**Apex Triggers:**

1) Get Started with Apex Triggers:

**Challenge :**

**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 for 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;  
 }  
 }  
}

# 2) Bulk Apex Triggers :

**Challenge :**

**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 for ClosedOpportunityTrigger :**

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 Task', WhatId = opp.Id));  
 }  
 }  
 if(tasklist.size()>0){  
 insert tasklist;  
 }  
}

# Apex Testing :

# 1) Get Started with Apex Unit Tests :

**Challenge :**

**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](https://github.com/developerforce/trailhead-code-samples/blob/master/VerifyDate.cls)**
* Place the unit tests in a separate test class:
  + Name: TestVerifyDate
  + Goal: 100% code coverage
* Run your test class at least once

**Code for VerifyDate :**

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;  
 }  
  
}

**Code for TestVerifyDate :**

@isTest  
private class TestVerifyDate {  
   
 @isTest static void Test\_CheckDates\_case1(){  
 Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));  
 System.assertEquals(date.parse('01/05/2020'),D);  
 }  
   
 @isTest static void Test\_CheckDates\_case2(){  
 Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));  
 System.assertEquals(date.parse('01/31/2020'),D);  
 }  
   
 @isTest static void Test\_DateWithin30Days\_case1(){  
 Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('12/30/2019'));  
 System.assertEquals(false,flag);   
 }  
   
 @isTest static void Test\_DateWithin30Days\_case2(){  
 Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('02/02/2020'));  
 System.assertEquals(false,flag);   
 }  
   
 @isTest static void Test\_DateWithin30Days\_case3(){  
 Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('01/15/2020'));  
 System.assertEquals(true,flag);   
 }  
   
 @isTest static void Test\_SetEndOfMonthDate(){  
 Date returndate=VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));  
 }  
  
}

# 2) Test Apex Triggers :

**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](https://github.com/developerforce/trailhead-code-samples/blob/master/RestrictContactByName.cls)**
* Place the unit tests in a separate test class
  + Name: TestRestrictContactByName
  + Goal: 100% test coverage
* Run your test class at least once

**Code for RestrictContactByName :**

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');  
 }  
  
 }  
  
  
  
}

Code for 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());  
 }  
  
}

# 3) Create Test Data for Apex Tests :

**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 for 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 :

# 1) Use Future Methods :

**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 for 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;   
 }  
}

**Code for AccountProcessorTest :**

@IsTest  
private class AccountProcessorTest {  
 @IsTest  
 private static void testCountContacts(){  
 Account newAccount=new Account(Name='Test Account');  
 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();  
 }  
  
}

# 2)Use Batch Apex :

**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 for 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);  
 }  
  
}

**Code for 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='Company';  
 L.status='Random Status';  
 L\_list.add(L);  
 }  
 insert L\_list;  
   
 Test.startTest();  
 LeadProcessor lp=new LeadProcessor();  
 Id batchId=Database.executeBatch(lp);  
 Test.stopTest();  
 }  
  
}

# 3) Control Processes with Queueable Apex :

**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 for AddPrimaryContact :**

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;  
 }  
   
 }  
  
}

**Code for 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')]);  
   
 }  
}

# 4) Schedule Jobs Using the Apex Scheduler :

**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 for DailyLeadProcessor :**

global class DailyLeadProcessor implements Schedulable {  
 global void execute(SchedulableContext ctx) {  
 List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];  
   
 if(!lList.isEmpty()) {  
 for(Lead l: lList) {  
 l.LeadSource = 'Dreamforce';  
 }  
 update lList;  
 }  
 }  
  
}

**Code for DailyLeadProcessorTest :**

@isTest  
public class DailyLeadProcessorTest {  
//Seconds Minutes Hours Day\_of\_month Month Day\_of\_week optional\_year  
 public static String CRON\_EXP = '0 0 0 2 4 ? 2023';  
   
 static testmethod void testScheduledJob(){  
 List<Lead> leads = new List<Lead>();  
   
 for(Integer i = 0; i < 200; i++){  
 Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i, Status = 'Open - Not Contacted');  
 leads.add(lead);  
 }  
   
 insert leads;  
   
 Test.startTest();  
 // Schedule the test job  
 String jobId = System.schedule('Update LeadSource to DreamForce', CRON\_EXP, new DailyLeadProcessor());  
   
 // Stopping the test will run the job synchronously  
 Test.stopTest();  
 }  
  
}

# Apex Integration Services :

# 1) Apex REST Callouts :

**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 for AnimalLocator :**

public class AnimalLocator{   
 public static String getAnimalNameById(Integer x){  
  
Http http = new Http();  
  
HttpRequest req = new HttpRequest();  
  
req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);  
  
req.setMethod('GET');   
Map<String, Object> animal= new Map<String, Object>();  
  
HttpResponse res = http.send(req);  
  
if (res.getStatusCode() == 200) {   
 Map<String, Object> results = (Map<String, Object>)JSON.deserializeUntyped(res.getBody());   
 animal = (Map<String, Object>) results.get('animal');  
}  
return (String)animal.get('name');  
}  
}

**Code for AnimalLocatorTest :**

@isTest  
  
private class AnimalLocatorTest{  
  
@isTest static void AnimalLocatorMock1() {   
 Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());  
 string result =AnimalLocator.getAnimalNameById(3);  
String expectedResult = 'chicken';  
System.assertEquals(result,expectedResult);  
  
}  
  
}

**Code for AnimalLocatorMock :**

@isTest  
  
global class AnimalLocatorMock implements HttpCalloutMock {  
  
// Implement this interface method  
  
global HTTPResponse respond(HTTPRequest request) {  
  
// Create a fake response   
HttpResponse response = new HttpResponse();  
response.setHeader('Content-Type','application/json');  
  
response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken","mighty moose"]}');  
  
response.setStatusCode(200);  
  
return response;  
  
}  
}

# 2) Apex SOAP Callouts :

**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](https://th-apex-soap-service.herokuapp.com/assets/parks.xml):
  + 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 for ParkLocator :**

public class ParkLocator {   
 public static string[] country(string theCountry)  
{  
ParkService.ParksImplPort parkSvc = new ParkService.ParksImplPort(); // remove space  
return parkSvc.byCountry(theCountry);  
}  
}

**Code for ParkLocatorTest :**

@isTest  
  
private class ParkLocatorTest {  
  
@isTest static void testCallout() {  
  
Test.setMock(WebServiceMock.class, new ParkServiceMock());  
  
String country = 'United States';  
  
List<String> result =ParkLocator.country(country);  
  
List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};  
  
System.assertEquals(parks, result);  
}  
}

**Code for ParkService :**

|  |
| --- |
| //Generated by wsdl2apex  public class ParkService {  public class byCountryResponse {  public String[] return\_x;  private String[] return\_x\_type\_info = new String[]{'return','http://parks.services/',null,'0','-1','false'};  private String[] apex\_schema\_type\_info = new String[]{'http://parks.services/','false','false'};  private String[] field\_order\_type\_info = new String[]{'return\_x'};  }  public class byCountry {  public String arg0;  private String[] arg0\_type\_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};  private String[] apex\_schema\_type\_info = new String[]{'http://parks.services/','false','false'};  private String[] field\_order\_type\_info = new String[]{'arg0'};  }  public class ParksImplPort {  public String endpoint\_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';  public Map<String,String> inputHttpHeaders\_x;  public Map<String,String> outputHttpHeaders\_x;  public String clientCertName\_x;  public String clientCert\_x;  public String clientCertPasswd\_x;  public Integer timeout\_x;  private String[] ns\_map\_type\_info = new String[]{'http://parks.services/', 'ParkService'};  public String[] byCountry(String arg0) {  ParkService.byCountry request\_x = new ParkService.byCountry();  request\_x.arg0 = arg0;  ParkService.byCountryResponse response\_x;  Map<String, ParkService.byCountryResponse> response\_map\_x = new |

Map<String, ParkService.byCountryResponse>();  
 response\_map\_x.put('response\_x', response\_x);  
 WebServiceCallout.invoke(  
 this,  
 request\_x,  
 response\_map\_x,  
 new String[]{endpoint\_x,

'',  
 'http://parks.services/',  
 'byCountry',  
 'http://parks.services/',  
 'byCountryResponse',  
 'ParkService.byCountryResponse'}  
 );

response\_x = response\_map\_x.get('response\_x');  
 return response\_x.return\_x;  
 }  
 }  
}

Code for ParkServiceMock :

@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) {  
  
ParkService.byCountryResponse response\_x= new ParkService.byCountryResponse();  
  
response\_x.return\_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};  
response.put('response\_x', response\_x);  
}  
}

# 3) Apex Web Services :

**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 for AccountManager :**

@RestResource(urlMapping='/Accounts/\*/contacts')  
global class AccountManager {  
 @HttpGet  
 global static Account getAccount() {  
 RestRequest req = RestContext.request;  
 String accId = req.requestURI.substringBetween('Accounts/', '/contacts');  
 Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)   
 FROM Account WHERE Id = :accId];  
 return acc;  
 }  
}

**Code for AccountManagerTest :**

@isTest  
private class AccountManagerTest {  
  
 private static testMethod void getAccountTest1() {  
 Id recordId = createTestRecord();  
 // Set up a test request  
 RestRequest request = new RestRequest();  
 request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts' ;  
 request.httpMethod = 'GET';  
 RestContext.request = request;  
 // Call the method to test  
 Account thisAccount = AccountManager.getAccount();  
 // Verify results  
 System.assert(thisAccount != null);  
 System.assertEquals('Test record', thisAccount.Name);  
  
 }  
  
 // Helper method  
 static Id createTestRecord() {  
 // Create test record  
 Account TestAcc = new Account(  
 Name='Test record');  
 insert TestAcc;  
 Contact TestCon= new Contact(  
 LastName='Test',   
 AccountId = TestAcc.id);  
 return TestAcc.Id;  
 }   
}

## Apex Specialist :

**Code for CreateDefaultData :**

public with sharing class CreateDefaultData{  
 Static Final String TYPE\_ROUTINE\_MAINTENANCE = 'Routine Maintenance';  
 //gets value from custom metadata How\_We\_Roll\_Settings\_\_mdt to know if Default data was created  
 @AuraEnabled  
 public static Boolean isDataCreated() {  
 How\_We\_Roll\_Settings\_\_c customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();  
 return customSetting.Is\_Data\_Created\_\_c;  
 }   
   
 //creates Default Data for How We Roll application  
 @AuraEnabled  
 public static void createDefaultData(){  
 List<Vehicle\_\_c> vehicles = createVehicles();  
 List<Product2> equipment = createEquipment();  
 List<Case> maintenanceRequest = createMaintenanceRequest(vehicles);  
 List<Equipment\_Maintenance\_Item\_\_c> joinRecords = createJoinRecords(equipment, maintenanceRequest);  
  
 updateCustomSetting(true);  
 }  
  
  
 public static void updateCustomSetting(Boolean isDataCreated){  
 How\_We\_Roll\_Settings\_\_c customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();  
 customSetting.Is\_Data\_Created\_\_c = isDataCreated;  
 upsert customSetting;  
 }

public static List<Vehicle\_\_c> createVehicles(){  
 List<Vehicle\_\_c> vehicles = new List<Vehicle\_\_c>();  
 vehicles.add(new Vehicle\_\_c(Name = 'Toy Hauler RV', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Toy Hauler RV'));  
 vehicles.add(new Vehicle\_\_c(Name = 'Travel Trailer RV', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 2, Bedrooms\_\_c = 2, Model\_\_c = 'Travel Trailer RV'));  
 vehicles.add(new Vehicle\_\_c(Name = 'Teardrop Camper', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Teardrop Camper'));  
 vehicles.add(new Vehicle\_\_c(Name = 'Pop-Up Camper', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Pop-Up Camper'));  
 insert vehicles;  
 return vehicles;  
 }  
  
 public static List<Product2> createEquipment(){  
 List<Product2> equipments = new List<Product2>();  
 equipments.add(new Product2(Warehouse\_SKU\_\_c = '55d66226726b611100aaf741',name = 'Generator 1000 kW', Replacement\_Part\_\_c = true,Cost\_\_c = 100 ,Maintenance\_Cycle\_\_c = 100));  
 equipments.add(new Product2(name = 'Fuse 20B',Replacement\_Part\_\_c = true,Cost\_\_c = 1000, Maintenance\_Cycle\_\_c = 30 ));  
 equipments.add(new Product2(name = 'Breaker 13C',Replacement\_Part\_\_c = true,Cost\_\_c = 100 , Maintenance\_Cycle\_\_c = 15));  
 equipments.add(new Product2(name = 'UPS 20 VA',Replacement\_Part\_\_c = true,Cost\_\_c = 200 , Maintenance\_Cycle\_\_c = 60));  
 insert equipments;  
 return equipments;  
   
 }  
  
 public static List<Case> createMaintenanceRequest(List<Vehicle\_\_c> vehicles){  
 List<Case> maintenanceRequests = new List<Case>();  
 maintenanceRequests.add(new Case(Vehicle\_\_c = vehicles.get(1).Id, Type = TYPE\_ROUTINE\_MAINTENANCE, Date\_Reported\_\_c = Date.today()));  
 maintenanceRequests.add(new Case(Vehicle\_\_c = vehicles.get(2).Id, Type = TYPE\_ROUTINE\_MAINTENANCE, Date\_Reported\_\_c = Date.today()));  
 insert maintenanceRequests;  
 return maintenanceRequests;  
 }  
  
 public static List<Equipment\_Maintenance\_Item\_\_c> createJoinRecords(List<Product2> equipment, List<Case> maintenanceRequest){   
 List<Equipment\_Maintenance\_Item\_\_c> joinRecords = new List<Equipment\_Maintenance\_Item\_\_c>();  
 joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(0).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));  
 joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(1).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));  
 joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(2).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));  
 joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(0).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));  
 joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(1).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));  
 joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(2).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));  
 insert joinRecords;  
 return joinRecords;  
  
 }  
}

**Code for CreateDefaultDataTest :**

@isTest  
private class CreateDefaultDataTest {  
 @isTest  
 static void createData\_test(){  
 Test.startTest();  
 CreateDefaultData.createDefaultData();  
 List<Vehicle\_\_c> vehicles = [SELECT Id FROM Vehicle\_\_c];  
 List<Product2> equipment = [SELECT Id FROM Product2];  
 List<Case> maintenanceRequest = [SELECT Id FROM Case];  
 List<Equipment\_Maintenance\_Item\_\_c> joinRecords = [SELECT Id FROM Equipment\_Maintenance\_Item\_\_c];  
  
 System.assertEquals(4, vehicles.size(), 'There should have been 4 vehicles created');  
 System.assertEquals(4, equipment.size(), 'There should have been 4 equipment created');  
 System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2 maintenance request created');  
 System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment maintenance items created');  
  
 }

@isTest  
 static void updateCustomSetting\_test(){  
 How\_We\_Roll\_Settings\_\_c customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();  
 customSetting.Is\_Data\_Created\_\_c = false;  
 upsert customSetting;  
  
 System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom setting How\_We\_Roll\_Settings\_\_c.Is\_Data\_Created\_\_c should be false');  
  
 customSetting.Is\_Data\_Created\_\_c = true;  
 upsert customSetting;  
   
 System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting How\_We\_Roll\_Settings\_\_c.Is\_Data\_Created\_\_c should be true');  
  
 }  
}

**Code for 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;  
 }  
 }  
}

**Code for 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);  
 }  
}

**Code for WarehouseCalloutService :**

public with sharing class WarehouseCalloutService implements Queueable {  
 private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';  
   
 //Write a 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(){  
 System.debug('go into runWarehouseEquipmentSync');  
 Http http = new Http();  
 HttpRequest request = new HttpRequest();  
   
 request.setEndpoint(WAREHOUSE\_URL);  
 request.setMethod('GET');  
 HttpResponse response = http.send(request);  
   
 List<Product2> product2List = new List<Product2>();  
 System.debug(response.getStatusCode());  
 if (response.getStatusCode() == 200){  
 List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());  
 System.debug(response.getBody());

//class maps the following fields:   
 //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce  
 for (Object jR : jsonResponse){  
 Map<String,Object> mapJson = (Map<String,Object>)jR;  
 Product2 product2 = new Product2();  
 //replacement part (always true),  
 product2.Replacement\_Part\_\_c = (Boolean) mapJson.get('replacement');  
 //cost  
 product2.Cost\_\_c = (Integer) mapJson.get('cost');  
 //current inventory  
 product2.Current\_Inventory\_\_c = (Double) mapJson.get('quantity');  
 //lifespan  
 product2.Lifespan\_Months\_\_c = (Integer) mapJson.get('lifespan');  
 //maintenance cycle  
 product2.Maintenance\_Cycle\_\_c = (Integer) mapJson.get('maintenanceperiod');  
 //warehouse SKU   
 product2.Warehouse\_SKU\_\_c = (String) mapJson.get('sku');  
   
 product2.Name = (String) mapJson.get('name');  
 product2.ProductCode = (String) mapJson.get('\_id');  
 product2List.add(product2);  
 }  
   
 if (product2List.size() > 0){  
 upsert product2List;  
 System.debug('Your equipment was synced with the warehouse one');  
 }  
 }  
 }

public static void execute (QueueableContext context){  
 System.debug('start runWarehouseEquipmentSync');  
 runWarehouseEquipmentSync();  
 System.debug('end runWarehouseEquipmentSync');  
 }  
   
}

**Code for WarehouseCalloutServiceMock :**

@isTest  
global class WarehouseCalloutServiceMock implements HttpCalloutMock {  
 // implement http mock callout  
 global static HttpResponse respond(HttpRequest request) {  
   
 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"},{"\_id":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');  
 response.setStatusCode(200);  
   
 return response;  
 }  
}

**Code for WarehouseCalloutServiceMock :**

@isTest  
global class WarehouseCalloutServiceMock implements HttpCalloutMock {  
 // implement http mock callout  
 global static HttpResponse respond(HttpRequest request) {  
   
 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"},{"\_id":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"\_id":"55d66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse 20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');  
 response.setStatusCode(200);  
   
 return response;  
 }  
}

**Code for WarehouseCalloutServiceTest :**

@IsTest  
private class WarehouseCalloutServiceTest {  
 // implement your mock callout test here  
 @isTest  
 static void testWarehouseCallout() {  
 test.startTest();  
 test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());  
 WarehouseCalloutService.execute(null);  
 test.stopTest();  
   
 List<Product2> product2List = new List<Product2>();  
 product2List = [SELECT ProductCode FROM Product2];  
   
 System.assertEquals(3, product2List.size());  
 System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);  
 System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);  
 System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);  
 }  
}

**Code for WarehouseSyncSchedule :**

global class WarehouseSyncSchedule implements Schedulable {  
 global void execute(SchedulableContext ctx) {  
   
 WarehouseCalloutService.runWarehouseEquipmentSync();  
 }  
}

**Code for 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 ');  
   
   
 }  
}

**Code for MaintenanceRequest trigger :**

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

|  |
| --- |
|  |