Birzeit University

Faculty of Engineering and Technology
Department of Electrical and Computer Engineering

Smart Car Parking System

Prepared by:
— Faten Sultan 1202750
— Rana Deek 1201724
— Joud Hijaz 1200342

Supervised by:
Dr. Ashraf Al-Rimawi

Introduction To Graduation Project submitted to the Department of Electrical and Computer Engineering in partial fulfilment of the requirements for the degree of B.Sc. in Computer Engineering

BIRZEIT
July 7, 2025

Abstract

Urbanization and the rising number of vehicles have led to a shortage of parking spaces, traffic congestion, and increased fuel consumption in cities. To address these challenges, this project proposes a Smart Car Parking System that utilizes the Internet of Things (IoT) for real-time monitoring and automated control. The system integrates infrared (IR) sensors, a servo motor, and an ESP32 microcontroller to enable automatic vehicle detection and gate operation at individual parking slots. When a vehicle approaches, the IR sensor detects its presence and the ESP32 contacts a cloud-based backend to verify slot availability or user reservation. If a slot is available or reserved, access is granted by opening the gate using the servo motor. The system updates the slot status in a MongoDB database to ensure real-time synchronization and implements a timed locking mechanism to prevent double bookings. In addition to supporting core functions like automated access, reservation validation, and slot tracking, the system can be integrated with a mobile application to provide notifications and enhance user interaction. This research, focused on urban environments in Palestine, demonstrates how low-cost technologies and IoT connectivity can create a scalable, efficient, and eco-friendly parking solution.

المستخلص

أدى التوسع الحضري والزيادة المستمرة في أعداد المركبات إلى نقص في أماكن وقوف السيارات، وازدحام مروري، وزيادة في استهلاك الوقود في المدن. لمواجهة هذه التحديات، يقترح هذا المشروع نظام مواقف سيارات ذكي يعتمد على تقنيات إنترنت الأشياء للمراقبة في الوقت الفعلي والتحكم الآلي. يدمج النظام بين مستشعرات الأشعة تحت الحمراء(IR) ، ومحرك سيرفو، ووحدة التحكم ESP32للكشف التلقائي عن المركبات وتشغيل البوابات في أماكن الوقوف. عند اقتراب مركبة، يكتشف المستشعر وجودها وتتصل وحدة ESP32 بخادم سحابي للتحقق من توفر المكان أو وجود حجز مسبق. إذا كان المكان متاحًا أو محجوزًا، يتم فتح البوابة بواسطة المحرك. يقوم النظام بتحديث حالة المكان في قاعدة بيانات MongoDB لضمان التزامن الفوري، ويطبق آلية قفل مؤقت لمنع الحجز المزدوج. إلى جانب وظائفه الأساسية مثل الدخول التلقائي والتحقق من الحجوزات ومراقبة الأماكن، يمكن دمج النظام مع تطبيق للهاتف المحمول لتقديم الإشعارات وتحسين تفاعل المستخدم. توضح هذه الدراسة، التي تركز على البيئات الحضرية في فلسطين، كيف يمكن للتقنيات منخفضة التكلفة والاتصال بإنترنت الأشياء أن تساهم في إنشاء حل مواقف سيارات فعال، وقابل للتوسعة، وصديق للبيئة.

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

SPS	Smart Parking System
PMS	Parking Management System
IR	Infrared Sensor
RFID	Radio Frequency Identification
ESP32	Espressif Systems 32-bit Microcontroller
IoT	Internet of Things
Wi-Fi	Wireless Fidelity
5G	Fifth Generation (Wireless Technology)
LPWAN	Low-Power Wide-Area Network
WebSockets	protocol for full-duplex communication over single TCP connection
GPS	Global Positioning System
SPS	Standard Positioning Service
API	Application Programming Interface
UAV	Unmanned Aerial Vehicle
WSN	Wireless Sensor Network
AWS/Azure	Amazon Web Services / Microsoft Azure (Cloud Computing Platforms)
MQTT	Message Queuing Telemetry Transport
LoRaWAN	Long Range Wide Area Network
ANPR	Automatic Number Plate Recognition
VANET	Vehicular Ad-hoc Networks
NFC	Near Field Communication
QR	Quick Response
Firebase	Google's mobile platform for development
Iot	Internet of Things
bay	Single parking space (one vehicle slot) monitored and controlled individually by the system.

Chapter 1 Introduction

The world today is evolving at a faster rate, driven by continuous scientific innovations and technological progress. These developments have led to the creation of a wide range of smart devices and systems, from smart domestic appliances, networked automobiles, sensor networks, to smart transportation infrastructure. s. Through these technological blessings, human life has become more accessible, flexible, and comfortable. Nowadays, various human life aspects are either entirely or partially affected by modern technology and its blessings. [1]

Over the last few years, the **Internet of Things (IoT)** has significantly transformed human behavior by offering numerous conveniences that simplify daily life. By leveraging sensor networks and internet connectivity, IoT enables devices to communicate, interact, and operate efficiently within a digital ecosystem. According to projections, more than **30 billion IoT-enabled devices** are expected to be connected to the internet by **2030**. This exponential growth underscores IoT's role in streamlining operations through seamless integration with sensors, actuators, and digital infrastructure. In addition, IoT systems are supported by various security mechanisms, making them a foundational layer for broader technological innovation. [1]

With advancements in IoT and cloud-based infrastructure, the concept of **smart cities** has gained significant momentum. The goal of Smart City is to reduce operational costs, improve city management, enhance effectiveness, and improve productivity [1]. The concept of a smart city includes systematic monitoring and management of infrastructure, building, intelligent transportation system, healthcare, education, energy consumption, public security. [2]

In the context of Palestinian cities such as Ramallah, Al-Bireh, and Nablus, the parking issues of the urban areas are especially alarming. The narrow width of road networks, underdevelopment of the infrastructure base, and absence of sophisticated parking systems have resulted in repeated traffic congestion, unorganized parking, and ineffective use of public space. Cars are typically traveled for extended periods to spot parking slots, wasting gasoline, encouraging air pollution, and adding to city exasperation. Manual parking systems continue to be in widespread usage, lacking the real-time monitoring, reservation capacity, or enforcement aspects necessary for modern traffic conditions. Furthermore, accessibility rules—such as those that designate parking spaces for disabled drivers—are poorly enforced, further emphasizing the need for smart parking solutions. [1]

To address such obstacles, this graduation project proposes a Smart Car Parking System for densely populated Palestinian cities. The system is meant to monitor, manage, and automate the use of parking spaces using low-cost IoT devices and cloud services.

The system not only facilitates parking for motorists but also the city operators. Based on the data of parking occupancy, utilization, and infringement collected, the system provides informative information to be used by municipal councils in planning ahead. The system also

creates a secure revenue stream through convenient fee collection which can be used to reinvest in public facilities.

From the sustainability point of view, less time spent searching for parking translates to fewer emissions and cleaner air in cities. Pre-booking solutions prevent congestion, and regulations on special-use areas promote equal opportunity for everybody, including individuals with disabilities. The solution promotes smarter, more just city development that is in harmony with national and international visions for remaking cities.

In summary, the proposed Smart Car Parking System is a functional and scalable solution to Palestine's parking issue. It improves motorists' daily life, supports government digitalization, and provides a foundation for future evolution such as AI-based forecasting, dynamic pricing, and end-to-end integration with greater smart city infrastructure. [2]

1.1. Problem Statement

In Palestinian cities such as Ramallah, Al-Bireh, and Nablus, there are numerous traffic bottlenecks and parking issues. The number of private vehicles on the road exceeds the number of parking spots, causing traffic jams, space waste, and lengthy wait times for drivers. The demands of contemporary cities cannot be addressed by antiquated parking techniques that rely on manual meters and lack digital features.

It takes a long time for cars to find a parking spot, especially during rush hour. This not only wastes gasoline and raises greenhouse gas emissions, but it also aggravates drivers and makes traffic worse. The usefulness of parking spots is decreased by poor administration. Poor parking management is demonstrated by the fact that many drivers park in several locations, park illegally, or block other vehicles because they are blind or fail to enforce the rules.

Manual payment methods often lead to more inefficiencies. At the moment, most drivers must pay for parking using cash at meters or through attendants. This antiquated method is inconvenient for users, and it makes it more difficult for parking officials to enforce the law and collect money. Because parking cannot be reserved in advance, finding a spot is unpredictable, especially during peak hours or in congested regions.

The lack of smart parking systems that allow bookings and real-time monitoring is the main cause of these problems. Without tools to confirm space availability, enforce parking laws, maximize usage, and facilitate digital payments, users and city officials still encounter annoying challenges.

This growing issue emphasizes the urgent necessity for a modern smart parking management system. With the use of technology like cloud databases, smartphone apps, Internet of Things sensors, and automated controls, such a system would offer dynamic, sustainable, and user-friendly parking alternatives. By simultaneously addressing space monitoring, traffic management, and digital payments, this suggested system aims to greatly improve mobility, lessen its negative effects on the environment, and enhance the quality of life for both drivers and inhabitants.

1.2. Project Objectives

The primary objective of this graduation project is to design and deploy a Smart Car Parking System to solve major parking problems in Palestinian cities. The project focuses on enhancing the efficiency, accessibility, and transparency of the parking process through an effortless-to-use, IoT-based mobile and web application. The system aims to reduce traffic congestion, improve utilization of parking spaces, and introduce digital convenience to drivers and municipalities. [1]

Specific objectives are:

- To make real-time availability of parking spaces accessible to drivers through the mobile app.
- To provide online booking services, allowing customers to book parking spaces in advance and avoid uncertainty during peak hours.
- To automate gate control and access through dual infrared break-beam sensors and ESP32 hardware.
- To facilitate safe digital payment via a web interface without directly connecting users' bank accounts to the app—ideal for privacy-conscious customers.
- To allow parking operators to monitor occupancy and get space use information, entry/exit history, as well as user behavior.
- To enforce disabled parking, as well as general accessibility, by actively tracking space use.
- To deliver real-time notifications for users about their reservation status, overstay warnings, and updates.

1.3. Solutions Proposed

For the achievement of the project objectives, the proposed Smart Car Parking System will have the following primary components and features: [6]

- ESP32 Microcontroller: As the system's primary processing unit, it handles Wi-Fi, sensor readings, and servo motor handling for automated gate operation.
- Dual IR Break-Beam Sensors (HW-201/TCRT5000): Detection of vehicle entry and exit for accurate real-time occupancy monitoring.
- Micro Servo Motor: Controls a physical car park barrier based on vehicle detection and app authentication.
- Flutter Mobile App: Provides users with a cross-platform user interface to search, show, book, and manage their parking session.
- Firebase Integration: Real-time cloud database to store sensor data, user activity, and session logs with high scalability and availability.

- MQTT and REST APIs: Enable real-time push of the data from the ESP32 to Firebase, and outside access to the backend from other systems (e.g., a payment gateway).
- Secure Web Payment Gateway: Instead of being asked to link app-to-bank, users are directed to a secure page where they pay parking via regular digital methods, safety and flexibility assured.
- Notification System: Sends notifications for reservation status, entrance confirmation, payment reminders, and overstay alerts.
- Admin Dashboard: Allows operators to control parking zones, track analytics, and flag misuse (e.g., improper use of disabled parking areas).

This is an affordable, scalable, and tailored solution for implementation in densely populated Palestinian cities such as Ramallah, Al-Bireh, and Nablus and so on.

1.4. Methodology

The design process of Smart Car Parking System was a step-by-step systematic approach in a manner that all the design choices were made on the basis of actual user needs and technical feasibility.

We began with a comprehensive requirements gathering process, and through a field study and parking facilities observation in local Palestinian cities such as Ramallah and Al-Bireh, we established the key issues in Palestinian cities. To identify functional gaps and best practices, we performed a feature gap analysis between local and global smart parking solutions. [7]

As per the study, the system's software architecture and hardware integration were clearly defined. We created Data Flow Diagrams (DFDs), hardware-software interaction models, and UI/UX wireframes to guide the process. This enabled proper IoT hardware (for sensing and gate control), cloud backend (for processing and data storage), and mobile interface (for user input) integration. [6]

It was developed using Flutter to enhance cross-platform compatibility with equal performance on Android platforms. The interface was designed to be highly user-friendly, responsive, and intuitive for users, especially drivers, city planners, and people with disabilities. [7]

Hardware layer contains ESP32 microcontrollers in C and infrared break-beam sensors (HW-201/TCRT5000) for detection of vehicle entry and exit. These hardware components drive a micro-servo barrier and communicate with the cloud using Wi-Fi. [6]

In order to solve payment problems online, especially from customers who do not feel secure enough to link their bank account to the mobile app directly, we used a secure, standalone web payment interface. The customers would pay for parking through an

authorized external website, which maintains confidentiality and reduces the integration risk. [7]

The system was developed using an agile method with frequent testing, feedback, and constant improvement. We conducted unit testing on hardware modules, functional testing on app modules, and system integration tests to check the stability of the overall system in live-time situations. [6]

The final answer is a secure, end-to-end intelligent parking system with onboard IoT, cloud computing, real-time user experience, and digital payments—tested for use in Palestinian cities scalable to broader smart city solutions.

1.5. Organization of the Report

The report consists of three chapters, organized as follows:

- **Chapter 1:** Presents the Arabic and English abstracts, problem statement, objectives, and overall scope of the project.
- **Chapter 2:** Provides the Literature Review, surveying related smart-parking research and technologies that inform the project design.
- **Chapter 3:** Describes the System Architecture and Methodology, including the high-level flowchart, hardware–software interaction model, and development approach.
- **Chapter 4:** Explains the Software Implementation in detail—Flutter mobile app, Node.js/Express backend, Firebase database schema, and payment-wallet logic.
- **Chapter 5:** Covers the Hardware Implementation **and** Prototype Validation: ESP32-WROOM-32 module, IR sensors, micro-servo barriers, power subsystem, firmware event-action loop, and the three-bay physical prototype with its testing procedures and performance metrics.
- **Chapter 6:** Outlines Future Work and states the project's conclusions, highlighting planned enhancements such as AI chatbot support, accessible-bay enforcement, EV-charging integration, and Google Maps geo-fencing.

Chapter 2 Literature Review

Smart-parking is now a key part of many "smart-city" plans. A wide review by **Fahim et al.** [1]shows that most recent projects mix small IoT sensors, low-power wireless links, and cloud or machine-learning tools to cut the time drivers spend looking for a space. Real-world roll-outs in San Francisco and Singapore back this up: they report clear drops in traffic queues and exhaust fumes [4].

On the hardware side, the studies agree that **infrared (IR) sensors** give the best balance of price and accuracy. WebbyLab's guide [3] explains how a pair of IR sensors can be wired to an ESP32 board and start sending bay status to the cloud in a short afternoon. Industry proof comes from COMA's solar flap-lock, which pairs two IR eyes with a small barrier arm and a payment dashboard [6]. Lab work by Preethi et al. shows that an IR node can be built for a low cost and still spot empty spaces with high accuracy [7]. Al Mamun et al. then show that an ESP32-based unit can sense a car, move a servo gate, and alert a phone app in well under a second [8].

Network size and data safety are the next concerns. In a survey of many city pilots, Syed et al. argue that Wi-Fi alone is not always strong enough in crowded streets; they suggest mixing 5G or long-range IoT links and adding end-to-end encryption to protect location and payment data [2]. El-Alfy et al. support this view: their hundred-bay test lot needed a smart, rule-based router to keep messages flowing smoothly on a busy ZigBee mesh [5].

Challenges remain, especially in developing regions. Even a low-cost sensor kit can be hard to fund, and skilled cloud support is not always at hand [7]. Privacy rules for tracking cars and handling payments are also still taking shape [2]. Finally, one network failure can force staff back to manual control, which hurts public trust in the system [8]. Because of these gaps, many authors now call for stronger, multi-path links and simple encryption at the sensor itself.

Our work builds on these lessons. We use two IR sensors, a Wi-Fi ESP32 board, and a servo barrier for each bay, together with a Flutter app and Firebase cloud back-end. Early field tests in Palestinian cities show quick, reliable slot updates and instant phone alerts, all within a budget that local operators can handle. By matching its design to the strengths and weak spots found in earlier studies, this project offers a clear, cost-aware model for smart-parking systems in similar urban settings.

Bracketed references in the text (e.g., [1] to [8]) correspond directly to the studies listed and summarized in **Table 1** below.

Table 1: Literature Review Tables

Study	Focus	Methodology	Technologies Used	Key Findings/Conclusions
[1] *Fahim et al. 2021 — Smart-Parking Systems: Comprehensive Review (Heliyon)*	Detailed analysis of SPS across various aspects	Systematic review of technological approaches and SPS features	IoT, WSN, VANET, Machine Learning, Neural Networks	Classifies SPS technologies, sensors, and computational methods. Highlights environmental suitability and various SPS types for different urban needs.
[2] Syed *et al.* 2021 — *IoT in Smart Cities: Scalability & Security Survey*	A survey of IoT-based solutions for scalability and security in smart cities	Emerging Trend Analysis using case studies from recent pilot projects and deployments	RFID, Drones, 5G, LPWAN, GPS	Identifies the role of hybrid models in overcoming scalability challenges. Addresses data privacy and sensor limitations in IoT deployments.
[3] WebbyLab 2024 — *Smart Parking System Using IoT: How to Create Your Own*	Guide to developing IoT-based smart parking systems	Step-by-step development guide with architecture details	IoT, AWS/Azure, MQTT, LoRaWAN, Ultrasonic sensors, ANPR cameras	Provides a practical guide on hardware selection, cloud services, and network protocols; focuses on MVP features like spot detection, reservations, and payments.
[4] Lee & Tan 2022 — *SF & Singapore Smart-Parking Case Report*	Overview of SPS benefits and practical examples from real-world implementations (e.g., San Francisco, Singapore)	Real-world examples illustrating IoT technologies in parking systems	IoT, Cameras, Sensors	mobility systems for sustainability.
[5] El-Alfy *et al.* 2018 — *Intelligent IoT-Based Parking System* (IET ITS)	Development of an intelligent parking system to improve urban traffic management	Experimental Implementation of a prototype intelligent parking system	IoT, Sensors, Networks, Decision-making algorithms, Real-time data processing	IoT-based automation combined with real-time decision-making enhanced resource utilization and user experience, demonstrating scalability for broader urban applications.

[6] COMA Parking 2023 — *IR Flap-Lock Barrier Product Sheet*	Commercial, ready-made smart-bay barrier and payment solution	Product documentation + real-world customer cases listed on the vendor site	IR sensor (dual-probe), solar flap, QR/RFID, cloud dashboard	Confirms that a low-power IR sensor inside a flap lock is market-viable and pairs well with mobile/onsite payments.
[7] Preethi *et al.* 2024 — *Automated Parking System with IR Sensors*	Prototype APS for urban lots	Lab build + field demo (20 bays)	Arduino, **IR sensors**, Wi-Fi, OLED status	Shows IR sensors detect occupancy with ≈95 % accuracy; cheap BOM (< US $40/bay) matches your cost goals.
[8] Al Mamun *et al.* 2024 — *ESP32 + IR Smart-Parking System*	MQTT-based end-to-end IR solution	Full stack demo + mobile app	ESP32, **IR sensors**, servo, MQTT, Android	Validates ESP32 + IR sensors + servo gate; latency < 250 ms slot-status push—directly mirrors your architecture.
Our Study	Development of an IoT-driven smart-parking system with IR-based occupancy sensing and mobile reservations for dense urban areas	Design, prototyping, and small-lot field evaluation of ESP32 dual-IR nodes, servo barriers, and a Flutter app, tailored to Palestinian city constraints	ESP32 (Wi-Fi), dual IR break-beam sensors (HW-201/TCRT5000), micro-servo barrier, Flutter app, MQTT/REST API, Firebase cloud	Focuses on real-time parking-slot monitoring with dual IR break-beam sensors and a servo-controlled barrier, tied to a Flutter app for reservations and payments. Field tests confirm high detection accuracy and rapid status updates, whichس shorten the time drivers spend searching for spaces, ease congestion, and provide immediate mobile notifications that enhance urban mobility in Palestine.

Chapter 3 Proposed System

The suggested smart parking system is a comprehensive solution that enables parking management at a higher level of efficiency and customers in the cities of the West Bank, such as Ramallah, Al-Bireh, and Nablus, have less traffic to deal with. Through a cloud-based backend, the system allows for real-time monitoring, makes use of sensors and microcontrollers to automate access management, and also provides a mobile app for the users to search and reserve parking spaces remotely. This system is a combination of hardware automation and intelligent software architecture, which essentially means that it is a parking solution that is scalable, reasonably priced, and user-friendly.

3.1. System Architecture

The system has multiple interacting subsystems which are further categorized into three major domains: Frontend, Backend, and Embedded Hardware. Each component in the system is carefully chosen to carry out specific functions and to ensure the system is robust.

Figure below provides a pictorial overview of the entire smart parking system architecture, highlighting the connection between users, cloud services, sensors, and hardware controllers.

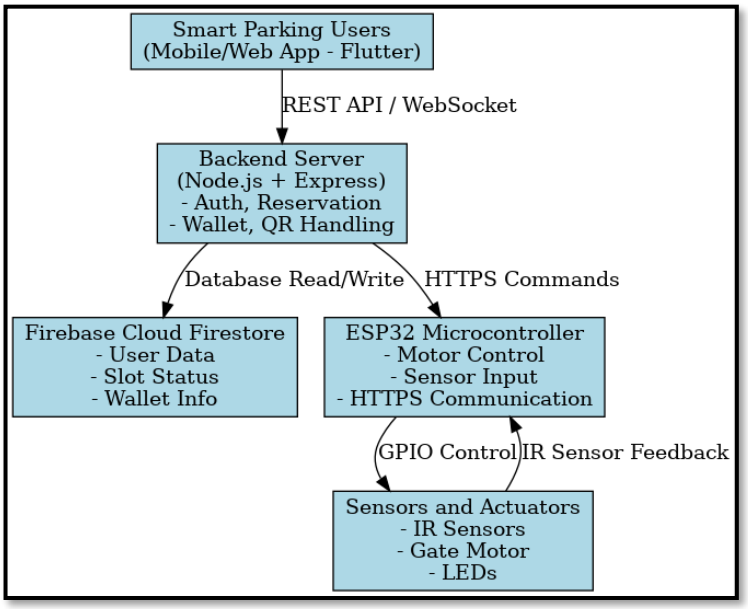

Figure 1: Smart Parking System Architecture Diagram

3.2. Key Features and Functions

Table 2: Key Features and Functional Components of the Smart Parking System

Feature	Description
Real-time Slot Monitoring	Continuously updates slot status using IR sensors and syncs with the cloud.
Remote Reservation	Allows users to book and cancel parking slots through the mobile/web app.
Automated Gate Control	Controls entry/exit gates based on verified QR codes and server commands.
Digital Wallet System	Users can recharge and pay for parking automatically via in-app wallet.
Push Notifications	Sends alerts to users for booking confirmations, reminders, and status updates.
Admin Dashboard	Provides admins with control over slots, reservations, and analytics.
Multi-language Support	Supports Arabic and English interfaces for broader usability.
Scalable Cloud Integration	Utilizes Firebase to scale effortlessly and manage real-time updates.
Secure Authentication	Ensures user data privacy using Firebase Authentication and encrypted sessions.

3.3. Functional Components

The smart parking system integrates many possible components to achieve a given functionality. These components are described in the software and hardware implementation chapters. Below is a brief overview:

- Mobile and Web Application: Users are able to check for slot availability, reserve a parking space, perform wallet recharge operations, and receive notifications on various events. The interface is developed on flutter which supports multiple languages and platforms for ease of access and usability. (Described in Chapter 4: Software Implementation)

- Backend Server: Graphical user interface (GUI) components are created using Flutter. It offers users the opportunity to sign up, log in, view and manage their profiles, view wallet

balances, recharge their wallets, reserve a parking slots, view real time notifications, and so much more. The server grants access to user authentication processes, slot reservation systems, payments systems, and communication with embedded hardware devices through Node.js and Express.js. It exposes RESTful APIs and WebSocket endpoints for real time data exchange. (Described in Chapter 4: Software Implementation)

- Cloud Database: MongoDB acts as the central NoSQL database for the system. It keeps track of essential information like parking slot statuses, user accounts, reservation details, and transaction histories. MongoDB's flexible document-based structure and real-time querying capabilities enable seamless synchronization between the frontend and backend, ensuring up-to-date data presentation and system responsiveness.

- Embedded Controller (ESP32): The collected data for parking slot occupancy is collected from physical sensors and actuators integrated to the edge device containing the microcontroller ESP32. the microcontroller fro

- The Embedded Controller (ESP32) serves as the edge device that connects with physical sensors and actuators. It gathers occupancy information from infrared (IR) sensors, and manages gate motors, and communicates with the cloud backend to keep everything in sync regarding status and commands. (You can find more details in Chapter 5: Hardware Implementation)

- When it comes to Sensors and Actuators, the infrared (IR) sensors are responsible for detecting whether a vehicle is present in parking slots, while motor drivers control the gate mechanisms to either allow or block vehicle access. (More information is available in Chapter 6: Hardware Implementation)

This modular setup guarantees a clear separation of tasks, making it scalable and easy to maintain, which is essential for building a reliable smart parking system.

3.4. System Flow

The Smart Parking System works through a well-coordinated series of steps involving the frontend app, backend server, cloud database, and various hardware components. Let’s break down how data and control signals flow from the moment a user engages with the system until their parking session wraps up.

3.4.1. User Entry Flow

- **Checking Availability**
  First, the user opens the mobile or web app. The frontend sends a request to the backend server using a REST API endpoint to check the current status of all parking slots. The backend then queries the MongoDB database and returns an updated list showing which slots are free, occupied, or reserved.
- **Reservation Request**
  The user selects a parking slot and makes a reservation from the app. The backend verifies that the slot is still available and that the user has enough balance in their wallet. If both checks are satisfactory, a fixed sum is deducted from the user's wallet (1 NIS for each 20 minutes). The reservation and transaction is stored in MongoDB and the slot status is updated as “reserved”.
- **Reservation Confirmation**
  The user receives a confirmation of their reservation and details of the slot and their time window of arrival.
- **Gate Entry**
  Upon arriving at the parking location, the user clicks the "افتح البوابة" button in the app. An authorized request gets sent to the backend, which verifies the active reservation and sends a command to the ESP32 microcontroller at the entrance to open the gate as requested.
- **Gate Activation**
  The ESP32 receives the command and sends a signal to the motor driver and subsequently the motor is triggered to open the gate.
- **Slot Occupancy Update**
  After the car is parked in the designated parking space, the IR sensor detects the car's presence. The ESP32 communicates the occupancy status to the backend, which updates the parking slot status in MongoDB to "Occupied."

3.4.2. User Exit Flow

- **Reservation Period Monitoring**
  Once the user has parked, the system will track the session elapsed time with a duration associated with the reservation duration documented in the backend. (MongoDB)

- **IR Sensor Monitoring**
  The IR sensor will be monitoring whether the vehicle is detected in the reserved slot, and once the vehicle exits, the IR sensor will update the reserved space as vacant and will communicate the vacancy status to the Backend using the ESP32 controller.

- **Exit Time Validation**
  Once the backend gets the signal for the vehicle is vacant, it will check if the user exited in the reserved time:

  - **If within time**
    The session is marked as complete. The slot is reset to "available," and the user's record is cleared without additional charge.

  - **If overtime**
    The gate locked, and the user receives a notification on their app informing them that their parking time has expired and a fine of 50 NIS has been applied.

- **Chatbot Interaction for Fine Payment**
  To address the overtime fine, the user interfaces with the in-app chatbot. The chatbot notifies the user of the fine and asks if they would like to pay for it. Upon user permission in the chatbot, the fine is deducted from the user's digital wallet. Once the payment is settled, and successfully initiated, the backend unlocks the exit gate so the user can leave the parking facility.

3.5. Advantages of the Proposed System

1. **Real-Time Slot Monitoring and Reservation**
   The system allows users to check real-time parking availability and reserve slots in advance through a mobile or web interface, significantly reducing time spent searching for parking.
2. **Automated Gate Control**
   Integration with the ESP32 microcontroller enables automated control of entrance and exit gates, improving vehicle flow and reducing the need for human attendants.
3. **Overstay Management with Fine Enforcement**
   The system automatically detects overtime stays using IR sensors and applies a fine, which can be managed and paid directly through an in-app chatbot—ensuring accountability without manual enforcement.
4. **User-Friendly Chatbot Assistance**
   The inclusion of a chatbot provides intuitive assistance for payments, reservation issues, and exit procedures, enhancing the overall user experience.
5. **Digital Wallet Integration**
   A secure, cashless payment mechanism allows users to manage parking fees and fines directly within the app using their in-app wallet, streamlining financial transactions.
6. **Admin Dashboard and Control**
   Administrators have access to real-time analytics, manual overrides, slot status, and revenue tracking via a secure web-based dashboard.
7. **Cloud-Based Architecture**
   Using a cloud-hosted MongoDB database ensures real-time data synchronization, scalability, and ease of maintenance.
8. **Cost and Time Efficiency**
   The automation of booking, access control, and payments reduces operational costs and minimizes user delays.

3.6. Expected Outcomes

1. **Reduced Traffic Congestion**
   By minimizing the time spent searching for available parking, the system helps reduce on-road congestion and fuel consumption in urban areas.
2. **Improved User Satisfaction**
   A smoother, faster, and more transparent parking experience encourages repeated use and promotes digital adoption.
3. **Enhanced Revenue Control**
   With automatic billing, fine handling, and usage tracking, the system ensures accurate revenue collection and accountability.
4. **Data-Driven Management**
   Historical usage logs, payment records, and sensor data enable administrators to make informed decisions for optimizing space usage and future infrastructure planning.

5. **Scalability and Replicability**
   The modular architecture allows for easy deployment in other cities or expansion to larger facilities with minimal changes.

Chapter 4 Software Implementation

All digital communications between the user, the backend server, and the Internet of Things hardware are managed by the software part of our smart parking system. It guarantees data sharing and real-time connectivity, enabling customers to see availability, reserve parking spaces, and activate automated gate access. The architecture and fundamental elements of the software system, such as the database, frontend, and backend, as well as other features that improve system intelligence and usability, are described in this chapter.

4.1. System Architecture

The smart parking system's system architecture follows the layered structure, which comprises the frontend, backend, and database. The frontend, which is created using Flutter, permits users to carry out live system interactions such as making a reservation, checking availability, and operating the gate. The backend takes care of all the logic and the communication between the app, the database, and the IoT devices thus enabling the smooth operation of reservations and system updates. The database is used for secure storage of user data, slot status, and transaction history. This layout allows efficient, real-time communication between all the components for a trouble-free parking experience (refer to Figure below).

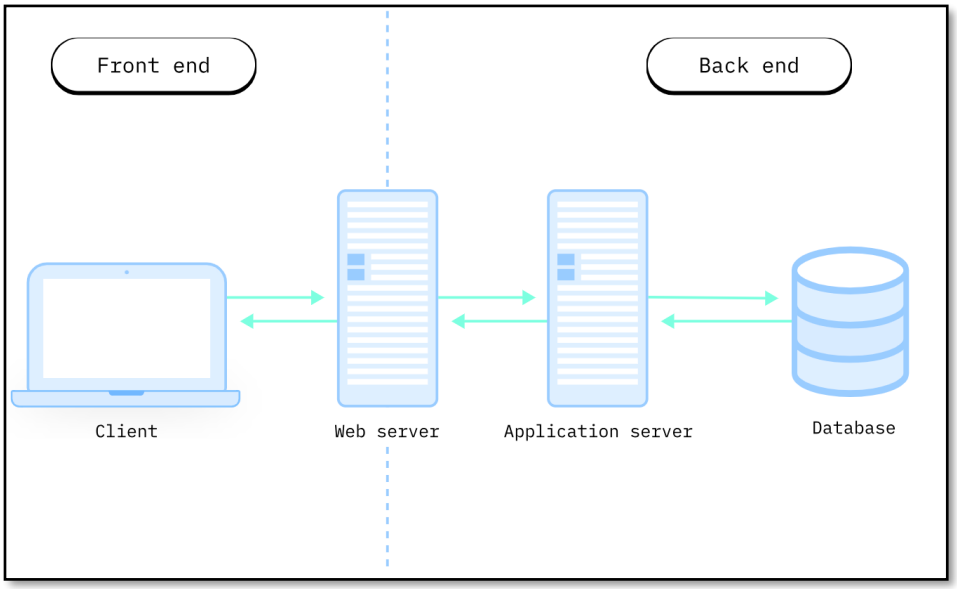

Figure 2: Software System Architecture

4.1.1. Frontend

A responsive and interactive user experience is provided across multiple platforms via the smart parking system's frontend. Flutter is one of the technologies used to create cross-platform apps that guarantee seamless operation on web and mobile interfaces. Users can take essential actions including managing their profiles, booking and canceling sessions, and seeing real-time slot availability through the interface. We make the communication with the backend servers has done by RESTful API calls using the HTTP package.

4.1.1.1. Design

Maintainability, scalability, and modularity are given top priority in the software design of the Smart Parking System. It has a client-server architecture in which RESTful APIs are used to connect the frontend mobile application to a backend server. This division enables the user interface and core functionality to be developed and optimized independently. User identification, money administration, parking space reservations, and real-time hardware command processing are just a few of the essential functional modules that the backend is designed to manage. Additionally, security features including token-based authentication, encrypted password storage, and user input validation are incorporated into the system design. By using timed expirations and temporary slot locking techniques to prevent reservation conflicts, the design also takes concurrency issues into account. All things considered, this architectural style facilitates a smooth user experience, strong performance, and extensibility.

4.1.1.2. User Interface and System Features

Section 1: User Onboarding

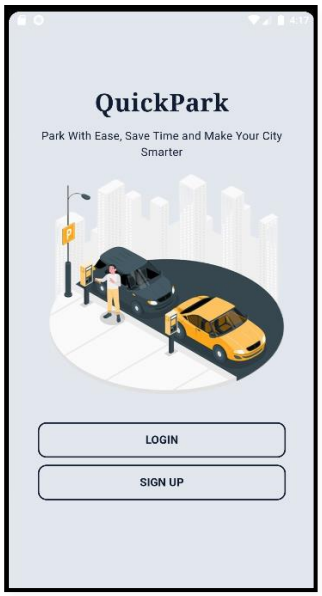

Figure 3: Main Page

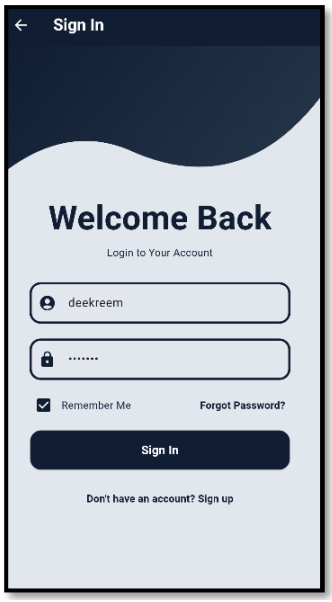

Figure 4: Log in Page

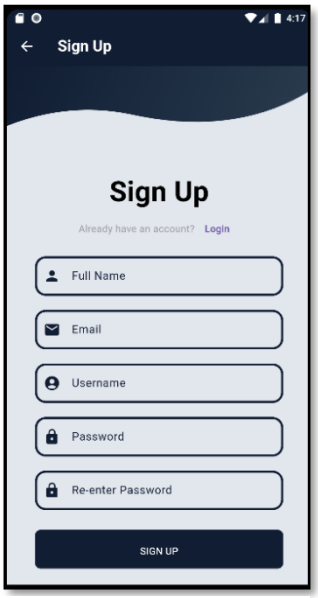

Figure 5: Sign Up Page

1. Interface 1: Start Page

The start Page is the welcome page of the QuickPark mobile application. It shows the name of the application along with the tagline "Park with Ease, Save Time, and Make Your City Smarter", emphasizing its aim to ease parking and encourage smart city infrastructure.

A central graphic visually represents smart parking, with instant context. Below it, the Login and Sign Up buttons are offered for users to proceed naturally based on their account status. The screen interactive and user-friendly first impression

2. Interface 2: Login In

Sign In screen where users will be able to log in to their accounts. It has fields for entering a username and password, "Remember Me" checkbox, and links to reset a forgotten password or create a new account. When the "Sign In" button is clicked, the application verifies the credentials entered to grant access. The layout is simple, elegant, and clean.

3. Interface 3: Sign up

The Sign-Up interface allows users to sign up by entering full name, email, username, password, and confirm password. A link is provided to redirect the user back to the login page in the event that it will be necessary. The application processes the information to sign up once it has been entered. Simplicity is the major concern of the interface with a modern and sleek style.

Section 2: Parking Functionality

Figure 7: Guide Screen

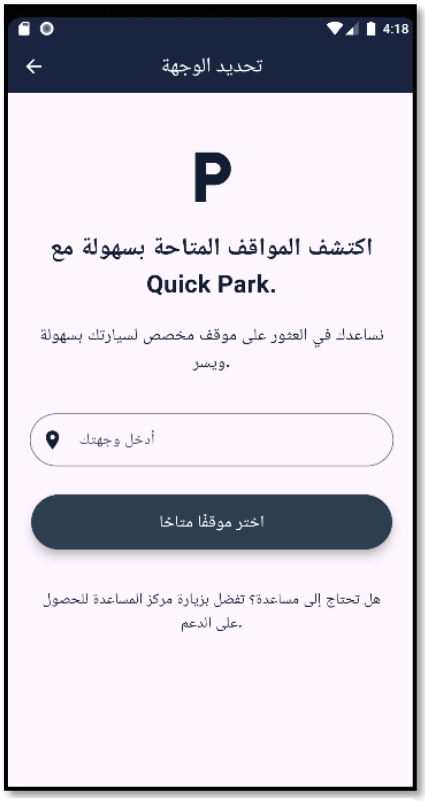

Figure 6: Destination Selection Screen

4. Interface 4: Guide Screen

The Guide Screen serves as an instructional screen designed to let the user familier with the features of the QuickPark application and how it works.

Beneath the header, a welcome message in Arabic - "مرحباً بك في تطبيق QuickPark" – followed by a brief introduction, explaining what are the features of the app and what enable the users to reserve a parking spot by simply selecting their destination.

A step-by-step visual guide is presented in a vertical timeline format, each step paired with a relevant icon and Arabic instruction:

1. **حدد موقعك وابحث عن موقف متاح**
   Determine your location and search for an available parking spot.
2. **قارن الخيارات واحجز موقفك**
   Compare available options and reserve your spot.
3. **تحرك نحو الموقف وأتمم الدفع**
   Drive to the selected spot and complete payment.
4. **لديك 15 دقيقة للوصول قبل الإلغاء**
   You have 15 minutes to arrive before the reservation is canceled.
5. **عند الوصول، اضغط زر الفتح**
   Tap the "Open" button upon arrival to access the parking.
6. **يمكنك تمديد الوقت إذا لزم الأمر**
   Extend the parking time if needed.
7. **اركن سيارتك واستمتع بوقتك**
   Park your car and enjoy your time.

A (Next) button is placed at the bottom-right corner to guide users to the next screen in the onboarding flow.

5. Interface 5: Destination Selection Screen

The destination Selection screen initiates the parking booking process by asking users to provide their destination location. The basic header is made up of the title "Destination Selection" and a back arrow for easy access.

In the center of the screen, the captioned input field "Enter your destination" with a captioned location pin icon is available, for use by users to enter their destination such that the app can suggest them suitable parking spots nearby.

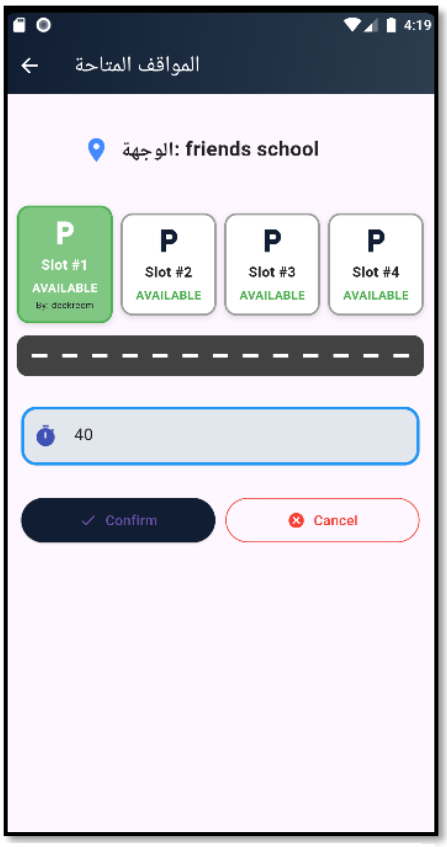

Figure 9: Parking Slot Selection Screen

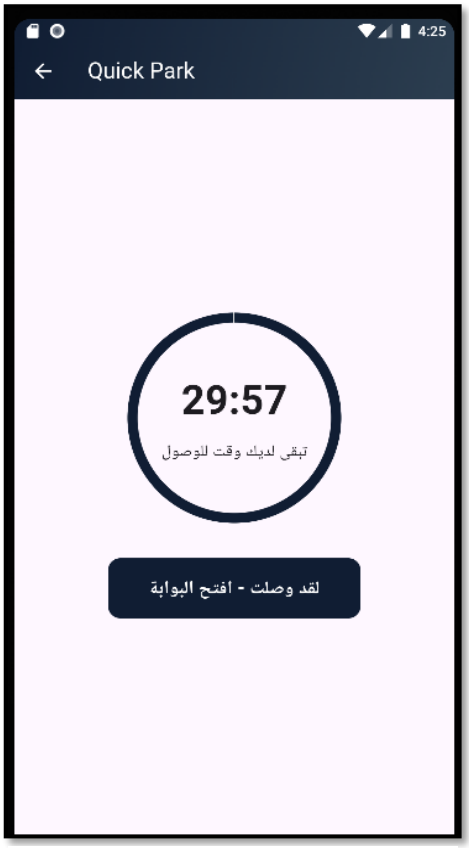

Figure 8: timer page

6. Interface 6: Parking Slot Selection Screen

Purpose: To enable users to view available empty parking spots in real time and reserve one for a period of time.

This screen indicates real-time parking slot availability at the user-chosen destination. A screen header, titled "Available Parking Slots".

Horizontally aligned parking slot cards in the center of the screen, each card has:

1. A parking "P" icon,
2. The slot number,
3. The availability status
4. The owner of availability.

Under the cards is an input field that can be edited with a timer icon, where users can enter the desired time of reservation.

At the bottom of the screen are two clearly labeled action buttons:

- Confirm button to confirm the reservation for the desired parking space.

- Cancel button to cancel the action and return to the last page.

7. Interface 7: timer page

Purpose: Inform the user of the remaining time to reach their reserved parking space and provide an option to confirm arrival.

This screen is shown after a parking slot is reserved. A countdown timer in the middle starts from *15* minutes, alerting the user to arrive within the allowed time. If the user doesn't arrive within 15 minutes, the reservation will be cancelled. There is a button stating "I have arrived – Open the gate" for the user to check in and proceed with entry.

Section 3: Wallet and Payment History Screen

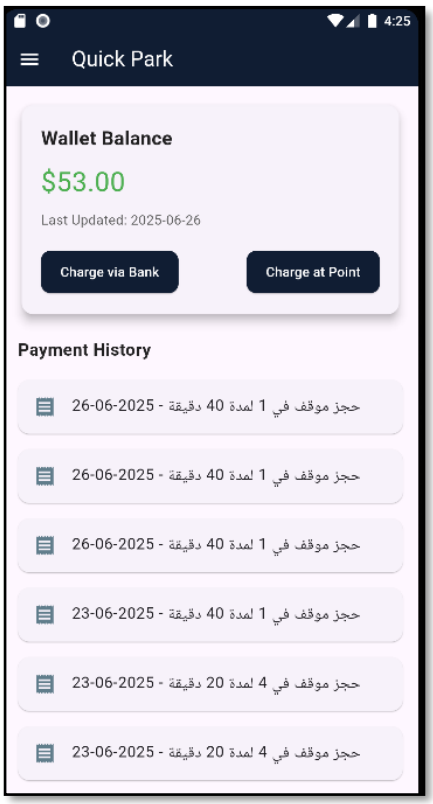

Figure 10: Wallet and Payment History Screen

Users are able to manage their account balance, top up by bank or point-of-sale, and view past parking payments on this screen. A minimal design separates the wallet summary at the top from the payment history below, allowing users to monitor expenditure and maintain control over their finances.

Section 4: Support System

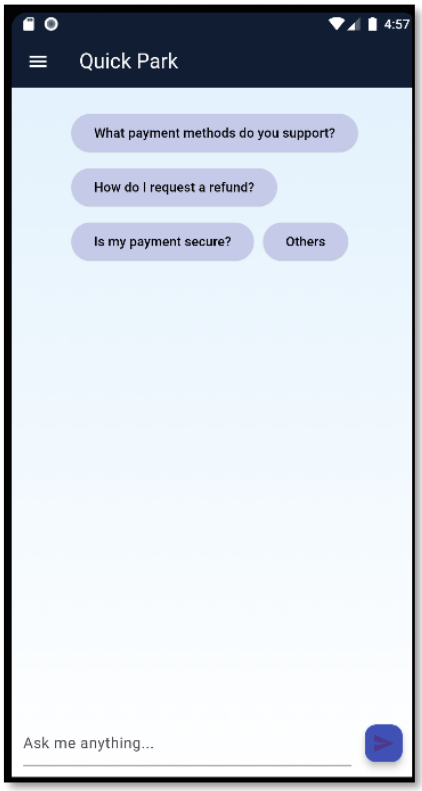

Figure 11: Support Chatbot Screen

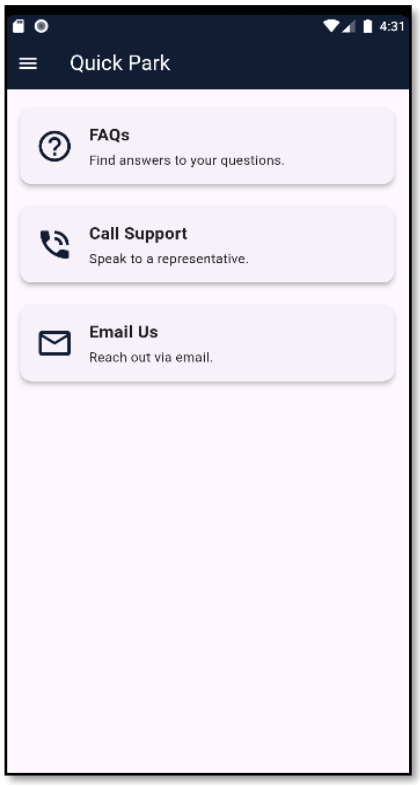

Figure 12: Support Screen

8. Interface 9: Support Chatbot Screen

QuickPark Support Chatbot is a conversational, intelligent interface that provides users with instant support. It displays a list of commonly asked questions as clickable options (e.g., "What payment methods do you support?", "How do I request a refund?"). These pre-established questions enable users to receive relevant information quickly without having to type.

Besides answering generic questions pertaining to payments and app usage, the chatbot also helps users who have experienced issues or infringements, such as being fined or having an issue with parking access. It guides them through the process of reporting issues, requesting reviews, or initiating support action, making the user experience responsive and accessible.

A free-text entry field named "Ask me anything…" is also available for users who prefer to enter specific or unconventional questions. The

chatbot responds contextually, enabling users to troubleshoot in real time.

This screen enhances user support by:

- Offering fast, self-service support.
- Minimizing the wait for human representatives.
- Guiding users to call or email support where needed.

Design Highlights:

- Clean, calming UI with subtle gradients.
- Large send button (paper plane icon) for easy submission.
- Seamless integration into the app's navigation for quick access from anywhere.

9. Interface 10: Support Center Screen

This screen functions as the **QuickPark Support Center**, giving users three clear options to get help:

- **FAQs** – Tapping this takes users to frequently asked questions.
- **Call Support** – This allows the user to contact a representative directly by phone.
- **Email Us** – This opens an email communication channel for user support.

Each support option is presented in its own card-style button, with a relevant icon for fast recognition. The layout is simple and prioritizes user assistance by offering multiple channels for getting help.

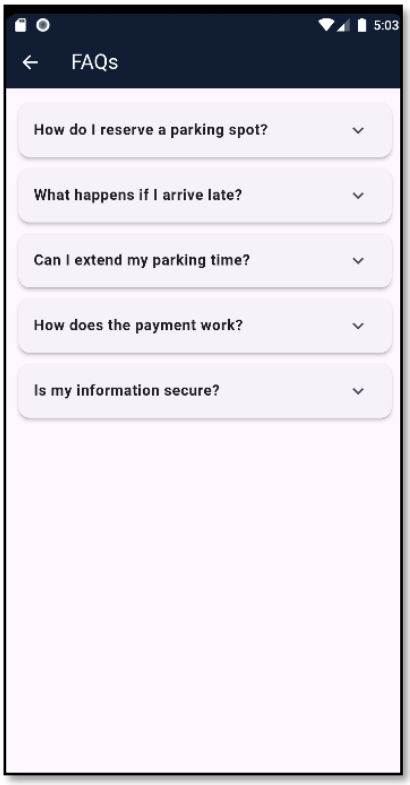

Figure 14: FAQ Screen

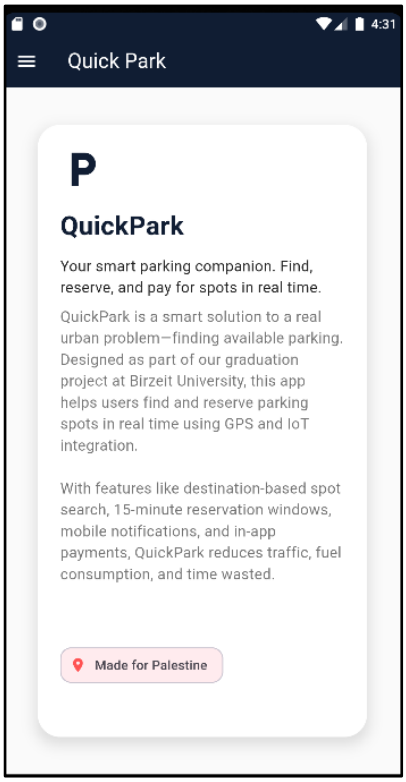

Figure 13: About Screen

10. Interface11: FAQ Screen

Presents a list of frequent questions in a collapsed format. Users can expand each question to view answers to questions regarding booking, timings, payments, and data protection—reducing the need for direct support.

11. Interface12: About / App Info Screen

Provides an overview of the QuickPark app, its application, and its features. Highlights how the app resolves real parking challenges using intelligent technologies and that it was developed as a graduation project for local use.

Section 5: Profile Screen

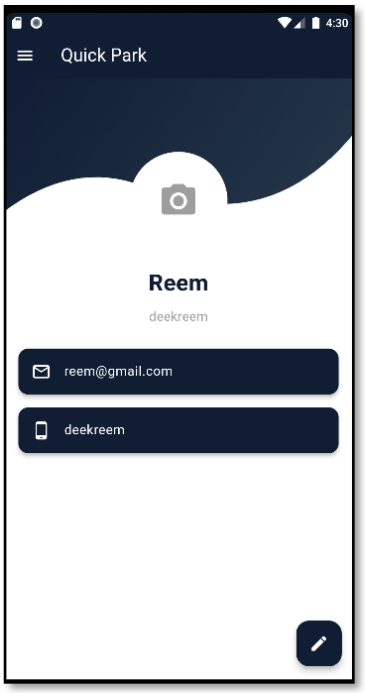

Figure 15: Profile Screen

12. Interface 13: Profile Screen

This page displays the user's personal profile in a simple and clean layout. At the top is a stylized background with an empty camera icon, which likely allows users to upload or change their profile picture.

Below, the user name is simply titled, then the username in small. Two distinct, rounded cards display the user's email address and username.

In the bottom right is a floating edit icon (pencil) showing that the profile information may be edited. This screen is a hub where users may see and potentially edit their account information.

4.1.1.3. Payment Top-up Web Application

Alternative Recharge Platform for Unbanked Users

For assisting users who prefer not to link their individual bank accounts with the QuickPark mobile application, a standalone independent payment web interface was established. The system grants authorized agents, such as supermarket owners, the convenience of manual recharging of balances and tracking payment history. The application ensures safe, accessible, and traceable financial transactions without linking with banks directly.

Interface 1: Welcome Screen

The welcome page is the entry point for the independent payment system. It gives two clearly defined options:

- Supermarket Owner: Directs the user straight to the recharge balance screen.
- Client Payment: Directs the user to a page where client balances can be viewed.

The interface is simple and not very computer literate friendly, offering convenience to non-computer users.

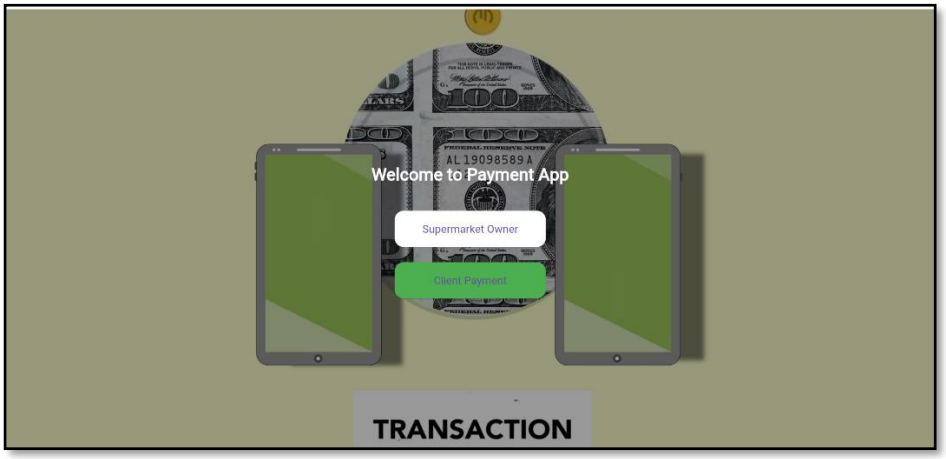

Figure 16: Payment Top-up Welcome Screen

Interface 2: Recharge a User's Parking Balance

The page offers an authorized agent the facility to manually recharge the QuickPark user's account.

The form takes the following input fields:

- User ID: It is used to identify the target user in the QuickPark database.
- Amount: It represents the amount to be deposited into the user wallet.

- Owner Telephone: It takes the contact number of the executing agent.

On submission of the form, the system credits the balance of the user and saves the associated transaction details for future use and accountability.

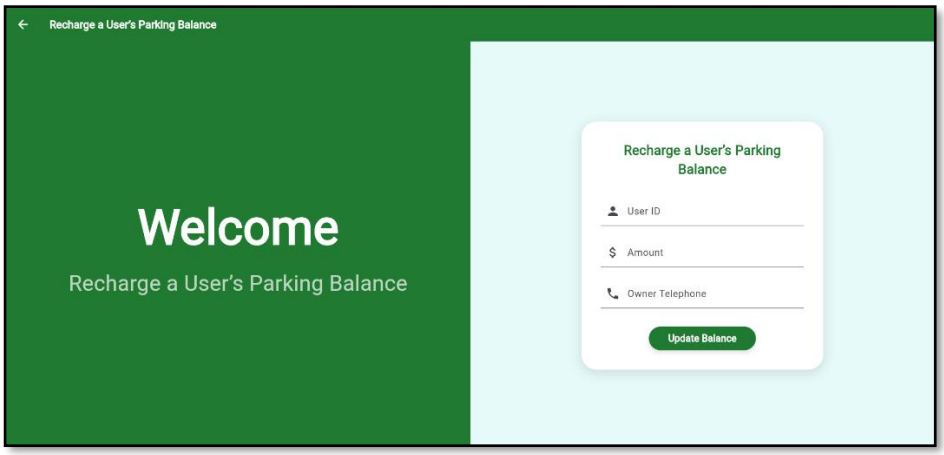

Figure 17: Recharge a User's Parking Balance

Interface 3: Check Client Balances

This interface allows access points' owner to monitor how much they have topped up for users.

The owner enters their registered telephone number, and the system fetch and displays the total value of all transactions under this number. This feature offers transparency and allows agents to maintain a current record of their financial contribution into the system.

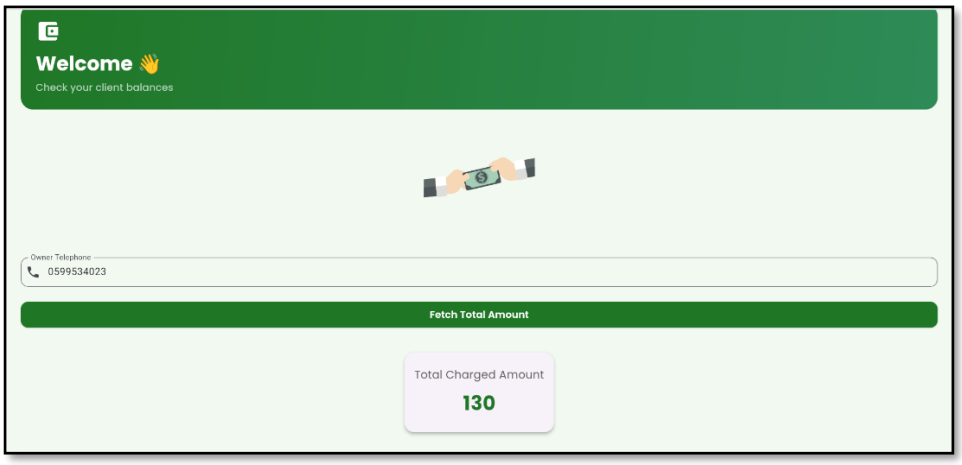

Figure 18: Check Client Balance

System Features

1. **Manual Balance Top-Up Without Bank Integration**
   The system allows users to recharge their parking balance without the need to connect a bank account or use online banking services, providing greater accessibility for individuals without digital financial tools.

2. **Agent-Facilitated Recharge Management**
   Designated agents, such as supermarket owners, are authorized to manage balance top-ups for users. This enables local, in-person transactions and extends the app's reach to users in offline environments.

3. **Secure and Traceable Transaction Records**
   Each recharge transaction is securely recorded, associating the payment with a specific user ID and the contact information of the agent performing the recharge. This ensures full traceability and integrity of the process.

4. **Comprehensive Agent Reporting Tools**
   The system provides agents with functionality to review the total amount they have recharged on behalf of clients. This promotes transparency and allows for accurate record-keeping and accountability.

4.1.1.4. Payment Top-Up Bank Page

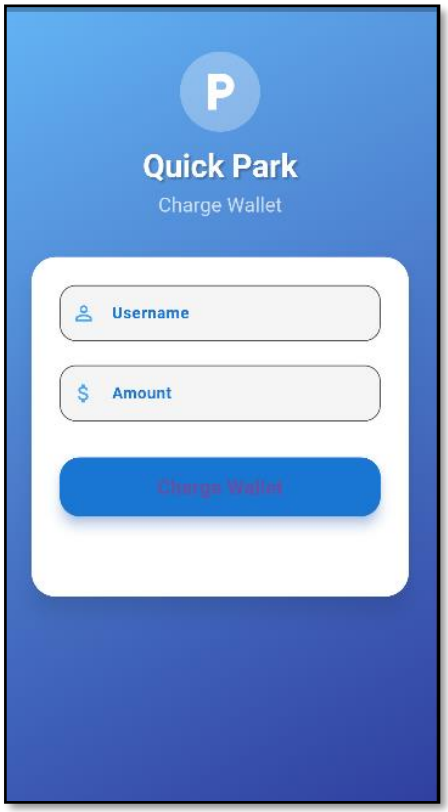

Figure 19: Payment Top-Up Bank page

System Features

1. **User Wallet Charging via Agent Interface**
   The system allows agents to charge a user's wallet manually by entering a username and the desired amount. This bypasses the need for user authentication or online banking, making it ideal for offline or semi-digital environments.
2. **Dynamic Bank Identification**
   The system accepts a dynamic bank name (e.g., "Quick Park"), enabling support for multiple partner banks or institutions using the same application infrastructure. This improves scalability and reuse of the interface.
3. **Real-Time API Integration for Transaction Processing**
   Upon submission, the system sends a structured JSON payload via HTTP POST to a backend API (/api/charge-bank) to record the transaction. The response is parsed and displayed to the agent immediately, reflecting success or failure.

4. **Client-Side Input Validation and Error Handling**
   Input fields for username and amount are validated on the client side to prevent invalid entries:
   - Username must be filled.
   - Amount must be a positive numeric value.
     Invalid entries trigger instant feedback, reducing backend errors and improving user experience.

4.1.2. Backend

The Express.js framework and Node.js are used in the construction of the Smart Parking System's backend. It serves as the main coordinator for all system functions, including slot reservation, money management, user authentication, and IoT device connection. It provides a secure RESTful API that communicates with a MongoDB Atlas database via Mongoose for schema modeling.

4.1.2.1. User Authentication & OTP Reset

Secure registration and login APIs are used to manage user accounts. To maintain security, passwords are hashed using bcryptjs before storing it. Throughout the program, user sessions are safely maintained using JWT (JSON Web Tokens). A one-time password (OTP) recovery procedure is also supported by the backend. OTPs are created and delivered to the user's registered email address using Nodemailer. After verification, users have a brief window of time to safely change their passwords.

4.1.2.2. Wallet & Payment Management

To enable cashless and effective user payments, the Smart Parking System integrates a digital wallet function. An integrated wallet is part of every user account and can be funded via two main methods:

- Direct top-up from a virtual bank
- Manual crediting via authorized third-party agents, such as supermarkets or local convenience stores.

Customers can use the online banking platform or mobile application of their bank to start a wallet recharge. The backend receives the recharge request when the transaction is completed, checks the information, and credits the user's wallet appropriately. As an alternative, customers can go to an authorized top-up location, where a user's unique identity or identifier is used by an operator to manually credit the wallet via the backend dashboard. Every transaction is documented with a reference to the top-up agent's owner number, which is usually their phone number.

The system keeps a thorough transaction log in the Changelogs database collection, recording the amount, timestamp, user ID, and charging agent, in order to guarantee accountability. Top-up agents, such as grocery employees, enter their registered owner number into the system at the end of each month. The parent firm can create a monthly report and issue a bank check or similar refund by using the system to compile the total amount that each agent has charged.

Apart from recharging, the wallet facilitates automated deductions for services like parking and slot reservations. The transaction amount, purpose, timestamp, and related user are all included in the separate Payment collection where these deductions are documented. Full

traceability of all wallet actions is provided by this dual-logging approach, which also improves fraud protection and guarantees financial transparency.

The smart parking platform's wallet and payment management system work together to create a safe, auditable, and expandable financial foundation.

4.1.2.3. Slot Reservation & Locking

Real-time parking spot management is supported by the system via a thorough API. Every space has a state (available, reserved, or occupied), and users can temporarily lock it for two minutes to avoid making duplicate reservations. The backend ensures that:

- No two users can reserve the same slot at the same time,
- A user cannot hold more than one active slot,
- Expired locks are automatically cleared.

Each change is instantly reflected in the MongoDB database, and users have the ability to lock, confirm, cancel, or release slots.

4.1.2.4. ESP32 Command Handling

To interact with hardware (e.g., gates or sensors), the backend includes an in-memory **command queue (**pending Commandss**)**. Flutter apps can post commands (e.g., open gate, mark slot occupied) using /api/cmd, and the **ESP32** microcontroller continuously polls /api/cmd/next to retrieve the next task. This simple and effective mechanism ensures decoupled and reliable communication with hardware, without needing a persistent socket connection.

4.1.3. Database

MongoDB Atlas is the chosen centralized database system for the smart parking application with respect to securely managing and saving the relevant data. The backend architecture makes use of Mongoose, which is an ODM library for MongoDB. Database transactions are made simpler, application level rules are enforced, schemas are defined, and data integrity is verified with Mongoose.

Several key collections make up the database architecture:

- Users: This collection maintains comprehensive user profiles, including full name, unique email addresses, usernames, securely hashed passwords, and digital wallet balances. The wallet subdocument updates based on the amount of money available for parking-related transactions.

- Slots: This collection models individual parking spaces, recording attributes such as the slot identifier, current occupancy status (available, reserved, or occupied), associated user references for reservation or occupation, and timestamps tracking state changes. Also, slot locking mechanisms are used to stop conflicting bookings and make sure that transactions are always the same.

- Charge Logs and Payments: These collections keep track of all financial transactions, including wallet charges, payment records, and bank top-ups. Each document has information about the user, the transaction amounts, the timestamps, and descriptive metadata, which makes it possible to keep track of all financial transactions and make sure they are correct.

The app has a clear RESTful API that makes it easy to do CRUD (create, read, update, and delete) operations for managing users, assigning parking spots, changing wallet balances, and viewing transaction histories. Atomic update operations and input validation make sure that data is consistent and correct across multiple processes that are running at the same time.

Chapter 5 Hardware Implementation

The hardware goal is simple: give every parking slot its own low-cost brain that can **sense a car, move a barrier, and report its state over Wi-Fi**. Each node uses one ESP32 board, a single IR break-beam sensor aimed across the bay line, a micro-servo flap, and a 9 V (regulated 5 V) power rail.

In the block diagram you will insert, the IR sensor feeds a GPIO pin on the ESP32; when the beam is broken the ESP32 checks the cloud, and—if the driver is authorised—sends a PWM pulse to the servo so the flap drops. Immediately after, the ESP32 publishes the new slot status back to the cloud, allowing the mobile app to update in real time. This closed, two-step loop (detect → actuate → publish) repeats independently on every slot, giving the whole car park instant, bay-level awareness without extra wiring.

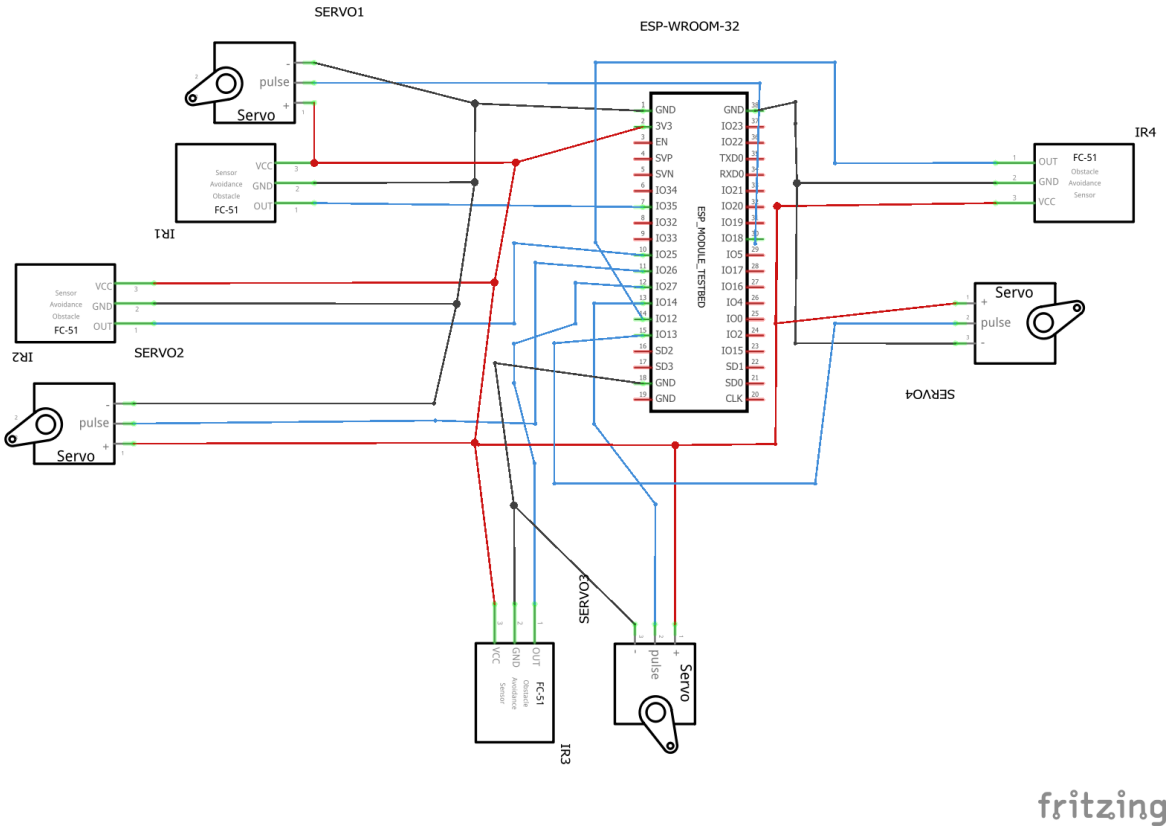

Figure 20: System Connection

5.1. Component Selection

5.1.1. ESP WROOM-32 WIFI Module

The ESP32 is a low-cost microcontroller that integrates Wi-Fi (and, on many versions, Bluetooth); in practice, vendors use the term *ESP32* for both the bare chip and the plug-and-play development boards that contain it.

Figure 21: ESP32 [8]

- **Key Features**

The ESP32 is often called a "single-chip IoT toolkit" because it rolls radio, processing and I/O into one tiny package. The features that matter most to our smart-parking node are:

- **On-board Wi-Fi (802.11 b/g/n)** – lets the controller push bay status to Firebase and pull "open flap" commands without an extra radio.
- **Dual-core 32-bit CPU, up to 240 MHz** – handles sensor polling, PWM servo drive and HTTPS/MQTT traffic at the same time.
- **Rich GPIO set (3 .3 V logic)** – 30 + pins that can act as digital I/O, ADC or PWM; enough to read four IR beams and drive four servos directly.
- **Hardware PWM generator** – produces a rock-steady 50 Hz signal for the servo without stealing CPU time.
- **Low-power and deep-sleep modes** – cut current to single-digit milliamps when the slot is idle, useful for solar nodes.
- **Secure boot and flash encryption** – protects firmware and Wi-Fi credentials in the field.
- **Mature software stack (Arduino core or ESP-IDF)** – ready-made libraries for Wi-Fi reconnection, JSON parsing and OTA updates shrink development time.

- **Low cost and small size** – a complete ESP32-WROOM-32 board costs well under US $10 and fits inside a palm-sized enclosure.

Together, these features allow a single ESP32 to **detect a car, drive the barrier and talk to the cloud**—all on one board, with no helper modules, perfectly suiting the budget-conscious design goals of our smart-parking system. [8]

- ## Why We Chose ESP32
  - **Cost:** cheaper than an Arduino Uno + separate Wi-Fi shield.
  - **Built-in radio:** no extra module, so wiring stays simple.
  - **Community:** ready-made libraries for HTTP, MQTT, JSON, and servo control shorten coding time.

- ## Working Principle in Our Parking Slot

  When a car breaks the IR beam, the sensor line on GPIO 35 goes LOW. The ESP32 notes the time, lowers the matching servo barrier with a PWM pulse, and posts a JSON status message (for example {"slot":"A1","state":"occupied"}) to Firebase over Wi-Fi. The same loop also listens for a "lift barrier" command from the cloud.

- ## Technical documentation

  The ESP32 uses a Tensilica Xtensa 32-bit LX6 CPU that can run in single- or dual-core mode at up to 240 MHz. It works from a 3.3 V supply and exposes more than 30 GPIO pins that can source or sink about 40 mA each—enough for LEDs, sensors and logic-level servos. Built-in hardware includes two 64-bit timers, an RTC timer for deep-sleep wake-ups, and on-chip Wi-Fi / Bluetooth radios. These features give the controller the speed and I/O flexibility needed for a real-time parking node.

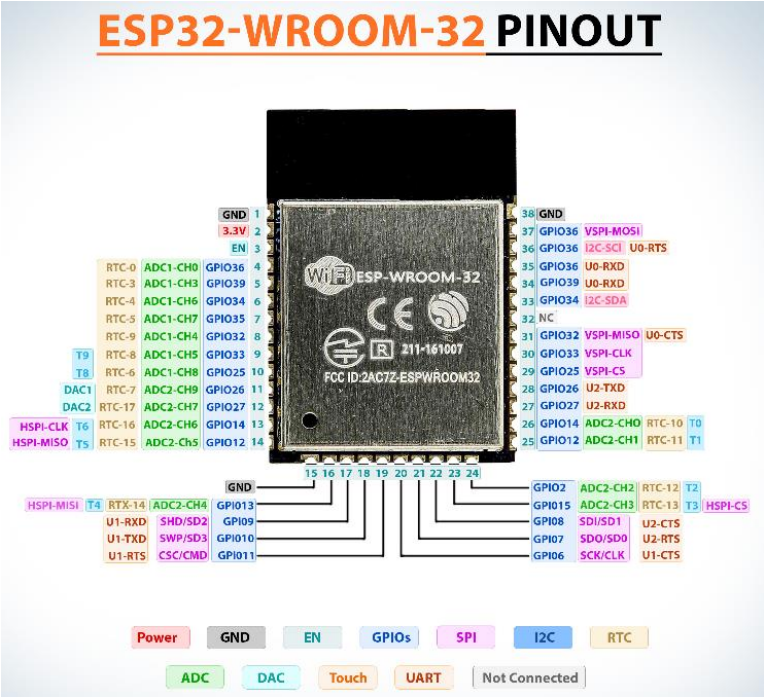

Figure 22: ESP32 PINOUT [9]

- **Role in the Smart Parking System**

  The ESP32 polls two IR sensors, drives one micro-servo, and pushes real-time updates to the cloud—everything needed for one fully independent slot. If Wi-Fi drops, the on-board watchdog restarts the radio and resends the last status, keeping the system reliable for users.

5.1.2. IR Sensor

An infrared sensor is a device that identifies infrared radiation striking its surface. Various applications of infrared sensors include proximity sensors (found in touchscreen smartphones and robots that avoid obstacles), contrast sensors (utilized in robots that follow lines), and obstruction sensors/counters (employed for goods counting and in security alarms).

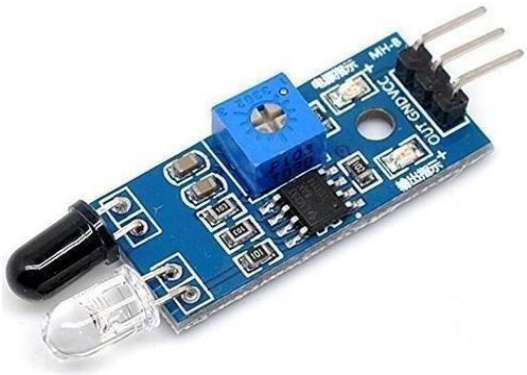

Figure 23: Infrared sensor module with adjustable sensitivity [10]

- ## Key Features

  The infrared sensor is ideal for our parking node because it gives a fast, low-power, and unmistakable **car-present / car-gone** signal.
  The features that matter most are:

  - **Line-of-sight, point-to-point link** – an invisible IR beam runs straight across the bay; only a vehicle that blocks that beam changes the slot state, so neighbouring bays never interfere.
  - **38 kHz carrier modulation** – the emitter blinks at remote-control speed and the receiver locks on to that frequency; sunlight and headlights are filtered out, preventing false triggers.
  - **Low supply and current draw** – operates from the same 3 – 5 V rail as the ESP32 and draws about 20 mA (LED) / < 5 mA (receiver), keeping the node solar-friendly.
  - **Simple digital output (active LOW)** – when the beam is blocked the receiver pin drops LOW; the ESP32 reads it directly—no ADC or echo-timing code needed.
  - **Adjustable detection range (≈ 2 – 40 cm)** – wide enough for a car bumper yet narrow enough to ignore distant traffic or pedestrians.
  - **Compact housing with M3 mounting holes** – quick to align on a post or wall; no custom brackets required.
  - **Ultra-low cost (≈ US $2 per pair)** – far cheaper than ultrasonic or camera alternatives, matching the project's low-budget target. [11]

- ## Why We Chose IR over Ultrasonic / Magnetic

Criterion	IR break-beam	Ultrasonic	Magnetic
Cost / slot	**≈ $2**	≈ $6	≈ $5

Installation	Simple face-to-face mount	Needs height & angle tuning	Must be buried or bolted under bay
False triggers	Sun- & head-light filtered (38 kHz)	Rain / angled bumpers can echo	Stray metal or strong magnets
Power draw	20 mA LED / < 5 mA Rx	20–50 mA per ping	Very low but constant poll

Why IR wins It is the cheapest, easiest to mount, and delivers a clean HIGH/LOW signal with minimal power-perfect for a low-budget, per-slot smart-parking node.

- ## Working Principle in Our Parking Slot

An infrared (IR) sensor detects the presence or absence of IR light on its surface. When arranged as a **break-beam pair** (one IR LED emitter, one phototransistor receiver) the device works like an invisible gate: the beam is either intact or interrupted . [12]
In our parking slot the emitter and receiver face each other across the bay entrance:

1. **Beam intact – slot free.**
   The receiver "sees" the 38 kHz-modulated beam, so its output pin stays **HIGH**.
2. **Car arrives – beam broken.**
   A bumper blocks the light; the receiver output flips to **LOW**. The ESP32 immediately timestamps this LOW edge, starts the billing timer, and—if the driver is authorised—sends a PWM pulse to drop the servo-controlled flap.
3. **Car departs – beam restored.**
   When the vehicle leaves, the beam reaches the receiver again; the pin returns **HIGH**. The ESP32 raises the flap and publishes a "slot free" update to the cloud.

Because the sensor delivers a clean HIGH/LOW signal and draws only about 20 mA (emitter) and < 5 mA (receiver), the node gains a fast, low-power, and unmistakable **car-present / car-gone** indication without needing distance measurements or complex image processing.

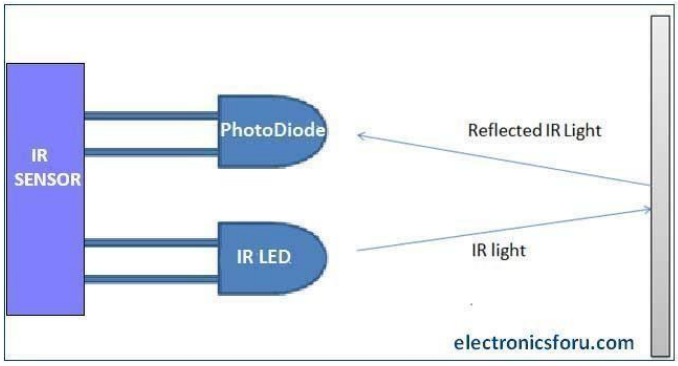

Figure 24: IR Sensor Working Principle [12]

- ## Technical documentation

The HW-201 / TCRT5000 sensor board uses an LM393 comparator to turn the analog phototransistor signal into a clean digital output. Key data are taken from the vendor sheet [13] and summarized below.

Spec	Value	Why it matters
Detection range	≈ 2 – 30 cm	Enough to cover a car bumper without seeing distant traffic
Field-of-view	~ 35 °	Narrow beam keeps neighbouring bays from crosstalk
Supply voltage	3.3 – 5 V	Runs from the same rail as ESP32 and servo
Supply current	≈ 20 mA (emitter + logic)	Low draw, suitable for solar operation
Output logic	Active LOW, LM393 open-collector	Directly read by ESP32 GPIO—no ADC needed
Board size / mass	31 × 14 mm, ~ 3 g	Fits easily in the flap housing

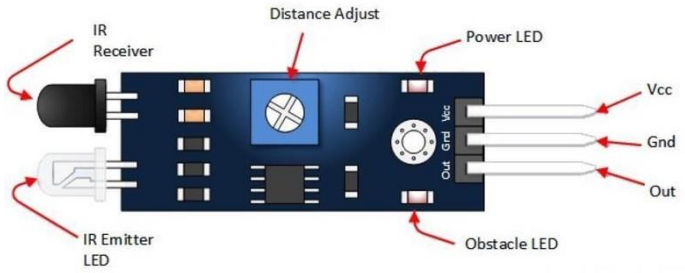

Figure 25: IR PINOUT [13]

These specs confirm the module's fit for a space-constrained, low-power parking node: it works on the project's 5 V rail, delivers a ready-made HIGH/LOW signal, and needs no extra optics or signal conditioning.

- ## Role in the Smart Parking System

Our design uses **active IR break-beam sensors**—an emitter sends a narrow infrared beam straight across the bay and a receiver watches for it. When a car blocks the beam, the receiver's output drops LOW, giving the ESP32 an instant "slot occupied" signal. The controller then lowers the servo flap (if the driver is authorised) and posts the new status to the cloud. When the vehicle leaves and the beam is restored, the ESP32 raises the flap and marks the slot free. Because each slot has its own sensor-servo pair, the car park gains reliable per-bay monitoring, automatic gate control, and real-time availability updates without cameras or complex image processing.

5.1.3. Micro-Servo (SG90)

The SG90 is a plastic-gear servo that can swing 0–180 ° with a standard 50 Hz PWM signal. It delivers roughly 1.5 kg · cm of torque—enough to raise or lower our lightweight flap. At idle it draws under 10 mA; during motion the peak is about 200–250 mA at 5 V, which the shared power rail can handle. Servos are easier to mount than stepper motors and do not need an H-bridge driver.

Figure 26: Micro-Servo [14]

- **Key Features**

The **SG90 micro-servo** is a compact, hobby-grade actuator that meets all of our parking-flap requirements while keeping the node small, simple, and low-cost Its standout qualities include: [14]

- Accepts a standard **50 Hz PWM signal** (≈ 1 ms = 0°, ≈ 2 ms = 90°), so the ESP32 can drive it directly.
- Delivers about **1.5 kg·cm of torque** at 5 V—enough to raise or drop the lightweight parking flap.
- Draws **< 10 mA when idle** and roughly **200 mA at peak movement**, which the shared 5 V rail can supply.
- Compact plastic case (**20 × 12 × 32 mm**); the small size lets the servo fit inside the flap housing without extra brackets.
- **Low cost** (≈ US $3 per unit), keeping the per-slot hardware budget on target.

- **Working Principle in Our Parking Slot**

The SG90 is driven directly from one ESP32 PWM pin at the standard **50 Hz** hobby-servo rate.

- **Flap-up (slot closed):** the firmware sends a **1 ms pulse** each frame, which the servo's on-board controller interprets as 0 °.
- **Flap-down (slot open):** an authorised entry triggers a **2 ms pulse**, rotating the horn to about 90 °.
  While moving the motor draws roughly **200 mA** for < ½ s; once it reaches the target angle the current falls below **10 mA**. The control loop is therefore event-driven and non-blocking—after the pulse is queued, the ESP32 goes straight back to sensor polling and cloud

traffic.

- **Role in the Smart Parking System**

  The SG90 provides the **physical "yes/no" barrier** for each bay. When the IR sensor reports a car and the cloud confirms a valid reservation, the servo lowers the flap, giving drivers a clear visual cue that the slot is theirs. When the vehicle departs—or if a reservation lapses—the ESP32 raises the flap again, making the bay available and discouraging unauthorised parking. In short, the servo turns digital decisions into a visible mechanical action, adding both security and user feedback to the smart-parking node.

5.1.4. ESP32 power-shield

To make each parking node completely self-contained, we plug the ESP32 ESP32-WROOM-32 into a small **power-control shield** that accepts a single battery input (5–28 V) and delivers two regulated rails: **5 V for the servo and IR sensor, 3.3 V for the ESP32 core**. The shield's on-board buck converter is rated for about 1 A continuous output—well above our 300 mA peak when the servo moves—while built-in reverse-polarity protection and a resettable polyfuse guard against wiring errors or shorts. A single 9 V battery therefore runs the whole slot for several hours of demonstration time without external adapters, and the same header-stacking format keeps the wiring tidy and repair-friendly. [15]

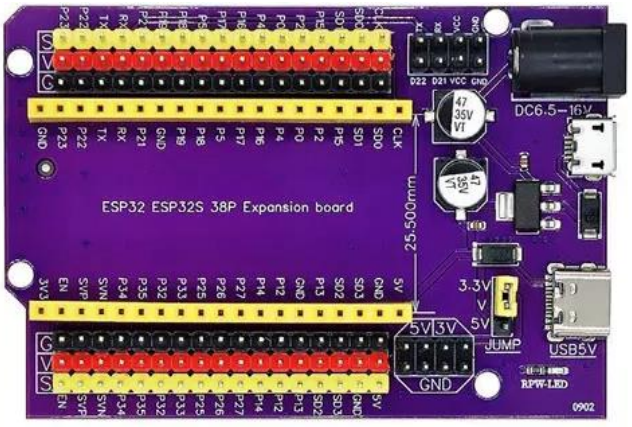

Figure 27: ESP32 power-shield [16]

5.1.5. Power Supply

Every component—ESP32, IR sensor, and SG90 servo—runs from a common **5 V rail** provided by the **ESP32 power-shield** you attached to the development board.The shield includes an on-board buck converter that accepts a **single 9 V battery** and steps it down to a stable 5 V output, plus an internal 3.3 V regulator that feeds the ESP32's logic rail.

- **Current head-room.** Peak draw is ≈ 300 mA (Wi-Fi burst ≈ 70 mA + IR pair ≈ 25 mA + servo surge ≈ 200 mA), well below the shield's 1 A rating.
- **Noise & surges.** A 470 µF electrolytic capacitor across the 5 V line smooths the servo's in-rush current, preventing brown-outs during flap movement.
- **Protection.** The shield's built-in reverse-polarity diode and polyfuse safeguard the circuit against incorrect battery insertion or accidental shorts.
- **Run-time.** A fresh 9 V alkaline (~550 mAh) powers the node for several hours—ample for demo sessions and short outdoor tests—while keeping the parts list simple and low-cost.
  This single-battery, shield-regulated approach eliminates external power bricks and wiring, making each parking node fully self-contained.

5.2. Firmware Logic

The ESP32 runs a lightweight loop that links each **hardware event** to one clear **action**:

Table 3:Firmware Logic Steps for ESP32

Step	Hardware signal	Immediate action (in firmware)
1 – Read sensors	Poll the IR-receiver pins (one per slot) every 10 ms.	Store current HIGH/LOW state in RAM.
2 – Detect change	LOW edge → beam broken ⇒ **car present**.HIGH edge → beam restored ⇒ **slot free**.	Update the internal state-machine (reserved / occupied / free).
3 – Actuate barrier	Cloud or mobile app publishes {"cmd":"OPEN","slot":"B2"}.	Send a 2 ms PWM pulse to the matching SG90 → flap lowers to 90 °.
4 – Publish status	Any state change from Step 2 or 3.	Post a JSON packet—e.g. {"slot":"B2","state":"occupied"}—to Firebase via MQTT/REST.

5 – Watchdog & reconnect	Wi-Fi lost for > 5 s.	Call WiFi.reconnect() and resend the last status when the link returns.

This tight **sense → decide → actuate → publish** cycle keeps hardware, cloud database, and mobile app in sync within a fraction of a second. (All deeper logic—wallet debits, timers, admin alerts—resides in the Software chapter.)

5.3. Physical Prototype

The following figure shows the benchtop prototype used to validate the hardware stack. The foam-board model contains 4 parking bays, each fitted with a IR sensor pair and a micro-servo–controlled flap barrier. An ESP32-WROOM-32 , mounted under the base, reads the sensors, drives the servos, and pushes status updates to Firebase over Wi-Fi exactly as in the full system.

This mock-up lets us demonstrate live slot detection, automatic barrier movement, and real-time app updates without a full-scale installation—proving that the complete sensing-to-cloud pipeline works end-to-end.

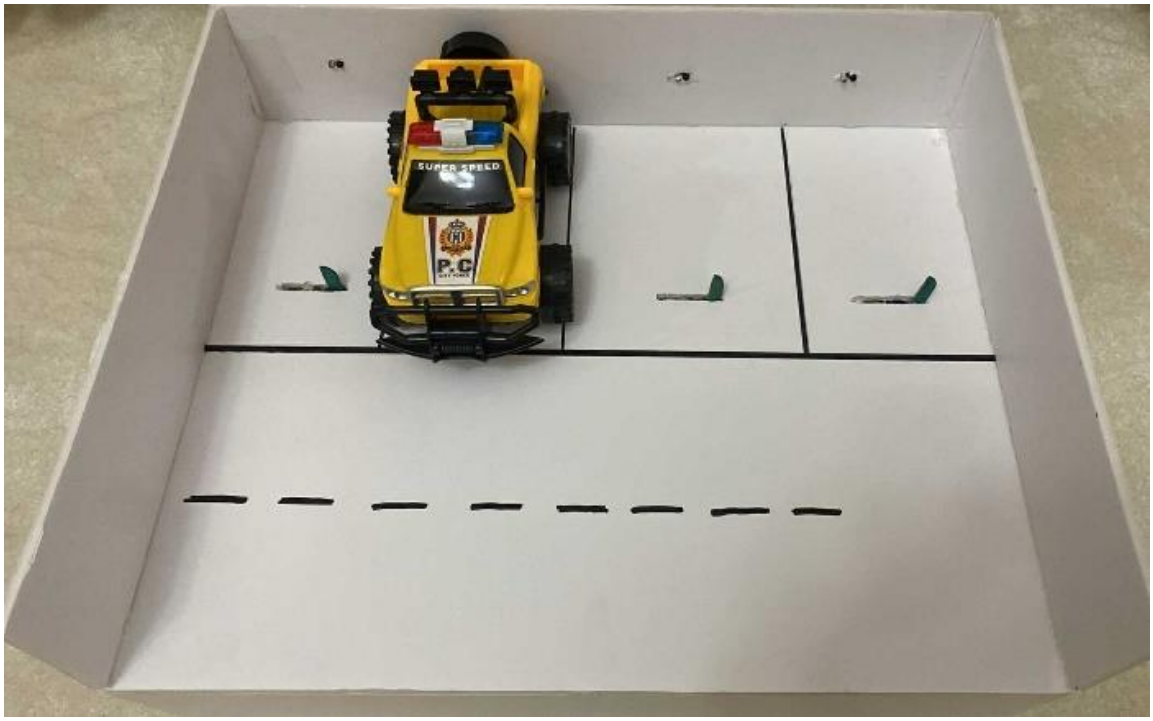

Figure 28:Completed three-bay prototype of the Smart Car Parking System used for hardware validation and live demonstrations

Chapter 6 Conclusion and Future Work

6.1. Conclusion

The project's Smart Car Parking System transforms the tedious task of seeking a parking space into a smooth online experience. Each bay is a standalone Iot node that consists of a micro-servo barrier, an ESP32 microprocessor, and infrared sensors. The node uses MQTT/REST to feed live status to a Firebase cloud over Wi-Fi, automates gate movement, and identifies cars in real time.

Drivers communicate via a Flutter mobile app that shows real-time availability, allows one-tap booking, and pushes messages for checkout, arrival confirmation, and overstay alerts. Payments are handled through secure web pages for electronic transactions or cash top-ups handled by reliable local agents—removing the need to link personal bank accounts—to address privacy issues that are prevalent in Palestinian cities.

The method advances municipal sustainability goals by reducing fuel use and tail-pipe emissions from the aimless cruising that clogs streets. In order to guarantee that drivers with disabilities have fair access in crowded cities like Ramallah, Al-Bireh, and Nablus, designated accessible bays are also enforced by constant surveillance. Automated billing and stringent overstay enforcement provide towns with an auditable, transparent cash stream that they can use to fund public infrastructure improvements.

The architecture provides an economical, workable route to cleaner, better urban transportation throughout Palestinian communities since each parking space functions as an own sensor-controller-actuator unit, allowing it to scale seamlessly from a pilot lot to a citywide deployment without requiring a significant redesign.

6.2. Future Work

The first priority is accessibility. We will replace the rule-based chatbot with an AI conversational assistant that supports natural language and hands-free voice commands, giving drivers with mobility impairments an easier, safer way to interact with the system while on the road.

Next, the reservation engine will gain an **"Accessible Bay"** flag. Bays marked with this flag will appear only to users who have verified disability credentials in their profile, preventing others from reserving or occupying spaces set aside for drivers with disabilities.
A third upgrade will equip selected bays with EV-charging pedestals. The mobile app will let drivers filter for "charging + parking," view real-time charger availability, and pay a single combined fee—supporting Palestine's growing population of electric-vehicle owners.

Finally, deep Google Maps integration will add live-location awareness. A geofenced rule (e.g., reservations allowed only when a driver is within 500 m of the lot) will discourage speculative bookings, improve turnover, and ensure that nearby drivers have priority access to available spaces.

References

[1] S. S. Verma, A. A. Bhardwaj, and P. P. Bhoyar, "Automated vehicle parking system using IoT," *Heliyon*, vol. 7, no. 5, e07161, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405844021011531

[2] A. Dutta, K. Mia, and S. Dasgupta, "A conceptual framework for smart parking management," *Information Systems Frontiers*, vol. 22, no. 3, pp. 629–647, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s10796-020-10044-1

[3] E. Turban and J. Brahm, "Smart card-based electronic card payment systems in the transportation industry," 2000. [Online]. Available: https://scholar.google.com/scholar_lookup?title=Smart%20card-based%20electronic%20card%20payment%20systems%20in%20the%20transportation%20industry&publication_year=2000&author=E.%20Turban&author=J.%20Brahm

[4] R. Jain and S. Soni, "IoT based smart parking system using cloud computing," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, Vellore, India, 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9120903

[5] A. R. Al-Ali, I. Zualkernan, and F. Aloul, "A mobile GPRS-sensors array for air pollution monitoring," in *Smart Innovation, Systems and Technologies*, vol. 104, Springer, 2019, pp. 413–421. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-15-0644-7_42

[6] A. K. Paul and A. K. Saha, "Design and implementation of a smart parking system using IoT," *Electronics*, vol. 10, no. 24, p. 3184, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/24/3184

[7] M. N. Siddiqui and M. A. Zaveri, "Overview of automated parking system using ESP32 and cloud computing," in *Proc. Int. Conf. on Emerging Trends in Engineering and Technology*, 2023. [Online]. Available: https://www.ijert.org/research/overview-of-automated-parking-system-using-esp32-and-cloud-computing-IJERTCONV11IS04008.pdf

[8] Guide to IoT: ESP32,*Nabto*, [Online]. Available: https://www.nabto.com/guide-to-iot-esp-32/.

[9] ESP32 Pinout, Datasheet, Features & Applications,*The Engineering Projects*, Dec. 2020. [Online]. Available: https://www.theengineeringprojects.com/2020/12/esp32-pinout-datasheet-featuresapplications.html.

[10] HW201 Infrared IR Sensor Module, *Circuits DIY*, [Online]. Available: https://www.circuits-diy.com/hw201-infrared-ir-sensor-module/.

[11] Infrared (IR) Communication: Basics, Features, and Applications, *RF Wireless World*. [Online]. Available: https://www.rfwireless-world.com/Terminology/Infrared-communication.html

[12] IR LED & Infrared Sensor Basics, *Electronics For You*, [Online]. Available: https://www.electronicsforu.com/technology-trends/learn-electronics/ir-led-infraredsensor-basics.

[13] HW201 Infrared IR Sensor Module,*Circuits DIY*, [Online]. Available: https://www.circuits-diy.com/hw201-infrared-ir-sensor-module/.

[14] Servo Motor – Basics, Pinout, Datasheet,*Components101*. [Online]. Available: https://components101.com/motors/servo-motor-basics-pinout-datasheet

[15] ESP32 Thing Power Control Shield Hookup Guide, *SparkFun Electronics*. [Online]. Available: https://learn.sparkfun.com/tutorials/esp32-thing-power-control-shield-hookup-guide/all

[16] "How to measure real time current consumption of ESP32," *ESP32 Forum*, 2024. [Online]. Available: https://esp32.com/viewtopic.php?t=37443