SMART PARKING SYSTEM

PHASE V

TEAM MEMBERS:

SIVAPRAKASH A (810021106079)

TONY CHACKO THOMAS (810021106089)

SINDHUJA U (810021106076)

SHRIVARSHA P (810021106075)

SRIHARI VV (810021106303)

VEERANANDHA KUMAR (810021106310)

SMART PARKING SYSTEM

INTRODUCTION:

Smart parking system using IoT is a modern innovation that leverages the Internet of Things to enhance parking management. It employs sensors and connectivity to provide real-time data on parking space availability, making it easier for drivers to find spots and for cities to optimize traffic flow and reduce congestion. This technology is revolutionizing the way we park in increasingly crowded urban areas, offering efficiency, convenience, and sustainability.

OVERVIEW:

1. Data Collection: IR sensors installed in parking spaces detect vehicle presence. When a vehicle enters or leaves a spot, the sensors send this information to the ESP8266 module.

2. Data Transmission: The ESP8266 collects data from the IR sensors and transmits it to the central server using Wi-Fi connectivity. The server is responsible for processing this data.

3. Server-Side Logic: The Python code running on the server processes the data received from the ESP8266. It updates the parking space status in real-time, manages reservations, and controls the servo motor to open or close parking gates as needed.

4. Mobile App Interaction: Users can access the mobile app (developed with Flutter) to view the availability of parking spaces. The app communicates with the server, displaying real-time information on parking spot availability.

5. Reservations and Payments: Users can reserve parking spots through the app. If required, they can make payments for their parking using integrated payment gateways.

6. User Convenience: The smart parking system offers a convenient and efficient experience for drivers, helping them save time and reduce stress associated with parking in crowded areas.

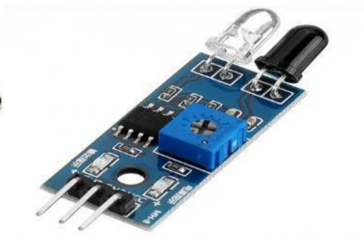
DESIGN FOR SMART PARKING MANAGEMENT:

COMPONENTS REQUIRED:

HARDWARE:

→ Node MCU ESP8266
→ IR Sensor- 5 nos
→ Servo Motor-2nos

ESP8266

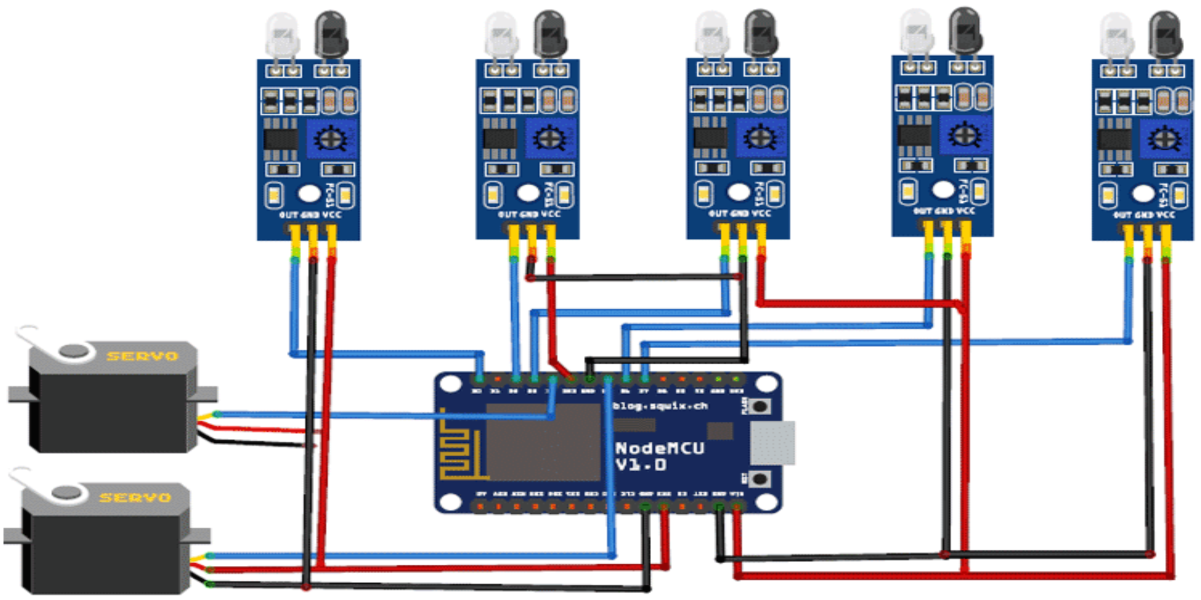

IR sensor

Servo motor

ONLINE SERVICES:

Flutter app

CIRCUIT DIAGRAM:



In this Smart Parking System using IOT, we are using five IR Sensors and two servo motors. IR sensors and Servo motors are connected to the NodeMCU. NodeMCU controls the complete process and sends the parking availability and parking time information to Adafruit IO so that it can be monitored from anywhere in the world using this platform. Two IR sensors are used at entry and exit gate so that it can detect the cars at entry and exit gate and automatically open and close the gate. Two servo motors are used as entry and exit gate, so whenever the IR sensor detects a car, the servo motor automatically rotates from 45° to 140°, and after a delay, it will return to its initial position. Another three IR sensors are used to detect if the parking slot is available or occupied and send the data to NodeMCU. Adafruit IO dashboard also has two buttons to manually operate the entry and exit gate.

ARDUINO CODE FOR ESP 8266:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
const char *ssid = "YOUR_SSID";
const char *password = "YOUR_PASSWORD";
ESP8266WebServer server(80);
int irPin1 = D1;  // Define IR Sensor Pins
int irPin2 = D2;
int servoPin = D3; // Define Servo Motor Pin
Servo gateServo; // Create a servo object
void setup() {
  Serial.begin(115200);
  pinMode(irPin1, INPUT);
  pinMode(irPin2, INPUT);
  gateServo.attach(servoPin);
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  // Route for root / web page
  server.on("/", HTTP_GET, handleRoot);
  server.begin();
}
void loop() {
  server.handleClient();
}
void handleRoot() {int sensor1Value = digitalRead(irPin1);
  int sensor2Value = digitalRead(irPin2);
  // Check if parking spots are occupied
  String status = "Spot 1: " + String(sensor1Value) + ", Spot 2: " +
String(sensor2Value);
  if (sensor1Value == HIGH && sensor2Value == HIGH) {
    // Both spots are occupied, close the gate
    gateServo.write(0); // Assuming 0 degrees is the closed position
    status += " - Gate Closed";
  } else {
    // At least one spot is available, open the gate
    gateServo.write(90); // Assuming 90 degrees is the open position
    status += " - Gate Open";
  }
  server.send(200, "text/plain", status);
}
```

WORKING OF THE CODE:

The provided code is an Arduino sketch for an ESP8266-based IoT project that controls a servo motor and reads data from two IR sensors to manage a parking gate. Here's how the code works:

1. It includes necessary libraries for Wi-Fi connectivity (ESP8266WiFi), handling web server functions (ESP8266WebServer), and controlling the servo motor.

2. The code defines the Wi-Fi credentials (SSID and password) to connect the ESP8266 to your Wi-Fi network.

3. It initializes an ESP8266WebServer object on port 80, which will handle incoming HTTP requests.

4. The code defines the GPIO pins for two IR sensors (irPin1 and irPin2) and a servo motor (servoPin).

5. A Servo object called gateServo is created to control the servo motor.

6. In the setup() function:

   - Serial communication is initiated for debugging.

   - The IR sensor pins are set as inputs.

   - The Servo object is attached to the specified pin.

   - The ESP8266 connects to the specified Wi-Fi network, and the code waits until the connection is established.

   - A root ("/") route is defined, associated with the handleRoot function.

   - The web server is started.

7. In the loop() function, the code continuously handles incoming client requests through the web server.

8. The handleRoot() function is called when a client (e.g., a web browser or a mobile app) accesses the root ("/") route.

   - It reads the digital values of the two IR sensors (sensor1Value and sensor2Value).

   - It checks if both parking spots are occupied by comparing the sensor values.

   - If both spots are occupied, the servo motor is set to the closed position (0 degrees), and the status message is updated to indicate the gate is closed.

   - If at least one spot is available, the servo motor is set to the open position (90 degrees), and the status message reflects that the gate is open.

   - The status message is sent as a response to the client's request with a 200 OK status code.

The code, as provided, creates a simple web server that you can access to check the status of the parking spaces and control the gate. When both parking spots are occupied, the gate is closed, and when at least one spot is available, the gate is open.

To use this code, make sure to replace "YOUR_SSID" and "YOUR_PASSWORD" with your Wi-Fi network credentials. You should also have the necessary hardware connections, including the IR sensors and servo motor, properly set up with the ESP8266. Additionally, you may need to install the required libraries if they are not already present in your Arduino IDE.

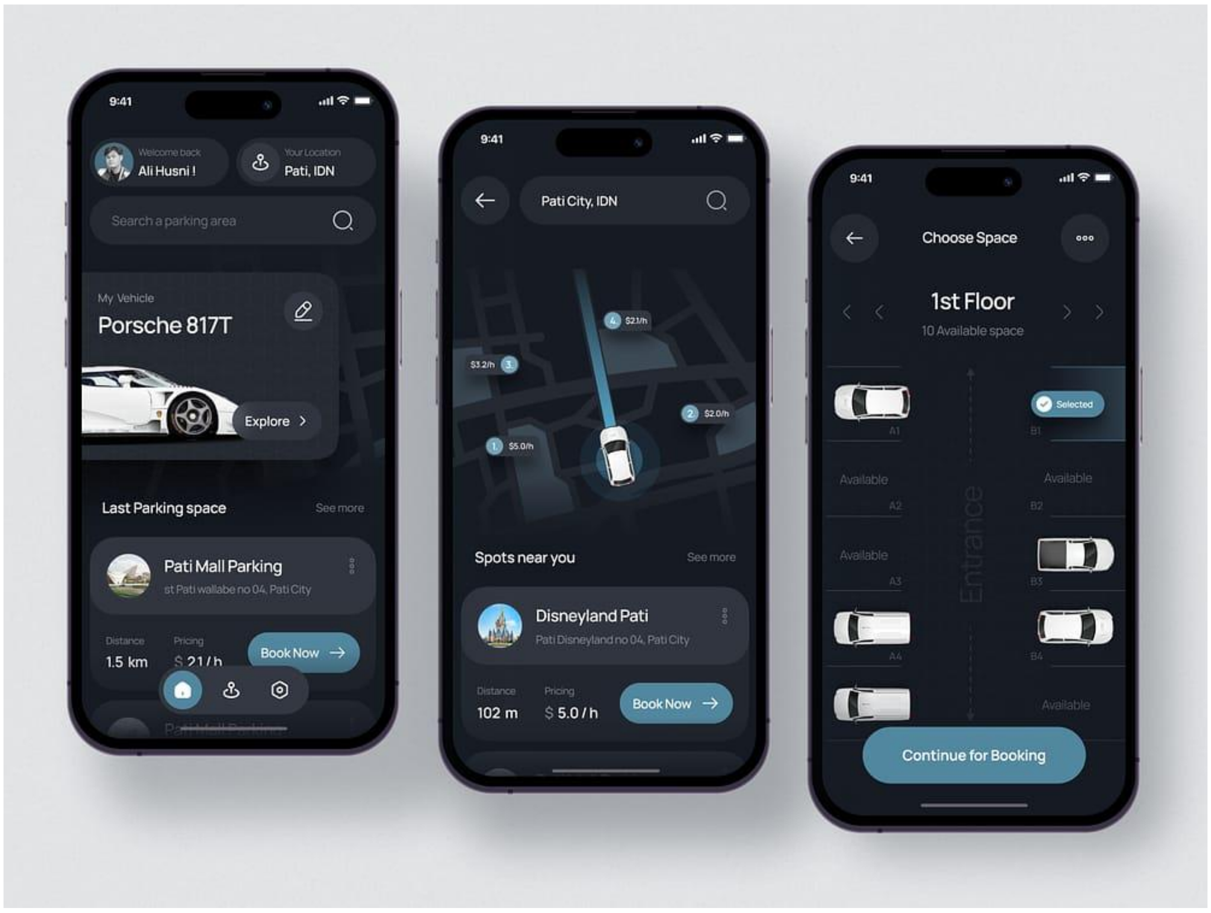
FLUTTER APP:

Flutter is a popular open-source framework developed by Google for building mobile, web, and desktop applications with a single codebase. It's known for its flexibility and performance. Flutter uses the Dart programming language and a widget-based approach for creating user interfaces. Developers appreciate its "hot reload" feature, enabling real-time code changes and efficient debugging. The platform's rich ecosystem includes numerous packages and plugins contributed by the community, enhancing its capabilities. Flutter compiles to native code for impressive performance and seamless user experiences. It has a growing community, offers adaptive design, and is open source, making it a compelling choice for cross-platform app development.

CODE:

```
import 'package:flutter/material.dart';
void main() {
  runApp(SmartParkingApp());
}
class SmartParkingApp extends StatefulWidget {
  @override
  _SmartParkingAppState createState() => _SmartParkingAppState();
}
class _SmartParkingAppState extends State<SmartParkingApp> {
  int parkingAvailability = 0;
  // Function to simulate data received from IoT devices.
  void updateParkingAvailability() {
    // Replace this logic with data received from server motors, IR sensors, or ESP8266.
    setState(() {
      parkingAvailability = (0 + (3 * 1)).toInt();
    });
  }
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
        title: Text('Smart Parking Using IoT'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text(
                'Parking Availability:',
                style: TextStyle(fontSize: 20),
              ),
              Text(
                '$parkingAvailability', // Display parking availability here.
                style: TextStyle(fontSize: 50, fontWeight: FontWeight.bold),
              ),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                  // Call the function to update parking availability.
                  updateParkingAvailability();
                },
                child: Text ('Refresh Availability'),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

OUTPUT:



WORKING OF THE CODE:

1.Import Statements: The code starts with an import statement, which includes the necessary Flutter libraries for creating the app's user interface.

2.Main Function: `void main () ` is the entry point for the Flutter application. It calls `runApp()` to start the app and specifies the root widget, which is `SmartParkingApp()` in this case.

3.SmartParkingApp Class: `SmartParkingApp` is a custom widget that extends `StatefulWidget`. This is a key concept in Flutter. `StatefulWidget` widgets have mutable states that can change during the lifetime of the widget.

4._SmartParkingAppState Class: `_SmartParkingAppState` is the state class associated with the `SmartParkingApp` widget. It manages the internal state of the widget, including the `parkingAvailability` variable.

5.parkingAvailability Variable: The `parkingAvailability` variable is an integer that represents the availability of parking spaces. It's initially set to 0.

6.updateParkingAvailability Function: This function is used to simulate data received from IoT devices. In the provided code, it uses a simple mathematical operation to update the `parkingAvailability`. You should replace this logic with real data retrieval from IoT components like server motors, IR sensors, or ESP8266.

7.Widget Build Function: The `build` method is where the widget's UI is defined. It returns a `MaterialApp` widget, which is the root of the app's widget tree.

8.AppBar: The `AppBar` is a navigation bar at the top of the screen with a title of 'Smart Parking Using IoT'.

9.Scaffold: The `Scaffold` widget provides a basic skeletal structure for the app, including the app bar and body.

10.Center Widget: The `Center` widget centers its child in both the horizontal and vertical directions. Inside the `Center` widget, there's a `Column` widget that arranges its children in a vertical column.

11.Text Widgets: Two `Text` widgets are used to display text on the screen. The first one displays 'Parking Availability' as a label, and the second one displays the value of the `parkingAvailability` variable.

12.ElevatedButton: This is a button widget that allows users to refresh the parking availability. When pressed, it calls the `updateParkingAvailability` function to update the `parkingAvailability` value.

This code sets up a Flutter app with a basic user interface. It simulates parking availability data, but you should replace the simulation with actual data retrieval from IoT components. When the "Refresh Availability" button is pressed, it calls a function to update the parking availability value and displays it on the screen. The UI updates automatically thanks to Flutter's reactive framework, making it easy to see changes in real-time during development.

HOW SMART PARKING USING IOT HELPS IN SOLVING REAL LIFE PARKING PROBLEMS:

A real-time parking availability system using IoT (Internet of Things) technology can offer numerous benefits to drivers and help alleviate parking issues. Here's how:

1. Reduced Search Time: The primary advantage of a real-time parking availability system is that it helps drivers find parking spaces more quickly. With the app interface, drivers can check the availability of parking spots in real-time, saving them the time and frustration of driving around in search of a parking space.

2. Cost Savings: Drivers can save on fuel costs and reduce emissions by spending less time circling for parking. This is not only environmentally friendly but also economically beneficial.

3. Convenience: The IoT app interface provides convenience by allowing drivers to reserve or pay for parking in advance. This minimizes the need for cash payments or physical tickets, making the entire parking process smoother and more user-friendly.

4. Reduced Traffic Congestion: By decreasing the amount of time spent looking for parking, traffic congestion can be reduced. Less congestion means less stress for drivers and a more efficient road network for everyone.

5. Better Resource Management: The system can help parking facility operators better manage their resources. They can optimize parking space allocation, pricing, and maintenance based on real-time data and historical patterns.

6. Accessibility: Some IoT parking systems can include features for disabled or special needs drivers, providing them with information on accessible parking spaces and making parking areas more inclusive.

7. Environmental Benefits: As mentioned earlier, reduced traffic congestion leads to lower emissions, which is beneficial for the environment and public health.

8. Safety: In some systems, parking areas can have security features integrated with IoT technology, such as cameras and emergency call buttons. This enhances the safety of the parking experience for users.

9. User Experience: Users can receive directions to available parking spaces, view ratings and reviews of parking facilities, and even get information on nearby amenities, making the overall parking experience more pleasant.

10. Revenue Generation: For parking facility owners, the IoT app interface can generate additional revenue through features like dynamic pricing, advertising, or partnerships with local businesses.

11. Data Analytics: The system can collect valuable data on parking patterns, which can be used to plan future urban development and improve traffic flow in the area.

CONCLUSION:

In conclusion, the integration of app development using Python for the "Smart Parking System Using IoT" is poised to revolutionize urban mobility and parking management. The ever-growing demand for efficient parking solutions in today's urban landscapes necessitates innovative approaches like Smart Parking driven by the Internet of Things (IoT). This introduction has highlighted the critical role of mobile applications developed in Python, a versatile and powerful programming language, in this transformative process.

The code presented demonstrates a practical embodiment of this concept using Flutter, a cross-platform framework. The Flutter app, while still in the early stages of development, showcases the potential to provide real-time parking availability data to drivers. It's important to note that the code is currently a simulation, but it acts as a stepping stone for what can be achieved when IoT components like server motors, IR sensors, and ESP8266 devices are integrated to deliver accurate parking information.

Ultimately, the marriage of Python-powered mobile applications and IoT technology in Smart Parking endeavors promises more than just convenience for drivers. It represents an opportunity to enhance urban mobility, alleviate traffic congestion, and contribute to a sustainable, interconnected urban future. As the development of Smart Parking solutions continues to advance, it is certain that Python and IoT will play increasingly pivotal roles in shaping the smart cities of tomorrow.