1. Apex Triggers

1. Create an Apex trigger:

- Apex Triggers is useful to perform a custom action before or after events to records in salesforce, such as insertion, updates, etc.

- In this module we build a simple apex trigger that sets an account's Shipping Postal Code to match the Billing Postal Code if the match Billing Address Option is selected.

- Our Trigger name is **AccountAddressTrigger** Account Object. Our Trigger work before insert and before update the record in salesforce.

Code:

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account account : Trigger.new){
        if((account.Match_Billing_Address__c == true) &&
    (account.BillingPostalCode != NULL)){
            account.ShippingPostalCode = account.BillingPostalCode;
        }
    }
}
```

2. Create a Bulk Apex trigger:

- Apex triggers are optimized to operate in bulk.When you use bulk design patterns, your triggers have better performance, consume less server resources, and are less likely to exceed platform limits.

- Here, we create bulkified Apex Trigger with name **ClosedOpportunityTrigger** on the Opportunity object for insert or update 200 or more opportunities.

- We use After updating and insertin. If the stage is closed won than create a task.

Code:

```
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. Create a Unit Test for a Simple Apex Class:

- The Apex testing framework enables you to write and execute tests for your Apex classes and triggers.Apex unit tests ensure high quality for your Apex code and let you meet requirements for deploying Apex.

- Here, we have a **VerifyDate** apex class which is given at GitHub for that we Create a **TestVerifyDate** Apex class to test the **VerifyDate** class.

- In **VerifyDate** class, if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month So that we Check for 2 test one is under 30 days and second is for greater than 30 days between two dates.

VerifyDate Code from GitHub:

```
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;
        }

}
```

Test Class Code:

```
@isTest
public class TestVerifyDate{
        @isTest static void test1(){
  Date d = VerifyDate.CheckDates(Date.parse('11/28/2001'),Date.parse('11/30/2001'));
  System.assertEquals(Date.parse('11/30/2001'), d);
    }
   @isTest static void test2(){
  Date d = VerifyDate.CheckDates(Date.parse('11/28/2001'),Date.parse('12/30/2001'));
  System.assertEquals(Date.parse('11/30/2001'), d);
    }
}
```

2. Create a Unit Test for a Simple Apex Trigger:

- Here is a simple Apex trigger which blocks inserts and updates to any contact with a last name of ‘INVALIDNAME’ Given at a GitHub.

- In Test Class we check for the only one test which is if the last name I “INVALID” than apex tigger gives an error message. For that we create on contact with “INVALID” last name and check weather the trigger give an error or not.

Trigger Code from GitHub:

```
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');
                }

        }
}
```

Test Class Code:

```
@IsTest
public class TestRestrictContactByName {
   @IsTest static void badContact(){
      Contact c = new Contact(FirstName='Lucky',LastName='INVALIDNAME');
      Test.startTest();
      Database.SaveResult result = Database.insert(c, false);
      Test.startTest();
      System.assert(!result.isSuccess());
   }
}
```

3. Create a Contact Test Factory:

- Here, we created one class called **RandomContactFactory** which is contain a **generateRandomContact** method of List Type to generate a contact and add into Contact Object with a unique First Name.

Code:

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

3. Asynchronous Apex

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

- Here, I Created 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 and also write a test class to check a AccountProcessor Class's Code coverage.

Code of Class:

```
public class AccountProcessor {
        @future
   public static void countContacts(List<Id> accountIds){
         List<Account> accList = [Select Id, Number_Of_Contacts__c, (Select Id from Contacts) from Account where Id in :accountIds];

        for(Account acc : accList){
           acc.Number_Of_Contacts__c = acc.Contacts.size();
        }

      update accList;
   }
}
```

Code of Test Class:

```
@isTest
public class AccountProcessorTest {
   public static testmethod void testAccountProcessor(){
      Account a = new Account();
      a.Name = 'Test Account';

      insert a;

      Contact c = new Contact();
```

```
        c.FirstName = 'Binary';
        c.LastName = 'Programming';
        c.AccountId = a.Id;

        insert c;

        List<Id> accListId = new List<Id>();
        accListId.add(a.Id);

        Test.startTest();
        AccountProcessor.countContacts(accListId);
        Test.stopTest();

        Account acc = [Select Number_Of_Contacts__c from Account where Id =: a.Id];
        System.assertEquals(Integer.valueOf(acc.Number_Of_Contacts__c),1);
    }
}
```

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

- I create an Apex class name with a LeadProcessor wich is implements the Database. Batchable interface to update all the Lead records in the Org with a specific LeadSource, here which is ‘Dreamforce’.

Code of Class:

```
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 off Test Class:

```
@isTest
public class LeadProcessorTest {
            @isTest
    public static void test1(){
        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();
    }
}
```

3. Create a Queueable Apex class that inserts Contacts for Accounts:

- Here, I Created 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 and also Created a

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

Code of Class:

```
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;
      }
   }
}
```

Code of Test Class:

```
@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 i=0; i<50; i++){
         testAccounts.add(new Account(Name='Account '+i,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')]);
    }
}
```

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

- I created a Class that implement Schedule interface to update Lead records with a specific LeadSource here Which is ‘Dreamforce’.

Code of Class:

```
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 of Test Class:

```
@isTest
private class DailyLeadProcessorTest{

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();

String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new DailyLeadProcessor());

Test.stopTest();
}
}
```

4. Apex Integration Services

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

- REST callouts are based on HTTP.callout request is associated with an HTTP method and an endpoint. The HTTP method indicates what type of action is desired.The simplest request is a GET request (GET is an HTTP method). Other Methods are Put, Delete and Post.

- In the code First we Created a Class which is Call the REST request then for test we Created a Mock Test method which is used in the Test Class to test our code's Coverage.

Code of 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 (response.getStatusCode() == 200)
     {
                              Map<String, Object> results = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());

        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 ;
    }
}
```

Code of MockTest:

```
@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;
    }
}
```

Code of Test Class:

```
@isTest
private class AnimalLocatorTest{
    @isTest static void AnimalLocatorMock() {
        Test.SetMock(HttpCallOutMock.class, new AnimalLocatorMock());

        string result=AnimalLocator.getAnimalNameById(3);
        string expectedResult='chicken';

        System.assertEquals(result, expectedResult);
    }
}
```

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

- First generate the apex class name ParkService using WSDL2Apex for a SOAP web service. I Created a Park Locator which is give the name of particular Country passed to web service. Here we also Create mock test and test class to ensure the 100% coverage of our class.

Code Of WSDL:

```
//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 of Class:

```
public class ParkLocator {
    public static String[] country(String country){
```

```
        ParkService.ParksImplPort parks = new ParkService.ParksImplPort();

        String[] parksname = parks.byCountry(country);

        return parksname;
    }
}
```

Code Of Mock Test:

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

            String responseType) {
        ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();

        List<String> lstOfDummyParks = new List<String> {'Park1','Park2','Park3'};
        response_x.return_x = lstOfDummyParks;

        response.put('response_x', response_x);
    }
}
```

Code of Test Class:

```
@isTest
private class ParkLocatorTest{
    @isTest
    static void testParkLocator() {
        Test.setMock(WebServiceMock.class, new ParkServiceMock());

        String[] arrayOfParks = ParkLocator.country('India');
```

```
        System.assertEquals('Park1', arrayOfParks[0]);
    }
}
```

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

- Here, I created an apex REST class that will return the account's ID and name plus the ID and name of all contacts associated with the account. I also created a test class to check the codes coverage.

Code of Class:

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {

    @HttpGet
    global static account getAccount() {
        RestRequest request = RestContext.request;

        String accountId =
request.requestURI.substring(request.requestURI.lastIndexOf('/')-18,
request.requestURI.lastIndexOf('/'));

          List<Account> a = [select id, name, (select id, name from contacts) from account
where id = :accountId];

        List<contact> co = [select id, name from contact where account.id = :accountId];

        system.debug('** a[0]= '+ a[0]);

        return a[0];
    }
}
```

Code of Test Class:

```
@Istest(SeeAllData=true)
public class AccountManagerTest {
@isTest static void testGetAccount() {
```

```
    Id recordId = createTestRecord();

    RestRequest request = new RestRequest();

    request.requestUri =
                                                        'https://resourceful-badger-76636-dev-
ed.my.salesforce.com/services/apexrest/Accounts/'+recordId+'/contacts'
        + recordId;

    request.httpMethod = 'GET';
    RestContext.request = request;

    // Call the method to test
    Account thisAcc = AccountManager.getAccount();

    // Verify results
    System.assert(thisAcc != null);
    System.assertEquals('Test record', thisAcc.Name);
}

// Helper method
static Id createTestRecord() {
    // Create test record
   Account accTest = new Account(Name=’Test record’);

    insert accTest;

    return accTest.Id;
    }

}
```

5. Apex Specialist

1. Automate record creation:

- Created a Trigger for before and after update. if the case is closed and it's in routine maintenance or repair than add ID.When an existing maintenance request of type Repair or Routine Maintenance is closed, create a new maintenance request for a future routine check-up.

- If multiple pieces of equipment are used in the maintenance request,define the due date by applying the shortest maintenance cycle to today's date.

Trigger Code:

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

Code of application logic:

```
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()){
            Map<Id,Case> closedCases = new Map<Id,Case>([SELECT Id, Vehicle__c, Product__c, Product.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r) FROM Case WHERE Id IN :validIds])
```

```
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();

        AggregateResult[] results = [SELECT Maintenance_Request__c,
                                     MIN(Equipment__r.Maintenance_Cycle__c)cycle
                                     FROM Equipment_Maintenance_Item__c
                                     WHERE Maintenance_Request__c IN :ValidIds GROUP
    BY Maintenance_Request__c];

        for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
    ar.get('cycle'));
        }

        List<Case> newCases = new List<Case>();

        for(Case cc : closedCases.values()){
          Case nc = new Case (
            ParentId = cc.Id,
            Status = 'New',
            Subject = 'Routine Maintenance',
            Type = 'Routine Maintenance',
            Vehicle__c = cc.Vehicle__c,
            Product__c =cc.Product__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.Product.Maintenance_Cycle__c);
          }

          newCases.add(nc);
        }

        insert newCases;

                                List<Equipment_Maintenance_Item__c> clonedList = new
    List<Equipment_Maintenance_Item__c>();
```

```
        for (Case nc : newCases){
                                for (Equipment_Maintenance_Item__c clonedListItem : closedCases.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c item = clonedListItem.clone();
                item.Maintenance_Request__c = nc.Id;
                clonedList.add(item);
            }
        }
        insert clonedList;
    }
  }
}
```

2. Synchronize Salesforce data with an external system:

- Here, I Write a 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.

Code:

```
public with sharing class WarehouseCalloutService implements Queueable {
        private static final String WAREHOUSE_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';
    @Future(callout=true)
     public static void runWarehouseEquipmentSync(){
     System.debug('go into runWarehouseEquipmentSync');

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

     request.setEndpoint(WAREHOUSE_URL);
     request.setMethod('GET');

     HttpResponse response = http.send(request);

     List<Product2> product2List = new List<Product2>();
     System.debug(response.getStatusCode());
     if (response.getStatusCode() == 200){
```

```
        List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());

            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;

                Product2 product2 = new Product2();

                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');

                product2.Cost__c = (Integer) mapJson.get('cost');

                product2.Current_Inventory__c = (Double) mapJson.get('quantity');

                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');

            product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');

                product2.Warehouse_SKU__c = (String) mapJson.get('sku');

                product2.Name = (String) mapJson.get('name');

                product2.ProductCode = (String) mapJson.get('_id');

                product2List.add(product2);
            }

            if (product2List.size() > 0){
                upsert product2List;
                System.debug('Your equipment was synced with the warehouse one');
            }
        }
    }

    public static void execute (QueueableContext context){
        System.debug('start runWarehouseEquipmentSync');
        runWarehouseEquipmentSync();
        System.debug('end runWarehouseEquipmentSync');
    }
}
```

3. **Schedule synchronization:**

- Using the Schedule create a job which is executed and call class WarehouseCalloutServices.

Code:

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

4. **Test automation logic:**

- Create the class that test the MaintenanceRequest and MaintenanceRequesthelper class.

Code:

```
@isTest
public with sharing class MaintenanceRequestHelperTest {

    private static Vehicle__c createVehicle(){
        Vehicle__c vehicle = new Vehicle__C(name = 'Testing Vehicle');
        return vehicle;
    }

    private static Product2 createEquipment(){
        product2 equipment = new product2(name = 'Testing equipment',
                                          lifespan_months__c = 10,
                                          maintenance_cycle__c = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    private static Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cse = new case(Type='Repair',
                            Status='New',
                            Origin='Web',
```

```
                    Subject='Testing subject',
                    Product__c=equipmentId,
                    Vehicle__c=vehicleId);
        return cse;
    }

     private static Equipment_Maintenance_Item__c createEquipmentMaintenanceItem(id equipmentId,id requestId){
                    Equipment_Maintenance_Item__c equipmentMaintenanceItem = new Equipment_Maintenance_Item__c(
            Equipment__c = equipmentId,
            Maintenance_Request__c = requestId);
        return equipmentMaintenanceItem;
    }

    @isTest
    private static void testPositive(){
        Vehicle__c vehicle = createVehicle();
        insert vehicle;
        id vehicleId = vehicle.Id;

        Product2 equipment = createEquipment();
        insert equipment;
        id equipmentId = equipment.Id;

        case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
        insert createdCase;

                        Equipment_Maintenance_Item__c equipmentMaintenanceItem = createEquipmentMaintenanceItem(equipmentId,createdCase.id);
        insert equipmentMaintenanceItem;

        test.startTest();
        createdCase.status = 'Closed';
        update createdCase;
        test.stopTest();

        Case newCase = [Select id,
                    subject,
                    type,
                    Product__c,
```

```
                Date_Reported__c,
                Vehicle__c,
                Date_Due__c
               from case
               where status ='New'];

    Equipment_Maintenance_Item__c workPart = [select id
                                from Equipment_Maintenance_Item__c
                                where Maintenance_Request__c =:newCase.Id];
    list<case> allCase = [select id from case];

    system.assert(allCase.size() == 2);
    system.assert(newCase != null);
    system.assert(newCase.Subject != null);
    system.assertEquals(newCase.Type, 'Routine Maintenance');
    SYSTEM.assertEquals(newCase.Product__c, equipmentId);
    SYSTEM.assertEquals(newCase.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newCase.Date_Reported__c, system.today());
  }

  @isTest
  private static void testNegative(){
    Vehicle__C vehicle = createVehicle();
    insert vehicle;
    id vehicleId = vehicle.Id;

    product2 equipment = createEquipment();
    insert equipment;
    id equipmentId = equipment.Id;

    case createdCase = createMaintenanceRequest(vehicleId,equipmentId);
    insert createdCase;

                                    Equipment_Maintenance_Item__c    workP    =
createEquipmentMaintenanceItem(equipmentId, createdCase.Id);
    insert workP;

    test.startTest();
    createdCase.Status = 'Working';
    update createdCase;
    test.stopTest();
```

```
        list<case> allCase = [select id from case];

        Equipment_Maintenance_Item__c equipmentMaintenanceItem = [select id
                                        from Equipment_Maintenance_Item__c
                                        where Maintenance_Request__c = :createdCase.Id];

        system.assert(equipmentMaintenanceItem != null);
        system.assert(allCase.size() == 1);
    }

    @isTest
    private static void testBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
            list<Equipment_Maintenance_Item__c> equipmentMaintenanceItemList = new
list<Equipment_Maintenance_Item__c>();
        list<case> caseList = new list<case>();
        list<id> oldCaseIds = new list<id>();

        for(integer i = 0; i < 300; i++){
            vehicleList.add(createVehicle());
            equipmentList.add(createEquipment());
        }
        insert vehicleList;
        insert equipmentList;

        for(integer i = 0; i < 300; i++){
                                    caseList.add(createMaintenanceRequest(vehicleList.get(i).id,
equipmentList.get(i).id));
        }
        insert caseList;

        for(integer i = 0; i < 300; i++){

equipmentMaintenanceItemList.add(createEquipmentMaintenanceItem(equipmentList.ge
t(i).id, caseList.get(i).id));
        }
        insert equipmentMaintenanceItemList;

        test.startTest();
```

```
        for(case cs : caseList){
            cs.Status = 'Closed';
            oldCaseIds.add(cs.Id);
        }
        update caseList;
        test.stopTest();

        list<case> newCase = [select id
                              from case
                              where status ='New'];

        list<Equipment_Maintenance_Item__c> workParts = [select id
                                                         from Equipment_Maintenance_Item__c
                                                         where Maintenance_Request__c in: oldCaseIds];

        system.assert(newCase.size() == 300);

        list<case> allCase = [select id from case];
        system.assert(allCase.size() == 600);
    }
}
```

5. **Test callout logic:**

- Create a mock class to callout and also create the test class to the check the mock callout output.

  Mock Callout Code:

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request) {
        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"},{"_id":"55d6
6226726b611100aaf742","replacement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"100004"},{"_id":"55d662267
26b611100aaf743","replacement":true,"quantity":143,"name":"Fuse
```

```
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"100005"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Test Class Code:

```
@IsTest
private class WarehouseCalloutServiceTest {
        @isTest
        static void testWarehouseCallout() {
        test.startTest();
        test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        WarehouseCalloutService.execute(null);
        test.stopTest();

        List<Product2> product2List = new List<Product2>();
        product2List = [SELECT ProductCode FROM Product2];

        System.assertEquals(3, product2List.size());
                                        System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
                                        System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
                                        System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
    }
}
```

6. Test scheduling logic:

- Use the callout service mock as a mock callout and create the Test class to test the schedule of the WarehouseSyncSchedule Class.

Code:

```
@isTest
public with sharing class WarehouseSyncScheduleTest {
    @isTest static void test() {
        String scheduleTime = '00 00 00 * * ? *';
```

```
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());

        String jobId = System.schedule('Warehouse Time to Schedule to test', scheduleTime,
new WarehouseSyncSchedule());

        CronTrigger c = [SELECT State FROM CronTrigger WHERE Id =: jobId];
        System.assertEquals('WAITING', String.valueOf(c.State), 'JobId does not match');

        Test.stopTest();
    }
}
```