Apex Triggers

1. Get Started with Apex Triggers

Create an Apex triggerthat sets an account's ShippingPostal Code to match the Billing Postal Code if the Match Billing Address option is selected. Fire the trigger before inserting an account or updating an account.

Pre-Work:

Add a checkboxfield to the Account object:

a. Field Label:`Match BillingAddress`

b. Field Name: `Match_Billing_Address`

Note: The resulting API Name shouldbe `Match_Billing_Address`

c. Create an Apex trigger:
   i. Name:`AccountAddressTrigger`
   ii. Object: **Account**
   iii. Events: before insert and before update
   iv. Condition: Match Billing Addressis true
   v. Operation: set the ShippingPostal Code to match the Billing PostalCode

Code for `AccountAddressTrigger`

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

    for(Account account:Trigger.New){
```

```
        if(account.Match_Billing_Address__c ==
        True){
            account.ShippingPostalCode
=account.BillingPostalCode;
        }
    }

}
```

2. Bulk Apex Triggers

Create a bulkifiedApex trigger that adds a follow-up task to an opportunity if its stage isClosedWon. Fire the Apex triggerafter inserting or updating an opportunity.

a. Create an Apex trigger:

1. Name:`ClosedOpportunityTrigger`
2. Object: **Opportunity**
3. Events: after insert and after update
4. Condition: Stage is `Closed Won`
5. Operation: Createa task:
    a. `Subject`: `Follow Up Test Task`
    b. `WhatId`: the opportunity ID (associates the task with the opportunity)
6. Bulkify the Apex triggerso that it can insertor update 200 or more opportunities

Code for `ClosedOpportunityTrigger`

```
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
TestTask',WhatId = opp.Id));
        }
    }

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

}
```

Apex Testing

1. Get Started with Apex Unit Tests

Create and install a simple Apex class to test if a date is within a proper range, and if not, returnsa date that occurs at the end of the month withinthe range. You'llcopy the code for the class from GitHub. Then write unit tests that achieve 100% code coverage.

a. Create an Apex class:
   i. Name:`VerifyDate`
   ii. Code: **Copy from GitHub**
b. Place the unit tests in a separate test class:
   i. Name:`TestVerifyDate`
   ii. Goal:100% code coverage
c. Runyour test class at least once

Code for VerifyDate

```
public class VerifyDate {

     //method to handle potential checks against two dates
     public staticDate CheckDates(Date date1,Date date2) {
          //if date2 is within the next 30 days of date1,
usedate2.  Otherwiseuse 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
ofdate1
     @TestVisible private static Boolean
DateWithin30Days(Datedate1, Date date2) {

          //check for date2 being in the
     pastif( date2 < date1){ return false;}

     //check that date2 is within (>=) 30 days of date1
     Date date30Days = date1.addDays(30); //createa 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 privatestatic Date
    SetEndOfMonthDate(Date
date1) {
        Integer totalDays =
Date.daysInMonth(date1.year(),date1.month());
        Date lastDay =
Date.newInstance(date1.year(),date1.month(),
totalDays);
        return lastDay;
    }
```

2. Test Apex Triggers

Create and install a simple Apex trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'. You'llcopy the code for the class fromGitHub. Then write unit tests that achieve 100% code coverage.

a. Create an Apex triggeron the Contact object
   i. Name:`RestrictContactByName`
   ii. Code: **Copy from GitHub**
b. Place the unit tests in a separate test class
   i. Name:`TestRestrictContactByName`
   ii. Goal:100% test coverage
c. Runyour test class at least once

Code for RestrictContactByName

```
trigger RestrictContactByName on Contact (before insert,
```

```
beforeupdate) {

      //check contacts prior to insert or update for invalid
      dataFor (Contactc : Trigger.New) {
            if(c.LastName == 'INVALIDNAME') {  //invalidname is
invalid
                  c.AddError('The Last Name "'+c.LastName+'" is not
allowed for DML');
            }

      }
}
```

Code for TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {

   @isTest static void testContactTrigger() {
      Test.StartTest();
      Contact c = new Contact(LastName = 'INVALIDNAME');
      Database.SaveResult result = Database.insert(c, false);
      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size() > 0);
      Test.StopTest();


  }
}
```

3. Create Test Data for Apex Tests

Create an Apex class that returnsa list of contacts based on two incoming parameters:

the number of contacts to generate and the last name. Do not insert the generatedcontact records into the database.

NOTE: For the purposesof verifying this hands-on challenge, don't specify the @isTestannotation for eitherthe class or the method,even though it's usually required.

a. Create an Apex class in the `public` scope
    i. Name: `RandomContactFactory` (without the @isTest annotation)
b. Use a Public Static Method to consistently generatecontacts with unique firstnamesbased on the iterated number in the format Test 1, Test 2 and so on.
    i. Method Name: `generateRandomContacts` (without the @isTestannotation)
    ii. Parameter 1: An integerthat controls the number of contacts beinggenerated with unique first names
    iii. Parameter 2: A string containing the last name of the contacts
    iv. Return Type: `List < Contact >`

Code for RandomContactFactory

```
public class RandomContactFactory {

   public static List<Contact> generateRandomContacts(Integer numOfContacts, StringlName) {
      List<Contact> cList = new List<Contact>();
      for(Integer i=0; i<numOfContacts; i++) {
         Contact c = new Contact(Firstname = 'Test' + i, Lastname= lName);
         cList.add(c);
```

```
    }
    return cList;
  }

}
```

Asynchronous Apex

Create an Apex class with a future method that accepts a List of Account IDs and updates a custom field on the Account objectwith the numberof contacts associatedto the Account. Write unit tests that achieve 100% code coverage for the class. Every hands-on challengein this module asks you to create a test class.

c. Create a field on the Accountobject:
   i. Label: `Number Of Contacts`
   ii. Name: `Number_Of_Contacts`
   iii. Type: **Number**
   iv. Thisfield will hold the total number of Contacts for the Account
d. Create an Apex class:
   i. Name: `AccountProcessor`
   ii. Method name: `countContacts`
   iii. The methodmust accept a List of Account IDs
   iv. Themethod must use the @futureannotation
   v. The method counts the number of Contact recordsassociated to eachAccount ID passed to the method and updates the 'Number_Of_Contacts_c'field with this value

e. Create an Apex test class:
   i. Name:`AccountProcessorTest`
   ii. The unit tests must cover all lines of code includedin the **AccountProcessor** class, resultingin 100% code coverage.
f. Before verifying this challenge, run your test class at least once using theDeveloper ConsoleRun All feature

Code for `AccountProcessor`

```
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)
fromAccount 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;
  }

}
```

code forAccountProcessorTest

```
@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='Jane',LastName='Doe',AccountId
= newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();


        accountIds.add(newAccount.Id);
        Test.startTest();
        AccountProcessor.countContacts(accountIds);


        Test.stopTest();
    }
```

}

Use Batch Apex

Create an Apex class that implements the Database.Batchable interface to update allLead recordsin the org with a specific LeadSource.

g. Create an Apex class:
   i. Name:`LeadProcessor`
   ii. Interface: `Database.Batchable`
   iii. Use a QueryLocator in the start method to collect all Lead recordsin theorg
   iv. The executemethod must update all Lead records in the org with theLeadSource value of `Dreamforce`
h. Create an Apex test class:
   i. Name: `LeadProcessorTest`
   ii. In the test class,insert 200 Lead records, executethe LeadProcessorBatchclass and test that all Lead recordswere updated correctly
   iii. Theunit tests must cover all lines of code includedin the **LeadProcessor**

   class, resulting in 100% code coverage
i. Before verifying this challenge, run your test class at least once using theDeveloper ConsoleRun All feature

Code for `LeadProcessor`

```
global class LeadProcessor implements Database.Batchable<sObject> {


  global Integer count = 0;
   global Database.QueryLocator start (Database.BatchableContext bc) {
```

```
        return Database.getQueryLocator('Select Id, LeadSource from lead');



    }
    global void execute (Database.BatchableContext bc,List<Lead> l_lst) {
        List<lead> l_lst_new = new List<lead>();
        for(lead l : l_lst) {
            l.leadsource = 'Dreamforce';
            l_lst_new.add(l);
            count+=1;
        }
        update l_lst_new;



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

Code for `LeadProcessorTest`

```
@isTest



public class LeadProcessorTest {
    @isTest
    public static void testit(){
```

```
        List<lead> L_list =  new List<lead>();
         for(Integer i=0;i<200;i++){
            Lead L= new lead();
            L.LastName = 'name' + i;
            L.Company = 'Company';
            L.Status = 'Random Status';
            L_list.add(L);
        }



        insert L_list;

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

Control Processes with QueueableApex

Create a Queueable Apex class that inserts the same Contactfor each Accountfor a specific state.

j. Create an Apex class:
   i. Name: `AddPrimaryContact`
   ii. Interface: `Queueable`
   iii. Create a constructor for the class that acceptsas its first argument aContact sObject and a second argument as a string for the State abbreviation

iv. The `execute` method must queryfor a maximum of 200 Accounts withthe `BillingState` specified by the State abbreviation passed into the constructor and insert the Contact sObject record associated to each Account. Look at the sObject `clone()` method.

k. Create an Apex test class:
   i. Name: `AddPrimaryContactTest`
   ii. In the test class, insert 50 Account recordsfor `BillingState NY` and 50Account recordsfor `BillingState CA`
   iii. Create an instance of the `AddPrimaryContact` class, enqueue the job,and assertthat a Contact record was inserted for each of the 50 Accountswith the `BillingState` of `CA`
   iv. The unit tests must cover all lines of code includedin the **AddPrimaryContact** class, resultingin 100% code coverage

l. Before verifying this challenge, run your test class at least once using theDeveloper ConsoleRun All feature

Code for `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
fromcontacts)
                          from Account Where BillingState = :state Limit 200];


     List<Contact> primaryContacts = new List<Contact>();
      for(Account acc:accounts){
         Contact c = con.clone();
         c.AccountId = acc.Id;
         primaryContacts.add(c);


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


       insert primaryContacts;
      }
  }

}
```

Code for `AddPrimaryContactTest`

```
 @isTest


public class AddPrimaryContactTest {
  static testmethod void testQueueable(){
```

```
        List<Account> testAccounts = new List<Account>();

            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 = 'Jhon', 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
(SelectID from Account Where BillingState='CA')]);
    }

}
```

5.Schedule Jobs Usingthe Apex Scheduler

Create an Apex class that implements the Schedulable interfaceto update Lead recordswith a specificLeadSource. (This is very similarto what you did for Batch Apex.)

m. Create an Apex class:
   i. Name:`DailyLeadProcessor`
   ii. Interface: `Schedulable`
   iii. The executemethod must find the first 200 Lead records with a blankLeadSource field and update them with the LeadSource value of `Dreamforce`
n. Create an Apex test class:
   i. Name: `DailyLeadProcessorTest`
   ii. In the test class, insert 200 Lead records, schedule the DailyLeadProcessor class to run and test that all Lead recordswereupdated correctly
   iii. Theunit tests must cover all lines of code includedin the **DailyLeadProcessor** class, resultingin 100% code coverage.
o. Before verifying this challenge, run your test class at least once using theDeveloper ConsoleRun All feature

Code for `DailyLeadProcessor`

```
global class DailyLeadProcessor implements Schedulable{
   global void execute(SchedulableContext sc){
      List<Lead> lstofLead = [Select ID From Lead Where LeadSource = NULL Limit200];

      List<Lead> lstOfUpdatedLead = new List<Lead>();
      if(!lstOfLead.isEmpty()){
      for(Lead ld : lstOfLead){
         ld.LeadSource = 'Dreamforce';
         lstOfUpdatedLead.add(ld);
```

```
        }
            update lstOfUpdatedLead;



        }
    }
}
```

Code for `DailyLeadProcessorTest`

```
@isTest
private class DailyLeadProcessorTest {
    @testSetup
    static void setup(){
        List<Lead> lstOfLead = new List<Lead>();
        for(Integer i = 1; i<=200; i++){
            Lead ld= new Lead(Company = 'Comp' + i,LastName = 'LN' + i, Status =

'Working - Contacted');
            lstOfLead.add(ld);



        }
        Insert lstOfLead;



    }
    static testmethod void testDailyLeadProcessorScheduledJob(){
        String sch = '0 5 12 * * ?';
```

```
        Test.startTest();
        String jobId = System.Schedule('ScheduledApexText', sch,
newDailyLeadProcessor());

        List<Lead> lstOfLead = [Select Id From Lead Where LeadSource = NULL
Limit200];



        system.assertEquals(200,lstOfLead.size());
        Test.stopTest();



    }
    }
```

Deploy Lightning Web Component Files

Code

html

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

```
</template>
```

javascript

```
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';
 }
```

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>
```

Lightning Web Components Basics

Create a Lightning app page that uses the wire serviceto display the current user's name.

Prework: You need files createdin the previous unit to complete this challenge. If youhaven’talready completed the activities in the previousunit, do that now.

p. Create a Lightning app page:
    i. Label: `Your Bike Selection`
    ii. Developer Name: `Your_Bike_Selection`
q. Add the current user's name to the app container:

    i. Edit selector.js
    ii. Edit selector.html

Data

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

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

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

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">Available Bikes for {name}</header>
    <section class="content">
        <div class="columns">
        <main class="main" >
            <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, wire } from 'lwc';
import { getRecord, getFieldValue } from 'lightning/uiRecordApi';
import Id from '@salesforce/user/Id';
import NAME_FIELD from '@salesforce/schema/User.Name';
```

```
const fields = [NAME_FIELD];

export default class Selector extends LightningElement {

    selectedProductId;

    handleProductSelected(evt) {

        this.selectedProductId = evt.detail;

    }

    userId = Id;

    @wire(getRecord, { recordId: '$userId', fields })

    user;

    get name() {

        return getFieldValue(this.user.data, NAME_FIELD);

    }

}
```

Selector.js-meta.html

```
<?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>
```

Title

title.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;
}
```

title.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>
```

title.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);
    }
}
```

title.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 Integration Services

ApexREST Callouts

Create an Apex class that calls a REST endpointto return the name of an animal,write unit tests that achieve 100% code coveragefor the class using a mock response, andrun your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

r. Create an Apex class:
   i. Name:`AnimalLocator`
   ii. Method name: `getAnimalNameById`
   iii. The methodmust accept an Integer and return a String.
   iv. The method must call https://th-apex-http-callout.herokuapp.com/animals/<id>, replacing<id> with the ID passedinto the method
   v. Themethod returns the value of the **name** property (i.e., the animalname)

s. Create a test class:
   i. Name:`AnimalLocatorTest`
   ii. The test class uses a mock class called`AnimalLocatorMock` to mockthe callout response

t. Create unit tests:
   i. Unit tests must cover all lines of code included in the

AnimalLocator class,resulting in 100% code coverage

u. Run your test class at least once (via **Run All** tests the Developer Console)beforeattempting to verify this challenge

Code for `AnimalLocator`

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

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

Code for `AnimalLocatorTest`

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

Apex SOAP Callouts

Generate an Apex class using WSDL2Apex for a SOAP web service,write unit tests that achieve 100% code coverage for the class using a mock response,and run your Apextests.

Prework: Be sure the Remote Sites from the first unit are set up.

v. Generate a class using this using this WSDL file:
    i. Name: `ParkService` (Tip: After you click the **Parse WSDL** button, changethe Apex classname from **parksServices** to`ParkService`)
    ii. Class must be in public scope
w. Create a class:
    i. Name:`ParkLocator`
    ii. Class must have a **country**method that uses the **ParkService** class
    iii. Method must return an array of available park names for a particularcountry passed to the web service (such as Germany,India, Japan, andUnited States)
x. Create a test class:
    i. Name:`ParkLocatorTest`
    ii. Test class uses a mock class called `ParkServiceMock` to mock thecallout response
y. Create unit tests:
    i. Unit tests must cover all lines of code included in the **ParkLocator** class,resulting in 100% code coverage.
z. Run your test class at least once (via **Run All** tests the Developer Console)beforeattempting to verify this challenge.

Code for ParkServiceMock

```
@isTest
global classParkServiceMock 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) {
        ParkService.byCountryResponse response_x =
newParkService.byCountryResponse();
        List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};

        response_x.return_x = lstOfDummyParks;
        response.put('response_x', response_x);
    }
}
```

Code for `ParkLocator`

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

Code for `ParkLocatorTest`

```
    @isTest
    private class ParkLocatorTest{
        @isTest
        static void testParkLocator() {
     Test.setMock(WebServiceMock.class, new
    ParkServiceMock());

        String[] arrayOfParks = ParkLocator.country('India');
        System.assertEquals('Park1', arrayOfParks[0]);
```

```
    }
}
```

ApexWeb Services

Create an Apex REST classthat is accessible at /Accounts/<Account_ID>/contacts. Theservice will return the account's ID and name plus the ID and name of all contacts associated with the account. Write unit tests that achieve 100% code coverage for theclass and run your Apex tests.

Prework: Be sure the Remote Sites from the first unit are set up.

aa. Create an Apex class
   i. Name:`AccountManager`
   ii. Class must have a method called`getAccount`
   iii. Method must be annotated with **@HttpGet**and return an **Account** object
   iv. Method must return the **ID** and **Name** forthe requested recordand allassociated contacts with their **ID** and **Name**

ab. Create unit tests
   i. Unittests must be in a separate Apex class called `AccountManagerTest`
   ii. Unit tests must cover all lines of code included in the **AccountManager** class, resulting in 100% code coverage

ac. Run your test class at least once (via **Run All** tests the Developer Console)beforeattempting to verify this challenge

Code for `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)
fromAccount where Id=:accountId Limit 1];
       return result;
   }
}
```

Code for `AccountManagerTest`

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

```
    }
    static Id createTestRecord(){
        Account accountTest = new Account(

          Name ='Testrecord');

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

        insert contactTest;
        return accountTest.Id;
    }
}
```

ApexSpecialist Super Badge

Automate record creation

code for MaintenanceRequest

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

```
    }
}
```

code for MaintenanceRequestHelper

```
public with sharingclass 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
isclosed,
        //create a new maintenance requestfor 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 requestdue dates by using the maintenance
cycledefined on the related equipmentrecords.
            AggregateResult[] results = [SELECT Maintenance_Request_c,
```

```
                        MIN(Equipment_r.Maintenance_Cycle_c)cycle
                        FROMEquipment_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.Vehiclec, Equipment_c
                =cc.Equipment_c,

                Origin = 'Web',
                Date_Reported_c = Date.Today()
            );

//If multiple piecesof equipment are used in the maintenance request,
//define the due date by applying the shortest maintenance cycle to today's

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;
    }
  }
}
```

Synchronize Salesforce data with an external system

code for WarehouseCalloutService

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

  //Write a class that makes a REST calloutto an external warehouse systemto get a
list of equipment that needs to be updated.
```

```
    //The callout’sJSON response returnsthe equipment recordsthat you upsert in Salesforce.

    @future(callout=true)
    public static void runWarehouseEquipmentSync(){
        System.debug('go into
        runWarehouseEquipmentSync');Http http = new Http();
        HttpRequest request= new HttpRequest();


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


        List<Product2> product2List = new
        List<Product2>();
        System.debug(response.getStatusCode());
        if (response.getStatusCode() ==
            200){List<Object> jsonResponse
            =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());
            //class maps the followingfields:
            //warehouse SKU will be external ID for identifying which equipment recordstoupdate withinSalesforce
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson= (Map<String,Object>)jR;
                  Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part_c= (Boolean) mapJson.get('replacement');
                //cost
                  product2.Cost_c = (Integer)mapJson.get('cost');
                //current inventory

                product2.Current_Inventory_c = (Double) mapJson.get('quantity');
```

```
            //lifespan
            product2.Lifespan_Months_c= (Integer) mapJson.get('lifespan');
            //maintenance cycle
            product2.Maintenance_Cycle_c = (Integer)
mapJson.get('maintenanceperiod');
            //warehouse SKU
            product2.Warehouse_SKU_c= (String) mapJson.get('sku');

            product2.Name = (String) mapJson.get('name');
            product2.ProductCode = (String) mapJson.get('_id');
            product2List.add(product2);
          }
           if (product2List.size() >
            0){upsert product2List;
            System.debug('Your equipment was synced with the warehouseone');
          }
        }
    }

    public static void execute(QueueableContext
      context){                    System.debug('start
      runWarehouseEquipmentSync');
      runWarehouseEquipmentSync();
      System.debug('end  runWarehouseEquipmentSync');
    }

}
```

schedule synchronization

code for WarehouseSyncSchedule

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

}
```

Test automation logic

code for MaintenanceRequestHelperTest

```
@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 Product2createEquipment(){
        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='We
```

```
                    b',
                    Subject='Testingsubject
                    ',
                    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();
        insertvehicle;
        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 =
                    [Selectid,
                    subject,
                    type,
                    Equipment_
                    c,
                    Date_Reported_
                    c,Vehicle_c,

                    Date_Due_c
                   from case
                   where status ='New'];


        Equipment_Maintenance_Item_cworkPart = [selectid
                                    from Equipment_Maintenance_Item_c
                                    where Maintenance_Request_c =:newCase.Id];
        list<case> allCase = [selectid from case];
        system.assert(allCase.size() == 2);


        system.assert(newCase != null);
        system.assert(newCase.Subject != null);
        system.assertEquals(newCase.Type, 'RoutineMaintenance');
        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();
    insertvehicle;
    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(equipmentLis
t. 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
                              wherestatus ='New'];



        list<Equipment_Maintenance_Item_c> workParts= [select id
                                          from Equipment_Maintenance_Item_c
                                          where Maintenance_Request_cin: oldCaseIds];

        system.assert(newCase.size() == 300);

        list<case> allCase = [selectid from
        case];system.assert(allCase.size() ==
        600);
    }
}
```

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;
    }
  }
}
```

MaintenanceRequest

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

Test calloutlogic

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);
}
```

```
        }

    }

}
```

code for 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":"55d662
26 726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d66226726
b6 11100aaf743","replacement":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);

        return response;
    }
}
```

code forWarehouseCalloutServiceTest

```
@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);
  }
}
```

test scheduling logic

code for 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 Scheduleto 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();
    }
}
```

WarehouseSyncSchedule

-

```
global class WarehouseSyncSchedule implements Schedulable {

    global void execute(SchedulableContext ctx) {



        WarehouseCalloutService.runWarehouseEquipmentSync();
```

```
    }

}
```