Apex Specialist SuperBadge -1

1. Apex Triggers

1. Get Started with Apex Triggers

1. AccountAddressTrigger.apxt

```
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

1. ClosedOpportunityTrigger.apxt

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

    for(Opportunity Opp:Trigger.New){
        if(Trigger.isInsert || Trigger.isUpdate)
          if(opp.StageName=='Closed Won')
            taskList.add(new task(Subject='Follow Up Test Task',
```

```
                    WhatId=opp.Id));
    }
    if(taskList.size()>0)
        insert taskList;

}
```

2. Apex Testing

1. Get Started with Apex Unit Tests

1. VerifyDate.apxc

```
public class VerifyDate {

    //method to handle potential checks against two dates
    public static Date CheckDates(Date date1, Date date2) {
        //if date2 is within the next 30 days of date1, use date2.
Otherwise use the end of the month
        if(DateWithin30Days(date1,date2)) {
            return date2;
        } else {
            return SetEndOfMonthDate(date1);
        }
    }

    //method to check if date2 is within the next 30 days of date1
    private static Boolean DateWithin30Days(Date date1, Date date2) {
        //check for date2 being in the past
```

```
        if( date2 < date1) { return false; }
        //check that date2 is within (>=) 30 days of date1
        Date date30Days = date1.addDays(30); //create a date 30 days away
from date1
                if( date2 >= date30Days ) { return false; }
                else { return true; }
        }

        //method to return the end of the month of a given date
        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.TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate {
   @isTest static void test1() {
      Date d = verifyDate.CheckDates(Date.parse('01/01/2022'),
Date.parse('01/03/2022'));
      System.assertEquals(Date.parse('01/03/2022'), d);
   }
   @isTest static void test2() {
      Date d = VerifyDate.CheckDates(Date.parse('01/01/2022'),
Date.parse('03/03/2022'));
      System.assertEquals(Date.parse('01/31/2022'), d);
```

```
    }
}
```

2. Test Apex Triggers

1. RestrictContactByName.apxt

```
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');
                }
        }
}
```

2.TestRestrictContactByName.apxc

```
@isTest
public class TestRestrictContactByName {
    @isTest
    public static void testContact() {
        Contact ct = new Contact();
        ct.LastName = 'INVALIDNAME';
        Database.SaveResult res = Database.insert(ct, false);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed
for DML', res.getErrors()[0].getMessage());
    }
```

```
}
```

3. Create Test Data for Apex Tests

1.RandomContactFactory.apxc

```
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;
    }
}
```

3. Asynchronous Apex

2. Use Future Methods

1. AccountProcessor.apxc

```
public without sharing class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds) {
        List<Account> accounts = [SELECT Id, (SELECT Id FROM Contacts)
FROM Account WHERE Id IN :accountIds];
        for(Account acc : accounts) {
```

```
            acc.Number_of_Contacts__c = acc.Contacts.size();
        }
        update accounts;
    }
}
```

2.AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest {
    @isTest
    private static void countContactsTest() {
        // load Test Data
        List<Account> accounts = new List<Account>();
        for (Integer i = 0; i < 300; i++) {
            accounts.add(new Account(Name = 'Test Account' + i));
        }
        insert accounts;
        List<Contact> contacts = new List<Contact>();
        List<Id> accountIds = new List<Id>();
        for(Account acc: accounts) {
            contacts.add(new Contact(FirstName=acc.Name,
LastName='TestContact', AccountId=acc.Id));
            accountIds.add(acc.Id);
        }
        insert contacts;
        // Do the test
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
        // Check result
```

```
        List<Account> accs = [SELECT Id, Number_Of_Contacts__c FROM
Account];
        for(Account acc: accs) {
            System.assertEquals(1, acc.Number_Of_Contacts__c, 'ERROR: At
least 1 Account record with incorrect');
        }
    }
}
```

3. Use Batch Apex

1.LeadProcessor.apxc

```
public without sharing class LeadProcessor implements
Database.Batchable<sObject> {
    public Integer recordCount = 0;
    public Database.QueryLocator start(Database.BatchableContext dbc) {
        return Database.getQueryLocator([SELECT Id, Name FROM Lead]);
    }
    public void execute(Database.BatchableContext dbc, List<Lead> Leads) {
        for(Lead l : leads) {
            l.LeadSource = 'Dreamforce';
        }
        update leads;
        recordCount = recordCount + leads.size();
    }
    public void finish(Database.BatchableContext dbc) {
        System.debug('Total records processed' + recordCount);
    }
}
```

2. LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {

    @isTest
    private static void testBatchClass() {
        // Load test data
        List<Lead> leads = new List<Lead>();
        for (Integer i = 0; i < 200; i++) {
            leads.add(new Lead(LastName='Connock', Company='Salesforce'));
        }
        insert leads;
        // Perform the test
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        Test.stopTest();
        // Check the result
        List<Lead> updateLeads = [SELECT Id FROM Lead WHERE LeadSource
= 'Dreamforce'];
        System.assertEquals(200, updateLeads.size(), 'ERROR: At least 1 Lead
record not updated correctly');
    }
}
```

4. Control Processes with Queueable Apex

1. AddPrimaryContact.apxc

```
public without sharing class AddPrimaryContact implements Queueable {

  private Contact contact;

  private String state;

  public AddPrimaryContact (Contact inputContact, String inputState) {

    this.contact = inputContact;

    this.state = inputState;

  }

  public void execute(QueueableContext context) {

    // Retrieve 200 Account records

    List<Account> accounts = [SELECT Id FROM Account WHERE
BillingState = :state LIMIT 200];

    // Create empty list of Contact records

    List<Contact> contacts = new List<Contact>();

    // Iterate through the Account records
```

```
        for (Account acc : accounts) {
            // Clone (copy) the contact record, make the clone a child of the specific Account record
            // and add to the list of Contacts
            Contact contactClone = contact.clone();
            contactClone.AccountId = acc.Id;
            contacts.add(contactClone);
        }
        insert contacts;
    }
}
```

2. AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest {
    @isTest
    private static void testQueueableClass() {
        // Load test data
        List<Account> accounts = new List<Account>();
        for (Integer i = 0; i < 500; i++) {
```

```
        Account acc = new Account(Name = 'Test Account');
        if (i < 250) {
            acc.BillingState = 'NY';
        } else {
            acc.BillingState = 'CA';
        }
        accounts.add(acc);
    }
    insert accounts;
    Contact contact = new Contact(FirstName = 'Simon', LastName =
'Connock');
    insert contact;
    // Perform the test
    Test.startTest();
    Id jobId = System.enqueueJob(new AddPrimaryContact(contact, 'CA'));
    Test.stopTest();
    // Check the result
    List<Contact> contacts = [SELECT Id FROM contact WHERE
Contact.Account.BillingState = 'CA'];
    System.assertEquals(200, contacts.size(), 'ERROR: Incorrect number of
```

```
Contact records found');
    }
}
```

5. Schedule Jobs Using the Apex Scheduler

1. DailyLeadProcessor.apxc

```
public without sharing class DailyLeadProcessor implements Schedulable {

    public void execute(SchedulableContext ctx) {
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE
LeadSource = null LIMIT 200];
        for (Lead l : leads) {
            l.LeadSource = 'Dreamforce';
        }
        // Update the modified records
        update leads;
    }
}
```

2. DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {

    private static String CRON_EXP = '0 0 0 ? * * *'; // Midnight every day
    @isTest
```

```
    private static void testSchedulableClass() {
        // Load test data
        List<Lead> leads = new List<Lead>();
        for (Integer i = 0; i < 500; i++) {
            if (i < 250) {
                leads.add(new Lead(LastName = 'Connock', Company =
'Salesforce'));
            } else {
                leads.add(new Lead(LastName = 'Connock', Company =
'Salesforce', LeadSource = 'Other'));
            }
        }
        insert leads;
        // Perform the test
        Test.startTest();
        String jobId = System.schedule('Process Leads', CRON_EXP, new
DailyLeadProcessor());
        Test.stopTest();
        // Check the result
        List<Lead> updatedLeads = [SELECT Id, LeadSource FROM Lead
WHERE LeadSource = 'Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'ERROR: At least 1
record not updated correctly');
        // Check the scheduled time
        List<CronTrigger> cts = [SELECT Id, TimesTriggered, NextFireTime
FROM CronTrigger WHERE Id = :jobId];
        System.debug('Next Fire Time ' + cts[0].NextFireTime);
    }
}
```

4. Apex Integration Services

2. Apex REST Callouts

1. AnimalLocator.apxc

```
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 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);
    }
}
```

2. AnimalLocatorTest.apxc

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

3. AnimalLocatorMock.apxc

```
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPresponse respond(HTTPrequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        return response;
    }
}
```

3. Apex SOAP Callouts

1. ParkLocator.apxc

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

2. ParkLocatorTest.apxc

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

3. ParkService.apxc

```
//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;
        }
    }
}
```

4. ParkServiceMock.apxc

```
@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
      ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac
National Park', 'Yosemite'};
      // end
      response.put('response_x', response_x);
  }
}
```

5. AsyncParkService.apxc

```
//Generated by wsdl2apex


public class AsyncParkService {
```

```
   public class byCountryResponseFuture extends
System.WebServiceCalloutFuture {

      public String[] getValue() {

         ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(thi
s);

         return response.return_x;

      }

   }

   public class AsyncParksImplPort {

      public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';

      public Map<String,String> inputHttpHeaders_x;

      public String clientCertName_x;

      public Integer timeout_x;

      private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};

      public AsyncParkService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {

         ParkService.byCountry request_x = new ParkService.byCountry();

         request_x.arg0 = arg0;

         return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(

           this,

           request_x,

           AsyncParkService.byCountryResponseFuture.class,

           continuation,
```

```
            new String[]{endpoint_x,
            '',
            'http://parks.services/',
            'byCountry',
            'http://parks.services/',
            'byCountryResponse',
            'ParkService.byCountryResponse'}
          );
      }
   }
}
```

4. Apex Web Services

1. AccountManager.apxc

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

```
    }
}
```

2.AccountManagerTest.apxc

```
@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;
    }
}
```

5. Apex Specialist

Step 2 : Automate record creation

1. MaintenanceRequest.apxt

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

2.MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders,
Map<Id,Case> nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status ==
'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }
```

```
        //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;
    }
  }
}
```

Step 3 : Synchronize Salesforce Data With an

External Organization

1. WarehouseCalloutService.apxc

```
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');
  }

}
```

Step 4 : Schedule Synchronization

1.WarehouseSyncSchedule.apxc

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

Step 5 : Test Automation Logic

1.MaintenanceRequest.apxt

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

2.MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders,
Map<Id,Case> nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status ==
'Closed'){
```

```
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                validIds.add(c.Id);
            }
        }
    }

    //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;
    }
  }
}
```

3.MaintenanceRequestHelperTest.apxc

```
@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(id equipmentId,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(eq
uipmentList.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);
    }
}
```

Step 6 : Test Callout Logic

1. WarehouseCalloutServiceMock.apxc

```
@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":fals
e,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id
":"55d66226726b611100aaf742","replacement":true,"quantity":183,"name":"C
ooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d
66226726b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

2.WarehouseCalloutServiceTest.apxc

```
@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);
    }
}
```

Step 7 : Test Scheduling Logic

1. WarehouseSyncScheduleTest.apxc

```
@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();
    }
}
```