Kód Arduina

```
#include <LiquidCrystal.h>
#include "Wire.h"
#include <Keypad.h>

#define DS3231_I2C_ADDRESS 0x68

byte decToBcd(byte val) {
  return ((val / 10 * 16) + (val % 10));
}

byte bcdToDec(byte val) {
  return ((val / 16 * 10) + (val % 16));
}

const int rs = 30, en = 31, d4 = 32, d5 = 33, d6 = 34, d7 = 35;
const byte numChars = 32;
char receivedChars[numChars];
char tempChars[numChars];
boolean newData = false;
unsigned long dutycycle = 85;
const byte ROWS = 4;
const byte COLS = 4;
byte rowPins[ROWS] = {50,51,52,53};
byte colPins[COLS] = {46,47,48,49};
const byte pin=23;
char mistnost= "N";
```

```
char messageFromESP[numChars] = {0};
int integerFromESP = 0;
float floatFromESP = 0.0;
float tempreq= 22.5;
char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
  Wire.begin();
  Serial.begin(115200);
  pinMode(23, OUTPUT);
}

void setDS3231time(byte second, byte minute, byte hour, byte dayOfWeek, byte dayOfMonth, byte
month, byte year) {
```

```
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0);                  // set next input to start at the seconds register
  Wire.write(decToBcd(second));     // set seconds
  Wire.write(decToBcd(minute));     // set minutes
  Wire.write(decToBcd(hour));       // set hours
  Wire.write(decToBcd(dayOfWeek));   // set day of week (1=Sunday, 7=Saturday)
  Wire.write(decToBcd(dayOfMonth));  // set date (1 to 31)
  Wire.write(decToBcd(month));      // set month
  Wire.write(decToBcd(year));       // set year (0 to 99)
  Wire.endTransmission();
}
void readDS3231time(byte *second, byte *minute, byte *hour, byte *dayOfWeek, byte *dayOfMonth,
byte *month, byte *year) {
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0);  // set DS3231 register pointer to 00h
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);

  *second = bcdToDec(Wire.read() & 0x7f);
  *minute = bcdToDec(Wire.read());
  *hour = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month = bcdToDec(Wire.read());
  *year = bcdToDec(Wire.read());
}
```

```
void recvWithStartEndMarkers() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (recvInProgress == true) {
            if (rc != endMarker) {
                receivedChars[ndx] = rc;
                ndx++;
                if (ndx >= numChars) {
                    ndx = numChars - 1;
                }
            }
            else {
                receivedChars[ndx] = '\0'; // terminate the string
                recvInProgress = false;
                ndx = 0;
```

```
            newData = true;
          }
        }

        else if (rc == startMarker) {
          recvInProgress = true;
        }
    }
}

void parseData() {     // split the data into its parts

    char * strtokIndx; // this is used by strtok() as an index

    strtokIndx = strtok(tempChars,",");      // get the first part - the string
    strcpy(messageFromESP, strtokIndx); // copy it to messageFromPC

    strtokIndx = strtok(NULL, ","); // this continues where the previous call left off
    integerFromESP = atoi(strtokIndx);     // convert this part to an integer

    strtokIndx = strtok(NULL, ",");
    floatFromESP = atof(strtokIndx);
}

void showParsedData() {
    Serial.print("Message ");
    Serial.println(messageFromESP);
    Serial.print("Integer ");
    Serial.println(integerFromESP);
```

```
    Serial.print("Float ");
    Serial.println(floatFromESP);
}



void SlowPWM(byte pin, unsigned long opened, unsigned long T) {
  static unsigned long then = millis();

    if(opened>100){opened = 100;}

  if ((millis()-then) >= T-(T*opened/100)) {
    digitalWrite(pin, false);
  }

  if ((millis()-then) >= T) {
    digitalWrite(pin, true);
    then = millis();
  }
}

void duty(float temp) {
  static unsigned long now = millis();
  static unsigned long start = millis();
  const unsigned long period = 3600000;
   if ((now-start) >= period) {
    if (tempreq=0){
      dutycycle=0;
    }
    if (temp>tempreq){
```

```
      dutycycle=dutycycle-5;
    }
    if (temp<tempreq){
      dutycycle=dutycycle+5;
    }
    start=now;
    Serial.println(dutycycle);
  }
}

void loop() {
  char customKey = customKeypad.getKey();
  if (customKey){
    if (customKey == '0')
    {
      tempreq = 0;
    }
    if (customKey == '1')
    {
      tempreq = 22.5;
    }
    if (customKey == 'A')
    {
      mistnost = 'A';
    }
    Serial.println(customKey);
    lcd.clear();
    lcd.print("Pozadavek ");
    lcd.print(tempreq);
```

```
  lcd.setCursor(0,1);
  lcd.print("Aktualne ");
  lcd.print(floatFromESP);
  lcd.setCursor(15,1);
  lcd.print(mistnost);
  }


 recvWithStartEndMarkers();
 if (newData == true) {
  strcpy(tempChars, receivedChars);
  parseData();
  showParsedData();
  newData = false;
  }

 SlowPWM(23, dutycycle, 36000);
 duty(floatFromESP);

}
```

Kód pro příjem teploty, zobrazení na stránce

```
#include <espnow.h>
#include <ESP8266WiFi.h>
#include "ESPAsyncWebServer.h"
#include "ESPAsyncTCP.h"
#include <Arduino_JSON.h>

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

typedef struct struct_message {
  int id;
  float temp;
  float hum;
  unsigned int readingId;
} struct_message;

struct_message incomingReadings;

JSONVar board;

AsyncWebServer server(80);
AsyncEventSource events("/events");

void OnDataRecv(uint8_t * mac_addr, uint8_t *incomingData, uint8_t len) {

  char macStr[18];
```

```
  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));

  board["id"] = incomingReadings.id;
  board["temperature"] = incomingReadings.temp;
  board["humidity"] = incomingReadings.hum;
  board["readingId"] = String(incomingReadings.readingId);
  String jsonString = JSON.stringify(board);
  events.send(jsonString.c_str(), "new_readings", millis());

  Serial.printf("<Hello, %u, %4.2f>\n", incomingReadings.id, incomingReadings.temp);
}

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <title>ESP-NOW DASHBOARD</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="anonymous">
  <link rel="icon" href="data:,">
  <style>
    html {font-family: Arial; display: inline-block; text-align: center;}
    h1 {  font-size: 2rem;}
    body {  margin: 0;}
    .topnav { overflow: hidden; background-color: #2f4468; color: white; font-size: 1.7rem; }
    .content { padding: 20px; }
    .card { background-color: white; box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5); }
    .cards { max-width: 700px; margin: 0 auto; display: grid; grid-gap: 2rem; grid-template-columns: repeat(auto-fit, minmax(320px, 1fr)); }
```

```
    .reading { font-size: 2.8rem; }
    .timestamp { color: #bebebe; font-size: 1rem; }
    .card-title{ font-size: 1.2rem; font-weight : bold; }
    .card.temperature { color: #B10F2E; }
    .card.humidity { color: #50B8B4; }
  </style>
</head>
<body>
  <div class="topnav">
    <h1>ESP-NOW DASHBOARD</h1>
  </div>
  <div class="content">
    <div class="cards">
      <div class="card temperature">
        <p class="card-title"><i class="fas fa-thermometer-half"></i> BOARD #1 - TEMPERATURE</p><p><span class="reading"><span id="t1"></span> &deg;C</span></p><p class="timestamp">Last Reading: <span id="rt1"></span></p>
      </div>
      <div class="card humidity">
        <p class="card-title"><i class="fas fa-tint"></i> BOARD #1 - HUMIDITY</p><p><span class="reading"><span id="h1"></span> &percnt;</span></p><p class="timestamp">Last Reading: <span id="rh1"></span></p>
      </div>
      <div class="card temperature">
        <p class="card-title"><i class="fas fa-thermometer-half"></i> BOARD #2 - TEMPERATURE</p><p><span class="reading"><span id="t2"></span> &deg;C</span></p><p class="timestamp">Last Reading: <span id="rt2"></span></p>
      </div>
      <div class="card humidity">
        <p class="card-title"><i class="fas fa-tint"></i> BOARD #2 - HUMIDITY</p><p><span class="reading"><span id="h2"></span> &percnt;</span></p><p class="timestamp">Last Reading: <span id="rh2"></span></p>
```

```
    </div>
   </div>
  </div>
<script>
function getDateTime() {
  var currentdate = new Date();
  var datetime = currentdate.getDate() + "/"
  + (currentdate.getMonth()+1) + "/"
  + currentdate.getFullYear() + " at "
  + currentdate.getHours() + ":"
  + currentdate.getMinutes() + ":"
  + currentdate.getSeconds();
  return datetime;
}
if (!!window.EventSource) {
 var source = new EventSource('/events');

 source.addEventListener('open', function(e) {
  console.log("Events Connected");
 }, false);
 source.addEventListener('error', function(e) {
  if (e.target.readyState != EventSource.OPEN) {
    console.log("Events Disconnected");
  }
 }, false);

 source.addEventListener('message', function(e) {
  console.log("message", e.data);
 }, false);
```

```
 source.addEventListener('new_readings', function(e) {
  console.log("new_readings", e.data);
  var obj = JSON.parse(e.data);
  document.getElementById("t"+obj.id).innerHTML = obj.temperature.toFixed(2);
  document.getElementById("h"+obj.id).innerHTML = obj.humidity.toFixed(2);
  document.getElementById("rt"+obj.id).innerHTML = getDateTime();
  document.getElementById("rh"+obj.id).innerHTML = getDateTime();
 }, false);
}
</script>
</body>
</html>)rawliteral";

void setup() {
  Serial.begin(115200);

  WiFi.mode(WIFI_AP_STA);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);

  }


  if (esp_now_init() != 0) {

    return;
```

```
  }

  esp_now_register_recv_cb(OnDataRecv);

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

  events.onConnect([](AsyncEventSourceClient *client){
    if(client->lastId()){

    }

    client->send("hello!", NULL, millis(), 10000);
  });
  server.addHandler(&events);
  server.begin();
}

void loop() {
  static unsigned long lastEventTime = millis();
  static const unsigned long EVENT_INTERVAL_MS = 5000;
  if ((millis() - lastEventTime) > EVENT_INTERVAL_MS) {
    events.send("ping",NULL,millis());
    lastEventTime = millis();
  }
}
```

Kód ESP pro čtení teploty, poslání do druhého ESP

```
#include <espnow.h>
#include <ESP8266WiFi.h>
#include <DHT.h>
#define DHTTYPE DHT22
#define DHTPIN  2

// Set your Board ID (ESP32 Sender #1 = BOARD_ID 1, ESP32 Sender #2 = BOARD_ID 2, etc)
#define BOARD_ID 2
DHT dht(DHTPIN, DHTTYPE, 22);

uint8_t broadcastAddress[] = {0xE8, 0xDB, 0x84, 0x99, 0x91, 0x6E};

typedef struct struct_message {
   int id;
   float temp;
   float hum;
   int readingId;
} struct_message;

//Create a struct_message called myData
struct_message myData;

unsigned long previousMillis = 0;
const long interval = 10000;

unsigned int readingId = 0;
constexpr char WIFI_SSID[] = "*****";
```

```
int32_t getWiFiChannel(const char *ssid) {
  if (int32_t n = WiFi.scanNetworks()) {
    for (uint8_t i=0; i<n; i++) {
      if (!strcmp(ssid, WiFi.SSID(i).c_str())) {
        return WiFi.channel(i);
      }
    }
  }
  return 0;
}



float readTemperature() {
  float t = dht.readTemperature();
  return t;
}

float readHumidity() {
  float h = dht.readHumidity();
  return h;
}

void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("Last Packet Send Status: ");
  if (sendStatus == 0){
    Serial.println("Delivery success");
  }
  else{
```

```
   Serial.println("Delivery fail");
  }
}

void setup() {
  //Init Serial Monitor
  Serial.begin(115200);
  dht.begin();

  WiFi.mode(WIFI_STA);

  int32_t channel = getWiFiChannel(WIFI_SSID);

  WiFi.printDiag(Serial); // Uncomment to verify channel number before
  wifi_promiscuous_enable(1);
  wifi_set_channel(channel);
  wifi_promiscuous_enable(0);
  WiFi.printDiag(Serial); // Uncomment to verify channel change after

  // Init ESP-NOW
  if (esp_now_init() != 0) {
   Serial.println("Error initializing ESP-NOW");
   return;
  }

  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
   esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
```

```
  esp_now_register_send_cb(OnDataSent);

  esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);
}

void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    myData.id = BOARD_ID;
    myData.temp = readTemperature();
    myData.hum = readHumidity();
    myData.readingId = readingId++;

    esp_now_send(broadcastAddress, (uint8_t *) &myData, sizeof(myData));

    Serial.print("loop");
  }
}
```