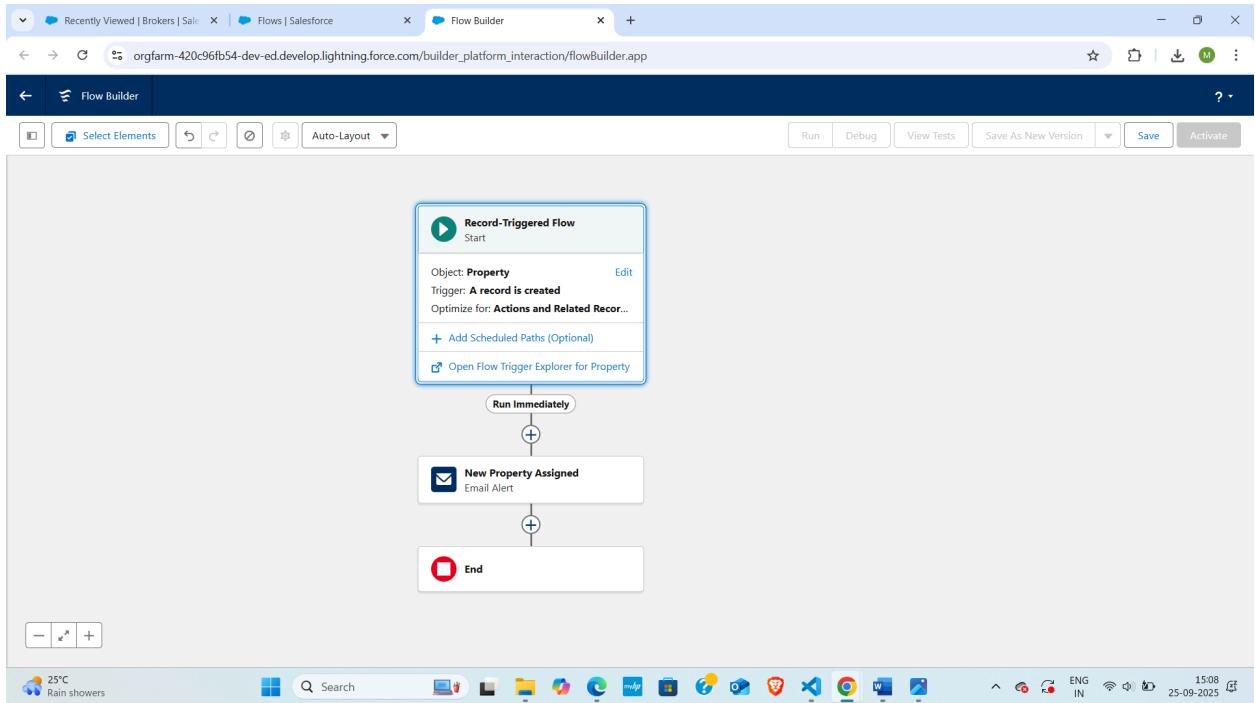
1. Email Alerts to Brokers

- **Purpose:** Notify brokers automatically when a property is assigned.
- **Navigation:**
 - Setup → Email Alerts → New Email Alert.
 - Choose object **Property__c**.
 - Selected **Broker Email__c** field.
 - Attached template: “New Property Assigned – Details Included”.
- **How it works:** When a Property is created with Broker__c field, an email is triggered to that broker.



2. Flow – Trigger Email on Property Assignment

- **Purpose:** Automate broker notification without manual effort.
- **Navigation:**
 - Setup → Flows → New Flow → Record-Triggered Flow.
 - Object: **Property__c**.
 - Trigger: When record is created or updated.
 - Condition: Broker__c field is not null.
 - Action: **Send Email Alert** using created template.
- **How it works:** Whenever a property is assigned to a broker, the broker instantly receives an email with property details.



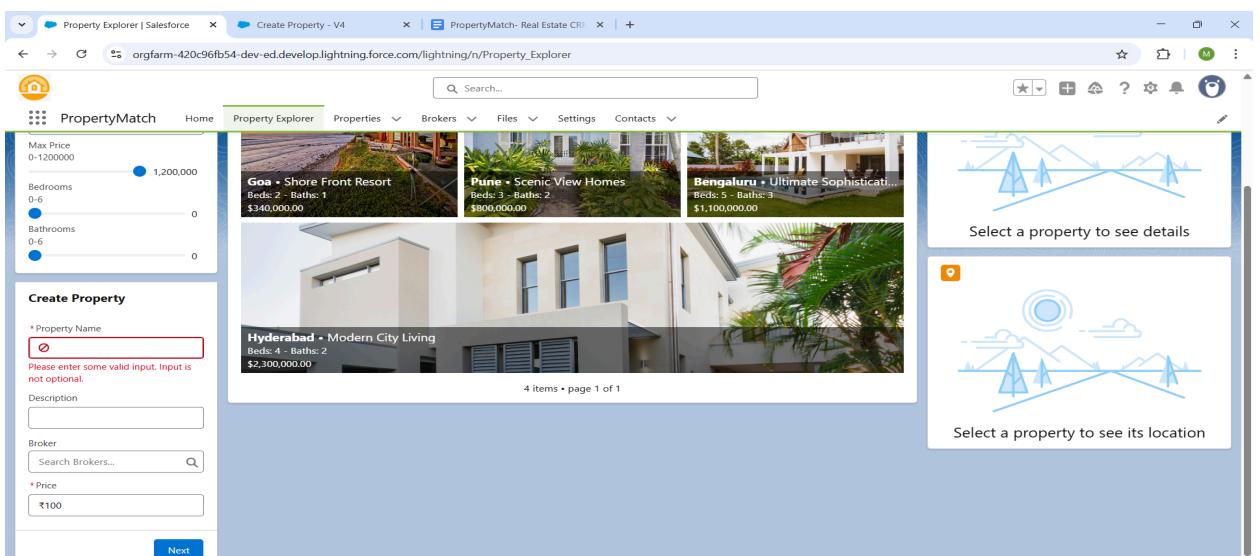
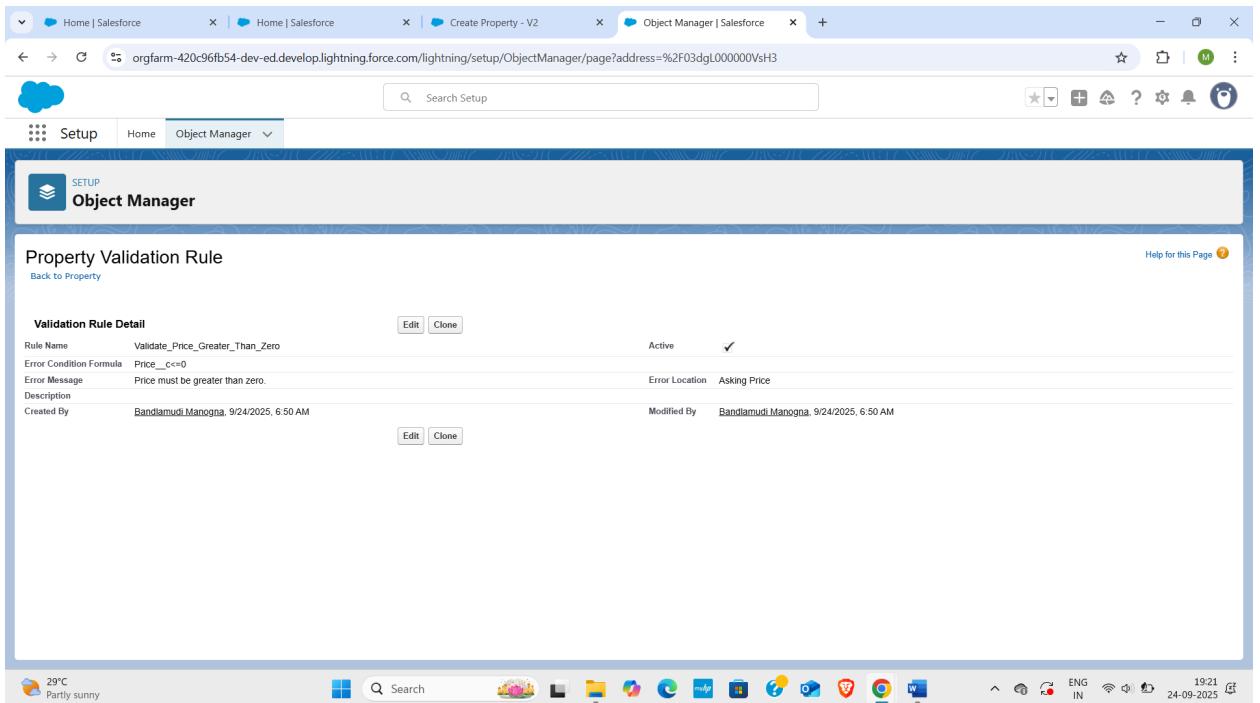
3. Validation Rules

- **Purpose:** Ensure important property details are always filled.
- **Navigation:**

Setup → Object Manager → Property__c → Validation Rules → New.

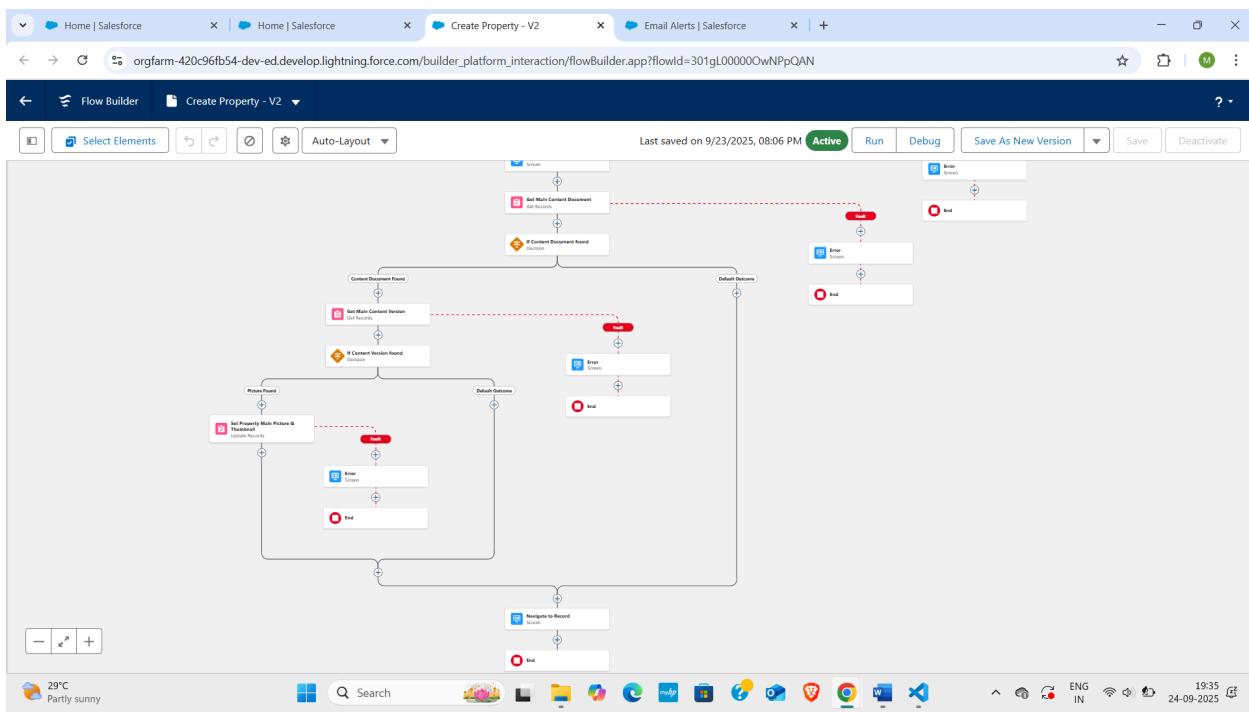
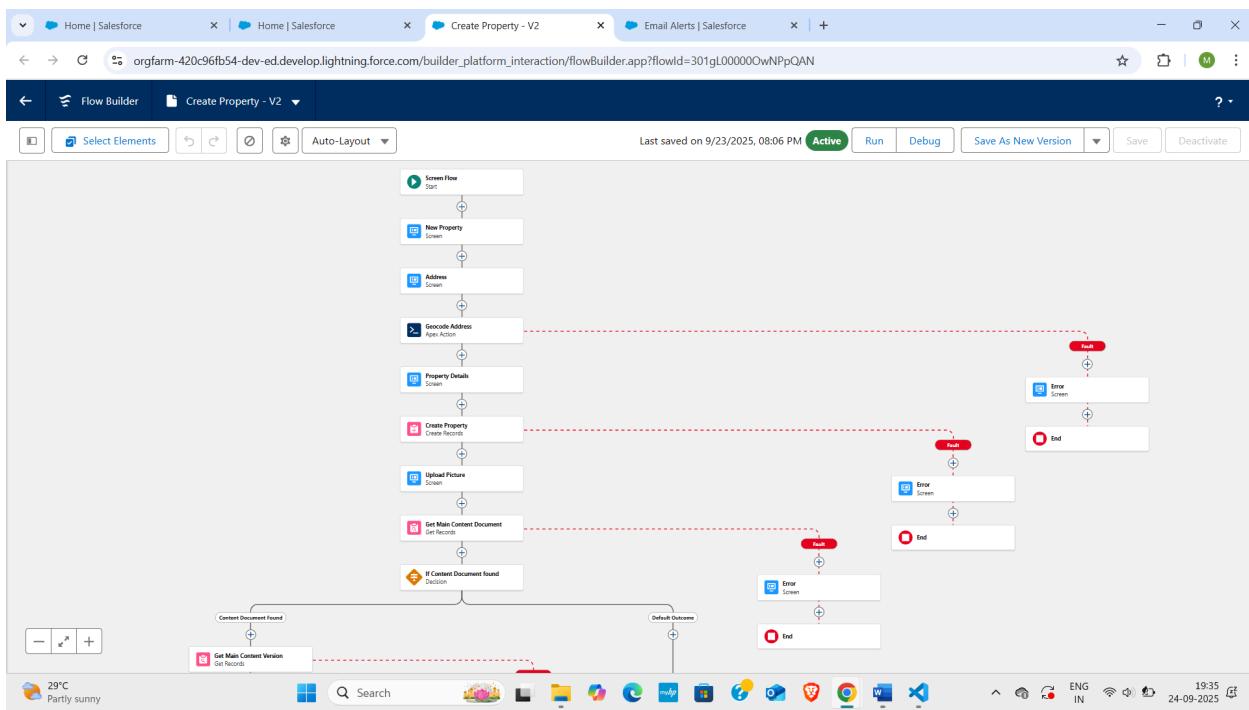
- **Rules Created:**
 - `ISBLANK(Price__c)` → Error: “Price must be greater than zero.”
 - `ISBLANK(Name__c)` → Error: “Please enter some valid input. Input is not optional.”

How it works: Prevents saving incomplete property records, ensuring data quality. Give an error page saying we hit a sang, review error on this page and the error details.



3. Screen Flow:

In the **Property Explorer** tab, **Screen Flows** were implemented to simplify property creation by guiding administrators through step-by-step input of fields such as city, price, bedrooms, bathrooms, and broker assignment, picture upload. This ensures that all required details are captured in an organized manner, reducing errors and improving the property onboarding process.

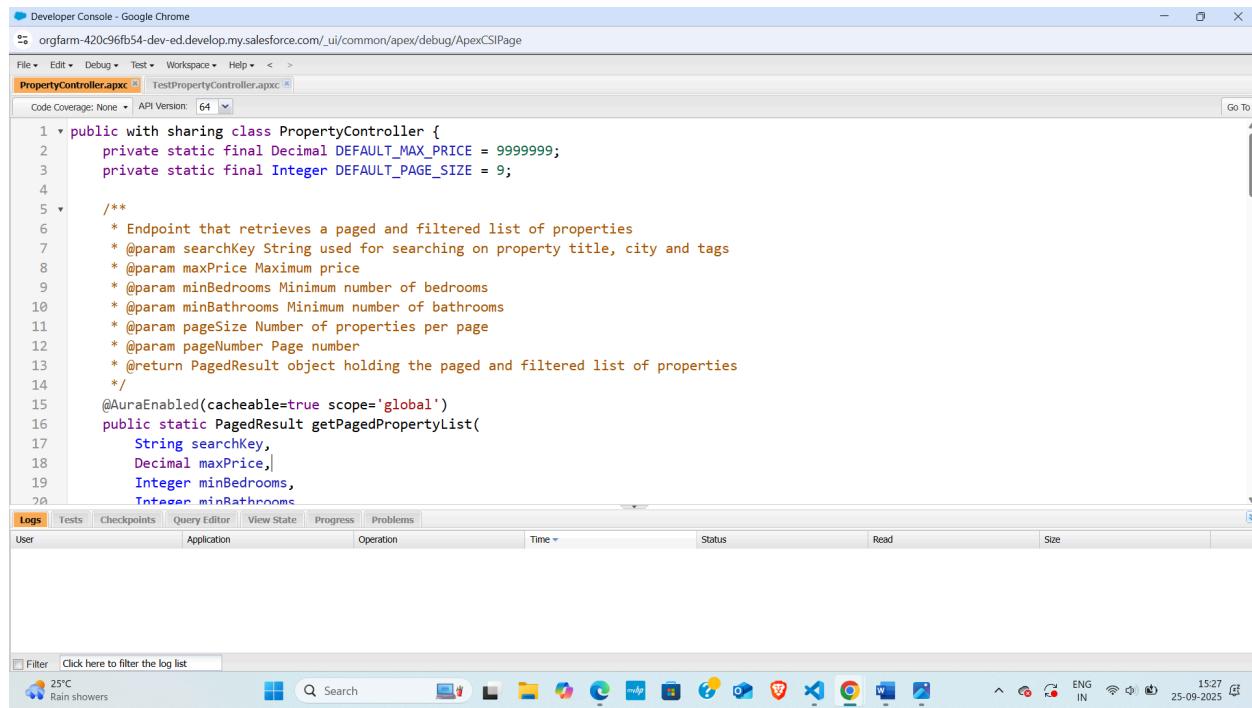


Phase 5: Apex Programming (Developer)

1. Apex Classes Created

a) PropertyController.cls

- **Purpose:** Fetch property records from `Property__c` based on filters.
- **Implementation Steps:**
 - Navigation: **Setup → Apex Classes → New → PropertyController.cls.**
 - Added methods for fetching property data.



The screenshot shows the Salesforce Developer Console in Google Chrome. The URL is `orgfarm-420c96fb54-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage`. The tab is titled `PropertyController.apex`. The code editor contains the following Apex class:

```
1 public with sharing class PropertyController {
2     private static final Decimal DEFAULT_MAX_PRICE = 9999999;
3     private static final Integer DEFAULT_PAGE_SIZE = 9;
4
5     /**
6      * Endpoint that retrieves a paged and filtered list of properties
7      * @param searchKey String used for searching on property title, city and tags
8      * @param maxPrice Maximum price
9      * @param minBedrooms Minimum number of bedrooms
10     * @param minBathrooms Minimum number of bathrooms
11     * @param pageSize Number of properties per page
12     * @param pageNumber Page number
13     * @return PagedResult object holding the paged and filtered list of properties
14     */
15     @AuraEnabled(cacheable=true scope='global')
16     public static PagedResult<Property> getPagedPropertyList(
17         String searchKey,
18         Decimal maxPrice,
19         Integer minBedrooms,
20         Integer minBathrooms
21     ) {
```

Below the code editor is a table with columns: Logs, Tests, Checkpoints, Query Editor, View State, Progress, and Problems. The Logs tab is selected. The status bar at the bottom shows: 25°C Rain showers, Search, File Explorer, Task List, Project Explorer, Database, Log, and Visualforce. The date and time are 25-09-2025 15:27.

The screenshot shows the Salesforce Setup Apex Classes page. The left sidebar is collapsed, and the main area displays the **PropertyController** class under the **Apex Classes** section. The class has a single method, `getAagedProperties`, which performs a query on the `Property` object based on search criteria and pagination.

```

1 public with sharing class PropertyController {
2     private static final Decimal DEFAULT_MAX_PRICE = 9999999;
3     private static final Integer DEFAULT_PAGE_SIZE = 5;
4
5     /**
6      * Endpoint that retrieves a pagged and filtered list of properties
7      * @param maxPrice Maximum price
8      * @param minBedrooms Minimum number of bedrooms
9      * @param maxBedrooms Maximum number of bedrooms
10     * @param pageNumber Page number
11     * @param pageSize Page size
12     */
13    @AuraEnabled(cacheable=true scope=global)
14    public static List<Property> getAagedProperties(
15        @Param(maxPrice) Decimal maxPrice,
16        @Param(minBedrooms) Integer minBedrooms,
17        @Param(maxBedrooms) Integer maxBedrooms,
18        @Param(pageNumber) Integer pageNumber,
19        @Param(pageSize) Integer pageSize
20    ) {
21        String searchPattern = '%' + safeKey + '%';
22        Integer offset = (pageNumber - 1) * safePageSize;
23        Integer resultCount = 0;
24        PageableResult<Property> result;
25        result = new PageableResult<Property>(
26            resultPageNumber = safePageNumber,
27            resultPageSize = safePageSize,
28            resultTotalCount = resultCount,
29            resultOffsetCount = 1
30        );
31
32        String query =
33            'SELECT Id, Name, Address__c, City__c, State__c, Description__c, Beds__c, Bath__c, Thumbnail__c, Location__Latitude__s, Location__Longitude__s
34            FROM Property
35            WHERE
36            (Name LIKE :searchPattern
37            OR Address__c LIKE :searchPattern
38            OR City__c LIKE :searchPattern
39            OR State__c LIKE :searchPattern
40            AND Beds__c >= :safeMinBedrooms
41            AND Beds__c <= :safeMaxBedrooms
42            AND Bath__c >= :safeMinBedrooms
43            AND Bath__c <= :safeMaxBedrooms
44            AND Price__c <= :maxPrice
45            AND Price__c >= :DEFAULT_MAX_PRICE
46            ORDER BY Price__c
47            LIMIT :safePageSize
48            OFFSET :offset
49        ';
50
51        result.records = query;
52        result.id = Id;
53        result.name = Name;
54        result.address = Address__c;
55        result.city = City__c;
56        result.state = State__c;
57        result.description = Description__c;
58        result.beds = Beds__c;
59        result.bath = Bath__c;
60        result.thumbnail = Thumbnail__c;
61        result.locationLatitude = Location__Latitude__s;
62        result.locationLongitude = Location__Longitude__s;
63
64        result.totalRecords = resultCount;
65        result.offsetCount = offset;
66    }
67
68    /**
69     * Endpoint that retrieves pictures associated with a property
70     * @param propertyId Property id
71     * @return List of ContentVersion holding the pictures
72     */
73    @AuraEnabled(cacheable=true scope=global)
74    public static List<ContentVersion> getPictures(@Id propertyId) {
75        List<ContentVersion> result = new List<ContentVersion>();
76
77        String query =
78            'SELECT Id, ContentDocumentId, Title, VersionNumber, LastModifiedDate, LastReferencedDate
79            FROM ContentVersion
80            WHERE ContentDocumentId = :propertyId
81            ORDER BY LastModifiedDate DESC
82            LIMIT :safePageSize
83            OFFSET :offset
84        ';
85
86        ContentVersion[] records = Database.query(query);
87
88        for (ContentVersion record : records) {
89            if (record.ContentDocument.FileType IN ('PNG', 'JPG', 'GIF')) {
90                result.add(record);
91            }
92        }
93    }
}

```

The screenshot shows the Salesforce Setup Apex Classes page. The left sidebar is collapsed, and the main area displays the **PropertyController** class code. The `getProperties` method is shown, which performs a query on the `Property` object based on search criteria and pagination.

```

36     result.pageNumber = safePageNumber;
37     result.records = [
38         SELECT COUNT()
39     ] FROM Property__c
40     WHERE
41         (Name LIKE :searchPattern
42         OR Address__c LIKE :searchPattern
43         OR City__c LIKE :searchPattern
44         OR State__c LIKE :searchPattern
45         AND Beds__c >= :safeMinBedrooms
46         AND Beds__c <= :safeMaxBedrooms
47         AND Price__c <= :maxPrice
48         AND Price__c >= :DEFAULT_MAX_PRICE
49         ORDER BY Price__c
50         LIMIT :safePageSize
51         OFFSET :offset
52     ];
53
54     return result;
55 }
56
57 /**
58  * Endpoint that retrieves pictures associated with a property
59  * @param propertyId Property id
60  * @return List of ContentVersion holding the pictures
61  */
62    @AuraEnabled(cacheable=true scope=global)
63    public static List<ContentVersion> getPictures(@Id propertyId) {
64        List<ContentVersion> result = new List<ContentVersion>();
65
66        String query =
67            'SELECT Id, ContentDocumentId, Title, VersionNumber, LastModifiedDate, LastReferencedDate
68            FROM ContentVersion
69            WHERE ContentDocumentId = :propertyId
70            ORDER BY LastModifiedDate DESC
71            LIMIT :safePageSize
72            OFFSET :offset
73        ';
74
75        ContentVersion[] records = Database.query(query);
76
77        for (ContentVersion record : records) {
78            if (record.ContentDocument.FileType IN ('PNG', 'JPG', 'GIF')) {
79                result.add(record);
80            }
81        }
82    }
83
84
85
86
87
88
89
90
91
92
93

```

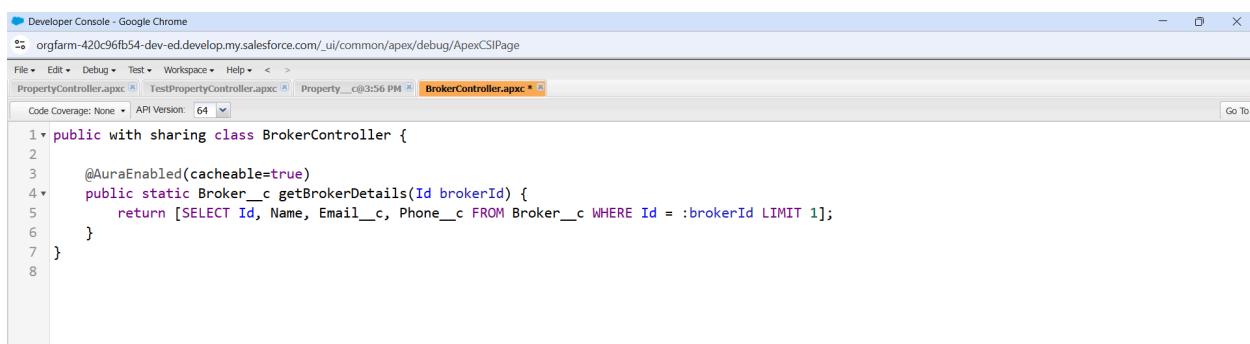
● How it works:

- Accepts filter inputs from LWC (searchKey, maxPrice, bedrooms, bathrooms).

- Dynamically builds SOQL query.
- Returns a list of matching properties.
- Used `@AuraEnabled(cacheable=true)` so LWC can call efficiently.

b) BrokerController.cls

- **Purpose:** Fetch broker details for display and email alerts.
- **Implementation Steps:**
 - Navigation: **Setup → Apex Classes → New → BrokerController.cls.**
- **Code Logic:**



The screenshot shows the Salesforce Developer Console in Google Chrome. The URL is orgfarm-420c96fb54-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage. The tab bar shows PropertyController.apx, TestPropertyController.apx, and BrokerController.apx (which is currently selected). The code editor displays the following Apex code:

```

1 public with sharing class BrokerController {
2
3     @AuraEnabled(cacheable=true)
4     public static Broker__c getBrokerDetails(Id brokerId) {
5         return [SELECT Id, Name, Email__c, Phone__c FROM Broker__c WHERE Id = :brokerId LIMIT 1];
6     }
7 }
8

```

- **How it works:**
 - LWC uses this to display broker details when a property record is opened.

2. Test Classes

Why required:

- Salesforce requires **minimum 75% code coverage** for deployment.
- Validates that Apex methods work with sample data.

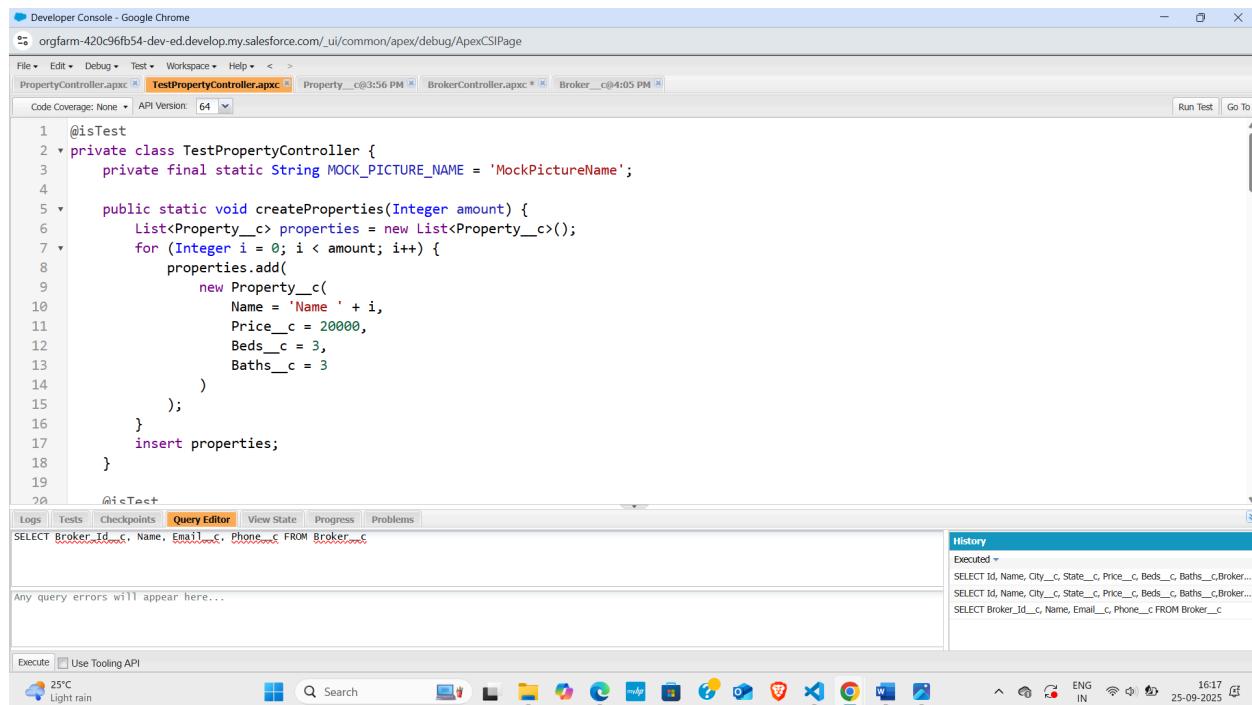
- **PropertyControllerTest.cls**

Steps to create Test Class:

1. Click **Setup** → **Apex Classes** → **New**.
2. Copy and paste the test class code below.
3. Click **Save**.

Test Logic Overview:

- Creates sample Property and Broker records.
- Tests property filtering and broker fetching.



```
Developer Console - Google Chrome
orgfarm-420c96fb54-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File ▾ Edit ▾ Debug ▾ Test ▾ Workspace ▾ Help ▾ < ▾ >
PropertyController.apxc [ TestPropertyController.apxc ] Property__c@3:56 PM [ BrokerController.apxc ] Broker__c@4:05 PM [ ]
Code Coverage: None ▾ API Version: 64 ▾ Run Test Go To
1  @isTest
2  private class TestPropertyController {
3      private final static String MOCK_PICTURE_NAME = 'MockPictureName';
4
5      public static void createProperties(Integer amount) {
6          List<Property__c> properties = new List<Property__c>();
7          for (Integer i = 0; i < amount; i++) {
8              properties.add(
9                  new Property__c(
10                      Name = 'Name ' + i,
11                      Price__c = 20000,
12                      Beds__c = 3,
13                      Baths__c = 3
14                  )
15              );
16          }
17          insert properties;
18      }
19  }
@isTest
Logs Tests Checkpoints Query Editor View State Progress Problems
SELECT Broker_Id__c, Name, Email__c, Phone__c FROM Broker__c
Any query errors will appear here...
History
Executed ▾
SELECT Id, Name, City__c, State__c, Price__c, Beds__c, Baths__c,Broker...
SELECT Id, Name, City__c, State__c, Price__c, Beds__c, Baths__c,Broker...
SELECT Broker_Id__c, Name, Email__c, Phone__c FROM Broker__c
Execute Use Tooling API
Cloud 25°C Light rain Search
16:17 25-09-2025 ENG IN
```

The screenshot shows the Salesforce Developer Console interface. The top navigation bar includes File, Edit, Debug, Test, Workspace, Help, and a dropdown for orgfarm-420c96fb54-dev-ed.develop.my.salesforce.com. The main area displays an Apex class named TestPropertyController.apxc. A modal window titled "Running tests asynchronously..." shows a single status message: "Success". Below the code editor, there are tabs for Logs, Tests, Checkpoints, Query Editor (which contains a simple SELECT query), Progress, and Problems. The bottom right corner shows system status icons for weather (25°C, Light rain), network, battery, and date/time (25-09-2025, 15:58).

```

1  @isTest
2  private class TestPropertyController {
3      private final static String MOCK_PICTURE_NAME = 'MockPictureName';
4
5      public static void createProperties(Integer amount) {
6          List<Property__c> properties = new List<Property__c>();
7          for (Integer i = 0; i < amount; i++) {
8              properties.add(
9                  new Property__c(
10                     Name = 'Name ' + i,
11                     Price__c = 20000,
12                     Beds__c = 3,
13                     Baths__c = 3
14                 )
15             );
16         }
17         insert properties;
18     }
19
20     @isTest

```

Logs Tests Checkpoints **Query Editor** View State Progress Problems

```

SELECT Id, Name, City__c, State__c, Price__c, Beds__c, Baths__c, Broker__c
FROM Property__c

```

Any query errors will appear here...

History
Executed ~

25°C Light rain Search 25-09-2025 15:58

● BrokerControllerTest.cls

The screenshot shows the Salesforce Developer Console interface. The top navigation bar includes File, Edit, Debug, Test, Workspace, Help, and a dropdown for orgfarm-420c96fb54-dev-ed.develop.my.salesforce.com. The main area displays an Apex class named BrokerControllerTest.apxc. It contains a single test method named testGetBrokerDetails() which creates a test broker record and asserts its name. Below the code editor, there are tabs for Logs, Tests, Checkpoints, Query Editor, Progress, and Problems. The bottom right corner shows system status icons for weather (25°C, Light rain), network, battery, and date/time (25-09-2025, 15:58).

```

1  @isTest
2  public class BrokerControllerTest {
3
4      @isTest
5      static void testGetBrokerDetails() {
6          // Create test broker
7          Broker__c broker = new Broker__c(Name='Test Broker', Email__c='test@broker.com', Phone__c='1234567890');
8          insert broker;
9
10         // Call Apex method
11         Broker__c result = BrokerController.getBrokerDetails(broker.Id);
12
13         System.assertEquals('Test Broker', result.Name);
14     }
15 }

```

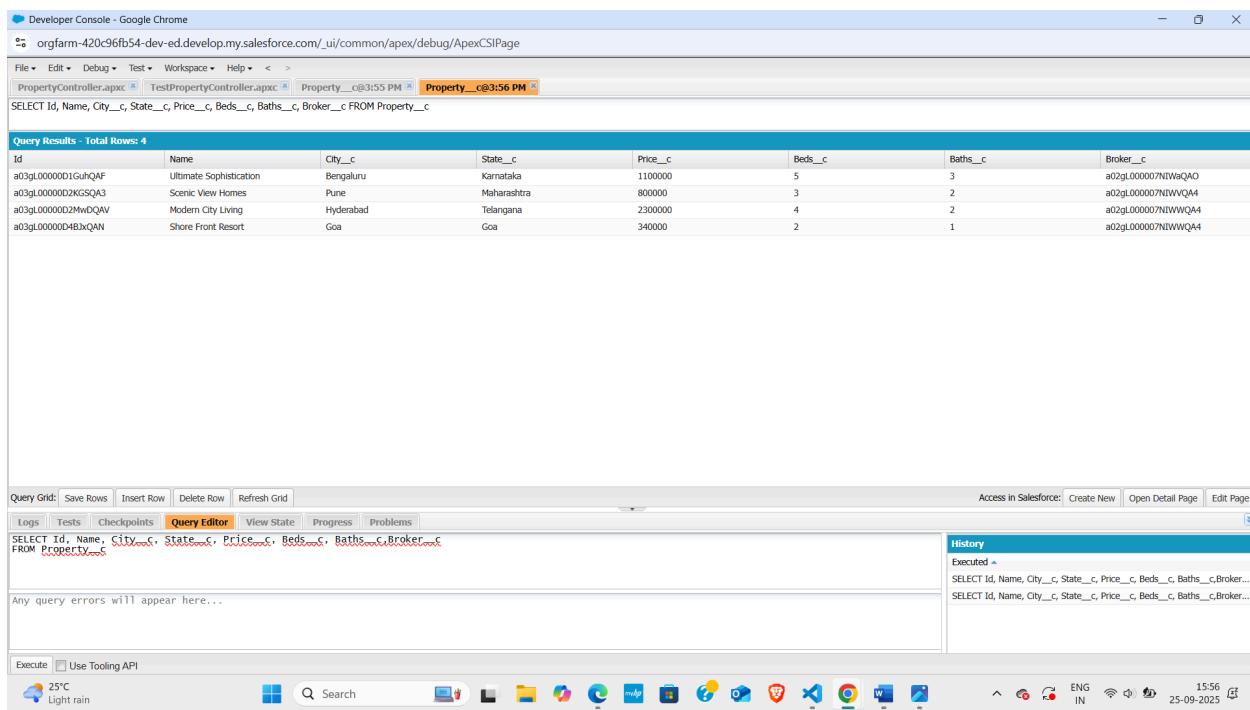
● How it works:

- Creates test records in memory (not visible in org).
- Runs Apex methods with test data.

- Asserts results to confirm correct behavior.
- Ensures >75% coverage for deployment.

3. SOQL Queries Used

Fetching properties with filters:



The screenshot shows the Salesforce Developer Console in Google Chrome. The URL is orgfarm-420c96fb54-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage. The tabs at the top are PropertyController.apxc, TestPropertyController.apxc, Property_c@3:55 PM, and Property_c@3:56 PM (highlighted in orange). The current query is:

```
SELECT Id, Name, City__c, State__c, Price__c, Beds__c, Baths__c, Broker__c FROM Property__c
```

The results table shows four rows of property data:

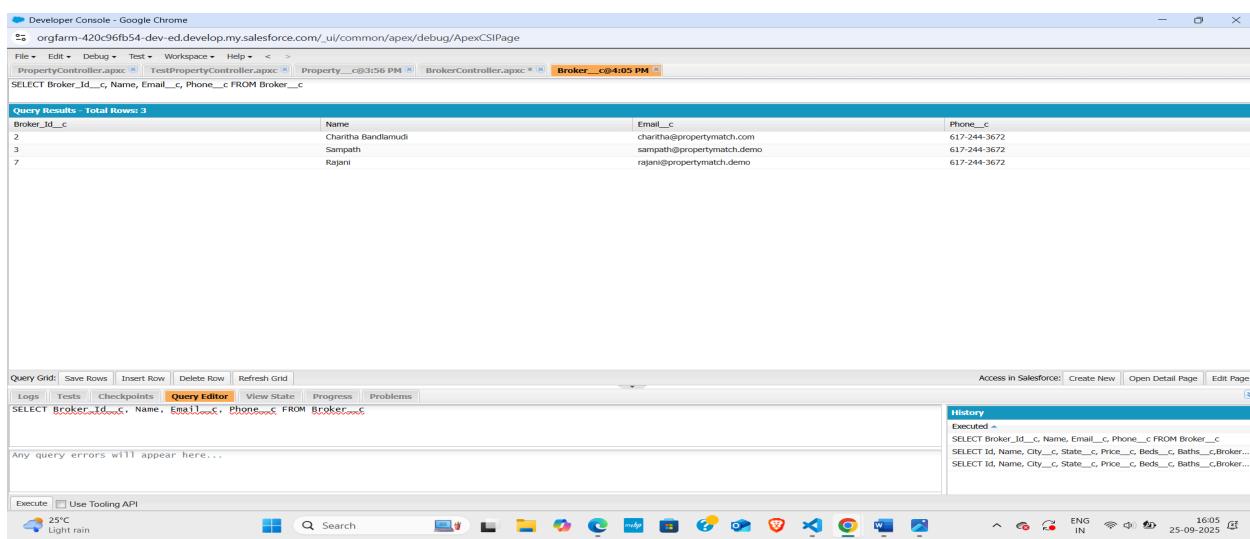
ID	Name	City__c	State__c	Price__c	Beds__c	Baths__c	Broker__c
a03gl000000D1GuhQAF	Ultimate Sophistication	Bengaluru	Karnataka	1100000	5	3	a02gl000007N1WaQAO
a03gl000000D2KGSSQA3	Scenic View Homes	Pune	Maharashtra	800000	3	2	a02gl000007N1WQ4A
a03gl000000D2MwDQAV	Modern City Living	Hyderabad	Telangana	2300000	4	2	a02gl000007N1WWQ4A
a03gl000000D4BjxQAN	Shore Front Resort	Goa	Goa	340000	2	1	a02gl000007N1WWQ4A

Below the results, the Query Editor tab is selected, showing the same query:

```
SELECT Id, Name, City__c, State__c, Price__c, Beds__c, Baths__c, Broker__c  
FROM Property__c
```

The History section shows the query was executed at 3:56 PM on 25-09-2025.

Fetching broker details:



The screenshot shows the Salesforce Developer Console in Google Chrome. The URL is orgfarm-420c96fb54-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage. The tabs at the top are PropertyController.apxc, TestPropertyController.apxc, Property_c@3:56 PM, and BrokerController.apxc, Broker_c@4:05 PM (highlighted in orange). The current query is:

```
SELECT Broker_Id__c, Name, Email__c, Phone__c FROM Broker__c
```

The results table shows three rows of broker data:

Broker_Id__c	Name	Email__c	Phone__c
2	Chairtha Bandlamudi	chartha@propertymatch.com	617-244-3672
3	Sampath	sampath@propertymatch.demo	617-244-3672
7	Rejani	rejani@propertymatch.demo	617-244-3672

Below the results, the Query Editor tab is selected, showing the same query:

```
SELECT Broker_Id__c, Name, Email__c, Phone__c FROM Broker__c
```

The History section shows the query was executed at 4:05 PM on 25-09-2025.

Exception Handling

- Why used:
 - To ensure Apex methods don't break when invalid filters or missing data are passed from LWC.
 - Helps log errors and show user-friendly messages.
- Implementation in PropertyController:

The screenshot shows the Salesforce Developer Console in Google Chrome. The URL is orgfarm-420c96fb54-dev-ed.develop.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage. The tab bar shows PropertyController.apxc (active), TestPropertyController.apxc, Property_c@3:56 PM, BrokerController.apxc, and Broker_c@4:05 PM. The code editor contains the following Apex code:

```
20     Integer minBathrooms,
21     Integer pageSize,
22     Integer pageNumber
23   )
24   @AuraEnabled(cacheable=true)
25   public static List<Property__c> getFilteredProperties(String searchKey, Decimal maxPrice, Integer beds, Integer baths) {
26     try {
27       // existing SOQL filter logic
28       return Database.query(soql);
29     } catch (Exception e) {
30       System.debug('Error in getFilteredProperties: ' + e.getMessage());
31       return new List<Property__c>(); // return empty list to LWC
32     }
33   }
34   // Normalize inputs
35   Decimal safeMaxPrice = maxPrice ?? DEFAULT_MAX_PRICE;
36   Integer safeMinBedrooms = minBedrooms ?? 0;
37   Integer safeMinBathrooms = minBathrooms ?? 0;
38 }
```

The Query Editor pane shows the query: SELECT Broker_Id__c, Name, Email__c, Phone__c FROM Broker__c. The History pane shows the following executed queries:

```
SELECT Id, Name, City__c, State__c, Price__c, Beds__c, Baths__c, Broker__c
SELECT Id, Name, City__c, State__c, Price__c, Beds__c, Baths__c, Broker__c
SELECT Broker_Id__c, Name, Email__c, Phone__c FROM Broker__c
```

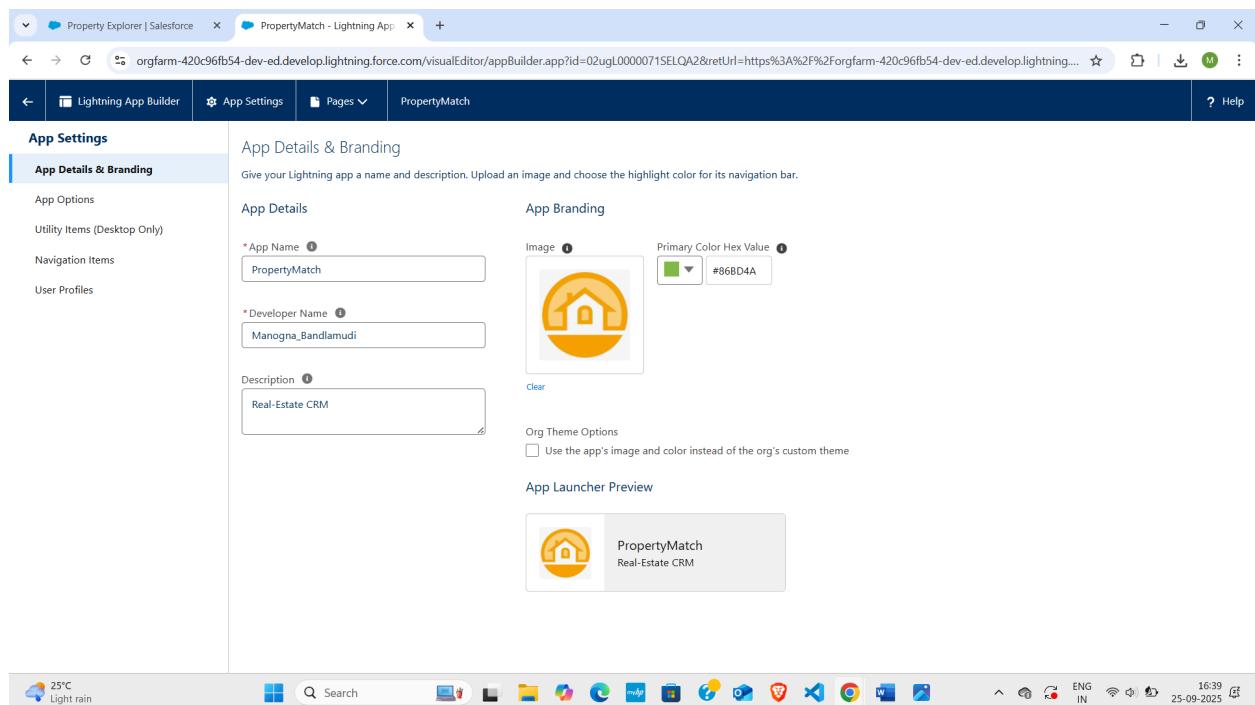
- How it works:
 - If invalid input/query error occurs, the system does not break.
 - Returns a safe empty list instead of crashing the LWC.

Phase 6: User Interface Development

1. Lightning App Setup

- Navigation: **Setup → App Manager → Lightning App Builder.**
- Created **PropertyMatch App** with tabs:
 - **Home**
 - **Properties**
 - **Brokers**

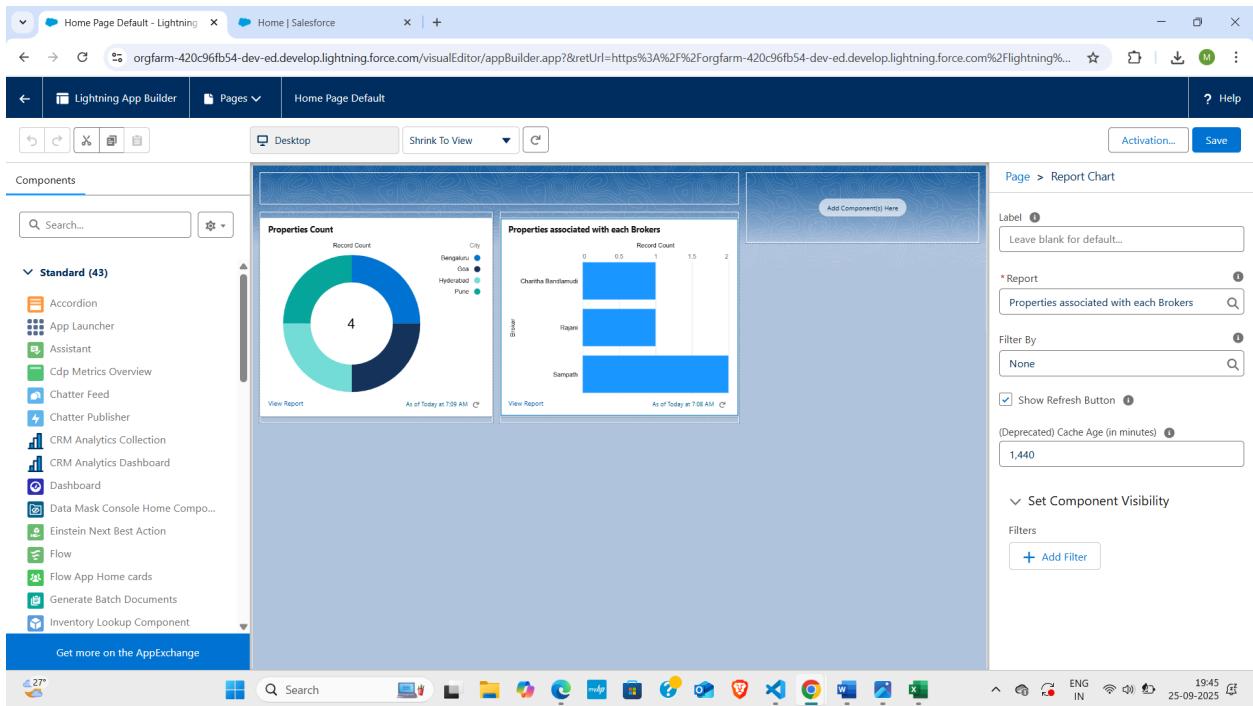
Why: Centralized navigation for all users (admins, brokers, buyers).



2. Home Page (Reports + Dashboard + Navigation)

- **Component Used:** Standard Dashboard + Report Chart.

- **How it works:** Displays property count by city in a donut chart (Bengaluru, Goa, Hyderabad, Pune).
- **Steps:**
 1. Go to **App Builder** → **Home Page**.
 2. Added **Report Chart Component** linked to “Properties Report”.
 3. Saved → Activated as org-wide default Home Page.
- **Screenshot:** Properties Report showing city-wise distribution.



3. Property Explorer Page (Custom LWC)

- **LWCs Used:** `propertyFilter`, `propertyTileList`, `propertyMap`.

- **How it works:**

1. `propertyFilter` → captures inputs (search key, max price, bedrooms, bathrooms).

2. Sends filters to **Apex (PropertyController)**.
3. `propertyTitleList` → displays property cards with price, city, beds, baths, and image.
4. Clicking card → shows details and location.
5. `propertyMap` → displays property markers on Google Maps.

4. LWC Components & How They Work

a) `propertyFilter` LWC

- **Purpose:** Capture user filter input (search key, max price, bedrooms, bathrooms).
- **How it works:**
 - HTML: Input fields (text, number).
 - JS: Stores user inputs and calls Apex method `getFilteredProperties()`.
 - On submit, it sends filters to `PropertyList` LWC.
 - Deploy your changes to your org by right-clicking the file and selecting `SFDX: Deploy This Source to Org`

```

force-app > main > default > lwc > propertyFilter > JS propertyFilter.js > ...
8  export default class PropertyFilter extends LightningElement {
28 }
29
30 handleMaxPriceChange(event) {
31   this.maxPrice = event.detail.value;
32   this.fireChangeEvent();
33 }
34
35 handleMinBedroomsChange(event) {
36   this.minBedrooms = event.detail.value;
37   this.fireChangeEvent();
38 }
39
40 handleMinBathroomsChange(event) {
41   this.minBathrooms = event.detail.value;
42   this.fireChangeEvent();
43 }
44
45 fireChangeEvent() {
46   // Debouncing this method: Do not actually fire the event as long as this function is
47   // being called within a delay of DELAY. This is to avoid a very large number of Apex
48   // method calls in components listening to this event.
49   window.clearTimeout(this.delayTimeout);
50   // eslint-disable-next-line @lwc/lwc/no-async-operation
51   this.delayTimeout = setTimeout(() => {
52     const filters = {
53       searchKey: this.searchKey,
54       maxPrice: this.maxPrice,
55       minBedrooms: this.minBedrooms,
56       minBathrooms: this.minBathrooms
57     };
58     publish(this.messageContext, FILTERSCHANGEMC, filters);
59   }, DELAY);
60 }
61 }

```

b) propertyList LWC

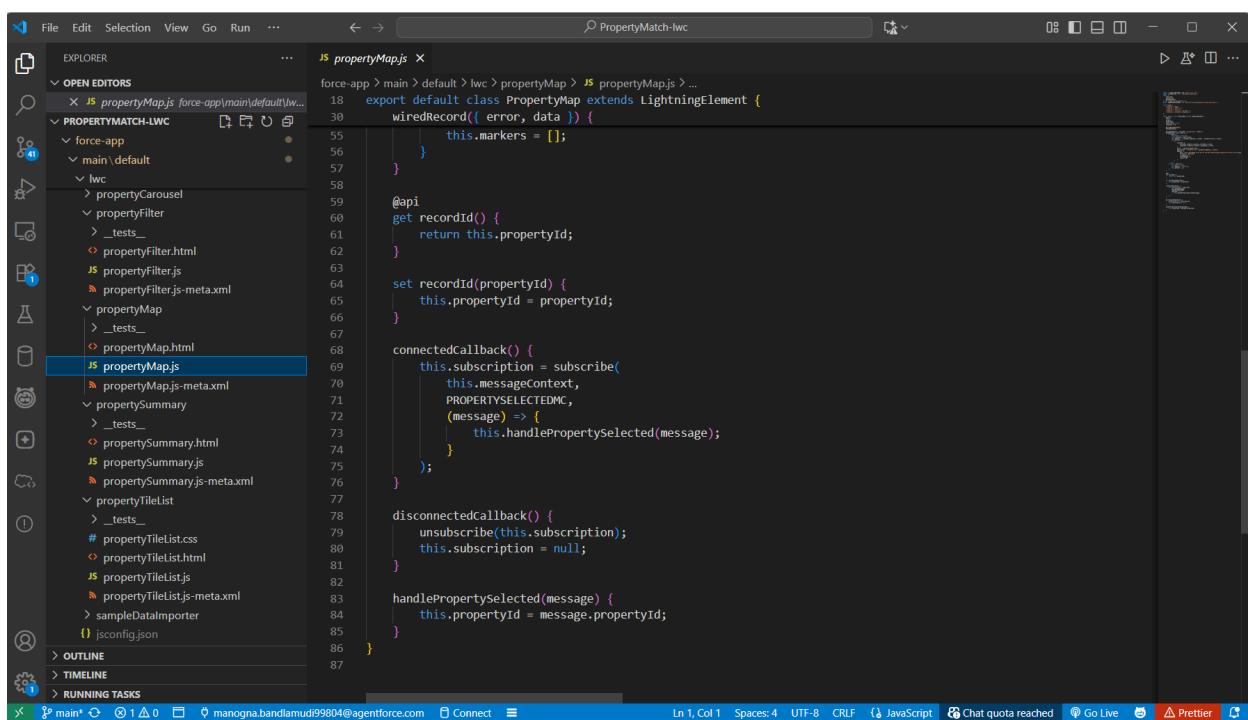
- **Purpose:** Display properties that match filters.
- **How it works:**
 - Calls `PropertyController.getFilteredProperties()` with filter values.
 - Displays property tiles (name, image, city, price, bedrooms, bathrooms).

c) propertyTile LWC

- **Purpose:** Individual card for each property.
- **How it works:**
 - Receives property record from propertyList.
 - Displays image, price, city, bedrooms, bathrooms.
 - On click → navigate to Property Record Page.

d) propertyMap LWC

- **Purpose:** Show property location on map.
- **How it works:**
 - Uses a lightning-map base component.
 - Takes Latitude__c and Longitude__c from Property__c.
 - Requires **Google OAuth 2.0 key** for map integration.



The screenshot shows the Salesforce Developer Console interface. The left sidebar displays the project structure under 'OPEN EDITORS' for 'PROPERTYSUMMARY-LWC'. The main editor window shows the code for 'propertyMap.js'. The code defines a class 'PropertyMap' extending 'LightningElement'. It includes methods for handling wired records, getting the record ID, setting the record ID, connecting to a message bus, and handling disconnected states. The code uses ES6 syntax and imports from 'lightning-map' and 'lightning-element'.

```
force-app > main > default > lwc > propertyMap > propertyMap.js
export default class PropertyMap extends LightningElement {
    wiredRecord({ error, data }) {
        if (!error) {
            this.markers = [];
        }
    }
    @api
    get recordId() {
        return this.propertyId;
    }
    set recordId(propertyId) {
        this.propertyId = propertyId;
    }
    connectedCallback() {
        this.subscription = subscribe(
            this.messageContext,
            PROPERTYSELECTEDMC,
            (message) => {
                this.handlePropertySelected(message);
            }
        );
    }
    disconnectedCallback() {
        unsubscribe(this.subscription);
        this.subscription = null;
    }
    handlePropertySelected(message) {
        this.propertyId = message.propertyId;
    }
}
```

e) brokerCard LWC

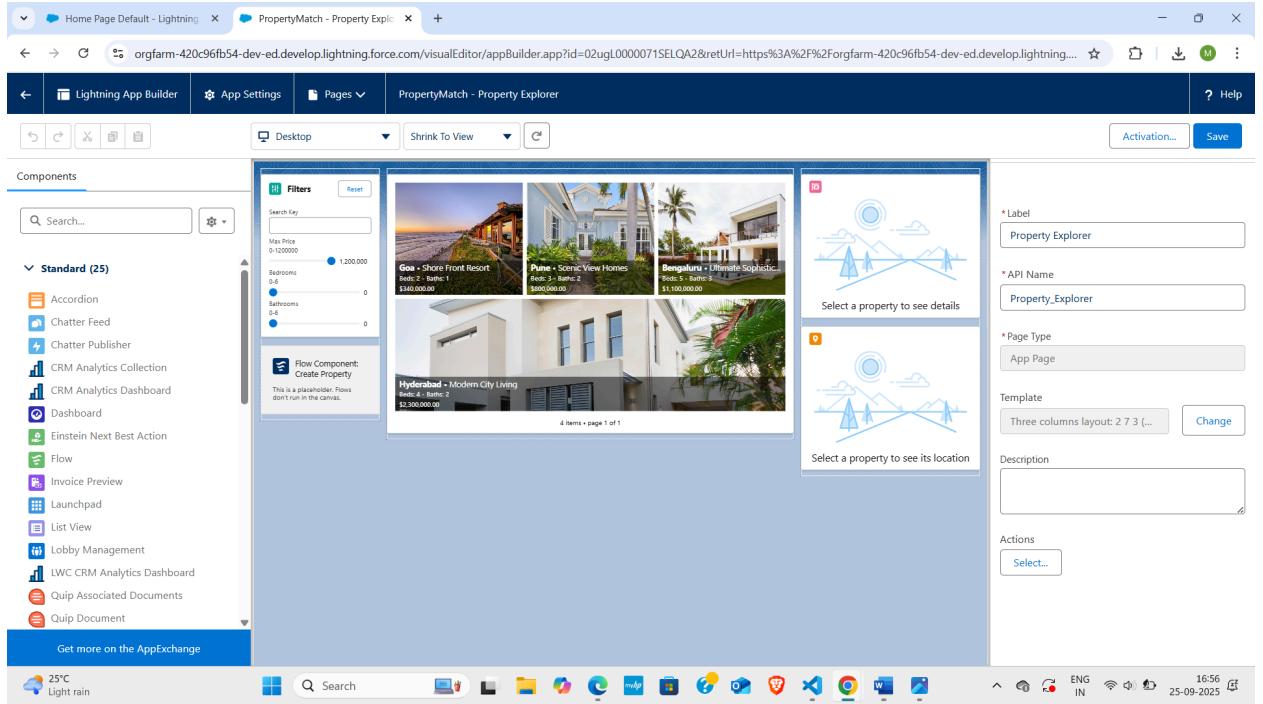
- **Purpose:** Show broker details when property record is opened.
- **How it works:**
 - Calls `BrokerController.getBrokerDetails()` using property's Broker__c.
 - Displays broker name, email, phone.

The screenshot shows the Salesforce Setup interface with the 'Lightning Components' page selected. The left sidebar shows various setup categories like Sales Schedules, Objects and Fields, and User Interface. Under 'Custom Code', 'Lightning Components' is expanded, and its sub-item 'Lightning Components' is also selected. The main content area displays a table titled 'Lightning Components' with columns for Action, Name +, Label, Type, Namespace Prefix, and API Version. The table lists numerous LWC components such as barcodeScanner, brokerCard, daysOnMarket, errorPanel, idList, listContactsFromDevice, navigateToRecord, pageTemplate_2_7_3, paginator, propertyCarousel, propertyFilter, propertyListMap, propertyLocation, propertyMap, propertySummary, propertyTitle, and propertyTileList. All components listed are of type 'LWC' and have an API version of 64.0.

- **Steps:**

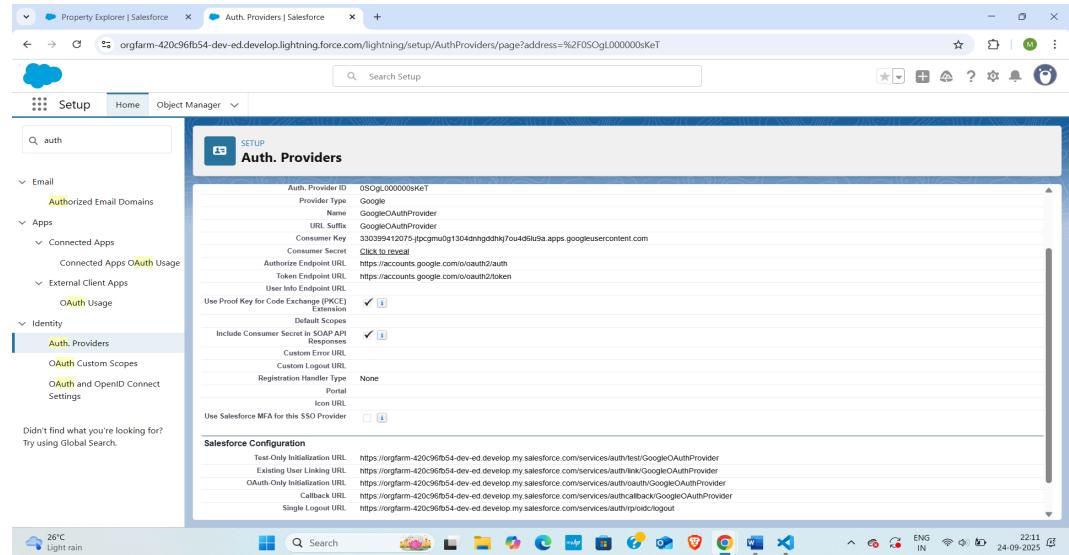
1. Created custom **Property Explorer Lightning Page**.
2. Dropped LWCs into layout (filters on left, list center, map/details on right).
3. Added **Create Property Form** section for admins.

- **Screenshot:** Property Explorer showing filters, property list, and create property tab.



3. How Filters & Location Were Implemented

- **Filters (Search, Max Price, Bedrooms, Bathrooms):**
 - Implemented in **propertyFilter LWC**.
 - Passed to **PropertyController Apex** → SOQL query filters properties.
 - Results displayed in **propertyList LWC** with tiles.
 - **Location (Map):**
 - Implemented in **propertyMap LWC** using **lightning-map**.
 - Uses **Latitude__c & Longitude__c** fields from **Property__c**.
 - Integrated with **Google OAuth 2.0** for map rendering.



6. Property Record Page

- **LWCs Used:** `propertyTile`, `brokerCard`
- **How it works:**
 1. Displays full property details (name, address, state, zip, status, description, image).
 2. Shows assigned broker info (`brokerCard`) with email and phone.
- **Steps:**
 1. Navigation: **Setup** → **Object Manager** → **Property__c** → **Lightning Record Pages** → **New Record Page**.
 2. Added LWC components into the property details section.
- **Screenshot:** Property record page with broker details and map.

The screenshot shows the Salesforce Setup interface under the Object Manager section. A sidebar on the left lists various configuration options like Details, Fields & Relationships, Page Layouts, and Lightning Record Pages. The main content area displays the properties of a Lightning Page named 'Property_Record_Page'. It includes sections for Information (Name: Property_Record_Page, Description), Assignments By App (App: PropertyMatch, Form Factor: Desktop and phone), and Assignments By App, Record Type, and Profile (No Assignments to display). The bottom right of the page has a note: 'Always show me' with a link to 'more records per related list'. The status bar at the bottom indicates it's 29°C Partly sunny.

The screenshot shows the Salesforce Visual Editor for creating a new page. The left sidebar lists components like Standard, Accordion, Action Launcher, etc. The main area shows a preview of a page titled 'Ultimate Sophistication' for a property in Bangalore, Karnataka, asking for ₹11,00,000. The page includes fields for Address, State, Zip, Tags, Status, and Owner. To the right, the configuration panel shows settings for the page: Label ('Property Record Page'), API Name ('Property_Record_Page'), Page Type ('Record Page'), Object ('Property'), and Template ('Header and Right Sidebar'). A checkbox for 'Enable page-level dynamic actions for the Salesforce mobile app' is checked. The status bar at the bottom indicates it's 25°C Light rain.

7. Broker Record Page

- **Components Used:** brokerCard, Related List (Properties).

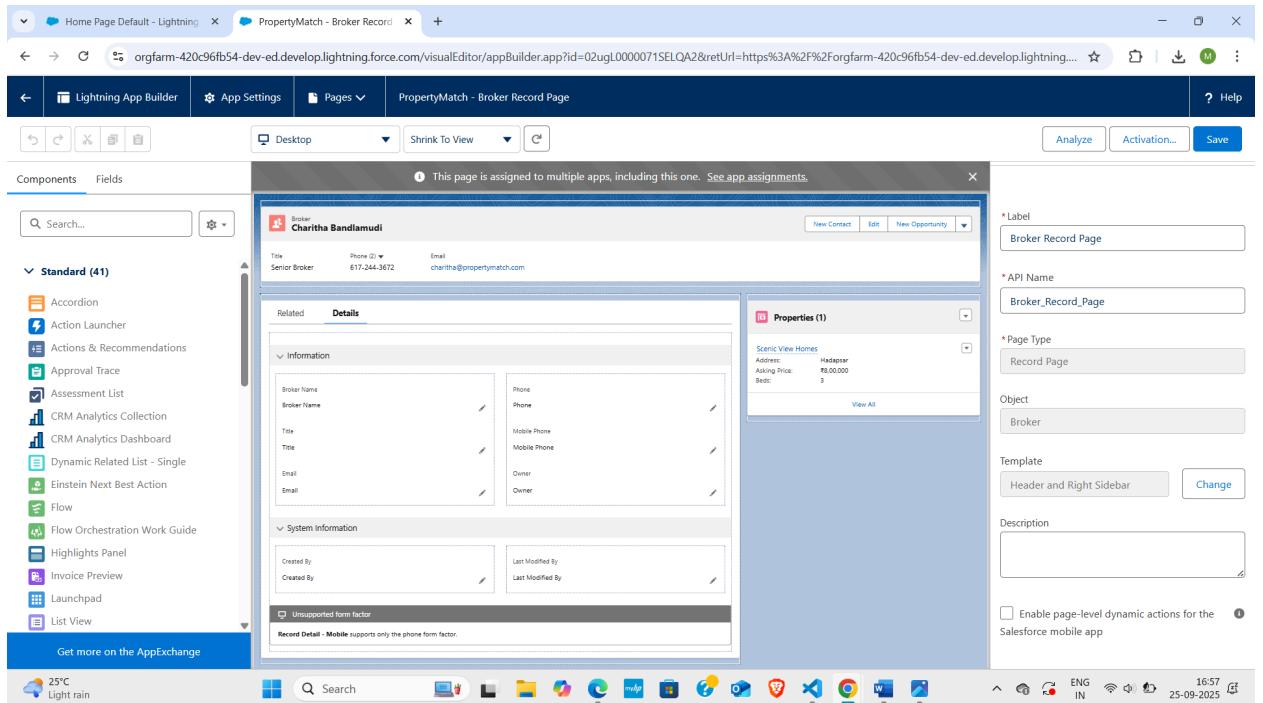
- **How it works:**

1. Displays broker details (name, title, phone, email).
2. Shows related properties assigned to the broker.

- **Steps:**

1. Navigation: **Setup → Object Manager → Broker__c → Lightning Record Pages.**
2. Added **brokerCard LWC and Properties Related List.**

- **Screenshot:** Broker record page with related property.



8. Tabs and Navigation

- **Tabs Created:**

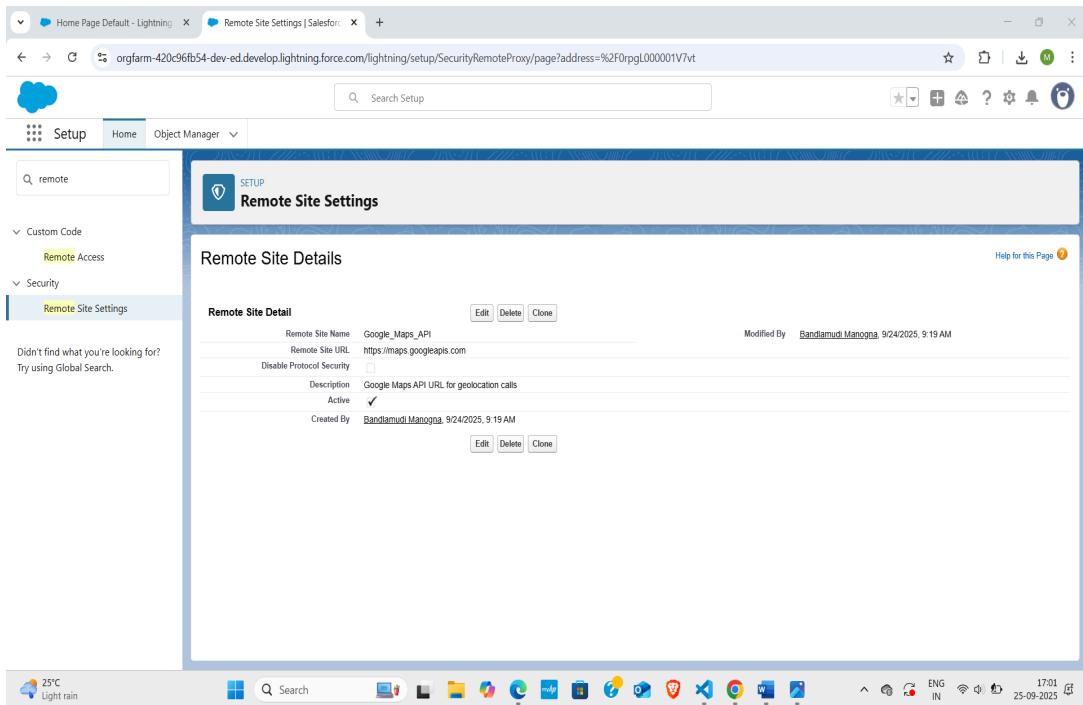
- **Home** → Default Home Page with reports.

- **Property Explorer** → Custom navigation item with filters + search.
 - **Properties** → Standard tab for Property__c records.
 - **Brokers** → Standard tab for Broker__c records.
- **How it all connects:**
 - User lands on **Home Page** (reports).
 - Goes to **Property Explorer** → filters/search → clicks property.
 - Opens **Property Record Page** → sees broker info + map.
 - Navigates to **Broker Tab** → sees broker details + assigned properties.

Phase 7: Integration & External Access

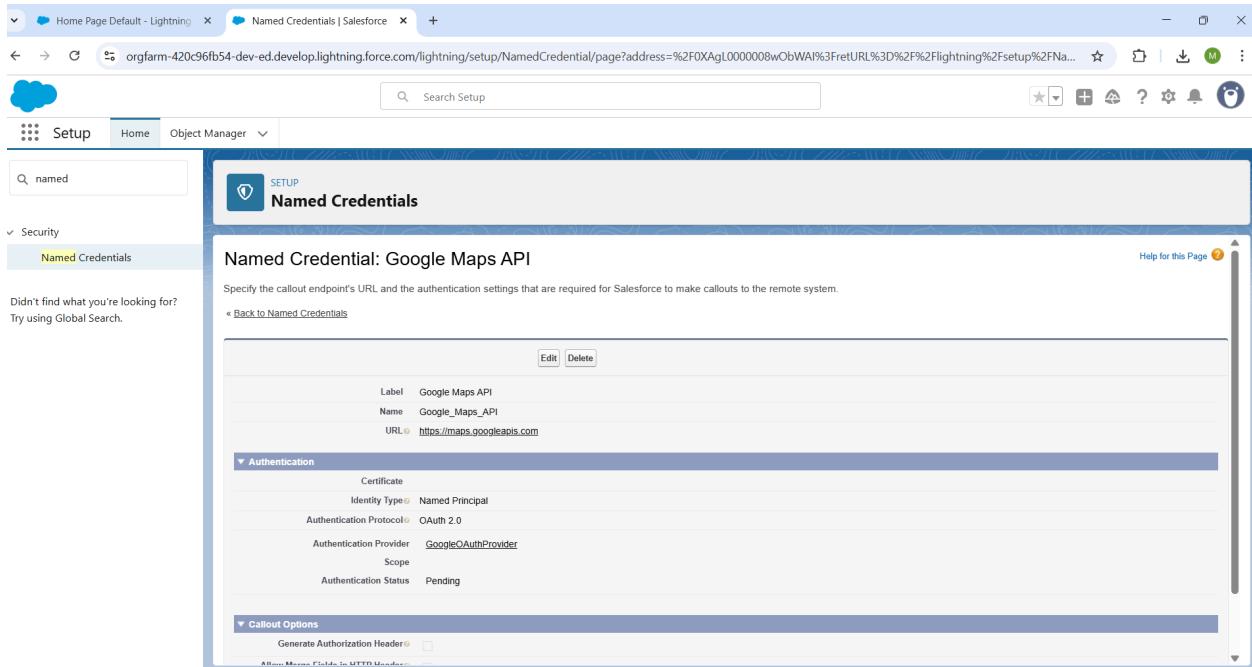
1. Remote Site Settings

- **Use Case:**
 - Needed to allow Salesforce org to make external callouts to Google Maps API.
- **Implementation Steps:**
 - Navigation: **Setup** → **Remote Site Settings** → **New Remote Site**.
 - Added Google Maps API base URL.
 - Enabled active status.
- **How it works:**
 - Allows Apex/LWC to send requests to Google Maps securely.



2. Named Credentials

- **Use Case:**
 - To securely store and authenticate credentials (API key or OAuth token) for Google API integration.
- **Implementation Steps:**
 - Navigation: **Setup → Named Credentials → New Named Credential.**
 - Added Google OAuth 2.0 endpoint with client ID, secret, and token.
 - Used in Apex for map integration.
- **How it works:**
 - Keeps credentials safe, avoids hardcoding in Apex.



3. OAuth & Authentication

- **Use Case:**

- Required for **Google Maps OAuth 2.0 integration** to fetch map data and render property locations.

- **Implementation Steps:**

- Registered project in **Google Cloud Console**.
- Enabled Maps JavaScript API + Places API.
- Generated **OAuth 2.0 Client ID & Secret**.
- Configured in Salesforce under **Auth Providers + Named Credential**.

- **How it works:**

- When a user views a property map, Salesforce makes authenticated requests to Google API.

4. Web Services (REST Callouts)

- **Use Case:**

- To fetch and display property locations on Google Map within Salesforce using coordinates.

- **Implementation Steps:**

- Configured callout to Google Maps API endpoint.
 - Used `lightning-map` LWC component which consumes the coordinates.

- **How it works:**

- Apex or LWC sends REST requests → Google API responds with map data → displayed in components.

Phase 8: Data Management & Deployment

1. Data Import Wizard

- **Use Case:**

- Used to upload initial property and broker records into the org.
 - Suitable for small data sets like test property listings.

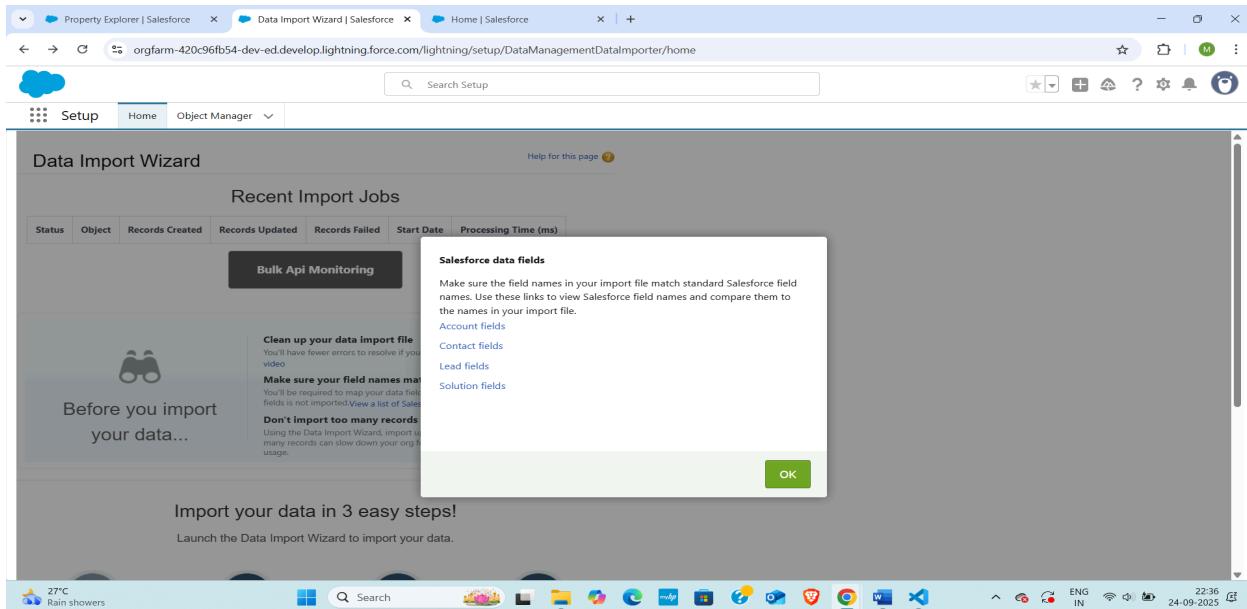
- **Implementation Steps:**

- Navigation: App Launcher → Data Import Wizard.
 - Selected **Property__c** object.

- Uploaded CSV with property details (City, State, Price, Bedrooms, Bathrooms, Latitude, Longitude, Broker).
- Mapped fields and imported records.

- **How it works:**

- Created sample data used for testing filters and map display.



2. Duplicate Rules

- **Use Case:**

- Prevented duplicate broker records with the same email address.

- **Implementation Steps:**

- Navigation: **Setup** → **Duplicate Rules** → **New Rule**.
- Selected **Broker__c** object.
- Matching rule: **Email__c** must be unique.

- **How it works:**

- When the admin tried to create a duplicate broker with the same email, the system blocked the save.

The screenshot shows the Matching Rules page in the Salesforce Setup. A matching rule named "Duplicate Email" is displayed. The rule details are as follows:

Object	Broker
Rule Name	Duplicate Email
Unique Name	Duplicate_Email
Description	When admin tried to create duplicate broker with same email, system blocked the save.
Matching Criteria	Broker: Email EXACT MatchBlank = FALSE
Status	Active
Created By	Bandlamudi Manogna, 9/25/2025, 5:19 AM
Modified By	Bandlamudi Manogna, 9/25/2025, 5:22 AM

The screenshot shows the Duplicate Rules page in the Salesforce Setup. A broker duplicate rule named "Distinct Email" is displayed. The rule details are as follows:

Rule Name	Distinct Email
Description	When admin tried to create duplicate broker with same email, system blocked the save.
Object	Broker
Record-Level Security	Enforce sharing rules
Action On Create	Allow
Action On Edit	Allow
Alert Text	Use one of these records?
Active	<input checked="" type="checkbox"/>
Matching Rule	<input checked="" type="checkbox"/> Duplicate_Email <input checked="" type="checkbox"/> Mapped
Conditions	
Created By	Bandlamudi Manogna, 9/25/2025, 5:22 AM
Matching Criteria	Broker: Email EXACT MatchBlank = FALSE
Operations On Create	<input checked="" type="checkbox"/> Alert <input checked="" type="checkbox"/> Report
Operations On Edit	<input checked="" type="checkbox"/> Alert <input checked="" type="checkbox"/> Report
Modified By	Bandlamudi Manogna, 9/25/2025, 5:23 AM

3. Data Export & Backup

- **Use Case:**

- For backup of property and broker records.

- **Implementation Steps:**

- Navigation: **Setup → Data Export → Schedule Export.**
 - Selected **Property__c** and **Broker__c** objects.
 - Scheduled weekly exports.

- **How it works:**

- Backup files stored as CSVs for safety in case of data loss.

A1	City	Street/Area	Postal Code	State	Latitude	Longitude
2	Mumbai	Nariman Point	400021	Maharashtra	18.92504	72.823925
3	Bengaluru	JP Nagar	560078	Karnataka	12.890753	77.579611
4	Chennai	Anna Nagar	600040	Tamil Nadu	13.087844	80.210493
5	Hyderabad	Banjara Hills	500034	Telangana	17.416471	78.438247
6	Pune	Hadapsar	411028	Maharashtra	18.496667	73.941667
7	Kolkata	Park Street	700016	West Bengal	22.5481	88.3549
8	Goa	Panaji (Altinho)	403001	Goa	15.49574	73.82624
9						
10						
11						
12						
13						
14						

4. VS Code & SFDX

- **Use Case:**

- Used for building LWCs, Apex classes, and deploying metadata.

- **Implementation Steps:**

- Open project in **VS Code**.
 - Developed LWCs (propertyFilter, propertyList, propertyTile, propertyMap, brokerCard).

Deployed to org with: `sfdx force:source:push`

Retrieved metadata from org with: `sfdx force:source:pull`

- **How it works:**

- Enabled smooth development → test → deployment cycle.

The screenshot shows the Deployment Status page in the Salesforce classic interface. The page title is "Deployment Status". It has two sections: "Failed" and "Succeeded". The "Failed" section is empty, displaying "No records to display.". The "Succeeded" section contains two entries:

Action	Name	Status	Date
View Details	0Agl00000AacPC	Deploy: Succeeded	9/23/2025, 7:36 AM
View Details	0Agl00000A9Mrb	Deploy: Succeeded	9/15/2025, 6:44 AM

At the bottom right of the table, there are links for "Previous (1 - 2 of 2)" and "Next". The left sidebar shows the "Deployment Status" tab is selected. The bottom navigation bar includes icons for search, home, and various applications like Microsoft Word, Excel, and Powerpoint.

Phase 9: Reporting, Dashboards & Security Review

1. Reports

- **Use Case:** To analyze property data (price ranges, city-wise distribution, broker assignments).
- **Types Used:**
 - **Tabular Report:** Simple list of all properties with city, price, bedrooms, bathrooms.

- **Joined Report:** List of each broker name with the number of properties they are assigned to.
- **Implementation Steps:**
 - Navigation: **App Launcher** → **Reports** → **New Report**.
 - Choose a custom report type for **Properties with Brokers**.
- **How it works:** Displays insights into availability and broker workload.
- **Use Case:** To join **Property__c** and **Broker__c** objects in one report.
- **Implementation Steps:**
 - Navigation: **Setup** → **Report Types** → **New Custom Report Type**.
 - Primary Object: **Property__c**.
 - Related Object: **Broker__c**.
- **How it works:** Allows reports combining property details with broker assignments.

Screenshot of a Salesforce report titled "New Properties Report". The report displays 4 total records across four categories: Modern City Living, Scenic View Homes, Shore Front Resort, and Ultimate Sophistication. Each category shows one record with details like Address, Asking Price, and Broker.

Category	Address	Asking Price	Broker
Modern City Living (1)	Banjara Hills	₹23,00,000	Sampath
Scenic View Homes (1)	Hadapsar	₹8,00,000	Charitha Bandlamudi
Shore Front Resort (1)	Panaji	₹3,40,000	Sampath
Ultimate Sophistication (1)	JP Nagar	₹11,00,000	Rajani

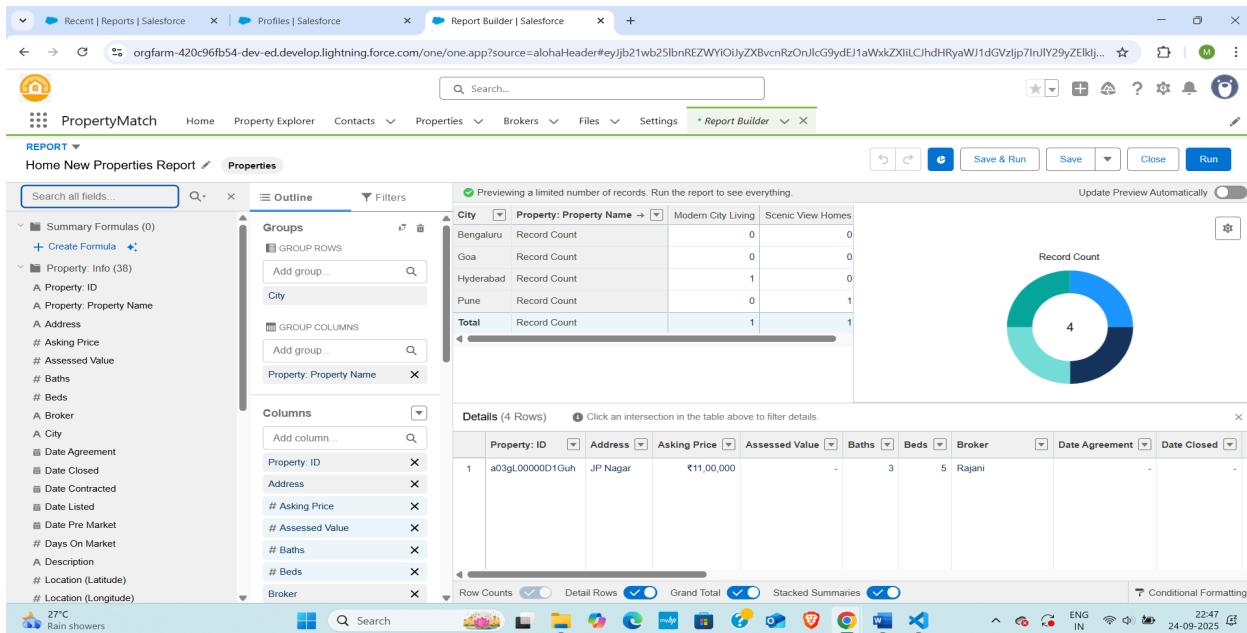
Screenshot of the Report Builder interface showing a joined report for properties per broker. The report includes a bar chart comparing the number of properties for three brokers: Charitha Bandlamudi, Rajani, and Sampath. Below the chart is a detailed table of properties for each broker.

Record Count

Broker	Record Count
Charitha Bandlamudi	1
Rajani	1
Sampath	2

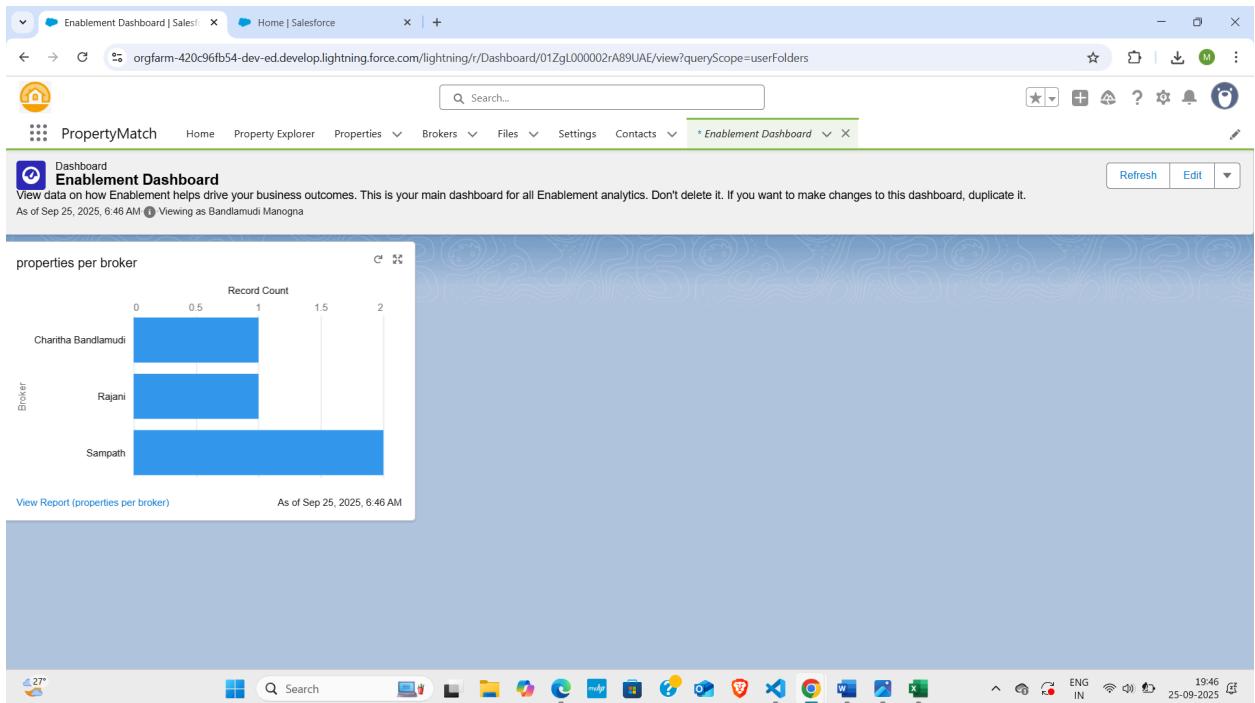
Properties block 1

Broker	Property: Property Name	City
Charitha Bandlamudi	Scenic View Homes	Pune
Subtotal	Count: 1	
Rajani	Ultimate Sophistication	Bengaluru
Subtotal	Count: 1	
Sampath	Modern City Living	Hyderabad
	Shore Front Resort	Goa



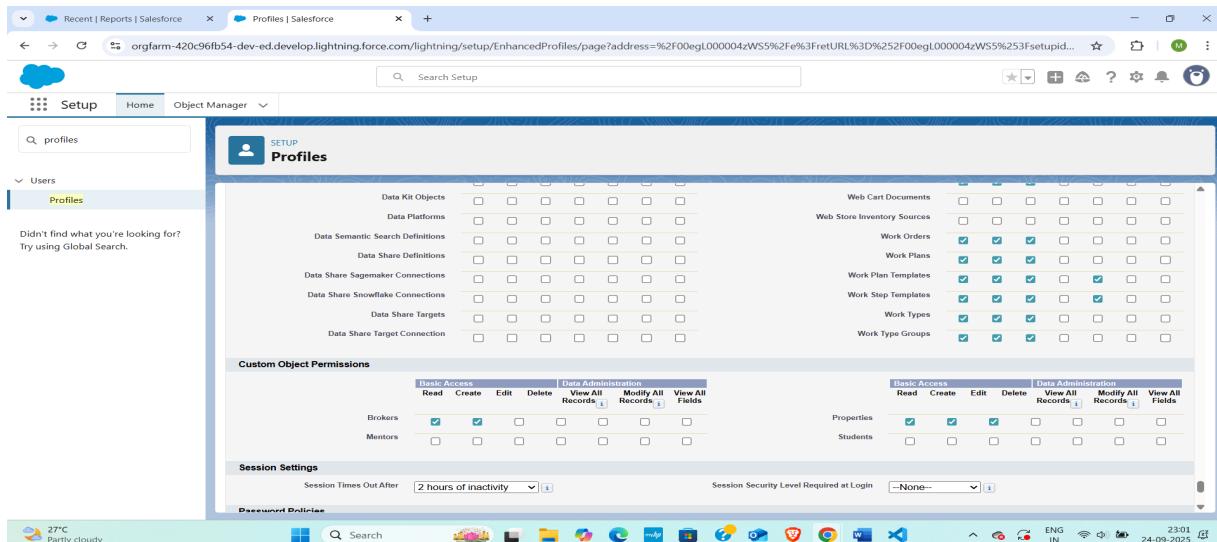
2. Dashboards

- **Use Case:** Visualize property and broker data for management.
- **Widgets Created:**
 - Pie Chart → Property distribution by city.
 - Bar Chart → No. of properties per broker.
- **Implementation Steps:**
 - Navigation: App Launcher → Dashboards → New Dashboard.
 - Added charts using saved reports.
- **How it works:** Quick, visual overview of CRM performance.



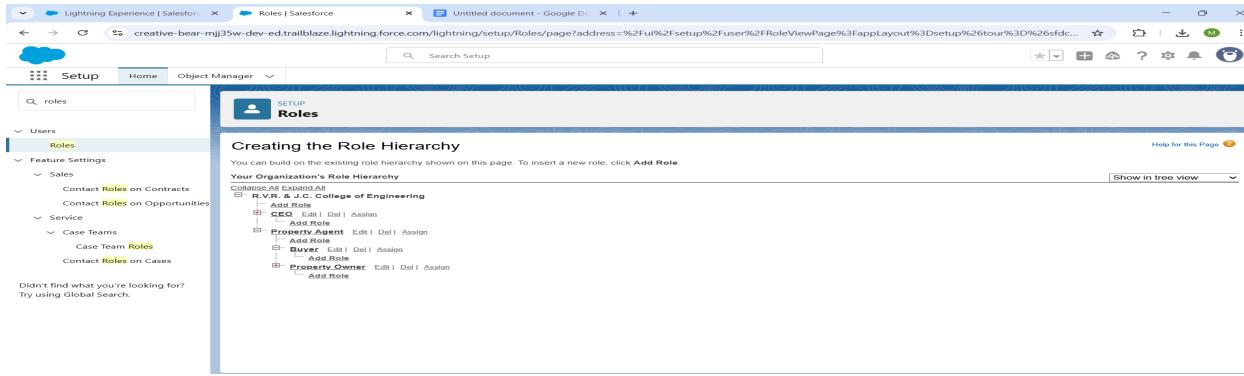
3. Profiles

- **Use Case:** Define access levels.
- **Profiles Created:**
 - **System Administrator** → Full access.
 - **Property Manager** → Manage properties and brokers.
 - **Broker User** → View only assigned properties.
- **How it works:** Restricts what each role can do in CRM.



4. Roles & Role Hierarchy

- **Use Case:** Data visibility based on hierarchy.
- **Roles Created:**
 - **Admin** (top).
 - **Property Manager** (middle).
 - **Broker** (bottom).
- **Implementation Steps:**
 - Setup → Roles → New Role.
 - Assigned users to roles.
- **How it works:** Managers can see data owned by brokers below them.



5. Users

- **Use Case:** Create different users for testing security & sharing.
- **Implementation Steps:**
 - Setup → Users → New User.
 - Created: Admin, 1 Manager, 2 Broker users.
- **How it works:** Tested record visibility & email alerts.

Action	Full Name	Alias	Username	Role	Active	Profile
<input type="checkbox"/>	Bandaru, Manoona	Mband	manoona.bandaru@creative-bear-mj35w.com		<input checked="" type="checkbox"/>	System Administrator
<input type="checkbox"/>	Brown, Courtney	cbrown	cbrown@apple2419.com	Manager	<input checked="" type="checkbox"/>	Standard Platform User
<input type="checkbox"/>	Buyer_Buyer	Buyer	manoona.bandaru@999@gmail.com	Buyer	<input checked="" type="checkbox"/>	Standard Platform User
<input type="checkbox"/>	Charter_Expert	Charter	chatty.00d200000mxukeaf.m4k0ooxi0xp@charter.salesforce.com	Charter	<input checked="" type="checkbox"/>	Charter Free User
<input type="checkbox"/>	User_Integration	integ	integration@00d200000mxukeaf.com	Integration	<input checked="" type="checkbox"/>	Analytics Cloud Integration User
<input type="checkbox"/>	User_Security	sec	insightssecurity@00d200000mxukeaf.com	Security	<input checked="" type="checkbox"/>	Analytics Cloud Security User
<input type="checkbox"/>	user_test	testuser	testusers196@gmail.com	Property Owner	<input checked="" type="checkbox"/>	Standard User
<input type="checkbox"/>	Wheeler, Allison	awheee	awheeler@apple05239.com	VP_North American Sales	<input checked="" type="checkbox"/>	Standard Platform User

6. Field-Level Security (FLS)

- **Use Case:** Restrict sensitive fields.

- **Example:** Hide `Price__c` from brokers, visible only to managers/admin.
- **Implementation Steps:**
 - Setup → Object Manager → `Property__c` → Fields & Relationships → Set Field-Level Security.
- **How it works:** Brokers can see property details but not price field, hide from standard users.

The screenshot shows the Salesforce Setup interface with the 'Field-Level Security' page open. The left sidebar is collapsed, showing various security-related settings like CORS, Delegated Administration, Event Monitoring, and Partner Encryption. The main content area displays the configuration for the 'Asking Price' field, which has a Data Type of 'Currency(0, 0)'. The 'Visible' column contains checked boxes for most user profiles, while the 'Read-Only' column contains empty boxes. The profiles listed include Analytics Cloud Integration User, Analytics Cloud Security User, AnyPoint Integration, Contact Manager, Cross Org Data Privacy User, Custom Marketing Profile, Custom Sales Profile, Custom Support Profile, Data Loader Agent User, Force.com - App Subscription User, Force.com - Free User, Gold Partner User, Identity User, Marketing User, Minimum Access - API Only Integrations, Minimum Access - Salesforce, Partner App Subscription User, Partner Community Login User, Partner Community User, PropertyMatch Profile, Read Only, Salesforce API Only System Integrations, Silver Partner User, Solution Manager, Standard Platform User, Standard User, and System Administrator.

7. Session Settings & Login IP Ranges

- **Use Case:** Extra security for admin access.
- **Implementation Steps:**
 - Setup → Session Settings → Reduced session timeout.
 - Setup → Profiles → Login IP Ranges → Allowed only office IP.
- **How it works:** Prevents unauthorized logins.

The screenshot shows the Salesforce Setup interface with the following details:

- Profiles Page:**
 - Login Hours:** A table showing login times for each day of the week.
 - Login IP Ranges:** A table listing IP addresses and their descriptions.
 - Enabled Apex Class Access:** A table listing Apex class names and their corresponding AppExchange Package Names.

The screenshot shows the Salesforce Setup interface with the following details:

- Sharing Settings Page:**
 - Sharing Rules:** A table listing sharing rules for various objects, such as Worklist, Warranty Term, Web Cart Document, etc., with columns for Object, Share Type, and Status.

8. Audit Trail

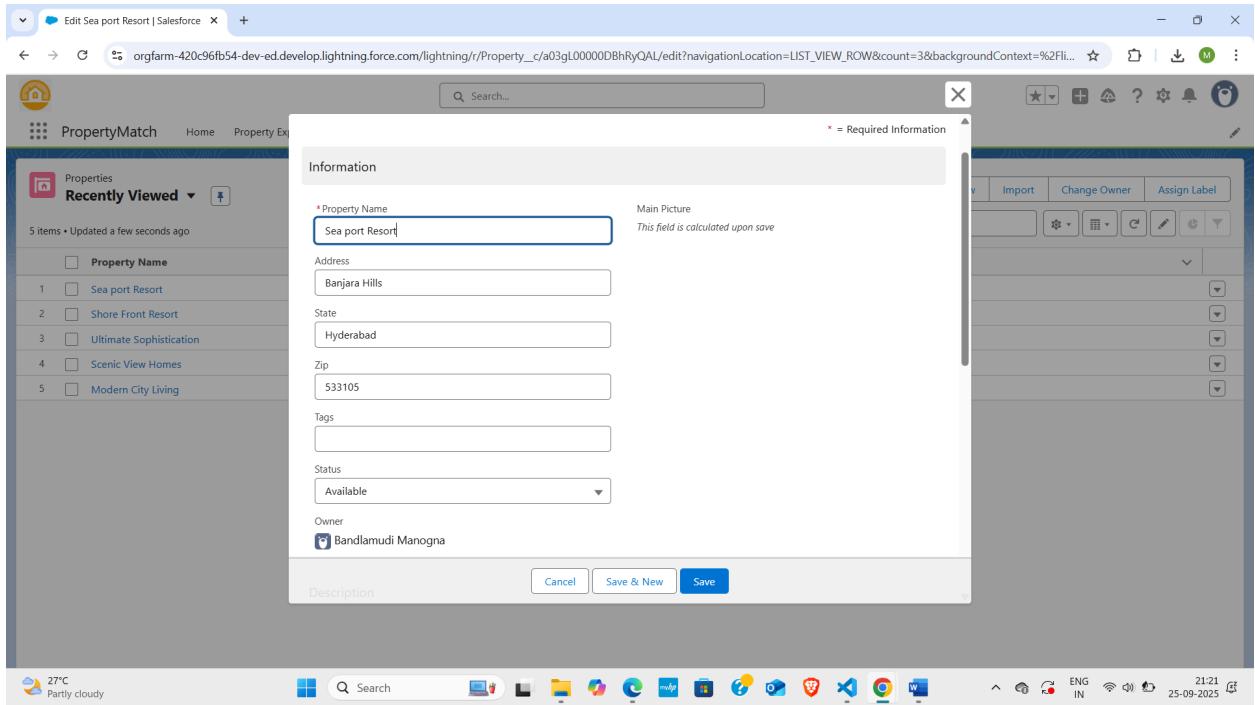
- **Use Case:** Track admin changes.
- **Implementation Steps:**
 - Setup → View Setup Audit Trail.
- **How it works:** Logged all setup/configuration changes for review.

The screenshot shows the 'View Setup Audit Trail' page in the Salesforce Setup interface. The page title is 'View Setup Audit Trail'. The left sidebar lists various setup categories: Health Check, Login Access Policies, Named Credentials, Network Access, Password Policies, Platform Encryption (selected), Encryption Settings, Key Management, Private Connect, Remote Site Settings, Session Management, Session Settings, Sharing Settings, Trusted URL and Browser Policy Violations, Trusted URLs, Trusted URLs for Redirects, and 'View Setup Audit Trail' (which is highlighted). The main content area displays a table of audit logs with columns for Date, Time, Action, Details, and Type. The logs show various system activities such as permission set assignments, organization setup changes, and rule activations. At the bottom of the page, there is a link to 'Download setup audit trail for last six months (Excel .csv file)'.

Phase 10: Quality Assurance Testing

Test Case 1: Property Record Creation

- **Use Case / Scenario:** Create a new property record with required details.
- **Test Steps (Input):**
 - Navigate to **Properties** → **New**.
 - Enter City = “Hyderabad”, State = “Telangana”, Price = 50,00,000, Bedrooms = 3, Bathrooms = 2.
 - Assign to Broker.
- **Expected Result:**
 - Property record should save successfully and appear in list view.
- **Actual Result:**
 - Property saved successfully, record visible under Properties tab.



Test Case 2: Email Notification to Broker on Property Assignment

Use Case / Scenario:

When a new property record is created and assigned to a broker, the system should automatically send an email notification to the broker with property details.

Test Steps (with Input):

1. Navigate to **Properties Tab** → **New Property**.
2. Enter property details:
3. Click **Save**.

Expected Result:

- A property record is created in Salesforce.
- An **email is triggered automatically** to the assigned broker.
- Email contains property name, price, city, and assigned broker details.

PropertyMatch - Sea port Resort

State: Hyderabad Asking Price: ₹20,00,000 Beds: 3 Baths: 2

Broker

Broker Name: Charitha Bandlamudi Phone: 617-244-3672
Mobile Phone: 617-244-3672 Email: charithasree.bandlamudi@gmail.com

Banjara Hills

There are currently no pictures for this property.
Add picture
Upload Files Or drop files

Actual Result (Obtained Email):

Gmail in:spam

New Property Created: modern

Bandlamudi Manogna via znuq4n318msw.gl-7pvbruua.can98.bnc.salesforce.com to me 11:30 (50 minutes ago)

Why is this message in spam? This message is similar to messages that were identified as spam in the past.

Hello!! A new property has been created:
Address: jkc nagar
Assigned Broker:

Please log in to Salesforce for more details.

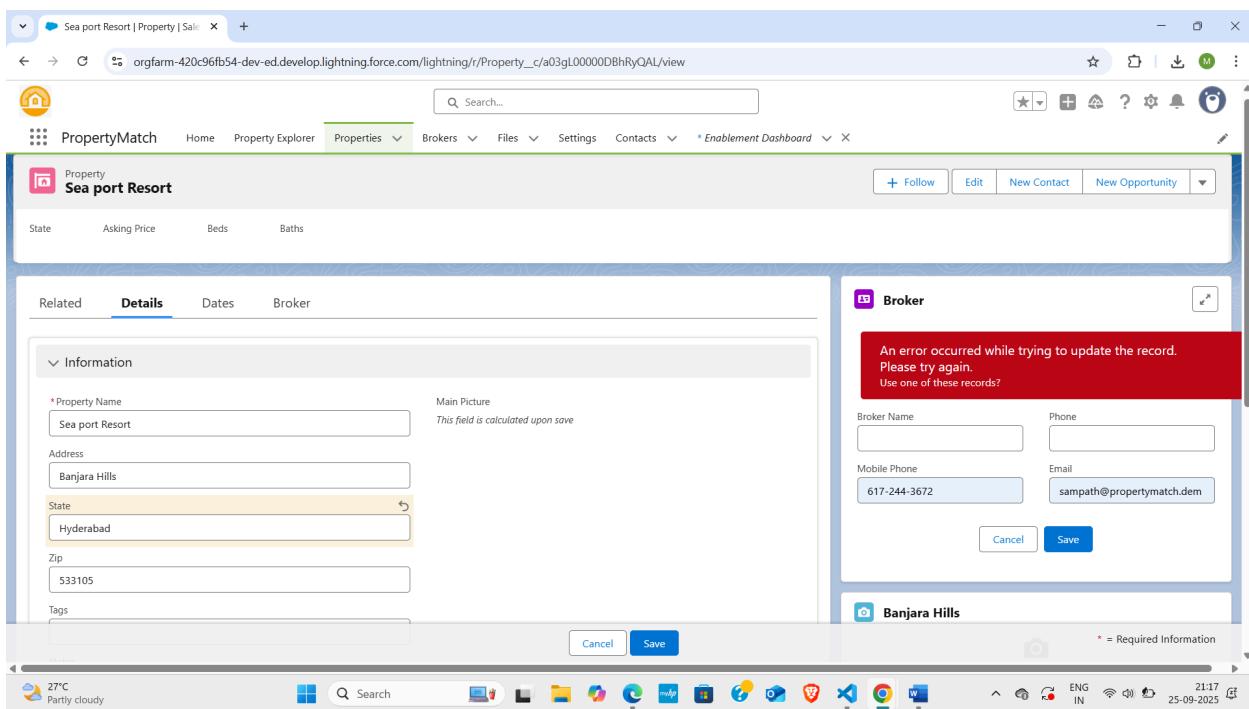
Thank you,
PropertyMatch CRM

Report as not spam

Status: Test Passed – Email triggered successfully with correct property details.

Test Case 3: Duplicate Rule (Broker Email)

- **Use Case / Scenario:** Prevent duplicate broker creation.
- **Test Steps (Input):**
 - Try creating a new Broker with Email = "sampath@propertymatch.com" (already exists).
- **Expected Result:**
 - Error preventing duplicate broker.
- **Actual Result:**
 - Error displayed.

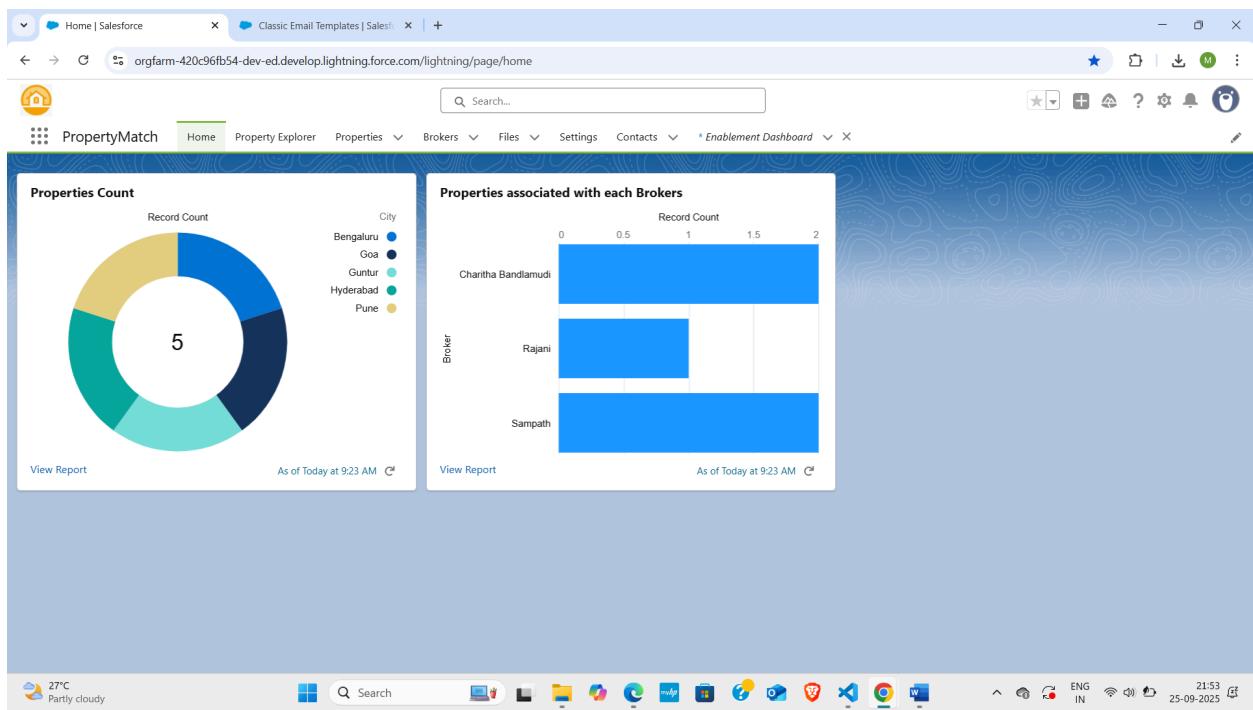


Test Case 4: Report & Dashboard Update

- **Use Case:** When a new Property and Broker are created, the “Properties count” and “Properties assigned to each broker’s” report and dashboard should reflect

the increased count.

- **Steps:** Create new Property = *Sea port Resort, Hyderabad* assigned to *Charitha Bandlamudi*.
- **Expected Result:** Report shows increased property count for Hyderabad; Dashboard chart updates automatically.
- **Actual Result:** Counts updated successfully



Test Case 5: SOQL Query Execution

- **Use Case:** Validate SOQL returns correct filtered property list.
- **Steps:** Run query → `SELECT Name, City__c FROM Property__c WHERE Price__c <= 1560000.`
- **Expected Result:** Returns all properties $\leq 1560,000$.
- **Actual Result:** Correct filtered list returned.

The screenshot shows the Salesforce Developer Console interface. At the top, there's a navigation bar with links like File, Edit, Debug, Test, Workspace, Help, and tabs for various apex classes. Below the navigation is a query result grid titled "Query Results - Total Rows: 3". The grid has two columns: "Name" and "City__c". The data rows are:

Name	City__c
Ultimate Sophistication	Bengaluru
Scenic View Homes	Pune
Shore Front Resort	Goa

Below the grid is a toolbar with buttons for "Query Grid", "Save Rows", "Insert Row", "Delete Row", and "Refresh Grid". To the right of the grid are buttons for "Access in Salesforce", "Create New", "Open Detail Page", and "Edit Page".

The main content area contains the "Query Editor" tab, which is currently selected. It shows a query editor window with the following SQL code:

```
SELECT Name, City__c FROM Property__c WHERE Price__c <= 1560000
```

Below the query editor is a message: "Any query errors will appear here..."

On the right side of the interface, there's a "History" section showing previously executed queries. The first query in the history is:

```
SELECT Id, Name, City__c, State__c, Price__c, Beds__c, Baths__c, Bro...
```

At the bottom of the interface, there's a toolbar with icons for Save, Undo, Redo, and other developer tools. The system status bar shows the date and time as "25-09-2025 22:03".

Conclusion

The **PropertyMatch – Real Estate CRM** project successfully addressed the key challenges in the real estate domain by providing a centralized, Salesforce-based platform for managing properties, brokers, and client interactions.

Through the use of **custom objects (Property, Broker)**, **Lightning Web Components**, and **Apex Controllers**, the application enabled users to filter properties, view detailed records, and visualize locations using **Google Maps integration with OAuth 2.0**. Administrators could seamlessly create and assign properties, while **flows and email alerts** ensured that brokers were instantly notified of new assignments.

The project also implemented strong **data management practices** (Data Import Wizard, Data Loader, Duplicate Rules, Data Export), along with robust **security and sharing settings** (Profiles, Roles, OWD, Sharing Rules, Field-Level Security), ensuring compliance and controlled data access. **Reports and Dashboards** provided actionable insights into property distribution, broker performance, and city-wise trends.

Comprehensive **testing and quality assurance** validated all features, from record creation to automation, integration, and security, guaranteeing system reliability and user satisfaction.

In conclusion, **PropertyMatch – Real Estate CRM** delivers a scalable, secure, and user-friendly solution that enhances operational efficiency for administrators, improves visibility for brokers, and provides an intuitive property search experience for end-users. This implementation not only fulfills the initial requirements but also establishes a strong foundation for future enhancements such as advanced analytics, AI-driven property recommendations, and mobile-first access.

Future Enhancements

In the future, this Real Estate CRM can be enhanced with advanced features such as integrating a chatbot for instant customer assistance, AI-based property recommendations based on user preferences, and predictive analytics for pricing trends. Mobile app support and integration with third-party payment or booking systems can also be included to make the platform more interactive, user-friendly, and business-ready.