NAPROCK International Procon 2017 Source Code List

Original Section

Title: MakeSense1

Country: Singapore

University/Institute:

Singapore Polytechnic

Members: Pan Ziyue

Wei Wen Jie

Mentor: Teo Shin Jen

# Contents

[Contents 2](#_Toc493794074)

[Foreword 3](#_Toc493794075)

[Sensing Subsystem 4](#_Toc493794076)

[ESP8266\_Side.ino 4](#_Toc493794077)

[Arduino\_Side.ino 13](#_Toc493794078)

[Cloud Computing Subsystem 17](#_Toc493794079)

[write\_timestamp\_calculate\_risk.py 17](#_Toc493794080)

[Visualization Subsystem 18](#_Toc493794081)

[index.html 18](#_Toc493794082)

# Foreword

All of the source code is available on <https://goo.gl/w89zo8> for download. Some source code may require installation of APIs and/or setting up of Amazon Web Services accounts, as well as requiring custom hardware tools to install the firmware on specialized microcontroller development boards.

Our system also contains 3D printed objects, which are available in the link above in the STL format.

# Sensing Subsystem

The source code files listed below are for the Sensing Subsystem, which uses the SPEEEduino, a custom designed Arduino. ESP8266\_Side.ino is loaded into the ESP8266 ESP-01 module’s flash memory using the Arduino IDE. Arduino\_Side.ino is loaded into the Arduino side of the SPEEEduino using the Arduino IDE.

Compiling the source code requires certain libraries to be installed, which are also included in the link in the Foreword. One must also install the ESP8266 boards using the Arduino Boards Manager in order to compile and upload to the ESP8266 module.

## ESP8266\_Side.ino

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Library include directives \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*#include <Arduino.h>*

*#include <Stream.h>*

*#include <ESP8266WiFi.h>*

*#include <ESP8266WiFiMulti.h>*

*#include <ESP8266TrueRandom.h>*

*//AWS*

*#include "sha256.h"*

*#include "Utils.h"*

*#include "AWSClient2.h"*

*//WEBSockets*

*#include <Hash.h>*

*#include <WebSocketsClient.h>*

*//MQTT PAHO*

*#include <SPI.h>*

*#include <IPStack.h>*

*#include <Countdown.h>*

*#include <MQTTClient.h>*

*//AWS MQTT Websocket*

*#include "Client.h"*

*#include "AWSWebSocketClient.h"*

*#include "CircularByteBuffer.h"*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constants declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*// Please change the parameters below to match your local configuration*

*// The variables below are for reference only*

**char** wifi\_ssid**[]**       **=** "WIFI-SSID"**;**

**char** wifi\_password**[]**   **=** "WIFI-PASSWORD"**;**

**char** aws\_endpoint**[]**    **=** "affym418ubqjw.iot.ap-northeast-1.amazonaws.com"**;**

**char** aws\_key**[]**         **=** "AKIAJV4GGWHNFMXRAEWQ"**;**

**char** aws\_secret**[]**      **=** "7ygAXPGdd0nLftV5Ca8EysYjf21ucGifhNynJqmR"**;**

**char** aws\_region**[]**      **=** "ap-northeast-1"**;**

**const** **char\*** aws\_topic  **=** "$aws/things/MakeSense1/shadow/update"**;**

**int** port **=** **443;**

*#define DEBUG\_PRINT 0*

**const** **int** maxMQTTpackageSize **=** **512;**

**const** **int** maxMQTTMessageHandlers **=** **1;**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

ESP8266WiFiMulti WiFiMulti**;**

AWSWebSocketClient awsWSclient**(1000);**

IPStack ipstack**(**awsWSclient**);**

MQTT**::**Client**<**IPStack**,** Countdown**,** maxMQTTpackageSize**,** maxMQTTMessageHandlers**>** **\***client **=** NULL**;**

**long** connection **=** **0;**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Helper function declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*//generate random mqtt clientID*

**char\*** generateClientID**()** **{**

**char\*** cID **=** **new** **char[23]();**

**for** **(int** i**=0;** i**<22;** i**+=1)**

  cID**[**i**]=(char)**random**(1,** **256);**

**return** cID**;**

**}**

*//count messages arrived*

**int** arrivedcount **=** **0;**

*//callback to handle mqtt messages*

**void** messageArrived**(**MQTT**::**MessageData**&** md**)** **{**

  MQTT**::**Message **&**message **=** md**.**message**;**

**if** **(**DEBUG\_PRINT**)** **{**

    Serial**.**print**(**"Message "**);**

    Serial**.**print**(++**arrivedcount**);**

    Serial**.**print**(**" arrived: qos "**);**

    Serial**.**print**(**message**.**qos**);**

    Serial**.**print**(**", retained "**);**

    Serial**.**print**(**message**.**retained**);**

    Serial**.**print**(**", dup "**);**

    Serial**.**print**(**message**.**dup**);**

    Serial**.**print**(**", packetid "**);**

    Serial**.**println**(**message**.**id**);**

    Serial**.**print**(**"Payload "**);**

**char\*** msg **=** **new** **char[**message**.**payloadlen**+1]();**

    memcpy **(**msg**,**message**.**payload**,**message**.**payloadlen**);**

    Serial**.**println**(**msg**);**

**delete** msg**;**

**}**

**}**

*//connects to websocket layer and mqtt layer*

**bool** connect**()** **{**

**if** **(**client **==** NULL**)** **{**

    client **=** **new** MQTT**::**Client**<**IPStack**,** Countdown**,** maxMQTTpackageSize**,** maxMQTTMessageHandlers**>(**ipstack**);**

**}** **else** **{**

**if** **(**client**->**isConnected**())** **{**

      client**->**disconnect**();**

**}**

**delete** client**;**

    client **=** **new** MQTT**::**Client**<**IPStack**,** Countdown**,** maxMQTTpackageSize**,** maxMQTTMessageHandlers**>(**ipstack**);**

**}**

*//delay is not necessary... it just help us to get a "trustful" heap space value*

  delay **(1000);**

**if** **(**DEBUG\_PRINT**)** **{**

    Serial**.**print **(**millis**());**

    Serial**.**print **(**" - conn: "**);**

    Serial**.**print **(++**connection**);**

    Serial**.**print **(**" - ("**);**

    Serial**.**print **(**ESP**.**getFreeHeap **());**

    Serial**.**println **(**")"**);**

**}**

**int** rc **=** ipstack**.**connect**(**aws\_endpoint**,** port**);**

**if** **(**rc **!=** **1)**

**{**

**if** **(**DEBUG\_PRINT**)** **{**

      Serial**.**println**(**"error connection to the websocket server"**);**

**}**

**return** false**;**

**}** **else** **{**

**if** **(**DEBUG\_PRINT**)** **{**

      Serial**.**println**(**"websocket layer connected"**);**

**}**

**}**

**if** **(**DEBUG\_PRINT**)** **{**

    Serial**.**println**(**"MQTT connecting"**);**

**}**

  MQTTPacket\_connectData data **=** MQTTPacket\_connectData\_initializer**;**

  data**.**MQTTVersion **=** **3;**

**char\*** clientID **=** generateClientID**();**

  data**.**clientID**.**cstring **=** clientID**;**

  rc **=** client**->**connect**(**data**);**

**delete[]** clientID**;**

**if** **(**rc **!=** **0)**

**{**

**if** **(**DEBUG\_PRINT**)** **{**

      Serial**.**print**(**"error connection to MQTT server"**);**

      Serial**.**println**(**rc**);**

**return** false**;**

**}**

**}**

**if** **(**DEBUG\_PRINT**)** **{**

    Serial**.**println**(**"MQTT connected"**);**

**}**

**return** true**;**

**}**

*//subscribe to a mqtt topic*

**void** subscribe**()** **{**

*//subscribe to a topic*

**int** rc **=** client**->**subscribe**(**aws\_topic**,** MQTT**::**QOS0**,** messageArrived**);**

**if** **(**rc **!=** **0)** **{**

**if** **(**DEBUG\_PRINT**)** **{**

      Serial**.**print**(**"rc from MQTT subscribe is "**);**

      Serial**.**println**(**rc**);**

**}**

**return;**

**}**

**if** **(**DEBUG\_PRINT**)** **{**

    Serial**.**println**(**"MQTT subscribed"**);**

**}**

**}**

**void** waitForWifi**()** **{**

**while(**WiFiMulti**.**run**()** **!=** WL\_CONNECTED**)** **{**

    delay**(100);**

**if** **(**DEBUG\_PRINT**)** **{**

      Serial**.**println**(**"waiting for wifi"**);**

**}**

**}**

**if** **(**DEBUG\_PRINT**)** **{**

    Serial**.**println**(**"\nconnected to network " **+** String**(**wifi\_ssid**)** **+** "\n"**);**

**}**

**}**

String getValue**(**String data**,** **char** separator**,** **int** index**)** **{**

**int** found **=** **0;**

**int** strIndex**[]** **=** **{** **0,** **-1** **};**

**int** maxIndex **=** data**.**length**()** **-** **1;**

**for** **(int** i **=** **0;** i **<=** maxIndex **&&** found **<=** index**;** i**++)** **{**

**if** **(**data**.**charAt**(**i**)** **==** separator **||** i **==** maxIndex**)** **{**

            found**++;**

            strIndex**[0]** **=** strIndex**[1]** **+** **1;**

            strIndex**[1]** **=** **(**i **==** maxIndex**)** **?** i**+1** **:** i**;**

**}**

**}**

**return** found **>** index **?** data**.**substring**(**strIndex**[0],** strIndex**[1])** **:** ""**;**

**}**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Program \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

**void** setup**()** **{**

  Serial**.**begin**(9600);**

  WiFiMulti**.**addAP**(**wifi\_ssid**,** wifi\_password**);**

  waitForWifi**();**

  awsWSclient**.**setAWSRegion**(**aws\_region**);**

  awsWSclient**.**setAWSDomain**(**aws\_endpoint**);**

  awsWSclient**.**setAWSKeyID**(**aws\_key**);**

  awsWSclient**.**setAWSSecretKey**(**aws\_secret**);**

  awsWSclient**.**setUseSSL**(**true**);**

**}**

**void** loop**()** **{**

*// Perform reading from Arduino side*

*// ESP8266 sends a single '.' character (ASCII code 0x2E)*

*// Arduino returns CSV style with LF line ending*

*// Format: Soil[0,1023],Rain[0,1023],Accl\_X[-255,255],Accl\_Y[-255,255],Accl\_Z[-255,255]*

**unsigned** **long** timeout **=** millis**()** **+** **2000UL;** *// 2 second timeout*

  Serial**.**print**(**"."**);**

  String inData **=** ""**;**

**while** **(1)** **{**

**if** **(**millis**()** **>** timeout**)** **{**

**break;**

**}**

**while** **(**Serial**.**available**())** **{**

**char** received **=** Serial**.**read**();**

**if** **(**received **==** '\n'**)** **{**

**goto** response**;**

**}**

      inData **+=** received**;**

**}**

**}**

response**:**

**bool** sensorError **=** false**;**

**if** **(**inData**.**length**()** **<** **1)** **{**

**if** **(**DEBUG\_PRINT**)** **{**

      Serial**.**println**(**"Error reading device sensor"**);**

**}**

    sensorError **=** true**;**

**}**

*// Split up response*

  String soil **=** getValue**(**inData**,** ','**,** **0);**

  String rain **=** getValue**(**inData**,** ','**,** **1);**

  String accl\_x **=** getValue**(**inData**,** ','**,** **2);**

  String accl\_y **=** getValue**(**inData**,** ','**,** **3);**

  String accl\_z **=** getValue**(**inData**,** ','**,** **4);**

  String gyro\_x **=** getValue**(**inData**,** ','**,** **5);**

  String gyro\_y **=** getValue**(**inData**,** ','**,** **6);**

  String gyro\_z **=** getValue**(**inData**,** ','**,** **7);**

**if** **(**sensorError**)** **{**

    soil **=** "null"**;**

    rain **=** "null"**;**

    accl\_x **=** "null"**;**

    accl\_y **=** "null"**;**

    accl\_z **=** "null"**;**

    gyro\_x **=** "null"**;**

    gyro\_y **=** "null"**;**

    gyro\_z **=** "null"**;**

**}**

*// Construct message to send to AWS IoT*

  String values **=** "{\"state\":{\"reported\":{}},\"entryUUID\":\""**;**

**uint8\_t** uuid**[16];**

  ESP8266TrueRandom**.**uuid**(**uuid**);**

  values **+=** ESP8266TrueRandom**.**uuidToString**(**uuid**);**

  values **+=** "\",\"deviceID\":0"**;** *//Change this device ID if needed*

**if** **(**sensorError**)** **{**

    values **+=** ",\"status\":-1"**;**

**}** **else** **{**

    values **+=** ",\"status\":0"**;**

    values **+=** ",\"soil\":"**+** soil**;**

    values **+=** ",\"rain\":"**+** rain**;**

    values **+=** ",\"accl\_x\":"**+** accl\_x**;**

    values **+=** ",\"accl\_y\":"**+** accl\_y**;**

    values **+=** ",\"accl\_z\":"**+** accl\_z**;**

    values **+=** ",\"gyro\_x\":"**+** gyro\_x**;**

    values **+=** ",\"gyro\_y\":"**+** gyro\_y**;**

    values **+=** ",\"gyro\_z\":"**+** gyro\_z**;**

**}**

  values **+=** "}"**;**

**const** **char** **\***publish\_message **=** values**.**c\_str**();**

*// Connect (if not yet connected) and send message*

**if** **(**awsWSclient**.**connected**())** **{**

    client**->**yield**();**

    subscribe**();**

*//publish*

    MQTT**::**Message message**;**

**char** buf**[1000];**

    strcpy**(**buf**,** publish\_message**);**

    message**.**qos **=** MQTT**::**QOS0**;**

    message**.**retained **=** false**;**

    message**.**dup **=** false**;**

    message**.**payload **=** **(void\*)**buf**;**

    message**.**payloadlen **=** strlen**(**buf**)+1;**

**int** rc **=** client**->**publish**(**aws\_topic**,** message**);**

**}** **else** **{**

*//handle reconnection*

    connect**();**

**}**

*// Send next message 30 seconds later*

  delay**(30000);**

**}**

## Arduino\_Side.ino

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Pin mapping \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*/\**

*\* GY-85 IMU*

*\* VCC\_IN  ->   5V*

*\* GND     ->   GND*

*\* SCL     ->   A5*

*\* SDA     ->   A4*

*\**

*\* Rainfall detector*

*\* VCC\_IN  ->   5V*

*\* GND     ->   GND*

*\* A0      ->   A0*

*\* D0      ->   NC*

*\**

*\* Soil Moisture Sensor*

*\* VCC\_IN  ->   5V*

*\* GND     ->   GND*

*\* A0      ->   A1*

*\* D0      ->   NC (no connect)*

*\*/*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Library include directives \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*#include "GY\_85.h"*

*#include <Wire.h>*

*#include <SoftwareSerial.h>*

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declarations \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

SoftwareSerial esp8266**(2,4);**

*//GY85 IMU*

GY\_85 gy85**;**

**int** ax**,** ay **,**az**;**

**float** gx**,** gy**,** gz**,** gt**;**

*// Rain and soil moisture values*

**int** rainValue**,** soilMoistureValue**;**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Helper functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

**void** readSensors**()** **{**

  ax **=** gy85**.**accelerometer\_x**(**gy85**.**readFromAccelerometer**());**

  ay **=** gy85**.**accelerometer\_y**(**gy85**.**readFromAccelerometer**());**

  az **=** gy85**.**accelerometer\_z**(**gy85**.**readFromAccelerometer**());**

  gx **=** gy85**.**gyro\_x**(**gy85**.**readGyro**());**

  gy **=** gy85**.**gyro\_y**(**gy85**.**readGyro**());**

  gz **=** gy85**.**gyro\_z**(**gy85**.**readGyro**());**

  gt **=** gy85**.**temp**(**gy85**.**readGyro**());**

  rainValue **=** analogRead**(**A0**);**

  soilMoistureValue **=** analogRead**(**A1**);**

**}**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Program \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

**void** setup**()** **{**

*// Setup ESP8266 serial port*

  Serial**.**begin**(9600);**

  esp8266**.**begin**(9600);**

*// Setup GY855 IMU*

  Wire**.**begin**();**

  delay**(10);**

  gy85**.**init**();**

  delay**(10);**

*// Setup onboard LED for status indication*

  pinMode**(**LED\_BUILTIN**,** OUTPUT**);**

**}**

**void** loop**()** **{**

*// Wait for '.' character*

**while** **(**esp8266**.**available**())** **{**

**if** **(**esp8266**.**read**()** **==** '.'**)** **{**

*// Read sensors*

      readSensors**();**

*// Construct data and send payload*

      esp8266**.**print**(**soilMoistureValue**);**

      esp8266**.**print**(**','**);**

      esp8266**.**print**(**rainValue**);**

      esp8266**.**print**(**','**);**

      esp8266**.**print**(**ax**);**

      esp8266**.**print**(**','**);**

      esp8266**.**print**(**ay**);**

      esp8266**.**print**(**','**);**

      esp8266**.**print**(**az**);**

      esp8266**.**print**(**','**);**

      esp8266**.**print**(**gx**);**

      esp8266**.**print**(**','**);**

      esp8266**.**print**(**gy**);**

      esp8266**.**print**(**','**);**

      esp8266**.**println**(**gz**);**

*// For debug only*

      Serial**.**print**(**soilMoistureValue**);**

      Serial**.**print**(**','**);**

      Serial**.**print**(**rainValue**);**

      Serial**.**print**(**','**);**

      Serial**.**print**(**ax**);**

      Serial**.**print**(**','**);**

      Serial**.**print**(**ay**);**

      Serial**.**print**(**','**);**

      Serial**.**print**(**az**);**

      Serial**.**print**(**','**);**

      Serial**.**print**(**gx**);**

      Serial**.**print**(**','**);**

      Serial**.**print**(**gy**);**

      Serial**.**print**(**','**);**

      Serial**.**println**(**gz**);**

*// Blink LED to indicate that ESP8266 has requested data*

      digitalWrite**(**LED\_BUILTIN**,** HIGH**);**

      delay**(0.5);**

      digitalWrite**(**LED\_BUILTIN**,** LOW**);**

**}**

**}**

**}**

# Cloud Computing Subsystem

The Cloud Computing Subsystem uses Amazon Web Services as the cloud hosting platform, and utilizes a variety of services, such as IoT, DynamoDB, Lambda and SNS. However, AWS does not contain much code, except for Lambda, which contains small Python scripts for serverless computation of data in the database. The **write\_timestamp\_calculate\_risk.py** script contains code that writes timestamps for every incoming MQTT update message, as well as runs the computation for the risk factor. **Disclaimer: We open source MakeSense1 to enable environmental scientists, researchers and users to evaluate their algorithms and findings on our device. The risk factor computation algorithm in this script is not a scientifically validated algorithm, and requires formal scientific inquiry to improve the accuracy of the algorithm.**

## write\_timestamp\_calculate\_risk.py

*# This file is for an AWS Lambda function of the same name*

*# This Lambda function automatically appends a createdDate property and*

*# computes roll and pitch data from an incoming AWS IoT MQTT update message*

*# and writes it into DynamoDB.*

*# Its secondary function is to recompute the risk factor whenever a new*

*# entry has been added to the database*

**import** datetime

**import** time

**import** math

**import** decimal

**import** boto3

dynamodb **=** boto3**.**resource**(**'dynamodb'**)**

entries\_table **=** dynamodb**.**Table**(**'entries'**)**

devices\_table **=** dynamodb**.**Table**(**'devices'**)**

**def** lambda\_handler**(**event**,** context**):**

*# For debugging purposes, print out event*

**print(**"DEBUG: Received event with data: "**)**

**print(**event**)**

*# Get current UNIX timestamp*

    dts **=** datetime**.**datetime**.**utcnow**()**

    epochtime **=** round**(**time**.**mktime**(**dts**.**timetuple**())** **+** dts**.**microsecond**/1e6)**

**print(**"DEBUG: Current UNIX timestamp: "**)**

**print(**epochtime**)**

*# Calculate roll and pitch (in degrees) for that sensor value in*

*# Formula available here: https://www.nxp.com/docs/en/application-note/AN3461.pdf*

*# Roll range: [0, 180], pitch range: [0, 90]*

    x **=** event**[**'accl\_x'**]**

    y **=** event**[**'accl\_z'**]** *# z and y are swapped as the sensor is mounted*

    z **=** event**[**'accl\_y'**]** *# vertically, not horizontally*

    roll **=** round**(**abs**(**math**.**atan2**(**y**,** z**))\*57.3)**

    pitch **=** round**(**abs**(**math**.**atan2**(-**x**,** math**.**sqrt**(**y**\***y **+** z**\***z**)))\*57.3)**

    insert\_entry**(**event**,** roll**,** pitch**,** epochtime**)**

    calculate\_riskfactor**(**event**,** roll**,** pitch**,** epochtime**)**

**def** insert\_entry**(**event**,** roll**,** pitch**,** timestamp**):**

*# Insert item into entries\_table*

**if** event**[**'status'**]** **==** **-1:**

*# Sensor has an issue, do not add other parameters*

        entries\_table**.**put\_item**(**Item**={**

            'entryUUID'**:** event**[**'entryUUID'**],**

            'status'**:** event**[**'status'**],**

            'deviceID'**:** event**[**'deviceID'**],**

            'createdDate'**:** timestamp

**})**

**elif** event**[**'status'**]** **==** **0:**

*# Sensor ok*

        entries\_table**.**put\_item**(**Item**={**

            'entryUUID'**:** event**[**'entryUUID'**],**

            'status'**:** event**[**'status'**],**

            'deviceID'**:** event**[**'deviceID'**],**

            'accl\_x'**:** decimal**.**Decimal**(**repr**(**event**[**'accl\_x'**])),**

            'accl\_y'**:** decimal**.**Decimal**(**repr**(**event**[**'accl\_y'**])),**

            'accl\_z'**:** decimal**.**Decimal**(**repr**(**event**[**'accl\_z'**])),**

            'gyro\_x'**:** decimal**.**Decimal**(**repr**(**event**[**'gyro\_x'**])),**

            'gyro\_y'**:** decimal**.**Decimal**(**repr**(**event**[**'gyro\_y'**])),**

            'gyro\_z'**:** decimal**.**Decimal**(**repr**(**event**[**'gyro\_z'**])),**

            'roll'**:** roll**,**

            'pitch'**:** pitch**,**

            'rain'**:** event**[**'rain'**],**

            'soil'**:** event**[**'soil'**],**

            'createdDate'**:** timestamp

**})**

*# Roll range: [0,180], pitch range: [0,90], soil range: [0,1023]*

*# riskFactor = d\*k*

*# where k = soil\_moisture/1023 + (roll/180+pitch/90)/2*

*# d = k1-k2*

**def** calculate\_riskfactor**(**event**,** roll**,** pitch**,** timestamp**):**

    risk\_factor **=** **0**

*# First step: retrieve deviceID from event*

**if** event**[**'status'**]** **!=** **0:**

**return** *#ignore this input as the sensor is not responding*

    device\_id **=** event**[**'deviceID'**]**

*# Second step: get 2 previous valid entries from this device ID and compute*

    response **=** entries\_table**.**scan**(**

        Limit**=10,**

        FilterExpression**=**boto3**.**dynamodb**.**conditions**.**Attr**(**'deviceID'**).**eq**(**device\_id**)** **&** boto3**.**dynamodb**.**conditions**.**Attr**(**'status'**).**eq**(0)**

**)**

    array **=** response**[**'Items'**]**

**if** len**(**array**)** **>** **2:**

*# The database has enough data*

        array**.**sort**(**key**=lambda** x**:** x**[**'createdDate'**],** reverse**=**False**)**

        k1 **=** array**[0][**'soil'**]/1023** **+** **(**array**[0][**'roll'**]/180** **+** array**[0][**'pitch'**]/90)/2**

**print(**'k1: '**)**

**print(**k1**)**

        k2 **=** array**[1][**'soil'**]/1023** **+** **(**array**[1][**'roll'**]/180** **+** array**[1][**'pitch'**]/90)/2**

**print(**'k2: '**)**

**print(**k2**)**

        d **=** abs**(**k1**-**k2**)**

**print(**'d: '**)**

**print(**d**)**

        k **=** decimal**.**Decimal**(**repr**(**event**[**'soil'**]/1023** **+** **(**roll**/180** **+** pitch**/90)/2))**

**print(**'k: '**)**

**print(**k**)**

        risk\_factor **=** d**\***k

**print(**'riskFactor: '**)**

**print(**risk\_factor**)**

*# Third step: Write results to database*

    devices\_table**.**update\_item**(**

        Key**={**

            'deviceID'**:** device\_id

**},**

        UpdateExpression**=**'SET riskFactor = :v1, lastUpdated = :v2'**,**

        ExpressionAttributeValues**={**

            ':v1'**:** risk\_factor**,**

            ':v2'**:** timestamp

**}**

**)**

## Trust Relationship Policy

**{**

**"Version":** "2012-10-17"**,**

**"Statement":** **[**

**{**

**"Effect":** "Allow"**,**

**"Principal":** **{**

**"Service":** **[**

          "edgelambda.amazonaws.com"**,**

          "lambda.amazonaws.com"**,**

          "apigateway.amazonaws.com"

**]**

**},**

**"Action":** "sts:AssumeRole"

**}**

**]**

**}**

## Body Mapping Template (Integration Request)

**{**

**"TableName":** "devices"

**}**

## Body Mapping Template (Integration Response)

#set($inputRoot = $input.path('$'))

**{**

    #foreach($elem in $inputRoot.Items)

**"ID$elem.deviceID.N":**

**[**$elem.lat.N**,** $elem.long.N**,** $elem.lastUpdated.N**,** $elem.riskFactor.N**]**#if($foreach.hasNext)**,**#end

#end

**}**

# Visualization Subsystem

The Visualization subsystem uses Amazon’s API Gateway service to provide data from AWS DynamoDB and a web interface containing D3.js to render the location of individual nodes and a custom Google Maps display. The Visualization Subsystem requires JQuery to function.

## index.html

*<!DOCTYPE html>*

**<html>**

**<head>**

**<title>**MakeSense1 Console**</title>**

**<meta** name="viewport" content="initial-scale=1.0, user-scalable=no"**>**

**<meta** charset="utf-8"**>**

**<script** src="jquery-3.2.1.min.js"**></script>**

**<style>**

**@import** **url(**'https://fonts.googleapis.com/css?family=Raleway:300,500,600'**);**

*/\* Always set the map height explicitly to define the size of the div*

*\* element that contains the map. \*/*

  #map **{**

**height:** **100%;**

**}**

*/\* Make the map fill the window. \*/*

**html,** **body** **{**

**height:** **100%;**

**margin:** **0;**

**padding:** **0;**

**}**

**h1** **{**

**padding-left:** **1.5em;**

**padding-top:** **0.5em;**

**color:** white**;**

**font-family:** 'Raleway'**,** **sans-serif;**

**font-weight:** **300;**

**font-size:** **300%;**

**}**

  .stations**,** .stations **svg** **{**

**position:** **absolute;**

**}**

  .stations **svg** **{**

**width:** **170px;**

**height:** **101px;**

**padding-right:** **100px;**

**font-size:** **20px;**

**font-family:** 'Raleway'**,** **sans-serif;**

**font-weight:** **500;**

**}**

  .loader **{**

**border:** **16px** **solid** **#f3f3f3;** */\* Light grey \*/*

**border-top:** **16px** **solid** **#3498db;** */\* Blue \*/*

**border-**radius**:** **50%;**

**width:** **120px;**

**height:** **120px;**

    animation**:** spin **2s** linear infinite**;**

**}**

**@keyframes** **spin** **{**

**0%** **{** transform**:** rotate**(0**deg**);** **}**

**100%** **{** transform**:** rotate**(360**deg**);** **}**

**}**

**</style>**

**</head>**

**<body>**

**<div** id="map"**></div>**

**<div** id="debug"**></div>**

**<div** id="loaderContainer"**></div>**

**<script** src="http://d3js.org/d3.v3.min.js"**></script>**

**<script>**

*// Function to easily calculate gradient from 2 colors and a percentage*

**function** makeGradientColor**(**color1**,** color2**,** percent**)** **{**

**var** newColor **=** **{};**

**function** makeChannel**(**a**,** b**)** **{**

**return(**a **+** Math**.**round**((**b**-**a**)\*(**percent**/100)));**

**}**

**function** makeColorPiece**(**num**)** **{**

      num **=** Math**.**min**(**num**,** **255);**   *// not more than 255*

      num **=** Math**.**max**(**num**,** **0);**     *// not less than 0*

**var** str **=** num**.**toString**(16);**

**if** **(**str**.**length **<** **2)** **{**

        str **=** "0" **+** str**;**

**}**

**return(**str**);**

**}**

    newColor**.**r **=** makeChannel**(**color1**.**r**,** color2**.**r**);**

    newColor**.**g **=** makeChannel**(**color1**.**g**,** color2**.**g**);**

    newColor**.**b **=** makeChannel**(**color1**.**b**,** color2**.**b**);**

    newColor**.**cssColor **=** "#" **+**

    makeColorPiece**(**newColor**.**r**)** **+**

    makeColorPiece**(**newColor**.**g**)** **+**

    makeColorPiece**(**newColor**.**b**);**

**return(**newColor**);**

**}**

*// var color1 = {r:255, g:251, b:213};*

**var** color1 **=** **{**r**:255,** g**:255,** b**:255};**

*// var color2 = {r:178, g:10, b:44};*

**var** color2 **=** **{**r**:255,** g**:0,** b**:0};**

*// Load up Google Maps*

**function** initMap**()** **{**

*// Styles a map in night mode.*

**var** map **=** **new** google**.**maps**.**Map**(**d3**.**select**(**"#map"**).**node**(),** **{**

      center**:** **{**lat**:** **1.350065,** lng**:** **103.778273},**

      zoom**:** **15,**

      styles**:** **[**

**{**

          "stylers"**:** **[**

**{**

              "hue"**:** "#ff1a00"

**},**

**{**

              "invert\_lightness"**:** **true**

**},**

**{**

              "saturation"**:** **-100**

**},**

**{**

              "lightness"**:** **33**

**},**

**{**

              "gamma"**:** **0.5**

**}**

**]**

**},**

**{**

          "featureType"**:** "water"**,**

          "elementType"**:** "geometry"**,**

          "stylers"**:** **[**

**{**

              "color"**:** "#2D333C"

**}**

**]**

**}**

**],**

      zoomControl**:** **true,**

      mapTypeControl**:** **false,**

      scaleControl**:** **true,**

      streetViewControl**:** **false,**

      rotateControl**:** **true,**

      fullscreenControl**:** **false**

**});**

*// Add elements into the view*

**var** x **=** document**.**getElementById**(**"debug"**);**

    map**.**controls**[**google**.**maps**.**ControlPosition**.**TOP\_LEFT**].**push**(**x**);**

    $**(**"#debug"**).**html**(**"<h1>MakeSense1 Console</h1>"**);**

**var** y **=** document**.**getElementById**(**"loaderContainer"**);**

    map**.**controls**[**google**.**maps**.**ControlPosition**.**CENTER**].**push**(**y**);**

    $**(**"#loaderContainer"**).**html**(**"<div class=\"loader\"></div>"**);**

*// Load the station data. When the data comes back, create an overlay.*

**var** url **=** "https://8t1vjclv99.execute-api.ap-northeast-1.amazonaws.com/Production"**;**

    d3**.**json**(**url**,** **function(**error**,** data**)** **{**

*// Remove spinner once loading completes*

**var** x **=** document**.**getElementById**(**"loaderContainer"**);**

      x**.**style**.**display **=** 'none'**;**

*// Handle error*

**if** **(**error**)** **{**

        window**.**alert**(**'An error occured while loading'**);**

**}** **else** **{**

        console**.**log**(**'Loaded Successfully'**);**

**}**

**var** overlay **=** **new** google**.**maps**.**OverlayView**();**

*// Add the container when the overlay is added to the map.*

      overlay**.**onAdd **=** **function()** **{**

**var** layer **=** d3**.**select**(this.**getPanes**().**overlayLayer**)**

**.**append**(**"div"**)**

**.**attr**(**"class"**,** "stations"**);**

*// Draw each marker as a separate SVG element.*

        overlay**.**draw **=** **function()** **{**

**var** projection **=** **this.**getProjection**();**

**var** padding **=** **50;**

**var** marker **=** layer**.**selectAll**(**"svg"**)**

**.**data**(**d3**.**entries**(**data**))**

**.**each**(**transform**)** *// update existing markers*

**.**enter**().**append**(**"svg"**)**

**.**each**(**transform**)**

**.**attr**(**"class"**,** "marker"**);**

*// Add a circle.*

          marker**.**append**(**"circle"**)**

**.**attr**(**"r"**,** **50)**

**.**attr**(**"cx"**,** padding**)**

**.**attr**(**"cy"**,** padding**)**

**.**attr**(**"fill-opacity"**,** "0.5"**)**

**.**attr**(**"fill"**,** **function(**d**)** **{**

**return** makeGradientColor**(**color1**,** color2**,** d**.**value**[3]\*100).**cssColor**;**

**});**

*// Add title label*

          marker**.**append**(**"text"**)**

**.**attr**(**"x"**,** padding **+** **60)**

**.**attr**(**"y"**,** padding **-** **15)**

**.**attr**(**"dy"**,** ".31em"**)**

**.**attr**(**"fill"**,** "white"**)**

**.**attr**(**"font-weight"**,** **600)**

**.**text**(function(**d**)** **{** **return** d**.**key**;** **});**

*// Add riskFactor label*

          marker**.**append**(**"text"**)**

**.**attr**(**"x"**,** padding **+** **60)**

**.**attr**(**"y"**,** padding **+** **15)**

**.**attr**(**"dy"**,** ".31em"**)**

**.**attr**(**"fill"**,** "white"**)**

**.**attr**(**"font-weight"**,** **500)**

**.**text**(function(**d**)** **{**

**return** "Risk Factor: " **+** Math**.**round**(**d**.**value**[3]** **\*** **100)** **+** "%"**;**

**});**

**function** transform**(**d**)** **{**

            d **=** **new** google**.**maps**.**LatLng**(**d**.**value**[0],** d**.**value**[1]);**

            d **=** projection**.**fromLatLngToDivPixel**(**d**);**

**return** d3**.**select**(this)**

**.**style**(**"left"**,** **(**d**.**x **-** padding**)** **+** "px"**)**

**.**style**(**"top"**,** **(**d**.**y **-** padding**)** **+** "px"**);**

**}**

**};**

**};**

*// Bind our overlay to the map*

      overlay**.**setMap**(**map**);**

**});**

**}**

**</script>**

**<script** src="https://maps.googleapis.com/maps/api/js?key=AIzaSyBbnHy\_9HBHYDYssKdBjJyX2W96lYoB5m8&callback=initMap&language=ja&region=SG" async defer**></script>**

**</body>**

**</html>**