1. Apex Triggers:

A. Get Started with Apex Triggers

AccountAddressTrigger:

```
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;
        }
    }
}
```

B. Bulk Apex Triggers

ClosedOpportunityTrigger:

```
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

A. Get Started with Apex Unit Tests

VerifyDate:

```
public class VerifyDate {
        public static Date CheckDates(Date date1, Date date2) {
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return SetEndOfMonthDate(date1);
                }
        }
```

```
        private static Boolean DateWithin30Days(Date date1, Date date2) {
        if( date2 < date1) { return false; }
        Date date30Days = date1.addDays(30);
                if( date2 >= date30Days ) { return false; }
                else { return true; }
        }
        private static Date SetEndOfMonthDate(Date date1) {
                Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
                Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
                return lastDay;
}

}
```

TestVerifyDate:

```
@isTest
public class TestVerifyDate {

   @isTest static void test1(){
      Date d=VerifyDate.CheckDates(Date.parse('01/01/2022'),Date.parse('01/03/2022'));
     System.assertEquals(Date.parse('01/03/2022'),d);
   }

    @isTest static void test2(){
      Date d=VerifyDate.CheckDates(Date.parse('01/01/2022'),Date.parse('03/03/2022'));
      System.assertEquals(Date.parse('01/31/2022'),d);
    }


}
```

B. **Test Apex Triggers**

RestrictContactByName:

```
trigger RestrictContactByName on Contact (before insert, before update) {
        For (Contact c : Trigger.New) {
                if(c.LastName == 'INVALIDNAME') {
                        c.AddError('The Last Name "'+c.LastName+'" is not allowed for DML');
                }
        }
}
```

TestRestrictContactByName:

```
trigger RejectDuplicateFavorite on Favorite__c (before insert) {
    Favorite__c favorite = Trigger.New[0];
    List<Favorite__c> dupes = [Select Id FROM Favorite__C WHERE Property__c =
:favorite.Property__c AND User__c = :favorite.User__c];
    if (!dupes.isEmpty()) {
        favorite.addError('duplicate');
    }

}
```

C. Create Test Data for Apex Tests

RandomContactFactory:

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer num, String lastName){
        List<Contact> contactList = new List<Contact>();
        for(Integer i=1;i<=num;i++){
            Contact ct=new Contact(FirstName = 'Test' +i,LastName=lastName);
            contactList.add(ct);
        }
        return contactList;
    }
}
```

3. Asynchronous Apex

A. Use Future Methods

AccountProcessor:

```
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;
    }
```

```
}
```

AccountProcessorTest:

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

        Contact con=new Contact();
        con.FirstName='Binary';
        con.LastName='Programming';
        con.AccountId=a.Id;

        insert con;

        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);
    }
}
```

B. Use Batch Apex

LeadProcessor:

```
global class LeadProcessor implements Database.Batchable<sObject> {
    global Integer count = 0;
    public 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);
    }
}
```

LeadProcessorTest:

```
@isTest
public class LeadProcessorTest {
    @isTest
    public static void testit(){
        List<lead> L_list=new List<lead>();
        for(Integer i=0;i<200;i++){
            Lead L = new lead();
            L.LastName = 'name'+i;
            L.Company = 'Company';
            L.Status = 'Random Status';
            L_list.add(L);
        }
        insert L_list;
        Test.startTest();
        LeadProcessor lp=new LeadProcessor();
        Id batchId=Database.executeBatch(lp);
        Test.stopTest();
    }
}
```

C. Control Processes with Queueable Apex

AddPrimaryContact:

```
public class AddPrimaryContact implements Queueable {
    public contact c;
    public String state;

    public AddPrimaryContact(Contact c, String state) {
        this.c = c;
        this.state = state;
```

```
    }

    public void execute(QueueableContext qc) {
        system.debug('this.c = '+this.c+' this.state = '+this.state);
        List<Account> acc_lst = new List<account>([select id, name, BillingState from account
where account.BillingState = :this.state limit 200]);
        List<contact> c_lst = new List<contact>();
        for(account a: acc_lst) {
            contact c = new contact();
            c = this.c.clone(false, false, false, false);
            c.AccountId = a.Id;
            c_lst.add(c);
        }
        insert c_lst;
    }

}
```

AddPrimaryContactTest:

```
@IsTest
public class AddPrimaryContactTest {

    @IsTest
    public static void testing() {
        List<account> acc_lst = new List<account>();
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(i),billingstate='NY');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }
        for (Integer i=0; i<50;i++) {
            account a = new account(name=string.valueOf(50+i),billingstate='CA');
            system.debug('account a = '+a);
            acc_lst.add(a);
        }

        insert acc_lst;
        Test.startTest();
        contact c = new contact(lastname='alex');

        AddPrimaryContact apc = new AddPrimaryContact(c,'CA');
```

```
        system.debug('apc = '+apc);
        System.enqueueJob(apc);
        Test.stopTest();

        List<contact> c_lst = new List<contact>([select id from contact]);
        Integer size = c_lst.size();
        system.assertEquals(50, size);

    }

}
```

D. Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor:

```
global class DailyLeadProcessor implements Schedulable {

    global void execute(SchedulableContext context) {
        List<Lead> leads = [Select Id, LeadSource from Lead where LeadSource = null limit 200];
        for (Lead currentLead : leads) {
            currentLead.LeadSource = 'Dreamforce';
        }
        update leads;
    }
}
```

DailyLeadProcessorTest:

```
@isTest
public class DailyLeadProcessorTest {
    static @IsTest
    void testScheduledJob(){
        List<Lead> leads = new List<Lead>();
        for (Integer i = 0; i < 200; i++) {
            leads.add(new Lead(LastName = 'Dreamforce'+i, Company = 'test'+i));
        }
        insert leads;

        Map<Id,Lead> leadMap = new Map<Id,Lead>(leads);
        List<Id> leadsId = new List<Id>(leadMap.keySet());
        Test.startTest();
        System.schedule('DailyLeadProcessor', '20 30 8 10 2 ?', new DailyLeadProcessor());
```

```
        Test.stopTest();
        System.assertEquals(200, [select count() from Lead where LeadSource = 'Dreamforce'
and Id in :leadsId]);
    }
}
```

4. Apex Integration Services

A. Apex REST Callouts

AnimalLocator:

```
public class AnimalLocator {
        public class cls_animal {
                public Integer id;
                public String name;
                public String eats;
                public String says;
        }
public class JSONOutput{
        public cls_animal animal;
}

    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);
        system.debug('response: ' + response.getBody());

        jsonOutput results = (jsonOutput) JSON.deserialize(response.getBody(), jsonOutput.class);

                system.debug('results= ' + results.animal.name);
        return(results.animal.name);
    }

}
```

AnimalLocatorTest:

```
@IsTest
public class AnimalLocatorTest {
    @isTest
    public static void testAnimalLocator() {
        Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
        String s =  AnimalLocator.getAnimalNameById(1);
        system.debug('string returned: ' + s);
    }

}
```

B. Apex SOAP Callouts

ParkService:

```
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;
    }
  }
}
```

ParkLocator:

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

}
```

ParkLocatorTest:

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

```
        parks.add('Another Park');
        System.assertEquals(parks,result);
    }
}
```

C. Apex Web Services

AccountManager:

```
 @RestResource(urlMapping='/Accounts/*/contacts')
global class AccountManager {
    @HttpGet
    global static Account getAccount() {
        RestRequest req = RestContext.request;
        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');
        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)
                       FROM Account WHERE Id = :accId];
                       return acc;
    }
}
```

AccountManagerTest:

```
@isTest
private class AccountManagerTest
{
    @isTest static void testGetAccount ()
    {
        Id recordId = createTestRecord ();
        RestRequest request = new RestRequest ();
        request.requestUri = 'https://yourInstance.salesforce.com/services/apexrest/Accounts/' +
recordId + '/contacts';
        request.httpMethod = 'GET';
        RestContext.request = request;
        Account thisAccount = AccountManager.getAccount();
        System.assert (thisAccount != null);
        System.assertEquals ('Test Record', thisAccount.Name);
}
    static Id createTestRecord ()
    {
        Account testAccount = new Account (Name = 'Test Record');
        insert testAccount;
        Contact testContact = new Contact (AccountId = testAccount.Id);
```

```
        return testAccount.Id;
    }

}
```

4. Apex Specialist

A. Automate record creation

MaintenanceRequestHelper.apxc:

```
public with sharing class MaintenanceRequestHelper {
    public static void updateworkOrders(List<Case> updWorkOrders, Map<Id,Case>
nonUpdCaseMap) {
        Set<Id> validIds = new Set<Id>();
        For (Case c : updWorkOrders){
            if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
                if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                    validIds.add(c.Id);
                }
            }
        }
              if (!validIds.isEmpty()){
            List<Case> newCases = new List<Case>();
            Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT Id,Equipment__c,Quantity__c
FROM Equipment_Maintenance_Items__r)
                                                  FROM Case WHERE Id IN :validIds]);
            Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
            AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM Equipment_Maintenance_Item__c
WHERE Maintenance_Request__c IN :ValidIds GROUP BY Maintenance_Request__c];

        for (AggregateResult ar : results){
            maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal) ar.get('cycle'));
        }
                    for(Case cc : closedCasesM.values()){
                Case nc = new Case (
                    ParentId = cc.Id,
                Status = 'New',
                    Subject = 'Routine Maintenance',
                    Type = 'Routine Maintenance',
```

```
                Vehicle__c = cc.Vehicle__c,
                Equipment__c =cc.Equipment__c,
                Origin = 'Web',
                Date_Reported__c = Date.Today()

            );

            If (maintenanceCycles.containskey(cc.Id)){
                nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
            }
                newCases.add(nc);
        }
          insert newCases;
              List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);
            }
        }
        insert ClonedWPs;
    }
  }
}
```

MaintenanceRequest.apxt

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

B. Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {

  private static final String WAREHOUSE_URL = 'https://th-superbadge-
```

```
apex.herokuapp.com/equipment';
    public static void runWarehouseEquipmentSync(){

        Http http = new Http();
        HttpRequest request = new HttpRequest();
              request.setEndpoint(WAREHOUSE_URL);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
                    List<Product2> warehouseEq = new List<Product2>();
              if (response.getStatusCode() == 200){
            List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody());
            System.debug(response.getBody());
                    for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }
                    if (warehouseEq.size() > 0){
                upsert warehouseEq;
                System.debug('Your equipment was synced with the warehouse one');
                System.debug(warehouseEq);
            }
                  }
    }
}
```

C. Schedule synchronization

WarehouseSyncSchecdule.apxc

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
```

```
    }
}
```

D. Test automation logic

MaintenanceRequestHelperTest.apxc:

```
@istest
public with sharing class MaintenanceRequestHelperTest {

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

    PRIVATE STATIC Vehicle__c createVehicle(){
        Vehicle__c Vehicle = new Vehicle__C(name = 'SuperTruck');
        return Vehicle;
    }

    PRIVATE STATIC Product2 createEq(){
        product2 equipment = new product2(name = 'SuperEquipment',
                                          lifespan_months__C = 10,
                                          maintenance_cycle__C = 10,
                                          replacement_part__c = true);
        return equipment;
    }

    PRIVATE STATIC Case createMaintenanceRequest(id vehicleId, id equipmentId){
        case cs = new case(Type=REPAIR,
                           Status=STATUS_NEW,
                           Origin=REQUEST_ORIGIN,
                           Subject=REQUEST_SUBJECT,
                           Equipment__c=equipmentId,
                           Vehicle__c=vehicleId);
        return cs;
    }
```

```
   PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id equipmentId,id
requestId){
      Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                                Maintenance_Request__c = requestId);
      return wp;
   }


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

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

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

      Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
      insert workP;

      test.startTest();
      somethingToUpdate.status = CLOSED;
      update somethingToUpdate;
      test.stopTest();

      Case newReq = [Select id, subject, type, Equipment__c, Date_Reported__c, Vehicle__c,
Date_Due__c
                  from case
                  where status =:STATUS_NEW];

      Equipment_Maintenance_Item__c workPart = [select id
                                    from Equipment_Maintenance_Item__c
                                    where Maintenance_Request__c =:newReq.Id];
```

```
    system.assert(workPart != null);
    system.assert(newReq.Subject != null);
    system.assertEquals(newReq.Type, REQUEST_TYPE);
    SYSTEM.assertEquals(newReq.Equipment__c, equipmentId);
    SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
    SYSTEM.assertEquals(newReq.Date_Reported__c, system.today());
}

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

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

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

    Equipment_Maintenance_Item__c workP = createWorkPart(equipmentId, emptyReq.Id);
    insert workP;

    test.startTest();
    emptyReq.Status = WORKING;
    update emptyReq;
    test.stopTest();

    list<case> allRequest = [select id
                             from case];

    Equipment_Maintenance_Item__c workPart = [select id
                                              from Equipment_Maintenance_Item__c
                                              where Maintenance_Request__c = :emptyReq.Id];

    system.assert(workPart != null);
    system.assert(allRequest.size() == 1);
}

@istest
```

```
    private static void testMaintenanceRequestBulk(){
        list<Vehicle__C> vehicleList = new list<Vehicle__C>();
        list<Product2> equipmentList = new list<Product2>();
        list<Equipment_Maintenance_Item__c> workPartList = new
list<Equipment_Maintenance_Item__c>();
        list<case> requestList = new list<case>();
        list<id> oldRequestIds = new list<id>();

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

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

        for(integer i = 0; i < 300; i++){
            workPartList.add(createWorkPart(equipmentList.get(i).id, requestList.get(i).id));
        }
        insert workPartList;

        test.startTest();
        for(case req : requestList){
            req.Status = CLOSED;
            oldRequestIds.add(req.Id);
        }
        update requestList;
        test.stopTest();

        list<case> allRequests = [select id
                                  from case
                                  where status =: STATUS_NEW];

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

```
        system.assert(allRequests.size() == 300);
    }
}
```

MaintenanceRequestHelper.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;
    }
  }
}
```

MaintenanceRequest.apxc:

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

E. **Test callout logic**

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL = 'https://th-superbadge-
apex.herokuapp.com/equipment';
    public static void runWarehouseEquipmentSync(){

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

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


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

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

            for (Object eq : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)eq;
                Product2 myEq = new Product2();
                myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                myEq.Name = (String) mapJson.get('name');
                myEq.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                myEq.Cost__c = (Decimal) mapJson.get('lifespan');
                myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
                myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
                warehouseEq.add(myEq);
            }

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

```
            System.debug(warehouseEq);
        }

    }
  }
}
```

WarehouseCalloutServiceTest.apxc

```
@isTest

private class WarehouseCalloutServiceTest {
@isTest
static void testWareHouseCallout(){
Test.startTest();
Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
WarehouseCalloutService.runWarehouseEquipmentSync();
WarehouseCalloutService que= new WarehouseCalloutService();
System.enqueueJob(que);
Test.stopTest();
System.assertEquals(1, [SELECT count() FROM Product2]);
}
}
```

WarehouseCalloutServiceMock.apxc

```
@isTest
global class WarehouseCalloutServiceMock implements HttpCalloutMock {
    global static HttpResponse respond(HttpRequest request){

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

F. Test scheduling logic

WarehouseSyncSchecdule.apxc

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

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