Animal Locator Mock

```
@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;
    }
}
```

Animal Locator

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

Animal Locator Test

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

Account Manager

```
@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;
    }

}
```

Account Manager Test

```
@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);
    }
}
```

Park Locator

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

Park Service Mock

```
@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);
   }
}
```

Park Locator Test

```
@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);
    }
}
```

Park Service

```
//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;
        }
    }
}
```

Async Park Service

```
//Generated by wsdl2apex

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

Contacts Today Controller

```
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;
    }

}
```

Contacts Today Controller Test

```
@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;
    }

}
```

Http Form Builder

```
public class HttpFormBuilder {

    private final static string Boundary = '1ff13444ed8140c7a32fc4e6451aa76d';

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

    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;
    }

    public static string WriteBoundary() {
        string value = '--' + Boundary + '\r\n';
        blob valueBlob = blob.valueOf(value);

        return EncodingUtil.base64Encode(valueBlob);
    }

    public static string WriteBoundary(
        EndingType ending) {
        string value = '';

        if (ending == EndingType.Cr) {
            value += '\n';
        } else if (ending == EndingType.None) {
            value += '\r\n';
        }

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

        blob valueBlob = blob.valueOf(value);

        return EncodingUtil.base64Encode(valueBlob);
    }
    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;
    }

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

JWT

```
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;
    }


}
```

JWT Bearer Flow

```
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;

    }

}
```

LIFX Controller

```
public with sharing class LIFXController {

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

  @AuraEnabled
  public static String getLights() {
    HttpRequest req = new HttpRequest();
    Http http = new Http();
    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() + '"}';
        }
    }

}
```

LIFX Controller Test

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

Post Price Change To Slack

```
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);
             }
        }

    }

}
```

Post Price Change To Slack Test

```
@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);
        }
    }

}
```

Property Controller

```
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
            ];
        }
    }


}
```

Property Controller Test

```
@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);
        }
    }

}
```

Property Paged Result

```
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; }

}
```

Push Price Change Notification

```
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);
            }
        }

    }

}
```

Push Price Change Notification Test

```
@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);
        }
    }

}
```

Reject Duplicate Favourite Test

```
@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);
        }
    }

}
```

Slack Opportunity Publisher

```
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);
            }
        }

    }

}
```

Slack Opportunity Publisher Test

```
@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);
        }
    }

}
```

Bot Controller

```
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'));
        }

    }

}
```

Bot Field

```
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;
    }

}
```

Bot Handler

```
public interface BotHandler {

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

Bot Item

```
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;
    }

}
```

Bot Message

```
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;
   }


}
```

Bot Message Button

```
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;
    }

}
```

Bot Record

```
public class BotRecord {

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

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

}
```

Bot response

```
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;
    }

}
```

Bot Test

```
@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);
    }

}
```

Dream House Sample Data Controller

```
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));

    }



}
```

Einsetein Vision Controller

```
global with sharing class EinsteinVisionController {

    public static String VISION_API = 'https://api.metamind.io/v1/vision';
            private static final Dreamhouse_Settings__c settings =
Dreamhouse_Settings__c.getOrgDefaults();

    public class Prediction {
        @AuraEnabled
        public String label {get;set;}
        @AuraEnabled
        public Double probability {get;set;}
    }
    private static String getAccessToken() {
        if (settings == null || String.isEmpty(settings.Einstein_Vision_Email__c)) {
            throw new AuraHandledException('Cannot create Einstein Vision token: "Einstein Vision
Email" not defined in Custom Settings');
        }
        ContentVersion base64Content;
        try {
            base64Content = [SELECT Title, VersionData FROM ContentVersion where
Title='einstein_platform' LIMIT 1];
```

```
        } catch (Exception e) {
                    throw new AuraHandledException('Cannot create Einstein Vision token:
einstein_platform.pem file not found');
        }
        String keyContents = base64Content.VersionData.tostring();
        keyContents = keyContents.replace('-----BEGIN RSA PRIVATE KEY-----', '');
        keyContents = keyContents.replace('-----END RSA PRIVATE KEY-----', '');
        keyContents = keyContents.replace('\n', '');

        JWT jwt = new JWT('RS256');
        jwt.pkcs8 = keyContents;
        jwt.iss = 'developer.force.com';
        jwt.sub = settings.Einstein_Vision_Email__c;
        jwt.aud = 'https://api.metamind.io/v1/oauth2/token';
        jwt.exp = '3600';
        String access_token;
        if (!Test.isRunningTest()) {
            access_token =
JWTBearerFlow.getAccessToken('https://api.metamind.io/v1/oauth2/token', jwt);
        }
        return access_token;
    }

    @AuraEnabled
    public static List<Prediction> predict(String fileName, String content, String modelId) {
        if (String.isBlank(modelId)) {
                    return EinsteinVisionController.predictDemo(fileName, content);
        } else {
                                return EinsteinVisionController.predictReal(fileName, content, modelId);
        }
    }

    @AuraEnabled
    public static List<Prediction> predictReal(String fileName, String content, String modelId) {
        String access_token;
        try {
                                access_token = EinsteinVisionController.getAccessToken();
        } catch (Exception e) {
                                throw new AuraHandledException('Cannot create Einstein Vision token.
Did you upload the einstein_platform.pem file and specify the Einstein Vision email address to
use in Custom Settings?');
```

```
        }
        List<Prediction> predictions = EinsteinVisionController.predictInternal(content,
access_token, modelId, true);
        return predictions;
    }

    @AuraEnabled
    public static List<Prediction> predictDemo(String fileName, String content) {
                    Integer pos = fileName.indexOf('_');
        String label;
        if (pos > 0) {

            label = fileName.substring(0, pos);
        } else {
                List<String> categories = new List<String>{'Victorian', 'Colonial', 'Contemporary'};
                Integer index = Math.mod(Math.round(Math.random()*1000), 3);
                label = categories[index];
        }
        List<Prediction> predictions = new List<Prediction>();
        Prediction prediction = new Prediction();
        prediction.label = label;
        prediction.probability = 1;
        predictions.add(prediction);
        return predictions;
    }

            @AuraEnabled
    public static String getDatasets() {
        String access_token = EinsteinVisionController.getAccessToken();
        HttpRequest req = new HttpRequest();
        req.setMethod('GET');
        req.setHeader('Authorization', 'Bearer ' + access_token);
        req.setHeader('Cache-Control', 'no-cache');
        req.setEndpoint(VISION_API + '/datasets');
                    try {
                Http http = new Http();
            if (!Test.isRunningTest()) {
                    HTTPResponse res = http.send(req);
                return res.getBody();
            } else {
                return '';
            }
        } catch(Exception ex){
            return '{"error": "' + ex.getMessage() + '"}';
        }
    }

            @AuraEnabled
    public static String getModelsByDataset(Integer datasetId) {
        String accessToken = EinsteinVisionController.getAccessToken();
        HttpRequest req = new HttpRequest();
        req.setMethod('GET');
        String endpoint = VISION_API + '/datasets/' + datasetId + '/models';
        req.setEndpoint(endpoint);
        req.setHeader('Authorization', 'Bearer ' + accessToken);
        req.setHeader('Cache-Control', 'no-cache');
                    try {
                Http http = new Http();
            if (!Test.isRunningTest()) {
                    HTTPResponse res = http.send(req);
                                         return res.getBody();
            } else {
                return null;
            }
        } catch(Exception ex){
            return '{"error": "' + ex.getMessage() + '"}';
        }
    }

    @AuraEnabled
    public static String deleteDataset(Integer datasetId) {
        String accessToken = EinsteinVisionController.getAccessToken();
        String endpoint = VISION_API + '/datasets/' + datasetId;
        HttpRequest req = new HttpRequest();
        req.setMethod('DELETE');
        req.setEndpoint(endpoint);
        req.setHeader('Authorization', 'Bearer ' + accessToken);
        req.setHeader('Cache-Control', 'no-cache');
                    try {
                Http http = new Http();
            if (!Test.isRunningTest()) {
                    HTTPResponse res = http.send(req);
```

```
                                        return res.getBody();
        } else {
            return null;
        }
    } catch(Exception ex){
        return '{"error": "' + ex.getMessage() + '"}';
    }
}

@AuraEnabled
public static String createDataset(String pathToZip) {
    System.debug(pathToZip);
    String accessToken = EinsteinVisionController.getAccessToken();
    String contentType = HttpFormBuilder.GetContentType();
    String form64 = '';
    form64 += HttpFormBuilder.WriteBoundary();
    form64 += HttpFormBuilder.WriteBodyParameter('path', pathToZip);
    form64 += HttpFormBuilder.WriteBoundary(HttpFormBuilder.EndingType.CrLf);
    Blob formBlob = EncodingUtil.base64Decode(form64);
    String contentLength = string.valueOf(formBlob.size());
    HttpRequest req = new HttpRequest();
    req.setBodyAsBlob(formBlob);
    req.setMethod('POST');
    req.setEndpoint(VISION_API + '/datasets/upload');
    req.setHeader('Authorization', 'Bearer ' + accessToken);
                  req.setHeader('Connection', 'keep-alive');
                  req.setHeader('Content-Length', contentLength);
    req.setHeader('Content-Type', contentType);

                  try {
             Http http = new Http();
        if (!Test.isRunningTest()) {
                 HTTPResponse res = http.send(req);
                                        return res.getBody();
        } else {
            return null;
        }
    } catch(Exception ex){
        return '{"error": "' + ex.getMessage() + '"}';
    }
}
```

```
    @AuraEnabled
    public static String trainModel(String modelName, Integer datasetId) {
        String accessToken = EinsteinVisionController.getAccessToken();
        string contentType = HttpFormBuilder.GetContentType();
        string form64 = '';
        form64 += HttpFormBuilder.WriteBoundary();
        form64 += HttpFormBuilder.WriteBodyParameter('name', modelName);
        form64 += HttpFormBuilder.WriteBoundary();
        form64 += HttpFormBuilder.WriteBodyParameter('datasetId', '' + datasetId);
        form64 += HttpFormBuilder.WriteBoundary(HttpFormBuilder.EndingType.CrLf);
        blob formBlob = EncodingUtil.base64Decode(form64);
        string contentLength = string.valueOf(formBlob.size());
        HttpRequest req = new HttpRequest();
                    req.setBodyAsBlob(formBlob);
        req.setMethod('POST');
        req.setEndpoint(VISION_API + '/train');
        req.setHeader('Authorization', 'Bearer ' + accessToken);
                    req.setHeader('Connection', 'keep-alive');
                    req.setHeader('Content-Length', contentLength);
        req.setHeader('Content-Type', contentType);
                    req.setHeader('Cache-Control', 'no-cache');
                    req.setTimeout(120000);

                    try {
                Http http = new Http();
            if (!Test.isRunningTest()) {
                  HTTPResponse res = http.send(req);
                                  return res.getBody();
            } else {
                return null;
            }
        } catch(Exception ex){
            return '{"error": "' + ex.getMessage() + '"}';
        }
    }

    private static List<Prediction> predictInternal(String sample, String access_token, String
  model, boolean isBase64) {
        string contentType = HttpFormBuilder.GetContentType();
        string form64 = '';
        form64 += HttpFormBuilder.WriteBoundary();
        form64 += HttpFormBuilder.WriteBodyParameter('modelId', EncodingUtil.urlEncode(model,
'UTF-8'));
        form64 += HttpFormBuilder.WriteBoundary();
        if(isBase64) {
          form64 += HttpFormBuilder.WriteBodyParameter('sampleBase64Content', sample);
        } else {
          form64 += HttpFormBuilder.WriteBodyParameter('sampleLocation', sample);
        }
        form64 += HttpFormBuilder.WriteBoundary(HttpFormBuilder.EndingType.CrLf);

        blob formBlob = EncodingUtil.base64Decode(form64);
        string contentLength = string.valueOf(formBlob.size());
        HttpRequest httpRequest = new HttpRequest();

        httpRequest.setBodyAsBlob(formBlob);
        httpRequest.setHeader('Connection', 'keep-alive');
        httpRequest.setHeader('Content-Length', contentLength);
        httpRequest.setHeader('Content-Type', contentType);
        httpRequest.setMethod('POST');
        httpRequest.setTimeout(120000);
        httpRequest.setHeader('Authorization','Bearer ' + access_token);
        httpRequest.setEndpoint(VISION_API + '/predict');

        Http http = new Http();
        List<Prediction> predictions = new List<Prediction>();
        if (!Test.isRunningTest()) {
          try {
            HTTPResponse res = http.send(httpRequest);
            if (res.getStatusCode() == 200) {
              System.JSONParser parser = System.JSON.createParser(res.getBody());
              while (parser.nextToken() != null) {
                if ((parser.getCurrentToken() == JSONToken.FIELD_NAME) && (parser.getText() ==
'probabilities')) {
                  parser.nextToken();
                  if (parser.getCurrentToken() == JSONToken.START_ARRAY) {
                    while (parser.nextToken() != null) {
                      // Advance to the start object marker to
                      //  find next probability object.
                      if (parser.getCurrentToken() == JSONToken.START_OBJECT) {
```

```
                    // Read entire probability object
                    Prediction probability = (Prediction)parser.readValueAs(Prediction.class);
                    predictions.add(probability);
                  }
                }
              }
              break;
            }
          }
        }
      } catch(System.CalloutException e) {
        System.debug('ERROR:' + e);
      }
    }
    return(predictions);
  }

}
```

Einstein Vision Controller Test

```
@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);
        }
    }

}
```

Handler Add Two Numbers

```
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);
        }

    }
```

```
}

Handler Cost Center
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'));
   }

}

Handler EmployeeId
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'));
   }

}

Handler File Upload
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'));
        }
    }

}
```

Handler Find Account

```
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));

    }

}
```

Handler Image Based Search

```
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;

    }

}
```

Handler Find Contact

```
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));

    }

}
```

Handler Find Properties

```
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'));
        }
    }

}
```

Handler Find Properties By Bedrooms

```
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));
    }

}
```

Handler Hello

```
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!'));
    }

}
```

Handler Help

```
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);
   }

}
```

Handler Help Topic

```
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);
    }

}
```

Handler My Open Cases

```
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);

    }

}
```

Handler Next

```
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;

    }

}
```

Handler Pipeline

```
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'));

    }

}
```

Handler Quarter

```
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'));

    }
}
```

Handler SOQL

```
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));

    }

}
```

Handler Top Opportunities

```
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));

    }

}
```

Handler Travel Approval

```
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));

    }


}
```

Daily Lead Processor Test

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

Daily Lead Processor

```
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;
      }
   }
}
```

Random Contact Factory

```
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;

   }

}
```

Lead Processor

```
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);
   }

}
```

Verify Date

```
public class VerifyDate {

        public static Date CheckDates(Date date1, Date date2) {
                if(DateWithin30Days(date1,date2)) {
                        return date2;
                } else {
                        return SetEndOfMonthDate(date1);
                }
        }

        @TestVisible 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; }
        }

        @TestVisible 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;
          }

}
```

Account Processor

```
public class AccountProcessor {
   @future
   public static void countContacts(List<Id> accountIds){

      List<Account> accountsToUpdate = new List<Account>();

      List<Account> accounts = [Select Id, Name, (Select Id from Contacts) from Account Where
Id in :accountIds];

      For(Account acc:accounts){
         List<Contact> contactList = acc.Contacts;
         acc.Number_Of_Contacts__c = contactList.size();
         accountsToUpdate.add(acc);

      }
      update accountsToUpdate;

   }

}
```

Test Verify Date

```
@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('12/30/2019'));
       System.assertEquals(false, flag);
   }

   @isTest static void Test_DateWithin30Days_case2(){
       Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));
       System.assertEquals(false, flag);
   }

   @isTest static void Test_DateWithin30Days_case3(){
       Boolean flag = VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('01/15/2020'));
       System.assertEquals(true, flag);
   }

   @isTest static void Test_SetEndOfMonthDate(){
       Date returndate = VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
   }

}
```

Test Restrict Contact By Name

```
@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 "INAVLIDNAME" is not allowed for DML',
result.getErrors()[0].getMessage());
   }

}
```

Lead Processor Test

```
@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();
   }

}
```

Add Primary Contact Test

```
@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')]);
    }

}
```

Add Primary Contact

```
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;
        }
    }

}
```

Account Processor Test

```
@IsTest
private class AccountProcessorTest {
    @IsTest
    private static void testCountContacts(){
        Account newAccount = new Account(Name='Test Account');
        insert newAccount;

        Contact newContact1 = new Contact(FirstName='John',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact1;

        Contact newContact2 = new Contact(FirstName='Jane',LastName='Doe',AccountId =
newAccount.Id);
        insert newContact2;

        List<Id> accountIds = new List<Id>();
        accountIds.add(newAccount.Id);

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

}
```

Closed Opportunity Trigger

```
trigger ClosedOpportunityTrigger on Opportunity (after insert, after update) {
 List<Task> tasklist = new List<Task>();

    for(Opportunity opp: Trigger.New){
        if(opp.StageName == 'Closed Won'){
            tasklist.add(new Task(Subject = 'Follow Up Test Task',WhatId = opp.Id));
        }
    }

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

Restrict Contact By Name

```
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');
        }

 }



}
```

Push Notification Trigger

```
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);
        }

    }
```

```
 */

}
```

Reject Duplicate Favourite

```
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');
   }

}
```

Account Address Trigger

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

   for(Account account:Trigger.New){
      if(account.Match_Billing_Address__c == True){
         account.ShippingPostalCode = account.BillingPostalCode;
      }
   }
}
```

Create Default Data

```
public with sharing class CreateDefaultData{
   Static Final String TYPE_ROUTINE_MAINTENANCE = 'Routine Maintenance';
   @AuraEnabled
   public static Boolean isDataCreated() {
      How_We_Roll_Settings__c          customSetting =
How_We_Roll_Settings__c.getOrgDefaults();
      return customSetting.Is_Data_Created__c;
   }
   @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;

    }
}
```

Maintenance Request Helper

```
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;
       }
   }
}
```

Maintainence Request Helper Test

```
@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);
    }
}
```

Warehouse Callout Service

```
public with sharing class WarehouseCalloutService {

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

    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);
      }

    }
  }
}
```

Warehouse Callout Service Mock

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

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

    HttpResponse response = new HttpResponse();
    response.setHeader('Content-Type', 'application/json');

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

```
        return response;
    }
}
```

Warehouse Callout Service Test

```
@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]);
    }
}
```

Create Default Data Test

```
@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');

    }
}
```

Warehouse Sync Schedule

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

        WarehouseCalloutService.runWarehouseEquipmentSync();
    }
}
```

Warehouse Sync Schedule Test

```
@isTest
public class WarehouseSyncScheduleTest {

    @isTest static void WarehousescheduleTest(){
        String scheduleTime = '00 00 01 * * ?';
        Test.startTest();
        Test.setMock(HttpCalloutMock.class, new WarehouseCalloutServiceMock());
        String jobID=System.schedule('Warehouse Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
        Test.stopTest();
        CronTrigger a=[SELECT Id FROM CronTrigger where NextFireTime > today];
        System.assertEquals(jobID, a.Id,'Schedule ');
        }
    }
```