```
//Setting Up Library Wifi & NTPClient
#include <NTPClient.h>
#include <WiFi.h>
#include <WiFiUdp.h>

const char *ssid     = "Y92018";
const char *password = "12345678910";

const long utcOffsetInSeconds = 25200;

// Setting tanggal menjadi nama hari
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"};

// Define NTP Client to get time
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "id.pool.ntp.org", utcOffsetInSeconds);

//Setting Up Firebase
#include <FirebaseESP32.h>
#define FIREBASE_HOST "monitoring-energy-2-default-rtdb.firebaseio.com"
#define FIREBASE_AUTH "iIwdDZ1cBkqP9LMaUAo7QXyAN7zvayzb2fkMuT0u"

//Define FirebaseESP32 data object
FirebaseData firebaseData;

//Setting Up Library Sensor PZEM
#include <PZEM004Tv30.h>
#define RXD1 14
#define TXD1 13
PZEM004Tv30 pzem_3(&Serial1);

//Setting Up Library OLED
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define OLED_RST -1
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RST);

//Setting Up Library LMIC
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

// LoRaWAN NwkSKey, network session key
// This should be in big-endian (aka msb).
static const PROGMEM u1_t NWKSKEY[16] = { 0x2E, 0xA8, 0xBD, 0xCA, 0xF4, 0x73, 0x5A,
0x95, 0x29, 0xFD, 0x64, 0xAE, 0x64, 0xE1, 0xF0, 0x82 };

// LoRaWAN AppSKey, application session key
// This should also be in big-endian (aka msb).
static const u1_t PROGMEM APPSKEY[16] = { 0xE5, 0xDD, 0xE4, 0xF3, 0xFD, 0x95, 0xC2,
0xE1, 0x2D, 0x3A, 0xB8, 0xCF, 0xFD, 0x10, 0xF1, 0x56 };

// LoRaWAN end-device address (DevAddr)
// See http://thethingsnetwork.org/wiki/AddressSpace
// The library converts the address to network byte order as needed, so this should
be in big-endian (aka msb) too.
static const u4_t DEVADDR = 0x26041ECE ; // <-- Change this address for every node!

// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in arduino-lmic/project_config/lmic_project_config.h,
// otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t payload[12];
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 10;
#define led 4

// Pin mapping
// Adapted for TTGO T-Beam
const lmic_pinmap lmic_pins = {
  .nss = 18,
  .rxtx = 0,
  .rst = 23,
  .dio = {26, 33, 32},
};

void onEvent (ev_t ev) {
  Serial.print(os_getTime());
  Serial.print(": ");
  switch (ev) {
    case EV_SCAN_TIMEOUT:
      Serial.println(F("EV_SCAN_TIMEOUT"));
      break;
    case EV_BEACON_FOUND:
      Serial.println(F("EV_BEACON_FOUND"));
      break;
    case EV_BEACON_MISSED:
      Serial.println(F("EV_BEACON_MISSED"));
      break;
    case EV_BEACON_TRACKED:
      Serial.println(F("EV_BEACON_TRACKED"));
      break;
    case EV_JOINING:
      Serial.println(F("EV_JOINING"));
      break;
    case EV_JOINED:
      Serial.println(F("EV_JOINED"));
      break;
    case EV_RFU1:
      Serial.println(F("EV_RFU1"));
      break;
    case EV_JOIN_FAILED:
      Serial.println(F("EV_JOIN_FAILED"));
      break;
    case EV_REJOIN_FAILED:
      Serial.println(F("EV_REJOIN_FAILED"));
      break;
    case EV_TXCOMPLETE:
      Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
      if (LMIC.txrxFlags & TXRX_ACK) {
        Serial.println(F("Received ack"));
      }
      //if (LMIC.dataLen) {
      Serial.println(F("Received "));
      Serial.print(LMIC.dataLen);
      Serial.println(F(" bytes of payload"));
      //}

      // Schedule next transmission
      os_setTimedCallback(&sendjob, os_getTime() + sec2osticks(TX_INTERVAL),
do_send);
      break;
    case EV_LOST_TSYNC:
      Serial.println(F("EV_LOST_TSYNC"));
      break;
    case EV_RESET:
      Serial.println(F("EV_RESET"));
      break;
    case EV_RXCOMPLETE:
      // data received in ping slot
      Serial.println(F("EV_RXCOMPLETE"));
      break;
    case EV_LINK_DEAD:
      Serial.println(F("EV_LINK_DEAD"));
      break;
    case EV_LINK_ALIVE:
      Serial.println(F("EV_LINK_ALIVE"));
      break;
    case EV_TXSTART:
      Serial.println(F("EV_TXSTART"));
      break;
    case EV_TXCANCELED:
      Serial.println(F("EV_TXCANCELED"));
      break;
    case EV_RXSTART:
      /* do not print anything -- it wrecks timing */
      break;
    case EV_JOIN_TXCOMPLETE:
      Serial.println(F("EV_JOIN_TXCOMPLETE: no JoinAccept"));
      break;
    default:
      Serial.print(F("Unknown event: "));
      Serial.println((unsigned) ev);
      break;
  }
}

void do_send(osjob_t* j) {
  // Check if there is not a current TX/RX job running
  if (LMIC.opmode & OP_TXRXPEND) {
    Serial.println(F("OP_TXRXPEND, not sending"));
  } else {
    //    float fVoltage3 = 380.8;
    //    float fCurrent3 = 13.4;
    //    float fPower3 = 15000;
    //    float fPf3 = 0.8;
    //    float fEnergy3 = 755.23;

    float fVoltage3 = pzem_3.voltage();
    float fCurrent3 = pzem_3.current();
    float fPower3 = pzem_3.power();
    float fPf3 = pzem_3.pf();
    float fEnergy3 = pzem_3.energy();

    Serial.println();
    Serial.print("Volt: "); Serial.print(fVoltage3, 2); Serial.print("V, ");
    Serial.print("current: "); Serial.print(fCurrent3, 3); Serial.print("A, ");
    Serial.print("pf: "); Serial.print(fPf3, 1); Serial.println("%, ");
    Serial.print("Power: "); Serial.print(fPower3, 1); Serial.print("W, ");
    Serial.print("Energy: "); Serial.print(fEnergy3, 3); Serial.print("kWh, ");
    //    Serial.print("freq: "); Serial.print(freq, 1); Serial.println("Hz, ");
    Serial.println();
    Serial.println();

    fVoltage3 = fVoltage3 / 1000;
    fCurrent3 = fCurrent3 / 100;
    fPower3 = fPower3 / 100000;
    fPf3 = fPf3 / 100;
    fEnergy3 = fEnergy3 / 1000;

    uint16_t uVoltage3 = LMIC_f2sflt16(fVoltage3);
    uint16_t uCurrent3 = LMIC_f2sflt16(fCurrent3);
    uint16_t uPower3 = LMIC_f2sflt16(fPower3);
    uint16_t uPf3 = LMIC_f2sflt16(fPf3);
    uint16_t uEnergy3 = LMIC_f2sflt16(fEnergy3);

    byte bVoltage3Low = lowByte(uVoltage3);
    byte bVoltage3High = highByte(uVoltage3);

    byte bCurrent3Low = lowByte(uCurrent3);
    byte bCurrent3High = highByte(uCurrent3);

    byte bPower3Low = lowByte(uPower3);
    byte bPower3High = highByte(uPower3);

    byte bPf3Low = lowByte(uPf3);
    byte bPf3High = highByte(uPf3);

    byte bEnergy3Low = lowByte(uEnergy3);
    byte bEnergy3High = highByte(uEnergy3);

    payload[0] = bVoltage3Low;
    payload[1] = bVoltage3High;
    payload[2] = bCurrent3Low;
    payload[3] = bCurrent3High;
    payload[4] = bPower3Low;
    payload[5] = bPower3High;
    payload[6] = bPf3Low;
    payload[7] = bPf3High;
    payload[8] = bEnergy3Low;
    payload[9] = bEnergy3High;

    // Prepare upstream data transmission at the next possible time.
```

```
    LMIC_setTxData2(1, payload, sizeof(payload) - 1, 0);
    Serial.println(F("LoRa Packet already sent"));

    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(BLACK, WHITE);
    display.setCursor(0, 0);
    display.print("Voltage:");
    display.setCursor(64, 0);
    display.print("Current:");
    display.setCursor(0, 24);
    display.print("Power:");
    display.setCursor(64, 24);
    display.print("Energy:");
    display.setCursor(0, 46);
    display.print("Frequency:");
    display.setCursor(64, 46);
    display.print("PowerFact:");
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.setCursor(0, 10);
    display.print(String(fVoltage3 * 1000, 1) + " V");
    display.setCursor(64, 10);
    display.print(String(fCurrent3 * 100, 2) + " A");
    display.setCursor(0, 34);
    display.print(String(fPower3 * 100000, 1) + " W");
    display.setCursor(64, 34);
    display.print(String(fEnergy3 * 1000, 3) + " kWh");
    display.setCursor(0, 56);
    //    display.print(String(49.93, 1) + " Hz");
    display.print(String(pzem_3.frequency(), 1) + " Hz");
    display.setCursor(64, 56);
    display.print(fPf3 * 100, 1);
    display.display();
  }
  // Next TX is scheduled after TX_COMPLETE event.

  timeClient.update();
  unsigned long epochTime = timeClient.getEpochTime();
  struct tm *ptm = gmtime ((time_t *)&epochTime);
  int monthDay = ptm->tm_mday;

  Serial.print(String(daysOfTheWeek[timeClient.getDay()]) + "(" + String(monthDay)
+ ")");
  //  Serial.print(monthDay);
  Serial.print(", ");
  Serial.print(timeClient.getHours());
  Serial.print(":");
  Serial.print(timeClient.getMinutes());
  Serial.print(":");
  Serial.println(timeClient.getSeconds());

  if (monthDay == 25) {
    if (timeClient.getHours() == 16) {
      if (timeClient.getMinutes() == 3) {
        if (timeClient.getSeconds() >= 30 && timeClient.getSeconds() <= 60) {
          Serial.println("RESET ENERGY");
          pzem_3.resetEnergy();
        }
      }
    }
  }


  if (Firebase.getInt(firebaseData, "/Control_Relay3/valRelay3")) {
    if (firebaseData.dataType() == "int") {
      int val = firebaseData.intData();
      Serial.println(val);
      digitalWrite(led, val);
    }
  }
}

void setup() {
  Serial.begin(115200);
  Serial1.begin(9600, SERIAL_8N1, RXD1, TXD1);
  Serial.println(F("Starting"));
  pinMode(4, OUTPUT);

  WiFi.begin(ssid, password);
  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
    Serial.print ( "." );
  }
  timeClient.begin();

  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);

  pinMode(OLED_RST, OUTPUT);
  digitalWrite(OLED_RST, LOW);
  delay(20);
  digitalWrite(OLED_RST, HIGH);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

#ifdef VCC_ENABLE
  // For Pinoccio Scout boards
  pinMode(VCC_ENABLE, OUTPUT);
  digitalWrite(VCC_ENABLE, HIGH);
  delay(1000);
#endif

  // LMIC init
  os_init();
  // Reset the MAC state. Session and pending data transfers will be discarded.
  LMIC_reset();
  LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);

  // Set static session parameters. Instead of dynamically establishing a session
  // by joining the network, precomputed session parameters are be provided.
#ifdef PROGMEM
  // On AVR, these values are stored in flash and only copied to RAM
  // once. Copy them to a temporary buffer here, LMIC_setSession will
  // copy them into a buffer of its own again.
  uint8_t appskey[sizeof(APPSKEY)];
  uint8_t nwkskey[sizeof(NWKSKEY)];
  memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
  memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
  LMIC_setSession (0x13, DEVADDR, nwkskey, appskey);
```

```
#else
  // If not running an AVR with PROGMEM, just use the arrays directly
  LMIC_setSession (0x13, DEVADDR, NWKSKEY, APPSKEY);
#endif

#if defined(CFG_as923)
  LMIC_setupChannel(0, 923200000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);
#else
# error Region not supported
#endif

  // Disable link check validation
  LMIC_setLinkCheckMode(0);

  // Set SF LMIC
  LMIC.dn2Dr = DR_SF10;

  // Set data rate and transmit power for uplink
  LMIC_setDrTxpow(DR_SF7, 14);

  // Start job
  do_send(&sendjob);
}

void loop() {
  os_runloop_once();
}
```