Apex Specialist Superbadge:

In this superbadge, initial step is to create a new playground. Now the steps which are mentioned in 'set up development org' has to be done. Then according to the given process, write the code for each step mentioned below:

Step 1 : Answering the multiple choice questions.

Step 2 - Automate Record Creation :

Automate record creation using apex triggers.

Go to developer console and edit the apex class and the triggers for below:

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

                    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));
                } 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> 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;
        }
    }
}
```

MaintenanceRequestHelperTest

```
@isTest

private class MaintenanceRequestHelperTest {

    //Leverage a @testSetup method to reduce execution time and increase maintainability
    @testSetup
    static void allTheDataForThisTestClass() {

        // Principle #1: Create records from scratch!
        // Remember that Records created in a test setup method are rolled back at the end of test class execution.
        // Test setup methods enable you to create common test data easily and efficiently.
        // By setting up records once for the class, you don’t need to re-create records for each test method.

        Account acc = new Account();
        acc.Name = 'test';
        insert acc;

        Contact contact = new Contact();
        contact.FirstName = 'test';
        contact.LastName = 'last';
        contact.Email = 'test@test.com';
        contact.AccountId = acc.Id;
        insert contact;

        // This is a Custom Object
        Vehicle__c vehicle = new Vehicle__c();
        vehicle.Name = 'car';
        insert vehicle;

        Product2 product = new Product2();
        product.Name = 'test';
        product.isActive = true;
        product.Maintenance_Cycle__c = 2;
        product.Replacement_Part__c = true;
        insert product;
    }

    static testMethod void test_triggerMaintenanceRequestHelperTest() {



        // Principle #2: Test the class for as much user Profiles as necessary.
        //               Here we're going to use an Standard User.
        //               Also, please consider using some Global Static variables to store the test user info.

        // This code runs as the system user
        Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
        User u = new User(Alias = 'stdtest', Email='stdtest@testorg.com',
                          EmailEncodingKey='UTF-8', LastName='Testing', LanguageLocaleKey='en_US',
                          LocaleSidKey='en_US', ProfileId = p.Id,
                          TimeZoneSidKey='America/Los_Angeles', UserName='stdtest@testorg.com');

        System.runAs(u) {

            List<Case> caseList = new List<Case>();
            List<Case> secondList = new List<Case>();

            //grab the data that was created in the allTheDataForThisTestClass method
            Account acc = [SELECT Id, Name FROM Account WHERE Name = 'test' LIMIT 1];
            Contact contact  = [SELECT Id, FirstName, LastName,Email,AccountId FROM Contact WHERE Email = 'test@test.com' LIMIT 1];
            Vehicle__c vehicle = [SELECT Id, Name FROM Vehicle__c WHERE Name = 'car' LIMIT 1];
            Product2 product = [SELECT Id, Name, isActive, Maintenance_Cycle__c, Replacement_Part__c FROM Product2 WHERE Name = 'test' LIMIT 1];

            // "Setup" data has been entered, begin testing
            // This trick gives us a new set of Governor Limits!
            Test.startTest();

            // Test in bulk (200+ records)!
            for(Integer i=1;i<=1000;i++){
                Case maintenanceNew             = new Case();
                maintenanceNew.Subject          = 'Other';
                maintenanceNew.Vehicle__c       = vehicle.Id;
                maintenanceNew.Product__c       = product.Id;
                maintenanceNew.ContactId        = contact.Id;
                maintenanceNew.AccountId        = acc.Id;
                maintenanceNew.Type             = 'Other';
                maintenanceNew.Status           = 'New';
                maintenanceNew.Equipment__c     = product.Id;
                maintenanceNew.Date_Reported__c = Date.today();
                maintenanceNew.Date_Due__c      = Date.today();

                caseList.add(maintenanceNew);
            }

            insert caseList;

            // Assert your results!
            System.assertEquals(1000,caseList.size());

            //Now you can validate the Repair/Closed cases.
            for(Case cas:caseList){
                //update information
                cas.Type = 'Repair';
                cas.Status = 'Closed';
                secondList.add(cas);
            }

            update secondList;
            List<Case> createdCases = [Select Id from Case where Type = 'Routine Maintenance'];
            System.assertEquals(1000,createdCases.size());

              //Remember to stop the test.
              Test.stopTest();

              // Please remember to test things that shouldn't work!
              // Example: If you deleted records, create a query trying to find the records.
              // Then use the System.assertEquals(0,ShouldBeDeletedCases.size()); or something similar.
          }

      }

```

Step 3 - Synchronize the salesforce data with an external system:

Modify the Apex Classes as below, save and run all.

WarehouseCalloutService

```
public with sharing class WarehouseCalloutService implements
```

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

    //class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.
    //The callout’s JSON response returns the equipment records that you upsert in Salesforce.

    @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());

            //class maps the following fields: replacement part (always true), cost, current inventory, lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update within Salesforce
            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 = (Integer) mapJson.get('cost');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                myEq.ProductCode = (String) mapJson.get('_id');
                warehouseEq.add(myEq);
            }

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

            }
        }
    }

    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
    }

}
```

<u>Step 4 - Schedule Synchronization:</u>

Modify the Apex Classes as below, save and run all.

WarehouseSyncSchdeule

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

<u>Step 5 - Test automation logic</u> :

Modify the Apex Classes as below, save and run all.

MaintenanceRequestHelper

```
public class MaintenanceRequestHelper {

    public static void updateWorkOrders(Map<Id, Case> cases){

      // When testing this method, consider using a Test Data Factory // class or create all the data

      // Create a list of Cases
        List<Case> maintenance_routineList = new List<Case>();

        // Create a list to store the Product Maintenance Cycle
        List<Product2> listProduct = [select Id, Maintenance_Cycle__c from Product2];
        Map<Id,decimal> mapProduct = new Map<Id, decimal>();

        for (Product2 p : listProduct) {
            if (p != null) {
                if(p.Maintenance_Cycle__c != null){
                    mapProduct.put(p.Id, p.Maintenance_Cycle__c);
                }
            }
        }

        // Now, let's make the magic happen
        for(Case maintenance:cases.values()){
            Case maintenanceNew = new Case();

            maintenanceNew.Subject = maintenance.Subject;

            if (mapProduct.get(maintenance.Equipment__c) != null) {

  // Set the Due Date for the next maintenance
                maintenanceNew.Date_Due__c = Date.today().addDays(Integer.valueOf(mapProduct.get(maintenan

            }
            maintenanceNew.Vehicle__c = maintenance.Vehicle__c;
            maintenanceNew.Product__c = maintenance.Product__c;
            maintenanceNew.ContactId  = maintenance.ContactId;
            maintenanceNew.AccountId  = maintenance.AccountId;
            maintenanceNew.AssetId    = maintenance.AssetId;
            maintenanceNew.Type       = 'Routine

            maintenanceNew.Status     = 'New';
            maintenanceNew.Equipment__c = maintenance.Equipment__c;
            maintenanceNew.Date_Reported__c = Date.today();

            // Add this new maintenance to the list
            maintenance_routineList.add(maintenanceNew);
        }

        // Finally, with all complete list of future maintenance, we can now add the records:
        insert maintenance_routineList;
    }
}
```

MaintenanceRequestHelperTest

```
@isTest
private class MaintenanceRequestHelperTest {

    //Leverage a @testSetup method to reduce execution time and increase maintainability
    @testSetup
    static void allTheDataForThisTestClass() {

        // Principle #1: Create records from scratch!
        // Remember that Records created in a test setup method are rolled back at the end of test class execution.
        // Test setup methods enable you to create common test data easily and efficiently.
        // By setting up records once for the class, you don’t need to re-create records for each test method.

        Account acc = new Account();
        acc.Name = 'test';
        insert acc;

        Contact contact = new Contact();
        contact.FirstName = 'test';
        contact.LastName = 'last';
        contact.Email = 'test@test.com';
        contact.AccountId = acc.Id;
        insert contact;

        // This is a Custom Object
        Vehicle__c vehicle = new Vehicle__c();
        vehicle.Name = 'car';
        insert vehicle;

        Product2 product = new Product2();
        product.Name = 'test';
        product.isActive = true;
        product.Maintenance_Cycle__c = 2;
        product.Replacement_Part__c = true;
        insert product;
    }

    static testMethod void test_triggerMaintenanceRequestHelperTest() {



        // Principle #2: Test the class for as much user Profiles as necessary.
        //               Here we're going to use an Standard User.
        //               Also, please consider using some Global Static variables to store the test user info.

        // This code runs as the system user
        Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
        User u = new User(Alias = 'stdtest', Email='stdtest@testorg.com',
                          EmailEncodingKey='UTF-8', LastName='Testing', LanguageLocaleKey='en_US',
                          LocaleSidKey='en_US', ProfileId = p.Id,
                          TimeZoneSidKey='America/Los_Angeles', UserName='stdtest@testorg.com');

        System.runAs(u) {

            List<Case> caseList = new List<Case>();
            List<Case> secondList = new List<Case>();

            //grab the data that was created in the allTheDataForThisTestClass method
            Account acc = [SELECT Id, Name FROM Account
```

```
WHERE Name = 'test' LIMIT 1];
            Contact contact  = [SELECT Id, FirstName, LastName,Email,AccountId FROM Contact WHERE Email = 'test@test.com' LIMIT 1];
            Vehicle__c vehicle = [SELECT Id, Name FROM Vehicle__c WHERE Name = 'car' LIMIT 1];
            Product2 product = [SELECT Id, Name, isActive, Maintenance_Cycle__c, Replacement_Part__c FROM Product2 WHERE Name = 'test' LIMIT 1];

            // "Setup" data has been entered, begin testing
            // This trick gives us a new set of Governor Limits!
            Test.startTest();

            // Test in bulk (200+ records)!
            for(Integer i=1;i<=1000;i++){
                Case maintenanceNew             = new Case();
                maintenanceNew.Subject          = 'Other';
                maintenanceNew.Vehicle__c       = vehicle.Id;
                maintenanceNew.Product__c       = product.Id;
                maintenanceNew.ContactId        = contact.Id;
                maintenanceNew.AccountId        = acc.Id;
                maintenanceNew.Type             = 'Other';
                maintenanceNew.Status           = 'New';
                maintenanceNew.Equipment__c     = product.Id;
                maintenanceNew.Date_Reported__c = Date.today();
                maintenanceNew.Date_Due__c      = Date.today();

                caseList.add(maintenanceNew);
            }

            insert caseList;

            // Assert your results!
            System.assertEquals(1000,caseList.size());

            //Now you can validate the Repair/Closed cases.
            for(Case cas:caseList){
                //update information
                cas.Type = 'Repair';
                cas.Status = 'Closed';
                secondList.add(cas);
            }

            update secondList;
            List<Case> createdCases = [Select Id from Case where Type = 'Routine Maintenance'];
            System.assertEquals(1000,createdCases.size());

            //Remember to stop the test.
            Test.stopTest();

            // Please remember to test things that shouldn't work!
            // Example: If you deleted records, create a query trying to find the records.
            // Then use the System.assertEquals(0,ShouldBeDeletedCases.size()); or something similar.
        }

    }
}
```

Step 6 - Test callout logic :

Modify the Apex Classes as below, save and run all.

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-
));
        System.assertEquals('GET', request.getMethod());

        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('[{"_id":"55d66226726b611100aaf741","repla
        response.setStatusCode(200);
        return response;
    }
}
```

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

                System.debug(warehouseEq);
            }

        }
    }
}
```

Step 7 - Test scheduling logic :

Modify the Apex Classes as below, save and run all.

WarehouseSyncSchedule

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

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

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 ');


    }
}
```