https://trailblazer.me/id/karankbhoi

Module:Apex, Testing And Debugging

Apex Triggers:-

Get Started with Apex Triggers:

File Name- AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert,before update) {
for(Account acc:Trigger.New)
 {
    if(acc.Match_Billing_Address__C){
      acc.ShippingPostalCode = acc.BillingPostalCode;
       System.debug(JSON.serializePretty(acc));
    }
}
}
```

Bulk Apex Triggers:-

Create a Bulk Apex trigger

File Name- ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (before insert) {

   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;
   }
}
```

Module:Apex Testing

Get Started with Apex Unit Tests:-

https://trailblazer.me/id/karankbhoi

Create a Unit Test for a Simple Apex Class:

File Name- `VerifyDate`.apxc

```
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;
	}

}
```

Test Apex Triggers:-

Create a Unit Test for a Simple Apex Trigger:

File Name- `RestrictContactByName`.apxt

```
trigger RestrictContactByName on Contact (before insert, before update) {
```

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

        }
}
```

Create Test Data for Apex Tests:-

Create a Contact Test Factory:

File Name- RandomContactFactory.apxc

```
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;
   }
}
```

Module:Asynchronous Apex:-

Use Future Methods:-

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

https://trailblazer.me/id/karankbhoi

File Name- AccountProcessor.apxc

```
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];
    // process account records to do awesome stuff
    For(Account acc:accounts)
    {
       List<Contact> contactList = acc.Contacts;
       acc.Number_of_Contacts__c = contactList.size();
       accountsToUpdate.add(acc);

    }
     update accountsToUpdate;
  }
}
```

File Name- AccountProcessorTest.apxc

```
@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='John',LastName='Doe',AccountID=newAccount.Id);
     insert newContact2;
     List<Id>accountIds =new List<Id>();
     accountIds.add(newAccount.Id);

     accountProcessor.countContacts(accountIds);
     Test.startTest();
     accountProcessor.countContacts(accountIds);
```

```
    Test.stopTest();
    // runs callout and check results
  }
}
```

Use Batch Apex:-

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

File Name- LeadProcessor.apxc

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

    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator(
            'SELECT ID from Lead'
        );
    }
    global void execute(Database.BatchableContext bc, List<Lead> scope){
        // process each batch of records
          List<Lead> leads =new List<Lead>();
          for (Lead lead : scope) {
            lead.LeadSource ='Dreamforce';
            leads.add(lead);
           }
        update leads;
    }
    global void finish(Database.BatchableContext bc)
    {

    }
}
```

File Name- LeadProcessorTest.apxc

```
@isTest
private class LeadProcessorTest {
    @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
```

```
        // insert 10 accounts
        for (Integer i=0;i<200;i++) {
            Leads.add(new Lead(Lastname='Lead'+i,Company='Test Co'));
        }
        insert leads;

    }
    @isTest static void test() {
        Test.startTest();
        LeadProcessor myLeads = new LeadProcessor();
        Id batchId = Database.executeBatch(myLeads);
        Test.stopTest();
        // after the testing stops, assert records were updated properly
        System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);
    }
}
```

Control Processes with Queueable Apex:-

Create a Queueable Apex class that inserts Contacts for Accounts:-

File Name- **AddPrimaryContact.apxc**

```
public class AddPrimaryContact implements Queueable {
    private Contact con;
    private String state;
    public AddPrimaryContact(Contact con, String state) {
        this.con = con;
        this.state = state;
    }
    public void execute(QueueableContext context) {
     List<Account>accounts =[Select Id,Name,(Select FirstName,LastName,Id from contacts)from Account where BillingState =:state Limit 200];
     List<Contact> primaryContacts= new List<Contact>();
```

```
    for(Account acc:accounts)
     {
        Contact c = con.clone();
        c.AccountId =acc.Id;
        primaryContacts.add(c);
     }
      if(primaryContacts.size()>0)
      {
         insert primaryContacts;
      }
   }
}
```

File Name- AddPrimaryContactTest.apxc

```
@isTest
public class AddPrimaryContactTest {

  static testmethod void testQueueable()
   {
      List<Account> testAccounts =new List<Account>();
      for(Integer i=0;i<50;i++)
      {
         testAccounts.add(new Account(Name='Account'+i,BillingState='CA'));
      }
      for(Integer j=0;j<50;j++)
      {
         testAccounts.add(new Account(Name='Account'+j,BillingState='NY'));
      }
      insert testAccounts;

      Contact testContact =new Contact(FirstName='John',LastName='Doe');
      insert testContact;
      AddPrimaryContact addit =new addPrimaryContact(testContact,'CA');
      Test.startTest();
      System.enqueueJob(addit);
      Test.stopTest();
```

```
      System.assertEquals(50,[select count()from Contact Where accountId in(Select Id from
Account where BillingState='CA')]);
   }
}
```

Schedule Jobs Using the Apex Scheduler:-

Create an Apex class that uses Scheduled Apex to update Lead records:

File Name- DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable {

   global void execute(SchedulableContext ctx) {

      //Retrieving the 200 first leads where lead source is in blank.
      List<Lead> leads = [SELECT ID, LeadSource FROM Lead where LeadSource = ''
LIMIT 200];

      //Setting the LeadSource field the 'Dreamforce' value.
      for (Lead lead : leads) {
         lead.LeadSource = 'Dreamforce';
      }

      //Updating all elements in the list.
      update leads;
   }

}
```

File Name- DailyLeadProcessorTest.apxc

```
@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();
    }
}
```

Intergration:-

Module:Apex Integration Services:-

Apex REST Callouts:

Create an Apex class that calls a REST endpoint and write a test class.

File Name- AnimalLocator.apxc

```
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');
    }
}
```

https://trailblazer.me/id/karankbhoi

File Name- AnimalLocatorTest.apxc

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

File Name- AnimalLocatorMock.apxc

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    // Implement this interface method
   global HTTPResponse respond(HTTPRequest request) {
      // Create a fake response
      HttpResponse response = new HttpResponse();
      response.setHeader('Content-Type', 'application/json');
      response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
      response.setStatusCode(200);
      return response;
   }
}
```

Apex SOAP Callouts:-

Generate an Apex class using WSDL2Apex and write a test class.

File Name- ParkService.apxc

```
//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;
        }
```

```
    }
}
```

File Name- ParkLocator.apxc

```
public class ParkLocator {
    public static List<String>country(String country)
    {
        ParkService.ParksImplPort parkservice = new parkService.ParksImplport();
         return parkservice.byCountry(country);
    }

}
```

File Name- ParkLocatoreTest.apxc

```
@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>();
          parks.add('Yosemite');
          parks.add('Yellowstone');
          parks.add('Another Park');
        System.assertEquals(parks, result);
    }
}
```

File Name- ParkServiceMock.apxc

```
@isTest
global class ParkServiceMock implements WebServiceMock
{
    global void doInvoke(
    Object stud,
    Object request,
    Map<String,Object>response,
    String endpoint,
    String soapAction,
    String requestName,
    String responseNS,
    String responseName,
    String responseType)
    {
```

```
    List<String>parks =new List<String>();
      parks.add('Yosemite');
      parks.add('Yellowstone');
      parks.add('Another Park');
    ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
    response_x.return_x = parks;
    response.put('response_x',response_x);
   }
}
```

Apex Web Services:-

Create an Apex REST service that returns an account and its contacts:

File Name- AccountManager.apxc

```
@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;
   }
}
```

File Name- AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {

   private static testMethod void getAccountTest1() {
      Id recordId = createTestRecord();
      // Set up a test request
      RestRequest request = new RestRequest();
      request.requestUri = 'https://empathetic-fox-635u6q-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts' ;
      request.httpMethod = 'GET';
      RestContext.request = request;
      // Call the method to test
```

https://trailblazer.me/id/karankbhoi

```
        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;
    }
}
```

Module:Apex Triggers:-

Get Started with Apex Triggers:-

Create an Apex trigger

File Name- AccountAddressTrigger.apxt

```
trigger AccountAddressTrigger on Account (before insert,before update) {
for(Account acc:Trigger.New)
 {
    if(acc.Match_Billing_Address__C){
       acc.ShippingPostalCode = acc.BillingPostalCode;
        System.debug(JSON.serializePretty(acc));
    }
}
}
```

Bulk Apex Triggers:-

https://trailblazer.me/id/karankbhoi

Create a Bulk Apex trigger:

File Name- ClosedOpportunityTrigger.apxt

```
trigger ClosedOpportunityTrigger on Opportunity (before insert) {
   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;
   }
}
```

Module:Apex Testing:-

Get Started with Apex Unit Tests:-

File Name- TestVerifyDate.apxc

```
@isTest
private class TestVerifyDate {

   @isTest static void Test_CheckDates_case1()
   {
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/05/2020'));
      System.assertEquals(date.parse('01/05/2020'),D);
   }
    @isTest static void Test_CheckDates_case2()
   {
      Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('05/05/2020'));
      System.assertEquals(date.parse('01/31/2020'),D);
   }

    @isTest static void Test_DateWithin30Days_case1()
   {
      Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('12/30/2019'));
      System.assertEquals(false,flag);
   }
   @isTest static void Test_DateWithin30Days_case2()
```

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

File Name- VerifyDate.apxc

```
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;
        }
}
```

Test Apex Triggers:-

Create a Unit Test for a Simple Apex Trigger:

File Name- RestrictContactByName.apxc

```
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');
                }

        }



}
```

https://trailblazer.me/id/karankbhoi

File Name- TestRestrictContactByName.apxc

```
@isTest
public class TestRestrictContactByName{

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

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

      System.assert(!result.isSuccess());
      System.assert(result.getErrors().size()>0);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getMessage());
   }
}
```

Create Test Data for Apex Tests:-

Create a Contact Test Factory:

File Name- RandomContactFactory.apxc

```
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;
    }
}
```

https://trailblazer.me/id/karankbhoi

Module:Asynchronous Apex:-

Use Future Methods:-

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

File Name- AccountProcessor.apxc

```
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];
    // process account records to do awesome stuff
    For(Account acc:accounts)
    {
       List<Contact> contactList = acc.Contacts;
       acc.Number_of_Contacts__c = contactList.size();
       accountsToUpdate.add(acc);


    }
     update accountsToUpdate;
  }
}
```

File Name- AccountProcessorTest.apxc

```
@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='John',LastName='Doe',AccountID=newAccount.Id);
     insert newContact2;
```

```
    List<Id>accountIds =new List<Id>();
    accountIds.add(newAccount.Id);

    accountProcessor.countContacts(accountIds);
    Test.startTest();
    accountProcessor.countContacts(accountIds);
  Test.stopTest();
  // runs callout and check results
 }
}
```

Use Batch Apex:-

Create an Apex class that uses Batch Apex to update Lead records:

File Name- LeadProcessor.apxc

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

  global Database.QueryLocator start(Database.BatchableContext bc) {
    return Database.getQueryLocator(
      'SELECT ID from Lead'
    );
  }
  global void execute(Database.BatchableContext bc, List<Lead> scope){
    // process each batch of records
     List<Lead> leads =new List<Lead>();
     for (Lead lead : scope) {
       lead.LeadSource ='Dreamforce';
       leads.add(lead);
      }
    update leads;
  }
  global void finish(Database.BatchableContext bc)
  {

  }
}
```

https://trailblazer.me/id/karankbhoi

File Name- LeadProcessorTest.apxc

```
@isTest
private class LeadProcessorTest {
   @testSetup
   static void setup() {
      List<Lead> leads = new List<Lead>();

      // insert 10 accounts
      for (Integer i=0;i<200;i++) {
         Leads.add(new Lead(Lastname='Lead'+i,Company='Test Co'));
      }
      insert leads;

   }
   @isTest static void test() {
      Test.startTest();
      LeadProcessor myLeads = new LeadProcessor();
      Id batchId = Database.executeBatch(myLeads);
      Test.stopTest();
      // after the testing stops, assert records were updated properly
      System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce']);
   }
}
```

Control Processes with Queueable Apex:-

Create a Queueable Apex class that inserts Contacts for Accounts.

File Name- AddPrimaryContact.apxc

```
public class AddPrimaryContact implements Queueable {
   private Contact con;
   private String state;
   public AddPrimaryContact(Contact con, String state) {
      this.con = con;
      this.state = state;
   }
   public void execute(QueueableContext context) {
    List<Account>accounts =[Select Id,Name,(Select FirstName,LastName,Id from contacts)from
Account where BillingState =:state Limit 200];
```

```
    List<Contact> primaryContacts= new List<Contact>();
    for(Account acc:accounts)
     {
        Contact c = con.clone();
        c.AccountId =acc.Id;
        primaryContacts.add(c);
     }
      if(primaryContacts.size()>0)
      {
         insert primaryContacts;
      }
   }
}
```

File Name- **AddPrimaryContactTest.apxc**

```
@isTest
public class AddPrimaryContactTest {

  static testmethod void testQueueable()
   {
      List<Account> testAccounts =new List<Account>();
      for(Integer i=0;i<50;i++)
      {
         testAccounts.add(new Account(Name='Account'+i,BillingState='CA'));
      }
      for(Integer j=0;j<50;j++)
      {
         testAccounts.add(new Account(Name='Account'+j,BillingState='NY'));
      }
      insert testAccounts;

      Contact testContact =new Contact(FirstName='John',LastName='Doe');
      insert testContact;
      AddPrimaryContact addit =new addPrimaryContact(testContact,'CA');
      Test.startTest();
      System.enqueueJob(addit);
      Test.stopTest();
      System.assertEquals(50,[select count()from Contact Where accountId in(Select Id from
Account where BillingState='CA')]);
   }
```

}

Schedule Jobs Using the Apex Scheduler:-

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

File Name- DailyLeadProcessor.apxc

```
global class DailyLeadProcessor implements Schedulable {

   global void execute(SchedulableContext ctx) {

      //Retrieving the 200 first leads where lead source is in blank.
      List<Lead> leads = [SELECT ID, LeadSource FROM Lead where LeadSource = ''
LIMIT 200];

      //Setting the LeadSource field the 'Dreamforce' value.
      for (Lead lead : leads) {
         lead.LeadSource = 'Dreamforce';
      }

      //Updating all elements in the list.
      update leads;
   }

}
```

File Name- DailyLeadProcessorTest.apxc

```
@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();
    }
}
```

SuperBrade:-

Apex Specialist:-

File Name- MaintenanceRequestHelperTest.apxc

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    public static final string STATUS_NEW = 'New';
    public static final string WORKING = 'Working';
    public static final string CLOSED = 'Closed';
    public static final string REPAIR = 'Repair';
    public static final string REQUEST_ORIGIN = 'Web';
    public static final string REQUEST_TYPE = 'Routine Maintenance';
    public static final string REQUEST_SUBJECT = 'Testing subject';

    public static Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }
```

File Name- MaintenanceRequestHelperHelper.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;
      }
   }
}
```

File Name- MaintenanceRequest.apxt

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

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

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

   }

}
```

==========

File Name- WarehouseCalloutService.apxc

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

https://trailblazer.me/id/karankbhoi

```
    //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();
    }

}
```

File Name- WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    // implement http mock callout
    global static HttpResponse respond(HttpRequest request){

        System.assertEquals('https://th-superbadge-apex.herokuapp.com/equipment',
request.getEndpoint());
        System.assertEquals('GET', request.getMethod());

        // Create a fake response
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quantity":5,"name":
"Generator 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

File Name- WarehouseCalloutServiceTest.apxc

```
@isTest

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

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

==========

File Name- WarehouseSyncSchedule.apxc

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

File Name- 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 ');


    }
}
```