1.Apex Triggers

Get Started with Apex Triggers

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account account:Trigger.New){
        if(Account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }

}
```

Bulk Apex Triggers

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

    for(Opportunity opp: Trigger.new){
        if(opp.StageName == 'Closed Won'){
            tasklist.add(new Task(Subject = 'Follow Up Test Task', WhatId = opp.Id));
        }
    }
    if(tasklist.size()>0){
        insert tasklist;
    }

}
```

2.Apex Testing

Get Started with Apex Unit Tests

VerifyDate

```
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;
	}

}
```

TestVerifyDate

```
@isTest
private class TestVerifyDate {
    @isTest static void Test_CheckDates_case1(){
        Date D =
VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'),D);
    }

    @isTest static void Test_CheckDates_case2(){
        Date D =
VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'),D);
    }

    @isTest static void Test_DateWithin30Days_case1(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
        System.assertEquals(false, flag);
    }

    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
        System.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
```

```
    }
}
```

4.Test Apex Triggers

RestrictContactByName

```
trigger RestrictContactByName on Contact (before insert, before update) {

        //check contacts prior to insert or update for invalid data
        For (Contact c : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {        //invalidname is invalid
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for
DML');
                }

        }
}
```

TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {
   @istest static void Test_insertupdateContact(){
       Contact cnt = new Contact();
      cnt.LastName = 'INVALIDNAME';

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

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

```
}
```

Create Test Data for Apex Tests

```
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 cnt = new Contact(FirstName = 'Test '+i, LastName = lastname);
         contacts.add(cnt);
      }
      return contacts;

   }
}
```

3.Asynchronous Apex

Use Future Methods

AccountProcessor

```
public class AccountProcessor {
@future
   public static void countContacts(List<Id> accountIds){
      List<Account> accounts = [SELECT Id, (SELECT Id FROM Contacts) FROM Account
WHERE Id IN :accountIds];
         for(Account acc : accounts){
            acc.Number_Of_Contacts__c = acc.Contacts.size();
         }
      update accounts;
   }
}
```

AccountProcessorTest

```
@isTest
```

```
private class AccountProcessorTest {
   @isTest
   private static void countContactsTest(){
      List<Account> accounts = new List<Account>();
      for(Integer i=0; i<300; i++){
         accounts.add(new Account(Name='Test Account' + i));
      }
      insert accounts;

      List<Contact> contacts = new List<Contact>();
      List<Id> accountIds = new List<Id>();
      for(Account acc: accounts){
         contacts.add(new
Contact(FirstName=acc.Name,LastName='TestContact',AccountId=acc.Id));
         accountIds.add(acc.Id);
      }
      insert contacts;

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

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

}
```

Use Batch Apex

LeadProcessor

```
public  class LeadProcessor implements
Database.Batchable<sobject>,Database.Stateful {
```

```
    public Integer recordCount = 0;

    public Database.QueryLocator start(Database.BatchableContext dbc){
        return Database.getQueryLocator([SELECT Id,Name from lEAD]);
    }

    public void execute(Database.BatchableContext dbc,List<Lead> leads){
        for(Lead l : leads){
            l.LeadSource = 'Dreamforce';
        }
        update leads;
        recordCount = recordCount + leads.size();
    }
    public void finish (Database.BatchableContext dbc){
        System.debug('Total records processed '+ recordCount);
    }

}
```

LeadProcessorTeste

```
@isTest
public class LeadProcessorTest {
    @isTest
    private static void testBatchClass(){
        List<Lead> leads = new List<Lead>();
        for(Integer i=0; i<200; i++){
            leads.add(new Lead(LastName='Connock', Company='Salesforce'));
        }
        insert leads;

        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 least 1 lead record not
updated correctly');
    }

}
```

Control Processes with Queueable Apex

AddPrimaryContact

```
public 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;
     }
```

}

AddPrimaryContactTest

```
@isTest
public 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: Incorrect number of Contact
records found');
    }
}
```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor

```
public 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;
   }

}
```

DailyLeadProcessorTest

```
@isTest
public class DailyLeadProcessorTest {

   private static String CRON_EXP = '0 0 0 ? * * *';

   @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 Leads',CRON_EXP, NEW
DailyLeadProcessor());
        Test.stopTest();

        List<Lead> updatedLeads = [SELECT Id, LeadSource FROM Lead WHERE
LeadSource = 'Dreamforce'];
        System.assertEquals(200, updatedLeads.size(), 'ERROR: At least 1 record not
updated correctly');

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

4.Apex Integration Services

Apex REST Callouts

AnimalLocator

```
public class AnimalLocator {
    public static String getAnimalNameById (Integer i) {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+i);
        request.setMethod('GET');
        HttpResponse response = http.send(request);

            Map<String, Object> result = (Map<String,
Object>)JSON.deserializeUntyped(response.getBody());
            Map<String, Object> animal = (Map<String, Object>)result.get('animal');
```

```
        System.debug('name: '+String.valueOf(animal.get('name')));
        return String.valueOf(animal.get('name'));
    }
}
```

AnimalLocatorTest

```
@isTest
private class AnimalLocatorTest {

    @isTest
    static void animalLocatorTest1(){
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String actual = AnimalLocator.getAnimalNameById(5);
        String expected = 'moose';
        System.assertEquals(actual, expected);
    }

}
```

AnimalLocatorMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {

    global HttpResponse respond(HttpRequest request) {
        HttpResponse response = new HttpResponse();
        response.setHeader('contentType', 'application/json');

response.setBody('{"animal":{"id":1,"name":"moose","eats":"plants","says":"bellows"}}');
        response.setStatusCode(200);
        return response;
    }

}
```

Apex SOAP Callouts

ParkLocator

```
public class ParkLocator {

    public static List < String > country(String country){
      ParkService.ParksImplPort prkSvc = new ParkService.ParksImplPort();
      return prkSvc.byCountry(country);
   }

}
```

ParkLocatorTest

```
@isTest
public class ParkLocatorTest {

   @isTest static void testCallout () {
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      String country = 'United States';
      List<String> expectedParks = new List<String>{'Yosemite', 'Sequoia', 'Crater Lake'};

      System.assertEquals(expectedParks,ParkLocator.country(country));
   }
}
```

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){
            parkService.byCountryResponse response_x = new
parkService.byCountryResponse();
            response_x.return_x = new List<String>{'Yosemite','Sequoia','Crater Lake'};
                response.put('response_x', response_x);
        }
    }
```

AsyncParkService

```
public class AsyncParkService {
    public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
        public String[] getValue() {
            ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
            return response.return_x;
        }
    }
    public class AsyncParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public String clientCertName_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
        public AsyncParkService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
              this,
              request_x,
```

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

Apex Web Services

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, FirstName, LastName FROM
Contacts)FROM Account WHERE Id = :accountId];
                  return result;

   }

}
```

AccountManagerTest

```
@isTest
private class AccountManagerTest {
   @isTest
   static void testGetAccount() {

Account a = new Account (Name='TestAccount');
       insert a;
       Contact c = new Contact (AccountId=a.Id, FirstName='Test', LastName='Test');
       insert c;
       RestRequest request = new RestRequest();
       request.requestUri
='https://yourInstance.salesforce.com/services/apexrest/Accounts/'+a.id+'/contacts';
       request.httpMethod = 'GET';
       RestContext.request = request;

       Account myAcct = AccountManager.getAccount();
       System.assert(myAcct != null);
       System.assertEquals('TestAccount', myAcct.Name);
       }
   }
```

5.Apex Specialist

Automate record creation

MaintenanceRequest

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

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);
            }
         }
      }

      //When an existing maintenance request of type Repair or Routine Maintenance is
closed,
      //create a new maintenance request for a future routine checkup.
      if (!validIds.isEmpty()){
         Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,
                                    (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                    FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

         //calculate the maintenance request due dates by using the maintenance cycle
defined on the related equipment records.
         AggregateResult[] results = [SELECT Maintenance_Request__c,
                        MIN(Equipment__r.Maintenance_Cycle__c)cycle
                        FROM Equipment_Maintenance_Item__c
                        WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

         for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
         }

         List<Case> newCases = new List<Case>();
```

```
        for(Case cc : closedCases.values()){
            Case nc = new Case (
                ParentId = cc.Id,
                Status = 'New',
                Subject = 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle__c = cc.Vehicle__c,
                Equipment__c =cc.Equipment__c,
                Origin = 'Web',
                Date_Reported__c = Date.Today()
            );

            //If multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to today's
date.
            //If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
            //} else {
            //    nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
            //}

            newCases.add(nc);
        }

        insert newCases;

        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
```

```
            }
            insert clonedList;
        }
    }
}
```

Synchronize Salesforce data with an external system

WarehouseCalloutService

```
public with sharing class WarehouseCalloutService implements Queueable,
Database.AllowsCallouts {
    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.setMethod('GET');
        request.setEndpoint(WAREHOUSE_URL);
        HttpResponse response = http.send(request);
        if(response.getStatusCode() == 200) {
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            system.debug('~~ '+jsonResponse);
            List<Product2> productList = new List<Product2>();
            for(Object ob : jsonResponse) {
                Map<String,Object> mapJson = (Map<String,Object>)ob;
                Product2 pr = new Product2();
                pr.Replacement_Part__c = (Boolean)mapJson.get('replacement');
                pr.Name = (String)mapJson.get('name');
                pr.Maintenance_Cycle__c = (Integer)mapJson.get('maintenanceperiod');
                pr.Lifespan_Months__c = (Integer)mapJson.get('lifespan');
                pr.Cost__c = (Decimal) mapJson.get('lifespan');
                pr.Warehouse_SKU__c = (String)mapJson.get('sku');
                pr.Current_Inventory__c = (Double) mapJson.get('quantity');
                productList.add(pr);
            }
```

```
            if(productList.size()>0)
                upsert productList;
        }
    }
    public static void execute(QueueableContext context){
        runWarehouseEquipmentSync();
    }
}
```

Schedule synchronization

WarehouseSyncSchedule

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

Test automation logic

MaintenanceRequest

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

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);
            }
         }
      }

      //When an existing maintenance request of type Repair or Routine Maintenance is
closed,
      //create a new maintenance request for a future routine checkup.
      if (!validIds.isEmpty()){
         Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,
                                        (SELECT Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
                                        FROM Case WHERE Id IN :validIds]);
         Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

         //calculate the maintenance request due dates by using the maintenance cycle
defined on the related equipment records.
         AggregateResult[] results = [SELECT Maintenance_Request__c,
                              MIN(Equipment__r.Maintenance_Cycle__c)cycle
                              FROM Equipment_Maintenance_Item__c
                              WHERE Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];

         for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
         }

         List<Case> newCases = new List<Case>();
         for(Case cc : closedCases.values()){
```

```
            Case nc = new Case (
               ParentId = cc.Id,
               Status = 'New',
               Subject = 'Routine Maintenance',
               Type = 'Routine Maintenance',
               Vehicle__c = cc.Vehicle__c,
               Equipment__c =cc.Equipment__c,
               Origin = 'Web',
               Date_Reported__c = Date.Today()
            );

            //If multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to today's
date.
            //If (maintenanceCycles.containskey(cc.Id)){
               nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
            //} else {
            //    nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
            //}

            newCases.add(nc);
         }

         insert newCases;

         List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
         for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
               Equipment_Maintenance_Item__c item = clonedListItem.clone();
               item.Maintenance_Request__c = nc.Id;
               clonedList.add(item);
            }
         }
```

```
            insert clonedList;
        }
    }
}
```

MaintenanceRequestHelperTest

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }

    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    // createMaintenanceRequest
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);
        return cse;
    }
```

```
    // createEquipmentMaintenanceItem
    private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id
equipmentId,id requestId){
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
            Maintenance_Request__c = requestId);
        return equipmentMaintenanceItem;
    }

    @isTest
    private static void testPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        Product2 equipment = createEquipment();
        insert equipment;
        id equipmentId = equipment.Id;

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

        Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
        insert equipmentMaintenanceItem;

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

        Case newCase = [Select id,
                        subject,
                        type,
                        Equipment__c,
                        Date_Reported__c,
```

```
                Vehicle__c,
                Date_Due__c
               from case
               where status ='New'];

    Equipment_Maintenance_Item__c workPart = [select id
                                from Equipment_Maintenance_Item__c
                                where Maintenance_Request__c =:newCase.Id];
    list<case> allCase = [select id from case];
    system.assert(allCase.size() == 2);

    system.assert(newCase != null);
    system.assert(newCase.Subject != null);
    system.assertEquals(newCase.Type, 'Routine Maintenance');
    SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
  }

  @isTest
  private static void testNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    product2 equipment = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;

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

    Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
    insert workP;

    test.startTest();
```

```
        createdCase.Status = 'Working';
        update createdCase;
        test.stopTest();

        list<case> allCase = [select id from case];

        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c = :createdCase.Id];

        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }

    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();

        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEquipment());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
            caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
        }
        insert caseList;

        for(integer i = 0; i < 300; i++){
```

```
equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(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);
   }
}
```

Test callout logic

WarehouseCalloutService

```
public with sharing class WarehouseCalloutService implements Queueable,
Database.AllowsCallouts {
  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.setMethod('GET');
    request.setEndpoint(WAREHOUSE_URL);
    HttpResponse response = http.send(request);
    if(response.getStatusCode() == 200) {
      List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
      system.debug('~~ '+jsonResponse);
      List<Product2> productList = new List<Product2>();
      for(Object ob : jsonResponse) {
        Map<String,Object> mapJson = (Map<String,Object>)ob;
        Product2 pr = new Product2();
        pr.Replacement_Part__c = (Boolean)mapJson.get('replacement');
        pr.Name = (String)mapJson.get('name');
            pr.Maintenance_Cycle__c = (Integer)mapJson.get('maintenanceperiod');
        pr.Lifespan_Months__c = (Integer)mapJson.get('lifespan');
        pr.Cost__c = (Decimal) mapJson.get('lifespan');
        pr.Warehouse_SKU__c = (String)mapJson.get('sku');
        pr.Current_Inventory__c = (Double) mapJson.get('quantity');
        productList.add(pr);
      }
      if(productList.size()>0)
        upsert productList;
    }
  }
  public static void execute(QueueableContext context){
    runWarehouseEquipmentSync();
  }
}
```

WarehouseCalloutServiceTest

```
@IsTest
private class WarehouseCalloutServiceTest {
   // implement your mock callout test here
         @isTest
   static void testWarehouseCallout() {
      test.startTest();
      test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
      WarehouseCalloutService.execute(null);
      test.stopTest();

      List<Product2> product2List = new List<Product2>();
      product2List = [SELECT ProductCode FROM Product2];

      System.assertEquals(3, product2List.size());
      System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
      System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
      System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
   }
}
```

WarehouseCalloutServiceMock

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
   // implement http mock callout
   global static HttpResponse respond(HttpRequest request) {

      HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5
```

,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}

test scheduling logic

WarehouseSyncSchedule

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WarehouseSyncScheduleTest

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    // implement scheduled code here
    //
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to Schedule to test',
scheduleTime, new WarehouseSyncSchedule());
```

```
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}
```