APEX MODULES

1.APEX TRIGGERS

1.1 Get Started with Apex Triggers

AccountAddressTrigger.apxt

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

}
```

1.2 Bulk Apex Triggers

ClosedOpportunityTrigger.apxt

```
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

2.1 Get Started with Apex Unit Tests

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
        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;
        }

}
```

<u>TestVerifyDate.apxc</u>

```
@isTest
public class TestVerifyDate {

  @isTest static void test1(){
    Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('01/03/2020'));
    System.assertEquals(Date.parse('01/03/2020'),d);
  }

  @isTest static void test2(){
    Date d = VerifyDate.CheckDates(Date.parse('01/01/2020'),Date.parse('03/03/2020'));
    System.assertEquals(Date.parse('01/31/2020'),d);
  }


}
```

2.2 Test Apex Triggers

<u>RestrictContactByName.apxt</u>

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

          }

}
```

TestRestrictContactByName.apxc

```
@isTest
public class TestRestrictContactByName {

   @isTest
   public static void testContact(){
      Contact ct = new Contact();
      ct.LastName = 'INVALIDNAME';
      Database.SaveResult res = Database.insert(ct,false);
      System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',res.getErrors()[0].getMessage());
   }

}
```

2.3 Create Test Data for Apex Tests

RandomContactFactory.apxc

```
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

3.2 Use Future Methods

AccountProcessor.apxc

```
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.apxc

```
@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='bin';
        con.LastName='pro';
        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);
    }

}
```

3.3 Use Batch Apex

LeadProcessor.apxc

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

    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator(
            'SELECT ID from Lead '
        );
    }
    public 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;
    }
    public void finish(Database.BatchableContext bc){

    }
}
```

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']);
    }
}
```

3.4 Control Processes with Queueable Apex

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;
        }

    }
}
```

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;

            String state = 'CA';




            AddPrimaryContact addit = new AddPrimaryContact(testContact, 'CA');
            // startTest/stopTest block to force async processes to run
            Test.startTest();
            System.enqueueJob(addit);
            Test.stopTest();
            // Validate the job ran. Check if record have correct parentId now
            System.assertEquals(50, [Select count() from Contact where accountId in (Select Id from Account
where BillingState ='CA')]);
}
}
```

3.5 Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor.apxc

```
public class DailyLeadProcessor implements Schedulable {
    public void execute(SchedulableContext ctx) {
        List<lead> leadstoupdate = new List<lead>();
        List<Lead> leads = [SELECT Id
            FROM Lead
            WHERE LeadSource = NULL Limit 200
            ];
        for(Lead l:leads){
            l.LeadSource = 'Dreamforce';
                                leadstoupdate.add(l);
        }
        update leadstoupdate;


    }
}
```

DailyLeadProcessorTest.apxc

```
@isTest
private class DailyLeadProcessorTest {
  // Dummy CRON expression: midnight on March 15.
  // Because this is a test, job executes
  // immediately after Test.stopTest().
  public static String CRON_EXP = '0 0 0 15 3 ? 2023';
  static testmethod void testScheduledJob() {
    // Create some out of date Opportunity records
    List<Lead> leads = new List<lead>();
    for (Integer i=0; i<200; i++) {
      Lead l = new Lead(
        FirstName = 'First ' + i,
        LastName = 'LastName',
        Company = 'The Inc'
      );
      leads.add(l);
    }
    insert leads;

    Test.startTest();
    // Schedule the test job
    String jobId = System.schedule('ScheduledApexTest',
      CRON_EXP,
      new DailyLeadProcessor());
              Test.stopTest();

    // Now that the scheduled job has executed,
    // check that we have 200 leads woth dreamforce
    List<Lead> checkleads = new List<Lead>();

    checkleads = [SELECT Id
      FROM Lead
      WHERE  LeadSource='Dreamforce' and Company='The Inc'];
    System.assertEquals(200,
      checkleads.size(),
      'Leads were not created');
  }
}
```

4.APEX INTEGRATION

4.2 APEX REST CALLOUTS

AnimalLocator.apxc

```
public class AnimalLocator {

    public static String getAnimalNameById(Integer animalId) {
        String animalName;
        Http http = new Http();
        HttpRequest request = new HttpRequest();
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+animalId);
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        // If the request is successful, parse the JSON response.
        if(response.getStatusCode() == 200) {
            Map<String,Object> r = (Map<String, Object>)
                JSON.deserializeUntyped(response.getBody());
            Map<String,Object> animal = (Map<String ,Object>)r.get('animal');
            animalName = String.valueOf(animal.get('name'));


        }
        return animalName;
    }


}
```

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('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');
        response.setStatusCode(200);
        return response;
    }
}
```

AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest {
    @isTest static void getAnimalNameByIdTest() {
    // Set mock callout class
    Test.setMock(HttpCalloutMock.class, new AnimalLocatorMock());
    // This causes a fake response to be sent
    // from the class that implements HttpCalloutMock.
    String response = AnimalLocator.getAnimalNameById(1);

    // Verify that the response received contains fake values

    System.assertEquals('chicken', response);
}

}
```

4.3 APEX SOAP CALLOUTS

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;
        }
    }
}
```

<u>ParkLocator.apxc</u>

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

}
```

<u>ParkServiceMock.apxc</u>

```
public class ParkServiceMock implements WebServiceMock{
    public void doInvoke(
                      Object stub,
                      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);
          }
}
```

ParkLocatorTest.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);
    }


}
```

4.4 APEX WEB SERVICES

AccountManager.apxc

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
    @HttpGet
    global static Account getAccount(){
        RestRequest request = RestContext.request;
        String accountId = request.requestURI.substringBetween('Accounts/','/contacts');
        Account result = [SELECT Id, Name , (Select Id, Name from Contacts) from Account where
Id=:accountId];
        return result;
    }



}
```

AccountManagerTest.apxc

```
@isTest
private class AccountManagerTest {
  @isTest static void testGetContactsByAccountId(){
    Id recordId = createTestRecord();
    RestRequest request = new RestRequest();
    request.requestURI =
      'https://yourInstance.salesforce.com/service/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 accountTest = new Account(
      Name = 'Test record');
    insert accountTest;

    Contact contactTest = new Contact(
       FirstName = 'John',
       LastName = 'Doe',
       AccountId = accountTest.Id
    );
    insert contactTest;
    return accountTest.Id;

  }


}
```

Apex Specialist Superbadge

Challenge 2 : Automate record creation

MaintenanceRequest.apxt

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

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);
                }
            }
        }

        //When an existing maintenance request of type Repair or Routine Maintenance is closed,
        //create a new maintenance request for a future routine checkup.
        if (!validIds.isEmpty()){
            Map<Id,Case> closedCases = 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>();

            //calculate the maintenance request due dates by using the maintenance cycle defined on the
related equipment records.
            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,
                    Equipment__c =cc.Equipment__c,
                    Origin = 'Web',
                    Date_Reported__c = Date.Today()
                );

                //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.
                If (maintenanceCycles.containskey(cc.Id)){
                    nc.Date_Due__c = Date.today().addDays((Integer) maintenanceCycles.get(cc.Id));
                } else {
                    nc.Date_Due__c = Date.today().addDays((Integer)
cc.Equipment__r.maintenance_Cycle__c);
                }

                newCases.add(nc);
            }

            insert newCases;

            List<Equipment_Maintenance_Item__c> 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;
        }
    }
}
```

Challenge 3 : Synchronize Salesforce data with an external system

WarehouseCalloutService.apxc

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

    //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.

    @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());

            //class maps the following fields:
            //warehouse SKU will be external ID for identifying which equipment records to update within
Salesforce
            for (Object jR : jsonResponse){
                Map<String,Object> mapJson = (Map<String,Object>)jR;
                Product2 product2 = new Product2();
                //replacement part (always true),
                product2.Replacement_Part__c = (Boolean) mapJson.get('replacement');
                //cost
                product2.Cost__c = (Integer) mapJson.get('cost');
                //current inventory
                product2.Current_Inventory__c = (Double) mapJson.get('quantity');
                //lifespan
                product2.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
                //maintenance cycle
                product2.Maintenance_Cycle__c = (Integer) mapJson.get('maintenanceperiod');
                //warehouse SKU
                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');
    }

}
```

Challenge 4 : Schedule synchronization

WarehouseSyncShedule.apxc

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

Challenge 5 : 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);
    }
}
```

<u>MaintenanceRequestHelper.apxc</u>

```
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);
    }
}
```

Challenge 6 : Test callout logic

WarehouseCalloutService.apxc

```
public with sharing class WarehouseCalloutService {

    private static final String WAREHOUSE_URL =
'https://th-superbadge-apex.herokuapp.com/equipment';

    //@future(callout=true)
    public static void runWarehouseEquipmentSync(){

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

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


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

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

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

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

       }
    }
}
```

WarehouseCalloutServiceTest.apxc

```
@isTest


private class WarehouseCalloutServiceTest {
    @isTest
    static void testWareHouseCallout(){
       Test.startTest();
       // implement mock callout test here
       Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
       WarehouseCalloutService.runWarehouseEquipmentSync();
       Test.stopTest();
       System.assertEquals(1, [SELECT count() FROM Product2]);
    }
}
```

WarehouseCalloutServiceMock.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":"Genera
tor 1000 kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
        response.setStatusCode(200);
        return response;
    }
}
```

Challenge 7 : Test scheduling logic

WarehouseSyncSchedule.apxc

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

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 ');

    }
}
```