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

Báo cáo cuối kỳ

Tên đề tài: Hệ thống đèn thông minh

Môn: IoT và ứng dụng

GV: Kim Ngọc Bách

Thành viên trong nhóm:

Nguyễn Việt Quang B22DCCN650

Phạm Công Minh B22DCCN542

Phạm Hải Dương B22DCCN169

Phùng Trung Kiên B22DCCN433

Hà Nội 2025

Mục lục

1 Giới thiệu đề tài

Trong bối cảnh công nghệ IoT (Internet of Things) ngày càng phát triển, việc tự động hóa các thiết bị điện trong gia đình trở nên phổ biến và cần thiết. Đề tài "Hệ thống đèn thông minh sử dụng ESP32" nhằm xây dựng một hệ thống điều khiển đèn tự động, có thể bật/tắt thông qua web hoặc dựa vào cảm biến chuyển động. Hệ thống sử dụng vi điều khiển ESP32 kết nối với máy chủ qua giao thức MQTT, giúp việc điều khiển và giám sát đèn trở nên linh hoạt, tiết kiệm năng lượng và tiện lợi cho người dùng.

Các công nghệ, lý thuyết sẽ áp dụng cho dự án

Phần cứng:

- ESP32: Vi điều khiển tích hợp WiFi, đóng vai trò trung tâm điều khiển và kết nối Internet.
- LED: Mô phỏng đèn trong mô hình, điều khiển bật/tắt thông qua GPIO của ESP32.
- Điện trở (220Ω – 330Ω): Giới hạn dòng điện để bảo vệ LED, đảm bảo hoạt động ổn định.
- Cảm biến chuyển động
- Mosfet

Phần mềm:

- Ngôn ngữ lập trình: Arduino C/C++ để lập trình cho ESP32.
- Giao tiếp: MQTT và HTTP để truyền dữ liệu giữa ESP32 và server.
- Server / Backend: Flask
- Cơ sở dữ liệu: Postgres để lưu thông tin thiết bị và trạng thái đèn.

Lý thuyết liên quan:

- Internet of Things (IoT): Kết nối và điều khiển thiết bị qua mạng Internet.
- Giao thức MQTT / REST API: Truyền dữ liệu giữa thiết bị và máy chủ.
- Tự động hóa dựa trên cảm biến: Đưa ra quyết định bật/tắt dựa trên dữ liệu môi trường.

2 Phân tích yêu cầu

2.1 Mục tiêu & phạm vi hệ thống

Vấn đề thực tế:
Người dùng thường quên tắt đèn khi ra khỏi phòng gây lãng phí điện năng và giảm tuổi thọ thiết bị.

Mục tiêu hệ thống IoT:

- Tự động bật/tắt đèn dựa trên cảm biến chuyển động.
- Thiết lập lịch trình bật tắt
- Cho phép người dùng điều khiển, giám sát đèn qua ứng dụng di động hoặc web.
- Tiết kiệm điện năng, nâng cao tiện nghi sinh hoạt.

Phạm vi triển khai:

- Môi trường: nhà ở, văn phòng, lớp học.
- Quy mô: 1–30 đèn thông minh, mỗi đèn gắn 1 cảm biến (chuyển động).
- Bộ xử lý trung tâm: ESP32 kết nối Wi-Fi, giao tiếp trực tiếp với server/cloud.
- Hệ thống thử nghiệm trong 1 căn hộ hoặc phòng học mẫu.

Tiêu chí thành công (KPIs):

Tiêu chí	Mục tiêu
Độ trễ điều khiển	< 2 giây
Tỷ lệ truyền dữ liệu thành công	≥ 98%
Độ ổn định hoạt động	≥ 99% uptime/tháng
Chi phí	<= 500k 1 thiết bị

2.2 Thu thập yêu cầu từ các bên liên quan

Nhóm liên quan	Yêu cầu chính
Người dùng cuối (chủ nhà, giáo viên, nhân viên)	- Bật/tắt đèn thủ công hoặc tự động. - Điều chỉnh độ sáng theo nhu cầu. - Giám sát trạng thái đèn qua điện thoại. - Nhận cảnh báo khi đèn không hoạt động hoặc mất kết nối. - Điều khiển bằng giọng nói: Cho phép người dùng ra lệnh bật/tắt đèn bằng giọng nói
Doanh nghiệp / Nhà phát triển	- Hệ thống dễ mở rộng thêm đèn. - Giao diện thân thiện, dễ bảo trì.
Kỹ thuật / IT	- Kết nối qua Wi-Fi chuẩn 2.4 GHz. - Giao thức MQTT hoặc HTTP để trao đổi dữ liệu. - Mã hóa dữ liệu - Hỗ trợ cập nhật firmware từ xa (OTA).

Phương pháp thu thập:
Phỏng vấn người dùng, khảo sát hành vi sử dụng điện, quan sát thực tế trong môi trường học/làm việc.

2.3 Yêu cầu chức năng (tính năng dự kiến triển khai)

a. Các chế độ hoạt động

Hệ thống hỗ trợ 3 chế độ vận hành, cho phép người dùng chuyển đổi linh hoạt tùy theo nhu cầu:

- **Chế độ Tự động (Auto Mode):**
  Hệ thống vận hành dựa hoàn toàn vào cảm biến chuyển động. Đèn sẽ bật khi phát hiện có người. Bộ hẹn giờ 30 giây bắt đầu đếm lùi sau lần phát hiện cuối cùng. Nếu trong thời gian này có chuyển động mới, bộ hẹn giờ sẽ được **reset** về 30 giây để đảm bảo đèn không tắt ngoài ý muốn. Đèn chỉ tắt khi bộ hẹn giờ kết thúc.
- **Chế độ Thủ công (Manual Mode):**
  Người dùng chủ động điều khiển đèn thông qua Web/App. Mọi tín hiệu từ cảm biến sẽ bị bỏ qua để tránh việc đèn tự bật hoặc tắt ngoài kiểm soát.
- **Chế độ Lịch trình (Schedule Mode):**
  Đèn hoạt động theo khung giờ đã được cài đặt trước (ví dụ: 18:00 bật – 06:00 tắt). Khi kết thúc thời gian của lịch trình, hệ thống sẽ tự động chuyển lại về **Auto Mode**.

b. Cảnh báo và hiển thị

- Hiển thị trạng thái đèn (On/Off).
- Cảnh báo lỗi đèn hoặc các trạng thái bất thường.

c. Cập nhật từ xa

- Hỗ trợ cập nhật firmware OTA thông qua Wi-Fi.

2.4 Yêu cầu phi chức năng

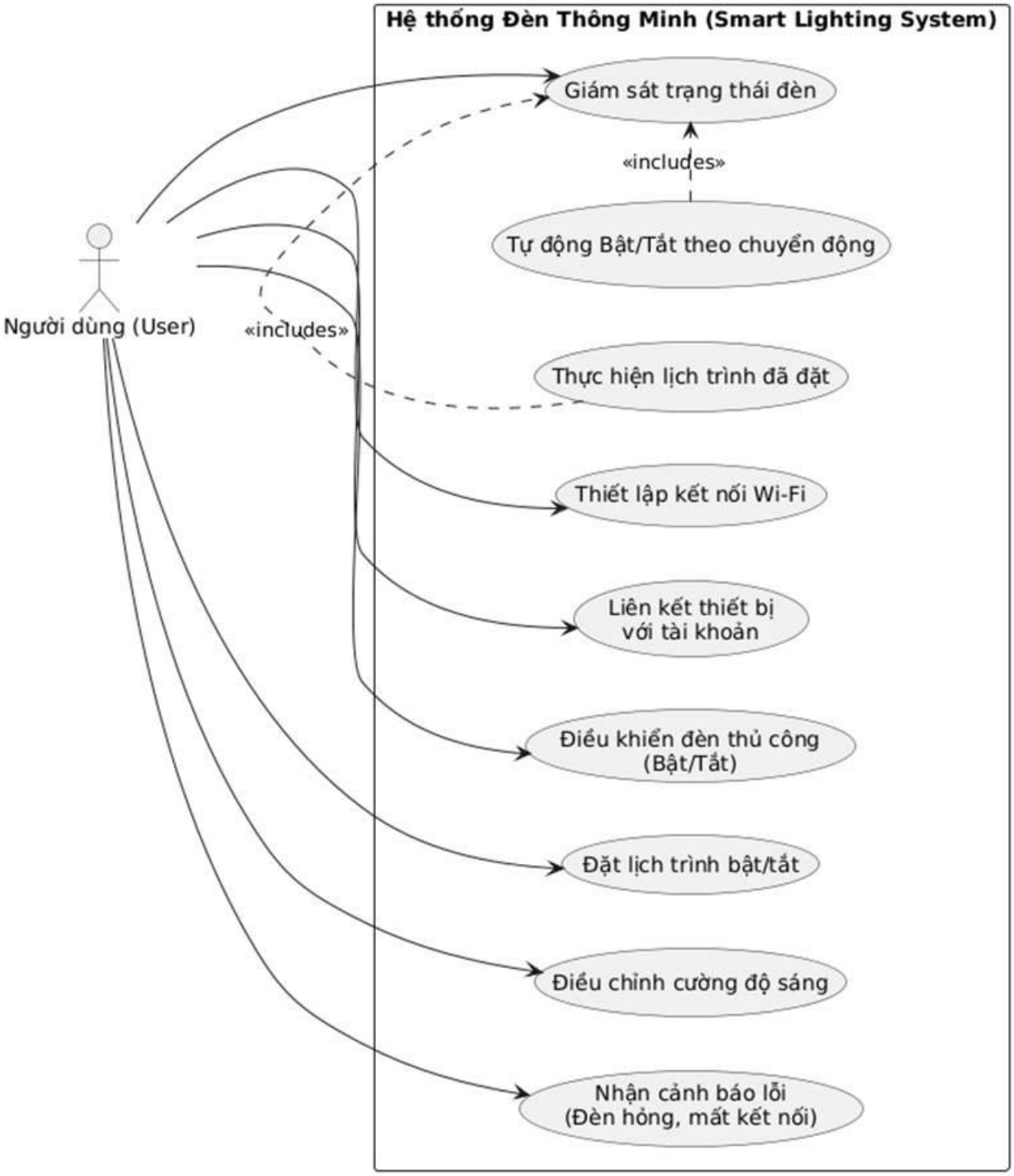
Nhóm yêu cầu	Mô tả
Hiệu năng	Độ trễ điều khiển ≤ 2 giây; cập nhật dữ liệu mỗi 5 giây.
Bảo mật	Xác thực thiết bị bằng token; phân quyền người dùng (admin/user).
Độ tin cậy	ESP32 tự phục hồi sau khi mất kết nối; lưu trạng thái đèn cục bộ khi không có Internet.
Khả năng mở rộng	Có thể thêm tới 50 thiết bị ESP32 trong cùng mạng Wi-Fi.
Chi phí & năng lượng	Tránh lãng phí điện khi người dùng quên tắt đèn và tăng tính tiện nghi trong sinh hoạt.

2.5 Phân tích ràng buộc kỹ thuật và môi trường

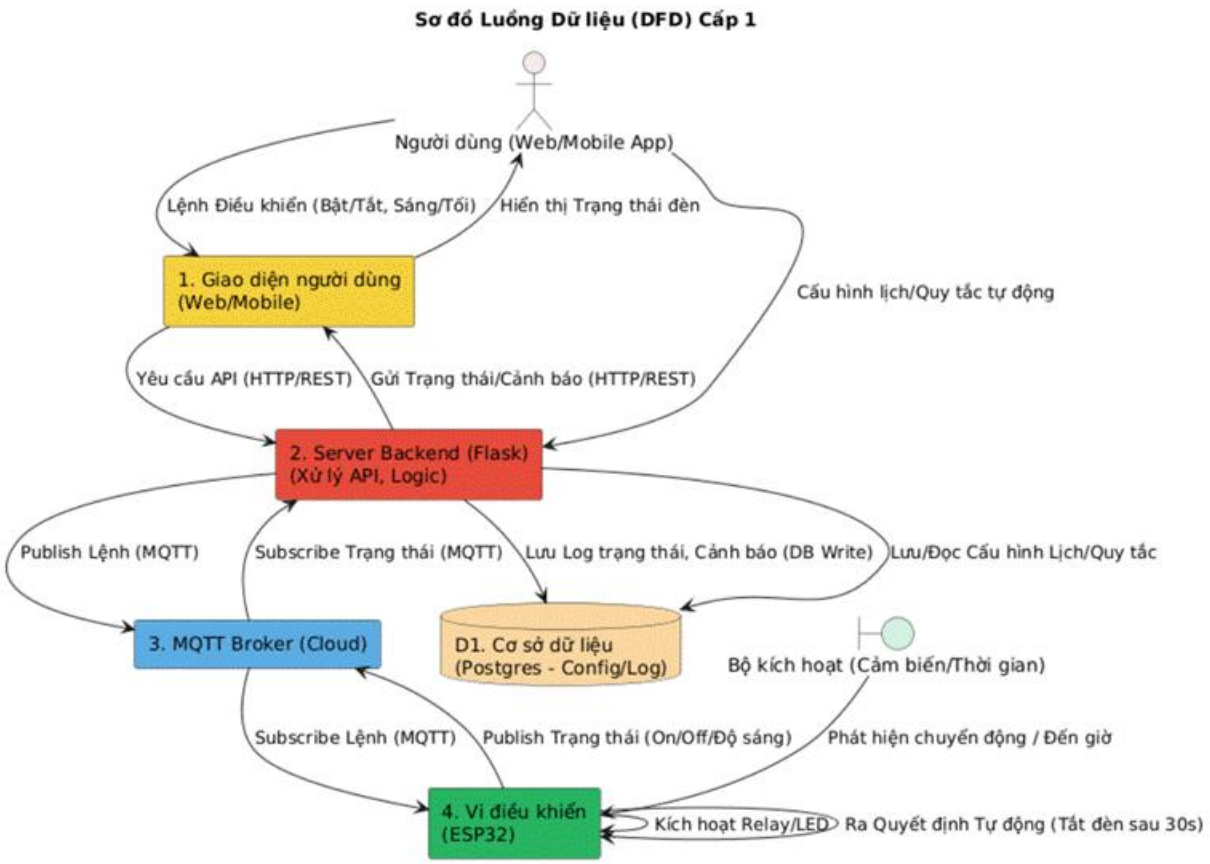
- Môi trường hoạt động: trong nhà (10–40°C, độ ẩm ≤ 85%).
- Kết nối: Wi-Fi 2.4GHz, hỗ trợ MQTT/HTTP với cloud.
- Nguồn cấp: điện lưới 220V AC (đèn), ESP32 dùng bộ nguồn 5V DC.
- Giới hạn phần cứng: RAM 520KB, Flash 4MB

2.6 Mô hình yêu cầu

Use case tổng quan



Luồng dữ liệu:



3 Thiết kế

3.1 Kiến trúc tổng thể của hệ thống

Mô tả chức năng từng khối

Thành phần	Mô tả
ESP32	Nhận lệnh điều khiển từ server, xử lý tín hiệu cảm biến, bật/tắt đèn
Cảm biến chuyển động (PIR)	Phát hiện người di chuyển, gửi tín hiệu Digital HIGH/LOW tới ESP32

Đèn LED / Mosfet	Thiết bị được điều khiển bật/tắt
Flask Backend	Nhận/gửi lệnh đến ESP32, lưu trạng thái, cung cấp API
Flask Frontend	Giao diện điều khiển và hiển thị trạng thái đèn

3.1.1 ESP32 (Firmware)

- Ngôn ngữ: Arduino C++ (hoặc MicroPython).
- Chức năng chính:
    - Kết nối Wi-Fi và MQTT Broker (HiveMQ).
    - Đăng ký (subscribe) topic điều khiển từ server, gửi (publish) dữ liệu trạng thái và cảm biến.
    - Điều khiển đèn thông qua Mosfet.

3.1.2 MQTT Broker

Quy tắc chung khi thiết kế MQTT

Cấu trúc Topic thống nhất

Sử dụng cấu trúc 4 phần:

home/<user_id>/<device_id>/<channel>

Trong đó:

Trường	Ý nghĩa
home	domain mặc định cho hệ thống

user_id	mã người dùng
device_id	mã thiết bị
channel	loại dữ liệu: cmd / state / sensor / log / mode

Danh sách các topic chính

Mục đích	Topic	Khi sử dụng
Gửi lệnh điều khiển	home/<uid>/<device>/cmd	Server → ESP32
Gửi trạng thái thiết bị	home/<uid>/<device>/state	ESP32 → Server
Gửi dữ liệu cảm biến	home/<uid>/<device>/sensor	ESP32 → Server
Gửi log sự kiện	home/<uid>/<device>/log	ESP32 → Server
Thay đổi chế độ hoạt động	home/<uid>/<device>/mode	Cả 2 chiều

Ví dụ đầy đủ:

home/123/light1/cmd

home/123/light1/state

home/123/light1/mode

home/123/light1/sensor

home/123/light1/log

Chuẩn hóa Payload JSON

Nguyên tắc chung

- Toàn bộ payload sử dụng JSON.
- Luôn kèm theo:
  timestamp theo UTC
  device_id
  user_id
- Các giá trị trạng thái phải đồng nhất toàn hệ thống:
  - state: "on" | "off"
  - mode: "auto" | "manual" | "schedule"

Payload điều khiển (Server → ESP32)

Topic:

home/<uid>/<device>/cmd

Payload mẫu (chuẩn):

```
{
  "command": "set",
  "state": "on",
  "brightness": 80,
  "mode": "manual",
  "timestamp": "2025-01-01T12:00:00Z"
}
```

Ý nghĩa trường:

Trường	Loại	Mô tả
command	string	luôn là "set"
state	string	"on" hoặc "off"
brightness	int	0–100
mode	string	chế độ hoạt động
timestamp	string	thời gian gửi

Payload trạng thái (ESP32 → Server)

Topic:

home/<uid>/<device>/state

Payload chuẩn:


```
{
  "device_id": "light1",
  "state": "on",
  "brightness": 75,
  "mode": "auto",
```

"last_motion": "2025-01-01T12:01:30Z",

"timestamp": "2025-01-01T12:01:30Z"

}

Payload dữ liệu cảm biến (ESP32 → Server)

Topic:

home/<uid>/<device>/sensor

Payload chuẩn:

{

"device_id": "light1",

"pir": 1,

"timestamp": "2025-01-01T12:01:00Z"

}

Payload thay đổi chế độ (Cả 2 chiều)

Topic:

home/<uid>/<device>/mode

Payload chuẩn:

{

"device_id": "light1",

"mode": "schedule",

"timestamp": "2025-01-01T12:02:00Z"

}

Payload log sự kiện (ESP32 → Server)

Topic:

home/<uid>/<device>/log

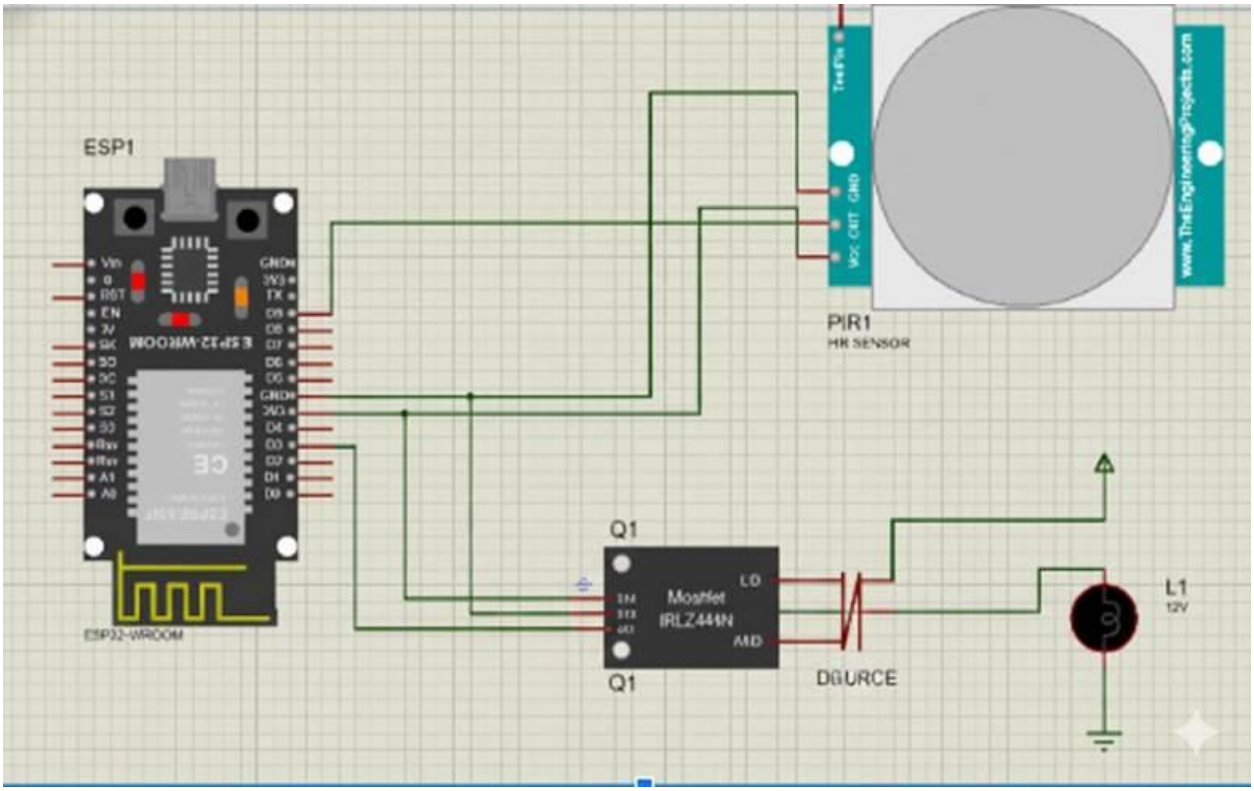
Payload chuẩn:


```
{
  "event": "auto_triggered",
  "message": "PIR detected motion",
  "device_id": "light1",
  "timestamp": "2025-01-01T12:01:10Z"
}
```


3.1.3 Flask (Web Server)

- Frontend (HTML/CSS/JS):
    - Hiển thị trạng thái đèn, nút bật/tắt, thanh trượt điều chỉnh độ sáng và lịch hẹn.
- Backend (Flask):
    - Xử lý route để hiển thị giao diện.
    - Cung cấp API để gửi lệnh MQTT, đọc trạng thái, và lấy log.
    - Hỗ trợ cập nhật realtime bằng WebSocket hoặc Server-Sent Events (SSE).

3.2 Thiết kế phần cứng



Hệ thống bao gồm 4 thành phần chính: ESP32, Cảm biến chuyển động PIR, Mosfet (N-Channel, ví dụ IRF520 hoặc tương đương) và tải là bóng đèn 12V. Các kết nối được thiết kế để ESP32 nhận tín hiệu từ PIR và điều khiển Mosfet làm công tắc bật/tắt đèn 12V.

3.2.1. Kết nối cảm biến PIR với ESP32

Kết nối này giữ nguyên vì PIR chỉ cung cấp tín hiệu điều khiển cho ESP32.

- PIR (Vcc) -> Nối vào chân 3.3V của ESP32 để cấp nguồn hoạt động.
- PIR (GND) -> Nối vào chân GND của ESP32.
- PIR (OUT) -> Nối vào chân Input của ESP32 (ví dụ: chân D13/GPIO13)
- Hoạt động: Khi PIR phát hiện chuyển động, chân OUT sẽ lên mức HIGH và được ESP32 đọc.

Khi PIR phát hiện chuyển động, chân OUT sẽ lên mức HIGH và được ESP32 đọc để bật Mosfet.

3.2.2. Kết nối ESP32 với Mosfet (Làm công tắc)

Mosfet (N-Channel) hoạt động như một công tắc điện tử điều khiển tải điện áp cao hơn (12V) bằng tín hiệu điện áp thấp (3.3V/5V) từ ESP32.

- Mosfet N-Channel có 3 chân chính: Gate (G), Drain (D), Source (S).

Chân Mosfet	Kết nối	Mục đích
Gate (G)	Nối vào chân điều khiển của ESP32 (ví dụ: D12 hoặc GPIO tương ứng).	Nhận tín hiệu PWM hoặc HIGH/LOW từ ESP32 để kích hoạt Mosfet.
Source (S)	Nối vào chân GND chung của hệ thống (GND của ESP32 và GND của nguồn 12V).	Hoàn thành mạch điều khiển tải.
Drain (D)	Nối vào Cực âm (-) của tải (bóng đèn 12V).	Điểm Mosfet điều khiển dòng điện chạy qua đèn.

3.2.3. Kết nối Mosfet với Đèn 12V (Mạch công suất)

Mosfet được dùng để đóng/ngắt mạch Mass (GND) của tải 12V.

- Nguồn dương 12V (+): Nối trực tiếp vào Cực dương (+) của bóng đèn 12V.
- Nguồn âm 12V (-): Nối vào chân Source (S) của Mosfet (đã nối chung với GND của ESP32).
- Đèn 12V (Cực âm -): Nối vào chân Drain (D) của Mosfet.

3.2.4. Sơ đồ hoạt động tổng quát

PIR phát hiện chuyển động -> OUT lên mức HIGH

ESP32 nhận tín hiệu -> xuất mức HIGH (3.3V) ra chân điều khiển Gate của Mosfet

Mosfet kích hoạt -> Dẫn điện từ Drain sang Source, hoàn thành mạch điện 12V -> đèn sáng.

Khi PIR không phát hiện nữa -> ESP32 tắt tín hiệu (xuất LOW 0V) ra Gate của Mosfet -> Mosfet ngưng dẫn điện -> đèn tắt

3.3 Thiết kế phần mềm

3.3.1 Luồng xử lý điều khiển thủ công và giám sát

Điều khiển thủ công (Web/App → Server → ESP32)

Bước 1 — Người dùng gửi lệnh điều khiển

Người dùng thao tác trên giao diện Web/App (Flask Frontend), ví dụ nhấn nút *Bật đèn* hoặc *Tắt đèn*.
Frontend gửi yêu cầu HTTP/REST đến Flask Backend, kèm theo thông tin:


```
{
  "device_id": "light1",
  "state": "on",
  "mode": "manual"
}
```


Bước 2 — Backend publish lệnh MQTT

Backend xử lý yêu cầu và gửi lệnh điều khiển đến thiết bị thông qua MQTT:

Topic:

home/<user_id>/<device_id>/cmd

Payload:


```
{
  "command": "set",
  "state": "on",
  "mode": "manual",
  "timestamp": "2025-01-01T12:00:00Z"
}
```


Bước 3 — ESP32 nhận lệnh điều khiển

ESP32 đã subscribe topic cmd, nên khi tin nhắn được publish:

- ESP32 nhận payload
- Giải mã JSON
- Điều khiển Mosfet để bật hoặc tắt đèn

Bước 4 — ESP32 gửi trạng thái mới về Server

Ngay sau khi xử lý lệnh, ESP32 gửi trạng thái thực tế của thiết bị:

Topic:

home/<uid>/<device>/state

Payload:


```
{
  "device_id": "light1",
  "state": "on",
  "mode": "manual",
  "brightness": 80,
  "timestamp": "2025-01-01T12:00:02Z"
}
```


Bước 5 — Backend cập nhật DB + đẩy realtime tới Frontend

Flask Backend (đã subscribe topic state) nhận được trạng thái mới:

- Cập nhật vào cơ sở dữ liệu (Postgres)
- Gửi dữ liệu theo thời gian thực lên giao diện Web bằng WebSocket hoặc SSE
- Frontend hiển thị ngay trạng thái mới (không cần refresh trang)

Giám sát trạng thái (ESP32 → Server → Frontend)

Bước 6 — ESP32 tự động gửi trạng thái

ESP32 có hai cơ chế gửi trạng thái:

1. Gửi ngay lập tức khi có thay đổi (bật/tắt, đổi mode…)
2. Gửi định kỳ (ví dụ mỗi 5 giây) để đảm bảo hệ thống luôn đồng bộ

Payload giống như ở bước 4.

Bước 7 — Backend đồng bộ sang Web/App

Backend nhận gói tin trạng thái → cập nhật CSDL → push realtime → người dùng thấy thông tin chính xác:

- Đèn đang ON/OFF
- Chế độ hoạt động (Auto / Manual / Schedule)
- Thời điểm phát hiện chuyển động cuối cùng (nếu Auto)
- Số liệu cảm biến (nếu có)

Luồng xử lý chế độ Đặt Lịch (Schedule Mode)

Chế độ đặt lịch cho phép hệ thống tự động bật/tắt đèn vào các thời điểm được cấu hình trước trên Web/App.

Luồng xử lý bao gồm ba phần chính: (1) tạo lịch – (2) xử lý lịch trên Backend – (3) cập nhật trạng thái thiết bị.

Bước 1 — Người dùng tạo lịch trên Web/App

Người dùng cấu hình các thông số:

- start_time: thời gian bật đèn
- end_time: thời gian tắt đèn
- repeat: none / daily / weekly
- brightness (tuỳ chọn)
- device_id

Frontend gửi yêu cầu REST API đến Backend:

```
{
  "device_id": "light1",
  "start_time": "18:00",
  "end_time": "23:00",
  "repeat": "daily",
  "brightness": 80
}
```

Bước 2 — Backend lưu lịch vào Cơ sở dữ liệu

Backend ghi dữ liệu vào bảng Schedules:

- Nếu là lịch mới → tạo bản ghi
- Nếu chỉnh sửa → update bản ghi
- Nếu xoá → đánh dấu is_active = false

Backend không gửi lệnh cho ESP32 ngay tại thời điểm tạo lịch.
ESP32 chỉ nhận lệnh khi lịch được kích hoạt tới thời điểm chạy.

Một module Scheduler (Cron job/Thread Timer) chạy trên Backend, thực hiện kiểm tra lịch định kỳ (ví dụ mỗi 1 phút).

Bước 3 — Scheduler kiểm tra thời điểm hiện tại

Backend kiểm tra trong DB:

- Lịch nào đang active?
- Lịch nào đến giờ bật/tắt đèn?
- Nếu repeat = daily/weekly → tự động reset vào ngày tiếp theo.

Bước 4 — Backend gửi lệnh MQTT khi đến giờ bật/tắt

Khi đến start_time, Backend publish lệnh bật đèn:

Topic:

home/<uid>/<device>/cmd

Payload:

```
{
  "command": "set",
  "state": "on",
  "brightness": 80,
  "mode": "schedule",
```

"timestamp": "2025-01-01T18:00:00Z"

}

Khi đến end_time, Backend gửi lệnh tắt:

{

"command": "set",

"state": "off",

"mode": "schedule",

"timestamp": "2025-01-01T23:00:00Z"

}

Bước 5 — ESP32 nhận lệnh bật/tắt

ESP32 đã subscribe topic cmd, nên khi nhận lệnh:

- Parse JSON
- Bật/tắt đèn đúng theo trạng thái
- Chuyển mode sang "schedule"
- Ghi nhớ rằng lệnh đến từ Schedule, không phải Auto/PIR

Bước 6 — ESP32 gửi trạng thái mới

Sau khi xử lý lệnh, ESP32 publish trạng thái thực tế:

Topic:

home/<uid>/<device>/state

Payload:

{

"device_id": "light1",

"state": "on",

"mode": "schedule",

"brightness": 80,

"timestamp": "2025-01-01T18:00:01Z"

}

Bước 7 — Backend cập nhật DB + push realtime

Backend nhận trạng thái → cập nhật DB → đẩy realtime lên UI (WebSocket/SSE).

Trên giao diện Web, người dùng sẽ thấy:

- Đèn đang bật/tắt theo lịch
- Mode hiển thị: Schedule Mode
- Thời gian bật gần nhất

3.3.2 Thiết kế Module ESP32 Firmware

Firmware chịu trách nhiệm xử lý logic cảm biến và giao tiếp với Server.

Module	Mô tả Chức năng	Ngôn ngữ
WiFi & MQTT Client	Kết nối lại tự động khi mất kết nối. Đăng ký/Gửi tin nhắn MQTT. Sử dụng cơ chế *retry* cho kết nối.	Arduino C/C++
PIR Sensor Handler	Đọc tín hiệu Digital (HIGH/LOW) từ Cảm biến chuyển động (PIR). Khi phát hiện chuyển động (HIGH), gửi lệnh BẬT nếu đèn đang TẮT. Khi chuyển động dừng (LOW), bắt đầu hẹn giờ 30 giây để TẮT đèn (nếu không có chuyển động	Arduino C/C++

	mới).	
Mosfet/LED Control	Xử lý lệnh từ MQTT để điều khiển Mosfet qua GPIO, bao gồm bật/tắt đèn và thay đổi độ sáng bằng PWM.	Arduino C/C++
OTA Update	Cho phép nhận và cài đặt firmware mới qua Wi-Fi từ Server mà không cần cắm cáp	Arduino C/C++

3.3.3 Thiết kế Flask Backend (Server)

Flask Backend đóng vai trò trung tâm điều phối toàn bộ hệ thống IoT. Backend xử lý các yêu cầu từ người dùng, giao tiếp với MQTT Broker, quản lý cơ sở dữ liệu, thực thi lịch trình và cập nhật trạng thái theo thời gian thực.

Backend của hệ thống được chia thành bốn module chính: API Module, MQTT Client Module, Scheduler Module, và Notification Module.

API Module (HTTP/REST)

ackend cung cấp các endpoint REST để Frontend có thể giao tiếp với hệ thống.

Các API chính:

API	Phương thức	Chức năng

/api/device/command	POST	Nhận lệnh từ Frontend → publish lệnh MQTT đến thiết bị
/api/device/status	GET	Trả về trạng thái mới nhất của thiết bị từ DB
/api/schedule	POST	Tạo, chỉnh sửa hoặc xóa lịch trình của thiết bị
/api/logs	GET	Truy vấn nhật ký hoạt động

Luồng hoạt động:

1. Frontend gửi request HTTP.
2. Backend xử lý logic (validate input, kiểm tra quyền user)
3. Backend publish payload JSON qua MQTT.
4. Backend trả phản hồi cho Frontend.

API module đảm bảo giao tiếp giữa người dùng và hệ thống diễn ra dễ dàng và an toàn.

MQTT Client Module

Backend sử dụng MQTT Client (paho-mqtt) để giao tiếp với MQTT Broker.

Chức năng chính:

(1) Subscribe các topic quan trọng

Backend đăng ký các topic dạng:

home/+/+/state

home/+/+/sensor

home/+/+/log

→ Cho phép nhận trạng thái tất cả thiết bị theo mô hình động.

(2) Nhận trạng thái từ ESP32

Khi ESP32 gửi dữ liệu:

- Decode JSON
- Cập nhật bảng Devices
- Lưu bản ghi vào bảng Logs nếu cần
- Gửi realtime đến Frontend qua WebSocket/SSE

(3) Publish lệnh điều khiển

Khi API yêu cầu bật/tắt đèn hoặc đổi mode:

- MQTT Client tạo payload JSON (cmd)
- Publish vào topic home/<uid>/<device>/cmd

MQTT Client là cầu nối giữa Backend và hệ thống thiết bị.

Scheduler Module (Bộ xử lý lịch tự động)

Module Scheduler thực hiện bật/tắt đèn theo thời gian được cấu hình trong bảng Schedules.

Cơ chế hoạt động:

1. Scheduler chạy định kỳ (ví dụ mỗi 30 giây).
2. Lấy danh sách lịch is_active = true.
3. So sánh thời gian hiện tại với start_time và end_time.
4. Nếu đến thời điểm bật/tắt → gửi lệnh MQTT:

Lệnh bật:

```
{
  "command": "set",
  "state": "on",
  "mode": "schedule"
}
```

Lệnh tắt:


```
{
  "command": "set",
  "state": "off",
  "mode": "schedule"
}
```


5. Chuyển thông tin trạng thái mới lên DB và UI.

Ưu điểm của thiết kế:

- ESP32 không cần tự lưu lịch → firmware đơn giản.
- Lịch được quản lý tập trung tại Backend → dễ mở rộng.
- Tự động xử lý daily/weekly → không cần cấu hình lại.

Notification & Alert Module

Module này chịu trách nhiệm theo dõi tình trạng kết nối và cảnh báo.

Chức năng:

1. Theo dõi last_online của thiết bị
    - Backend cập nhật last_online mỗi khi nhận message từ ESP32.
    - Nếu quá 5 phút không nhận được tín hiệu → đánh dấu thiết bị *offline*.
2. Gửi cảnh báo
    - Gửi thông báo lên Frontend (popup/toast).
    - Ghi log lỗi vào bảng Logs.
    - (Tùy chọn) gửi Email hoặc Push Notification.
3. Giám sát trạng thái bất thường

- ESP32 gửi lỗi qua topic /log → Backend hiển thị trên giao diện.

3.3.4 Thiết kế cơ sở dữ liệu

Hệ thống sử dụng cơ sở dữ liệu quan hệ Postgres để lưu trữ thông tin người dùng, thiết bị, lịch trình và nhật ký hoạt động.

Mô Hình Quan Hệ Thực Thể (ERD - Entity-Relationship Diagram)

- Users: Thông tin người dùng hệ thống
- Devices: Thông tin các thiết bị đèn thông minh
- Schedules: Lịch tự động bật/tắt đèn theo thời gian
- Logs: Nhật ký hoạt động

Quan hệ giữa các thực thể:

- Users (1) ——< (n) Devices : Một người dùng có thể sở hữu nhiều thiết bị
- Devices (1) ——< (n) Schedules : Mỗi thiết bị có thể có nhiều lịch điều khiển.
- Devices (1) ——< (n) Logs : Mỗi thiết bị sinh ra nhiều bản ghi nhật ký

Thiết Kế Các Bảng Dữ Liệu

Bảng 1: Users

Tên cột	Kiểu dữ liệu	Mô tả
user_id	INT (PK, Auto Increment)	Mã định danh người dùng
username	VARCHAR(50)	Tên đăng nhập
password	VARCHAR(100)	Mật khẩu (đã mã hóa)
email	VARCHAR(100)	Email liên hệ

role	ENUM('admin', 'user')	Phân quyền người dùng
created_at	TIMESTAMP WITH TIME ZONE	Thời điểm tạo tài khoản

Bảng 2: Devices

Tên cột	Kiểu dữ liệu	Mô tả
device_id	INT (PK, Auto Increment)	Mã thiết bị
user_id	INT (FK → Users.user_id)	Người sở hữu thiết bị
device_name	VARCHAR(100)	Tên hiển thị của thiết bị
is_on	BOOLEAN	Trạng thái bật/tắt hiện tại của đèn. TRUE = ON, FALSE = OFF
mode	ENUM('auto', 'manual', 'schedule')	Chế độ hoạt động hiện tại của đèn
last_online	TIMESTAMP WITH TIME ZONE	Thời điểm cuối cùng thiết bị kết nối

brightness	INT	Cường độ sáng (0–100)

Bảng 3: Schedules

Tên cột	Kiểu dữ liệu	Mô tả
schedule_id	INT (PK, Auto Increment)	Mã lịch
device_id	INT (FK → Devices.device_id)	Thiết bị được áp dụng lịch
start_time	TIME	Thời gian bật đèn
end_time	TIME	Thời gian tắt đèn
repeat	ENUM('none', 'daily', 'weekly')	Kiểu lặp lại lịch
brightness	INT	Mức sáng của đèn (0–100%)

is_active	BOOLEAN	Trạng thái kích hoạt của lịch

Bảng 4 logs

Tên cột	Kiểu dữ liệu	Mô tả
log_id	INT (PK, Auto Increment)	Mã bản ghi nhật ký
device_id	INT (FK → Devices.device_id)	Thiết bị phát sinh sự kiện
event_type	ENUM('power_on', 'power_off', 'mode_change', 'schedule_trigger', 'error', 'brightness_change')	Loại sự kiện xảy ra
old_value	VARCHAR(100)	Giá trị cũ trước khi thay đổi (nếu có)
new_value	VARCHAR(100)	Giá trị mới sau khi thay đổi (nếu có)
description	TEXT	Mô tả chi tiết sự kiện
created_at	TIMESTAMP WITH TIME ZONE	Thời điểm ghi nhận sự kiện

4 Kết quả đạt được

Sau quá trình xây dựng và triển khai hệ thống điều khiển đèn thông minh sử dụng ESP32, MQTT và Flask Backend, nhóm đã hoàn thành các chức năng chính sau:

Điều khiển đèn thủ công qua Web/App

- Người dùng có thể bật/tắt đèn trực tiếp từ giao diện Web.
- Lệnh điều khiển được Backend gửi xuống ESP32 thông qua MQTT.
- Đèn phản hồi ngay sau khi nhận lệnh và cập nhật trạng thái lên giao diện.

Tự động bật/tắt đèn bằng cảm biến chuyển động (Auto Mode)

- Hệ thống bật đèn khi cảm biến PIR phát hiện chuyển động.
- Khi không còn chuyển động, đèn tự tắt sau thời gian trễ 30 giây.
- ESP32 gửi trạng thái mới về Backend để hiển thị lên Web.

Chế độ đặt lịch bật/tắt (Schedule Mode)

- Người dùng tạo lịch bật/tắt đèn theo thời gian mong muốn.
- Backend Scheduler kiểm tra lịch và tự động gửi lệnh MQTT đúng thời điểm.
- ESP32 chuyển sang mode “schedule” và thực thi lệnh một cách chính xác.

Đồng bộ trạng thái theo thời gian thực

- ESP32 gửi trạng thái đèn (on/off, mode, brightness…) lên topic `/state`.
- Backend cập nhật DB và đẩy trạng thái realtime lên giao diện.
- Giao diện luôn hiển thị đúng trạng thái thiết bị.

Gửi và ghi nhận dữ liệu cảm biến (sensor)

- ESP32 gửi dữ liệu PIR dạng JSON qua MQTT.
- Backend lưu thông tin vào DB và sử dụng để hiển thị/ghi log.

Theo dõi trạng thái thiết bị (online/offline)

- Backend lưu `last_online` mỗi khi ESP32 gửi bất kỳ tin nhắn nào.
- Khi thiết bị không gửi dữ liệu quá lâu → đánh dấu offline.

5 Hướng mở rộng

Nhận diện giọng nói / AI Assistant

- Điều khiển bằng Google Assistant hoặc Alexa
- Gửi lệnh qua MQTT hoặc REST API

Tích hợp OTA nâng cao

- Cho phép cập nhật firmware từ xa
- Tự động kiểm tra và tải firmware mới
- Log trạng thái cập nhật để tránh lỗi gián đoạn

Mở rộng nhiều thiết bị và nhiều phòng

- Mỗi người dùng quản lý nhiều đèn trong từng phòng
- Tạo group control: bật/tắt theo nhóm