

COLLEGE CODE: 9133
COURSE:Internet of Things(IoT)
PHASE 4
PROJECT TITLE: smart parking

TEAM MEMBERS

DEEPAK KUMAR

HARIHARASUDHAN

SUDHARSAN

SIYAHARI

SUBASH CHANDRA BOSE

1. WEB-BASED DASHBOARD FOR ADMINISTRATORS:

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.

✦ **HTML/CSS:**

Design the dashboard's layout and style using HTML and CSS.

✦ **JavaScript:**

Implement interactivity for real-time updates, charts, and user management.

✦ **Web Framework:**

You can use popular frameworks like React, Angular, or Vue.js for a organized and responsive interface.

2. 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.*

3. 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).*

4. 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.*

5. 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.*

6. 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 user data securely in a database.*

7. DATA ANALYTICS AND REPORTING:

Utilize web technologies to create data analytics and reporting features for administrators. You can use JavaScript libraries for data visualization and reporting tools.

***Data Visualization Libraries:** Integrate libraries like Chart.js or D3.js to display parking utilization statistics and trends.*

***Backend:** Develop APIs for fetching historical parking data and generating reports.*

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 stepby-step guide on how to achieve this:

1. 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.*

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:



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.*

4. 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.*

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 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.*

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. 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.*

9. 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.*

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.*

11. DEPLOYMENT:

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

12. USER SUPPORT AND UPDATES:

- *Provide ongoing support and maintenance for the mobile app.*
Implement updates as needed, addressing user feedback and making improvements.

By creating a well-designed set of APIs and integrating them into mobile app, that can establish a robust connection between the Smart Parking System and the mobile app, ensuring a seamless and user-friendly experience for students.

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. Install Kivy if you haven't already. You can do this using pip:*

○
pip install kivy

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

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

```
    return layout

defreserve_spot(self, instance):
    # Implement reservation logic here

print("Reserving a parking spot...")

defmake_payment(self, instance):
    # Implement payment logic here

print("Making a payment...")

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 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.