SMART PARKING SYSTEM WITH REAL-TIME PARKING LOT STATUS MONITORING USING INTERNET OF THINGS (IOT) AND RADIO-FREQUENCY IDENTIFICATION (RFID)

TAN HONG ZHENG

UNIVERSITI TUNKU ABDUL RAHMAN

SMART PARKING SYSTEM WITH REAL-TIME PARKING LOT STATUS MONITORING USING INTERNET OF THINGS (IOT) AND RADIO-FREQUENCY IDENTIFICATION (RFID)

TAN HONG ZHENG

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Electronics Engineering With Honours-(EE)

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

May 2025

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature	:	[illegible]
Name	:	Tan Hong Zheng
ID No.	:	20AGB02853
Date	:	21-05-2025

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"SMART PARKING SYSTEM WITH REAL-TIME PARKING LOT STATUS MONITORING USING INTERNET OF THINGS (IOT) AND RADIO-FREQUENCY IDENTIFICATION (RFID)"** was prepared by **TAN HONG ZHENG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Electronics Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	[illegible]
Supervisor	:	Ts. Dr. Toh Pek Lan
Date	:	21/5/2025

Specially dedicated to

my beloved parents, for their unwavering support and endless love.

ACKNOWLEDGEMENTS

I would like to extend my deepest gratitude to everyone who contributed to the successful completion of this project. My sincere thanks go to my supervisor, Ts Dr. Toh Pek Lan, for her invaluable advice, guidance, and unwavering patience throughout the development of this research. Her expertise and insights have been instrumental in shaping the direction and quality of this work.

In addition, I am also profoundly grateful to my loving parents and friends for their constant support, encouragement, and assistance. Their belief in my abilities provided me with the strength and motivation needed to persevere through challenges.

I also appreciate the support of the university's staff and the resources provided, which were crucial in completing this research.

SMART PARKING SYSTEM WITH REAL-TIME PARKING LOT STATUS MONITORING USING INTERNET OF THINGS (IOT) AND RADIO-FREQUENCY IDENTIFICATION (RFID)

ABSTRACT

The ever-growing number of vehicles in urban areas has significantly intensified the challenge of finding available parking spaces, leading to increased frustration for drivers and contributing to environmental pollution due to prolonged vehicle idling and unnecessary driving. Real-time updates and efficient space utilization are rare features of most parking management systems on the market today. Apart from this, visibility is usually limited, especially in large parking lots, and the signs can be unclear or difficult to read, which causes drivers to have no idea where the available parking spots are. This project presents a Smart Parking System using IoT and RFID Technology designed to address these issues by providing a real-time parking lot status monitoring solution. The system integrates various hardware components, including eight IR sensors to detect vehicle presence at the parking slots, two servo motors for gate control, two ESP32 microcontrollers, two RFID readers with tags, an LCD display for showing parking status and time, an OLED display for showing RFID scanning messages, and a buzzer for audio feedback. Additionally, two IR sensors monitor vehicle presence at each of the gates. Besides that, custom developed HTML-PHP integrated web pages enable public users to view the status of each parking slot in real-time, whether it is “AVAILABLE”, “OCCUPIED”, or “RESERVED” and then access features like sign-up, login, top-up, and reservation, which allow the users to reserve a parking slot for a particular time before arriving. The user information, RFID scanning timestamps, reservation user and time, and status of each parking slot are stored in a MySQL database. Users can obtain parking information on the web page using their mobile devices by scanning a QR code or by visiting the provided URL link and logging into their registered accounts. The system operates by detecting vehicles at the entrance, verifying RFID UID with database, and managing gate operations based on slot availability and account balance. Furthermore, a ESP32-CAM is used to capture vehicle images at the gate entrance when the IR sensor detects movement and save them to Google Drive. The performance of the system will then be assessed based on its accuracy. Based on the results, the system prototype achieved 100 % accuracy. When compared to other proposed Smart Parking Systems, it is clear that this method is more cost-effective and reliable. This system aims to reduce parking-related frustrations and environmental impact to enhance overall urban mobility and sustainability.

Keywords: IoT, RFID, Smart Parking System, Real-time Monitoring, Reservation

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

°	Degree
%	Percent
AC	Alternating Current
AI	Artificial Intelligence
ANPR	Automatic Number Plate Recognition
CO_2	Carbon Dioxide
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DC	Direct Current
GMT	Greenwich Mean Time
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IR	Infrared
IoT	Internet of Things
I2C	Inter-Integrated Circuit
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LiDAR	Light Detection and Ranging
Mbps	Megabits per Second
MISO	Master In Slave Out

MOSI	Master Out Slave In
MP	Megapixels
MySQL	My Structured Query Language
NTP	Network Time Protocol
OCR	Optical Character Recognition
OLED	Organic Light Emitting Diode
OpenCV	Open Source Computer Vision Library
PHP	Hypertext Preprocessor (originally Personal Home Page)
PIR	Passive Infrared
PWM	Pulse Width Modulation
QR	Quick Response (Code)
RFID	Radio Frequency Identification
RTC	Real-Time Clock
SCL	Serial Clock Line
SDA	Serial Data Line
SMS	Short Message Service
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver/Transmitter
UID	Unique Identifier
USB	Universal Serial Bus
UTC	Coordinated Universal Time
WiFi	Wireless Fidelity
YOLO	You Only Look Once

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Introduction

In Chapter 1, the basics of the Internet of Things (IoT) and its importance will be introduced, along with an overview of the key technologies that enable IoT. Following this, the chapter will present the problem statement, focusing on the parking challenges in Malaysia and the disadvantages of traditional parking systems. The research aim and objectives will then be outlined. This chapter will also discuss the significance of this study, highlighting how smart parking systems can enhance urban mobility and reduce environmental impact. Lastly, the scope and limitations of the research will be covered.

1.2 Background of the Study

The Internet of Things (IoT) consists of a dynamic network involving physical devices and objects such as vehicles and appliances that use embedded sensors and software for connectivity. Through this network devices are able to collect and share data which leads to the development of diverse applications (Oracle, 2024). IoT devices play a vital role in modern life by connecting everything from basic smart home appliances to advanced industrial equipment and transport systems. These IoT devices assist in environmental farm monitoring, smart vehicle traffic management, controlling factory operations, and warehouse inventory tracking.

IoT demonstrates its importance by improving operational efficiency, facilitating informed decision-making based on data, and helping organizations save costs. Business processes become more productive and efficient when automation and real-time data are utilised. Besides that, businesses can make informed decisions because the data generated by IoT devices provide essential information about market trends, customer behavior, and operational performance. Additionally, IoT reduces costs by enabling automated processes and energy efficiency optimization which support sustainable practices.

In addition to that, some important technologies that make the IoT work include sensors, actuators, connectivity technologies, cloud, big data analytics, and security and privacy measures (IBM, 2024). Sensors are devices that can detect changes in the environment, such as temperature or light. Meanwhile, actuators are devices that perform actions in response to those changes, like opening a valve or switching on a motor. Connectivity technologies, which include Wi-Fi, Bluetooth, and cellular technologies are critical for transmitting data from IoT devices to the cloud. Cloud computing platforms provide the infrastructure required to store, process, and analyse data. Advanced analytics technologies, such as machine learning algorithms, assist in extracting valuable insights from enormous datasets. Furthermore, security and privacy technologies like encryption and intrusion detection systems are critical for defending IoT devices and the data they generate from cyber threats. Figure 1.1 shows the key components of IoT.

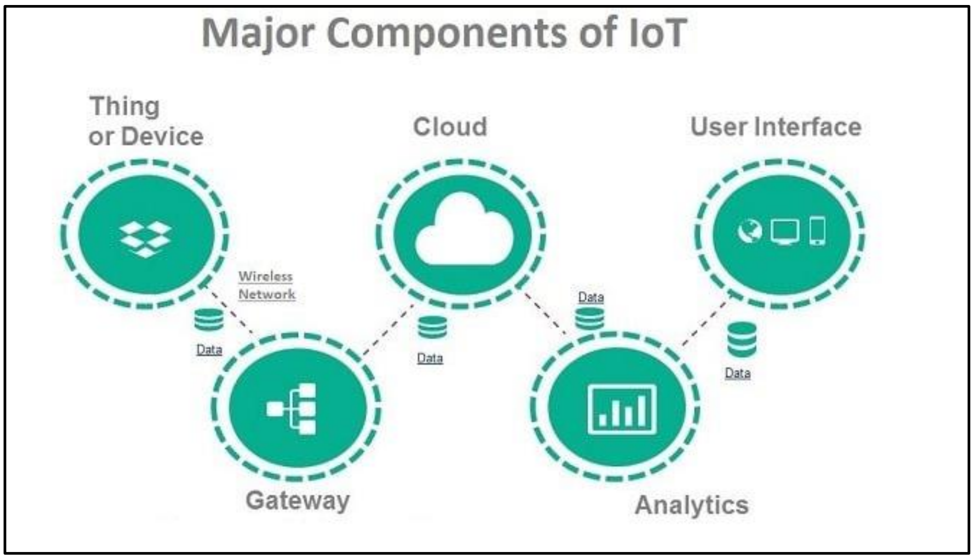


Figure 1.1: Major Components of Internet of Things (Rajiv, 2018)

Next, a notable application of IoT technology is in the realm of parking management through the use of Radio Frequency Identification (RFID). RFID technology employs electromagnetic fields to monitor and identify objects automatically. When combined with IoT, RFID enhances the capabilities of parking systems, leading to more secure, efficient, and automated processes.

The Smart Parking System proposed shows the integration of IoT and RFID technologies in urban areas. This system addresses common parking challenges, including congestion, inefficient space utilization, and environmental impact. By incorporating RFID tags into vehicles and equipping parking spaces with RFID readers, and IoT sensors, the Smart Parking System offers real-time monitoring of parking space availability. Drivers receive real-time information about the location of the nearest available parking spaces via smartphone apps, websites, or digital displays, substantially reducing the amount of time spent looking for parking.

1.3 Problem Statement

Malaysia is facing a rising parking challenge as the number of vehicles on the road grows rapidly. A survey conducted by Rakuten Insight in 2024 found that nearly 73 % of Malaysians own a car, which is a higher rate of car ownership compared to neighboring countries (Rakuten Insight, 2024). As of October, 2023, Malaysia had more than 36.3 million registered vehicles, surpassing the population of 32.4 million (Daim, 2023). This surge in vehicles has led to severe traffic congestion, particularly in urban areas like Klang Valley, where people spend an average of 44 hours each month stuck in traffic (BusinessToday, 2023). Shopping malls such as Sunway Pyramid, which sees over 50 000 vehicles daily, are struggling to manage the growing demand for parking (Lee, 2021).

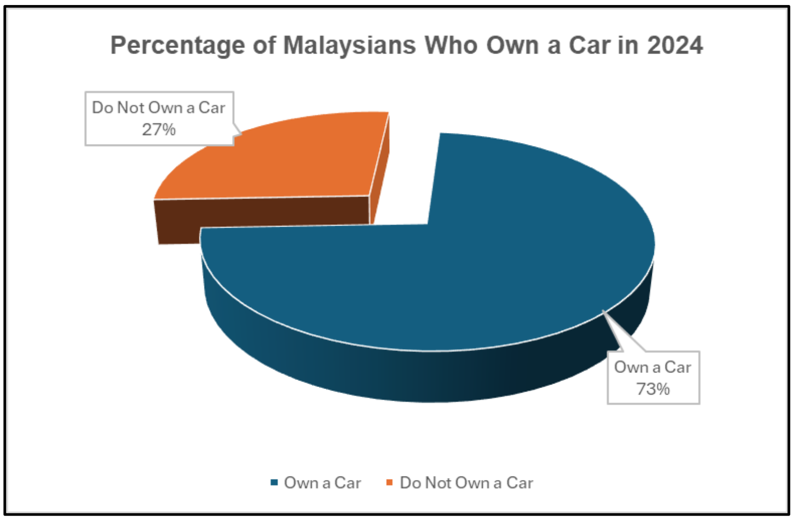


Figure 1.2: A Pie Chart Illustrating the Percentage of Malaysians who Own a Car in 2024

Next, traditional parking systems are outdated and inefficient. They usually involve manual processes, where drivers need to stop to collect a ticket and then pay later at a machine, or they may have to interact with a parking attendant. This process can create long queues and can cause bottlenecks at the entrances, especially in busy shopping malls or public venues. The queues not only delay entry but also cause traffic congestion on surrounding roads, contributing to overall traffic congestion in the area (Goh, 2024).

Additionally, real-time information regarding available parking spaces is not provided by traditional parking systems, this forces drivers to circle around parking lots or streets in search of an open space. On average, it takes about 7.8 minutes to find a parking spot (Hong *et al*., 2023). This process can take up to 20 minutes or more in crowded areas, especially during peak hours. As a result, drivers waste fuel and emit more greenhouse gases like CO_2. The lack of smart parking systems also means that parking spaces are often used inefficiently. During peak times or holidays, the demand for parking exceeds what is available, leading to even more frustration for drivers.

Besides that, in large parking lots or multi-story garages, it can be difficult for drivers to find their way, especially if the signage is unclear or if the parking facility is crowded. Traditional systems do not provide guidance or real-time updates, leaving drivers to navigate on their own (Elfaki *et al*., 2023). This can lead to drivers getting lost, wasting time, and becoming frustrated. In some cases, drivers may even forget where they parked, causing further delays and stress when trying to locate their vehicle.

Moreover, many traditional parking systems require cash payments, which can be inconvenient for drivers who do not carry cash or prefer to pay electronically. The need to find cash or wait in line to pay at a machine can be time-consuming and frustrating. This inconvenience can lead to longer exit times, further contributing to congestion at parking facilities. It also detracts from the overall customer experience, making people less likely to return to that parking facility in the future.

In addition, the environmental consequences of inefficient parking are significant. Cars that are idling while searching for parking spots consume more fuel and emit more CO_2. Research shows that just one hour of traffic congestion can increase air pollutants and carbon dioxide emissions by up to 30 % (Elfaki *et al*., 2023). This not only contributes to climate change but also worsens air quality, especially in crowded cities.

Next, security is another concern with traditional parking systems. Traditional parking systems often lack advanced security measures, making it easier for unauthorized vehicles to enter or for theft and vandalism to occur. Without real-time monitoring and tracking, it is difficult for parking operators to ensure the safety of parked vehicles (Koya *et al*., 2024). Security concerns can deter people from using certain parking facilities, especially in areas where crime rates are higher. This can result in lost revenue for parking operators and increased anxiety for drivers (Goh, 2024).

Operationally, traditional parking systems make it difficult for shopping malls and other facilities to manage their parking spaces efficiently. They do not provide the data needed to optimize parking operations, leading to lost revenue and unhappy customers. In contrast, smart parking systems can analyze parking usage data, helping management make better decisions about how to run their parking facilities (Hong *et al*., 2023).

Given these challenges, the implementation of Smart Parking Systems is not just a convenience but a necessity. By leveraging the IoT and cloud technologies, these systems provide real-time information on available parking slots, lead drivers directly to open spots, and offer flexible payment options through mobile applications. This approach reduces traffic congestion, minimizes environmental impact, and significantly enhances the overall driving and parking experience. Moreover, Smart Parking Systems enhance security by using technologies like RFID and automatic license plate recognition to monitor and control access to parking facilities, reducing the risk of theft and vandalism. These systems also enable advanced data analytics, allowing parking operators to optimize the use of their facilities and improve customer satisfaction.

Malaysia’s current parking systems are struggling to keep up with the growing number of vehicles. The lack of smart parking solutions is causing traffic jams, increasing pollution, and frustrating drivers. Smart parking systems offer a comprehensive solution to the challenges faced by drivers and parking operators by improving parking efficiency, reducing environmental impact, and enhancing driver satisfaction across the nation.

1.4 Aim and Objectives

The aim of this project is to develop a Smart Parking System using IoT and RFID technology that allows real-time parking lot status monitoring and access control.

The objective of this project is to design an IR sensor-based parking lot monitoring system using two ESP32 microcontrollers with RFID readers for access control. Besides that, the second objective is to develop custom-designed HTML-PHP integrated web pages for real-time parking lot status monitoring with a login page, sign-up page, top-up page, and reservation page. Moreover, the third objective is to allow users to reserve a parking slot through the reservation page before they arrive. Next, the fourth objective is to store user information, RFID scanning timestamps, reservation user and time, and status of each parking slot in the MySQL database. Furthermore, the fifth objective is to use the ESP32-CAM to capture images and save them to Google Drive when the IR sensor detects movement at the entrance. Finally, the sixth objective is to assess the performance of the system based on its accuracy.

1.5 Importance of the Study

The growing urbanization and rise in vehicle numbers on the road have worsened the parking problem, which caused significant time wastage and increased environmental pollution levels. Hence, the importance of an efficient parking management system cannot be ignored. An IoT-based smart parking system can provide real-time data on parking availability, this decreases search durations for parking spots and the associated environmental impact.

Moreover, integrating RFID technology for access control can enhance security, streamline the parking process, and eliminate the need for manual labour to control the gate. This infrastructure's one-time installation cost can help parking lot owners save on the recurring monthly salary costs of manual labor. The cross-checking between scanned RFID UID and the UID stored in the database adds another security layer, this ensures that only authorized users can enter the parking lot. By requiring users to log in to view parking lot statuses, the system ensures that this information is accessible only to legitimate users, thereby preventing misuse. Furthermore, the stored user information from account registration and captured images at the entrance can be retrieved to assist law enforcement agencies in tracking individuals involved in illegal activities.

Besides that, the integration of IoT and RFID technology allows the parking lot owner to gather information about occupancy rates, and length of stay through the RFID scanning timestamp stored in the MySQL database. This information can be used to make better judgements about whether the current parking lot is adequate, especially during peak hours, whether a new car park should be built, and if yes, where to build the new car park. For example, by tracking real-time occupancy rates and adjusting entry prices accordingly, owners can maximize income from paid parking and ensure that adequate spots are available.

Furthermore, the reservation feature allows users to book a parking slot before arriving. This reduces the time spent searching for available parking slots, minimizes congestion, and makes the entire parking experience smoother. The ability to manage parking spaces efficiently not only benefits individual drivers but also contributes to the overall optimization of urban infrastructure. This study's significance lies in its potential to address these issues through innovative technological solutions.

1.6 Scope and Limitation of the Study

1.6.1 Scope

This research focuses on developing a smart parking system with real-time parking lot status monitoring and access control using IoT and RFID technology. The study includes designing both the hardware and software components and evaluating the system’s performance in a controlled environment.

1.6.2 Limitations

The research is limited to a prototype system and does not encompass large-scale deployment. For large-scale deployment, more powerful microcontrollers, higher quality RFID readers, faster internet speed, or other advanced hardware may be required to maintain system reliability and efficiency. The accuracy of the system may be influenced by external factors such as hardware and infrastructure limitations, and environmental conditions such as lighting. Besides that, internet speed can impact the responsiveness of the web pages, server, and database refresh speed, which may affect the speed of gate opening and closing due to data verification with the database. This project assumes access to high-speed internet.

Additionally, the integration of the ESP32-CAM with an IR sensor for motion detection comes with some limitations. The IR sensor may trigger false detections due to factors like the movement of small animals. Poor lighting can also reduce image quality, making the captured photos less reliable. Furthermore, the ESP32-CAM is a budget camera, so the image quality may not be as clear as higher-end alternatives, which can affect the overall reliability of the system. Moreover, the Google Drive cloud storage can fill up quickly if there are frequent detections.

Furthermore, the experiment trials assume an ideal condition in which all of the vehicles occupying the parking slots are properly parked. As a result, the generated results might not be a perfect representation of the system’s accuracy when it is implemented in real life. For example, a poor vehicle parking position can have an impact on the accuracy of the system. The IR sensor may not be able to detect a car if it is not parked inside the box drawn in the parking space or if it is positioned too far away from the sensor, which is outside of its detectable range.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In Chapter 2, a literature review will be conducted to explore the work of various researchers on smart parking systems. The review will examine the different methods and technologies proposed by different authors, highlighting both their strengths and limitations. This analysis will provide a through overview of the current technologies and identify gaps in existing research. Following this comparison, the chapter will explain the rationale behind the choices made for this project, based on insights gained from the literature review. The objective is to demonstrate how this project aligns with or diverges from existing studies.

2.2 Smart Car Parking Mobile Application based on RFID and IoT Presented by Saeliw *et al.* (2019)

This paper presents the development of a smart car parking system that integrates RFID and IoT technologies with a mobile application. The system utilises the Arduino ESP8266 microcontroller and the HY-SRF05 ultrasonic sensor module to monitor the occupancy status of parking spaces. The data collected by the sensors is transmitted via Wi-Fi to a server managed by the Apache web server, where it is stored in a MySQL database and Firebase for real-time data management. The block diagram of the system is illustrated in Figure 2.1.

The system includes an Android-based mobile application that provides users with real-time updates on parking space availability. The application features a login and registration interface, where users can create accounts to access the parking data, as illustrated in Figure 2.2. In addition to the mobile app, the system is integrated with the Line messaging application, which sends notifications to users about the availability of parking spots, as illustrated in Figure 2.3. The application is developed using Java for Android mobile application, PHP for server-side scripting, and JavaScript, CSS, and Bootstrap for the web interface. Additionally, the system allows parking managers to generate daily usage reports as illustrated in Figure 2.4.

Advantages:

The system offers real-time monitoring of parking spaces, ensuring that users have access to the most current information about parking availability. By using the Line messaging app to notify users when parking status changes, the system makes it easier for users to find parking by keeping them informed in real-time about available spaces. The use of RFID technology enhances safety by automating the entry and exit process, reducing the risk of unauthorized access, and making the process more secure. Additionally, the system eliminates the need for manual ticketing or cash payments, which offers a more convenient and efficient user experience. Additionally, the system includes administrative tools that allow parking managers to update availability and generate daily usage reports in MS Excel format, which is useful for planning and operational purposes. The use of open-source software, such as MySQL, Firebase, Apache, and Android Studio, helps keep the development and deployment costs low.

Disadvantages:

The system is currently limited to Android devices, which restricts its usability for people using other operating systems. Its effectiveness is also heavily dependent on having a stable internet connection for real-time updates and notifications, which could be a challenge in areas with poor connectivity. Additionally, the hardware used, including the Arduino ESP8266 and ultrasonic sensors, may not be robust enough for larger or more complex parking environments.

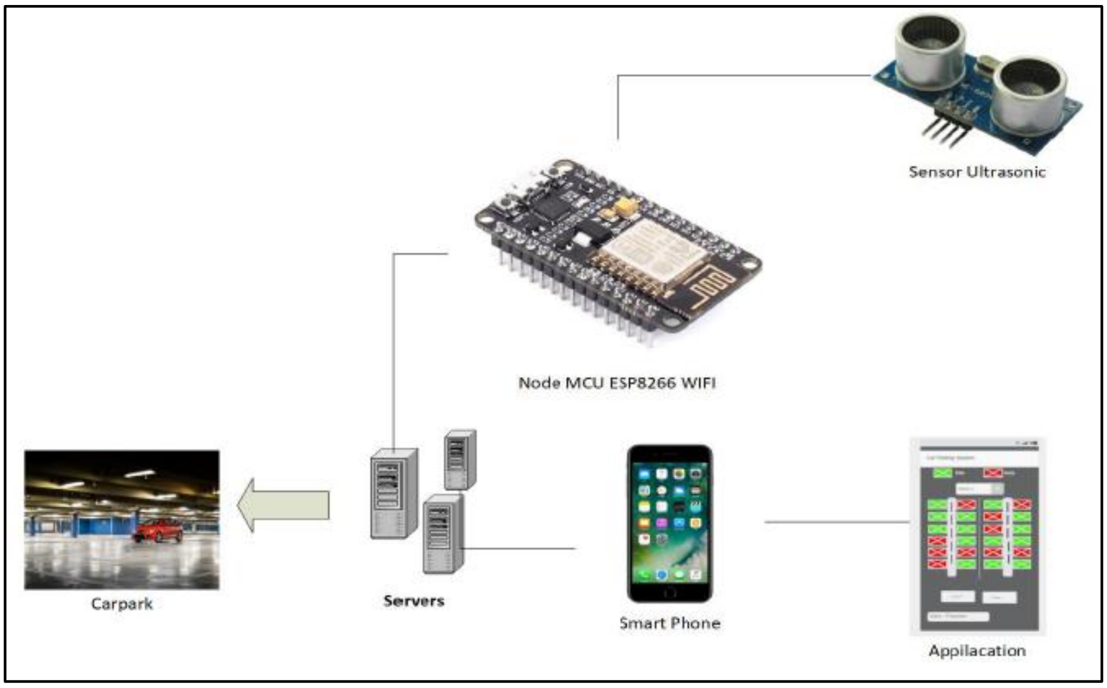


Figure 2.1: The Block Diagram of the System (Saeliw *et al.*, 2019)

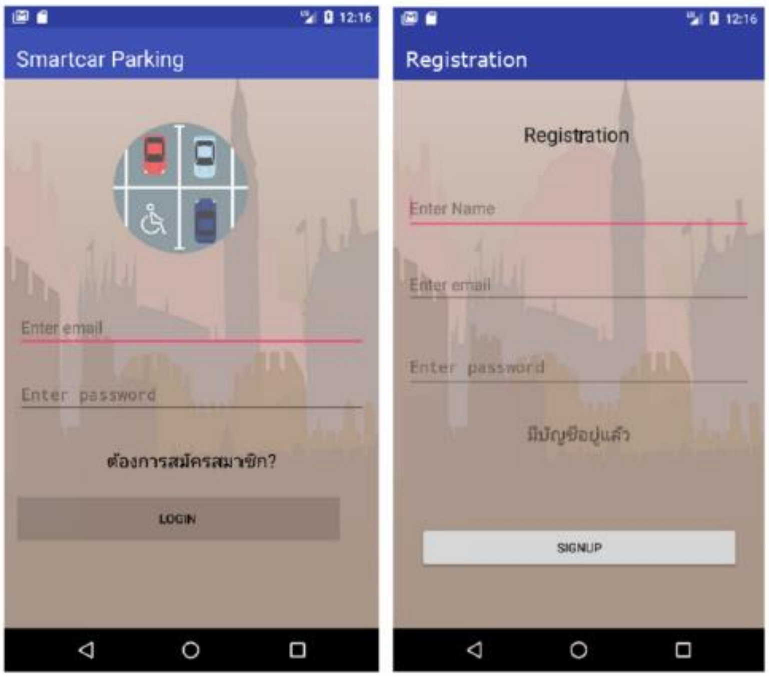


Figure 2.2: The Android Application Login and Registration Interface (Saeliw *et al.*, 2019)

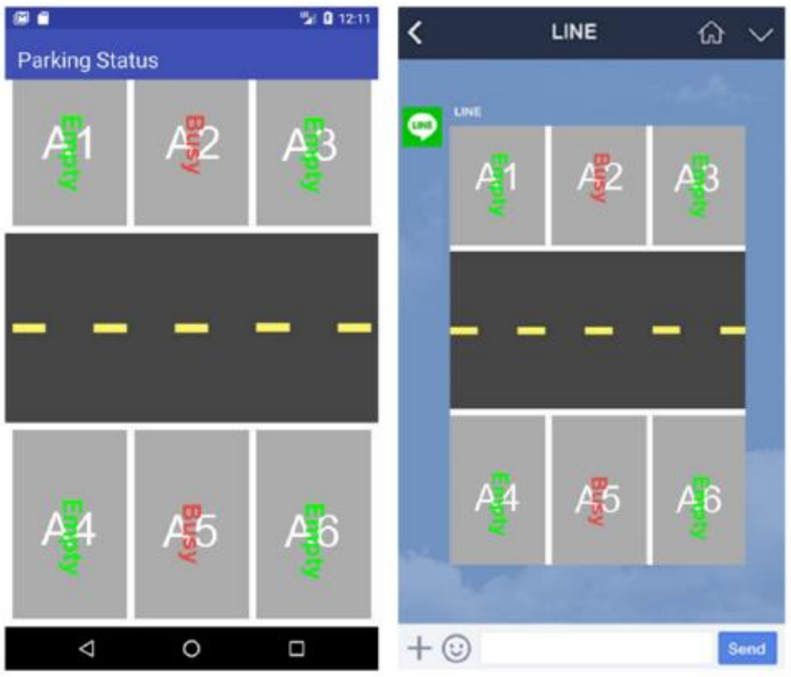
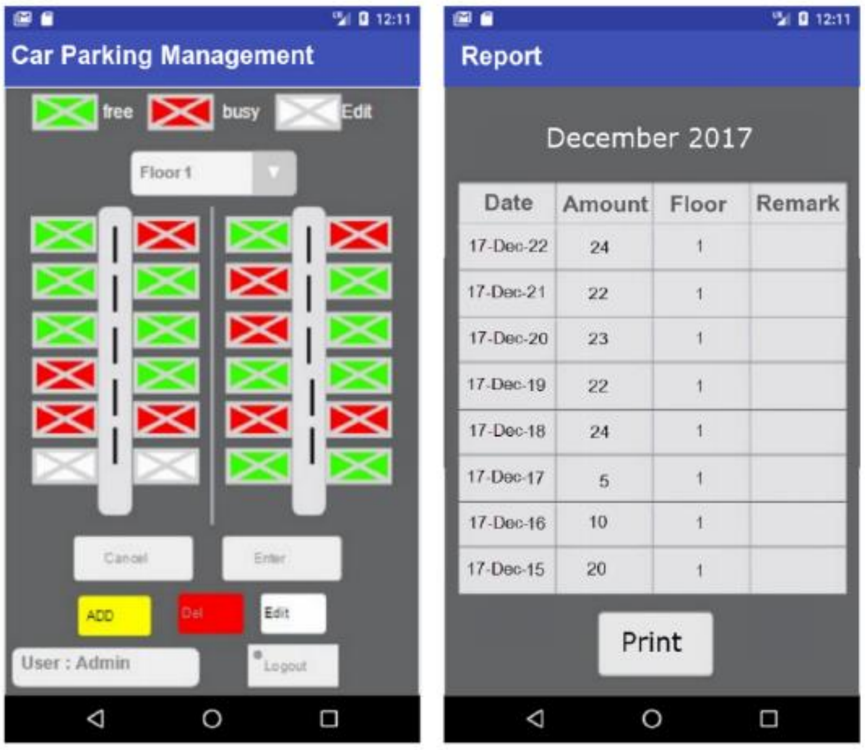


Figure 2.3: Car Parking Status Interface and Line Mobile Application Notification Update (Saeliw *et al.*, 2019)



December 2017

Date	Amount	Floor	Remark
17-Dec-22	24	1	
17-Dec-21	22	1	
17-Dec-20	23	1	
17-Dec-19	22	1	
17-Dec-18	24	1	
17-Dec-17	5	1	
17-Dec-16	10	1	
17-Dec-15	20	1	

Figure 2.4: Car Parking Management and Daily Usage Report Interface (Saeliw *et al.*, 2019)

2.3 Smart Parking Guidance System Using 360° Camera and Haar-Cascade Classifier on IoT System Presented by Salma, Olanrewaju, and Arman (2019)

This paper presents a Smart Parking Guidance System that integrates IoT technology with advanced image processing (edge detection) and Haar-Cascade classifiers to enhance parking management efficiency. Image processing involves the use of algorithms to analyze and manipulate visual data captured by cameras to allow the systems to identify and interpret various elements within an image. Haar-Cascade classifier is a popular technique in image processing, it is used to detect objects within an image by identifying patterns of light and dark areas that form the boundaries of objects. The overall process of the system is shown in Figure 2.5. In this system, a 360° camera is placed in the center of the parking lot to capture images of all parking slots using the image processing method, as illustrated in Figure 2.6. These images are then processed by a Raspberry Pi 3 Model B using Python and OpenCV libraries to determine the occupancy status of the parking slots. The processed data is uploaded in real-time to a Firebase cloud platform and the mobile application. Then, the user can access the parking lot information and make reservations on an Android mobile application developed in Android Studio.

Advantages:

One of the major benefits of this system is its cost-effectiveness. By using a single 360-degree camera instead of multiple sensors, the system significantly reduces the cost, making it a more feasible option for large parking facilities. The system combined with image processing has success rate of 99.74 % in ideal conditions, which is a very high accuracy. This means the users can trust the information provided about available parking spots. The real-time updates through the Firebase cloud and Android app, offer convenience and immediate access to parking data, making it easier for drivers to find a spot quickly. Moreover, the booking and reservation system adds an extra layer of convenience, allowing drivers to secure a parking spot before they even arrive.

Disadvantages:

However, there are some downsides. The accuracy of the system can drop slightly in low-light conditions, which might cause some parking spots to be incorrectly reported as occupied or vacant. The quality of the camera used (e.g., 5 MP versus 20 MP) also plays a significant role. Higher resolution cameras can offer better accuracy and detail in detecting vehicles, but they also increase the cost of the system. Moreover, the 360° camera also requires a lot of bandwidth, which could slow down data transmission if the internet connection is weak. Additionally, setting up the system can be a bit complex and time-consuming, as the Haar-Cascade classifier needs to be trained with a large number of images to work effectively. Furthermore, while the 360° camera method is cost-effective for large parking lots, it becomes less practical for smaller parking facilities. In such cases, the component and development costs are significantly higher than traditional sensor methods like IR sensors and ultrasonic sensors, which makes the system less economical for smaller-scale implementations.

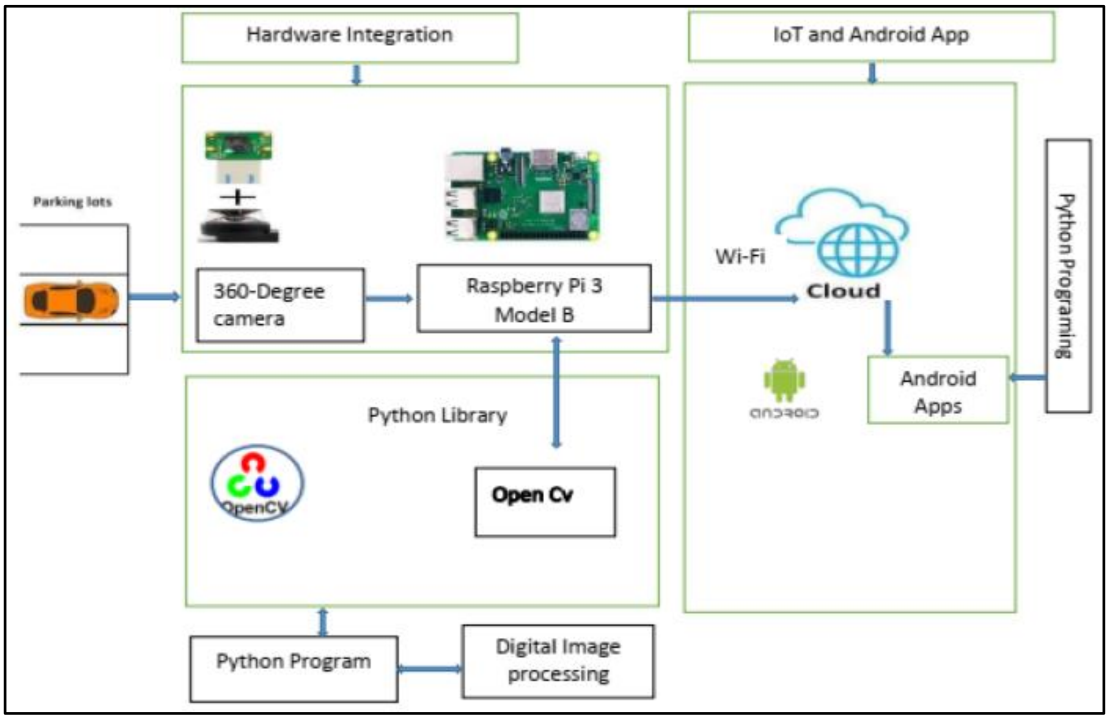


Figure 2.5: The Whole Process of the Smart Parking Guidance System (Salma, Olanrewaju, and Arman, 2019)

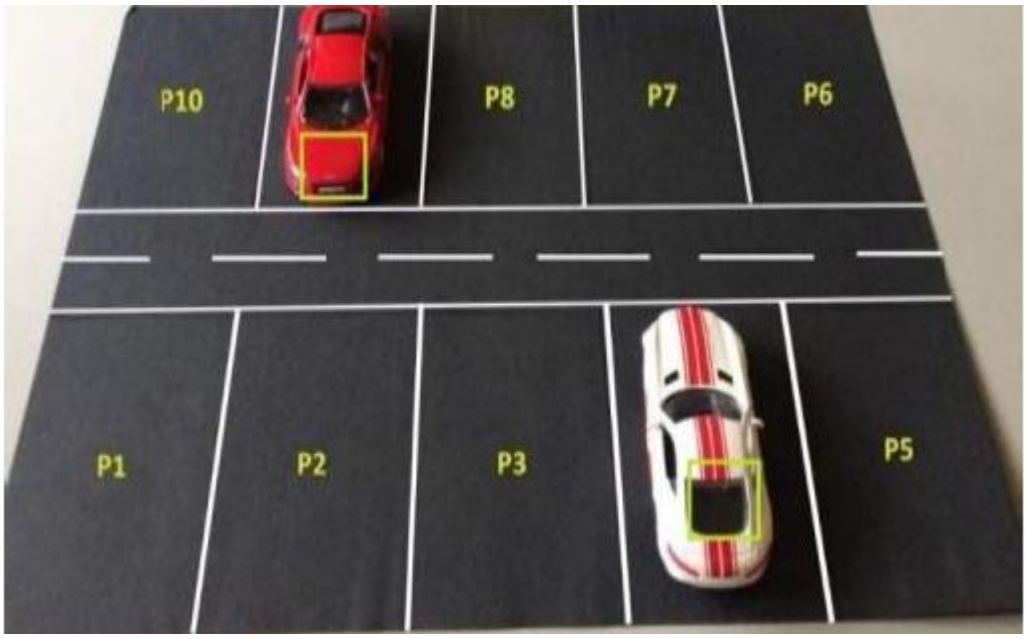


Figure 2.6: Vehicle Detection Image Processing for the Prototype (Salma, Olanrewaju, and Arman, 2019)

2.4 Smart Parking System using IoT Presented by Elakya *et al*. (2019)

This paper introduces a Smart Parking System that combines IoT with GSM and RFID technologies to create a user-friendly and secure parking experience. At the core of the system is an Arduino Nano microcontroller, which is linked to both a GSM module and a Wi-Fi module. The GSM module, which is illustrated in Figure 2.7 enables communication between the parking system and the mobile phones of the users to send updates about parking availability via SMS or voice messages. The Wi-Fi module is used to upload data from IR sensors, which detect whether a parking space is occupied or not to a cloud server. RFID technology is also used so that only registered users can access the parking facility by scanning their RFID cards at the entrance. Once the card is scanned, the system checks the parking status stored in the cloud and sends this information directly to the user's phone. At the end of this paper, the authors also suggest that for future improvements, GPS, license plate scanner, and reservation function could be added to improve the user experience.

Advantages:

One of the key benefits of this system is the use of GSM technology to provide real-time parking updates directly to users' phones via SMS. This makes it convenient and accessible as there is no need for users to download mobile applications or visit websites as the information is delivered straight to them via SMS. This approach is particularly useful in areas with limited internet access. Additionally, the use of RFID technology offers an added level of protection by ensuring that only authorized users can enter the parking area, which helps protect vehicles from unauthorized access. Besides that, the use of IR sensors has several advantages, like being cost-effective, easy to implement, having a fast response time, low power consumption, and capable of detecting a wide range of objects and distances, depending on the calibration as the sensitivity can be adjusted.

Disadvantages:

However, the system's reliance on additional hardware like GSM and Wi-Fi modules adds the complexity and cost of implementation of the system. The Arduino Nano, by itself, cannot connect to the internet or mobile networks, so Wi-Fi modules are essential for the system to function. This complicates the circuit design and can make large-scale deployment more difficult due to higher hardware prices and potential challenges in handling the extra components. Besides that, the performance of IR sensors is highly dependent on the surrounding light intensity and can be significantly affected by the color and surface characteristics of objects. For instance, darker surfaces tend to absorb more IR rays, resulting in fewer rays being reflected back to the sensor, which can lead to inaccurate readings. This could cause the system to mistakenly indicate that a parking slot is available when it is actually occupied. Therefore, an object with a high reflective index will provide more reliable detection, and vice-versa.

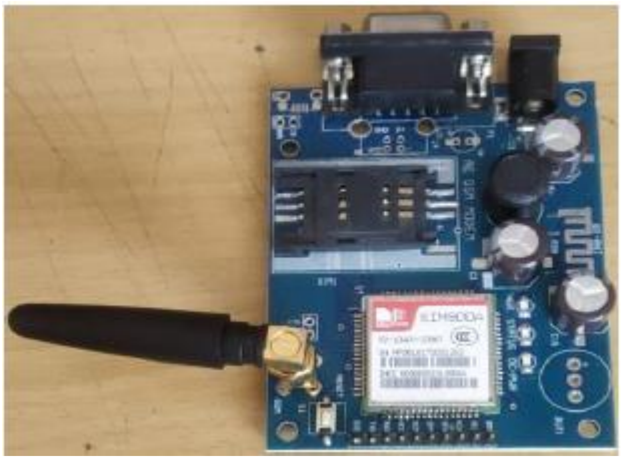


Figure 2.7: GSM Module (Elakya *et al*., 2019)

2.5 Intelligent Parking Management Using ANPR Technology Presented by Lincy *et al*. (2024)

The paper introduces a smart parking system with two main modules, which are the ANPR (Automatic Number Plate Recognition) module and the Vacant Slot Detection module. The ANPR module uses YOLO v4 for real-time detection of vehicle license plates and Tesseract OCR (Optical Character Recognition) for character recognition, automating the detection of vehicles and logging their entry and exit times. This module was trained on 1,500 vehicle images under various lighting and weather conditions to ensure accurate recognition. The Vacant Slot Detection module uses OpenCV for image processing and applies techniques such as thresholding, contour detection, and edge detection to determine whether a parking space is occupied or vacant. This module was trained on 1,000 images of parking spaces (vacant and occupied) to enhance its accuracy in different environments. The ANPR and Vacant Slot Detection Process Flow are shown in Figure 2.8 and Figure 2.9, respectively. Figure 2.10 shows the license plate detection GUI while Figure 2.11 shows the vacant slot detection interface.

Advantages:

The combination of ANPR and Vacant Slot Detection brings modules several advantages. The ANPR module enhances security by automating vehicle identification, ensuring only authorized vehicles can enter the facility. It streamlines operations by removing the need for manual checks and improves overall efficiency. The Vacant Slot Detection module eliminates the need for multiple physical sensors. Instead, it uses a single camera to monitor multiple spaces, which reduces the costs of installation and maintenance. This setup is particularly useful for larger parking lots, where managing each individual space with sensors would be expensive and complex. Overall, the system is cost-effective, scalable, and more flexible in terms of deployment.

Disadvantages:

Despite its benefits, the system does face some limitations. The accuracy of the ANPR module becomes lower if the captured images are unclear, which can happen in poor lighting, bad weather, or when plates are damaged. Similarly, the Vacant Slot Detection module relies on clear, high-quality images for precise detection, meaning a high-quality camera is essential, which will increase the costs. Both modules also require significant computing power, which adds to the overall complexity and expense of the system. Additionally, if the parking layout changes, the system might need adjustments, which could be time-consuming. ANPR systems also sometimes struggle with characters that look similar, like 'O' and '0' or 'B' and '8', which can lead to occasional errors in vehicle identification. Although this system is efficient, it may need optimal conditions and infrastructure to perform at its best.

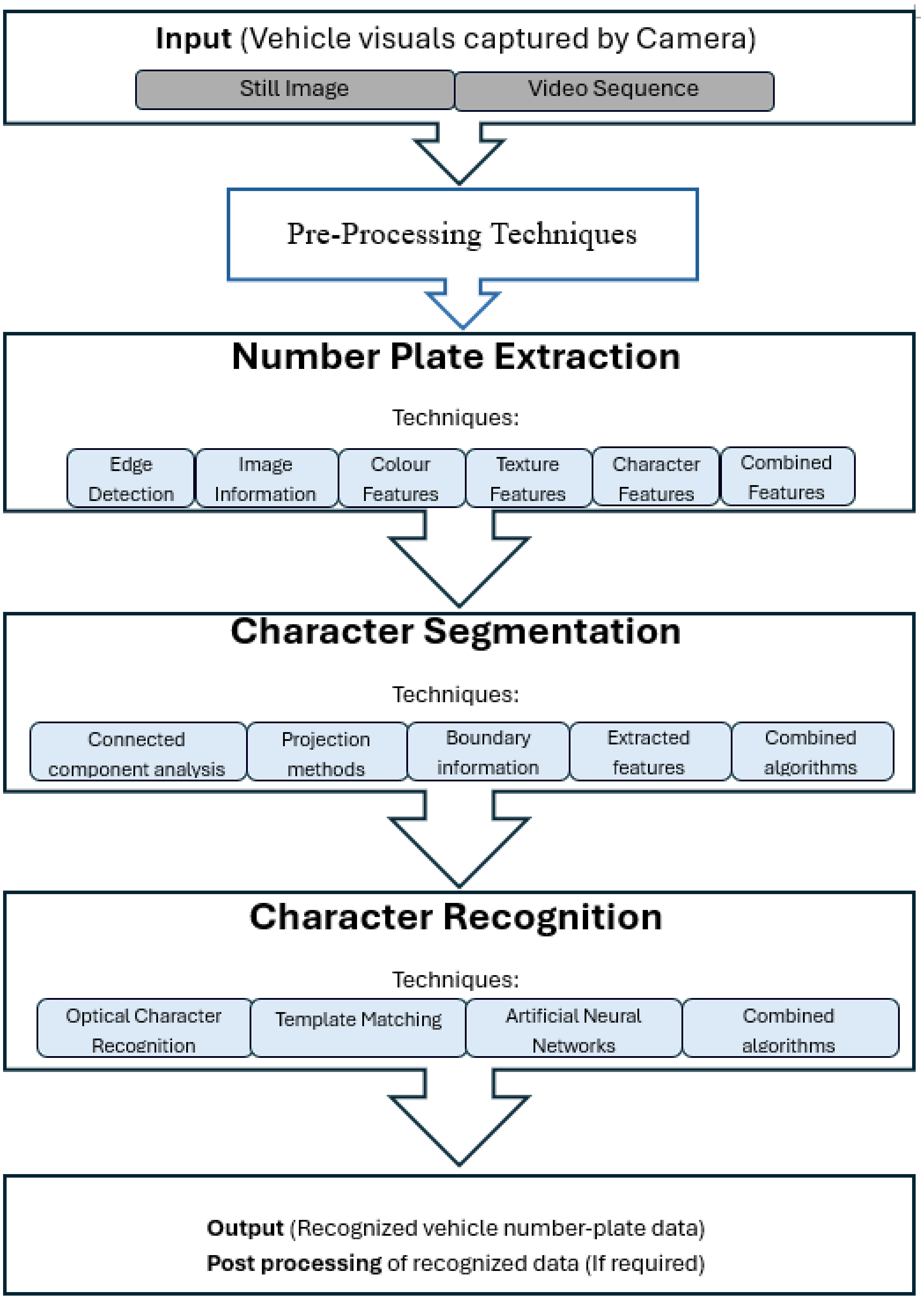


Figure 2.8: ANPR Process Flow (Lincy *et al*., 2024)

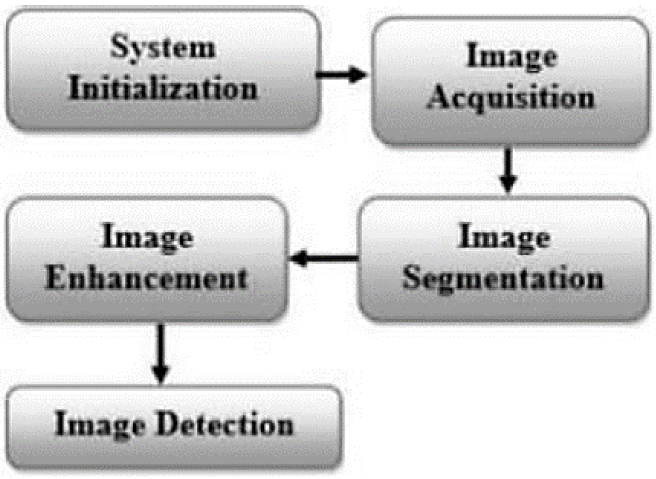
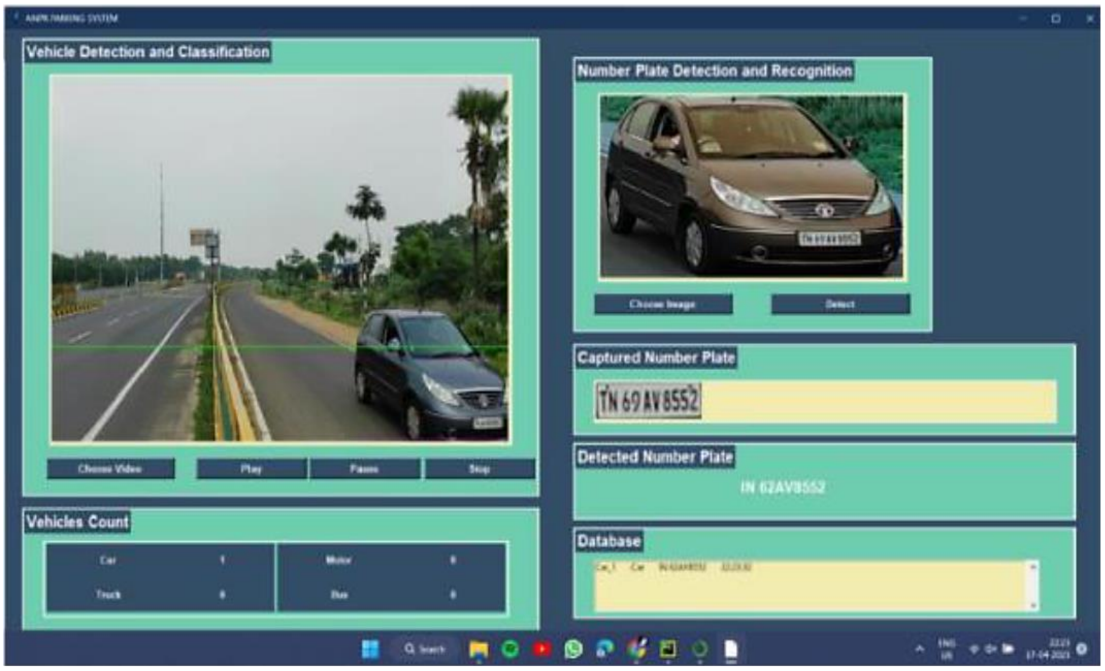


Figure 2.9: Vacant Slot Detection Process Flow (Lincy *et al*., 2024)



TN 69 AV 8552

IN 62AV8552

Figure 2.10: License Plate Detection (ANPR) Graphical User Interface (Lincy *et al*., 2024)

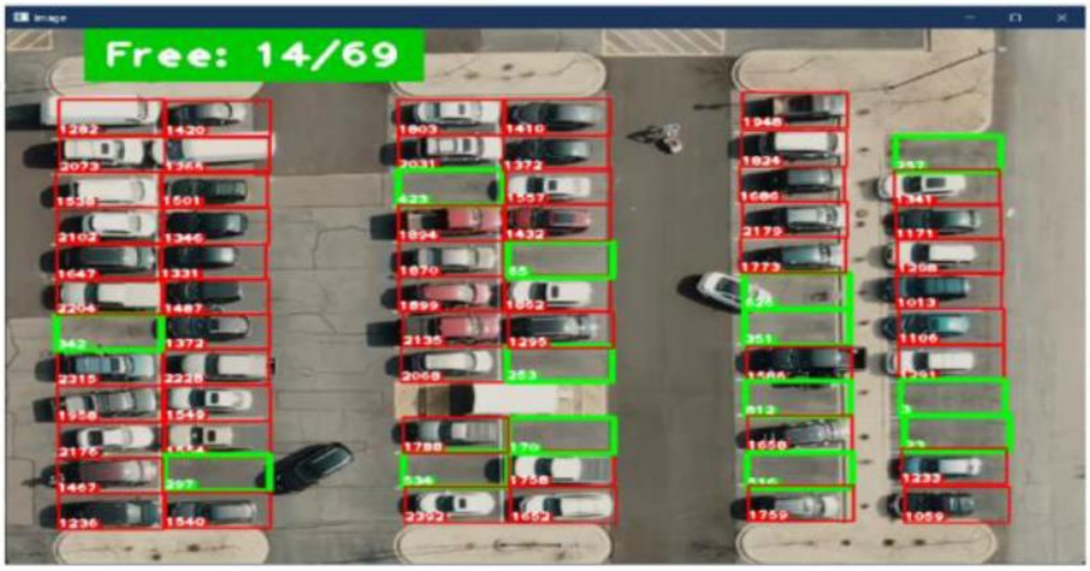


Figure 2.11: Vacant Slot Detection (Lincy *et al*., 2024)

2.6 A Smart Real-Time Parking Control and Monitoring System Presented by Elfaki *et al*. (2023)

This paper discusses a smart real-time parking control and monitoring system that integrates ANPR and reservation management to address parking challenges such as congestion, inefficiency, and incorrect parking in urban areas. The system consists of monitoring units at each parking slot, each parking slot is equipped with sensors (motion sensor like PIR sensor or range-finder sensor like HY-SRF05 ultrasonic sensor) to detect vehicles presence and a camera (ESP32-CAM) to capture license plate images, as shown in Figure 2.12 and Figure 2.13.

Firstly, a mobile application that allows visitors to reserve parking slots was developed, which is shown in Figure 2.14. The visitor can request a parking slot, the system will then check slot availability in real-time and assigns a parking slot to the visitor. The assigned slot information is then communicated back to the visitor through the application. Upon the arrival of the visitor’s car at the parking slot, the monitoring unit detects its presence and captures the license plate image, which is processed by the ANPR system to identify the characters using optical character recognition (OCR).

The recognized license plate data is then verified against the reservation records stored in a cloud server to ensure the vehicle is parked in the correct slot. If the visitors park in the wrong space, alerts are immediately sent to both the visitors and the parking administrator.

Next, for the employees, parking slots are assigned to them based on their official work hours or shifts using a priority queue data structure. Once assigned, the employees are expected to park in the designated slots. The main function of the mobile application for employees is to receive notifications or alerts if they park in the wrong slot, rather than to make reservations manually.

Advantages:

The reservation function offers several advantages, including dynamic slot allocation based on real-time demand, which optimizes parking space usage and reduces congestion and time wasted to search for available parking slot. The automatic detection of vehicles and verification against reservation records enhances parking security and convenience. The integration of a cloud-based system and mobile application facilitates efficient communication, with real-time notifications sent to users for confirmation or alerts if a vehicle is parked at the wrong slot. Additionally, the use of range-finder sensors provides higher accuracy in vehicle detection, which makes the system more reliable.

Disadvantages:

However, there are also some limitations. The accuracy of the ANPR module is highly dependent on the quality of the images captured. Adverse weather conditions, such as heavy rain, fog, or poor lighting, can reduce the effectiveness of the ANPR system. Furthermore, out-of-control scenarios like vehicle plate tampering, damaged plates, or vehicles entering without license plates could prevent the system from correctly verifying reservations, potentially causing security concerns or unauthorized access to the facility. Another issue is that the reservation system depends on stable internet connectivity for real-time updates and synchronization. In areas with unreliable or weak network infrastructure, this could lead to delays or failures in verifying reservations, which might cause inconvenience for users. Next, the initial cost of deploying IoT devices, sensors, and cameras for each parking slot may be high, especially for large-scale implementations. Lastly, since parking slots are automatically assigned based on work shifts, employees have no control over which slots they get. This could be inconvenient if an assigned slot is far from their workplace or if they prefer a specific parking area.

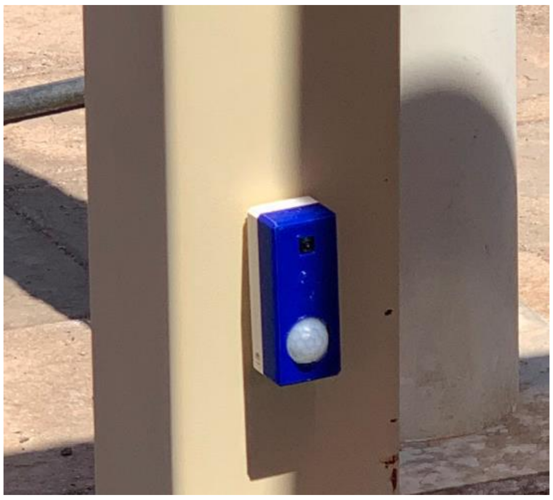


Figure 2.12: The Monitoring Unit Consists of Motion Sensor and Camera (Elfaki *et al.*, 2023)

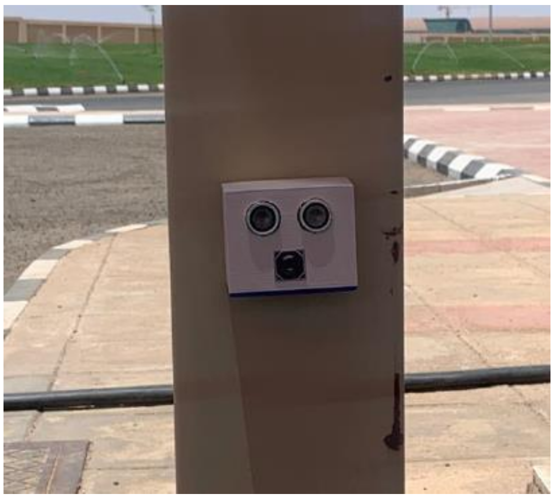
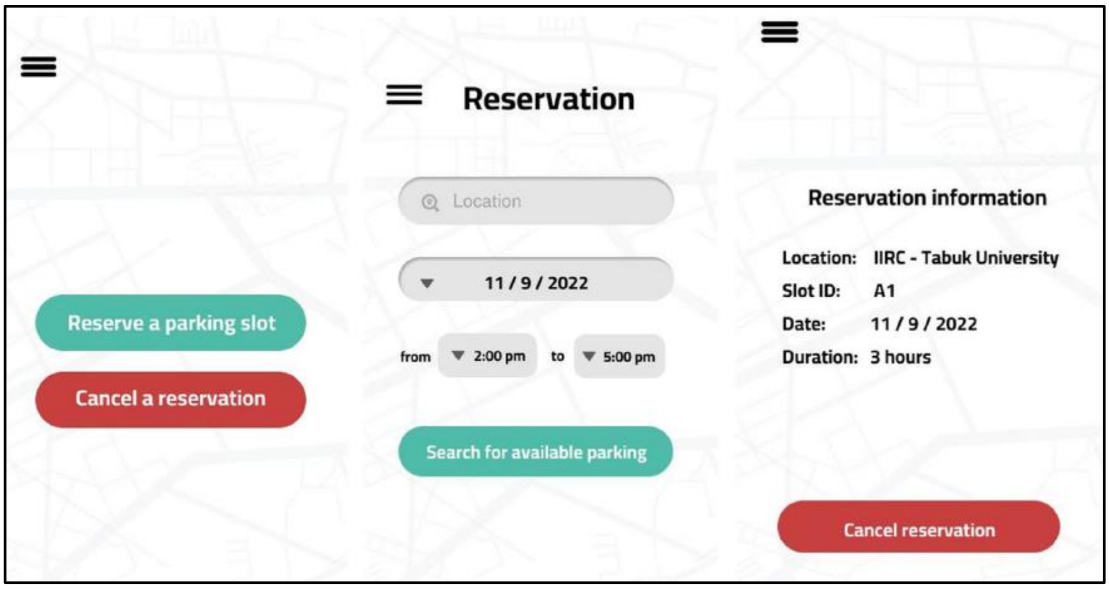


Figure 2.13: The Monitoring Unit Consists of Range Finder Sensor and Camera (Elfaki *et al.*, 2023)



Location:	IIRC - Tabuk University
Slot ID:	A1
Date:	11 / 9 / 2022
Duration:	3 hours

Figure 2.14: Mobile Application Reservation Function for User (Elfaki *et al.*, 2023)

2.7 Summary

The studies included in the literature review were chosen for their range of technologies and approaches in smart parking systems. These works cover a broad range, from sensors, RFID, and GSM to camera and ANPR, providing insight into how different methods are being used to tackle parking issues. The variety in user interfaces, whether through mobile applications, web applications, or systems that do not require an application, highlights how user interaction varies, which is a key factor in the effectiveness of these systems. The works also include both well-established technologies such as RFID and newer, more innovative solutions, such as ANPR, which involves advanced image processing algorithm. Table 2.1 shows the summary of the technologies and user interfaces used by different authors and my project choices.

Table 2.1: Literature Review Summary

Reference	**Technology**							**User Interface**		
	Sensors	**Camera**	**ANPR**	**RFID**	**GSM**	**Database**	**Reservation**	**Web Application**	**Mobile Application**	**Without Application**
Saeliw *et al.* (2019)	/			/		/			/	
Salma, Olanrewaju, and Arman (2019)		/				/			/	
Elakya *et al.* (2019)	/			/	/	/				/
Lincy *et al.* (2024)		/	/			/				/
Elfaki *et al.* (2023)	/	/	/			/	/		/	
My Project	/	/		/		/	/	/		

Firstly, IR sensors were chosen for their cost-effectiveness, ease of installation, and consistent performance in detecting vehicles. Compared to ultrasonic sensors, which are more expensive and require precise calibration to work accurately, IR sensors are simpler to deploy and maintain. While magnetic sensors are effective in detecting vehicles, they require more complex installation, such as embedding in the ground. Camera-based detection with image processing, though highly accurate, involves significant computational resources and is sensitive to lighting conditions, making it a more complex and costly option. Hence, IR sensors provide a reliable, low-maintenance solution that meets the project's needs without unnecessary complexity.

Next, RFID was chosen over ANPR due to its speed, accuracy, and security. RFID systems do not rely on the quality of visual data like ANPR, which can suffer from errors due to poor lighting or damaged license plates. RFID ensures reliable vehicle identification and access control without the complexities associated with image processing.

Moreover, MySQL database was chosen for several reasons. First, it is an open-source platform, meaning there are no licensing costs, which is a big plus for cost-effective development. Additionally, MySQL is very popular in the industry, known for its reliability and scalability, which makes it a perfect choice for storing the data needed for this system. In addition, because of its widespread use, MySQL has a lot of community support and resources, which makes it simpler to debug and expand as needed.

Furthermore, the decision to use a web application was made to enhance user accessibility and convenience. The web application was prioritized because it allows users to easily access parking information, manage their accounts, top up balances, and make reservations from any device with an internet connection. This approach ensures that users can interact with the system seamlessly, without needing to install additional software, which helps to save the phone storage. The web application allows users to register, log in, view real-time parking availability, make reservations, and top up their account balance if needed. This makes the parking experience more convenient and user-friendly.

Lastly, the decision to include the ESP32-CAM in the system was made to enhance security and provide additional visual evidence. The ESP32-CAM will automatically capture images when the IR sensor detects movement at the entrance. By saving the images to Google Drive, it ensures that these records are securely stored and can be easily accessed when necessary, such as for law enforcement use in investigations in cases of illegal activities, such as theft or unauthorized access. The ESP32-CAM is an ideal choice for this purpose due to its compact size, low cost, and built-in Wi-Fi capabilities, which enable seamless image capture and cloud storage integration.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Design Architecture

In this project, a Smart Parking System utilizing IoT sensors and RFID technology is developed. The IoT sensors selected for this system are IR sensors, with a total of eight sensors corresponding to the eight parking slots in the system. Each parking slot is equipped with one IR sensor to detect the presence of a vehicle. The system uses two NodeMCU ESP32 microcontrollers, as illustrated in the overall system block diagram in Figure 3.1. The ESP32 microcontrollers are equipped with built-in WiFi modules, which enable communication with the database over the internet. The WiFi credentials of the hotspot or broadband network are embedded in the code, which will be shown in the “Software” section. For the server and ESP32 to communicate effectively, the laptop running the server must be connected to the same WiFi network as the ESP32.

To set up the server, the XAMPP package was downloaded and installed, the package includes both the MySQL database server and Apache web server. The ESP32 microcontrollers communicate with the database by sending HTTP requests. The Apache web server listens for these requests from the ESP32 and serves the necessary PHP scripts stored within the laptop's Apache server directory. These PHP scripts are executed by the Apache server to process the requests from the ESP32, interact with the MySQL database, and return responses in JSON format. The MySQL database stores information such as user information, RFID scanning timestamps, reservation user and time, and status of each parking slot. There is also a custom-designed web application using HTML, CSS, JavaScript, and PHP scripts. This includes a registration, login, top-up, reservation, and parking status monitoring page, where the user can view the parking information after successfully logging in.

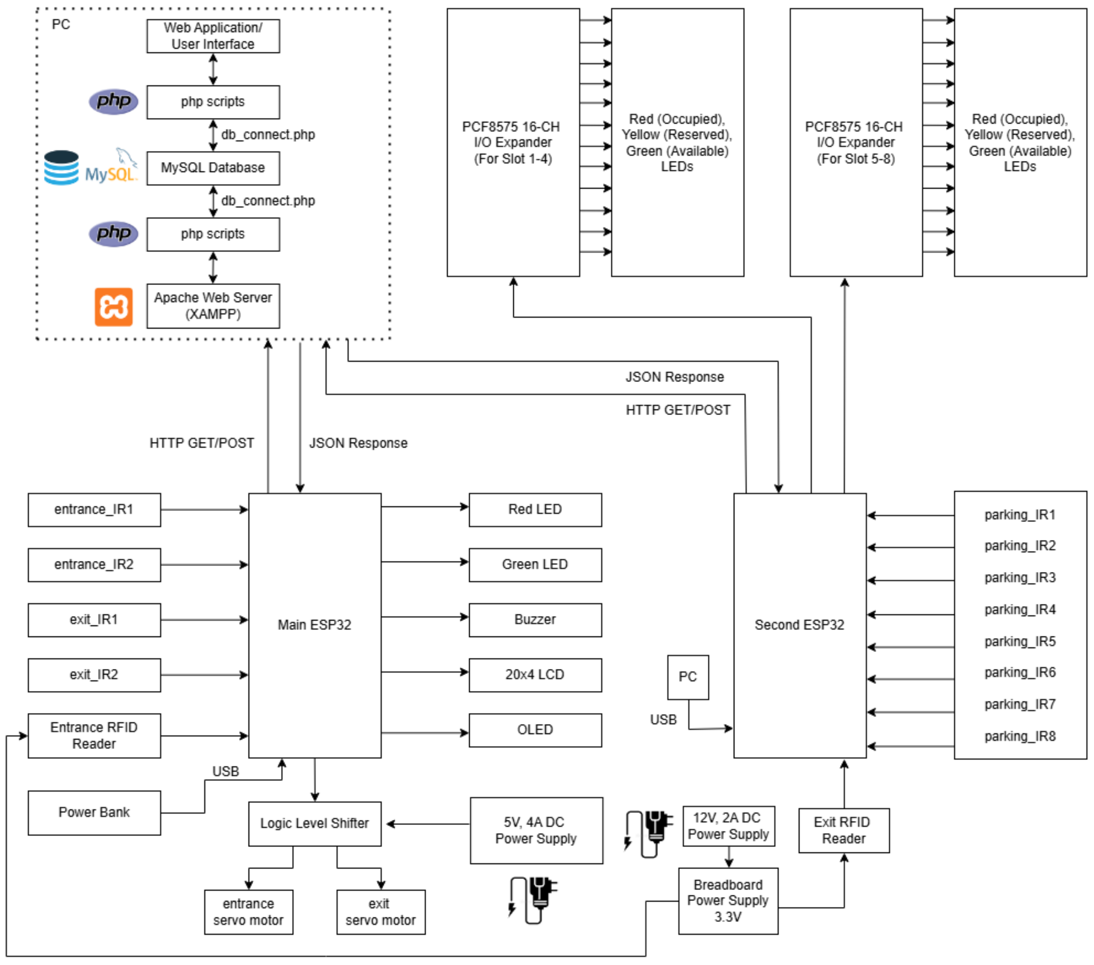


Figure 3.1: The Block Diagram of the Overall IoT RFID Smart Parking System

Next, a power bank with a USB cable is used to power up the main ESP32, while the second ESP32 is powered up using the USB cable that is connected to the PC. The 5 V 4 A DC adapter provides high voltage (HV-5 V) to the logic level shifter. Both ESP32 and the DC adapter would need to connect to common ground. The ESP32 microcontrollers provide 3.3 V to power all connected IR sensors. Besides the eight IR sensors to detect the presence of vehicles in the parking lot, there will be two IR sensors placed at both the entrance and exit gates to detect the presence of vehicles. The working principle of the two entrance IR sensors is illustrated in Figure 3.2. A similar principle applies to the two exit IR sensors. Two servo motors would be used to simulate the entrance and exit gates. The main ESP32 is responsible for handling the input devices like the four IR sensors for entrance and exit gates, the entrance RFID reader, and the output devices like LEDs, buzzer, LCD, OLED, and the entrance and exit servo motors. In contrast, the second ESP32 will handle the eight parking slot IR sensors, the exit RFID reader, and the two PCF8575 I/O expanders.

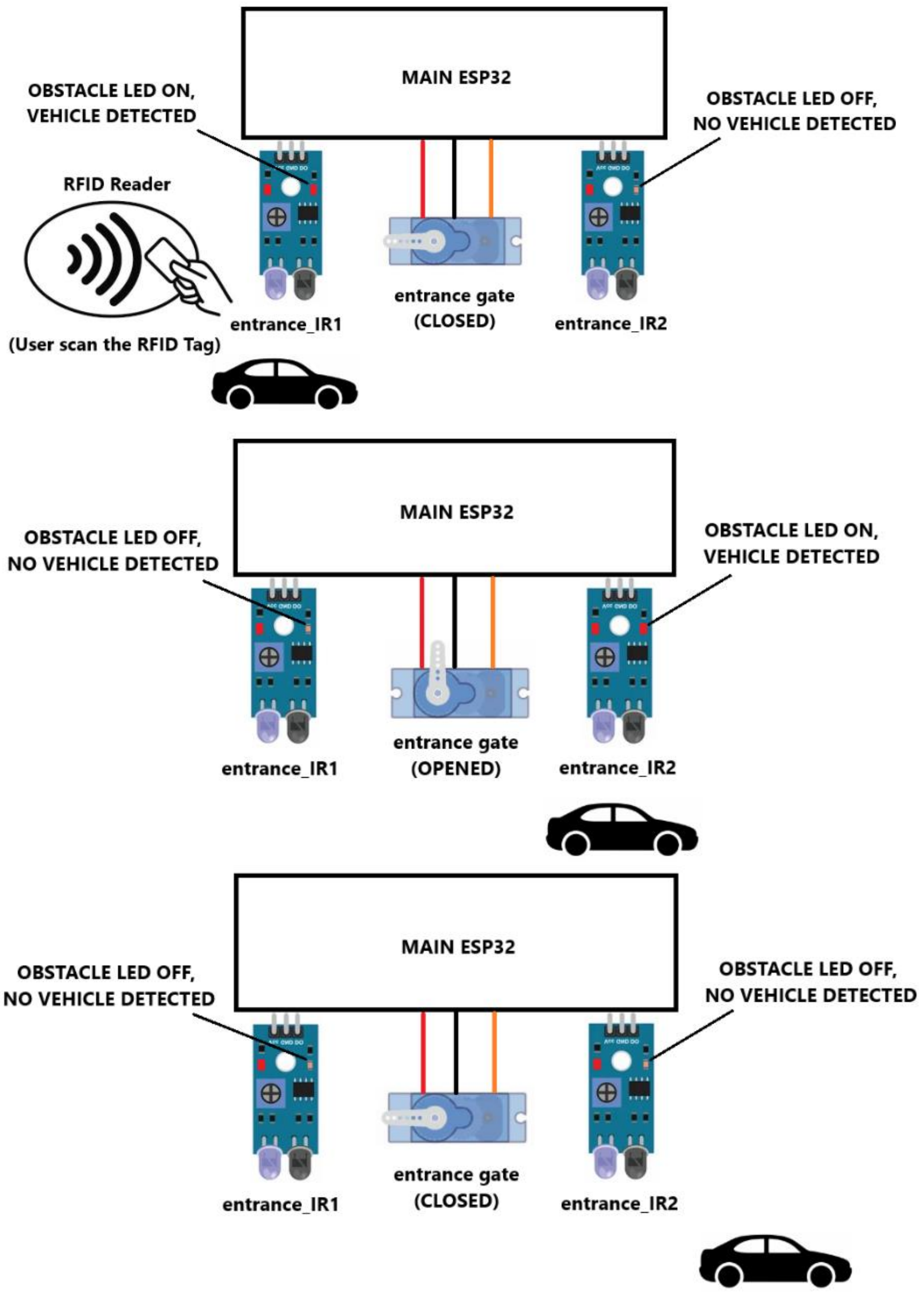

MAIN ESP32
OBSTACLE LED ON, VEHICLE DETECTED
OBSTACLE LED OFF, NO VEHICLE DETECTED
RFID Reader
(User scan the RFID Tag)
entrance_IR1
entrance gate (CLOSED)
entrance_IR2
MAIN ESP32
OBSTACLE LED OFF, NO VEHICLE DETECTED
OBSTACLE LED ON, VEHICLE DETECTED
entrance_IR1
entrance gate (OPENED)
entrance_IR2
MAIN ESP32
OBSTACLE LED OFF, NO VEHICLE DETECTED
OBSTACLE LED OFF, NO VEHICLE DETECTED
entrance_IR1
entrance gate (CLOSED)
entrance_IR2

Figure 3.2: The Working Principle of the Two Entrance IR Sensors

3.2 The Operation of the System

The operation of the system is illustrated in the flowchart as shown in Figure 3.3, Figure 3.4, Figure 3.5 and Figure 3.6, respectively. The operation of the Smart Parking System begins when a vehicle approaches the entrance gate, activating the entrance IR sensors (entrance_IR1 ON, entrance_IR2 OFF). The user scans their registered RFID tag at the entrance RFID reader, then, the Main ESP32 will check if the scanned UID matches the database. If the UID does not match, a buzzer sounds with a long beep and the red LED turns on, indicating access is denied. If the UID matches, the system activates a short beep on the buzzer twice and turns on the green LED, displaying the user's information, such as name, license plate, and latest account balance after parking fee is deducted on the OLED. Then, the system checks if there are available parking lots. If none are available, the LCD displays "Sorry, Parking Full". If parking is available, the system checks whether the scanned UID has enough account balance, if enough, it proceeds to open the entrance gate. Then, the ESP32-CAM will take a picture of the vehicle and save it to Google Drive. As the vehicle passes through, the IR sensors detect the movement of the vehicle, the gate will not close if the vehicle does not pass through the entrance_IR2 sensor. When the vehicle fully passes through, the entrance gate is closed.

Next, the user parks the car, and the Second ESP32 checks for changes in the parking lot status. If a change is detected, the Second ESP32 sends an HTTP POST request to update the database, and the Main ESP32 continuously polls the database by sending HTTP GET requests to the server to keep the LCD updated. The LCD displays "FREE" for available parking slots, "FILL" for occupied parking slots, and "RESV" for reserved parking slots. Additionally, it shows the total number of available parking slots and the current time to ensure that users have real-time information at a glance. Moreover, the users can visit the web application by scanning the provided QR code or directly visiting the URL link. If the user does not have an account, they can register one. The user will need to log in to the registered account in order to see the real-time parking lot status or make reservations.

When the vehicle approaches the exit gate, the exit IR sensors (exit_IR1 ON, exit_IR2 OFF) are activated. The user scans the RFID tag at the exit RFID reader, and the Second ESP32 verifies the UID. The system generates a JSON file containing the user's RFID information, such as UID, car owner name, license plate, and authorization status. The Main ESP32 will then retrieve the information from this JSON file and decide the response of the servo motors (gates), LEDs, and buzzer. If the UID does not match (authorization status is FALSE), the buzzer sounds with a long beep and the red LED turns on, indicating access is denied. If the UID matches (authorization status is TRUE), a buzzer sounds with a short beep, the green LED turns on, and the exit gate opens, indicating access is authorized. As the vehicle exits, the IR sensors detect the movement, and the exit gate is closed, and the process keeps repeating.

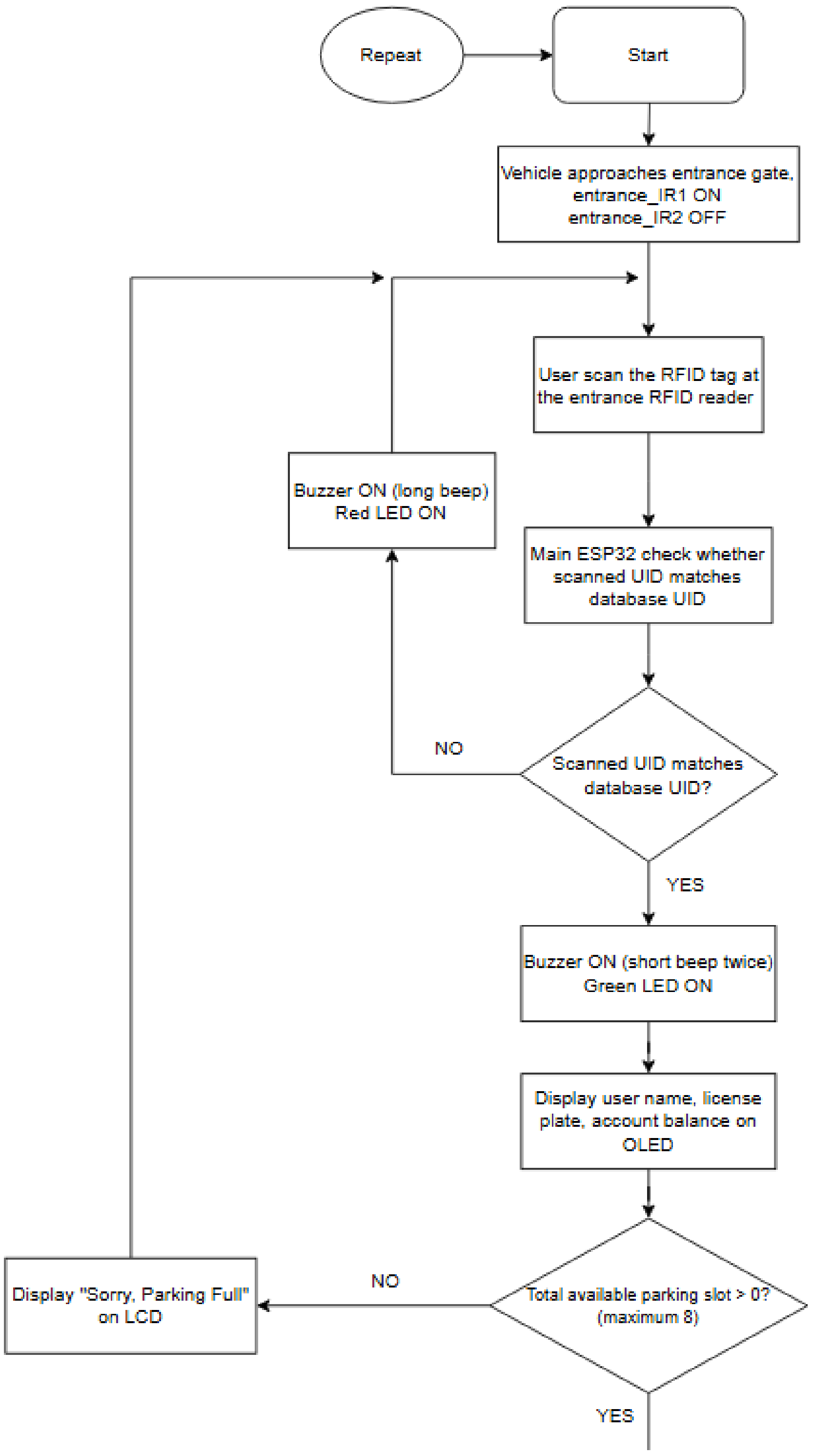

Repeat
Start
Vehicle approaches entrance gate, entrance_IR1 ON entrance_IR2 OFF
User scan the RFID tag at the entrance RFID reader
Buzzer ON (long beep) Red LED ON
Main ESP32 check whether scanned UID matches database UID
NO
Scanned UID matches database UID?
YES
Buzzer ON (short beep twice) Green LED ON
Display user name, license plate, account balance on OLED
Display "Sorry, Parking Full" on LCD
NO
Total available parking slot > 0? (maximum 8)
YES

Figure 3.3: The Flowchart Showing the Operation of the Smart Parking System in General (First Part)

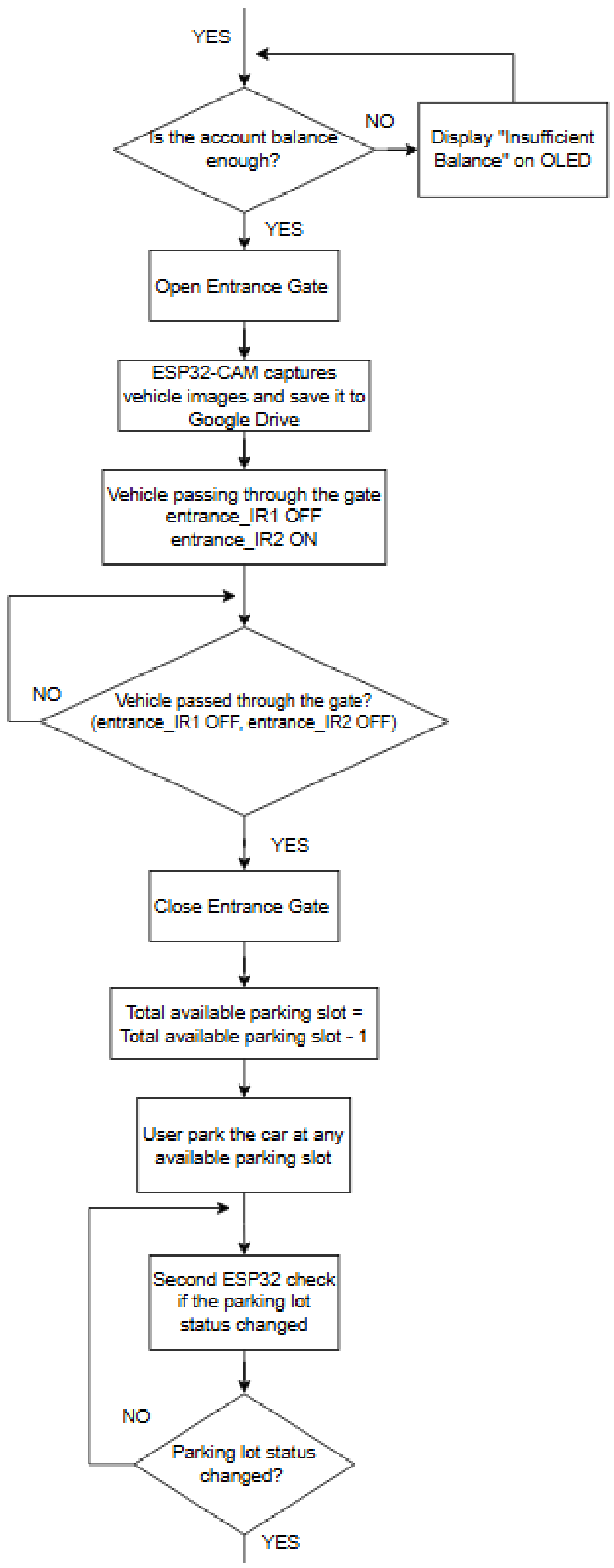

YES
Is the account balance enough?
NO
Display "Insufficient Balance" on OLED
YES
Open Entrance Gate
ESP32-CAM captures vehicle images and save it to Google Drive
Vehicle passing through the gate entrance_IR1 OFF entrance_IR2 ON
NO
Vehicle passed through the gate? (entrance_IR1 OFF, entrance_IR2 OFF)
YES
Close Entrance Gate
Total available parking slot = Total available parking slot - 1
User park the car at any available parking slot
Second ESP32 check if the parking lot status changed
NO
Parking lot status changed?
YES

Figure 3.4: The Flowchart Showing the Operation of the Smart Parking System in General (Second Part)

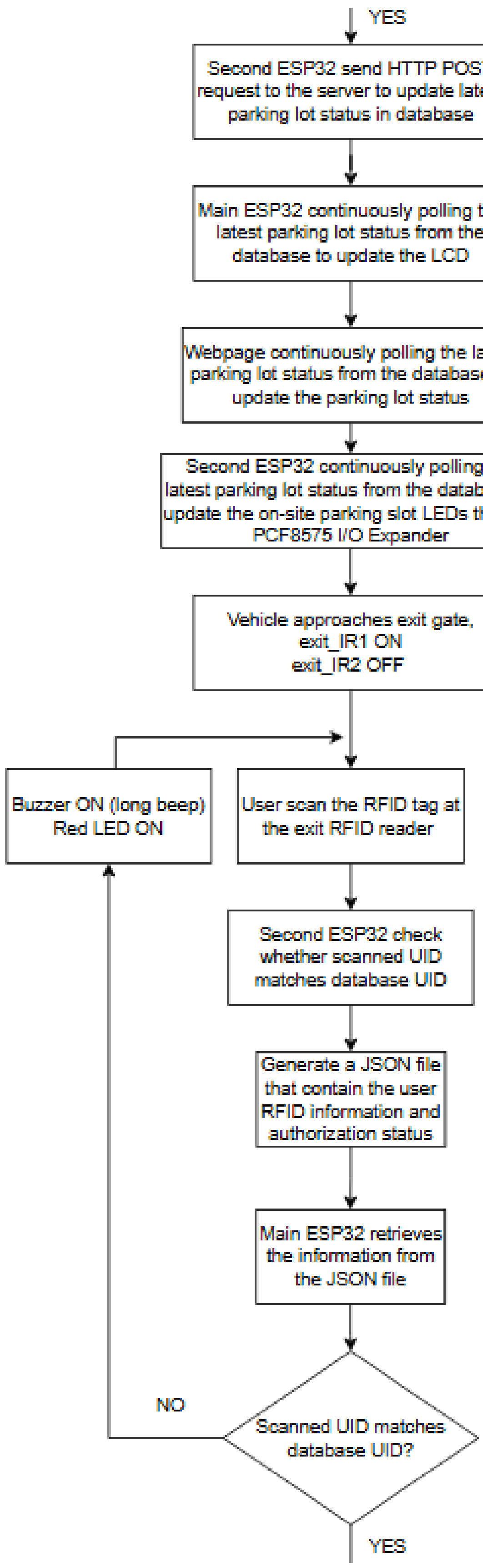

YES
Second ESP32 send HTTP POST request to the server to update latest parking lot status in database
Main ESP32 continuously polling the latest parking lot status from the database to update the LCD
Webpage continuously polling the latest parking lot status from the database to update the parking lot status
Second ESP32 continuously polling the latest parking lot status from the database to update the on-site parking slot LEDs through PCF8575 I/O Expander
Vehicle approaches exit gate, exit_IR1 ON exit_IR2 OFF
Buzzer ON (long beep) Red LED ON
User scan the RFID tag at the exit RFID reader
Second ESP32 check whether scanned UID matches database UID
Generate a JSON file that contain the user RFID information and authorization status
Main ESP32 retrieves the information from the JSON file
NO
Scanned UID matches database UID?
YES

Figure 3.5: The Flowchart Showing the Operation of the Smart Parking System in General (Third Part)

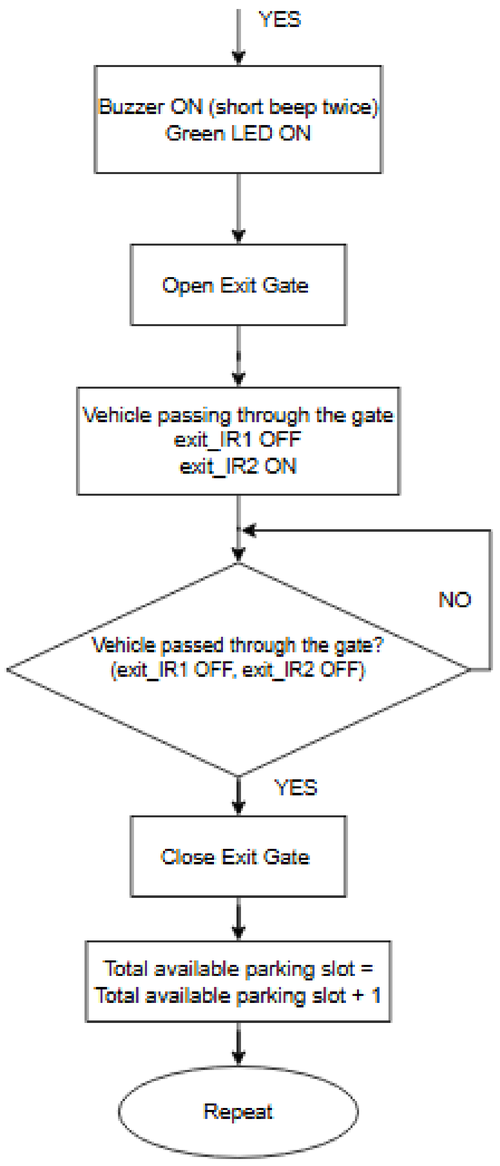

YES
Buzzer ON (short beep twice)
Green LED ON
Open Exit Gate
Vehicle passing through the gate
exit_IR1 OFF
exit_IR2 ON
NO
Vehicle passed through the gate?
(exit_IR1 OFF, exit_IR2 OFF)
YES
Close Exit Gate
Total available parking slot =
Total available parking slot + 1
Repeat

Figure 3.6: The Flowchart Showing the Operation of the Smart Parking System in General (Fourth Part)

Figure 3.7 and Figure 3.8 show the simplified step-by-step process for regular users and reserved users accessing and exiting the parking lot. Although regular users and reserved users may differ in terms of when they visit the parking lot, they follow the same basic process, as shown in the flowchart in Figure 3.3, Figure 3.4, Figure 3.5 and Figure 3.6, respectively.

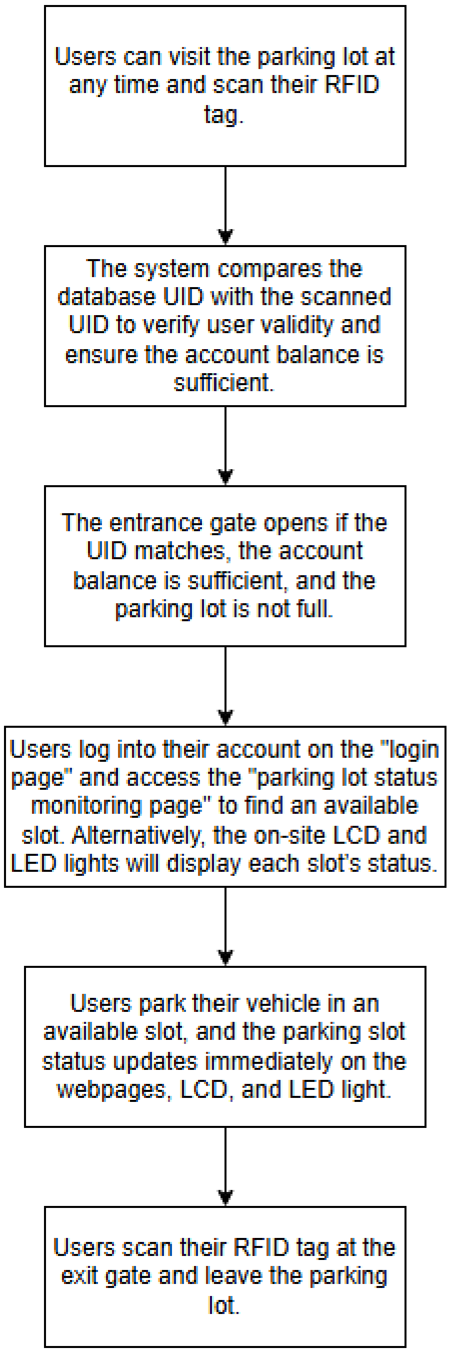

Users can visit the parking lot at any time and scan their RFID tag.
The system compares the database UID with the scanned UID to verify user validity and ensure the account balance is sufficient.
The entrance gate opens if the UID matches, the account balance is sufficient, and the parking lot is not full.
Users log into their account on the "login page" and access the "parking lot status monitoring page" to find an available slot. Alternatively, the on-site LCD and LED lights will display each slot's status.
Users park their vehicle in an available slot, and the parking slot status updates immediately on the webpages, LCD, and LED light.
Users scan their RFID tag at the exit gate and leave the parking lot.

Figure 3.7: Step-by-Step Process for Regular Users Accessing and Exiting the Parking Lot

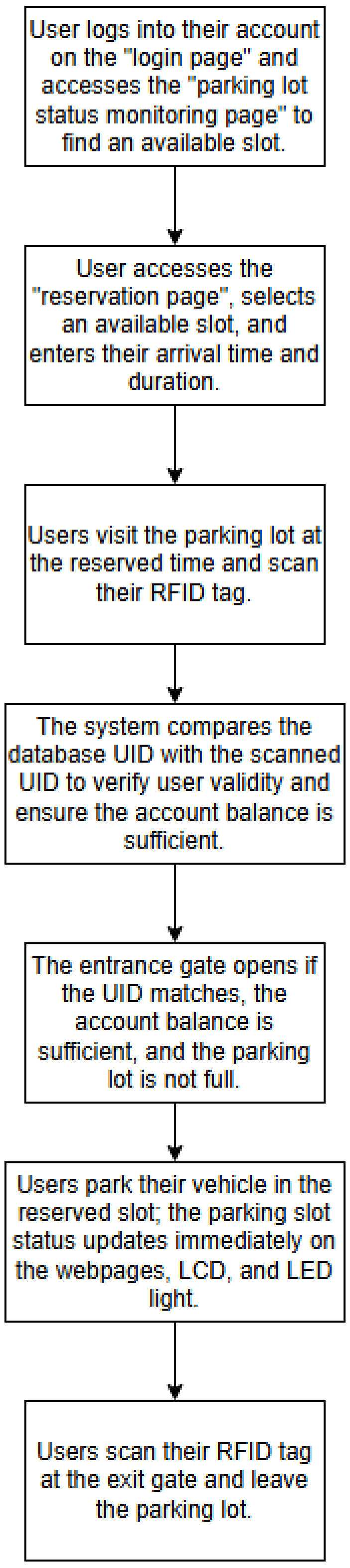

User logs into their account on the "login page" and accesses the "parking lot status monitoring page" to find an available slot.
User accesses the "reservation page", selects an available slot, and enters their arrival time and duration.
Users visit the parking lot at the reserved time and scan their RFID tag.
The system compares the database UID with the scanned UID to verify user validity and ensure the account balance is sufficient.
The entrance gate opens if the UID matches, the account balance is sufficient, and the parking lot is not full.
Users park their vehicle in the reserved slot; the parking slot status updates immediately on the webpages, LCD, and LED light.
Users scan their RFID tag at the exit gate and leave the parking lot.

Figure 3.8: Step-by-Step Process for Reserved Users Accessing and Exiting the Parking Lot

In this Smart Parking System, an ESP32-CAM is set up at the entrance to capture images of vehicles whenever movement is detected by an IR sensor. These captured images can be retrieved to assist law enforcement agencies in tracking individuals involved in illegal activities. Initially, the plan was to store the images on an SD card. However, due to the hardware limitations of the ESP32-CAM, the images saved on the SD card often appear with a purplish tint and poor color balance. On the other hand, when using the same camera settings for streaming or saving the images to Google Drive, the quality is much better.

Therefore, the decision was made to store the images directly on Google Drive. This not only ensures better image quality but also eliminates the hassle of having to unplug the SD card to view the images. With Google Drive, the images can be accessed directly without needing an SD card reader.

The way it operates is when the IR sensor detects vehicles, it triggers the ESP32-CAM to take an image. The image is then encoded in Base64 format and sent through an HTTP POST request to a Google Apps Script. The script decodes the image and saves it to Google Drive. The Base64 encoding library (Base64.h and Base64.cpp) used in this project is credited to Adam Rudd (Farad, 2020). The flowchart in Figure 3.9 shows the operation of the ESP32-CAM at the entrance gate.

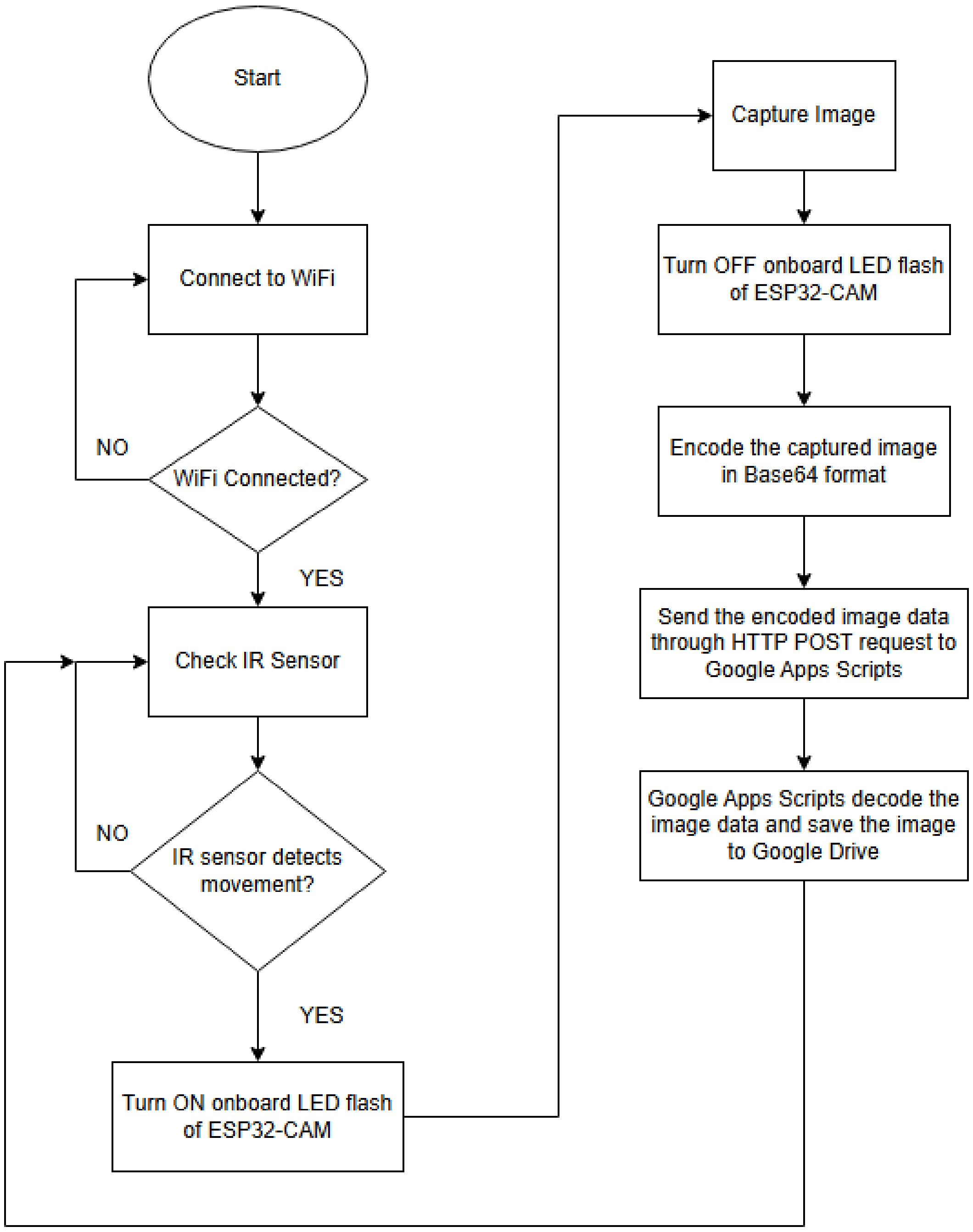

Start
Connect to WiFi
WiFi Connected?
NO
YES
Check IR Sensor
IR sensor detects movement?
NO
YES
Turn ON onboard LED flash of ESP32-CAM
Capture Image
Turn OFF onboard LED flash of ESP32-CAM
Encode the captured image in Base64 format
Send the encoded image data through HTTP POST request to Google Apps Scripts
Google Apps Scripts decode the image data and save the image to Google Drive

Figure 3.9: The Flowchart Showing the Operation of the ESP32-CAM at the Entrance Gate

3.3 Hardware Components

3.3.1 NodeMCU ESP32

The ESP32-DevKitC is a 30-pin development board based on the ESP32 microcontroller, designed by Espressif Systems for embedded and IoT projects. The ESP32 uses an Xtensa® dual-core 32-bit LX6 microprocessor with 448 KB of Flash and 520 KB of SRAM, supporting wireless standards 802.11 b/g/n at up to 150 Mbps (Wilson, 2020). The board is compact, lightweight, operates at 3.3 V, and can be powered via USB. It includes 30 GPIO pins, a PWM controller with 16 channels, and supports SPI, I2C, and UART protocols.

Compared to the Arduino Uno, which lacks built-in wireless connectivity, the ESP32 stands out with its dual-core processing power, higher number of GPIO pins, and has a built-in WiFi chip. Both ESP8266 and ESP32 have a built-in WiFi chip. However, the ESP32 is a dual-core 160 MHz to 240 MHz CPU, while the ESP8266 is a single-core processor that runs at 80 MHz (Wilson, 2020). Unlike the ESP8266, the ESP32 is a more powerful successor, offering additional features such as Bluetooth support, more GPIOs, and better overall performance, but at a slightly higher cost. The ESP32 has advantages like low cost, small size, and open-source nature, combined with its compatibility with the Arduino IDE, making it a highly accessible and scalable solution for developers, providing a seamless transition from Arduino boards while offering much greater functionality. The open-source design ensures easy access to community support and resources. The board can be powered via USB, making it easy to set up and use with the Arduino IDE, especially for developers familiar with Arduino boards.

Figure 3.10 shows the pinout diagram of the ESP32-DevKitC development board. GPIO 21 and 22 are used for I2C communication, connecting the OLED, LCD, and RTC modules. This setup reduces the number of GPIOs needed since these devices can share the same lines as long as they have unique I2C addresses. Most GPIOs on the board can be used for PWM, except for GPIOs 34, 35, 36, and 39. PWM is important because servo motors require PWM-capable GPIOs to function properly.

Lastly, GPIOs 5, 18, 19, and 23 are used for SPI communication, which is used by the RFID readers.

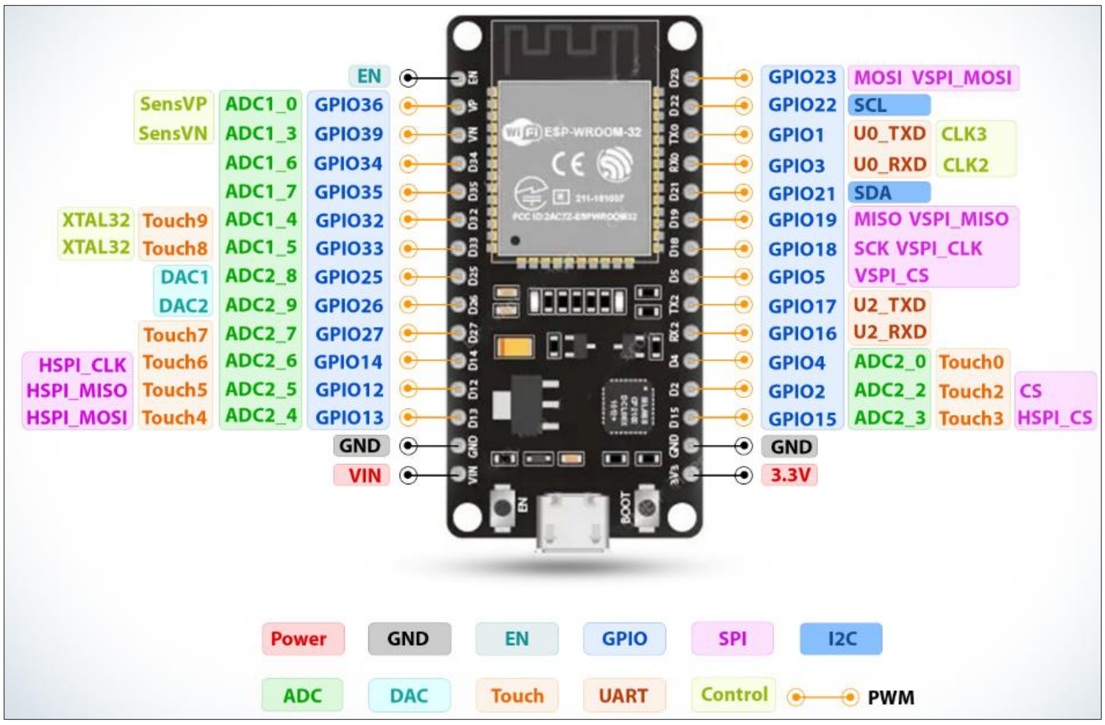


Figure 3.10: Pinout Diagram of the ESP32-DevKitC Development Board (30 GPIOs) (Wilson, 2020)

3.3.2 Infrared (IR) Sensors

An IR sensor is a simple electronic device that uses infrared light to detect objects nearby. It works by emitting infrared light through an IR LED (Transmitter), and when this light hits an object such as a car, it reflects back to an IR photodiode (Receiver). The amount of light received by the photodiode changes its resistance and voltage, which is how the sensor knows something is there. The sensor also includes an onboard power LED that lights up when the sensor is powered on. Additionally, there is an obstacle LED that illuminates when the sensor detects an object, such as a car, within its detection range. The sensitivity of the IR sensor can be adjusted using the onboard potentiometer (Ch'ng, 2019). By turning the potentiometer knob clockwise, the detection range can be increased, and vice versa for counterclockwise.

The IR sensors used in this project operate at 3.3 V, and they have three pins, which are V_{CC}, GND, and OUT pins. When the IR sensor detects a car, it provides a ‘0’ as an input to the ESP32, indicating that a car is present. A ‘1’ as an input would indicate that no car is detected. Figure 3.11 shows the diagram of the IR sensor, while Figure 3.12 shows its working principle.

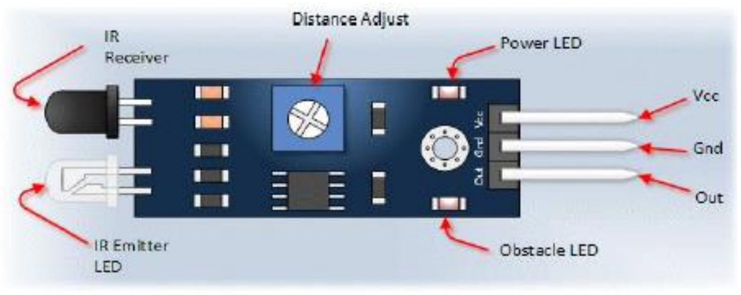


Figure 3.11: The Diagram of an Infrared (IR) Sensor (Ch'ng, 2019)

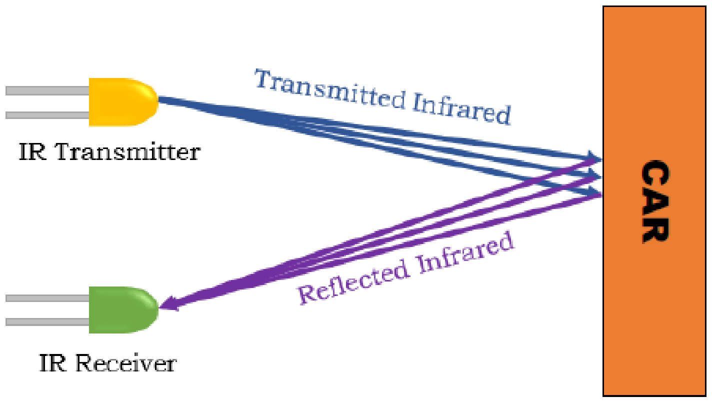


Figure 3.12: The Working Principle of an IR Sensor (Ch'ng, 2019)

3.3.3 MFRC522 RFID Reader

Radio Frequency Identification (RFID) is a wireless system that identifies objects by transmitting data through radio wave signals. It consists of two main components, which are an RFID reader and an RFID tag. An electromagnetic field is generated by the RFID reader to power the tag, which typically has no battery (Last Minute Engineers, 2018). The tag contains a microchip that stores data, such as a unique identifier (UID). When the tag is within the range of the reader, it sends its UID back to the reader. The reader then interprets this data and transfers it to a connected system for further processing. This technology allows for the efficient and automated identification of objects without the need for manual intervention. Figure 3.13 shows the working principle of RFID.

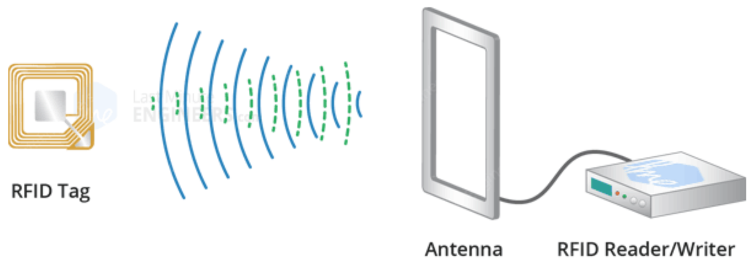


Figure 3.13: The Working Principle of RFID (Last Minute Engineers, 2018)

In this project, MFRC522 is the specific RFID reader module used to interact with MIFARE 1K RFID tags. The MFRC522 operates at 13.56 MHz and is based on the NXP MFRC522 IC. It has eight pins in total, which are V_{CC}, GND, MISO (Master In Slave Out), SS/SDA (Slave Select), SCK (Serial Clock), MOSI (Master Out Slave In), RST (Reset), and IRQ (Interrupt). The IRQ pin is usually left disconnected. It can support communication using I2C, UART or SPI protocol. SPI protocol is used in this project due to its faster speed. In the code, the SS and RST pins need to be specified. The SS pin allows the microcontroller to identify which RFID reader it is communicating with. The reader module is responsible for generating the electromagnetic field that powers passive RFID tags, reading their unique identifier (UID), and sending this data to the microcontroller for processing. The read range is about 5 cm. Figure 3.14 shows the pinout diagram of the MFRC522 RFID Reader module, while Figure 3.15 shows the MIFARE 1K 13.56 MHz RFID Tags.

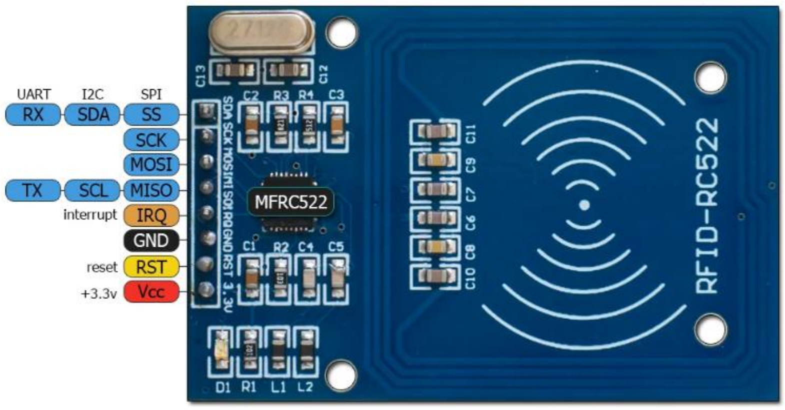


Figure 3.14: Pinout Diagram of MFRC522 RFID Reader Module

Figure 3.15: MIFARE 1K 13.56 MHz RFID Tags

3.3.4 SG90 Servo Motor

The SG90 micro servo motor is an affordable, compact, and lightweight servo motor widely used in robotics and embedded systems. In this project, two SG90 servo motors are used for controlling the entrance and exit gates in the smart parking system. It offers a high level of precision with a maximum 180° angle rotation (Ch'ng, 2019). Therefore, it has the sufficient angle for this project as the opening and closing of the gate only require 90°. It operates on PWM signals, which means it needs to be connected to a PWM-capable GPIO pin as specified earlier. The SG90 runs on an operating voltage of 4.8 to 5 V DC. Since the GPIO pins of ESP32 output only 3.3 V, a four-channel 3.3 V to 5 V logic level shifter is used in this project to ensure the servo motor receives the correct operating voltage, allowing it to function properly within the system. Figure 3.16 shows the SG90 servo motor, while Figure 3.17 shows the 4-channel 3.3 V to 5 V logic level shifter. The 'LV' pin of the level shifter is connected to 3.3 V, 'HV' pin is connected to 5 V, while the PWM-capable GPIO pin of the ESP32 is connected to either of the low voltage channel and the PWM pin of SG90 servo motor is connected to the corresponding high voltage channel.

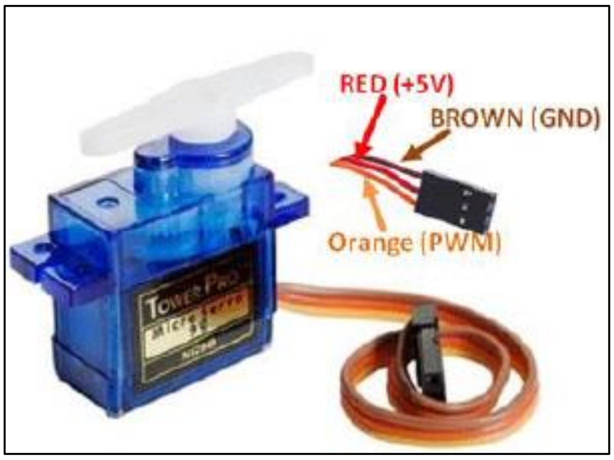


Figure 3.16: SG90 Servo Motor (Ch'ng, 2019)

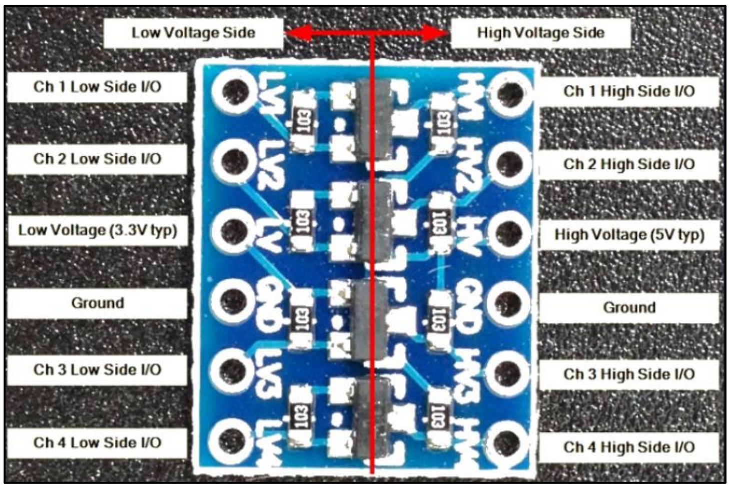


Figure 3.17: 4-Channel 3.3 V to 5 V Logic Level Shifter

3.3.5 Organic Light-Emitting Diode (OLED) Display

The 0.96-inch OLED display as shown in Figure 3.18 is a small, low-power consumption screen with a resolution of 128 × 64 pixels, which is perfect for projects with limited space. In this project, the OLED has a white display (font) and is connected to the ESP32 microcontroller using the I2C protocol. It is used to display RFID scanning messages like the scanning status and the owner's name, license plate, and account balance. The default address found after running the I2C scanner code is 0 × 3C. It is important to ensure no conflict of address between different I2C devices, otherwise it will not work. This OLED does not require a backlight, allowing for deep blacks and high contrast, making it ideal for clear text and simple graphics.

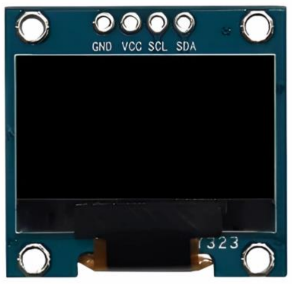


Figure 3.18: 0.96-inch OLED Display

3.3.6 Liquid Crystal Display (LCD)

In this project, the LCD is used to display the real-time clock and the parking lot status to the users. Figure 3.19 shows the 20 × 4 I2C LCD. It has 20 characters per line across 4 lines, therefore, more information could be displayed. The I2C module significantly simplifies the wiring and minimizes the number of GPIO pins required to control the display, requiring only two pins (SDA and SCL) for communication with the ESP32. The default I2C address is 0 × 27. The I2C address can be changed by connecting or disconnecting the A0, A1, and A2 solder jumpers on the I2C module. Without this I2C module, at least 7 GPIO pins would be needed for communication with the ESP32. The contrast of the backlight could be adjusted by turning the potentiometer knob on the I2C module.

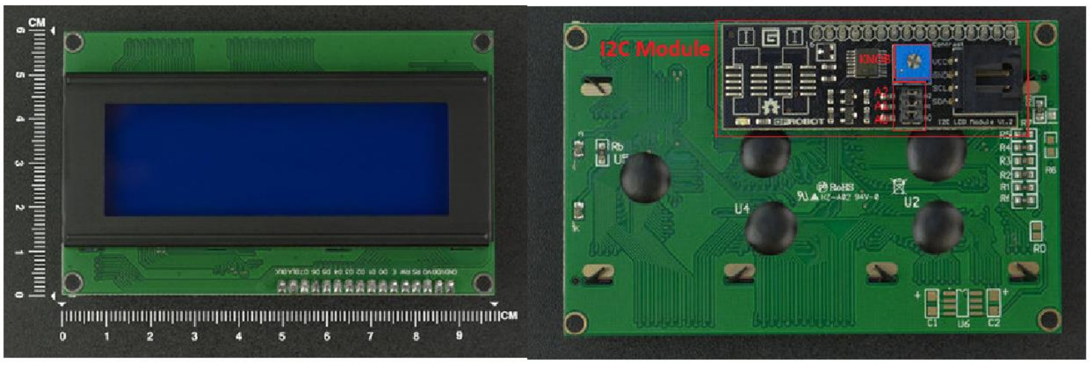


Figure 3.19: 20 × 4 I2C LCD

3.3.7 AI-Thinker ESP32-CAM

The ESP32-CAM is a small and affordable camera module that includes the ESP32-S chip. It comes with the OV2640 camera, a few GPIOs for attaching peripherals, and a microSD card slot for storing images (Santos, 2019). Figure 3.20 shows the ESP32-CAM, whereas Figure 3.21 shows the pinout diagram. In this project, the ESP32-CAM is used to take images of vehicles entering the parking lot whenever the IR sensor detects motion. These images are saved to Google Drive, creating a record that can be used by law enforcement if there is any criminal activity. Furthermore, the ESP32-CAM supports live video streaming.

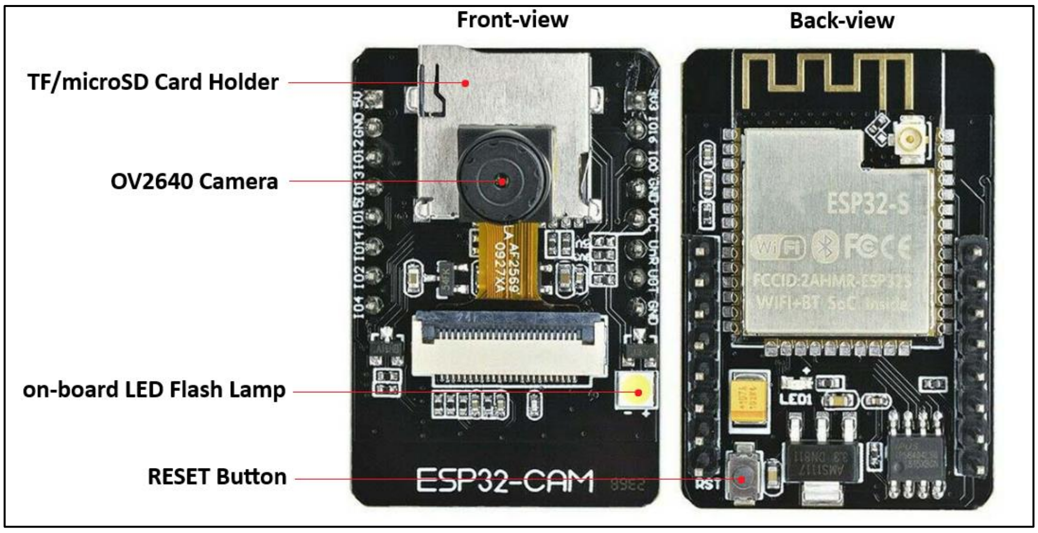


Figure 3.20: AI-Thinker ESP32-CAM

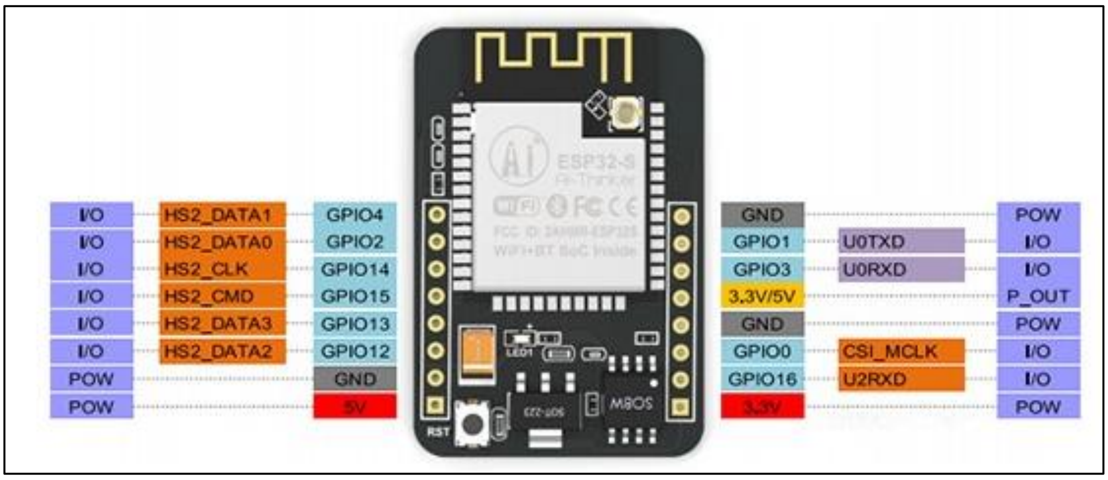


Figure 3.21: AI-Thinker ESP32-CAM Pinout Diagram (Santos, 2019)

However, the ESP32-CAM comes with some challenges, such as its limited number of GPIO pins compared to a standard ESP32, which can restrict the number of peripherals that can be connected. Besides that, programming the ESP32-CAM requires an USB to TTL adapter (CH340 is used in this project), as shown in Figure 3.22. To enter flashing mode, GPIO 0 needs to be connected to GND (Santos, 2019). GPIO 1 and 3 are used to program the ESP32-CAM. The microSD card reader is connected internally to GPIO 14 (CLK) and GPIO 15 (CMD). The CMD pin sends commands to the SD card, while the CLK pin provides the timing signal to synchronize data transfer between the microcontroller and the SD card. By default, the ESP32-CAM connects to the SD card in 4-bit mode, which uses GPIOs 2, 4, 12, and 13 for data transfer. However, switching to 1-bit mode allows the use of only GPIO 2 for data transfer, which frees up GPIO 12 and 13 for input or output (Mountain, 2022). While GPIO 4 is also freed, it is connected to the onboard LED, so it is best left unconnected. This adjustment is essential when additional GPIOs are needed for other components or sensors in the project. Despite these challenges, the ESP32-CAM remains an excellent choice, offering image capture capability at a very affordable price.

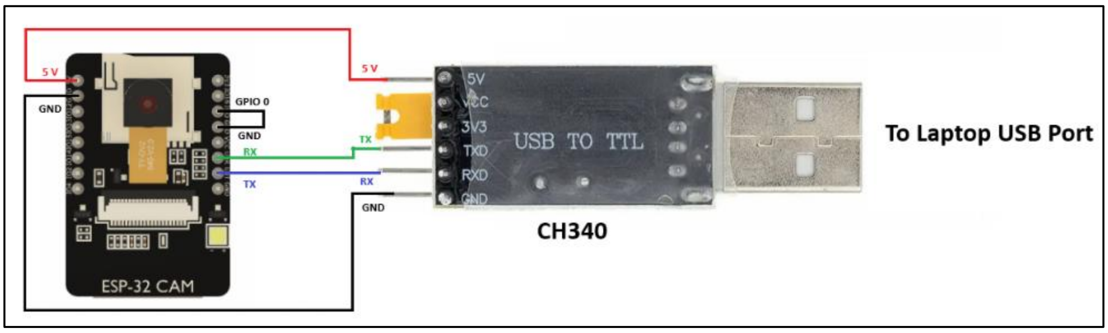


Figure 3.22: Hardware Connection to Program ESP32-CAM Using CH340

3.3.8 PCF8575 16-bit I/O Expander

The PCF8575 is a 16-bit I/O expander designed for 2.5 V to 5.5 V V_{CC} operation (Mischianti, 2019). It uses the I2C protocol for communication with microcontrollers and provides remote I/O expansion with 16 quasi-bidirectional input/output pins (P07–P00, P17–P10) that can be used for tasks like controlling LEDs (Mischianti, 2019). These pins feature latched outputs with high-current drive capability and operate in an active-low configuration, meaning the LED will light up when the pin is set to LOW (Mischianti, 2019). Figure 3.23 shows the top view and bottom view of the PCF8575. The default I2C address of the PCF8575 is 0 × 20. The I2C address can be changed by connecting the address pins (A0, A1, A2) to either V_{DD} or GND.

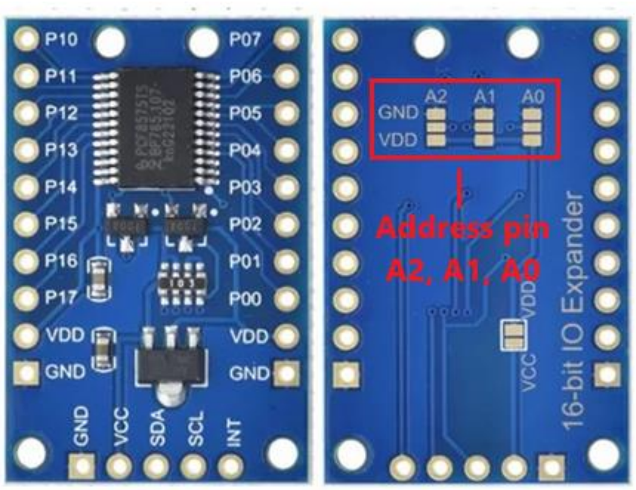


Figure 3.23: Top View and Bottom View of PCF8575 16-bit I/O Expander

Next, two PCF8575 I/O expanders are used on the second ESP32 in this project, as there are twenty-four LEDs representing the status of parking slots (three LEDs per slot), and the GPIOs provided by the ESP32 are insufficient. The LEDs are connected in a current-sink configuration due to the open-drain behavior of the PCF8575. Additionally, the I2C address of the first PCF8575 is set to 0 × 20, and it controls the first twelve LEDs for slots 1 to 4. The I2C address of the second PCF8575 is set to 0 × 24, and it controls the remaining twelve LEDs for slots 5 to 8.

3.4 Software Components

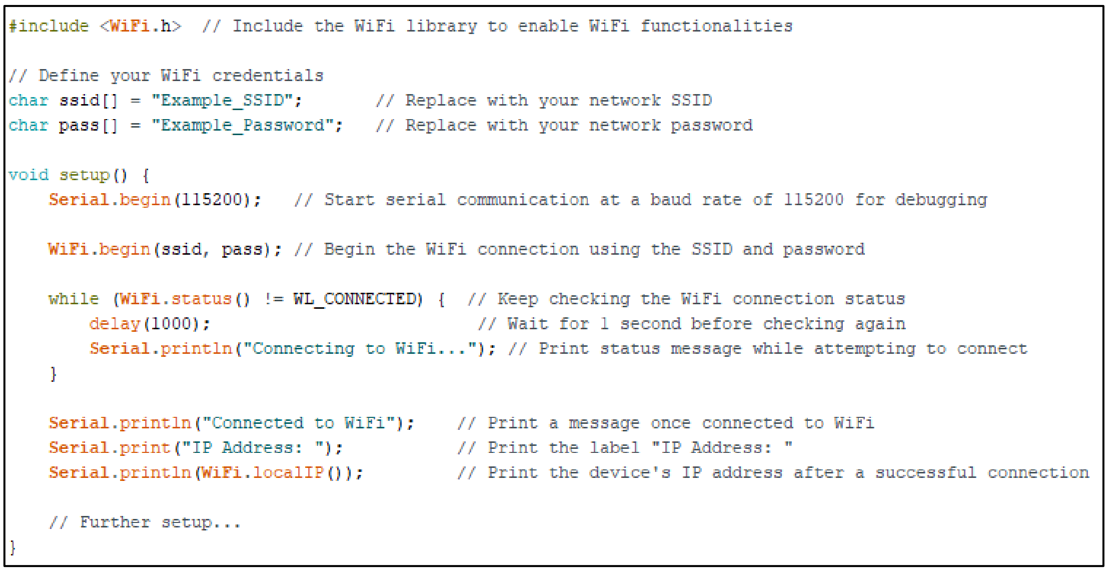
3.4.1 Arduino IDE

The Arduino IDE is a user-friendly, open-source platform designed for programming microcontrollers like the ESP32. It allows for writing, compiling, and uploading code to the microcontroller. In programming with the Arduino IDE, the typical code structure begins with including the necessary libraries, such as WiFi, RFID, display, and servo motor. Following this, functions are declared, GPIO pins are defined, and instances of objects like servo motors or displays are created. Then, global variables are set up to manage important data.

In the setup() function, hardware components are initialized, and a WiFi connection is established, it will only run once. The loop() function includes the main part of the program and executes continuously, managing tasks like reading sensors, controlling outputs, and communicating with other devices or servers.

3.4.1.1 WiFi Setup Function

The code snippet in Figure 3.24 is written to connect the ESP32 to a WiFi network, which is essential for enabling communication with the Apache web server and the MySQL database in the project. It starts by including the “WiFi.h” library. Next, the WiFi credentials are defined. In the setup() function, serial communication is initialized at a baud rate of 115200, which is helpful for debugging through the serial monitor. The WiFi.begin() function is then called to start the connection process using the provided SSID and password. Then, the code enters a loop where it continuously checks if the ESP32 has connected to the WiFi network, and prints "Connecting to WiFi..." every second until the connection is successful. Once connected, a confirmation message "Connected to WiFi" is displayed, and the ESP32’s IP address is printed. The IP address of the laptop can be obtained by running the “ipconfig” command in the Windows Command Prompt, which can be accessed by pressing the Windows key and typing “cmd”. This laptop’s IP address is important for communicating with the server, accessing the database, and it is a part of the URL link when accessing the web application. After this setup, the ESP32 is ready to interact with the server, sending and receiving data as needed for the smart parking system. This code needs to be included in both the code of the main ESP32 and the second ESP32. For the server and ESP32 to communicate effectively, the laptop running the server must be connected to the same WiFi network as the ESP32.



```
#include <WiFi.h>  // Include the WiFi library to enable WiFi functionalities

// Define your WiFi credentials
char ssid[] = "Example_SSID";       // Replace with your network SSID
char pass[] = "Example_Password";   // Replace with your network password

void setup() {
    Serial.begin(115200);   // Start serial communication at a baud rate of 115200 for debugging

    WiFi.begin(ssid, pass); // Begin the WiFi connection using the SSID and password

    while (WiFi.status() != WL_CONNECTED) {  // Keep checking the WiFi connection status
        delay(1000);                          // Wait for 1 second before checking again
        Serial.println("Connecting to WiFi..."); // Print status message while attempting to connect
    }

    Serial.println("Connected to WiFi");    // Print a message once connected to WiFi
    Serial.print("IP Address: ");           // Print the label "IP Address: "
    Serial.println(WiFi.localIP());         // Print the device's IP address after a successful connection

    // Further setup...
}
```

Figure 3.24: WiFi Setup Code for Main ESP32 and Second ESP32

3.4.1.2 Defining Main ESP32 and Second ESP32 Pins

In this study, the main and second ESP32 are used. The summary of pins defined for ESP32 connected components are shown in Table 3.1.

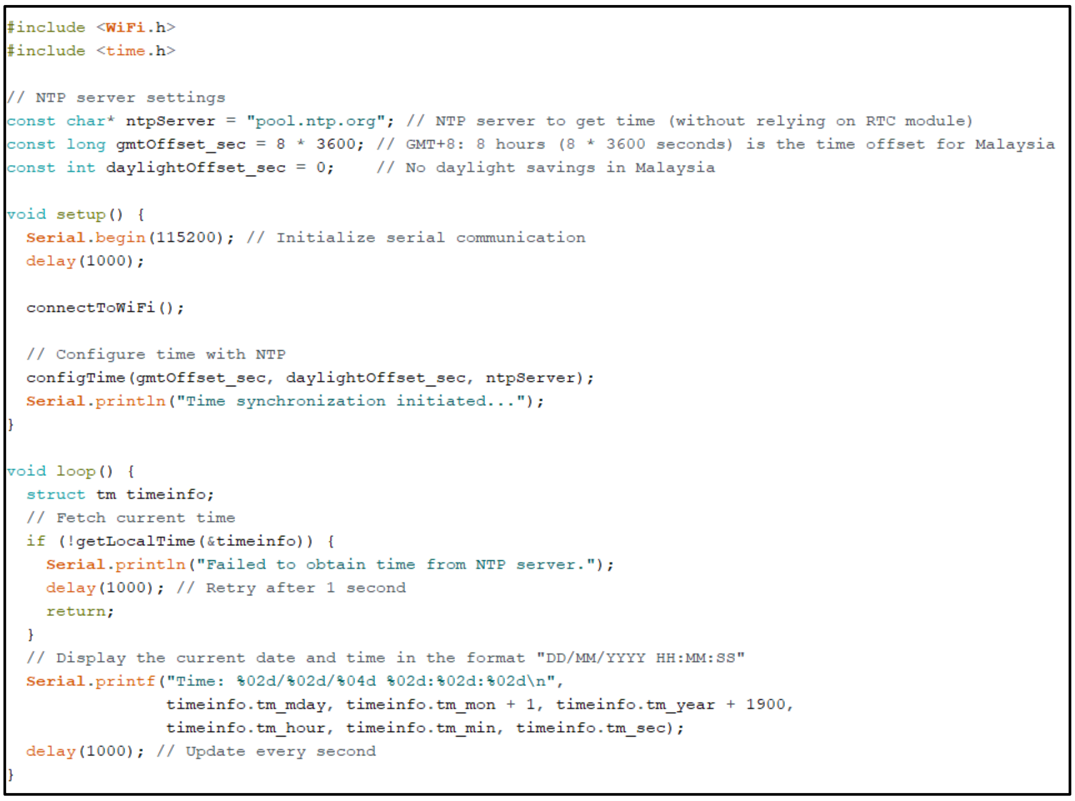
Table 3.1: Summary of Pins Defined for Main ESP32 and Second ESP32 Connected Components

Electronic Devices	**Pin Attached**	
	Main ESP32	**Second ESP32**
Entrance MFRC522 RFID Reader	5 (SS), 27 (RST)	
Buzzer	4	
Green LED	13	
Red LED	12	
entrance_IR1	15	
entrance_IR2	36	
exit_IR1	34	
exit_IR2	39	
entrance_gate_SERVO	14	
exit_gate_SERVO	32	
OLED, LCD	21 (SDA), 22 (SCL)	
Exit MFRC522 RFID Reader		5 (SS), 27 (RST)
parking_IR1		35
parking_IR2		32
parking_IR3		33
parking_IR4		25
parking_IR5		26
parking_IR6		14
parking_IR7		12
parking_IR8		13
PCF8575 I/O Expanders		21 (SDA), 22 (SCL)

3.4.2 Network Time Protocol (NTP) Server

In this project, an NTP server is used to provide accurate and synchronized time for the system. NTP is an internet protocol used to synchronize the system clock by obtaining accurate time from an NTP server. The server provides time in UTC, which is a global standard (Last Minute Engineers, 2019). The system then adjusts this UTC time based on its local timezone setting (GMT+8 for Malaysia), converting it to the correct local time for the region. The code snippet in Figure 3.25 syncs the ESP32's time with an NTP server and displays the current date and time.

The system originally used an RTC module (DS1304), which caused small delays. While a few seconds would not be a problem, over time these delays added up, causing the time to drift by several minutes, which became an issue. By switching to an NTP server, the system can receive real-time, internet-based time, which ensures accurate time on the LCD and precise timestamps for RFID scans on OLED. This change also eliminates the need for battery-powered hardware and guarantees the system always has the correct time, as long as an internet connection is available.



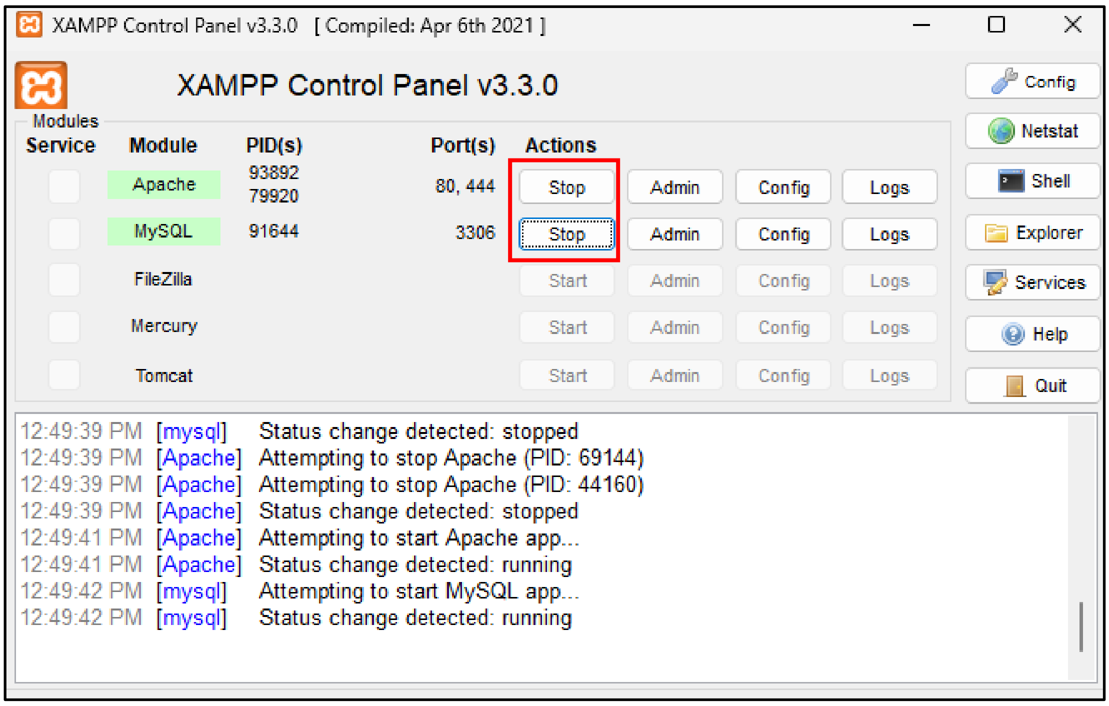
```
#include <WiFi.h>
#include <time.h>

// NTP server settings
const char* ntpServer = "pool.ntp.org"; // NTP server to get time (without relying on RTC module)
const long gmtOffset_sec = 8 * 3600; // GMT+8: 8 hours (8 * 3600 seconds) is the time offset for Malaysia
const int daylightOffset_sec = 0;    // No daylight savings in Malaysia

void setup() {
  Serial.begin(115200); // Initialize serial communication
  delay(1000);

  connectToWiFi();

  // Configure time with NTP
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  Serial.println("Time synchronization initiated...");
}

void loop() {
  struct tm timeinfo;
  // Fetch current time
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to obtain time from NTP server.");
    delay(1000); // Retry after 1 second
    return;
  }
  // Display the current date and time in the format "DD/MM/YYYY HH:MM:SS"
  Serial.printf("Time: %02d/%02d/%04d %02d:%02d:%02d\n",
              timeinfo.tm_mday, timeinfo.tm_mon + 1, timeinfo.tm_year + 1900,
              timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec);
  delay(1000); // Update every second
}
```

Figure 3.25: NTP Server Setup Code to Print the Date and Time

3.4.3 XAMPP Package

In this project, XAMPP is used as the local server environment. The XAMPP package contains Apache (web server), MySQL (database server), and phpMyAdmin (a tool for managing the database). To get the system running, both Apache and MySQL must be manually started from the XAMPP control panel, which is shown in Figure 3.26. If these services are not running, the web pages and database will not be accessible.

The ESP32 microcontrollers collect data from sensors, such as parking slot availability, and send HTTP requests to the Apache web server. The Apache web server listens for these requests, processes them, and serves the appropriate PHP scripts. These PHP scripts communicate with the MySQL database to retrieve and update data, such as parking statuses, and user information. The results are then returned in JSON format to be used by the ESP32 or displayed on the web interface. The MySQL database, accessed through phpMyAdmin, stores all system data, including parking lot status, RFID user details, RFID scan logs, and reservation details.



XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]

XAMPP Control Panel v3.3.0

Service	Module	PID(s)	Port(s)	Actions			
	Apache	93892 79920	80, 444	Stop	Admin	Config	Logs
	MySQL	91644	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
12:49:39 PM  [mysql]   Status change detected: stopped
12:49:39 PM  [Apache]  Attempting to stop Apache (PID: 69144)
12:49:39 PM  [Apache]  Attempting to stop Apache (PID: 44160)
12:49:39 PM  [Apache]  Status change detected: stopped
12:49:41 PM  [Apache]  Attempting to start Apache app...
12:49:41 PM  [Apache]  Status change detected: running
12:49:42 PM  [mysql]   Attempting to start MySQL app...
12:49:42 PM  [mysql]   Status change detected: running
```

Figure 3.26: Starting the Server Using XAMPP Control Panel

3.4.4 MySQL Database

MySQL (phpMyAdmin) is the open-source database used in this project and it is included in the XAMPP package together with the Apache Web Server. In this project, the name of the database is **‘parking_system’**, which contains four tables, which are **‘parking_status’**, **‘reservations’**, **‘rfid_scan_log’**, and **‘rfid_user_info’**, as shown in Figure 3.27.

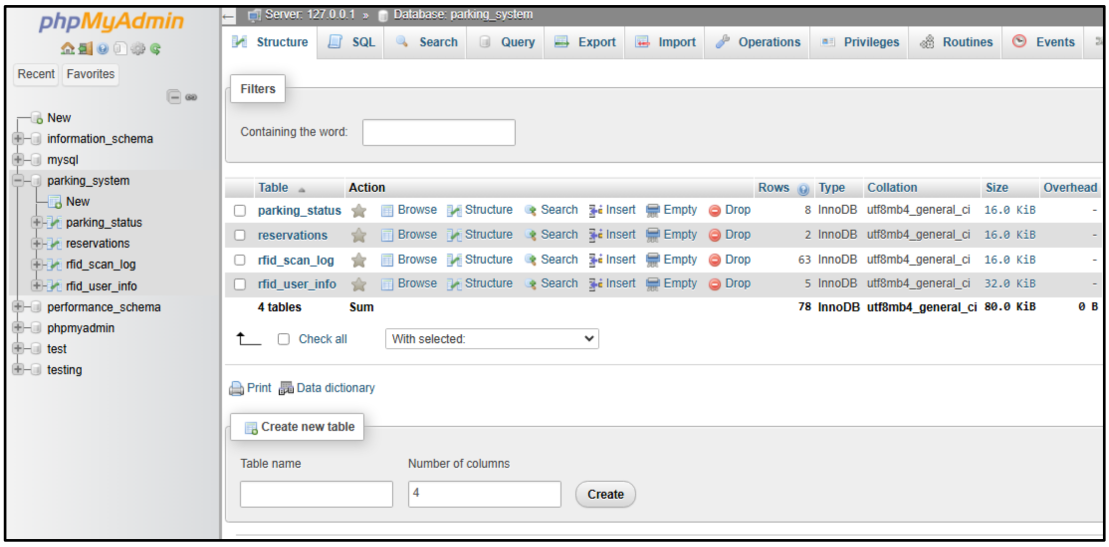
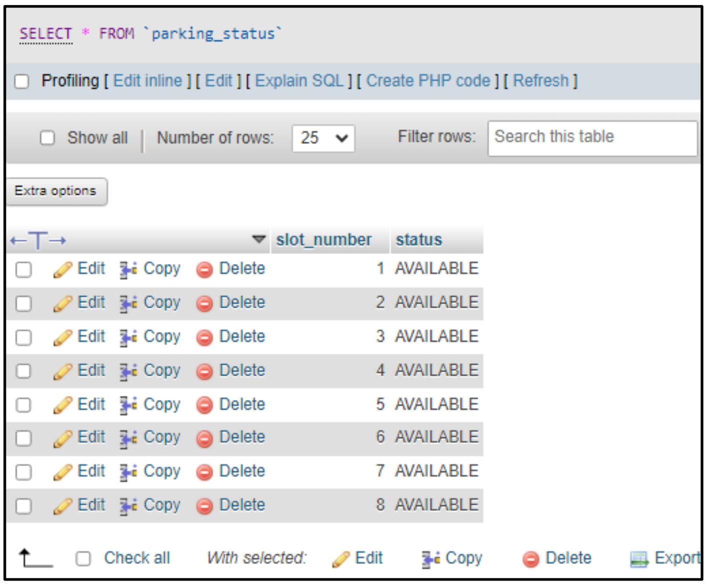


Table	Action	Rows	Type	Collation	Size	Overhead
parking_status	Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	16.0 KiB	-
reservations	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_general_ci	16.0 KiB	-
rfid_scan_log	Browse Structure Search Insert Empty Drop	63	InnoDB	utf8mb4_general_ci	16.0 KiB	-
rfid_user_info	Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_general_ci	32.0 KiB	-
4 tables	**Sum**	**78**	**InnoDB**	**utf8mb4_general_ci**	**80.0 KiB**	**0 B**

Figure 3.27: The ‘parking_system’ Database with Four Tables, Which are ‘parking_status’, ‘rfid_scan_log’, ‘rfid_user_info’, and ‘reservations’

Next, the **‘parking_status’** table stores the latest parking lot status, which is illustrated in Figure 3.28. The second ESP32 writes latest updates of the parking lot status to this table, while the main ESP32, the web application, and the two PCF8575 I/O expanders continuously poll it to refresh the LCD display, the web application, and the parking slot LEDs with the current status. Besides that, the **‘rfid_user_info’** table stores information about each RFID tag, including the tag UID, car color, license plate, owner name, account balance, account ID, password, and telephone number, which is illustrated in Figure 3.29. When a user scans their RFID tag, the ESP32 sends an HTTP GET request to the server to verify the tag UID. If the UID exists, the server responds with the relevant information in JSON format, which is then processed by the ESP32. This table is also where the account balance is updated whenever a tag is scanned or when a new account is registered. Moreover, the **'rfid_scan_log'** table records each RFID scan, such as whether the scan was successful or failed, along with the timestamp of the scan, which is illustrated in Figure 3.30. This allows for tracking the history of scans and ensuring the integrity of the parking system. Lastly, the **'reservations'** table stores the reservation details such as the user name, reserved slot, and reservation time (arrival time), which is illustrated in Figure 3.31. This table will be updated whenever a user reserves a slot through the reservation page.



```
SELECT * FROM `parking_status`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

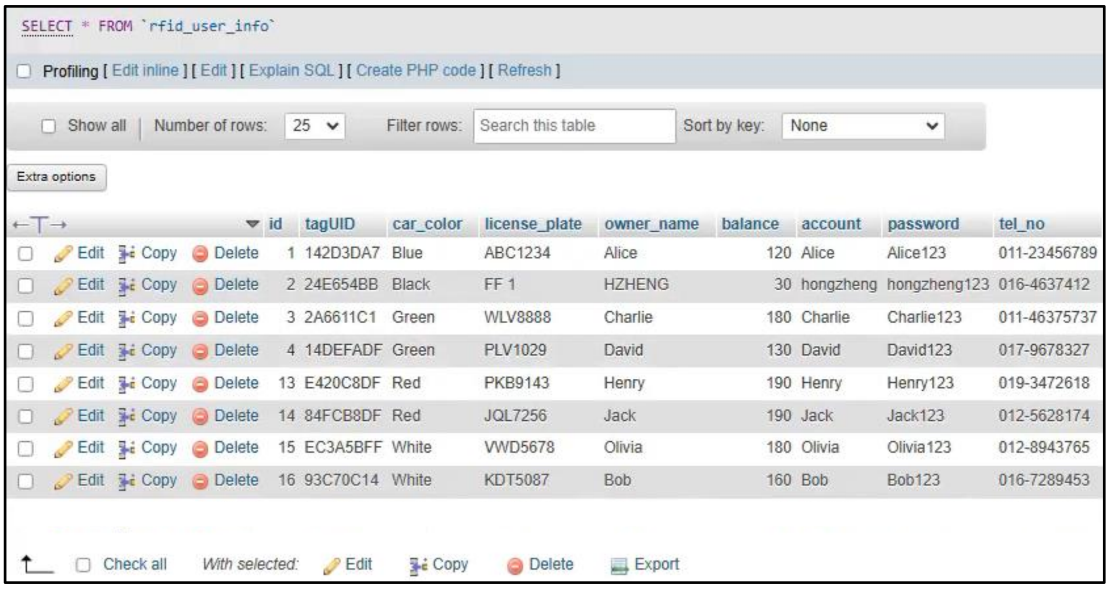
Show all | Number of rows: 25 Filter rows: Search this table

Extra options

slot_number	status
1	AVAILABLE
2	AVAILABLE
3	AVAILABLE
4	AVAILABLE
5	AVAILABLE
6	AVAILABLE
7	AVAILABLE
8	AVAILABLE

Check all With selected: Edit Copy Delete Export

Figure 3.28: The 'parking_status' Table



```
SELECT * FROM `rfid_user_info`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

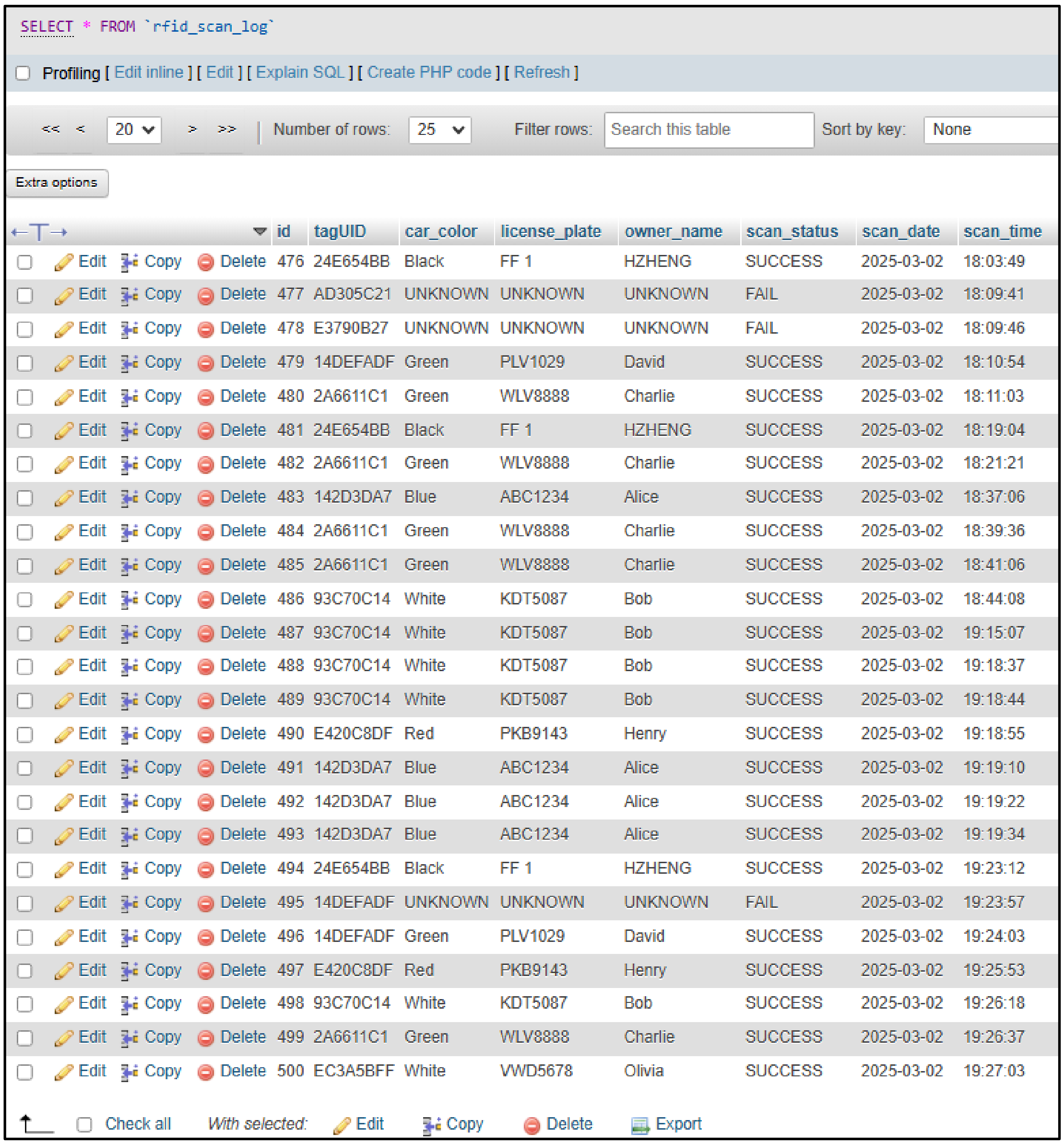
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

id	tagUID	car_color	license_plate	owner_name	balance	account	password	tel_no
1	142D3DA7	Blue	ABC1234	Alice	120	Alice	Alice123	011-23456789
2	24E654BB	Black	FF 1	HZHENG	30	hongzheng	hongzheng123	016-4637412
3	2A6611C1	Green	WLV8888	Charlie	180	Charlie	Charlie123	011-46375737
4	14DEFADF	Green	PLV1029	David	130	David	David123	017-9678327
13	E420C8DF	Red	PKB9143	Henry	190	Henry	Henry123	019-3472618
14	84FCB8DF	Red	JQL7256	Jack	190	Jack	Jack123	012-5628174
15	EC3A5BFF	White	VWD5678	Olivia	180	Olivia	Olivia123	012-8943765
16	93C70C14	White	KDT5087	Bob	160	Bob	Bob123	016-7289453

Check all With selected: Edit Copy Delete Export

Figure 3.29: The 'rfid_user_info' Table



```
SELECT * FROM `rfid_scan_log`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

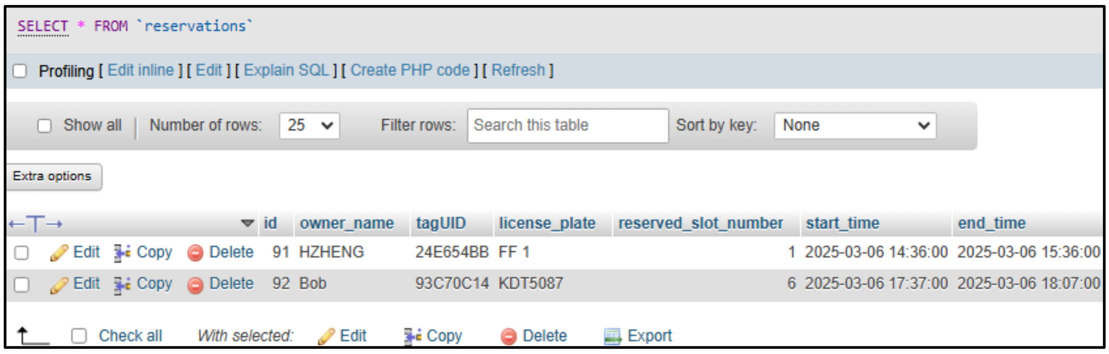
<< < 20 > >> | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

	id	tagUID	car_color	license_plate	owner_name	scan_status	scan_date	scan_time
Edit Copy Delete	476	24E654BB	Black	FF 1	HZHENG	SUCCESS	2025-03-02	18:03:49
Edit Copy Delete	477	AD305C21	UNKNOWN	UNKNOWN	UNKNOWN	FAIL	2025-03-02	18:09:41
Edit Copy Delete	478	E3790B27	UNKNOWN	UNKNOWN	UNKNOWN	FAIL	2025-03-02	18:09:46
Edit Copy Delete	479	14DEFADF	Green	PLV1029	David	SUCCESS	2025-03-02	18:10:54
Edit Copy Delete	480	2A6611C1	Green	WLV8888	Charlie	SUCCESS	2025-03-02	18:11:03
Edit Copy Delete	481	24E654BB	Black	FF 1	HZHENG	SUCCESS	2025-03-02	18:19:04
Edit Copy Delete	482	2A6611C1	Green	WLV8888	Charlie	SUCCESS	2025-03-02	18:21:21
Edit Copy Delete	483	142D3DA7	Blue	ABC1234	Alice	SUCCESS	2025-03-02	18:37:06
Edit Copy Delete	484	2A6611C1	Green	WLV8888	Charlie	SUCCESS	2025-03-02	18:39:36
Edit Copy Delete	485	2A6611C1	Green	WLV8888	Charlie	SUCCESS	2025-03-02	18:41:06
Edit Copy Delete	486	93C70C14	White	KDT5087	Bob	SUCCESS	2025-03-02	18:44:08
Edit Copy Delete	487	93C70C14	White	KDT5087	Bob	SUCCESS	2025-03-02	19:15:07
Edit Copy Delete	488	93C70C14	White	KDT5087	Bob	SUCCESS	2025-03-02	19:18:37
Edit Copy Delete	489	93C70C14	White	KDT5087	Bob	SUCCESS	2025-03-02	19:18:44
Edit Copy Delete	490	E420C8DF	Red	PKB9143	Henry	SUCCESS	2025-03-02	19:18:55
Edit Copy Delete	491	142D3DA7	Blue	ABC1234	Alice	SUCCESS	2025-03-02	19:19:10
Edit Copy Delete	492	142D3DA7	Blue	ABC1234	Alice	SUCCESS	2025-03-02	19:19:22
Edit Copy Delete	493	142D3DA7	Blue	ABC1234	Alice	SUCCESS	2025-03-02	19:19:34
Edit Copy Delete	494	24E654BB	Black	FF 1	HZHENG	SUCCESS	2025-03-02	19:23:12
Edit Copy Delete	495	14DEFADF	UNKNOWN	UNKNOWN	UNKNOWN	FAIL	2025-03-02	19:23:57
Edit Copy Delete	496	14DEFADF	Green	PLV1029	David	SUCCESS	2025-03-02	19:24:03
Edit Copy Delete	497	E420C8DF	Red	PKB9143	Henry	SUCCESS	2025-03-02	19:25:53
Edit Copy Delete	498	93C70C14	White	KDT5087	Bob	SUCCESS	2025-03-02	19:26:18
Edit Copy Delete	499	2A6611C1	Green	WLV8888	Charlie	SUCCESS	2025-03-02	19:26:37
Edit Copy Delete	500	EC3A5BFF	White	VWD5678	Olivia	SUCCESS	2025-03-02	19:27:03

Check all With selected: Edit Copy Delete Export

Figure 3.30: The ‘rfid_scan_log’ Table



```
SELECT * FROM `reservations`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Extra options

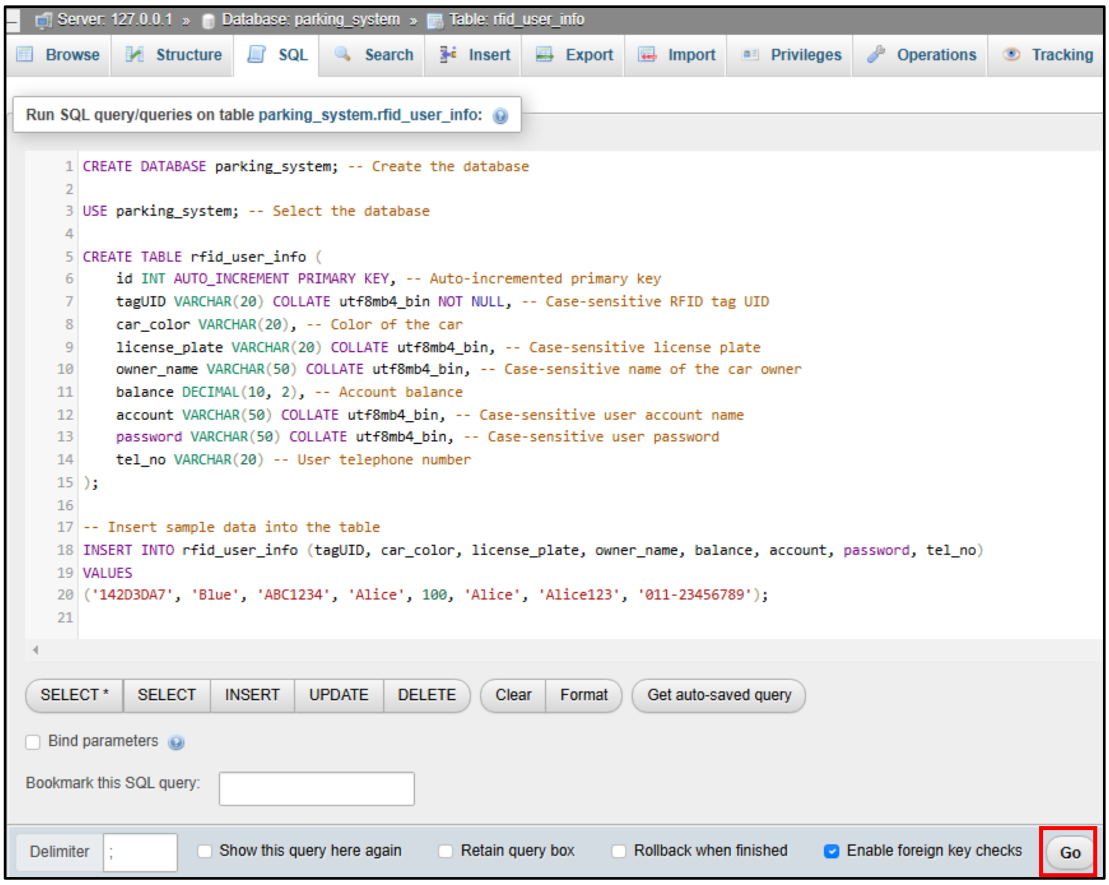
	id	owner_name	tagUID	license_plate	reserved_slot_number	start_time	end_time
Edit Copy Delete	91	HZHENG	24E654BB	FF 1	1	2025-03-06 14:36:00	2025-03-06 15:36:00
Edit Copy Delete	92	Bob	93C70C14	KDT5087	6	2025-03-06 17:37:00	2025-03-06 18:07:00

Check all With selected: Edit Copy Delete Export

Figure 3.31: The ‘reservations’ Table

3.4.2.1 How to Create a Database and a Table

The code snippet in Figure 3.32 shows an example of creating a database named 'parking_system', and creates a table named 'rfid_user_info', with nine columns where each column corresponds to a variable. The 'INSERT' command is used to insert information into the table. To make the variables case-sensitive, a binary collation such as 'utf8mb4_bin' is applied to the relevant columns. This ensures that variables like owner's name, usernames, passwords, and UID are stored and compared exactly as entered, which can enhance security and data accuracy at the login page and reservation page. After inserting the code in the 'SQL' tab, the 'Go' button at the bottom right of the page is clicked and the database and table will be generated.



Run SQL query/queries on table parking_system.rfid_user_info:

```sql
CREATE DATABASE parking_system; -- Create the database

USE parking_system; -- Select the database

CREATE TABLE rfid_user_info (
    id INT AUTO_INCREMENT PRIMARY KEY, -- Auto-incremented primary key
    tagUID VARCHAR(20) COLLATE utf8mb4_bin NOT NULL, -- Case-sensitive RFID tag UID
    car_color VARCHAR(20), -- Color of the car
    license_plate VARCHAR(20) COLLATE utf8mb4_bin, -- Case-sensitive license plate
    owner_name VARCHAR(50) COLLATE utf8mb4_bin, -- Case-sensitive name of the car owner
    balance DECIMAL(10, 2), -- Account balance
    account VARCHAR(50) COLLATE utf8mb4_bin, -- Case-sensitive user account name
    password VARCHAR(50) COLLATE utf8mb4_bin, -- Case-sensitive user password
    tel_no VARCHAR(20) -- User telephone number
);

-- Insert sample data into the table
INSERT INTO rfid_user_info (tagUID, car_color, license_plate, owner_name, balance, account, password, tel_no)
VALUES
('142D3DA7', 'Blue', 'ABC1234', 'Alice', 100, 'Alice', 'Alice123', '011-23456789');
```

☐ Bind parameters

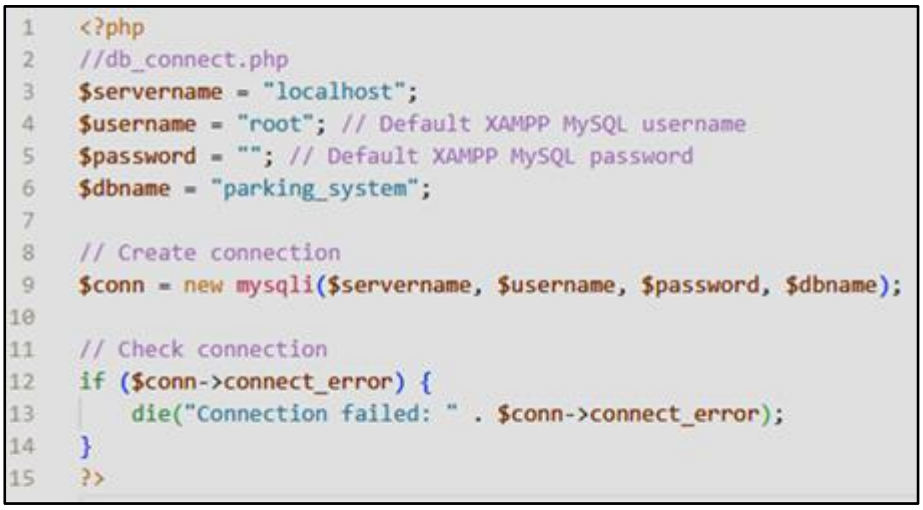
Bookmark this SQL query:

Delimiter ; ☐ Show this query here again ☐ Retain query box ☐ Rollback when finished ☑ Enable foreign key checks

Figure 3.32: An Example of Creating a Database and a Table

3.4.2.2 How to Connect to the Database

The PHP code in Figure 3.33 called ‘db_connect.php’ is written to enable connection to the database called ‘parking_system’. In the code, the server name, username, password, and database name need to be specified. This ‘db_connect.php’ needs to be included at the beginning of every PHP script that requires database interaction, such as retrieving, updating, inserting, or deleting data.



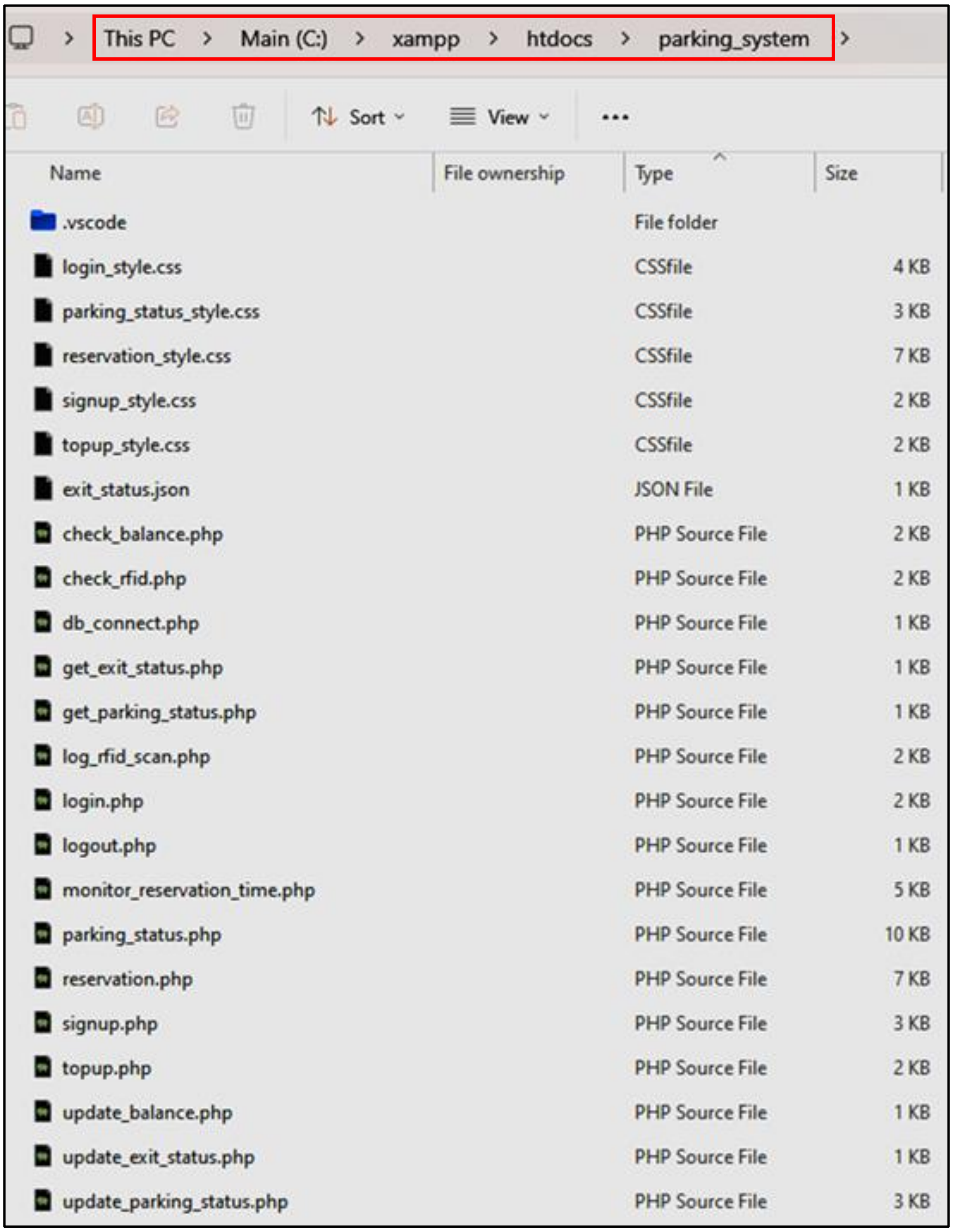
```php
<?php
//db_connect.php
$servername = "localhost";
$username = "root"; // Default XAMPP MySQL username
$password = ""; // Default XAMPP MySQL password
$dbname = "parking_system";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

Figure 3.33: The ‘db_connect.php’ File That Is Used to Connect to the Database

3.4.3 PHP Scripts

In this project, PHP scripts play a crucial role as intermediaries between the ESP32 microcontroller and the MySQL database. Directly connecting the ESP32 to MySQL might sound simple, but it introduces issues like security risks, heavy resource demands on the ESP32, and unencrypted data transmission. By using PHP scripts over HTTP, these problems are avoided. If no PHP scripts were used, all the codes, including the HTML, CSS, and JavaScript would be written and stored on the ESP32 alone, this would make the code very lengthy and overload the ESP32. By using the PHP scripts, the codes are stored on the server (laptop) and can be accessed when needed.

Firstly, a folder called ‘parking_system’, which is the same name as the database name was created manually in the XAMPP Apache Web Server file path folder called ‘htdocs’. Then, all the PHP scripts were written using Microsoft Visual Studio Code and were stored in the ‘parking_system’ folder, which is illustrated in Figure 3.34.



Name	File ownership	Type	Size
.vscode		File folder	
login_style.css		CSSfile	4 KB
parking_status_style.css		CSSfile	3 KB
reservation_style.css		CSSfile	7 KB
signup_style.css		CSSfile	2 KB
topup_style.css		CSSfile	2 KB
exit_status.json		JSON File	1 KB
check_balance.php		PHP Source File	2 KB
check_rfid.php		PHP Source File	2 KB
db_connect.php		PHP Source File	1 KB
get_exit_status.php		PHP Source File	1 KB
get_parking_status.php		PHP Source File	1 KB
log_rfid_scan.php		PHP Source File	2 KB
login.php		PHP Source File	2 KB
logout.php		PHP Source File	1 KB
monitor_reservation_time.php		PHP Source File	5 KB
parking_status.php		PHP Source File	10 KB
reservation.php		PHP Source File	7 KB
signup.php		PHP Source File	3 KB
topup.php		PHP Source File	2 KB
update_balance.php		PHP Source File	1 KB
update_exit_status.php		PHP Source File	1 KB
update_parking_status.php		PHP Source File	3 KB

Figure 3.34: The Location where all the PHP Scripts are Stored

3.4.4 The Roles of Each PHP, CSS, JSON Files Used in This Project

All the files in Figure 3.34 have their own purposes. When a vehicle approaches the entrance and scans the RFID tag, the main ESP32 first uses the ‘check_rfid.php’ script to verify if the scanned RFID tag exists in the ‘rfid_user_info’ table of the database. If the RFID tag is valid, this script retrieves details like the owner’s name, license plate, and account balance and returns this information in a JSON format. The main ESP32 then executes the ‘check_balance.php’ script to query the same table for the user’s account balance. If the balance is sufficient, the parking fee is deducted and the balance field in the ‘rfid_user_info’ table is updated using the ‘update_balance.php’ script. After successfully entering, the RFID scan is logged using ‘log_rfid_scan.php’, which records the scan event in the ‘rfid_scan_log’ table.

Next, for exit operations, the second ESP32 RFID reader scans the RFID tag and updates the ‘exit_status.json’ file using ‘update_exit_status.php’ with details such as UID, owner’s name, car color, license plate, and the database check status. This JSON file is then read by the main ESP32 through ‘get_exit_status.php’. If the JSON file shows database check status success, main ESP32 retrieves the UID, owner’s name, car colour, license plate from the JSON file and print it on the OLED and serial monitor, and updates the ‘exitRFIDAuthorized’ flag to open the exit gate accordingly. The second ESP32 also uses the ‘update_parking_status.php’ script to update the ‘parking_status’ table in the database based on the eight IR sensors monitoring the parking slots whenever there is a change in the IR sensor’s status.

Besides that, the web application includes several pages and functionalities. For example, ‘login.php’, ‘signup.php’, ‘topup.php’, ‘parking_status.php’, and ‘reservation.php’ manage user login, registration, balance top-ups, parking lot status monitoring, and parking slots reservation respectively. Each of these scripts integrates PHP, HTML, CSS, and JavaScript in one place. The ‘login.php’ page allows users to log in by verifying their account and password against the ‘rfid_user_info’ table. After successful login, users will be redirected to the ‘parking_status.php’ page, which displays the parking lot status and their account balance. The parking lot status on the webpage, LCD, and parking slot LEDs are continuously updated by polling the ‘parking_status’ table in the database.

Next, the top-up functionality is handled by ‘topup.php’. The users can select a top-up amount from a dropdown list and confirm the transaction by entering their password. After successful top-up, the balance in the ‘rfid_user_info’ table will be updated, and the user will be redirected to the login page. In addition, the ‘signup.php’ page allows new users to register by entering their RFID tag details and personal information, which is then stored in the ‘rfid_user_info’ table.

Furthermore, the reservation functionality is handled by ‘reservation.php’. This page allows users to reserve parking slots by entering their name, license plate, slot number, arrival time, and duration. The script validates the inputs by checking if the user exists in the ‘rfid_user_info’ table and if the selected parking slot is available during the specified time by querying the ‘reservations’ table. If no conflicts are found, the reservation details are stored in the ‘reservations’ table. The ‘monitor_reservation_time.php’ script's main function is to continuously monitor the ‘reservations’ table in the database and update parking slot statuses accordingly. When a reservation's start time is reached, it automatically sets the corresponding parking slot's status to “RESERVED” in the ‘parking_status’ table. For reservations where the end time has passed or expired, it updates the status to “AVAILABLE”. Additionally, it cleans up the database by deleting expired reservations.

Lastly, the CSS files like ‘login_style.css’, ‘parking_status_style.css’, ‘signup_style.css’, ‘topup_style.css’, and ‘reservation_style.css’ are used to style the web pages to ensure that they are visually appealing and user-friendly.

3.4.5 Ngrok Tunneling

One limitation of the current setup is that the web application is hosted locally using the XAMPP package, which means it can only be accessed by users connected to the same network as the server and the ESP32. There is no point in developing a web application that only local machines can access but other people in the world using different networks cannot access. The solution is to use Ngrok, a tool that creates a secure tunnel by generating a public URL to route traffic from the internet to the local server. This enables global access to the web application, regardless of the user's network connection. Ngrok also uses HTTPS encryption, which ensures that data transmission is secure. The details of how to set up Ngrok to generate a public URL are illustrated in Appendix A.

Although using Ngrok is sufficient for the webpages to work, it is only a temporary solution since the server needs to be manually started each time before the web pages can be accessed. To make the system a permanent solution, public web hosting services could be used in the future. This would involve purchasing a domain name. This way, the webpages are always online and do not require starting the server every time. Alternatively, the XAMPP package could be installed on a Raspberry Pi and configured to run continuously. Although these alternatives offer long-term solutions, for now, using XAMPP together with Ngrok is sufficient and will be the chosen approach for this project.

3.5 Project Management / Gantt Chart

The Gantt charts for FYP 1 and FYP 2 are presented in Table 3.2, and Table 3.3, respectively.

Table 3.2: Gantt Chart for FYP 1

Task	Week													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Dicsussion With Supervisor	🟦													
FYP Title Selection	🟦													
Research	🟦	🟦	🟦											
Purchase Components Online			🟦											
Testing Components			🟦	🟦										
Build Hardware Prototype on Breadboard			🟦	🟦	🟦	🟦	🟦	🟦	🟦					
Develop Web Application						🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	
FYP 1 Report Writing														
Literature Review	🟦	🟦	🟦	🟦										
Introduction				🟦	🟦	🟦	🟦							
Methodology							🟦	🟦	🟦	🟦	🟦	🟦		
Presentation														
FYP 1 Presentation													🟦	

Table 3.3: Gantt Chart for FYP 2

Task	Week													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Build Hardware Prototype on Stripboard	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦					
Final Prototype Testing									🟦	🟦	🟦			
FYP 2 Report Writing														
Results								🟦	🟦	🟦	🟦			
Conclusion											🟦	🟦		
FYP Report Formatting Checking												🟦	🟦	🟦
Presentation														
Poster Presentation												🟦		
FYP 2 Presentation													🟦	

3.6 Cost of Components

In this study, managing the cost used in FYP is a crucial part. It is important to select components and materials, which able to meet the project requirements. The cost of components used in this project is shown in Table 3.4.

Table 3.4: Components List with Price

No.	Components	Unit Price (RM)	Quantity	Total Price (RM)	Remarks
1.	NodeMCU ESP32	25.00	2	50.00	Shopee, Robotronik
2.	IR Sensor	2.90	13	37.70	Shopee, Robotedu
3.	MFRC522 RFID Reader	5.49	2	10.98	Shopee, Robotronik
4.	SG90 Servo Motor	5.84	2	11.68	Shopee, Robotedu
5.	I2C 2004 LCD Display	19.90	1	19.90	Shopee, Robotedu
6.	OLED Display	14.90	1	14.90	Shopee, Robotedu
7.	AI-Thinker ESP32-CAM	29.90	1	29.90	Shopee, Robotedu
8.	PCF8575 I/O Expander	14.00	2	28.00	Shopee, WeiWang
9.	LED	0.10	26	2.60	Shopee, Robotedu
10.	24 AWG Wires (1 m)	1.00	42	42.00	Shopee, SYNACORP
11.	Jumper Wires	3.20	4	12.80	Shopee, littlecraft
12.	Stripboard (10 × 24.5 cm)	5.00	1	5.00	Shopee, Robotronik
13	5V 4A AC-DC Power Adapter	15.99	1	15.99	Shopee, Robotronik
14	12V 2A AC-DC Power Adapter	10.09	1	10.09	Shopee, Robotronik
			TOTAL	**291.54**	Not including equipment

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Schematic Diagram

The schematic diagrams of the main ESP32, the second ESP32, and the ESP32-CAM are shown in Figures 4.3, Figure 4.4, and Figure 4.5, respectively. The main ESP32 handles the input devices, such as four IR sensors for the entrance and exit gates and the entrance RFID reader. It also controls the output devices, including LEDs, a buzzer, an LCD, an OLED display, and the entrance and exit servo motors. In contrast, the second ESP32 manages the eight parking slot IR sensors, the exit RFID reader, and two PCF8575 I/O expanders. Both ESP32 share a common ground. The two PCF8575 I/O expanders manage the twenty-four parking slot status LEDs.

Firstly, the main ESP32 is powered by a power bank. Since the GPIO pins of the ESP32 only output 3.3 V, a four-channel 3.3 V to 5 V logic level shifter is used to convert the 3.3 V PWM signal from the ESP32 to 5 V before sending it to the signal pin of the servo motor. This ensures the servo motor receives the correct operating voltage (5 V) for proper functioning.

Besides that, the second ESP32 is powered via a USB connection to a PC. Both the entrance RFID reader on the main ESP32 and the exit RFID reader on the second ESP32 are powered by an independent breadboard power supply, which itself is powered by a 12 V / 4 A DC adapter, instead of using the 3.3 V pin on the ESP32. This ensures that the RFID readers receive a stable 3.3 V operating voltage. When too many components share the same 3.3 V pin on the ESP32, the voltage may not be sufficient, which could prevent the RFID readers from receiving the correct 3.3 V. This can lead to the RFID readers failing to read RFID tags or not functioning properly.

Next, the twenty-four LEDs connected to the PCF8575 I/O expanders are configured in a current sink configuration, as illustrated in Figure 4.1. In this current sink configuration, the anode of the LED is connected to V_{CC} through a current-limiting resistor, and its cathode is connected to the I/O pin of the PCF8575, which acts as a current sink. When the GPIO pin of the PCF8575 is set to LOW, it creates a path to the ground, allowing current to flow through the LED. This turns the LED ON. In contrast, when the GPIO pin is set to HIGH, the voltage at both ends of the LED is the same, preventing current from flowing, and the LED remains OFF.

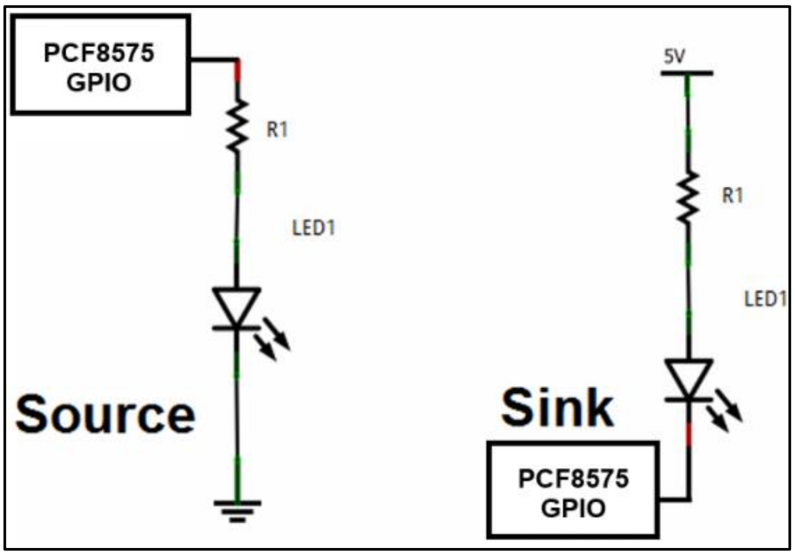
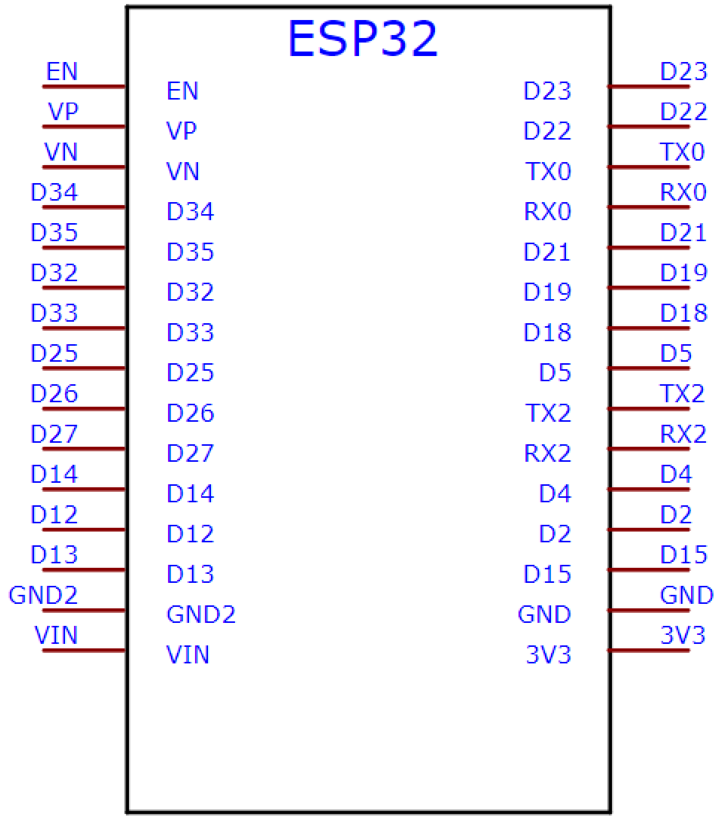


Figure 4.1: Current Source and Current Sink Configuration of a LED

One of the main advantages of sinking current in this configuration is that it simplifies the design by allowing the I/O pin to only handle lower voltage levels and small amounts of current. This reduces the risk of excessive power dissipation on the microcontroller, as it does not need to supply the current directly (Bishop, 2023). Additionally, sinking current helps reduce noise and voltage fluctuations, leading to more stable operation of the circuit, especially when controlling multiple LEDs or other output devices (Bishop, 2023).



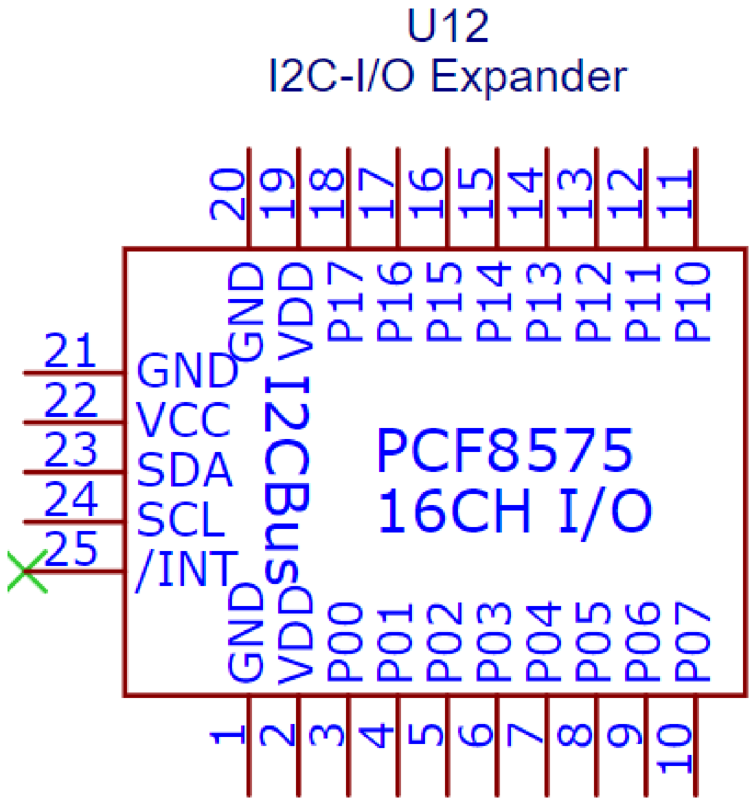


Figure 4.2: Zoomed in View of ESP32 and PCF8575 I/O Expander Pinout

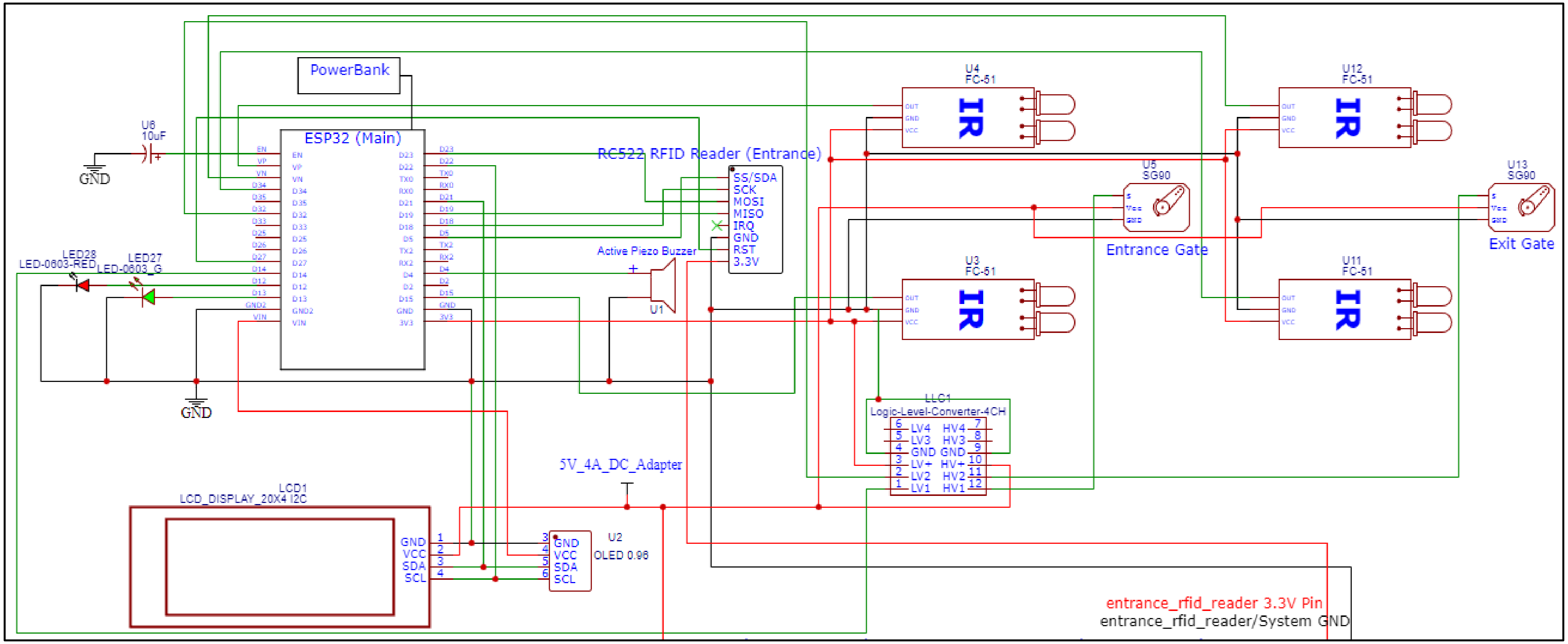


Figure 4.3: Schematic Diagram of IoT RFID Smart Parking System (Main ESP32) Drawn Using EasyEDA Software

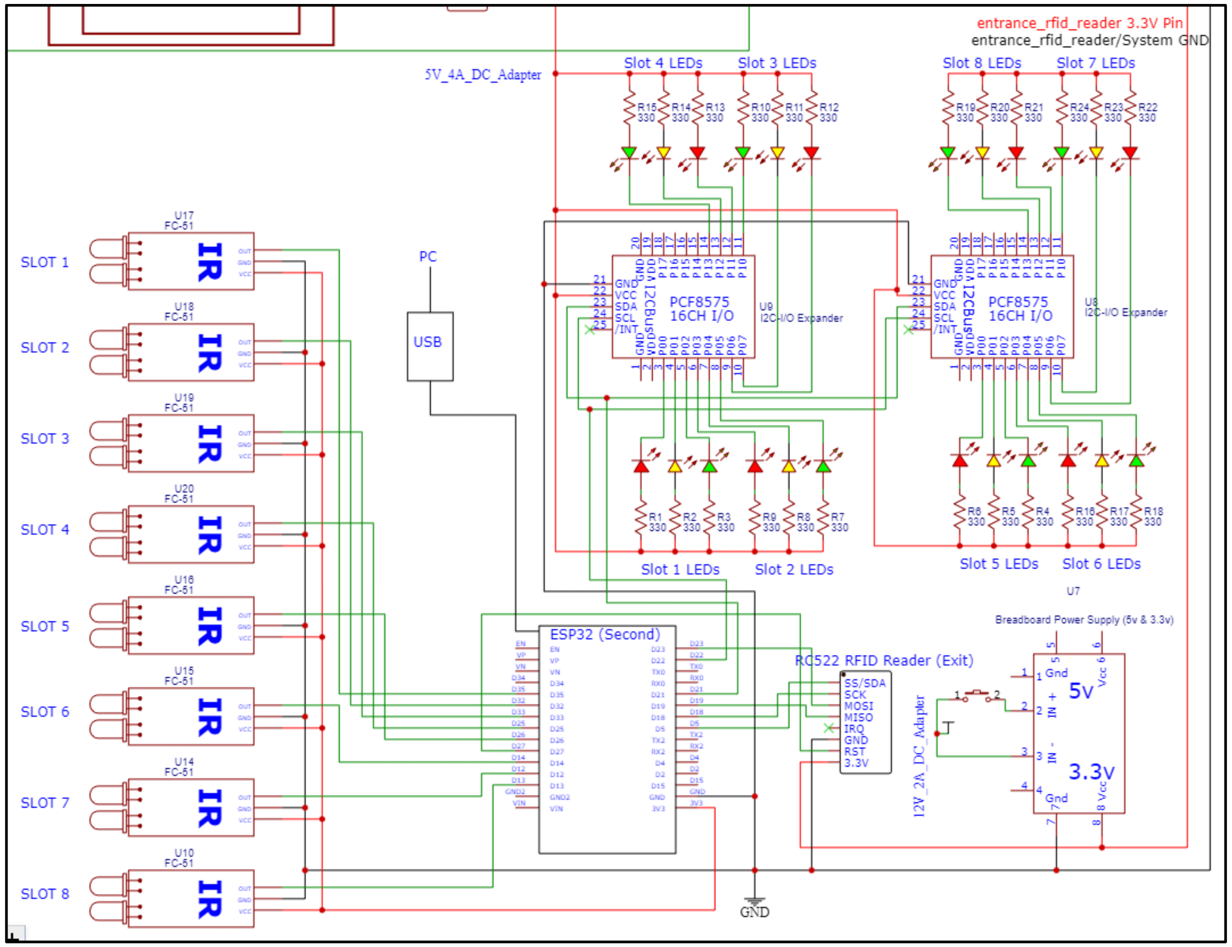


Figure 4.4: Schematic Diagram of IoT RFID Smart Parking System (Second ESP32) Drawn Using EasyEDA Software

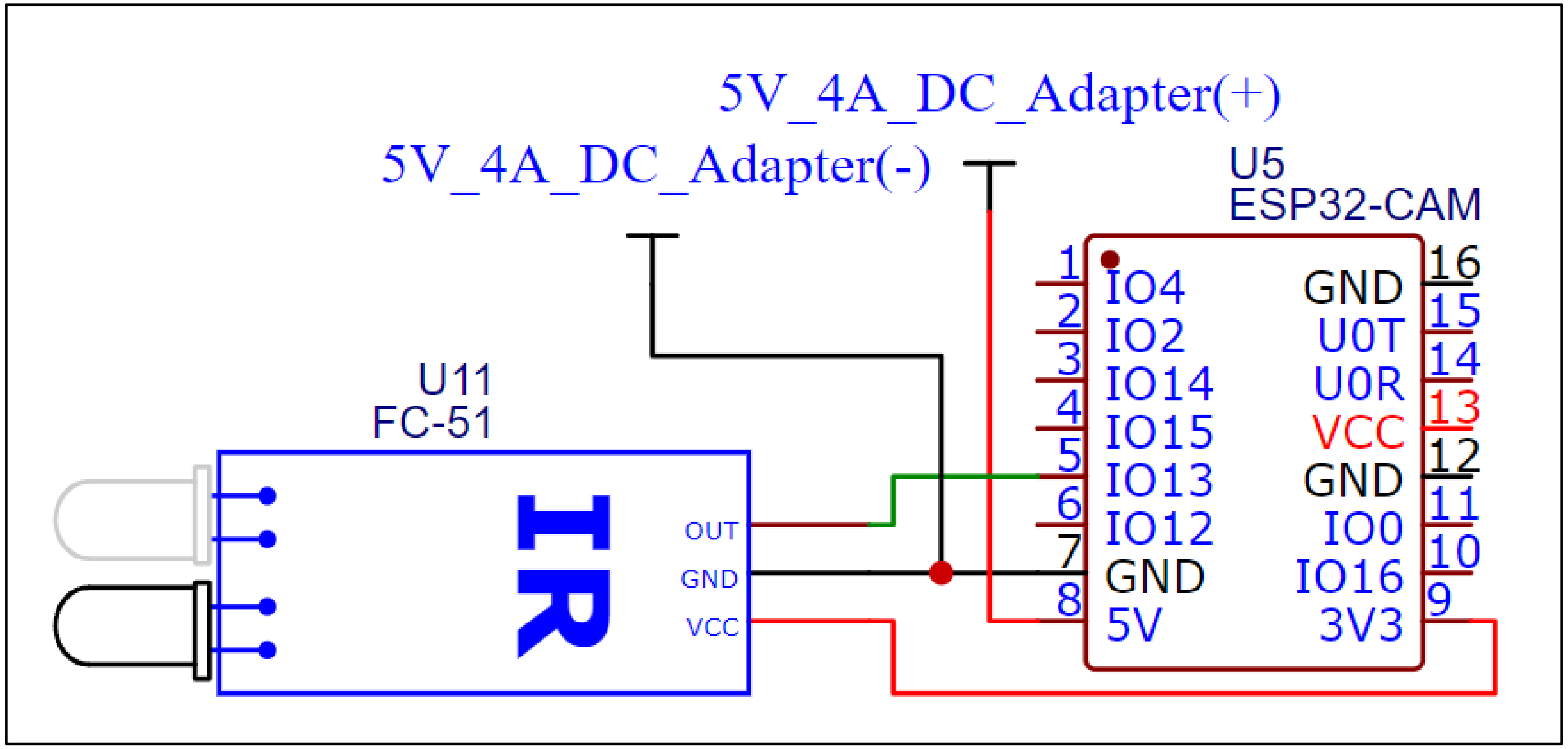


Figure 4.5: Schematic Diagram of ESP32-CAM Image Capturing Camera System at the Entrance Drawn Using EasyEDA Software

4.2 Webpages/User Interface

4.2.1 Login Page

Figure 4.6 shows the overview, while Figure 4.7 shows a zoomed-in view of the “Login Page”. If the user does not have an account, they can press the “Sign Up” button to go to the “Sign Up Page”. If the user already has an account, they need to enter the correct username and password. Note that both fields are case-sensitive. In Figure 4.8, the account stored in the “rfid_user_info” table of the database is “Bob”, with the password “Bob123”. In Figure 4.9, the user enters “bob” in the account field. Although the password is correct, the system does not recognise the account due to the case difference in the account field. As a result, the login attempt fails, as shown in Figure 4.10.

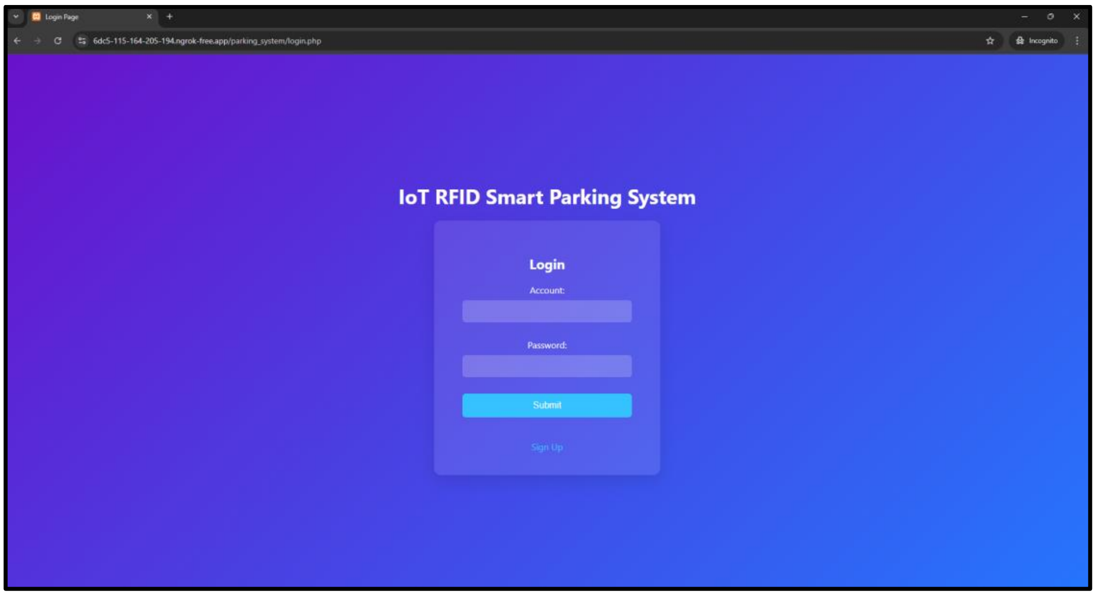

Login Page
6dc5-115-164-205-194.ngrok-free.app/parking_system/login.php
Incognito
IoT RFID Smart Parking System
Login
Account:
Password:
Submit
Sign Up

Figure 4.6: The Overview of Login Page (login.php)

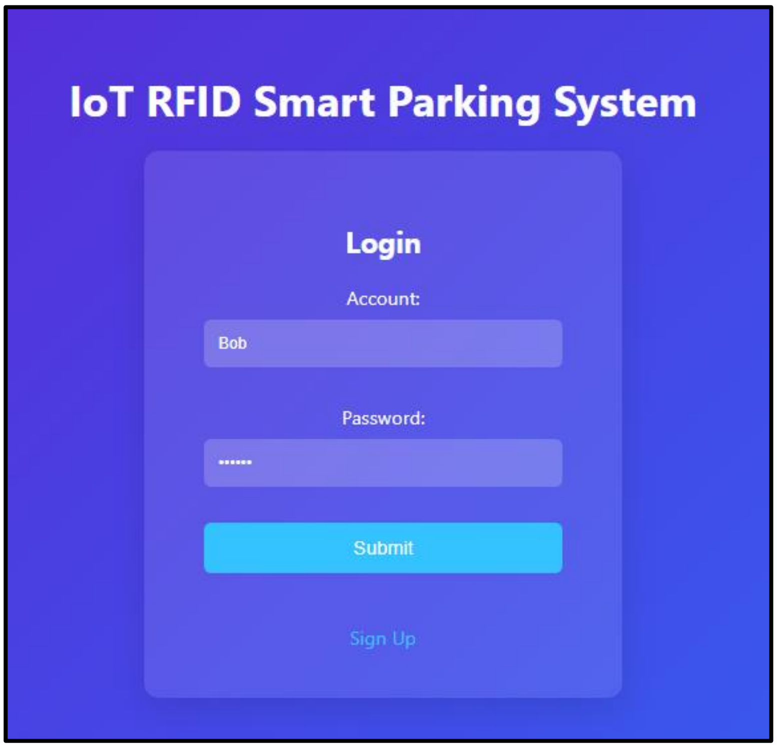
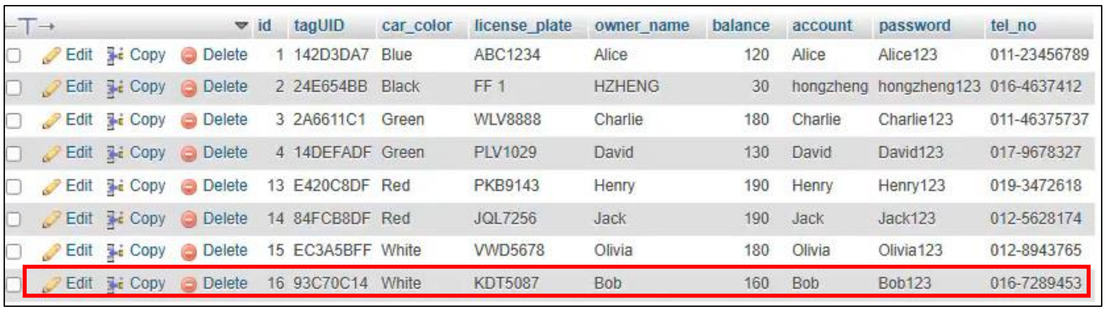


Figure 4.7: Zoomed in View of Login Page with Login Credentials Filled in (Case-Sensitive)



			id	tagUID	car_color	license_plate	owner_name	balance	account	password	tel_no
Edit	Copy	Delete	1	142D3DA7	Blue	ABC1234	Alice	120	Alice	Alice123	011-23456789
Edit	Copy	Delete	2	24E654BB	Black	FF 1	HZHENG	30	hongzheng	hongzheng123	016-4637412
Edit	Copy	Delete	3	2A6611C1	Green	WLV8888	Charlie	180	Charlie	Charlie123	011-46375737
Edit	Copy	Delete	4	14DEFADF	Green	PLV1029	David	130	David	David123	017-9678327
Edit	Copy	Delete	13	E420C8DF	Red	PKB9143	Henry	190	Henry	Henry123	019-3472618
Edit	Copy	Delete	14	84FCB8DF	Red	JQL7256	Jack	190	Jack	Jack123	012-5628174
Edit	Copy	Delete	15	EC3A5BFF	White	VWD5678	Olivia	180	Olivia	Olivia123	012-8943765
Edit	Copy	Delete	16	93C70C14	White	KDT5087	Bob	160	Bob	Bob123	016-7289453

Figure 4.8: Account (Bob) and Password (Bob123) Stored in Database

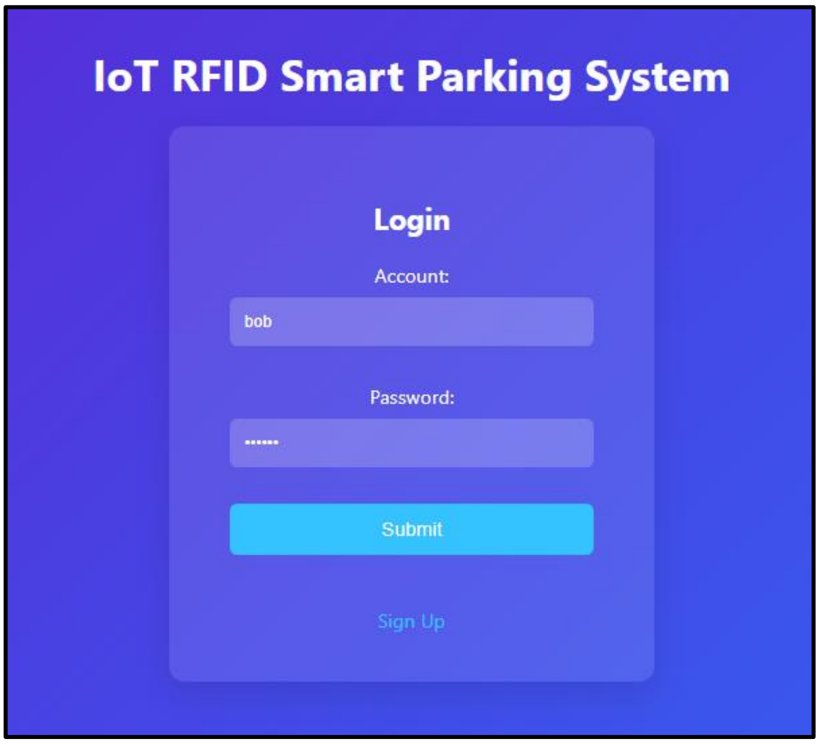
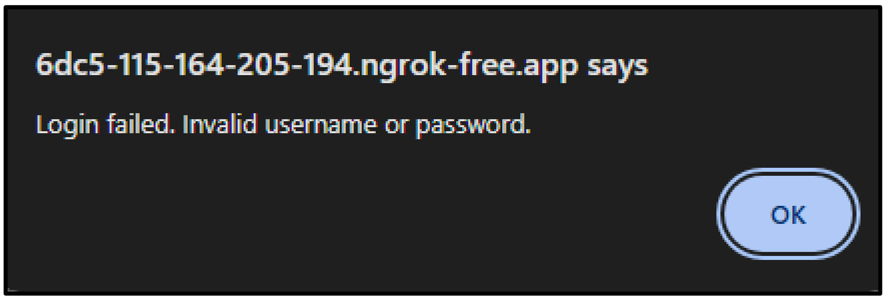


Figure 4.9: Login Attempt Using the Correct Account with Incorrect Letter Case (bob)


6dc5-115-164-205-194.ngrok-free.app says

Login failed. Invalid username or password.

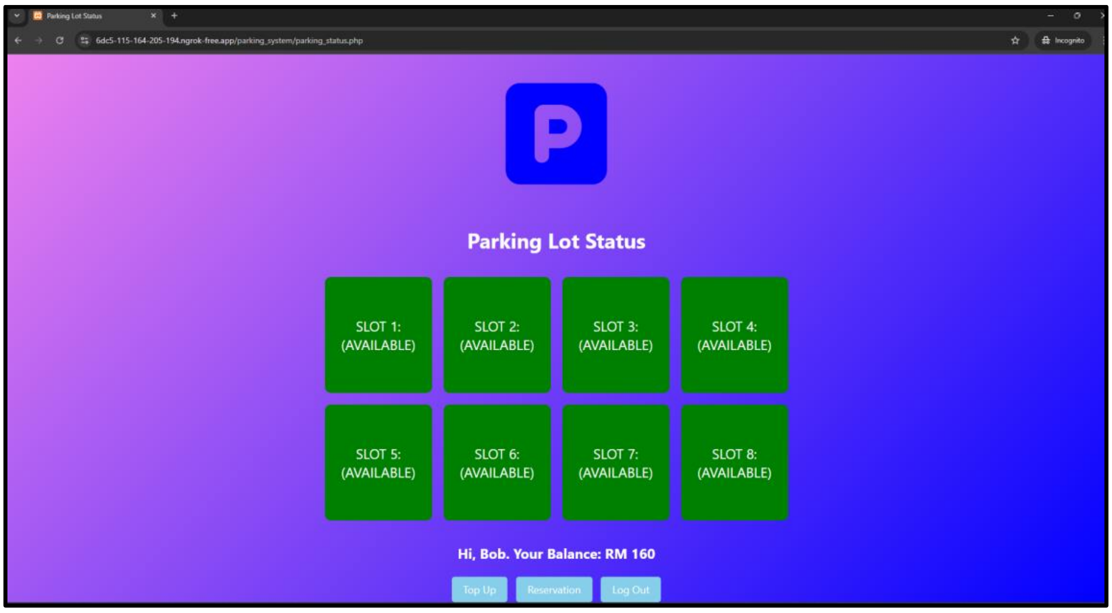
Figure 4.10: Login Failed

4.2.2 <u>Parking Lot Status Monitoring Page (Accessible After Successful Login)</u>

Figure 4.11 shows the overview, while Figure 4.12 shows a zoomed-in view of the "Parking Lot Status Monitoring Page". This webpage is only accessible after a successful login. On this page, the user can view the status of each parking slot and the account balance.

For example, Figure 4.12 shows that all slots are available, as indicated by the green-coloured parking slot boxes. In Figure 4.13, slots 2, 4, 6, and 8 are occupied, as indicated by the red-coloured parking slot boxes. Figure 4.14 shows that all slots are occupied, and an alert message is sent to the user to indicate that the parking lot is full, as shown in Figure 4.15.

Next, at the bottom of the webpage, there are three buttons which are "Top Up", "Reservation", and "Log Out". Pressing the "Top Up" button will navigate the user to the "Top Up Page", pressing the "Reservation" button will navigate the user to the "Reservation Page", and pressing the "Log Out" button will return the user to the "Login Page".


Parking Lot Status
6dc5-115-164-205-194.ngrok-free.app/parking_system/parking_status.php
Incognito
P
Top Up
Reservation
Log Out

Parking Lot Status

SLOT 1: (AVAILABLE)
SLOT 2: (AVAILABLE)
SLOT 3: (AVAILABLE)
SLOT 4: (AVAILABLE)
SLOT 5: (AVAILABLE)
SLOT 6: (AVAILABLE)
SLOT 7: (AVAILABLE)
SLOT 8: (AVAILABLE)

Hi, Bob. Your Balance: RM 160

Figure 4.11: The Overview of Parking Lot Status Monitoring Page (parking_status.php)

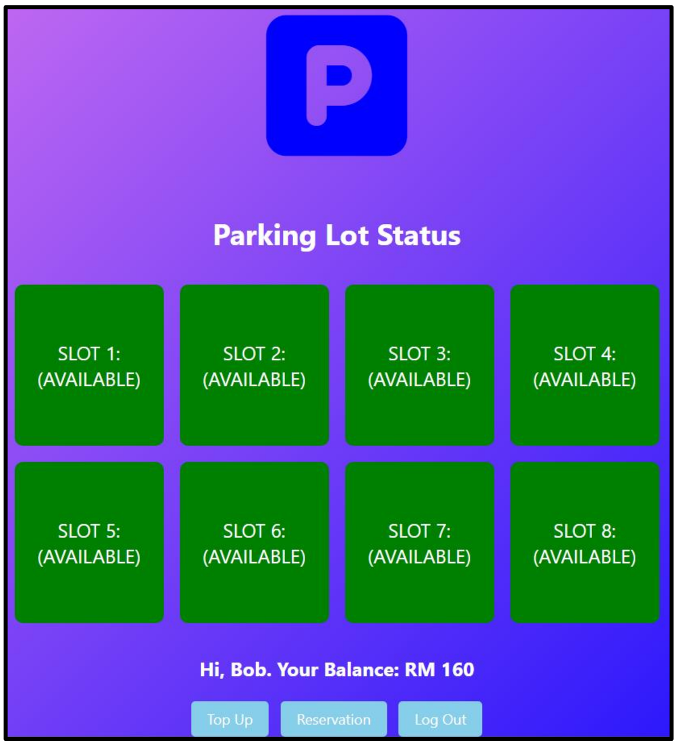


Figure 4.12: Zoomed in View of Parking Lot Status Monitoring Page where All Slots are Available.

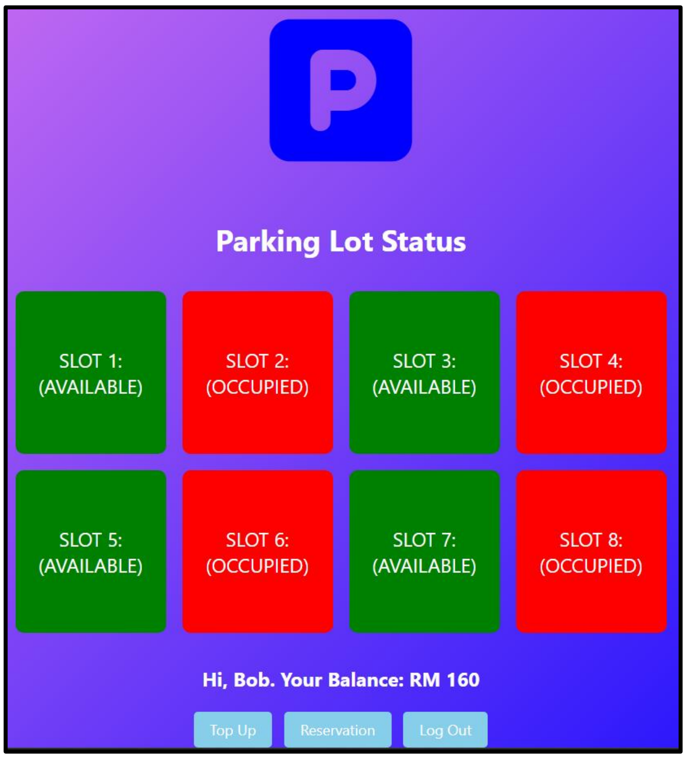
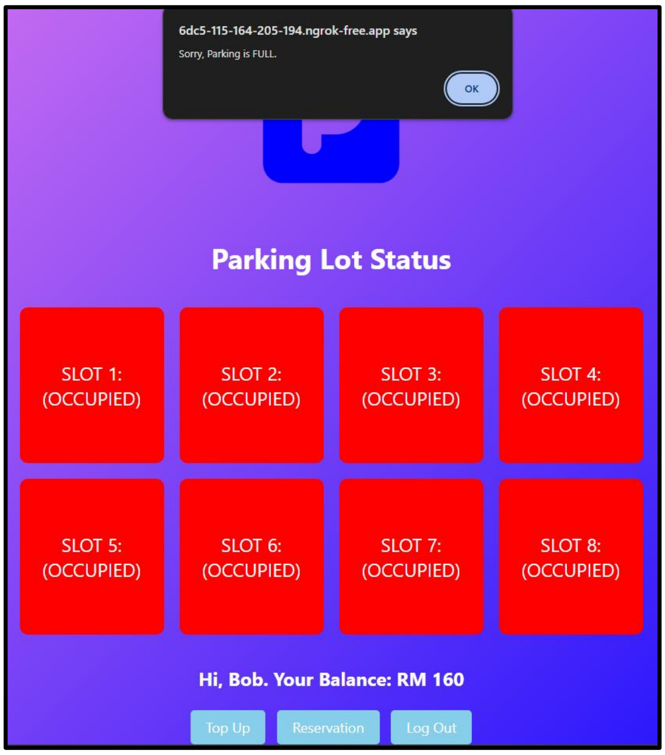


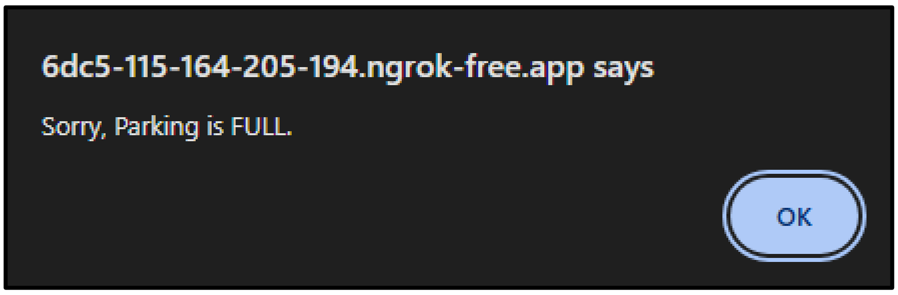
Figure 4.13: Zoomed in View of Parking Lot Status Monitoring Page where Slot 2, 4, 6, 8 are Occupied



6dc5-115-164-205-194.ngrok-free.app says

Sorry, Parking is FULL.

Figure 4.14: Zoomed in View of Parking Lot Status Monitoring Page where All Slots are Occupied



6dc5-115-164-205-194.ngrok-free.app says

Sorry, Parking is FULL.

Figure 4.15: The Popped-Up Message Indicates Parking Full

4.2.3 Top-Up Page

Figure 4.16 shows the overview of the “Top Up Page”. The user is required to select a top-up amount from the drop-down list, which ranges from RM 10 to RM 100. For example, in Figure 4.17, the user selects RM 30 as the top-up amount. After that, the user must enter their account password as a security measure, as shown in Figure 4.18.

Next, Figure 4.19 and Figure 4.20 show the top-up success and failure messages, respectively. When the top-up is successful, the account balance increases from RM 160 in Figure 4.12 to RM 190 in Figure 4.21. The updated balance will also be reflected in the “rfid_user_info” table in the database.

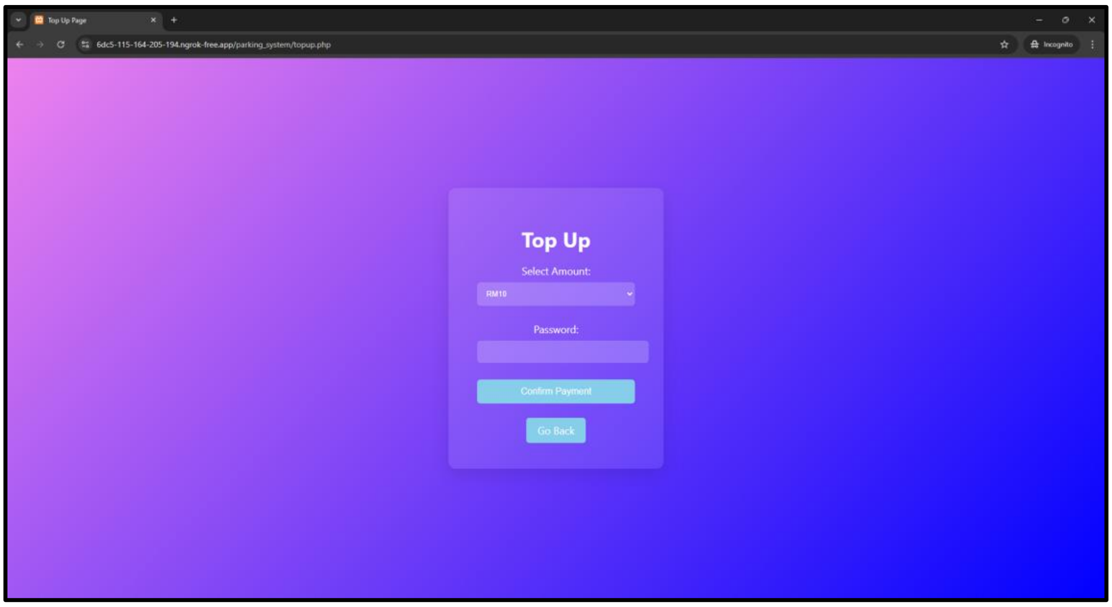


Figure 4.16: The Overview of Top-Up Page (topup.php)

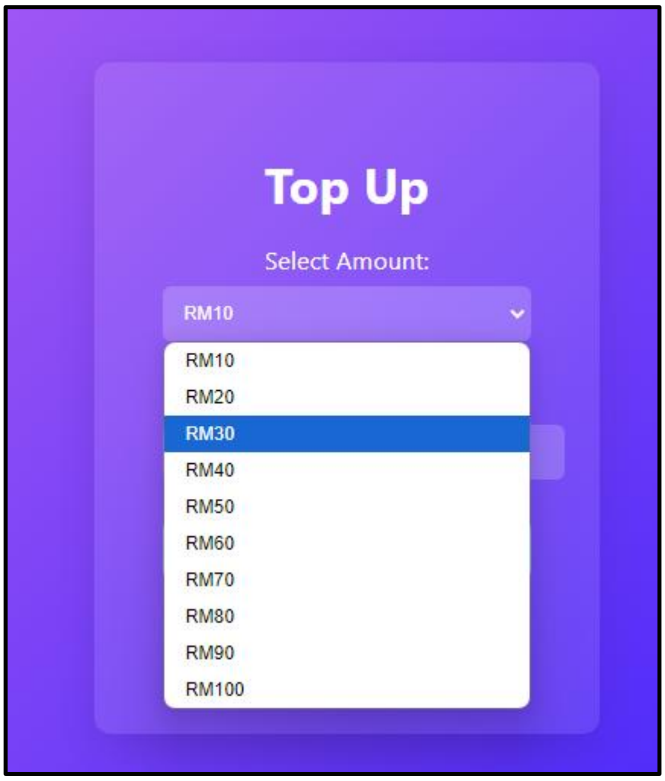


Figure 4.17: The Top-Up Amount Drop Down Selection List

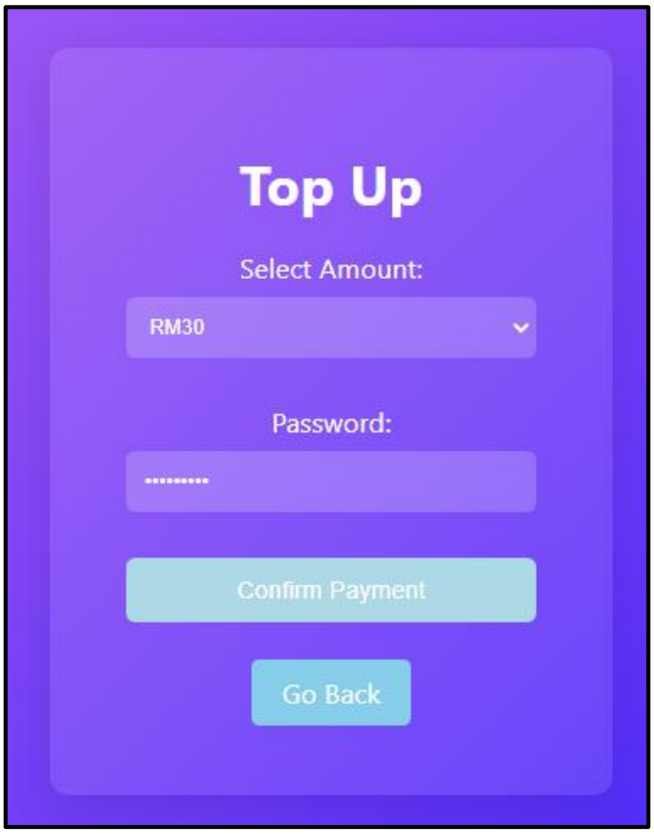
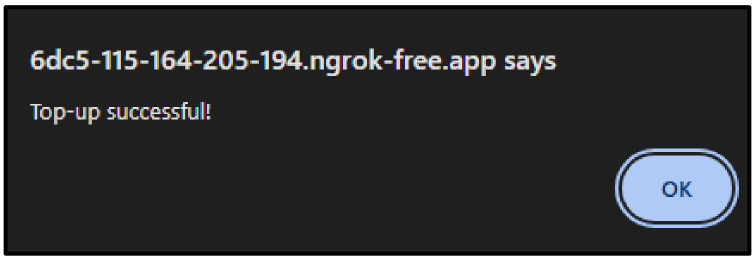
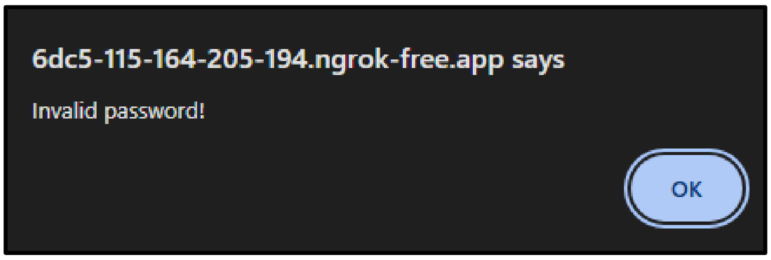


Figure 4.18: The Security Measure (Password) of the Top-Up Function


6dc5-115-164-205-194.ngrok-free.app says

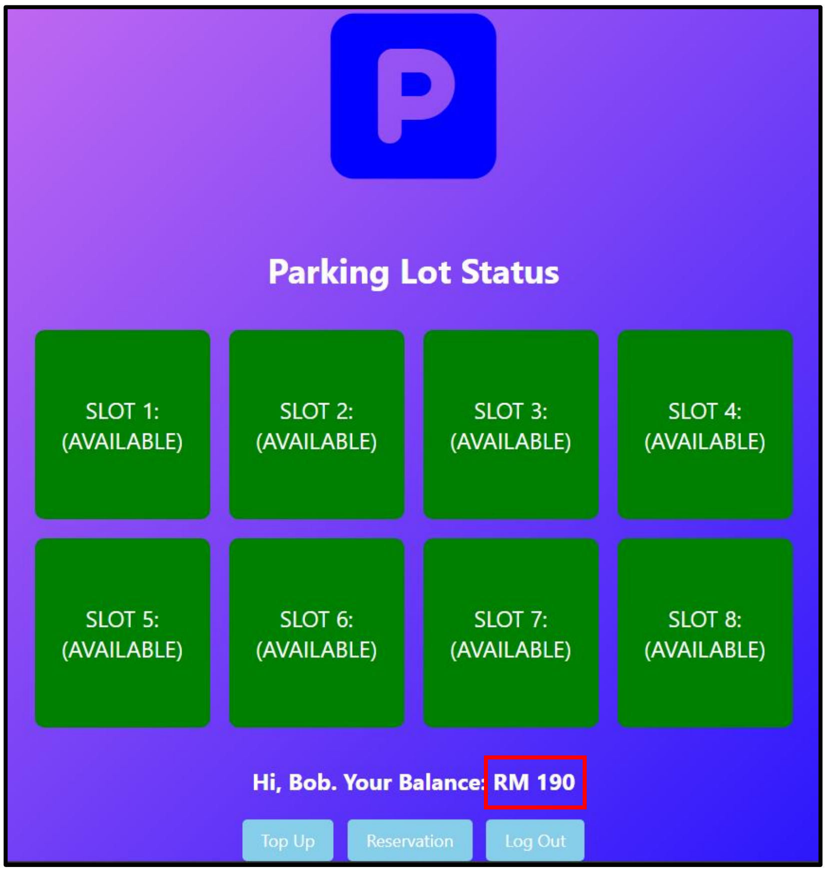
Top-up successful!

Figure 4.19: Top Up Successful if Password Is Correct


6dc5-115-164-205-194.ngrok-free.app says

Invalid password!

Figure 4.20: Top Up Unsuccessful if Password Is Wrong


Parking Lot Status

SLOT 1: (AVAILABLE)

SLOT 2: (AVAILABLE)

SLOT 3: (AVAILABLE)

SLOT 4: (AVAILABLE)

SLOT 5: (AVAILABLE)

SLOT 6: (AVAILABLE)

SLOT 7: (AVAILABLE)

SLOT 8: (AVAILABLE)

Hi, Bob. Your Balance: RM 190

Figure 4.21: The Parking Lot Status Monitoring Page Account Balance Updated Immediately After Top-Up Successful

4.2.4 <u>Account Registration / Sign-Up Page</u>

Figure 4.22 shows the overview, while Figure 4.23 shows a zoomed-in view of the “Sign Up Page”. The user is required to enter information such as tag UID, car colour, license plate number, owner name, account, password, and telephone number. The tag UID is provided by the parking lot owner and is obtained using an RFID reader.

After a successful sign-up, as shown in Figure 4.24, the user will be redirected to the “Login Page”. The newly registered account, with a default balance of RM 0, will be updated in the “rfid_user_info” table in the database, as shown in Figure 4.25.

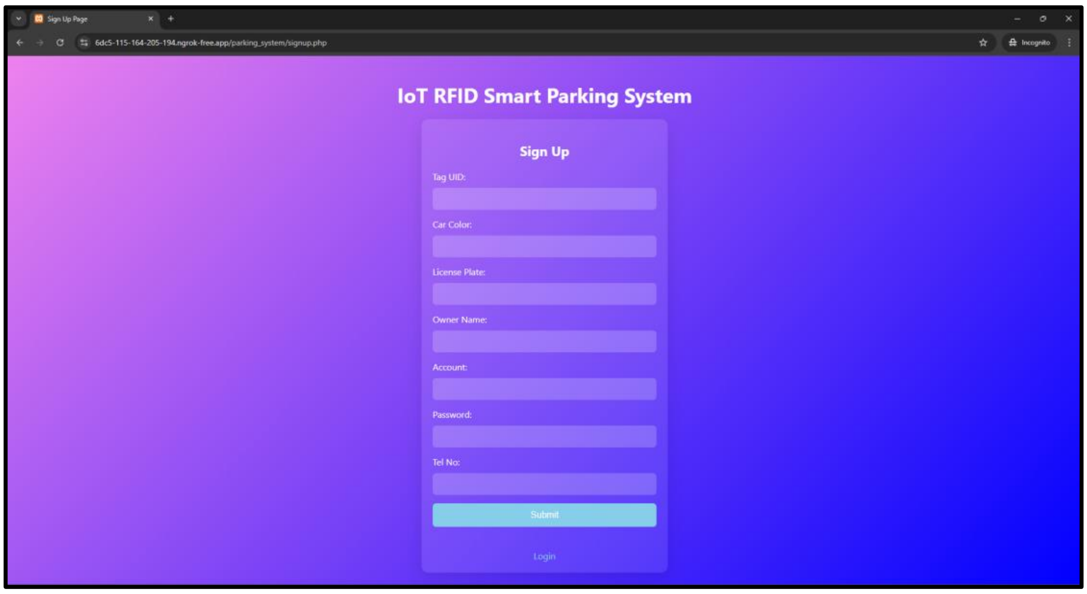
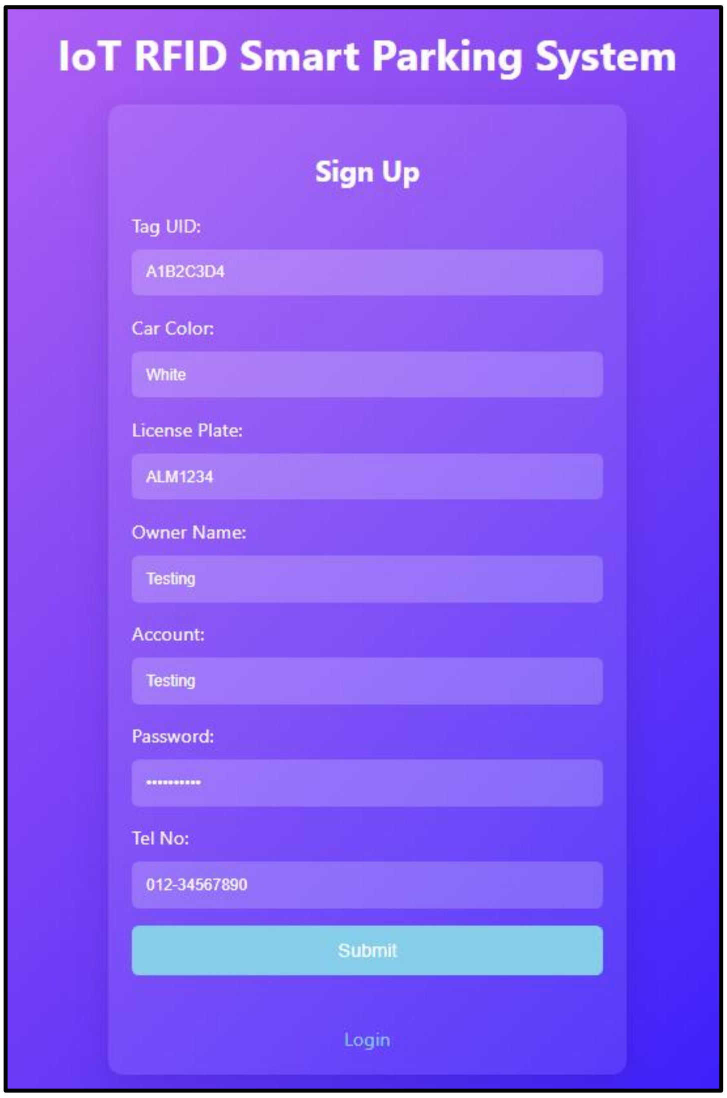


Figure 4.22: The Overview of Sign-Up Page (signup.php)



IoT RFID Smart Parking System

Sign Up

Tag UID: A1B2C3D4

Car Color: White

License Plate: ALM1234

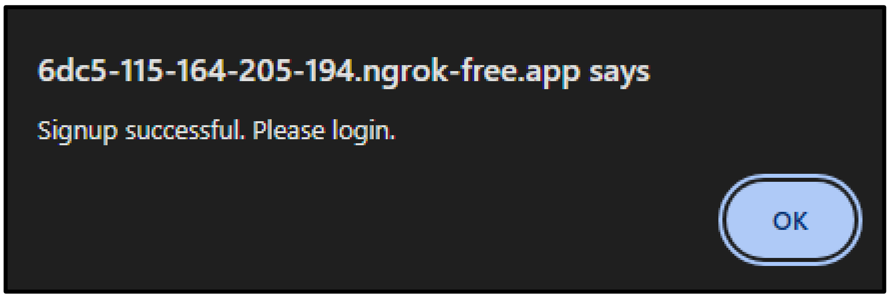
Owner Name: Testing

Account: Testing

Password: ••••••••••

Tel No: 012-34567890

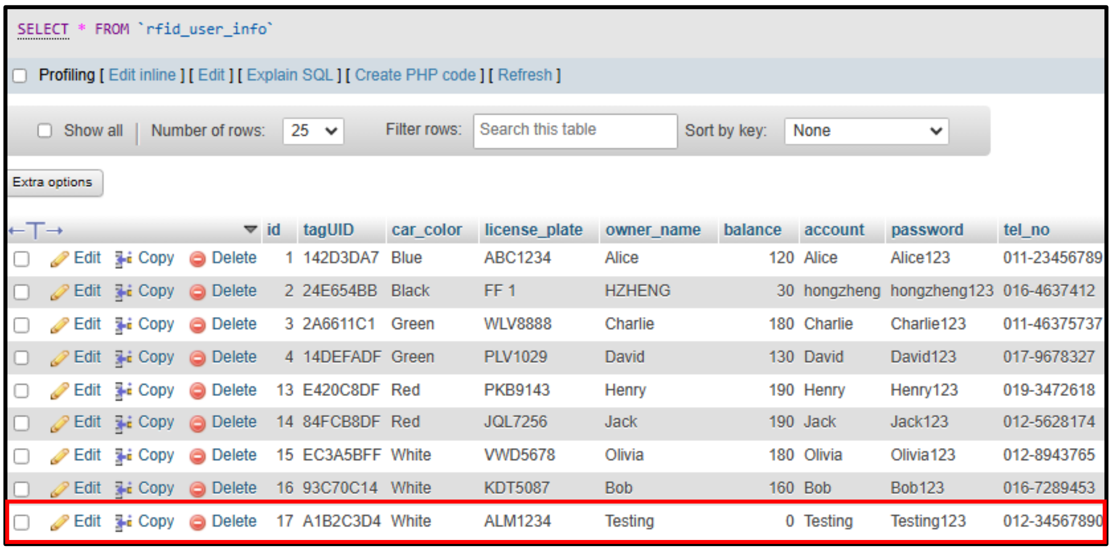
Figure 4.23: Zoomed in View of Sign-Up Page with Information Filled in



6dc5-115-164-205-194.ngrok-free.app says

Signup successful. Please login.

Figure 4.24: Sign-Up Successful Message



```
SELECT * FROM `rfid_user_info`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	id	tagUID	car_color	license_plate	owner_name	balance	account	password	tel_no
Edit Copy Delete	1	142D3DA7	Blue	ABC1234	Alice	120	Alice	Alice123	011-23456789
Edit Copy Delete	2	24E654BB	Black	FF 1	HZHENG	30	hongzheng	hongzheng123	016-4637412
Edit Copy Delete	3	2A6611C1	Green	WLV8888	Charlie	180	Charlie	Charlie123	011-46375737
Edit Copy Delete	4	14DEFADF	Green	PLV1029	David	130	David	David123	017-9678327
Edit Copy Delete	13	E420C8DF	Red	PKB9143	Henry	190	Henry	Henry123	019-3472618
Edit Copy Delete	14	84FCB8DF	Red	JQL7256	Jack	190	Jack	Jack123	012-5628174
Edit Copy Delete	15	EC3A5BFF	White	VWD5678	Olivia	180	Olivia	Olivia123	012-8943765
Edit Copy Delete	16	93C70C14	White	KDT5087	Bob	160	Bob	Bob123	016-7289453
Edit Copy Delete	17	A1B2C3D4	White	ALM1234	Testing	0	Testing	Testing123	012-34567890

Figure 4.25: Database “rfid_user_info” Table Updated with the Newly Registered Account

4.2.5 <u>Reservation Page</u>

Figure 4.26 shows the overview, while Figure 4.27 shows a zoomed-in view of the “Reservation Page”. This page allows the user to reserve an available parking slot before arriving at the parking lot. The user is required to enter information such as the owner name (case-sensitive), license plate number, the slot number they want to reserve, the reservation start time, and duration in minutes.

In Figure 4.8, the owner name stored in the “rfid_user_info” table of the database is “Bob”. However, in Figure 4.28, the user enters “bob” in the owner name field. Although the name is correct, the system does not recognise it due to the case difference. As a result, the reservation attempt fails, as shown in Figure 4.29.

If the owner name is entered correctly, the user can proceed to fill in the remaining information, as shown in Figure 4.30, and select the reservation time, as shown in Figure 4.31. If the selected slot is available and there is no conflict with existing reservations, the reservation will be successful, as shown in Figure 4.32.

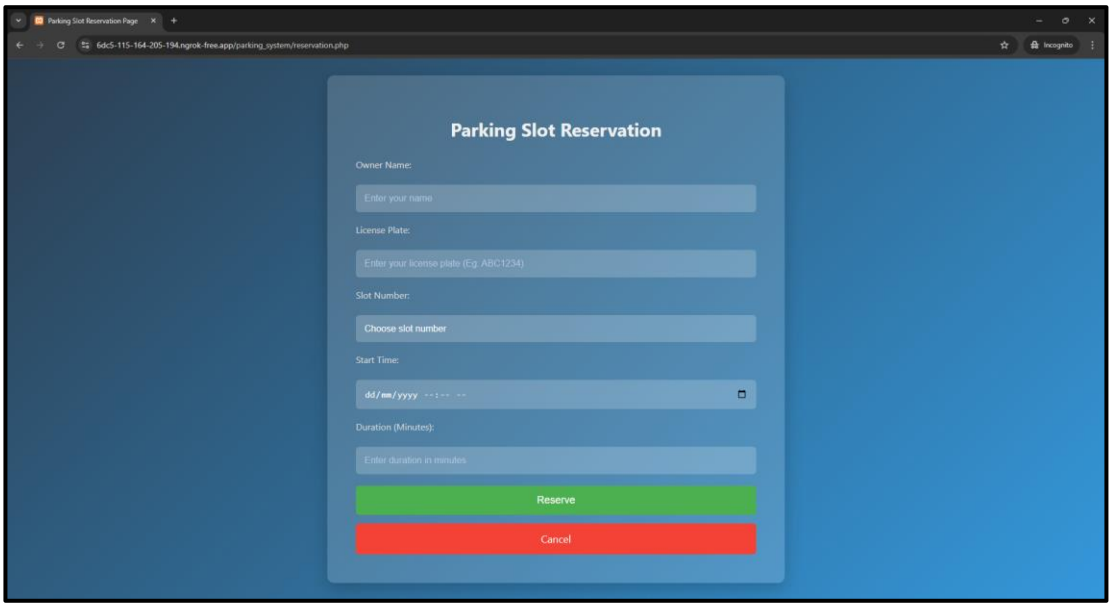
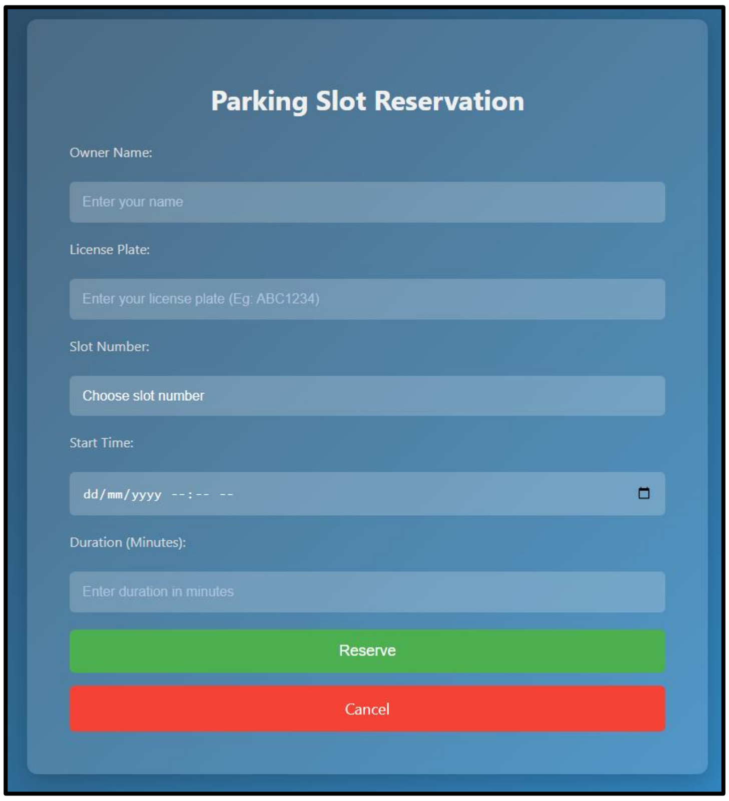


Figure 4.26: The Overview of Reservation Page (reservation.php)


Parking Slot Reservation

Owner Name:

Enter your name

License Plate:

Enter your license plate (Eg: ABC1234)

Slot Number:

Choose slot number

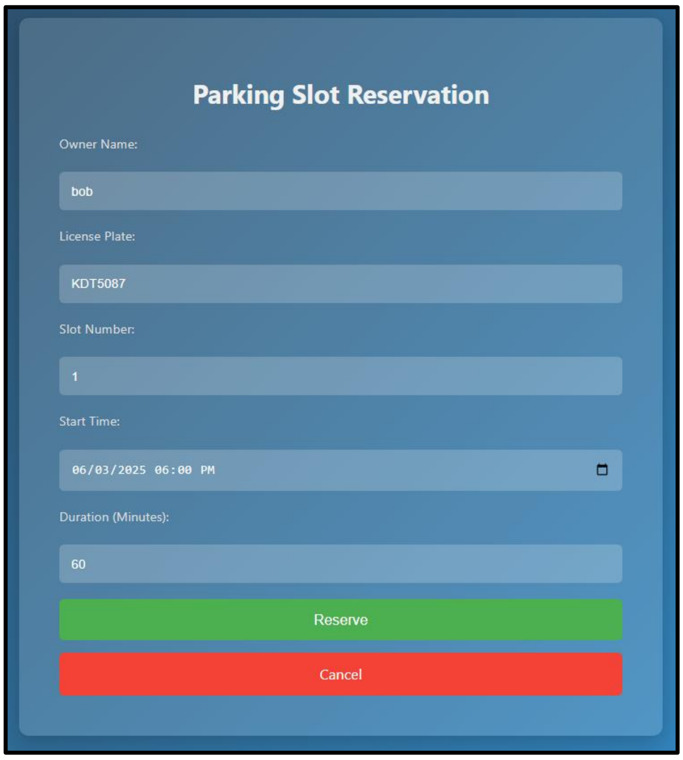
Start Time:

dd/mm/yyyy --:-- --

Duration (Minutes):

Enter duration in minutes

Figure 4.27: Zoomed in View of Reservation Page


Parking Slot Reservation

Owner Name:

bob

License Plate:

KDT5087

Slot Number:

1

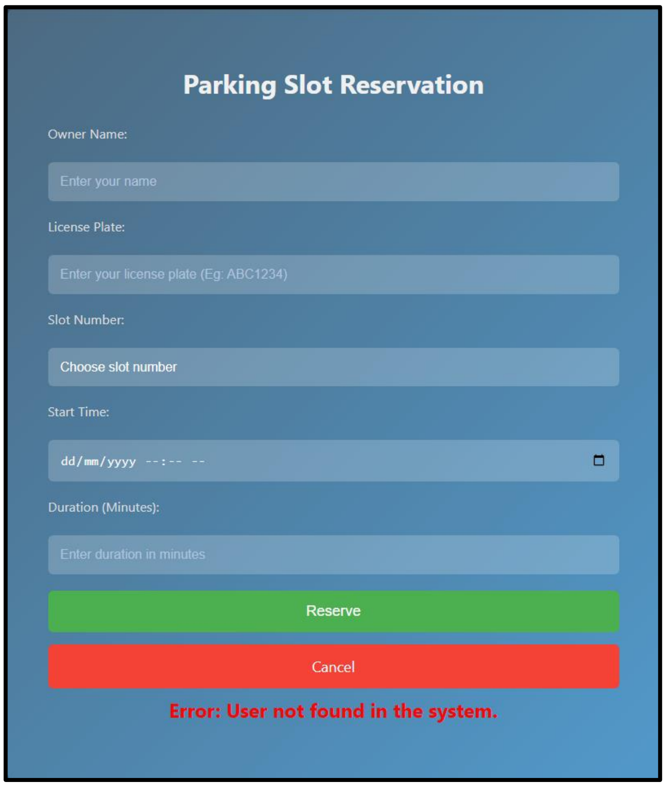
Start Time:

06/03/2025 06:00 PM

Duration (Minutes):

60

Figure 4.28: Demonstration of Attempting to Reserve a Parking Slot with an Unregistered Owner Name (Case-Sensitive Input Required)



Parking Slot Reservation

Owner Name:

Enter your name

License Plate:

Enter your license plate (Eg: ABC1234)

Slot Number:

Choose slot number

Start Time:

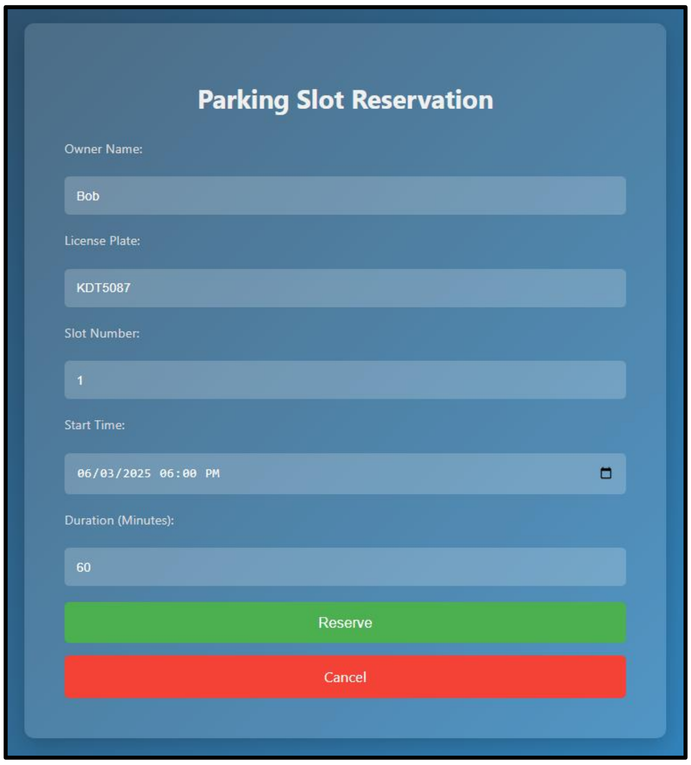
dd/mm/yyyy --:-- --

Duration (Minutes):

Enter duration in minutes

Error: User not found in the system.

Figure 4.29: Pop-Up Error Message Indicating User Not Found in the System



Parking Slot Reservation

Owner Name:

Bob

License Plate:

KDT5087

Slot Number:

1

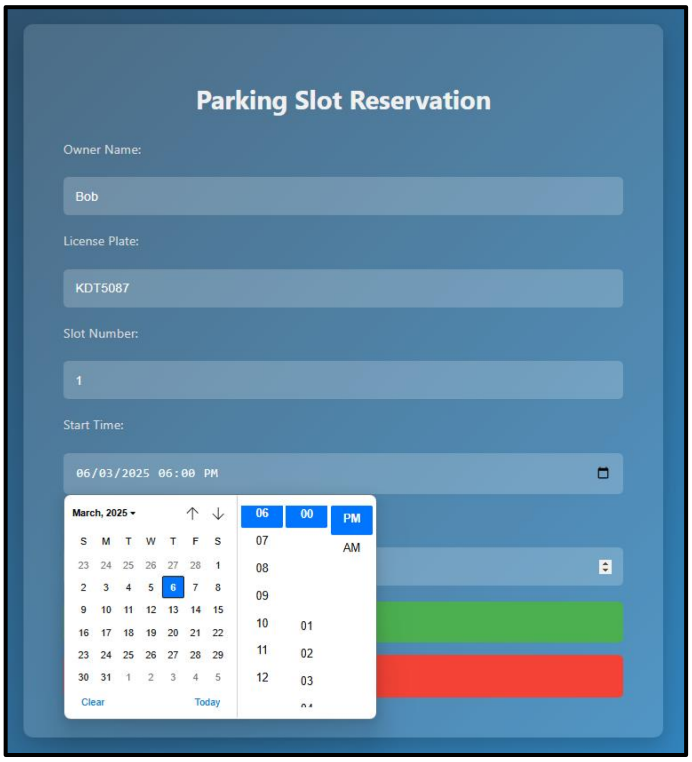
Start Time:

06/03/2025 06:00 PM

Duration (Minutes):

60

Figure 4.30: Demonstration of Attempting to Reserve a Parking Slot with a Registered User (Case-Sensitive) and Filling in Other Information


Parking Slot Reservation

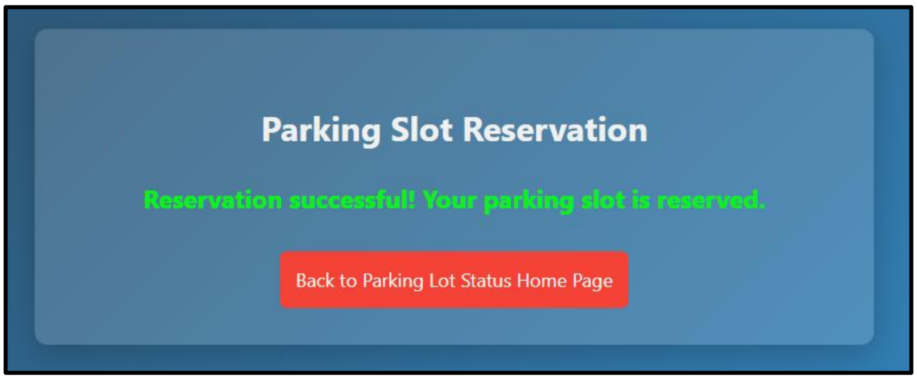
Owner Name: Bob

License Plate: KDT5087

Slot Number: 1

Start Time: 06/03/2025 06:00 PM

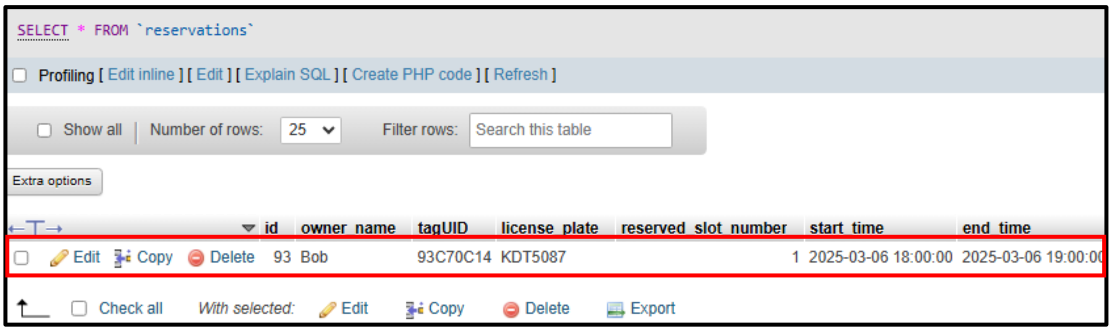
Figure 4.31: Selecting Reservation Date and Time


Parking Slot Reservation

Reservation successful! Your parking slot is reserved.

Figure 4.32: Pop-up Message Indicating Reservation Successful

Figure 4.33 shows that the 'reservations' table in the database is automatically updated with reservation details when the correct owner name is provided. The tag UID is filled in automatically once the system detects the correct owner name, while the user must manually enter the license plate since one user may have multiple vehicles. If the user provides the correct owner name, it is assumed that they own the RFID tag, as each name is unique and set by the user during the sign-up process.



```
SELECT * FROM `reservations`
```

id	owner_name	tagUID	license_plate	reserved_slot_number	start_time	end_time
93	Bob	93C70C14	KDT5087	1	2025-03-06 18:00:00	2025-03-06 19:00:00

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all
Number of rows: 25
Filter rows: Search this table
Extra options
Edit
Copy
Delete
Check all
With selected:
Edit
Copy
Delete
Export

Figure 4.33: Database "reservations" Table Automatically Updated with Reservation Details

Figure 4.34 shows that when the reservation time is reached, the reserved slot (Slot 1) changes its status to RESERVED, indicated by a yellow parking slot box. When the user parks in the reserved slot, its status changes to OCCUPIED, as shown in Figure 4.35. After the user leaves the slot and the reservation expires, the slot status returns to AVAILABLE, as shown in Figure 4.36.

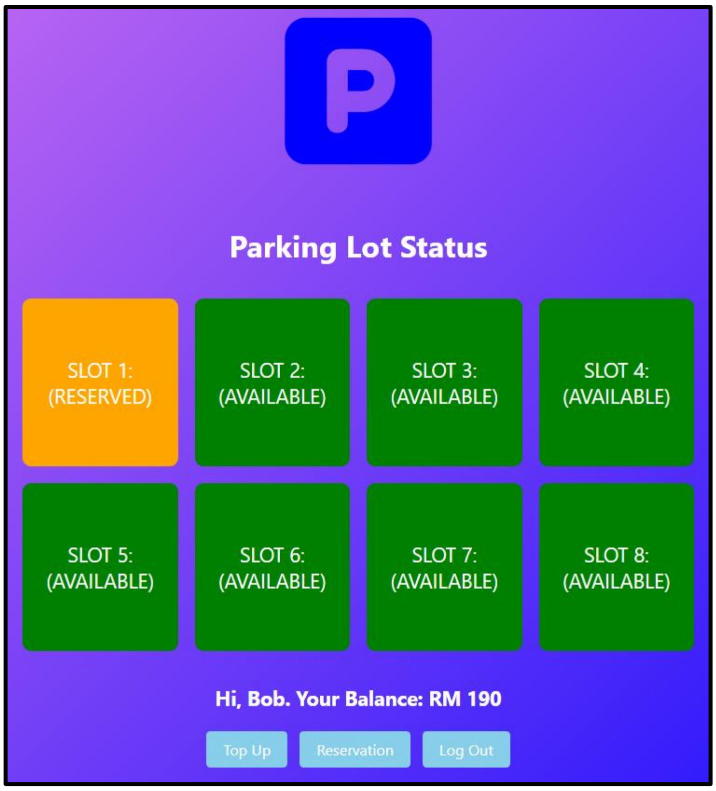


Figure 4.34: Parking Lot Status Monitoring Page Automatically Updates Slot 1 Status to "RESERVED" when Reaching the Reservation Time

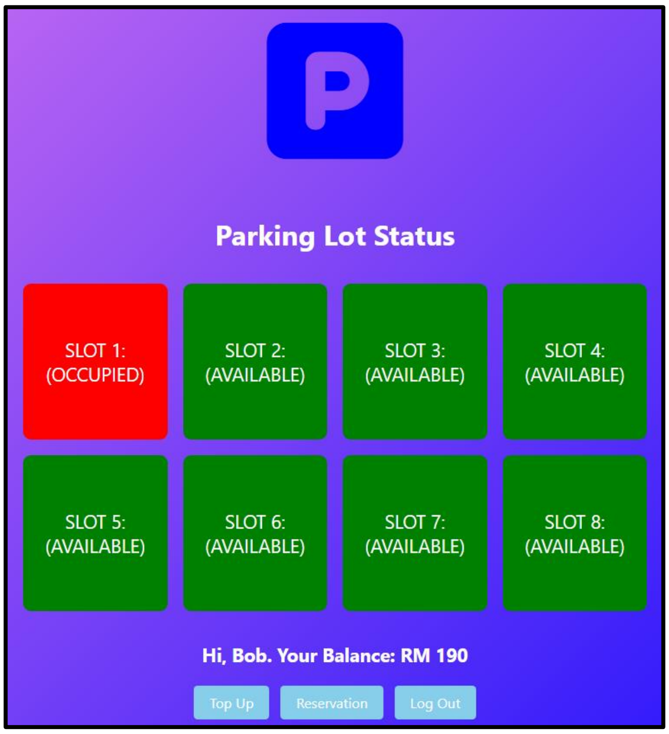


Figure 4.35: Parking Lot Status Monitoring Page Automatically Updates Slot 1 Status to "OCCUPIED" when the User Parks in the Reserved Slot

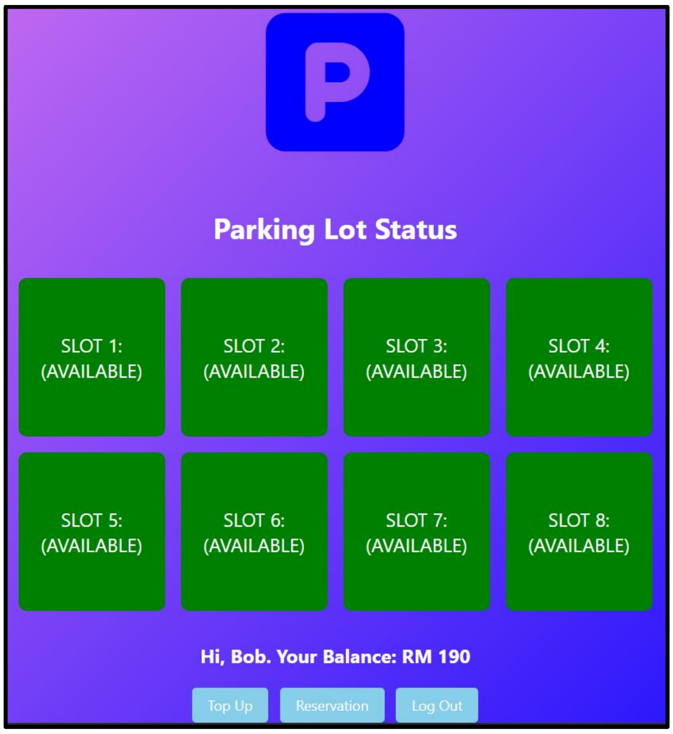
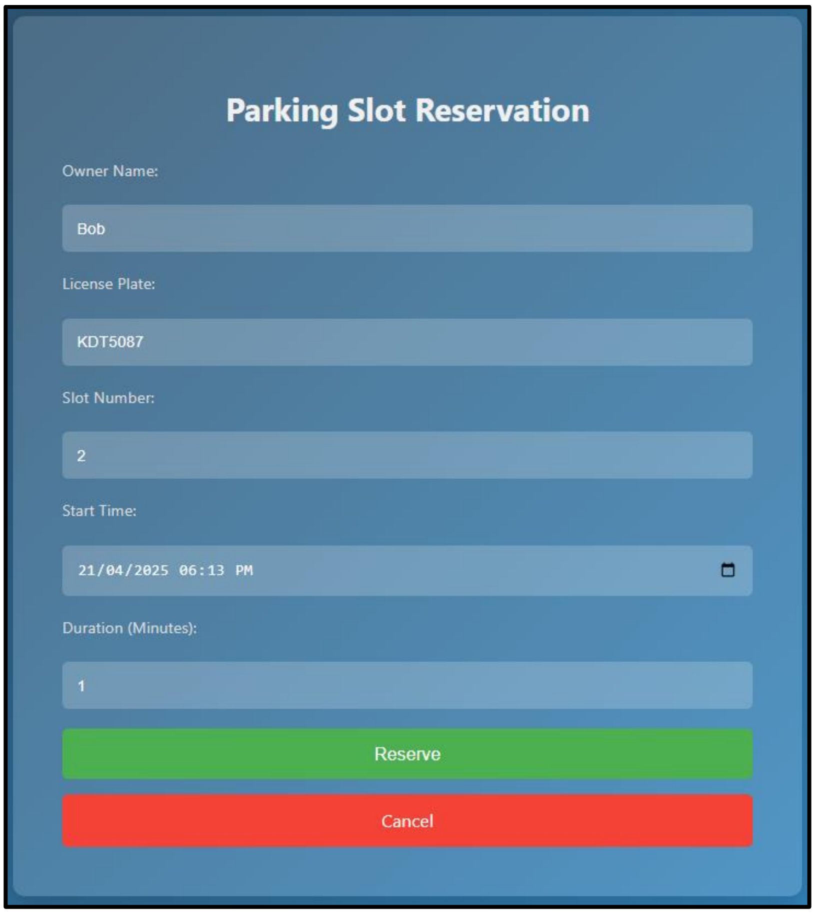


Figure 4.36: Parking Lot Status Monitoring Page Automatically Updates Slot 1 Status to "AVAILABLE" After Reservation Time Passes (No Vehicle in Slot 1)

Figure 4.37 shows a user attempting to reserve a parking slot (Slot 2) that is currently OCCUPIED. Since it is unknown when the user will leave the slot, the reservation cannot be made. Figure 4.38 displays an error message prompting the user to select another slot that is marked as AVAILABLE. Next, Figure 4.39 shows Alice attempting to reserve Slot 1 on March 6, 2025, from 6:30 PM to 7:30 PM. However, as shown in Figure 4.40, the reservation fails because Bob has already reserved that slot from 6:00 PM to 7:00 PM (refer to Figure 4.33). This causes a reservation conflict. Alice can either choose a different time or select another available slot.


Parking Slot Reservation

Owner Name: Bob

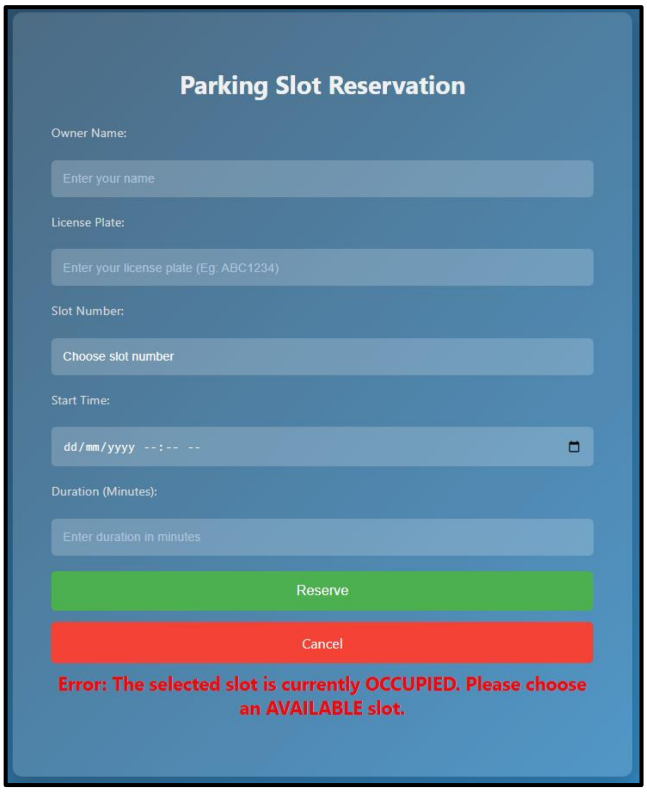
License Plate: KDT5087

Slot Number: 2

Start Time: 21/04/2025 06:13 PM

Duration (Minutes): 1

Figure 4.37: Demonstration of an Attempt to Reserve a Parking Slot that Is Currently Occupied (Not Available)


Parking Slot Reservation

Owner Name: Enter your name

License Plate: Enter your license plate (Eg: ABC1234)

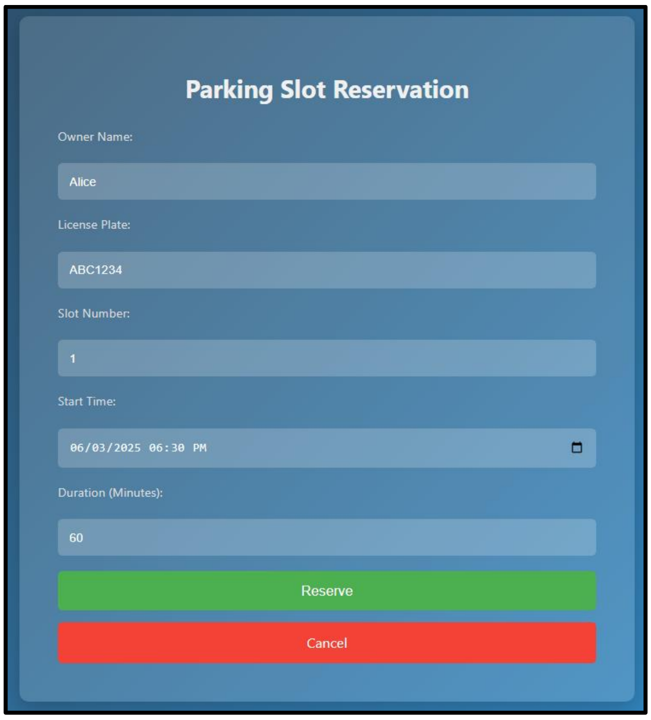
Slot Number: Choose slot number

Start Time: dd/mm/yyyy --:-- --

Duration (Minutes): Enter duration in minutes

Error: The selected slot is currently OCCUPIED. Please choose an AVAILABLE slot.

Figure 4.38: Pop-Up Error Message Indicating the Selected Slot Is Not Available for Reservation


Parking Slot Reservation

Owner Name: Alice

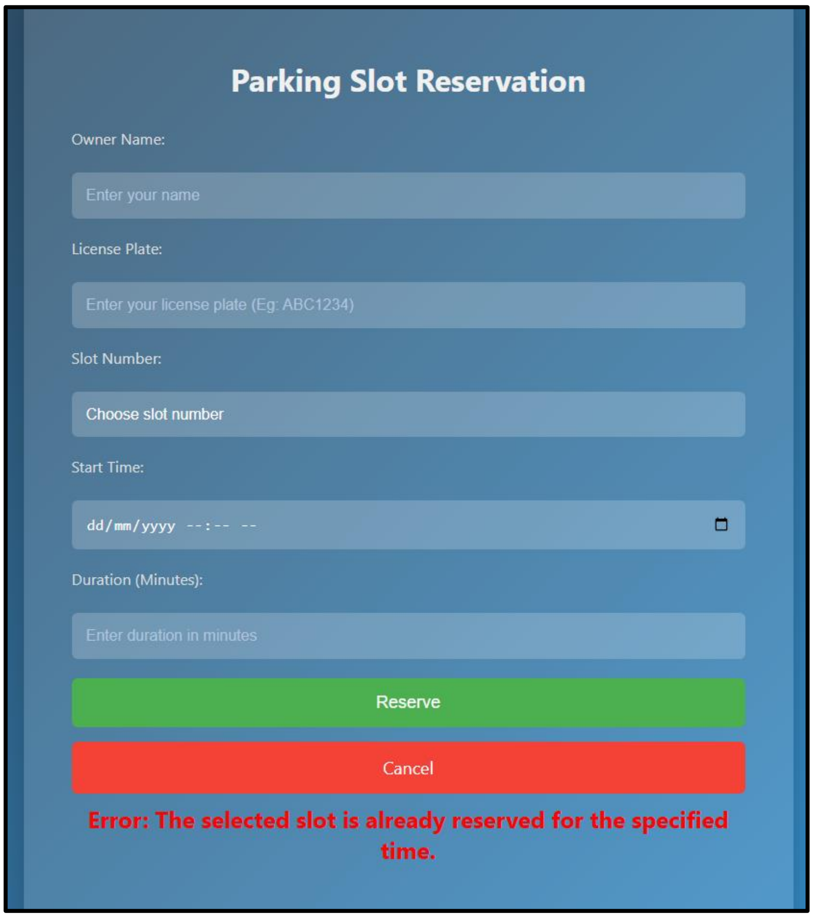
License Plate: ABC1234

Slot Number: 1

Start Time: 06/03/2025 06:30 PM

Duration (Minutes): 60

Figure 4.39: Demonstration of Attempting to Reserve a Parking Slot Already Reserved for the Specified Time (Refer Figure 4.33)



Parking Slot Reservation

Owner Name:

Enter your name

License Plate:

Enter your license plate (Eg: ABC1234)

Slot Number:

Choose slot number

Start Time:

dd/mm/yyyy --:-- --

Duration (Minutes):

Enter duration in minutes

Error: The selected slot is already reserved for the specified time.

Figure 4.40: Pop-Up Error Message Indicating Time Conflict with an Existing Reservation for the Selected Slot (Refer Figure 4.33)

4.3 Hardware Prototype

4.3.1 Overview

Figure 4.41 shows the top view while Figure 4.42 shows the front view of the hardware prototype, built using a two-layer plywood platform with the wiring hidden in the space between Layer 1 and Layer 2. The roofs of the eight parking slots are custom-designed and 3D-printed. The two ESP32 microcontrollers are soldered onto a stripboard and embedded inside a black PVC box to protect them from the external environment, as shown in Figure 4.43. Additionally, the entrance and exit RFID readers are positioned at an appropriate distance from each other to avoid electromagnetic wave interference. RFID readers are prone to signal interference when using long wires, so the wires need to be as short as possible, ideally less than 15 cm.

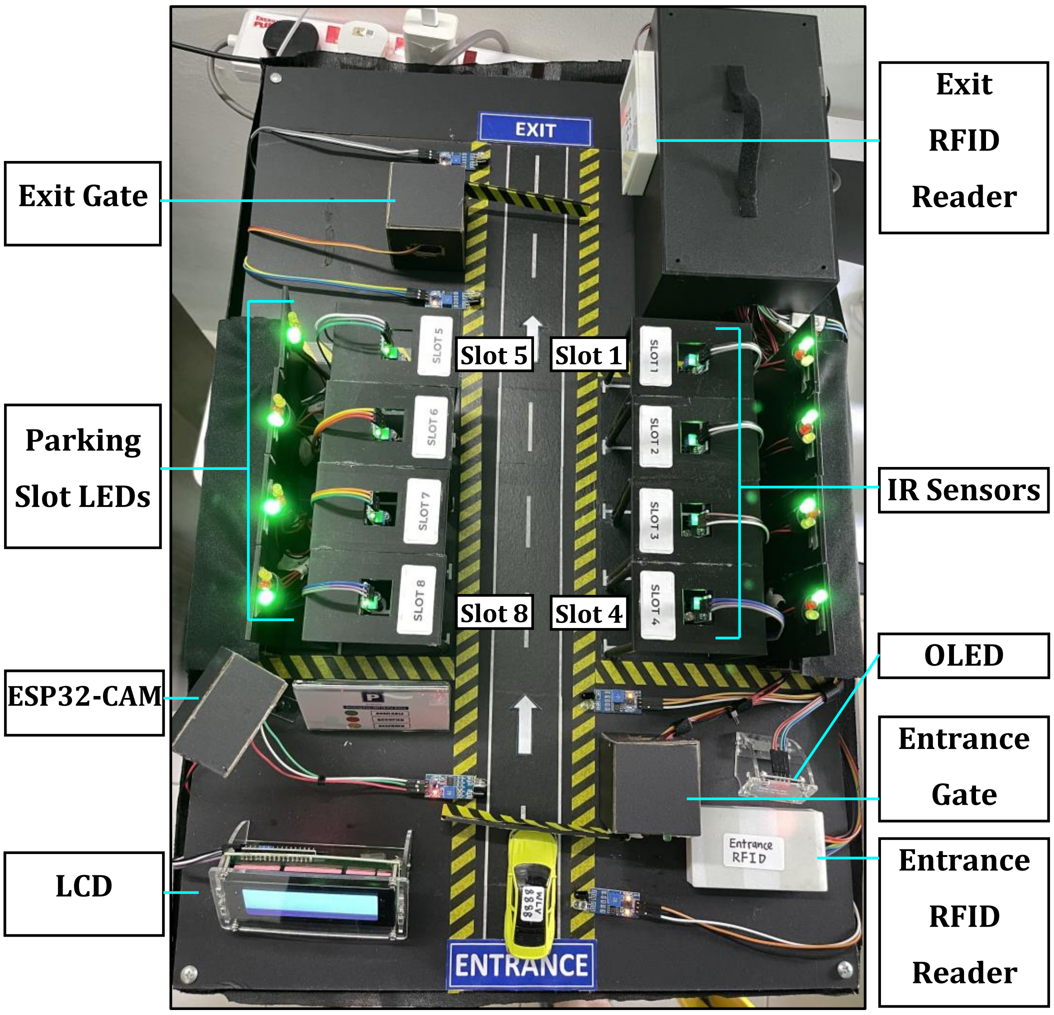


Figure 4.41: Top View of the Hardware Prototype

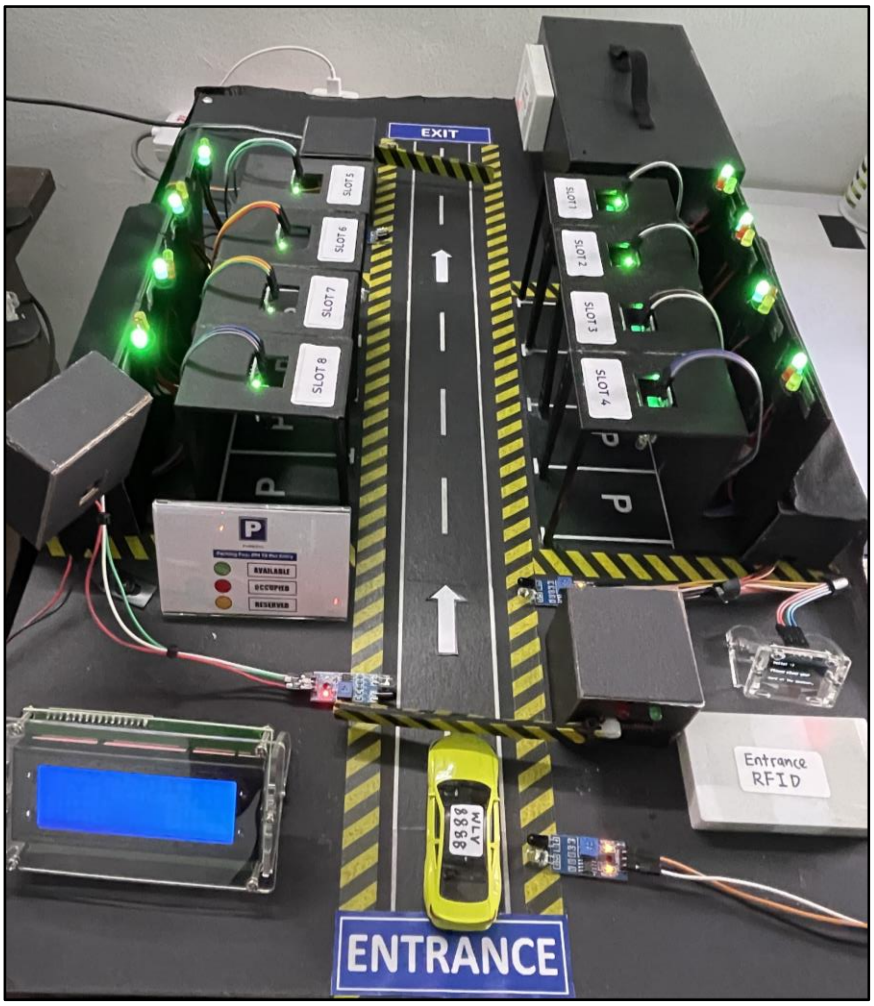


Figure 4.42: Front View of the Hardware Prototype

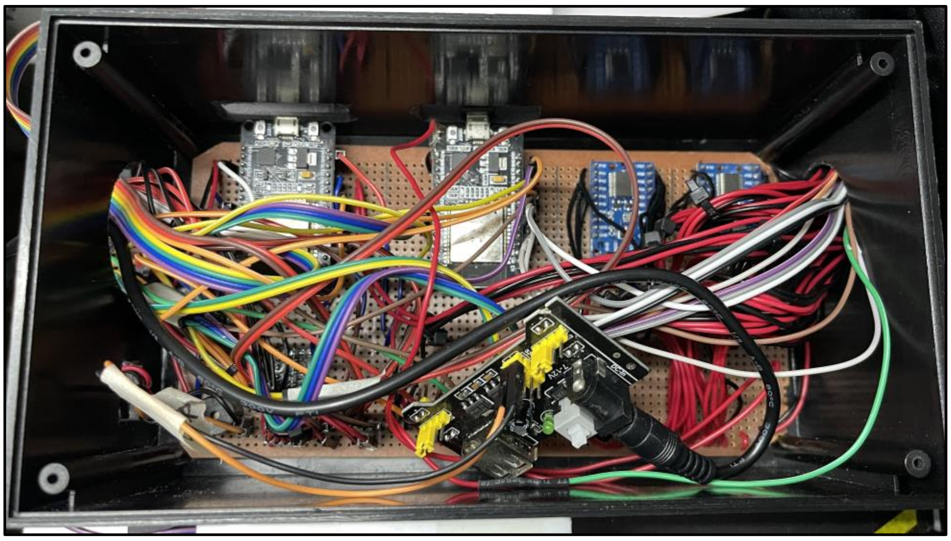


Figure 4.43: Two ESP32 Microcontrollers Soldered onto a Stripboard and Embedded Inside a Black PVC Box

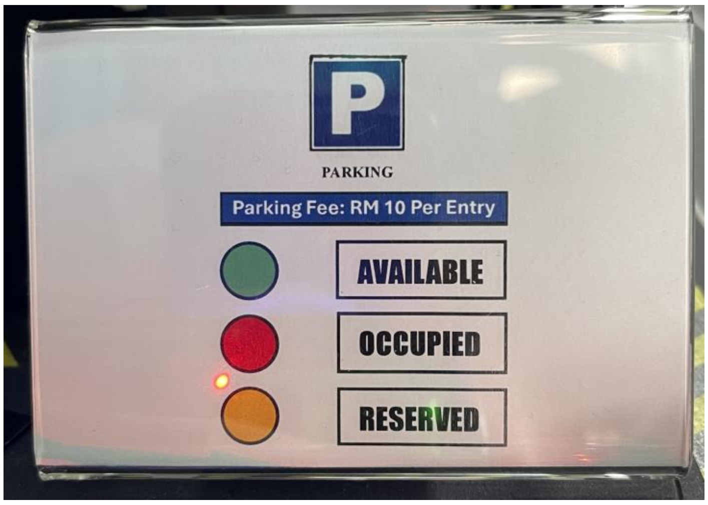
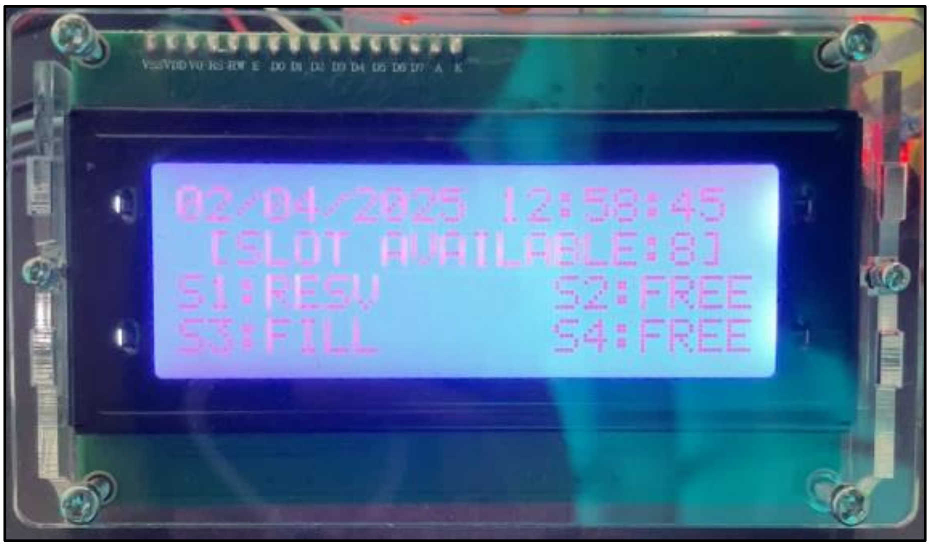


Figure 4.44: Entrance Display Board Showing Parking Fees and LED Light Meanings

4.3.2 LCD Display Results

The time displayed on the LCD in Figure 4.45 is obtained using the NTP server. The two bottom rows of the LCD show the status of each slot, where RESV means reserved, FILL means occupied, and FREE means available. It continuously polls the ‘parking_status’ table in the database to retrieve the latest status.



```
02/04/2025 12:58:45
 [SLOT AVAILABLE:8]
S1:RESV      S2:FREE
S3:FILL      S4:FREE
```

Figure 4.45: LCD Display Showing the Current Time, Number of Available Parking Slots, and the Status of Each Slot

4.3.3 <u>ESP32-CAM Results</u>

The ESP32-CAM, as shown in Figure 4.46, captures a vehicle image when the IR sensor detects movement, encodes the image in Base64 format, and sends it through an HTTP POST request to a Google Apps Script. The script decodes the image and saves it to a Google Drive folder named ESP32_CAM_FYP, as shown in Figure 4.47. The captured image is automatically renamed based on the timestamp when it is taken.

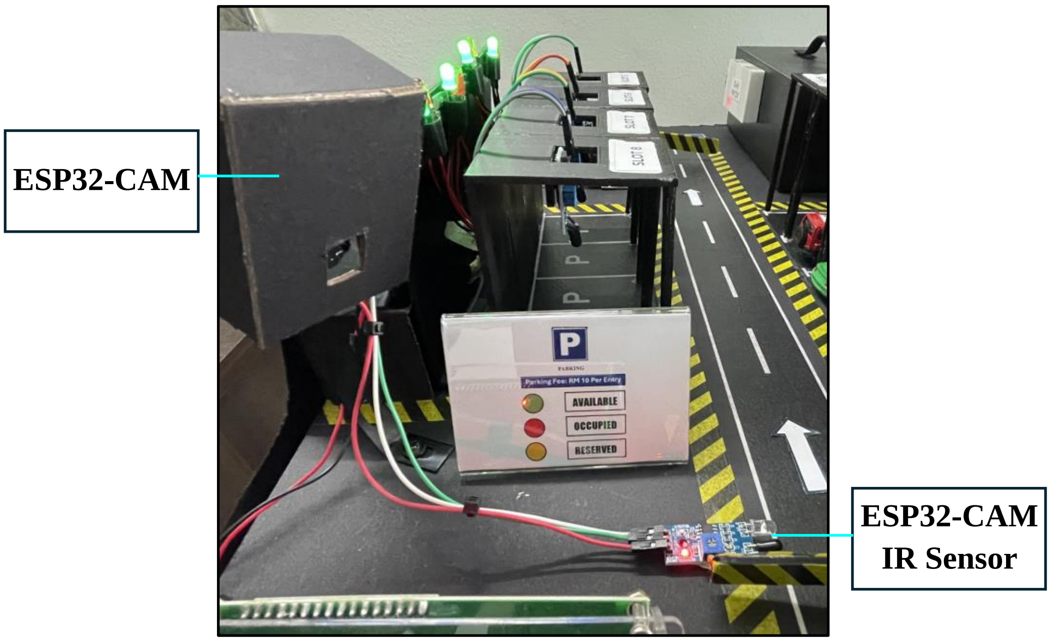


Figure 4.46: ESP32-CAM and IR Sensor Image-Capturing System at the Entrance

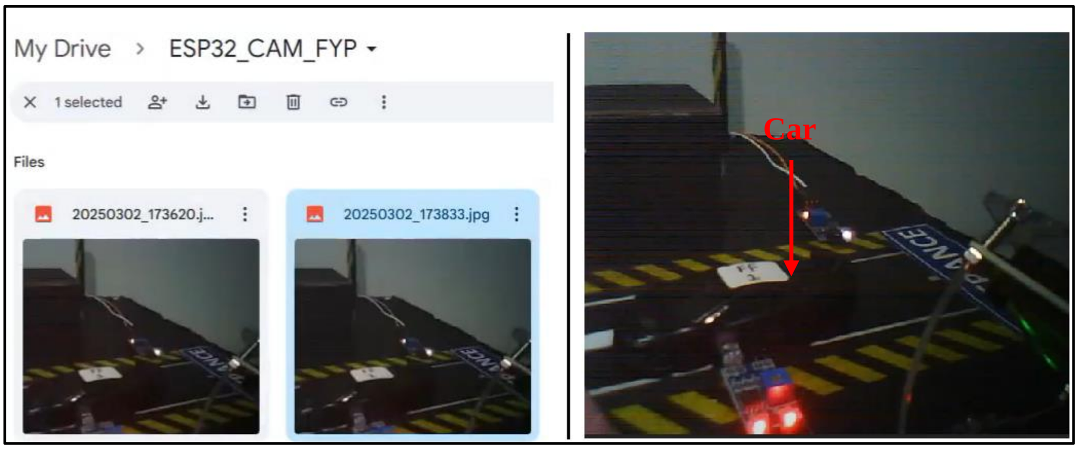


Figure 4.47: Captured Image Renamed Using the Timestamp and Stored in Google Drive

4.3.4 Entrance and Exit Gate Results

A servo motor is used to simulate the entrance and exit gate, as shown in Figure 4.48 and Figure 4.49, respectively. The gate opens when access is authorized, the account balance is sufficient, and the parking lot is not full. When the user is authorized, the green LED lights up, and the buzzer makes a short beep twice. Conversely, if access is unauthorized or the account balance is insufficient, the red LED lights up, and the buzzer makes a long beep. After the gate opens, it will only close once the car has passed through the second IR sensor to prevent it from closing too early and hitting the car. At the exit gate, the account balance is not checked. The gate opens as long as the RFID tag is valid.

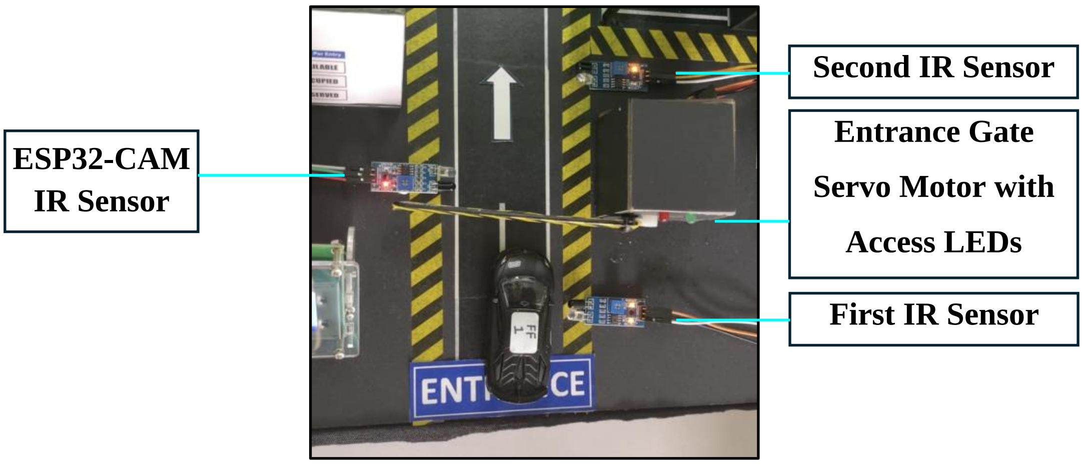


Figure 4.48: Entrance Gate Servo Motor with Access LEDs

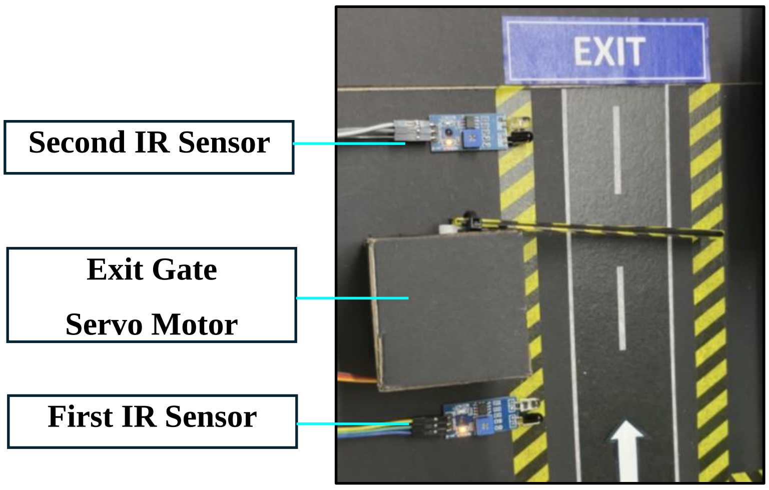
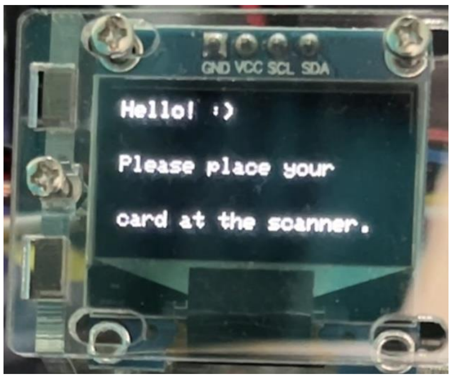


Figure 4.49: Exit Gate Servo Motor

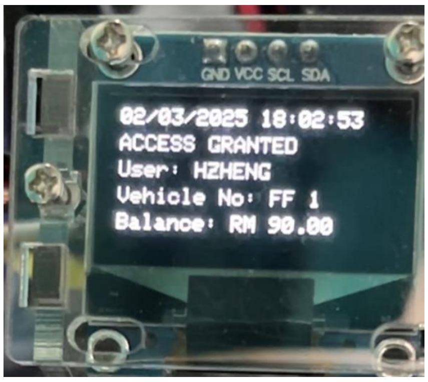
4.3.5 OLED Display Results

The OLED display shows the “Welcome Page” by default, as shown in Figure 4.50. When the user scans the RFID tags, the RFID scanning messages will be shown. For example, if access is authorized, it displays the timestamp, owner name, license plate, and account balance, as shown in Figure 4.51. If access is denied, it shows whether the RFID tag is unregistered or if the account balance is insufficient, as shown in Figure 4.52 and Figure 4.53.



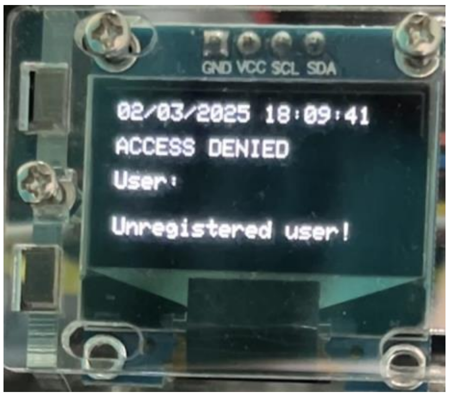
```
Hello! :)
Please place your
card at the scanner.
```

Figure 4.50: OLED Display Showing Welcome Page



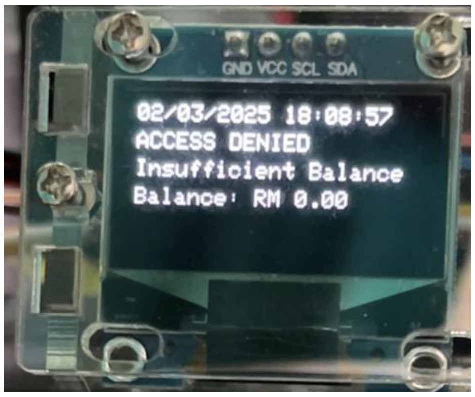
```
02/03/2025 18:02:53
ACCESS GRANTED
User: HZHENG
Vehicle No: FF 1
Balance: RM 90.00
```

Figure 4.51: OLED Display Showing Access Authorized Messages



```
02/03/2025 18:09:41
ACCESS DENIED
User:
Unregistered user!
```

Figure 4.52: OLED Display Showing Access Denied Messages



```
02/03/2025 18:08:57
ACCESS DENIED
Insufficient Balance
Balance: RM 0.00
```

Figure 4.53: OLED Display Showing Insufficient Balance Messages

4.3.6 Parking Lot Results

This project includes a total of eight parking slots, each equipped with an IR sensor and three LEDs to indicate the slot's status. The red LED indicates OCCUPIED, the green LED indicates AVAILABLE, and the yellow LED indicates RESERVED. Figure 4.54 shows the parking slots 1, 2, 3, and 4, while Figure 4.55 shows the parking slots 5, 6, 7, and 8. All the parking slots were custom-designed and produced using 3D printing. When a vehicle is parked in a slot, the obstacle LED on the IR sensor lights up, and the second ESP32 sends an HTTP POST request to update the 'parking_status' table in the database. The LCD and webpage continuously poll this table to display the latest parking slot status. Additionally, the second ESP32 polls the table and sends signals to the PCF8575 I/O expanders to light up the correct parking slot LED.

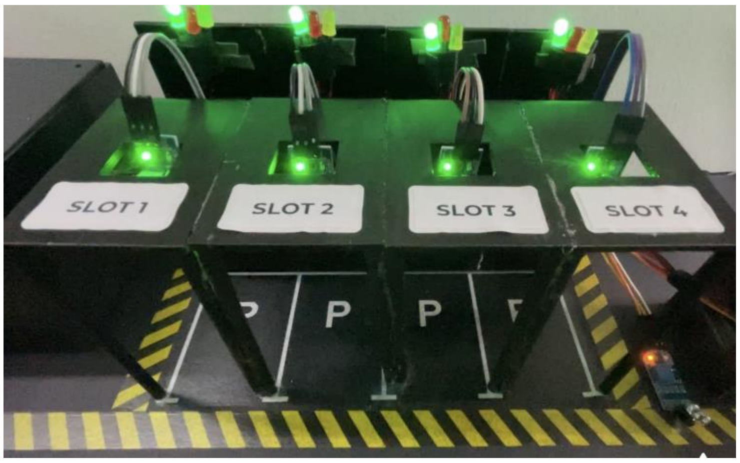


Figure 4.54: Parking Slots 1, 2, 3, and 4

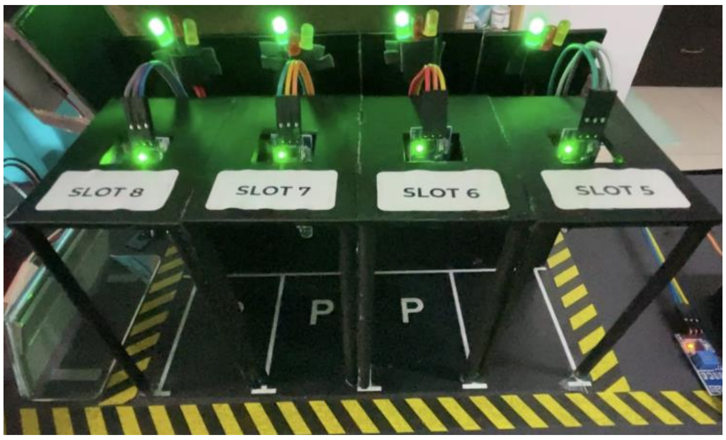


Figure 4.55: Parking Slots 5, 6, 7, and 8

Figure 4.56 shows that all the parking slots are available, as all green parking slot LEDs light up. The user may park in any available slot.

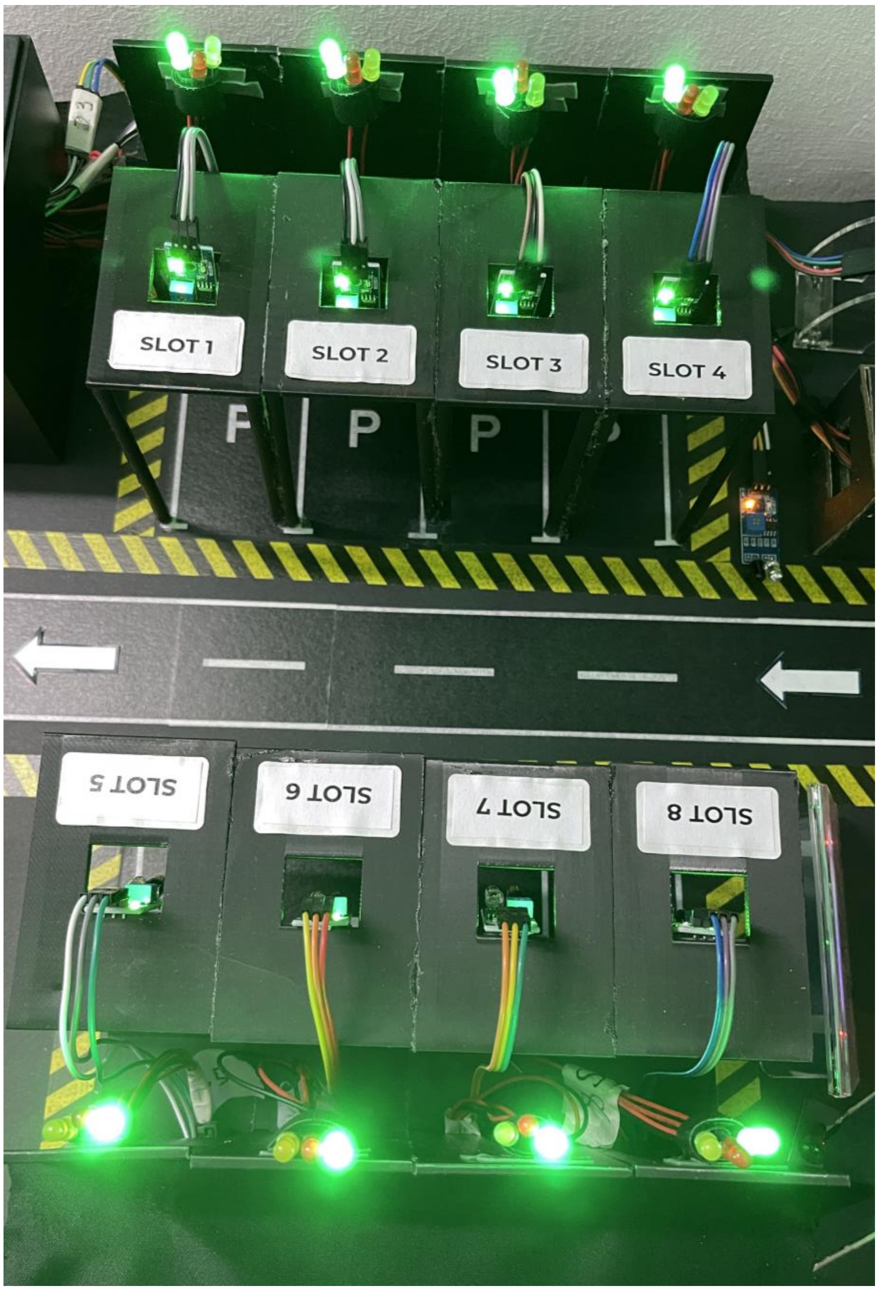


Figure 4.56: Parking Lot when All Slots are Available

Figure 4.57 shows that parking slots 2, 4, 6, and 8 are occupied, as the red parking slot LEDs light up. The user can only park in parking slots 1, 3, 5, and 7, where the slots are available, as the green parking slot LEDs light up.

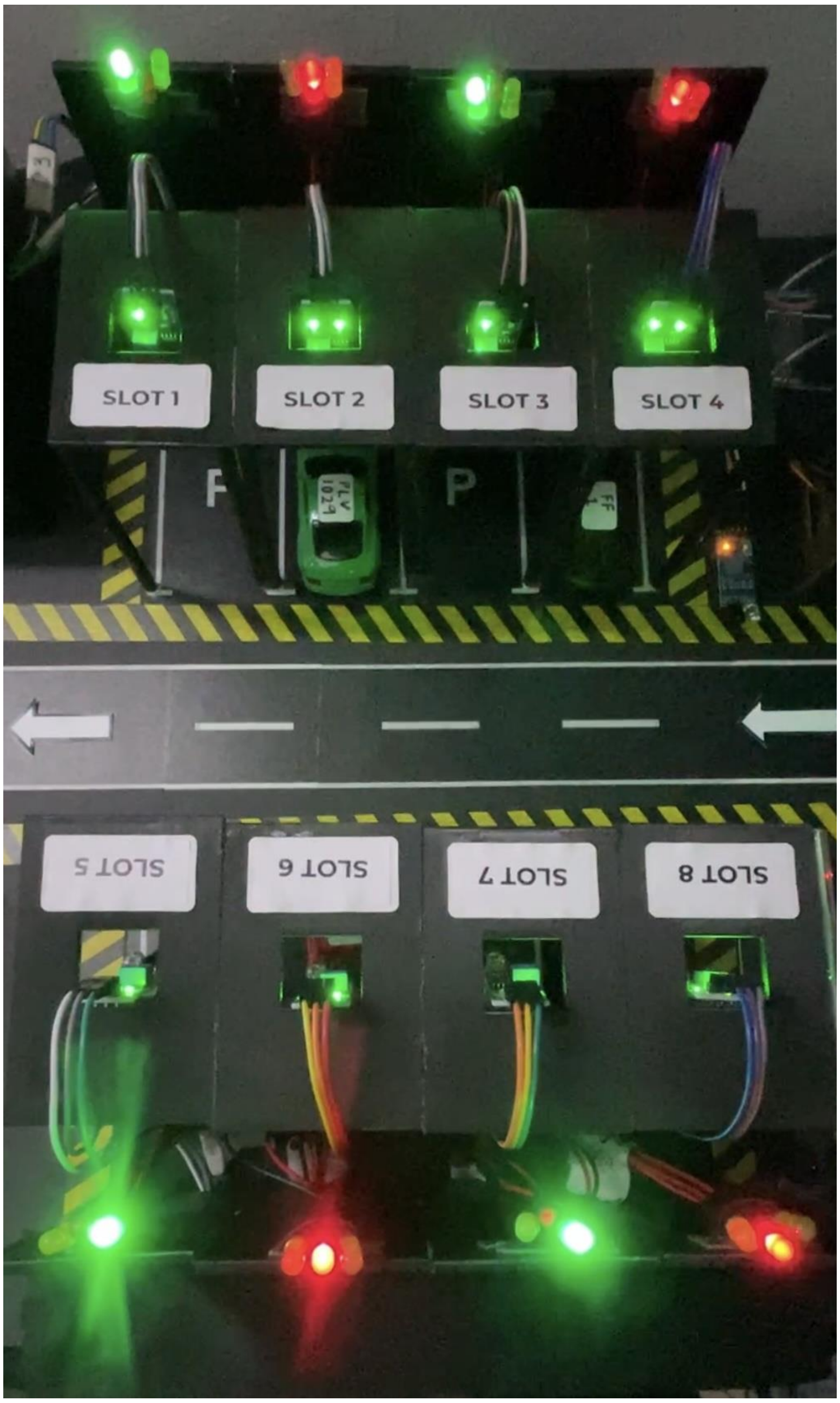


Figure 4.57: Parking Lot when Slots 1, 3, 5, and 7 are Available, while Slots 2, 4, 6, and 8 are Occupied

Figure 4.58 shows that all the parking slots are occupied, as all red parking slot LEDs light up. This means the parking lot is full, and the user must wait for a vehicle to exit before he can enter the parking lot. The LCD displays that the number of available parking slots is zero, along with a "SORRY, PARKING FULL…" message, as shown in Figure 4.59.

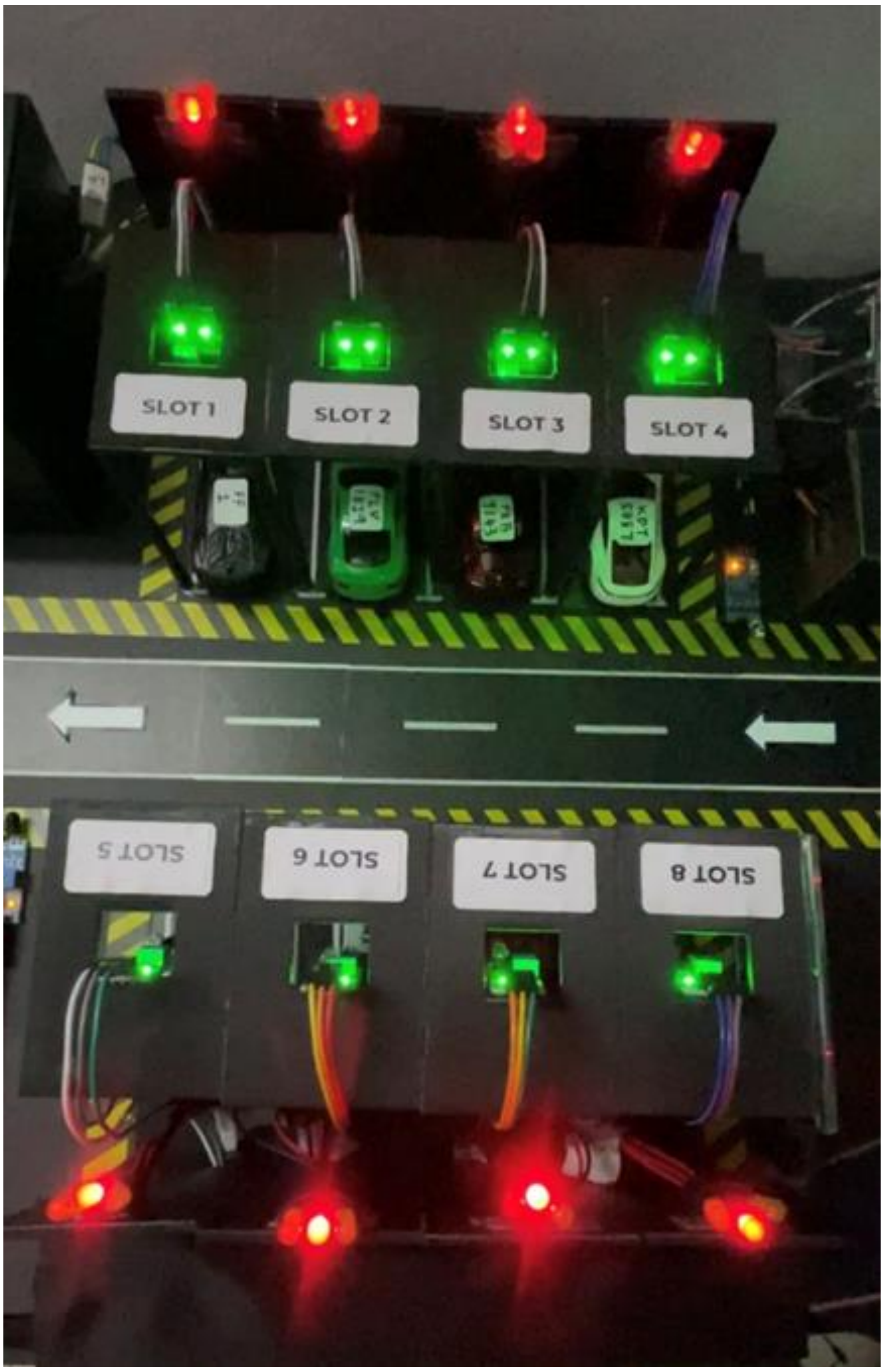
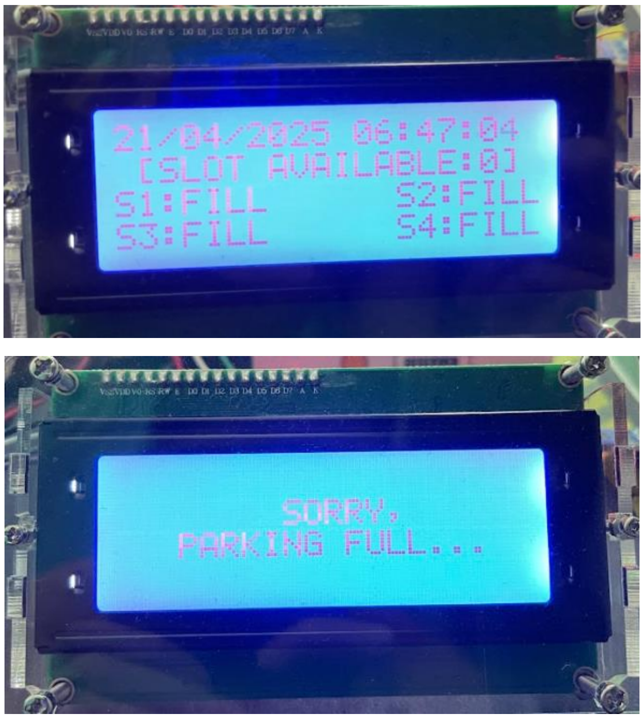


Figure 4.58: Parking Lot when All Slots are Occupied (Parking Full)



```
21/04/2025 06:47:04
 [SLOT AVAILABLE:0]
S1:FILL       S2:FILL
S3:FILL       S4:FILL
```

```
       SORRY,
  PARKING FULL...
```

Figure 4.59: LCD Messages when All Parking Slots are Occupied (Parking Full)

Referring to Figure 4.33, Bob has reserved Slot 1 from 6:00 PM to 7:00 PM. Figure 4.60 shows that Slot 1 yellow LED lights up when reaching the reservation time, which is 6:00 PM. This indicates that Slot 1 is reserved. The user can enter the parking lot as usual at the reservation time by scanning their RFID tags at the entrance. Once the user parks in Slot 1, the red LED lights up to show that the slot is occupied, as shown in Figure 4.61. After the user leaves Slot 1 when the reservation period expires, the green LED lights up, as shown in Figure 4.62.

Besides that, if the user leaves the slot during the reservation period, the yellow LED will light up again. This accounts for situations where the user may need to readjust their vehicle or leave temporarily, without losing the reserved slot. Additionally, if another user mistakenly parks in the reserved slot and realizes the mistake, they can leave the slot, at which point the yellow LED will light up again to indicate that the slot is still reserved.

Next, if a user parks in a reserved slot without having made a reservation, their vehicle will be clamped by security. Security personnel can verify unauthorized parking by cross-checking the vehicle's license plate, which is provided during the reservation process.

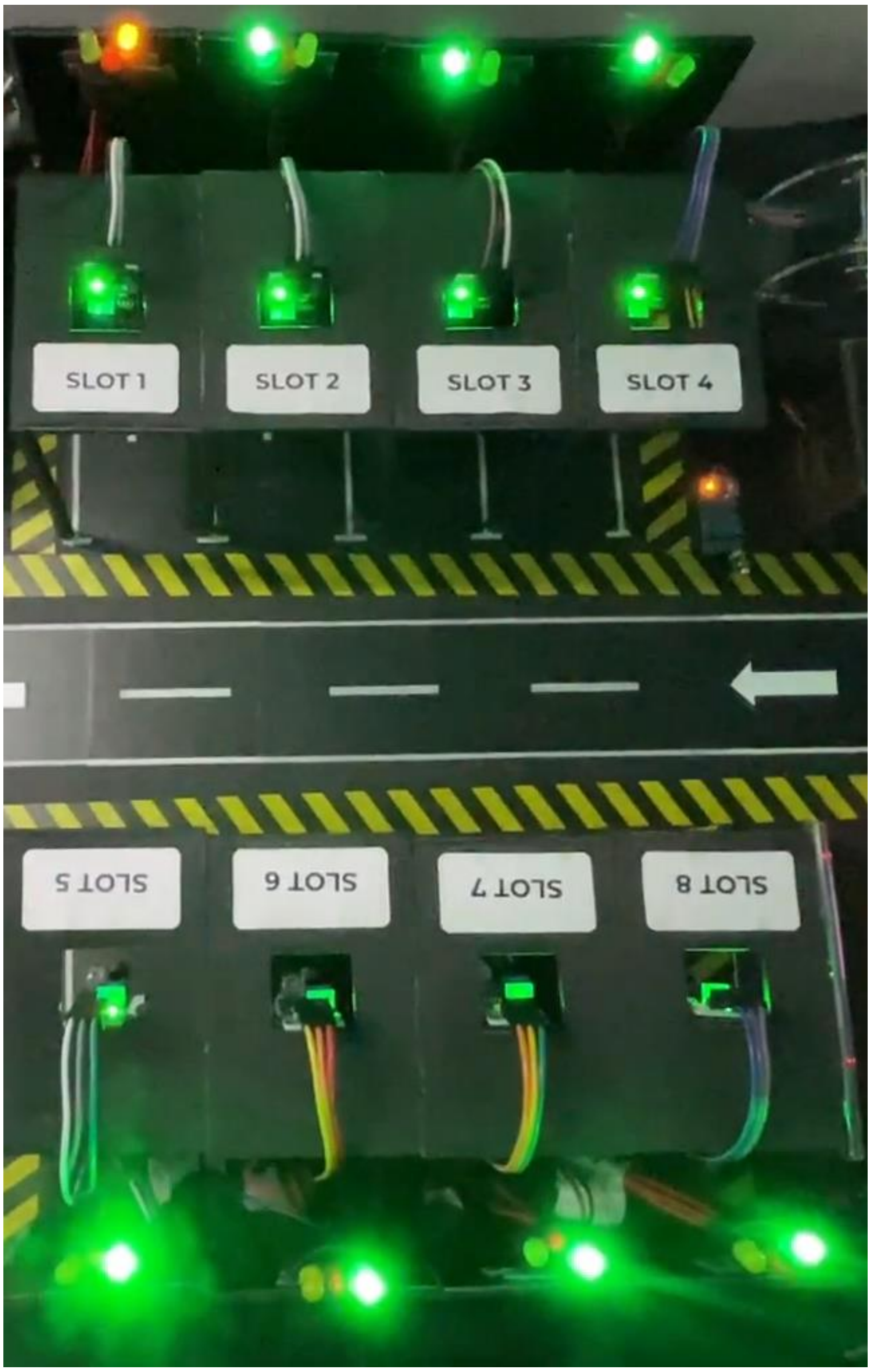


Figure 4.60: Slot 1 Yellow LED Lights Up at the Reserved Time to Indicate the Slot Is Reserved

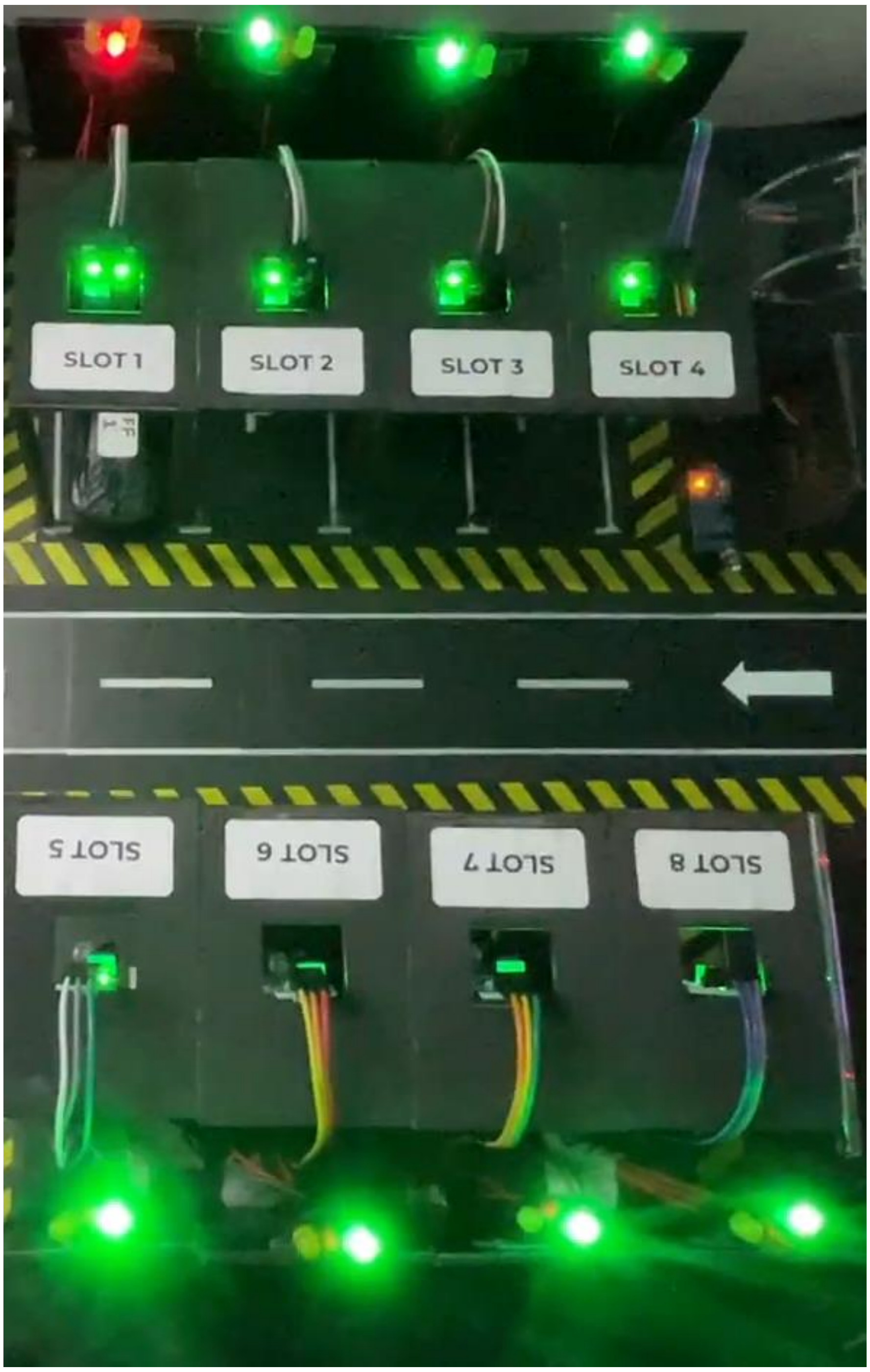


Figure 4.61: Slot 1 Red LED Lights Up After the User Parks in the Reserved Slot

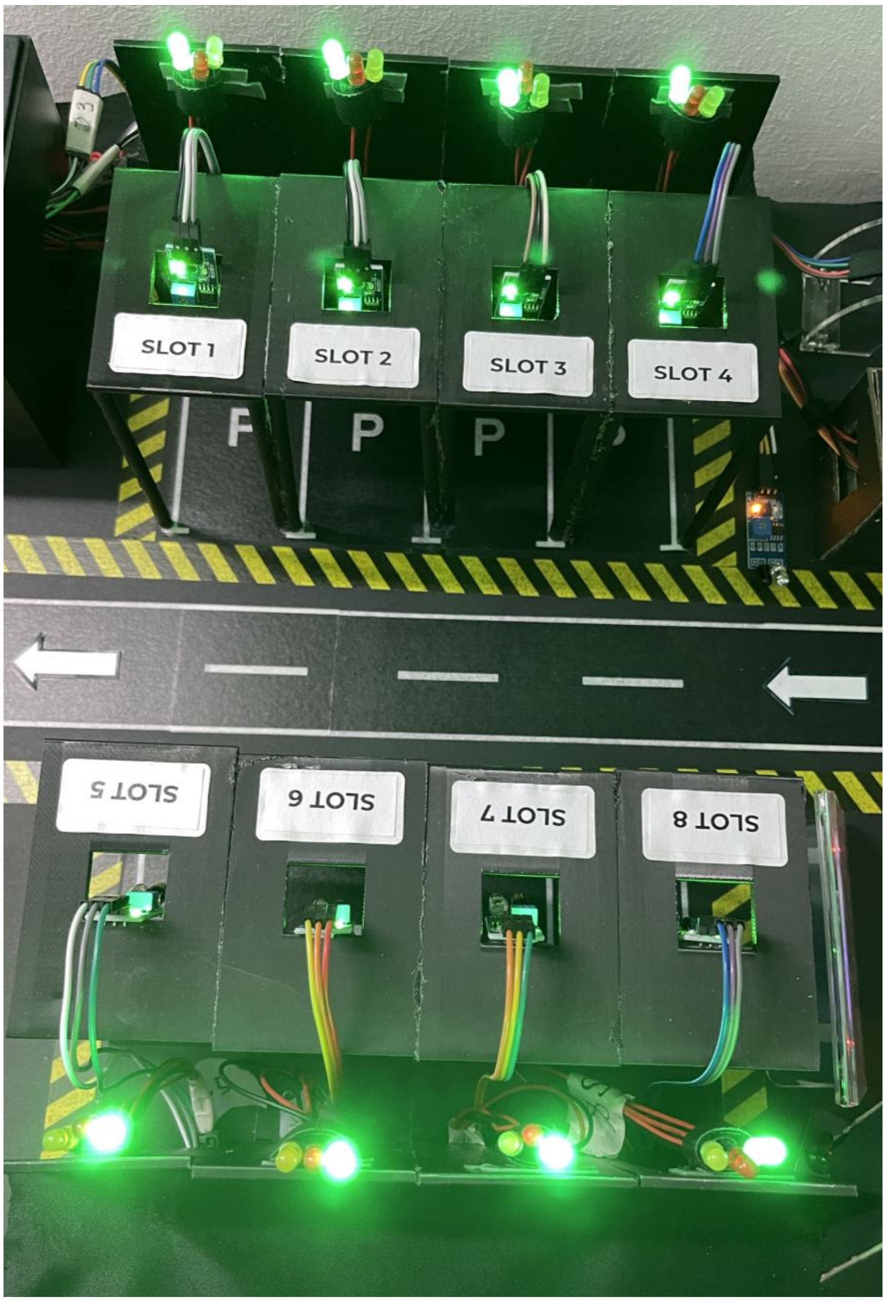


Figure 4.62: Slot 1 Green LED Lights Up After Reservation Time Passes

4.4 Detection Accuracy

After completing the hardware prototype, a series of tests were conducted to evaluate the detection accuracy of the IR sensor under different scenarios. Table 4.1 shows the detection accuracy results. Case 1 tests whether the IR sensor can detect a vehicle as it enters the parking lot. Case 2 tests whether the IR sensor can detect a vehicle as it exits the parking lot. Case 3 tests whether the ESP32-CAM successfully captures an image of the vehicle when entering the parking lot. This depends on whether the IR sensor detects the vehicle and triggers the ESP32-CAM to take a photo. Case 4 tests whether the IR sensor can detect a vehicle parked in the designated parking slot.

The results in Table 4.1 indicate that vehicle detection at the entrance, exit, and parking slot achieved 100 % accuracy. This high accuracy is attributed to the fact that vehicles must stop at the entrance gate, exit gate, and parking slots, allowing sufficient time for the IR sensor to detect them. However, the ESP32-CAM achieved 93.33 % accuracy due to challenges in detecting vehicles moving at high speeds. The IR sensor requires the vehicle to slow down slightly to ensure proper detection and image capture. Overall, the core function of the system demonstrated a high level of accuracy, which proved its effectiveness in vehicle detection.

Table 4.1: Detection Accuracy of the IR Sensor

Case	**Scenario**	**Number of Cars Tested**	**Number of Cars Detected**	**Detection Accuracy**
1	Vehicle Entering the Parking Lot	30	30	100 %
2	Vehicle Exiting the Parking Lot	30	30	100 %
3	ESP32-CAM Capturing the Vehicle Image	30	28	93.33 %
4	Vehicle Parking in the Slot	30	30	100 %

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

In conclusion, the proposed Smart Parking System was successfully developed and achieved all its aims and objectives. A real-time parking lot monitoring system was implemented using IR sensors, ESP32 microcontrollers, and RFID for access control. In addition, five webpages were developed using HTML and PHP, including a login page, sign-up page, top-up page, reservation page, and parking lot status monitoring page. All the webpages function well. Users can make reservations through the reservation page before they arrive. All user information, RFID scan logs, reservation details, and parking lot status are successfully stored in a MySQL database. Furthermore, the entrance image-capturing system using the ESP32-CAM was successfully developed. The ESP32-CAM managed to capture vehicle images when the IR sensor detected movement and saved them to Google Drive. The system achieved an accuracy rate of 100 % in vehicle detection in parking slots. Besides that, the smart parking system developed is highly affordable, which is less than RM 300.

However, the system has some disadvantages. Since this is an IoT project, a stable and high-speed internet connection is very important. The webpages, parking slot LEDs, and LCD display need to continuously poll the database to update the latest parking slot status. Similarly, the RFID scanning process also requires internet access to cross-check UID with the database. If the internet is slow, users may have to wait longer before the entrance and exit gates respond. Additionally, the ESP32-CAM has limited image quality since it is a budget camera, and the surrounding environment like lighting can also affect the clarity of the captured images.

5.2 Recommendations

For future improvements, several modifications can be made to further improve the Smart Parking System. Firstly, instead of using multiple IR sensors for each parking slot, a camera-based system with image processing and machine learning could provide more accurate detection while reducing the need for individual sensors. If IR sensors continue to be used, upgrading to higher-quality IR sensors or integrating multiple sensors such as ultrasonic sensors and LiDAR sensors could enhance reliability. Besides that, magnetic sensors could be used since they offer more reliable vehicle detection by responding only to the magnetic field of a car, reducing false triggers from people or animals. Magnetic sensors are usually embedded underground, so they are less affected by environmental conditions such as rain or fog.

Additionally, replacing the ESP32-CAM with a higher-quality camera would result in clearer images, especially under varying lighting conditions. Furthermore, integrating Automatic Number Plate Recognition (ANPR) with RFID and facial recognition could further enhance security and streamline vehicle identification. Moreover, adding a panic button would improve safety by allowing users to request assistance when needed.

Furthermore, using a more stable network connection, such as wired Ethernet could help minimize delays and improve system reliability. Lastly, implementing a dynamic pricing model, where parking fees adjust based on demand, such as lower rates during off-peak hours depending on the number of vehicles in the parking lot, could make the system more efficient and cost-effective for users.

REFERENCES

Bishop, J., 2023. *Difference between sourcing and sinking in a circuit*. [online] CircuitBread. Available at: <https://www.circuitbread.com/tutorials/difference-between-sourcing-sinking> [Accessed 28 November 2024].

BusinessToday., 2023. The Average Malaysian Spends 44 Hours In Traffic In A Month. *BusinessToday*. [online] 27 January. Available at: <https://www.businesstoday.com.my/2023/01/27/the-average-malaysian-spends-44-hours-in-traffic-in-a-month/.> [Accessed 13 August 2024].

Ch'ng, S.F., 2019. *Web-Based Car Parking Slot Monitoring System*. Bachelor’s Degree. Universiti Tunku Abdul Rahman. Available at: <http://eprints.utar.edu.my/id/eprint/3915> [Accessed 21 August 2024].

Daim, N., 2023. 36.3 million vehicles in Malaysia. *New Straits Times*. [online] 6 December. Available at: <https://www.nst.com.my/news/nation/2023/12/987062/363-million-vehicles-malaysia.> [Accessed 13 August 2024].

Elakya, R., Seth, J., Ashritha, P., and Namith, R., 2019. Smart Parking System using IoT. *International Journal of Engineering and Advanced Technology (IJEAT)*, 9(1), pp.6091-6096. https://doi.org/10.35940/ijeat.A1963.109119

Elfaki, A.O., Messoudi, W., Bushnag, A., Abuzneid, S. and Alhmiedat, T., 2023. A Smart Real-Time Parking Control and Monitoring System. *Sensors*, 23(24), p.9741. https://doi.org/10.3390/s23249741

Fahad, E., 2020. *ESP32-CAM send images to Google Drive – IoT Security Camera*. [online] Available at: <https://www.electroniclinic.com/esp32-cam-send-images-to-google-drive-iot-security-camera/> [Accessed 29 November 2024].

Goh, Y., 2023. *A Study on Smart Parking System Using IoT Technology in Shopping Mall*. Bachelor's Degree. Universiti Tunku Abdul Rahman. Available at: <http://eprints.utar.edu.my/id/eprint/6242> [Accessed 13 August 2024].

Hong, S.Y.C., Kang, C.C., Tan, J.D. and Ariannejad, M., 2023. Smart Parking System Using IoT Sensors. *Journal of Engineering Technology and Applied Physics*, 5(1), pp.1-10. https://doi.org/10.33093/jetap.2023.5.1.7

IBM., 2024. *What is the internet of things?* [online] IBM. Available at: <https://www.ibm.com/topics/internet-of-things.> [Accessed 9 August 2024].

Koya, H., Likhitha, K., Srilatha, K., Saida Babu, G. and Vaishnavi, G., 2024. IoT Based Smart Vehicle Parking System Using RFID. *International Journal for Modern Trends in Science and Technology*, 10(02), pp.53-60. https://doi.org/10.46501/IJMTST1002008

Last Minute Engineers, 2018. *What is RFID? How It Works? Interface RC522 RFID Module with Arduino*. [online] Last Minute Engineers. Available at: <https://lastminuteengineers.com/how-rfid-works-rc522-arduino-tutorial/.> [Accessed 21 August 2024]

Last Minute Engineers, 2019. *ESP32 NTP Server - Get Date and Time with NTP Protocol*. [online] Last Minute Engineers. Available at: <https://lastminuteengineers.com/esp32-ntp-server-date-time-tutorial/> [Accessed 23 December 2024].

Lee, O., 2021. Malaysia's Largest LPR Parking System for Sunway Pyramid. *Parking Network*, [online] Available at: <https://www.parking.net/parking-news/jieshun/malaysias-largest-lpr-parking-system-for-sunway-pyramid> [Accessed 13 August 2024].

Lincy, A., Natarajan, V., Murugan, S., Daniel, S., & Madhan, T. 2024. Intelligent parking management using ANPR technology. *International Journal of Progressive Research in Engineering Management and Science*, 4(6), pp. 1362-1373. https://doi.org/10.58257/IJPREMS34940

Mischianti, R., 2019. *PCF8575 I2C 16-bit Digital Input/Output Expander*. [online] Available at: <https://www.hackster.io/xreef/pcf8575-i2c-16-bit-digital-input-output-expander-48a7c6> [Accessed 28 November 2024].

Mountain, D., 2022. *Using the SD Card in 1-Bit Mode on the ESP32-CAM from AI-Thinker*. [online]. Available at: <https://dr-mntn.net/2021/02/using-the-sd-card-in-1-bit-mode-on-the-esp32-cam-from-ai-thinker> [Accessed 9 October 2024].

Oracle., 2024. *Accelerate Your Operations with IOT*. [online] Available at: <https://www.oracle.com/my/internet-of-things/.> [Accessed 9 August 2024].

Rajiv., 2018. *What are the major components of Internet of Things - RF Page*. [online] Available at: <https://www.rfpage.com/what-are-the-major-components-of-internet-of-things/.> [Accessed 9 August 2024].

Santos, S., 2019. ESP32-CAM Video Streaming and Face Recognition with Arduino IDE. *Random Nerd Tutorials* [online]. Available at: <https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/> [Accessed 21 August 2024].

Saeliw, A., Hualkasin, W., Puttinaovarat, S. and Khaimook, K., 2019. Smart car parking mobile application based on RFID and IoT. *International Journal of Interactive Mobile Technologies (iJIM),* 13(5). https://doi.org/10.3991/ijim.v13i05.10096

Salma, O., Olanrewaju, R.F. and Arman, M.M., 2019. Smart parking guidance system using 360° camera and Haar-Cascade classifier on IoT system. *International Journal of Recent Technology and Engineering*, 8(2S11), pp.864-872. https://doi.org/10.35940/ijrte.B1142.0982S1119

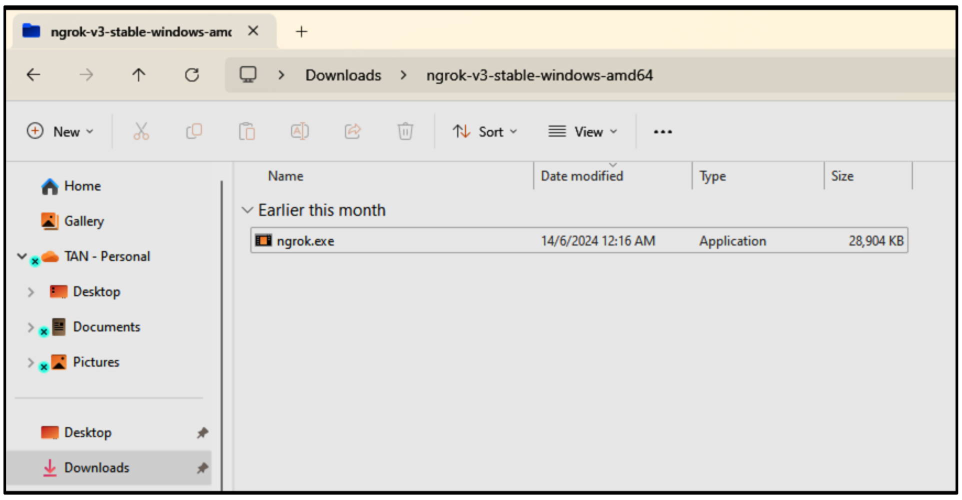
Statista., n.d. *Malaysia: car ownership among consumers 2019*. [online] Available at: <https://www.statista.com/statistics/1029277/malaysia-car-ownership-among-consumers/.> [Accessed 13 August 2024].

Wilson, J., 2020. *ESP32 Pinout, Datasheet, Features & Applications - The Engineering Projects*. [online] Available at: <https://www.theengineeringprojects.com/2020/12/esp32-pinout-datasheet-features-applications.html.> [Accessed 21 August 2024].

APPENDICES

Appendix A: Ngrok Tunneling Steps

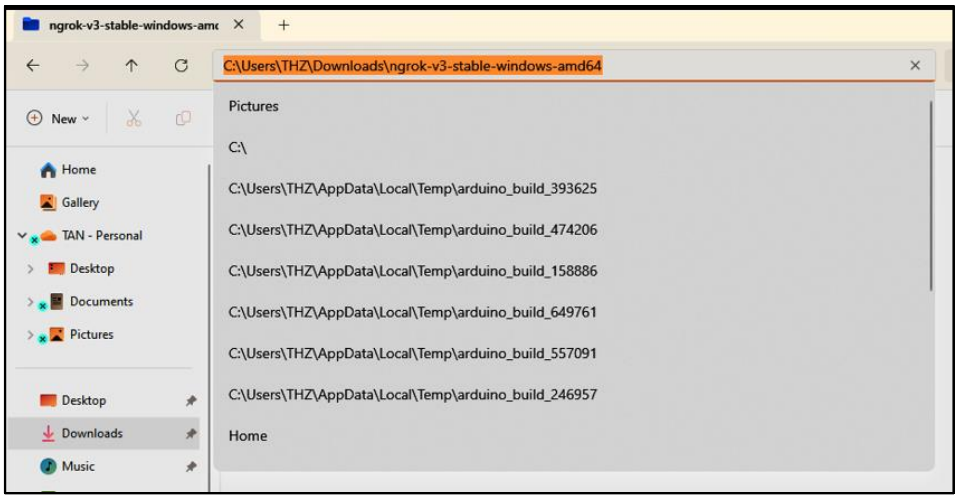
Firstly, ngrok is downloaded from this ***link*** (click). Then, the location of where the ‘ngrok.exe’ file is installed is located, as illustrated in Figure A-1. Then, the directory where the file is installed is added to the system’s path environment variable. The highlighted directory in Figure A-2 is copied. In Windows 11 PC, “Settings > System > Advanced system settings” is clicked. In the popped-up window, under the “Advanced” tab, “Environment Variables” is clicked. In the Environment Variables window, the "Path" variable under "System variables" is selected, and the “Edit” button is clicked. Next, the “New” button is clicked, the directory copied earlier in Figure A-2 is pasted in the blank space, and the “OK” button is pressed on all windows to apply the changes. The detailed steps are shown in Figure A-3. In addition, a new ngrok account is registered, the user would be provided with an authentication token as shown in Figure A-4. The authentication token is copied. Next, in the command prompt, ‘ngrok config add-authtoken’ followed by the authentication token copied earlier is typed, as illustrated in Figure A-5. Now, the ngrok is ready to run. The command ‘ngrok http 80’ is typed and run as shown in Figure A-6, this creates a secure tunnel, making the local web server running on port 80 accessible from the internet via a public URL. The public URL is generated as shown in Figure A-7. In order to access the login page, the public URL, followed by ‘\’, the database name, followed by ‘\’, and the PHP file name serving the HTML page is typed, which is illustrated in Figure A-8. Now, the web application is no longer limited to the local machine, it can be accessed from anywhere in the world using any network connection, whether WiFi or mobile data. The URL link can be converted later to a QR code for easier access.



Earlier this month

Name	Date modified	Type	Size
ngrok.exe	14/6/2024 12:16 AM	Application	28,904 KB

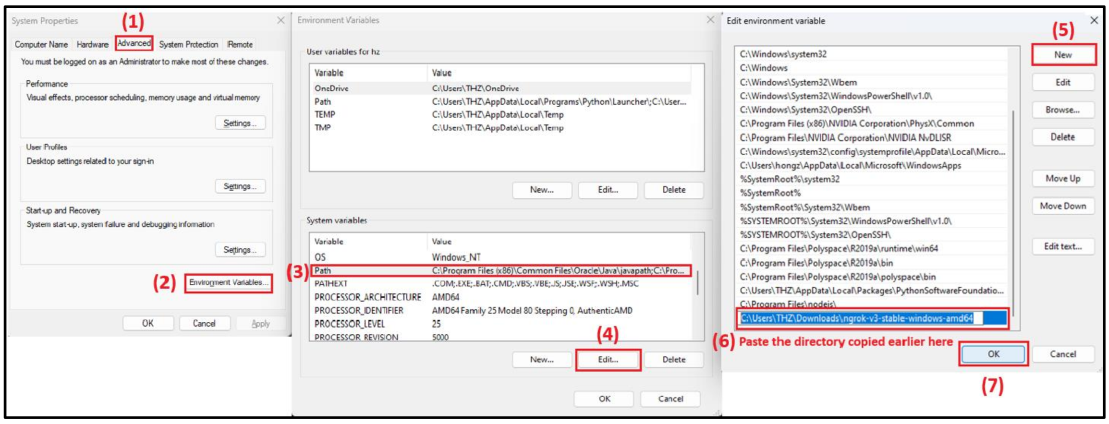
Figure A-1: The Location of ‘ngrok.exe’ File.



C:\Users\THZ\Downloads\ngrok-v3-stable-windows-amd64

```
Pictures
C:\
C:\Users\THZ\AppData\Local\Temp\arduino_build_393625
C:\Users\THZ\AppData\Local\Temp\arduino_build_474206
C:\Users\THZ\AppData\Local\Temp\arduino_build_158886
C:\Users\THZ\AppData\Local\Temp\arduino_build_649761
C:\Users\THZ\AppData\Local\Temp\arduino_build_557091
C:\Users\THZ\AppData\Local\Temp\arduino_build_246957
Home
```

Figure A-2: The File Path where ‘ngrok.exe’ Is Installed was Copied.



You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage and virtual memory

User Profiles
Desktop settings related to your sign-in

Start-up and Recovery
System start-up, system failure and debugging information

User variables for hz

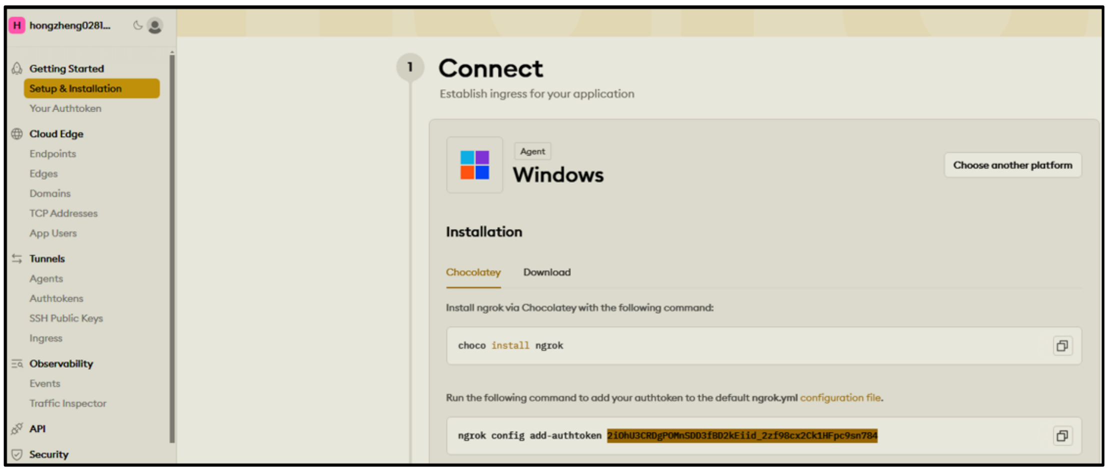
Variable	Value
OneDrive	C:\Users\THZ\OneDrive
Path	C:\Users\THZ\AppData\Local\Programs\Python\Launcher\;C:\User...
TEMP	C:\Users\THZ\AppData\Local\Temp
TMP	C:\Users\THZ\AppData\Local\Temp

System variables

Variable	Value
OS	Windows_NT
Path	C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Pro...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	AMD64 Family 25 Model 80 Stepping 0, AuthenticAMD
PROCESSOR_LEVEL	25
PROCESSOR_REVISION	5000

```
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Windows\System32\OpenSSH\
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR
C:\Windows\system32\config\systemprofile\AppData\Local\Micro...
C:\Users\hongz\AppData\Local\Microsoft\WindowsApps
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Polyspace\R2019a\runtime\win64
C:\Program Files\Polyspace\R2019a\bin
C:\Program Files\Polyspace\R2019a\polyspace\bin
C:\Users\THZ\AppData\Local\Packages\PythonSoftwareFoundatio...
C:\Program Files\nodejs\
C:\Users\THZ\Downloads\ngrok-v3-stable-windows-amd64
```

Figure A-3: The Directory Is Added to the System’s Path Environment Variable.



1 Connect

Establish ingress for your application

Windows

Installation

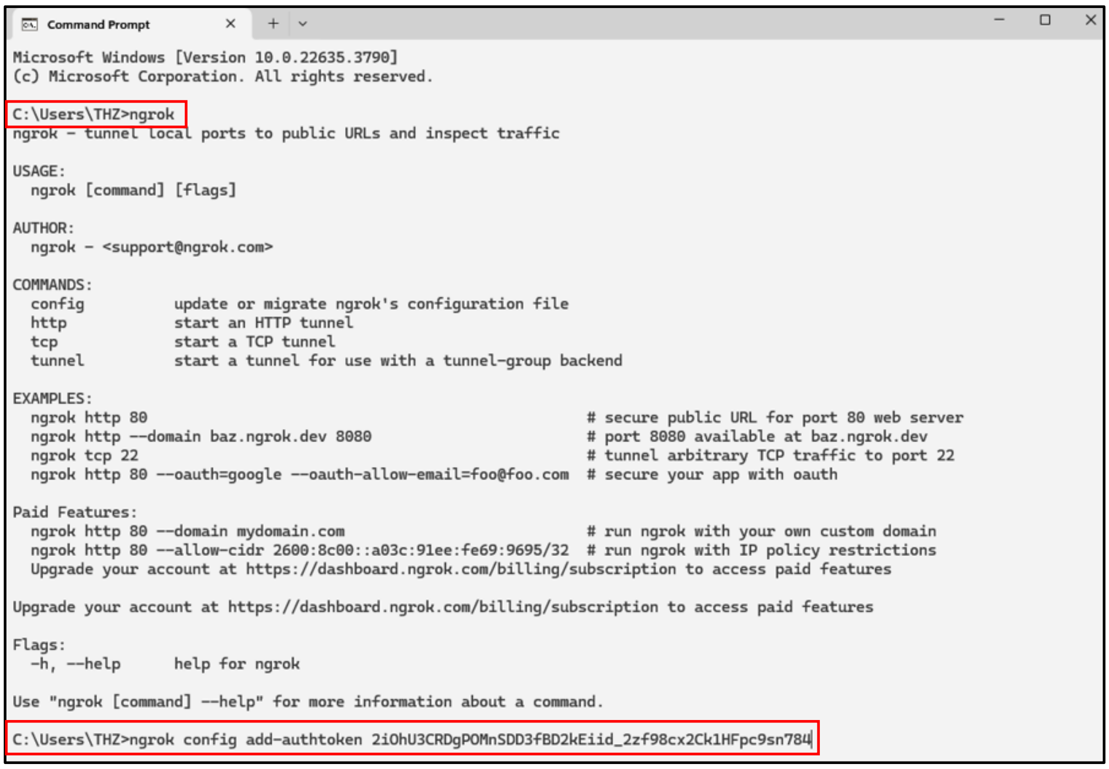
Install ngrok via Chocolatey with the following command:

```
choco install ngrok
```

Run the following command to add your authtoken to the default ngrok.yml configuration file.

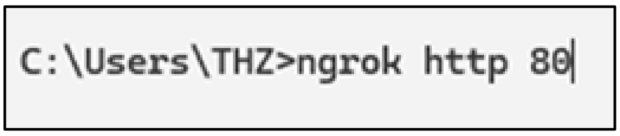
```
ngrok config add-authtoken 2iOhU3CRDgPOMnSDD3fBD2kEiid_2zf98cx2Ck1HFpc9sn784
```

Figure A-4: Ngrok Authentication Token Is Copied.



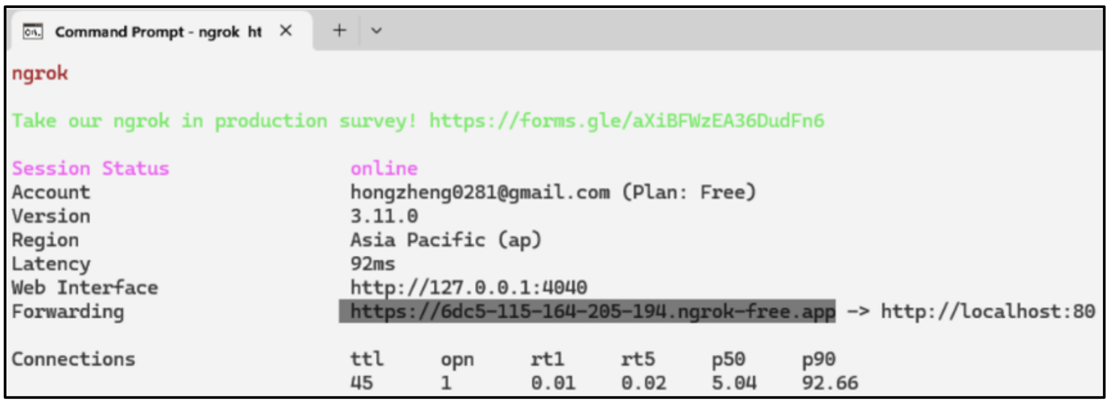
```
Microsoft Windows [Version 10.0.22635.3790]
(c) Microsoft Corporation. All rights reserved.

C:\Users\THZ>ngrok
ngrok - tunnel local ports to public URLs and inspect traffic

USAGE:
  ngrok [command] [flags]

AUTHOR:
  ngrok - <support@ngrok.com>

COMMANDS:
  config          update or migrate ngrok's configuration file
  http            start an HTTP tunnel
  tcp             start a TCP tunnel
  tunnel          start a tunnel for use with a tunnel-group backend

EXAMPLES:
  ngrok http 80                                              # secure public URL for port 80 web server
  ngrok http --domain baz.ngrok.dev 8080                     # port 8080 available at baz.ngrok.dev
  ngrok tcp 22                                               # tunnel arbitrary TCP traffic to port 22
  ngrok http 80 --oauth=google --oauth-allow-email=foo@foo.com  # secure your app with oauth

Paid Features:
  ngrok http 80 --domain mydomain.com                        # run ngrok with your own custom domain
  ngrok http 80 --allow-cidr 2600:8c00::a03c:91ee:fe69:9695/32  # run ngrok with IP policy restrictions
  Upgrade your account at https://dashboard.ngrok.com/billing/subscription to access paid features

Upgrade your account at https://dashboard.ngrok.com/billing/subscription to access paid features

Flags:
  -h, --help      help for ngrok

Use "ngrok [command] --help" for more information about a command.

C:\Users\THZ>ngrok config add-authtoken 2iOhU3CRDgPOMnSDD3fBD2kEiid_2zf98cx2Ck1HFpc9sn784
```

Figure A-5: The Command to Add the Ngrok Authentication Token.



```
C:\Users\THZ>ngrok http 80
```

Figure A-6: The Command to Create a Secure Tunnel to Expose Port 80 to the Internet.



```
Command Prompt - ngrok ht

ngrok

Take our ngrok in production survey! https://forms.gle/aXiBFWzEA36DudFn6

Session Status                online
Account                       hongzheng0281@gmail.com (Plan: Free)
Version                       3.11.0
Region                        Asia Pacific (ap)
Latency                       92ms
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://6dc5-115-164-205-194.ngrok-free.app -> http://localhost:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              45      1       0.01    0.02    5.04    92.66
```

Figure A-7: The Public URL Generated by Ngrok.

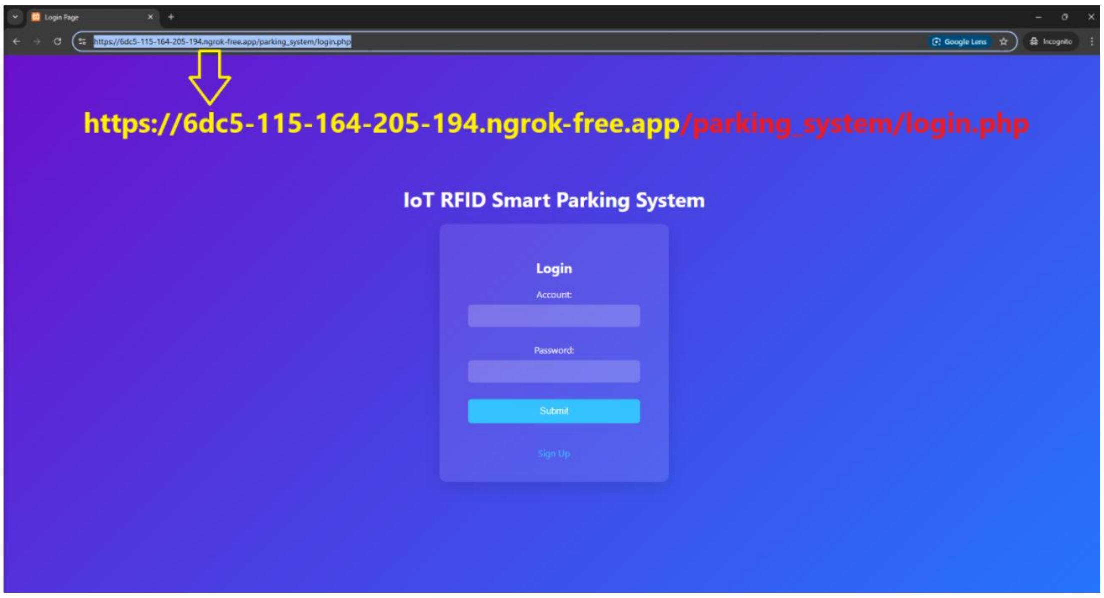

Login Page
https://6dc5-115-164-205-194.ngrok-free.app/parking_system/login.php
https://6dc5-115-164-205-194.ngrok-free.app/parking_system/login.php
IoT RFID Smart Parking System
Login
Account:
Password:
Submit
Sign Up

Figure A-8: The Login Page Accessed Using the Public URL.

Appendix B: Google Apps Script Deployment and ESP32-CAM Integration

A Google Apps Script is created to handle the reception and decoding of images from the ESP32-CAM and store them in Google Drive. Firstly, the Google Drive is opened, and ‘New > More > Google Apps Script’ is selected, as illustrated in Figure B-1. Then, a new project is created, as illustrated in Figure B-2. In the script editor, the default content is replaced with the script provided to process the received image data, as illustrated in Figure B-3. The script extracts three parameters from the ESP32-CAM, which are the Base64-encoded image data, the MIME type (jpeg), and the filename. The script is then renamed and saved, as illustrated in Figure B-4. After that, the script is deployed as a web app by selecting ‘Deploy > New deployment’, as illustrated in Figure B-5. The deployment is configured as a web app, a description is added, and access is set to ‘Anyone’, as illustrated in Figure B-6 and Figure B-7. Once the script is authorized in Figure B-8, Figure B-9, and Figure B-10, a unique web app URL is generated and copied for later use, as illustrated in Figure B-11. Then, a new folder is created in Google Drive with the same name as the script, as illustrated in Figure B-12.

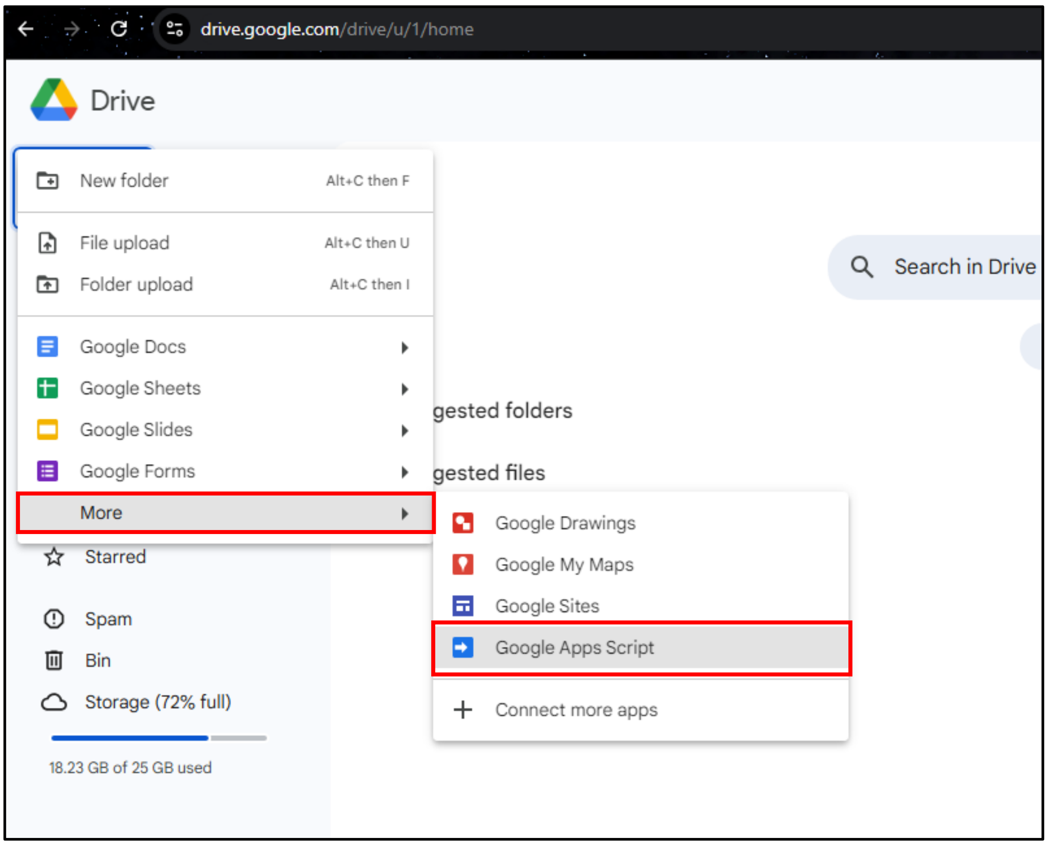


Figure B-1: Accessing Google Apps Script from Google Drive.

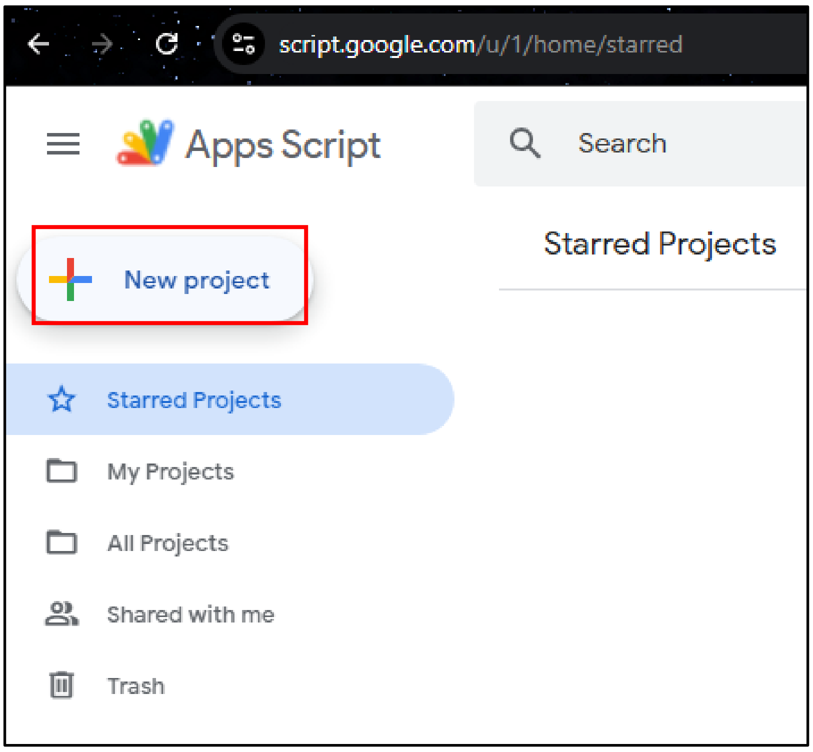
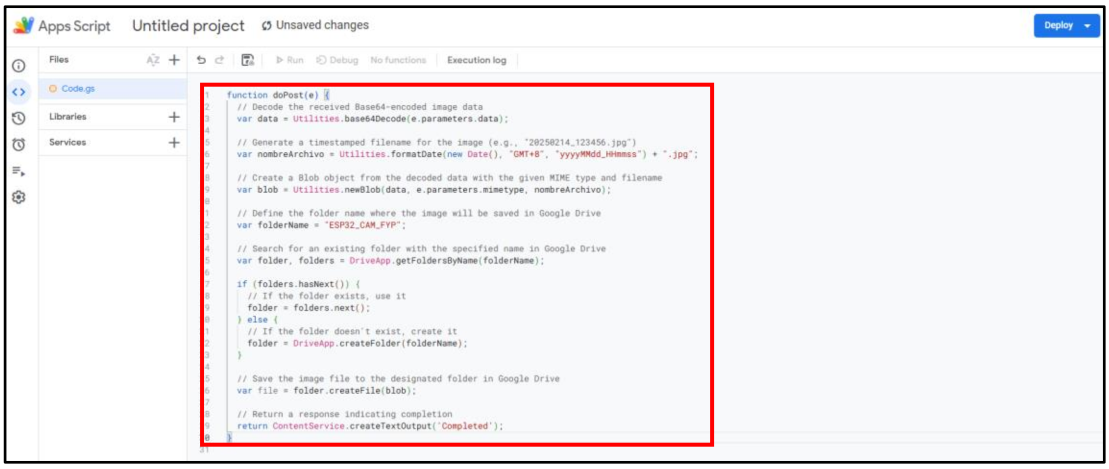
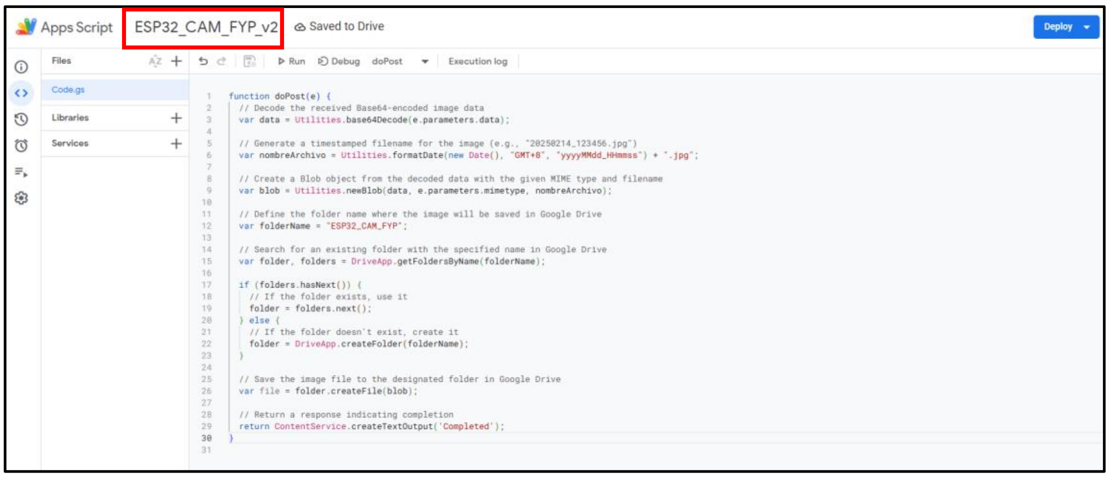


Figure B-2: Creating a New Google Apps Script Project.



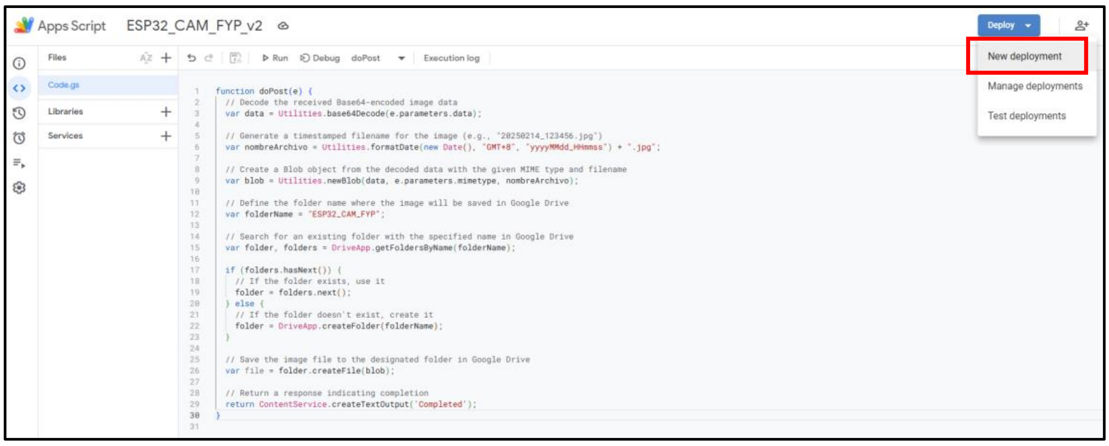
```
function doPost(e) {
  // Decode the received Base64-encoded image data
  var data = Utilities.base64Decode(e.parameters.data);

  // Generate a timestamped filename for the image (e.g., "20250214_123456.jpg")
  var nombreArchivo = Utilities.formatDate(new Date(), "GMT+8", "yyyyMMdd_HHmmss") + ".jpg";

  // Create a Blob object from the decoded data with the given MIME type and filename
  var blob = Utilities.newBlob(data, e.parameters.mimetype, nombreArchivo);

  // Define the folder name where the image will be saved in Google Drive
  var folderName = "ESP32_CAM_FYP";

  // Search for an existing folder with the specified name in Google Drive
  var folder, folders = DriveApp.getFoldersByName(folderName);

  if (folders.hasNext()) {
    // If the folder exists, use it
    folder = folders.next();
  } else {
    // If the folder doesn't exist, create it
    folder = DriveApp.createFolder(folderName);
  }

  // Save the image file to the designated folder in Google Drive
  var file = folder.createFile(blob);

  // Return a response indicating completion
  return ContentService.createTextOutput('Completed');
}
```

Figure B-3: Pasting the Script Into the Google Apps Script Editor.



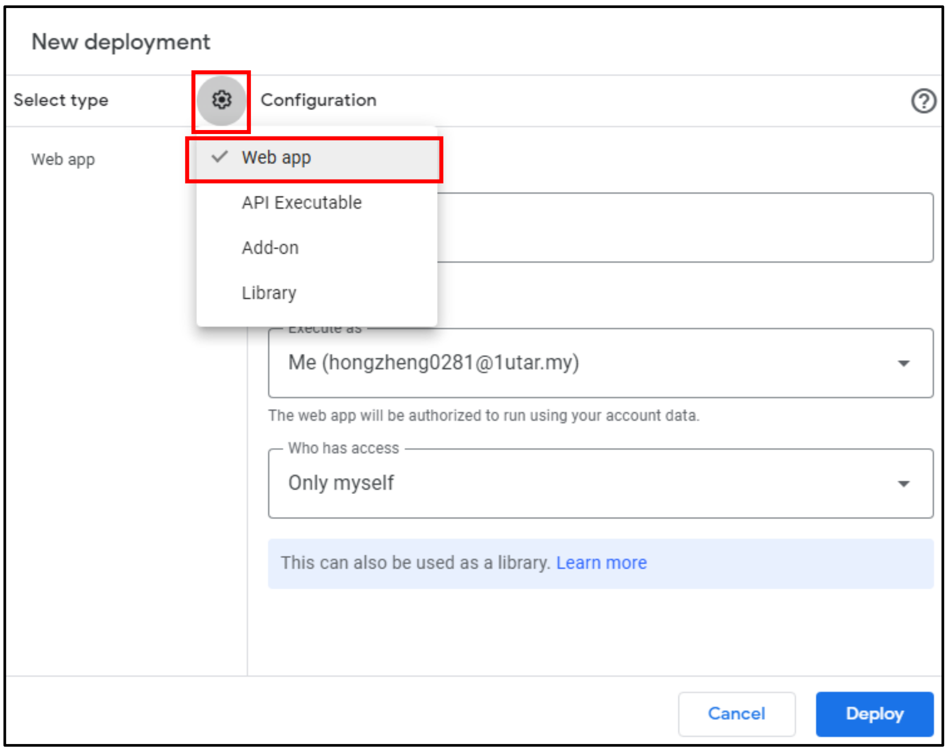
```
function doPost(e) {
  // Decode the received Base64-encoded image data
  var data = Utilities.base64Decode(e.parameters.data);

  // Generate a timestamped filename for the image (e.g., "20250214_123456.jpg")
  var nombreArchivo = Utilities.formatDate(new Date(), "GMT+8", "yyyyMMdd_HHmmss") + ".jpg";

  // Create a Blob object from the decoded data with the given MIME type and filename
  var blob = Utilities.newBlob(data, e.parameters.mimetype, nombreArchivo);

  // Define the folder name where the image will be saved in Google Drive
  var folderName = "ESP32_CAM_FYP";

  // Search for an existing folder with the specified name in Google Drive
  var folder, folders = DriveApp.getFoldersByName(folderName);

  if (folders.hasNext()) {
    // If the folder exists, use it
    folder = folders.next();
  } else {
    // If the folder doesn't exist, create it
    folder = DriveApp.createFolder(folderName);
  }

  // Save the image file to the designated folder in Google Drive
  var file = folder.createFile(blob);

  // Return a response indicating completion
  return ContentService.createTextOutput('Completed');
}
```

Figure B-4: Renaming the Script.



```
function doPost(e) {
  // Decode the received Base64-encoded image data
  var data = Utilities.base64Decode(e.parameters.data);

  // Generate a timestamped filename for the image (e.g., "20250214_123456.jpg")
  var nombreArchivo = Utilities.formatDate(new Date(), "GMT+8", "yyyyMMdd_HHmmss") + ".jpg";

  // Create a Blob object from the decoded data with the given MIME type and filename
  var blob = Utilities.newBlob(data, e.parameters.mimetype, nombreArchivo);

  // Define the folder name where the image will be saved in Google Drive
  var folderName = "ESP32_CAM_FYP";

  // Search for an existing folder with the specified name in Google Drive
  var folder, folders = DriveApp.getFoldersByName(folderName);

  if (folders.hasNext()) {
    // If the folder exists, use it
    folder = folders.next();
  } else {
    // If the folder doesn't exist, create it
    folder = DriveApp.createFolder(folderName);
  }

  // Save the image file to the designated folder in Google Drive
  var file = folder.createFile(blob);

  // Return a response indicating completion
  return ContentService.createTextOutput('Completed');
}
```

Figure B-5: Deploying the Script.



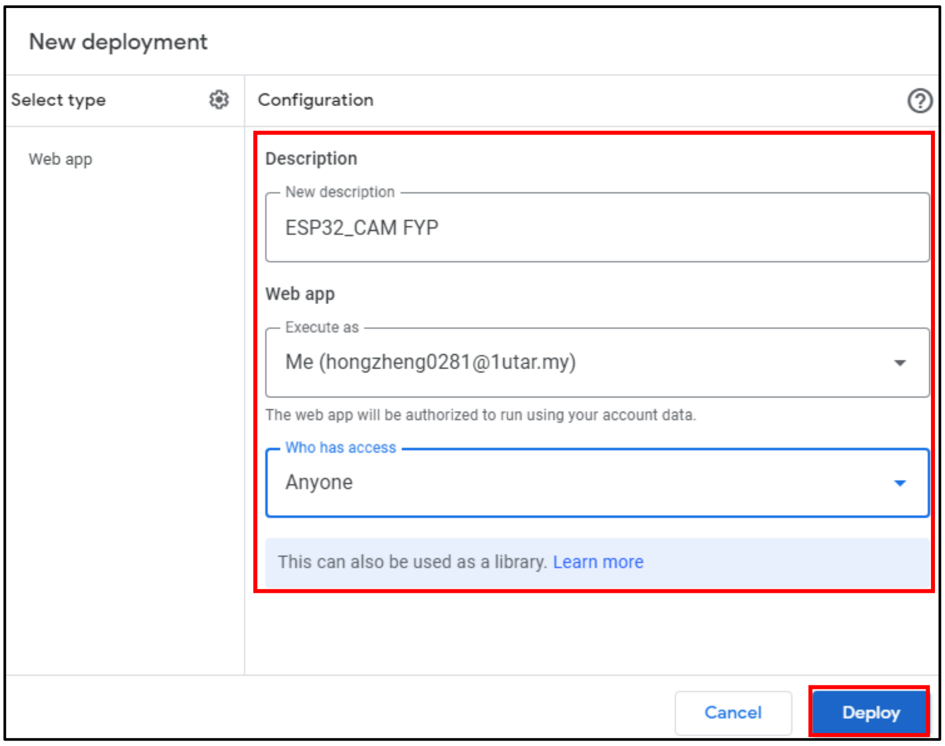
Execute as: Me (hongzheng0281@1utar.my)

The web app will be authorized to run using your account data.

Who has access: Only myself

This can also be used as a library. Learn more

Figure B-6: Selecting "Web App" as the Deployment Type.



Description

New description: ESP32_CAM FYP

Web app

Execute as: Me (hongzheng0281@1utar.my)

The web app will be authorized to run using your account data.

Who has access: Anyone

This can also be used as a library. Learn more

Figure B-7: Configuring Deployment Settings and Setting Access Permissions.

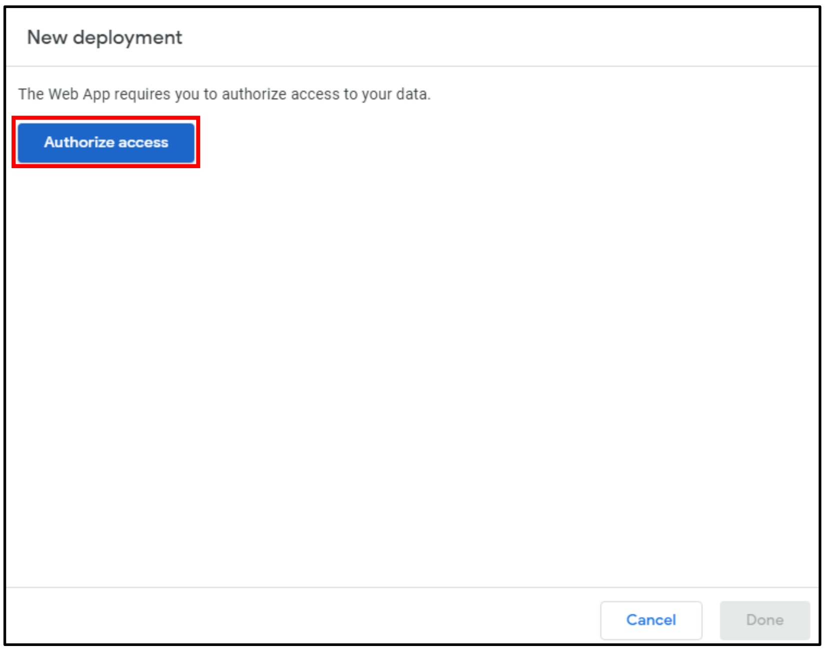


Figure B-8: Authorizing Access for Web App Deployment.

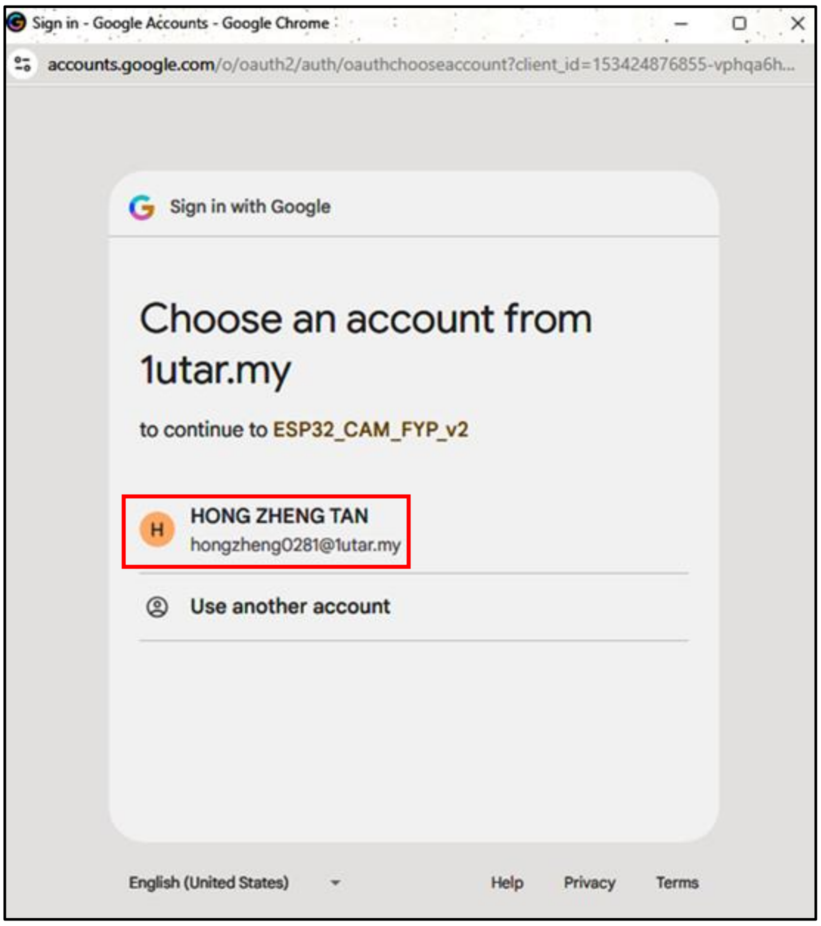
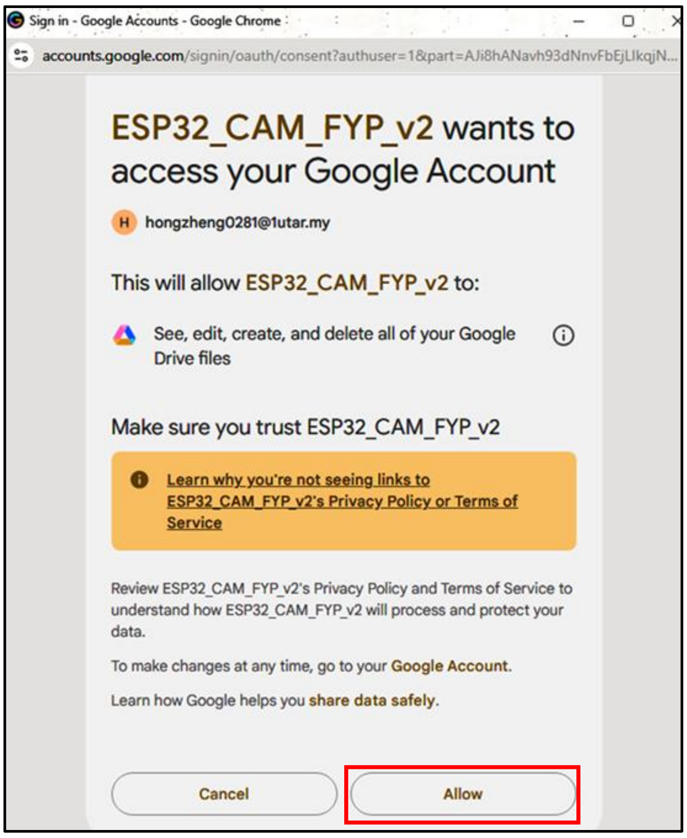

Sign in - Google Accounts - Google Chrome
accounts.google.com/o/oauth2/auth/oauthchooseaccount?client_id=153424876855-vphqa6h...
Sign in with Google
Choose an account from 1utar.my
to continue to ESP32_CAM_FYP_v2
HONG ZHENG TAN
hongzheng0281@1utar.my
Use another account
English (United States)
Help
Privacy
Terms

Figure B-9: Logging Into Google Account.


Sign in - Google Accounts - Google Chrome

accounts.google.com/signin/oauth/consent?authuser=1&part=AJi8hANavh93dNnvFbEjLlkqjN...

ESP32_CAM_FYP_v2 wants to access your Google Account

hongzheng0281@1utar.my

This will allow ESP32_CAM_FYP_v2 to:

See, edit, create, and delete all of your Google Drive files

Make sure you trust ESP32_CAM_FYP_v2

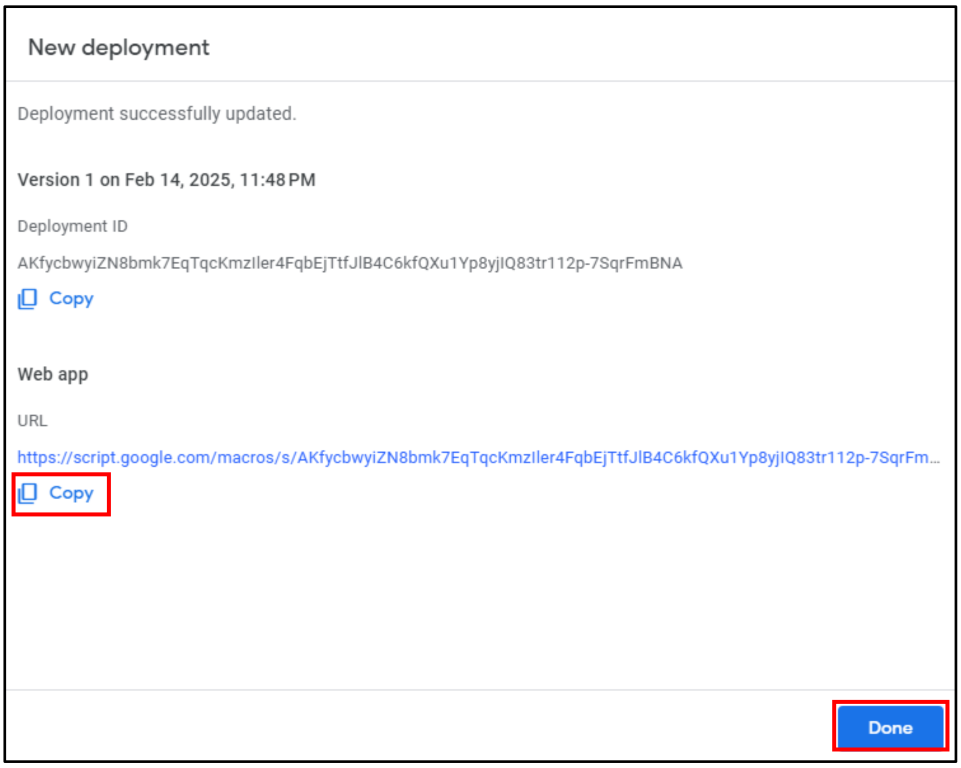
Learn why you're not seeing links to ESP32_CAM_FYP_v2's Privacy Policy or Terms of Service

Review ESP32_CAM_FYP_v2's Privacy Policy and Terms of Service to understand how ESP32_CAM_FYP_v2 will process and protect your data.

To make changes at any time, go to your Google Account.

Learn how Google helps you share data safely.

Figure B-10: Granting Permissions to the Web Application.


New deployment

Deployment successfully updated.

Version 1 on Feb 14, 2025, 11:48 PM

Deployment ID

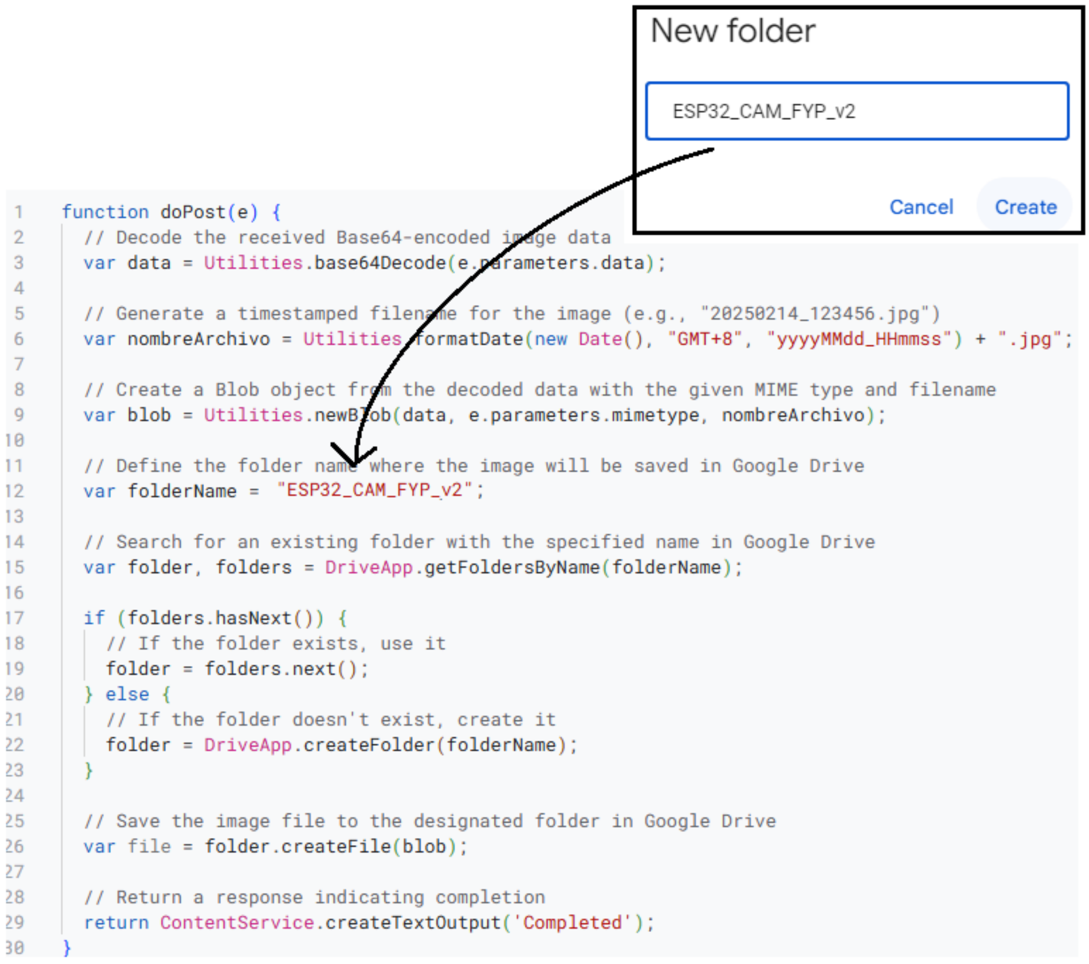
AKfycbwyiZN8bmk7EqTqcKmzIler4FqbEjTtfJlB4C6kfQXu1Yp8yjlQ83tr112p-7SqrFmBNA

Web app

URL

https://script.google.com/macros/s/AKfycbwyiZN8bmk7EqTqcKmzIler4FqbEjTtfJlB4C6kfQXu1Yp8yjlQ83tr112p-7SqrFm...

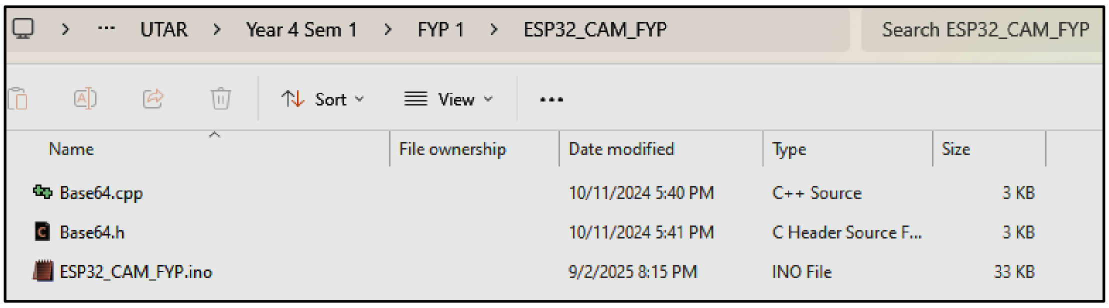
Figure B-11: Copying the Web App Deployment URL.



```
function doPost(e) {
  // Decode the received Base64-encoded image data
  var data = Utilities.base64Decode(e.parameters.data);

  // Generate a timestamped filename for the image (e.g., "20250214_123456.jpg")
  var nombreArchivo = Utilities.formatDate(new Date(), "GMT+8", "yyyyMMdd_HHmmss") + ".jpg";

  // Create a Blob object from the decoded data with the given MIME type and filename
  var blob = Utilities.newBlob(data, e.parameters.mimetype, nombreArchivo);

  // Define the folder name where the image will be saved in Google Drive
  var folderName = "ESP32_CAM_FYP_v2";

  // Search for an existing folder with the specified name in Google Drive
  var folder, folders = DriveApp.getFoldersByName(folderName);

  if (folders.hasNext()) {
    // If the folder exists, use it
    folder = folders.next();
  } else {
    // If the folder doesn't exist, create it
    folder = DriveApp.createFolder(folderName);
  }

  // Save the image file to the designated folder in Google Drive
  var file = folder.createFile(blob);

  // Return a response indicating completion
  return ContentService.createTextOutput('Completed');
}
```

Figure B-12: Manually Creating a New Folder in Google Drive with the Same Name as the Script.

Next, the Base64 encoding library, credited to Adam Rudd, is obtained. The *Base64.h* header file is obtained from this ***link*** (click), while the *Base64.cpp* implementation file is obtained from this ***link*** (click). Both files need to be placed in the same location as the *ESP32_CAM_FYP.ino* file, as illustrated in Figure B-13. Furthermore, some information needs to be changed in the *ESP32_CAM_FYP.ino* file. The WiFi credentials must be replaced, and the Web App URL copied in Figure B-11 must be pasted into the highlighted section of the *ESP32_CAM_FYP.ino* file, as illustrated in Figure B-14.

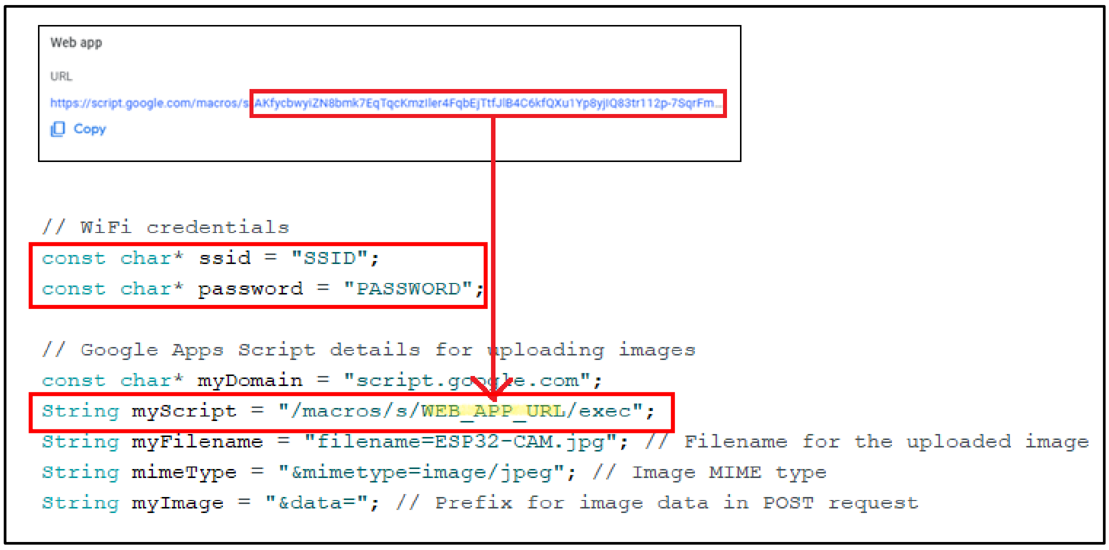

› ··· UTAR › Year 4 Sem 1 › FYP 1 › ESP32_CAM_FYP

Search ESP32_CAM_FYP

Sort View ···

Name	File ownership	Date modified	Type	Size
Base64.cpp		10/11/2024 5:40 PM	C++ Source	3 KB
Base64.h		10/11/2024 5:41 PM	C Header Source F...	3 KB
ESP32_CAM_FYP.ino		9/2/2025 8:15 PM	INO File	33 KB

Figure B-13: The *ESP32_CAM_FYP.ino* File and Base64 Library Are Stored in the Same Directory.


Web app

URL

https://script.google.com/macros/s/AKfycbwyiZN8bmk7EqTqcKmzIler4FqbEjTtfJIB4C6kfQXu1Yp8yjIQ83tr112p-7SqrFm...

Copy

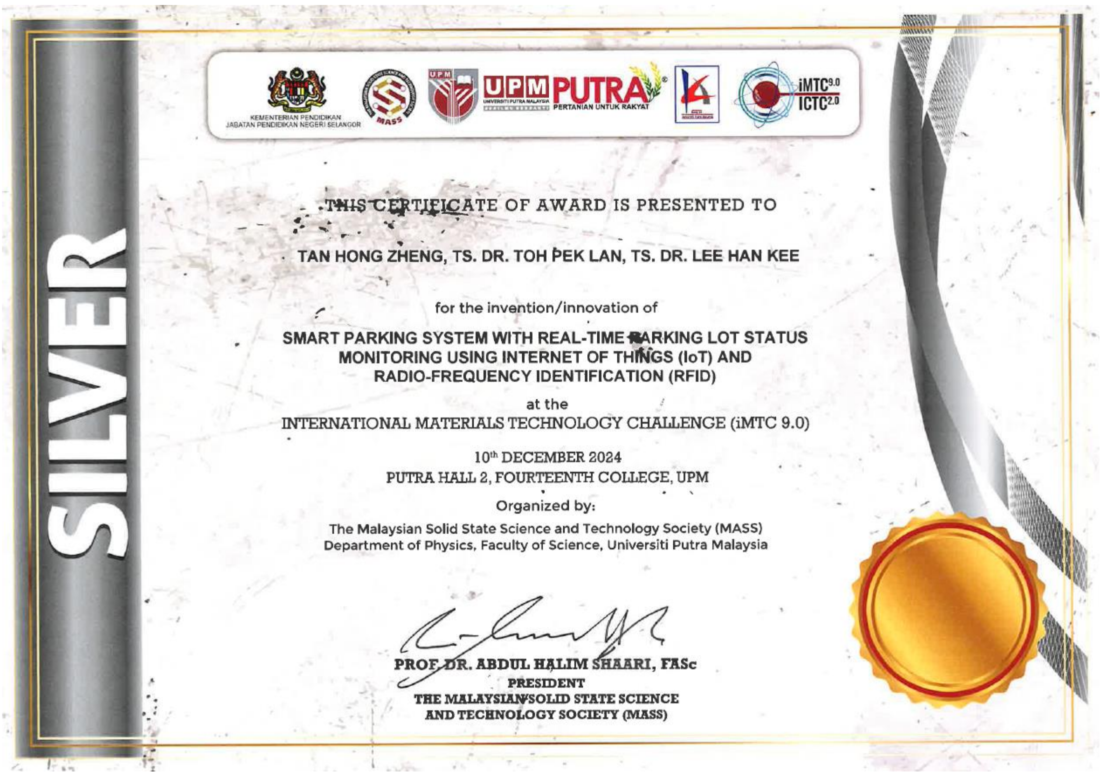
```
// WiFi credentials
const char* ssid = "SSID";
const char* password = "PASSWORD";

// Google Apps Script details for uploading images
const char* myDomain = "script.google.com";
String myScript = "/macros/s/WEB_APP_URL/exec";
String myFilename = "filename=ESP32-CAM.jpg"; // Filename for the uploaded image
String mimeType = "&mimetype=image/jpeg"; // Image MIME type
String myImage = "&data="; // Prefix for image data in POST request
```

Figure B-14: Some Information That Need to Be Changed in the *ESP32_CAM_FYP.ino* File.

Appendix C: Demonstration Video and Award Won

The demonstration video, linked to a YouTube video can be accessed by clicking this ***link*** (click) or by scanning the QR code in Figure C-1 for a clearer understanding of this project. Besides that, this project won a Silver Medal at the International Materials Technology Challenge (iMTC 9.0), organized by the Malaysia Solid State Science and Technology Society (MASS) Chapter UPM and co-organized by the Department of Physics, Selangor Education Department, on December 10, 2024.

Figure C-1: QR Code Linked to a YouTube Video Demonstrating This Project.


SILVER

KEMENTERIAN PENDIDIKAN
JABATAN PENDIDIKAN NEGERI SELANGOR

THIS CERTIFICATE OF AWARD IS PRESENTED TO

TAN HONG ZHENG, TS. DR. TOH PEK LAN, TS. DR. LEE HAN KEE

for the invention/innovation of

SMART PARKING SYSTEM WITH REAL-TIME PARKING LOT STATUS MONITORING USING INTERNET OF THINGS (IoT) AND RADIO-FREQUENCY IDENTIFICATION (RFID)

at the

INTERNATIONAL MATERIALS TECHNOLOGY CHALLENGE (iMTC 9.0)

10th DECEMBER 2024
PUTRA HALL 2, FOURTEENTH COLLEGE, UPM

Organized by:

The Malaysian Solid State Science and Technology Society (MASS)
Department of Physics, Faculty of Science, Universiti Putra Malaysia

[illegible]

PROF. DR. ABDUL HALIM SHAARI, FASc
PRESIDENT
THE MALAYSIAN SOLID STATE SCIENCE
AND TECHNOLOGY SOCIETY (MASS)

Figure C-2: Silver Medal Certificate for Participation in iMTC 9.0.

Appendix D: Code for ESP32, ESP32-CAM, and Google Apps Script

```
// Define WiFi credentials
char ssid[] = "WIFI_SSID";
char pass[] = "WIFI_PASSWORD";
const String server_ip = "http://10.0.30.20/parking_system";
//-----------------------------------------------------------------------------
// Include necessary libraries
#include <SPI.h>               // For SPI communication (used by RFID)
#include <MFRC522.h>           // For RFID module functionality
#include <Wire.h>              // For I2C communication
#include <Adafruit_GFX.h>      // For OLED display graphics
#include <Adafruit_SSD1306.h>  // For OLED display
#include <LiquidCrystal_I2C.h> // For LCD display
#include <WiFi.h>              // For WiFi connectivity
#include <WiFiClient.h>        // For WiFi client functionality
#include <ESP32Servo.h>        // For servo motor control
#include <HTTPClient.h>        // For making HTTP requests
#include <ArduinoJson.h>       // For JSON parsing and serialization
#include "time.h"              // For time functions
//-----------------------------------------------------------------------------
// Function declarations
void checkRFIDEntrance();
void getExitStatusFromSecondESP32();
void checkRFIDExit();
void checkVehicleAndControlGate();
String checkRFIDInDatabase(String uid, String &ownerName, String &carColor, String
&licensePlate);
bool checkBalance(String uid, float &balance);
void updateBalance(String uid, float newBalance);
void logRFIDScan(String uid, String status, String ownerName = "UNKNOWN", String carColor =
"UNKNOWN", String licensePlate = "UNKNOWN");
void fetchParkingStatusFromDatabase();
String getTimestamp();
void updateClock();
void grantAccessEntrance(String ownerName, String carColor, String licensePlate, float
balance);
void grantAccessExit(String ownerName, String carColor, String licensePlate);
void denyAccess();
void openEntranceGate();
void closeEntranceGate();
void openExitGate();
void closeExitGate();
//-----------------------------------------------------------------------------
// Pin definitions
#define SS_PIN_ENTRANCE 5      // GPIO 5 for entrance RFID Slave Select Pin
#define RST_PIN_ENTRANCE 27    // GPIO 27 for entrance RFID Reset Pin
#define buzzer_PIN 4           // GPIO 4 connects to buzzer
#define green_LED 13           // GPIO 13 connects to green LED
#define red_LED 12             // GPIO 12 connects to red LED
#define entrance_IR1 15        // Entrance IR sensor 1
#define entrance_IR2 36        // Entrance IR sensor 2
#define exit_IR1 34            // Exit IR sensor 1
#define exit_IR2 39            // Exit IR sensor 2
int entrance_gate_SERVO = 14;  // GPIO 14 connects to entrance servo motor
int exit_gate_SERVO = 32;      // GPIO 32 connects to exit servo motor

// OLED Display configuration
#define SCREEN_WIDTH 128       // OLED display width in pixels
#define SCREEN_HEIGHT 64       // OLED display height in pixels

// Initialize hardware components
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, 0x3C); // 0x3C is the I2C
address of OLED
LiquidCrystal_I2C lcd(0x27, 20, 4); // 0x27 is the I2C address of 20x4 LCD
MFRC522 rfidEntrance(SS_PIN_ENTRANCE, RST_PIN_ENTRANCE); // RFID reader for entrance
Servo entranceGateServo; // Create a servo object for the entrance gate
Servo exitGateServo; // Create a servo object for the exit gate

// Authorization Status & Message flags
bool entranceRFIDAuthorized = false; // Flag to store RFID authorization status for entrance
bool exitRFIDAuthorized = false; // Flag to store RFID authorization status for exit
bool Authorized = false; // Flag to store authorization status from secondary ESP32
bool introDisplayed = false; // Flag to track if the intro message has been displayed
bool messageDisplayed = false; // Global flag to track if a message has been displayed

// Parking slot variables
int parking_IR1;      // Parking slot 1 status
int parking_IR2;      // Parking slot 2 status
int parking_IR3;      // Parking slot 3 status
int parking_IR4;      // Parking slot 4 status
int parking_IR5;      // Parking slot 5 status
int parking_IR6;      // Parking slot 6 status
int parking_IR7;      // Parking slot 7 status
int parking_IR8;      // Parking slot 8 status

// Vehicle detection flags
int entrance_flag1 = 0; // Flag for IR1 (entrance) sensor
int entrance_flag2 = 0; // Flag for IR2 (entrance) sensor
int exit_flag1 = 0; // Flag for IR1 (exit) sensor
int exit_flag2 = 0; // Flag for IR2 (exit) sensor
int Slot = 8; // Total parking slots (8)

// User and vehicle information
String entrancelastUID = "";          // Last scanned UID at entrance
String exitLastUID = "";              // Last scanned UID at exit
String ownerName = "";                // Vehicle owner name
String carColor = "";                 // Vehicle color
String licensePlate = "";             // Vehicle license plate
float balance = 0.0;                  // User Account Balance

unsigned long messageDisplayStart = 0;
const long messageDisplayDuration = 3000; // 3 seconds display time on OLED
bool showingMessage = false;

// NTP Server settings (Internet Time)
const char* ntpServer = "pool.ntp.org";  // NTP server to get time (without relying on RTC
module)
const long gmtOffset_sec = 8 * 3600;     // GMT+8: 8 hours (8 * 3600 seconds) is the time
offset for Malaysia
const int daylightOffset_sec = 0;        // No daylight savings in Malaysia
//---------------------------------------------------------------------------
// Function to initialize NTP server and sync time
void setupNTP() {
  // Configure time with the specified NTP server, GMT offset, and daylight offset
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

  // Attempt to get the time from the NTP server once
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    // If time retrieval fails, print an error message and continue
    Serial.println("Failed to obtain time from NTP server.");
  } else {
    // If time retrieval is successful, print the fetched time
    Serial.println("Time successfully synchronized with NTP server");
    Serial.printf("Time: %02d/%02d/%04d %02d:%02d:%02d\n",
        timeinfo.tm_mday, timeinfo.tm_mon + 1, timeinfo.tm_year + 1900,
        timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec);
  }
}
//---------------------------------------------------------------------------
// Function to get the current timestamp as a formatted string (RFID scanning timestamp on
OLED)
String getTimestamp() {
  struct tm timeinfo;
  getLocalTime(&timeinfo); // Fetch the current local time
  char buffer[20]; // Create a buffer to store the formatted timestamp

  // Format the time as "DD/MM/YYYY HH:MM:SS" and store it in the buffer
  // timeinfo.tm_mon is 0-based (January = 0), so we add 1 to get the correct month number.
  // timeinfo.tm_year stores the number of years since 1900, so we add 1900 to get the actual
year.
  sprintf(buffer, "%02d/%02d/%04d %02d:%02d:%02d",
          timeinfo.tm_mday, timeinfo.tm_mon + 1, timeinfo.tm_year + 1900,
          timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec);

  // Return the formatted timestamp as a String
  return String(buffer);
}
//---------------------------------------------------------------------------
// Grant access for entrance (display balance)
void grantAccessEntrance(String ownerName, String carColor, String licensePlate, float
balance)
{
 String timestamp = getTimestamp(); // Get current timestamp
 Serial.println(timestamp);
 Serial.println("User: " + ownerName);
 Serial.println("Access authorized");
 Serial.println();
 // Display access granted messages on OLED
 display.clearDisplay();
 display.setTextSize(1); // Set font size
 display.setTextColor(SSD1306_WHITE); // Set font colour
 display.setCursor(0, 0); // Set the cursor to the top-left corner (row 0, column 0)
 display.println(timestamp); // Print current timestamp
 display.setCursor(0, 12); // Adjust vertical position
 display.println("ACCESS GRANTED");
 display.setCursor(0, 24); // Adjust vertical position
 display.println("User: " + ownerName); // Print owner name
 display.setCursor(0, 36); // Adjust vertical position
 display.println("Vehicle No: " + licensePlate); // Print license plate
 display.setCursor(0, 48); // Adjust vertical position
 display.println("Balance: RM " + String(balance)); // Print account balance
 display.display();

 // Set message display timing
    messageDisplayStart = millis();
    showingMessage = true;

 digitalWrite(green_LED, HIGH); // Turns on Green LED
 // Buzzer sounds twice quickly
 for (int i = 0; i < 2; i++) {
   digitalWrite(buzzer_PIN, HIGH);
   delay(100); // Short beep
   digitalWrite(buzzer_PIN, LOW);
   delay(100); // Short beep
 }
 digitalWrite(green_LED, LOW); // Turns off Green LED
}
//---------------------------------------------------------------------------
// Grant access for exit (without account balance)
void grantAccessExit(String ownerName, String carColor, String licensePlate)
{
 String timestamp = getTimestamp();
 Serial.println(timestamp);
 Serial.println("User: " + ownerName);
 Serial.println("Access authorized");
 Serial.println();
  // Display access granted messages on OLED (without account balance)
 display.clearDisplay();
 display.setTextSize(1); // Set font size
 display.setTextColor(SSD1306_WHITE);
 display.setCursor(0, 0);
 display.println(timestamp);
 display.setCursor(0, 12); // Adjust vertical position
 display.println("ACCESS GRANTED");
 display.setCursor(0, 24); // Adjust vertical position
 display.println("User: " + ownerName);
 display.setCursor(0, 36); // Adjust vertical position
 display.println("Vehicle No: " + licensePlate);
 display.setCursor(0, 48); // Adjust vertical position
 display.println("TQ & See you again!");
 display.display();

 // Set message display timing
    messageDisplayStart = millis();
    showingMessage = true;

 digitalWrite(green_LED, HIGH); // Turns on Green LED
  // Buzzer sounds twice quickly
 for (int i = 0; i < 2; i++) {
   digitalWrite(buzzer_PIN, HIGH);
   delay(100);
   digitalWrite(buzzer_PIN, LOW);
   delay(100);
 }
 digitalWrite(green_LED, LOW); // Turns off Green LED
}
//---------------------------------------------------------------------------
// Deny access to unauthorized users
void denyAccess()
{
 String timestamp = getTimestamp();
 Serial.println(timestamp);
 Serial.println("User: Unregistered User!");
 Serial.println("Access denied");
 Serial.println();

 display.clearDisplay();
 display.setTextSize(1); // Set font size
 display.setTextColor(SSD1306_WHITE);
 display.setCursor(0, 0);
 display.println(timestamp);
 display.setCursor(0, 15); // Adjust vertical position
 display.println("ACCESS DENIED");
 display.setCursor(0, 30); // Adjust vertical position
 display.println("User:");
 display.setCursor(0, 50); // Adjust vertical position
 display.println("Unregistered user!");
 display.display();

 // Set message display timing
```

```
    messageDisplayStart = millis();
    showingMessage = true;

 digitalWrite(red_LED, HIGH); // Turns on Red LED
 // Buzzer sounds for a longer period
 digitalWrite(buzzer_PIN, HIGH);
 delay(1000); // Long beep
 digitalWrite(buzzer_PIN, LOW);
 digitalWrite(red_LED, LOW); // Turns off Red LED
}
//---------------------------------------------------------------------------
 // Display insufficient balance messages on OLED (if account balance < RM 10)
void insufficientBalance(float balance) {
 String timestamp = getTimestamp();
 Serial.println(timestamp);
 Serial.println("Access Denied: Insufficient Balance");
 Serial.println();

 display.clearDisplay();
 display.setTextSize(1); // Set font size
 display.setTextColor(SSD1306_WHITE);
 display.setCursor(0, 0);
 display.println(timestamp);
 display.setCursor(0, 12); // Adjust vertical position
 display.println("ACCESS DENIED");
 display.setCursor(0, 24); // Adjust vertical position
 display.println("Insufficient Balance");
 display.setCursor(0, 36); // Adjust vertical position
 display.println("Balance: RM " + String(balance));
 display.display();

 // Set message display timing
    messageDisplayStart = millis();
    showingMessage = true;

 digitalWrite(red_LED, HIGH); // Turns on Red LED
 // Buzzer sounds for a longer period
 digitalWrite(buzzer_PIN, HIGH);
 delay(1000); // Long beep
 digitalWrite(buzzer_PIN, LOW);
 digitalWrite(red_LED, LOW); // Turns off Red LED
}
//---------------------------------------------------------------------------
void openEntranceGate() {
 entranceGateServo.write(0); // Rotate to 0 degrees (open)
 Serial.println("Entrance gate opened");
}

void closeEntranceGate() {
 entranceGateServo.write(90); // Rotate to 90 degrees (closed)
 Serial.println("Entrance gate closed");
}

void openExitGate() {
 exitGateServo.write(0); // Rotate to 0 degrees (open)
 Serial.println("Exit gate opened");
}

void closeExitGate() {
 exitGateServo.write(90); // Rotate to 90 degrees (closed)
 Serial.println("Exit gate closed");
}
//---------------------------------------------------------------------------
// Check RFID at entrance
void checkRFIDEntrance() {
  // Check every 400ms (offset from exit reader)
  // Avoid entrance and exit reader on at the same time to avoid interference
  static unsigned long lastCheck = 100;  // Starts at 100ms offset
  if (millis() - lastCheck < 400) return;
  lastCheck = millis();

     // Check if new RFID card is present and readable
    if (rfidEntrance.PICC_IsNewCardPresent() && rfidEntrance.PICC_ReadCardSerial()) {
        messageDisplayed = false;// New scan detected, reset the message flag
        // Show UID on serial monitor
        Serial.print("Entrance UID tag: ");
        String content = "";
        // Read and format UID bytes
        for (byte i = 0; i < rfidEntrance.uid.size; i++) {
            Serial.print(rfidEntrance.uid.uidByte[i] < 0x10 ? " 0" : " ");
            Serial.print(rfidEntrance.uid.uidByte[i], HEX);
            content += String(rfidEntrance.uid.uidByte[i] < 0x10 ? "0" : "");
            content += String(rfidEntrance.uid.uidByte[i], HEX);
        }
        Serial.println();
        content.toUpperCase(); // Converts all letters to uppercase
        entrancelastUID = content;  // Store UID

        // Variables to store user and vehicle details
        String ownerName = "";
        String carColor = "";
        String licensePlate = "";

        // Check if RFID UID read exists in the database or not, if yes, set status as
"success"
        String status = checkRFIDInDatabase(entrancelastUID, ownerName, carColor,
licensePlate);
        Serial.println("Entrance RFID Database check status: " + status);
        Serial.println("Owner name: " + ownerName);
        Serial.println("Car Color: " + carColor);
        Serial.println("License Plate: " + licensePlate);

        // Registered RFID Tag (Matches with database UID)
        if (status == "success") {
            entranceRFIDAuthorized = true; // set entrance RFID authorization flag to true

            // Check balance and handle insufficient balance first
            if (checkBalance(entrancelastUID, balance)) {
                // Insufficient balance: deny access
                if (balance < 10) {  // RM 10 is the required balance threshold (parking fee)
                    if (!messageDisplayed) {
                        insufficientBalance(balance); //Display insufficient balance message
on OLED
                        Serial.println("Insufficient balance. Please top up your card and try
again.");
                        messageDisplayed = true;  // Set flag to avoid repeated messages
                    }
                    // Do not display access granted
                    entranceRFIDAuthorized = false; // Reset authorization
                    rfidEntrance.PICC_HaltA();      // Halt PICC
                    rfidEntrance.PCD_StopCrypto1(); // Stop encryption on PCD
                    return;  // Exit the function early to prevent further processing
                } else {
                    // Sufficient balance: grant access
                    if (!messageDisplayed) {
                        grantAccessEntrance(ownerName, carColor, licensePlate, balance);
                        logRFIDScan(entrancelastUID, "SUCCESS", ownerName, carColor,
licensePlate);
                        messageDisplayed = true;  // Set flag to avoid repeated messages
                    }
                }
            } else {
                // Balance check failed (e.g., server error)
                Serial.println("Failed to retrieve balance.");
            }
        }
        // Unregistered RFID Tag (Not matches with database UID)
        else {
            entranceRFIDAuthorized = false;  // set entrance RFID authorization flag to false
            denyAccess(); // Display access denied on OLED
            logRFIDScan(entrancelastUID, "FAIL"); //logs RFID Scan as "FAIL" in database and
records the UID
        }
        // Halt RFID scanning to avoid multiple reads
        rfidEntrance.PICC_HaltA();
        rfidEntrance.PCD_StopCrypto1();
    }
}
//---------------------------------------------------------------------------
// Get exit RFID scan status from secondary ESP32 by querying the 'exit_status.json' file
void getExitStatusFromSecondESP32() {
    HTTPClient http;  // Create HTTP client object
    // Execute 'get_exit_status.php' to get exit status from 'exit_status.json'
    String url = server_ip + "/get_exit_status.php";
    http.begin(url);  // Initialize HTTP connection

    int httpCode = http.GET();  // Send HTTP GET request
    if (httpCode == HTTP_CODE_OK) {  // If request is successful
        String payload = http.getString();  // Get response payload
        Serial.println("Exit Status Response: " + payload);  // Print response for debugging
        DynamicJsonDocument doc(1024);  // Create JSON document
        deserializeJson(doc, payload);  // Parse JSON response

        const char* status = doc["status"];  // Get status from JSON
        if (status && String(status) == "success") {  // If status is success
            Authorized = doc["Authorized"].as<bool>();  // Get authorization status
            exitLastUID = doc["uid"].as<String>();  // Get RFID UID
            ownerName = doc["owner_name"].as<String>();  // Get owner name
            carColor = doc["car_color"].as<String>();  // Get car color
            licensePlate = doc["license_plate"].as<String>();  // Get license plate
            Serial.println("Status: " + String(status));  // Print status
        } else {  // If status is not success
```

```
            Serial.println("Status not success, setting exitLastUID to empty");
            exitLastUID = "";  // Clear UID
        }
    } else {  // If HTTP request failed
        Serial.println("HTTP GET request failed");  // Error message
    }
    http.end();  // Close HTTP connection
}
//-----------------------------------------------------------------------------
 // Variable to store the last processed UID to prevent same card read twice in a row
String lastProcessedUID = ""; // act as a temporary memory

// Check RFID at exit gate (depending on results from second ESP32)
void checkRFIDExit() {
    getExitStatusFromSecondESP32(); // Always get the exit rfid authorization status from second ESP32

    if (exitLastUID != "" && exitLastUID != lastProcessedUID) {
        // Only process if exitLastUID is not empty and different from the last processed UID
        Serial.println("Exit UID tag: " + exitLastUID);
        Serial.println("Exit RFID Database check status: " + String(Authorized));
        Serial.println("Owner name: " + ownerName);
        Serial.println("Car Color: " + carColor);
        Serial.println("License Plate: " + licensePlate);

        if (Authorized) {
            // Grant access since the user is authorized
            exitRFIDAuthorized = true; // Set exit RFID authorization flag to true
            grantAccessExit(ownerName, carColor, licensePlate); // Print access authorized message on OLED
            logRFIDScan(exitLastUID, "SUCCESS", ownerName, carColor, licensePlate); // Logs RFID scan
        } else {
            // Deny access if the RFID is not authorized
            exitRFIDAuthorized = false;
            denyAccess();
            logRFIDScan(exitLastUID, "FAIL");
        }

        // Update the last processed UID to the current one
        lastProcessedUID = exitLastUID;

        // Reset exitLastUID after processing
        exitLastUID = "";
    } else if (exitLastUID == lastProcessedUID) {
        // If the UID hasn't changed, do not process it again
        Serial.println("No new RFID tag read or already processed");
    }
}
//-----------------------------------------------------------------------------
// Call this function after user exits and gate closes to
// reset the exit_status.json JSON content after each scan
void resetExitStatus()
{
    HTTPClient http;
    String url = server_ip + String("/update_exit_status.php"); // Build reset URL
    http.begin(url); // Initialize HTTP connection
    http.addHeader("Content-Type", "application/json");  // Set content type

    // Reset the exit_status.json JSON content after each scan
    DynamicJsonDocument doc(1024);
    doc["uid"] = ""; // Clear the UID
    doc["owner_name"] = ""; // Clear owner name
    doc["car_color"] = ""; // Clear car color
    doc["license_plate"] = ""; // Clear license plate
    doc["Authorized"] = false; // Reset authorization status

    String requestBody; // Create request body
    serializeJson(doc, requestBody); // Serialize JSON to string

    int httpCode = http.POST(requestBody);  // Send HTTP POST request
    if (httpCode == HTTP_CODE_OK) {
        Serial.println("Exit status reset.");
    } else {
        Serial.println("Failed to reset exit status.");
    }
    http.end();
}
//-----------------------------------------------------------------------------
// Control gates based on vehicle detection and authorization status
void checkVehicleAndControlGate() {
    // Read current states of IR sensors (LOW means vehicle detected)
    bool entrance_ir1State = digitalRead(entrance_IR1) == LOW;   // Entrance IR sensor 1 state
    bool entrance_ir2State = digitalRead(entrance_IR2) == LOW;   // Entrance IR sensor 2 state
    bool exit_ir1State = digitalRead(exit_IR1) == LOW;            // Exit IR sensor 1 state
    bool exit_ir2State = digitalRead(exit_IR2) == LOW;            // Exit IR sensor 2 state
    int required_amount = 10;                                     // Parking fee

    // Check if parking lot is full when vehicle presents at the entrance and scan RFID tags
    if (Slot <= 0 && entrance_ir1State && entranceRFIDAuthorized) {
      // Display "Sorry, parking full" message on LCD
      lcd.clear();
      lcd.setCursor(8, 1);
      lcd.print("SORRY,");
      lcd.setCursor(3, 2);
      lcd.print("PARKING FULL...");
      delay(500);
      }

    // Vehicle detected at entrance and entrance gate closed
    //entrance_flag1 = system memory, entrance_ir1State = current state
    if (entrance_ir1State && entrance_flag1 == 0 && Slot > 0 && entrance_flag2 == 0 && entranceRFIDAuthorized) {
        Serial.println("Checking balance..."); // Balance checking already done at checkRFIDEntrance()

        if (balance >= required_amount) {
            // Sufficient balance: grant access
            entrance_flag1 = 1; // Set IR1 memory to 1 as vehicle has passed through it
            openEntranceGate();   // Open the gate when vehicle detected at entrance, RFID check passed, and sufficient balance
            updateBalance(entrancelastUID, balance - required_amount); // Deduct balance and update in database
            float latestBalance = balance - required_amount; // Calculate new balance
            Serial.print("Latest balance: ");
            Serial.println(latestBalance);

            // Display access authorized and account balance on OLED
            if (!messageDisplayed) {
                grantAccessEntrance(ownerName, carColor, licensePlate, latestBalance);
                messageDisplayed = true;  // Set flag to avoid repeated messages
            }
        } else {
            // If insufficient balance, no gate opening, show insufficientBalance() message on OLED
            if (!messageDisplayed) {
                insufficientBalance(balance);
                Serial.println("Insufficient balance. Please top up your card and try again.");
                messageDisplayed = true;  // Set flag to avoid repeated messages
            }
            entranceRFIDAuthorized = false;  // Reset authorization
            return;  // Exit the function to prevent further processing
        }
    }

    // Vehicle fully passed through entrance
    // Vehicle has passed through IR1, no longer at IR1, at IR2 and has not passed through IR2
    if (entrance_flag1 == 1 && !entrance_ir1State && entrance_ir2State && entrance_flag2 == 0) {
        entrance_flag2 = 1; // Set IR2 memory to 1 as vehicle has passed through it
        delay(1000);
        closeEntranceGate();  // Close the gate after vehicle passes through IR2
        Slot--;       // Decrease available slot count by one
        entrance_flag1 = 0; // Reset entrance IR1 memory
        entrance_flag2 = 0; // Reset entrance IR2 memory
        entranceRFIDAuthorized = false; // Reset RFID authorization after vehicle passes
        messageDisplayed = false;       // Reset flag for the next scan
    }

    // Vehicle detected at exit and exit gate closed
    if (exit_ir1State && exit_flag1 == 0 && exit_flag2 == 0 && exitRFIDAuthorized) {
        exit_flag1 = 1;
        openExitGate();   // Open the gate when vehicle detected at exit
        Serial.println("Thank you for visiting. See you again :)");
    }

    // Vehicle fully passed through exit
    if (exit_flag1 == 1 && !exit_ir1State && exit_ir2State && exit_flag2 == 0) {
        exit_flag2 = 1;
        delay(1000);
        closeExitGate();  // Close the gate after vehicle passes through IR2
        Slot++;       // Increase available slot count by one
        exit_flag1 = 0; // Reset exit IR1 memory
        exit_flag2 = 0; // Reset exit IR2 memory
        exitRFIDAuthorized = false; // Reset RFID authorization after vehicle passes
        resetExitStatus();  // Call the function to clear exit_status.json JSON data
    }
}
```

```
//--------------------------------------------------------------------------
// Check if scanned UID exists in 'rfid_user_info' table in database and get associated information
String checkRFIDInDatabase(String uid, String &ownerName, String &carColor, String &licensePlate) {
   HTTPClient http;
   String url = server_ip + String("/check_rfid.php?rfid=") + uid; // Build URL with UID to execute check_rfid.php
   Serial.println("Requesting URL: " + url);
   http.begin(url); // Initialize connection

   int httpCode = http.GET(); // Send HTTP GET request
   if (httpCode == HTTP_CODE_OK) { // If request successful
       String payload = http.getString();  // Get response
       Serial.println("Server response: " + payload); // Debug response
       DynamicJsonDocument doc(1024); // Create JSON document
       deserializeJson(doc, payload); // Parse JSON

       if (doc["status"] == "success") { // If status is success
           ownerName = doc["owner_name"].as<String>(); // Get owner name
           carColor = doc["car_color"].as<String>();  // Get car color
           licensePlate = doc["license_plate"].as<String>();  // Get license plate
           http.end(); // Close connection
           return "success";
       } else { // If status not success
           http.end(); // Close connection
           return "fail";
       }
   } else { // If request failed
       http.end(); // Close connection
       return "error";
   }
}
//--------------------------------------------------------------------------
// Check user's account balance from 'rfid_user_info' table in database
bool checkBalance(String uid, float &balance) {
   HTTPClient http;
   String url = server_ip + String("/check_balance.php?rfid=") + uid; // Build URL
   Serial.println("Requesting URL: " + url);
   http.begin(url); // Initialize connection

   int httpCode = http.GET(); // Send HTTP GET request
   Serial.print("HTTP Code: ");
   Serial.println(httpCode); // Print the HTTP code for debugging
   if (httpCode == HTTP_CODE_OK) { // If request successful
       String payload = http.getString(); // Get response
       Serial.print("Server response: ");
       Serial.println(payload);

       DynamicJsonDocument doc(1024); // Create JSON document
       deserializeJson(doc, payload); // Parse JSON

       if (doc["status"] == "success") { // If status is success
           balance = doc["balance"]; // Get balance value
           http.end();  // Close connection
           return true; //this means balance check is success, not neccessary indicates enough balance
       } else { // If status not success
           Serial.println("Failed to retrieve balance, status not success.");
           Serial.println(doc["message"].as<String>()); // Print the failure message
       }
   } else { // If request failed
       Serial.print("Failed to connect, HTTP code: ");
       Serial.println(httpCode);
   }
   http.end(); // Close connection
   return false;
}
//--------------------------------------------------------------------------
// Update user's account balance in 'rfid_user_info' table after vehicle passed through entrance gate
void updateBalance(String uid, float newBalance) {
 HTTPClient http;
 String url = server_ip + String("/update_balance.php");
 Serial.println("Requesting URL: " + url);
 http.begin(url);
 http.addHeader("Content-Type", "application/x-www-form-urlencoded");
 // Build POST data consists of UID and latest account balance
 String postData = "rfid=" + uid + "&new_balance=" + String(newBalance);
 int httpCode = http.POST(postData); // Send HTTP POST request

 if (httpCode == HTTP_CODE_OK) {
   String payload = http.getString();
   Serial.print("Server response: ");
   Serial.println(payload); // Print the payload for debugging
 }
 http.end();
}
//----------------------------------------------------------------------------
// Log RFID scan to 'rfid_scan_log' table in database
void logRFIDScan(String uid, String status, String ownerName, String carColor, String
licensePlate) {
   HTTPClient http;
   String url = server_ip + String("/log_rfid_scan.php");
   Serial.println("Requesting URL: " + url);
   http.begin(url);
   http.addHeader("Content-Type", "application/x-www-form-urlencoded");
   // Build POST data with scan status and all details
   String postData = "rfid=" + uid + "&scan_status=" + status + "&owner_name=" + ownerName +
"&car_color=" + carColor + "&license_plate=" + licensePlate;
   Serial.println("Post Data: " + postData);

   int httpCode = http.POST(postData); // Send HTTP POST request
   if (httpCode == HTTP_CODE_OK) {
       String payload = http.getString();
       Serial.print("Server response: ");
       Serial.println(payload);
   }
   http.end();
}
//----------------------------------------------------------------------------
// Fetch latest parking status from 'parking_status' table in database
void fetchParkingStatusFromDatabase() {
   HTTPClient http; // Create HTTP client
   String url = server_ip + "/get_parking_status.php"; // Build URL
   http.begin(url); // Initialize connection

   int httpCode = http.GET(); // Send HTTP GET request
   if (httpCode == HTTP_CODE_OK) { // If request successful
       String payload = http.getString(); // Get response
       Serial.println("Server response: " + payload);  // Debug response

       DynamicJsonDocument doc(1024); // Create JSON document
       deserializeJson(doc, payload); // Parse JSON

       // Update the parking_IR variables based on the fetched status
       // (1=AVAILABLE, 2=RESERVED, 0=OCCUPIED)
       parking_IR1 = (doc["1"] == "AVAILABLE") ? 1 : (doc["1"] == "RESERVED") ? 2 : 0;
       parking_IR2 = (doc["2"] == "AVAILABLE") ? 1 : (doc["2"] == "RESERVED") ? 2 : 0;
       parking_IR3 = (doc["3"] == "AVAILABLE") ? 1 : (doc["3"] == "RESERVED") ? 2 : 0;
       parking_IR4 = (doc["4"] == "AVAILABLE") ? 1 : (doc["4"] == "RESERVED") ? 2 : 0;
       parking_IR5 = (doc["5"] == "AVAILABLE") ? 1 : (doc["5"] == "RESERVED") ? 2 : 0;
       parking_IR6 = (doc["6"] == "AVAILABLE") ? 1 : (doc["6"] == "RESERVED") ? 2 : 0;
       parking_IR7 = (doc["7"] == "AVAILABLE") ? 1 : (doc["7"] == "RESERVED") ? 2 : 0;
       parking_IR8 = (doc["8"] == "AVAILABLE") ? 1 : (doc["8"] == "RESERVED") ? 2 : 0;
   } else { // If request failed
       Serial.print("Failed to fetch parking lot status, HTTP code: ");
       Serial.println(httpCode);
   }
   http.end(); // Close connection
}
//----------------------------------------------------------------------------
void updateDisplay() {
  static unsigned long lastDisplayUpdate = 0;  // Tracks last update time
  static String lastTimeStr = "";              // Stores last displayed time string
  static int lastSlot = -1;                    // Stores last displayed slot count
  static int lastParkingIR[4] = {-1, -1, -1, -1}; // Stores last parking slot states

  unsigned long currentMillis = millis();  // Get current time in milliseconds

  // Update display every second (1000ms)
  if(currentMillis - lastDisplayUpdate >= 1000) {
    lastDisplayUpdate = currentMillis;  // Update the last update time

    // 1. TIME DISPLAY (Row 0)
    struct tm timeinfo;    // Get current time structure

    // If time is successfully retrieved
    if(getLocalTime(&timeinfo)) {
      lcd.setCursor(0, 0);      // Set cursor to beginning of first row (0,0)

      //Display the current date and time in the format "DD/MM/YYYY HH:MM:SS" on the LCD
      // %02d = 2-digit with leading zeros
      // %04d - Four digits for year (YYYY) (e.g., 2025).
      lcd.printf("%02d/%02d/%04d %02d:%02d:%02d",
                timeinfo.tm_mday,        // Day of month (1-31)
                timeinfo.tm_mon + 1,     // Month (0-11, so +1)
                timeinfo.tm_year + 1900, // Years since 1900
                timeinfo.tm_hour,        // Hours (0-23)
                timeinfo.tm_min,         // Minutes (0-59)
                timeinfo.tm_sec);        // Seconds (0-59)
    }

    // 2. SLOT AVAILABILITY (Row 1)
    lcd.setCursor(1, 1);    // Set cursor to column 1, row 1 (second row)
    lcd.print("[SLOT AVAILABLE:");
    lcd.print(Slot);    // Print current available slot count
    lcd.print("]");

    // 3. PARKING SLOT STATUS (Rows 2-3)
    // ----------------------------------
    // Only update if slot 1 or 2 status changed
    if(parking_IR1 != lastParkingIR[0] || parking_IR2 != lastParkingIR[1]) {
      lcd.setCursor(0, 2);      // Set cursor to start of third row (0,2)
      lcd.print("S1:");
      lcd.print(parking_IR1 == 1 ? "FREE      " :  // Available
               (parking_IR1 == 2 ? "RESV      " :  // Reserved
                                  "FILL      "));  // Occupied
      lcd.print("S2:");
      lcd.print(parking_IR2 == 1 ? "FREE      " :
               (parking_IR2 == 2 ? "RESV      " :
                                  "FILL      "));

      // Update last known states
      lastParkingIR[0] = parking_IR1;
      lastParkingIR[1] = parking_IR2;
    }

    // Only update if slot 3 or 4 status changed
    if(parking_IR3 != lastParkingIR[2] || parking_IR4 != lastParkingIR[3]) {
      lcd.setCursor(0, 3);      // Set cursor to start of fourth row (0,3)
      lcd.print("S3:");
      lcd.print(parking_IR3 == 1 ? "FREE      " :
               (parking_IR3 == 2 ? "RESV      " :
                                  "FILL      "));
      lcd.print("S4:");
      lcd.print(parking_IR4 == 1 ? "FREE      " :
               (parking_IR4 == 2 ? "RESV      " :
                                  "FILL      "));

      // Update last known states
      lastParkingIR[2] = parking_IR3;
      lastParkingIR[3] = parking_IR4;
    }
  }
}
//-----------------------------------------------------------------------------------------------
-------
// Setup function - runs once at startup
void setup() {
 Serial.begin(115200);  // Initialize serial communication at 115200 baud

  // Connect to WiFi network
 WiFi.begin(ssid, pass); // Start connection
 while (WiFi.status() != WL_CONNECTED) { // Wait for connection
   delay(1000); // Wait 1 second
   Serial.println("Connecting to WiFi...");
 }
 Serial.println("Connected to WiFi"); // WiFi Connection Success message
 Serial.print("IP Address: ");
 Serial.println(WiFi.localIP()); // Print local IP address

 delay(100);
// Initialize RFID reader with proper timing
 SPI.begin();
 pinMode(SS_PIN_ENTRANCE, OUTPUT);
 digitalWrite(SS_PIN_ENTRANCE, HIGH); // keep SS high when not active
 rfidEntrance.PCD_Init();
 rfidEntrance.PCD_SetAntennaGain(rfidEntrance.RxGain_23dB); //Reduce receiver gain by half
 rfidEntrance.PCD_AntennaOn();
 Serial.print("Approximate your card to the reader...");
 Serial.println();
 Serial.print("Setup completed");

 setupNTP(); // Initialize network time synchronization

 // Configure all pins as INPUT or OUTPUT
 pinMode(buzzer_PIN, OUTPUT);
 pinMode(green_LED, OUTPUT);
 pinMode(red_LED, OUTPUT);
 pinMode(entrance_IR1, INPUT);
 pinMode(entrance_IR2, INPUT);
 pinMode(exit_IR1, INPUT);
 pinMode(exit_IR2, INPUT);
 pinMode(entrance_gate_SERVO, OUTPUT);
 pinMode(exit_gate_SERVO, OUTPUT);

// Configure and test entrance servo
```

```
 entranceGateServo.setPeriodHertz(50); // PWM frequency for SG90
 entranceGateServo.attach(entrance_gate_SERVO, 500, 2400); // Minimum and maximum pulse width (in µs) to go from 0° to 180
 entranceGateServo.write(0);
 delay(100);
 entranceGateServo.write(90);
 delay(100);
 // Configure and test exit servo
 exitGateServo.setPeriodHertz(50); // PWM frequency for SG90
 exitGateServo.attach(exit_gate_SERVO, 500, 2400); // Minimum and maximum pulse width (in µs) to go from 0° to 180
 exitGateServo.write(0);
 delay(100);
 exitGateServo.write(90);
 delay(100);

 // Initialize OLED display
 if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Try to initialize
   Serial.println(F("SSD1306 allocation failed")); // Error message
   for(;;); // Infinite loop if failed
 }

 // Display welcome message on OLED
 display.display(); // Display initialization
 display.clearDisplay(); // Clear display
 display.setTextSize(2); // Set font size to 2
 display.setTextColor(SSD1306_WHITE); // White text
 display.setCursor(40, 0); // Position cursor
 display.println("RFID");
 display.setCursor(25, 30); // Position cursor
 display.println("Scanner");
 display.display(); // Update display

 // Initialize LCD display
 lcd.init(); // Initialize LCD
 lcd.backlight(); // Turn on backlight
 lcd.setCursor(3, 0); // Position cursor
 lcd.print("IoT RFID Smart");
 lcd.setCursor(0, 1);
 lcd.print("Parking System with");
 lcd.setCursor(0, 2);
 lcd.print("Real-time Lot Status");
 lcd.setCursor(5, 3);
 lcd.print("Monitoring");
 delay(1000);
 lcd.clear();
 lcd.setCursor(9, 1);
 lcd.print("By:");
 lcd.setCursor(3, 2);
 lcd.print("TAN HONG ZHENG");
 delay(1000);
 introDisplayed = true;  // Set intro flag
 lcd.clear(); // Clear display
}
//------------------------------------------------------------------------------
// Main loop - runs continuously
void loop() {

  updateDisplay(); // Handles all display updates
  fetchParkingStatusFromDatabase(); // Get latest parking lot status
    // Only show default Welcome Page if not currently showing the RFID Scanning Messages on OLED
    if (!showingMessage || (millis() - messageDisplayStart > messageDisplayDuration)) {
        if (showingMessage) {
            showingMessage = false; // Message duration elapsed (3 seconds)
        }

        display.clearDisplay();
        display.setTextSize(1);
        display.setTextColor(SSD1306_WHITE);
        display.setCursor(0, 0);
        display.println("Hello! :)");
        display.setCursor(0, 25);
        display.println("Please place your");
        display.setCursor(0, 50);
        display.println("card at the scanner.");
        display.display();
    }
  checkRFIDEntrance(); // Check entrance RFID
  checkRFIDExit(); // Check exit RFID (get scan results from second ESP32)
  checkVehicleAndControlGate(); // Control gates
}
```

Figure D-1: Code for Main ESP32

```
// Include necessary libraries
#include <WiFi.h>
#include <HTTPClient.h>
#include <SPI.h>
#include <MFRC522.h>
#include <ArduinoJson.h>
#include <PCF8575.h>
#include <Arduino.h>
//---------------------------------------------------------------------------
// Define WiFi credentials and server IP address
const char* ssid = "WIFI_SSID";
const char* password = "WIFI_PASSWORD";
const String server_ip = "http://10.0.30.20/parking_system";
//---------------------------------------------------------------------------
// Define pins for RFID and parking slot IR sensors
#define SS_PIN_EXIT 5       // RFID Slave Select pin for exit reader
#define RST_PIN_EXIT 27     // RFID Reset pin for exit reader
#define parking_IR1 35      // IR sensor for parking slot 1
#define parking_IR2 32      // IR sensor for parking slot 2
#define parking_IR3 33      // IR sensor for parking slot 3
#define parking_IR4 25      // IR sensor for parking slot 4
#define parking_IR5 26      // IR sensor for parking slot 5
#define parking_IR6 14      // IR sensor for parking slot 6
#define parking_IR7 12      // IR sensor for parking slot 7
#define parking_IR8 13      // IR sensor for parking slot 8

// Initialize RFID module for exit gate
MFRC522 rfidExit(SS_PIN_EXIT, RST_PIN_EXIT);  // RFID reader instance
String lastUID = "";                          // Store the last scanned UID

// Initialize parking slots default status as "AVAILABLE"
String parkingStatus[8] = {"AVAILABLE", "AVAILABLE", "AVAILABLE", "AVAILABLE",
                          "AVAILABLE", "AVAILABLE", "AVAILABLE", "AVAILABLE"};

// PCF8575 GPIO Expander setup for additional GPIOs
PCF8575 pcf8575_1(0x20);  // I2C address of first PCF8575 (handles slots 1-4)
PCF8575 pcf8575_2(0x24);  // I2C address of second PCF8575 (handles slots 5-8)
//---------------------------------------------------------------------------
void setup() {
  Serial.begin(115200); // Start serial communication with baud rate 115200
  WiFi.begin(ssid, password); // Connect to WiFi

  // Wait for WiFi connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");

  delay(300);
 // Initialize RFID reader with proper timing
  SPI.begin();
  pinMode(SS_PIN_EXIT, OUTPUT);
  digitalWrite(SS_PIN_EXIT, HIGH); // keep SS high when not active
  rfidExit.PCD_Init();
  rfidExit.PCD_SetAntennaGain(rfidExit.RxGain_23dB); // reduce receiver gain by half
  rfidExit.PCD_AntennaOn();
  delay(1000);   // Add small delay to ensure main ESP32's RFID initializes first

  // Set parking slot IR sensors pins as INPUT
  pinMode(parking_IR1, INPUT);
  pinMode(parking_IR2, INPUT);
  pinMode(parking_IR3, INPUT);
  pinMode(parking_IR4, INPUT);
  pinMode(parking_IR5, INPUT);
  pinMode(parking_IR6, INPUT);
  pinMode(parking_IR7, INPUT);
  pinMode(parking_IR8, INPUT);

  // Initialize PCF8575 GPIO expanders
  // Set all pins on both PCF8575 expanders (from 0 to 15) as OUTPUT
  for (int i = 0; i < 16; i++) {
    pcf8575_1.pinMode(i, OUTPUT);  // Set all pins on pcf8575_1 to output
    pcf8575_2.pinMode(i, OUTPUT);  // Set all pins on pcf8575_2 to output
  }
  pcf8575_1.begin();
  pcf8575_2.begin();
}
//---------------------------------------------------------------------------
void loop() {
  checkRFIDExit(); // Check for RFID tags at exit
  fetchParkingStatusFromDatabase();  // Get latest parking status from server

  // Check each parking sensor
  checkParkingSensor(parking_IR1, 1);
  checkParkingSensor(parking_IR2, 2);
  checkParkingSensor(parking_IR3, 3);
  checkParkingSensor(parking_IR4, 4);
  checkParkingSensor(parking_IR5, 5);
  checkParkingSensor(parking_IR6, 6);
  checkParkingSensor(parking_IR7, 7);
  checkParkingSensor(parking_IR8, 8);
  delay(1000); // Delay to reduce CPU usage
}
//----------------------------------------------------------------------------
// Function to check RFID card at exit gate
void checkRFIDExit() {
 // Check every 400ms but offset by 200ms from entrance
  static unsigned long lastCheck = 300;  // Starts at 300ms offset
  if (millis() - lastCheck < 400) return;
  lastCheck = millis();

    // Check if new RFID card is present and readable
  if (rfidExit.PICC_IsNewCardPresent() && rfidExit.PICC_ReadCardSerial()) {
    Serial.print("Exit UID tag: ");
    String content = "";
    // Read and format UID bytes
    for (byte i = 0; i < rfidExit.uid.size; i++) {
      Serial.print(rfidExit.uid.uidByte[i] < 0x10 ? " 0" : " ");
      Serial.print(rfidExit.uid.uidByte[i], HEX);
      content += String(rfidExit.uid.uidByte[i] < 0x10 ? "0" : "");
      content += String(rfidExit.uid.uidByte[i], HEX);
    }
    Serial.println();
    content.toUpperCase(); // Converts all letters to uppercase
    lastUID = content;  // Store UID

    // Variables to store user and vehicle details
    String ownerName = "";
    String carColor = "";
    String licensePlate = "";

    // Check if RFID UID read exists in the database or not, if yes, set status as "success"
    String status = checkRFIDInDatabase(lastUID, ownerName, carColor, licensePlate);
    Serial.println("Exit RFID Database check status: " + status);
    Serial.println("Owner name: " + ownerName);
    Serial.println("Car Color: " + carColor);
    Serial.println("License Plate: " + licensePlate);
    bool authorized = (status == "success");  // Determine if the RFID tag is authorized or
not

    // Send RFID data to main ESP32
    sendRFIDDataToMainESP32(lastUID, ownerName, carColor, licensePlate, authorized);
    rfidExit.PICC_HaltA(); // Halt PICC
    rfidExit.PCD_StopCrypto1(); // Stop encryption on PCD
  }
}
//----------------------------------------------------------------------------
// Function to check RFID in database
String checkRFIDInDatabase(String uid, String &ownerName, String &carColor, String
&licensePlate) {
  HTTPClient http;
   // Build URL consists of the UID that need to be checked
  String url = server_ip + String("/check_rfid.php?rfid=") + uid;
  http.begin(url); // Start HTTP connection

  int httpCode = http.GET(); // Send HTTP GET request
  if (httpCode == HTTP_CODE_OK) {
    String payload = http.getString();  // Get response
    Serial.println("Server response: " + payload);

    // Parse JSON response
    DynamicJsonDocument doc(1024);
    deserializeJson(doc, payload);

    if (doc["status"] == "success") {
      // Extract user and vehicle details from JSON response
      ownerName = doc["owner_name"].as<String>();
      carColor = doc["car_color"].as<String>();
      licensePlate = doc["license_plate"].as<String>();
      http.end();
      return "success";
    }
  }
  http.end();  // Close connection
  return "fail"; // Return fail if error
}
//----------------------------------------------------------------------------
// Function to send RFID data to 'exit_status.json' where main ESP32 retrieves from it
void sendRFIDDataToMainESP32(String uid, String ownerName, String carColor, String
licensePlate, bool authorized) {
  HTTPClient http;
  String url = server_ip + String("/update_exit_status.php");
  http.begin(url);
  http.addHeader("Content-Type", "application/json"); // Set content type as JSON

  // Create JSON payload
  DynamicJsonDocument doc(1024);
  doc["uid"] = uid;
  doc["owner_name"] = ownerName;
  doc["car_color"] = carColor;
  doc["license_plate"] = licensePlate;
  doc["Authorized"] = authorized;

  String requestBody;
  serializeJson(doc, requestBody);  // Serialize JSON to string

  int httpCode = http.POST(requestBody); // Send HTTP POST request
  String payload = http.getString();     // Get response
  Serial.println("HTTP Response code: " + String(httpCode));
  Serial.println("Response: " + payload);
  http.end();  // Close connection
}
//-----------------------------------------------------------------------------
// Function to check parking IR sensor status
void checkParkingSensor(int sensorPin, int slotNumber) {
  String currentStatus = parkingStatus[slotNumber - 1];  // Get current status
  if (digitalRead(sensorPin) == LOW) { // Vehicle detected (LOW = detected)
    if (currentStatus == "RESERVED") {
      updateParkingStatus("OCCUPIED", slotNumber); // Change to OCCUPIED if currently is
RESERVED and now vehicle detected
    } else if (currentStatus == "AVAILABLE") {
      updateParkingStatus("OCCUPIED", slotNumber); // Change to OCCUPIED if currently is
AVAILABLE and now vehicle detected
    }
  } else { // No vehicle detected (HIGH = no detected)
    if (currentStatus == "OCCUPIED") {
      updateParkingStatus("AVAILABLE", slotNumber); // Change to AVAILABLE if no vehicle
detected
    }
  }
}
//-----------------------------------------------------------------------------
// Function to fetch latest status from 'parking_status' table in the database (for Parking
Slot LEDs)
void fetchParkingStatusFromDatabase() {
  HTTPClient http;
  String url = server_ip + "/get_parking_status.php";
  http.begin(url);

  int httpCode = http.GET();  // Send HTTP GET request
  if (httpCode == HTTP_CODE_OK) {
    String payload = http.getString();
    DynamicJsonDocument doc(1024);
    deserializeJson(doc, payload);  // Parse JSON response

    // Update each parking slot status
    for (int i = 1; i <= 8; i++) {
      String status = doc[String(i)].as<String>();
      parkingStatus[i - 1] = status; // Update old status with latest fetched status
      controlLEDs(i, status);  // Update the LEDs based on the fetched status
    }
  } else {
    Serial.print("Failed to fetch parking lot status, HTTP code: ");
    Serial.println(httpCode);
  }
  http.end();  // Close connection
}
//-----------------------------------------------------------------------------
//Function to update the 'parking_status' table in the database based on local IR sensors
status
void updateParkingStatus(String newStatus, int slotNumber) {
  parkingStatus[slotNumber - 1] = newStatus;  // Update local status according to IR sensors
status

  HTTPClient http;
  String url = server_ip + String("/update_parking_status.php");
  http.begin(url);
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");

  // Build POST data string with all slot statuses
  String postData = "status1=" + String(parkingStatus[0]) +
                   "&status2=" + String(parkingStatus[1]) +
                   "&status3=" + String(parkingStatus[2]) +
                   "&status4=" + String(parkingStatus[3]) +
                   "&status5=" + String(parkingStatus[4]) +
                   "&status6=" + String(parkingStatus[5]) +
                   "&status7=" + String(parkingStatus[6]) +
                   "&status8=" + String(parkingStatus[7]);
  Serial.print("Sending data to server: ");
  Serial.println(postData);

  int httpCode = http.POST(postData);  // Send HTTP POST request
  if (httpCode == HTTP_CODE_OK) {
    String payload = http.getString();
    Serial.print("Response: ");
    Serial.println(payload);
  } else {
    Serial.print("Failed to update parking status. HTTP code: ");
    Serial.println(httpCode);
    Serial.print("HTTP error: ");
    Serial.println(http.errorToString(httpCode).c_str());
  }
  http.end();
}
//---------------------------------------------------------------------------
// Function to control parking slot LEDs
void controlLEDs(int slotNumber, String status)
{
  int redPin, yellowPin, greenPin; // GPIO Pins on PCF8575 I/O Expanders

  // Adjust pin calculation for slots 1-4 (First PCF8575)
  if (slotNumber <= 4) {
    redPin = slotNumber * 3 - 3; // e.g. Slot 2 RED LED = (2*3)-3 = Pin P03
    yellowPin = slotNumber * 3 - 2; // e.g. Slot 2 Yellow LED = (2*3)-2 = Pin P04
    greenPin = slotNumber * 3 - 1; // e.g. Slot 2 Green LED = (2*3)-1 = Pin P05

    // Set Parking Slot LEDs state based on the status (Active LOW)
    if (status == "OCCUPIED") {
      pcf8575_1.digitalWrite(redPin, LOW);     // Turn on red LED
      pcf8575_1.digitalWrite(yellowPin, HIGH);  // Turn off yellow LED
      pcf8575_1.digitalWrite(greenPin, HIGH);   // Turn off green LED
    } else if (status == "AVAILABLE") {
      pcf8575_1.digitalWrite(redPin, HIGH);     // Turn off red LED
      pcf8575_1.digitalWrite(yellowPin, HIGH);  // Turn off yellow LED
      pcf8575_1.digitalWrite(greenPin, LOW);   // Turn on green LED
    } else if (status == "RESERVED") {
      pcf8575_1.digitalWrite(redPin, HIGH);     // Turn off red LED
      pcf8575_1.digitalWrite(yellowPin, LOW); // Turn on yellow LED
      pcf8575_1.digitalWrite(greenPin, HIGH);   // Turn off green LED
    }
  }
    // Adjust pin calculation for slots 5-8 (Second PCF8575)
    else {
    redPin = (slotNumber - 4) * 3 - 3; // e.g. Slot 6 RED LED = [(6-4)*3]-3 = Pin P03
    yellowPin = (slotNumber - 4) * 3 - 2; // e.g. Slot 6 Yellow LED = [(6-4)*3]-2 = Pin P04
    greenPin = (slotNumber - 4) * 3 - 1; // e.g. Slot 6 Green LED = [(6-4)*3]-1 = Pin P05

    // Set Parking Slot LEDs state based on the status (Active LOW)
    if (status == "OCCUPIED") {
      pcf8575_2.digitalWrite(redPin, LOW);     // Turn on red LED
      pcf8575_2.digitalWrite(yellowPin, HIGH);  // Turn off yellow LED
      pcf8575_2.digitalWrite(greenPin, HIGH);   // Turn off green LED
    } else if (status == "AVAILABLE") {
      pcf8575_2.digitalWrite(redPin, HIGH);     // Turn off red LED
      pcf8575_2.digitalWrite(yellowPin, HIGH);  // Turn off yellow LED
      pcf8575_2.digitalWrite(greenPin, LOW);   // Turn on green LED
    } else if (status == "RESERVED") {
      pcf8575_2.digitalWrite(redPin, HIGH);     // Turn off red LED
      pcf8575_2.digitalWrite(yellowPin, LOW); // Turn on yellow LED
      pcf8575_2.digitalWrite(greenPin, HIGH);   // Turn off green LED
    }
  }
}
```

Figure D-2: Code for Secondary ESP32

```
// Include necessary libraries
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "Base64.h"
#include "esp_camera.h"
//--------------------------------------------------------------------------------
// WiFi credentials
const char* ssid = "WIFI_SSID";
const char* password = "WIFI_PASSWORD";
//--------------------------------------------------------------------------------
// Google Apps Script details for uploading images
const char* myDomain = "script.google.com";
String myScript =
"/macros/s/AKfycbxB3CdeLGaYvvqEBy5UXlmA_au8_Zsu2qFnECzGbkR1MOkRb5B85hGGgCgGGKYEEP6XBw/exec";
String myFilename = "filename=ESP32-CAM.jpg"; // Filename for the uploaded image
String mimeType = "&mimetype=image/jpeg"; // Image MIME type
String myImage = "&data="; // Prefix for image data in POST request

int waitingTime = 30000; // Maximum wait time for server response (30 seconds)
//--------------------------------------------------------------------------------
// ESP32-CAM Pin Definitions
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22
//--------------------------------------------------------------------------------
// IR sensor and onboard LED flash pin definitions
#define IR_PIN 13 // IR sensor input pin
#define LED_PIN 4 // Onboard LED flash pin
//--------------------------------------------------------------------------------
// Function prototypes
void connectWiFi();
void configureCamera();
void captureAndSendImage();
String urlencode(String str);
//--------------------------------------------------------------------------------
void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // Disable brownout detector to prevent resets
  Serial.begin(115200); // Initialize serial communication
  delay(10);

  pinMode(IR_PIN, INPUT); // Set IR sensor pin as input
  pinMode(LED_PIN, OUTPUT); // Set LED flash pin as output
  digitalWrite(LED_PIN, LOW); // Ensure LED is off initially

  connectWiFi(); // Connect to WiFi network
  configureCamera(); // Initialize and configure the camera
}
//--------------------------------------------------------------------------------
void loop() {
  // Check if movement is detected by the IR sensor (LOW = movement detected)
  if (digitalRead(IR_PIN) == LOW) {
    Serial.println("Movement detected! Capturing image...");
    captureAndSendImage(); // Capture image and upload to Google Drive
    delay(3000); // Wait 3 seconds to avoid multiple triggers from the same movement
  } else {
    Serial.println("No movement detected.");
    delay(1000); // Check again in 1 second if no movement is detected
  }
}
//--------------------------------------------------------------------------------
// Function to connect to WiFi
void connectWiFi() {
  WiFi.mode(WIFI_STA); // Set WiFi to station mode
  Serial.print("Connecting to WiFi: ");
  Serial.println(ssid);
  WiFi.begin(ssid, password); // Begin WiFi connection

  // Wait until connected to WiFi
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  Serial.println("\nWiFi connected!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP()); // Print the assigned IP address
}
//--------------------------------------------------------------------------------
// Function to configure the ESP32-CAM camera settings
void configureCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0; // LEDC channel for camera
  config.ledc_timer = LEDC_TIMER_0; // LEDC timer for camera
  config.pin_d0 = Y2_GPIO_NUM; // Camera data pins
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM; // XCLK pin
  config.pin_pclk = PCLK_GPIO_NUM; // PCLK pin
  config.pin_vsync = VSYNC_GPIO_NUM; // VSYNC pin
  config.pin_href = HREF_GPIO_NUM; // HREF pin
  config.pin_sscb_sda = SIOD_GPIO_NUM; // SDA pin
  config.pin_sscb_scl = SIOC_GPIO_NUM; // SCL pin
  config.pin_pwdn = PWDN_GPIO_NUM; // Power-down pin
  config.pin_reset = RESET_GPIO_NUM; // Reset pin
  config.xclk_freq_hz = 20000000; // XCLK frequency (20 MHz)
  config.pixel_format = PIXFORMAT_JPEG; // Set pixel format to JPEG
  config.frame_size = FRAMESIZE_VGA; // Set frame size to VGA (640 x 480)
  config.jpeg_quality = 10; // JPEG quality (lower value = higher quality)
  config.fb_count = 1; // Number of frame buffers

  // Initialize the camera and check for errors
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x\n", err);
    delay(1000);
    ESP.restart(); // Restart the ESP32 if camera initialization fails
  }

  Serial.println("Camera initialized successfully.");
}
//--------------------------------------------------------------------------------
// Function to capture an image and send it to Google Drive
void captureAndSendImage() {
  Serial.println("Turning on LED and capturing image...");
  digitalWrite(LED_PIN, HIGH); // Turn on LED flash for illumination
  delay(100); // Allow LED to reach full brightness

  // Capture image from camera
  camera_fb_t * fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed.");
    digitalWrite(LED_PIN, LOW); // Turn off LED if capture fails
    delay(1000);
    ESP.restart(); // Restart ESP32 if capture fails
    return;
  }

  Serial.println("Image captured. Turning off LED.");
  digitalWrite(LED_PIN, LOW); // Turn off the LED flash

  WiFiClientSecure client;
  client.setInsecure(); // Disable SSL certificate verification

  // Connect to the Google Apps Script server
  if (client.connect(myDomain, 443)) {
    Serial.println("Connection successful. Sending image...");

    // Encode image data to Base64
    char *input = (char *)fb->buf;
    char output[base64_enc_len(3)];
    String imageFile = "";
    for (int i = 0; i < fb->len; i++) {
      base64_encode(output, (input++), 3);
      if (i % 3 == 0) imageFile += urlencode(String(output));
    }
    String Data = myFilename + mimeType + myImage;

    esp_camera_fb_return(fb); // Release the frame buffer to free up memory

    // Send POST request with image data
    client.println("POST " + myScript + " HTTP/1.1");
    client.println("Host: " + String(myDomain));
    client.println("Content-Length: " + String(Data.length() + imageFile.length()));
    client.println("Content-Type: application/x-www-form-urlencoded");
    client.println();
    client.print(Data);
    for (int Index = 0; Index < imageFile.length(); Index += 1000) {
      client.print(imageFile.substring(Index, Index + 1000));
    }

    Serial.println("Waiting for server response...");
    long int StartTime = millis();
    while (!client.available()) {
      Serial.print(".");
      delay(100);
      if ((StartTime + waitingTime) < millis()) {
        Serial.println("\nNo response.");
        break;
      }
    }

    // Print server response
    Serial.println();
    while (client.available()) {
      Serial.print(char(client.read()));
    }
  } else {
    Serial.println("Failed to connect to Google Apps Script.");
  }

  client.stop(); // Close the connection
}
//--------------------------------------------------------------------------------
// Function to URL encode the Base64 image data
String urlencode(String str) {
  String encodedString = "";
  char c;
  char code0;
  char code1;
  for (int i = 0; i < str.length(); i++) {
    c = str.charAt(i);
    if (c == ' ') {
      encodedString += '+';
    } else if (isalnum(c)) {
      encodedString += c;
    } else {
      code1 = (c & 0xf) + '0';
      if ((c & 0xf) > 9) {
        code1 = (c & 0xf) - 10 + 'A';
      }
      c = (c >> 4) & 0xf;
      code0 = c + '0';
      if (c > 9) {
        code0 = c - 10 + 'A';
      }
      encodedString += '%';
      encodedString += code0;
      encodedString += code1;
    }
    yield(); // Allow the system to handle other tasks
  }
  return encodedString;
}
```

Figure D-3: Code for ESP32-CAM (*ESP32_CAM_FYP.ino*)

```javascript
function doPost(e) {
  // Decode the received Base64-encoded image data
  var data = Utilities.base64Decode(e.parameters.data);

  // Generate a timestamped filename for the image (e.g., "20250214_123456.jpg")
  var nombreArchivo = Utilities.formatDate(new Date(), "GMT+8", "yyyyMMdd_HHmmss") + ".jpg";

  // Create a Blob object from the decoded data with the given MIME type and filename
  var blob = Utilities.newBlob(data, e.parameters.mimetype, nombreArchivo);

  // Define the folder name where the image will be saved in Google Drive
  var folderName = "ESP32_CAM_FYP";

  // Search for an existing folder with the specified name in Google Drive
  var folder, folders = DriveApp.getFoldersByName(folderName);

  if (folders.hasNext()) {
    // If the folder exists, use it
    folder = folders.next();
  } else {
    // If the folder doesn't exist, create it
    folder = DriveApp.createFolder(folderName);
  }

  // Save the image file to the designated folder in Google Drive
  var file = folder.createFile(blob);

  // Return a response indicating completion
  return ContentService.createTextOutput('Completed');
}
```

Figure D-4: Code for Google Apps Script