

SMART PARKING SYSTEM USING IOT

Creating a mobile app for smart parking based on IoT using a Python framework like Flutter involves developing the frontend part of the app in Flutter while the IoT backend is implemented in Python. Below, I'll outline the steps to create a smart parking mobile app using this combination:

Frontend (Mobile App using Flutter):

Set Up Flutter:

Install Flutter and Dart on your development machine.

Set up your development environment, including an IDE like Android Studio or Visual Studio Code.

Create a New Flutter Project:

Use the flutter create command to set up a new Flutter project.

Design the User Interface:

Design the user interface for the mobile app. You can use Flutter widgets to create screens for parking availability, reservations, and navigation.

User Authentication:

Implement user authentication for user accounts using Firebase Authentication or other authentication solutions for Flutter.

Fetch IoT Data:

Use HTTP requests, WebSocket communication, or other data-fetching mechanisms to retrieve real-time parking availability data from the Python IoT backend.

Display Parking Availability:

Create widgets in Flutter to display parking availability data. Update the UI as new data is received from the backend.

Booking and Navigation:

Implement functionalities for users to make parking reservations and provide navigation to available parking spots.

Push Notifications: Integrate push notification services like Firebase Cloud Messaging (FCM) to notify users about booking confirmations, parking availability updates, and reminders.

Testing:

Thoroughly test the app on both Android and iOS devices to ensure it functions correctly.

Publish Your App:

Submit your app to the Google Play Store and Apple App Store for distribution.

Backend (Python for IoT):

Set Up the Python Environment:

Ensure you have Python installed on the server or device where the IoT components will run.

IoT Device Integration:

Set up IoT devices, such as sensors, cameras, or occupancy detectors, to detect parking spot occupancy.

Develop Python scripts to collect data from these IoT devices and make it available over the network. You may use protocols like MQTT or HTTP to transmit the data.

Web Backend (Python):

Create a Python web backend (Flask, Django, or Fast API) to receive data from the IoT devices.

Store the parking availability data in a database (e.g., SQLite, PostgreSQL, or a NoSQL database).

API for the Flutter App:

Implement RESTful APIs on the Python backend to provide data to the Flutter app. The APIs should allow the app to retrieve real-time parking availability and manage reservations.

Security:

Implement security measures to protect the IoT devices and backend, especially if the system will be used in a real-world environment.

Testing:

Thoroughly test the IoT components and the backend to ensure data is being collected accurately and provided to the app.

This combination of Flutter for the mobile app and Python for the IoT backend offers flexibility and efficiency for building a smart parking system. However, remember that creating a full-fledged system like this involves various complex tasks and may require expertise in mobile app development, IoT, and server-side development.

To design an app function in your mobile app (built with a framework like Flutter) to receive and display parking availability data received from a Raspberry Pi (or any IoT device), you'll need to establish a communication protocol and integrate it into your app. Below are the steps to create this functionality:

1. Set Up Raspberry Pi IoT Device:

On the Raspberry Pi, develop a Python script that collects parking availability data from sensors or cameras.

Use a communication protocol like MQTT or HTTP to transmit this data.

2. Establish a Data Communication Protocol:

Choose a data communication protocol that your app and the Raspberry Pi will use to exchange data. MQTT and HTTP are commonly used for IoT applications.

Ensure that the Raspberry Pi's IoT script publishes data in a format that can be easily consumed by the mobile app.

3. Mobile App Setup:

In your Flutter app, install any necessary packages for communicating with the Raspberry Pi. For example, you can use the `mqtt_client` package for MQTT communication or the `http` package for HTTP requests.

4. Implement Data Reception in Flutter:

Create a Flutter function or class that can subscribe to MQTT topics or make HTTP requests to the Raspberry Pi's API endpoint.

For MQTT:

dart

import 'package:mqtt_client/mqtt_client.dart';

void subscribeToMqttTopic() {

**final client = MqttServerClient('broker.example.com', ''); // Replace with
your MQTT broker details**

client.logging(on: false);

client.keepAlivePeriod = 30;

client.onSubscribed = (String topic) {

print('Subscribed to topic: \$topic');

};

client.onMessage = (String topic, MqttMessage message) {

final String payload =

MqttPublishPayload.bytesToStringAsString(message.payload.message);

// Process the received data (availability status) and update your UI.

};

client.connect();

client.subscribe('parking/availability', MqttQos.exactlyOnce);

}

For HTTP:

dart

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

Future<void> fetchParkingAvailability() async {

**final response = await http.get(Uri.parse('http://raspberrypi-
ip:port/parking/availability')); // Replace with your Raspberry Pi's API
endpoint**

if (response.statusCode == 200) {

final availabilityData = response.body;

// Process the received data and update your UI.

```

    } else {
        throw Exception('Failed to load data');
    }
}

```

5. Display Parking Availability:

Update your app's UI to display parking availability data in a user-friendly way. You can use Flutter widgets such as Text, List View, or Grid View to present the information.

```

    dart
    Text('Parking Availability: $availabilityData'),

```

6. Real-time Updates:

To enable real-time updates, set up periodic data fetching (e.g., using Flutter's Timer.periodic) for MQTT or implement WebSocket communication for HTTP. Update the UI with new data as it arrives.

7. Error Handling:

Implement error handling to deal with situations where the app cannot connect to the Raspberry Pi or if there are issues with data retrieval.

Remember to replace placeholders in the code (e.g., 'broker.example.com', 'http://raspberrypi-ip:port') with your actual server and MQTT broker details. Also, ensure that your IoT device and the mobile app are connected to the same network and that the Raspberry Pi's API or MQTT broker is accessible from the app.

Creating a complete smart parking app using Python for IoT involves several components, including IoT devices to detect parking availability, a server to manage the data, and a mobile app to display real-time information. Here's a high-level overview of how to create a Python-based smart parking app based on IoT:

Requirements:

Python 3.x

Raspberry Pi or IoT devices

Sensors or cameras for detecting parking availability

Flask (a Python web framework)

A mobile app development framework (e.g., Kivy, Flutter, or React Native)

Steps:

1. Set Up IoT Devices:

Connect IoT devices (e.g., Raspberry Pi) to sensors or cameras to detect parking spot occupancy.

Develop a Python script to collect data from these IoT devices.

2. Develop a Python Server:

Create a Python server using Flask to handle IoT data and provide it to the mobile app.

Store the parking availability data in a database (e.g., SQLite or PostgreSQL).

python

```
from flask import Flask, request, jsonify
```

```
import sqlite3
```

```
app = Flask(__name__)
```

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

```
def get_parking_data():
```

```
    # Connect to the database and retrieve parking availability data
```

```
    conn = sqlite3.connect('parking.db')
```

```
    cursor = conn.cursor()
```

```
    cursor.execute('SELECT spot_id, status FROM parking_spots')
```

```
    data = [{'spot_id': spot_id, 'status': status} for spot_id, status in  
    cursor.fetchall()]
```

```
    conn.close()
```

```
    return jsonify(data)
```

```
if __name__ == '__main__':
```

```
    app.run(host='0.0.0.0', port=5000)
```

3. Mobile App Development:

Use a mobile app development framework like Kivy, Flutter, or React Native to create the app.

4. Design the User Interface:

Design the user interface of the mobile app, including screens for displaying parking availability and spot details.

5. Fetch IoT Data:

Implement a function in your app to request data from the Python server.

python

import requests

def fetch_parking_data():

url = 'http://your-server-ip:5000/parking' # Replace with your server's IP or domain

response = requests.get (url)

if response.status_code == 200:

data = response.json()

return data

return []

6. Display Parking Availability:

Create UI components to display parking availability data. Update the UI as new data arrives from the server.

7. Real-time Updates:

Implement real-time updates for parking availability data by periodically querying the server or using WebSockets for real-time communication.

8. Testing and Deployment:

Test the mobile app on various devices and platforms. Deploy the Python server and the mobile app to their respective platforms (e.g., a cloud server for the Python server and app stores for the mobile app).

Consider adding features like user authentication, booking capabilities, payment processing, and navigation to available parking spots.

Here is the sample code create an app in python for smart parking:

```
class ParkingSystem:
```

```
    def __init__(self, total_spots):
```

```
        self.total_spots = total_spots
```

```
        self.available_spots = total_spots
```

```
        self.reserved_spots = set()
```

```
def check_availability(self):
```

```
    return self.available_spots
```

```
def reserve_spot(self, spot_number):
```

```
    if spot_number not in self.reserved_spots and self.available_spots > 0:
```

```
        self.reserved_spots.add(spot_number)
```

```
        self.available_spots -= 1
```

```
        return True
```

```
    else:
```

```
        return False
```

```
def leave_spot (self, spot_number):
```

```
    if spot_number in self.reserved_spots:
```

```
        self.reserved_spots.remove(spot_number)
```

```
        self.available_spots += 1
```

```
        return True
```

```
    else:
```

```
        return False
```

```
def main():
```

```
    parking_system = ParkingSystem(20)
```

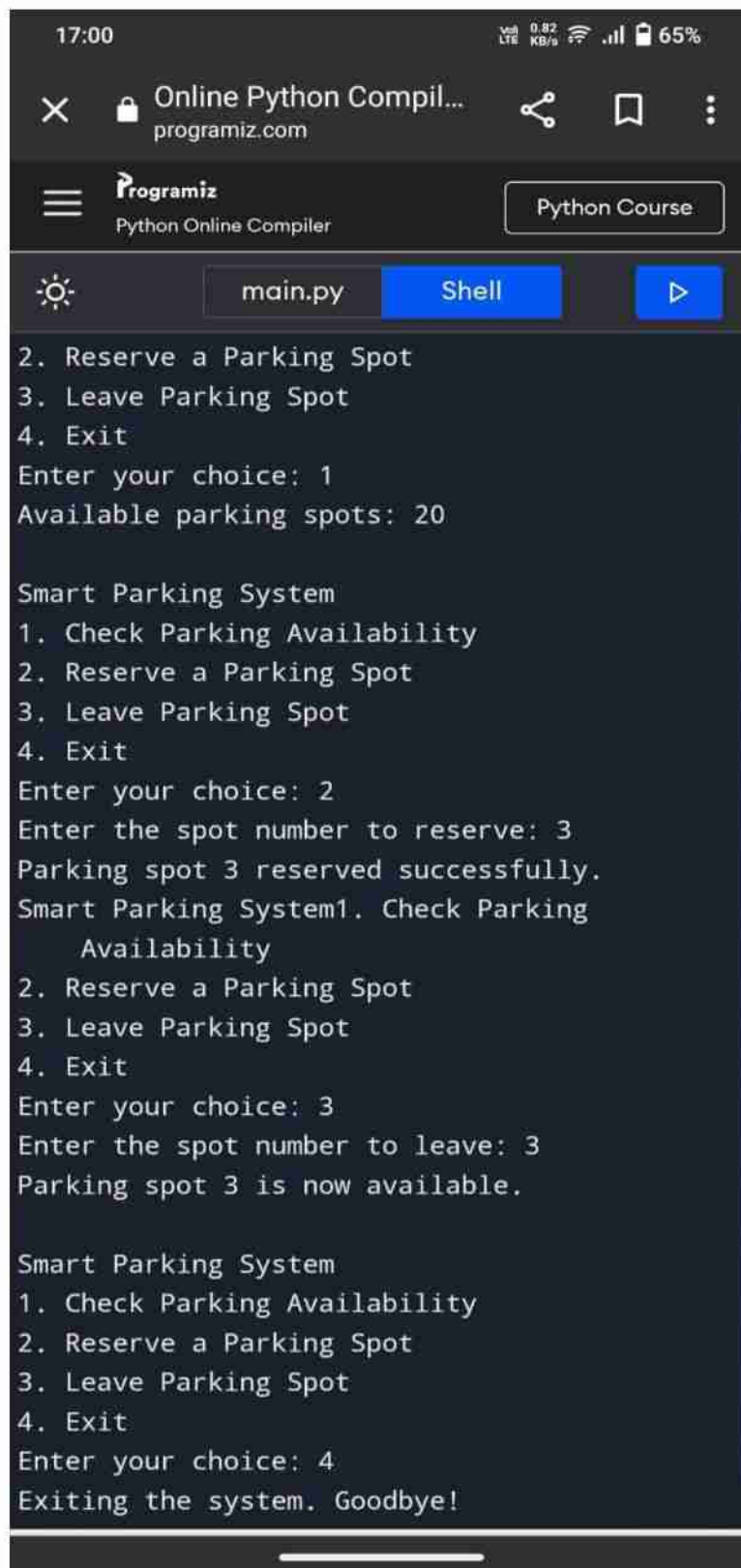
```
while True:
```

```
    print("\nSmart Parking System")
```



```
print("1. Check Parking Availability")
print("2. Reserve a Parking Spot")
print("3. Leave Parking Spot")
print("4. Exit")
choice = input("Enter your choice: ")
if choice == '1':
    printf("Available parking spots:
    {parking_system.check_availability()}")
elif choice == '2':
    spot_number = int(input("Enter the spot number to reserve: "))
    if parking_system.reserve_spot(spot_number):
        printf("Parking spot {spot_number} reserved successfully.")
    else:
        printf("Spot already reserved or no spots available.")
elif choice == '3':
    spot_number = int(input("Enter the spot number to leave: "))
    if parking_system.leave_spot(spot_number):
        printf("Parking spot {spot_number} is now available.")
    else:
        print("Spot not found or already available.")
elif choice == '4':
    print("Exiting the system. Goodbye!")
    break
else:
    print("Invalid choice. Please try again.")
if __name__ == '__main__':
    main()
```

OUTPUT:



The screenshot shows a mobile browser interface for an online Python compiler. The address bar displays 'Online Python Compil...' and 'programiz.com'. The page header includes the 'Programiz' logo, 'Python Online Compiler', and a 'Python Course' button. The editor area shows a file named 'main.py' in 'Shell' mode. The output area displays the execution of a 'Smart Parking System' program. The program starts with a menu: '2. Reserve a Parking Spot', '3. Leave Parking Spot', and '4. Exit'. The user enters '1', and the program shows 'Available parking spots: 20'. The user then enters '2', and the program prompts for a spot number to reserve. The user enters '3', and the program confirms 'Parking spot 3 reserved successfully.' The program then restarts the menu. The user enters '3', and the program prompts for a spot number to leave. The user enters '3', and the program confirms 'Parking spot 3 is now available.' The program restarts the menu. The user enters '4', and the program outputs 'Exiting the system. Goodbye!'.

```
17:00 0.82 KB/s 65%
Online Python Compil...
programiz.com
Programiz
Python Online Compiler
Python Course
main.py Shell
2. Reserve a Parking Spot
3. Leave Parking Spot
4. Exit
Enter your choice: 1
Available parking spots: 20

Smart Parking System
1. Check Parking Availability
2. Reserve a Parking Spot
3. Leave Parking Spot
4. Exit
Enter your choice: 2
Enter the spot number to reserve: 3
Parking spot 3 reserved successfully.
Smart Parking System1. Check Parking
    Availability
2. Reserve a Parking Spot
3. Leave Parking Spot
4. Exit
Enter your choice: 3
Enter the spot number to leave: 3
Parking spot 3 is now available.

Smart Parking System
1. Check Parking Availability
2. Reserve a Parking Spot
3. Leave Parking Spot
4. Exit
Enter your choice: 4
Exiting the system. Goodbye!
```

