Smart Parking System
IOE Mini Project

Submitted in complete fulfillment of the requirements of the degree for the

IOE (Mini Project) Lab

Submitted by

Mr. Mikil Lalwani Roll No.**37**
Mr. Nilay Pophalkar Roll No. **56**
Ms. Sanskruti Punyarthi Roll No. **58**
Mr. Shree Samal Roll No. 61

Under the guidance of

Prof. Charusheela Nehete

Department of Information Technology

Vivekanand Education Society's Institute of Technology

University of Mumbai

2023-2024

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY (VESIT)

Chembur, Mumbai 400074

CERTIFICATE

This is to certify that **Mr. Shree Samal (61)**, **Ms. Sanskruti Punyarthi (58)**, **Mr. Nilay Pophalkar (56)** and **Mr. Mikil lalwani (37)** have satisfactorily carried out the project work, under the head - Internet of Everything Lab at Semester VII of BE in Information Technology as prescribed by the Mumbai University.

Prof. Charusheela Nehete **Subject Teacher**	**External Examiner**
Dr.(Mrs.) Shalu Chopra **H.O.D**	**Dr.(Mrs.) J.M.Nair** **Principal**
Date: 13/10/2022	**Place: VESIT, Chembur**

Declaration

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original source. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

- - - - - - - - - - -
(Signature)

Mr
(Nilay Pophalkar, 56)
B.E. INFT

- - - - - - - - - - -
(Signature)

Ms
(Sanskruti Punyarthi, 58)
B.E. INFT

- - - - - - - - - - -
(Signature)

Mr
(Mikil Lalwani, 37)
B.E. INFT

- - - - - - - - - - -
(Signature)

Mr
(Shree Samal, 61)
B.E. INFT

Contents

List of Figures

List of Tables

Abstract

The Internet of Things (IoT) extends to smart parking systems, which encompass a sophisticated network of physical sensors and devices. These systems are designed to streamline parking space management by employing cutting-edge technologies like Passive Infrared (PIR) sensors for motion detection, Light Dependent Resistors (LDR) for light detection, and Ultrasonic sensors for proximity detection. The integration of these sensors and technologies allows for the efficient and automated management of parking spaces. The PIR sensors detect the presence of vehicles or individuals, the LDR sensors gauge ambient light levels to improve security and visibility, and the Ultrasonic sensors determine the proximity of vehicles to parking spaces. This interconnected system offers real-time monitoring and decision-making capabilities to optimize the utilization of parking resources, leading to more convenient and efficient parking solutions.

Keywords - ***IoT, Smart Parking System, proximity , motion , light intensity***

Chapter 1

Introduction

1.1 INTRODUCTION

Nowadays, the world is fully overtaken by the internet and internet of things. The Internet is used for the basic needs of all human beings. The Internet of Things (IOT) is the network of physical objects. It simply means to monitor a physical device or machine, or it is inter-networking of physical devices which is embedded with electronics, sensors, software and network connectivity to enable it to achieve greater value and services by exchanging data with the manufacturer. IOT permits objects to be sensed or controlled remotely across the network infrastructure. The result improves accuracy, economic benefits, efficiency and reduces intervention of humans.

The Smart Parking System, harnessing the capabilities of IoT, is a project focused on the intelligent control of lighting in parking areas. Through the utilization of sensors such as PIR motion detectors and ultrasonic sensors, the system can discern whether an arriving vehicle has its headlights on or off. This pivotal feature leads to significant energy savings by automatically adjusting the lighting conditions in response to the presence of vehicles. Consequently, drivers benefit from improved visibility and reduced energy waste, enhancing the overall parking experience. Moreover, parking facility operators can reduce operating costs, as unnecessary lighting is eliminated, thus lowering electricity bills and maintenance expenses. Furthermore, this innovative system provides valuable real-time data insights that can inform more efficient energy usage and contribute to environmentally responsible urban planning.

1.2 LITERATURE SURVEY:

Paper 1:

Title: Smart parking systems: comprehensive review based on various aspects

- Authors: Abrar Fahim and Mehedi Hasan
- Summary: The system implements the Adaptive Background Subtraction algorithm as a solution to shadow effects and changes in light. The CNN classifier runs on a smart camera with a very marginal amount of resources. The system is highly robust against the change of lighting conditions, effects of shadow

Paper 2:

Title: "Smart Car Parking System"

- Authors: Aashish Joshi and Arni Tharakaram Hariram
- Summary: The proximity sensor is mounted on the ceiling of the parking lot which consists of an Infra-Red emitter and a receiver. The IR emitter emits infra-red rays and these rays generally bounce off objects.

Paper Title	**Authors**	**Key Points**	**Components Used**	**Merits**	**Demerits**
"Smart parking systems: comprehensive review based on various aspects"	Abrar Fahim and Mehedi Hasan	IoT-based smart parking system uses CNN classifier, Adaptive Background Subtraction algorithm and LIDAR for light detection	LDR, Arduino, cloud platform	Offers superior accuracy, adaptability, and energy efficiency compared to headlight-based approaches.	Limited discussion on the MQTT protocol. Potential scalability concerns not addressed.
"Smart Car Parking System"	Aashish Joshi and Arni Tharakaram Hariram	The proximity sensor is mounted on the ceiling of the parking lot which consists of an Infra-Red emitter and a receiver. The IR emitter emits infra-red rays and these rays generally bounce off objects.	Proximity sensor, Infrared sensor, LDR	Provides a straightforward and cost-effective solution, but it may lack the accuracy and adaptability of advanced technologies like CNN and LIDAR used in the alternative approach.	Lacks discussion on data transmission protocols like MQTT. Limited focus on cloud integration and scalability.

1.3 OBJECTIVES:

- **Demonstrate IoT Integration**: To showcase the successful integration of Internet of Things (IoT) technology in developing a parking system.
- **Real-time Data Collection:** To collect real-time vehicle presence data, lighting control data, motion related data.
- **Alerting Mechanism:** To implement an alerting mechanism using actuators like a buzzer and LED to notify the presence of a vehicle.
- **Accuracy and Precision:** To ensure the accuracy and precision of the vehicular presence data and lighting control system by calibrating and validating the sensors.
- **Cost-effectiveness**: To develop a cost-effective smart parking solution that can be easily replicated.

1.4 HARDWARE AND SOFTWARE REQUIREMENTS:

Hardware:-

- Ultrasonic
- PIR motion detector
- LDR sensor
- Jumper wires
- Buzzer
- LED
- Breadboard
- Raspberry Pi

Software:-

- VS code
- AWS
- Flask

1.5 WIRELESS TECHNOLOGY USED:

Wireless Technologies

1. MQTT:

- **Purpose:** Data collection and storing.
- **Description:** MQTT (Message Queuing Telemetry Transport) is employed for efficient and real-time communication between the IoT sensors and the cloud server. It ensures reliable data transmission and storage.

2. Wi-Fi:

- **Purpose:** Connection of NodeMCU board to AWS.
- **Description:** Wi-Fi technology is employed to establish a reliable connection between the NodeMCU board and the AWS (Amazon Web Services) cloud server. This enables the seamless transmission of sensor data to the cloud for storage and analysis.

1.6 COST ESTIMATION:

Component	Price
Raspberry Pi	4,000
Ultrasonic	100
PIR	150
LDR sensor	20
Buzzer	20
LED	5
Resistor (10Kohm)	1

Total Cost estimated = INR 4296

Chapter 2

IOT System Design

2.1 PURPOSE AND REQUIREMENTS:

Project Purpose:

The Car parking System project seeks to leverage Internet of Things (IoT) technology to create a versatile and accessible solution for real-time data collection and analysis. It aims to empower users with accurate insights into essential parameters including motion detection, proximity detection,Light intensity. Through the integration of sensors, an alerting mechanism with a buzzer and LED, and an intuitive user interface, the project intends to enable informed decision-making and immediate responses to changing parking conditions across various domains. Additionally, it strives to contribute to the knowledge base of IoT-based parking systems and their practical applications.

Requirements:

Hardware Requirements:

- **Sensor Suite:** Ultrasonic sensor, Motion sensor, LDR sensor.
- **Actuators :** LED, Buzzer.
- **Microcontroller:** Deploy a microcontroller for interfacing with the sensors, data processing, and communication
- **Internet Connectivity:** Ensure reliable internet connectivity for seamless data transmission to a cloud-based platform
- **Power Supply:** Provide a stable power source for sensors and microcontroller, considering options like batteries or external power sources
- **Cloud Server:** Set up a cloud server with adequate storage capacity, high-speed internet connection, and robust security measures for data storage and retrieval.

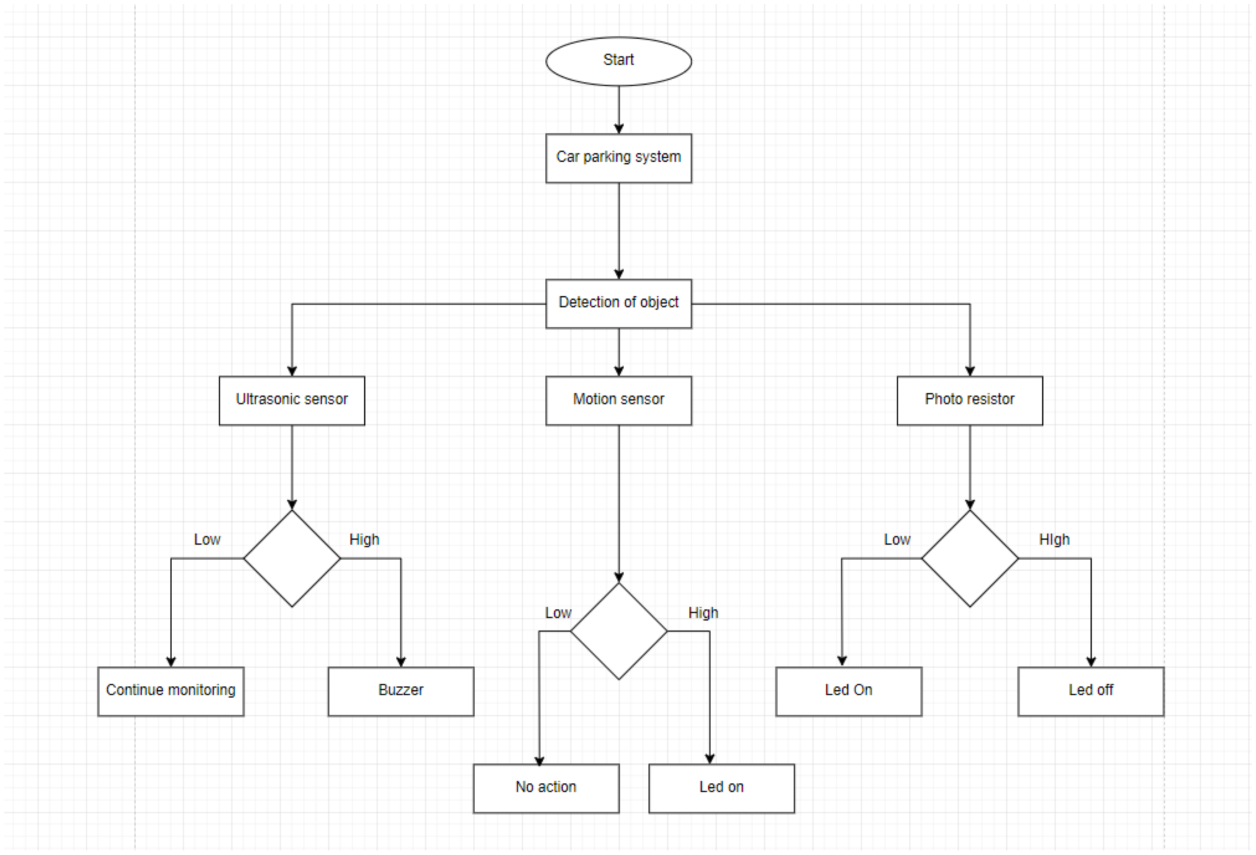
Software Requirements:

- **Microcontroller Programming:** Use programming languages such as C/C++ to program the microcontrollerd for data collection and transmission.
- **Cloud Platform:** Employ a cloud platform (e.g., AWS, Google Cloud, or Microsoft Azure) for data storage, retrieval, and remote access.
- **User Interface Development:** Create a user-friendly dashboard using web development tools (HTML, CSS, JavaScript) for easy data visualization and user interaction.
- **Security Protocols:** Implement security protocols and encryption algorithms to secure data transmission between the system components and the cloud platform.
- **Documentation:** Use documentation tools (e.g., LaTeX or Markdown) to create comprehensive project documentation, including system architecture, sensor specifications, and user guides.

2.2 System Design

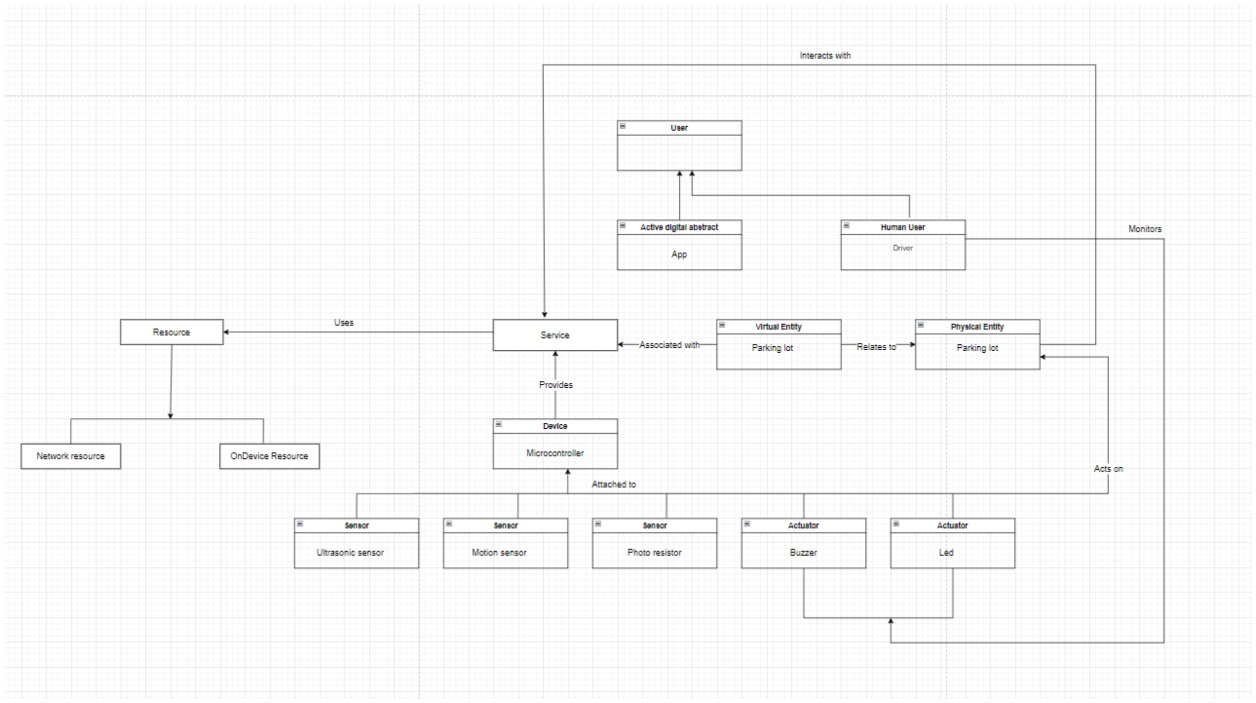
2.2.1 PROCESS MODEL SPECIFICATION :

The process specification shows that the sensors are read after fixed intervals and the sensor measurements are stored.

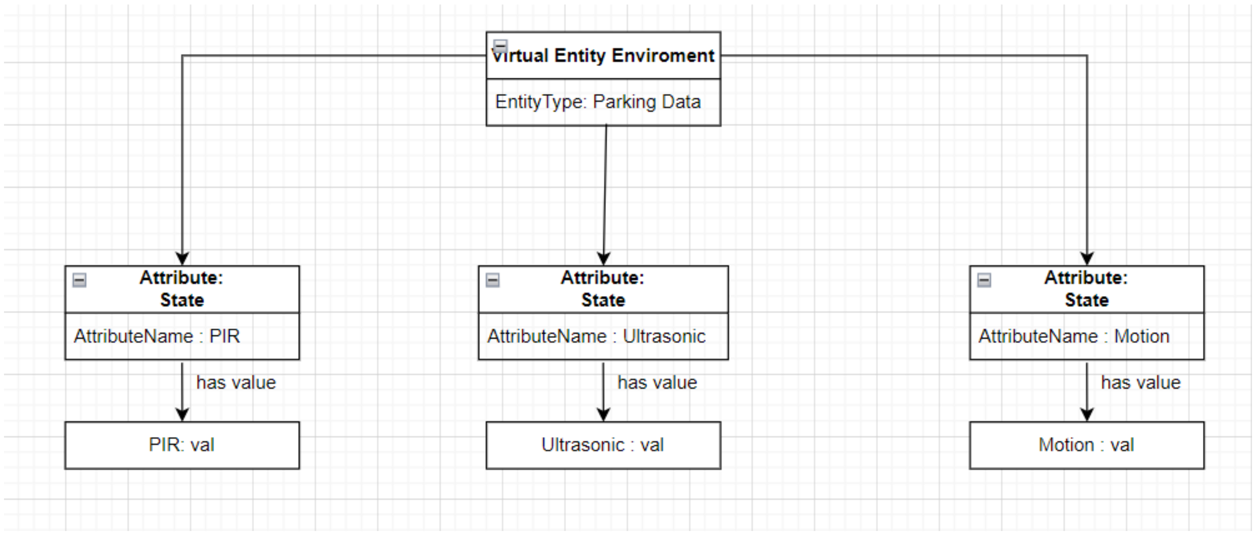


2.2.2 DOMAIN MODEL SPECIFICATION

In this domain model the physical entity is the environment which is being monitored. Hence, there is a virtual entity for the environment. Sensors connected to the single board mini computer include PIR sensor, ultrasonic sensor and LDR sensor. Resources are software components which can be either on device or network-resources.



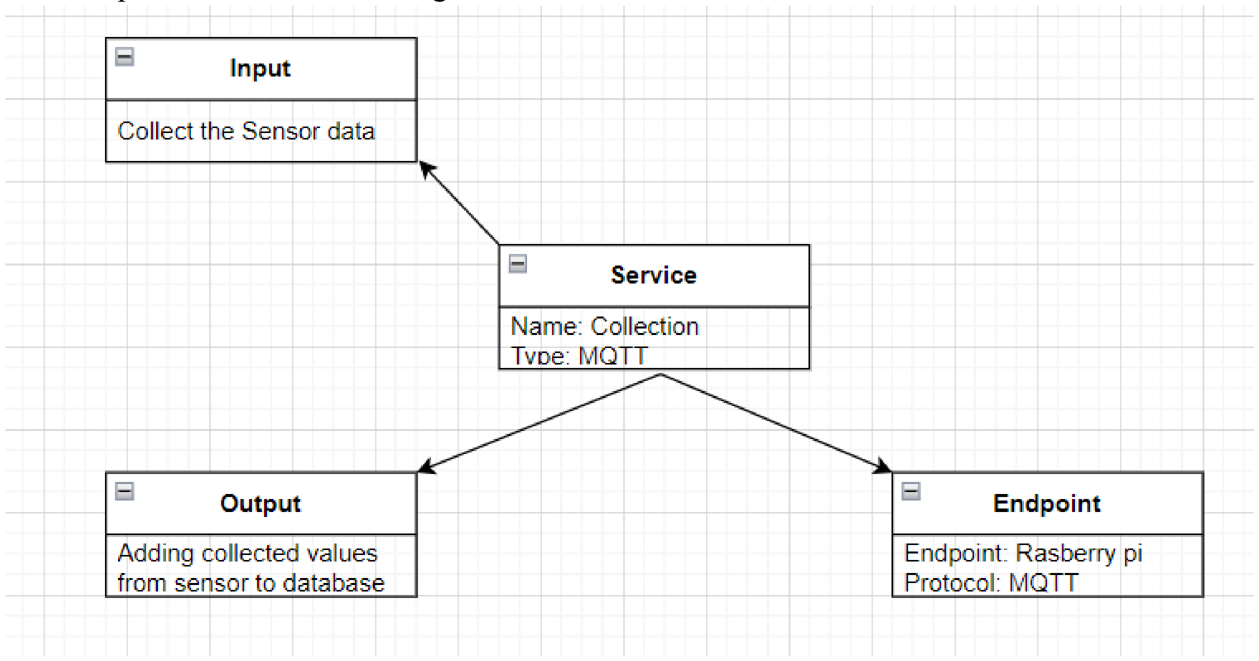
2.2.3 INFORMATION MODEL SPECIFICATION :



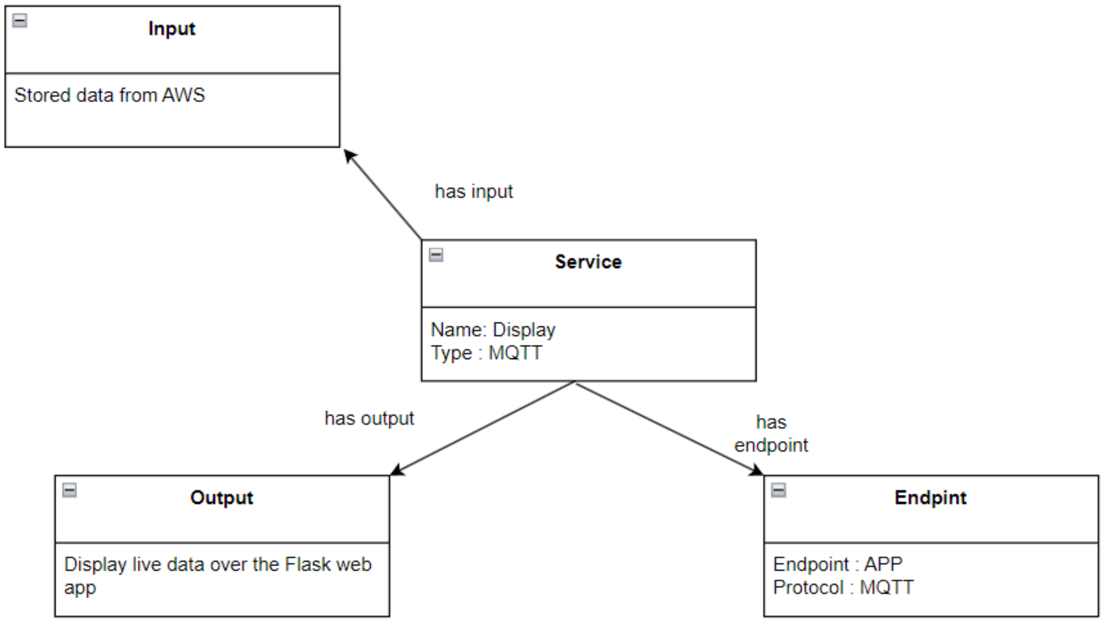
2.2.4 SERVICE SPECIFICATION

Services include the controller service. The controller service runs as a native service on the device and monitors Motion, Proximity, Light intensity once every 3 seconds. The controller service calls the AWS MQTT service to store these measurements in the cloud.

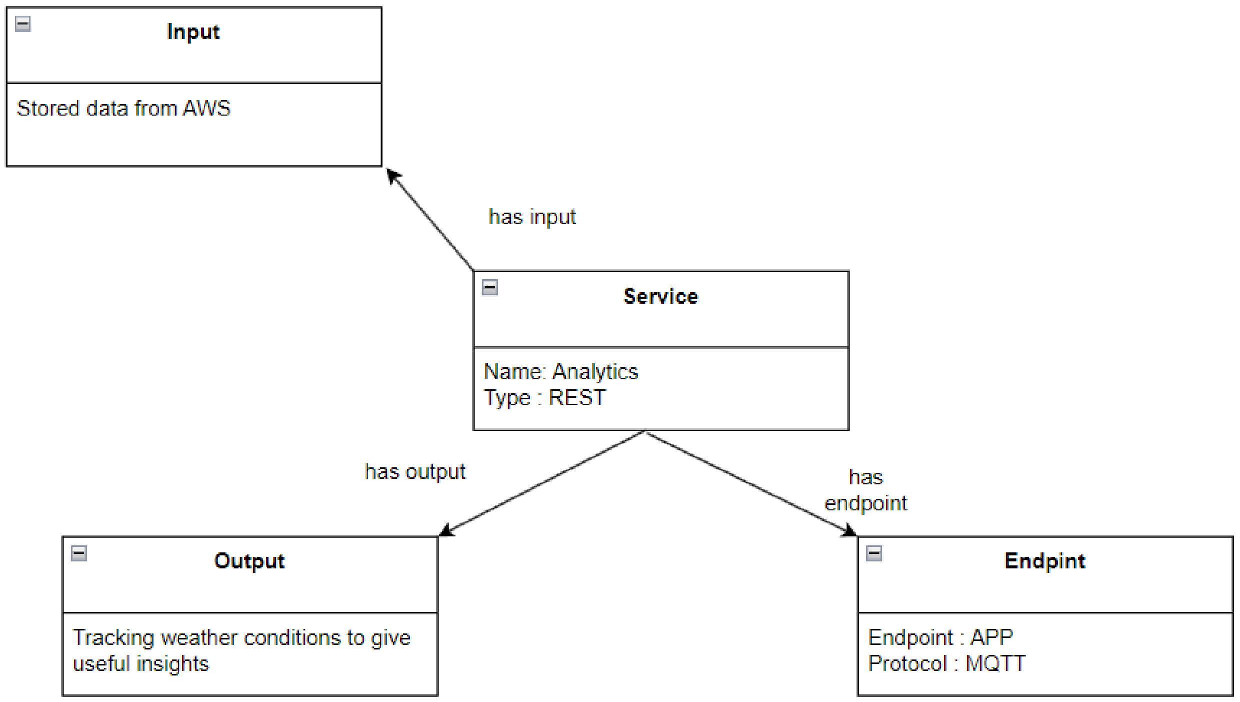
1. Service specification for collecting the data



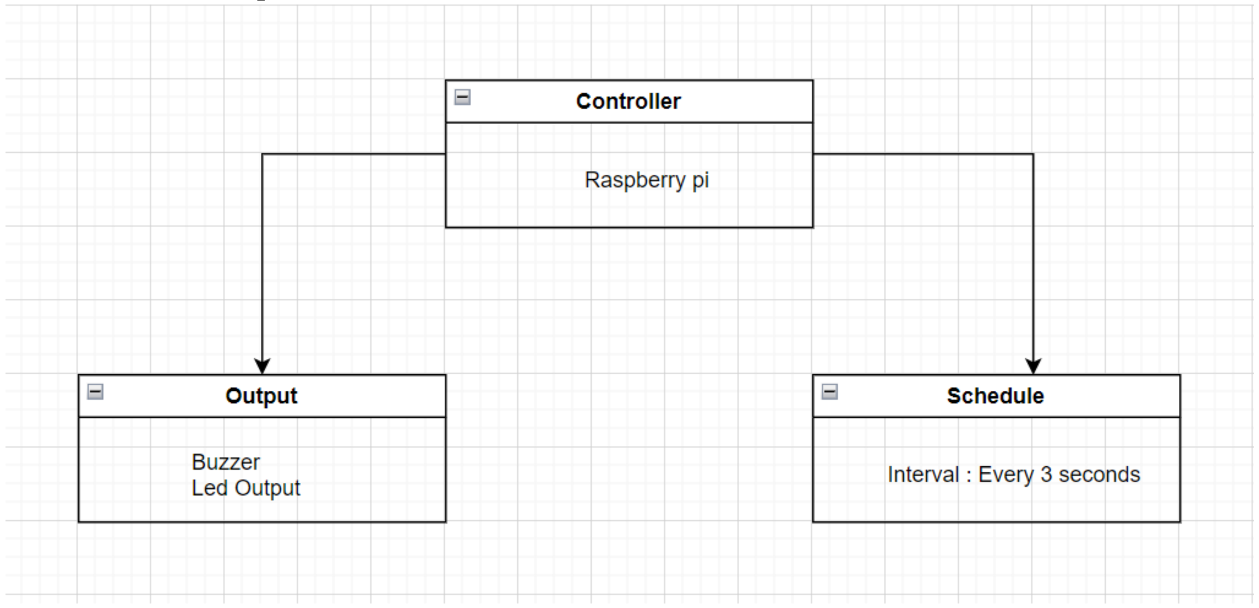
2. Service specifications for displaying data



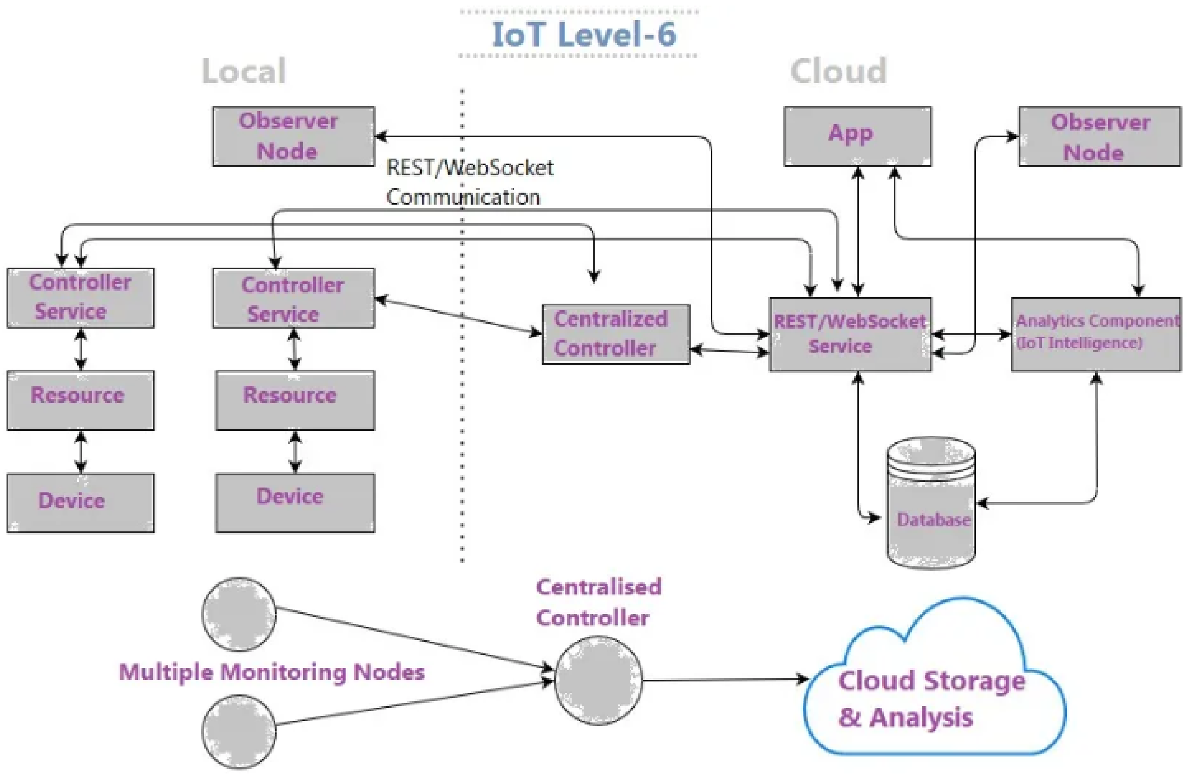
3. Service specifications for analysis of data



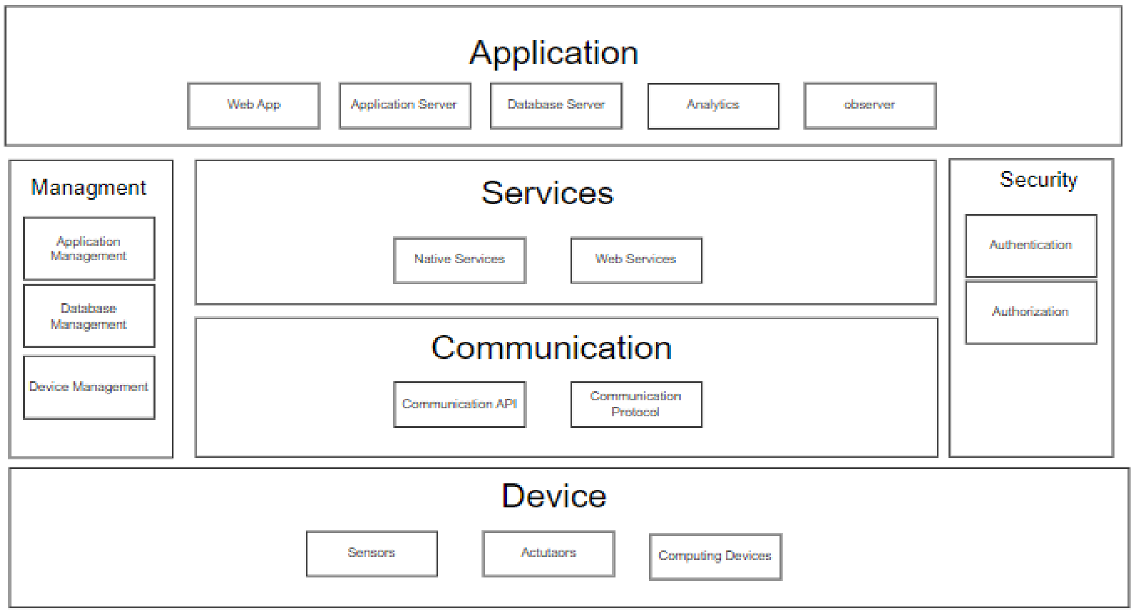
4. Controller Service specification



2.2.5 IOT DEPLOYMENT LEVEL SPECIFICATION



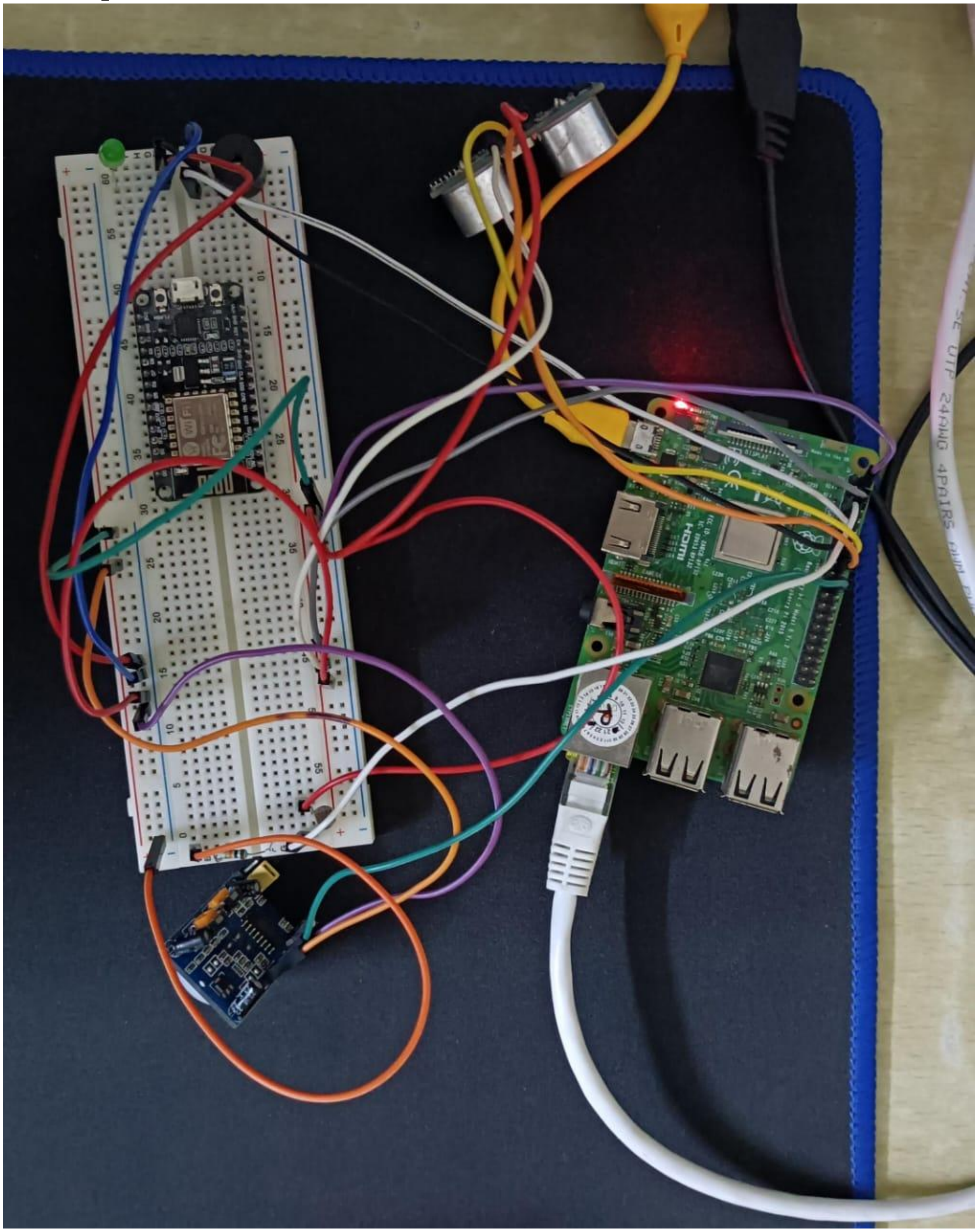
2.2.6 FUNCTIONAL VIEW SPECIFICATION



2.2.7 OPERATIONAL VIEW SPECIFICATION

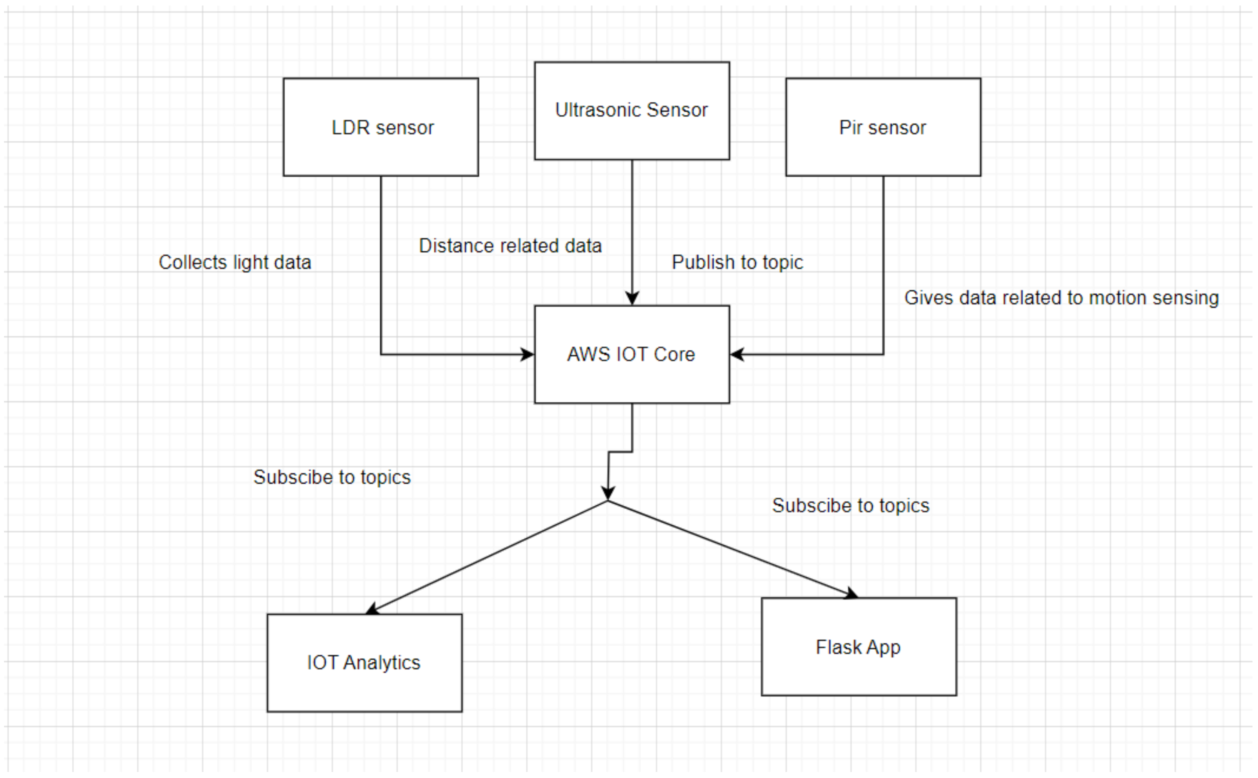
NATIVE SERVICE	**CONTROLLER SERVICE**
Web app :	Flask app
Native service:	Controller Service
Application server :	Flask server
Database server :	Lively cloud storage
Analytics :	Python
Observer :	Cloud app, Web app
Application management :	Flask app management
Database management :	AWS DynamoDB management
Authentication :	Web app
Authorization:	Web App
Device management :	Raspberry pi management
Communication API's:	REST API's, MQTT
Communication protocols :	**IPv6,** TCP, HTTP
Computing device :	Raspberry pi
Sensors:	PIR,Motion,Ultrasonic

2.2.8 Implementation Circuit



2.3 BLOCK DIAGRAM :

The system has functions as shown in parts as shown in the block diagram. The three sensors, namely PIR, LDR, Ultrasonic sensor publish the data to the AWS IoT Core using MQTT protocol. The IoT Analytics application and the Flutter application access the data from AWS MQTT client and analyze or display the data.



Chapter 3

Implementation and Result Analysis

3.1 Interfacing with Real-time Data

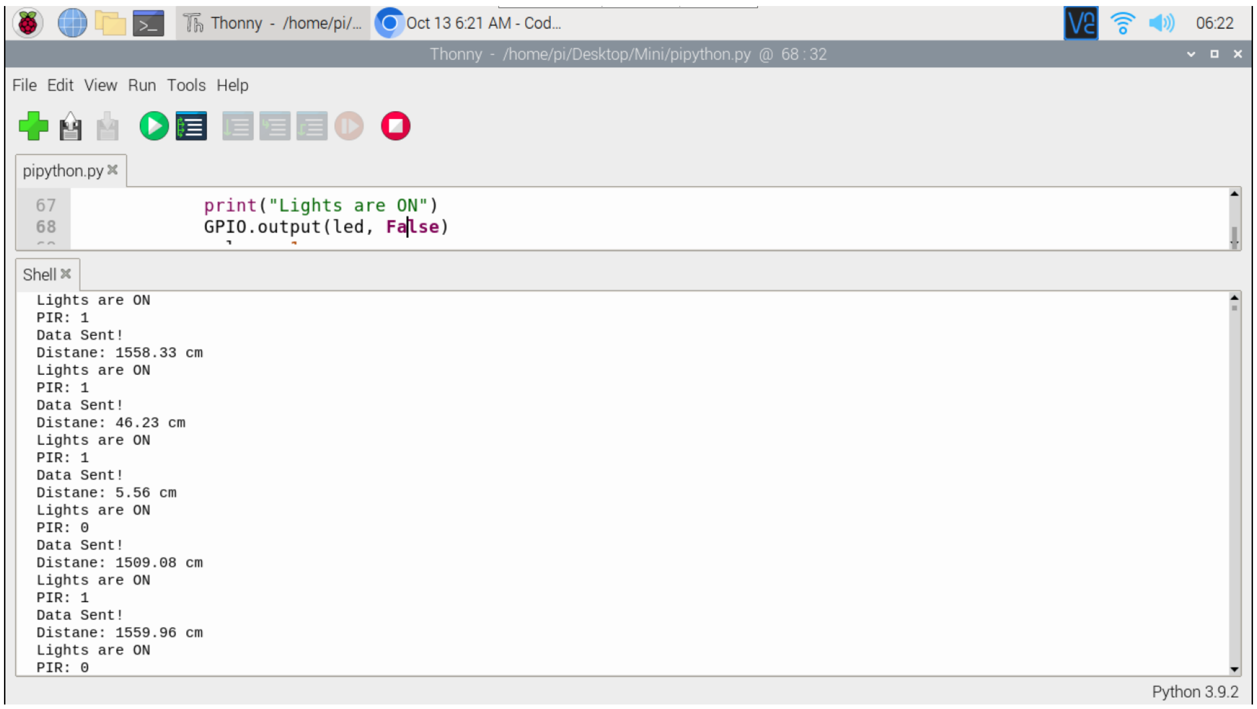
3.1.1 Physical Interfacing with Real-time Data

To establish a sensor data collection and cloud connectivity system using a Raspberry Pi, you can follow these steps. Start by connecting the sensors, such as Ultrasonic, PIR, LDR, Buzzer, and LED, to the Raspberry Pi's GPIO pins, ensuring proper wiring and power supply for each sensor. Next, set up your Raspberry Pi with the required operating system, typically Raspberry Pi OS, and ensure that Python is installed on the device.

Proceed by installing any essential Python libraries for your sensors. You can do this conveniently using Python's package manager, pip, or by following specific installation instructions provided for each sensor. With the sensors properly connected and the requisite libraries in place, write Python code to collect data from the sensors and display it on the terminal. Your Python script should be designed to read sensor values, process the data as needed, and present the results in the terminal.

```
import time
import paho.mqtt.client as mqtt
import ssl
import json
import _thread
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
led = 15
echo = 16
trig = 18
ldr = 7
pir_sensor = 22
buzzer = 11
delayt = .1

GPIO.setup(trig,GPIO.OUT)
GPIO.setup(echo,GPIO.IN)
GPIO.setup(buzzer,GPIO.OUT)
GPIO.setup(led,GPIO.OUT)
GPIO.setup(pir_sensor, GPIO.IN)
```

```
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

def rc_time (ldr):
    count = 0
    GPIO.setup(ldr, GPIO.OUT)
    GPIO.output(ldr, False)
    time.sleep(delayt)
    GPIO.setup(ldr, GPIO.IN)
    while (GPIO.input(ldr) == 0):
        count += 1
        if count > 10000:
            return count
    return count

client = mqtt.Client()
client.on_connect = on_connect
client.tls_set(ca_certs='./rootCA.pem',
certfile='./399b1a3e251bdcda1a6b957eeaf121fd7d96c1538373d1f7d6c505d0fb19f3f0-ce
rtificate.pem.crt',
keyfile='./399b1a3e251bdcda1a6b957eeaf121fd7d96c1538373d1f7d6c505d0fb19f3f0-pri
vate.pem.key', tls_version=ssl.PROTOCOL_SSLv23)
client.tls_insecure_set(True)
client.connect("a1ek9vqv3jnbud-ats.iot.eu-west-1.amazonaws.com", 8883, 60)


def sensors():
    while (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
        distance = pulse_duration * 17150
        print(f'Distane: {distance:.2f} cm')

        if distance <= 20:
            GPIO.output(buzzer, GPIO.HIGH)
            time.sleep(1)
            GPIO.output(buzzer, GPIO.LOW)
        else:
            time.sleep(1)

        value = rc_time(ldr)
        if ( value <= 10000 ):
            print("Lights are ON")
            GPIO.output(led, False)
            value = 1
            time.sleep(1)
        else:
            print("Lights are OFF")
            GPIO.output(led, True)
            time.sleep(1)
            GPIO.output(led, False)
            value = 0


        current_state = GPIO.input(pir_sensor)
        print("PIR:",str(current_state))
        if current_state == 1:
            GPIO.output(led,True)
            GPIO.output(buzzer, GPIO.HIGH)
            time.sleep(2)
            GPIO.output(led,False)
            GPIO.output(buzzer, GPIO.LOW)
        time.sleep(1)

        if (True):
            print("Data Sent!")
            client.publish("device/data",
payload=f"{distance:.2f},{current_state},{value}" , qos=0, retain=False)
        time.sleep(3)

_thread.start_new_thread(sensors,())

client.loop_forever()
```


Thonny - /home/pi/...
Oct 13 6:21 AM - Cod...
06:22
Thonny - /home/pi/Desktop/Mini/pipython.py @ 68 : 32
File Edit View Run Tools Help
pipython.py
Shell
Python 3.9.2

```
            print("Lights are ON")
            GPIO.output(led, False)
```

```
Lights are ON
PIR: 1
Data Sent!
Distane: 1558.33 cm
Lights are ON
PIR: 1
Data Sent!
Distane: 46.23 cm
Lights are ON
PIR: 1
Data Sent!
Distane: 5.56 cm
Lights are ON
PIR: 0
Data Sent!
Distane: 1509.08 cm
Lights are ON
PIR: 1
Data Sent!
Distane: 1559.96 cm
Lights are ON
PIR: 0
```

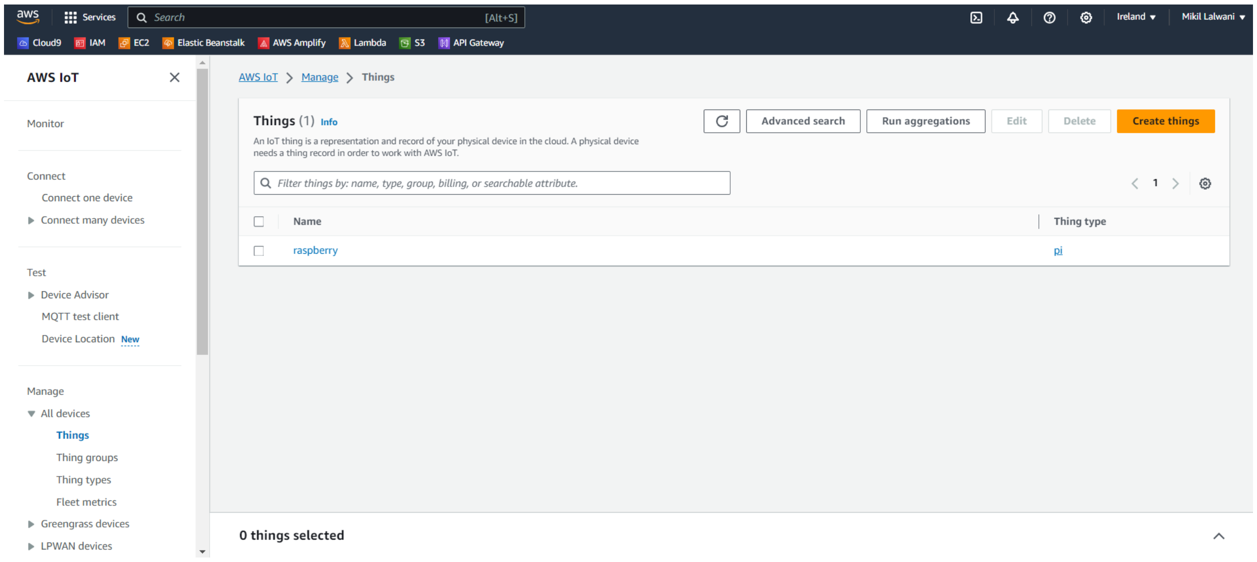
Output of the in terminal

3.1.2 Cloud Interfacing with Real-time Data

AWS IoT provides cloud services and device support that you can use to implement IoT solutions. AWS IoT Core provides the services that connect your IoT devices to the smart parking system the AWS Cloud so that other cloud services and applications can interact with your internet-connected devices. AWS IoT lets you select the most appropriate and up-to-date technologies for your solution. To help you manage and support your IoT devices in the field, AWS IoT Core supports these protocols:

- MQTT (Message Queuing and Telemetry Transport)

We are using the AWS MQTT client for publishing real-time data to AWS. MQTT is a lightweight and widely adopted messaging protocol that is designed for constrained devices. Create an account on AWS using the email ID and password. Go to the IoT Core service from the AWS console. Specifying thing properties, configure device certificate and attach policies to certificate. Download and store the certificates and keys in the same folder as the .ino code file.



Things (1) Info

An IoT thing is a representation and record of your physical device in the cloud. A physical device needs a thing record in order to work with AWS IoT.

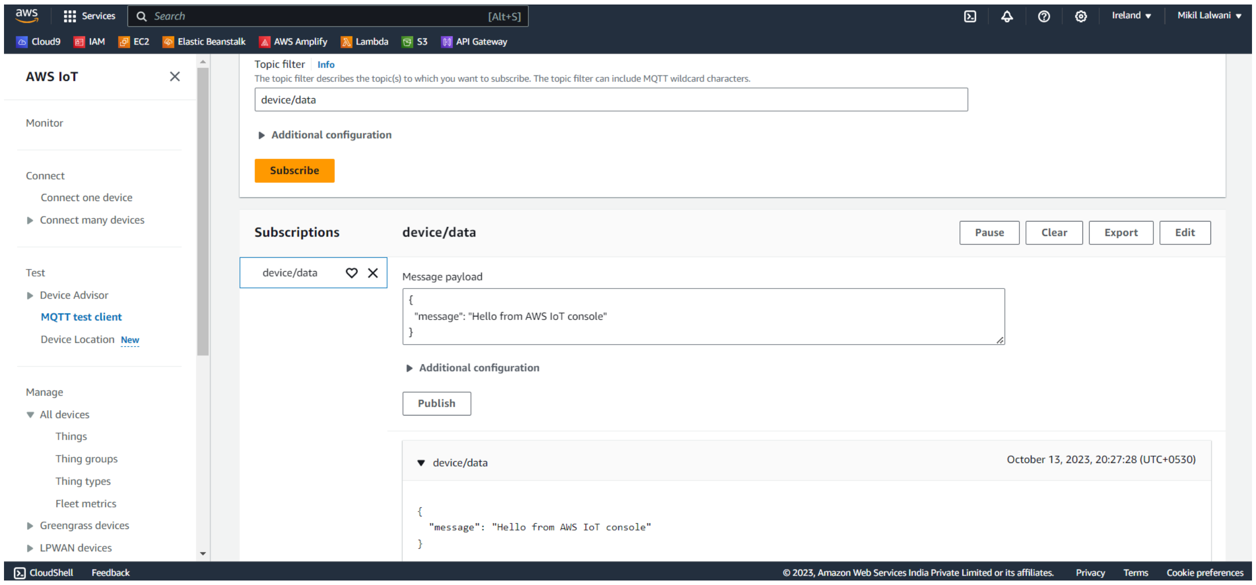
Name	Thing type
raspberry	pi

0 things selected

Install the required libraries and write the following code that connects and publishes the code to the AWS MQTT client.

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
import time
import json

def customCallback(client, userdata, message):
    print("Callback came...")
    print("Topic: " + message.topic)
    payload = message.payload.decode()
    print("Message: " + payload)

    save_to_json(payload)

def save_to_json(data):
    try:
        with open('received_data.txt', 'a') as txt_file:
            txt_file.write(data)
            txt_file.write('\n')
        print("Data saved to received_data.txt")
    except Exception as e:
        print("Error saving data to text file:", str(e))
```

```
myMQTTClient = AWSIoTMQTTClient("device")
myMQTTClient.configureEndpoint("a1ek9vqv3jnbud-ats.iot.eu-west-1.amazonaws.com", 8883)

myMQTTClient.configureCredentials("./rootCA.pem",
"./399b1a3e251bdcda1a6b957eeaf121fd7d96c1538373d1f7d6c505d0fb19f3f0-private.pem.key",
"./399b1a3e251bdcda1a6b957eeaf121fd7d96c1538373d1f7d6c505d0fb19f3f0-certificate.pem.crt
")

myMQTTClient.connect()
print("Client Connected")

# Subscribe to the analytics_topic with QoS 1
myMQTTClient.subscribe("device/data", 1, customCallback)
print('Waiting for the callback. Press Enter to continue...')

# Wait for user input before unsubscribing
x = input()

# Unsubscribe from the analytics_topic
myMQTTClient.unsubscribe("device/data")
print("Client unsubscribed")

# Disconnect from AWS IoT Core
myMQTTClient.disconnect()
print("Client Disconnected")
```



Topic filter Info

The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

device/data

device/data

Message payload

```
{
 "message": "Hello from AWS IoT console"
}
```

device/data October 13, 2023, 20:27:28 (UTC+0530)

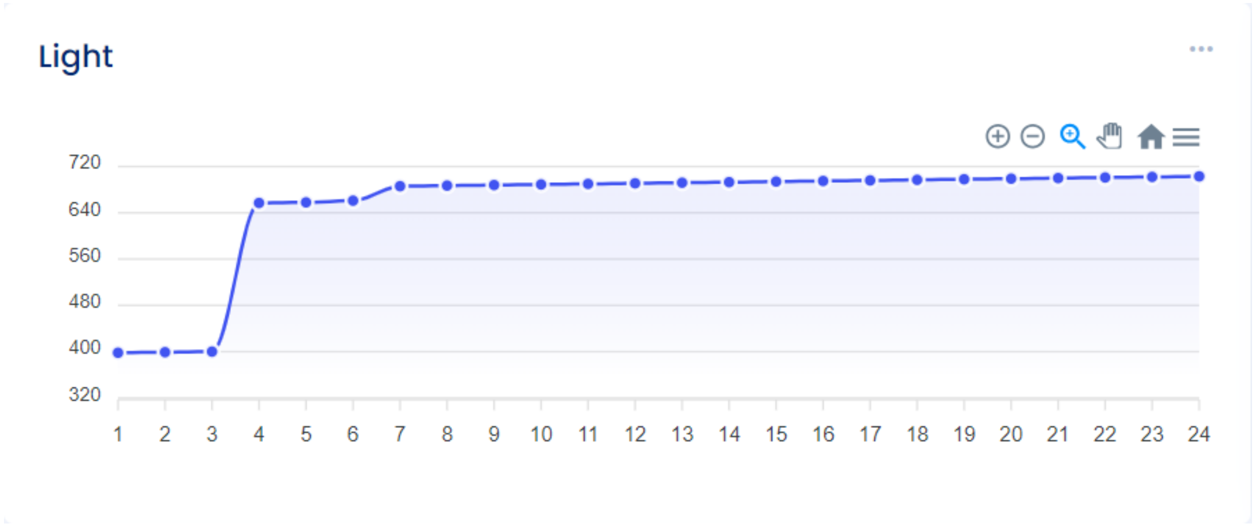
```
{
  "message": "Hello from AWS IoT console"
}
```

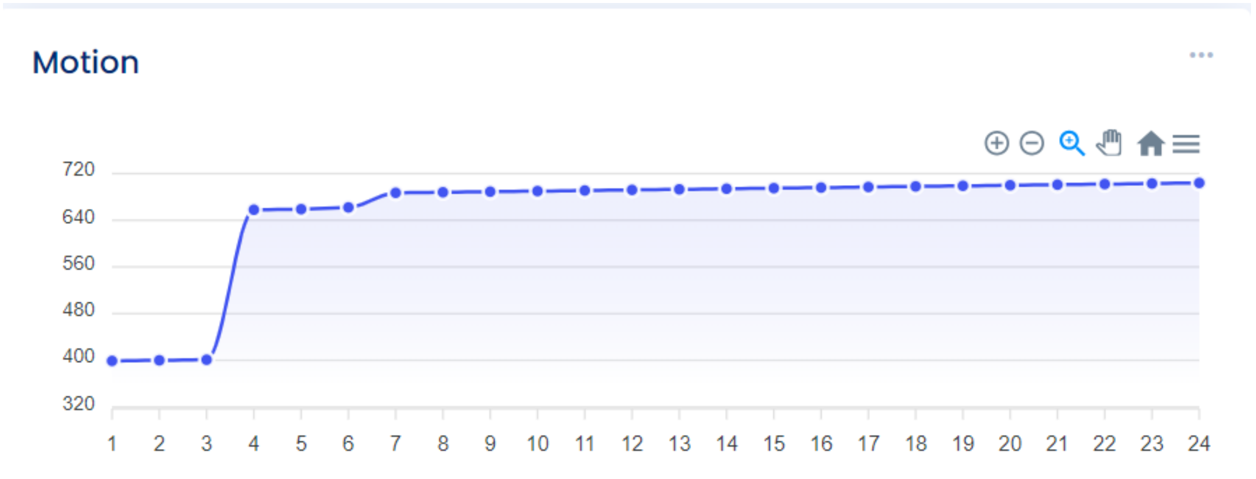
© 2023, Amazon Web Services India Private Limited or its affiliates.

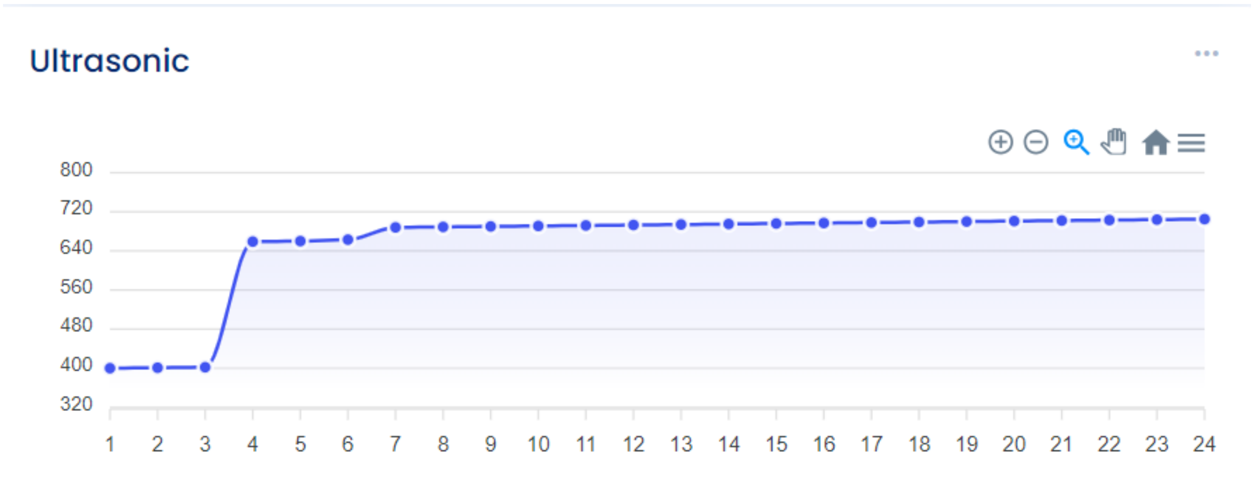
Subscribing to the topic on AWS

3.2 Outcome of the analysis

We have built the analytics application using the Python Flask framework. Flask is a web framework, it’s a Python module that lets you develop web applications easily. It has a small and easy-to-extend core. It does have many cool features like url routing, template engine. It is a WSGI web app framework. Flask is based on the Werkzeug WSGI toolkit and the Jinja2 template engine. Flask is often referred to as a microframework. It is designed to keep the core of the application simple and scalable. Instead of an abstraction layer for database support, Flask supports extensions to add such capabilities to the application.The analytics application captures the MQTT data for a fixed number of times and stores it in a dataset that will be used later. The stored data can be visualized using the web app as shown.







Stored Data Visualisation

3.3 Application Development

The application is built using Flask, an open-source web framework for developing web applications. Flask is widely used for creating web applications that can run on various platforms, including websites. The application allows users to monitor the parking lot and receive notifications about current status.

To connect to AWS, we have used the MQTT client package for Python. This package is designed to handle MQTT protocol tasks automatically, simplifying the process of connecting to AWS and exchanging MQTT messages.

The main functions implemented in the Flask web application are as follows:
1. Home Page: This page serves as the landing page as well as page where all graphs and outcome of the analysis will be shown for our application, providing an overview.

The following packages and dependencies were used in building the Flask web application:

- Flask: The core web framework for building the application.
- Jinja2: The template engine used for rendering HTML templates.
- Other packages for handling MQTT communication, HTTP requests, and data manipulation.

3.4 Implementation Code and Snapshots

```
from flask import Flask, render_template, jsonify
import math
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html',title='Home')

@app.route('/api/getData')
def getData():
    pir_list = []
    motion_list = []
    ultrasonic_list = []
    with open('received_data.txt', 'r') as file:
        for line in file:
            values = line.strip().split(',')
            pir_list.append(float(values[2]))
            motion_list.append(float(values[1]))
            ultrasonic_list.append(float(values[0]))
    last = 0
    for i in range(1,len(ultrasonic_list)):
        if math.floor(ultrasonic_list[i-1]) == math.floor(ultrasonic_list[i]):
            last+=1
        else:
            last=1
    print(last)
    maxi = 0
    cnt = 1
    for i in range(0, len(ultrasonic_list)):
        if i>0 and ultrasonic_list[i-1]==ultrasonic_list[i]:
            cnt+=1
        maxi = max(maxi, cnt)
    print(maxi)
    return jsonify({'pir': pir_list[-10:], 'motion': motion_list[-10:], 'ultrasonic':
ultrasonic_list[-10:],'last':last,'maxi':maxi})

# Enable debug mode
app.debug = True

# Your Flask app routes and other configurations go here
if __name__ == '__main__':
    app.run(debug=True)
```

3.4.1 Script to plot Graphs & Analysis

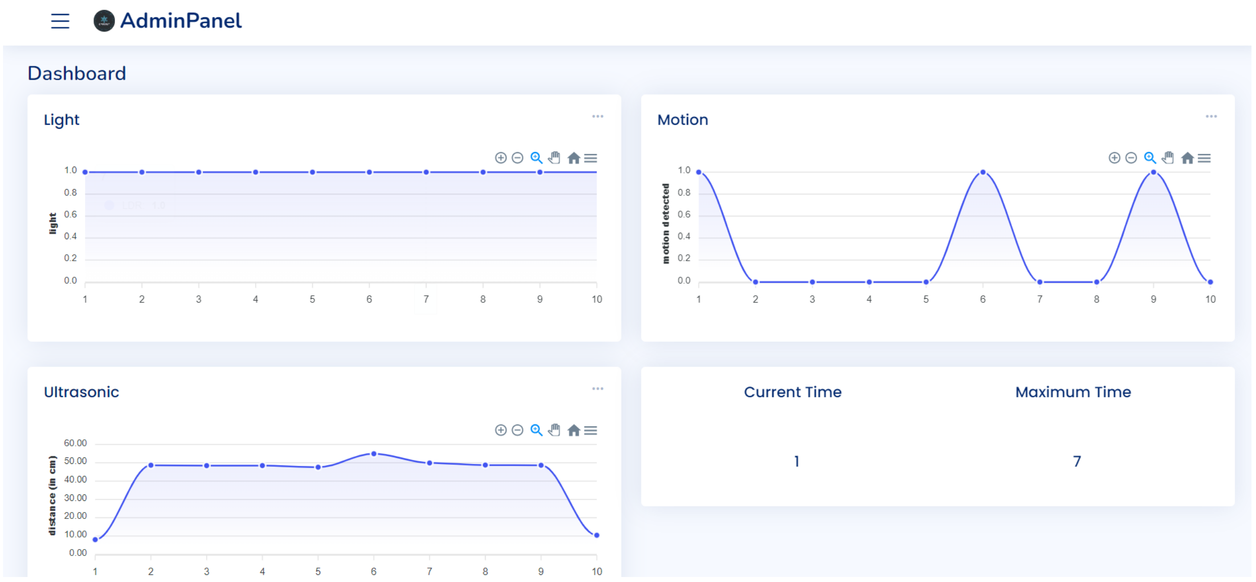
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta content="width=device-width, initial-scale=1.0" name="viewport" />

    <title>{{title}}</title>
    <meta content="" name="description" />
    <meta content="" name="keywords" />

    <!-- Favicons -->
    <link href="static/assets/img/favicon.png" rel="icon" />
    <link
      href="static/assets/img/apple-touch-icon.png"
      rel="apple-touch-icon"
    />
    <!-- Google Fonts -->
    <link href="https://fonts.gstatic.com" rel="preconnect" />
    <link

href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i|
Nunito:300,300i,400,400i,600,600i,700,700i|Poppins:300,300i,400,400i,500,500i,600,600i,700,7
00i"
      rel="stylesheet"
    />

    <link
      href="static/assets/vendor/bootstrap/css/bootstrap.min.css"
      rel="stylesheet"
    />
    <link
      href="static/assets/vendor/bootstrap-icons/bootstrap-icons.css"
      rel="stylesheet"
    />
    <link
      href="static/assets/vendor/boxicons/css/boxicons.min.css"
      rel="stylesheet"
    />
    <link href="static/assets/vendor/quill/quill.snow.css" rel="stylesheet" />
    <link href="static/assets/vendor/quill/quill.bubble.css" rel="stylesheet" />
    <link
      href="static/assets/vendor/remixicon/remixicon.css"
      rel="stylesheet"
    />
    <link
      href="static/assets/vendor/simple-datatables/style.css"
      rel="stylesheet"
    />

    <link href="static/assets/css/style.css" rel="stylesheet" />

    <script>
      var pirData = [];
      var ultrasonicData = [];
      var motionData = [];
      var pirChart;
      var ultrasonicChart;
      var motionChart;
    </script>
  </head>

  <body class="toggle-sidebar">
    <header id="header" class="header fixed-top d-flex align-items-center">
      <div class="d-flex align-items-center justify-content-between">
        <i class="bi bi-list toggle-sidebar-btn m-4"></i>
        <a href="index.html" class="logo d-flex align-items-center">
          <img
            src="static/assets/img/cfrost-logo.png"
            alt=""
            style="border-radius: 50%"
          />
          <span class="d-none d-lg-block">AdminPanel</span>
        </a>
      </div>

    </header>

    <main id="main" class="main">
      <div class="pagetitle">
        <h1>Dashboard</h1>
      </div>

      <section class="section dashboard">
        <div class="row">
          <div class="col-lg-6">
            <div class="row">

              <div class="col-12">
                <div class="card">
                  <div class="filter">
                    <a class="icon" href="#" data-bs-toggle="dropdown"
                      ><i class="bi bi-three-dots"></i
                    ></a>
                    <ul
          class="dropdown-menu dropdown-menu-end dropdown-menu-arrow"
        >
          <li class="dropdown-header text-start">
            <h6>Filter</h6>
          </li>

          <li>
            <a class="dropdown-item" href="#" id="phDay">Today</a>
          </li>
          <li>
            <a class="dropdown-item" href="#" id="phWeek"
              >This Week</a
            >
          </li>
          <li>
            <a class="dropdown-item" href="#" id="phMonth"
              >This Month</a
            >
          </li>
        </ul>
      </div>

      <div class="card-body">
        <h5 class="card-title">Light</h5>

        <div id="pirChart"></div>

        <script></script>
      </div>
    </div>
  </div>
  <div class="col-12">
    <div class="card">
      <div class="filter">
        <a class="icon" href="#" data-bs-toggle="dropdown"
          ><i class="bi bi-three-dots"></i
        ></a>
        <ul
          class="dropdown-menu dropdown-menu-end dropdown-menu-arrow"
        >
```

```
            <li class="dropdown-header text-start">
              <h6>Filter</h6>
            </li>

            <li>
              <a class="dropdown-item" href="#" id="phDay">Today</a>
            </li>
            <li>
              <a class="dropdown-item" href="#" id="phWeek"
                >This Week</a
              >
            </li>
            <li>
              <a class="dropdown-item" href="#" id="phMonth"
                >This Month</a
              >
            </li>
          </ul>
        </div>

        <div class="card-body">
          <h5 class="card-title">Ultrasonic</h5>

          <div id="ultrasonicChart"></div>

          <script></script>
        </div>
      </div>
    </div>
  </div>
</div>

<div class="col-lg-6">
  <div class="col-12">
    <div class="card">
      <div class="filter">
        <a class="icon" href="#" data-bs-toggle="dropdown"
          ><i class="bi bi-three-dots"></i
        ></a>
        <ul
        class="dropdown-menu dropdown-menu-end dropdown-menu-arrow"
      >
        <li class="dropdown-header text-start">
          <h6>Filter</h6>
        </li>

        <li>
          <a class="dropdown-item" href="#" id="turbidityDay"
            >Today</a
          >
        </li>
        <li>
          <a class="dropdown-item" href="#" id="turbidityWeek"
            >This Week</a
          >
        </li>
        <li>
          <a class="dropdown-item" href="#" id="turbidityMonth"
            >This Month</a
          >
        </li>
      </ul>
    </div>

    <div class="card-body">
      <h5 class="card-title">Motion</h5>

      <div id="motionChart"></div>

      <script></script>
    </div>
  </div>
</div>
<div class="col-12">
  <div class="card">

    <div class="card-body d-flex justify-content-around">
      <h5 class="card-title">Current Time</h5>
      <h5 class="card-title">Maximum Time</h5>
```

```
            <script></script>
          </div>
          <div class="card-body d-flex justify-content-around">
            <h5 class="card-title"  id="last"></h5>
            <h5 class="card-title" id="maxi"></h5>

            <script></script>
          </div>
        </div>
      </div>
    </div>
  </div>
 </section>
</main>
<script src="static/assets/vendor/apexcharts/apexcharts.min.js"></script>
<script src="static/assets/vendor/bootstrap/js/bootstrap.bundle.min.js"></script>
<script src="static/assets/vendor/chart.js/chart.umd.js"></script>
<script src="static/assets/vendor/echarts/echarts.min.js"></script>
<script src="static/assets/vendor/quill/quill.min.js"></script>
<script src="static/assets/vendor/simple-datatables/simple-datatables.js"></script>
<script src="static/assets/vendor/tinymce/tinymce.min.js"></script>
<script src="static/assets/vendor/php-email-form/validate.js"></script>

<script src="static/assets/js/main.js"></script>
<script>
 var pirData = [];
 var ultrasonicData = [];
 var motionData = [];
 var pirChart;
 var ultrasonicChart;
 var motionChart;

 document.addEventListener("DOMContentLoaded", () => {
   ultrasonicChart = new ApexCharts(
     document.querySelector("#ultrasonicChart"),
     {
       series: [
         {
           name: "Ultrasonic",
      data: ultrasonicData,
    },
  ],
  chart: {
    height: 200,
    type: "area",
    toolbar: {
      show: true,
    },
  },
  markers: {
    size: 4,
  },
  colors: ["#4154f1"],
  fill: {
    type: "gradient",
    gradient: {
      shadeIntensity: 1,
      opacityFrom: 0.3,
      opacityTo: 0.4,
      stops: [0, 90, 100],
    },
  },
  dataLabels: {
    enabled: false,
  },
  stroke: {
    curve: "smooth",
    width: 2,
  },
  xaxis: {
    type: "number",
    categories: ultrasonicData,
  },
  yaxis:{
    title: {
          text: 'distance (in cm)'
    },
  }
}
```

```
);
motionChart = new ApexCharts(document.querySelector("#motionChart"), {
  series: [
    {
      name: "Pir",
      data: motionData,
    },
  ],
  chart: {
    height: 200,
    type: "area",
    toolbar: {
      show: true,
    },
  },
  markers: {
    size: 4,
  },
  colors: ["#4154f1"],
  fill: {
    type: "gradient",
    gradient: {
      shadeIntensity: 1,
      opacityFrom: 0.3,
      opacityTo: 0.4,
      stops: [0, 90, 100],
    },
  },
  dataLabels: {
    enabled: false,
  },
  stroke: {
    curve: "smooth",
    width: 2,
  },
  xaxis: {
    type: "number",
    categories: motionData,
  },
  yaxis:{
      title: {
            text: 'motion detected'
      },
    }
});

pirChart = new ApexCharts(document.querySelector("#pirChart"), {
  series: [
    {
      name: "LDR",
      data: pirData,
    },
  ],
  chart: {
    height: 200,
    type: "area",
    toolbar: {
      show: true,
    },
  },
  markers: {
    size: 4,
  },
  colors: ["#4154f1"],
  fill: {
    type: "gradient",
    gradient: {
      shadeIntensity: 1,
      opacityFrom: 0.3,
      opacityTo: 0.4,
      stops: [0, 90, 100],
    },
  },
  dataLabels: {
    enabled: false,
  },
  stroke: {
    curve: "smooth",
    width: 2,
  },
    xaxis: {
      type: "number",
      categories: pirData,
    },
    yaxis:{
        title: {
              text: 'light'
        },
      }
  });
  pirChart.render();
  ultrasonicChart.render();
  motionChart.render();
});

setInterval(() => {
  fetch("http://127.0.0.1:5000/api/getData")
    .then((response) => response.json())
    .then((data) => {
      console.log(data);
      document.getElementById("last").innerHTML = `${data.last}`
      document.getElementById("maxi").innerHTML = `${data.maxi}`
      pirData = data.pir;
      ultrasonicData = data.ultrasonic;
      motionData = data.motion;
      pirChart.updateSeries([
        {
          data: pirData,
        },
      ]);
      ultrasonicChart.updateSeries([
        {
          data: ultrasonicData,
        },
      ]);
      motionChart.updateSeries([
        {
          data: motionData,
        },
      ]);
```

```
          })
          .catch((error) => console.log(error));
      }, 3000);
    </script>
  </body>
</html>
```



Current Time 1

Maximum Time 7

Screenshot of the Application

Chapter 4

Conclusion and Future Scope

4.1 Conclusion

In conclusion, our IoT-based Smart Parking System, which adapts parking area lighting based on vehicle headlight status, signifies a pivotal stride towards a more energy-efficient, user-centric, and data-informed urban environment. With its ability to reduce operational costs, enhance the user experience, optimize parking space usage, enable data-driven decision-making, and integrate with broader smart city solutions, it embodies the promise of a smarter and sustainable urban future.

The implementation of our Smart Parking Light System significantly transforms day-to-day life in urban settings. By efficiently managing parking area lighting, it leads to a host of positive outcomes. Firstly, it promotes energy conservation and sustainability by automatically turning off lights when they are not required, thereby reducing energy consumption and lowering costs for parking facility operators. Moreover, it enhances the safety and comfort of drivers and passengers as well-lit parking spaces ensure a secure and pleasant experience, particularly in dimly lit or underground parking areas. Furthermore, it streamlines the parking process, saving time for drivers by reducing the search for available parking spaces, which, in turn, minimizes traffic congestion. Additionally, the data-driven insights obtained from the system play a crucial role in urban planning, leading to more efficient parking facility layouts and dynamic pricing strategies, ultimately benefiting residents by making their daily commute more seamless and convenient. Lastly, the system's scalability and integration potential offer an interconnected urban infrastructure that responds to residents' needs, ultimately contributing to a higher quality of life in cities.

4.2 Future Scope

1. **Advanced Sensor Integration:** Consider incorporating more advanced sensors such as lidar, cameras, or advanced image processing technology to enhance vehicle detection and parking space occupancy accuracy. These sensors can provide real-time, high-resolution data for improved system performance.

2. **Data Connectivity:** Expand the system's connectivity capabilities by integrating it with cloud-based platforms and IoT infrastructure. This enables remote monitoring, data analysis, and management, as well as the potential for real-time updates and reporting.

3. **Sustainable Practices:** Implement energy-efficient lighting technologies, such as smart LED lighting, and explore renewable energy sources like solar panels to power the system. This reduces operational costs and contributes to sustainability.

References

[1] Abrar Fahim and Mehedi Hasan, "Smart parking systems: comprehensive review based on various aspects", 2021, - “https://www.sciencedirect.com/science/article/pii/S2405844021011531”

[2] Aashish Joshi and Arni Tharakaram Hariram, "Smart Car Parking System" New Horizon College of Engineering, Bangalore, Karnataka, India, 2020.