Name: V SHIVA KUMAR CHARY

Salesforce Developer Catalyst Self-Learning & Super Badges Codes:

AccountManager :

@RestResource(urlMapping='/Accounts/\*/contacts')

global with sharing class AccountManager{

    @HttpGet

    global static Account getAccount(){

        RestRequest req = RestContext.request;

        String accId = req.requestURI.substringBetween('Accounts/', '/contacts');

        Account acc = [SELECT Id, Name, (SELECT Id, Name FROM Contacts)

                       FROM Account WHERE Id = :accId];

        return acc;

    }

}

AccountManagerTest :

@IsTest  
private class AccountManagerTest{  
    @isTest static void testAccountManager(){  
        Id recordId = getTestAccountId();  
        // Set up a test request  
        RestRequest request = new RestRequest();  
        request.requestUri =  
            'https://ap5.salesforce.com/services/apexrest/Accounts/'+ recordId +'/contacts';  
        request.httpMethod = 'GET';  
        RestContext.request = request;  
          
        // Call the method to test  
        Account  acc = AccountManager.getAccount();  
          
        // Verify results  
        System.assert(acc != null);  
    }  
      
    private static Id getTestAccountId(){  
        Account acc = new Account(Name = 'TestAcc2');  
        Insert acc;  
           
        Contact con = new Contact(LastName = 'TestCont2', AccountId = acc.Id);  
        Insert con;  
          
        return acc.Id;  
    }  
}

AccountProcessor :

public class AccountProcessor {  
  @future  
    public static void countContacts(List<Id> accountIds){  
        List<Contact> contactList;  
        Account currentAccount;  
        for(Id accountId :accountIds){  
      contactList = [SELECT Id FROM Contact WHERE AccountId = :accountId];  
            currentAccount = [SELECT Id FROM Account WHERE Id = :accountId];  
            currentAccount.Number\_of\_Contacts\_\_c = contactList.size();  
            update currentAccount;  
        }  
    }  
}

AccountProcessorTest :

@isTest  
public class AccountProcessorTest {  
    public static testMethod void asdf(){  
        Account a = new Account(Name='asdf');  
        insert a;  
        Test.startTest();  
        AccountProcessor.countContacts(new List<Id>{a.Id});  
        Test.stopTest();  
    }  
}

AddPrimaryContact :

public class AddPrimaryContact implements Queueable  
{  
    private Contact c;  
    private String state;  
    public  AddPrimaryContact(Contact c, String state)  
    {  
        this.c = c;  
        this.state = state;  
    }  
    public void execute(QueueableContext context)   
    {  
         List<Account> ListAccount = [SELECT ID, Name ,(Select id,FirstName,LastName from contacts ) FROM ACCOUNT WHERE BillingState = :state LIMIT 200];  
         List<Contact> lstContact = new List<Contact>();  
         for (Account acc:ListAccount)  
         {  
                 Contact cont = c.clone(false,false,false,false);  
                 cont.AccountId =  acc.id;  
                 lstContact.add( cont );  
         }  
           
         if(lstContact.size() >0 )  
         {  
             insert lstContact;  
         }  
               
    }  
  
}

AddPrimaryContactTest :

@isTest  
public class AddPrimaryContactTest   
{  
     @isTest static void TestList()  
     {  
         List<Account> Teste = new List <Account>();  
         for(Integer i=0;i<50;i++)  
         {  
             Teste.add(new Account(BillingState = 'CA', name = 'Test'+i));  
         }  
         for(Integer j=0;j<50;j++)  
         {  
             Teste.add(new Account(BillingState = 'NY', name = 'Test'+j));  
         }  
         insert Teste;  
  
         Contact co = new Contact();  
         co.FirstName='demo';  
         co.LastName ='demo';  
         insert co;  
         String state = 'CA';  
        
          AddPrimaryContact apc = new AddPrimaryContact(co, state);  
          Test.startTest();  
            System.enqueueJob(apc);  
          Test.stopTest();  
      }  
}

AnimalLocator :

public class AnimalLocator  
{  
  
  public static String getAnimalNameById(Integer id)  
   {  
        Http http = new Http();  
        HttpRequest request = new HttpRequest();  
        request.setEndpoint('https://th-apex-http-callout.herokuapp.com/animals/'+id);  
        request.setMethod('GET');  
        HttpResponse response = http.send(request);  
          String strResp = '';  
           system.debug('\*\*\*\*\*\*response '+response.getStatusCode());  
           system.debug('\*\*\*\*\*\*response '+response.getBody());  
        // If the request is successful, parse the JSON response.  
        if (response.getStatusCode() == 200)   
        {  
            // Deserializes the JSON string into collections of primitive data types.  
           Map<String, Object> results = (Map<String, Object>) JSON.deserializeUntyped(response.getBody());  
            // Cast the values in the 'animals' key as a list  
           Map<string,object> animals = (map<string,object>) results.get('animal');  
            System.debug('Received the following animals:' + animals );  
            strResp = string.valueof(animals.get('name'));  
            System.debug('strResp >>>>>>' + strResp );  
        }  
        return strResp ;  
   }  
    
}

AnimalLocatorMock :

@isTest  
global class AnimalLocatorMock implements HttpCalloutMock {  
    // Implement this interface method  
    global HTTPResponse respond(HTTPRequest request) {  
        // Create a fake response  
        HttpResponse response = new HttpResponse();  
        response.setHeader('Content-Type', 'application/json');  
        response.setBody('{"animal":{"id":1,"name":"chicken","eats":"chicken food","says":"cluck cluck"}}');  
        response.setStatusCode(200);  
        return response;   
    }  
}

AnimalLocatorTest :

@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);  
    }  
}

AsyncParkService :

//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'}  
            );  
        }  
    }  
}

AsyncParksServices :

//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'}  
            );  
        }  
    }  
}

BotController :

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'));  
        }  
          
    }  
      
}

BotField :

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;  
    }  
  
}

BotHandler :

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

BotItem :

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;  
    }  
  
}

BotMessage :

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;  
    }  
  
  
}

BotMessageButton :

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;  
    }  
      
}

BotRecord :

public class BotRecord {  
  
    @AuraEnabled   
    public List<BotField> fields { get;set; }  
      
    public BotRecord(List<BotField> fields) {  
        this.fields = fields;  
    }  
  
}

BotResponse :

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;  
    }  
  
    /\*\*  
     \* Convenience constructor to create a response with a single message  
     \*/  
    public BotResponse(BotMessage message) {  
        this.messages = new BotMessage[]{message};  
    }  
  
    /\*\*  
     \* Convenience constructor to create a response with a single message  
     \*/  
    public BotResponse(BotMessage message, Map<String, String> session) {  
        this.messages = new BotMessage[]{message};  
        this.session = session;  
    }  
      
}

BotTest :

@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);  
    }  
  
}

DailyLeadProcessor :

global class DailyLeadProcessor implements Schedulable {  
  
    global void execute(SchedulableContext ctx) {  
        List<Lead> lList = [Select Id, LeadSource from Lead where LeadSource = null];  
          
        if(!lList.isEmpty()) {  
      for(Lead l: lList) {  
        l.LeadSource = 'Dreamforce';  
      }  
      update lList;  
    }  
    }  
}

DailyLeadProcessorTest :

@isTest  
private class DailyLeadProcessorTest{  
    //Seconds Minutes Hours Day\_of\_month Month Day\_of\_week optional\_year  
    public static String CRON\_EXP = '0 0 0 2 6 ? 2022';  
      
    static testmethod void testScheduledJob(){  
        List<Lead> leads = new List<Lead>();  
          
        for(Integer i = 0; i < 200; i++){  
            Lead lead = new Lead(LastName = 'Test ' + i, LeadSource = '', Company = 'Test Company ' + i, Status = 'Open - Not Contacted');  
            leads.add(lead);  
        }  
          
        insert leads;  
          
        Test.startTest();  
        // Schedule the test job  
        String jobId = System.schedule('Update LeadSource to DreamForce', CRON\_EXP, new DailyLeadProcessor());  
          
        // Stopping the test will run the job synchronously  
        Test.stopTest();  
    }  
}

DreamHouseSampleDataController  :

global with sharing class DreamHouseSampleDataController {  
  
    @RemoteAction  
    global static void deleteAll() {  
        DELETE [SELECT ID FROM favorite\_\_c];      
        DELETE [SELECT ID FROM property\_\_c];      
        DELETE [SELECT ID FROM broker\_\_c];  
        DELETE [SELECT ID FROM bot\_command\_\_c];  
    }  
      
}

EinsteinVisionController :

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;}  
    }  
  
    // You can upload the 'einstein\_platform.pem' into your Salesforce org as 'File' sObject and read it as below  
    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', '');  
  
        // Get a new token  
        JWT jwt = new JWT('RS256');  
        jwt.pkcs8 = keyContents; // Comment this if you are using jwt.cert  
        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) {  
            // if the filename is like "victorian\_01.jpg", we return "victorian"  
          label = fileName.substring(0, pos);      
        } else {  
            // else we return a category selected randomly  
          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();  
        //  Compose the form  
        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());  
        //  Compose the http request  
        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);  
    }  
  
}

EinsteinVisionControllerTest :

@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);  
        }  
    }  
  
}

CreateDefaultData :

public with sharing class CreateDefaultData{  
    Static Final String TYPE\_ROUTINE\_MAINTENANCE = 'Routine Maintenance';  
    //gets value from custom metadata How\_We\_Roll\_Settings\_\_mdt to know if Default data was created  
    @AuraEnabled  
    public static Boolean isDataCreated() {  
        How\_We\_Roll\_Settings\_\_c  customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();  
        return customSetting.Is\_Data\_Created\_\_c;  
    }   
      
    //creates Default Data for How We Roll application  
    @AuraEnabled  
    public static void createDefaultData(){  
        List<Vehicle\_\_c> vehicles = createVehicles();  
        List<Product2> equipment = createEquipment();  
        List<Case> maintenanceRequest = createMaintenanceRequest(vehicles);  
        List<Equipment\_Maintenance\_Item\_\_c> joinRecords = createJoinRecords(equipment, maintenanceRequest);  
  
        updateCustomSetting(true);  
    }  
  
  
    public static void updateCustomSetting(Boolean isDataCreated){  
        How\_We\_Roll\_Settings\_\_c  customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();  
        customSetting.Is\_Data\_Created\_\_c = isDataCreated;  
        upsert customSetting;  
    }  
  
    public static List<Vehicle\_\_c> createVehicles(){  
        List<Vehicle\_\_c> vehicles = new List<Vehicle\_\_c>();  
        vehicles.add(new Vehicle\_\_c(Name = 'Toy Hauler RV', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Toy Hauler RV'));  
        vehicles.add(new Vehicle\_\_c(Name = 'Travel Trailer RV', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 2, Bedrooms\_\_c = 2, Model\_\_c = 'Travel Trailer RV'));  
        vehicles.add(new Vehicle\_\_c(Name = 'Teardrop Camper', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Teardrop Camper'));  
        vehicles.add(new Vehicle\_\_c(Name = 'Pop-Up Camper', Air\_Conditioner\_\_c = true, Bathrooms\_\_c = 1, Bedrooms\_\_c = 1, Model\_\_c = 'Pop-Up Camper'));  
        insert vehicles;  
        return vehicles;  
    }  
  
    public static List<Product2> createEquipment(){  
        List<Product2> equipments = new List<Product2>();  
        equipments.add(new Product2(Warehouse\_SKU\_\_c = '55d66226726b611100aaf741',name = 'Generator 1000 kW', Replacement\_Part\_\_c = true,Cost\_\_c = 100 ,Maintenance\_Cycle\_\_c = 100));  
        equipments.add(new Product2(name = 'Fuse 20B',Replacement\_Part\_\_c = true,Cost\_\_c = 1000, Maintenance\_Cycle\_\_c = 30  ));  
        equipments.add(new Product2(name = 'Breaker 13C',Replacement\_Part\_\_c = true,Cost\_\_c = 100  , Maintenance\_Cycle\_\_c = 15));  
        equipments.add(new Product2(name = 'UPS 20 VA',Replacement\_Part\_\_c = true,Cost\_\_c = 200  , Maintenance\_Cycle\_\_c = 60));  
        insert equipments;  
        return equipments;  
          
    }  
  
    public static List<Case> createMaintenanceRequest(List<Vehicle\_\_c> vehicles){  
        List<Case> maintenanceRequests = new List<Case>();  
        maintenanceRequests.add(new Case(Vehicle\_\_c = vehicles.get(1).Id, Type = TYPE\_ROUTINE\_MAINTENANCE, Date\_Reported\_\_c = Date.today()));  
        maintenanceRequests.add(new Case(Vehicle\_\_c = vehicles.get(2).Id, Type = TYPE\_ROUTINE\_MAINTENANCE, Date\_Reported\_\_c = Date.today()));  
        insert maintenanceRequests;  
        return maintenanceRequests;  
    }  
  
    public static List<Equipment\_Maintenance\_Item\_\_c> createJoinRecords(List<Product2> equipment, List<Case> maintenanceRequest){   
        List<Equipment\_Maintenance\_Item\_\_c> joinRecords = new List<Equipment\_Maintenance\_Item\_\_c>();  
        joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(0).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));  
        joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(1).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));  
        joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(2).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(0).Id));  
        joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(0).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));  
        joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(1).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));  
        joinRecords.add(new Equipment\_Maintenance\_Item\_\_c(Equipment\_\_c = equipment.get(2).Id, Maintenance\_Request\_\_c = maintenanceRequest.get(1).Id));  
        insert joinRecords;  
        return joinRecords;  
  
    }  
}

CreateDefaultDataTest :

@isTest  
private class CreateDefaultDataTest {  
    @isTest  
    static void createData\_test(){  
        Test.startTest();  
        CreateDefaultData.createDefaultData();  
        List<Vehicle\_\_c> vehicles = [SELECT Id FROM Vehicle\_\_c];  
        List<Product2> equipment = [SELECT Id FROM Product2];  
        List<Case> maintenanceRequest = [SELECT Id FROM Case];  
        List<Equipment\_Maintenance\_Item\_\_c> joinRecords = [SELECT Id FROM Equipment\_Maintenance\_Item\_\_c];  
  
        System.assertEquals(4, vehicles.size(), 'There should have been 4 vehicles created');  
        System.assertEquals(4, equipment.size(), 'There should have been 4 equipment created');  
        System.assertEquals(2, maintenanceRequest.size(), 'There should have been 2 maintenance request created');  
        System.assertEquals(6, joinRecords.size(), 'There should have been 6 equipment maintenance items created');  
  
    }  
  
    @isTest  
    static void updateCustomSetting\_test(){  
        How\_We\_Roll\_Settings\_\_c  customSetting = How\_We\_Roll\_Settings\_\_c.getOrgDefaults();  
        customSetting.Is\_Data\_Created\_\_c = false;  
        upsert customSetting;  
  
        System.assertEquals(false, CreateDefaultData.isDataCreated(), 'The custom setting How\_We\_Roll\_Settings\_\_c.Is\_Data\_Created\_\_c should be false');  
  
        customSetting.Is\_Data\_Created\_\_c = true;  
        upsert customSetting;  
         
        System.assertEquals(true, CreateDefaultData.isDataCreated(), 'The custom setting How\_We\_Roll\_Settings\_\_c.Is\_Data\_Created\_\_c should be true');  
  
    }  
}

MaintenanceRequestHelper :

public with sharing class MaintenanceRequestHelper {  
      
    public static void updateWorkOrders() {  
        // TODO: Complete the method to update workorders  
          
    }          
      
}

MaintenanceRequestHelperTest :

public with sharing class MaintenanceRequestHelperTest {  
    // implement scheduled code here  
}

WarehouseCalloutService :

public with sharing class WarehouseCalloutService {  
  
    private static final String WAREHOUSE\_URL = 'https://th-superbadge-apex.herokuapp.com/equipment';  
      
}

WarehouseCalloutServiceMock :

public class WarehouseCalloutServiceMock {  
    // implement http mock callout  
}

WarehouseCalloutServiceTest :

@IsTest  
private class WarehouseCalloutServiceTest {  
    // implement your mock callout test here  
  
}

WarehouseSyncSchedule :

public with sharing class WarehouseSyncSchedule {  
    // implement scheduled code here  
}

WarehouseSyncScheduleTest :

public with sharing class WarehouseSyncScheduleTest {  
    // implement scheduled code here  
}