

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN**



**IOT VÀ ỨNG DỤNG
NHÓM HỌC PHẦN: 06**

Đề tài: Theo dõi thông số trong nhà: Ánh sáng, nhiệt độ, độ ẩm và điều khiển các thiết bị

**Giảng viên: Nguyễn Quốc Uy
Tên Sinh Viên: Nguyễn Trọng Trường
Mã Sinh Viên: B21DCCN740**

Hà Nội 2024

Nội dung

1. Đặt vấn đề.....	3
2. Mục tiêu.....	3
3. Các thiết bị được sử dụng.....	3
3.1. EPS8266	3
3.1.1. Mô tả ESP8266.....	3
3.2. Cảm biến nhiệt độ độ ẩm DHT11	4
3.2.1. Mô tả.....	4
3.2.2. Thiết kế	4
3.2.3. Thông số kỹ thuật	4
3.3. Module cảm biến ánh sáng	5
3.3.1. Mô tả.....	5
3.3.2. Thiết kế	5
3.3.3. Thông số kỹ thuật:	5
3.4. Led	5
4. Công nghệ sử dụng.....	6
4.1. Express	6
4.1.1. Tổng quan về Express:	6
4.1.2. Các tính năng chính của Express:.....	6
4.1.3. Cài đặt Express:	6
4.2. ReactJs	7
5. Giao diện	7
5.1. Thiết kế tổng thể.....	7
5.2. Thiết kế chi tiết.....	8
5.2.1. Luồng dữ liệu.....	8
5.2.2. Sơ đồ tuần tự.....	8
5.2.3. Database	9
6. Code.....	10
7. API Docs.....	14
8. Kết quả.....	17

1. Đặt vấn đề

Một hệ thống thời gian thực cho phép truyền dữ liệu từ cảm biến để hiển thị cho người dùng. Người dùng có thể tương tác ngược lại cho thiết bị. Một hệ thống IoT đơn giản.

2. Mục tiêu

Hệ thống sử dụng hai cảm biến để thông báo nhiệt độ và độ ẩm, ánh sáng cho người dùng thông qua trang web có thể truy cập với mạng LAN. Trang web sẽ cung cấp nhiệt độ và độ ẩm, ánh sáng với dashboard hiển thị dễ nhìn, cho phép người dùng có thể xem thông tin nhiệt độ và độ ẩm trực tiếp ở thời gian thực và có thể điều khiển led.

3. Các thiết bị được sử dụng

3.1. ESP8266

3.1.1. Mô tả ESP8266

Kit phát triển Wifi ESP8266 NodeMCU Lua CP2102 Development Board là kit phát triển dựa trên nền chip Wifi SoC ESP8266 với thiết kế dễ sử dụng và đặc biệt là có thể sử dụng trực tiếp trình biên dịch của Arduino để lập trình và nạp code, điều này khiến việc sử dụng và lập trình các ứng dụng trên ESP8266 trở nên rất đơn giản.

Kit phát triển Wifi ESP8266 NodeMCU Lua CP2102 Development Board được dùng cho các ứng dụng cần kết nối, thu thập dữ liệu và điều khiển qua sóng Wifi, đặc biệt là các ứng dụng liên quan đến IoT.

Kit phát triển Wifi ESP8266 NodeMCU Lua CP2102 Development Board tại Hshop.vn sử dụng chip nạp và giao tiếp UART CP2102 có độ ổn định và độ bền cao và khả năng tự nhận Driver trên tất cả các hệ điều hành Windows và Linux.



Hình 1: ESP8266

3.1.2. Thông số kỹ thuật:

- IC chính: ESP8266
- Phiên bản firmware: NodeMCU Lua
- Chip nạp và giao tiếp UART: CP2102.
- GPIO tương thích hoàn toàn với firmware Node MCU.
- Cấp nguồn: 5VDC MicroUSB hoặc Vin.
- GPIO giao tiếp mức 3.3VDC
- Tích hợp Led báo trạng thái, nút Reset, Flash.
- Tương thích hoàn toàn với trình biên dịch Arduino.

- Kích thước: 25 x 50 mm

3.2. Cảm biến nhiệt độ độ ẩm DHT11



Hình 2: DHT11

3.2.1. Mô tả

Cảm biến nhiệt độ và độ ẩm DHT11 là một cảm biến giá rẻ và phổ biến trong các dự án liên quan đến đo nhiệt độ và độ ẩm môi trường. Nó được sử dụng để đo và cung cấp thông tin về nhiệt độ và độ ẩm hiện tại trong một môi trường cụ thể.

Cảm biến DHT11 sử dụng giao thức 1-wire đơn giản để giao tiếp với vi điều khiển. Nó truyền dữ liệu nhiệt độ và độ ẩm dưới dạng tín hiệu kỹ thuật số qua chân Data.

Cảm biến DHT11 thường được sử dụng trong các ứng dụng như đo nhiệt độ và độ ẩm trong phòng, điều khiển tự động, hệ thống quản lý môi trường và các dự án IoT. Để sử dụng cảm biến, bạn cần cài đặt thư viện DHT11 cho vi điều khiển hoặc vi xử lý mà bạn đang sử dụng và thực hiện các lệnh đọc dữ liệu từ cảm biến thông qua chân Data.

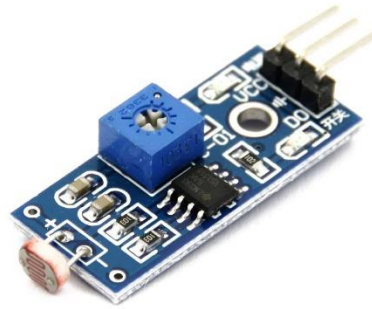
3.2.2. Thiết kế

- VCC: Chân nguồn dương, được kết nối với nguồn cung cấp 3.3V - 5V.
- Data: Chân dữ liệu, được sử dụng để giao tiếp với vi điều khiển hoặc vi xử lý để truyền dữ liệu nhiệt độ và độ ẩm.
- GND: Chân nguồn âm, được kết nối với chân GND của nguồn cung cấp và vi điều khiển.

3.2.3. Thông số kỹ thuật

- Phạm vi đo nhiệt độ: 0°C đến 50°C với độ chính xác $\pm 2^\circ\text{C}$.
- Phạm vi đo độ ẩm: 20% đến 90% RH với độ chính xác $\pm 5\%$ RH.
- Điện áp hoạt động: 3.3V - 5V.
- Điện áp logic: 3.3V - 5V (hỗ trợ giao tiếp với các vi điều khiển 3.3V và 5V).
- Tốc độ truyền dữ liệu: Tối đa 1Hz (1 lần đo dữ liệu mỗi giây).
- Kích thước: Thường có kích thước nhỏ gọn và hình dạng hình hộp chữ nhật.

3.3. Module cảm biến ánh sáng



Hình 3: Cảm biến ánh sáng

3.3.1. Mô tả

Module cảm biến ánh sáng là một linh kiện điện tử có điện trở thay đổi giảm theo ánh sáng chiếu vào. Quang trở làm bằng chất bán dẫn trở kháng cao, và không có tiếp giáp nào. Trong bóng tối, quang trở có điện trở đến vài MΩ. Khi có ánh sáng, điện trở giảm xuống mức một vài trăm Ω Độ chính xác: $\pm 5\%RH$ và $\pm 2\sigma$

Hoạt động của quang trở dựa trên hiệu ứng quang điện trong khối vật chất. Khi photon có năng lượng đủ lớn đập vào, sẽ làm bật electron khỏi phân tử, trở thành tự do trong khối chất và làm chất bán dẫn thành dẫn điện. Mức độ dẫn điện tùy thuộc số photon được hấp thụ.

3.3.2. Thiết kế

- VCC: Chân nguồn dương, được kết nối với nguồn cung cấp 3.3V - 5V.
- D0: Chân dữ liệu, được sử dụng để giao tiếp với vi điều khiển hoặc vi xử lý để truyền dữ liệu nhiệt độ và độ ẩm.
- GND: Chân nguồn âm, được kết nối với chân GND của nguồn cung cấp và vi điều khiển.

3.3.3. Thông số kỹ thuật:

- Điện áp hoạt động: 3.3V-5V
- Kích thước PCB: 3cm * 1.6cm
- Led xanh báo nguồn và ánh sáng
- IC so sánh : LM393
- VCC: 3.3V-5V
- GND: 0V
- DO: Đầu ra tín hiệu số (0 và 1)
- AO: Đầu ra Analog (Tín hiệu tương tự)

3.4. Led

LED 2 chân (Light Emitting Diode) là một loại diode phát sáng khi có dòng điện đi qua theo chiều thuận. Nó là một linh kiện điện tử phổ biến, được sử dụng rộng rãi trong các thiết bị điện tử để chiếu sáng hoặc làm đèn báo hiệu.

Cấu tạo: 2 chân

- **Chân dương (Anode):** Đây là chân dài hơn, được kết nối với cực dương (+) của nguồn điện.
- **Chân âm (Cathode):** Chân này ngắn hơn và được kết nối với cực âm (-) của nguồn điện.

Nguyên lý hoạt động: Khi có dòng điện đi từ Anode sang Cathode (theo chiều thuận của diode), các electron sẽ chuyển động và tái hợp với các lỗ trống trong cấu trúc vật liệu bán dẫn của LED. Quá trình này giải phóng năng lượng dưới dạng ánh sáng.

LED là một diode nên chỉ phát sáng khi dòng điện đi theo một hướng nhất định (từ chân dương sang chân âm). Nếu kết nối ngược chiều, LED sẽ không sáng.

Đặc điểm:

- **Tiêu thụ năng lượng thấp:** LED sử dụng ít năng lượng hơn so với các loại đèn khác.
- **Tuổi thọ cao:** LED có tuổi thọ dài, có thể hoạt động hàng nghìn giờ. Kích thước nhỏ gọn: Thường rất nhỏ và dễ dàng tích hợp vào các mạch điện tử.
- **Ánh sáng đa dạng:** LED có thể phát ra nhiều màu sắc khác nhau (đỏ, xanh lá, xanh dương, trắng, vàng, v.v.) tùy thuộc vào loại vật liệu bán dẫn được sử dụng.
- **Ứng dụng:** Được sử dụng trong đèn chỉ báo, màn hình LED, đèn nền cho màn hình hiển thị, thiết bị điện tử gia dụng, hệ thống chiếu sáng và nhiều ứng dụng khác.

4. Công nghệ sử dụng

4.1. Express

4.1.1. Tổng quan về Express:

Express là một framework web minimal và linh hoạt cho Node.js, hỗ trợ xây dựng các ứng dụng web và API mạnh mẽ. Được thiết kế để dễ dàng mở rộng, Express cung cấp các công cụ và tính năng để quản lý yêu cầu HTTP, định tuyến, middleware và tích hợp dễ dàng với các cơ sở dữ liệu và dịch vụ bên ngoài.

4.1.2. Các tính năng chính của Express:

- **Định tuyến linh hoạt:** Express có hệ thống định tuyến mạnh mẽ, cho phép xây dựng các đường dẫn URL linh hoạt và phức tạp, hỗ trợ các phương thức HTTP như GET, POST, PUT, DELETE.
- **Middleware:** Express có khả năng xử lý middleware, giúp quản lý chuỗi xử lý yêu cầu một cách dễ dàng. Middleware có thể xử lý xác thực, xử lý lỗi, hay thay đổi yêu cầu/trả về trước khi gửi.
- **Khả năng mở rộng cao:** Express dễ dàng tích hợp với các thư viện bên ngoài như Mongoose (MongoDB), Sequelize (SQL), và các dịch vụ khác.
- **Tối ưu hóa hiệu suất:** Express được tối ưu hóa để xử lý lượng lớn yêu cầu HTTP, phù hợp cho các ứng dụng web thời gian thực hoặc API tốc độ cao.
- **Hỗ trợ view engines:** Express hỗ trợ nhiều view engine như EJS, Pug (Jade), Handlebars, giúp dễ dàng hiển thị giao diện từ phía máy chủ.

4.1.3. Cài đặt Express:

- Để cài đặt Express, đầu tiên cần có **Node.js** và **npm**.
- Mở terminal và tạo một thư mục mới cho dự án:

`mkdir myapp`

`cd myapp`

- Khởi tạo dự án với npm:

`npm init -y`

- Cài đặt Express:

`npm install express --save`

- Sau khi cài đặt, tạo file `index.js` và bắt đầu viết mã để khởi chạy server:

```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

Hình 4: Tạo chương server bằng express

- Chạy server:

`node index.js`

4.2. ReactJs

ReactJS là thư viện JavaScript phát triển bởi Facebook, dùng để xây dựng giao diện người dùng (UI), đặc biệt là các ứng dụng một trang (SPA). React cho phép tạo các thành phần (components) có thể tái sử dụng, giúp dễ quản lý và phát triển

Các tính năng chính:

- Component-based: Xây dựng UI bằng các thành phần có thể tái sử dụng.
- Virtual DOM: Cải thiện hiệu suất bằng cách cập nhật DOM hiệu quả.
- One-Way Data Binding: Dữ liệu được truyền theo một chiều, giúp dễ kiểm soát.

Cài đặt ReactJS:

- Cài đặt Node.js từ [Node.js](https://nodejs.org/en/).
- Tạo ứng dụng React mới: `npx create-react-app my-app`
- Chạy ứng dụng: `cd my-app npm start`

5. Giao diện

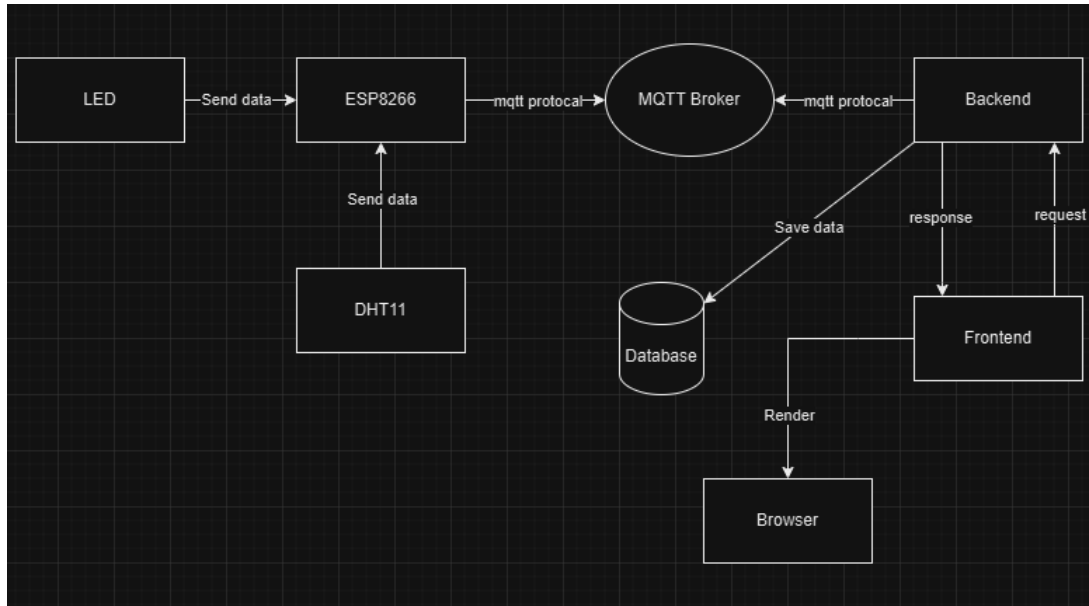
5.1. Thiết kế tổng thể

Giao diện web sẽ gồm 4 trang:

- Dashboard: Sẽ chứa biểu đồ, các nút điều khiển thiết bị và 3 ô giá trị thời gian thực nhận được từ thiết bị
- New Page: Chứa một biểu đồ, một nút điều khiển, một ô nhận giá trị thời gian thực từ thiết bị
- DataHistory: Một bảng gồm các cột thông tin về sensor và có thể tìm kiếm và sắp xếp
- Devices: Một bảng gồm các cột thông tin về lịch sử bật tắt thiết bị và có thể tìm kiếm.
- Profile: Chứa các thông tin các nhân và link src, api, báo cáo

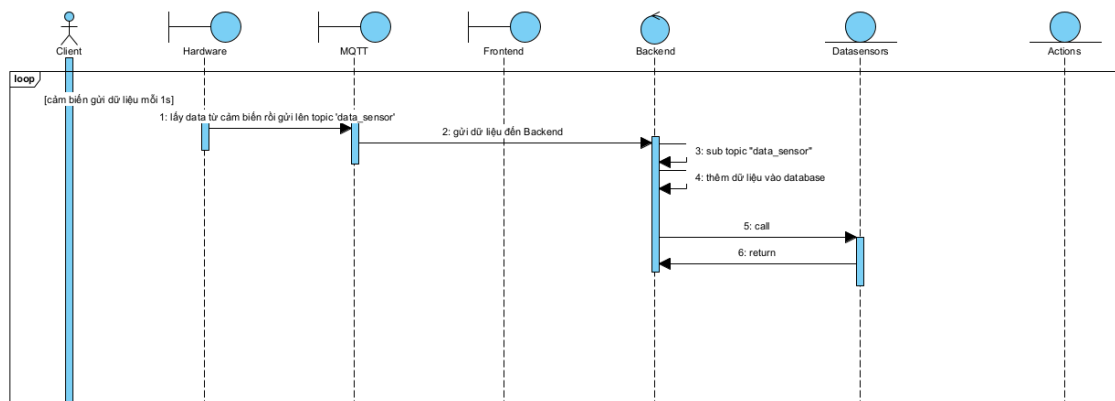
5.2. Thiết kế chi tiết

5.2.1. Luồng dữ liệu

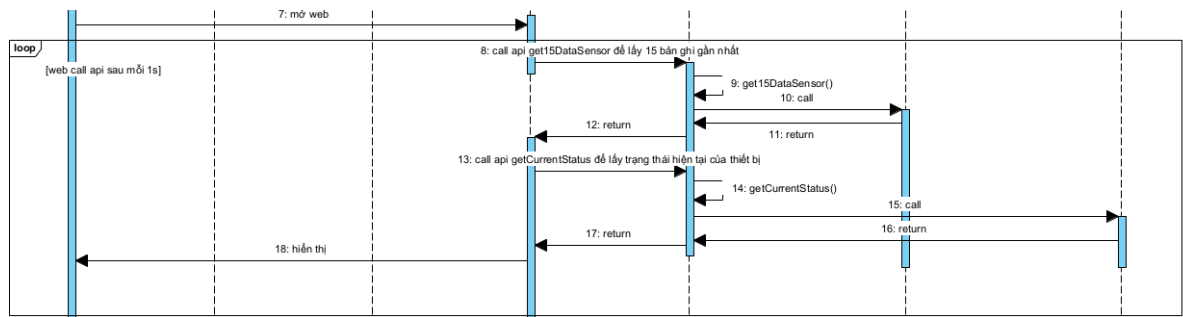


Hình 5: Sơ đồ luồng dữ liệu

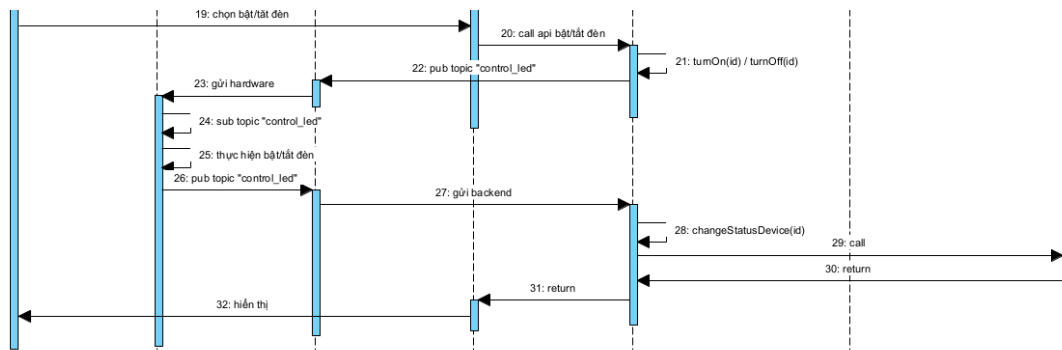
5.2.2. Sơ đồ tuần tự



Hình 6: Sơ đồ tuần tự 1



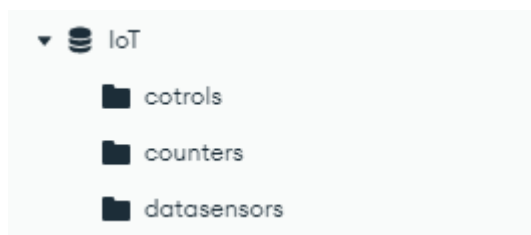
Hình 7: Sơ đồ tuần tự 2



Hình 8: Sơ đồ tuần tự 3

5.2.3. Database

Sử dụng MongoDB gồm 3 bảng:



Hình 9: Dữ liệu trong MongoDB

- cotrols: Chứa dữ liệu các thiết bị

```

1  _id: 1
2  device: "Air Conditioner"
3  action: "On"
4  time: "27/10/2024 00:02:10"
5  __v: 0

```

Int32
String
String
String
Int32

Hình 10: Dữ liệu trong collection cotrols

- datasensors: Chứa dữ liệu đo các chỉ số của cảm biến

1	<code>_id: ObjectId('671d205ca35f09db8d8b6c59')</code>	ObjectId
2	<code>id: 1</code>	Int32
3	<code>temperature: 31.3</code>	Double
4	<code>humidity: 41</code>	Int32
5	<code>light: 9830</code>	Int32
6	<code>time: "27/10/2024 00:01:16"</code>	String
7	<code>dust: 42</code>	Int32
8	<code>__v: 0</code>	Int32

Hình 11: Dữ liệu trong collection *datasensors*

- counters: Được tự khởi tạo bởi thư viện mongoose-sequence giúp tự động tăng id cho dữ liệu khi được thêm mới

6. Code

- `get15DataSensor`:

```
const get15DataSensor = async (req, res) => {
  const data = await DataSensor.find().sort({ _id: -1 }).limit(15);
  res.status(200).json(data);
};
```

Hình 12: Hàm lấy 15 dữ liệu từ *DataSensor*

- `getAllData`:

```

const getDataSensor = async (req, res) => {
  let { page = 1, keyword, field, sortField, sortOrder } = req.query;
  page = parseInt(page) < 1 ? 1 : parseInt(page);
  try {
    let query = {};
    if (keyword) {
      if (field) {
        if (field === "temperature") {
          query[field] = parseFloat(keyword);
        } else if (["humidity", "id", "dust", "light"].includes(field)) {
          query[field] = parseInt(keyword);
        } else if (field === "time") {
          query[field] = { $regex: keyword.replace(/-/g, " "), $options: "i" };
        }
      } else {
        query = {
          $or: [
            { id: parseInt(keyword) },
            { temperature: parseFloat(keyword) },
            { humidity: parseInt(keyword) },
            { dust: parseInt(keyword) },
            { light: parseInt(keyword) },
            { time: { $regex: keyword.replace(/-/g, " "), $options: "i" } },
          ],
        };
      }
    }
    let sort = {};
    if (sortField && sortOrder) {
      sort[sortField] = sortOrder === "desc" ? -1 : 1;
    }
    const totalData = await DataSensor.countDocuments(query);
    const totalPages = Math.ceil(totalData / pageSize);
    const data = await DataSensor.find(query)
      .sort(sort)
      .skip((page - 1) * pageSize)
      .limit(pageSize);

    res.json({
      data: [...data],
      currentPage: page,
      totalPages: totalPages,
      totalData: totalData,
    });
  } catch (error) {
    res.status(500).json({ message: "Lỗi khi xử lý yêu cầu", error });
  }
};

```

Hình 13: Hàm lấy dữ liệu từ DataSensor

- getControlHistory:

```

const getControlHistory = async (req, res) => {
  let { page = 1, keyword, field, sortField, sortOrder } = req.query;
  page = parseInt(page) < 1 ? 1 : parseInt(page);
  try {
    let query = {};
    if (keyword) {
      if (field) {
        if (["device", "action", "time"].includes(field)) {
          query[field] = { $regex: keyword.replace(/-/g, " "), $options: "i" };
        } else if (field === "_id") {
          query[field] = parseInt(keyword);
        }
        console.log(query);
      } else {
        query = {
          $or: [
            { id: parseInt(keyword) },
            { device: { $regex: keyword.replace(/-/g, " "), $options: "i" } },
            { action: { $regex: keyword.replace(/-/g, " "), $options: "i" } },
            { time: { $regex: keyword.replace(/-/g, " "), $options: "i" } },
          ],
        };
      }
    }
    let sort = {};
    if (sortField && sortOrder) {
      sort[sortField] = sortOrder === "desc" ? -1 : 1;
    }
    const totalData = await Control.countDocuments(query);
    const totalPages = Math.ceil(totalData / pageSize);
    const data = await Control.find(query)
      .sort(sort)
      .skip((page - 1) * pageSize)
      .limit(pageSize);

    res.json({
      data: [...data],
      currentPage: page,
      totalPages: totalPages,
      totalData: totalData,
    });
  } catch (error) {
    res.status(500).json({ message: "Lỗi khi xử lý yêu cầu", error });
  }
};

```

Hình 14: Hàm lấy dữ liệu Control

- getCurrentStatus:

```

const getCurrentStatus = async (req, res) => {
  try {
    const latestStates = await Control.aggregate([
      { $sort: { _id: -1 } },
      {
        $group: {
          _id: "$device",
          latestRecord: { $first: "$$ROOT" },
        },
      },
      {
        $replaceRoot: { newRoot: "$latestRecord" },
      },
    ]);
    res.status(200).json(latestStates);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: "Error retrieving device states" });
  }
};

```

Hình 15: Hàm lấy trạng thái hiện tại của các thiết bị

- postControlHistory:

```

const postControlHistory = async (req, res) => {
  const message = req.body;
  const newControl = new Control({
    ...message,
    time: getCurrentTime(),
  });
  client.publish("control_led", JSON.stringify(message), (err) => {
    if (err) {
      console.log(err);
    } else {
      console.log("Publishing to topic: contro_led");
      newControl.save().then(() => {
        req.io.emit("control_update", message);
        res
          .status(200)
          .send("Successfully controlled and notified via Socket.io!");
      });
    }
  });
};

```

Hình 16: Hàm nhận và xử lý dữ liệu Control được gửi từ client

- sub và lắng nghe topic bên phía server:

```

client.subscribe("data_sensor", (err) => {
  if (!err) {
    console.log(`Subscribed to topic: data_sensor`);
  } else {
    console.error("Subscription error:", err);
  }
});

client.on("message", async (topic, message) => {
  try {
    const jsonData = JSON.parse(message.toString());
    const data = {
      ...jsonData,
      time: getCurrentTime(),
    };
    const newDataSensor = new DataSensor(data);
    await newDataSensor.save();
    console.log(`Save: ${newDataSensor}`);
  } catch (err) {
    console.error(err);
  }
});

```

Hình 17: Subscribe và lắng nghe dữ liệu từ topic

7. API Docs

The screenshot displays the Swagger UI for the `GetAllControl` API endpoint. The URL is `http://localhost:8080/api/control?page=1&keyword=12&field=time&sortField=time&sortOrder=desc`. The query parameters are listed in a table below.

Key	Value	Description
page	1	
keyword	12	
field	time	include: null, device, action, time
sortField	time	include: null, id, device, action, time
sortOrder	desc	include: asc, desc

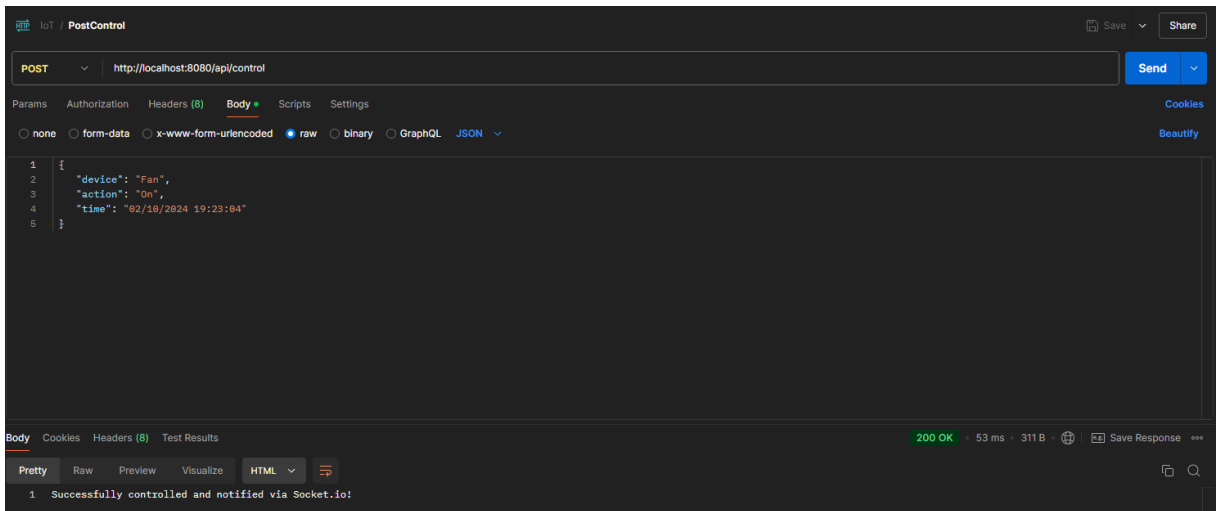
The response body is a JSON array of objects, each representing a sensor data point. The response status is `200 OK` with a response time of `75 ms` and a size of `1.14 KB`.

```

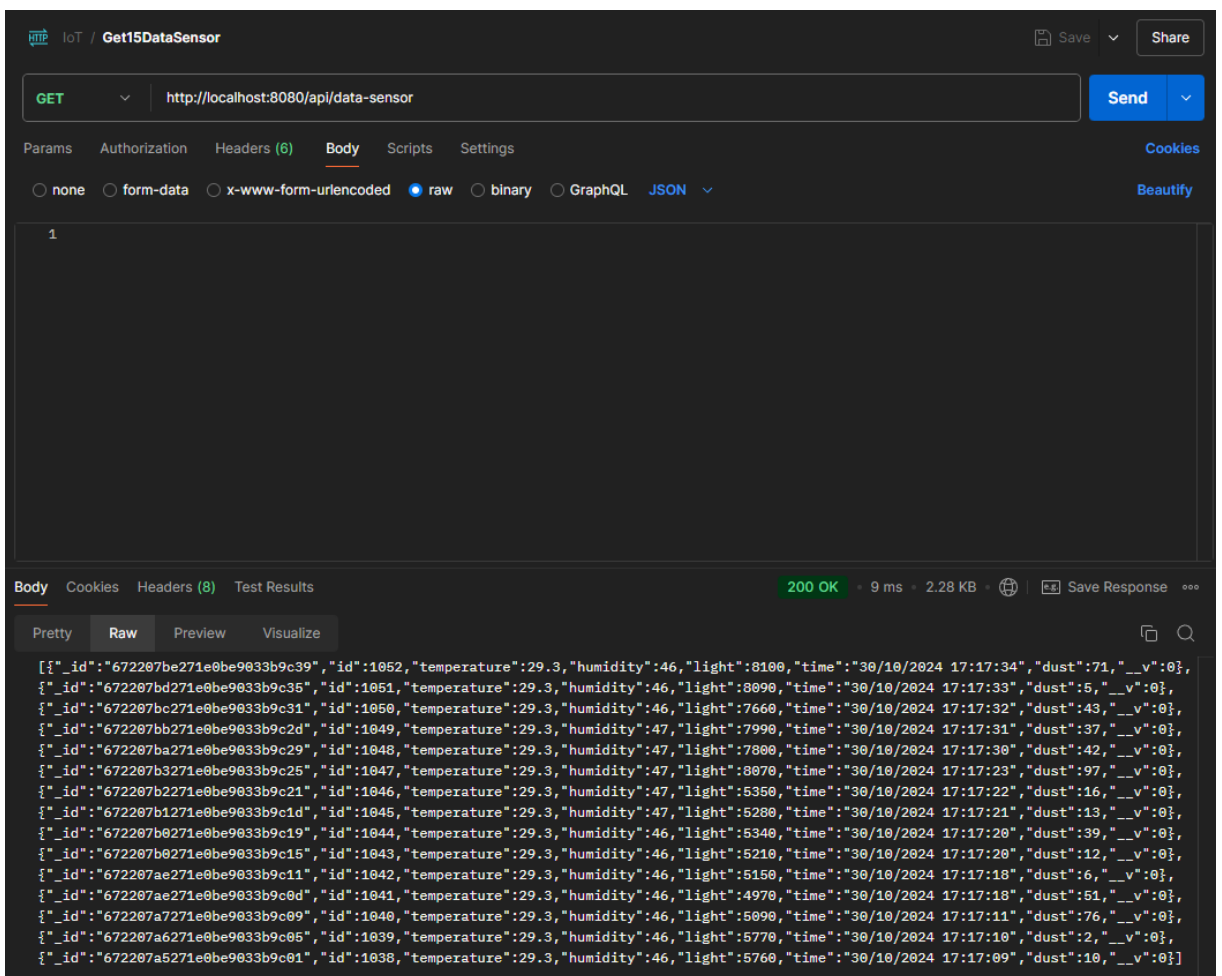
{
  "data": [
    {
      "_id": 55,
      "device": "Refrigerator",
      "action": "On",
      "time": "28/10/2024 23:57:12",
      "__v": 0
    },
    {
      "_id": 28,
      "device": "Refrigerator",
      "action": "Off",
      "time": "28/10/2024 23:12:39",
      "__v": 0
    },
    {
      "_id": 27,
      "device": "Air Conditioner",
      "action": "Off",
      "time": "28/10/2024 23:12:38",
      "__v": 0
    },
    {
      "_id": 26,
      "device": "Fan",
      "action": "Off",
      "time": "28/10/2024 23:12:37",
      "__v": 0
    },
    {
      "_id": 25,
      "device": "Refrigerator",
      "action": "On",
      "time": "28/10/2024 23:12:37",
      "__v": 0
    },
    {
      "_id": 24,
      "device": "Fan",
      "action": "On",
      "time": "28/10/2024 23:12:36",
      "__v": 0
    },
    {
      "_id": 23,
      "device": "Air Conditioner",
      "action": "On",
      "time": "28/10/2024 23:12:36",
      "__v": 0
    },
    {
      "_id": 22,
      "device": "Refrigerator",
      "action": "Off",
      "time": "28/10/2024 23:12:35",
      "__v": 0
    },
    {
      "_id": 21,
      "device": "Air Conditioner",
      "action": "Off",
      "time": "28/10/2024 23:12:35",
      "__v": 0
    },
    {
      "_id": 20,
      "device": "Fan",
      "action": "Off",
      "time": "28/10/2024 23:12:34",
      "__v": 0
    }
  ],
  "currentPage": 1,
  "totalPage": 2,
  "totalData": 19
}

```

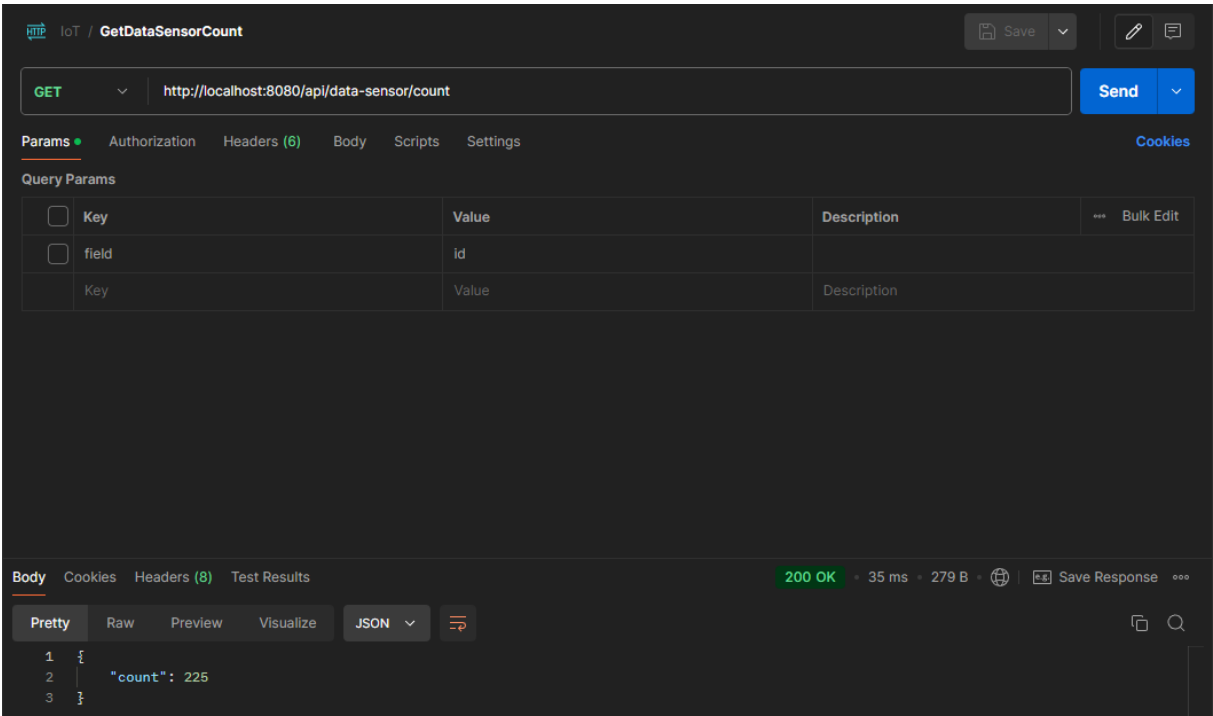
Hình 18: API GetAllControl



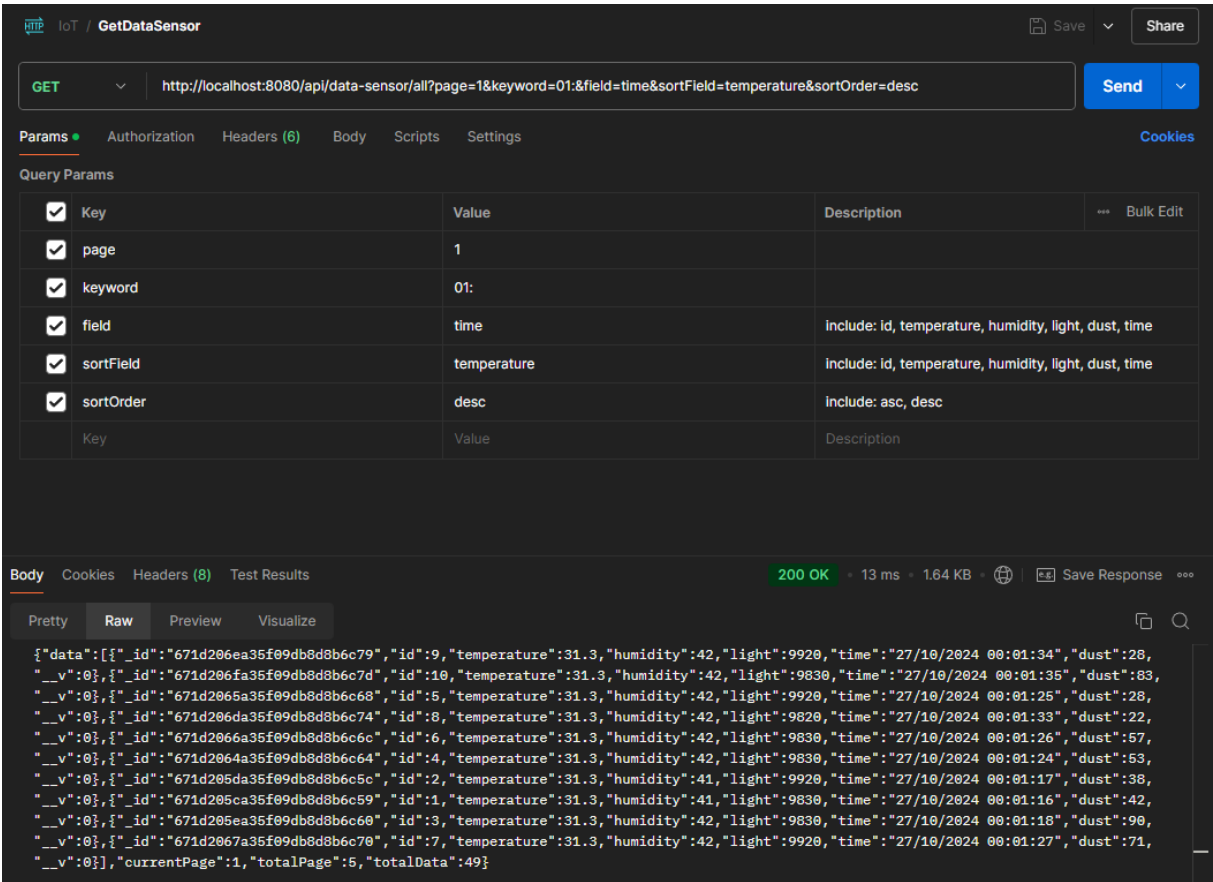
Hình 19: API PostControl



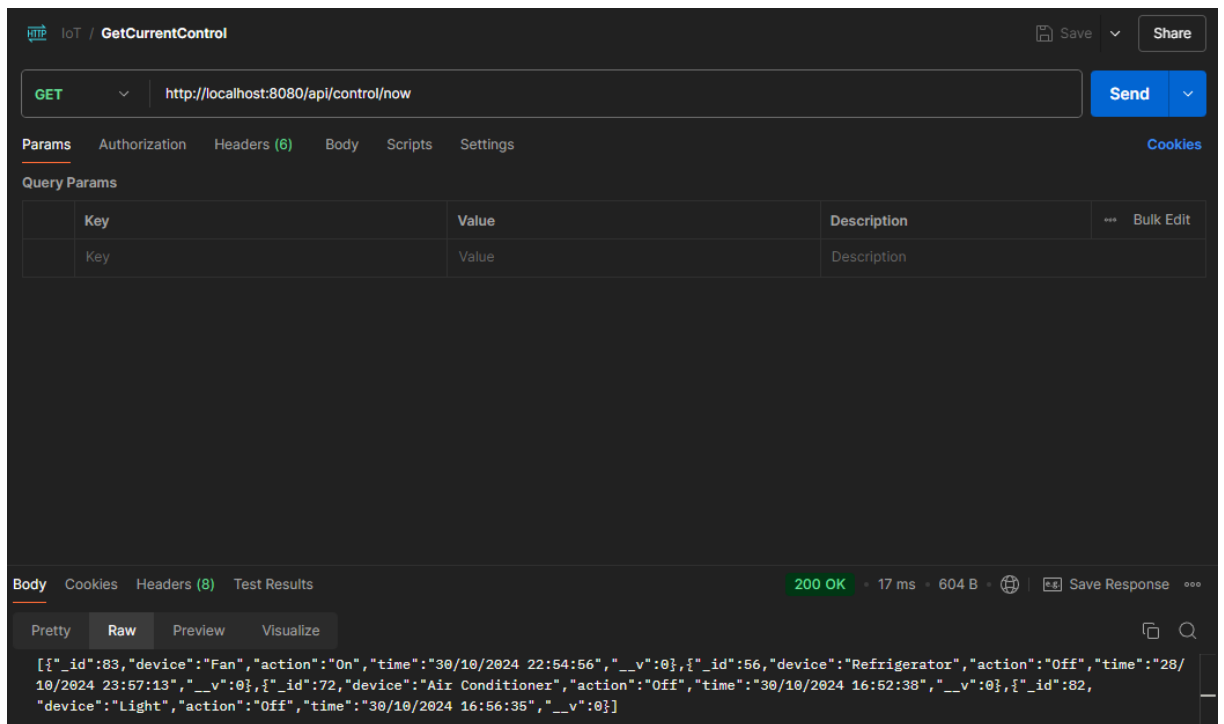
Hình 20: API Get15DataSensor



Hình 21: API `GetDataSensorCount`

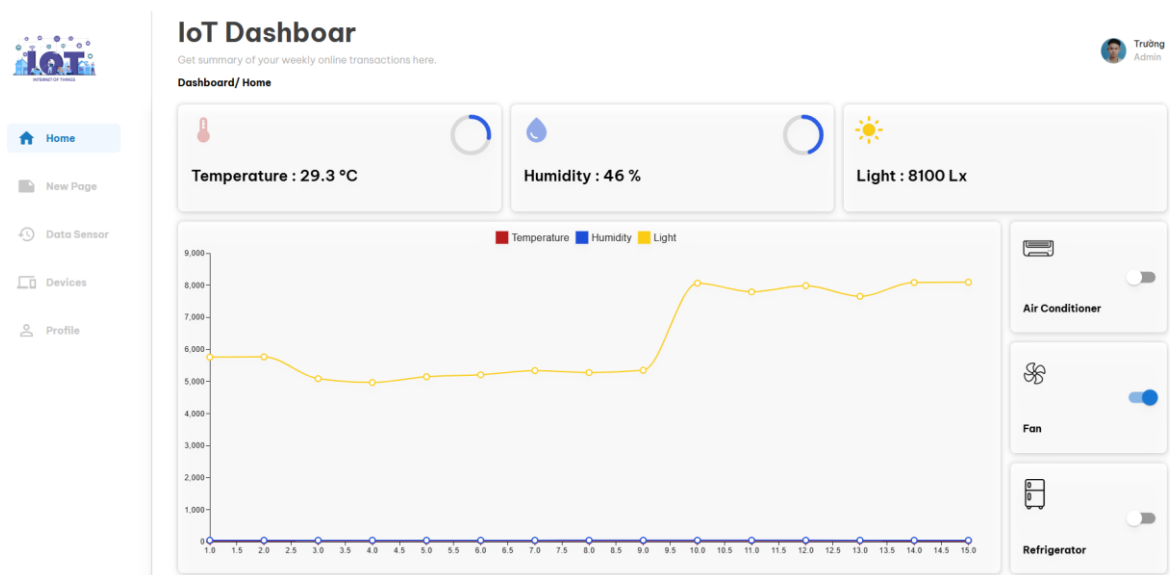


Hình 22: API `GetDataSensor`

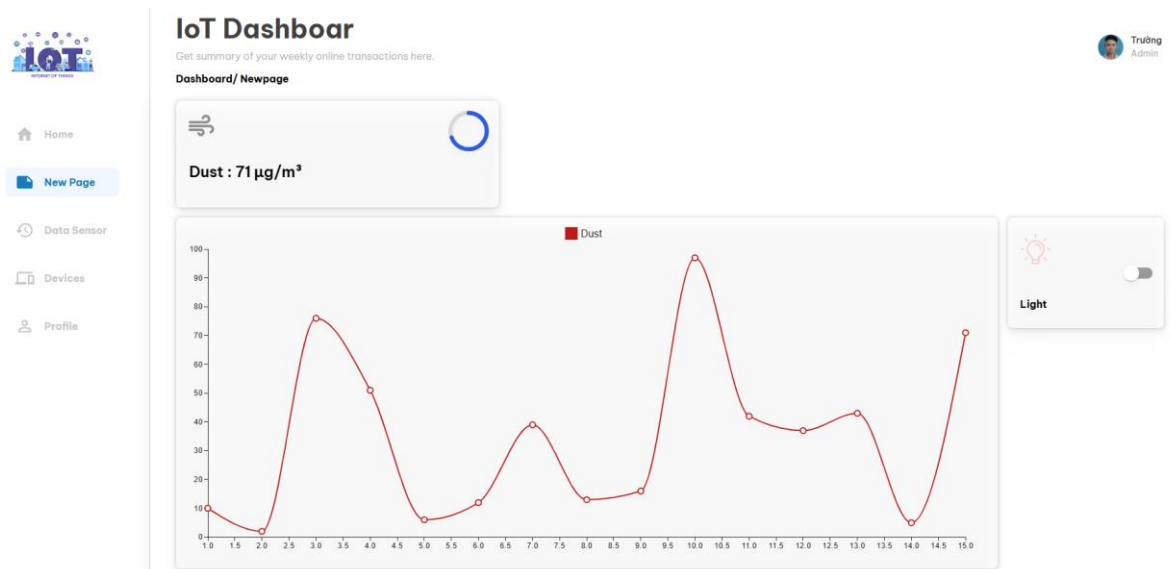


Hình 23: API GetCurrentControl

8. Kết quả



Hình 24: Trang chủ dashboard



Hình 25: Trang New Page

IoT Dashboard
Get summary of your weekly online transactions here.


Dashboard/ Data Sensor

Search... Sort by: --- Ascending

ID	Temperature	Humidity	Light	Dust	Time
1	31.3	41	9830	42	27/10/2024 00:01:16
2	31.3	41	9920	38	27/10/2024 00:01:17
3	31.3	42	9830	90	27/10/2024 00:01:18
4	31.3	42	9830	53	27/10/2024 00:01:24
5	31.3	42	9920	28	27/10/2024 00:01:25
6	31.3	42	9830	57	27/10/2024 00:01:26
7	31.3	42	9920	71	27/10/2024 00:01:27
8	31.3	42	9820	22	27/10/2024 00:01:33
9	31.3	42	9920	28	27/10/2024 00:01:34
10	31.3	42	9830	83	27/10/2024 00:01:35

< 1 2 3 4 5 ... 106 > Go to Page

Hình 26: Trang Data Sensor



Home

New Page

Data Sensor

Devices

Profile

IoT Dashboard

Get summary of your weekly online transactions here.

Trưởng Admin

Dashboard/ Devices

Số lần cảnh báo: 225

Search...


Sort by: --- Ascending

ID	Device	Action	Time
1	Air Conditioner	On	27/10/2024 00:02:10
2	Air Conditioner	Off	27/10/2024 00:02:12
3	Fan	On	27/10/2024 00:02:13
4	Fan	Off	27/10/2024 00:02:14
5	Refrigerator	On	27/10/2024 00:02:15
6	Refrigerator	Off	27/10/2024 00:02:16
7	Light	On	27/10/2024 00:02:18
8	Light	Off	27/10/2024 00:02:19
9	Air Conditioner	On	28/10/2024 23:11:57
10	Fan	On	28/10/2024 23:11:58

< 1 2 3 4 5 ... 9 >

Go to Page

Hình 27: Trang Devices



Home

New Page

Data Sensor

Devices

Profile

IoT Dashboard

Get summary of your weekly online transactions here.

Trưởng Admin

Dashboard/ Profile

Name:

Student ID:

Email:

Report Document:


Github:


API Docs:


Nguyễn Trọng Trường


B21DCCN740

TruongNT.B21DCCN740@stu.ptit.edu.vn













Hello, I'm Truong!

I'm student majoring in Information Technology at Posts and Telecommunications Institute of Technology



Hình 28: Trang Profile