**SMART PARKING**

**Aim:**

Using a mobile app development framework to create an app that displays realtime parking availability. To Design app functions to receive and display parking availability data received from theRaspberry Pi.

**Implementation :**

1.Set up the development environment:

Make sure you have Python installed on your system. For Flutter, you'll need to install the Flutter SDK, Dart SDK, and a code editor such as Visual Studio Code or Android Studio.

2. Create a new Flutter project:

Run `flutter create parking\_availability\_app` in your terminal to create a new Flutter project named `parking\_availability\_app`.

3. Set up the user interface:

Open the `lib/main.dart` file in your project directory and start building the user interface of your app. You may want to create a simple layout that displays the parking availability data in a list or a grid.

```dart

import 'package:flutter/material.dart';

void main() {

runApp(ParkingAvailabilityApp());

}

class ParkingAvailabilityApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Parking Availability',

home: ParkingAvailabilityScreen(),

);

}

}

class ParkingAvailabilityScreen extends StatefulWidget {

@override

\_ParkingAvailabilityScreenState createState() => \_ParkingAvailabilityScreenState();

}

class \_ParkingAvailabilityScreenState extends State<ParkingAvailabilityScreen> {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Parking Availability'),

),

body: // Add your UI components here,

);

}

}

```

4. Receive data from Raspberry Pi:

You'll need to set up a communication channel between the mobile app and the Raspberry Pi. You can use HTTP, WebSocket, MQTT, or any other communication protocol that suits your needs. For example, you can use the `http` package in Flutter to send HTTP requests and receive responses from the Raspberry Pi.

First, add the `http` package to your `pubspec.yaml` file:

```yaml

dependencies:

http: ^0.13.3

```

Then, import the package and create a function to fetch parking availability data from the Raspberry Pi:

```dart

import 'dart:convert';

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

Future<List<ParkingSlot>> fetchParkingAvailability() async {

final response = await http.get(Uri.parse('http://raspberrypi.local/parking\_availability'));

if (response.statusCode == 200) {

final List<dynamic> data = jsonDecode(response.body);

return data.map((e) => ParkingSlot.fromJson(e)).toList();

} else {

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

}

}

class ParkingSlot {

final int id;

final bool isAvailable;

ParkingSlot({required this.id, required this.isAvailable});

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

return ParkingSlot(

id: json['id'],

isAvailable: json['isAvailable'],

);

}

}

```

5. Display parking availability data:

Update the `ParkingAvailabilityScreen` widget to display the parking availability data received from the Raspberry Pi. You can use a `FutureBuilder` widget to handle the asynchronous nature of the data fetching process.

```dart

class \_ParkingAvailabilityScreenState extends State<ParkingAvailabilityScreen> {

late Future<List<ParkingSlot>> \_futureParkingAvailability;

@override

void initState() {

super.initState();

\_futureParkingAvailability = fetchParkingAvailability();

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Parking Availability'),

),

body: FutureBuilder<List<ParkingSlot>>(

future: \_futureParkingAvailability,

builder: (context, snapshot) {

if (snapshot.hasData) {

return ListView.builder(

itemCount: snapshot.data!.length,

itemBuilder: (context, index) {

final slot = snapshot.data![index];

return ListTile(

title: Text('Slot ${slot.id}'),

trailing: Icon(slot.isAvailable ? Icons.check : Icons.close),

);

},

);

} else if (snapshot.hasError) {

return Text('Error: ${snapshot.error}');

}

return CircularProgressIndicator();

},

),

);

}

}

```

6. Test the app:

Run your app on an emulator or a physical device to test the functionality. Make sure the Raspberry Pi is properly configured to serve the parking availability data, and the mobile app can successfully communicate with the Raspberry Pi and display the data.

7. Deploy the app:

Once you've tested the app and are satisfied with its functionality, you can build and publish the app to the Apple App Store and/or Google Play Store. Follow the Flutter documentation for the platform-specific deployment steps.