GET STARTED WITH APEX TRIGGERS:

```
trigger AccountAddressTrigger on Account (before insert, before update) {
    for(Account a:Trigger.New){
        if(a.Match_Billing_Address__c == True){
            a.ShippingPostalCode = a.BillingPostalCode;
        }
    }

}
```

BULK APEX TRIGGERS:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> tasklist = new List<Task>();
    for(Opportunity opp: Trigger.New){
        if(opp.StageName == 'Closed Won'){
          tasklist.add(new Task(Subject = 'Follow

        }
    }
    if(tasklist.size()>0){
        insert tasklist;
    }

}
```

APPEX TESTING

GET STARTED WITH APEX UNIT TEST:

```
public class VerifyDate {

  //method to handle potential checks against two
```

```
  dates
4    public static Date CheckDates(Date date1, Date
  date2) {
5        //if date2 is within the next 30 days of
  date1, use date2.  Otherwise use the end of the
  month
6        if(DateWithin30Days(date1,date2)) {
7            return date2;
8        } else {
9            return SetEndOfMonthDate(date1);
10       }
11   }
12
13   //method to check if date2 is within the next 30
  days of date1
14   @TestVisible      private      static      Boolean
  DateWithin30Days(Date date1, Date date2) {
15       //check for date2 being in the past
16   if( date2 < date1) { return false; }
17
18   //check that date2 is within (>=) 30 days of
  date1
19   Date date30Days = date1.addDays(30); //create a
  date 30 days away from date1
20       if( date2 >= date30Days ) { return false; }
21       else { return true; }
22   }
23
24   //method to return the end of the month of a
  given date
25   @TestVisible      private      static      Date
  SetEndOfMonthDate(Date date1) {
26       Integer              totalDays              =
  Date.daysInMonth(date1.year(), date1.month());
```

```
27      Date                    lastDay                 =
  Date.newInstance(date1.year(),      date1.month(),
  totalDays);
28      return lastDay;
29  }
30
31}
```

Test verify date:

```
1 @isTest
2 public class TestVerifyDate {
3
4     @isTest static void Test_CheckDates_case1(){
5                                                   Date
  D=VerifyDate.CheckDates(date.parse('01/01/2020'),d

6
  System.assertEquals(date.parse('01/05/2020'),D);
7     }
8
9     @isTest static void Test_CheckDates_case2(){
10                                                  Date
  D=VerifyDate.CheckDates(date.parse('01/01/2020'),d

11
  System.assertEquals(date.parse('01/31/2020'),D);
12    }
13
14                      @isTest     static      void
  Test_DateWithin30Days_case1(){
15                                  Boolean    flag    =
  VerifyDate.DateWithin30Days(date.parse('01/01/2020
```

```
16        System.assertEquals(false, flag);
17
18    }
19                        @isTest      static      void
  Test_DateWithin30Days_case2(){
20                                  Boolean    flag    =
  VerifyDate.DateWithin30Days(date.parse('01/01/2020

21        System.assertEquals(false, flag);
22
23    }
24                        @isTest      static      void
  Test_DateWithin30Days_case3(){
25                                  Boolean    flag    =
  VerifyDate.DateWithin30Days(date.parse('01/01/2020

26        System.assertEquals(true, flag);
27
28    }
29    @isTest static void Test_SetEndOfMonthDate(){
30                                                  Date
  returndate=VerifyDate.SetEndOfMonthDate(date.parse
  ('01/01/2020'));
31    }
32
33
34}
```

TEST APEX TRIGGERS:

RestrictContactByName.apxt:

```
1 trigger RestrictContactByName on Contact (before
  insert, before update) {
2
3   //check contacts prior to insert or update for
```

```
  invalid data
4    For (Contact c : Trigger.New) {
5        if(c.LastName == 'INVALIDNAME') {
  //invalidname is invalid
6            c.AddError('The Last Name
  ed for DML');
7        }
8
9    }
10}
```

CREATE TEST DATA FOR APEX TESTS:

RandomContactFactory.apxc:

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

ASYNCHRONOUS APEX

USE FUTURE METHODS:

```
1 public class AccountProcessor {
2     @future
3     public static void countContacts(List<Id>
  accountIds){
4         List<Account> accounts = [Select Id, Name
```

```
   from Account Where Id IN : accountIds];
5          List<Account> updatedAccounts = new
   List<Account>();
6          for(Account account : accounts){
7             account.Number_of_Contacts__c = [Select
   count() from Contact Where AccountId =:
   account.Id];
8              System.debug('No Of Contacts = ' +
   account.Number_of_Contacts__c);
9              updatedAccounts.add(account);
10         }
11         update updatedAccounts;
12     }
13
14}
```

```
1 @isTest
2 public class AccountProcessorTest {
3     @isTest
4     public static void testNoOfContacts(){
5         Account a = new Account();
6         a.Name = 'Test Account';
7         Insert a;
8
9         Contact c = new Contact();
10        c.FirstName = 'Bob';
11        c.LastName =  'Willie';
12        c.AccountId = a.Id;
13
14        Contact c2 = new Contact();
15        c2.FirstName = 'Tom';
16        c2.LastName = 'Cruise';
```

```
        c2.AccountId = a.Id;

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

        Test.startTest();
        AccountProcessor.countContacts(acctIds);
        Test.stopTest();
    }

}
```

USE BATCH APEX:

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

     public Database.QueryLocator start(Database.BatchableContext bc) {
        // collect the batches of records or objects to be passed to execute
          return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
    public void execute(Database.BatchableContext bc, List<Lead> leads){
         // process each batch of records
            for (Lead Lead : leads) {
                lead.LeadSource = 'Dreamforce';
            }
        update leads;
    }
    public void finish(Database.BatchableContext bc){
     }
```

```
16
17}
```

Test:

```
1 @isTest
2 public class LeadProcessorTest {
3
4         @testSetup
5     static void setup() {
6         List<Lead> leads = new List<Lead>();
7         for(Integer counter=0 ;counter
  <200;counter++){
8             Lead lead = new Lead();
9             lead.FirstName ='FirstName';
10             lead.LastName ='LastName'+counter;
11             lead.Company ='demo'+counter;
12             leads.add(lead);
13         }
14         insert leads;
15     }
16
17     @isTest static void test() {
18         Test.startTest();
19         LeadProcessor leadProcessor = new
  LeadProcessor();
20         Id batchId =
  Database.executeBatch(leadProcessor);
21         Test.stopTest();
22     }
23
24}
```

CONTROL PROCESSES WITH QUEUEABLE APEX:

```
1 public class AddPrimaryContact implements
```

```
  Queueable
{
    private Contact c;
    private String state;
    public  AddPrimaryContact(Contact c, String
  state)
    {
        this.c = c;
        this.state = state;
    }
    public void execute(QueueableContext context)
    {
         List<Account> ListAccount = [SELECT ID,
  Name ,(Select id,FirstName,LastName from contacts
  ) FROM ACCOUNT WHERE BillingState = :state LIMIT
  200];
         List<Contact> lstContact = new
  List<Contact>();
         for (Account acc:ListAccount)
         {
                 Contact cont =
  c.clone(false,false,false,false);
                 cont.AccountId =  acc.id;
                 lstContact.add( cont );
         }

         if(lstContact.size() >0 )
         {
             insert lstContact;
         }

    }
```

```
@isTest
public class AddPrimaryContactTest
{
      @isTest static void TestList()
      {
          List<Account> Teste = new List
  <Account>();
          for(Integer i=0;i<50;i++)
          {
              Teste.add(new Account(BillingState =
  'CA', name = 'Test'+i));
          }
          for(Integer j=0;j<50;j++)
          {
              Teste.add(new Account(BillingState =
  'NY', name = 'Test'+j));
          }
          insert Teste;

          Contact co = new Contact();
          co.FirstName='demo';
          co.LastName ='demo';
          insert co;
          String state = 'CA';

           AddPrimaryContact apc = new
  AddPrimaryContact(co, state);
           Test.startTest();
             System.enqueueJob(apc);
           Test.stopTest();
      }
}
```

SCHEDULE JOBS USING APEX SCHEDULER:

```
1 public class DailyLeadProcessor implements
  Schedulable  {
2     Public void execute(SchedulableContext SC){
3        List<Lead> LeadObj=[SELECT Id from Lead
  where LeadSource=null limit 200];
4         for(Lead l:LeadObj){
5             l.LeadSource='Dreamforce';
6             update l;
7         }
8     }
9 }
```

```
1    @isTest
2 private class DailyLeadProcessorTest {
3    static testMethod void testDailyLeadProcessor()
  {
4        String CRON_EXP = '0 0 1 * * ?';
5        List<Lead> lList = new List<Lead>();
6         for (Integer i = 0; i < 200; i++) {
7            lList.add(new
  Lead(LastName='Dreamforce'+i, Company='Test1

8        }
9        insert lList;
10
11       Test.startTest();
12       String jobId =
  System.schedule('DailyLeadProcessor', CRON_EXP,
  new DailyLeadProcessor());
13   }
14}
```

APEX INTEGRATION SERVICES

APEX REST CALLOUTS:

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

        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');
    }
}
```

```
@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
```

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

```
@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
);
    }
}
```

APEX SOAP CALLOUTS:

```
//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

        private String[] apex_schema_type_info =
new
String[]{'http://parks.services/','false','false'
};
```

```
8           private String[] field_order_type_info =
  new String[]{'return_x'};
9       }
10      public class byCountry {
11          public String arg0;
12          private String[] arg0_type_info = new
  String[]{'arg0','http://parks.services/',null,'0',
  '1','false'};
13          private String[] apex_schema_type_info =
  new
  String[]{'http://parks.services/','false','false'
  };
14          private String[] field_order_type_info =
  new String[]{'arg0'};
15      }
16      public class ParksImplPort {
17          public String endpoint_x = 'https://th-

18          public Map<String,String>
  inputHttpHeaders_x;
19          public Map<String,String>
  outputHttpHeaders_x;
20          public String clientCertName_x;
21          public String clientCert_x;
22          public String clientCertPasswd_x;
23          public Integer timeout_x;
24          private String[] ns_map_type_info = new
  String[]{'http://parks.services/', 'ParkService'};
25          public String[] byCountry(String arg0) {
26              ParkService.byCountry request_x = new
  ParkService.byCountry();
27              request_x.arg0 = arg0;
28              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;
        }
    }
}
```

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

```
  List<String>{'Yellowstone', 'Mackinac National

8           System.assertEquals(parks, result);
9       }
10}
```

APEX WEB SERVICES:

```
1 @isTest
2 private class AccountManagerTest {
3
4       private static testMethod void
  getAccountTest1() {
5           Id recordId = createTestRecord();
6           // Set up a test request
7           RestRequest request = new RestRequest();
8           request.requestUri =
  'https://na1.salesforce.com/services/apexrest/Acco

9           request.httpMethod = 'GET';
10          RestContext.request = request;
11          // Call the method to test
12          Account thisAccount =
  AccountManager.getAccount();
13          // Verify results
14          System.assert(thisAccount != null);
15          System.assertEquals('Test record',
  thisAccount.Name);
16
17      }
18
19      // Helper method
20          static Id createTestRecord() {
21          // Create test record
```

```
        Account TestAcc = new Account(
          Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}
```

```
@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;
    }
}
```

APEX SPECIALIST SUPERBADGE
*AUTOMATE RECORD CREATION:

```
public with sharing class MaintenanceRequestHelper
  {
public static void updateWorkOrders(List<Case>
  caseList) {
```

```
3 List<case> newCases = new List<Case>();
4 Map<String,Integer> result=getDueDate(caseList);
5 for(Case c : caseList){
6 if(c.status=='closed')
7 if(c.type=='Repair' || c.type=='Routine

8 Case newCase = new Case();
9 newCase.Status='New';
10newCase.Origin='web';
11newCase.Type='Routine Maintenance';
12newCase.Subject='Routine Maintenance of Vehicle';
13newCase.Vehicle__c=c.Vehicle__c;
14newCase.Equipment__c=c.Equipment__c;
15newCase.Date_Reported__c=Date.today();
16if(result.get(c.Id)!=null)
17newCase.Date_Due__c=Date.today()+result.get(c.Id);
18else
19newCase.Date_Due__c=Date.today();
20newCases.add(newCase);
21}
22}
23insert newCases;
24}
25//
26public static  Map<String,Integer>
  getDueDate(List<case> CaseIDs){
27Map<String,Integer> result = new
  Map<String,Integer>();
28Map<Id, case> caseKeys = new Map<Id, case>
  (CaseIDs);
29List<AggregateResult> wpc=[select
  Maintenance_Request__r.ID
  cID,min(Equipment__r.Maintenance_Cycle__c)cycle
30from Work_Part__c where  Maintenance_Request__r.ID
```

```
  in :caseKeys.keySet() group by
  Maintenance_Request__r.ID ];
31for(AggregateResult res :wpc){
32Integer addDays=0;
33if(res.get('cycle')!=null)
34addDays+=Integer.valueOf(res.get('cycle'));
35result.put((String)res.get('cID'),addDays);
36}
37return result;
38}
39}
```

```
1 trigger MaintenanceRequest on Case (before update,
  after update) {
2 // ToDo: Call
  MaintenanceRequestHelper.updateWorkOrders
3 if(Trigger.isAfter)
4 MaintenanceRequestHelper.updateWorkOrders(Trigger.
  New);
5 }
```

SYNCHRONIZATION SALESFORCE DATA WITH AN EXTERNAL SYSTEM:

```
1 public with sharing class WarehouseCalloutService
  {
2 private static final String WAREHOUSE_URL =
  'https://th-superbadge-

3 @future(callout=true)
4 public static void runWarehouseEquipmentSync() {
5 //ToDo: complete this method to make the callout
  (using @future) to the
6 //      REST endpoint and update equipment on
```

```
  hand.
7 HttpResponse response = getResponse();
8 if(response.getStatusCode() == 200)
9 {
10List<Product2> results = getProductList(response);
  //get list of products from Http callout response
11if(results.size() >0)
12upsert results Warehouse_SKU__c; //Upsert the
  products in your org based on the external ID SKU
13}
14}
15//Get the product list from the external link
16public static List<Product2>
  getProductList(HttpResponse response)
17{
18List<Object> externalProducts = (List<Object>)
  JSON.deserializeUntyped(response.getBody());
  //desrialize the json response
19List<Product2> newProducts = new List<Product2>();
20for(Object p : externalProducts)
21{
22Map<String, Object> productMap = (Map<String,
  Object>) p;
23Product2 pr = new Product2();
24//Map the fields in the response to the
  appropriate fields in the Equipment object
25pr.Replacement_Part__c =
  (Boolean)productMap.get('replacement');
26pr.Cost__c = (Integer)productMap.get('cost');
27pr.Current_Inventory__c =
  (Integer)productMap.get('quantity');
28pr.Lifespan_Months__c =
  (Integer)productMap.get('lifespan') ;
29pr.Maintenance_Cycle__c =
```

```
  (Integer)productMap.get('maintenanceperiod');
pr.Warehouse_SKU__c =
  (String)productMap.get('sku');
pr.ProductCode = (String)productMap.get('_id');
pr.Name = (String)productMap.get('name');
newProducts.add(pr);
}
return newProducts;
}
// Send Http GET request and receive Http response
public static HttpResponse getResponse() {
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
HttpResponse response = http.send(request);
return response;
}
}
```

SCHEDULE SYNCHRONIZATION USING APEX CODE:

```
global  class WarehouseSyncSchedule implements
Schedulable{
// implement scheduled code here
global  void execute (SchedulableContext sc){
WarehouseCalloutService.runWarehouseEquipmentSync(
);
//optional this can be done by debug mode
String sch = '00 00 01 * * ?';//on 1 pm
System.schedule('WarehouseSyncScheduleTest', sch,
new WarehouseSyncSchedule());
}
}
```

TEST AUTOMATION LOGIC:

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

```
@IsTest
private class InstallationTests {
private static final String STRING_TEST = 'TEST';
private static final String NEW_STATUS = '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 = 'AMC

public static String CRON_EXP = '0 0 1 * * ?';
static testmethod void testMaintenanceRequestNegative() {
Vehicle__c vehicle = createVehicle();
insert vehicle;
Id vehicleId = vehicle.Id;
Product2 equipment = createEquipment();
insert equipment;
Id equipmentId = equipment.Id;
Case r = createMaintenanceRequest(vehicleId, equipmentId);
insert r;
```

```
Work_Part__c w = createWorkPart(equipmentId,
  r.Id);
insert w;
Test.startTest();
r.Status = WORKING;
update r;
Test.stopTest();
List<case> allRequest = [SELECT Id
FROM Case];
Work_Part__c workPart = [SELECT Id
FROM Work_Part__c
WHERE Maintenance_Request__c =: r.Id];
System.assert(workPart != null);
System.assert(allRequest.size() == 1);
}
static testmethod void testWarehouseSync() {
Test.setMock(HttpCalloutMock.class, new
  WarehouseCalloutServiceMock());
Test.startTest();
String jobId =
  System.schedule('WarehouseSyncSchedule',
CRON_EXP,
new WarehouseSyncSchedule());
CronTrigger ct = [SELECT Id, CronExpression,
  TimesTriggered, NextFireTime
FROM CronTrigger
WHERE id = :jobId];
System.assertEquals(CRON_EXP, ct.CronExpression);
System.assertEquals(0, ct.TimesTriggered);
Test.stopTest();
}
private static Vehicle__c createVehicle() {
Vehicle__c v = new Vehicle__c(Name = STRING_TEST);
```

```
return v;
}
private static Product2 createEquipment() {
Product2 p = new Product2(Name = STRING_TEST,
Lifespan_Months__c = 10,
Maintenance_Cycle__c = 10,
Replacement_Part__c = true);
return p;
}
private static Case createMaintenanceRequest(Id
  vehicleId, Id equipmentId) {
Case c = new Case(Type = REPAIR,
Status = NEW_STATUS,
Origin = REQUEST_ORIGIN,
Subject = REQUEST_SUBJECT,
Equipment__c = equipmentId,
Vehicle__c = vehicleId);
return c;
}
private static Work_Part__c createWorkPart(Id
  equipmentId, Id requestId) {
Work_Part__c wp = new Work_Part__c(Equipment__c =
  equipmentId,
Maintenance_Request__c = requestId);
return wp;
}
}
```

```
public with sharing class MaintenanceRequestHelper
  {
public static void updateWorkOrders(List<case>
  caseList) {
List<case> newCases = new List<case>();
```

```
4 Map<String,Integer> result=getDueDate(caseList);
5 for(Case c : caseList){
6 if(c.status=='closed')
7 if(c.type=='Repair' || c.type=='Routine

8 Case newCase = new Case();
9 newCase.Status='New';
10newCase.Origin='web';
11newCase.Type='Routine Maintenance';
12newCase.Subject='Routine Maintenance of Vehicle';
13newCase.Vehicle__c=c.Vehicle__c;
14newCase.Equipment__c=c.Equipment__c;
15newCase.Date_Reported__c=Date.today();
16if(result.get(c.Id)!=null)
17newCase.Date_Due__c=Date.today()+result.get(c.Id);
18else
19newCase.Date_Due__c=Date.today();
20newCases.add(newCase);
21}
22}
23insert newCases;
24}
25//
26public static  Map<String,Integer>
  getDueDate(List<case> CaseIDs){
27Map<String,Integer> result = new
  Map<String,Integer>();
28Map<Id, case> caseKeys = new Map<Id, case>
  (CaseIDs);
29List<aggregateresult> wpc=[select
  Maintenance_Request__r.ID
  cID,min(Equipment__r.Maintenance_Cycle__c)cycle
30from Work_Part__c where  Maintenance_Request__r.ID
  in :caseKeys.keySet() group by
```

```
  Maintenance_Request__r.ID ];
for(AggregateResult res :wpc){
Integer addDays=0;
if(res.get('cycle')!=null)
addDays+=Integer.valueOf(res.get('cycle'));
result.put((String)res.get('cID'),addDays);
}
return result;
}
}
```

```
@isTest
public  class MaintenanceRequestTest {
static  List<case> caseList1 = new List<case>();
static List<product2> prodList = new
  List<product2>();
static List<work_part__c> wpList = new
  List<work_part__c>();
@testSetup
static void getData(){
caseList1= CreateData( 300,3,3,'Repair');
}
public static List<case>   CreateData( Integer
  numOfcase, Integer numofProd, Integer
  numofVehicle,
String type){
List<case> caseList = new List<case>();
//Create Vehicle
Vehicle__c vc = new Vehicle__c();
vc.name='Test Vehicle';
upsert vc;
//Create Equiment
for(Integer i=0;i<numofProd;i++){
```

```
Product2 prod = new Product2();
prod.Name='Test Product'+i;
if(i!=0)
prod.Maintenance_Cycle__c=i;
prod.Replacement_Part__c=true;
prodList.add(prod);
}
upsert  prodlist;
//Create Case
for(Integer i=0;i< numOfcase;i++){
Case newCase = new Case();
newCase.Status='New';
newCase.Origin='web';
if( math.mod(i, 2) ==0)
newCase.Type='Routine Maintenance';
else
newCase.Type='Repair';
newCase.Subject='Routine Maintenance of Vehicle'
  +i;
newCase.Vehicle__c=vc.Id;
if(i<numofProd)
newCase.Equipment__c=prodList.get(i).ID;
else
newCase.Equipment__c=prodList.get(0).ID;
caseList.add(newCase);
}
upsert caseList;
for(Integer i=0;i<numofProd;i++){
Work_Part__c wp = new Work_Part__c();
wp.Equipment__c   =prodlist.get(i).Id   ;
wp.Maintenance_Request__c=caseList.get(i).id;
wplist.add(wp) ;
}
```

```
51upsert wplist;
52return caseList;
53}
54public static testmethod void
  testMaintenanceHelper(){
55Test.startTest();
56getData();
57for(Case cas: caseList1)
58cas.Status ='Closed';
59update caseList1;
60Test.stopTest();
61}
62}
```

Test callout logic:

```
1 @IsTest
2 private class WarehouseCalloutServiceTest {
3 // implement your mock callout test here
4 @isTest
5 static void testWareHouseCallout(){
6 Test.setMock(HttpCalloutMock.class, new
  WarehouseCalloutServiceMock());
7 WarehouseCalloutService.runWarehouseEquipmentSync(
  );
8 }
9 }
```

```
1 @isTest
2 public class WarehouseCalloutServiceMock
  implements HTTPCalloutMock {
3 // implement http mock callout
```

```
4 public HTTPResponse respond (HttpRequest request){
5 HttpResponse response = new HTTPResponse();
6 response.setHeader('Content-

7 response.setBody('[{"_id":"55d66226726b611100aaf74









8 response.setStatusCode(200);
9 return response;
10}
11}
```

Test scheduling Logic:

```
1 @isTest
2 private class WarehouseSyncScheduleTest {
3 public static String CRON_EXP = '0 0 0 15 3 ?

4 static testmethod void testjob(){
5 MaintenanceRequestTest.CreateData(
  5,2,2,'Repair');
6 Test.startTest();
7 Test.setMock(HttpCalloutMock.class, new
  WarehouseCalloutServiceMock());
8 String joBID= System.schedule('TestScheduleJob',
```

```
  CRON_EXP, new WarehouseSyncSchedule());
// List<Case> caselist = [Select count(id) from
  case where case]
Test.stopTest();
}
}
```