**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!=trigger.oldMap.get(opp.id).stageName)
        {
            taskList.add(new Task(Subject = 'Follow Up Test

                                  WhatId = opp.Id));
        }
    }

    if(taskList.size()>0){
        insert taskList;
```

```
14      }
15 }
```

**MODULE - APEX TESTING**

**CHALLENGE - Create a Unit Test for a Simple Apex Class**

```
1  public class VerifyDate {
2
3
4   public static Date CheckDates(Date date1, Date date2) {
5
6         if(DateWithin30Days(date1,date2)) {
7               return date2;
8         } else {
9               return SetEndOfMonthDate(date1);
10        }
11  }
12
13
14  private static Boolean DateWithin30Days(Date date1, Date
    date2) {
15
16  if( date2 < date1) { return false; }
17  //check that date2 is within (>=) 30 days of date1
18  Date date30Days = date1.addDays(30); //create a date 30
    days away from date1
19        if( date2 >= date30Days ) { return false; }
20        else { return true; }
21  }
22
23
```

```
 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) {


  For (Contact c : Trigger.New) {
        if(c.LastName == 'INVALIDNAME') {
              c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
        }

  }



}

```

**CHALLENGE - Create a Contact Test Factory**

```
public class RandomContactFactory{

    public static List<Contact> generateRandomContacts(integer n,string LastName){
```

```
    integer n1=n;
    List<contact> c1 = new list<contact>();
    list<contact> c2 =new list<contact>();
     c1 = [select FirstName from Contact Limit : n1];
     integer i=0;
     for(contact cnew : c1){
     contact cnew1 = new contact();
     cnew1.firstname = cnew.firstname + i;
     c2.add(cnew1);
     i++;
     }
    return c2;
    }
}
```

**MODULE - ASYNCHRONOUS APEX**

**CHALLENGE - Create an Apex class that uses the @future annotation to update Account records.**

```
public class AccountProcessor
{
  @future
  public static void countContacts(Set<id> setId)
  {
      List<Account> lstAccount = [select id,Number_of_Contacts__c
  , (select id from contacts ) from account where id in :setId ];
      for( Account acc : lstAccount )
      {
          List<Contact> lstCont = acc.contacts ;
          acc.Number_of_Contacts__c = lstCont.size();
      }
      update lstAccount;
  }
}
```

```
@IsTest
```

```
2  public class AccountProcessorTest {
3      public static testmethod void TestAccountProcessorTest()
4      {
5          Account a = new Account();
6          a.Name = 'Test Account';
7          Insert a;
8
9          Contact cont = New Contact();
10         cont.FirstName ='Bob';
11         cont.LastName ='Masters';
12         cont.AccountId = a.Id;
13         Insert cont;
14         set<Id> setAccId = new Set<ID>();
15         setAccId.add(a.id);
16
17         Test.startTest();
18             AccountProcessor.countContacts(setAccId);
19         Test.stopTest();
20         Account ACC = [select Number_of_Contacts__c from Account
   where id = :a.id LIMIT 1];
21         System.assertEquals (
   Integer.valueOf(ACC.Number_of_Contacts__c) ,1);
22   }
23 }
```

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

```
1  global class LeadProcessor implements
2  Database.Batchable<sObject>, Database.Stateful {
3      global Integer recordsProcessed = 0;
4
5      global Database.QueryLocator
   start(Database.BatchableContext bc) {
6          return Database.getQueryLocator('SELECT Id,

7      }
8
9      global void execute(Database.BatchableContext bc,
```

```
   List<Lead> scope){
10          // process each batch of records
11          List<Lead> leads = new List<Lead>();
12          for (Lead lead : scope) {
13                  lead.LeadSource = 'Dreamforce';
14                  // increment the instance member counter
15                  recordsProcessed = recordsProcessed + 1;
16          }
17          update leads;
18      }
19
20      global void finish(Database.BatchableContext bc){
21          System.debug(recordsProcessed + ' records

22      }
23  }
```

```
1  @isTest
2  public class LeadProcessorTest {
3   @testSetup
4      static void setup() {
5          List<Lead> leads = new List<Lead>();
6          for (Integer i=0;i<200;i++) {
7              leads.add(new Lead(LastName='Lead '+i,
8                  Company='Lead', Status='Open - Not

9          }
10         insert leads;
11     }
12
13     static testmethod void test() {
14         Test.startTest();
15         LeadProcessor lp = new LeadProcessor();
16         Id batchId = Database.executeBatch(lp, 200);
17         Test.stopTest();
```

```
18
19          System.assertEquals(200, [select count() from lead
    where LeadSource = 'Dreamforce']);
20      }
21  }
22
```

**CHALLENGE - Create a Queueable Apex class that inserts Contacts for Accounts.**

```
1  public class AddPrimaryContact implements Queueable
2  {
3      private Contact c;
4      private String state;
5      public  AddPrimaryContact(Contact c, String state)
6      {
7          this.c = c;
8          this.state = state;
9      }
10     public void execute(QueueableContext context)
11     {
12          List<Account> ListAccount = [SELECT ID, Name
    ,(Select id,FirstName,LastName from contacts ) FROM ACCOUNT
    WHERE BillingState = :state LIMIT 200];
13          List<Contact> lstContact = new List<Contact>();
14          for (Account acc:ListAccount)
15          {
16                  Contact cont =
    c.clone(false,false,false,false);
17                  cont.AccountId =  acc.id;
18                  lstContact.add( cont );
19          }
20          if(lstContact.size() >0 )
21          {
22              insert lstContact;
23          }
```

```
    }

}
```

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

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

```
global class DailyLeadProcessor implements Schedulable {
 global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];
        if(!lList.isEmpty()) {
    for(Lead l: lList) {
     l.LeadSource = 'Dreamforce';
    }
    update lList;
   }
    }

}
```

```
@isTest
public class DailyLeadProcessorTest {

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

        // Stopping the test will run the job synchronously
        Test.stopTest();
    }
```

```
17
18 }
```

**MODULE - APEX INTEGRATION SERVICES**

**CHALLENGE - Create an Apex class that calls a REST endpoint and write a test class.**

```
1  public class AnimalLocator {
2      public class Animal {
3          public Integer id;
4          public String name;
5          public String eats;
6          public String says;
7      }
8      public class AnimalResult {
9          public Animal animal;
10     }
11
12   public static String getAnimalNameById(Integer id) {
13          Http http = new Http();
14          HttpRequest request = new HttpRequest();
15          request.setEndpoint('https://th-apex-http-

16          request.setMethod('GET');
17          HttpResponse response = http.send(request);
18          AnimalResult result = (AnimalResult)
    JSON.deserialize(response.getBody(), AnimalResult.class);
19          return result.animal.name;
20     }
21 }
```

```
1  @isTest
2  private class AnimalLocatorTest{
3      @isTest static void AnimalLocatorMock1() {
4          Test.setMock(HttpCalloutMock.class, new
```

```
    AnimalLocatorMock());
5           string result = AnimalLocator.getAnimalNameById(3);
6           String expectedResult = 'chicken';
7           System.assertEquals(result,expectedResult );
8       }
9   }
```

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

```
1   //Generated by wsdl2apex
2
3   public class ParkService {
4       public class byCountryResponse {
5           public String[] return_x;
6           private String[] return_x_type_info = new
    String[]{'return','http://parks.services/',null,'0','-

7           private String[] apex_schema_type_info = new
    String[]{'http://parks.services/','false','false'};
8           private String[] field_order_type_info = new
    String[]{'return_x'};
9       }
10      public class byCountry {
11          public String arg0;
12          private String[] arg0_type_info = new
    String[]{'arg0','http://parks.services/',null,'0','1','fals

13          private String[] apex_schema_type_info = new
    String[]{'http://parks.services/','false','false'};
14          private String[] field_order_type_info = new
    String[]{'arg0'};
15      }
16      public class ParksImplPort {
17          public String endpoint_x = 'https://th-apex-soap-

18          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;
        }
    }
}
```

```
public class ParkLocator {
```

```
2      public static string[] country(string theCountry) {
3          ParkService.ParksImplPort  parkSvc = new
   ParkService.ParksImplPort(); // remove space
4          return parkSvc.byCountry(theCountry);
5      }
6  }
7
```

```
1  @isTest
2  private class ParkLocatorTest {
3      @isTest static void testCallout() {
4          Test.setMock(WebServiceMock.class, new
   ParkServiceMock ());
5          String country = 'United States';
6          List<String> result = ParkLocator.country(country);
7          List<String> parks = new
   List<String>{'Yellowstone', 'Mackinac National Park',
   'Yosemite'};
8           System.assertEquals(parks, result);
9      }
10 }
```

```
1  @isTest
2  global class ParkServiceMock implements WebServiceMock {
3     global void doInvoke(
4              Object stub,
5              Object request,
6              Map<String, Object> response,
7              String endpoint,
8              String soapAction,
9              String requestName,
10             String responseNS,
```

```
            String responseName,
            String responseType) {
        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);
    }
}
```

**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/Accounts/'
7               + recordId+'/contacts';
8           request.httpMethod = 'GET';
9           RestContext.request = request;
10          Account thisAccount = AccountManager.getAccount();
11          System.assert(thisAccount != null);
12          System.assertEquals('Test record', thisAccount.Name);
13      }
14      static Id createTestRecord(){
15          Account accountTest = new Account(
16   Name ='Test record');
17          insert accountTest;
18          Contact contactTest = new Contact(
19   FirstName='John',
20   LastName = 'Doe',
21   AccountId = accountTest.Id
22          );
23          insert contactTest;
24          return accountTest.Id;
25      }
26  }
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);
```

```
    }
}
```

```
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
    ){
48                      Equipment_Maintenance_Item__c item =
    clonedListItem.clone();
49                      item.Maintenance_Request__c = nc.Id;
50                      clonedList.add(item);
51                  }
52              }
53              insert clonedList;
54          }
55      }
56  }
```

```
1  public with sharing class WarehouseCalloutService {
2
3      private static final String WAREHOUSE_URL =
    'https://th-superbadge-apex.herokuapp.com/equipment';
4      public static void runWarehouseEquipmentSync(){
5          Http http = new Http();
6          HttpRequest request = new HttpRequest();
7          request.setEndpoint(WAREHOUSE_URL);
8          request.setMethod('GET');
9          HttpResponse response = http.send(request);
10         List<Product2> warehouseEq = new List<Product2>();
11         if (response.getStatusCode() == 200){
12             List<Object> jsonResponse =
    (List<Object>)JSON.deserializeUntyped(response.getBody());
13             System.debug(response.getBody());
14             for (Object eq : jsonResponse){
15                 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

                System.debug(warehouseEq);
            }
        }
    }
}
```

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

```

```
1       @isTest
2  public with sharing class MaintenanceRequestHelperTest {
3      // createVehicle
4      private static Vehicle__c createVehicle(){
5          Vehicle__c vehicle = new Vehicle__C(name = 'Testing

6          return vehicle;
7      }
8      // createEquipment
9      private static Product2 createEquipment(){
10         product2 equipment = new product2(name = 'Testing

11                                           lifespan_months__c =
   10,
12                                           maintenance_cycle__c =
   10,
13                                           replacement_part__c =
   true);
14         return equipment;
15     }
16     // createMaintenanceRequest
17     private static Case createMaintenanceRequest(id vehicleId, id
   equipmentId){
18         case cse = new case(Type='Repair',
19                             Status='New',
20                             Origin='Web',
21                             Subject='Testing subject',
22                             Equipment__c=equipmentId,
23                             Vehicle__c=vehicleId);
24         return cse;
25     }
26     // createEquipmentMaintenanceItem
27     private static Equipment_Maintenance_Item__c
   createEquipmentMaintenanceItem(id equipmentId,id requestId){
28         Equipment_Maintenance_Item__c equipmentMaintenanceItem =
   new Equipment_Maintenance_Item__c(
29             Equipment__c = equipmentId,
30             Maintenance_Request__c = requestId);
31         return equipmentMaintenanceItem;
32     }
```

```
33      @isTest
34      private static void testPositive(){
35          Vehicle__c vehicle = createVehicle();
36          insert vehicle;
37          id vehicleId = vehicle.Id;
38          Product2 equipment = createEquipment();
39          insert equipment;
40          id equipmentId = equipment.Id;
41          case createdCase =
    createMaintenanceRequest(vehicleId,equipmentId);
42          insert createdCase;
43          Equipment_Maintenance_Item__c equipmentMaintenanceItem =
    createEquipmentMaintenanceItem(equipmentId,createdCase.id);
44          insert equipmentMaintenanceItem;
45          test.startTest();
46          createdCase.status = 'Closed';
47          update createdCase;
48          test.stopTest();
49          Case newCase = [Select id,
50                          subject,
51                          type,
52                          Equipment__c,
53                          Date_Reported__c,
54                          Vehicle__c,
55                          Date_Due__c
56                         from case
57                         where status ='New'];
58          Equipment_Maintenance_Item__c workPart = [select id
59                                                    from
    Equipment_Maintenance_Item__c
60                                                    where
    Maintenance_Request__c =:newCase.Id];
61          list<case> allCase = [select id from case];
62          system.assert(allCase.size() == 2);
63          system.assert(newCase != null);
64          system.assert(newCase.Subject != null);
65          system.assertEquals(newCase.Type, 'Routine Maintenance');
66          SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
67          SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
68          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++){
```

```
101                 vehicleList.add(createVehicle());
102                 equipmentList.add(createEquipment());
103             }
104             insert vehicleList;
105             insert equipmentList;
106             for(integer i = 0; i < 300; i++){
107 
   caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
   equipmentList.get(i).id));
108             }
109             insert caseList;
110             for(integer i = 0; i < 300; i++){
111 
   equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(e

112             }
113             insert equipmentMaintenanceItemList;
114             test.startTest();
115             for(case cs : caseList){
116                 cs.Status = 'Closed';
117                 oldCaseIds.add(cs.Id);
118             }
119             update caseList;
120             test.stopTest();
121             list<case> newCase = [select id
122                                   from case
123                                   where status ='New'];
124 
125             list<Equipment_Maintenance_Item__c> workParts =
   [select id
126 
   from Equipment_Maintenance_Item__c
127 
   where Maintenance_Request__c in: oldCaseIds];
128             system.assert(newCase.size() == 300);
129             list<case> allCase = [select id from case];
130             system.assert(allCase.size() == 600);
131         }
132     }
```

```
 1   @isTest
 2 global class WarehouseCalloutServiceMock implements
   HttpCalloutMock {
 3     global static HttpResponse respond(HttpRequest request){
 4         System.assertEquals('https://th-superbadge-
   ));
 5         System.assertEquals('GET', request.getMethod());
 6
 7         HttpResponse response = new HttpResponse();
 8         response.setHeader('Content-Type', 'application/json');
 9
   response.setBody('[{"_id":"55d66226726b611100aaf741","replacement



10         response.setStatusCode(200);
11         return response;
12     }
13 }
```

```
 1   @isTest
 2
 3 private class WarehouseCalloutServiceTest {
 4     @isTest
 5     static void testWareHouseCallout(){
 6         Test.startTest();
 7         Test.setMock(HTTPCalloutMock.class, new
   WarehouseCalloutServiceMock());
 8         WarehouseCalloutService.runWarehouseEquipmentSync();
 9         Test.stopTest();
10         System.assertEquals(1, [SELECT count() FROM Product2]);
11     }
12 }
```

```
1 @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();
        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);
    }

```

```
20
21      public static void updateCustomSetting(Boolean
    isDataCreated){
22          How_We_Roll_Settings__c   customSetting =
    How_We_Roll_Settings__c.getOrgDefaults();
23          customSetting.Is_Data_Created__c = isDataCreated;
24          upsert customSetting;
25      }
26
27      public static List<Vehicle__c> createVehicles(){
28          List<Vehicle__c> vehicles = new List<Vehicle__c>();
29          vehicles.add(new Vehicle__c(Name = 'Toy Hauler RV',
    Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1,
    Model__c = 'Toy Hauler RV'));
30          vehicles.add(new Vehicle__c(Name = 'Travel Trailer RV',
    Air_Conditioner__c = true, Bathrooms__c = 2, Bedrooms__c = 2,
    Model__c = 'Travel Trailer RV'));
31          vehicles.add(new Vehicle__c(Name = 'Teardrop Camper',
    Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1,
    Model__c = 'Teardrop Camper'));
32          vehicles.add(new Vehicle__c(Name = 'Pop-Up Camper',
    Air_Conditioner__c = true, Bathrooms__c = 1, Bedrooms__c = 1,
    Model__c = 'Pop-Up Camper'));
33          insert vehicles;
34          return vehicles;
35      }
36
37      public static List<Product2> createEquipment(){
38          List<Product2> equipments = new List<Product2>();
39          equipments.add(new Product2(Warehouse_SKU__c =
    '55d66226726b611100aaf741',name = 'Generator 1000 kW',
    Replacement_Part__c = true,Cost__c = 100 ,Maintenance_Cycle__c =
    100));
40          equipments.add(new Product2(name = 'Fuse

    Maintenance_Cycle__c = 30  ));
41          equipments.add(new Product2(name = 'Breaker

    Maintenance_Cycle__c = 15));
42          equipments.add(new Product2(name = 'UPS 20
```

```
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));
63          insert joinRecords;
64          return joinRecords;
65
66      }
67  }
```

```
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          System.assertEquals(true,
    CreateDefaultData.isDataCreated(), 'The custom setting

30
31      }
32  }
```