

**KGiSL INSTITUTE OF TECHNOLOGY**

(Approved By AICTE, New Delhi, Affiliate to Anna University

Recognized by UGC, Accredited by NBA(IT)

265, KGISL Campus, Thudiyalur Road, Saravanampatti, Coimbatore-641035**.)**

**DEPARTMENT OF**

**ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**NAAN MUDHALVAN - INTERNET OF THINGS**

**SMART PARKING**

**NAME:** KISHORE S

**REG NO:** 711721243048

**NM ID:** au711721243048

**TEAM MENTOR:** Mr**.** Mohankumar M

**TEAM EVALUATOR:** Ms. Akilandeeshwari M

**Phase 4: Development Part 2**

**Problem Statement:**

Our task is to create a smart parking system based on IoT technologies. We intend to track real-time parking space occupancy, provide users with dynamic parking assistance, and seamlessly incorporate these capabilities into a mobile app. The ultimate goal is to improve the efficiency and convenience of public parking services, thereby easing the common challenge of finding available parking places in urban settings.

**DESIGNING AN APP FUNCTIONS TO RECEIVE AND DISPLAY PARKING AVAILABILITY DATA RECEIVED FROM THE RASBERRY PI.**

**PROCEDURE:**

1. Acquire Raspberry Pi: Choose a Raspberry Pi model with built-in Wi-Fi or a USB Wi-Fi dongle for internet connectivity.
2. Install Raspberry Pi OS: Download and install the Raspberry Pi operating system, such as Raspberry Pi OS, on your Raspberry Pi board.
3. Connect Sensors: Use GPIO pins on the Raspberry Pi to connect parking sensors, like ultrasonic or infrared sensors. You may need a breadboard and jumper wires.
4. Python Script: Write a Python script to interface with the sensors and determine parking space occupancy using libraries like RPi.GPIO. Ensure the script can collect data from the sensors accurately.
5. Internet Connectivity: Ensure the Raspberry Pi can access the internet via Wi-Fi or Ethernet, as needed.

Data Transmission from Raspberry Pi:

1. Python Data Collection Script: Create a Python script that periodically collects parking availability data from the sensors.
2. Transmit Data: Use a communication protocol like HTTP requests, MQTT, or others to transmit this data securely to a server or cloud service. For sensitive data, use HTTPS for secure communication.

**SERVER OR CLOUD SERVICE:**

1. Set Up Server or Cloud Service: Deploy a server or use a cloud service (e.g., AWS, Azure, or a web hosting platform) to receive, store, and process data from the Raspberry Pi.
2. Database: Create a database to store parking availability data using databases like MySQL, PostgreSQL, or cloud-based solutions like Firebase or AWS DynamoDB.
3. API: Implement an API on the server to receive data from the Raspberry Pi and to serve this data to the mobile app.

**MOBILE APP DEVELOPMENT (USING FLUTTER):SETUP FLUTTER PROJECT: CREATE A NEW FLUTTER PROJECT USING YOUR PREFERRED IDE.**

1. User Interface (UI): Design the app's user interface, incorporating Flutter widgets for displaying parking availability information and Google Maps or other mapping libraries for parking locations.
2. Data Retrieval from Server: Implement logic to fetch data from the server using HTTP requests (use the http package in Flutter) and update the app's UI to display the received parking availability data.
3. Real-Time Updates: Implement real-time functionality to continuously receive and update parking availability data using packages like WebSocket or Firebase Realtime Database.
4. Mapping and Markers: Utilize mapping libraries in Flutter to show parking locations on a map and display markers indicating availability status.
5. User Alerts and Interaction: Add features for user interaction, like searching for parking, filtering by location, and providing additional information about parking spots. Implement real-time user alerts or notifications.
6. Security: Ensure secure data transmission and implement authentication and authorization mechanisms to control access to parking availability data.
7. Testing and Deployment: Thoroughly test the app with simulated and real data, deploy the server and database components to a reliable hosting environment, and publish the app to app stores or distribute it.
8. Scaling: If necessary, scale your system to handle a large number of parking sensors and users.

**CODE**

**// main.dart**

**import 'package:flutter/material.dart';**

**void main() {**

**runApp(MyApp());**

**}**

**class MyApp extends StatelessWidget {**

**@override**

**Widget build(BuildContext context) {**

**return MaterialApp(**

**title: 'Smart Parking App',**

**home: ParkingMapScreen(),**

**);**

**}**

**}**

**class ParkingMapScreen extends StatelessWidget {**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: AppBar(**

**title: Text('Smart Parking App'),**

**),**

**body: Center(**

**child: Text('Map and Parking Spots'),**

**),**

**);**

**}**

**}**

**const mongoose = require('mongoose');**

**mongoose.connect('mongodb://localhost/smart\_parking\_db', {**

**useNewUrlParser: true,**

**useUnifiedTopology: true,**

**});**

**// Sample route for fetching parking spots**

**app.get('/api/parking-spots', (req, res) => {**

**// Implement logic to fetch parking spots from the database**

**res.json(parkingSpots);**

**});**

**const passport = require('passport');**

**const LocalStrategy = require('passport-local').Strategy;**

**passport.use(new LocalStrategy((username, password, done) => {**

**// Implement user authentication logic here**

**if (authenticated) {**

**return done(null, user);**

**} else {**

**return done(null, false);**

**}**

**}));** **const mongoose = require('mongoose');**

**const parkingSpotSchema = new mongoose.Schema({**

**name: String,**

**location: String,**

**isAvailable: Boolean,**

**});**

**const userSchema = new mongoose.Schema({**

**username: String,**

**password: String,**

**reservations: [{ type: mongoose.Schema.Types.ObjectId, ref: 'ParkingSpot' }],**

**});**

**CONCLUSION:**

Building a smart parking system with IoT sensors and Raspberry Pi integration is a worthwhile project that can provide practical solutions to urban parking issues. You can construct a dependable and efficient parking management system by following the step-by-step techniques and code presented in this topic. A system like this can help optimize parking space utilization, reduce congestion, and improve the entire parking experience for both drivers and parking facility operators. It's a great illustration of how technology can be used to solve real-world problems and improve urban infrastructure.