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

PROJECT REPORT: SMART PARKING SYSTEM

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.

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,
    );
  }
}
```

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.

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.
```

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.

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.