Smart Car Parking System with Sensor-Based Slot Detection

Nalan Gelir - 20200808030

May 21, 2025

Project Objective

Background and Motivation

In many urban areas, drivers waste significant time and fuel searching for available parking spots. This inefficiency leads to traffic congestion, increased emissions, and user frustration. The motivation behind this project is to design a smart parking system that detects available parking slots in real-time and assists drivers in locating vacant spots efficiently.

Problem Definition and Relevance

Traditional parking systems are often manual or lack real-time monitoring capabilities. This project addresses the need for a scalable, automated, and sensor-based solution using ESP32 microcontrollers, enabling seamless communication between hardware and a cloud-based backend system for real-time data management.

Functionalities

Vehicle Detection Using Ultrasonic Sensors

Each parking slot is equipped with an HC-SR04 ultrasonic sensor connected to an ESP32 development board. These sensors measure the distance to detect the presence of a vehicle. If the distance falls below a predefined threshold, the ESP32 classifies the slot as *occupied*; otherwise, it remains *vacant*.

Real-Time Monitoring and IoT Central Integration

The ESP32 device continuously monitors sensor readings and publishes the updated slot status to **Azure IoT Central**. The IoT Central App acts as the cloud bridge, enabling centralized real-time data collection, visualization, and rule-based automation. When a change in parking status is detected, a rule in IoT Central exports this update and pushes it to an **Azure Service Bus Queue**.

Event-Driven Backend Updates via Azure Function
A serverless **Azure Function** is configured to listen to the Service Bus Queue. When a new message is received (indicating a slot state change), the function parses the data and updates the corresponding record in the backend database. This design ensures loose coupling between the edge device, the cloud, and the backend logic, improving reliability and scalability.

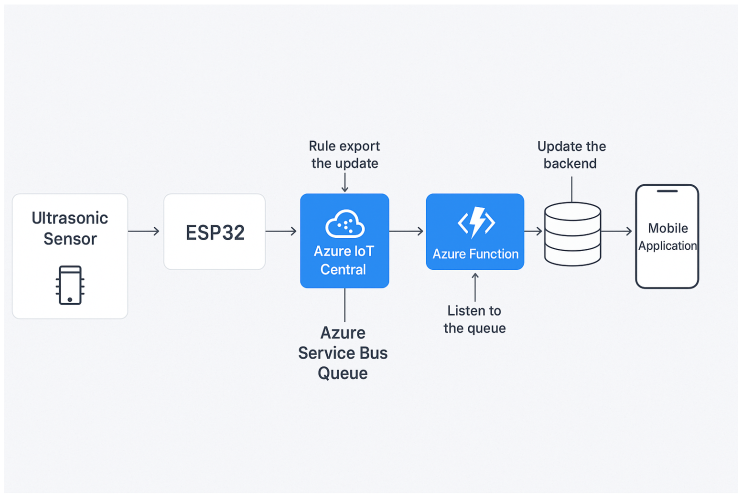


Figure 1: Data Flow Diagram

Duration Tracking and Fee Calculation
When a vehicle occupies a slot, a new parking session is initiated, and the start timestamp is recorded. Upon vacancy detection, the session is closed, and the total duration is calculated. Based on this duration, the system computes the parking fee using pre-configured pricing rules in the backend. The calculated fee and session metadata are stored for later user access and reporting.

Data Processing and Storage (Backend)
All data is processed by a backend server developed using C# (.NET). The backend manages session logic, handles pricing calculations, and updates the status of each parking slot in a **Microsoft SQL Server** database. The system is designed to handle multiple concurrent updates securely and efficiently using entity validation and transactional operations.

User Interaction via Mobile Application
The mobile app, built with React Native and Expo, provides a user-friendly interface that allows drivers to:

- View the availability of parking slots in real-time

- Track their ongoing parking duration and fee
- Review past parking sessions and associated payments

The app fetches data from the backend via secure RESTful API endpoints.

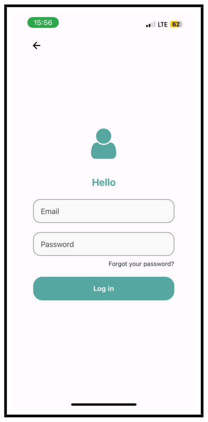


Figure 2: *
Screen 1

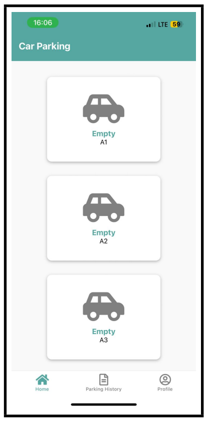


Figure 3: *
Screen 2

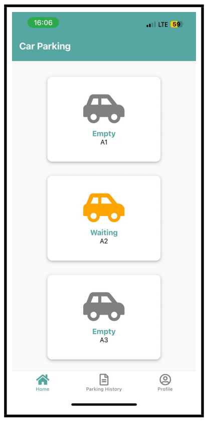


Figure 4: *
Screen 3

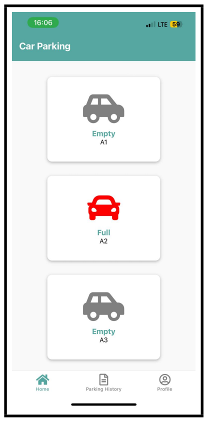
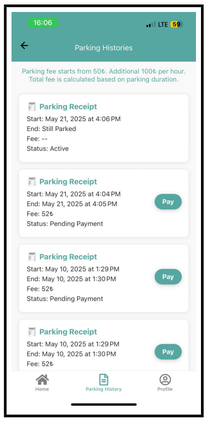


Figure 5: *
Screen 4



Parking fee starts from 50₺. Additional 100₺ per hour.
Total fee is calculated based on parking duration.

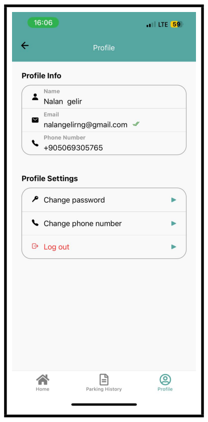
Parking Receipt
Start: May 21, 2025 at 4:06 PM
End: Still Parked
Fee: --
Status: Active

Parking Receipt
Start: May 21, 2025 at 4:04 PM
End: May 21, 2025 at 4:05 PM
Fee: 52₺
Status: Pending Payment

Parking Receipt
Start: May 10, 2025 at 1:29 PM
End: May 10, 2025 at 1:30 PM
Fee: 52₺
Status: Pending Payment

Parking Receipt
Start: May 10, 2025 at 1:29 PM
End: May 10, 2025 at 1:30 PM
Fee: 52₺

Figure 6: *
Screen 5



Profile Info

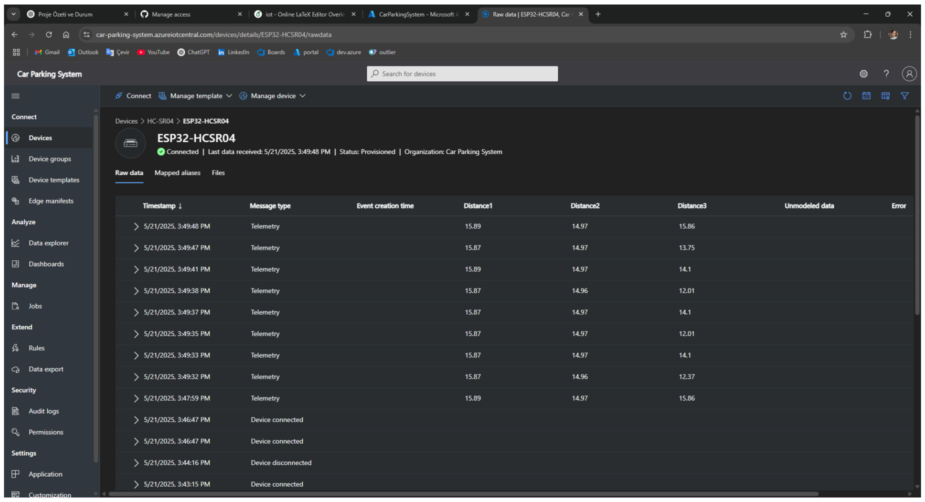
Name
Nalan gelir
Email
nalangelirng@gmail.com
Phone Number
+905069305765

Profile Settings

Figure 7: *
Screen 6

Communication Protocols

The ESP32 communicates with Azure IoT Central using the MQTT protocol. Internally, the backend and Azure Function use HTTP and JSON-based messaging to maintain loosely coupled services. Data flows are event-driven and optimized for minimal latency.



ESP32-HCSR04

Connected | Last data received: 5/21/2025, 3:49:48 PM | Status: Provisioned | Organization: Car Parking System

Timestamp ↓	Message type	Event creation time	Distance1	Distance2	Distance3	Unmodeled data	Error
5/21/2025, 3:49:48 PM	Telemetry		15.89	14.97	15.86		
5/21/2025, 3:49:47 PM	Telemetry		15.87	14.97	13.75		
5/21/2025, 3:49:41 PM	Telemetry		15.89	14.97	14.1		
5/21/2025, 3:49:38 PM	Telemetry		15.87	14.96	12.01		
5/21/2025, 3:49:37 PM	Telemetry		15.87	14.97	14.1		
5/21/2025, 3:49:35 PM	Telemetry		15.87	14.97	12.01		
5/21/2025, 3:49:33 PM	Telemetry		15.87	14.97	14.1		
5/21/2025, 3:49:32 PM	Telemetry		15.87	14.96	12.37		
5/21/2025, 3:47:59 PM	Telemetry		15.89	14.97	15.86		
5/21/2025, 3:46:47 PM	Device connected						
5/21/2025, 3:46:47 PM	Device connected						
5/21/2025, 3:44:16 PM	Device disconnected						
5/21/2025, 3:43:15 PM	Device connected						

Figure 8: Telemetry Result

Cloud Interaction and Scalability

The architecture is cloud-native and designed for scalability. All messages are decoupled via Azure Service Bus, allowing for horizontal scaling of backend services. IoT Central allows for centralized monitoring of multiple devices, while the use of serverless Azure Functions reduces operational complexity.

Security Considerations

Security is ensured at multiple layers:

- Sensor-to-cloud communication is protected via Azure IoT Device credentials.
- Azure Function endpoints are secured via managed identities and access control.
- Backend APIs implement validation, authentication mechanisms, and secure database access.

System Architecture

The system consists of the following main components:

1. **Sensor Layer (HC-SR04)**
   Ultrasonic sensors installed at each parking spot continuously measure the distance to detect the presence of the vehicle.
2. **Microcontroller Unit (ESP32)**
   Each sensor is connected to an ESP32 microcontroller that reads sensor data and transmits it wirelessly to the back-end server.

3. **RGB LED Indicator**
   RGB LEDs provide visual feedback on parking spot status using different colors (e.g. green for vacant, red for occupied, yellow for reserved).

4. **LCD Display**
   The LCD displays real-time parking spot status.

5. **Backend Server (C# .NET)**
   Developed using C# on the .NET platform, the backend handles incoming sensor data, manages business logic, calculates parking fees, maintains user sessions and exposes RESTful APIs for communication with the mobile app.

6. **Database (Microsoft SQL Server)**
   Microsoft SQL Server securely stores user profiles, parking session histories, and other metadata. The database is optimized for performance and scalability.

7. **Mobile Application (React Native with Expo)** A cross-platform mobile app built with the React Native and Expo framework provides users with availability of parking slots in real time, tracking of parking sessions, overview of fees, and historical data through secure API communication with the back end.

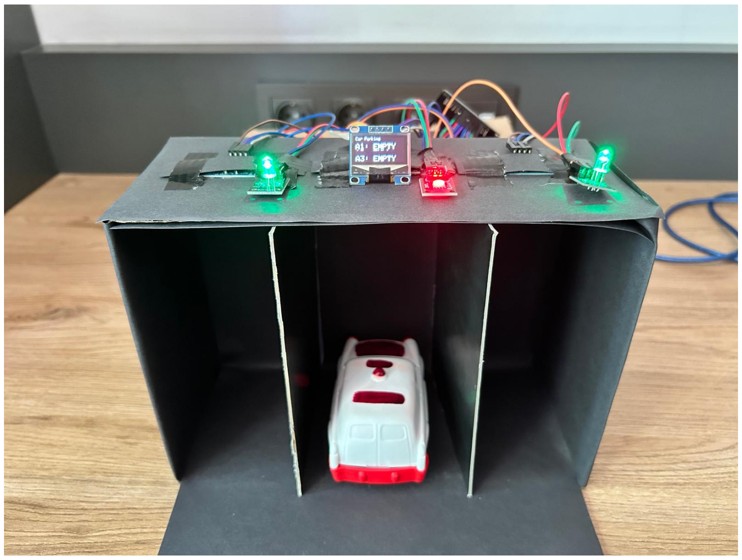


Figure 9: Park Area

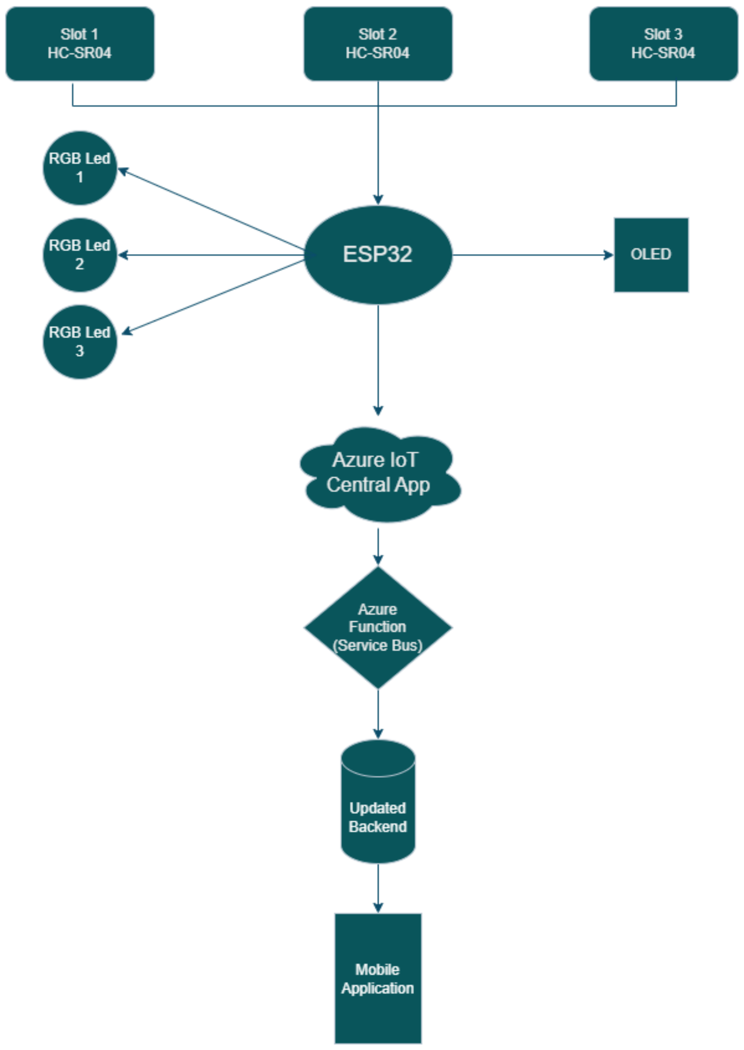


Figure 10: System Architecture

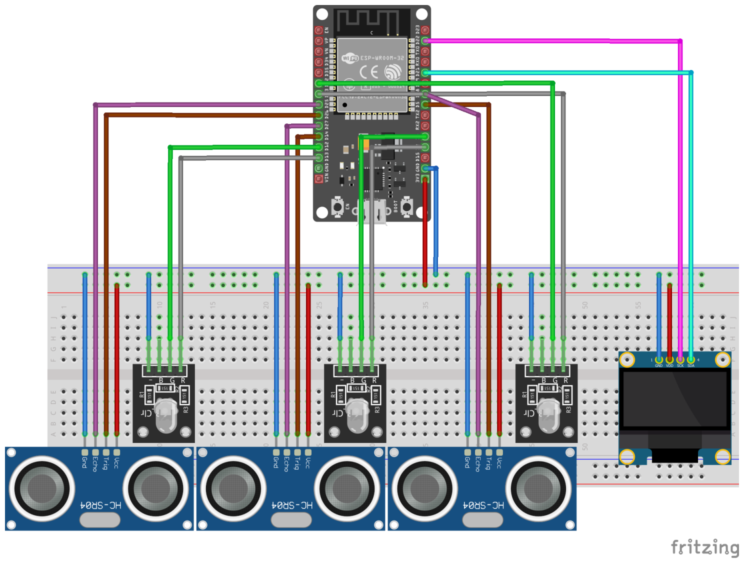


Figure 11: Circuit Scheme

Expected Benefits

- Users will quickly locate available parking spaces.
- Parking operations will become more automated and accurate.
- Parking fees will be calculated precisely based on usage.
- The system will be adaptable to multi-floor and large-scale implementations.

Conclusion and Future Work

In this project, an IoT-based solution was successfully developed to monitor and control environmental conditions using ESP32 and various sensors. The system architecture, integrating hardware components, cloud services, and communication protocols, enabled real-time data acquisition and remote control effectively. The implementation demonstrated that the chosen sensors and communication methods, particularly MQTT and Azure IoT Central, provided reliable performance and seamless connectivity.

The results showed that the system can accurately collect sensor data and trigger actuators based on predefined thresholds, confirming the practical applicability of the design. However, some limitations were identified, such as occasional network latency affecting real-time responsiveness and the need for enhanced security measures to protect data integrity and user privacy.

For future work, several improvements are recommended. Firstly, integrating advanced data analytics and machine learning algorithms could enhance predictive capabilities and automate decision-making. Secondly, expanding the range of supported sensors and actuators would increase the system's versatility for broader applications. Additionally, implementing stronger security protocols, including end-to-end encryption and robust authentication, will be essential for deployment in sensitive environments. Finally, developing a more user-friendly mobile application and incorporating energy-efficient hardware designs would further improve usability and sustainability.

Overall, this project lays a solid foundation for scalable and secure IoT solutions tailored to real-world monitoring and control scenarios, and the suggested enhancements open avenues for continued development and innovation.

References

[1] Smart Car Parking System using Internet of Things (IoT), Arduino, *YouTube Video*, Available at: `https://www.youtube.com/watch?v=aFN77gNgRzQ`, Accessed May 2025.

[2] ESP32 With Azure IoT — Control Led by Azure Direct Methods, *YouTube Video*, Available at: `https://www.youtube.com/watch?v=HK8da4DFgHc&t=202s`, Accessed May 2025.

GitHub Repository

You can access the project's GitHub repository from the link below:

GitHub - Car Parking System