SALESFORCE PROJECT

PROJECT TITTLE :RealEstate CRM (Property & Agent Management)

PROBLEM: Real estate firms struggle with property listings, agent performance, and client leads. Solution

1. Classes & Objects

- **Apex Class**: Encapsulates business logic.
- **Objects**: Custom Salesforce objects (Property__c, Agent__c, Client__c, Deal__c, Contract__c).

```
public with sharing class PropertyRefGenerator {
    public static void assignRefs(List<Property__c> newProperties){
        for(Property__c p : newProperties){
            if(String.isBlank(p.Property_Ref__c)){
                String year = String.valueOf(Date.today().year());
                String rand = String.valueOf(Math.abs(Crypto.getRandomInteger())).right(6);
                p.Property_Ref__c = 'PROP-' + year + '-' + rand;
            }
        }
    }
}
```

2. Apex Triggers (before/after insert/update/delete)

- **Before Insert**: Good for setting field values before save.
- **After Insert**: Good for operations requiring the record ID (like creating related records).

PropertyTrigger.apxt *

Code Coverage: None API Version: 64

```
trigger PropertyTrigger on Property__c (before insert) {
    if(Trigger.isBefore && Trigger.isInsert){
        PropertyRefGenerator.assignRefs(Trigger.new);
    }
}

```

3. Trigger Design Pattern

- **One Trigger per Object**
- **Handler Class** for logic (bulk-safe, reusable).
- **Example Structure**:

PropertyTrigger.trigger

PropertyTriggerHandler.cls

File Edit Debug Test Workspace Help < >

PropertyTrigger.apxt * PropertyTriggerhand.apxt *

Code Coverage: None API Version: 64

```
public class PropertyTriggerHandler {
    public static void beforeInsert(List<Property__c> newList){
        // Call helper class
        PropertyRefGenerator.assignRefs(newList);
    }
}

```

4. SOQL & SOSL

- **SOQL**: Query Salesforce objects

List<Property__c> props = [SELECT Id, Name, Price__c, City__c FROM Property__c WHERE Price__c > 500000];

- **SOSL**: Full-text search across multiple objects

List<List<SObject>> searchResults = [FIND 'Villa*' IN ALL FIELDS RETURNING Property__c(Name, City__c), Deal

5. Collections: List, Set, Map

File Edit Debug Test Workspace Help

PropertyTrigger.apxt * | PropertyTriggerhand.apxt * | Deal.apxt *

Code Coverage: None | API Version: 64

```
List<Deal__c> dealList = new List<Deal__c>();
Set<Id> agentIds = new Set<Id>();
Map<Id, Property__c> propMap = new Map<Id, Property__c>();

for(Deal__c d : [SELECT Id, Property__c FROM Deal__c]){
    propMap.put(d.Property__c, null); // example placeholder
    agentIds.add(d.Agent__c);
    dealList.add(d);
}
```

6. Batch Apex

File Edit Debug Test Workspace Help

PropertyTrigger.apxt * | PropertyTriggerhand.apxt * | BatchUpdateProperty.apxc *

Code Coverage: None | API Version: 64

```
}global class BatchUpdateProperty implements Database.Batchable<sObject> {
    global Database.QueryLocator start(Database.BatchableContext BC){
        return Database.getQueryLocator([SELECT Id, Status__c FROM Property__c WHERE Status__c='Available']);
    }

    global void execute(Database.BatchableContext BC, List<Property__c> scope){
        for(Property__c p : scope){
            p.Status__c = 'Active';
        }
        update scope;
    }

    global void finish(Database.BatchableContext BC){
        System.debug('Batch Complete');
    }
}
```

Execute: Database.executeBatch(new BatchUpdateProperty(), 200);

8. Queueable Apex

- **Use Case**: Chainable asynchronous jobs, heavier processing than future methods.

```
public class UpdateAgentMetrics implements Queueable {
    public void execute(QueueableContext ctx){
        List<Agent__c> agents = [SELECT Id, Total_Deals__c FROM Agent__c];
        for(Agent__c a : agents){
            a.Total_Deals__c = [SELECT count() FROM Deal__c WHERE Agent__c=:a.Id];
        }
        update agents;
    }
}
```

.9)Scheduled Apex

- **Use Case**: Send daily follow-up reminders or generate weekly reports.

```
global class DailyReminderScheduler implements Schedulable {
    global void execute(SchedulableContext sc){
        System.enqueueJob(new SendVisitReminderQueueable());
    }
}

// Schedule via UI or Apex:
// '0 0 8 * * ?' → every day at 8 AM
System.schedule('Daily Visit Reminder', '0 0 8 * * ?', new DailyReminderScheduler());
```

10. Future Methods

Use Case: Callouts (e.g., Google Maps API) asynchronously.

```
@future(callout=true)
public static void geocodeProperty(String address, Id propertyId){
    // Callout to Google Maps API to get lat/lng
}
```

11. Exception Handling

```
try{
    update props;
} catch(DmlException e){
    System.debug('Error updating properties: ' + e.getMessage());
}
```

12. Test Classes

- **Requirement**: ≥ 75% coverage for deployment.

```
@IsTest
private class PropertyRefGeneratorTest {
    @IsTest static void testAssignRefs(){
        List<Property__c> props = new List<Property__c>{
            new Property__c(Name='Test Villa')
        };
        Test.startTest();
        PropertyRefGenerator.assignRefs(props);
        Test.stopTest();

        System.assertNotEquals(null, props[0].Property_Ref__c, 'Property Ref should be assigned');
    }
}
```

13. Asynchronous Processing

- **Batch Apex** → for very large data sets
- **Queueable Apex** → for jobs requiring chaining
- **Future Methods** → simple asynchronous tasks (like callouts)
- **Scheduled Apex** → time-based execution