**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG ĐIỆN – ĐIỆN TỬ**

**ĐỒ ÁN TỐT NGHIỆP**

**Thiết kế hệ thống nhà thông minh**

**NGUYỄN MẠNH DŨNG**

[Dung.NM213838@sis.hust.edu.vn](mailto:Dung.NM213838@sis.hust.edu.vn)

**Ngành Kỹ thuật điện tử viễn thông**

**Chuyên ngành Kỹ thuật máy tính**

**Giảng viên hướng dẫn**: TS. Trần Ngọc Tuấn

Chữ ký của GVHD

**Khoa**: Khoa Kỹ thuật truyền thông

**HÀ NỘI, 6/2025**

**ĐỀ TÀI TỐT NGHIỆP**

**Thiết kế hệ thống nhà thông minh**

Giảng viên hướng dẫn

Ký và ghi rõ họ tên

**Lời cảm ơn**

Trước tiên, em xin bày tỏ lòng biết ơn sâu sắc đến TS. Trần Ngọc Tuấn, giảng viên hướng dẫn của em trong suốt quá trình thực hiện đồ án tốt nghiệp này. Thầy đã dành nhiều thời gian để định hướng, truyền đạt những kiến thức chuyên môn quan trọng cũng như những kinh nghiệm thực tế giúp em có thể hoàn thiện đề tài của mình

Bên cạnh đó, em xin gửi lời cảm ơn chân thành đến các thầy cô trong Trường Đại học Bách Khoa đã tận tâm giảng dạy, truyền đạt kiến thức và kỹ năng cho em trong suốt những năm học tập tại trường.

Em cũng xin gửi lời tri ân đến nhà trường, ban giám hiệu và toàn thể cán bộ giảng viên của Trường Đại học Bách Khoa, nơi đã tạo điều kiện thuận lợi về cơ sở vật chất, tài liệu và môi trường học tập lý tưởng để em có thể nghiên cứu, thực hiện đề tài một cách hiệu quả.

**Tóm tắt nội dung đồ án**

Đồ án Tốt nghiệp này tập trung xây dựng và phát triển hệ thống nhà thông minh với khả năng tự động hoá và điều khiển từ xa nhằm năng cao tính tiện dụng và tăng cường tính an toàn cho người sử dụng. Hiện nay, khi xã hội phát triển, nhu cầu về nhà thông minh ngày càng tăng, mục tiêu của Đồ án là thiết kế một mô hình nhà thông minh có tính ứng dụng cao, có thể dễ dàng triển khai và sử dụng.

Hệ thống được thiết kế gồm bốn thành phần chính: *Ứng dụng điều khiển*(Android): Được phát triển bằng Kotlin với Jetpack Compose để xây dựng giao diện. *Phần cứng*: Sử dụng mạch ESP32 làm gateway, kết hợp với các cảm biến khí gas, tiệm cận, nhiệt độ, độ ẩm, …. để thu thập dữ liệu và điều khiển thiết bị điện. Phần firmware lập trình bằng C++ trên PlatformIO, áp dụng OOP giúp dễ dàng mở rộng thiết bị. *Điều khiển giọng nói:* Sử dụng mạch ESP32 cùng với Microphone thu âm thanh để xử lý đưa về dạng văn bản rồi từ đó điều khiển thiết bị dựa trên yêu cầu của văn bản. *Cloud* Backend: Sử dụng Firebase Realtime Database để đồng bộ lệnh điều khiển giữa app và thiết bị, cùng với Firebase Authentication để hỗ trợ tính năng đăng nhập và quản lý người dùng.

**MỤC LỤC**

[CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI 1](#_Toc198409830)

[**1.1** **Giới thiệu về nhà thông minh** 1](#_Toc198409831)

[**1.2** **Các thành phần cơ bản của hệ thống nhà thông minh** 1](#_Toc198409832)

[**1.3** **Các lĩnh vực ứng dụng** 1](#_Toc198409833)

[**1.4** **Ưu nhược điểm của smarthome** 2](#_Toc198409834)

[**1.5** **Tình hình phát triển hiện nay** 3](#_Toc198409835)

[**1.6** **Các thách thức khi triển khai hệ thống** 3](#_Toc198409836)

[**1.7** **Yêu cầu chức năng của đề tài** 4](#_Toc198409837)

[**1.8** **Yêu cầu phi chức năng của đề tài** 4](#_Toc198409838)

[CHƯƠNG 2: NỀN TẢNG CÔNG NGHỆ VÀ CÁC THÀNH PHẦN HỆ THỐNG 6](#_Toc198409839)

[2.1 Lựa chọn linh kiện và thiết bị 6](#_Toc198409840)

[2.2 Giao thức truyền dữ liệu và truyền thông 21](#_Toc198409841)

[2.3 Nền tảng Cloud được sử dụng 21](#_Toc198409842)

[2.4 Ngôn ngữ lập trình, IDE và quản lý mã nguồn 22](#_Toc198409843)

[2.4.1 Lập trình phần cứng 22](#_Toc198409844)

[2.4.2 Lập trình ứng dụng Android 24](#_Toc198409845)

[2.4.3 Quản lý mã nguồn với Git và GitHub 26](#_Toc198409846)

[**CHƯƠNG 3: THIẾT KẾ TỔNG QUAN HỆ THỐNG VÀ ĐIỀU KHIỂN GIÁM SÁT** 28](#_Toc198409847)

[**3.1 Sơ đồ cấu trúc hệ thống** 28](#_Toc198409848)

[**3.2 Nguyên lý điều khiển và truyền nhận dữ liệu** 28](#_Toc198409849)

[**3.3 Tích hợp và quản lý dữ liệu trên Cloud** 29](#_Toc198409850)

[**3.3.1 Quản lý đăng nhập, đăng ký và xác thực người dùng** 29](#_Toc198409851)

[**3.3.2 Dữ liệu điều khiển và dữ liệu hệ thống** 31](#_Toc198409852)

[**3.4 Thiết kế ứng dụng điều khiển** 32](#_Toc198409853)

[**3.4.1 Giao diện ứng dụng điều khiển** 32](#_Toc198409854)

[**3.4.2 Chức năng điều khiển và quản lý trên ứng dụng** 35](#_Toc198409855)

[CHƯƠNG 4: THIẾT KẾ PHẦN CỨNG VÀ LẬP TRÌNH NHÚNG HỆ THỐNG NHÀ THÔNG MINH 40](#_Toc198409856)

[**4.1** **Thiết kế phần cứng hệ thống nhà thông minh** 40](#_Toc198409857)

[**4.2** **Lập trình phần cứng cho hệ thống** 41](#_Toc198409858)

[**4.2.1** **Tổng quan về luồng hoạt động** 41](#_Toc198409859)

[**4.2.2** **Thiết lập môi trường lập trình và các thành phần cơ bản** 43](#_Toc198409860)

[**4.2.3** **Lập trình giao tiếp với Firebase** 44](#_Toc198409861)

[**4.2.4** **Thiết kế phần mềm nhúng theo hướng đối tượng** 46](#_Toc198409862)

[**4.2.5** **Lập trình thu thập dữ liệu từ cảm biến và nút nhấn** 50](#_Toc198409863)

[**4.2.6** **Lập trình điều khiển thiết bị (Actuator Unit)** 53](#_Toc198409864)

[**CHƯƠNG 5: ĐIỀU KHIỂN THIẾT BỊ BẰNG GIỌNG NÓI** 56](#_Toc198409865)

[**5.1 Tổng quan về điều khiển bằng giọng nói** 56](#_Toc198409866)

[**5.2 Cấu hình phần cứng cho điều khiển bằng giọng nói** 58](#_Toc198409867)

[**5.2.1 Giới thiệu về giao thức I2S** 58](#_Toc198409868)

[**5.2.2 Micro I2S INMP441 kết nối với ESP32.** 59](#_Toc198409869)

[**5.3** **Lập trình thu âm và xử lý tín hiệu âm thanh** 59](#_Toc198409870)

[**5.4** **Gửi dữ liệu âm thanh lên Wit.ai và nhận phản hồi** 60](#_Toc198409871)

[**5.4.1 Giới thiệu về Wit.ai** 60](#_Toc198409872)

[**5.4.2** **Cấu hình Wit.ai:** 62](#_Toc198409873)

[**5.4.3 Gửi dữ liệu âm thanh lên Wit.ai:** 63](#_Toc198409874)

[**5.4.4** **Nhận phản hồi từ Wit.ai:** 63](#_Toc198409875)

[**5.5 Phân tích phản hồi và cập nhật trạng thái** 64](#_Toc198409876)

[**5.5.1** **Phân tích phản hồi** 64](#_Toc198409877)

[**5.5.2** **Cập nhật trạng thái điều khiển** 65](#_Toc198409878)

[**5.6 Kết quả** 65](#_Toc198409879)

[CHƯƠNG 6: XÂY DỰNG VÀ KIỂM NGHIỆM MÔ HÌNH THỰC TẾ 67](#_Toc198409880)

[**6.1 Mô hình thực tế** 67](#_Toc198409881)

[**6.2 Kết quả vận hành hệ thống và đánh giá hiệu quả hệ thống** 68](#_Toc198409882)

[CHƯƠNG 7: KẾT LUẬN 71](#_Toc198409883)

[TÀI LIỆU THAM KHẢO 72](#_Toc198409884)

**DANH MỤC HÌNH VẼ**

No table of figures entries found.

CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

* 1. **Giới thiệu về nhà thông minh**

- Nhà thông minh là mô hình nhà được tích hợp và trang bị những công nghệ và thiết bị điện tử hiện đại, với khả năng kết nối tự động, tự động hoá và điều khiển thông qua Internet hoặc các công nghệ không dây. Người dùng có thể điều khiển và quản lý các thiết bị trong nhà như đèn, điều hoà, hệ thống an ninh, ... chỉ với điện thoại thông minh, máy tính bảng hoặc máy tính.

- Ý tưởng về nhà thông minh xuất hiện từ những năm 70 của thế kỷ trước nhưng gần đây mới bùng nổ nhờ sự phát triển của Internet và IoT. Ngày nay, nhà thông minh không chỉ đem đến sự tiện lợi mà còn tăng cường an ninh, cải thiện chất lượng cuộc sống đối với mọi người.

* 1. **Các thành phần cơ bản của hệ thống nhà thông minh**
* *Bộ điều khiển trung tâm*: Là thiết bị kết nối tất cả các thiết bị thông minh trong nhà, cho phép chúng trao đổi dữ liệu và thực hiện các lệnh điều khiển. Bộ điều khiển có thể là một thiết bị chuyên dụng hoặc được tích hợp trong các trợ lý ảo như Google Home, Amazon Echo.
* *Thiết bị thông minh*: Bao gồm cảm biến (cửa, nhiệt độ, chuyển động), công tắc điện tử, đèn thông minh, camera giám sát, điều hòa, khóa cửa, ... Các thiết bị này có khả năng giao tiếp với bộ điều khiển hoặc với nhau để thực hiện các tác vụ tự động.
* *Ứng dụng quản lý*: Là phần mềm trên điện thoại thông minh, máy tính bảng hoặc web, giúp người dùng giám sát, cấu hình và điều khiển hệ thống nhà từ bất cứ đâu. Ứng dụng cũng có thể hỗ trợ điều khiển bằng giọng nói hoặc thiết lập các kịch bản tự động.
  1. **Các lĩnh vực ứng dụng**
* *Điều khiển chiếu sáng*: Hệ thống đèn có thể tự động bật/tắt, điều chỉnh độ sáng dựa trên thời gian hoặc sự hiện diện.
* *Quản lý an ninh và giám sát*: Cảm biến chuyển động, khóa cửa thông minh, hệ thống camera giám sát giúp đảm bảo an toàn cho ngôi nhà.
* Điều khiển nhiệt độ và điều hòa không khí: Hệ thống HVAC tự động điều chỉnh theo nhiệt độ môi trường và thói quen người dùng.
* *Quản lý năng lượng*: Giúp tối ưu hóa mức tiêu thụ điện, cảnh báo khi thiết bị tiêu thụ quá mức.
* *Dịch vụ tự động hóa khác*: Ví dụ như tưới nước tự động cho vườn, đóng mở rèm cửa tự động, hệ thống giải trí đa phòng, ...
  1. **Ưu nhược điểm của smarthome**
     1. **Ưu điểm**
* *Tiện lợi*: Hệ thống nhà thông minh cho phép người dùng điều khiển các thiết bị trong nhà chỉ bằng vài thao tác đơn giản trên điện thoại, máy tính bảng, hoặc thậm chí qua giọng nói. Người dùng không cần phải di chuyển đến từng thiết bị để bật/tắt hoặc điều chỉnh, mà có thể thực hiện tất cả từ xa, mang lại sự tiện lợi và tiết kiệm thời gian.
* *Tiết kiệm năng lượng*: Các thiết bị trong hệ thống smart home có khả năng tự động điều chỉnh theo nhu cầu sử dụng và môi trường, giúp tiết kiệm điện năng. Ví dụ, đèn có thể tự tắt khi không có người trong phòng, hoặc điều hòa sẽ tự động điều chỉnh nhiệt độ khi người dùng ra ngoài. Điều này không chỉ giúp giảm hóa đơn điện mà còn bảo vệ môi trường bằng cách giảm lượng năng lượng tiêu thụ.
* *An toàn*: Hệ thống nhà thông minh có thể cung cấp các tính năng bảo mật tiên tiến như camera giám sát, cảm biến chuyển động, cảm biến khói và khí gas, giúp bảo vệ an toàn cho ngôi nhà. Những cảm biến này có thể phát hiện kịp thời các nguy cơ như trộm cắp, cháy nổ, hoặc rò rỉ khí, và gửi cảnh báo trực tiếp đến điện thoại của người dùng. Điều này giúp chủ nhà phản ứng nhanh chóng và có biện pháp xử lý kịp thời.
  + 1. **Nhược điểm**
* *Chi phí cao*: Việc lắp đặt một hệ thống nhà thông minh hoàn chỉnh thường yêu cầu một khoản đầu tư ban đầu khá lớn. Chi phí mua các thiết bị như đèn thông minh, cảm biến, khóa cửa thông minh và các thiết bị khác có thể cao hơn so với các thiết bị truyền thống. Bên cạnh đó, chi phí duy trì và bảo trì hệ thống cũng có thể phát sinh theo thời gian.
* *Vấn đề bảo mật*: Mặc dù các hệ thống nhà thông minh mang lại nhiều lợi ích, nhưng chúng cũng có thể đối mặt với các nguy cơ bảo mật. Nếu không được bảo vệ đúng cách, hệ thống có thể bị tấn công bởi hacker, dẫn đến việc rò rỉ dữ liệu cá nhân hoặc thậm chí xâm nhập vào các thiết bị trong nhà. Để đảm bảo an toàn, người dùng cần thường xuyên cập nhật phần mềm và sử dụng các biện pháp bảo mật như mật khẩu mạnh hoặc mã hóa.
* *Sự phụ thuộc vào Internet*: Hệ thống nhà thông minh hầu hết hoạt động dựa vào kết nối Internet. Khi mạng Internet gián đoạn hoặc mất kết nối, các chức năng điều khiển từ xa có thể bị ảnh hưởng. Điều này có thể làm gián đoạn hoạt động của các thiết bị, chẳng hạn như không thể bật/tắt đèn hoặc điều chỉnh nhiệt độ máy điều hòa từ xa. Hệ thống cũng có thể hoạt động kém hiệu quả nếu không có mạng ổn định.
  1. **Tình hình phát triển hiện nay**
* *Trên thế giới:*Các quốc gia phát triển như Mỹ, Nhật Bản, Hàn Quốc đang đi đầu trong việc ứng dụng và phổ cập nhà thông minh. Các hãng lớn như Google (Google Nest), Amazon (Alexa), Apple (HomeKit) liên tục tung ra các sản phẩm mới.
* *Tại Việt Nam:*Xu hướng nhà thông minh đang phát triển nhanh, với sự tham gia của nhiều công ty trong nước như BKAV (SmartHome), Lumi, ACIS, và các sản phẩm nhập khẩu từ Xiaomi, Samsung... Tuy nhiên, độ phổ biến vẫn chưa đồng đều do hạn chế về chi phí và nhận thức người dùng.
  1. **Các thách thức khi triển khai hệ thống**
* *Bảo mật và quyền riêng tư:* Các thiết bị kết nối Internet luôn tiềm ẩn nguy cơ bị hack, xâm nhập dữ liệu cá nhân.
* *Khả năng tương thích:* Các thiết bị đến từ nhiều nhà sản xuất khác nhau đôi khi khó đồng bộ và kết nối với nhau.
* *Chi phí triển khai:* Đối với nhiều hộ gia đình, chi phí đầu tư cho một hệ thống hoàn chỉnh vẫn còn là rào cản lớn.
* *Khả năng sử dụng:* Không phải ai cũng dễ dàng sử dụng hệ thống thông minh, đặc biệt là người lớn tuổi.
  1. **Yêu cầu chức năng của đề tài**
* *Điều khiển các thiết bị:* Cho phép người dùng có thể bật/ tắt các thiết bị như đèn, quạt, điều hoà, … thông qua nút bấm vật lý hoặc điều khiển thông qua ứng dụng trên điện thoại hoặc module nhận diện giọng nói.
* *Điều khiển led tự động khi có người:* Cho phép hệ thống đèn có thể tự động bật/ tắt khi có/ không có sự hiện diện của người dùng ở khu vực.
* *Cảnh báo phát hiện khí gas và gửi thông báo:* Khi có gas sẽ gửi thông báo đến người dùng đồng thời sẽ bật quạt thông khí ở phòng bếp.
* *Hệ thống khoá cửa thông minh:* Cho phép mở cửa thông qua mật khẩu, thẻ hoặc ứng dụng trên điện thoại.
* *Quản lý điều hoà nhiệt độ:* Cho phép điều chỉnh, bật/tắt và theo dõi nhiệt độ phòng thông qua điện thoại.
  1. **Yêu cầu phi chức năng của đề tài**

Bên cạnh yêu cầu chức năng, hệ thống cần đảm bảo các yêu cầu phi chức năng để đảm bảo sự ổn định, hiệu quả và dễ sử dụng

* *Hiệu năng*: Thời gian phản hồi cần phải nhanh khi nhận lệnh điều khiển qua Internet hoặc nút bấm, đảm bảo sự ổn định và liên tục
* *Khả năng mở rộng:* hệ thống cần có khả năng mở rộng dễ dàng, thêm/ sửa/ xoá các thiết bị một cách dễ dàng
* *Thân thiện với người dùng*: Giao diện ứng dụng cần đơn giản, dễ sử dụng với mọi lứa tuổi
* *Bảo mật:* Đảm bảo những thông tin người dùng, hệ thống không thể bị truy cập trái phép

CHƯƠNG 2: NỀN TẢNG CÔNG NGHỆ VÀ CÁC THÀNH PHẦN HỆ THỐNG

2.1 Lựa chọn linh kiện và thiết bị

2.1.1 Module ESP32



Hình 0.1

ESP32 là một dòng vi điều khiển tích hợp khả năng kết nối Wi-Fi và Bluetooth, được phát triển bởi Espressif Systems, một công ty công nghệ có trụ sở tại Thâm Quyến, Trung Quốc. ESP32 được ra mắt nhằm kế thừa và mở rộng những ưu điểm của người tiền nhiệm ESP8266, với hiệu năng mạnh mẽ hơn, khả năng kết nối đa dạng hơn và tích hợp nhiều tính năng hỗ trợ các ứng dụng Internet of Things (IoT), hệ thống nhúng và tự động hóa.

Về cấu trúc phần cứng, ESP32 sử dụng bộ vi xử lý Xtensa® dual-core 32-bit LX6, có thể hoạt động ở tốc độ lên tới 240MHz, cùng với bộ nhớ RAM tích hợp và bộ nhớ Flash mở rộng. Điều này giúp ESP32 xử lý các tác vụ yêu cầu tốc độ cao, đồng thời tiết kiệm năng lượng cho các ứng dụng cần hoạt động liên tục.

Một trong những điểm mạnh của ESP32 là khả năng hỗ trợ đa giao tiếp:

* Wi-Fi chuẩn 802.11 b/g/n cho phép thiết bị dễ dàng kết nối Internet hoặc xây dựng mạng nội bộ.
* Bluetooth 4.2 và Bluetooth Low Energy (BLE) phục vụ cho các ứng dụng yêu cầu giao tiếp không dây ở khoảng cách ngắn với mức tiêu thụ điện thấp.
* Ngoài ra, ESP32 còn hỗ trợ các giao thức truyền thông phổ biến như UART (Universal Asynchronous Receiver-Transmitter), SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), PWM (Pulse Width Modulation), ADC (Analog to Digital Converter), và DAC (Digital to Analog Converter), giúp nó dễ dàng tương tác với các cảm biến, module ngoại vi, và các thiết bị điện tử khác.

***Thông số kỹ thuật***

|  |  |
| --- | --- |
| **Thành phần** | **Thông số kỹ thuật** |
| Vi điều khiển | Tensilica Xtensa® Dual-Core 32-bit LX6 |
| Điện áp hoạt động | 3.0V – 3.6V (chuẩn 3.3V) |
| Điện áp đầu vào | 5V (qua chân VIN) hoặc 3.3V (trực tiếp) |
| Dòng điện tiêu thụ | Trung bình 80 – 260 mA |
| Số chân Digital I/O | 34 (GPIO 0–39, một số chân có chức năng đặc biệt) |
| Số chân Analog Input | 18 (ADC độ phân giải 12-bit) |
| PWM Output | Hỗ trợ trên nhiều chân GPIO |
| Bộ nhớ Flash | Tùy module, thường 4 MB (SPI Flash ngoài) |
| Bộ nhớ SRAM | 520 KB |
| Tốc độ xung nhịp | Lên đến 240 MHz |
| Giao tiếp | UART, SPI, I2C, CAN, Ethernet MAC, IR, PWM, ADC, DAC |
| Kết nối không dây | Wi-Fi 802.11 b/g/n, Bluetooth v4.2 BR/EDR và BLE |
| Kích thước | Tùy phiên bản (ví dụ ESP32 DevKit v1: ~ 51mm x 25.5mm) |

Các phiên bản phổ biến

* ESP32-WROOM-32: Phiên bản tiêu chuẩn, tích hợp Wi-Fi và Bluetooth Classic/BLE, phổ biến nhất trong các dự án IoT hiện nay.
* ESP32-WROVER: Phiên bản mở rộng với bộ nhớ PSRAM tích hợp, phù hợp cho các ứng dụng yêu cầu xử lý dữ liệu lớn như camera, AI.
* NodeMCU-32S: Phiên bản phát triển tích hợp USB-to-Serial, dễ dàng lập trình thông qua cáp USB, rất tiện cho các dự án DIY.
* ESP32-S2, ESP32-C3, ESP32-S3: Các phiên bản cải tiến hỗ trợ thêm AI acceleration, bảo mật phần cứng, BLE 5.0 và tiết kiệm năng lượng.

Tính năng đặc biệt

* Khả năng kết nối Wi-Fi và Bluetooth: ESP32 hỗ trợ đồng thời Wi-Fi (802.11 b/g/n) và Bluetooth Classic/BLE, cho phép giao tiếp không dây linh hoạt.
* Công suất xử lý mạnh mẽ: Trang bị bộ vi xử lý lõi kép 32-bit với tốc độ lên tới 240 MHz, đáp ứng các tác vụ tính toán phức tạp.
* Giao tiếp đa dạng: Hỗ trợ nhiều giao thức như UART, SPI, I2C, PWM, ADC, DAC, CAN, Ethernet MAC, giúp dễ dàng kết nối với cảm biến và thiết bị ngoại vi.
* Khả năng tiết kiệm năng lượng: Hỗ trợ nhiều chế độ ngủ (Deep Sleep, Light Sleep), tối ưu tiêu thụ điện năng cho các ứng dụng IoT lâu dài.
* Bảo mật cao: Tích hợp phần cứng mã hóa AES, RSA, SHA-2, và quản lý khóa bảo mật.

Ứng dụng phổ biến

* Nhà thông minh (Smart Home): Điều khiển thiết bị điện tử, đèn, máy lạnh từ xa qua Wi-Fi/Bluetooth.
* Thiết bị IoT: Các hệ thống giám sát môi trường, nông nghiệp thông minh, đo lường từ xa.
* Robot và drone: Dùng trong robot tự hành, truyền nhận dữ liệu từ cảm biến, điều khiển động cơ.
* Thiết bị đeo thông minh (Wearable Devices): Đồng hồ thông minh, vòng đeo tay theo dõi sức khỏe.
* Hệ thống an ninh: Ứng dụng trong camera IP (ESP32-CAM), giám sát hình ảnh thời gian thực.

Lập trình phát triển

* Arduino IDE: ESP32 hỗ trợ lập trình dễ dàng bằng Arduino IDE với rất nhiều thư viện có sẵn.
* Espressif IDF (ESP-IDF): Bộ công cụ phát triển chính thức do Espressif cung cấp, cho phép lập trình chuyên sâu.
* MicroPython và Lua: Hỗ trợ lập trình bằng ngôn ngữ MicroPython hoặc Lua cho các ứng dụng nhanh gọn, đơn giản.
* PlatformIO: Một môi trường phát triển hiện đại hỗ trợ nhiều board và nhiều nền tảng lập trình.

Ưu điểm và nhược điểm

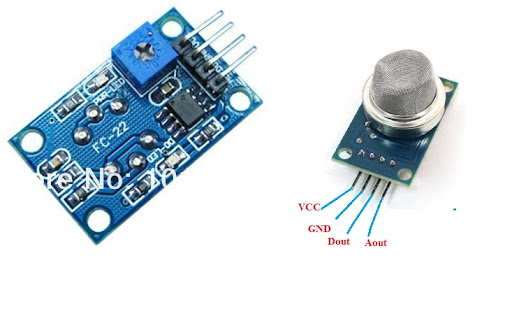
* Ưu điểm:
  + Giá thành hợp lý: ESP32 có mức giá rất phù hợp so với hiệu năng mạnh mẽ mà nó mang lại.
  + Kết nối mạnh mẽ: Hỗ trợ cả Wi-Fi chuẩn 802.11 b/g/n và Bluetooth v4.2 (BLE và Classic), cho phép nhiều chế độ giao tiếp.
  + Hiệu năng cao: Sử dụng bộ xử lý lõi kép (dual-core) với tốc độ lên đến 240MHz, phù hợp cho các ứng dụng yêu cầu xử lý nhanh.
  + Đa dạng giao tiếp: Hỗ trợ nhiều giao thức như UART, SPI, I2C, PWM, ADC, DAC, CAN, Ethernet, I2S,… rất linh hoạt khi kết nối với cảm biến và thiết bị ngoại vi.
  + Bộ nhớ lớn hơn: Flash và RAM lớn hơn nhiều so với các module như ESP8266, thuận lợi cho các ứng dụng phức tạp.
* Nhược điểm:
  + Tiêu thụ điện năng cao: So với ESP8266 hay các vi điều khiển siêu tiết kiệm điện khác, ESP32 tiêu thụ nhiều năng lượng hơn, đặc biệt ở chế độ hoạt động liên tục.
  + Lập trình phức tạp hơn: Do số lượng tính năng và giao thức nhiều, lập trình ESP32 có thể phức tạp hơn cho người mới bắt đầu.
  + Độ nhạy Wi-Fi giảm nếu thiết kế anten không tốt: Cần lưu ý bố trí phần cứng đúng chuẩn để đạt hiệu suất kết nối tối ưu.

Tài nguyên và cộng đồng hỗ trợ

* Tài liệu chính thức: Espressif Systems cung cấp đầy đủ datasheet, technical reference manual, tài liệu hướng dẫn phát triển phần mềm (ESP-IDF).
* Cộng đồng phát triển lớn mạnh: Nhiều diễn đàn (như ESP32 Forum, Reddit, GitHub) với hàng ngàn dự án mẫu, thư viện mở, hỗ trợ người dùng ở mọi cấp độ.
* Hỗ trợ nhiều môi trường lập trình: Arduino IDE, PlatformIO, ESP-IDF, MicroPython, Lua, ... đều có bộ thư viện, tài liệu hướng dẫn cài đặt và ví dụ minh họa.
* Kho tài nguyên mã nguồn mở: Rất nhiều dự án mẫu từ cơ bản tới nâng cao có sẵn trên GitHub, giúp rút ngắn thời gian phát triển ứng dụng.

2.1.2 Các loại cảm biến và linh kiện khác

\* Cảm Biến Khí Gas (LPG/CO/CH4) MQ-2



Cảm biến khí gas MQ-2 sử dụng phần tử bán dẫn SnO₂ (thiếc dioxide), có đặc tính độ dẫn điện thấp trong môi trường không khí sạch. Khi phát hiện khí dễ cháy như LPG, methane (CH₄), propane hoặc hydrogen tồn tại trong môi trường, độ dẫn điện của SnO₂ tăng lên đáng kể. Nồng độ khí càng cao thì độ dẫn điện càng lớn. Tín hiệu điện tương ứng sẽ được xuất ra, giúp xác định mức độ hiện diện của các khí cháy trong không khí.

Cảm biến MQ-2 nổi bật với độ nhạy cao, chi phí thấp, dễ sử dụng và được ứng dụng rộng rãi trong các hệ thống cảnh báo rò rỉ khí gas, phát hiện cháy nổ, hệ thống an toàn thông minh.

*- Cấu tạo và chân kết nối*

* Vỏ bảo vệ: Cảm biến được bao bọc bởi một lớp lưới kim loại giúp bảo vệ phần tử nhạy bên trong khỏi bụi bẩn và va chạm cơ học.
* Phần tử cảm biến: Là lớp SnO₂ được đốt nóng để phát hiện sự thay đổi điện trở khi tiếp xúc với khí gas.
* Heater Coil: Cuộn dây đốt nóng giúp duy trì nhiệt độ cần thiết cho phần tử SnO₂ hoạt động chính xác.
* Chân kết nối:
* VCC: Chân cấp nguồn (5V).
* GND: Chân nối đất.
* AO (Analog Output): Chân xuất tín hiệu analog tỉ lệ với nồng độ khí.
* DO (Digital Output): Chân xuất tín hiệu digital, mức HIGH/LOW tùy theo ngưỡng cài đặt.

*- Chức năng và hoạt động*

+ Khi cấp nguồn, cuộn heater bên trong cảm biến sẽ làm nóng phần tử SnO₂ tới một nhiệt độ nhất định.

+ Ở môi trường bình thường, độ dẫn điện của phần tử rất thấp. Khi có khí dễ cháy xuất hiện:

* Các phân tử khí sẽ tương tác với bề mặt của SnO₂, làm thay đổi độ dẫn điện.
* Độ dẫn điện tăng tỉ lệ thuận với nồng độ khí phát hiện được.
* Tín hiệu điện thay đổi này sẽ được đưa ra chân AO (Analog Output) và có thể đọc bằng vi điều khiển để tính toán nồng độ khí.
* Nếu sử dụng chân DO, ta có thể thiết lập ngưỡng phát hiện, ví dụ khi nồng độ khí vượt ngưỡng an toàn, DO sẽ xuất tín hiệu HIGH để cảnh báo.

\* RFID NFC 13.56MHz RC522



RC522 là một module đọc/ghi thẻ RFID/NFC hoạt động ở tần số 13.56MHz, được sử dụng rộng rãi trong các ứng dụng nhận dạng, kiểm soát truy cập và hệ thống thanh toán không dây. Với kích thước nhỏ gọn, giá thành rẻ và khả năng giao tiếp linh hoạt (SPI, I2C, UART), RC522 rất phù hợp cho các dự án nhúng và IoT.

*- Cấu tạo và chân kết nối*

+ Cấu tạo:

* Chip RC522: Xử lý giao tiếp với thẻ RFID/NFC ở tần số 13.56MHz.
* Ăng-ten: Tích hợp trên mạch in, dùng để phát/thu tín hiệu từ thẻ.
* Bộ điều khiển giao tiếp: Hỗ trợ giao tiếp SPI, I2C hoặc UART với vi điều khiển.

+ Chân kết nối:

* SDA (SS): Chân chọn thiết bị khi giao tiếp SPI.
* SCK: Chân xung clock SPI.
* MOSI: Gửi dữ liệu từ MCU đến module.
* MISO: Nhận dữ liệu từ module về MCU.
* IRQ: Ngắt khi có thẻ tiếp xúc
* GND: Chân nối đất.
* RST: Chân reset module.
* 3.3V: Cấp nguồn 3.3V cho module.

*- Chức năng và hoạt động*:

+ Chức năng:

* + Đọc/ghi dữ liệu từ các thẻ RFID/NFC chuẩn ISO/IEC 14443A.
  + Hỗ trợ phát hiện, nhận diện thẻ trong vùng quét.

+ Hoạt động:

* + Khi thẻ RFID/NFC đi vào vùng phủ sóng của ăng-ten module, thẻ sẽ được cấp nguồn qua cảm ứng từ trường.
  + Module RC522 gửi/nhận dữ liệu với thẻ thông qua giao tiếp không dây.
  + Thông tin thu được sẽ được gửi về vi điều khiển để xử lý.

\* SSD1306 OLED I2C 128x64 Display



SSD1306 OLED là một loại màn hình hiển thị sử dụng công nghệ diode phát quang hữu cơ (OLED), cho phép hiển thị văn bản, hình ảnh và đồ họa với độ tương phản cao, tiêu thụ điện năng thấp. Với độ phân giải 128x64 pixel, kích thước nhỏ gọn và giao tiếp qua I2C, loại màn hình này rất phù hợp cho các dự án nhúng, IoT hoặc các ứng dụng cần giao diện hiển thị đơn giản.

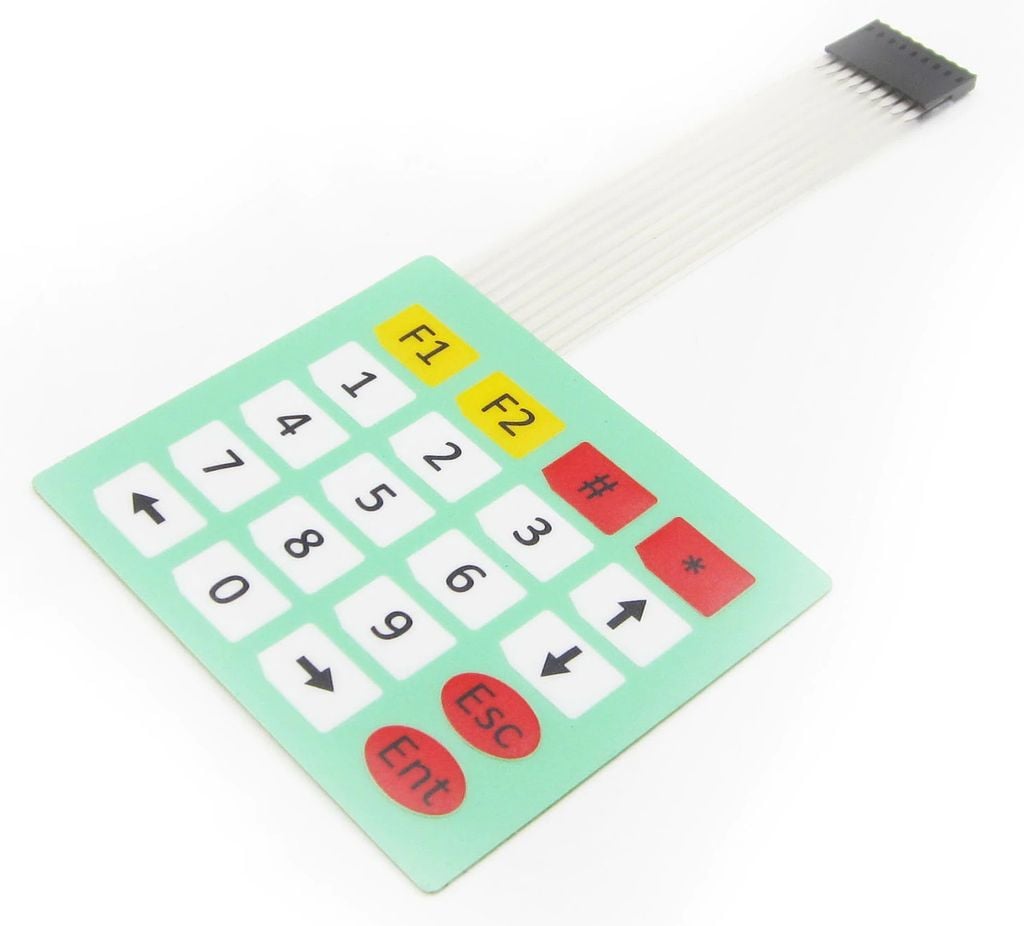
*- Chân kết nối:*

* GND: Chân nối đất.
* VCC: Chân cấp nguồn (thường 3.3V hoặc 5V).
* SCL (Serial Clock): Chân xung clock cho giao tiếp I2C.
* SDA (Serial Data): Chân truyền dữ liệu I2C.

*- Thông số kỹ thuật*

* Kích thước hiển thị: 0.96 inch
* Độ phân giải: 128 x 64 pixel
* Giao tiếp: I2C
* Điện áp hoạt động: 3.3V ~ 5V DC
* Dòng tiêu thụ: ~20mA
* Màu hiển thị: Đơn sắc trắng
* IC điều khiển: SSD1306
* Địa chỉ I2C mặc định: 0x3C
* Kích thước module: Khoảng 27mm x 27mm x 4mm

\* Module bàn phím ma trận 4\*5



Là một thiết bị nhập liệu bao gồm 20 phím nhấn (4 hàng x 5 cột) được tổ chức theo dạng ma trận. Khi người dùng nhấn một phím, hệ thống sẽ xác định phím đó thông qua giao điểm giữa hàng và cột tương ứng.

*- Cấu tạo và chân kết nối*

* Gồm 9 chân tín hiệu (5 chân cột và 4 chân hàng).
* Các chân này sẽ được kết nối trực tiếp đến vi điều khiển, quét theo hàng/cột để nhận dạng phím được nhấn.
* Mỗi phím vật lý nằm tại giao điểm của một hàng và một cột.

*- Chức năng và hoạt động*:

* Vi điều khiển sẽ quét tuần tự các cột (hoặc hàng) và đọc trạng thái các hàng (hoặc cột) còn lại.
* Khi một phím được nhấn, mạch giữa một hàng và một cột được nối thông, cho phép xác định chính xác vị trí của phím đó.
* Thuật toán quét phím phổ biến thường sử dụng phương pháp scan liên tục kết hợp với chống nhiễu để đảm bảo việc nhận dạng phím chính xác.

\* Công tắc chạm cảm ứng Ttp223 mini



TTP223 Mini là một mạch cảm ứng điện dung 1 kênh, có độ nhạy cao, thường được sử dụng như một công tắc cảm ứng thay thế cho các loại công tắc cơ truyền thống. Khi người dùng chạm tay vào vùng cảm ứng, module sẽ thay đổi trạng thái đầu ra (HIGH hoặc LOW), rất phù hợp cho các ứng dụng nhà thông minh, điều khiển thiết bị, hoặc các hệ thống IoT cần giao diện điều khiển cảm ứng đơn giản.

* *Cấu tạo và chân kết nối*:
* Vùng cảm ứng: Là bề mặt hình vuông nhỏ phía trước module, nhận diện sự thay đổi điện dung khi có ngón tay chạm vào.
* Chân VCC: Chân cấp nguồn cho module (hoạt động tốt ở mức điện áp từ 2.0V đến 5.5V, phổ biến là 3.3V hoặc 5V).
* Chân GND: Chân nối đất.
* Chân I/O (OUT): Chân tín hiệu đầu ra. Khi phát hiện chạm, chân này sẽ xuất mức HIGH (hoặc LOW tùy cấu hình).
* *Chức năng và hoạt động*:
* Mạch sẽ cảm nhận sự thay đổi điện dung khi chạm vào phần cảm ứng và kích hoạt ngõ ra OUT.
* TTP223 có thể thiết lập chế độ hoạt động khi hàn/không hàn 2 nút A và B trên mạch



* *Thông số kỹ thuật*



\* Servo SG90



Servo SG90 là một loại động cơ servo mini rất phổ biến trong các dự án DIY, robot, mô hình tự động hóa và ứng dụng nhúng. Với thiết kế nhỏ gọn, giá thành thấp và khả năng điều khiển chính xác vị trí góc quay

*- Cấu tạo và chân kết nối*

+ Dây kết nối:

* Dây đỏ: cấp nguồn VCC (3.3V ~ 6V)
* Dây nâu/đen: nối GND (mass)
* Dây cam/vàng: tín hiệu điều khiển PWM

+ Cấu trúc cơ khí:

* Bao gồm một mô tơ DC nhỏ, hệ thống bánh răng giảm tốc và mạch điều khiển điện tử bên trong.

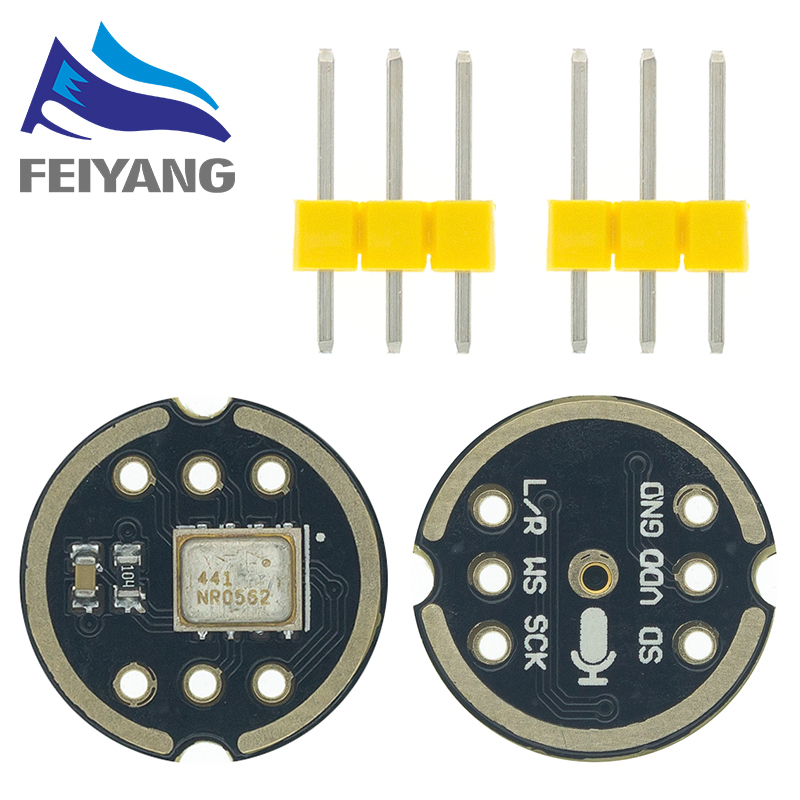
*- Chức năng và hoạt động*:

+ Servo SG90 nhận tín hiệu PWM để điều khiển góc quay của trục động cơ.

+ Góc quay giới hạn từ 0° đến 180°

+ Servo giữ cố định vị trí trục theo lệnh điều khiển cho đến khi có tín hiệu PWM mới.

\*Microphone đa hướng I2S INMP441



INMP441 là một loại microphone kỹ thuật số nhỏ gọn, sử dụng giao tiếp I2S để truyền dữ liệu âm thanh số trực tiếp đến vi điều khiển. Với khả năng thu âm đa hướng, độ nhiễu thấp và dễ tích hợp, nó được ứng dụng rộng rãi trong các hệ thống nhận dạng giọng nói, điều khiển bằng giọng nói và thiết bị IoT thông minh.

*- Chân kết nối:*

|  |  |  |
| --- | --- | --- |
| **Chân** | **Chức năng** | **Ghi chú** |
| VDD | Nguồn cấp | 1.8V – 3.3V |
| GND | Mass | Nối GND vi điều khiển |
| WS | Word Select | Chọn kênh trái/phải trong truyền dữ liệu |
| SCK | Serial Clock | Clock đồng bộ dữ liệu I2S |
| SD | Serial Data | Dữ liệu âm thanh dạng số |
| L/R | Lựa chọn kênh | Nối GND = kênh trái, VDD = kênh phải |

- *Chức năng và hoạt động:*

* Micro thu âm thanh môi trường và chuyển đổi nó thành tín hiệu số thông qua giao diện I2S.
* Cho phép truyền trực tiếp dữ liệu audio kỹ thuật số đến vi điều khiển hoặc SoC như ESP32, STM32, ...
* Micro hỗ trợ thu âm đa hướng, phù hợp cho nhận diện giọng nói trong không gian mở.
* Thường được sử dụng trong các ứng dụng như:
* Trợ lý ảo,
* Điều khiển giọng nói,
* Ghi âm IoT

2.2 Giao thức truyền dữ liệu và truyền thông

2.2.1 Giao thức truyền dữ liệu

2.2.2 Giao thức truyền thông

* 1. Nền tảng Cloud được sử dụng

Trong dự án này, Firebase được sử dụng làm nền tảng cloud chính, nhằm phục vụ cho việc lưu trữ dữ liệu và xác thực người dùng. Cụ thể, hai dịch vụ chính của Firebase được dùng là:

* Firebase Realtime Database:

Là một cơ sở dữ liệu NoSQL thời gian thực, cho phép lưu trữ và đồng bộ dữ liệu tức thì giữa các thiết bị IoT (như ESP32) và ứng dụng di động. Các dữ liệu như trạng thái thiết bị, thông tin cảm biến, lệnh điều khiển, … được cập nhật trực tiếp trên cơ sở dữ liệu và được phân phối tới tất cả các thiết bị đang kết nối một cách nhanh chóng. Điều này giúp hệ thống nhà thông minh phản hồi tức thời với các thay đổi, đảm bảo trải nghiệm người dùng mượt mà và chính xác.

* Firebase Authentication:

Được sử dụng để xác thực và quản lý tài khoản người dùng một cách an toàn. Với Firebase Auth, người dùng có thể đăng ký, đăng nhập và quản lý thông tin cá nhân một cách dễ dàng, đồng thời đảm bảo chỉ những người dùng hợp lệ mới có quyền truy cập và điều khiển hệ thống. Điều này tăng cường lớp bảo mật cho hệ thống smart home, hạn chế các truy cập trái phép.

* *Lý cho chọn Firebase:*
* Tích hợp nhanh chóng: Firebase cung cấp thư viện hỗ trợ tốt cho cả thiết bị nhúng và ứng dụng Android, giúp quá trình phát triển trở nên nhanh chóng và dễ dàng.
* Chi phí hợp lý: Với quy mô dự án vừa và nhỏ, Firebase cung cấp nhiều gói dịch vụ miễn phí phù hợp, tối ưu chi phí triển khai ban đầu.
* Khả năng mở rộng: Khi số lượng thiết bị hoặc người dùng tăng lên, Firebase có khả năng tự động mở rộng quy mô mà không cần can thiệp nhiều về hạ tầng.
* Bảo mật và ổn định: Firebase áp dụng các cơ chế bảo mật tiêu chuẩn cao, đồng thời đảm bảo độ tin cậy vận hành.

2.4 Ngôn ngữ lập trình, IDE và quản lý mã nguồn

2.4.1 Lập trình phần cứng

*\*Ngôn ngữ lập trình C/C ++*

- C++ là ngôn ngữ lập trình phổ biến được sử dụng rộng rãi trong phát triển phần mềm cho vi điều khiển, đặc biệt trong các ứng dụng thiết bị nhúng, hệ thống nhúng và Internet of Things (IoT). C++ kế thừa sức mạnh từ ngôn ngữ C và bổ sung thêm các tính năng lập trình hướng đối tượng, giúp tăng khả năng tổ chức, tái sử dụng và mở rộng mã nguồn.

- Trong dự án này, em sử dụng ngôn ngữ C++ kết hợp với lập trình hướng đối tượng nhằm dễ dàng quản lý các thiết bị, tổ chức mã nguồn khoa học, đồng thời thuận tiện cho việc mở rộng và bảo trì hệ thống trong tương lai.

*\* IDE Platform IO*



PlatformIO IDE là một môi trường phát triển tích hợp (IDE) hiện đại, được thiết kế chuyên biệt cho việc lập trình các hệ thống nhúng, vi điều khiển và ứng dụng IoT. PlatformIO hỗ trợ nhiều nền tảng phần cứng khác nhau như ESP32, STM32, Arduino, Raspberry Pi, và nhiều loại vi điều khiển khác.

PlatformIO được tích hợp như một extension trên các trình soạn thảo mã nguồn phổ biến như Visual Studio Code, Atom, hoặc cũng có thể chạy độc lập thông qua PlatformIO Core (CLI).

Một số tính năng nổi bật của PlatformIO IDE:

* Quản lý thư viện tự động: Dễ dàng cài đặt, cập nhật và quản lý các thư viện cần thiết cho dự án.
* Hỗ trợ đa nền tảng: Cho phép biên dịch và nạp chương trình cho nhiều dòng vi điều khiển khác nhau chỉ trong cùng một IDE.
* Tính năng IntelliSense mạnh mẽ: Gợi ý lệnh, tự động hoàn thiện mã nguồn và hỗ trợ kiểm tra lỗi khi viết code.
* Quản lý dự án chuyên nghiệp: Cấu trúc dự án rõ ràng, hỗ trợ nhiều tệp cấu hình như platformio.ini giúp quản lý cấu hình build, upload, debug dễ dàng.
* Hệ thống biên dịch tối ưu: Hỗ trợ các trình biên dịch tiên tiến như GCC cho nhiều kiến trúc vi điều khiển, đảm bảo chương trình chạy hiệu quả.

Với các tính năng mạnh mẽ này, PlatformIO IDE là lựa chọn lý tưởng cho việc phát triển ứng dụng trên ESP32 bằng ngôn ngữ C++, vừa tăng hiệu suất lập trình, vừa giúp việc phát triển phần mềm trở nên nhanh chóng

2.4.2 Lập trình ứng dụng Android

\* Kotlin với UI Jetpack Comopose



Trong dự án này, ứng dụng Android được phát triển bằng ngôn ngữ lập trình Kotlin, kết hợp với Jetpack Compose để xây dựng giao diện người dùng.

Kotlin là ngôn ngữ lập trình chính thức do Google khuyến nghị cho phát triển ứng dụng Android. Kotlin mang lại cú pháp ngắn gọn, an toàn hơn so với Java, đồng thời hỗ trợ đầy đủ các tính năng lập trình hướng đối tượng và lập trình hàm, giúp tăng tốc quá trình phát triển ứng dụng.

Jetpack Compose là bộ công cụ hiện đại của Google để thiết kế giao diện trên Android theo phong cách khai báo (declarative). Thay vì sử dụng XML truyền thống, lập trình viên viết trực tiếp UI bằng code Kotlin, giúp giao diện linh hoạt, dễ bảo trì và dễ dàng cập nhật trạng thái.

Một số ưu điểm nổi bật của việc sử dụng Kotlin và Jetpack Compose:

* Tối ưu hóa tốc độ phát triển: Việc xây dựng giao diện trở nên đơn giản, trực quan và dễ chỉnh sửa hơn.
* Tính năng lập trình khai báo: UI tự động cập nhật khi dữ liệu thay đổi, giảm thiểu lỗi và tiết kiệm công sức.
* Tính tương thích cao: Jetpack Compose tích hợp chặt chẽ với các thành phần Android hiện có như ViewModel, LiveData, Navigation, ...
* Hiệu suất tốt: Các thành phần UI của Jetpack Compose được tối ưu hóa để đảm bảo hiệu suất cao khi chạy trên thiết bị Android.

Với sự kết hợp giữa Kotlin và Jetpack Compose, ứng dụng Android trong dự án có giao diện hiện đại, dễ mở rộng, đồng thời đảm bảo hiệu quả phát triển và khả năng bảo trì lâu dài.

2.4.3 Quản lý mã nguồn với Git và GitHub



**Git** là một hệ thống quản lý phiên bản phân tán *(Distributed Version Control System - DVCS)* được phát triển bởi Linus Torvalds vào năm 2005, nhằm hỗ trợ việc quản lý mã nguồn trong các dự án phần mềm lớn như Linux Kernel. Ngày nay, Git đã trở thành công cụ tiêu chuẩn cho việc quản lý mã nguồn trong hầu hết các dự án phần mềm.

Khác với các hệ thống quản lý phiên bản tập trung, Git lưu trữ toàn bộ lịch sử của dự án ngay trên mỗi máy tính của người dùng. Điều này mang lại nhiều ưu điểm:

* *Hiệu suất cao:* Các thao tác như commit, xem lịch sử, hoặc tạo nhánh (branch) đều được thực hiện nhanh chóng vì không cần truy cập server liên tục.
* *An toàn dữ liệu:* Git sử dụng cơ chế băm SHA-1 để kiểm tra và bảo vệ toàn vẹn dữ liệu, đảm bảo rằng mọi thay đổi đều có thể được kiểm soát và phát hiện ngay lập tức nếu có sự cố.
* *Hỗ trợ phân nhánh và hợp nhất mạnh mẽ:* Git cho phép tạo branch nhanh chóng, phục vụ tốt cho mô hình phát triển song song và thử nghiệm các tính năng mới mà không làm ảnh hưởng đến nhánh chính.

Git vận hành dựa trên ba khu vực chính:

* *Working Directory:* Nơi người dùng chỉnh sửa, thêm, hoặc xóa file.
* *Staging Area (Index):* Khu vực tạm để lưu trữ những thay đổi sẽ được commit.
* *Repository:* Lưu trữ lịch sử của tất cả các commit trong dự án.

Trong dự án này, **Git** được sử dụng để quản lý mã nguồn, kiểm soát phiên bản và phối hợp nhóm. **GitHub** đóng vai trò là kho lưu trữ trực tuyến, cho phép lưu trữ mã nguồn Git

* Mã nguồn của Project: <https://github.com/nguyenmanhdung183/DATN>

# **CHƯƠNG 3:** THIẾT KẾ TỔNG QUAN HỆ THỐNG VÀ ĐIỀU KHIỂN GIÁM SÁT

## **3.1 Sơ đồ cấu trúc hệ thống**

A diagram of a cloud platform

AI-generated content may be incorrect.

## **3.2 Nguyên lý điều khiển và truyền nhận dữ liệu**

* Hệ thống nhà thông minh được thiết kế với cơ cấu truyền nhận dữ liệu tập trung thông qua Cloud Platform, như thể hiện trong sơ đồ cấu trúc hệ thống
* Dữ liệu từ Sensors & Buttons được gửi trực tiếp đến Gateway (sử dụng module ESP32) để thu thập thông tin môi trường như nhiệt độ, trạng thái cảm biến hoặc trạng thái nút bấm.
* Ứng dụng điều khiển, được triển khai trên nền tảng Android, gửi dữ liệu (Data) lên Cloud Platform, bao gồm các lệnh điều khiển, thông tin trạng thái thiết bị hoặc, cài đặt hệ thống (mật khẩu, tên thiết bị, id thiết bị, …)
* Cloud Platform, đóng vai trò như một database chung, lưu trữ dữ liệu ứng dụng điều khiển và trung chuyển dữ liệu này (Data) đến Gateway.
* Gateway, với bộ vi xử lý tích hợp trên ESP32, xử lý các dữ liệu nhận được từ Sensors & Buttons và Cloud Platform, sau đó chuyển tiếp các lệnh thực thi đến bộ chấp hành để điều khiển thiết bị như đèn, quạt hoặc cửa, …
* Sau khi thực thi, nếu lệnh thực thi xuất phát từ nút bấm hoặc cảm biến, gateway gửi trạng thái thiết bị (Data) lên Cloud Platform để lưu trữ. App điều khiển truy xuất dữ liệu phản hồi từ Cloud Platform để cập nhật giao diện người dùng, tạo thành một luồng dữ liệu khép kín thông qua Cloud Platform

## **3.3 Tích hợp và quản lý dữ liệu trên Cloud**

### **3.3.1 Quản lý đăng nhập, đăng ký và xác thực người dùng**

A screenshot of a computer

AI-generated content may be incorrect.

Firebase Authentication được tích hợp để quản lý đăng nhập, đăng ký và xác thực người dùng, đảm bảo chỉ những người dùng hợp lệ mới có thể truy cập hệ thống. Các tính năng chính bao gồm:

* *Đăng ký người dùng*: Người dùng có thể đăng ký tài khoản bằng email và mật khẩu thông qua ứng dụng. Firebase Authentication lưu trữ thông tin người dùng (email, UID) trong hệ thống của nó.
* *Đăng nhập*: Người dùng đăng nhập bằng email và mật khẩu. Firebase Authentication xác minh thông tin và trả về một token xác thực (ID Token) để ứng dụng sử dụng trong các yêu cầu tiếp theo.
* *Quên mật khẩu (lấy lại qua email):* Trong trường hợp người dùng quên mật khẩu, ứng dụng cung cấp tùy chọn "Quên mật khẩu" trên màn hình đăng nhập. Người dùng nhập email đã đăng ký, sau đó nhấn nút "Gửi yêu cầu". Firebase Authentication sẽ gửi một email chứa liên kết đặt lại mật khẩu đến địa chỉ email của người dùng. Người dùng nhấp vào liên kết, nhập mật khẩu mới, và Firebase cập nhật mật khẩu trong hệ thống. Tính năng này đảm bảo người dùng có thể khôi phục quyền truy cập một cách an toàn và tiện lợi.
* *Xác thực người dùng:* Mỗi yêu cầu từ Control Application đến Firebase Realtime Database (như gửi lệnh điều khiển hoặc truy xuất dữ liệu) đều được xác thực thông qua token. Firebase Security Rules được thiết lập để chỉ cho phép người dùng đã đăng nhập truy cập dữ liệu liên quan đến tài khoản của họ. Ví dụ, một quy tắc có thể là: chỉ người dùng có UID khớp với UID trong nhánh dữ liệu mới được phép đọc/ghi.
* *Bảo mật:* Firebase Authentication hỗ trợ mã hóa mật khẩu và cung cấp các tùy chọn bảo mật bổ sung như xác thực hai yếu tố (nếu cần), đảm bảo an toàn cho thông tin người dùng.

### **3.3.2 Dữ liệu điều khiển và dữ liệu hệ thống**

A screenshot of a computer

AI-generated content may be incorrect.

Firebase Realtime Database được sử dụng để lưu trữ và quản lý dữ liệu của hệ thống nhà thông minh, với cấu trúc được tổ chức theo từng người dùng, phòng và thiết bị, giúp việc truy xuất và đồng bộ dữ liệu trở nên hiệu quả. Dữ liệu được chia thành hai loại chính: *dữ liệu điều khiển* (các lệnh từ người dùng) và *dữ liệu hệ thống* (thông tin về thiết bị, phòng và trạng thái).

- Dữ liệu điều khiển:

+ Gồm các lệnh mà người dùng gửi từ Control Application để điều khiển thiết bị, chẳng hạn như bật/tắt đèn. Trong hệ thống này, các lệnh điều khiển được thực hiện bằng cách cập nhật trực tiếp trạng thái (status) của thiết bị trong nhánh device\_id dưới mỗi room\_id. Gateway (ESP32) lắng nghe sự thay đổi của status để thực thi lệnh.

*+ Ví dụ minh họa:* Giả sử người dùng muốn bật đèn trong phòng bếp: App cập nhập trạng thái lên Firebase -> Gateway fetch thông tin về nhận thấy “status” của bếp đang có giá trị “on”’-> bật đèn phòng bếp

- Dự liệu khác:

+ Dữ liệu về tên và loại thiết bị hoặc phòng

+ Dữ liệu về email của người dùng để tiện lợi cho việc theo dõi

+ newstDevice: Dùng khi tạo 1 thiết bị mới giúp đồng bộ giữa các máy khác nhau (đăng nhập cùng 1 tài khoản)

+ newsetRoom: Dùng khi tạo 1 phòng mới giúp đồng bộ giữa các máy khác nhau (đăng nhập cùng 1 tài khoản)

+ newestNoti: Dùng khi có 1 thông báo mới, giúp cho thông báo đến app khi có thông báo từ gateway (thông báo rò khí gas), tránh việc phải fetch liên tuc list thông báo có sẵn

## **3.4 Thiết kế ứng dụng điều khiển**

### **3.4.1 Giao diện ứng dụng điều khiển**

*a) Thiết kế giao diện với Figma*

Screens screenshot of a phone

AI-generated content may be incorrect.

Figma được sử dụng để thiết kế giao diện ứng dụng trước khi triển khai, giúp hình dung bố cục và phong cách trực quan. Các màn hình chính của ứng dụng bao gồm:

* Màn hình chính: hiển thị danh sách phòng trong ngôi nhà và thẻ thêm phòng
* Giao diện quản lý thiết bị trong phòng: hiển thị danh sách các thiết bị trong phòng và thẻ thêm thiết bị
* Giao diện thông báo: Hiển thị danh sách các thông báo từ hệ thống
* Giao diện cài đặt: hiển thị thông tin app, tích hợp với phần quản lý tài khoản (đăng ký, đăng nhập, quên mật khẩu).
* Các giao diện chỉnh sửa thiết bị và phòng: Sửa/xoá, hiển thị thông tin của thiết bị/phòng
* Các giao diện thêm thiết bị và phòng: Nhập tên phòng/thiết bị, chọn loại thiết bị
* Các giao diện điều khiển chuyên sâu: Điều chỉnh điều hoà, đặt mật khẩu cho cửa, …

Trong Figma, giao diện được thiết kế với phong cách tối giản, sử dụng bảng màu trung tính (trắng, xám, xanh dương) để tạo cảm giác hiện đại. Các nút điều khiển được thiết kế nổi bật (ví dụ: nút bật/tắt có màu xanh/đỏ) để dễ nhận diện. Sau khi hoàn thiện, thiết kế Figma được xuất thành các tài nguyên (hình ảnh, kích thước, màu sắc) để sử dụng trong giai đoạn triển khai.

*b) Triển khai code giao diện*

Giao diện của ứng dụng được triển khai trên Android Studio, sử dụng **Jetpack Compose**, Jetpack Compose cho phép viết giao diện bằng mã Kotlin theo cách khai báo (declarative), giúp mã nguồn ngắn gọn, dễ đọc và dễ bảo trì hơn so với cách truyền thống sử dụng XML. Các thành phần giao diện chính của ứng dụng được thiết kế để đảm bảo tính trực quan và thân thiện với người dùng.

A screen shot of a phone

AI-generated content may be incorrect.

Phần giao diện được code dựa trên thiết kế Figma như mục ở trên, đồng thời để có thể sử dụng tính năng cập nhật giao diện tự động khi có data thay đổi ở Cloud, trong phần code có sử dụng tính năng Laucher Effect để theo dõi sự thay đổi của biến được gán tương ứng

Screens screenshot of a phone

AI-generated content may be incorrect.

### **3.4.2 Chức năng điều khiển và quản lý trên ứng dụng**

1. *Kết nối với Firebase*

- *Mô tả chức năng*: Ứng dụng sử dụng Firebase SDK để thiết lập kết nối với Firebase Realtime Database, cho phép gửi và nhận dữ liệu thời gian thực (như trạng thái thiết bị, dữ liệu cảm biến).

- *Vấn đề kỹ thuật:*

* *Tích hợp Firebase SDK*: Ứng dụng cần thêm dependency com.google.firebase:firebase-database vào file build.gradle để sử dụng Firebase Realtime Database. Sau đó, ứng dụng khởi tạo kết nối bằng cách sử dụng FirebaseDatabase.getInstance() và tham chiếu đến database.
* *Cấu hình kết nối Firebase:*
* Cung cấp package name của ứng dụng và thêm SHA-1 key để hỗ trợ xác thực người dùng
* Tải về file cấu hình google-services.json từ Firebase Console và đặt vào thư mục app/ của dự án Android.
* Thêm plugin com.google.gms.google-services vào file build.gradle cấp project và cấp module để kích hoạt dịch vụ Google.
* *Bảo mật kết nối:* Tất cả dữ liệu trao đổi giữa ứng dụng và Firebase đều được mã hóa bằng giao thức *HTTPS*. Ngoài ra, để truy xuất dữ liệu chính xác, ứng dụng cần xác định user\_id cho mỗi người dùng và tuân thủ các quy tắc bảo mật (Security Rules) được cấu hình trên Firebase Database.

1. *Quản lý xác thực người dùng*

* *Mô tả chức năng:* Ứng dụng tích hợp Firebase Authentication để quản lý đăng nhập, đăng ký, lấy lại mật khẩu và duy trì phiên đăng nhập của người dùng.
* *Vấn đề kỹ thuật:*
* *Đăng nhập/đăng ký:* Ứng dụng sử dụng FirebaseAuth để xử lý đăng nhập và đăng ký bằng email/mật khẩu. Khi người dùng đăng nhập, FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password) được gọi, trả về một FirebaseUser với UID. UID này được sử dụng để truy cập nhánh dữ liệu của người dùng trên Firebase Realtime Database.
* *Quên mật khẩu:* Tính năng "Quên mật khẩu" sử dụng FirebaseAuth.getInstance().sendPasswordResetEmail(email) để gửi email đặt lại mật khẩu.
* *Quản lý phiên đăng nhập:* Ứng dụng kiểm tra trạng thái đăng nhập bằng FirebaseAuth.getInstance().currentUser khi khởi động. Nếu người dùng đã đăng nhập (token còn hiệu lực), ứng dụng tự động chuyển đến màn hình chính; nếu không, chuyển đến màn hình đăng nhập
* *Bảo mật:* Firebase Security Rules được thiết lập để chỉ cho phép người dùng đã xác thực truy cập dữ liệu của họ

1. *Quản lý thiết bị/phòng*

* *Mô tả chức năng:* Ứng dụng cho phép người dùng xem danh sách thiết bị/phòng, thêm thiết bị/phòng mới, và xóa thiết bị khỏi hệ thống.
* *Vấn đề kỹ thuật:*
* *Hiển thị danh sách thiết bị:* Ứng dụng truy xuất danh sách thiết bị từ Firebase. Dữ liệu được lấy bằng ValueEventListener và hiển thị trên giao diện bằng LazyColumn (Jetpack Compose).
* *Thêm thiết bị:* Khi người dùng thêm thiết bị mới với dữ liệu người dùng nhập gồm tên, và loại thiết bị, phòng chứa thiết bị, ứng dụng tạo một device\_id mới và id được tạo ngẫu nhiên 1 dãy số, nếu dãy số đó trùng với 1 device\_id trong database thì sẽ được tạo lại đến khi hợp lệ và ghi thông tin thiết bị (như devicename, devicetype, status, roomname) vào nhánh room\_id. Đồng thời, ứng dụng cập nhật newDevice trong nhánh main để lưu ID thiết bị mới nhất.
* *Xóa thiết bị:* Khi người dùng xóa thiết bị, ứng dụng xóa nhánh device\_id tương ứng trên Firebase.
* *Thêm phòng:* Khi người dùng thêm phòng mới với dữ liệu là tên phòng, ứng dụng tạo một room\_id mới và id được tạo ngẫu nhiên 1 dãy số, nếu dãy số đó trùng với 1 room\_id trong database thì sẽ được tạo lại đến khi hợp lệ và ghi thông tin phòng vào nhánh user\_id. Đồng thời, ứng dụng cập nhật newRoom trong nhánh main để lưu ID phòng mới nhất
* *Xóa phòng:* Khi người dùng xóa thiết bị, ứng dụng xóa nhánh room\_id tương ứng trên Firebase. Đồng thời các dữ liệu nhu phòng kèm theo sẽ được tự động xoá

1. *Điều khiển thiết bị*

* Khi người dùng chỉnh các thông tin như (trạng thái bật tắt, mật khẩu,…) thông qu giao diện của ứng dụng thì dữ liệu sẽ được đẩy lên database với nhánh trùng với thiết bị và phòng tương ứng, từ đó Gateway có thể dựa vào để điều chỉnh trạng thái các thiết bị

1. *Nhận dữ liệu từ cập nhập từ Gateway*

* Ứng dụng lắng nghe nhánh device\_id bằng ValueEventListener. Khi Gateway cập nhật status (ví dụ: từ off thành on), trạng thái cửa mở/đóng hoặc một số thông tin khác như nhiệt độ của phòng, ứng dụng nhận trạng thái và cập nhật giao diện (dùng Launch Effect) đồng thời thay đổi giá trị biến tương ứng với mỗi thiết bị ở trên máy

1. *Nhận thông báo giám sát từ Gateway*

* *Mô tả chức năng:* Ứng dụng nhận các thông báo giám sát từ Gateway (như cảnh báo về trạng thái hệ thống hoặc sự kiện quan trọng) và hiển thị trên màn hình thông báo để người dùng theo dõi.
* *Cấu trúc dữ liệu thông báo*:

+ Thông báo được lưu trữ trong nhánh noti dưới User\_id trên Firebase Realtime Database. Mỗi thông báo là một mục (entry) với ID duy nhất (ví dụ: 18336\_3917), chứa các thông tin:

*text: Nội dung thông báo (ví dụ: "Nút nhấn đã thay đổi trạng thái").*

*time: Thời gian xảy ra (ví dụ: "esp32w2.15.47.12.19.03.2025").*

+ Ngoài ra, nhánh main/newNoti (ví dụ: "esp32w2.15.48.45.19.03.2025") lưu trữ thời gian của thông báo mới nhất, giúp ứng dụng xác định thời điểm cập nhật thông báo để có thể cập nhật những thông báo mới mà không cần lắng nghe và so sánh tất cả các thông báo hiện tại

+ Ở phần *time* trong thông báo là sự kết hợp giữa Gateway\_id + thời gian thông báo, tại ứng dụng sẽ bóc tách 2 phần ra để hiển thị thời gian thông báo lên hệ thống và sắp xếp theo thứ tự thời gian

+ Phần id của mỗi thông báo sẽ khác nhau, chúng được tạo ngẫu nhiên từ Gateway tránh việc trùng lặp thông báo

A screenshot of a computer

AI-generated content may be incorrect.

CHƯƠNG 4: THIẾT KẾ PHẦN CỨNG VÀ LẬP TRÌNH NHÚNG HỆ THỐNG NHÀ THÔNG MINH

## **Thiết kế phần cứng hệ thống nhà thông minh**

* **Tổng quan thiết kế phần cứng**:  
  Hệ thống nhà thông minh được xây dựng với một trung tâm điều khiển là ESP32, kết nối với các cảm biến, thiết bị điều khiển, và giao diện người dùng. Mục tiêu là tạo ra một hệ thống linh hoạt, tiết kiệm năng lượng, và dễ mở rộng để áp dụng trong không gian gia đình. Các thành phần được cấp nguồn từ điện áp tiêu chuẩn, với các kết nối được sắp xếp để giảm thiểu nhiễu và đảm bảo hoạt động ổn định.
* **Sơ đồ khối hệ thống**:
* **Trung tâm điều khiển (ESP32)**: Làm nhiệm vụ chính, thu thập thông tin từ các cảm biến, nhận lệnh từ người dùng qua các giao diện, và điều khiển các thiết bị trong nhà. ESP32 hỗ trợ kết nối không dây để giao tiếp với hệ thống lưu trữ dữ liệu từ xa.
* **Cảm biến**:
  + Cảm biến khí gas: Phát hiện sự hiện diện của khí gas để kích hoạt quạt thông gió.
  + Cảm biến tiệm cận: Phát hiện sự di chuyển của người để điều khiển đèn.
  + Thiết bị nhận diện thẻ: Đọc thông tin từ thẻ để kiểm soát cửa.
* **Thiết bị điều khiển**:
  + Đèn chiếu sáng: Có thể bật/tắt dựa trên lệnh từ người dùng hoặc cảm biến, công tắc.
  + Quạt thông gió: Hoạt động khi cần làm mát hoặc khi phát hiện khí gas.
  + Thiết bị điều khiển cửa: Mở hoặc đóng cửa dựa trên lệnh từ người dùng.
* **Giao diện người dùng**:
  + Nút cảm ứng: Cho phép người dùng điều khiển đèn và quạt trực tiếp.
  + Bàn phím: Dùng để nhập thông tin mở cửa.
  + Màn hình hiển thị: Hiển thị trạng thái hoặc thông báo cho người dùng.
  + Thiết bị thu âm: Hỗ trợ nhận lệnh bằng giọng nói từ người dùng.
* **Nguồn cấp**:
  + ESP32 được cấp nguồn từ nguồn điện tiêu chuẩn, cung cấp năng lượng cho các cảm biến và thiết bị.
  + Thiết bị lớn hơn (như quạt), dùng nguồn 5V và được nhận lệnh điều khiển thông qua 1 transitor với tín hiệu điều khiển đến từ GPIO của ESP32.

## **Lập trình phần cứng cho hệ thống**

### **Tổng quan về luồng hoạt động**

**a) Đèn chiếu sáng**

* **Mô tả**: Đèn được điều khiển trực tiếp bởi người dùng, từ xa qua hệ thống lưu trữ dữ liệu, hoặc bằng giọng nói.
* **Luồng hoạt động**:
  1. **Khởi tạo**: Trong giai đoạn khởi động, hệ thống thiết lập đèn ở trạng thái như trạng thái ở database.
  2. **Nhận lệnh**:
     1. **Nút cảm ứng**: Hệ thống phát hiện tín hiệu từ nút cảm ứng, chuyển đổi trạng thái đèn.
     2. **Hệ thống lưu trữ dữ liệu**: Hệ thống kiểm tra định kỳ trạng thái từ xa, nếu nhận lệnh bật, đèn sẽ được kích hoạt.
     3. **Giọng nói**: Thiết bị thu âm nhận lệnh từ người dùng, xử lý và bật đèn tương ứng.
  3. **Đồng bộ trạng thái**: Trạng thái đèn được gửi đến hệ thống lưu trữ dữ liệu sau mỗi lần thay đổi.

**b) Quạt thông gió**

* **Mô tả**: Quạt được điều khiển tự động dựa trên cảm biến khí gas, trực tiếp bởi người dùng, từ xa qua hệ thống lưu trữ dữ liệu, hoặc bằng giọng nói.
* **Luồng hoạt động**:
  1. **.Khởi tạo**: Trong giai đoạn khởi động, hệ thống thiết lập quạt ở trạng thái như trạng thái ở database.
  2. **Nhận lệnh**:
     + - **Cảm biến khí gas**: Hệ thống kiểm tra liên tục tín hiệu từ cảm biến, nếu phát hiện khí gas, quạt bật và gửi thông báo. Ví dụ: Lúc 04:03 PM, quạt bật khi phát hiện khí.
       - **Nút cảm ứng**: Hệ thống phát hiện tín hiệu từ nút cảm ứng, chuyển đổi trạng thái quạt.
       - **Hệ thống lưu trữ dữ liệu**: Hệ thống kiểm tra trạng thái từ xa, nếu nhận lệnh tắt, quạt dừng hoạt động.
       - **Giọng nói**: Lệnh từ người dùng được xử lý, quạt tắt theo yêu cầu.
  3. **Đồng bộ trạng thái**: Trạng thái quạt được gửi đến hệ thống lưu trữ dữ liệu sau mỗi lần thay đổi.

**c) Cửa**

* 1. **Mô tả**: Cửa được điều khiển bằng thiết bị cơ học, mở/đóng dựa trên thẻ nhận diện, mật khẩu từ bàn phím, lệnh từ xa.
  2. **Luồng hoạt động**:
     1. **Khởi tạo**: Trong giai đoạn khởi động, hệ thống thiết lập cửa ở trạng thái ban đầu là đóng.
     2. **Nhận lệnh**:
        + **Thẻ nhận diện**: Hệ thống kiểm tra thông tin từ thẻ, nếu hợp lệ, cửa mở hoặc đóng.
        + **Bàn phím**: Hệ thống nhận thông tin từ bàn phím, nếu mật khẩu đúng, cửa mở.
        + **Hệ thống lưu trữ dữ liệu**: Hệ thống kiểm tra trạng thái từ xa, nếu nhận lệnh mở, cửa được kích hoạt.
     3. **Thực thi**: Cửa chuyển sang trạng thái mở hoặc đóng, hiển thị thông báo trên màn hình.
     4. **Đồng bộ trạng thái**: Trạng thái cửa được gửi đến hệ thống lưu trữ dữ liệu sau mỗi lần thay đổi.

**d) Đèn cảm biến có người**

* 1. **Mô tả**: Đèn được điều khiển tự động dựa trên cảm biến tiệm cận, bật khi phát hiện sự hiện diện của người và tắt khi không có người.
  2. **Luồng hoạt động**:
     1. **Khởi tạo**: Trong giai đoạn khởi động, hệ thống thiết lập đèn và cảm biến sẵn sàng hoạt động.
     2. **Nhận dữ liệu**: Hệ thống kiểm tra tín hiệu từ cảm biến tiệm cận, nếu phát hiện người, đèn được kích hoạt.
* **Vấn đề kỹ thuật trong luồng hoạt động**:
* **Xung đột tín hiệu từ nút bấm và bàn phím**: Nếu xử lý trực tiếp trong luồng chính, tín hiệu từ nút bấm hoặc bàn phím có thể bị bỏ sót khi hệ thống bận xử lý các tác vụ khác (như thu âm thanh).
* **Giải pháp**: Tách luồng xử lý nút bấm và bàn phím thành một luồng riêng, đẩy tín hiệu vào một hàng đợi (queue). Luồng chính sau đó đọc queue để thực hiện lệnh, đảm bảo không bỏ sót tín hiệu và giảm nguy cơ lỗi.
* **Độ trễ xử lý âm thanh**: Quá trình thu âm thanh, lưu vào RAM, và gửi lên Wit.ai có thể gây chậm nếu kết nối không dây không ổn định.
* **Giải pháp**: Giới hạn thời gian thu âm ở mức 2.5 giây, đảm bảo dữ liệu âm thanh nhỏ gọn (khoảng 80KB), phù hợp với RAM của ESP32, và tối ưu kết nối không dây để giảm độ trễ.
* **Quản lý tài nguyên**: Thu âm thanh (80KB trong 2.5 giây) chiếm một phần bộ nhớ RAM, có thể gây quá tải nếu xử lý cùng lúc nhiều nhiệm vụ.
* **Giải pháp**: Gửi dữ liệu âm thanh trực tiếp lên Wit.ai ngay sau khi xử lý thành file WAV, không lưu trữ lâu dài trên ESP32, và sử dụng cơ chế hàng đợi để quản lý các tác vụ khác

### **Thiết lập môi trường lập trình và các thành phần cơ bản**

* 1. Các thư viện cần thiết:
* *Thư viện Firebase\_ESP\_Client*: Được sử dụng để giao tiếp với Firebase Realtime Database, cho phép ESP32 gửi/nhận dữ liệu thời gian thực (như trạng thái thiết bị, thông báo giám sát). Thư viện này được cài đặt thông qua Library Manager trong Arduino IDE bằng cách tìm "Firebase ESP Client".
* *Thư viện WiFi*: Có sẵn trong Arduino IDE, được sử dụng để kết nối ESP32 với mạng Wi-Fi, là bước đầu tiên để giao tiếp với Firebase.
* *Thư viện time*: Có sẵn trong Arduino IDE, được sử dụng để đồng bộ thời gian từ NTP server (pool.ntp.org), giúp ESP32 tạo thời gian chính xác cho các thông báo (ví dụ: 15.47.12.19.03.2025).
* *Thư viện ESP32Servo*: Được sử dụng trong mã điều khiển cửa RFID để điều khiển servo (mở/đóng cửa).
* Thư viện MFRC522: Được sử dụng để giao tiếp với module RFID trong mã điều khiển cửa, cho phép đọc UID của thẻ RFID. Thư viện này được cài đặt qua Library Manager bằng cách tìm "MFRC522".
* *Thư viện Adafruit\_GFX và Adafruit\_SSD1306*: Được sử dụng để điều khiển màn hình OLED (hiển thị thông báo như "Open Door", "Wrong Password"). Các thư viện này được cài đặt qua Library Manager bằng cách tìm "Adafruit GFX" và "Adafruit SSD1306".
* *Thư viện Wire:* Có sẵn trong Arduino IDE, được sử dụng để giao tiếp I2C với màn hình OLED (chân SDA 21, SCL 22).
* *Thư viện SPI:* Có sẵn trong Arduino IDE, được sử dụng để giao tiếp SPI với module RFID (các chân SCK 18, MOSI 23, MISO 19, SDA/SS 5).
* *Thư viện vector*: Được sử dụng để quản lý danh sách động (như danh sách thiết bị Device::dvList hoặc danh sách chân nút nhấn buttonPins). Thư viện này là một phần của C++ STL và được hỗ trợ trên ESP32.
  1. Cấu hình kết nối Wifi
* Hàm *setupWifiFirebase*() thực hiện kết nối Wi-Fi bằng cách gọi WiFi.begin(WIFI\_SSID, WIFI\_PASSWORD). Quá trình kết nối được lặp lại cho đến khi thành công.
  1. Đồng bộ thời gian thực từ Internet
* Để tạo thời gian chính xác cho các thông báo, ESP32 đồng bộ thời gian từ NTP server (pool.ntp.org) bằng hàm *configTime*()

### **Lập trình giao tiếp với Firebase**

*a) Kết nối và xác thực với Firebase*

- ESP32 sử dụng thư viện Firebase\_ESP\_Client để thiết lập kết nối với Firebase Realtime Database

- Thông tin Firebase được cấu hình bao gồm URL database và API key. Hàm Firebase.begin() được gọi để khởi tạo kết nối, với cơ chế tự động kết nối lại khi mất kết nối.

*b) Gửi dữ liệu lên Firebase*

- SP32 đọc dữ liệu từ Firebase để nhận lệnh điều khiển từ ứng dung, như trạng thái thiết bị (status) hoặc mật khẩu cửa (password), …. Hàm *downloadData*() được sử dụng để đọc dữ liệu từ một đường dẫn cụ thể:

* Đường dẫn được tạo bằng cách kết hợp USER\_ID (ví dụ: TcJzM8sKGeWkiaeQJgR7e6G5qxq1) với đường dẫn thiết bị (như /room\_id/23714458/device\_id/803872522/status).
* Hàm Firebase.RTDB.getString() được gọi để lấy dữ liệu dưới dạng chuỗi (ví dụ: on hoặc off). Nếu thành công, dữ liệu được trả về; nếu thất bại, trả về chuỗi "error".
* Hàm StreamData() gọi downloadData() để đọc trạng thái của các thiết bị trong danh sách Device::dvList (như fanBep, lightBep) realtime với khoảng thời gian có thể điều chỉnh (ở project này đang set ở 1000ms).

*c) Nhận dữ liệu từ Firebase*

- ESP32 gửi dữ liệu lên Firebase để cập nhật trạng thái thiết bị hoặc thông báo giám sát. Hàm *uploadData*() được sử dụng để ghi dữ liệu:

* Đường dẫn được tạo tương tự như khi đọc dữ liệu, kết hợp USER\_ID với đường dẫn thiết bị.
* Hàm *Firebase.RTDB.setString()* được gọi để ghi dữ liệu dưới dạng chuỗi.
* Ví dụ: Trong lớp Light, phương thức *sendStateToFirebase*() gọi *uploadData*() để gửi trạng thái đèn (on/off) lên Firebase (như /23714458/903872522/status).
  + 1. *Gửi thông báo hệ thống lên Firebase*
* ESP32 gửi thông báo giám sát lên Firebase khi phát hiện sự kiện quan trọng (như phát hiện khí gas). Hàm *pushNotification*() được sử dụng để gửi thông báo:
* Tạo thời gian hiện tại bằng hàm *getCurrentTime*().
* Tạo một khóa random duy nhất (pushKey) bằng hàm *generateKey*()
* Gửi thông báo lên nhánh /main/noti/ với các trường:

*text: Nội dung thông báo (ví dụ: "Có khí gas").*

*time: Thời gian xảy ra(ví dụ: esp32w2.15.47.12.19.03.2025).*

* Cập nhật thời gian thông báo mới nhất vào nhánh /main/newestNoti.
* Trong lớp MQ2, phương thức sentNotiToFb() gọi pushNotification("Có khí gas", DEVICE\_ID) để gửi thông báo khi phát hiện khí gas.
  + 1. **Thiết kế phần mềm nhúng theo hướng đối tượng**
  1. *Mô tả tổng quan*
* Lập trình hướng đối tượng (OOP) được áp dụng trong mã nhúng cho Gateway (ESP32) với mục đích tổ chức mã nguồn một cách có cấu trúc, dễ đọc, dễ bảo trì và tái sử dụng và quản lý thiết bị một cách hiệu quả. Ở đây mỗi thiết bị trong hệ thống được biểu diễn bằng một lớp riêng biệt để đóng gói dữ liệu.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

* 1. *Áp dụng trong mã*

**Lớp Device (lớp cơ sở)**: Được sử dụng trong mã điều khiển thiết bị (đèn, quạt, điều hòa) để định nghĩa các thuộc tính và hành vi chung cho tất cả thiết bị.

* Thuộc tính: *pin* (GPIO pin), *id* (ID thiết bị), name (tên thiết bị), *roomId* (ID phòng), *mainState* (trạng thái bật/tắt), *otherState* (nhiệt độ/độ ẩm), *buttonPin* (chân nút nhấn), *path* (đường dẫn Firebase).
* Phương thức: Các getter/setter (như getMainState(), setMainState()), và một phương thức ảo FbUpdate() để cập nhật trạng thái từ Firebase.
* Lớp này sử dụng một vector tĩnh dvList để lưu trữ danh sách các thiết bị cần cập nhật trạng thái từ Firebase, giúp quản lý tập trung các thiết bị.

**Lớp Light, Fan, AirConditioner (kế thừa từ Device)**:

* **Lớp Light**: Quản lý đèn, với các phương thức như *turnOnLed*(), *turnOffLed*(), *toggleLed*(), và *buttonPress*() để điều khiển đèn qua nút nhấn.
* **Lớp Fan**: Quản lý quạt, với các phương thức tương tự như turnOnFan(), *turnOffFan*(), *toggleFan*(), và *buttonPress*().
* **Lớp AirConditioner**: Quản lý điều hòa, với các phương thức như *setTemperature*() để cài đặt nhiệt độ, và *sendOtherStateToFirebase*() để gửi nhiệt độ lên Firebase.
* Các lớp này kế thừa từ Device, tận dụng các thuộc tính và phương thức chung, đồng thời triển khai phương thức ảo *FbUpdate*() để xử lý trạng thái từ Firebase (ví dụ: bật/tắt thiết bị khi status thay đổi).

**Lớp MQ2**: Quản lý cảm biến khí gas.

* Thuộc tính: *APin* (chân analog), *Dpin* (chân digital), *sensorThres* (ngưỡng phát hiện khí), dectectedGas (trạng thái phát hiện khí).
* Phương thức: *detectGas*() để phát hiện khí gas, *sentNotiToFb*() để gửi thông báo lên Firebase khi phát hiện khí, và *getDectectedGas*() để kiểm tra trạng thái.

**Lớp FM52**: Quản lý cảm biến tiệm cận.

* Thuộc tính: *pin* (GPIO pin), *state* (trạng thái phát hiện).
* Phương thức: *isDetectedClose*() để kiểm tra trạng thái tiệm cận.
* Lớp này được sử dụng để tự động bật/tắt đèn khi phát hiện vật thể gần.

**Lớp RFID\_Door**: Quản lý hệ thống cửa RFID.

* Thuộc tính: *SERVO* (chân servo), *state* (trạng thái cửa), *pw* (mật khẩu), *validUID* (danh sách UID thẻ hợp lệ).
* Phương thức: *openDoor*(), *closeDoor*(), *checkCard*() (kiểm tra thẻ RFID), *checkPassword*() (kiểm tra mật khẩu), và *updateToFB*() (cập nhật trạng thái cửa lên Firebase).
* Lớp này tích hợp với servo để điều khiển cửa và giao tiếp với Firebase để nhận lệnh mở/đóng cửa.

**Lớp OLED**: Quản lý màn hình OLED để hiển thị thông tin.

* Thuộc tính: *display* (đối tượng Adafruit\_SSD1306), *previousMillis* (thời gian cập nhật trước đó).
* Phương thức: *showText*(), *showPassword*(), *showCardAccepted*(), *showOpenDoor*(), và *goToHomeScreen*() để hiển thị các thông báo như trạng thái cửa, mật khẩu, hoặc thông báo lỗi.

**Lớp keyBoard**: Quản lý bàn phím ma trận để nhập mật khẩu.

* Thuộc tính: *rowPins*, *colPins* (các chân GPIO), *str* (chuỗi mật khẩu), *lastDebounceTime* (thời gian debounce).
* Phương thức: *getPressKey*() để lấy phím nhấn, *updateStr*() để cập nhật chuỗi mật khẩu, và *deleteStr*() để xóa chuỗi.

### **Lập trình thu thập dữ liệu từ cảm biến và nút nhấn**

- Gateway (ESP32) được lập trình để thu thập dữ liệu từ các cảm biến (cảm biến khí gas MQ2, cảm biến tiệm cận FM52, thẻ RFID) và nút nhấn (nút cảm ứng TTP223 Mini, bàn phím ma trận) nhằm giám sát trạng thái môi trường và nhận lệnh điều khiển thủ công từ người dùng. Dữ liệu thu thập được sử dụng để kích hoạt các hành động (như bật quạt khi phát hiện khí gas, bật/tắt đèn khi phát hiện vật thể, mở/đóng cửa khi quét thẻ RFID hoặc nhập mật khẩu) và gửi thông báo giám sát lên Firebase Realtime Database.

*- Triển khai thu thập dữ liệu từ cảm biến*

* **Cảm biến khí gas (MQ2)**:  
  Lớp MQ2 được sử dụng để thu thập dữ liệu từ cảm biến khí gas.

Phương thức *begin*(): Khởi tạo chân digital (Dpin) ở chế độ INPUT để đọc tín hiệu từ cảm biến.

Phương thức *detectGas*(): Đọc trạng thái từ chân digital (digitalRead(Dpin)) và kiểm tra giá trị analog (analogRead(APin)) nếu có. Nếu giá trị vượt ngưỡng (sensorThres = 200) hoặc tín hiệu digital ở mức LOW, cảm biến phát hiện khí gas.

Cơ chế thời gian: Sử dụng millis() để đảm bảo phát hiện khí gas liên tục trong 20ms trước khi xác nhận (dectectedGas = true), tránh phát hiện sai do nhiễu tín hiệu. Nếu không còn khí gas trong 5 giây, trạng thái được đặt lại và yêu cầu tắt quạt

Kết quả: Nếu phát hiện khí gas, gọi *sentNotiToFb*() để gửi thông báo "Có khí gas" lên Firebase và kích hoạt quạt thông gió.

* **Cảm biến tiệm cận (FM52)**:  
  Lớp FM52 được sử dụng để thu thập dữ liệu từ cảm biến tiệm cận (chân digital GPIO 35).

Phương thức begin(): Khởi tạo chân digital ở chế độ INPUT.

Phương thức isDetectedClose(): Đọc tín hiệu từ chân digital (digitalRead(pin)); nếu tín hiệu ở mức LOW, trả về true (phát hiện vật thể gần), nếu ở mức HIGH, trả về false (không có vật thể).

Kết quả: Dữ liệu từ cảm biến tiệm cận được sử dụng để tự động bật/tắt đèn trong hẻm.

* **Cảm biến RFID (MFRC522)**:  
  Lớp RFID\_Door được sử dụng để thu thập dữ liệu từ module RFID.

Phương thức init(): Khởi tạo giao tiếp SPI và module RFID (SPI.begin(), rfid.PCD\_Init()).

Phương thức *checkCard*(): Kiểm tra thẻ RFID mới (rfid.PICC\_IsNewCardPresent()) và đọc UID của thẻ (rfid.PICC\_ReadCardSerial()). So sánh UID với danh sách thẻ hợp lệ (validUID); nếu hợp lệ, trả về true, nếu không, trả về false.

Kết quả: Nếu thẻ hợp lệ, gọi openDoor() hoặc closeDoor() để mở/đóng cửa, đồng thời cập nhật trạng thái lên Firebase.

- **Thu thập dữ liệu từ nút nhấn**:

**Nút cảm ứng TTP223 Mini**:  
Nút cảm ứng TTP223 Mini được sử dụng để điều khiển các thiết bị như đèn và quạt. TTP223 Mini là module cảm ứng điện dung, xuất tín hiệu digital (HIGH khi chạm, LOW khi không chạm), không yêu cầu điện trở pull-up/pull-down bên ngoài nhờ tích hợp sẵn trong module. Dữ liệu từ các nút cảm ứng được thu thập thông qua hàm *getPressedButton*()

* Hàm buttonBegin(): Khởi tạo các chân nút cảm ứng ở chế độ INPUT. Vector buttonStates, lastButtonStates, và lastDebounceTime được khởi tạo để quản lý trạng thái và thời gian debounce của từng nút. TTP223 Mini được cấu hình để xuất tín hiệu HIGH khi chạm, nên không cần thêm điện trở pull-up/pull-down.
* Hàm *getPressedButton*(): Quét từng nút cảm ứng trong buttonPins, đọc trạng thái và áp dụng cơ chế debounce để tránh phát hiện sai do nhiễu tín hiệu hoặc chạm không ổn định. TTP223 Mini có độ nhạy cao, nên cơ chế debounce đặc biệt quan trọng để đảm bảo chỉ phát hiện một lần chạm duy nhất. Khi phát hiện chạm (chuyển từ LOW sang HIGH), trả về số chân GPIO của nút, nếu không có nút nào được chạm, trả về -1.
* Kết quả: Trong loop(), nếu getPressedButton() trả về 26, gọi fanBep.buttonPress() để bật/tắt quạt; nếu trả về 33, gọi lightBep.buttonPress() để bật/tắt đèn. Trạng thái thiết bị sau đó được gửi lên Firebase.

**Bàn phím ma trận**:  
Lớp keyBoard được sử dụng để thu thập dữ liệu từ bàn phím ma trận.

* Phương thức init(): Khởi tạo các chân hàng ở chế độ INPUT PULLDOWN và các chân cột ở chế độ OUTPUT, đặt trạng thái ban đầu là LOW.
* Phương thức getPressKey(): Quét bàn phím bằng cách kích hoạt từng cột (digitalWrite(colPins[i], HIGH)), đọc trạng thái từng hàng (digitalRead(rowPins[j])), và áp dụng cơ chế debounce (thời gian debounce 100ms) để tránh phát hiện sai. Nếu phát hiện phím nhấn, trả về ký tự tương ứng (như "1", "2", "L", "R"); nếu không, trả về "null".
* Phương thức updateStr() và getStr(): Cập nhật và lấy chuỗi mật khẩu người dùng nhập (ví dụ: "1234").
* Kết quả: Trong loop(), nếu getPressKey() trả về một ký tự (như "1"), chuỗi mật khẩu được cập nhật (updateStr()); nếu trả về "R", kiểm tra mật khẩu (door.checkPassword()); nếu trả về "L", xóa chuỗi mật khẩu (deleteStr()).

### **Lập trình điều khiển thiết bị (Actuator Unit)**

* **Điều khiển đèn (Light)**:  
  Lớp Light được sử dụng để điều khiển đèn
* Phương thức begin(): Khởi tạo chân GPIO ở chế độ OUTPUT và đặt trạng thái ban đầu là LOW (đèn tắt).
* Phương thức turnOnLed(): Đặt chân GPIO ở mức LOW để bật đèn, cập nhật trạng thái (mainState = true)
* Phương thức turnOffLed(): Đặt chân GPIO ở mức HIGH để tắt đèn, cập nhật trạng thái (mainState = false)
* Phương thức toggleLed(): Chuyển đổi trạng thái đèn
* Phương thức buttonPress(): Gọi toggleLed() khi nút cảm ứng TTP223 Mini được chạm, sau đó gọi sendStateToFirebase() để gửi trạng thái (on/off) lên Firebase.
* Phương thức FbUpdate(): Nhận trạng thái từ Firebase (như on/off) và gọi turnOnLed() hoặc turnOffLed() để cập nhật trạng thái đèn.
* **Điều khiển quạt thông gió (Fan)**:  
  Lớp Fan được sử dụng để điều khiển quạt. Quạt được kết nối qua transistor SN8050 (chân B nối với GPIO, chân E nối GND, chân C nối với quạt, VCC nối với chân còn lại của quạt).
* Phương thức begin(): Khởi tạo chân GPIO ở chế độ OUTPUT và đặt trạng thái ban đầu là LOW (quạt tắt).
* Phương thức turnOnFan(): Đặt chân GPIO ở mức HIGH để bật quạt, cập nhật trạng thái (mainState = true)
* Phương thức turnOffFan(): Đặt chân GPIO ở mức LOW để tắt quạt, cập nhật trạng thái (mainState = false)
* Phương thức toggleFan(): Chuyển đổi trạng thái quạt (bật nếu đang tắt, tắt nếu đang bật).
* Phương thức buttonPress(): Gọi toggleFan() khi nút cảm ứng TTP223 Mini được chạm, sau đó gọi sendStateToFirebase() để gửi trạng thái lên Firebase.
* Phương thức FbUpdate(): Nhận trạng thái từ Firebase và gọi turnOnFan() hoặc turnOffFan() để cập nhật trạng thái quạt.
* Kết quả:
* Quạt fanBep được điều khiển thủ công qua nút cảm ứng TTP223 Mini (GPIO 26) hoặc từ xa qua Firebase.
* Quạt cũng được điều khiển tự động dựa trên cảm biến khí gas MQ2 (bật khi phát hiện khí gas, tắt sau 5 giây nếu không còn khí gas).
* **Điều khiển điều hòa (AirConditioner)**:  
  Lớp AirConditioner được thiết kế để điều khiển điều hòa (chưa được triển khai cụ thể trong mã nguồn hiện tại).
* Phương thức turnOnAC(): Cập nhật trạng thái (mainState = true
* Phương thức turnOffAC(): Cập nhật trạng thái (mainState = false)
* Phương thức toggleAC(): Chuyển đổi trạng thái điều hòa.
* Phương thức setTemperature(): Cài đặt nhiệt độ (otherState)
* Phương thức sendStateToFirebase(): Gửi trạng thái (on/off) và nhiệt độ (otherState) lên Firebase.
* **Điều khiển servo (cửa RFID)**:  
  Lớp RFID\_Door được sử dụng để điều khiển servo mở/đóng cửa dựa trên thẻ RFID hoặc mật khẩu từ bàn phím ma trận hoặc từ ứng dụng điện thoại.
* Phương thức init(): Khởi tạo servo (myServo.attach(SERVO)) và đặt vị trí ban đầu là 0 độ (cửa đóng).
* Phương thức openDoor(): Quay servo đến 0 độ để mở cửa, cập nhật trạng thái
* Phương thức closeDoor(): Quay servo để đóng cửa, cập nhật trạng thái
* Phương thức updateToFB(): Gửi trạng thái cửa (on/off) lên Firebase
* Kết quả:
* Cửa được mở/đóng khi quét thẻ RFID hợp lệ (checkCard() trả về true).
* Cửa cũng được mở khi nhập mật khẩu đúng qua bàn phím ma trận (checkPassword() trả về true).
* Cửa có thể được điều khiển từ xa qua Firebase (nhận lệnh on/off từ StreamDoorData()).
* **Tích hợp với các lớp OOP và Firebase**:

Các thiết bị được điều khiển dựa trên dữ liệu từ cảm biến (MQ2, FM52), nút cảm ứng (getPressedButton()), bàn phím ma trận (keyBoard), hoặc lệnh từ Firebase (StreamData(), StreamDoorData()).  
Việc tích hợp này giúp tách biệt logic điều khiển khỏi mã chính, tăng tính tái sử dụng và dễ bảo trì.

# **CHƯƠNG 5: ĐIỀU KHIỂN THIẾT BỊ BẰNG GIỌNG NÓI**

## **5.1 Tổng quan về điều khiển bằng giọng nói**

*Mục tiêu****:***

Triển khai chức năng điều khiển thiết bị trong nhà thông minh (như đèn, quạt, cửa) bằng giọng nói, nhằm tăng tính tiện lợi và hiện đại cho hệ thống. Người dùng có thể ra lệnh bằng giọng nói (ví dụ: "Bật đèn phòng bếp", "Bật đèn phòng khách") thay vì sử dụng nút cảm ứng hoặc ứng dụng.

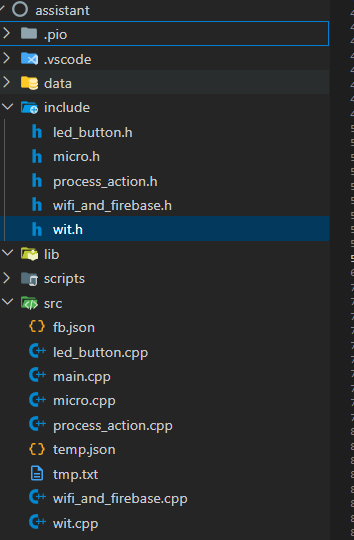
*Công nghệ và phần cứng*:

* ***Micro I2S INMP441 kết hợp ESP32*:** Micro MEMS kỹ thuật số, giao tiếp I2S, độ nhạy cao, phù hợp để thu âm giọng nói trong môi trường gia đình.
* ***Wit.ai****:* Dịch vụ xử lý ngôn ngữ tự nhiên (NLP) của Meta AI, hỗ trợ nhận diện giọng nói và phân tích ý định, trả về kết quả dưới dạng JSON.
* ***Firebase Realtime Database****:* Lưu trữ và đồng bộ trạng thái thiết bị.

*Quy trình tổng quan:*

* Thu âm giọng nói của người dùng bằng micro I2S INMP441 kết nối với ESP32.
* Gửi dữ liệu âm thanh (dưới dạng file WAV) lên Wit.ai để nhận diện và xử lý giọng nói.
* Nhận phản hồi từ Wit.ai dưới dạng JSON, phân tích ý định (intent) và các tham số (entities) từ câu lệnh.
* Cập nhật trạng thái thiết bị lên Firebase Realtime Database (ví dụ: bật/tắt đèn, mở/đóng cửa) dựa trên ý định đã phân tích.
* ESP32 đọc trạng thái từ Firebase và thực thi lệnh điều khiển thiết bị (đã trình bày ở Chương 4).

*Cấu trúc các module của chương trình:*

****

* Module led\_button sẽ để xử lý nút bấm kích hoạt nhận diện giọng nói và điều khiển trạng thái led để báo hiệu hành động đang được thực hiện (kết nối thành công với wifi và firebase thì sẽ sáng đèn, trong thời gian record thì sẽ nháy đèn với chu kỳ 100 ms).
* Module wit sẽ triển khai một class nhằm quản lý công việc gửi dữ liệu lên và nhận dữ liệu về, xử lý dữ liệu nhận được để lấy được nội dung điều khiển chính
* Module process\_action để ra quyết định điều khiển thiết bị (tìm và update dữ liệu lên cơ sở dữ liệu chung)
* Module wifi\_and\_firebase dùng để kết nối wifi và các hàm liên quan đến Firebase (upload, download, fetch, init).
* Module micro để kết nối với micro i2s và trả về 1 buffer chứa dữ liệu âm thanh và xử lý chèn header để trở thành file .wav trước khi đẩy lên wit.ai

## **5.2 Cấu hình phần cứng cho điều khiển bằng giọng nói**

### **5.2.1 Giới thiệu về giao thức I2S**

*Giới thiệu chung:*

* Là giao thức truyền dữ liệu âm thanh số giữa các IC (vi điều khiển, DAC, ADC, codec…).
* Được phát triển bởi Philips (NXP), chuyên dùng cho tín hiệu PCM (Pulse Code Modulation).
* Không truyền dữ liệu điều khiển, chỉ truyền dữ liệu âm thanh.
* Thư viện ***<driver/i2s.h>***

*Tín hiệu chính trong I2S:*

* SCK (Serial Clock): Xung clock truyền dữ liệu.
* WS (Word Select): Chọn kênh trái/phải (Left/Right).
* SD (Serial Data): Dữ liệu âm thanh số (truyền từng bit).
* MCLK (Master Clock): Xung nhịp chính (tùy chọn – dùng cho một số DAC).

*Cách hoạt động:*

* Master phát xung SCK và WS, Slave nhận và truyền dữ liệu theo đó.
* Dữ liệu âm thanh truyền theo từng bit, MSB trước.
* Stereo audio được phân biệt qua WS (0: Left, 1: Right).

*Ưu điểm:*

* Đồng bộ tốt, chuyên dụng cho âm thanh.
* Hỗ trợ nhiều độ phân giải âm thanh (16, 24, 32 bit).
* Phù hợp với thiết bị âm thanh nhúng (micro, loa, codec…).

*Hạn chế:*

* Không có truyền điều khiển, không kiểm tra lỗi.
* Khoảng cách truyền ngắn.
* Cần đồng bộ chính xác giữa các thiết bị.

*Ứng dụng thực tế:*

* Truyền âm thanh từ micro I2S (INMP441) về vi điều khiển (ESP32).
* Kết nối với DAC (MAX98357A) để phát âm thanh ra loa.
* Dùng trong loa thông minh, trợ lý ảo, nhận diện giọng nói.

### **5.2.2 Micro I2S INMP441 kết nối với ESP32.**

* SD (Serial Data): Kết nối với GPIO 32 của ESP32 (dữ liệu âm thanh).
* SCK (Serial Clock): Kết nối với GPIO 14 (đồng hồ I2S).
* WS (Word Select): Kết nối với GPIO 15 (chọn kênh trái/phải, INMP441 hoạt động ở chế độ mono nên chỉ dùng một kênh).
* VCC: Kết nối với 3.3V của ESP32.
* GND: Kết nối với GND của ESP32.
* L/R: Kết nối với GND để chọn kênh trái (mono).

## **Lập trình thu âm và xử lý tín hiệu âm thanh**

*Khởi tạo giao tiếp I2S trên ESP32:*

* Sử dụng thư viện I2S (driver/i2s.h) để cấu hình giao tiếp với INMP441.
* Cấu hình I2S:
* Tần số lấy mẫu: 16kHz (phù hợp với giọng nói và yêu cầu của Wit.ai).
* Độ phân giải: 16-bit (để đảm bảo chất lượng âm thanh).
* Chế độ: I2S Master Mode, mono (chỉ sử dụng kênh trái).

*Thu âm giọng nói:*

* Phương pháp thu âm:
* Thu âm trong 3 giây (đủ để người dùng nói một câu lệnh ngắn như "Bật đèn phòng bếp").
* Đọc dữ liệu âm thanh từ I2S buffer bằng i2s\_read().
* Lưu dữ liệu âm thanh vào bộ đệm tạm (buffer) dưới dạng mảng 16-bit PCM (Pulse Code Modulation).
* Tạo file WAV:

+ Thêm header WAV (44 byte) vào dữ liệu âm thanh để tạo file WAV, bao gồm các thông tin như tần số lấy mẫu (16kHz), số kênh (mono), độ phân giải (16-bit).

**A screen shot of a computer

AI-generated content may be incorrect.**

## **Gửi dữ liệu âm thanh lên Wit.ai và nhận phản hồi**

**5.4.1 Giới thiệu về Wit.ai**

- **Tổng quan về Wit.ai**:  
Wit.ai là một nền tảng xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP) được phát triển bởi Meta AI, cung cấp dịch vụ nhận diện và phân tích giọng nói cũng như văn bản để chuyển đổi chúng thành các lệnh hoặc ý định có thể hiểu được bởi máy tính. Wit.ai được thiết kế để hỗ trợ các ứng dụng thông minh, bao gồm trợ lý ảo, điều khiển thiết bị IoT, và chatbot, với ưu điểm là dễ tích hợp, miễn phí cho các dự án không thương mại, và hỗ trợ nhiều ngôn ngữ, bao gồm tiếng Việt. Trong hệ thống nhà thông minh của bạn, Wit.ai đóng vai trò trung tâm trong việc xử lý giọng nói thu từ micro I2S INMP441 qua ESP32, chuyển đổi thành các lệnh điều khiển thiết bị (vd: "Bật đèn phòng bếp") và gửi kết quả dưới dạng JSON để ESP32 phân tích và cập nhật trạng thái lên Firebase.

- **Cấu trúc và chức năng chính**:

* **Nhận diện giọng nói (Speech-to-Text)**:  
  Wit.ai có khả năng chuyển đổi tín hiệu âm thanh (file WAV hoặc streaming audio) thành văn bản, sử dụng các mô hình học máy được huấn luyện trước.
* **Phân tích ý định (Intent Recognition)**:  
  Wit.ai phân tích văn bản để xác định ý định của người dùng (ví dụ: on\_off) và các tham số liên quan (entities như state, device, room). Ví dụ, câu "Bật quạt phòng khách".
  + Intent: on\_off.
  + Entities: state = “bật”, device = "quạt", room = "phòng khách".
* **Trả về JSON**:  
  Wit.ai trả về kết quả dưới dạng JSON, cung cấp thông tin chi tiết về ý định và tham số cùng với độ tin cậy (confidence score).

A screenshot of a computer

AI-generated content may be incorrect.

* + 1. **Cấu hình Wit.ai:**
* Tạo ứng dụng trên Wit.ai (<https://wit.ai>) và cấu hình các ý định (intents) và tham số (entities):

+ Intents: on\_off

+ Entities: state (bật, tắt), device (đèn, quạt, điều hòa), room (phòng bếp, phòng khách).

+ Ví dụ: Câu lệnh "Bật đèn phòng bếp" sẽ được phân tích thành:

* + - State: bật.
    - Device: đèn.
    - Room: phòng bếp.
* Lấy API key của Wit.ai (Bearer Token) để gửi yêu cầu HTTP (dạng: Bearer XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX).

### **5.4.3 Gửi dữ liệu âm thanh lên Wit.ai:**

- Sử dụng thư viện HTTPClient trên ESP32 để gửi yêu cầu HTTP POST đến endpoint của Wit.ai (https://api.wit.ai/speech).

- Yêu cầu bao gồm:

+ Header:

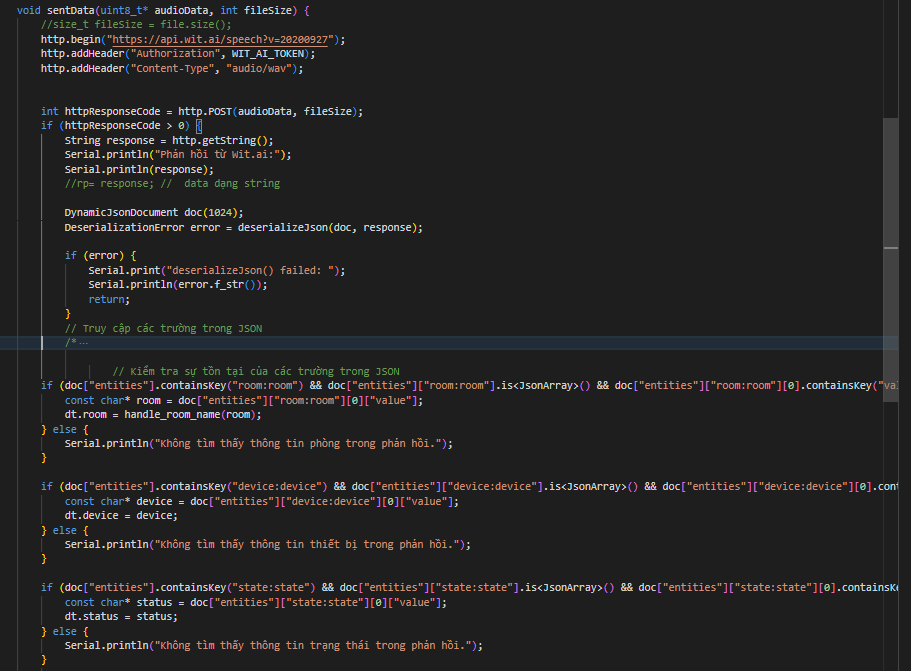
* Authorization: Bearer [WIT\_AI\_API\_KEY].
* Content-Type: audio/wav.

+Body: Dữ liệu âm thanh (file WAV, gồm header và dữ liệu PCM).

* + 1. **Nhận phản hồi từ Wit.ai:**
* Wit.ai trả về phản hồi dưới dạng JSON, chứa thông tin về ý định và tham số.
* Ví dụ: Đối với câu lệnh "Bật đèn phòng khách", phản hồi là 

## **5.5 Phân tích phản hồi và cập nhật trạng thái**

* + 1. **Phân tích phản hồi**
* Sử dụng thư viện ArduinoJson để phân tích chuỗi JSON từ Wit.ai.
* Trích suất thông tin về phòng, thiết bị, trạng thái từ chuỗi Json nhận lại từ Wit.ai



* + 1. **Cập nhật trạng thái điều khiển**
* Sau khi lấy được trạng thái, phòng, thiết bị thì ở thiết bị sẽ fetch nhánh user\_id để lấy data về người dùng (bao gồm thông tin về phòng, thiết bị,..). Sau đó sẽ tìm kiếm trong danh sách từ Firebase đến nội dung nhận từ dữ liệu giọng nói.
* Nếu thông tin được tìm thấy thì thiết bị sử dụng hàm uploadData() (đã triển khai ở chương 4) để gửi trạng thái thiết bị lên Firebase.

## **5.6 Kết quả**

- Module hoạt động gần như chính xác trong môi trường lý tưởng khi không có tiếng ồn và người dùng có thể nói trong vòng 2.5s thu âm

- Khi bắt đầu thu âm sẽ có âm thanh báo hiệu bắt đầu ghi âm khi ấn nút ghi âm, nếu sau khi âm thanh được gửi đi để xử lý và nhận được phản hồi có đầy đủ các trường dữ liệu (trạng thái, thiết bị, phòng) thì sẽ phát âm thanh 2 tiếng “bíp” để báo hiệu, nếu không thì sẽ phát 3 lần âm thanh để báo lỗi, để người dùng có thể thực hiện hành động lại

- Nếu đầy đủ các trường dữ liệu mà dữ liệu đó không được tìm thấy trên cơ sở dữ liệu của Firebase thì module sẽ bỏ qua lệnh cập nhật trạng thái lên Database.

CHƯƠNG 6: XÂY DỰNG VÀ KIỂM NGHIỆM MÔ HÌNH THỰC TẾ

## **6.1 Mô hình thực tế**

A white wall with many small objects

AI-generated content may be incorrect.

A rectangular object with wires and wires

AI-generated content may be incorrect.

A screenshot of a phone

AI-generated content may be incorrect.

## **6.2 Kết quả vận hành hệ thống và đánh giá hiệu quả hệ thống**

* Tổng quan về quá trình vận hành và đánh giá:  
  Hệ thống nhà thông minh đã được xây dựng và kiểm nghiệm với các chức năng chính bao gồm giám sát môi trường, điều khiển thiết bị thủ công, từ xa, và bằng giọng nói. Hiệu quả của hệ thống được đánh giá dựa trên kết quả vận hành thực tế và các tiêu chí như độ chính xác, thời gian phản hồi, cùng khả năng sử dụng tài nguyên.
* Kết quả vận hành các chức năng:
  + Giám sát môi trường: Hệ thống phát hiện sự hiện diện của khí gas, đo được nhiệt độ và vật thể di chuyển một cách ổn định
  + Điều khiển thủ công: Người dùng sử dụng nút bấm để bật/tắt đèn và quạt, cũng như nhập mật khẩu trên bàn phím để mở cửa. Với hệ thống nút bấm có bị chậm một thời gian ngắn do hàm chính còn phải lấy nhiều data từ Firebase. Còn thời gian nhập mật khẩu đến khi mở cửa gần như ngay lập tức
  + Điều khiển từ xa: Hệ thống nhận lệnh từ hệ thống lưu trữ dữ liệu và thực hiện điều khiển thiết bị từ xa. Một lệnh từ xa được gửi đến hệ thống, đèn chiếu sáng bật theo yêu cầu nhưng vẫn bị trễ một khoảng thời gian ngắn.
  + Điều khiển bằng giọng nói: Thiết bị thu âm nhận lệnh từ người dùng, xử lý và thực thi trong thời gian khoảng 8 giây, do thời gian thu âm, gửi dữ liệu lên và qua mô hình AI để trích suất dữ liệu nên bị trễ đáng kể.
* Tình trạng hoạt động:
  + Hệ thống hoạt động ổn định trong phần lớn thời gian kiểm nghiệm.
  + Một số trường hợp lỗi xảy ra, như lệnh giọng nói bị nhận diện sai hoặc không nhận diện được khi môi trường có tiếng ồn lớn hoặc người dùng chưa đọc rõ ràng.
  + Kết nối không dây với hệ thống lưu trữ dữ liệu duy trì tốt, không có gián đoạn đáng kể trong suốt quá trình kiểm nghiệm.
* Đánh giá hiệu quả điều khiển:
  + Độ chính xác:
    - Điều khiển thủ công phản hồi chính xác với các lệnh từ nút bấm và bàn phím .
    - Điều khiển tự động hoạt động ổn định với cảm biến khí gas và cảm biến tiệm cận, kích hoạt quạt và đèn chính xác trong hầu hết các thử nghiệm.
    - Điều khiển từ xa thực thi chính xác trong phần lớn thời gian, với một số trường hợp trễ do kết nối không ổn định.
    - Điều khiển bằng giọng nói hoạt động tốt trong môi trường yên tĩnh, nhưng hiệu quả giảm khi có tiếng ồn.
* Ưu điểm của hệ thống:
  + Hỗ trợ đa dạng phương thức điều khiển (thủ công, tự động, từ xa, giọng nói), mang lại sự linh hoạt cho người dùng.
  + Khả năng đồng bộ trạng thái với hệ thống lưu trữ dữ liệu giúp giám sát và điều khiển từ xa hiệu quả.
  + Giao diện giọng nói mang lại trải nghiệm hiện đại, đặc biệt hữu ích cho những người có hạn chế về vận động.
* Hạn chế
  + Độ chính xác giọng nói giảm trong môi trường ồn ào, gây nhầm lẫn trong một số lệnh.
  + Thời gian phản hồi của điều khiển qua app hay bằng giọng nói hơi chậm so với nhu cầu tức thời.

CHƯƠNG 7: KẾT LUẬN

Trong thời đại công nghệ phát triển nhanh chóng và nhu cầu nâng cao chất lượng cuộc sống của mọi người, nghiên cứu và phát triển hệ thống nhà thông minh trở nên rất cần hiết. Đề tài đã đáo ứng được yêu cầu và mục tiêu đặt ra ban đầu.

Kết quả đạt được:

* Xây dựng được phần ứng dụng thân thiện với người dùng, người dùng có thể tuỳ chỉnh phù hợp với từng yêu cầu thiết kế mà không cần phải thay đổi mã nguồn
* Phần nhúng trong phần cứng được thiết kế dựa trên hướng đối tượng nên có thể dễ dàng quản lý, thêm các thiết bị mà không cần quá nhiều thời gian để code lại từ đầu
* Hệ thống có độ trễ thấp và độ bảo mật cao, có thể điều khiển các thiết bị trong nhà như cửa, đèn, quạt, điều hoà,… và nhận được các thông báo về tình hình ngôi nhà.

TÀI LIỆU THAM KHẢO