APEX TRIGGERS

1. Get Started with Apex Triggers

Create an Apex trigger that sets an account's Shipping Postal Code to match the BillingPostal 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 Match Billing Address:

```
trigger AccountAddressTrigger on Account (before insert,before update) {
for (Account account : trigger.new){
        if((account.Match_Billing_Address__c == true) && (account.BillingPostalCode != NULL)){
                                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 isClosed 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 theopportunity)
  - Bulkify the Apex trigger so that it can insert or update 200 or moreopportunities

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 ifnot, returns a date that occurs at the end of the month within the range.

- 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 {
        public static Date CheckDates(Date date1, Date date2) {
      //Otherwise use the end of the month
     if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return  SetEndOfMonthDate(date1);
                }
        }
private static Boolean DateWithin30Days(Date date1,Date date2) {
        if( date2 < date1) { return false; }
        Date date30Days = date1.addDays(30); //create a date 30 days awayfrom date1
                if( date2 >= date30Days ) { return false; }
                else { return true; }
```

```
            }
            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 anycontact with a last name of 'INVALIDNAME'.

- 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 RestrictContactByName:

```
trigger RestrictContactByName on Contact (before insert, before update) {For (Contact c :
          Trigger.New) {
                    if(c.LastName == 'INVALIDNAME') {
c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
                    }
```

```
  }
}
```

Code for TestRestrictContactByName:

```
@isTest
public class TestRestrictContactByName {

    @isTest static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';
        
        Test.startTest();
        Database.SaveResult result = Database.insert(cnt, false);
        Test.stopTest();
        
        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowedfor DML',
result.getErrors()[0].getMessage());
    }
}
```

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 @isTestannotation 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 firstnames based on the iterated number in the format Test 1, Test 2 and so on.
    - Method Name: `generateRandomContacts` (without the @isTestannotation)
    - Parameter 1: An integer that controls the number of contacts beinggenerated 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 num,String lastName){
        List<Contact> contactList = new List<Contact>();
        for(Integer i=1;i<=num;i++){
            Contact ct = new Contact(FirstName = 'Test '+i, LastName =lastName);
            contactList.add(ct);
        }
        return contactList;
    }
}
```

Asynchronous Apex

1. 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 associatedto 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 eachAccount 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.

Code for AccountProcessor:

```
public class AccountProcessor {
@future
   public static void countContacts(List<Id> accountIds){

    List<Account> accList = [Select Id,  Number_Of_Contacts__c, (Select Id from

      Contacts) from Account Where Id in :accountIds];

      For(Account acc:accList){
        acc.Number_Of_Contacts__c = acc.Contacts.size();
      }
      update  accList;

   }
}
```

Code for AccountProcessorTest:

```
@isTest
public class AccountProcessorTest {
    public static testmethod void testAccountProcessor(){
        Account a = new Account();
        a.Name = 'Test Account';
        insert a;

        Contact con = new Contact();
        con.FirstName = 'Uday Kiran';
        con.LastName = 'Sangepu';
        con.AccountId = a.Id;

        insert con;

        List<Id> accListId = new List<Id>();
        accListId.add(a.Id);

        Test.startTest();
        AccountProcessor.countContacts(accListId);
        Test.stopTest();

        Account acc = [Select Number_Of_Contacts__c from Account where Id =: a.Id];
        System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c),1);
    }
}
```

2. Use Batch Apex

Create an Apex class that implements the Database.Batchable interface to update allLead 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 theorg

    - The execute method must update all Lead records in the org with theLeadSource value of `Dreamforce`
- Create an Apex test class:
    - Name: `LeadProcessorTest`
    - In the test class, insert 200 Lead records, execute the LeadProcessorBatch 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 theDeveloper Console Run All feature

Code for LeadProcessor:

```
public class LeadProcessor implements
    Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator(
            'SELECT ID from Lead'
        );
    }
    public void execute(Database.BatchableContext bc, List<Lead> scope){
        // process each batch of records
        List<Lead> leads = new List<Lead>();
        for (Lead lead : scope) {
            lead.LeadSource = 'Dreamforce';
            leads.add(lead);
        }
        update leads;
    }
    public void finish(Database.BatchableContext bc){

    }
}
```

Code for LeadProcessorTest:

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 10 accounts
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i, Company = 'Test Co'));
        }
        insert leads;

    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor myLeads = new LeadProcessor();
        Id batchId = Database.executeBatch(myLeads);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);
    }
}
```

3. Control Processes with Queueable Apex

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

- Create an Apex class:
  - Name: `AddPrimaryContact`
  - Interface: `Queueable`
  - Create a constructor for the class that accepts as its first argument aContact sObject and a second argument as a string for the State abbreviation
  - The `execute` method must query for a maxi
  - mum 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 50Account 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 Accountswith 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 theDeveloper Console Run All feature

Code for AddPrimaryContact:

```
public class AddPrimaryContact implements Queueable
{
   private Contact c;
   private String state;
   public  AddPrimaryContact(Contact c, String state)
   {
      this.c = c;
      this.state = state;
   }
   public void execute(QueueableContext context)
   {
        List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts ) FRO
M ACCOUNT WHERE BillingState = :state LIMIT 200];
        List<Contact> lstContact = new List<Contact>();
        for (Account acc:ListAccount)
        {
              Contact cont = c.clone(false,false,false,false);
              cont.AccountId =  acc.id;
              lstContact.add( cont );
        }

        if(lstContact.size() >0 )
        {
            insert lstContact;
        }

    }

}
```

Code for AddPrimaryContactTest:

```
@isTest
public class AddPrimaryContactTest
{
    @isTest static void TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
            Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
            Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;

        Contact co = new Contact();
        co.FirstName='demo';
        co.LastName ='demo';
        insert co;
        String state = 'CA';

         AddPrimaryContact apc = new AddPrimaryContact(co, state);
         Test.startTest();
           System.enqueueJob(apc);
         Test.stopTest();
    }
 }
```

4. Schedule Jobs Using the Apex Scheduler

Create an Apex class that implements the Schedulable interface to update Lead recordswith 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 blankLeadSource 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 wereupdated 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 theDeveloper Console Run All feature

Code for DailyLeadProcessor:

```
global class DailyLeadProcessor implements Schedulable {

    global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null limit 200];
        list<lead> led = new list<lead>();
        if(!lList.isEmpty()) {
            for(Lead l: lList) {
                l.LeadSource = 'Dreamforce';
                led.add(l);
            }
            update led;
        }
    }
}
```

Code for DailyLeadProcessorTest:____

```
@isTest
public class DailyLeadProcessorTest{

    static testMethod void testMethod1()
    {
                Test.startTest();

        List<Lead> lstLead = new List<Lead>();
        for(Integer i=0 ;i <200;i++)
        {
            Lead led = new Lead();
            led.FirstName ='FirstName';
            led.LastName ='LastName'+i;
            led.Company ='demo'+i;
            lstLead.add(led);
        }

        insert lstLead;

        DailyLeadProcessor ab = new DailyLeadProcessor();
         String jobId = System.schedule('jobName','0 5 * * * ? ' ,ab) ;
        Test.stopTest();
    }
}
```

Lightning Web Components Basics

1. Add Styles and Data to a Lightning Web Component

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

Prework: You need files created in the previous unit to complete this challenge. If youhaven'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

Code for 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;

}
```

Code for 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

1. 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, andrun 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 passedinto 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 mockthe 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) beforeattempting to verify this challenge

Code for AnimalLocator:

```
public class AnimalLocator {
    public static String getAnimalNameById(Integer animalId) {
        String animalName;
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+animalId);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // If the request is successful, parse the JSON response.
        if(response.getStatusCode() == 200) {
          Map<String, Object> r = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());
          Map<String, Object> animal = (Map<String, Object>)r.get('animal');
          animalName = string.valueOf(animal.get('name'));
        }
        return animalName;
    }
}
```

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('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

Code for AnimalLocatorTest:

```
@isTest
private class AnimalLocatorTest {
@isTest
static void getAnimalNameByIdTest() {
    // Set mock callout class
    Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
    // This causes a fake response to be sent
    // from the class that implements HttpCalloutMock.
    String response = AnimalLocator.getAnimalNameById(1);
    // Verify that the response received contains fake values

    System.assertEquals('chicken', response);
}
}
```

2. Apex SOAP Callouts

Generate an Apex class using WSDL2Apex for a SOAP web service, write unit tests thatachieve 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, changethe 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, andUnited States)
- Create a test class:
  - Name: `ParkLocatorTest`

  - Test class uses a mock class called `ParkServiceMock` to mock thecallout 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) beforeattempting to verify this challenge.

Code for ParkLocator:

```
public class ParkLocator {
    public static List<String> country(String country) {
        ParkService.ParksImplPort parkservice =
            new parkService.ParksImplPort();
        return parkservice.byCountry(country);
    }
}
```

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) {
        // start - specify the response you want to send
        List<String> parks = new List<string>();
               parks.add('Yosemite');
               parks.add('Yellowstone');
               parks.add('Another Park');
        ParkService.byCountryResponse response_x =
            new ParkService.byCountryResponse();
        response_x.return_x = parks;
        // end
        response.put('response_x', response_x);
   }
}
```

Code for ParkLocatorTest:

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        // Call the method that invokes a callout
        String country = 'United States';
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>();
            parks.add('Yosemite');
            parks.add('Yellowstone');
            parks.add('Another Park');
        // Verify that a fake result is returned
        System.assertEquals(parks, result);
    }
}
```

3. Apex Web Services

Create an Apex REST class that is accessible at /Accounts/<Account_ID>/contacts. Theservice 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 allassociated 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) beforeattempting 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;
        // grab the caseId from the end of the URL
        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
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://yourInstance.salesforce.com/services/apexrest/Accounts/'+recordId+'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);
    }
    // Helper method
    static Id createTestRecord() {
        // Create test record
        Account 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

2. 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);


                }
            }
        }

        if (!validIds.isEmpty()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c, Equipme
nt__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_Cyc
le__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;
        }
    }
}
```

3. Synchronize Salesforce data with an external system:

<u>Code for WarehouseCalloutService:</u>

```
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);
            }

        }
    }
}
```

4. Schedule synchronization:

Code for WarehouseSyncSchedule:

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

5. Test automation logic:

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 = equipm
entId,
                                                                    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);
    }
}
```

6. Test callout logic:

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":"Gener
ator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Code for WarehouseCalloutServiceTest:

```
@isTest

private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

7. Test scheduling logic:

Code for WarehouseSyncScheduleTest :

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new WarehouseSy
ncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX system
s.
        // 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 ');


    }
}
```