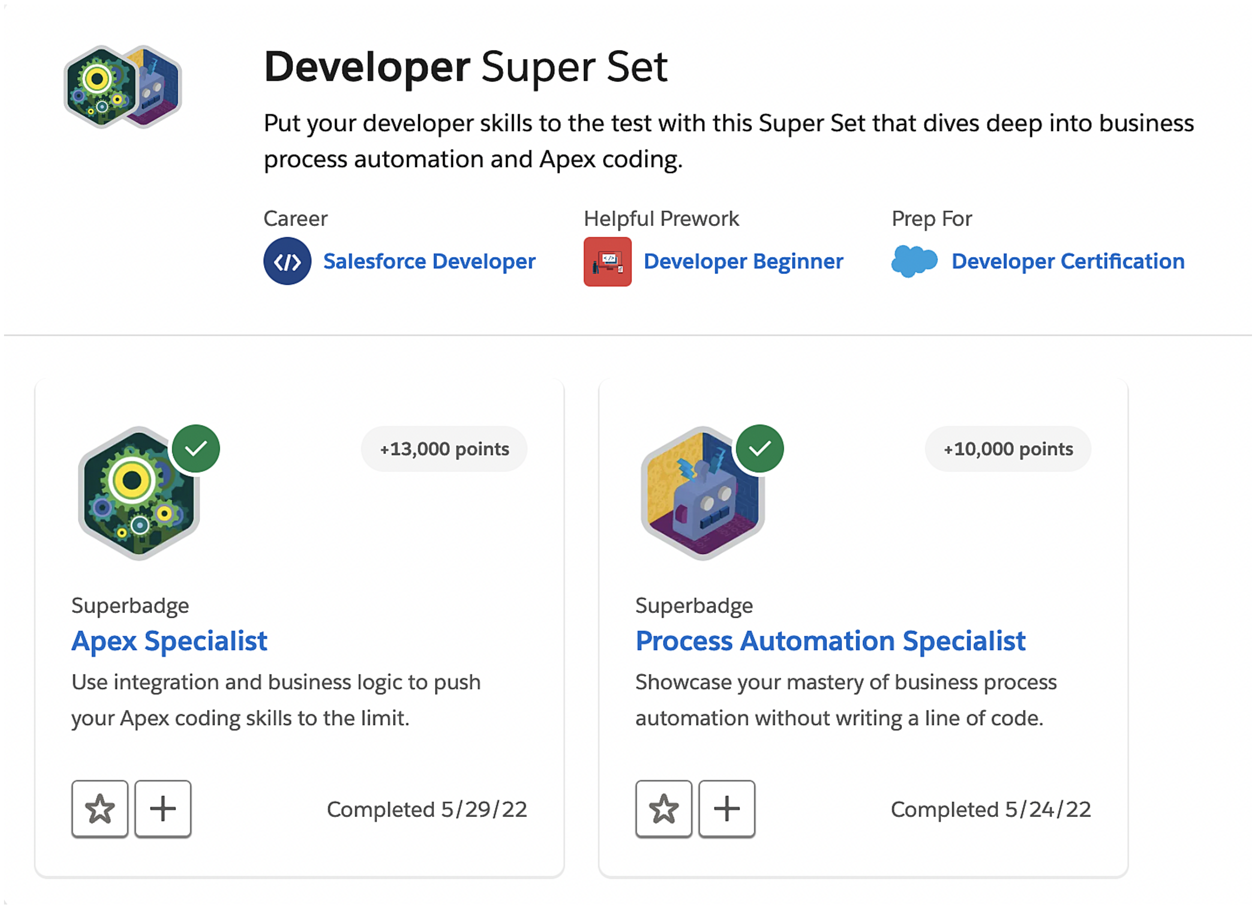
ACKNOWLEDGEMENT

I would like to thank Salesforce and SmartInterz for providing this opportunity.
I would like to thank my mentors, friends and professors who helped me achieve this developer superset

ABSTRACT



Developer Super Set

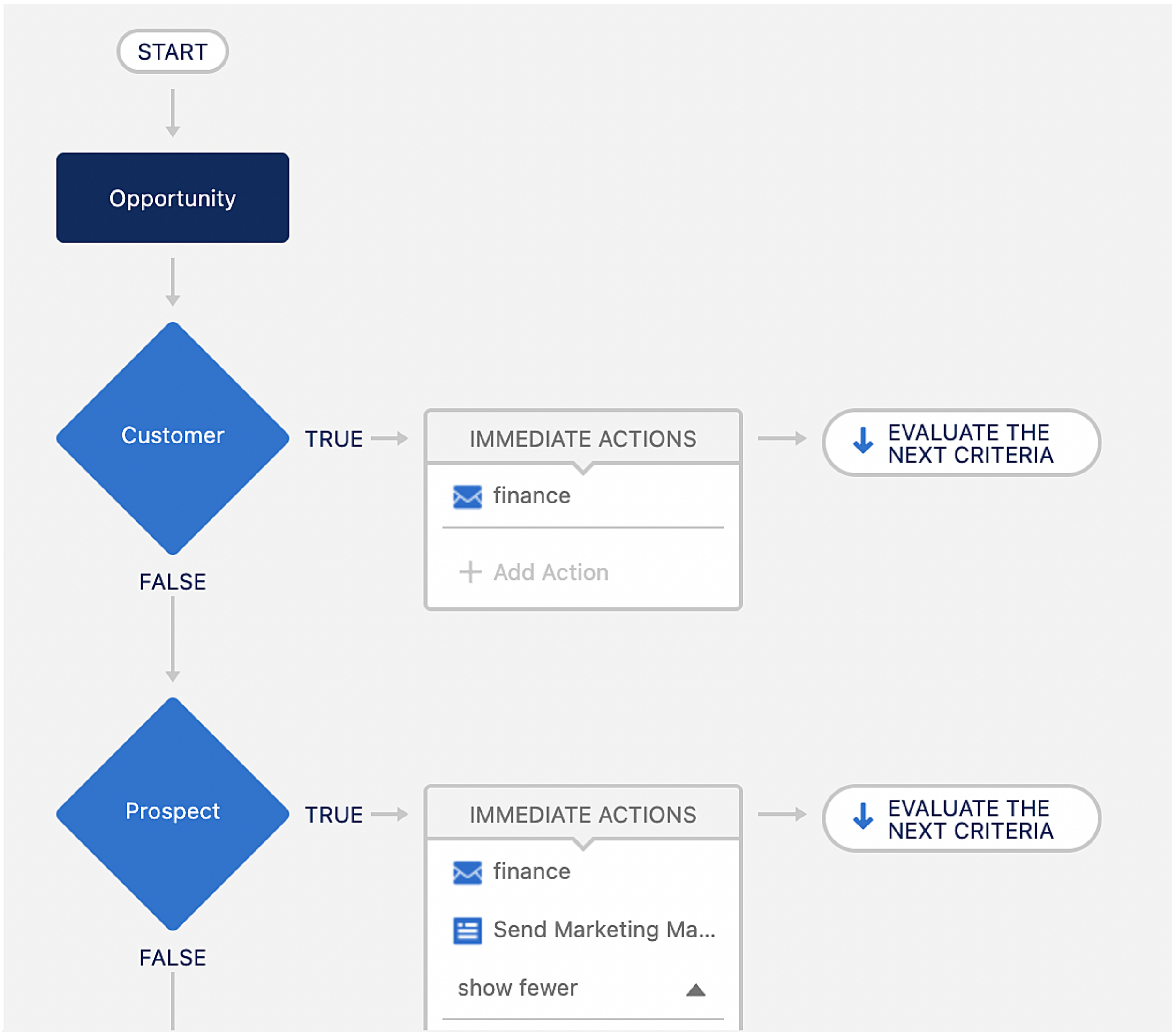
Put your developer skills to the test with this Super Set that dives deep into business process automation and Apex coding.

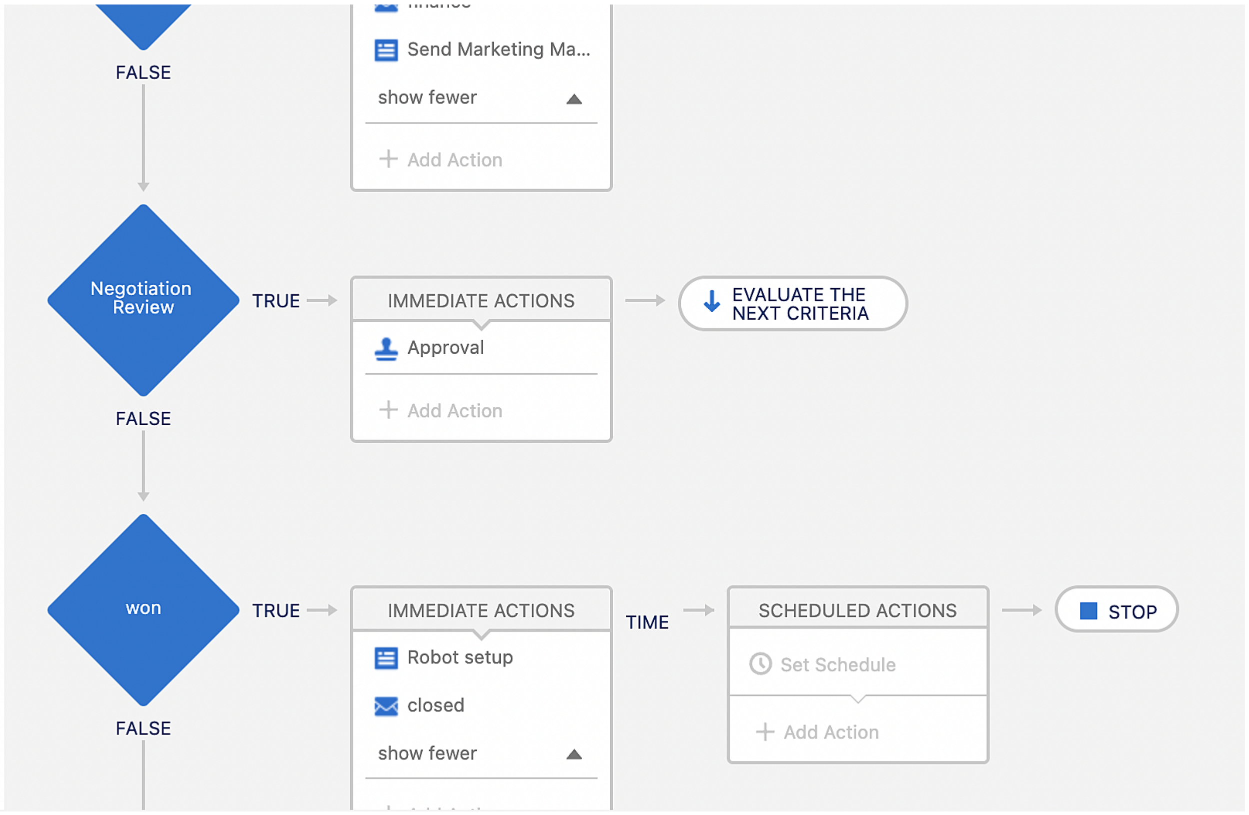
Career: Salesforce Developer
Helpful Prework: Developer Beginner
Prep For: Developer Certification

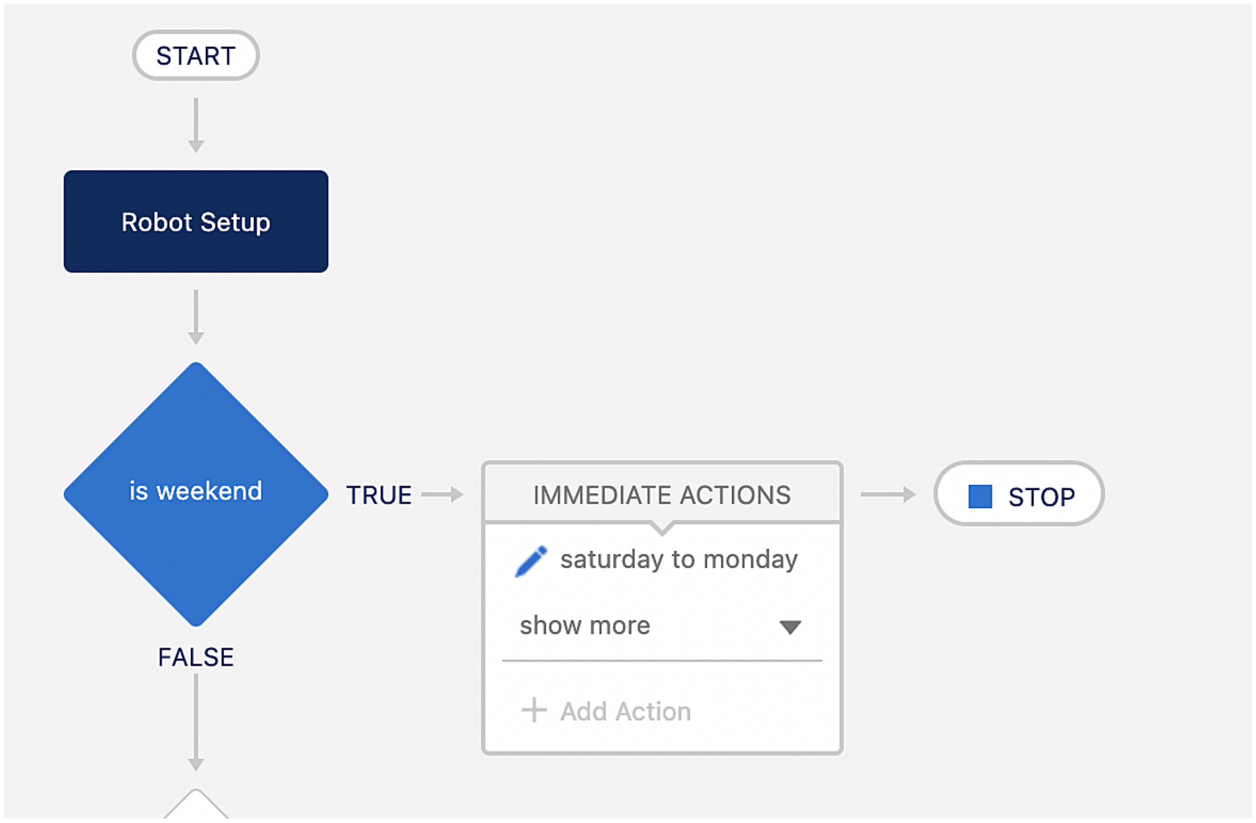
Superbadge
Apex Specialist
+13,000 points
Use integration and business logic to push your Apex coding skills to the limit.
Completed 5/29/22

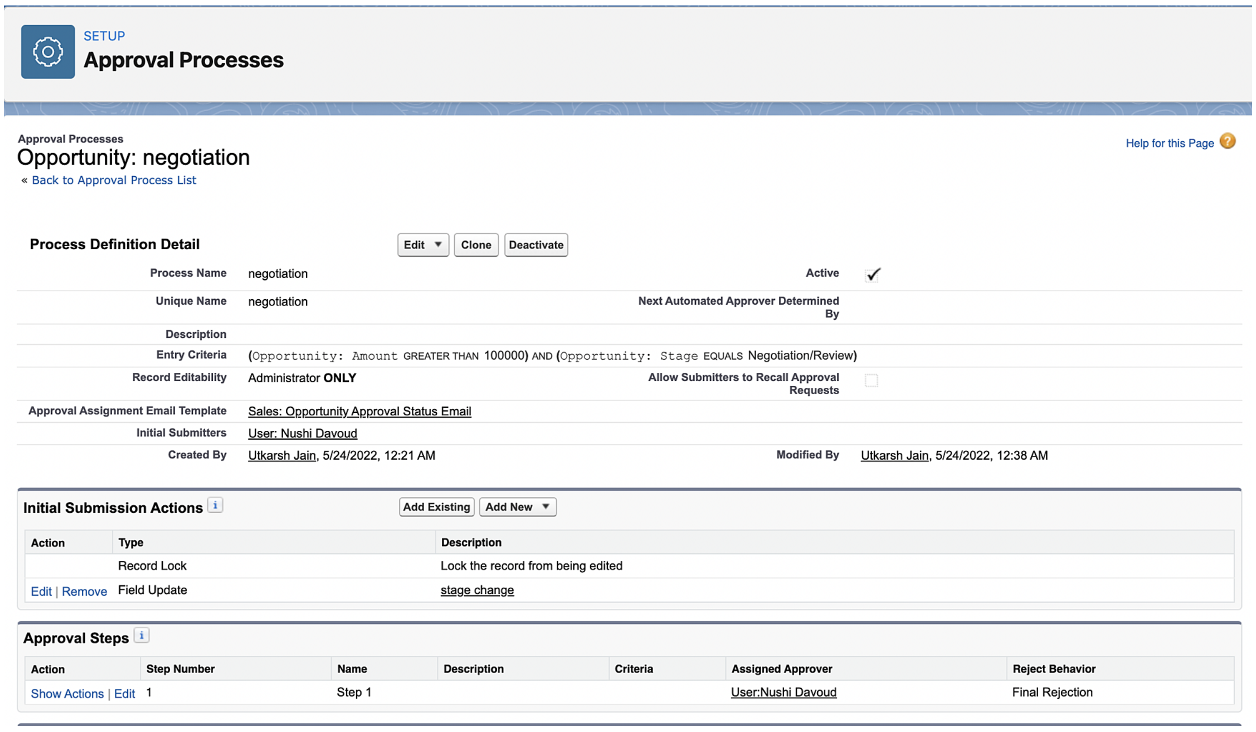
Superbadge
Process Automation Specialist
+10,000 points
Showcase your mastery of business process automation without writing a line of code.
Completed 5/24/22

Process Automation Specialist :




Send Marketing Ma...
show fewer
Add Action
FALSE
Negotiation Review
TRUE
IMMEDIATE ACTIONS
Approval
Add Action
EVALUATE THE NEXT CRITERIA
FALSE
won
TRUE
IMMEDIATE ACTIONS
Robot setup
closed
show fewer
TIME
SCHEDULED ACTIONS
Set Schedule
Add Action
STOP
FALSE


START
Robot Setup
is weekend
TRUE
IMMEDIATE ACTIONS
saturday to monday
show more
Add Action
STOP
FALSE


SETUP
Approval Processes

Approval Processes
Opportunity: negotiation
« Back to Approval Process List

Help for this Page

Process Definition Detail Edit | Clone | Deactivate

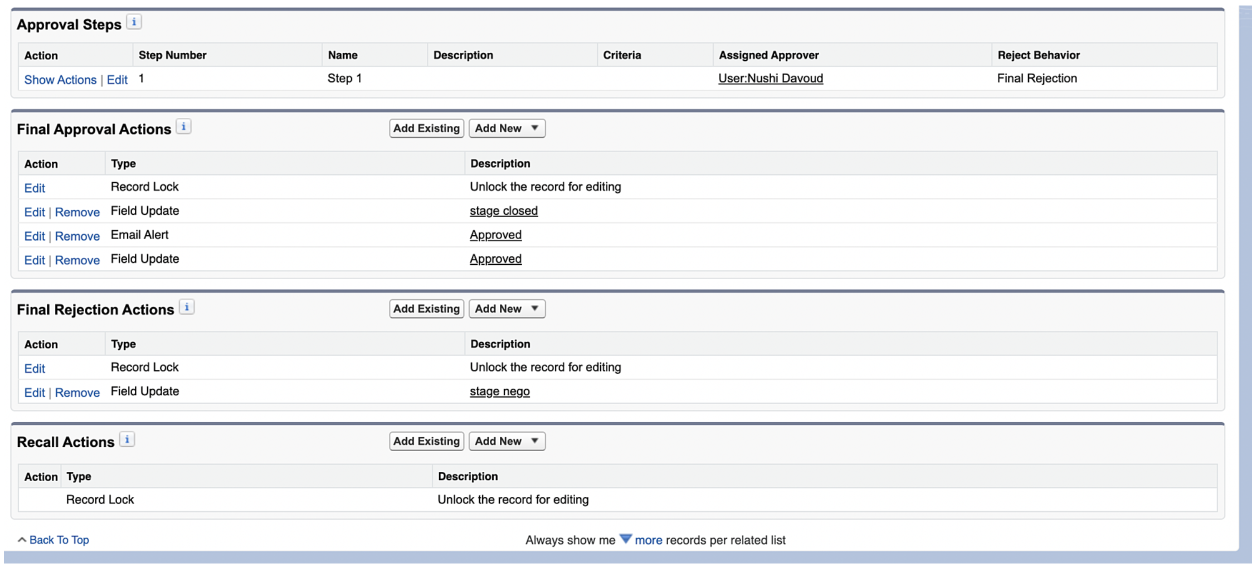
Process Name	negotiation	Active	✓
Unique Name	negotiation	Next Automated Approver Determined By	
Description			
Entry Criteria	(Opportunity: Amount GREATER THAN 100000) AND (Opportunity: Stage EQUALS Negotiation/Review)		
Record Editability	Administrator ONLY	Allow Submitters to Recall Approval Requests	☐
Approval Assignment Email Template	Sales: Opportunity Approval Status Email		
Initial Submitters	User: Nushi Davoud		
Created By	Utkarsh Jain, 5/24/2022, 12:21 AM	Modified By	Utkarsh Jain, 5/24/2022, 12:38 AM

Initial Submission Actions Add Existing | Add New

Action	Type	Description
	Record Lock	Lock the record from being edited
Edit | Remove	Field Update	stage change

Approval Steps

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Show Actions | Edit	1	Step 1			User:Nushi Davoud	Final Rejection


Approval Steps

Action	Step Number	Name	Description	Criteria	Assigned Approver	Reject Behavior
Show Actions | Edit	1	Step 1			User:Nushi Davoud	Final Rejection

Final Approval Actions Add Existing | Add New

Action	Type	Description
Edit	Record Lock	Unlock the record for editing
Edit | Remove	Field Update	stage closed
Edit | Remove	Email Alert	Approved
Edit | Remove	Field Update	Approved

Final Rejection Actions Add Existing | Add New

Action	Type	Description
Edit	Record Lock	Unlock the record for editing
Edit | Remove	Field Update	stage nego

Recall Actions Add Existing | Add New

Action	Type	Description
	Record Lock	Unlock the record for editing

Back To Top
Always show me more records per related list

Apex specialist:

```
trigger MaintenanceRequest on Case (before update, after update) {
    // ToDo: Call MaintenanceRequestHelper.updateWorkOrders
    //
    if(Trigger.isUpdate && Trigger.isAfter){
MaintenanceRequestHelper.updateWorkOrders(Trigger.new,Trigger.Old

    }
}
```

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

                    Ids.add(c.Id);
                }
            }
        }
        if (!Ids.isEmpty()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, ProductId, Product.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
                                                          FROM Case WHERE Id IN :Ids]);
            Map<Id,Decimal> mCycles = 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 :Ids GROUP BY Maintenance_Request__c];
        for (AggregateResult ar : results){
            mCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
        }
            for(Case cc : closedCases.values()){
                Case nc = new Case (
                    ParentId = cc.Id,
                   Status = 'New',
                    Subject = 'Routine Maintenance',
                    Date_Reported__c = Date.Today(),
                    Type = 'Routine Maintenance',
                    Vehicle__c = cc.Vehicle__c,
                    ProductId =cc.ProductId,
                    Origin = 'Web'
                );
                If (mCycles.containskey(cc.Id)){
                    nc.Date_Due__c = Date.today().addDays((Integer) mCycles.get(cc.Id));
                }
                newCases.add(nc);
            }
           insert newCases;
           List<Equipment_Maintenance_Item__c> ans = new List<Equipment_Maintenance_Item__c>();
           for (Case nc : newCases){
                for (Equipment_Maintenance_Item__c wp : closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                    Equipment_Maintenance_Item__c wpClone = wp.clone();
                    wpClone.Maintenance_Request__c = nc.Id;
                    ans.add(wpClone);
                }
            }
            insert ans;
        }
    }
}
```

```
@isTest
private with sharing class MaintenanceRequestHelperTest {
  private static Vehicle__c createVehicle(){
      Vehicle__c v = new Vehicle__C(name = 'Testing Vehicle');
      return v;
  }
  private static Product2 createEquipment(){
      product2 e = new product2(name = 'Testing equipment',lifespan_months__c = 10,maintenance_cycle__c = 10,replacement_part__c = true);
      return e;
    }
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case c = new case(Type='Repair',Status='New',Origin='Web',Subject='Testing

        return c;
    }
    private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id equipmentId,id requestId){
        Equipment_Maintenance_Item__c eMI = new Equipment_Maintenance_Item__c(Equipment__c = equipmentId,Maintenance_Request__c = requestId);
        return eMI;
    }
    // Success
    @isTest
    private static void pTest(){
        Vehicle__c v = createVehicle();
        insert v;
        id vId = v.Id;
        Product2 e = createEquipment();
        insert e;
        id eId = e.Id;
        case createdCase = createMaintenanceRequest(vId,eId);
        insert createdCase;
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = createEquipmentMaintenanceItem(eId,createdCase.id);
        insert equipmentMaintenanceItem;
        //test
        test.startTest();
        createdCase.status = 'Closed';
        update createdCase;
        test.stopTest();
        //test end
        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.assertEquals(newCase.Type, 'Routine Maintenance');
        SYSTEM.assertEquals(newCase.Equipment__c, eId);
        SYSTEM.assertEquals(newCase.Vehicle__c, vId);
    }
    //Fail
    @isTest
    private static void nTest(){
        Vehicle__C v = createVehicle();
        insert v;
        id vId = v.Id;
        product2 e = createEquipment();
        insert e;
        id eId = e.Id;
        case createdCase = createMaintenanceRequest(vId,eId);
        insert createdCase;
        Equipment_Maintenance_Item__c workP = createEquipmentMaintenanceItem(eId,
createdCase.Id);
        insert workP;
        //test
        test.startTest();
        createdCase.Status = 'Working';
        update createdCase;
        test.stopTest();
        //test end
        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(allCase.size() == 1);
    }
    //Bulk
    @isTest
    private static void bTest(){
        list<Vehicle__C> vList = new list<Vehicle__C>();
        list<Product2> eList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> eMIList = 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++){
            vList.add(createVehicle());
            eList.add(createEquipment());
        }
        insert vList;
        insert eList;
        for(integer i = 0; i < 300; i++){
            caseList.add(createMaintenanceRequest(vList.get(i).id, eList.get(i).id));
        }
        insert caseList;
        for(integer i = 0; i < 300; i++){
            eMIList.add(createEquipmentMaintenanceItem(eList.get(i).id, caseList.get(i).id));
        }
        insert eMIList;
        //test
        test.startTest();
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();
        //test end
        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);
    }
}

public with sharing class WarehouseCalloutService implements
```

```
Queueable {
    private static final String WAREHOUSE_URL = 'https://th-

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        //BoilerPlate
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        //End of BoilerPlate
        List<Product2> p2List = new List<Product2>();
        if (response.getStatusCode() == 200){
            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 p2 = new Product2();
                p2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                p2.Warehouse_SKU__c = (String) mapJson.get('sku');
                p2.Name = (String) mapJson.get('name');
                p2.ProductCode = (String) mapJson.get('_id');
                p2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                p2.Cost__c = (Integer) mapJson.get('cost');
                p2.Current_Inventory__c = (Double) mapJson.get('quantity');
                p2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                p2List.add(p2);
            }
            if (p2List.size() > 0){
                upsert p2List;
            }
        }
    }
    //function call
    public static void execute (QueueableContext context){

        runWarehouseEquipmentSync();

    }
}

```

```
@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":"55d66226726b611100aaf73f","replacement":false,
"quantity":10,"name":"UPS 2000 fespan":60,"cost":1350,
"sku":"100001"},{"_id":"55d66226726b611100aaf740",
"replacement":true,"quantity":194,"name":"Fuse ,"sku":"100002"}]');
        response.setStatusCode(200);
        return response;
    }
}

```

```
 @IsTest
private class WarehouseCalloutServiceTest {
    @isTest
    static void test(){
```

```
    // Boiler Plate
    test.startTest();
    Test.setMock(HttpCalloutMock.class, new 
WarehouseCalloutServiceMock());
    WarehouseCalloutService.execute(null);
        test.stopTest();
    //Boiler PLate end
    List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];
        //Assert
        System.assertEquals(2, product2List.size());
        System.assertEquals('55d66226726b611100aaf73f', 
product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf740', 
product2List.get(1).ProductCode);
    }

}

```

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

```

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new 
WarehouseCalloutServiceMock());
        String jobId = System.schedule('Warehouse Time to 
());
        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id
=: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State),
'Error');
        Test.stopTest();
    }
}

```