SMART PARKING PROJECT

-TEAM NAME:Proj_224689_Team_2

TEAM MEMBERS:

1. R. SATHANA
2. N.C. SRI VAIBAVA LAKSHMI
3. S. HIMRITHA
4. P. MAHALAKSHMI
5. N. KAVITHA

PHASE 1:

PROBLEM DEFINITION AND DESIGN THINKING

PHASE 2:

INNOVATION AND PREDICTIVE MAINTENANCE

PHASE 3:

DEVELOPMENT PART 1

PHASE 4:

DEVELOPMENT PART 2

PHASE 5:

PROJECT DOCUMENTATION & SUBMISSION

TABLE OF CONTENT

SMART PARKING PROJECT

DEFINING COMPREHENSIVE OBJECTIVES

INTRODUCTION:

The Smart Parking project represents a visionary initiative aimed at transforming the traditional parking experience through the application of design thinking principles. This document delineates the specific objectives that will steer the project towards success. The primary goals encompass real-time parking space monitoring, mobile app integration, and the establishment of an efficient parking guidance system, all central components of the broader "Smart Parking" concept.

OBJECTIVE 1: REAL-TIME PARKING SPACE MONITORING

Objective Statement:

The project's foremost objective is the implementation of a robust real-time parking space monitoring system to enhance user convenience and optimize parking space utilization, a critical aspect of Smart Parking.

Key Components and Technologies:

- Deployment of advanced IoT (Internet of Things) sensors in parking spaces.
- Integration with a centralized monitoring system utilizing cloud computing infrastructure.
- Real-time display of parking space availability through digital signage and mobile applications, requiring touchscreen displays and mobile app development.
- Implementation of sophisticated data analytics algorithms, potentially leveraging machine learning models, to optimize parking space allocation based on historical usage patterns and real-time demand, involving data storage, processing, and analytics infrastructure.

OBJECTIVE 2: MOBILE APP INTEGRATION

Objective Statement:

To facilitate a seamless and user-friendly parking experience, the project will develop a dedicated mobile application designed to complement the real-time monitoring system, a core feature of Smart Parking.

Key Components and Technologies:

- Development of an intuitive and user-friendly mobile app interface using cross-platform mobile app development frameworks such as React Native or Flutter.
- Seamless integration with the real-time parking space monitoring system, facilitated through RESTful APIs (Application Programming Interfaces).
- Inclusion of user-centric features such as parking space reservations, secure payment processing using mobile payment gateways, and turn-by-turn navigation powered by GPS technology, involving GPS receivers and payment processing modules.
- Establishment of a user feedback and rating mechanism within the app, potentially integrating with cloud-based databases for data storage and analysis, requiring cloud infrastructure and databases.

OBJECTIVE 3: EFFICIENT PARKING GUIDANCE

Objective Statement:

To reduce the time and effort required for parking space identification, the project will implement an efficient parking guidance system, a key component of Smart Parking.

Key Components and Technologies:

- Installation of dynamic signage and LED indicators at strategic locations within parking areas, potentially connected through a wireless network, utilizing LED displays and wireless communication technology.

- Integration of the parking guidance system with the mobile app, providing users with real-time turn-by-turn navigation to the nearest available parking spots, leveraging GPS and location-based services.
- Leveraging machine learning algorithms for predictive parking space availability based on historical data and real-time inputs, involving machine learning models and data processing infrastructure.
- Implementation of traffic management strategies, potentially incorporating smart traffic lights and cameras for real-time traffic monitoring and control within the parking area, requiring traffic management hardware and software.

PARKING PROBLEM FORMULATIONS AND PROPOSED SOLUTIONS

Parking Problems:

1. **Limited Parking Space Availability:**

   Urban areas often face a shortage of parking spaces, leading to frustration and extended search times for drivers.

Solution:

Implementing real-time monitoring and optimization of parking space allocation will help make the most efficient use of available parking spaces, aligning with the Smart Parking concept.

2. **Inefficient Space Utilization:**

   Lack of real-time information about parking space availability can result in inefficient use of parking areas.

Solution:

Real-time monitoring combined with data analytics can optimize parking space allocation based on actual demand, ensuring efficient utilization, in line with Smart Parking principles.

3. **Congestion and Traffic Jams:**

   Parking areas can become congested, leading to traffic jams and delays, a challenge that Smart Parking systems aim to address.

Solution:

Efficient parking guidance through dynamic signage and mobile app navigation can alleviate congestion by directing drivers to available parking spots, promoting smoother traffic flow, and supporting the Smart Parking concept.

4. **Difficulty in Finding Parking:**

   Drivers often struggle to locate available parking spaces quickly, a problem addressed through Smart Parking solutions.

Solution:

Predictive analytics can forecast parking space availability, while a user-friendly mobile app can guide drivers to the nearest open spots, aligning with the core tenets of Smart Parking.

5. **Environmental Impact:**

   Prolonged searches for parking contribute to increased emissions and pollution, a concern that Smart Parking initiatives aim to mitigate.

Solution:

Efficient parking guidance systems reduce the time spent searching for parking, thus minimizing the environmental impact, in line with Smart Parking's sustainability objectives.

KEY ASPECTS AND COMPONENTS OF SMART PARKING

Smart parking systems involve various key aspects and components to make parking more efficient and convenient.

Here are the key aspects and components of smart parking:

1. SENSORS:

- **Ultrasonic Sensors:**

  These sensors are installed in parking spaces to detect the presence of vehicles.

- **Magnetic Sensors:**

  These sensors use magnetic fields to detect vehicle presence.

- **Infrared Sensors:**

  Infrared sensors can sense the heat emitted by vehicles.

- **Camera-based Sensors:**

  Cameras can be used for visual monitoring of parking spaces.

2. COMMUNICATION INFRASTRUCTURE:

- **Wireless Networks:**

  To transmit data from sensors to a central server or database.

- **Internet Connectivity:**

  To enable users to access real-time parking information through mobile apps or websites.

3. CENTRALIZED SERVER (OR) CLOUD INFRASTRUCTURE:

- To collect and process data from sensors.
- To provide real-time information to users and operators.

4. DATA PROCESSING AND ANALYTICS:

- Algorithms and software for processing sensor data.
- Predictive analytics to forecast parking space availability.
- Historical data analysis for optimization.

5. USER INTERFACES:

- **Mobile Apps:**

  User-friendly mobile applications for drivers to find, reserve, and pay for parking spaces.

- **Digital Signage:**

  Dynamic displays showing real-time parking availability and guidance within parking areas.

- **Web Portals:**

  Online platforms for users to access parking information and make reservations.

6. PAYMENT SYSTEMS:

- Integration with mobile payment gateways and payment processing systems for seamless payments.

7. NAVIGATION AND GUIDANCE SYSTEMS:

- GPS-based navigation to guide drivers to available parking spaces.

- Dynamic signage and LED indicators for real-time parking guidance.

8. Security Systems:

- Surveillance cameras and security measures to ensure the safety of parked vehicles and users.

9. User Feedback Mechanisms:

- Ratings and feedback systems in mobile apps to gather user opinions and improve services.

10. Energy Management:

- Efficient energy use for sensors, signage, and lighting to reduce operational costs.

11. Smart Traffic Management Integration:

- Integration with traffic management systems to manage traffic flow within parking areas.

12. Environmental Sensors:

- Monitoring air quality and emissions to support sustainability goals.

13. Scalability and Expansion:

- Systems designed for easy expansion to accommodate growing demand.

14. Data Security and Privacy Measures:

- Protection of user data and privacy through encryption and access controls.

15. MAINTENANCE AND MONITORING:

- Regular maintenance of sensors and systems to ensure reliability.
- Real-time monitoring of system health and performance.

16. ACCESSIBILITY FEATURES:

- Provisions for accessible parking and user interfaces for individuals with disabilities.

17. REPORTING AND ANALYTICS TOOLS:

- Reporting tools for operators to analyse parking trends and make data-driven decisions.

18. SUSTAINABILITY AND GREEN INITIATIVES:

- Integration of electric vehicle charging stations and support for eco-friendly transportation.

These components and aspects work together to create a smart parking ecosystem that enhances the parking experience for users, optimizes space utilization, and contributes to more efficient urban mobility while addressing environmental and accessibility concerns.

OUR PROJECT :

The Smart Parking project is poised to make a significant impact on the parking industry by adopting a design thinking approach and pursuing specific objectives that encompass the use of advanced components and technologies, including:

- IoT sensors
- Cloud computing
- Cross-platform mobile app development

- GPS and location-based services
- Machine learning algorithms
- Smart traffic management components

These objectives and solutions collectively aim to elevate the parking experience for users, optimize parking space utilization, and contribute to the overall improvement of urban mobility through cutting-edge technologies and innovative solutions, aligning with the principles of Smart Parking for smarter, more efficient parking solutions.

INTEGRATED CAMERA-BASED SOLUTION FOR PARKING SPACE AVAILABILITY DETECTION

INTRODUCTION:

This part outlines our collaborative deployment of camera-based IoT solutions for parking space management. By combining our expertise in image processing and mathematical formulas, this system aims to accurately detect parking space availability and enhance parking asset utilization.

OBJECTIVE:

Our team's primary objectives in developing this solution are as follows:

1. **ACCURATELY DETECTING PARKING SPACE AVAILABILITY:** Through the collective effort of our team members, we aim to provide real-time data on available parking spaces.
2. **OPTIMIZING PARKING SPACE ALLOCATION:**

   Leveraging the skills and knowledge of our team, we intend to optimize parking space allocation for maximum efficiency.
3. **REDUCING CONGESTION:**

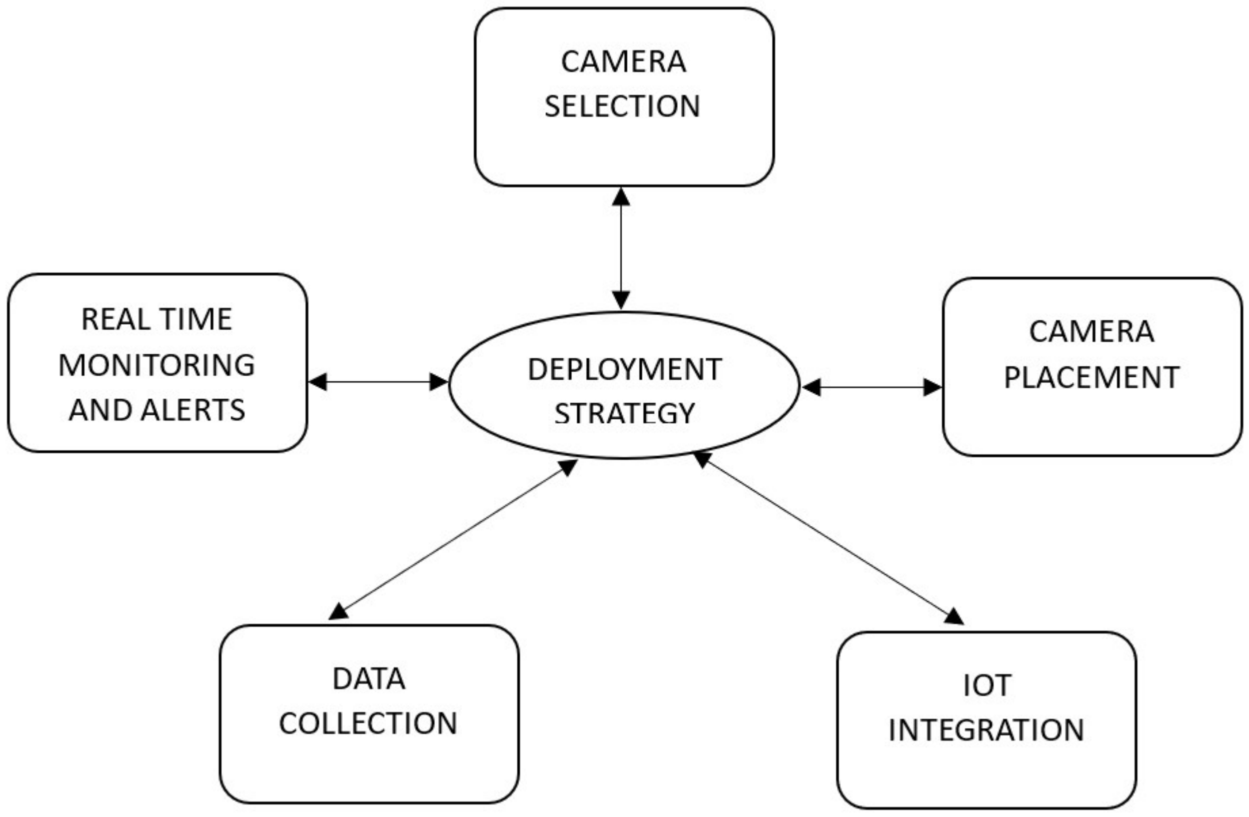
   By providing users with real-time parking information, we collectively strive to reduce traffic congestion in and around our parking facilities.
4. **ENHANCING USER EXPERIENCES:**

   Our team is committed to delivering a seamless and stress-free parking experience for our customers.

VIRTUAL DEPLOYMENT STRATEGY:

1. **CAMERA SELECTION:**
   Our team has diligently selected cameras with the following features:

- High-resolution image capture.
- Wide-angle lenses to maximize coverage.
- Low-light capabilities for 24/7 operation.
- Seamless integration capabilities with our IoT platform.



2. **CAMERA PLACEMENT:**
   Collectively, we've strategically positioned our cameras to ensure optimal coverage of our parking spaces. To determine the number of cameras needed for a given area, we used the following formula:

Number of Cameras = (Total Parking Spaces * Coverage Factor) / Camera Coverage Per Camera

Where:

- Total Parking Spaces: The total number of parking spaces in the area.
- Coverage Factor: An adjustment factor accounting for blind spots and overlapping coverage.
- Camera Coverage Per Camera: The number of parking spaces one camera can effectively cover.

3. **IOT INTEGRATION:**
   Our combined efforts have led to the seamless integration of cameras with our IoT platform, enhancing data accuracy and providing real-time monitoring capabilities.
4. **DATA COLLECTION:**
   We've implemented IoT sensor-based data collection to accurately detect parking space occupancy. The following formula calculates the parking space occupancy rate:

Occupancy Rate (%) = (Occupied Spaces / Total Spaces) * 100

Where:

- Occupied Spaces: The number of parking spaces currently occupied.
- Total Spaces: The total number of parking spaces in the area.

5. **REAL-TIME MONITORING AND ALERTS:**
   Collectively, we've enabled real-time monitoring to notify users when parking occupancy reaches a predefined threshold, improving user experiences and reducing congestion.

DATA ANALYSIS AND OPTIMIZATION:

1. **DATA STORAGE AND ANALYSIS:**
   Our team has worked together to store historical occupancy data for analysis. We use the following formula to calculate average occupancy over a specified period:

Average Occupancy (%) = (Total Occupied Spaces / Total Spaces) * 100

Where:

- Total Occupied Spaces: The sum of occupied spaces over the specified period.
- Total Spaces: The total number of parking spaces in the area.

2. **OPTIMIZATION:**
   Together, we've developed advanced algorithms to optimize parking space allocation based on historical data. The following formula calculates the parking space optimization rate:

Optimization Rate (%) = (Optimized Spaces / Total Spaces) * 100

Where:

- Optimized Spaces: The number of parking spaces reallocated for better utilization.
- Total Spaces: The total number of parking spaces in the area.

PRIVACY AND COMPLIANCE:

1. **PRIVACY CONSIDERATIONS:**
   Privacy concerns have been meticulously addressed collectively, ensuring compliance with local regulations and guidelines regarding data collection and storage.

2. **COMPLIANCE:**
   Our team is collectively committed to full compliance with data protection laws.

To measure our compliance level, we use the following formula:

Compliance Level (%) = (Compliance Score / Maximum Possible Score) * 100

Where:

- Compliance Score: A score assigned based on our collective adherence to data protection regulations.
- Maximum Possible Score: The highest achievable score for full compliance.

This part of project represents our team's collective dedication to innovation and excellence in parking space management. Through the collaborative deployment of camera-based IoT solutions and our joint focus on data privacy and compliance, we aim to deliver an outstanding parking experience for all our users.

SMART PARKING SYSTEM WITH IOT SENSORS

The "Smart Parking System with IoT Sensors" project is dedicated to developing an intelligent and efficient parking management solution using IoT sensors and the Raspberry Pi platform.

This part of project documents the progress achieved in Development Part 1, emphasizing sensor selection, Raspberry Pi integration, Python script development for data collection, and secure data transmission to a cloud or mobile app server.

PROBLEM STATEMENT

In urban environments, parking management is a significant challenge. Drivers often struggle to find available parking spaces, leading to congestion and wasted time. Furthermore, parking facilities face difficulties in monitoring space occupancy efficiently.

Our project addresses these issues by creating a system that provides real-time parking space availability information to both drivers and facility operators.

TARGET AND METHODOLOGY

TARGET:

1. Develop a reliable system for detecting parking space occupancy.
2. Create a user-friendly mobile app for real-time parking space availability updates.
3. Implement data transmission to a designated cloud or mobile app server for real-time monitoring.
4. Ensure data security and privacy during transmission.

METHODOLOGY:

MILESTONE 1: IOT SENSOR SELECTION AND SETUP

- **Sensor Selection:**

  HC-SR04 ultrasonic sensors were chosen for their accuracy and cost-effectiveness.

- **Sensor Configuration:**

  Ultrasonic sensors were connected to a Raspberry Pi, configured to measure distances and detect objects, particularly vehicles in parking spaces.

MILESTONE 2: RASPBERRY PI INTEGRATION

- **Role of Raspberry Pi:**

  The Raspberry Pi serves as the central control unit, receiving data from the sensors.

- **GPIO Interface:**

  The GPIO pins of the Raspberry Pi were configured to trigger measurements and collect data.

MILESTONE 3: PYTHON SCRIPTS FOR DATA COLLECTION AND DATA TRANSMISSION

- **Data Collection and Sensor Interface:**

  Python scripts were developed to collect data from the ultrasonic sensors accurately.

- **Data Transmission:**

  Data collected from the sensors was transmitted to a designated cloud platform or mobile app server for real-time monitoring.

SOLUTION AND FUTURE DEVELOPMENT

SOLUTION AND DISCUSSION

In Development Part 1, we successfully addressed the initial project milestones:

1. **Sensor Selection**:

   HC-SR04 ultrasonic sensors were chosen for their precision and cost-effectiveness. These sensors have a wide detection range and are well-suited for accurate distance measurements in parking spaces. The choice of sensors ensures that the system can reliably detect vehicle presence.

2. **Raspberry Pi Integration**:

   The Raspberry Pi plays a central role in our solution. It serves as the core controller, responsible for orchestrating data collection from the ultrasonic sensors and facilitating communication with the designated cloud platform or mobile app server. The Raspberry Pi's versatility and GPIO capabilities make it an ideal choice for this task.

3. **Data Collection and Sensor Interface**:

   The development of Python scripts to handle data collection from the ultrasonic sensors is a critical aspect of our solution. These scripts create a seamless interface between the Raspberry Pi and the sensors, allowing for efficient and precise data capture. They are designed to:

   - Trigger the sensors to initiate distance measurements by sending ultrasonic signals.
   - Calculate precise distance measurements based on the time it takes for ultrasonic waves to bounce back from objects, particularly vehicles in parking spaces.
   - Capture data from the sensors at regular intervals and store it for further processing. This ensures a continuous flow of information about parking space occupancy.

The use of Python as the programming language simplifies script development and provides the flexibility to make real-time adjustments and enhancements.

4. **Data Transmission**:

   The Python scripts were extended to include data transmission capabilities. This addition allows the system to share parking space occupancy data with a designated cloud platform or mobile app server in real-time. The implementation of data transmission ensures that users have access to up-to-the-minute information about available parking spaces.

   - **Security and Efficiency**: Security and efficiency were paramount in the data transmission process. Robust encryption measures and data integrity checks were implemented to safeguard user data during transmission. Additionally, the transmission process was optimized for efficiency, ensuring that data is transferred seamlessly to the designated server.
   - **Continuous Monitoring**: The main loop within the Python scripts enables continuous data collection and transmission. This loop ensures that parking space occupancy data remains current and relevant. Users can trust the system to provide real-time updates, contributing to an improved parking experience.

FUTURE DEVELOPMENT :

1. **User-Friendly Mobile App**:

   We plan to create a user-friendly mobile app that enables drivers to easily access real-time parking space availability information. The app will feature an intuitive interface and provide location-based services, making it a valuable tool for urban commuters.

2. **Advanced Detection Algorithms**:

   To further enhance the system's capabilities, we will implement advanced machine learning algorithms. These algorithms will improve detection accuracy, enabling the system to differentiate between various types of vehicles and even detect anomalies or obstructions in parking spaces.

3. **<u>Security and Scalability</u>**:

   As the system gains more users, we will concentrate on enhancing its security and scalability. Measures will be taken to fortify data encryption and access controls. Scalability improvements will ensure the system's performance and responsiveness as it accommodates a growing user base.

REAL TIME PARKING AVAILABILITY

Creating a mobile app in Python for real-time parking availability using a framework like Flutter is a great choice. Flutter allows you to build natively compiled applications for mobile, web, and desktop from a single codebase. Here, we'll outline the steps to develop the app.

1.DEVELOPMENT ENVIRONMENT SETUP:

We've ensured that we have Flutter and Dart installed. Following the official Flutter installation guide for our specific platform made it easy to get everything set up.

2. CREATE A NEW FLUTTER PROJECT:

Using the Flutter CLI, we created a new project with the following command:

CODE:

```
flutter create parking_availability_app
```

This command generated a new Flutter project named "parking_availability_app," and we navigated to the project directory with:

CODE:

```
cd parking_availability_app
```

3. DESIGNING OUR USER INTERFACE:

We defined our UI widgets to showcase parking availability data. The widget-based approach in Flutter allowed us to design our user interface effectively. In the lib/main.dart file, we created a screen displaying a list of parking locations along with their availability status.

4. Fetch and Display Real-Time Data:

To display real-time parking availability data received from the Raspberry Pi, need to be implemented for data retrieval and for updating the UI accordingly. A package like http is used to make HTTP requests to Raspberry Pi server.

5.Displaying Real-Time Data:

Once we fetched the data, we updated the UI with the parking availability information. A ListView was employed to present a list of parking locations, their availability status, and other relevant details.

RASPBERRY PI SIMULATION AND PYTHON INTEGRATION

CREATING A RASPBERRY PI SIMULATION IN WOKWI TO FIND PARKING OCCUPANCY

It involves setting up a virtual Raspberry Pi and Simulating Parking Sensors.

Step 1: Create a Wokwi Account

To start, we need to create a Wokwi account if we don't have one already. This account is essential for creating and saving our Raspberry Pi simulations.

Step 2: Begin a New Project

1. After logging into our Wokwi account, we can initiate a new project.
2. By clicking on "New Project," we can set up a fresh simulation.

Step 3: Raspberry Pi Setup

1. To add a virtual Raspberry Pi to our project, we should click on the "Add Component" button.
2. Searching for "Raspberry Pi" and selecting it will introduce the Raspberry Pi into our project.

Step 4: Incorporate Parking Sensors

1. Once more, we can click the "Add Component" button.
2. In this case, we're looking for either an "Ultrasonic Sensor" or an "IR Sensor," depending on our preference for parking occupancy detection. We can select the sensor type that suits our requirements.
3. This will integrate the chosen parking sensor into our project.

Step 5: Connect the Components

Now, it's time to make the physical connections between the Raspberry Pi and the sensors. The exact wiring depends on the type of sensor we're using. Here's a general guideline for both Ultrasonic and IR sensors:

Ultrasonic Sensor Wiring:

- Connect the Ultrasonic Sensor's VCC (power) to the Raspberry Pi's 5V pin.
- Link the Ultrasonic Sensor's GND (ground) to the Raspberry Pi's GND pin.
- Attach the Ultrasonic Sensor's Echo pin to a GPIO pin on the Raspberry Pi (e.g., GPIO17).
- Connect the Ultrasonic Sensor's Trigger pin to another GPIO pin on the Raspberry Pi (e.g., GPIO18).

IR Sensor Wiring:

- Connect the IR Sensor's VCC (power) to the Raspberry Pi's 5V pin.
- Link the IR Sensor's GND (ground) to the Raspberry Pi's GND pin.
- Attach the IR Sensor's OUT pin to a GPIO pin on the Raspberry Pi (e.g., GPIO17).

we can adjust the pin numbers as needed based on our specific setup and how we configure the sensors in our Python code.

Step 6: Develop Python Code

Next, we had created Python code to simulate parking occupancy detection. We have used Python's GPIO library to interact with the sensors. Here's a basic example of simulating parking occupancy detection using Python with an Ultrasonic sensor.

CODE :

```
import RPi.GPIO as GPIO

import time


# Set up GPIO pins

GPIO.setmode(GPIO.BCM)
```

```
TRIG = 18
ECHO = 17
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# Simulate parking occupancy detection
try:
    while True:
        GPIO.output(TRIG, False)
        time.sleep(2)

        GPIO.output(TRIG, True)
        time.sleep(0.00001)
        GPIO.output(TRIG, False)

        while GPIO.input(ECHO) == 0:
            pulse_start = time.time()

        while GPIO.input(ECHO) == 1:
            pulse_end = time.time()

        pulse_duration = pulse_end - pulse_start
        distance = pulse_duration * 17150
        if distance < 20:
            print("Parking Occupied")
        else:
            print("Parking Available")

except KeyboardInterrupt:
    GPIO.cleanup()
```

Step 7: Run the Simulation

1. To observe the simulation, we clicked the "Run" button in Wokwi.
2. We monitored how our virtual Raspberry Pi and parking sensor behave in real-time. The Python code will simulate parking occupancy detection based on the sensor's behavior.

BENEFITS OF REAL-TIME PARKING AVAILABILITY SYSTEM

USER BENEFITS :

Our Smart Parking system delivers a host of benefits to drivers and the community:

- Time savings for drivers by providing quick access to available parking spots.
- Reduced traffic congestion and its associated environmental benefits.
- Lower fuel consumption and emissions, contributing to a cleaner urban environment.
- Enhanced urban planning by optimizing parking space utilization.

SUBMISSION:

- **GITHUB REPOSITORY:**

  The complete source code, documentation, and associated files for our project are hosted on our GitHub repository

- **REPLICATION AND DEPLOYMENT INSTRUCTIONS:** To replicate and deploy our Smart Transportation Real-time Transit Information System, follow these steps:

  1. **HARDWARE SETUP:**

     Deploy the IoT sensors on public transportation vehicles, including GPS trackers, passenger counters, and environmental sensors.

2. Raspberry Pi Configuration:

Set up Raspberry Pi devices on the vehicles and install the required software for data collection and transmission.

3. Central Server Setup:

Deploy the central server on a cloud platform. Install the server software and configure it to receive and process data from Raspberry Pi devices.

4. Mobile App Development:

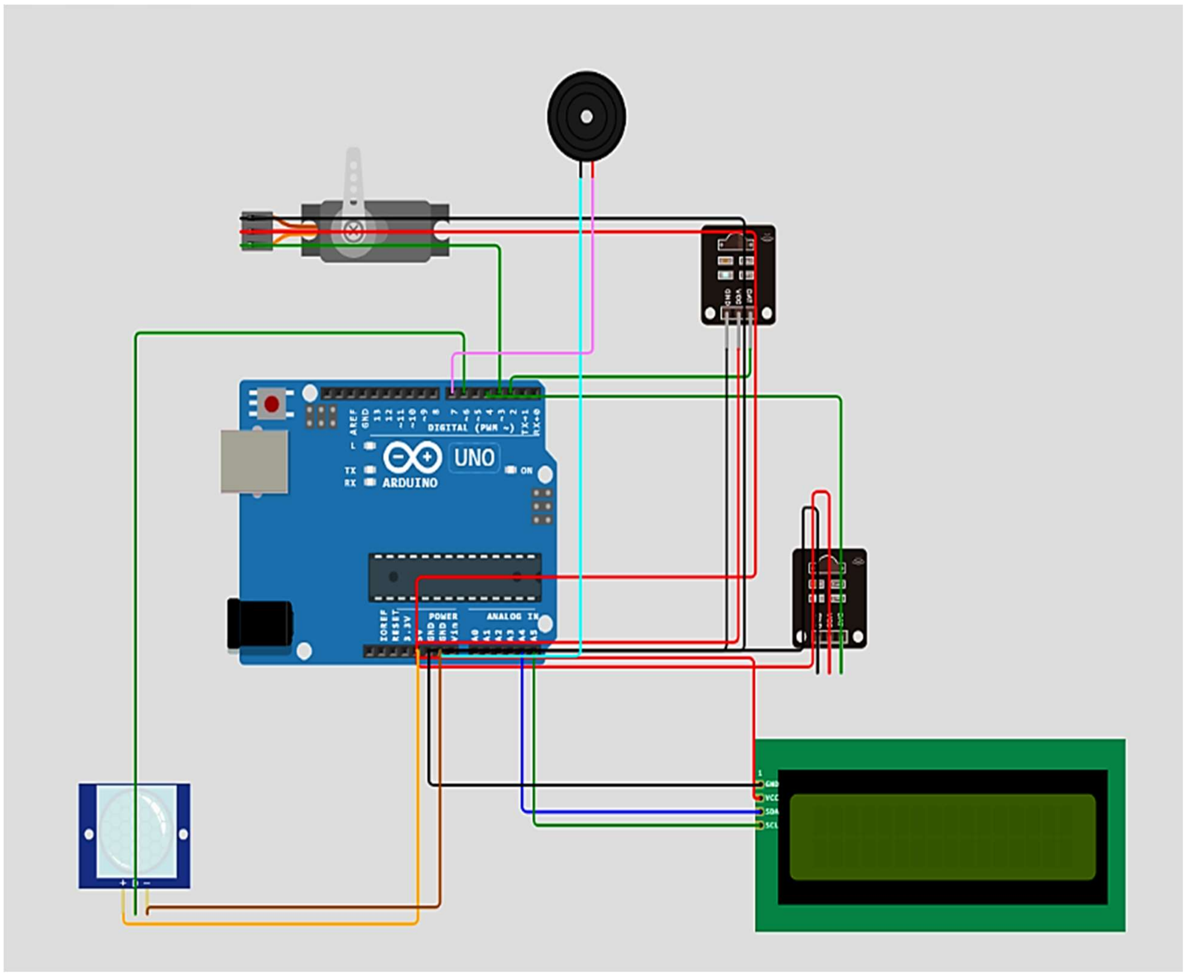
Create a mobile application using the Flutter framework and integrate it with the central server using APIs.

CONCLUSION

Thus, our project “ SMARK PARKING SYSTEM” is an comprehensive solution which aimed at revolutionizing urban parking. Our objective is to address the perennial issue of parking availability in congested areas. This project makes the urban driving experience more efficient and less frustrated by providing real-time parking spot availability information. Thus, it encompassed real-time parking space monitoring, mobile app integration, and the establishment of an efficient parking guidance system, all central components of the broader "Smart Parking" concept.

PROJECT CODE AND SIMULATION OUTPUT

SIMULATION CIRCUIT:



DATA COLLECTION AND SENSOR INTERFACE:

******Data Collection and Sensor Interface******

```
import RPi.GPIO as GPIO
import time

# Set GPIO mode to BCM
GPIO.setmode(GPIO.BCM)

# Define GPIO pins for the sensor
TRIGGER_PIN = 18
ECHO_PIN = 24

# Set up GPIO pins
GPIO.setup(TRIGGER_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

# Function to trigger a distance measurement
def trigger_measurement():
    GPIO.output(TRIGGER_PIN, True)
    time.sleep(0.00001)
    GPIO.output(TRIGGER_PIN, False)
```

```
# Function to measure the distance
def measure_distance():
    pulse_start = time.time()
    pulse_end = time.time()

    while GPIO.input(ECHO_PIN) == 0:
        pulse_start = time.time()

    while GPIO.input(ECHO_PIN) == 1:
        pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    distance = (pulse_duration * 34300) / 2  # Speed of sound is 343
m/s

    return distance
```

****Data Transmission to the Cloud/Mobile App Server****

```
import requests
# Define the server endpoint for data transmission
SERVER_ENDPOINT = "http://example.com/api/parking"
# Main loop for data collection and transmission
try:
    while True:
        # Trigger a distance measurement
        trigger_measurement()

        # Measure the distance
        distance = measure_distance()

        # Send data to the server
        data = {"distance": distance}
        response = requests.post(SERVER_ENDPOINT, json=data)

        if response.status_code == 200:
            print(f"Data sent: Distance - {distance} cm")
        else:
            print("Data transmission failed")
        # Adjust the data collection and transmission interval
        time.sleep(5)
except KeyboardInterrupt:
    print("Data collection stopped by the user")
finally:
    # Clean up GPIO settings
    GPIO.cleanup()
```

WOWKI CODE :

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);  // Change the HEX address

#include <Servo.h>
Servo myservo1;

int IR1 = 2;
int IR2 = 4;
int SmokeDetectorPin = 6;  // Digital pin for the smoke detector
int BuzzerPin = 7;         // Digital pin for the buzzer

int Slot = 4;  // Enter Total number of parking Slots

bool flag1 = false;
bool flag2 = false;
```

```
unsigned long lastLcdUpdate = 0;  // Variable to track the time of the
last LCD update
unsigned long lcdUpdateInterval = 1000;  // Update the LCD every 1000
milliseconds (1 second)

void setup()
{
  lcd.begin(16, 2);  // Initialize LCD with 16 columns and 2 rows
  lcd.backlight();
  pinMode(IR1, INPUT);
  pinMode(IR2, INPUT);
  pinMode(SmokeDetectorPin, INPUT);
  pinMode(BuzzerPin, OUTPUT);

  myservo1.attach(3);
  myservo1.write(100);

  lcd.setCursor(0, 0);
  lcd.print("     ARDUINO     ");
  lcd.setCursor(0, 1);
  lcd.print(" PARKING SYSTEM ");
  delay(2000);
  lcd.clear();

  Serial.begin(9600);  // Start serial communication for debugging
}

void loop()
{
  if (digitalRead(IR1) == LOW && !flag1)
 {
    if (Slot > 0)
    {
      flag1 = true;
      if (!flag2)
      {
        myservo1.write(0);
        Slot--;
      }
    }
   else
   {
      displayMessage("    SORRY :(    ", "  Parking Full  ");
    }
  }

  if (digitalRead(IR2) == LOW && !flag2)
 {
    flag2 = true;
    if (!flag1)
   {
      myservo1.write(0);
      Slot++;
    }
  }

  if (flag1 && flag2)
 {
    delay(1000);
    myservo1.write(100);
    Serial.println("Servo returned to initial position.");
    flag1 = false;
    flag2 = false;
  }

  // Update the LCD display with a delay
  if (millis() - lastLcdUpdate >= lcdUpdateInterval) {
    updateLcdDisplay();
    lastLcdUpdate = millis();
  }

}

void updateLcdDisplay()
{
  if (digitalRead(SmokeDetectorPin) == HIGH)
{
    displayMessage("   WARNING!   ", " Smoke Detected ");
    digitalWrite(BuzzerPin, HIGH);  // Turn on the buzzer
  }
else
{
    displayMessage("    WELCOME!    ", "Slot Left: " + String(Slot));
    digitalWrite(BuzzerPin, LOW);   // Turn off the buzzer
  }
}

void displayMessage(const char *line1, const String &line2)
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(line1);
  lcd.setCursor(0, 1);
  lcd.print(line2);
}
```

DESIGN INTERFACE

CODE :

```
import 'package:flutter/material.dart';
void main() {
  runApp(ParkingAvailabilityApp());
}

class ParkingAvailabilityApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ParkingAvailabilityScreen(),
    );
  }
}

class ParkingAvailabilityScreen extends StatefulWidget {
  @override
```

```
 _ParkingAvailabilityScreenState createState() =>
_ParkingAvailabilityScreenState();
}

class _ParkingAvailabilityScreenState extends
State<ParkingAvailabilityScreen> {
  // State variables and methods to display parking availability data.

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Parking Availability'),
      ),
      body: //  UI to display parking availability data goes here,
    );
  }
}
```

FETCH AND DISPLAY REAL TIME DATA

CODE :

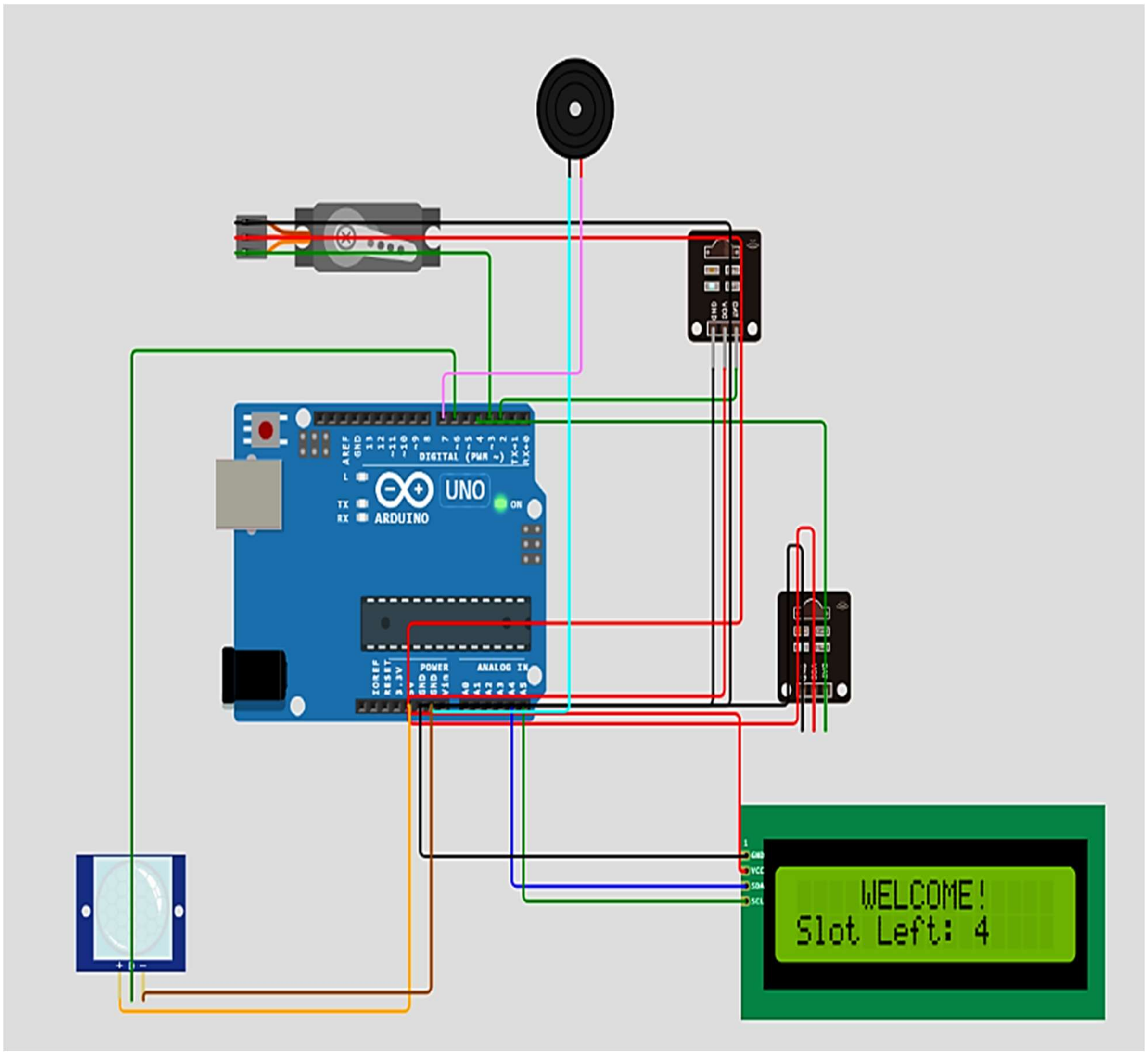
```
import 'package:http/http.dart' as http;
import 'dart:convert';
```

```
Future<Map<String, dynamic>> fetchParkingAvailabilityData() async {
  final response = await http.get('http://your-raspberry-pi-
ip/parking_data');
  if (response.statusCode == 200) {
    return json.decode(response.body);
  } else {
    throw Exception('Failed to load parking availability data');
  }
}
// fetchParkingAvailabilityData to update the UI with parking data.
```

PARKING OCCUPANCY DETECTION USING PYTHON CODE :

```
import RPi.GPIO as GPIO
import time

# Set up GPIO pins
GPIO.setmode(GPIO.BCM)
TRIG = 18
ECHO = 17
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

# Simulate parking occupancy detection
try:
    while True:
        GPIO.output(TRIG, False)
        time.sleep(2)

        GPIO.output(TRIG, True)
        time.sleep(0.00001)
        GPIO.output(TRIG, False)

        while GPIO.input(ECHO) == 0:
            pulse_start = time.time()

        while GPIO.input(ECHO) == 1:
            pulse_end = time.time()

        pulse_duration = pulse_end - pulse_start
        distance = pulse_duration * 17150

        if distance < 20:
            print("Parking Occupied")
        else:
            print("Parking Available")

except KeyboardInterrupt:
    GPIO.cleanup()
```

SIMULATION OUTPUT:


DIGITAL (PWM ~)
UNO
ARDUINO
ON
POWER
ANALOG IN
GND
VCC
SDA
SCL
WELCOME!
Slot Left: 4