Apex Triggers

Get Started with Apex Triggers

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

```
1  //AccountAddressTrigger
2  trigger AccountAddressTrigger on Account (before insert,before
   update) {
3
4      List<Account> acclst=new List<Account>();
5        for(account a:trigger.new){
6          if(a.Match_Billing_Address__c==true &&
   a.BillingPostalCode!=null){
7          a.ShippingPostalCode=a.BillingPostalCode;
8          }
9      }
10 }
```

Bulk Apex Triggers

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
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> tasklist= new List<Task>();
    for(Opportunity opp: Trigger.New){
        if(opp.Stagename=='Closed Won') {
            tasklist.add(new Task(Subject='Follow Up Test Task',WhatId = opp.Id));
        }
    }
    if(tasklist.size()>0){
        insert tasklist;
    }
}
```

Apex Testing

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, return 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

```
1  //VerifyDate
2  public class VerifyDate {
3      // method to handle potential checks against two dates
4      public static Date CheckDates(Date date1, Date date2) {
5          // if date2 is within the next 30 days of date1, use date2.
   Otherwise use the
6          // end of the month
7          if (DateWithin30Days(date1, date2)) {
8              return date2;
9          } else {
10             return SetEndOfMonthDate(date1);
11         }
12     }
13
14     // method to check if date2 is within the next 30 days of date1
15     @TestVisible
16     private static Boolean DateWithin30Days(Date date1, Date date2) {
17         // check for date2 being in the past
18         if (date2 < date1) {
19             return false;
20         }
21
```

```
22          // check that date2 is within (>=) 30 days of date1
23          Date date30Days = date1.addDays(30); // create a date 30 days
    away from date1
24          if (date2 >= date30Days) {
25              return false;
26          } else {
27              return true;
28          }
29      }
30
31      // method to return the end of the month of a given date
32      @TestVisible
33      private static Date SetEndOfMonthDate(Date date1) {
34          Integer totalDays = Date.daysInMonth(date1.year(),
    date1.month());
35          Date lastDay = Date.newInstance(date1.year(), date1.month(),
    totalDays);
36          return lastDay;
37      }
38
39  }
40
41  // TestVerifyDate
42  @isTest
43  private class TestVerifyDate {
44
45      @isTest static void Test_CheckDates_case1(){
46          Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
    date.parse('01/05/2020'));
47          System.assertEquals(date.parse('01/05/2020'), D);
48      }
49
50      @isTest static void Test_CheckDates_case2(){
51          Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
    date.parse('05/05/2020'));
52          System.assertEquals(date.parse('01/31/2020'), D);
53  }
54
55      @isTest static void Test_DateWithin30Days_case1(){
56          Boolean flag =
    VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('12/30
```

```
57          System.assertEquals(false, flag);
58      }
59
60      @isTest static void Test_DateWithin30Days_case2(){
61          Boolean flag =
    VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('02/02

62          System.assertEquals(false, flag);
63      }
64
65      @isTest static void Test_DateWithin30Days_case3(){
66          Boolean flag =
    VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('01/15

67          System.assertEquals(true, flag);
68      }
69
70      @isTest static void Test_SetEndOfMonthDate(){
71          Date returndate =
    VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
72
73              }
74
75      }
```

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'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

    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

         }

    }

}

    // TestRestrictContactByName
    @isTest
    public class TestRestrictContactByName {

    @isTest static void Test_insertupdateContact(){
        Contact cnt = new Contact();
        cnt.LastName = 'INVALIDNAME';

        Test.startTest();
        Database.SaveResult result = Database.insert(cnt, false);
        Test.stopTest();

        System.assert(!result.isSuccess());
        System.assert(result.getErrors().size() > 0);
        System.assertEquals('The Last Name "INVALIDNAME"is not allowed

    }

    }
```

Create Test Data for Apex Tests

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 >

```
1      // RandomContactFactory
2      public class RandomContactFactory {
3
4      public static List<Contact> generateRandomContacts(Integer
   numcnt, string lastname){
5          List<Contact> contacts = new List<Contact>();
6          for(Integer i=0;i<numcnt;i++){
7              Contact cnt = new Contact(FirstName = 'Test' +i,
   lastname = lastname);
8              contacts.add(cnt);
9          }
10         return contacts;
11     }
12 }
```

Asynchronous Apex

Use Future Methods

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

- Create a field on the Account object:
  - Label: Number Of Contacts
  - Name: Number_Of_Contacts
  - Type: Number
  - This field will hold the total number of Contacts for the Account
- Create an Apex class:
  - Name: AccountProcessor
  - Method name: countContacts
  - The method must accept a List of Account IDs
  - The method must use the @future annotation
  - The method counts the number of Contact records associated to each Account ID passed to the method and updates the 'Number_Of_Contacts__c' field with this value
- Create an Apex test class:
  - Name: AccountProcessorTest
  - The unit tests must cover all lines of code included in the AccountProcessor class, resulting in 100% code coverage.
- Before verifying this challenge, run your test class at least once using the Developer Console Run All feature

```
1 //AccountProcessor
2 public class AccountProcessor {
3   @future
4   public static void countContacts(List<Id> accountIds) {
5     List<Account> accountstoUpdate = new List<Account>();
6     List<Account> accounts = [Select Id, Name, (Select Id from
  Contacts) from Account Where Id IN :accountIds];
```

```
7      // process account records to do awesome stuff
8        for(Account acc:accounts){
9            List<Contact> contactlist = acc.Contacts;
10           acc.Number_Of_Contacts__c = contactlist.size();
11           accountstoUpdate.add(acc);
12       }
13       update accountstoUpdate;
14   }
15 }
16
17 //AccountProcessorTest
18 @IsTest
19 private class AccountProcessorTest {
20   @IsTest
21   private static void testCountContacts() {
22       Account newAccount = new Account(Name='Test Account');
23       insert newAccount;
24
25       Contact newContact1 = new Contact(FirstName='John',
   LastName='Doe', AccountId=newAccount.Id);
26
27        insert newContact1;
28
29       Contact newContact2 = new Contact(FirstName='Jane',
   LastName='Doe', AccountId=newAccount.Id);
30
31        insert newContact2;
32
33           List<Id> accountIds =  new List<Id>();
34
35           accountIds.add(newAccount.Id);
36
37
38     Test.startTest();
39       AccountProcessor.countContacts(accountIds);
40     Test.stopTest();
41
42   }
43 }
```

Use Batch Apex

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

```
1  //LeadProcessor
2  global class LeadProcessor implements Database.Batchable<sObject> {
3   global Integer count = 0;
4
5      global Database.QueryLocator start(Database.BatchableContext
   bc){
6          return Database.getQueryLocator('SELECT ID, LeadSource FROM

7      }
8
9      global void execute(Database.BatchableContext bc, List<Lead>
   L_list){
10         List<lead> L_list_new = new List<lead>();
11
12         for(lead L:L_list){
13             L.leadsource = 'Dreamforce';
14             L_list_new.add(L);
15             count += 1;
```

```
16          }
17          update L_list_new;
18      }
19      global void finish(Database.BatchableContext bc){
20          System.debug('count = '+count);
21      }
22  }
23
24  //LeadProcessorTest
25
26  @isTest
27
28  public class LeadProcessorTest {
29
30        @isTest
31
32        public static void testit() {
33
34        List<lead> L_list = new List<lead>();
35
36       for (Integer i=0; i<200; i++) {
37
38          Lead L = new lead();
39
40          L.LastName = 'name' + i;
41
42          L.Company = 'Company';
43
44          L.Status = 'Random Status';
45
46          L_list.add(L);
47
48     }
49
50         insert L_list;
51
52         Test.startTest();
53
54         LeadProcessor lp = new LeadProcessor();
55
56        Id batchId = Database.executeBatch(lp); Test.stopTest();
```

```
57
58
59     }
60
61 }
```

Control Processes with Queueable Apex

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

```
1 //AddPrimaryContact
2 public class AddPrimaryContact implements Queueable {
3     private Contact con;
4     private String state;
5     public AddPrimaryContact(Contact con,String state) {
```

```
6           this.con = con;
7           this.state = state;
8       }
9       public void execute(QueueableContext context) {
10          List<Account> accounts = [Select Id,Name,(Select FirstName,
    LastName, ID from contacts)
11                                    from Account where BillingState =
    :state Limit 200];
12          List<Contact> primaryContacts = new List<Contact>();
13            for(Account acc:accounts){
14              Contact c = con.clone();
15              c.AccountID = acc.ID;
16                primaryContacts.add(c);
17          }
18          if(primaryContacts.size() > 0){
19              insert primaryContacts;
20          }
21
22      }
23  }
24
25
26
27  //AddPrimaryContactTest
28  @isTest
29  public class AddPrimaryContactTest
30  {
31
32     static testmethod void testQueueable() {
33
34      List<Account> testAccounts = new List<Account> ();
35
36       for (Integer i=0;i<50; i++) {
37
38      testAccounts.add(new
    Account(Name='Account'+i,BillingState='CA'));
39
40       }
41      for (Integer j=0; j<50; j++){
42
43       testAccounts.add(new Account (Name = 'Account'+j,
```

```
  BillingState='NY'));
44
45      }
46
47       insert testAccounts;
48
49        Contact testContact = new Contact(FirstName = 'John', LastName
  = 'Doe');
50
51         insert testContact;
52
53         AddPrimaryContact addit = new addPrimaryContact(testContact,
  'CA');
54         Test.startTest();
55
56         system.enqueueJob(addit);
57
58          Test.stopTest();
59
60        System.assertEquals (50, [Select count() from Contact where
  accountId in (Select Id from Account where BillingState='CA')]);
61
62     }
63
64 }
```

Schedule Jobs Using the Apex Scheduler

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
global class DailyLeadProcessor implements Schedulable {

    global void execute(SchedulableContext ctx) {
        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];

        if(!lList.isEmpty()) {
            for(Lead l: lList) {
                l.LeadSource = 'Dreamforce';
            }
            update lList;
       }
    }
}


//DailyLeadProcessorTest
@isTest
private class DailyLeadProcessorTest {
    @testsetup
    static void setup() {

        List<Lead> leadList = new List<Lead>();

        for(Integer i = 1; i <= 200; i++) {
            Lead ld = new Lead(Company = 'Comp' + i, LastName = 'LN' +i, Status ='Working Contacted');

            leadList.add(ld);

        }

         Insert leadList;
    }
```

```
34
35      static testmethod void testDailyLeadProcessorScheduledJob(){
36
37      string sch = '0 5 12 * * ?';
38
39      Test.startTest();
40
41      string jobId = System.Schedule('ScheduledApexText', sch, new
  DailyLeadProcessor());
42
43      List<Lead> leadList  = [SELECT Id FROM Lead WHERE LeadSource =
  null LIMIT 200];
44      system.assertEquals(200, leadList.size());
45
46      Test.stopTest();
47
48     }
49 }
```

Apex Integration Services

Apex REST Callouts

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

```
1 //AnimalLocator
2 public class AnimalLocator{
3     public static String getAnimalNameById(Integer x){
4         Http http = new Http();
5         HttpRequest req = new HttpRequest();
6         req.setEndpoint('https://th-apex-http-

7         req.setMethod('GET');
8         Map<String, Object> animal= new Map<String, Object>();
```

```
9          HttpResponse res = http.send(req);
10             if (res.getStatusCode() == 200) {
11         Map<String, Object> results = (Map<String,
   Object>)JSON.deserializeUntyped(res.getBody());
12       animal = (Map<String, Object>) results.get('animal');
13         }
14 return (String)animal.get('name');
15     }
16 }
17
18 //AnimalLocatorTest
19 @isTest
20 private class AnimalLocatorTest{
21     @isTest static void AnimalLocatorMock1() {
22         Test.setMock(HttpCalloutMock.class, new
   AnimalLocatorMock());
23         string result = AnimalLocator.getAnimalNameById(3);
24         String expectedResult = 'chicken';
25         System.assertEquals(result,expectedResult );
26     }
27 }
28
29 //AnimalLocatorMock
30 @isTest
31 global class AnimalLocatorMock implements HttpCalloutMock {
32      // Implement this interface method
33     global HTTPResponse respond(HTTPRequest request) {
34         // Create a fake response
35         HttpResponse response = new HttpResponse();
36         response.setHeader('Content-Type', 'application/json');
37         response.setBody('{"animals": ["majestic badger", "fluffy

38         response.setStatusCode(200);
39         return response;
40     }
41 }
```

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 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.

```
1  //ParkLocator
2  public class ParkLocator {
3      public static string[] country(string theCountry) {
4          ParkService.ParksImplPort  parkSvc = new
   ParkService.ParksImplPort(); // remove space
5          return parkSvc.byCountry(theCountry);
6      }
7  }
8
9  // ParkLocatorTest
10 @isTest
11 private class ParkLocatorTest {
12     @isTest static void testCallout() {
13         Test.setMock(WebServiceMock.class, new ParkServiceMock());
```

```
14          String country = 'United States';
15          List<String> result = ParkLocator.country(country);
16          List<String> parks = new List<String>{'Yellowstone',
  'Mackinac National Park', 'Yosemite'};
17           System.assertEquals(parks, result);
18      }
19  }
20
21  //ParkServiceMock
22  @isTest
23  global class ParkServiceMock implements WebServiceMock {
24     global void doInvoke(
25              Object stub,
26              Object request,
27              Map<String, Object> response,
28              String endpoint,
29              String soapAction,
30              String requestName,
31              String responseNS,
32              String responseName,
33              String responseType) {
34          // start - specify the response you want to send
35          ParkService.byCountryResponse response_x = new
  ParkService.byCountryResponse();
36          response_x.return_x = new List<String>{'Yellowstone',
  'Mackinac National Park', 'Yosemite'};
37          // end
38          response.put('response_x', response_x);
39     }
40  }
```

Apex Web Services

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

```
1  //AccountManager
2  @RestResource(urlMapping='/Accounts/*/contacts')
3  global class AccountManager {
4      @HttpGet
5      global static Account getAccount() {
6          RestRequest req = RestContext.request;
7          String accId = req.requestURI.substringBetween('Accounts/',
   '/contacts');
8          Account acc = [SELECT Id, Name, (SELECT Id, Name FROM
   Contacts)
9                         FROM Account WHERE Id = :accId];
10         return acc;
11     }
12 }
13
14 //AccountManagerTest
15 @isTest
16 private class AccountManagerTest {
17
18     private static testMethod void getAccountTest1() {
19         Id recordId = createTestRecord();
20         // Set up a test request
```

```
        RestRequest request = new RestRequest();
        request.requestUri =
  'https://na1.salesforce.com/services/apexrest/Accounts/'+ recordId
  +'/contacts' ;
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);

    }

    // Helper method
        static Id createTestRecord() {
        // Create test record
        Account TestAcc = new Account(
          Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}
```

Apex Specialist Superbadge

There are two kinds of people in this world: those who would travel in a recreational vehicle (RV) and those who wouldn't. The RV community is increasing exponentially across the globe. Over the past few years, HowWeRoll Rentals, the world's largest RV rental company, has increased its global footprint and camper fleet tenfold. HowWeRoll offers travelers superior RV rental and roadside assistance services. Their tagline is, "We have great service, because that's How We Roll!" Their rental fleet includes every style of camper vehicle, from extra large, luxurious homes on wheels to bare bones, retro Winnebagos.

You have been hired as the lead Salesforce developer to automate and scale HowWeRoll's reach. For travelers, not every journey goes according to plan. Unfortunately, there's bound to be a bump in the road at some point along the way. Thankfully, HowWeRoll has an amazing RV repair squad who can attend to any maintenance request, no matter where you are. These repairs address a variety of technical difficulties, from a broken axle to a clogged septic tank.

As the company grows, so does HowWeRoll's rental fleet. While it's great for business, the current service and maintenance process is challenging to scale. In addition to service requests for broken or malfunctioning equipment, routine maintenance requests for vehicles have grown exponentially. Without routine maintenance checks, the rental fleet is susceptible to avoidable breakdowns.

That's where you come in! HowWeRoll needs you to automate their Salesforce-based routine maintenance system. You'll ensure that anything that might cause unnecessary damage to the vehicle, or worse, endanger the customer is flagged. You'll also integrate Salesforce with HowWeRoll's back-office system that keeps track of warehouse inventory. This completely separate system needs to sync on a regular basis with Salesforce. Synchronization ensures that HowWeRoll's headquarters (HQ) knows exactly how much equipment is available when making a maintenance request, and alerts them when they need to order more equipment.

1. Automate record creation using Apex triggers
2. Synchronize Salesforce data with an external system using asynchronous REST

callouts

3. Schedule synchronization using Apex code
4. Test automation logic to confirm Apex trigger side effects
5. Test integration logic using callout mocks
6. Test scheduling logic to confirm action gets queued

Following is the Final Apex Code Workspace:

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, 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',
```

```
34                      Date_Reported__c = Date.Today()
35
36                  );
37
38                  If (maintenanceCycles.containskey(cc.Id)){
39                      nc.Date_Due__c =
  Date.today().addDays((Integer)
  maintenanceCycles.get(cc.Id));
40                  }
41
42                  newCases.add(nc);
43              }
44
45              insert newCases;
46
47              List<Equipment_Maintenance_Item__c> clonedWPs =
  new List<Equipment_Maintenance_Item__c>();
48              for (Case nc : newCases){
49                  for (Equipment_Maintenance_Item__c wp :
  closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__

50                      Equipment_Maintenance_Item__c wpClone =
  wp.clone();
51                      wpClone.Maintenance_Request__c = nc.Id;
52                      ClonedWPs.add(wpClone);
53
54                  }
55              }
56              insert ClonedWPs;
57          }
58      }
59}
```

WarehouseCalloutService.apxc

```
1 public with sharing class WarehouseCalloutService {
2
3     private static final String WAREHOUSE_URL = 'https://th-
4
5     //@future(callout=true)
6     public static void runWarehouseEquipmentSync(){
7
8         Http http = new Http();
9         HttpRequest request = new HttpRequest();
10
11        request.setEndpoint(WAREHOUSE_URL);
12        request.setMethod('GET');
13        HttpResponse response = http.send(request);
14
15
16        List<Product2> warehouseEq = new List<Product2>();
17
18        if (response.getStatusCode() == 200){
19            List<Object> jsonResponse =
  (List<Object>)JSON.deserializeUntyped(response.getBody());
20            System.debug(response.getBody());
21
22            for (Object eq : jsonResponse){
23                Map<String,Object> mapJson =
  (Map<String,Object>)eq;
24                Product2 myEq = new Product2();
25                myEq.Replacement_Part__c = (Boolean)
  mapJson.get('replacement');
26                myEq.Name = (String) mapJson.get('name');
27                myEq.Maintenance_Cycle__c = (Integer)
  mapJson.get('maintenanceperiod');
28                myEq.Lifespan_Months__c = (Integer)
  mapJson.get('lifespan');
29                myEq.Cost__c = (Decimal)
```

```
  mapJson.get('lifespan');
30                myEq.Warehouse_SKU__c = (String)
  mapJson.get('sku');
31                myEq.Current_Inventory__c = (Double)
  mapJson.get('quantity');
32                warehouseEq.add(myEq);
33            }
34
35            if (warehouseEq.size() > 0){
36                upsert warehouseEq;
37                System.debug('Your equipment was synced with

38                System.debug(warehouseEq);
39            }
40        }
41    }
42}
```

WarehouseSyncSchedule.apxc

```
1 global class WarehouseSyncSchedule implements Schedulable
  {
2     global void execute(SchedulableContext ctx) {
3
  WarehouseCalloutService.runWarehouseEquipmentSync();
4     }
5 }
```

MaintenanceRequestHelperTest.apxc

```
1 @istest
2 public with sharing class MaintenanceRequestHelperTest {
3
4     private static final string STATUS_NEW = 'New';
5     private static final string WORKING = 'Working';
```

```
6       private static final string CLOSED = 'Closed';
7       private static final string REPAIR = 'Repair';
8       private static final string REQUEST_ORIGIN = 'Web';
9       private static final string REQUEST_TYPE = 'Routine

10      private static final string REQUEST_SUBJECT = 'Testing

11
12      PRIVATE STATIC Vehicle__c createVehicle(){
13          Vehicle__c Vehicle = new Vehicle__C(name =
    'SuperTruck');
14          return Vehicle;
15      }
16
17      PRIVATE STATIC Product2 createEq(){
18          product2 equipment = new product2(name =
    'SuperEquipment',
19
    lifespan_months__C = 10,
20
    maintenance_cycle__C = 10,
21
    replacement_part__c = true);
22          return equipment;
23      }
24
25      PRIVATE STATIC Case createMaintenanceRequest(id
    vehicleId, id equipmentId){
26          case cs = new case(Type=REPAIR,
27                             Status=STATUS_NEW,
28                             Origin=REQUEST_ORIGIN,
29                             Subject=REQUEST_SUBJECT,
30                             Equipment__c=equipmentId,
31                             Vehicle__c=vehicleId);
32          return cs;
```

```
33      }
34
35      PRIVATE STATIC Equipment_Maintenance_Item__c
  createWorkPart(id equipmentId,id requestId){
36          Equipment_Maintenance_Item__c wp = new
  Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
37
  Maintenance_Request__c = requestId);
38          return wp;
39      }
40
41
42      @istest
43      private static void testMaintenanceRequestPositive(){
44          Vehicle__c vehicle = createVehicle();
45          insert vehicle;
46          id vehicleId = vehicle.Id;
47
48          Product2 equipment = createEq();
49          insert equipment;
50          id equipmentId = equipment.Id;
51
52          case somethingToUpdate =
  createMaintenanceRequest(vehicleId,equipmentId);
53          insert somethingToUpdate;
54
55          Equipment_Maintenance_Item__c workP =
  createWorkPart(equipmentId,somethingToUpdate.id);
56          insert workP;
57
58          test.startTest();
59          somethingToUpdate.status = CLOSED;
60          update somethingToUpdate;
61          test.stopTest();
62
```

```
63          Case newReq = [Select id, subject, type,
  Equipment__c, Date_Reported__c, Vehicle__c, Date_Due__c
64                        from case
65                        where status =:STATUS_NEW];
66
67          Equipment_Maintenance_Item__c workPart = [select
  id
68                                                  from
  Equipment_Maintenance_Item__c
69                                                  where
  Maintenance_Request__c =:newReq.Id];
70
71          system.assert(workPart != null);
72          system.assert(newReq.Subject != null);
73          system.assertEquals(newReq.Type, REQUEST_TYPE);
74          SYSTEM.assertEquals(newReq.Equipment__c,
  equipmentId);
75          SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
76          SYSTEM.assertEquals(newReq.Date_Reported__c,
  system.today());
77      }
78
79      @istest
80      private static void testMaintenanceRequestNegative(){
81          Vehicle__C vehicle = createVehicle();
82          insert vehicle;
83          id vehicleId = vehicle.Id;
84
85          product2 equipment = createEq();
86          insert equipment;
87          id equipmentId = equipment.Id;
88
89          case emptyReq =
  createMaintenanceRequest(vehicleId,equipmentId);
90          insert emptyReq;
```

```
91
92          Equipment_Maintenance_Item__c workP =
  createWorkPart(equipmentId, emptyReq.Id);
93          insert workP;
94
95          test.startTest();
96          emptyReq.Status = WORKING;
97          update emptyReq;
98          test.stopTest();
99
100          list<case> allRequest = [select id
101                                   from case];
102
103          Equipment_Maintenance_Item__c workPart = [select
  id
104                                                    from
  Equipment_Maintenance_Item__c
105                                                    where
  Maintenance_Request__c = :emptyReq.Id];
106
107          system.assert(workPart != null);
108          system.assert(allRequest.size() == 1);
109      }
110
111      @istest
112      private static void testMaintenanceRequestBulk(){
113          list<Vehicle__C> vehicleList = new
  list<Vehicle__C>();
114          list<Product2> equipmentList = new
  list<Product2>();
115          list<Equipment_Maintenance_Item__c> workPartList
  = new list<Equipment_Maintenance_Item__c>();
116          list<case> requestList = new list<case>();
117          list<id> oldRequestIds = new list<id>();
118
```

```
119         for(integer i = 0; i < 300; i++){
120            vehicleList.add(createVehicle());
121             equipmentList.add(createEq());
122         }
123         insert vehicleList;
124         insert equipmentList;
125 
126         for(integer i = 0; i < 300; i++){
127 
  requestList.add(createMaintenanceRequest(vehicleList.get(i
  ).id, equipmentList.get(i).id));
128         }
129         insert requestList;
130 
131         for(integer i = 0; i < 300; i++){
132 
  workPartList.add(createWorkPart(equipmentList.get(i).id,
  requestList.get(i).id));
133         }
134         insert workPartList;
135 
136         test.startTest();
137         for(case req : requestList){
138             req.Status = CLOSED;
139             oldRequestIds.add(req.Id);
140         }
141         update requestList;
142         test.stopTest();
143 
144         list<case> allRequests = [select id
145                                   from case
146                                   where status =:
  STATUS_NEW];
147 
148         list<Equipment_Maintenance_Item__c> workParts =
```

```
  [select id
149
  from Equipment_Maintenance_Item__c
150
  where Maintenance_Request__c in: oldRequestIds];
151
152         system.assert(allRequests.size() == 300);
153     }
154}
```

WarehouseCalloutServiceTest.apxc

```
1 @isTest
2 private class WarehouseCalloutServiceTest {
3     @isTest
4     static void testWareHouseCallout(){
5         Test.startTest();
6         // implement mock callout test here
7         Test.setMock(HTTPCalloutMock.class, new
  WarehouseCalloutServiceMock());
8
  WarehouseCalloutService.runWarehouseEquipmentSync();
9         Test.stopTest();
10        System.assertEquals(1, [SELECT count() FROM
  Product2]);
11    }
12}
```

WarehouseCalloutServiceMock.apxc

```
1 @isTest
2 global class WarehouseCalloutServiceMock implements
  HttpCalloutMock {
3     // implement http mock callout
```

```
4      global static HttpResponse respond(HttpRequest
  request){
5
6          System.assertEquals('https://th-superbadge-
  ));
7          System.assertEquals('GET', request.getMethod());
8
9          // Create a fake response
10         HttpResponse response = new HttpResponse();
11         response.setHeader('Content-Type',
  'application/json');
12
  response.setBody('[{"_id":"55d66226726b611100aaf741","repl




13         response.setStatusCode(200);
14         return response;
15     }
16}
```

WarehouseSyncScheduleTest.apxc

```
1 @isTest
2 public class WarehouseSyncScheduleTest {
3
4      @isTest static void WarehousescheduleTest(){
5          String scheduleTime = '00 00 01 * * ?';
6          Test.startTest();
7          Test.setMock(HttpCalloutMock.class, new
  WarehouseCalloutServiceMock());
8          String jobID=System.schedule('Warehouse Time To

  WarehouseSyncSchedule());
9          Test.stopTest();
```

```
10          //Contains schedule information for a scheduled
  job. CronTrigger is similar to a cron job on UNIX systems.
11          // This object is available in API version 17.0
  and later.
12          CronTrigger a=[SELECT Id FROM CronTrigger where
  NextFireTime > today];
13          System.assertEquals(jobID, a.Id,'Schedule ');
14
15
16      }
17}
```