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

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

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

trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
   List<Task> taskList = new List<Task>();
   for(Opportunity opp : [SELECT Id, StageName FROM Opportunity WHERE
StageName='Closed Won' AND Id IN : Trigger.New]){
      taskList.add(new Task(Subject='Follow Up Test Task', WhatId = opp.Id));
   }

   if(taskList.size()>0){
      insert tasklist;
```

```
    }
}

@isTest
global class AnimalLocatorMock implements HttpCalloutMock {
    global HTTPResponse respond(HTTPRequest request) {
        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;
    }
}

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;
        }
    }
}

@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());
        }
}

@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();
    }
}

@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();
    }
}

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;
    }
}

@isTest
private class AccountManagerTest {
    private static testMethod void getAccountTest1() {
        Id id = createTestRecord();
        RestRequest request = new RestRequest();
        request.requestUri =
'https://na1.salesforce.com/services/apexrest/Accounts/'+id+'/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 TestAcc = new Account(Name='Test record');
        insert TestAcc;
        Contact TestCon= new Contact(LastName='Test', AccountId = TestAcc.id);
        return TestAcc.Id;
    }
}

@isTest
public class TestVerifyDate {
    static testMethod void testMethod1() {
        Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
        Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
    }
}
```

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

@isTest
private class TestRestrictContactByName {
   static testMethod void  metodoTest()
   {
      List<Contact> listContact= new List<Contact>();
      Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' ,
email='Test@test.com');
      Contact c2 = new Contact(FirstName='Francesco1', LastName =
'INVALIDNAME',email='Test@test.com');
      listContact.add(c1);
      listContact.add(c2);
      Test.startTest();
         try
         {
            insert listContact;
         }
         catch(Exception ee) { }
      Test.stopTest();
   }
}

public with sharing class SlackOpportunityPublisher {

   private static final String slackURL =
Dreamhouse_Settings__c.getOrgDefaults().Slack_Opportunity_Webhook_URL__c;

   @InvocableMethod(label='Post to Slack')
   public static void postToSlack(List<Id> opportunityId) {
      Id oppId = opportunityId[0]; // If bulk, only post first to avoid overloading Slack
channel
```

```
      Opportunity opportunity = [SELECT Name, StageName from Opportunity WHERE
Id=:oppId];
                  Map<String,Object> msg = new Map<String,Object>();
                  msg.put('text', 'The following opportunity has changed:\n' +
opportunity.Name + '\nNew Stage: *'
            + opportunity.StageName + '*');
                  msg.put('mrkdwn', true);
      String body = JSON.serialize(msg);
      System.enqueueJob(new QueueableSlackCall(slackURL, 'POST', body));
   }

   public class QueueableSlackCall implements System.Queueable,
Database.AllowsCallouts {
      private final String url;
      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);
         }
      }
   }
}
```

```
@isTest
public class SlackOpportunityPublisherTest {

   static testMethod void testPost() {
      Boolean success = true;
      try {
         Opportunity opp = new Opportunity(Name='test opportunity', StageName='Close
Won', CloseDate=date.today());
         insert opp;
               SlackOpportunityPublisher.postToSlack(new List<Id> { opp.Id });
      } catch (Exception e) {
         success = false;
      } finally {
               System.assert(success);
      }
   }
}

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');
   }
}

@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);
    }
}

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);
        }

    }

    static List<HandlerMapping> handlerMappings;

    static {
        List<Bot_Command__c> commands = [SELECT apex_class__c, pattern__c FROM
Bot_Command__c WHERE Active__c = True ORDER BY Name];
        System.debug(commands);
        List<HandlerMapping> mappings = new List<HandlerMapping>();
        for (Bot_Command__c command : commands) {
                            mappings.add(new HandlerMapping(command.apex_class__c,
command.pattern__c));
        }
        handlerMappings = mappings;
    }
```

```
    @AuraEnabled
    public static BotResponse submit(String utterance, Map<String, String> session,
String fileName, String fileContent) {

        try {

            if (session != null) {
                String nextCommand = session.get('nextCommand');
                if (nextCommand != null) {
                    Type t = Type.forName('', nextCommand);
                    BotHandler h = (BotHandler)t.newInstance();
                    return h.handle(utterance, null, session, fileName, fileContent);
                }
            }

            for (HandlerMapping mapping : BotController.handlerMappings) {
                Matcher utteranceMatcher = mapping.utterancePattern.matcher(utterance);
                if (utteranceMatcher.matches()) {
                    Type t = Type.forName('', mapping.handlerClassName);
                    BotHandler h = (BotHandler)t.newInstance();
                    List<String> params = new List<String>();
                    for (Integer i=1; i<=utteranceMatcher.groupCount(); i=i+1) {
                        params.add(utteranceMatcher.group(i).trim());
                    }
                    return h.handle(utterance, params, session, fileName, fileContent);
                }
            }
            return new BotResponse(new BotMessage('Bot', 'I don\'t know how to answer
that'));

        } catch (Exception e) {
            System.debug(e);
            return new BotResponse(new BotMessage('Bot', 'Oops, something went wrong
invoking that command'));
        }
    }
}
```

```
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;
   }
}

public interface BotHandler {

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

}

public class BotItem {

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

   public BotItem(String name) {
      this.name = name;
   }

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

```
}

public virtual class BotMessage {

    @AuraEnabled public String author { get;set; }
    @AuraEnabled public String messageText { get;set; }
    @AuraEnabled public List<BotRecord> records { get;set; }
    @AuraEnabled public List<BotItem> items { get;set; }
    @AuraEnabled public List<BotMessageButton> buttons { get;set; }
    @AuraEnabled public String imageURL { get;set; }

    public BotMessage() {
    }

    public BotMessage(String author, String messageText) {
        this.author = author;
        this.messageText = messageText;
    }

    public BotMessage(String author, String messageText, List<BotRecord> records) {
        this.author = author;
        this.messageText = messageText;
        this.records = records;
    }

    public BotMessage(String author, String messageText, List<BotItem> items) {
        this.author = author;
        this.messageText = messageText;
        this.items = items;
    }

    public BotMessage(String author, String messageText, List<BotMessageButton>
buttons) {
        this.author = author;
        this.messageText = messageText;
        this.buttons = buttons;
    }
```

```
    public BotMessage(String author, String messageText, String imageURL) {
        this.author = author;
        this.messageText = messageText;
        this.imageURL = imageURL;
    }

}

public class BotMessageButton {

    @AuraEnabled public String label { get;set; }
    @AuraEnabled public String value { get;set; }

    public BotMessageButton(String label, String value) {
        this.label = label;
        this.value = value;
    }

}

public class BotRecord {

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

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

}

public class BotResponse {

    @AuraEnabled public List<BotMessage> messages { get; set; }
    @AuraEnabled public Map<String, String> session { get; set; }

    public BotResponse() {
```

```
    }

    public BotResponse(BotMessage[] messages) {
        this.messages = messages;
    }

    public BotResponse(List<BotMessage> messages, Map<String, String> session) {
        this.messages = messages;
        this.session = session;
    }

    public BotResponse(BotMessage message) {
        this.messages = new BotMessage[]{message};
    }

    public BotResponse(BotMessage message, Map<String, String> session) {
        this.messages = new BotMessage[]{message};
        this.session = session;
    }

}

@isTest
public class BotTest {

    static testMethod void testBotController() {
                    Bot_Command__c bc = new Bot_Command__c(Sample_Utterance__c='help
lightning', apex_class__c='HandlerHelpTopic', pattern__c='help (.*)');
        insert bc;
        BotResponse response = BotController.submit('help lightning', null, null, null);
        Map<String, String> session = response.session;
        response = BotController.submit('Developer', session, null, null);
        System.assert(response.messages[0].items.size() > 0);
    }

    static testMethod void testHello() {
        BotHandler handler = new HandlerHello();
        BotResponse response = handler.handle('', null, null, null, null);
```

```
    System.assert(response.messages[0].messageText == 'Hi there!');
}

static testMethod void testAddTwoNumbers() {
    BotHandler handler = new HandlerAddTwoNumbers();
    BotResponse response = handler.handle('', null, null, null, null);
    Map<String, String> session = response.session;
    response = handler.handle('1', null, session, null, null);
    session = response.session;
    response = handler.handle('2', null, session, null, null);
    System.assert(response.messages[0].messageText == '1 + 2 = 3');
}

static testMethod void testCostCenter() {
    BotHandler handler = new HandlerCostCenter();
    BotResponse response = handler.handle('', null, null, null, null);
    System.assert(response.messages[0].messageText == 'Your cost center is 21852');
}

static testMethod void testEmployeeId() {
    BotHandler handler = new HandlerEmployeeId();
    BotResponse response = handler.handle('', null, null, null, null);
    System.assert(response.messages[0].messageText == 'Your employee id is 9854');
}

static testMethod void testFindAccount() {
                Account a = new Account(Name='TestAccount');
                insert a;
    BotHandler handler = new HandlerFindAccount();
    BotResponse response = handler.handle('', new String[]{'Test'}, null, null, null);
    System.assert(response.messages[0].records.size() == 1);
}

static testMethod void testFindContact() {
                Contact c = new Contact(LastName='TestContact');
    insert c;
    BotHandler handler = new HandlerFindContact();
    BotResponse response = handler.handle('', new String[]{'Test'}, null, null, null);
```

```
        System.assert(response.messages[0].records.size() == 1);
    }

        static testMethod void testHelp() {
                Bot_Command__c bc = new
Bot_Command__c(Sample_Utterance__c='Hello', apex_class__c='HelloHandler',
pattern__c='Hello');
        insert bc;
        BotHandler handler = new HandlerHelp();
        BotResponse response = handler.handle('', null, null, null, null);
        System.assert(response.messages[0].items.size() == 1);
    }

        static testMethod void testHelpTopic() {
        BotHandler handler = new HandlerHelpTopic();
        BotResponse response = handler.handle('', null, null, null, null);
        Map<String, String> session = response.session;
                handler.handle('User', null, session, null, null);

        response = handler.handle('', null, null, null, null);
        session = response.session;
                response = handler.handle('Admin', null, session, null, null);

        response = handler.handle('', null, null, null, null);
        session = response.session;
                response = handler.handle('Developer', null, session, null, null);

        System.assert(response.messages[0].items.size() > 0);
    }

        static testMethod void testMyOpenCases() {
                Case c = new Case(Subject='TestCase');
                insert c;
        BotHandler handler = new HandlerMyOpenCases();
        BotResponse response = handler.handle('', null, null, null, null);
        System.assert(response.messages[0].records.size() == 1);
    }
```

```
        static testMethod void testTopOpportunities() {
                Account a = new Account(Name='TestAccount');
                insert a;
                Opportunity o = new Opportunity(Name='TestOpportunity', AccountId=a.id,
StageName='Prospecting', CloseDate=System.today().addMonths(1));
                insert o;
    BotHandler handler = new HandlerTopOpportunities();
    BotResponse response = handler.handle('', new String[]{'3'}, null, null, null);
    System.assert(response.messages[0].records.size() == 1);
  }

        static testMethod void testTravelApproval() {
    BotHandler handler = new HandlerTravelApproval();
    BotResponse response = handler.handle('', null, null, null, null);
    Map<String, String> session = response.session;
                handler.handle('Boston', null, session, null, null);
                handler.handle('Customer Facing', null, session, null, null);
                handler.handle('02/23/2017', null, session, null, null);
                handler.handle('1000', null, session, null, null);
                handler.handle('1000', null, session, null, null);
    System.assert(response.messages[0].messageText.length() > 0);
  }

        static testMethod void testPipeline() {
    BotHandler handler = new HandlerPipeline();
    BotResponse response = handler.handle('', null, null, null, null);
    System.assert(response.messages[0].imageURL != null);
  }

        static testMethod void testQuarter() {
    BotHandler handler = new HandlerQuarter();
    BotResponse response = handler.handle('', null, null, null, null);
    System.assert(response.messages[0].imageURL != null);
  }

  static testMethod void testNext() {
                Account a = new Account(Name='TestAccount');
```

```
                insert a;
                Opportunity o = new Opportunity(Name='TestOpportunity', AccountId=a.id,
StageName='Prospecting', CloseDate=System.today().addMonths(1));
                insert o;
                Case c = new Case(Subject='TestCase', Priority='High');
                insert c;
        BotHandler handler = new HandlerNext();
        BotResponse response = handler.handle('', null, null, null, null);
        System.assert(response.messages.size() > 1);
    }

    static testMethod void testSOQL() {
                Account a = new Account(Name='TestAccount');
                insert a;
        BotHandler handler = new HandlerSOQL();
        BotResponse response = handler.handle('select id from account', null, null, null,
null);
        System.assert(response.messages[0].records.size() == 1);
    }

    static testMethod void testFindPropertiesByBedrooms() {
        Property__c p = new Property__c(Name='TestProperty', Beds__c=3,
City__c='Boston');
        insert p;
        BotHandler handler = new HandlerFindPropertiesByBedrooms();
        BotResponse response = handler.handle('', new String[]{'3', 'Boston'}, null, null, null);
        System.assert(response.messages[0].records.size() == 1);
    }

    static testMethod void testFindProperties() {
        Property__c p = new Property__c(Name='TestProperty', Price__c=450000,
City__c='Boston');
        insert p;
        BotHandler handler = new HandlerFindProperties();
        Map<String, String> session = handler.handle('', null, null, null, null).session;
        session = handler.handle('Boston', null, session, null, null).session;
        session = handler.handle('Single Family', null, session, null, null).session;
```

```
        session = handler.handle('400000', null, session, null, null).session;
        BotResponse response = handler.handle('500000', null, session, null, null);
        System.assert(response.messages[0].records.size() == 1);
    }

}

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];
    }

}

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);
      }
   }
}


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

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

@isTest
global class ParkServiceMock implements WebServiceMock {
   global void doInvoke(Object stub, Object request, Map<String, Object> response, String
endpoint, String soapAction, String requestName, String responseNS, String
responseName, String responseType) {
      ParkService.byCountryResponse response_x = new
ParkService.byCountryResponse();
      response_x.return_x = new List<String>{'Yellowstone', 'Mackinac National Park',
'Yosemite'};
```

```
        response.put('response_x', response_x);
    }
}

//Generated by wsdl2apex

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

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

@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);
    }
}

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

```
        return parkSvc.byCountry(theCountry);
    }
}

@isTest
public class LIFXControllerTest {

    static testMethod void testGetLights() {
        Boolean success = true;
        try {
                LIFXController.getLights();
        } catch (Exception e) {
            success = false;
        } finally {
                System.assert(success);
        }
    }

    static testMethod void testSetPower() {
        Boolean success = true;
        try {
                LIFXController.setPower('1', true);
        } catch (Exception e) {
            success = false;
        } finally {
                System.assert(success);
        }
    }

    static testMethod void testSetBrightness() {
        Boolean success = true;
        try {
                LIFXController.setBrightness('1', 1);
        } catch (Exception e) {
            success = false;
        } finally {
                System.assert(success);
```

```
    }
  }
}

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();
    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() + '"}';
        }
    }

}

@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();
    }

}
```

```
public class LeadProcessor implements Database.Batchable<sObject> {
    public Database.QueryLocator start(Database.BatchableContext bc) {
        return Database.getQueryLocator([Select LeadSource From Lead ]);
   }
   public void execute(Database.BatchableContext bc, List<Lead> leads){
         for (Lead Lead : leads) {
            lead.LeadSource = 'Dreamforce';
         }
      update leads;
   }
   public void finish(Database.BatchableContext bc){
    }
}

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;

    }

}

public class JWT {

    public String alg {get;set;}
    public String iss {get;set;}
    public String sub {get;set;}
    public String aud {get;set;}
    public String exp {get;set;}
    public String iat {get;set;}
    public Map<String,String> claims {get;set;}
    public Integer validFor {get;set;}
    public String cert {get;set;}
    public String pkcs8 {get;set;}
    public String privateKey {get;set;}


    public static final String HS256 = 'HS256';
    public static final String RS256 = 'RS256';
    public static final String NONE = 'none';


    public JWT(String alg) {
        this.alg = alg;
        this.validFor = 300;
    }


    public String issue() {

        String jwt = '';
```

```
JSONGenerator header = JSON.createGenerator(false);
header.writeStartObject();
header.writeStringField('alg', this.alg);
header.writeEndObject();
String encodedHeader = base64URLencode(Blob.valueOf(header.getAsString()));

JSONGenerator body = JSON.createGenerator(false);
body.writeStartObject();
body.writeStringField('iss', this.iss);
body.writeStringField('sub', this.sub);
body.writeStringField('aud', this.aud);
Long rightNow = (dateTime.now().getTime()/1000)+1;
body.writeNumberField('iat', rightNow);
body.writeNumberField('exp', (rightNow + validFor));
if (claims != null) {
   for (String claim : claims.keySet()) {
      body.writeStringField(claim, claims.get(claim));
   }
}
body.writeEndObject();

jwt = encodedHeader + '.' + base64URLencode(Blob.valueOf(body.getAsString()));

if ( this.alg == HS256 ) {
   Blob key = EncodingUtil.base64Decode(privateKey);
   Blob signature = Crypto.generateMac('hmacSHA256',Blob.valueof(jwt),key);
   jwt += '.' + base64URLencode(signature);
} else if ( this.alg == RS256 ) {
   Blob signature = null;

   if (cert != null ) {
      signature = Crypto.signWithCertificate('rsa-sha256', Blob.valueOf(jwt), cert);
   } else {
      Blob privateKey = EncodingUtil.base64Decode(pkcs8);
      signature = Crypto.sign('rsa-sha256', Blob.valueOf(jwt), privateKey);
   }
   jwt += '.' + base64URLencode(signature);
```

```
        } else if ( this.alg == NONE ) {
            jwt += '.';
        }

        return jwt;

    }


    public String base64URLencode(Blob input){
        String output = encodingUtil.base64Encode(input);
        output = output.replace('+', '-');
        output = output.replace('/', '_');
        while ( output.endsWith('=')){
            output = output.subString(0,output.length()-1);
        }
        return output;
    }

}

public class HttpFormBuilder {

    private final static string Boundary = '1ff13444ed8140c7a32fc4e6451aa76d';
    public static string GetContentType() {
        return 'multipart/form-data; charset="UTF-8"; boundary="' + Boundary + '"';
    }

    /**
     *  Pad the value with spaces until the base64 encoding is no longer padded.
     */
    private static string SafelyPad(
        string value,
        string valueCrLf64,
        string lineBreaks) {
        string valueCrLf = '';
        blob valueCrLfBlob = null;
```

```
    while (valueCrLf64.endsWith('=')) {
        value += ' ';
        valueCrLf = value + lineBreaks;
        valueCrLfBlob = blob.valueOf(valueCrLf);
        valueCrLf64 = EncodingUtil.base64Encode(valueCrLfBlob);
    }

    return valueCrLf64;
}

/**
 *  Write a boundary between parameters to the form's body.
 */
public static string WriteBoundary() {
    string value = '--' + Boundary + '\r\n';
    blob valueBlob = blob.valueOf(value);

    return EncodingUtil.base64Encode(valueBlob);
}

/**
 *  Write a boundary at the end of the form's body.
 */
public static string WriteBoundary(
    EndingType ending) {
    string value = '';

    if (ending == EndingType.Cr) {
        //  The file's base64 was padded with a single '=',
        //  so it was replaced with '\r'. Now we have to
        //  prepend the boundary with '\n' to complete
        //  the line break.
        value += '\n';
    } else if (ending == EndingType.None) {
        //  The file's base64 was not padded at all,
        //  so we have to prepend the boundary with
        //  '\r\n' to create the line break.
        value += '\r\n';
```

```
        }
        //  Else:
        //  The file's base64 was padded with a double '=',
        //  so they were replaced with '\r\n'. We don't have to
        //  do anything to the boundary because there's a complete
        //  line break before it.

        value += '--' + Boundary + '--';

        blob valueBlob = blob.valueOf(value);

        return EncodingUtil.base64Encode(valueBlob);
    }

    /**
     *  Write a key-value pair to the form's body.
     */
    public static string WriteBodyParameter(
        string key,
        string value) {
        string contentDisposition = 'Content-Disposition: form-data; name="' + key + '"';
        string contentDispositionCrLf = contentDisposition + '\r\n\r\n';
        blob contentDispositionCrLfBlob = blob.valueOf(contentDispositionCrLf);
        string contentDispositionCrLf64 =
EncodingUtil.base64Encode(contentDispositionCrLfBlob);
        string content = SafelyPad(contentDisposition, contentDispositionCrLf64, '\r\n\r\n');
        string valueCrLf = value + '\r\n';
        blob valueCrLfBlob = blob.valueOf(valueCrLf);
        string valueCrLf64 = EncodingUtil.base64Encode(valueCrLfBlob);

        content += SafelyPad(value, valueCrLf64, '\r\n');

        return content;
    }

    /**
     *  Helper enum indicating how a file's base64 padding was replaced.
     */
```

```
    public enum EndingType {
        Cr,
        CrLf,
        None
    }
}

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));

  }

}

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));

    }

}

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));

    }

}

public with sharing class HandlerQuarter implements BotHandler {

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

        return new BotResponse(new BotMessage('Bot', 'Your quarter so far:', 'https://s3-us-
west-1.amazonaws.com/sfdc-demo/charts/quarter2.png'));

    }

}

public with sharing class HandlerPipeline implements BotHandler {

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

        return new BotResponse(new BotMessage('Bot', 'Here is your pipeline:', 'https://s3-
us-west-1.amazonaws.com/sfdc-demo/charts/pipeline.png'));

    }

}

public with sharing class HandlerNext implements BotHandler {

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

```
    List<Opportunity> opportunities =
        [SELECT Id, Name, Amount, Probability, StageName, CloseDate FROM
Opportunity WHERE isClosed=false ORDER BY amount DESC LIMIT 1];

    List<BotRecord> opportunityRecords = 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));
        opportunityRecords.add(new BotRecord(fields));
    }
    BotMessage opportunityMessage = new BotMessage('Bot', 'You have an overdue
item for the following opportunity:', opportunityRecords);

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

    List<BotRecord> caseRecords = 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('Status', c.Status));
        fields.add(new BotField('Contact', c.Contact.Name, '#/sObject/' + c.Contact.Id +
'/view'));
        caseRecords.add(new BotRecord(fields));
    }
    BotMessage caseMessage = new BotMessage('Bot', 'You should work on these
high priority cases assigned to you:', caseRecords);

    BotResponse r = new BotResponse();
```

```
        r.messages = new BotMessage[] {opportunityMessage, caseMessage};

        return r;

    }

}

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);

    }

}
```

```
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>();
      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;

    }

}

public with sharing class HandlerHelpTopic implements BotHandler {

    public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
                    if (session == null) {
                            List<BotMessageButton> buttons = new
List<BotMessageButton>();
            buttons.add(new BotMessageButton('User', 'User'));
            buttons.add(new BotMessageButton('Admin', 'Admin'));
            buttons.add(new BotMessageButton('Developer', 'Developer'));
            BotMessage message = new BotMessage('Bot', 'What best describes your role?',
buttons);
            session = new Map<String, String>();
            session.put('nextCommand', 'HandlerHelpTopic');
            return new BotResponse(message, session);
        }
                    List<BotItem> items = new List<BotItem>();
        if (utterance == 'User') {
            items.add(new BotItem('Salesforce User Tour',
'https://trailhead.salesforce.com/modules/lex_salesforce_tour'));
            items.add(new BotItem('Lightning Experience Features',
```

```
'https://trailhead.salesforce.com/modules/lex_migration_whatsnew'));
            items.add(new BotItem('Lightning Experience Chatter Basics',
'https://trailhead.salesforce.com/modules/lex_implementation_chatter'));
        } else if (utterance == 'Admin') {
            items.add(new BotItem('Lightning Experience Basics',
'https://trailhead.salesforce.com/modules/lex_migration_introduction'));
            items.add(new BotItem('Lightning Experience Features',
'https://trailhead.salesforce.com/modules/lex_migration_whatsnew'));
            items.add(new BotItem('Lightning Apps',
'https://trailhead.salesforce.com/modules/lightning_apps'));
            items.add(new BotItem('Lightning Experience Reports & Dashboards',
'https://trailhead.salesforce.com/modules/lex_implementation_reports_dashboards'));
        } else if (utterance == 'Developer') {
            items.add(new BotItem('Lightning Experience Development',
'https://trailhead.salesforce.com/modules/lex_dev_overview'));
            items.add(new BotItem('Lightning Components Basics',
'https://trailhead.salesforce.com/modules/lex_dev_lc_basics'));
            items.add(new BotItem('Visualforce & Lightning Experience',
'https://trailhead.salesforce.com/modules/lex_dev_visualforce'));
        }
        BotMessage message = new BotMessage('Bot', 'I recommend the following
Trailhead Modules:', items);
        return new BotResponse(message);
    }

}

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);
    }

}

public with sharing class HandlerHello implements BotHandler {

    public BotResponse handle(String utterance, String[] params, Map<String, String>
session, String fileName, String fileContent) {
        return new BotResponse(new BotMessage('Bot', 'Hi there!'));
    }

}

public with sharing class HandlerFindPropertiesByBedrooms 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) {
        List<Property__c> properties =
            [SELECT Id, Name, Beds__c, Baths__c, Price__c FROM Property__c
             WHERE City__c = :params[1] AND
             Beds__c = :Decimal.valueOf(params[0])
             ORDER BY Price__c
             LIMIT 10];
```

```
        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 ' + params[0] + '
bedrooms in ' + params[1] + ':', records));
    }

}

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);
                            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'));
        }
    }

}

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>();
          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));
```

```
    }

}

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));

    }

}

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'));
      }
   }

}

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'));
   }

}

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'));
   }

}
```

```
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;
      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);
      }
    }

  }

}

@isTest
public class PostPriceChangeToSlackTest {

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

```
}

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
            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
            ];
        }
    }


}

@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;
            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);
        }
    }

}

public class PropertyPagedResult {

    @AuraEnabled
    public Integer pageSize { get;set; }

    @AuraEnabled
    public Integer page { get;set; }

    @AuraEnabled
    public Integer total { get;set; }

    @AuraEnabled
    public List<Property__c> properties { get;set; }

}

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) {
                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);
            }
        }

    }

}

@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);
```

```
        }
    }

}

public class RandomContactFactory {
    public static List<Contact> generateRandomContacts(Integer
numContactsToGenerate, String FName) {
        List<Contact> contactList = new List<Contact>();
            for(Integer i=0;i<numContactsToGenerate;i++) {
            Contact c = new Contact(FirstName=FName + ' ' + i, LastName = 'Contact '+i);
            contactList.add(c);
            System.debug(c);
        }
        System.debug(contactList.size());
        return contactList;
    }
}

@isTest
public class RejectDuplicateFavoriteTest {

    public static String getUserNamePrefix(){
        return UserInfo.getOrganizationId() + System.now().millisecond();
    }

    public static User getTestUser(){
        Profile p = [SELECT Id FROM Profile WHERE Name='Standard User'];
        return new User(Alias='testuser', Email='test@user.com',
                    EmailEncodingKey='UTF-8', LastName='test', LanguageLocaleKey='en_US',
                    LocaleSidKey='en_US', ProfileId = p.Id,
                    TimeZoneSidKey='America/Los_Angeles',
UserName=getUserNamePrefix() + 'test@test.com');
    }

    static testMethod void acceptNonDuplicate() {
        Boolean success = true;
        try {
```

```
            Property__c p = new Property__c();
            insert p;
            User u = getTestUser();
            insert u;
            Favorite__c f1 = new Favorite__c(property__c=p.Id, user__c=u.Id);
                        insert f1;
        } catch (Exception e) {
            System.debug(e);
            success = false;
        } finally {
                System.assert(success);
        }
    }

    static testMethod void rejectDuplicate() {
        Boolean success = true;
        try {
            Property__c p = new Property__c();
            insert p;
            User u = getTestUser();
            insert u;
            Favorite__c f1 = new Favorite__c(property__c=p.Id, user__c=u.Id);
                        insert f1;
            Favorite__c f2 = new Favorite__c(property__c=p.Id, user__c=u.Id);
                        insert f2;
        } catch (Exception e) {
            success = false;
        } finally {
                System.assert(!success);
        }
    }

}

public with sharing class SlackOpportunityPublisher {

    private static final String slackURL =
```

```
Dreamhouse_Settings__c.getOrgDefaults().Slack_Opportunity_Webhook_URL__c;

   @InvocableMethod(label='Post to Slack')
   public static void postToSlack(List<Id> opportunityId) {
      Id oppId = opportunityId[0]; // If bulk, only post first to avoid overloading Slack
channel
      Opportunity opportunity = [SELECT Name, StageName from Opportunity WHERE
Id=:oppId];
                  Map<String,Object> msg = new Map<String,Object>();
                  msg.put('text', 'The following opportunity has changed:\n' +
opportunity.Name + '\nNew Stage: *'
            + opportunity.StageName + '*');
                  msg.put('mrkdwn', true);
      String body = JSON.serialize(msg);
      System.enqueueJob(new QueueableSlackCall(slackURL, 'POST', body));
   }

   public class QueueableSlackCall implements System.Queueable,
Database.AllowsCallouts {
      private final String url;
      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);
            }
        }
    }
}

@isTest
public class SlackOpportunityPublisherTest {

    static testMethod void testPost() {
        Boolean success = true;
        try {
            Opportunity opp = new Opportunity(Name='test opportunity', StageName='Close
Won', CloseDate=date.today());
            insert opp;
                SlackOpportunityPublisher.postToSlack(new List<Id> { opp.Id });
        } catch (Exception e) {
            success = false;
        } finally {
                System.assert(success);
        }
    }

}

@isTest
private class TestRestrictContactByName {
    static testMethod void  metodoTest()
    {
        List<Contact> listContact= new List<Contact>();
        Contact c1 = new Contact(FirstName='Francesco', LastName='Riggio' ,
email='Test@test.com');
        Contact c2 = new Contact(FirstName='Francesco1', LastName =
'INVALIDNAME',email='Test@test.com');
        listContact.add(c1);
        listContact.add(c2);
        Test.startTest();
```

```
        try
        {
            insert listContact;
        }
        catch(Exception ee) { }
    Test.stopTest();
  }
}

@isTest
public class TestVerifyDate {
  static testMethod void testMethod1() {
    Date d = VerifyDate.CheckDates(System.today(),System.today()+1);
    Date d1 = VerifyDate.CheckDates(System.today(),System.today()+60);
  }
}

public class VerifyDate {

    public static Date CheckDates(Date date1, Date date2) {

        if(DateWithin30Days(date1,date2)) {
            return date2;
        } else {
            return SetEndOfMonthDate(date1);
        }
    }

    private static Boolean DateWithin30Days(Date date1, Date date2) {

    if( date2 < date1) { return false; }

    Date date30Days = date1.addDays(30);
        if( date2 >= date30Days ) { return false; }
        else { return true; }
    }

    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;
        }

}

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);
      }

   }
         */

}
```