

SMART PARKING

BUILDING THE PROJECT BY DEVELOPING THE MOBILE APP

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.

➤ REACT NATIVE OR FLUTTER: BUILD THE APP'S FRONTEND USING THESE FRAMEWORKS, WHICH ALLOW YOU TO WRITE CODE ONCE AND DEPLOY IT ON MULTIPLE PLATFORMS. ➤ API INTEGRATION: CONNECT THE APP TO THE BACKEND SERVER FOR USER AUTHENTICATION, RESERVATION PROCESSING, AND PAYMENT HANDLING.

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.

- HTML/CSS: Design the reservation interface.
- JavaScript: Develop interactive features, such as selecting a parking spot and specifying the reservation duration.
- Backend: Implement reservation logic on the server side, making use of frameworks like Express.js (Node.js) or Django (Python)

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 APIs for this purpose. Here's a simplified example using Python and Flask:

- Flask: Create an API endpoint to handle payment requests.
- Payment Gateway API: Utilize the API provided by the payment gateway provider (e.g., Stripe, PayPal) for processing payments.
- Frontend Integration: Integrate the payment process into your mobile app or web app, allowing users to enter payment details securely.

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 WebSocket for real-time communication between the server and clients.

- WebSocket: Implement WebSocket communication to push real-time updates to the web and mobile clients when a parking spot's status changes.

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 backend logic:

- HTML/CSS: Design registration and login forms.
- JavaScript: Implement form validation and submission handling.
- Backend: Create user accounts, manage authentication, and store

MOBILE APP DEVELOPMENT

- Mobile App Development To connect your IoT Smart Parking System with a mobile app, need to create APIs that allow the mobile app to interact with the backend system. Here's a step-by-step guide on how to achieve this

Develop Backend APIs:

- ❖ Create a set of API endpoints on your server to handle various functionalities of the Smart Parking System, such as user authentication, parking spot availability, reservations, and payments. You can use a web framework like Express.js (Node.js) or Django (Python) to develop these APIs.

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.

Parking Spot Availability:

Develop an API endpoint to provide real-time information about parking spot availability.

The mobile app can query this endpoint to display available parking spots to users.

Reservations:

- ❑ 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

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.

Real-Time Updates:

- ❑ 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.

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.

API 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.

User Notifications:

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

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.

Deployment:

- ❖ Deploy the mobile app to app stores (Google Play Store and Apple App Store) for public use.

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 Kivy 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 Kivy framework, follow these steps:

1. Create a Python script for mobile app. This script will serve as a basic user interface for accessing the parking system features:
2. Install Kivy if you haven't already. You can do this using pip: `pip install kivy`

PROGRAM:

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.button import Button
class SmartParkingApp(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.addWidget(label1)
layout.addWidget(label2)
layout.addWidget(reserve_button)
layout.addWidget(payment_button)
return layout
def reserve_spot(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__':
    SmartParkingApp().run()
```

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 e responses from the server, and manage the app's navigation flow.



THANK YOU