

CLOUD COMPUTING PROJECT

Using cloud to store and retrieve data



Kadirvelan L
2019202021

Abstract:

Smart energy monitoring system technique tracks the usage of energy.

Smart power meter provides data for optimization and reduce the power consumption.

The electricity consumption can be observed by the user through a user-friendly mobile app called Blynk that uses cloud to store and retrieve data.

As the results can be observed through the mobile application, the user can get some idea of saving and reducing electricity than earlier.

Hardware used:

- CT sensor.
- Esp8266.

Software used:

- Blynk cloud.
- Twilio whatsapp messaging api.

Language used:

- C++

Source code:

```
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <ThingESP.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <Hash.h>
#include <FS.h>
#include <ESP8266mDNS.h>
```

```
#include <WiFiUdp.h>
#include <ArduinoOTA.h>
#include "EmonLib.h" //https://github.com/openenergymonitor/EmonLib
#include <BlynkSimpleEsp8266.h>
```

```
ThingESP8266 thing("Kadir","EmonHome","ESP8266");
```

```
AsyncWebServer server(80);
```

```
float currCalibration_input=53;
```

```
float Volt_input=240.00;
```

```
EnergyMonitor emon;
```

```
BlynkTimer timer;
```

```
char auth[] = "sUqvrFO__RChzEwI59y_Y9x6fA55WQw5";
```

```
char ssid[] = "D-link EXT";
```

```
char pass[] = "XXXXXXXX";
```

```
const char* UNITS_INPUT = "unitsinput";
```

```
const char* CALIBRATION_INPUT = "calibinput";
```

```
const char* VOLT_INPUT = "voltinput";
```

```
const char index_html[] PROGMEM = R"rawliteral(
```

```
<!DOCTYPE HTML><html><head>
```

```
<title>ESP Energy Monitor Input Form</title>
```

```

<meta name="viewport" content="width=device-width, initial-scale=1">
</head><body>
<form action="/get">
    UNITS : <input type="number" placeholder="0.00" step="0.01" name="unitsinput"/>
    <input type="submit" value="Submit">
</form><br>
<form action="/get">
    CALIBRATION (Leave it... Don't enter any value) : <input type="number"
placeholder="0.00" step="0.01" name="calibinput"/>
    <input type="submit" value="Submit">
</form><br>
<form action="/get">
    VOLT (Leave it... Don't enter any value) : <input type="number" placeholder="0.00"
step="0.01" name="voltinput"/>
    <input type="submit" value="Submit">
</form><br>
<h7>&emsp;&emsp; Calibration = current_calibration* main reading / Emon reading </h7>
</body></html>rawliteral";

```

```

void notFound(AsyncWebServerRequest *request) {
    request->send(404, "text/plain", "Not found");
}

```

```

String readFile(fs::FS &fs, const char * path){
    Serial.printf("Reading file: %s\r\n", path);
    File file = fs.open(path, "r");
    if(!file || file.isDirectory()){

```

```

    Serial.println("- empty file or failed to open file");
    return String();
}

Serial.println("- read from file:");
String fileContent;
while(file.available()){
    fileContent+=String((char)file.read());
}
Serial.println(fileContent);
file.close();
return fileContent;
}

void writeFile(fs::FS &fs, const char * path, float Storevalue){
// Serial.printf("Writing file: %s\r\n", path);
File file = fs.open(path, "w");
if(!file){
    Serial.println("- failed to open file for writing");
    return;
}
if(file.print(Storevalue)){
// Serial.println("- file written");
    Serial.println(file.size());
} else {
// Serial.println("- write failed");
}
file.close();
}

```

```

void deleteFile(fs::FS &fs, const char * path){
    Serial.printf("Deleting file: %s\r\n", path);
    if(fs.remove(path)){
        Serial.println("- file deleted");
    } else {
        Serial.println("- delete failed");
    }
}

```

```

double price = 0.00;
float kWh = 0.00;
double Current = 0.00;
double Power = 0.00;
unsigned long lastmillis = millis();
unsigned long lasttimemillis=0;
unsigned long loopmillis=0;
unsigned long looptimemillis=0;
unsigned long periodicMillis=0;
const long INTERVAL=1000*86400;
int looptimes=1;
int times=1;

```

```

String HandleResponse(String query){
    if(query=="test"){
        return "Yes! im online.";
    }else if(query=="units"){
        return "Units: "+String(kWh);
    }else if(query=="current"){
        return "Current: "+String(Current)+" Amps";
    }
}

```

```

}else if(query=="power"){
    return "Power: "+String(Power)+" kWh";
}else if(query=="price"){
    return "Price: "+String(price)+" Rupees";}
else if(query=="status"){
    return "Status: \n Units: "+String(kWh)
        +"\n Current: "+String(Current)+" Amps"
        +"\n Power: "+String(Power)+" kWh"
        +"\n Price: "+String(price)+" Rupees"
        ;}
else return "Enter a valid query! queries: Units, Current, Power, Price, Status ";

}

```

```

void calc_all(){
    double Irms = emon.calcIrms(1480);
    Current=Irms;
    Serial.print("\tIrms: ");
    Serial.println(Irms, 6);
    Serial.print("\tkWh: ");
    Serial.println(Irms * Volt_input, 6);
    Power=Irms*Volt_input;
    Serial.print(millis()-lastmillis);

    Serial.print("\tUnits Consumed: ");
    kWh = kWh + (Power) * (millis()-lastmillis) / 3600000000.0;
    lastmillis = millis();
}

```

```
Serial.print(kWh, 4);  
Serial.println(" Units");  
Serial.print("\tCalibration: ");  
Serial.print(currCalibration_input);
```

```
if (kWh >= 0 and kWh <= 100)
```

```
{  
    price = 0;  
}
```

```
else if (kWh > 100 and kWh <= 200)
```

```
{  
    price = (kWh * 1.50) + 20;  
}
```

```
else if (kWh > 200 and kWh <= 500)
```

```
{  
    price = 150 - 150 + 200 + ((kWh - 200) * 3.0) + 30;  
}
```

```
else
```

```
{  
    price = 150 - 150 + 350 + 1380 + ((kWh - 500) * 6.60) + 50;  
}
```

```
delay(1000);
```

```
}
```

```
void myTimerEvent() {
```

```
    Blynk.virtualWrite(V0, analogRead(A0));
```

```
    Blynk.virtualWrite(V1, Current);
```

```
    Blynk.virtualWrite(V2, Power);
```



```
Blynk.virtualWrite(V4, price);
```

```
const char Streamdata_html[] PROGMEM = R"(
<!DOCTYPE HTML><html><head>

<title>ESP Smart Energy Monitor</title>

<h1>&emsp;&emsp;&emsp;&emsp;ESP Smart Energy Monitor</h1>

<meta name="viewport" content="\width=device-width, initial-scale=1>

</head><body>

<script>

    setInterval(function() {AjaxFunction();}, 100);

    function AjaxFunction()

    {

        //Streaming Price data....

        var ajax = new XMLHttpRequest();

        ajax.onreadystatechange = function() {

            if(this.readyState == 4 &&this.status==200) {

                document.getElementById("PriceStream").innerHTML="&emsp; Price:
"+this.responseText+" Rupees";

            } else {

                console.warn('Warning');

            }

        };

    }

);
```

```

//ajax.open("GET", "", true);

//ajax.send();

ajax.open("GET", "StreamPriceData", true);
ajax.send();


//Streaming Current data.....

var ajax = new XMLHttpRequest();

ajax.onreadystatechange = function() {
    if(this.readyState == 4 &&this.status==200) {
        document.getElementById("CurrentStream").innerHTML+="&nbsp; Current:
"+this.responseText+" Amps";
    } else {
        console.warn('Warning');
    }
};

ajax.open("GET", "StreamCurrentData", true);
ajax.send();

```

```

//Streaming Units data.....

var ajax = new XMLHttpRequest();

ajax.onreadystatechange = function() {
    if(this.readyState == 4 &&this.status==200) {
        document.getElementById("UnitsStream").innerHTML+="&nbsp; Units
Consumed: "+this.responseText+" Units";
    } else {
        console.warn('Warning');
    }
};

```

```

    }

};

ajax.open("GET", "StreamUnitsData", true);
ajax.send();


//Streaming Power data.....

var ajax = new XMLHttpRequest();

ajax.onreadystatechange = function() {
    if(this.readyState == 4 &&this.status==200) {
        document.getElementById("PowerStream").innerHTML="&emsp; Power:
"+this.responseText+" kWh";
    } else {
        console.warn('Warning');
    }
};

ajax.open("GET", "StreamPowerData", true);
ajax.send();


//Streaming Calibration data.....

var ajax = new XMLHttpRequest();

ajax.onreadystatechange = function() {
    if(this.readyState == 4 &&this.status==200) {
        document.getElementById("CalibStream").innerHTML="&emsp; Calibration:
"+this.responseText+" ";
    } else {
        console.warn('Warning');
    }
};

```

```

    }

};

ajax.open("GET", "StreamCalibrationData", true);

ajax.send();


//Streaming Volt data.....

var ajax = new XMLHttpRequest();


ajax.onreadystatechange = function() {

    if(this.readyState == 4 &&this.status==200) {

        document.getElementById("VoltStream").innerHTML="&nbsp; Volt:
"+this.responseText+" V";

    } else {

        console.warn('Warning');

    }

};

ajax.open("GET", "StreamVoltData", true);

ajax.send();


//.....

}


</script>

<br></h1><br><h1 style="color: #ed9d00" id="PriceStream"></h1>
<br><h1 style="color: #05c0f7" id="UnitsStream">Streaming...</h1>
<br><h1 style="color: #23c48e" id="CurrentStream">Streaming...</h1>
<br><h1 style="color: #d3435c" id="PowerStream">Streaming...</h1>
<br><h1 style="color: #4B0082" id="VoltStream">Streaming...</h1>

```


<h1 style="color: yellow" id="CalibStream">Streaming...</h1>

Click Here to Change values</body> Streaming...</html></h1>

</body></html>");

```
void setup() {  
  Serial.begin(9600);  
  thing.SetWiFi(ssid,pass);  
  thing.initDevice();  
  Serial.println("Booting");  
  
  if(!SPIFFS.begin()){  
    Serial.println("An Error has occurred while mounting SPIFFS");  
    return;  
  }  
  
  WiFi.mode(WIFI_STA);  
  WiFi.hostname("Emon");  
  WiFi.begin(ssid, pass);  
  
  while (WiFi.waitForConnectResult() != WL_CONNECTED) {  
    Serial.println("Connection Failed! Rebooting...");  
    delay(5000);  
    ESP.restart();  
  }
```

```

ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH) {
        type = "sketch";
    } else { // U_FS
        type = "filesystem";
    }

    Serial.println("Start updating " + type);
});
ArduinoOTA.onEnd([]() {
    Serial.println("\nEnd");
});
ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
});
ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) {
        Serial.println("Auth Failed");
    } else if (error == OTA_BEGIN_ERROR) {
        Serial.println("Begin Failed");
    } else if (error == OTA_CONNECT_ERROR) {
        Serial.println("Connect Failed");
    } else if (error == OTA_RECEIVE_ERROR) {
        Serial.println("Receive Failed");
    } else if (error == OTA_END_ERROR) {
        Serial.println("End Failed");
    }
}

```

```

});

ArduinoOTA.begin();

Serial.println("Ready");

Serial.print("IP address: ");

Serial.println(WiFi.localIP());

server.on("/", HTTP_GET, [](AsyncWebServerRequest * request) {
    request->send_P(200, "text/html", index_html);
});

```

```

server.on("/get", HTTP_GET, [] (AsyncWebServerRequest * request) {
    String inputMessage;
    String inputParam;
    if (request->hasParam(UNITS_INPUT)) {
        inputMessage = request->getParam(UNITS_INPUT)->value();
        inputParam = UNITS_INPUT;
        kWh=inputMessage.toFloat();
    } else if (request->hasParam(CALIBRATION_INPUT)) {
        inputMessage = request->getParam(CALIBRATION_INPUT)->value();
        inputParam = CALIBRATION_INPUT;
        currCalibration_input=inputMessage.toFloat();
        emon.current(A0, currCalibration_input);
    } else if (request->hasParam(VOLT_INPUT)) {
        inputMessage = request->getParam(VOLT_INPUT)->value();
        inputParam = VOLT_INPUT;
        Volt_input=inputMessage.toFloat();
        if(Volt_input<230.00 or NULL){
            Volt_input=230.00;
        }
    }
});

```

```

    }else{
        Volt_input=Volt_input;
    }
    } else {
        inputMessage = "No message sent";
        inputParam = "none";
    }
    Serial.println(inputMessage);
    request->send(200, "text/html", "HTTP GET request sent to your ESP on input field ("
        + inputParam + ") with value: " + inputMessage +
        "<br><a href=\"/\">Return to Home Page</a><meta http-equiv=\"refresh\"
content=\"1; URL=/Streamdata\" />");
    });

    server.on("/Streamdata", HTTP_GET, [] (AsyncWebServerRequest * request) {
        request->send_P(200, "text/html",Streamdata_html);
    });

    server.on("/StreamPriceData", HTTP_GET, [] (AsyncWebServerRequest * request) {
        request->send(200,"text/strings",String(price));
    });

    server.on("/StreamUnitsData", HTTP_GET, [] (AsyncWebServerRequest * request) {
        request->send(200,"text/strings",String(kWh));
    });

    server.on("/StreamCurrentData", HTTP_GET, [] (AsyncWebServerRequest * request) {
        request->send(200,"text/strings",String(Current));
    });

```



```
server.on("/StreamPowerData", HTTP_GET, [] (AsyncWebServerRequest * request) {  
    request->send(200,"text/strings",String(Power));  
});
```

```
server.on("/StreamCalibrationData", HTTP_GET, [] (AsyncWebServerRequest * request) {  
    request->send(200,"text/strings",String(currCalibration_input));  
});
```

```
server.on("/StreamVoltData", HTTP_GET, [] (AsyncWebServerRequest * request) {  
    request->send(200,"text/strings",String(Volt_input));  
});
```

```
emon.current(A0,currCalibration_input);
```

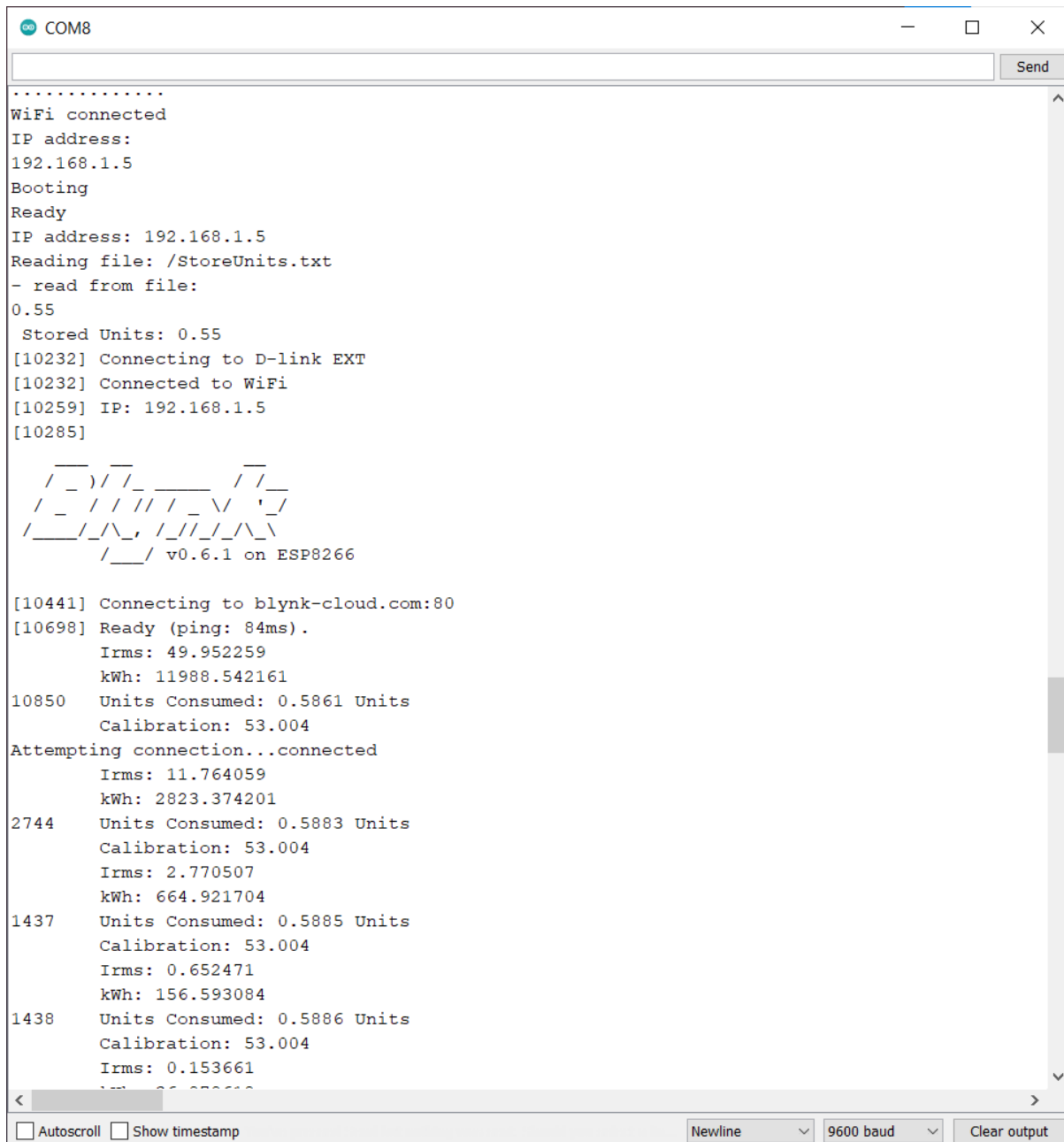
```
server.onNotFound(notFound);  
server.begin();
```

```
float StoredUnits = readFile(SPIFFS, "/StoreUnits.txt").toFloat();  
kWh = StoredUnits;  
Serial.print(" Stored Units: ");  
Serial.println(StoredUnits);  
Blynk.begin(auth, ssid, pass);  
timer.setInterval(1000L, myTimerEvent);  
}
```

```
void loop() {  
    MDNS.update();  
    ArduinoOTA.handle();  
    calc_all();
```

```
Blynk.run();  
timer.run();  
thing.Handle();  
if(millis()-periodicMillis>=INTERVAL){  
    periodicMillis = millis();  
  
    String msg = "Price: \n"+String(price);  
  
    thing.sendMsg("+918778580892",msg);  
}  
}
```

Serial Output of the esp8266 that connects to blynk-cloud:



```
COM8
.....
WiFi connected
IP address:
192.168.1.5
Booting
Ready
IP address: 192.168.1.5
Reading file: /StoreUnits.txt
- read from file:
0.55
  Stored Units: 0.55
[10232] Connecting to D-link EXT
[10232] Connected to WiFi
[10259] IP: 192.168.1.5
[10285]

  / _ ) / / _ _ _ _ / / _
 / _ / / / / _ \ ' /
 / _ _ / _ \ _ , / / / _ \ \
      / _ _ / v0.6.1 on ESP8266

[10441] Connecting to blynk-cloud.com:80
[10698] Ready (ping: 84ms).
      Irms: 49.952259
      kWh: 11988.542161
10850  Units Consumed: 0.5861 Units
      Calibration: 53.004
Attempting connection...connected
      Irms: 11.764059
      kWh: 2823.374201
2744  Units Consumed: 0.5883 Units
      Calibration: 53.004
      Irms: 2.770507
      kWh: 664.921704
1437  Units Consumed: 0.5885 Units
      Calibration: 53.004
      Irms: 0.652471
      kWh: 156.593084
1438  Units Consumed: 0.5886 Units
      Calibration: 53.004
      Irms: 0.153661
      kWh: 2823.374201

< [Autoscroll] [Show timestamp] [Newline] [9600 baud] [Clear output]
```



Mobile application through which the data can be viewed.

Output of twilio cloud whatsapp messaging api that responding to our query:

