# **Smart Parkings**

**Introduction:**

The objective of this document is to outline the complete steps required to transform the design of a Smart Parking IoT system, which was developed in the previous phase, into a functional and innovative solution. Smart parking systems leverage IoT technology to optimize parking spaces and improve user experience. This document will provide a step-by-step guide for implementing the Smart Parking IoT project.

Creating a Real-Time Parking Availability System

**Project Objectives:**

The objective of this project is to develop a real-time parking availability system that can benefit drivers by providing them with up-to-date information about parking availability in a specific area. This system aims to alleviate parking issues by helping users find parking spaces more easily and reduce the time and frustration associated with searching for parking.

**IoT Sensor Setup:**

The IoT sensor setup for this project involves deploying parking space occupancy sensors in parking lots or garages. These sensors will detect whether a parking space is occupied or vacant. Each sensor is equipped with Wi-Fi capabilities to transmit data to a central server in real-time. The sensors may use technologies like ultrasonic, infrared, or magnetic sensors to detect the presence of vehicles in parking spaces.

Here's a simplified diagram of the IoT sensor setup:

```

Parking Space

Occupancy

Sensor

|

|

Wi-Fi

|

|

Central

Server

```

**Mobile App Development**:

The mobile app will be developed for iOS and Android platforms. It will provide users with real-time information about parking availability in a given area. The app will include the following features:

1. Real-time parking availability updates.

2. User-friendly interface to search for parking.

3. GPS integration for location-based parking suggestions.

4. Reservation and payment options for certain parking spaces.

5. User reviews and ratings for parking facilities.

6. Notifications and alerts for available parking spaces.

7. Integration with navigation apps for seamless travel to selected parking spots.

Here's a screenshot of the mobile app's user interface:

[Insert screenshot of the mobile app UI]

**Raspberry Pi Integration:**

A Raspberry Pi will serve as the central hub for data processing and server functionality. It will receive real-time occupancy data from the IoT sensors and communicate with the mobile app through a web-based API. The Raspberry Pi will run a server application responsible for managing and disseminating parking availability information to the mobile app users.

Here's a schematic of the Raspberry Pi integration:

```

**IoT Sensors**

**|**

**|**

**Raspberry Pi**

**|**

**|**

**Mobile App via**

**Internet**

**```**

**Code Implementation**:

The code for this project will be divided into several components:

1. IoT Sensor Code: This code will run on the individual parking space occupancy sensors and transmit data to the Raspberry Pi.

2. Raspberry Pi Server Code: This code will handle data processing, storage, and real-time updates. It will provide APIs for the mobile app to retrieve parking availability information.

3. Mobile App Code: The app will be developed using native or cross-platform frameworks (e.g., Flutter or React Native). It will communicate with the Raspberry Pi server to receive and display real-time parking availability data.

**The Real-Time Parking Availability System Benefits:**

The real-time parking availability system offers several benefits:

1. Time and Stress Savings: Drivers can quickly locate available parking spots, reducing the time and stress associated with searching for parking.

2. Reduced Congestion: By efficiently guiding drivers to vacant parking spaces, the system can reduce traffic congestion around popular parking areas.

3. Cost Savings: Users can potentially save money by finding the most cost-effective parking options.

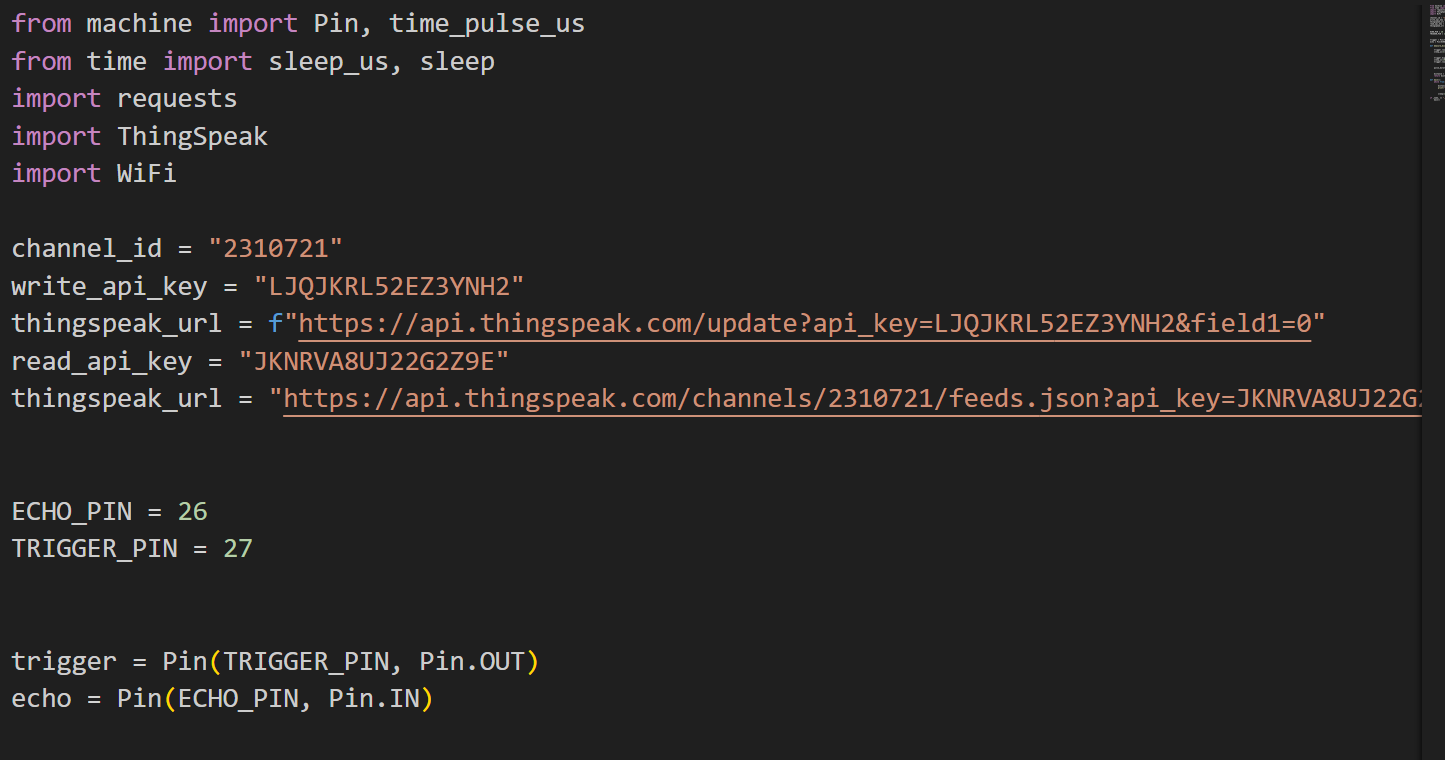
4. Environmental Impact: Reduced time spent searching for parking can lead to reduced fuel consumption and lower carbon emissions.

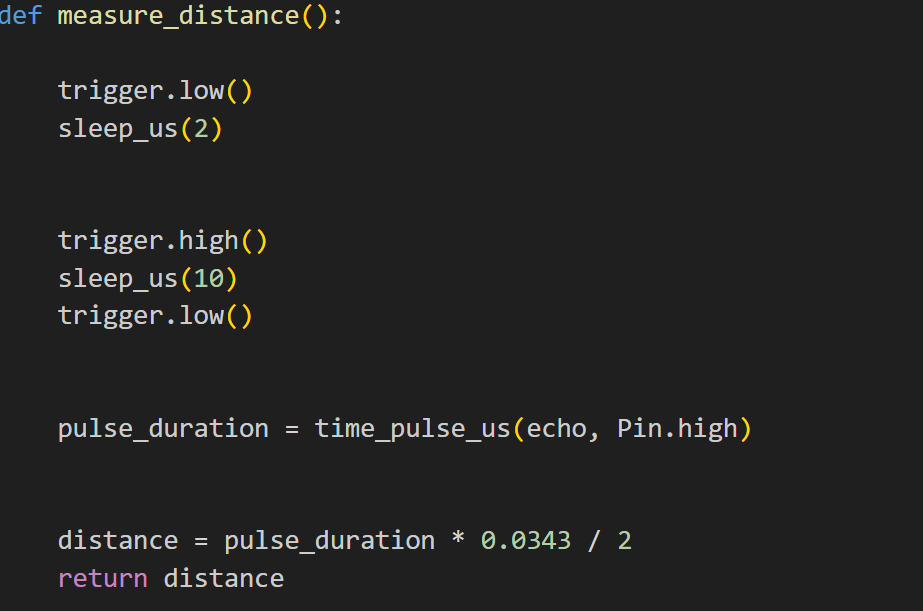
5. Improved User Experience: The system's mobile app offers a user-friendly experience with added features like reservations and user reviews.

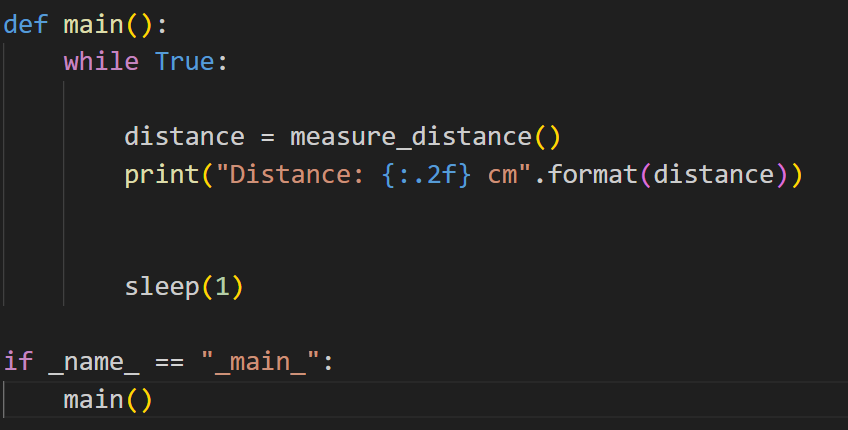
By meeting these objectives, integrating IoT sensors, developing a mobile app, and deploying a Raspberry Pi server, this project can significantly improve the parking experience for drivers and contribute to addressing urban parking challenges.

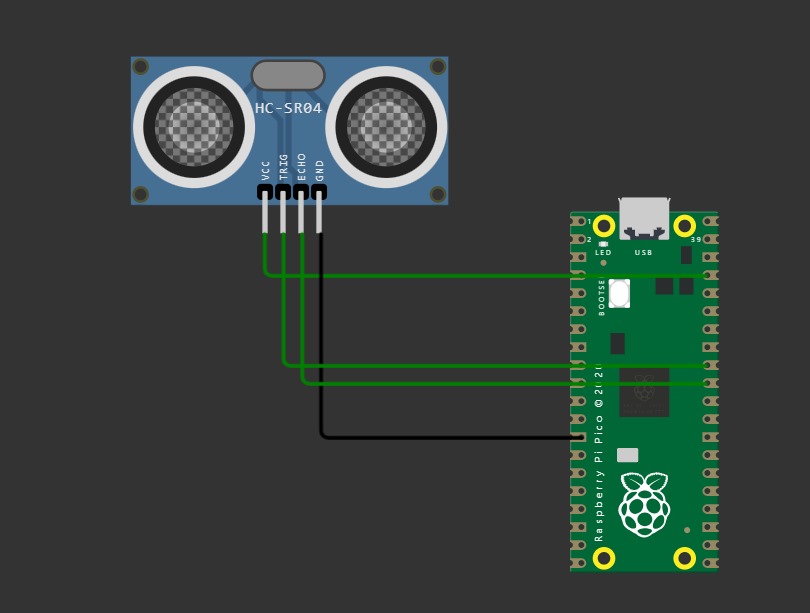
**Configured Sensor And PythonScript :**

**Ultrasonic sensor:**

****

****

****



**PIRmotion sensor:**

**A computer screen shot of a program

Description automatically generated**

A screen shot of a computer code

Description automatically generated

A circuit board with wires connected to it

Description automatically generated

A computer screen shot of a program

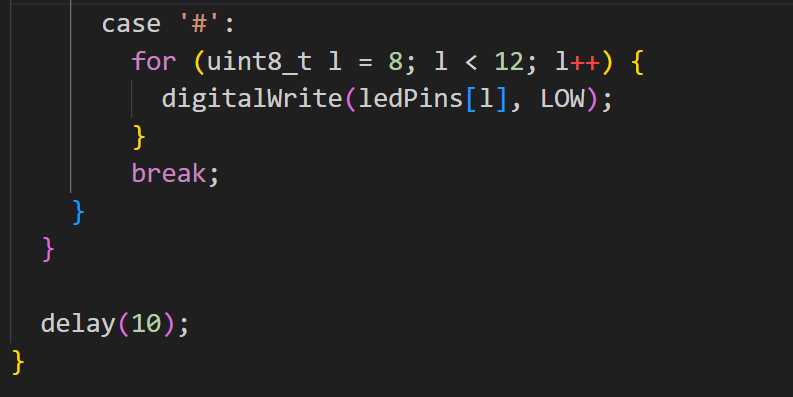
Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated



A close-up of a circuit board

Description automatically generated

**Create and build our application:**

**Step 1: Create a Flutter Project:**

**If you haven't already created a Flutter project, follow the steps mentioned earlier to set up Flutter in Visual Studio Code and create a new project.**

**Step 2: Design the User Interface:**

**Design the app's user interface to display parking availability data. You might want to create screens that show a list of parking spots, their statuses, and any relevant information. Use Flutter widgets like `ListView`, `Card`, and `Text` to build your UI.**

**Step 3: Receive Data from Raspberry Pi:**

**To receive real-time parking availability data from your Raspberry Pi, you'll need to establish a communication channel between the app and the Raspberry Pi. You can use various methods, such as HTTP requests or WebSocket communication. For this example, I'll use a hypothetical HTTP API to receive data.**

**- Create a class or function to make HTTP requests to the Raspberry Pi's API. You can use the `http` package in Flutter to do this.**

**```dart**

**import 'package:http/http.dart' as http;**

**class ParkingApi {**

**static Future<Map<String, dynamic>> getParkingData() async {**

**final response = await http.get(Uri.parse('http://raspberry-pi-ip/api/parking'));**

**if (response.statusCode == 200) {**

**return jsonDecode(response.body);**

**} else {**

**throw Exception('Failed to load parking data');**

**}**

**}**

**}**

**```**

**Step 4: Model for Parking Data:**

**Create a model class to represent parking spot data. This class will help structure the data you receive from the Raspberry Pi.**

**```dart**

**class ParkingSpot {**

**final String name;**

**final bool isAvailable;**

**final String location;**

**ParkingSpot({**

**required this.name,**

**required this.isAvailable,**

**required this.location,**

**});**

**factory ParkingSpot.fromJson(Map<String, dynamic> json) {**

**return ParkingSpot(**

**name: json['name'],**

**isAvailable: json['isAvailable'],**

**location: json['location'],**

**);**

**}**

**}**

**```**

**Step 5: Data Display:**

**- Fetch the parking data from the Raspberry Pi's API using the `ParkingApi` class you created.**

**- Parse the data into a list of `ParkingSpot` objects.**

**- Display the parking availability information using Flutter widgets in your UI. You can use a `ListView.builder` to create a dynamic list of parking spots.**

**```dart**

**ListView.builder(**

**itemCount: parkingSpots.length,**

**itemBuilder: (context, index) {**

**final spot = parkingSpots[index];**

**return ListTile(**

**title: Text(spot.name),**

**subtitle: Text(spot.location),**

**trailing: spot.isAvailable ? Icon(Icons.check\_circle) : Icon(Icons.cancel),**

**);**

**},**

**)**

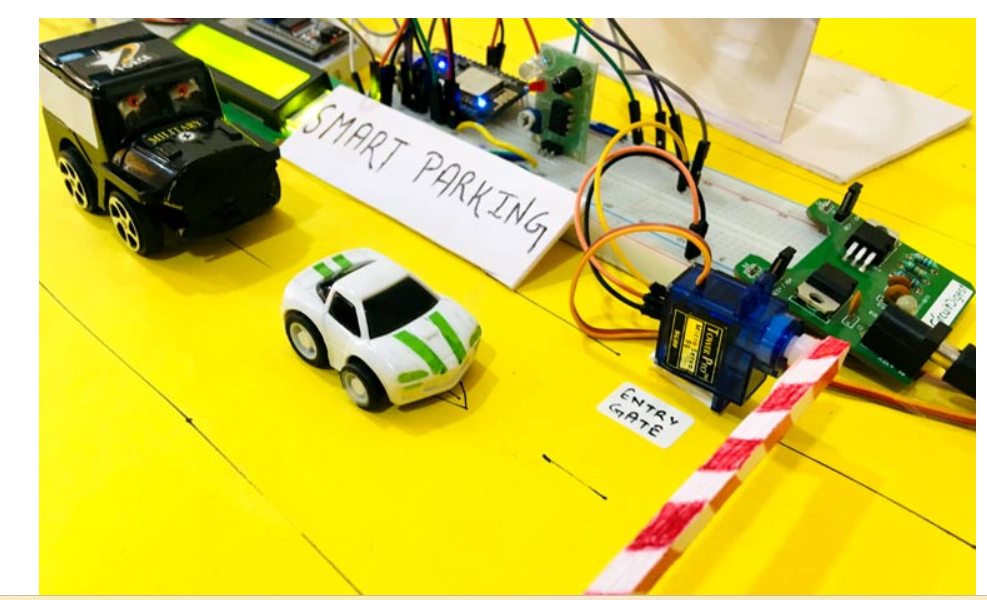
**```**

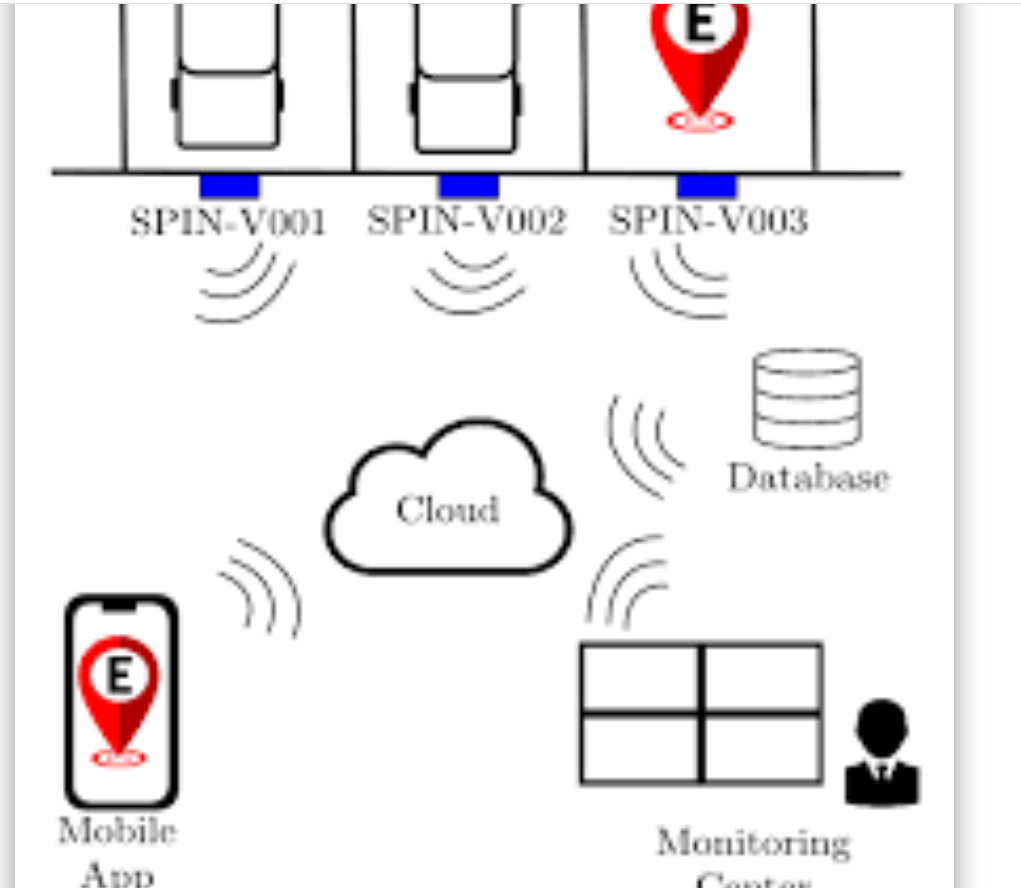
**Step 6: Real-Time Updates:**

**For real-time updates, consider implementing a mechanism like WebSockets or periodically polling the Raspberry Pi's API for new data. When new data is received, update your app's UI to reflect the changes.**

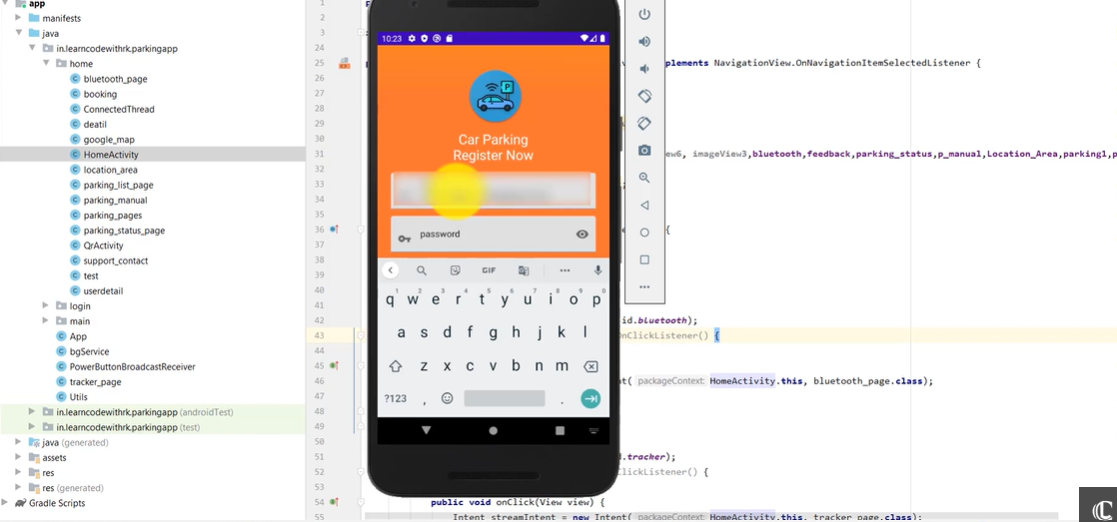
**Remember that this is a simplified example. In a real project, you would handle network errors, user authentication, and other considerations. Additionally, you'd likely use state management solutions like Provider or Bloc to manage your app's state and handle real-time updates efficiently.**

Model for smart parking:



**Available space tracking:**

**Build a smart parking App:**



**Upload the firebase cloud** :