

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



BÁO CÁO BÀI TẬP CUỐI KÌ

ĐỀ TÀI: THIẾT KẾ NHÀ THÔNG MINH CÓ KHOÁ CỬA SỬ
DỤNG NFC, HỆ THỐNG CHIẾU SÁNG, ÂM THANH TRONG
PHÒNG CÓ KẾT NỐI TỪ XA SỬ DỤNG WEBSERVER RIÊNG
VÀ ĐIỀU KHIỂN HỒNG NGOẠI

Giảng viên hướng dẫn: Nguyễn Tài Tuyên
Môn học: IoT và ứng dụng
Nhóm lớp: Nhóm 08

Thành viên nhóm:

Trần Việt Dũng	B21DCCN032
Dương Thế Anh	B21DCCN144

Hà Nội - 2024

LỜI MỞ ĐẦU

Ngày nay khoa học công nghệ phát triển nhu cầu của con người ngày càng cao. Việc nghiên cứu khoa học ngày càng được đầu tư để đáp ứng nhu cầu đó, các ngành công nghệ kỹ thuật điện tử đã có sự phát triển vượt bậc đưa khoa học vào kỷ nguyên mới. Kỹ thuật vi xử lý vi điều khiển là một ứng dụng lớn của khoa học kỹ thuật vào cuộc sống phục vụ trực tiếp con người.

Nhà thông minh là ngôi nhà được trang bị các hệ thống tự động thông minh cùng với cách bố trí hợp lý, các hệ thống này có khả năng tự điều phối các hoạt động trong ngôi nhà theo thói quen sinh hoạt và nhu cầu cá nhân của gia chủ. Giải pháp nhà thông minh sẽ biến những món đồ điện tử bình thường trong ngôi nhà trở nên thông minh và gần gũi với người dùng hơn, chúng được kiểm soát thông qua các thiết bị truyền thông như điều khiển từ xa, điện thoại di động... ngôi nhà thông minh đơn giản nhất có thể được hình dung bao gồm một mạng điều khiển liên kết một số lượng cố định các thiết bị điện, điện tử gia dụng trong ngôi nhà và chúng được điều khiển thông qua một chiếc điều khiển từ xa.

Ở đề tài này, nhóm chúng em thực hiện “Thiết kế nhà thông minh có khoá cửa sử dụng NFC kết hợp hệ thống chiếu sáng, âm thanh trong phòng có kết nối từ xa sử dụng webserver riêng và điều khiển hồng ngoại”. Đây cũng là một trong những đề tài rất sát với thực tế, mang tính ứng dụng thực tiễn rất cao. Điều đó càng tạo động lực và cảm hứng cho sinh viên tìm tòi và nghiên cứu.

Trong báo cáo chắc hẳn còn nhiều sai sót, chúng em rất mong nhận được sự chỉ bảo, hướng dẫn của thầy để đồ án hoàn thiện hơn.

Chúng em chân thành cảm ơn!

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU CHUNG	7
1.1. Giới thiệu tổng quan về IoT.....	7
1.2. Đặt vấn đề và mục tiêu nghiên cứu.....	8
1.3. Giới thiệu chung hệ thống.....	8
1.4. Thiết bị phần cứng và nền tảng phần mềm.....	9
1.4.1. Các thiết bị phần cứng	9
1.4.2. Các nền tảng phần mềm.....	23
CHƯƠNG 2. THIẾT KẾ HỆ THỐNG	29
2.1. Lập bảng dữ liệu	29
2.2. Thiết kế sơ đồ mạch điện	31
2.3. Lưu đồ thuật toán	33
2.4. Đặc tả cấp độ IoT.....	35
2.5. Thiết kế ứng dụng web.....	35
2.6. Thêm mục các thư viện đã sử dụng và nêu ý nghĩa của từng thư viện.	37
2.7. Code chương trình.....	39
2.7.1. Khai báo thư viện, hằng số và khởi tạo:	39
2.7.2. Cấu hình các thiết lập cho thiết bị khi ESP32 khởi động:	41
2.7.3. Hàm loop() - vòng lặp chính:.....	42
2.7.4. Các hàm quản lý người dùng	45
2.7.5. Hàm đọc thẻ người dùng.....	46
2.7.6. Hàm điều khiển đèn và điều hoà.....	49
2.7.7. Hàm httpSync() và updateToPin(String s)	51
2.7.8. Các hàm trong code server Flask Python	52
CHƯƠNG 3. KẾT QUẢ THỰC NGHIỆM	61
3.1. Kiểm tra.....	61
3.2. Kết quả đạt được	63
3.3. Hướng phát triển	63

3.4. Kết luận	64
---------------------	----

DANH MỤC CÁC HÌNH ẢNH

Hình 1. ESP32	9
Hình 2. Động cơ	13
Hình 3. Màn hình OLED	15
Hình 4. Module DHT11	16
Hình 5. Module RFID MFRC-522 NFC 13.56MHz	17
Hình 6. Module cảm biến ánh sáng	18
Hình 7. Breadboard	19
Hình 8. Còi thụ động	20
Hình 9. LED	21
Hình 10. Module thu hồng ngoại + Remote IR1838	22
Hình 11. Cửa sổ chương trình Arduino IDE	23
Hình 12. Giao diện thiết đặt thiết bị IoT	24
Hình 13. Giao diện thiết kế giao diện web app	25
Hình 14. Sơ đồ mạch điện in	31
Hình 15. Sơ đồ mạch điện thực tế	32
Hình 16. Lưu đồ thuật toán	34
Hình 17. Đặc tả cấp độ IoT	35
Hình 18. Giao diện web Blynk	36
Hình 19. Login page của web tự host	36
Hình 20. Trang DashBoard web tự host	37
Hình 21: HttpSync()	51
Hình 22: Hàm Update to pin	51
Hình 23: Cơ chế hoạt động của WebServer	52
Hình 24. Giao diện app Blynk	61
Hình 25. Giao diện web tự host	62

DANH MỤC CÁC BẢNG

Bảng 1. Bảng dữ liệu thiết kế hệ thống.....	30
Bảng 2. Dữ liệu phản hồi của hệ thống.....	63

CHƯƠNG 1. GIỚI THIỆU CHUNG

1.1. Giới thiệu tổng quan về IoT

Lần đầu tiên cụm từ “cách mạng công nghiệp 4.0” được đưa ra trong một báo cáo của chính phủ Đức năm 2013. Nó có ảnh hưởng rất lớn đối với tất cả quốc gia trên thế giới trong đó có Việt Nam chúng ta. Cách mạng công nghiệp 4.0 mang đến cho nước ta cả cơ hội lẫn cả thách thức và rủi ro. Vậy cách mạng công nghệ 4.0 là gì? Tại sao lại quan trọng đến vậy?

Klaus Schwab, người sáng lập và chủ tịch điều hành Diễn đàn Kinh tế Thế Giới đã mang đến cái nhìn đơn giản hơn về Cách mạng Công nghiệp 4.0 như sau:

"Cách mạng công nghiệp đầu tiên sử dụng năng lượng nước và hơi nước để cơ giới hóa sản xuất. Cuộc cách mạng lần 2 diễn ra nhờ ứng dụng điện năng để sản xuất hàng loạt. Cuộc cách mạng lần 3 sử dụng điện tử và công nghệ thông tin để tự động hóa sản xuất. Bây giờ, cuộc Cách mạng Công nghiệp Thứ tư đang nảy nở cuộc cách mạng lần ba, nó kết hợp các công nghệ lại với nhau, làm mờ ranh giới giữa vật lý, kỹ thuật số và sinh học".

Công nghiệp 4.0 sẽ diễn ra trên 3 lĩnh vực chính gồm Công nghệ sinh học, Kỹ thuật số và Vật lý. Những yếu tố cốt lõi của Kỹ thuật số trong CMCN 4.0 sẽ là: Trí tuệ nhân tạo (AI), Vạn vật kết nối - Internet of Things (IoT) và dữ liệu lớn (Big Data).

Internet of Things (IoT) hay cụ thể hơn là Mạng lưới vạn vật kết nối Internet hoặc là Mạng lưới thiết bị kết nối Internet là một liên mạng, trong đó các thiết bị kết nối với nhau. Việc kết nối thì có thể thực hiện qua Wifi, mạng viễn thông băng rộng (3G, 4G), Bluetooth, ZigBee, hồng ngoại... Các thiết bị có thể là điện thoại thông minh, máy pha cafe, máy giặt, tai nghe, bóng đèn, và nhiều thiết bị khác. Cisco, nhà cung cấp giải pháp và thiết bị mạng hàng đầu hiện nay dự báo: Đến năm 2020, sẽ có khoảng 50 tỷ đồ vật kết nối vào Internet, thậm chí con số này còn gia tăng nhiều hơn nữa. IoT sẽ là mạng khổng lồ kết nối tất cả mọi thứ, bao gồm cả con người và sẽ tồn tại các mối quan hệ giữa người và người, người và thiết bị, thiết bị và thiết bị. Một mạng lưới IoT có thể chứa đến 50 đến 100 nghìn tỉ đối tượng được kết nối và mạng lưới này có thể theo dõi sự di chuyển của từng đối tượng. Một con người sống trong thành thị có thể bị bao bọc xung quanh bởi 1000 đến 5000 đối tượng có khả năng theo dõi.

1.2. Đặt vấn đề và mục tiêu nghiên cứu

Đặt vấn đề nghiên cứu:

Ngày nay khoa học công nghệ phát triển nhu cầu của con người ngày càng cao. Việc nghiên cứu khoa học ngày càng được đầu tư để đáp ứng nhu cầu đó, các ngành công nghệ kỹ thuật điện tử đã có sự phát triển vượt bậc đưa khoa học vào kỷ nguyên mới. Kỹ thuật vi xử lý vi điều khiển là một ứng dụng lớn của khoa học kỹ thuật vào cuộc sống phục vụ trực tiếp con người.

Nhà thông minh được hiểu đơn giản là ngôi nhà được lắp đặt các trang thiết bị điện, điện tử chúng có thể được điều khiển bằng các phương pháp tự động hoặc bán tự động. Với nhà thông minh, chủ nhân ngôi nhà có thể điều khiển các thiết bị từ xa mà không cần có mặt trong ngôi nhà. Ngoài ra, nhà thông minh còn tự động theo dõi các thay đổi bất thường như trộm, cháy nổ... để giúp ngôi nhà an toàn hơn.

Mục tiêu nghiên cứu của đề tài

Đề tài đặt ra giải quyết vấn đề hiểu về các bước cơ bản lập trình Arduino, ESP32 hiển thị lên màn hình LCD, đọc dữ liệu từ các cảm biến và điều khiển các thiết bị ngoại vi.

Đề tài được đưa ra với các mục tiêu cụ thể:

Điều khiển các thiết bị điện trong ngôi nhà từ xa thông qua internet

Giám sát nhiệt độ, độ ẩm trong phòng thông qua cảm biến

Đối tượng sử dụng:

Các gia đình hiện đại mong muốn có một không gian sống tiện nghi, an toàn và tự động hóa, đặc biệt là những gia đình trẻ yêu thích công nghệ, mong muốn sự tiện lợi trong sinh hoạt hàng ngày.

1.3. Giới thiệu chung hệ thống

Trong bối cảnh khoa học công nghệ phát triển không ngừng, nhu cầu về sự tiện nghi và an toàn trong không gian sống ngày càng trở nên thiết yếu. Nhà thông minh (Smart Home) là một giải pháp công nghệ nổi bật, mang đến khả năng tự động hóa và tối ưu hóa cuộc sống của con người thông qua các thiết bị hiện đại được kết nối với nhau và điều khiển từ xa. Đề tài "Thiết kế nhà thông minh có khóa cửa sử dụng NFC kết hợp hệ thống chiếu sáng và âm thanh trong phòng có kết nối từ xa" hướng đến việc xây dựng một hệ thống nhà thông minh với những tính năng vượt trội, phù hợp với nhu cầu của người dùng hiện đại.

1.4. Thiết bị phần cứng và nền tảng phần mềm

1.4.1.1. ESP WROOM 32 (30P)

Sử dụng bo mạch ESP32 DEVKIT V1 DOIT phiên bản 30 chân GPIOs.



- Module trung tâm: Wifi BLE Soc ESP32, Dual core
- Vi xử lí: Xtensa lõi kép 32-bit LX6, tốc độ hoạt động là 240MHz
- Bộ nhớ (SRAM): 512 KB

- Kết nối không dây: Chuẩn Wifi 802. 11 b/g/n ; Bluetooth (v4.2 BR/EDR và BLE)
- Tích hợp Led Status, nút BOOT và UNABLE
- Tích hợp mạch và giao tiếp UART
- Các ngoại vi:
 - + 18 kênh ADC 12-bit, chuyển đổi tín hiệu tương tự thành tín hiệu số.
 - + 2 kênh DAC 8-bit, chuyển đổi tín hiệu số thành tương tự.
 - + 10 chân có cảm ứng chạm.
 - + Chuẩn giao tiếp không đồng bộ: SPI, I2C, I2S.
 - + Chuẩn giao tiếp không đồng bộ: UART.
 - + PWM: hỗ trợ điều khiển thiết bị như động cơ, đèn LED.
 - + CAN bus 2.0

Các chân ESP32:

- VIN: Chân nguồn cấp điện trực tiếp cho bo mạch khi không dùng qua USB, có thể cung cấp 5V đến 12V.
- GND: Chân nối đất, nối với các thiết bị ngoại vi.
- 3V3: Chân cấp nguồn cho các cảm biến hoặc ngoại vi, có thể cung cấp 3.3V.
- EN: Chân kích hoạt (Enable) để bật nguồn hoặc chuyển sang chế độ tiêu thụ ít năng lượng (bật/tắt ESP32).
- BOOT: Chân để xác định chế độ khởi động, được sử dụng trong quá trình lập trình.
- GPIO 34 - 39: Các chân Input Only nhận tín hiệu đầu vào từ thiết bị ngoài, đo lường tín hiệu điện áp tương tự.
- USB: Cổng USB được sử dụng để kết nối board với máy tính để lập trình và cung cấp nguồn.

Nguyên lý hoạt động:

- Vi xử lý ESP32 là trung tâm, đảm nhận xử lý tín hiệu và điều khiển các thiết bị ngoại vi.
- Kết nối không dây qua Wi-Fi hoặc Bluetooth để giao tiếp với các thiết bị hoặc đám mây.
- Giao tiếp ngoại vi qua GPIO và các giao thức I2C, SPI, UART để thu thập dữ liệu từ cảm biến hoặc điều khiển các thiết bị.
- Quản lý điện năng hiệu quả với các chế độ tiết kiệm năng lượng, đặc biệt cho các ứng dụng IoT chạy bằng pin.

Cách giao tiếp với cảm biến:

ESP32 có thể giao tiếp với các cảm biến thông qua các chuẩn giao thức như I2C, SPI, UART hoặc thông qua các chân analog/digital.

a. Giao tiếp Digital (Tín hiệu số):

Giao tiếp đơn giản nhất: Sử dụng tín hiệu số (HIGH hoặc LOW, tương ứng với 1 và 0) để truyền tải thông tin, như phát hiện chuyển động hoặc trạng thái ON/OFF.

b. Giao tiếp Analog (Tín hiệu tương tự):

Cảm biến analog xuất ra tín hiệu điện áp tương tự, thay đổi liên tục dựa trên giá trị đo được (như nhiệt độ, độ ẩm). ESP32 có thể đọc giá trị này qua các chân ADC (Analog to Digital Converter), chuyển đổi tín hiệu analog thành giá trị số mà ESP32 có thể xử lý.

c. Giao thức I2C:

Còn gọi là TWI (Two Wire Interface), là một phương thức truyền thông nối tiếp đồng bộ, thường được sử dụng để kết nối vi điều khiển với các cảm biến, EEPROM và các thiết bị ngoại vi khác. I2C rất phổ biến trong các ứng dụng nhúng do tính đơn giản và khả năng kết nối nhiều thiết bị với chỉ hai dây.

Các Đặc Điểm Của Giao Thức I2C:

- Sử Dụng Hai Dây: I2C chỉ cần hai dây để truyền thông:
 - + SDA (Serial Data Line): Dữ liệu.
 - + SCL (Serial Clock Line): Tín hiệu đồng hồ.
- Địa Chỉ Hệ Thống: Mỗi thiết bị trong mạng I2C có một địa chỉ duy nhất. Điều này cho phép nhiều thiết bị kết nối trên cùng một đường dây.
- Master-Slave: Trong giao thức I2C, có một thiết bị chính (master) và một hoặc nhiều thiết bị phụ (slave). Master sẽ điều khiển việc truyền dữ liệu.

d. Giao thức SPI:

Là một phương thức truyền thông nối tiếp đồng bộ, rất hữu ích cho việc giao tiếp giữa vi điều khiển (như ESP32) và các thiết bị ngoại vi, như cảm biến, bộ nhớ flash, hoặc màn hình. SPI cho phép truyền dữ liệu nhanh chóng và thường được sử dụng trong các ứng dụng yêu cầu tốc độ cao.

- Các Đặc Điểm Của Giao Thức SPI:

- Đồng bộ: SPI sử dụng một tín hiệu xung đồng hồ để đồng bộ hóa truyền dữ liệu.
- Nhiều dây: Thường yêu cầu bốn dây:
 - + MOSI (Master Out Slave In): Dữ liệu từ Master tới Slave.
 - + MISO (Master In Slave Out): Dữ liệu từ Slave tới Master.
 - + SCK (Serial Clock): Tín hiệu đồng hồ do Master phát ra.
 - + SS/CS (Slave Select/Chip Select): Tín hiệu để chọn Slave nào đang hoạt động.

e. Giao thức UART:

Là một phương thức truyền thông nối tiếp rất phổ biến được sử dụng để giao tiếp giữa vi điều khiển (như ESP32) và các thiết bị khác như cảm biến, module GPS, hoặc thậm chí các thiết bị khác thông qua cổng nối tiếp.

Các Đặc Điểm Của Giao Thức UART:

- Đơn giản: Chỉ cần hai dây (RX và TX) để truyền và nhận dữ liệu.
- Không đồng bộ: Không cần tín hiệu xung đồng hồ.
- Tốc độ: Tốc độ truyền dữ liệu có thể cấu hình, thường được đo bằng baud rate (ví dụ: 9600 bps, 115200 bps).

Cách giao tiếp với ThingSpeak, Weather API, Web Server:

Giao tiếp giữa ESP32 và các dịch vụ như ThingSpeak, Weather API, hoặc server riêng sử dụng các giao thức HTTP, MQTT, WebSocket

HTTP là một giao thức truyền thông dựa trên mô hình client-server. Trong đó, ESP32 đóng vai trò là client và gửi yêu cầu (request) đến server. Server phản hồi (response) lại với dữ liệu hoặc kết quả của yêu cầu đó.

Quy trình giao tiếp:

- Kết nối WiFi: Đầu tiên, ESP sẽ cần kết nối với mạng WiFi để có thể truy cập Internet.
- Gửi yêu cầu HTTP
- Nhận phản hồi
- Xử lý và hiển thị

MQTT là giao thức nhẹ và hiệu quả dùng trong các hệ thống IoT (Internet of Things). Sử dụng mô hình publisher/subscriber (nhà phát hành/nhà đăng ký) để gửi và nhận tin nhắn qua máy chủ trung gian (broker). Thường dùng cho các thiết bị có tài nguyên hạn chế và cần truyền thông tin nhỏ như cảm biến và thiết bị IoT

WebSocket cho phép giao tiếp hai chiều giữa máy chủ và máy khách. Không giống như HTTP, WebSocket giữ kết nối mở, cho phép trao đổi dữ liệu theo thời gian thực mà không cần gửi nhiều yêu cầu HTTP. Thường được sử dụng trong các ứng dụng chat, trò chơi trực tuyến, và các ứng dụng thời gian thực.

1.4.1.2. Module Động Cơ RC Servo MG996R



Hình 2. Động cơ

Thông số kỹ thuật:

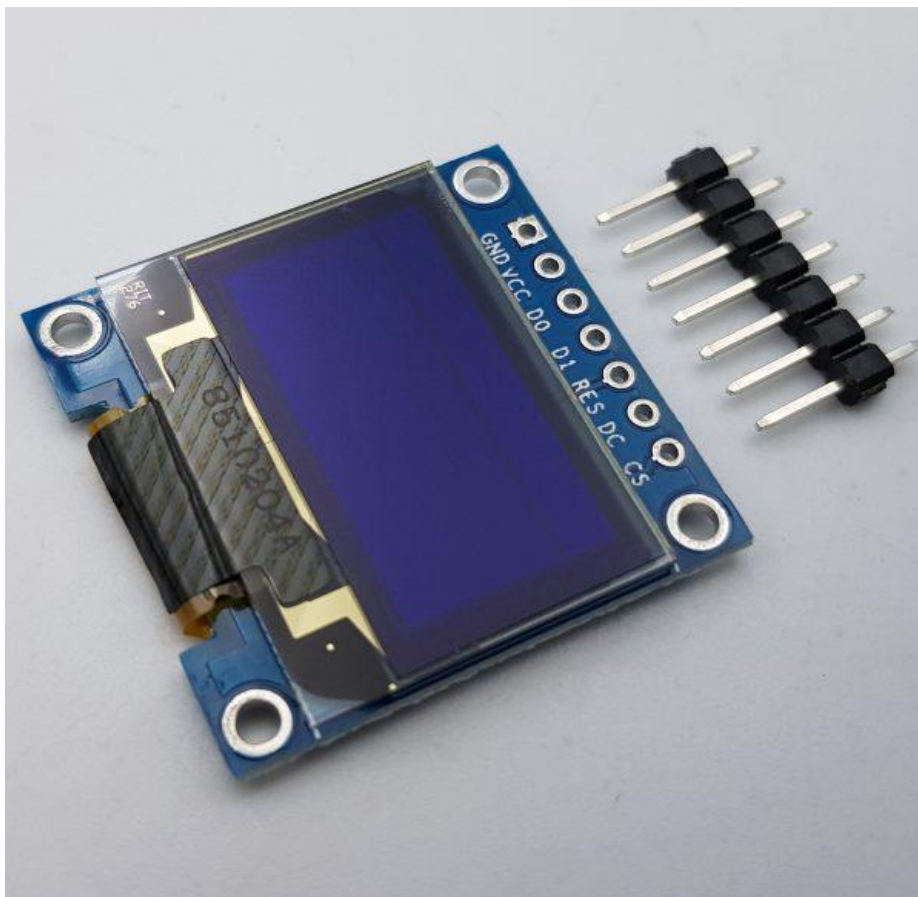
- Servo MG996R (nâng cấp MG995) có momen xoắn lớn
- Lực kéo ở 6V: 11kg
- Đây là bản nâng cấp từ servo MG995 về tốc độ, lực kéo và độ chính xác
- Phù hợp với máy bay cánh quạt loại 50 -90 methanol và máy bay cánh cố định xăng 26cc-50cc
- So với MG946R, MG996R nhanh hơn, nhưng hơi nhỏ hơn.

- Tên sản phẩm: MG996R
- Trọng lượng sản phẩm: 55g
- Kích thước sản phẩm: 40.7 * 19.7 * 42.9mm
- Lực kéo: 9.4kg / cm (4.8V), 11kg / cm (6V)
- Tốc độ xoay: 0.17 giây / 60 độ (4.8 v) 0.14 giây / 60 độ (6 v)
- Điện áp làm việc: 4.8-7.2V
- Nhiệt độ hoạt động: 0 °C -55 °C
- Dòng điện không tải: ~0.15A
- Vật liệu bánh răng: Kim loại
- Các mô hình thích hợp: Máy bay cánh cố định 50 – 90 methanol và máy bay cánh động cơ xăng 26cc-50cc

1.4.1.3. Màn hình OLED

Thông số kỹ thuật:

- *Điện áp sử dụng: 2.2~5.5VDC.*
- *Công suất tiêu thụ: 0.04w*
- *Góc hiển thị: lớn hơn 160 độ*
- *Số điểm hiển thị: 128×64 điểm.*
- *Độ rộng màn hình: 0.96 inch*
- *Màu hiển thị: Trắng / Vàng.*
- *Giao tiếp: SPI*
- *Driver: SSD1306*



Hình 3. Màn hình OLED

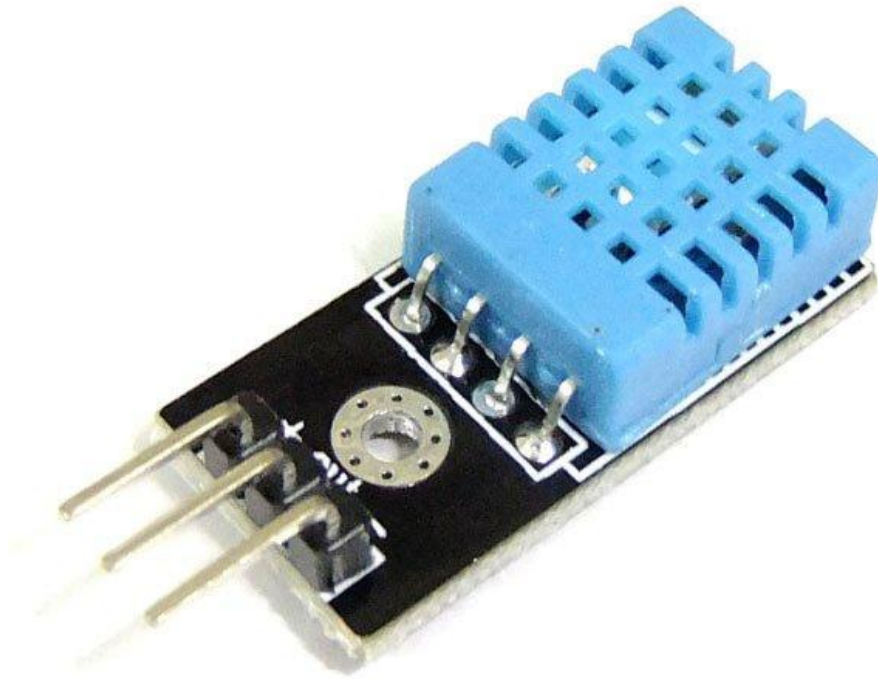
1.4.1.4. Module DHT11 Cảm Biến Nhiệt Độ và Độ Ẩm

Module DHT11 là cảm biến nhiệt độ và độ ẩm giao tiếp với 1 chân dữ liệu, DHT11 đo được giá trị độ ẩm từ 20% đến 90%RH và nhiệt độ từ 0oC đến 50oC, độ chính xác: $\pm 5\%RH$ và $\pm 2oC$.

Thông số kỹ thuật:

- Điện áp hoạt động: 5VDC.
- Chuẩn giao tiếp: TTL, 1 wire.
- Khoảng đo độ ẩm: 20%-80%RH sai số $\pm 5\%RH$.
- Khoảng đo nhiệt độ: 0-50°C sai số $\pm 2^{\circ}C$.
- Tần số lấy mẫu tối đa 1Hz (1 giây / lần).
- Kích thước: 28mm x 12mm x10m.
- DHT11 có 4 chân: VCC, DATA, NC, GND
- Module DHT11 đã được gắn sẵn điện trở và led báo nguồn, nên có 3 chân.
- VCC: Nguồn 3.3 - 5.5VDC
- DATA: Chân dữ liệu

- GND: Nối đất, cực âm



Hình 4. Module DHT11

Nguyên lý hoạt động:

- Khởi tạo: Khi vi điều khiển gửi tín hiệu khởi động, module DHT11 sẽ bắt đầu quá trình đo nhiệt độ và độ ẩm.
- Đo độ ẩm: Cảm biến độ ẩm điện dung sẽ đo độ ẩm dựa trên sự thay đổi điện dung do độ ẩm môi trường.
- Đo nhiệt độ: Điện trở nhiệt bên trong sẽ đo nhiệt độ dựa trên sự thay đổi điện trở theo nhiệt độ môi trường.
- Gửi dữ liệu: IC tích hợp chuyển đổi tín hiệu từ cảm biến thành dữ liệu số và gửi về vi điều khiển qua dây tín hiệu đơn.

Cách sử dụng :

- Chân VCC: Cấp nguồn 3.3V - 5V từ vi điều khiển hoặc nguồn ngoài cho module DHT11.
- Chân GND: Nối đất (GND) với vi điều khiển.
- Chân DATA: Đây là chân truyền dữ liệu. Kết nối chân này với một chân digital trên vi điều khiển để nhận dữ liệu nhiệt độ và độ ẩm.

- Điện trở Pull-up (nếu cần): Kết nối điện trở $10k\Omega$ giữa chân DATA và VCC để ổn định tín hiệu dữ liệu.

1.4.1.5. Module cảm biến thẻ từ RFID MFRC-522 NFC 13.56MHz

Module RFID RC522 NFC 13.56mhz dùng để đọc và ghi dữ liệu cho thẻ NFC tần số 13.56mhz. Với mức thiết kế nhỏ gọn, linh hoạt module này là sự lựa chọn thích hợp cho các ứng dụng đọc – ghi thẻ NFC, đặc biệt khi sử dụng kết hợp với ARDUINO. RFID – Radio Frequency Identification Detection là công nghệ nhận dạng đối tượng bằng sóng vô tuyến. Là một phương pháp nhận dạng tự động dựa trên việc lưu trữ dữ liệu từ xa, sử dụng thiết bị Thẻ RFID và một Đầu đọc RFID.



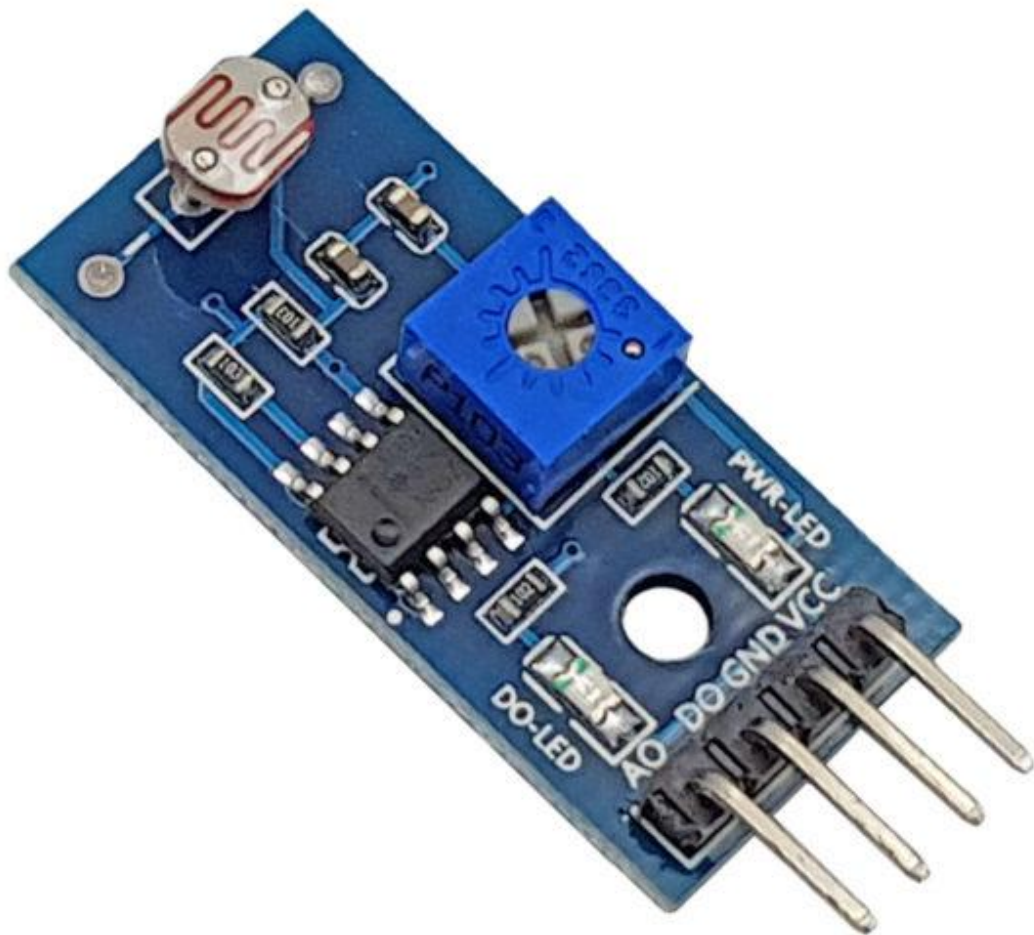
Hình 5. Module RFID MFRC-522 NFC 13.56MHz

Thông số kỹ thuật:

- Nguồn: 3.3VDC, 13 – 26mA
- Dòng ở chế độ chờ: 10-13mA - Dòng ở chế độ nghỉ: <80uA
- Tần số sóng mang: 13.56MHz
- Khoảng cách hoạt động: 0~60mm (mifare1 card)

- Giao tiếp: SPI
- Tốc độ truyền dữ liệu: tối đa 10Mbit/s
- Nhiệt độ hoạt động: -20 đến 80 ° C
- Tốc độ cao SPI: 10Mbit /
- Hỗ trợ: ISO / IEC 14443A /MIFAR
- Kích thước: 60mm×40mm

1.4.1.6. Module cảm biến ánh sáng



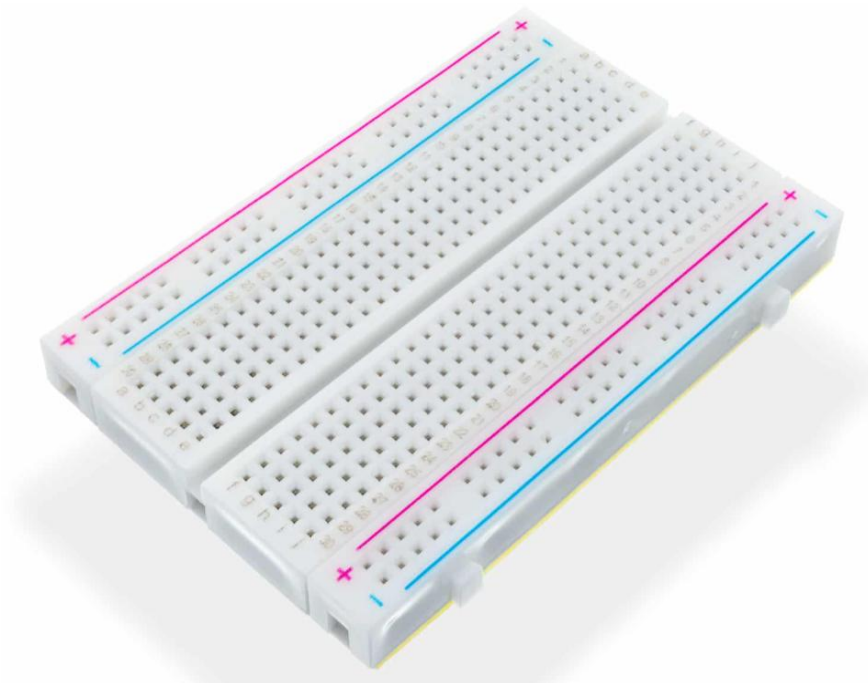
Hình 6. Module cảm biến ánh sáng

Thông số kỹ thuật:

- *điện áp hoạt động 3.3 – 5V*
- *hỗ trợ cả 2 dạng tín hiệu ra Analog và TTL*
- *Ngõ ra Analog 0 – 5V*
- *Kích thước 32 x 14mm*

1.4.1.7. Breadboard

Breadboard bao gồm nhiều lỗ nhỏ được sắp xếp thành hàng và cột, giúp cắm các linh kiện điện tử và dây nối. Bên dưới bề mặt của breadboard có các dây kim loại dẫn điện giúp kết nối các linh kiện với nhau mà không cần phải hàn.



Hình 7. Breadboard

Thông số kỹ thuật:

- Hàng và cột:
 - + Các lỗ được sắp xếp theo hàng ngang và dọc.
 - + Ở giữa breadboard thường có một khe hở, chia thành hai phần: hai nửa này không nối với nhau và được gọi là đường bus dọc và đường bus ngang.
- Dải ngang (hàng): Mỗi hàng ngang gồm 5 lỗ liên tiếp được nối với nhau về mặt điện, giúp kết nối các linh kiện mà không cần thêm dây nối.
- Dải dọc (nguồn và đất): Ở hai bên của breadboard thường có hai dải dọc dài, thường được dùng để cấp nguồn (Vcc) và nối đất (GND) cho toàn bộ mạch.

Chức năng:

- Breadboard cho phép người dùng lắp ráp mạch bằng cách cắm các linh kiện như điện trở, tụ điện, transistor, và dây nối vào các lỗ trên bo mạch. Các chân linh kiện được kết nối với nhau qua các thanh dẫn kim loại nằm bên dưới.
- Đường dẫn ngang: Các lỗ trong mỗi hàng được nối với nhau, giúp linh kiện ở cùng hàng được kết nối.
- Đường dẫn dọc: Dùng để cấp nguồn cho toàn bộ mạch.

1.4.1.8. Còi thụ động



Hình 8. Còi thụ động

Thông số kỹ thuật:

- Điện áp: 5Vdc
- Tần số hoạt động: 2Khz -5Khz
- Kích thước: 12mm* 8.5mm

- Trọng lượng: 1g

1.4.1.9. Đèn Led



Hình 9. LED

Thông số kỹ thuật:

- Led trong siêu sáng
- Chiều dài chân: >20mm
- Đường kính: 5mm
- Điện áp led:
 - Đỏ: 1.8 – 2 V
 - Vàng: 1.8 – 2 V
 - Trắng: 2.8 – 3.2 V

- Xanh lá: 2.8 – 3.2 V
- Xanh dương: 2.8 – 3.2 V
- Vàng Chanh: 2 – 2.4 V
- Tím Nhạt: 3 – 3.3 V
- Dòng: 10 – 20 mA

1.4.1.10. Module thu hồng ngoại + Remote IR1838



Hình 10. Module thu hồng ngoại + Remote IR1838

Thông số kỹ thuật:

- Mạch thu: IR1838
- Điện áp hoạt động: 5VDC.
- Dạng ngõ ra: Digital.
- Sơ Đồ Chân
 - Chân VCC: cung cấp điện áp ngoài 3.3V đến 5V.
 - Chân GND: nối mass nguồn.
 - Chân IN: có điện trở kéo lên 10K, kết nối trực tiếp đến vi điều khiển.

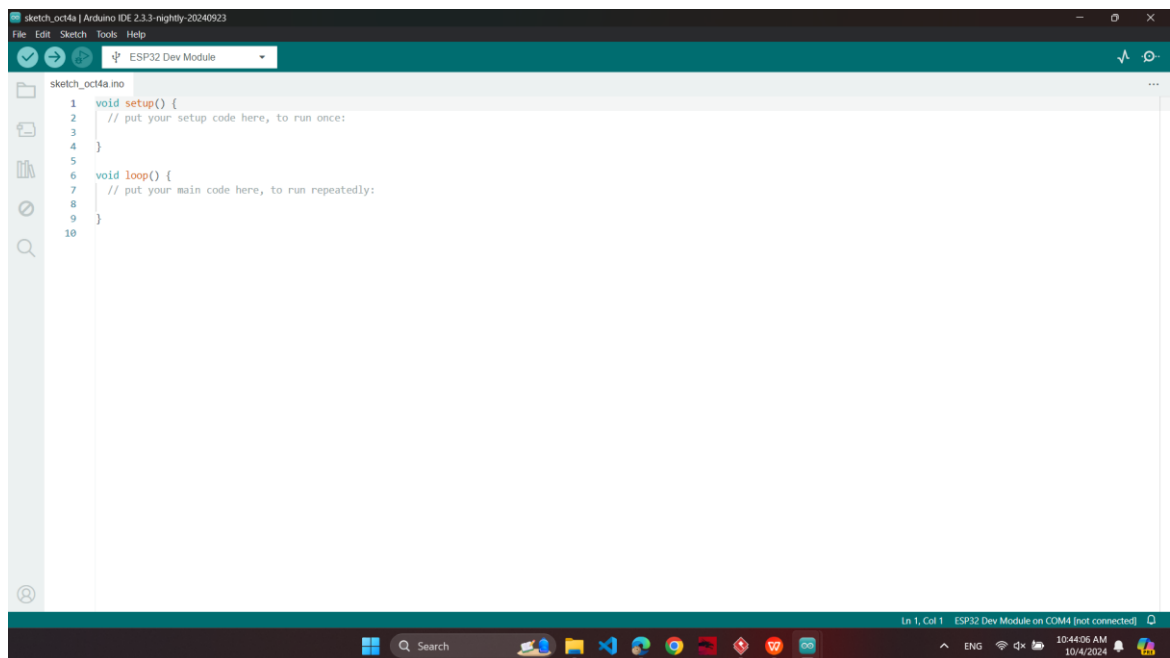
1.4.2. Các nền tảng phần mềm

1.4.2.1. Arduino IDE

Arduino IDE là một nền tảng mã nguồn mở được sử dụng để xây dựng các dự án điện tử, chủ yếu để viết và biên dịch mã vào module. Có rất nhiều các module Arduino như Arduino Uno, Arduino Mega, Arduino Leonardo, Arduino Micro... Mỗi module chứa một bộ vi điều khiển trên bo mạch được lập trình và chấp nhận thông tin dưới dạng mã.

Một chương trình viết cho Arduino, còn được gọi là một sketch, được tạo trên nền tảng Arduino IDE sẽ tạo ra một file Hex, sau đó file này được tải lên (upload) bộ vi điều khiển trên bảng mạch Arduino.

Môi trường IDE chủ yếu chứa hai phần cơ bản: Trình soạn thảo và Trình biên dịch, phần đầu sử dụng để viết mã được yêu cầu và phần sau được sử dụng để biên dịch và tải mã lên module Arduino.



Hình 11. Cửa sổ chương trình Arduino IDE

Môi trường IDE hỗ trợ cả ngôn ngữ C và C ++. Điều này giúp tối giản hóa cú pháp và làm cho việc viết mã trở nên đơn giản hơn. Arduino IDE cũng cung cấp một loạt các thư viện hỗ trợ, giúp tối ưu hóa thời gian và công sức trong việc tích hợp các chức năng như điều khiển cảm biến, động cơ, và giao tiếp với các thiết bị ngoại vi.

Để viết và nạp chương trình cho KIT Arduino ESP32, thực hiện các bước sau:

- Kết nối dây giữa cổng USB của máy tính và lõi vào cổng MicroUSB của ESP32.
- Khởi động chương trình soạn thảo và biên dịch mã nguồn Arduino IDE (*Chú ý: luôn cập nhật phiên bản mới nhất có thể*).
- Lựa chọn bảng mạch: ESP32 Dev Module
- Chọn cổng COM ảo cho phù hợp để truyền dữ liệu từ bảng mạch lên terminal của máy tính (tùy từng máy tính khác nhau số hiệu cổng sẽ khác nhau). Trong ví dụ này, cổng COM được chọn là COM3.

1.4.2.2. Web Blynk

The screenshot shows the Blynk web interface. At the top, there's a header with a device icon and the name 'Nha thông minh BTL IoT N03', followed by an 'Edit' button. Below this is the 'Datastreams' section, which includes a search bar and a table of data streams.

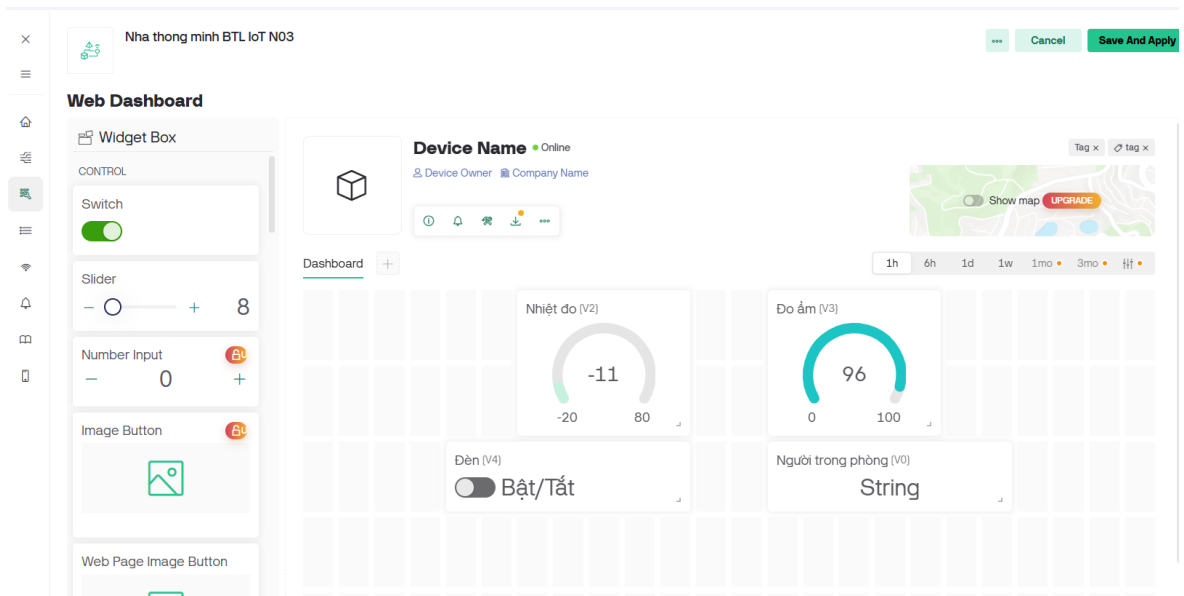
Id	Name	Alias	Color	Pin	Data Type	Units	Is Raw	Min	Max
3	Nhiệt độ	Nhiệt độ	Orange	V2	Double		false	-20	80
4	Độ ẩm	Độ ẩm	Teal	V3	Double		false	0	100
5	Đèn	đèn	Yellow	V4	Integer		false	0	1
6	Có người	CóNgười	Light Orange	V0	String		false		

Giao diện thiết lập chân ảo

The screenshot shows the Blynk web interface's 'Home' section. On the left is a sidebar with navigation links: Home, Datastreams, Web Dashboard, Metadata, Connection Lifecycle, Events & Notifications, User Guides, and Mobile Dashboard. The main content area is titled 'Home' and shows '1 Devices'. A table lists the device 'Nha thông minh BTL IoT N03' with status 'Inactive' and an auth token. To the right, there's a 'What's next?' section and a 'Template settings' section for 'ESP32, WiFi'. The 'Firmware configuration' section shows the template ID and name, and a code block for the Blynk template.

```
#define BLYNK_TEMPLATE_ID  
"TRPL6Axj_BH55"  
#define BLYNK_TEMPLATE_NAME "Nha  
thông minh BTL IoT N03"
```

Hình 12. Giao diện thiết đặt thiết bị IoT



Hình 13. Giao diện thiết kế giao diện web app

Tính năng nổi bật:

- Thông báo nhiệt độ độ ẩm của phòng cập nhật liên tục
- Thông báo cho người dùng biết có người trong phòng không
- Bật/tắt đèn từ xa

Cách sử dụng:

- Nhấn nút Bật/Tắt để điều khiển đèn
- Tự động

1.4.2.3. Framework Flask



- Flask là loại framework web phổ biến được viết bằng trình lập ngôn ngữ Python. Công nghệ thường được sử dụng để xây dựng trang web từ những ứng dụng đơn giản đến những hệ thống phức tạp hơn.
- Flask được thiết kế để hoạt động và mở rộng một cách, đồng thời nó cũng cung cấp các công cụ và thư viện cần thiết để phát triển ứng dụng web hiệu quả. Flask cũng có cộng đồng sáng tạo và hỗ trợ mạnh mẽ từ cộng đồng Python.
- Cũng giống như Django, Python Flask là một micro-framework được viết bằng ngôn ngữ lập trình Python dùng cho các nhà phát triển web. Nó được phát triển bởi Armin Ronacher, người dẫn đầu một nhóm những người đam mê Python quốc tế có tên là Poocco. Flask dựa trên bộ công cụ Werkzeug WSGI và template engine Jinja2. Cả hai đều là các dự án của Poocco. Micro ở đây không có nghĩa là framework này thiếu các chức năng mà thể hiện ở việc nó sẽ cung cấp những chức năng “cốt lõi” nhất cho các ứng dụng web và có khả năng mở rộng, người dùng cũng có thể mở rộng bất cứ lúc nào vì Flask hỗ trợ rất nhiều các tiện ích mở rộng như tích hợp CSDL, hệ thống upload, xác thực, template, email... Việc là một micro-framework cũng giúp cho flask có một môi trường xử lý độc lập và ít phải sử dụng các thư viện bên ngoài, điều này giúp nó nhẹ và ít gặp phải các lỗi hơn, việc phát hiện và xử lý các lỗi cũng dễ dàng và đơn giản hơn.

- Ưu điểm của Python Flask

- Siêu nhỏ nhẹ, là một công cụ tối giản
- Tốc độ hoạt động cực nhanh
- Có khả năng hỗ trợ NoQuery
- Tương đối đơn giản (so với các framework có cùng chức năng khác)
- Mang lại khả năng kết nối với các tiện ích mở rộng bởi không có ORM
- Trình duyệt được nhúng sẵn trình gỡ rối
- Sử dụng các mã ngắn, đơn giản trong những bộ xương Python
- Ngăn chặn các rủi ro về bảo mật khi lập trình web do ít phụ thuộc vào bên thứ ba
- Có khả năng kiểm soát mọi vấn đề khi dùng Flask.
- Cho phép biên dịch mô-đun, thư viện, giúp việc lập trình nhanh chóng, dễ dàng hơn và không cần gõ code bậc thấp

- Nhược điểm của Python Flask

Chính vì siêu nhỏ nhẹ và tối giản, Flask không phải là một lựa chọn tốt nếu lập trình viên muốn một framework có đầy đủ các tính năng. Thay vào đó, lập trình viên sẽ phải tự gọi các tiện ích mà mình có nhu cầu sử dụng vì nó không được

tích hợp sẵn trong framework, và đôi khi việc này trở nên bất tiện và khiến cho khối lượng công việc phải làm tăng lên đáng kể.

1.4.2.4. Render

Render.com là một nền tảng **Cloud Hosting** được thiết kế để triển khai, vận hành và quản lý các ứng dụng, dịch vụ web, cơ sở dữ liệu, và các tài nguyên khác. Render.com mang đến sự kết hợp giữa khả năng mở rộng của dịch vụ đám mây và sự đơn giản trong việc triển khai, giúp các nhà phát triển dễ dàng tập trung vào việc xây dựng ứng dụng thay vì quản lý hạ tầng.

Tính năng nổi bật của Render

Triển khai dễ dàng

Render cho phép triển khai ứng dụng chỉ bằng một vài bước. Kết nối trực tiếp với GitHub hoặc GitLab để tự động triển khai sau mỗi lần đẩy mã (push).

Hỗ trợ các ứng dụng được viết bằng nhiều ngôn ngữ như Python, Node.js, Ruby, Java, Go, và PHP.

Hỗ trợ nhiều loại dịch vụ

- **Web Services:** Lưu trữ các ứng dụng web.
- **Static Sites:** Lưu trữ các trang web tĩnh với tích hợp CDN miễn phí.
- **Background Workers:** Chạy các tác vụ nền hoặc các công việc định kỳ.
- **Docker Services:** Hỗ trợ triển khai các container Docker tùy chỉnh.
- **Cron Jobs:** Tự động hóa các công việc theo lịch trình.
- **Databases:** Quản lý cơ sở dữ liệu PostgreSQL.

Tích hợp HTTPS miễn phí

Render tự động cung cấp và gia hạn chứng chỉ SSL (HTTPS) miễn phí thông qua Let's Encrypt.

Tính năng mở rộng (Scaling)

Dễ dàng mở rộng ứng dụng cả chiều ngang (horizontal scaling) lẫn chiều dọc (vertical scaling).

Tăng hoặc giảm số lượng máy chủ chỉ bằng một vài thao tác.

Giám sát và Logging

Giao diện đơn giản để theo dõi trạng thái của ứng dụng.

Cung cấp log thời gian thực để kiểm tra và khắc phục sự cố.

Giá cả hợp lý

Render cung cấp một mô hình định giá linh hoạt với **miễn phí bắt đầu** cho các dự án nhỏ và trả phí dựa trên tài nguyên sử dụng.

Lợi ích của Render.com

- **Dễ sử dụng:** Không yêu cầu kiến thức sâu về hạ tầng hoặc DevOps.
- **Tích hợp mạnh mẽ:** Tích hợp với các công cụ phát triển như GitHub, GitLab, và CI/CD pipelines.
- **Nhanh chóng và an toàn:** Đảm bảo tốc độ truy cập nhanh thông qua CDN, cùng với bảo mật HTTPS.
- **Phù hợp với nhiều quy mô:** Từ các cá nhân đến doanh nghiệp lớn.

CHƯƠNG 2. THIẾT KẾ HỆ THỐNG

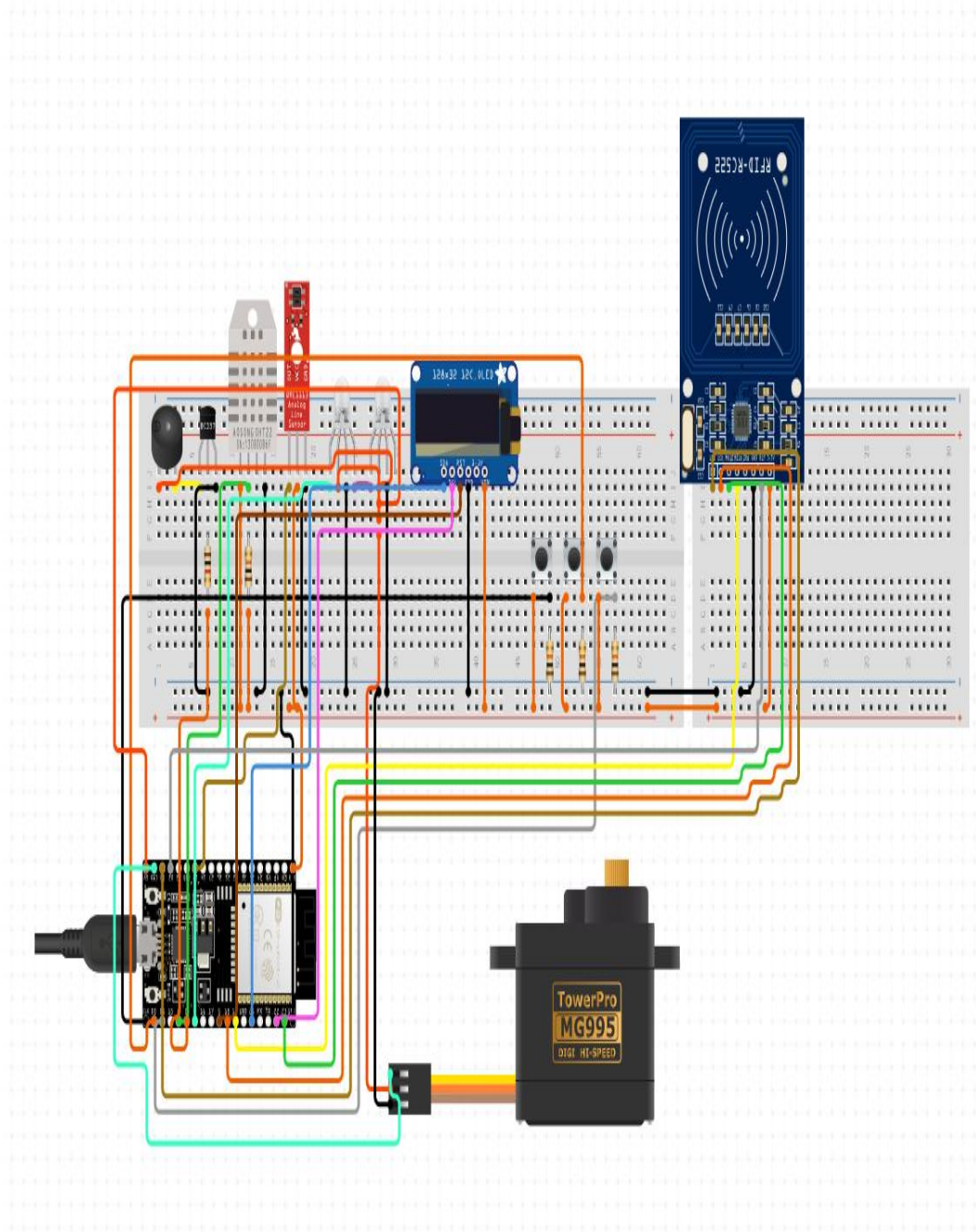
2.1. Lập bảng dữ liệu

Trên web		Trên ESP32
Chức năng		Chân kết nối
Đầu đọc chip NFC <i>RFID - RC522</i>	Dùng để giao tiếp với thẻ vật lý, có thẻ đọc và ghi dữ liệu lên thẻ.	<ul style="list-style-type: none"> • <i>SDA - D05</i> • <i>SCK - D18</i> • <i>MOSI - D23</i> • <i>MISO - D19</i> • <i>IRQ - XXX</i> • <i>RST - D2</i> • <i>GDN - GND</i> • <i>3V3 - 3V3</i>
Kết nối màn LCD <i>Oled screen (I2C)</i>	Hiển thị thông tin người dung ra vào, thông tin nhiệt độ độ ẩm	<ul style="list-style-type: none"> • <i>SDA - D21</i> • <i>SCL - D22</i>
Động cơ Servo	Sử dụng để thay thế then chốt cửa	<ul style="list-style-type: none"> • <i>In - D13 (Out)</i> • <i>VCC - 5v</i> • <i>GND - GND</i>
Cảm biến ánh sáng <i>LDR</i>	Sử dụng để đo cường độ ánh sáng trong phòng	<ul style="list-style-type: none"> • <i>Out - D35 (In)</i> • <i>VCC - 3v3</i> • <i>GND - GND</i>
Cảm biến nhiệt độ, độ ẩm <i>DHT11</i>	Sử dụng để đo nhiệt độ và độ ẩm môi trường trong phòng	<ul style="list-style-type: none"> • <i>Out - D26 (In)</i> • <i>VCC - 3v3</i> • <i>GND - GND</i>
Đèn trong nhà (8 <i>Led mắc song song</i>)	Minh họa cho đèn trong nhà	<ul style="list-style-type: none"> • <i>GND - GND</i> • <i>VCC - D12 (Out)</i>
Điều hòa (<i>Led</i>)	Minh họa cho điều hòa trong nhà	<ul style="list-style-type: none"> • <i>GND - GND</i> • <i>VCC - D14 (Out)</i>

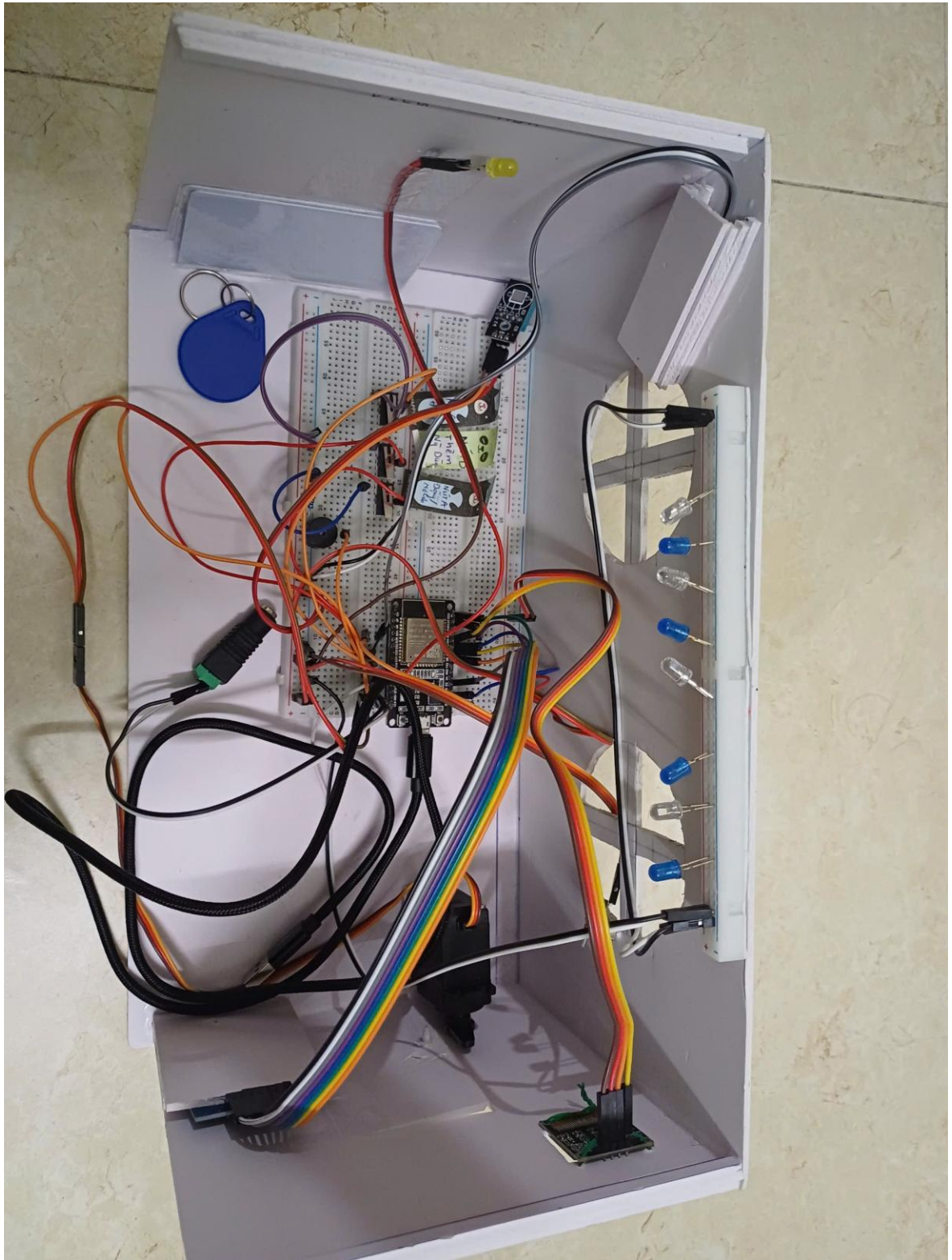
Công tắc đóng/mở cửa (<i>Button A</i>)	Mở/Đóng cửa ra vào	<ul style="list-style-type: none"> • <i>In - 3V3</i> • <i>Out - R(10k) - D33 (In)</i>
Nút chức năng thêm người dùng (<i>Button B</i>)	Bật/Tắt chức năng thêm người dùng	<ul style="list-style-type: none"> • <i>In - 3V3</i> • <i>Out - R(10k) - D32 (InChân D12)</i>
Công tắc bật/tắt đèn (<i>Button C</i>)	Bật/Tắt đèn thủ công	<ul style="list-style-type: none"> • <i>In - 3V3</i> • <i>Out - R(10k) - D25 (In)</i>
Buzzer thụ động	Phát ra âm thanh	<ul style="list-style-type: none"> • <i>GND - GND</i> • <i>VCC - D27 (Out)</i>
Remote IR1838	Cảm biến hồng ngoại	<ul style="list-style-type: none"> • <i>GND – GND</i> • <i>VCC – 3,3V</i> • <i>Out – D34</i>

Bảng 1. Bảng dữ liệu thiết kế hệ thống

2.2. Thiết kế sơ đồ mạch điện

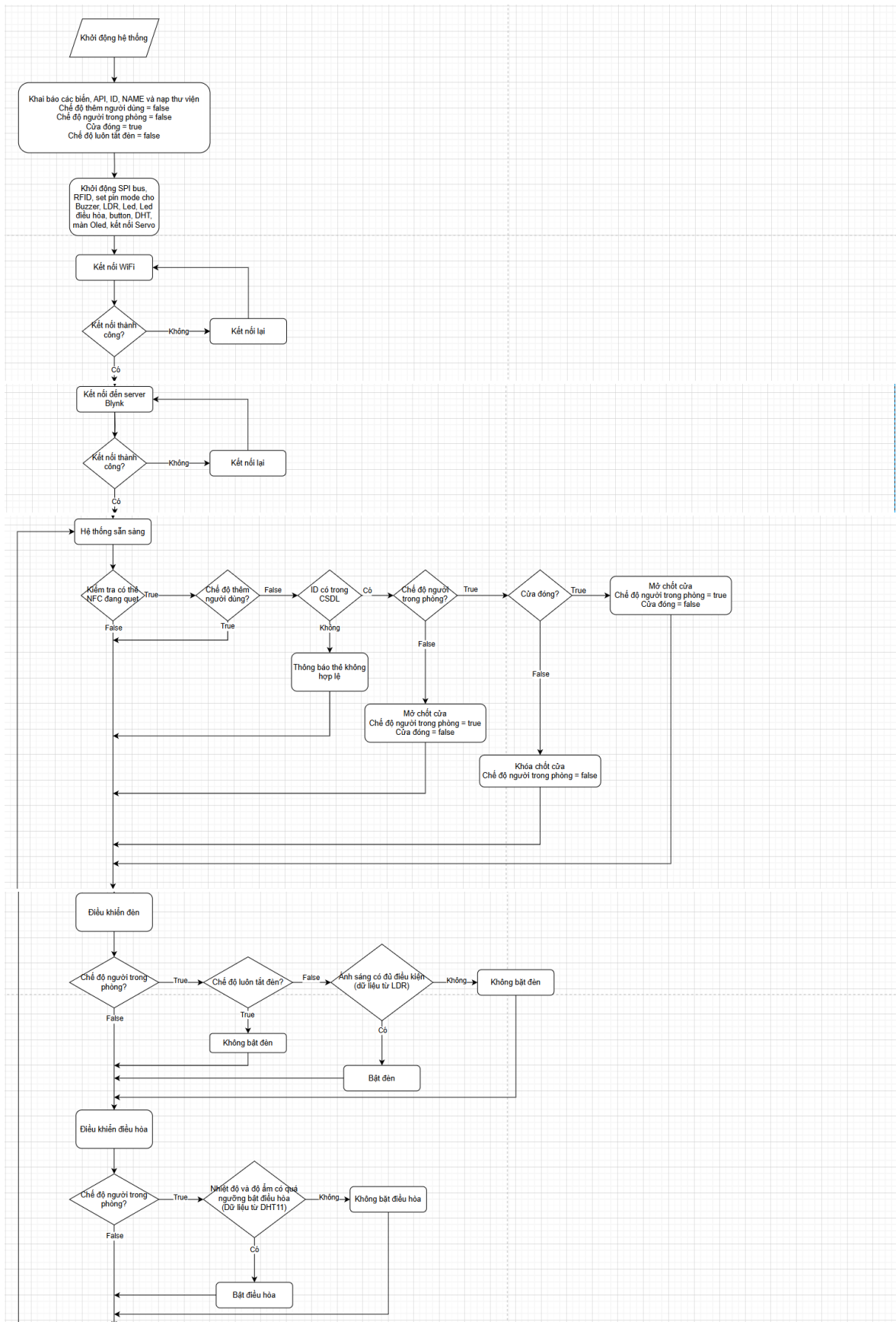


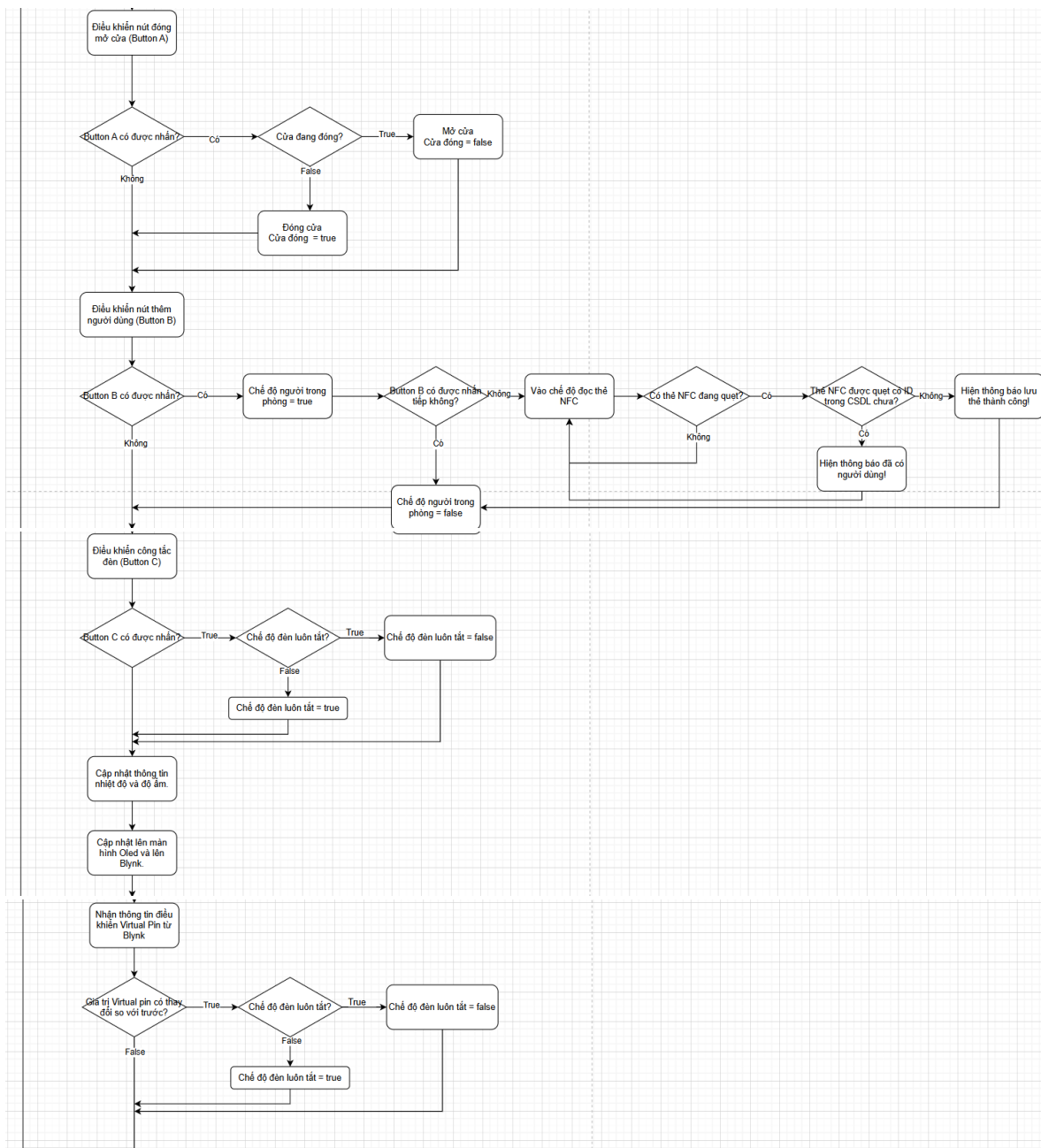
Hình 14. Sơ đồ mạch điện in



Hình 15. Sơ đồ mạch điện thực tế

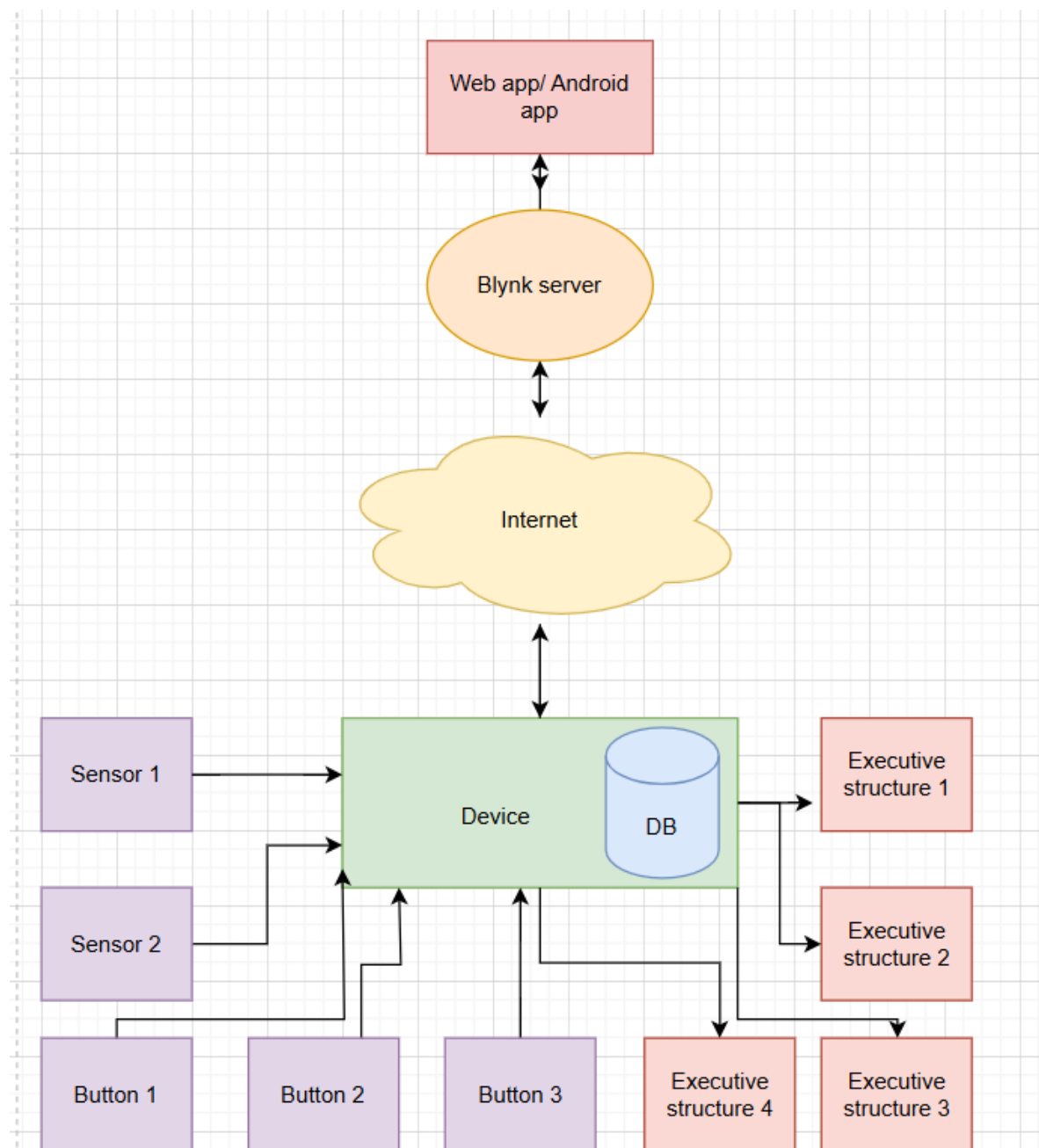
2.3. Lưu đồ thuật toán





Hình 16. Lưu đồ thuật toán

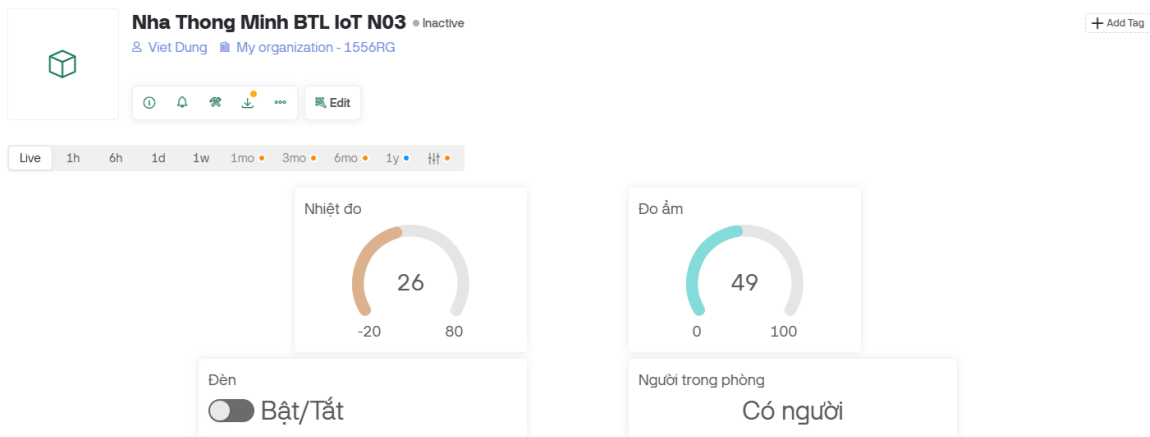
2.4. Đặc tả cấp độ IoT



Hình 17. Đặc tả cấp độ IoT

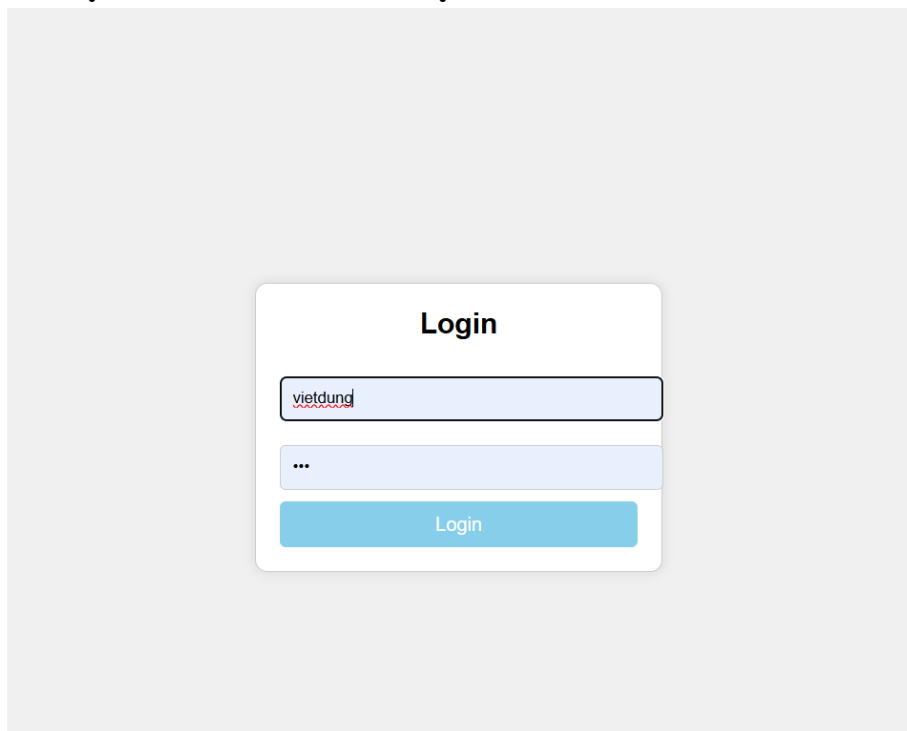
2.5. Thiết kế ứng dụng web

- *Giao diện website Blynk:*

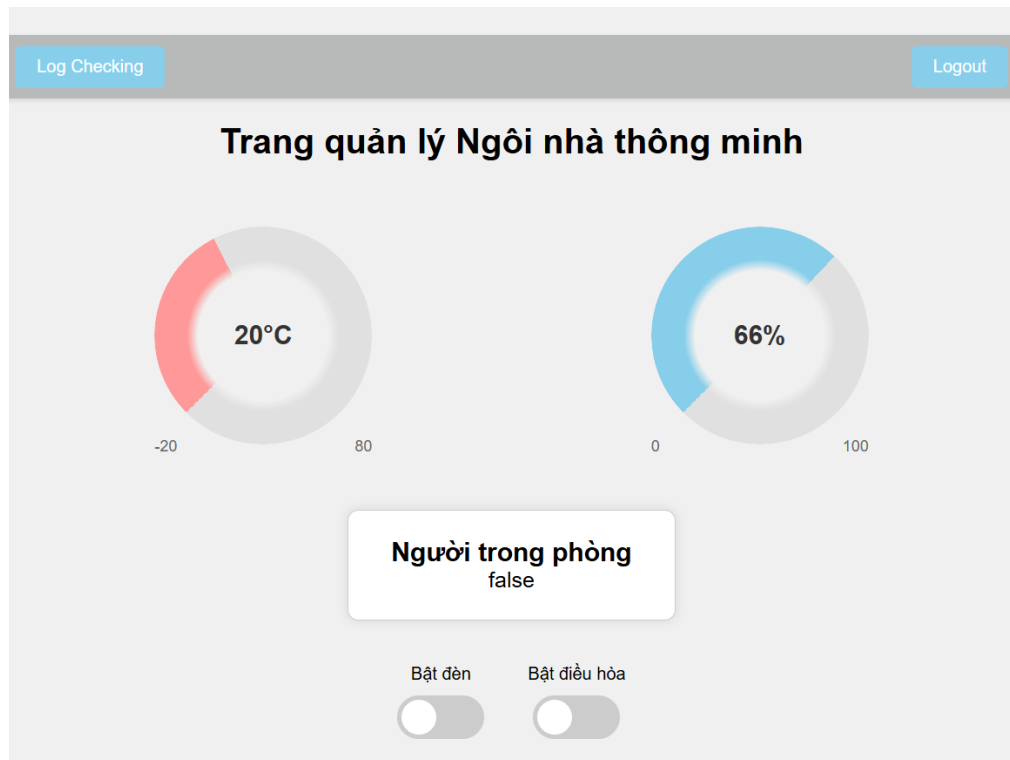


Hình 18. Giao diện web Blynk

- **Giao diện website Webserver tự host:**



Hình 19. Login page của web tự host



Hình 20. Trang DashBoard web tự host

2.6. Thêm mục các thư viện đã sử dụng và nêu ý nghĩa của từng thư viện

#include <BlynkSimpleEsp32.h>

- Thư viện Blynk cho ESP32 cho phép kết nối Arduino hoặc ESP32 với ứng dụng Blynk. Qua đó, bạn có thể điều khiển thiết bị IoT từ xa và giám sát các cảm biến qua điện thoại hoặc máy tính bảng.

#include <Wire.h>

- Thư viện Wire hỗ trợ giao tiếp I2C giữa vi điều khiển và các thiết bị ngoại vi như màn hình, cảm biến. Đây là giao thức phổ biến để kết nối nhiều thiết bị dùng hai dây (SDA và SCL).

#include <Adafruit_GFX.h> và #include <Adafruit_SSD1306.h>

- Adafruit_GFX là thư viện hỗ trợ đồ họa cơ bản như vẽ hình, hiển thị văn bản. Thư viện này thường kết hợp với các màn hình đồ họa.
- Adafruit_SSD1306 dành riêng cho màn hình OLED SSD1306, giúp điều khiển và hiển thị trên màn hình.

#include <MFRC522.h>

- Đây là thư viện điều khiển module RFID RC522, giúp đọc và ghi dữ liệu từ thẻ RFID. Nó rất hữu ích trong các dự án liên quan đến truy cập và xác thực (ví dụ: hệ thống khóa cửa thông minh).

#include <PlayNote.h>

- Thư viện PlayNote hỗ trợ phát nhạc hoặc âm thanh. Thư viện này thường sử dụng để tạo âm thanh đơn giản qua loa buzzer trên vi điều khiển Arduino.

#include "DHTesp.h"

- DHTesp hỗ trợ các cảm biến DHT (DHT11, DHT22) đo nhiệt độ và độ ẩm. Thư viện này đặc biệt hữu ích trong các dự án liên quan đến giám sát môi trường.

#include <ESP32Servo.h>

- ESP32Servo hỗ trợ điều khiển servo với ESP32, giúp quay và điều khiển các động cơ servo trong các dự án robot hoặc điều khiển từ xa.

#include <WiFi.h> và #include <WiFiClient.h>

- WiFi cho phép ESP32 kết nối với mạng Wi-Fi, hỗ trợ giao tiếp không dây.
- WiFiClient tạo kết nối khách hàng (client) để giao tiếp với máy chủ thông qua giao thức TCP/IP, thích hợp để gửi/nhận dữ liệu từ internet hoặc mạng cục bộ.

2.7. Code chương trình

2.7.1. Khai báo thư viện, hằng số và khởi tạo:

- Khai báo các thư viện cần thiết và định nghĩa WiFi

```
#define BLYNK_TEMPLATE_ID "TMPL6Axj_8H35"
#define BLYNK_TEMPLATE_NAME "Nha thong minh BTL IoT N03"
#define BLYNK_AUTH_TOKEN "7R8DrwAGMsVPuD1pmEsjjKqi4aCz0Jp3"
#define BLYNK_PRINT Serial
#include <BlynkSimpleEsp32.h>

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <MFRC522.h>
#include <PlayNote.h>
#include "DHTesp.h"
#include <ESP32Servo.h>
#include <WiFi.h>
#include <WiFiClient.h>

const char* ssid = "Dung";
const char* pass = "khongcomatkhou!";
```

Hình 2.8: Code khai báo thư viện và hằng số

Khởi tạo các thành phần như màn OLED, điều khiển MFRC522, điều khiển động cơ, âm thanh và cảm biến nhiệt độ, độ ẩm.

```

// OLED
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// RFID
MFRC522 mfrc522(SS_PIN, RST_PIN);

// Servo
Servo myServo;
const int ViTriDongCua = 150;
const int ViTriMoCua = 110;

PlayNote playNote;

DHTesp dhtSensor;

```

Hình 2.9: Code khởi tạo điều khiển các module

```

// Định nghĩa pin kết nối
#define SS_PIN 5
#define RST_PIN 2
#define SERVO_PIN 13
#define BUZZER_PIN 27
#define LDR_PIN 35
#define DHT11_PIN 26
#define LED_PIN 12
#define DieuHoa_PIN 14
#define BUTTON_Cua_PIN 33
#define BUTTON_ThemNg_PIN 32
#define BUTTON_Den_PIN 25

```

Hình 3.0: Code định nghĩa chân cảm biến

Khởi tạo biến lưu trạng thái dữ liệu hệ thống

```

// Biến lưu trạng thái hệ thống
String savedUIDs[] = {" 04 43 0E C5 20 02 89", " 13 29 B2 E2", " 04 63 F8 BA 20 02 89", " 04 13 CC B7 20 02 89", "", "", "", "", "", ""}; // UID thẻ đã lưu
String usernames[] = {"Việt Dũng", "Van Kien", "Hai Long", "The Anh", "", "", "", "", "", ""};
int currentPosServo = ViTriDongCua;
int savedUsers = 4; // số lượng người dùng đã lưu
bool addUserMode = false; // chế độ thêm người dùng
bool hasPeopleMode = false; // chế độ con người dùng trong phòng
bool isDoorClosed = true;
bool isTurnOffLight = false;
bool pressedButtonThemNguoi = false;

unsigned long lastClearTime = 0;
const unsigned long clearInterval = 8000;
unsigned long lastMessageTime = 0;
const unsigned long displayDuration = 3000;

```

Hình 3.1: Code khởi tạo biến lưu trạng thái hệ thống

2.7.2. Cấu hình các thiết lập cho thiết bị khi ESP32 khởi động:

- Thực hiện các bước khởi tạo cho hệ thống, bao gồm cấp nguồn cho động cơ, NFC, cảm biến hồng ngoại, cảm biến độ ẩm, thiết lập kết nối Wi-Fi, kết nối đến máy chủ Blink và Webserver cá nhân.
- Tất cả những bước này đảm bảo hệ thống đã sẵn sàng để vận hành.

```
void setup() {  
    Serial.begin(115200);  
    SPI.begin(); // khởi động SPI bus  
    mfrc522.PCD_Init(); // khởi động RFID  
    mfrc522.PCD_DumpVersionToSerial();  
  
    playNote.setBuzzerPin(BUZZER_PIN);  
    pinMode(LDR_PIN, INPUT);  
    pinMode(LED_PIN, OUTPUT);  
    pinMode(DieuHoa_PIN, OUTPUT);  
    pinMode(BUTTON_Cua_PIN, INPUT_PULLUP);  
    pinMode(BUTTON_ThemNg_PIN, INPUT_PULLUP);  
    pinMode(BUTTON_Den_PIN, INPUT_PULLUP);  
  
    dhtSensor.setup(DHT11_PIN, DHTesp::DHT11);  
  
    // Khởi động màn hình OLED  
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {  
        Serial.println(F("SSD1306 allocation failed"));  
        for(;;);  
    }  
    display.clearDisplay();  
    display.setTextSize(1);  
    display.setTextColor(SSD1306_WHITE);  
    display.display();  
  
    myServo.attach(SERVO_PIN);  
    myServo.write(ViTriDongCua); // ban đầu servo ở vị trí đóng cửa  
    delay(1000);  
    pinMode(SERVO_PIN, INPUT);  
  
    WiFi.begin(ssid, pass);  
    Serial.print("connecting");  
    displayMessage("connecting");  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(500);  
        Serial.print(".");  
        display.print(".");  
        display.display();  
    }  
    Serial.println("");  
    Serial.println("Connected to WiFi network with IP Address: " + String(WiFi.localIP()));  
    displayMessage("Connected to WiFi network with IP Address: " + String(WiFi.localIP()));  
    Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);  
    blynkTimer.setInterval(1000L, myTimer);  
    // Khởi động hoàn tất  
    Serial.println("Hệ thống sẵn sàng.");  
    displayMessage("He thong san sang.");  
}
```

Hình 3.2: Code cấu hình các thiết lập cho thiết bị khi ESP32 khởi động

2.7.3. Hàm loop() - vòng lặp chính:

Thực hiện lặp lại các công việc sau trong khoảng thời gian ESP32 hoạt động:

- Chạy các chức năng chính của Blynk để xử lý giao tiếp và cập nhật dữ liệu với ứng dụng Blynk, giúp duy trì kết nối và quản lý thời gian của các sự kiện trong hệ thống.
- Kiểm tra và đọc thẻ RFID. Nếu thẻ có UID hợp lệ, thực hiện xử lý mở cửa; nếu không, xử lý thất bại.
- Tự động điều khiển đèn và điều hòa thông qua các hàm `dieuKhienDen()` và `dieuKhienDieuHoa()`.
- Kiểm tra trạng thái nút điều khiển cửa. Khi nhấn nút, chuyển đổi giữa mở và đóng cửa, đồng thời phát âm thanh và hiển thị thông báo tương ứng.
- Khi nhấn nút thêm người dùng, chuyển đổi vào/ra chế độ thêm người dùng mới.
- Kiểm tra nút bật/tắt đèn. Khi nhấn nút, chuyển đổi giữa trạng thái bật và tắt đèn, phát âm thanh và hiển thị thông báo tương ứng.
- Kiểm tra đọc tín hiệu từ điều khiển từ xa, ứng với mỗi nút sẽ điều khiển cho một chức năng tương ứng.
- Định kỳ hiển thị nhiệt độ và độ ẩm môi trường, cho biết hệ thống đang sẵn sàng hoạt động.
- Hàm `HttpSync()` là để đồng bộ dữ liệu của Esp32 và Webserver.

```

void loop() {
    Blynk.run();
    blynkTimer.run();
    if (!addUserMode) {
        // mfrc522.PCD_Init();
        if (mfrc522.PICC_IsNewCardPresent() && mfrc522.PICC_ReadCardSerial()) {
            String cardUID = getCardUID();
            Serial.printf("Card UID: %s\n", cardUID);
            //Serial.println(cardUID);

            // Kiểm tra thẻ có trong hệ thống không
            int userIndex = checkUser(cardUID);
            if (userIndex != -1) {
                // Đúng UID
                handleSuccess(cardUID, userIndex);
            } else {
                // Sai UID
                handleFailure();
            }

            mfrc522.PICC_HaltA(); // Ngắt giao tiếp với thẻ
            // IrReceiver.restartTimer();
        }
    }

    // Kiểm tra nút A để khóa cửa
    if (digitalRead(BUTTON_Cua_PIN) == LOW) {
        if (isDoorClosed) {
            isDoorClosed = false;
            displayMessage("Dang mo cua...");
            buttonCuaMoSound();
            openDoor();
        } else {
            isDoorClosed = true;
            displayMessage("Dang dong cua...");
            buttonCuaDongSound();
            lockDoor();
        }
    }

    // Kiểm tra nút B để vào chế độ thêm người dùng mới
    if (digitalRead(BUTTON_ThemNg_PIN) == LOW) {
        buttonThemNgSound();
        if (addUserMode) {
            addUserMode = false;
        } else {
            addUserMode = true;
        }
        pressedButtonThemNguoi = false;
        addUser();
    }
}

```

```

if (digitalRead(BUTTON_Den_PIN) == LOW) {
    if (isTurnOffLight) {
        isTurnOffLight = false;
        displayMessage("Bat den...");
        buttonBatDenSound();
    } else {
        isTurnOffLight = true;
        displayMessage("Tat den...");
        buttonTatDenSound();
    }
}

// dieuKhienDen();

if (IrReceiver.decode()) {
    /*
     * Print a summary of received data
     */
    if (IrReceiver.decodedIRData.protocol == UNKNOWN) {
        Serial.println(F("Received noise or an unknown (or not yet enabled) protocol"));
        // We have an unknown protocol here, print extended info
        IrReceiver.printIRResultRawFormatted(&Serial, true);
        IrReceiver.resume(); // Do it here, to preserve raw data for printing with print
    } else {
        IrReceiver.resume(); // Early enable receiving of the next IR frame
        IrReceiver.printIRResultShort(&Serial);
        IrReceiver.printIRSendUsage(&Serial);
    }
    Serial.println();

    /* Finally, check the received data and perform actions according to the received command
     */
    if (IrReceiver.decodedIRData.flags & IRDATA_FLAGS_IS_REPEAT) {
    }
    else {
        if (IrReceiver.decodedIRData.command == 0x45) {
            Serial.println("On/Off");
            if (isTurnOffLight)
            {
                isTurnOffLight = false;
                displayMessage("Bat den...");
                buttonBatDenSound();
            }
            else
            {
                isTurnOffLight = true;
                displayMessage("Tat den...");
                buttonTatDenSound();
            }
        }
        else if (IrReceiver.decodedIRData.command == 0x47) {
            Serial.println("Speed");
            if (isTurnOffDieuHoa)
            {
                isTurnOffDieuHoa = false;
                displayMessage("Bat dieu hoa...");
                buttonBatDieuHoaSound();
            }
            else
            {
                isTurnOffDieuHoa = true;
                displayMessage("Tat dieu hoa...");
                buttonTatDieuHoaSound();
            }
        }
        else if (IrReceiver.decodedIRData.command == 0x7) {
            Serial.println("Timer");
        }
    }
}

```



```
String getCardUID() {
    String uidString = "";
    for (byte i = 0; i < mfrc522.uid.size; i++) {
        uidString += String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
        uidString += String(mfrc522.uid.uidByte[i], HEX);
    }
    uidString.toUpperCase();
    return uidString;
}
```

Hình 3.4: Hàm chuyển đổi ký tự người dùng

- Hàm kiểm tra người dùng đã tồn tại chưa

```
int checkUser(String uid) {
    for (int i = 0; i < savedUsers; i++) {
        Serial.printf("Nguoi dung da luu UID: %s\n", savedUIDs[i]);
        if (savedUIDs[i] == uid) {
            return i; // trả về index của người dùng
        }
    }
    return -1; // không tìm thấy người dùng
}
```

Hình 3.5: Hàm kiểm tra người dùng đã tồn tại chưa

2.7.5. Hàm đọc thẻ người dùng

- Khi đọc thẻ người dùng thành công và thất bại

```

void handleSuccess(String uid, int userIndex) {
    displayMessage("Doc the thanh cong.");
    readDoneSound();
    delay(500);

    // Hiển thị thông tin người dùng

    if(hasPeopleMode)
    {
        if(isDoorClosed)
        {
            displayMessage("Xin chao " + usernames[userIndex] + ".\nXin moi vao.");
            openDoor();
            hasPeopleMode = true;
            isDoorClosed = false;
        }
        else
        {
            displayMessage("Tam biet " + usernames[userIndex] + ".\nHen gap lai.");
            lockDoor();
            hasPeopleMode = false;
            isDoorClosed = false;
        }
    }
    else
    {
        displayMessage("Xin chao " + usernames[userIndex] + ".\nXin moi vao.");
        openDoor();
        hasPeopleMode = true;
        isDoorClosed = false;
    }
}

```

Hình 3.6: Hàm đọc thẻ người dùng thành công

```

void handleFailure() {
    displayMessage("The khong hop le.");
    errorSound();
}

```

Hình 3.7: Hàm đọc thẻ người dùng thất bại

```

void openDoor()
{
    // Mở khóa cửa
    Serial.println("Mo khoa cua");
    smoothMove(ViTriDongCua, ViTriMoCua, 15);
    //delay(2500);
    openSound();
    displayMessage("Cua da duoc mo!.");
    delay(100);
}

void lockDoor() {
    Serial.println("Dong cua");
    smoothMove(ViTriMoCua, ViTriDongCua, 15);
    //delay(2500); // khóa cửa
    closeSound();
    displayMessage("Cua da duoc khoa.");
}

```

Hình 3.8: Hàm mở cửa và đóng cửa

- Hàm thêm người dùng

```

void addUser() {
    if(addUserMode)
    {
        displayMessage("Che do them nguoi dung.");
        while(!readButtonThemNguoi())
        {
            if (mfr522.PICC_IsNewCardPresent() && mfr522.PICC_ReadCardSerial())
            {
                // if(!pressedButtonThemNguoi)
                {
                    String newUID = getCardUID();
                    int checkedUser = checkUser(newUID);
                    if(checkedUser == -1)
                    {
                        savedUIDs[savedUsers] = newUID;
                        usernames[savedUsers] = "Nguoi dung " + String(char(savedUsers - 4 + 'A'));
                        savedUsers++;

                        Serial.println("Da them: " + usernames[savedUsers - 1]);
                        displayMessage("Da them: " + usernames[savedUsers - 1]);
                        thanhCongSound();
                        delay(500);
                        displayMessage("Thoat che do them nguoi dung.");
                        addUserMode = false;
                        break;
                    }
                }
            }
        }
    }
}

```



```

        else
        {
            errorSound();
            Serial.println("Trung the: " + usernames[checkedUser] + "\nXin moi thu lai!");
            displayMessage("Trung the: " + usernames[checkedUser] + "\nXin moi thu lai!");
            // addUserMode = false;
            // break;
        }
    }
}

if(pressedButtonThemNguoi)
{
    Serial.println("Huy che do them nguoi!");
    displayMessage("Huy che do them nguoi!");
    addUserMode = false;
}

mfrc522.PICC_HaltA();
}
}

```

Hình 3.9: Hàm chức năng thêm người dùng

2.7.6. Hàm điều khiển đèn và điều hoà

- Hàm điều khiển đèn

```

int prevLightState = 0;
void dieuKienDen()
{
    if(hasPeopleMode && !isTurnOffLight)
    {
        // Kiểm tra độ sáng
        int lightLevel = digitalRead(LDR_PIN);
        // Serial.printf("Cam bien anh sang: %d\n", lightLevel);
        if (lightLevel == 1) { // trời tối
            if(prevLightState != lightLevel)
            {
                prevLightState = lightLevel;
                digitalWrite(LED_PIN, HIGH); // bật đèn
                displayMessage("Den phong bat.");
            }
        }
        else
        {
            if(prevLightState != lightLevel)
            {
                digitalWrite(LED_PIN, LOW);
                displayMessage("Den phong tat.");
            }
        }

        prevLightState = lightLevel;
    }
    else
    {
        prevLightState = 0;
        digitalWrite(LED_PIN, LOW);
    }
}

```

Hình 4.0: Hàm điều khiển đèn led

- Hàm điều khiển điều hoà

```
int prevDieuHoaState = 0;
void dieuKhienDieuHoa() {
    int currDieuHoaState = 0;
    TempAndHumidity data = dhtSensor.getTempAndHumidity();
    float nhietDo = data.temperature;
    float doAm = data.humidity;
    nhietDoGuiDi = nhietDo;
    doAmGuiDi = doAm;
    if (hasPeopleMode && !isTurnOffDieuHoa) {
        // Serial.println("Nhiet do: " + String(nhietDo, 1) + "°C");
        // Serial.println("Do am: " + String(doAm, 1) + "%");
        // Serial.printf("Cam bien anh sang: %d\n", lightLevel);
        if (nhietDo > 30 || doAm > 75) {
            currDieuHoaState = 1;
        } else {
            currDieuHoaState = 0;
        }
        if (currDieuHoaState == 1) { // trời nóng
            if (prevDieuHoaState != currDieuHoaState) {
                displayMessage("Nhiet do: " + String(nhietDo, 1) + "°C\n" + "Do am: " + String(doAm, 1) + "%");
                prevDieuHoaState = currDieuHoaState;
                digitalWrite(DieuHoa_PIN, HIGH); // bật điều hoà
                displayMessage("Dieu hoa bat.");
            }
        } else {
            if (prevDieuHoaState != currDieuHoaState) {
                displayMessage("Nhiet do: " + String(nhietDo, 1) + "°C\n" + "Do am: " + String(doAm, 1) + "%");
                digitalWrite(DieuHoa_PIN, LOW);
                displayMessage("Dieu hoa tat.");
            }
        }
        prevDieuHoaState = currDieuHoaState;
    } else {
        prevDieuHoaState = 0;
        digitalWrite(DieuHoa_PIN, LOW);
    }
}
```

Hình 4.1: Hàm điều khiển điều hoà

2.7.7. Hàm httpSync() và updateToPin(String s)

```
void httpSync() {
    unsigned long currentMillis = millis();

    // Check if 1 second has passed since the last HTTP request
    if (currentMillis - lastHttpRequestTime >= httpInterval) {
        lastHttpRequestTime = currentMillis; // Update the last request time

        // Send HTTP request
        // Serial.println("Sending HTTP request...");
        HTTPClient http;
        http.begin(server_name);
        http.addHeader("Content-type", "text/plain");

        String httpRequestData = "nhietdo=" + String(nhietDoGuiDi) + ", doAm=" + String(doAmGuiDi) + ", conNguoi=" + String(hasPeopleMode ? "true" : "false");
        int httpResponseCode = http.POST(httpRequestData);

        // if (httpResponseCode > 0) {
        //     Serial.print("Good HTTP response code: ");
        // } else {
        //     Serial.print("Bad HTTP response code: ");
        // }
        // Serial.println(httpResponseCode);

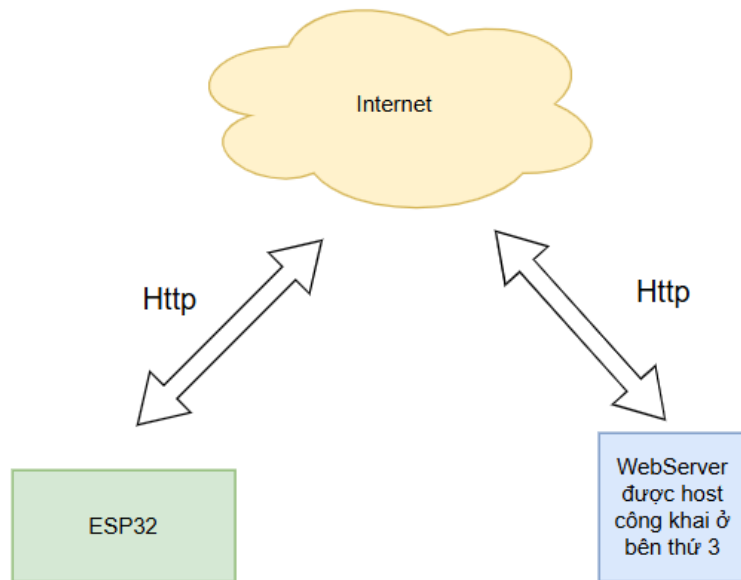
        String response = http.getString();
        // Serial.print("Response: ");
        // Serial.println(response);
        updateToPin(response);
        http.end(); // Close connection
    }
}
```

Hình 21: HttpSync()

```
void updateToPin(String s) {
    s.replace(",", " ");
    s.replace("=", " ");
    int index = 0; // Variable to track parsing position
    while (index < s.length()) {
        // Extract the next token
        int nextSpace = s.indexOf(' ', index);
        if (nextSpace == -1) nextSpace = s.length();
        String token = s.substring(index, nextSpace);
        index = nextSpace + 1;

        // Check the token and assign the corresponding value
        if (token == "den") {
            pin_control_den = s.substring(index, s.indexOf(' ', index)).toInt();
        } else if (token == "dieuhoa") {
            pin_control_dieuhoa = s.substring(index, s.indexOf(' ', index)).toInt();
        }
    }
    setPinDenAutoFromWeb();
    setPinDieuHoaAutoFromWeb();
}
```

Hình 22: Hàm Update to pin



Hình 23: Cơ chế hoạt động của WebServer

- Mỗi giây, ESP32 sẽ gửi lên WebServer thông qua Internet dựa vào giao thức kết nối HTTP các thông tin về nhiệt độ và độ ẩm.
- WebServer nhận được và trả về kết nối thành công cũng như các thay đổi của nút bấm.
- Giao diện Web sẽ thể hiện thay đổi về dữ liệu nhiệt độ và độ ẩm lên trang chủ.
- ESP32 nhận được phản hồi, phân tích chuỗi gửi về để thực thi yêu cầu tương ứng.


2.7.8. Các hàm trong code server Flask Python

2.7.8.1. Import các thư viện

```
1 from flask import Flask, request, send_file, jsonify, render_template, redirect, url_for, session, flash
2 import threading
3 import time
4 import sys
5 import logging
6 app = Flask(__name__)
7 app.secret_key = 'MONHAUVD121'
8
9 from datetime import datetime
```

- **from flask import ...:** Import các thành phần từ Flask:
 - Flask: Tạo ứng dụng web.
 - request: Xử lý yêu cầu HTTP (GET, POST...).
 - send_file: Gửi tệp như một phản hồi HTTP.
 - jsonify: Tạo phản hồi JSON.
 - render_template: Kết xuất tệp HTML.
 - redirect, url_for: Chuyển hướng và xây dựng URL.
 - session: Lưu trữ thông tin người dùng trong phiên.
 - flash: Hiển thị thông báo ngắn.
- **import threading, time, sys, logging:** Các thư viện Python chuẩn để xử lý luồng (thread), thời gian, hệ thống, và logging.
- **from datetime import datetime:** Làm việc với thời gian và ngày tháng.


2.7.8.2. Cấu hình ứng dụng Flask



```
1 app = Flask(__name__)
2 app.secret_key = 'MONHAUVD121'
```

- **app = Flask(__name__):** Tạo một ứng dụng Flask.
- **app.secret_key = 'MONHAUVD121':** Khóa bí mật để mã hóa dữ liệu trong session.


2.7.8.3. Biến toàn cục



```
1  # Global variables for pin states
2  pin_den = 0
3  pin_dieuhoa = 0
4
```

- **pin_den = 0, pin_dieuhoa = 0:** Trạng thái của các "chân" (pin) điều khiển thiết bị.

2.7.8.4. Lớp DualStream



```
1  # Open log file
2  log_file = open('server.log', 'w')
3  dual_stream = DualStream(sys.stdout, log_file)
4  sys.stdout = dual_stream
5  sys.stderr = dual_stream
```

- Mục đích: Kết hợp việc ghi log ra console và ghi vào tệp **server.log**.
- **write:** Ghi dữ liệu vào cả terminal và tệp log.
- **flush:** Đẩy dữ liệu từ bộ nhớ đệm ra thiết bị.
- Ghi đè **sys.stdout** và **sys.stderr** để mọi log sẽ được chuyển qua DualStream.

2.7.8.5. Cấu hình logging

```
1 log = logging.getLogger('werkzeug')
2 log.setLevel(logging.ERROR)
3
```

- **log.setLevel(logging.ERROR):** Giảm mức log của Flask để tránh in log không cần thiết.

2.7.8.6. Middleware (before_request và after_request)

```
1 @app.before_request
2 def before_request():
3     if request.path == '/log':
4         dual_stream.suppress_terminal = True
5
6
7 @app.after_request
8 def after_request(response):
9     dual_stream.suppress_terminal = False
10    return response
```

- **@app.before_request:** Thực thi trước mỗi yêu cầu HTTP. Tạm thời ngăn ghi log ra terminal khi truy cập /log.

- **@app.after_request:** Đặt lại trạng thái ghi log ra terminal sau mỗi yêu cầu HTTP.

2.7.8.7. Các route (đường dẫn)

- Trang chính (/)


```
1 @app.route('/')
2 def index():
3     return redirect(url_for('login'))
```

Chuyển hướng tới trang đăng nhập.

- Trang đăng nhập (/login)

```
1 @app.route('/login', methods=['GET', 'POST'])
2 def login():
3     if request.method == 'POST':
4         username = request.form['username']
5         password = request.form['password']
6         if (username == 'vietdung' and password == '123') or (username == 'theanh' and password == '456'):
7             session['username'] = username
8             return redirect(url_for('dashboard'))
9         else:
10            flash('Invalid username or password!', 'error')
11            return redirect(url_for('login'))
12    return render_template('login.html')
```

- Xử lý cả hai phương thức GET (hiển thị trang đăng nhập) và POST (xác thực người dùng).
- Kiểm tra tài khoản và mật khẩu:
 - vietdung:123
 - theanh:456
- Lưu thông tin vào session nếu thành công, nếu không, hiển thị thông báo lỗi.
- Đăng xuất




```

1 @app.route('/logout')
2 def logout():
3     session.pop('username', None)
4     return redirect(url_for('login'))

```

Xóa thông tin phiên và chuyển về trang đăng nhập.

- Kiểm tra log (/log_checking)




```

1 @app.route('/log_checking')
2 def log_checking():
3     if 'username' not in session:
4         return redirect(url_for('login'))
5     with open('server.log', 'r') as file:
6         log_content = file.read()
7     return render_template('log_checking.html', log_content=log_content)

```

Hiển thị nội dung tệp server.log nếu người dùng đã đăng nhập.

- Tải log dưới dạng tệp (/log)



```

1 @app.route('/log')
2 def log():
3     return send_file('server.log', mimetype='text/plain')

```

Gửi tệp log như văn bản thuần túy.

- Xử lý yêu cầu POST (/process)

```

1 @app.route('/process', methods=['POST'])
2 def process():
3     # Get current time
4     current_time = datetime.now()
5     # Print current time
6     current_time.strftime("%Y-%m-%d %H:%M:%S")
7     body = request.data.decode('utf-8')
8     # print("Body: ", body)
9     log_file.flush()
10    response_message = str(f"den={pin_den}, dieuhoa={pin_dieuhoa}")
11    print(current_time, " - Body: ", body, ", response_message = ", response_message)
12    try:
13        if "nhietdo" in body and "doAm" in body:
14            parts = body.split(", ")
15            nhietDo = parts[0].split("=")[1]
16            doAm = parts[1].split("=")[1]
17            coNguoi = parts[2].split("=")[1]
18            # Store the values in a global variable or a database for later use
19            app.config['nhietDo'] = nhietDo
20            app.config['doAm'] = doAm
21            app.config['coNguoi'] = coNguoi
22    except:
23        pass
24    return response_message

```

- Nhận dữ liệu cảm biến (nhiệt độ, độ ẩm, có người hay không) qua POST.
- Lưu các giá trị vào app.config để sử dụng sau.

- Bảng điều khiển (/dashboard)

```

1 # Dashboard route to display the values
2 @app.route('/dashboard')
3 def dashboard():
4     if 'username' not in session:
5         return redirect(url_for('login'))
6     return render_template('dashboard.html', pin_den=pin_den, pin_dieuhoa=pin_dieuhoa)

```

- Hiển thị trạng thái của pin_den và pin_dieuhoa.
- Yêu cầu người dùng phải đăng nhập.

- Lấy dữ liệu (/data)

```

1  @app.route('/data')
2  def data():
3      nhietDo = app.config.get('nhietDo', 'N/A')
4      doAm = app.config.get('doAm', 'N/A')
5      coNguoi = app.config.get('coNguoi', 'N/A')
6      return jsonify(nhietDo=nhietDo, doAm=doAm, coNguoi=coNguoi)
7

```

Trả về giá trị nhietDo, doAm, và coNguoi dưới dạng JSON.

- Thay đổi trạng thái pin (/toggle_pin)


```

1  # Endpoint to toggle pin states
2  @app.route('/toggle_pin', methods=['POST'])
3  def toggle_pin():
4      global pin_den, pin_dieuhoa
5      data = request.json
6      pin = data.get('pin')
7      if pin == 'pin_den':
8          pin_den = 1 - pin_den
9      elif pin == 'pin_dieuhoa':
10         pin_dieuhoa = 1 - pin_dieuhoa
11     return jsonify({pin: eval(pin)})
12
13

```

Nhận yêu cầu JSON để thay đổi trạng thái của pin_den hoặc pin_dieuhoa.

2.8.8.8. Chạy ứng dụng



```
1 if __name__ == "__main__":
2     # Start a logging thread
3     # logging_thread = threading.Thread(target=log_writer, daemon=True)
4     # logging_thread.start()
5
6     # Run the Flask server
7     app.run(host='192.168.0.102', port=8080)
```

Chạy ứng dụng Flask tại địa chỉ 192.168.0.102 trên cổng 8080.

CHƯƠNG 3. KẾT QUẢ THỰC NGHIỆM

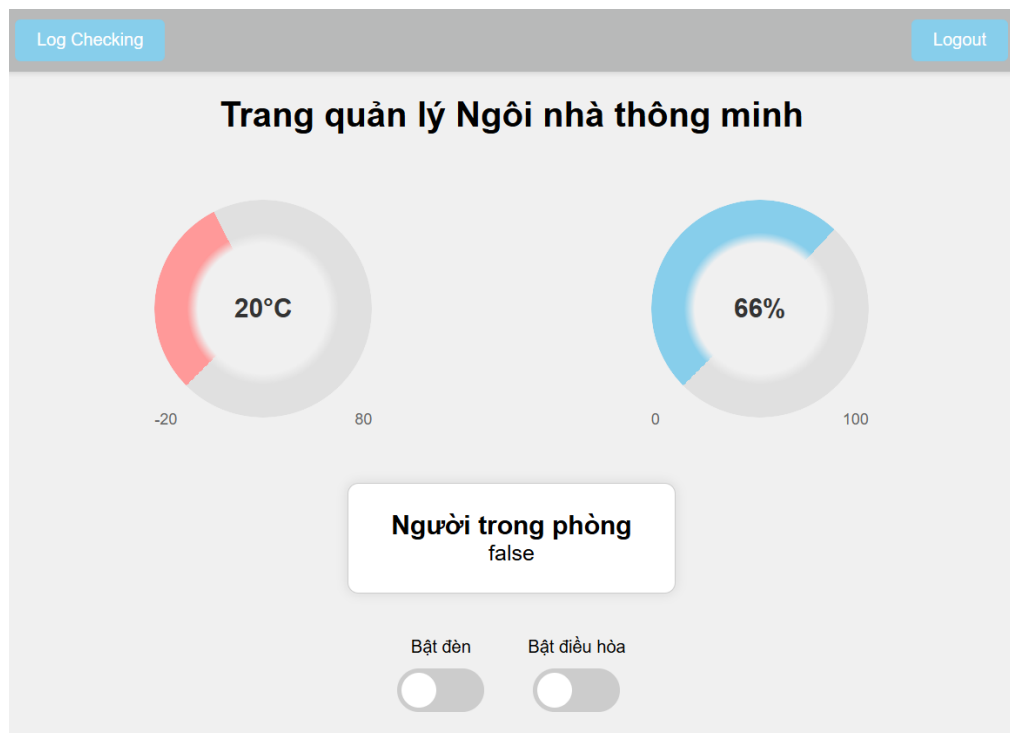
3.1. Kiểm tra

- Giao diện App:



Hình 24. Giao diện app Blynk

- Giao diện Webserver:



Hình 25. Giao diện web tự host

- Nhận xét chung:

Trong quá trình sử dụng mô hình, nhóm đã thao tác và thực hiện ghi lại dữ liệu thống kê trong bảng dưới:

Thao tác	Số lần thao tác	Thời gian phản hồi
Kiểm tra cảm biến ánh sáng	20	1 giây
Kiểm tra cảm biến DHT	20	1 giây
Kiểm tra cảm biến NFC	50	1 giây
Điều khiển đèn qua app	50	1.5 giây

Điều khiển qua cảm biến hồng ngoại	50	1 giây
Điều khiển qua WebServer	50	2 giây

Bảng 2. Dữ liệu phản hồi của hệ thống

- Đánh giá:
Mô hình đơn giản, chi phí thấp, dễ sử dụng.
Tốc độ phản hồi hệ thống nhanh.
Giao diện ứng dụng thân thiện dễ dùng.
Hệ thống có thể phát triển thêm một vài chức năng.

3.2. Kết quả đạt được

Về phần cứng

- Tìm hiểu về ESP32, các module và cách thiết kế lắp đặt mạch hoàn chỉnh và vận hành ổn định...
- Có thêm kiến thức về các linh kiện điện tử, cách thiết kế mạch trên phần mềm và lắp đặt thực tế

Về phần mềm

- Tìm hiểu về Blynk để lưu trữ dữ liệu từ phần cứng.
- Thiết kế mạch điện trên phần mềm
- Viết chương trình cho hệ thống dựa trên phần mềm Arduino IDE..
- Thiết kế phần mềm di động điều khiển và kết nối dữ liệu với web.

3.3. Hướng phát triển

1. Tích hợp cảm biến môi trường: Sử dụng các cảm biến khí gas, và cảm biến khói để giám sát môi trường bên trong nhà. Dữ liệu từ các cảm biến này có thể giúp tự động điều chỉnh điều hòa hoặc đưa ra cảnh báo khi phát hiện các chỉ số bất thường (ví dụ: khi có rò rỉ khí gas hoặc khói).

2. Điều khiển bằng giọng nói: Tích hợp công nghệ nhận diện giọng nói như Google Assistant hoặc Alexa để người dùng có thể điều khiển các thiết bị trong nhà thông qua giọng nói. Điều này giúp tăng tính tiện lợi và trải nghiệm người dùng.

3. Lên lịch tự động: Cung cấp tính năng lập lịch để tự động bật/tắt đèn, điều hòa, và các thiết bị khác theo thời gian thiết lập sẵn, ví dụ: tự động tắt điều hòa vào ban đêm hoặc bật đèn trước khi chủ nhà về.

4. Hệ thống an ninh mở rộng: Kết hợp với camera IP và cảm biến chuyển động để tăng cường an ninh. Khi có người lạ xuất hiện hoặc có chuyển động bất thường, hệ thống sẽ thông báo cho chủ nhà qua app và có thể chụp ảnh hoặc quay video.

5. Cảnh báo từ xa và thông báo khẩn cấp: Tích hợp hệ thống cảnh báo để gửi thông báo đến điện thoại của chủ nhà trong trường hợp có vấn đề khẩn cấp, như trộm đột nhập hoặc rò rỉ khí gas. Hệ thống có thể kết nối với các dịch vụ khẩn cấp để phản ứng nhanh.

6. Điều khiển từ xa qua internet: Hiện tại, đang chỉ sử dụng app để bật/tắt đèn và điều hòa, nhưng có thể mở rộng điều này với nhiều thiết bị khác trong nhà và điều khiển từ bất kỳ đâu có kết nối internet.

7. Chế độ tiết kiệm và tối ưu hóa tự động: Khi không có người ở nhà, hệ thống sẽ tự động chuyển sang chế độ tiết kiệm năng lượng, tắt tất cả các thiết bị không cần thiết và giữ các thiết bị an ninh trong trạng thái bật.

3.4. Kết luận

- Mô hình IoT cho hệ thống nhà thông minh sử dụng NFC và các cảm biến trong nhà đã thể hiện tiềm năng rõ rệt trong việc cải thiện sự tiện nghi và tối ưu hóa năng lượng.
- Hệ thống mở cửa bằng NFC đảm bảo an ninh và dễ dàng quản lý, trong khi ứng dụng điều khiển đèn và điều hòa mang lại khả năng kiểm soát linh hoạt từ xa, giúp người dùng chủ động trong việc thiết lập môi trường sống.
- Tính năng tự động tắt đèn khi trời sáng thông qua cảm biến ánh sáng không chỉ tiết kiệm năng lượng mà còn bảo vệ môi trường, giảm thiểu tiêu thụ điện năng không cần thiết.
- Mô hình này có thể mở rộng thêm với nhiều cảm biến khác để tăng cường trải nghiệm và tự động hóa hơn nữa.
- Hệ thống là một minh chứng cho thấy việc áp dụng IoT vào cuộc sống hàng ngày có thể mang lại lợi ích đáng kể, giúp tạo ra một môi trường sống thông minh, an toàn và tiết kiệm.
- Việc phát triển webserver cho hệ thống thuận tiện hơn cho việc điều khiển hệ thống từ xa. Và hỗ trợ nhiều người sử dụng hệ thống.

Tài liệu tham khảo

- [1] “Bài giảng Phát triển hệ thống và ứng dụng IOT” - khoa Viễn thông - Học viện Công nghệ Bưu chính Viễn thông.
- [2] <https://www.espressif.com/en/products/socs/esp32>
- [3] [DOIT ESP32 DevKit-v1 30P/README.md at main · TronixLab/DOIT ESP32 DevKit-v1 30P \(github.com\)](#)