1. **Apex Triggers**

Get Started with Apex Triggers

Challenge:

Code:

```
trigger AccountAddressTrigger on Account (before insert, before update) {

    for(Account acct : Trigger.new) {
        if(acct.Match_Billing_Address__c == True)

            acct.ShippingPostalCode = acct.BillingPostalCode;

    }
}
```

Bulk Apex Triggers

Challenge:

Code:

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {

    List<Task> taskList = new List<Task>();

    for(Opportunity opp : Trigger.New)
    {
        if(opp.StageName == 'Closed Won')
        {
            taskList.add(new Task(Subject = 'Follow Up Test Task', WhatId= opp.ID));
        }
    }
        if(taskList.size()>0){
            insert taskList;

    }
}
```

2. **Apex Testing**

Get Started with Apex Unit Tests

Challenge:

Code:

```
@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);

}
}
```

Test Apex Triggers

Challenge:

Code:

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

Create Test Data for Apex Tests

Challenge:

Code:

```
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 cont = new Contact(FirstName= 'Test' +i, LastName=lastName);
            contactList.add(cont);
```

```
        }
      return contactList;
  }
}
```

3. Apex Integration Services

Apex REST Callouts

Challenge:

Code:

Class AnimalLocator:

```
public class AnimalLocator {

  public static String getAnimalNameById(Integer id) {

      Http http = new Http();
      HttpRequest request = new HttpRequest();
      request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/' + id);
      request.setMethod('GET');
      HttpResponse response = http.send(request);
      String strResp = '';
      system.debug('******response' + response.getStatusCode());
      system.debug('******response' + response.getBody());
      if (response.getStatusCode() == 200)
      {
        Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());
        Map<String, Object> animals = (Map<String, Object>) results.get('animal');
        System.debug('Received the following animals: ' + animals);
        strResp = string.valueOf(animals.get('name'));
        System.debug('strResp >' + strResp);

      }
      return strResp;
  }

}
```

Class AnimalLocatorTest:

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

    }
}
```

Class AnimalLocatorMock:

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {

    global HTTPResponse respond(HTTPRequest request) {
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animal": {"id":1,"name":"cow","eats":"grass"}}');
        response.setStatusCode(200);
        return response;
    }

}
```

Apex SOAP Callouts:

```
//Generated by wsdl2apex

public class AsyncParkService {
    public class byCountryResponseFuture extends System.WebServiceCalloutFuture {
        public String[] getValue() {
            ParkService.byCountryResponse response =
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);
            return response.return_x;
        }
    }
    public class AsyncParksImplPort {
        public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';
        public Map<String,String> inputHttpHeaders_x;
        public String clientCertName_x;
        public Integer timeout_x;
        private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};
        public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation
continuation,String arg0) {
            ParkService.byCountry request_x = new ParkService.byCountry();
            request_x.arg0 = arg0;
            return (AsyncParkService.byCountryResponseFuture) System.WebServiceCallout.beginInvoke(
              this,
              request_x,
              AsyncParkService.byCountryResponseFuture.class,
              continuation,
              new String[]{endpoint_x,
              '',
              'http://parks.services/',
              'byCountry',
              'http://parks.services/',
              'byCountryResponse',
              'ParkService.byCountryResponse'}
            );
        }
    }
}
```

Challenge:

<u>Code</u>:

<u>ParkLocator</u>

```
public class ParkLocator {
    public static string[] country(string country){
        ParkService.ParksImplPort prk = new ParkService.ParksImplPort();
        return prk.byCountry(country);
    }

}
```

<u>ParkLocatorTest</u>

```
@isTest
private class ParkLocatorTest {
   @isTest
    static void testCallout(){
        Test.setMock(WebServiceMock.class, new ParkServiceMock());
        String country = 'USA';
        System.assertEquals(new List<String>{'Me', 'You', 'Him'}, ParkLocator.country(country));
    }

}
```

<u>ParkServiceMock</u>

```
@isTest
```

```
global class ParkServiceMock implements WebServiceMock{

   global void doInvoke (
   object stub,
   object request,
      Map<String, object> response,
      String endpoint,
      String soapAction,
      String requestName,
      String responseNS,
      String responseName,
      String responseType){
         parkService.byCountryResponse response_x = new parkService.byCountryResponse();
         response_x.return_x = new List<String>{'Me', 'You', 'Him'};
            response.put('response_x', response_x);
      }
}
```

Apex Web Services

Challenge:

Code:

AccountManager

```
@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');
      system.debug(accountId);
      Account objAccount = [Select Id, Name, (Select Id, Name FROM Contacts) FROM Account
WHERE
                      id =: accountId LIMIT 1];
      return objAccount;
   }

}
```

AccountManagerTest

```
@isTest
private class AccountManagerTest {
    static testMethod void testMethod1(){
        Account objAccount = new Account(Name = 'Test Account');
        insert objAccount;
        Contact objContact = new Contact(LastName = 'Test Contact',
                                         AccountId = objAccount.Id);

        insert objContact;

        Id recordId = objAccount.Id;
        RestRequest request = new RestRequest();
        request.requestURI =
            'https://chandigarhuniversity-11a-dev-ed.lightning.force.com/services/apexrest/Accounts/'+
          recordId + '/contacts' ;
        request.httpMethod = 'GET' ;
        RestContext.request = request ;

        Account thisAccount = AccountManager.getAccount();

        system.assert(thisAccount != null);
        system.assertEquals('Test Account', thisAccount.Name);
            }
    }
```

4. Visualforce Basics

Create & Edit Visualforce Pages

Challenge:

Code:

```
<apex:page showHeader="false" title="DisplayImage" sidebar="false">
    <apex:form>
    <table>
        <tr>
            <td width="800px" height="400px" align="center">

                <apex:image url="https://developer.salesforce.com/files/salesforce-developer-network-
logo.png"/>

     </td>
           </tr>
        </table>
    </apex:form>
</apex:page>
```

Use Simple Variables and Formulas

Challenge:

Code:

```
<apex:page >
   <apex:pageBlockSection columns="1">
 {! $User.FirstName} {! $User.LastName} {! $User.UserName}
   </apex:pageBlockSection>
</apex:page>
```

Code:

```
<apex:page standardController="Contact">
   <apex:pageBlock title="Account Summary">
      <apex:pageBlockSection>
         First Name: {! Contact.FirstName} <br/>
         Last Name: {! Contact.LastName} <br/>
         Owner's Email: {! Contact.Owner.Email} <br/>
      </apex:pageBlockSection>
   </apex:pageBlock>
</apex:page>
```

Display Records, Fields, and Tables

Code:

```
<apex:page standardController="Opportunity">
   <apex:pageBlock title="Opportunity Page">
  <apex:pageBlockSection>
      <apex:outputField value="{! Opportunity.Name}"/>
     <apex:outputField value="{! Opportunity.Amount}"/>
     <apex:outputField value="{! Opportunity.CloseDate}"/>
     <apex:outputField value="{! Opportunity.Account.Name}"/>

      </apex:pageBlockSection>
   </apex:pageBlock>
</apex:page>
```

Input Data Using Forms

Challenge:

Code:

```
<apex:page standardController="Contact">
   <apex:form>
   <apex:pageBlock title="Add contects">
     <apex:pageBlockSection columns="1">
      <apex:inputField value= "{! Contact.FirstName}"/>
        <apex:inputField value= "{! Contact.LastName}"/>
        <apex:inputField value= "{! Contact.email}"/>

      </apex:pageBlockSection>
      <apex:pageBlockButtons>
      <apex:commandButton action= "{! save}" value= "Save"/>
      </apex:pageBlockButtons>
      </apex:pageBlock>
   </apex:form>
</apex:page>
```

Use Standard List Controllers

Challenge:

Code:

```
<apex:page standardController="Account" recordSetVar="Accounts">
   <apex:pageBlock>

<apex:repeat var="a" value="{!Accounts}" rendered="true" id="account_list">
    <li>'
       <apex:outputLink value="/{!a.ID}">
          <apex:outputText value="{!a.Name}"/>
</apex:outputLink>
     </li>

      </apex:repeat>
   </apex:pageBlock>
  </apex:page>
```

Use Static Resources

Challenge:

Code:

```
<apex:page >
   <apex:image url="{!URLFOR($Resource.vfimagetest, 'cats/kitten1.jpg')}"/>
```

```
</apex:page>
```

Create & Use Custom Controllers

Challenge:

Code:

NewCaseListController

```
public class NewCaseListController {
    public List<Case> getNewCases(){
        List<case> cases = [SELECT Id, CaseNumber FROM Case WHERE status = 'New'];
        return cases;
    }
}
```

Visualforce Page:

```
<apex:page controller="NewCaseListController">
    <apex:pageBlock title="New Case List" id="cases_list">
        <li>
            <apex:repeat value="{!NewCases}" var="Case" rendered="true">
                <p>
                    <apex:outputLink value="/{!Case.ID}">
                        {!Case.CaseNumber}
                    </apex:outputLink>
                </p>
            </apex:repeat>
        </li>
    </apex:pageBlock>
</apex:page>
```

5. Add a Standard Controller to the Page

Challenge:

Code:

```
<apex:page standardController="Contact">
          <head>
                    <meta charset="utf-8" />
      <meta name="viewport" content="width=device-width, initial-scale=1" />
      <title>Quick Start: Visualforce</title>
      <!-- Import the Design System style sheet -->
      <apex:slds />

         </head>
         <body>
                  <apex:form>
      <apex:pageBlock title="New Contact">
        <!--Buttons -->
         <apex:pageBlockButtons>
            <apex:commandButton action="{!save}" value="Save"/>
         </apex:pageBlockButtons>
         <!--Input form -->
         <apex:pageBlockSection columns="1">
         <apex:inputField value="{!Contact.Firstname}"/>
         <apex:inputField value="{!Contact.Lastname}"/>
         <apex:inputField value="{!Contact.Email}"/>
        </apex:pageBlockSection>
      </apex:pageBlock>
      </apex:form>

         </body>

</apex:page>
```

Asynchronous Apex

Challenge:

Code:

AccountProcessor :

```
public class AccountProcessor {
@future
    public static void countContacts(Set<id> setId){
        List<Account> lstAccount=[SELECT Id, Number_Of_Contacts__c, (SELECT Id FROM Contacts)
FROM Account WHERE Id IN:setId];
        for(Account acct : lstAccount){
            List<Contact>lstCont = acct.Contacts;
            acct.Number_Of_Contacts__c = lstCont.size();
        }
    update lstAccount;
    }
}
```

AccountProcessorTest:

```
@isTest
public class AccountProcessorTest {
    public static testmethod void TestAccountProcessorTest(){
        Account a = new Account();
        a.Name = 'Test Account';
        Insert a;
        Contact cont = New Contact();
        cont.FirstName = 'John';
        cont.LastName = 'Smith';
        cont.AccountId = a.Id;
        Insert cont;

        set<Id> setAccId = new Set<Id>();
        setAccId.add(a.Id);
        Test.startTest();
        AccountProcessor.countContacts(setAccId);
        Test.stopTest();

        Account Acc = [SELECT Number_Of_Contacts__c FROM Account WHERE Id = :a.Id LIMIT 1];
        System.assertEquals(Integer.valueOf(Acc.Number_Of_Contacts__c), 1);
    }
}
```

Use Batch Apex

Challenge:

<u>Code:</u>

LeadProcessor

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

    global Database.QueryLocator start(Database.BatchableContext bc){
        return Database.getQueryLocator('SELECT ID, LeadSource FROM Lead');
        }

    global void execute (Database.BatchableContext bc, List<Lead> l_lst){
        List<lead> l_lst_new = new List<lead>();
        for(lead l : l_lst){
        l.leadsource = 'Dreamforce';
        l_lst_new.add(l);
        count += 1;
    }
        update l_lst_new;
    }
    global void finish(Database.BatchableContext bc) {
        system.debug('count = ' + count);

    }
}
```

LeadProcessorTest

```
@isTest
public class LeadProcessorTest {
@isTest
    public static void Testit(){
        List<lead> l_lst = 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_lst.add(l);
        }
        insert l_lst;

        Test.startTest();
        LeadProcessor lp = new LeadProcessor();
        Id batchId = Database.executeBatch(lp);
        Test.stopTest();
    }
}
```

Control Processes with Queueable Apex

Challenge:

<u>Code:</u>

AddPrimaryContact

```
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);
            cont.AccountId = acc.id;
            lstContact.add(cont);
        }
        if(lstContact.size()>0){
            insert lstContact;
        }
    }

}
```

Code:

AddPrimaryContactTest:

```
@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 = 'Delhi', name = 'Test' + i ));
        }
        for(Integer j=0;j<50;j++){
            Teste.add(new Account(BillingState = 'Telangana', name = 'Test' + j));
    }
        insert Teste;
        Contact co = new Contact();
        co.FirstName = 'John';
        co.LastName= 'Richardson';
        insert co;
        String state = 'Delhi';

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

Schedule Jobs Using the Apex Scheduler

Challenge:

Code:

DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext sc){
        List<Lead> lstofLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];

        List<Lead> lstofUpdatedLead = new List<Lead>();
        if(!lstofLead.isEmpty()){
            for(Lead ld : lstofLead){
                ld.LeadSource = 'Dreamforce';
                lstofUpdatedLead.add(ld);
            }

            UPDATE lstofUpdatedLead;
        }
    }
}
```

<u>Code:</u>

DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {
@testSetup
    static void setup(){
        List<Lead> lstofLead = new List<Lead>();
        for(Integer i = 1; i <=200 ; i++){
            Lead ld = new Lead(Company = 'Company' + i, LastName = 'Smith' + i, Status = 'Working -
Contacted');
            lstofLead.add(ld);
        }
        Insert lstofLead;
    }
    static testmethod void testDailyLeadProcessorScheduledJob(){
        String sch = '0 3 8 * * ?';
        Test.startTest();
        String jobId = System.Schedule('ScheduledApexTest', sch, new DailyLeadProcessor());

        List<Lead> lstofLead = [SELECT Id FROM Lead WHERE LeadSource = null LIMIT 200];
        system.assertEquals(200, lstofLead.size());
        Test.stopTest();
    }
}
```

Apex Specialist (Superbadge)

CHALLENGE 1:

Code:

MaintenanceRequestHelper:

```
@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:

```
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;
        }
    }
}
```

CHALLENGE 2:

Code:

WarehouseCalloutService:

```
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);
            }
        }
    }
}
```

Execute anonymous window:

```
System.enqueueJob(new WarehouseCalloutService());
```

CHALLENGE 3:

Code:

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

CHALLENGE 4:

Code:

MaintenanceRequestHelperTest :

```
@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:

```
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:

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

CHALLENGE 5:

Code:

WarehouseCalloutService

```
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:

```
@isTest

private class WarehouseCalloutServiceTest {
   @isTest
   static void testWareHouseCallout(){
      Test.startTest();

      Test.setMock(HTTPCalloutMock.class, new WarehouseCalloutServiceMock());
      WarehouseCalloutService.runWarehouseEquipmentSync();
      Test.stopTest();
      System.assertEquals(1, [SELECT count() FROM Product2]);
   }
}
```

WarehouseCalloutServiceMock:

```
@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;
    }
}
```

<u>CHALLENGE 6:</u>

<u>Code:</u>

WarehouseSyncSchedule:

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

WarehouseSyncSchedule Test:

```
@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 ');


    }
}
```