HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÁO CÁO BÀI TẬP LỚN
IOT VÀ ỨNG DỤNG

Đề tài: Nhà thông minh

Giảng viên: Nguyễn Quốc Uy

Nhóm môn học: 05

Họ và tên: Đào Xuân Trí

Mã sinh viên: B21DCCN721

Lớp: D21CNPM1

Hà Nội, tháng 10/2024

LỜI CẢM ƠN

Đầu tiên, em xin gửi lời cảm ơn sâu sắc đến Học viện nghệ Bưu chính Viễn thông và khoa CNTT1 đã đưa môn học IoT và ứng dụng vào trong chương trình giảng dạy. Đặc biệt, em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn Nguyễn Quốc Uy đã dạy dỗ, rèn luyện và truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua.

Trong thời gian được tham dự lớp học của thầy, em đã được tiếp thu thêm nhiều kiến thức bổ ích, học tập được tinh thần làm việc hiệu quả, nghiêm túc. Đây thực là những điều rất cần thiết cho quá trình học tập và công tác sau này của em. Thêm vào đó, nhờ sự dẫn dắt và chỉ bảo của thầy, chúng em đã thực hiện được một đề tài bài tập lớn hoàn chỉnh cho môn học này, chúng em rất biết ơn điều đó.

Em xin chân thành cám ơn, chúc thầy luôn khỏe mạnh và tiếp tục đạt được nhiều thành công trong cuộc sống ạ!

Mục lục

I.Giới thiệu

1.IoT là gì?

Thuật ngữ IoT hay Internet vạn vật đề cập đến mạng lưới tập hợp các thiết bị thông minh và công nghệ tạo điều kiện thuận lợi cho hoạt động giao tiếp giữa thiết bị và đám mây cũng như giữa các thiết bị với nhau. Nhờ sự ra đời của chip máy tính giá rẻ và công nghệ viễn thông băng thông cao, ngày nay, chúng ta có hàng tỷ thiết bị được kết nối với internet. Điều này nghĩa là các thiết bị hàng ngày như bàn chải đánh răng, máy hút bụi, ô tô và máy móc có thể sử dụng cảm biến để thu thập dữ liệu và phản hồi lại người dùng một cách thông minh.

Internet vạn vật tích hợp "vạn vật" với Internet mỗi ngày. Các kỹ sư máy tính đã và đang thêm các cảm biến và bộ xử lý vào các vật dụng hàng ngày kể từ những năm 90. Tuy nhiên, tiến độ ban đầu rất chậm vì các con chip còn to và cồng kềnh. Loại chip máy tính công suất thấp gọi là thẻ tag RFID, lần đầu tiên được sử dụng để theo dõi các thiết bị đắt đỏ. Khi kích cỡ của các thiết bị điện toán dần nhỏ lại, những con chip này cũng trở nên nhỏ hơn, nhanh hơn và thông minh hơn theo thời gian.

Chi phí tích hợp công suất điện toán vào trong các vật dụng nhỏ bé hiện nay đã giảm đáng kể. Ví dụ: bạn có thể thêm khả năng kết nối với các tính năng của dịch vụ giọng thoại Alexa vào các MCU tích hợp sẵn RAM chưa đến 1 MB, chẳng hạn như cho công tắc đèn. Nguyên cả một ngành công nghiệp đã bất ngờ xuất hiện với trọng tâm xoay quanh việc trang bị các thiết bị IoT khắp mọi ngóc ngách căn nhà, doanh nghiệp và văn phòng của chúng ta. Những vật dụng thông minh này có thể tự động truyền và nhận dữ liệu qua Internet. Tất cả những "thiết bị điện toán vô hình" này và công nghệ liên quan được gọi chung là Internet vạn vật.

2.Dự án nhà thông minh

Dự án quản lý nhà thông minh IoT cho phép người dùng theo dõi và điều khiển các thiết bị trong nhà một cách dễ dàng thông qua mạng internet. Hệ thống sử dụng các cảm biến như DHT11 để đo nhiệt độ và độ ẩm, cảm biến ánh sáng để theo dõi cường độ sáng, và các thiết bị điều khiển (đèn, quạt, máy điều hòa, v.v.) được bật tắt từ xa. Các thành phần chính bao gồm:

- ESP8266: Vi điều khiển đóng vai trò giao tiếp giữa cảm biến và các thiết bị điều khiển với nền tảng IoT.

- Mosquitto mqtt: Giao thức truyền thông tin giữa các thiết bị và server, cho phép người dùng theo dõi và điều khiển thiết bị qua điện thoại hoặc máy tính từ bất kỳ đâu.

- Cơ sở dữ liệu MySQL: Lưu trữ dữ liệu cảm biến (nhiệt độ, độ ẩm, ánh sáng) và trạng thái của các thiết bị trong hệ thống.

- Frontend: Giao diện điều khiển web, hiển thị dữ liệu và điều khiển các thiết bị thiết bị trong thời gian thực.

- Backend: nhận và xử lý dữ liệu từ được gửi tới từ mqtt, cập nhật trạng thái thiết bị và cảm biến vào cơ sở dữ liệu; gửi dữ liệu lên giao diện.

Người dùng có thể bật tắt thiết bị, giám sát môi trường trong nhà (nhiệt độ, độ ẩm, ánh sáng) và xem lại lịch sử hoạt động của các thiết bị để tối ưu hóa việc sử dụng năng lượng và tạo môi trường sống thoải mái.

II.Thiết kế giao diện

1.Thiết kế tổng thể

Giao diện của hệ thống gồm có 4 trang lần lượt:

- Trang dashboard: Hiển thị giá trị nhiệt độ, độ ẩm, ánh sáng được gửi từ các cảm biến tới theo thời gian thực; một biểu đồ để thấy sự thay đổi của các giá trị cảm biến theo thời gian; một bảng điều khiển để bật tắt các thiết bị.
- Trang datasensor: Hiển thị dữ liệu của các cảm biến dưới dạng bảng gồm các cột id, nhiệt độ, độ ẩm, ánh sáng, thời gian; có ô tìm kiếm để lọc theo các giá trị cụ thể.
- Trang history: Hiển thị trạng thái bật tắt của các thiết bị dưới dạng bảng gồm các cột id, tên thiết bị, trạng thái, thời gian; có ô tìm kiếm để lọc theo thời gian.
- Trang profile: Hiển thị các thông tin cá nhân.

2.Thiết kế chi tiết

Giao diện hệ thống được thiết kế dựa trên framework React.js.Nó là một thư viện javascript được tạo ra bởi sự cộng tác giữa Facebook và Instagram. Nó cho phép những nhà phát triển web tạo ra giao diện người dung nhanh chóng. Phần Views của React.js thường được hiển thị bằng việc chủ yếu dùng các component mà chứa trong các component cụ thể là các thẻ HTML.Ngoài ra React.js còn được xây dựng dựa trên kiến trúc SPA (Single Page Application) với nhiều ưu điểm:

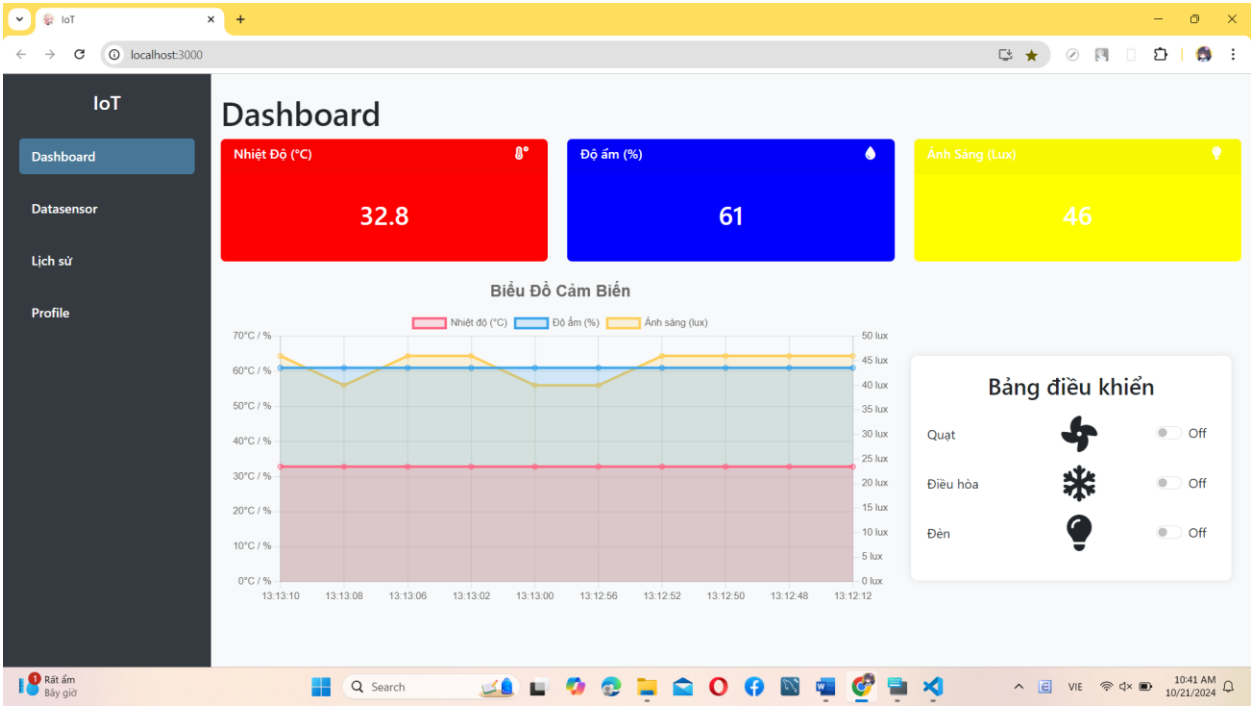
- Tốc độ tải trang nhanh hơn: SPA chỉ tải một lần duy nhất và sau đó chỉ cập nhật nội dung khi cần thiết mà không cần phải tải lại toàn bộ trang. Điều này

giúp ứng dụng phản hồi nhanh hơn khi người dùng thao tác, tạo trải nghiệm mượt mà hơn so với MPA.

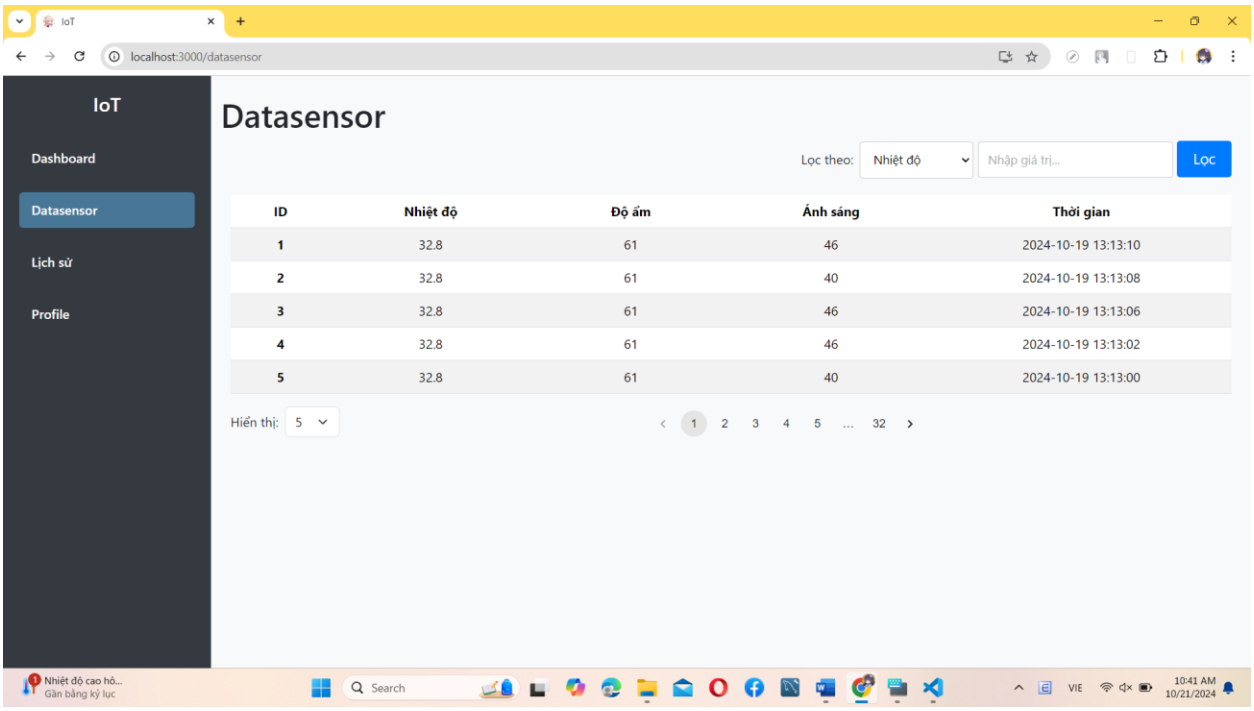
- Giảm băng thông: SPA chỉ yêu cầu tải dữ liệu khi cần, thay vì tải lại toàn bộ trang khi có thay đổi như trong MPA. Điều này giúp tiết kiệm băng thông và tăng hiệu quả trong việc truyền dữ liệu.
- Trải nghiệm người dùng tốt hơn: SPA cung cấp cảm giác giống như một ứng dụng di động nhờ việc không cần tải lại trang khi chuyển trang. Điều này giúp giảm thời gian chờ đợi và cải thiện trải nghiệm người dùng.
- Tương tác tức thời với API: SPA thường sử dụng AJAX để giao tiếp với server, gửi và nhận dữ liệu theo thời gian thực mà không cần tải lại trang. Điều này giúp SPA phản hồi nhanh và tức thời khi tương tác với API.
- Tách biệt frontend và backend: SPA cho phép tách biệt hoàn toàn giữa frontend và backend, giúp dễ bảo trì, phát triển và tái sử dụng code ở các phần khác nhau của hệ thống.

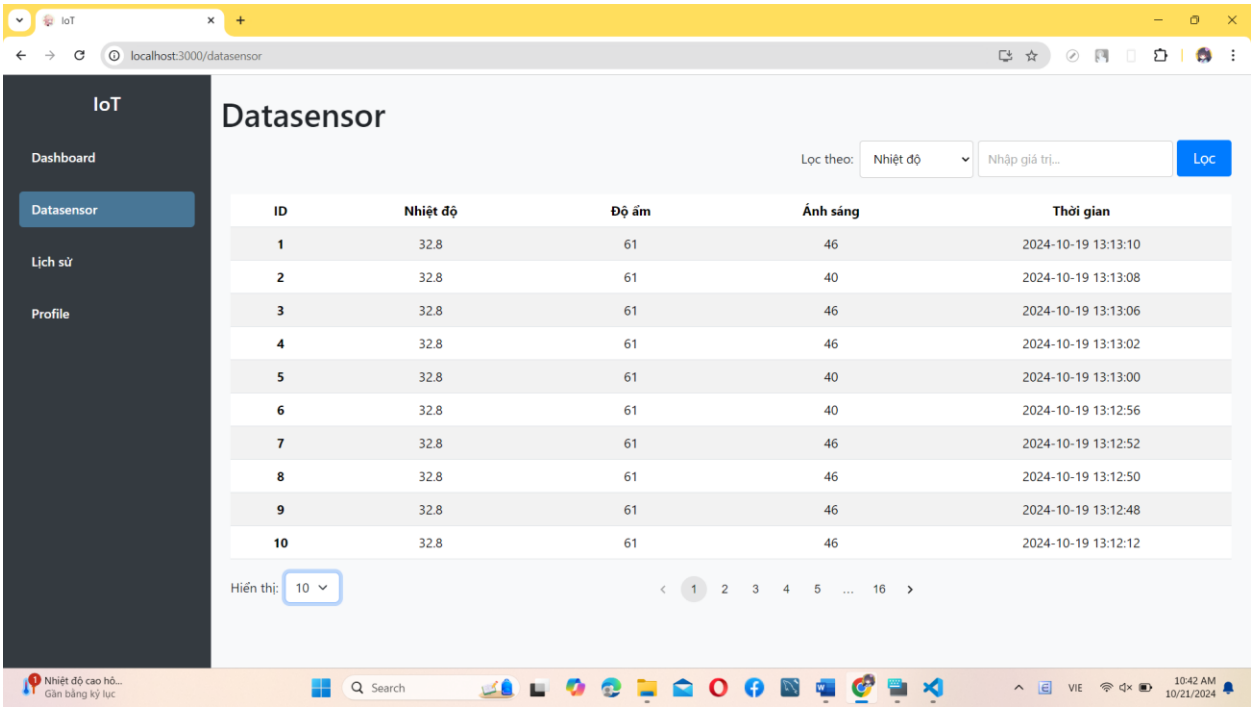
Giao diện chi tiết của hệ thống

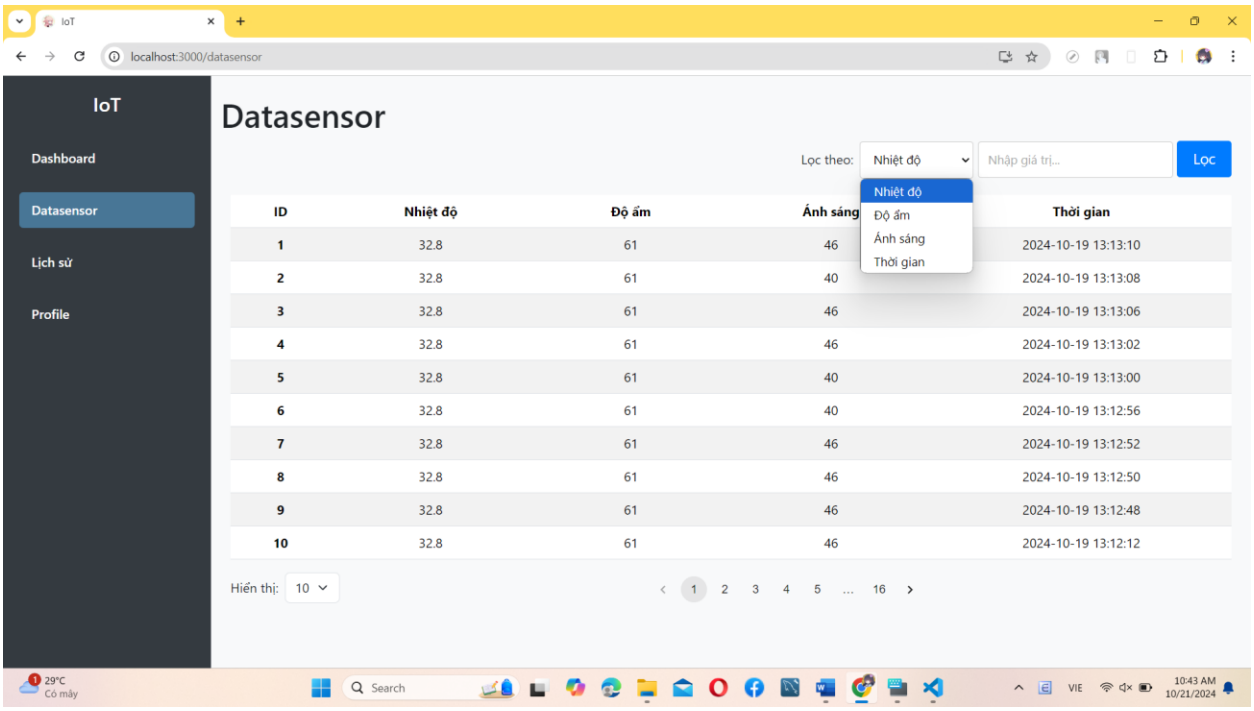
- Giao diện dashboard



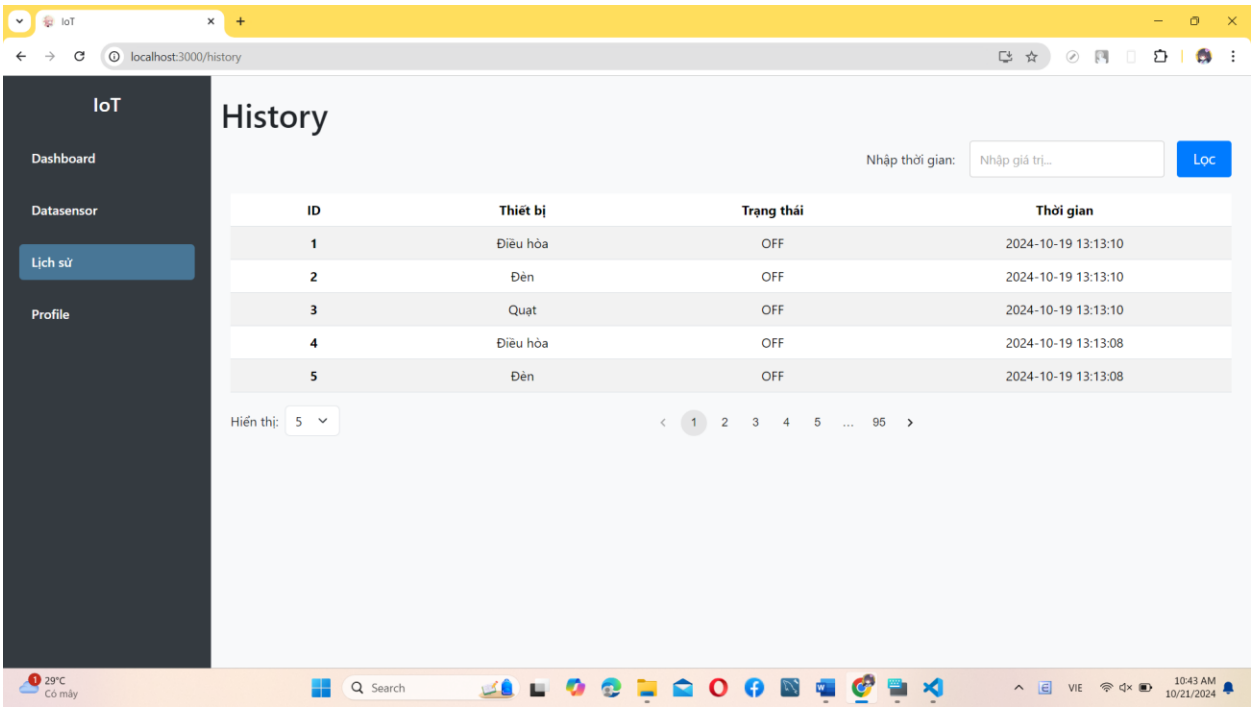
- Giao diện datasensor

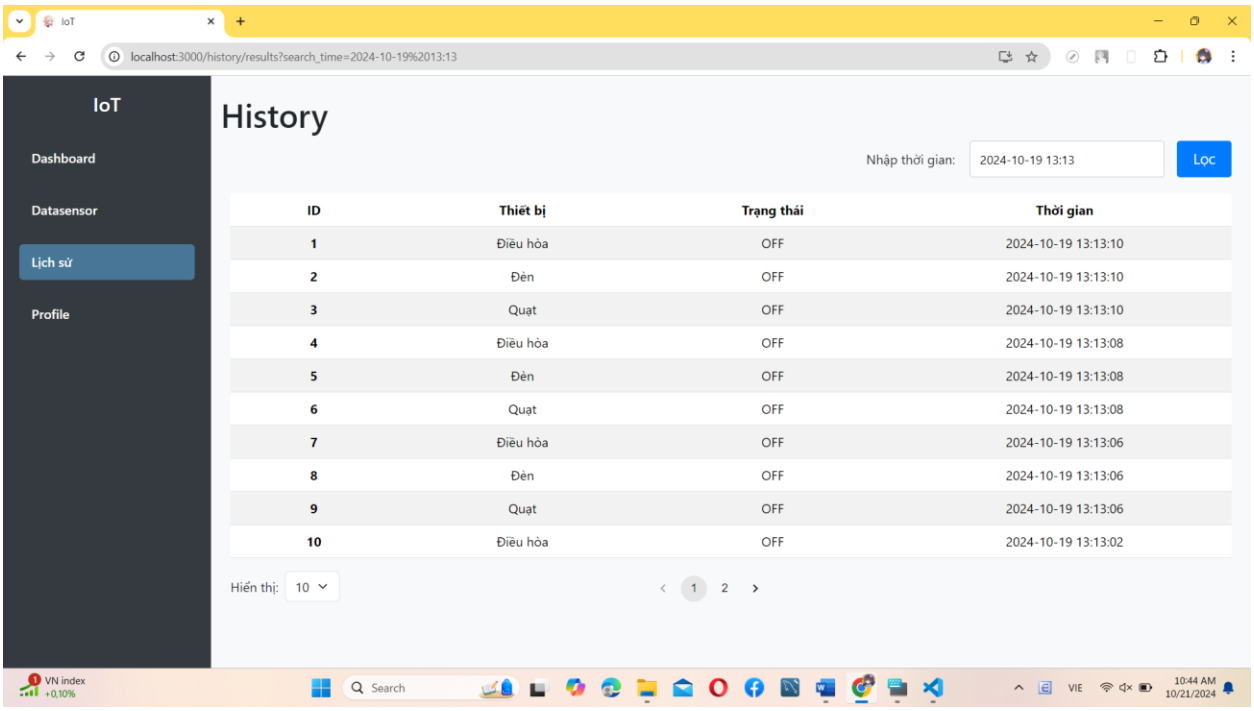

IoT
localhost:3000/datasensor
IoT
Dashboard
Datasensor
Lịch sử
Profile
Datasensor
Lọc theo: Nhiệt độ
Nhập giá trị...
Lọc
ID Nhiệt độ Độ ẩm Ánh sáng Thời gian
1 32.8 61 46 2024-10-19 13:13:10
2 32.8 61 40 2024-10-19 13:13:08
3 32.8 61 46 2024-10-19 13:13:06
4 32.8 61 46 2024-10-19 13:13:02
5 32.8 61 40 2024-10-19 13:13:00
Hiển thị: 5
1 2 3 4 5 ... 32
Search
10:41 AM
10/21/2024


IoT
localhost:3000/datasensor
IoT
Dashboard
Datasensor
Lịch sử
Profile
Datasensor
Lọc theo: Nhiệt độ
Nhập giá trị...
Lọc
ID Nhiệt độ Độ ẩm Ánh sáng Thời gian
1 32.8 61 46 2024-10-19 13:13:10
2 32.8 61 40 2024-10-19 13:13:08
3 32.8 61 46 2024-10-19 13:13:06
4 32.8 61 46 2024-10-19 13:13:02
5 32.8 61 40 2024-10-19 13:13:00
6 32.8 61 40 2024-10-19 13:12:56
7 32.8 61 46 2024-10-19 13:12:52
8 32.8 61 46 2024-10-19 13:12:50
9 32.8 61 46 2024-10-19 13:12:48
10 32.8 61 46 2024-10-19 13:12:12
Hiển thị: 10
1 2 3 4 5 ... 16
Search
10:42 AM
10/21/2024

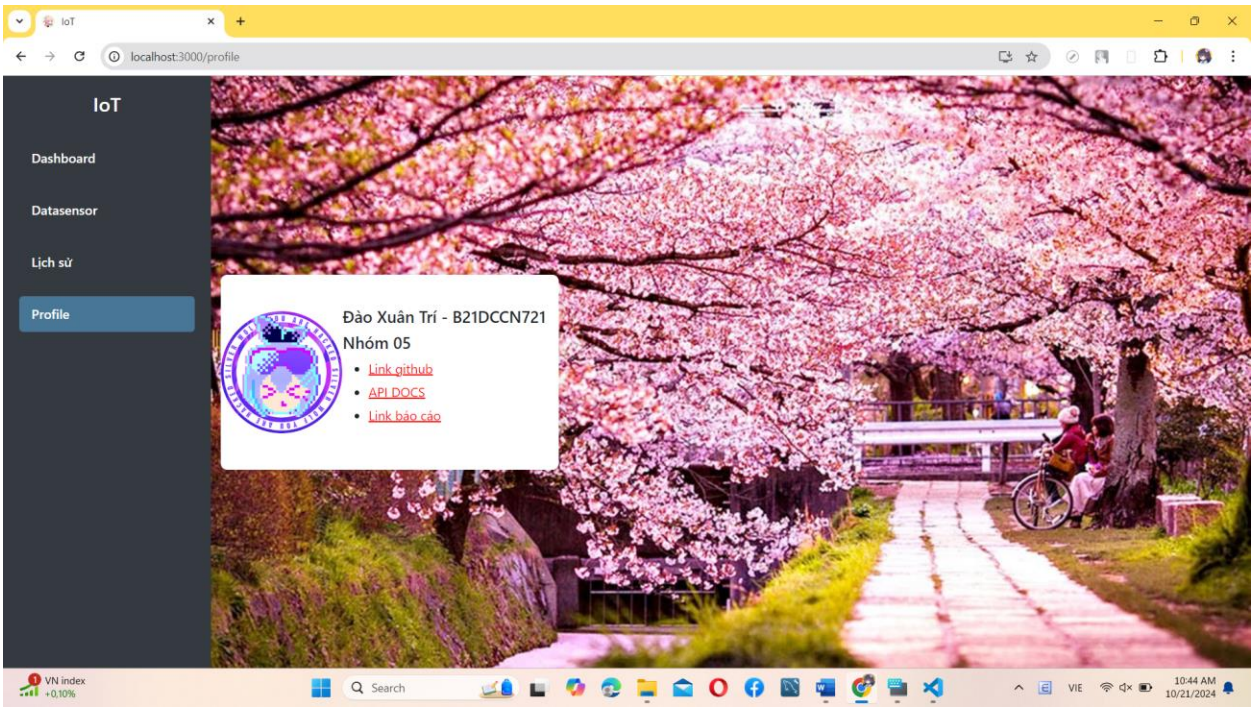


- Giao diện lịch sử





- Giao diện profile



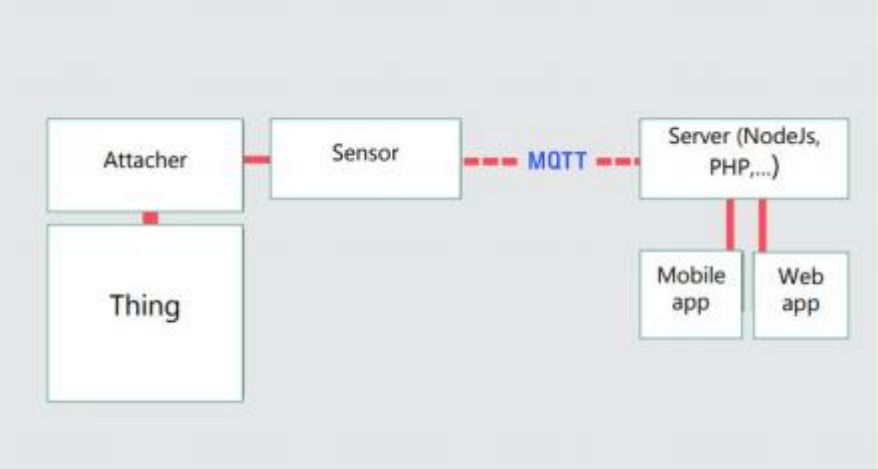
III.Thiết kế Server

1.Giao thức truyền tin giữa hardware và Server

IoT đã phát triển từ sự hội tụ của công nghệ không dây, công nghệ vi cơ điện tử và Internet. Nói đơn giản là một tập hợp các thiết bị có khả năng kết nối với nhau, với Internet và với thế giới bên ngoài để thực hiện một công việc nào đó. Thời đại IoT yêu cầu một giao thức kết nối mới để đảm bảo hỗ trợ đầy đủ cho các thiết bị vật lý thực tế. Để giải quyết vấn đề này, Message Queuing Telemetry Transport (MQTT) đang dần trở nên phổ biến.

MQTT (Message Queuing Telemetry Transport) là giao thức truyền thông điệp (message) theo mô hình publish/subscribe (cung cấp / thuê bao), được sử dụng cho các thiết bị IoT với băng thông thấp, độ tin cậy cao và khả năng được sử dụng trong mạng lưới không ổn định. Nó dựa trên một Broker (tạm dịch là "Máy chủ môi giới") "nhẹ" (khá ít xử lý) và được thiết kế có tính mở (tức là không đặc trưng cho ứng dụng cụ thể nào), đơn giản và dễ cài đặt.

Một số ưu điểm nổi bật của MQTT như: băng thông thấp, độ tin cậy cao và có thể sử dụng ngay cả khi hệ thống mạng không ổn định, tốn rất ít byte cho việc kết nối với server và connection có thể giữ trạng thái open xuyên suốt, có thể kết nối nhiều thiết bị (MQTT client) thông qua một MQTT server (broker). Bởi vì giao thức này sử dụng băng thông thấp trong môi trường có độ trễ cao nên nó là một giao thức lý tưởng cho các ứng dụng IoT.



Để sử dụng giao thức mqtt trong ứng dụng IoT thì có khá nhiều broker khác nhau được viết bằng nhiều loại ngôn ngữ lập trình khác nhau.Trong đó một trong những broker phổ biến nhất chính là Mosquitto. Mosquitto là một MQTT Broker mã nguồn

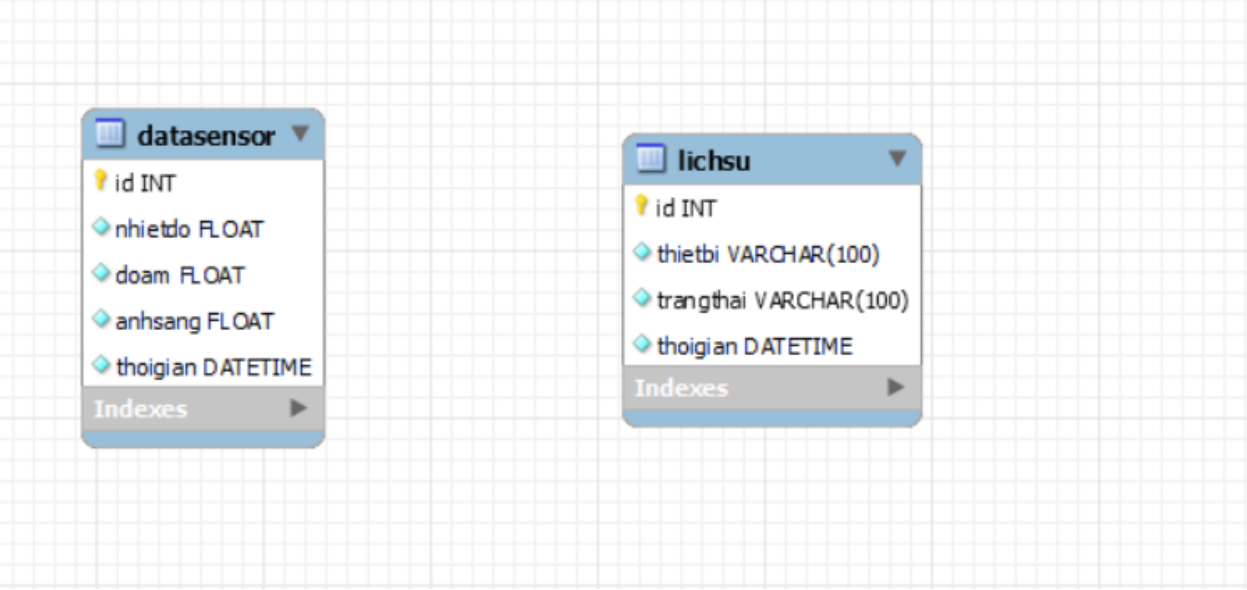
mở cho phép thiết bị truyền nhận dữ liệu theo giao thức MQTT version 5.0, 3.1.1 và 3.1 – Một giao thức nhanh, nhẹ theo mô hình publish/subscribe được sử dụng rất nhiều trong lĩnh vực IoT. Mosquitto cung cấp một thư viện viết bằng ngôn ngữ C để triển khai các MQTT Client và có thể dễ dàng sử dụng bằng dòng lệnh: "mosquitto_pub" và "mosquitto_sub". Nó có nhiều ưu điểm như:

- Tốc độ truyền nhận và xử lí dữ liệu nhanh, độ ổn định cao, được sử dụng rộng rãi và phù hợp với những ứng dụng embedded.
- Rất nhẹ và phù hợp để sử dụng trên tất cả các thiết bị.
- Được hỗ trợ các giao thức TLS/SSL (các giao thức nhằm xác thực server và client, mã hóa các message để bảo mật dữ liệu).

2.Database

Cơ sở dữ liệu được sử dụng trong hệ thống là mysql gồm có 2 bảng là datasensor và lichsu.

- Bảng datasensor lưu trữ dữ liệu của các cảm biến ứng với các giá trị nhiệt độ, độ ẩm, ánh sáng và thời gian dữ liệu được lưu vào database
- Bảng lichsu lưu trữ trạng thái bật tắt của các thiết bị trong thời gian tương ứng.



3.Backend

Backend được xây dựng dựa trên node.js với framework được sử dụng là Express.js. Expressjs là một framework được xây dựng trên nền tảng của Nodejs. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile. Expressjs hỗ trợ các method HTTP và midleware tạo ra API vô cùng mạnh mẽ và dễ sử dụng. Nó còn hỗ trợ định tuyến URL thông qua một cú pháp rất rõ ràng, cho phép bạn dễ dàng xử lý các yêu cầu HTTP như GET, POST, PUT, DELETE; cho phép sử dụng các hàm middleware để xử lý các yêu cầu trước khi đến các route cụ thể, Middleware có thể được sử dụng để xác thực, kiểm tra quyền hạn, hoặc thao tác với dữ liệu yêu cầu; cung cấp các công cụ đơn giản để làm việc với yêu cầu và phản hồi HTTP, giúp việc xử lý form, cookies, tải lên tệp, v.v. trở nên dễ dàng.Các ưu điểm nổi bật của Express.js có thể kể tới như sau:

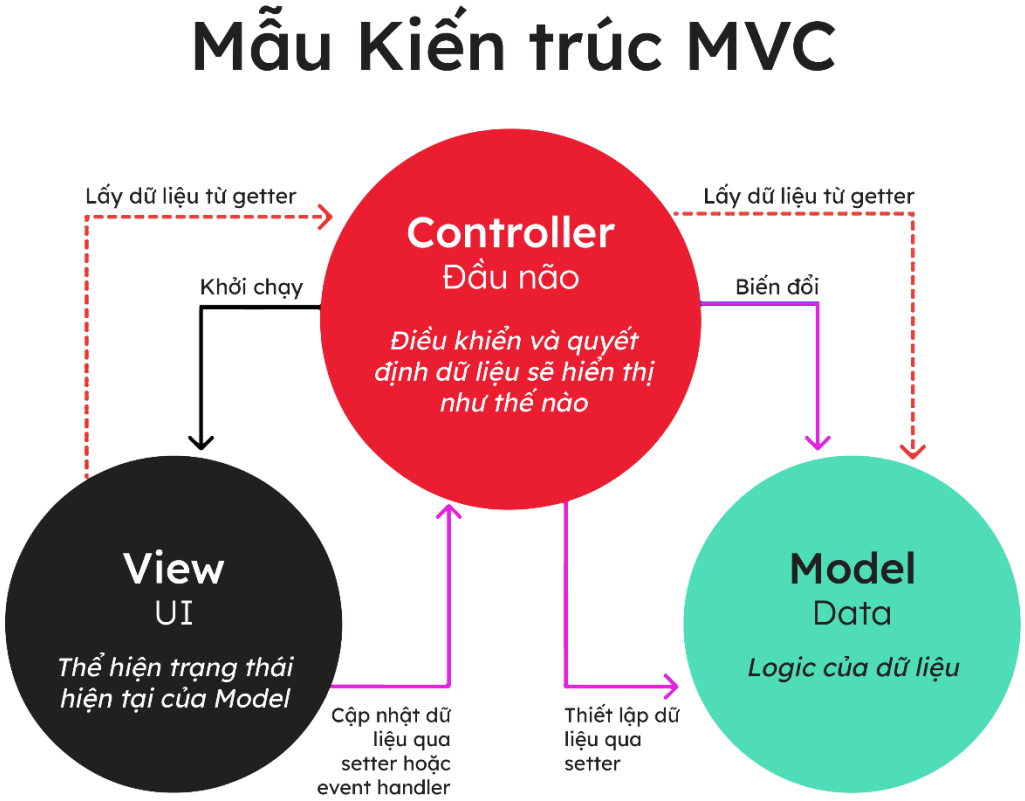
- Hiệu năng cao: Express.js được xây dựng trên nền tảng Node.js, vốn được biết đến với khả năng xử lý không đồng bộ và không bị chặn. Điều này giúp tăng tốc độ phản hồi và hiệu suất khi xây dựng các ứng dụng web quy mô lớn.
- Tối giản và linh hoạt: Express cung cấp các công cụ cơ bản nhất cần thiết cho việc phát triển ứng dụng, không có quá nhiều tính năng phức tạp dư thừa. Điều này giúp nhà phát triển linh hoạt lựa chọn các công cụ bổ sung khác nếu cần.
- Dễ học và sử dụng: Express.js có cấu trúc và cú pháp đơn giản, dễ hiểu, giúp cho những người mới bắt đầu làm quen với Node.js hoặc phát triển web nhanh chóng nắm bắt và triển khai.
- Xử lý routing dễ dàng: Routing trong Express.js được tổ chức một cách rõ ràng, giúp việc xử lý các request cho các endpoint khác nhau của API hoặc trang web trở nên dễ dàng hơn; cú pháp đơn giản và dễ nhớ, cho phép bạn định nghĩa các route cho ứng dụng với các phương thức HTTP như GET, POST, PUT, DELETE.

Backend của hệ thống còn được xây dựng dựa trên mô hình MVC. MVC là viết tắt của Model-View-Controller. Cấu trúc Model-View-Controller (MVC) là một mẫu kiến trúc/mẫu thiết kế (design pattern) tách ứng dụng thành ba thành phần logic chính: Model, View và Controller. Mỗi thành phần kiến trúc được xây dựng để xử lý các khía cạnh phát triển cụ thể của một ứng dụng.Đặc điểm của mô hình MVC:

- Cung cấp sự phân tách rõ ràng giữa logic nghiệp vụ, logic Ul và logic đầu vào.
- Cung cấp toàn quyền kiểm soát HTML và URL, giúp bạn dễ dàng thiết kế kiến trúc ứng dụng web.

- Có thể sử dụng để xây dựng các ứng dụng có URL dễ hiểu và có thể tìm kiếm được.
- Hỗ trợ Lập trình dựa trên thử nghiệm (Test-driven Development).

Các thành phần chính của mô hình MVC

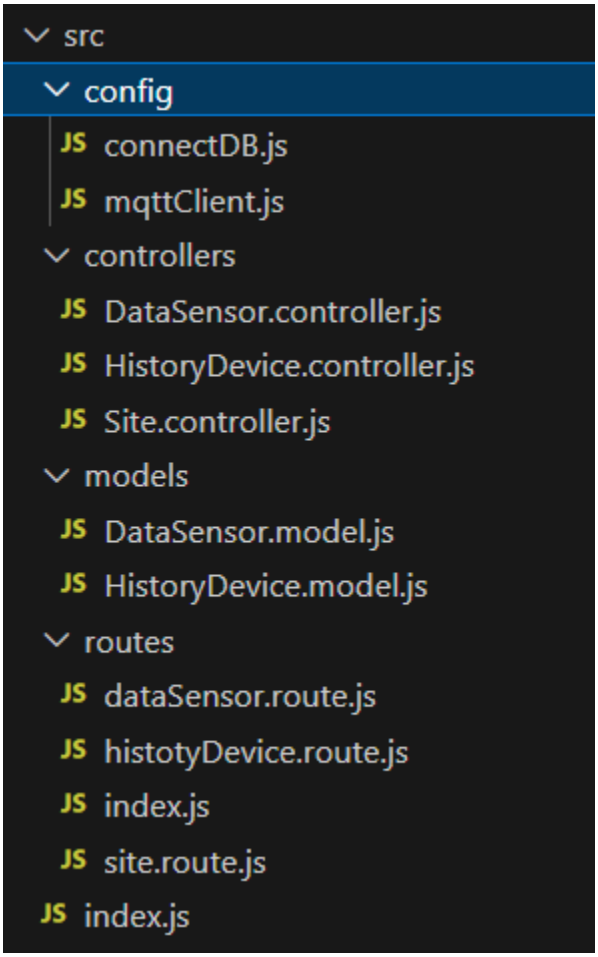


- **Model** đại diện cho lớp xử lý dữ liệu của ứng dụng. Nó chịu trách nhiệm quản lý dữ liệu, tương tác với cơ sở dữ liệu, và áp dụng các quy tắc kinh doanh của doanh nghiệp. Mô hình có thể bao gồm các tác vụ như lưu trữ, truy xuất, cập nhật và xóa dữ liệu. Model không biết về View hoặc Controller, mà chỉ cung cấp dữ liệu cho Controller khi được yêu cầu.
- **View** chịu trách nhiệm hiển thị dữ liệu cho người dùng. Đây là lớp giao diện của ứng dụng, nơi người dùng tương tác trực tiếp. View sẽ nhận dữ liệu từ Controller và trình bày nó dưới dạng đồ họa hoặc văn bản để người dùng có thể hiểu và sử dụng. View không có logic xử lý dữ liệu hay logic kinh doanh, nó chỉ nhận và hiển thị dữ liệu.Trong hệ thống của ứng dụng này view được tách riêng và thiết kế bởi React.js như đã được trình bày ở trên.

- **Controller** đóng vai trò trung gian giữa Model và View. Controller nhận các yêu cầu từ người dùng (thông qua View), sau đó yêu cầu Model xử lý dữ liệu và cuối cùng gửi dữ liệu đã xử lý về View để hiển thị. Controller chịu trách nhiệm kiểm soát luồng của ứng dụng, thực thi các logic, xử lý các sự kiện người dùng như click chuột, gửi form, v.v.

4.Chi tiết backend.

Cấu trúc thư mục backend



- Thư mục config chứa các file thực hiện kết nối nối database, mqtt broker.
    - File connectDB.js

```
const { Sequelize } = require("sequelize");
const sequelize = new Sequelize("iot", "root", "123456", {
  host: "localhost",
  dialect: "mysql",
  timezone: '+07:00',
});
const connection = async () => {
  try {
    await sequelize.authenticate();
    console.log("Connection db has been established successfully.");
  } catch (error) {
    console.error("Unable to connect to the database:", error);
  }
};

module.exports = { connection, sequelize };
```

- File mqttClient.js

```
const mqtt = require('mqtt')
const  DataSensor  = require('../models/DataSensor.model')
const  HistoryDevice  = require('../models/HistoryDevice.model')

const client = mqtt.connect('mqtt://192.168.43.22:2003', {
    username: 'xuantri',
    password: 'B21DCCN721'
});

client.on('connect', () => {
    console.log('Connected to MQTT broker');
    client.subscribe('data');
    client.subscribe('led');
    client.subscribe('cnt');
});

function mqttClient() {···
}

module.exports = { mqttClient, client }
```

- Thư mục controllers chứa các file xử lý logic tướng ứng với lớp controller của mô hình mvc.
  - File DataSensor.controller.js chứa các phương thức xử lý các logic tương ứng với trang Datasensor ở phần giao diện.Phương thức

getAllDatasensors dùng để lấy tất cả dữ liệu từ database và gửi về client để hiển thị lên giao diện; phương thức searchDate dùng để xử lý logic cho mục tìm kiếm ở phía client.

```
const DataSensor = require('../models/DataSensor.model')
const { Op } = require('sequelize')

const getAllDataSensors = async (req, res) => {···
}

const searchData = async (req, res) => {···
}



module.exports = { getAllDataSensors, searchData }
```

- File HistoryDevice.controller.js chứa các phương thức xử lý logic tương ứng với trang lịch sử trên giao diện.Phương thức getAllHistoryDevice dùng để lấy tất cả dữ liệu của các thiết bị từ database gửi về phía client để hiển thị, phương thức searchTime dùng để xử lý logic tìm kiếm theo thời gian ở phía client.

```
const HistoryDevice = require('../models/HistoryDevice.model')
const { Op } = require('sequelize')

const getAllHistoryDevice = async (req, res) => {···
}
const searchTime = async (req, res) => {···
}

module.exports = { getAllHistoryDevice, searchTime }
```

- File Site.controller.js chứa các phương thức xử lý logic ở trang dashboard.Phương thức getNewData dùng để lấy ra dữ liệu mới nhất của cảm biến gửi về, phương thức getDataChart dùng để lấy dữ liệu để hiển thị cho phần biểu đồ, phương thức controllDevice dùng để xử lý logic bật tắt các thiết bị, phương thức getStatusDevice dùng để lấy ra

trạng thái hiện tại của các thiết bị giúp đồng bộ hóa trạng thái bật tắt của thiết bị với màn hình hiển thị phía client.

```
const DataSensor = require('../models/DataSensor.model')
const HistoryDevice = require('../models/HistoryDevice.model')
const { client } = require('../config/mqttClient')

const getNewData = async (req, res) => {···
}

const getDataChart = async (req, res) => {···
}
const controllDevice = async (req, res) => {···
}

const getStatusDevice = async (req, res) => {···
}



module.exports = { getNewData, getDataChart, controllDevice, getStatusDevice }
```

- Thư mục models chứa các model để tương tác với database gồm 2 model là Datasensor và HistoryDevice

```
const { Sequelize, DataTypes } = require("sequelize");
const { sequelize } = require("../config/connectDB");

const DataSensor = sequelize.define("DataSensor", {···
});

module.exports = DataSensor;
```

```
const { Sequelize, DataTypes } = require("sequelize");
const { sequelize } = require("../config/connectDB");

const HistoryDevice = sequelize.define("HistoryDevice", {···
});

module.exports = HistoryDevice;
```

- Thư mục routes chia nhỏ và định tuyến các tuyến đường trong hệ thống

```
// site
const siteRoute = require('./site.route')

// datasensor
const datasensorRoute = require('./dataSensor.route')

//history
const historyRoute = require('./histotyDevice.route')

function route(app){
    app.use('/datasensor', datasensorRoute)
    app.use('/history', historyRoute)
    app.use('/', siteRoute)
}

module.exports = route;
```

  - File dataSensor.route.js định nghĩa các tuyến đường ứng với path /datasensor

```
const express = require('express')
const router = express.Router()

const { searchData, getAllDataSensors } = require('../controllers/DataSensor.controller')

router.get('/search', searchData)
router.get('/',getAllDataSensors)

module.exports = router
```

  - File historyDevice.route.js định nghĩa các tuyến đường ứng với path /history

```
const express = require('express')
const router = express.Router()

const { searchTime, getAllHistoryDevice } = require('../controllers/HistoryDevice.controller')
                                    getAllHistoryD…   (alias) const getAllHistoryDevice: (r…
router.get('/search', searchTime)
router.get('/', getAllHistoryDevice)

module.exports = router
```

- File site.route.js định nghĩa các tuyến đường ứng với path /

```
const express = require('express')
const router = express.Router()

const { getNewData, getDataChart, controllDevice, getStatusDevice } = require('../controllers/Site.controller')

router.get('/status', getStatusDevice)
router.post('/controll', controllDevice)
router.get('/chart', getDataChart)
router.get('/', getNewData)

module.exports = router
```

5.Danh sách các API

Phương thức	API	Param	Body	Response	Vai trò
GET	/			{ "id": 160, "nhietdo": 32.8, "doam": 61, "anhsang": 46, "thoigian": "2024-10-19 13:13:10" }	Lấy ra dữ liệu mới nhất của các cảm biến được lưu vào database và gửi về client
GET	/chart	limit=10		Mảng các dữ liệu	Lấy ra 10 dòng

					dữ liệu mới của các cảm biến được lưu trong database và gửi về client
GET	/status			{ "quat": **false**, "den": **false**, "dieuhoa": **false** }	Lấy ra trạng thái mới nhất của các thiết bị và gửi về client
POST	/controll		deviceId:quat state:true	{ message: 'Gửi yêu cầu thành công' }	Gửi yêu cầu bật tắt đèn tới hardware
GET	/datasensor			Mảng các dữ liệu	Lấy ra tất cả dữ liệu về cảm biến được lưu trong database và

					gửi về client

GET	/datasensor/search	q=nhietdo&value=31.8		[
				{
				"id": 2,
				"nhietdo": 31.8,
				"doam": 67,
				"anhsang": 43,
				"thoigian": "2024-10-19 12:34:41"
				},
				{
				"id": 1,
				"nhietdo": 31.8,
				"doam": 67,
				"anhsang": 43,
				"thoigian": "2024-10-19 12:34:39"
				}
					Lấy ra dữ liệu tìm kiếm theo yêu cầu từ phía client và gửi dữ liệu tìm kiếm được về client

				]	
GET	/history			Mảng các dữ liệu	Lấy ra tất cả dữ liệu về trạng thái các thiết bị và gửi về phía client
GET	/history/search	search_time=2024-10-19 12:34:39		[{ "id": 3, "thietbi": "Điều hòa", "trangthai": "OFF", "thoigian": "2024-10-19 12:34:39" }, { "id": 2, "thietbi": "Đèn", "trangthai": "OFF", "thoigian":	Lấy ra dữ liệu được tìm kiếm theo thời gian và gửi về client

				"2024-10-19 12:34:39"
				},
				{
				"id": 1,
				"thietbi": "Quạt",
				"trangthai": "OFF",
				"thoigian": "2024-10-19 12:34:39"
				}
				]	

IV.Kết quả

Qua quá trình học tập và nghiên cứu, em đã xây dựng thành công một hệ thống IoT ứng dụng trong nhà thông minh. Hệ thống này cho phép giám sát các yếu tố môi trường như nhiệt độ, độ ẩm, và cường độ ánh sáng trong thời gian thực, giúp người dùng nắm bắt chính xác tình trạng của ngôi nhà. Ngoài ra, hệ thống còn tích hợp tính năng điều khiển các thiết bị điện trong nhà từ xa, cho phép bật/tắt dễ dàng thông qua giao diện trực quan. Đặc biệt, hệ thống lưu lại lịch sử hoạt động của các thiết bị, lịch sử dữ liệu của các cảm biến giúp người dùng theo dõi và quản lý hiệu quả hơn.

Với những chức năng trên, hệ thống không chỉ mang lại sự tiện lợi mà còn góp phần nâng cao hiệu quả quản lý năng lượng và tạo ra một không gian sống thông minh, hiện đại hơn.

Tài liệu tham khảo

-Giao thức mqtt: https://viblo.asia/p/mqtt-la-gi-vai-tro-cua-mqtt-trong-iot-V3m5WL3bKO7

-Mosquitto: https://viblo.asia/p/tim-hieu-ve-mqtt-mosquitto-broker-va-cach-cai-dat-yMnKMjgrZ7P

-Pub/sub mqtt: https://microcontrollerslab.com/esp8266-nodemcu-mqtt-publish-subscribe-dht22-readings/