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

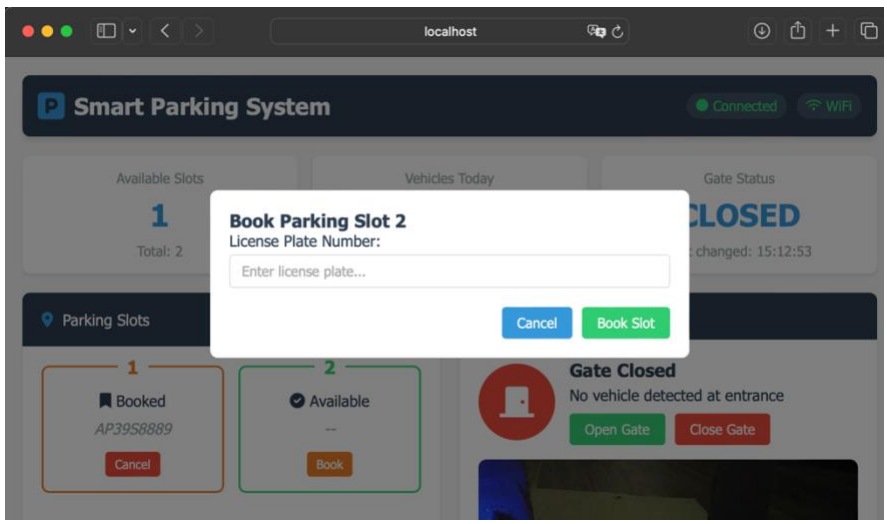


Figure 5. Web dashboard interface allowing users to book Parking Slot 2 by entering a license plate number.

- 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.

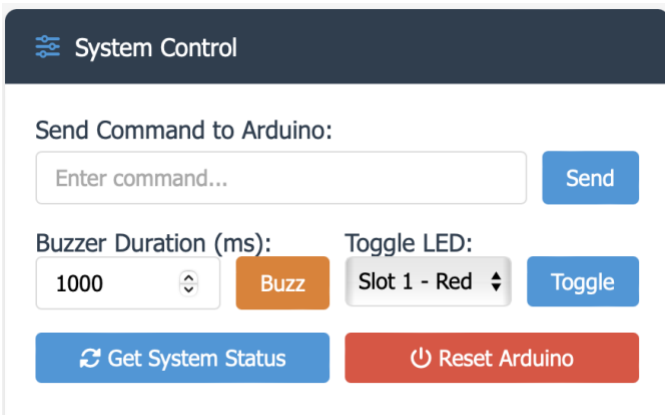


Figure 7. System control panel for sending commands, toggling LEDS, activating the buzzer

- 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.

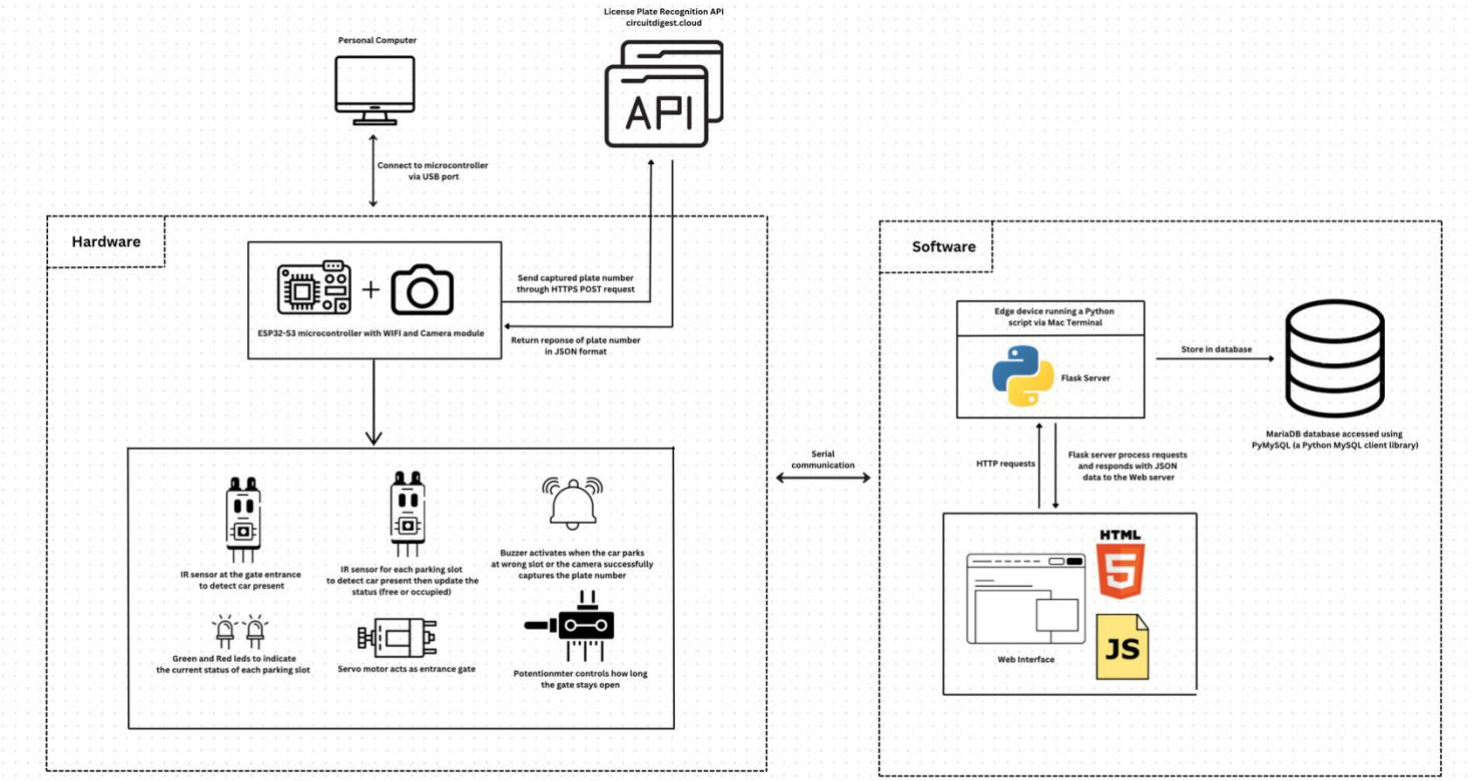


Figure 8. Block diagram of proposed system

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.

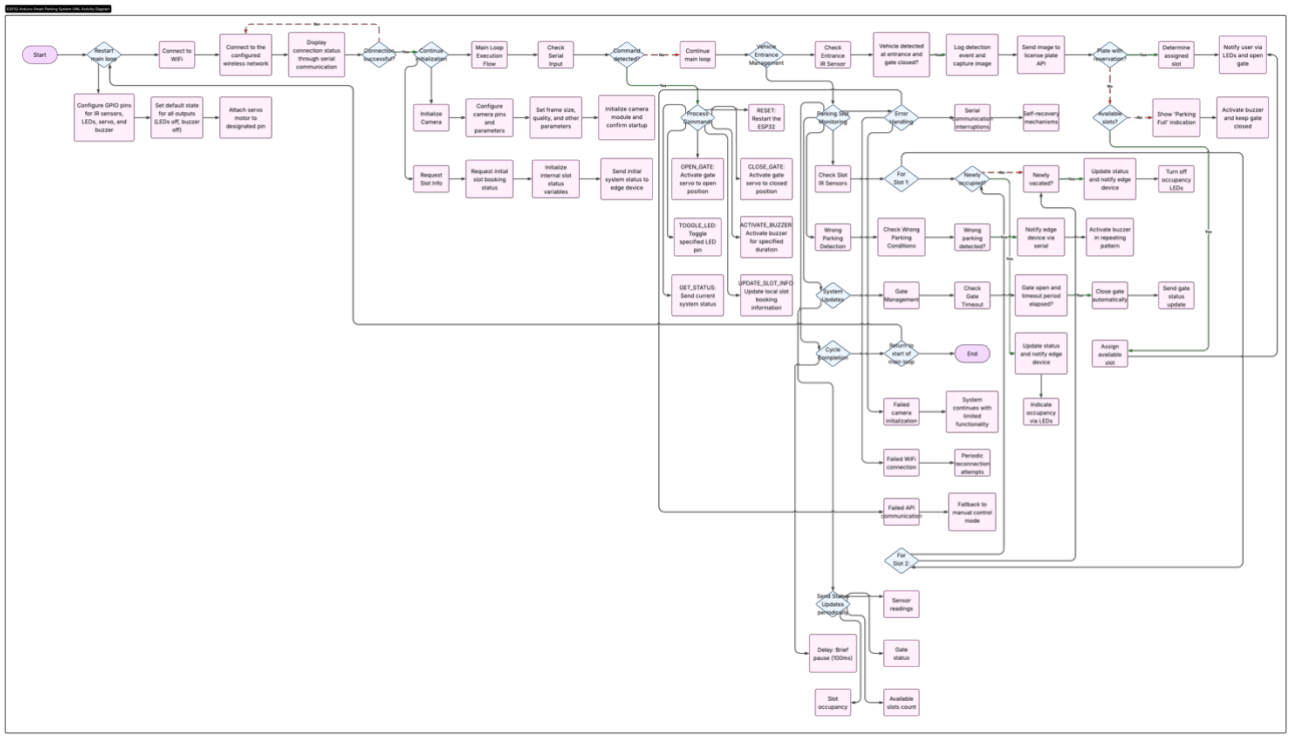


Figure 9. Flowchart of hardware implementation

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.

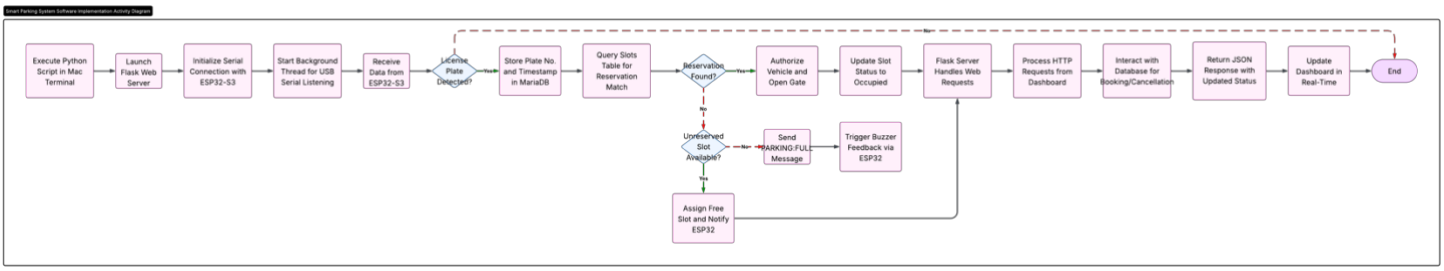


Figure 10. Flowchart of software implementation

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:

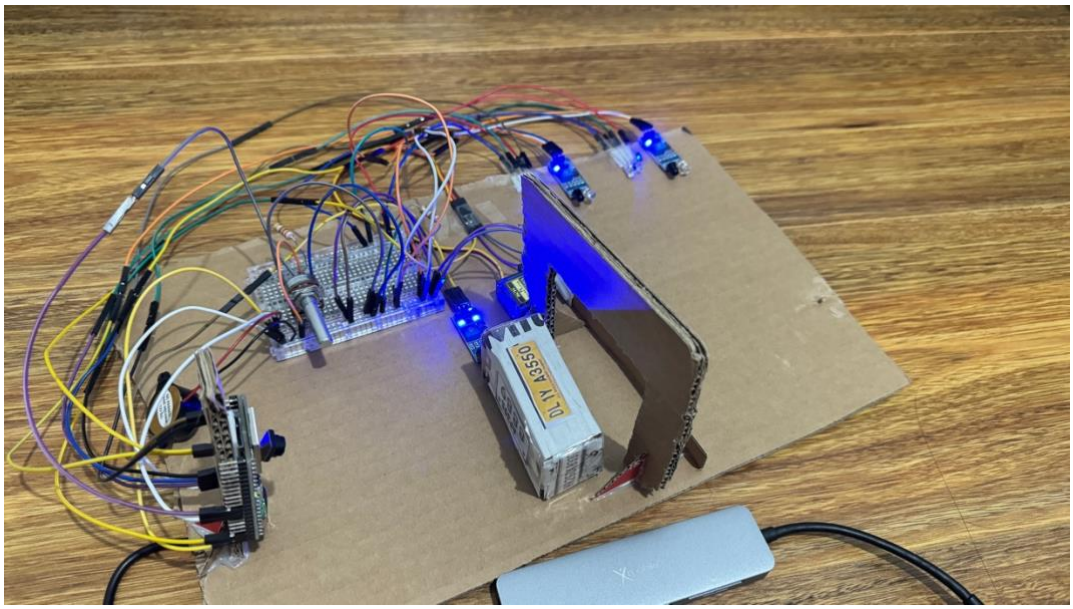
- **Circuit Digest – License Plate Recognition Using ESP32-CAM**: Used as a reference for integrating the camera module and connecting to an external OCR API service - https://circuitdigest.com/projects/license-plate-recognition-using-esp32-cam
- **Circuit Digest – AI-based Smart Parking System using 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

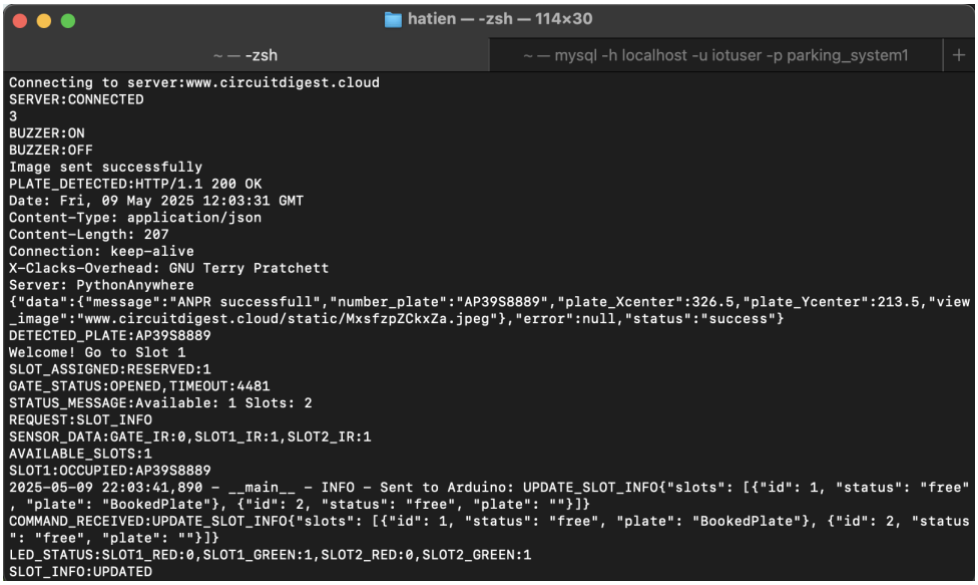
- https://www.youtube.com/watch?v=-jaxCG_dILg

6. Appendix

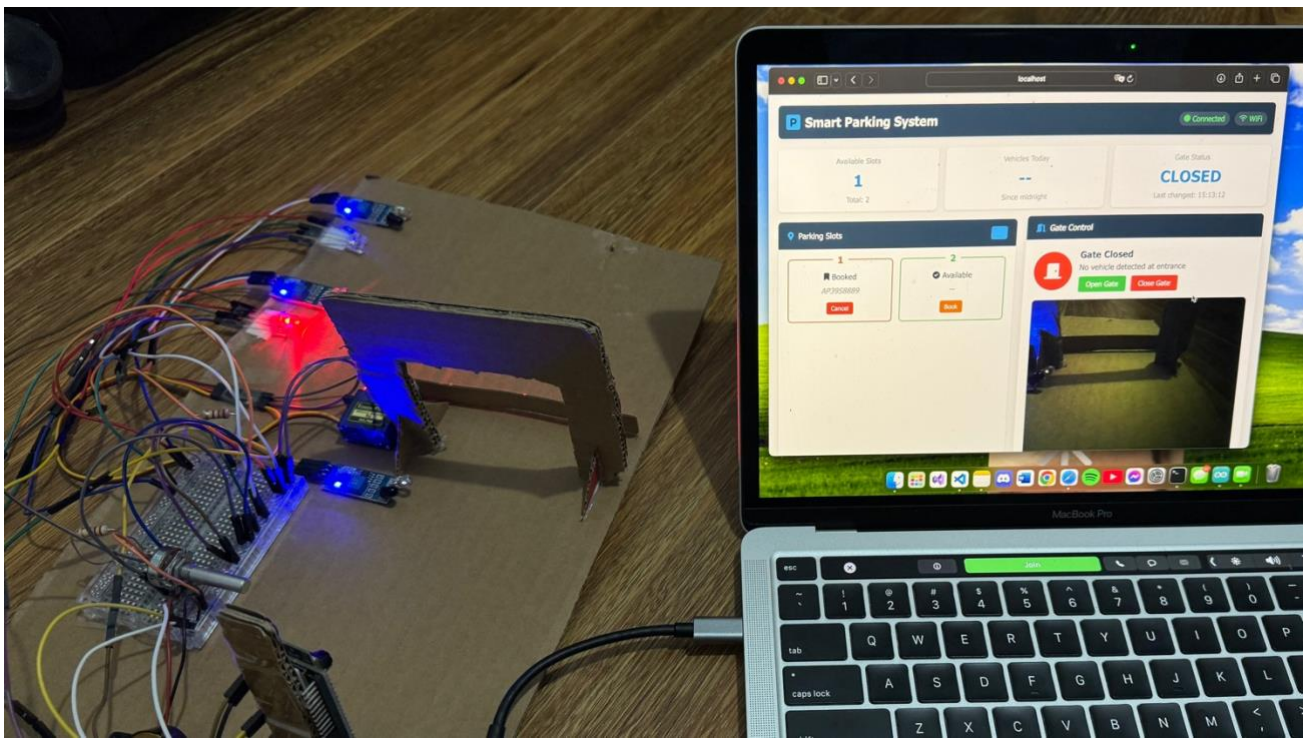
- The physical working IoT individual project – Smart Parking System Demonstrates the full hardware prototype including ESP32-S3, servo-controlled gate, IR sensors, LEDs, buzzer, and a mock vehicle used for license plate detection and automated access control.



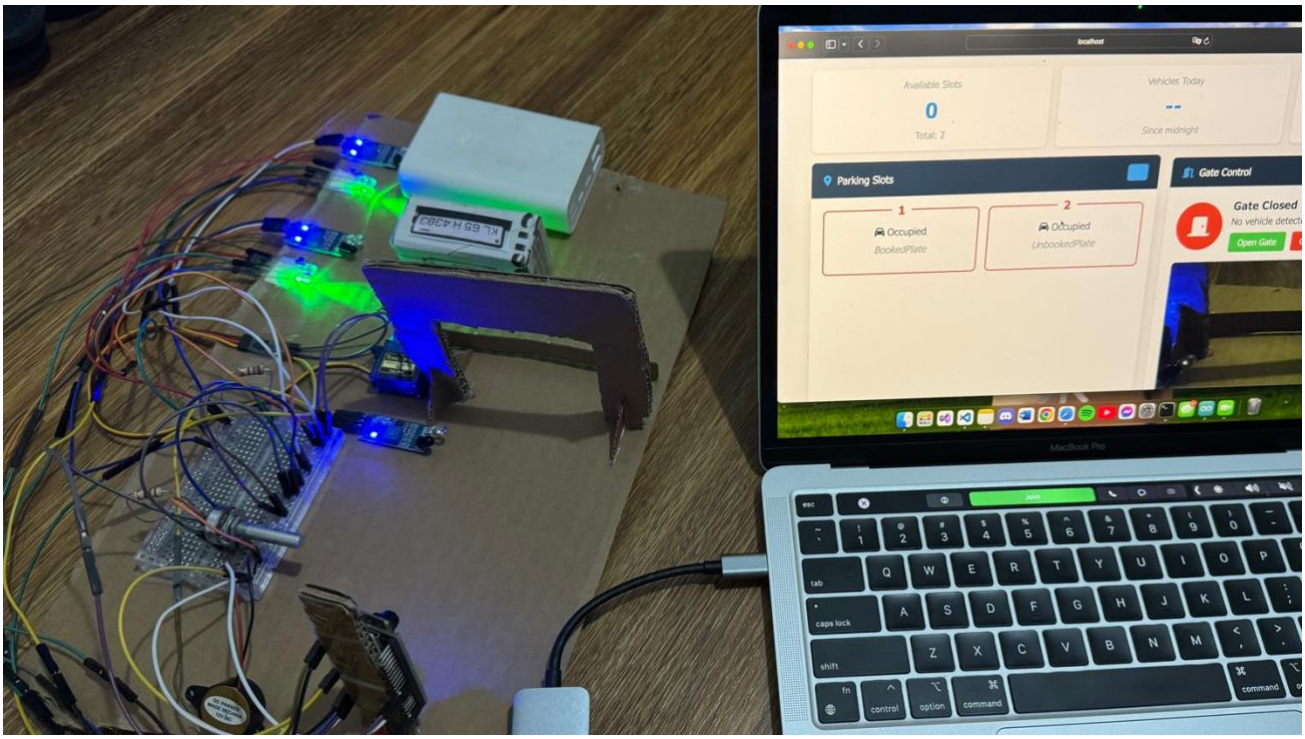
- Edge server serial output – Real-time communication between ESP32-S3 and Python Flask server: Displays live status updates, license plate detection via HTTPS API, slot assignment logic, and LED/buzzer control commands processed through the Mac terminal.



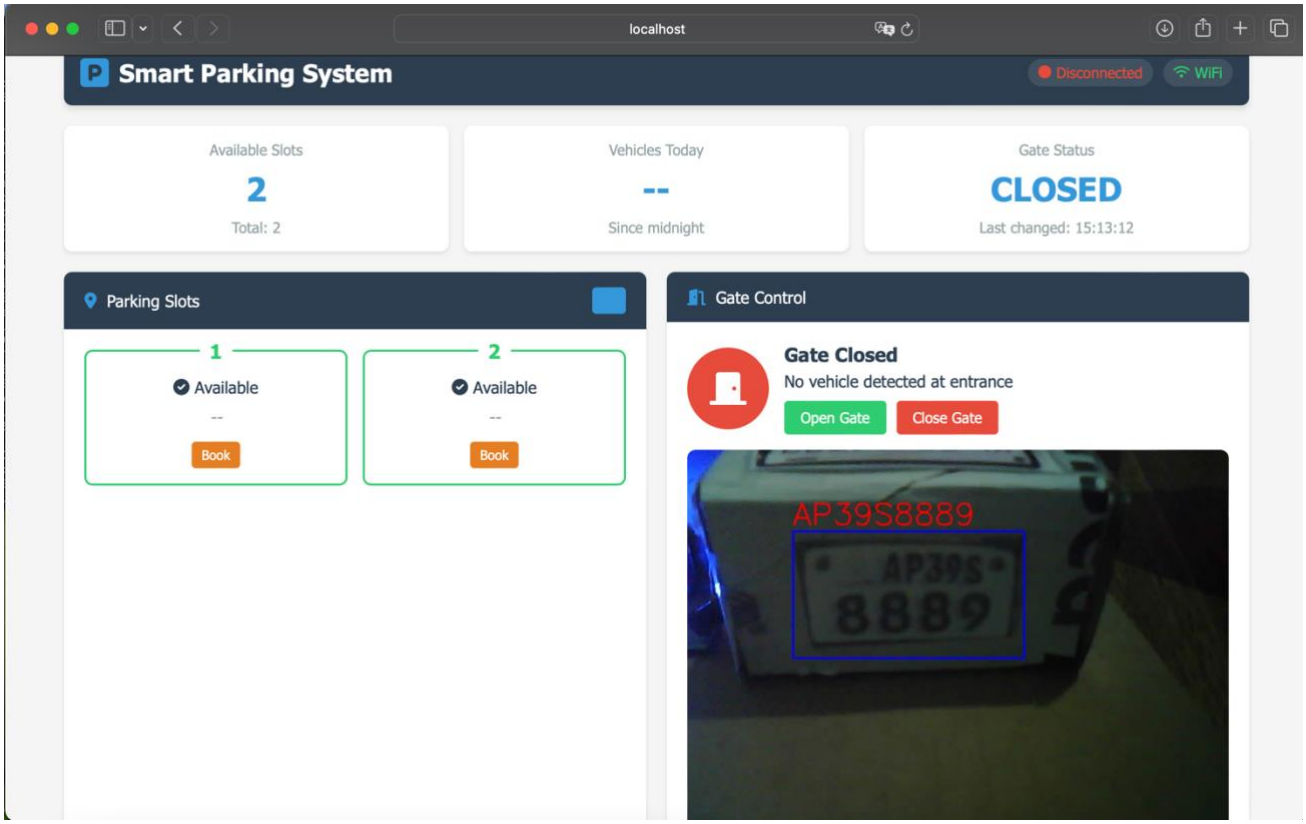
- The picture shows that the red LED is turned on to indicate that slot is booked already.



- 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 |
```

~/Documents/Arduino/smartparking/smartparking.ino

```
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "esp_camera.h"
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <ESP32Servo.h>
#include <ArduinoJson.h>

// WiFi credentials for license plate API connection
const char* ssid = "Optus_81BADE";
const char* password = "sweerhadespUe6K";

// API Server for License Plate Detection
String serverName = "www.circuitdigest.cloud";
String serverPath = "/readnumberplate";
const int serverPort = 443;
String apiKey = "MxsfzpZCkxZa";

// Initialize secure client for HTTPS API calls
WiFiClientSecure client;

// Camera GPIO pins for ESP32-S3 (correctly mapped from the pinout diagram)
#define PWDN_GPIO_NUM -1
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 15
#define SIOD_GPIO_NUM 4
#define SIOC_GPIO_NUM 5

#define Y2_GPIO_NUM 11
#define Y3_GPIO_NUM 9
#define Y4_GPIO_NUM 8
#define Y5_GPIO_NUM 10
#define Y6_GPIO_NUM 12
#define Y7_GPIO_NUM 18
#define Y8_GPIO_NUM 17
#define Y9_GPIO_NUM 16

#define VSYNC_GPIO_NUM 6
#define HREF_GPIO_NUM 7
#define PCLK_GPIO_NUM 13

// NTP Client setup
const char* ntpServer = "pool.ntp.org";
const long utcOffsetInSeconds = 25200;  // UTC+7 (Adjust to your time zone)
WiFiUDP ntpUDP;
```

```
NTPClient timeClient(ntpUDP, ntpServer, utcOffsetInSeconds);

// Define GPIO pins - CORRECTED for ESP32-S3 WROOM
// Entrance sensor and servo
const int gateIrSensorPin = 1;   // IR Sensor at entrance gate
const int servoPin = 14;         // Servo motor for gate control
const int buzzerPin = 46;        // Buzzer for alerts

// Slot 1
const int slot1IrSensorPin = 19; // IR Sensor for slot 1
const int slot1RedLedPin = 20;   // Red LED for slot 1
const int slot1GreenLedPin = 2;  // Green LED for slot 1

// Slot 2
const int slot2IrSensorPin = 21; // IR Sensor for slot 2
const int slot2RedLedPin = 35;   // Red LED for slot 2
const int slot2GreenLedPin = 36; // Green LED for slot 2

// Initialize components
Servo gateServo;
const int gateTimeoutPotPin = 3;

// System state variables
String detectedPlate = "";
String imageLink = "";
bool isGateOpen = false;
int availableSlots = 2; // Total number of slots
unsigned long gateOpenTime = 0;
unsigned long gateOpenDuration = 5000; // 10 seconds to close gate automatically

// Serial communication parameters
const unsigned long SERIAL_BAUD_RATE = 115200;
const unsigned long SERIAL_TIMEOUT = 50;  // ms
const unsigned long DATA_SEND_INTERVAL = 4000; // ms
unsigned long lastDataSentTime = 0;

// Serial protocol commands
// Commands from edge device
const String CMD_OPEN_GATE = "OPEN_GATE";
const String CMD_CLOSE_GATE = "CLOSE_GATE";
const String CMD_TOGGLE_LED = "TOGGLE_LED";
const String CMD_ACTIVATE_BUZZER = "ACTIVATE_BUZZER";
const String CMD_GET_STATUS = "GET_STATUS";
const String CMD_UPDATE_SLOT_INFO = "UPDATE_SLOT_INFO";
const String CMD_RESET = "RESET";

// Define a buffer for incoming serial data
String serialBuffer = "";
```

```
95
96  // Slot status
97  struct SlotStatus {
98    bool isBooked;         // Booked on website
99    bool isOccupied;       // Physically occupied by a car
100   String bookedPlate;    // Plate number that booked this slot
101 };
102
103 SlotStatus slots[2]; // Array for 2 slots (index 0 for slot 1, index 1 for slot
    2)
104
105 // Function prototypes
106 bool initCamera();
107 void connectToWiFi();
108 void openGate();
109 void closeGate();
110 void blinkLED(int pin, int times, int delayTime);
111 void toggleLED(int ledPin);
112 void activateBuzzer(int durationMs);
113 void updateSlotLEDs();
114 String captureAndRecognizePlate();
115 int findMatchingSlot(String plateNumber);
116 int findAvailableSlot();
117 bool checkWrongParking(int slotIndex);
118 void sendSystemStatus();
119 void parseSlotInfo(String infoString);
120 void processSerialCommand(String command);
121 void checkSerialInput();
122 String extractJsonStringValue(const String& jsonString, const String& key);
123 int count = 0;        // Counter for image uploads
124
125 // Function to extract a JSON string value by key (from API response)
126 String extractJsonStringValue(const String& jsonString, const String& key) {
127   int keyIndex = jsonString.indexOf(key);
128   if (keyIndex == -1) {
129     return "";
130   }
131
132   int startIndex = jsonString.indexOf(':', keyIndex) + 2;
133   int endIndex = jsonString.indexOf('"', startIndex);
134
135   if (startIndex == -1 || endIndex == -1) {
136     return "";
137   }
138
139   return jsonString.substring(startIndex, endIndex);
140 }
141
```

```
142 // Initialize the camera
143 bool initCamera() {
144   camera_config_t config;
145   config.ledc_channel = LEDC_CHANNEL_0;
146   config.ledc_timer = LEDC_TIMER_0;
147   config.pin_d0 = Y2_GPIO_NUM;
148   config.pin_d1 = Y3_GPIO_NUM;
149   config.pin_d2 = Y4_GPIO_NUM;
150   config.pin_d3 = Y5_GPIO_NUM;
151   config.pin_d4 = Y6_GPIO_NUM;
152   config.pin_d5 = Y7_GPIO_NUM;
153   config.pin_d6 = Y8_GPIO_NUM;
154   config.pin_d7 = Y9_GPIO_NUM;
155   config.pin_xclk = XCLK_GPIO_NUM;
156   config.pin_pclk = PCLK_GPIO_NUM;
157   config.pin_vsync = VSYNC_GPIO_NUM;
158   config.pin_href = HREF_GPIO_NUM;
159   config.pin_sscb_sda = SIOD_GPIO_NUM;
160   config.pin_sscb_scl = SIOC_GPIO_NUM;
161   config.pin_pwdn = PWDN_GPIO_NUM;
162   config.pin_reset = RESET_GPIO_NUM;
163   config.xclk_freq_hz = 20000000;
164   config.pixel_format = PIXFORMAT_JPEG;
165
166   // Adjust frame size and quality based on PSRAM availability
167     if (psramFound()) {
168     config.frame_size = FRAMESIZE_VGA;
169     config.jpeg_quality = 5;
170     config.fb_count = 2;
171     Serial.println("PSRAM found");
172   } else {
173     config.frame_size = FRAMESIZE_SVGA;
174     config.jpeg_quality = 10;
175     config.fb_count = 1;
176     config.fb_location = CAMERA_FB_IN_DRAM;
177     Serial.println("PSRAM not found");
178   }
179   // Initialize camera
180   esp_err_t err = esp_camera_init(&config);
181   if (err != ESP_OK) {
182     Serial.printf("Camera init failed with error 0x%x", err);
183     return false;
184   }
185   Serial.println("Camera initialized successfully");
186   sensor_t * s = esp_camera_sensor_get();
187   s->set_brightness(s, 2);
188   s->set_saturation(s, -1);
189   return true;
```

```
}


// Function to connect to WiFi
void connectToWiFi() {
  WiFi.mode(WIFI_STA);
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 20) {
    delay(500);
    Serial.print(".");
    attempts++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nWiFi connected");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
  } else {
    Serial.println("\nWiFi connection failed");
  }
}

// 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;

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

// Function to close the gate
void closeGate() {
  gateServo.write(90); // Position for closed gate
  isGateOpen = false;
  Serial.println("GATE_STATUS:CLOSED");
}
```

```
238
239 // Function to blink LED
240 void blinkLED(int pin, int times, int delayTime) {
241   for (int i = 0; i < times; i++) {
242     digitalWrite(pin, HIGH);
243     delay(delayTime);
244     digitalWrite(pin, LOW);
245     delay(delayTime);
246   }
247 }
248
249 // Function to toggle LED state
250 void toggleLED(int ledPin) {
251   bool currentState = digitalRead(ledPin);
252   digitalWrite(ledPin, !currentState);
253   Serial.print("LED_TOGGLED:");
254   Serial.println(ledPin);
255 }
256
257 // Function to activate buzzer
258 void activateBuzzer(int durationMs) {
259   digitalWrite(buzzerPin, HIGH);
260   Serial.println("BUZZER:ON");
261   delay(durationMs);
262   digitalWrite(buzzerPin, LOW);
263   Serial.println("BUZZER:OFF");
264 }
265
266 // Function to update slot LEDs based on status
267 void updateSlotLEDs() {
268   // Slot 1
269   digitalWrite(slot1RedLedPin, slots[0].isBooked ? HIGH : LOW);
270   // digitalWrite(slot1GreenLedPin, (slots[0].isOccupied) ? HIGH : LOW);
271
272   // Slot 2
273   digitalWrite(slot2RedLedPin, slots[1].isBooked ? HIGH : LOW);
274   // digitalWrite(slot2GreenLedPin, (slots[1].isOccupied && !slots[1].isBooked)
    ? HIGH : LOW);
275
276   // Send LED status to edge device through serial
277   Serial.print("LED_STATUS:SLOT1_RED:");
278   Serial.print(digitalRead(slot1RedLedPin));
279   Serial.print(",SLOT1_GREEN:");
280   Serial.print(digitalRead(slot1GreenLedPin));
281   Serial.print(",SLOT2_RED:");
282   Serial.print(digitalRead(slot2RedLedPin));
283   Serial.print(",SLOT2_GREEN:");
284   Serial.println(digitalRead(slot2GreenLedPin));
```

```
}

// Function to capture image and get license plate number
String captureAndRecognizePlate() {
  camera_fb_t* fb = NULL;
  // Take a photo
  Serial.println("CAMERA:CAPTURING");

  delay(300);
  fb = esp_camera_fb_get();
  delay(300);

  if (!fb) {
    Serial.println("CAMERA:FAILED");
    return "";
  }

  Serial.println("Connecting to server:" + serverName);
  client.setInsecure(); // Skip certificate validation for simplicity

  if (client.connect(serverName.c_str(), serverPort)) {
    Serial.println("SERVER:CONNECTED");

    // Prepare file name for the image
    count++;
    Serial.println(count);
    String filename = apiKey + ".jpeg";

    // Prepare HTTP POST request
    String head = "--CircuitDigest\r\nContent-Disposition: form-data;
name=\"imageFile\"; filename=\"" + filename + "\"\r\nContent-Type:
image/jpeg\r\n\r\n";
    String tail = "\r\n--CircuitDigest--\r\n";
    uint32_t imageLen = fb->len;
    uint32_t extraLen = head.length() + tail.length();
    uint32_t totalLen = imageLen + extraLen;

    client.println("POST " + serverPath + " HTTP/1.1");
    client.println("Host: " + serverName);
    client.println("Content-Length: " + String(totalLen));
    client.println("Content-Type: multipart/form-data; boundary=CircuitDigest");
    client.println("Authorization:" + apiKey);
    client.println();
    client.print(head);

    // Send the image data in chunks
    uint8_t* fbBuf = fb->buf;
    size_t fbLen = fb->len;
```

```
    for (size_t n = 0; n < fbLen; n += 1024) {
      if (n + 1024 < fbLen) {
        client.write(fbBuf, 1024);
        fbBuf += 1024;
      } else {
        size_t remainder = fbLen % 1024;
        client.write(fbBuf, remainder);
      }
    }

    client.print(tail);
    // Release resources
  esp_camera_fb_return(fb);
  activateBuzzer(200);
  Serial.println("Image sent successfully");

    // Wait for response
    String response = "";
    long startTime = millis();
    while (client.connected() && millis() - startTime < 10000) {
      if (client.available()) {
        char c = client.read();
        response += c;
      }
    }

    // Extract plate number from response
    String plateNumber = extractJsonStringValue(response, "\"number_plate\"");
    String imageLink = extractJsonStringValue(response, "\"view_image\"");

    client.stop();
    Serial.print("PLATE_DETECTED:");
    Serial.println(response);
    return plateNumber;
  } else {
    Serial.println("SERVER:CONNECTION_FAILED");
    esp_camera_fb_return(fb);
    return "";
  }
}

// Function to find if a detected plate matches any booked slot
int findMatchingSlot(String plateNumber) {
  for (int i = 0; i < 2; i++) {
    // Check if this slot is booked with the matching plate
    if (slots[i].isBooked && slots[i].bookedPlate == plateNumber) {
      return i + 1; // Return slot number (1-based)
    }
```

```
  }
  return 0; // No matching slot found
}

// Function to find an available slot for an unregistered car
int findAvailableSlot() {
  // Check if any slot is not booked
  for (int i = 0; i < 2; i++) {
    if (!slots[i].isBooked && !slots[i].isOccupied) {
      return i + 1; // Return slot number (1-based)
    }
  }
  return 0; // No available slot
}

// 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;
    }
  }
  return false;
}

// Function to send system status to the edge device
void sendSystemStatus() {
  // Use StaticJsonDocument on the stack instead of heap allocation
  StaticJsonDocument<1024> statusDoc;

  // System information
  statusDoc["system"]["gate_open"] = isGateOpen;
  statusDoc["system"]["available_slots"] = availableSlots;
  statusDoc["system"]["wifi_connected"] = (WiFi.status() == WL_CONNECTED);
  statusDoc["system"]["time"] = timeClient.getFormattedTime();

  // Slot information
  for (int i = 0; i < 2; i++) {
    JsonObject slot = statusDoc["slots"][i].to<JsonObject>();
    slot["id"] = i + 1;
    slot["booked"] = slots[i].isBooked;
```

```
    slot["occupied"] = slots[i].isOccupied;
    slot["plate"] = slots[i].bookedPlate;
  }

  // Sensor readings
  statusDoc["sensors"]["gate_ir"] = digitalRead(gateIrSensorPin);
  statusDoc["sensors"]["slot1_ir"] = digitalRead(slot1IrSensorPin);
  statusDoc["sensors"]["slot2_ir"] = digitalRead(slot2IrSensorPin);

  // Output directly to serial
  Serial.print("STATUS:");
  serializeJson(statusDoc, Serial);
  Serial.println();

  // No need to free memory as we're using a stack-allocated document
}

// Parse slot info received from edge device
void parseSlotInfo(String infoString) {
  // Use StaticJsonDocument on the stack instead of heap allocation
  StaticJsonDocument<1024> doc;

  DeserializationError error = deserializeJson(doc, infoString); // Removed * operator
  if (error) {
    Serial.print("JSON parse failed: ");
    Serial.println(error.c_str());
    return; // Removed delete as we're using stack allocation
  }

  JsonArray slotsArray = doc["slots"].as<JsonArray>(); // Removed * operator
  int i = 0;
  for (JsonObject slot : slotsArray) {
    if (i < 2) { // Only handle 2 slots
      const char* status = slot["status"];
      const char* plate = slot["plate"];

      slots[i].isBooked = (String(status) == "booked");
      slots[i].bookedPlate = plate ? String(plate) : "";
      i++;
    }
  }

  updateSlotLEDs();
  Serial.println("SLOT_INFO:UPDATED");

}
```

```
471
472
473 // Process commands received from edge device
474 void processSerialCommand(String command) {
475   // Log the received command
476   Serial.print("COMMAND_RECEIVED:");
477   Serial.println(command);
478
479   // Parse the command
480   if (command.startsWith(CMD_OPEN_GATE)) {
481     openGate();
482   }
483   else if (command.startsWith(CMD_CLOSE_GATE)) {
484     closeGate();
485   }
486   else if (command.startsWith(CMD_TOGGLE_LED)) {
487     // Format: TOGGLE_LED:PIN
488     int pinIndex = command.indexOf(':');
489     if (pinIndex != -1) {
490       String pinStr = command.substring(pinIndex + 1);
491       int pin = pinStr.toInt();
492       toggleLED(pin);
493     }
494   }
495   else if (command.startsWith(CMD_ACTIVATE_BUZZER)) {
496     // Format: ACTIVATE_BUZZER:DURATION
497     int durationIndex = command.indexOf(':');
498     if (durationIndex != -1) {
499       String durationStr = command.substring(durationIndex + 1);
500       int duration = durationStr.toInt();
501       activateBuzzer(duration);
502     }
503   }
504   else if (command.equals(CMD_GET_STATUS)) {
505     sendSystemStatus();
506   }
507   else if (command.startsWith(CMD_UPDATE_SLOT_INFO)) {
508     // Format: UPDATE_SLOT_INFO:{json_data}
509     int dataIndex = command.indexOf('{');
510     if (dataIndex != -1) {
511       String slotData = command.substring(dataIndex);
512       parseSlotInfo(slotData);
513     }
514   }
515   else if (command.equals(CMD_RESET)) {
516     Serial.println("SYSTEM:RESETTING");
517     ESP.restart();
518   }
```

```
519   else {
520     Serial.println("COMMAND:UNKNOWN");
521   }
522 }
523 
524 // Check for incoming serial commands
525 void checkSerialInput() {
526   while (Serial.available() > 0) {
527     char inChar = (char)Serial.read();
528 
529     // If newline or carriage return, process the command
530     if (inChar == '\n' || inChar == '\r') {
531       if (serialBuffer.length() > 0) {
532         processSerialCommand(serialBuffer);
533         serialBuffer = ""; // Clear buffer after processing
534       }
535     } else {
536       serialBuffer += inChar; // Add character to buffer
537     }
538   }
539 }
540 
541 void setup() {
542   // Initialize serial communication with edge device
543   Serial.begin(SERIAL_BAUD_RATE);
544   Serial.setTimeout(SERIAL_TIMEOUT);
545 
546   Serial.println("SYSTEM:INITIALIZING");
547 
548   // Initialize GPIO pins
549   pinMode(gateIrSensorPin, INPUT);
550   pinMode(slot1IrSensorPin, INPUT);
551   pinMode(slot2IrSensorPin, INPUT);
552   pinMode(slot1RedLedPin, OUTPUT);
553   pinMode(slot1GreenLedPin, OUTPUT);
554   pinMode(slot2RedLedPin, OUTPUT);
555   pinMode(slot2GreenLedPin, OUTPUT);
556   pinMode(buzzerPin, OUTPUT);
557 
558   // Set default state for outputs
559   digitalWrite(slot1RedLedPin, LOW);
560   digitalWrite(slot1GreenLedPin, LOW);
561   digitalWrite(slot2RedLedPin, LOW);
562   digitalWrite(slot2GreenLedPin, LOW);
563   digitalWrite(buzzerPin, LOW);
564 
565 
566   // Initialize servo
```

```
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  gateServo.setPeriodHertz(50);
  gateServo.attach(servoPin, 500, 2400);
  closeGate();

  // Connect to WiFi
  connectToWiFi();

  // Initialize NTP
  timeClient.begin();

  // Initialize camera
  if (!initCamera()) {
    Serial.println("CAMERA:INIT_FAILED");
    // Continue execution even if camera fails, system can still work partially
  }

  // Initialize slot status
  for (int i = 0; i < 2; i++) {
    slots[i].isBooked = false;
    slots[i].isOccupied = false;
    slots[i].bookedPlate = "";
  }

  // Request initial slot info from edge device
  Serial.println("REQUEST:SLOT_INFO");
  Serial.print("AVAILABLE_SLOTS:");
  Serial.println(availableSlots);

  // Send initial status to the edge device
  sendSystemStatus();

  Serial.println("SYSTEM:READY");
}

void loop() {
  // Check for commands from edge device
  checkSerialInput();

  // Update NTP time
  timeClient.update();

  // Check if gate should auto-close after timeout
  if (isGateOpen && (millis() - gateOpenTime > gateOpenDuration)) {
    closeGate();
  }
```

```
614
615   // Request slot information from edge device periodically (every 5 seconds)
616   static unsigned long lastDbUpdate = 0;
617   if (millis() - lastDbUpdate > 5000) {
618     Serial.println("REQUEST:SLOT_INFO");
619     lastDbUpdate = millis();
620   }
621
622   // Send sensor data to the edge device periodically
623   if (millis() - lastDataSentTime > DATA_SEND_INTERVAL) {
624     // Format: SENSOR_DATA:GATE_IR:value,SLOT1_IR:value,SLOT2_IR:value
625     Serial.print("SENSOR_DATA:GATE_IR:");
626     Serial.print(digitalRead(gateIrSensorPin));
627     Serial.print(",SLOT1_IR:");
628     Serial.print(digitalRead(slot1IrSensorPin));
629     Serial.print(",SLOT2_IR:");
630     Serial.println(digitalRead(slot2IrSensorPin));
631
632     // Update status of available slots
633     Serial.print("AVAILABLE_SLOTS:");
634     Serial.println(availableSlots);
635
636     lastDataSentTime = millis();
637   }
638
639   // Check entrance sensor for vehicle detection
640    if (digitalRead(gateIrSensorPin) == LOW && !isGateOpen) {
641     Serial.println("VEHICLE:DETECTED_AT_ENTRANCE");
642     Serial.println("Reading plate...");
643
644     // Capture and recognize license plate
645     detectedPlate = captureAndRecognizePlate();
646
647     // Notify edge device about the detected plate for database storage
648     if (detectedPlate.length() > 0 && detectedPlate != "NULL") {
649       Serial.print("DETECTED_PLATE:");
650       Serial.println(detectedPlate);
651
652       // Find if this plate has a reserved slot - using our local data
653       int matchingSlot = findMatchingSlot(detectedPlate);
654
655       if (matchingSlot > 0) {
656         // Car has a reservation
657         Serial.println("Welcome! Go to Slot " + String(matchingSlot));
658         Serial.print("SLOT_ASSIGNED:RESERVED:");
659         Serial.println(matchingSlot);
660         openGate();
```

```
661
662         // Turn on green LED for the designated slot
663         if (matchingSlot == 1) {
664           digitalWrite(slot1RedLedPin, LOW);
665           digitalWrite(slot1GreenLedPin, HIGH);
666         } else {
667           digitalWrite(slot2RedLedPin, LOW);
668           digitalWrite(slot2GreenLedPin, HIGH);
669         }
670       } else {
671         // No reservation found - check for available slots
672         int availableSlot = findAvailableSlot();
673
674         if (availableSlot > 0) {
675           // Available slot found
676           Serial.println("Available Slot: " + String(availableSlot));
677           Serial.print("SLOT_ASSIGNED:AVAILABLE:");
678           Serial.println(availableSlot);
679           openGate();
680
681           // Mark slot as occupied and update LEDs
682           if (availableSlot == 1) {
683             digitalWrite(slot1GreenLedPin, HIGH);
684           } else {
685             digitalWrite(slot2GreenLedPin, HIGH);
686           }
687         } else {
688           // No available slots
689           Serial.println("Sorry! No slots available");
690           Serial.println("PARKING:FULL");
691           activateBuzzer(500);
692         }
693       }
694     } else {
695       // Failed to detect plate
696       Serial.println("Error: Plate not detected or invalid");
697       Serial.println("PLATE:DETECTION_FAILED");
698       activateBuzzer(500);
699     }
700   }
701 // Continuous wrong parking alert
702 for (int i = 0; i < 2; i++) {
703   if (checkWrongParking(i)) {
704     Serial.printf("Wrong Parking! Not your slot %d\n", i + 1);
705     Serial.printf("WRONG_PARKING:SLOT%d\n", i + 1);
706
707     // Loop beep while car is still wrongly parked
708     while (checkWrongParking(i)) {
```

```
709       activateBuzzer(200);  // Short beep
710       delay(300);           // Pause between beeps
711     }
712   }
713 }
714
715   // Update available slots count
716   availableSlots = 0;
717   for (int i = 0; i < 2; i++) {
718     if (!slots[i].isBooked && !slots[i].isOccupied) {
719       availableSlots++;
720     }
721   }
722
723   // Check parking slots occupancy via IR sensors
724   static bool lastSlot1Occupied = false;
725   bool slot1Occupied = (digitalRead(slot1IrSensorPin) == LOW);
726   slots[0].isOccupied = slot1Occupied;
727
728   // Handle slot 1 status changes
729   if (slot1Occupied != lastSlot1Occupied) {
730     if (slot1Occupied) {
731       //Car has arrived in slot 1
732       Serial.println("SLOT1:OCCUPIED:" + detectedPlate);
733
734       if (slots[0].isBooked) {
735         // This car booked the slot → blink and keep LED on
736         blinkLED(slot1GreenLedPin, 3, 300);
737         digitalWrite(slot1GreenLedPin, HIGH);
738       } else {
739         // Unbooked car → just turn on LED
740         blinkLED(slot1GreenLedPin, 3, 300);
741         digitalWrite(slot1GreenLedPin, HIGH);
742       }
743     } else {
744       // Car has left slot 1
745       Serial.println("SLOT1:VACATED");
746       digitalWrite(slot1GreenLedPin, LOW);
747     }
748
749     // Save new state
750     lastSlot1Occupied = slot1Occupied;
751     // updateSlotLEDs();
752   }
753
754   // Check parking slots occupancy via IR sensors
755   static bool lastSlot2Occupied = false;
```

```
  bool slot2Occupied = (digitalRead(slot2IrSensorPin) == LOW);
  slots[1].isOccupied = slot2Occupied;

  // Handle slot 2 status changes
  if (slot2Occupied != lastSlot2Occupied) {
    if (slot2Occupied) {
      //Car has arrived in slot 2
      Serial.println("SLOT2:OCCUPIED:" + detectedPlate);

      if (slots[1].isBooked) {
        // This car booked the slot → blink and keep LED on
        blinkLED(slot2GreenLedPin, 3, 300);
        digitalWrite(slot2GreenLedPin, HIGH);
      } else {
        // Unbooked car → just turn on LED
        blinkLED(slot2GreenLedPin, 3, 300);
        digitalWrite(slot2GreenLedPin, HIGH);
      }
    } else {
      // Car has left slot 2
      Serial.println("SLOT2:VACATED");
      digitalWrite(slot2GreenLedPin, LOW);
    }

    // Save new state
    lastSlot2Occupied = slot2Occupied;
    // updateSlotLEDs();
  }

  // Output parking status through serial
  if (millis() - lastDbUpdate > 5000) {  // Update every 5 seconds
    String statusMessage = "Available: " + String(availableSlots) + " Slots: ";

    if (!slots[0].isBooked && !slots[0].isOccupied) {
      statusMessage += "1 ";
    }
    if (!slots[1].isBooked && !slots[1].isOccupied) {
      statusMessage += "2";
    }

    if (availableSlots == 0) {
      statusMessage += "No slots free";
    }

    Serial.println("STATUS_MESSAGE:" + statusMessage);
  }
```

```
803   // Delay to prevent excessive looping
804   delay(100);
805 }
```

~/Documents/parking.py

```python
from flask import Flask, render_template, request, jsonify, redirect, url_for, flash
import serial
import json
import threading
import time
import pymysql
from datetime import datetime
import os
import logging
from collections import deque

# Configure your app logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)
logger = logging.getLogger(__name__)

# Suppress default Flask/Werkzeug access logs
logging.getLogger('werkzeug').setLevel(logging.ERROR)


# Flask application setup
app = Flask(__name__)
app.secret_key = os.urandom(24)

# Configure database connection
DB_CONFIG = {
    'host': 'localhost',
    'user': 'iotuser',
    'password': 'hatien2107',
    'database': 'parking_system1'
}

# Serial communication configuration
SERIAL_PORT = '/dev/cu.wchusbserial58760815361'
SERIAL_BAUD_RATE = 115200
SERIAL_TIMEOUT = 1

# Global variables
serial_conn = None
serial_data_buffer = deque(maxlen=100)  # Store recent serial messages
system_status = {
    'gate_open': False,
    'available_slots': 2,
```

```
    'wifi_connected': False,
    'timestamp': None
}
slots_status = [
    {'id': 1, 'booked': False, 'occupied': False, 'plate': None},
    {'id': 2, 'booked': False, 'occupied': False, 'plate': None}
]


# ========================
# Database Functions
# ========================

def initialize_database():
    """Initialize database tables if they don't exist"""
    try:
        conn = pymysql.connect(**DB_CONFIG)
        cursor = conn.cursor()

        # Create slots table
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS slots (
                id INT PRIMARY KEY,
                status VARCHAR(20) NOT NULL DEFAULT 'free',
                plate VARCHAR(20) NULL
            )
        """)

        # Create plates table for detected plates
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS plates (
                id INT AUTO_INCREMENT PRIMARY KEY,
                plate_number VARCHAR(20) NOT NULL,
                timestamp DATETIME NOT NULL,
                slot_assigned INT NULL,
                status VARCHAR(20) NOT NULL DEFAULT 'detected'
            )
        """)

        # Create log table for system events
        cursor.execute("""
            CREATE TABLE IF NOT EXISTS system_logs (
                id INT AUTO_INCREMENT PRIMARY KEY,
                event_type VARCHAR(50) NOT NULL,
                event_data TEXT NULL,
                timestamp DATETIME NOT NULL
            )
        """)
```

```
94
95          # Insert default slots if not exist
96          cursor.execute("SELECT COUNT(*) FROM slots")
97          if cursor.fetchone()[0] < 2:
98              cursor.execute("DELETE FROM slots")  # Clear existing if partial
99              cursor.execute("INSERT INTO slots (id, status) VALUES (1, 'free')")
100             cursor.execute("INSERT INTO slots (id, status) VALUES (2, 'free')")
101
102         conn.commit()
103         logger.info("Database initialized successfully")
104
105         # Load initial slot status
106         load_slots_from_database()
107     except Exception as e:
108         logger.error(f"Database initialization error: {str(e)}")
109     finally:
110         if conn:
111             conn.close()
112
113 # This function logs system events to the database: system_logs
114 def log_system_event(event_type, event_data=None):
115     """Log system event to database"""
116     try:
117         conn = pymysql.connect(**DB_CONFIG)
118         cursor = conn.cursor()
119
120         timestamp = datetime.now()
121         cursor.execute(
122             "INSERT INTO system_logs (event_type, event_data, timestamp) VALUES
    (%s, %s, %s)",
123             (event_type, event_data, timestamp)
124         )
125
126         conn.commit()
127     except Exception as e:
128         logger.error(f"Failed to log system event: {str(e)}")
129     finally:
130         if conn:
131             conn.close()
132
133 def load_slots_from_database():
134     """Load slot information from database"""
135     try:
136         conn = pymysql.connect(**DB_CONFIG)
137         cursor = conn.cursor(pymysql.cursors.DictCursor)
138         cursor.execute("SELECT * FROM slots")
139         db_slots = cursor.fetchall()
140
```

```
        # Update global slots status
        for db_slot in db_slots:
            slot_idx = db_slot['id'] - 1
            if slot_idx >= 0 and slot_idx < len(slots_status):
                slots_status[slot_idx]['booked'] = db_slot['status'] == 'booked'
                slots_status[slot_idx]['plate'] = db_slot['plate']

        # Update Arduino about current slot status
        send_slot_info_to_arduino()
    except Exception as e:
        logger.error(f"Failed to load slots from database: {str(e)}")
    finally:
        if conn:
            conn.close()

# This function updates the slot status in the database and also updates the global state
def update_slot_in_database(slot_id, status, plate=None):
    """Update slot status in database"""
    try:
        conn = pymysql.connect(**DB_CONFIG)
        cursor = conn.cursor()

        # Handle different statuses properly
        if status == "free":
            cursor.execute(
                "UPDATE slots SET status = %s, plate = NULL WHERE id = %s",
                (status, slot_id)
            )
        elif status == "occupied":
            # Check if this slot was already booked - if so, maintain its status
            cursor.execute("SELECT status, plate FROM slots WHERE id = %s", (slot_id,))
                # It's an unbooked slot being occupied
            cursor.execute(
                "UPDATE slots SET status = %s, plate = %s WHERE id = %s",
                (status, plate, slot_id)
            )
        else:
            # Booked or other status
            cursor.execute(
                "UPDATE slots SET status = %s, plate = %s WHERE id = %s",
                (status, plate, slot_id)
            )

        conn.commit()
        # Update global state
        slots_status[slot_id-1]['booked'] = (status == 'booked')
```

```
187         slots_status[slot_id-1]['occupied'] = (status == 'occupied' or
188                                                (status == 'booked' and
    slots_status[slot_id-1]['occupied']))
189         slots_status[slot_id-1]['plate'] = plate
190 
191         # Notify Arduino about the change
192         send_slot_info_to_arduino()
193 
194         return True
195     except Exception as e:
196         logger.error(f"Database update error for slot {slot_id}: {str(e)}")
197         return False
198     finally:
199         if conn:
200             conn.close()
201 
202 # This function stores detected plates in the database
203 def store_detected_plate(plate_number, slot_assigned=None):
204     """Store a detected license plate in the database"""
205     try:
206         conn = pymysql.connect(**DB_CONFIG)
207         cursor = conn.cursor()
208 
209         timestamp = datetime.now()
210         status = 'assigned' if slot_assigned else 'detected'
211 
212         cursor.execute(
213             "INSERT INTO plates (plate_number, timestamp, slot_assigned, status)
    VALUES (%s, %s, %s, %s)",
214             (plate_number, timestamp, slot_assigned, status)
215         )
216 
217         conn.commit()
218         return True
219     except Exception as e:
220         logger.error(f"Failed to store detected plate: {str(e)}")
221         return False
222     finally:
223         if conn:
224             conn.close()
225 
226 #THIS IS REALLY IMPORTANT PART: this function checks if a plate has a
    reservation and returns the slot if found
227 def check_plate_reservation(plate_number):
228     """Check if a plate has a reservation and return slot if found"""
229     conn = None
230 
231     try:
```

```
232         conn = pymysql.connect(**DB_CONFIG)
233         cursor = conn.cursor(pymysql.cursors.DictCursor)
234 
235         cursor.execute(
236             "SELECT id FROM slots WHERE status = 'booked' AND plate = %s",
237             (plate_number,)
238         )
239         result = cursor.fetchone()
240 
241         if result:
242             return result['id']
243         return None
244     except Exception as e:
245         logger.error(f"Error checking plate reservation: {str(e)}")
246         return None
247     finally:
248         if conn:
249             conn.close()
250 
251 # This function finds an available parking slot
252 def find_available_slot():
253     """Find an available parking slot"""
254     try:
255         conn = pymysql.connect(**DB_CONFIG)
256         cursor = conn.cursor(pymysql.cursors.DictCursor)
257 
258         cursor.execute("SELECT id FROM slots WHERE status = 'free' LIMIT 1")
259         result = cursor.fetchone()
260 
261         if result:
262             return result['id']
263         return None
264     except Exception as e:
265         logger.error(f"Error finding available slot: {str(e)}")
266         return None
267     finally:
268         if conn:
269             conn.close()
270 
271 # This function retrieves recent logs from the database and then uploads them to
    the web interface
272 def get_recent_logs(limit=20):
273     """Get recent system logs"""
274     try:
275         conn = pymysql.connect(**DB_CONFIG)
276         cursor = conn.cursor(pymysql.cursors.DictCursor)
277 
278         cursor.execute(
```

```
            "SELECT * FROM system_logs ORDER BY timestamp DESC LIMIT %s",
            (limit,)
        )

        return cursor.fetchall()
    except Exception as e:
        logger.error(f"Error retrieving logs: {str(e)}")
        return []
    finally:
        if conn:
            conn.close()

# This function retrieves recent detected plates from the database and then uploads them to the web interface
def get_recent_plates(limit=20):
    """Get recently detected plates"""
    try:
        conn = pymysql.connect(**DB_CONFIG)
        cursor = conn.cursor(pymysql.cursors.DictCursor)

        cursor.execute(
            "SELECT * FROM plates ORDER BY timestamp DESC LIMIT %s",
            (limit,)
        )

        return cursor.fetchall()
    except Exception as e:
        logger.error(f"Error retrieving recent plates: {str(e)}")
        return []
    finally:
        if conn:
            conn.close()


# ========================
# Serial Communication
# ========================

def initialize_serial():
    """Initialize serial connection to Arduino"""
    global serial_conn
    try:
        serial_conn = serial.Serial(SERIAL_PORT, SERIAL_BAUD_RATE, timeout=SERIAL_TIMEOUT)
        logger.info(f"Serial connection established on {SERIAL_PORT}")
        return True
    except Exception as e:
        logger.error(f"Failed to initialize serial connection: {str(e)}")
```

```
        return False

# This function sends commands to Arduino and handles the response
def send_to_arduino(command):
    """Send a command to Arduino through serial connection"""
    global serial_conn
    try:
        if serial_conn and serial_conn.is_open:
            # Add newline to end command
            if not command.endswith('\n'):
                command += '\n'
            serial_conn.write(command.encode())
            serial_conn.flush()
            logger.info(f"Sent to Arduino: {command.strip()}")
            return True
        else:
            logger.error("Serial connection not available")
            return False
    except Exception as e:
        logger.error(f"Failed to send command to Arduino: {str(e)}")
        return False


def send_slot_info_to_arduino():
    """Send current slot information to Arduino"""
    try:
        # Create JSON structure for Arduino
        slots_data = {
            "slots": [
                {
                    "id": status['id'],
                    "status": "booked" if status['booked'] else "free",
                    "plate": status['plate'] if status['plate'] else ""
                }
                for status in slots_status
            ]
        }

        # Send command with JSON data
        command = f"UPDATE_SLOT_INFO{json.dumps(slots_data)}"
        send_to_arduino(command)
        return True
    except Exception as e:
        logger.error(f"Failed to send slot info to Arduino: {str(e)}")
        return False


def read_from_arduino():
```

```
    """Read data from Arduino"""
    global serial_conn, system_status, slots_status, serial_data_buffer

    try:
        if serial_conn and serial_conn.is_open:
            if serial_conn.in_waiting > 0:
                line = serial_conn.readline().decode('utf-8').strip()
                if line:
                    # Add to buffer
                    serial_data_buffer.append(f"
{datetime.now().strftime('%H:%M:%S')} - {line}")
                    process_arduino_message(line)
                return line
        return None
    except Exception as e:
        logger.error(f"Error reading from Arduino: {str(e)}")
        return None

# This function processes messages received from Arduino and updates the system
state
def process_arduino_message(message):
    """Process messages received from Arduino"""
    global system_status, slots_status

    try:
        # Add debugging - print all messages being processed
        print(f"{message}")

        # Handle different message types
        if message.startswith("DETECTED_PLATE:"):
            plate = message.split(":")[1]
            log_system_event("PLATE_READ", plate)
            # Check if plate has a booking
            reserved_slot = check_plate_reservation(plate)

            if reserved_slot:
                # It's a reserved plate - log as assigned
                store_detected_plate(plate, slot_assigned=reserved_slot)
            else:
                # No reservation - try to auto-assign a free slot
                free_slot = find_available_slot()

                if free_slot:
                    # Update DB to assign this free slot to the plate
                    store_detected_plate(plate, slot_assigned=free_slot)
                else:
                    # No free slot available - just log the plate as detected
                    store_detected_plate(plate)
```

```
419
420         elif message.startswith("GATE_STATUS:"):
421             status = message.split(":")[1]
422             system_status['gate_open'] = (status == "OPENED")
423             log_system_event("GATE_STATUS", status)
424
425         elif message.startswith("VEHICLE:DETECTED_AT_ENTRANCE"):
426             log_system_event("VEHICLE_DETECTED", "Vehicle at entrance")
427
428         elif message.startswith("WRONG_PARKING:"):
429             slot = message.split(":")[1]
430             log_system_event("WRONG_PARKING", f"Wrong parking in {slot}")
431
432         elif message.strip() == "PARKING:FULL":
433             log_system_event("PARKING_FULL", "No parking slots available")
434
435         elif message.startswith("SLOT_ASSIGNED:"):
436             parts = message.split(":")
437             assignment_type = parts[1]
438             slot_num = int(parts[2])
439
440             if assignment_type == "RESERVED":
441                 log_system_event("SLOT_ASSIGNED", f"Reserved slot {slot_num}
    assigned")
442             else:
443                 log_system_event("SLOT_ASSIGNED", f"Available slot {slot_num}
    assigned")
444
445         elif message.startswith("SLOT1:") or message.startswith("SLOT2:"):
446             slot_num = int(message[4:5])
447             status = message.split(":")[1]
448
449             # Update occupancy but preserve booking status
450             if status == "OCCUPIED":
451                 # Get current slot status from database
452                 conn = pymysql.connect(**DB_CONFIG)
453                 cursor = conn.cursor(pymysql.cursors.DictCursor)
454                 cursor.execute("SELECT status, plate FROM slots WHERE id = %s",
    (slot_num,))
455                 current = cursor.fetchone()
456                 conn.close()
457
458                 if current and current['status'] == "booked":
459                     # It's a booked slot that is now physically occupied
460                     slots_status[slot_num-1]['occupied'] = True
461
462                     # Update DB: preserve booking but mark as occupied
463                     update_slot_in_database(slot_num, "occupied", "BookedPlate")
```

```
464                     log_system_event("SLOT_OCCUPIED", f"Booked slot {slot_num}
    is now occupied")
465 
466                 else:
467                     # Walk-in car (no booking)
468                     slots_status[slot_num-1]['occupied'] = True
469                     # Update DB: mark slot as occupied and store detected plate
    if needed
470                     update_slot_in_database(slot_num, "occupied",
    "UnbookedPlate")
471                     log_system_event("SLOT_OCCUPIED", f"Unbooked slot {slot_num}
    is now occupied")
472 
473 
474             elif status == "VACATED":
475                 slots_status[slot_num-1]['occupied'] = False
476 
477                 # Directly update DB to clear booking & free the slot
478                 update_slot_in_database(slot_num, "free")
479                 # log the removal
480                 log_system_event("SLOT_FREED", f"Slot {slot_num} marked as free
    (vacated)")
481 
482 
483     except Exception as e:
484         logger.error(f"Error processing Arduino message '{message}': {str(e)}")
485 
486 
487 def serial_monitor_thread():
488     """Background thread to monitor serial communication"""
489     while True:
490         try:
491             read_from_arduino()
492             time.sleep(0.1)  # Small delay to prevent CPU hogging
493         except Exception as e:
494             logger.error(f"Serial monitor thread error: {str(e)}")
495             time.sleep(1)  # Longer delay on error
496 
497 
498 # ========================
499 # Flask Routes
500 # ========================
501 
502 @app.route('/')
503 def index():
504     """Main dashboard page"""
505     return render_template('index.html',
506                            slots=slots_status,
```

```
                           system=system_status,
                           serial_messages=list(serial_data_buffer))


@app.route('/slots')
def slots_page():
    """Slot management page"""
    return render_template('slots.html', slots=slots_status)


@app.route('/logs')
def logs_page():
    """System logs page"""
    logs = get_recent_logs(50)
    plates = get_recent_plates(20)
    return render_template('logs.html', logs=logs, plates=plates)


@app.route('/book_slot', methods=['POST'])
def book_slot():
    """Book a parking slot"""
    slot_id = int(request.form.get('slot_id'))
    plate_number = request.form.get('plate_number')

    if not plate_number:
        flash('License plate number is required', 'error')
        return redirect(url_for('index'))

    # Check if the plate already has a booking
    existing_slot = check_plate_reservation(plate_number)
    if existing_slot:
        flash(f'This plate already has a booking for slot {existing_slot}',
'error')
        return redirect(url_for('index'))

    # Update database
    success = update_slot_in_database(slot_id, 'booked', plate_number)

    if success:
        flash(f'Slot {slot_id} successfully booked for plate {plate_number}',
'success')
    else:
        flash('Failed to book slot. Please try again.', 'error')

    return redirect(url_for('index'))


@app.route('/cancel_booking', methods=['POST'])
```

```python
def cancel_booking():
    """Cancel a slot booking"""
    slot_id = int(request.form.get('slot_id'))

    # Update database
    success = update_slot_in_database(slot_id, 'free')

    if success:
        flash(f'Booking for slot {slot_id} has been cancelled', 'success')
    else:
        flash('Failed to cancel booking. Please try again.', 'error')

    return redirect(url_for('index'))


@app.route('/api/system_status')
def get_system_status():
    try:
        conn = pymysql.connect(**DB_CONFIG)
        cursor = conn.cursor(pymysql.cursors.DictCursor)

        # Count slots that are actually free
        cursor.execute("SELECT COUNT(*) AS available FROM slots WHERE status = 'free'")
        available = cursor.fetchone()['available']

        # Fetch current full slot status
        cursor.execute("SELECT * FROM slots")
        all_slots = cursor.fetchall()

        conn.close()

        return jsonify({
            "system": {
                "available_slots": available,
                "wifi_connected": True,  # set based on your actual check
                "timestamp": time.time()
            },
            "slots": all_slots
        })

    except Exception as e:
        logger.error(f"System status error: {e}")
        return jsonify({
            "system": {
                "available_slots": 0,
                "wifi_connected": False,
                "timestamp": time.time()
```

```
            },
            "slots": []
        })


@app.route('/api/serial_messages')
def api_serial_messages():
    """API endpoint for recent serial messages"""
    return jsonify(list(serial_data_buffer))


@app.route('/api/send_command', methods=['POST'])
def api_send_command():
    """API endpoint to send commands to Arduino"""
    command = request.form.get('command')

    if not command:
        return jsonify({'success': False, 'message': 'No command provided'})

    success = send_to_arduino(command)
    return jsonify({'success': success})


@app.route('/arduino/open_gate')
def arduino_open_gate():
    """Direct command to open gate"""
    success = send_to_arduino("OPEN_GATE")
    if success:
        flash('Command sent to open gate', 'success')
    else:
        flash('Failed to send open gate command', 'error')
    return redirect(url_for('index'))


@app.route('/arduino/close_gate')
def arduino_close_gate():
    """Direct command to close gate"""
    success = send_to_arduino("CLOSE_GATE")
    if success:
        flash('Command sent to close gate', 'success')
    else:
        flash('Failed to send close gate command', 'error')
    return redirect(url_for('index'))


@app.route('/arduino/toggle_led', methods=['POST'])
def arduino_toggle_led():
```

```python
    """Toggle an LED on Arduino"""
    pin = request.form.get('pin')
    if not pin:
        flash('No LED pin specified', 'error')
        return redirect(url_for('index'))

    success = send_to_arduino(f"TOGGLE_LED:{pin}")
    if success:
        flash(f'Command sent to toggle LED on pin {pin}', 'success')
    else:
        flash('Failed to send toggle LED command', 'error')
    return redirect(url_for('index'))


@app.route('/arduino/buzzer', methods=['POST'])
def arduino_buzzer():
    """Activate buzzer on Arduino"""
    duration = request.form.get('duration', '1000')
    success = send_to_arduino(f"ACTIVATE_BUZZER:{duration}")
    if success:
        flash(f'Command sent to activate buzzer for {duration}ms', 'success')
    else:
        flash('Failed to send buzzer command', 'error')
    return redirect(url_for('index'))


@app.route('/arduino/get_status')
def arduino_get_status():
    """Request status update from Arduino"""
    success = send_to_arduino("GET_STATUS")
    if success:
        flash('Status update requested from Arduino', 'success')
    else:
        flash('Failed to request status from Arduino', 'error')
    return redirect(url_for('index'))


@app.route('/arduino/reset')
def arduino_reset():
    """Reset Arduino"""
    success = send_to_arduino("RESET")
    if success:
        flash('Reset command sent to Arduino', 'success')
    else:
        flash('Failed to send reset command', 'error')
    return redirect(url_for('index'))


```

```python
# ========================
# Application Initialization
# ========================

def initialize_app():
    """Initialize the application - database and serial connection"""
    # Initialize database
    initialize_database()

    # Try to initialize serial connection
    serial_success = initialize_serial()
    if not serial_success:
        logger.warning("Serial connection failed, will retry in background")

    # Start serial monitor thread
    thread = threading.Thread(target=serial_monitor_thread, daemon=True)
    thread.start()

    # Request initial status from Arduino if connection successful
    if serial_success:
        send_to_arduino("GET_STATUS")


if __name__ == '__main__':
    # Initialize app components
    initialize_app()

    # Run Flask app
    app.run(host='0.0.0.0', port=8080, debug=True, use_reloader=False)
```

~/Downloads/indexhehe.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smart Parking System</title>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.6.0/jquery.min.js">
</script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.4.1/socket.io.min.js">
</script>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css">
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="toast-container" id="toastContainer">
    </div>

    <div class="container">
        <header>
            <div class="logo">
                <i class="fas fa-parking"></i>
                <h1>Smart Parking System</h1>
            </div>
            <div class="status-indicator">
                <div class="indicator online" id="connectionStatus">
                    <i class="fas fa-circle"></i>
                    <span>Connected</span>
                </div>
                <div class="indicator" id="wifiStatus">
                    <i class="fas fa-wifi"></i>
                    <span>WiFi</span>
                </div>
            </div>
        </header>

        <div class="statistics">
            <div class="stat-card">
                <div class="stat-label">Available Slots</div>
                <div class="stat-value" id="availableSlots">--</div>
                <div class="stat-label">Total: 2</div>
            </div>
            <div class="stat-card">
                <div class="stat-label">Vehicles Today</div>
```

```
                <div class="stat-value" id="vehiclesCount">--</div>
                <div class="stat-label">Since midnight</div>
            </div>
            <div class="stat-card">
                <div class="stat-label">Gate Status</div>
                <div class="stat-value" id="gateStatusIndicator">--</div>
                <div class="stat-label" id="gateStatusTime">Last changed: --
</div>
            </div>
        </div>

        <div class="dashboard">
            <div class="card">
                <div class="card-header">
                    <div class="title">
                        <i class="fas fa-map-marker-alt"></i>
                        <span>Parking Slots</span>
                    </div>
                    <button class="btn btn-sm" id="refreshSlotsBtn">
                        <i class="fas fa-sync-alt"></i>
                    </button>
                </div>
                <div class="card-body">
                    <div class="slot-container">
                        <div class="slot free" id="slot1">
                            <div class="slot-number">1</div>
                            <div class="slot-status">
                                <i class="fas fa-check-circle"></i>
                                <span>Available</span>
                            </div>
                            <div class="slot-plate" id="slot1Plate">--</div>
                            <div class="controls">
                                <button class="btn btn-sm btn-warning bookBtn"
data-slot="1">Book</button>
                                <button class="btn btn-sm btn-danger cancelBtn"
data-slot="1" style="display: none;">Cancel</button>
                            </div>
                        </div>
                        <div class="slot free" id="slot2">
                            <div class="slot-number">2</div>
                            <div class="slot-status">
                                <i class="fas fa-check-circle"></i>
                                <span>Available</span>
                            </div>
                            <div class="slot-plate" id="slot2Plate">--</div>
                            <div class="controls">
                                <button class="btn btn-sm btn-warning bookBtn"
data-slot="2">Book</button>
```

```
                                <button class="btn btn-sm btn-danger cancelBtn"
data-slot="2" style="display: none;">Cancel</button>
                            </div>
                        </div>
                    </div>
                </div>

                <div class="card">
                    <div class="card-header">
                        <div class="title">
                            <i class="fas fa-door-open"></i>
                            <span>Gate Control</span>
                        </div>
                    </div>
                    <div class="card-body">
                        <div class="gate-status">
                            <div class="gate-indicator gate-closed"
id="gateIndicator">
                                <i class="fas fa-door-closed" id="gateIcon"></i>
                            </div>
                            <div class="gate-info">
                                <h3 id="gateStatusText">Gate Closed</h3>
                                <p id="gateMessage">No vehicle detected at
entrance</p>
                                <div class="gate-actions">
                                    <button class="btn btn-success"
id="openGateBtn">Open Gate</button>
                                <button class="btn btn-danger"
id="closeGateBtn">Close Gate</button>
                                </div>
                            </div>
                        </div>

                        <div class="camera-feed">
                            <img
src="https://www.circuitdigest.cloud/static/MxsfzpZCkxZa.jpeg"
                                id="cameraFeed"
                                alt="Camera Feed"
                                style="width: 100%; max-width: 100%; border-radius:
8px;">
                        </div>
                    </div>
                </div>

                <div class="card">
                    <div class="card-header">
                        <div class="title">
                            <i class="fas fa-history"></i>
```

```
                        <span>System Logs</span>
                    </div>
                    <button class="btn btn-sm" id="clearLogsBtn">
                        <i class="fas fa-eraser"></i>
                    </button>
                </div>
                <div class="card-body">
                    <div class="tab-container">
                        <div class="tab-buttons">
                            <button class="tab-btn active" data-tab="serialLogs">Serial Logs</button>
                            <button class="tab-btn" data-tab="systemLogs">System Events</button>
                            <button class="tab-btn" data-tab="plateDetections">Plate Detections</button>
                        </div>

                        <div class="tab-content active" id="serialLogs">
                            <div class="serial-log" id="serialLogContainer">
                                <div class="log-entry">Waiting for serial communication...</div>
                            </div>
                        </div>

                        <div class="tab-content" id="systemLogs">
                            <div class="recent-events" id="systemEventsContainer">
                                <div class="event-item">
                                    <div class="event-info">
                                        <div class="event-type">System Started</div>
                                        <div class="event-data">Application initialized</div>
                                    </div>
                                    <div class="event-time">Just now</div>
                                </div>
                            </div>
                        </div>

                        <div class="tab-content" id="plateDetections">
                            <div class="recent-events" id="plateDetectionsContainer">
                                <div class="event-item">
                                    <div class="event-info">
                                        <div class="event-type">No plates detected yet</div>
                                    </div>
                                </div>
```

```
                            </div>
                        </div>
                    </div>
                </div>
            </div>

            <div class="card">
                <div class="card-header">
                    <div class="title">
                        <i class="fas fa-sliders-h"></i>
                        <span>System Control</span>
                    </div>
                </div>
                <div class="card-body">
                    <div class="form-group">
                        <label for="commandInput">Send Command to Arduino:
</label>
                        <div style="display: flex; gap: 10px;">
                            <input type="text" class="form-control"
id="commandInput" placeholder="Enter command...">
                            <button class="btn btn-primary"
id="sendCommandBtn">Send</button>
                        </div>
                    </div>

                    <div style="display: grid; grid-template-columns: 1fr 1fr;
gap: 10px; margin-bottom: 15px;">
                        <div>
                            <label for="buzzerDuration">Buzzer Duration (ms):
</label>
                            <div style="display: flex; gap: 10px;">
                                <input type="number" class="form-control"
id="buzzerDuration" value="1000" min="100" max="5000">
                                <button class="btn btn-warning"
id="activateBuzzerBtn">Buzz</button>
                            </div>
                        </div>
                        <div>
                            <label for="ledToggle">Toggle LED:</label>
                            <div style="display: flex; gap: 10px;">
                                <select class="form-control" id="ledToggle">
                                    <option value="20">Slot 1 - Red LED</option>
                                    <option value="2">Slot 1 - Green
LED</option>
                                    <option value="35">Slot 2 - Red LED</option>
                                    <option value="36">Slot 2 - Green
LED</option>
                                </select>
                                <button class="btn"
```

```
id="toggleLedBtn">Toggle</button>
                            </div>
                        </div>
                    </div>

                    <div style="display: grid; grid-template-columns: 1fr 1fr; gap: 10px;">
                        <button class="btn btn-primary" id="getStatusBtn">
                            <i class="fas fa-sync-alt"></i> Get System Status
                        </button>
                        <button class="btn btn-danger" id="resetArduinoBtn">
                            <i class="fas fa-power-off"></i> Reset Arduino
                        </button>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <!-- Booking Modal -->
    <div id="bookingModal" style="display: none; position: fixed; z-index: 100; left: 0; top: 0; width: 100%; height: 100%; background-color: rgba(0,0,0,0.5);">
        <div style="background-color: white; margin: 15% auto; padding: 20px; border-radius: 8px; width: 80%; max-width: 500px;">
            <h3>Book Parking Slot <span id="bookingSlotNumber"></span></h3>
            <div class="form-group">
                <label for="licensePlateInput">License Plate Number:</label>
                <input type="text" class="form-control" id="licensePlateInput" placeholder="Enter license plate...">
            </div>
            <div style="display: flex; justify-content: flex-end; gap: 10px; margin-top: 20px;">
                <button class="btn" id="cancelBookingModalBtn">Cancel</button>
                <button class="btn btn-success" id="confirmBookingBtn">Book Slot</button>
            </div>
        </div>
    </div>

    <footer>
        Smart Parking System © 2025 | Edge Device: <span id="edgeDeviceStatus">Connected</span>
    </footer>

    <script src="script.js"></script>
</body>
</html>
```

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