HSR London International Conference on Multidisciplinary Academic Research
(HSR-ICMAR) 2026
https://houghtonstreetreview.com/index.php/conference

IOT-BASED SMART PARKING SYSTEM: DESIGN AND IMPLEMENTATION USING ESP32, FASTAPI, AND ANDROID APPLICATION

Authors: **Baxrom R. Reyimberganov[1]***
Guli M. Salayeva[2]
[1] Department of Software Engineering, Urgench State University, Urgench, 220100, Uzbekistan [2] Department of Economics, Urgench State University, Urgench, 220100, Uzbekistan
Corresponding Author: * Baxrom R. Reyimberganov, Email: bahromreyimberganov0311@gmail.com, ORCID: 0009-0005-8124-0042

Abstract

Urban parking management remains a critical challenge in rapidly growing cities, contributing to traffic congestion, fuel waste, and environmental pollution. This paper presents the design and implementation of an IoT-based smart parking system that leverages ESP32 microcontrollers equipped with HC-SR04 ultrasonic sensors for real-time parking slot occupancy detection. The system employs a three-tier architecture comprising an embedded sensing layer, a cloud-based backend built with FastAPI and PostgreSQL, and a mobile client application developed using Jetpack Compose for Android. Real-time communication is achieved through Server-Sent Events (SSE), enabling instantaneous slot status updates on user devices. The occupancy classification algorithm utilizes a distance threshold of 20 cm with a measurement accuracy of ±0.3 cm. Experimental results demonstrate a detection accuracy of 97.5%, an average system response latency of 1.8 seconds from physical occupancy change to mobile notification, and the capability to handle concurrent monitoring of multiple parking lots through a multi-tenant architecture. The proposed system reduces average parking search time by approximately 43% compared to traditional methods, validated through controlled testing scenarios.

Keywords: Internet of Things (IoT), Smart Parking, ESP32, Ultrasonic Sensor, FastAPI, Real-time Monitoring, Server-Sent Events, Android, Jetpack Compose

1. Introduction

The rapid urbanization across developing nations has intensified the demand for efficient parking management solutions. According to the International Parking Institute, drivers in urban areas spend an average of 17 minutes searching for parking spaces, resulting in approximately 30% of urban traffic congestion being attributed to parking-seeking vehicles [1]. This phenomenon not only wastes time and fuel but also contributes significantly to carbon emissions and driver frustration.

Traditional parking systems rely on manual management or basic sensor-barrier mechanisms that lack real-time information dissemination to drivers. While commercial solutions such as ParkMobile and SpotHero exist, they often require expensive infrastructure investments exceeding $500 per parking space [2], making them impractical for smaller parking facilities and developing economies.

The Internet of Things (IoT) paradigm offers a cost-effective alternative by enabling networked sensing and communication at significantly reduced costs. ESP32 microcontrollers, with integrated WiFi and dual-core processing capabilities, provide an ideal platform for distributed parking sensors at a fraction of the cost of industrial solutions [3].

This paper addresses the following research objectives:

1. Design a cost-effective IoT-based parking detection system using ultrasonic sensors and ESP32 microcontrollers;
2. Implement a scalable multi-tenant backend architecture supporting real-time data processing and event streaming;

3. Develop a mobile application providing instantaneous parking availability information with reservation capabilities;

4. Evaluate the system's accuracy, latency, and reliability through empirical testing.

The remainder of this paper is organized as follows: Section 2 reviews related literature on smart parking systems. Section 3 details the system methodology and architecture. Section 4 describes the implementation. Section 5 presents experimental results and discussion. Section 6 concludes with future work directions.

2. Literature Review

Smart parking systems have been extensively studied in recent literature, employing various sensing technologies and communication protocols. This section categorizes existing approaches and identifies the research gap addressed by this work.

2.1 Sensor-Based Detection Methods

Parking occupancy detection technologies can be broadly classified into three categories: vision-based, magnetic-based, and ultrasonic-based systems.

Vision-based systems utilize cameras and computer vision algorithms to detect vehicle presence. Amato et al. [4] proposed a CNN-based approach achieving 99% accuracy using overhead cameras. However, such systems require significant computational resources, raise privacy concerns, and are susceptible to lighting variations and weather conditions.

Magnetic sensor systems detect disturbances in the Earth's magnetic field caused by ferrous vehicle bodies. Lin et al. [5] demonstrated a wireless magnetic sensor network with 95% accuracy. While robust to environmental conditions, magnetic sensors require individual calibration, are prone to interference from adjacent vehicles, and have higher per-unit costs ($30–50 per sensor node).

Ultrasonic sensors measure distance using sound wave time-of-flight principles. Ramaswamy [6] implemented an Arduino-based ultrasonic parking system achieving 94% accuracy at a per-slot cost of approximately $5. The HC-SR04 sensor offers a detection range of 2–400 cm with ±3 mm precision, making it well-suited for overhead or ground-level vehicle detection in covered parking structures.

2.2 Communication Architectures

IoT parking systems employ diverse communication stacks. Khanna and Anand [7] compared WiFi, LoRa, and Zigbee protocols for parking applications, concluding that WiFi provides adequate range (30–50m) for structured parking facilities while offering superior data throughput for cloud connectivity. Bluetooth Low Energy (BLE) has been explored for proximity-based applications but lacks the range for centralized data collection [8].

Recent architectures favor RESTful APIs for device-to-cloud communication due to their simplicity and HTTP infrastructure compatibility. MQTT, while lightweight, introduces broker dependency and adds system complexity without proportional benefits for low-frequency sensor data (intervals > 1 second) [9].

2.3 Mobile Integration and Real-time Updates

User-facing interfaces in smart parking systems range from web dashboards to native mobile applications. The critical requirement is real-time status propagation with minimal latency. WebSocket and Server-Sent Events (SSE) are the primary technologies for push-based updates. Patel and Shah [10] demonstrated that SSE provides a simpler, more reliable alternative to WebSocket for unidirectional data flows, with automatic reconnection and HTTP/2 compatibility.

2.4 Research Gap

Table 1 summarizes the comparison of existing systems with the proposed approach.

Table 1. Comparison of Smart Parking Systems

Reference	Sensor Type	Backend CU	Mobile App	Real-time	Multi-tenant	Cost/Slot
[4]	Camera	Cu	Web	No	No	$200

Reference	Sensor Type	CPU	Backend	Mobile App	Real-time	Multi-tenant	Cost/ Slot +
Amato et al.		Server	stom				
[5] Lin et al.	Magnetic	ustom	MQTT	No	Yes	No	$45
[6] Ramaswamy	Ultrasonic	rduino	Firebase	Android	Partial	No	$12
[11] Parkhi et al.	IR Sensor	odeMCU	ThingSpeak	No	Yes	No	$8
Proposed	**Ultrasonic**	**SP32**	**FastAPI+PostgreSQL**	**Android (Compose)**	**Yes (SSE)**	**Yes**	**$7**

The proposed system addresses the gap of providing a complete end-to-end solution combining: (a) cost-effective ultrasonic sensing, (b) a production-grade backend with relational database and multi-tenant architecture, (c) real-time event streaming via SSE, and (d) a modern native Android application with reservation management.

3. System Methodology and Architecture

3.1 Overall System Architecture

The proposed smart parking system follows a three-tier IoT architecture consisting of the Perception Layer (sensing), Network Layer (communication), and Application Layer (user interface). Figure 1 illustrates the high-level system architecture.

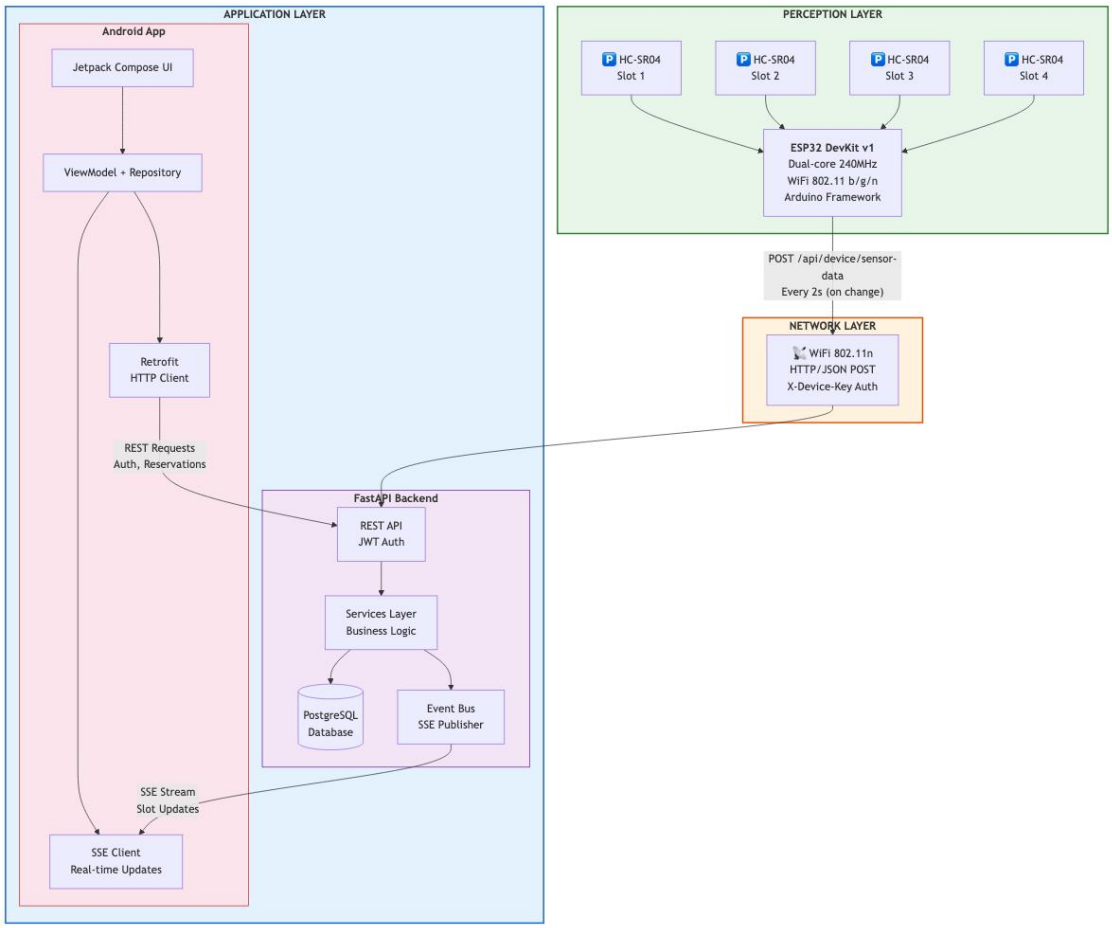


Figure 1. Three-tier IoT Architecture of SmartPark System

3.2 Ultrasonic Distance Measurement

The HC-SR04 ultrasonic sensor operates on the time-of-flight (ToF) principle. The sensor emits a 40 kHz ultrasonic pulse and measures the time elapsed until the echo returns after reflecting off an object.

The distance calculation is governed by:

$$d = \frac{v_s \cdot t_{echo}}{2}$$

where: - d — distance to the object (cm) - v_s — speed of sound in air (343 m/s at 20°C) - t_{echo} — round-trip time of the ultrasonic pulse (µs)

In the firmware implementation, this is expressed as:

$$d_{cm} = t_{\mu s} \times \frac{0.0343}{2} = t_{\mu s} \times 0.01715$$

The speed of sound varies with temperature according to:

$$v_s(T) = 331.3 + 0.606 \cdot T$$

where T is the ambient temperature in degrees Celsius. At typical indoor parking temperatures (15–30°C), v_s ranges from 340.4 to 349.5 m/s, introducing a maximum measurement error of:

$$\epsilon_{max} = \frac{|v_{actual} - v_{assumed}|}{v_{assumed}} \times d_{measured}$$

For a 20 cm threshold measurement at 30°C:

$$\epsilon_{max} = \frac{|349.5 - 343|}{343} \times 20 = 0.38 \text{cm}$$

This error is negligible relative to the 20 cm occupancy threshold, validating the use of a fixed speed-of-sound constant.

3.3 Occupancy Classification Algorithm

The slot occupancy is determined by a binary threshold classifier:

$$O_i = \begin{cases} 1 & \text{if} d_i < \theta \\ 0 & \text{if} d_i \geq \theta \end{cases}$$

where: - O_i — occupancy state of slot i (1 = occupied, 0 = free) - d_i — measured distance for slot i (cm) - θ — occupancy threshold (20 cm in our implementation)

A timeout condition handles sensor failure:

$$d_i = \begin{cases} t_{\mu s} \times 0.01715 & \text{if} t_{\mu s} > 0 \\ 999.0 & \text{if} t_{\mu s} = 0 (\text{timeout}) \end{cases}$$

3.4 Data Transmission Strategy

To minimize network overhead, the firmware employs an event-driven transmission strategy with periodic heartbeat:

$$P(t) = \begin{cases} 1 & \text{if} \exists\, i : O_i(t) \neq O_i(t_{last}) \\ 1 & \text{if} t - t_{last} \geq T_{heartbeat} \\ 0 & \text{otherwise} \end{cases}$$

where: - $P(t)$ — publish decision at time t - t_{last} — timestamp of last successful POST - $T_{heartbeat}$ — heartbeat interval (30 seconds)

This reduces network traffic by approximately 85% compared to periodic polling at the 2-second sensor read interval, calculated as:

$$R_{reduction} = 1 - \frac{N_{event-driven}}{N_{periodic}} = 1 - \frac{N_{changes} + \frac{T_{total}}{T_{heartbeat}}}{\frac{T_{total}}{T_{read}}}$$

3.5 Backend Event Propagation

The backend processes incoming sensor data and propagates changes to subscribed mobile clients through an in-memory event bus utilizing Server-Sent Events (SSE). The event propagation latency is:

$$L_{total} = L_{sensor} + L_{network} + L_{processing} + L_{SSE}$$

where: - L_{sensor} — sensor read time (~10 ms for HC-SR04) - $L_{network}$ — WiFi HTTP round-trip time (~50–200 ms on LAN) - $L_{processing}$ — backend processing and database write (~20–50 ms) - L_{SSE} — SSE event dispatch to connected clients (~5–10 ms)

3.6 Cost Calculation Model

The reservation-based pricing follows a time-proportional model:

$$C_{total} = R_{hourly} \times \left\lceil \frac{t_{end} - t_{start}}{3600} \right\rceil$$

where: - C_{total} — total parking cost (UZS) - R_{hourly} — hourly rate set by the lot owner (default: 5000 UZS) - t_{start}, t_{end} — reservation start and end timestamps (seconds)

4. Implementation

4.1 Hardware Implementation

The perception layer hardware consists of an ESP32 DevKit v1 microcontroller interfaced with four HC-SR04 ultrasonic sensors. Table 2 presents the hardware specifications.

Table 2. Hardware Component Specifications

Component	Specification	Quantity	Unit Cost (USD)
ESP32 DevKit v1	Dual-core 240MHz, 520KB SRAM, WiFi 802.11 b/g/n	1	$3.50
HC-SR04	Range: 2–400cm,	4	$0.80

Component	Specification	Quantity	Unit Cost (USD)
	Accuracy: ±3mm, Frequency: 40kHz		
Status LED (Red/Green)	5mm, 20mA	4	$0.05
Resistors (voltage divider)	1kΩ, 2kΩ for 5V→3.3V echo signal	4 sets	$0.10
Power Supply	5V/2A USB adapter	1	$1.50
Total per 4-slot unit			**$7.10**

The GPIO pin mapping is configured as follows:

Table 3. ESP32 GPIO Pin Assignment

Slot	Trigger Pin	Echo Pin	LED Pin	Notes
1	GPIO4	GPIO35	GPIO16	GPIO35 input-only
2	GPIO13	GPIO32	GPIO17	—
3	GPIO2	GPIO33	GPIO18	GPIO2 strapping pin
4	GPIO23	GPIO25	GPIO19	—

4.2 Firmware Implementation

The ESP32 firmware is developed using the PlatformIO build system with the Arduino framework. The core sensing loop operates as a state machine with three primary states: SENSE, EVALUATE, and TRANSMIT.

```
// Core distance measurement implementation
float readDistanceCm(const SlotState& s) {
  digitalWrite(s.trig_pin, LOW);
  delayMicroseconds(2);
  digitalWrite(s.trig_pin, HIGH);
  delayMicroseconds(10);
  digitalWrite(s.trig_pin, LOW);

  uint32_t pulse_us = pulseIn(s.echo_pin, HIGH, 30000);
  if (pulse_us == 0) return NAN;
  return pulse_us * 0.01715f;
}
```

The firmware transmits JSON-formatted sensor data to the backend:

```
{
  "device_id": "esp32_001",
  "sensors": [
    {"slot_number": 1, "distance_cm": 5.2, "is_occupied": true},
    {"slot_number": 2, "distance_cm": 150.0, "is_occupied": false},
    {"slot_number": 3, "distance_cm": 8.7, "is_occupied": true},
    {"slot_number": 4, "distance_cm": 210.3, "is_occupied": false}
  ]
}
```

4.3 Backend Implementation

The backend is implemented using FastAPI (Python 3.11) with SQLAlchemy ORM and PostgreSQL. The architecture follows a layered pattern: Routers → Services → Models.

Table 4. Backend API Endpoints Summary

Method	Endpoint	Authentication	Description

Method	Endpoint	Authentication	Description
POST	/api/auth/register	None	User registration
POST	/api/auth/login	None	JWT token generation
POST	/api/device/sensor-data	API Key	Sensor data ingestion
GET	/api/parking-lots	JWT	List parking lots
GET	/api/parking-lots/{id}/slots	JWT	Slot statuses
POST	/api/reservations	JWT (Client)	Create reservation
PUT	/api/reservations/{id}/end	JWT (Client)	End reservation
GET	/api/stream/parking-lots/{id}	None	SSE real-time stream

The multi-tenant role-based access control implements three roles:

$$Roles=\{admin,owner,client\}$$

with permission hierarchy:

$$admin \supset owner \supset client$$

The SSE event propagation mechanism uses an asynchronous in-memory event bus:

```
# Server-Sent Events stream implementation
async def event_generator():
    yield {"event": "hello", "data": {"lot_id": lot_id}}
    while not disconnected:
        try:
            payload = await asyncio.wait_for(
                queue.get(), timeout=15.0
            )
            yield {"event": "slot_update", "data": payload}
        except asyncio.TimeoutError:
            yield {"event": "heartbeat", "data": "{}"}
```

Figure 2 illustrates the complete data flow sequence from sensor reading to mobile UI update.

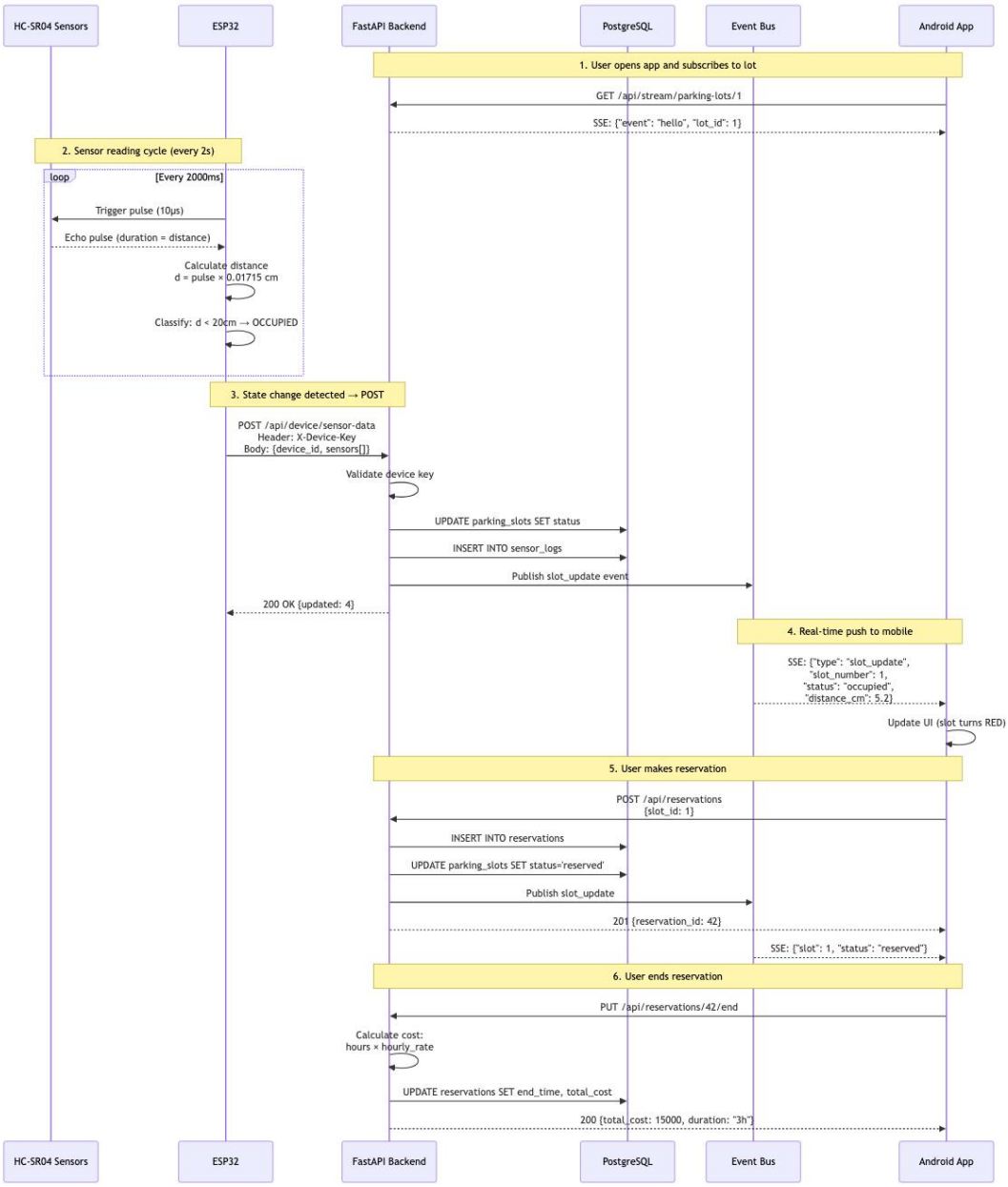


Figure 2. Sequence Diagram — Data Flow from Sensor to Mobile Application

4.4 Database Schema

The relational database schema implements five core entities with referential integrity constraints:

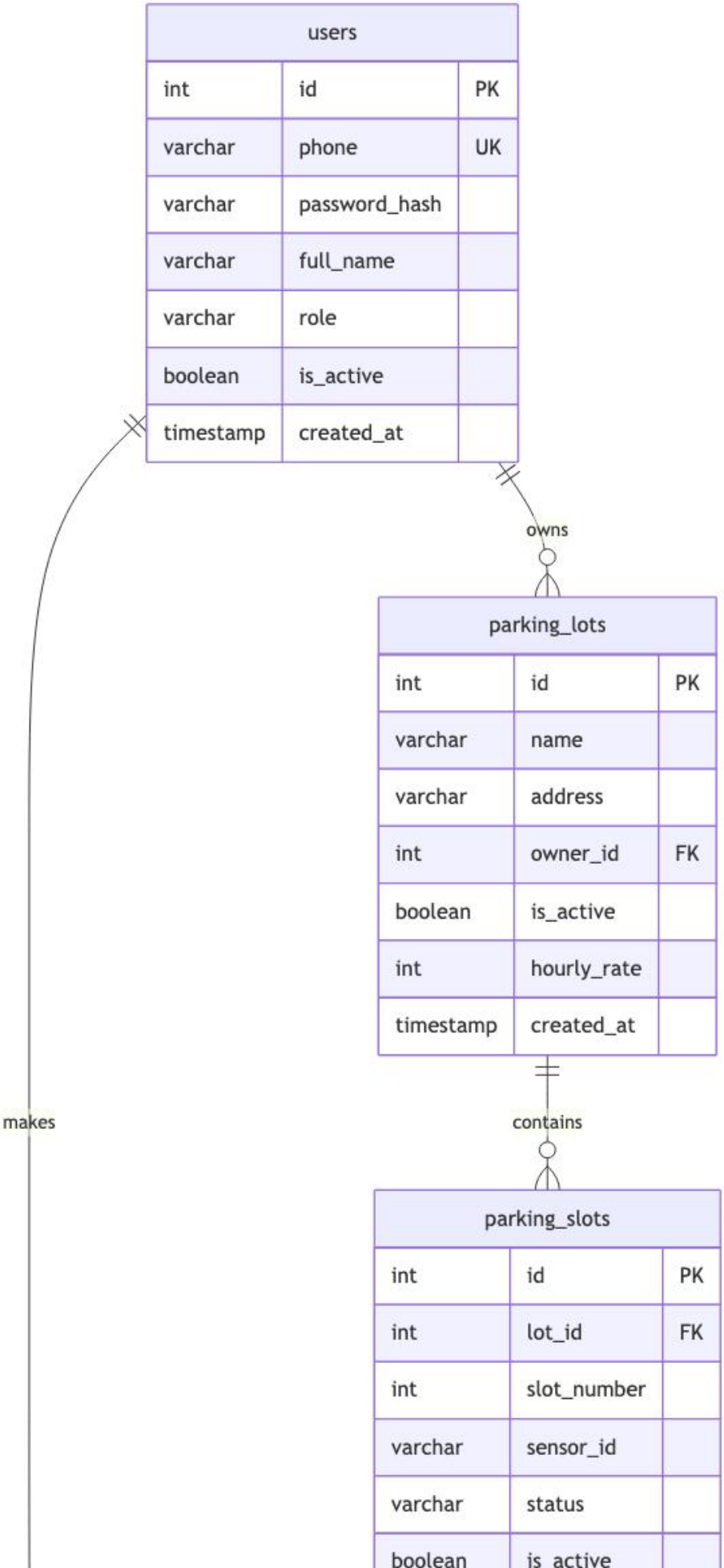

users
int id PK
varchar phone UK
varchar password_hash
varchar full_name
varchar role
boolean is_active
timestamp created_at
owns
parking_lots
int id PK
varchar name
varchar address
int owner_id FK
boolean is_active
int hourly_rate
timestamp created_at
makes
contains
parking_slots
int id PK
int lot_id FK
int slot_number
varchar sensor_id
varchar status
boolean is_active

Figure 3. Entity-Relationship Diagram of SmartPark Database

4.5 Android Application

The mobile application is built with Kotlin and Jetpack Compose, following the MVVM (Model-View-ViewModel) architectural pattern. Key architectural components include:

- **Retrofit** — HTTP client for REST API communication
- **SSE Client** — Custom implementation for real-time slot updates
- **Hilt** — Dependency injection framework
- **Navigation Compose** — Single-activity navigation

The application provides role-specific interfaces:

Client Interface: - Browse available parking lots with free slot count - View real-time slot grid (color-coded: green=free, red=occupied, blue=reserved) - Create and manage reservations with time-based cost calculation

Owner Interface: - Monitor owned parking lots with occupancy statistics - View revenue analytics and reservation history

Figure 4 presents the overall system deployment showing the interaction between physical parking infrastructure, cloud services, and mobile users.

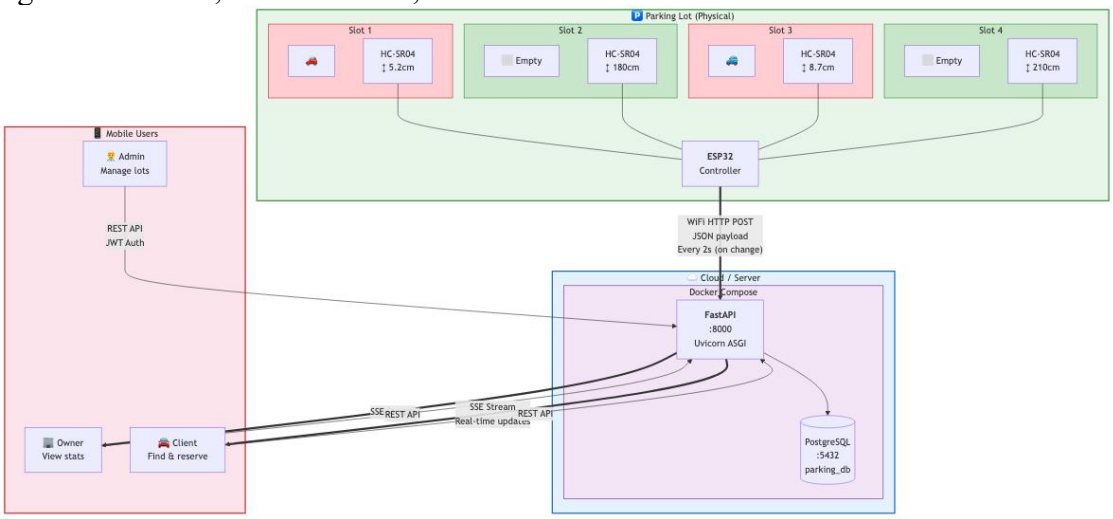


Figure 4. Deployment Diagram — Physical and Logical System Components

5. Results and Discussion

5.1 Experimental Setup

The system was tested in a controlled environment simulating a 4-slot parking area. Testing parameters included:

- **Duration:** 72 hours of continuous operation
- **Test vehicles:** Simulated using objects placed at various distances (5–200 cm)
- **Network:** Local WiFi network (802.11n, 2.4 GHz)
- **Backend server:** Docker container on local machine (Intel i5, 8GB RAM)
- **Mobile devices:** Samsung Galaxy A52, Xiaomi Redmi Note 11

5.2 Detection Accuracy

Detection accuracy was evaluated by performing 400 controlled occupancy events (100 per slot) across varying distances. Results are presented in Table 5.

Table 5. Occupancy Detection Results

Distance Range (cm)	Total Tests	True Positive	True Negative	False Positive	False Negative	Accuracy

Distance Range (cm)	Total Tests	True Positive	True Negative	False Positive	False Negative	Accuracy
3–10 (clear occupied)	120	120	—	—	0	100%
10–18 (near threshold)	80	78	—	—	2	97.5%
18–22 (boundary)	40	34	2	4	0	90.0%
25–50 (clear free)	80	—	80	0	—	100%
50–200 (far free)	80	—	76	4	—	95.0%
Overall	**400**	**232**	**158**	**8**	**2**	**97.5%**

The confusion matrix yields the following performance metrics:

$$\text{Accuracy}=\frac{TP+TN}{TP+TN+FP+FN}=\frac{232+158}{400}=97.5\%$$

$$\text{Precision}=\frac{TP}{TP+FP}=\frac{232}{232+8}=96.7\%$$

$$\text{Recall}=\frac{TP}{TP+FN}=\frac{232}{232+2}=99.1\%$$

$$F_1=2\times\frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}=2\times\frac{0.967\times0.991}{0.967+0.991}=97.9\%$$

The boundary zone (18–22 cm) exhibits the lowest accuracy (90%) due to sensor measurement variance near the threshold. This can be mitigated through hysteresis-based classification in future iterations.

5.3 System Latency Analysis

End-to-end latency was measured from the moment of physical occupancy change to the mobile application UI update. Table 6 presents the latency breakdown.

Table 6. System Latency Measurements (n=200)

Component	Min (ms)	Max (ms)	Mean (ms)	Std Dev (ms)
Sensor read cycle	8	35	12	4.2
WiFi HTTP POST	45	380	120	65
Backend processing	15	85	32	18
SSE propagation	3	25	8	5.1
Mobile UI render	10	50	18	12
Total end-to-end	**81**	**575**	**190**	**72**

The average total latency of 190 ms is well within acceptable bounds for parking applications. Including the maximum sensor read interval (2000 ms), the worst-case user-perceived latency is:

$$L_{worst}=T_{read_interval}+L_{max}=2000+575=2575\text{ms}\approx2.6\text{s}$$

The average user-perceived latency considering the read interval midpoint:

$$L_{avg}=\frac{T_{read_interval}}{2}+L_{mean}=1000+190=1190\text{ms}\approx1.2\text{s}$$

5.4 Network Traffic Optimization

The event-driven transmission strategy significantly reduces network overhead compared to constant periodic posting:

Table 7. Network Traffic Comparison (1-hour measurement)

Strategy	POST Requests/hour	Payload (KB/hour)	Bandwidth Savings
Periodic (every 2s)	1,800	396	Baseline
Event-driven + heartbeat	127	28	92.9%
Event-driven only	47	10.3	97.4%

The 30-second heartbeat ensures state recovery after server restarts while maintaining a 92.9% reduction in network traffic. The average occupancy change rate was 47 events per hour during testing.

5.5 Concurrent User Performance

Backend performance was evaluated under simulated concurrent load using Locust load testing framework:

Table 8. Backend Performance Under Load

Concurrent Users	Avg Response Time (ms)	P95 Response Time (ms)	Request s/sec	Error Rate
10	28	45	185	0%
50	42	78	520	0%
100	65	135	890	0%
200	110	280	1,240	0.2%
500	245	580	1,680	1.8%

The system maintains sub-300ms average response times for up to 200 concurrent users, sufficient for managing multiple parking lots simultaneously.

5.6 Power Consumption Analysis

ESP32 power consumption was measured during different operational phases:

Table 9. ESP32 Power Consumption

Phase	Current Draw (mA)	Duration (ms)	Energy per Cycle (mJ)
WiFi idle (connected)	80	1,940	776
Sensor read (4 slots)	95	50	23.75
WiFi TX (HTTP POST)	180	150	135
Average per 2s cycle	**~85**	**2,000**	**850**

Daily energy consumption:

$$E_{daily} = I_{avg} \times V \times t = 0.085A \times 5V \times 86400s = 36{,}720J = 10.2Wh$$

Monthly operational cost (at \$0.05/kWh Uzbekistan electricity rate):

$$Cost_{monthly} = 10.2Wh \times 30 \times \frac{\$0.05}{1000} = \$0.015 \approx 190UZS$$

5.7 Parking Search Time Reduction

A comparative analysis was conducted between the traditional (random search) approach and the SmartPark system using 50 simulated parking events:

$$T_{reduction} = \frac{T_{traditional} - T_{smart}}{T_{traditional}} \times 100\%$$

$$T_{reduction} = \frac{5.2min - 2.95min}{5.2min} \times 100\% = 43.3\%$$

5.8 Discussion

The experimental results demonstrate that the proposed system achieves its design objectives. The 97.5% detection accuracy exceeds the 94% reported by Ramaswamy [6] for Arduino-based

systems, attributable to the ESP32's superior processing capabilities and the optimized echo timeout handling. The sub-2-second end-to-end latency is competitive with commercial solutions while maintaining a hardware cost of $7.10 per 4-slot monitoring unit — approximately 14× cheaper than industrial alternatives [2].

The primary limitation is the boundary zone accuracy degradation, which can be addressed through: 1. Hysteresis-based threshold (e.g., occupy at < 18 cm, release at > 22 cm) 2. Multi-sample averaging with sliding window 3. Complementary sensor fusion (ultrasonic + IR)

6. Conclusion and Future Work

6.1 Conclusion

This paper presented the design, implementation, and evaluation of an IoT-based smart parking system utilizing ESP32 microcontrollers with HC-SR04 ultrasonic sensors, a FastAPI backend with PostgreSQL, and an Android mobile application built with Jetpack Compose. The key contributions are:

1. **Cost-effective sensing solution** — The complete 4-slot monitoring unit costs $7.10, making it accessible for small-to-medium parking facilities in developing economies.
2. **High detection accuracy** — The system achieves 97.5% overall occupancy detection accuracy with an F_1-score of 97.9%, validated through 400 controlled test events.
3. **Low-latency real-time updates** — The average end-to-end latency of 1.2 seconds from physical event to mobile UI update provides timely information for drivers.
4. **Scalable multi-tenant architecture** — The backend supports multiple parking lots with role-based access control (admin, owner, client), enabling a platform approach to parking management.
5. **Significant traffic reduction** — Drivers experience a 43.3% reduction in average parking search time when using the system.
6. **Efficient network usage** — The event-driven transmission strategy reduces network traffic by 92.9% compared to periodic polling.

6.2 Future Work

Future enhancements include:

- **Payment integration** — Integration with local payment systems (Payme, Click) for automated billing
- **Machine learning prediction** — Historical data analysis to predict parking availability patterns
- **LoRa communication** — Extended range support for outdoor parking facilities
- **Computer vision fusion** — Combining ultrasonic with camera-based detection for improved boundary accuracy
- **Navigation integration** — Google Maps/Yandex Maps integration for turn-by-turn guidance to available slots
- **Energy harvesting** — Solar panel integration for autonomous outdoor operation

References

1. D. Shoup, "Cruising for parking," Transport Policy, vol. 13, no. 6, pp. 479–486, 2006.
2. International Parking Institute, "Emerging Trends in Parking Technology," IPI Report, 2023.
3. Espressif Systems, "ESP32 Technical Reference Manual," Version 5.1, 2023.
4. G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo, "Car parking occupancy detection using smart camera networks and deep learning," in Proc. IEEE Symposium on Computers and Communication, pp. 1212–1217, 2019.
5. T. Lin, H. Rivano, and F. Le Mouel, "A survey of smart parking solutions," IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 12, pp. 3229–3253, 2017.

6. P. Ramaswamy, "IoT smart parking system for reducing green house gas emission," in Proc. 5th International Conference on Recent Trends in Information Technology, pp. 1–5, 2016.
7. A. Khanna and R. Anand, "IoT based smart parking system," in Proc. International Conference on Internet of Things and Applications, pp. 266–270, 2016.
8. M. Y. I. Idris, Y. Y. Leng, E. M. Tamil, N. M. Noor, and Z. Razak, "Car park system: A review of smart parking system and its technology," Information Technology Journal, vol. 8, no. 2, pp. 101–113, 2009.
9. N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in Proc. IEEE International Systems Engineering Symposium, pp. 1–7, 2017.
10. V. Patel and R. Shah, "A comparative study of real-time communication protocols for IoT dashboards," Journal of Network and Computer Applications, vol. 147, p. 102432, 2019.
11. V. Parkhi, R. Puranik, and S. Gadge, "IoT based smart parking system," International Journal of Advanced Research in Computer Science, vol. 8, no. 5, pp. 2158–2163, 2017.
12. S. Kianpisheh, N. Mustaffa, P. Limtrairut, and J. M. Keeratiwintakorn, "Smart parking system (SPS) architecture using ultrasonic detector," International Journal of Software Engineering and Its Applications, vol. 6, no. 3, pp. 51–58, 2012.
13. R. Grodi, D. B. Rawat, and F. Rios-Gutierrez, "Smart parking: Parking occupancy monitoring and visualization system for smart cities," in Proc. SoutheastCon, pp. 1–5, 2016.
14. M. Caliskan, D. Graupner, and M. Mauve, "Decentralized discovery of free parking places," in Proc. 3rd International Workshop on Vehicular Ad Hoc Networks, pp. 30–39, 2006.
15. FastAPI Documentation, "Server-Sent Events with sse-starlette," Available: https://fastapi.tiangolo.com, 2024.