**SMART PARKING USING IOT**

**Plan your app:**

Determine the user interface, the navigation structure, and the app's main features. For this app, you may want to consider a feature that allows users to search for nearby parking spots, filter results based on the parking type (street, garage, lot), and display the current availability status (available, occupied) along with real-time data.

**Select a framework:**

 Flutter is an excellent choice for this app. It is a cross-platform UI toolkit that allows you to build natively compiled applications for mobile, web, and desktop from a single codebase.

**Collect and Transmit Data:**

On the Raspberry Pi, create a Python script to collect parking availability data and transmit it. You can use Wi-Fi or any other suitable communication method.

**Create a RESTful API:**

Develop a Python-based RESTful API on the Raspberry Pi using a framework like Flask or FastAPI. This API will expose endpoints for retrieving parking availability data.

**Fetch parking availability data:**

 In your Flutter app, use the 'http' package to make HTTP requests to the Raspberry Pi. The Raspberry Pi should have an API endpoint that sends the current parking availability data. The API response should contain the necessary information such as the parking spot ID, location, parking type, and availability status.

**Collect and Transmit Data:**

On the Raspberry Pi, create a Python script to collect parking availability data and transmit it. You can use Wi-Fi or any other suitable communication method.

**Mobile App Development:**

Choose a mobile app development framework. I'll provide a Flutter example:

**Flutter:**

Install Flutter and set up your development environment.

Create a new Flutter project and configure it.

Use Flutter's Dart programming language to build the mobile app.

**App Features:**

Implement the following features in your app:

a. **User Interface:** Design a user-friendly interface to display parking availability information. Use Flutter's widget library for this purpose.

b. **Integration with API:** Implement HTTP requests to the Raspberry Pi's API to fetch real-time parking availability data.

c. **Real-time Updates:** Make use of WebSockets or periodic polling to keep the parking availability data updated in real-time.

Here's a step-by-step guide to create a parking availability mobile app using Python with the Flutter framework:

**1. Set up the development environment:**

a. Install Flutter: Go to https://flutter.dev/docs/get-started/install and follow the instructions for your operating system.

b. Install Dart: Install Dart SDK by following the instructions at https://dart.dev/get-dart.

c. Set up an editor: You can use any text editor or IDE you like, but we recommend Visual Studio Code.

**2. Create a new Flutter project:**

a. Open a terminal window and run the following command to create a new project:

flutter create parking\_availability

b. Navigate to the project's directory:

cd parking\_availability

**3. Set up Python backend server:**

We need to have a Python backend server to fetch the parking data received from the Raspberry Pi.

a. Install Flask: Create a Python virtual environment and install Flask by running:

python3 -m venv venv

source venv/bin/activate

pip install Flask

b. Create a simple Flask API server to fetch the parking availability data. Create a file named `app.py` and add the following code:

python

from flask import Flask, jsonify

app = Flask(\_\_name\_\_)

@app.route('/api/parking', methods=['GET'])

def get\_parking\_data():

# Replace with your code to fetch parking availability data from Raspberry Pi

parking\_data = {

"available\_spots": 5,

"total\_spots": 10

}

return jsonify(parking\_data)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

c. Run the backend server using the following command:

python app.py

**4. Create the app's user interface:**

a. Open the `lib/main.dart` file in your project directory, remove its contents, and replace them with:

dart

import 'package:flutter/material.dart';

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

import 'dart:convert';

void main() {

runApp(ParkingApp());

}

class ParkingApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Parking Availability',

theme: ThemeData(

primarySwatch: Colors.blue,

),

home: ParkingHomePage(),

);

}

}

class ParkingHomePage extends StatefulWidget {

@override

\_ParkingHomePageState createState() => \_ParkingHomePageState();

}

class \_ParkingHomePageState extends State<ParkingHomePage> {

int \_availableSpots = 0;

int \_totalSpots = 0;

@override

void initState() {

super.initState();

\_fetchParkingData();

}

Future<void> \_fetchParkingData() async {

final response = await http.get('http://localhost:5000/api/parking');

if (response.statusCode == 200) {

final jsonResponse = json.decode(response.body);

setState(() {

\_availableSpots = jsonResponse['available\_spots'];

\_totalSpots = jsonResponse['total\_spots'];

});

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text('Parking Availability'),

),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

crossAxisAlignment: CrossAxisAlignment.center,

children: <Widget>[

Text(

'Available Spots: $\_availableSpots',

style: Theme.of(context).textTheme.headline4,

),

Text(

'Total Spots: $\_totalSpots',

style: Theme.of(context).textTheme.headline4,

),

],

),

),

floatingActionButton: FloatingActionButton(

onPressed: \_fetchParkingData,

tooltip: 'Refresh',

child: Icon(Icons.refresh),

),

);

}

**5. Update your app dependencies:**

In `pubspec.yaml` file, add the following line under the dependencies section:

http: ^0.13.3

Then, run `flutter pub get` in the terminal.

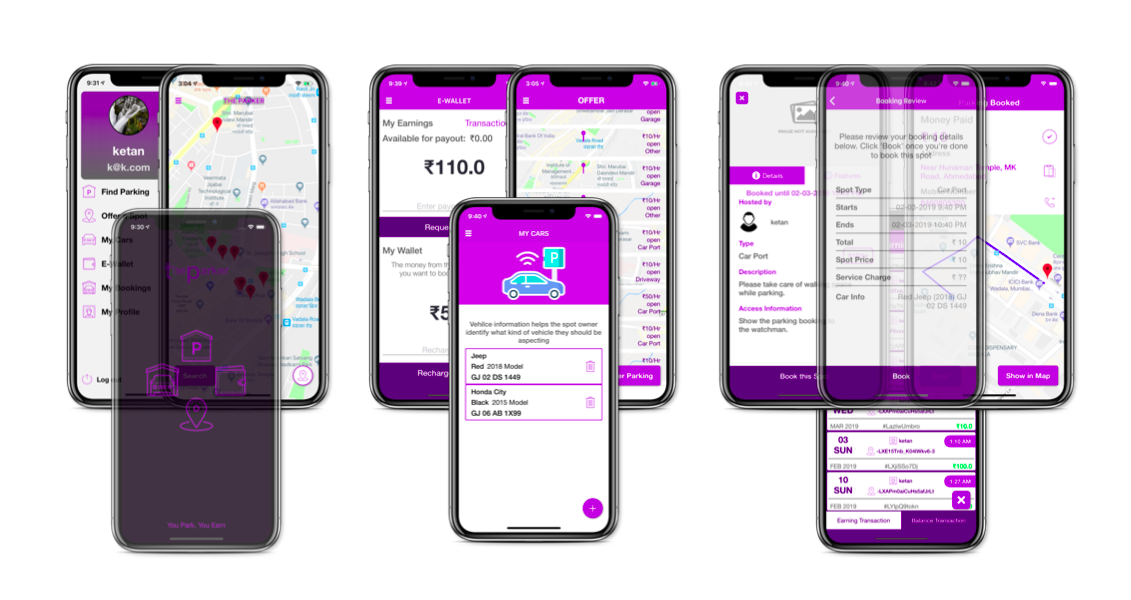
**6. Run the app:**

a. Start an emulator or connect a physical device to your computer.

b. Run the following command in the terminal:

flutter run.

**UI FOR MOBILE APP:**



**User Authentication (Optional):**

If needed, implement user authentication to restrict access to the parking availability information.

**Deployment:**

Deploy your Raspberry Pi in the parking lot and ensure it has a reliable power source and network connection.

**User Documentation:**

Create user documentation or a user manual explaining how to use the mobile app.

**Continuous Improvement:**

Consider adding features like navigation to available parking spots, notification alerts, or even integration with navigation apps.

Remember to secure your communication between the Raspberry Pi and the mobile app, especially if the data is sensitive. You can use HTTPS and implement proper authentication and authorization mechanisms.