NAME: NALIN BEDI
APPLIED TRACK: DEVELOPER
SLOT: 6PM-9PM
COLLEGE NAME: Maharaja Agrasen Instiute of Technology, Delhi
email ID: nalinbedi2871@gmail.com
trailhead profile URL: https://trailblazer.me/id/nbedi8

SALESFORCE SUPPORTED VIRTUAL INTERNSHIP PROGRAM

APEX CODES(trailmix +superbadges)

Trailmix followed: Salesforce Developer Catalyst

superbadges: 1. Process Automation Specialist
2. Apex Specialist

APEX TRIGGERS MODULE:

GET STARTED WITH APEX TRIGGERS

CHALLENGE:

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:

```
//trigger to set Shipping Postal Code equal to Billing Postal Code if  Match Billing Address is checked
trigger AccountAddressTrigger on Account (before insert,before update) {
   for(Account a:trigger.new){
      if(a.Match_Billing_Address__c==true && a.BillingPostalCode!=null)
      {
          a.ShippingPostalCode=a.BillingPostalCode;
      }
   }
}
```

BULK APEX TRIGGERS:

CHALLENGE:

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:

```
//trigger adds a follow-up task to an opportunity if its stage is Closed Won
trigger ClosedOpportunityTrigger on Opportunity (after insert,after update) {
   List<Task> opptask=new List<Task>();
   List<Opportunity> opplist=new List<Opportunity>([SELECT Id,StageName,
                                                    (Select WhatId,Subject FROM Tasks)
                                                     FROM Opportunity
                                                     WHERE Id IN:Trigger.new AND StageName
                                                      LIKE'%Closed Won%']);
   //adding the tasks to opptask list created above
   for(Opportunity opp:opplist)  //iterating over 'opplist'
   {
     opptask.add(new Task(WhatId=opp.Id,Subject='Follow Up Test Task'));
   }
   if(opptask.size()>0)
   {
      insert opptask;
   }
}
```

APEX TESTING MODULE:

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:

VerifyDate class:

```
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
```

```
        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 class:

```
@isTest
public class TestVerifyDate
{
   static testMethod void TestVerifyDate()
   {
     //method to test code coverage of CheckDates() method
     VerifyDate.CheckDates(System.today(),System.today().addDays(10));
     VerifyDate.CheckDates(System.today(),System.today().addDays(78));
   }

}
```

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:

RestrictContactByName trigger:

```
trigger RestrictContactByName on Contact (before insert) {
   //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 test class:

```
@isTest
public class TestRestrictContactByName {
   static testMethod void  metodoTest()
   {
      List<Contact> listContact= new List<Contact>();
      Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' , email='Test@test.com');
      Contact c2 = new Contact(FirstName='Francesco1', LastName =
                                             'INVALIDNAME',email='Test@test.com');
      listContact.add(c1);
      listContact.add(c2);

      Test.startTest();
      try
      {
         insert listContact;    //inserting 'listContact' list of type <Contact> into the database
      }
      catch(Exception ee)
      {
         System.debug('Insert failed');   //if exception caught,print the message in debug logs
      }
      Test.stopTest();     //end the test
   }
}
```

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:

```
public class RandomContactFactory {
        //method to return list of Contacts
        public static List<Contact> generateRandomContacts( Integer noOfContacts, String lastName )
        {
          List<Contact> contacts = new List<Contact>();     //list of type Contact initialized
          for(Integer i = 0; i < noOfContacts; i++ )
          {
                  Contact con = new Contact( FirstName = 'Test '+i, LastName = lastName );
                  contacts.add( con );       //adding contacts initialised above to the list
          }
          return contacts;
        }
}
```

ASYNCHRONOUS APEX MODULE:

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:

AccountProcessor class:

```
public class AccountProcessor {
  @future
//method to  count the number of Contact records associated to each Account ID
public static void countContacts(Set<id> setId)
  {
     //list of Account type created to fetch Id, no. of contacts associated with each account
     List<Account> listAccount = [select id,Number_of_Contacts__c , (select id from contacts ) from
                                            account where id in :setId];
     for( Account acc : listAccount )
     {
        List<Contact> listCont = acc.contacts ;     //List of Contact type initialised with ID of all contacts
                                                     //associated with that account
        acc.Number_of_Contacts__c = listCont.size();  //no. of contacts = size of list listCont
     }
     update listAccount;        //update the changes to database
  }

}
```

AccountProcessorTest test class:

```
@isTest
public class AccountProcessorTest {
   public static testmethod void TestAccountProcessorTest()        //test method created
   {
      Account a = new Account();              //sObject variable of Account Object created
      a.Name = 'Test Account';                //Name field value given to Account sObject variable
      Insert a;                               //new Account record inserted to database
      Contact cont = New Contact();           //sObject variable of Contact Object created
      cont.FirstName ='Bob';                  //FirstName field value given to Contact sObject variable
      cont.LastName ='Masters';               //LastName field value given to Contact sObject variable
      cont.AccountId = a.Id;                  //assigning AccountId of contact to ID of Account
      Insert cont;                            //new Contact record inserted to database
      set<Id> setAccId = new Set<ID>();       //Set of ID type initialised
      setAccId.add(a.id);                     //adding Account ID to set initialised above
      Test.startTest();                       //start the test
      AccountProcessor.countContacts(setAccId);
      Test.stopTest();                        //end the test
      Account ACC = [select Number_of_Contacts__c from Account where id = :a.id LIMIT 1];
      //method to test and compare the values passed as parameters
      System.assertEquals ( Integer.valueOf(ACC.Number_of_Contacts__c) ,1);
  }
}
```

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:

LeadProcessor class:

```
global class LeadProcessor implements Database.Batchable<Sobject> {
    //method to collect LeadSource from all lead records
   global Database.QueryLocator start(Database.BatchableContext bc)
   {
      return Database.getQueryLocator([Select LeadSource From Lead ]);
   }
   //method to update all Lead records in the org with the LeadSource value of Dreamforce
   global void execute(Database.BatchableContext bc, List<Lead> scope)
   {
         for (Lead Leads : scope)       //iterate over the records collected in start() method above
         {
            Leads.LeadSource = 'Dreamforce';     //update the LeadSource of all records
         }
         update scope;              //commit the updates to database
   }
   global void finish(Database.BatchableContext bc){   }

}
```

LeadProcessorTest test class:

```
@isTest
public class LeadProcessorTest {
   //method to insert Lead test records into the database
   static testMethod void testMethod1()
   {
      List<Lead> lstLead = new List<Lead>();
      for(Integer i=0 ;i <200;i++)
      {
         Lead led = new Lead();
         led.FirstName ='FirstName';
         led.LastName ='LastName'+i;
         led.Company ='demo'+i;
         lstLead.add(led);
      }
      insert lstLead;               //insert records into the database

      Test.startTest();             //start the test
       LeadProcessor obj = new LeadProcessor();    //new object of LeadProcessor initialised
       DataBase.executeBatch(obj);        //to execute batch class
      Test.stopTest();                //stop the test
   }

}
```

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:

AddPrimaryContact class:

```
public class AddPrimaryContact implements Queueable{
   private String st;
   private Contact primecontact;
   //constructor to initialise String and Contact sObject variable
   public AddPrimaryContact(Contact aContact, String aState)
   {
        this.st=aState;
        this.primecontact = aContact;
    }
    //method to insert the Contact sObject record associated to each Account.
    public void execute(QueueableContext context)
    {
        //200 records inserted into list
        List<Account> accounts = [select name from account where billingstate=:st LIMIT 200];
        List<Contact> contacts = new List<Contact>();  //new list initialist of sObject type Contact
        for (Account acc : accounts) {
          contact con=primecontact.clone(false,false,false,false);
          contacts.add(con);
        }
        insert contacts;          //insert records added in list to database
    }
}
```

AddPrimaryContactTest test class:

```
@isTest
public class AddPrimaryContactTest {
    @testSetup
    static void setup() {
        List<Account> accounts = new List<Account>();
        // add 50 NY account
        for (Integer i = 0; i < 50; i++) {
        accounts.add(new Account(Name='NY'+i, billingstate='NY'));
        }
         // add 50 CA account
        for (Integer j = 0; j < 50; j++) {
        accounts.add(new Account(Name='CA'+j, billingstate='CA'));
          }
        insert accounts;
      }
       static testmethod void testQueueable(){
        contact a=new contact(Lastname='mary', Firstname='rose');
        Test.startTest();         //start the test
        AddPrimaryContact updater=new AddPrimaryContact(a, 'CA');
        System.enqueueJob(updater);       //method to enqueue the task into database
        Test.stopTest();        //stop the test
        //method to confirm whether records with LastName='mary' and FirstName='rose' are 50
        System.assertEquals(50, [SELECT count() FROM Contact WHERE Lastname='mary' AND
                                                Firstname='rose']) ;
    }
}
```

SCHEDULE JOBS USING 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:

DailyLeadProcessor test:

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource =' '];
        //Id,LeadSource of Lead records with blank LeadSource field inserted into the list
        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();     //new list initialised
            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';               //assign LeadSource field with value 'DreamForce'
                newLeads.add(lead);                           //add the records into new list created above
            }
            update newLeads;                                 //update the changes to records in database
        }
    }

}
```

DailyLeadProcessorTest test class:

```
@isTest
public 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';
    //method to insert 200 Lead records
    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();
    }
}
```

LIGHTNING WEB COMPONENTS BASICS MODULE:

DEPLOY LIGHTNING WEB COMPONENTS FILES:

CHALLENGE:

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

CODE:

bikeCard.Html file:

```
<template>
```

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

bikeCard.js file:

```
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 file:

```
<?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:

CHALLENGE:

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

CODE:

selector.html file:

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

```
    </div>
</template>
```

selector.js file:

```
import { LightningElement, wire } from 'lwc';
import { getRecord, getFieldValue } from 'lightning/uiRecordApi';
import Id from '@salesforce/user/Id';
import NAME_FIELD from '@salesforce/schema/User.Name';
const fields = [NAME_FIELD];
export default class Selector extends LightningElement {
    selectedProductId;
    handleProductSelected(evt) {
        this.selectedProductId = evt.detail;
    }
    userId = Id;
    @wire(getRecord, { recordId: '$userId', fields })
    user;
    get name() {
        return getFieldValue(this.user.data, NAME_FIELD);
    }
}
```

selector.js-meta.xml file:

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

```
</LightningComponentBundle>
```

selector.css file:

```
body {
  margin: 0;
}
.wrapper{
  min-height: 100vh;
  background: #ccc;
  display: flex;
  flex-direction: column;
}
.header, .footer{
  height: 50px;
  background: rgb(255, 255, 255);
  color: rgb(46, 46, 46);
  font-size: x-large;
  padding: 10px;
}
.content {
  display: flex;
  flex: 1;
  background: #999;
  color: #000;
}
.columns{
  display: flex;
  flex:1;
}
.main{
  flex: 1;
  order: 2;
  background: #eee;
}
.sidebar-first{
  width: 20%;
  background: #ccc;
  order: 1;
}
.sidebar-second{
  width: 30%;
  order: 3;
  background: #ddd;
}
```

APEX INTEGRATION SERVICES MODULE

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:

AnimalLocator class

```
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 test class:

```
@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);  //method to compare both parameter values
    }
}
```

AnimalLocatorMock class:

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPResponse respond(HTTPRequest request) {
        HttpResponse response = new HttpResponse();
       response.setHeader('Content-Type', 'application/json');
       response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
       response.setStatusCode(200);
       return response;
   }
}
```

APEX SOAP CALLOUTS

CHALLENGE

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.

<u>CODE:</u>

ParkLocator class:

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

ParkServiceMock test class:

```
@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();
         response_x.return_x = new List<String>{'Garner State Park', 'Fowler Park', 'Hoosier National
                                                       Forest Park'};
            response.put('response_x',response_x);
      }
}
```

ParkLocatorTest test class:

```
@isTest
private class ParkLocatorTest {
   testMethod static void testCallout(){
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'United States';
        String[] result = ParkLocator.country(country);
       System.assertEquals(new List<String>{'Garner State Park', 'Fowler Park', 'Hoosier National Forest
                                                                Park'}, result);
    }
}
```

APEX WEB SERVICES

CHALLENGE

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

CODE:

AccountManager class:

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

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

AccountManagerTest class:

```
@isTest
private class AccountManagerTest{
   static testMethod void testMethod1(){
      Account objAccount = new Account(Name = 'test Account');
      insert objAccount;
      Contact objContact = new Contact(LastName = 'test Contact',
                                       AccountId = objAccount.Id);
      insert objContact;
      Id recordId = objAccount.Id;
      RestRequest request = new RestRequest();
      request.requestUri =
         'https://sandeepidentity-dev-ed.my.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 Account', thisAccount.Name);
   }
}
```

APEX SPECIALIST SUPERBADGE

CHALLENGE-2:

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.

CODE:

MaintenanceRequestHelper class:

```
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;
}
```

```
    }
}
```

MaintenanceRequest Trigger:

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

CHALLENGE-3:

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**.

CODE:

WarehouseCalloutService class:

```
public with sharing class WarehouseCalloutService implements Queueable{

    //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 are upserted in Salesforce.
   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());

           /*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();
}
}
```

ExecuteAnonymousWindow:

```
System.enqueueJob(new WarehouseCalloutService());
```

CHALLENGE-4:

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.**

CODE:

WarehouseSyncShedule.apxc class:

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

CHALLENGE-5:

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.

CODE:

MaintenanceRequestHelperTest.apxc class:

```
@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
                         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 class:

```
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;
      }
   }
}
```

CHALLENGE-6:

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.

CODE:

WarehouseCalloutService class:

```
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);
        }

      }
   }
}
```

WarehouseCalloutServiceTest class:

```
@IsTest
private class WarehouseCalloutServiceTest {
   // implement your mock callout test here
   @isTest
   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]);
   }
}
```

WarehouseCalloutServiceMock class:

```
@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,"quantity":5,"name":
                  "Generator  1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
      response.setStatusCode(200);
      return response;
   }
}
```

CHALLENGE-7:

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.

CODE:

WarehouseSyncSchedule class:

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

```
    }
}
```

WarehouseSyncScheduleTest class:

```
@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 a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');
    }
}
```