SMART PARKING SYSTEM PHASE 4

IoT Smart Parking project, it can enhance the functionality and user experience by incorporating web development technologies. Here's how can integrate web technologies into various aspects of the project:

WEB BASED DASHBOARD ADMINISTRATOR:

Create a web-based dashboard for administrators to monitor and manage the parking system. This dashboard should provide real-time information about parking spot occupancy, reservations, and transaction history. Use web development technologies like HTML, CSS, and JavaScript, and consider using a web framework for efficiency

MOBILE APP :

Develop a mobile app to reserve parking spots, make payments, and receive notifications. Use cross-platform mobile app development frameworks like React Native or Flutter to streamline app development for both Android and iOS.

ONLINE RESERVATION SYSTEM:

Implement a web-based reservation system for students to check parking spot availability and make reservations. This system can be integrated with the mobile app and can be developed using standard web technologies.

PAYMENT GATEWAY INTEGRATION:

If you include a payment system, you'll need to integrate a payment gateway into your web app for processing payments. Popular payment gateways often provide API s for this purpose. Here's a simplified example using Python and Flask

REAL TIME UPDATES:

Use web development technologies to ensure real-time updates on parking spot availability, reservation confirmation, and payment status.

You can achieve this with technologies like Web Socket for real-time communication between the server and clients.

USER AUTHENTICATION AND MANAGEMENT:

For user authentication and management, you can create user registration and login systems within the mobile app and web interface. Use web development technologies for user interfaces and back end logic.

DATA ANALYSING AND REPORTING:

Utilise web technologies to create data analytic and reporting features for administrators. You can use JavaScript libraries for data visualisation and reporting tools.

MOBILE APP DEVELOPMENT:

To connect your IOT Smart Parking System with a mobile app, need to create API s that allow the mobile app to interact with the back end system. Here's a step-by-step guide on how to achieve this.

1.DEVELOP BACKEND APIS:

- Create a set of API endpoints on your server to handle various Functionalists of the Smart Parking System, such as user authentication, parking spot availability, reservations, and payments. You can use a web framework like Express.

2.USER AUTHENTICATION:

- Allow users to register and log in to the mobile app.
- Create API endpoints for user registration and login.
- Implement token-based authentication for secure access to the app.

3.PARKING SPOT AVAILABILITY:

- spot availability Develop an API endpoint to provide real-time information about parking. The mobile app can query this endpoint to display available parking spots to users.

4.RESERVATION:

- ➢ Create APIs for reserving parking spots. When a user selects a spot and reserves it, the mobile app should send a request to the reservation API.
- ➢ Implement logic to check spot availability and confirm the reservation.
- ➢ Return a response to the mobile app with the reservation status.

5.PAYMENT INTEGRATION:

- ➢ Integrate payment gateway APIs, such as Stripe or PayPal, for processing payments.
- ➢ Create API endpoints for initiating and verifying payments. The mobile app can call these endpoints to handle payments.

6.REAL TIME UPDATE:

- ➢ Implement WebSocket communication to provide real-time updates on parking spot availability and reservation confirmation. When a parking spot becomes available or a reservation is confirmed, use WebSockets to push updates to the mobile app.

7.MOBILE APP DEVELOPMENT:

- ➢ Develop the mobile app using a cross-platform framework like React Native or Flutter to ensure compatibility with both Android and iOS.Implement user interfaces for registration, login, parking spot selection, reservations, and payment processing.

8.APP INTEGRATION:

- ➢ Use HTTP requests (e.g., GET, POST, PUT, DELETE) in the mobile app to communicate with the backend APIs.
- ➢ Handle API responses in the app to update the user interface and provide feedback to the user.

9.USER NOTIFICATION:

- ➢ Implement push notifications to notify users of reservation confirmations, payment status, and other important updates.
- ➢ Utilise Firebase Cloud Messaging (FCM) for Android and Apple Push Notification Service (APNs) for iOS.

10.TESTING AND DEBUGGING:

- ➢ Test the mobile app's functionality by creating test scenarios and debugging any issues that arise. Verify that the app can interact seamlessly with the backend APIs.

PROGRAM:

Creating a complete mobile app for an IoT Smart Parking System is a complex task that requires a significant amount of code and development effort. I can provide you with a simplified example of a Python program using the kiva framework to create a basic user interface for a mobile app. Please note that this example is a basic starting point, and it would need to extend it significantly to implement the full functionality of the Smart Parking System.

To create a Python mobile app using the kiva framework, follow these steps.

1. Install Kiva if you haven't already. You can do this using pip .

2.Create a Python script for mobile app. This script will serve as a basic user interface for accessing the parking system features.

```
Class Smart Parking
App:     def build(self):
        layout = BoxLayout(orientation='vertical')


        # Create labels and buttons for different
functionalities         label1 = Label(text="Welcome to
Smart Parking")         label2 = Label(text="Available
Parking Spots: 10")
reserve_button = Button(text="Reserve a Spot")
 payment_button = Button(text="Make a Payment")
```

```
        # Bind functions to buttons

reserve_button.bind(on_release=self.reserve_spot)
payment_button.bind(on_release=self.make_payment
)

        layout.add_widget(label1)
layout .add_widget (label2)
layout .add_widget (reserve_button)
layout .add_widget(payment_button)

        return layout

    Def  Reserve (self, instance):
        # Implement reservation logic here
print ("Reserving a parking spot...")

    Def make payment
( self, instance):         #
Implement payment logic
here         print("Making a
payment...")
If __name__ == '__main__'
```

This code provides a very basic user interface for the Smart Parking System. For a complete app, that would need to design more advanced UI components, implement user authentication, handle responses from the server, and manage the app's navigation flow.

Additionally, for a production-ready app, that might want to consider using a dedicated cross-platform mobile app development framework like React Native, Flutter, or others, as they offer a more robust and scalable approach to mobile app development.