Virtual Internship Program- Salesforce Developer Catalyst

Apex specialist SuperBadge Codes:

1. **General Quiz**
2. **Automate Record Creation:**

Trigger: MaintenanceRequest

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

Class: 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;
       }
    }
}
```

<u>Test Class: Maintenance Request Helper Test</u>

```
@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);
    }
}
```

3. Synchronize Salesforce Data with an External System:
Class: 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);
         }
      }
   }
}
```

4.Schedule Synchronization:

Class: WarehouseSyncSchedule

```
global class WarehouseSyncSchedule implements Schedulable {
   global void execute(SchedulableContext ctx) {
       WarehouseCalloutService.runWarehouseEquipmentSync();
   }
}
```

5.Test Automation Logic:

Trigger: MaintenanceRequest

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

Class: 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;
       }
   }
}
```

<u>Test Class: MaintenanceRequestHelperTest</u>

```
@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);
    }
}
```

6. Test Callout Logic:

Class: 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);
            }
        }
    }
}
```

Test Class: WearhouseCalloutServiceTest

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

Test Class: 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","repl
acement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku
":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

7. Test Scheduling Logic:

Class: WarehouseSyncSchedule

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

Test Class: 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","repl
acement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku
":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Test Class: 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();
    }
}
```