Apex Triggers

Get Started with Apex Triggers

Create an Apex trigger

Create an Apex trigger that sets an account's Shipping Postal Code to match the Billing Postal Code if the Match Billing Address option is selected. Fire the trigger before inserting an account or updating an account.

Pre-Work:

Add a checkbox field to the Account object:

- Field Label: `Match Billing Address`
- Field Name: `Match_Billing_Address`

  Note: The resulting API Name should be `Match_Billing_Address__c`.
- Create an Apex trigger:
  - Name: `AccountAddressTrigger`
  - Object: **Account**
  - Events: before insert and before update
  - Condition: Match Billing Address is true
  - Operation: set the Shipping Postal Code to match the Billing Postal Code

CODE SOLUTION:

Apex Trigger Name:`AccountAddressTrigger`

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

Bulk Apex Triggers

Create a Bulk Apex trigger

Create a bulkified Apex trigger that adds a follow-up task to an opportunity if its stage is Closed Won. Fire the Apex trigger after inserting or updating an opportunity.

- Create an Apex trigger:
  - Name: *`ClosedOpportunityTrigger`*
  - Object: **Opportunity**
  - Events: after insert and after update
  - Condition: Stage is `Closed Won`
  - Operation: Create a task:
    - `Subject: Follow Up Test Task`
    - `WhatId`: the opportunity ID (associates the task with the opportunity)
  - Bulkify the Apex trigger so that it can insert or update 200 or more opportunities

CODE SOLUTION:

Apex Trigger Name: *`ClosedOpportunityTrigger`*

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {


List<Task> taskList= new List <Task>();

  For( Opportunity opp: Trigger.new){
    if(Trigger.isInsert){
      if(Opp.StageName=='Closed Won'){
        tasklist.add(new Task(Subject ='Follow Up Test Task', WhatId = opp.Id));

      }
    }
    if(Trigger.isupdate){
      if(Opp.StageName =='Closed Won'
      && Opp.StageName != Trigger.oldMap.get(opp.Id).StageName){

        taskList.add(new Task(Subject = 'Follow up test task ',  WhatId = opp.Id));
      }
    }

}
if(taskList.size()>0){
  insert tasklist;
}

}
```

Apex Testing

Get Started with Apex Unit Tests

Create a Unit Test for a Simple Apex Class

Create and install a simple Apex class to test if a date is within a proper range, and if not, returns a date that occurs at the end of the month within the range. You'll copy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

- Create an Apex class:
  - Name: `VerifyDate`
  - Code: **Copy from GitHub**
- Place the unit tests in a separate test class:
  - Name: `TestVerifyDate`
  - Goal: 100% code coverage
- Run your test class at least once

CODE SOLUTION:

Apex Class Name:`VerifyDate(From GitHub)`

```
public classVerifyDate{

        //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
        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
        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;
        }

}
```

Apex Class Name:`TestVerifyDate`

```
@isTest
public class TestVerifyDate {
  @isTest static void test1(){
    Date d = VerifyDate.CheckDates(Date.parse('01/01/2022'),Date.parse('01/03/2022'));
    System.assertEquals(Date.parse('01/03/2022'), d);
  }

  @isTest static void test2(){
    Date d = VerifyDate.CheckDates(Date.parse('01/01/2022'),Date.parse('03/03/2022'));
    System.assertEquals(Date.parse('01/31/2022'), d);
  }
}
```

Test Apex Triggers

GET READY

You'll be completing this unit in your own hands-on org. Click **Launch** to get started, or click the name of your org to choose a different one.

If you use Trailhead in a language other than English, make sure that your hands-on org is set to the same language as the challenge instructions. Otherwise you may run into issues passing this unit. Want to find out more about using hands-on orgs on Trailhead? Check out Trailhead Playground Management.

YOUR CHALLENGE

Create a Unit Test for a Simple Apex Trigger

Create and install a simple Apex trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'. You'll copy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

- Create an Apex trigger on the Contact object
  - Name: `RestrictContactByName`
  - Code: **Copy from GitHub**
- Place the unit tests in a separate test class
  - Name: `TestRestrictContactByName`
  - Goal: 100% test coverage
- Run your test class at least once

CODE SOLUTION:

Apex Class Name: `RestrictContactByName(From GitHub)`

```
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 not allowed for DML');
                    }

            }
    }
```

Apex Class Name: `TestRestrictContactByName`

```
@isTest
public class TestRestrictContactByName {

   @isTest static void Test_insertupdateContact(){
       Contact cnt = new Contact();
       cnt.LastName = 'INVALIDNAME';
        Test.startTest();

         Database.SaveResult result = Database.insert(cnt , false);

         Test.stopTest();

         System.assert(!result.isSuccess());

         System.assert (result.getErrors().size() > 0);

         System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());

}

 }
```

Create Test Data for Apex Tests

YOUR CHALLENGE

Create a Contact Test Factory

Create an Apex class that returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name. Do not insert the generated contact records into the database.

NOTE: For the purposes of verifying this hands-on challenge, don't specify the @isTest annotation for either the class or the method, even though it's usually required.

- Create an Apex class in the `public` scope
  - Name: `RandomContactFactory` (without the @isTest annotation)
- Use a Public Static Method to consistently generate contacts with unique first names based on the iterated number in the format Test 1, Test 2 and so on.
  - Method Name: `generateRandomContacts` (without the @isTest annotation)
  - Parameter 1: An integer that controls the number of contacts being generated with unique first names
  - Parameter 2: A string containing the last name of the contacts
  - Return Type: `List < Contact >`

CODE SOLUTION:

Apex Class Name: RandomContactFactory

```
public class RandomContactFactory {

   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;
   }
}
```

Asynchronous Apex

Use Future Methods

YOUR CHALLENGE

Create an Apex class that uses the @future annotation to update Account records.

Create an Apex class with a future method that accepts a List of Account IDs and updates a custom field on the Account object with the number of contacts associated to the Account. Write unit tests that achieve 100% code coverage for the class. Every hands-on challenge in this module asks you to create a test class.

- Create a field on the Account object:
  - Label: `Number Of Contacts`
  - Name: `Number_Of_Contacts`
  - Type: **Number**
  - This field will hold the total number of Contacts for the Account
- Create an Apex class:
  - Name: `AccountProcessor`
  - Method name: `countContacts`
  - The method must accept a List of Account IDs
  - The method must use the @future annotation
  - The method counts the number of Contact records associated to each Account ID passed to the method and updates the 'Number_Of_Contacts__c' field with this value
- Create an Apex test class:
  - Name: `AccountProcessorTest`
  - The unit tests must cover all lines of code included in the **AccountProcessor** class, resulting in 100% code coverage.
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

CODE SOLUTION:

Apex Class Name : AccountProcessor

```
public class AccountProcessor {

@future
   public static void countContacts(List<Id> accountIds){
         List<Account> accountsToUpdate = new List<Account>();
         List<Account> accounts = [Select Id , Name , (Select Id from Contacts) from
Account Where Id in : accountIds];
    For(Account acc:accounts){
        List<Contact> contactList = acc.Contacts;
        acc.Number_Of_Contacts__c = contactList.size();
        accountsToUpdate.add(acc);
    }
                    update accountsToUpdate;
   }
}
```

Apex Class Name : AccountProcessorTest

```
@isTest
private class AccountProcessorTest {
   @isTest static void testAcctProMultipleAcctSingleContact(){
        List<Account> accts = new List<Account>();
        for(Integer i=0;i<300;i++){
           accts.add(new Account(Name='Test Account: '+i));
        }
        insert accts;
        List<Contact> cts = new List<Contact>();
        List<Id> accountIds = new List<Id>();
        for(Account acc: accts){
           cts.add(new Contact(FirstName=acc.Name, LastName= 'TestContact',
AccountId=acc.Id));
           accountIds.add(acc.Id);
        }
        insert cts;
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
        List<Account> accountsList = [SELECT Id, Number_Of_Contacts__c FROM
Account WHERE Id IN :accountids];
        for(Account a: accountsList){
          System.assertEquals(1, a.Number_Of_Contacts__c);
        }
    }
}
```

Use Batch Apex

YOUR CHALLENGE

Create an Apex class that uses Batch Apex to update Lead records.

Create an Apex class that implements the Database.Batchable interface to update all Lead records in the org with a specific LeadSource.

- Create an Apex class:
    - Name: `LeadProcessor`
    - Interface: `Database.Batchable`
    - Use a QueryLocator in the start method to collect all Lead records in the org
    - The execute method must update all Lead records in the org with the LeadSource value of `Dreamforce`
- Create an Apex test class:
    - Name: `LeadProcessorTest`
    - In the test class, insert 200 Lead records, execute the LeadProcessor Batch class and test that all Lead records were updated correctly
    - The unit tests must cover all lines of code included in the **LeadProcessor** class, resulting in 100% code coverage
- Before verifying this challenge, run your test class at least once using the

Developer Console Run All feature

CODE SOLUTION:

Apex Class Name : `LeadProcessor`

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

   global Integer count = 0;

   global Database.QueryLocator start(Database.BatchableContext bc){
      return Database.getQueryLocator('Select Id , LeadSource From Lead');
   }

   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) ;
   }
}
```

Apex Class Name : `LeadProcessorTest`

```
@isTest
public class LeadProcessorTest {

   @isTest
   public static void testit(){
      List<lead> L_list = new List<lead>();

      for(Integer i = 0 ; i < 200 ; i++){
                              Lead L = new lead();
                              L.lastName = 'name' + i;
                              L.Company = 'Cpmpany';
                              L.Status = 'Radnom Status' ;
                              L_list.add(L);
      }
      insert L_list;
      Test.startTest();
      LeadProcessor lp = new LeadProcessor();
      Id batchId = Database.executeBatch(lp);
      Test.stopTest();
   }
}
```

Control Processes with Queueable Apex

YOUR CHALLENGE

Create a Queueable Apex class that inserts Contacts for Accounts.

Create a Queueable Apex class that inserts the same Contact for each Account for a specific state.

- Create an Apex class:
  - Name: `AddPrimaryContact`
  - Interface: `Queueable`
  - Create a constructor for the class that accepts as its first argument a Contact sObject and a second argument as a string for the State abbreviation
  - 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.

- Create an Apex test class:
    - Name: `AddPrimaryContactTest`
    - In the test class, insert 50 Account records for `BillingState NY` and 50 Account records for `BillingState CA`
    - Create an instance of the `AddPrimaryContact` class, enqueue the job, and assert that a Contact record was inserted for each of the 50 Accounts with the `BillingState` of `CA`
    - The unit tests must cover all lines of code included in the **AddPrimaryContact** class, resulting in 100% code coverage
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

CODE SOLUTION:

Apex Class Name : `AddPrimaryContact`

```
global 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 ;
        }
    }
}
```

Apex Class Name : `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')]);
    }  }
```

Schedule Jobs Using the Apex Scheduler

YOUR CHALLENGE

Create an Apex class that uses Scheduled Apex to update Lead records.

Create an Apex class that implements the Schedulable interface to update Lead records with a specific LeadSource. (This is very similar to what you did for Batch Apex.)

- Create an Apex class:
    - Name: `DailyLeadProcessor`
    - Interface: `Schedulable`
    - The execute method must find the first 200 Lead records with a blank LeadSource field and update them with the LeadSource value of `Dreamforce`
- Create an Apex test class:
    - Name: `DailyLeadProcessorTest`
    - In the test class, insert 200 Lead records, schedule the DailyLeadProcessor class to run and test that all Lead records were updated correctly
    - The unit tests must cover all lines of code included in the **DailyLeadProcessor** class, resulting in 100% code coverage.
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

CODE SOLUTION:

Apex Class Name : `DailyLeadProcessor`

```
public without sharing class DailyLeadProcessor implements Schedulable{

   public void execute(SchedulableContext ctx){

      List<lead> leads = [Select Id ,LeadSource from Lead where LeadSource = NULL
limit 200];
```

```
        for(Lead l:leads){
            l.LeadSource = 'Dreamforce';

        }
        update leads;
     }
}
```

Apex Class Name : `DailyLeadProcessorTest`

```
@isTest
public class DailyLeadProcessorTest {

    private static String CRON_EXP = '0 0 0 ? * * *';

    @isTest
    private static void testSchedulableClass(){
        List<Lead> leads = new List<lead>();
        for(Integer i = 0 ; i < 500 ; i++){
            if(i < 250){
                leads.add(new Lead(LastName = 'Connock' , Company = 'Salesforce'));
            }
            else{
                leads.add(new Lead(LastName = 'Connock' , Company = 'Salesforce' ,
LeadSource = 'Other'));
            }
        }
        insert leads;
        Test.startTest();
        String jobId = System.schedule('Process Leads' ,CRON_EXP , new
DailyLeadProcessor());
        Test.stopTest();

        List<Lead> checkleads = [Select Id , Leadsource from Lead where LeadSource =
'Dreamforce'];
                        System.assertEquals(200 , checkleads.size() , 'ERROR: At least 1 record
updated correctly');  } }
```

Apex Integration Services

Apex REST Callouts

YOUR CHALLENGE

Create an Apex class that calls a REST endpoint and write a test class.

Create an Apex class that calls a REST endpoint to return the name of an animal, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

- Create an Apex class:
    - Name: `AnimalLocator`
    - Method name: `getAnimalNameById`
    - The method must accept an Integer and return a String.
    - The method must call https://th-apex-http-callout.herokuapp.com/animals/<id>, replacing <id> with the ID passed into the method
    - The method returns the value of the **name** property (i.e., the animal name)
- Create a test class:
    - Name: `AnimalLocatorTest`
    - The test class uses a mock class called `AnimalLocatorMock` to mock the callout response
- Create unit tests:
    - Unit tests must cover all lines of code included in the **AnimalLocator** class, resulting in 100% code coverage
- Run your test class at least once (via **Run All** tests the Developer Console) before attempting to verify this challenge

CODE SOLUTION:

Apex Class Name : `AnimalLocator`

```
public class AnimalLocator {

   public static String getAnimalNameById(Integer i){
        Http http =new Http();
       HttpRequest request = new HttpRequest();

       request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i);
       request.setMethod('GET');
       HttpResponse response = http.send(request);

       Map<String ,Object> result =
(Map<String,Object>)JSON.deserializeUntyped(response.getBody());
       Map<String,Object> animal = (Map<String,Object>)result.get('animal');
                   System.debug('name: '+string.valueOf(animal.get('name')));
       return string.valueOf(animal.get('name'));
   }
}
```

Apex Class Name : `AnimalLocatorTest`

```
@isTest
public class AnimalLocatorTest {

   @isTest
   static void animalLocatorTest1(){
       Test.setMock(HttpCalloutMock.class , new AnimalLocatorMock());
       String actual = AnimalLocator.getAnimalNameById(1);
       String expected = 'moose';
       System.assertEquals(actual , expected);
   }

}
```

Apex SOAP Callouts

YOUR CHALLENGE

Generate an Apex class using WSDL2Apex and write a test class.

Generate an Apex class using WSDL2Apex for a SOAP web service, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

- Generate a class using this using this WSDL file:
    - Name: `ParkService` (Tip: After you click the **Parse WSDL** button, change the Apex class name from **parksServices** to `ParkService`)
    - Class must be in public scope
- Create a class:
    - Name: `ParkLocator`
    - Class must have a **country** method that uses the **ParkService** class
    - Method must return an array of available park names for a particular country passed to the web service (such as Germany, India, Japan, and United States)
- Create a test class:
    - Name: `ParkLocatorTest`
    - Test class uses a mock class called `ParkServiceMock` to mock the callout response
- Create unit tests:
    - Unit tests must cover all lines of code included in the **ParkLocator** class, resulting in 100% code coverage.
- Run your test class at least once (via **Run All** tests the Developer Console) before attempting to verify this challenge.

CODE SOLUTION:

Apex Class Name : `ParkLocator`

```
public class ParkLocator {

   public static List<String> country(String country){
      ParkService.parksImplPort prkSvc = new ParkService.ParksImplPort();
      return prkSvc.byCountry(country);
   }

}
```

Apex Class Name : `ParkLocatorTest`

```
@isTest
public class ParkLocatorTest {

   @isTest static void testCallout(){
      Test.setMock(WebServiceMock.class, new parkServiceMock());
      String Country = 'United States';
      List<String> expectedParks = new List<String>{'Yosemite' ,'Sequoia', 'Crater Lake'};
         System.assertEquals(expectedParks, ParkLocator.country(country));
   }
}
```

Apex Web Services

YOUR CHALLENGE

Create an Apex REST service that returns an account and its contacts.

Create an Apex REST class that is accessible at /Accounts/<Account_ID>/contacts. The service will return the account's ID and name plus the ID and name of all contacts associated with the account. Write unit tests that achieve 100% code coverage for the class and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

- Create an Apex class
  - Name: `AccountManager`
  - Class must have a method called `getAccount`
  - Method must be annotated with **@HttpGet** and return an **Account** object
  - Method must return the **ID** and **Name** for the requested record and all associated contacts with their **ID** and **Name**
- Create unit tests
  - Unit tests must be in a separate Apex class called `AccountManagerTest`
  - Unit tests must cover all lines of code included in the **AccountManager** class, resulting in 100% code coverage
- Run your test class at least once (via **Run All** tests the Developer Console) before attempting to verify this challenge

CODE SOLUTION:

Apex Class Name : `AccountManager`

```
@RestResource(urlMapping = '/Accounts/*/contacts')
global with sharing class AccountManager {

   @HttpGet
   global static Account getAccount(){
      RestRequest request = RestContext.request;
      string accountId = request.requestURI.substringBetween('Accounts/','/contacts');
      Account result = [SELECT Id, Name, (Select Id, Name from Contacts) from Account
where Id=:accountId Limit 1];
      return result;
   }
}
```

Apex Class Name : `AccountManagerTest`

```
@IsTest
private class AccountManagerTest {
   @isTest static void testGetContactsByAccountId(){
      Id recordId = createTestRecord();
      RestRequest request = new RestRequest();
      request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'
                           + recordId+'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
      Account thisAccount = AccountManager.getAccount();
      System.assert(thisAccount != null);
      System.assertEquals('Test record', thisAccount.Name);
   }

   static Id createTestRecord(){
      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

2. Automate record creation

CODE SOLUTION:

Apex Class Name : MaintenanceRequestHelper

```
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 Maintenance'){
                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,
Equipment__c, Equipment__r.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,
                    Equipment__c =cc.Equipment__c,
                    Origin = 'Web',
                    Date_Reported__c = Date.Today()

                );

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

                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;       }}}
```

Apex Trigger Name : `MaitenanceRequest`

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

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

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

  }

}
```

3. Synchronize Salesforce data with an external system

CODE SOLUTION:

Apex Class Name: WarehouseCalloutService

```
 public with sharing class WarehouseCalloutService implements Queueable {
  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

  //class that makes a REST callout to an external warehouse system to get a list of
equipment that needs to be updated.
  //The callout’s JSON response returns the equipment records that you upsert in
Salesforce.
```

```
   @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());

         //class maps the following fields: replacement part (always true), cost, current
inventory, lifespan, maintenance cycle, and warehouse SKU
         //warehouse SKU will be external ID for identifying which equipment records to
update within Salesforce
         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 = (Integer) mapJson.get('cost');
            myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
            myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
            myEq.ProductCode = (String) mapJson.get('_id');
            warehouseEq.add(myEq);
         }

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

```
        }
      }
   }

   public static void execute (QueueableContext context){
      runWarehouseEquipmentSync();
   }

}
```

4. Schedule synchronization

CODE SOLUTION:

Apex Class Name: WarehouseSyncSchedule

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

5. Test automation logic

CODE SOLUTION:

Apex Class Name: MaintenanceRequestHelperTest

```
@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 Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    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,
                           Equipment__c=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, Equipment__c, 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.Equipment__c, 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);
    }
}
```

Apex Class Name: MaintenanceRequestHelper

```
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 Maintenance'){
                    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,
Equipment__c, Equipment__r.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,
                Equipment__c =cc.Equipment__c,
                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;
    }
  }
}
```

Apex Trigger Name: MaintenanceRequest

```
trigger MaintenanceRequest on Case (before update, after update) {
if(Trigger.isUpdate && Trigger.isAfter){
MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);    } }
```

6.Test callout logic

CODE SOLUTION:

Apex Class Name: WarehouseCalloutService

```
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 with the warehouse one');
            System.debug(warehouseEq);
          }

        }
      }
    }
```

Apex Class Name: WarehouseCalloutServiceTest

```
@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]);
   }
}
```

Apex Class Name: WarehouseCalloutServiceMock

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

      System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
      System.assertEquals('GET', request.getMethod());

      // Create a fake response
      HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity"
:5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
      response.setStatusCode(200);
      return response;
   }
```

}

7. Test scheduling logic

CODE SOLUTION:

Apex Class Name: WarehouseSyncSchedule

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

       WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

Apex Class Name: WarehouseSyncScheduleTest

```
@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 Schedule to Test',
scheduleTime, new 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 ');



   }
}
```