APEX SUPER BADGE CODES

APEX TRIGGERS

AccountAddressTrigger.apxt:-

```
trigger AccountAddressTrigger on Account (beforeinsert,before update) {
    for(Account a:Trigger.New){
        if(a.Match_Billing_Address_c==true){
            a.ShippingPostalCode=a.BillingPostalCode;
        }
    }

}
```

ClosedOpportunityTrigger.apxt:-

```
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

VerifyDate.apxc:-

```
public classVerifyDate {
```

```
            public static Date CheckDates(Date date1, Date date2) {
            if(DateWithin30Days(date1,date2)) {
              return date2;
            }
else {
   return SetEndOfMonthDate(date1);}

            }

            private static BooleanDateWithin30Days(Date date1, Date date2)
                  {if( date2 < date1){ return false;}

                  Date date30Days =
              date1.addDays(30);if( date2 >=
              date30Days ) { return false; } else { return
              true; }
}
            private staticDate SetEndOfMonthDate(Date date1){
              Integer totalDays= Date.daysInMonth(date1.year(), date1.month());
              Date lastDay = Date.newInstance(date1.year(), date1.month(),
              totalDays);return lastDay;
            }
          }
```

TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate
{
   static testMethod void testMethod1()
   {
      Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
      Date d1 =
      VerifyDate.CheckDates(System.today(),System.today()+60);
   }
}
```

RestrictContactByName.apxt

```
trigger RestrictContactByName on Contact (beforeinsert, before update){
        for (Contact c : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {
                c.AddError('The Last Name "'+c.LastName+'" is not allowedfor DML');
                }
          }
 }
@isTestprivate class
        TestRestrictContactByName {
        statictestMethod void  metodoTest()
        {
            List<Contact> listContact= new List<Contact>();
            Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' ,
        email='Test@test.com');
            Contact c2 = new Contact(FirstName='Francesco1', LastName=
        'INVALIDNAME',email='Test@test.com');
            listContact.add(c
            1);
            listContact.add(c
            2);
            Test.startTest();
               try{
                  insert listContact;
               }
```

```
                    catch(Exception
                ee){}Test.stopTest();
            }
        }
```

RandomContactFactory.apxc:

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer numContactsToGenerate,
StringFName) {
        List<Contact> contactList = new
        List<Contact>();for(Integer
        i=0;i<numContactsToGenerate;i++) {
            Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
            contactList.add(c);
            System.debug(c);
        }
        System.debug(contactList.size());
        return contactList;
        Contact c = new
        Contact();c.FirstName =
        'Bob'; c.LastName =
        'Willie'; c.AccountId =
        a.Id;
        Contact c2 = new Contact();
        c2.FirstName = 'Tom';
        c2.LastName = 'Cruise';
```

```
        c2.AccountId = a.Id;
        List<Id> acctIds = new List<Id>();
        acctIds.add(a.Id);
        Test.startTest();
        AccountProcessor.countContacts(acctId
        s);Test.stopTest();
    }
}
```

LeadProcessor.apxc:

```
public class LeadProcessor implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
    public void execute(Database.BatchableContext bc, List<Lead>
        leads){for (Lead Lead : leads){
            lead.LeadSource = 'Dreamforce';
        }
        update leads;
    }
    public void finish(Database.BatchableContext bc){
    }
}
```

LeadProcessorTest.apxc

```
@isTest
```

```
public class LeadProcessorTest {
    @testSetup
   static void setup(){
    List<Lead> leads = new List<Lead>();
    for(Integer counter=0 ;counter
    <200;counter++){
      Lead lead = new Lead();
      lead.FirstName ='FirstName';
      lead.LastName
      ='LastName'+counter;
      lead.Company
      ='demo'+counter;leads.add(lead);
    }
    insert leads;
   }
   @isTest static void test() {
    Test.startTest();
    LeadProcessor leadProcessor = new LeadProcessor();
    Id batchId = Database.executeBatch(leadProcessor);
    Test.stopTest();
   }
}
```

AddPrimaryContact.apxc

```
public classAddPrimaryContact implements Queueable
```

```
{private Contact
c; private String
state;
public  AddPrimaryContact(Contact c, String state)
{
   this.c = c;
   this.state =
   state;
}
public void execute(QueueableContext context)
{
    List<Account> ListAccount = [SELECT ID, Name ,(Selectid,FirstName,LastName from
contacts) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
    List<Contact> lstContact = new
    List<Contact>();for (Account acc:ListAccount)
    {
        Contact cont = c.clone(false,false,false,false);
        cont.AccountId =  acc.id;
        lstContact.add( cont );
    }
    if(lstContact.size() >0{
       insert lstContact;
    }
}
```

```
}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest
{
    @isTest staticvoid TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
            Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
            Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
        insert Teste;
        Contact co = new
        Contact();
        co.FirstName='demo';
        co.LastName
        ='demo';insert co;
        String state = 'CA';
```

```
        AddPrimaryContact apc = new AddPrimaryContact(co,
        state);Test.startTest();
          System.enqueueJob(apc);
        Test.stopTest();
      }
    }
```

DailyLeadProcessor.apxc

```
    public class DailyLeadProcessor implements Schedulable  {Public void
     execute(SchedulableContext SC){List<Lead> LeadObj=[SELECT Id from Lead
     where LeadSource=null limit 200]; for(Leadl:LeadObj){
        l.LeadSource='Dreamforce';
        update l;
      }
   }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
        static testMethod void testDailyLeadProcessor() {
                String CRON_EXP = '0 0 1 * * ?';
                List<Lead> lList = new List<Lead>();
            for (Integer i = 0; i < 200; i++) {
                        lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1
```

```
        Inc.',Status='Open - Not Contacted'));
                        }
insert lList;
                        Test.startTest();
                        String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
        DailyLeadProcessor());
                }
        }
```

Apex Integration Services

AnimalLocator.apxc:

```
public class AnimalLocator{
    public static StringgetAnimalNameById(Integer
        x){Http http = new Http();
        HttpRequest req = new HttpRequest();
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        Map<String, Object> animal=new Map<String, Object>();
        HttpResponse res = http.send(req);
          if (res.getStatusCode() == 200) {
      Map<String, Object> results = (Map<String,
    Object>)JSON.deserializeUntyped(res.getBody());
     animal = (Map<String, Object>) results.get('animal');
        }
```

```
    return (String)animal.get('name');
    }
}
```

AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest{
    @isTest static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        string result= AnimalLocator.getAnimalNameById(3);
        String expectedResult = 'chicken';
        System.assertEquals(result,expectedResult );
    }
}
```

AnimalLocatorMock.apxc

```
@isTest
global classAnimalLocatorMock implements HttpCalloutMock {
    // Implementthis interface method
    global HTTPResponse respond(HTTPRequest request) {
        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type',
        'application/json');
```

```
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
"mighty moose"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

ParkLocator.apxc

```
public class ParkLocator {
    public staticstring[] country(string theCountry) {
        ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort(); // remove space
        return parkSvc.byCountry(theCountry);
    }
}
```

ParkLocatorTest.apxc

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout()
    'Yosemite'};
}
```

```
Test.setMock(WebServiceMock.class, new ParkServiceMock ());String
country = 'United States';
List<String> result = ParkLocator.country(country);
List<String> parks = new List<String>{'Yellowstone', 'MackinacNational Park',
```

```
System.assertEquals(parks, result);
```

ParkServiceMock.apxc

```
@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) {
    // start - specifythe response you want to send
        ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
  response_x.return_x = new List<String>{'Yellowstone', 'MackinacNational Park','Yosemite'};
                                                                                    }
// end
response.put('response_x', response_x);
}
```

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount(){
        RestRequestrequest=RestContext.reques
        t;
        stringaccountId=request.requestURI.substringBetween('Accounts/','/contacts');
        Account result=[SELECT Id,Name,(Select Id,Name from Contacts) from Account where
Id=:accountId Limit 1];
        return result;
    }
}
```

AccountManagerTest.apxc

```
@IsTest
private classAccountManagerTest {
    @isTest static void
        testGetContactsByAccountId(){Id
        recordId=createTestRecord();
        RestRequest request=new RestRequest();
        request.requestUri='https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+
recordId+'/contacts';
        request.httpMethod='GET';
        RestContext.request=request;
```

```
        Account thisAccount=AccountManager.getAccount();
        System.assert(thisAccount != null);
        System.assertEquals('Test record',thisAccount.Name);
    }
    static Id createTestRecord(){

        Account accountTest=new
        Account(Name='Test record'
        );
        insert accountTest;
        Contact contactTest=new Contact(
        FirstName='John',LastName='Doe',AccountId=accountTest.Id
        );insertcontactTest;
        return accountTest.Id;
    }
}
```

APEXSPECIALIST SUPER BADGE

Challenge 1:

MaintenanceRequestHelper.apxc

```
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_
cFROM Equipment_Maintenance_Items_r)
                                                FROM Case WHEREId 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_cIN :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.apxt

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

Challenge-2:

WarehouseCalloutService.apxc

```
public with sharingclass WarehouseCalloutService implements Queueable
    {privatestatic final StringWAREHOUSE_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 (Objecteq : jsonResponse){

                      Map<String,Object> mapJson =

                      (Map<String,Object>)eq;Product2myEq = 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){upsertwarehouseEq;

                    System.debug('Your equipmentwas synced with the warehouse one');

}


            }

        }


        public static void execute (QueueableContext context){

            runWarehouseEquipmentSync();

        }

    }
```

Challenge-3:

WarehouseSyncSchedule.apxc

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

Challenge-4:

MaintenanceRequestHelperTest.apxc

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string STATUS_NEW =
    'New'; privatestatic final stringWORKING =
    'Working'; private static final string CLOSED=
    'Closed';

    private staticfinal string REPAIR = 'Repair';
    private staticfinal string REQUEST_ORIGIN = 'Web';
    private static finalstring 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_SUBJE
                        CT,
                        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();
            insertequipment;
            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();insertequipment;
    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 = [selectid
                                                 from Equipment_Maintenance_Item__c
                                                 where Maintenance_Request__c in: oldRequestIds];


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

```
    }
}
```

MaintenanceRequestHelper.apxc

```
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.apxt

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

Challenge-5:

WarehouseCalloutService.apxc

```
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 (Objecteq : jsonResponse){
            Map<String,Object> mapJson =
            (Map<String,Object>)eq;Product2myEq = 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){upsertwarehouseEq;
            System.debug('Your equipment was synced with the warehouseone');
            System.debug(warehouseEq);
        }}}}
```

WarehouseCalloutServiceTest.apxc

```
@isTest
        private class
        WarehouseCalloutServiceTest {@isTest
        static void
            testWareHouseCallout(){
            Test.startTest();

            Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
            WarehouseCalloutService.runWarehouseEquipmentSync();
            Test.stopTest();
```

```
        System.assertEquals(1, [SELECTcount() FROM Product2]);
    }
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock
  {globalstatic HttpResponse respond(HttpRequest request){
      System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
      System.assertEquals('GET', request.getMethod());
      HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type',
      'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"nam
e":"Generator
1000kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
      response.setStatusCode(200);
      return response;
  }
}
```

Challenge-6:

WarehouseSyncSchedule.apxc

```
global class WarehouseSyncSchedule implements Schedulable {

    globalvoid execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();
     }
}
```

WarehouseSyncScheduleTest.apxc

```
@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();

        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];

        System.assertEquals(jobID, a.Id,'Schedule ');

    }
```

}

Process Automation Specialist Super BadgeCodes

PROCESS AUTOMATIONSUPER BADGE

Challenge 1: **Automate Leads:**

Validation rule on Lead

Search for Validation rule and create a new under Leads

Rule Name: Anything

Error ConditionFormula :

OR(AND(LEN(State) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:
MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN
:TX:UT:VT:VA:WA:WV:WI:WY", State))), NOT(OR(Country ="US",Country
="USA",Country ="United States", ISBLANK(Country))))

Create two Queues:

Search in quick box and select lead as object and create the below queues.

Queue Name: Rainbow Sales; **AND** Assembly SystemSales

Assignment Rule:

we should create lead assignment rule for Rainbow Sales and System Sales.

Challenge 2: Automate Accounts:

Create 4 Roll Up Summary fields as below:

Field 1: Label: **Number of deals**

Summary Type: **COUNT**

Summarized Object: **Opportunity**

Filter Criteria: **None**

Field 2: Label: **Number of won deals**Summary Type: **COUNT**

Summarized Object: **Opportunity**

Filter Criteria: **Stage EQUALS Closed Won**

Field 3: Label: **Last won deal date**

Summary Type: **MAX**

Field to Aggregate: **Opportunity: Close Date**

Summarized Object: **Opportunity**

Filter Criteria: **Stage EQUALS Closed Won**

Field 4: Label: **Amount of won deals**

Summary Type: **SUM**

Field to Aggregate: **Opportunity: Amount**

Summarized Object: **Opportunity**

Filter Criteria: **Stage EQUALSClosed Won**

create aformula relationships with below data:

Field 5: Label: **Deal win percent**

Return Type: **Percent**

Decimal Places: 2

Formula: ***(Number_of_won_deals___c / Number_of_deals___c)***

Field 6: Label: **Call for Service**

Return Type: **Text**

Formula: ***IF(DATE(YEAR(Last_won_deal_date_c)+2 , MONTH(Last_won_deal_date_c),DAY(Last_won_deal_date_c)) <= TODAY(),"Yes","No")***

Create 2 validation rules as below

Validation Rule 1 : **Rule Name :**

(*Anything*)**Error Condition Formula:**

OR(AND(LEN(BillingState) > 2,

NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", BillingState))

),AND(LEN(ShippingState) > 2,

NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", ShippingState))

),NOT(OR(BillingCountry ="US",BillingCountry ="USA",BillingCountry ="United States", ISBLANK(BillingCountry))),

NOT(OR(ShippingCountry ="US",ShippingCountry ="USA",ShippingCountry ="United States", ISBLANK(ShippingCountry))))

Error Message : You can not save a new account unless the shipping and billing state fields are valid US state abbreviations, and the country field is either blank or US, USA,or United States.

Error Location: Top Of Page

VALIDATION RULE 2 : **Rule Name :** NameChange

Error ConditionFormula :
ISCHANGED(Name) && (OR(ISPICKVAL(Type ,'Customer - Direct')
,ISPICKVAL(Type ,'Customer - Channel')))

Error Message : You can't change the Account name for "Customer– Direct" or "Customer – Channel"
Error Location: Account Name
Sometimes when validation is right and it doesn'twork rightly just delete and recreateit from scratch.

Challenge 3: **Create RobotSetup Object**

creating a:
Robot Setup with a Master-Detail relationship to the **opportunity** include Autonumber the record name, starting with 0 using name format:ROBOT SETUP-{0000}.
Usethe following field names.

Date, Date_c : Date type
Notes, Notes_c : Text type
Day of the Week, Day_of_the_Week_c: Number

Challenge 4: **Create Sales Process and Validate**

Opportunities:

by creatinga Field in Oppurtunity we Should:

Approval: Checkboxtype

the sales reps shouldn't be able to check that box and only system administrators like

and sales managers should be able to check it. Though it doesn't throw an error forthat condition.

Also, Click on the Opportunity field **STAGE** and add a picklistvalue as **"Awaiting Approval"**

Next, **create a salesprocess** under opportunities by searching the sales processin the Search box.

Next add the **Opportunity Validation Rule** with errorformula as below:

IF((Amount> 100000 && Approved_c <> True && ISPICKVAL(StageName,'ClosedWon')),True,False)

Challenge 5: **Automate Opportunities:**

Create Three Email Templates:

Finance: Account Creation,

SALES: Opportunity Needs Approval,

Sales:Opportunity Approval Status

Create related Email Alert from search box for the templates above.

Create an approval process:

Search for the approval processand select an **opportunity** object.

Criteria :

(Opportunity: Stage EQUALS Negotiation/Review**)** AND **(**Opportunity: AmountGREATER THAN 100000**)**

SALES: Opportunity Needs Approval——->Template. Make sure to populate your manager as **Nushi Davoud** in**Manage Users**.

Create a process with the process builder

Opportunity object with option createdand updated.

Node 1 Criteria.: Opportunity.Account Type = customerand Opportunity.account id not equal to null

Node 2 Criteria.: Opportunity.Account Type = Prospect, Opportunity stage =

prospecting and Opportunity.account id not equal to null

Node 3 Criteria.: Opportunity Stage = Negotiation/Review and Opportunity Amount > 100,000

Node 4 Criteria.: Opportunity Stage = ClosedWon

Action for **Node 1 Email Alert** to mail notifies account creation : Finance: Account Creation.

Action for **Node 2 :**

Email Alert to mail notifies accountcreation : Finance:Account Creation.

Create a Record: Task with any name but mandatory subjectline 'Send MarketingMaterials'.

Makesure the string has no full stop or comma to it.

Assigned to the **Account owner**

Action for **Node 3**: Approvals

Choose the one we created for theopportunity here. And it takes care of the process thereby.

Action for **Node 4: Record** for Robot Setup

Set fields as below and Date formulabeing (closed date +180)

Challenge 6: **C**reate Flow for Opportunities:

Create **Flow** named **Product QuickSearch**

Create Flow for Opportunities

Element 1: Screen componentfrom the paletteName:**Product Quick Search**

Add **Record Button** fromthe Input as below:

Label: **Product Type**

Data Type: Text

Required: Check

Under Choices: Add new resource

Type:Choice
Create three choices as below for **RainbowBot**, **CloudyBot,and Assemble Systems.Element 2**: **Get Record**
Label: Search Prod select object as Product.
Under RecordCollection: Add New Resource **Filterresult**:
VariableElement3: Loop
Add New Resource **Loop : Variable Type**
Element 4: Assignment
Add New Resource **Looptxt1 : Variable**
TypeElement 5: Screen
Save and Activate the flow.
Now search **Lightning App Builder**Add **New page**: Select
Record Type ***Label:***
Product_Quick_Search Object:
Opportunity
Pick any template
And Drag and drop Flows from Left palette, select the flowwe made and Save.

Challenge 7: **Automate Setups:**

Search for the field “Day of the Week” on robot object and change the field type from Numberto formula field of return type: text and use the below formula:

```
[FORMU
LA ]
CASE(
MOD([Opportunity].CloseDate + 180 - DATE(1900, 1, 7),7),
 0, [Opportunity].CloseDate + 181,
 6, [Opportunity].CloseDate + 182,
 [Opportunity].CloseDate + 180
)
```

By Changingthis Formula ,Savingand Activing The Process BuilderWe Can Complete This

Challenge.

```
Contact c = new Contact();c.FirstName = 'Bob'; c.LastName = 'Willie'; c.AccountId = a.Id;
Contact c2 = new Contact();
c2.FirstName = 'Tom';
c2.LastName = 'Cruise';
c2.AccountId = a.Id;
List<Id> acctIds = new List<Id>();
acctIds.add(a.Id);
Test.startTest();
AccountProcessor.countContacts(acctId
s);Test.stopTest();
}
}
```

LeadProcessor.apxc:

```
public class LeadProcessor implements Database.Batchable<sObject> {
public Database.QueryLocator start(Database.BatchableContext bc) {
return Database.getQueryLocator([Select LeadSource From Lead ]);
}
public void execute(Database.BatchableContext bc, List<Lead>
leads){for (Lead Lead : leads){
lead.LeadSource = 'Dreamforce';
}
update leads;
}
```

```
    public void finish(Database.BatchableContext bc){
                                                        }

}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {
    @testSetup
   static void setup(){
    List<Lead> leads = new List<Lead>();
    for(Integer counter=0 ;counter
    <200;counter++){
      Lead lead = new Lead();
      lead.FirstName ='FirstName';
      lead.LastName
      ='LastName'+counter;
      lead.Company
      ='demo'+counter;leads.add(lead);
    }
    insert leads;
   }
   @isTest static void test() {
    Test.startTest();
    LeadProcessor leadProcessor = new LeadProcessor();
    Id batchId = Database.executeBatch(leadProcessor);
```

```
        Test.stopTest();
    }
}
```

AddPrimaryContact.apxc

```
public classAddPrimaryContact implements Queueable
{

 private Contact
c; private String
state;
public  AddPrimaryContact(Contact c, String state)
{
    this.c = c;
    this.state =
    state;
}
public void execute(QueueableContext context)
{
    List<Account> ListAccount = [SELECT ID, Name ,(Selectid,FirstName,LastName from
contacts) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];
    List<Contact> lstContact = new
    List<Contact>();for (Account acc:ListAccount)
    {
        Contact cont = c.clone(false,false,false,false);
```

```
            cont.AccountId =  acc.id;
            lstContact.add( cont );
        }
        if(lstContact.size() >0)
        {
            insert lstContact;}
    }
}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest
{
    @isTest staticvoid TestList()
    {
        List<Account> Teste = new List <Account>();
        for(Integer i=0;i<50;i++)
        {
            Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));
        }
        for(Integer j=0;j<50;j++)
        {
            Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));
        }
```

```
        insert Teste;
        Contact co = new
        Contact();
        co.FirstName='demo';
        co.LastName
        ='demo';insert co;
        String state = 'CA';
         AddPrimaryContact apc = new AddPrimaryContact(co,
         state);Test.startTest();
           System.enqueueJob(apc);
         Test.stopTest();
      }
   }
```

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements Schedulable  {Public void execute(SchedulableContext
SC){

      List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
       for(Leadl:LeadObj){
          l.LeadSource='Dreamforce';
          update l;
       }
    }
  }
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
        static testMethod void testDailyLeadProcessor() {
                String CRON_EXP = '0 0 1 * * ?';
                List<Lead> lList = new List<Lead>();
            for (Integer i = 0; i < 200; i++) {
                        lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1
Inc.',Status='Open - Not Contacted'));
                }
                insert lList;


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

Apex Integration Services

AnimalLocator.apxc:

```
public class AnimalLocator{
    public static StringgetAnimalNameById(Integer
        x){Http http = new Http();
        HttpRequest req = new HttpRequest();
```

```
        req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + x);
        req.setMethod('GET');
        Map<String, Object> animal=new Map<String, Object>();
        HttpResponse res = http.send(req);
          if (res.getStatusCode() == 200) {
       Map<String, Object> results = (Map<String,
    Object>)JSON.deserializeUntyped(res.getBody());
      animal = (Map<String, Object>) results.get('animal');
        }
    return (String)animal.get('name');
      }
}
```

AnimalLocatorTest.apxc

```
    @isTest
    private class AnimalLocatorTest{
      @isTest static void AnimalLocatorMock1() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        string result= AnimalLocator.getAnimalNameById(3);
        String expectedResult = 'chicken';
        System.assertEquals(result,expectedResult );
      }
    }
```

AnimalLocatorMock.apxc

```
    @isTest
```

```
global classAnimalLocatorMock implements HttpCalloutMock {
    // Implementthis interface method
   global HTTPResponse respond(HTTPRequest request) {
      // Create a fake response
      HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type',
      'application/json');
      response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken",
"mighty moose"]}');
      response.setStatusCode(200);
      return response;
   }
}
```

ParkLocator.apxc

```
public class ParkLocator {
   public staticstring[] country(string theCountry) {
      ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort(); // remove space
      return parkSvc.byCountry(theCountry);
   }
}
```

ParkLocatorTest.apxc

```
@isTest
  private class ParkLocatorTest { @isTest static void testCallout()
{'Yosemite'};
}}
```

```
Test.setMock(WebServiceMock.class, new ParkServiceMock ());String country = 'United States';
 List<String> result = ParkLocator.country(country);
 List<String> parks = new List<String>{'Yellowstone', 'MackinacNational Park',
 System.assertEquals(parks, result);
```

ParkServiceMock.apxc

```
@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) {

            // start - specifythe response you want to send
                ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
          response_x.return_x = new List<String>{'Yellowstone', 'MackinacNational Park','Yosemite'};

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

        }
```

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount(){
        RestRequestrequest=RestContext.reques
        t;
        stringaccountId=request.requestURI.substringBetween('Accounts/','/contacts');
        Account result=[SELECT Id,Name,(Select Id,Name from Contacts) from Account where
Id=:accountId Limit 1];
        return result;
    }
}
```

AccountManagerTest.apxc

```
@IsTest
private classAccountManagerTest {
    @isTest static void
        testGetContactsByAccountId(){Id
        recordId=createTestRecord();
        RestRequest request=new RestRequest();
        request.requestUri='https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+
recordId+'/contacts';
        request.httpMethod='GET';
        RestContext.request=request;
        Account thisAccount=AccountManager.getAccount();
        System.assert(thisAccount != null);
```

```
        System.assertEquals('Test record',thisAccount.Name);}

    static Id createTestRecord(){
        Account accountTest=new
        Account(Name='Test record'
        );
        insert accountTest;
        Contact contactTest=new Contact(
        FirstName='John',LastName='Doe',AccountId=accountTest.Id
        );insertcontactTest;
        return accountTest.Id;
    }
}
```

APEXSPECIALIST SUPER BADGE

Challenge 1:

MaintenanceRequestHelper.apxc

```
    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_
cFROM Equipment_Maintenance_Items_r)

                                                    FROM Case WHEREId 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_cIN :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.apxt

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

            }
```

Challenge-2:

WarehouseCalloutService.apxc

```
public with sharingclass WarehouseCalloutService implements Queueable

    {privatestatic final StringWAREHOUSE_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 (Objecteq : jsonResponse){

              Map<String,Object> mapJson =

              (Map<String,Object>)eq;Product2myEq = 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){upsertwarehouseEq;
                System.debug('Your equipmentwas synced with the warehouse one');
            }

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

    }
```

Challenge-3:

WarehouseSyncSchedule.apxc

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

Challenge-4:

MaintenanceRequestHelperTest.apxc

```
@istest
public with sharing class MaintenanceRequestHelperTest {

   private static final string STATUS_NEW =
   'New'; privatestatic final stringWORKING =
   'Working'; private static final string CLOSED=
   'Closed';

   private staticfinal string REPAIR = 'Repair';
   private staticfinal string REQUEST_ORIGIN = 'Web';
   private static finalstring 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_SUBJE

                    CT,

                    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();insertequipment;
    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();
            insertequipment;
            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 = [selectid
                                                    from Equipment_Maintenance_Item_c
                                                    where Maintenance_Request_c in: oldRequestIds];
system.assert(allRequests.size() == 300);
        }
    }
```

MaintenanceRequestHelper.apxc

```
    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.apxt

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

Challenge-5:

WarehouseCalloutService.apxc

```
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 (Objecteq : jsonResponse){

      Map<String,Object> mapJson =

      (Map<String,Object>)eq;Product2myEq = 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){upsertwarehouseEq;

        System.debug('Your equipment was synced with the warehouseone');
        System.debug(warehouseEq)}}}}
```

WarehouseCalloutServiceTest.apxc

```
        @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, [SELECTcount() FROM Product2]);

           }

        }
```

WarehouseCalloutServiceMock.apxc

```
        @isTest
```

```
global class WarehouseCalloutServiceMock implements HttpCalloutMock

    {globalstatic HttpResponse respond(HttpRequest request){

        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());

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

        HttpResponse response = new HttpResponse();

        response.setHeader('Content-Type',

        'application/json');


response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"nam
e":"Generator
1000kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');

        response.setStatusCode(200);

        return response;

    }

}
```

WarehouseSyncSchedule.apxc

```
global class WarehouseSyncSchedule implements Schedulable {

    globalvoid execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();

    }
```

```
}
```

WarehouseSyncScheduleTest.apxc

```
@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();
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');
    }
}
```

Process Automation Specialist Super BadgeCodes

PROCESS AUTOMATIONSUPER BADGE

Challenge 1: **Automate Leads:**

Validation rule on Lead

Search for Validation rule and create a new under Leads

Rule Name: Anything

Error ConditionFormula :

OR(AND(LEN(State) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", State))), NOT(OR(Country ="US",Country ="USA",Country ="United States", ISBLANK(Country))))

Create two Queues:

Search in quick box and select lead as object and create the below queues.

Queue Name: Rainbow Sales; **AND** Assembly SystemSales

Assignment Rule:we should create lead assignment rule for Rainbow Sales and System Sales.

Challenge 2: Automate Accounts:

Create 4 Roll Up Summary fields as below:

Field 1: Label: **Number of deals**
Summary Type: **COUNT**
Summarized Object: **Opportunity**
Filter Criteria: **None**

Field 2: Label: **Number of won deals**Summary Type: **COUNT**
Summarized Object: **Opportunity**
Filter Criteria: **Stage EQUALS Closed Won**
Field 3: Label: **Last won deal date**
Summary Type: **MAX**
Field to Aggregate: **Opportunity: Close Date**
Summarized Object: **Opportunity**
Filter Criteria: **Stage EQUALS Closed Won**
Field 4: Label: **Amount of won deals**
Summary Type: **SUM**
Field to Aggregate: **Opportunity: Amount**
Summarized Object: **Opportunity**
Filter Criteria: **Stage EQUALSClosed Won**
create aformula relationships with below data:
Field 5: Label: **Deal win percent**
Return Type: **Percent**
Decimal Places: 2
Formula: ***(Number_of_won_deals___c / Number_of_deals___c)***

Field 6: Label: **Call for Service**
Return Type: **Text**
Formula: ***IF(DATE(YEAR(Last_won_deal_date_c)+2 , MONTH(Last_won_deal_date_c),DAY(Last_won_deal_date_c)) <= TODAY(),"Yes","No")***

Create 2 validation rules as below
Validation Rule 1 : **Rule Name :**
(*Anything*)**Error Condition Formula:**
OR(AND(LEN(BillingState) > 2,NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:M

E:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", BillingState))
),AND(LEN(ShippingState) > 2,
NOT(CONTAINS("AL:AK:AZ:AR:CA:CO:CT:DE:DC:FL:GA:HI:ID:IL:IN:IA:KS:KY:LA:ME:MD:MA:MI:MN:MS:MO:MT:NE:NV:NH:NJ:NM:NY:NC:ND:OH:OK:OR:PA:RI:SC:SD:TN:TX:UT:VT:VA:WA:WV:WI:WY", ShippingState))
),NOT(OR(BillingCountry ="US",BillingCountry ="USA",BillingCountry ="United States", ISBLANK(BillingCountry))),
NOT(OR(ShippingCountry ="US",ShippingCountry ="USA",ShippingCountry ="United States", ISBLANK(ShippingCountry))))

Error Message : You can not save a new account unless the shipping and billing state fields are valid US state abbreviations, and the country field is either blank or US, USA,or United States.
Error Location: Top Of Page

VALIDATION RULE 2 : **Rule Name :** NameChange

Error ConditionFormula :
ISCHANGED(Name) && (OR(ISPICKVAL(Type ,'Customer - Direct') ,ISPICKVAL(Type ,'Customer - Channel')))

Error Message : You can't change the Account name for "Customer– Direct" or "Customer – Channel"
Error Location: Account Name
Sometimes when validation is right and it doesn'twork rightly just delete and recreateit from scratch.

Challenge 3: **Create RobotSetup Object**

creating a:
Robot Setup with a Master-Detail relationship to the **opportunity** include Autonumber

the record name, starting with 0 using name format:ROBOT SETUP-{0000}.

Usethe following field names.

Date, Date_c : Date type
Notes, Notes_c : Text type
Day of the Week, Day_of_the_Week_c: Number

Challenge 4: **Create Sales Process and Validate**

Opportunities:

by creatinga Field in Oppurtunity we Should:

Approval: Checkboxtype

the sales reps shouldn't be able to check that box and only system administrators like and sales managers should be able to check it. Though it doesn't throw an error forthat condition.

Also, Click on the Opportunity field **STAGE** and add a picklistvalue as **"Awaiting Approval"**

Next, **create a salesprocess** under opportunities by searching the sales processin the Search box.

Next add the **Opportunity Validation Rule** with errorformula as below:

IF((Amount> 100000 && Approved_c <> True && ISPICKVAL(StageName,'ClosedWon')),True,False)

Challenge 5: **Automate Opportunities:**

Create Three Email Templates:

Finance: Account Creation,

SALES: Opportunity Needs Approval,

Sales:Opportunity Approval Status

Create related Email Alert from search box for the templates above.

Create an approval process:

Search for the approval processand select an **opportunity** object.

Criteria :

(Opportunity: Stage EQUALS Negotiation/Review) AND (Opportunity: AmountGREATER THAN 100000)

SALES: Opportunity Needs Approval——->Template. Make sure to populate your manager as **Nushi Davoud** in**Manage Users**.

Create a process with the process builder

Opportunity object with option createdand updated.

Node 1 Criteria.: Opportunity.Account Type = customerand Opportunity.account id not equal to null

Node 2 Criteria.: Opportunity.Account Type = Prospect, Opportunity stage = prospecting and Opportunity.account id not equal to null

Node 3 Criteria.: Opportunity Stage = Negotiation/Review and Opportunity Amount > 100,000

Node 4 Criteria.: Opportunity Stage = ClosedWon

Action for **Node 1 Email Alert** to mail notifies account creation : Finance: Account Creation.

Action for **Node 2 :**

Email Alert to mail notifies accountcreation : Finance:Account Creation.

Create a Record: Task with any name but mandatory subjectline 'Send MarketingMaterials'.

Makesure the string has no full stop or comma to it.

Assigned to the **Account owner**

Action for **Node 3**: Approvals

Choose the one we created for theopportunity here. And it takes care of the process thereby.

Action for **Node 4: Record** for Robot Setup

Set fields as below and Date formulabeing (closed date +180)

Challenge 6: **C**reate Flow for Opportunities:

Create **Flow** named **Product QuickSearch**

Create Flow for Opportunities

Element 1: Screen componentfrom the paletteName:**Product Quick Search**

Add **Record Button** fromthe Input as below:

Label: **Product Type**

Data Type: Text

Required: Check

Under Choices: Add new resource

Type:Choice

Create three choices as below for **RainbowBot**, **CloudyBot,and Assemble Systems.Element 2**: **Get Record**

Label: Search Prod select object as Product.

Under RecordCollection: Add New Resource **Filterresult**: **VariableElement3: Loop**

Add New Resource **Loop : Variable Type**

Element 4: Assignment

Add New Resource **Looptxt1 : Variable TypeElement 5: Screen**

Save and Activate the flow.

Now search **Lightning App Builder**Add **New page**: Select Record Type ***Label:***

Product_Quick_Search Object:
Opportunity
Pick any template

And Drag and drop Flows from Left palette, select the flowwe made and Save.

Challenge 7: **Automate Setups:**

Search for the field “Day of the Week” on robot object and change the field type from Numberto formula field of return type: text and use the below formula:

```
[FORMU
LA ]
CASE(
MOD([Opportunity].CloseDate + 180 - DATE(1900, 1, 7),7),
 0, [Opportunity].CloseDate + 181,
 6, [Opportunity].CloseDate + 182,
 [Opportunity].CloseDate + 180
)
```

By Changingthis Formula ,Savingand Activing The Process BuilderWe Can Complete This Challenge.