

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN 1**

_____o0o_____



BÁO CÁO BÀI TẬP LỚN
MÔN HỌC: IOT VÀ ỨNG DỤNG

Sinh viên: Nguyễn Đắc Thành
Lớp: D21CQCN06-B
Mã sinh viên: B21DCCN678
Số điện thoại: 0326739576

Giảng viên hướng dẫn: Nguyễn Quốc Uy

HÀ NỘI, 10/2024

LỜI CẢM ƠN

Đầu tiên, em xin gửi lời cảm ơn sâu sắc đến Học viện Công nghệ Bưu chính Viễn thông cùng với Khoa Công nghệ Thông tin 1 đã đưa môn học "IoT và ứng dụng" vào chương trình giảng dạy. Đây là một môn học rất thiết thực và hiện đại, giúp em mở rộng kiến thức và nắm bắt những xu hướng công nghệ mới, từ đó có cái nhìn toàn diện hơn về sự phát triển của công nghệ trong thời đại số.

Đặc biệt, em xin gửi lời tri ân sâu sắc đến giảng viên bộ môn, thầy Nguyễn Quốc Uy, người đã tận tâm truyền đạt những kiến thức quý báu, hướng dẫn trong suốt thời gian học tập vừa qua. Những giá trị này không chỉ giúp ích cho em trong môn học mà còn là hành trang quý báu cho con đường học tập và phát triển sự nghiệp sau này.

Trong thời gian học tập với thầy, em đã có cơ hội thực hiện một đề tài bài tập lớn hoàn chỉnh cho môn học, từ đó hiểu sâu hơn về cách ứng dụng lý thuyết vào thực tế. Đó thực sự là một trải nghiệm quý giá, giúp em tự tin hơn khi đối diện với những thách thức trong tương lai. Em rất biết ơn sự hướng dẫn của thầy trong quá trình thực hiện đề tài này.

Em xin chân thành cảm ơn thầy, chúc thầy luôn mạnh khỏe, tràn đầy nhiệt huyết và tiếp tục đạt được nhiều thành công trong cuộc sống cũng như sự nghiệp giảng dạy. Hy vọng rằng sẽ còn nhiều thế hệ sinh viên được học hỏi và trưởng thành nhờ những bài giảng tâm huyết và kiến thức sâu rộng của thầy.

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU	1
1.1 Đặt vấn đề.....	1
1.2 Mục tiêu và phạm vi đề tài.....	1
1.2.1 Mục tiêu.....	1
1.2.2 Phạm vi đề tài.....	1
1.3 Định hướng giải pháp.....	2
1.3.1 Phân tích yêu cầu	2
1.3.2 Thiết kế hệ thống	2
1.3.3 Phát triển và thử nghiệm	2
1.4 Bố cục bài tập lớn.....	3
CHƯƠNG 2. Giao diện.....	5
2.1 Thiết kế tổng thể.....	5
2.2 Thiết kế chi tiết.....	5
2.2.1 Giao diện Menu	5
2.2.2 Màn hình Dashboard	5
2.2.3 Màn hình Data Sensor	6
2.2.4 Màn hình Action History	7
2.2.5 Màn hình Profile.....	8
CHƯƠNG 3. Chi tiết về đề tài.....	9
3.1 Các thiết bị phần cứng.....	9
3.1.1 ESP32.....	9
3.1.2 Cảm biến nhiệt độ và độ ẩm.....	10
3.1.3 Cảm biến ánh sáng	10
3.1.4 Đèn LED.....	11

3.1.5 Breadboard.....	12
3.1.6 Các thiết bị phụ trợ khác	12
3.1.7 MQTT Broker	13
3.1.8 Frontend.....	14
3.1.9 Backend	16
CHƯƠNG 4. Code.....	18
4.1 Embedded Code	18
4.1.1 Khai báo thư viện và định nghĩa các biến	18
4.1.2 Hàm setup().....	19
4.1.3 Hàm loop().....	20
4.1.4 Hàm callback()	21
4.2 Frontend Code.....	22
4.2.1 Tổng Quan.....	22
4.2.2 Thư Mục public	22
4.2.3 Thư Mục src	23
4.2.4 File App.js.....	24
4.3 Backend Code	25
4.3.1 Tổng quan	25
4.3.2 Lớp Controller	25
4.3.3 Lớp MQTTController.....	26
4.3.4 Lớp Sensor và SensorController	27
4.3.5 Lớp LedFan và LedFanController	28
CHƯƠNG 5. Kết quả thực nghiệm	29
5.1 Giới thiệu.....	29
5.2 Kết quả thực nghiệm phần cứng	29
5.3 Kết quả thực nghiệm Frontend	30

5.4 Kết quả thực nghiệm Backend.....	31
5.5 Lưu trữ và đưa ra dữ liệu cho người dùng	31
5.6 Kết luận	32

DANH MỤC HÌNH VẼ

Hình 2.1	Thiết kế tổng thể	5
Hình 2.2	Giao diện Menu	5
Hình 2.3	Màn hình Dashboard	6
Hình 2.4	Màn hình Data Sensor	7
Hình 2.5	Màn hình Action History	7
Hình 2.6	Màn hình Profile	8
Hình 3.1	Ảnh về module ESP32	9
Hình 3.2	Ảnh về cảm biến DHT11	10
Hình 3.3	Ảnh cảm biến ánh sáng LDR – TH078	11
Hình 3.4	Ảnh đèn LED	11
Hình 3.5	Ảnh về Breadboard	12
Hình 3.6	Ảnh về MQTT Broker	13
Hình 3.7	Ảnh về cách hoạt động của Frontend, Backend và Database	17
Hình 4.1	Khai báo thư viện và định nghĩa các biến	18
Hình 4.2	Kết nối ESP32 với wifi	19
Hình 4.3	Kết nối với MQTT	20
Hình 4.4	Hàm loop và đọc dữ liệu	20
Hình 4.5	Chuyển dữ liệu sang Json và gửi đi	21
Hình 4.6	Xử lý lắng nghe từ MQTT	21
Hình 4.7	Điều khiển bật/tắt đèn	22
Hình 4.8	Tổng quan chính về cấu trúc code Frontend	22
Hình 4.9	Thư mục public	23
Hình 4.10	Thư mục src	24
Hình 4.11	File App.js	25
Hình 4.12	Tổng quan chính về cấu trúc code Backend	25
Hình 4.13	Lớp Controller	26
Hình 4.14	Lớp MQTTController	27
Hình 4.15	Lớp Sensor	27
Hình 4.16	Lớp LedFan	28
Hình 5.1	Kết quả thử nghiệm phần cứng	29
Hình 5.2	Kết quả thử nghiệm Frontend	30

CHƯƠNG 1. GIỚI THIỆU

1.1 Đặt vấn đề

Trong thời đại công nghệ phát triển mạnh mẽ, việc tự động hóa và kết nối các thiết bị điện tử thông qua mạng Internet đã trở thành nhu cầu thiết yếu, đặc biệt là trong các hệ thống nhà thông minh và quản lý môi trường. Các yếu tố môi trường như nhiệt độ, độ ẩm và ánh sáng đóng vai trò quan trọng trong việc duy trì sự thoải mái và an toàn cho con người, đồng thời ảnh hưởng trực tiếp đến hiệu suất hoạt động của nhiều thiết bị điện. Tuy nhiên, việc giám sát và điều khiển các thiết bị điện thủ công gặp nhiều hạn chế, không đảm bảo tính chính xác và phản ứng kịp thời. Vì vậy, việc xây dựng một hệ thống IoT để theo dõi nhiệt độ, độ ẩm, ánh sáng và điều khiển các thiết bị điện từ xa trở nên cần thiết, nhằm tối ưu hóa việc quản lý môi trường, cải thiện hiệu suất sử dụng năng lượng, và mang lại sự tiện lợi cho người sử dụng. Hệ thống này sẽ cung cấp khả năng giám sát thời gian thực, tự động điều chỉnh các thiết bị điện phù hợp với điều kiện môi trường, giúp người dùng tiết kiệm thời gian và đảm bảo môi trường sống luôn ở trạng thái tối ưu.

1.2 Mục tiêu và phạm vi đề tài

1.2.1 Mục tiêu

- Xây dựng một hệ thống IoT hoàn chỉnh có khả năng đo lường và giám sát các chỉ số môi trường như nhiệt độ, độ ẩm, và ánh sáng thông qua giao diện web.
- Cung cấp khả năng điều khiển từ xa các thiết bị điện trong hệ thống nhà thông minh nhằm nâng cao tính tiện lợi và tự động hóa.
- Tạo ra một giao diện người dùng thân thiện trên web, cho phép theo dõi các dữ liệu cảm biến theo thời gian thực và điều khiển thiết bị một cách linh hoạt.
- Tối ưu hóa hiệu suất và tính ổn định của hệ thống, đảm bảo khả năng hoạt động liên tục và đáng tin cậy.

1.2.2 Phạm vi đề tài

- Xây dựng hệ thống đo lường và giám sát các chỉ số môi trường như nhiệt độ, độ ẩm, và ánh sáng thông qua các cảm biến tích hợp.
- Phát triển một nền tảng web để hiển thị dữ liệu từ các cảm biến, với khả năng cập nhật theo thời gian thực và lưu trữ lịch sử dữ liệu để phân tích.
- Thiết kế và triển khai các chức năng điều khiển từ xa cho các thiết bị điện (ví dụ: đèn, quạt, máy điều hòa), thông qua giao diện web.
- Kết nối và truyền dữ liệu từ cảm biến đến hệ thống web sử dụng giao thức

truyền thông (như MQTT)

- Đảm bảo bảo mật cho hệ thống, bao gồm bảo vệ dữ liệu và kiểm soát truy cập người dùng khi điều khiển các thiết bị.

1.3 Định hướng giải pháp

1.3.1 Phân tích yêu cầu

Dựa trên các yêu cầu đã phân tích, giải pháp sẽ tập trung vào việc xây dựng một hệ thống IoT hoàn chỉnh với khả năng giám sát và điều khiển thiết bị điện thông qua giao diện web. Đầu tiên, các cảm biến đo nhiệt độ, độ ẩm, và ánh sáng sẽ được kết nối với một bộ vi điều khiển (như ESP32), giúp thu thập và truyền dữ liệu lên máy chủ. Để truyền dữ liệu một cách ổn định và nhanh chóng, hệ thống sẽ sử dụng giao thức MQTT, đảm bảo thông tin từ cảm biến đến giao diện web là liên tục và theo thời gian thực.

Về phần giao diện người dùng, việc phát triển một nền tảng web thân thiện, trực quan và dễ sử dụng là ưu tiên hàng đầu, giúp người dùng có thể theo dõi các thông số môi trường và điều khiển thiết bị điện từ xa một cách dễ dàng. Ngoài ra, tính bảo mật trong việc quản lý người dùng và truy cập hệ thống sẽ được đảm bảo thông qua các biện pháp mã hóa và xác thực, giúp bảo vệ dữ liệu cá nhân và ngăn chặn truy cập trái phép. Giải pháp cũng sẽ chú trọng vào khả năng mở rộng và tích hợp, giúp hệ thống có thể kết nối thêm nhiều thiết bị mới khi cần thiết, nhằm đáp ứng nhu cầu phát triển của người dùng.

1.3.2 Thiết kế hệ thống

Trong bước thiết kế hệ thống, kiến trúc hệ thống được chia thành ba phần chính: thiết bị IoT, máy chủ xử lý, và giao diện người dùng. Thiết bị IoT bao gồm các cảm biến đo nhiệt độ, độ ẩm, ánh sáng và bộ vi điều khiển (như ESP32). Các cảm biến sẽ liên tục thu thập dữ liệu và truyền về bộ vi điều khiển, từ đó gửi lên máy chủ xử lý thông qua mạng Wi-Fi. Máy chủ xử lý đảm nhiệm vai trò nhận, phân tích và lưu trữ dữ liệu vào cơ sở dữ liệu, đồng thời cung cấp giao tiếp giữa người dùng và thiết bị. Trên nền tảng này, giao diện người dùng được xây dựng dưới dạng một ứng dụng web thân thiện, cho phép người dùng theo dõi các thông số môi trường và điều khiển thiết bị điện từ xa một cách dễ dàng và hiệu quả. Việc thiết kế hệ thống cũng chú trọng đến tính bảo mật và khả năng mở rộng, đảm bảo hệ thống có thể phát triển để tích hợp thêm nhiều cảm biến và thiết bị khi cần thiết.

1.3.3 Phát triển và thử nghiệm

Trong giai đoạn phát triển, hệ thống IoT giám sát nhiệt độ, độ ẩm, ánh sáng và điều khiển thiết bị điện được xây dựng dựa trên các bước thiết kế đã đề ra. Phần

cứng của hệ thống bao gồm các cảm biến và bộ vi điều khiển, được lập trình để thu thập và truyền tải dữ liệu một cách chính xác và liên tục. ESP32 được chọn làm bộ vi điều khiển chính vì tính linh hoạt và khả năng kết nối Wi-Fi mạnh mẽ. Các cảm biến nhiệt độ, độ ẩm và ánh sáng được tích hợp và kết nối với ESP32 để cung cấp dữ liệu môi trường theo thời gian thực. Phần mềm điều khiển ESP32 được lập trình bằng Arduino IDE, đảm bảo rằng dữ liệu được gửi đến máy chủ xử lý mà không bị gián đoạn. Trong quá trình phát triển, các vấn đề như tính ổn định của kết nối và việc xử lý dữ liệu không đồng bộ đã được cân nhắc kỹ lưỡng, đảm bảo rằng hệ thống có thể hoạt động liên tục và phản hồi nhanh chóng trước các thay đổi của môi trường.

Phần mềm máy chủ và giao diện người dùng cũng được phát triển song song để đảm bảo tính tích hợp của toàn bộ hệ thống. Máy chủ xử lý được xây dựng bằng Java, hỗ trợ kết nối dữ liệu từ ESP32 và lưu trữ vào cơ sở dữ liệu MySQL. Máy chủ này cũng đảm nhận vai trò cung cấp API cho giao diện người dùng để truy xuất dữ liệu và gửi các lệnh điều khiển. Giao diện web được phát triển bằng React.js, với các thành phần UI được thiết kế để cung cấp trải nghiệm trực quan và dễ sử dụng. Người dùng có thể theo dõi các chỉ số môi trường qua bảng điều khiển, quan sát dữ liệu lịch sử qua các biểu đồ, và điều khiển thiết bị từ xa chỉ với vài thao tác đơn giản. Sau khi phát triển, hệ thống đã trải qua giai đoạn thử nghiệm kỹ lưỡng, bao gồm các bước kiểm thử đơn vị, kiểm thử tích hợp và kiểm thử hệ thống. Mỗi thành phần phần cứng và phần mềm đều được kiểm tra độc lập trước khi tích hợp vào hệ thống tổng thể. Thử nghiệm trong các tình huống thực tế đã được thực hiện để đảm bảo hệ thống có thể xử lý dữ liệu chính xác và ổn định, ngay cả khi điều kiện môi trường thay đổi liên tục. Hệ thống cũng được thử nghiệm với nhiều thiết bị điện khác nhau để đảm bảo tính tương thích và độ tin cậy cao trong việc điều khiển từ xa.

1.4 Bố cục bài tập lớn

Chương 1: Giới thiệu

Giới thiệu tổng quan về đề tài

Chương 2: Giao diện

Trình bày các giao diện người dùng, cách thiết kế tổng thể hệ thống, sơ đồ khối, và sơ đồ kết nối các thành phần.

Chương 3: Chi tiết về đề tài

Mô tả chi tiết từng thành phần của hệ thống, từ cảm biến, bộ điều khiển đến các thành phần như broker mqtt, backend, frontend, ...

Chương 4: Code

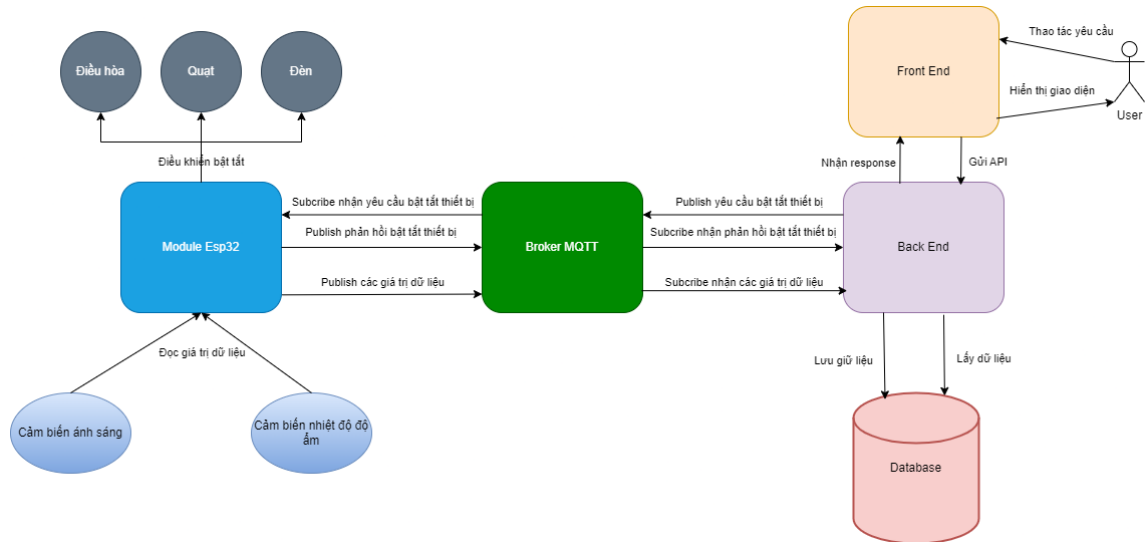
Trình bày mã nguồn của hệ thống, các thư mục và file, và cách thức hoạt động của phần mềm.

Chương 5: Kết quả thực nghiệm

Đánh giá kết quả thử nghiệm hệ thống, so sánh với mục tiêu đề ra và đưa ra nhận xét về tính hiệu quả của hệ thống.

CHƯƠNG 2. Giao diện

2.1 Thiết kế tổng thể

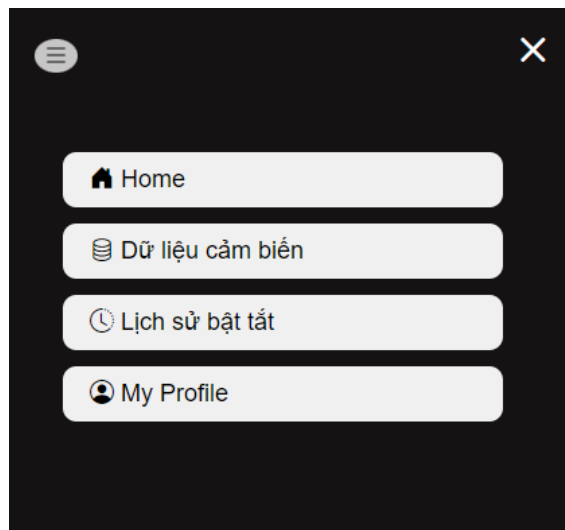


Hình 2.1: Thiết kế tổng thể

2.2 Thiết kế chi tiết

2.2.1 Giao diện Menu

Giao diện Menu sẽ bao gồm các lựa chọn để hiển thị các giao diện như màn hình Dashboard, màn hình Data Sensor, màn hình Action History và màn hình Profile.

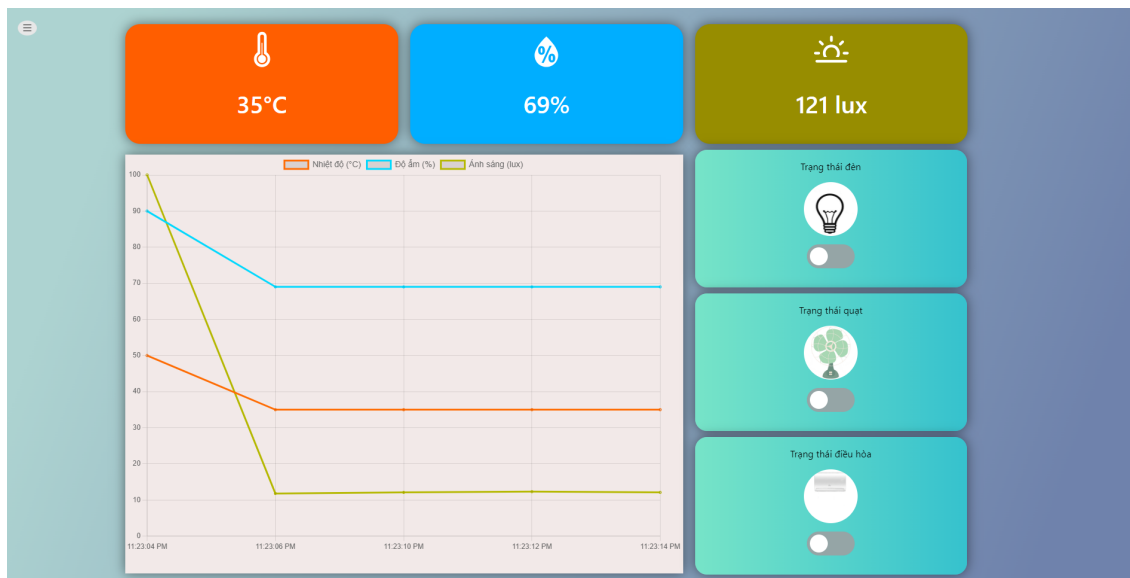


Hình 2.2: Giao diện Menu

2.2.2 Màn hình Dashboard

Màn hình Dashboard cho phép người dùng có thể giám sát các chỉ số môi trường theo thời gian thực. Tại đây, các thông số như nhiệt độ, độ ẩm và ánh sáng được

hiển thị rõ ràng, giúp người dùng dễ dàng quan sát sự biến đổi của chúng. Ngoài ra, màn hình cũng cung cấp các biểu đồ trực quan để mô tả xu hướng và sự thay đổi của các thông số này, hỗ trợ người dùng nắm bắt các biến động quan trọng. Hơn nữa, Dashboard còn tích hợp các nút điều khiển, cho phép bật/tắt các thiết bị điện như quạt, điều hòa và đèn, giúp việc quản lý các thiết bị trở nên nhanh chóng và thuận tiện hơn.



Hình 2.3: Màn hình Dashboard

2.2.3 Màn hình Data Sensor

Màn hình Data Sensor hiển thị tất cả các thông số đã đo được. Người dùng có thể dễ dàng xem và quản lý dữ liệu lịch sử của các cảm biến thông qua màn hình này. Để thuận tiện cho việc điều hướng qua lượng lớn dữ liệu, màn hình hỗ trợ chức năng phân trang, cùng với các công cụ tìm kiếm, giúp người dùng nhanh chóng tìm kiếm thông tin cần thiết.



Hình 2.4: Màn hình Data Sensor

2.2.4 Màn hình Action History

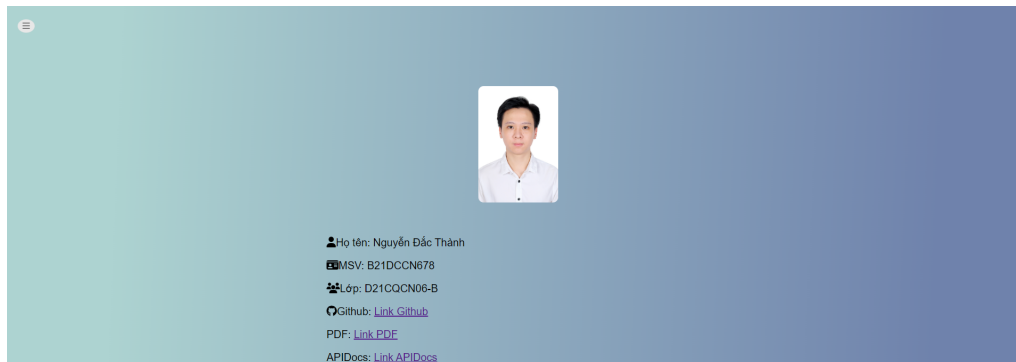
Màn hình Action History được thiết kế để lưu trữ và quản lý lịch sử các thao tác bật/tắt thiết bị điện. Tại đây, người dùng có thể xem lại toàn bộ các hành động đã thực hiện, với tính năng phân trang và tìm kiếm theo thiết bị, giúp dễ dàng theo dõi các sự kiện cụ thể.



Hình 2.5: Màn hình Action History

2.2.5 Màn hình Profile

Màn hình Profile hiển thị thông tin chi tiết của sinh viên đang sử dụng hệ thống. Màn hình này mang đến cái nhìn tổng quan về thông tin cá nhân, bao gồm ảnh đại diện, tên, mã sinh viên, lớp và các thông tin liên quan khác như tài khoản GitHub, báo cáo và apidocs.



Hình 2.6: Màn hình Profile

CHƯƠNG 3. Chi tiết về đề tài

3.1 Các thiết bị phần cứng

3.1.1 ESP32

ESP32 là một vi điều khiển mạnh mẽ và phổ biến do Espressif Systems phát triển, được thiết kế dành cho các ứng dụng IoT (Internet of Things). Được trang bị vi xử lý lõi kép, ESP32 có khả năng xử lý hiệu suất cao, cùng với tích hợp kết nối Wi-Fi và Bluetooth, giúp nó trở thành một giải pháp lý tưởng cho các ứng dụng kết nối không dây. Được hỗ trợ bởi một cộng đồng lớn và nhiều tài liệu hướng dẫn, ESP32 là lựa chọn lý tưởng cho cả người mới bắt đầu và các nhà phát triển chuyên nghiệp trong việc phát triển các dự án nhúng và IoT.

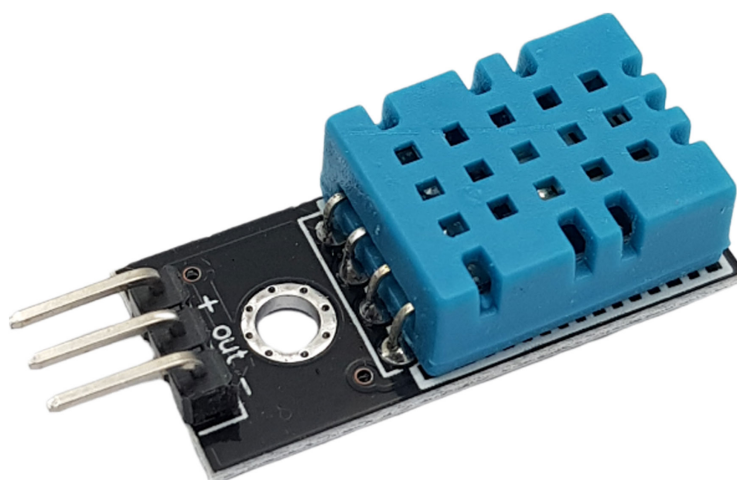
Đây là bộ điều khiển trung tâm trong hệ thống, đảm nhận nhiệm vụ nhận tín hiệu từ các cảm biến và điều khiển các thiết bị điện (như đèn LED) dựa trên các tín hiệu đó. Ngoài ra, ESP32 còn chịu trách nhiệm giao tiếp với server qua giao thức MQTT để gửi dữ liệu thu thập được từ các cảm biến lên hệ thống và nhận lệnh điều khiển từ server để thực hiện các tác vụ điều khiển.



Hình 3.1: Ảnh về module ESP32

3.1.2 Cảm biến nhiệt độ và độ ẩm

DHT11 là một cảm biến đo nhiệt độ và độ ẩm phổ biến, được thiết kế để sử dụng dễ dàng trong các ứng dụng điện tử và IoT. Cảm biến này có kích thước nhỏ gọn và giá thành phải chăng, phù hợp cho các dự án DIY và hệ thống giám sát môi trường. DHT11 cung cấp giá trị nhiệt độ từ 0°C đến 50°C với độ chính xác $\pm 2^{\circ}\text{C}$ và độ ẩm từ 20% đến 90% với độ chính xác $\pm 5\%$. Nó hoạt động với điện áp từ 3 đến 5V và giao tiếp thông qua một giao diện đơn giản 1-wire, giúp việc kết nối với vi điều khiển như Arduino hoặc ESP32 trở nên thuận tiện. Với tính dễ sử dụng và độ ổn định tương đối, DHT11 được lựa chọn rộng rãi để giám sát nhiệt độ và độ ẩm trong các dự án nhà thông minh, hệ thống nông nghiệp tự động, và nhiều ứng dụng đo đạc khác. Cảm biến này có nhiệm vụ gửi dữ liệu thu thập được về ESP32 để xử lý và truyền dữ liệu đó đến server hoặc sử dụng trực tiếp cho việc điều khiển các thiết bị khác.

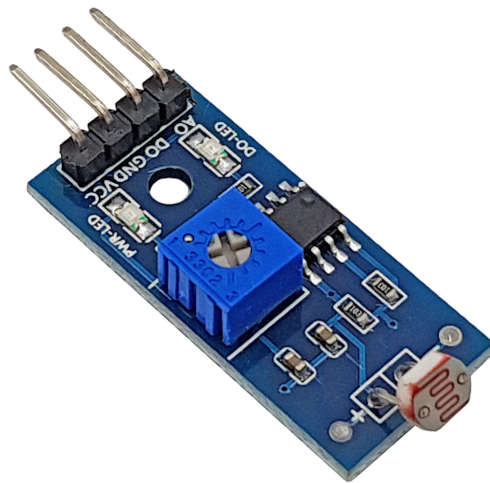


Hình 3.2: Ảnh về cảm biến DHT11

3.1.3 Cảm biến ánh sáng

Cảm biến ánh sáng LDR – TH078 là một loại cảm biến quang điện, được sử dụng rộng rãi để đo cường độ ánh sáng trong các ứng dụng khác nhau như hệ thống nhà thông minh, hệ thống điều khiển tự động và các thiết bị điện tử. LDR (Light Dependent Resistor) là một loại điện trở có giá trị thay đổi tùy thuộc vào mức độ

ánh sáng mà nó nhận được. Cảm biến LDR – TH078 có khả năng phản ứng nhạy bén với sự thay đổi cường độ ánh sáng, giúp xác định chính xác mức độ chiếu sáng của môi trường xung quanh. Với cấu tạo đơn giản, dễ dàng sử dụng và giá thành hợp lý, LDR – TH078 là sự lựa chọn phổ biến cho việc phát triển các hệ thống IoT nhằm tối ưu hóa việc sử dụng ánh sáng, tiết kiệm năng lượng, và tự động hóa các thiết bị chiếu sáng trong các môi trường khác nhau. Thông tin về mức độ ánh sáng sẽ được sử dụng để điều khiển các thiết bị liên quan hoặc gửi về server để lưu trữ và phân tích



Hình 3.3: Ảnh cảm biến ánh sáng LDR – TH078

3.1.4 Đèn LED

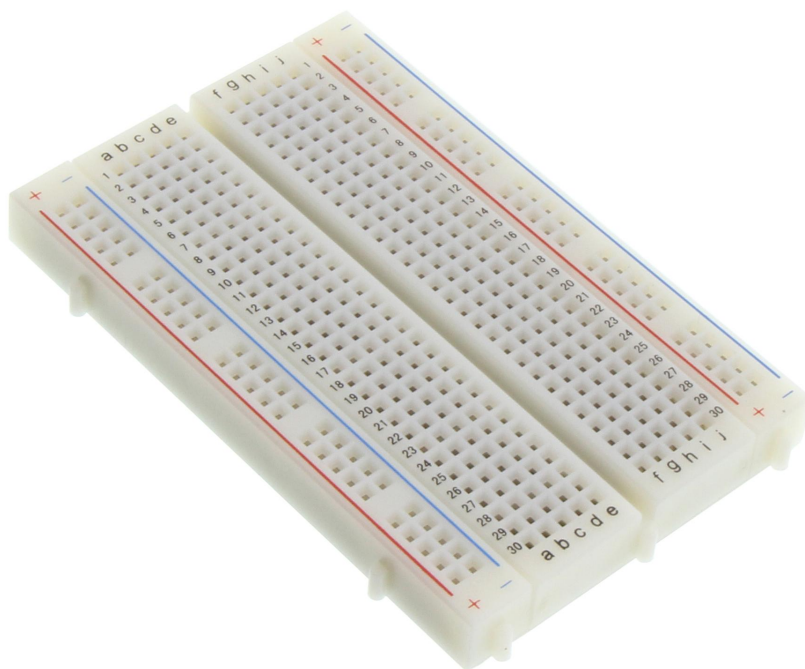
Sử dụng 3 đèn LED đỏ để tượng trưng cho 3 thiết bị điện là đèn chiếu sáng, quạt và điều hòa. ESP32 sẽ điều khiển bật/tắt các đèn LED khi nhận được tín hiệu từ server.



Hình 3.4: Ảnh đèn LED

3.1.5 Breadboard

Breadboard là một bảng mạch thử nghiệm, được sử dụng phổ biến trong lĩnh vực điện tử để xây dựng và thử nghiệm các mạch điện một cách nhanh chóng mà không cần hàn. Breadboard có cấu trúc gồm các hàng lỗ nhỏ được kết nối theo hàng hoặc cột, giúp người dùng dễ dàng cắm và kết nối các linh kiện như điện trở, tụ điện, LED, vi điều khiển, và dây dẫn để tạo ra mạch hoàn chỉnh. Điểm đặc biệt của breadboard là tính linh hoạt và khả năng tái sử dụng, cho phép người dùng thay đổi hoặc sửa đổi mạch điện dễ dàng trong quá trình thử nghiệm mà không gây hư hại cho các linh kiện. Nhờ sự tiện lợi và thiết kế thân thiện, breadboard là một công cụ hữu ích cho những người học tập, nghiên cứu và phát triển các dự án điện tử, từ cấp độ cơ bản đến phức tạp.



Hình 3.5: Ảnh về Breadboard

3.1.6 Các thiết bị phụ trợ khác

- Dây nối: Các dây nối (jumper wires) được sử dụng để kết nối các thành phần khác nhau trên breadboard, đảm bảo việc truyền tín hiệu giữa ESP32, cảm biến và đèn LED.

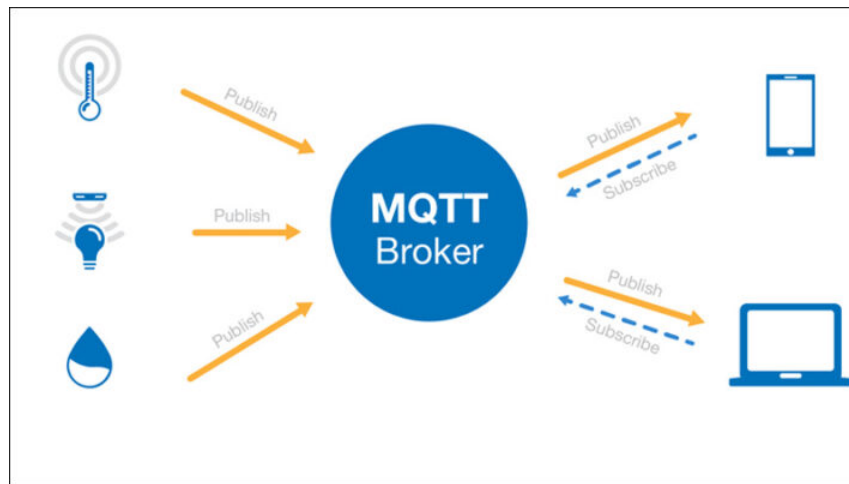
- Điện trở: Điện trở được sử dụng để hạn chế dòng điện trong mạch, đặc biệt là khi kết nối đèn LED để tránh tình trạng cháy hỏng do quá dòng.

Các thành phần phần cứng này kết hợp với nhau tạo thành một hệ thống IoT hoàn chỉnh, cho phép thu thập dữ liệu từ môi trường, xử lý thông tin và điều khiển các thiết bị điện tử từ xa thông qua mạng internet.

3.1.7 MQTT Broker

a, Giới thiệu về MQTT Broker

MQTT (Message Queuing Telemetry Transport) là một giao thức truyền tải tin nhắn nhẹ, được thiết kế để tối ưu hóa việc truyền thông tin giữa các thiết bị IoT qua mạng. MQTT Broker là thành phần trung tâm trong mô hình kiến trúc của MQTT, đảm nhận nhiệm vụ nhận và chuyển tiếp tin nhắn giữa các thiết bị (còn gọi là publisher và subscriber). Broker đảm bảo các tin nhắn được truyền đi một cách an toàn và đến đúng đích, hỗ trợ nhiều mức độ QoS (Quality of Service) khác nhau, từ đó đáp ứng yêu cầu về độ tin cậy của hệ thống IoT.



Hình 3.6: Ảnh về MQTT Broker

b, Vai trò của MQTT Broker trong Hệ thống IoT

Trong hệ thống IoT, MQTT Broker đóng vai trò quan trọng trong việc kết nối và trao đổi dữ liệu giữa các thiết bị. Các cảm biến và bộ điều khiển hoạt động như publisher hoặc subscriber, gửi và nhận dữ liệu thông qua broker. Broker giúp đảm bảo việc truyền tải thông tin diễn ra mượt mà, đồng thời hỗ trợ việc quản lý các kết nối, giúp hệ thống có khả năng mở rộng và duy trì sự ổn định. Với MQTT Broker, các thiết bị IoT có thể truyền thông tin về nhiệt độ, độ ẩm, ánh sáng, và trạng thái thiết bị điện theo thời gian thực mà không cần thiết lập các kết nối điểm-điểm phức tạp.

c, Nguyên lý Hoạt động của MQTT Broker

Nguyên lý hoạt động của MQTT Broker dựa trên mô hình "publish-subscribe". Trong mô hình này, một thiết bị có thể đăng ký (subscribe) hoặc xuất bản (publish) một thông điệp đến một "chủ đề" (topic) cụ thể. MQTT Broker sẽ tiếp nhận các thông điệp từ publisher và gửi chúng đến tất cả các subscriber có đăng ký chủ đề tương ứng. Quá trình này đảm bảo rằng tất cả các thiết bị liên quan đều nhận được thông tin cần thiết, mà không cần thiết lập kết nối trực tiếp với nhau. Broker cũng

quản lý các phiên làm việc của thiết bị, hỗ trợ kết nối lại khi thiết bị bị gián đoạn, từ đó cải thiện độ tin cậy của hệ thống.

d, Cài đặt và Cấu hình MQTT Broker

Để xây dựng một hệ thống IoT sử dụng MQTT, việc cài đặt và cấu hình broker là một bước quan trọng. Có nhiều broker phổ biến mà bạn có thể lựa chọn, chẳng hạn như Mosquitto, HiveMQ, và EMQX. Đối với dự án này, Mosquitto được sử dụng vì nó nhẹ và dễ dàng cài đặt. Quá trình cài đặt Mosquitto bao gồm các bước cơ bản như tải về gói phần mềm, cấu hình tập tin `mosquitto.conf` để thiết lập cổng kết nối, bảo mật, và xác định mức độ QoS.

e, Quản lý Bảo mật cho MQTT Broker

An toàn và bảo mật là một yếu tố không thể thiếu trong bất kỳ hệ thống IoT nào. Đối với MQTT Broker, bảo mật có thể được thực hiện thông qua nhiều cơ chế như xác thực người dùng bằng tài khoản và mật khẩu, hoặc sử dụng ACL (Access Control List) để quản lý quyền truy cập. Việc triển khai bảo mật giúp ngăn chặn các cuộc tấn công và đảm bảo dữ liệu được truyền đi một cách an toàn.

f, Kết nối MQTT Broker với Các Thiết Bị

Khi broker đã được cài đặt và cấu hình, bước tiếp theo là kết nối các thiết bị vào broker. Các thiết bị như cảm biến và bộ điều khiển ESP32 sẽ sử dụng thư viện MQTT để thực hiện kết nối và gửi/nhận dữ liệu từ broker. Thông qua giao thức MQTT, các thiết bị này sẽ xuất bản các chỉ số môi trường (nhiệt độ, độ ẩm, ánh sáng) và nhận lệnh điều khiển từ broker để bật/tắt các thiết bị điện.

g, Kết nối ESP32 tới MQTT Broker

Trong việc triển khai hệ thống IoT, kết nối ESP32 với MQTT Broker là một bước quan trọng giúp thu thập và truyền tải dữ liệu từ các cảm biến. ESP32 có thể được lập trình thông qua Arduino IDE để thực hiện chức năng này một cách hiệu quả. Ví dụ, khi cảm biến nhiệt độ và độ ẩm (như DHT11) thu thập dữ liệu, ESP32 sẽ sử dụng lệnh `publish` để gửi thông tin lên các chủ đề cụ thể trên MQTT Broker.

3.1.8 Frontend

Frontend của hệ thống IoT được phát triển dựa trên ReactJS, một thư viện JavaScript nổi bật chuyên về xây dựng giao diện người dùng (UI). Với khả năng tạo ra các ứng dụng web năng động và phản hồi nhanh, ReactJS trở thành lựa chọn lý tưởng để hiển thị dữ liệu theo thời gian thực từ các thiết bị IoT. Giao diện người dùng được thiết kế để đảm bảo tính thân thiện, giúp người dùng dễ dàng theo dõi các chỉ số môi trường, quản lý thiết bị và kiểm tra lịch sử hoạt động thông qua nhiều trang giao diện khác nhau.

Hệ thống giao diện chính bao gồm bốn trang chức năng, mỗi trang phục vụ một mục đích cụ thể:

Dashboard:

Chức năng: Trang Dashboard cung cấp cái nhìn tổng quát về toàn bộ hệ thống, hiển thị các thông số như nhiệt độ, độ ẩm và ánh sáng theo thời gian thực. Trang này cũng cung cấp thông tin về trạng thái hiện tại của các thiết bị như quạt, đèn và điều hòa, kèm theo các biểu đồ thể hiện sự biến động của các thông số này.

Tương tác: Người dùng có thể thực hiện các thao tác bật/tắt thiết bị trực tiếp trên Dashboard và nhận phản hồi ngay lập tức về các hành động của mình.

Data Sensor:

Chức năng: Trang Data Sensor trình bày chi tiết dữ liệu thu thập từ các cảm biến như nhiệt độ, độ ẩm và ánh sáng. Dữ liệu được hiển thị dưới dạng bảng, cho phép người dùng theo dõi các xu hướng và biến động theo thời gian.

Tương tác: Người dùng có thể thực hiện tìm kiếm dựa trên các thông số cụ thể và xóa dữ liệu.

History Action:

Chức năng: Trang History Action ghi lại và hiển thị lịch sử các hành động điều khiển thiết bị, như việc bật/tắt quạt, đèn và điều hòa. Người dùng có thể theo dõi những thay đổi đã được thực hiện và thời gian của các hành động đó.

Tương tác: Người dùng có thể lọc để xem lịch sử bật/tắt của mỗi thiết bị và xóa dữ liệu.

Profile:

Chức năng: Trang Profile cung cấp thông tin cá nhân của người dùng, bao gồm các chi tiết về tài khoản, tùy chọn cài đặt và khả năng quản lý thông tin cá nhân.

Frontend đảm nhận vai trò giao tiếp với backend để xử lý và hiển thị dữ liệu cho người dùng. Các chức năng chính bao gồm:

Gửi request cho backend:

- Frontend gửi các request HTTP tới backend để lấy dữ liệu thời gian thực từ các cảm biến, dữ liệu lịch sử (history action), và thông tin từ trang Data Sensor.
- Ngoài ra, frontend cũng gửi các yêu cầu bật/tắt thiết bị từ giao diện người dùng đến backend để điều khiển phần cứng.

Nhận và hiển thị response:

- Sau khi nhận được phản hồi từ backend, frontend cập nhật giao diện người dùng với các thông tin mới nhất, như giá trị cảm biến hiện tại, trạng thái thiết bị, và thông tin lịch sử hành động.

- Dữ liệu được hiển thị một cách trực quan và dễ hiểu, giúp người dùng có thể theo dõi và điều khiển hệ thống một cách hiệu quả.

3.1.9 Backend

Backend của hệ thống IoT được xây dựng bằng ngôn ngữ lập trình Java, một trong những ngôn ngữ phổ biến và mạnh mẽ nhất hiện nay trong phát triển ứng dụng web. Với khả năng xử lý mạnh mẽ và tính ổn định cao, Java cho phép xây dựng các ứng dụng backend có khả năng mở rộng tốt, đảm bảo hiệu suất và bảo mật cho dữ liệu người dùng.

Backend chịu trách nhiệm xử lý tất cả các yêu cầu từ frontend, bao gồm việc thu thập và lưu trữ dữ liệu từ các cảm biến IoT, xử lý thông tin và gửi phản hồi về cho người dùng. Sử dụng các framework nổi tiếng như Spring Boot, hệ thống backend đảm bảo dễ dàng tích hợp với các công nghệ khác và cung cấp các dịch vụ cần thiết cho việc quản lý thiết bị IoT.

Hệ thống backend cũng tích hợp với cơ sở dữ liệu để lưu trữ dữ liệu cảm biến, lịch sử hành động của người dùng và thông tin cá nhân. Điều này giúp người dùng dễ dàng truy cập và quản lý thông tin của mình một cách hiệu quả. Từ việc gửi yêu cầu đến việc nhận dữ liệu, toàn bộ quá trình được tối ưu hóa để đảm bảo tốc độ và độ tin cậy, đáp ứng nhu cầu ngày càng cao trong quản lý và giám sát các hệ thống IoT hiện đại.

Backend thực hiện các nhiệm vụ chính như sau:

Trao đổi dữ liệu với phần cứng thông qua broker MQTT:

- Vai trò là client: Trong hệ thống MQTT, backend hoạt động như một client, kết nối với broker MQTT để gửi và nhận các thông điệp. Từ đây, backend nhận các thông số từ các thiết bị phần cứng như cảm biến nhiệt độ, độ ẩm và ánh sáng. Nó cũng có khả năng gửi các yêu cầu điều khiển từ frontend, chẳng hạn như bật hoặc tắt quạt, đèn và điều hòa đến phần cứng.

- Xử lý dữ liệu: Sau khi nhận được dữ liệu từ phần cứng, backend sẽ thực hiện xử lý và có thể lưu trữ vào cơ sở dữ liệu hoặc phản hồi lại cho frontend.

Giao tiếp với cơ sở dữ liệu:

- Lưu trữ dữ liệu: Backend thực hiện kết nối với SQL để lưu trữ thông tin từ các cảm biến và trạng thái của thiết bị. Điều này giúp duy trì một bản ghi liên tục về

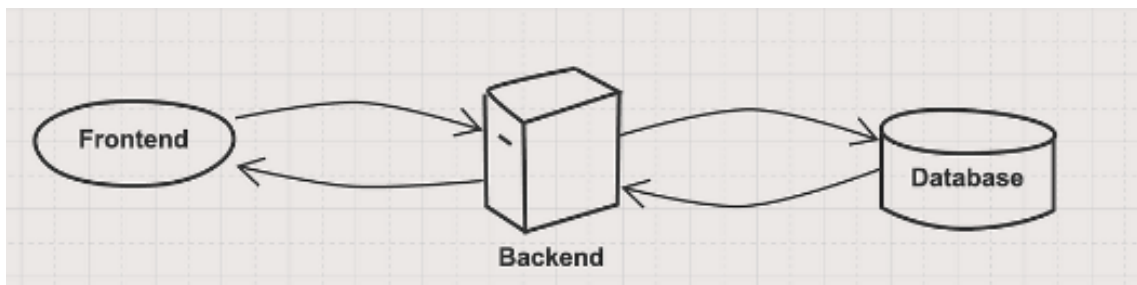
các giá trị môi trường cũng như lịch sử điều khiển thiết bị, bao gồm cả dữ liệu về việc bật và tắt các thiết bị.

- Truy xuất dữ liệu: Ngoài ra, backend cũng hỗ trợ truy xuất dữ liệu từ SQL theo yêu cầu từ frontend, ví dụ như lấy các thông số đã lưu trong một khoảng thời gian nhất định để phục vụ cho việc phân tích hoặc hiển thị dưới dạng biểu đồ.

Nhận yêu cầu từ frontend và trả về phản hồi tương ứng:

- Xử lý yêu cầu HTTP: Backend tiếp nhận các yêu cầu từ frontend, như yêu cầu lấy dữ liệu cảm biến, yêu cầu điều khiển thiết bị hoặc truy vấn lịch sử dữ liệu.

- Gửi phản hồi: Dựa trên các yêu cầu nhận được, backend sẽ xử lý và gửi phản hồi tương ứng trở lại cho frontend, đảm bảo rằng người dùng có thể theo dõi các chỉ số môi trường hoặc điều khiển thiết bị một cách chính xác và kịp thời.



Hình 3.7: Ảnh về cách hoạt động của Frontend, Backend và Database

CHƯƠNG 4. Code

4.1 Embedded Code

4.1.1 Khai báo thư viện và định nghĩa các biến

- Khai báo các thư viện cần thiết cho WiFi, MQTT, cảm biến DHT, và sử dụng JSON để xử lý dữ liệu.
- Khai báo các chân GPIO: bao gồm chân kết nối cảm biến DHT11, chân đọc tín hiệu từ cảm biến ánh sáng LDR, và các chân điều khiển các thiết bị như quạt, điều hòa, và đèn.
- Khai báo thông tin mạng WiFi, bao gồm SSID và mật khẩu để kết nối ESP32 với mạng.
- Định nghĩa địa chỉ IP và cổng của broker MQTT, cùng với thông tin đăng nhập, và khai báo các chủ đề MQTT để trao đổi thông tin như điều khiển các thiết bị điện và gửi dữ liệu cảm biến.
- Khởi tạo đối tượng WiFiClient để kết nối WiFi và đối tượng PubSubClient để xử lý kết nối MQTT, giúp ESP32 có thể giao tiếp với MQTT Broker.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"
#include <ArduinoJson.h>

#define dhtpin 25
#define dhthtype DHT11
DHT dht(dhtpin, dhthtype);

#define LDR_PIN 32 // Chân ADC để đọc tín hiệu từ LDR

#define FAN_PIN 4
#define AC_PIN 5
#define LIGHT_PIN 2

// Thông tin mạng WiFi
const char* ssid = "TiGii";
const char* password = "12345689";

// MQTT Server
const char *mqtt_broker = "192.168.43.105"; // Phong 192.168.1.244
const int mqtt_port = 1993;
const char* mqtt_user = "thanh"; // Username MQTT
const char* mqtt_pass = "678"; // Password MQTT
const char* topicIotWeather = "iot/weather";
const char* mqtt_status_topic = "home/device/status";
const char* TOPIC_CONTROL_LED = "control/led";
const char* TOPIC_CONTROL_FAN = "control/fan";
const char* TOPIC_CONTROL_AIR = "control/air";

WiFiClient espClient;
PubSubClient client(espClient);
```

Hình 4.1: Khai báo thư viện và định nghĩa các biến

4.1.2 Hàm setup()

a, Kết nối Wifi

Đầu tiên, `WiFi.begin(ssid, password)` được sử dụng để bắt đầu kết nối với mạng WiFi bằng SSID và mật khẩu đã được cung cấp. Vòng lặp `while (WiFi.status() != WL_CONNECTED)` kiểm tra trạng thái kết nối và tiếp tục lặp lại cho đến khi kết nối thành công, mỗi lần lặp sẽ đợi 1 giây và in ra thông báo "Dang ket noi den WiFi...". Khi kết nối thành công, thông báo "Da ket noi WiFi" sẽ được in ra.

```
void setup() {
  Serial.begin(9600);
  dht.begin();

  // Cấu hình chân xuất ra cho 3 thiết bị
  pinMode(FAN_PIN, OUTPUT);
  pinMode(AC_PIN, OUTPUT);
  pinMode(LIGHT_PIN, OUTPUT);

  // Kết nối WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Dang ket noi den WiFi...");
  }
  Serial.println("Da ket noi WiFi");
}
```

Hình 4.2: Kết nối ESP32 với wifi

b, Kết nối MQTT

Đầu tiên, `client.setServer(mqtt_broker, mqtt_port)` thiết lập địa chỉ và cổng của MQTT Broker. Hàm `client.setCallback(callback)` được sử dụng để định nghĩa hàm xử lý khi có thông điệp MQTT đến.

Vòng lặp `while (!client.connected())` kiểm tra kết nối và tiếp tục thử kết nối cho đến khi thành công. Một ID client được tạo từ địa chỉ MAC của ESP32 (`client_id`) để xác định thiết bị khi kết nối với MQTT Broker. Khi kết nối thành công (`client.connect(client_id.c_str(), mqtt_user, mqtt_pass)`), thông báo "Public EMQX MQTT broker connected" được in ra. Nếu kết nối thất bại, trạng thái lỗi được in và chương trình đợi 2 giây trước khi thử lại.

Sau khi kết nối thành công, ESP32 đăng ký (subscribe) vào các chủ đề MQTT liên quan đến việc điều khiển LED, quạt và điều hòa, để nhận lệnh từ MQTT Broker và thực hiện điều khiển các thiết bị này.

```
// Kết nối đến MQTT Broker
client.setServer(mqtt_broker, mqtt_port);
client.setCallback(callback);
while (!client.connected()) {
    String client_id = "esp32-client-";
    client_id += String(WiFi.macAddress());
    Serial.printf("The client %s connects to the public MQTT broker\n", client_id.c_str());

    if (client.connect(client_id.c_str(), mqtt_user, mqtt_pass)) {
        Serial.println("Public EMQX MQTT broker connected");
        // isConnected = true;
    } else {
        Serial.print("Failed with state ");
        Serial.print(client.state());
        delay(2000);
    }
}
client.subscribe(TOPIC_CONTROL_LED);
client.subscribe(TOPIC_CONTROL_FAN);
client.subscribe(TOPIC_CONTROL_AIR);
```

Hình 4.3: Kết nối với MQTT

4.1.3 Hàm loop()

- Lắng nghe các tin nhắn từ MQTT
- Mỗi 2 giây thực hiện đọc nhiệt độ, độ ẩm, ánh sáng

```
unsigned long previousMillis = 0; // Biến lưu thời gian trước đó
const long interval = 2000; // Khoảng thời gian giữa các lần gửi dữ liệu (2 giây)

void loop() {
    client.loop(); // Lắng nghe các tin nhắn từ MQTT

    // Lấy thời gian hiện tại
    unsigned long currentMillis = millis();

    // Kiểm tra nếu đã đủ 2 giây kể từ lần gửi dữ liệu trước
    if (currentMillis - previousMillis >= interval) {
        // Cập nhật thời gian của lần gửi dữ liệu mới
        previousMillis = currentMillis;

        // Đọc nhiệt độ, độ ẩm, ánh sáng
        float humidity = dht.readHumidity();
        float temperature = dht.readTemperature();
        int sensorValue = analogRead(LDR_PIN);
        float voltage = sensorValue * (3.3 / 4095.0); // Chuyển đổi giá trị ADC sang điện áp (0 - 3.3V)
        float lux = (2500 / voltage - 500) / 3.3; // Công thức chuyển đổi gần đúng từ điện áp sang lux
        if (lux > 200000) {
            lux = 200000;
        }
        float light = lux;
    }
}
```

Hình 4.4: Hàm loop và đọc dữ liệu

- Chuyển sang Json và gửi đi (publish)

```
// Chuyển dữ liệu sang chuỗi để publish qua JSON
String temperature_payload = String(int(temperature));
String humidity_payload = String(int(humidity));
String light_payload = String(int(light));

// Thêm dữ liệu vào JSON
StaticJsonDocument<200> weatherJson;
weatherJson["temperature"] = temperature_payload;
weatherJson["humidity"] = humidity_payload;
weatherJson["light"] = light_payload;

// Chuyển đổi JSON thành chuỗi và gửi lên MQTT
char weatherJsonStr[200];
serializeJson(weatherJson, weatherJsonStr);
Serial.println(weatherJsonStr);
client.publish(topicIotWeather, weatherJsonStr);
}

// client.loop() sẽ được chạy liên tục để lắng nghe các tin nhắn từ MQTT
}
```

Hình 4.5: Chuyển dữ liệu sang Json và gửi đi

4.1.4 Hàm callback()

- Xử lý lắng nghe từ MQTT

```
// Hàm xử lý lệnh từ MQTT
void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("topic: ");
    Serial.print(topic);
    String message;
    for (int i = 0; i < length; i++) {
        message += (char)payload[i];
    }
    Serial.print(" Messenger: " );
    Serial.println(message);
}
```

Hình 4.6: Xử lý lắng nghe từ MQTT

- Khi nhận được các topic "control/led", "control/fan", "control/air" thì lần lượt xử lý bật/tắt các đèn led.
- Sau khi bật/tắt xong đèn thì gửi thông điệp đến MQTT để Backend nhận, lưu trữ vào database và gửi lại cho Frontend.

```

if (String(topic) == TOPIC_CONTROL_LED) {
  // Điều khiển đèn từ MQTT
  if (message == "On") {
    digitalWrite(LIGHT_PIN, HIGH); // Bật đèn
    client.publish(mqtt_status_topic, "light_on_success");
  } else if (message == "Off") {
    digitalWrite(LIGHT_PIN, LOW); // Tắt đèn
    client.publish(mqtt_status_topic, "light_off_success");
  }
}
else if (String(topic) == TOPIC_CONTROL_FAN) {
  // Điều khiển quạt từ MQTT
  if (message == "On") {
    digitalWrite(FAN_PIN, HIGH); // Bật quạt
    client.publish(mqtt_status_topic, "fan_on_success");
  } else if (message == "Off") {
    digitalWrite(FAN_PIN, LOW); // Tắt quạt
    client.publish(mqtt_status_topic, "fan_off_success");
  }
}
else if (String(topic) == TOPIC_CONTROL_AIR) {
  // Điều khiển điều hòa từ MQTT
  if (message == "On") {
    digitalWrite(AC_PIN, HIGH); // Bật điều hòa
    client.publish(mqtt_status_topic, "air_on_success");
  } else if (message == "Off") {
    digitalWrite(AC_PIN, LOW); // Tắt điều hòa
    client.publish(mqtt_status_topic, "air_ff_success");
  }
}
}

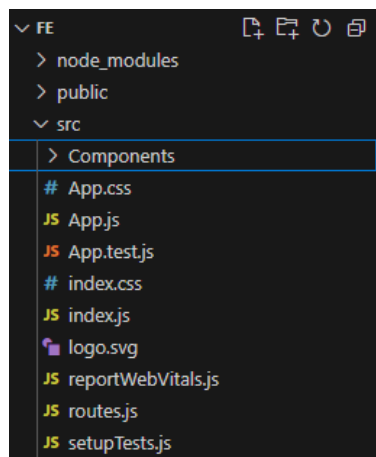
```

Hình 4.7: Điều khiển bật/tắt đèn

4.2 Frontend Code

4.2.1 Tổng Quan

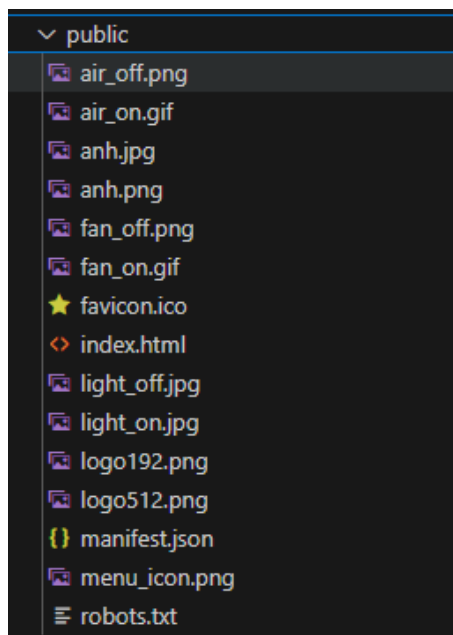
Dưới đây là tổng quan chính về cấu trúc code Frontend:



Hình 4.8: Tổng quan chính về cấu trúc code Frontend

4.2.2 Thư Mục public

Mục đích: Thư mục public lưu giữ các tệp tĩnh như hình ảnh, biểu tượng, và tệp HTML chính (index.html). Đây là nơi chứa các tài nguyên cần thiết mà không cần thay đổi khi ứng dụng hoạt động.



Hình 4.9: Thư mục public

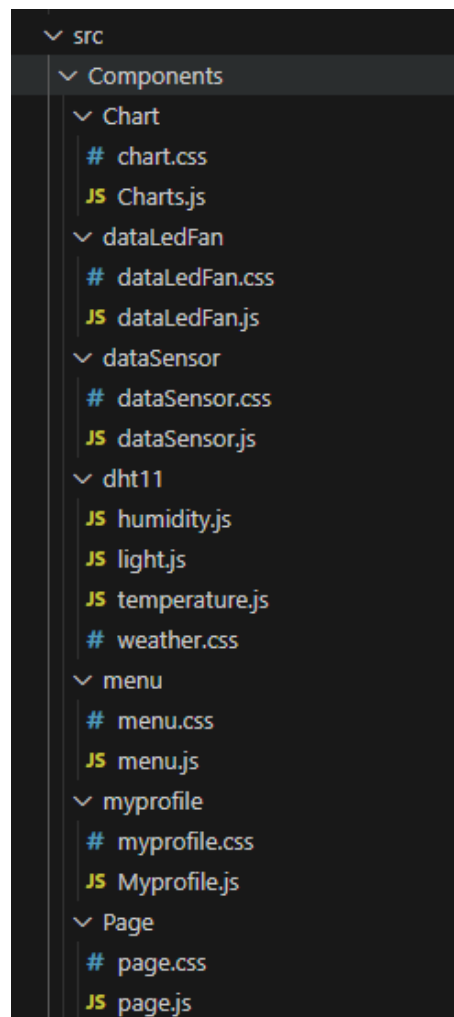
4.2.3 Thư Mục src

Mục đích: Toàn bộ mã nguồn chính của ứng dụng nằm trong thư mục src. Thư mục này là nơi chứa các thành phần và logic cốt lõi của frontend.

Thư Mục components:

Nội dung: Chứa các component chính như Page, Profile, DataLedFan, và DataSensor, cùng với các component phụ như Menu, Chart (dùng để hiển thị đồ thị),... Ngoài ra còn chứa các file css để debug dạng giao diện của từng component như page.css, dataSensor.css,...

Vai trò: Mỗi component là một phần của giao diện người dùng và chứa logic liên quan đến phần đó. Ví dụ, Page để hiển thị thông tin tổng quan, Profile để quản lý thông tin người dùng, DataLedFan lưu trữ lịch sử các thao tác, và DataSensor để hiển thị dữ liệu từ cảm biến.



Hình 4.10: Thư mục src

4.2.4 File App.js

Vai trò: App.js là tệp chính của ứng dụng React, nơi tập hợp tất cả các thành phần khác để tạo nên một ứng dụng hoàn chỉnh.

Cấu trúc:

Thành phần App trong React, được dùng để xây dựng giao diện chính của ứng dụng. Đầu tiên, App import các thành phần và thư viện cần thiết, bao gồm cả file CSS để định dạng giao diện, các thành phần từ react-router-dom để điều hướng trong ứng dụng, và một danh sách các route từ file routes.

App là một hàm (gọi là Function Component) trả về một đoạn JS, trong đó chứa một thẻ div và thành phần Routes. Thành phần Routes này chứa danh sách các Route, và mỗi Route đại diện cho một đường dẫn (URL) cụ thể của ứng dụng.

Sử dụng routes.map() để lặp qua danh sách các route đã được định nghĩa trong file routes. Mỗi route chứa các thông tin như đường dẫn (path) và thành phần cần hiển thị (main). Các thông tin này được sử dụng để tạo ra các Route tương ứng.

Khi người dùng truy cập vào một đường dẫn cụ thể trong trình duyệt, thành phần Route sẽ hiển thị nội dung phù hợp, được xác định bởi đường dẫn đó. Toàn bộ phần điều hướng này giúp ứng dụng có thể chuyển đổi giữa các trang một cách linh hoạt mà không cần phải tải lại toàn bộ trang web.

Cuối cùng, App được xuất ra (export default App) để có thể sử dụng ở những nơi khác trong ứng dụng, như một phần của giao diện chính.

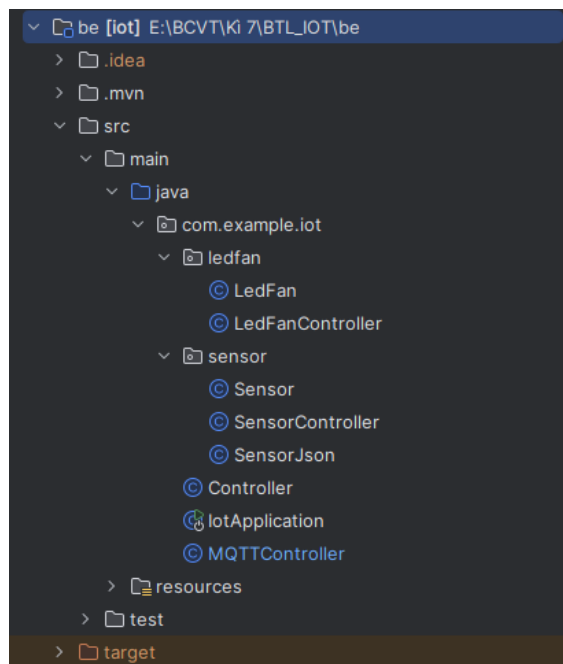
```
function App() {
  return (
    <div>
      <Routes>
        {
          routes.map((item) => {
            return (
              <Route key={item.id} path={item.path} element={item.main()} />
            )
          })
        }
      </Routes>
    </div>
  )
}
```

Hình 4.11: File App.js

4.3 Backend Code

4.3.1 Tổng quan

Dưới đây là tổng quan chính về cấu trúc code Backend:



Hình 4.12: Tổng quan chính về cấu trúc code Backend

4.3.2 Lớp Controller

Lớp này chứa các thông tin để kết nối đến cơ sở dữ liệu

Biến connection là một đối tượng của lớp Connection từ thư viện java.sql, giúp quản lý kết nối đến cơ sở dữ liệu. Biến này được khai báo là static, nghĩa là chỉ có một kết nối duy nhất được tạo ra và dùng chung cho tất cả các đối tượng của lớp.


Trong hàm khởi tạo Controller(), nếu kết nối chưa được tạo (connection == null), chương trình sẽ cố gắng tạo kết nối bằng cách:

Nạp driver MySQL (Class.forName("com.mysql.cj.jdbc.Driver")).

Sử dụng DriverManager.getConnection() để thiết lập kết nối với thông tin đã cung cấp. Nếu thành công, nó sẽ in ra "Kết nối thành công".

Nếu có lỗi xảy ra (lỗi class không tồn tại hoặc lỗi SQL), nó sẽ in ra "Kết nối thất bại" và in chi tiết lỗi (e.printStackTrace()).

Phương thức closeConnection() được dùng để đóng kết nối với cơ sở dữ liệu nếu kết nối đã tồn tại, giúp giải phóng tài nguyên.



```

public Controller() {
    if (connection == null) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            connection = DriverManager.getConnection(DB_URL + SCHEMA, DB_USERNAME, DB_PASSWORD);
            System.out.println("Ket noi thanh cong");
        } catch (ClassNotFoundException | SQLException e) {
            System.out.println("Ket noi that bai");
            e.printStackTrace();
        }
    }
}

```

Hình 4.13: Lớp Controller

4.3.3 Lớp MQTTController

Lớp này được dùng để kết nối với MQTTBroker và xử lý publish và subscribe

Khởi tạo và kết nối với broker MQTT:

Phương thức Init() khởi tạo một MQTT client để kết nối với broker, sử dụng địa chỉ và cổng đã được chỉ định (tcp://192.168.43.105:1993).

Sau khi kết nối thành công, chương trình đăng ký lắng nghe (subscribe) một topic có tên iot/weather. Khi có thông báo mới từ topic này, dữ liệu sẽ được lưu vào weatherJson và được thêm vào cơ sở dữ liệu qua SensorController.addData(weatherJson).

Điều khiển các thiết bị (đèn, quạt, điều hòa):

Các phương thức SetLightOn(), SetFanOn(), và SetAirOn() được sử dụng để điều khiển đèn, quạt, và điều hòa.

Mỗi phương thức này đều tạo một thông điệp (MqttMessage) với nội dung "On" hoặc "Off" tùy thuộc vào trạng thái thiết bị (lightStatus, fanStatus, airStatus).

Sau đó, thông điệp được gửi đến các topic tương ứng để điều khiển thiết bị thông

qua MQTT.

```
public static void Init() { 1 usage  ThanhND *
    String broker = "tcp://192.168.43.105:1993"; // Địa chỉ broker MQTT
    String clientId = "JavaClient";
    String weatherTopic = "iot/weather";

    try {
        // Tạo client
        MqttClient client = new MqttClient(broker, clientId);
        MqttConnectOptions options = new MqttConnectOptions();
        options.setCleanSession(true);
        client.connect(options);
        System.out.println("Connected to MQTT broker.");

        // Đăng ký lắng nghe weatherTopic
        client.subscribe(weatherTopic, (receivedTopic, msg) -> {
            String message = new String(msg.getPayload());
            weatherJson = message;
            SensorController.addSensorData(weatherJson);
        });
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

Hình 4.14: Lớp MQTTController

4.3.4 Lớp Sensor và SensorController

Lớp Sensor chứa các thuộc tính.

```
public Sensor(int id, String temperature, String humidity, String light, Timestamp date) {
    this.id = id;
    this.temperature = temperature;
    this.humidity = humidity;
    this.light = light;
    this.date = date;
}
```

Hình 4.15: Lớp Sensor

Các phần chính trong lớp SensorController bao gồm:

API lấy tất cả dữ liệu cảm biến (/sensors):

Phương thức `getAllSensors()` truy vấn tất cả dữ liệu từ bảng sensor trong cơ sở dữ liệu, chuyển đổi kết quả thành danh sách các đối tượng `Sensor`, và trả về danh sách này.

Dữ liệu được đảo ngược (sắp xếp từ mới đến cũ) trước khi trả về.

Thêm dữ liệu cảm biến (`addSensorData()`):

Phương thức tĩnh `addSensorData()` nhận dữ liệu cảm biến dạng chuỗi JSON, phân tích và lấy ra các giá trị như nhiệt độ, độ ẩm và ánh sáng.

Sau đó, tạo một đối tượng Sensor và thêm vào cơ sở dữ liệu bằng câu truy vấn INSERT.

API tìm kiếm dữ liệu cảm biến (/search-sensor):

Phương thức `getSensor()` cho phép tìm kiếm dữ liệu cảm biến dựa trên các tham số như `temperature`, `humidity`, và `light`.

Xóa dữ liệu cảm biến (/clear-sensor):

Phương thức `clearSensor()` xóa tất cả dữ liệu trong bảng `sensor`.

API lấy dữ liệu cảm biến từ MQTT (/get-sensor):

Phương thức `getSensor()` trả về chuỗi JSON chứa dữ liệu thời tiết, được lấy từ `MQTTController`.

4.3.5 Lớp `LedFan` và `LedFanController`

Lớp `LedFan` chứa các thuộc tính và lớp `LedFanController` chứa các hàm api để giao tiếp với frontend và lưu trữ, truy xuất dữ liệu từ database và gọi đến các hàm điều khiển thiết bị ở `MQTTController`.

```
public LedFan(int id, String name, String status, Timestamp date) {  
    this.id = id;  
    this.name = name;  
    this.status = status;  
    this.date = date;  
}
```

Hình 4.16: Lớp `LedFan`

Lớp `LedFanController` quản lý các thiết bị quạt và đèn LED bằng cách sử dụng các REST API, kết nối đến cơ sở dữ liệu, và giao tiếp MQTT. Có các phương thức chính như:

`getAllLedFans()`: Lấy toàn bộ danh sách thiết bị LED fan từ cơ sở dữ liệu.

`searchLedFans()`: Tìm kiếm các thiết bị theo tên.

`addLedFan()`: Thêm một thiết bị LED fan vào cơ sở dữ liệu.

`deleteAll()`: Xóa toàn bộ dữ liệu thiết bị từ bảng `ledFan`.

`setLightLed()`, `setFanLed()`, `setAirLed()`: Điều khiển bật/tắt đèn, quạt, và điều hòa thông qua MQTT và thêm thông tin vào cơ sở dữ liệu.

CHƯƠNG 5. Kết quả thực nghiệm

5.1 Giới thiệu

Trình bày kết quả thực nghiệm của hệ thống IoT giám sát nhiệt độ, độ ẩm, ánh sáng và điều khiển các thiết bị điện. Các kết quả bao gồm việc kiểm tra khả năng thu thập dữ liệu từ cảm biến, kết nối và giao tiếp MQTT, cùng với khả năng tương tác của người dùng qua giao diện frontend.

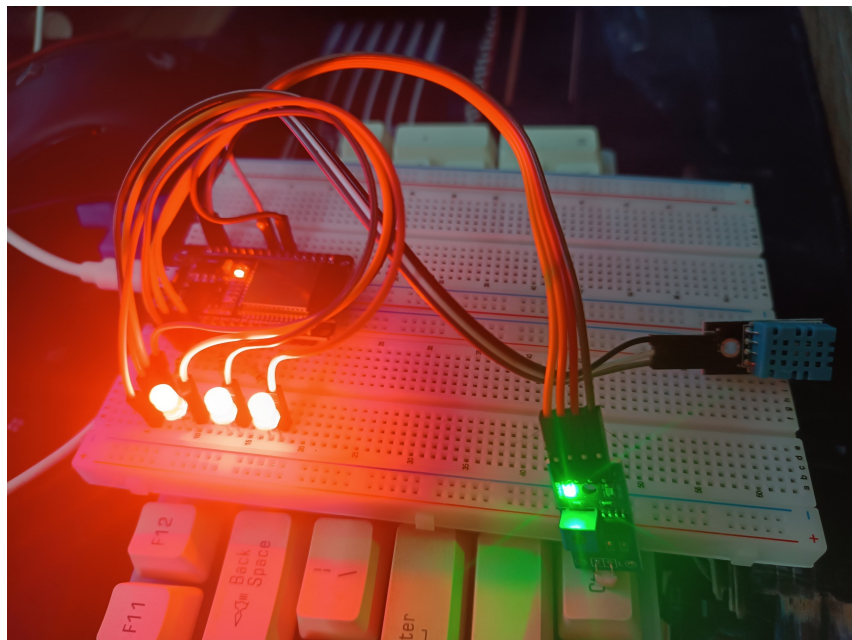
5.2 Kết quả thực nghiệm phần cứng

Trong quá trình phát triển hệ thống IoT, thực nghiệm phần cứng đã được thực hiện với kết quả như sau:

Lắp đặt mạch: Mạch điện được lắp đặt thành công trên breadboard với các linh kiện như ESP32, cảm biến DHT11, cảm biến LDR và các đèn LED đại diện cho quạt, điều hòa, đèn.

Hoạt động của cảm biến: Cảm biến DHT11 và LDR đã hoạt động ổn định, cung cấp dữ liệu nhiệt độ, độ ẩm và cường độ ánh sáng chính xác.

Điều khiển thiết bị: Các thiết bị điều khiển đã được bật/tắt thành công thông qua lệnh từ giao diện người dùng, cho thấy khả năng điều khiển từ xa hiệu quả.



Hình 5.1: Kết quả thử nghiệm phần cứng

Kiểm tra kết nối: Kết nối WiFi và MQTT broker đã được thực hiện thành công, cho phép gửi và nhận thông điệp mà không gặp sự cố.

Thử nghiệm toàn diện: Hệ thống hoạt động đồng bộ và hiệu quả trong việc thu

thập dữ liệu và điều khiển thiết bị điện.

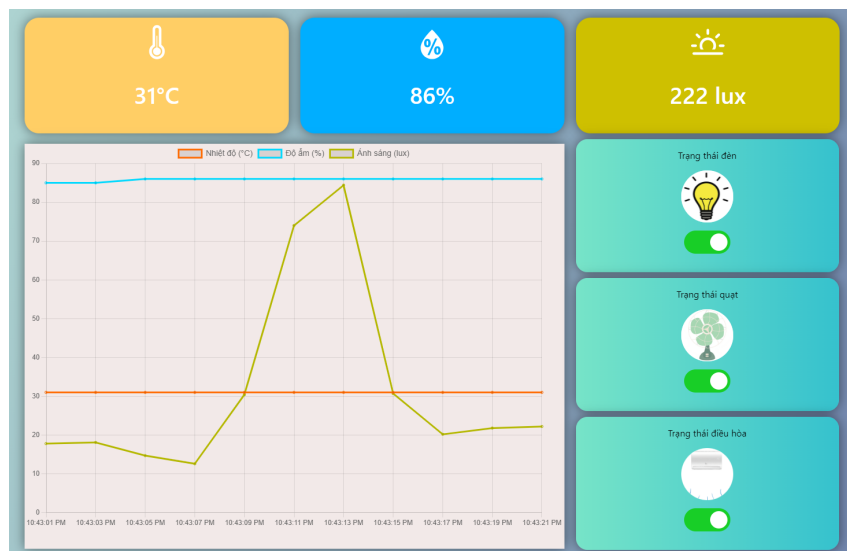
Kết quả thực nghiệm phần cứng khẳng định tính khả thi và hiệu suất của hệ thống IoT, tạo nền tảng cho các giai đoạn phát triển tiếp theo.

5.3 Kết quả thực nghiệm Frontend

Kết quả thực nghiệm frontend của hệ thống IoT được thực hiện với các bước và kết quả như sau:

Xây dựng giao diện người dùng: Giao diện được phát triển bằng ReactJS đã hoàn thành và hoạt động trơn tru, với cấu trúc rõ ràng và dễ sử dụng.

Hiển thị dữ liệu: Các thông số môi trường như nhiệt độ, độ ẩm, và ánh sáng được hiển thị chính xác và cập nhật theo thời gian thực, cho phép người dùng dễ dàng theo dõi.



Hình 5.2: Kết quả thử nghiệm Frontend

Tính năng điều khiển thiết bị: Các nút điều khiển trên dashboard cho phép bật/tắt quạt, điều hòa, và đèn hoạt động hiệu quả. Người dùng nhận được phản hồi ngay lập tức khi thực hiện các thao tác.

Tương tác người dùng: Các trang giao diện như Dashboard, Data Sensor, Action History, và Profile đều hoạt động mượt mà. Người dùng có thể dễ dàng chuyển đổi giữa các trang và thực hiện các thao tác mà không gặp vấn đề gì.

Kiểm tra tính năng tìm kiếm: Tính năng tìm kiếm dữ liệu trong trang Data Sensor hoạt động tốt, cho phép người dùng truy cập nhanh chóng vào thông tin cần thiết.

Kết quả thực nghiệm frontend khẳng định rằng hệ thống đáp ứng tốt các yêu cầu thiết kế và tính năng, mang lại trải nghiệm người dùng tích cực và hiệu quả.

5.4 Kết quả thực nghiệm Backend

Kết quả thực nghiệm backend của hệ thống IoT được thực hiện với các bước và kết quả như sau:

Kết nối với MQTT Broker: Backend đã thành công trong việc thiết lập kết nối với broker MQTT, cho phép nhận và gửi thông điệp từ các cảm biến và thiết bị điều khiển.

Xử lý dữ liệu từ cảm biến: Backend có khả năng nhận dữ liệu từ cảm biến như nhiệt độ, độ ẩm và ánh sáng. Các thông số này được xử lý và lưu trữ vào cơ sở dữ liệu MongoDB một cách chính xác.

Giao tiếp với cơ sở dữ liệu: Backend thực hiện thành công các thao tác lưu trữ và truy xuất dữ liệu từ MongoDB. Điều này giúp duy trì bản ghi liên tục của các thông số môi trường cũng như lịch sử điều khiển thiết bị.

Phản hồi yêu cầu từ frontend: Backend có khả năng xử lý các yêu cầu từ frontend, bao gồm yêu cầu lấy dữ liệu cảm biến và điều khiển thiết bị. Phản hồi từ backend được gửi đến frontend một cách nhanh chóng, đảm bảo tính kịp thời cho người dùng.

Quản lý trạng thái thiết bị: Backend đã tích hợp thành công tính năng theo dõi trạng thái của các thiết bị (quạt, điều hòa, đèn) và phản hồi trạng thái này khi có thay đổi.

Kết quả thực nghiệm backend cho thấy hệ thống hoạt động ổn định và hiệu quả, đáp ứng đầy đủ các yêu cầu của ứng dụng IoT, đồng thời đảm bảo việc quản lý và điều phối dữ liệu giữa phần cứng và giao diện người dùng.

5.5 Lưu trữ và đưa ra dữ liệu cho người dùng

Trong quá trình thực nghiệm, hệ thống đã thực hiện thành công việc lưu trữ và cung cấp dữ liệu cho người dùng với các kết quả như sau:

Lưu trữ dữ liệu từ cảm biến: Dữ liệu từ các cảm biến (nhiệt độ, độ ẩm, ánh sáng) được thu thập và lưu trữ vào cơ sở dữ liệu SQL một cách liên tục và chính xác. Mỗi thông số được ghi lại theo thời gian thực, tạo thành một bản ghi hoàn chỉnh cho người dùng.

Truy xuất dữ liệu: Người dùng có thể truy xuất dữ liệu một cách nhanh chóng. Hệ thống cung cấp các tùy chọn lọc và sắp xếp để người dùng dễ dàng tìm kiếm thông tin theo loại cảm biến hoặc giá trị đo lường.

Hiển thị dữ liệu thời gian thực: Dữ liệu được hiển thị trên giao diện người dùng một cách trực quan và dễ hiểu. Các biểu đồ và bảng điều khiển giúp người dùng

theo dõi các thông số môi trường theo thời gian thực và nhận diện nhanh chóng các biến đổi quan trọng.

Phản hồi chính xác: Hệ thống đảm bảo rằng mọi yêu cầu từ người dùng đều nhận được phản hồi chính xác và kịp thời, tạo sự thuận lợi trong việc theo dõi và quản lý các thông số môi trường.

Kết quả thực nghiệm cho thấy hệ thống đã đáp ứng hiệu quả nhu cầu lưu trữ và cung cấp dữ liệu cho người dùng, đồng thời nâng cao trải nghiệm người dùng trong việc theo dõi và điều khiển các thiết bị IoT.

5.6 Kết luận

Qua quá trình thử nghiệm, đề tài đã đạt được những kết quả khả quan trong việc phát triển hệ thống IoT tích hợp cảm biến và điều khiển thiết bị. Các kết quả chính bao gồm:

Hoạt động ổn định của phần cứng: Mạch điện đã được lắp ráp hoàn chỉnh và các thành phần như cảm biến nhiệt độ, độ ẩm, ánh sáng, và thiết bị điều khiển (quạt, đèn, điều hòa) hoạt động hiệu quả, cung cấp dữ liệu chính xác và kịp thời.

Giao diện người dùng thân thiện: Frontend được phát triển bằng ReactJS đã cho phép người dùng dễ dàng theo dõi các thông số môi trường và quản lý thiết bị. Các trang giao diện được thiết kế trực quan, giúp người dùng nhanh chóng nắm bắt thông tin cần thiết.

Xử lý và lưu trữ dữ liệu hiệu quả: Backend, được xây dựng với Java và sử dụng SQL, đã cho thấy khả năng lưu trữ và truy xuất dữ liệu tốt. Dữ liệu từ cảm biến được lưu trữ liên tục và có thể truy xuất nhanh chóng, đáp ứng nhu cầu phân tích và theo dõi của người dùng.

Tích hợp thành công giữa các thành phần: Hệ thống đã tích hợp thành công giữa phần cứng, backend, và frontend, cho phép người dùng điều khiển thiết bị từ xa và theo dõi thông số môi trường một cách hiệu quả.

Tóm lại, kết quả thử nghiệm của đề tài đã chứng minh tính khả thi và hiệu quả của hệ thống IoT trong việc giám sát và điều khiển các thiết bị, đồng thời mở ra hướng phát triển cho các ứng dụng IoT trong tương lai.