Module Name: Apex Triggers

Statement: Create an Apex trigger that sets an account's Shipping Postal Code to match the Billing Postal Code if the Match Billing Address option is selected. Fire the trigger before inserting an account or updating an account.

Trigger Name: AccountAddressTrigger

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

Statement: Create a bulkified Apex trigger that adds a follow-up task to an opportunity if its stage is Closed Won. Fire the Apex trigger after inserting or updating an opportunity.

Trigger Name: ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
    List<Task> taskList = new List<Task>();
    for(Opportunity opp: Trigger.new){
        if(Trigger.isInsert){
            if(opp.StageName =='Closed Won'){
                taskList.add(new Task(Subject='Follow Up Test Task', WhatId=opp.Id));
            }
        }
```

```
        if(Trigger.isUpdate){
            if(opp.StageName!='Closed Won' && opp.StageName!=
Trigger.oldMap.get(opp.Id).StageName){
                taskList.add(new Task(Subject='Follow Up Test Task', WhatId=opp.Id));
            }
        }
    }
    if(taskList.size()>0){
        insert taskList;
    }
}
```

Module Name: Apex Testing

Statement: Create and install a simple Apex class to test if a date is within a proper range, and if not, returns a date that occurs at the end of the month within the range. Write unit tests that achieve 100% code coverage.

Class Name: VerifyDate.apxc

```
public class VerifyDate {
    public static Date CheckDates(Date date1, Date date2) {
        //if date2 is within the next 30 days of date1, use date2. Otherwise use the end of the month
        if(DateWithin30Days(date1,date2)) {
            return date2;
        }
        else {
            return SetEndOfMonthDate(date1);
        }
    }

    //method to check if date2 is within the next 30 days of date1
    private static Boolean DateWithin30Days(Date date1, Date date2) {
            //check for date2 being in the past
            if( date2 < date1) {
                return false;
            }
            //check that date2 is within (>=) 30 days of date1
            Date date30Days = date1.addDays(30);
            //create a date 30 days away from date1
            if( date2 >= date30Days ){
                return false;
            }
```

```
        else {
            return true;
        }
    }

    //method to return the end of the month of a given date
    private static Date SetEndOfMonthDate(Date date1) {
        Integer totalDays = Date.daysInMonth(date1.year(), date1.month());
        Date lastDay = Date.newInstance(date1.year(), date1.month(), totalDays);
        return lastDay;
    }
}
```

Test Class Name: TestVerifyDate.apxc

```
@isTest
public class TestVerifyDate {
    @isTest static void testIfSmaller(){
        date d1 = date.newInstance(2016,01,12);
         date d2 = date.newinstance(2016,02,09);

        Date dtest = VerifyDate.CheckDates( d1 , d2 );
        system.assertEquals(d2, dtest);
    }
    @isTest static void testDateNotWithin30(){

        date d1 = date.newinstance(2016,01,12);
        date d3 = date.newInstance(2016,02,22);
```

```
        date d4endofmonth = date.newinstance(2016,01,31);

        Date dtest = VerifyDate.CheckDates(d1, d3);
        system.assertEquals(d4endofmonth, dtest);
    }
}
```

Statement: Create and install a simple Apex trigger which blocks inserts and updates to any contact with a last name of 'INVALIDNAME'. Write unit tests that achieve 100% code coverage.

Class Name: RestrictContactByName.apxc

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

Test Class Name: TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {
    @isTest static void testContactName(){
```

```
        Contact c=new Contact();
        c.LastName='INVALIDNAME';
        Database.SaveResult result=Database.insert(c,false);
        System.assertEquals('The Last Name "INVALIDNAME" is not allowed for
DML',result.getErrors()[0].getMessage());
    }
}
```

Statement: Create an Apex class that returns a list of contacts based on two incoming parameters: the number of contacts to generate and the last name. Do not insert the generated contact records into the database.

Class Name: RandomContactFactory

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer noOfContacts, String
lastName){
        List<Contact> contacts = new List<Contact>();
        for(Integer i=0;i<noOfContacts;i++){
            Contact c=new Contact(FirstName='Test'+i,LastName=lastName);
            contacts.add(c);
        }
        return contacts;
    }
}
```

Module Name: Asynchronous Apex

Statement: Create an Apex class that uses the @future annotation to update Account records. Create an Apex class with a future method that accepts a List of Account IDs and updates a custom field on the Account object with the number of contacts associated to the Account. Write unit tests that achieve 100% code coverage for the class. Every hands-on challenge in this module asks you to create a test class

Class Name: 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 acc : lstAccount ){
        List<Contact> lstCont = acc.contacts ;
        acc.Number_of_Contacts__c = lstCont.size();
     }
     update lstAccount;
  }
}
```

Test Class Name : 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 ='Bob';
        cont.LastName ='Masters';
        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);
    }
}
```

Statement: Create an Apex class that uses Batch Apex to update Lead records. Create an Apex class that implements the Database.Batchable interface to update all Lead records in the org with a specific LeadSource.

Class Name: LeadProcessor

```
global class LeadProcessor implements Database.Batchable<Sobject> {
    global Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator([Select LeadSource From Lead ]);
    }
    global void execute(Database.BatchableContext bc, List<Lead> scope) {
```

```
        for(Lead Leads : scope){
            Leads.LeadSource = 'Dreamforce';
        }
    update scope;
  }
  global void finish(Database.BatchableContext bc){ }
}
```

Test Class Name: LeadProcessorTest

```
@isTest
public class LeadProcessorTest
{
  static testMethod void testMethod1()
  {
    List<Lead> lstLead = new List<Lead>();
    for(Integer i=0 ;i <200;i++)
    {
      Lead led = new Lead();
      led.FirstName ='FirstName';
      led.LastName ='LastName'+i;
      led.Company ='demo'+i;
      lstLead.add(led);
    }
    insert lstLead;
    Test.startTest();
      LeadProcessor obj = new LeadProcessor();
      DataBase.executeBatch(obj);
```

```
        Test.stopTest();
    }
}
```

Statement: Create a Queueable Apex class that inserts Contacts for Accounts. Create a Queueable Apex class that inserts the same Contact for each Account for a specific state.

Class Name: AddPrimaryContact

```
public class AddPrimaryContact implements Queueable {
    private String st;
    private Contact primecontact;

    public AddPrimaryContact(Contact aContact, String aState) {
        this.st = aState;
        this.primecontact = aContact;
    }
    public void execute(QueueableContext context) {
        List<Account> accounts = [select name from account where billingstate=:st LIMIT 200];
        List<Contact> contacts = new List<Contact>();
        for (Account acc : accounts) {
            contact con=primecontact.clone(false,false,false,false);
            contacts.add(con);
        }
        insert contacts;
    }
}
```

Test Class Name: AddPrimaryContactTest

```
@isTest
public class AddPrimaryContactTest {
    @testSetup
    static void setup(){
        List<Account> accounts =new List<Account>();
        for(Integer i = 0; i < 50; i++){
            accounts.add(new Account(Name='NY'+ i,billingstate ='CA'));
        }
        for(Integer j=0;j<50;j++){
            accounts.add(new Account(Name='CA'+j,billingstate='CA'));
        }
        insert accounts;
    }
    static testMethod void testQueueable(){
        contact a=new Contact(LastName='mary',FirstName='rose');
        Test.startTest();
        AddPrimaryContact updater=new AddPrimaryContact(a,'CA');
        System.enqueueJob(updater);
        Test.stopTest();
    }
}
```

Statement: Create an Apex class that uses Scheduled Apex to update Lead records. Create an Apex class that implements the Schedulable interface to update Lead records with a specific LeadSource.

Class Name: DailyLeadProcessor

```
global class DailyLeadProcessor implements Schedulable{
    global void execute(SchedulableContext ctx){
        List<Lead> leads = [SELECT Id, LeadSource FROM Lead WHERE LeadSource = ''];

        if(leads.size() > 0){
            List<Lead> newLeads = new List<Lead>();

            for(Lead lead : leads){
                lead.LeadSource = 'DreamForce';
                newLeads.add(lead);
            }

            update newLeads;
        }
    }
}
```

Test Class Name: DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest{
    //Seconds Minutes Hours Day_of_month Month Day_of_week optional_year
    public static String CRON_EXP = '0 0 0 2 6 ? 2022';
```

```
    static testmethod void testScheduledJob(){
        List<Lead> leads = new List<Lead>();

        for(Integer i = 0; i < 200; i++){
            Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test
Company ' + i, Status = 'Open - Not Contacted');
            leads.add(lead);
        }

        insert leads;

        Test.startTest();
        // Schedule the test job
        String jobId = System.schedule('Update LeadSource to DreamForce', CRON_EXP, new
DailyLeadProcessor());

        // Stopping the test will run the job synchronously
        Test.stopTest();
    }
}
```

Module Name: Apex Integration Services

Statement: Create an Apex class that calls a REST endpoint to return the name of an animal, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

Class Name: AnimalLocator.apxc

```
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);
        /*Map<String,Object> results =
(Map<String,Object>)JSON.deserializeUntyped(response.getBody());
        system.debug('---->results'+results);
        List<Object> animals = (List<Object>) results.get('animal');
        system.debug('---->animal'+animals);*/
        Map<Integer,String> mapAnimal = new Map<Integer,String>();
        Integer varId;
        String varName;
        JSONParser parser1= JSON.createParser(response.getBody());
        while (parser1.nextToken() != null) {
            if ((parser1.getCurrentToken() == JSONToken.FIELD_NAME) && (parser1.getText()
== 'id')) {
                // Get the value.
                parser1.nextToken();
                // Fetch the ids for all animals in JSON Response.
```

```
            varId=parser1.getIntegerValue();
            System.debug('---->varId-->'+varID);
            parser1.nextToken();
        }
        if ((parser1.getCurrentToken() == JSONToken.FIELD_NAME) && (parser1.getText()
== 'name')) {
            parser1.nextToken();
            // Fetch the names for all animals in JSON Response.
            varName=parser1.getText();
            System.debug('---->varName-->'+varName);
        }
        mapAnimal.put(varId,varName);
      }
      system.debug('---->mapAnimal-->'+mapAnimal);
      return mapAnimal.get(id);
   }
}
```

Class Name: 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"},{"id":2,"name":"duck","eats":"worms","says":"pek pek"}]}');
        response.setStatusCode(200);
        return response;
    }
}
```

Class Name: AnimalLocatorTest.apxc

```
@isTest
private class AnimalLocatorTest {
    @isTest
    static void testGetCallout() {
        // 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);
        system.debug(response);
        String expectedValue = 'chicken';
        System.assertEquals(expectedValue,response);
        String response2 = AnimalLocator.getAnimalNameById(2);
        system.debug(response2);
        String expectedValue2 = 'duck';
        System.assertEquals(expectedValue2,response2);
    }
}
```

Statement: Generate an Apex class using WSDL2Apex for a SOAP web service, write unit tests that achieve 100% code coverage for the class using a mock response, and run your Apex tests.

Class Name: ParkServiceMock.apxc

```
@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, responseName, String responseType){
     ParkService.byCountryResponse response_x = new ParkService.byCountryResponse();
     response_x.return_x = new List<String> {'India','SriLanka','Australia','England'};
       response.put('response_x',response_x);
     }
}
```

Class Name: ParkLocator.apxc

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

Test Class Name: ParkLocatorTest.apxc

```
@isTest
public class ParkLocatorTest {
   @isTest
   static void testCalloout(){
      Test.setMock(WebServiceMock.class, new ParkServiceMock());
      String country='United States';
      System.assertEquals(new List<String>
{'India','SriLanka','Australia','England'},ParkLocator.country(country));
   }
}
```

Statement: Create Apex REST class that is accessible at /Accounts/<Account_ID>/contacts. The service will return the account's ID and name plus the ID and name of all contacts associated with the account. Write unit tests that achieve 100% code coverage for the class and run your Apex tests.

Class Name: AccountManager.apxc

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

```
    }
}
```

Class Name: AccountManagerTest.apxc

```
@isTest
private class {
    @isTest
    static void testGetAccount(){
        Account a=new Account(Name='TestAccount');
        insert a;
        Contact c=new Contact(AccountId=a.Id, FirstName='Test', LastName='Test');
        insert c;
        //Create a new RestRequest
        RestRequest request=new RestRequest();

request.requestURI='https://yourInstance.my.salesforce.com/services/apexrest/Accounts/'+a.id
+'/contacts';
        request.httpMethod='GET';
        RestContext.request=request;
        Account myAccount= AccountManager.getAccount();
        //assert the results
        System.assert(myAccount!=Null);
        System.assertEquals('TestAccount', myAccount.Name);
    }
}
```

Module Name: Apex Specialist SuperBadge

Class Name: 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));
            }
            else{
                nc.Date_Due__c = Date.today().addDays((Integer)
cc.Product.Maintenance_Cycle__c);
            }
                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;
    }
  }
}
```

Class Name: MaintenanceRequest.apxt

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

Class Name: WarehouseCalloutService

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

  //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(){
        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());

            //class maps the following fields: replacement part (always true), cost, current inventory,
lifespan, maintenance cycle, and warehouse SKU
            //warehouse SKU will be external ID for identifying which equipment records to update
within Salesforce
            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 = (Integer) mapJson.get('cost');
            myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
            myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
            myEq.ProductCode = (String) mapJson.get('_id');
            warehouseEq.add(myEq);
        }
        if (warehouseEq.size() > 0){
            upsert warehouseEq;
            System.debug('Your equipment was synced with the warehouse one');
        }
    }
  }
    public static void execute (QueueableContext context){
     runWarehouseEquipmentSync();
  }
}
```

Class Name: WarehouseSyncSchedule

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

Class Name: MaintenanceRequestHelperTest

```
@istest
public with sharing class MaintenanceRequestHelperTest {

    private static final string Status_Check = '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_Check, 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_Check];
        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_Check];

        list<Equipment_Maintenance_Item__c> workParts = [select id from
Equipment_Maintenance_Item__c where Maintenance_Request__c in: oldRequestIds];
        system.assert(allRequests.size() == 300);
    }
}
```

Module Name: Visualforce Basics

Page Name: DisplayImage.vfp

```
<apex:page showHeader="false">
<apex:image value="https://developer.salesforce.com/files/salesforce-developer-network-logo.png" />
</apex:page>
```

Page Name: DisplayUserInfo.vfp

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

Page Name: ContactView.vfp

```
<apex:page standardController="Contact">
  <apex:pageBlock>
  <apex:pageBlockSection>
      First Name:{!Contact.FirstName}
```

```
        Last Name:{!Contact.LastName}
        Owner Email:{!Contact.Owner.Email}
    </apex:pageBlockSection>
    </apex:pageBlock>
</apex:page>
```

Page Name: OppView.vfp

```
<apex:page standardController="Opportunity">
    <apex:outputField value="{!Opportunity.Name}"/>
    <apex:outputField value="{!Opportunity.Amount}"/>
    <apex:outputField value="{!Opportunity.Closedate}"/>
    <apex:outputField value="{!Opportunity.Account.Name}"/>
</apex:page>
```

Page Name: CreateContact.vfp

```
<apex:page standardController="Contact">
    <apex:form >
    <apex:pageBlock >
        <apex:pageBlockSection >

        <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>
```

Page Name: AccountList.vfp

```
<apex:page standardController="Account" recordSetVar="accounts">
  <apex:repeat value="{! accounts}" var="a">
    <li>
      <apex:outputLink value="/{! a.ID }">
      Click here for information on {!a.ID}
      </apex:outputLink>
    </li>
  </apex:repeat>
</apex:page>
```

Page Name: ShowImage.vfp

```
<apex:page >

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

</apex:page>
```

Page Name: NewCaseList.vfp

```
<apex:page controller="NewCaseListController">
   <apex:pageBlock >
   <apex:repeat value="{!newCases}" var="case" id="theRepeat">
      <apex:outputLink value="/!{case.id}">{!case.id}
         <apex:outputText value="{!case.casenumber}">
         </apex:outputText>
      </apex:outputLink>
   </apex:repeat>
      </apex:pageBlock>
</apex:page>
```

Controller Class Name: NewCaseListController.apxc

```
public class NewCaseListController {
   public List<Case> getNewCases(){
      List<Case> ListCase=[select Id, CaseNumber from case where status='New'];
      return ListCase;
   }
}
```

Module Name: Lightning Web Component Basics

File Name: Selector.html

```
<template>
   <div class="wrapper">
   <!-- <header class="header">Select a Bike</header> -->
   <header class="header">Available Bikes for {name}</header>
   <section class="content">
      <div class="columns">
      <main class="main" >
        <c-list onproductselected={handleProductSelected}></c-list>
      </main>
      <aside class="sidebar-second">
         <c-detail product-id={selectedProductId}></c-detail>
      </aside>
      </div>
   </section>
   </div>
</template>
```

File Name: Selector.js

```
import { LightningElement, wire } from 'lwc';
import { getRecord, getFieldValue } from 'lightning/uiRecordApi';
import Id from '@salesforce/user/Id';
import NAME_FIELD from '@salesforce/schema/User.Name';
const fields = [NAME_FIELD];
export default class Selector extends LightningElement {
   selectedProductId;
```

```
    handleProductSelected(evt) {
        this.selectedProductId = evt.detail;
    }
    userId = Id;
    @wire(getRecord, { recordId: '$userId', fields })
    user;
    get name() {
        return getFieldValue(this.user.data, NAME_FIELD);
    }
}
```