SMART PARKING: PHASE – 4

Note: Ensure that you have Flutter and Dart set up on your development environment before starting.

Here's a simple Flutter app that displays parking availability data:

1. Create a new Flutter project using the Flutter CLI:

```bash
flutter create smart_parking_app
```

2. Replace the default `lib/main.dart` with the following code:

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

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

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  String parkingStatus = "Loading...";
```

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

Future<void> fetchParkingAvailability() async {
  final response = await http.get('http://raspberry-pi-ip-address/parking_data');
  if (response.statusCode == 200) {
    // Parse the response and update the parkingStatus variable.
    setState(() {
      parkingStatus = "Available"; // Replace with actual data
    });
  } else {
    setState(() {
      parkingStatus = "Failed to load data";
    });
  }
}

@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('Smart Parking App'),
      ),
      body: Center(
```

```
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'Parking Availability:',
              style: TextStyle(fontSize: 24),
            ),
            Text(
              parkingStatus,
              style: TextStyle(fontSize: 48, fontWeight: FontWeight.bold),
            ),
            ElevatedButton(
              onPressed: () {
                fetchParkingAvailability();
              },
              child: Text('Refresh'),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

3. Update the `http` package in your `pubspec.yaml`:

Add the `http` package to your project's dependencies. Make sure you have the correct version specified. Run `flutter pub get` to fetch the package.

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

4. Customize and Expand:

- Replace `'http://raspberry-pi-ip-address/parking_data'` with the actual URL of your Raspberry Pi server.
- Implement the parsing logic to update the `parkingStatus` variable based on your data.
- Customize the UI and styling to match your app's design.

5. Run Your App:

Use the following command to run your Flutter app:

```bash
flutter run
```

This basic app fetches parking availability data from a server (replace it with your Raspberry Pi's data source) and displays it in a simple UI. You can enhance it with additional features, real-time updates, and a more polished UI to create a full-fledged smart parking app.

To design the app functions for receiving and displaying parking availability data from the Raspberry Pi, you'll need to create a Flutter app with appropriate functionality. Here's a step-by-step guide to design these app functions:

1.Create a Flutter Project if you haven't already, following the steps mentioned earlier.

- 2. Define the Data Model:

- Create a data model to represent parking availability. This model should have properties to store information such as parking space ID, availability status (occupied/available), and any other relevant data.

```dart
Class ParkingSpace {
  Final String id;
  Final bool isOccupied;

  ParkingSpace({
    Required this.id,
    Required this.isOccupied,
  });
}
```

3. Fetch Data from Raspberry Pi:

- Implement a function to fetch data from your Raspberry Pi server. You can use HTTP requests, as mentioned earlier. Make sure to parse the response and convert it into a list of `ParkingSpace` objects.

```dart
Future<List<ParkingSpace>> fetchParkingAvailability() async {
  Final response = await http.get('http://raspberry-pi-ip-address/parking_data');
```

```
  If (response.statusCode == 200) {
    Final List<dynamic> data = json.decode(response.body);
    Return data.map((space) => ParkingSpace(id: space['id'], isOccupied: space['isOccupied'])).toList();
  } else {
    Throw Exception('Failed to load parking availability data');
  }
 }
```

4. Create a UI to Display Data:

- Design the user interface to display parking availability data. You can use Flutter widgets to create a visually appealing UI.

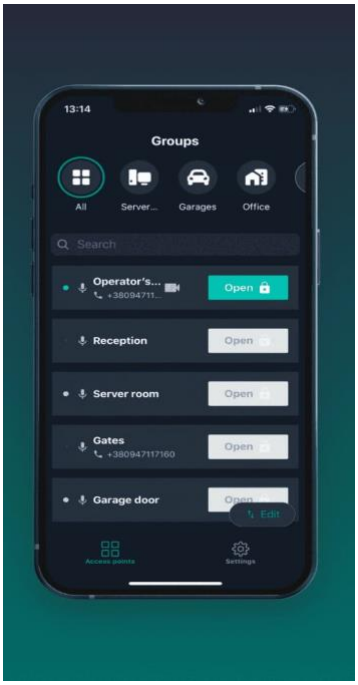
```dart
Class ParkingAvailabilityScreen extends StatelessWidget {
  Final List<ParkingSpace> parkingSpaces;

  ParkingAvailabilityScreen({required this.parkingSpaces});

  @override
  Widget build(BuildContext context) {
    Return Scaffold(
      appBar: AppBar(
        title: Text('Parking Availability'),
      ),
      Body: ListView.builder(
        itemCount: parkingSpaces.length,
        itemBuilder: (context, index) {
          final space = parkingSpaces[index];
```

```
          return ListTile(
            title: Text('Space ${space.id}'),
            subtitle: Text(space.isOccupied ? 'Occupied' : 'Available'),
          );
        },
      ),
    );
  }
}
```

5. Fetch and Display Data:

- In your app, call the `fetchParkingAvailability` function to fetch data and then navigate to the `ParkingAvailabilityScreen` to display the information.

```dart
Future<void> displayParkingAvailability(BuildContext context) async {
  Try {
    Final parkingSpaces = await fetchParkingAvailability();
    Navigator.push(
      Context,
      MaterialPageRoute(builder: (context) => ParkingAvailabilityScreen(parkingSpaces: parkingSpaces)),
    );
  } catch € {
    // Handle errors or display a message to the user.
    Print('Error: $e');
  }
}
```



6. Trigger Data Retrieval:

- You can trigger the data retrieval and screen navigation based on user interactions. For example, you can add a button in your app that, when pressed, calls the `displayParkingAvailability` function to fetch and display the data.

```dart
ElevatedButton(
  onPressed: () {
    displayParkingAvailability(context);
  },
  Child: Text('Check Parking Availability'),
),
```

7. Test and Enhance:

- Test your app on emulators or physical devices, and enhance it with features like real-time updates, error handling, and styling to create a complete smart parking app.

This is a basic outline of how to design app functions to receive and display parking availability data from the Raspberry Pi. You can further refine and customize the app according to your specific needs and design preferences.