

**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:** JASFER I

**REG NO:** 711721243038

**NM ID:** au711721243038

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

**TEAM EVALUATOR:** Ms. Akilandeeshwari M

**Phase 5: Project Documentation & Submission**

**Problem Statement:**

Our challenge is to develop a smart parking solution using IoT technology. We aim to monitor real-time parking space occupancy, offer dynamic parking guidance to users, and seamlessly integrate these features into a mobile app. The ultimate goal is to enhance the efficiency and convenience of public parking services, alleviating the common difficulties of finding available parking spaces in urban areas.

**Project Overview**

1. **Objectives**

* **Define the purpose of the project:** Creating a real-time parking availability system using IoT sensors, Raspberry Pi, and a mobile app.
* **Describe the primary goals:** Enhancing parking management, providing real-time information to users, and improving the parking experience.

1. **IoT Sensor Setup**

**Ultrasonic Sensors**

Model: HC-SR04

Purpose: Detection of vehicle presence within specific parking spaces.

Installation: Mounted above each parking spot for accurate detection.

Specifications: Operating voltage 5V, operating current 15mA.

**PIR Motion Sensors**

Model: HC-SR501

Purpose: Detection of general movement in the parking area.

Installation: Positioned at key locations to monitor general occupancy.

Specifications: Operating voltage 5V, standby current <50µA, detection range up to 7 meters.

1. **Raspberry Pi Integration**

**Configuration and Setup:**

The Raspberry Pi serves as the central processing unit for the real-time parking availability system. The following steps outline the configuration and setup process:

**Hardware Configuration**

Raspberry Pi Model: Raspberry Pi 4 Model B

Operating System: Raspbian Buster

Connectivity: Connected to local Wi-Fi network

Power Supply: 5V, 3A power adapter

**Software Installation**

Installed Raspbian Buster OS on a 32GB microSD card following official Raspberry Pi documentation.

Configured the Wi-Fi connection through the terminal using raspi-config.

Enabled SSH for remote access and interfacing.

1. **Integration with IoT Sensors**

The Raspberry Pi was integrated with the following IoT sensors to capture and process parking space occupancy data:

**Ultrasonic Sensors**

Utilized HC-SR04 Ultrasonic Distance Sensor for detecting vehicle presence in parking spaces.

**Connection Diagram:**

VCC pin connected to Pin 2 (5V)

GND pin connected to Pin 6 (Ground)

Trigger pin connected to GPIO Pin 23

Echo pin connected to GPIO Pin 24

**PIR Motion Sensors**

Used HC-SR501 PIR Motion Sensor for detecting movement in specific zones of the parking lot.

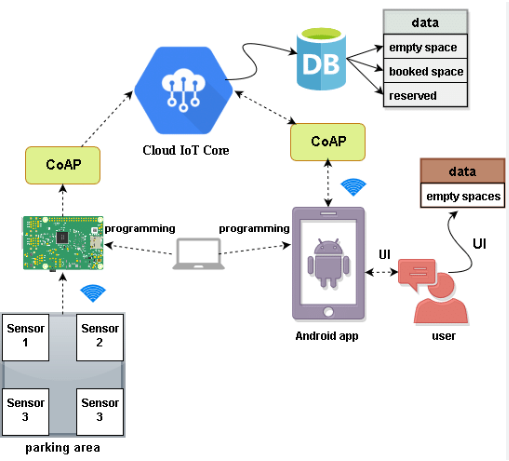
**Connection Diagram:**

VCC pin connected to Pin 4 (5V)

GND pin connected to Pin 9 (Ground)

Signal pin connected to GPIO Pin 17

**Data Processing and Storage**

Python scripts were developed to interact with the sensors and process the data on the Raspberry Pi. These scripts analyzed sensor inputs and updated the parking availability status in real-time. The data was stored locally in JSON format and also transmitted to the cloud for remote access.

1. **Mobile App Development**

**Mobile Platform**

Android OS for wider compatibility.

Programming Language: Java using Android Studio.

**Features**

Real-time Parking Updates

Display available parking spaces to users in real-time.

Navigation to the nearest available parking spot.

User Interface Design

Intuitive interface for ease of use.

Screens with parking availability status and navigation options.

Visual Representations

Include screenshots or mock-ups of the mobile app's interface, highlighting various features and functionalities.

1. **Code Implementation**

**Raspberry Pi Code:**

Python scripts to process sensor data and update parking availability status.

Details on data collection, processing, and storage methods.

**Code:**

import time

import RPi.GPIO as GPIO

import time

import os,sys

from urllib.parse import urlparse

import paho.mqtt.client as paho

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

&#39;&#39;&#39;

define pin for lcd

&#39;&#39;&#39;

# Timing constants

E\_PULSE = 0.0005

E\_DELAY = 0.0005

delay = 1

# Define GPIO to LCD mapping

LCD\_RS = 7

LCD\_E = 11

LCD\_D4 = 12

LCD\_D5 = 13

LCD\_D6 = 15

LCD\_D7 = 16

slot1\_Sensor = 29

slot2\_Sensor = 31

GPIO.setup(LCD\_E, GPIO.OUT) # E

GPIO.setup(LCD\_RS, GPIO.OUT) # RS

GPIO.setup(LCD\_D4, GPIO.OUT) # DB4

GPIO.setup(LCD\_D5, GPIO.OUT) # DB5

GPIO.setup(LCD\_D6, GPIO.OUT) # DB6

GPIO.setup(LCD\_D7, GPIO.OUT) # DB7

GPIO.setup(slot1\_Sensor, GPIO.IN)

GPIO.setup(slot2\_Sensor, GPIO.IN)

# Define some device constants

LCD\_WIDTH = 16 # Maximum characters per line

LCD\_CHR = True

LCD\_CMD = False

LCD\_LINE\_1 = 0x80 # LCD RAM address for the 1st line

LCD\_LINE\_2 = 0xC0 # LCD RAM address for the 2nd line

LCD\_LINE\_3 = 0x90# LCD RAM address for the 3nd line

def on\_connect(self, mosq, obj, rc):

self.subscribe(&quot;Fan&quot;, 0)

def on\_publish(mosq, obj, mid):

print(&quot;mid: &quot; + str(mid))

mqttc = paho.Client() # object declaration

# Assign event callbacks

mqttc.on\_connect = on\_connect

mqttc.on\_publish = on\_publish

url\_str = os.environ.get(&#39;CLOUDMQTT\_URL&#39;,

&#39;tcp://broker.emqx.io:1883&#39;)

url = urlparse(url\_str)

mqttc.connect(url.hostname, url.port)

&#39;&#39;&#39;

Function Name :lcd\_init()

Function Description : this function is used to initialized lcd by

sending the different commands

&#39;&#39;&#39;

def lcd\_init():

# Initialise display

lcd\_byte(0x33,LCD\_CMD) # 110011 Initialise

lcd\_byte(0x32,LCD\_CMD) # 110010 Initialise

lcd\_byte(0x06,LCD\_CMD) # 000110 Cursor move direction

lcd\_byte(0x0C,LCD\_CMD) # 001100 Display On,Cursor Off, Blink Off

lcd\_byte(0x28,LCD\_CMD) # 101000 Data length, number of lines, font

size

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

time.sleep(E\_DELAY)

&#39;&#39;&#39;

Function Name :lcd\_byte(bits ,mode)

Fuction Name :the main purpose of this function to convert the byte

data into bit and send to lcd port

&#39;&#39;&#39;

def lcd\_byte(bits, mode):

# Send byte to data pins

# bits = data

# mode = True for character

# False for command

GPIO.output(LCD\_RS, mode) # RS

# High bits

GPIO.output(LCD\_D4, False)

GPIO.output(LCD\_D5, False)

GPIO.output(LCD\_D6, False)

GPIO.output(LCD\_D7, False)

if bits&amp;0x10==0x10:

GPIO.output(LCD\_D4, True)

if bits&amp;0x20==0x20:

GPIO.output(LCD\_D5, True)

if bits&amp;0x40==0x40:

GPIO.output(LCD\_D6, True)

if bits&amp;0x80==0x80:

GPIO.output(LCD\_D7, True)

# Toggle &#39;Enable&#39; pin

lcd\_toggle\_enable()

# Low bits

GPIO.output(LCD\_D4, False)

GPIO.output(LCD\_D5, False)

GPIO.output(LCD\_D6, False)

GPIO.output(LCD\_D7, False)

if bits&amp;0x01==0x01:

GPIO.output(LCD\_D4, True)

if bits&amp;0x02==0x02:

GPIO.output(LCD\_D5, True)

if bits&amp;0x04==0x04:

GPIO.output(LCD\_D6, True)

if bits&amp;0x08==0x08:

GPIO.output(LCD\_D7, True)

# Toggle &#39;Enable&#39; pin

lcd\_toggle\_enable()

&#39;&#39;&#39;

Function Name : lcd\_toggle\_enable()

Function Description:basically this is used to toggle Enable pin

&#39;&#39;&#39;

def lcd\_toggle\_enable():

# Toggle enable

time.sleep(E\_DELAY)

GPIO.output(LCD\_E, True)

time.sleep(E\_PULSE)

GPIO.output(LCD\_E, False)

time.sleep(E\_DELAY)

&#39;&#39;&#39;

Function Name :lcd\_string(message,line)

Function Description :print the data on lcd

&#39;&#39;&#39;

def lcd\_string(message,line):

# Send string to display

message = message.ljust(LCD\_WIDTH,&quot; &quot;)

lcd\_byte(line, LCD\_CMD)

for i in range(LCD\_WIDTH):

lcd\_byte(ord(message[i]),LCD\_CHR)

lcd\_init()

lcd\_string(&quot;welcome &quot;,LCD\_LINE\_1)

time.sleep(0.5)

lcd\_string(&quot;Car Parking &quot;,LCD\_LINE\_1)

lcd\_string(&quot;System &quot;,LCD\_LINE\_2)

time.sleep(0.5)

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

# Define delay between readings

delay = 5

while 1:

# Print out results

rc = mqttc.loop()

slot1\_status = GPIO.input(slot1\_Sensor)

time.sleep(0.2)

slot2\_status = GPIO.input(slot2\_Sensor)

time.sleep(0.2)

if (slot1\_status == False):

lcd\_string(&quot;Slot1 Parked &quot;,LCD\_LINE\_1)

mqttc.publish(&quot;slot1&quot;,&quot;1&quot;)

time.sleep(0.2)

else:

lcd\_string(&quot;Slot1 Free &quot;,LCD\_LINE\_1)

mqttc.publish(&quot;slot1&quot;,&quot;0&quot;)

time.sleep(0.2)

if (slot2\_status == False):

lcd\_string(&quot;Slot2 Parked &quot;,LCD\_LINE\_2)

mqttc.publish(&quot;slot2&quot;,&quot;1&quot;)

time.sleep(0.2)

else:

lcd\_string(&quot;Slot2 Free &quot;,LCD\_LINE\_2)

mqttc.publish(&quot;slot2&quot;,&quot;0&quot;)

time.sleep(0.2)

**Mobile App Code:**

Java code snippets for real-time data updates and user interactions.

Highlighting key functionalities and data display mechanisms.

**Code:**

var createError = require(&#39;http-errors&#39;);

var express = require(&#39;express&#39;);

var path = require(&#39;path&#39;);

var cookieParser = require(&#39;cookie-parser&#39;);

var logger = require(&#39;morgan&#39;);

var methodoverride = require(&#39;method-override&#39;);

var hbs = require(&#39;hbs&#39;);

var session = require(&#39;express-session&#39;);

var connection = require(&#39;./models&#39;);

var indexRouter = require(&#39;./routes/index&#39;);

var usersRouter = require(&#39;./routes/users&#39;);

var carsRouter = require(&#39;./routes/cars&#39;);

var app = express();

// view engine setup

app.set(&#39;views&#39;, path.join(\_\_dirname, &#39;views&#39;));

app.set(&#39;view engine&#39;, &#39;hbs&#39;);

// Helpers hbs

hbs.registerHelper(&#39;equals&#39;, (val1, val2, options) =&gt; {

return val1 == val2 ? options.fn(this) : options.inverse(this);

});

app.use(session({

secret: &#39;parkingsystem&#39;,

}));

app.use(logger(&#39;dev&#39;));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(methodoverride((req, res, next) =&gt; {

if(req.body &amp;&amp; typeof req.body == &#39;object&#39; &amp;&amp; req.body.\_method) {

var method = req.body.\_method;

delete req.body.\_method;

return method;

}

}));

app.use(express.static(path.join(\_\_dirname, &#39;public&#39;)));

app.use(&#39;/&#39;, indexRouter);

app.use(&#39;/users&#39;, usersRouter);

app.use(&#39;/cars&#39;, carsRouter);

// catch 404 and forward to error handler

app.use(function(req, res, next) {

next(createError(404));

});

// error handler

app.use(function(err, req, res, next) {

// set locals, only providing error in development

res.locals.message = err.message;

res.locals.error = req.app.get(&#39;env&#39;) === &#39;development&#39; ? err : {};

// render the error page

res.status(err.status || 500);

res.render(&#39;error&#39;);

});

module.exports = app;

{

&quot;name&quot;: &quot;parking&quot;,

&quot;version&quot;: &quot;0.0.0&quot;,

&quot;lockfileVersion&quot;: 1,

&quot;requires&quot;: true,

&quot;dependencies&quot;: {

&quot;accepts&quot;: {

&quot;version&quot;: &quot;1.3.5&quot;,

&quot;resolved&quot;: &quot;https://registry.npmjs.org/accepts/-/accepts-1.3.5.tgz&quot;,

&quot;integrity&quot;: &quot;sha1-63d99gEXI6OxTopywIBcjoZ0a9I=&quot;,

&quot;requires&quot;: {

&quot;mime-types&quot;: &quot;2.1.19&quot;,

&quot;negotiator&quot;: &quot;0.6.1&quot;

}

},

&quot;align-text&quot;: {

&quot;version&quot;: &quot;0.1.4&quot;,

&quot;resolved&quot;: &quot;https://registry.npmjs.org/align-text/-/align-text-

0.1.4.tgz&quot;,

&quot;integrity&quot;: &quot;sha1-DNkKVhCT810KmSVsIrcGlDP60Rc=&quot;,

&quot;requires&quot;: {

&quot;kind-of&quot;: &quot;3.2.2&quot;,

&quot;longest&quot;: &quot;1.0.1&quot;,

&quot;repeat-string&quot;: &quot;1.6.1&quot;

}

}

{

&quot;name&quot;: &quot;parking&quot;,

&quot;version&quot;: &quot;0.0.0&quot;,

&quot;private&quot;: true,

&quot;scripts&quot;: {

&quot;start&quot;: &quot;node ./bin/www&quot;

},

&quot;dependencies&quot;: {

&quot;cookie-parser&quot;: &quot;~1.4.3&quot;,

&quot;debug&quot;: &quot;~2.6.9&quot;,

&quot;express&quot;: &quot;~4.16.0&quot;,

&quot;express-session&quot;: &quot;^1.15.6&quot;,

&quot;hbs&quot;: &quot;~4.0.1&quot;,

&quot;http-errors&quot;: &quot;~1.6.2&quot;,

&quot;method-override&quot;: &quot;^3.0.0&quot;,

&quot;mongoose&quot;: &quot;^5.2.4&quot;,

&quot;morgan&quot;: &quot;~1.9.0&quot;

}

}

var Car = require(&#39;./../models/car&#39;);

exports.find = (req, res) =&gt; {

Car.find({}, (err, cars) =&gt; {

if(err) {

return;

}

res.render(&#39;cars\_list&#39;, {

cars: cars

});

});

}

exports.new = (req, res) =&gt; {

res.render(&#39;cars\_new&#39;);

}

exports.create = (req, res) =&gt; {

Car.create(req.body, (err, car) =&gt; {

if(err) {

return;

}

res.redirect(&#39;/cars&#39;);

});

}

exports.edit = (req, res) =&gt; {

Car.findById(req.params.id, (err, car) =&gt; {

if(err) {

return;

}

res.render(&#39;cars\_edit&#39;, {

car: car

});

});

}

exports.update = (req, res) =&gt; {

Car.update({

\_id: req.params.id

}, req.body, (err, car) =&gt; {

if(err) {

return;

}

res.redirect(&#39;/cars&#39;);

});

}

exports.remove = (req, res) =&gt; {

Car.remove({

\_id: req.params.id

}, (err) =&gt; {

if(err) {

return;

}

res.redirect(&#39;/cars&#39;);

});

}

exports.index = (req, res) =&gt; {

res.render(&#39;index&#39;, {

user: req.session.user

});

}

var User = require(&#39;./../models/users&#39;);

exports.login = (req, res) =&gt; {

res.render(&#39;login&#39;);

}

exports.signin = (req, res) =&gt; {

User.findOne({

username: req.body.username,

password: req.body.password

}, (err, user) =&gt; {

if(err) {

return;

}

req.session.user = {

username: user.username

}

res.redirect(&#39;/&#39;);

});

}

exports.register = (req, res) =&gt; {

res.render(&#39;register&#39;);

}

exports.create = (req, res) =&gt; {

User.create(req.body, (err, user) =&gt; {

if(err) {

return;

}

res.redirect(&#39;/users/login&#39;);

});

}

1. **System Integration**

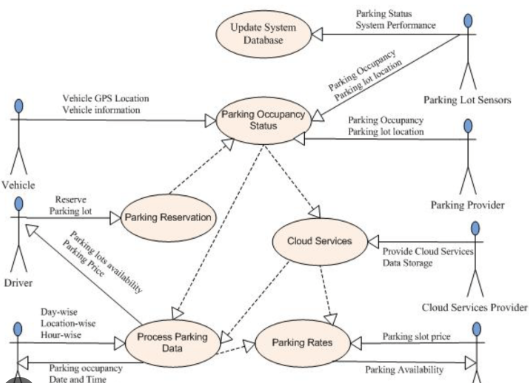
**Data Flow Diagram**

**flow of data from IoT sensors to the Raspberry Pi and subsequently to the mobile app.**



**Real-Time Functionalities**

**the real-time synchronization of parking availability status occurs across the entire system.**

****

1. **Benefits of the Real-Time Parking System**

Improved Driver Experience

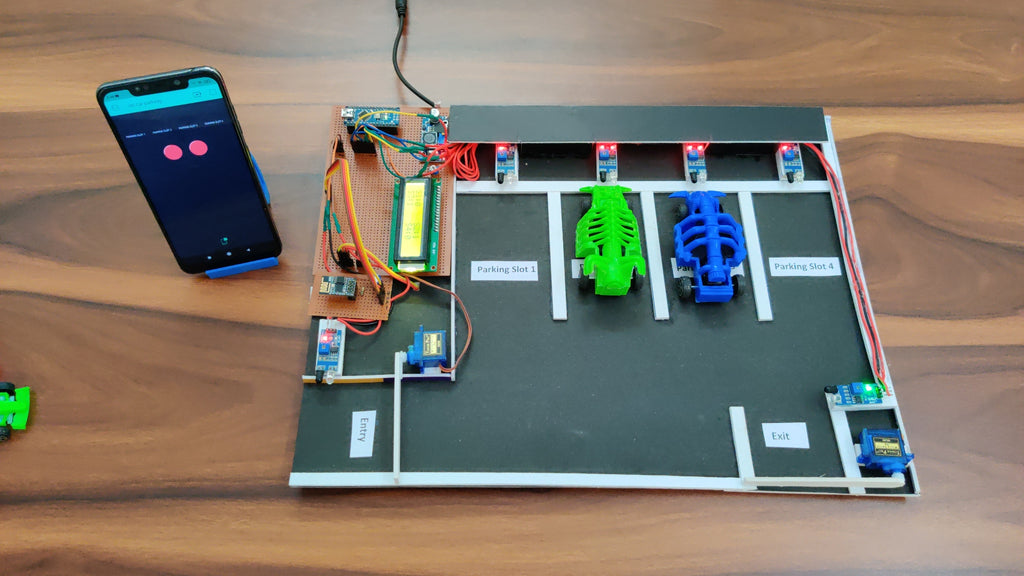
Reduced search time for parking spaces.

Convenience of accessing real-time parking availability updates.

Alleviating Parking Issues

Optimizing parking space utilization.

Contributing to reduced congestion and traffic in urban areas.



**CONCLUSION:**

In conclusion, building a smart parking system using IoT sensors and Raspberry Pi integration is a valuable project that offers solutions to urban parking challenges