WEBHOOK Update Account (APPS)

Update the Account of the Authenticated User if and only if they are Owner.

Host: `https://iot.algobel.com`
URL: `/update?`
Method: `GET`
Auth required: YES (“KEY”)

Permissions required: User is Account Owner

Data constraints

```
{
        key=xxxxxxxxxx
        pin_2    =       1 or 0
        pin_13   =       1 or 0
        pin_12   =       1 or 0
        pin_14   =       1 or 0
        pin_16   =       1 or 0
        pin_4    =       1 or 0
        pin_5    =       1 or 0
        analog   =       0 - 1024
        v_d_1    =       0 to any value
        v_d_2    =       0 to any value
        v_d_3    =       0 to any value
        v_d_4    =       0 to any value
        v_d_5    =       0 to any value

}
```

Data example Partial data is allowed, but at least one field must include.

```
{
        "key=xxxxxxxxx&pin_2=1" ( Pin -2 output set to HIGH)
        "key=xxxxxxxxx&pin_2=0" ( Pin -2 output set to LOW)

}
```

Success Responses

Condition: Update can be performed either fully or partially by the Owner of the Account.

Code: `200 OK - included in overhead with server time and other relevant details`
Content example : For the example above, when the 'pin_2' is updated and posted to the /update?key=xxxxxxx&pin_2=1

```
{
    {"pin":"pin_2","updated":1519035042}
}
```

Arduino Example:

```
void update_conditions(float TEM, float HUM, float LDR, float ULT)
{ // number of field which required to send to server
    // Use WiFiClient class to create TCP connections
    if (!client.connect(host, httpsPort))
    {
        Serial.println("connection failed");
        return;
    }
    // We now create a URI for the request
    String url = "/update";
    // url += streamId;
    url += "?key=";
    url += Key;
    url += "&v_d_1="; // Virtual data point -1
    url += TEM;       // data
    url += "&v_d_2="; // Virtual data point -2
    url += HUM;       // data
    url += "&v_d_3="; // Virtual data point -3
    url += LDR;
    url += "&v_d_4=";// Virtual data point -4
    url += ULT;
    url += "&analog=";
    url += analogRead(A0); // Analog Data

    // This will send the request to the server
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: " + host + "\r\n" +
                 "Connection: close\r\n\r\n");
    int timeout = millis() + 5000;
    while (client.available() == 0)
    {
        if (timeout - millis() < 0)
        {
            Serial.println(">>> Client Timeout !");
            client.stop();
            return;
        }
    }
    // Read all the lines of the reply from server and print them to Serial
    while (client.available())
    {
        String line = client.readStringUntil('\r');

        Serial.print(line); // Server result ( response from server)
    }
    Serial.println("closing connection");}
```

Multi-data update : For the example above, when the 'pin_2', 'pin_13', 'pin_12' is updated and posted to the /update?key=xxxxxxxx&&pin_2=1&pin_13=1&pin_12=1

{
{"pin":"pin_2","updated":1519035200}{"pin":"pin_13","updated":1519035200}{"pin":"pin_12","updated":1519035200}
}

Error Response

Condition: for updates

Code: `nothing will show (with less overhead)`
Content: `{}`

Or

Condition: Authorized User is not Owner of Account at URL.

Code: `nothing will send`
Content: `{}`

Notes

Data ignored

The endpoint will ignore irrelevant and read-only data such as parameters that don't exist, or some fields which are not editable.
E.g. if Account already exits:

Data example

```
{
    Nothing will show
}
```

Code: `200 OK`
Content example

```
{
    Json responds
}
```

Example Code :

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <ArduinoJson.h>


float T = 27.0; // Temperature
float H = 62.0; // Humidity
float L = 98.0; // LDR value
float U = 15.0; // Ultra sonic data

int Count = 0; // number of counts

const char *ssid = "kten";           // Wifi name (hotspot)
const char *password = "1234567a"; // Password

const char *host = "iot.algobel.com"; // iot platform
const int httpsPort = 443;

const char *Key = "789bc3b3a779e95a30af1f24xxxxxx"; // API KEY

// this will change every 6 months
const char *fingerprint = "5b 2e 4d 17 50 3e e8 1b f6 90 90 bd 9e 72 89 c1 fb
a8 2d 40";

WiFiClientSecure client; // Creating Secure Wifi client

void setup()
{
    Serial.begin(115200);
    delay(10);
    pinMode(2,OUTPUT);
    pinMode(14,INPUT);
    Serial.print("Connecting to ");

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(1000);
        Serial.print(".");
    }
}
```

```
double ptimer =0;
void loop()
{
    if (millis() - ptimer > 60000L)
    {
        ptimer = millis();
        update_conditions(T, H, L, U);
        yield(); // 0 delay
    }
    if (digitalRead(14) == 1) // press button
    {
        update_pin(2, 1);
    }

}

void update_pin(int pin, int con)
{
    if (!client.connect(host, httpsPort))
    {
        Serial.println("connection failed");
        return;
    }

    // We now create a URI for the request
    String url = "/update";
    // url += streamId;
    url += "?key=";
    url += Key;
    url += "&pin_";
    url += String(pin);
    url += "=";
    url += con;

    // This will send the request to the server
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: " + host + "\r\n" +
                 "Connection: close\r\n\r\n");
    int timeout = millis() + 5000;
    while (client.available() == 0)
    {
        if (timeout - millis() < 0)
        {
            Serial.println(">>> Client Timeout !");
            client.stop();
            return;
        }
    }
```

```
    // Read all the lines of the reply from server and print them to Serial
    while (client.available())
    {
        String line = client.readStringUntil('\r');

        Serial.print(line); // Server result ( response from server)
    }
    Serial.println("closing connection");
}

void update_conditions(float TEM, float HUM, float LDR, float ULT)
{ // number of field which required to send to server

    // Use WiFiClient class to create TCP connections

    if (!client.connect(host, httpsPort))
    {
        Serial.println("connection failed");
        return;
    }

    // We now create a URI for the request
    String url = "/update";
    // url += streamId;
    url += "?key=";
    url += Key;
    url += "&v_d_1="; // Virtual data point -1
    url += TEM;       // data
    url += "&v_d_2="; // Virtual data point -2
    url += HUM;       // data
    url += "&v_d_3="; // Virtual data point -3
    url += LDR;
    url += "&v_d_4=";// Virtual data point -4
    url += ULT;
    url += "&analog=";
    url += analogRead(A0); // Analog Data

    // This will send the request to the server
    client.print(String("GET ") + url + " HTTP/1.1\r\n" +
                 "Host: " + host + "\r\n" +
                 "Connection: close\r\n\r\n");
    int timeout = millis() + 5000;
    while (client.available() == 0)
    {
        if (timeout - millis() < 0)
        {
            Serial.println(">>> Client Timeout !");
```

```
            client.stop();
            return;
        }
    }

    // Read all the lines of the reply from server and print them to Serial
    while (client.available())
    {
        String line = client.readStringUntil('\r');

        Serial.print(line); // Server result ( response from server)
    }
    Serial.println("closing connection");
}
```