BUILDING SMART PARKING APP USING PYTHON AND FLUTTER

Step 1: Set Up Your Development Environment

Before you start, ensure you have Flutter installed and set up. You can follow the official Flutter documentation to install it on your machine.

Step 2: Create a New Flutter Project

Run the following commands in your terminal to create a new Flutter project:

```
flutter create smart_parking_app
cd smart_parking_app
```

Step 3: Design the User Interface

You'll need to design the user interface for your app. For a simple implementation, you can create a main screen that displays the parking availability data. You can use Flutter's widgets to design your UI. Here's a basic example of how your app's UI might look:

```
import 'package:flutter/material.dart';

void main() {
  runApp(SmartParkingApp());
}

class SmartParkingApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ParkingAvailabilityScreen(),
```

```
    );
  }
}

class ParkingAvailabilityScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // Replace this with the real-time data received from the Raspberry Pi.
    String parkingStatus = "Available";

    return Scaffold(
      appBar: AppBar(
        title: Text('Smart Parking'),
      ),
      body: Center(
        child: Text(
          'Parking Status: $parkingStatus',
          style: TextStyle(fontSize: 24),
        ),
      ),
    );
  }
}
```

Step 4: Receive Data from Raspberry Pi

To receive real-time parking availability data from a Raspberry Pi, you can use a communication protocol such as WebSocket or HTTP. Here's an example of how to make an HTTP request to fetch parking availability data:

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

Future<String> fetchParkingAvailability() async {
  final response = await http.get('http://raspberry-pi-ip/parking-status');

  if (response.statusCode == 200) {
    return response.body;
  } else {
    throw Exception('Failed to fetch parking availability');
  }
}
```

Step 5: Update the UI with Real-time Data

You can update the parking availability status in your app by making periodic requests to the Raspberry Pi and updating the UI accordingly. You can use a `FutureBuilder` widget to handle asynchronous data fetching and UI updates:

```
FutureBuilder<String>(
  future: fetchParkingAvailability(),
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return CircularProgressIndicator();
    } else if (snapshot.hasError) {
      return Text('Error: ${snapshot.error}');
    } else {
      String parkingStatus = snapshot.data;
      return Text(
        'Parking Status: $parkingStatus',
        style: TextStyle(fontSize: 24),
      );
```

```
    }
  }
)
```

Step 6: Build and Deploy

Once your app is ready, you can use Flutter's build and deploy tools to create APK or IPA files for Android and iOS respectively. Deploy the app to your target devices or app stores as needed.

Remember to replace the sample URLs and data with the actual endpoints and data your Raspberry Pi will provide.

This is a basic outline for building a smart parking app using Python and Flutter. Depending on your specific requirements, you may need to implement additional features like user authentication, maps integration, and notifications.

SAMPLE FROM APPLICATION