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 ANDMANAGEMENT:

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:

MIT App Inventor is a visual programming environment that allows you to create mobile applications for Android devices. Here's a brief overview of how to get started with mobile application development using MIT App Inventor:

1. *Create a Google Account*: You'll need a Google account to use MIT App Inventor.
2. *Access MIT App Inventor*: Visit the MIT App Inventor website (http://appinventor.mit.edu) and log in with your Google account.
3. *Start a New Project*: Click on "Start New Project" to create a new app.
4. *Design the User Interface (UI)*: MIT App Inventor offers a drag-and-drop interface for designing your app's user interface. You can add buttons, labels, text boxes, and other elements.

6. *Connect Components*: If you're using sensors or external components like Bluetooth devices, you can connect them to your app.
7. *Testing on a Device*: To test your app on an Android device, you can use the MIT AI2 Companion app, which is available on the Google Play Store. Install it on your device and scan the QR code provided in the MIT App Inventor to load your app for testing.
8. *Save and Share*: Be sure to save your project frequently. You can also share your project with others for collaboration.
9. *Publish Your App*: Once your app is complete, you can package it as an APK file for distribution through the Google Play Store or other means.
10. *Learn and Experiment*: MIT App Inventor provides a range of tutorials and resources to help you learn and experiment with app development. Explore these resources to enhance your skills.Remember that MIT App Inventor is a great tool for beginners and those who want to quickly prototype and create simple Android apps. It's suitable for a wide range of projects, from educational apps to utility tools and games.

PROGRAM:

```
import RPi.GPIO as GPIO
import time
# Set up GPIO pins
TRIG = 23
# Ultrasonic sensor trigger pin
ECHO = 24
# Ultrasonic sensor echo pin
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
def distance():
GPIO.output(TRIG, False)
time.sleep(0.1)
GPIO.output(TRIG, True)
time.sleep(0.00001)
GPIO.output(TRIG, False )
while GPIO.input(ECHO) == 0:
```

```
pulse_start = time.time()
while GPIO.input(ECHO) == 1:
pulse_end = time.time()
pulse_duration = pulse_end - pulse_start
# Speed of sound at 20°C is approximately 343 meters per second
# We use the formula: Distance = Speed * Time
distance = (pulse_duration * 34300) / 2
return distance
try:
while True:
dist = distance()
if dist < 20:
# Adjust this value based on your parking space size
print("Parking space occupied")
 else:
print("Parking space available")
time.sleep(2)
except KeyboardInterrupt:
GPIO.cleanup()
```

THANK YOU