Apex All Codes

Apex Superbadge

step 2

Helper class ///////

```
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)
```

Apex All Codes

```
                                    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)){
```

Apex All Codes

```
                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> 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;
    }
  }
```

Apex All Codes

}

Trigger Class ///

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

step 3

WarehouseCalloutService class //

/// first you need to add url in remote site settings ->>
name = WarehouseCalloutService URL
https://th-superbadge-apex.herokuapp.com/equipment

/// after that paste this code

```
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.
```

Apex All Codes

```
  @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');
      }
```

Apex All Codes

```
        }
    }

    public static void execute (QueueableContext context){
        runWarehouseEquipmentSync();
    }

}
```

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

step 4

WarehouseSyncSchedule class /////

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

step 5

Test class //

```
@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';
```

Apex All Codes

```
  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();
```

Apex All Codes

```
        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(){
```

Apex All Codes

```
        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>();
```

Apex All Codes

```
        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);
    }
```

Apex All Codes

```
}

/////////////////
Helper class ///


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'));
      }
```

Apex All Codes

```
        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;
    }
  }
```

Apex All Codes

```
}
```

////////////////
MaintenanceRequest.apxc

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

step 6

Test class //

```
@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;
   }
```

Apex All Codes

```
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;
```

Apex All Codes

```
}



@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();
```

Apex All Codes

```
    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;
```

Apex All Codes

```
        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);
    }
```

Apex All Codes

```
   @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));
      }
```

Apex All Codes

```
        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);
    }
}
```

////////////////

Helper class ///

Apex All Codes

```
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,
```

Apex All Codes

```
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));
            }
```

Apex All Codes

```
            newCases.add(nc);
        }


        insert newCases;


        List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
        for (Case nc : newCases){
            for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                Equipment_Maintenance_Item__c wpClone = wp.clone();
                wpClone.Maintenance_Request__c = nc.Id;
                ClonedWPs.add(wpClone);


            }
        }
        insert ClonedWPs;
    }
  }
}
```

////////////////

MaintenanceRequest.apxc

Apex All Codes

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

step 7

Test class //

```
@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){
```

Apex All Codes

```
        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;
```

Apex All Codes

```
        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;
```

Apex All Codes

```
        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));
```

Apex All Codes

```
        }
        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);
    }
}

//////////////////
Helper class ///


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);
```

Apex All Codes

```
            }
        }
    }

    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));
```

Apex All Codes

```
            }

            newCases.add(nc);
          }

          insert newCases;

          List<Equipment_Maintenance_Item__c> clonedWPs = new
List<Equipment_Maintenance_Item__c>();
          for (Case nc : newCases){
              for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_Items__r){
                  Equipment_Maintenance_Item__c wpClone = wp.clone();
                  wpClone.Maintenance_Request__c = nc.Id;
                  ClonedWPs.add(wpClone);

              }
          }
          insert ClonedWPs;
      }
    }
}

////////////////
MaintenanceRequest.apxc

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

DailyLeadProcessor

```
public class DailyLeadProcessor implements Schedulable  {
    Public void execute(SchedulableContext SC){
       List<Lead> LeadObj=[SELECT Id from Lead where LeadSource=null limit 200];
        for(Lead l:LeadObj){
```

Apex All Codes

```
            l.LeadSource='Dreamforce';
            update l;
        }
    }
}
```

DailyLeadProcessorTest

```
@isTest
private class DailyLeadProcessorTest {
        static testMethod void testDailyLeadProcessor() {
                String CRON_EXP = '0 0 1 * * ?';
                List<Lead> lList = new List<Lead>();
            for (Integer i = 0; i < 200; i++) {
                        lList.add(new Lead(LastName='Dreamforce'+i, Company='Test1
Inc.', Status='Open - Not Contacted'));
                }
                insert lList;

                Test.startTest();
                String jobId = System.schedule('DailyLeadProcessor', CRON_EXP, new
DailyLeadProcessor());
        }
}
```

AccountManager

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

Apex All Codes

```
}
```

AccountManagerTest

```
@isTest
private class AccountManagerTest {

    private static testMethod void getAccountTest1() {
        Id recordId = createTestRecord();
        // Set up a test request
        RestRequest request = new RestRequest();
        request.requestUri = 'https://na1.salesforce.com/services/apexrest/Accounts/'+
 recordId +'/contacts' ;
        request.httpMethod = 'GET';
        RestContext.request = request;
        // Call the method to test
        Account thisAccount = AccountManager.getAccount();
        // Verify results
        System.assert(thisAccount != null);
        System.assertEquals('Test record', thisAccount.Name);

    }

    // Helper method
        static Id createTestRecord() {
        // Create test record
        Account TestAcc = new Account(
          Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(
        LastName='Test',
        AccountId = TestAcc.id);
        return TestAcc.Id
;
    }
}
```

Animal Locator

Apex All Codes

```
public class AnimalLocator{
   public static String getAnimalNameById(Integer x){
      Http http = new Http();
      HttpRequest req = new HttpRequest();
      req.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'
 + x);
      req.setMethod('GET');
      Map<String, Object> animal= new Map<String, Object>();
      HttpResponse res = http.send(req);
         if (res.getStatusCode() == 200) {
      Map<String, Object> results = (Map<String,
Object>)JSON.deserializeUntyped(res.getBody());
     animal = (Map<String, Object>) results.get('animal');
      }
return (String)animal.get('name');
   }
}
```

Animal LocatorTest

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

Animal LocatorMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    // Implement this interface method
   global HTTPResponse respond(HTTPRequest request) {
      // Create a fake response
```

Apex All Codes

```
        HttpResponse response = new HttpResponse();
        response.setHeader('Content-Type', 'application/json');
        response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear",
"chicken", "mighty moose"]}');
        response.setStatusCode(200);
        return response;
    }
}
```

ParkLocator

```
public class ParkLocator {
    public static string[] country(string theCountry) {
        ParkService.ParksImplPort  parkSvc = new  ParkService.ParksImplPort(); // remove
space
        return parkSvc.byCountry(theCountry);
    }
}
```

ParkLocatorTest

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

ParkSevice

```
//Generated by wsdl2apex

public class ParkService {
```

Apex All Codes

```
  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,
```

Apex All Codes

```
            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;
      }
   }
}
```

ParkSeviceMock

```
@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) {
      // start - specify the response you want to send
      ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
      // end
      response.put('response_x', response_x);
  }
}
```

Apex All Codes

AsynParkservice

```
//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'}
```

```
            );
        }
    }
}
```

NewCaseListController

```
public class NewCaseListController {
    public List<Case> getNewCases(){
        List<Case> filterList = [Select ID ,CaseNumber from Case where status = 'New'];
        return filterList;
    }

}
```

AccountProcessor

```
public class AccountProcessor {
    @future
    public static void countContacts(List<Id> accountIds){
        List<Account> accounts = [Select Id, Name from Account Where Id IN : accountIds];
        List<Account> updatedAccounts = new List<Account>();
        for(Account account : accounts){
          account.Number_of_Contacts__c = [Select count() from Contact Where AccountId
=: account.Id];
            System.debug('No Of Contacts = ' + account.Number_of_Contacts__c);
            updatedAccounts.add(account);
        }
        update updatedAccounts;
    }

}
```

AccountProcessorTest

```
@isTest
public class AccountProcessorTest {
    @isTest
```

Apex All Codes

```
   public static void testNoOfContacts(){
      Account a = new Account();
      a.Name
 = 'Test Account';
      Insert a;

      Contact c = new Contact();
      c.FirstName = 'Bob';
      c.LastName =  'Willie';
      c.AccountId = a.Id
;

      Contact c2 = new Contact();
      c2.FirstName = 'Tom';
      c2.LastName = 'Cruise';
      c2.AccountId = a.Id
;

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

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

}
```

LeadProcessor

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

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

Apex All Codes

```
        for (Lead Lead : leads) {
            lead.LeadSource = 'Dreamforce';
        }
        update leads;
    }
    public void finish(Database.BatchableContext bc){
     }

}
```

LeadProcessorTest

```
@isTest
public class LeadProcessorTest {

        @testSetup
    static void setup() {
        List<Lead> leads = new List<Lead>();
        for(Integer counter=0 ;counter <200;counter++){
            Lead lead = new Lead();
            lead.FirstName ='FirstName';
            lead.LastName ='LastName'+counter;
            lead.Company
 ='demo'+counter;
            leads.add(lead);
        }
        insert leads;
    }

    @isTest static void test() {
        Test.startTest();
        LeadProcessor leadProcessor = new LeadProcessor();
        Id batchId = Database.executeBatch(leadProcessor);
        Test.stopTest();
    }

}
```

Apex All Codes

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,false);
            cont.AccountId =  acc.id
;
            lstContact.add( cont );
       }

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

   }

}
```

AddPrimaryContactTest

```
@isTest
public class AddPrimaryContactTest
{
```

Apex All Codes

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

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

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

PushPriceChangeNotification:

```
public with sharing class PushPriceChangeNotification {

    @InvocableMethod(label='Push Price Change Notification')
    public static void pushNotification(List<Id> propertyId) {
        String pushServerURL;
              Dreamhouse_Settings__c settings =
Dreamhouse_Settings__c.getOrgDefaults();
        if (!Test.isRunningTest()) {
            if (settings == null || settings.Push_Server_URL__c == null) {
```

Apex All Codes

```
                System.debug('Push_Server_URL not set. Aborting
PushPriceChangeNotification process action');
            return;
          } else {
          pushServerURL = settings.Push_Server_URL__c;
          }
      }
      Id propId = propertyId[0]; // If bulk, only post first to avoid spamming
      Property__c property = [SELECT Name, Price__c from Property__c WHERE
Id=:propId];
      String message = property.Name + '. New Price: $' +
property.Price__c.setScale(0).format();

      Set<String> userIds = new Set<String>();

      List<Favorite__c> favorites = [SELECT user__c from favorite__c WHERE
property__c=:propId];
      for (Favorite__c favorite : favorites) {
         userIds.add(favorite.user__c);
      }

                Map<String,Object> payload = new Map<String,Object>();
                payload.put('message', message);
                payload.put('userIds', userIds);
      String body = JSON.serialize(payload);
      System.enqueueJob(new QueueablePushCall(pushServerURL, 'POST', body));
   }

   public class QueueablePushCall implements System.Queueable,
Database.AllowsCallouts {

      private final String url;
      private final String method;
      private final String body;

      public QueueablePushCall(String url, String method, String body) {
         this.url = url;
```

```
        this.method = method;
        this.body = body;
    }

    public void execute(System.QueueableContext ctx) {
        HttpRequest req = new HttpRequest();
        req.setMethod(method);
        req.setHeader('Content-Type', 'application/json');
        req.setBody(body);
        Http http = new Http();
        HttpResponse res;
        if (!Test.isRunningTest()) {
                req.setEndpoint(url);
                res = http.send(req);
        }
    }

  }

}
```

PushPriceChangeNotificationTest:

```
@isTest
public class PushPriceChangeNotificationTest {

  static testMethod void testPush() {
    Boolean success = true;
    try {
      Property__c p = new Property__c(Name='test property', Price__c=200000);
      insert p;
          PushPriceChangeNotification.pushNotification(new List<Id> { p.Id });
    } catch (Exception e) {
      success = false;
    } finally {
          System.assert(success);
    }
```

```
    }

}
```

PrpertyController

```
global with sharing class PropertyController {

    @AuraEnabled
    public static PropertyPagedResult findAll(String searchKey, Decimal minPrice,
Decimal maxPrice, Decimal pageSize, Decimal pageNumber) {
                    Integer pSize = (Integer)pageSize;
        String key = '%' + searchKey + '%';
        Integer offset = ((Integer)pageNumber - 1) * pSize;
        PropertyPagedResult r =  new PropertyPagedResult();
        r.pageSize = pSize;
        r.page = (Integer) pageNumber;
        r.total = [SELECT count() FROM property__c
                WHERE (title__c LIKE :key OR city__c LIKE :key OR tags__c LIKE :key)
                AND price__c >= :minPrice
            AND price__c <= :maxPrice];
        r.properties = [SELECT Id, title__c, city__c, description__c, price__c, baths__c,
beds__c, thumbnail__c FROM property__c
                WHERE (title__c LIKE :key OR city__c LIKE :key OR tags__c LIKE :key)
                AND price__c >= :minPrice
                                        AND price__c <= :maxPrice
                ORDER BY price__c LIMIT :pSize OFFSET :offset];
        System.debug(r);
        return r;
    }

    @AuraEnabled
    public static Property__c findById(Id propertyId) {
        return [SELECT id, name, beds__c, baths__c, address__c, city__c, state__c,
assessed_value__c, price__c, Date_Listed__c, Location__Latitude__s,
Location__Longitude__s
            FROM Property__c
```

Apex All Codes

```
            WHERE Id=:propertyId];
    }

    @RemoteAction @AuraEnabled
    public static Property__c[] getAvailableProperties() {
        return [SELECT id, name, address__c, city__c, price__c, Date_Listed__c,
Days_On_Market__c, Date_Agreement__c, Location__Latitude__s,
Location__Longitude__s
            FROM Property__c
            WHERE Date_Listed__c != NULL AND (Date_Agreement__c = NULL OR
Date_Agreement__c = LAST_N_DAYS:90)];
    }

    @AuraEnabled
    public static List<Property__c> getSimilarProperties (Id propertyId, Decimal
bedrooms, Decimal price, String searchCriteria) {
        if (searchCriteria == 'Bedrooms') {
            return [
                SELECT Id, Name, Beds__c, Baths__c, Price__c, Broker__c, Status__c,
Thumbnail__c
                FROM Property__c WHERE Id != :propertyId AND Beds__c = :bedrooms
            ];
        } else {
            return [
                SELECT Id, Name, Beds__c, Baths__c, Price__c, Broker__c, Status__c,
Thumbnail__c
                FROM Property__c WHERE Id != :propertyId AND Price__c > :price - 100000
AND Price__c < :price + 100000
            ];
        }
    }


}
```

PrpertyControllerTest:

Apex All Codes

```
@isTest
public class PropertyControllerTest {

    static testMethod void testFindAll() {
        Boolean success = true;
        try {
            Property__c p = new Property__c(Location__Latitude__s=-
71.110448,Location__Longitude__s=42.360642);
            insert p;
                PropertyPagedResult r = PropertyController.findAll('', 0, 1000000, 8, 1);
        } catch (Exception e) {
            success = false;
        } finally {
                System.assert(success);
        }
    }

    static testMethod void testFindById() {
        Boolean success = true;
        try {
            Property__c p = new Property__c(Location__Latitude__s=-
71.110448,Location__Longitude__s=42.360642);
            insert p;
                Property__c property = PropertyController.findById(p.Id);
        } catch (Exception e) {
            success = false;
        } finally {
                System.assert(success);
        }
    }

    static testMethod void getAvailableProperties() {
        Boolean success = true;
        try {
            Property__c p = new Property__c(Location__Latitude__s=-
71.110448,Location__Longitude__s=42.360642);
            insert p;
```

Apex All Codes

```
            Property__c[] r = PropertyController.getAvailableProperties();
        } catch (Exception e) {
            success = false;
        } finally {
            System.assert(success);
        }
    }

    static testMethod void getSimilarProperties() {
        Boolean success = true;
        try {
            Property__c p = new Property__c(Location__Latitude__s=-
71.110448,Location__Longitude__s=42.360642);
            insert p;
            Property__c[] r = PropertyController.getSimilarProperties(p.Id, 3, 500000,
'Bedrooms');
        } catch (Exception e) {
            success = false;
        } finally {
            System.assert(success);
        }
    }

}
```

LIFX Controller

```
public with sharing class LIFXController {

    private static final Dreamhouse_Settings__c settings =
Dreamhouse_Settings__c.getOrgDefaults();

    @AuraEnabled
    public static String getLights() {
        HttpRequest req = new HttpRequest();
        Http http = new Http();
```

Apex All Codes

```
    req.setMethod('GET');
    req.setHeader('Authorization', 'Bearer ' + settings.LIFX_TOKEN__C);
    req.setEndpoint(settings.LIFX_URL__C + '/all');
                try {
        HTTPResponse res = http.send(req);
                        return res.getBody();
    } catch(Exception ex){
        return '{"error": "' + ex.getMessage() + '"}';
    }
}

@AuraEnabled
public static String setPower(String lightId, Boolean isOn) {
    return LIFXController.setState(lightId, '{"power": "' + (isOn == true ? 'on' : 'off') + '"}');
}

@AuraEnabled
public static String setBrightness(String lightId, Decimal brightness) {
    return LIFXController.setState(lightId, '{"brightness": ' + (brightness / 100) + '}');
}

public static String setState(String lightId, String state) {
    HttpRequest req = new HttpRequest();
    Http http = new Http();
    req.setMethod('PUT');
    req.setEndpoint(settings.LIFX_URL__C + '/' + lightId + '/state');
    req.setHeader('Authorization', 'Bearer ' + settings.LIFX_TOKEN__C);
    req.setHeader('Content-Type', 'application/json');
    req.setBody(state);
                try {
        HTTPResponse res = http.send(req);
                        return res.getBody();
    } catch(Exception ex){
        return '{"error": "' + ex.getMessage() + '"}';
    }
}
```

Apex All Codes

```
}
```

LIFX ControllerTest:

```
public class PostPriceChangeToSlack {

   @InvocableMethod(label='Post Price Change Notification to Slack')
   public static void postToSlack(List<Id> propertyId) {
                String slackURL;
           Dreamhouse_Settings__c settings =
Dreamhouse_Settings__c.getOrgDefaults();
      if (!Test.isRunningTest()) {
         if (settings == null || settings.Slack_Property_Webhook_URL__c == null) {
                     System.Debug('Slack_Property_Webhook_URL not set. Aborting
PostPriceChangeToSlack process action');
            return;
         } else {
         slackURL = settings.Slack_Property_Webhook_URL__c;
         }
      }
      Id propId = propertyId[0]; // If bulk, only post first to avoid spamming
      Property__c property = [SELECT Address__c, City__c, State__c, Price__c from
Property__c WHERE Id=:propId];
      String message = 'Price change: ' + property.Address__c + ', ' + property.City__c + ' '
+ property.State__c + ' is now *$' + property.Price__c.setScale(0).format() + '*';
      System.Debug(message);

                Map<String,Object> payload = new Map<String,Object>();
                payload.put('text', message);
                payload.put('mrkdwn', true);
      String body = JSON.serialize(payload);
      System.Debug(body);
      System.enqueueJob(new QueueableSlackCall(slackURL, 'POST', body));
   }

   public class QueueableSlackCall implements System.Queueable,
Database.AllowsCallouts {

      private final String url;
```

Apex All Codes

```
    private final String method;
    private final String body;

    public QueueableSlackCall(String url, String method, String body) {
      this.url = url;
      this.method = method;
      this.body = body;
    }

    public void execute(System.QueueableContext ctx) {
      HttpRequest req = new HttpRequest();
      req.setMethod(method);
      req.setBody(body);
      Http http = new Http();
      HttpResponse res;
                        if (!Test.isRunningTest()) {
              req.setEndpoint(url);
                                res = http.send(req);
      }
    }

  }

}
```

DreamHouseSampleDataController.apxc

```
global with sharing class DreamHouseSampleDataController {

  @RemoteAction
  global static void deleteAll() {
    DELETE [SELECT ID FROM favorite__c];
    DELETE [SELECT ID FROM property__c];
    DELETE [SELECT ID FROM broker__c];
    DELETE [SELECT ID FROM bot_command__c];
  }
```

Apex All Codes

```
}
```

HandlerMyOpenCases:

```
public with sharing class HandlerMyOpenCases implements BotHandler {

    public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {

        List<Case> cases =
            [SELECT Id, CaseNumber, Subject, Status, Priority, Contact.Id, Contact.Name
             FROM Case WHERE OwnerId =:UserInfo.getUserId() AND Status != 'Closed'];

        List<BotRecord> records = new List<BotRecord>();

        for (Case c : cases) {
            List<BotField> fields = new List<BotField>();
            fields.add(new BotField('Case Number', c.CaseNumber, '#/sObject/' + c.Id +
'/view'));
            fields.add(new BotField('Subject', c.Subject));
            fields.add(new BotField('Priority', c.Priority));
            fields.add(new BotField('Status', c.Status));
            fields.add(new BotField('Contact', c.Contact.Name, '#/sObject/' + c.Contact.Id +
'/view'));
            records.add(new BotRecord(fields));
        }
        BotMessage message = new BotMessage('Bot', 'Here are your open cases:',
records);
        return new BotResponse(message);

    }

}
```

HandlerTravellerApproval:

Apex All Codes

```
public class HandlerTravelApproval implements BotHandler {

        public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
      if (session == null) {
         BotMessage message = new BotMessage('Bot', 'Where are you going?');
         session = new Map<String, String>();
         session.put('nextCommand', 'HandlerTravelApproval');
         session.put('step', 'destination');
         return new BotResponse(message, session);
      }
                  String step = session.get('step');
      if (step == 'destination') {
         session.put('destination', utterance);
                        List<BotMessageButton> buttons = new
List<BotMessageButton>();
         buttons.add(new BotMessageButton('Customer Facing', 'Customer Facing'));
         buttons.add(new BotMessageButton('Internal Meetings', 'Internal Meetings'));
         buttons.add(new BotMessageButton('Billable Work', 'Billable Work'));
         BotMessage message = new BotMessage('Bot', 'What\'s the reason for the trip?',
buttons);
         session.put('nextCommand', 'HandlerTravelApproval');
         session.put('step', 'reason');
         return new BotResponse(message, session);
      } else if (step == 'reason') {
         session.put('reason', utterance);
         BotMessage message = new BotMessage('Bot', 'When are you leaving?');
         session.put('nextCommand', 'HandlerTravelApproval');
         session.put('step', 'travelDate');
         return new BotResponse(message, session);
      } else if (step == 'travelDate') {
         session.put('travelDate', utterance);
         BotMessage message = new BotMessage('Bot', 'What\'s the estimated airfare
cost?');
         session.put('nextCommand', 'HandlerTravelApproval');
         session.put('step', 'airfare');
         return new BotResponse(message, session);
```

Apex All Codes

```
    } else if (step == 'airfare') {
        session.put('airfare', utterance);
        BotMessage message = new BotMessage(' Bot', 'What\'s the estimated hotel
cost?');
        session.put('nextCommand', 'HandlerTravelApproval');
        session.put('step', 'hotel');
        return new BotResponse(message, session);
    }
    List<Botrecord> records = new List<BotRecord>();
    List<BotField> fields = new List<BotField>();
    fields.add(new BotField('Destination', session.get('destination')));
    fields.add(new BotField('Reason', session.get('reason')));
    fields.add(new BotField('Travel Date', session.get('travelDate')));
    fields.add(new BotField('Airfare', session.get('airfare')));
    fields.add(new BotField('Hotel', utterance));
    records.add(new BotRecord(fields));
            return new BotResponse(new BotMessage('Bot', 'OK, I submitted the
following travel approval request on your behalf:', records));

  }


}
```

HandlerTopOppurtunities:

```
public with sharing class HandlerTopOpportunities implements BotHandler {

  public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
    Integer qty = Integer.valueof(params[0]);
    List<Opportunity> opportunities =
      [SELECT Id, Name, Amount, Probability, StageName, CloseDate FROM
Opportunity where isClosed=false ORDER BY amount DESC LIMIT :qty];
```

Apex All Codes

```
        List<BotRecord> records = new List<BotRecord>();

        for (Opportunity o : opportunities) {
            List<BotField> fields = new List<BotField>();
            fields.add(new BotField('Name', o.Name, '#/sObject/' + o.Id + '/view'));
            fields.add(new BotField('Amount', '$' + o.Amount));
            fields.add(new BotField('Probability', '' + o.Probability + '%'));
            fields.add(new BotField('Stage', o.StageName));
            records.add(new BotRecord(fields));
        }
        return new BotResponse(new BotMessage('Bot', 'Here are your top ' + params[0] + '
opportunities:', records));

    }

}
```

HandlerSOQL :

```
public with sharing class HandlerSOQL implements BotHandler {

    public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {

        SObject[] objects = Database.query(utterance);

        List<BotRecord> records = new List<BotRecord>();

        for (sObject o : objects) {
            List<BotField> fields = new List<BotField>();
            Map<String, Object> fieldMap = o.getPopulatedFieldsAsMap();
            for (String fieldName : fieldMap.keySet()) {
                String linkURL;
                if (fieldName == 'Id') {
                    linkURL = '#/sObject/' + o.Id + '/view';
```

Apex All Codes

```
            }
            fields.add(new BotField(fieldName, '' + fieldMap.get(fieldName), linkURL));
          }
          records.add(new BotRecord(fields));
        }
        return new BotResponse(new BotMessage('Bot', 'Here is the result of your query:',
records));

    }

}
```

HandlerFindProperties:

```
public class HandlerFindProperties implements BotHandler {

    private String formatCurrency(Decimal i) {
        if (i == null) return '0.00';
        i = Decimal.valueOf(Math.roundToLong(i * 100)) / 100;
        String s = (i.setScale(2) + (i >= 0 ? 0.001 : -0.001)).format();
        return s.substring(0, s.length() - 1);
    }

            public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
        if (session == null) {
            BotMessage message = new BotMessage('Bot', 'What City?');
            session = new Map<String, String>();
            session.put('nextCommand', 'HandlerFindProperties');
            session.put('step', 'city');
            return new BotResponse(message, session);
        }
                    String step = session.get('step');
        if (step == 'city') {
            session.put('city', utterance);
```

Apex All Codes

```
                List<BotMessageButton> buttons = new
List<BotMessageButton>();
        buttons.add(new BotMessageButton('Single Family', 'Single Family'));
        buttons.add(new BotMessageButton('Condominium', 'Condominium'));
        BotMessage message = new BotMessage('Bot', 'What type of property?',
buttons);
        session.put('nextCommand', 'HandlerFindProperties');
        session.put('step', 'type');
        return new BotResponse(message, session);
      } else if (step == 'type') {
        session.put('type', utterance);
        BotMessage message = new BotMessage('Bot', 'Price range from?');
        session.put('nextCommand', 'HandlerFindProperties');
        session.put('step', 'minPrice');
        return new BotResponse(message, session);
      } else if (step == 'minPrice') {
        session.put('minPrice', utterance);
        BotMessage message = new BotMessage('Bot', 'Price range to?');
        session.put('nextCommand', 'HandlerFindProperties');
        session.put('step', 'maxPrice');
        return new BotResponse(message, session);
      } else if (step == 'maxPrice') {
        session.put('maxPrice', utterance);
        String city = session.get('city');
        Decimal minPrice = Decimal.valueOf(session.get('minPrice'));
        Decimal maxPrice = Decimal.valueOf(session.get('maxPrice'));
        List<Property__c> properties =
          [SELECT Id, Name, Beds__c, Baths__c, Price__c FROM Property__c
           WHERE City__c = :city AND
           Price__c >= :minPrice AND
           Price__c <= :maxPrice
           ORDER BY Price__c
           LIMIT 5];

        List<BotRecord> records = new List<BotRecord>();

        for (Property__c p : properties) {
```

```
            List<BotField> fields = new List<BotField>();
            fields.add(new BotField('Name', p.Name, '#/sObject/' + p.Id + '/view'));
            fields.add(new BotField('Bedrooms', '' + p.Beds__c));
            fields.add(new BotField('Baths', '' + p.Baths__c));
            fields.add(new BotField('Price', '' + this.formatCurrency(p.Price__c)));
            records.add(new BotRecord(fields));
          }
          return new BotResponse(new BotMessage('Bot', 'Here is a list of properties in ' +
city + ' between ' + this.formatCurrency(minPrice) + ' and ' +
this.formatCurrency(maxPrice) + ': ', records));
        } else {
          return new BotResponse(new BotMessage('Bot', 'Sorry, I don\'t know how to
handle that'));
        }
    }

}
```

HandlerFindContact:

```
public with sharing class HandlerFindContact implements BotHandler {

    public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
        String key = '%' + params[0] + '%';
        List<Contact> contacts =
            [SELECT Id, Name, MobilePhone FROM Contact
             WHERE Name LIKE :key
             ORDER BY Name
             LIMIT 5];

        List<BotRecord> records = new List<BotRecord>();

        for (Contact c : contacts) {
            List<BotField> fields = new List<BotField>();
```

Apex All Codes

```
            fields.add(new BotField('Name', c.Name, '#/sObject/' + c.Id + '/view'));
            fields.add(new BotField('Phone', c.MobilePhone, 'tel:' + c.MobilePhone));
            records.add(new BotRecord(fields));
        }
        return new BotResponse(new BotMessage('Bot', 'Here is a list of contacts matching
''' + params[0] + ''':', records));

    }

}
```

HandlerFindAccount:

```
public with sharing class HandlerFindAccount implements BotHandler {

    public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
        String key = '%' + params[0] + '%';
        List<Account> accounts =
            [SELECT Id, Name, Phone FROM Account
             WHERE Name LIKE :key
             ORDER BY Name
             LIMIT 5];

        List<BotRecord> records = new List<BotRecord>();

        for (Account a : accounts) {
            List<BotField> fields = new List<BotField>();
            fields.add(new BotField('Name', a.Name, '#/sObject/' + a.Id + '/view' ));
            fields.add(new BotField('Phone', a.Phone, 'tel:' + a.Phone));
            records.add(new BotRecord(fields));
        }
        return new BotResponse(new BotMessage('Bot', 'Here is a list of accounts
matching ''' + params[0] + ''':', records));
```

```
    }

}
```

HandlerFieldUpdate:

```
public with sharing class HandlerFileUpload implements BotHandler {

        public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
      try {
        ContentVersion v = new ContentVersion();
        v.versionData = EncodingUtil.base64Decode(fileContent);
        v.title = fileName;
        v.pathOnClient = fileName;
        insert v;
                        ContentDocument doc = [SELECT Id FROM ContentDocument
where LatestPublishedVersionId = :v.Id];
                        List<BotRecord> records = new List<BotRecord>();
        List<BotField> fields = new List<BotField>();
        fields.add(new BotField('Id', v.Id, '#/sObject/ContentDocument/' + doc.Id));
        fields.add(new BotField('Name', v.title));
        records.add(new BotRecord(fields));
            return new BotResponse(new BotMessage('Bot', 'Your file was uploaded
successfully', records));
      } catch (Exception e) {
                        return new BotResponse(new BotMessage('Bot', 'An error occured
while uploading the file'));
      }
    }

}
```

HandlerImageBasedSearch:

```
public with sharing class HandlerImageBasedSearch implements BotHandler {
```

Apex All Codes

```
  private String modelId = 'VNAIIMX543MNUEKPW6UWAJPKKY';

  private String formatCurrency(Decimal i) {
    if (i == null) return '0';
    i = Decimal.valueOf(Math.roundToLong(i * 100)) / 100;
    String s = (i.setScale(2) + (i >= 0 ? 0.001 : -0.001)).format();
    return '$' + s.substring(0, s.length() - 1);
  }

      public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {

    List<EinsteinVisionController.Prediction> predictions =
EinsteinVisionController.predict('', fileContent, modelId);
    List<BotRecord> records = new List<BotRecord>();
    for (EinsteinVisionController.Prediction p : predictions) {
      List<BotField> fields = new List<BotField>();
      fields.add(new BotField('House Type', p.label));
      fields.add(new BotField('Probability', '' + (p.probability * 100).round() +'%'));
      records.add(new BotRecord(fields));
    }

    BotMessage predictionMessage = new BotMessage('DreamBot', null, records);

    String key = '%' + predictions[0].label + '%';
    List<Property__c> properties =
      [SELECT Id, Name, Beds__c, Baths__c, Tags__c, Price__c FROM Property__c
       WHERE tags__c LIKE :key
       ORDER BY Price__c
       LIMIT 5];
    List<BotRecord> propertyRecords = new List<BotRecord>();
    for (Property__c p : properties) {
      List<BotField> fields = new List<BotField>();
      fields.add(new BotField('Name', p.Name, '#/sObject/' + p.Id + '/view'));
      fields.add(new BotField('Bedrooms', '' + p.Beds__c));
      fields.add(new BotField('Category', '' + p.Tags__c));
```

Apex All Codes

```
            fields.add(new BotField('Price', '' + this.formatCurrency(p.Price__c)));
            propertyRecords.add(new BotRecord(fields));
        }
        BotMessage propertyMessage = new BotMessage('DreamBot', 'Here is a list of
houses that look similar:', propertyRecords);

        BotResponse r = new BotResponse();

        r.messages = new BotMessage[] {predictionMessage, propertyMessage};

        return r;

    }

}
```

HandlerHelpTopic:

```
public with sharing class HandlerImageBasedSearch implements BotHandler {

    private String modelId = 'VNAIIMX543MNUEKPW6UWAJPKKY';

    private String formatCurrency(Decimal i) {
        if (i == null) return '0';
        i = Decimal.valueOf(Math.roundToLong(i * 100)) / 100;
        String s = (i.setScale(2) + (i >= 0 ? 0.001 : -0.001)).format();
        return '$' + s.substring(0, s.length() - 1);
    }

            public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {

        List<EinsteinVisionController.Prediction> predictions =
EinsteinVisionController.predict('', fileContent, modelId);
        List<BotRecord> records = new List<BotRecord>();
```

Apex All Codes

```
        for (EinsteinVisionController.Prediction p : predictions) {
            List<BotField> fields = new List<BotField>();
            fields.add(new BotField('House Type', p.label));
            fields.add(new BotField('Probability', '' + (p.probability * 100).round() +'%'));
            records.add(new BotRecord(fields));
        }

        BotMessage predictionMessage = new BotMessage('DreamBot', null, records);

        String key = '%' + predictions[0].label + '%';
        List<Property__c> properties =
            [SELECT Id, Name, Beds__c, Baths__c, Tags__c, Price__c FROM Property__c
             WHERE tags__c LIKE :key
             ORDER BY Price__c
             LIMIT 5];
        List<BotRecord> propertyRecords = new List<BotRecord>();
        for (Property__c p : properties) {
            List<BotField> fields = new List<BotField>();
            fields.add(new BotField('Name', p.Name, '#/sObject/' + p.Id + '/view'));
            fields.add(new BotField('Bedrooms', '' + p.Beds__c));
            fields.add(new BotField('Category', '' + p.Tags__c));
            fields.add(new BotField('Price', '' + this.formatCurrency(p.Price__c)));
            propertyRecords.add(new BotRecord(fields));
        }
        BotMessage propertyMessage = new BotMessage('DreamBot', 'Here is a list of
houses that look similar:', propertyRecords);

        BotResponse r = new BotResponse();

        r.messages = new BotMessage[] {predictionMessage, propertyMessage};

        return r;

    }

}
```

Apex All Codes

HandlerHelper:

```
public with sharing class HandlerHelp implements BotHandler {

   public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {

                List<Bot_Command__c> commands =
         [SELECT Id, Sample_Utterance__c FROM Bot_Command__c
          WHERE Sample_Utterance__c != null And Active__C = True ORDER BY
Sample_Utterance__c];

                List<BotItem> items = new List<BotItem>();

      for (Bot_Command__c c : commands) {
         items.add(new BotItem(c.Sample_Utterance__c));
      }

      BotMessage message = new BotMessage('Bot', 'You can ask me things like:',
items);
      return new BotResponse(message);
   }

}
```

HandlerEmployeeId:

```
public with sharing class HandlerEmployeeId implements BotHandler {

   public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
      return new BotResponse(new BotMessage('Bot', 'Your employee id is 9854'));
   }
```

Apex All Codes

```
}
```

HandlerCostCenter:

```
public with sharing class HandlerCostCenter implements BotHandler {

   public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
      return new BotResponse(new BotMessage('Bot', 'Your cost center is 21852'));
   }

}
```

HandlerAddTwoNumbers:

```
public with sharing class HandlerAddTwoNumbers implements BotHandler {

   public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
      if (session == null) {
         session = new Map<String, String>();
         session.put('nextCommand', 'HandlerAddTwoNumbers');
         session.put('step', 'askFirstNumber');
         return new BotResponse(new BotMessage('Bot', 'What\'s the first number?'),
session);
      }
      String step = session.get('step');
      if (step == 'askFirstNumber') {
         session.put('firstNumber', utterance);
         session.put('nextCommand', 'HandlerAddTwoNumbers');
         session.put('step', 'askSecondNumber');
         return new BotResponse(new BotMessage('Bot', 'What\'s the second number?'),
session);
      } else {
                              Integer firstNumber = Integer.valueof(session.get('firstNumber'));
         Integer secondNumber = Integer.valueof(utterance);
         Integer total = firstNumber + secondNumber;
         BotMessage message = new BotMessage('Bot', '' + firstNumber + ' + ' +
```

Apex All Codes

```
secondNumber + ' = ' + total);
            return new BotResponse(message);
        }

    }

}
```

WAreHouseCalloutService:

```
public with sharing class WarehouseCalloutService {

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

}
```

TestRestictByName:

```
@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());
    }

}
```

RandomContactFactory:

```
public class RandomContactFactory {
    public static List<Contact> generateRandomContacts (Integer num,String lastName){
        List<Contact>contactList = new List<Contact>();
```

Apex All Codes

```
        for (Integer i = 1;i<=num ;i++){
            Contact ct = new Contact(FirstName = 'Test'+i, LastName = lastName);
            contactList.add(ct);
        }
        return contactlist;
    }

}
```

VarifyDate:

```
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;
        }

}
```

VarifyDateTest:

```
@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);
   }


}
```

AccountAddressTrigger:

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

Apex All Codes

```
}
```

CloseOpportunityTriger:

```
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;
   }

}
```

RestrictContactByName:

```
trigger RestrictContactByName on Contact (before insert, before update) {

        //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');
                }

        }



}
```

CreateDefaultData:

```
public with sharing class CreateDefaultData{
```

Apex All Codes

```
    Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine Maintenance';
    //gets value from custom metadata How_We_Roll_Settings__mdt to know if Default
data was created
    @AuraEnabled
    public static Boolean isDataCreated() {
        How_We_Roll_Settings__c      customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        return customSetting.Is_Data_Created__c;
    }

    //creates Default Data for How We Roll application
    @AuraEnabled
    public static void createDefaultData(){
        List<Vehicle__c> vehicles = createVehicles();
        List<Product2> equipment = createEquipment();
        List<Case> maintenanceRequest = createMaintenanceRequest(vehicles);
        List<Equipment_Maintenance_Item__c> joinRecords =
createJoinRecords(equipment, maintenanceRequest);

        updateCustomSetting(true);
    }


    public static void updateCustomSetting(Boolean isDataCreated){
        How_We_Roll_Settings__c      customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = isDataCreated;
        upsert customSetting;
    }

    public static List<Vehicle__c> createVehicles(){
        List<Vehicle__c> vehicles = new List<Vehicle__c>();
        vehicles.add(new Vehicle__c(Name = 'Toy Hauler RV', Air_Conditioner__c = true,
Bathrooms__c = 1, Bedrooms__c = 1, Model__c = 'Toy Hauler RV'));
        vehicles.add(new Vehicle__c(Name = 'Travel Trailer RV', Air_Conditioner__c = true,
Bathrooms__c = 2, Bedrooms__c = 2, Model__c = 'Travel Trailer RV'));
        vehicles.add(new Vehicle__c(Name = 'Teardrop Camper', Air_Conditioner__c = true,
```

Apex All Codes

```
Bathrooms__c = 1, Bedrooms__c = 1, Model__c = 'Teardrop Camper'));
        vehicles.add(new Vehicle__c(Name = 'Pop-Up Camper', Air_Conditioner__c = true,
Bathrooms__c = 1, Bedrooms__c = 1, Model__c = 'Pop-Up Camper'));
        insert vehicles;
        return vehicles;
    }

    public static List<Product2> createEquipment(){
        List<Product2> equipments = new List<Product2>();
        equipments.add(new Product2(Warehouse_SKU__c =
'55d66226726b611100aaf741',name = 'Generator 1000 kW', Replacement_Part__c =
true,Cost__c = 100 ,Maintenance_Cycle__c = 100));
        equipments.add(new Product2(name = 'Fuse 20B',Replacement_Part__c =
true,Cost__c = 1000, Maintenance_Cycle__c = 30  ));
        equipments.add(new Product2(name = 'Breaker 13C',Replacement_Part__c =
true,Cost__c = 100  , Maintenance_Cycle__c = 15));
        equipments.add(new Product2(name = 'UPS 20 VA',Replacement_Part__c =
true,Cost__c = 200  , Maintenance_Cycle__c = 60));
        insert equipments;
        return equipments;

    }

    public static List<Case> createMaintenanceRequest(List<Vehicle__c> vehicles){
        List<Case> maintenanceRequests = new List<Case>();
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(1).Id, Type =
TYPE_ROUTINE_MAINTENANCE, Date_Reported__c = Date.today()));
        maintenanceRequests.add(new Case(Vehicle__c = vehicles.get(2).Id, Type =
TYPE_ROUTINE_MAINTENANCE, Date_Reported__c = Date.today()));
        insert maintenanceRequests;
        return maintenanceRequests;
    }

    public static List<Equipment_Maintenance_Item__c>
createJoinRecords(List<Product2> equipment, List<Case> maintenanceRequest){
        List<Equipment_Maintenance_Item__c> joinRecords = new
List<Equipment_Maintenance_Item__c>();
```

Apex All Codes

```
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(0).Id, Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(1).Id, Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(2).Id, Maintenance_Request__c = maintenanceRequest.get(0).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(0).Id, Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(1).Id, Maintenance_Request__c = maintenanceRequest.get(1).Id));
        joinRecords.add(new Equipment_Maintenance_Item__c(Equipment__c =
equipment.get(2).Id, Maintenance_Request__c = maintenanceRequest.get(1).Id));
        insert joinRecords;
        return joinRecords;

    }
}
```

CreateDefaultDataTest:

```
@isTest
private class CreateDefaultDataTest {
    @isTest
    static void createData_test(){
        Test.startTest();
        CreateDefaultData.createDefaultData();
        List<Vehicle__c> vehicles = [SELECT Id FROM Vehicle__c];
        List<Product2> equipment = [SELECT Id FROM Product2];
        List<Case> maintenanceRequest = [SELECT Id FROM Case];
        List<Equipment_Maintenance_Item__c> joinRecords = [SELECT Id FROM
Equipment_Maintenance_Item__c];

        System.assertEquals(4, vehicles.size(), 'There should have been 4 vehicles
created');
        System.assertEquals(4, equipment.size(), 'There should have been 4 equipment
created');
        System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2
```

```
maintenance request created');
        System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment
maintenance items created');

    }

    @isTest
    static void updateCustomSetting_test(){
        How_We_Roll_Settings__c      customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
        customSetting.Is_Data_Created__c = false;
        upsert customSetting;

        System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom setting
How_We_Roll_Settings__c.Is_Data_Created__c should be false');

        customSetting.Is_Data_Created__c = true;
        upsert customSetting;

        System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting
How_We_Roll_Settings__c.Is_Data_Created__c should be true');

    }
}
```

SampleDataController:

```
public with sharing class SampleDataController {
    @AuraEnabled
    public static void importSampleData() {
        delete [SELECT Id FROM Case];
        delete [SELECT Id FROM Property__c];
        delete [SELECT Id FROM Broker__c];
        delete [SELECT Id FROM Contact];

        insertBrokers();
        insertProperties();
        insertContacts();
```

Apex All Codes

```
    }

    private static void insertBrokers() {
        StaticResource brokersResource = [
            SELECT Id, Body
            FROM StaticResource
            WHERE Name = 'sample_data_brokers'
        ];
        String brokersJSON = brokersResource.body.toString();
        List<Broker__c> brokers = (List<Broker__c>) JSON.deserialize(
            brokersJSON,
            List<Broker__c>.class
        );
        insert brokers;
    }

    private static void insertProperties() {
        StaticResource propertiesResource = [
            SELECT Id, Body
            FROM StaticResource
            WHERE Name = 'sample_data_properties'
        ];
        String propertiesJSON = propertiesResource.body.toString();
        List<Property__c> properties = (List<Property__c>) JSON.deserialize(
            propertiesJSON,
            List<Property__c>.class
        );
        randomizeDateListed(properties);
        insert properties;
    }

    private static void insertContacts() {
        StaticResource contactsResource = [
            SELECT Id, Body
            FROM StaticResource
            WHERE Name = 'sample_data_contacts'
        ];
```

Apex All Codes

```
        String contactsJSON = contactsResource.body.toString();
        List<Contact> contacts = (List<Contact>) JSON.deserialize(
            contactsJSON,
            List<Contact>.class
        );
        insert contacts;
    }

    private static void randomizeDateListed(List<Property__c> properties) {
        for (Property__c property : properties) {
            property.Date_Listed__c =
                System.today() - Integer.valueof((Math.random() * 90));
        }
    }
}
```

TestSampleDataController:

```
@isTest
private class TestSampleDataController {
    @isTest
    static void importSampleData() {
        Test.startTest();
        SampleDataController.importSampleData();
        Test.stopTest();

        Integer propertyNumber = [SELECT COUNT() FROM Property__c];
        Integer brokerNumber = [SELECT COUNT() FROM Broker__c];
        Integer contactNumber = [SELECT COUNT() FROM Contact];

        System.assert(propertyNumber > 0, 'Expected properties were created.');
        System.assert(brokerNumber > 0, 'Expected brokers were created.');
        System.assert(contactNumber > 0, 'Expected contacts were created.');
    }
}
```

Apex All Codes

PropertyController :

```
public with sharing class PropertyController {
    private static final Decimal DEFAULT_MAX_PRICE = 9999999;
    private static final Integer DEFAULT_PAGE_SIZE = 9;

    /**
     * Endpoint that retrieves a paged and filtered list of properties
     * @param searchKey String used for searching on property title, city and tags
     * @param maxPrice Maximum price
     * @param minBedrooms Minimum number of bedrooms
     * @param minBathrooms Minimum number of bathrooms
     * @param pageSize Number of properties per page
     * @param pageNumber Page number
     * @return PagedResult object holding the paged and filtered list of properties
     */
    @AuraEnabled(cacheable=true)
    public static PagedResult getPagedPropertyList(
        String searchKey,
        Decimal maxPrice,
        Integer minBedrooms,
        Integer minBathrooms,
        Integer pageSize,
        Integer pageNumber
    ) {
        // Normalize inputs
        Decimal safeMaxPrice = (maxPrice == null
            ? DEFAULT_MAX_PRICE
            : maxPrice);
        Integer safeMinBedrooms = (minBedrooms == null ? 0 : minBedrooms);
        Integer safeMinBathrooms = (minBathrooms == null ? 0 : minBathrooms);
        Integer safePageSize = (pageSize == null
            ? DEFAULT_PAGE_SIZE
            : pageSize);
        Integer safePageNumber = (pageNumber == null ? 1 : pageNumber);
```

Apex All Codes

```
String searchPattern = '%' + searchKey + '%';
Integer offset = (safePageNumber - 1) * safePageSize;

PagedResult result = new PagedResult();
result.pageSize = safePageSize;
result.pageNumber = safePageNumber;
result.totalItemCount = [
    SELECT COUNT()
    FROM Property__c
    WHERE
        (Name LIKE :searchPattern
        OR City__c LIKE :searchPattern
        OR Tags__c LIKE :searchPattern)
        AND Price__c <= :safeMaxPrice
        AND Beds__c >= :safeMinBedrooms
        AND Baths__c >= :safeMinBathrooms
];
result.records = [
    SELECT
        Id,
        Address__c,
        City__c,
        State__c,
        Description__c,
        Price__c,
        Baths__c,
        Beds__c,
        Thumbnail__c,
        Location__Latitude__s,
        Location__Longitude__s
    FROM Property__c
    WHERE
        (Name LIKE :searchPattern
        OR City__c LIKE :searchPattern
        OR Tags__c LIKE :searchPattern)
        AND Price__c <= :safeMaxPrice
```

Apex All Codes

```
            AND Beds__c >= :safeMinBedrooms
            AND Baths__c >= :safeMinBathrooms
        WITH SECURITY_ENFORCED
        ORDER BY Price__c
        LIMIT :safePageSize
        OFFSET :offset
    ];
    return result;
}

/**
 * Endpoint that retrieves pictures associated with a property
 * @param propertyId Property Id
 * @return List of ContentVersion holding the pictures
 */
@AuraEnabled(cacheable=true)
public static List<ContentVersion> getPictures(Id propertyId) {
    List<ContentDocumentLink> links = [
        SELECT Id, LinkedEntityId, ContentDocumentId
        FROM ContentDocumentLink
        WHERE
            LinkedEntityId = :propertyId
            AND ContentDocument.FileType IN ('PNG', 'JPG', 'GIF')
        WITH SECURITY_ENFORCED
    ];

    if (links.isEmpty()) {
        return null;
    }

    Set<Id> contentIds = new Set<Id>();

    for (ContentDocumentLink link : links) {
        contentIds.add(link.ContentDocumentId);
    }

    return [
```

```
            SELECT Id, Title
            FROM ContentVersion
            WHERE ContentDocumentId IN :contentIds AND IsLatest = TRUE
            WITH SECURITY_ENFORCED
            ORDER BY CreatedDate
        ];
    }
}
```

TestPropertyController:

```
@isTest
private class TestPropertyController {
    private final static String MOCK_PICTURE_NAME = 'MockPictureName';

    public static void createProperties(Integer amount) {
        List<Property__c> properties = new List<Property__c>();
        for (Integer i = 0; i < amount; i++) {
            properties.add(
                new Property__c(
                    Name = 'Name ' + i,
                    Price__c = 20000,
                    Beds__c = 3,
                    Baths__c = 3
                )
            );
        }
        insert properties;
    }
    static testMethod void testGetPagedPropertyList() {
        TestPropertyController.createProperties(5);
        Test.startTest();
        PagedResult result = PropertyController.getPagedPropertyList(
            '',
            999999,
            0,
            0,
            10,
```

Apex All Codes

```
            1
        );
        Test.stopTest();
        System.assertEquals(5, result.records.size());
    }

    static testMethod void testGetPicturesNoResults() {
        Property__c property = new Property__c(Name = 'Name');
        insert property;

        Test.startTest();
        List<ContentVersion> items = PropertyController.getPictures(
            property.Id
        );
        Test.stopTest();

        System.assertEquals(null, items);
    }

    static testMethod void testGetPicturesWithResults() {
        Property__c property = new Property__c(Name = 'Name');
        insert property;

        // Insert mock picture
        ContentVersion picture = new Contentversion();
        picture.Title = MOCK_PICTURE_NAME;
        picture.PathOnClient = 'picture.png';
        picture.Versiondata = EncodingUtil.base64Decode('MockValue');
        insert picture;

        // Link picture to property record
        List<ContentDocument> documents = [
            SELECT Id, Title, LatestPublishedVersionId
            FROM ContentDocument
            LIMIT 1
        ];
        ContentDocumentLink link = new ContentDocumentLink();
```

Apex All Codes

```
        link.LinkedEntityId = property.Id;
        link.ContentDocumentId = documents[0].Id;
        link.shareType = 'V';
        insert link;

        Test.startTest();
        List<ContentVersion> items = PropertyController.getPictures(
            property.Id
        );
        Test.stopTest();

        System.assertEquals(1, items.size());
        System.assertEquals(MOCK_PICTURE_NAME, items[0].Title);
    }
}
```

GeocodingService:

```
public with sharing class GeocodingService {
    private static final String BASE_URL =
'https://nominatim.openstreetmap.org/search?format=json';

    @InvocableMethod(callout=true label='Geocode address')
    public static List<Coordinates> geocodeAddresses(
        List<GeocodingAddress> addresses
    ) {
        List<Coordinates> computedCoordinates = new List<Coordinates>();

        for (GeocodingAddress address : addresses) {
            String geocodingUrl = BASE_URL;
            geocodingUrl += (String.isNotBlank(address.street))
                ? '&street=' + address.street
                : '';
            geocodingUrl += (String.isNotBlank(address.city))
                ? '&city=' + address.city
                : '';
            geocodingUrl += (String.isNotBlank(address.state))
                ? '&state=' + address.state
```

```
            : '';
        geocodingUrl += (String.isNotBlank(address.country))
            ? '&country=' + address.country
            : '';
        geocodingUrl += (String.isNotBlank(address.postalcode))
            ? '&postalcode=' + address.postalcode
            : '';

        Coordinates coords = new Coordinates();
        if (geocodingUrl != BASE_URL) {
            Http http = new Http();
            HttpRequest request = new HttpRequest();
            request.setEndpoint(geocodingUrl);
            request.setMethod('GET');
            request.setHeader(
                'http-referer',
                URL.getSalesforceBaseUrl().toExternalForm()
            );
            HttpResponse response = http.send(request);
            if (response.getStatusCode() == 200) {
                List<Coordinates> deserializedCoords = (List<Coordinates>)
JSON.deserialize(
                    response.getBody(),
                    List<Coordinates>.class
                );
                coords = deserializedCoords[0];
            }
        }

        computedCoordinates.add(coords);
      }
      return computedCoordinates;
    }

    public class GeocodingAddress {
      @InvocableVariable
      public String street;
```

```
        @InvocableVariable
        public String city;
        @InvocableVariable
        public String state;
        @InvocableVariable
        public String country;
        @InvocableVariable
        public String postalcode;
    }

    public class Coordinates {
        @InvocableVariable
        public Decimal lat;
        @InvocableVariable
        public Decimal lon;
    }
}
```

GeocodingServiceTest:

```
@isTest
private with sharing class GeocodingServiceTest {
    private static final String STREET = 'Camino del Jueves 26';
    private static final String CITY = 'Armilla';
    private static final String POSTAL_CODE = '18100';
    private static final String STATE = 'Granada';
    private static final String COUNTRY = 'Spain';
    private static final Decimal LATITUDE = 3.123;
    private static final Decimal LONGITUDE = 31.333;

    @isTest
    static void successResponse() {
        // GIVEN
        GeocodingService.GeocodingAddress address = new
GeocodingService.GeocodingAddress();
        address.street = STREET;
        address.city = CITY;
        address.postalcode = POSTAL_CODE;
```

Apex All Codes

```
        address.state = STATE;
        address.country = COUNTRY;

        Test.setMock(
            HttpCalloutMock.class,
            new OpenStreetMapHttpCalloutMockImpl()
        );

        // WHEN
        List<GeocodingService.Coordinates> computedCoordinates =
GeocodingService.geocodeAddresses(
            new List<GeocodingService.GeocodingAddress>{ address }
        );

        // THEN
        System.assert(
            computedCoordinates.size() == 1,
            'Expected 1 pair of coordinates were returned'
        );
        System.assert(
            computedCoordinates[0].lat == LATITUDE,
            'Expected mock lat was returned'
        );
        System.assert(
            computedCoordinates[0].lon == LONGITUDE,
            'Expected mock lon was returned'
        );
    }
    @isTest
    static void blankAddress() {
        // GIVEN
        GeocodingService.GeocodingAddress address = new
GeocodingService.GeocodingAddress();

        Test.setMock(
            HttpCalloutMock.class,
            new OpenStreetMapHttpCalloutMockImpl()
```

```
        );

        // WHEN
        List<GeocodingService.Coordinates> computedCoordinates =
GeocodingService.geocodeAddresses(
            new List<GeocodingService.GeocodingAddress>{ address }
        );

        // THEN
        System.assert(
            computedCoordinates.size() == 1,
            'Expected 1 pair of coordinates were returned'
        );
        System.assert(
            computedCoordinates[0].lat == null,
            'Expected null lat was returned'
        );
        System.assert(
            computedCoordinates[0].lon == null,
            'Expected null lon was returned'
        );
    }
    @isTest
    static void errorResponse() {
        // GIVEN
        GeocodingService.GeocodingAddress address = new
GeocodingService.GeocodingAddress();
        address.street = STREET;
        address.city = CITY;
        address.postalcode = POSTAL_CODE;
        address.state = STATE;
        address.country = COUNTRY;

        Test.setMock(
            HttpCalloutMock.class,
            new OpenStreetMapHttpCalloutMockImplError()
        );
```

Apex All Codes

```
        // WHEN
        List<GeocodingService.Coordinates> computedCoordinates =
GeocodingService.geocodeAddresses(
            new List<GeocodingService.GeocodingAddress>{ address }
        );

        // THEN
        System.assert(
            computedCoordinates.size() == 1,
            'Expected 1 pair of coordinates were returned'
        );
        System.assert(
            computedCoordinates[0].lat == null,
            'Expected null lat was returned'
        );
        System.assert(
            computedCoordinates[0].lon == null,
            'Expected null lon was returned'
        );
    }

    public class OpenStreetMapHttpCalloutMockImpl implements HttpCalloutMock {
        public HTTPResponse respond(HTTPRequest req) {
            HttpResponse res = new HttpResponse();
            res.setHeader('Content-Type', 'application/json');
            res.setBody('[{"lat": ' + LATITUDE + ',"lon": ' + LONGITUDE + '}]');
            res.setStatusCode(200);
            return res;
        }
    }

    public class OpenStreetMapHttpCalloutMockImplError implements HttpCalloutMock {
        public HTTPResponse respond(HTTPRequest req) {
            HttpResponse res = new HttpResponse();
            res.setHeader('Content-Type', 'application/json');
            res.setStatusCode(400);
```

```
            return res;
        }
    }
}
```

ContactsTodayController:

```
public class ContactsTodayController {

    @AuraEnabled
    public static List<Contact> getContactsForToday() {

        List<Task> my_tasks = [SELECT Id, Subject, WhoId FROM Task WHERE OwnerId =
:UserInfo.getUserId() AND IsClosed = false AND WhoId != null];
        List<Event> my_events = [SELECT Id, Subject, WhoId FROM Event WHERE OwnerId
= :UserInfo.getUserId() AND StartDateTime >= :Date.today() AND WhoId != null];
        List<Case> my_cases = [SELECT ID, ContactId, Status, Subject FROM Case WHERE
OwnerId = :UserInfo.getUserId() AND IsClosed = false AND ContactId != null];

        Set<Id> contactIds = new Set<Id>();
        for(Task tsk : my_tasks) {
            contactIds.add(tsk.WhoId);
        }
        for(Event evt : my_events) {
            contactIds.add(evt.WhoId);
        }
        for(Case cse : my_cases) {
            contactIds.add(cse.ContactId);
        }

        List<Contact> contacts = [SELECT Id, Name, Phone, Description FROM Contact
WHERE Id IN :contactIds];

        for(Contact c : contacts) {
            c.Description = '';
            for(Task tsk : my_tasks) {
                if(tsk.WhoId == c.Id) {
```

Apex All Codes

```
                c.Description += 'Because of Task "'+tsk.Subject+'"\n';
            }
        }
        for(Event evt : my_events) {
            if(evt.WhoId == c.Id) {
                c.Description += 'Because of Event "'+evt.Subject+'"\n';
            }
        }
        for(Case cse : my_cases) {
            if(cse.ContactId == c.Id) {
                c.Description += 'Because of Case "'+cse.Subject+'"\n';
            }
        }
    }

    return contacts;
  }

}
```

ContactsTodayControllerTest:

```
@IsTest
public class ContactsTodayControllerTest {

  @IsTest
  public static void testGetContactsForToday() {

    Account acct = new Account(
      Name = 'Test Account'
    );
    insert acct;

    Contact c = new Contact(
      AccountId = acct.Id,
      FirstName = 'Test',
      LastName = 'Contact'
```

```
        );
        insert c;

        Task tsk = new Task(
            Subject = 'Test Task',
            WhoId = c.Id,
            Status = 'Not Started'
        );
        insert tsk;

        Event evt = new Event(
            Subject = 'Test Event',
            WhoId = c.Id,
            StartDateTime = Date.today().addDays(5),
            EndDateTime = Date.today().addDays(6)
        );
        insert evt;

        Case cse = new Case(
            Subject = 'Test Case',
            ContactId = c.Id
        );
        insert cse;

        List<Contact> contacts = ContactsTodayController.getContactsForToday();
        System.assertEquals(1, contacts.size());
        System.assert(contacts[0].Description.containsIgnoreCase(tsk.Subject));
        System.assert(contacts[0].Description.containsIgnoreCase(evt.Subject));
        System.assert(contacts[0].Description.containsIgnoreCase(cse.Subject));

    }

    @IsTest
    public static void testGetNoContactsForToday() {

        Account acct = new Account(
            Name = 'Test Account'
```

```
        );
        insert acct;

        Contact c = new Contact(
            AccountId = acct.Id,
            FirstName = 'Test',
            LastName = 'Contact'
        );
        insert c;

        Task tsk = new Task(
            Subject = 'Test Task',
            WhoId = c.Id,
            Status = 'Completed'
        );
        insert tsk;

        Event evt = new Event(
            Subject = 'Test Event',
            WhoId = c.Id,
            StartDateTime = Date.today().addDays(-6),
            EndDateTime = Date.today().addDays(-5)
        );
        insert evt;

        Case cse = new Case(
            Subject = 'Test Case',
            ContactId = c.Id,
            Status = 'Closed'
        );
        insert cse;

        List<Contact> contacts = ContactsTodayController.getContactsForToday();
        System.assertEquals(0, contacts.size());

    }
```

Apex All Codes

```
}
```

TestPropertyController:

```
@isTest
private class TestPropertyController {
    private final static String MOCK_PICTURE_NAME = 'MockPictureName';

    public static void createProperties(Integer amount) {
        List<Property__c> properties = new List<Property__c>();
        for (Integer i = 0; i < amount; i++) {
            properties.add(
                new Property__c(
                    Name = 'Name ' + i,
                    Price__c = 20000,
                    Beds__c = 3,
                    Baths__c = 3
                )
            );
        }
        insert properties;
    }
    static testMethod void testGetPagedPropertyList() {
        TestPropertyController.createProperties(5);
        Test.startTest();
        PagedResult result = PropertyController.getPagedPropertyList(
            '',
            999999,
            0,
            0,
            10,
            1
        );
        Test.stopTest();
        System.assertEquals(5, result.records.size());
    }

    static testMethod void testGetPicturesNoResults() {
```

Apex All Codes

```
        Property__c property = new Property__c(Name = 'Name');
        insert property;

        Test.startTest();
        List<ContentVersion> items = PropertyController.getPictures(
            property.Id
        );
        Test.stopTest();

        System.assertEquals(null, items);
    }

    static testMethod void testGetPicturesWithResults() {
        Property__c property = new Property__c(Name = 'Name');
        insert property;

        // Insert mock picture
        ContentVersion picture = new Contentversion();
        picture.Title = MOCK_PICTURE_NAME;
        picture.PathOnClient = 'picture.png';
        picture.Versiondata = EncodingUtil.base64Decode('MockValue');
        insert picture;

        // Link picture to property record
        List<ContentDocument> documents = [
            SELECT Id, Title, LatestPublishedVersionId
            FROM ContentDocument
            LIMIT 1
        ];
        ContentDocumentLink link = new ContentDocumentLink();
        link.LinkedEntityId = property.Id;
        link.ContentDocumentId = documents[0].Id;
        link.shareType = 'V';
        insert link;

        Test.startTest();
        List<ContentVersion> items = PropertyController.getPictures(
```

```
            property.Id
        );
        Test.stopTest();

        System.assertEquals(1, items.size());
        System.assertEquals(MOCK_PICTURE_NAME, items[0].Title);
    }
}
```

AccountAddressTrigger on Account :

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

}
```

PushNotificationTrigger on Property__c:

```
trigger PushNotificationTrigger on Property__c (after update) {


    for (Property__c property : Trigger.New) {

        if (property.Price__c != Trigger.oldMap.get(property.Id).Price__c) {
            Messaging.PushNotification msg = new Messaging.PushNotification();
            String text = property.Name + '. New Price: $' +
property.Price__c.setScale(0).format();
            Map<String, Object> payload = Messaging.PushNotificationPayload.apple(text, '',
null, null);
            msg.setPayload(payload);
            Set<String> users = new Set<String>();
            users.add(UserInfo.getUserId());
            msg.send('DreamHouzz', users);
```

```
        }


    }


}
```

RejectDuplicateFavorite on Favorite__c:

```
trigger RejectDuplicateFavorite on Favorite__c (before insert) {

    // NOTE: this trigger needs to be bulkified

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

}
```