**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

Logo, company name

Description automatically generated

**BÀI TẬP LỚN**

**TÊN HỌC PHẦN: TỔNG QUAN IOT VÀ LẬP TRÌNH NHÚNG**

**ĐỀ TÀI: BÃI ĐỖ XE THÔNG MINH**

**Giáo viên hướng dẫn: ThS Nguyễn Văn Nhân**

**ThS Lê Thái Bảo**

**Sinh viên thực hiện:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Stt** | **Mã sv** | **Họ và tên** | **Lớp** |
| 1 | 1771020477 | Ngô Văn Minh | CNTT 17-01 |
| 2 | 1771020051 | Nguyễn Trung Anh | CNTT 17-01 |
| 3 | 1771020387 | Lý Gia Khánh | CNTT 17-01 |
| 4 | 1771020251 | Vũ Văn Hậu | CNTT 17-01 |

**Hà Nội, năm 2025**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC ĐẠI NAM**

Logo, company name

Description automatically generated

**BÀI TẬP LỚN**

**TÊN HỌC PHẦN: TỔNG QUAN VỀ IOT VÀ**

**LẬP TRÌNH NHÚNG**

**ĐỀ TÀI: BÃI ĐỖ XE THÔNG MINH**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| STT | Mã Sinh Viên | Họ và Tên | Ngày Sinh | Điểm | |
| Bằng Số | Bằng Chữ |
| 1 | 1771020477 | Ngô Văn Minh | 17/06/2005 |  |  |
| 2 | 1771020051 | Nguyễn Trung Anh | 30/03/2005 |  |  |
| 3 | 1771020387 | Lý Gia Khánh | 18/02/2005 |  |  |
| 4 | 1771020251 | Vũ Văn Hậu | 10/08/2005 |  |  |

***CÁN BỘ CHẤM THI***

**Hà Nội, năm 2025**

# LỜI NÓI ĐẦU

Trong bối cảnh cuộc cách mạng công nghiệp 4.0 phát triển mạnh mẽ, Internet of Things (IoT) và lập trình nhúng trở thành nền tảng quan trọng cho các hệ thống tự động hóa và thông minh. IoT cho phép các thiết bị trong môi trường thực kết nối với nhau và với Internet, từ đó thu thập dữ liệu, xử lý và đưa ra các phản hồi theo thời gian thực. Nhờ khả năng kết nối và trao đổi thông tin liên tục, IoT mang đến nhiều ứng dụng hữu ích trong đời sống như nhà thông minh, nông nghiệp thông minh, giám sát an ninh, và đặc biệt là quản lý bãi đỗ xe.

Lập trình nhúng đóng vai trò cốt lõi trong việc tạo nên khả năng hoạt động của các thiết bị IoT. Các hệ thống nhúng là những bộ điều khiển nhỏ gọn, tích hợp phần cứng và phần mềm, cho phép thiết bị thực hiện một chức năng cụ thể một cách chính xác và ổn định. Thông qua việc lập trình cho vi điều khiển (như Arduino, ESP32), hệ thống có thể đọc dữ liệu từ cảm biến, điều khiển các thiết bị như động cơ, màn hình, đèn báo và giao tiếp với các hệ thống khác.

Khi kết hợp IoT và lập trình nhúng vào mô hình bãi đỗ xe thông minh, hệ thống có thể tự động nhận diện xe vào – ra, ghi nhận số lượng xe còn trống, điều khiển cổng barrier, và hiển thị thông tin theo thời gian thực. Điều này không chỉ nâng cao hiệu quả vận hành mà còn giảm thiểu sai sót, hỗ trợ quản lý thuận tiện và mang lại trải nghiệm tốt hơn cho người dùng. Đồng thời, việc triển khai đề tài giúp sinh viên làm quen với các công nghệ hiện đại, phát triển kỹ năng lập trình, phân tích hệ thống và ứng dụng thực tế của IoT trong đời sống.

**MỤC LỤC**

[LỜI NÓI ĐẦU 3](#_Toc215080882)

[CHƯƠNG 1. TỔNG QUAN VỀ HỆ THỐNG: 10](#_Toc215080883)

[1.1. Giới thiệu đề tài và bài toán thực tế: 10](#_Toc215080884)

[1.2. Mục tiêu và yêu cầu của hệ thống: 11](#_Toc215080885)

[1.3. Phạm vi thực hiện dự án: 12](#_Toc215080886)

[CHƯƠNG 2. THIẾT KẾ HỆ THỐNG: 15](#_Toc215080887)

[2.1. Phân tích và lựa chọn phần cứng: 15](#_Toc215080888)

[2.1.1. Vi điều khiển Arduino: 15](#_Toc215080889)

[2.1.2. Cảm biến hồng ngoại IR: 16](#_Toc215080890)

[2.1.3. Động cơ servo SG90: 17](#_Toc215080891)

[2.1.4. Màn hình LCD 16x2 I2C: 18](#_Toc215080892)

[2.1.5. ESP32-Cam: 20](#_Toc215080893)

[2.1.6. Cảm biến khí ga: 22](#_Toc215080894)

[2.1.7. Module RFID RC522: 23](#_Toc215080895)

[2.2. Thiết kế sơ đồ khối hệ thống: 24](#_Toc215080896)

[2.3. Thiết kế sơ đồ mạch điện: 27](#_Toc215080897)

[2.3.1. Bảng phân bổ chân kết nối chi tiết: 27](#_Toc215080898)

[2.3.2. Sơ đồ kết nối nguồn điện: 29](#_Toc215080899)

[2.3.3. Sơ đồ nguyên lý kết nối: 30](#_Toc215080900)

[2.4. Thiết kế thuật toán và lưu đồ chương trình: 31](#_Toc215080901)

[CHƯƠNG 3. TRIỂN KHAI VÀ KẾT QUẢ: 37](#_Toc215080902)

[3.1. Lập trình và cài đặt hệ thống: 37](#_Toc215080903)

[3.1.1. Code arduno. 37](#_Toc215080904)

[3.1.2. Code esp32wroom: 42](#_Toc215080905)

[3.1.3. Code camera: 57](#_Toc215080906)

[3.1.4. Giao diện: 61](#_Toc215080907)

[3.2. Lắp ráp mạch và kiểm tra hoạt động: 75](#_Toc215080908)

[3.2.1. lắp ráp mạch: 75](#_Toc215080909)

[3.2.2. Kiểm tra hoạt động: 76](#_Toc215080910)

[3.3. Chạy thử nghiệm và đánh giá: 79](#_Toc215080911)

[3.3.1. Giao diện web: 79](#_Toc215080912)

[3.3.2. đánh giá: 87](#_Toc215080913)

[KẾT LUẬN 90](#_Toc215080914)

[DANH MỤC TÀI LIỆU THAM KHẢO 91](#_Toc215080915)

**MỤC LỤC HÌNH ẢNH**

[Hình 1. Vi điều khiển Arduino 15](#_Toc215082910)

[Hình 2. Cảm biến hồng ngoại IR 16](#_Toc215082911)

[Hình 3. Động cơ servo SG90 17](#_Toc215082912)

[Hình 4. Màn hình LCD 16x2 I2C 19](#_Toc215082913)

[Hình 5. ESP32-Cam 20](#_Toc215082914)

[Hình 6. Cảm biến khí ga 22](#_Toc215082915)

[Hình 7. Module RC522 và bộ thẻ RFID 23](#_Toc215082916)

[Hình 8. Sơ đồ khối hệ thống bãi đỗ xe thông minh IoT 25](#_Toc215082917)

[Hình 9. Lưu đồ hoạt động Arduino Uno (I2C Slave) 32](#_Toc215082918)

[Hình 10. Lưu đồ thuật toán điều khiển hệ thống bãi đỗ xe thông minh IoT 33](#_Toc215082919)

[Hình 11. Sơ đồ lắp đặt mạch bãi đỗ xe Arduino 75](#_Toc215082920)

[Hình 12. Mô phỏng giao diện web 80](#_Toc215082921)

[Hình 13. Mô phỏng hiển thị lịch sử 81](#_Toc215082922)

[Hình 14. Mô phỏng quản lý xe 82](#_Toc215082923)

[Hình 15. Mô phỏng hiển thị báo cáo doanh thu 83](#_Toc215082924)

[Hình 16. Mô phỏng khi xe vào 83](#_Toc215082925)

[Hình 17. Mô phỏng khi xe vào 84](#_Toc215082926)

[Hình 18. Mô phỏng cơ chế quẹt thẻ 85](#_Toc215082927)

[Hình 19. Mô phỏng khi cửa đã quẹt xong và tự mở 85](#_Toc215082928)

[Hình 20. Mô phỏng dữ liệu xe vào được lưu trữ 86](#_Toc215082929)

[Hình 21. Mô phỏng cơ chế quẹt thẻ cho xe ra 86](#_Toc215082930)

[Hình 22. Mô phỏng cơ chế mở cổng tự động 87](#_Toc215082931)

[Hình 23. Hiển thị xe ra thành công 87](#_Toc215082932)

**MỤC LỤC BẢNG**

[Bảng 1. Chức năng và kết nối các khối trong hệ thống 26](#_Toc215083763)

[Bảng 2. Kết nối Arduino Uno R3 28](#_Toc215083764)

[Bảng 3. Kết nối ESP32 DevKit V1 28](#_Toc215083765)

[Bảng 4. Phân bổ nguồn điện hệ thống 29](#_Toc215083766)

[Bảng 5. Chức năng các hàm trong hệ thống 34](#_Toc215083767)

[Bảng 6. Trạng thái hệ thống bãi đỗ xe IoT 35](#_Toc215083768)

[Bảng 7. Đấu nối cảm biến IR với Arduino Uno 76](#_Toc215083769)

**BẢNG CÁC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **STT** | **Từ viết tắt** | **Viết đầy đủ** |
| **1** | **IoT** | **Internet of Things**  **(Internet vạn vật)** |
| **2** | **MCU** | **Microcontroller Unit**  **(Vi điều khiển như ESP32, Arduino)** |
| **3** | **ESP** | **Espressif – Dòng vi điều khiển hỗ trợ WiFi/Bluetooth (ESP8266, ESP32)** |
| **4** | **LCD** | **Liquid Crystal Display**  **(Màn hình tinh thể lỏng – VD: LCD 16x2)** |
| **5** | **API** | **Application Programming Interface**  **(Giao diện lập trình ứng dụng)** |
| **6** | **UART** | **Universal Asynchronous Receiver/Transmitter**  **(Giao tiếp nối tiếp không đồng bộ)** |
| **7** | **PWM** | **Pulse Width Modulation**  **(Điều chế độ rộng xung – điều khiển servo/motor)** |
| **8** | **GPIO** | **General Purpose Input Output**  **(Chân vào/ra đa chức năng)** |
| **9** | **WiFi** | **Wireless Fidelity**  **(Kết nối mạng không dây)** |
| **10** | **RFID** | **Radio Frequency Identification**  **(Nhận dạng bằng sóng vô tuyến – dùng quẹt thẻ)** |
| **11** | **HTTP** | **HyperText Transfer Protocol**  **(Giao thức truyền dữ liệu web)** |
| **12** | **HTTPS** | **HyperText Transfer Protocol Secure**  **(HTTP bảo mật bằng SSL/TLS)** |
| **13** | **MQTT** | **Message Queuing Telemetry Transport**  **(Giao thức truyền tin nhẹ trong IoT)** |
| **14** | **AI** | **Artificial Intelligence**  **(Trí tuệ nhân tạo – dùng nhận diện biển số)** |
| **15** | **LPR** | **License Plate Recognition**  **(Hệ thống nhận diện biển số xe)** |
| **16** | **FE** | **Free Slot**  **(Ô trống)** |
| **17** | **PS** | **Parking Slot**  **(Ô đỗ xe)** |
| **18** | **SPM** | **Smart Parking Management**  **(Hệ thống quản lý bãi đỗ xe thông minh)** |

# CHƯƠNG 1. TỔNG QUAN VỀ HỆ THỐNG:

## Giới thiệu đề tài và bài toán thực tế:

Sự phát triển nhanh chóng của các đô thị hiện đại đi kèm với sự gia tăng đáng kể về số lượng phương tiện giao thông cá nhân. Theo thống kê, tại các thành phố lớn của Việt Nam, số lượng ô tô đăng ký mới tăng trung bình 15-20% mỗi năm, dẫn đến tình trạng quá tải nghiêm trọng tại các bãi đỗ xe công cộng, trung tâm thương mại, bệnh viện và khu dân cư. Người dân thường xuyên phải đối mặt với tình trạng không tìm được chỗ đỗ xe hoặc mất quá nhiều thời gian để tìm kiếm vị trí phù hợp.

Hệ thống quản lý bãi đỗ xe truyền thống hiện nay chủ yếu dựa vào nhân công với các nhân viên thu phí và bảo vệ làm việc thủ công. Phương thức này không chỉ tốn kém nguồn nhân lực mà còn dễ gây ra các sai sót như tính phí không chính xác, mất vé xe, tranh chấp giữa chủ xe và nhân viên quản lý. Đặc biệt, trong bối cảnh dịch bệnh COVID-19, việc tiếp xúc trực tiếp giữa nhân viên và khách hàng cũng tiềm ẩn nguy cơ lây nhiễm cao.

Hơn nữa, vấn đề an ninh trong các bãi đỗ xe cũng đang là mối quan ngại lớn. Hàng năm có hàng trăm vụ việc mất cắp tài sản trong xe, trộm xe máy, ô tô hoặc va chạm gây hư hỏng mà không xác định được thủ phạm do thiếu hệ thống giám sát. Ngoài ra, tại các bãi đỗ ngầm hoặc khu vực kín, chất lượng không khí thường kém do khí thải từ động cơ xe tích tụ, gây ảnh hưởng đến sức khỏe nhưng lại không được theo dõi và cảnh báo.

Để giải quyết các vấn đề trên, đề tài "Bãi đỗ xe thông minh" được nghiên cứu và phát triển dựa trên nền tảng công nghệ IoT (Internet of Things) và hệ thống nhúng. Hệ thống kết hợp nhiều công nghệ tiên tiến bao gồm: cảm biến hồng ngoại để phát hiện xe ra vào tự động, thẻ từ RFID để xác thực và quản lý người dùng, camera giám sát để đảm bảo an ninh, và cảm biến chất lượng không khí để bảo vệ sức khỏe. Tất cả được điều khiển bởi vi điều khiển Arduino Mega với khả năng xử lý đa nhiệm và kết nối đa thiết bị. Qua đó, dự án không chỉ mang lại giải pháp thực tiễn cho bài toán quản lý bãi đỗ xe mà còn là cơ hội để sinh viên tiếp cận và làm chủ các công nghệ hiện đại trong lĩnh vực tự động hóa và IoT.

## Mục tiêu và yêu cầu của hệ thống:

Mục tiêu của đề tài là xây dựng hệ thống bãi đỗ xe thông minh quản lý 4 vị trí, tích hợp xác thực người dùng, giám sát an ninh, theo dõi môi trường và điều khiển tự động.

**Về chức năng xác thực người dùng:**

Hệ thống sử dụng module RFID RC522 (tần số 13.56MHz) để xác thực quyền truy cập. Mỗi thẻ có mã UID duy nhất được đăng ký trước. Khi xe đến cổng vào, người lái quét thẻ tại đầu đọc. Nếu thẻ hợp lệ và còn chỗ trống, hệ thống mở cổng và ghi log thông tin (UID, thời gian). Nếu thẻ không hợp lệ, màn hình hiển thị "Access Denied" và từ chối truy cập. Chức năng này nâng cao bảo mật, chỉ cho phép người được phép sử dụng bãi đỗ và tạo dữ liệu lịch sử để quản lý.

**Về chức năng điều khiển xe ra vào:**

Sau khi xác thực RFID thành công, cảm biến hồng ngoại tại cổng vào (chân 2) phát hiện xe tiếp cận. Hệ thống kiểm tra số chỗ trống qua 4 cảm biến slot: nếu còn chỗ, servo quay về 0° mở cổng và hiển thị thông báo trên LCD; nếu đầy, hiển thị "Parking FULL" và từ chối. Sau khi xe vào hoàn toàn, cổng tự động đóng (90°). Cổng ra hoạt động tương tự nhưng yêu cầu quét thẻ để xác nhận xe rời đi, ghi log thời gian ra và cập nhật trạng thái. Hệ thống dùng kỹ thuật edge detection (HIGH→LOW) để tránh xử lý lặp.

**Về chức năng giám sát hình ảnh:**

Hai module camera ESP32-CAM được triển khai để bao quát toàn bộ khu vực. Camera 1 đặt tại cổng vào, tự động chụp ảnh độ phân giải 1280x720 mỗi khi xe được phép vào sau xác thực, lưu trên thẻ SD kèm timestamp và UID. Camera 2 đặt ở vị trí cao bao quát 4 chỗ đỗ, hoạt động streaming hoặc ghi hình theo chu kỳ. Khi phát hiện chuyển động bất thường, hệ thống lưu video 10 giây trước/sau sự kiện. Dữ liệu có thể truy cập từ xa qua web hoặc app mobile, hỗ trợ night mode với LED hồng ngoại để giám sát 24/7.

**Về chức năng giám sát chất lượng không khí:**

Cảm biến MQ135 đặt ở vị trí trung tâm, đo liên tục nồng độ CO, CO2, NH3, NOx và VOC. Hệ thống đọc giá trị analog mỗi 5 giây, chuyển đổi sang ppm và so sánh với ngưỡng an toàn. Khi vượt ngưỡng (CO > 50ppm, CO2 > 1000ppm), hệ thống cảnh báo đa cấp: Level 1 hiển thị màu vàng trên LCD, Level 2 kích hoạt buzzer, Level 3 gửi thông báo khẩn qua SMS/Email. Dữ liệu được ghi log vào file CSV trên thẻ SD. Chức năng này quan trọng với bãi đỗ ngầm, bảo vệ sức khỏe người dùng.

**Về chức năng giám sát và hiển thị trạng thái:**

Bốn cảm biến IR (chân 4-7) phát hiện liên tục trạng thái chỗ đỗ (HIGH = trống, LOW = có xe). Dữ liệu lưu vào mảng, hệ thống tự động đếm số chỗ trống để kiểm tra trước khi cho xe vào. Màn hình LCD 16x2 I2C hiển thị trạng thái theo format "S1:X S2:O S3:X S4:O" với X=có xe, O=trống. Màn hình cập nhật sau mỗi sự kiện. Hệ thống cũng in thông tin chi tiết ra Serial Monitor để giám sát và debug.

**Về yêu cầu phần cứng:**

Arduino Mega 2560 làm bộ điều khiển với 54 chân I/O, bộ nhớ 256KB Flash và 8KB SRAM. Module RFID RC522 kết nối SPI (chân 50-53), cảm biến MQ135 kết nối A0, 6 cảm biến IR ở chân 2-7 với INPUT\_PULLUP, 2 servo SG90 ở chân PWM 9-10, LCD I2C địa chỉ 0x27 ở chân SDA/SCL (20-21), 2 camera ESP32-CAM hoạt động độc lập qua UART hoặc WiFi. Nguồn cấp 12V/3A với bộ chuyển đổi DC-DC xuống 5V và 3.3V.

**Về yêu cầu phần mềm:**

Chương trình viết bằng C/C++ trên Arduino IDE, cấu trúc modular với các file riêng. Sử dụng thư viện: MFRC522.h (RFID), Servo.h, Wire.h, LiquidCrystal\_I2C.h, SD.h, WiFi.h. Xử lý đa nhiệm bằng millis() thay vì delay(), có cơ chế xử lý lỗi cho mất kết nối SD, camera không phản hồi, cảm biến lỗi. Thời gian phản hồi từ quét thẻ đến mở cổng <2 giây, hoạt động liên tục tối thiểu 72 giờ không treo.

## Phạm vi thực hiện dự án:

Để đảm bảo dự án được hoàn thành đúng tiến độ và đạt chất lượng cao, phạm vi thực hiện được xác định rõ ràng về quy mô, chức năng và thời gian triển khai.

**Về quy mô hệ thống:**

Dự án quản lý 4 vị trí đỗ xe, quy mô phù hợp mục đích nghiên cứu và demo. Mô hình vật lý kích thước 70x50x30cm (dài x rộng x cao), khung nhôm và mica trong. Mỗi vị trí đỗ 15x10cm, đủ cho mô hình xe tỷ lệ 1:24. Cổng vào/ra ở hai đầu đối diện, mỗi cổng có thanh chắn servo và đầu đọc RFID. Thiết kế module hóa, linh kiện gắn trên breadboard/PCB, dễ tháo lắp.

**Về chức năng cốt lõi:**

Năm chức năng chính: (1) Xác thực RFID với 10 thẻ mẫu, (2) Điều khiển cổng vào/ra tự động, (3) Giám sát chỗ đỗ realtime, (4) Giám sát an ninh với 2 camera (lưu 100 ảnh trên SD 16GB), (5) Theo dõi chất lượng không khí (log mỗi 5 giây). Các tính năng mở rộng như OCR biển số, thanh toán tự động không nằm trong phạm vi nhưng được đề xuất trong phần phát triển.

**Về công nghệ áp dụng:**

Phần cứng: Arduino Mega 2560, RFID RC522, 6 cảm biến IR FC-51, 2 servo SG90, cảm biến MQ135, 2 camera ESP32-CAM, LCD 16x2 I2C, thẻ SD, module RTC DS3231, breadboard và nguồn 5V/3A. Phần mềm: Arduino IDE 1.8+, thư viện chuẩn từ Library Manager, Fritzing vẽ sơ đồ, Proteus mô phỏng. Tổng kinh phí 2-3 triệu VNĐ.

**Về kế hoạch thực hiện:**

Dự án 10 tuần. Tuần 1-2: Nghiên cứu lý thuyết Arduino, RFID, camera, MQ135 và giao thức SPI/I2C/UART, đọc datasheet, khảo sát dự án mẫu. Tuần 3: Thiết kế sơ đồ khối, nguyên lý, mô hình 3D (SketchUp), lập danh sách linh kiện và đặt hàng. Tuần 4-5: Lập trình module riêng (RFID, servo, LCD, MQ135, camera), test độc lập. Tuần 6: Tích hợp code, xử lý xung đột, tối ưu bộ nhớ, viết error handling. Tuần 7: Lắp ráp phần cứng, xây mô hình, kết nối theo sơ đồ, điều chỉnh vị trí cảm biến/camera. Tuần 8: Kiểm thử từng tính năng (test 100 lần quét thẻ, độ chính xác phát hiện xe, test cảnh báo khí), xử lý bug. Tuần 9: Stress test - vận hành 24h liên tục, mô phỏng 50 lượt xe ra vào, thu thập dữ liệu đánh giá. Tuần 10: Hoàn thiện báo cáo, chuẩn bị slide, quay video demo, chuẩn bị trả lời câu hỏi.

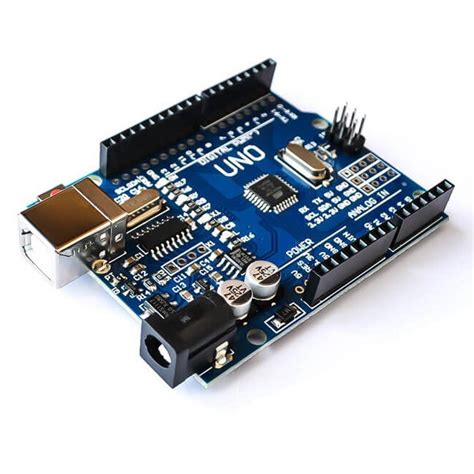
**Về tiêu chí đánh giá:**

Độ chính xác: RFID ≥99%, phát hiện xe ≥95%, sai số MQ135 ≤10%. Hiệu năng: phản hồi quét thẻ→mở cổng ≤2s, chụp/lưu ảnh ≤3s, cập nhật LCD ≤0.5s. Độ tin cậy: hoạt động ≥48h không treo, mất dữ liệu <1%. Tính năng: đầy đủ 5 chức năng ổn định, giao diện thân thiện. Tài liệu: báo cáo với sơ đồ đầy đủ, code có chú thích, video demo. Kết quả so sánh với mục tiêu để đánh giá hoàn thành.

# CHƯƠNG 2. THIẾT KẾ HỆ THỐNG:

## 2.1. Phân tích và lựa chọn phần cứng:

### 2.1.1. Vi điều khiển Arduino:



Hình 1. Vi điều khiển Arduino

Trong dự án bãi đỗ xe thông minh này, vi điều khiển Arduino được lựa chọn làm bộ xử lý trung tâm của hệ thống. Arduino là một nền tảng điện tử mã nguồn mở được thiết kế dựa trên phần cứng và phần mềm dễ sử dụng, đặc biệt phù hợp cho các dự án nhúng và IoT.

**Thông số kỹ thuật Arduino Uno:**

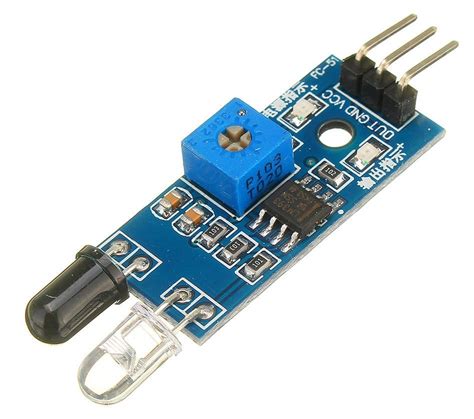
* Vi xử lý: ATmega328P, tần số 16MHz
* Bộ nhớ: Flash 32KB, SRAM 2KB, EEPROM 1KB
* Chân I/O: 14 chân digital (6 chân PWM), 6 chân analog
* Điện áp hoạt động: 5V, điện áp đầu vào 7-12V

Dựa vào code, hệ thống sử dụng tổng cộng 8 chân digital (chân 2-10) để kết nối 6 cảm biến và 2 servo, nằm hoàn toàn trong khả năng cung cấp của Arduino Uno. Arduino Mega có thể được sử dụng nếu cần mở rộng thêm nhiều chỗ đỗ hoặc tính năng phức tạp hơn với 54 chân I/O và bộ nhớ lớn hơn.

**Lý do lựa chọn Arduino:**

* Cộng đồng hỗ trợ lớn với nhiều thư viện sẵn có (Servo.h, Wire.h, LiquidCrystal\_I2C.h)
* Arduino IDE trực quan, lập trình C/C++ đơn giản
* Giá thành phải chăng, dễ mua và thay thế
* Tương thích cao với hầu hết cảm biến và module ngoại vi
* Hỗ trợ các hàm cơ bản như pinMode(), digitalRead(), digitalWrite()

### 2.1.2. Cảm biến hồng ngoại IR:



Hình 2. Cảm biến hồng ngoại IR

Cảm biến hồng ngoại là thiết bị quan trọng nhất, đóng vai trò "mắt" phát hiện xe. Dựa vào code, hệ thống sử dụng 6 cảm biến IR.

**Phân loại và vị trí:**

* 2 cảm biến cổng: SENSOR\_ENTRY (chân 2), SENSOR\_EXIT (chân 3)
* 4 cảm biến slot: SLOT1-4 (chân 4-7)

**Nguyên lý hoạt động:**

* LED phát hồng ngoại (IR Transmitter) phát tia hồng ngoại
* Photodiode thu hồng ngoại (IR Receiver) nhận tín hiệu phản xạ
* Không có vật cản: đầu ra HIGH
* Có vật cản (xe): đầu ra LOW

**Thông số kỹ thuật:**

* Điện áp hoạt động: 5V DC
* Khoảng cách phát hiện: 2-30cm (có thể điều chỉnh)
* Đầu ra: Digital (HIGH/LOW)
* Dòng tiêu thụ: 20mA

**Lý do lựa chọn:**

* Đầu ra digital dễ đọc bằng digitalRead()
* Tốc độ phản hồi nhanh, phù hợp ứng dụng thời gian thực
* Giá thành rẻ (10,000-20,000 VNĐ/module)
* Kích thước nhỏ gọn, dễ lắp đặt
* Tiêu thụ điện năng thấp

Trong code, cảm biến được cấu hình INPUT\_PULLUP: pinMode(SENSOR\_ENTRY, INPUT\_PULLUP). Chế độ này kích hoạt điện trở kéo lên nội 20-50kΩ, đảm bảo chân luôn ở mức HIGH khi không có tín hiệu, tránh nhiễu và đọc sai.

### 2.1.3. Động cơ servo SG90:



Hình 3. Động cơ servo SG90

Động cơ servo điều khiển cổng vào/ra của bãi đỗ. Code sử dụng 2 servo kết nối chân 9 và 10.

**Thông số kỹ thuật SG90:**

* Điện áp: 4.8-6V (tương thích 5V Arduino)
* Dòng tiêu thụ: 100-250mA tùy tải
* Moment xoắn: 1.8kg.cm ở 5V
* Tốc độ: 0.1s/60° ở 4.8V
* Góc quay: 0-180°
* Trọng lượng: 9g
* Kích thước: 23x12.2x29mm

**Điều khiển trong code:**

* Khai báo: Servo servoEntry; Servo servoExit;
* Gắn chân: servoEntry.attach(SERVO\_ENTRY\_PIN);
* Logic đảo ngược: 0° = mở cổng, 90° = đóng cổng
* Hàm openGate(): gate.write(0); delay(1500);
* Hàm closeGate(): gate.write(90); delay(1500);

**Lý do lựa chọn:**

* Điều khiển góc quay chính xác
* Thư viện Servo.h hỗ trợ sẵn, lập trình đơn giản
* Giá thành rẻ (30,000-50,000 VNĐ/con)
* Kích thước nhỏ gọn
* Không cần driver bổ sung

**Lưu ý nguồn**: 2 servo có thể tiêu thụ 500mA, vượt khả năng chân 5V Arduino. Nên dùng nguồn ngoài 5V/2A riêng cho servo, đấu chung GND với Arduino.

### 2.1.4. Màn hình LCD 16x2 I2C:



Hình 4. Màn hình LCD 16x2 I2C

Màn hình LCD là thiết bị hiển thị thông tin đầu ra cho người dùng trong hệ thống bãi đỗ xe thông minh. LCD 16x2 với module I2C được sử dụng để hiển thị trạng thái các chỗ đỗ xe một cách trực quan và dễ theo dõi. LCD 16x2 có nghĩa là màn hình có khả năng hiển thị 16 ký tự trên mỗi dòng với tổng cộng 2 dòng.

**Thông số kỹ thuật LCD 16x2 I2C:**

* Kích thước hiển thị: 16 ký tự x 2 dòng (tổng 32 ký tự)
* Giao tiếp: I2C (Inter-Integrated Circuit)
* Chip chuyển đổi: PCF8574 hoặc PCF8574A
* Địa chỉ I2C: 0x27 (hoặc 0x3F tùy module)
* Điện áp hoạt động: 5V DC
* Đèn nền: LED xanh lá/xanh dương
* Số chân kết nối: 4 chân (VCC, GND, SDA, SCL)

**Các hàm điều khiển LCD được sử dụng trong code:**

* lcd.init(); - Khởi tạo màn hình LCD trong hàm setup()
* lcd.backlight(); - Bật đèn nền để màn hình sáng và dễ đọc
* lcd.clear(); - Xóa toàn bộ nội dung trên màn hình
* lcd.setCursor(col, row); - Đặt vị trí con trỏ tại cột col, dòng row (bắt đầu từ 0)
* lcd.print("text"); - In chuỗi ký tự hoặc biến lên màn hình

**Lý do lựa chọn LCD 16x2 I2C:**

* Giao tiếp I2C tiết kiệm chân I/O của Arduino, chỉ cần 2 chân (SDA, SCL) thay vì 6-8 chân
* Thư viện LiquidCrystal\_I2C.h dễ sử dụng, có nhiều tài liệu hướng dẫn chi tiết
* Hiển thị rõ ràng, dễ đọc ngay cả trong điều kiện ánh sáng yếu nhờ có đèn nền LED
* Giá thành hợp lý (khoảng 70,000 - 100,000 VNĐ), phù hợp với ngân sách dự án
* Tiêu thụ điện năng thấp (~50mA), hoạt động ổn định với nguồn 5V từ Arduino
* Kích thước nhỏ gọn, dễ lắp đặt vào mô hình

**Lưu ý:** nếu LCD không hoạt động sau khi kết nối, cần sử dụng chương trình I2C Scanner để quét và xác định đúng địa chỉ I2C của module, vì một số module sử dụng địa chỉ 0x3F thay vì 0x27. Biến trở trên mặt sau module cho phép điều chỉnh độ tương phản (contrast) của màn hình để hiển thị rõ ràng nhất.RetryClaude can make mistakes. Please double-check responses.

### 2.1.5. ESP32-Cam:



Hình 5. ESP32-Cam

Module ESP32-CAM là một sự nâng cấp đáng kể trong hệ thống bãi đỗ xe thông minh, thay thế chức năng hiển thị ký tự đơn thuần bằng khả năng giám sát bằng hình ảnh và kết nối IoT mạnh mẽ hơn. Nó không chỉ là thiết bị hiển thị thông tin đầu ra mà còn là bộ xử lý thông minh cho hệ thống.

**Thông số Kỹ thuật Chính của ESP32-CAM**

* Chức năng chính: Xử lý, Giám sát Hình ảnh, Kết nối IoT
* Vi xử lý:ESP32 (Dual-Core, 32-bit, lên đến 240 MHz)
* Đầu ra: Giao diện Web Server, Video Stream, Dữ liệu hình ảnh
* Giao tiếp:Wi-Fi 802.11 b/g/n, Bluetooth (Toàn cầu)
* Điện áp hoạt động: 5V DC (Thường được cấp nguồn qua chân 5V)
* Số chân kết nối: Nhiều chân GPIO (Cần cấp nguồn, UART cho lập trình, các chân I/O)

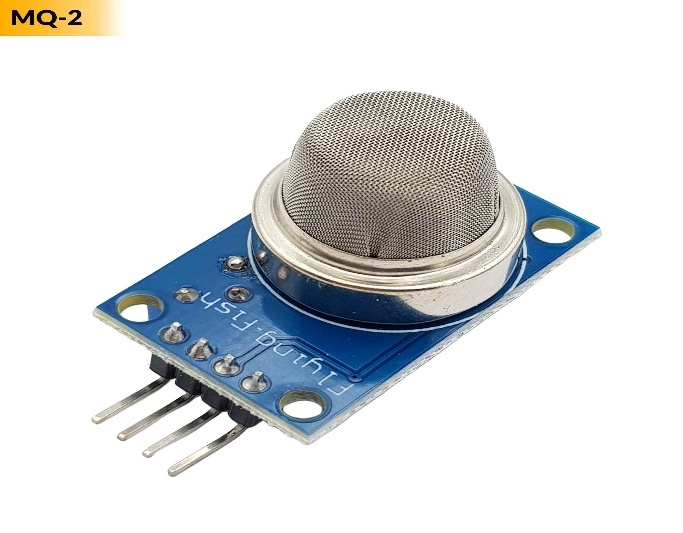
**Lý do Lựa chọn và Lợi ích Vượt trội:**

* Tích hợp IoT: Có sẵn Wi-Fi/Bluetooth nên không cần thêm module mạng như ESP8266; dễ gửi dữ liệu lên Cloud hoặc tạo Web Server giám sát.
* Giám sát bằng hình ảnh: Camera OV2640 giúp quan sát nhiều ô đỗ xe thay cho dùng nhiều cảm biến → tiết kiệm chi phí.
* Hiệu suất mạnh: Vi xử lý Dual-Core 32-bit hỗ trợ xử lý ảnh, nén JPEG và truyền video—những tác vụ Arduino Uno không thể làm.
* Nâng cao an ninh: Có thể live-stream, hỗ trợ nhận diện biển số cơ bản, lưu hình vào thẻ MicroSD khi có sự cố.
* Tích hợp cao: Một module chứa MCU + Wi-Fi + Camera → giảm số lượng linh kiện và dây nối.

**Lưu ý Khi Sử dụng ESP32-CAM:**

* Lập trình phức tạp hơn: Cần cấu hình mạng, Web Server và xử lý ảnh → khó hơn Arduino truyền thống.
* Nguồn cấp: Tiêu thụ dòng lớn (đặc biệt khi bật Wi-Fi và camera). Cần nguồn 5V ổn định.
* Nạp code: Phải dùng USB-to-TTL và đưa chân GPIO0 xuống GND khi upload chương trình.

### 2.1.6. Cảm biến khí ga:



Hình 6. Cảm biến khí ga

Cảm biến khí ga đóng vai trò là *"mũi"* của hệ thống, giúp giám sát chất lượng không khí và đảm bảo an toàn, đặc biệt quan trọng đối với các bãi đỗ xe ngầm hoặc kín.

**Phân loại và Vị trí (Ví dụ Tích hợp)**

* Thay vì đếm xe, cảm biến khí ga được bố trí để giám sát môi trường an toàn.
* Số lượng**:** Tùy theo quy mô bãi đỗ, có thể dùng 1-2 cảm biến.

**Phân loại & Vị trí:**

**+** SENSOR\_CO (MQ-7): Đặt gần lối ra/vào hoặc khu vực dễ tích tụ khí thải để phát hiện Carbon monoxide (CO), sản phẩm độc hại từ động cơ xe. (Kết nối với chân A0và D0).

**+** SENSOR\_LPG (MQ-6): Đặt ở vị trí thấp (khí LPG nặng hơn không khí) để phát hiện rò rỉ khí đốt từ bình nhiên liệu hoặc xe chạy bằng khí. (Kết nối với chân A1 và D1). **Nguyên lý Hoạt động (Bán dẫn Kim loại - MQ-X)**

* Thành phần chính: Lớp vật liệu bán dẫn SnO\_2 được nung nóng bởi một dây nhiệt.
* Trong không khí sạch: Điện trở của cảm biến (R\_2 cao)
* Có Khí mục tiêu: Khí (ví dụ: CO) phản ứng với oxy trên bề mặt SnO\_2 giải phóng electron, làm cho điện trở (R\_s) giảm xuống.
* Đầu ra**:**

**+** Analog (A0): Điện áp đầu ra (V\_out) tăng tỷ lệ thuận với nồng độ khí (khiR\_sgiảm).

**+** Digital (D}0): Đầu ra LOW khi nồng độ khí vượt qua ngưỡng đã cài đặt bằng biến trở.

**Lý do Lựa chọn và Cách Xử lý Tín hiệu**

* Đầu ra Digital/Analog: Có cả hai đầu ra $\text{A}0$ và $\text{D}0$
* Tính thời gian thực: Phản hồi nhanh với sự thay đổi nồng độ khí.
* Giá thành/Kích thước: Giá thành rẻ (25,000-50,000 VNĐ/module), kích thước nhỏ gọn.
* Độ nhạy: Có thể điều chỉnh ngưỡng $\text{D}0$ bằng biến trở.

### 2.1.7. Module RFID RC522:



Hình 7. Module RC522 và bộ thẻ RFID

Module RFID RC522 là thiết bị đọc thẻ từ không tiếp xúc được tích hợp vào hệ thống để quản lý xe ưu tiên như xe nhân viên, xe cư dân, hoặc xe đã đăng ký dài hạn. Đây là giải pháp bổ sung quan trọng cho hệ thống nhận diện biển số bằng camera, đảm bảo xe vẫn có thể ra vào bãi đỗ ngay cả khi camera gặp sự cố, thời tiết xấu (mưa, sương mù), hoặc biển số bị bẩn không đọc được.

**Thông số kỹ thuật RC522:**

* Tần số hoạt động: 13.56MHz (chuẩn ISO/IEC 14443 Type A)
* Khoảng cách đọc: 0-6cm (tùy thuộc kích thước antenna của thẻ)
* Giao tiếp: SPI (Serial Peripheral Interface)
* Điện áp hoạt động: 3.3V DC
* Dòng tiêu thụ: 13-26mA khi đang đọc thẻ, <80µA ở chế độ standby
* Tốc độ truyền: Lên đến 10Mbit/s
* Thẻ tương thích: Mifare Classic 1K, Mifare 4K, Mifare Ultralight, NTAG213/215/216
* Kích thước module: 40mm × 60mm
* Nhiệt độ hoạt động: -20°C đến 80°C

**Nguyên lý hoạt động:**

RC522 hoạt động dựa trên nguyên lý cảm ứng từ. Module tạo ra trường điện từ tần số 13.56MHz qua antenna. Khi thẻ RFID tiếp cận trong phạm vi 0-6cm, trường từ này cảm ứng dòng điện trong cuộn dây của thẻ, cung cấp năng lượng cho chip bên trong thẻ. Chip thẻ sau đó phát lại dữ liệu UID bằng cách thay đổi tải của cuộn dây, tạo ra biến điệu trường từ mà RC522 nhận biết.

**Lý do lựa chọn RC522:**

* Độ tin cậy cao: Không bị ảnh hưởng bởi thời tiết, ánh sáng yếu như camera
* Tốc độ nhanh: Đọc thẻ và xác thực chỉ trong 1-2 giây
* Bảo mật: Mỗi thẻ có UID duy nhất, khó làm giả
* Giá thành rẻ: Module RC522 khoảng 30,000-50,000 VNĐ, thẻ Mifare 1K chỉ 3,000-5,000 VNĐ/thẻ
* Dễ quản lý: Có thể cấp/thu hồi quyền truy cập thẻ qua database
* Tuổi thọ cao: Thẻ RFID có thể sử dụng trên 100,000 lần đọc/ghi

## 2.2. Thiết kế sơ đồ khối hệ thống:

Sơ đồ khối mô tả tổng quan kiến trúc hệ thống bãi đỗ xe thông minh tích hợp IoT với kiến trúc Master-Slave, kết nối Cloud và hệ thống AI nhận diện biển số. Hệ thống được chia thành 3 tầng chính: Tầng Cảm biến (Sensor Layer), Tầng Xử lý (Processing Layer), và Tầng Ứng dụng (Application & Cloud Layer).

**Sơ đồ khối tổng quan:**

A diagram of a computer

AI-generated content may be incorrect.

Hình 8. Sơ đồ khối hệ thống bãi đỗ xe thông minh IoT

**Bảng mô tả chức năng các khối:**

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên khối** | **Chức năng** | **Kết nối** |
| 1 | IR Sensors (×6) | Phát hiện xe tại cổng vào/ra và 4 chỗ đỗ | Arduino Pin D2–D7 |
| 2 | MQ135 Sensor | Giám sát chất lượng không khí (CO2, NH3, khói) | Arduino Pin A0 |
| 3 | RFID RC522 | Đọc thẻ từ cho xe ưu tiên (nhân viên, cư dân) | ESP32 SPI (GPIO 5, 4) |
| 4 | ESP32-CAM Entry | Chụp ảnh biển số xe tại cổng vào, gửi Server AI | WiFi HTTP (/capture) |
| 5 | ESP32-CAM Exit | Chụp ảnh biển số xe tại cổng ra, gửi Server AI | WiFi HTTP (/capture) |
| 6 | Arduino Uno R3 | Slave I2C: Đọc IR + MQ135, điều khiển servo, gửi 3 bytes data | I2C Slave (0x08) |
| 7 | ESP32 DevKit V1 | Master I2C: Nhận data, xử lý RFID, gọi API, giao tiếp Server | I2C Master, WiFi, SPI |
| 8 | Servo SG90 (×2) | Mở/đóng cổng vào (Pin 9) và cổng ra (Pin 10) | Arduino PWM D9, D10 |
| 9 | LCD 16x2 I2C | Hiển thị trạng thái 4 slot, thông báo hệ thống | ESP32 I2C (0x27) |
| 10 | Flask Server AI | YOLOv5 detect + EasyOCR đọc biển số, quản lý database | HTTP Port 5000 |
| 11 | Web Dashboard | Giám sát realtime, điều khiển manual từ xa | HTTP từ Server |

Bảng 1. Chức năng và kết nối các khối trong hệ thống

**Nguyên lý hoạt động:**

**Bước 1:** Các cảm biến IR và MQ135 liên tục đọc dữ liệu, Arduino Uno xử lý và đóng gói thành 3 bytes gửi qua I2C cho ESP32.

**Bước 2:** ESP32 nhận data từ Arduino, đồng thời quét thẻ RFID nếu có người quẹt thẻ.

**Bước 3:** Khi cảm biến IR Entry/Exit phát hiện xe, ESP32 gọi HTTP đến Camera tương ứng để chụp ảnh biển số.

**Bước 4:** ESP32 forward ảnh lên Flask Server, Server dùng YOLOv5 detect biển số và EasyOCR đọc ký tự, trả về kết quả (allow/deny).

**Bước 5:** ESP32 xử lý kết quả từ Server hoặc RFID, gửi lệnh cho Arduino mở/đóng servo, đồng thời cập nhật LCD hiển thị trạng thái.

**Bước 6:** Tất cả dữ liệu được gửi lên Server Flask để lưu log và hiển thị trên Web Dashboard cho quản lý giám sát từ xa.

## 2.3. Thiết kế sơ đồ mạch điện:

Sơ đồ mạch điện trình bày chi tiết cách kết nối các linh kiện với Arduino Uno và ESP32, bao gồm phân bổ chân I/O, cấu hình tín hiệu và nguồn cấp điện để đảm bảo hệ thống hoạt động ổn định.

### 2.3.1. Bảng phân bổ chân kết nối chi tiết:

**Bảng kết nối Arduino Uno R3:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Linh kiện** | **Chân Arduino** | **Kết nối** | **Loại tín hiệu** |
| IR Sensor Entry | Digital Pin 2 | OUT → D2, VCC → 5V, GND → GND | Digital Input (Pull-up) |
| IR Sensor Exit | Digital Pin 3 | OUT → D3, VCC → 5V, GND → GND | Digital Input (Pull-up) |
| IR Sensor Slot 1 | Digital Pin 4 | OUT → D4, VCC → 5V, GND → GND | Digital Input (Pull-up) |
| IR Sensor Slot 2 | Digital Pin 5 | OUT → D5, VCC → 5V, GND → GND | Digital Input (Pull-up) |
| IR Sensor Slot 3 | Digital Pin 6 | OUT → D6, VCC → 5V, GND → GND | Digital Input (Pull-up) |
| IR Sensor Slot 4 | Digital Pin 7 | OUT → D7, VCC → 5V, GND → GND | Digital Input (Pull-up) |
| MQ135 Sensor | Analog Pin A0 | AOUT → A0, VCC → 5V, GND → GND | Analog Input (0–1023) |
| Servo Entry | Digital Pin 9 | Signal → D9, VCC → 5V ext, GND → GND | PWM Output (50 Hz) |
| Servo Exit | Digital Pin 10 | Signal → D10, VCC → 5V ext, GND → GND | PWM Output (50 Hz) |
| ESP32 (I2C) | A4 (SDA), A5 (SCL) | SDA ↔ GPIO21, SCL ↔ GPIO22 | I2C Slave (0x08) |

Bảng 2. Kết nối Arduino Uno R3

**Bảng kết nối ESP32 DevKit V1:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Linh kiện** | **Chân ESP32** | **Kết nối** | **Loại tín hiệu** |
| RFID RC522 SS | GPIO 5 | SS → GPIO5 | SPI Slave Select |
| RFID RC522 RST | GPIO 4 | RST → GPIO4 | Digital Output |
| RFID RC522 SCK | GPIO 18 | SCK → GPIO18 | SPI Clock |
| RFID RC522 MOSI | GPIO 23 | MOSI → GPIO23 | SPI Master Out |
| RFID RC522 MISO | GPIO 19 | MISO → GPIO19 | SPI Master In |
| RFID RC522 VCC | 3.3V | VCC → 3.3V (**KHÔNG dùng 5V**) | Power 3.3V |
| LCD 16x2 I2C | GPIO 21 (SDA), 22 (SCL) | SDA → GPIO21, SCL → GPIO22 | I2C Bus (0x27) |
| Arduino Uno (I2C) | GPIO 21 (SDA), 22 (SCL) | SDA ↔ A4, SCL ↔ A5 | I2C Master |
| ESP32-CAM Entry | WiFi | HTTP: http://192.168.2.X/capture | WiFi Client |
| ESP32-CAM Exit | WiFi | HTTP: http://192.168.2.Y/capture | WiFi Client |
| Flask Server | WiFi | HTTP: http://192.168.2.173:5000 | WiFi Client |

Bảng 3. Kết nối ESP32 DevKit V1

### 2.3.2. Sơ đồ kết nối nguồn điện:

**Bảng phân bổ nguồn điện:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Thiết bị** | **Điện áp** | **Dòng tiêu thụ** | **Nguồn cấp** |
| Arduino Uno R3 | 5V | ~50 mA | USB 5V hoặc DC Adapter 7–12V |
| ESP32 DevKit V1 | 5V (VIN) | ~250 mA (WiFi bật) | USB 5V hoặc Adapter 5V/2A |
| IR Sensor (×6) | 5V | ~120 mA (tổng) | Arduino 5V Rail |
| MQ135 Sensor | 5V | ~170 mA | Nguồn ngoài 5V/1A RIÊNG |
| Servo SG90 (×2) | 5V | ~1000 mA peak (2 servo) | Nguồn ngoài 5V/2A RIÊNG |
| RFID RC522 | 3.3V | ~26 mA | ESP32 3.3V Rail |
| LCD 16x2 I2C | 5V | ~50 mA | ESP32 VIN (5V) |
| ESP32-CAM (×2) | 5V | ~600 mA (2 camera) | Adapter 5V/1A riêng |

Bảng 4. Phân bổ nguồn điện hệ thống

**Lưu ý quan trọng về nguồn:**

1. Servo phải dùng nguồn ngoài riêng: Kết nối VCC Servo → Nguồn 5V/2A ngoài, GND phải nối chung với Arduino
2. RFID dùng 3.3V: TUYỆT ĐỐI không cấp 5V sẽ hỏng chip RC522 ngay lập tức
3. GND chung bắt buộc: Tất cả GND (Arduino, ESP32, Nguồn ngoài, Servo) phải nối chung với nhau
4. MQ135 nên tách nguồn: Dòng heater lớn (~150mA), nên dùng nguồn riêng để tránh nhiễu ADC của Arduino
5. ESP32-CAM dùng nguồn riêng: Mỗi camera tiêu thụ ~300mA khi streaming, không nên dùng chung với ESP32 chính

### ****2.3.3. Sơ đồ nguyên lý kết nối:****

**A. Khối đầu vào Arduino Uno (Input):**

* 6 cảm biến IR được cấu hình INPUT\_PULLUP trên chân D2-D7
* Mỗi cảm biến có 3 chân: VCC (5V), GND, OUT (tín hiệu)
* Trạng thái: LOW = có vật cản (xe), HIGH = không có xe
* Điện trở pull-up nội bộ 20-50kΩ của Arduino được sử dụng
* MQ135 kết nối chân AOUT → A0, sử dụng ADC 10-bit (0-1023)

**B. Khối điều khiển servo (Output):**

* 2 động cơ servo SG90 kết nối chân tín hiệu (dây cam) → D9, D10
* Góc quay: 0° = mở cổng, 90° = đóng cổng
* Tín hiệu PWM tần số 50Hz (chu kỳ 20ms)
* Quan trọng: VCC servo (dây đỏ) → Nguồn ngoài 5V/2A, GND (dây nâu) → GND chung

**C. Khối giao tiếp I2C Arduino ↔ ESP32:**

* Arduino Uno: SDA (A4) ↔ ESP32 GPIO 21
* Arduino Uno: SCL (A5) ↔ ESP32 GPIO 22
* Arduino hoạt động ở chế độ I2C Slave với địa chỉ 0x08
* ESP32 hoạt động ở chế độ I2C Master
* Tốc độ truyền: 100 kHz (chuẩn I2C)
* Không cần điện trở pull-up ngoài (đã có trong module LCD I2C)

**D. Khối RFID trên ESP32:**

* Giao tiếp SPI với ESP32:
  + SS (Slave Select) → GPIO 5
  + RST (Reset) → GPIO 4
  + SCK (Serial Clock) → GPIO 18
  + MOSI (Master Out) → GPIO 23
  + MISO (Master In) → GPIO 19
* **VCC → 3.3V** (không dùng 5V!)
* GND → GND chung

**E. Khối LCD I2C:**

* Địa chỉ I2C: 0x27 (hoặc 0x3F tùy module)
* SDA → ESP32 GPIO 21 (chia sẻ với I2C Arduino)
* SCL → ESP32 GPIO 22 (chia sẻ với I2C Arduino)
* VCC → ESP32 VIN (5V), GND → GND
* Module I2C đã tích hợp điện trở pull-up 4.7kΩ

## 2.4. Thiết kế thuật toán và lưu đồ chương trình:

Thuật toán mô tả logic xử lý của chương trình từ khâu khởi tạo, đọc cảm biến, giao tiếp I2C/WiFi/SPI, xử lý AI đến điều khiển servo và cập nhật LCD. Hệ thống sử dụng kiến trúc Master-Slave với Arduino Uno xử lý tín hiệu thời gian thực và ESP32 xử lý logic phức tạp thông qua FreeRTOS.

**Lưu đồ thuật toán tổng quát:**

A diagram of a computer program

AI-generated content may be incorrect.

Hình 9. Lưu đồ hoạt động Arduino Uno (I2C Slave)

A diagram of a computer

AI-generated content may be incorrect.

Hình 10. Lưu đồ thuật toán điều khiển hệ thống bãi đỗ xe thông minh IoT

**Bảng phân tích các hàm chính:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Module** | **Tên hàm / API** | **Tham số** | **Chức năng** | **Giá trị trả về** |
| **Arduino** | readSensors() | void | Đọc 6 IR + MQ135, lưu vào biến toàn cục | void |
| **Arduino** | requestEvent() | void | Gửi 3 bytes (IR states + MQ135) qua I2C khi ESP32 yêu cầu | void |
| **Arduino** | receiveEvent(int numBytes) | numBytes | Nhận lệnh từ ESP32 qua I2C → điều khiển servo | void |
| **ESP32** | TaskSystem() | void | Task FreeRTOS chính: đọc I2C, xử lý RFID, gọi API, cập nhật LCD | void |
| **ESP32** | callServer(type, action, uid) | type, action, uid | Gửi HTTP API đến Flask Server hoặc ESP32-CAM | String (JSON) |
| **ESP32** | sendToUno(String cmd) | cmd | Gửi lệnh điều khiển servo cho Arduino (OPEN/CLOSE ENTRY/EXIT) | void |
| **ESP32** | updateLCDGrid() | void | Cập nhật trạng thái 4 slot lên LCD | void |
| **Flask** | /api/parking/entry | – | Nhận request, gọi camera, chạy AI YOLO + OCR, trả kết quả | JSON |
| **Flask** | /api/rfid/entry | uid | Kiểm tra UID trong DB, quyết định cho phép hay từ chối | JSON |

Bảng 5. Chức năng các hàm trong hệ thống

**Bảng mã trạng thái hệ thống:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Trạng thái** | **Giá trị** | **Ý nghĩa** | **Hiển thị LCD** |
| **Slot trống** | IR sensor = HIGH | Không có xe | S1:O |
| **Slot có xe** | IR sensor = LOW | Có xe đỗ | S1:X |
| **Cổng mở** | Servo = 0° | Cho phép xe vào/ra | Gate Opening... |
| **Cổng đóng** | Servo = 90° | Chặn xe | Gate Closing... |
| **Bãi đầy** | freeSlots = 0 | Không còn chỗ trống | !! FULL !! / Entry Denied |
| **Chờ Camera** | Gọi HTTP /capture | Đang chụp ảnh biển số | CHECKING CAM... / WAIT |
| **AI Processing** | Server đang xử lý | YOLOv5 + OCR nhận diện biển số | Processing... |
| **Cho phép vào** | allow\_entry | Xe hợp lệ | WELCOME / OPEN GATE |
| **Từ chối vào** | deny\_entry | Xe không hợp lệ hoặc chưa đăng ký | CAM DENIED / USE CARD |
| **RFID hợp lệ** | UID có trong DB | Thẻ hợp lệ | RFID OK / OPEN GATE |
| **RFID không hợp lệ** | UID không có trong DB | Thẻ lạ | RFID DENIED / INVALID |
| **Cần thanh toán** | payment\_due | Xe ra phải trả phí | PAY FEE / OPEN GATE |
| **Không khí tốt** | MQ135 < 600 | An toàn (CO2/NH3 thấp) | (Không hiển thị) |
| **Cảnh báo khí** | MQ135 > 600 | Không khí kém, khí độc cao | AIR WARNING! |

Bảng 6. Trạng thái hệ thống bãi đỗ xe IoT

**Ưu điểm của thuật toán:**

* Kiến trúc Master-Slave rõ ràng: Arduino xử lý realtime, ESP32 xử lý logic phức tạp
* Đa nhiệm với FreeRTOS: ESP32 chạy Task độc lập, không bị blocking
* Xử lý AI tích hợp: YOLOv5 + EasyOCR nhận diện biển số tự động
* Chống nhiễu tốt: Smoothing cho MQ135, edge detection cho IR
* Dễ mở rộng: Có thể thêm tính năng mới (payment gateway, mobile app) mà không ảnh hưởng core system
* Phản hồi nhanh: Xử lý song song I2C, WiFi, RFID trong cùng một Task

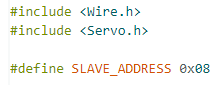
# CHƯƠNG 3. TRIỂN KHAI VÀ KẾT QUẢ:

## 3.1. Lập trình và cài đặt hệ thống:

Sau khi hoàn thành thiết kế phần cứng, bước tiếp theo là lập trình phần mềm để điều khiển toàn bộ hệ thống.

### 3.1.1. Code arduno.

-Khai báo thư viện.

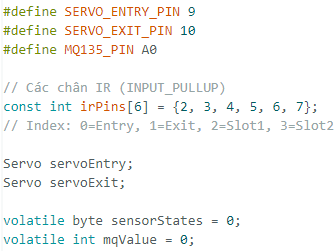


#include <Wire.h>: Cho phép giao tiếp I2C (giao tiếp hai dây), được sử dụng để Arduino Uno (Slave) giao tiếp với ESP32 (Master).

#include <Servo.h>: Cho phép điều khiển các động cơ Servo.

#define SLAVE\_ADDRESS 0x08: Định nghĩa địa chỉ duy nhất của bo mạch Arduino này trên bus I2C là 0x08.

-Định nghĩa các chân và biến toàn cục.



#define ...: Định nghĩa chân cho hai động cơ Servo cửa vào/ra (chân 9, 10) và cảm biến khí MQ-135 (chân Analog A0).

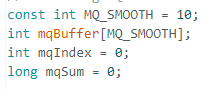
const int irPins[6] = {2, 3, 4, 5, 6, 7};: Mảng chứa các chân Digital cho 6 cảm biến hồng ngoại (IR). Các cảm biến này dùng để phát hiện xe tại cửa vào, cửa ra và 4 vị trí đỗ xe (Slot 1-4).

Servo servoEntry; Servo servoExit;: Khai báo các đối tượng Servo để điều khiển hai động cơ cửa.

volatile byte sensorStates = 0;: Biến kiểu byte lưu trạng thái của 6 cảm biến IR (mỗi bit đại diện cho một cảm biến). Từ khóa volatile quan trọng vì biến này được cập nhật trong loop() và được đọc trong hàm ngắt I2C (requestEvent).

volatile int mqValue = 0;: Biến lưu giá trị đọc được (đã làm mịn) từ cảm biến MQ-135.

- Làm mịn Cảm biến MQ-135 (Moving Average)



const int MQ\_SMOOTH = 10;: Định nghĩa số lượng mẫu (10 mẫu) được sử dụng để tính giá trị trung bình.

int mqBuffer[MQ\_SMOOTH];: Mảng (buffer) để lưu trữ 10 giá trị thô gần nhất.

int mqIndex = 0;: Chỉ mục (index) để theo dõi vị trí tiếp theo trong buffer cần được ghi đè.

long mqSum = 0;: Tổng của 10 giá trị trong buffer, dùng để tính trung bình nhanh hơn.

- Hàm setup()

Hàm này chạy một lần khi Arduino khởi động, dùng để thiết lập tất cả các giao tiếp và trạng thái ban đầu.

Serial.begin(9600);: Khởi tạo giao tiếp Serial để gỡ lỗi/theo dõi qua máy tính.

I2C Setup:

Wire.begin(SLAVE\_ADDRESS);: Khởi động I2C, đặt Arduino làm Slave với địa chỉ 0x08.

Wire.onRequest(requestEvent);: Đăng ký hàm sẽ được gọi khi Master yêu cầu dữ liệu.

Wire.onReceive(receiveEvent);: Đăng ký hàm sẽ được gọi khi Master gửi dữ liệu/lệnh tới Slave.

Cài đặt chân I/O:

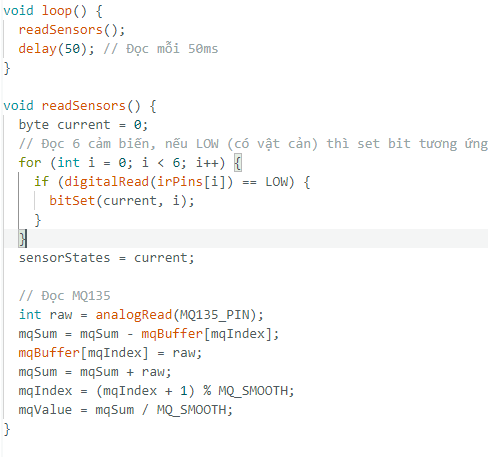
Các chân IR được đặt là INPUT\_PULLUP. Khi cảm biến phát hiện vật (xe), tín hiệu là LOW (0); khi không có vật, tín hiệu là HIGH (1).

Chân MQ-135 được đặt là INPUT (Analog Input).

Khởi tạo Servo: Gắn Servo với các chân điều khiển và quan trọng nhất là đặt chúng ở trạng thái ĐÓNG (servo.write(90); góc $\mathbf{90^\circ}$) khi khởi động.

- Hàm loop() và readSensors()

Hàm loop() là vòng lặp chính của chương trình, và nó liên tục gọi hàm readSensors() để cập nhật dữ liệu.



loop(): Gọi readSensors() và tạm dừng 50ms.

Đọc IR:

Vòng lặp chạy qua 6 cảm biến.

Nếu tín hiệu là LOW (phát hiện xe/vật cản), hàm bitSet(current, i) sẽ đặt bit tương ứng trong biến current thành 1.

Kết quả cuối cùng là một byte sensorStates, nơi mỗi bit là trạng thái của một cảm biến.

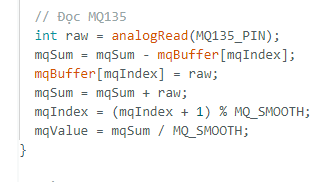
Đọc MQ-135 (Làm mịn):

Đọc giá trị thô (raw).

Thực hiện thuật toán trung bình trượt (Moving Average) 10 mẫu: Trừ mẫu cũ nhất, cộng mẫu mới nhất, cập nhật vị trí (mqIndex), và chia trung bình để lấy mqValue.

- Hàm Ngắt I2C - Gửi Dữ liệu (requestEvent)

Hàm này được gọi tự động khi Master (ESP32) yêu cầu dữ liệu từ Slave.



Tạo một mảng 3 byte để chứa dữ liệu cần gửi.

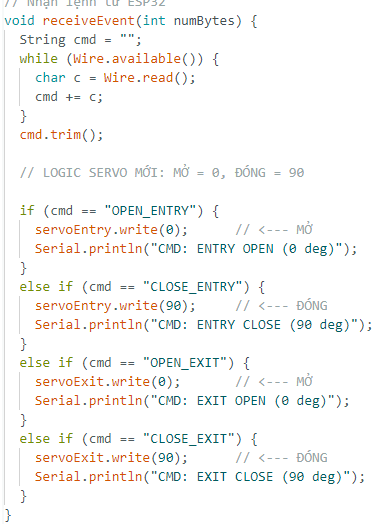
Byte 1 (data[0]): Chứa trạng thái của 6 cảm biến IR (sensorStates).

Byte 2 & 3 (data[1], data[2]): Giá trị MQ-135 (mqValue) là một số nguyên 16-bit. Để gửi qua I2C, nó được chia thành hai byte: Byte cao (highByte()) và Byte thấp (lowByte()).

Wire.write(data, 3);: Gửi 3 byte dữ liệu này qua I2C cho Master.

- Hàm Ngắt I2C - Nhận Lệnh (receiveEvent)

Hàm này được gọi tự động khi Master (ESP32) gửi lệnh đến Slave.



Nhận lệnh: Đọc tất cả các byte có sẵn trong bộ đệm I2C và nối chúng thành một chuỗi (cmd). cmd.trim() loại bỏ khoảng trắng thừa.

Thực thi lệnh:

* Code kiểm tra chuỗi cmd nhận được có khớp với các lệnh định sẵn hay không.
* Nếu là lệnh MỞ (ví dụ: "OPEN\_ENTRY"), Servo sẽ được đặt ở góc 0
* Nếu là lệnh ĐÓNG (ví dụ: "CLOSE\_ENTRY"), Servo sẽ được đặt ở góc 90

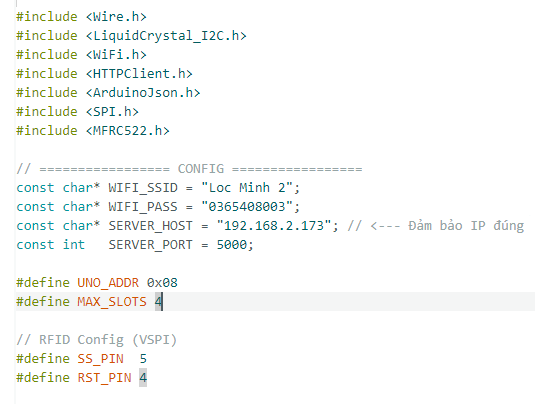
In thông báo xác nhận lệnh ra Serial.

### 3.1.2. Code esp32wroom:

Đây là code cho ESP32 đóng vai trò là Master(Thiết bị Chính) trong hệ thống bãi đỗ xe thông minh. ESP32 chịu trách nhiệm chính về giao tiếp mạng (Wi-Fi, HTTP), quản lý logic bãi đỗ xe, đọc thẻ RFID và giao tiếp với Arduino Uno Slave (đã giải thích ở câu trước) qua I2C để đọc cảm biến và điều khiển Servo.

- Khai báo Thư viện & Cấu hình

Phần này thiết lập các thư viện cần thiết và thông tin mạng/địa chỉ cố định.

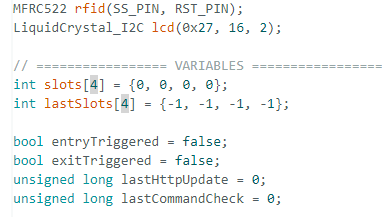


Thư viện Tải các thư viện I2C, LCD, WiFi, HTTP, JSON và RFID.

Cấu hình Định nghĩa SSID/Mật khẩu Wi-Fi địa chỉ IP Server (được dùng để giao tiếp với hệ thống backend), địa chỉ I2C của Arduino Uno (`0x08`), và số lượng chỗ đỗ tối đa (`MAX\_SLOTS = 4`).

* Khởi tạo Đối tượng & Biến

Khởi tạo các đối tượng phần cứng và các biến logic để quản lý trạng thái bãi đỗ xe.



Rfid Đối tượng điều khiển module RFID MFRC522 .

lcd` : Đối tượng điều khiển màn hình LCD 16x2 qua I2C (địa chỉ `0x27`).

slots[4 Mảng lưu trữ trạng thái hiện tại của 4 vị trí đỗ xe (0: Trống, 1: Có xe).

lastSlots[4]` : Dùng để so sánh và chỉ cập nhật màn hình LCD khi trạng thái bãi đỗ thay đổi .

entryTriggered`, `exitTriggered` : Biến cờ chống lặp lại (debounce) khi cảm biến IR cửa vào/ra bị kích hoạt, đảm bảo logic chỉ chạy một lần cho mỗi sự kiện.

`lastHttpUpdate`, `lastCommandCheck` : Các biến thời gian dùng để định kỳ gửi dữ liệu cảm biến và kiểm tra lệnh điều khiển thủ công từ Server.

* Các Hàm Hỗ trợ Hiển thị (LCD)

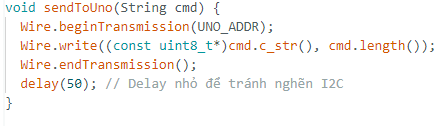
Các hàm này giúp đơn giản hóa việc tương tác và cập nhật thông tin lên màn hình LCD.

`updateLCDGrid()` : Hiển thị trạng thái của 4 vị trí đỗ (S1-S4). Nếu vị trí có xe, hiển thị 'X', nếu trống thì ' '. Hàm chỉ thực hiện cập nhật khi trạng thái thực sự thay đổi.

`lcdMessage()` : Xóa màn hình và hiển thị hai dòng thông báo tùy chỉnh (dùng cho các sự kiện như Quẹt thẻ, Camera Check, v.v.).

* Điều khiển Phần cứng (I2C)

Hàm gửi lệnh đến Arduino Uno Slave qua I2C.

```

`Wire.beginTransmission(UNO\_ADDR)` : Bắt đầu truyền dữ liệu tới địa chỉ Uno Slave (`0x08`).

`Wire.write(...)` : Gửi chuỗi lệnh (ví dụ: `"OPEN\_ENTRY"`, `"CLOSE\_EXIT"`) tới Slave.

`Wire.endTransmission()` : Kết thúc truyền. Arduino Uno sẽ nhận và xử lý lệnh trong hàm `receiveEvent`.

* Giao tiếp Server (HTTP)

Đây là các hàm chính để trao đổi dữ liệu và logic với Server Backend.

`callServer()` (Giao tiếp Logic)



Hàm đa năng dùng để gọi các API Server như `/api/parking/entry` (Camera vào) hoặc `/api/rfid/exit` (Thẻ ra).

Sử dụng `HTTPClient` để tạo yêu cầu POST .

Thiết lập `http.setConnectTimeout(3000)` (3 giây) để tránh treo thiết bị nếu Server không phản hồi.

Sử dụng thư viện `ArduinoJson` để phân tích phản hồi từ Server (thường là quyết định `action` và thông tin `plate` - biển số xe).

Trả về quyết định của Server (`allow\_entry`, `deny\_entry`, `payment\_due`, v.v.).

`checkServerCommand()` (Lệnh thủ công)



Hàm này gọi API Server định kỳ để kiểm tra xem Admin có gửi lệnh Mở/Đóng cửa thủ công từ giao diện Web hay không.

Nếu nhận được lệnh (không phải "none"/"null"), ESP32 sẽ hiển thị thông báo LCD và gửi lệnh điều khiển tới Uno Slave.

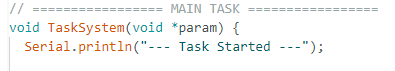
`sendSensorData()` (Cập nhật trạng thái)



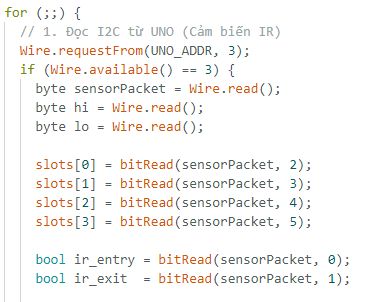
Hàm này gửi trạng thái hiện tại của 4 vị trí đỗ (đọc được từ Uno) lên Server để Web/Admin có thể theo dõi.

* Task Chính: `TaskSystem()` (Sử dụng FreeRTOS)

ESP32 chạy code này trong một Task riêng biệt (`xTaskCreate`), cho phép nó thực hiện các tác vụ song song.



Đọc I2C từ UNO (Cảm biến IR & MQ135)



Đọc 3 byte: sensorPacket, highByte(mqValue), lowByte(mqValue)

Tách trạng thái 4 vị trí đỗ (Slot 1-4) từ Bit 2-5

Tách trạng thái IR Entry (Bit 0) và IR Exit (Bit 1)

-LOGIC CAMERA (IR TRIGGER)

-ENTRY:



Nếu IR Entry có xe VÀ chưa được kích hoạt (`!entryTriggered`)

Kiểm tra xem bãi có đầy không (`count >= MAX\_SLOTS`)

Nếu chưa đầy, gọi Server API /api/parking/entry

Nếu Server trả về "allow\_entry" thì gửi lệnh OPEN/CLOSE\_ENTRY tới Uno.

- EXIT:



Tương tự, nếu IR Exit có xe, gọi Server API /api/parking/exit

-> Nếu Server cho phép ("allow\_exit" hoặc "payment\_due"), gửi lệnh OPEN/CLOSE\_EXIT.

}

-LOGIC RFID (QUẸT THẺ)



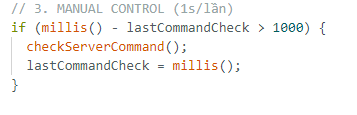
Dùng logic: Thử gọi Server API rfid/entry trước.

Nếu Server báo "deny\_entry" (thẻ đã vào), thì thử gọi rfid/exit

-> Nếu Server cho phép, gửi lệnh OPEN/CLOSE tương ứng tới Uno.

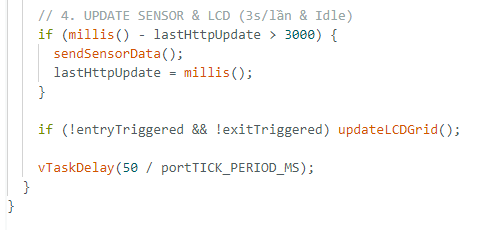
}

-MANUAL CONTROL (1s/lần)



if (millis() - lastCommandCheck > 1000) checkServerCommand();

-UPDATE SENSOR & LCD (3s/lần & Idle)



Logic I2C : Đọc 3 byte từ Uno. Tách 6 trạng thái cảm biến IR thành các biến logic (4 slot, IR Entry, IR Exit).

Logic Camera/IR : Sử dụng trạng thái IR Entry/Exit để kích hoạt (trigger) luồng kiểm tra Server (giả định đây là thời điểm camera chụp biển số). Logic kiểm tra chỗ đầy và xử lý phản hồi cho phép/từ chối của Server.

Logic RFID : Phát hiện thẻ mới, đọc UID . Sử dụng phương pháp thử và sai (try-catch logic):

Thử coi là sự kiện Entry trước.

Nếu Server báo thẻ đã ở trong bãi (`deny\_entry`), thử lại coi là sự kiện Exit .

Cập nhật định kỳ : Gọi các hàm `checkServerCommand()` và `sendSensorData()` theo chu kỳ thời gian (1s và 3s).

LCD : Cập nhật trạng thái lưới bãi đỗ xe chỉ khi không có sự kiện nào đang diễn ra (idle).

`vTaskDelay()` : Giải phóng CPU cho các Task khác, đây là chuẩn trong FreeRTOS.

- Hàm `setup()` và `loop()`

Hàm `setup()` thiết lập các thành phần, `loop()` để trống vì logic chính chạy trong Task FreeRTOS.

 Khởi tạo tất cả các giao tiếp phần cứng (I2C, LCD, SPI, RFID).

Kết nối Wi-Fi : Thực hiện vòng lặp chờ kết nối, hiển thị lên LCD.

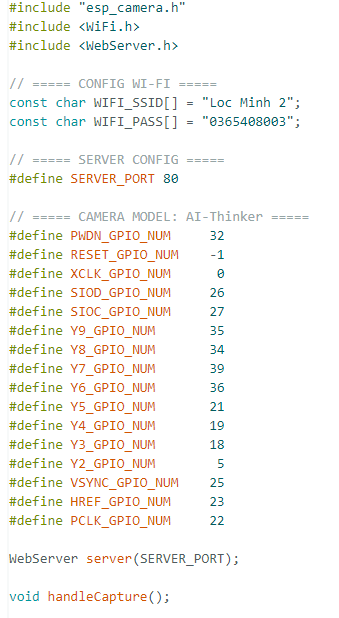
`xTaskCreate(...)` : QUAN TRỌNG . Tạo một Task mới tên "System" để chạy hàm `TaskSystem()`. Việc này giúp chương trình chạy ổn định và có thể thực hiện nhiều việc cùng lúc (ví dụ: Task Wi-Fi chạy nền trong khi `TaskSystem` xử lý I2C/RFID).

`loop()` : Được để trống, vì mọi logic đều được xử lý trong `TaskSystem` (FreeRTOS).

### 3.1.3. Code camera:

- Khai báo Thư viện và Cấu hình

Phần này thiết lập các thư viện cơ bản cho ESP32 và các hằng số cấu hình.



WebServer server(SERVER\_PORT); // Khởi tạo đối tượng Web Server trên cổng 80

Thư viện: Cần thiết cho chức năng camera, kết nối mạng và tạo API đơn giản.

Cấu hình Wi-Fi: Định nghĩa SSID và mật khẩu mạng Wi-Fi mà ESP32-CAM sẽ kết nối.

Cấu hình Server: Đặt cổng HTTP là 80 (cổng web mặc định).

Khai báo chân GPIO: Các định nghĩa PWDN\_GPIO\_NUM, XCLK\_GPIO\_NUM, v.v., là cấu hình bắt buộc và cố định cho module ESP32-CAM AI-Thinker (module phổ biến nhất).

- Hàm setup()

Hàm này thiết lập kết nối mạng, cấu hình camera, và khởi động máy chủ web.





Kết nối Wi-Fi: Code cố gắng kết nối Wi-Fi. Nếu thất bại sau 40 lần thử, ESP32 sẽ tự khởi động lại (ESP.restart()).

Cấu hình Camera: Đây là khối quan trọng nhất:

Thiết lập chi tiết các chân GPIO theo tiêu chuẩn của module.

Sử dụng định dạng PIXFORMAT\_JPEG và kích thước FRAMESIZE\_VGA (640x480). Kích thước này đủ để nhận dạng biển số xe mà vẫn đảm bảo tốc độ truyền tải nhanh.

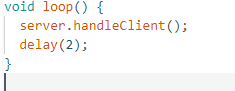
jpeg\_quality = 12: Đặt chất lượng thấp nhất để nén tối đa, giúp ảnh nhỏ hơn và truyền qua mạng nhanh hơn (quan trọng cho tốc độ xử lý LPR).

esp\_camera\_init(&config): Khởi tạo camera. Nếu lỗi, nó sẽ khởi động lại ESP32.

Khởi động Server: Tạo một endpoint (điểm cuối) tại /capture. Bất kỳ thiết bị nào truy cập URL này bằng phương thức HTTP\_GET sẽ kích hoạt hàm handleCapture.

- Hàm loop()

Hàm này là vòng lặp chính của ESP32.



server.handleClient(): Hàm bắt buộc để Web Server lắng nghe và xử lý các yêu cầu HTTP đến (ví dụ: yêu cầu /capture).

delay(2): Tạm dừng rất ngắn để không làm quá tải CPU.

- Hàm Xử lý /capture

Hàm này được gọi mỗi khi có yêu cầu HTTP đến /capture, và có nhiệm vụ chụp ảnh, đóng gói ảnh và gửi nó qua mạng.

C++

void handleCapture() {



camera\_fb\_t\* fb = esp\_camera\_fb\_get();: Lấy một khung hình (frame buffer) từ camera. Đây là hành động chụp ảnh thực tế. Nếu thất bại, gửi lỗi 500.

Thiết lập Header:

server.setContentLength(fb->len);: Đặt kích thước nội dung bằng với kích thước byte của ảnh JPEG.

server.send(200, "image/jpeg", "");: Gửi Header HTTP thành công (mã 200) và khai báo nội dung là image/jpeg. Nội dung (body) được để trống trong hàm này.

Gửi Dữ liệu Ảnh:

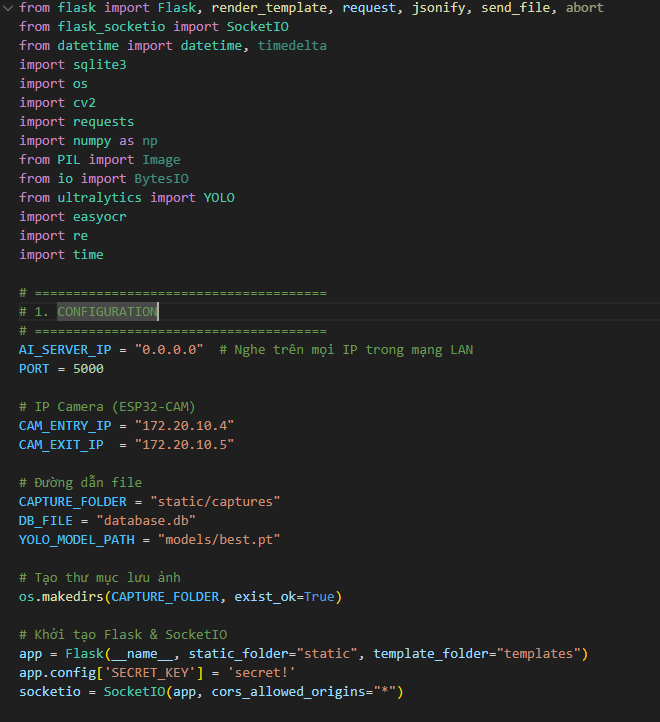
client.write(fb->buf, fb->len);: Sử dụng đối tượng client (kết nối mạng hiện tại) để gửi trực tiếp dữ liệu ảnh JPEG thô (fb->buf) với độ dài (fb->len) tới thiết bị yêu cầu.

esp\_camera\_fb\_return(fb);: Rất quan trọng. Trả lại bộ đệm đã sử dụng cho hệ thống camera để nó có thể được tái sử dụng cho lần chụp tiếp theo, tránh tràn bộ nhớ.

### 3.1.4. Giao diện:

-Khai báo Thư viện và Cấu hình

Phần này thiết lập môi trường Python, các thư viện cần dùng, và các hằng số mạng/đường dẫn.



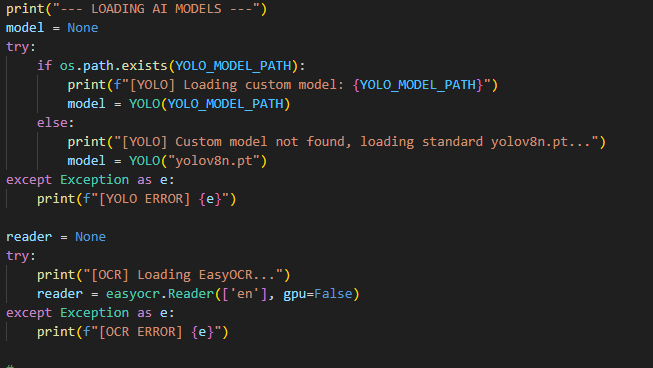
Thư viện: Nhập các module cơ bản như Flask (Web Framework), SocketIO (Real-time), sqlite3 (Database), requests (gọi HTTP), cv2 (OpenCV), YOLO, và easyocr (AI).

Cấu hình: Định nghĩa IP/Port của Server và địa chỉ IP cố định của hai camera ESP32-CAM tại cổng vào và ra.

Khởi tạo: Khởi tạo ứng dụng Flask và SocketIO (cho giao tiếp real-time với Web Dashboard).

- Tải Model AI

Tải các mô hình trí tuệ nhân tạo (AI) cần thiết ngay khi Server khởi động.

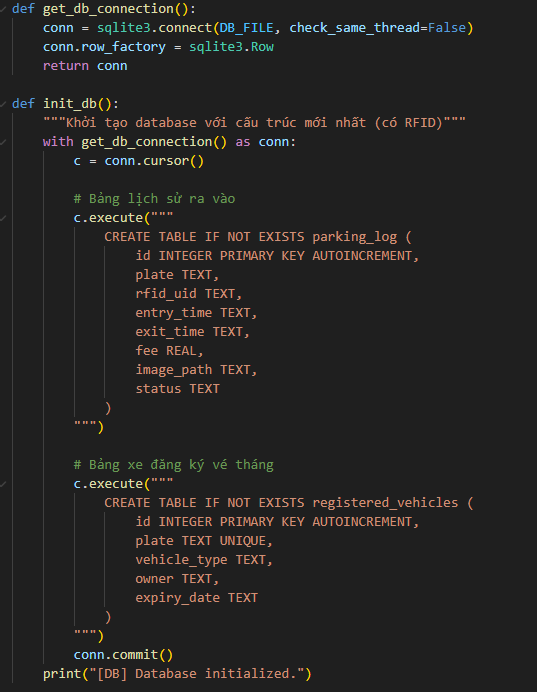


YOLO: Tải mô hình YOLOv8 (hoặc custom model best.pt) để thực hiện Phát hiện vật thể (tìm vị trí biển số).

EasyOCR: Tải thư viện EasyOCR (chỉ dùng tiếng Anh ['en']) để Nhận dạng ký tự quang học (đọc ký tự trên biển số).

- Hỗ trợ Database (SQLite)

Các hàm quản lý kết nối và cấu trúc dữ liệu SQLite.



get\_db\_connection(): Thiết lập kết nối cơ sở dữ liệu (Database) để truy cập dữ liệu đa luồng (multi-threaded).

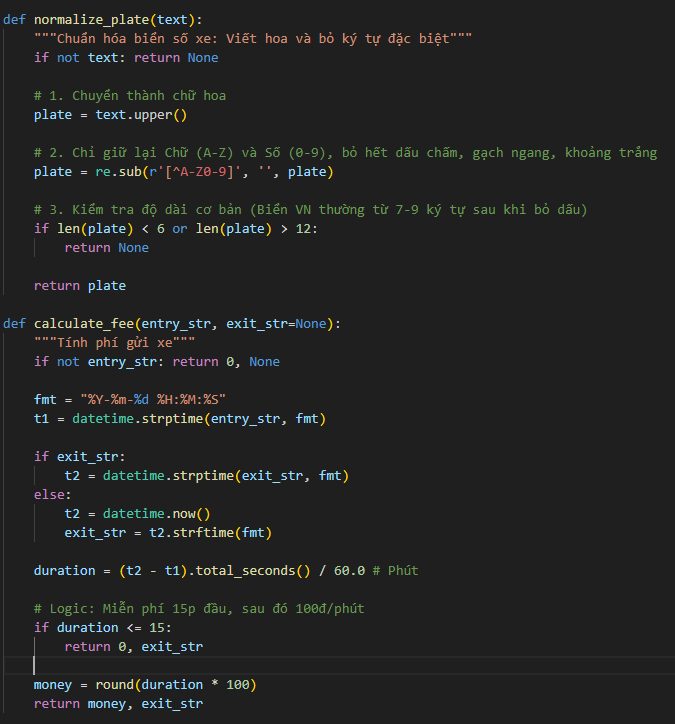
init\_db(): Tạo hai bảng:

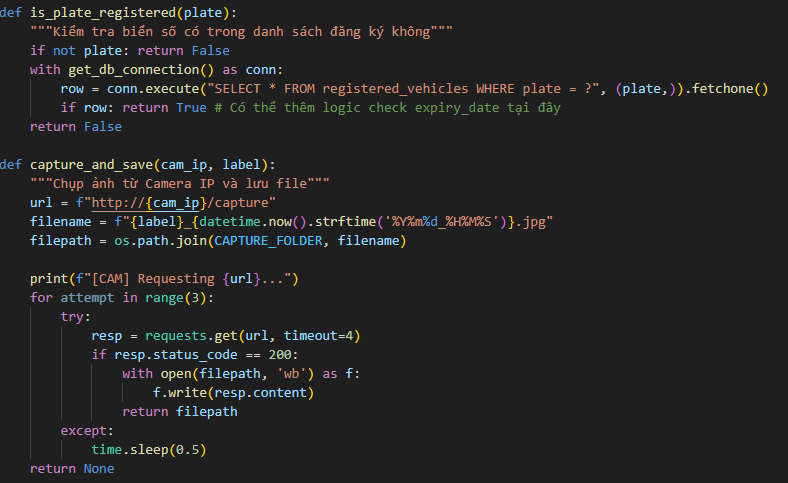
parking\_log: Lưu trữ toàn bộ lịch sử ra vào (biển số, UID, thời gian, phí, trạng thái).

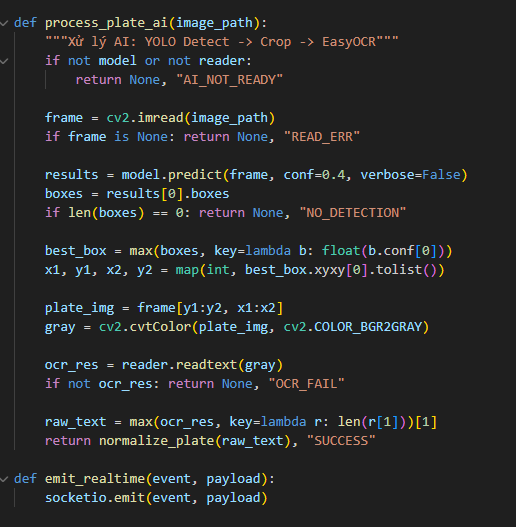
registered\_vehicles: Lưu trữ danh sách xe đã đăng ký (vé tháng).

- Logic Nghiệp vụ Cốt lõi

Các hàm Python chứa logic kinh doanh của bãi đỗ xe.







normalize\_plate(): Chuẩn hóa biển số (viết hoa, loại bỏ ký tự không cần thiết) để đảm bảo tính nhất quán.

calculate\_fee(): Tính phí dịch vụ theo thời gian gửi (logic: 15 phút miễn phí, sau đó 100 VNĐ/phút).

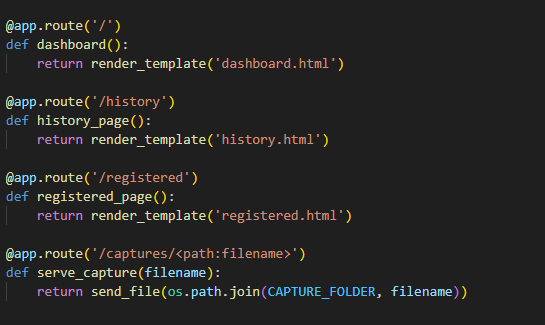
capture\_and\_save(): Gọi HTTP GET tới ESP32-CAM để lấy ảnh, sau đó lưu ảnh xuống ổ đĩa Server.

process\_plate\_ai(): Quá trình xử lý AI: YOLO phát hiện >Cắt ảnh biển số >EasyOCR đọc >Chuẩn hóa và trả về kết quả.

emit\_realtime(): Dùng SocketIO để gửi thông báo tức thì về Web Dashboard (ví dụ: thông báo có xe mới vào).

- API Web Routes (Giao diện người dùng)

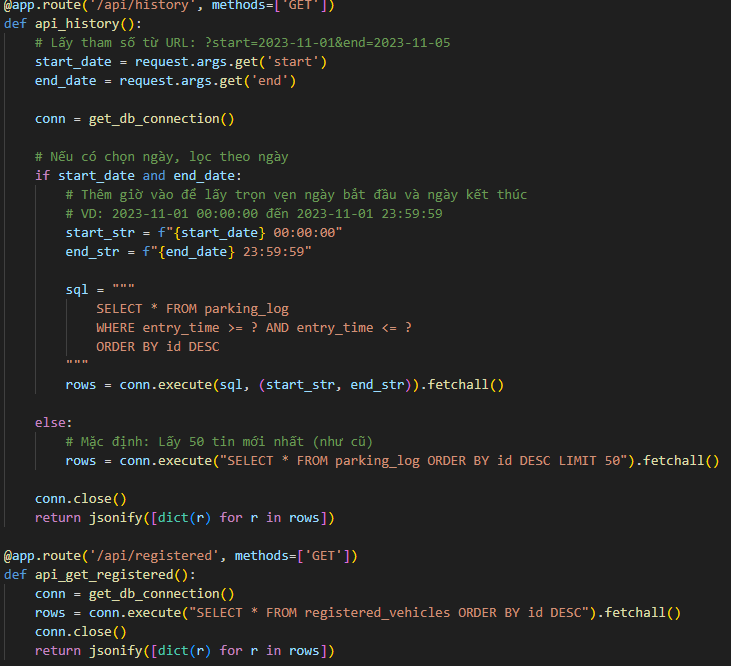
Các routes phục vụ các trang giao diện Web cho người quản lý.

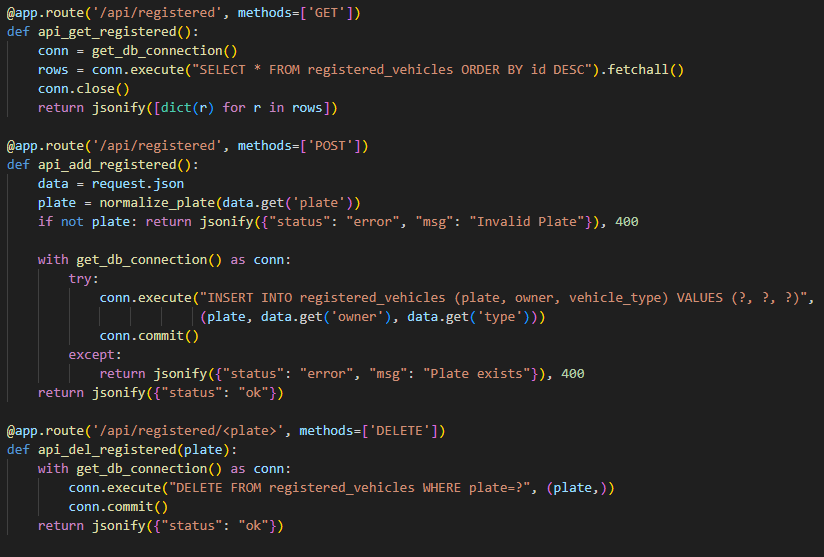
Các

route này chỉ đơn giản là trả về các file HTML (UI) đã được thiết kế sẵn (dashboard.html, history.html, v.v.)

- API Quản lý Dữ liệu

Các routes để tương tác với Database từ Web UI.





/api/history: Lấy dữ liệu lịch sử ra vào bãi đỗ xe (có hỗ trợ lọc theo ngày).

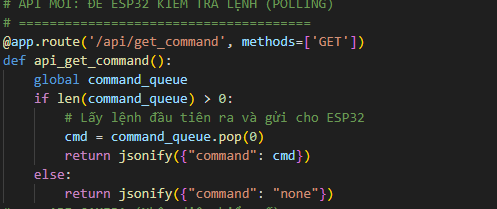
/api/registered: Thao tác CRUD (Thêm/Xóa/Lấy) danh sách xe đã đăng ký.

- API Logic Phần cứng

Các API trung tâm, được ESP32 Master gọi để thực thi các tác vụ chính.

- API Cập nhật và Điều khiển thủ công





/api/update\_data: Nhận trạng thái cảm biến (slot, MQ135) từ ESP32 và gửi cập nhật real-time lên Dashboard.

command\_queue: Dùng để quản lý các lệnh Mở/Đóng cửa thủ công do Admin tạo ra.

/api/get\_command: Cung cấp lệnh mới nhất từ hàng đợi cho ESP32 Master (thông qua cơ chế polling).

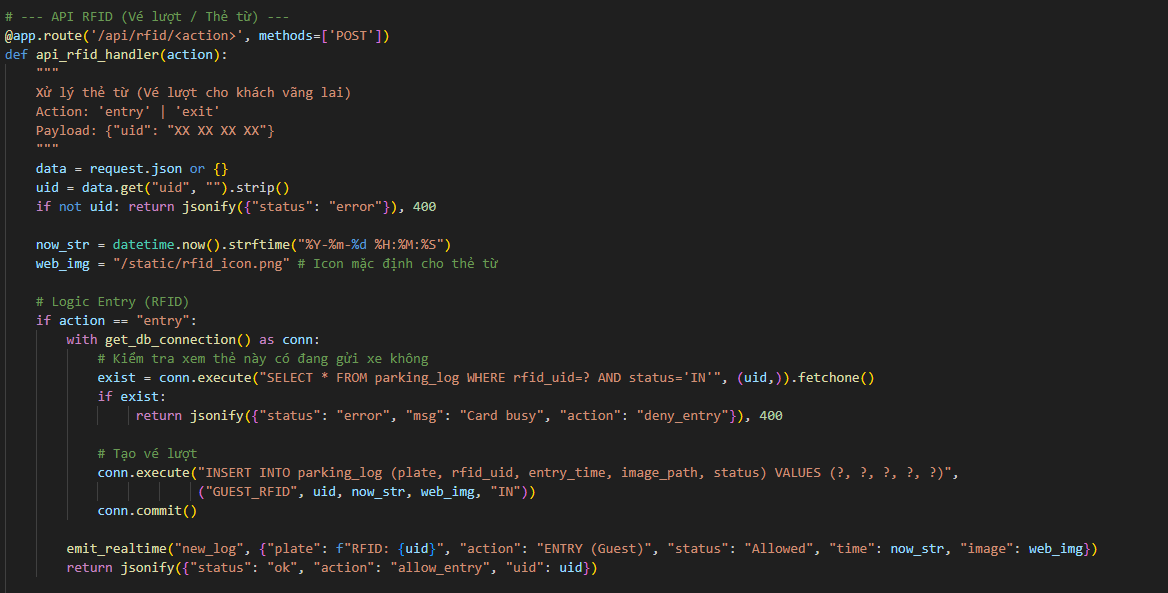
- Xử lý Camera (/api/parking/<action>)

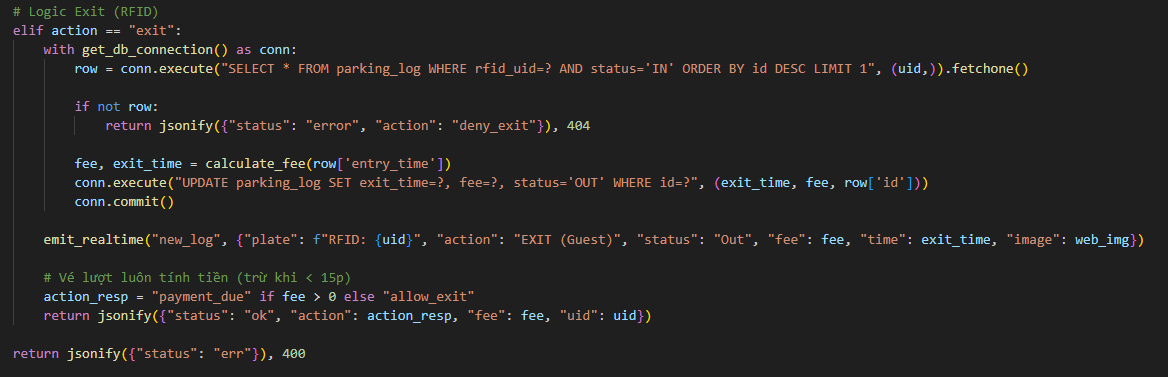
Python



API này điều phối toàn bộ quá trình LPR (License Plate Recognition). Nó là điểm quyết định chính trong hệ thống.

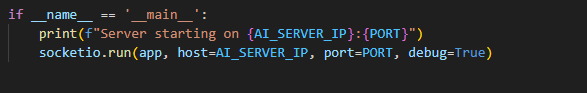
Xử lý RFID (/api/rfid/<action>)





Quản lý vé lượt cho khách vãng lai bằng cách theo dõi trạng thái UID của thẻ trong database, tính phí khi ra.

- Chạy Server

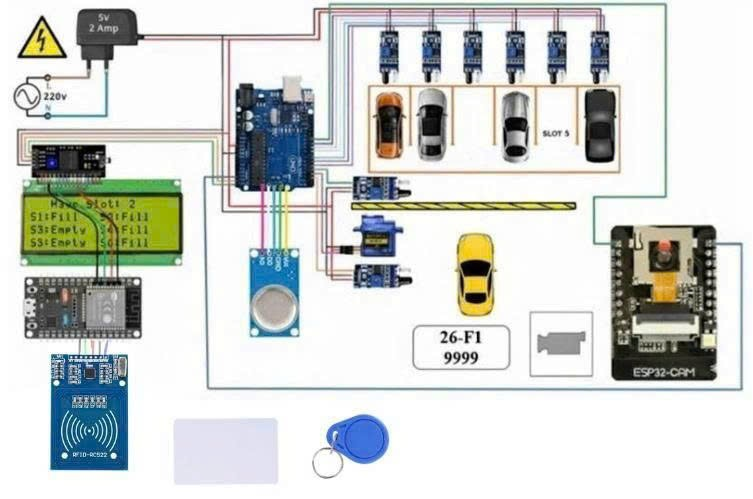


Khối lệnh này đảm bảo ứng dụng Flask/SocketIO được khởi chạy, lắng nghe các yêu cầu trên cổng 5000.

## 3.2. Lắp ráp mạch và kiểm tra hoạt động:

### 3.2.1. lắp ráp mạch:

-Sơ đồ lắp ráp mạch:



Hình 11. Sơ đồ lắp đặt mạch bãi đỗ xe Arduino

Giao tiếp I2C (Kết nối với ESP32 Master):

SDA (Data) >Chân A4 trên Arduino Uno.

SCL (Clock) >Chân A5 trên Arduino Uno.

GND >Chân GND (Nối đất chung).

VCC (5V) >Chân 5V trên Arduino Uno.

Điều khiển Servo:

Servo Entry (Signal) >Chân Digital 9 (D9).

Servo Exit (Signal) >Chân Digital 10 (D10).

Servo VCC >Nguồn điện ngoài 5V-6V (quan trọng nếu dùng nhiều Servo).

Servo GND >Chân GND (Nối đất chung).

Cảm biến Hồng ngoại (IR Sensors)

|  |  |
| --- | --- |
| **Chức năng** | **Chân Arduino Uno** |
| IR Entry (0) | Digital 2 (D2) |
| IR Exit (1) | Digital 3 (D3) |
| IR Slot 1 (2) | Digital 4 (D4) |
| IR Slot 2 (3) | Digital 5 (D5) |
| IR Slot 3 (4) | Digital 6 (D6) |
| IR Slot 4 (5) | Digital 7 (D7) |
| IR VCC | 5V |
| IR GND | GND |

Bảng 7. Đấu nối cảm biến IR với Arduino Uno

Cảm biến Chất lượng không khí (MQ-135)

MQ135 A0 (Analog Out) >Chân Analog 0 (A0).

MQ135 VCC >Chân 5V.

MQ135 GND >Chân GND.

### 3.2.2. Kiểm tra hoạt động:

Quy Trình Kiểm Tra Hệ thống Tích hợp (Arduino + Flask)

Khởi động và Trạng thái Ban đầu

Cắm nguồn và Tải Code:

Phần cứng: Hai động cơ Servo (cổng vào và cổng ra) phải đóng (góc 90)

Kiểm tra Màn hình/Serial:

LCD: Hiển thị trạng thái "O" cho cả 4 vị trí đỗ (Ví dụ: S1:O S2:O, S3:O S4:O).

Serial Monitor: In ra trạng thái: Slots: [S1:O] [S2:O] [S3:O] [S4:O] | Free: 4.

Kiểm tra Kết nối Server:

Hành động: Đảm bảo Arduino/ESP32 Master đã sẵn sàng gửi dữ liệu trạng thái cảm biến lên Server Flask.

Logic Backend: Server Flask đã sẵn sàng lắng nghe ở route /api/update\_data và command\_queue đang trống.

Kiểm tra Luồng Xe Vào (Entry)

A. Cho phép vào (Có chỗ trống)

Kích hoạt cảm biến vào: Che cảm biến cổng vào (SENSOR\_ENTRY - D2).

Logic Backend (Giả định): Đây là điểm để Backend Flask kiểm soát. Arduino sẽ gọi API LPR nếu đó là xe đăng ký, hoặc chờ RFID. Giả định đây là xe đăng ký và được phép vào.

Hành động: Arduino gọi API /api/parking/entry (hoặc /api/rfid/entry).

Phản hồi Server: Flask trả về lệnh allow\_entry.

Quan sát Servo:

Serial: Hệ thống sẽ in ra Slot available -> Opening Entry Gate.

Phản ứng cổng vào: Servo Entry phải mở ($0^\circ$), duy trì $1.5 \text{ giây}$, và sau đó đóng ($90^\circ$).

Cập nhật chỗ đỗ: Che cảm biến vị trí đỗ bất kỳ, ví dụ SLOT1 (D4).

Hành động: Arduino gọi API Flask /api/update\_data với trạng thái mới (Slot1: X).

Cập nhật hiển thị:

* + - LCD: Trạng thái của vị trí 1 phải chuyển thành "X".
    - Serial: Phải in ra Slots: [S1:X] [S2:O]... | Free: 3.

B. Từ chối vào (Bãi đầy)

* Làm đầy bãi đỗ: Lặp lại bước A.1-A.4 cho 3 vị trí còn lại (che SLOT2, SLOT3, và SLOT4) cho đến khi cả 4 vị trí đều hiển thị "X" và Serial in ra | Free: 0.
* Kích hoạt cảm biến vào lần nữa: Che cảm biến cổng vào (SENSOR\_ENTRY - D2).
* Logic Backend: Arduino kiểm tra biến nội bộ (hoặc Server Flask gửi lại trạng thái đầy). Giả định Arduino tự kiểm tra.

**Quan sát Phản ứng:**

* Serial: Hệ thống phải in ra Parking FULL – Entry Denied.
* LCD: Màn hình hiển thị Parking FULL! và Entry Denied trong $1.5 \text{ giây}$.
* Servo Entry: Rào chắn phải giữ nguyên trạng thái đóng ($90^\circ$).

**Kiểm tra Luồng Xe Ra (Exit)**

* Kích hoạt cảm biến ra: Khi bãi đang đầy (hoặc có ít nhất một xe), che cảm biến cổng ra (SENSOR\_EXIT - D3).
* Logic Backend: Đây là điểm để Backend Flask tính phí.
* Hành động: Arduino gọi API /api/parking/exit (hoặc /api/rfid/exit) để tính phí và lấy lệnh mở cổng.
* Phản hồi Server (Giả định): Flask trả về lệnh allow\_exit (giả định phí bằng 0).

**Quan sát Servo:**

* Serial: Hệ thống sẽ in ra Xe ra -> Mo cong ra.
* Phản ứng cổng ra: Rào chắn ra (Servo Exit) phải mở ($0^\circ$), sau đó đóng ($90^\circ$) trở lại.
* Cập nhật chỗ đỗ: Ngay sau đó, bỏ che cảm biến vị trí đỗ mà chiếc xe vừa rời đi (ví dụ: SLOT1 - D4).
* Hành động: Arduino gọi API Flask /api/update\_data với trạng thái mới (Slot1: O).

**Cập nhật hiển thị:**

LCD: Trạng thái của vị trí đó phải chuyển thành "O" (Trống).

Serial: Phải in ra | Free: 1 (hoặc số chỗ trống mới).

Kiểm tra Lệnh Điều khiển Thủ công (Manual Control)

Hành động: Admin trên Web UI Server Flask ra lệnh Mở cổng VÀO thủ công (gọi /api/control/open\_entry).

Logic Backend: Lệnh OPEN\_ENTRY được đẩy vào command\_queue trên Server Flask.

Hành động Arduino: ESP32 Master định kỳ gọi API /api/get\_command.

Quan sát Servo:

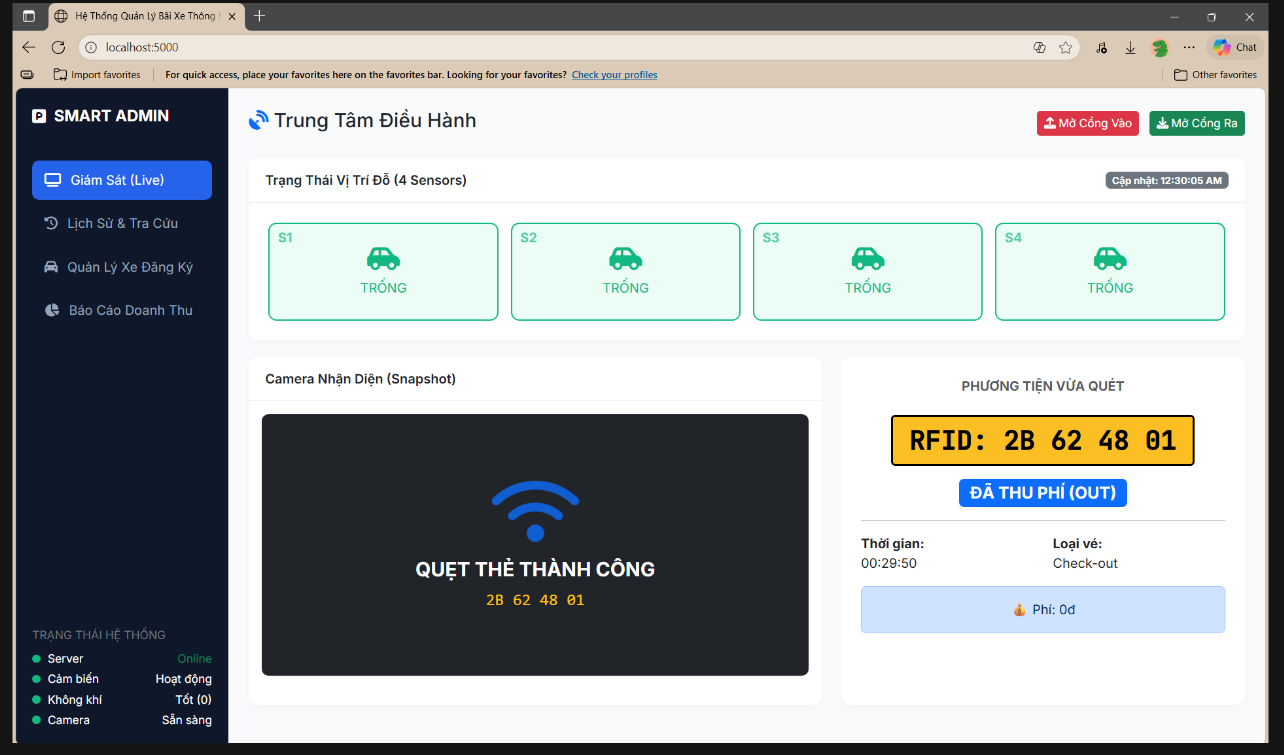
Serial: Arduino nhận lệnh và in ra Manual Command: OPEN\_ENTRY.

Phản ứng cổng vào: Rào chắn vào (Servo Entry) phải mở ($0^\circ$) và giữ nguyên trạng thái này (cho đến khi có lệnh đóng).

## 3.3. Chạy thử nghiệm và đánh giá:

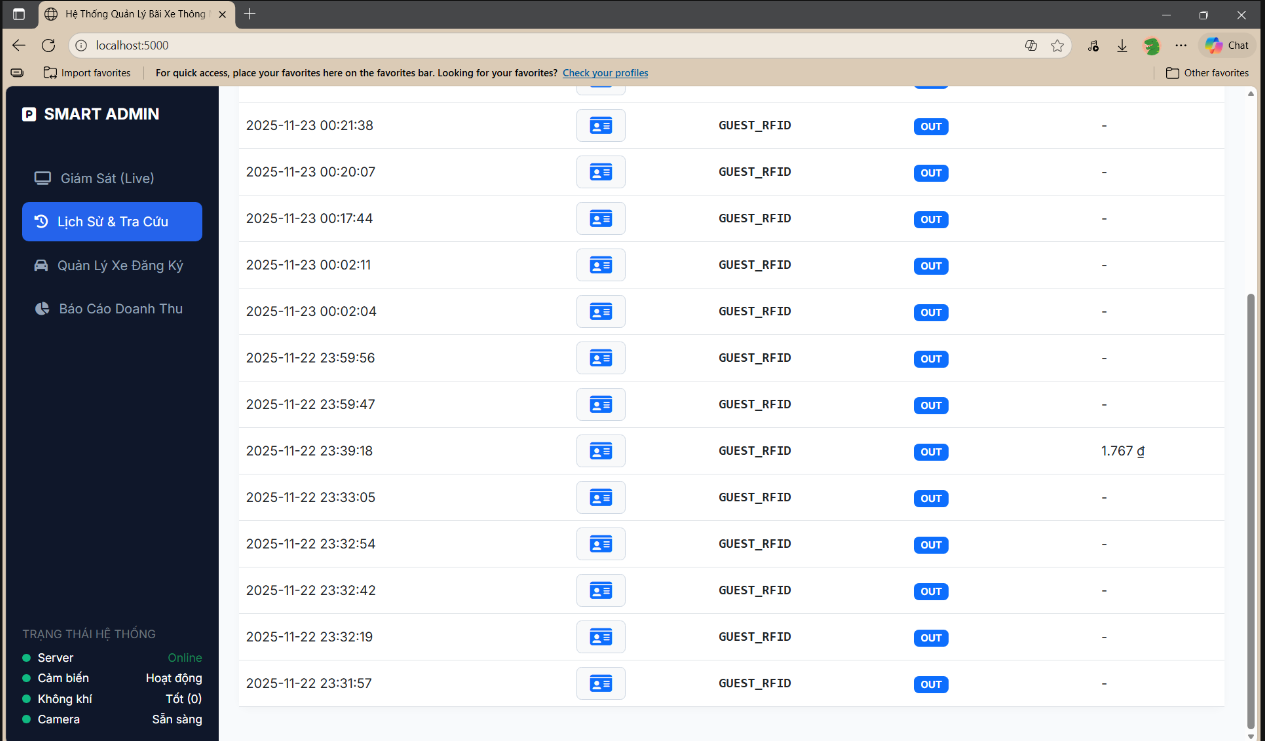
### 3.3.1. Giao diện web:

Giao diện chính của trang web khi truy cập.



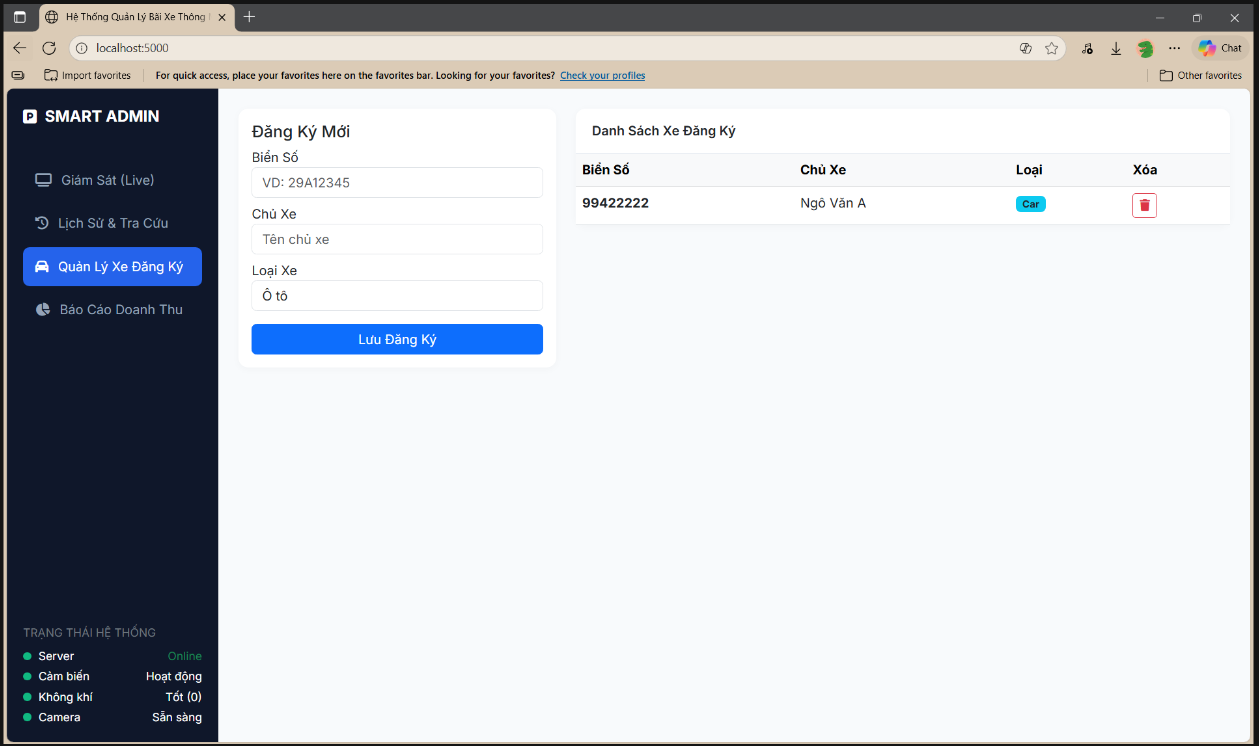
Hình 12. Mô phỏng giao diện web

Lịch sử và tra cứu thông tin dữ liệu xe.



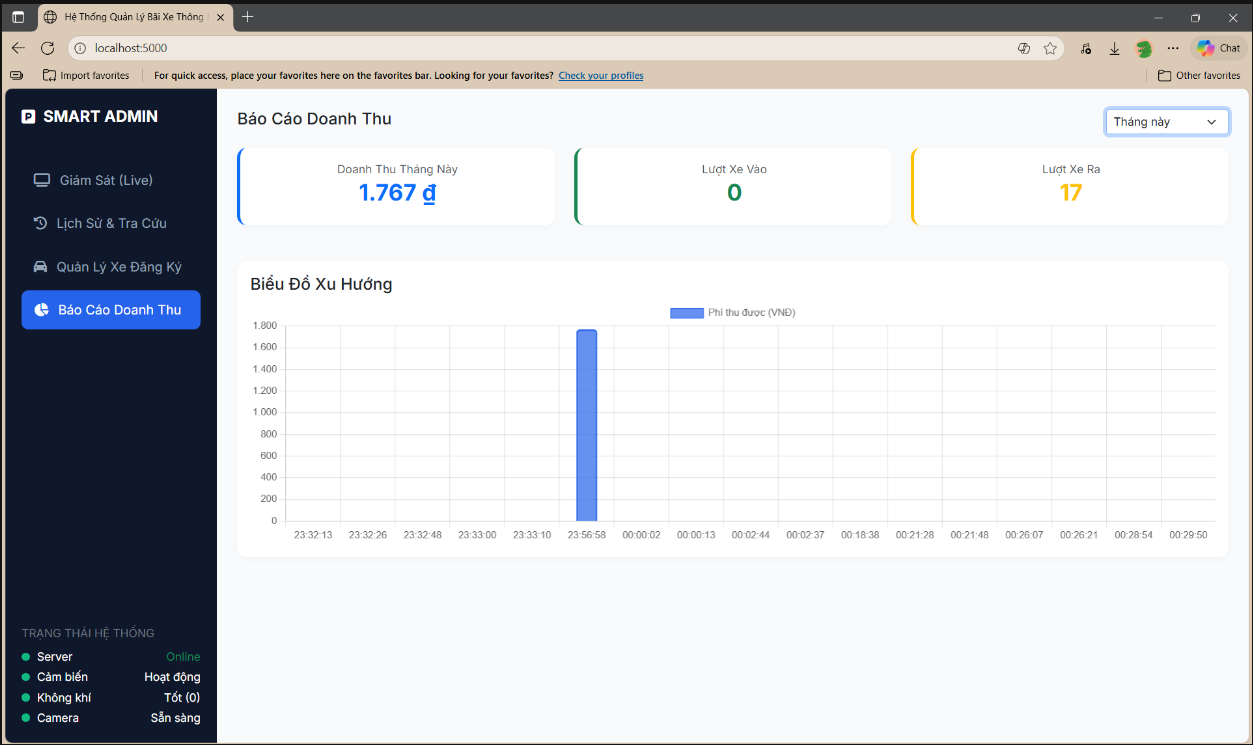
Hình 13. Mô phỏng hiển thị lịch sử

Quản lý xe đăng ký và hiện thị danh sách.



Hình 14. Mô phỏng quản lý xe

Báo cáo thu tiền gửi xe.



Hình 15. Mô phỏng hiển thị báo cáo doanh thu

***3.3.2. Kiểm tra và chạy thử.***

Khi xe vào sẽ phải quẹt thẻ.



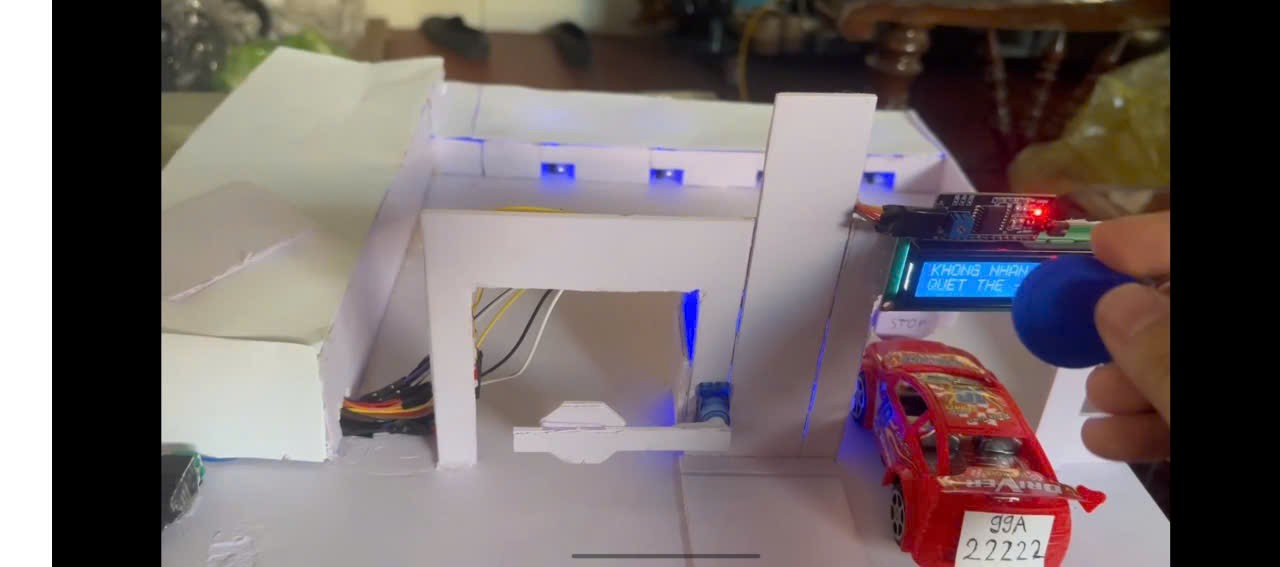
Hình 16. Mô phỏng khi xe vào

Hoặc khi xe vào camera sẽ soi biển số.



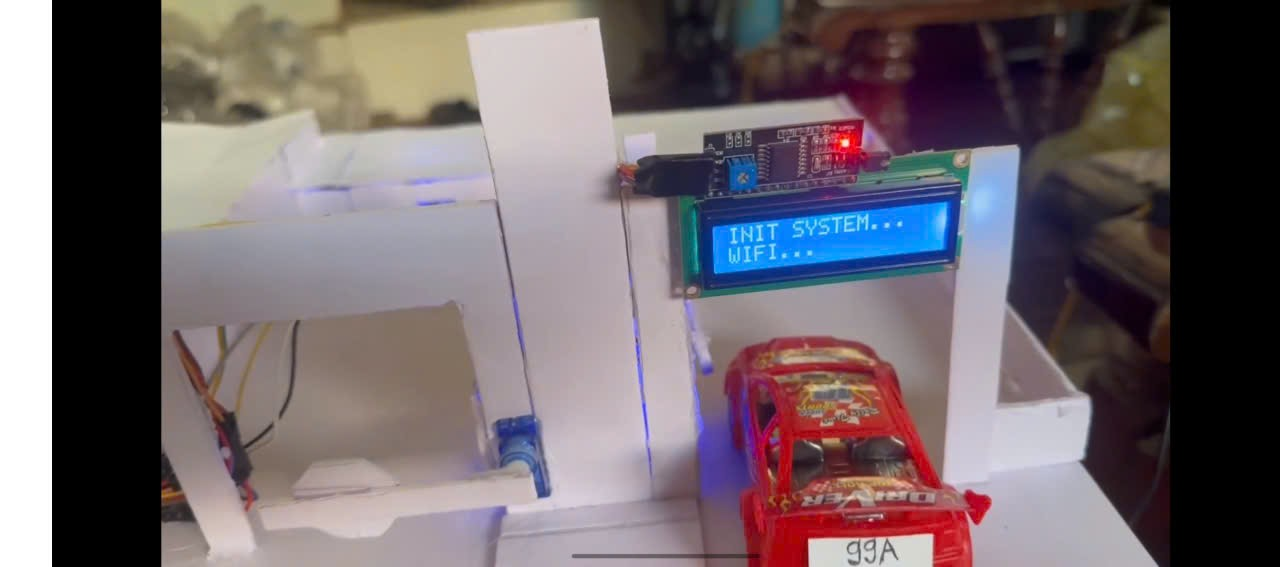
Hình 17. Mô phỏng khi xe vào

Cổng sẽ không mở khi chưa quẹt thẻ.



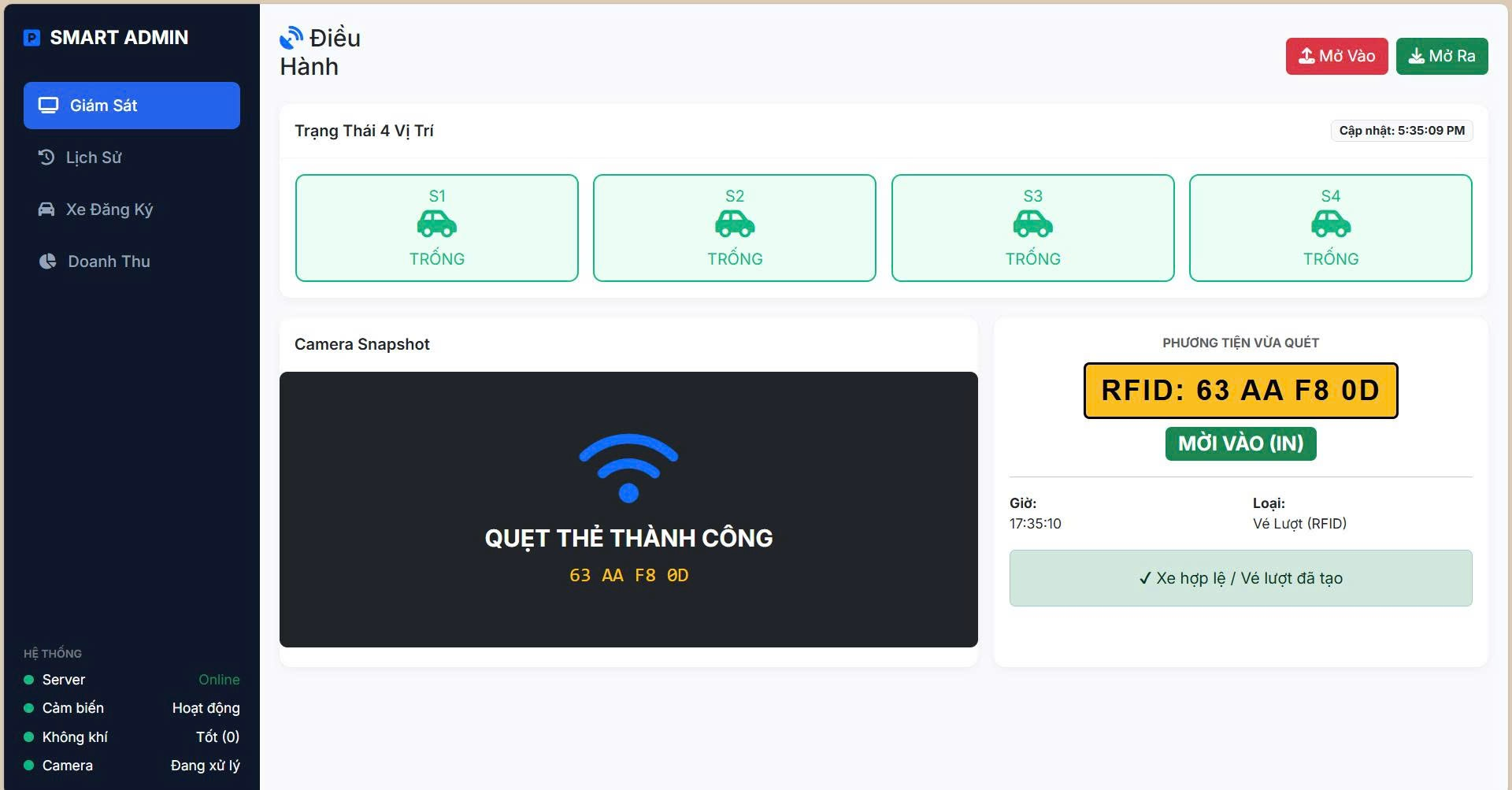
Hình 18. Mô phỏng cơ chế quẹt thẻ

Khi xe vào và quẹt thẻ cổng sẽ mở.

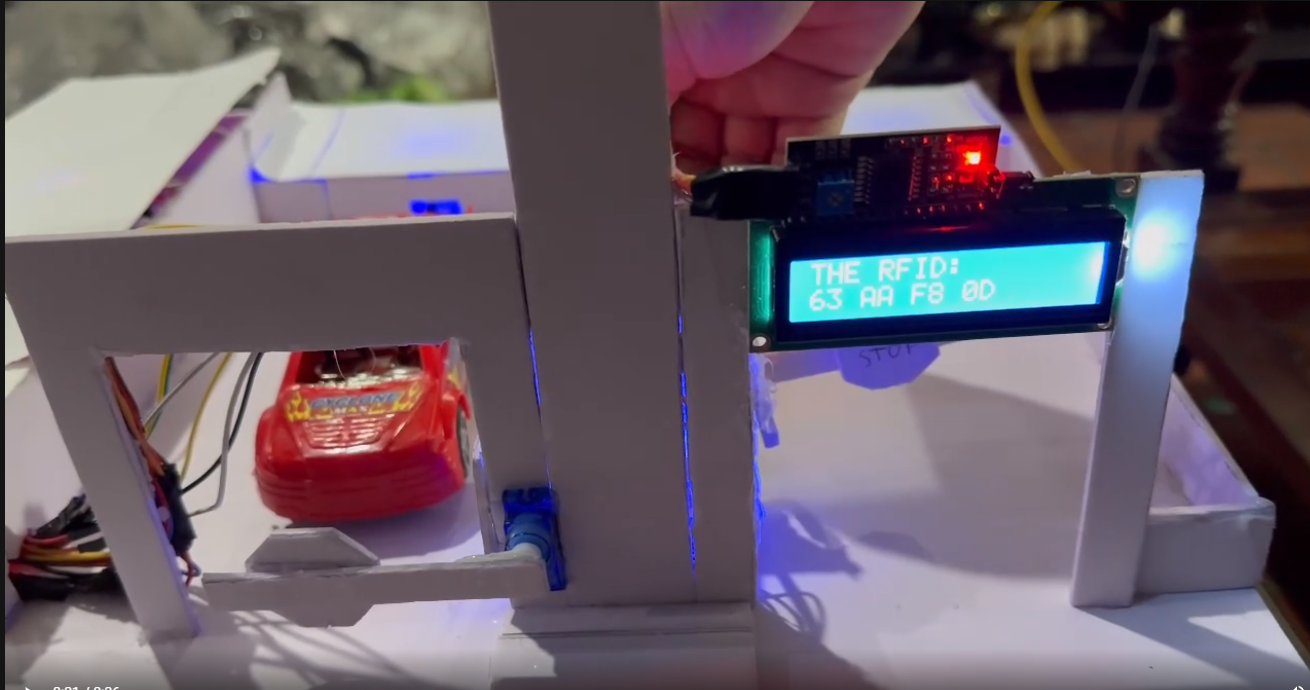


Hình 19. Mô phỏng khi cửa đã quẹt xong và tự mở

Khi xe vào chỗ sẽ tải dữ liệu lên web:



Hình 20. Mô phỏng dữ liệu xe vào được lưu trữ

Quẹt thẻ để cho xe ra

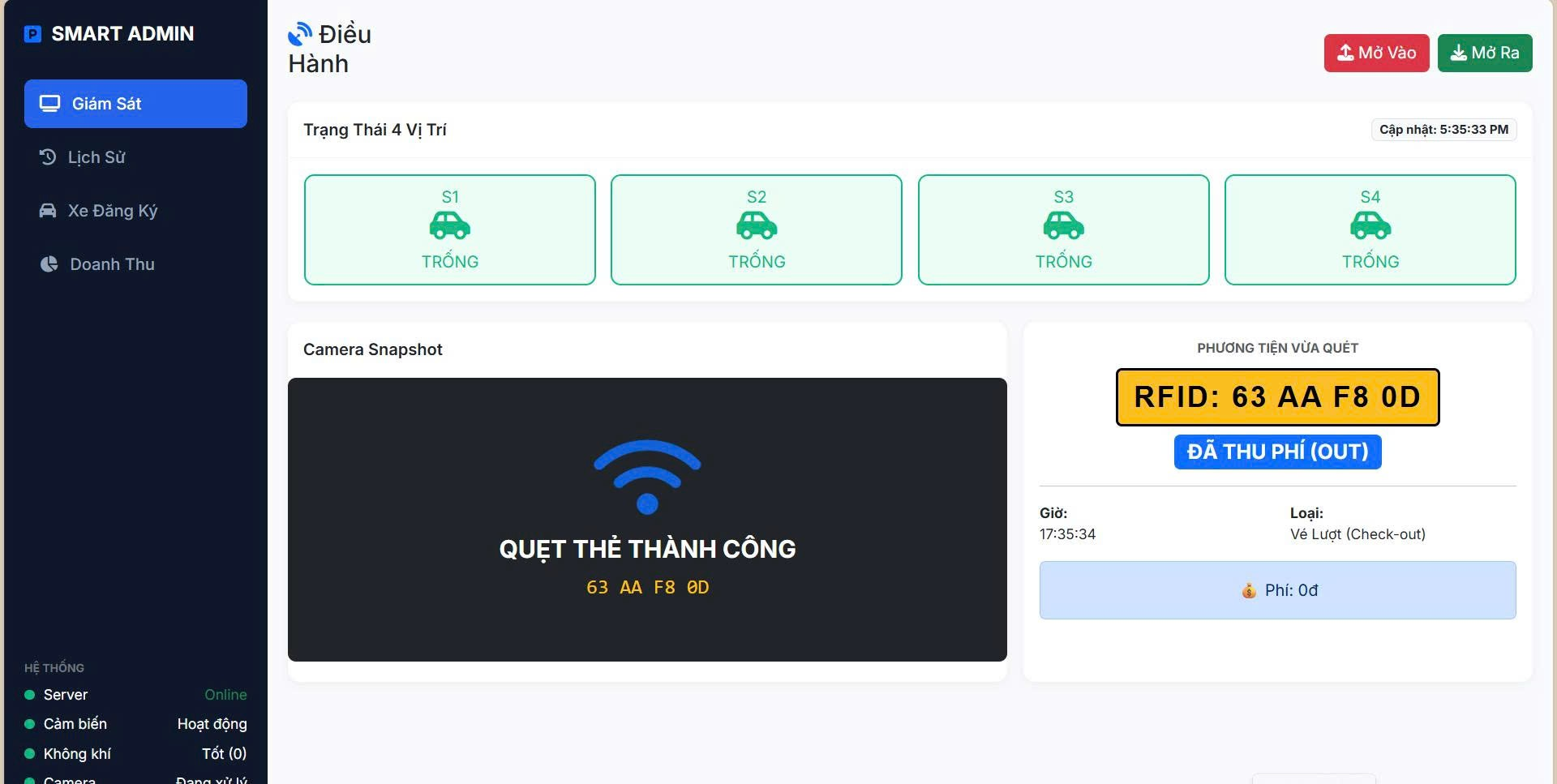
Hình 21. Mô phỏng cơ chế quẹt thẻ cho xe ra

Cửa sẽ tự động mở khi xe ra:



Hình 22. Mô phỏng cơ chế mở cổng tự động

Và tự động đóng :



Hình 23. Hiển thị xe ra thành công

### 

### 3.3.2. đánh giá:

Việc sử dụng Arduino Uno làm Slave để thu thập dữ liệu cảm biến và thực hiện lệnh điều khiển (actuator) là một chiến lược thiết kế hệ thống phân tán rất hiệu quả. Code có tính tổ chức cao, sử dụng các hằng số và macro một cách hợp lý, đặc biệt là kỹ thuật làm mượt dữ liệu (smoothing) cho cảm biến MQ135 cho thấy sự quan tâm đến chất lượng dữ liệu.

**Ưu Điểm Nổi Bật**

Cấu Trúc Rõ Ràng: Code được tổ chức tốt với các hàm riêng biệt cho các tác vụ (setup, loop, readSensors, requestEvent, receiveEvent). Việc sử dụng các #define và mảng giúp mã dễ đọc và bảo trì.

Giao tiếp I2C Hiệu quả: Đã triển khai chính xác giao thức I2C ở chế độ Slave với cáchàm callback Wire.onRequest (gửi dữ liệu) và Wire.onReceive (nhận lệnh).

Xử lý Dữ liệu MQ135 Tốt: Việc sử dụng kỹ thuật làm mượt (smoothing/running average) bằng mảng mqBuffer và tính tổng mqSum giúp giảm nhiễu (noise) đáng kể từ cảm biến Analog, tăng độ tin cậy của giá trị mqValue.

Sử dụng Ngắt (Interrupt) cho I2C: Sử dụng volatile cho sensorStates và mqValue là chính xác, vì các biến này được chia sẻ giữa hàm loop (đọc cảm biến) và hàm ngắt requestEvent (gửi I2C).

Logic Servo Rõ ràng: Đã xác định rõ ràng logic điều khiển servo (MỞ = 0, ĐÓNG = 90) và sử dụng nó nhất quán trong setup và receiveEvent.

**Khả năng Cải Thiện**

Xử lý I2C Lệnh Nhận: Trong receiveEvent, việc sử dụng String để xây dựng lệnh nhận (cmd += c) có thể gây ra hiện tượng phân mảnh bộ nhớ (memory fragmentation) trên Arduino Uno (bộ nhớ RAM hạn chế).

Đề xuất: Nên chuyển sang sử dụng mảng ký tự (char[]) và hàm so sánh chuỗi C tiêu chuẩn (strcmp) hoặc dùng các ký tự lệnh đơn giản (ví dụ: 'A' cho OPEN\_ENTRY, 'B' cho CLOSE\_ENTRY) để tránh vấn đề này.

Tên Biến/Hằng số: Việc sử dụng F() macro (ví dụ: Serial.println(F("--- UNO READY ---"))) là tốt, nhưng bạn có thể thêm nó cho tất cả các chuỗi in ra Serial (Serial.println("CMD: ENTRY OPEN (0 deg)");) để lưu trữ chúng trong bộ nhớ Flash (Program Memory) thay vì RAM.

Đọc Cảm biến Tần suất cố định: Hàm loop() chỉ chứa readSensors() và delay(50). Điều này tạo ra chu kỳ đọc cảm biến 1/50ms = 20Hz, khá tốt. Tuy nhiên, nếu bạn muốn các tác vụ chạy với tần suất khác nhau hoặc muốn code không bị chặn, bạn có thể sử dụng phương pháp non-blocking delay (dùng millis()) thay cho delay(50).

# KẾT LUẬN

Hệ thống bãi đỗ xe thông minh mang lại nhiều ưu điểm nổi bật như khả năng tự động hóa quá trình quản lý, giảm sai sót của con người và tiết kiệm thời gian cho người sử dụng. Việc áp dụng cảm biến và bộ điều khiển giúp theo dõi và hiển thị chính xác số lượng xe trong bãi. Đồng thời, mô hình còn giúp nâng cao hiệu quả vận hành nhờ khả năng hoạt động ổn định và dễ dàng triển khai trong thực tế. Đây là cơ sở quan trọng để phát triển các ứng dụng tự động hóa trong môi trường đô thị hiện đại.

Bên cạnh những ưu điểm, hệ thống vẫn tồn tại một số hạn chế nhất định, chủ yếu do quy mô mô hình và mức độ đơn giản của thiết bị. Các cảm biến có thể gặp sai số trong môi trường nhiều nhiễu hoặc khi khoảng cách nhận diện không ổn định. Ngoài ra, mô hình chưa tích hợp nhiều tính năng nâng cao như bảo mật, kiểm soát vé xe hay nhận dạng biển số, khiến khả năng ứng dụng thực tiễn còn bị hạn chế. Những yếu tố này cần được cải thiện để hệ thống đáp ứng tốt hơn nhu cầu thực tế.

Trong tương lai, hệ thống có thể được phát triển theo hướng mở rộng chức năng và nâng cao độ chính xác của các thiết bị cảm biến. Việc bổ sung các công nghệ như nhận dạng biển số, kết nối IoT và ứng dụng điều khiển qua điện thoại sẽ giúp tối ưu hóa trải nghiệm người dùng. Mô hình cũng có thể được mở rộng thành một hệ thống quản lý bãi đỗ xe hoàn chỉnh, phù hợp với nhu cầu của các khu vực đông phương tiện. Đây sẽ là hướng đi quan trọng để nâng cao tính thông minh và hiệu quả cho các bãi đỗ xe hiện đại.

# DANH MỤC TÀI LIỆU THAM KHẢO

1. Trần Minh Hoàng (2018), *Giáo trình Hệ thống nhúng và ứng dụng*, NXB Khoa học & Kỹ thuật.
2. Lê Văn Dũng (2019), *Điện tử cơ bản dành cho sinh viên công nghệ*, NXB Giáo dục Việt Nam.
3. Phạm Quốc Khánh (2020), *Cảm biến và kỹ thuật đo lường trong tự động hóa*, NXB Bách Khoa.
4. Nguyễn Thị Ngọc Anh (2021), *Kỹ thuật vi điều khiển AVR và Arduino*, NXB Lao động – Xã hội.