Inidividual Practical Assignment – Smart Parking System

Duong Ha Tien Le - 104700948

1. Summary

1.1 Topic Background

Managing parking effectively has become a growing concern in modern urban environments, where the rise in vehicle numbers continues to outpace the development of supporting infrastructure. As cities become more crowded, drivers face longer wait times and increased frustration due to the limited availability of parking spaces. This inefficiency not only causes delays but also adds to traffic congestion and environmental impact, as vehicles emit more pollutants while idling or searching for spots [1]. Conventional parking systems often rely on manual processes and lack the ability to provide real-time updates or automated slot management. Without access to live data or guided entry, users must navigate parking areas blindly, increasing the chances of unauthorized parking, slot misuse, and inefficient space allocation [2]. These problems are especially noticeable during busy periods, where delays and confusion become more frequent.

While some smart parking solutions have been introduced in recent years, many of them fail to deliver reliable performance due to limited automation, poor integration with user-facing applications, or an inability to scale efficiently [2]. Without a well-connected system that links hardware, software, and user interactions, such implementations fall short of solving the core issues. This individual project addresses these limitations by designing a smart indoor parking system that implements Internet of Things (IoT) technologies to enhance both usability and control. The system brings together sensors, a camera-equipped microcontroller, a cloud-based license plate recognition API, a local database, and a web interface to create a fully integrated solution. Vehicles are detected automatically at the entrance, license plates are captured and verified, and the system opens the gate and guides the user to the correct slot using LED indicators. All parking activity is logged and monitored in real time, and users can view slot availability, bookings, and system logs through an interactive dashboard. By automating key processes and connecting each component through a shared IoT framework, the system improves accuracy, reduces the need for manual oversight, and creates a more efficient and user-friendly parking experience. This approach contributes to smarter urban mobility and offers a scalable model for future smart city infrastructure.

1.2 Overview of Proposed System

The proposed smart parking system consists of three core layers: the hardware layer powered by the ESP32-S3-WROOM microcontroller, a Python-based edge server, and a web-based user interface. Together, these layers automate parking management, enforce access control, and display real-time system information to users.

a. Physical layer

The hardware layer uses a FREENOVE ESP32-S3-WROOM microcontroller to manage all input and output components. The reason why this microcontroller has been chosen is its support for WIFI connectivity and onboard camera integration. Three infrared (IR) sensors are connected to detect vehicle presence, one positioned at the entrance gate and one at each of the two parking slots. The entrance IR sensor plays a key role in triggering the license plate recognition process; when it detects a vehicle, the ESP32-S3 captures an image of the car's license plate using its attached camera module. The slot IR sensors continuously monitor whether a car is present in each bay, enabling the system to detect correct or incorrect parking and update the slot occupancy status accordingly.

In addition to these digital sensors, a potentiometer is integrated into the system to allow manual adjustment of the gate open duration. The ESP32-S3 reads its analog input and maps the value to a time interval, which determines how long the gate remains open after access is granted. This provides flexibility during setup or testing without requiring changes to the firmware.

The ESP32-S3 sends the captured image to an external cloud API via HTTPS for license plate recognition. Once the plate number is returned, the microcontroller communicates with the edge server over a USB serial connection to relay the result. Based on commands received from the server, the microcontroller triggers various actuators:

- Servo motor: Opens or closes the entrance gate.
- Red and green LEDs: Indicate slot status (booked, available and occupied).
- Buzzer: Activates when a vehicle parks in the wrong slot or when the camera successfully captures the plate number on a car.

b. Edge Device (Python Flask Server)

The edge computing layer is implemented using a Python script running on a Mac terminal as the local server. This script listens to incoming serial data from the ESP32-S3 and performs real-time processing based on the received messages. Once the ESP32-S3 transmits a detected license plate number to the edge server, the Python script stores this plate number in the local MariaDB database for record-keeping and tracking purposes.

```python
# This function stores detected plates in the database
def store_detected_plate(plate_number, slot_assigned=None):
    """Store a detected license plate in the database"""
    try:
        conn = pymysql.connect(**DB_CONFIG)
        cursor = conn.cursor()

        timestamp = datetime.now()
        status = 'assigned' if slot_assigned else 'detected'

        cursor.execute(
            "INSERT INTO plates (plate_number, timestamp, slot_assigned, status) VALUES (%s, %s, %s, %s)",
            (plate_number, timestamp, slot_assigned, status)
        )

        conn.commit()
        return True
    except Exception as e:
        logger.error(f"Failed to store detected plate: {str(e)}")
        return False
    finally:
        if conn:
            conn.close()
```

Figure 1. Python function for storing detected license plates in the MariaDB database with timestamp and status.

After logging the entry, the script performs a database query to determine whether the detected plate matches any active booking in the system. If a match is found, the server considers the vehicle authorized and proceeds to send a command (OPEN_GATE) to the microcontroller to open the gate and update the slot status accordingly. If no match exists, the script checks for any unbooked available slots and either assigns one or alerts the user, depending on the system configuration.

```python
#THIS IS REALLY IMPORTANT PART: this function checks if a plate has a reservation and returns the slot if found
def check_plate_reservation(plate_number):
    """Check if a plate has a reservation and return slot if found"""
    conn = None

    try:
        conn = pymysql.connect(**DB_CONFIG)
        cursor = conn.cursor(pymysql.cursors.DictCursor)

        cursor.execute(
            "SELECT id FROM slots WHERE status = 'booked' AND plate = %s",
            (plate_number,)
        )
        result = cursor.fetchone()

        if result:
            return result['id']
        return None
    except Exception as e:
        logger.error(f"Error checking plate reservation: {str(e)}")
        return None
    finally:
        if conn:
            conn.close()
```

Figure 2. Python function to check if a detected plate number matches a booked slot in the database.

Depending on the verification result, the server sends corresponding commands to the ESP32-S3, such as:

- Opening the gate (OPEN_GATE)

- Activating the buzzer for wrong-slot parking (WRONG_SLOT)
- Toggling slot LEDs based on new occupancy

In addition to access control, the server performs basic analytics using the database records. It tracks unauthorized parking attempts, measures how often each slot is used, and logs these events for display on the web dashboard.

```
MariaDB [parking_system1]> select * from slots;
+----+--------+-----------+
| id | status | plate     |
+----+--------+-----------+
|  1 | booked | AP39S8889 |
|  2 | booked | KL65H4383 |
+----+--------+-----------+
2 rows in set (0.009 sec)
```

Figure 3. Output from the MariaDB "slots" table showing current booking and associated license plates.

```
MariaDB [parking_system1]> select * from plates;
+-----+--------------+---------------------+---------------+----------+
| id  | plate_number | timestamp           | slot_assigned | status   |
+-----+--------------+---------------------+---------------+----------+
|  41 | AP3P9S8889   | 2025-05-06 20:52:44 |             1 | assigned |
|  42 | APIP39S8889  | 2025-05-06 21:09:54 |             1 | assigned |
|  43 | AP39S8889    | 2025-05-06 21:14:59 |             1 | assigned |
|  44 | AP39S8889    | 2025-05-06 21:16:46 |             2 | assigned |
```

Figure 4. Output from the MariaDB "plates" table showing detected license plate number with assigned slot.

c. Web Dashboard (HTML, CSS, Javascript)

The system includes a responsive web dashboard built using HTML, CSS, and JavaScript. This dashboard allows users to:

- Book or cancel parking slots by entering a license plate number

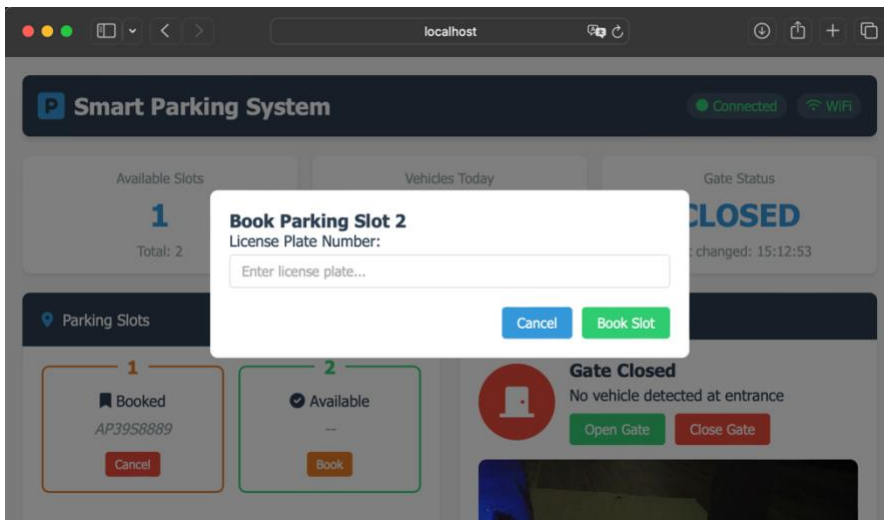


Figure 5. Web dashboard interface allowing users to book Parking Slot 2 by entering a license plate number.

- Monitor current slot status and gate state

Figure 6. Gate control section of dashboard showing live gate status and manual options to open or close the gate, with a real-time camera view of the entrance

- Provide manual controls through the dashboard to test actuators and interact directly with the system.

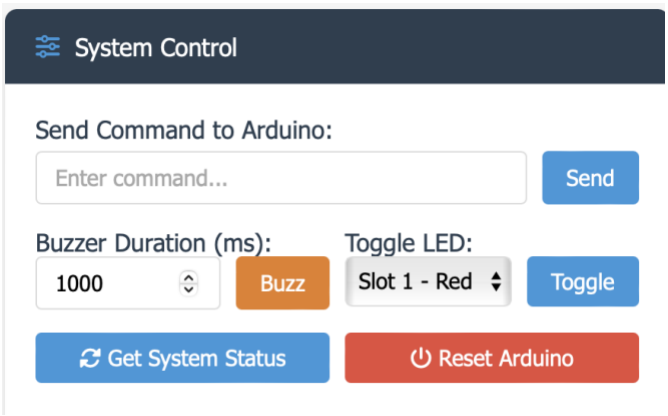


Figure 7. System control panel for sending commands, toggling LEDS, activating the buzzer

- View recent events such as vehicle entries, detected plates, and warnings

The interface communicates with the Python backend using HTTP endpoints. Data is displayed in real time, and visual elements such as status indicators, toast notifications, and logs help users quickly understand the current state of the system.

d. Integration of License Plate Recognition API

This project integrates an external cloud-based license plate recognition API to perform automatic vehicle identification. The implementation takes inspiration from the tutorial "How to use ESP32 CAM for Automatic Number Plate Recognition (ANPR)" published by Circuit Digest [3], which outlines the core steps for capturing images on an ESP32 and sending them to a remote server for processing.

In this project, the ESP32-S3-WROOM captures an image of the vehicle's license plate and sends it to the Circuit Digest API via a secure HTTPS POST request. The request includes the captured image and an API key for authentication. The API processes the image using optical character recognition (OCR) and returns a response in JSON format, containing the detected plate number. The firmware uses this response to extract the license plate number, which is then passed on to the edge server for further logic. This process eliminates the need for on-device OCR, allowing the microcontroller to remain lightweight while leveraging powerful cloud-based image processing. While the project follows the general method provided in the original guide, several aspects were modified to fit the needs of a real-time, multi-slot smart parking system. These changes include adapting the code for the ESP32-S3 board, restructuring the image upload process to fit the system's timing and control flow, and integrating the API response into a broader parking slot assignment workflow. By reusing the core API integration approach and enhancing it with custom logic and additional components, this project successfully applies and extends an existing solution for use in a practical IoT environment.

2. Conceptual Design

2.1 Block diagram: IoT System Architecture

The block diagram illustrates the complete system architecture of the smart parking system, highlighting the interaction between the hardware, software, and cloud-based license plate recognition service. The system is divided into two main sections: the hardware layer and the software layer, with communication handled through serial and HTTP protocols.

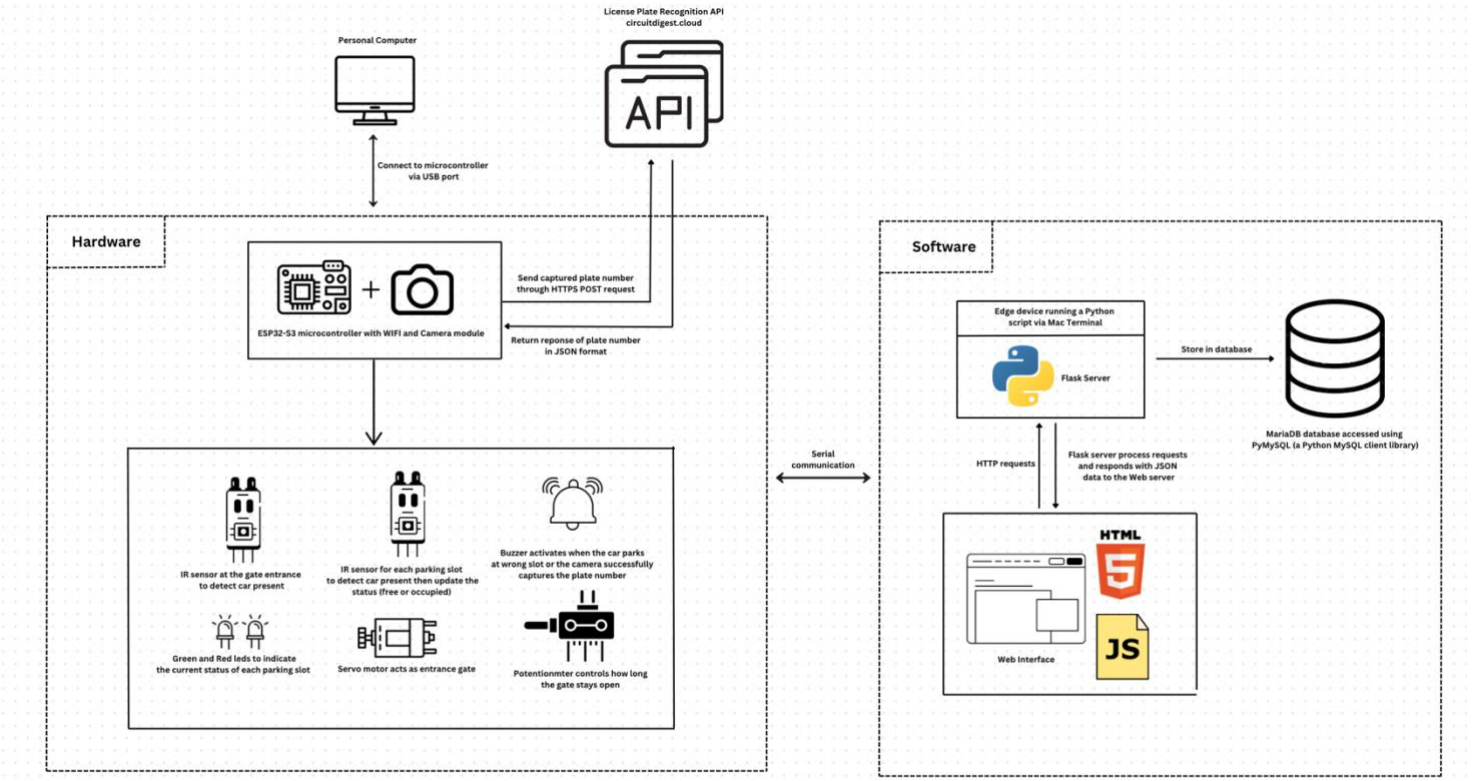


Figure 8. Block diagram of proposed system

The system is centered around the ESP32-S3 microcontroller, which integrates Wi-Fi and a camera module to manage sensor inputs, actuator control, and image-based license plate recognition. When a vehicle is detected by the entrance IR sensor, the ESP32-S3 captures a photo and sends it via HTTPS to an external API for processing. The API returns the recognized plate number in JSON format, which the ESP32-S3 forwards to the edge server via serial communication. On the software side, the edge server runs a Python script within a Mac terminal environment, using the Flask framework to process data and serve the web interface. The server communicates with a MariaDB database, accessed through the PyMySQL library, to log detected plates, manage bookings, and update slot status. The web interface, built with HTML, CSS, and JavaScript, interacts with the Flask server through HTTP requests to display real-time information, allow user bookings, and issue manual control commands. This integration of hardware, cloud services, and local edge logic enables a fully automated and interactive parking system that provides real-time monitoring, secure access control, and efficient space management.

2.2 UML diagram

The activity diagram below illustrates the complete operational workflow of an ESP32-based Smart Parking System on hardware side. It begins with system initialization, including GPIO configuration and Wi-Fi connection, followed by camera setup and slot status requests from the edge server. The main loop continuously monitors serial input and checks sensor triggers. When a vehicle approaches, the system captures an image, sends it to a license plate API, and verifies booking. If matched, the gate opens and the assigned slot is indicated via LEDs. The system detects incorrect parking through IR sensors and triggers the buzzer if needed. It also handles slot updates, gate control, and fallback mechanisms in case of camera, Wi-Fi, or API failures, ensuring robust, automated parking management.

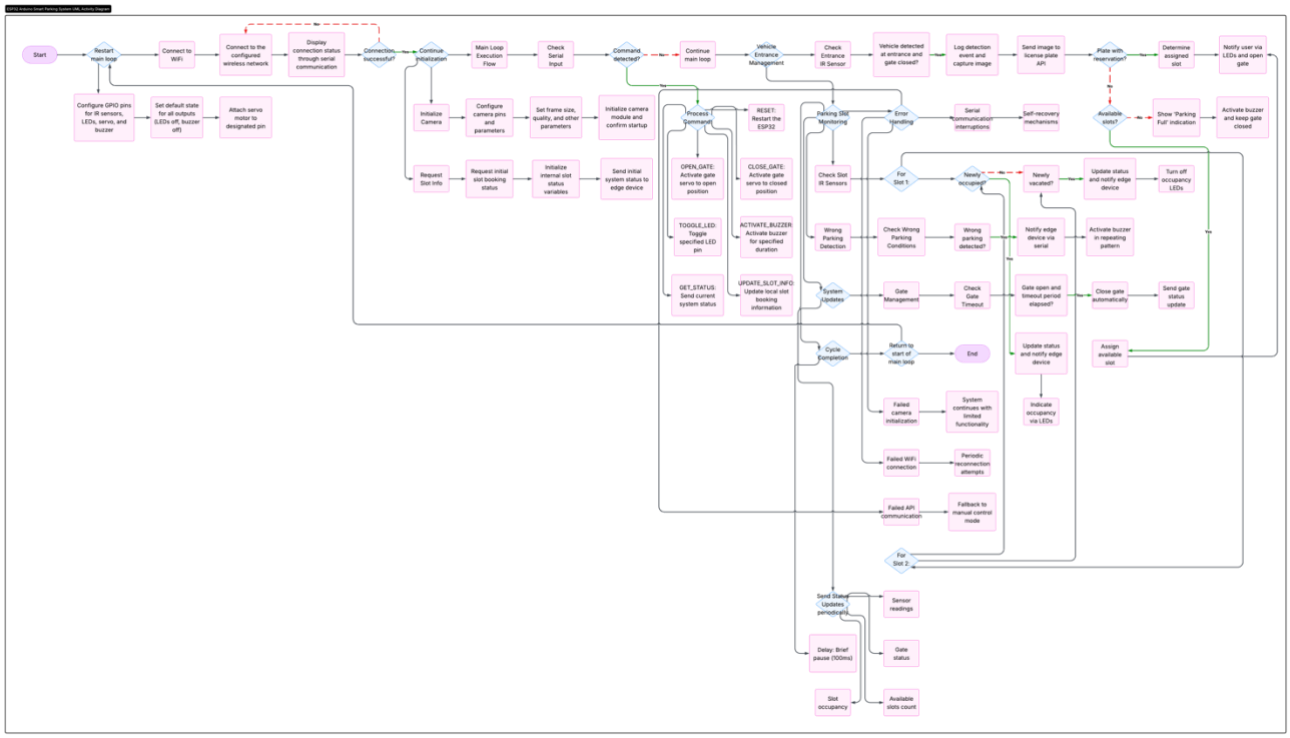


Figure 9. Flowchart of hardware implementation

The activity diagram below illustrates the software-side workflow of the Smart Parking System implemented on the edge device running on Mac Terminal. The process begins with executing a Python script, which launches the Flask web server and initializes a USB serial connection to the ESP32-S3. Once data is received, typically a detected license plate, the system stores it in a MariaDB database and checks for a reservation match. If a match is found, the gate is authorized to open and the slot status is updated. If no reservation exists but a free slot is available, it is assigned and the ESP32 is notified. Otherwise, a "PARKING-FULL" message is sent, and a buzzer alert is triggered. Simultaneously, the web server handles booking or cancellation requests, updates the database and dashboard, and returns real-time status updates via JSON responses.

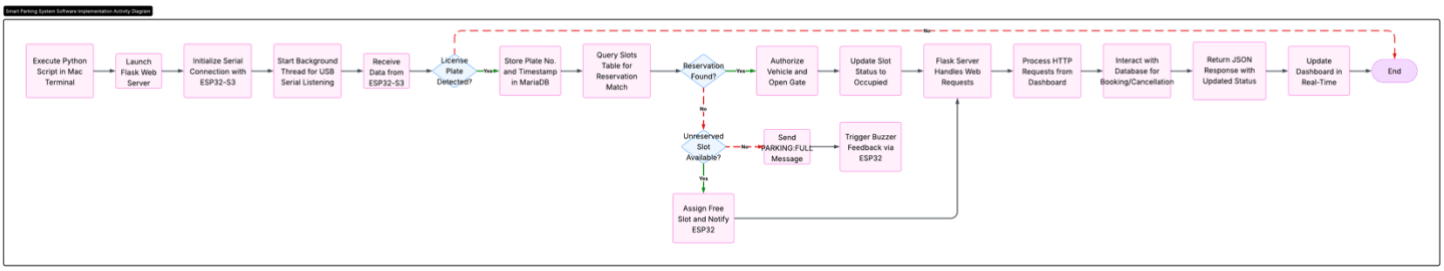


Figure 10. Flowchart of software implementation

3. Implementation

3.1 Sensors

This project integrates a combination of digital and analog sensors to monitor the physical environment and trigger automated system responses in real time.

a. Digital Sensors – Infrared (IR) Obstacle Sensors

The system uses three infrared (IR) sensors, each functioning as a digital input device: One IR sensor is placed at the entrance gate to detect the arrival of a vehicle. The other two IR sensors are installed at Parking Slot 1 and Parking Slot 2 to determine if

a car is present in each slot. Each IR sensor outputs either HIGH or LOW depending on object proximity. These digital signals are read by the ESP32-S3 using digitalRead() and used to trigger specific behaviors such as:

- Initiating license plate capture when a car is detected at the gate.

```
// Check entrance sensor for vehicle detection
if (digitalRead(gateIrSensorPin) == LOW && !isGateOpen) {
 Serial.println("VEHICLE:DETECTED_AT_ENTRANCE");
 Serial.println("Reading plate...");

 // Capture and recognize license plate
 detectedPlate = captureAndRecognizePlate();
```

Figure 11. Arduino code snippet showing how the entrance IR sensor triggers license plate recognition when a vehicle is detected.

- Logging a slot as occupied when a car is parked.

```
// Check parking slots occupancy via IR sensors
static bool lastSlot1Occupied = false;
bool slot1Occupied = (digitalRead(slot1IrSensorPin) == LOW);
slots[0].isOccupied = slot1Occupied;

// Handle slot 1 status changes
if (slot1Occupied != lastSlot1Occupied) {
  if (slot1Occupied) {
    //Car has arrived in slot 1
    Serial.println("SLOT1:OCCUPIED:" + detectedPlate);
```

Figure 12. Arduino code snippet showing how the slot IR sensor detects vehicle presence and logs the corresponding license plate upon occupancy.

- Detecting wrong-slot parking, which triggers the buzzer.

```
// Function to check for wrong parking
bool checkWrongParking(int slotIndex) {
  // If IR sensor detects a car but the slot is booked for another plate
  bool isOccupied = (digitalRead(slotIndex == 0 ? slot1IrSensorPin : slot2IrSensorPin) == LOW);

  // If a slot is occupied and booked, but not for the current car
  if (isOccupied) {
    if (slots[slotIndex].isBooked && detectedPlate != slots[slotIndex].bookedPlate) {
      // This slot is booked but not by the current car
      return true;
    }
  }
```

Figure 13. Arduino code snippet showing how the system checks for wrong-slot parking by comparing the detected license plate with the booked plate for each slot.

The IR sensors are integrated via GPIO digital pins on the ESP32-S3, and their logic is embedded within the Arduino sketch (smartparking.ino).

b. Analog Sensor - Potentionmeter

The system includes a rotary potentiometer, which functions as an analog input device used to manually adjust the gate's open duration. A potentiometer is a variable resistor that outputs a voltage corresponding to its rotational position. This analog voltage is read by the ESP32-S3 through an analog input pin using the analogRead() function. In this project, the potentiometer plays an important role in giving users or developers real-time control over how long the entrance gate remains open after it has been triggered. The ESP32 reads the raw analog value (ranging from 0 to 4095 due to its 12-bit ADC resolution) and maps it to

```
// Function to open the gate
void openGate() {
  gateServo.write(0); // Position for open gate
  isGateOpen = true;

  // Read potentiometer to set timeout duration
  int potValue = analogRead(gateTimeoutPotPin);
  // Map to a reasonable range
  unsigned long timeoutDuration = map(potValue, 0, 4095, 500, 5000);

  gateOpenTime = millis();
  gateOpenDuration = timeoutDuration; // Make gateOpenDuration a variable instead of const

  Serial.print("GATE_STATUS:OPENED,TIMEOUT:");
  Serial.println(timeoutDuration);
}
```

Figure 14. Arduino code snippet showing how the gate is opened using a servo motor and how the gate open duration is dynamically set based on potentiometer input.

a time range suitable for gate control, for example, from 500 milliseconds (0.5 seconds) to 5000 milliseconds (5 seconds). This mapped duration is then used to delay the gate's closing, allowing enough time for a car to pass through.

5.2 Actuators

a. Servo Motor – Entrance Gate Control

A SG90 micro servo motor is used to physically open and close the entrance gate, acting as the primary mechanical component for access control. When a valid vehicle is detected based on license plate recognition and booking validation, the system sends a signal to the servo to rotate to the 0° position, which opens the gate. After a preset duration which is determined dynamically by the potentiometer or when commanded by the edge server, the servo returns to the 90° position, closing the gate. The servo is connected to a PWM-capable pin on the ESP32-S3 and controlled using the ESP32Servo.h library, which allows precise angle adjustment for both entry and exit operations. This setup ensures secure, automated access to the parking facility and reinforces proper entry behavior without manual intervention.

b. LEDs

Each parking slot is equipped with two LEDs: one red and one green, that provide clear visual cues to indicate the status of the slot. The red LED is turned on when the slot is booked in advance, while the green LED is turned on to indicate the user that is the right slot to park or correctly occupied by a vehicle with a valid reservation. These indicators help drivers easily identify their assigned parking slot and visually confirm whether it is occupied, available or reserved.

The LED behavior is managed through digitalWrite() functions within the ESP32-S3 firmware. Based on the current system state, such as a successful booking, license plate match or slot occupancy change, the corresponding LEDs are toggled in real time. This visual feedback mechanism significantly enhances user experience and minimizes parking errors, especially in multi-slot environments.

For example, if a user parks in the correct slot, the green LED blinks three times and stays on, reinforcing proper behavior. If a slot is booked and another vehicle attempts to park there, the system prevents confusion by maintaining the red LED and activating an alert. The LEDs are connected to digital output pins on the ESP32-S3, with current-limiting resistors in place to ensure safe operation. This simple yet effective use of low-cost hardware greatly improves the system's usability and accessibility.

c. Buzzer

A passive buzzer is integrated into the system as an audio alert mechanism to signal incorrect behavior or recognition failure. The buzzer is used primarily in the following scenarios:

- A car parks in a slot that does not match the assigned license plate
- The license plate detection fails or returns an invalid response
- A vehicle is unbooked and no slots are available

The buzzer connects to a digital GPIO pin on the ESP32-S3 and is activated using a pattern of high/low signals with short delays to produce a beeping effect. This alerts the driver to take corrective action, such as moving to the correct slot or rebooking. To reinforce correct behavior, the buzzer continues to beep at intervals as long as the vehicle remains in the wrong slot. Once the condition is resolved (e.g., the vehicle is moved or manually corrected via the web dashboard), the buzzer is deactivated.

```
// Function to activate buzzer
void activateBuzzer(int durationMs) {
  digitalWrite(buzzerPin, HIGH);
  Serial.println("BUZZER:ON");
  delay(durationMs);
  digitalWrite(buzzerPin, LOW);
  Serial.println("BUZZER:OFF");
}
```

Figure 14. Arduino code snippet showing how the buzzer is activated for a specified duration

3.3 Software and Libraries

The project integrates a variety of libraries across the Arduino (ESP32-S3) and Python environments to enable reliable communication, automation, data processing, and user interface rendering. Each library and technology plays a distinct role in ensuring the system operates seamlessly across the hardware, backend, and web layers.

a. Arduino (ESP32-S3) libraries

- **esp_camera.h**: This library enables image capture using the onboard OV2640 camera module connected to the ESP32-S3. It provides control over image resolution, quality, and frame settings, allowing the system to generate snapshots for license plate recognition.
- **WiFiClientSecure.h**: This library handles secure HTTPS communication, allowing the ESP32-S3 to send image data to the external license plate recognition API. It ensures that the data is transmitted over an encrypted channel using SSL/TLS protocols.
- **ArduinoJson.h**: After the image is processed by the API, the response is received in JSON format. This library is used to parse the JSON response and extract the recognized license plate string, which is then sent to the edge server.
- **ESP32Servo.h:** This library is responsible for controlling the SG90 servo motor that operates the entrance gate. It allows precise angle control using PWM signals and supports smooth gate movement.
- **NTPClient.h**: Retrieves real-time timestamps from a Network Time Protocol (NTP) server to ensure that plate detection events, bookings, and gate operations are accurately time-stamped. This enhances the reliability of the system's logs and database entries, especially when analyzing usage patterns.

b. Python edge server libraries

- **serial (pyserial):** Used for USB-based serial communication between the ESP32-S3 and the Python edge server. This library allows the server to receive real-time data (such as detected plate numbers) and send control commands (e.g., open gate, trigger buzzer).
- **pymysql:** This library facilitates connection and interaction with the MariaDB database, allowing the system to store, retrieve, and update records such as plate logs, slot statuses, and bookings. It enables robust SQL operations through Python.

- **Flask**: This is a lightweight web framework used to host the HTTP server that handles user requests. It serves the HTML dashboard, manages backend logic and processes API routes like /book, /cancel, and /get_slots.

4. Resources

The following resources were used throughout the development of this project to guide hardware integration, software implementation, and system architecture:

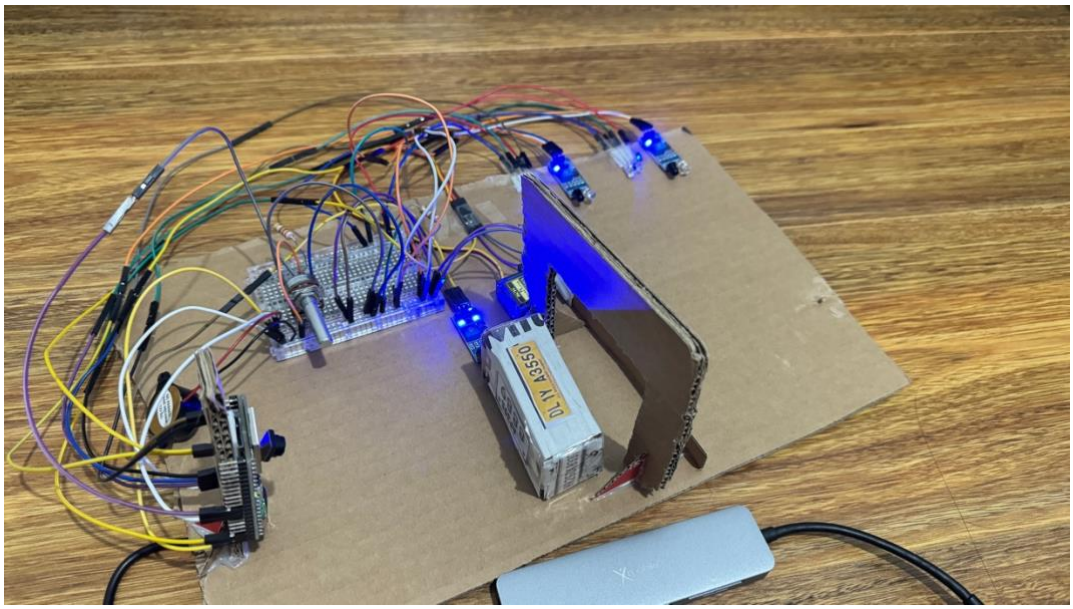
- **Circuit Digest – License Plate Recognition Using ESP32-CAM**: Used as a reference for integrating the camera module and connecting to an external OCR API service - https://circuitdigest.com/projects/license-plate-recognition-using-esp32-cam
- **Circuit Digest – AI-based Smart Parking System using ESP32-CAM**: Provided conceptual guidance on smart parking logic - https://circuitdigest.com/projects/ai-based-smart-parking-system
- **Random Nerd Tutorials – ESP32-CAM Projects and Camera Setup**: Provided guidance on configuring the camera with the ESP32-S3 module - https://randomnerdtutorials.com/getting-started-freenove-esp32-wrover-cam/
- **Flask Documentation**: Used to develop the Python-based backend server and expose HTTP endpoints - https://flask.palletsprojects.com/en/stable/
- **PyMySQL Documentation**: Referenced for connecting to and querying the MariaDB database from the Python server - https://pymysql.readthedocs.io/en/latest/

5. Video Demonstration

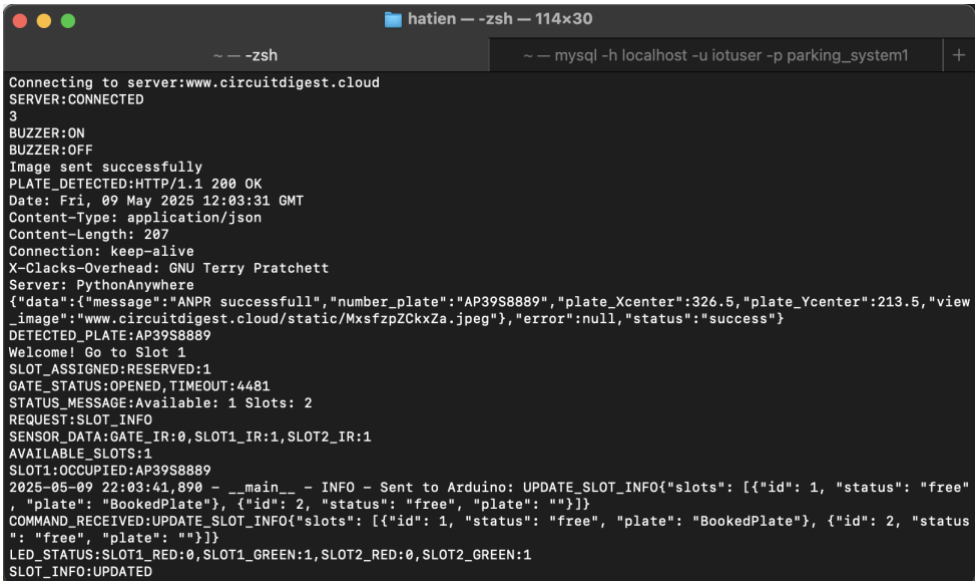
- https://www.youtube.com/watch?v=-jaxCG_dILg

6. Appendix

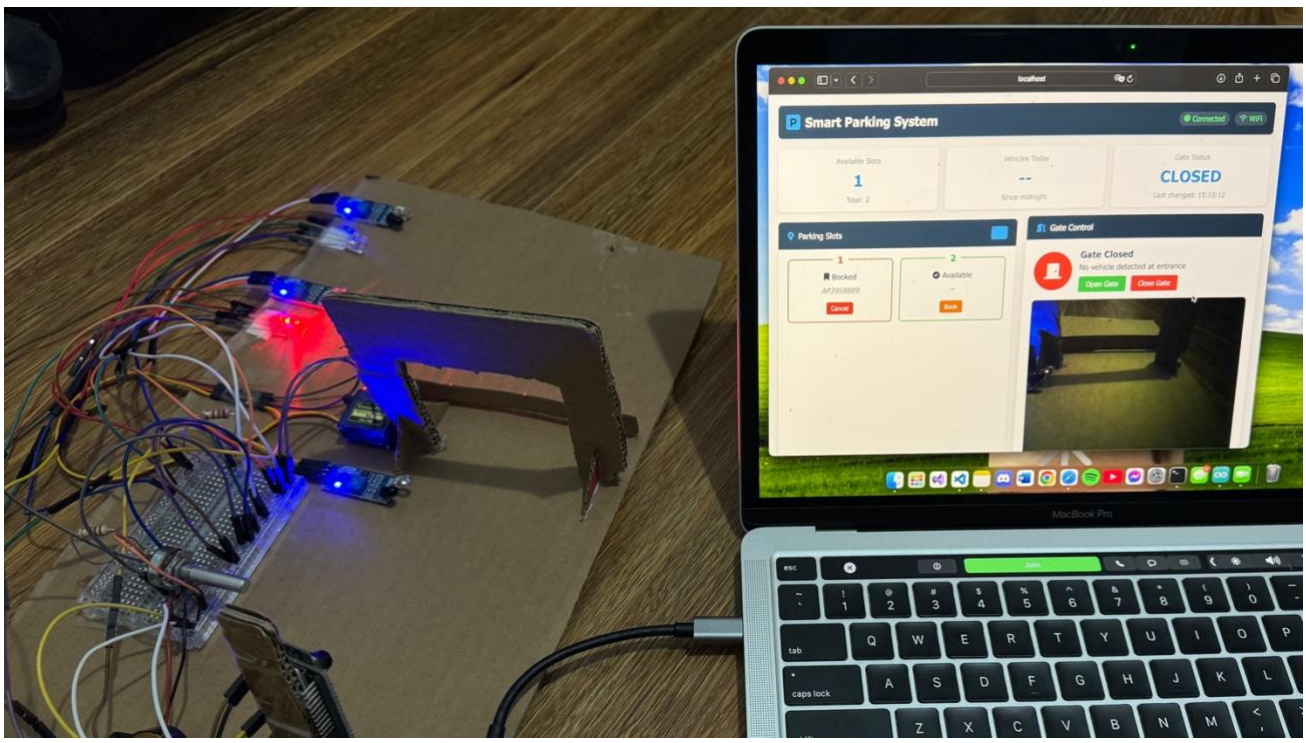
- The physical working IoT individual project – Smart Parking System Demonstrates the full hardware prototype including ESP32-S3, servo-controlled gate, IR sensors, LEDs, buzzer, and a mock vehicle used for license plate detection and automated access control.



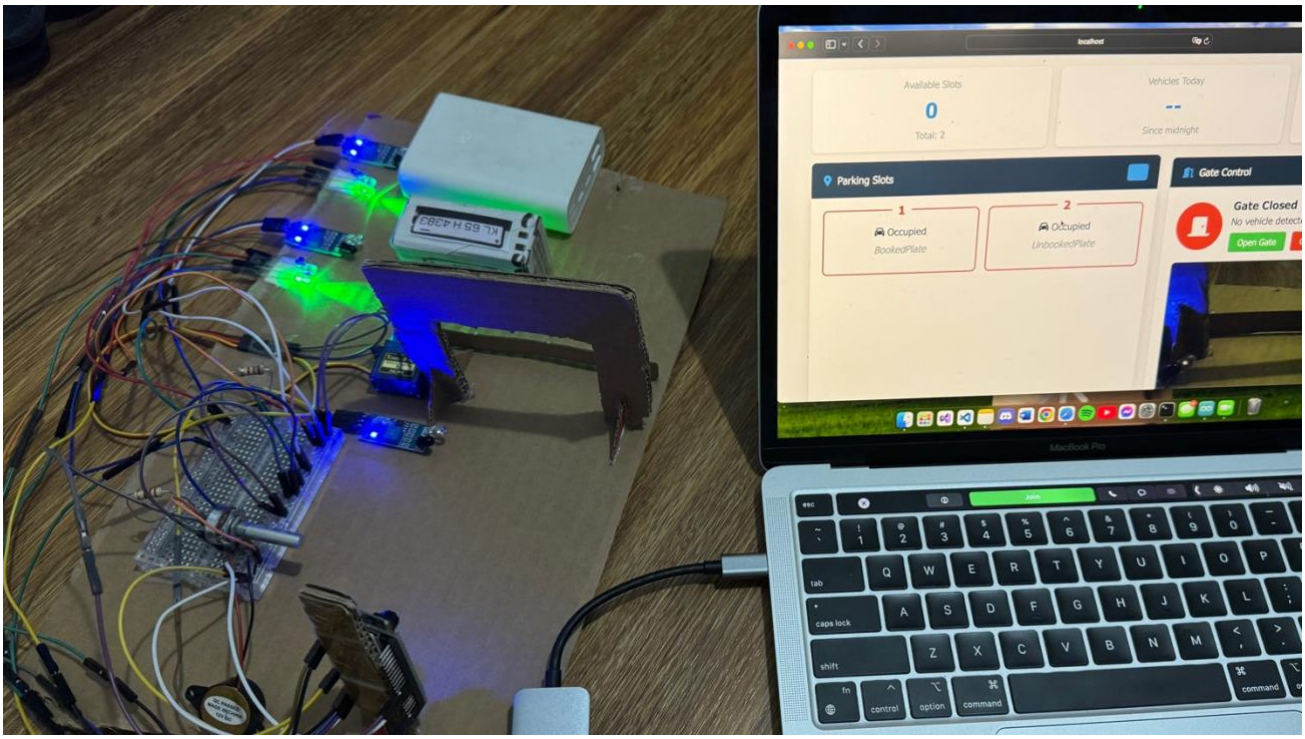
- Edge server serial output – Real-time communication between ESP32-S3 and Python Flask server: Displays live status updates, license plate detection via HTTPS API, slot assignment logic, and LED/buzzer control commands processed through the Mac terminal.



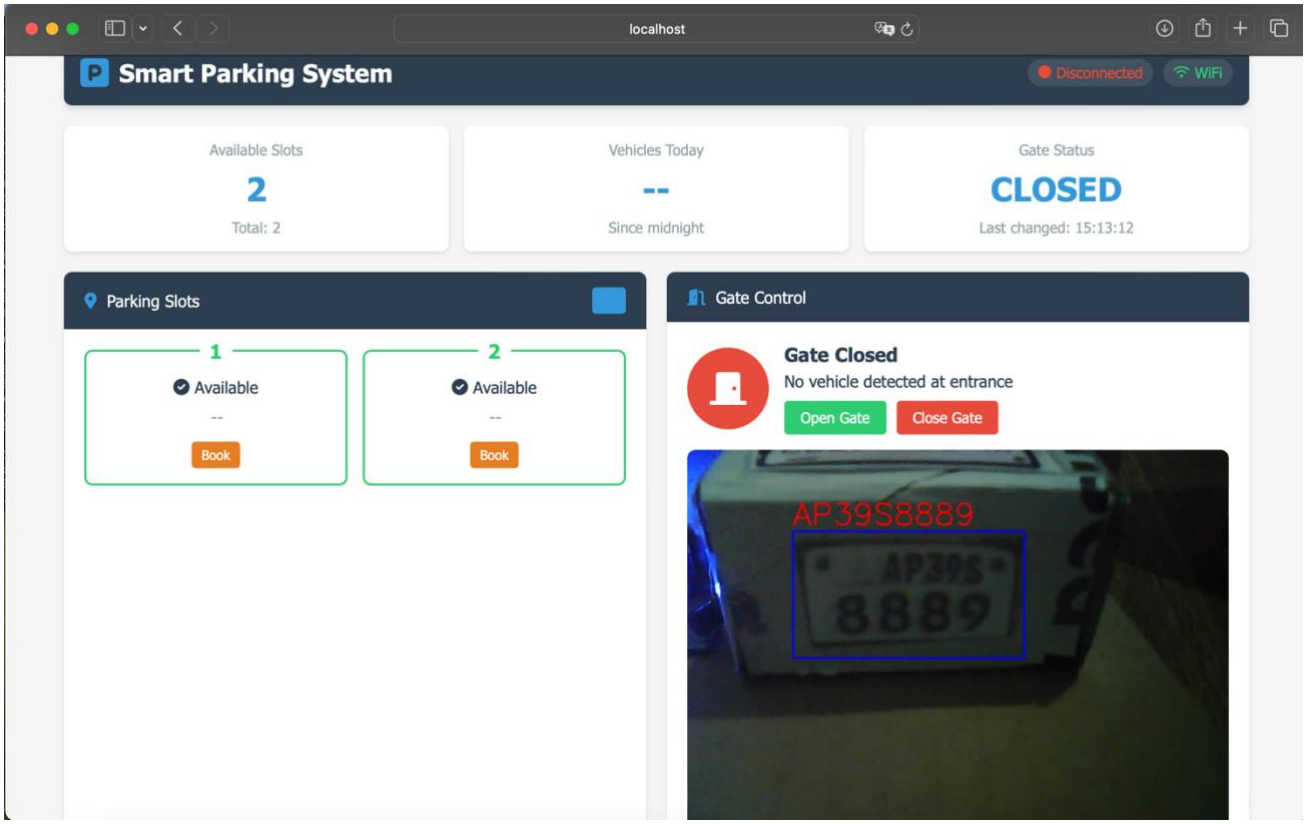
- The picture shows that the red LED is turned on to indicate that slot is booked already.



- The image shows one slot booked (car with plate AP39S8889) and the other unbooked but available, as reflected by the LEDs and the web dashboard display.



- The interface shows that the vehicle with plate number AP39S8889 has been recognized by the camera, while both slots remain unbooked and available for selection.



- Slots and plates tables from database.

```
MariaDB [parking_system1]> select * from slots;
+----+--------+-------+
| id | status | plate |
+----+--------+-------+
|  1 | free   | NULL  |
|  2 | free   | NULL  |
+----+--------+-------+
2 rows in set (0.001 sec)

MariaDB [parking_system1]> select * from plates;
+----+--------------+---------------------+---------------+----------+
| id | plate_number | timestamp           | slot_assigned | status   |
+----+--------------+---------------------+---------------+----------+
| 41 | AP3P9S8889   | 2025-05-06 20:52:44 |             1 | assigned |
| 42 | APIP39S8889  | 2025-05-06 21:09:54 |             1 | assigned |
| 43 | AP39S8889    | 2025-05-06 21:14:59 |             1 | assigned |
| 44 | AP39S8889    | 2025-05-06 21:16:46 |             2 | assigned |
```

- Table system_logs stores data collecting from hardware and software implementations.

```
| 6836 | VEHICLE_DETECTED | Vehicle at entrance                                | 2025-05-09 22:02:43 |
| 6837 | VEHICLE_DETECTED | Vehicle at entrance                                | 2025-05-09 22:03:07 |
| 6838 | PLATE_READ       | AP19S8889                                          | 2025-05-09 22:03:21 |
| 6839 | SLOT_ASSIGNED    | Available slot 2 assigned                          | 2025-05-09 22:03:21 |
| 6840 | GATE_STATUS      | OPENED,TIMEOUT                                     | 2025-05-09 22:03:21 |
| 6841 | GATE_STATUS      | CLOSED                                             | 2025-05-09 22:03:24 |
| 6842 | VEHICLE_DETECTED | Vehicle at entrance                                | 2025-05-09 22:03:25 |
| 6843 | PLATE_READ       | AP39S8889                                          | 2025-05-09 22:03:39 |
| 6844 | SLOT_ASSIGNED    | Reserved slot 1 assigned                           | 2025-05-09 22:03:40 |
| 6845 | GATE_STATUS      | OPENED,TIMEOUT                                     | 2025-05-09 22:03:40 |
| 6846 | SLOT_OCCUPIED    | Booked slot 1 is now occupied                      | 2025-05-09 22:03:41 |
| 6847 | GATE_STATUS      | CLOSED                                             | 2025-05-09 22:03:44 |
| 6848 | SLOT_OCCUPIED    | Unbooked slot 2 is now occupied                    | 2025-05-09 22:03:52 |
| 6849 | SLOT_FREED       | Slot 2 marked as free (vacated)                    | 2025-05-09 22:03:55 |
| 6850 | SLOT_OCCUPIED    | Unbooked slot 2 is now occupied                    | 2025-05-09 22:04:12 |
| 6851 | SLOT_FREED       | Slot 1 marked as free (vacated)                    | 2025-05-09 22:04:45 |
| 6852 | SLOT_FREED       | Slot 2 marked as free (vacated)                    | 2025-05-09 22:04:46 |
```

References

[1] Abdelrahman Osman Elfaki, Wassim Messoudi, Anas Bushnag, Shakour Abuzneid, and Tareq Alhmiedat, "A Smart Real-Time Parking Control and Monitoring System," Sensors, vol. 23, no. 24, pp. 9741–9741, Dec. 2023, doi: https://doi.org/10.3390/s23249741.

[2] A. Aditya, S. Anwarul, R. Tanwar, and S. K. V. Koneru, "An IoT assisted Intelligent Parking System (IPS) for Smart Cities," Procedia Computer Science, vol. 218, pp. 1045–1054, Jan. 2023, doi: https://doi.org/10.1016/j.procs.2023.01.084.

[3] How to use ESP32 CAM for Automatic Number Plate Recognition (ANPR). (2024). Circuitdigest.com. https://circuitdigest.com/projects/license-plate-recognition-using-esp32-cam