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:

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

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

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

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

Code 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;
}
```

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 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 for RestrictContactByNametrigger 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 testContactTrigger() {
      Test.StartTest();
      Contact c = new Contact(LastName = 'INVALIDNAME');
      Database.SaveResult result = Database.insert(c, false);
      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size() > 0);
      Test.StopTest();
   }

}
```

3.Create Test Data for Apex Tests

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 numOfContacts, String
lName) {
      List<Contact> cList = new List<Contact>();
      for(Integer i=0; i<numOfContacts; i++) {
         Contact c = new Contact(Firstname = 'Test' + i, Lastname = lName);
         cList.add(c);
      }
      return cList;
   }
}
```

Asynchronous Apex

Use Future Methods 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 AccountProcessorpublic 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 forAccountProcessorTest

```
@IsTest
public 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();

    }
}
```

Use Batch Apex

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_lst) {
        List<lead> l_lst_new = new List<lead>();
        for(lead l : l_lst) {
            l.leadsource = 'Dreamforce';
```

```
            l_lst_new.add(l);
            count+=1;
        }
        update l_lst_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();
    }
}
```

Control Processes with Queueable Apex

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 AddPrimaryContactpublic 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 = 'Jhon', 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')]);
    }
}
```

5.Schedule Jobs Using the Apex Scheduler

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 sc){
      List<Lead> lstofLead = [Select ID From Lead Where LeadSource = NULL Limit 200];

      List<Lead> lstOfUpdatedLead = new List<Lead>();
      if(!lstOfLead.isEmpty()){
      for(Lead ld : lstOfLead){
         ld.LeadSource = 'Dreamforce';
         lstOfUpdatedLead.add(ld);
      }

      update lstOfUpdatedLead;
      }

   }
}
```

Code for DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {
```

```
    @testSetup
    static void setup(){
        List<Lead> lstOfLead = new List<Lead>();
        for(Integer i = 1; i<=200; i++){
            Lead ld= new Lead(Company = 'Comp' + i,LastName = 'LN' + i, Status ='Working -
Contacted');
            lstOfLead.add(ld);
        }

        Insert lstOfLead;
    }

    static testmethod void testDailyLeadProcessorScheduledJob(){
        String sch = '0 5 12 * * ?';
        Test.startTest();
        String jobId = System.Schedule('ScheduledApexText', sch, new
DailyLeadProcessor());

        List<Lead> lstOfLead = [Select Id From Lead Where LeadSource = NULL Limit
200];
        system.assertEquals(200,lstOfLead.size());

        Test.stopTest();
    }

    }
```

Lightning Web Components Basics

Add Styles and Data to a Lightning Web Component

Create a Lightning app page that uses the wire service to display the current user's name.

Prework: You need files created in the previous unit to complete this challenge. If you haven’t already completed the activities in the previous unit, do that now.

● Create a Lightning app page:

○ Label: Your Bike Selection

○ Developer Name: Your_Bike_Selection

● Add the current user's name to the app container:○ Edit selector.js
○ Edit selector.html

selector.js

```
import { LightningElement, wire,track } from 'lwc';
import {
    getRecord
} from 'lightning/uiRecordApi';
import Id from '@salesforce/user/Id';
import NAME_FIELD from '@salesforce/schema/User.Name';
import EMAIL_FIELD from '@salesforce/schema/User.Email';
export default class Selector extends LightningElement {
    @track selectedProductId;
    @track error ;
    @track email ;
    @track name;
    @wire(getRecord, {
        recordId: Id,
        fields: [NAME_FIELD, EMAIL_FIELD]
    }) wireuser({
        error,
        data
    }) {
        if (error) {
           this.error = error ;
        } else if (data) {
            this.email
 = data.fields.Email.value;
            this.name
 = data.fields.Name.value;
        }    }
    handleProductSelected(evt) {
        this.selectedProductId = evt.detail;
    }
    userId = Id;
}
```

selector.html

```
<template>
```

```
    <div class="wrapper">
    <header class="header">Available Bikes for {name}</header>
    <section class="content">
       <div class="columns">
       <main class="main" >
          <c-list onproductselected={handleProductSelected}></c-list>
       </main>
       <aside class="sidebar-second">
          <c-detail product-id={selectedProductId}></c-detail>
       </aside>
       </div>
    </section>
    </div>
</template>
```

Apex Integration Services

Apex REST Callouts

Create an Apex class that calls a REST endpoint to return the name of an animal, writeunit 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 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;
   }
}
```

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 );
    }
}
```

Apex SOAP Callouts

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

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

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

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]);
   }
}
```

Apex Web Services

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 with sharing class AccountManager {

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

Code for AccountManagerTest

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

```
        System.assertEquals('Test record', thisAccount.Name);
    }

    static Id createTestRecord(){
        Account accountTest = new Account(          Name ='Test record');
        insert accountTest;

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

        return accountTest.Id;
    }
}
```

Apex Specialist Super Badge

Automate record creation

code for MaintenanceRequest

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

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);                }
            }
```

```
    }
    //When an existing maintenance request of type Repair or Routine Maintenance is
closed,
    //create a new maintenance request for a future routine checkup.
    if (!validIds.isEmpty()){
      Map<Id,Case> closedCases = 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>();
      //calculate the maintenance request due dates by using the maintenance cycle
defined on the related equipment records.
      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'));
      }
      List<Case> newCases = new List<Case>();
      for(Case cc : closedCases.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 multiple pieces of equipment are used in the maintenance request,
        //define the due date by applying the shortest maintenance cycle to today's
date.
        If (maintenanceCycles.containskey(cc.Id)){
```

```
                nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
            } else {
                nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
            }
            newCases.add(nc);
        }
        insert newCases;
        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}
```

Synchronize Salesforce data with an external systemcode 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');
    }
}
```

schedule synchronization

code for WarehouseSyncSchedule

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

Test automation logic

code for MaintenanceRequestHelperTest

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }
    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
    }
    // createMaintenanceRequest
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
```

```
                    Vehicle__c=vehicleId);
        return cse;
    }
    // createEquipmentMaintenanceItem
    private static Equipment_Maintenance_Item__c
createEquipmentMaintenanceItem(idequipmentId,id requestId){
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
            Maintenance_Request__c = requestId);
        return equipmentMaintenanceItem;
    }
    @isTest
    private static void testPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
        Product2 equipment = createEquipment();
        insert equipment;
        id equipmentId = equipment.Id;
        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
        insert createdCase;
        Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
        insert equipmentMaintenanceItem;
        test.startTest();
        createdCase.status = 'Closed';
        update createdCase;
        test.stopTest();
        Case newCase = [Select id,
                        subject,
                        type,
                        Equipment__c,
                        Date_Reported__c,
                        Vehicle__c,                    Date_Due__c
                       from case
                       where status ='New'];
```

```
        Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newCase.Id];
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 2);
        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
        system.assertEquals(newCase.Type, 'Routine Maintenance');
        SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
    }
    @isTest
    private static void testNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
        product2 equipment = createEquipment();
        insert equipment;
        id equipmentId = equipment.Id;
        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
        insert createdCase;
        Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
        insert workP;
        test.startTest();       createdCase.Status = 'Working';
        update createdCase;
        test.stopTest();
        list<case> allCase = [select id from case];
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c = :createdCase.Id];
        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }
    @isTest
    private static void testBulk(){
```

```
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();
        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEquipment());
        }
        insert vehicleList;
        insert equipmentList;
        for(integer i = 0; i < 300; i++){
            caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
        }
        insert caseList;        for(integer i = 0; i < 300; i++){

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.
get(i).id, caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;
        test.startTest();
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();
        list<case> newCase = [select id
                              from case
                              where status ='New'];
        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in: oldCaseIds];
        system.assert(newCase.size() == 300);
        list<case> allCase = [select id from case];
```

```
        system.assert(allCase.size() == 600);
    }
}
```

Test callout logic

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":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

code forWarehouseCalloutServiceTest

```
@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);
    }
}
```

test scheduling logic

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();
    }
}
```