SALESFORCE DEVELOPER

Salesforce Developer Catalyst

Shahi Shreshth

Mentor Name - Sai Manikh

Content

Programming Part -1

a) Get Started with Apex Trigger

1. First Make a new field with name "`Match Billing Address`"

```
  a) Datatype : Checkbox
  b) All the field level should be visible.
2. Go to Setting icon and click on Developer console after this
   a) create a new apex trigger by click on File and assign
name
      and object
      i) Name : - AccountAddressTrigger
      ii) Object : - account
   AccountAddressTrigger.apxt
   Code:-
```

```
trigger AccountAddressTrigger on Account (before insert ,
before update)
{
for(Account account:Trigger.New){
   if(account.Match_Billing_Address__c == True){
     account.ShippingPostalCode = account.BillingPostalCode;
}
}
}
```

```
3. Save
4. Go to Sales App -> Account -> Dickson plc.
5. Checkbox to Match Billing Address.
```

b) Build Apex Trigger

```
1. Go to Setting icon and click on Developer console after this
   a) create a new apex trigger by click on File and and assign
name
      and object
      i) Name : - ClosedOpportunityTrigger
      ii) Object : - **Opportunity**
   ClosedOpportunityTrigger.apxt
```

```
//after inser1t and after update check after user enter
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
 List<Task> tasklist =new List<Task>();
    for(Opportunity opp: Trigger.New){
        if(opp.StageName == 'Closed Won'){
            tasklist.add(new Task(Subject='Follow Up Test
        }
    }
//if your lask list contains atleast one task then it will insert tasklist
    If(tasklist.size()>0){
        insert tasklist;

    }
}
```

2. Save it and Go to Sales app and go to Opportunity tab ,then go to account and write next step you want to see in next step.Then check and run.

c) Get Started with Apex Unit Test

```
1. Go to Setting icon and click on Developer console after this
   a) create a new apex class by click on File and and assign
name
      and object
      i) Name : – VerifyDate
   VerifyDate.apxc
```

```
public class VerifyDate {

    //method to handle potential checks against two dates
    public static Date CheckDates(Date date1, Date date2) {
        //if date2 is within the next 30 days of date1, use date2.  Otherwise use the end of the month
        if(DateWithin30Days(date1,date2)) {
            return date2;
        } else {
            return SetEndOfMonthDate(date1);
        }
    }

    //method to check if date2 is within the next 30 days of date1
@TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
        //check for date2 being in the past
    if( date2 < date1) { return false; }

    //check that date2 is within (>=) 30 days of date1
    Date date30Days = date1.addDays(30); //create a date 30 days away from date1
        if( date2 >= date30Days ) { return false; }
        else { return true; }
    }

    //method to return the end of the month of a given date
    @TestVisible private static Date SetEndOfMonthDate(Date date1) {
          Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
          Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
          return lastDay;
    }

}
//make @test visible used when specifier is private

```

2. Create a new Test class "`TestVerifyDate`" `to test the custom data.`

```
@isTest                         //Check
private class TestVerifyDate{
@isTest  static void Test_CheckDates_case1(){
Date D = VerifyDate.CheckDates(date.parse('01/01/2022'), date.parse('01/05/2022'));
System.assertEquals(date.parse('01/05/2022'), D);
}

    @isTest  static void Test_CheckDates_case2(){
Date D = VerifyDate.CheckDates(date.parse('01/01/2022'), date.parse('05/05/2022'));
System.assertEquals(date.parse('01/31/2022'), D);
    }

    @isTest  static void Test_DateWithin30Days_case1(){
Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'), date.parse('12/30/2021'));
System.assertEquals(false,flag);
```

```
    }
     @isTest  static void Test_DateWithin30Days_case2(){
Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'), date.parse('02/02/2021'));
System.assertEquals(false,flag);
    }

     @isTest  static void Test_DateWithin30Days_case3(){
Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2022'), date.parse('01/15/2022'));
System.assertEquals(false,flag);
    }
//If the dates is in end of month
    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2022'));
    }

}

```

3. Save All and Run it.

d) Test Apex Trigger

```
1. Go to Setting icon and click on Developer console after this
   a) create a new apex trigger by click on File and and assign
name
      and object
      i) Name : - RestrictContactByName
      ii) Object : - Account
   RestrictContactByName.apxt
```

```
trigger RestrictContactByName on Contact (before insert, before update) {

 //check contacts prior to insert or update for invalid data
 For (Contact c : Trigger.New) {
        if(c.LastName == 'INVALIDNAME') {  //invalidname is invalid
                c.AddError('The Last Name "'+c.LastName+'" is

        }

 }



}
```

2) Make a new class TestRestrictContactByName
TestRestrictContactByName.apxc

```
@isTest
public class TestRestrictContactByName {

    @isTest static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';
//start testing
        Test.startTest();
        Database.SaveResult result = Database.insert(cnt,false);
//after inserting database succesfully it need to stop to due to further overrideing
        Test.stopTest();
```

```
//check if the result got inserted or not
```

```
        System.assert(!result.isSuccess());
//check if the size of the error is greater than 0 or not
        System.assert(result.getErrors().size()>0);

        System.assertEquals('The Last Name "INVALIDNAME" is not

    }
}
```

3) save it and run it.

e) Create Test Data for Apex Tests

```
1. Go to Setting icon and click on Developer console after this
   a) create a new apex class by click on File and and assign
name
      and object
```

Name :- `RandomContactFactory`

`RandomContactFactory.apxc`

```
 public class RandomContactFactory {
//get total contact number and lastname of contacts
    public static List<Contact> generateRandomContacts(Integer
numcnt , string lastname){
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt = new Contact(FirstName = 'Test'+i
,LastName = lastname);
            contacts.add(cnt);
        }
        return contacts;
    }

}
```

2. Save it and run it.

f) Use Future Methods

```
1. First Create a new field in the object called "Account"
   Field name :-Number Of Contacts
   Field type :-Number
```

2. Now create an apex class and assign its name i.e `AccountProcessor with method name countContacts.It should accept the list of Account Ids.`

`AccountProcessor.apxc`

```
    public class AccountProcessor {
 @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accountsToUpdate = new List<Account>();
//select id, name ,(id from contacts) from account
        List<Account> accounts =[Select Id, Name, (Select Id from Contacts) from Account Where Id in :accountIds];
//looping in accounts to add number of contacts insert in accounts and update it.
        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);

        }
        update accountsToUpdate;
    }
}
```

3. Now enter apex code for any account ,choose any account e.g Dickson Account. and copy the ID from the url ....lightining/Account/{account_id}/view.
Enter debug code :-

```
List<Id> accountIds = new List<Id>();
accountIds.add('{account_id}');
AccountProcessor.countContacts(accountIds);
```

```
//to check if it work or give error
```

Execute the code .

5. Now create a new class with name " `AccountProcessorTest" to test the Accountprocessor class with custom data to check.`

```
@isTest
private class AccountProcessorTest {
@isTest
//check if new contacts added
    private static void testCountContacts(){
        Account newAccount = new Account(Name='Test

        insert newAccount;

        Contact newContact1 = new Contact(FirstName='John',
LastName='Doe', AccountId = newAccount.Id);
        insert newContact1;

        Contact newContact2 = new Contact(FirstName='Jane',
LastName='Doe', AccountId = newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
    }

}
```

g) Use Batch Apex

1. Create an apex class "LeadProcessor" with Interface called "Database.Batchable" with QueryLocator to collect all Lead Records.

```
global class LeadProcessor implements Database.Batchable<sObject> {

    global Integer count =0;

    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID,

    }
    global void execute (Database.BatchableContext bc, List<Lead> L_list){
        List<lead> L_list_new = new List<lead>();

        for(lead L:L_list){
            L.leadsource= 'Dreamforce';
            L_list_new.add(L);
            count +=1;
        }
        update L_list_new;
    }
    global void finish(Database.BatchableContext bc){
        system.debug('count = '+ count);
    }
}
```

2. create an apex test class `LeadProcessorTest for check custom data 200 records.`

h) Control Process with Queueable Apex

`1. Create an apex class "AddPrimaryContact" with an interface called "Queueable" .With parameters (`Contact sObject, State) in class as constructor parameter.
2. The `execute` method must query for a maximum of 200 Accounts with the `BillingState` specified by the State abbreviation passed into the constructor and insert the Contact sObject record associated to each Account. Look at the sObject `clone()` method.

```
public class AddPrimaryContact implements Queueable{

    private Contact con;
    private String state;

    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;

    }
    public void execute(QueueableContext context){
        List<Account> accounts =[Select Id, name, (Select FirstName ,LastName ,Id from contacts)
                                        from Account where BillingState = :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);
        }
        if(primaryContacts.size() > 0){
            insert primaryContacts;
        }
    }

}
```

3. create an Apex class test "AddPrimaryContactTest".

```
@isTest
public class AddPrimaryContactTest {
    static testmethod void testQueueable(){
        List<Account> testAccounts = new List<Account>();
        for(Integer i=0;i<50;i++ ){
            testAccounts.add(new Account(Name ='Account'+i, BillingState='CA'));
        }
        for(Integer j=0;j<50;j++ ){
            testAccounts.add(new Account(Name ='Account'+j, BillingState='NY'));
        }
        insert testAccounts;
        Contact testContact = new Contact(FirstName ='John', LastName ='Doe');
        insert testContact;

        AddPrimaryContact addit = new addPrimaryContact(testContact,'CA');

        Test.startTest();
        system.enqueueJob(addit);
        Test.stopTest();

        System.assertEquals(50, [Select count() from Contact where accountId in (Select Id from Account where BillingState='CA') ]);
    }
}
```

i) Schedule Jobs Using Apex Scheduler

`1. Create an apex class "DailyLeadProcessor" with an interface called "Schedulable" .`

```
public class DailyLeadProcessor implements Schedulable {
    public void execute(SchedulableContext SC){
        List<Lead> LeadObj= [SELECT Id from Lead where LeadSource = null limit 200];
        for(Lead l:LeadObj){
            l.LeadSource ='Dreamforce';
            update l;
        }
    }
}
```

2. create an apex test class "`DailyLeadProcessorTest`" .

```
@isTest
private class DailyLeadProcessorTest {
    static testMethod void testDailyLeadProcessor(){
        String CRON_EXP = '0 0 1 * * ?';
        List<Lead> lList = new List<Lead>();
        for(Integer i = 0; i < 200; i++){
            lList.add(new Lead(LastName = 'Dreamforce' + i, Company = 'Test1 Inc.',Status='Open - Not Contacted'));
        }
        insert lList;

        Test.startTest();
        String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new DailyLeadProcessor());
}
    }
```

j) Apex REST Callout

a) Create an apexclass :

Name : `AnimalLocator`

Method name: `getAnimalNameById`

```
public class AnimalLocator {
    public static String getAnimalNameById(Integer animalId){
        String animalName;
        Http http = new Http();
        HttpRequest  request = new HttpRequest();
        request.setEndpoint
('https://th-apex-http-herokuapp.com/animals/' +  animalId);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

        if(response.getStatusCode() == 200 ){
            Map<String, Object> r = (Map<String, Object>)
                JSON.deserializeUntyped(response.getBody());
            Map<String, Object> animal = (Map<String, Object>)r.get('animal');
            animalName = string.valueOf(animal.get('name'));
        }
        return animalName;
    }

}
```

2. Create an Apex Test Class called "`AnimalLocatorTest`" `usses an mock class called " AnimalLocatorMock"` to mock the callout response.

`AnimalLocatorTest.apxc`

```
@isTest
private class AnimalLocatorTest {
@isTest static void getAnimalNameByIdTest(){
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String response = AnimalLocator.getAnimalNameById(1);

        System.assertEquals('chicken', response);
    }

}
```

3. Create an apex class called "`AnimalLocatorMock`" that was called in `AnimalLocatorTest.`

`AnimalLocatorMock.apxc`

```
                                    @isTest
global class AnimalLocatorMock implements HttpCalloutMock{

    global HTTPResponse respond(HTTPRequest request){
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animal":{"id":1, :"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

4. Save all
5. Run it.

k) Apex SOAP Callout

1. Copy the wsdl file as mentioned and named it as **parksServices.xml.**
2. create a remote setting named as "`ParkService`" .
3. And remote url: https://th-apex-soap-callout.herokuapp.com and save it.
4. Now generate a wsdl filw which you saved on your device
5. Change the name as `ParkService` .
6. Go to Setting (gear) icon then click on Developer console.
7. Create an apex class called "`ParkLocator`"

```
public class ParkLocator {
    public static List<String> country(String country){
        ParkService.ParksImplPort parkservice = new
parkService.ParksImplPort();
        return parkservice.byCountry(country);
    }

}
```

8. Create a test class called "`ParkLocatorTest`". Test class uses a mock class called `ParkServiceMock` to mock the callout response.

`ParkLocatorTest.apxc`

```
@isTest
private class ParkLocatorTest {
 @isTest static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new
ParkServiceMock());
        // Call the method that invokes a callout
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
         List<String> parks = new List<String>();
               parks.add('Yosemite');
               parks.add('Yellowstone');
```

```
                  parks.add('Another Park');
           // Verify that a fake result is returned
           System.assertEquals(parks, result);
     }
}
```

9. Create a new apex class called "`ParkServiceMock`".

`ParkServiceMock.apxc`

```
@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(
           Object stub,
           Object request,
           Map<String, Object> response,
           String endpoint,
           String soapAction,
           String requestName,
           String responseNS,
           String responseName,
           String responseType) {
        // start - specify the response you want to send
        List<String> parks = new List<string>();
              parks.add('Yosemite');
              parks.add('Yellowstone');
              parks.add('Another Park');
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
        response_x.return_x = parks;
        // end
        response.put('response_x', response_x);
   }
}
```

9. Save all and Run it.

I) Apex Web Services

a) Create an apex class called "`AccountManager`" and class have method called getAccount

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest request = RestContext.request;
        // grab the caseId from the end of the URL
         String accountId =
request.requestURI.substringBetween('/Accounts/','/contacts

        Account result =  [SELECT Id, Name ,(Select Id ,
Name from Contacts)from Account where Id=:accountId];
        return result;
    }

}
```

2. Create a Test class called "`AccountManagerTest`".

```
@IsTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =

'https://yourInstance.my.salesforce.com/services/apexrest/Account

        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);
    }
    // Helper method
    static Id createTestRecord() {
        // Create test record
        Account accountTest = new Account(Name='Test record');
        insert accountTest;

        Contact contactTest = new Contact(
        FirstName='John',
        LastName ='Doe',
        AccountId = accountTest.Id);
        insert contactTest;

        return accountTest.Id;
    }
}
```

Apex Specialist Superbadge

Pre requistes :-

1. Create a new Trailhead Playground .
2. Install the package.
3. Add 2 picklist Repair and Routine Maintainenance to the type field on the case Object.
4. Update the Case Page layout assignment "HoeWeRoll" Layout for your profile.
5. Rename the tab from Case -> Maintenance Request.
6. Update the Product Page layout assignment "HoeWeRoll" Layout for your profile.
7. Rename the tab for the product object -> Equipment.
8. Create default data by searching to generate a sample data for application.
9. In the process builder and assign relations between as given.

a) Automate Records Creation

1.click on app launcher -> go to (How we roll Maintenance) and then click on Maintenance Request.
2. Click on first case as mentioned and click on Details and chage
a) the type Repir -> Routine Maintenance.
b) origin : Phone
c) Vehicle : Teardrop Camper
3. Save it.
In the same page click on CloseCase and save it.
4. Go to the Object Manager then click on Maintenance Request and click on Field and Relationship and create a new field.
i) Type: Lookup Relationship
ii) Field Label : Equipment

Go to Developer console and create a new class MaintenanceRequestHelper.apxc
MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
public static void updateWorkOrders(List<Case> updWorkOrders,
Map<Id,Case> nonUpdCaseMap) {
    Set<Id> validIds = new Set<Id>();


    For (Case c : updWorkOrders){
        if (nonUpdCaseMap.get(c.Id).Status != 'Closed' &&
c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine

                validIds.add(c.Id);


            }
        }
    }

    if (!validIds.isEmpty()){
        List<Case> newCases = new List<Case>();
        Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT
Id, Vehicle__c, ProductId,
Product.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                                     FROM
Case WHERE Id IN :validIds]);
        Map<Id,Decimal> maintenanceCycles = new
Map<ID,Decimal>();
        AggregateResult[] results = [SELECT
Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN
:ValidIds GROUP BY Maintenance_Request__c];

    for (AggregateResult ar : results){
        maintenanceCycles.put((Id)
ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
    }
```

```

        for(Case cc : closedCasesM.values()){
            Case nc = new Case (
                ParentId = cc.Id,
            Status = 'New',
                Subject = 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle__c = cc.Vehicle__c,
                ProductId =cc.ProductId,
                Origin = 'Web',
                Date_Reported__c = Date.Today()

            );

            If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            }

            newCases.add(nc);
        }

       insert newCases;

       List<Equipment_Maintenance_Item__c> clonedWPs = new List<Equipment_Maintenance_Item__c>();
       for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp : closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);

            }
        }
        insert ClonedWPs;
    }
}
}
```

2. Create a new trigger MaitenanceRequest.apxt

```
trigger MaintenanceRequest on Case (before update, after update)
{

if(Trigger.isUpdate && Trigger.isAfter){

    MaintenanceRequestHelper.updateWorkOrders(Trigger.New,
Trigger.OldMap);

}
 }
```

3. Save all
4. Go to Lightining Experience Salesforce
5. Click on Maintenance Request ,clickon second case and then click Details :
a) Type : Change Repair to Routine Maintenance
b) Origin: Phone
 c) Vehicle : Teardrop Camper
5. Save all.
6. Go to the Feed and then click on Close Case and save it.

b) Synchronize Salesforce data with an External System

1.Go to remote setting ,create a new remote
a) Name: Warehouse
b) URL : https://th-superbadge-apex.herokuapp.com

2. Go to Developer called "WarehouseCalloutService.apxc "

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL =
'https://th-superbadge-apex.herokuapp.com/equipment';

    //@future(callout=true)
    public static void runWarehouseEquipmentSync(){

        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);


        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            for (Object eq : jsonResponse){
                Map<String,Object> mapJson =
(Map<String,Object>)eq;
```

```
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }

            if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was synced

                System.debug(warehouseEq);
            }

        }
    }
}
```

3. Save all.
4. Open execute and write the code.

```
System.enqueueJob(new WarehouseCalloutService());

```

c) **Schedule synchronization**

WarehouseSyncShedule.apxc

```
global with sharing class WarehouseSyncSchedule implements Schedulable{
    global void execute(SchedulableContext ctx){
        System.enqueueJob(new WarehouseCalloutService());
    }
}
```

Go to Lightning Salesforce Page then go to Apex class and Schedule Apex class:
a) Job Name: WarehouseSyncScheduleJob
b) Apex : WarhouseSyncSchedule
c) Click on all weekdays .
d) Assign start date and end date.
e) Assign time.

d) **Test Automation Logic**

Go to developer console and click to open "MaintenanceRequestHelperTest.apxc" with the following code:-

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine

    private static final string REQUEST_SUBJECT = 'Testing


    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new case(Type=REPAIR,
                           Status=STATUS_NEW,
                           Origin=REQUEST_ORIGIN,
                           Subject=REQUEST_SUBJECT,
                          ProductId=equipmentId,
                          Vehicle__c=vehicleId);
        return cs;
    }

    PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id requestId){
        Equipment_Maintenance_Item__c wp = new Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                                                              Maintenance_Request__c = requestId);
        return wp;
    }


    @istest
    private static void testMaintenanceRequestPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        Product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

        case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
        insert somethingToUpdate;

        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,somethingToUpdate.id);
        insert workP;

        test.startTest();
        somethingToUpdate.status = CLOSED;
        update somethingToUpdate;
        test.stopTest();

        Case newReq = [Select id, subject, type, ProductId, Date_Reported__c, Vehicle__c, Date_Due__c
```

```
                      from case
                      where status =:STATUS_NEW];

        Equipment_Maintenance_Item__c workPart = [select id
                                                 from Equipment_Maintenance_Item__c
                                                 where Maintenance_Request__c =:newReq.Id];

        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.ProductId, equipmentId);
        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
    }

    @istest
    private static void testMaintenanceRequestNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

        case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
        insert emptyReq;

        Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
        insert workP;

        test.startTest();
        emptyReq.Status = WORKING;
        update emptyReq;
        test.stopTest();

        list<case> allRequest = [select id
                                 from case];

        Equipment_Maintenance_Item__c workPart = [select id
                                                  from Equipment_Maintenance_Item__c
                                                  where Maintenance_Request__c = :emptyReq.Id];

        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }

    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

        for(integer i = 0; i < 300; i++){
           vehicleList.add(createVehicle());
            equipmentList.add(createEq());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
            requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
        }
        insert requestList;

        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id,
requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                 from case
                                 where status =: STATUS_NEW];

        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                        from Equipment_Maintenance_Item__c
                                                        where Maintenance_Request__c in: oldRequestIds];

        system.assert(allRequests.size() == 300);
    }
}
```

2. Then go to MaintainerRequestHelper.apxc with the following code:-

```
public with sharing class MaintenanceRequestHelper {
public static void updateWorkOrders(List<Case> updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
    Set<Id> validIds = new Set<Id>();


    For (Case c : updWorkOrders){
        if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine

                validIds.add(c.Id);


            }
        }
    }

    if (!validIds.isEmpty()){
        List<Case> newCases = new List<Case>();
        Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c, ProductId, Product.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
                                                     FROM Case WHERE Id IN :validIds]);
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
        AggregateResult[] results = [SELECT Maintenance_Request__c, MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

    for (AggregateResult ar : results){
        maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
    }

        for(Case cc : closedCasesM.values()){
            Case nc = new Case (
                ParentId = cc.Id,
            Status = 'New',
                Subject = 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle__c = cc.Vehicle__c,
                ProductId =cc.ProductId,
                Origin = 'Web',
                Date_Reported__c = Date.Today()

            );

            If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            }

            newCases.add(nc);
        }

       insert newCases;

       List<Equipment_Maintenance_Item__c> clonedWPs = new List<Equipment_Maintenance_Item__c>();
       for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp : closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items_

                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);

            }
        }
        insert ClonedWPs;
    }
}
```

```
}
```

3. Create an apex trigger called "MaintainenceRequest.apxt"

```
trigger MaintenanceRequest on Case (before update, after update) {

if(Trigger.isUpdate && Trigger.isAfter){

    MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);

}
 }
```

4. Save all.
5. Run all to check if all the test passed.

e) **Test Callout Logic**

1. Go to developer console and open WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-

    //@future(callout=true)
    public static void runWarehouseEquipmentSync(){

        Http http = new Http();
        HttpRequest request = new HttpRequest();

         request.setEndpoint(WAREHOUSE_URL);
         request.setMethod('GET');
         HttpResponse response = http.send(request);


         List<Product2> warehouseEq = new List<Product2>();

         if (response.getStatusCode() == 200){
             List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
             System.debug(response.getBody());

             for (Object eq : jsonResponse){
                 Map<String,Object> mapJson = (Map<String,Object>)eq;
                 Product2 myEq = new Product2();
                 myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                 myEq.Name = (String) mapJson.get('name');
                 myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
```

```
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }

            if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was synced with

                System.debug(warehouseEq);
            }

        }
    }
}
```

2. open the class called "WarehouseCallotServiceTest.apxc" with the following code:-

```
@isTest

private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

3. Open "WarehouseCalloutServiceMock.apxc" with the following code.

```
@isTest
global class WarehouseCalloutServiceMock implements
HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request){

        System.assertEquals('https://th-superbadge-
));
        System.assertEquals('GET', request.getMethod());

        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement



        response.setStatusCode(200);
        return response;
    }
}
```

4. Save all
5. Run it all.

f) Test Scheduling Logic

1. Go to developer console and open WarehouseSyncSchedule.apxc with the following code.

```
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

2. Go to open "WarehouseSyncScheduleTest.apxc" with the following code.

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To

WarehouseSyncSchedule());
        Test.stopTest();
         //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX systems.
         // This object is available in API version 17.0 and later.
         CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
         System.assertEquals(jobID, a.Id,'Schedule ');


     }
}
```

Save all and run it.