ECOLE NATIONALE
SUPÉRIEURE
D'INFORMATIQUE

الجـــمهوريـــة الجزائريـــة الديمـــقراطيـــة الشـعبيـة
وزارة التعليـــم العـــالي والبـــحث العلمـــي
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

Intelligent and Communicating Systems, ICS

2^{nd} Year Specialty SIQ G02, 2CS SIQ2

Project report

Title:

Smart Parking System
Technical report

Studied by:

SOLTANI Meriem
HEDDADJI Nourelimane
TAIB Salma
ZERROUKI Selma

Contents

List of Figures

Introduction

In the ever-evolving landscape of technological advancements, the integration of Internet of Things (IoT) solutions has significantly reshaped traditional systems across diverse domains, such as home automation, traffic management, and more, contributing to an enhanced quality of life and simplifying mundane and time-consuming tasks.

Motivated by this paradigm shift, we undertook the challenge of developing an intelligent parking system designed to automatically guide users to available parking spaces at the lowest cost through a dedicated mobile application.

This report will exclusively focus on the technical aspects of our intelligent parking system. It delves into the circuit design, providing detailed insights into the connections and functionalities of each component.

Implementation

In this chapter, we delve into the detailed implementation of the Smart Parking System. The following sections outline the user-centered parking process, system architecture, component connections, and the code responsible for the system's functionality.

Each component is thoroughly explained, accompanied by relevant circuit diagrams and code snippets.

1. User-Centred Parking Process

- User accesses the mobile app and enters the booking details.
- Chooses a payment method and completes the transaction.
- Receives payment confirmation and reservation details.
- As the user approaches the parking facility gate, scans their RFID card.
- Raspberry Pi-controlled RFID ID reader verifies the card details and checks with the database for reservation validity.
- Distance sensor measures the proximity of the user's car to the gate.
- Gate opens automatically when the user is close enough. The app starts a countdown timer.
- Gate closes automatically after the vehicle enters, ensuring only authorized users get in.
- Users have the option to extend their booking time.

1.1. System Architecture

The illustration below depicts the system architecture

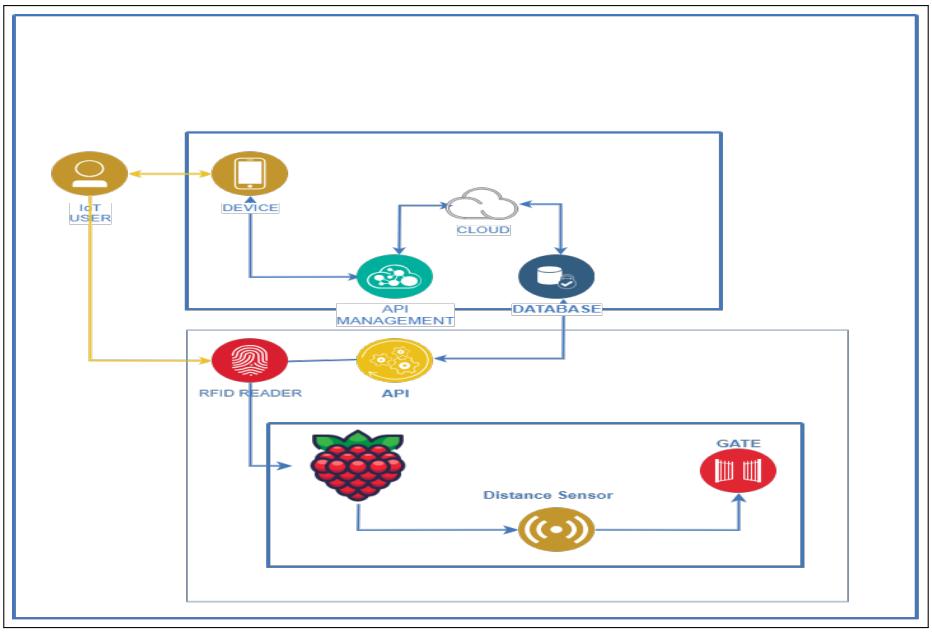


Figure 2.1: System Architecture

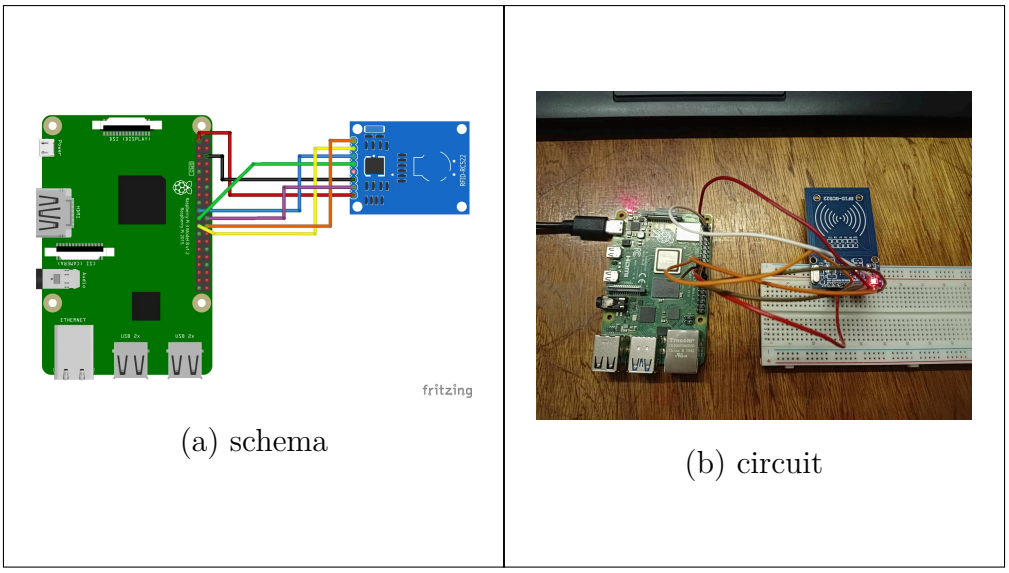
2. Components Connections

2.1. RFID Reader

An RFID reader is a device that uses Radio-Frequency Identification technology to communicate with RFID tags. In the context of our system, the RC522 RFID reader facilitates card identification for parking reservations.

Connect the components as follows:

- Connect the 3v3 pin of the RC522 to Raspberry Pi pin 1 (3v3).
- Connect the RST of the RC522 to Raspberry Pi pin 22.
- Connect the GND of the RC522 to Raspberry Pi to pin 6.
- Connect the MISO pin of the RC522 to Raspberry Pi 21.
- Connect the MOSI pin of the RC522 to of Raspberry Pi pin 19.
- Connect the SCK pin of the RC522 to Raspberry Pi pin 23.
- Connect the SDA pin of the RC522 to Raspberry Pi pin 24.



(a) schema

(b) circuit

Figure 2.2: RFID reader circuit connection

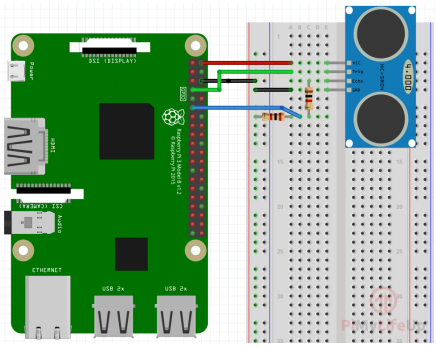
2.2. Distance Sensor

A sensor is a device that detects and measures physical quantities or environmental conditions.

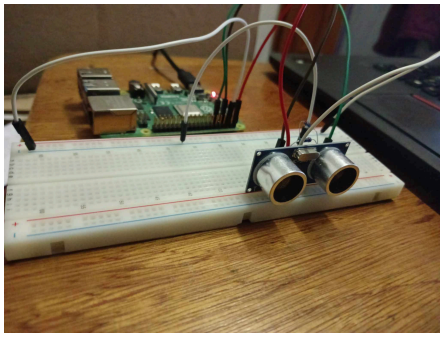
The HC-SR04 distance sensor utilizes ultrasonic signals to measure the distance between the car and the gate. It plays a crucial role in determining when to open the gate for users with parking reservations.

Connect the components as follows:

- Connect the VCC pin of the HC-SR04 to Raspberry Pi pin 2 (5v).
- Connect the GND pin of the HC-SR04 to Raspberry Pi pin 6.
- Connect the trig pin of the HC-SR04 to Raspberry Pi pin 7.
- Connect one end of the first resistor (1kohm) to the echo pin of the HC-SR04.
- Connect the other end of the first resistor to Raspberry Pi pin 11.
- Connect one end of the second resistor (2kohm) to Raspberry Pi pin 11.
- Connect the other end of the second resistor to Raspberry Pi pin 6 (GND).



(a) schema



(b) circuit

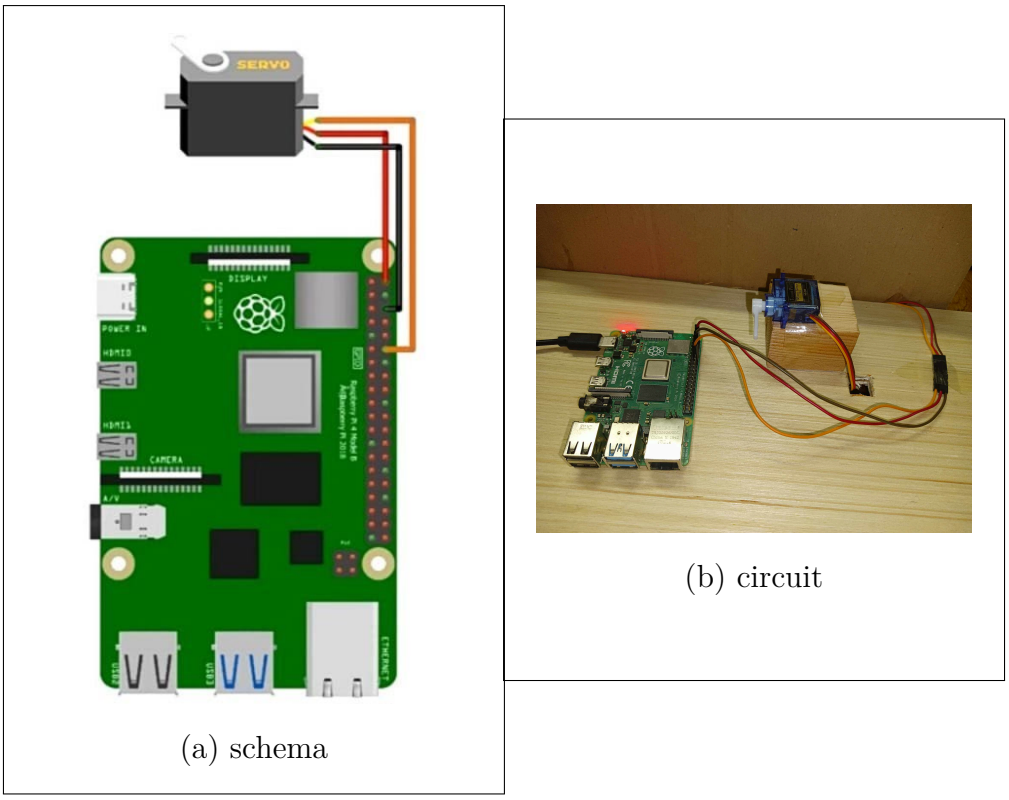
Figure 2.3: Distance Sensor circuit connection

2.3. Servo Motor

A servo motor, specifically the SG90 model, is employed to control the movement of the parking lot gate. It operates based on Pulse Width Modulation (PWM) signals to open and close the gate automatically.

Connect the components as follows:

- Connect the VCC pin (red wire) of the SG90 to Raspberry Pi pin 2 (5v).
- Connect the GND pin (brown wire) of the SG90 to Raspberry Pi pin 6.
- Connect the signal pin (orange wire) of the SG90 of Raspberry Pi pin 12.



(a) schema

(b) circuit

Figure 2.4: Servo motor circuit connection

3. Code Snippets

3.1. Setting up GPIO Pins

This code snippet initializes and configures the GPIO pins on the Raspberry Pi for various components of our smart parking system. Specifically, it defines constants for the trigger pin, echo pin, and servo motor pin. The GPIO pins are set up using the RPi.GPIO library.

```
1
2     import RPi.GPIO as GPIO
3
4     # Global Constants
5     PIN_TRIGGER = 7
6     PIN_ECHO = 11
7     SERVO_PIN = 12
8
9     # Set up GPIO pins
10    GPIO.setmode(GPIO.BOARD)
11    GPIO.setup(PIN_TRIGGER, GPIO.OUT)
12    GPIO.setup(PIN_ECHO, GPIO.IN)
13    GPIO.setup(SERVO_PIN, GPIO.OUT)
14
15    # Set up HC-SR04 sensor
16    GPIO.setup(PIN_TRIGGER, GPIO.OUT)
17    GPIO.setup(PIN_ECHO, GPIO.IN)
```

```
18
19      # Set up servo motor
20      GPIO.setup(SERVO_PIN, GPIO.OUT)
21      servo = GPIO.PWM(SERVO_PIN, 50)
22      servo.start(0)
```

Listing 2.1: Setting up GPIO Pins

3.2. Database Connection

This code establishes a connection to the Supabase database, our cloud databse solution. The `create_client` function from the `supabase` library is used to create a client, and the connection parameters (URL and key) are provided.

```
1
2       from supabase import create_client
3
4       # Set up Supabase connection
5       url = 'https://bgynecsgqrdfrkcyshsz.supabase.co'
6       key = 'your_supabase_key'
7       supabase = create_client(url, key)
```

Listing 2.2: Supabase Connection

3.3. RFID Access Control

This section of code handles RFID access control.

- The `read_rfid()` function uses a SimpleMFRC522 reader to capture RFID card details.
- The `check_pending_bookings` function queries the Supabase database to identify pending bookings for a specific RFID card (user).
- The `check_valid_booking` function ensures the validity of a booking based on date and time, and `update_active_booking` updates the booking status in the database.

```
1
2       def read_rfid():
3           try:
4               reader = SimpleMFRC522()
5               id, _ = reader.read()  # Ignore the text, only capture the ID
6               return id
7           finally:
8               pass
9
10      # Function to check pending bookings
11      def check_pending_bookings(rfid):
12          try:
13              response = supabase.table("booking").select("*").eq("rfid",
    rfid).eq("status", "pending").execute()
```

```
            data = response.data
            return data
        except Exception as e:
            print("Error checking pending bookings:", e)
            return None

    # Function to check if the booking is valid
    def check_valid_booking(booking):
        try:
            # Extract booking details
            start_time = booking['start_time']
            end_time = booking['end_time']
            booking_date = booking['date']

            # Get current date and time
            current_date = datetime.datetime.now().date()
            current_time = datetime.datetime.now().time()

            # Convert start_time and end_time strings to time objects
            start_time_obj = datetime.datetime.strptime(start_time, '%H:%M:%S').time()
            end_time_obj = datetime.datetime.strptime(end_time, '%H:%M:%S').time()
            booking_date_obj = datetime.datetime.strptime(booking_date, '%Y-%m-%d').date()

            # Check if the booking date is the current date
            if booking_date_obj == current_date:
                # Check if the current time is between start_time and end_time
                if start_time_obj <= current_time <= end_time_obj:
                    return True
                else:
                    return False
            else:
                return False
        except Exception as e:
            print("Error checking valid booking:", e)
            return False
    # Function to update the booking status in supabase
    def update_active_booking(rfid):
        try:
            # Get the current time
            current_time = datetime.datetime.now().isoformat()
            # Perform the update operation in the Supabase table
            response = supabase.table("booking").update({"status": "active", "check_in_time": current_time}).eq("rfid", rfid).execute()
        except Exception as e:
            print("Error updating booking status:", e)
```

Listing 2.3: RFID Access Control

3.4. Gate control

This portion of the code includes functions to measure the distance using an ultrasonic sensor (HC-SR04) and control the parking gate using a servo motor (SG90).

The `measure_distance()` function calculates the distance between the sensor and the vehicle, and `control_gate()` orchestrates the opening and closing of the gate.

```
    # Function to measure distance
    def measure_distance():
        try:
            GPIO.output(PIN_TRIGGER, GPIO.HIGH)
            time.sleep(0.00001)
            GPIO.output(PIN_TRIGGER, GPIO.LOW)

            while GPIO.input(PIN_ECHO) == 0:
                pulse_start_time = time.time()
            while GPIO.input(PIN_ECHO) == 1:
                pulse_end_time = time.time()

            pulse_duration = pulse_end_time - pulse_start_time
            distance = round(pulse_duration * 17150, 2)
            return distance
        except Exception as e:
            print("Error measuring distance:", e)
            return None

    # Function to control the gate using the servo motor
    def control_gate():
        try:
            # Open gate
            servo.ChangeDutyCycle(0)
            time.sleep(1)
            servo.ChangeDutyCycle(2)
            time.sleep(1)
            servo.ChangeDutyCycle(4)
            time.sleep(1)
            servo.ChangeDutyCycle(6)
            time.sleep(1)
            servo.ChangeDutyCycle(8)
            time.sleep(6)
            # Close gate
            servo.ChangeDutyCycle(7)
            time.sleep(1)
            servo.ChangeDutyCycle(5)
            time.sleep(1)
            servo.ChangeDutyCycle(3)
            time.sleep(1)
            servo.ChangeDutyCycle(1)
            time.sleep(1)
            servo.ChangeDutyCycle(0)
        except Exception as e:
            print("Error opening gate:", e)
```

Listing 2.4: Gate Control

3.5. Main function

The main function encapsulates the core logic of our smart parking system.

- `read_rfid()` is called in a loop to continually check and read RFID tags from users approaching the gate.
- `check_pending_bookings(rfid)` is invoked to determine if the user has an ongoing and valid parking reservation.
- Upon a valid booking, `update_active_booking(rfid)` is called to transition the booking status from pending to active, triggering the countdown timer in the mobile app.
- The functions `measure_distance()` and `control_gate()` are utilized to manage the opening and closing of the gate based on the proximity of the user's vehicle.

```
# Main function
if __name__ == "__main__":
    try:
        while True:
            # Read RFID tag
            rfid = read_rfid()
            if rfid:
                # Check if the booking is pending
                data = check_pending_bookings(rfid)
                # Check if the booking is valid
                isBookingValid = False
                if data:
                    isBookingValid = check_valid_booking(data[0])
                if isBookingValid:
                    #Activate the distance sensor
                    calculateDistance = True
                    while calculateDistance:
                        distance = measure_distance()
                        #Open the gate
                        if distance and distance < 10:
                            calculateDistance = False
                            control_gate()
                            update_active_booking(rfid)

                        time.sleep(1)
            time.sleep(1)

    except Exception as e:
        print("Error main:", e)
```

Listing 2.5: Main Function

4. System Overview

To provide a comprehensive view of our Smart Parking System, the following images showcase the practical implementation of the entire system:

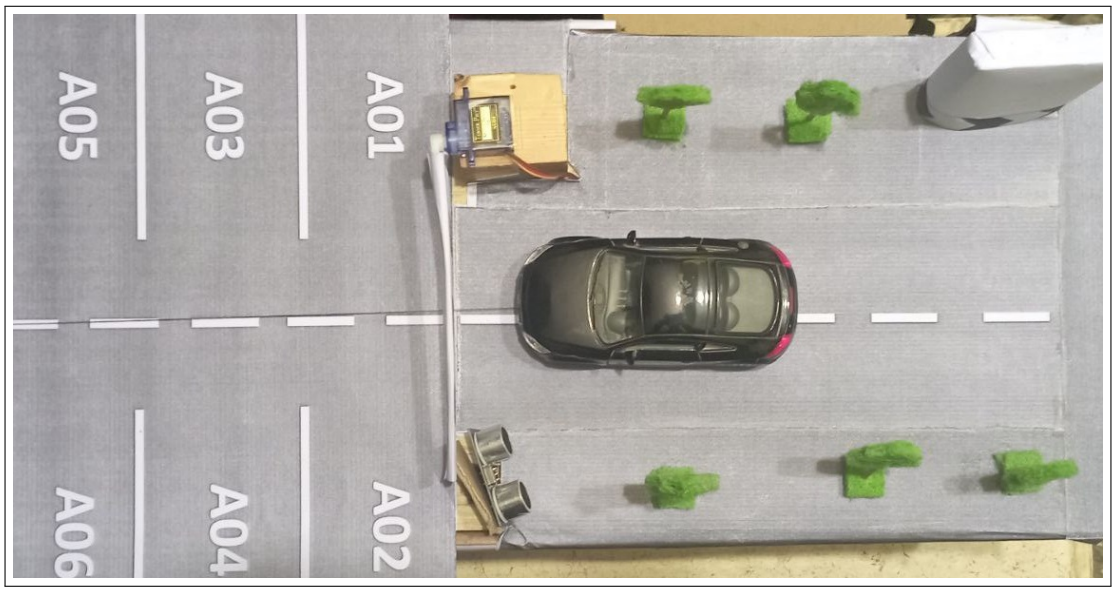


Figure 2.5: System Overview - Smart Parking Implementation

5. Smartphone Application

5.1. Parking Reservation

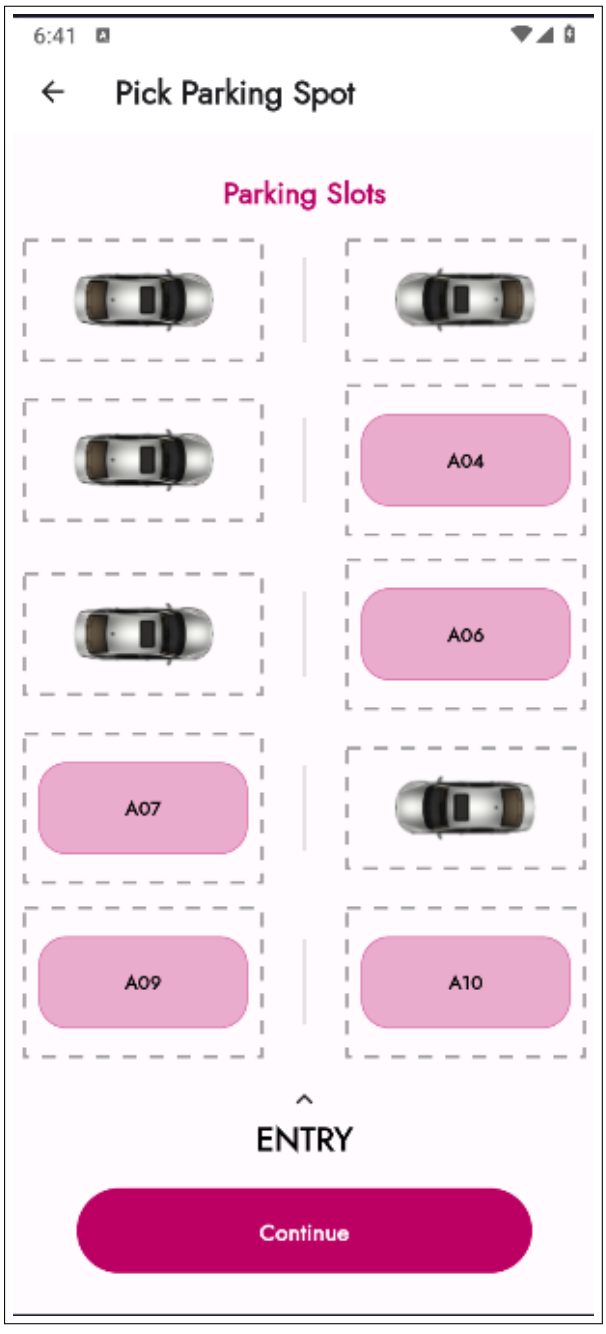


Figure 2.6: Parking Map Page

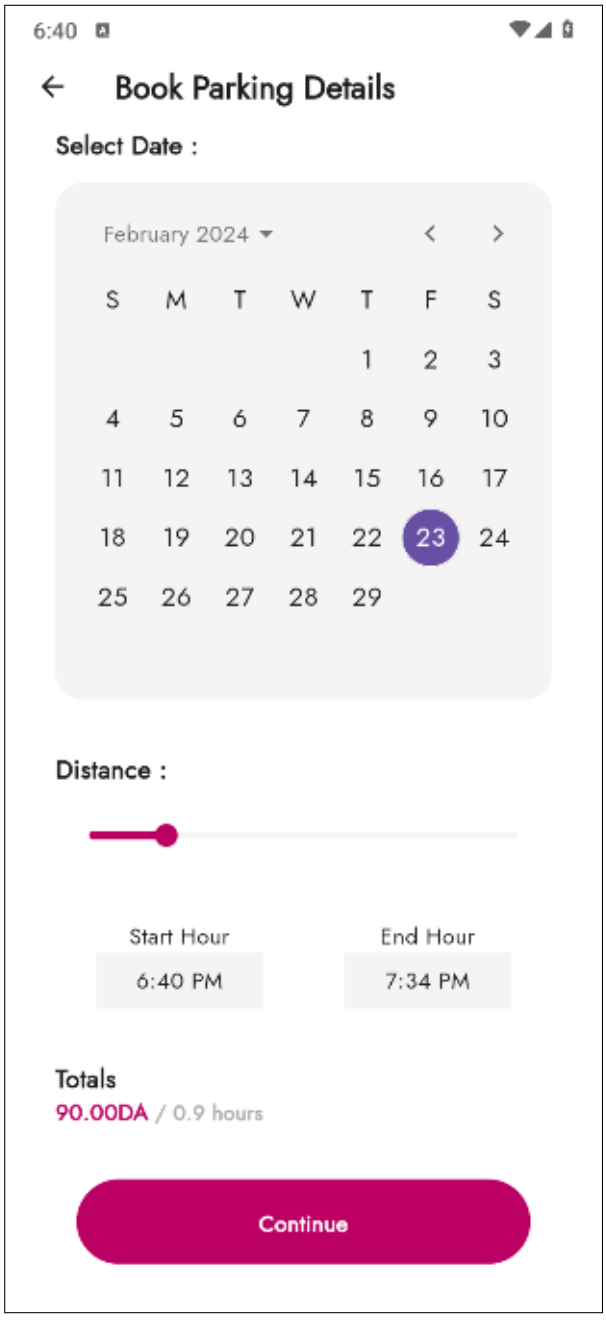


Figure 2.7: Booking Details

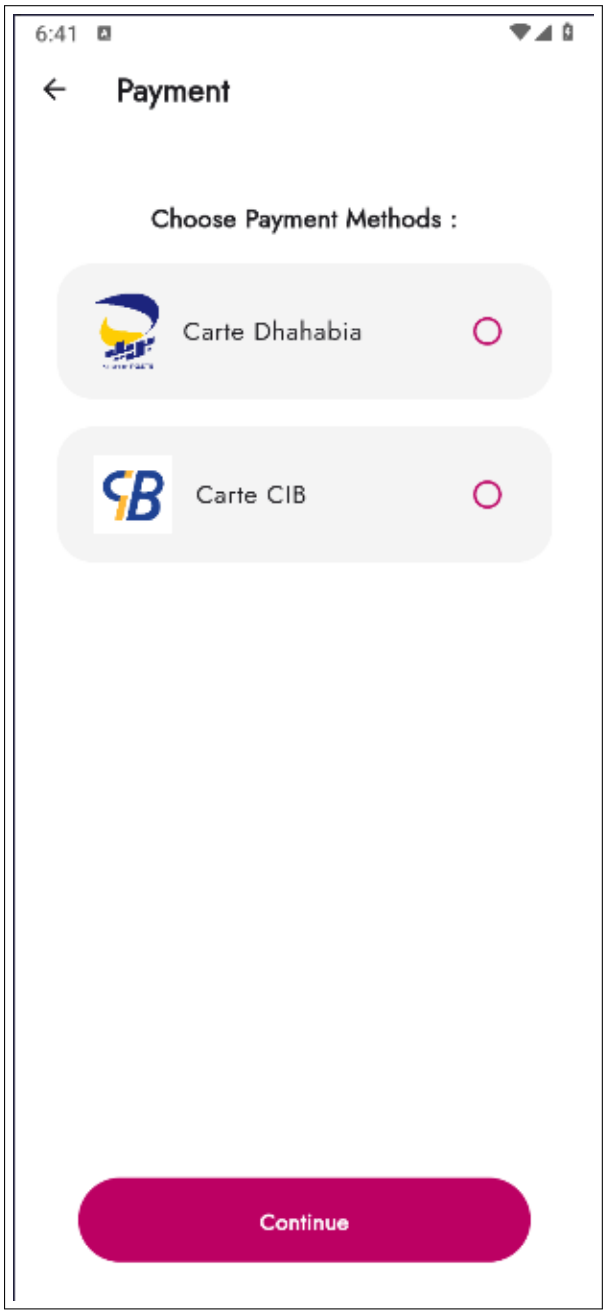


Figure 2.8: Payment Page

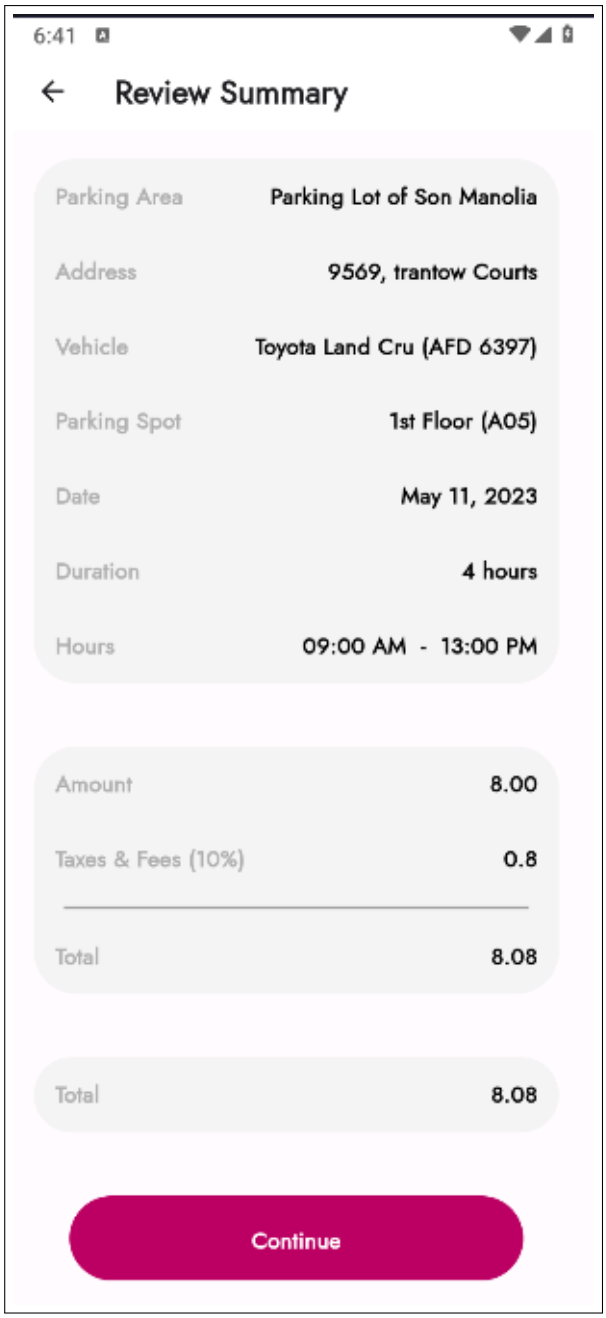


Figure 2.9: Booking Summary Page

5.2. Extend Booking Time

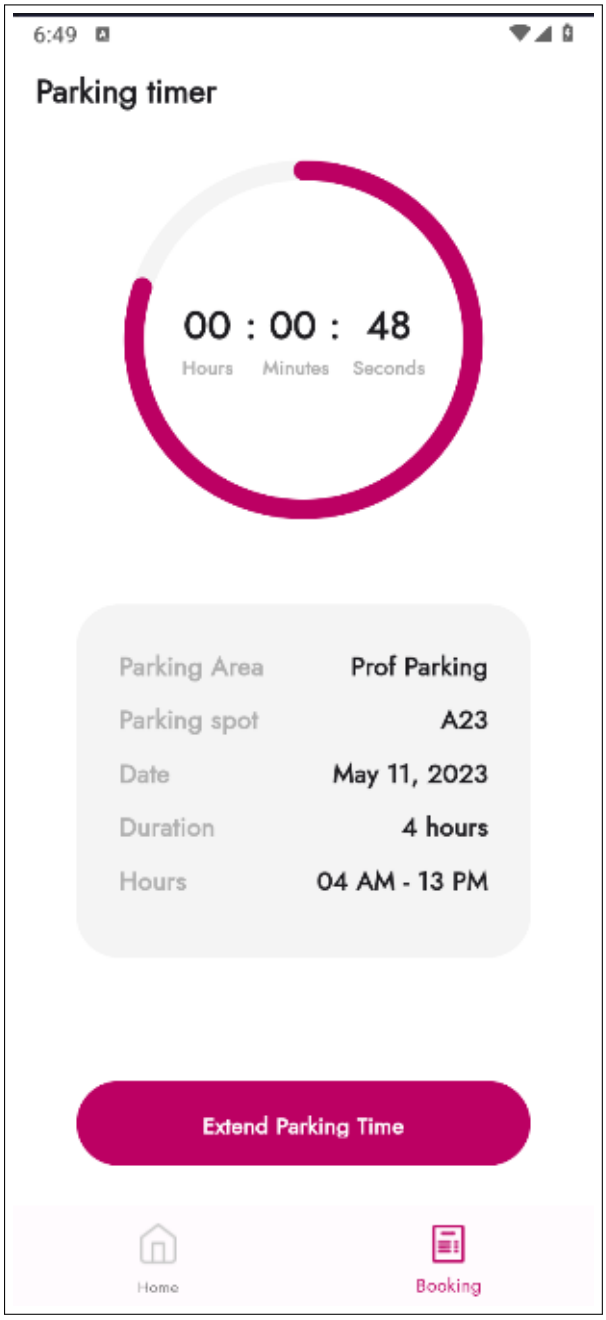


Figure 2.10: Parking Time Page

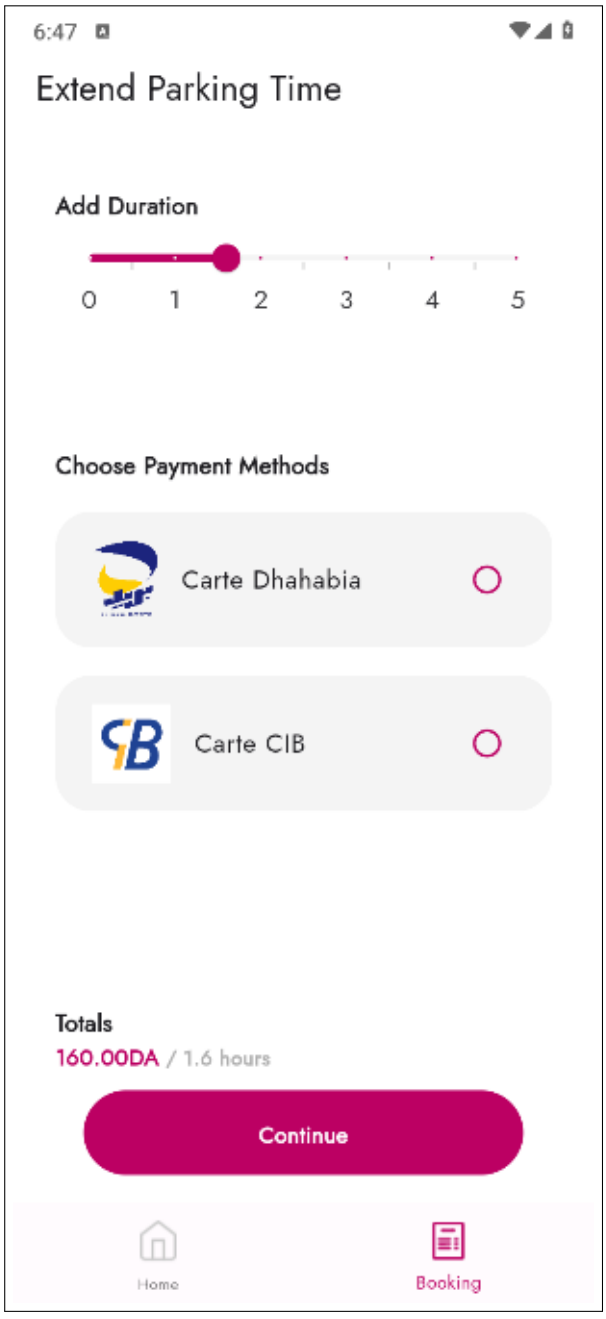


Figure 2.11: Extend Time page

Conclusion

In conclusion, the development of our intelligent parking system has been a journey marked by challenges, innovation, and a commitment to improving urban mobility.

The implementation of RFID technology, distance sensors, and servo motors has resulted in a robust and secure parking solution. Users can effortlessly navigate the application to reserve parking slots, while our system, driven by a Raspberry Pi-based infrastructure, ensures smooth entry and exit procedures.

While initially designed to meet the parking challenges within educational institutions, it is important to emphasize the adaptability and scalability of our solution. The modular and scalable design of our system enables seamless integration into various urban environments, making it applicable beyond educational institutions. Our technology and principles can be easily generalized, offering a versatile solution to address parking challenges in diverse settings.

Looking to the future, our project perspectives open avenues for further refinement and expansion. We aspire to explore additional features for the mobile app, providing users with a more personalized and dynamic parking experience. Investigating alternative user authentication systems, like camera recognition, presents exciting possibilities for enhanced security and convenience.

In essence, our smart parking system represents a contribution to the evolution of intelligent urban environments. Our commitment to continuous improvement and innovation drives us to explore new horizons and embrace the possibilities that lie ahead.

Appendix

In this appendix, you will find the complete source code for the Smart Parking System.

```
import RPi.GPIO as GPIO
import time
import datetime
from mfrc522 import SimpleMFRC522
from supabase import create_client

# Set up GPIO pins
GPIO.setmode(GPIO.BOARD)
PIN_TRIGGER = 7
PIN_ECHO = 11
SERVO_PIN = 12

# Set up Supabase connection
url = 'https://bgynecsgqrdfrkcyshsz.supabase.co'
key = *****
supabase = create_client(url, key)

# Set up HC-SR04 sensor
GPIO.setup(PIN_TRIGGER, GPIO.OUT)
GPIO.setup(PIN_ECHO, GPIO.IN)

# Set up servo motor
GPIO.setup(SERVO_PIN, GPIO.OUT)
servo = GPIO.PWM(SERVO_PIN, 50)
servo.start(0)

# Function to read RFID tag
def read_rfid():
    try:
        reader = SimpleMFRC522()
        id, _ = reader.read()  # Ignore the text, only capture the ID
        return id
    finally:
        pass

# Function to check pending bookings
def check_pending_bookings(rfid):
    try:
        response = supabase.table("booking").select("*").eq("rfid",
rfid).eq("status", "pending").execute()
        data = response.data
        return data
    except Exception as e:
        print("Error checking pending bookings:", e)
        return None

# Function to check if the booking is valid
def check_valid_booking(booking):
    try:
        # Extract booking details
        start_time = booking['start_time']
```

```
            end_time = booking['end_time']
            booking_date = booking['date']

            # Get current date and time
            current_date = datetime.datetime.now().date()
            current_time = datetime.datetime.now().time()

            # Convert start_time and end_time strings to time objects
            start_time_obj = datetime.datetime.strptime(start_time, '%H:%M:%S').time()
            end_time_obj = datetime.datetime.strptime(end_time, '%H:%M:%S').time()
            booking_date_obj = datetime.datetime.strptime(booking_date, '%Y-%m-%d').date()

            # Check if the booking date is the current date
            if booking_date_obj == current_date:
                # Check if the current time is between start_time and end_time
                if start_time_obj <= current_time <= end_time_obj:
                    return True
                else:
                    return False
            else:
                return False
        except Exception as e:
            print("Error checking valid booking:", e)
            return False

    # Function to measure distance
    def measure_distance():
        try:
            GPIO.output(PIN_TRIGGER, GPIO.HIGH)
            time.sleep(0.00001)
            GPIO.output(PIN_TRIGGER, GPIO.LOW)

            while GPIO.input(PIN_ECHO) == 0:
                pulse_start_time = time.time()
            while GPIO.input(PIN_ECHO) == 1:
                pulse_end_time = time.time()

            pulse_duration = pulse_end_time - pulse_start_time
            distance = round(pulse_duration * 17150, 2)
            return distance
        except Exception as e:
            print("Error measuring distance:", e)
            return None

    # Function to control the gate using the servo motor
    def control_gate():
        try:
            # Open gate
            servo.ChangeDutyCycle(0)
            time.sleep(1)
            servo.ChangeDutyCycle(2)
            time.sleep(1)
            servo.ChangeDutyCycle(4)
            time.sleep(1)
```

```
105             servo.ChangeDutyCycle(6)
106             time.sleep(1)
107             servo.ChangeDutyCycle(8)
108             time.sleep(6)
109             # Close gate
110             servo.ChangeDutyCycle(7)
111             time.sleep(1)
112             servo.ChangeDutyCycle(5)
113             time.sleep(1)
114             servo.ChangeDutyCycle(3)
115             time.sleep(1)
116             servo.ChangeDutyCycle(1)
117             time.sleep(1)
118             servo.ChangeDutyCycle(0)
119         except Exception as e:
120             print("Error opening gate:", e)
121
122     # Function to update the booking status in supabase
123     def update_active_booking(rfid):
124         try:
125             # Get the current time
126             current_time = datetime.datetime.now().isoformat()
127             # Perform the update operation in the Supabase table
128             response = supabase.table("booking").update({"status": "active"
    , "check_in_time": current_time}).eq("rfid", rfid).execute()
129         except Exception as e:
130             print("Error updating booking status:", e)
131     # Main function
132     if __name__ == "__main__":
133         try:
134             while True:
135                 # Read RFID tag
136                 rfid = read_rfid()
137                 if rfid:
138                     # Check if the booking is pending
139                     data = check_pending_bookings(rfid)
140                     # Check if the booking is valid
141                     isBookingValid = False
142                     if data:
143                         isBookingValid = check_valid_booking(data[0])
144                     if isBookingValid:
145                         #Activate the distance sensor
146                         calculateDistance = True
147                         while calculateDistance:
148                             distance = measure_distance()
149                             #Open the gate
150                             if distance and distance < 10:
151                                 calculateDistance = False
152                                 control_gate()
153                                 update_active_booking(rfid)
154
155                             time.sleep(1)
156                 time.sleep(1)
157
158         except Exception as e:
159             print("Error main:", e)
```

Listing A.1: full code