AccountManager

```
 @RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
   @HttpGet
   global static Account getAccount() {
      RestRequest request = RestContext.request;
      String accountId = request.requestURI.substringBetween('/Accounts/','/contacts');
      Account result =  [SELECT Id, Name, (SELECT Id, Name from contacts) from Account where
Id=:accountId];
                       return result;
   }
}
```

AccountManagerTest

```
@IsTest
private class AccountManagerTest {
   @isTest static void testGetContactsByAccountId() {
      Id recordId = createTestRecord();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://yourInstance.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 accountTest = new Account(
         Name='Test record');
         insert accountTest;
```

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

AccountProcessor

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


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

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

        For(Account acc:account){
            List<Contact> contactList =acc.Contacts;
            acc.Number_of_contacts__c = contactList.size();
            accountsToUpdate.add(acc);
        }
        update accountsToUpdate;
    }
}
```

AccountProcessorTest

```
@IsTest
public 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='John',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact2;

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

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

AddPrimaryContact

```
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);
      }
```

```
    }
}
```

AddPrimaryContactTest

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

AnimalLocator

```
public class AnimalLocator {

    public static String getAnimalNameById(Integer ID) {
        String animal = ' ';
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        String s = string.valueOf(ID);
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+ ID);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        Map<String,Object> animals = new Map<String,Object>();

        if(response.getStatusCode() == 200) {

            Map<String,Object> results =
(Map<String,Object>)JSON.deserializeUntyped(response.getBody());
            animals = (Map<String,Object>) results.get('animal');
            animal = String.valueOf(animals.get('name'));
        } else {

            system.debug(response.getBody());

        }
        Return animal;

    }

}
```

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'}
        );
      }
   }
}
```

ContactsTodayController

```
public class ContactsTodayController {

   @AuraEnabled
   public static List<Contact> getContactsForToday() {

      List<Task> my_tasks = [SELECT Id, Subject, WhoId FROM Task WHERE OwnerId =
:UserInfo.getUserId() AND IsClosed = false AND WhoId != null];
      List<Event> my_events = [SELECT Id, Subject, WhoId FROM Event WHERE OwnerId =
:UserInfo.getUserId() AND StartDateTime >= :Date.today() AND WhoId != null];
      List<Case> my_cases = [SELECT ID, ContactId, Status, Subject FROM Case WHERE OwnerId
= :UserInfo.getUserId() AND IsClosed = false AND ContactId != null];

      Set<Id> contactIds = new Set<Id>();
      for(Task tsk : my_tasks) {
         contactIds.add(tsk.WhoId);
      }
      for(Event evt : my_events) {
         contactIds.add(evt.WhoId);
      }
      for(Case cse : my_cases) {
         contactIds.add(cse.ContactId);
```

```
    }

    List<Contact> contacts = [SELECT Id, Name, Phone, Description FROM Contact WHERE Id
IN :contactIds];

    for(Contact c : contacts) {
      c.Description = '';
      for(Task tsk : my_tasks) {
        if(tsk.WhoId == c.Id) {
          c.Description += 'Because of Task '''+tsk.Subject+'''\n';
        }
      }
      for(Event evt : my_events) {
        if(evt.WhoId == c.Id) {
          c.Description += 'Because of Event '''+evt.Subject+'''\n';
        }
      }
      for(Case cse : my_cases) {
        if(cse.ContactId == c.Id) {
          c.Description += 'Because of Case '''+cse.Subject+'''\n';
        }
      }
    }

    return contacts;
  }

}
```

ContactsTodayControllerTest

```
@IsTest
public class ContactsTodayControllerTest {

  @IsTest
  public static void testGetContactsForToday() {

    Account acct = new Account(
      Name = 'Test Account'
    );
    insert acct;

    Contact c = new Contact(
        AccountId = acct.Id,
        FirstName = 'Test',
        LastName = 'Contact'
    );
    insert c;

    Task tsk = new Task(
        Subject = 'Test Task',
        WhoId = c.Id,
        Status = 'Not Started'
    );
    insert tsk;

    Event evt = new Event(
        Subject = 'Test Event',
        WhoId = c.Id,
        StartDateTime = Date.today().addDays(5),
        EndDateTime = Date.today().addDays(6)
    );
    insert evt;

    Case cse = new Case(
        Subject = 'Test Case',
        ContactId = c.Id
    );
    insert cse;

    List<Contact> contacts = ContactsTodayController.getContactsForToday();
    System.assertEquals(1, contacts.size());
    System.assert(contacts[0].Description.containsIgnoreCase(tsk.Subject));
    System.assert(contacts[0].Description.containsIgnoreCase(evt.Subject));
    System.assert(contacts[0].Description.containsIgnoreCase(cse.Subject));

}

@IsTest
public static void testGetNoContactsForToday() {
        Account acct = new Account(
            Name = 'Test Account'
        );
        insert acct;

        Contact c = new Contact(
            AccountId = acct.Id,
            FirstName = 'Test',
            LastName = 'Contact'
        );
        insert c;

        Task tsk = new Task(
            Subject = 'Test Task',
            WhoId = c.Id,
            Status = 'Completed'
        );
        insert tsk;

        Event evt = new Event(
            Subject = 'Test Event',
            WhoId = c.Id,
            StartDateTime = Date.today().addDays(-6),
            EndDateTime = Date.today().addDays(-5)
        );
        insert evt;

        Case cse = new Case(
            Subject = 'Test Case',
            ContactId = c.Id,
            Status = 'Closed'
        );
        insert cse;

        List<Contact> contacts = ContactsTodayController.getContactsForToday();
        System.assertEquals(0, contacts.size());

    }

}
```

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();
    }
}
```

NewCaseListController

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

}
```

PagedResult

```
public with sharing class PagedResult {
    @AuraEnabled
    public Integer pageSize { get; set; }

    @AuraEnabled
    public Integer pageNumber { get; set; }

    @AuraEnabled
    public Integer totalItemCount { get; set; }

    @AuraEnabled
    public Object[] records { get; set; }
}
```

ParkLocator

```
public class ParkLocator {
    public static string[] country(String country) {
        parkService.parksImplPort park = new parkService.parksImplPort();
        return park.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
        //Double x = 1.0;
        //Double result = AwesomeCalculator.add(x, y);

        String country = 'Germany';
        String[] result = ParkLocator.Country(country);


        // Verify that a fake result is returned
        System.assertEquals(new List<String>{'Hamburg Wadden Sea National Park', 'Hainich
National Park', 'Bavarian Forest National Park'}, result);
    }
}
```

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>{'Hamburg Wadden Sea National Park', 'Hainich
National Park', 'Bavarian Forest National Park'};

      //calculatorServices.doAddResponse response_x = new
calculatorServices.doAddResponse();
      //response_x.return_x = 3.0;
      // end
      response.put('response_x', response_x);
  }
}
```

PropertyController

```
public with sharing class PropertyController {
   private static final Decimal DEFAULT_MAX_PRICE = 9999999;
   private static final Integer DEFAULT_PAGE_SIZE = 9;

   /**
    * Endpoint that retrieves a paged and filtered list of properties
    * @param searchKey String used for searching on property title, city and tags
    * @param maxPrice Maximum price
    * @param minBedrooms Minimum number of bedrooms
    * @param minBathrooms Minimum number of bathrooms
```

```
 * @param pageSize Number of properties per page
 * @param pageNumber Page number
 * @return PagedResult object holding the paged and filtered list of properties
 */
@AuraEnabled(cacheable=true)
public static PagedResult getPagedPropertyList(
    String searchKey,
    Decimal maxPrice,
    Integer minBedrooms,
    Integer minBathrooms,
    Integer pageSize,
    Integer pageNumber
) {
    // Normalize inputs
    Decimal safeMaxPrice = (maxPrice == null
        ? DEFAULT_MAX_PRICE
        : maxPrice);
    Integer safeMinBedrooms = (minBedrooms == null ? 0 : minBedrooms);
    Integer safeMinBathrooms = (minBathrooms == null ? 0 : minBathrooms);
    Integer safePageSize = (pageSize == null
        ? DEFAULT_PAGE_SIZE
        : pageSize);
    Integer safePageNumber = (pageNumber == null ? 1 : pageNumber);

    String searchPattern = '%' + searchKey + '%';
    Integer offset = (safePageNumber - 1) * safePageSize;

    PagedResult result = new PagedResult();
    result.pageSize = safePageSize;
    result.pageNumber = safePageNumber;
    result.totalItemCount = [
        SELECT COUNT()
        FROM Property__c
        WHERE
            (Name LIKE :searchPattern
            OR City__c LIKE :searchPattern
            OR Tags__c LIKE :searchPattern)
            AND Price__c <= :safeMaxPrice
            AND Beds__c >= :safeMinBedrooms
            AND Baths__c >= :safeMinBathrooms
    ];
    result.records = [
        SELECT
            Id,
            Address__c,
            City__c,
            State__c,
            Description__c,
            Price__c,
            Baths__c,
            Beds__c,
            Thumbnail__c,
            Location__Latitude__s,
            Location__Longitude__s
        FROM Property__c
        WHERE
            (Name LIKE :searchPattern
            OR City__c LIKE :searchPattern
            OR Tags__c LIKE :searchPattern)
            AND Price__c <= :safeMaxPrice
            AND Beds__c >= :safeMinBedrooms
            AND Baths__c >= :safeMinBathrooms
        WITH SECURITY_ENFORCED
        ORDER BY Price__c
        LIMIT :safePageSize
        OFFSET :offset
    ];
    return result;
}

/**
 * Endpoint that retrieves pictures associated with a property
 * @param propertyId Property Id
 * @return List of ContentVersion holding the pictures
 */
@AuraEnabled(cacheable=true)
public static List<ContentVersion> getPictures(Id propertyId) {
    List<ContentDocumentLink> links = [
        SELECT Id, LinkedEntityId, ContentDocumentId
        FROM ContentDocumentLink
        WHERE
            LinkedEntityId = :propertyId
            AND ContentDocument.FileType IN ('PNG', 'JPG', 'GIF')
          WITH SECURITY_ENFORCED
        ];

        if (links.isEmpty()) {
          return null;
        }

        Set<Id> contentIds = new Set<Id>();

        for (ContentDocumentLink link : links) {
          contentIds.add(link.ContentDocumentId);
        }

        return [
          SELECT Id, Title
          FROM ContentVersion
          WHERE ContentDocumentId IN :contentIds AND IsLatest = TRUE
          WITH SECURITY_ENFORCED
          ORDER BY CreatedDate
        ];
      }
    }
```

AccountAddressTrigger

```
trigger AccountAddressTrigger on Account (before insert,before update) {
   for (account account:trigger.new){
       if(account.Match_Billing_Address__c == true){
           account.ShippingPostalCode = account.BillingPostalCode;
       }
   }
}
```

ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (before 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;
   }
}
```

RestrictContactByName

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


            For (Contact c : Trigger.New) {
                        if(c.LastName == 'INVALIDNAME') {        //invalidname is invalid
                                    c.AddError('The Last Name "'+c.LastName+'" is not allowed for
DML');
                        }

            }



}
```

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;
      }
    }
}
```

class 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';

    //@future(callout=true)
    public static void runWarehouseEquipmentSync(){

        Http http = new Http();
        HttpRequest request = new HttpRequest();

        request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);


        List<Product2> warehouseEq = new List<Product2>();

        if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }

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

        }
    }
}
```

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 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 ');


    }
}
```

trigger MaintenanceRequest

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