**Ministerul Educației, Culturii și Cercetării a Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Ingineria Software și Automatică**

**Tema: Comunicare**

**Lucrare de laborator nr. 7**

**Disciplina: Internetul lucrurilor (IoT)**

**Student gr. TI-194: Ceban Vitalie**

**Conducător: Dinu Litra, asis. univ.**

**Chișinău 2022**

1. **Descrierea problemei**

Sa se realizeze o aplicație ce va implementa comunicațiile intre echipamente după cum urmează:

1. Protocol fizic de comunicare - Comunicarea intre DOUA Microcontrollere prin interfața I2C

* **MCU1** – implementează senzorul digital cu interfața I2C pentru senzorul ultrasonic HCS-04, unde se executa colectarea datelor de la interfața senzorului si se retransmite către interfața I2C la detectarea unei cereri de citire a datelor.
* **MCU2** – executa cererea prin interfața I2C către serosul digital ultrasonic (MCU+HCS-04).

1. Protocol logic de comunicare - cererea de date prin interfața serial, in format text respectând un protocol de comunicare care va avea câmpurile:

* indicator de start pachet
* indicator de sfârșit
* contorizare pachete
* ID emițător
* ID receptor
* tipul pachetului
* <alte câmpuri opțional>
* date pachet - Payload
* suma de control - suma tuturor valorilor numerice din pachet

Cererile venite din interfața seriala vor fi verificate după patern, si in caz de pachet valid se va executa comanda și se va răspunde cu un pachet conform aceluiași protocol.

Comanda obligatorie pentru implementare este cererea de date de la senzorul digital implementat în punctul 1. Să se implementeze încă o comandă la alegere, pentru diversitate.

1. Realizarea conexiunii și raportării datelor achiziționate către un MQTT server cum ar fi https://thingsboard.io sau echivalent.
2. **Schema**

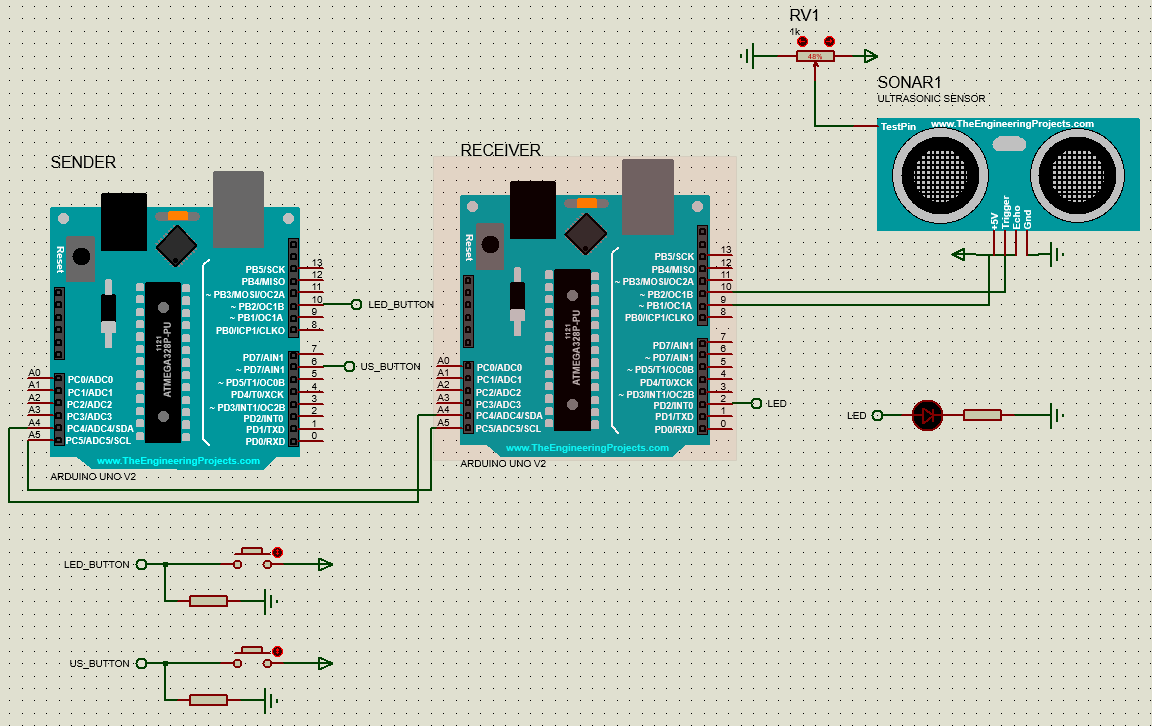


Figura 1 – Schema circuitului

1. **Implementarea**

#define START\_INDEX 0

#define SENDER\_INDEX 1

#define RECEIVER\_INDEX 2

#define COUNT\_INDEX 3

#define TYPE\_INDEX 4

#define DATA\_INDEX 5

#define CRC\_INDEX 6

#define STOP\_INDEX 7

#define START\_CONDITION 'S'

#define STOP\_CONDITION 'E'

#define PACKAGE\_LENGTH 8

class Package {

int getCrc();

void validate();

public:

enum Type {

CORRUPTED = 0,

ULTRASONIC\_REQUEST = 1,

ULTRASONIC\_RESPONSE = 2,

LED\_REQUEST = 3,

LED\_RESPONSE = 4,

TEST\_REQUEST = 5,

TEST\_RESPONSE = 6

};

int startCondition;

int senderId;

int receiverId;

int packageNumber;

Type type;

int payload;

int crc;

int stopCondition;

bool isValid;

Package();

Package(int\* rawPackage);

Package(int senderId, int receiverId, int packageNumber, Type type, int payload);

static Package createInvalidPackage();

};

Package::Package(int senderId, int receiverId, int packageNumber, Package::Type type, int payload) {

this->startCondition = START\_CONDITION;

this->senderId = senderId;

this->receiverId = receiverId;

this->packageNumber = packageNumber;

this->type = type;

this->payload = payload;

this->crc = this->getCrc();

this->stopCondition = STOP\_CONDITION;

this->isValid = true;

}

Package::Package(int\* rawPackage) {

this->startCondition = rawPackage[START\_INDEX];

this->senderId = rawPackage[SENDER\_INDEX];

this->receiverId = rawPackage[RECEIVER\_INDEX];

this->packageNumber = rawPackage[COUNT\_INDEX];

this->type = (Package::Type)rawPackage[TYPE\_INDEX];

this->payload = rawPackage[DATA\_INDEX];

this->crc = rawPackage[CRC\_INDEX];

this->stopCondition = rawPackage[STOP\_INDEX];

this->isValid = true;

validate();

}

Package::Package() {

this->type = Package::Type::CORRUPTED;

this->isValid = false;

}

Package Package::createInvalidPackage() {

return Package();

}

void Package::validate() {

if (this->startCondition == START\_CONDITION && this->stopCondition == STOP\_CONDITION

&& this->crc == this->getCrc()) {

return;

}

this->isValid = false;

}

int Package::getCrc() {

return this->senderId + this->receiverId + this->packageNumber + this->type + this->payload;

}

class Comms {

int address;

int arduinoId;

void requestRepose();

public:

Package receivedPackage;

Comms(int address, int arduinoId);

Comms(int address, int arduinoId, void (\*onReceive)(int), void (\*onRequest)(void));

void sendRequest(int receiverId, Package::Type type);

void sendResponse(int receiverId, Package::Type type, int data);

void updateReceivedPackage(int size);

};

Comms::Comms(int address, int arduinoId) {

this->address = address;

this->arduinoId = arduinoId;

this->receivedPackage = Package::createInvalidPackage();

Wire.begin();

}

Comms::Comms(int address, int arduinoId, void (\*onReceive)(int), void (\*onRequest)(void)) {

this->address = address;

this->arduinoId = arduinoId;

this->receivedPackage = Package::createInvalidPackage();

Wire.begin(this->address);

Wire.onReceive(onReceive);

Wire.onRequest(onRequest);

}

void Comms::sendRequest(int receiverId, Package::Type type) {

int packageNumber = this->receivedPackage.isValid ?

this->receivedPackage.packageNumber + 1 : 1;

Package package(this->arduinoId, receiverId, packageNumber, type, 0);

Wire.beginTransmission(this->address);

Wire.write(package.startCondition);

Wire.write(package.senderId);

Wire.write(package.receiverId);

Wire.write(package.packageNumber);

Wire.write(package.type);

Wire.write(package.payload);

Wire.write(package.crc);

Wire.write(package.stopCondition);

Wire.endTransmission();

int delayTime = type == Package::Type::ULTRASONIC\_REQUEST ? 100 : 50;

delay(delayTime);

this->requestRepose();

}

void Comms::requestRepose() {

int bytes = Wire.requestFrom(this->address, PACKAGE\_LENGTH);

this->updateReceivedPackage(bytes);

}

void Comms::sendResponse(int receiverId, Package::Type type, int data) {

int packageNumber = this->receivedPackage.isValid ?

this->receivedPackage.packageNumber + 1 : 1;

Package package(this->arduinoId, receiverId, packageNumber, type, data);

Wire.write(package.startCondition);

Wire.write(package.senderId);

Wire.write(package.receiverId);

Wire.write(package.packageNumber);

Wire.write(package.type);

Wire.write(package.payload);

Wire.write(package.crc);

Wire.write(package.stopCondition);

}

void Comms::updateReceivedPackage(int size) {

bool isValid = true;

if (size != PACKAGE\_LENGTH) isValid = false;

if (!isValid) {

this->receivedPackage = Package::createInvalidPackage();

return;

}

int received[PACKAGE\_LENGTH];

for(int i = 0; i < size; i++) {

received[i] = Wire.read();

if (i == START\_INDEX && received[i] != START\_CONDITION) {

isValid = false;

break;

}

}

if (!isValid) {

this->receivedPackage = Package::createInvalidPackage();

return;

}

this->receivedPackage = Package(received);

}

void onReceive(int bytes);

void onRequest();

Comms comms(ADDRESS, MY\_ID, &onReceive, &onRequest);

void onReceive(int bytes) {

comms.updateReceivedPackage(bytes);

}

void onRequest() {

if (!comms.receivedPackage.isValid) {

comms.sendResponse(MCU1, Package::Type::CORRUPTED, 0);

return;

}

switch (comms.receivedPackage.type)

{

case Package::Type::LED\_REQUEST:

executeLedCommand();

break;

case Package::Type::ULTRASONIC\_REQUEST:

executeUltrasonicCommand();

break;

case Package::Type::TEST\_REQUEST:

comms.sendResponse(MCU1, Package::Type::TEST\_RESPONSE, 2);

break;

}

}

void executeLedCommand() {

ledState = !ledState;

digitalWrite(LED\_PIN, ledState);

comms.sendResponse(MCU1, Package::Type::LED\_RESPONSE, ledState);

}

void executeUltrasonicCommand() {

unsigned int rawDistanceCm = microsecondsToCentimeters(getUltraSonicDuration());

unsigned int saturatedDistanceCm = saturate(rawDistanceCm, MIN\_DISTANCE\_CM, MAX\_DISTANCE\_CM);

comms.sendResponse(MCU1, Package::Type::ULTRASONIC\_RESPONSE, saturatedDistanceCm);

}

#define SSID "name"

#define PASSWORD "pass"

#define TOKEN "YOUR\_ACCESS\_TOKEN"

#define THINGSBOARD\_SERVER "thingsboard.cloud"

void initWiFi()

{

Serial.println("Connecting to AP ...");

// attempt to connect to WiFi network

WiFi.begin(SSID, PASSWORD);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("Connected to AP");

}

void reconnect() {

Serial.println("Reconnecting to AP ...");

// Loop until we're reconnected

int status = WiFi.status();

if ( status != WL\_CONNECTED) {

WiFi.begin(SSID, PASSWORD);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("Connected to AP");

}

}

void requesTestPackage(uint8\_t \*data, size\_t len) {

String d = "";

for(int i=0; i < len; i++){

d += char(data[i]);

}

WebSerial.println("Sending request");

comms.sendRequest(MCU2, Package::Type::TEST\_REQUEST);

WebSerial.println("Received:");

WebSerial.println(comms.receivedPackage.senderId);

WebSerial.println(comms.receivedPackage.receiverId);

WebSerial.println(comms.receivedPackage.packageNumber);

WebSerial.println(comms.receivedPackage.type);

WebSerial.println(comms.receivedPackage.payload);

WebSerial.println(comms.receivedPackage.crc);

WebSerial.println(comms.receivedPackage.type == Package::Type::TEST\_RESPONSE ? "Good test package" : "Bad test package");

WebSerial.println("==================");

}

void checkLedButtonAndSend() {

bool buttonState = digitalRead(LED\_BUTTON);

if (buttonState && lastLedButtonState == false) {

WebSerial.println("LED");

comms.sendRequest(MCU2, Package::Type::LED\_REQUEST);

};

lastLedButtonState = buttonState;

}

void checkUltrasonicButtonAndSend() {

bool buttonState = digitalRead(ULTRASONIC\_BUTTON);

if (buttonState && lastUltrasonicButtonState == false) {

WebSerial.println("Ultra");

comms.sendRequest(MCU2, Package::Type::ULTRASONIC\_REQUEST);

};

lastUltrasonicButtonState = buttonState;

}

void checkAndsendToThingsBoard() {

if (WiFi.status() != WL\_CONNECTED) {

reconnect();

}

if (!tb.connected()) {

// Connect to the ThingsBoard

WebSerial.print("Connecting to: ");

WebSerial.print(THINGSBOARD\_SERVER);

WebSerial.print(" with token ");

WebSerial.println(TOKEN);

if (!tb.connect(THINGSBOARD\_SERVER, TOKEN)) {

WebSerial.println("Failed to connect");

return;

}

}

if (!comms.receivedPackage.isValid) return;

if (lastSentToThingsBoardPackageId == comms.receivedPackage.packageNumber) return;

if (comms.receivedPackage.type == Package::Type::TEST\_RESPONSE) return;

WebSerial.println(comms.receivedPackage.senderId);

WebSerial.println(comms.receivedPackage.receiverId);

WebSerial.println(comms.receivedPackage.packageNumber);

WebSerial.println(comms.receivedPackage.type);

WebSerial.println(comms.receivedPackage.payload);

WebSerial.println(comms.receivedPackage.crc);

WebSerial.println("==================");

switch (comms.receivedPackage.type)

{

case Package::Type::LED\_RESPONSE:

WebSerial.println("Sending led data");

tb.sendTelemetryInt(LED\_TELEMETRY\_TAG, comms.receivedPackage.payload);

break;

case Package::Type::ULTRASONIC\_RESPONSE:

WebSerial.println("Sending ultrasonic data");

tb.sendTelemetryInt(ULTRASONIC\_TELEMETRY\_TAG, comms.receivedPackage.payload);

break;

}

lastSentToThingsBoardPackageId = comms.receivedPackage.packageNumber;

}

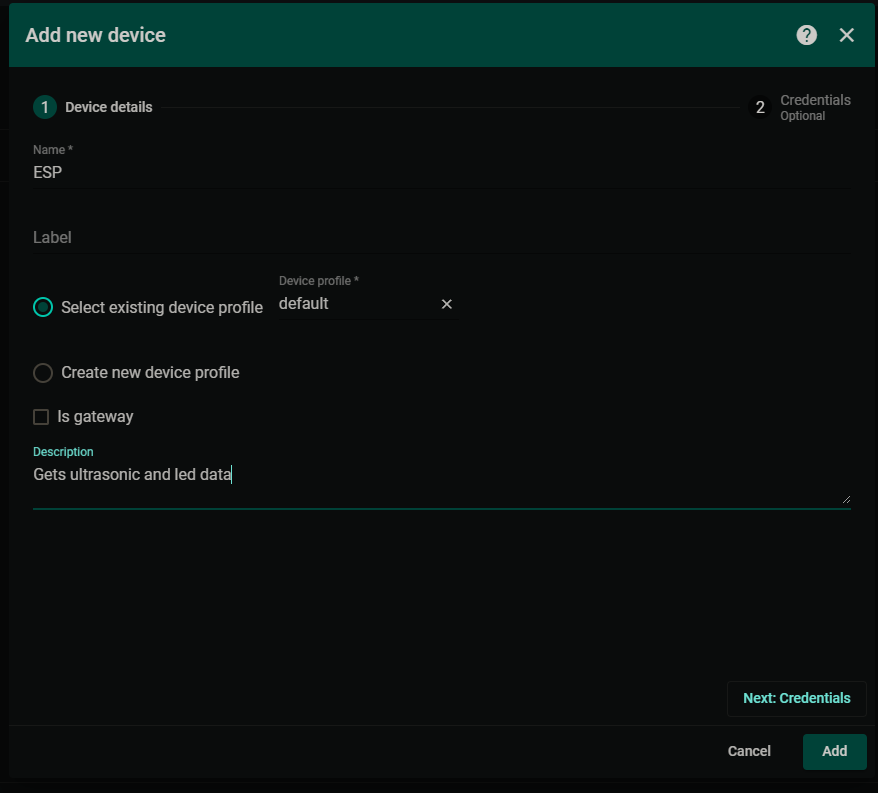


Figura 2 – Crearea dispozitivului

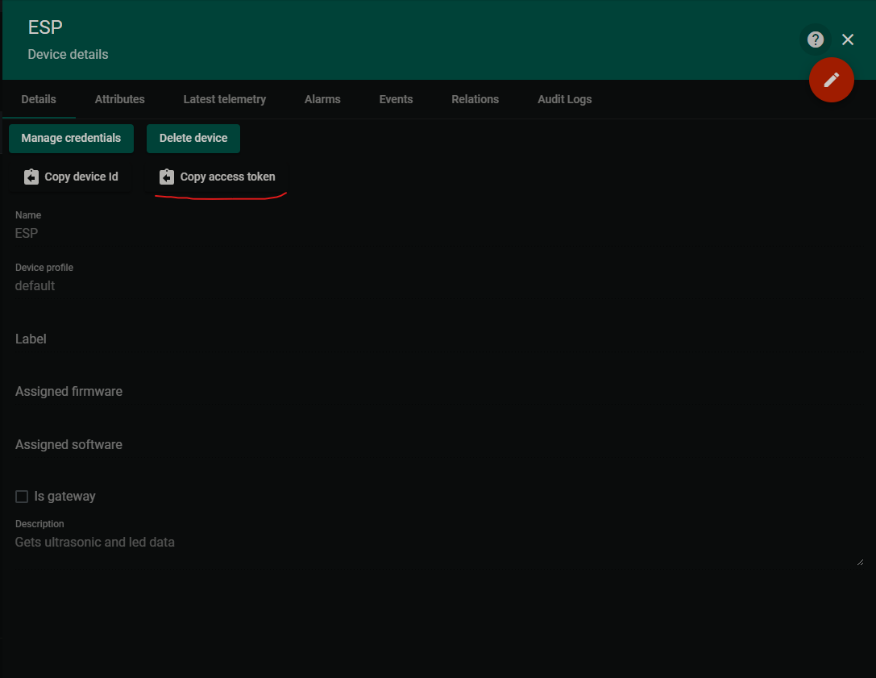


Figura 3 – Detaliile dispozitivului

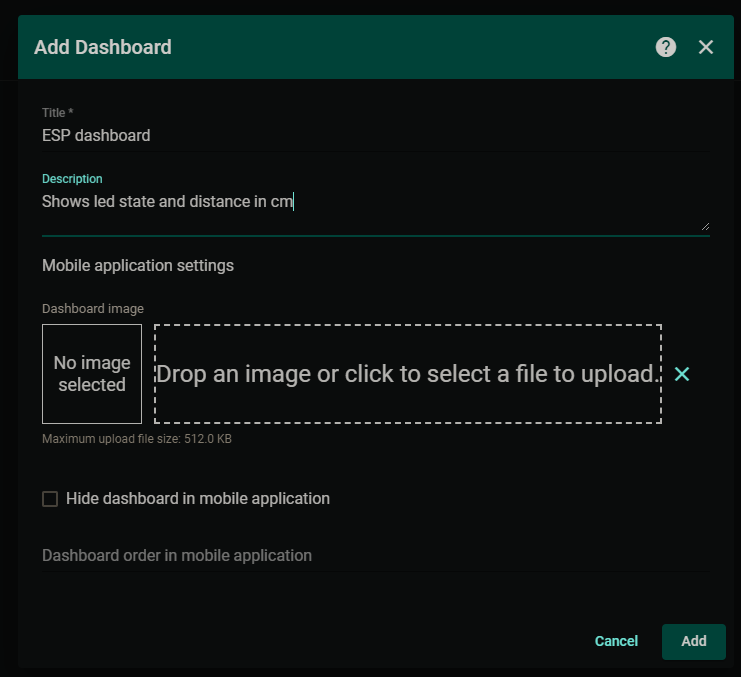


Figura 4 – Creare dashboard

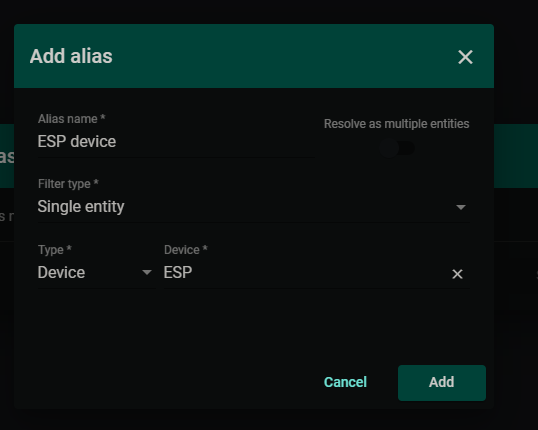


Figura 4 – Adăugare dispozitiv

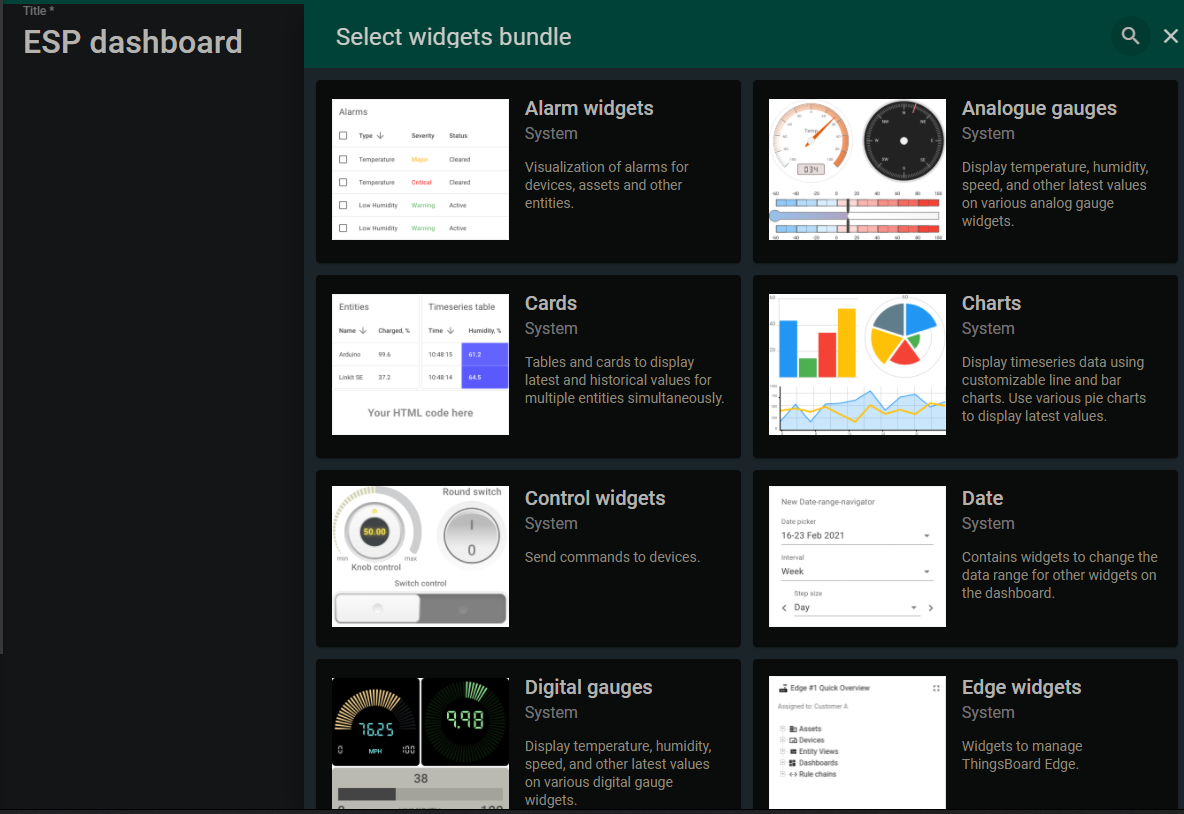
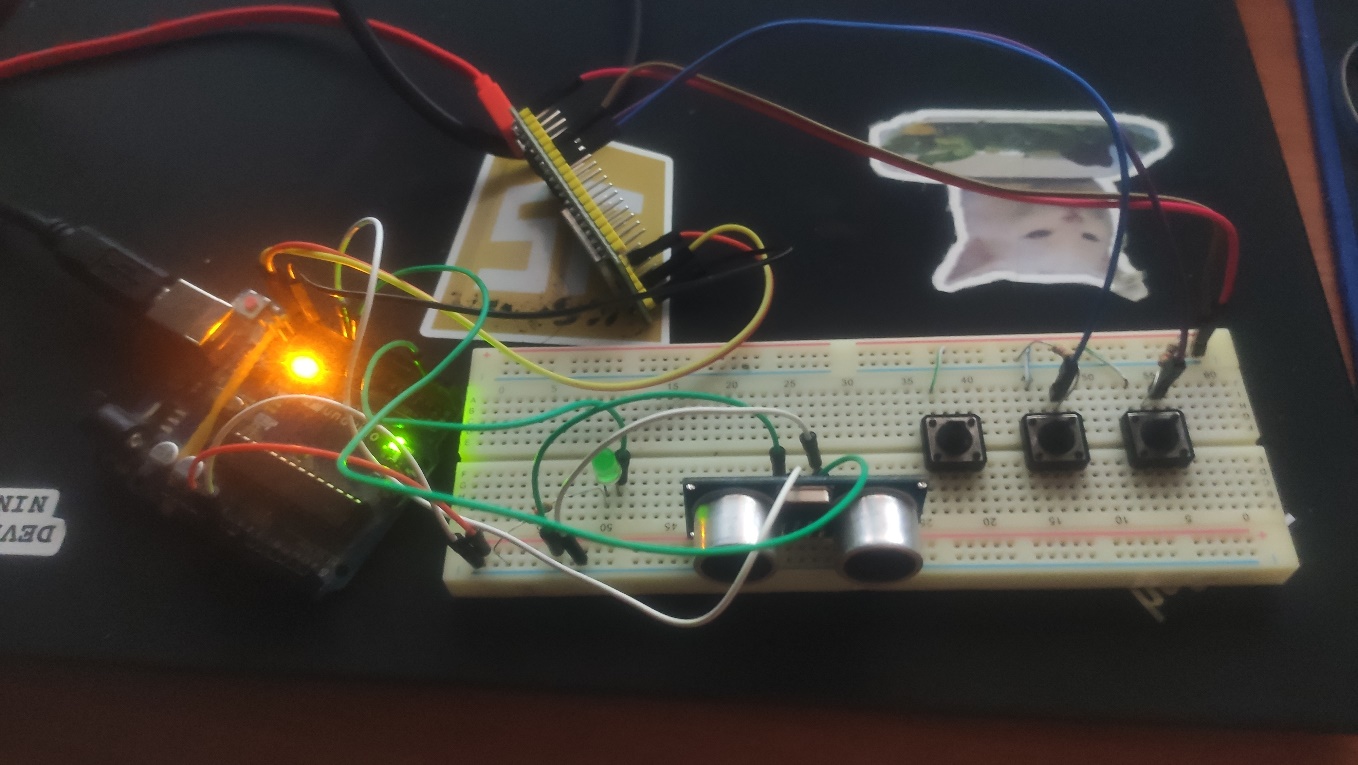


Figura 5 – Adăugare widget

1. **Rezultatul**



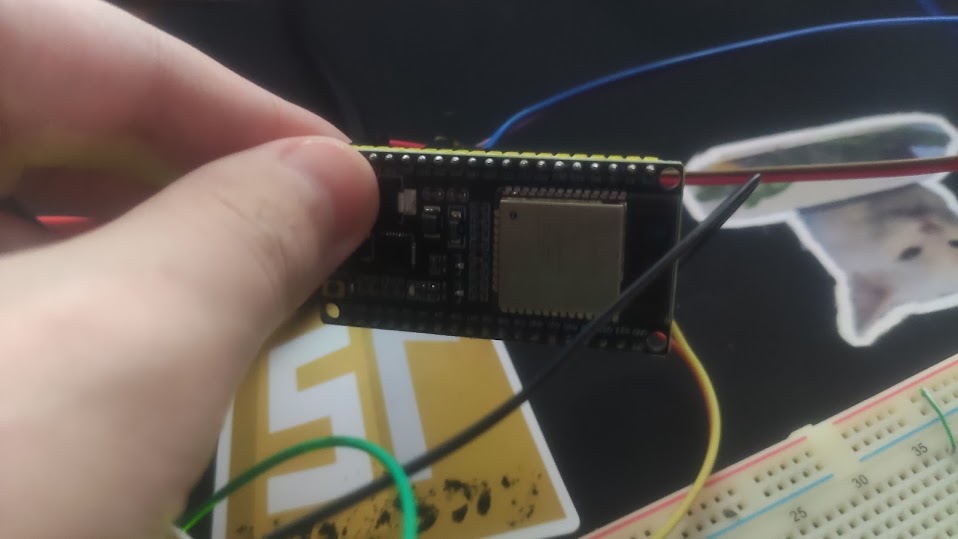


Figura 6 – Sistemul implementat

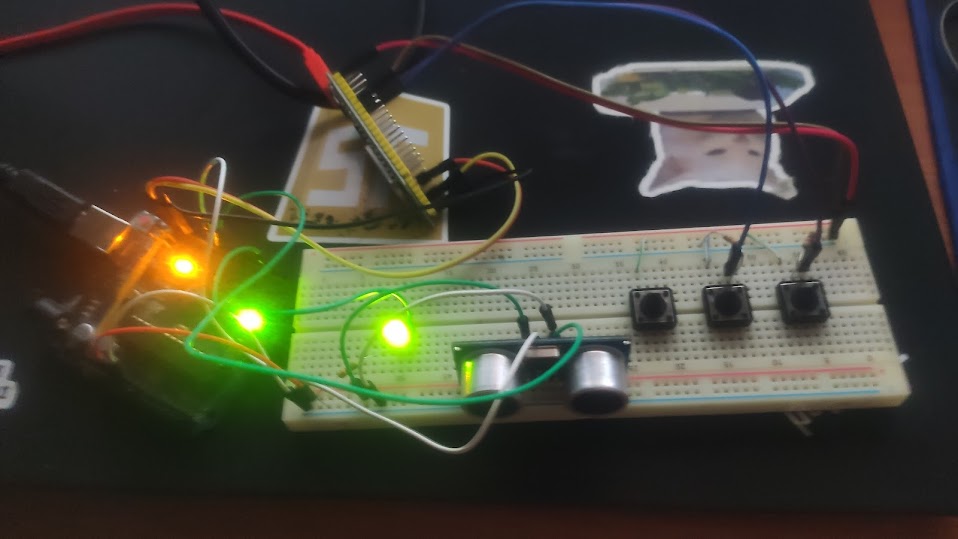


Figura 7 – Led-ul în stare On

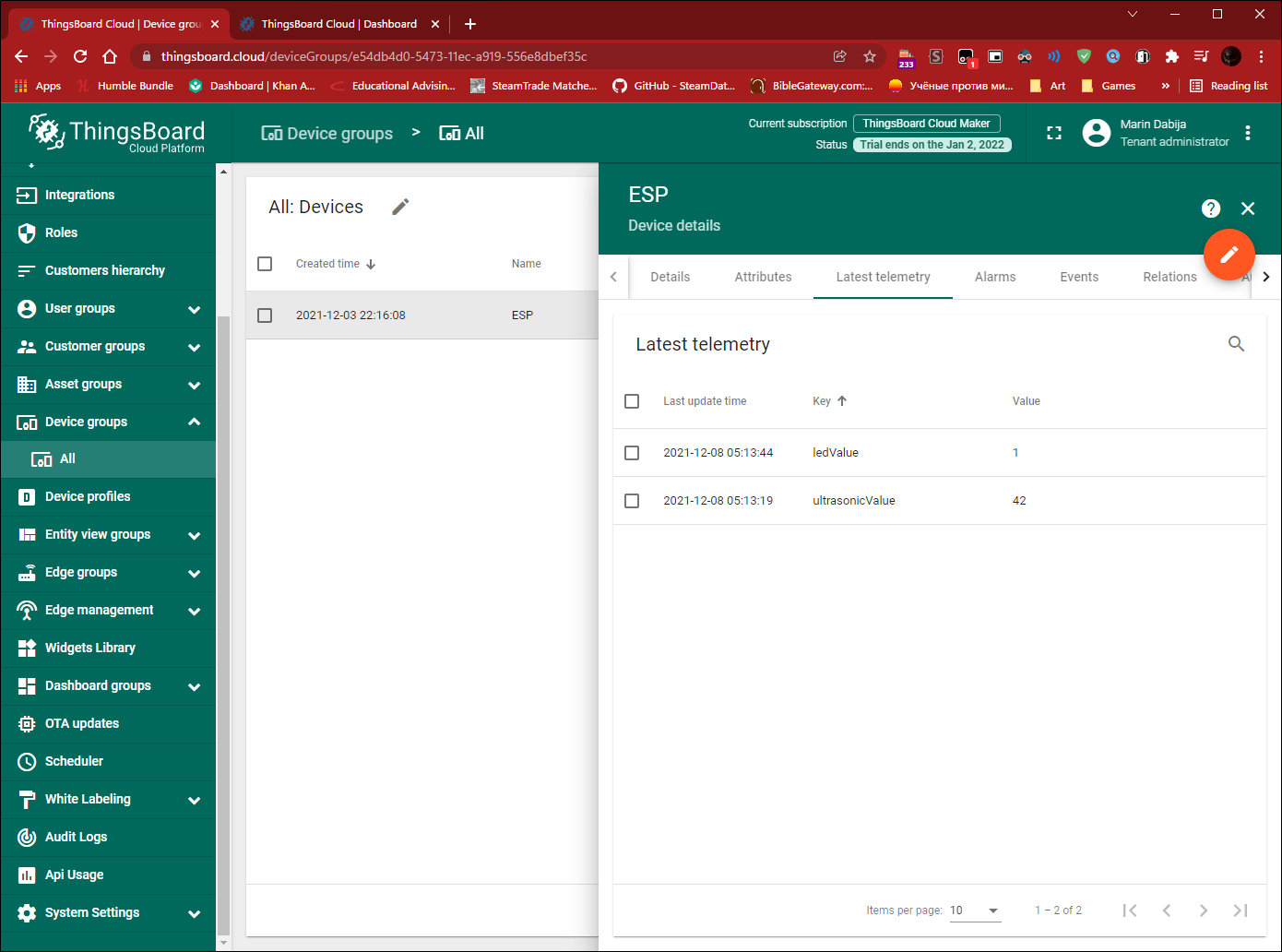


Figura 8 – Vizualizarea ultimelor date telemetrice recepționate

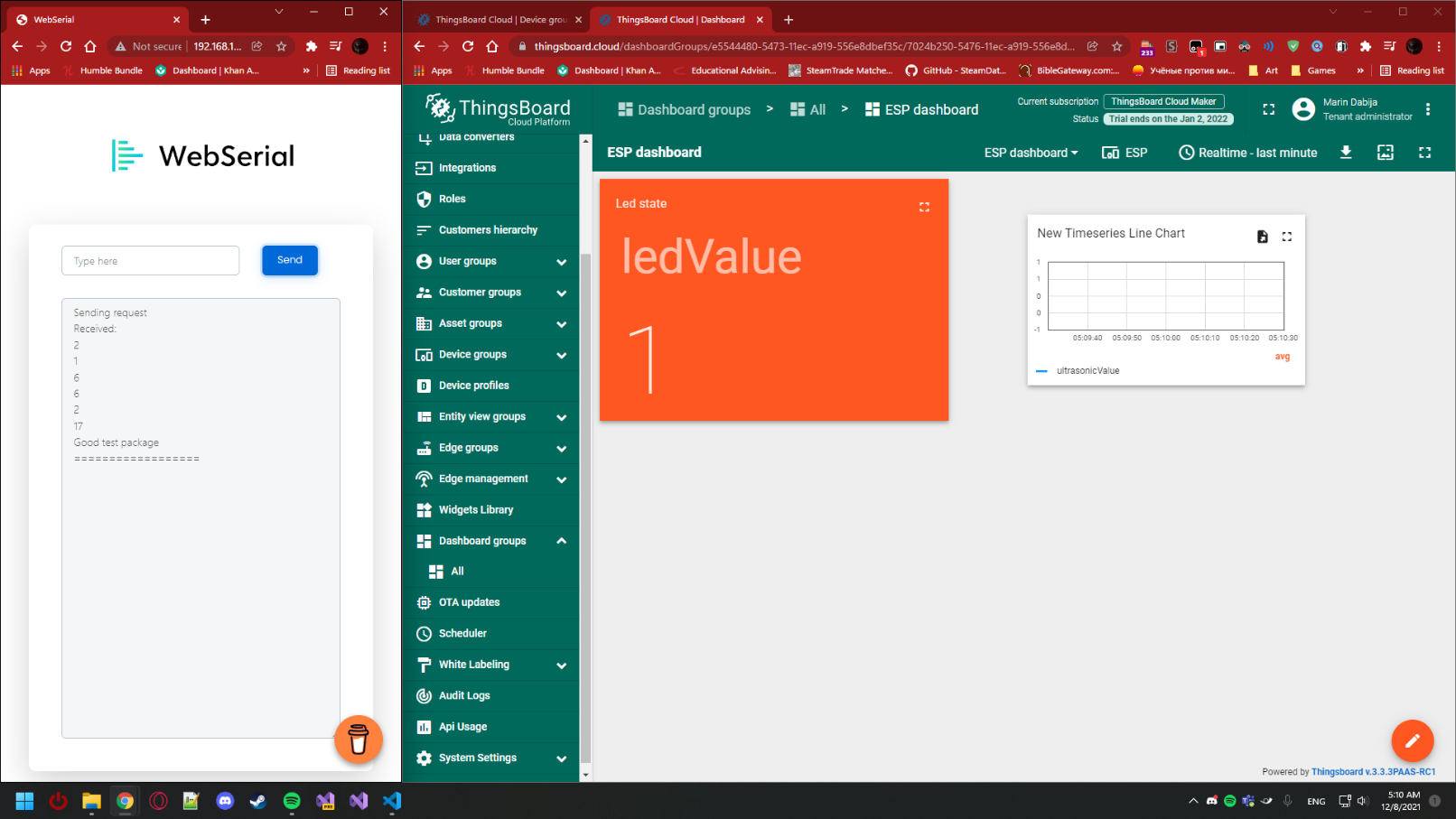


Figura 9 – Trimiterea unui pachet de testare și recepționarea răspunsului

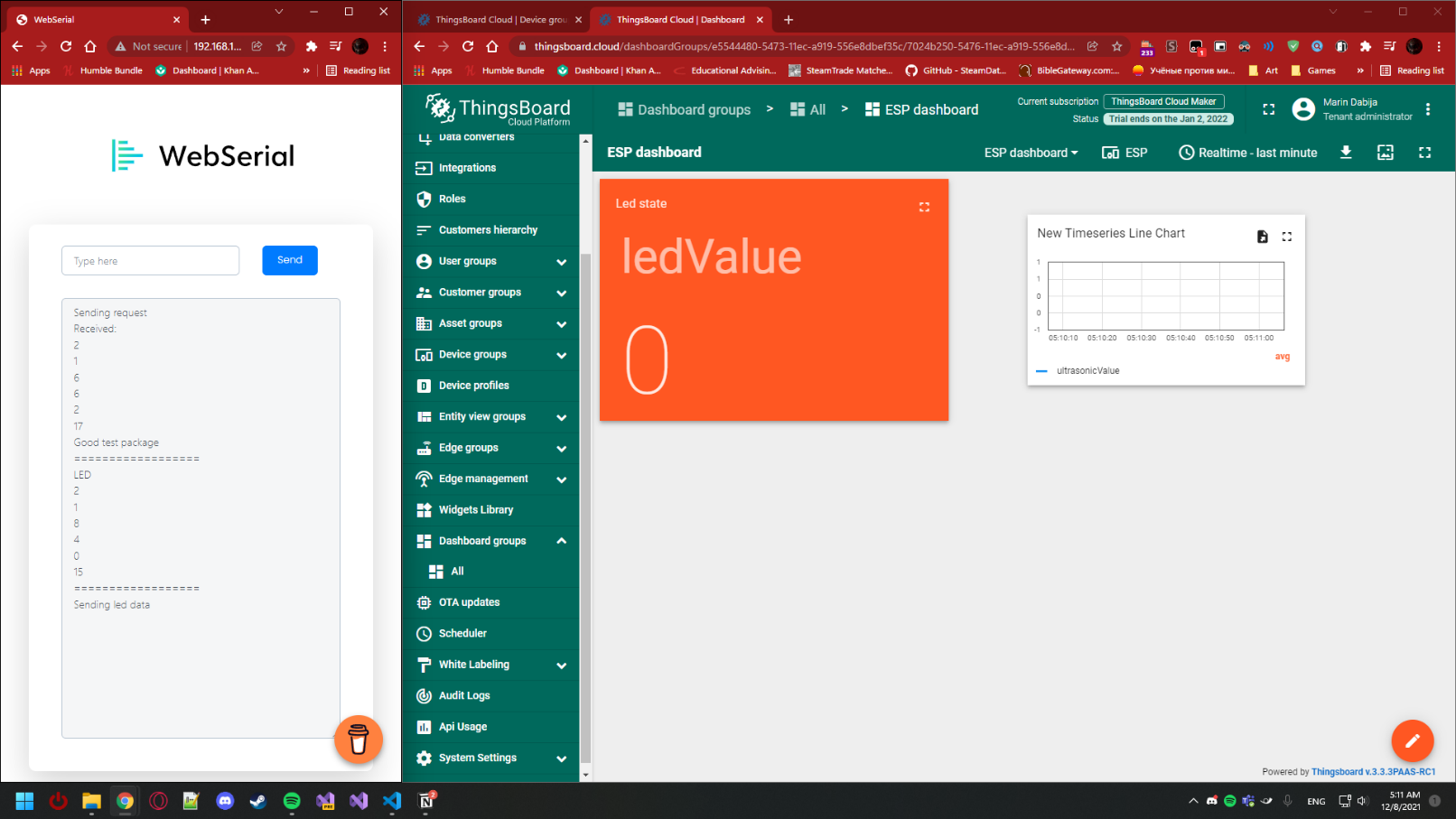


Figura 10 – Trimiterea unui pachet Led și recepționarea răspunsului

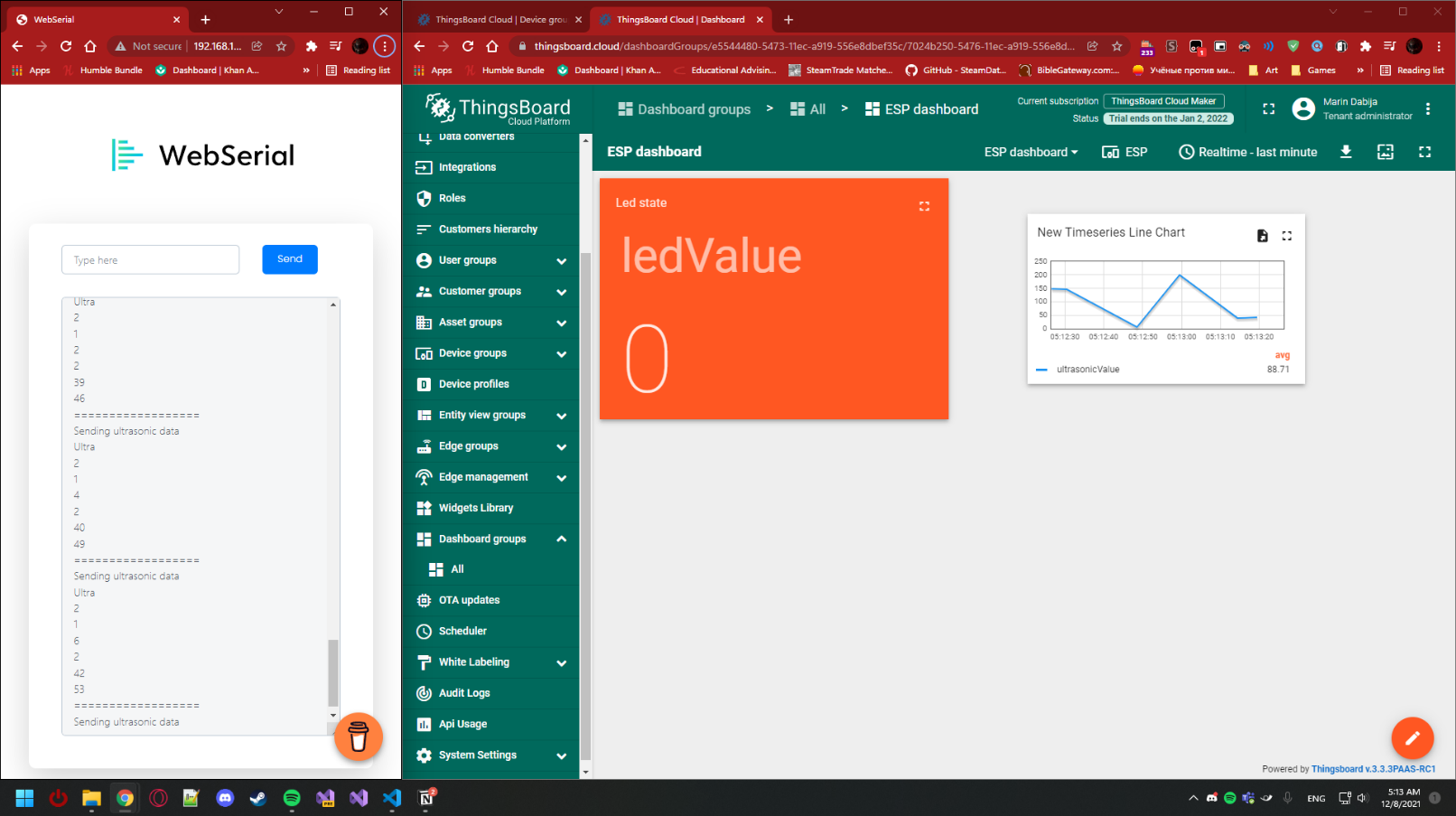


Figura 11 – Trimiterea pachetelor Ultrasonic și recepționarea răspunsurilor

**Concluzie**

In urma realizării acestei lucrări de laborator am elaborat o aplicație care implementează comunicațiile intre echipamente după 2 metode:

1. Protocol fizic de comunicare - Comunicarea intre DOUA Microcontrollere prin interfața I2C

* **MCU1** – implementează senzorul digital cu interfața I2C pentru senzorul ultrasonic HCS-04, unde se executa colectarea datelor de la interfața senzorului si se retransmite către interfața I2C la detectarea unei cereri de citire a datelor.
* **MCU2** – executa cererea prin interfața I2C către serosul digital ultrasonic (MCU+HCS-04).

2. Protocol logic de comunicare - cererea de date prin interfața serial, in format text respectând un protocol de comunicare care va avea câmpurile:

* indicator de start pachet
* indicator de sfârșit
* contorizare pachete
* ID emițător
* ID receptor
* tipul pachetului
* <alte câmpuri opțional>
* date pachet - Payload
* suma de control - suma tuturor valorilor numerice din pachet