## #CreateDefaultData

```
public with sharing class CreateDefaultData{
  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, 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>();
      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');

  }
}
```

## #MaintenanceRequestHelper

```
public with sharing class MaintenanceRequestHelper {
  public static void updateworkOrders(List<Case> updWorkOrders,
Map<Id,Case> nonUpdCaseMap) {
    Set<Id> validIds = new Set<Id>();


    For (Case c : updWorkOrders){
      if (nonUpdCaseMap.get(c.Id).Status != 'Closed' && c.Status == 'Closed'){
            if (c.Type == 'Repair' || c.Type == 'Routine Maintenance'){
                validIds.add(c.Id);


            }
        }
    }

    if (!validIds.isEmpty()){
        List<Case> newCases = new List<Case>();
        Map<Id,Case> closedCasesM = new Map<Id,Case>([SELECT Id, Vehicle__c,
Equipment__c, Equipment__r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c FROM Equipment_Maintenance_Items__r)
                                FROM Case WHERE Id IN :validIds]);
        Map<Id,Decimal> maintenanceCycles = new Map<ID,Decimal>();
        AggregateResult[] results = [SELECT Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE Maintenance_Request__c IN
:ValidIds GROUP BY Maintenance_Request__c];

    for (AggregateResult ar : results){
        maintenanceCycles.put((Id) ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
    }

        for(Case cc : closedCasesM.values()){
            Case nc = new Case (
                ParentId = cc.Id,
            Status = 'New',
                Subject = 'Routine Maintenance',
                Type = 'Routine Maintenance',
                Vehicle__c = cc.Vehicle__c,
            Equipment__c =cc.Equipment__c,
            Origin = 'Web',
            Date_Reported__c = Date.Today()

          );

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

          newCases.add(nc);
        }

       insert newCases;

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

          }
        }
        insert ClonedWPs;
      }
   }
}
```

## #MaintenanceRequestHelperTest

```
@istest
public with sharing class MaintenanceRequestHelperTest {

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

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

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

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

   PRIVATE STATIC Equipment_Maintenance_Item__c createWorkPart(id
equipmentId,id requestId){
     Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c = equipmentId,
                                              Maintenance_Request__c = requestId);
     return wp;
  }
  @istest
   private static void testMaintenanceRequestPositive(){
     Vehicle__c vehicle = createVehicle();
     insert vehicle;
     id vehicleId = vehicle.Id;

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

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

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

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

```
      test.stopTest();

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

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

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

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

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

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

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

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

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

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

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

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

```
    }
    insert vehicleList;
    insert equipmentList;

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

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

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

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

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

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

## #WareHouseCalloutServiceclass

```
public with sharing class WarehouseCalloutService {

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

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

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

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


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

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

          for (Object eq : jsonResponse){
              Map<String,Object> mapJson = (Map<String,Object>)eq;
              Product2 myEq = new Product2();
```

```
            myEq.Replacement_Part__c = (Boolean) mapJson.get('replacement');
            myEq.Name = (String) mapJson.get('name');
            myEq.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
            myEq.Lifespan_Months__c = (Integer) mapJson.get('lifespan');
            myEq.Cost__c = (Decimal) mapJson.get('lifespan');
            myEq.Warehouse_SKU__c = (String) mapJson.get('sku');
            myEq.Current_Inventory__c = (Double) mapJson.get('quantity');
            warehouseEq.add(myEq);
          }

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

        }
    }
}

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

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

        // Create a fake response
        HttpResponse response = new HttpResponse();
```

```
      response.setHeader('Content-Type', 'application/json');

response.setBody('[{"_id":"55d66226726b611100aaf741","replacement":false,"quan
tity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"sku":"100003"}]');
      response.setStatusCode(200);
      return response;
   }
}
```

## #WareHouseCalloutServiceTest

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

## #WareHouseSyncSheduleclass

```
global class WarehouseSyncSchedule implements Schedulable{
     global void execute(SchedulableContext ctx){
       WarehouseCalloutService.runWarehouseEquipmentSync();
   }  }
```

## #WareHouseSyncSheduleTestclass

```
@isTest
public class WarehouseSyncScheduleTest {

  @isTest static void WarehousescheduleTest(){
     String scheduleTime = '00 00 01 * * ?';
     Test.startTest();
     Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
     String jobID=System.schedule('Warehouse Time To Schedule to Test',
scheduleTime, new WarehouseSyncSchedule());
     Test.stopTest();
     //Contains schedule information for a scheduled job. CronTrigger is similar
to a cron job on UNIX systems.
     // This object is available in API version 17.0 and later.
     CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime >
today];
     System.assertEquals(jobID, a.Id,'Schedule ');
 }
}
```

## #MaintenanceRequestclass

```
trigger MaintenanceRequest on Case (before update, after update) {
   if(Trigger.isUpdate && Trigger.isAfter){

      MaintenanceRequestHelper.updateWorkOrders(Trigger.New,
Trigger.OldMap);

   }  }
```

## #HandlerSOQLclass

```
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';
            }
            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));

  }   }
```

## #HandlerTopOpportunitiesClass

```
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];

      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));     }  }
```

## #BotControllerClass

```
public with sharing class BotController {

   class HandlerMapping {

      public String handlerClassName;
      public Pattern utterancePattern;
```

```
    public HandlerMapping(String handlerClassName, String patternStr) {
      this.handlerClassName = handlerClassName;
      this.utterancePattern = Pattern.compile(patternStr);
    }    }
```

## #BotFieldClass

```
public class BotField {

  @AuraEnabled public String name { get;set; }
  @AuraEnabled public String value { get;set; }
  @AuraEnabled public String linkURL { get;set; }

  public BotField(String name, String value) {
    this.name = name;
    this.value = value;
  }

  public BotField(String name, String value, string linkURL) {
    this.name = name;
    this.value = value;
    this.linkURL = linkURL;
  }

}
```

## #TestVerifyDate ApexClass

```
@IsTest
public class TestVerifyDate {
  @IsTest  static void Test_CheckDates_case1(){
      Date D
=VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('01/05/2020'));
      System.assertEquals(date.parse('01/05/2020'), D);

   }
     @IsTest static void Test_CheckDates_case2(){
      Date D
=VerifyDate.CheckDates(date.parse('01/01/2020'),date.parse('05/05/2020'));
      System.assertEquals(date.parse('01/31/2020'), D);

      }
  @IsTest static void Test_DateWithin30Days_case1(){
      Boolean
flag=VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('12/30/20
19'));
      System.assertEquals(false,flag);

   }
    @IsTest static void Test_DateWithin30Days_case2(){
      Boolean
flag=VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('02/02/2
019'));
      System.assertEquals(false,flag);
    }
    @IsTest  static void Test_DateWithin30Days_case3(){
      Boolean
flag=VerifyDate.DateWithin30Days(date.parse('01/01/2020'),date.parse('02/02/2
020'));
      System.assertEquals(true,flag);
     }
    @IsTest static void Test_SetEndOfMonthDate(){
      Date returndate= VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));

   }

}
```

## #AccountManager ApexClass

```
@RestResource(urlMapping = '/Accounts/*/contacts')
global with sharing class AccountManager {
   @HttpGet
   global static Account getAccount(){
      RestRequest request = RestContext.request;
      string accountId =
request.requestURI.substringBetween('Accounts/','/contacts');
      Account result = [SELECT Id,Name,(Select Id,name from Contacts) from
Account where Id=:accountId Limit 1];
      return result;
   }

}
```

## #DialyLeadProcessor

```
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){
      l.LeadSource = 'Dreamforce';
      update l;

    }    }      }
```

## #LeadProcessorTest_ApexClass

```
@isTest
public class LeadProcessorTest {

  @isTest
  public static void testit(){
    List<lead> L_list =new List<lead>();

    for(Integer i=0; i< 200; i++){
      Lead L= new lead();
      L.LastName ='name' + i;
      L.Company = 'Company';
      L.Status = 'Random Status';
      L_list.add(L);
      }
    insert L_list;

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

  }      }
```

## #LeadProcessor ApexClass

```
global class LeadProcessor implements Database.Batchable<sObject> {
   global Integer count = 0;
   global Database.QueryLocator start(Database.BatchableContext bc){
      return Database.getQueryLocator('SELECT ID,LeadSource FROM Lead');

   }
   global void execute(Database.BatchableContext bc,List<Lead> L_list){
      List<Lead> L_list_new=new List<Lead>();

      for(Lead L:L_list){
         L.leadSource = 'Dreamforce';
         L_list_new.add(L);
         count += 1;

      }
      update L_list_new;

   }
   global void finish(Database.BatchableContext bc){
      system.debug('count =' +count);
   }
}
```

## #TestRestrictContactByName

```
@isTest
public class TestRestrictContactByName {
   @isTest static void Test_insertupdateContact(){
        Contact cnt=new Contact();
        cnt.LastName = 'INVALIDNAME';
        Test.startTest();
        Database.SaveResult result = Database.insert(cnt,false);


       Test.stopTest();
       system.assert(!result.isSuccess());
       system.assert(result.getErrors().size()>0);
       system.assertEquals('The Last Name "INVALIDNAME"  is not allowed for
DML',result.getErrors()[0].getMessage());

    }
}
```

## #AccountProcessorTest

```
@isTest
public class AccountProcessorTest {
  @isTest
    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();       }       }
```

## #RandomContactFactory_ ApexClass

```
public class RandomContactFactory {
   public static List<Contact> generateRandomContacts(Integer numcnt,string
lastname){
      List<Contact> contacts = new List<Contact>();
      for(Integer i=0;i<numcnt;i++){
         Contact cnt =new Contact(FirstName = 'Test'+i,LastName=lastname);
         contacts.add(cnt);

      }
      return contacts;

   }
}
```

## #AccountManagerTest ApexClass

```
@isTest
private class AccountManagerTest {
   @isTest static void testGetContactsByAccountId(){
      Id recordId = createTestRecord();
      RestRequest request = new RestRequest();
      request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/Accounts/' +
recordId +'/contacts';

      request.httpMethod = 'GET';
      RestContext.request = request;
      Account thisAccount = AccountManager.getAccount();
      System.assert(thisAccount != null);
      System.assertEquals('Test record',thisAccount.Name);

   }
   static Id createTestRecord(){
      Account accountTest =  new Account(
          Name = 'Test record');
      insert accountTest;

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

         return accountTest.Id;

      }
   }
```

## _#ParkLocatorTest_ApexClass

```
 @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);
   }       }
```

## #ParkLocator_ApexClass

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

   }

}
```

## #AddPrimaryContact_ApexClass

```
public class AddPrimaryContact implements Queueable{
   private Contact con;
   private String state;

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

   }
   public void execute(QueueableContext context){
      List<Account> accounts =[Select Id,Name,(Select FirstName,LastName,Id
from contacts) from Account where BillingState = :state Limit 200];
      List<Contact> primaryContacts = new List<Contact>();

      for(Account acc:accounts){
         Contact c = con.clone();
         c.AccountId= acc.Id;
         primaryContacts.add(c);

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

}
```

## #AddPrimaryContactTest ApexClass

```
@isTest
public class AddPrimaryContactTest {
  static testmethod void testQueueable(){
    List<Account> testAccounts = new List<Account>();
    for(Integer i=0;i<50;i++){
      testAccounts.add(new Account(Name='Account' +i,BillingState='CA'));
    }
    for(Integer j=0;j<=50;j++){
      testAccounts.add(new Account(Name='Account'+j,BillingState='NY'));

    }
    insert testAccounts;

    Contact testContact = new Contact(FirstName = 'John',LastName='Doe');
    insert testContact;

    AddPrimaryContact addit = new addPrimaryContact(testContact,'CA');

    Test.startTest();
    system.enqueueJob(addit);
    Test.stopTest();

    System.assertEquals(50,[Select count() from Contact where accountId in
(Select Id from Account where BillingState='CA')]);

    }
  }
```

## #ParkServiceMock  ApexClass

```
@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 responseNam,
   String responseType
      )
   {
   ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
   response_x.return_x = new List<String>{'Yellowstone','Mackinac National
Park','Yosemite'};
   response.put('response_x',response_x);
   }
}
```

## #AsyncParkService  Apex Class

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

## #ParkService ApexClass

```
public class ParkService {
   public class byCountryResponse {
      public String[] return_x;
      private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};
      private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
      private String[] field_order_type_info = new String[]{'return_x'};
   }
   public class byCountry {
      public String arg0;
      private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};
      private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};
      private String[] field_order_type_info = new String[]{'arg0'};
   }
   public class ParksImplPort {
      public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';
      public Map<String,String> inputHttpHeaders_x;
      public Map<String,String> outputHttpHeaders_x;
      public String clientCertName_x;
      public String clientCert_x;
      public String clientCertPasswd_x;
      public Integer timeout_x;
      private String[] ns_map_type_info = new String[]{'http://parks.services/',
'ParkService'};
```

```
    public String[] byCountry(String arg0) {
      ParkService.byCountry request_x = new ParkService.byCountry();
      request_x.arg0 = arg0;
      ParkService.byCountryResponse response_x;
      Map<String, ParkService.byCountryResponse> response_map_x = new
Map<String, ParkService.byCountryResponse>();
      response_map_x.put('response_x', response_x);
      WebServiceCallout.invoke(
       this,
       request_x,
       response_map_x,
       new String[]{endpoint_x,
       '',
       'http://parks.services/',
       'byCountry',
       'http://parks.services/',
       'byCountryResponse',
       'ParkService.byCountryResponse'}
      );
      response_x = response_map_x.get('response_x');
      return response_x.return_x;
    }
  }
}
```

## #AnimalLocator_ApexClass

```
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');
  }

}
```

## #AnimalLocatorTest ApexClass

```
@isTest
private class AnimalLocatorTest {
  @isTest static void AnimalLocatorMock1(){
    try{
      Test.setMock(HttpCalloutMock.class , new AnimalLocatorMock());

      string result = AnimalLocator.getAnimalNameById(1);
      String expectedResult  = 'fox';
      System.assertEquals(result,expectedResult);

    }
    catch(exception e){
     System.debug('The following exception has occurred: ' + e.getMessage()); }}}
```

## #AnimallocatorMock Apex Class

```
 @isTest
global  class AnimalLocatorMock implements HttpCalloutMock {
   global HTTPResponse respond(HTTPRequest request){
   HttpResponse response = new HttpResponse();
   response.setHeader('Content-Type', 'application/json');
   response.setBody('{"animals":["lion","fox","bear","panda","snake","raccoon"]}');
   response.setStatusCode(200);
   return response;
   }
}
```

## #DialyLeadProcessorTest  Apex Class

```
@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 ,
          Status='Open - Not Contacted',
          Company ='Test1 Inc'));

      }
```

## #ContactsTodayController ApexClass

```
 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) {
             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 ApexClass

```
@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());
}      }
```

## #HandlerFindAccount ApexClass

```
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));

  }

}
```

## #HandlerEmployeeid Apex Class

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

## #HandlerFileUpload ApexClass

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

-

## #HandlerCostCenter Apex Class

```
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 Apex Class

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

   }
```

}
# EinsteinVisionControllerTest  ApexClass

```
@isTest
public class EinsteinVisionControllerTest {

  static testMethod void testPredict() {
     insert new Dreamhouse_Settings__c(Einstein_Vision_Email__c =
'user@host.com');
     Boolean success = true;
     try {
       ContentVersion cv = new ContentVersion(Title='einstein_platform',
PathOnClient='/', VersionData=Blob.valueof('some key'));
       insert cv;
            EinsteinVisionController.predict('victorian.jpg', '', 'theModelId');
            EinsteinVisionController.predict('victorian_01.jpg', '', '');
     } catch (Exception e) {
       success = false;
     } finally {
            System.assert(success);
     }
  }

  static testMethod void testGetDataSets() {
     insert new Dreamhouse_Settings__c(Einstein_Vision_Email__c =
'user@host.com');
     Boolean success = true;
     try {
       ContentVersion cv = new ContentVersion(Title='einstein_platform',
PathOnClient='/', VersionData=Blob.valueof('some key'));
       insert cv;
```

```
            EinsteinVisionController.getDataSets();
      } catch (Exception e) {
        System.debug(e);
        success = false;
      } finally {
            System.assert(success);
      }
    }

    static testMethod void testGetModelByDataset() {
      insert new Dreamhouse_Settings__c(Einstein_Vision_Email__c =
'user@host.com');
      Boolean success = true;
      try {
        ContentVersion cv = new ContentVersion(Title='einstein_platform',
PathOnClient='/', VersionData=Blob.valueof('some key'));
        insert cv;
            EinsteinVisionController.getModelsByDataset(101);
      } catch (Exception e) {
        success = false;
      } finally {
            System.assert(success);
      }
    }

    static testMethod void testDeleteDataset() {
      insert new Dreamhouse_Settings__c(Einstein_Vision_Email__c =
'user@host.com');
      Boolean success = true;
      try {
        ContentVersion cv = new ContentVersion(Title='einstein_platform',
PathOnClient='/', VersionData=Blob.valueof('some key'));
```

```
        insert cv;
        EinsteinVisionController.deleteDataset(101);
      } catch (Exception e) {
        success = false;
      } finally {
              System.assert(success);
      }
    }

    static testMethod void testCreateDataset() {
      insert new Dreamhouse_Settings__c(Einstein_Vision_Email__c =
'user@host.com');
      Boolean success = true;
      try {
        ContentVersion cv = new ContentVersion(Title='einstein_platform',
PathOnClient='/', VersionData=Blob.valueof('some key'));
        insert cv;
              EinsteinVisionController.createDataset('path/to/zip');
      } catch (Exception e) {
        success = false;
      } finally {
              System.assert(success);
      }
    }

    static testMethod void testTrainModel() {
      insert new Dreamhouse_Settings__c(Einstein_Vision_Email__c =
'user@host.com');
      Boolean success = true;
      try {
        ContentVersion cv = new ContentVersion(Title='einstein_platform',
PathOnClient='/', VersionData=Blob.valueof('some key'));
```

```
        insert cv;
                EinsteinVisionController.trainModel('theModelId', 101);
      } catch (Exception e) {
        success = false;
      } finally {
                System.assert(success);
      }
   }

   static testMethod void JTWIssue() {
      Boolean success = true;
      try {
        JWT jwt = new JWT('RS256');
        jwt.pkcs8 = 'some key';
        jwt.iss = 'developer.force.com';
        jwt.sub = 'user@server.com';
        jwt.aud = 'https://api.metamind.io/v1/oauth2/token';
        jwt.exp = '3600';
        try {
          String token = jwt.issue();
        } catch (Exception e1) {

        }
      } catch (Exception e2) {
        success = false;
      } finally {
        System.assert(success);
      }
   }

}
```

## #DreamHouseSampleDataController

```
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];
  }
}
```

## #BotField ApexClass

```
public class BotField {

  @AuraEnabled public String name { get;set; }
  @AuraEnabled public String value { get;set; }
  @AuraEnabled public String linkURL { get;set; }

  public BotField(String name, String value) {
    this.name = name;
    this.value = value;
  }

  public BotField(String name, String value, string linkURL) {
    this.name = name;
    this.value = value;
    this.linkURL = linkURL;
  }
}
```

#HandlerTravelApproval Apex Class

```
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);
      } 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));

  }


}
```

## #JWTBearerFlow

```
public class JWTBearerFlow {

  public static String getAccessToken(String tokenEndpoint, JWT jwt) {

      String access_token = null;
      String body = 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-
type%3Ajwt-bearer&assertion=' + jwt.issue();
      HttpRequest req = new HttpRequest();
      req.setMethod('POST');
      req.setEndpoint(tokenEndpoint);
      req.setHeader('Content-type', 'application/x-www-form-urlencoded');
      req.setBody(body);
      Http http = new Http();
      HTTPResponse res = http.send(req);

      if ( res.getStatusCode() == 200 ) {
         System.JSONParser parser = System.JSON.createParser(res.getBody());
         while (parser.nextToken() != null) {
            if ((parser.getCurrentToken() == JSONToken.FIELD_NAME) &&
(parser.getText() == 'access_token')) {
               parser.nextToken();
               access_token = parser.getText();
               break;
            }
         }
      }
      return access_token;

  }      }
```

## #BotRecord

```
public class BotRecord {

   @AuraEnabled
   public List<BotField> fields { get;set; }

   public BotRecord(List<BotField> fields) {
      this.fields = fields;
   }

}
```

## #trigger PushNotificationTrigger

```
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);
      }
```

## #trigger RejectDuplicateFavorite

```
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');
   }  }
```

## #trigger ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (before 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;
   }
}
```

## #trigger AccountAddressTrigger

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

## # trigger RestrictContactByName

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

         }



}
```