

Projeto Final

Sistema de Monitoramento da Qualidade
do Ar com ESP8266 e Bitdoglab Utilizando
Blockchain IoT

Nome: Tiago Lauriano Copelli

1. Introdução

O monitoramento da qualidade do ar é uma preocupação crescente nas sociedades modernas, especialmente em ambientes urbanos, onde a poluição pode afetar a saúde pública e o bem-estar dos cidadãos. Para atender a essa demanda, a utilização de sensores que medem diferentes parâmetros ambientais, como temperatura, umidade e qualidade do ar, tornou-se essencial. Entre os sensores mais comuns para este tipo de monitoramento estão o DHT11 e o MQ135. O DHT11 é utilizado para medir a temperatura e a umidade, enquanto o MQ135 é projetado para detectar gases nocivos, como amônia, dióxido de carbono (CO₂), álcool e outros poluentes presentes no ar.

O sistema em questão utiliza as placas ESP8266 e Bitdoglab para coletar e processar esses dados, além de enviá-los para uma rede blockchain, no caso a loTeX, que garante a segurança, integridade e rastreabilidade das informações.

2. Objetivo

O objetivo deste projeto é desenvolver um sistema de monitoramento da qualidade do ar utilizando as placas ESP8266 e Bitdoglab, conectadas aos sensores DHT11 e MQ135, para coletar dados de temperatura, umidade e gases poluentes. Esses dados serão enviados de forma segura e imutável para a blockchain IoT, garantindo transparência, integridade e rastreabilidade das informações, contribuindo para o controle ambiental em tempo real.

3. Justificativa

A justificativa deste trabalho reside na crescente necessidade de monitoramento ambiental, especialmente da qualidade do ar, devido aos impactos da poluição na saúde pública e no meio ambiente. A utilização de sensores acessíveis, como o DHT11 e o MQ135, aliados à tecnologia blockchain, oferece uma solução eficiente e segura para a coleta, armazenamento e análise de dados em tempo real, promovendo o controle e a transparência das informações, fundamentais para a implementação de políticas públicas e ações de preservação ambiental.

4. Originalidade

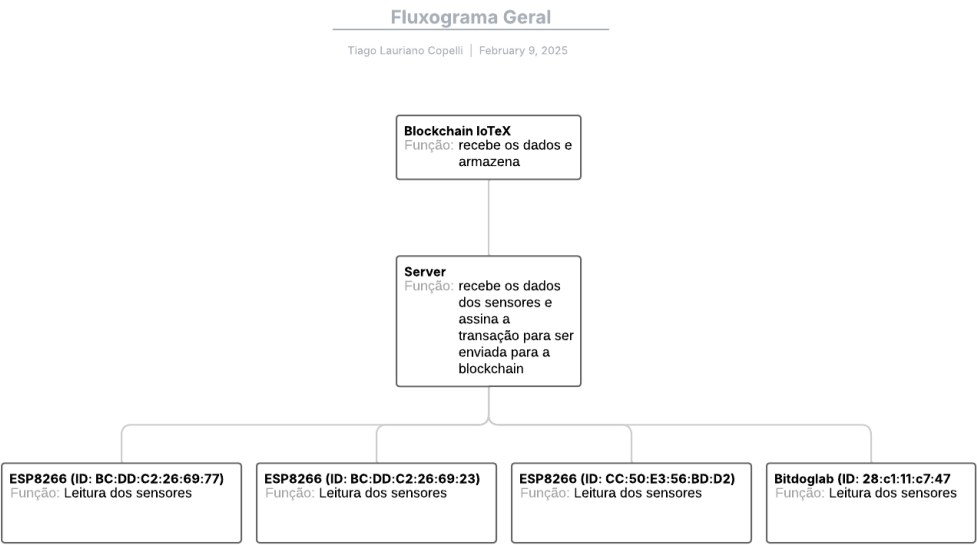
O OpenSensors é uma plataforma de IoT que integra sensores físicos com soluções em nuvem, permitindo a coleta, armazenamento e análise de dados em tempo real. Seu principal objetivo é facilitar o monitoramento ambiental e a automação inteligente, utilizando uma rede de sensores conectados à internet. A plataforma oferece uma interface gráfica intuitiva, fácil de usar, para visualização e análise dos dados. É compatível com diversos sensores e permite a configuração de automações baseadas em regras. O armazenamento de dados é feito na nuvem, facilitando o acesso remoto. Entre suas vantagens estão a escalabilidade e a possibilidade de personalização, enquanto as desvantagens incluem a necessidade de configuração técnica avançada em casos mais complexos. O OpenSensors é ideal para projetos de monitoramento de larga escala e automação inteligente.

Enquanto o OpenSensors é uma plataforma genérica para IoT, o meu projeto se diferencia pelo uso de blockchain IoT, pela validação local de dados e pela falta de dependência de um serviço centralizado de nuvem. Esse modelo oferece maior transparência, segurança e confiabilidade, garantindo que os dados ambientais registrados não possam ser alterados.

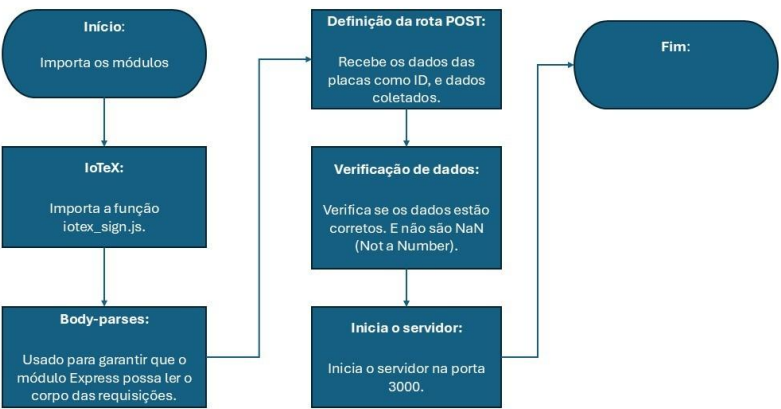
Portanto, embora existam projetos correlatos, o projeto é único na forma como integra IoT, processamento local e blockchain para um monitoramento ambiental seguro e imutável.

5. Fluxograma

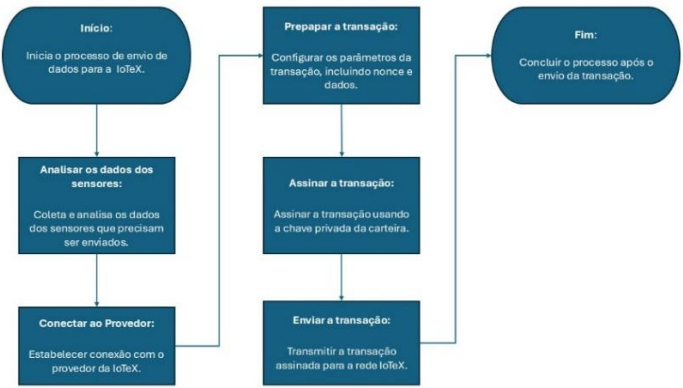
5.1 Fluxograma do projeto



5.2 Fluxograma do server.js

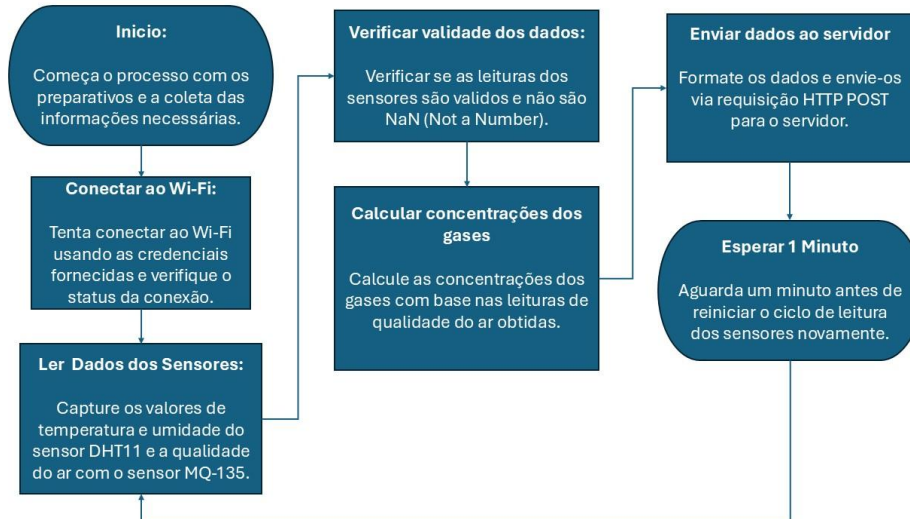


5.3 Fluxograma da função iotex_sign.js



5.4 Fluxograma das placas ESP8266 e da Bitdoglab

A única diferença nos códigos das placas são as bibliotecas e os pinos que conectam nos sensores.



6. Códigos

6.1 Server.js

```
import express from 'express'; // Usando import para módulos no formato ES6
import bodyParser from 'body-parser';
import { sendTransactionToIoTeX } from './iotex_sign.js'; // Importa a função do iotex_sign.js

const app = express();
const port = 3000;

// Usando body-parser para garantir que o Express possa ler o corpo das requisições
app.use(bodyParser.json());

// Defina a rota POST para /api/submitData
app.post('/api/submitData', async (req, res) => {
  const {
    id, // Recebendo o ID da placa
    temperatura,
    umidade,
    qualidadeAr,
    alcool,
    benzeno,
    CO2,
    fumaca
  } = req.body;

  // Verificação de dados
  if (
    id === undefined || // Verificando o ID
    temperatura === undefined ||
    umidade === undefined ||
    qualidadeAr === undefined ||
    alcool === undefined ||
    benzeno === undefined ||
    CO2 === undefined ||
    fumaca === undefined
  ) {
    return res.status(400).send('Dados incompletos ou inválidos!');
  }

  console.log(`
    ID da Placa: ${id}
    Temperatura: ${temperatura}°C
    Umidade: ${umidade}%
    Qualidade do ar (MQ135): ${qualidadeAr}
    Álcool: ${alcool}
    Benzeno: ${benzeno}
    CO2: ${CO2}
    Fumaça: ${fumaca}
  `);
  // Preparando os dados para enviar para a IoTeX
  const sensorData = {
    id, // Incluindo o ID no pacote de dados
    temperatura,
    umidade,
    qualidadeAr,
    alcool,
    benzeno,
    CO2,
    fumaca
  };
  // Enviar os dados para a IoTeX
  await sendTransactionToIoTeX(sensorData);

  // Respondendo para o ESP8266
  res.status(200).send('Dados recebidos com sucesso e enviados para IoTeX!');
});

// Inicia o servidor na porta 3000
app.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${port}`);
});
```


6.2 iotex_sign.js

```
import { ethers } from 'ethers';
import fs from 'fs';

// Função para assinar e enviar a transação para a IoTeX
async function sendTransactionToIoTeX(sensorData) {
  const provider = new ethers.JsonRpcProvider('https://babel-api.testnet.iotex.io');
  const privateKey = 'Ocultada'; // Chave privada ocultada por segurança. Com ela, é possível
  acessar e realizar transações na carteira IoTeX. Para criar uma carteira, acesse o site da IoTeX e
  gere um par de chaves (pública e privada), ambas necessárias para este projeto.
  const wallet = new ethers.Wallet(privateKey, provider);

  try {
    console.log('Enviando dados para a IoTeX...');

    const nonce = await provider.getTransactionCount(wallet.address, 'latest');
    const gasPrice = await provider.getFeeData();

    const dataHex = ethers.hexlify(ethers.toUtf8Bytes(JSON.stringify(sensorData)));

    const tx = {
      to: wallet.address,
      value: ethers.parseEther("0"), // Nenhum valor está sendo transferido
      gasLimit: 100000,
      gasPrice: gasPrice.gasPrice,
      data: dataHex,
      nonce: nonce,
      chainId: 4690, // IoTeX Testnet Chain ID
    };

    const signedTx = await wallet.signTransaction(tx);
    const txResponse = await provider.broadcastTransaction(signedTx);
    console.log('Transação enviada para IoTeX! Hash:', txResponse);
  } catch (error) {
    console.error('Erro ao enviar transação para IoTeX:', error);
  }
}

// Exporte a função corretamente
export { sendTransactionToIoTeX };
```

6.3 Código ESP8266 utilizando a plataforma Arduino IDE

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <DHT.h>

// Configuração dos sensores
#define DHTPIN 12 // Pino digital 12 para o DHT11
#define DHTTYPE DHT11 // Tipo do sensor DHT11
#define MQ135PIN A0 // Pino analógico do MQ135

DHT dht(DHTPIN, DHTTYPE);

// Configuração do Wi-Fi
const char* ssid = "COPELLI-2G"; // Nome da rede wifi
const char* password = "copelli1"; // Senha da rede wifi

const char* serverUrl = "http://192.168.0.38:3000/api/submitData"; // IP do servidor local

WiFiClient client;
HTTPClient http;

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);

  Serial.print("Conectando ao Wi-Fi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\n Conectado ao Wi-Fi!");
```

```

    dht.begin();
}

float readMQ135() {
    int sensorValue = analogRead(MQ135PIN); // Leitura do valor analógico do MQ-135
    // Mapeamento do valor do MQ-135 para estimativa de concentração de gases
    return sensorValue;
}

float getGasConcentration(int mq135Value) {
    // Mapeamento fictício de valores para cada gás
    float alcool = mq135Value * 0.1; // Fictício, apenas para teste
    float benzeno = mq135Value * 0.2;
    float co2 = mq135Value * 0.05;
    float fumaca = mq135Value * 0.15;

    // Retorna os valores dos gases
    Serial.print("Álcool: "); Serial.println(alcool);
    Serial.print("Benzeno: "); Serial.println(benzeno);
    Serial.print("CO2: "); Serial.println(co2);
    Serial.print("Fumaça: "); Serial.println(fumaca);

    return mq135Value; // Retorna o valor do MQ135 para qualidade do ar
}

void loop() {
    if (WiFi.status() == WL_CONNECTED) {
        // Leitura dos sensores
        float temperatura = dht.readTemperature();
        float umidade = dht.readHumidity();
        int qualidadeAr = readMQ135(); // Leitura do MQ135

        // Verifica se a leitura do DHT11 foi bem-sucedida
        if (isnan(temperatura) || isnan(umidade)) {
            Serial.println("Erro ao ler o sensor DHT11!");
            return;
        }

        // Imprime os dados no Serial Monitor
        Serial.print("Temperatura: ");
        Serial.print(temperatura);
        Serial.println(" °C");

        Serial.print("Umidade: ");
        Serial.print(umidade);
        Serial.println(" %");

        Serial.print("Qualidade do ar (MQ135): ");
        Serial.println(qualidadeAr);

        // Analisando os gases detectados
        float alcool = getGasConcentration(qualidadeAr);
        float benzeno = alcool * 2;
        float co2 = alcool * 0.5;
        float fumaca = alcool * 0.8;

        // Enviando os dados para o servidor local
        http.begin(client, serverUrl);
        http.addHeader("Content-Type", "application/json");

        // Geração do payload com o Node ID e os dados de sensores
        String nodeID = WiFi.macAddress(); // Usando o MAC address como ID
        String payload = "{\"id\": \"" + nodeID +
            "\", \"temperatura\": " + String(temperatura) +
            ", \"umidade\": " + String(umidade) +
            ", \"qualidadeAr\": " + String(qualidadeAr) +
            ", \"alcool\": " + String(alcool) +
            ", \"benzeno\": " + String(benzeno) +
            ", \"CO2\": " + String(co2) +
            ", \"fumaca\": " + String(fumaca) + "}";

        int httpResponseCode = http.POST(payload);
        if (httpResponseCode > 0) {
            Serial.print("Dados enviados com sucesso! Resposta do servidor: ");
            Serial.println(httpResponseCode);
        } else {
            Serial.print("Falha ao enviar os dados. Código de erro: ");
            Serial.println(httpResponseCode);
        }
    }
}

```

```

    }

    http.end();
} else {
    Serial.println("Wi-Fi desconectado! Tentando reconectar...");
    WiFi.begin(ssid, password);
}

delay(60000); // Aguarda 1 minuto antes de fazer nova leitura
}

```

6.4 Código da placa Bitdoglab utilizando a plataforma Arduino IDE

O código é semelhante ao do ESP8266, com alterações nas bibliotecas e nos pinos dos sensores. Como essa placa não possui Serial Monitor, os LEDs RGB são usados para indicar o estado da conexão. Pino 11 aceso: conectado ao Wi-Fi; Pino 12 aceso: dados enviados com sucesso; Pino 13 aceso: falha no envio dos dados.

```

#include <WiFi.h>           // Biblioteca para wifi)
#include <HTTPClient.h>     // Biblioteca para requisições HTTP
#include <DHT.h>            // Biblioteca para o sensor DHT11

// Configuração dos sensores
#define DHTPIN 18          // Pino digital GP18 para o DHT11
#define DHTTYPE DHT11     // Tipo do sensor DHT11
#define MQ135PIN 28        // Pino analógico GP28 para o MQ135

DHT dht(DHTPIN, DHTTYPE);

// Configuração dos LEDs para status
#define LED_WIFI_PIN 11    // LED aceso se conectado ao Wi-Fi
#define LED_SUCCESS_PIN 12 // LED aceso se os dados forem enviados com sucesso
#define LED_FAIL_PIN 13    // LED aceso se houver falha no envio dos dados

// Configuração do Wi-Fi
const char* ssid = "COPELLI-2G"; // Nome da rede wifi
const char* password = "copelli1"; // Senha da rede wifi

// URL do servidor para onde os dados serão enviados
const char* serverUrl = "http://192.168.0.38:3000/api/submitData";

WiFiClient client; // Instância para conexão HTTP

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

    // Configura os pinos dos LEDs como saída
    pinMode(LED_WIFI_PIN, OUTPUT);
    pinMode(LED_SUCCESS_PIN, OUTPUT);
    pinMode(LED_FAIL_PIN, OUTPUT);

    // Conectar ao wifi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
    }
    // Acende o LED de Wi-Fi (pino 11)
    digitalWrite(LED_WIFI_PIN, HIGH);

    // Inicializa o sensor DHT
    dht.begin();
}

// Função para ler sensor MQ-135
float readMQ135() {
    int sensorValue = analogRead(MQ135PIN); // Leitura do pino analógico GP28
    return sensorValue;
}

// Função para calcular as concentrações dos gases para teste
void getGasConcentration(int mq135Value, float &alcool, float &benzeno, float &co2, float &fumaca) {
    alcool = mq135Value * 0.1; // Calculo apenas para teste
    benzeno = mq135Value * 0.2;
    co2 = mq135Value * 0.05;
    fumaca = mq135Value * 0.15;
}

```

```

void loop() {

  if (WiFi.status() == WL_CONNECTED) {
    // Leitura dos sensores:
    float temperatura = dht.readTemperature();
    float umidade = dht.readHumidity();
    int qualidadeAr = readMQ135();

    // Verifica se a leitura do DHT11 foi bem-sucedida
    if (isnan(temperatura) || isnan(umidade)) {
      return;
    }

    // Variáveis para os gases
    float alcool, benzeno, co2, fumaca;
    getGasConcentration(qualidadeAr, alcool, benzeno, co2, fumaca);

    // Preparando os dados para envio (incluindo o ID da placa)
    String nodeID = WiFi.macAddress();
    String payload = "{\"id\": \"" + nodeID +
      "\", \"temperatura\": " + String(temperatura) +
      ", \"umidade\": " + String(umidade) +
      ", \"qualidadeAr\": " + String(qualidadeAr) +
      ", \"alcool\": " + String(alcool) +
      ", \"benzeno\": " + String(benzeno) +
      ", \"CO2\": " + String(co2) +
      ", \"fumaca\": " + String(fumaca) + "}";

    // Envia os dados para o servidor via HTTP POST
    HTTPClient http;
    http.begin(client, serverUrl);
    http.addHeader("Content-Type", "application/json");

    int httpResponseCode = http.POST(payload);
    if (httpResponseCode > 0) {
      // Dados enviados com sucesso: acende LED no pino 12 e apaga LED de falha
      digitalWrite(LED_SUCCESS_PIN, HIGH);
      digitalWrite(LED_FAIL_PIN, LOW);
    } else {
      // Falha ao enviar os dados: acende LED no pino 13 e apaga LED de sucesso
      digitalWrite(LED_SUCCESS_PIN, LOW);
      digitalWrite(LED_FAIL_PIN, HIGH);
    }
    http.end();
  } else {
    WiFi.begin(ssid, password);
    // Desliga os LEDs de status se não conectado
    digitalWrite(LED_WIFI_PIN, LOW);
    digitalWrite(LED_SUCCESS_PIN, LOW);
    digitalWrite(LED_FAIL_PIN, LOW);
  }

  delay(60000); // Aguarda 1 minuto antes de enviar novamente
}

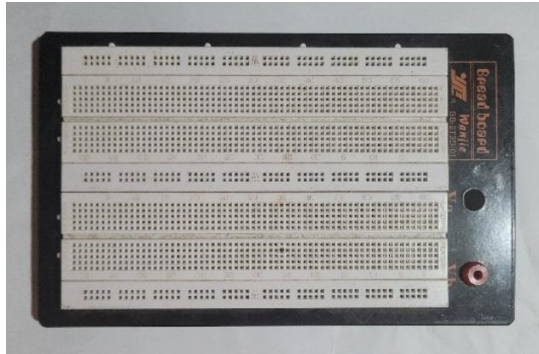
```

7. Itens utilizados

Para a montagem do projeto foi utilizado três placas que desenvolvi em projetos passados com o microcontrolador ESP8266 e a placa Bitdoglab para testar na prática utilizei um protoboard com os sensores DHT11 e o sensor MQ-135.

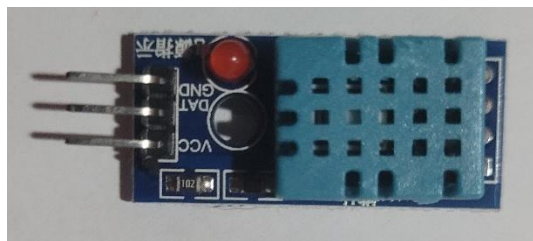
7.1 Protoboard

Utilizada para montar e conectar os sensores.



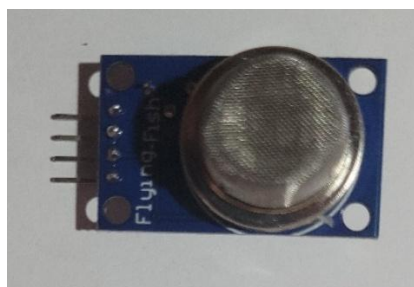
7.2 Sensor DHT11

Utilizado para medir a temperatura e umidade.



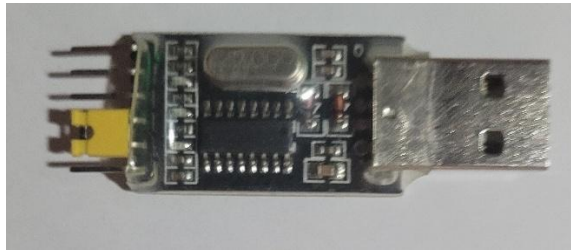
7.3 Sensor MQ-135

Utilizado para medir a concentração de gases no ambiente, como dióxido de carbono (CO₂), amônia (NH₃) e outros compostos.

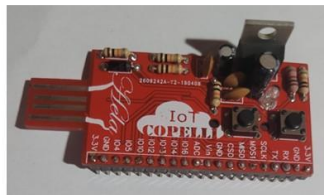
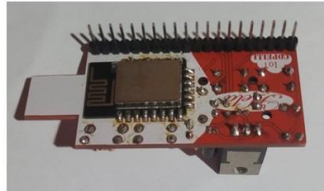


7.4 Conversor USB to TTL

Utilizado para gravar o código nos microcontrolados ESP8266.



7.5 Placas com microcontrolador ESP8266



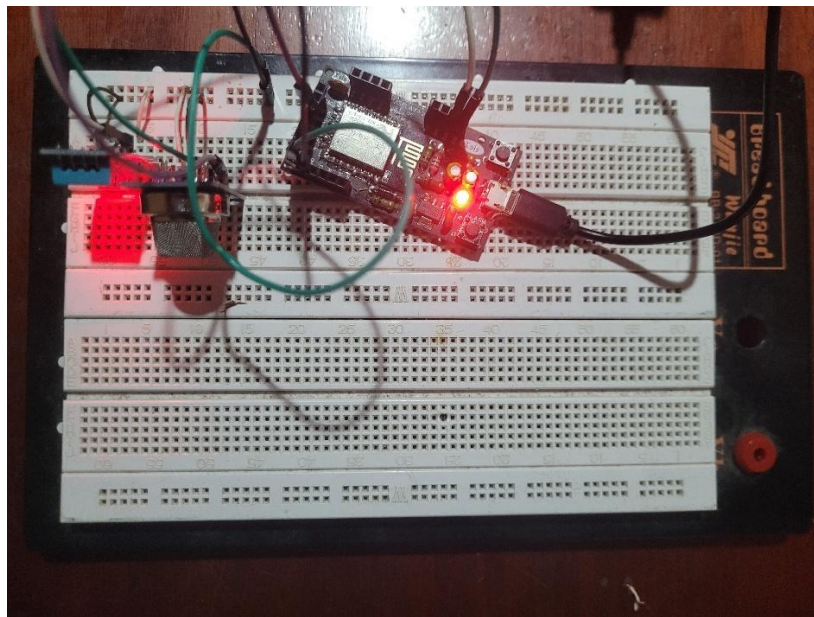
7.6 Placa Bitdoglab com o microcontrolador Raspberry Pi Pico W



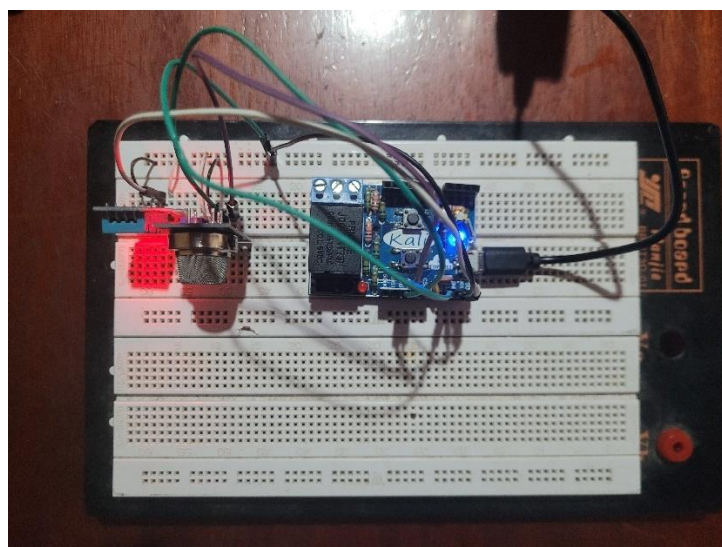
8. Testando

Realizei testes enviando dados para a blockchain IoTEx utilizando um sensor DHT11 e um sensor MQ-135. O circuito foi montado na protoboard, e durante os testes, alternei entre as placas. Não realizei testes com o envio simultâneo de dados com várias placas. Para este processo, utilizei a testnet da blockchain IoTEx, o que garantiu que não houvesse cobrança de taxas pelas transações realizadas.

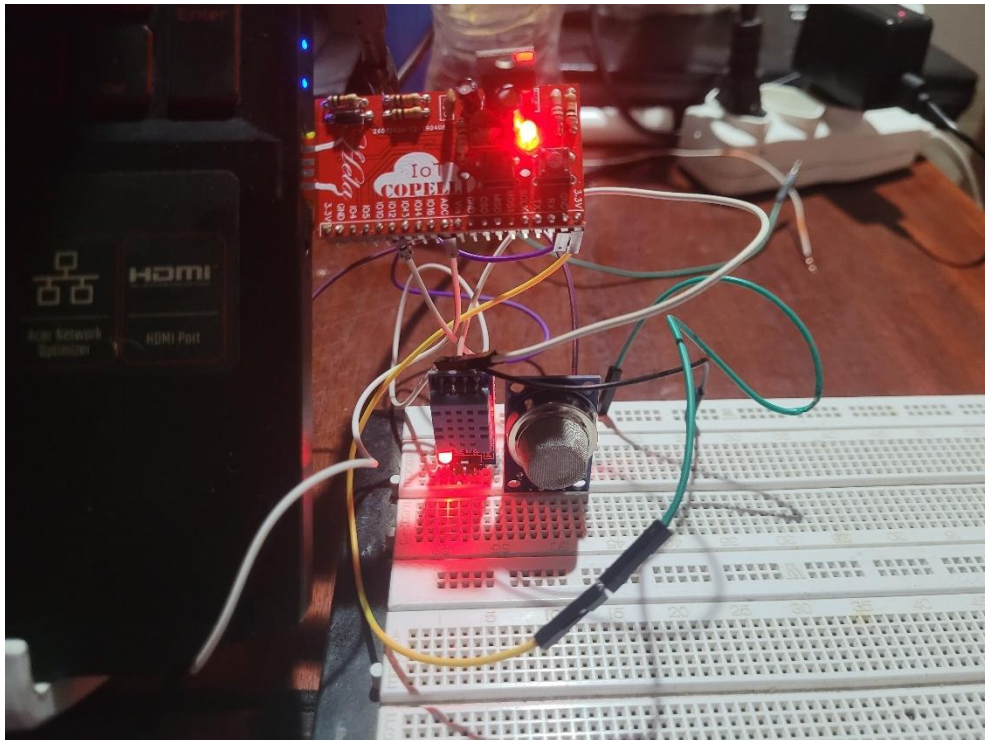
8.1 Placa ID (BC:DD:C2:26:69:77)



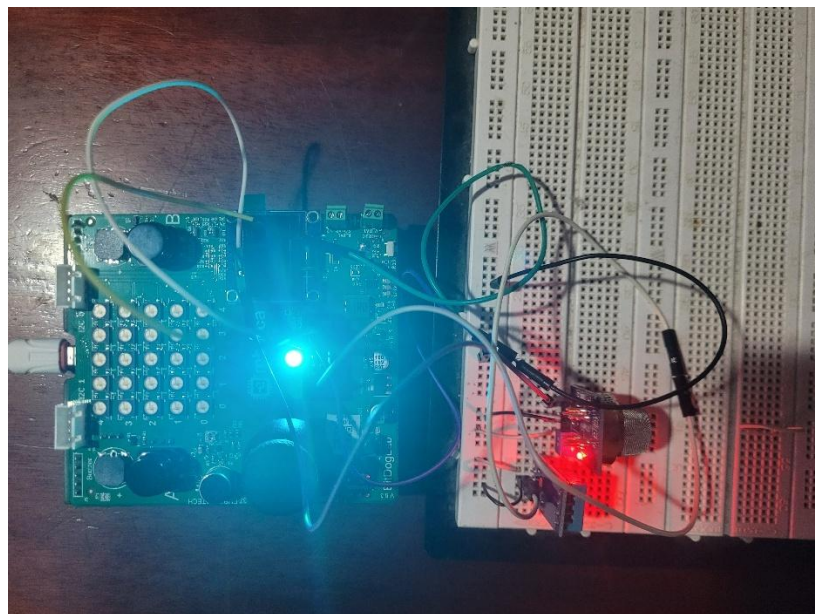
8.2 Placa ID (BC:DD:C2:26:69:23)



8.3 Placa ID (CC:50:E3:56:BD:D2)



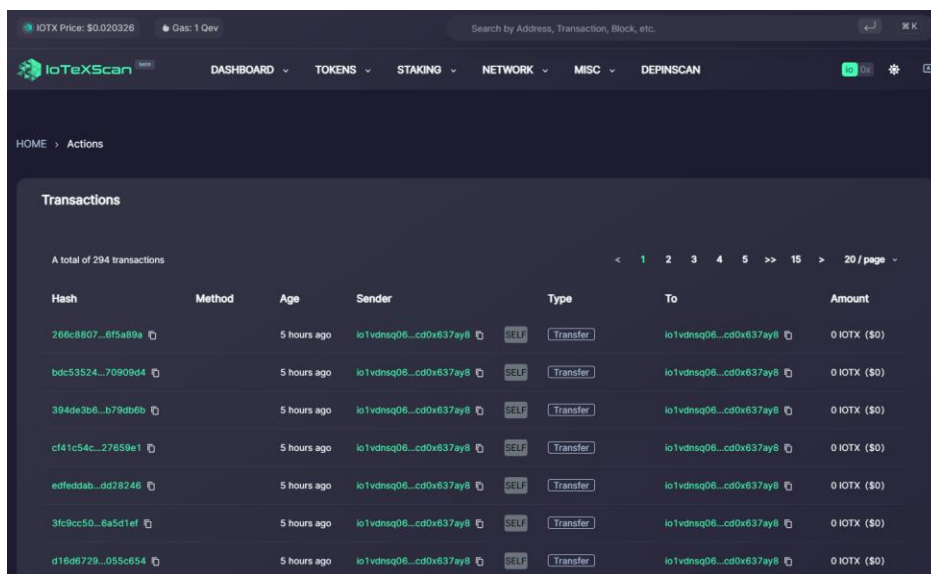
8.4 Placa ID (28:cd:c1:11:c7:47)



9. Resultados

Os resultados foram obtidos com sucesso e podem ser visualizados. Os links de referência foram devidamente inseridos em links do projeto. Ao todo, foram realizadas 294 transações. Inicialmente, os valores transmitidos foram apenas de temperatura, umidade e qualidade do ar, com valores fictícios, a fim de testar a transmissão dos dados. Como se trata de uma blockchain, é importante destacar que, uma vez registrados, os dados não podem ser apagados, tornando-os imutáveis.

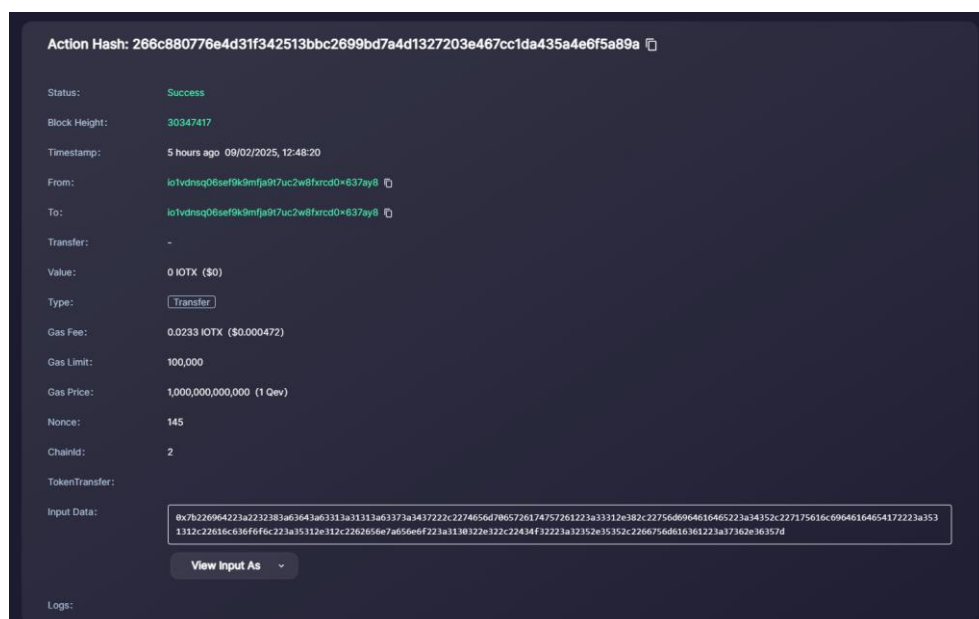
9.1 IoTEx scan - Página principal



The screenshot shows the IoTExScan dashboard. At the top, there's a header with 'IOTX Price: \$0.020328', 'Gas: 1 Qev', and a search bar. Below the header is a navigation menu with 'DASHBOARD', 'TOKENS', 'STAKING', 'NETWORK', 'MISC', and 'DEPINSCAN'. The main content area is titled 'Actions' and 'Transactions'. It displays a table of transactions with columns: Hash, Method, Age, Sender, Type, To, and Amount. The table shows a list of transactions, all with a 'Type' of 'Transfer' and an 'Amount' of '0 IOTX (\$0)'. The 'Sender' and 'To' fields are truncated, showing 'io1vdsnq06...cd0x637ay8'. The 'Age' column shows '5 hours ago' for all transactions. The 'Hash' column shows various hexadecimal strings.

Hash	Method	Age	Sender	Type	To	Amount
266c8807...6f5a89a		5 hours ago	io1vdsnq06...cd0x637ay8	Transfer	io1vdsnq06...cd0x637ay8	0 IOTX (\$0)
bdc53524...70909d4		5 hours ago	io1vdsnq06...cd0x637ay8	Transfer	io1vdsnq06...cd0x637ay8	0 IOTX (\$0)
394de3b6...b79db6b		5 hours ago	io1vdsnq06...cd0x637ay8	Transfer	io1vdsnq06...cd0x637ay8	0 IOTX (\$0)
cf41c54c...27859e1		5 hours ago	io1vdsnq06...cd0x637ay8	Transfer	io1vdsnq06...cd0x637ay8	0 IOTX (\$0)
edfeddab...dd28246		5 hours ago	io1vdsnq06...cd0x637ay8	Transfer	io1vdsnq06...cd0x637ay8	0 IOTX (\$0)
3fc9cc50...8a5d1ef		5 hours ago	io1vdsnq06...cd0x637ay8	Transfer	io1vdsnq06...cd0x637ay8	0 IOTX (\$0)
d16d6729...055c654		5 hours ago	io1vdsnq06...cd0x637ay8	Transfer	io1vdsnq06...cd0x637ay8	0 IOTX (\$0)

9.2 IoTEx scan – Transação ocorrida com sucesso

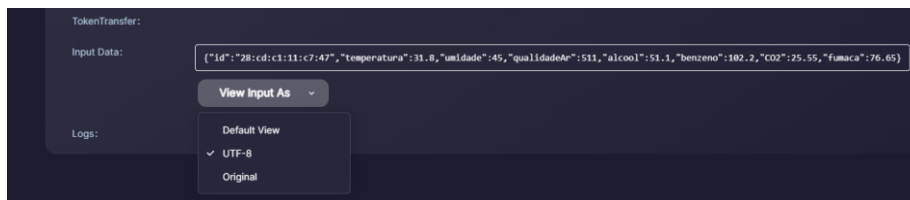


The screenshot shows the details of a successful transaction. The 'Action Hash' is '266c880776e4d31f342513bbc2699bd7a4d1327203e467cc1da435a4e6f5a89a'. The 'Status' is 'Success'. The 'Block Height' is '30347417'. The 'Timestamp' is '5 hours ago 09/02/2025, 12:48:20'. The 'From' and 'To' fields are truncated, showing 'io1vdsnq06...cd0x637ay8'. The 'Transfer' field is '-'. The 'Value' is '0 IOTX (\$0)'. The 'Type' is 'Transfer'. The 'Gas Fee' is '0.0233 IOTX (\$0.000472)'. The 'Gas Limit' is '100,000'. The 'Gas Price' is '1,000,000,000,000 (1 Qev)'. The 'Nonce' is '145'. The 'ChainId' is '2'. The 'TokenTransfer' is '-'. The 'Input Data' is a long hexadecimal string. There is a 'View Input As' button at the bottom.

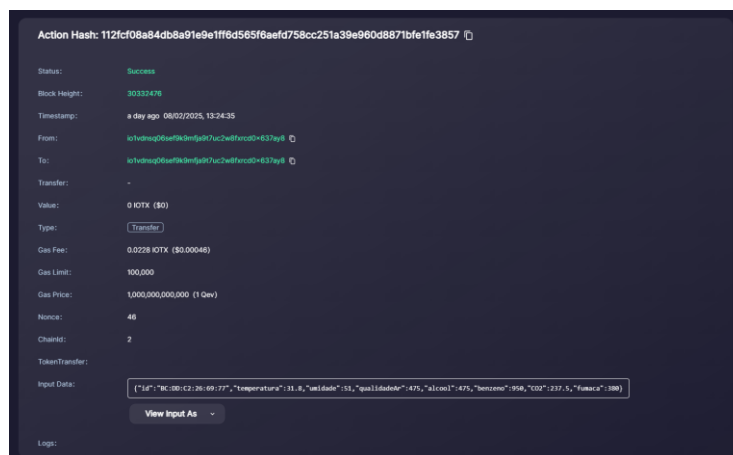
Action Hash:	266c880776e4d31f342513bbc2699bd7a4d1327203e467cc1da435a4e6f5a89a
Status:	Success
Block Height:	30347417
Timestamp:	5 hours ago 09/02/2025, 12:48:20
From:	io1vdsnq06...cd0x637ay8
To:	io1vdsnq06...cd0x637ay8
Transfer:	-
Value:	0 IOTX (\$0)
Type:	Transfer
Gas Fee:	0.0233 IOTX (\$0.000472)
Gas Limit:	100,000
Gas Price:	1,000,000,000,000 (1 Qev)
Nonce:	145
ChainId:	2
TokenTransfer:	-
Input Data:	8x7b126964222a222383a63643a6313a2131a3377a3437222c2274656d7065726174757261223a3311e382c2275646964616465222a24353c227175616c69646164654172223a3531312c22616c6964616465223a35112c2262656e7a656e6f223a1138122e322c22434f32223a12352e35352c2266756d616361223a17362e36357d
View Input As:	
Logs:	

9.3 IoTEx scan – Transação ocorrida com sucesso – resultados dos sensores na placa ID (28:cd:c1:11:c7:47)

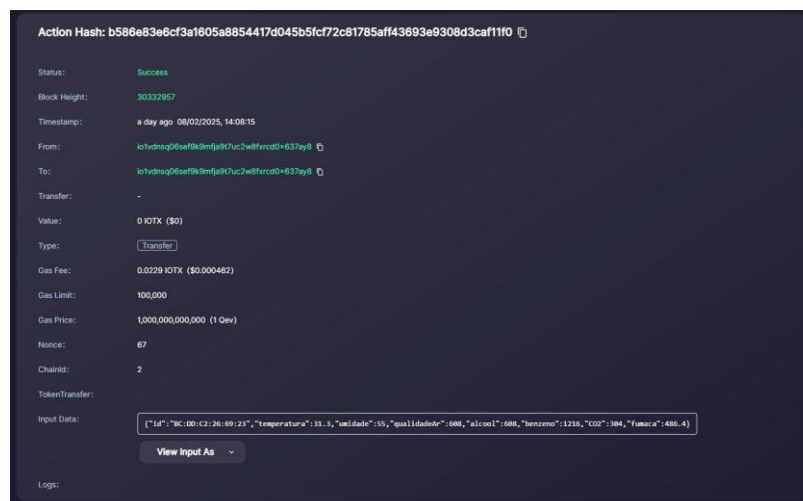
Para ver os resultados coletados em Input Data colocar no formato UTF-8.



9.4 IoTEx scan – Transação ocorrida com sucesso – resultados dos sensores na placa ID (BC:DD:C2:26:69:77).



9.5 IoTEx scan – Transação ocorrida com sucesso – resultados dos sensores na placa ID (BC:DD:C2:26:69:23).



9.6 IoTEx scan – Transação ocorrida com sucesso – resultados dos sensores na placa ID (BC:DD:C2:26:69:77).

Action Hash: 668ffc0acc30243914fa2e7b41aadbc18e1fade3702617508d50d5d89e18c19 [🔗](#)

Status: Success

Block Height: 30334783

Timestamp: a day ago 08/02/2025, 17:04:35

From: io1vdrag0seef9k9m9jd7uc2w8fmc0D+637ay8 [🔗](#)

To: io1vdrag0seef9k9m9jd7uc2w8fmc0D+637ay8 [🔗](#)

Transfer: -

Value: 0 IOTX (\$0)

Type: Transfer

Gas Fee: 0.0231 IOTX (\$0.000486)

Gas Limit: 100,000

Gas Price: 1,000,000,000,000 (1 Qev)

Nonce: 86

ChainId: 2

TokenTransfer:

Input Data:

[{"id": "28::cd:c1:11:c7:43", "temperature": 33.3, "umidade": 52, "qualidade": 642, "alcohol": 64.2, "benzeno": 128.4, "CO2": 32.1, "fumaca": 96.3}]

View Input As ▾

Logs:

10. Futuras melhorias

Em vez de usar a placa Bitdoglab apenas para enviar os dados dos sensores ao `server.js`, onde a transação é assinada e enviada para a blockchain IoTEx, a proposta é que a própria Bitdoglab assine a transação e a envie diretamente para a blockchain.

Além disso, a blockchain será alterada para a Hyperledger Fabric no modelo permissionado, onde a Bitdoglab atuará como validadora das transações. As placas com microcontroladores serão responsáveis pela coleta dos dados dos sensores e os enviarão para a Bitdoglab, que validará as informações e enviara os dados para a blockchain permissionada. Dessa forma, evita-se a cobrança por transação, garantindo maior controle e eficiência no processamento dos dados. E criar uma API para filtrar os dados enviados pelo ID e mostrar gráficos com os resultados de temperatura, umidade e gases coletados.

11. Referencias

MetaMask

METAMASK. Disponível em: <https://metamask.io>.

Documentação IoTEx

IOTEX DOCS. Disponível em: <https://docs.iotex.io/>.

Developers IoTEx

IOTEX DEVELOPERS. Disponível em: <https://developers.iotex.io/>.

Organização dos Pinos da BitDogLab

BITDOGLAB. Organização dos Pinos da BitDogLab. Disponível em:
<https://pt.scribd.com/document/814073680/Organizacao-dos-Pinos-da-BitDogLab>.

12. Links do projeto

Endereço Público da carteira – também serve para monitorar os dados
0x6367003f50ca4B62Ed32e957Ee614e3a4c3c35e6

Name Tag – Também serve para monitorar os dados
io1vdsnq06sef9k9mfja9t7uc2w8fxrcd0x637ay8

Blockchain IoTEx – Link aceso direto a blockchain
<https://testnet.iotexscan.io/address/0x6367003f50ca4b62ed32e957ee614e3a4c3c35e6#transactions>

GitHub:
<https://github.com/tiagocopelli/Capacitacao-Profissional-em-Sistemas-Embarcados>

Google Drive:
https://drive.google.com/drive/folders/1pi-S9Y6_CfmEu0peuRbLs0RBXc9Y3X6T