Quick Start: Lightning Web Components

helloWorld.html

```
<template>
  <lightning-card title="HelloWorld" icon-name="custom:custom14">
    <div class="slds-m-around_medium">
      <p>Hello, {greeting}!</p>
      <lightning-input label="Name" value={greeting}
onchange={changeHandler}></lightning-input>
    </div>
  </lightning-card>
</template>
```

helloWorld.js

```
import { LightningElement } from 'lwc';
export default class HelloWorld extends LightningElement {
  greeting = 'World';
  changeHandler(event) {
    this.greeting = event.target.value;
  }
}
```

helloWorld.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata"
fqn="helloWorld">
  <apiVersion>52.0</apiVersion>
  <isExposed>true</isExposed>
  <targets>
    <target>lightning__AppPage</target>
    <target>lightning__RecordPage</target>
    <target>lightning__HomePage</target>
  </targets>
</LightningComponentBundle>
```

Apex Triggers

AccountAddressTrigger.apxt

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

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> tasklist = new List<Task>();

    for(Opportunity opp: Trigger.New){
        if(opp.StageName == 'Closed won'){
            tasklist.add(new Task(Subject = 'Follow Up Test Task',WhatId = opp.Id));

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

Apex Testing

VerifyDate.apxc

```
public 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;
        }

}
```

TestVeriyDate.apxc

```
@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('02/02/2019'));
      System.assertEquals(false, flag);
   }
   @isTest static void Test_Datewithin30Days_case3(){
      Boolean flag =
VerifyDate.Datewithin30Days(date.parse('01/01/2020'),date.parse('01/15/2019'));
      System.assertEquals(true, flag);
   }
   @isTest static void Test_SetEndOfMonthDate(){
      Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
   }

}
```

RestrictContactByName.apxt

```
trigger RestrictContactByName on Contact (before insert, before update) {
                //check contacts prior to insert or update for invalid data
                For (Contact c : Trigger.New) {
                  if(c.LastName == 'INVALIDNAME') {        //invalidname is invalid
                            c.AddError('The Last Name "'+c.LastName+'" is not allowed for
DML');
                  }
                }
}
```

TestRestrictContactByName.apxc

```
@isTest
public class TestRestrictContactByName {
   @isTest
   public static void testContact(){
      Contact ct = new Contact();
      ct.LastName = 'INVALIDNAME';
      Database.SaveResult res = Database.insert(ct,false);
     System.assertEquals('The Last Name "INVLIDNAME" is not allowed for DMl',
res.getErrors()[0].getMessage());

   }
}
```

RandomContactFactory.apxc

```
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;
   }
}
```

Asynchronous Apex

AccountProcessor.apxc

```
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;
   }
}
```

AccountProcessorTest.apxc

```
@isTest
public class AccountProcessorTest{
   public static testmethod void testAccountProcessor(){
      Account a = new Account();
      a.Name = 'Test Account';
      insert a;
      Contact con = new Contact();
      con.FirstName = 'dilip';
      con.LastName = 'byella';
      con.AccountId = a.Id;
      Insert con;
      List<Id> accListId = new List<Id>();
      accListId.add(a.Id);
      Test.startTest();
      AccountProcessor.countContacts(accListId);
      Test.stopTest();
      Account acc = [Select Number_Of_Contacts__c from Account where Id =: a.Id];
      System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c),1);
   }
}
```

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable{
    Contact con;
    String state;
    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account WHERE BillingState = :state
LIMIT 200];
        List<Contact> lstOfConts = new List<Contact>();
        for(Account acc : lstOfAccs){
            Contact conInst = con.clone(false,false,false,false);
            conInst.AccountId = acc.Id;
            lstOfConts.add(conInst);
        }
        INSERT lstOfConts;
    }
}
```

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> lstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'NY'));
            else
                lstOfAcc.add(new Account(name='AC'+i, BillingState = 'CA'));
        }
        INSERT lstOfAcc;
    }
    static testmethod void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new AddPrimaryContact(CON ,'CA');
        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();
        System.assertEquals(50, [select count() from Contact]);
    }
}
```

DailyLeadProcessor.apxc

```
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.apxc

```
@isTest
private class DailyLeadProcessorTest{
   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();
      String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP,
new DailyLeadProcessor());
      // Stopping the test will run the job synchronously
      Test.stopTest();
   }
}
```

LeadProcessor.apxc

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

LeadProcessorTest.apxc

```
@isTest
private class LeadProcessorTest {
   @isTest
   private static void testBatchClass(){
      List<lead> leads = new List<Lead>();
      for(Integer i=0; i<20;i++){
         leads.add(new Lead(LastName = 'Connock', Company = 'Salesforce'));
      }
      insert leads;
      Test.startTest();
      LeadProcessor lp = new LeadProcessor();
      Id batchId = Database.executeBatch(lp,200);
      Test.stopTest();
      List<Lead> updatedLeads = [SELECT Id FROM Lead WHERE LeadSource
='Dreamforce'];
      System.assertEquals(200,updatedLeads.size(), 'ERROr: At least 1 Lead record not
updated correctly ');
   }
}
```

Apex Integration Services

AccountManager.apxc

```
@RestResource(urlMapping = '/Accounts/*/contacts')
global with sharing class AccountManager
{
@HttpGet
   global static Account getAccount()
   {
     RestRequest request = RestContext.request;
     String accId = request.requestURI.substringBetween('Accounts/','/contacts');
     system.debug(accId);
     Account acct = [SELECT Id,Name,(SELECT Id,Name FROM Contacts) FROM Account
WHERE Id = :accId LIMIT 1];
     return acct;
   }
}
```

AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest
{
   static testMethod void testMethod1()
   {
      Account accTst = new Account(Name = 'Test Account');
      insert accTst;
      Contact conTst = new Contact(LastName = 'Test Contact', AccountId = accTst.Id);
      insert conTst;
      ID recId = accTst.Id;
      RestRequest request = new RestRequest();
      request.requestUri =
'https://login.salesforce.com/services/apexrest/Accounts/'+recId+'/contacts';
      request.httpMethod = 'GET';
      RestContext.request = request;
      Account thisAcc = AccountManager.getAccount();
      system.debug(thisAcc);
      system.assert(thisAcc != null);
      system.assertEquals('Test Account', thisAcc.Name);
   }
}
```

AnimalLocator.apxc

```
public class AnimalLocator {
  public class cls_animal {
        public Integer id;
        public String name;
        public String eats;
        public String says;
  }
public class JSONOutput{
  public cls_animal animal;

  //public JSONOutput parse(String json){
  //return (JSONOutput) System.JSON.deserialize(json, JSONOutput.class);
  //}
}
   public static String getAnimalNameById (Integer id) {
      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
      //request.setHeader('id', String.valueof(id)); -- cannot be used in this challenge :)
      request.setMethod('GET');
      HttpResponse response = http.send(request);
      system.debug('response: ' + response.getBody());
      //Map<String,Object> map_results = (Map<String,Object>)
JSON.deserializeUntyped(response.getBody());
      jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(),
jsonOutput.class);
      //Object results = (Object) map_results.get('animal');
         system.debug('results= ' + results.animal.name);
      return(results.animal.name);
   }

}
```

AnimalLocatorMock.apxc

```
@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.apxc

```
@isTest
public class AnimalLocatorTest {
  @isTest public static void AnimalLocatorMock() {
      //Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
     test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
      string result = AnimalLocator.getAnimalNameById(1);
     system.debug(result);
      String expectedResult = 'chicken';
      System.assertEquals(result,expectedResult );
   }
}
```

AsyncparksService.apxc

```
//Generated by wsdl2apex
public class AsyncParksService {
   public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
      public String[] getValue() {
         parksService.byCountryResponse response =
(parksService.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/',
'parksService'};
      public AsyncParksService.byCountryResponseFuture
beginByCountry(System.Continuation continuation,String arg0) {
         parksService.byCountry request_x = new parksService.byCountry();
         request_x.arg0 = arg0;
         return (AsyncParksService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
           this,
           request_x,
           AsyncParksService.byCountryResponseFuture.class,continuation,
           new String[]{endpoint_x,
           '',
           'http://parks.services/','byCountry',
           'http://parks.services/',
           'byCountryResponse',
           'parksService.byCountryResponse'}
         );
      }
   }
}
```

ParkLocator.apxc

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

ParkLocatorTest.apxc

```
@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
      List<String> result = new List<String>();
      List<String> expectedvalue = new List<String>{'Park1','Park2','Park3'};
      result = ParkLocator.country('India');
      // Verify that a fake result is returned
      System.assertEquals(expectedvalue, result);
   }
}
```

ParkService.apxc

```
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.apxc

```
@isTest
global class ParkServiceMock implements WebServiceMock {
  global void doInvoke(
        Object stub,
        Object request,
        Map<String, Object> response,
        String endpoint,
        String soapAction,
        String requestName,
        String responseNS,
        String responseName,
        String responseType) {
      // start - specify the response you want to send
      ParkService.byCountryResponse response_x =
         new ParkService.byCountryResponse();
      List<String> myStrings = new List<String> {'Park1','Park2','Park3'};
      response_x.return_x = myStrings;
      // end
      response.put('response_x', response_x);
  }
}
```

parksService.apxc

```
//Generated by wsdl2apex

public class parksService {
    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/',
'parksService'};
        public String[] byCountry(String arg0) {
            parksService.byCountry request_x = new parksService.byCountry();
            request_x.arg0 = arg0;
            parksService.byCountryResponse response_x;
            Map<String, parksService.byCountryResponse> response_map_x = new
Map<String, parksService.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',
          'parksService.byCountryResponse'}
        );
        response_x = response_map_x.get('response_x');
        return response_x.return_x;
      }
    }
}
```

parksServices.apxc

```
//Generated by wsdl2apex

public class parksServices {
   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/',
'parksServices'};
      public String[] byCountry(String arg0) {
         parksServices.byCountry request_x = new parksServices.byCountry();
         request_x.arg0 = arg0;
         parksServices.byCountryResponse response_x;
         Map<String, parksServices.byCountryResponse> response_map_x = new
Map<String, parksServices.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',
          'parksServices.byCountryResponse'}
        );
        response_x = response_map_x.get('response_x');
        return response_x.return_x;
      }
    }
}
```

Lightning Web Component Basics

bikeCard.html

```
<template>
   <div>
      <div>Name: (name)</div>
      <div>Description: (description)</div>
      <lghtining-badge label={material}></lghtining-badge>
      <lghtining-badge label={category}></lghtining-badge>
      <div>Price: (price)</div>
      <div><img src={pictureUrl}</div>
   </div>
</template>
```

bikeCard.js

```
import { LightningElement } from 'lwc';
export default class BikeCard extends LightningElement {
  name = 'Electra X4';
  description = 'A sweet bike built for comfort.';
  category = 'Mountain';
  material = 'Steel';
  price = '$2,700';
  pictureUrl = 'https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/electrax4.jpg';
 }
```

bikeCard.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
   <!-- The apiVersion may need to be increased for the current release -->
   <apiVersion>52.0</apiVersion>
   <isExposed>true</isExposed>
   <masterLabel>Product Card</masterLabel>
   <targets>
      <target>lightning__AppPage</target>
      <target>lightning__RecordPage</target>
      <target>lightning__HomePage</target>
   </targets>
</LightningComponentBundle>
```

data.js

export const bikes = [

{"apiName":"Product__c","childRelationships":{},"fields":{"Category__c":{"displayValue":"Mountain","value":"Mountain"},"CreatedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Description__c":{"displayValue":null,"value":"A durable e-bike with great looks."},"Id":{"displayValue":null,"value":"a0256000001F1arAAC"},"LastModifiedDate":{"displayValue":null,"value":"2018-10-12T02:57:48.000Z"},"Level__c":{"displayValue":"Racer","value":"Racer"},"MSRP__c":{"displayValue":"$7,800","value":7800},"Material__c":{"displayValue":"Carbon","value":"Carbon"},"Name":{"displayValue":null,"value":"DYNAMO X1"},"Picture_URL__c":{"displayValue":null,"value":"https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/dynamox1.jpg"},"SystemModstamp":{"displayValue":null,"value":"2018-10-12T02:57:48.000Z"}},"id":"a0256000001F1arAAC","lastModifiedById":null,"lastModifiedDate":"2018-10-12T02:57:48.000Z","recordTypeInfo":null,"systemModstamp":"2018-10-12T02:57:48.000Z"},

{"apiName":"Product__c","childRelationships":{},"fields":{"Category__c":{"displayValue":"Mountain","value":"Mountain"},"CreatedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Description__c":{"displayValue":null,"value":"A durable e-bike with great looks."},"Id":{"displayValue":null,"value":"a0256000001F1atAAC"},"LastModifiedDate":{"displayValue":null,"value":"2018-10-10T17:26:47.000Z"},"Level__c":{"displayValue":"Racer","value":"Racer"},"MSRP__c":{"displayValue":"$6,802","value":6802},"Material__c":{"displayValue":"Aluminum","value":"Aluminum"},"Name":{"displayValue":null,"value":"DYNAMO X2"},"Picture_URL__c":{"displayValue":null,"value":"https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/dynamox2.jpg"},"SystemModstamp":{"displayValue":null,"value":"2018-10-10T17:26:47.000Z"}},"id":"a0256000001F1atAAC","lastModifiedById":null,"lastModifiedDate":"2018-10-10T17:26:47.000Z","recordTypeInfo":null,"systemModstamp":"2018-10-10T17:26:47.000Z"},

{"apiName":"Product__c","childRelationships":{},"fields":{"Category__c":{"displayValue":"Mountain","value":"Mountain"},"CreatedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Description__c":{"displayValue":null,"value":"A durable e-bike with great looks."},"Id":{"displayValue":null,"value":"a0256000001F1auAAC"},"LastModifiedDate":{"displayValue":null,"value":"2018-10-

09T04:37:56.000Z"},"Level__c":{"displayValue":"Enthusiast","value":"Enthusiast"},"MSRP__c":{"displayValue":"$5,601","value":5601},"Material__c":{"displayValue":"Aluminum","value":"Aluminum"},"Name":{"displayValue":null,"value":"DYNAMO X3"},"Picture_URL__c":{"displayValue":null,"value":"https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/dynamox3.jpg"},"SystemModstamp":{"displayValue":null,"value":"2018-10-09T04:37:56.000Z"}},"id":"a0256000001F1auAAC","lastModifiedById":null,"lastModifiedDate":"2018-10-09T04:37:56.000Z","recordTypeInfo":null,"systemModstamp":"2018-10-09T04:37:56.000Z"},

{"apiName":"Product__c","childRelationships":{},"fields":{"Category__c":{"displayValue":"Mountain","value":"Mountain"},"CreatedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Description__c":{"displayValue":null,"value":"A durable e-bike with great looks."},"Id":{"displayValue":null,"value":"a0256000001F1avAAC"},"LastModifiedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Level__c":{"displayValue":"Enthusiast","value":"Enthusiast"},"MSRP__c":{"displayValue":"$5,500","value":5500},"Material__c":{"displayValue":"Aluminum","value":"Aluminum"},"Name":{"displayValue":null,"value":"DYNAMO X4"},"Picture_URL__c":{"displayValue":null,"value":"https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/dynamox4.jpg"},"SystemModstamp":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"}},"id":"a0256000001F1avAAC","lastModifiedById":null,"lastModifiedDate":"2018-10-09T03:29:52.000Z","recordTypeInfo":null,"systemModstamp":"2018-10-09T03:29:52.000Z"},

{"apiName":"Product__c","childRelationships":{},"fields":{"Category__c":{"displayValue":"Mountain","value":"Mountain"},"CreatedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Description__c":{"displayValue":null,"value":"A durable e-bike with great looks."},"Id":{"displayValue":null,"value":"a0256000001F1azAAC"},"LastModifiedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Level__c":{"displayValue":"Enthusiast","value":"Enthusiast"},"MSRP__c":{"displayValue":"$4,600","value":4600},"Material__c":{"displayValue":"Aluminum","value":"Aluminum"},"Name":{"displayValue":null,"value":"FUSE X1"},"Picture_URL__c":{"displayValue":null,"value":"https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/fusex1.jpg"},"SystemModstamp":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"}},"id":"a0256000001F1azAAC","lastModifiedById":null,"lastModifiedDate":"2018-10-09T03:29:52.000Z","recordTypeInfo":null,"systemModstamp":"2018-10-09T03:29:52.000Z"},

{"apiName":"Product__c","childRelationships":{},"fields":{"Category__c":{"displayValue":"Commuter","value":"Commuter"},"CreatedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Description__c":{"displayValue":null,"value":"A durable e-bike with great looks."},"Id":{"displayValue":null,"value":"a0256000001F1b2AAC"},"LastModifiedDate":{"displayValue":null,"value":"2018-10-09T04:41:56.000Z"},"Level__c":{"displayValue":"Beginner","value":"Beginner"},"MSRP__c":{"displayValue":"$3,200","value":3200},"Material__c":{"displayValue":"Aluminum","value":"Aluminum"},"Name":{"displayValue":null,"value":"ELECTRA X1"},"Picture_URL__c":{"displayValue":null,"value":"https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/electrax1.jpg"},"SystemModstamp":{"displayValue":null,"value":"2018-10-09T04:41:56.000Z"}},"id":"a0256000001F1b2AAC","lastModifiedById":null,"lastModifiedDate":"2018-10-09T04:41:56.000Z","recordTypeInfo":null,"systemModstamp":"2018-10-09T04:41:56.000Z"},

{"apiName":"Product__c","childRelationships":{},"fields":{"Category__c":{"displayValue":"Commuter","value":"Commuter"},"CreatedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Description__c":{"displayValue":null,"value":"A durable e-bike with great looks."},"Id":{"displayValue":null,"value":"a0256000001F1b3AAC"},"LastModifiedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Level__c":{"displayValue":"Beginner","value":"Beginner"},"MSRP__c":{"displayValue":"$3,200","value":3200},"Material__c":{"displayValue":"Aluminum","value":"Aluminum"},"Name":{"displayValue":null,"value":"ELECTRA X2"},"Picture_URL__c":{"displayValue":null,"value":"https://s3-us-west-1.amazonaws.com/sfdc-demo/ebikes/electrax2.jpg"},"SystemModstamp":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"}},"id":"a0256000001F1b3AAC","lastModifiedById":null,"lastModifiedDate":"2018-10-09T03:29:52.000Z","recordTypeInfo":null,"systemModstamp":"2018-10-09T03:29:52.000Z"},

{"apiName":"Product__c","childRelationships":{},"fields":{"Category__c":{"displayValue":"Commuter","value":"Commuter"},"CreatedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Description__c":{"displayValue":null,"value":"A durable e-bike with great looks."},"Id":{"displayValue":null,"value":"a0256000001F1b6AAC"},"LastModifiedDate":{"displayValue":null,"value":"2018-10-09T03:29:52.000Z"},"Level__c":{"displayValue":"Beginner","value":"Beginner"},"MSRP__c":{"displayValue":"$2,700","value":2700},"Material__c":{"displayValue":"Aluminum","value":"Alu

minum"},"Name":{"displayValue":null,"value":"ELECTRA
X3"},"Picture_URL__c":{"displayValue":null,"value":"https://s3-us-west-
1.amazonaws.com/sfdc-
demo/ebikes/electrax3.jpg"},"SystemModstamp":{"displayValue":null,"value":"2018-10-
09T03:29:52.000Z"}},"id":"a0256000001F1b6AAC","lastModifiedById":null,"lastModifiedD
ate":"2018-10-09T03:29:52.000Z","recordTypeInfo":null,"systemModstamp":"2018-10-
09T03:29:52.000Z"},

{"apiName":"Product__c","childRelationships":{},"fields":{"Category__c":{"displayValue":"Co
mmuter","value":"Commuter"},"CreatedDate":{"displayValue":null,"value":"2018-10-
09T03:29:52.000Z"},"Description__c":{"displayValue":null,"value":"A durable e-bike with
great
looks."},"Id":{"displayValue":null,"value":"a0256000001F1b7AAC"},"LastModifiedDate":{"dis
playValue":null,"value":"2018-10-
09T03:29:52.000Z"},"Level__c":{"displayValue":"Beginner","value":"Beginner"},"MSRP__c":{"
displayValue":"$2,700","value":2700},"Material__c":{"displayValue":"Aluminum","value":"Alu
minum"},"Name":{"displayValue":null,"value":"ELECTRA
X4"},"Picture_URL__c":{"displayValue":null,"value":"https://s3-us-west-
1.amazonaws.com/sfdc-
demo/ebikes/electrax4.jpg"},"SystemModstamp":{"displayValue":null,"value":"2018-10-
09T03:29:52.000Z"}},"id":"a0256000001F1b7AAC","lastModifiedById":null,"lastModifiedD
ate":"2018-10-09T03:29:52.000Z","recordTypeInfo":null,"systemModstamp":"2018-10-
09T03:29:52.000Z"}
];

data.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>false</isExposed>
</LightningComponentBundle>
```

detail.css

```
ebody{
  margin: 0;
}
```

detail.html

```
<template>
  <template if:true={product}>
    <div class="container">
      <div>{product.fields.Name.value}</div>
      <div class="price">{product.fields.MSRP__c.displayValue}</div>
      <div class="description">{product.fields.Description__c.value}</div>
      <img class="product-img" src={product.fields.Picture_URL__c.value}></img>
      <p>
        <lightning-badge label={product.fields.Material__c.value}></lightning-badge>
        <lightning-badge label={product.fields.Level__c.value}></lightning-badge>
      </p>
      <p>
        <lightning-badge label={product.fields.Category__c.value}></lightning-badge>
      </p>
    </div>
  </template>
  <template if:false={product}>
    <div>Select a bike</div>
  </template>
</template>
```

detail.js

```
import { LightningElement, api } from 'lwc';
import { bikes } from 'c/data';

export default class Detail extends LightningElement {
  // Ensure changes are reactive when product is updated
  product;
  // Private var to track @api productId
  _productId = undefined;
  // Use set and get to process the value every time it's
  // requested while switching between products
  set productId(value) {
    this._productId = value;
    this.product = bikes.find(bike => bike.fields.Id.value === value);
  }
  // getter for productId
  @api get productId(){
    return this._productId;
  }
}
```

detail.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>false</isExposed>
</LightningComponentBundle>
```

list.css

```
.container {
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
}
```

list.html

```
<template>
    <div class="container">
        <template for:each={bikes} for:item="bike">
            <c-tile
                key={bike.fields.Id.value}
                product={bike}
                ontileclick={handleTileClick}>
            </c-tile>
        </template>
    </div>
</template>
```

list.js

```
import { LightningElement } from 'lwc';
import { bikes } from 'c/data';
export default class List extends LightningElement {
    bikes = bikes;
    handleTileClick(evt) {
        // This component wants to emit a productselected event to its parent
        const event = new CustomEvent('productselected', {
            detail: evt.detail
        });
        // Fire the event from c-list
        this.dispatchEvent(event);
    }
}
```

list.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>false</isExposed>
</LightningComponentBundle>
```

selector.css

```
body {
  margin: 0;
}
.wrapper{
  min-height: 100vh;
  background: #ccc;
  display: flex;
  flex-direction: column;
}
.header, .footer{
  height: 50px;
  background: rgb(255, 255, 255);
  color: rgb(46, 46, 46);
  font-size: x-large;
  padding: 10px;
}
.content {
  display: flex;
  flex: 1;
  background: #999;
  color: #000;
}
.columns{
  display: flex;
  flex:1;
}
.main{
  flex: 1;
  order: 2;
  background: #eee;
```

```
}
.sidebar-first{
  width: 20%;
  background: #ccc;
  order: 1;
}
.sidebar-second{
  width: 30%;
  order: 3;
  background: #ddd;
}
```

selector.html

```
<template>
    <div class="wrapper">
    <header class="header">Select a Bike</header>
    <section class="content">
        <div class="columns">
        <main class="main" >
            <b>{name}</b>
            <c-list onproductselected={handleProductSelected}></c-list>
        </main>
        <aside class="sidebar-second">
            <c-detail product-id={selectedProductId}></c-detail>
        </aside>
        </div>
    </section>
    </div>
</template>
```

selector.js

```
import { LightningElement } from 'lwc';
export default class Selector extends LightningElement {
    selectedProductId;
    handleProductSelected(evt) {
        this.selectedProductId = evt.detail;
    }
}
```

selector.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__AppPage</target>
        <target>lightning__RecordPage</target>
        <target>lightning__HomePage</target>
    </targets>
</LightningComponentBundle>
```

tile.css

```
.container {
    border: 1px rgb(168, 166, 166) solid;
    border-radius: 5px;
    background-color: white;
    margin:5px;
    padding: 2px;
    max-width: 110px;
    display: flex;
}
.title {
    font-weight: strong;
}
.product-img {
    max-width: 100px;
}
a {
    text-decoration: none;
}
a:link {
    color: rgb(159, 159, 159);
}
a:visited {
    color: green;
}
a:hover {
```

```
    color: hotpink;
}
a:active {
    color: blue;
}
```

tile.html

```
<template>
    <div class="container">
        <a onclick={tileClick}>
            <div class="title">{product.fields.Name.value}</div>
            <img class="product-img" src={product.fields.Picture_URL__c.value}></img>
        </a>
    </div>
</template>
```

tile.js

```
 import { LightningElement, api } from 'lwc';

export default class Tile extends LightningElement {
    @api product;

    tileClick() {
        const event = new CustomEvent('tileclick', {
            // detail contains only primitives
            detail: this.product.fields.Id.value
        });
        // Fire the event from c-tile
        this.dispatchEvent(event);
    }
}
```

tile.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>48.0</apiVersion>
    <isExposed>false</isExposed>
</LightningComponentBundle>
```

Apex Basics and Database

StringArrayTest.apxc

```
public class StringArrayTest {
    public static List<String> generateStringArray(Integer N){
        List<String> TestList = new List<String>();
        for(Integer i=0;i<N;i++){
            TestList.add('Test ' + i);
            system.debug(TestList[i]);
        }
        return TestList;
    }
}
```

AccountHandler.apxc

```
public class AccountHandler {
    public static Account insertNewAccount(String AccountName){
        try {
             Account newacct = new Account(Name=AccountName);
             insert newacct;
             return newacct;
          } catch (DmlException e) {
            System.debug('A DML exception has occurred: ' +
              e.getMessage());
             return null;
}
    }

}
```

ContactSearch.apxc

```
public class ContactSearch{
    public static list<Contact> searchForContacts(string name1, string name2){
        List <Contact> con = new List<contact>();
        con = [SELECT ID,FirstName from Contact where LastName =:name1  and
MailingPostalCode=:name2];
        return con;
    }
}
```

ContactAndLeadSearch.apxc

```
public class ContactAndLeadSearch {

   //a public static method that accepts an incoming string as a parameter
   public static List<List<sObject>> searchContactsAndLeads (String incoming) {
                  //write a SOSQL query to search by lead or contact name fields for the
incoming string.
                  List<List<sObject>> searchList = [FIND :incoming IN NAME FIELDS
               RETURNING Contact(FirstName,LastName),Lead(FirstName,LastName)];
                  //return the list of the same kind
                  return searchList;
                }
}
```

Apex Specialist Superbadge

1. Automate Record Creation:

MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
   public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
      Set<Id> validIds = new Set<Id>();
      For (Case c : updWorkOrders){
         if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
               validIds.add(c.Id);
            }
         }
      }
      //When an existing maintenance request of type Repair or Routine Maintenance is
closed,
      //create a new maintenance request for a future routine checkup.
      if (!validIds.isEmpty()){
         Map<Id,Case> closedCases = 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>();
```

```
        //calculate the maintenance request due dates by using the maintenance cycle
defined on the related equipment records.
        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'));
        }

        List<Case> newCases = new List<Case>();
        for(Case cc : closedCases.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 multiple pieces of equipment are used in the maintenance request,
           //define the due date by applying the shortest maintenance cycle to today's
date.
           //If (maintenanceCycles.containskey(cc.Id)){
              nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
           //} else {
           //    nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
           //}

           newCases.add(nc);
        }

        insert newCases;
```

```
        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
           for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
              Equipment_Maintenance_Item__c item = clonedListItem.clone();
              item.Maintenance_Request__c = nc.Id;
              clonedList.add(item);
           }
        }
        insert clonedList;
     }
  }
}
```

MaintenanceRequest.apxt

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

2. Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc

```
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();
  }

}
```

3.Schedule synchronization using Apex code

WarehouseSyncSchedule

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

4.Test automation logic

MaintenanceRequestHelperTest.apxc

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    // createVehicle
    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }

    // createEquipment
    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    // createMaintenanceRequest
    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
                            Subject='Testing subject',
                            Equipment__c=equipmentId,
                            Vehicle__c=vehicleId);
        return cse;
    }

    // createEquipmentMaintenanceItem
    private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id
equipmentId,id requestId){
        Equipment_Maintenance_Item__c equipmentMaintenanceItem = new
Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
            Maintenance_Request__c = requestId);
        return equipmentMaintenanceItem;
```

```
    }

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

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

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

        Equipment_Maintenance_Item__c equipmentMaintenanceItem =
createEquipmentMaintenanceItem(equipmentId,createdCase.id);
        insert equipmentMaintenanceItem;

        test.startTest();
        createdCase.status = 'Closed';
        update createdCase;
        test.stopTest();

        Case newCase = [Select id,
                        subject,
                        type,
                        Equipment__c,
                        Date_Reported__c,
                        Vehicle__c,
                        Date_Due__c
                       from case
                       where status ='New'];

        Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newCase.Id];
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 2);
        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
```

```
        system.assertEquals(newCase.Type, 'Routine Maintenance');
        SYSTEM.assertEquals(newCase.Equipment__c, equipmentId);
        SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
        SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
    }

    @isTest
    private static void testNegative(){
        Vehicle__C vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        product2 equipment = createEquipment();
        insert equipment;
        id equipmentId = equipment.Id;
        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
        insert createdCase;

        Equipment_Maintenance_Item__c workP =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
        insert workP;

        test.startTest();
        createdCase.Status = 'Working';
        update createdCase;
        test.stopTest();

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

        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c = :createdCase.Id];

        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }
    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
```

```
list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();
        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEquipment());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
            caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
        }
        insert caseList;

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

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.
get(i).id, caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;

        test.startTest();
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();
        list<case> newCase = [select id
                              from case
                              where status ='New'];
        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in: oldCaseIds];
        system.assert(newCase.size() == 300);
        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 600);
    }
}
```

MaintenanceRequestHelper.apxc

```
public with sharing class MaintenanceRequestHelper {
   public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
      Set<Id> validIds = new Set<Id>();
      For (Case c : updWorkOrders){
         if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
               validIds.add(c.Id);
            }
         }
      }
      //When an existing maintenance request of type Repair or Routine Maintenance is
closed,
      //create a new maintenance request for a future routine checkup.
      if (!validIds.isEmpty()){
         Map<Id,Case> closedCases = 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>();
         //calculate the maintenance request due dates by using the maintenance cycle
defined on the related equipment records.
         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'));
         }
         List<Case> newCases = new List<Case>();
         for(Case cc : closedCases.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 multiple pieces of equipment are used in the maintenance request,
            //define the due date by applying the shortest maintenance cycle to today's
date.
            //If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
            //} else {
            //    nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
            //}
            newCases.add(nc);
        }
        insert newCases;
        List<Equipment_Maintenance_Item__c> clonedList = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c clonedListItem :
closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}
```

MaintenanceRequest.apxt

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

5.Test callout logic

WarehouseCalloutService.apxc

```
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();
  }

}
```

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);
  }
}
```

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":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"},{"_id":"55d66226
726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726b6
11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

6.Test scheduling logic

WarehouseSyncSchedule

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

WarehouseSyncScheduleTest

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
   // implement scheduled code here
   //
   @isTest static void test() {
      String scheduleTime = '00 00 00 * * ? *';
      Test.startTest();
      Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
      String jobId = System.schedule('Warehouse Time to Schedule to test',
scheduleTime, new WarehouseSyncSchedule());
      CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
      System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

      Test.stopTest();
   }
}
```