# Apex Triggers

## Get Started with Apex Triggers

**Create an Apex trigger**

Create an Apex trigger that sets an account's Shipping Postal 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 checkbox field to the Account object:

- Field Label: `Match Billing Address`
- Field Name: `Match_Billing_Address`
  Note: The resulting API Name should be `Match_Billing_Address__c`.
- Create an Apex trigger:
  - Name: `AccountAddressTrigger`
  - Object: **Account**
  - Events: before insert and before update
  - Condition: Match Billing Address is true
  - Operation: set the Shipping Postal Code to match the Billing Postal Code

AccountAddressTrigger.apxt

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

}

# Bulk Apex Triggers

**Create a Bulk Apex trigger**

Create a bulkified Apex trigger that adds a follow-up task to an opportunity if its stage is Closed Won. Fire the Apex trigger after inserting or updating an opportunity.

- Create an Apex trigger:
    - Name: `ClosedOpportunityTrigger`
    - Object: **Opportunity**
    - Events: after insert and after update
    - Condition: Stage is `Closed Won`
    - Operation: Create a task:
        - `Subject`: `Follow Up Test Task`
        - `WhatId`: the opportunity ID (associates the task with the opportunity)
    - Bulkify the Apex trigger so that it can insert or update 200 or more opportunities

ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {
List<Task> tList = new List<Task>();
for(Opportunity o:Trigger.new) {
if(Trigger.isInsert) {
if(o.StageName == 'Closed Won') {
tList.add(new Task(Subject = 'Follow Up Test Task', WhatId = o.Id));
}
}
if(Trigger.isUpdate) {
if(o.StageName == 'Closed Won' && o.StageName !=
Trigger.oldMap.get(o.Id).StageName) {
tList.add(new Task(Subject='Follow Up Test Task',WhatId =o.Id));
}
}
```

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

# Apex Testing

## Get Started with Apex Unit Tests

**Create a Unit Test for a Simple Apex Class**

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

- Create an Apex class:
  - Name: `VerifyDate`
  - Code: **Copy from GitHub**
- Place the unit tests in a separate test class:
  - Name: `TestVerifyDate`
  - Goal: 100% code coverage
- Run your test class at least once

VerifyDate.apxc

```
public class VerifyDate {
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
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
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;
}
}
```

TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {
@isTest static void CheckDatesTesttrue() {
Date date1=date.today();
Date date2=date1.addDays(29);
Date t = VerifyDate.CheckDates(date1, date2);
System.assertEquals(t, date2);
}
@isTest static void DateOver() {
Date date1=date.today();
Date date2=date1.addDays(31);
Date t = VerifyDate.CheckDates(date1, date2);
System.assertNotEquals(t, date1);
}
}
```

# Test Apex Triggers

**Create a Unit Test for a Simple Apex Trigger**

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

- Create an Apex trigger on the Contact object
    - Name: `RestrictContactByName`
    - Code: **Copy from GitHub**
- Place the unit tests in a separate test class
    - Name: `TestRestrictContactByName`
    - Goal: 100% test coverage
- Run your test class at least once

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 static void createBadContact(){
 Contact c=new Contact(FirstName='John',LastName='INVALIDNAME');
 Test.startTest();
 Database.SaveResult result=Database.insert(c,false);
 Test.stopTest();
 System.assert(!result.isSuccess());
 }

}
```

# Create Test Data for Apex Tests

**Create a Contact Test Factory**

Create an Apex class that returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name. Do not insert the generated contact records into the database.

NOTE: For the purposes of verifying this hands-on challenge, don't specify the @isTest annotation for either the class or the method, even though it's usually required.

- Create an Apex class in the `public` scope
    - Name: `RandomContactFactory` (without the @isTest annotation)
- Use a Public Static Method to consistently generate contacts with unique first names based on the iterated number in the format Test 1, Test 2 and so on.
    - Method Name: `generateRandomContacts` (without the @isTest annotation)
    - Parameter 1: An integer that controls the number of contacts being generated with unique first names
    - Parameter 2: A string containing the last name of the contacts
    - Return Type: `List < Contact >`

RandomContactFactory.apxc

```
public class RandomContactFactory {
public static List<Contact> generateRandomContacts(Integer
NumberofContacts, String name){
List<Contact> con = new List<Contact>();
for(Integer i=0; i<NumberofContacts; i++){
name = 'Test'+i;
Contact c = new Contact(FirstName=name, LastName=name);
con.add(c);
}
return con;
}
}
```

# Asynchronous Apex

## Use Batch Apex

**Create an Apex class that uses Batch Apex to update Lead records.**

Create an Apex class that implements the Database.Batchable interface to update all Lead records in the org with a specific LeadSource.

- Create an Apex class:
    - Name: `LeadProcessor`
    - Interface: `Database.Batchable`
    - Use a QueryLocator in the start method to collect all Lead records in the org
    - The execute method must update all Lead records in the org with the LeadSource value of `Dreamforce`
- Create an Apex test class:
    - Name: `LeadProcessorTest`
    - In the test class, insert 200 Lead records, execute the LeadProcessor Batch class and test that all Lead records were updated correctly
    - The unit tests must cover all lines of code included in the **LeadProcessor** class, resulting in 100% code coverage
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

LeadProcessor.apxc

```
public class LeadProcessor implements Database.Batchable<SObject>,
 Database.Stateful{
 public Integer recordProcessed = 0;
 public Database.QueryLocator start(Database.BatchableContext bc) {
 return Database.getQueryLocator([Select LeadSource From Lead ]);
 }
 public void execute(Database.BatchableContext bc, List<Lead> scope) {
```

```
for (Lead lead : scope) {
lead.LeadSource = 'Dreamforce';
}
update scope;
}
public void finish(Database.BatchableContext bc){
}
}
```

LeadProcessorTest.apxc

```
@isTest
public class LeadProcessorTest {
@isTest
static void test(){
List<Lead> leads = new List<Lead>();
for (Integer i = 0; i < 200; i++) {
Lead lead = new Lead();
lead.FirstName = 'testF';
lead.LastName = 'testL' + i;
lead.Company = 'testCompany' + i;
leads.add(lead);
}
insert leads;
Test.startTest();
LeadProcessor leadProcessor = new LeadProcessor();
Database.executeBatch(leadProcessor);
Test.stopTest();
System.assertEquals(200, [select count() from Lead where LeadSource =
'Dreamforce' ]);
}
}
```

# Control Processes with Queueable Apex

**Create a Queueable Apex class that inserts Contacts for Accounts.**

Create a Queueable Apex class that inserts the same Contact for each Account for a specific state.

- Create an Apex class:
    - Name: `AddPrimaryContact`

    - Interface: `Queueable`
    - Create a constructor for the class that accepts as its first argument a Contact sObject and a second argument as a string for the State abbreviation
    - The `execute` method must query for a maximum of 200 Accounts with the `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.

- Create an Apex test class:
    - Name: `AddPrimaryContactTest`
    - In the test class, insert 50 Account records for `BillingState NY` and 50 Account records for `BillingState CA`
    - Create an instance of the `AddPrimaryContact` class, enqueue the job, and assert that a Contact record was inserted for each of the 50 Accounts with the `BillingState` of `CA`
    - The unit tests must cover all lines of code included in the **AddPrimaryContact** class, resulting in 100% code coverage
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable {
private Contact contact;
private String state;
public AddPrimaryContact(Contact contact, String state){
this.contact = contact;
this.state = state;
}
public void execute(QueueableContext context){
List<Account> accounts = [select ID, Name, (select id, FirstName, LastName
from contacts) from Account where BillingState = :state limit 200 ];
List<Contact> contacts = new List<Contact>();
for (Account account : accounts) {
Contact temp = contact.clone(false, false, false, false);
temp.AccountId = account.Id;
contacts.add(temp);
}
```

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

AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest {
@testSetup
static void setup(){
List<Account> accounts = new List<Account>();
for (Integer i = 0; i < 50; i++) {
accounts.add(new Account(Name='NY'+i, billingstate='NY'));
}
for (Integer i = 0; i < 50; i++) {
accounts.add(new Account(Name='CA'+i, billingstate='CA'));
}
insert accounts;
}
static testmethod void testQueueable(){
Contact contact = new Contact(FirstName = 'AddPrimaryContactTest',
LastName = 'Queueable');
Test.startTest();
AddPrimaryContact addPrimaryContact = new AddPrimaryContact(contact,
'CA');
System.enqueueJob(addPrimaryContact);
Test.stopTest();
System.assertEquals(50, [select count() from Contact where AccountId in
(select Id from Account where BillingState = 'CA')]);
}
}
```

# Schedule Jobs Using the Apex Scheduler

**Create an Apex class that uses Scheduled Apex to update Lead records.**

Create an Apex class that implements the Schedulable interface to update Lead records with a specific LeadSource. (This is very similar to what you did for Batch Apex.)

- Create an Apex class:

    - Name: `DailyLeadProcessor`
    - Interface: `Schedulable`
    - The execute method must find the first 200 Lead records with a blank LeadSource field and update them with the LeadSource value of `Dreamforce`
- Create an Apex test class:
    - Name: `DailyLeadProcessorTest`
    - In the test class, insert 200 Lead records, schedule the DailyLeadProcessor class to run and test that all Lead records were updated correctly
    - The unit tests must cover all lines of code included in the **DailyLeadProcessor** class, resulting in 100% code coverage.
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements Schedulable {
 public void execute(SchedulableContext context){
 List<Lead> leads = [select Id, LeadSource from Lead where LeadSource = null];
 for (Lead lead : leads) {
 lead.LeadSource = 'Dreamforce';
 }
 update leads;
 }
 }
```

DailyLeadProcessorTest.apxc

```
@isTest
public class DailyLeadProcessorTest {
static @IsTest
void testScheduledJob(){
List<Lead> leads = new List<Lead>();
for (Integer i = 0; i < 200; i++) {
leads.add(new Lead(LastName = 'Dreamforce'+i, Company = 'test'+i));
}
insert leads;
Map<Id,Lead> leadMap = new Map<Id,Lead>(leads);
List<Id> leadsId = new List<Id>(leadMap.keySet());
```

```
Test.startTest();
System.schedule('DailyLeadProcessor', '20 30 8 10 2 ?', new
DailyLeadProcessor());
Test.stopTest();
System.assertEquals(200, [select count() from Lead where LeadSource =
'Dreamforce' and Id in :leadsId]);
}
}
```

# Apex Integration Services

## Apex REST Callouts

**Create an Apex class that calls a REST endpoint and write a test class.**
Create an Apex class that calls a REST endpoint to return the name of an animal, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

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

- Create an Apex class:
    - Name: `AnimalLocator`
    - Method name: `getAnimalNameById`
    - The method must accept an Integer and return a String.
    - The method must call https://th-apex-http-callout.herokuapp.com/animals/<id>, replacing <id> with the ID passed into the method
    - The method returns the value of the **name** property (i.e., the animal name)
- Create a test class:
    - Name: `AnimalLocatorTest`
    - The test class uses a mock class called `AnimalLocatorMock` to mock the callout response
- Create unit tests:
    - Unit tests must cover all lines of code included in the **AnimalLocator** class,

resulting in 100% code coverage

- Run your test class at least once (via **Run All** tests the Developer Console) before attempting to verify this challenge

AnimalLocator.apxc

```
public class AnimalLocator
{
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.setMethod('GET');
HttpResponse response = http.send(request);
String strResp = '';
system.debug('******response '+response.getStatusCode());
system.debug('******response '+response.getBody());
// If the request is successful, parse the JSON response.
if (response.getStatusCode() == 200)
{
// Deserializes the JSON string into collections of primitive data types.
Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
// Cast the values in the 'animals' key as a list
Map<string,object> animals = (map<string,object>) results.get('animal');
System.debug('Received the following animals:' + animals );
strResp = string.valueof(animals.get('name'));
System.debug('strResp >>>>>>' + strResp );
}
return strResp ;
}
}
```

AnimalLocatorTest.apxc

```
@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 and write a test class.**

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 Apex tests.

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

- Generate a class using this using this WSDL file:
    - Name: `ParkService` (Tip: After you click the **Parse WSDL** button, change the Apex class name from **parksServices** to `ParkService`)
    - Class must be in public scope
- Create a class:
    - Name: `ParkLocator`
    - Class must have a **country** method that uses the **ParkService** class
    - Method must return an array of available park names for a particular country passed to the web service (such as Germany, India, Japan, and United States)
- Create a test class:
    - Name: `ParkLocatorTest`
    - Test class uses a mock class called `ParkServiceMock` to mock the callout response
- Create unit tests:
    - Unit tests must cover all lines of code included in the **ParkLocator** class, resulting in 100% code coverage.
- Run your test class at least once (via **Run All** tests the Developer Console) before attempting to verify this challenge.

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 testParkLocator() {
 Test.setMock(WebServiceMock.class, new ParkServiceMock());
 String[] arrayOfParks = ParkLocator.country('India');
 System.assertEquals('Park1', arrayOfParks[0]);
 }
 }
```

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) {
ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
response_x.return_x = lstOfDummyParks;
response.put('response_x', response_x);
}
}
```

# Apex Web Services

**Create an Apex REST service that returns an account and its contacts.**
Create an Apex REST class that is accessible at /Accounts/<Account_ID>/contacts. The service 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 the class and run your Apex tests.

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

- Create an Apex class
    - Name: `AccountManager`
    - Class must have a method called `getAccount`
    - Method must be annotated with **@HttpGet** and return an **Account** object
    - Method must return the **ID** and **Name** for the requested record and all associated contacts with their **ID** and **Name**
- Create unit tests
    - Unit tests must be in a separate Apex class called `AccountManagerTest`
    - Unit tests must cover all lines of code included in the **AccountManager** class, resulting in 100% code coverage

- Run your test class at least once (via **Run All** tests the Developer Console) before attempting to verify this challenge

AccountManager.apxc

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

AccountManagerTest.apxc

```
@IsTest
private class AccountManagerTest{
@isTest static void testAccountManager(){
Id recordId = getTestAccountId();
// Set up a test request
RestRequest request = new RestRequest();
request.requestUri =
'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId
+'/contacts';
request.httpMethod = 'GET';
RestContext.request = request;
// Call the method to test
Account acc = AccountManager.getAccount();
// Verify results
System.assert(acc != null);
}
private static Id getTestAccountId(){
Account acc = new Account(Name = 'TestAcc2');
Insert acc;
Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);
Insert con;
return acc.Id;
}
}
```

# Apex Specialist Superbadge

## Automate record creation

Install the unlocked package and configure the development org. Use the included package content to automatically create a Routine Maintenance request every time a maintenance request of type Repair or Routine Maintenance is updated to Closed. Follow the specifications and naming conventions outlined in the business requirements.

MaintenanceRequest.apxt

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

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);
}
}
}
if (!validIds.isEmpty()){
List<Case> newCases = new List<Case>();
Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
ProductId, Product.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,
ProductId =cc.ProductId,
Origin = 'Web',
Date_Reported__c = Date.Today()
);
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.Product.maintenance_Cycle__c);
}
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;
}
}
```

}

## Synchronize Salesforce data with an external system

Implement an Apex class (called WarehouseCalloutService) that implements the queueable interface and makes a callout to the external service used for warehouse inventory management. This service receives updated values in the external system and updates the related records in Salesforce. Before checking this section, enqueue the job at least once to confirm that it's working as expected.

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements Queueable {
private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
@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());
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() >=1){
upsert product2List;
System.debug('Your equipment was synced with the warehouse one');
}
}
}
public static void execute (QueueableContext context){
System.debug('start runWarehouseEquipmentSync');
runWarehouseEquipmentSync();
System.debug('end runWarehouseEquipmentSync');
}
}
```

## Schedule synchronization

Build scheduling logic that executes your callout and runs your code daily. The name of the schedulable class should be WarehouseSyncSchedule, and the scheduled job should be named WarehouseSyncScheduleJob.

WarehouseSyncSchedule.apxc

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

## Test automation logic

Build tests for all cases (positive, negative, and bulk) specified in the business requirements by using a class named MaintenanceRequestHelperTest. You must have 100% test

coverage to pass this section and assert values to prove that your logic is working as expected. Choose Run All Tests in the Developer Console at least once before attempting to submit this section. Be patient as it may take 10-20 seconds to process the challenge check.

MaintenanceRequestHelperTest.apxc

```
@istest
public with sharing class MaintenanceRequestHelperTest {
private static final string STATUS_NEW = 'New';
private static final string WORKING = 'Working';
private static final string CLOSED = 'Closed';
private static final string REPAIR = 'Repair';
private static final string REQUEST_ORIGIN = 'Web';
private static final string REQUEST_TYPE = 'Routine Maintenance';
private static final string REQUEST_SUBJECT = 'Testing subject';
PRIVATE STATIC Vehicle__c createVehicle(){
Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
return Vehicle;
}
PRIVATE STATIC Product2 createEq(){
product2 equipment = new product2(name = 'SuperEquipment',
lifespan_months__C = 10,
maintenance_cycle__C = 10,
replacement_part__c = true);
return equipment;
}
PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
case cs = new case(Type=REPAIR,
Status=STATUS_NEW,
Origin=REQUEST_ORIGIN,
Subject=REQUEST_SUBJECT,
Equipment__c=equipmentId,
Vehicle__c=vehicleId);
return cs;
}
PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id
equipmentId,id requestId){
Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(equipment__c = equipmentId,
Maintenance_Request__c = requestId);
```

```
return wp;
}
@istest
private static void testMaintenanceRequestPositive(){
Vehicle__c vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
Product2 equipment = createEq();
insert equipment;
id equipmentId = equipment.Id;
case somethingToUpdate = createMaintenanceRequest(vehicleId,equipmentId);
insert somethingToUpdate;
Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
insert workP;
test.startTest();
somethingToUpdate.status = CLOSED;
update somethingToUpdate;
test.stopTest();
Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c,
Vehicle__c, Date_Due__c
from case
where status =:STATUS_NEW];
Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c =:newReq.Id];
system.assert(workPart != null);
system.assert(newReq.Subject != null);
system.assertEquals(newReq.Type, REQUEST_TYPE);
SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}
@istest
private static void testMaintenanceRequestNegative(){
Vehicle__C vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
product2 equipment = createEq();
insert equipment;
id equipmentId = equipment.Id;
```

```
case emptyReq = createMaintenanceRequest(vehicleId,equipmentId);
insert emptyReq;
Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId,
emptyReq.Id);
insert workP;
test.startTest();
emptyReq.Status = WORKING;
update emptyReq;
test.stopTest();
list<case> allRequest = [select id
from case];
Equipment_Maintenance_Item__c workPart = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c = :emptyReq.Id];
system.assert(workPart != null);
system.assert(allRequest.size() == 1);
}
@istest
private static void testMaintenanceRequestBulk(){
list<Vehicle__C> vehicleList = new list<Vehicle__C>();
list<Product2> equipmentList = new list<Product2>();
list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
list<case> requestList = new list<case>();
list<id> oldRequestIds = new list<id>();
for(integer i = 0; i < 300; i++){
vehicleList.add(createVehicle());
equipmentList.add(createEq());
}
insert vehicleList;
insert equipmentList;
for(integer i = 0; i < 300; i++){
requestList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
}
insert requestList;
for(integer i = 0; i < 300; i++){
workPartList.add(createWorkPart(equipmentList.get(i).id,
requestList.get(i).id));
}
insert workPartList;
test.startTest();
for(case req : requestList){
```

```
req.Status = CLOSED;
oldRequestIds.add(req.Id);
}
update requestList;
test.stopTest();
list<case> allRequests = [select id
from case
where status =: STATUS_NEW];
list<Equipment_Maintenance_Item__c> workParts = [select id
from Equipment_Maintenance_Item__c
where Maintenance_Request__c in: oldRequestIds];
system.assert(allRequests.size() == 300);
}
}
```

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);
}
}
}
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.apxt

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

## Test callout logic

Build tests for your callout using the included class for the callout mock

(WarehouseCalloutServiceMock) and callout test class (WarehouseCalloutServiceTest) in the package. You must have 100% test coverage to pass this challenge and assert values to prove that your logic is working as expected.

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService implements Queueable {
private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
@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());
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() >=1){
upsert product2List;
System.debug('Your equipment was synced with the warehouse one');
}
}
}
public static void execute (QueueableContext context){
System.debug('start runWarehouseEquipmentSync');
runWarehouseEquipmentSync();
System.debug('end runWarehouseEquipmentSync');
}
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
 global class WarehouseCalloutServiceMock implements HttpCalloutMock {
 // implement http mock callout
 global static HttpResponse respond(HttpRequest request){
 System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
 request.getEndpoint());
 System.assertEquals('GET', request.getMethod());
 // Create a fake response
 HttpResponse response = new HttpResponse();
 response.setHeader('Content-Type', 'application/json');
 response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quan
 tity":5,"name":"Generator 1000
 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
 response.setStatusCode(200);
 return response;
 }
 }
```

WarehouseCalloutServiceTest.apxc

```
@isTest
 private class WarehouseCalloutServiceTest {
 @isTest
 static void testWareHouseCallout(){
 Test.startTest();
 // implement mock callout test here
 Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
```

```
WarehouseCalloutService.runWarehouseEquipmentSync();
WarehouseCalloutService que= new WarehouseCalloutService();
System.enqueueJob(que);
Test.stopTest();
System.assertEquals(1, [SELECT count() FROM Product2]);
}
}
```

## Test scheduling logic

Build unit tests for the class WarehouseSyncSchedule in a class named WarehouseSyncScheduleTest. You must have 100% test coverage to pass this challenge and assert values to prove that your logic is working as expected.

WarehouseSyncSchedule.apxc

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

WarehouseSyncScheduleTest.apxc

```
@isTest
 public class WarehouseSyncScheduleTest {
 @isTest static void WarehousescheduleTest(){
 String scheduleTime = '00 00 01 * * ?';
 Test.startTest();
 Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
 String jobID=System.schedule('Warehouse Time To Schedule to Test',
 scheduleTime, new WarehouseSyncSchedule());
 Test.stopTest();
 //Contains schedule information for a scheduled job. CronTrigger is similar to a
 cron job on UNIX systems.
 // This object is available in API version 17.0 and later.
 CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
 System.assertEquals(jobID, a.Id,'Schedule ');
 }
 }
```

# Lightning Web Components Basics

# Deploy Lightning Web Component Files

**Create an app page for the bike card component**

Deploy your files to your Trailhead Playground or Developer Edition org and then use Lightning App Builder to create an app page.

**Prework:** If you haven't already completed the activities in the What You Need section of this unit, do that now, or this challenge won't pass. Make sure that both Dev Hub and My Domain are enabled in your org and that the org is authorized with Visual Studio Code.

- Create an SFDX project in Visual Studio Code:
    - Template: **Standard**
    - Project name: `bikeCard`
- Add a Lightning Web Component to the project:
    - Folder: **lwc**
    - Component name: `bikeCard`
- Copy the content for the component files from this unit into your files in Visual Studio Code:
    - bikeCard.html
    - bikeCard.js
    - bikeCard.js-meta.xml
- Deploy the bikeCard component files to your org
- Create a Lightning app page:
    - Label: `Bike Card`
    - Developer Name: `Bike_Card`
    - Add your bikeCard component to the page
    - Activate the page for all users

bikeCard.html

```
<template>
<div>
<div>Name: {name}</div>
<div>Description: {description}</div>
<lightning-badge label={material}></lightning-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>
```

# Add Styles and Data to a Lightning Web Component

**Import the current user’s name into your Lightning app page**

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

**Prework**: You need files created in the previous unit to complete this challenge. If you haven't already completed the activities in the previous unit, do that now.

- Create a Lightning app page:
    - Label: `Your Bike Selection`
    - Developer Name: `Your_Bike_Selection`
- Add the current user's name to the app container:
    - Edit selector.js
    - Edit selector.html

Selector.js (after editing the given code)

```
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.html(after editing the given code)

```
<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>
```