Project Documentation: Smart Parking using IoT

Jeya Krishnan
Artificial Intelligence & Data Science
Kings Engineering College
(Affiliated to Anna University)
Chennai, India

Dhivyan E
Artificial Intelligence & Data Science
Kings Engineering College
(Affiliated to Anna University)
Chennai, India

Ajith Kumar P
Artificial Intelligence & Data Science
Kings Engineering College
(Affiliated to Anna University)
Chennai, India

Mohammed Musthaba Majith A
Artificial Intelligence & Data Science
Kings Engineering College
(Affiliated to Anna University)
Chennai, India

Building the project by developing the mobile app using Python

Developing a mobile app for smart parking using Python typically involves using a framework like Kivy or Pygame to create the user interface, integrating backend logic for parking space availability and user interactions, and incorporating features like maps and payment processing. Here, I'll provide you with a basic outline to create a smart parking mobile app using the Kivy framework, which is popular for building cross-platform mobile applications in Python.

Install Kivy:
First, make sure you have Python installed on your system. You can then install Kivy using pip:

```
pip install kivy
```

Design the User Interface:
Create a file named `main.py` for your Python script. Define the user interface in a separate Kivy file (`main.kv`). For example:

```python
# main.py
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout

class SmartParkingApp(App):
    def build(self):
        return BoxLayout()

if __name__ == '__main__':
    SmartParkingApp().run()
```

```yaml
# main.kv
<BoxLayout>:
    orientation: 'vertical'
    Label:
        text: 'Smart Parking App'
    Button:
        text: 'Check Parking Availability'
    Label:
        id: availability_label
    Button:
        text: 'Reserve Parking'
```

Implement Backend Logic:
In `main.py`, add backend logic to handle parking space availability and user interactions. You can use functions to check availability, reserve parking, and update the UI accordingly.

```python
# main.py (continued)
class SmartParkingApp(App):
    def check_availability(self):
        # Implement logic to check parking availability
        # Update availability_label text based on availability
        pass

    def reserve_parking(self):
        # Implement logic to reserve parking space
        pass

    def build(self):
        layout = BoxLayout(orientation='vertical')
        label = Label(text='Smart Parking App')
        check_button = Button(text='Check Parking Availability')
        check_button.bind(on_press=self.check_availability)
        reserve_button = Button(text='Reserve Parking')
        reserve_button.bind(on_press=self.reserve_parking)
        availability_label = Label(id='availability_label')

        layout.add_widget(label)
        layout.add_widget(check_button)
        layout.add_widget(availability_label)
        layout.add_widget(reserve_button)
        return layout

if __name__ == '__main__':
    SmartParkingApp().run()
```

Integrate Maps and Payment Processing:
You can further enhance your app by integrating maps for parking locations and payment processing using appropriate APIs and libraries. For maps, you can use libraries like `kivy-garden.mapview`. For payment processing, you can integrate payment gateways or services like Stripe.

Testing and Deployment:
Test your app thoroughly on different devices and screen sizes to ensure a seamless user experience. Once you are satisfied, you can package the app for deployment on various platforms using tools like PyInstaller (for desktop) or buildozer (for Android/iOS).

Remember that this is a basic outline, and you can customize and expand the functionality of your smart parking app according to your specific requirements and user needs.

Certainly! Creating a real-time parking availability app using a popular framework like Flutter allows for a seamless and interactive user experience. Below is a step-by-step guide to creating a parking availability app using Flutter.

Set Up Flutter:
First, make sure you have Flutter installed on your system. You can follow the official Flutter documentation to install Flutter: [Flutter Installation Guide](https://flutter.dev/docs/get-started/install).

Create a New Flutter Project:
Create a new Flutter project using the following command:

```bash
flutter create parking_availability_app
cd parking_availability_app
```

3. Update `pubspec.yaml`:
Add any dependencies you might need. For real-time data, you may want to include packages like `http` for making API calls.

```yaml
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.3
```

Run `flutter pub get` to install the dependencies.

Design the UI:
Replace the contents of `lib/main.dart` with the following code to design the UI:

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

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

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> {
  String availability = "Loading...";

  Future<void> fetchParkingAvailability() async {
    final response =
        await http.get(Uri.parse('YOUR_PARKING_API_ENDPOINT'));

    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      setState(() {
        availability = data['available'] ? 'Available' : 'Full';
      });
    } else {
      throw Exception('Failed to load parking availability');
    }
  }

  @override
  void initState() {
    super.initState();
    fetchParkingAvailability();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Parking Availability'),
      ),
```

```
      body: Center(
        child: Text(
          'Parking Status: $availability',
          style: TextStyle(fontSize: 24),
        ),
      ),
    );
  }
}
```

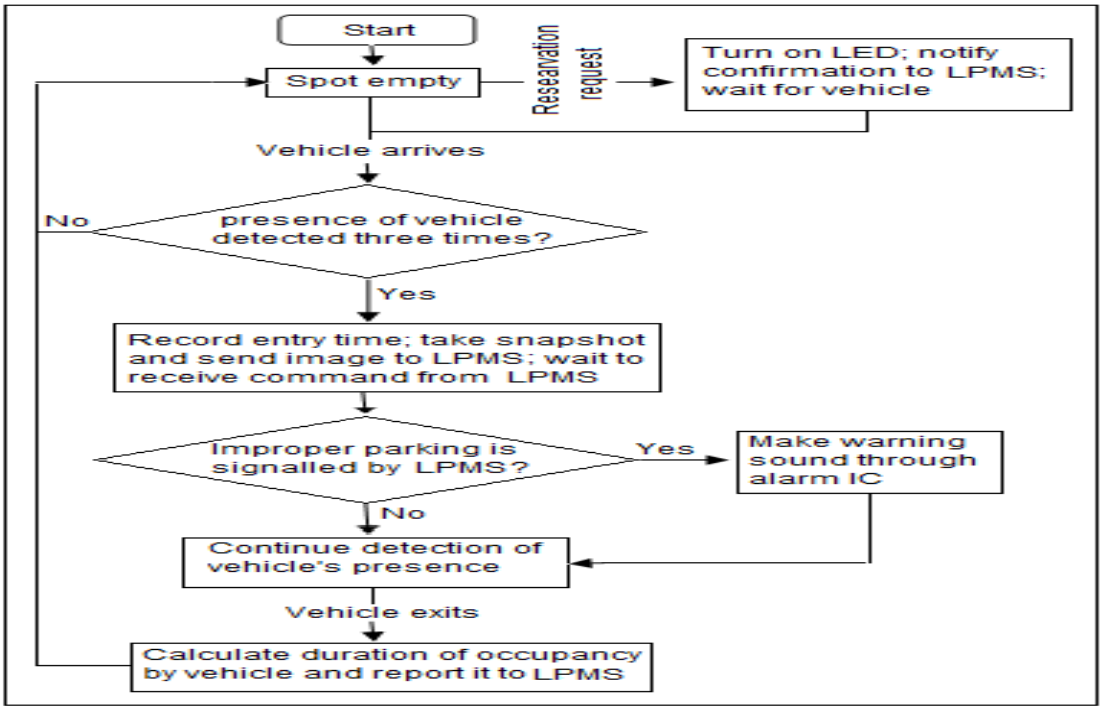
Replace `'YOUR_PARKING_API_ENDPOINT'` with the actual endpoint URL where you can fetch real-time parking availability data.

Test Your App:
Run your app using the following command:

```bash
flutter run
```

This command will launch your app in an emulator or connected device for testing.

This example demonstrates a basic Flutter app that displays real-time parking availability. Make sure to replace the placeholder endpoint with the actual API endpoint providing parking availability data. Additionally, you can enhance the app by adding features like maps, user reviews, and reservation options based on your project requirements.



To design a Flutter app that receives and displays parking availability data from a Raspberry Pi server, you'll need to set up a communication protocol between the app and the Raspberry Pi, such as HTTP or WebSocket. In this example, I'll use HTTP to fetch data from the Raspberry Pi server.

Set Up the Raspberry Pi Server:

On your Raspberry Pi, you can create a simple Python script that acts as a server, providing parking availability data. Here's an example using Flask, a lightweight web framework for Python:

```python
from flask import Flask, jsonify
app = Flask(__name__)

# Simulated parking availability data
parking_data = {
    "available": True
}

@app.route('/api/parking', methods=['GET'])
def get_parking_status():
    return jsonify(parking_data)

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

Save the above code in a file named `app.py` and run it on your Raspberry Pi.

Flutter App Implementation:

Now, modify your Flutter app to fetch and display parking availability data from the Raspberry Pi server. Update the `_ParkingAvailabilityScreenState` class in your `main.dart` file as follows:

```dart
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

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> {
  bool available = false;

  Future<void> fetchParkingAvailability() async {
    final response =
        await http.get(Uri.parse('http://YOUR_RASPBERRY_PI_IP:5000/api/parking'));

    if (response.statusCode == 200) {
      final data = json.decode(response.body);
      setState(() {
        available = data['available'];
      });
    } else {
      throw Exception('Failed to load parking availability');
    }
  }

  @override
  void initState() {
    super.initState();
    fetchParkingAvailability();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Parking Availability'),
      ),
      body: Center(
        child: Text(
          'Parking Status: ${available ? 'Available' : 'Full'}',
          style: TextStyle(fontSize: 24),
        ),
      ),
    );
  }
}
```

```

In this code, replace `'http://YOUR_RASPBERRY_PI_IP:5000/api/parking'` with the IP address of your Raspberry Pi where the Flask server is running.

Now, when you run your Flutter app, it will fetch parking availability data from the Raspberry Pi server and display it in the app. Ensure that your Raspberry Pi and the device running the Flutter app are on the same network for successful communication.
```