SMART PARKING SYSTEM USING IOT

PHASE IV

TEAM MEMBERS:

SIVAPRAKASH A (810021106079)

TONY CHACKO THOMAS (810021106089)

SINDHUJA U (810021106076)

SHRIVARSHA P (810021106075)

SRIHARI VV (810021106303)

VEERANANDHA KUMAR (810021106310)

SMART PARKING MANAGEMENT USING IOT

INTRODUCTION:

In today's urban landscape, the demand for efficient parking solutions has never been greater. Smart Parking, driven by the Internet of Things (IoT), represents a transformative approach to tackle the challenges of parking in crowded cities. At the heart of this innovative solution lies the development of mobile applications using Python, a versatile and powerful programming language. These applications serve as the digital gateway to a complex network of IoT devices, such as sensors, server motors, and ESP8266 modules, which work in harmony to provide real-time parking availability information. In this introduction, we'll explore the fundamental principles and the pivotal role that Python plays in developing applications that empower drivers to find, reserve, and navigate to the nearest available parking spaces. Smart Parking using Python and IoT is not just about simplifying parking but also about enhancing urban mobility, reducing congestion, and contributing to a more sustainable and connected future.

DESIGN FOR SMART PARKING MANAGEMENT:

COMPONENTS REQUIRED:
HARDWARE:

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

ESP8266

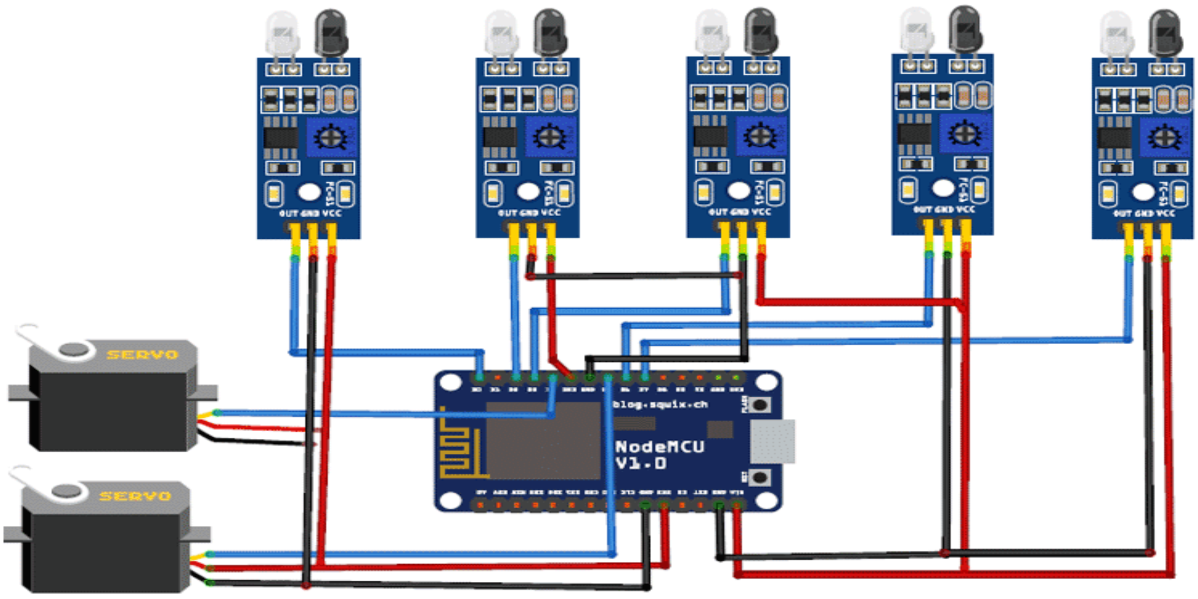
IR sensor

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.

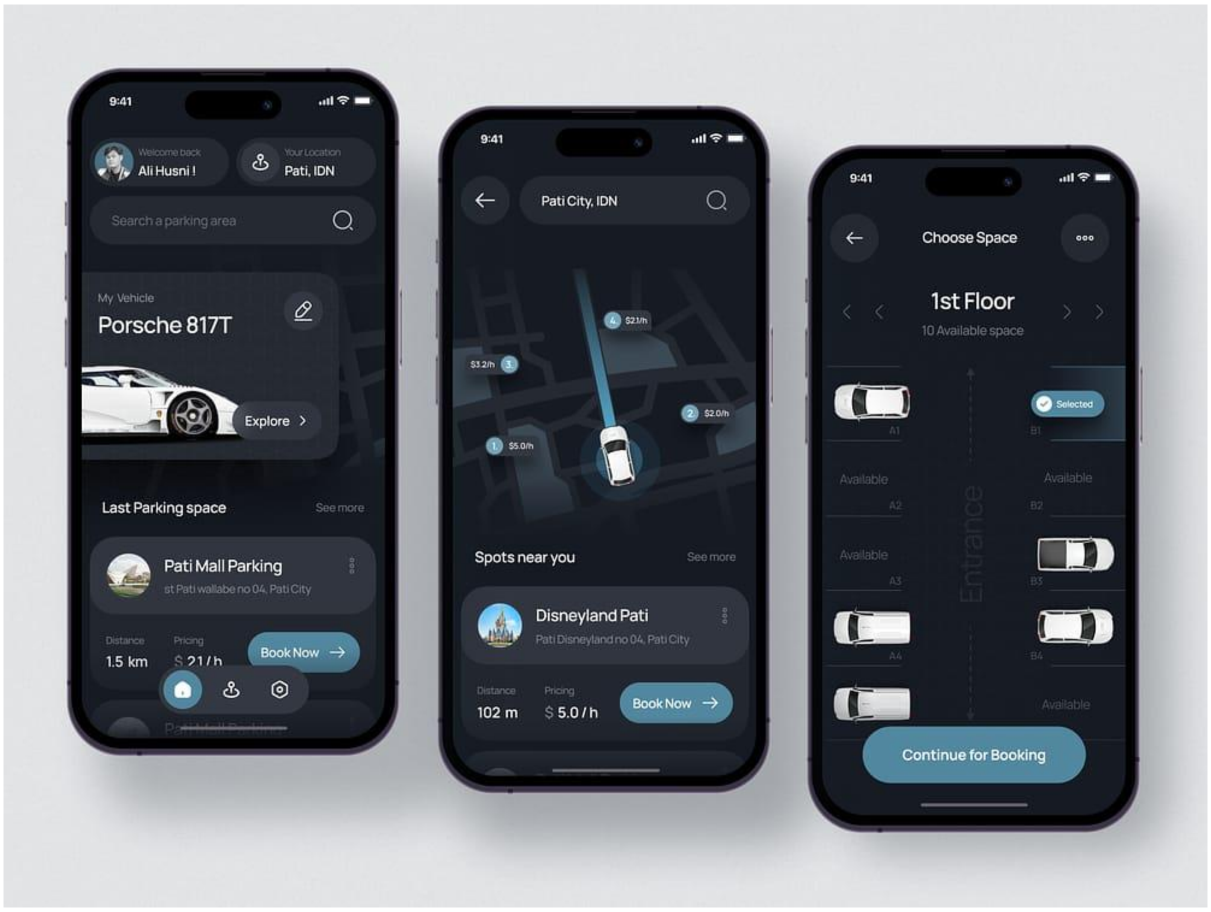
FLUTTER APP:

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.

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.