% Code description: this is the main code for establishing a connection with the cloud through the Arduino microcontroller to upload the captured sensors' data.

% Code developed by:

%Adham Saleh, Mahmoud AL-Nuimat, and supervised by Dr. Ala' Khalifeh from
%School of Electrical Eng. and IT German-Jordanian University
%Amman, Jordan. Email: ala.khalifeh@gju.edu.jo
%Last accessed Jul 2016
%The code is freely available to use as far as it follows Creative Commons (CC) licensing model https://creativecommons.org/
% Note: the code calls functions to capture the medical sensors. The functions %are provided by the folowing reference:

%e-Health Sensor Platform V2.0 for Arduino and Raspberry Pi [Biometric / %Medical Applications], available online at:
%https://www.cooking-hacks.com/documentation/tutorials/ehe

```
// We start first by including the libraries

#include <SPI.h>                    // include the Serial Peripheral Interface library

#include <Ethernet.h>               // include the Ethernet library

#include <eHealth.h>                // include the eHealth library


/* Since we want to protect the patient data we decided to send it encrypted using the Advanced Encryption standard (AES). */

#include <AESLib.h>                 // include the AES library


//Define the length of the key used to encrypt the data.

uint8_tkey[]={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,
```

```
30,31};

// define the Ethernet shield mac address. This address is unique for each shield.
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

// define the Azure Mobile Service address.
// you can find this in your service dashboard.
const char *server = "ehealth12.azure-mobile.net";

// define the Azure Mobile Service table name.
const char *table_name = "tbl_BodyTemperature";

// Define the Azure Mobile Service Application Key.
// you can find this key in the 'Manage Keys' menu on the dashboard.
const char *ams_key = " RMLvRVPrlwpncSmPgIdDTYCZoeCZfw20";

// Array of String type JSON.
char JSON[80];

// Define Parameters.
EthernetClient client;
int value;
int Read;
char buffer[64];
```

```
// Send an HTTP POST request to the Azure Mobile Service data API.
Serial.println("\nconnecting...");
if (client.connect(server, 80)) {
Serial.print("sending ");
Serial.println(value);

// POST URI.
sprintf(buffer, "POST /tables/%s HTTP/1.1", table_name);
client.println(buffer);

// Host header.
sprintf(buffer, "Host: %s", server);
client.println(buffer);

// Azure Mobile Services application key.
sprintf(buffer, "X-ZUMO-APPLICATION: %s", ams_key);
client.println(buffer);

// JSON content type.
client.println("Content-Type: application/json");

// POST body.
// switch case to choose which table to update.
```

```
switch(value){

// update the value of the Airflow sensor table.
case 48:
Read = eHealth.getAirFlow();
table_name="tbl_Airflow";
sprintf(buffer, "{\"AirflowValue\":%d,\"UserID_FK\": %d}",Read ,1);
break;

// update the ECG Sensor table.
case 49:
Read = eHealth.getECG();
table_name="tbl_ECG";
sprintf(buffer, "{\"ECGValue\":%d,\"UserID_FK\": %d}",Read ,1);
break;

// update the value of the skin conductance Sensor table.
case 50:
Read = eHealth.getSkinConductanceVoltage();
table_name="tbl_GalvanicSkinResponse";
sprintf(buffer, "{\"GSRValue\":%d,\"UserID_FK\": %d}",Read ,1);
break;
```

```
// update the value of the EMG Sensor table.
case 51:
Read = eHealth.getEMG();
table_name="tbl_EMG";
sprintf(buffer, "{\"EMGValue\":%d,\"UserID_FK\": %d}",Read ,1);
break;

// update the temperature sensor table.
case 52:
Read = eHealth.getTemperature();
table_name="tbl_BodyTemperature";
sprintf(buffer, "{\"BodyTemperatureValue\":%d,\"UserID_FK\": %d}",Read,1);
delay(3000);
break;

// update the blood pressure sensor table.
case 53:
int parameter;
eHealth.initBloodPressureSensor(parameter);
int ystolicPressure= eHealth.getSystolicPressure();
delay(100);
int DiastolicPressure= eHealth.getDiastolicPressure();
table_name="tbl_GalvanicSkinResponse";
sprintf(buffer,"{\"MaxBloodPressureValue\":%d,\"MinBloodPressureValue\":
```

```
%d,\"UserID_FK\": %d}",DiastolicPressure,ystolicPressure,1);
delay(3000);
break;
}

// Content length.
client.print("Content-Length: ");
client.println(strlen(buffer));

// End of headers.
client.println();

// Request body.
client.println(buffer);
}

// If the connection has not been successfully established.
else {
Serial.println("connection failed");
}
}

// Wait for a response.
void wait_response() {
```

```
while (!client.available()) {
if (!client.connected()) {
return;
        }
        }
        }

        // Read the response and dump to serial.
        void read_response()
        {
        bool print = true;
        while (client.available()) {
        char c = client.read();

        // Print only until the first carriage return
                if (c == '\n')
        print = false;
        if (print)
        Serial.print(c);
        }
        }

    // Close the connection.
    void end_request()
```

```
        {
        client.stop();
        }

    //Arduino Setup, here the code will run only one time.
        void setup()
        {
        Serial.begin(9600);
        // wait for serial port to connect.
        while (!Serial) {
        ;
        }

        //Print the word Ethernet after serial port has connected
        Serial.println("ethernet");

// Wrong mac addresse
        if (Ethernet.begin(mac) == 0) {
        Serial.println("ethernet failed");
        for (;;) ;
        }
        // give the Ethernet shield a second to initialize:
        delay(1000);
        }
```

```
// Arduino Loop, the code will run here over and over.
    void loop() {


    // check if data has been sent from the computer:

       if (Serial.available()) {

    // read the most recent byte

         value = Serial.read();

       }

       }

    send_request(value);

       wait_response();

       read_response();

       end_request();

       delay(1000);

    }
```