MODULE - APEX TRIGGERS

CHALLENGE- Create Apex Trigger

```
trigger AccountAddressTrigger on Account (before insert, before update) {

    List<Account> acclst=new List<Account>();
      for(account a:trigger.new){
          if(a.Match_Billing_Address__c == true && a.BillingPostalCode != null){
              a.ShippingPostalCode=a.BillingPostalCode;
          }
      }
}

```

CHALLENGE - Create a Bulk Apex Trigger

```
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

        }
    }

    if(tasklist.size()>0){
        insert tasklist;
    }
}
```

MODULE - APEX TESTING

CHALLENGE - Create a Unit Test for Simple Apex Class

```
public class VerifyDate {

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

 //method to check if date2 is within the next 30 days of date1
 @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
        //check for date2 being in the past
 if( date2 < date1) { return false; }

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

 //method to return the end of the month of a given date
 @TestVisible private static Date SetEndOfMonthDate(Date date1) {
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
        Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
        return lastDay;
 }

}
```

CHALLENGE - Create a Unit Test for a Simple Apex Trigger

```
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

        }

  }

```

CHALLENGE - Create a Contact Test Factory

```
public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numcnt, string lastname){
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cont = new Contact(FirstName = 'Test '+i, LastName = lastname);
            contacts.add(cont);
        }
        return contacts;

    }

}
```

MODULE - ASYNCHRONOUS APEX

CHALLENGE - Create an Apex Class thst uses the @future annotation to update Account record

```
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 acct:accounts){
            List<Contact> contactList = acct.Contacts;
            acct.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acct);

        }
        update accountsToUpdate;
    }

}
```

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

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

        Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId = newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

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

    }

```

```
23 }
```

CHALLENGE - Create an Apex Class that uses Batch Apex to update Lead record

```
1  public without sharing class LeadProcessor implements
   Database.Batchable<sObject>, Database.Stateful {
2
3      public Integer recordCount = 0;
4
5      public Database.QueryLocator start(Database.BatchableContext
   dbc) {
6          return Database.getQueryLocator([SELECT Id, Name FROM
   Lead]);
7      }
8
9      public void execute(Database.BatchableContext dbc, List<Lead>
   leads) {
10         for(Lead l : leads) {
11             l.leadSource = 'Dreamforce';
12         }
13         update leads;
14         recordCount = recordCount + leads.size();
15     }
16     public void finish (Database.BatchableContext dbc) {
17         system.debug('Total records processed' + recordcount);
18     }
19 }
```

```
1  @isTest
2  private class LeadProcessorTest {
3
4      @isTest
5      private static void testBatchClass() {
6
7          List<Lead> leads = new List<Lead>();
8          for(Integer i=0; i<200; i++) {
9              leads.add(new Lead(LastName='Cannock',
   Company='Salesforce'));
10         }
11         insert leads;
12
```

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

        List<Lead> updatedLeads = [SELECT Id FROM Lead WHERE
  LeadSource = 'Dreamforce'];
        system.assertEquals(200, updatedLeads.size(), 'ERROR: At

    }

}
```

CHALLENGE - Create a Queueable Apex class that inserts Contact for Account

```
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) {

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

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

        for(Account acc : accounts) {

            Contact contactClone = contact.clone();
            contactClone.AccountId = acc.Id;
            contacts.add(contactClone);
        }

```

```
        insert contacts;
    }

}
```

```
@isTest
private class AddPrimaryContactTest {

    @isTest
    private static void testQueueableClass() {

        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;

        Test.startTest();
        Id jobId = system.enqueueJob(new
AddPrimaryContact(contact, 'CA'));
        Test.stopTest();

        List<Contact> contacts = [SELECT Id FROM Contact WHERE
Contact.Account.BillingState = 'CA'];
        System.assertEquals(200, contacts.size(), 'ERROR:


    }

}
```

CHALLENGE - Create an Apex class that uses Schedule Apex to update Lead records

```
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 leads;
    }
      }
}
```

```
@isTest
public class DailyLeadProcessorTest {

    private static String CRON_EXP = '0 4 0 2 6 ? 2023';

    @isTest
    private static void testSchedulableClass() {

        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;

        Test.startTest();
        String jobId = System.schedule('Process Lead', CRON_EXP, new DailyLeadProcessor());
```

```
        Test.stopTest();

        List<Lead> updatedLeads = [SELECT Id, LeadSource FROM
  Lead WHERE LeadSource='Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'ERROR: At


        List<CronTrigger> cts = [SELECT Id, TimesTriggered,
  NextFireTime FROM CronTrigger WHERE Id = :jobId];
        System.debug('Next Fire Time' + cts[0].NextFireTime);
    }
}
```

MODULE - APEX INTEGRATION SERVICES

CHALLENGE - Create an Apex class that uses a REST endpoint and write a test class

```
public class AnimalLocator {

    public static String getAnimalNameById (Integer i) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(' https://th-apex-http-

        request.setMethod('GET');
        Map<String, Object> animal = new Map<String, Object>();
        HttpResponse response = http.send(request);

        if(response.getStatusCode()==200) {
            Map<String, Object> result = (Map<String,
  Object>)JSON.deserializeUntyped(response.getBody());
            animal = (Map<String, Object>) result.get('animal');
        }
        return (String)animal.get('name');
    }

}
```

```
@isTest
```

```
2  private class AnimalLocatorTest {
3      @isTest static void AnimalLocatorMock1() {
4          try{
5              Test.setMock(HttpCalloutMock.class , new
   AnimalLocatorMock());
6
7              string result = AnimalLocator.getAnimalNameById(1);
8              string expectedResult = 'fox';
9              System.assertEquals(result,expectedResult);
10         }
11         catch(exception e) {
12             System.debug('The following exception has occurred: '
   + e.getMessage());
13         }
14     }
```

CHALLENGE - Generate an Apex class using WSDL2Apex and write a test class

```
1  public class ParkService {
2      public class byCountryResponse {
3          public String[] return_x;
4          private String[] return_x_type_info = new
   String[]{'return','http://parks.services/',null,'0','-

5          private String[] apex_schema_type_info = new
   String[]{'http://parks.services/','false','false'};
6          private String[] field_order_type_info = new
   String[]{'return_x'};
7      }
8      public class byCountry {
9          public String arg0;
10         private String[] arg0_type_info = new
   String[]{'arg0','http://parks.services/',null,'0','1','false'};
11         private String[] apex_schema_type_info = new
   String[]{'http://parks.services/','false','false'};
12         private String[] field_order_type_info = new
   String[]{'arg0'};
13     }
14     public class ParksImplPort {
15         public String endpoint_x = 'https://th-apex-soap-
```

```
16          public Map<String,String> inputHttpHeaders_x;
17          public Map<String,String> outputHttpHeaders_x;
18          public String clientCertName_x;
19          public String clientCert_x;
20          public String clientCertPasswd_x;
21          public Integer timeout_x;
22          private String[] ns_map_type_info = new
    String[]{'http://parks.services/', 'ParkService'};
23          public String[] byCountry(String arg0) {
24              ParkService.byCountry request_x = new
    ParkService.byCountry();
25              request_x.arg0 = arg0;
26              ParkService.byCountryResponse response_x;
27              Map<String, ParkService.byCountryResponse>
    response_map_x = new Map<String,
    ParkService.byCountryResponse>();
28              response_map_x.put('response_x', response_x);
29              WebServiceCallout.invoke(
30                this,
31                request_x,
32                response_map_x,
33                new String[]{endpoint_x,
34                '',
35                'http://parks.services/',
36                'byCountry',
37                'http://parks.services/',
38                'byCountryResponse',
39                'ParkService.byCountryResponse'}
40              );
41              response_x = response_map_x.get('response_x');
42              return response_x.return_x;
43          }
44      }
45  }public class ParkLocator {
46      public static string[] country(string theCountry) {
47          ParkService.ParksImplPort parkSvc = new
    ParkService.ParksImplPort();
48          return parkSvc.byCountry(theCountry);
49
50      }
51
52  }
```

```
@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);
    }

}
```

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

}

```

CHALLENGE - Create an Apex REST service that returns an account and its contacts

```
@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;
    }
}
```

```
@isTest
private class AccountManagerTest {
    @isTest static void testGetContactsByAccountId(){
         Id recordId = createTestRecord();
         RestRequest request = new RestRequest();
         request.requestUri
='https://yourInstance.my.salesforce.com/services/apexrest/Accoun
ts/'
                             + 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;

```

```
27 }
```

APEX SPECIALIST SUPERBADGE

```
1  trigger MaintenanceRequest on Case (before update, after update)
   {
2      if(Trigger.isUpdate && Trigger.isAfter){
3          MaintenanceRequestHelper.updateWorkOrders(Trigger.New,
   Trigger.OldMap);
4
5      }
6  }
```

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

7                      validIds.add(c.Id);
8                  }
9              }
10         }
11         if (!validIds.isEmpty()){
12             Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT
   Id, Vehicle__c,
   Equipment__c,Equipment__r.Maintenance_Cycle__c,(SELECT
   Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
   FROM Case WHERE Id IN :validIds]);
13             Map<Id,Decimal> maintenanceCycles = new
   Map<ID,Decimal>();
14             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 : 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> clonedList = new
List<Equipment_Maintenance_Item__c>();
            for (Case nc : newCases) {
                for (Equipment_Maintenance_Item__c clonedListItem
: closedCasesM.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;
        }
```

```
    }
}
```

```
public with sharing class WarehouseCalloutService  {
    private static final String WAREHOUSE_URL = 'https://th-


    @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 = (Integer) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String)
mapJson.get('sku');
                myEq.Current_Inventory__c = (Double)
mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }
```

```
31
32              if (warehouseEq.size() > 0){
33                  upsert warehouseEq;
34                  System.debug('Your equipment was synced with the

35                  System.debug(warehouseEq);
36              }
37          }
38      }
39  }
```

```
1  global class WarehouseSyncSchedule implements Schedulable {
2      global void execute(SchedulableContext ctx) {
3
4          WarehouseCalloutService.runWarehouseEquipmentSync();
5      }
6  }
```

```
1    @isTest
2  public with sharing class MaintenanceRequestHelperTest {
3      private static final string STATUS_NEW = 'New';
4      private static final string WORKING = 'Working';
5      private static final string CLOSED = 'Closed';
6      private static final string REQUEST_TYPE = 'Routine

7      private static final string REQUEST_SUBJECT = 'Testing';
8
9
10     private static Vehicle__c createVehicle(){
11         Vehicle__c vehicle = new Vehicle__C(name = 'SuperTrack');
12         return vehicle;
13     }
14     private static Product2 createEq(){
15         product2 equipment = new product2(name ='SuperEquipment',
   lifespan_months__c = 10, maintenance_cycle__c = 10,
   replacement_part__c = true);
16         return equipment;
17     }
18     private static Case createMaintenanceRequest(id vehicleId, id
   equipmentId){
```

```
        case cs = new case(Type='REPAIR',
                           Status='STATUS_NEW',
                           Origin='REQUEST_ORIGIN',
                           Subject='REQUEST_subject',
                           Equipment__c=equipmentId,
                           Vehicle__c=vehicleId);
        return cs;
    }
    private static Equipment_Maintenance_Item__c
createWorkPart(id equipmentId,id requestId){
        Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
Maintenance_Request__c = requestId);
        return wp;
    }
    @isTest
    private static void testMaintenanceRequestPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;
        Product2 equipment = createEq();
        insert equipment;
        id equipmentId = equipment.Id;

        case somethingToUpdate =
createMaintenanceRequest(vehicleId,equipmentId);
        insert somethingToUpdate;

        Equipment_Maintenance_Item__c workP =
createworkPart(equipmentId,somethingToUpdate.id);
        insert workP;

        test.startTest();
        somethingToUpdate.status = 'Closed';
        update somethingToUpdate;
        test.stopTest();

        Case newReq = [Select id, subject, type, Equipment__c,
Date_Reported__c, Vehicle__c, Date_Due__c from case where status
=:STATUS_NEW];
        Equipment_Maintenance_Item__c workPart = [select id from
Equipment_Maintenance_Item__c where Maintenance_Request__c
```

```
=:newReq.Id];
53
54          system.assert(workPart != null);
55          system.assert(newreq.Subject != null);
56          system.assertEquals(newReq.Type, REQUEST_TYPE);
57          SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
58          SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
59
SYSTEM.assertEquals(newReq.Date_Reported__c,system.today());
60      }
61
62      @isTest
63      private static void testMaintenanceRequestNegative(){
64          Vehicle__C vehicle = createVehicle();
65          insert vehicle;
66          id vehicleId = vehicle.Id;
67          product2 equipment = createEq();
68          insert equipment;
69          id equipmentId = equipment.Id;
70          case emptyReq =
createMaintenanceRequest(vehicleId,equipmentId);
71          insert emptyReq;
72          Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId, emptyReq.Id);
73          insert workP;
74
75          test.startTest();
76          emptyReq.Status = WORKING ;
77          update emptyReq;
78          test.stopTest();
79
80          list<case> allRequest = [select id from case];
81          Equipment_Maintenance_Item__c workPart = [select id from
Equipment_Maintenance_Item__c where Maintenance_Request__c =
:emptyReq.Id];
82          system.assert(workPart != null);
83          system.assert(allRequest.size() == 1);
84      }
85      @isTest
86      private static void testMaintenanceRequestBulk(){
87          list<Vehicle__C> vehicleList = new list<Vehicle__C>();
88          list<Product2> equipmentList = new list<Product2>();
89          list<Equipment_Maintenance_Item__c> workPartList = new
```

```
   list<Equipment_Maintenance_Item__c>();
90          list<case> requestList = new list<case>();
91          list<id> oldRequestIds = new list<id>();
92
93          for(integer i = 0; i < 300; i++){
94              vehicleList.add(createVehicle());
95              equipmentList.add(createEq());
96          }
97          insert vehicleList;
98          insert equipmentList;
99
100          for(integer i = 0; i < 300; i++){
101
   requestList.add(createMaintenanceRequest(vehicleList.get(i).id,eq

102          }
103          insert requestList;
104
105          for(integer i = 0; i < 300; i++){
106
   workPartList.add(createWorkPart(equipmentList.get(i).id,
   requestList.get(i).id));
107          }
108          insert workPartList;
109
110          test.startTest();
111          for(case req : requestList){
112              req.Status = CLOSED;
113              oldRequestIds.add(req.Id);
114          }
115
116          update requestList;
117          test.stopTest();
118
119          list<case> allRequest = [select id from case where
   status =:STATUS_NEW];
120          list<Equipment_Maintenance_Item__c> workParts = [select
   id from Equipment_Maintenance_Item__c where
   Maintenance_Request__c in: oldRequestIds];
121          system.assert(allRequest.size() == 300);
122      }
123 }
```

```
@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




        response.setStatusCode(200);
        return response;
    }
}
```

```
@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]);
    }
}
```

```
@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

        Test.stopTest();
        //Contains schedule information for a scheduled job.
CronTrigger is similar to a cron job on UNIX systems.
        // 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 ');


    }
}
```

```
public with sharing class CreateDefaultData{
    Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine

    //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

Maintenance_Cycle__c = 30  ));
```

```
42          equipments.add(new Product2(name = 'Breaker
    Maintenance_Cycle__c = 15));
43          equipments.add(new Product2(name = 'UPS 20
    Maintenance_Cycle__c = 60));
44          insert equipments;
45          return equipments;
46
47      }
48
49      public static List<Case>
    createMaintenanceRequest(List<Vehicle__c> vehicles){
50          List<Case> maintenanceRequests = new List<Case>();
51          maintenanceRequests.add(new Case(Vehicle__c =
    vehicles.get(1).Id, Type = TYPE_ROUTINE_MAINTENANCE,
    Date_Reported__c = Date.today()));
52          maintenanceRequests.add(new Case(Vehicle__c =
    vehicles.get(2).Id, Type = TYPE_ROUTINE_MAINTENANCE,
    Date_Reported__c = Date.today()));
53          insert maintenanceRequests;
54          return maintenanceRequests;
55      }
56
57      public static List<Equipment_Maintenance_Item__c>
    createJoinRecords(List<Product2> equipment, List<Case>
    maintenanceRequest){
58          List<Equipment_Maintenance_Item__c> joinRecords = new
    List<Equipment_Maintenance_Item__c>();
59          joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
    Maintenance_Request__c = maintenanceRequest.get(0).Id));
60          joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
    Maintenance_Request__c = maintenanceRequest.get(0).Id));
61          joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
    Maintenance_Request__c = maintenanceRequest.get(0).Id));
62          joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
    Maintenance_Request__c = maintenanceRequest.get(1).Id));
63          joinRecords.add(new
```

```
    Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
    Maintenance_Request__c = maintenanceRequest.get(1).Id));
64          joinRecords.add(new
    Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
    Maintenance_Request__c = maintenanceRequest.get(1).Id));
65          insert joinRecords;
66          return joinRecords;
67
68      }
69  }
```

```
1   @isTest
2   private class CreateDefaultDataTest {
3       @isTest
4       static void createData_test(){
5           Test.startTest();
6           CreateDefaultData.createDefaultData();
7           List<Vehicle__c> vehicles = [SELECT Id FROM Vehicle__c];
8           List<Product2> equipment = [SELECT Id FROM Product2];
9           List<Case> maintenanceRequest = [SELECT Id FROM Case];
10          List<Equipment_Maintenance_Item__c> joinRecords = [SELECT
    Id FROM Equipment_Maintenance_Item__c];
11
12          System.assertEquals(4, vehicles.size(), 'There should

13          System.assertEquals(4, equipment.size(), 'There should

14          System.assertEquals(2, maintenanceRequest.size(), 'There

15          System.assertEquals(6, joinRecords.size(), 'There should


16
17      }
18
19      @isTest
20      static void updateCustomSetting_test(){
21          How_We_Roll_Settings__c   customSetting =
    How_We_Roll_Settings__c.getOrgDefaults();
22          customSetting.Is_Data_Created__c = false;
23          upsert customSetting;
24
25          System.assertEquals(false,
```

```
    CreateDefaultData.isDataCreated(), 'The custom setting

26
27          customSetting.Is_Data_Created__c = true;
28          upsert customSetting;
29
30          System.assertEquals(true,
    CreateDefaultData.isDataCreated(), 'The custom setting

31
32      }
33  }
```