DESIGN AND IMPLEMENTATION OF IoT-BASED CAR PARKING MANAGEMENT SYSTEMS IN URBAN AREAS

BY: CIKURU MATABA Arnold

ROLL NUMBER: 202150371

Research project submitted in partial fulfillment of the requirement for award of advanced diploma in Electronics and Telecommunication Technology

Supervised by: Eng. MUSABYIMANA Jean Pierre

September 2024

DECLARATION-A

This research study is my original work and has not been presented for a Degree or any other academic award in any University or Institution of Learning". No part of this research should be reproduced without the authors' consent or that of ULK POLYTECHNIC INSTITUTE.

Student name: CIKURU MATABA Arnold

Signature: Date:

DECLARATION-B

I confirm that the work reported in this research project was carried out by the candidate under my supervision and it has been submitted with my approval as the UPI supervisor.

Supervisor name:

Signature: Date:

DEDICATION

I would like to dedicate this work to my parents, older brothers, little sister, and friends-they gave me means that were really priceless, financial, moral, and with their presence during the whole career path.

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to several individuals who have been instrumental in the completion of this research project. Firstly, I am immensely thankful to my twin brother Armand and my friend Victoire for their invaluable advice and great support throughout this journey.

I extend my sincere appreciation to my supervisor for his guidance and expertise, as well as to the university for providing the necessary resources and environment for this study. Special thanks go to the dedicated lab staff, Jean Damascene and Jean de Dieu, for their technical assistance and support.

I am also grateful to my brothers for their assistance in procuring materials and tools essential for the project. Their support has been a cornerstone of this work.

ABSTRACT

Finding parking in busy urban areas has become increasingly difficult due to the growing number of vehicles, leading to traffic jams and frustration for drivers who sometimes forget or lose items when parking. This project aims to solve these problems by creating an intelligent parking management system using Django, which makes parking easier for both parking lot owners and drivers.

The web application lets parking lot owners list their parking spaces on the platform so that drivers can check the real-time availability of parking slots. Drivers can see which slots are free, occupied, or reserved and can even reserve a spot in advance.

The system uses ESP32 and ESP8266 hardware to simulate real-time updates on parking slot status. It includes features for user authentication, parking management, reservations, lost and found items, and statistics making the whole process smooth and efficient. Users can sign up, log in, update their profiles, view lost and found items, view available parking slots, and make reservations, all in one place.

Keywords: Django, IoT, ESP32.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ACCRONYMS AND ABBREVIATIONS

API: Application Programming Interface

CLI: Command Line Interface

CSS: Cascading Style Sheets

ESP32: Espressif Systems 32-bit Microcontroller

FTP: File Transfer Protocol

GPIO: General Purpose Input/Output

GUI: Graphical User Interface

HTML: Hypertext Markup Language

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol Secure

IDE: Integrated Development Environment

IoT: Internet of Things

JS: JavaScript

JSON: JavaScript Object Notation

LDR: Light Dependent Resistor

NodeMCU: Node MicroController Unit

PCB: Printed Circuit Board

REST: Representational State Transfer

Rwf: Rwandan Franc

SQL: Structured Query Language

SSID: Service Set Identifier

SSL: Secure Sockets Layer

TAM: Technological Adoption Model

TLS: Transport Layer Security

UI: User Interface

URL: Uniform Resource Locator

VSC: Visual Studio Code

WiFi: Wireless Fidelity

CHAPTER 1: GENERAL INTRODUCTION

1.0 Introduction

In today's world, the rise in vehicle numbers has created significant difficulties in efficiently managing them. Advanced technologies like the Internet of Things are utilized in these systems to offer real-time parking slot availability information, simplifying the parking process. Instead of aimlessly searching for parking, drivers can utilize our platform to efficiently locate and reserve parking spaces, and in the event of a forgotten or lost item, there's still an opportunity to recover it through the platform. The introduction of IoT in 1999 has transformed various facets of our lives, including parking management. This section provides an outline of the challenges linked with traditional parking systems and the potential advantages of implementing intelligent car parking solutions. Additionally, we detail the goals and structure of this research, which aims to create an intelligent parking management system using Django and IoT principles.

1.1 Background of the study

The increasing number of vehicles worldwide has led to a significant problem in urban areas: a lack of available parking spaces. In bustling cities like Kigali or Kinshasa, the demand for parking spots far exceeds the supply, resulting in drivers spending more time in traffic and wasting fuel. This issue is prevalent in urban areas with concentrated commercial activities around shopping centers, high-rise parking structures, and technology centers. While traditional parking management systems have been in place for many years, they are proving to be inefficient in modern urban environments. Drivers often spend time circling in search of an open parking space, contributing to traffic congestion and environmental pollution. These challenges have prompted researchers and civil engineers to explore innovative solutions to alleviate the stress of finding parking spaces through the implementation of intelligent parking systems. (Barriga, et al., 2019).

1.2 Statement of the Problem

The primary issue this research aims to tackle is the difficulty and annoyance of locating parking spots in urban areas. Despite the surge in the number of vehicles and advancements in technology, traditional parking systems have not effectively handled the increasing demand for parking. This problem is evident in reliable sources such as urban mobility reports and studies on traffic congestion, which emphasize the adverse effects of inadequate parking infrastructure on urban mobility and quality of life. Current solutions, like conventional parking structures and manual parking enforcement, have been inadequate in meeting the requirements of modern cities. Although some

attempts have been made to introduce smart parking technologies, such as sensor-based parking guidance systems, these solutions often lack integration with broader urban mobility frameworks and fail to provide a comprehensive solution to the parking problem. What's lacking is a holistic approach to parking management that encompasses the latest developments in IoT technology, data analytics, and user-friendly design principles. (Raj & Shetty, 2024)

1.3 Research objectives

After reviewing existing research and understanding the current state of parking management systems, the objectives of this study are as follows:

1.3.1 Main objective

During this research, the main objective was to design and develop a simulation of the intelligent parking management system, incorporating features such as real-time parking availability updates and reservation capabilities, guided by best practices identified in the literature by

1.3.2 Specific objectives

a. To analyze the limitations of traditional parking management systems and understand the challenges encountered by urban environments. (Biyik, et al., 2021).
b. To explore the potential of integrating Django and IoT principles to develop an intelligent parking management system, drawing from previous studies in the field (Raj & Shetty, 2024).
c. To design a database to store all information and media files related to the platform.

1.4 Research questions

To guide this research and ensure that it meets its objectives, the following research questions have been formulated:

a. What are the primary weaknesses of traditional parking management systems in urban settings?
b. In what ways can the integration of Django and IoT principles enhance the performance and efficiency of parking management systems?
c. How to design a database of scale of this type of platform?

1.5 Scope

The scope of this study covers both theoretical and practical aspects. Geographically, the research will focus on urban centers, which are often faced with significant parking challenges, case of capital cities in the East Africa Community for instance. The theoretical scope includes an analysis of

traditional parking management systems, an exploration of Django and IoT technologies, and the development of a smart parking system prototype. The content scope covers the design, development, and evaluation of the prototype, with a specific focus on features like real-time updates and reservation capabilities, lost and found items management and the assessment of its effectiveness and ease of use.

1.6 Significance of study

This research addresses real-life challenges that many people face, including myself. I came up with the idea for this project while I was still in my first year. It motivated me to learn Python and later explore Django. My interest in microcontrollers grew after discovering Arduino, and during my internship, I learned about other microcontrollers like the ESP32.

Living in both the Democratic Republic of the Congo (DRC) and Rwanda, I have personally experienced the parking issues due to the problem described above and thereafter I observed a lack of an online platform to manage them. I have even forgotten important documents at parking areas because of these inefficiencies, which can be very frustrating.

1.7 Organization of the study

This study is structured into five chapters, each dedicated to specific aspects of the research. This ***Figure 1*** show a process chart that outlines the steps involved in each chapter of the study:

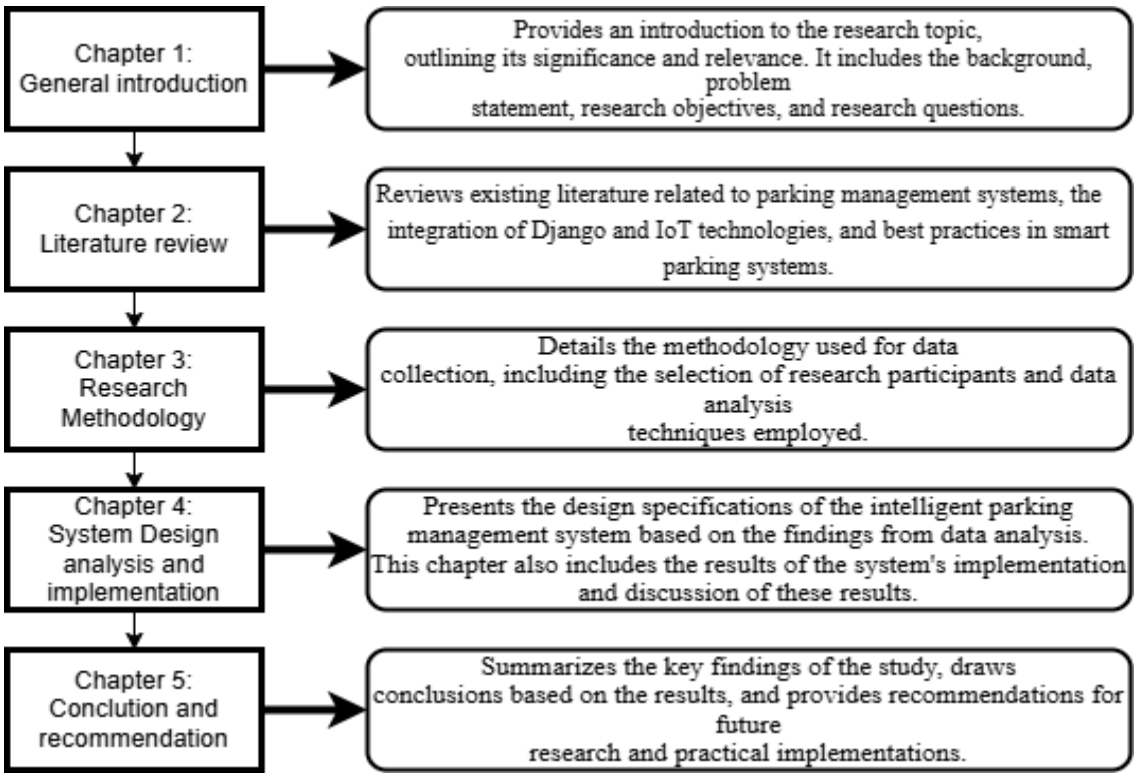


Figure 1 Organisation of the study

CHAPTER 2: LITERATURE

2.0 Introduction

Exploring the evolution from traditional methods to modern systems, particularly those integrating web development and IoT technologies, reveals important information and innovations. By examining the strengths and weaknesses of current practices and new technologies, we can identify gaps and opportunities for developing an advanced parking management solution designed for urban environments. Through this exploration, we aim to place our research within the wider field, highlighting both the progress made and the challenges that remain.

2.1 Concepts, Opinions, and Ideas from Experts

Researchers such as Smart Parking Systems: Reviewing the Literature, Architecture, and ways forward (Barriga, et al., 2019), the authors of the study on Smart Parking Management algorithms for smart city (Jemmali, Melhim, T. Alharbi, Bajahzar , & Omri , 2022), (Barriga, et al., 2019) have highlighted various aspects of this transformation, from the limitations of outdated systems to the potential of integrating advanced technologies like Django and IoT.

2.1.1 Limitations of Traditional Parking Management Systems

The study describes the challenges posed by traditional parking systems. With the rise in car ownership, urban areas face parking problems and increased carbon emissions. The study shows the need for smart parking solutions to reduce the time spent searching for parking and to minimize gas emissions. They underline the incorporation of advanced systems together with sensors to be more effective in creating a smart parking solution.(Barriga, et al., 2019): view ***Figure 2*** .

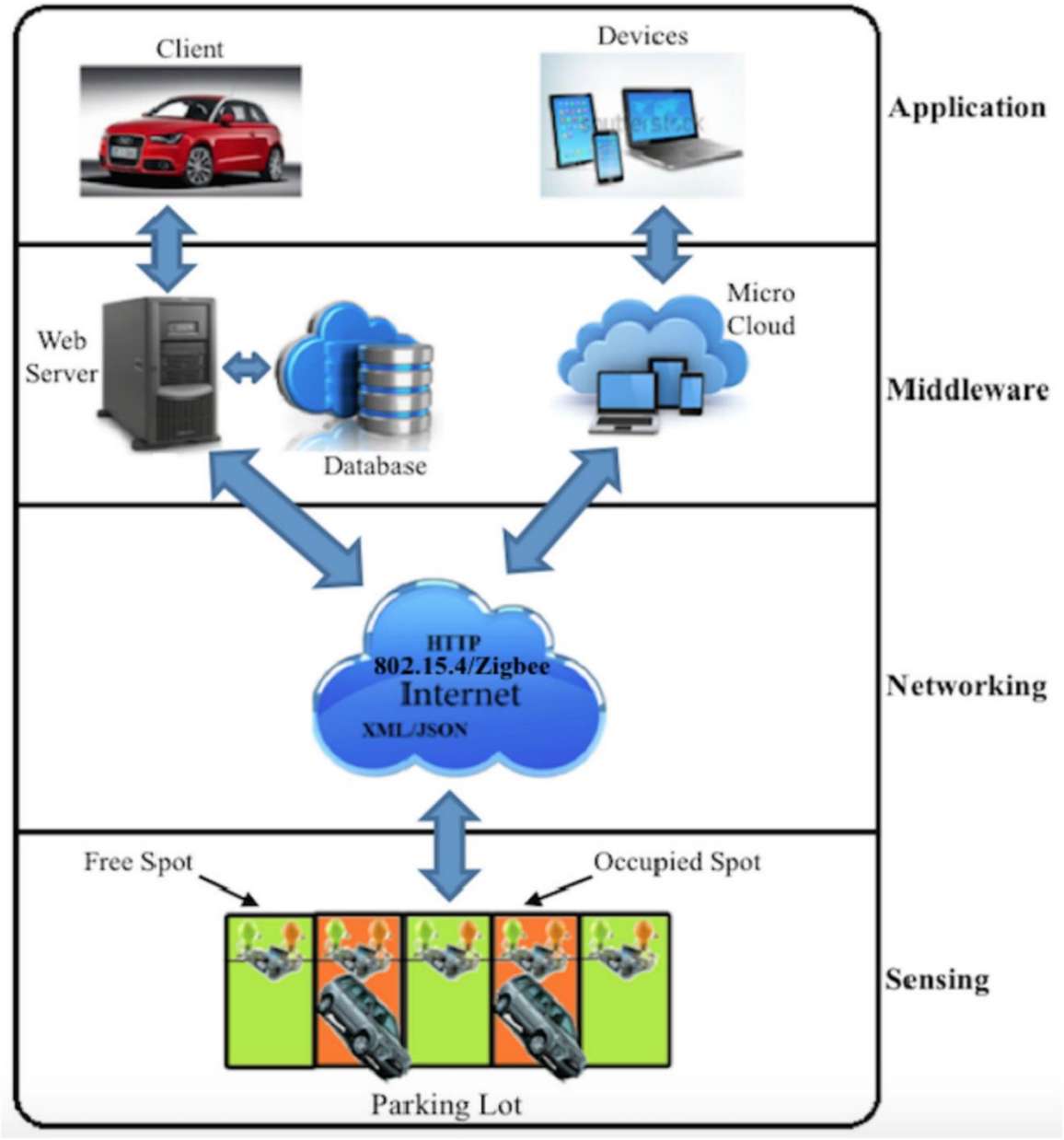


Figure 2 Layer architecture for integrated smart parking systems. Adapted by Biyik et al

Abrar Fahim et al provide a comparison and analysis of various smart parking systems, the limitations of traditional methods. They highlight the technological approaches, sensors used, and networking technologies to show advantages and disadvantages of these systems. Their work explain the need for more solutions to solve the parking challenges in urban environments, read sensors used in such sytems in the organisation of their study ***Figure 3*** (Abrar Fahim, Hasan, & Chowdhury, 2021).

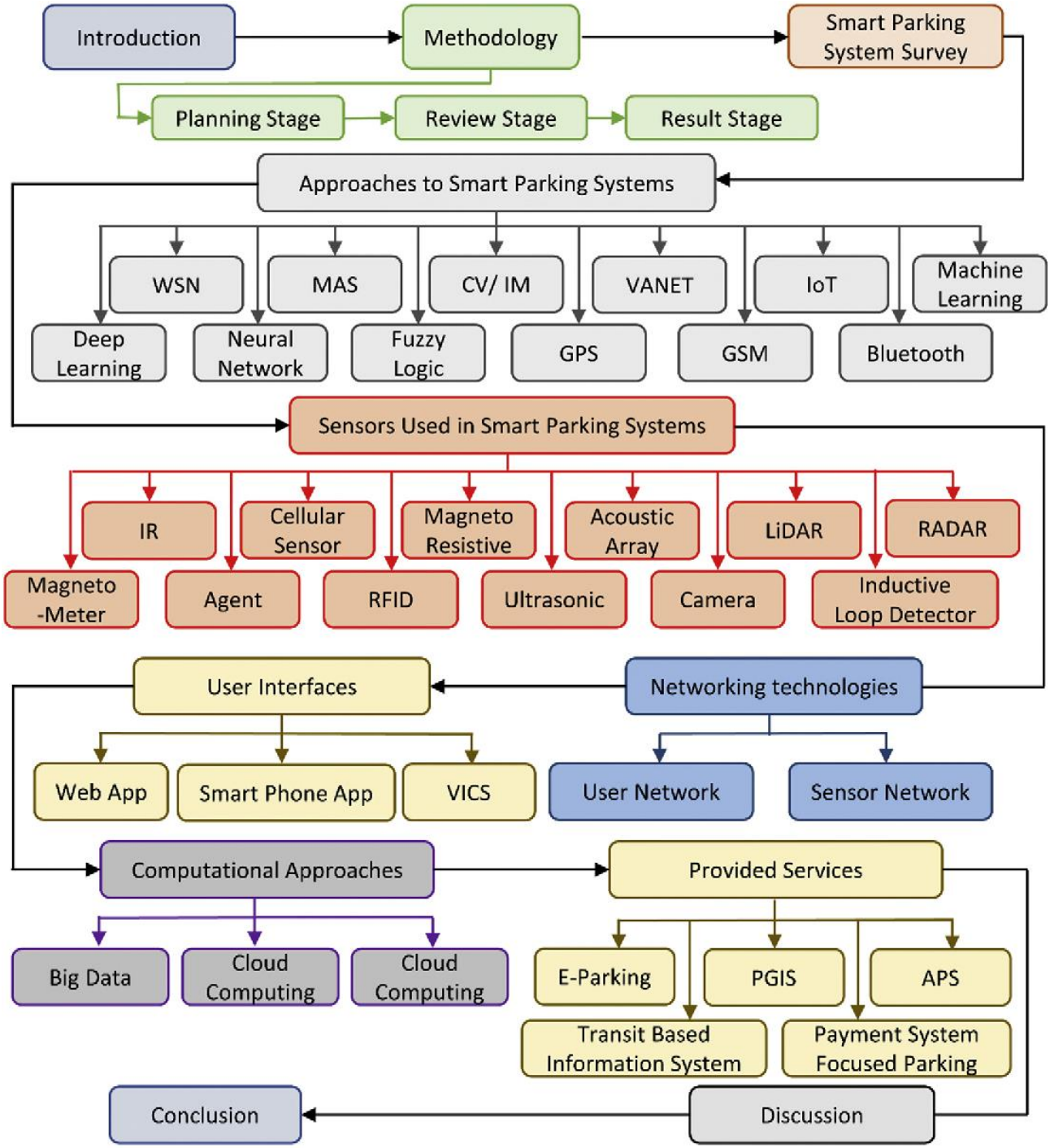


Figure 3 Organization of study by Abrar Fahim et al

2.1.2 Integration of Django and IoT Principles

The study "Smart Parking Systems Technologies, Tools, and Challenges for Implementing in a Smart City Environment" explores the integration of IoT and machine learning in smart parking systems.

These technologies can improve parking management in reducing traffic and optimizing fuel and energy consumption with the potential of combining Python web framework, with IoT (Raj & Shetty, 2024).

2.1.3 Design and Development of Intelligent Parking Systems

Barriga et al. and also Raj & Shetty (***Figure 4***) identify components of smart parking, such as sensors, protocols, and software(Barriga, et al., 2019) (Raj & Shetty, 2024).

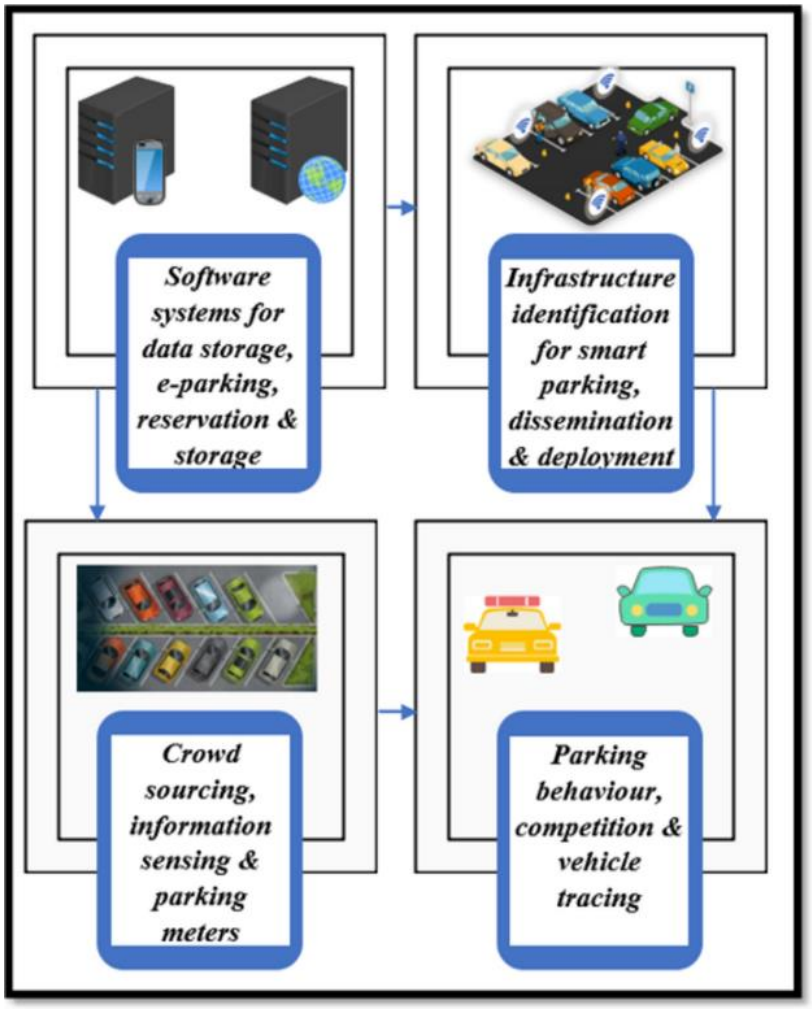


Figure 4 Development of Smart Parking by Raj & Shetty

Fahim and Hasan also explore the design and development of smart parking systems. Their analysis includes technological approaches and user interfaces, providing a clear understanding of best practices. This comparison aids researchers in identifying the most suitable smart parking systems for different urban environments (Abrar Fahim, Hasan, & Chowdhury, 2021).

2.1.4 Effectiveness and Usability of Smart Parking Systems

The study "Research on Smart Parking Solution" outlines methodologies for evaluating the effectiveness and ease of use of smart parking systems. They emphasize the importance of design focused on users and evaluation, providing insights into field testing and user feedback. This research ensures that smart parking systems meet user needs and improve overall efficiency (Lumbanga, 2021).

Barriga et al. also discuss the impact of different components on system performance, offering a framework for evaluating and improving usability (Barriga, et al., 2019).

2.1.5 Identified Gaps and Focus Areas

While these studies provide valuable insights, several gaps remain. There is a notable lack of efficient visibility of available parking slots and low booking capabilities in current smart parking solutions and absence of the management of the lost items. Despite advancements in technology, many systems still struggle to provide real-time updates and seamless reservation options. This gap highlights the need for further research and development to enhance the user experience and operational efficiency of smart parking systems.

2.2 Theoretical perspectives

As we explore parking management systems, we must use theories that help us understand the principles behind intelligent solutions. These theories offer insights into how technology, user behavior, and city infrastructure work together.

2.2.1 Technology Acceptance Model (TAM)

The Technology Acceptance Model (TAM) is used to understand people's adoption of new technology. In this project, we apply TAM to analyze user perceptions of smart parking features like real-time parking availability updates and reservation options. Our goal is to determine if these features are perceived as helpful and easy to use, thereby ensuring the system's widespread acceptance and success.

(Technology Acceptance Model, n.d.)

2.2.2 IoT Framework for Smart Cities

The Internet of Things (IoT) is crucial for modern cities, connecting physical objects like parking sensors to digital systems for real-time data collection and analysis.

a. Sensor Deployment: Sensors are installed in parking spaces to detect whether they are occupied or available. These sensors send data to a central system.
b. Real-Time Data Collection: The central system collects data from all sensors continuously, providing a real-time view of parking space availability.
c. Data Analysis: The system analyzes the data to identify parking availability.
d. User Interface: The information is displayed to users through a mobile app or digital signs, showing them where parking is available and allowing them to reserve spots in advance.
e. Integration with City Systems: The smart parking system can be integrated with other city management systems, such as traffic control and public transportation, to optimize overall city mobility.

2.2.3 Conceptual Framework

Four research, we combine the key aspects of TAM, IoT, and urban planning and sustainability. Here is an outline of this framework referring coming Figure 2:

a. User Perception and Acceptance (TAM)
   This evaluates the user's perceived ease of use and usefulness of the smart parking system, focusing on its navigation and real-time availability updates.
b. Technology Integration (IoT Framework)
   Sensor Deployment, Data Collection and Analysis, and Communication Protocols.

c. Urban Planning and Sustainability

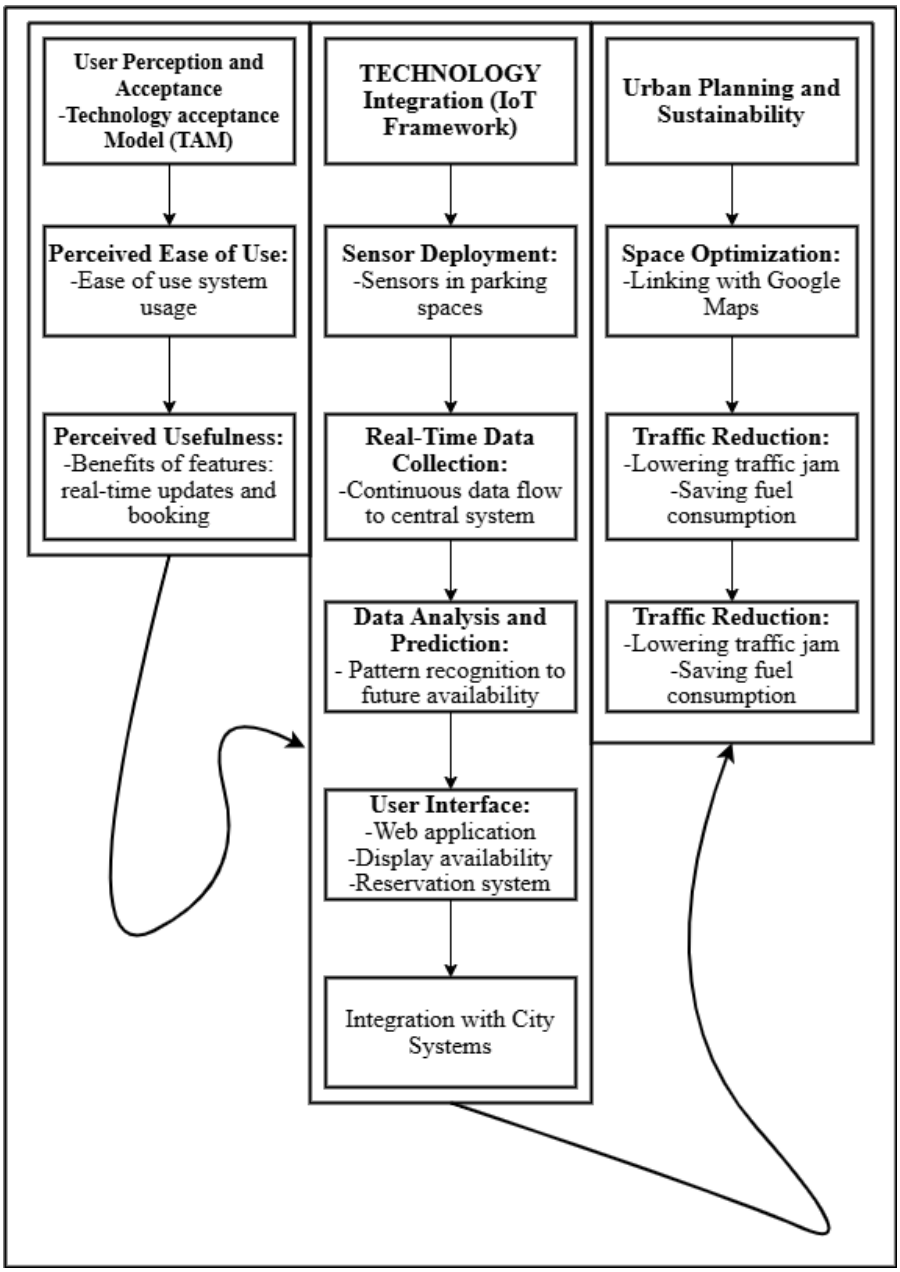


Figure 5 Conceptual framework

2.3 Related Studies

This section provides a comprehensive analysis of previous research on smart parking systems, outlining their methods, findings, and areas for improvement.

Study by Biyik et al. (2021)

Refering to Figure 2, Biyik et al. provide a detailed review of smart parking systems, focusing on their architecture and technical elements. They emphasize the role of IoT in making parking more efficient, examining various sensors and communication technologies. The study found that smart parking systems reduce the time spent searching for parking and decrease carbon emissions. (Biyik, et al., 2021).

Smart Parking Systems Technologies, Tools, and Challenges (2024)

This study surveys smart parking systems, emphasizing their role in reducing traffic congestion and improving urban mobility. The authors discuss technological advancements, challenges, and the need for cost-effective, secure, and standardized solutions. Refer to Figure 4. (Raj & Shetty, 2024)

Smart Parking: A Literature Review from the Technological Perspective (2019)

The study examines smart parking systems, highlighting sensors, communication protocols, and software solutions, but highlighting lack of standardization and environmental benefits, while calling for further research on long-term effectiveness and urban integration.
(Barriga, et al., 2019)

Study by Fahim and Hasan (2021)

Refer to Figure 3, Fahim and Hasan review smart parking systems, analyzing their effectiveness in controlled settings and real-world conditions, highlighting the need for adaptable solutions for diverse urban scenarios. (Abrar Fahim, Hasan, & Chowdhury, 2021).

Identifying Gaps in the Literature

From these studies, several critical gaps occur that our research aims to address:

a. Affordability
b. Visualization of Parking Slots
c. Data Security
d. Local Reservation Systems
e. Lost and found items
f. Adaptability

CHAPTER 3: RESEARCH METHODOLOGY

3.0 Introduction

Our research needs to have valid data. We describe the process by which the data were collected and analyzed. This will describe the tools and techniques to be used for reliability and validity. The information then informs our results and conclusions of the research. To do so, some effective methods of data collection will be chosen to give an overview of the existing smart parking systems and their potential, based on which the analysis can be carried out.

3.1 Research Design

The research design uses Action Research, a hands-on, flexible approach for practical problem-solving in real-world situations.

Why Action Research?

a. Working Together: In this study, I work closely with users, parking managers, and drivers (all friends) to find and fix problems with parking and managing lost items.
b. Step-by-Step Process: Action Research involves a cycle of planning, acting, observing, and reflecting.
c. Practical Focus: The main goal of this research is to solve practical problems.
d. Methodology: I will combine quantitative methods, like collecting data on parking usage and availability, and qualitative observations to understand user interactions with the system.

3.2 Research population

The study focuses on urban parking facilities and their users, including street parking, garages, and lots, and their target population, including drivers. In this project, the target population includes:

a. Drivers: This group includes anyone who uses parking facilities, both in the Democratic Republic of Congo (RDC) and Rwanda and countries surrounding.
b. Parking Managers: These are individuals responsible for managing parking.
c. Urban Residents: This includes people who live in urban areas where parking is a common issue.

Sample Selection:

a. Inclusion Criteria: The study will include drivers who frequently use parking facilities, parking managers who oversee these facilities, and urban residents familiar with parking issues.

b. Exclusion Criteria: Individuals who do not drive or use parking facilities, or who do not reside in urban areas, will be excluded from the study.

3.3 Sample Size

The sample size for this project will include users who interact with the smart parking platform: regular users, parking managers, and administrators. The exact number of participants depends on the availability of individuals willing to test and provide feedback on the platform.

3.3.1 Sampling Procedure

For this project, there is use of purposive sampling, which means choosing people who are willing to test the platform, acting like superadmins, parking managers, and normal users.

a. Superadmins: To test if they can manage everything, including users and parking managers. They also check if the system is secure.
b. Parking Managers: To test how well the system helps them manage lost and found items and how they can respond to normal users' questions and problems and reports of lost items.
c. Normal Users: To test the platform by doing things like parking (using toy cars for practice), reserving spots, finding lost items, and reporting lost items, but also when there are many, test the responsivity of the platform.

3.4 Research Instrument

3.4.1 Choice of the Research Instrument

For my project, which involves developing smart parking management. The checklist included various scenarios, such as:

a. Super Admins: Managing user permissions, overseeing parking lot data, and handling system security.
b. Parking Managers: Managing lost and found items, responding to user inquiries, and monitoring parking space availability.
c. Normal Users: Booking parking spots, navigating the parking lot (simulated with toy cars), reporting and searching for lost items.

3.4.2 Validity and Reliability of the Instrument

I did a small pre-test with some groups not involved in the main study, just to assure that it really captured the user interaction properly. These have helped locate unclear instructions and components that were missing. From this result, I refined the checklist so that it would be more valid and reliable, consistent in the observations.

3.5 Data Gathering Procedures

3.5.1 Before Data Collection

a. Preparation: Observe objectives for Super Admin, Parking Manager, and Normal Users using an Observation Checklist, select diverse participants, and ensure the parking management system is operational and configured for testing.
b. Brief Participants: The process involves providing instructions on system interaction and obtaining participant consent for data collection and observation.

1.5.2 During Data Collection

a. Observation: Observation Checklist helps record user interactions, observe usage, and gather feedback on system usability.
b. Document Findings: Take detailed notes on user interactions, including problems and suggestions for improvement, and if applicable, record video/screen recordings with consent for later analysis.

3.5.3 After Data Collection

a. Data Analysis: The process involves reviewing notes, summarizing findings, and identifying trends in user feedback to identify common issues and recurring problems.
b. Report Findings: The process involves organizing collected data into a report format and providing recommendations to enhance the system's functionality and user experience.
c. Follow-Up: The process involves implementing changes based on feedback and conducting further testing to ensure the system effectively addresses the identified issues.

3.6 Data Analysis and Interpretation

In this section, I will explain how to organize and analyze the data collected during the study, focusing on the observations and feedback from users of the smart parking management system.

Data Organization

a. Categorize Data: The data is organized based on user types (Super Admins, Parking Managers, Normal Users) and based on the Observation Checklist's specific points.
b. Data Compilation: The summary outlines common issues, user feedback, and system performance, while the feedback collection process identifies common concerns and issues.

Data Analysis

a. Quantitative Analysis: Frequency analysis involves counting specific issues or feedback points, while statistical measures analyze patterns in data like response times or error rates.
b. Qualitative Analysis: Identify repeated suggestions in feedback, such as system improvements, and transform them into clear, actionable ideas.

Interpretation of Results

a. Identify Key Issues: The system should address major issues identified through observations and feedback, such as user permission issues or lost item management, and evaluate how well it meets user needs and expectations.
b. Recommend Improvements: The analysis will suggest improvements to the system, such as enhancing the user interface or introducing new features to show common issues.
c. Compare Against Objectives: The objective alignment of the findings with the project's initial objectives is crucial to ensure the system effectively addresses the intended problems.

For instance, to understand user experiences and system performance, we used simple statistical methods. For example, we measured the average time it took for users to complete a parking reservation on our website. We collected data from 10 users with the following times (in minutes): 2, 3, 4, 3, 2, 4, 5, 3, 2, and 4. To find the average (mean) response time before improvement, we used the formula:

$$Mean = \frac{Sum\ of\ all\ response\ times}{number\ of\ users}$$

$$Mean = \frac{3 + 4 + 3 + 2 + 4 + 5 + 3 + 2 + 4}{10} = 3.2$$

So, the average time it took users to complete reservation was 3.2 minutes before improvement. This led me to think that I had to use good database system able to handle multiple tasks at a time, for that I migrate to a new one and the average time response went down to about 1minute.

We also tracked errors during the reservation process. Out of the 15 users, 1 encountered an issue where their report of lost item didn't save correctly. We calculated the error rate using the formula:

$$Error\ rate = \left(\frac{number\ of\ errors}{total\ users}\right) x\ 100$$

$$Error\ rate = \frac{1}{15} x\ 100 = 6.67\%$$

This means 6.67% of users experienced an error.

We also analyzed the types of issues users reported while using the platform. We categorized the feedback into four main issues: Navigation Difficulties, Slow Loading Times, Account Login Issues, and User Help Problems. Here are the counts for each issue type from 20 users:

Navigation Difficulties: 4 users, Slow Loading Times: 3 users, Account Login Issues: 2 users, User Help Problems: 1 user

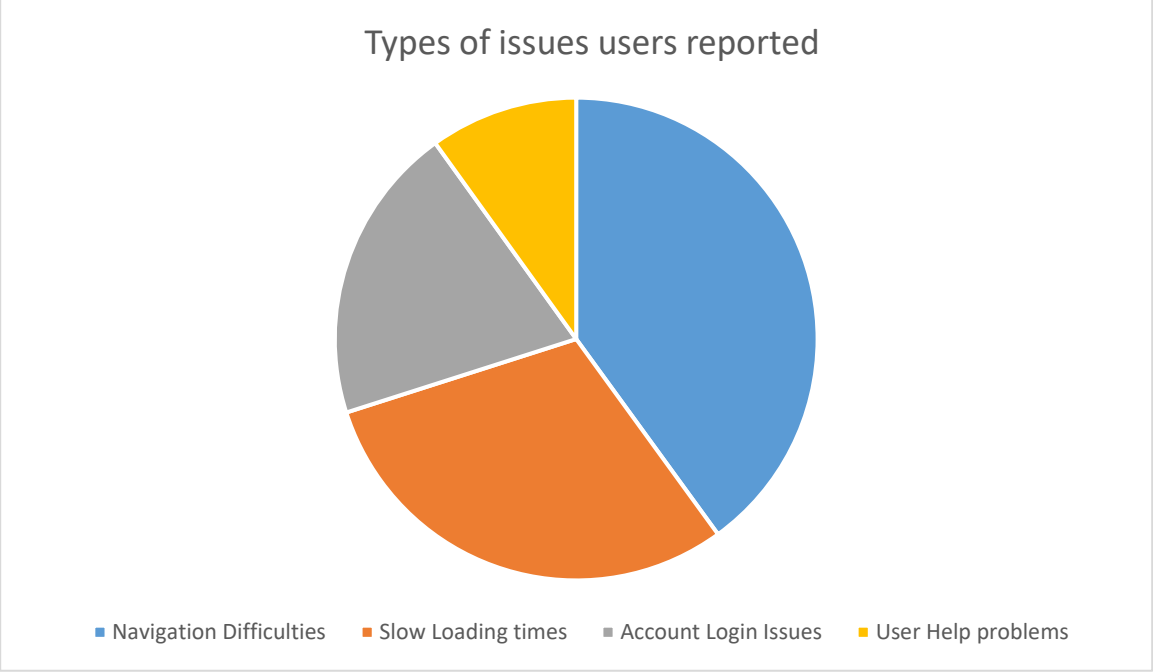


Figure 6 Types of issues users reported

3.7 Ethical Considerations

For this study, I took the following ethical measures:

a. Informed Consent: All participants were informed about the study's purpose and their right to withdraw. They provided consent before participating.
b. Anonymity and Confidentiality: Personal information was kept confidential and anonymized to protect participants' identities.
c. Data Security: All collected data was securely stored and accessible only to authorized personnel.
d. No Harm Principle: The study was designed to prevent any physical, psychological, or social harm to participants.

3.8 Limitations of the Study

The study may face the following limitations:

a. Sample Size: Limited to a small group of participants, which may not fully represent the broader population.
b. Technical Issues: Potential technical problems with the website or sensors that could affect data collection.
c. User Bias: Participant feedback might not cover all potential user experiences.

CHAPTER 4: SYSTEM DESIGN ANALYSIS AND IMPLEMENTATION

4.0 Introduction

This chapter focuses on the development of the smart parking management system. It details the design, analysis, and implementation stages, providing plan and strategy for how the system addresses common parking issues. We will cover the architecture of the system, the design of the database, and the implementation of features such as real-time parking availability, reservations, and lost-and-found management.

4.1 Design

After building the web app to manage parking data and display availability in real time, I needed to create two sample parking lots to demonstrate the system's capabilities. The idea was to show that multiple parking lots, each with different configurations (number of slots, microcontroller types, etc.), could independently send their respective data to the web application without any conflicts or issues. I designed two parking simulations:

- **Parking A**: 48 parking slots.
- **Parking B**: 24 parking slots.

Challenges and Questions:

At the start, I had several questions:

a. Can a single microcontroller handle 48 sensors (one per parking slot)?
b. What types of sensors and microcontrollers would work best?
c. How can both parking lots communicate with the web application independently?

Choosing Microcontrollers

Parking A (48 slots) – ESP32

I chose the **ESP32** as the microcontroller for Parking A due to several advantages:

a. Built-in Wi-Fi: The ESP32 has an integrated Wi-Fi module, enabling direct communication with the web application.
b. Low Power Consumption**:** With multiple low-power modes, the ESP32 is highly efficient, consuming minimal power when idle.

c. Cost Efficiency: ESP32 is cost-effective for a project requiring multiple sensor connections.

Handling 48 Sensors with ESP32:

The ESP32 has several analog pins and digital pins. However, there are limitations with using analog pins over Wi-Fi: Only 6 analog pins can be used when Wi-Fi is enabled on the ESP32, and not all pins can read sensor data while handling Wi-Fi communication.

To handle more than 6 sensors, I decided to use a **multiplexer (CD74HC4067)**. This component allows one analog pin to read signals from multiple sensors by using 4 digital pins as switches: The multiplexer takes 16 analog inputs and feeds them through a single output pin (in this case, one of the 6 available analog pins on the ESP32). The 4 digital pins on the ESP32 act as control switches to select which sensor's data to read.

Thus, with 6 analog pins and 1 multiplexer, I could handle 21 sensors (6 direct + 16 through the multiplexer).

Remaining 27 Sensors**:** To cover the remaining sensors, I would need a second multiplexer to handle another set of 16 sensors. The ESP32's digital pins could handle the switching for both multiplexers.

Parking B (24 slots) – NodeMCU ESP8266:

For Parking B, I chose the **NodeMCU ESP8266**. This choice was made to show the flexibility of using different Wi-Fi-enabled microcontrollers. While the NodeMCU is similar to the ESP32 in terms of Wi-Fi connectivity, it has limitations: **Only 1 analog pin (A0)**: This presented a challenge, as I needed to connect 16 sensors. To solve this, I again used a **multiplexer (CD74HC4067)** to connect **16 sensors** through the single analog pin. The NodeMCU's **4 digital pins** controlled the switching for the multiplexer.

Sensors:

Though parking management systems use advanced sensors (such as with esp32-camera or advanced ultrasonic), they are expensive and unnecessary for a prototype. Instead, I opted for **LDR (Light Dependent Resistor) sensors** for each parking slot in both parking lots. The LDR sensors detect changes in light, indicating whether a slot is occupied or vacant.

Justification for LDR Sensors:

a. **Cost-Effective**: LDR sensors are inexpensive (400 Rwf each) and easy to integrate into the system.
b. **Size**: Their small size makes them ideal for a prototype.
c. **Simplicity**: LDR sensors are straightforward to use with the ESP32 and NodeMCU, requiring only a resistor to form a voltage divider circuit that converts light intensity into a measurable signal.

Resistor Selection:

Each sensor requires a **pull-down resistor** in a voltage divider circuit. This will allow finding the threshold value below or above which we will decide whether a place is occupied or not.

For ESP32 (Parking A):

a. I used a **10kΩ resistor** for the ESP32. This value worked very well during testing, providing stable sensor readings when connected to the ESP32's analog pins.
b. The resistor is part of the voltage divider circuit, with the LDR on one side and the resistor on the other. When light levels change, the resistance of the LDR changes, altering the voltage across the resistor, which is then measured by the microcontroller.

$V_{Out} = V_{In} \times \frac{R_{Fixed}}{R_{Fixed}+R_{LDR}}$ (formula of the voltage divider)

c. **Unoccupied Spot (bright light):**

LDR resistance (R_{LDR}) = 10kΩ

For the ESP32 to detect the unoccupied state, $\mathbf{V_{Out}}$ should be below a certain threshold. $V_{In}(voltage\ from\ ESP32)$.

$\mathbf{V_{Out}} = 3.3 \times \frac{10K\Omega}{10K\Omega+10K\Omega} \approx 1.65V$

Unlike Arduino, that is 10-bit resolution for ADC values, ESP32 reads ADC values up to 4096 (it is 12-bit resolution) so

ADC value= $4096 \times \frac{1.65}{3.3} = 2048$

d. **Occupied Spot (dark, vehicle present):**

R_{LDR} = 1MΩ

Check if V_{Out} is lower the threshold(seuil=2000) to confirm detection:

$$V_{Out} = 3.3 \times \frac{10\text{kΩ}}{10\text{kΩ} + 1\text{MΩ}} \approx 0.0326V$$

ADC value= $4096 \times \frac{0.0326}{3.3} \approx 41$

This is to say that for ESP32, I was expecting any read ADCvalue above 2000 to show unoccupied slot while the ADC values less than 2000 indicate an occupied slot. Hence I fixed 2000 to be my threshold (seuil in French) value.

For NodeMCU (Parking B):

a. I used a **4.7kΩ resistor** with the NodeMCU. The ESP8266's analog pin works optimally with lower resistance values, and the 4.7kΩ resistor produced reliable readings without noise.
b. The same calculations but knowing that NodeMCU reads up to 1024(10-bit resolution)
c. **Unoccupied Spot (bright light):**

LDR resistance (R_{LDR}) = 10kΩ

For the ESP8266(NodeMCU) to detect the unoccupied state, $\mathbf{V_{Out}}$ should be below a certain threshold.
$V_{In}(voltage\ from\ ESP32)$

$$\mathbf{V_{Out}} = 3.3 \times \frac{4.7K\Omega}{4.7K\Omega + 10K\Omega} \approx 1.055\text{V}$$

like Arduino,that is 10-bit resolution for ADC values,ESP8266 reads ADC values up to 1024 so

ADC value= $1024 \times \frac{1.055}{3.3} = 327$

d. **Occupied Spot (dark, vehicle present):**

R_{LDR} = 1MΩ

Check if V_{Out} is lower the threshold(seuil=300) to confirm detection:

$$V_{Out} = 3.3 \times \frac{4.7\text{k}\Omega}{4.7\text{k}\Omega + 1\text{M}\Omega} \approx 0.01543V$$

ADC value= $1024 \times \frac{0.01543}{3.3} \approx 4.7$

This is to say that for ESP8266, I was expecting any read ADCvalue above 300 to show unoccupied slot while the ADC values less than 300 indicate an occupied slot. Hence I fixed 2000 to be my threshold (seuil in French) value.

Wiring and Connections:

For both parking prototypes, I used **thin-gauge wiring** (small diameter) to keep the circuit organized and to reduce resistance losses over short distances. This is particularly important when connecting multiple sensors to the microcontroller. Using PCBs and wires made it easy to connect the sensors to the ESP32 and NodeMCU with soldering.

Power Supply:

Both the ESP32 and NodeMCU operate on **5V power**. However, to simplify the design and reduce the need for multiple power sources, I used a **single 5V power supply** for each microcontrollerThis setup was also convenient for powering the LDR sensors, which require very low current.

Measuring Power Consumption:

a. **ESP32**: The ESP32 has a power consumption of about **160-240 mA** when using Wi-Fi. Each LDR sensor consumes only a few microamps, so the total power draw remains well within the capacity of a 5V USB power supply: 0.8 to 1.2W.
b. **NodeMCU**: The NodeMCU consumes about **70-200 mA** during operation, which is also compatible with a standard USB power source: 0.35 to 1W.

4.2 Drawings

4.2.1 System Architecture Diagram

Description:

The smart parking management system uses an ESP32 microcontroller as the central unit, detecting vehicles and sending data to the microcontroller. The data is processed by the server, which hosts a web interface and database. The system integrates hardware components, server infrastructure, database management, and user interface for efficient and reliable operation.

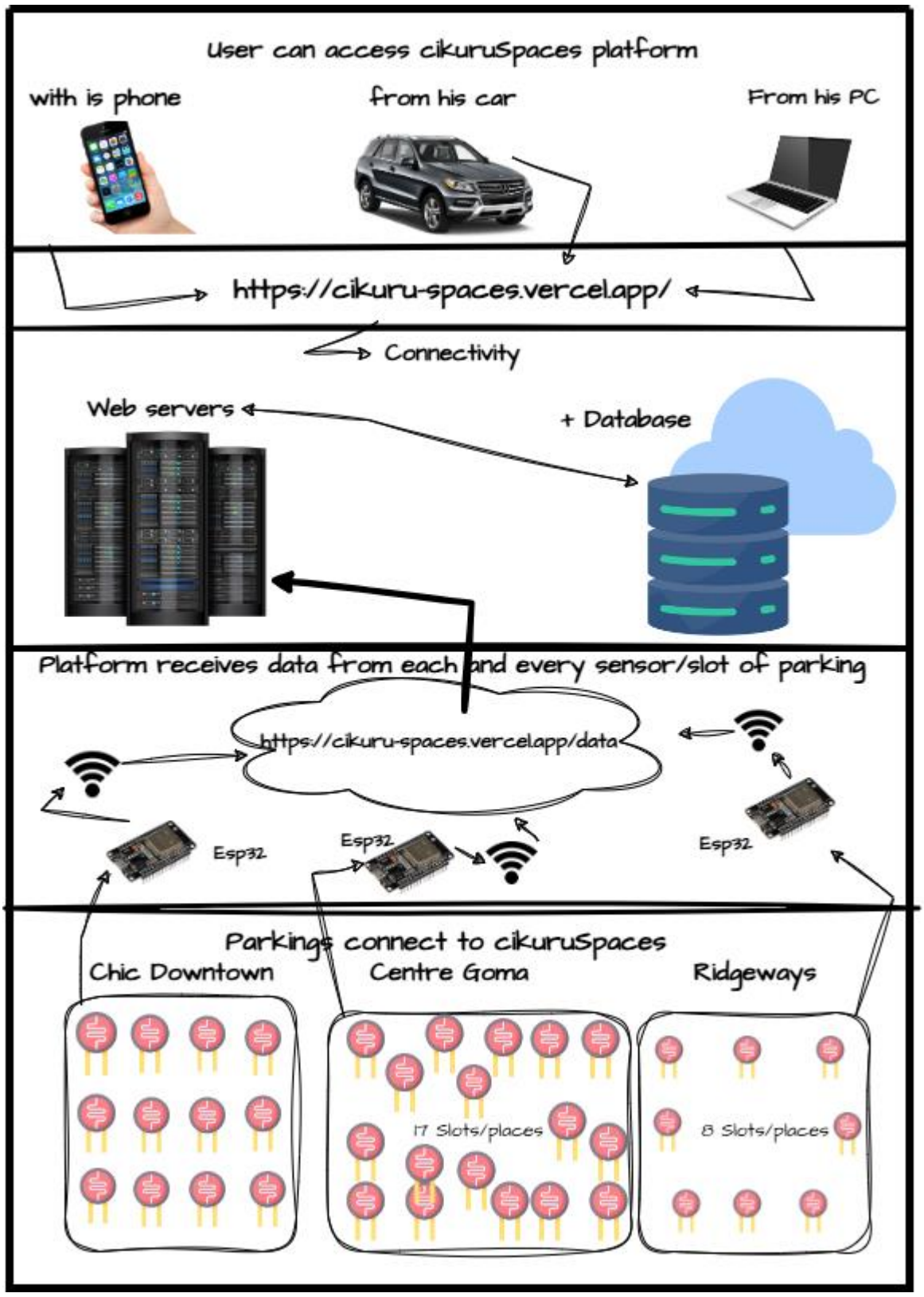


Figure 7 System Architecture Diagram

4.2.2 Circuit diagrams

Description:

Circuit diagrams provide a clear view of the electrical connection within a system. It shows how each LDR sensor connects to the ESP32 microcontroller. ESP32 features several GPIO pins, used to read data from sensors. The necessary resistors are included in series with the LDRs to form a voltage divider to ensure compatible signal levels with the ESP32 analog inputs. Power supply connections also show how the system gets powered and which parts are grounded. This setup is quite critical in order to have proper detection of whether a parking spot is occupied or not, since the ESP32 reads voltage changes from the sensors, interpreting them for the detection of the presence of a vehicle. Both diagrams are representative of different parking areas with the respective sets of sensors and ESP32 connections that ensure coverage for an area and the accuracy of monitoring.

Coming to the prototype, here are two different circuit diagrams for a parking lot with its sensors corresponding to the slots.

So, Fig. 8 will show the scheme of parking A - here an example of Chic Downtown parking, and Fig. 9 the scheme of parking B - here an example of Centre Commercial de Goma, just as examples not the real ones.

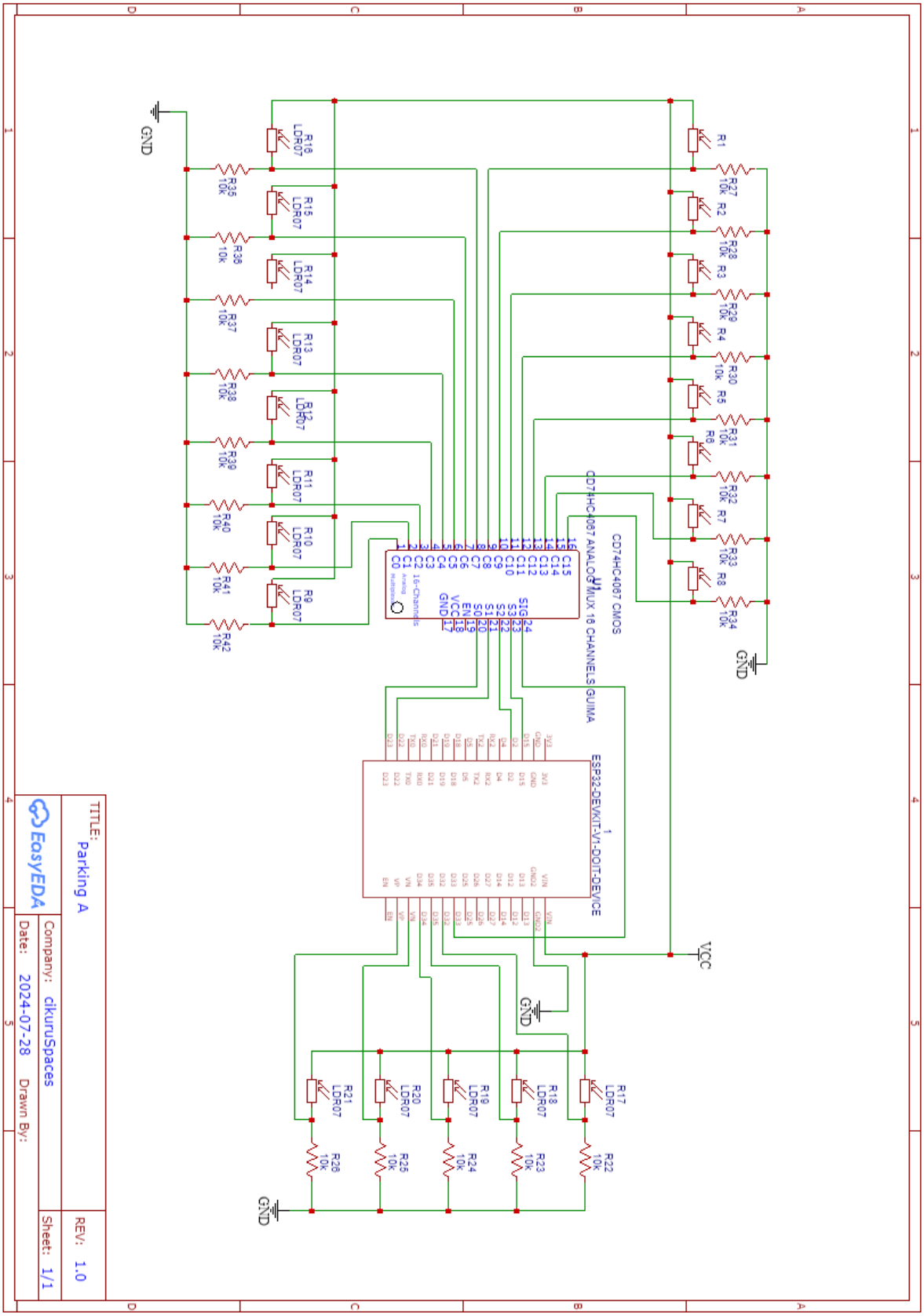


Figure 8 Circuit Diagram Parking A

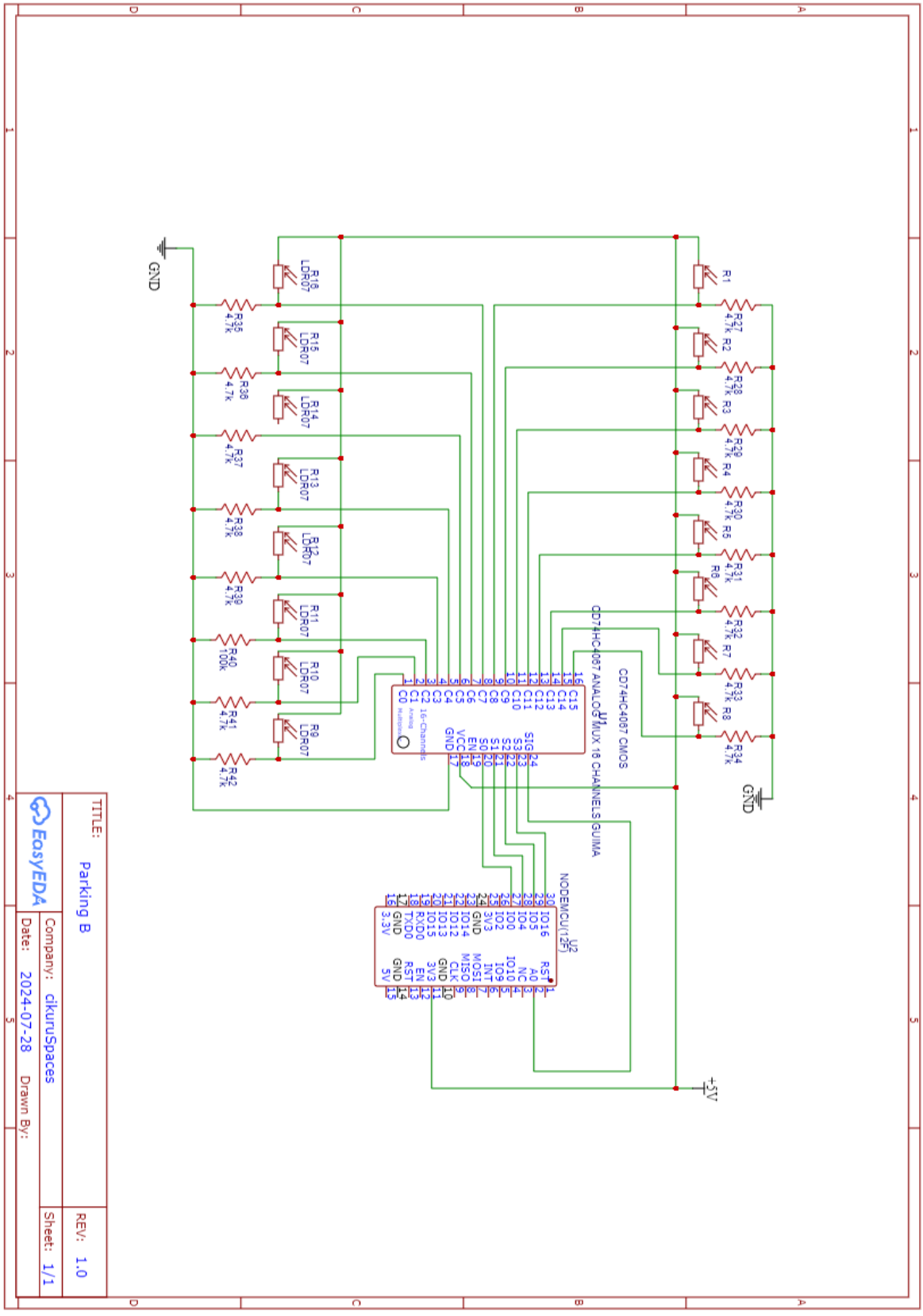


Figure 9 Circuit Diagram Parking B

4.2.3 Database Design

The database tables, which store information such as user data, parking availability, and system logs, will be hosted on a server.Refer to Fig. 10. The ESP32 microcontroller communicates with the server to update and retrieve data as needed. The server-side application makes database interactions, processing data from the ESP32 and serving it to the web interface for users.

In the System Architecture Diagram in Fig. 7, the server and database were typically represented as part of the backend infrastructure. The diagram showed the ESP32 sending data to the server, which then updates the database. The server also retrieves data from the database when needed, such as displaying available parking spots on the web interface, making reservations or adding found items.

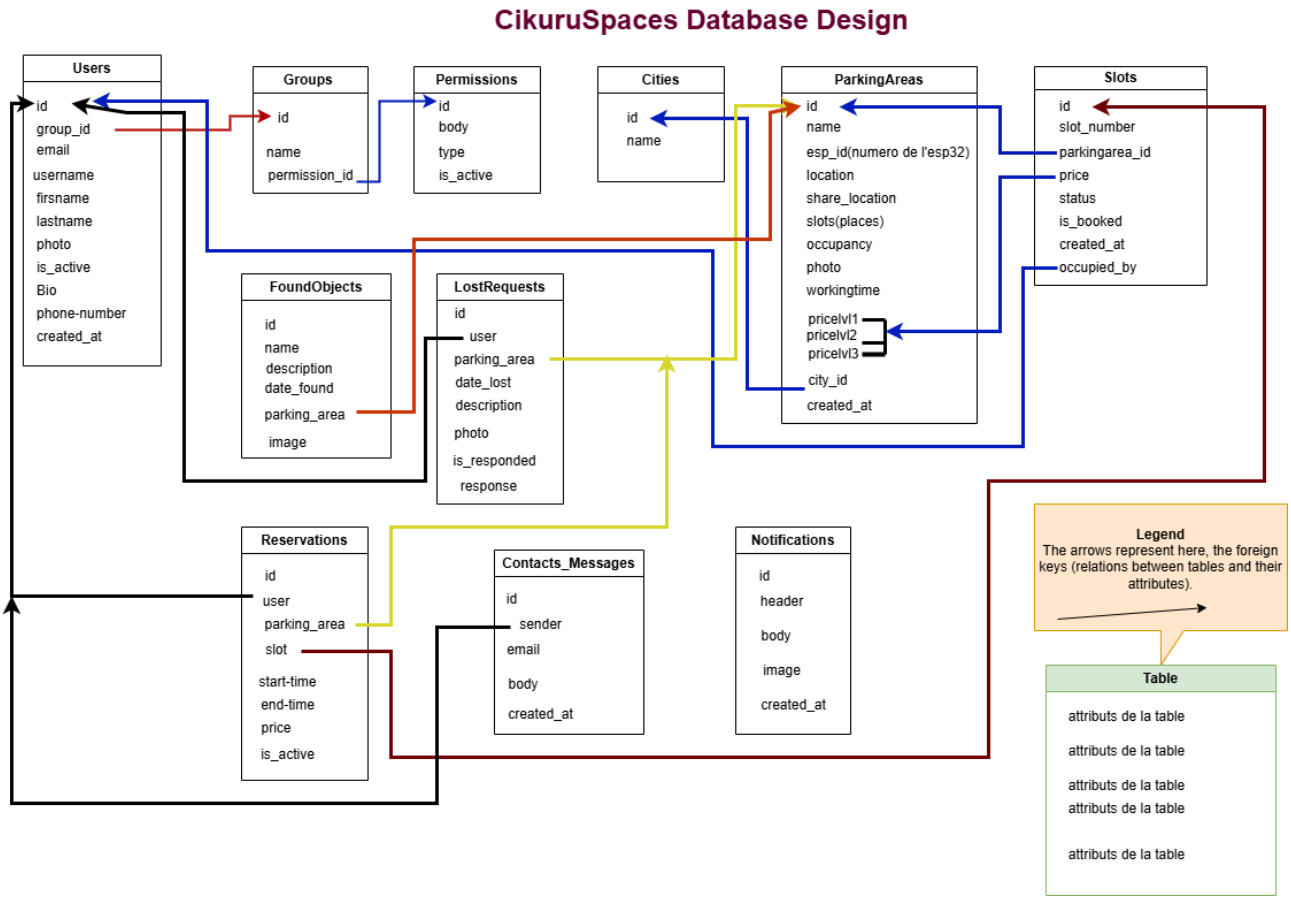
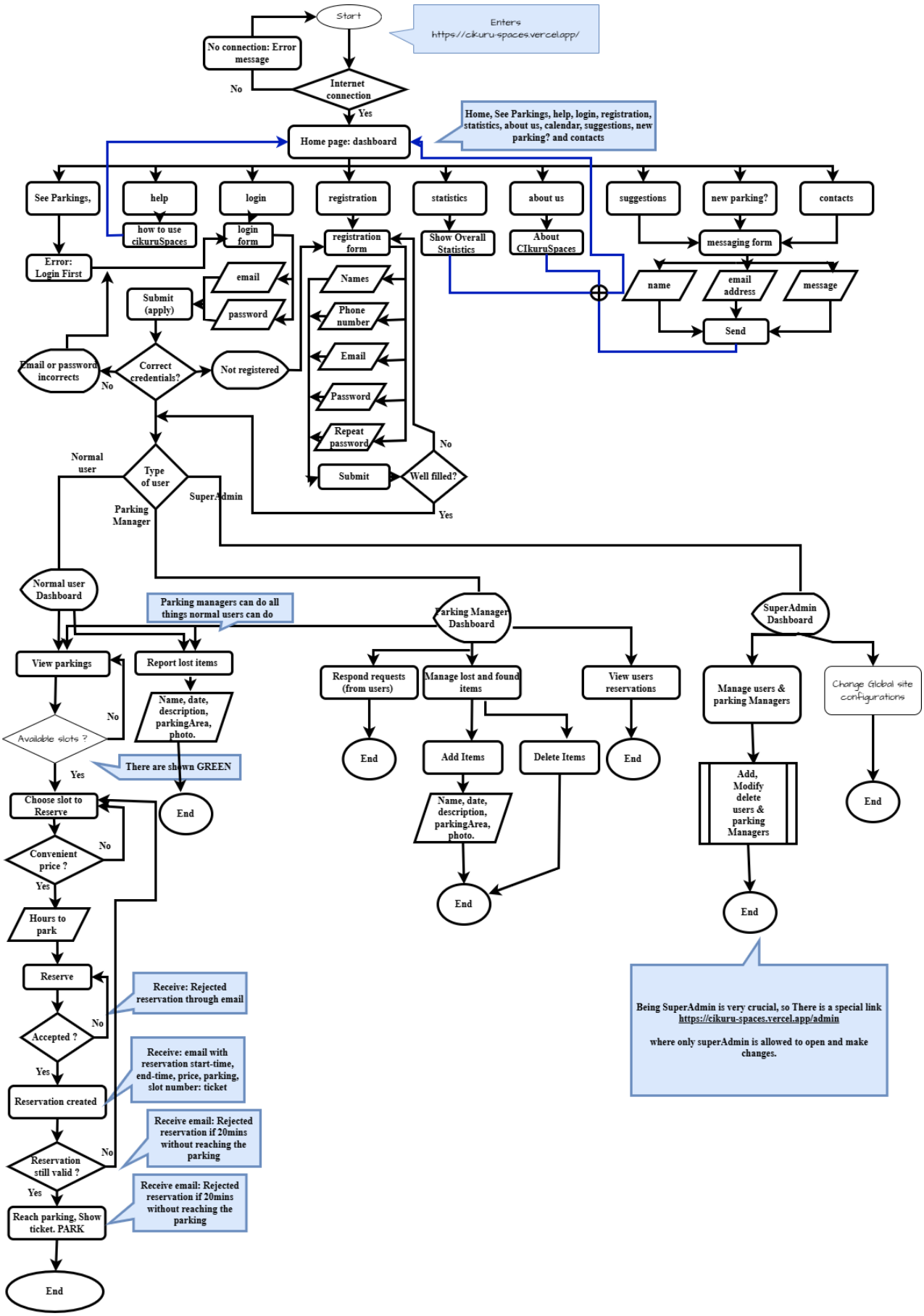


Figure 10 Database Design

4.2.3 Flowchart of operation

Figure 11 Flowchart of operations


Start
Enters https://cikuru-spaces.vercel.app/
No connection: Error message
No
Internet connection
Yes
Home page: dashboard
Home, See Parkings, help, login, registration, statistics, about us, calendar, suggestions, new parking? and contacts
See Parkings,
help
login
registration
statistics
about us
suggestions
new parking?
contacts
how to use cikuruSpaces
login form
registration form
Show Overall Statistics
About CIkuruSpaces
messaging form
Error: Login First
email
password
Names
Phone number
Email
Password
Repeat password
name
email address
message
Submit (apply)
Submit
Send
Email or password incorrects
No
Correct credentials?
Not registered
No
Well filled?
Yes
Normal user
Type of user
Super.Admin
Parking Manager
Normal user Dashboard
Parking managers can do all things normal users can do
Parking Manager Dashboard
SuperAdmin Dashboard
View parkings
Report lost items
Respond requests (from users)
Manage lost and found items
View users reservations
Manage users & parking Managers
Change Global site configurations
Available slots ?
No
Name, date, description, parkingArea, photo.
End
Add Items
Delete Items
End
Yes
There are shown GREEN
Choose slot to Reserve
Convenient price ?
No
Yes
Name, date, description, parkingArea, photo.
End
Add, Modify delete users & parking Managers
End
Hours to park
Reserve
Receive: Rejected reservation through email
Accepted ?
No
Yes
Receive: email with reservation start-time, end-time, price, parking, slot number: ticket
Reservation created
Receive email: Rejected reservation if 20mins without reaching the parking
Reservation still valid ?
No
Yes
Receive email: Rejected reservation if 20mins without reaching the parking
Reach parking, Show ticket. PARK
End
Being SuperAdmin is very crucial, so There is a special link https://cikuru-spaces.vercel.app/admin
where only superAdmin is allowed to open and make changes.

The diagram Fig 11 is to allow understanding each possible operation that can take place on this system, starting from authentication of users, view of real-time availability of parking slots and son on.

4.3 Specifications

4.3.1 Hardware Specifications

Specification key	Specification value
Microcontrollers	
Model 1	**ESP32**
Processor	32-bit dual-core
Clock Speed	Up to 240 MHz
Memory	520 KB SRAM
Connectivity	Wi-Fi, Bluetooth
GPIO Pins	34, with multiple analog and digital inputs (ESP-32 WiFi, n.d.)
Model 2	ESP8266 NodeMCuU
Processor	Tensilica L106 32-bit RISC processor
Clock Speed	Up to 80 MHz
Memory	64 KB (for data and instructions)SRAM
Connectivity	Wi-Fi
GPIO Pins	30, with multiple analog and digital inputs
Sensors	
Type	Light Dependent Resistors (LDR)
Resistance Range	1kΩ to 1MΩ (depending on light levels)
Sensitivity	Detects changes in ambient light, indicating the presence or absence of a vehicle. (Photoresistor, n.d.) Note: LDR sensors are used for presentation purposes in this prototype. New, more accurate parking sensors are available on the market for actual implementation.
Resistors	
Value	100 and 4.7kΩ fixed resistors, used in conjunction with LDRs to create a voltage divider circuit. (Voltage divider, n.d.)
Power Supply	
Input Voltage	5V (regulated to 3.3V for ESP32 and sensors)
Power Consumption	Approximately 0.5W per module, including sensors and microcontroller
Additional Components	
Wiring and Connectors	For connecting sensors, power, and communication lines

Table 1 Hardware Specifications

4.3.2 Software and environmental specifications, tools and platforms used

Used	Specifications
Firmware	
Programming Language	C/C++ (Arduino IDE) (Arduino, n.d.)
Development Environment	Visual Studio Code
Functionality	Read LDR sensor values through ADC Process data to determine parking spot status Communicate with the web server for data logging and user notifications
Web Interface	
Backend Framework	(The web framework for perfectionists with deadlines, n.d.)
Frontend Framework	HTML, CSS, JavaScript (Chart.js for data visualization) (Learn Web development, n.d.)
Hosting	Vercel (Vercel Hosting, n.d.)
Database	PostgreSQL, hosted on Aiven (Aiven-Your trusted Data % AI Platform, n.d.)
Media Storage	Cloudinary, for storing and serving images
Features	Real-time display of parking spot availability. User registration and authentication.
Environmental Specifications	
Operating Temperature Range	Microcontroller and Sensors: -40°C to 85°C Power Supply: 0°C to 50°C
Tools and Platforms Used	
Circuit Design:	EasyEDA
Database Design:	draw.io
Web Hosting and Deployment	Vercel
Media Management	Cloudinary
Database Management	Aiven (PostgreSQL)

Table 2 Software, tools and platforms used

4.4 Cost estimation

This includes hardware, software, and operational costs. The costs are estimated based on current market prices and are subject to change.

4.4.1 Hardware Costs

Material	Details	Quantity	Unity price (Rwf)	Total (Rwf)
Microcontroller ESP32	For parking A	1	15,500	15,500
Microcontroller ESP8266 or NodeMCU	For parking B	1	11,500	11,500
LDR Sensors		37	400	14,800
Resistors (10kΩ)		40	100	4,000
Power Supply		1	4,000	4,000
Wiring and Connectors	Estimated total			8,000
Diverse Components (e.g., PCBs,cables, enclosures):	Estimated total			20,000
Total hardware cost				77,800

Table 3 Hardware costs

4.4.2 Software and Development Costs

Tools	Price (Rwf)
Development Tools	
Arduino IDE	Free
Visual Studio Code	Free
PlatformIO	Free
Web Development	
Django Framework	Free
Chart.js (for data visualization)	Free
Hosting and Storage	
Vercel (Web Hosting)	Free for basic usage, potential upgrade cost if usage exceeds free tier
Cloudinary (Media Storage)	Free for basic usage, potential upgrade cost if storage needs exceed free tier
Aiven (PostgreSQL Database)	Basic Plan: Free (actual cost may vary based on usage)
Design Tools	
EasyEDA (Circuit Design)	Free
draw.io (Database Design)	Free
Estimated Monthly Operational Cost	0 (but potential hosting/storage upgrades cost)
Additional costs(for possible future use)	
Labor (Development and Testing):	• Estimated Hours: 100 hours • Hourly Rate: 10,000 (for example) • Total: 1,000,000
Testing and Prototyping	Materials and Miscellaneous Expenses: 50,000
Total Additional Costs: 1,050,000	

Table 4 Software and development costs

Note: These estimates are based on current market prices and available free tools. All costs can vary depending on project needs, component prices, and other unforeseen expenses. Consequently, this budget in no way contemplates any kind of upgrade or scaling higher than this prototype. An example of this would be Table 5.

Category	Details
Cost for Presentation Prototype	
Components	ESP32 microcontrollers, LDR sensors, resistors, prototyping boards, jumper wires, power supplies, etc.
Software Tools	Arduino IDE, Visual Studio Code, EasyEDA, draw.io
Hosting and Media	Vercel for web hosting, Cloudinary for media storage, Aiven for PostgreSQL database
Total Estimated Cost	85,000 Rwf
Future Implementation Cost	
Upgraded Components	Advanced parking sensors, commercial-grade ESP32 boards, additional hardware for enhanced functionality (e.g., cameras, more durable sensors)
Software Development	Additional features and security enhancements, integration with existing infrastructure
Hosting and Scalability	More robust hosting solutions, additional storage, and bandwidth requirements
Maintenance and Support	Ongoing maintenance, updates, and user support
Total Estimated Cost	1,000,000 - 10,000,000 Rwf (depending on scale and additional features)
Cost Per Parking Space	
Components for Each Space	Sensor, necessary wiring, connection to the central system
Installation and Setup	Labor costs for installation and setup of sensors and associated hardware
Total Estimated Cost Per Parking Space	50,000 Rwf – 100,000 Rwf
User Pricing	
Estimated User Cost	Based on the exact amount parked

Table 5 Summarized cost estimations

4.5 Implementation

Realization involves implementing the planned system to ensure its functionality. Due to limited space, crucial parts of implementation are included in Appendix 2. For more details and source code, visit the GitHub repository at https://github.com/go-arnold/matabaSpaces/.

The implementation steps for the smart parking management system are detailed below:

4.5.1 System Setup and Configuration

a. **Hardware Setup:**

1. **ESP32 and ESP8266 Microcontrollers:** Configure microcontrollers with firmware, connect them to LDR sensors and peripheral devices, ensure proper power supply, and secure connections to prevent disconnections or shorts.
2. **LDR Sensors:** Position LDR sensors in designated parking spots in the prototype setup and calibrate them to accurately detect light intensity changes, indicating occupied spots with good resistors.
3. **Networking:** The ESP32 is configured to communicate with a web server via Wi-Fi and send sensor data via REST API endpoints.

b. **Software Development:**

1. **Arduino IDE & PlatformIO:** The ESP32 firmware was developed and uploaded using Arduino IDE .
2. **Web Development (Django Framework):** The backend system, including models, views, and controllers, was developed using Django while the frontend interface will be developed using HTML, CSS, JavaScript, and Chart.js.

c. **Database Setup (Aiven PostgreSQL):** Created tables and schemas for parking data, user information, and logs on Aiven's PostgreSQL database.

d. **Media Management (Cloudinary):** Cloudinary is set up for storing media files like parking spot images,lost items images and user avatars.

e. **Web Hosting and Domain and SSL(Secure Sockect Layer) on Vercel:** The Django application was deployed on Vercel for hosting both frontend and backend services.

4.5.2 Final Deployment and Launch

The final version of the application was deployed on Vercel. User onboarding is provided with documentation and training. Post-launch monitoring and maintenance include performance monitoring, user feedback, and system enhancements.

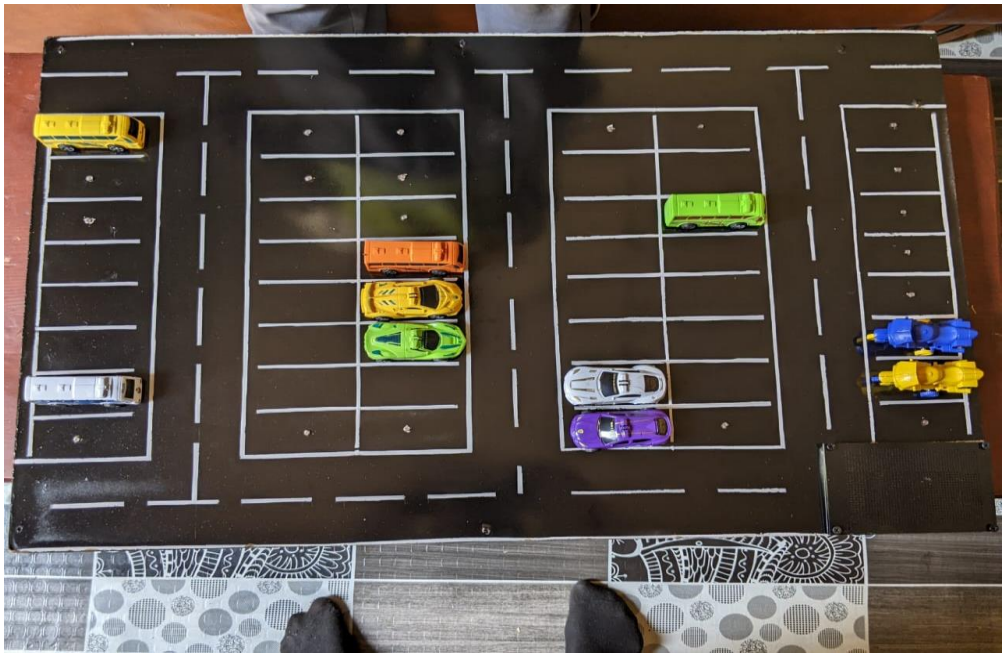


Figure 12 Parking A implemented

Figure 13 Implemented Parking A and Parking B

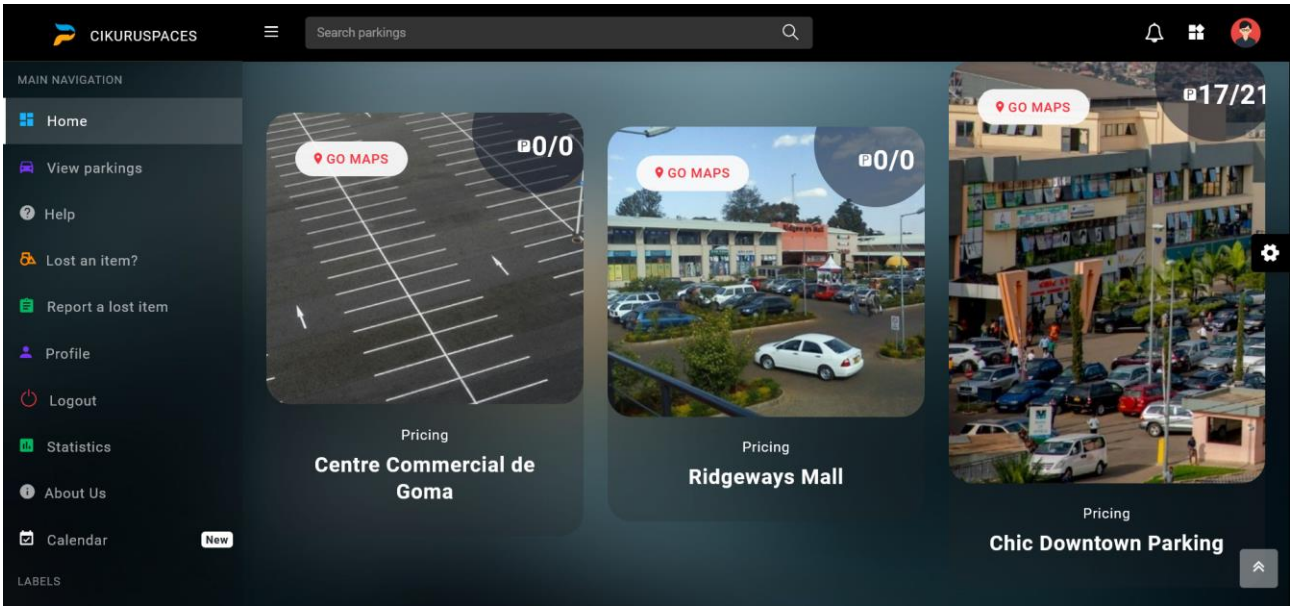


Figure 14 View of Available parkings

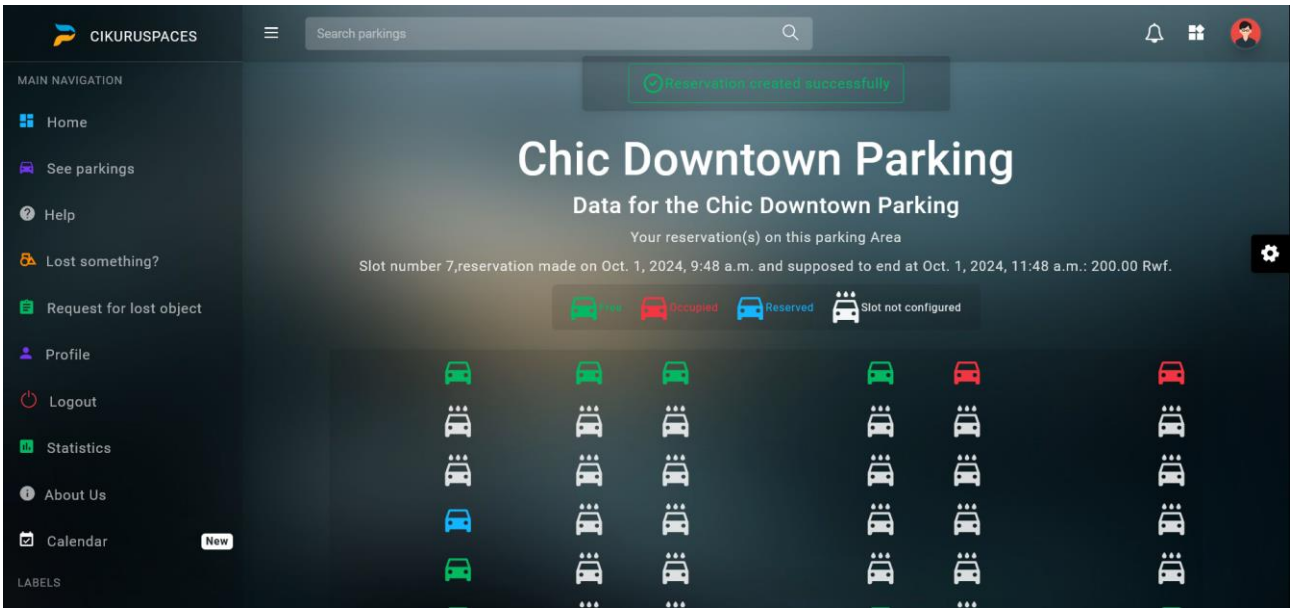


Figure 15 Real-time availabilityof slots

CHAPTER 5: CONCLUSION AND RECOMMENDATIONS

5.0 Introduction

This chapter summarizes the findings of the study, interprets them in the context of the research objectives, and makes recommendations for further studies or practical uses. Finally, it identifies avenues for further research to be conducted based on the gains from this study.

5.1 Conclusions

The main objective of this study was to design and develop a prototype of an intelligent parking management system, incorporating real-time parking availability updates and reservation capabilities. This objective was guided by best practices identified in existing literature (Barriga, et al., 2019).

Key conclusions drawn from the study include:

a. Limitations of Traditional Systems: Traditional parking management systems have significant limitations, such as inefficient space utilization and poor user experience. These limitations were thoroughly analyzed and confirmed (Biyik, et al., 2021).
b. Integration of Django and IoT: The potential of integrating Django with IoT principles to develop a smart parking management system was explored. The study found that using ESP32 microcontrollers, cloud services (Vercel, Cloudinary, Aiven), and real-time data handling provided a robust framework for developing such systems (Raj & Shetty, 2024).
c. Database Design: A comprehensive database was designed to store all information and media files related to the platform. This database supports efficient data management and retrieval, crucial for the system's functionality.

These findings validate the effectiveness of the proposed smart parking management system and address the specific research objectives.

5.2 Recommendations

Based on the conclusions, the following recommendations are proposed:

a. Sensor Upgrades: Future implementations should consider using advanced sensors for higher accuracy and reliability in detecting vehicle presence.

b. Security Enhancements: Strengthen security measures to protect user data and prevent unauthorized access, ensuring user trust and system integrity.
c. User Feedback: Regularly update the system based on user feedback to continuously improve user experience and add new features.
d. Scalability Planning: Plan for future scalability to support a larger user base and additional functionalities, ensuring the system can grow with increasing demand.

5.3 Suggestions for Further Study

To further enhance the project, the following areas could be explored:

a. Advanced Sensor Technology: Investigate other sensor technologies, such as ultrasonic or infrared sensors, for more accurate vehicle detection.
b. Data Analytics: Explore advanced data analytics to predict parking availability trends and optimize parking space allocation.
c. Mobile Integration: Develop a dedicated mobile application to provide a more seamless user experience and additional features like navigation to available parking spots.
d. Environmental Impact: Study the environmental benefits of smart parking systems, such as reduced emissions from vehicles searching for parking.

By addressing these areas, future research can build on the foundation laid by this study and further advance the development of intelligent parking management systems.

REFERENCES

Abrar Fahim, Hasan, M., & Chowdhury, M. A. (2021). *Smart parking systems: comprehensive review based on various aspects.*

Aiven-Your trusted Data % AI Platform. (n.d.). Retrieved from Aiven: https://aiven.io/

Arduino. (n.d.). Retrieved from Arduino: https://www.arduino.cc/

Barriga, J. J., Sulca, J., León, J. L., Ulloa, A., Portero, D., Andrade, O. R., & Yoo, G. S. (2019). *Smart Parking: A Literature Review from the Technological Perspective.*

Biyik, C., Zaheer , A., Gabriele , P., David, M., Muftah , O., Eoin , O., . . . Muhammad, K. (2021). *Smart Parking Systems: Reviewing the Literature, Architecture and Ways Forward.*

ESP-32 WiFi. (n.d.). Retrieved from Espressif: https://www.espressif.com/en/products/socs/esp32

Image and Video Upload, Storage... (n.d.). Retrieved from Cloudinary: https://cloudinary.com

Jemmali, M., Melhim, L. K., T. Alharbi, M., Bajahzar , A., & Omri , M. N. (2022). *Smart-parking management algorithms in smart city.*

Learn Web development. (n.d.). Retrieved from mdn: https://developer.mozilla.org/en-US/docs/Learn

Lumbanga, P. (2021). *Research on smart parking Solution*. Retrieved from Academia: https://www.academia.edu/60879497/

Photoresistor. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Photoresistor#:~:text=A%20photoresistor%20%28also%20known%20as%20a%20light-dependent%20resistor%2C,sensitive%20surface%2C%20in%20other%20words%2C%20it%20exhibits%20photoconductivity.

Raj, A., & Shetty, S. (2024). Smart parking systems technologies, tools, and challenges for implementing in a smart city environment: a survey based on IoT & ML perspective. *International Journal of Machine Learning and Cybernetics*.

Technology Acceptance Model. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Technology_acceptance_model

The web framework for perfectionists with deadlines. (n.d.). Retrieved from Djangoproject: https://www.djangoproject.com/

The web framework for perfectionists with deadlines. (n.d.). Retrieved from Djangoproject: https://www.djangoproject.com/

Vercel Hosting. (n.d.). Retrieved from Vercel: https://vercel.com/

Voltage divider. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Voltage_divider

What is SSL , TLS and HTTPS ? (n.d.). Retrieved from digicert: https://www.digicert.com/what-is-ssl-tls-and-https

APPENDICES

APPENDIX 1: OBSERVATION CHECKLIST

Observation Checklist for SuperAdmins	
General Information	
Date of Observation:	
Time of Observation:	
Observer's Name:	
Role of Participant:	
To check	
System Login	Successfully logs into the system.
User Management	Adds or removes users (Super Admins, Parking Managers).
Permission Settings	Changes user roles and permissions.
System Monitoring	Views and interprets system performance reports.
Security Settings	Updates security settings and checks system logs.
Data Backup	Performs a backup of the system data.
Problem Resolution	Identifies and addresses system issues or bugs.
Feedback Handling	Reviews and responds to feedback from Parking Managers or Normal Users.
System Updates	Applies and verifies system updates.
Access Logs	Reviews access logs for unauthorized activity.
Configuration Changes	Makes changes to system configurations and verifies they are applied.
Emergency Protocols	Tests and verifies emergency protocols and notifications.
Integration Checks	Ensures integrations with other systems are functioning correctly.
User Training	Provides or updates training materials for users.
Report Customization	Customizes and generates various reports as needed.

Table 6 Super Admins Observation Checklist

Observation Checklist for Parking Managers	
General Information	
Date of Observation:	
Time of Observation:	
Observer's Name:	
Role of Participant:	
To check	
Login Process	Successfully logs into the system.
Lost and Found Management	Adds or updates lost and found items.
User Interaction	Responds to inquiries from Normal Users.
Parking Space Management	Updates parking space availability.
Incident Reporting	Reports issues or incidents to Super Admins.
Reservation Handling	Manages parking reservations and cancellations.
Report Generation	Generates reports on parking usage and availability.
System Navigation	Efficiently navigates through system features and tools.
Fee Management	Updates and manages parking fees and billing.
Complaint Handling	Addresses and resolves complaints from Normal Users.
Maintenance Scheduling	Schedules and tracks maintenance tasks for parking facilities.
System Alerts	Responds to system alerts and notifications
Data Accuracy	Verifies the accuracy of data entered into the system.
System Testing	Tests new features or updates before they go live.
Compliance Checks	Ensures parking operations comply with local regulations.
Inventory Management	Manages inventory for parking-related equipment and supplies.
Training Materials	Updates and provides training materials for new Parking Managers.

Table 7 Observation Checklist for Parking Managers.

Observation Checklist for Normal users	
General information	
Date of Observation:	
Time of Observation:	
Observer's Name:	
Role of Participant:	
To check	
Account Creation	Creates and sets up a user account.
Parking Reservation	Successfully reserves a parking spot.
Parking Navigation	Uses toy cars to simulate parking spot location and accessibility.
Lost Item Reporting	Reports a lost item using the system.
Lost Item Search	Searches for a lost item within the system.
Feedback Submission	Provides feedback or suggests improvements.
Booking Cancellation	Cancels a parking reservation if needed.
System Usability	Finds the system easy to use and understand.
Error Reporting	Identifies and reports any errors encountered during use.
Overall Experience	Rates overall satisfaction with the system.
Parking Spot Availability	Checks real-time availability of parking spots.
Payment Processing	Completes payment for parking reservations.
Accessing Notifications	Receives and views notifications about parking status or updates.
Interaction with Staff	Interacts with Parking Managers or support staff if needed.
Usage of Mobile Features	Uses mobile features, if applicable, for booking or navigation.
System Accessibility	Accesses the system from different devices (desktop, mobile).
Account Settings	Updates personal account settings and preferences.
Security Features	Uses and evaluates security features such as login authentication.
Help and Support	Accesses and uses help and support features effectively.
Compliance with Rules	Adheres to system rules and guidelines for parking.
Additional Comments	

Table 8 Observation Checklist for normal users

APPENDIX 2: Detailed Implementation Process of the Smart Parking Management System

After completing the prototype, the next step was to fully implement the smart parking management system, which involved creating a superuser, setting up parkings and parking managers, and enabling various functionalities. Here's a detailed account of each step involved in the process:

Step 1: Creating the Superuser = SuperAdmin

To manage the overall system, a superuser needs to be created. This involves using Django's built-in user authentication system. The following command wass used in the terminal to create a superuser in the main directory of my project.

```
python manage.py createsuperuser
```

The superuser is granted access to all administrative functionalities, including managing users, viewing logs, and configuring system settings.

Step 2: Setting Up Parking Areas

For demonstration purposes, two parking areas were created. Each parking area includes several parking slots monitored by LDR sensors connected to an ESP32 microcontroller.

1. **Define Parking Areas and Slots:**

a. Using the Django admin interface, navigate to the "Parking" section.
b. Add two new parking areas: "Chich Downtown Parking" and "Centre commercial de Goma".
c. For each parking area, define multiple parking slots, each associated with an LDR sensor.

Step 3: Assigning Parking Managers

Two parking managers were assigned, one for each parking area. Parking managers are responsible for overseeing the operations within their respective areas, managing lost items, and handling reservations.

1. **Create Parking Manager Accounts:**

a. In the Django admin interface, navigate to the "Users" section.
b. Create new user accounts for the parking managers and assign them the "Parking Manager" role.

2. **Associate Managers with Parking Areas:**

Assign each parking manager to their respective parking areas (Parking Area 1 and Parking Area 2) using the admin interface.

Step 4: Enabling Real-Time Slot Monitoring

The ESP32 microcontrollers in each parking area continuously monitor the status of the parking slots using LDR sensors. The data is sent to the Django server, which updates the database in real time.

1. **Configure ESP32 to Send Data:**
   - Program the ESP32 microcontrollers to read data from the LDR sensors.
   - Establish a Wi-Fi connection and send the sensor data to the Django server via HTTP requests.

For Nyakumatt Holding parking, this code was used

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>
#define S0 16
#define S1 5
#define S2 4
#define S3 0
#define SIG A0
const char* ssid = "Arnold";
const char* password = "123456789";
const char* serverUrl = "https://cikuru-spaces.vercel.app/data/";
const char* esp_id = "No4";

int ldrValues[16];
int seuil = 135;
WiFiClientSecure wifiClient;

void setup() {
  Serial.begin(9600);
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(SIG, INPUT);

  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  Serial.println(WiFi.localIP());

  wifiClient.setInsecure();
}

void loop() {

  DynamicJsonDocument jsonDoc(8192);
  jsonDoc["esp_id"] = esp_id;

  for (int i = 0; i < 16; i++) {
    digitalWrite(S0, (i & 0x01) ? HIGH : LOW);
    digitalWrite(S1, (i & 0x02) ? HIGH : LOW);
    digitalWrite(S2, (i & 0x04) ? HIGH : LOW);
    digitalWrite(S3, (i & 0x08) ? HIGH : LOW);
    delay(10);
    ldrValues[i] = analogRead(SIG);

  }

  for (int i = 0; i < 16; i++) {
    String slot = "Slot" + String(i + 1);
    jsonDoc[slot] = ldrValues[i] > seuil;
  }

  String payload;
  serializeJson(jsonDoc, payload);
  Serial.println(payload);

  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    Serial.println("Sending HTTPS request...");

    http.begin(wifiClient, serverUrl);
    http.addHeader("Content-Type", "application/json");
    int httpResponseCode = http.POST(payload);

    if (httpResponseCode == 308) {
      String redirectUrl = http.getLocation();
      http.end();

      // Handle the redirection
```

```
      http.begin(wifiClient, redirectUrl);
      http.addHeader("Content-Type", "application/json");
      httpResponseCode = http.POST(payload);
    }

    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      Serial.println(response);
    } else {
      Serial.print("Error sending POST request: ");
      Serial.println(httpResponseCode);
    }
    http.end();
  } else {
    Serial.println("WiFi Disconnected");
  }

  delay(400);
}
```

Server-Side Processing:

- Django views handle incoming data and update the status of parking slots in the database.
- The updated status is reflected on the web interface in real time.

The view or function that I wrote to do so is written in the file views.py located in the Django application park.

Step 5: Lost Item Management

Parking managers can log and manage lost items reported within their parking areas.

1. **Log Lost Items:**

a. Parking managers use the web interface to add details of lost items found in their parking area.
b. Details include a description of the item, date found, and contact information.

2. **View and Update Lost Items:**

Managers can view all reported lost items and update their status (e.g., returned to owner).

Step 6: User Interactions and Reservations

1. **Create User Accounts:** Users can create accounts via the web interface.
2. **View Parking Availability:** Users log in to view real-time availability of parking slots in both parking areas.
3. **Make Reservations:**

a. Users select an available slot and make a reservation.
b. The system updates the slot status and sends a confirmation notification to the user.

4. **Cancel Reservations:** Users can cancel their reservations if needed, and the slot becomes available for others.
5. **Report Lost Items:**

a. Users can report lost items using the web interface.
b. The system notifies the respective parking manager of the reported item.

Step 7: Administrative Functions

The superuser can perform various administrative functions to ensure smooth operation and security.

1. **User Management:**

Add or remove users, including superusers, parking managers, and normal users.
Modify user roles and permissions.

2. **System Monitoring and Security:**

View system performance reports and logs, update security settings and review access logs for unauthorized activity, perform regular backups of system data.

3. **System Updates and Configuration:**

Apply system updates and verify their functionality, make configuration changes as needed and ensure they are properly applied.