SMART PARKING

PHASE 4 - Development part 2

Project Title: Smart Parking

Introduction:

Creating a real-time parking availability app using a framework like Flutter involves several steps. Here's a high-level overview of the process:

Project Setup:

Install Flutter and set up your development environment.
Create a new Flutter project.

UI Design:

Design the user interface for your app, including screens for searching for parking, viewing results, and more.
Use Flutter's widgets to create the UI components.

Data Source:

Decide on a data source for real-time parking availability data. This can be an API from a parking service provider, sensors, or user-generated data.

API Integration:

If you're using an API, integrate it into your app using Flutter's http package or a suitable package for making API requests.

Real-Time Updates:

Implement a mechanism to receive real-time updates. You may need to use technologies like WebSockets or periodic API polling.

Displaying Availability:

Update the UI to display parking availability based on the real-time data.

User Location:

Utilize the device's location services to determine the user's current location.

Search and Filters:

Implement search functionality and filters to help users find parking options based on their preferences (e.g., price, distance, availability).

Navigation and Maps:

Integrate maps to show parking locations and provide directions to selected parking spots.

User Accounts and Booking:

If you want to allow users to book parking spots, implement user account management and booking functionality.

Notifications:

Send notifications to users about parking availability changes or booking confirmations.

Testing:

Thoroughly test your app to ensure it works smoothly and handles real-time data updates correctly.

Optimization and Performance:

Optimize the app's performance and make sure it doesn't drain the device's battery when running in the background.

Deployment:

Deploy your app to the Google Play Store (for Android) or the Apple App Store (for iOS).

Maintenance and Updates:

Regularly maintain and update your app to ensure it continues to function correctly as data sources and technologies evolve.

Parking Availability App Design

User Interface Design:

Create a user-friendly interface for the app that displays parking availability information. This can include a map with parking spots, a list view, or any other visual representation.

Raspberry Pi Setup:

Ensure that the Raspberry Pi is set up with the necessary sensors or cameras to detect parking availability. You might use ultrasonic sensors, cameras, or other IoT devices to collect this data.

Data Transfer Protocol:

Data once on a communication protocol to send data from the Raspberry Pi to the app. Common options are HTTP, MQTT, or WebSocket. Ensure that the Raspberry Pi is programmed to send data using this protocol.

App Development:

Create the app using a programming language and framework of your choice (e.g., Android Studio for Android apps).
Implement the following functions:

a. Connection Setup:

Establish a connection with the Raspberry Pi using the chosen protocol. This involves specifying the IP address and port of the Raspberry Pi.

b. Data Reception:

Implement code to receive data from the Raspberry Pi. This data should include information about parking spot availability.

c. Data Display:

Update the user interface with the received data. For example, you can change the color of parking spots on the map (green for available, red for occupied).

d. Real-Time Updates:

Implement a mechanism for real-time updates. This can be achieved using websockets or by periodically polling the Raspberry Pi for new data.

Data Processing:

Depending on the data format from the Raspberry Pi, you might need to process it to extract relevant information, such as the number of available parking spots or their locations.

Error Handling:

Implement error-handling routines to deal with connection issues, data format errors, and other potential problems.

User Interactions:

Consider allowing users to interact with the app, such as reserving a parking spot or getting directions to an available spot.

Testing:

Thoroughly test the app to ensure that it correctly receives and displays parking availability data from the Raspberry Pi. Test on different devices and under various network conditions.

Deployment:

Once the app is ready, deploy it to the desired platform (e.g. , Google Play Store for Android). Ensure that the Raspberry Pi is accessible over the network.

Security Considerations:

Implement security measures to protect the communication between the app and the Raspberry Pi, as well as the data being transmitted.

Scalability:

If you plan to expand the system, consider how it can handle data from multiple Raspberry Pi units in various parking lots.

User Documentation:

Create user documentation or tooltips to help users understand how to use the app.

Maintenance and Updates:

Plan for future maintenance and updates, including bug fixes and feature enhancements.