AccountManager

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing 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;
    }
}
```

AccountManagerTest

```
@IsTest
private class AccountManagerTest{
    @isTest static void testAccountManager(){
        Id recordId = getTestAccountId();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri =
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;

        // Call the method to test
        Account  acc = AccountManager.getAccount();

        // Verify results
        System.assert(acc != null);
    }

    private static Id getTestAccountId(){
        Account acc = new Account(Name = 'TestAcc2');
        Insert acc;

        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
        Insert con;

        return acc.Id;
```

```
    }
}
```

AnimalLocator

```
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');
    }
}
```

AnimalLocatorMock

```
@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('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

AnimalLocatorTest

```
@isTest
private class AnimalLocatorTest {

    @isTest
    static void testGetCallout() {

        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());

        String animalName = AnimalLocator.getAnimalNameById(1);
        system.debug('AnimalName : ' + animalName);
        System.assertEquals(animalName, 'chicken');
    }
}
```

AsyncParkService

```
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'}
            );
        }
    }
}
```

ParkLocator

```
public class ParkLocator {
    public  static string[] country(String country) {
        ParkService.ParksImplPort prk = new
ParkService.ParksImplPort();
        return prk.byCountry(country);
    }
}
```

ParkLocatorTest

```
@isTest
private class ParkLocatorTest {
    @isTest static void testCallout() {
        // This causes a fake response to be generated
        Test.setMock(WebServiceMock.class, new ParkServiceMock()
        // Call the method that invokes a callout


        String country = 'India';




        System.assertEquals(new List<String>{'Lal Bhag', 'Cubbon Pa
Dam'},     ParkLocator.country(country));
    }
}
```

ParkService

```
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;
        }
    }
}
```

ParkServiceMock

```
@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>{'Lal Bhag', 'Cubbon Park', 'Pazhassi Dam'};
              // end
```

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

ParkLocatorTest

```
@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>{'Lal Bhag', 'Cubbon Park', 'Pazhassi Dam'};
                // end
        response.put('response_x', response_x);
   }
}
```

AccountProcessor

```
public class AccountProcessor {

    //Writting the countContacts method and marking it whit the @future label.
    @future
    public static void countContacts(Set<Id> accountIDs) {

        // Creating a list that will contain all those accounts that are referenced through the accounIDs list.
        List<Account> accounts = [SELECT Id, Number_of_Contacts__c, (SELECT id FROM Contacts) from Account where id
in :accountIDs];

        //Assigment from the total contact number to the Number_of_Contacts__c field for each account at accounts list.
        for( Account account : accounts ) {
          account.Number_of_Contacts__c = account.contacts.size();
        }

        //Updating all accounts in list
        update accounts;
```

```
    }
}
```

AccountProcessorTest

```
@isTest
public class AccountProcessorTest {

    @isTest
    public static void countContactsTest(){
        //Creating an account and inserting it
        Account account = New Account(Name = 'Account Number 1');
        insert account;

        //Creating some contacts related to the account and inserting them
        List<Contact> contacts = new List<Contact>();
        contacts.add(New Contact(lastname = 'Related Contact 1', AccountId = account.Id));
        contacts.add(New Contact(lastname = 'Related Contact 2', AccountId = account.Id));
        contacts.add(New Contact(lastname = 'Related Contact 3', AccountId = account.Id));
        contacts.add(New Contact(lastname = 'Related Contact 4', AccountId = account.Id));
        insert contacts;

        //Creating a List with account Ids to pass them throught the AccountProcessor.countContacts method
        Set<Id> accountIds = new Set<Id>();
        accountIds.add(account.id);

        //Starting Test:
        Test.startTest();

        //Calling the AccountProcessor.countContacts method
        AccountProcessor.countContacts(accountIds);

        //Finishing Test:
        Test.stopTest();
        Account ACC = [SELECT Number_of_Contacts__c FROM Account WHERE id = :account.Id LIMIT 1];

        //Setting Assert (We have to parse the account.Number_of_Contacts__c
        //to integer to avoid some comparasion error between decimal and integer)
        System.assertEquals( Integer.valueOf(ACC.Number_of_Contacts__c) , 4);
    }


}
```

AddPrimaryContact

```
public class AddPrimaryContact 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 ,(Select id,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;
        }
    }
}
```

AddPrimaryContactTe@isTest

```
public class AddPrimaryContactTest
{
    @isTest static void 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

```
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;
    }
  }
}
```

DailyLeadProcessorTest

```
@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();
  }
}
```

LeadProcessorTest

```
@isTest
public class LeadProcessorTest
{
  static testMethod void testMethod1()
  {
    List<Lead> lstLead = new List<Lead>();
    for(Integer i=0 ;i <200;i++)
    {
        Lead led = new Lead();
        led.FirstName ='FirstName';
        led.LastName ='LastName'+i;
        led.Company ='demo'+i;
        lstLead.add(led);
    }

    insert lstLead;
```

```
        Test.startTest();


            LeadProcessor obj = new LeadProcessor();
            DataBase.executeBatch(obj);

        Test.stopTest();
    }
}
```

RandomContactFactory

```
public class RandomContactFactory
{
  public static List<Contact> generateRandomContacts(integer numofContacts,string LastNameGen)
    {
      List<Contact> con= new List<Contact>();
      for(Integer i=0;i<numofContacts;i++)
      {
          LastNameGen='Test'+ i;
          Contact a=new Contact(FirstName=LastNameGen,LastName=LastNameGen);
            con.add(a);
        }
        return con;
    }
}
```

TestRestrictContactByName

```
@isTest
private class TestRestrictContactByName{
    @isTest static void TestLastNameInvalidName(){
        Contact c = new Contact(LastName = 'INVALIDNAME');
        upsert c;
        Test.startTest();
        Database.UpsertResult result = Database.upsert(c,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());
    }
}
```

TestVerifyDate

```
@isTest
public class TestVerifyDate {
    private static Date dateToday = date.today();
    private static Integer totalDays = Date.daysInMonth(dateToday.year(), dateToday.month());

    @isTest static void testOldDate(){
        Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(-1));
        System.assertEquals(date.newInstance(dateToday.year(), dateToday.month(), totalDays), dateTest);
    }
```

```
    @isTest static void testLessThan30Days(){
        Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(20));
        System.assertEquals(dateToday.addDays(20), dateTest);
    }

    @isTest static void testMoreThan30Days(){
        Date dateTest = VerifyDate.CheckDates(dateToday, dateToday.addDays(31));
        System.assertEquals(date.newInstance(dateToday.year(), dateToday.month(), totalDays), dateTest);
    }

}
```

VerifyDate

```
public class VerifyDate {

 //method to handle potential checks against two dates
 public static Date CheckDates(Date date1, Date date2) {
    //if date2 is within the next 30 days of date1, use date2.  Otherwise use the end of the month
    if(DateWithin30Days(date1,date2)) {
      return date2;
    } else {
      return SetEndOfMonthDate(date1);
    }
 }

 //method to check if date2 is within the next 30 days of date1
 @TestVisible private static Boolean DateWithin30Days(Date date1, Date date2) {
    //check for date2 being in the past
         if( date2 < date1) { return false; }

         //check that date2 is within (>=) 30 days of date1
         Date date30Days = date1.addDays(30); //create a date 30 days away from date1
    if( date2 >= date30Days ) { return false; }
    else { return true; }
 }

 //method to return the end of the month of a given date
 @TestVisible private static Date SetEndOfMonthDate(Date date1) {
    Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
    Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
    return lastDay;
 }

}
```

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 {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';

    // complete this method to make the callout (using @future) to the
    // REST endpoint and update equipment on hand.
    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setMethod('GET');
        request.setEndpoint(WAREHOUSE_URL);
        HttpResponse response = http.send(request);
        if(response.getStatusCode() == 200) {
            List<Object> jsonResponse = (List<Object>)JSON.deserializeUntyped(response.getBody());
            system.debug('~~ '+jsonResponse);
            List<Product2> productList = new List<Product2>();
            for(Object ob : jsonResponse) {
                Map<String,Object> mapJson = (Map<String,Object>)ob;
                Product2 pr = new Product2();
                pr.Replacement_Part__c = (Boolean)mapJson.get('replacement');
                pr.Name = (String)mapJson.get('name');
                pr.Maintenance_Cycle__c = (Integer)mapJson.get('maintenanceperiod');
                pr.Lifespan_Months__c = (Integer)mapJson.get('lifespan');
                pr.Cost__c = (Decimal) mapJson.get('lifespan');
                pr.Warehouse_SKU__c = (String)mapJson.get('sku');
                pr.Current_Inventory__c = (Double) mapJson.get('quantity');
                productList.add(pr);
            }

            if(productList.size()>0)
                upsert productList;
        }

    }

}
```

WarehouseCalloutServiceMock

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

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

        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

WarehouseCalloutServiceTest

```
@isTest

private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

WarehouseSyncSchedule

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

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 ');


  }
}
```

Visualforce Page

ContactForm

```
<apex:page standardController="Contact">
         <head>
         </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 />
         <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>
```

NewCaseList

```
<apex:page controller="NewCaseListController">
    <apex:form >
        <apex:pageBlock title="Case List" id="Case_list">
         <apex:repeat value="{!NewCases}" var="case">
         <table style="width:1000px;">
```

```
<tr>
 <apex:repeat value="{!NewCases}" var="case">
           <apex:outputLink
value="https://ap2.salesforce.com/{!case.Id}">{!case.CaseNumber}</apex:outputLink>
         <apex:outputLink value="{!case.CaseNumber}">{!case.CaseNumber}</apex:outputLink>
        </apex:repeat>
             </tr>
        </table>
             </apex:repeat>
        </apex:pageBlock>
    </apex:form>
</apex:page>
```

ShowImage

```
<apex:page >
            <apex:pageBlock title="Pic" >
                    <apex:pageBlockSection >
                         <apex:image value="{!URLfor($Resource.vfimagetest,
'cats/kitten1.jpg')}"/>
                    </apex:pageBlockSection>
             </apex:pageBlock>
</apex:page>
```

AccountList

```
<apex:page standardController="Account" recordSetVar="Accounts">
    <apex:pageBlock >
        <apex:repeat var="a" value="{!Accounts}" rendered="true" id="account_list">
            <li>
                <apex:outputLink value="/{!a.ID}">
                    <apex:outputText value="{!a.Name}"/>
                </apex:outputLink>
            </li>
        </apex:repeat>
    </apex:pageBlock>
</apex:page>
```

CreateContact

```
<apex:page standardController="Contact">
    <apex:form >
        <apex:pageBlock title="Add contacts">
            <apex:pageBlockSection columns="1">
                <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>
```

OppView

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

ContactView

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

DisplayUserInfo

```
<apex:page >
    <apex:pageBlockSection columns="1">
        {! $User.FirstName} {! $User.LastName} {! $User.Username})
    </apex:pageBlockSection>
</apex:page>
```

DisplayImage

```
<apex:page showHeader="false" title="DisplayImage" sidebar="false">
    <apex:form >
        <table>
            <tr>
                <td width="1000px" height="600px" align="center">
                    <apex:image url="https://developer.salesforce.com/files/salesforce-
developer-network-logo.png"/>
                </td>
```

```
            </tr>
        </table>
    </apex:form>
</apex:page>
```