APEX SPECIALIST SUPERBADGE

CreateDefaultData

```
public with sharing class CreateDefaultData{
   Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine Maintenance';
   //gets value from custom metadata How_We_Roll_Settings__mdt to know if Default data was created
   @AuraEnabled
   public static Boolean isDataCreated() {
      How_We_Roll_Settings__c customSetting = How_We_Roll_Settings__c.getOrgDefaults();
      return customSetting.Is_Data_Created__c;
   }

   //creates Default Data for How We Roll application
   @AuraEnabled
   public static void createDefaultData(){
      List<Vehicle__c> vehicles = createVehicles();
      List<Product2> equipment = createEquipment();
      List<Case> maintenanceRequest = createMaintenanceRequest(vehicles);
      List<Equipment_Maintenance_Item__c> joinRecords = createJoinRecords(equipment,
maintenanceRequest);

      updateCustomSetting(true);
   }


   public static void updateCustomSetting(Boolean isDataCreated){
      How_We_Roll_Settings__c customSetting = How_We_Roll_Settings__c.getOrgDefaults();
      customSetting.Is_Data_Created__c = isDataCreated;
      upsert customSetting;
   }

   public static List<Vehicle__c> createVehicles(){
      List<Vehicle__c> vehicles = new List<Vehicle__c>();
      vehicles.add(new Vehicle__c(Name = 'Toy Hauler RV', Air_Conditioner__c = true, Bathrooms__c = 1,
Bedrooms__c = 1, Model__c = 'Toy Hauler RV'));
      vehicles.add(new Vehicle__c(Name = 'Travel Trailer RV', Air_Conditioner__c = true, Bathrooms__c =
2, Bedrooms__c = 2, Model__c = 'Travel Trailer RV'));
      vehicles.add(new Vehicle__c(Name = 'Teardrop Camper', Air_Conditioner__c = true, Bathrooms__c
= 1, Bedrooms__c = 1, Model__c = 'Teardrop Camper'));
      vehicles.add(new Vehicle__c(Name = 'Pop-Up Camper', Air_Conditioner__c = true, Bathrooms__c =
1, Bedrooms__c = 1, Model__c = 'Pop-Up Camper'));
      insert vehicles;
      return vehicles;
   }

   public static List<Product2> createEquipment(){
      List<Product2> equipments = new List<Product2>();
```

```
        equipments.add(new Product2(Warehouse_SKU__c = '55d66226726b611100aaf741',name =
'Generator 1000 kW', Replacement_Part__c = true,Cost__c = 100 ,Maintenance_Cycle__c = 100));
        equipments.add(new Product2(name = 'Fuse 20B',Replacement_Part__c = true,Cost__c = 1000,
Maintenance_Cycle__c = 30  ));
        equipments.add(new Product2(name = 'Breaker 13C',Replacement_Part__c = true,Cost__c = 100  ,
Maintenance_Cycle__c = 15));
        equipments.add(new Product2(name = 'UPS 20 VA',Replacement_Part__c = true,Cost__c = 200  ,
Maintenance_Cycle__c = 60));
        insert equipments;
        return equipments;

    }

    public static List<Case> createMaintenanceRequest(List<Vehicle__c> vehicles){
        List<Case> maintenanceRequests = new List<Case>();
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(1).Id, Type =
TYPE_ROUTINE_MAINTENANCE, Date_Reported__c = Date.today()));
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(2).Id, Type =
TYPE_ROUTINE_MAINTENANCE, Date_Reported__c = Date.today()));
        insert maintenanceRequests;
        return maintenanceRequests;
    }

    public static List<Equipment_Maintenance_Item__c> createJoinRecords(List<Product2> equipment,
List<Case> maintenanceRequest){
        List<Equipment_Maintenance_Item__c> joinRecords = new
List<Equipment_Maintenance_Item__c>();
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(0).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(1).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c = equipment.get(2).Id,
Maintenance_Request__c = maintenanceRequest.get(1).Id));
        insert joinRecords;
        return joinRecords;

    }
}
```

CreateDefaultDataTest

```
@isTest
```

```
private class CreateDefaultDataTest {
   @isTest
   static void createData_test(){
      Test.startTest();
      CreateDefaultData.createDefaultData();
      List<Vehicle__c> vehicles = [SELECT Id FROM Vehicle__c];
      List<Product2> equipment = [SELECT Id FROM Product2];
      List<Case> maintenanceRequest = [SELECT Id FROM Case];
      List<Equipment_Maintenance_Item__c> joinRecords = [SELECT Id FROM
Equipment_Maintenance_Item__c];

      System.assertEquals(4, vehicles.size(), 'There should have been 4 vehicles created');
      System.assertEquals(4, equipment.size(), 'There should have been 4 equipment created');
      System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2 maintenance request
created');
      System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment maintenance items
created');

   }

   @isTest
   static void updateCustomSetting_test(){
      How_We_Roll_Settings__c customSetting = How_We_Roll_Settings__c.getOrgDefaults();
      customSetting.Is_Data_Created__c = false;
      upsert customSetting;

      System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom setting
How_We_Roll_Settings__c.Is_Data_Created__c should be false');

      customSetting.Is_Data_Created__c = true;
      upsert customSetting;

      System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting
How_We_Roll_Settings__c.Is_Data_Created__c should be true');

   }
}
```

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);


          }
        }
      }

      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;
    }
  }
}
```

MaintenanceRequestHelperTest

```
@istest
public with sharing class MaintenanceRequestHelperTest {

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

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

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

  PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
    case cs = new case(Type=REPAIR,
              Status=STATUS_NEW,
              Origin=REQUEST_ORIGIN,
              Subject=REQUEST_SUBJECT,
              Equipment__c=equipmentId,
              Vehicle__c=vehicleId);
    return cs;
  }
```

```
  PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id requestId){
    Equipment_Maintenance_Item__c wp = new Equipment_Maintenance_Item__c(Equipment__c =
equipmentId,
                                                Maintenance_Request__c = requestId);
    return wp;
  }


  @istest
  private static void testMaintenanceRequestPositive(){
    Vehicle__c vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    Product2 equipment = createEq();
    insert equipment;
    id equipmentId = equipment.Id;

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

    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,somethingToUpdate.id);
    insert workP;

    test.startTest();
    somethingToUpdate.status = CLOSED;
    update somethingToUpdate;
    test.stopTest();

    Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
            from case
            where status =:STATUS_NEW];

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

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

  @istest
```

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

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

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

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

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

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

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

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

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

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

        for(integer i = 0; i < 300; i++){
```

```
        requestList.add(createMaintenanceRequest(vehicleList.get(i).id, equipmentList.get(i).id));
      }
      insert requestList;

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

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

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

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

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

WarehouseCalloutService

```
public with sharing class WarehouseCalloutService implements Queueable {
  private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';

  //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 warehouse one');
         }
      }
   }

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

}
```

WarehouseCalloutServiceMock

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
   // implement http mock callout
   global static HttpResponse respond(HttpRequest request) {

      HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Ge
```

```
nerator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226726b61110
0aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b611100aaf7
43","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

WarehouseCalloutServiceTest

```
@IsTest
private class WarehouseCalloutServiceTest {
    // implement your mock callout test here
            @isTest
    static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
        System.assertEquals('55d66226726b611100aaf741', product2List.get(0).ProductCode);
        System.assertEquals('55d66226726b611100aaf742', product2List.get(1).ProductCode);
        System.assertEquals('55d66226726b611100aaf743', product2List.get(2).ProductCode);
    }
}
```

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 Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
        Test.stopTest();
        //Contains schedule information for a scheduled job. CronTrigger is similar to a cron job on UNIX
systems.
        // This object is available in API version 17.0 and later.
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');


    }
}
```

MaintenanceRequest

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

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    // Implement this 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;
    }
}

@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 - specify the response you want to send
      ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
      // end
      response.put('response_x', response_x);
  }
}


public class AnimalLocator{
   public static String getAnimalNameById(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');
   }
}


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


@isTest
private class AccountManagerTest {

   private static testMethod void getAccountTest1() {
      Id recordId = createTestRecord();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId
+'/contacts' ;
```

```
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);

    }

    // Helper method
        static Id createTestRecord() {
        // Create test record
        Account TestAcc = new Account(
          Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}


@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 );
    }
}



//Generated by wsdl2apex

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

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



//Generated by wsdl2apex

public class AsyncParkService {
  public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
    public String[] getValue() {
      ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
      return response.return_x;
    }
```

```
    }
    public class AsyncParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public String clientCertName_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
        public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation
continuation,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture) System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParkService.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
        }
    }
}



@RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                       FROM Account WHERE Id = :accId];
        return acc;
    }
}



@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock ());
        String country = 'United States';
```

```
        List<String> result = ParkLocator.country(country);
        List<String> parks = new List<String>{'Yellowstone', 'Mackinac National Park', 'Yosemite'};
         System.assertEquals(parks, result);
    }
}
```

```
<apex:page >
  <apex:pageBlock title="User Status">
    <apex:pageBlockSection columns="1">
      {! $User.FirstName & ' ' & $User.LastName }
      ({! IF($User.isActive, $User.Username, 'inactive') })
    </apex:pageBlockSection>
  </apex:pageBlock>
</apex:page>
```

```
<apex:page showHeader="false" sidebar="false" >
  <apex:image url="https://developer.salesforce.com/files/salesforce-developer-network-logo.png"/>
</apex:page>
```

```
global class DailyLeadProcessor implements Schedulable{
  global void execute(SchedulableContext ctx){
    List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

    if(leads.size() > 0){
      List<Lead> newLeads = new List<Lead>();

      for(Lead lead : leads){
        lead.LeadSource = 'DreamForce';
        newLeads.add(lead);
      }

      update newLeads;
    }
  }
}
```

```
public class AddPrimaryContact implements Queueable{

  private Contact con;
```

```
    private String state;

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

    public void execute(QueueableContext context){
        List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from contacts)
                                  from Account where BillingState = :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);
        }

        if(primaryContacts.size() > 0){
            insert primaryContacts;
        }
    }

}




public class AddPrimaryContact implements Queueable{

    private Contact con;
    private String state;

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

    public void execute(QueueableContext context){
        List<Account> accounts = [Select Id, Name, (Select FirstName, LastName, Id from contacts)
                                  from Account where BillingState = :state Limit 200];
        List<Contact> primaryContacts = new List<Contact>();

        for(Account acc:accounts){
            Contact c = con.clone();
            c.AccountId = acc.Id;
            primaryContacts.add(c);
        }
```

```
        if(primaryContacts.size() > 0){
            insert primaryContacts;
        }
    }

}



global class LeadProcessor implements Database.Batchable<sObject> {
    global Integer count = 0;

    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');
    }

    global void execute (Database.BatchableContext bc, List<Lead> L_List){
        List<lead> L_list_new = new List<lead>();

        for(lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
        update L_list_new;
    }

    global void finish(Database.BatchableContext bc){
        system.debug('count = ' + count);
    }

}




@isTest
public class LeadProcessorTest {

    @isTest
    public static void testit(){
        List<lead> L_list = new List<lead>();

        for(Integer i=0; i<200; i++){
            Lead L = new lead();
            L.LastName = 'name' + i;
            L.Company = 'Company';
```

```
            L.Status = 'Random Status';
            L_list.add(L);
        }
        insert L_list;

        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();

    }

}
```

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){

        List<Account> accountsToUpdate = new List<Account>();

        List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account Where Id in
:accountIds];

        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_Of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);

        }
        update accountsToUpdate;
    }

}
```

```
@IsTest
private class AccountProcessorTest {
    @Istest
    private static void testCountContacts(){
        Account newAccount = new Account(Name='Test Account');
        insert newAccount;

        Contact newContact1 = new Contact(FirstName='John',LastName='Doe',AccountId=newAccount.Id);
```

```
        insert newContact1;

        Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId=newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

        Test.startTest();
        AccountProcessor.countContacts(accountIds);
        Test.stopTest();
    }

}



@isTest
private class TestVerifyDate {

    @isTest static void Test_CheckDates_case1(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('01/05/2020'));
        System.assertEquals(date.parse('01/05/2020'), D);
    }

    @isTest static void Test_CheckDates_case2(){
        Date D = VerifyDate.CheckDates(date.parse('01/01/2020'), date.parse('05/05/2020'));
        System.assertEquals(date.parse('01/31/2020'), D);
    }

    @istest static void Test_DateWithin30Days_case1(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('12/30/2019'));
        System.assertEquals(false, flag);
    }

    @istest static void Test_DateWithin30Days_case2(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('02/02/2020'));
        System.assertEquals(false, flag);
    }

    @istest static void Test_DateWithin30Days_case3(){
        Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'), date.parse('01/15/2020'));
        System.assertEquals(true, flag);
    }

    @isTest static void Test_SetEndOfMonthDate(){
        Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
```

```
    }

}



public class RandomContactFactory {

    public static List<Contact> generateRandomContacts(Integer numcnt, string lastname){
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt = new Contact(FirstName = 'Test '+i, LastName = lastname);
            contacts.add(cnt);
        }
        return contacts;
    }

}



@isTest
public class AddPrimaryContactTest {

    static testmethod void testQueueable(){
        List<Account> testAccounts = new List<Account>();
        for(Integer i=0;i<50;i++){
            testAccounts.add(new Account(Name='Account '+i,BillingState='CA'));
        }
        for(Integer j=0;j<50;j++){
            testAccounts.add(new Account(Name='Account '+j,BillingState='NY'));
        }
        insert testAccounts;

        Contact testContact = new Contact(FirstName = 'John', LastName = 'Doe');
        insert testContact;

        AddPrimaryContact addit = new addPrimaryContact(testContact, 'CA');

        Test.startTest();
        system.enqueueJob(addit);
        Test.stopTest();

        System.assertEquals(50,[Select count() from Contact where accountId in (Select Id from Account
where BillingState='CA')]);
    }
```

```
}



@isTest
private class DailyLeadProcessorTest{
   //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
   public static String CRON_EXP = '0 0 0 2 6 ? 2022';

   static testmethod void testScheduledJob(){
      List<Lead> leads = new List<Lead>();

      for(Integer i = 0; i < 200; i++){
         Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i,
Status = 'Open - Not Contacted');
         leads.add(lead);
      }

      insert leads;

      Test.startTest();
      // Schedule the test job
      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new
DailyLeadProcessor());

      // Stopping the test will run the job synchronously
      Test.stopTest();
   }
}



@isTest
public class TestRestrictContactByName {

   @isTest static void Test_insertupdateContact(){
      Contact cnt = new Contact();
      cnt.LastName = 'INVALIDNAME';

      Test.startTest();
      Database.SaveResult result = Database.insert(cnt, false);
      Test.stopTest();

      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size() > 0);
```

```
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
    }
}
```

```
public class NewCaseListController {
    public List<Case> getNewCases(){
        List<Case> filterList = [Select Id, CaseNumber from Case where status = 'New'];
        return filterList;
    }

}
```

```
trigger RestrictContactByName on Contact (before insert) {

}
```

```
trigger AccountAddressTrigger on Account (before insert, before update) {

    for(Account account:Trigger.New){
        if(account.Match_Billing_Address__c == True){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }
}
```

```
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:page standardController="Opportunity">
      <apex:outputField value="{! Opportunity.Name }"/>
      <apex:outputField value="{! Opportunity.Amount }"/>
      <apex:outputField value="{! Opportunity.CloseDate }"/>
      <apex:outputField value="{! Opportunity.Account.Name }"/>
</apex:page>
```

```
<apex:page standardController="Account" recordSetVar="accounts" >
    <apex:repeat var="a" value="{!accounts}">
        <li>
            <apex:outputLink value="/{!a.Id}">
                <apex:outputText value="{!a.Name}">
                </apex:outputText>
            </apex:outputLink>
        </li>
    </apex:repeat>
</apex:page>
```

```
<apex:page standardController="Contact">
    <apex:pageBlock title="Contact Summary">
        <apex:pageBlockSection >
            First Name: {! Contact.FirstName } <br/>
            Last Name: {! Contact.LastName } <br/>
            Owner Email: {! Contact.Owner.Email }  <br/>
        </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

```
<apex:page controller="NewCaseListController">
```

```
  <apex:repeat var="case" value="{!newCases}">
    <apex:outputLink value="/{!case.ID}">
      <apex:outputText value="{!case.CaseNumber}">
      </apex:outputText>
    </apex:outputLink>
  </apex:repeat>
</apex:page>
```

```
<apex:page >
  <apex:pageBlock title="User Status">
    <apex:pageBlockSection columns="1">
      {! $User.FirstName }

    </apex:pageBlockSection>
  </apex:pageBlock>
</apex:page>
```

```
<apex:page >
  <apex:image alt="eye" title="eye"
          url="{!URLFOR($Resource.vfimagetest, 'cats/kitten1.jpg')}"/>
</apex:page>
```

```
<apex:page standardController="Contact">
      <head>
            <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Quick Start: Visualforce</title>
    <!-- Import the Design System style sheet -->
    <apex:slds />

      </head>
      <body>
            <apex:form >
    <apex:pageBlock title="New Contact">
     <!--Buttons -->
```

```
        <apex:pageBlockButtons >
          <apex:commandButton action="{!save}" value="Save"/>
        </apex:pageBlockButtons>
        <!--Input form -->
        <apex:pageBlockSection columns="1">
        <apex:inputField value="{!Contact.Firstname}"/>
        <apex:inputField value="{!Contact.Lastname}"/>
        <apex:inputField value="{!Contact.Email}"/>
       </apex:pageBlockSection>
     </apex:pageBlock>
     </apex:form>

       </body>

</apex:page>
```

```
<apex:page standardController="Contact">
   <apex:form >
   <apex:pageBlock title="Edit Contact">
      <apex:pageBlockSection >
         <apex:inputField value="{! Contact.FirstName }"/>
         <apex:inputField value="{! Contact.LastName }"/>
         <apex:inputField value="{! Contact.Email }"/>
      </apex:pageBlockSection>
      <apex:pageBlockButtons >
         <apex:commandButton action="{! save }" value="Save" />
      </apex:pageBlockButtons>
   </apex:pageBlock>
   </apex:form>
</apex:page>
```