Apex Triggers

1. Get started with Apex Triggers

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:

- 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 trigger

Create a bulkified Apex Trigger that add 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
- Operation: Create a task:
- Subject: Follow Up Test Task
- WhatId: the opportunity ID (associates the task with the opportunity)
- Bulkify the Apex trigger s that it can insert or update 200 or more opportunities

Code for ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (before insert) {
     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

Create and install a simple Apex class 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 copythe code for the clas from GitGub. 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 spearate 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

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 form GitHub
- Place the unit tests in a separate test class
- Name: TestRestrictContactByName
- Goal: 100% test coverage
- Run your test class atleast 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');
   }
  }
}
```

3.Create Test Data for Apex Tests

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

- Create an Apex class in the public scope
- Name: RandomContactFactory
- Use a Public Static Method to consistenly generate contacts with unique first names
- Parameter 1: An integer that controls the number of contacts being generated
- Return Type: List

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

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 acheive 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:
- Lable: 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 units must cover all lines of code inclded in the AccountProcessor class
- resulting in 100% code coverage.
- Befor verifying the challenge
- run your test class at least once using the Developer Console Rull 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

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

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

- Create an Apex class:
- Name: LeadProcessor
- Interface: Database.Batchable
- Use a QueryLocator in the satrt method to collect all Lead records in the org.
- The execute method 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 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 LeadProcesor

```
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 = 'Randam 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

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 a Apex class:
- Name: AddPrimaryContact
- Interface: Queueable
- Createa constructor for thet 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 Schedular

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 filed 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 atleast once using the Developer Console Run All Feauture

Code for DailyLeadProcessor

```
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

```
@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';

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

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

To pass this challenge create an Apex class a Rest endpoint to return the name of a animal, write unit tests that achieve 100% code coverage for the class using a mock repsonse, and run your Apex tests

- The Apex class must be create AnimalLocator, have a getAnimalNameById method that accepts an integer and returns a string.
- The 'getAnimalNameById' method must can https://th-apex-http-callout.herokupp.com/animals/id, using the ID passes in the method. he method returns the vallue of the 'name' property (i.e.,the animal name)
- Createa test class named AnimalLocatorTest that uses a mock class called AnimalLocatorMock to mock the coallout response.
- The unit test must cover all the code include in the AnimalLocator class, resulting in the 100% code

coverage.

- Run your test class at least once(via Run All the tests the Developer Console) before attemptinng to verify this challenge

Code for AnimalLocator

```
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 AnimalLocatorMock

```
@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 food","says":"cluck
cluck"}}');
    return response;
  }
}
```

Code for AnimalLocatorTest

```
@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);
  }
}
```

2.Apex SOAP Callouts

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

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

- Generate a class using this WSDL file.
- Name: ParkService(Tip: After) you click the Parse WSDL button, change the Apex class Name from parkService to ParkService)
- Class must be in public scope.
- Create a class:
- Name ParkLocator
- Class must have a country method that uses the ParkService class.
- Class must return an array of available park names for a particular country passed to web service(such as Germany, India, Japan, and United States)
- Create a test class:
- Name: ParkLocatorTest

- Testclass users a mock class called ParServiceMock to mock callout response.
- Create unit tests.
- Unit tests must cover all lines inluded in the ParkLocator class, resulting in 100% code coverage.
- Run your test class at least once(via Run All the tests the Developer Console) before attempting to verify the challenge.

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

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

Code for ParkLocator

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

Code for ParkLocatorTest

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

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

3.Apex Web Services

Create an Apex Rest service that returns an accounts and its contacts.

Create an Apex Rest class that is accrssible at/Accounts/contacts. The service will return the accounts ID's 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.

- 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 Test
- Unit tests must be in a separate apex class clled AccountManagerTest
- Unit tests must cover all lines of code(via Run all tests the Developer Console) before attempting to verify the

challenge.

Code for 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 ];
      return result;
   }
}
```

Code for AccountManagerTest

```
@Istest
public 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 SuperBadge

1.Automate Record Creation

- Install the unlocked package and configure the development org.
- Use the included package content to automatically create a Routine Maintenance request of type Repair or Routine Maintenance is updated to Closed. Follow the specifications and naming conventions outlined in the business requirements.

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 MaintenanceRequest

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

2.Synchronize Salesforce data with an external system

- Implement an Apex Class(WarehouseCalloutService) that implements the queuable interface and makes a callout to the external service used for warehouse investory management. This service receives updated values in the external system and updates the realted records in Salesforce. Before checking this section, enqueue the job atleast once to confirm that it's working as expected.

Code for 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() {
    //ToDo: complete this method to make the callout (using @future) to the
    //      REST endpoint and update equipment on hand.


    HttpResponse response = getResponse();
    if(response.getStatusCode() == 200)
    {
      List<Product2> results = getProductList(response); //get list of products from Http callout response

      if(results.size() >0)
      upsert results Warehouse_SKU__c; //Upsert the products in your org based on the external ID SKU
    }

  }
    //Get the product list from the external link
    public static List<Product2> getProductList(HttpResponse response)
    {

      List<Object> externalProducts = (List<Object>) JSON.deserializeUntyped(response.getBody()); //desrialize the json response
      List<Product2> newProducts = new List<Product2>();

      for(Object p : externalProducts)
      {
        Map<String, Object> productMap = (Map<String, Object>) p;
        Product2 pr = new Product2();
        //Map the fields in the response to the appropriate fields in the Equipment object
        pr.Replacement_Part__c = (Boolean)productMap.get('replacement');
        pr.Cost__c = (Integer)productMap.get('cost');
```

```
            pr.Current_Inventory__c = (Integer)productMap.get('quantity');
            pr.Lifespan_Months__c = (Integer)productMap.get('lifespan') ;
            pr.Maintenance_Cycle__c = (Integer)productMap.get('maintenanceperiod');
            pr.Warehouse_SKU__c = (String)productMap.get('sku');
            pr.ProductCode = (String)productMap.get('_id');
            pr.Name = (String)productMap.get('name');


            newProducts.add(pr);
        }

        return newProducts;

    }

    // Send Http GET request and receive Http response

  public static HttpResponse getResponse() {

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

    return response;

  }
  }
```

3.Schedule Sychronization

- Build scheduling logic that executes your callout and runs your code daily. The name of the schedulable class should be WarehouseSyncSchedule, and the scheduled job should be named WarehouseSyncScheduleJob.

Code for WarehouseSyncSchedule

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

4.Test automation logic

- Build tests for all cases(positive, negative, and bulk)specified in the business requirements by using a class named MaintenanceRequestHelperTest. You must have 100% test coverage to pass this section and assert values to prove that your logic is working as expected. Choose Run All Tests in the Developer Console at least once before attempting to submit this section. Be patient as it may take 10-20 seconds to process the challenge check.

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);
    }
}
```

5.Test callout logic

- Build tests for your callout using the included class for the callout mock(WarehouseCalloutServiceMock) and callout test class(WarehouseCalloutServiceTest) in the package. You must have 100% test coverage to pass this challenge and assert values to prove that your logic is working as expected.

Code for 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;
    }
}
```

Code for WarehouseCalloutServiceTest

```
@isTest
private class WarehouseCalloutServiceTest {
  // implement your mock callout test here
    @isTest
    static void WarehouseEquipmentSync(){
        Test.startTest();
        // Set mock callout class
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        // This causes a fake response to be sent from the class that implements HttpCalloutMock.
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
```

6.Test Scheduling logic

Build unit tests for the class WarehouseSyncSchedule in class named WarehouseSyncScheduleTest. You must have 100% test coverage to pass this schallenge and assert values to prove that your logic is working as expected.

Code for WarehouseSyncScheduleTest

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime, new
WarehouseSyncSchedule());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}
```