Apex trigger

1. Get Started with Apex Triggers:

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

```
trigger AccountAddressTrigger on Account (before insert,before update) {

    for(Account a:Trigger.New){
        if(a.Match_Billing_Address__c== True){
            a.ShippingPostalCode = a.BillingPostalCode;
        }
    }
}
```

2.Bulk Apex Trigger:

- 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

```
trigger ClosedOpportunityTrigger on Opportunity (after insert , after update) {
    List<Task> tasklist = new List<Task>();
```

```
    for(Opportunity o:Trigger.New){
        if(o.StageName == 'Closed Won'){
            tasklist.add(new Task(Subject='Follow Up Test Task',WhatId = o.Id));
        }
    }
    if (tasklist.size()>0){
        insert tasklist;
    }
}
```

Apex Testing

1.Get Started with Apex Unit Tests:

- 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 from GItHub:

```
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
 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; }
```

```
22  }
23
24  //method to return the end of the month of a given date
25  private static Date SetEndOfMonthDate(Date date1) {
26    Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
27    Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
28    return lastDay;
29  }
30
31 }
```

Code for Test class

```
1  @isTest
2  public class TestVerifyDate
3  {
4      static testMethod void testMethod1()
5      {
6          Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
7          Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
8      }
9  }
```

2.Test Apex Triggers:

- 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 from GitHub:

```
1  trigger RestrictContactByName on Contact (before insert, before update) {
2
3      //check contacts prior to insert or update for invalid data
4      For (Contact c : Trigger.New) {
5              if(c.LastName == 'INVALIDNAME') {       //invalidname is invalid
6                      c.AddError('The Last Name "'+c.LastName+'" is not allowed for

7              }
8
```

```
    }


}
```

Code for test class:

```
@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.getErrors1().size() >0);
System.assert('The Last Name 'INVALIDNAME' is not allowed for DML',
result.getErrors()[0].getMessage());
}
}
```

3.Create Test Data for Apex Tests:

- 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 >

```
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;
    }
}
```

3.Asynchronous Apex

2.Use Future Methods:

- 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

AccountProcessor class code

```
public class AccountProcessor {

   @future
   public static void countContacts(List<Id> accountId_lst) {

      Map<Id,Integer> account_cno = new Map<Id,Integer>();
      List<account> account_lst_all = new List<account>([select id, (select id from contacts)
from account]);
      for(account a:account_lst_all) {
         account_cno.put(a.id,a.contacts.size()); //populate the map

      }

      List<account> account_lst = new List<account>(); // list of account that we will upsert

```

```
    for(Id accountId : accountId_lst) {
        if(account_cno.containsKey(accountId)) {
            account acc = new account();
            acc.Id = accountId;
            acc.Number_of_Contacts__c = account_cno.get(accountId);
            account_lst.add(acc);
        }

    }
    upsert account_lst;
}

}
```

Use future methods test code:

```
@isTest
public class AccountProcessorTest {

    @isTest
    public static void testFunc() {
        account acc = new account();
        acc.name = 'MATW INC';
        insert acc;

        contact con = new contact();
        con.lastname = 'Mann1';
        con.AccountId = acc.Id;
        insert con;
        contact con1 = new contact();
        con1.lastname = 'Mann2';
        con1.AccountId = acc.Id;
        insert con1;


        List<Id> acc_list = new List<Id>();
        acc_list.add(acc.Id);
        Test.startTest();
                        AccountProcessor.countContacts(acc_list);
        Test.stopTest();
        List<account> acc1 = new List<account>([select Number_of_Contacts__c
```

```
    from account where id = :acc.id]);
26        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
27    }
28
29 }
```

3.Use Batch Apex:

- 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 class:

```
1  global class LeadProcessor implements Database.Batchable<sObject> {
2     global Integer count = 0;
3
4     global Database.QueryLocator start (Database.BatchableContext bc) {
5        return Database.getQueryLocator('Select Id, LeadSource from lead');
6     }
7
8     global void execute (Database.BatchableContext bc,List<Lead> l_lst) {
9        List<lead> l_lst_new = new List<lead>();
10       for(lead l : l_lst) {
11          l.leadsource = 'Dreamforce';
12          l_lst_new.add(l);
13          count+=1;
14       }
15       update l_lst_new;
16    }
```

```
17
18    global void finish (Database.BatchableContext bc) {
19        system.debug('count = '+count);
20    }
21 }
```

Code for LeadProcessorTest class:

```
1  @isTest
2  public class LeadProcessorTest {
3
4      @isTest
5      public static void testit() {
6          List<lead> l_lst = new List<lead>();
7          for (Integer i = 0; i<200; i++) {
8              Lead l = new lead();
9              l.LastName = 'name'+i;
10             l.company = 'company';
11             l.Status =  'somestatus';
12             l_lst.add(l);
13         }
14         insert l_lst;
15
16         test.startTest();
17
18         Leadprocessor lp = new Leadprocessor();
19         Id batchId = Database.executeBatch(lp);
20         Test.stopTest();
21
22     }
23
24 }
```

4.Control Processes with Queueable Apex:

- 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 class:

```
public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;

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

    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, BillingState from account where account.BillingState = :this.state limit 200]);
        List<contact> c_lst = new List<contact>();
        for(account a: acc_lst) {
            contact c = new contact();
            c = this.c.clone(false, false, false, false);
            c.AccountId = a.Id;
            c_lst.add(c);
        }
        insert c_lst;
    }
```

```
22
23 }
```

Code for AddPrimaryContactTest class:

```
1  @IsTest
2  public class AddPrimaryContactTest {
3
4      @IsTest
5      public static void testing() {
6          List<account> acc_lst = new List<account>();
7          for (Integer i=0; i<50;i++) {
8              account a = new account(name=string.valueOf(i),billingstate='NY');
9              system.debug('account a = '+a);
10             acc_lst.add(a);
11         }
12         for (Integer i=0; i<50;i++) {
13             account a = new account(name=string.valueOf(50+i),billingstate='CA');
14             system.debug('account a = '+a);
15             acc_lst.add(a);
16         }
17         insert acc_lst;
18         Test.startTest();
19         contact c = new contact(lastname='alex');
20         AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
21         system.debug('apc = '+apc);
22         System.enqueueJob(apc);
23         Test.stopTest();
24         List<contact> c_lst = new List<contact>([select id from contact]);
25         Integer size = c_lst.size();
26         system.assertEquals(50, size);
27     }
28
29 }
```

5.Schedule Jobs Using the Apex Scheduler:

- 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 class:

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();

            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }

            update newLeads;
        }
    }
}
```

Code for DailyLeadProcessorTest class:

```
@isTest
private class DailyLeadProcessorTest{
    //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
    public static String CRON_EXP = '0 0 0 2 6 ? 2022';

```

```
6     static testmethod void testScheduledJob(){
7         List<Lead> leads = new List<Lead>();
8 
9         for(Integer i = 0; i < 200; i++){
10            Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company =
    'Test Company ' + i, Status = 'Open - Not Contacted');
11            leads.add(lead);
12        }
13 
14        insert leads;
15 
16        Test.startTest();
17        // Schedule the test job
18        String jobId = System.schedule('Update LeadSource to DreamForce',
    CRON_EXP, new DailyLeadProcessor());
19 
20        // Stopping the test will run the job synchronously
21        Test.stopTest();
22    }
23 }
```

4.Apex Integration Services

2.Apex REST Callouts:

- 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 class:

```
public class AnimalLocator {
                    public class cls_animal {
                     public Integer id;
                     public String name;
                     public String eats;
                     public String says;
                    }
public class JSONOutput{
                    public cls_animal animal;

                    //public JSONOutput parse(String json){
                    //return (JSONOutput) System.JSON.deserialize(json,
JSONOutput.class);
                    //}
}

  public static String getAnimalNameById (Integer id) {
     Http http = new Http();
     HttpRequest request = new HttpRequest();
     request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' +
id);
     //request.setHeader('id', String.valueof(id)); -- cannot be used in this
challenge :)
     request.setMethod('GET');
     HttpResponse response = http.send(request);
     system.debug('response: ' + response.getBody());
     //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
     jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
     //Object results = (Object) map_results.get('animal');
                     system.debug('results= ' + results.animal.name);
     return(results.animal.name);
  }

}
```

Code for AnimalLocator Mock class:

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPresponse respond(HTTPrequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
        //-- directly output the JSON, instead of creating a logic
        //response.setHeader('key, value)
        //Integer id = Integer.valueof(request.getHeader('id'));
        //Integer id = 1;
        //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
        //system.debug('animal return value: ' + lst_body[id]);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken

        return response;
    }

}
```

Code for AnimalLocatorTest class:

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        //Httpresponse response = AnimalLocator.getAnimalNameById(1);
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }

}
```

3.Apex SOAP Callouts:

- 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 for ParkService class:

```
1  //Generated by wsdl2apex
2
3  public class ParkService {
4      public class byCountryResponse {
5          public String[] return_x;
6          private String[] return_x_type_info = new
   String[]{'return','http://parks.services/',null,'0','-1','false'};
7          private String[] apex_schema_type_info = new
   String[]{'http://parks.services/','false','false'};
8          private String[] field_order_type_info = new String[]{'return_x'};
9      }
10     public class byCountry {
11         public String arg0;
12         private String[] arg0_type_info = new
   String[]{'arg0','http://parks.services/',null,'0','1','false'};
13         private String[] apex_schema_type_info = new
   String[]{'http://parks.services/','false','false'};
14         private String[] field_order_type_info = new String[]{'arg0'};
15     }
16     public class ParksImplPort {
17         public String endpoint_x = 'https://th-apex-soap-

18         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 class:

```
@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();
    List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
    response_x.return_x = lstOfDummyParks;

    response.put('response_x', response_x);
  }
}
```

Code for ParkLocator class:

```
public class ParkLocator {
  public static String[] country(String country){
    ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
    String[] parksname = parks.byCountry(country);
    return parksname;
  }
}
```

Code for ParkLocatorTest class:

```
@isTest
private class ParkLocatorTest{
  @isTest
  static void testParkLocator() {
    Test.setMock(WebServiceMock.class, new ParkServiceMock());
    String[] arrayOfParks = ParkLocator.country('India');

    System.assertEquals('Park1', arrayOfParks[0]);
  }
}
```

4.Apex Web Services:

- 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 class

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

```
@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId
    +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;

        // Call the method to test
        Account  acc = AccountManager.getAccount();
```

```
14
15      // Verify results
16      System.assert(acc != null);
17  }
18
19  private static Id getTestAccountId(){
20      Account acc = new Account(Name = 'TestAcc2');
21      Insert acc;
22
23      Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
24      Insert con;
25
26      return acc.Id;
27  }
28 }
```

Apex Specialist Superbadge solution

Set up Development Org

1. Create a new Trailhead Playground for this superbadge.
2. Install this unlocked package (package ID: 04t6g000008av9iAAA).
3. Add picklist values Repair and Routine Maintenance to the Type field on the Case object.
4. Update the Case page layout assignment to use the Case (HowWeRoll) Layout for your profile.
5. Rename the tab/label for the Case tab to Maintenance Request.
6. Update the Product page layout assignment to use the Product (HowWeRoll) Layout for your profile.
7. Rename the tab/label for the Product object to Equipment.
8. Click on App Launcher and search Create Default Data then Click Create Data to

Challenge 1

Automated Record Creation

1.Go to the App Launcher -> Search How We Roll Maintenance -> click on Maintenance Requests -> click on first case -> click Details -> change the type Repair to Routine Maintenance -> select Origin = Phone -> Vehicle = select Teardrop Camper , save it.
2.Feed -> Close Case = save it..
3.Go to the Object Manager -> Maintenance Request ->Field & Relationships ->New ->Lookup Relationship -> next -> select Equipment ->next -> Field Label = Equipment ->next->next->next -> save it .
4.Now go to the developer console use below code.

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 Maintenance'){
                    validIds.add(c.Id);

```

```
9
10              }
11          }
12      }
13
14      if (!validIds.isEmpty()){
15          List<Case> newCases = new List<Case>();
16          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)
17                                                  FROM Case WHERE Id IN :validIds]);
18          Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
19          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];
20
21      for (AggregateResult ar : results){
22          maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
23      }
24
25          for(Case cc : closedCasesM.values()){
26              Case nc = new Case (
27                  ParentId = cc.Id,
28              Status = 'New',
29                  Subject = 'Routine Maintenance',
30                  Type = 'Routine Maintenance',
31                  Vehicle__c = cc.Vehicle__c,
32                  Equipment__c =cc.Equipment__c,
33                  Origin = 'Web',
34                  Date_Reported__c = Date.Today()
35
36              );
37
38              If (maintenanceCycles.containskey(cc.Id)){
39                  nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
40              } else {
41                  nc.Date_Due__c = Date.today().addDays((Integer) cc.Equipment__r.maintenance_Cycle__c);
42              }
43
44              newCases.add(nc);
45          }
46
47          insert newCases;
48
49          List<Equipment_Maintenance_Item__c> clonedWPs = new List<Equipment_Maintenance_Item__c>();
50          for (Case nc : newCases){
51              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;
      }
    }
}
```

MaitenanceRequest.apxt

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

1. After saving the code go back the How We Roll Maintenance ,
2. click on Maintenance Requests -> click on 2nd case -> click Details -> change the type Repair to Routine Maintenance -> select Origin = Phone -> Vehicle = select Teardrop Camper , save it.
3. Feed -> Close Case = save it..

Now check challenge.

Challenge 2

Synchronize Salesforce data with an external system

- Setup -> Search in quick find box -> click Remote Site Settings -> Name = Warehouse URL , Remote Site URL = https://th-superbadge-apex.herokuapp.com , make sure active is selected.
- Go to the developer console use below code .

WarehouseCalloutService.apxc :-

```apex
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();
    }

}
```

After saving the code open execute anonymous window (CTRl+E) and run this method ,

System.enqueueJob(new WarehouseCalloutService());

Now check Challenge.

Challenge 3

Schedule synchronization using Apex code

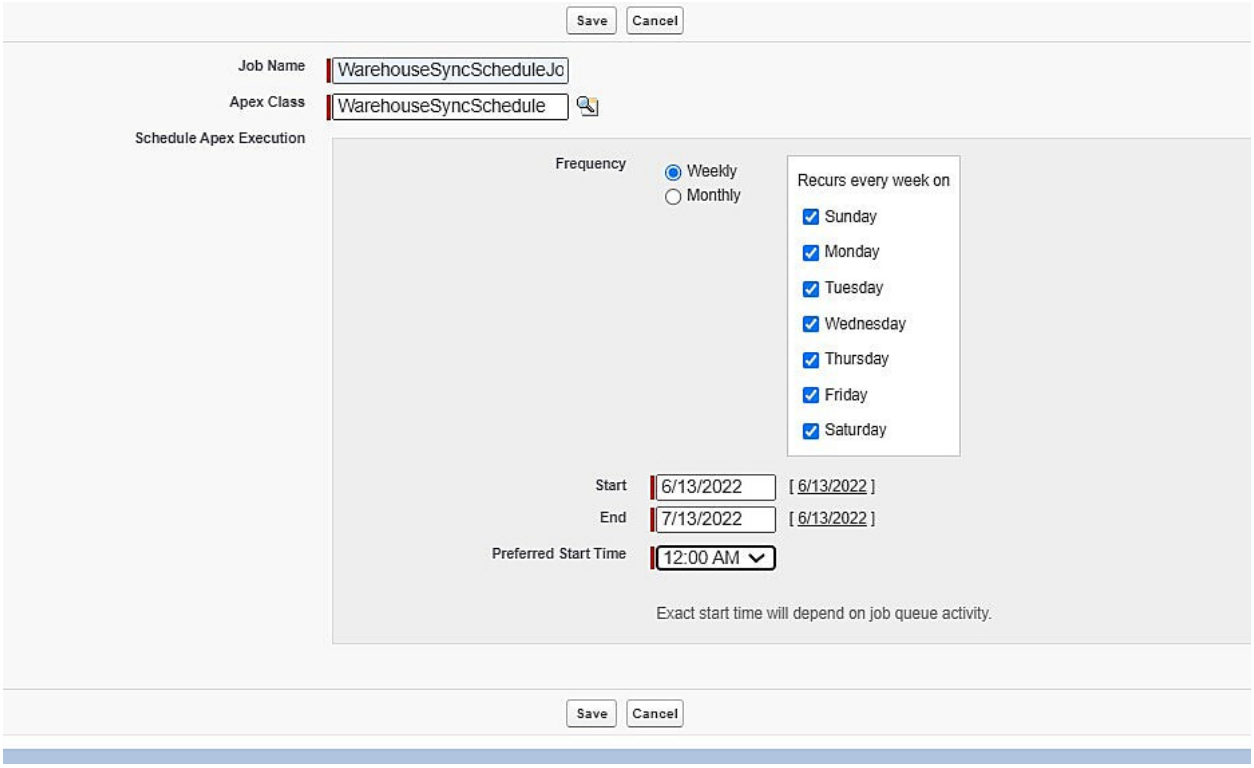
- Go to the developer console use below code ,

WarehouseSyncShedule.apxc :-

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

Save it , after that...

- Go to setup -> Seacrh in Quick find box -> Apex Classes -> click Schedule Apex and Jb Name = WarehouseSyncScheduleJob , Apex Class = WarehouseSyncSchedule as it is below shown in the image ,



Now check challenge.

Challenge 4

Test automation logic

- Go to the developer console use below code ,

MaintenanceRequestHelperTest.apxc :-

```
@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;

```

```
52          case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
53          insert somethingToUpdate;
54
55          Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,somethingToUpdate.id);
56          insert workP;
57
58          test.startTest();
59          somethingToUpdate.status = CLOSED;
60          update somethingToUpdate;
61          test.stopTest();
62
63          Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c, Date_Due__c
64                  from case
65                  where status =:STATUS_NEW];
66
67          Equipment_Maintenance_Item__c workPart = [select id
68                                      from Equipment_Maintenance_Item__c
69                                      where Maintenance_Request__c =:newReq.Id];
70
71          system.assert(workPart != null);
72          system.assert(newReq.Subject != null);
73          system.assertEquals(newReq.Type, REQUEST_TYPE);
74          SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
75          SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
76          SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
77      }
78
79      @istest
80      private static void testMaintenanceRequestNegative(){
81          Vehicle__C vehicle = createVehicle();
82          insert vehicle;
83          id vehicleId = vehicle.Id;
84
85          product2 equipment = createEq();
86          insert equipment;
87          id equipmentId = equipment.Id;
88
89          case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
90          insert emptyReq;
91
92          Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
93          insert workP;
94
95          test.startTest();
96          emptyReq.Status = WORKING;
97          update emptyReq;
98          test.stopTest();
99
100         list<case> allRequest = [select id
101                                  from case];
102
103         Equipment_Maintenance_Item__c workPart = [select id
104                                      from Equipment_Maintenance_Item__c
105                                      where Maintenance_Request__c = :emptyReq.Id];
106
107         system.assert(workPart != null);
108         system.assert(allRequest.size() == 1);
109     }
110
111     @istest
112     private static void testMaintenanceRequestBulk(){
113         list<Vehicle__C> vehicleList = new list<Vehicle__C>();
114         list<Product2> equipmentList = new list<Product2>();
115         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);
    }
}
```

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 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;
    }
  }
}
```

MaintenanceRequest.apxt :-

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

run all

Now check challenge.

Challenge 5

Test callout logic

- Go to the developer console use below code ,

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

        }
    }
}
```

WarehouseCalloutServiceTest.apxc :-

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

WarehouseCalloutServiceMock.apxc :-

```
@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

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

run all

Now check challenge.

Challenge 6

Test scheduling logic

- Go to the developer console use below code ,

WarehouseSyncSchedule.apxc :-

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

WarehouseSyncScheduleTest.apxc :-

```
@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 ');


    }
}
```

run all

Now check challenge.