

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH  
TRƯỜNG ĐẠI HỌC BÁCH KHOA  
KHOA KHOA HỌC ỨNG DỤNG



BÁO CÁO  
ĐỒ ÁN THIẾT KẾ LUẬN LÝ (CO3091)

BÃI ĐỖ XE THÔNG MINH

Nhóm 09 - HK241

Giảng viên hướng dẫn: Nguyễn Thiên Ân

STT	Họ và tên	MSSV	Ghi chú
1	Ngô Trường Bách	2210183	
2	Nguyễn Phúc An	2210022	

Thành phố Hồ Chí Minh, tháng 12 năm 2024

## Danh sách bảng

1	Use Case: Quét thẻ RFID vào . . . . .	25
2	Use Case: Quét thẻ RFID ra . . . . .	26
3	Use Case: ESP32 . . . . .	27
4	Use Case: Firebase . . . . .	28

## Danh sách hình vẽ

1	Hình ảnh minh họa bãi đỗ xe . . . . .	5
2	ESP32 Development Board - DEVKIT V1 . . . . .	9
3	HC-SR04 Ultrasonic Distance Sensor . . . . .	10
4	IR sensor . . . . .	11
5	Module RFID RC522 . . . . .	12
6	LCD Display Module 16x02 . . . . .	13
7	Servo . . . . .	14
8	Led . . . . .	14
9	Ứng dụng RFID trong bãi đỗ xe . . . . .	16
10	Cấu tạo RFID . . . . .	17
11	Firebase . . . . .	18
12	Sơ đồ hệ thống . . . . .	21
13	flowchart . . . . .	22
14	Biểu đồ trường hợp sử dụng tổng quát . . . . .	23
15	Biểu đồ trường hợp sử dụng ESP 32 . . . . .	23
16	Biểu đồ trường hợp quản lý dữ liệu . . . . .	24
17	Sequence Diagram . . . . .	29
18	Sơ đồ cắm chân vào ESP32 . . . . .	30
19	Sơ đồ tổng quan hệ thống mạch điều khiển . . . . .	31
20	Mạch PCB 2D ở lớp top . . . . .	31
21	Mạch PCB 2D ở lớp bottom . . . . .	31
22	Mạch PCB 3D ở lớp bottom . . . . .	32
23	Mạch PCB 3D ở lớp bottom . . . . .	32
24	Kiến trúc hệ thống . . . . .	33
25	Lưu đồ quá trình nhận diện biển số xe . . . . .	46
26	Hình ảnh ban đầu . . . . .	46
27	Hình ảnh nhị phân . . . . .	47
28	Lưu đồ quá trình nhận diện biển số xe . . . . .	48
29	Hình ảnh của 36 SVM đã được huấn luyện . . . . .	48

# Mục lục

Danh sách bảng	1
Danh sách hình vẽ	2
1 Giới thiệu đề tài	5
2 Yêu cầu hệ thống	6
2.1 Functional Requirements . . . . .	6
2.2 Non-functional Requirements . . . . .	6
2.3 Nội dung đầu vào, đầu ra . . . . .	8
3 Công nghệ và thiết bị ngoại vi sử dụng	9
3.1 Thiết bị ngoại vi . . . . .	9
3.1.1 ESP32 . . . . .	9
3.1.2 Ultrasonic sensor - cảm biến siêu âm . . . . .	10
3.1.3 IR sensor - cảm biến hồng ngoại . . . . .	11
3.1.4 RFID . . . . .	12
3.1.5 LCD I2C . . . . .	13
3.1.6 Servo . . . . .	14
3.1.7 Led . . . . .	14
3.2 Tổng quan về RFID . . . . .	15
3.3 Tổng quan về Firebase . . . . .	17
3.3.1 Giới thiệu . . . . .	17
3.3.2 Các thành phần chính của Firebase . . . . .	18
4 Kiến trúc hệ thống	21
4.1 Sơ đồ hệ thống . . . . .	21
4.1.1 Sơ đồ tổng quan . . . . .	21
4.1.2 Block Diagram . . . . .	22
4.1.3 Use-case Diagram . . . . .	23



---

4.2	Đặc tả trường hợp sử dụng của hệ thống . . . . .	25
4.3	Sequence Diagram . . . . .	29
<b>5</b>	<b>Thiết kế</b>	<b>30</b>
5.1	Thiết kế phần cứng . . . . .	30
5.1.1	Sơ đồ chân . . . . .	30
5.1.2	Thiết kế mạch . . . . .	30
5.1.3	Xây dựng sơ đồ mạch . . . . .	31
5.1.4	Thiết kế PCB . . . . .	31
5.2	Thiết kế hệ thống . . . . .	32
5.2.1	Tổng quan . . . . .	32
5.2.2	Kiến trúc hệ thống . . . . .	33
<b>6</b>	<b>Hiện thực</b>	<b>34</b>
6.1	Hiện thực ESP32 . . . . .	34
6.1.1	Khởi tạo thư viện và chân . . . . .	34
6.1.2	Wifi và Firebase . . . . .	35
6.1.3	Các hàm phụ . . . . .	40
6.2	Nhận diện số biển số xe bằng phương pháp vector hỗ trợ (SVM) . . . . .	46
6.2.1	Phát triển thuật toán cho việc nhận diện số biển số xe . . . . .	46
<b>7</b>	<b>Mô phỏng</b>	<b>50</b>
7.1	Phần Cứng . . . . .	50
7.2	Web App và Mã nguồn . . . . .	51

## 1 Giới thiệu đề tài

Hiện nay, nhu cầu sử dụng phương tiện để đi lại của người dân ở các vùng đô thị là đặc biệt lớn. Điều này tạo lên áp lực không nhỏ vào việc quản lý số lượng xe trong cùng một không gian vào cùng một thời điểm, hay cụ thể hơn là các bãi đỗ xe.

Sự phát triển của công nghệ, cụ thể trong lĩnh vực IOT đã cung cấp cho ta những công cụ để giảm thiểu vấn đề thiếu hụt chỗ đỗ xe, quản lý hiệu hơn và giảm thiểu tác động môi trường đô thị.



**Hình 1:** Hình ảnh minh họa bãi đỗ xe

Trong đồ án này, nhóm hướng tới việc hiện thực một hệ thống bãi đỗ xe thông minh ứng dụng công nghệ IOT. Hệ thống bao gồm các cảm biến để xác định vị trí của các xe trong bãi đỗ xe. Ở phía bên ngoài là cổng chắn được tự động hóa bằng servo và thẻ RFID lưu thông tin liên quan của khách hàng: mã khách hàng, biển số xe (được trích xuất từ camera đặt ở cổng). Khách hàng đồng thời cũng có thể biết được thông tin của bãi đỗ xe qua màn hình LCD ở phía cổng ra/vào.

Ngoài ra, nhóm cũng cung cấp cho người quản lý (admin) giao diện trên web app để theo dõi thông tin từ bãi đỗ xe.

## 2 Yêu cầu hệ thống

### 2.1 Functional Requirements

**Đối với khách hàng:**

- Xem thông tin bãi: khách hàng được xem thông tin qua LCD ở cổng trước như số lượng xe trong bãi.
- Xác thực ra vào bãi xe bằng thẻ RFID.

**Đối với admin:**

- Có thể xem các thông tin qua app như vị trí đỗ (còn trống hoặc đã đầy), số lượng vị trí đỗ, thời gian vào ra, biển số xe của khách hàng,...

**Đối với người phát triển hệ thống:**

- Người phát triển hệ thống có thể mở rộng và bảo trì dễ dàng, ứng dụng có giao diện dễ sử dụng.
- Tích hợp liền mạch giữa các thành phần phần cứng (cảm biến, barrier) và phần mềm (app, server).
- Đảm bảo hệ thống hoạt động ổn định, bảo mật và đáp ứng đúng yêu cầu của nhà quản lý và khách hàng.

### 2.2 Non-functional Requirements

**Về hiệu suất (Performance)**

- Hệ thống khi hoạt động cần phản hồi nhanh những sự thay đổi từ bãi về app cũng như chuyển tín hiệu nhanh từ app đến cổng barrier trong trường hợp admin muốn điều khiển thủ công.
- Thời gian mở và đóng barrier cần diễn ra rất nhanh chóng sau khi xác nhận thẻ RFID hoặc biển số xe.

## Về sẵn sàng (Availability)

- Hệ thống cần đảm bảo hoạt động ổn định.
- Hệ thống phải có khả năng phục hồi nhanh sau các sự cố.

## Bảo mật (Security)

- Bảo vệ dữ liệu người dùng như biển số xe,....
- Chỉ những xe có thẻ RFID hợp lệ mới có thể ra vào bãi.

## Về khả năng mở rộng (Scalability)

- Có thể tăng cường số lượng slot đỗ xe và mở rộng ra nhiều bãi đỗ xe khác trong một khu vực rộng hơn.

## Về khả năng sử dụng (Usability)

- Giao diện dễ sử dụng: Ứng dụng và giao diện phải trực quan, dễ sử dụng cho người dùng ở mọi lứa tuổi và trình độ công nghệ.
- Các thông tin như số lượng xe trong bãi, vị trí đỗ (trống hay đã đầy),... cần được hiển thị rõ ràng, dễ hiểu trên màn hình LCD tại cổng barrier và trong app.

## Về độ tin cậy (Reliability)

- Độ chính xác của thông tin về trạng thái, vị trí đỗ, số lượng xe,... phải cực kỳ chính xác.
- Những tín hiệu được kết nối với nhau như cảm biến, cổng barrier,... phải đồng bộ.

## Về sự linh hoạt (Flexibility)

- Tích hợp với các hệ thống thanh toán khác như thẻ tín dụng, ví điện tử, ngân hàng,...

## 2.3 Nội dung đầu vào, đầu ra

Đối với Mobile app:

- Input:

- Cảm biến ở chỗ đỗ xe
- Camera được lắp ở cổng
- Nhận sự điều khiển của admin

- Output:

- Trạng thái của slot sau khi xe vào/ra
- Số lượng xe sau khi xe vào/ra
- Hiển thị biển số xe
- Hiển thị thời gian vào/ra
- Số lượng slot còn trống và đã đầy
- Gửi tín hiệu đóng/mở đến cổng barrier trong trường hợp đóng/mở thủ công

Đối với hệ thống nói chung:

- Input:

- Tín hiệu từ cảm biến trước cổng barrier vào, thẻ RFID, biển số xe qua camera
- Yêu cầu đóng/mở barrier từ hệ thống hoặc admin khi xe ra/vào

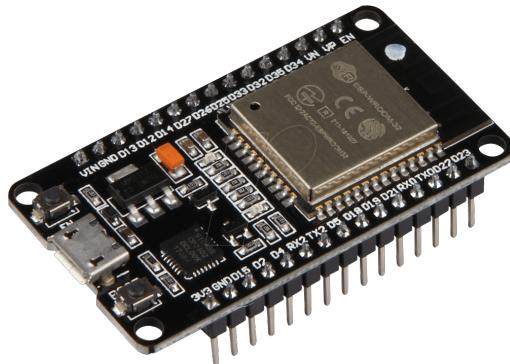
- Output:

- Mở/dóng barrier tự động khi nhận thông tin hợp lệ (biển số, thời gian vào ra) từ RFID
- Dữ liệu về số lượng xe trong bãi, số lượng, trạng thái slot (còn trống hay đã đầy), biển số xe

### 3 Công nghệ và thiết bị ngoại vi sử dụng

#### 3.1 Thiết bị ngoại vi

##### 3.1.1 ESP32



**Hình 2:** ESP32 Development Board - DEVKIT V1

**Số lượng sử dụng:** 1

**Đặc điểm kỹ thuật:**

- Kết nối: Wi-Fi, Bluetooth.
- GPIO: 30 chân GPIO, hỗ trợ ADC, PWM, I2C, UART, SPI,..
- Bộ nhớ: 520 KB SRAM.

**Chức năng:**

- Là bộ điều khiển chính trong hệ thống, kết nối và điều phối các cảm biến, servo, và hiển thị thông tin.
- Quản lý giao tiếp với Firebase và ứng dụng web.

**Cách hoạt động:**

- Kết nối với các cảm biến (RFID, siêu âm, hồng ngoại), nhận tín hiệu từ các thiết bị ngoại vi, xử lý và điều khiển các thiết bị như LCD, servo, LED.

### 3.1.2 Ultrasonic sensor - cảm biến siêu âm



**Hình 3:** HC-SR04 Ultrasonic Distance Sensor

**Số lượng sử dụng:** 2

**Đặc điểm kỹ thuật:**

- Khoảng cách đo: 2 cm đến 400 cm.
- Độ chính xác:  $\pm 3$  mm.
- Tần số hoạt động: 40 kHz.

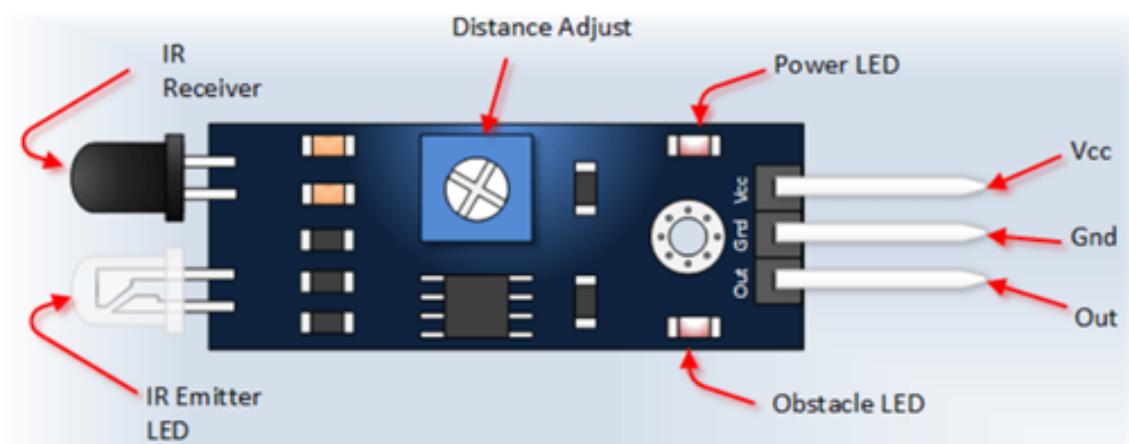
**Chức năng:**

- Đo khoảng cách và xác định trạng thái các chỗ đỗ xe (trống hay đầy).

**Cách hoạt động:**

- Gửi sóng siêu âm và đo thời gian phản hồi từ vật thể để tính toán khoảng cách.

### 3.1.3 IR sensor - cảm biến hồng ngoại



Hình 4: IR sensor

Số lượng sử dụng: 2

Đặc điểm kỹ thuật:

- Khoảng cách phát hiện: 2 - 30 cm.
- Điện áp hoạt động: 5V.
- Loại tín hiệu: Dạng số (0 hoặc 1).

Chức năng:

- Phát hiện sự có mặt của xe trong các khu vực hẹp, bổ sung cho cảm biến siêu âm.

Cách hoạt động:

- Cảm biến có khả năng nhận biết vật cản ở môi trường với một cặp LED thu phát hồng ngoại để truyền và nhận dữ liệu hồng ngoại. Tia hồng ngoại phát ra với tần số nhất định, khi có vật cản trên đường truyền của LED phát nó sẽ phản xạ vào LED thu hồng ngoại, khi đó LED báo vật cản trên module sẽ sáng, khi không có vật cản, LED sẽ tắt.

### 3.1.4 RFID



**Hình 5:** Module RFID RC522

**Số lượng sử dụng:** 2

**Đặc điểm kỹ thuật:**

- Loại: RC522.
- Tần số: 13.56 MHz.
- Khoảng cách đọc: 0 - 60mm (mifare1 card).

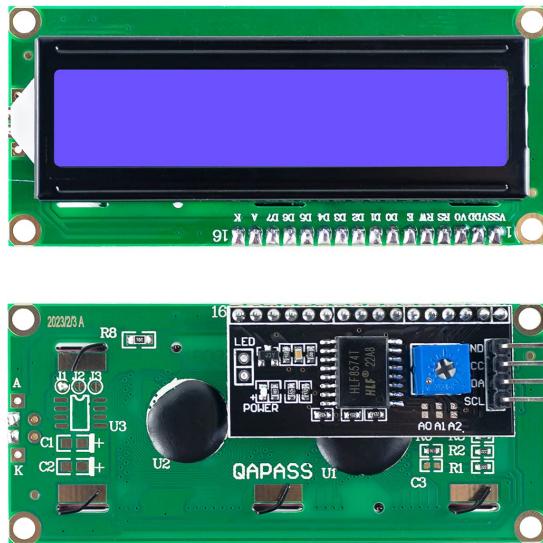
**Chức năng:**

- Quét và nhận diện thẻ RFID để xác thực xe vào/ra.

**Cách hoạt động:**

- Module RFID đọc dữ liệu từ thẻ RFID (UID), xác thực và điều khiển cổng barrier mở hoặc đóng.

### 3.1.5 LCD I2C



**Hình 6:** LCD Display Module 16x02

**Số lượng sử dụng:** 1

**Đặc điểm kỹ thuật:**

- Loại: LCD 16x2 với giao tiếp I2C.
- Điện áp hoạt động: 5V.
- Kích thước hiển thị: 16 ký tự x 2 dòng.

**Chức năng:**

- Hiển thị thông tin về trạng thái bãi xe, số lượng xe còn lại, thông báo lỗi hoặc thông tin khác.

**Cách hoạt động:**

- ESP32 gửi dữ liệu qua giao thức I2C để hiển thị lên màn hình.

### 3.1.6 Servo



Hình 7: Servo

Số lượng sử dụng: 2

Đặc điểm kỹ thuật:

- Góc quay:  $0^\circ$  đến  $180^\circ$ .
- Điện áp hoạt động: 4.8V đến 6V.

Chức năng:

- Điều khiển thanh chắn tự động tại các cổng vào/ra của bãi đỗ xe.

Cách hoạt động:

- Servo quay theo góc yêu cầu để mở hoặc đóng barrier (cổng bãi đỗ xe).

### 3.1.7 Led



Hình 8: Led

Số lượng sử dụng: 2

Chức năng:

- Sáng đèn khi quét thẻ thành công.

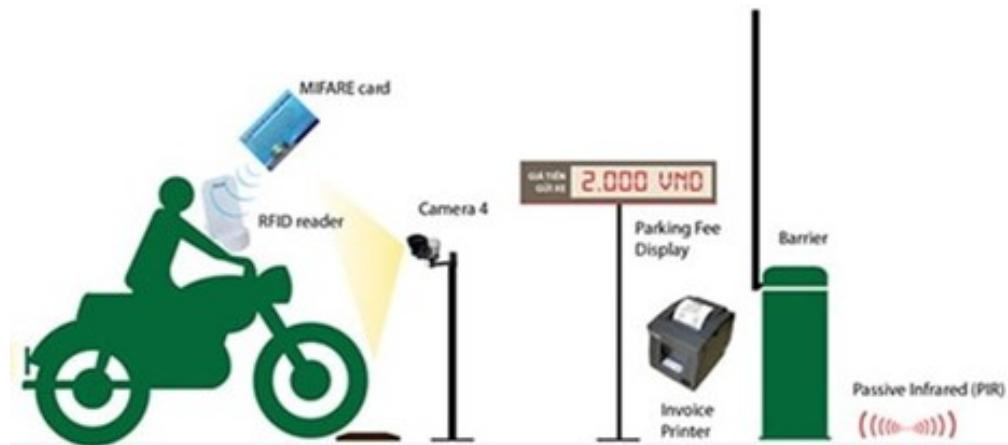
Cách hoạt động:

- Khi người dùng quét thẻ RFID, hệ thống sẽ kiểm tra tính hợp lệ của thẻ qua module RFID.
- Nếu thẻ hợp lệ (có mã UID đúng), hệ thống sẽ gửi tín hiệu tới ESP32 để bật đèn LED, báo hiệu rằng thẻ đã được xác nhận thành công.
- Đèn LED sẽ sáng lên để báo hiệu rằng chỗ đỗ xe còn trống hoặc thẻ đã được xác nhận thành công.

### 3.2 Tổng quan về RFID

Công nghệ RFID (Radio Frequency Identification) là một trong những công nghệ nhận dạng dữ liệu tự động tiên tiến nhất hiện nay. Với tính khả thi cao và hiệu quả thực tế, RFID đã và đang hiện diện trong nhiều lĩnh vực tự động hóa, quản lý và tổ chức nhằm đem lại giải pháp nhận dạng dữ liệu tự động tối ưu.

RFID cho phép một thiết bị đọc thông tin chứa trong chip mà không cần tiếp xúc trực tiếp ở khoảng cách xa, không cần giao tiếp vật lý hay nhìn thấy đối tượng. Kỹ thuật này sử dụng truyền thông không dây qua sóng vô tuyến để truyền dữ liệu từ *tag* (thẻ) đến *reader* (đầu đọc).



**Hình 9:** Ứng dụng RFID trong bãi đỗ xe

Hệ thống RFID bị động, dạng đơn giản nhất, hoạt động như sau:

- Reader phát tín hiệu sóng vô tuyến qua ăng-ten đến tag.
- Tag phản hồi thông tin trả lại reader.
- Reader gửi thông tin này đến máy tính điều khiển để xử lý.

Các tag RFID không cần nguồn năng lượng riêng, chúng sử dụng năng lượng từ tín hiệu của reader để hoạt động.

### Cấu tạo hệ thống RFID

Là một thẻ gắn chip + anten. Được lập trình điện tử với thông tin duy nhất. Thẻ RFID là một thiết bị lưu trữ và truyền dữ liệu đến một đầu đọc trong một môi trường tiếp xúc bằng sóng vô tuyến. Thẻ RFID mang dữ liệu một vật, một sản phẩm (item...) nào đó và gắn lên sản phẩm đó. Mỗi thẻ có các phần lưu trữ dữ liệu bên trong và cách giao tiếp với dữ liệu đó.

Vài thẻ RFID giống như những nhãn giấy và được ứng dụng để bỏ vào hộp và đóng gói. Một số khác được sáp nhập thành vách của thùng chứa plastic được đúc. Còn một số khác được xây dựng thành miếng da bao cổ tay. Mỗi thẻ được lập trình với một nhận dạng duy nhất cho phép theo dõi không dây đối tượng hoặc con người đang gần thẻ đó. Thông thường, mỗi thẻ RFID có một cuộn dây hoặc anten nhưng không phải tất cả RFID đều có vi chip và nguồn năng lượng riêng.



Hình 10: Cấu tạo RFID

Gồm 2 phần chính:

- **Chip:** Bộ nhớ của chip có thẻ chứa tối 96 bit đến 512 bit dữ liệu, gấp 64 lần so với mã vạch. Chip lưu trữ một số thứ tự duy nhất hoặc thông tin khác dựa trên loại thẻ:
  - *Read-only*: Chỉ đọc.
  - *Read-write*: Đọc và ghi dữ liệu.
- **Antenna:** Được gắn với vi mạch để truyền thông tin từ chip đến reader. Công suất của anten càng lớn thì phạm vi đọc càng lớn.

### 3.3 Tổng quan về Firebase

#### 3.3.1 Giới thiệu

Firebase là một nền tảng phát triển ứng dụng di động và web toàn diện được Google cung cấp. Với khả năng hỗ trợ từ phía máy chủ, quản lý cơ sở dữ liệu, xác thực người dùng, đến triển khai ứng dụng, Firebase trở thành công cụ mạnh mẽ cho các nhà phát triển trong việc xây dựng ứng dụng nhanh chóng và hiệu quả.

Firebase cung cấp nhiều dịch vụ đa dạng, tích hợp dễ dàng và có thể mở rộng, giúp đơn giản hóa quy trình phát triển và quản lý ứng dụng.



# Firebase

**Hình 11:** Firebase

### 3.3.2 Các thành phần chính của Firebase

**a. Realtime Database** Firebase Realtime Database là cơ sở dữ liệu NoSQL, lưu trữ và đồng bộ dữ liệu theo thời gian thực giữa người dùng và máy chủ.

- Dữ liệu được lưu trữ dưới dạng JSON.
- Đồng bộ hóa tự động giữa các thiết bị.
- Hỗ trợ truy cập ngoại tuyến (Offline Mode).

**b. Firestore** Firestore (Cloud Firestore) là cơ sở dữ liệu nâng cao hơn, hỗ trợ truy vấn linh hoạt và cấu trúc dữ liệu phức tạp:

- Dữ liệu được tổ chức theo *collections* và *documents*.
- Khả năng mở rộng tốt, phù hợp cho các ứng dụng lớn.
- Hỗ trợ thời gian thực và truy vấn mạnh mẽ.

**c. Authentication** Firebase Authentication giúp quản lý người dùng và xác thực ứng dụng:

- Hỗ trợ xác thực qua email, số điện thoại, Google, Facebook, GitHub, v.v.
- Tích hợp API dễ sử dụng để đăng nhập và quản lý tài khoản.
- Đảm bảo an toàn và bảo mật thông tin người dùng.

**d. Cloud Functions** Firebase Cloud Functions là dịch vụ thực thi mã nguồn trên đám mây dựa trên các sự kiện:

- Tự động chạy khi có sự kiện trong ứng dụng hoặc cơ sở dữ liệu.
- Giảm tải công việc phía máy khách (client).
- Có thể tích hợp với các dịch vụ khác của Google Cloud.

**e. Hosting** Firebase Hosting là dịch vụ lưu trữ và triển khai ứng dụng web:

- Hỗ trợ HTTPS, CDN và cache để tăng tốc độ truy cập.
- Dễ dàng triển khai với một dòng lệnh.
- Hỗ trợ Progressive Web Apps (PWA).

**f. Cloud Messaging (FCM)** Firebase Cloud Messaging cung cấp dịch vụ gửi thông báo:

- Gửi thông báo từ máy chủ đến ứng dụng trên Android, iOS và web.
- Hỗ trợ thông báo theo nhóm hoặc cá nhân.
- Miễn phí và dễ tích hợp.

**g. Analytics** Firebase Analytics giúp theo dõi và phân tích hành vi người dùng:

- Cung cấp các báo cáo chi tiết về sự tương tác của người dùng.
- Hỗ trợ tạo các sự kiện tùy chỉnh.
- Kết nối dễ dàng với Google Ads và các công cụ marketing khác.

**h. Ưu điểm của Firebase**

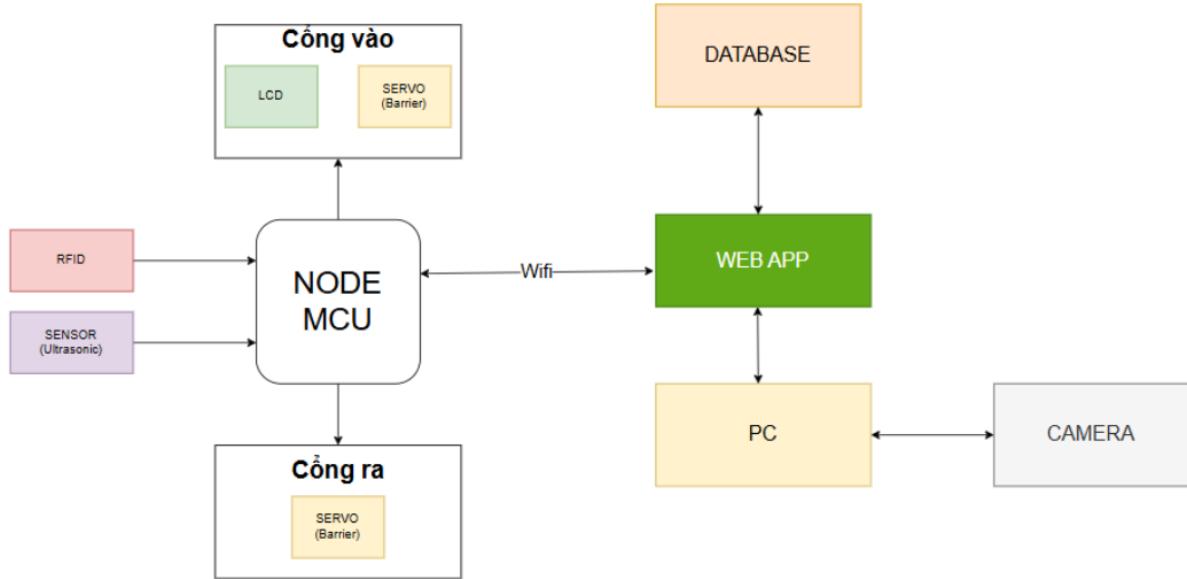
- **Tích hợp dễ dàng:** Firebase cung cấp SDK và API sẵn sàng để sử dụng.

- **Hỗ trợ đa nền tảng:** Hỗ trợ Android, iOS và web.
- **Mở rộng linh hoạt:** Firebase có thể đáp ứng yêu cầu của các ứng dụng từ nhỏ đến lớn.
- **Miễn phí cho nhu cầu cơ bản:** Firebase cung cấp gói miễn phí phù hợp cho các dự án nhỏ.
  - i. **Ứng dụng thực tiễn**
    - Ứng dụng chat thời gian thực.
    - Quản lý dữ liệu trong hệ thống IoT.
    - Tích hợp xác thực người dùng.
    - Gửi thông báo đến người dùng.

## 4 Kiến trúc hệ thống

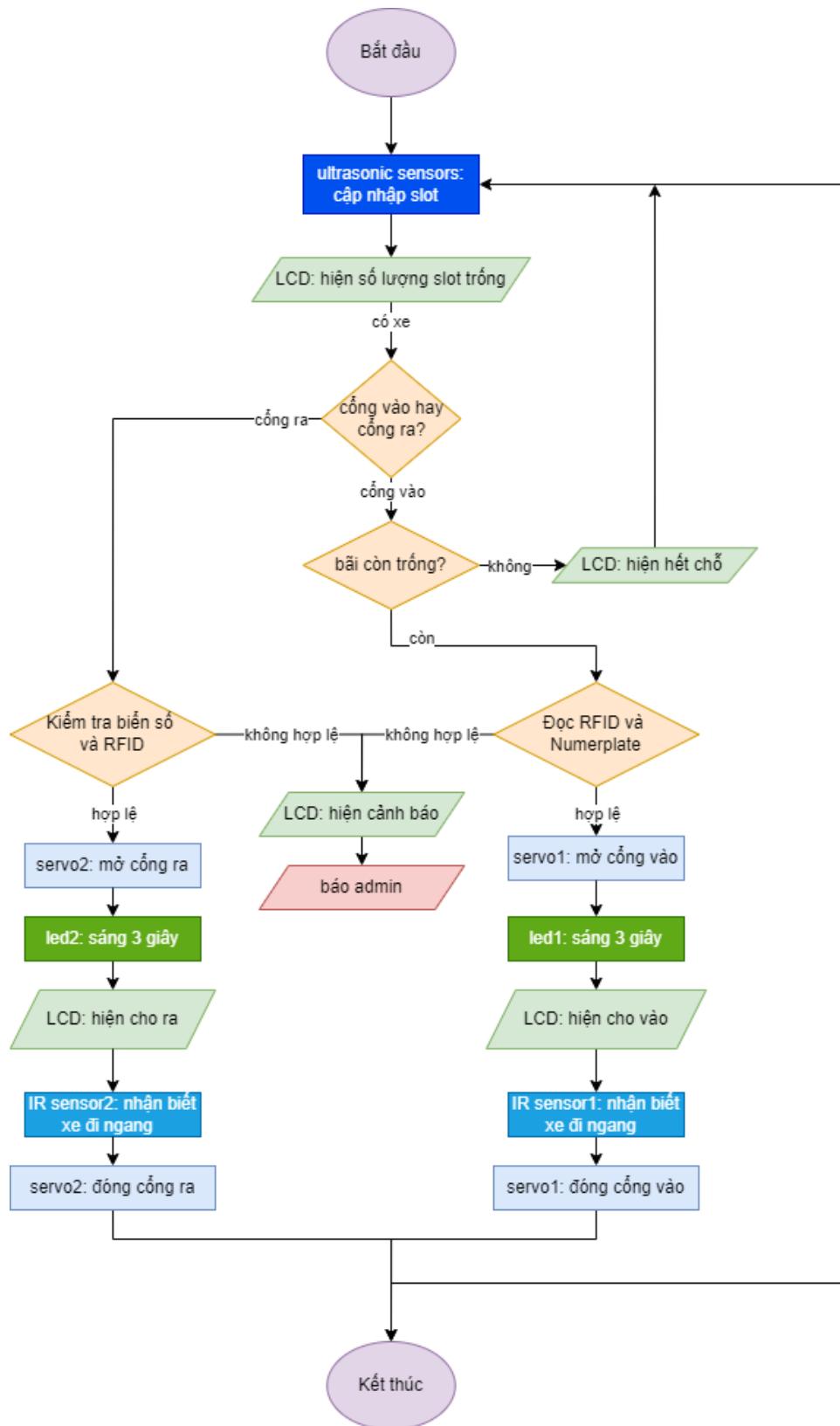
### 4.1 Sơ đồ hệ thống

#### 4.1.1 Sơ đồ tổng quan



**Hình 12:** Sơ đồ hệ thống

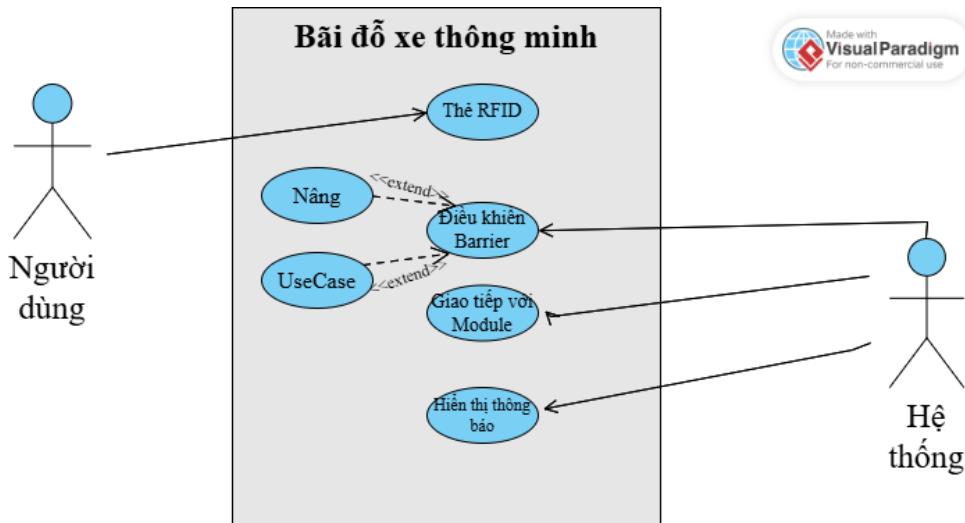
#### 4.1.2 Block Diagram



Hình 13: flowchart

#### 4.1.3 Use-case Diagram

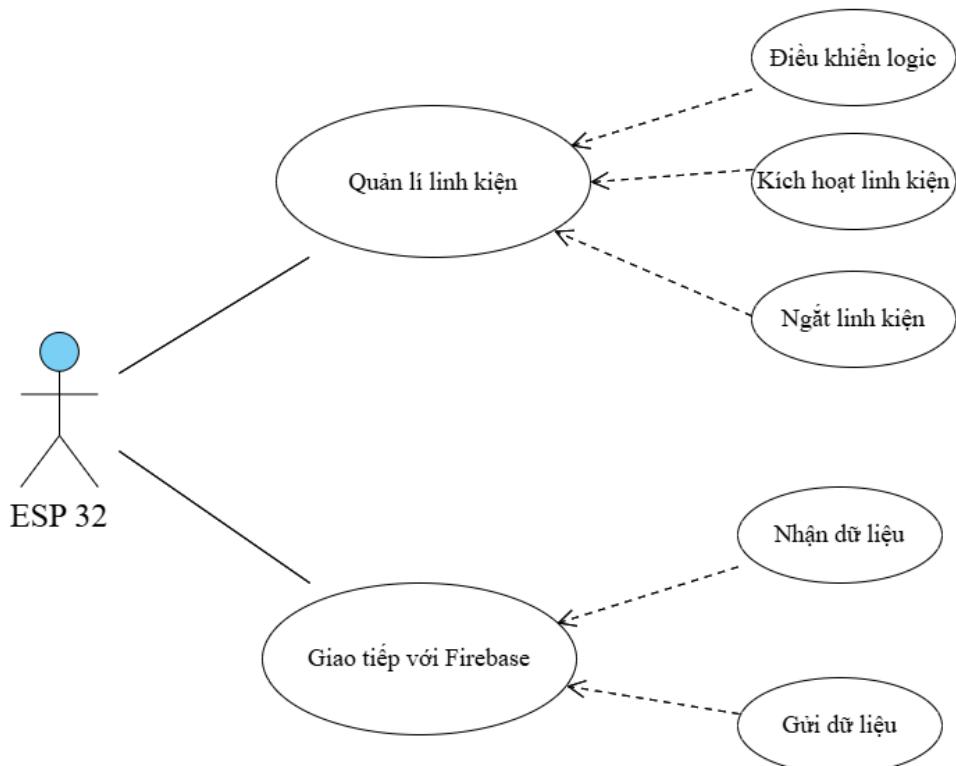
Biểu đồ trường hợp sử dụng tổng quát



Hình 14: Biểu đồ trường hợp sử dụng tổng quát

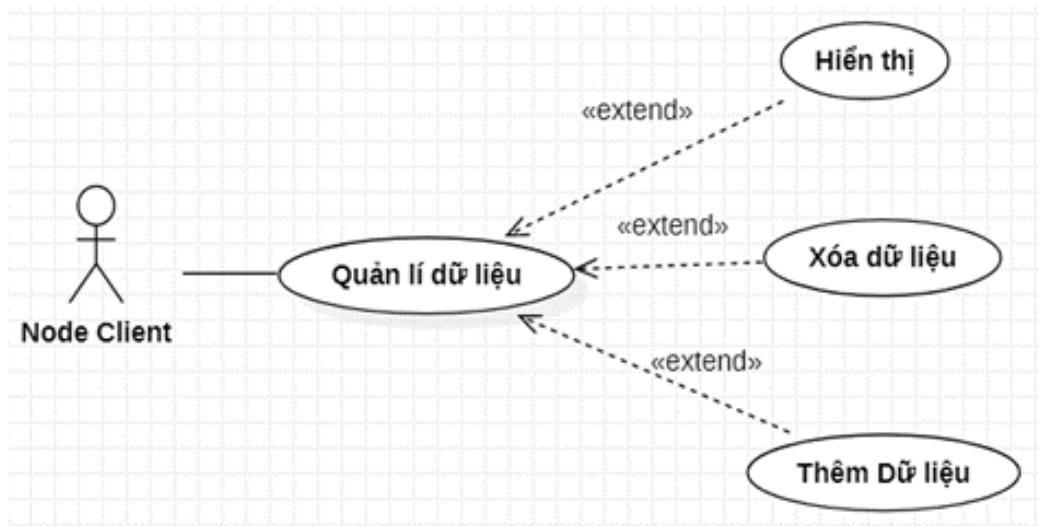
Biểu đồ trường hợp sử dụng chi tiết

a) Use case Esp32



Hình 15: Biểu đồ trường hợp sử dụng ESP 32

b) Usecase quản lí dữ liệu



**Hình 16:** Biểu đồ trường hợp quản lí dữ liệu

## 4.2 Đặc tả trường hợp sử dụng của hệ thống

Bảng 1: Use Case: Quét thẻ RFID vào

Use case	Quét RFID vào
Tác nhân	Người dùng
Mô tả	Module RFID quét thẻ RFID, lưu dữ liệu xe lên database
Luồng sự kiện	<ol style="list-style-type: none"><li>1. Người dùng quét thẻ RFID vào Module RFID.</li><li>2. Sau khi nhận tín hiệu quét thẻ, thực hiện các thao tác tạo gói dữ liệu chuẩn bị gửi lên database:<ol style="list-style-type: none"><li>(a) Dữ liệu mã ID Card RFID.</li><li>(b) Dữ liệu ảnh chụp từ camera.</li></ol></li><li>3. Sau khi khởi tạo xong dữ liệu:<ol style="list-style-type: none"><li>3.1 Nếu đúng barrier mở và chuyển sang bước 4.</li><li>3.2 Nếu sai, thực hiện lại bước 2.</li></ol></li><li>4. Dữ liệu được gửi lên database.</li><li>5. Hệ thống thực hiện chức năng và thông báo cho người dùng.</li></ol>

**Bảng 2:** Use Case: Quét thẻ RFID ra

Thuộc tính	Mô tả
Use Case	Quét thẻ RFID ra
Tác nhân	Người dùng
Mô tả	Module RFID quét thẻ RFID, kiểm tra dữ liệu trên database
Luồng sự kiện	<ol style="list-style-type: none"><li>1. Người dùng quét thẻ RFID vào Module RFID.</li><li>2. Sau khi nhận tín hiệu quét thẻ, thực hiện các thao tác tạo gói dữ liệu chuẩn bị gửi lên database:<ol style="list-style-type: none"><li>(a) Dữ liệu mã ID Card RFID.</li><li>(b) Dữ liệu ảnh chụp từ camera.</li></ol></li><li>3. Dữ liệu được gửi lên database.</li><li>4. Kiểm tra với gói dữ liệu cồng vào:<ul style="list-style-type: none"><li>• Nếu đúng thì chuyển sang bước 4.</li><li>• Ngược lại, thực hiện lại.</li></ul></li><li>5. Cổng barrier mở</li></ol>

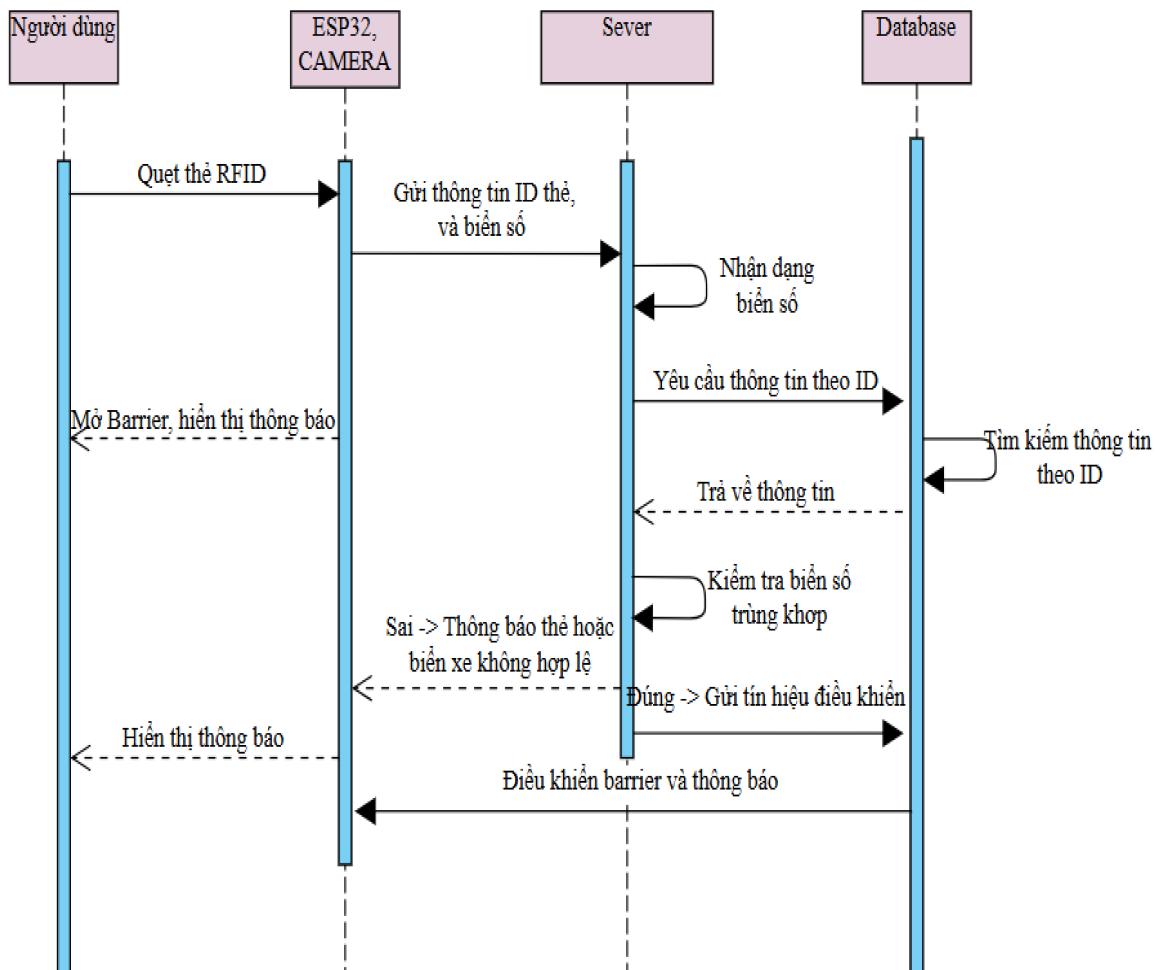
**Bảng 3:** Use Case: ESP32

Thuộc tính	Mô tả
Use Case	ESP32
Tác nhân	Hệ Thống
Mô tả	Quản lý các module và gửi tín hiệu lên server
Luồng sự kiện	<ol style="list-style-type: none"><li>1. ESP32 kiểm tra Module RFID để đọc thẻ và gửi tín hiệu cho Module điều khiển</li><li>2. Module điều khiển:<ol style="list-style-type: none"><li>(a) Mở barrier cổng vào nếu nhận được tín hiệu, thông báo qua LCD.</li><li>(b) Thông báo thẻ hoặc biển số xe không hợp lệ.</li></ol></li><li>3. Gửi dữ liệu ID, biển số xe và số lượng xe lên server.</li><li>4. Nhận dữ liệu từ server khi người dùng quét RFID cổng ra.<ul style="list-style-type: none"><li>• Mở barrier nếu dữ liệu đúng. Thực thi bước 5.</li><li>• Ngược lại, thông báo thẻ hoặc biển không hợp lệ</li></ul></li><li>5. Gửi yêu cầu xóa gói dữ liệu cũ</li></ol>

**Bảng 4:** Use Case: Firebase

Thuộc tính	Mô tả
Use Case	Firebase
Tác nhân	Hệ Thống
Mô tả	Quản lý dữ liệu
Luồng sự kiện	<ol style="list-style-type: none"><li>1. Nhận dữ liệu ID, biển số xe cồng vào<ol style="list-style-type: none"><li>(a) Bài đỗ còn trống, nhận dữ liệu ID và biển số</li><li>(b) Ngược lại, không nhận ID và thông báo.</li></ol></li><li>2. Kiểm tra dữ liệu ID, biển số xe cồng ra<ol style="list-style-type: none"><li>(a) Trùng khớp, xóa dữ liệu cũ.</li><li>(b) Ngược lại, thông báo thẻ hoặc biển không hợp lệ.</li></ol></li><li>3. Nhận dữ liệu từ Web khi Admin điều khiển.<ul style="list-style-type: none"><li>• Gửi tín hiệu cho ESP.</li><li>• Ngược lại, thông báo không thành công</li></ul></li></ol>

### 4.3 Sequence Diagram

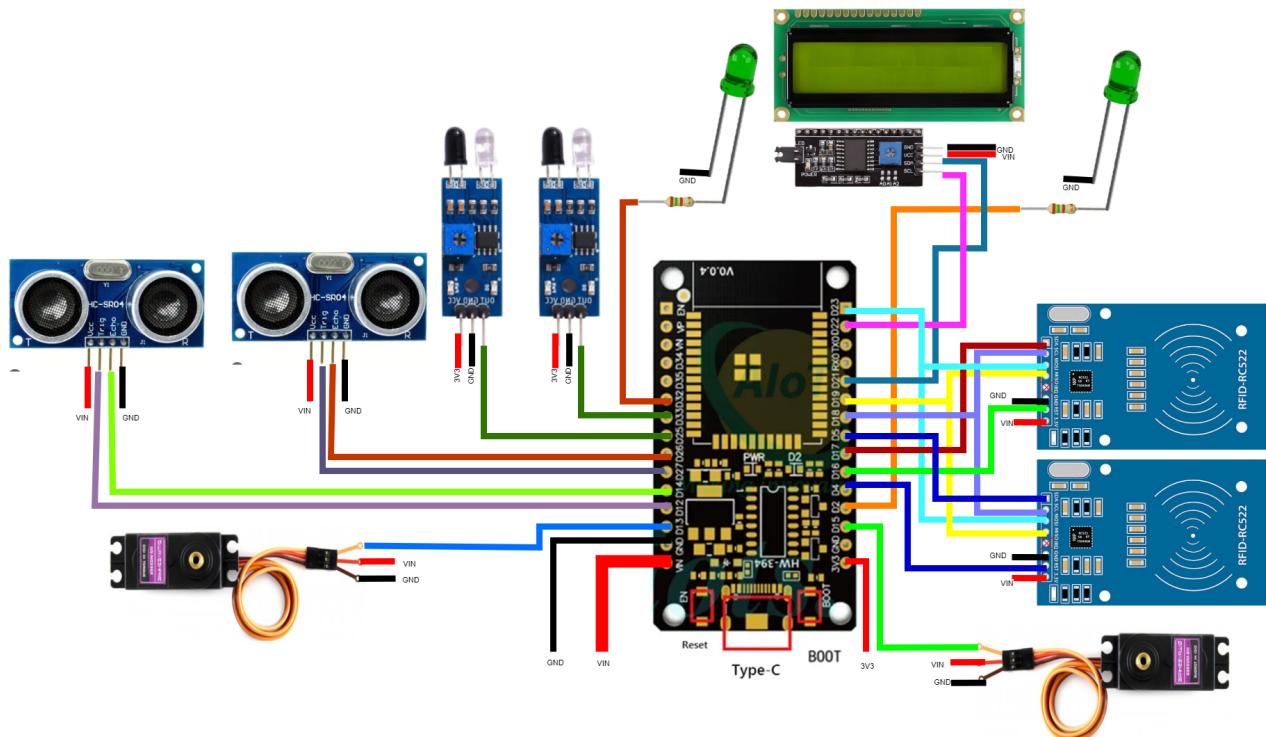


Hình 17: Sequence Diagram

## 5 Thiết kế

### 5.1 Thiết kế phần cứng

#### 5.1.1 Sơ đồ chân



Hình 18: Sơ đồ cắm chân vào ESP32

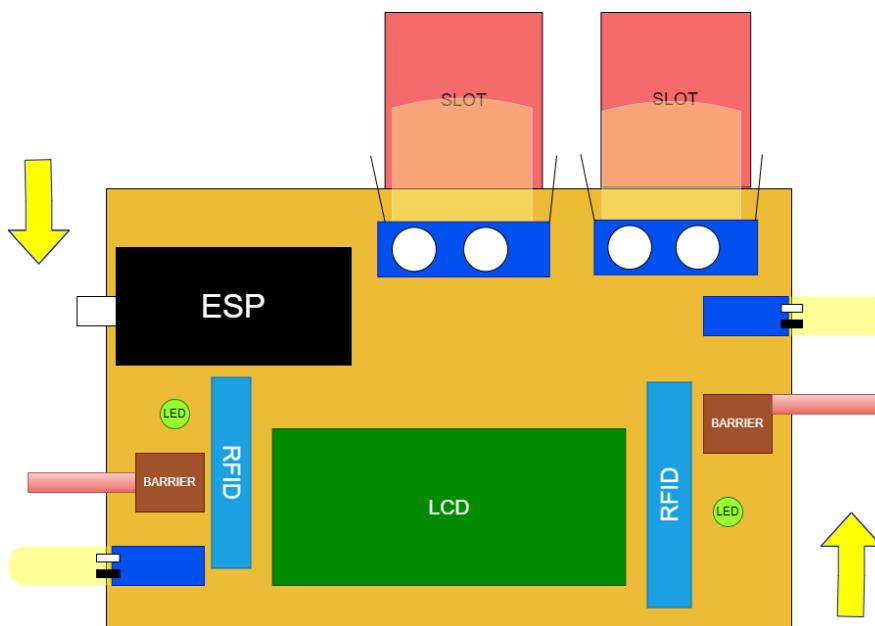
#### 5.1.2 Thiết kế mạch

Nhóm quyết định sẽ thiết kế mạch PCB bằng phần mềm Altium thay vì sử dụng testboard hoặc breadboard để cắm dây trực tiếp, với các lý do như sau:

- + Không cần phải cắm dây lại nhiều lần, dễ dàng kiểm tra và sửa chữa.
- + Kết nối chắc chắn, tránh tình trạng lỏng dây.
- + Tránh nhầm lẫn trong việc cắm dây.
- + Không sử dụng quá nhiều dây cắm, mạch dễ dàng tổ chức do nhóm sử dụng hầu hết các chân của ESP32.
- + Dễ dàng nạp code, không mất thời gian để cắm dây từ đầu.

### 5.1.3 Xây dựng sơ đồ mạch

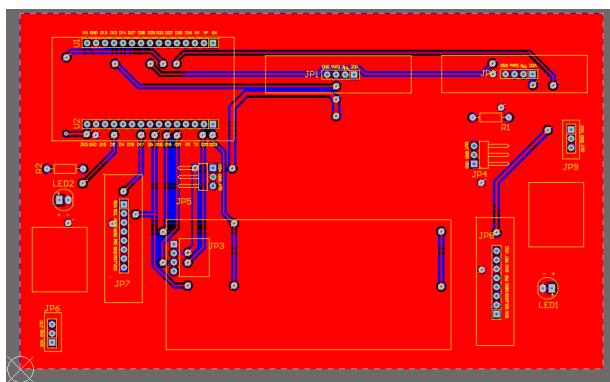
Đầu tiên, nhóm muốn thể hiện một cái nhìn tổng quan về cấu trúc và các thành phần chính trong hệ thống mạch điều khiển. Sơ đồ bên dưới giúp dễ dàng hình dung và hiểu rõ vị trí các linh kiện và module, từ đó hỗ trợ việc thiết kế và kiểm tra mạch.



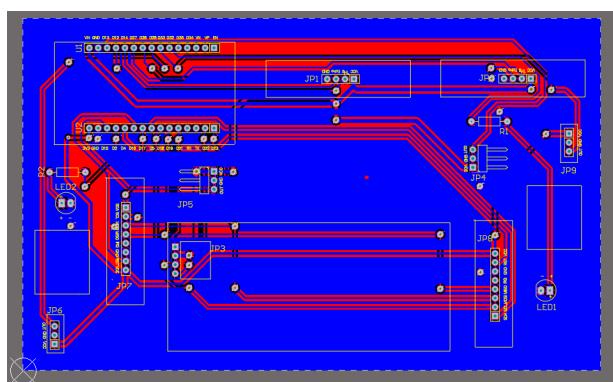
**Hình 19:** Sơ đồ tổng quan hệ thống mạch điều khiển

### 5.1.4 Thiết kế PCB

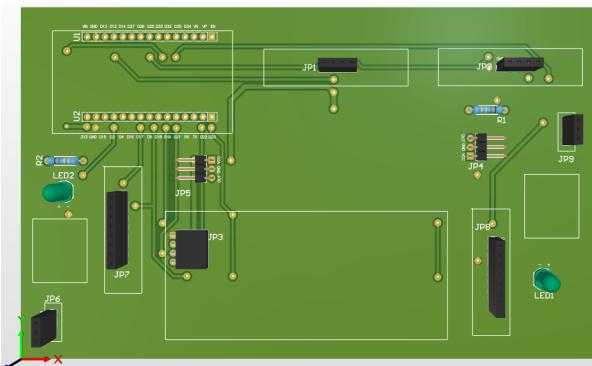
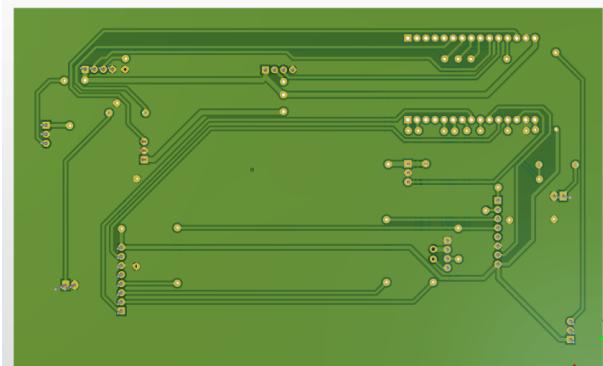
Sau đó, nhóm sử dụng phần mềm Altium để thiết kế mạch:



**Hình 20:** Mạch PCB 2D ở lớp top



**Hình 21:** Mạch PCB 2D ở lớp bottom

**Hình 22:** Mạch PCB 3D ở lớp bottom**Hình 23:** Mạch PCB 3D ở lớp bottom

## 5.2 Thiết kế hệ thống

### 5.2.1 Tổng quan

Trong dự án này, chúng em sử dụng các dịch vụ **Firebase Realtime Database** và **Firebase Hosting** từ Firebase. Các dịch vụ này được cung cấp theo gói Spark miễn phí với một số giới hạn, tuy nhiên chúng không gây ảnh hưởng đến mục đích của dự án.

**Firebase Realtime Database** được sử dụng để:

- Lưu trữ trạng thái, giá trị của các thiết bị IoT.
- Thực hiện giao tiếp hai chiều giữa thiết bị IoT và ứng dụng web.

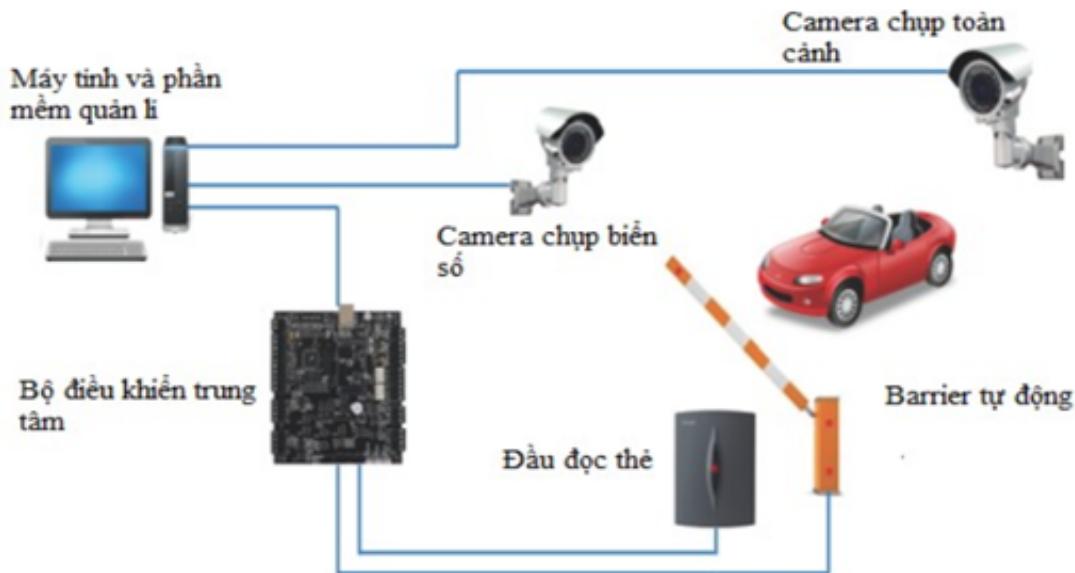
### Cấu trúc dữ liệu

- Là một tệp JSON dạng lồng nhau (*nested JSON*) chứa tất cả các cặp **Key** và **Value**.
- Giá trị (*Value*) có thể được thay đổi từ cả thiết bị IoT lẫn giao diện web, dựa vào các logic được thiết lập.

**Firebase Hosting** được sử dụng để:

- Lưu trữ ứng dụng web để tương tác với Firebase Realtime Database.

### 5.2.2 Kiến trúc hệ thống



**Hình 24:** Kiến trúc hệ thống

Sau khi camera đọc biển số và module RFID đọc dữ liệu, cấu trúc dữ liệu dạng *JSON lồng nhau* (nested JSON) với các trường chính sẽ như sau:

```
1  {
2      "parking_lot": {
3          "slot_1": {
4              "status": "occupied",
5              "UID": "1234567890"
6          },
7          "slot_2": {
8              "status": "empty",
9              "UID": null
10         }
11     },
12     "barrier": {
13         "status": "closed"
14     },
15     "vehicleCount": 1
16 }
```

## 6 Hiệu thực

### 6.1 Hiệu thực ESP32

#### 6.1.1 Khởi tạo thư viện và chân

```
1 // Lib
2 #include <SPI.h>
3 #include <MFRC522.h>
4 #include <ESP32Servo.h>
5 #include <LiquidCrystal_I2C.h>
6 #include <FirebaseESP32.h>
7
8 // RFID
9 #define RST_PIN_R 4
10 #define SS_PIN_R 5
11 #define RST_PIN_L 16
12 #define SS_PIN_L 17
13
14 // LEDs
15 #define LED_PIN_R 32
16 #define LED_PIN_L 2
17
18 // Sensor
19 #define IR_SENSOR_PIN 33
20 #define TRIG_PIN_1 12
21 #define ECHO_PIN_1 14
22 #define TRIG_PIN_2 27
23 #define ECHO_PIN_2 26
24
25 // Servo
26 Servo servo1, servo2;
27 #define SERVO_PIN_R 13
28 #define SERVO_PIN_L 15
29
30 // LCD
```

```
31 LiquidCrystal_I2C lcd(0x27, 16, 2);  
32  
33 // MRC  
34 MFRC522 mfrc522_R(SS_PIN_R, RST_PIN_R);  
35 MFRC522 mfrc522_L(SS_PIN_L, RST_PIN_L);  
36 MFRC522 mfrc522_R2(SS_PIN_R, RST_PIN_R);
```

## 6.1.2 Wifi và Firebase

### 6.1.2.1 Xử lí Wifi

```
1 // WiFi info  
2 #define WIFI_SSID "Beckhcmut"  
3 #define WIFI_PASSWORD "12345678"  
4  
5     //Connect Wifi and Firebase  
6 void ensureFirebaseConnection() {  
7     if (WiFi.status() != WL_CONNECTED) {  
8         Serial.println("WiFi disconnected. Reconnecting...");  
9         WiFi.begin(WIFI_SSID, WIFI_PASSWORD);  
10        while (WiFi.status() != WL_CONNECTED) {  
11            delay(500);  
12            Serial.print("...");  
13        }  
14        Serial.println("\nWiFi reconnected.");  
15    }  
16    if (!Firebase.ready()) {  
17        Serial.println("Firebase not ready. Reinitializing...");  
18        Firebase.begin(&config, &auth);  
19    }  
20}
```

**Mục đích:** Kiểm tra xem thiết bị có đang kết nối WiFi không.

**Hoạt động:**

1. Nếu WiFi không kết nối (WiFi.status() != WL\_CONNECTED):

- In ra thông báo "WiFi disconnected. Reconnecting..." trên Serial Monitor.
- Thực hiện kết nối lại WiFi bằng WiFi.begin(WIFI\_SSID, WIFI\_PASSWORD).
- Chờ cho đến khi kết nối thành công:
  - Kiểm tra trạng thái mỗi 500ms.
  - In dấu ... để hiển thị trạng thái đang chờ.
- Khi kết nối thành công, in ra "WiFi reconnected.".

#### 6.1.2.2 Xử lí Firebase

```
1 // Firebase info
2 #define FIREBASE_HOST "webs-943c5-default-rtbd.firebaseio.com"
3 #define FIREBASE_AUTH "AyUxXwk7jGS06rJ4nDyGwSEK7Pwiuwy9qxdeaNg0"
4
5 // Firebase config
6 FirebaseData firebaseData;
7 FirebaseConfig config;
8 FirebaseAuth auth;
9 FirebaseJson json;
10 String path = "/ParkingData";
```

#### Xử lý luồng Firebase

```
1 // Firebase
2 void handleFirebaseStream() {
3     if (Firebase.readStream(firebaseData)) {
4         if (firebaseData.streamTimeout()) {
5             Serial.println("Stream timeout, reconnecting...");
6             Firebase.beginStream(firebaseData, path);
7         } else if (firebaseData.streamAvailable()) {
8             Serial.println("Stream available");
9             String key = firebaseData.dataPath();
10            String value = firebaseData.stringData();
11            if (key == "/availableSlots") {
```

```
12             availableSlots = value.toInt();
13             Serial.println("Available Slots: " + String(
14                 availableSlots));
15         }
16         else if (key == "/openGate_L") {
17             gateOpen_L = value == "true";
18             Serial.println("Open Gate Left: " + String(
19                 gateOpen_L));
20         }
21         else if (key == "/openGate_R") {
22             gateOpen_R = value == "true";
23             Serial.println("Open Gate Right: " + String(
24                 gateOpen_R));
25         }
26     } else {
27         Serial.println("Error in Firebase stream: " + firebaseData.
errorReason());
28     }
29 }
```

**Mục đích** Hàm này được sử dụng để:

- Theo dõi và nhận các thay đổi dữ liệu từ Firebase theo thời gian thực.
- Xử lý các sự kiện timeout của luồng và khởi động lại luồng nếu cần.
- Cập nhật các trạng thái hoặc giá trị từ Firebase vào các biến trong chương trình.

### Hoạt động của hàm

#### 1. Kiểm tra trạng thái luồng Firebase:

- Gọi hàm `Firebase.readStream(firebaseData)` để đọc dữ liệu luồng.
- Nếu xảy ra lỗi, thông báo lỗi được in ra bằng hàm `Serial.println()`.

#### 2. Xử lý sự kiện timeout:

- Nếu luồng bị timeout (`firebaseData.streamTimeout()` trả về `true`):
  - In thông báo "Stream timeout, reconnecting...".
  - Khởi động lại luồng bằng `Firebase.beginStream(firebaseData, path)`.

### 3. Xử lý khi có dữ liệu luồng khả dụng:

- Nếu có dữ liệu mới (`firebaseData.streamAvailable()` trả về `true`):
  - Lấy đường dẫn dữ liệu bằng `firebaseData.dataPath()` và giá trị bằng `firebaseData.stringValue()`.
  - So sánh key với các giá trị cụ thể:
    - \* Nếu `key == "/availableSlots"`: Cập nhật số chỗ trống (`availableSlots`) bằng `value.toInt()` và in thông báo.
    - \* Nếu `key == "/openGate_L"`: Cập nhật trạng thái mở cổng trái (`gateOpen_L`).
    - \* Nếu `key == "/openGate_R"`: Cập nhật trạng thái mở cổng phải (`gateOpen_R`).

### 4. In lỗi: Nếu có lỗi trong luồng, thông báo lỗi được in ra qua `firebaseData.errorReason()`.

#### 6.1.2.3 Cập nhật Firebase Hàm updateFirebase

```
1 void updateFirebase(int nodeNumber, boolean gate_L, boolean gate_R,
2                     String uid_R, String uid_L, int available, int count, String
3                     carLicenseRight, boolean leStatus, boolean riStatus, String
4                     carLicenseLeft) {
5
6     FirebaseJson jsonNode;
7
8     jsonNode.set("/UID_Right", uid_R);
9     jsonNode.set("/UID_Left", uid_L);
10    jsonNode.set("/CarlicenseRight", carLicenseRight);
11    jsonNode.set("/CarlicenseLeft", carLicenseLeft);
12
13    String pathNode = "ParkingData/" + String(nodeNumber);
14
15
16    if (Firebase.set(firebaseData, pathNode, jsonNode)) {
17        Serial.println("Firebase node updated successfully at " +
18                      pathNode);
19    } else {
```

```
12         Serial.println("Error updating Firebase node: " +
13             firebaseData.errorReason());
14     }
15
16     FirebaseJson jsonStatus;
17     jsonStatus.set("/barrierLeft", gate_L);
18     jsonStatus.set("/barrierRight", gate_R);
19     jsonStatus.set("/leftStatus", leStatus);
20     jsonStatus.set("/rightStatus", riStatus);
21
22     String pathStatus = "ParkingData/Status";
23
24     if (Firebase.set(firebaseData, pathStatus, jsonStatus)) {
25         Serial.println("Firebase status updated successfully at " +
26             pathStatus);
27     } else {
28         Serial.println("Error updating Firebase status: " +
29             firebaseData.errorReason());
30     }
31
32     FirebaseJson jsonCarCount;
33     jsonCarCount.set("/vehicleCount", count);
34     jsonCarCount.set("/available slot", available);
35
36     String pathCarCount = "ParkingData/carCount";
37
38     if (Firebase.set(firebaseData, pathCarCount, jsonCarCount)) {
39         Serial.println("Firebase carCount updated successfully at " +
40             + pathCarCount);
41     } else {
42         Serial.println("Error updating carCount: " + firebaseData.
43             errorReason());
44     }
45 }
```

**Mục đích:** Hàm updateFirebase được sử dụng để cập nhật thông tin UID và trạng

thái cổng trên Firebase. Hàm này đảm bảo rằng dữ liệu Firebase được duy trì đồng bộ giữa các cổng trái, phải và cổng bổ sung.

### Hoạt động:

#### 1. Thiết lập dữ liệu JSON:

- jsonLeft: Lưu trữ UID của cổng trái (uid\_L).
- jsonRight: Lưu trữ UID của cổng phải (uid\_R).
- jsonRight2: Lưu trữ UID của cổng phải bổ sung (uid\_R2).

#### 2. Xác định đường dẫn: Tạo các đường dẫn dữ liệu trên Firebase:

- pathLeft: Đường dẫn lưu thông tin cổng trái.
- pathRight: Đường dẫn lưu thông tin cổng phải.
- pathRight2: Đường dẫn lưu thông tin cổng phải bổ sung.

#### 3. Kiểm tra và cập nhật cổng phải:

- Kiểm tra xem UID tại pathRight có tồn tại không:
  - Nếu tồn tại, kiểm tra tiếp UID tại pathRight2.
  - Nếu pathRight2 chưa có dữ liệu, cập nhật UID từ jsonRight2.
- Nếu pathRight chưa có dữ liệu, cập nhật UID từ jsonRight.

#### 4. Cập nhật cổng trái: Cập nhật UID của cổng trái từ jsonLeft.

##### 6.1.3 Các hàm phụ

###### 6.3.1 Xử lý quét thẻ RFID

```
1 void handleCard(MFRC522 &reader, Servo &servo, int ledPin, bool &
2   gateState, const char *side) {
3   String uid = "";
4   for (byte i = 0; i < reader.uid.size; i++) {
5     if (reader.uid.uidByte[i] < 0x10) uid += "0";
```

```
5         uid += String(reader.uid.uidByte[i], HEX);
6     }
7     uid.toUpperCase();
8     Serial.print("UID (");
9     Serial.print(side);
10    Serial.print(": ");
11    Serial.println(uid);
12
13    int nodeToUse = findEmptyNode();
14    String carLicense = "";
15    if (strcmp(side, "Right") == 0) {
16
17        if (nodeToUse == -1) {
18            currentNode++;
19            nodeToUse = currentNode;
20        }
21
22        if (uid != lastUid_R || available1 == 4) {
23            uid_R = uid;
24            lastUid_R = uid;
25            String carLicenseRight = "";
26            rightUidList.push_back(uid);
27            if (vehicleCount < totalSlots && available1 > 0) {
28                rightStatus = true;
29                updateFirebase(nodeToUse, gateOpen_L, gateOpen_R,
30                               uid_R, uid_L, available1, vehicleCount,
31                               carLicense, leftStatus, rightStatus,
32                               carlicenseLeft);
33                delay(1000);
34                rightStatus = false;
35                updateFirebase(nodeToUse, gateOpen_L, gateOpen_R,
36                               uid_R, uid_L, available1, vehicleCount,
37                               carLicense, leftStatus, rightStatus,
38                               carlicenseLeft);
39                delay(10000);
40                String nodePath = "ParkingData/" + String(nodeToUse
41                                              ) + "/CarlicenseRight";
```





```
62     leftStatus = true;
63     uid_L = uid;
64
65     for (size_t i = 0; i < rightUidList.size(); i++) {
66         if (rightUidList[i] == uid) {
67             nodeToUse = i + 1;
68             updateFirebase(nodeToUse, gateOpen_L, gateOpen_R, uid_R
69                           , uid_L, available1, vehicleCount, carLicense,
70                           leftStatus, rightStatus, carlicenseLeft);
71             delay(1000);
72             leftStatus = false;
73             updateFirebase(nodeToUse, gateOpen_L, gateOpen_R, uid_R
74                           , uid_L, available1, vehicleCount, carLicense,
75                           leftStatus, rightStatus, carlicenseLeft);
76
77             delay(10000);
78             String pathNode = "ParkingData/" + String(nodeToUse) +
79                           "/CarlicenseLeft";
80             String carLicenseLeft = "";
81             if (Firebase.getString(firebaseData, pathNode)) {
82                 carLicenseLeft = firebaseData.stringValue();
83             } else {
84                 continue;
85             }
86             if (carLicenseLeft != "") {
87
88                 if (vehicleCount > 0 && available1 <= totalSlots) {
89                     available1++;
90                     vehicleCount--;
91                 }
92                 gateOpen_L = true;
93                 updateFirebase(nodeToUse, gateOpen_L, gateOpen_R,
94                               rightUidList[i], uid_L, available1, vehicleCount
95                               , carlicenseRight, leftStatus, rightStatus,
96                               carLicenseLeft);
97                 openGate(servo, ledPin, gateState);
98             }
99         }
100     }
101 }
```

```
90             rightUidList[i] = "DELETED";
91             uid_L = "DELETED";
92             delay(7000);
93             gateOpen_L = false;
94
95             updateFirebase(nodeToUse, gateOpen_L, gateOpen_R,
96                             rightUidList[i], uid_L, available1, vehicleCount
97                             , carlicenseRight, leftStatus, rightStatus,
98                             carLicenseLeft);
99             closeGate(servo, gateState);
100            delay(2000);
101            deleteNodeFromFirebase(nodeToUse);
102            break;
103        } else {
104    }
105
106    reader.PICC_HaltA();
107 }
```

**Mục đích:** Hàm handleCard được sử dụng để xử lý quá trình quét thẻ RFID, đọc UID từ thẻ, hiển thị UID trên Serial Monitor, và lưu UID tương ứng vào biến phù hợp dựa trên vị trí cổng.

**Hoạt động:**

1. **Khởi tạo biến UID:** Tạo chuỗi uid để lưu giá trị UID từ thẻ.
2. **Duyệt qua UID của thẻ:** Sử dụng vòng lặp để đọc từng byte trong UID:
  - Nếu giá trị byte nhỏ hơn 0x10, thêm ký tự "0" để đảm bảo luôn có 2 chữ số.
  - Chuyển đổi giá trị byte sang chuỗi số hệ thập lục phân (HEX) và nối vào chuỗi uid.

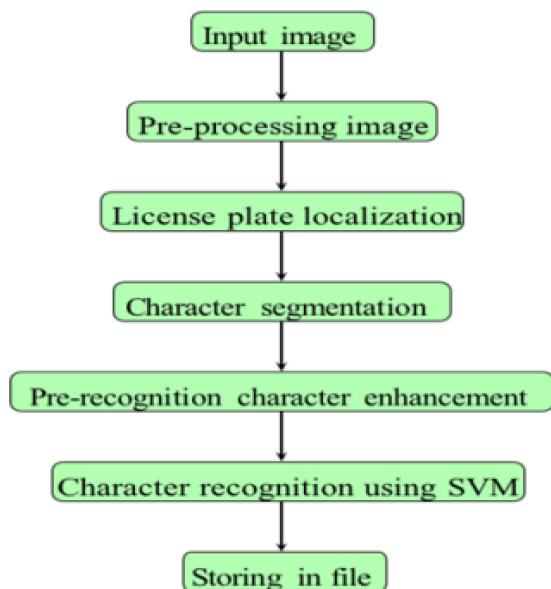


- 
3. **Chuyển UID thành chữ in hoa:** Sử dụng hàm `toUpperCase()` để chuyển chuỗi UID thành dạng in hoa.
  4. **Hiển thị UID:** In UID vừa đọc được cùng với tên cổng (`side`) ra Serial Monitor.
  5. **Lưu UID vào biến tương ứng:** Kiểm tra giá trị `side`:
    - Nếu `side` là "Right", lưu uid vào `uid_R`.
    - Nếu `side` là "Left", lưu uid vào `uid_L`.
    - Nếu `side` là "Right2", lưu uid vào `uid_R2`.

## 6.2 Nhận diện số biển số xe bằng phương pháp vector hõ trợ (SVM)

### 6.2.1 Phát triển thuật toán cho việc nhận diện số biển số xe

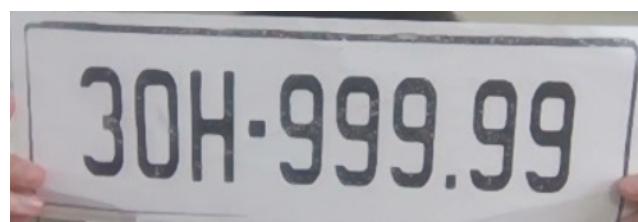
Trong nghiên cứu này, quá trình phát triển bao gồm bảy giai đoạn như được minh họa trong hình :



**Hình 25:** Lưu đồ quá trình nhận diện biển số xe

#### a. Tiền xử lý

Hình ảnh ban đầu:



**Hình 26:** Hình ảnh ban đầu

Sau đó, chuyển ảnh thành màu xám và nhị phân hóa được thực hiện bằng lệnh:

```
# Chuyển ảnh biển số về gray
```

```
gray = cv2.cvtColor(LpImg[0], cv2.COLOR_BGR2GRAY)
```

```
# Áp dụng threshold để phân tách số và nền
```

```
binary = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV)[1]
```

Hình ảnh nhị phân được minh họa trong hình



**Hình 27:** Hình ảnh nhị phân

Trong giai đoạn này, hình ảnh đầu vào của biển số xe sẽ được chuyển đổi thành ảnh mức xám. Hình ảnh sẽ được xử lý để giảm nhiễu, loại bỏ hiệu ứng bóng và loại bỏ các vùng có độ tương phản thấp trên hình ảnh một cách thích ứng. Quá trình này được gọi là tiền xử lý hình ảnh.

### b. Định vị biển số xe và phân đoạn ký tự

Giới hạn kích thước để giảm thời gian xử lý và đảm bảo tính hiệu quả của thuật toán. Kích thước của mỗi ký tự được tính bằng:

```
Dmax = 1000: //Chiều lớn nhất của ảnh sau khi xử lý không vượt quá 1000 pixels.
```

```
Dmin = 200: //Chiều nhỏ nhất của ảnh sẽ không dưới 200 pixels.
```

```
# Lấy tỷ lệ kích thước ảnh cần xử lý
```

```
rate = float(max(Ivehicle.shape[:2])) / min(Ivehicle.shape[:2])
```

```
edge_size = int(rate * Dmin)
```

```
Res_size = min(edge_size, Dmax)
```

Hộp giới hạn với màu xanh lá cây được sử dụng để phân đoạn ký tự. Lệnh sử dụng là:

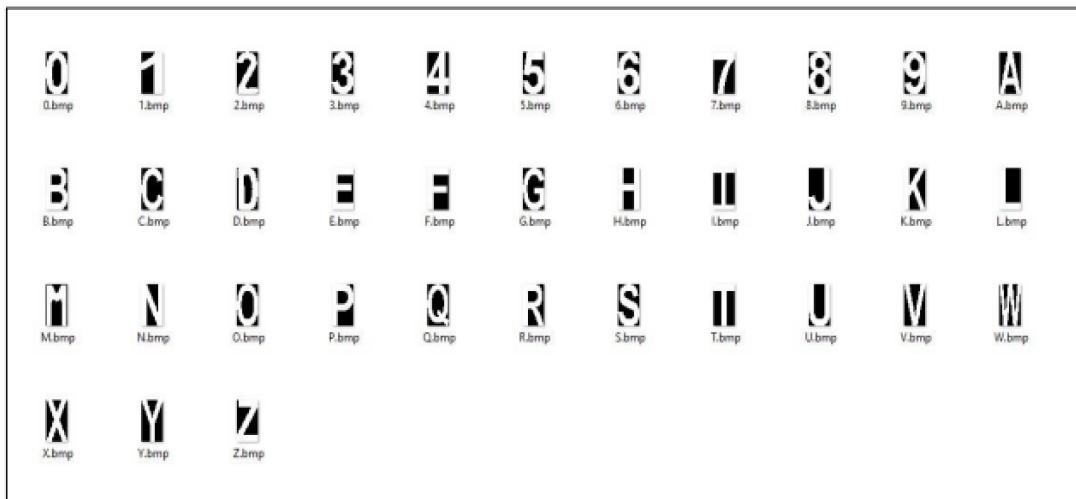
```
cv2.rectangle(Contour_Img, (x, y), (x + w, y + h), (0, 255, 0), 2)  
end
```



Hình 28: Lưu đồ quá trình nhận diện biển số xe

### c. Nhận dạng ký tự sử dụng SVM

Trong giai đoạn này, mỗi ký tự đã được cắt và nhị phân hóa sẽ được phân loại bằng 36 SVM đã được huấn luyện trước. 36 SVM đã được huấn luyện trước là một tập dữ liệu huấn luyện thuộc các lớp khác nhau, bao gồm chữ cái và số. Chữ cái bao gồm 26 chữ cái in hoa, trong khi số bao gồm 10 chữ số như được minh họa trong hình:



Hình 29: Hình ảnh của 36 SVM đã được huấn luyện

Mỗi ký tự sau khi huấn luyện sẽ được phân loại vào một trong các lớp đã được định nghĩa trước bằng cách kiểm tra vị trí và xác định lớp nào phù hợp nhất với từng ký tự. Quá trình huấn luyện và phân loại được thực hiện bằng cách sử dụng bộ phân loại SVM với kernel tuyến tính. Phân loại nhị phân của ký tự được sử dụng trong dạng cơ bản của SVM. Dữ liệu huấn luyện của ký tự sẽ được sử dụng trong phép tính SVM. Phép tính SVM được thực hiện như sau [?]:

$$y = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = w_0 + \sum_{n=1}^m w_nx_n$$

Với:

- $m$  = số lượng huấn luyện tổng cộng của biển số xe.
- $w$  = biển số xe đầu vào.
- $x$  = số lượng huấn luyện của biển số xe.
- $n$  = thứ tự đầu vào.
- $T$  = tổng số thứ tự.
- $X$  = tổng số lượng huấn luyện của biển số xe.

Xét 3 siêu phẳng:

$$w_0 + w^T X = -1$$

$$w_0 + w^T X = 0$$

$$w_0 + w^T X = 1$$

Nếu ảnh là:

$$(w_0 + w^T X) \geq 1 - \alpha_n$$

với  $\alpha$  là tham số. Ảnh được phân loại chính xác khi  $\alpha_n = 0$ , trong khi nếu  $\alpha_n > 0$  thì nó bị phân loại sai. Sai số trung bình được tính bằng:

$$\frac{1}{n} \sum_n \alpha_n$$

và có thể được mô tả với phương trình bổ sung như sau:

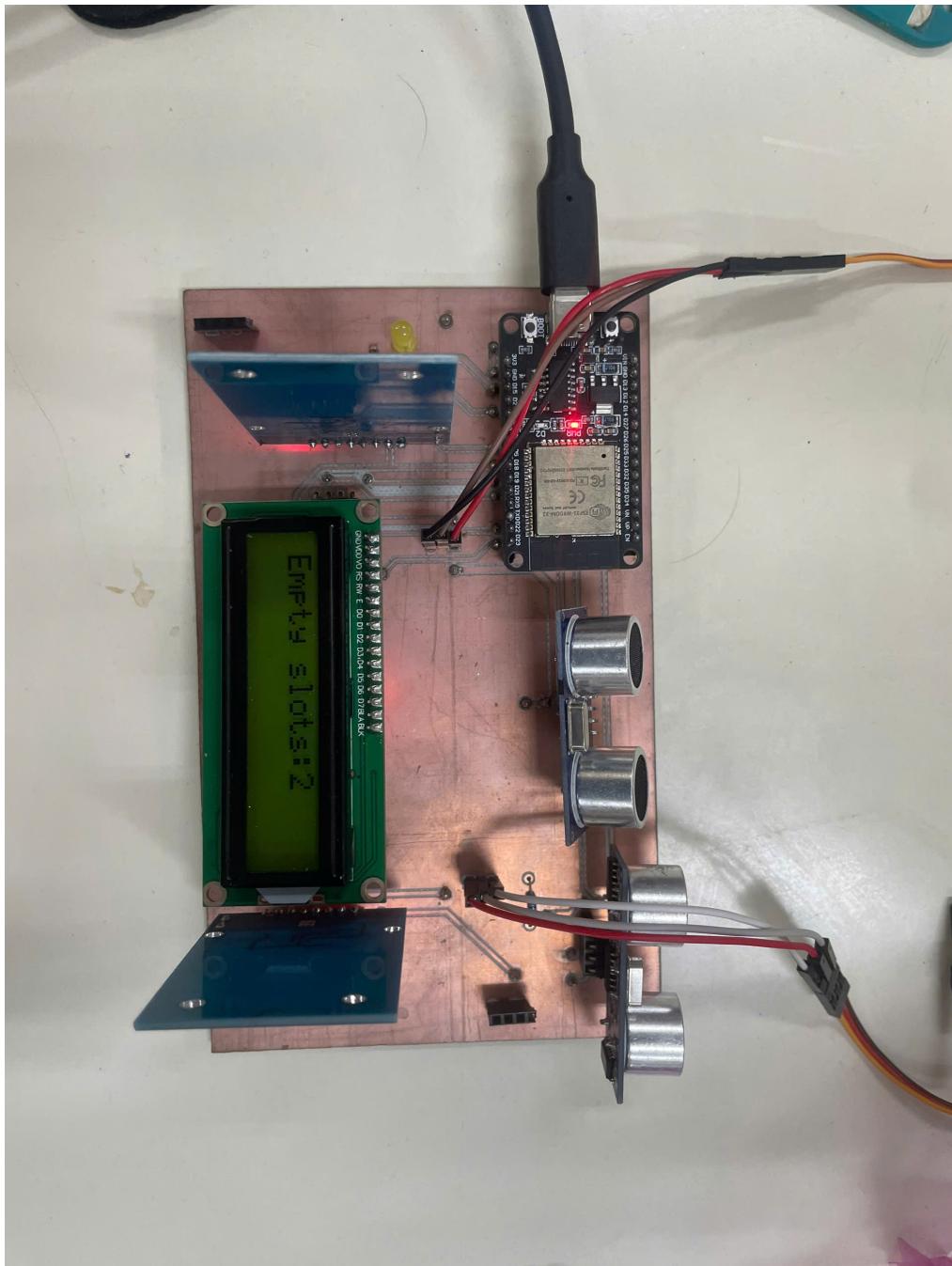
$$\min |\vec{w}|^2 + \frac{1}{n} \sum \alpha_n$$

Với:

$$(w_0 + w^T X) \geq 1 - \alpha_n$$

## 7 Mô phỏng

### 7.1 Phần Cứng





## 7.2 Web App và Mã nguồn

Ứng dụng web bãi giữ xe thông minh được triển khai trên nền tảng **Firebase Hosting**. Chúng ta có thể truy cập ứng dụng thông qua liên kết dưới đây:

<https://parkingsystem-9d434.web.app>

[Source code](#)