

SALESFORCE PROJECT

PROJECT TITLE :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
1 trigger PropertyTrigger on Property__c (before insert) {
2     if(trigger.isBefore && trigger.isInsert){
3         PropertyRefGenerator.assignRefs(trigger.new);
4     }
5 }
6
```

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
1 public class PropertyTriggerHandler {
2     public static void beforeInsert(List<Property__c> newList){
3         // Call helper class
4         PropertyRefGenerator.assignRefs(newList);
5     }
6 }
7
```

4. SOQL & SOSL

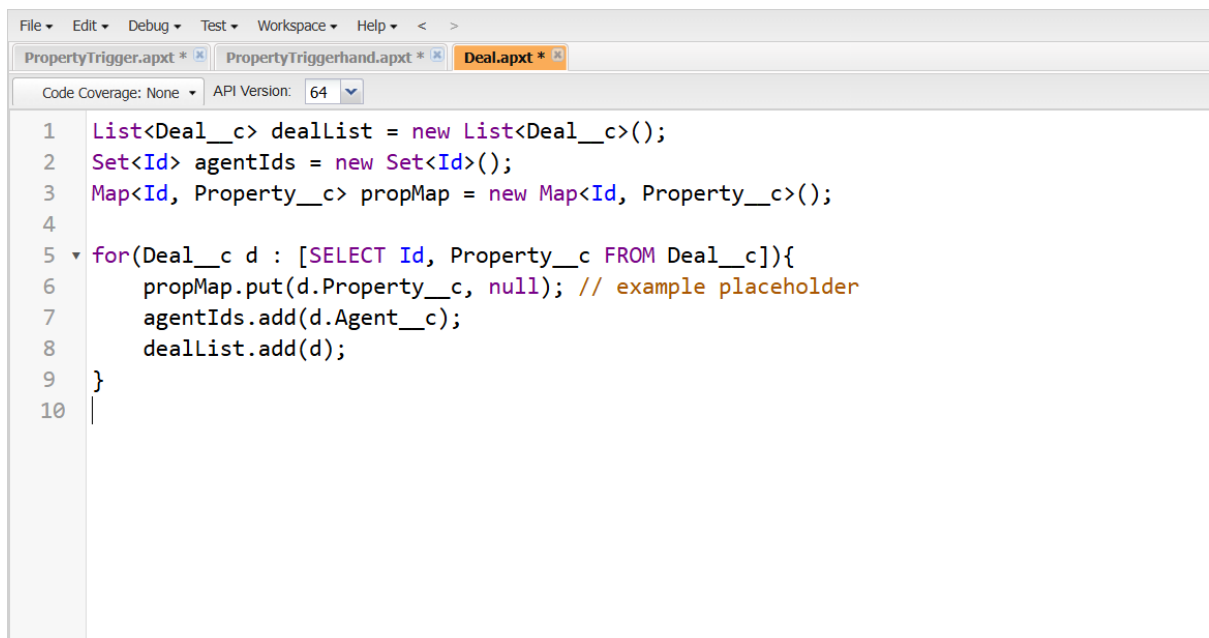
- **SOQL:** Query Salesforce objects

```
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
```

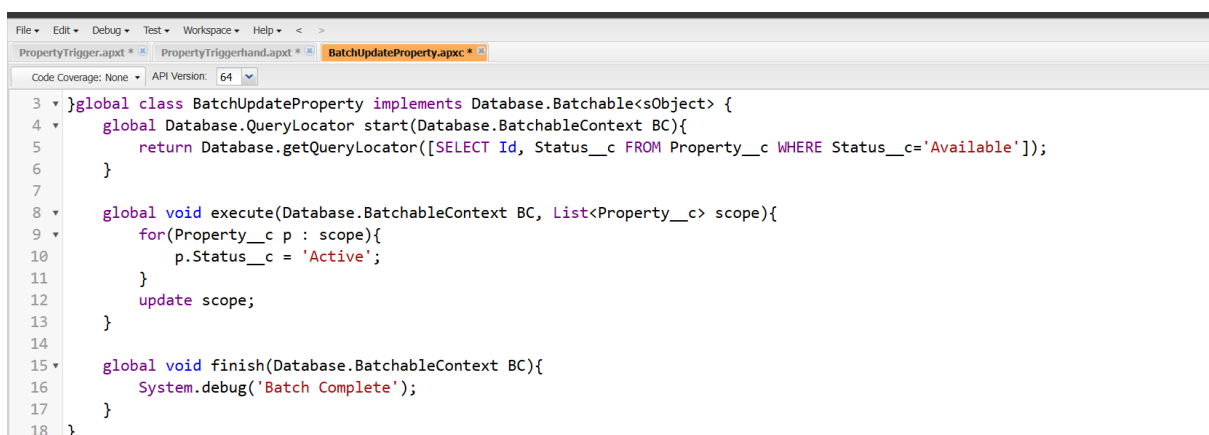
5. Collections: List, Set, Map



The screenshot shows an IDE window with three tabs: PropertyTrigger.apxt, PropertyTriggerhand.apxt, and Deal.apxt. The Deal.apxt tab is active, showing the following Apex code:

```
1 List<Deal__c> deallist = new List<Deal__c>();  
2 Set<Id> agentIds = new Set<Id>();  
3 Map<Id, Property__c> propMap = new Map<Id, Property__c>();  
4  
5 for(Deal__c d : [SELECT Id, Property__c FROM Deal__c]){  
6     propMap.put(d.Property__c, null); // example placeholder  
7     agentIds.add(d.Agent__c);  
8     deallist.add(d);  
9 }  
10 |
```

6. Batch Apex



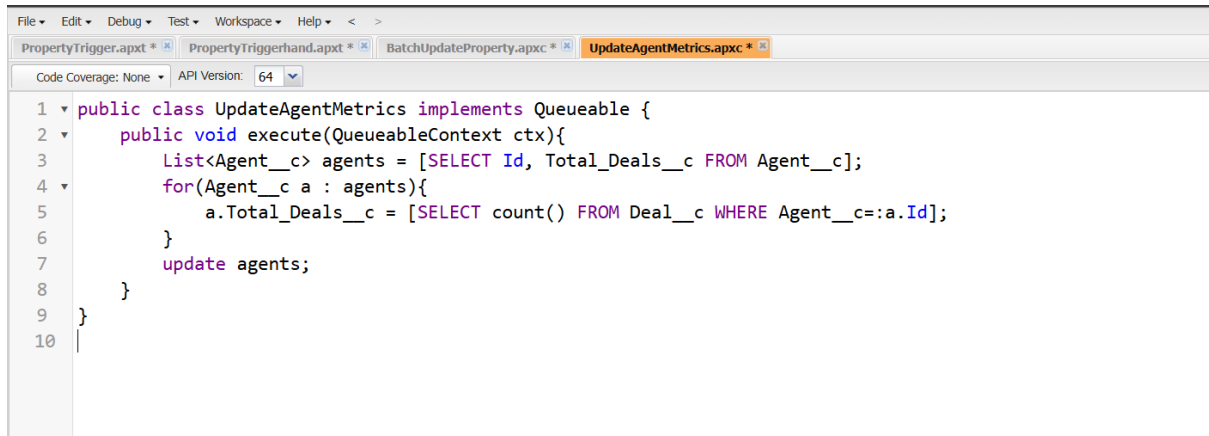
The screenshot shows an IDE window with three tabs: PropertyTrigger.apxt, PropertyTriggerhand.apxt, and BatchUpdateProperty.apxc. The BatchUpdateProperty.apxc tab is active, showing the following Apex code:

```
3 }global class BatchUpdateProperty implements Database.Batchable<SObject> {  
4     global Database.QueryLocator start(Database.BatchableContext BC){  
5         return Database.getQueryLocator([SELECT Id, Status__c FROM Property__c WHERE Status__c='Available']);  
6     }  
7  
8     global void execute(Database.BatchableContext BC, List<Property__c> scope){  
9         for(Property__c p : scope){  
10             p.Status__c = 'Active';  
11         }  
12         update scope;  
13     }  
14  
15     global void finish(Database.BatchableContext BC){  
16         System.debug('Batch Complete');  
17     }  
18 }
```

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

8. Queueable Apex

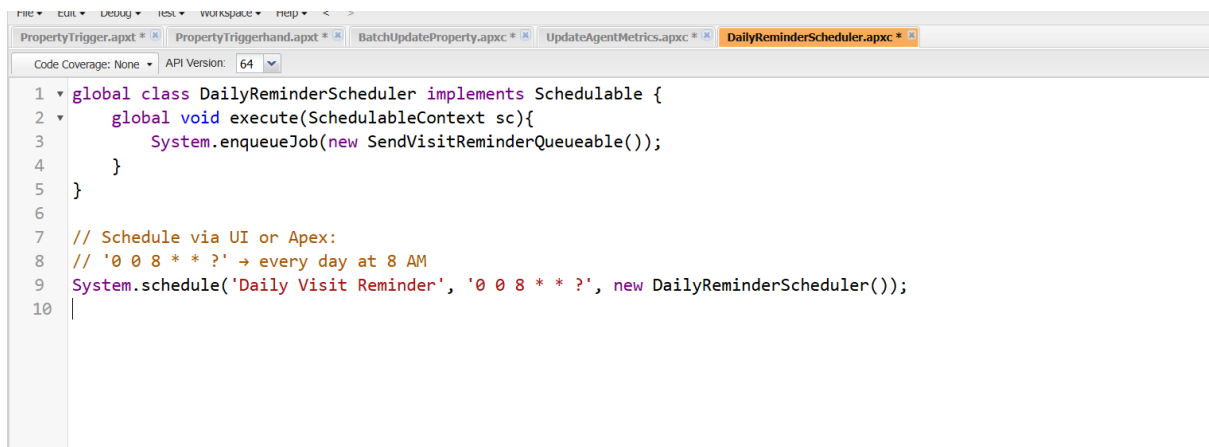
- **Use Case:** Chainable asynchronous jobs, heavier processing than future methods.



```
File Edit Debug Test Workspace Help < >
PropertyTrigger.apxt * PropertyTriggerhand.apxt * BatchUpdateProperty.apxc * UpdateAgentMetrics.apxc *
Code Coverage: None API Version: 64
1 public class UpdateAgentMetrics implements Queueable {
2     public void execute(QueueableContext ctx){
3         List<Agent__c> agents = [SELECT Id, Total_Deals__c FROM Agent__c];
4         for(Agent__c a : agents){
5             a.Total_Deals__c = [SELECT count() FROM Deal__c WHERE Agent__c=:a.Id];
6         }
7         update agents;
8     }
9 }
10 |
```

9) Scheduled Apex

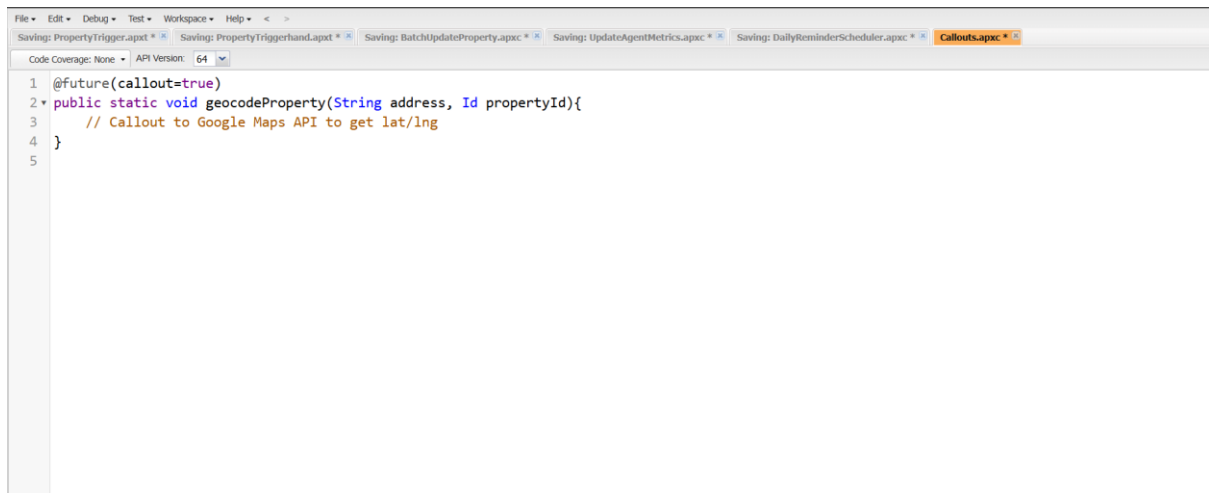
- **Use Case:** Send daily follow-up reminders or generate weekly reports.



```
File Edit Debug Test Workspace Help < >
PropertyTrigger.apxt * PropertyTriggerhand.apxt * BatchUpdateProperty.apxc * UpdateAgentMetrics.apxc * DailyReminderScheduler.apxc *
Code Coverage: None API Version: 64
1 global class DailyReminderScheduler implements Schedulable {
2     global void execute(SchedulableContext sc){
3         System.enqueueJob(new SendVisitReminderQueueable());
4     }
5 }
6
7 // Schedule via UI or Apex:
8 // '0 0 8 * * ?' -> every day at 8 AM
9 System.schedule('Daily Visit Reminder', '0 0 8 * * ?', new DailyReminderScheduler());
10 |
```

10. Future Methods

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



The screenshot shows an IDE window with the file `Callouts.apxc` open. The code is as follows:

```
1 @future(callout=true)
2 public static void geocodeProperty(String address, Id propertyId){
3     // Callout to Google Maps API to get lat/lng
4 }
5
```

11. Exception Handling

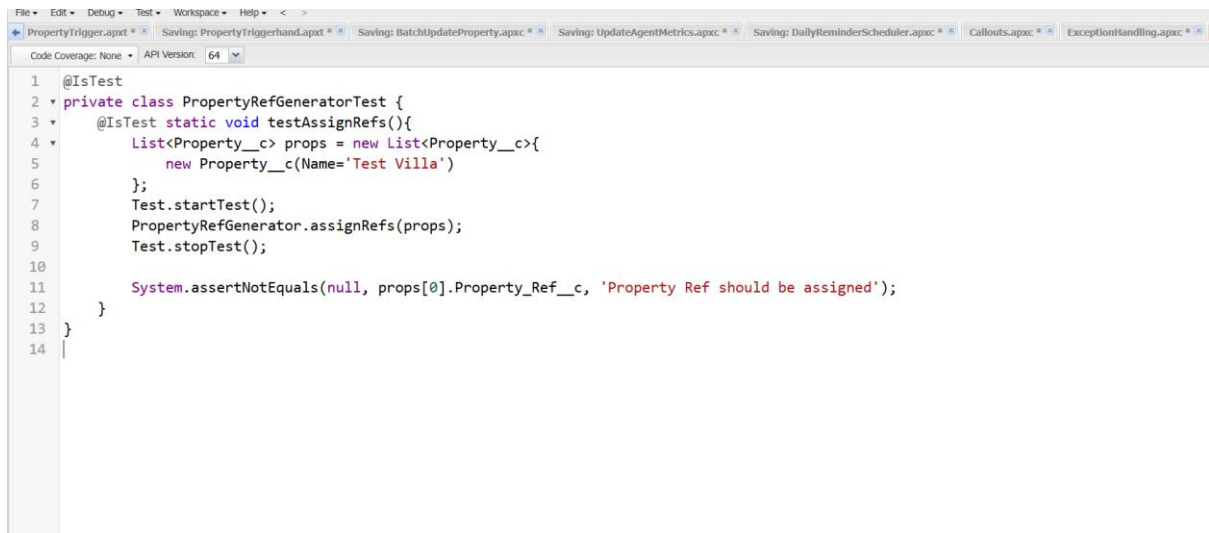


The screenshot shows an IDE window with the file `ExceptionHandling.apxc` open. The code is as follows:

```
1 try{
2     update props;
3 } catch(DmlException e){
4     System.debug('Error updating properties: ' + e.getMessage());
5 }
6
```

12. Test Classes

- **Requirement:** $\geq 75\%$ coverage for deployment.

A screenshot of an IDE window showing Apex code. The top bar includes menus like File, Edit, Debug, Test, Workspace, and Help. Below the menu bar, there are several tabs for different Apex classes: PropertyTrigger.apxt, PropertyTriggerHandler.apxt, BatchUpdateProperty.apxc, UpdateAgentMetrics.apxc, DailyReminderScheduler.apxc, Callouts.apxc, and ExceptionHandling.apxc. The main editor area shows the code for PropertyRefGeneratorTest. The code is as follows:

```
1  @IsTest
2  private class PropertyRefGeneratorTest {
3      @IsTest static void testAssignRefs(){
4          List<Property__c> props = new List<Property__c>{
5              new Property__c(Name='Test Villa')
6          };
7          Test.startTest();
8          PropertyRefGenerator.assignRefs(props);
9          Test.stopTest();
10
11          System.assertNotEquals(null, props[0].Property_Ref__c, 'Property Ref should be assigned');
12      }
13  }
14  |
```

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