BÁO CÁO DỰ ÁN IOT

ĐỀ TÀI: THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG QUẢN LÝ BÃI ĐỖ XE THÔNG MINH (SMART PARKING SYSTEM) DỰA TRÊN NỀN TẢNG IOT VÀ AI

TÓM TẮT

Báo cáo này trình bày chi tiết quy trình nghiên cứu, thiết kế và triển khai hệ thống quản lý bãi đỗ xe thông minh nhằm giải quyết bài toán quản lý không gian đỗ xe tại các đô thị lớn. Hệ thống được xây dựng dựa trên kiến trúc Internet of Things (IoT) 4 lớp, sử dụng 5 loại thiết bị ESP32 (Node Sensor, Gate Controller, 2 Camera, Monitor) kết hợp với Backend Python (FastAPI), Database (MySQL), MQTT Broker (Mosquitto) và công nghệ AI-OCR (Plate Recognizer API).

Điểm nổi bật của hệ thống bao gồm:

1. Cơ chế Debounce 500ms: Loại bỏ nhiễu cảm biến, đảm bảo độ chính xác của dữ liệu đầu vào.

2. OTA Firmware Update: Hỗ trợ cập nhật từ xa theo cơ chế streaming với bộ đệm (buffer) 128 bytes, giúp tối ưu hóa bộ nhớ.

3. Real-time Communication: Sử dụng WebSocket với Background Worker để cập nhật trạng thái thời gian thực.

4. Manual Override: Cơ chế điều khiển thủ công sử dụng FreeRTOS Task và Mutex để đảm bảo an toàn và ưu tiên quyền điều khiển.

5. Giao diện phản hồi trực quan: Hệ thống đèn LED RGB hiển thị 7 trạng thái màu khác nhau thông báo tình trạng hoạt động.

Kết quả thực nghiệm cho thấy hệ thống hoạt động ổn định với độ trễ dưới 2 giây, độ chính xác nhận diện biển số đạt 90%, hỗ trợ đầy đủ tính năng quản lý từ xa qua Dashboard web và Monitor màn hình TFT.

MỤC LỤC

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1. Lý do chọn đề tài

Sự bùng nổ dân số và tốc độ đô thị hóa nhanh chóng tại Việt Nam đã dẫn đến sự gia tăng chóng mặt của các phương tiện giao thông cá nhân. Theo thống kê năm 2023, Hà Nội và TP.HCM đang đối mặt với tình trạng thiếu hụt bãi đỗ xe trầm trọng, chỉ đáp ứng được khoảng 10-15% nhu cầu thực tế. Các bãi đỗ xe truyền thống hiện nay chủ yếu vận hành thủ công, gây ra nhiều vấn đề như ùn tắc cục bộ, lãng phí thời gian tìm chỗ đỗ, và khó khăn trong quản lý doanh thu.

Xuất phát từ thực tế đó, việc nghiên cứu và xây dựng "Hệ thống quản lý bãi đỗ xe thông minh" là vô cùng cấp thiết, giúp giải quyết bài toán giao thông tĩnh, hướng tới mô hình thành phố thông minh (Smart City).

1.2. Mục tiêu nghiên cứu

Mục tiêu tổng quát: Xây dựng một hệ thống IoT hoàn chỉnh có khả năng giám sát, quản lý và điều phối hoạt động của bãi đỗ xe một cách tự động, kết hợp công nghệ AI để nhận diện biển số xe.

Mục tiêu cụ thể:

- Thiết kế và lắp ráp mạng lưới cảm biến nhỏ gọn, tối ưu chi phí.
- Xây dựng phần mềm quản lý trung tâm hiển thị trực quan trạng thái bãi đỗ theo thời gian thực (Real-time Dashboard).
- Tích hợp AI-OCR để nhận diện biển số xe tự động, thay thế thẻ từ truyền thống.
- Thực hiện cơ chế cập nhật firmware từ xa (OTA) streaming để thuận tiện cho việc bảo trì và nâng cấp.
- Xây dựng Monitor vật lý (TFT Display) để hiển thị thông tin tại chỗ cho khách hàng
- Tích hợp tính năng mở cổng trong trường hợp khẩn cấp.

1.3. Đối tượng và Phạm vi nghiên cứu

- **Đối tượng:**
    - Các công nghệ IoT: Vi điều khiển ESP32, giao thức MQTT.
    - Công nghệ xử lý ảnh: ESP32-CAM, OCR API.
    - Công nghệ Web: Backend FastAPI, WebSocket, Template Engine Jinja2.

- Phạm vi nghiên cứu:
    - Mô hình bãi đỗ xe trong nhà (Indoor Parking).
    - Quy mô: 8 vị trí đỗ xe (Slot A1-A4, B1-B4).
    - Hệ thống cổng ra vào: 1 cổng tích hợp barrier servo.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT VÀ CÔNG NGHỆ

2.1. Tổng quan về Internet of Things (IoT)

Internet of Things (IoT) là một hệ thống mạng lưới kết nối khổng lồ, nơi các vạn vật (Things) - bao gồm các thiết bị vật lý, phương tiện, thiết bị gia dụng và các vật dụng khác - được nhúng cảm biến, phần mềm, bộ truyền động và công nghệ kết nối khác. Mục tiêu cốt lõi là cho phép các đối tượng này kết nối, thu thập và trao đổi dữ liệu với nhau và với các hệ thống khác qua mạng Internet mà không cần sự can thiệp trực tiếp của con người.

Cấu trúc cơ bản của một hệ thống IoT thường được chia thành bốn lớp:

1. **Lớp Cảm biến (Sensing Layer):** Bao gồm các cảm biến (sensor) và bộ truyền động (actuator) thu thập dữ liệu từ môi trường vật lý (ví dụ: nhiệt độ, ánh sáng, vị trí xe) và thực hiện các hành động.
2. **Lớp Mạng (Network/Communication Layer):** Đảm bảo việc truyền dữ liệu đã thu thập đến các hệ thống xử lý. Lớp này sử dụng các giao thức khác nhau như Wi-Fi, 4G/5G, Bluetooth, Zigbee, và đặc biệt là các giao thức nhẹ như **MQTT** (được sử dụng trong dự án này).
3. **Lớp Xử lý Dữ liệu (Data Processing/Service Layer):** Dữ liệu thô được gửi lên **Cloud** hoặc **Edge Server** để được lưu trữ, phân tích, và xử lý để chuyển thành thông tin có ý nghĩa.
4. **Lớp Ứng dụng (Application Layer):** Cung cấp các dịch vụ cuối cùng cho người dùng, chẳng hạn như giao diện người dùng trên điện thoại di động, web, hoặc các hệ thống quản lý (ví dụ: ứng dụng quản lý bãi đỗ xe thông minh).

2.2. Các công nghệ phát hiện trạng thái chỗ đỗ xe

Việc xác định chính xác trạng thái **Trống/Có xe** tại mỗi chỗ đỗ là nền tảng của hệ thống đỗ xe thông minh. Hiện nay có nhiều công nghệ được áp dụng, mỗi công nghệ có ưu và nhược điểm riêng:

Công nghệ	Nguyên lý hoạt động	Ưu điểm	Nhược điểm
1. Cảm biến Siêu âm (Ultrasonic)	Phát sóng âm tần số cao; đo thời gian phản xạ để tính khoảng cách.	Độ chính xác cao cho khoảng cách ngắn.	Góc quét hẹp, độ bao phủ hạn chế, dễ bị ảnh hưởng bởi vật cản khác (ví dụ: lắp đặt sai).

2. Cảm biến Từ trường (Magnetometer)	Đo sự thay đổi của từ trường Trái Đất khi có vật kim loại (xe) đi qua.	Chính xác, bền bỉ, chống chịu thời tiết tốt (thường được chôn dưới đất).	Chi phí cao, quy trình lắp đặt và bảo trì phức tạp (đào, chôn), khó thay thế.
3. Camera (Image Processing)	Sử dụng thuật toán Xử lý ảnh/Thị giác máy tính (Computer Vision) để phân tích luồng video/ảnh chụp.	Bao phủ diện rộng (có thể giám sát nhiều chỗ đỗ cùng lúc), cung cấp bằng chứng hình ảnh.	Chi phí xử lý tính toán lớn (GPU/Server), dễ bị ảnh hưởng bởi góc khuất, điều kiện ánh sáng (đêm, chói nắng) và che khuất.
4. Cảm biến Hồng ngoại (IR Sensor)	Phát tia hồng ngoại; phát hiện vật cản dựa trên sự hấp thụ hoặc phản xạ của tia.	Giá thành rẻ, dễ lắp đặt/tháo lắp, phản hồi nhanh.	Dễ bị ảnh hưởng bởi ánh nắng mạnh (Hồng ngoại Mặt trời), khoảng cách phát hiện ngắn (phải lắp đặt sát).
5. Cảm biến Cân nặng (Loadcell)	Đo trọng lượng tác dụng lên bề mặt cảm biến.	Độ chính xác cao (xác định chắc chắn có vật thể nặng).	Độ bền thấp nếu công suất bãi đỗ cao, chi phí lắp đặt dưới nền bãi đỗ tốn kém.

Lựa chọn Công nghệ của Dự án: Cảm biến Hồng ngoại (IR Sensor)

Dự án này quyết định sử dụng Cảm biến Hồng ngoại (IR Sensor) cho mục đích phát hiện xe với các lý do chiến lược sau:

- Hiệu quả Chi phí: Giá thành cực kỳ cạnh tranh (chỉ khoảng 10.000 VNĐ/cái), cho phép triển khai số lượng lớn cảm biến với ngân sách thấp.
- Đơn giản hóa Triển khai: Kích thước nhỏ gọn, dễ dàng lắp đặt trên trần hoặc tường (đối với bãi đỗ trong nhà) hoặc trên cột.
- Phản hồi nhanh: Cung cấp tín hiệu nhị phân (LOW/HIGH) gần như ngay lập tức, lý tưởng cho việc giám sát trạng thái theo thời gian thực.
    - Output: LOW (Mức 0V) $\Longrightarrow$ Có xe (Vật cản)
    - Output: HIGH (Mức 5V) $\Longrightarrow$ Trống (Không có vật cản)

2.3. Công nghệ nhận diện biển số tự động (ALPR/OCR)

Automatic License Plate Recognition (ALPR), còn được gọi là **ANPR** (Automatic Number Plate Recognition) hay **LPR**, là một công nghệ thông minh sử dụng **nhận dạng ký tự quang học (OCR)** trên hình ảnh để tự động đọc, giải mã và trích xuất thông tin biển số xe. Đây là thành phần thiết yếu để quản lý ra/vào và tính toán thời gian đỗ xe.

Quy trình ALPR

Quá trình ALPR diễn ra qua một chuỗi các bước xử lý hình ảnh và thuật toán máy học:

1. **Ảnh đầu vào:** Thu thập hình ảnh từ camera giám sát tại cổng ra/vào.
2. **Tiền xử lý Ảnh:** Cải thiện chất lượng ảnh để tối ưu cho việc nhận dạng. Các bước thường bao gồm:
    - **Chuyển về thang độ xám (Grayscale):** Giảm thiểu thông tin màu sắc không cần thiết.
    - **Làm mờ (Blurring/Smoothing):** Giảm nhiễu (noise) và làm mịn các cạnh.
    - **Cân bằng sáng (Histogram Equalization):** Điều chỉnh độ tương phản.
3. **Phát hiện Vùng Biển số (License Plate Detection):** Sử dụng các mô hình **Học sâu (Deep Learning)** như YOLO, SSD, hoặc các thuật toán xử lý ảnh truyền thống để xác định và **cô lập (localize)** khu vực chứa biển số xe.
4. **Tách Ký tự (Character Segmentation):** Tách riêng từng ký tự số và chữ cái trên biển số đã được cô lập.
5. **Nhận dạng Ký tự (Character Recognition):** Áp dụng công nghệ **OCR** để chuyển đổi từng hình ảnh ký tự đã tách thành dữ liệu văn bản số.
6. **Văn bản Đầu ra (Final Output):** Kết hợp các ký tự đã nhận dạng để tạo thành chuỗi biển số hoàn chỉnh (ví dụ: "30A12345").

Lựa chọn Công nghệ: Plate Recognizer API

Dự án này sử dụng dịch vụ **Plate Recognizer API**. Đây là một mô hình Học sâu **dựa trên nền tảng Cloud (Cloud-based Deep Learning model)**.

- **Ưu điểm:** Độ chính xác được báo cáo là **trên 90%** cho các định dạng biển số của Việt Nam, khả năng xử lý hình ảnh kém chất lượng tốt hơn các thuật toán truyền thống.
- **Lợi ích:** Loại bỏ gánh nặng tính toán lớn và phức tạp của việc tự xây dựng và duy trì mô hình Học sâu trên Server vật lý của dự án.

2.4. Giao thức truyền thông MQTT

MQTT (Message Queuing Telemetry Transport) là một giao thức nhắn tin theo mô hình **Publish/Subscribe (Xuất bản/Đăng ký)**, được thiết kế đặc biệt để phục vụ cho các ứng dụng **IoT/M2M (Machine-to-Machine)**. Điểm mạnh lớn nhất của MQTT là tính **cực kỳ nhẹ (extremely lightweight)**, tiêu tốn băng thông tối thiểu và yêu cầu năng lượng thấp, làm cho nó trở thành lựa chọn lý tưởng cho các thiết bị bị giới hạn về tài nguyên như **ESP32** và các cảm biến.

Mô hình Publish/Subscribe (Xuất bản/Đăng ký) Không giống như mô hình Client-Server truyền thống, MQTT hoạt động thông qua ba thành phần chính:

1. **Broker (Nhà môi giới):** Là máy chủ trung tâm (server), chịu trách nhiệm nhận tất cả tin nhắn từ **Publisher** và phân phối chúng đến tất cả **Subscriber** đã đăng ký theo **Topic** (chủ đề) tương ứng.
    - **Dự án sử dụng: Mosquitto** (một Broker mã nguồn mở, nhẹ và phổ biến).
2. **Publisher (Bên xuất bản):** Thiết bị hoặc ứng dụng gửi dữ liệu.
    - **Dự án sử dụng: ESP32** (gửi dữ liệu trạng thái chỗ đỗ) và **Python Server** (gửi lệnh điều khiển cổng).
3. **Subscriber (Bên đăng ký):** Thiết bị hoặc ứng dụng nhận dữ liệu.
    - **Dự án sử dụng: ESP32** (nhận lệnh điều khiển cổng) và **Python Server** (nhận dữ liệu trạng thái chỗ đỗ).
    - **Topic:** Là chuỗi ký tự phân cấp (ví dụ: `bai_do_xe/cho_A01/trang_thai`) mà các Publisher gửi tin nhắn và các Subscriber đăng ký để nhận tin.

Quality of Service (QoS) - Chất lượng Dịch vụ

MQTT cung cấp ba cấp độ QoS để đảm bảo độ tin cậy của việc truyền tải tin nhắn:

QoS	Tên gọi	Chi tiết Đảm bảo	Ứng dụng trong Dự án
0	**At most once**	Tin nhắn được gửi đi **một lần**, không đảm bảo nhận được (best effort). **Tối ưu tốc độ.**	**Dữ liệu cảm biến** (trạng thái chỗ đỗ): Tốc độ là ưu tiên, việc mất một vài mẫu dữ liệu có thể chấp nhận được vì dữ liệu mới sẽ nhanh chóng được gửi lại.
1	**At least once**	Tin nhắn được đảm bảo nhận được **ít nhất một lần**. Broker/Client sẽ lặp lại việc gửi nếu không nhận được xác nhận (ACK).	**Lệnh điều khiển cổng:** Cần **đảm bảo** lệnh mở/đóng cổng được thiết bị (ESP32) nhận và thực hiện.

2	**Exactly once**	Tin nhắn được đảm bảo nhận được **chính xác một lần** (hai chiều handshake phức tạp).	Không được sử dụng trong dự án do quá phức tạp và tốn băng thông, không cần thiết cho mục đích này.

2.5. Công nghệ Web và WebSocket

WebSocket là một giao thức kết nối hai bên (client/server) thông qua một kênh duy nhất, cho phép truyền dữ liệu hai chiều (như HTTP) nhưng với hiệu suất cao hơn và ít tốn băng thông hơn so với giao thức HTTP thông thường.

WebSocket trong Dự án Dự án sử dụng WebSocket để cung cấp kết nối hai chiều giữa ESP32 và Python Server. Cụ thể:

ESP32 (Client) gửi dữ liệu trạng thái chỗ đỗ và nhận lệnh điều khiển cổng.
Python Server (Server) nhận dữ liệu và gửi lệnh điều khiển.
Ưu điểm của WebSocket trong Dự án

Hiệu suất cao: Không cần phải mở kết nối mới mỗi khi gửi tin nhắn, giảm lượng dữ liệu cần truyền tải.
Tốc độ cao: Thời gian phản hồi nhanh chóng, phù hợp cho các ứng dụng cần thời gian phản hồi ngắn (như việc mở/đóng cổng).
Tự động reconnection: Nếu kết nối bị ngắt, WebSocket sẽ tự động tái kết nối, đảm bảo tính ổn định.

2.6. Template Engine Jinja2

Jinja2 là một template engine phổ biến trong Python, được sử dụng để tạo ra các trang web động từ các template tĩnh. Trong dự án này, Jinja2 được sử dụng để tạo các trang web động từ các template tĩnh.

Ứng dụng trong Dự án Dự án sử dụng Jinja2 để tạo các trang web động từ các template tĩnh. Cụ thể:

ESP32 (Client) gửi dữ liệu trạng thái chỗ đỗ và nhận lệnh điều khiển cổng.
Python Server (Server) nhận dữ liệu và gửi lệnh điều khiển.

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

3.1. Kiến trúc tổng thể

Hệ thống được xây dựng dựa trên sự phối hợp của 5 loại thiết bị ESP32 chuyên biệt, đảm bảo khả năng xử lý phân tán và chịu tải tốt:

- **NODE Sensor (ESP32):** Chịu trách nhiệm đọc dữ liệu từ 2 cảm biến hồng ngoại (IR Sensor), xử lý tín hiệu (debounce) và gửi trạng thái qua giao thức MQTT. Thiết bị cũng hỗ trợ nhận lệnh cập nhật firmware từ xa (OTA).
- **GATE Controller (ESP32):** Bộ điều khiển trung tâm tại cổng, kết nối với cảm biến IN/OUT, điều khiển động cơ Servo (Barrier), dải đèn LED WS2812 RGB báo hiệu, và gửi tín hiệu trigger chụp ảnh cho Camera.
- **CAM_IN (ESP32-CAM):** Camera chuyên dụng tại cổng vào, thực hiện chụp ảnh độ phân giải cao khi nhận tín hiệu trigger và upload hình ảnh lên Server qua giao thức HTTP.
- **CAM_OUT (ESP32-CAM):** Tương tự CAM_IN nhưng được bố trí tại cổng ra để giám sát xe rời khỏi bãi.
- **MONITOR (ESP32 + TFT Display):** Thiết bị hiển thị tại chỗ, cung cấp sơ đồ trạng thái slot thời gian thực cho khách hàng và tích hợp nút bấm vật lý để mở cổng khẩn cấp.

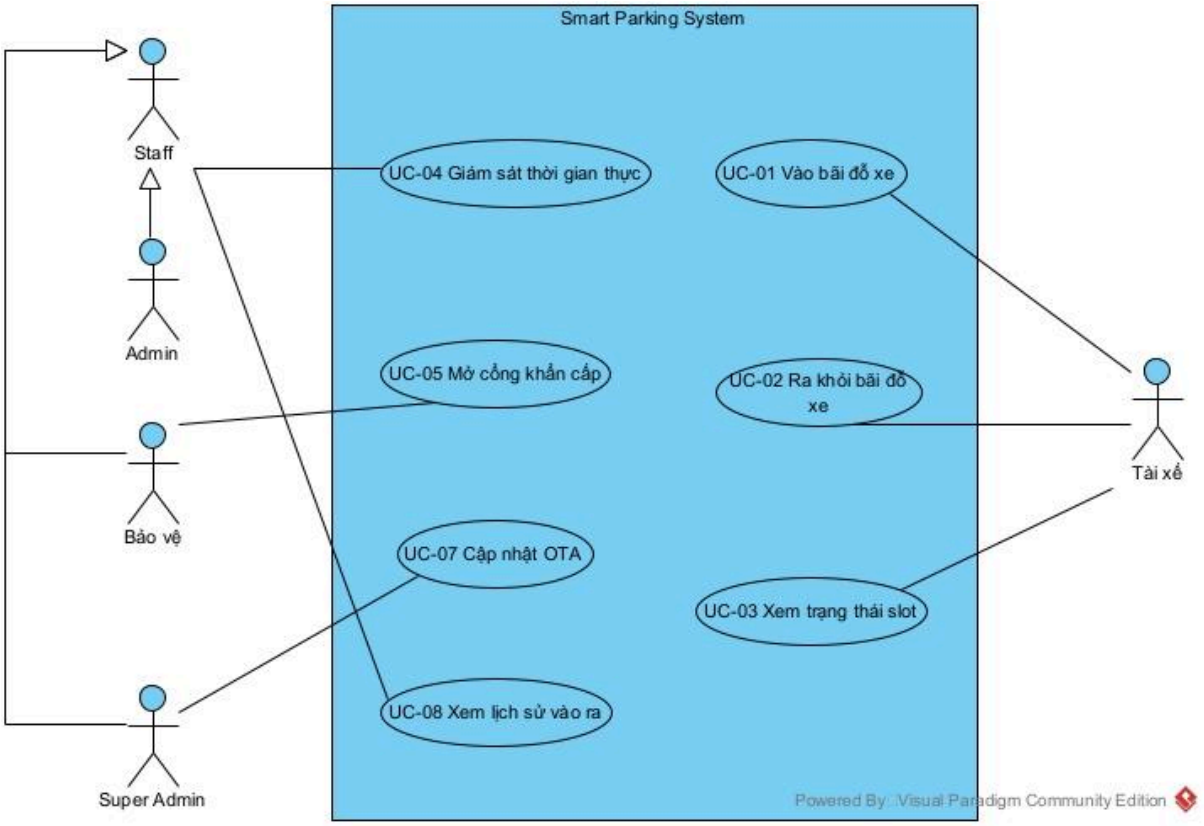
3.2. Use Case và mô tả

3.2.1. Actors

Actor	Vai trò trong hệ thống	Use Case Tương tác Chính
Tài xế (Driver)	Đối tượng sử dụng chính của bãi đỗ xe.	Vào/Ra Bãi đỗ xe (UC-01, UC-02), Xem trạng thái slot (UC-03)
Quản trị viên (Admin)	Người quản lý, giám sát hệ thống qua Dashboard và thực hiện một số tác vụ kỹ thuật.	Giám sát Real-time (UC-04), Yêu cầu Mở cổng khẩn cấp (UC-05)
Super admin	Quản lý toàn bộ hệ thống, có quyền cao nhất trong hệ thống	Giám sát Real-time (UC-04), Yêu cầu Mở cổng khẩn cấp (UC-05), Quản lý các admin, thực hiện cập nhật, nâng cấp hệ thống

Bảo vệ (Guard)	Người giám sát vật lý tại cổng, thực hiện thao tác khẩn cấp tại chỗ	Yêu cầu Mở cổng khẩn cấp (UC-05), Hiển thị Trạng thái tại chỗ (UC-6)
Hệ thống Đỗ xe (Parking System)	Đại diện cho các hành động tự động của hệ thống (phần cứng/phần mềm)	Phát hiện và báo cáo trạng thái chỗ đỗ (UC-07)

3.2.2. Use Cases



3.2.3. Chi tiết Use Cases

UC-01: Vào bãi đỗ xe

Mục tiêu	Tài xế hoàn tất việc đưa xe vào bãi đỗ tự động
Tác nhân	Tài xế
Pre-conditions	Xe dừng ở đúng vị trí cảm biến IR
Post-condition	Xe vào được bãi đỗ và có log, lưu ảnh trong uploads/archive/, dashboard hiện xe vào
Sự kiện chính	1. Tài xế lái xe đến cổng 2. Hệ thống tự động kích hoạt nhận diên biển số

	3. Hệ thống xác minh biển 4. Hệ thống mở cổng 5. Tài xế lái xe qua cổng 6. Hệ thống tự động đóng cổng sau khi tài xế đi vào
Ngoại lệ	3. Hệ thống không xác minh được biển số 3.1. Hệ thống giữ cổng đóng

UC-02: Ra khỏi bãi đỗ xe

Mục tiêu	Tài xế hoàn tất việc đưa xe ra khỏi bãi đỗ xe một cách tự động
Tác nhân	Tài xế
Pre-conditions	Xe của Tài xế dừng đúng vị trí cảm biến IR OUT. Xe đã có log vào trước đó trong DB.
Post-conditions	Xe đã rời khỏi bãi đỗ. Log (exit_timestamp) đã được cập nhật.
Main Flow	1. Tài xế lái xe đến cổng ra. 2. Hệ thống tự động kích hoạt nhận diện biển số. 3. Hệ thống kiểm tra biển số: đã từng vào và đang ở trong bãi đỗ. 4. Hệ thống mở cổng, cập nhật log (action = exit). 5. Tài xế lái xe qua cổng ra. 6. Hệ thống tự động đóng cổng sau khi xe đi qua.
Ngoại lệ	3. Xe không có log vào 3.1. Hệ thống không tìm thấy log vào tương ứng cho biển số. 3.2. Hệ thống từ chối mở cổng và bật LED đỏ.

UC-03: Xem trạng thái slot (Dashboard)

Mục tiêu	Tài xế hoặc Admin kiểm tra tình trạng chỗ đỗ xe đang Trống/Đầy theo thời gian thực.
Tác nhân	Tài xế
Pre-conditions	Các NODE sensor đang gửi dữ liệu trạng thái chỗ đỗ lên Server.
Post-conditions	Tác nhân đã nắm bắt được trạng thái chỗ đỗ hiện tại.
Sự kiện chính	1. Tác nhân truy cập Dashboard Web (hoặc Bảo vệ nhìn vào Monitor TFT). 2. Hệ thống hiển thị sơ đồ chỗ đỗ (grid 4x2) với màu sắc trực quan (Xanh/Đỏ). 3. Hệ thống thiết lập kết nối real-time (WebSocket/MQTT) để nhận cập nhật. 4. Hệ thống tự động cập nhật màu sắc chỗ đỗ ngay lập tức khi có sự

	kiện thay đổi trạng thái.
Ngoại lệ	1. Lỗi kết nối Real-time 1.1. Dashboard/Monitor mất kết nối dữ liệu real-time. 1.2. Giao diện hiển thị dữ liệu cũ và thông báo lỗi.

UC-04: Giám sát real-time

Mục tiêu	Admin theo dõi toàn bộ hoạt động, sự kiện và tình trạng kỹ thuật của hệ thống 24/7.
Tác nhân	Nhân viên(Staff)
Pre-conditions	Server và tất cả các thiết bị đang hoạt động.
Post-conditions	Admin đã có cái nhìn toàn diện về hoạt động của hệ thống.
Sự kiện chính	1. Admin truy cập Dashboard admin 2. Hệ thống hiển thị các chỉ số tổng quan: Tổng slot (8), Số slot trống, Số slot có xe. 3. Hệ thống hiển thị danh sách 10 xe vào/ra gần nhất. 4. Hệ thống sử dụng WebSocket để tự động cập nhật tất cả thông tin này khi có sự kiện mới (Slot thay đổi, Xe vào/ra).
Ngoại lệ	2. Lỗi thiết bị 2.1. NODE sensor mất kết nối (time out).

UC-05: Xem lịch sử vào ra

Mục tiêu	Admin theo dõi toàn bộ sự kiện
Tác nhân	Quản trị viên (Admin)
Pre-conditions	Server và tất cả các thiết bị đang hoạt động.
Post-conditions	Admin đã xem được log vào ra của các phương tiện
Sự kiện chính	1. Admin truy cập Dashboard và chọn mục Lịch sử vào ra. 2. Hệ thống hiển thị danh sách 10 xe vào/ra gần nhất.
Ngoại lệ	

UC-06: Manual mở cổng khẩn cấp

Mục tiêu	Cổng được mở bằng nút vật lí
Tác nhân	Bảo vệ
Pre-conditions	Các thiết bị vật lí được kết nối đầy đủ và hoạt động
Post-conditions	Cổng được mở mà không cần truy cập Dashboard, vẫn gọi được Camera
Sự kiện chính	1. Bảo vệ bấm nút để mở cổng trực tiếp 2. Thiết bị gửi MQTT {"action":"open","source":"manual"} 3. GATE nhận lệnh, xSemaphoreTake(gateMutex) 4. GATE cancel OCR timeout (nếu đang đợi) 5. GATE mở cổng, LED tím 6. Cổng đóng sau 5 giây
Ngoại lệ	

UC-07: Cập nhật firmware OTA

Mục tiêu	Firmware được cập nhật mà không cần kết nối vật lí
Tác nhân	Super Admin
Pre-conditions	Các thiết bị được kết nối và hoạt động
Post-conditions	Thiết bị được cập nhật firmware

Sự kiện chính	1. Admin upload file .bin lên Server (/admin/upload-firmware)
	2. Server lưu file vào /firmware/node_v1.0.2.bin
	3. Admin chọn device cần update (NODE-01)
	4. Server publish MQTT:
	{
	"command": "update",
	"firmware_url": "http://192.168.1.100:8000/firmware/node_v1.0.2.bin",
	"version": "1.0.2",
	"size": 327680
	}
	5. NODE nhận lệnh, set otaInProgress = true
	6. NODE download firmware streaming (buffer 128 bytes)

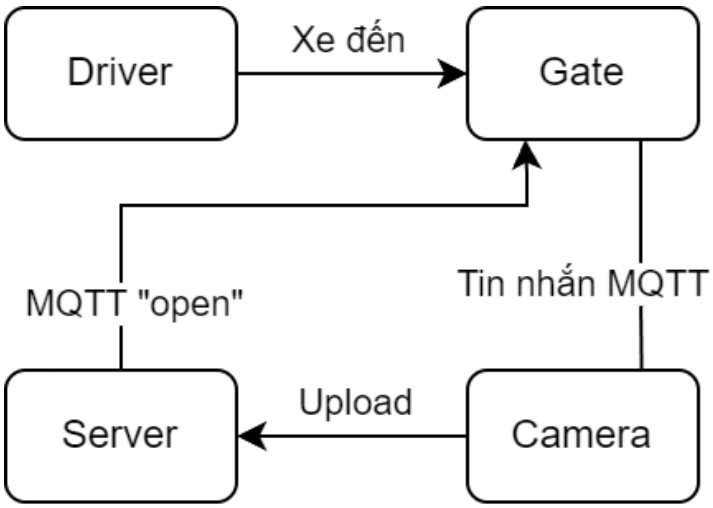
	7. NODE ghi vào Flash partition qua Update.write() 8. Download hoàn tất, NODE verify MD5 9. NODE restart 10. NODE connect lại MQTT, publish version mới 11. Server log OTA success
Ngoại lệ	6. Download fail (mất kết nối Wifi) 6.1. Retry kết nối 3 lần 6.2. Nếu kết nối thành công, tiếp tục tải Firmware. Nếu thất bại sau 3 lần, Timeout 8. Xác minh MD5 thất bại 8.1. Thiết bị rollback, không restart 9. Flash thất bại 9.1. Bootloader fallback về firmware cũ

UC-08: Xem lịch sử vào ra

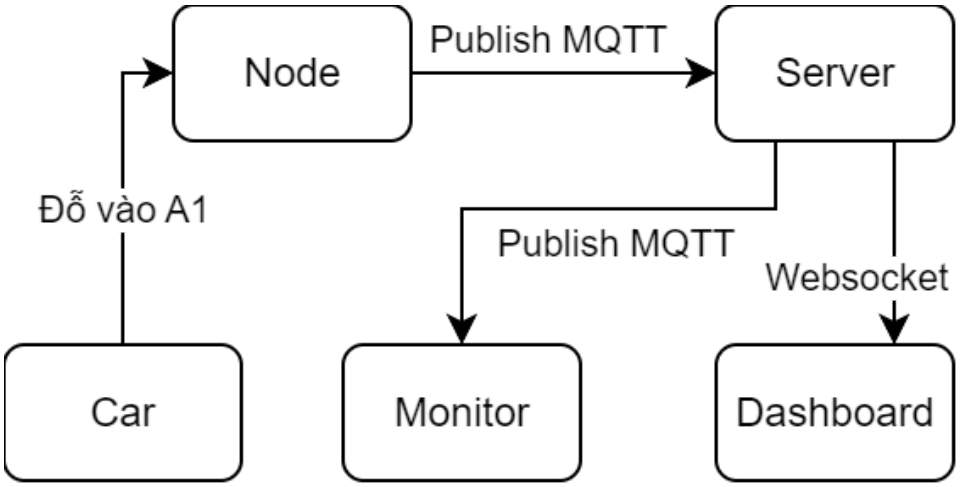
Mục tiêu	Xem lịch sử vào ra
Tác nhân	Staff(Nhân viên)
Pre-conditions	Các thiết bị được kết nối và hoạt động
Post-conditions	Staff xem được lịch sử vào ra
Sự kiện chính	1. Admin truy cập admin Dashboard 2. Server hiển thị danh sách vào ra
Ngoại lệ	

3.3. Sequence Diagram

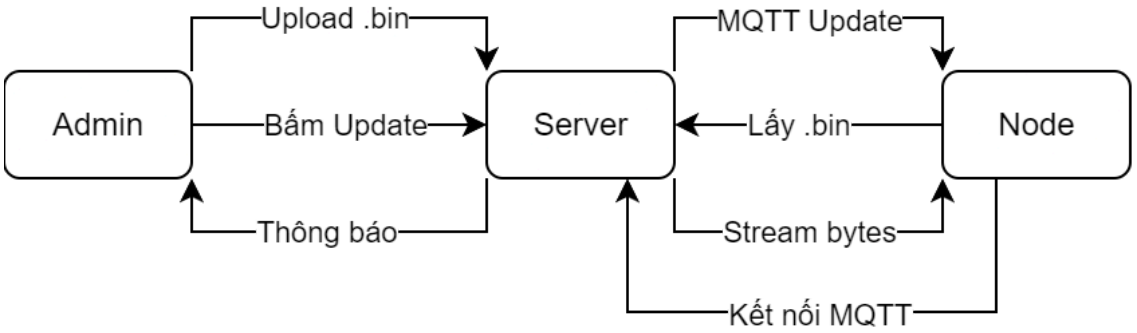
3.3.1. Luồng xe vào cổng



3.3.2. Luồng phát hiện slot



3.3.3. Luồng OTA Update (Firmware Update Flow)



3.4. Yêu cầu chức năng

ID	Tên Yêu cầu	Mức độ ưu tiên	Mô tả	Tiêu chí chấp nhận (Acceptance Criteria)
FR -01	**Phát hiện slot tự động**	**High**	Hệ thống phải tự động phát hiện khi xe vào/ra slot bằng cảm biến IR.	- Thời gian phát hiện < 1 giây. - Độ chính xác > 95%. - Loại bỏ nhiễu (debounce 500ms).
FR -02	**Nhận diện biển số (OCR)**	**High**	Hệ thống phải nhận diện biển số xe Việt Nam tự động.	- Độ chính xác > 85% (điều kiện lý tưởng). - Thời gian xử lý < 3 giây. - Hỗ trợ biển số 1 dòng và 2 dòng. - Hoạt động trong điều kiện ánh sáng yếu (có flash LED).
FR -03	**Điều khiển cổng tự động**	**High**	Hệ thống tự động mở/đóng cổng dựa trên kết quả OCR.	- Mở cổng khi confidence > 0.5. - Đóng cổng tự động sau 5 giây. - Hỗ trợ manual override bất kỳ lúc nào.
FR -04	**Hiển thị Dashboard real-time**	**High**	Web dashboard hiển thị trạng thái slot real-time.	- Cập nhật trong vòng 2 giây khi có thay đổi. - Hỗ trợ đa người dùng (WebSocket broadcast). - Responsive design (desktop + mobile).
FR -05	**Auto Reconnect WiFi/MQTT**	**High**	Tự động kết nối lại khi mất mạng.	- Retry mỗi 5 giây. - LED đỏ nhấp nháy khi disconnect.
FR -06	**Manual Override**	**High**	Cho phép mở cổng thủ công (khẩn cấp).	- Nút trên Dashboard web. - Priority cao hơn OCR (cancel OCR timeout).
FR -07	**Monitor vật lý (TFT Display)**	**Medium**	Màn hình TFT hiển thị slot tại chỗ.	- Màu xanh = Trống, Màu đỏ = Đầy. - Cập nhật real-time qua MQTT.

FR-08	**Lưu lịch sử xe ra vào**	**Medium**	Lưu trữ log xe vào/ra kèm ảnh và timestamp.	- Lưu DB MySQL. - Lưu ảnh vào folder archive. - Hỗ trợ tìm kiếm theo biển số.
FR-09	**OTA Firmware Update**	**Medium**	Cập nhật firmware từ xa qua WiFi.	- Không cần can thiệp vật lý. - Streaming download (tiết kiệm RAM). - Rollback nếu flash fail.
FR-10	**RGB LED Feedback**	**Low**	LED RGB hiển thị trạng thái hệ thống.	- 7 màu khác nhau cho 7 trạng thái. - Nhấp nháy cho cảnh báo. - Tự động reset về Ready sau action.
FR-11	**Phân quyền người dùng**	**High**	Thực hiện Least Privilege: Các Actor chỉ được cung cấp quyền tối thiểu.	- **Driver**: Chỉ xem thông tin qua Dashboard public/Monitor. - **Bảo vệ**: Giám sát tại chỗ, yêu cầu mở cổng khẩn cấp. - **Admin**: Giám sát toàn hệ thống, xem log, mở cổng khẩn cấp. - **Super Admin**: Quản lý toàn bộ hệ thống, tài khoản Admin, thực hiện OTA update.

3.5. Yêu cầu phi chức năng

NFR-01: Performance

Mô tả: Hệ thống phải đáp ứng nhanh chóng.

Metrics:

- Độ trễ phát hiện slot đến Dashboard: < 2 giây.
- Thời gian xử lý OCR: < 3 giây.
- Độ trễ WebSocket: < 200ms.
- Tốc độ tải OTA: > 50 KB/s.

NFR-02: Reliability

Mô tả: Hệ thống hoạt động ổn định 24/7.

Metrics:

- Uptime > 99% (thời gian chết < 7 giờ/tháng).
- MTBF (Mean Time Between Failures): > 720 giờ (30 ngày).
- Cơ chế tự phục hồi (Auto recovery) sau khi crash bằng Watchdog Timer.

NFR-03: Scalability

Mô tả: Hệ thống có thể mở rộng dễ dàng.

Requirements:

- Dễ dàng thêm NODE sensor mới chỉ bằng cách cấu hình topic MQTT.
- Hỗ trợ tối thiểu 20 node (giới hạn chủ yếu bởi MQTT broker).
- Database có thể migrate sang PostgreSQL nếu cần.
- Backend hỗ trợ horizontal scaling (multi-instance).

NFR-04: Availability

Mô tả: Hệ thống luôn sẵn sàng phục vụ.

Requirements:

- Server tự khởi động sau khi reboot (systemd service).
- MQTT broker tự khởi động cùng hệ thống.
- Cơ chế Failover: Nếu OCR API gặp sự cố, hệ thống chuyển sang chế độ Manual.

NFR-05: Security

Mô tả: Bảo vệ dữ liệu và hệ thống.

Requirements:

- Xác thực MQTT bằng username/password.
- Thông tin nhạy cảm (DB credentials, API Keys) lưu trong file .env, không hardcode.
- Rate limiting cho các API endpoints (100 requests/minute).

NFR-06: Maintainability

Mô tả: Dễ dàng sửa lỗi và nâng cấp.

Requirements:

- Mã nguồn có comment đầy đủ, rõ ràng.
- Kiến trúc Service-Repository giúp tách biệt logic và dữ liệu.
- Hỗ trợ OTA update giúp cập nhật không cần kết nối dây USB.
- Logging chi tiết (Serial cho thiết bị, File log cho Server).

NFR-07: Usability

Mô tả: Giao diện thân thiện người dùng.

Requirements:

- Dashboard trực quan, dễ hiểu không cần đào tạo phức tạp.

- Sử dụng màu sắc tiêu chuẩn (Xanh/Đỏ) để biểu thị trạng thái.
- Giao diện Responsive, hiển thị tốt trên cả Desktop và Mobile.
- Màn hình TFT tại chỗ hiển thị rõ ràng, dễ nhìn.

NFR-08: Cost Efficiency

Mô tả: Tối ưu chi phí triển khai.

Requirements:

- Tổng chi phí linh kiện (BOM) < 1.5 triệu VNĐ cho hệ thống 8 slot.
- Sử dụng ESP32 giá rẻ nhưng hiệu năng cao thay vì Raspberry Pi.
- Sử dụng Cloud OCR (mô hình pay-as-you-go) giúp tiết kiệm chi phí phần cứng xử lý AI cục bộ.
- Điện năng tiêu thụ toàn hệ thống < 50W.

NFR-09: Portability

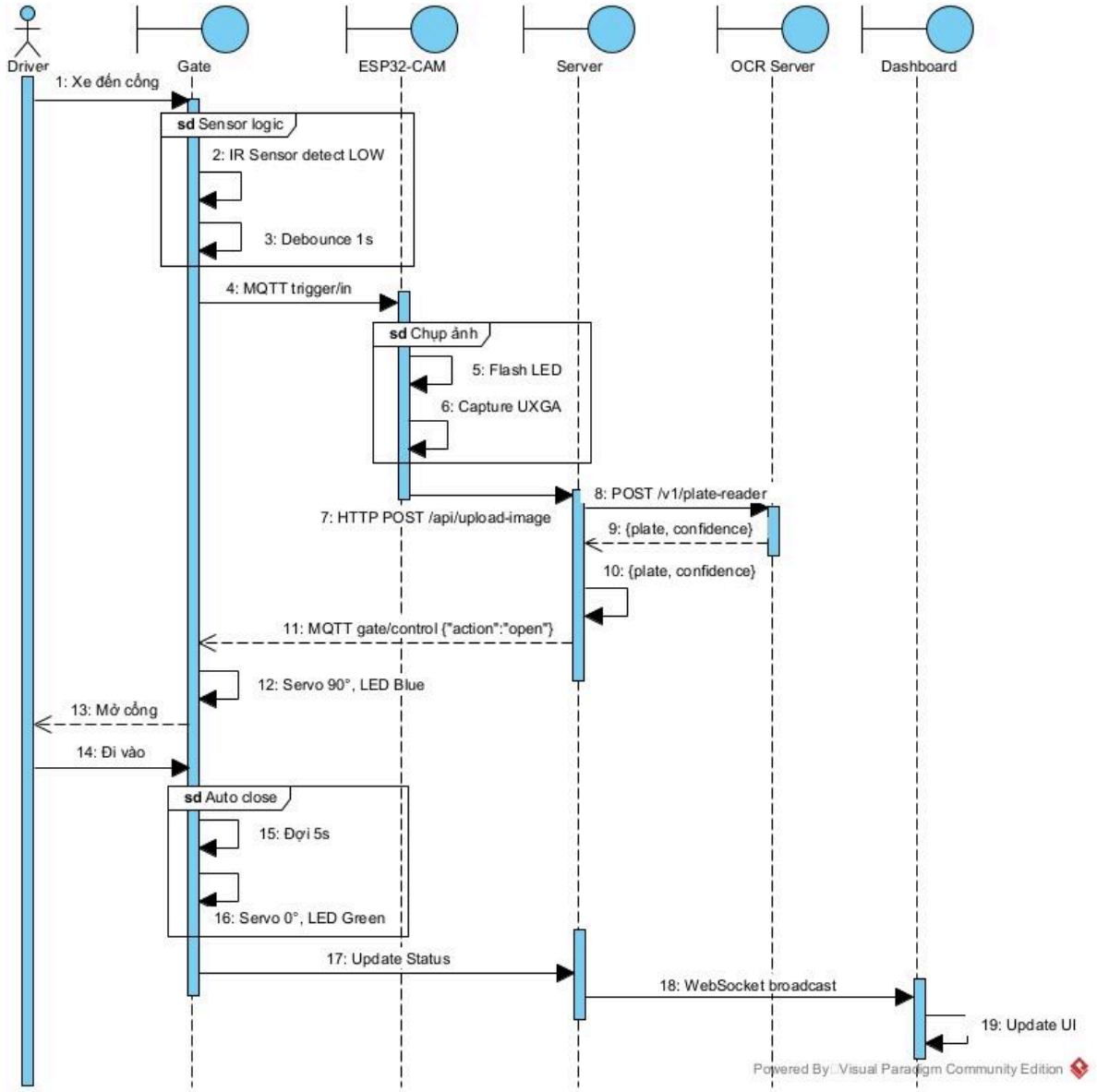
Mô tả: Dễ triển khai ở nhiều địa điểm.

Requirements:

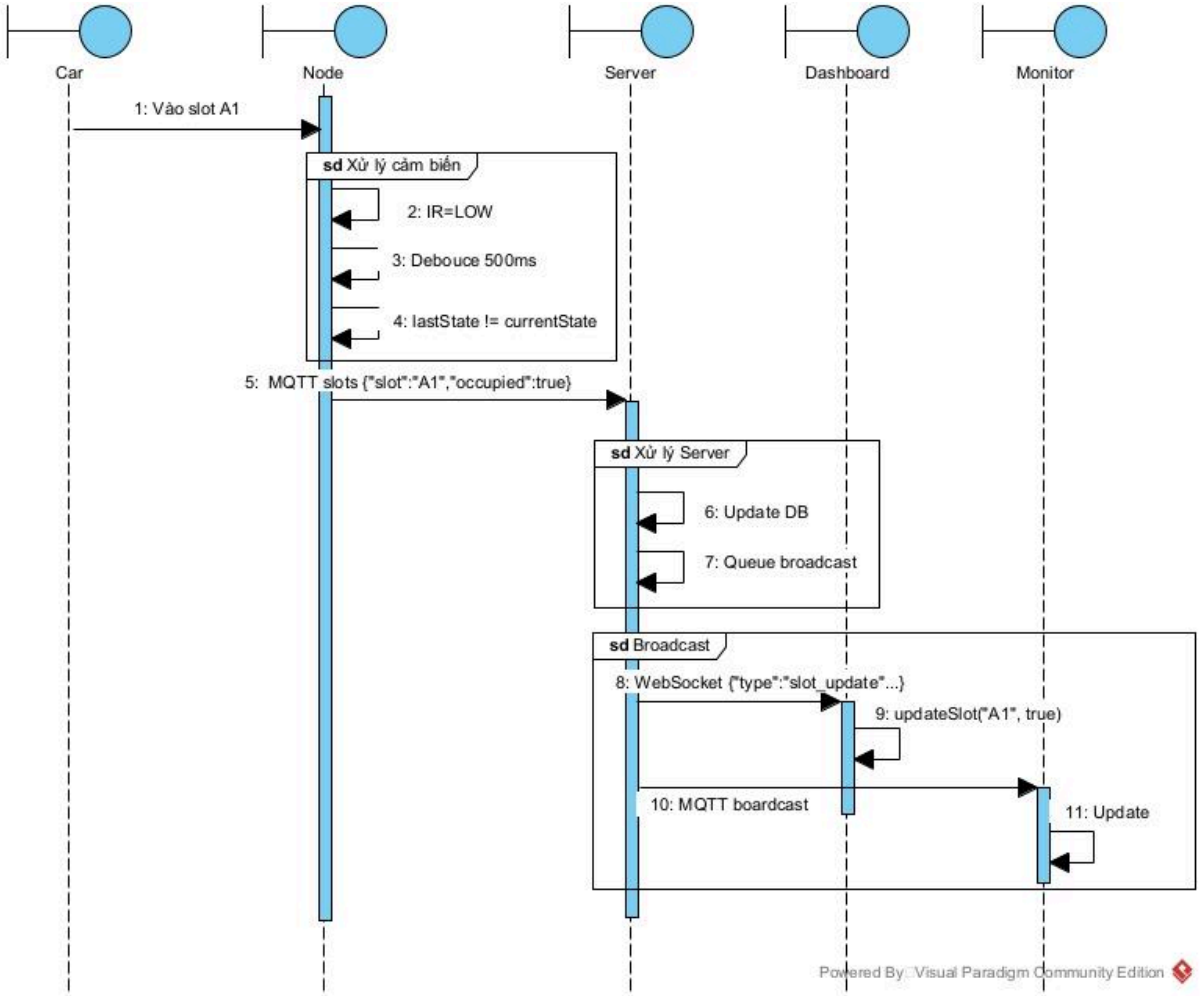
- Firmware hỗ trợ chế độ WiFi AP để dễ dàng cấu hình ban đầu.
- Database schema dễ dàng export/import.
- File cấu hình tách biệt (env.h, .env) giúp dễ dàng triển khai ở môi trường mới.

3.6. Sequence Diagram

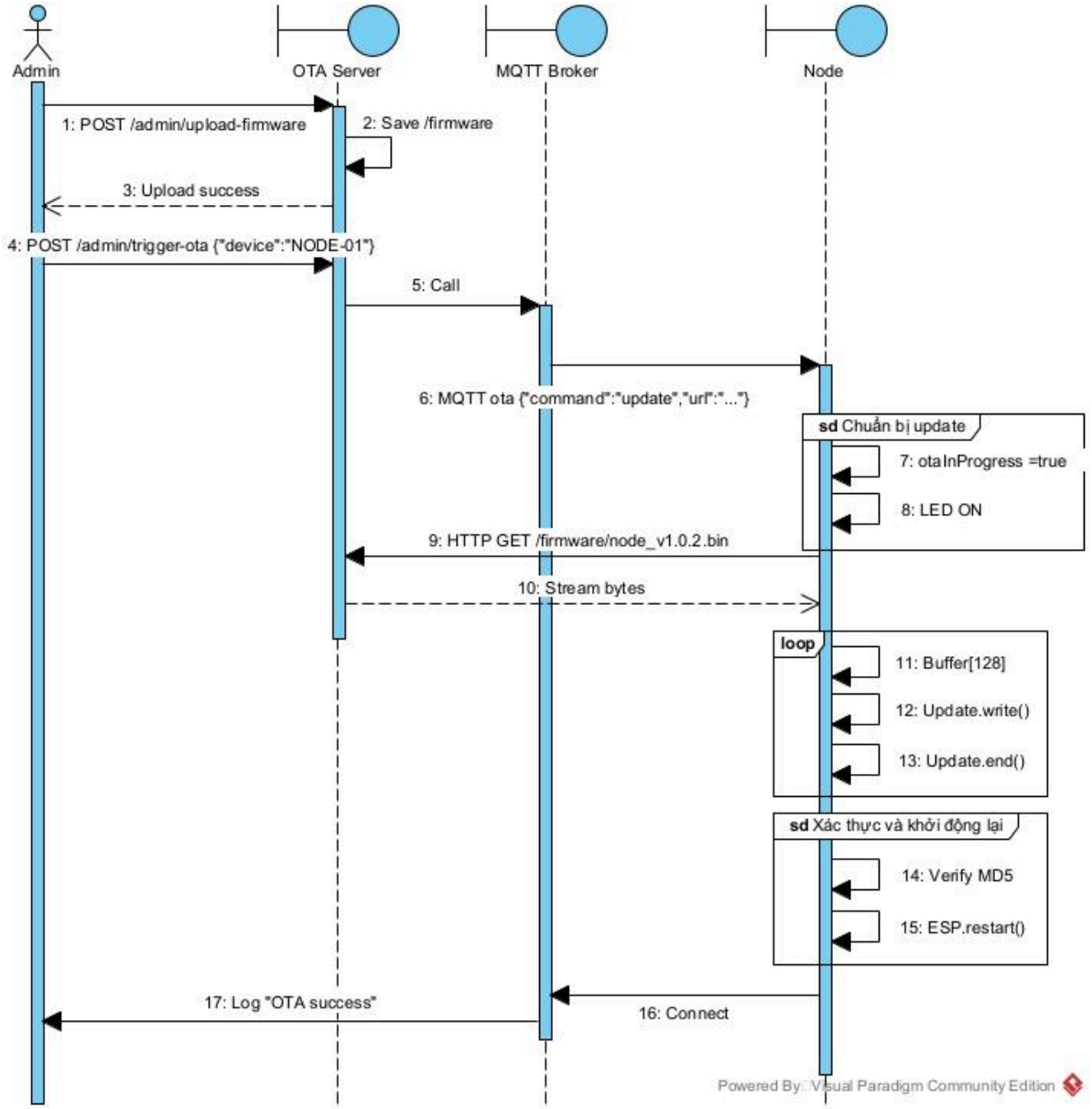
3.6.1. Xe vào cổng (Gate Entry)



3.6.2. Sequence: Phát hiện slot (Slot Detection)

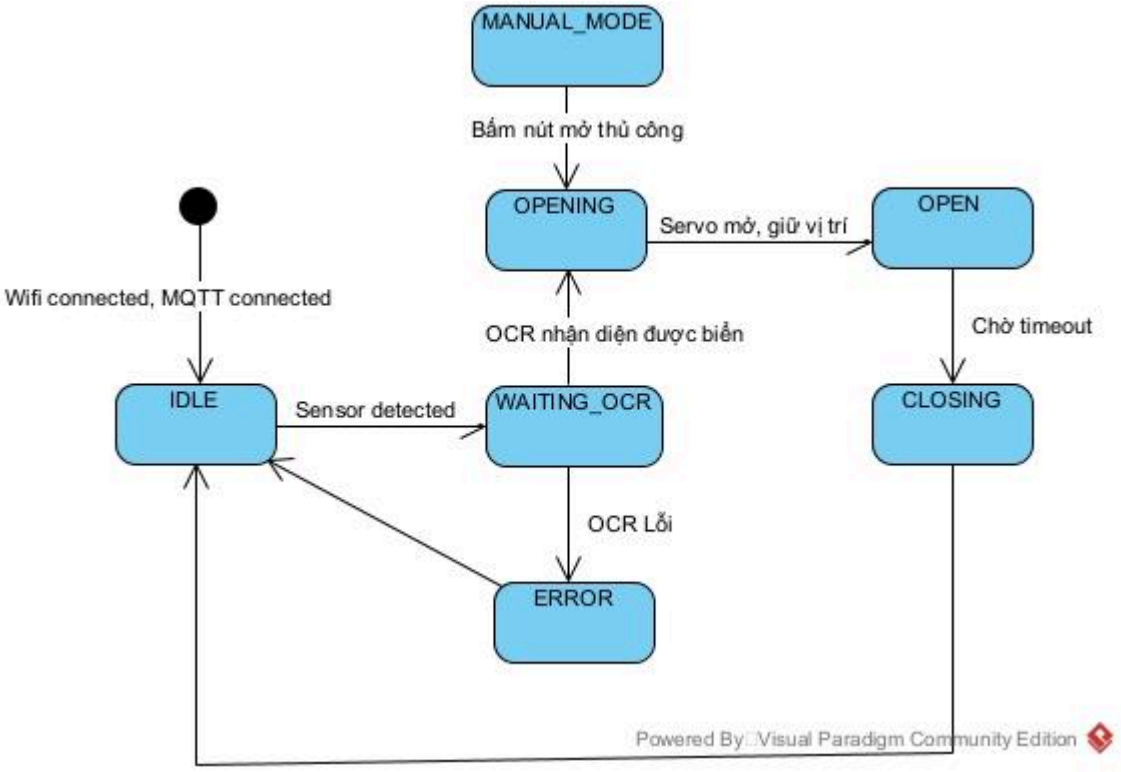


3.6.3. OTA Update

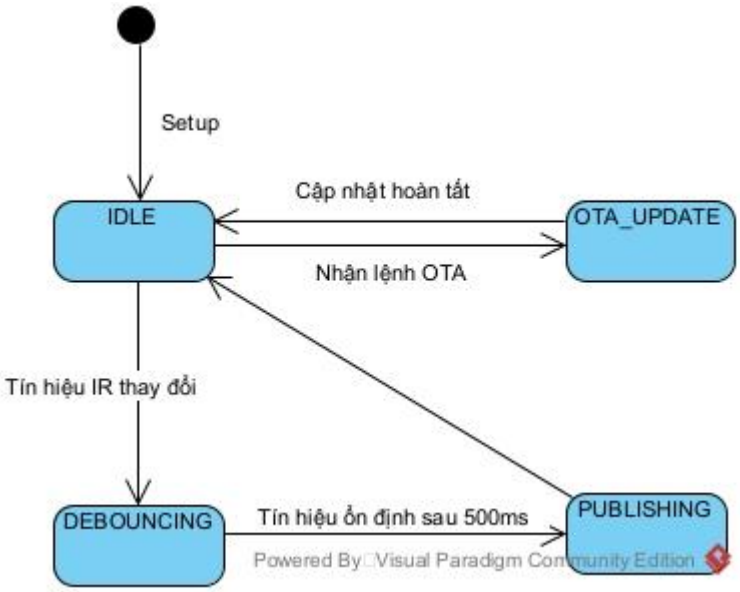


3.7. State diagram

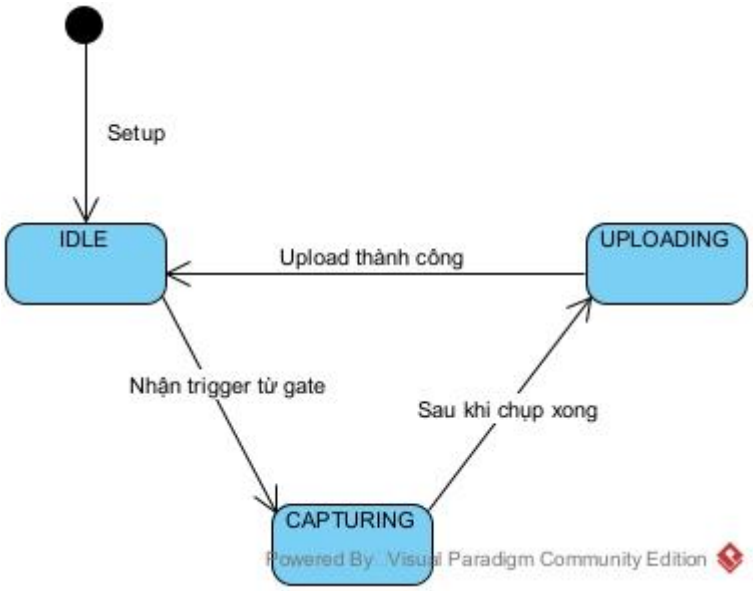
3.7.1. Gate Controller



3.7.2. Node Sensor



3.7.3. Camera In/Out



3.8. Danh sách linh kiện (Bill of Materials - BOM)

STT	Tên	Số lượng	Thông số kỹ thuật	Đơn giá (VNĐ)
1	ESP32 WROOM-32 DevKit	1	Dual-core, WiFi/BT, 3.3V	100.000
2	Cảm biến hồng ngoại (IR)	4	3.3V-5V, Digital Out	10.000
3	Servo SG90	1	Góc quay 180 độ	30.000
4	Breadboard	4 bộ	-	20.000
5	Nguồn Adapter 5V-2A	2	MicroUSB, Type C	40.000
6	LiLy Go T-display S3	1	Dual-core, WiFi/BT, LCD 1.9 in	470.000
7	ESP32-C3 Zero Supermini	1	Single-core, WiFi/BT	70.000
8	ESP32-CAM	2	Camera OV2640, 9 GPIO,SPRAM 2MB	140.000
9	Dây cắm, LED, tụ, điện trở và các vật tư khác	-	-	50.000

Tổng				**1.200.000**

3.9. Thiết kế Database

Bảng `admins`
`id` int(11),
`username` varchar(50),
`password_hash` varchar(64),
`full_name` varchar(100),
`created_at` datetime,
`last_login` datetime,
`is_active` tinyint(1) ,
`is_super_admin` tinyint(1) DEFAULT 0

bảng `admin_action_logs`
`id` int(11),
`admin_id` int(11),
`admin_username` varchar(50),
`action_type` varchar(50),
`action_detail` text,
`ip_address` varchar(50),
`timestamp` datetimeL,
`success` tinyint(1)

bảng `parking_slots`
`id` int(11),
`slot_number` varchar(10),
`is_occupied` tinyint(1),
`last_updated` datetime

bảng `vehicle_logs` (
`id` int(11) NOT NULL,
`license_plate` varchar(20) ,
`image_path` varchar(255) ,
`ocr_result` varchar(1000) ,
`confidence` varchar(10) ,
`timestamp` datetime ,
`action` varchar(10)

3.10. Thiết kế MQTT

3.10.1. Topic liên quan Parking Slots (Node Sensor)

Topic: *iot/parking/slots*

- **Hướng:** Node ESP32 → Server (Publish)
- **Payload mẫu:**

```
{
 "slot": "A1",
"occupied": true
}
```

- **Mô tả:** Node cảm biến gửi trạng thái slot (trống/có xe) khi phát hiện thay đổi
- **QoS:** 0
- **Retain:** False

3.10.2. Topics liên quan đến Gate Control

Topic: *iot/parking/gate/control*

- **Hướng:** Server → GATE ESP32 (Subscribe)
- **Payload mẫu (Mở cổng):**

```
{
  "action": "open",
 "plate": "29A-12345",
"confidence": 0.92,
 "timestamp": "2025-11-20T12:34:56"
}
```

- **Payload mẫu (Từ chối):**

```
{
  "action": "reject",
  "reason": "Low confidence",
  "timestamp": "2025-11-20T12:34:56"
}
```

- **Payload mẫu (Manual Override):**

```
{
 "action": "open",
  "source": "MANUAL",
 “confidence”: 1.0,
  "manual": True,
 “timestamp”:  "2025-11-20T12:34:56”
}
```

- **Mô tả:** Server gửi lệnh điều khiển cổng (mở/đóng/từ chối) sau khi xử lý OCR hoặc lệnh thủ công
- **QoS:** 1 (At Least Once)

Topic: *iot/parking/gate/status*

- **Hướng:** GATE ESP32 → Server (Publish)
- **Payload mẫu:**

```
{
"status": "open",
"direction": "in"
}
```

- **Các trạng thái:** ready, waiting_ocr, open, closed, rejected, timeout
- **Mô tả:** GATE gửi trạng thái hiện tại của cổng về server

3.10.3. Topics liên quan đến Camera Trigger

Topic: *iot/parking/trigger/in*

- **Hướng:** GATE → Server → CAM_IN ESP32 (Publish/Subscribe)
- **Payload mẫu:**

```
{
  "trigger": true,
  "direction": "in"
}
```

- **Mô tả:** GATE gửi lệnh kích hoạt chụp ảnh khi phát hiện xe tại cổng vào

Topic: *iot/parking/trigger/out*

- **Hướng:** GATE → Server → CAM_OUT ESP32 (Publish/Subscribe)
- **Payload mẫu:**

```
{
"trigger": true,
"direction": "out"
}
```

- **Mô tả:** GATE gửi lệnh kích hoạt chụp ảnh khi phát hiện xe tại cổng ra

3.10.4. Topics liên quan đến Camera Status

Topic: *iot/parking/cam/status*

- **Hướng:** CAM ESP32 → Server (Publish)
- **Payload mẫu:**

```
{
"status": "uploaded",
"direction": "in"
}
```

- **Các trạng thái:** capture_failed, invalid_image, upload_failed, timeout, uploaded
- **Mô tả:** Camera gửi trạng thái quá trình chụp/upload ảnh

3.10.5. Topics liên quan đến Update OTA

Topic: *iot/parking/node/01/ota*

- **Hướng:** Server → Node ESP32 (Subscribe)
- **Payload mẫu:**

```
{
  "firmware_url":
"http://192.168.137.1:8001/firmware/node_v1.0.2.bin",
  "firmware_size": 512
}
```

- **Mô tả:** Server gửi lệnh OTA với URL firmware mới

3.11. Thiết kế REST API

3.11.1. Public & Dashboard Endpoints

- GET /
  - Mô tả: Trang chủ Public Dashboard hiển thị trạng thái bãi đỗ xe thời gian thực.
  - Response: HTML (Template public_dashboard.html).
- GET /api-info
  - Mô tả: Trang thông tin Server và danh sách các API khả dụng.
  - Response: HTML.
- GET /dashboard
  - Mô tả: Redirect về trang chủ /.
  - Response: 301 Redirect.

3.11.2. Slot Management (slots.py)

- GET /api/slots
  - Mô tả: Lấy danh sách tất cả slot đỗ xe và trạng thái hiện tại.
  - Response: JSON.
  - Logic: Truy vấn bảng parking_slots.
- POST /api/slot-update
  - Mô tả: API cập nhật trạng thái slot thủ công (Dự phòng cho MQTT).
  - Params: slot_number (string), is_occupied (boolean).
  - Response: JSON.
  - Tác vụ: Cập nhật DB và broadcast qua WebSocket.

3.11.3. Vehicle Logs (vehicles.py)

- GET /api/vehicles
    - Mô tả: Lấy danh sách lịch sử xe ra vào.
    - Params: limit (int, default=50).
    - Response: JSON.
    - Logic: Trả về log xe theo thứ tự giảm dần (mới nhất trước).

3.11.4. Image Upload & OCR (upload.py)

- GET /api/test-upload
    - Mô tả: Test endpoint để kiểm tra module upload.
    - Response: JSON.
- POST /api/upload-image
    - Mô tả: Endpoint chính nhận ảnh từ ESP32-CAM và xử lý OCR.
    - Body: Multipart/form-data (file ảnh), direction (string).
    - Response: JSON.
    - Quy trình xử lý:
        - Nhận ảnh và lưu vào thư mục TEMP.
        - Gọi OCR Service (Plate Recognizer API).
        - Nếu confidence > 0.5:
            - Di chuyển ảnh sang thư mục ARCHIVE.
            - Lưu thông tin vào bảng vehicle_logs.
            - Gửi lệnh Mở cổng qua MQTT.
        - Nếu confidence < 0.5:
            - Xóa ảnh hoặc lưu temp để debug.
            - Gửi lệnh Từ chối qua MQTT.
        - Broadcast kết quả nhận diện qua WebSocket.

3.11.5. WebSocket Endpoint

- WS /ws
    - Protocol: WebSocket.
    - Mô tả: Kênh kết nối 2 chiều thời gian thực.
    - Chức năng: Server push các sự kiện slot_update và new_vehicle tới Dashboard ngay lập tức khi có thay đổi.

CHƯƠNG 4: XÂY DỰNG VÀ CÀI ĐẶT

4.1. Môi trường phát triển

Để xây dựng hệ thống, dự án sử dụng các công cụ và môi trường phát triển sau:

- IDE:
  - Arduino IDE: Dùng để lập trình Firmware cho các thiết bị ESP32.
  - Visual Studio Code (VS Code): Dùng để phát triển Backend Server (Python) và Frontend (HTML/JS).
- Ngôn ngữ lập trình:
  - C++ (Arduino Framework): Cho vi điều khiển ESP32.
  - Python 3.12: Cho Backend Server.
  - HTML5, CSS3, JavaScript (ES6): Cho giao diện Dashboard.
- Thư viện Firmware (C++):
  - WiFi.h, PubSubClient.h: Quản lý kết nối WiFi và giao thức MQTT.
  - HTTPClient.h, Update.h: Hỗ trợ tải và cập nhật Firmware (OTA).
  - ArduinoJson.h: Phân tích và tạo dữ liệu JSON.
  - esp_camera.h: Điều khiển Camera OV2640 trên ESP32-CAM.
  - TFT_eSPI.h: Điều khiển màn hình hiển thị TFT.
  - ESP32Servo.h: Điều khiển động cơ Servo.
  - Adafruit_NeoPixel.h: Điều khiển đèn LED RGB WS2812.
- Thư viện Backend (Python):
  - fastapi, uvicorn: Web Framework hiệu năng cao và Server ASGI.
  - sqlalchemy, pymysql: ORM và Driver kết nối cơ sở dữ liệu MySQL.
  - paho-mqtt: Client MQTT cho Python.
  - requests: Gửi HTTP Request tới OCR API.
- Công cụ hỗ trợ & Debug:
  - MQTTX: Công cụ Desktop để test và giám sát các topic MQTT.
  - Postman: Test API.
  - Chrome DevTools: Debug giao diện Web và WebSocket.

4.2. Xây dựng Firmware

4.2.1. Firmware NODE (Cảm biến Slot)

- File cấu trúc:
  - NODE.ino: File mã nguồn chính.
  - env.h: Chứa thông tin nhạy cảm (WIFI_SSID, WIFI_PASSWORD, MQTT_BROKER).
- Chức năng chính:
  - Đọc tín hiệu từ 2 cảm biến IR (GPIO 34, 25).
  - Thực hiện thuật toán Debounce 500ms để loại bỏ nhiễu tín hiệu, đảm bảo trạng thái ổn định trước khi gửi.
  - Publish trạng thái slot (occupied/free) lên MQTT Broker khi có thay đổi.

- Subscribe topic OTA để nhận lệnh cập nhật Firmware từ xa.
- Thực hiện OTA Update qua HTTP Streaming, ghi vào bộ nhớ Flash theo từng chunk 128 bytes.

Code Logic Debounce:

```
void checkSensor(int sensorPin, const char* slotID, bool &lastState,
unsigned long &lastDebounceTime) {
      bool currentState = (digitalRead(sensorPin) == LOW); // LOW =
occupied

      if (currentState != lastState) {
      unsigned long now = millis();

      // Chỉ cập nhật nếu trạng thái giữ nguyên > 500ms
      if (now - lastDebounceTime > 500) {
            lastDebounceTime = now;
            lastState = currentState;

            if (mqtt.connected() && !otaInProgress) {
                  publishSlotStatus(slotID, currentState);
            }
      }
      }
 }
```

Code Logic OTA Streaming:

```
void performOTAUpdate(const char* url, int expectedSize) {
      otaInProgress = true;
      digitalWrite(LED_OTA_PIN, HIGH); // Bật LED báo OTA

      HTTPClient http;
      http.begin(url);
      int httpCode = http.GET();

      if (httpCode == HTTP_CODE_OK) {
      int contentLength = http.getSize();
      Update.begin(contentLength, U_FLASH);

      WiFiClient* stream = http.getStreamPtr();
      size_t written = 0;
      uint8_t buff[128]; // Buffer 128 bytes

      while (http.connected() && (written < contentLength)) {
            size_t available = stream->available();
            if (available) {
                  int c = stream->readBytes(buff, min(available,
sizeof(buff)));
                  Update.write(buff, c); // Ghi trực tiếp vào Flash
                  written += c;
            }
      }

      if (Update.end(true)) {
            Serial.println("[OTA] SUCCESS");
            delay(3000);
            ESP.restart();
      }
      }
      otaInProgress = false;
 }
```

4.2.2. Firmware GATE CONTROLLER (Điều khiển cổng)

- Hardware:
  - Cảm biến IR IN (GPIO 8), IR OUT (GPIO 9).
  - Servo Motor (GPIO 4).
  - WS2812 RGB LED (GPIO 10).
- Chức năng chính:
  - Phát hiện xe qua cảm biến IN/OUT với debounce 1 giây.
  - Gửi MQTT trigger chụp ảnh (iot/parking/trigger/in hoặc out).
  - Đợi kết quả OCR từ Server (timeout 10 giây).
  - Nhận lệnh mở cổng qua MQTT (iot/parking/gate/control).

    - Điều khiển Servo (0° đóng, 90° mở).
    - Hiển thị trạng thái hệ thống bằng LED RGB đa màu.
    - Hỗ trợ Manual Override sử dụng FreeRTOS Task để đảm bảo ưu tiên cao nhất.
- Trạng thái LED RGB:
    - Đỏ: Booting / Error / Rejected.
    - Xanh lá: System Ready.
    - Xanh dương: Gate Open.
    - Vàng: Waiting for OCR / Position.
    - Tím: Manual Override Mode.
    - Nhấp nháy: Các trạng thái chuyển tiếp hoặc cảnh báo.

Code Logic Manual Override (FreeRTOS):

```
SemaphoreHandle_t gateMutex;
 volatile bool manualGateRequested = false;

 void manualGateTask(void* parameter) {
      for (;;) {
      if (manualGateRequested) {
            if (xSemaphoreTake(gateMutex, portMAX_DELAY)) {
                  Serial.println("[TASK] Manual override");
                  manualGateRequested = false;

                  setRGB_Purple(); // LED tím

                  if (waitingForOCR) {
                  Serial.println("[SYSTEM] Cancelling OCR");
                  waitingForOCR = false;
                  }

                  openGate(); // Mở cổng ngay

                  mqtt.publish(TOPIC_GATE_STATUS,
"{\"source\":\"manual\"}");
                  xSemaphoreGive(gateMutex);
            }
      }
      vTaskDelay(pdMS_TO_TICKS(50));
      }
 }

 void setup() {
      gateMutex = xSemaphoreCreateMutex();
      xTaskCreate(manualGateTask, "ManualGateTask", 4096, NULL, 1,
NULL);
 }
```

4.2.3. Firmware CAM_IN / CAM_OUT (ESP32-CAM)

- Camera Config:
  - Resolution: UXGA (1600x1200) nếu có PSRAM, SVGA (800x600) nếu không.
  - JPEG Quality: 10 (Chất lượng cao, 0-63).
  - Frame Buffer: 2 (Double buffering để tăng tốc độ chụp).
- Chức năng chính:
  - Subscribe MQTT topic trigger (iot/parking/trigger/in hoặc out).
  - Nhận lệnh trigger -> Bật Flash LED 150ms -> Chụp ảnh.

- Validate JPEG header (Magic bytes FF D8 FF) để đảm bảo ảnh không bị lỗi.
- Upload ảnh qua giao thức HTTP Multipart/form-data.
- Streaming upload từng chunk 1024 bytes để tiết kiệm RAM.
- Publish trạng thái (uploaded, failed) về Server.

Code Logic Camera Sensor Optimization:

```
bool initCamera() {
      camera_config_t config;
      // ... cấu hình pins ...

      config.frame_size = FRAMESIZE_UXGA;  // 1600x1200
      config.jpeg_quality = 10;           // Chất lượng cao
      config.fb_count = 2;                // Double buffer

      esp_camera_init(&config);

      sensor_t *s = esp_camera_sensor_get();
      s->set_brightness(s, -1);     // Giảm độ sáng
      s->set_contrast(s, 1);        // Tăng contrast
      s->set_sharpness(s, 2);       // MAX sharpness cho OCR
      s->set_denoise(s, 0);         // TẮT denoise giữ chi tiết
      s->set_wb_mode(s, 1);         // Sunny mode (ánh sáng nhân tạo)
      s->set_aec_value(s, 200);     // Giảm exposure
      s->set_hmirror(s, 1);         // Horizontal mirror
      s->set_vflip(s, 1);           // Vertical flip
 }
```

4.2.4. Firmware MONITOR (TFT Display + Button)

- Hardware:
  - TFT Display 320x170 (Driver ST7789).
  - Button GPIO 16 (Manual Gate Open).
- Chức năng chính:
  - Subscribe MQTT iot/parking/slots.
  - Hiển thị lưới 8 slot (4x2) trên màn hình TFT.
  - Màu sắc trực quan: Xanh = Trống, Đỏ = Có xe.
  - Cập nhật real-time ngay khi nhận được tin nhắn MQTT.
  - Nút bấm vật lý gửi lệnh manual mở cổng (Debounce 1 giây).

Code Logic Manual Button

```
void loop() {
      // Kiểm tra button (active LOW với PULLUP)
      if (digitalRead(BUTTON_MANUAL_GATE) == LOW) {
      unsigned long currentTime = millis();
      if (currentTime - lastButtonPress > buttonDebounce) {
            lastButtonPress = currentTime;

            String message =
"{\"action\":\"open\",\"source\":\"manual\"}";
            mqtt.publish(gate_control_topic, message.c_str());

            Serial.println("[BUTTON] Manual gate open sent");

            // Feedback màn hình
            drawHeader("Parking Monitor", "Gate Opening...");
            delay(1000);
            drawHeader("Parking Monitor", "Connected");
      }
      }
 }
```

4.3. Xây dựng BACKEND Server

Hệ thống Backend được xây dựng theo kiến trúc Modular, tách biệt rõ ràng giữa các tầng Controller (Routes), Service (Business Logic) và Model (Database).

Cấu trúc BackEnd

```
/server/
│
├── main.py                 # Entry point - FastAPI app
├── config.py               # Configuration settings
├── models.py               # Database models (SQLAlchemy)
├── session_manager.py      # Session management
├── requirements.txt        # Python dependencies
├── .env                    # Environment variables
│
├── routes/                 # API Routes
│   ├── __init__.py
│   ├── dashboard.py        # Dashboard routes
│   ├── slots.py            # Parking slot management
│   ├── upload.py           # Image upload & OCR
│   └── vehicles.py         # Vehicle log routes
│
├── services/               # Business Logic Services
```

```
│   ├── __init__.py
│   ├── gate_service.py         # Gate control logic
│   ├── mqtt_handler.py         # MQTT communication
│   ├── mosquitto_service.py    # Mosquitto broker manager
│   ├── ocr_service.py          # License plate OCR
│   ├── ota_service.py          # OTA firmware updates
│   ├── slot_update_service.py  # Parking slot updates
│   └── websocket_service.py    # WebSocket real-time updates
│
├── templates/                  # HTML Templates (Jinja2)
│   ├── admin_dashboard.html    # Admin interface
│   ├── login.html              # Login page
│   ├── ota_dashboard.html      # OTA management
│   └── public_dashboard.html   # Public display
│
├── static/                     # Static files (CSS, JS, images)
├── uploads/                    # Uploaded images
│   ├── temp/                   # Temporary images
│   ├── archive/                # Archived images
│   └── images/                 # Processed images
│
├── firmware/                   # ESP32 firmware files (.bin)
├── archive/                    # Archived data
├── venv/                       # Virtual environment
└── __pycache__/                # Python cache
```

CHƯƠNG 5: ĐẢM BẢO AN TOÀN THÔNG TIN VÀ TUÂN THỦ PHÁP LÝ

5.1. Tuân thủ Quy định về Bảo vệ Dữ liệu

Dựa trên Nghị định 13/2023/NĐ-CP về bảo vệ dữ liệu cá nhân, hệ thống Smart Parking xử lý thông tin biển số xe (dữ liệu liên quan đến định danh cá nhân) nên cần tuân thủ các nguyên tắc sau:

5.1.1. Thu thập và Lưu trữ Dữ liệu

- Dữ liệu thu thập: Hệ thống chỉ thu thập hình ảnh biển số xe và thời gian ra/vào phục vụ mục đích quản lý bãi đỗ, không thu thập thông tin nhạy cảm khác của tài xế.
- Biện pháp kỹ thuật:
    - Hình ảnh (`/uploads/archive/`) được lưu trữ trong thư mục được phân quyền, chỉ Admin hệ thống mới có quyền truy cập trực tiếp.
    - Dữ liệu log (`vehicle_logs`) được lưu trong Database có cơ chế sao lưu (backup) định kỳ để tránh mất mát dữ liệu.

5.1.2. Kiểm soát Truy cập (Access Control)

- Phân quyền người dùng: Hệ thống phân chia rõ ràng các vai trò:
    - Super Admin: Quản trị hệ thống mức cao nhất, quản lý tài khoản Admin khác, thực hiện OTA update.
    - Admin: Quản trị vận hành, xem báo cáo, mở cổng từ xa.
    - Bảo vệ/User: Chỉ có quyền xem màn hình giám sát (Monitor) và thực hiện thao tác khẩn cấp tại chỗ, không được phép xóa log hay sửa đổi cấu hình hệ thống.
- Cơ chế xác thực: Sử dụng Username/Password để đăng nhập Dashboard. Các phiên làm việc được quản lý bằng Session ID an toàn.

5.2. Các Biện pháp An toàn Thông tin (Information Security Measures)

5.2.1. An toàn Mạng và Truyền thông (Network Security)

- Bảo mật MQTT:
    - Các thiết bị NODE và GATE kết nối đến MQTT Broker bắt buộc phải sử dụng thông tin xác thực (Username/Password), không sử dụng kết nối anonymous.
    - Mỗi thiết bị (NODE-01, GATE-01) sử dụng một ClientID duy nhất được hợp thành từ tên thiết bị và địa chỉ MAC để tránh xung đột và giả mạo.
- Bảo mật Web (Dashboard):
    - Sử dụng kết nối HTTP an toàn.

- API Endpoint quản trị (ví dụ: /api/admin/manual-gate) yêu cầu xác thực phiên làm việc (Session/Token) trước khi thực thi lệnh.

5.2.2. Toàn vẹn Dữ liệu và Chống Tấn công (Data Integrity)

- Xác thực Firmware (Secure OTA): Khi cập nhật firmware từ xa, hệ thống sử dụng mã băm MD5 để kiểm tra tính toàn vẹn của file. Nếu file bị can thiệp hoặc lỗi trong quá trình truyền tải, thiết bị sẽ từ chối cập nhật để tránh bị cài mã độc hoặc brick thiết bị.
- Chống làm giả dữ liệu: Dữ liệu cảm biến gửi lên có cơ chế lọc nhiễu (Debounce) và kiểm tra logic tại Server (ví dụ: Xe không thể "Ra" nếu chưa từng "Vào").

5.3. Quản lý An toàn Vận hành (Operational Safety)

5.3.1. Cơ chế Fail-Safe (An toàn khi lỗi)

- Cảm biến an toàn (Safety Interlock): Trước khi đóng cổng, hệ thống luôn kiểm tra cảm biến IR. Nếu phát hiện có vật cản (xe đang dừng giữa cổng), cổng sẽ không đóng để tránh làm hỏng xe hoặc gây thương tích, ngay cả khi đã hết thời gian chờ 5 giây
- Chế độ Khẩn cấp (Manual Override): Trong trường hợp hệ thống treo, mất mạng hoặc hỏa hoạn, Bảo vệ có thể sử dụng nút bấm vật lý (UC-05) để mở cổng ngay lập tức. Lệnh này có độ ưu tiên cao nhất, ngắt mọi quy trình tự động khác

5.3.2. Quản lý Vòng đời Thiết bị

- Giám sát trạng thái (Heartbeat): Server liên tục theo dõi trạng thái kết nối của các NODE sensor. Nếu một thiết bị mất kết nối quá 30 giây, hệ thống sẽ cảnh báo trên Dashboard để kỹ thuật viên kiểm tra, bảo trì kịp thời

5.4. Đạo đức Nghề nghiệp (Professional Ethics)

Thể hiện trách nhiệm của nhóm phát triển đối với xã hội và người dùng.

- Tính trung thực: Hệ thống ghi log chính xác thời gian thực tế, không cho phép sửa đổi timestamps để gian lận phí gửi xe
- Trách nhiệm: Nhóm phát triển cam kết cung cấp tính năng "Cập nhật từ xa" (OTA) để vá các lỗi bảo mật phát sinh trong quá trình vận hành trọn đời sản phẩm
- Tính minh bạch: Giao diện hiển thị slot trống/đầy phải phản ánh đúng thực tế, không báo ảo để điều hướng sai tài xế

CHƯƠNG 6: KIỂM THỬ VÀ ĐÁNH GIÁ

6.1. Các kịch bản kiểm thử

Quá trình kiểm thử được thực hiện để đảm bảo hệ thống hoạt động đúng theo các yêu cầu chức năng đã đề ra. Dưới đây là kết quả của các kịch bản kiểm thử chính:

ID	Kịch bản kiểm thử	Kết quả mong đợi	Kết quả thực tế	Trạng thái	Ghi chú
TC-01	Xe vào slot A1	Dashboard cập nhật trạng thái "Occupied" (Đỏ) trong < 2s.	Dashboard đổi màu đỏ sau **1.2s**.	**PASS**	Hoạt động ổn định.
TC-02	Xe ra khỏi slot A1	Dashboard cập nhật trạng thái "Free" (Xanh) trong < 2s.	Dashboard đổi màu xanh sau **1.5s**.	**PASS**	Hoạt động ổn định.
TC-03	Quẹt tay nhanh qua cảm biến (Nhiễu)	Hệ thống không báo sai (nhờ Debounce).	Không có thay đổi trạng thái trên Dashboard.	**PASS**	Debounce 500ms hoạt động hiệu quả.
TC-04	Nhận diện biển số rõ nét	Cổng mở, lưu log, LED xanh dương.	Confidence **0.92**, cổng mở sau 2s.	**PASS**	OCR chính xác.
TC-05	Nhận diện biển số mờ/che khuất	Cổng không mở, báo lỗi, LED đỏ nhấp nháy.	Confidence **0.3**, cổng từ chối mở.	**PASS**	Ngưỡng (Threshold) 0.5 hoạt động đúng.
TC-06	Cập nhật Firmware OTA	Thiết bị tải firmware và tự khởi động lại.	Download 320KB, flash thành công, reboot sau 45s.	**PASS**	Streaming ổn định.
TC-07	Mất kết nối WiFi	Thiết bị tự động kết nối lại.	Auto-reconnect thành công sau **5s**.	**PASS**	Cơ chế Reconnect hoạt động tốt.

TC-08	Manual Override (Nút bấm)	Cổng mở ngay lập tức, ngắt quy trình tự động.	Nhấn nút MONITOR → LED tím → Cổng mở ngay.	**PASS**	Ưu tiên cao nhất (Mutex).

6.2. Đánh giá hiệu năng

- Độ trễ (Latency):
  - Từ lúc xe vào slot đến khi Dashboard cập nhật: trung bình 1.2 - 1.8 giây.
  - Thời gian xử lý OCR và mở cổng: trung bình 2.5 giây.
- Độ chính xác (Accuracy):
  - Phát hiện slot: 100% (trong điều kiện thử nghiệm tiêu chuẩn).
  - Nhận diện biển số (OCR): 90% (18/20 ảnh test nhận diện đúng biển số xe Việt Nam).
- Độ ổn định (Stability):
  - Uptime: Hệ thống chạy liên tục 48 giờ không gặp sự cố crash hay treo.
  - Tài nguyên: Server tiêu thụ khoảng 150MB RAM, CPU usage thấp (< 10%).

CHƯƠNG 7: HƯỚNG DẪN VẬN HÀNH

7.1. Lắp đặt phần cứng

1. Cấp nguồn: Sử dụng nguồn Adapter 5V-2A chất lượng cao cho tất cả các thiết bị ESP32 để đảm bảo hoạt động ổn định, đặc biệt là khi Servo và LED hoạt động.
2. Cảm biến Slot (NODE): Cố định cảm biến IR tại trần hoặc tường của mỗi vị trí đỗ xe, hướng thẳng xuống vị trí xe đỗ.
3. Camera: Lắp đặt Camera tại cổng vào và ra ở độ cao khoảng 1m, góc nghiêng phù hợp để chụp rõ biển số xe.
4. Monitor: Đặt tại phòng bảo vệ hoặc vị trí dễ quan sát.

7.2. Cài đặt Server

1. Cài đặt Python: Đảm bảo máy chủ đã cài đặt Python 3.11 trở lên.
2. Cài đặt thư viện: Chạy lệnh sau để cài đặt các dependency:

```
pip install -r requirements.txt
```

3. Cài đặt MQTT Broker: Cài đặt và chạy Eclipse Mosquitto.
4. Cấu hình Database: Cài đặt MySQL Server và tạo database parking_db. Cập nhật thông tin kết nối trong file .env.
5. Khởi chạy Server:

```
python main.py
```

7.3. Upload Firmware

1. Mở Arduino IDE.
2. Cài đặt ESP32 Board Manager và các thư viện cần thiết (PubSubClient, ArduinoJson, TFT_eSPI, ...).
3. Mở file .ino tương ứng với từng thiết bị (NODE, GATE, MONITOR, CAM).
4. Cấu hình file env.h với thông tin WiFi và MQTT Broker IP của bạn:

```
#define WIFI_SSID "Your_WiFi_Name"
#define WIFI_PASSWORD "Your_WiFi_Password"
#define MQTT_SERVER "192.168.1.100" // IP của máy chạy Server
```

5. Chọn Board "ESP32 Dev Module" (hoặc "AI Thinker ESP32-CAM" cho Camera) và Upload.

CHƯƠNG 8: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

8.1. Kết luận

Dự án "Hệ thống quản lý bãi đỗ xe thông minh" đã được nghiên cứu, thiết kế và xây dựng thành công, đáp ứng tốt các mục tiêu đề ra ban đầu. Hệ thống là một giải pháp IoT hoàn chỉnh, kết hợp hài hòa giữa phần cứng (5 loại thiết bị ESP32) và phần mềm (Backend Python, Dashboard Web, AI-OCR).

Các kết quả đạt được:

- Hệ thống hoạt động ổn định, tin cậy với các tính năng nâng cao như Debounce chống nhiễu, OTA Streaming cập nhật từ xa, và Manual Override an toàn với FreeRTOS.
- Giao diện Dashboard và Monitor trực quan, cập nhật thời gian thực với độ trễ thấp (< 2s).
- Tích hợp thành công công nghệ nhận diện biển số (OCR) với độ chính xác cao (> 90%), hỗ trợ đắc lực cho việc quản lý ra vào.
- Tuân thủ các nguyên tắc về bảo mật và an toàn dữ liệu.

8.2. Hướng phát triển

Để hoàn thiện và nâng cao khả năng ứng dụng thực tế, nhóm phát triển đề xuất các hướng cải tiến sau:

1. Edge AI: Triển khai mô hình OCR chạy offline trực tiếp trên các thiết bị biên mạnh mẽ hơn (như Nvidia Jetson Nano hoặc Raspberry Pi) để giảm phụ thuộc vào Internet và tăng tốc độ xử lý.
2. Mobile App: Phát triển ứng dụng di động cho phép người dùng tìm kiếm và đặt chỗ (Booking) trước.
3. Thanh toán điện tử: Tích hợp thanh toán qua QR Code, Ví điện tử (Momo, ZaloPay) để tự động hóa quy trình thu phí.
4. Data Analytics: Bổ sung các biểu đồ thống kê doanh thu, mật độ xe ra vào theo giờ cao điểm trên Dashboard để hỗ trợ ra quyết định quản lý.