//Apex Triggers

//Get Started with Apex Triggers

```
//ACCOUNT ADDRESS TRIGGER
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

```
//CLOSED OPPORTUNITY 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 Task', WhatId = opp.Id));
            }
        }
    if(tasklist.size()>0) {
```

```
        insert tasklist;
    }
}
```

//Apex Testing

//Get Started with Apex Unit Tests

```
//VERIFY DATE
public class VerifyDate {


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


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

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

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

}

//VERIFY DATE TEST
@isTest
private class TestVerifyDate {

    //testing that if date2 is within 30 days of date1, should return date 2
    @isTest static void testDate2within30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 11);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 04, 11);
        System.assertEquals(testDate,resultDate);
```

```
    }

    //testing that date2 is before date1. Should return "false"
    @isTest static void testDate2beforeDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 02, 11);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 02, 11);
        System.assertNotEquals(testDate, resultDate);
    }

    //Test date2 is outside 30 days of date1. Should return end of month.
    @isTest static void testDate2outside30daysofDate1() {
        Date date1 = date.newInstance(2018, 03, 20);
        Date date2 = date.newInstance(2018, 04, 25);
        Date resultDate = VerifyDate.CheckDates(date1,date2);
        Date testDate = Date.newInstance(2018, 03, 31);
        System.assertEquals(testDate,resultDate);
    }
}
```

//Test Apex Triggers

```
//RESTRICT CONTACT BY NAME
trigger RestrictContactByName on Contact (before insert, before update) {
    For (Contact c : Trigger.New) {
        if(c.LastName == 'INVALIDNAME') { //invalidname is invalid
```

```
            c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
        }
    }
}
//TEST TESTRICT BY CONTACT NAME
@isTest
public class TestRestrictContactByName{
    @isTest static void testRestrictContactByName () {
        Contact c = new Contact(LastName='INVALIDNAME');
        try{
          insert c;
        }
        catch(DMLException e){
              System.assert(e.getMessage().contains('The Last Name "'+c.LastName+'" is not
allowed for DML'));
           }
        }
    }
```

//Create Test Data for Apex Tests

```
//RANDOM CONTACT FACTORY


public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer numContacts, String lastName)
{
        List<Contact> contacts = new List<Contact>();
```

```
        for (Integer i=0; i<numContacts; i++) {
            Contact c = new Contact(FirstName = 'Name ' + i,
                                    LastName = lastName);
            contacts.add(c);
        }
        return contacts;
    }
}
```

//Asynchronous Apex

//Use Future Methods

```
//ACCOUNT PROCESSOR
public class AccountProcessor {

    @future
    public static void countContacts(List<Id> accountId_lst) {

        Map<Id,Integer> account_cno = new Map<Id,Integer>();
        List<account> account_lst_all = new List<account>([select id, (select id from contacts)
from account]);
        for(account a:account_lst_all) {
            account_cno.put(a.id,a.contacts.size()); //populate the map

        }

        List<account> account_lst = new List<account>(); // list of account that we will upsert
```

```
        for(Id accountId : accountId_lst) {
            if(account_cno.containsKey(accountId)) {
                account acc = new account();
                acc.Id = accountId;
                acc.Number_of_Contacts__c = account_cno.get(accountId);
                account_lst.add(acc);
            }

        }
        upsert account_lst;
    }

}

//ACCOUNT PROCESSOR TEST
@isTest
public class AccountProcessorTest {

    @isTest
    public static void testFunc() {
        account acc = new account();
        acc.name = 'MATW INC';
        insert acc;

        contact con = new contact();
        con.lastname = 'Mann1';
```

```
        con.AccountId = acc.Id;
        insert con;
        contact con1 = new contact();
        con1.lastname = 'Mann2';
        con1.AccountId = acc.Id;
        insert con1;



        List<Id> acc_list = new List<Id>();
        acc_list.add(acc.Id);
        Test.startTest();
          AccountProcessor.countContacts(acc_list);
        Test.stopTest();
        List<account> acc1 = new List<account>([select Number_of_Contacts__c from account
where id = :acc.id]);
        system.assertEquals(2,acc1[0].Number_of_Contacts__c);
    }
}
```

//Use Batch Apex

```
//LEAD PROCESSOR
public class LeadProcessor implements Database.Batchable<sObject>, Database.Stateful {
    /**
     * コンストラクタ
     */
    public LeadProcessor() {
```

```
}

/**
 * Start
 */
public Database.QueryLocator start(Database.BatchableContext BC) {
    String query = 'SELECT Id FROM Lead';
    return Database.getQueryLocator (query);
}

/**
 * Execute
 */
public void execute(Database.BatchableContext BC, List<Lead> leads) {
    for (Lead l : leads) {
        l.LeadSource  = 'Dreamforce';
    }
    update leads;
}

/**
 * Finish
 */
public void finish(Database.BatchableContext BC) {

}
```

```
}
//LEAD PROCESSOR TEST
@isTest
private class LeadProcessorTest {

    private static User testAdminUser = new User(Id = UserInfo.getUserId());

    static testMethod void LeadProcessorTest() {

        System.runAs(testAdminUser) {

            List<Lead> leads = new List<Lead>();
            for (Integer i = 0; i < 200; i++) {
                leads.add(new Lead(LastName = 'Yoshikawa', Company = 'T.Yoshikawa Labs'));
            }
            insert leads;
            System.assertEquals(leads.size(), 200);

            Test.startTest();

            LeadProcessor batchable = new LeadProcessor();
            Database.executeBatch(batchable);

            Test.stopTest();

            List<Lead> results = [SELECT Id,LeadSource FROM Lead];
            for (Lead l : results) {
```

```
            System.assertEquals(l.LeadSource, 'Dreamforce');
        }
        System.assertEquals(results.size(), 200);
    }
  }
}
```

//Control Processes with Queueable Apex

```
//ADD PRIMARY CONTACT
public class AddPrimaryContact implements Queueable {
  public contact c;
  public String state;

  public AddPrimaryContact(Contact c, String state) {
    this.c = c;
    this.state = state;
  }

  public void execute(QueueableContext qc) {
    system.debug('this.c = '+this.c+' this.state = '+this.state);
    List<Account> acc_lst = new List<account>([select id, name, BillingState from account
where account.BillingState = :this.state limit 200]);
    List<contact> c_lst = new List<contact>();
    for(account a: acc_lst) {
      contact c = new contact();
      c = this.c.clone(false, false, false, false);
```

```
            c.AccountId = a.Id;
            c_lst.add(c);
        }
        insert c_lst;
    }

}

//ADD PRIMARY CONTACT TEST
@IsTest
public class AddPrimaryContactTest {

    @IsTest
    public static void testing() {
        List<account> acc_lst = new List<account>();
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(i),billingstate='NY');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(50+i),billingstate='CA');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        insert acc_lst;
        Test.startTest();
```

```
        contact c = new contact(lastname='alex');
        AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
        system.debug('apc = '+apc);
        System.enqueueJob(apc);
        Test.stopTest();
        List<contact> c_lst = new List<contact>([select id from contact]);
        Integer size = c_lst.size();
        system.assertEquals(50, size);
    }

}
```

//Schedule Jobs Using the Apex Scheduler

```
//DAILY LEAD PROCESSOR
public without sharing class DailyLeadProcessor implements Schedulable {

    public void execute(SchedulableContext ctx) {
        //System.debug('Context ' + ctx.getTriggerId()); // Returns the ID of the CronTrigger scheduled job

          // Get 200 Lead records and modify the LeadSource field
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = null LIMIT 200];
        for ( Lead l : leads) {
            l.LeadSource = 'Dreamforce';
        }
```

```
        // Update the modified records
        update leads;
    }
}
//DAILY LEAD PROCESSOR TEST
@isTest
private class DailyLeadProcessorTest {

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

    @isTest
    private static void testSchedulableClass() {

        // Load test data
        List<Lead> leads = new List<Lead>();
        for (Integer i=0; i<500; i++) {
            if ( i < 250 ) {
                leads.add(new Lead(LastName='Connock', Company='Salesforce'));
            } else {
                leads.add(new Lead(LastName='Connock', Company='Salesforce',
LeadSource='Other'));
            }
        }
        insert leads;

        // Perform the test
```

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

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

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

        // Not sure this works for all timezones
          //Datetime midnight = Datetime.newInstance(Date.today(), Time.newInstance(0,0,0,0));
        //System.assertEquals(midnight.addHours(24), cts[0].NextFireTime, 'ERROR: Not scheduled for Midnight local time');
    }
}
```

//Apex REST Callouts

```
//AnimalLocator
public class AnimalLocator {
    public class cls_animal {
        public Integer id;
```

```
        public String name;
        public String eats;
        public String says;
    }
public class JSONOutput{
    public cls_animal animal;

    //public JSONOutput parse(String json){
    //return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
    //}
}

   public static String getAnimalNameById (Integer id) {
       Http http = new Http();
       HttpRequest request = new HttpRequest();
       request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
       //request.setHeader('id', String.valueof(id)); -- cannot be used in this challenge :)
       request.setMethod('GET');
       HttpResponse response = http.send(request);
       system.debug('response: ' + response.getBody());
       //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
       jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(), jsonOutput.class);
       //Object results = (Object) map_results.get('animal');
         system.debug('results= ' + results.animal.name);
       return(results.animal.name);
   }
```

```
}

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

}

//ANIMAL LOCATOR MOCK
@IsTest
global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPresponse respond(HTTPrequest request) {
        Httpresponse response = new Httpresponse();
        response.setStatusCode(200);
        //-- directly output the JSON, instead of creating a logic
        //response.setHeader('key, value)
        //Integer id = Integer.valueof(request.getHeader('id'));
        //Integer id = 1;
```

```
        //List<String> lst_body = new List<String> {'majestic badger', 'fluffy bunny'};
        //system.debug('animal return value: ' + lst_body[id]);
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        return response;
    }

}
```

//Apex SOAP Callouts

```
//PARK SERVICE
//Generated by wsdl2apex

public class ParkService {
    public class byCountryResponse {
        public String[] return_x;
        private String[] return_x_type_info = new String[]{'return','http://parks.services/',null,'0','-1','false'};
        private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
        private String[] field_order_type_info = new String[]{'return_x'};
    }
    public class byCountry {
        public String arg0;
        private String[] arg0_type_info = new String[]{'arg0','http://parks.services/',null,'0','1','false'};
        private String[] apex_schema_type_info = new String[]{'http://parks.services/','false','false'};
```

```
        private String[] field_order_type_info = new String[]{'arg0'};
    }
    public class ParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public Map<String,String> outputHttpHeaders_x;
        public String clientCertName_x;
        public String clientCert_x;
        public String clientCertPasswd_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
        public String[] byCountry(String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            ParkService.byCountryResponse response_x;
            Map<String, ParkService.byCountryResponse> response_map_x = new Map<String,
ParkService.byCountryResponse>();
            response_map_x.put('response_x', response_x);
            WebServiceCallout.invoke(
              this,
              request_x,
              response_map_x,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
```

```
            'byCountryResponse',
            'ParkService.byCountryResponse'}
          );
          response_x = response_map_x.get('response_x');
          return response_x.return_x;
      }
   }
}

//PARK LOCATOR
public class ParkLocator {
   public static String[] country(String country){
      ParkService.ParksImplPort parks = new ParkService.ParksImplPort();
      String[] parksname = parks.byCountry(country);
      return parksname;
   }
}
//PARK LOCATOR TEST
@isTest
private class ParkLocatorTest{
   @isTest
   static void testParkLocator() {
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      String[] arrayOfParks = ParkLocator.country('India');

      System.assertEquals('Park1', arrayOfParks[0]);
   }
```

```
}

//PARK SERVICE MOCK
@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();
        List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;

        response.put('response_x', response_x);
    }
}
```

//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.substring(request.requestURI.lastIndexOf('/')-18,
          request.requestURI.lastIndexOf('/'));
        List<Account> a = [select id, name, (select id, name from contacts) from account where id
= :accountId];
        List<contact> co = [select id, name from contact where account.id = :accountId];
        system.debug('** a[0]= '+ a[0]);
        return a[0];


    }


}
//AccountManagerTest
@istest
public class AccountManagerTest {


@isTest static void testGetAccount() {
    Id recordId = createTestRecord();
    // Set up a test request
    RestRequest request = new RestRequest();
```

```
    request.requestUri =
        'https://resourceful-badger-76636-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/contacts'
        + recordId;
    request.httpMethod = 'GET';
    RestContext.request = request;
    // Call the method to test
    Account thisAcc = AccountManager.getAccount();
    // Verify results
    System.assert(thisAcc != null);
    System.assertEquals('Test record', thisAcc.Name);
}

// Helper method
static Id createTestRecord() {
    // Create test record
    Account accTest = new Account(
        Name='Test record');
    insert accTest;
    return accTest.Id;
}
}
```

//Apex Specialist

```
//CHALLENGE 1
//MAINTENANCE REQUEST HELPER
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()){

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

         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]);

         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'));
```

```
    }

        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> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
```

```
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);

            }
        }
        insert ClonedWPs;
    }
  }
}

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

//CHALLENGE 2
//WAREHOUSECALLOUTSERVICES
public with sharing class WarehouseCalloutService {

  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
```

```
    //@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 = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
```

```
            myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
            warehouseEq.add(myEq);
        }

        if (warehouseEq.size() > 0){
            upsert warehouseEq;
            System.debug('Your equipment was synced with the warehouse one');
            System.debug(warehouseEq);
        }

    }
  }
}


//CHALLENGE 3
//WAREHOUSESYNCSCHEDULE
global class WarehouseSyncSchedule implements Schedulable {
   global void execute(SchedulableContext ctx) {

      WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}

//CHALLENGE 4
//MAINTENANCEREQUESTHELPERTEST
@istest
```

```
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW = 'New';
    private static final string WORKING = 'Working';
    private static final string CLOSED = 'Closed';
    private static final string REPAIR = 'Repair';
    private static final string REQUEST_ORIGIN = 'Web';
    private static final string REQUEST_TYPE = 'Routine Maintenance';
    private static final string REQUEST_SUBJECT = 'Testing subject';

    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    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];


        system.assert(workPart != null);
        system.assert(newReq.Subject != null);
        system.assertEquals(newReq.Type, REQUEST_TYPE);
        SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
    }
```

```
@istest
private static void testMaintenanceRequestNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

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

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

    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
    insert workP;

    test.startTest();
    emptyReq.Status = WORKING;
    update emptyReq;
    test.stopTest();

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

    Equipment_Maintenance_Item__c workPart = [select id
                                              from Equipment_Maintenance_Item__c
                                              where Maintenance_Request__c = :emptyReq.Id];
```

```
        system.assert(workPart != null);
        system.assert(allRequest.size() == 1);
    }

    @istest
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

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

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

```
        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in: oldRequestIds];

        system.assert(allRequests.size() == 300);
    }
}

//MAINTENCEREQUESTHELPER
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()){

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

         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]);

         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'));
```

```
    }

        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> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
```

```
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);

            }
        }
        insert ClonedWPs;
    }
  }
}

//MAINTENANCEREQUEST
trigger MaintenanceRequest on Case (before update, after update) {
  if(Trigger.isUpdate && Trigger.isAfter){
    MaintenanceRequestHelper.updateWorkOrders(Trigger.New, Trigger.OldMap);
  }
}
//CHALLENGE 5
//WAREHOUSECALLOUTSERVICE
public with sharing class WarehouseCalloutService {

  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

  //@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 = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
```

```
            warehouseEq.add(myEq);
        }

        if (warehouseEq.size() > 0){
            upsert warehouseEq;
            System.debug('Your equipment was synced with the warehouse one');
            System.debug(warehouseEq);
        }

    }
  }
}

//WAREHOUSECALLOUTSERVICETEST
@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]);
  }
}
```

```
//WAREHOUSECALLOUTSERVICEMOCK
@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":false,"quantity":5,"nam
e":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}

//CHALLENGE 6
//WAREHOUSESYNCSCHEDULE
global class WarehouseSyncSchedule implements Schedulable {
    global void execute(SchedulableContext ctx) {
```

```
        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
//WAREHOUSESYNCSCHEDULETEST

@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 to Test', scheduleTime, new
WarehouseSyncSchedule());
        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 ');



    }
}
```