

1.Apex Triggers:

1)Get Started with Apex Triggers:

Challenge:

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

Code For Apex Triggers:

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

```
for(Account account: Trigger.New){
if(account.Match_Billing_Address__c == True){
account.ShippingPostalCode =account.BillingPostalCode;
```

```
    }  
  }  
}
```

2)Bulk Apex Triggers:

Challenge:

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

Code For Bulk Apex Triggers:

```
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;  
    }  
}
```

2.Apex Testing:

1.Get Started with Apex Unit Tests:

Challenge:

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

Code for VerifyDate:

```
public class VerifyDate {  
  
    //method to handle potential checks against two dates  
    public static Date CheckDates(Date date1, Date date2) {
```

//if date2 is within the next 30 days of date1, use date2.
Otherwise use the end of the month

```
        if(DateWithin30Days(date1,date2)) {  
            return date2;  
        } else {  
            return SetEndOfMonthDate(date1);  
        }  
    }
```

 //method to check if date2 is within the next 30 days of
date1

```
        @TestVisible private static Boolean DateWithin30Days(Date  
date1, Date date2) {  
            //check for date2 being in the past  
            if( date2 < date1) { return false; }  
            //check that date2 is within (>=) 30 days of date1  
            Date date30Days = date1.addDays(30); //create a date 30  
days away from date1  
            if( date2 >= date30Days ) { return false; }  
            else { return true; }  
        }
```

 //method to return the end of the month of a given date
 @TestVisible private static Date SetEndOfMonthDate(Date
date1) {
 Integer totalDays = Date.daysInMonth(date1.year(),
date1.month());

```

        Date lastDay = Date.newInstance(date1.year(),
date1.month(), totalDays);
        return lastDay;
    }

```

```

}

```

Code for TestVerifyDate:

```

@Test

```

```

private class TestVerifyDate {

```

```

    @Test static void Test_CheckDates_case1(){

```

```

        Date D =

```

```

VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/0
5/2020'));

```

```

        System.assertEquals(date.parse('01/05/2020'), D);

```

```

    }

```

```

    @Test static void Test_CheckDates_case2(){

```

```

        Date D =

```

```

VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('05/0
5/2020'));

```

```

        System.assertEquals(date.parse('01/31/2020'), D);

```

```

    }

```

```

    @Test static void Test_DateWithin30Days_case1(){

```

```

        Boolean flag =

```

```

VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.pars
e('12/30/2019'));

```

```

        System.assertEquals(false, flag);

```

```

    }
    @isTest static void Test_DateWithin30Days_case2(){
        Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.pars
e('02/02/2020'));
        System.assertEquals(false, flag);
    }
    @isTest static void Test_DateWithin30Days_case3(){
        Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.pars
e('01/15/2020'));
        System.assertEquals(true, flag);
    }
    @isTest static void Test_SetEndOfMonthDate(){
Datereturndate=
VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
    }
}

```

2.Test Apex Triggers:

challenge:

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

Code for 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
allowed for DML');
        }
    }
}

```

Code for 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 for DML', result.getErrors()[0].getMessage());
    }
}
```

3.Create Test Data for Apex Tests:

Challenge:

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 >`

Code for RandomContactFactory:

```
public class RandomContactFactory {
```



```

    public static List<Contact> generateRandomContacts(Integer
numcnt,string lastname){
        List<Contact> contacts= new List<Contact>();
        for(Integer i=0;i<numcnt;i++){
            Contact cnt = new Contact(FirstName = 'Test'+i, LastName
= lastname);
            contacts.add(cnt);
        }
        return contacts;
    }
}

```

3.Asynchronous Apex:

1.Use Future Methods:

challenge:

Create an Apex class that uses the @future annotation to update Account records.

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

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) from Account Where Id in :accountIds];

        For(Account acc:accounts){
            List<Contact> contactList = acc.Contacts;
            acc.Number_of_Contacts__c = contactList.size();
            accountsToUpdate.add(acc);

        }
        update accountsToUpdate;
    }
}
```

```
}  
}
```

Code for AccountProcessorTest:

```
@IsTest  
private 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();  
    }  
}
```

```
}
```

3.Use Batch Apex:

challenge:

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

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_list){
        list<lead> L_list_new = new List<lead>();

        for(lead L:L_list){
            L.leadsource = 'Dreamforce';
            L_list_new.add(L);
            count += 1;
        }
        update L_list_new;
    }

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

}

```

Code for LeadProcessorTest:

```

@Test
public class LeadProcessorTest {

    @Test
    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;

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

```

4.Control Processes with Queueable Apex:

challenge:

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

Code for AddPrimaryContact:

```
public class AddPrimaryContact implements Queueable{
    Contact con;
    String state;

    public AddPrimaryContact(Contact con, String state){
        this.con = con;
        this.state = state;
    }

    public void execute(QueueableContext qc){
        List<Account> lstOfAccs = [SELECT Id FROM Account
WHERE BillingState = :state LIMIT 200];
```

```

List<Contact> IstOfConts = new List<Contact>();
for(Account acc : IstOfAccs){
    Contact conInst = con.clone(false,false,false,false);
    conInst.AccountId = acc.Id;

    IstOfConts.add(conInst);
}

INSERT IstOfConts;
}
}

Code for AddPrimaryContactTest:
@Test
public class AddPrimaryContactTest{
    @testSetup
    static void setup(){
        List<Account> IstOfAcc = new List<Account>();
        for(Integer i = 1; i <= 100; i++){
            if(i <= 50)
                IstOfAcc.add(new Account(name='AC'+i, BillingState =
'NY'));
            else
                IstOfAcc.add(new Account(name='AC'+i, BillingState =
'CA'));
        }

        INSERT IstOfAcc;
    }
}

```



```

    }

    static testmethod void testAddPrimaryContact(){
        Contact con = new Contact(LastName = 'TestCont');
        AddPrimaryContact addPCIns = new
AddPrimaryContact(CON, 'CA');

        Test.startTest();
        System.enqueueJob(addPCIns);
        Test.stopTest();

        System.assertEquals(50, [select count() from Contact]);
    }
}

```

5.Schedule Jobs Using the Apex Scheduler:

Challenge:

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

Code for DailyLeadProcessor:

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead
WHERE LeadSource = "];

        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();

            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }

            update newLeads;
        }
    }
}
```

Code for DailyLeadProcessorTest:

```
@isTest
private class DailyLeadProcessorTest{
    //Seconds Minutes Hours Day_of_month Month Day_of_week
    optional_year
    public static String CRON_EXP = '0 0 0 2 6 ? 2022';

    static testmethod void testScheduledJob(){
        List<Lead> leads = new List<Lead>();

        for(Integer i = 0; i < 200; i++){
            Lead lead = new Lead(LastName = 'Test ' + i, LeadSource =
            ", Company = 'Test Company ' + i, Status = 'Open - Not Contacted');
            leads.add(lead);
        }

        insert leads;

        Test.startTest();
        // Schedule the test job
        String jobId = System.schedule('Update LeadSource to
        DreamForce', CRON_EXP, new DailyLeadProcessor());

        // Stopping the test will run the job synchronously
        Test.stopTest();
    }
}
```

}

4.Apex Integration Services:

1.Apex REST Callouts:

Challenge:

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

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 AnimalLocatorTest:

```

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

```

```
}  
}
```

2.Apex SOAP Callouts:

Challenge:

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.

Pework: 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.

Code for ParkService:

//Generated by wsdl2apex

```
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;
    }

```



```
}  
}
```

Code for ParkLocator:

```
public class ParkLocator {  
    public static string[] country(string theCountry) {  
        ParkService.ParksImplPort parkSvc = new  
ParkService.ParksImplPort();  
        return parkSvc.byCountry(theCountry);  
    }  
}
```

Code for ParkLocatorTest:

```
@isTest  
private class ParkLocatorTest {  
    @isTest static void testCallout() {  
        Test.setMock(WebServiceMock.class, new ParkServiceMock  
());  
        String country = 'United States';  
        List<String> result = ParkLocator.country(country);  
        List<String> parks = new List<String>{'Yellowstone',  
'Mackinac National Park', 'Yosemite'};  
        System.assertEquals(parks, result);  
    }  
}
```

3.Apex Web Services:

challenge:

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.

Pework: 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

Code for AccountManager:

```
@RestResource(urlMapping='/Accounts/*/contacts')
global 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;
}
}

```

Code for AccountManagerTest:

```

@Test
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri=
'https://na1.salesforce.com/services/apexrest/Accounts/'+record
Id +'/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;
}
}

```

5.Apex Specialist:

Challenge:

1.Automated Record Creation:

Code for 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));
    } 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> 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;
}
}
}
}

```

Code for MaintenanceRequest.apxt:

trigger MaintenanceRequest on Case (before update, after
update) {

```
if(Trigger.isUpdate && Trigger.isAfter){  
    MaintenanceRequestHelper.updateWorkOrders(Trigger.New,  
Trigger.OldMap);  
}  
}
```

Challenge:

2.Synchronize Salesforce data with an external system:

Code for WarehouseCalloutService.apxc :-

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

//class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.

//The callout's JSON response returns the equipment records that you upsert in Salesforce.

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



```

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

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

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

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

    //class maps the following fields: replacement part
    (always true), cost, current inventory, lifespan, maintenance
    cycle, and warehouse SKU
    //warehouse SKU will be external ID for identifying which
    equipment records to update within Salesforce
    for (Object eq : jsonResponse){
        Map<String,Object> mapJson =
(Map<String,Object>)eq;
        Product2 myEq = new Product2();
        myEq.Replacement_Part__c = (Boolean)
mapJson.get('replacement');
        myEq.Name = (String) mapJson.get('name');
        myEq.Maintenance_Cycle__c = (Integer)

```

```

mapJson.get('maintenanceperiod');
    myEq.Lifespan_Months__c = (Integer)
mapJson.get('lifespan');
    myEq.Cost__c = (Integer) mapJson.get('cost');
    myEq.Warehouse_SKU__c = (String)
mapJson.get('sku');
    myEq.Current_Inventory__c = (Double)
mapJson.get('quantity');
    myEq.ProductCode = (String) mapJson.get('_id');
    warehouseEq.add(myEq);
}

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

public static void execute (QueueableContext context){
    runWarehouseEquipmentSync();
}

}

```

Challenge:

3.Schedule synchronization using Apex code:

Code for WarehouseSyncSchedule.apxc :-

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

Challenge:

4.Test automation logic:

Code for 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).i
d, 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);
    }
}

```

Code for 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;

```

```
}  
}  
}
```

Code for MaintenanceRequest.apxt :-

trigger MaintenanceRequest on Case (before update, after update) {

 if(Trigger.isUpdate && Trigger.isAfter){

 MaintenanceRequestHelper.updateWorkOrders(Trigger.New,
 Trigger.OldMap);

 }
}

Challenge:

5.Test callout logic:

Code for WarehouseCalloutService.apxc :-

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 WarehouseCalloutServiceTest.apxc :-

```

@Test

private class WarehouseCalloutServiceTest {
    @Test
    static void testWareHouseCallout(){
        Test.startTest();
        // implement mock callout test here
        Test.setMock(HTTPCalloutMock.class, new
WarehouseCalloutServiceMock());
        WarehouseCalloutService.runWarehouseEquipmentSync();
        Test.stopTest();
        System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}

```

```
}
```

Code for 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","replac  
ement":false,"quantity":5,"name":"Generator 1000  
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"10  
0003"}');
```

```
        response.setStatusCode(200);
```

```
        return response;
```

```
    }
```

```
}
```

Challenge:

6.Test scheduling logic:

Code for WarehouseSyncSchedule.apxc :-


```

global class WarehouseSyncSchedule implements Schedulable
{
    global void execute(SchedulableContext ctx) {

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}

```

Code for 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 ');
    }
}

```

}

}