Apex Triggers:

1) Get Started with Apex Triggers:

challenge:

Create an Apex trigger:

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:

challenge:

Create a Bulk Apex trigger:

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 (before insert,after update) {
  List<Task> taskList = new List<Task>();
 for(Opportunity opp : Trigger.new) {

                    if(Trigger.isInsert) {
                              if(Opp.StageName == 'Closed Won') {
                                        taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId =
opp.Id));
                              }
                    }
if(Trigger.isUpdate) {
if(Opp.StageName == 'Closed Won'
&& Opp.StageName != Trigger.oldMap.get(opp.Id).StageName) {
          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:

challenge:

Create a Unit Test for a Simple Apex Class

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 Class:

```
public class VerifyDate {
	public static Date CheckDates(Date date1, Date date2) {
		if(DateWithin30Days(date1,date2)) {
			return date2;
		} else {
			return SetEndOfMonthDate(date1);
		}
	}

	@TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
	if( date2 < date1) { return false; }

		Date date30Days = date1.addDays(30); //create a date 30 days away from date1
		if( date2 >= date30Days ) { return false; }
		else { return true; }
	}
	@TestVisible private static Date SetEndOfMonthDate(Date date1) {
		Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
		Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
```

```
                return lastDay;
        }

}
```

Code for TestVerifyDate:

```
        @isTest
public 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:

challenge:

Create a Unit Test for a Simple Apex Trigger

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 forRestrictContactByName:

```
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') {
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
                }
}
}
```

Code for TestRestrictContactByName Class:

```
@isTest
public class TestRestrictContactByName {

    @isTest  static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';

        Test.startTest();
        Database.SaveResult result = Database.insert(cnt, false);
        Test.stopTest();

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

3) Create Test Data for Apex Tests:

challenge:

Create a Contact Test Factory

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 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:
challenge:
Create an Apex class that uses the @future annotation to update Account records.
Create an Apex class with a future method that accepts a List of Account IDs and updates a custom field on the Account object with the number of contacts associated to the Account. Write unit tests that 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 AccountProcessor:

```
public class AccountProcessor {
   @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accountsToUpdate = new List<Account>();
        List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account Where Id IN :accountIds];
        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}
```

Code for AccountProcessorTest:

```
@IsTest
private class AccountProcessorTest {
    @IsTest
    private static void testCountContacts() {
        Account newAccount = new Account(Name='Test Account');
        insert newAccount;

        Contact newContact1 = new Contact(FirstName='John',
                                          LastName='Doe',
                                          AccountId=newAccount.Id);
```

```
        insert newContact1;
        Contact newContact2 = new Contact(FirstName='Jane',
                                          LastName='Doe',
                                          AccountId=newAccount.Id);
        insert newContact2;
        List<Id> accountIds = new List<Id>();
           accountIds.add(newAccount.Id);
        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
    }
}
```

2) Use Batch Apex:

challenge:

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

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>, Database.Stateful {
    // instance member to retain state across transactions
    global Integer recordsProcessed = 0;
    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator('SELECT Id, LeadSource FROM Lead');
    }
    global 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';
                // increment the instance member counter
                recordsProcessed = recordsProcessed + 1;
        }
        update leads;
    }
    global void finish(Database.BatchableContext bc){
        System.debug(recordsProcessed + ' records processed. Shazam!');
    }
}
```

Code for LeadProcessorTest:

```
@isTest
public class LeadProcessorTest {
 @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        // insert 200 leads
        for (Integer i=0;i<200;i++) {
            leads.add(new Lead(LastName='Lead '+i,
                Company='Lead', Status='Open - Not Contacted'));
        }
        insert leads;
    }
    static testmethod void test() {
```

```
        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp, 200);
        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:

challenge:

Create a Queueable Apex class that inserts Contacts for Accounts.

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

Create an Apex class:

Name: AddPrimaryContact

Interface: Queueable

Create a constructor for the class that accepts as its first argument a Contact sObject and a second argument as a string for the State abbreviation

The execute method must query for a maximum of 200 Accounts with the BillingState specified by the State abbreviation passed into the constructor and insert the Contact sObject record associated to each Account. Look at the sObject clone() method.

Create an Apex test class:

Name: AddPrimaryContactTest

In the test class, insert 50 Account records for BillingState NY and 50 Account records for BillingState CA

Create an instance of the AddPrimaryContact class, enqueue the job, and assert that a Contact record was inserted for each of the 50 Accounts with the BillingState of CA

The unit tests must cover all lines of code included in the AddPrimaryContact class, resulting in 100% code coverage

Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

Code for AddPrimaryContact:

```
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 ) FROM 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:

challenge:

Create an Apex class that uses Scheduled Apex to update Lead records.

Create an Apex class that implements the Schedulable interface to update Lead records with a specific LeadSource. (This is very similar to what you did for Batch Apex.)

Create an Apex class:

Name: DailyLeadProcessor

Interface: Schedulable

The execute method must find the first 200 Lead records with a blank LeadSource 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 ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];
        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();
            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }
            update newLeads;
        }
    }
}
```

Code for DailyLeadProcessorTest:

```
@isTest
private class DailyLeadProcessorTest{
    //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
    public static String CRON_EXP = '0 0 0 2 6 ? 2022';
    static testmethod void testScheduledJob(){
        List<Lead> leads = new List<Lead>();
        for(Integer i = 0; i < 200; i++){
            Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test
Company ' + i, Status = 'Open - Not Contacted');
            leads.add(lead);
        }
        insert leads;
```

```
        Test.startTest();

        String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new
DailyLeadProcessor());

        Test.stopTest();
    }
}
```

Apex Integration Services:
1) Apex REST Callouts:
challenge:
Create an Apex class that calls a REST endpoint and write a test class.
Create an Apex class that calls a REST endpoint to return the name of an animal, 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.
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 AnimalLocatorTest:

```
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 {
    global HTTPResponse respond(HTTPRequest request) {
        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 );
    }
}
```

2) Apex SOAP Callouts:

challenge:

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

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 ParkService:

```
public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new String[]{'return','http://parks.services/',null,'0','-
1','false'};
        private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
```

```
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new Map<String,
ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
            response_x = response_map_x.get('response_x');
            return response_x.return_x;
        }
    }
}
```

Code for ParkLocator:

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

Code for ParkLocatorTest:

```
@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);
    }
}
```

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
        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);
    }
}
```

3) Apex Web Services:
challenge:
Create an Apex REST service that returns an account and its contacts.
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:

Code for CreateDefaultData:

```
public with sharing class CreateDefaultData{
```

```
    Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine Maintenance';
    //gets value from custom metadata How_We_Roll_Settings__mdt to know if Default data was created
    @AuraEnabled
    public static Boolean isDataCreated() {
        How_We_Roll_Settings__c       customSetting = How_We_Roll_Settings__c.getOrgDefaults();
        return customSetting.Is_Data_Created__c;
    }

    //creates Default Data for How We Roll application
    @AuraEnabled
    public static void createDefaultData(){
        List<Vehicle__c> vehicles = createVehicles();
        List<Product2> equipment = createEquipment();
        List<Case> maintenanceRequest = createMaintenanceRequest(vehicles);
        List<Equipment_Maintenance_Item__c> joinRecords = createJoinRecords(equipment, maintenanceRequest);

        updateCustomSetting(true);
    }


    public static void updateCustomSetting(Boolean isDataCreated){
        How_We_Roll_Settings__c       customSetting = How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = isDataCreated;
        upsert customSetting;
    }

    public static List<Vehicle__c> createVehicles(){
        List<Vehicle__c> vehicles = new List<Vehicle__c>();
        vehicles.add(new Vehicle__c(Name = 'Toy Hauler RV', Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1, Model__c = 'Toy Hauler RV'));
```

```
        vehicles.add(new Vehicle__c(Name = 'Travel Trailer RV', Air_Conditioner__c = true,
Bathrooms__c = 2, Bedrooms__c = 2, Model__c = 'Travel Trailer RV'));
        vehicles.add(new Vehicle__c(Name = 'Teardrop Camper', Air_Conditioner__c = true,
Bathrooms__c = 1, Bedrooms__c = 1, Model__c = 'Teardrop Camper'));
        vehicles.add(new Vehicle__c(Name = 'Pop-Up Camper', Air_Conditioner__c = true,
Bathrooms__c = 1, Bedrooms__c = 1, Model__c = 'Pop-Up Camper'));
        insert vehicles;
        return vehicles;
    }

    public static List<Product2> createEquipment(){
        List<Product2> equipments = new List<Product2>();
        equipments.add(new Product2(Warehouse_SKU__c =
'55d66226726b611100aaf741',name = 'Generator 1000 kW', Replacement_Part__c =
true,Cost__c = 100 ,Maintenance_Cycle__c = 100));
        equipments.add(new Product2(name = 'Fuse 20B',Replacement_Part__c = true,Cost__c =
1000, Maintenance_Cycle__c = 30  ));
        equipments.add(new Product2(name = 'Breaker 13C',Replacement_Part__c =
true,Cost__c = 100  , Maintenance_Cycle__c = 15));
        equipments.add(new Product2(name = 'UPS 20 VA',Replacement_Part__c = true,Cost__c
= 200  , Maintenance_Cycle__c = 60));
        insert equipments;
        return equipments;

    }

    public static List<Case> createMaintenanceRequest(List<Vehicle__c> vehicles){
        List<Case> maintenanceRequests = new List<Case>();
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(1).Id, Type =
TYPE_ROUTINE_MAINTENANCE, Date_Reported__c = Date.today()));
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(2).Id, Type =
TYPE_ROUTINE_MAINTENANCE, Date_Reported__c = Date.today()));
        insert maintenanceRequests;
        return maintenanceRequests;
```

```
    }

    public static List<Equipment_Maintenance_Item__c> createJoinRecords(List<Product2>
equipment, List<Case> maintenanceRequest){
        List<Equipment_Maintenance_Item__c> joinRecords = new
List<Equipment_Maintenance_Item__c>();
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(0).Id, Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(1).Id, Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(2).Id, Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(0).Id, Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(1).Id, Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(2).Id, Maintenance_Request__c = maintenanceRequest.get(1).Id));
        insert joinRecords;
        return joinRecords;

    }
}
```

Codefor CreateDefaultDataTest:

```
@isTest
private class CreateDefaultDataTest {
    @isTest
    static void createData_test(){
        Test.startTest();
        CreateDefaultData.createDefaultData();
        List<Vehicle__c> vehicles = [SELECT Id FROM Vehicle__c];
        List<Product2> equipment = [SELECT Id FROM Product2];
        List<Case> maintenanceRequest = [SELECT Id FROM Case];
        List<Equipment_Maintenance_Item__c> joinRecords = [SELECT Id FROM
```

```
Equipment_Maintenance_Item__c];

        System.assertEquals(4, vehicles.size(), 'There should have been 4 vehicles created');
        System.assertEquals(4, equipment.size(), 'There should have been 4 equipment created');
        System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2
maintenance request created');
        System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment
maintenance items created');

    }

    @isTest
    static void updateCustomSetting_test(){
        How_We_Roll_Settings__c       customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = false;
        upsert customSetting;

        System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom setting
How_We_Roll_Settings__c.Is_Data_Created__c should be false');

        customSetting.Is_Data_Created__c = true;
        upsert customSetting;

        System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting
How_We_Roll_Settings__c.Is_Data_Created__c should be true');

    }
}
```

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()){
        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>();
        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 (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;
       }}}
```

Code for MaintenanceRequestHelperTest:

```
@isTest
public with sharing class MaintenanceRequestHelperTest {
   private static Vehicle__c createVehicle(){
      Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
      return vehicle;
   }
   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;
```

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

Code for WarehouseCalloutService:

```
public with sharing class WarehouseCalloutService {
    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    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);
            }
        }
    }
}
```

Code for WarehouseCalloutServiceMock:

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request){
        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name
":"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
```

```
        return response;
    }
}
```

Code for WarehouseCalloutServiceTest:

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

Code for WarehouseSyncSchedule:

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

Code for WarehouseSyncScheduleTest:

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
     @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();
```

```
    }
}
```

Code for MaintenanceRequest Trigger:

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