DESIGN AND IMPLEMENTATION OF AN APPLICATION FOR SMART PARKING MANAGEMENT

Schifîrneț Petre-Iustin

Petroleum-Gas University of Ploiești
Faculty of Mechanical and Electrical Engineering – Applied Informatics and Automation

This project addresses two common issues in urban parking: the lack of real-time information about available parking spots and the difficulty of locating the exact available one. The proposed solution consists of a smart parking system based on IR sensors, an ESP32 microcontroller, and two web applications: one allows users to view available spots and report problems, while the other enables administrators to control the barriers both automatically and manually via Firebase.

Introduction

Parking is often seen as a frustrating part of daily urban life. Drivers frequently waste time circling around crowded lots, hoping to find a free spot. Even when digital signs indicate that spaces are available, locating the exact free spot can be nearly impossible, especially in large parking areas. These issues lead to delays, stress, traffic congestion, and in some cases, missed appointments or work hours (4).

This project aims to address these everyday frustrations by designing and implementing an intelligent parking management system. The solution provides real-time information about the availability of parking spots and automates barrier control to improve flow and reduce human intervention (3),(6).

At the core of the system is an ESP32 microcontroller, which collects data from infrared (IR) sensors installed in each parking space. The ESP32 processes the data and sends a binary code representing the state of each spot to a Firebase Realtime Database via Wi-Fi. This allows seamless synchronization with two dedicated web applications: one for users to view parking availability and report issues, and another for administrators to manually control the barriers and monitor the system (1).

Hardware System Desing

The hardware structure of the smart parking system consists of several electronic components working together to detect the presence of vehicles, control access through entry and exit barriers, and exchange data with an online database.

The central component is the ESP32 microcontroller. It features multiple digital and analog input/output pins and built-in Wi-Fi connectivity (7). The ESP32 continuously monitors the state of the connected sensors, processes their input, and transmits structured data to the Firebase Realtime Database. It also receives commands from the administrator interface and triggers actions on the servomotors controlling the barriers. Internally, each pin of the ESP32 reads digital HIGH or LOW signals depending on the voltage it receives from external components.

Each of the five parking spots is equipped with an infrared (IR) proximity sensor, which consists of an IR LED (emitter) and a phototransistor (receiver). When a vehicle is parked in front of the sensor, the infrared light emitted by the LED is reflected back toward the phototransistor. This reflection alters the current flowing through the phototransistor, pulling the output pin of the sensor to a LOW voltage level (typically under 1V) (2). The ESP32 detects this change on the corresponding digital input pin and registers the spot as occupied.

To prevent false readings caused by brief interferences such as people passing by or shadows, a simple method is implemented: the ESP32 checks if the signal remains in the LOW state for at least one second before confirming that the spot is truly occupied. If the signal returns to HIGH too quickly, it is ignored.

In addition to these five sensors, two more IR sensors are placed near the entry and exit points of the parking lot. These work similarly and are used to detect the presence of vehicles approaching the barriers. When the entry sensor detects a car, the ESP32 sends a signal to the corresponding servo motor to rotate to 90 degrees, lifting the barrier. Once the vehicle has passed and the sensor no longer detects an object, the ESP32 initiates a delay of one second, after which it sends a new signal to return the servo to 0 degrees, lowering the barrier. This delay ensures that the barrier does not close prematurely while the car is still passing.

The system uses two servo motors — one for the entrance and one for the exit. Each motor is connected to a separate PWM-capable pin on the ESP32. The ESP32 sends position values to the servo through pulse-width modulation (PWM) signals. A pulse of around 1.5 milliseconds corresponds to the neutral (midpoint) position, while longer or shorter pulses rotate the motor shaft to a specific angle. In this implementation, 0 degrees corresponds to the closed position of the barrier, and 90 degrees to the open position. For one of the motors, the orientation required inverting the logic due to its physical mounting.

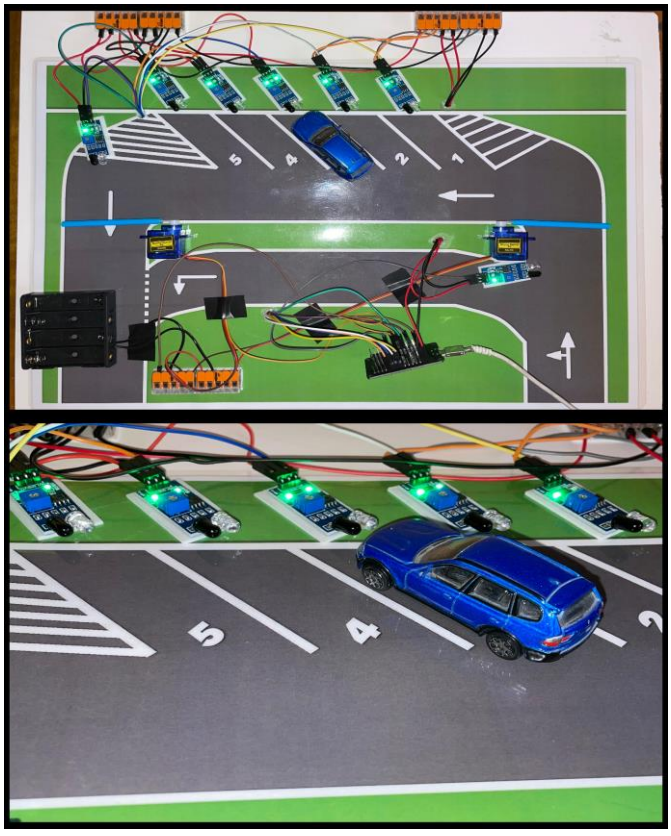

Figure 1. Physical Model - Practical Implementation

Electrical Connections

The power distribution for the system is designed to be modular and effective in supporting all connected components. The ESP32 microcontroller provides a 3.3V output and a common ground (GND), which are routed to a set of terminal blocks (WAGO-type). These connectors serve as distribution hubs, allowing the same voltage and ground lines to be shared efficiently across multiple devices without soldering or complex wiring. This setup simplifies maintenance and offers flexibility in future expansions or hardware replacements.

All seven IR sensors in the system, five for the individual parking spots and two for the entrance and exit proximity detection, are powered directly through these terminal connectors using the regulated 3.3V output of the ESP32. As illustrated in Figure 1, this layout enables a clean and structured organization of the power lines.

The control wires, which carry the digital output signals from each IR sensor, are routed discretely underneath the surface of the physical parking model. This decision was made to optimize the visual appearance and minimize physical interference with other components. Each of these signal wires is connected to a dedicated digital input pin on the ESP32, allowing individual monitoring of each sensor's state.

The servo motors used to actuate the entry and exit barriers operate at a higher current and voltage than the sensors, and therefore require a separate power source. A 6V DC supply is provided via a battery holder containing four AA batteries, delivering both sufficient voltage and current capacity to drive the motors smoothly. This design avoids drawing large currents from the ESP32's onboard voltage regulator, which could cause system instability or resets during motor operation (5). The servo motors share a common ground with the ESP32 to maintain proper signal reference, ensuring consistent response to the PWM control signals.

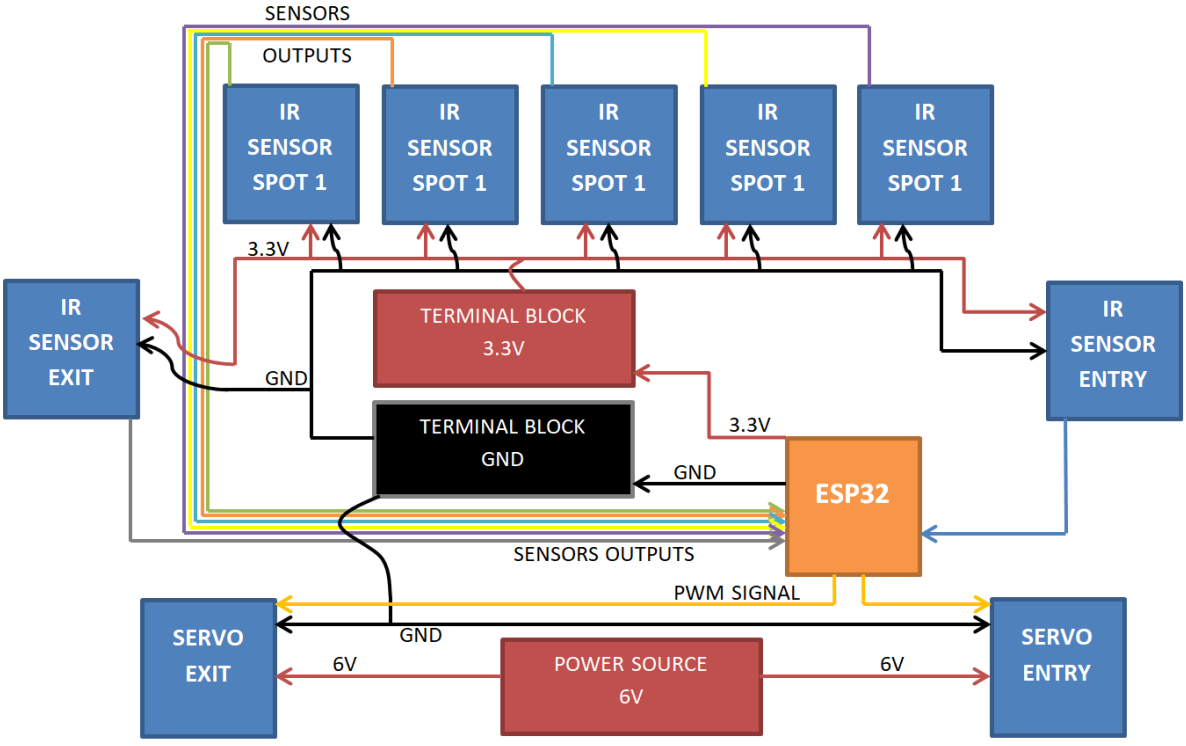

Figure 2. Electrical Schematic of the System

To provide a better understanding of the hardware implementation, Table 1 presents the complete list of components and their corresponding connections to the ESP32 microcontroller.

Table 1. Hardware Connections

Component	ESP32 Pin	Signal Type	Voltage
IR Sensor 1 (Parking Spot 1)	5	Digital Input	3.3V
IR Sensor 2 (Parking Spot 2)	18	Digital Input	3.3V
IR Sensor 3 (Parking Spot 3)	19	Digital Input	3.3V
IR Sensor 4 (Parking Spot 4)	21	Digital Input	3.3V
IR Sensor 5 (Parking Spot 5)	27	Digital Input	3.3V
IR Sensor - Entry	32	Digital Input	3.3V
IR Sensor - Exit	23	Digital Input	3.3V
Servo Motor - Entry Barrier	25	PWM Output	6V (external)
Servo Motor - Exit Barrier	26	PWM Output	6V (external)

Software System Design

The software architecture of the smart parking system is based on the interaction between a microcontroller, a real-time cloud database, and two separate web applications. All components communicate through HTTP requests and real-time synchronization mechanisms to ensure data is updated efficiently and consistently.

The central element of the system's data flow is the Firebase Realtime Database. It acts as the bridge between hardware and software, storing all relevant information in real time (9). The ESP32 microcontroller writes data to Firebase every two seconds, sending the binary code that represents the state of the five parking spots. Additionally, it reads specific values from Firebase that determine the current mode of operation ("automatic" or "manual") and, if in manual mode, listens for commands to raise or lower the entry and exit barriers. Firebase also receives reports from users and displays them in the administrator interface. Communication between the ESP32 and Firebase is made via HTTPS requests using a secret authorization key.

The two web interfaces (user and admin) are hosted on GitHub Pages, providing a simple and reliable deployment method (8). The websites use JavaScript to send and retrieve data from Firebase via RESTful API calls (10). Data is fetched at a controlled interval (e.g., every 1 or 3 seconds), and optimizations are implemented to ensure that requests are only made while the browser tab is active, reducing unnecessary traffic and staying within Firebase's daily quota limits.

The interaction between hardware components, the Firebase database, and the two web interfaces is summarized in Figure 2, which illustrates the system's overall functionality and data flow.

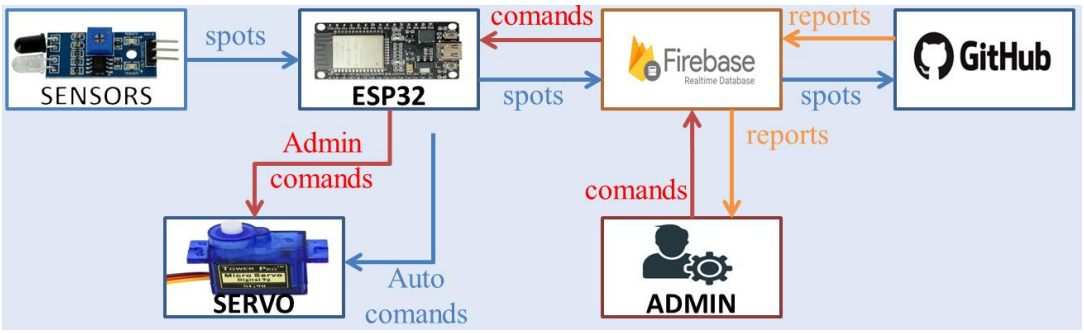

Figure 3. System Functionality Diagram

The user interface is focused on accessibility and real-time visualization. It displays the parking layout with live updates showing which spots are currently occupied. Changes in the parking state are highlighted using notifications, and the number of available spots is shown at the top of the screen. The interface is designed to be fully responsive, adapting its layout and design depending on the screen size, allowing users to access the system comfortably from both desktop and mobile devices. Additionally, the interface includes a simple report form that allows users to report issues such as blocked spots or malfunctioning barriers. These reports are sent to Firebase and displayed for the administrator.

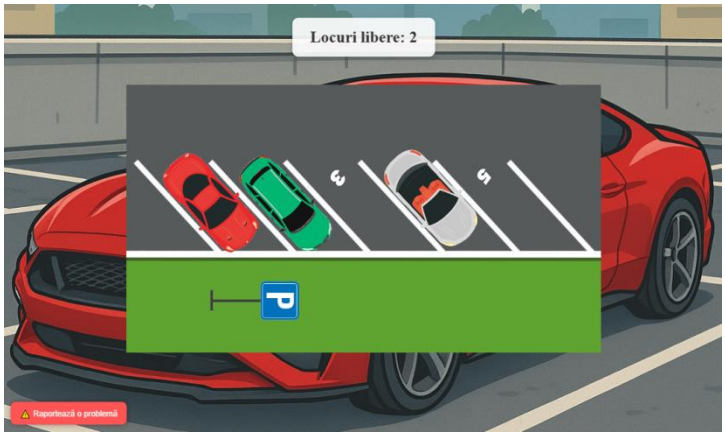

Figure 4. Desktop Application Interface

Figure 5. Mobile Application Interface

The admin interface provides full control and monitoring capabilities. Administrators can switch between automatic and manual operation modes, and in manual mode, can directly control the entry and exit barriers using action buttons. All received user reports are displayed on the admin page, along with timestamps, and can be cleared after being handled. The interface is password-protected and styled for clarity, making it easy to operate from any device.

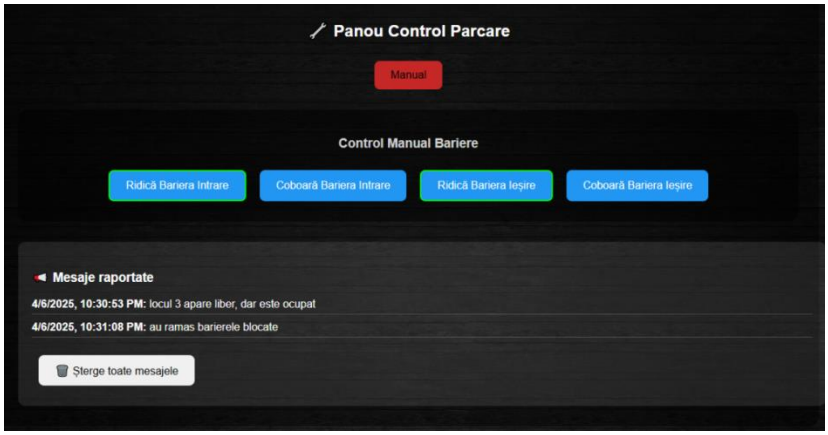

Figure 6. Administration Page

Conclusion

This project demonstrates a functional and scalable solution to real-world parking challenges by combining simple hardware with cloud-based services and intuitive web interfaces. By using infrared sensors, an ESP32 microcontroller, Firebase, and GitHub Pages, the system provides real-time parking availability updates and remote control of access barriers. The modular design, low cost, and potential for future improvements, such as AI-based detection and online reservations, make this implementation both practical and adaptable for smart city applications.

Bibliography

1. A. Khanna, R. Anand (2016), *IoT based smart parking system*. In: 2016 International Conference on Internet of Things and Applications (IOTA), pp. 266-270.
2. Bucur, G. (2010), Tehnici de măsurare. Senzori şi traductoare de mărimi neelectrice. *Sisteme pentru măsurat mărimi neelectrice*, Ploieşti, Editura UPG.
3. Sadhukhan, Pampa (2017), *An IoT-based E-parking system for smart cities*. In: International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1062-1066.
4. Sudheer Hanumanthakari (2024), *Intelligent and real-time Parking System*. In: E3S Web of Conferences 472, 03003.
5. T. D. Burd, R. W. Brodersen (2012), *Energy Efficient Microprocessor Design*, Springer.
6. Y. Agarwal, P. Ratnani, U. Shah and P. Jain (2021), *IoT based Smart Parking System*. In: 5th International Conference on Intelligent Computing and Control Systems (ICICCS).
7.***.Espressif Systems (2023), *ESP32 Technical Reference Manual*, https://www.espressif.com
8.***GitHub (2025), *GitHub Pages Documentation*, https://docs.github.com/en/pages
9.***Google Firebase (2025), *Firebase Realtime Database Documentation*, https://firebase.google.com/docs/database.
10.***Mozilla Developer Network (2025), *JavaScript Fetch API Guide*, https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.