**Problem statement: Design and suggest automation implementations for a room containing an AC, fan, lights, windows, electrical** **appliances, routers, and a fire alert system.**

**Goal:** Enhance efficiency and convenience, reduce carbon footprint, and increase safety through automation

**Abstract:**

This project outlines the comprehensive design and implementation of an automation system for a room, integrating various smart devices for enhanced comfort, safety, and energy efficiency. The system encompasses temperature control using sensors and actuators, motion-triggered lighting and window automation, a fire alert system, and remote control through a mobile app. The central controller, a Raspberry Pi, facilitates communication among devices via MQTT. Security measures, emergency protocols, and energy management strategies are incorporated to ensure a robust and user-friendly automation solution. The design adheres to a step-by-step approach, emphasizing scalability, regulatory compliance, and regular maintenance. A practical example illustrates the application of this design approach in creating a smart room environment that meets user requirements and aligns with contemporary automation standards.

**Design Approach Implementation:**

**User Requirements:**

* Temperature control for comfort.
* Motion-triggered lighting and window control for energy efficiency.
* Fire alert system for safety.
* Remote control and monitoring through a mobile app.
* Energy management features to optimize electrical appliance usage.

**Functional Requirements:**

* Temperature control
* Motion-triggered lighting
* Window control
* Fire alert
* Mobile app control
* Energy management

**System Architecture:**

* **Central Controller:** Raspberry Pi with Wi-Fi capabilities.
* **Communication Protocol:** MQTT for device communication.

**Device Selection:**

* **Sensors:** DHT22 for temperature/humidity, PIR for motion, window/door contact sensors, MQ-2 for fire/smoke.
* **Actuators:** Relay modules for lights, fan, and AC, motorized window control.
* **Smart Plugs:** for electrical appliances.

**Integration and Communication:**

* Connecting sensors and actuators to the Raspberry Pi using GPIO pins.
* Implementing MQTT for communication between the devices.

**Automation Logic:**

* Using DHT22 readings to control the AC (turn on if temperature > 25°C, turn off if < 22°C).
* Activate lights and open/close windows based on motion sensor readings.
* Trigger fire alerts and emergency shutdown if MQ-2 detects smoke or fire.

**User Interface Design:**

A mobile app displaying real-time temperature, motion, and fire alerts with controls for lights, AC and window operations

**Security Measures:**

* User authentication in the mobile app.
* Encrypted MQTT communication.

**Emergency Protocols:**

* Fail-safes to ensure fire alerts are implemented.
* Push notifications will be sent to the user in case of emergencies.

**Energy Management:**

* Scheduled smart plugs to turn off during certain hours.
* Provided insights on energy consumption through the mobile app.

**Integration with External Systems:**

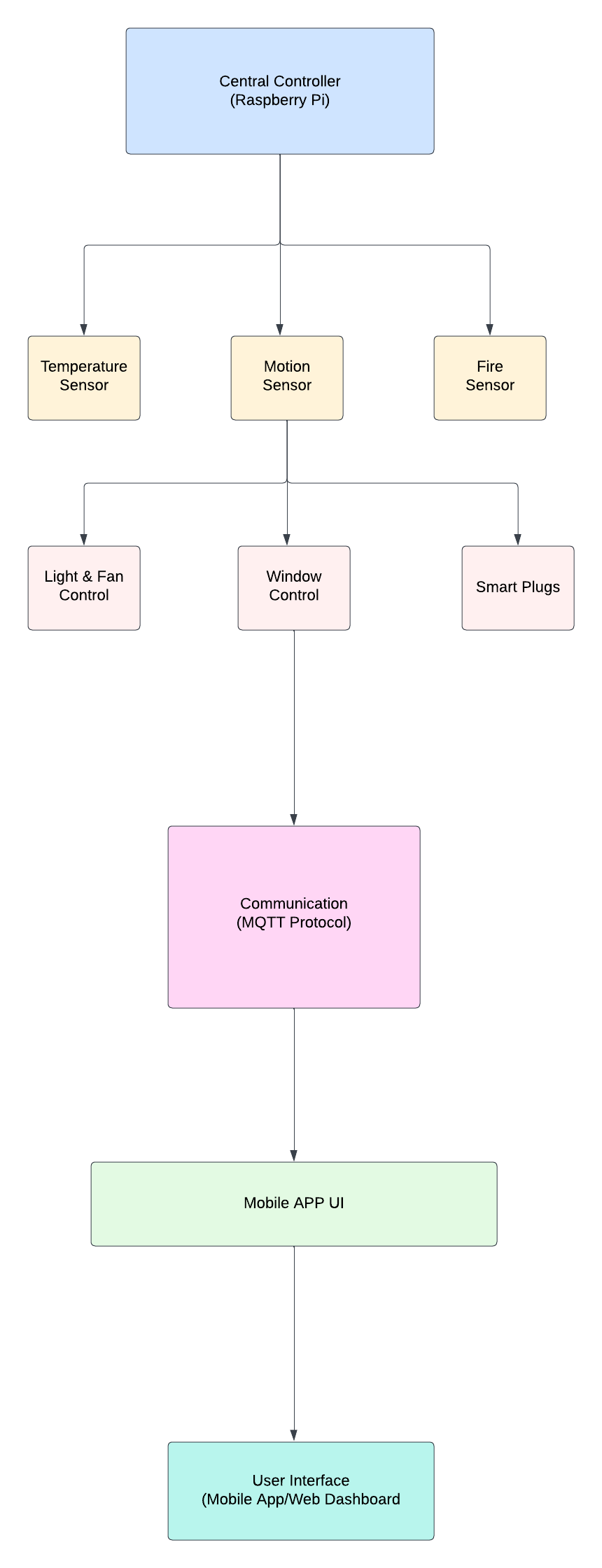
* A weather API to adjust temperature settings is integrated.
* Connected to a utility monitoring system for energy usage insights.

**Implementation:**

* Install sensors, actuators, and the Raspberry Pi in the room.
* Set up the MQTT broker and configure devices to communicate.

.

**Block Diagrams and Schematics:**

****

**Component Specifications:**

**Central Controller (Raspberry Pi):**

**Model:** Raspberry Pi 4

**Processor:** Quad-core ARM Cortex-A72

**Memory:** 4GB RAM

**Connectivity:** Wi-Fi, Bluetooth

**GPIO Pins:** Sufficient for connecting sensors and actuators

**Operating System:** Raspbian or a compatible OS



*Quad-core ARM Cortex-A72*

**Sensors:**

**Temperature and Humidity Sensor (DHT22):**

* + - Operating Range: -40°C to 80°C, 0-100% humidity
    - Accuracy: ±0.5°C temperature, ±2% humidity



*DHT22*

**Motion Sensor (Passive Infrared - PIR):**

* Detection Range: Adjustable, typically up to 7 meters
* Detection Angle: 120 degrees



*PIR Sensor*

**Window/Door Contact Sensors:**

* Type: Magnetic Reed Switch
* Operating Gap: Adjustable, depending on the specific sensor model



*Window Contact Sensor*

**Fire/Smoke Sensor (MQ-2):**

* Detection Gases: Methane, Butane, LPG, Smoke, etc.
* Sensitivity: Adjustable



*MQ-2*

**Actuators:**

**Relay Modules (for Lights, Fan, AC):**

* Number of Channels: Sufficient for controlling lights, fan, and AC
* Load Capacity: Suitable for connected devices

**Motorized Window Control:**

* Type: DC Motor or Servo Motor
* Control Interface: GPIO or PWM control



*Servo motor*

**Smart Plugs:**

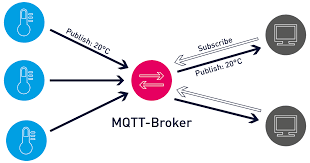
* Type: Wi-Fi Smart Plugs
* Compatibility: Works with common smart home platforms (e.g., Amazon Alexa, Google Assistant)

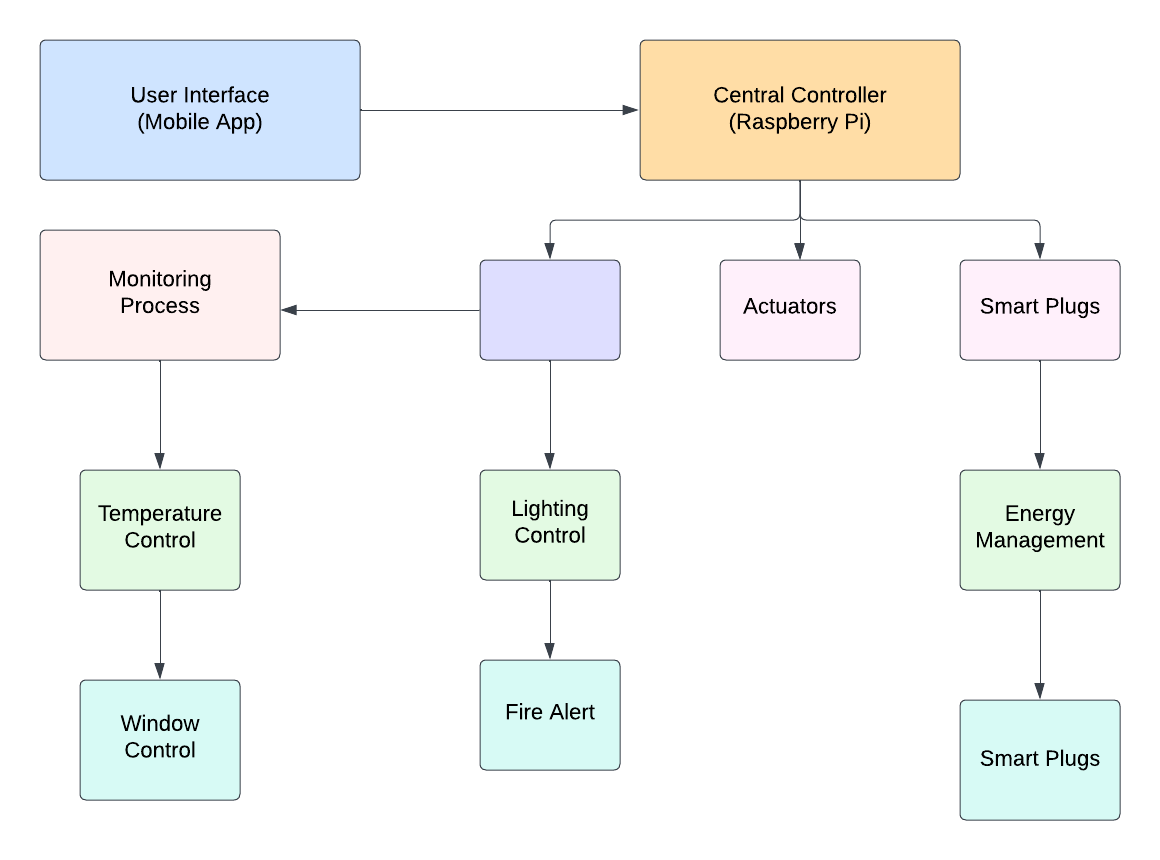
*Wifi Smart plug Amazon Alexa*

**Communication:**

* Communication Protocol: MQTT (Message Queuing Telemetry Transport)
* Wireless Protocol: Wi-Fi for communication between the central controller and devices



**Data Flow Diagram:**

****

**Execution Approach:**

**1. Define Project Scope and Objectives:**

* Scope: Implement an automation system for a room with temperature control, motion-triggered lighting, window automation, fire alerts, and remote control via a mobile app.
* Objectives: Enhance comfort, safety, and energy efficiency through smart technologies.

**2. Requirements Gathering:**

* Collect user requirements through interviews and surveys.
* Prioritize features based on user needs.

**3. System Architecture Design:**

* Choose Raspberry Pi as the central controller.
* Design communication using MQTT.
* Specify sensors (DHT22, PIR, contact sensors, MQ-2) and actuators (relays, motorized window control).

**4. Component Selection and Procurement:**

* Order Raspberry Pi, sensors, actuators, and other components.
* Verify compatibility and purchase necessary accessories.

**5. Software Development:**

* Develop software for the Raspberry Pi using Python.
* Implement logic for temperature control, motion-triggered lighting, window automation, and fire alerts.
* Set up MQTT communication between devices.

**6. Mobile App Development:**

* Develop a mobile app (iOS/Android) using a cross-platform framework (e.g., React Native).
* Implement features for real-time monitoring, device control, and emergency notifications.

**7. Integration and Testing:**

* Connect sensors and actuators to the Raspberry Pi.
* Test each component's functionality individually.
* Conduct integration testing to ensure proper communication.

**8. User Interface Testing:**

* Test the mobile app for usability and responsiveness.
* Verify synchronization with the central controller.

**9. Security Implementation:**

* Implement user authentication in the mobile app.
* Enable encrypted communication using SSL/TLS.

**Special Features:**

**Temperature Control:**

* **Smart Regulation**: The system intelligently regulates the temperature based on real-time data from the DHT22 sensor, ensuring optimal comfort.

**Motion-Triggered Lighting:**

* **Energy Efficiency:** Lights are activated only when motion is detected, reducing energy consumption and promoting sustainability.
* **User Convenience:** Provides hands-free lighting in response to user presence.

**Window Automation:**

* **Adaptive Control:** Automated window control based on environmental factors such as temperature, ensuring a comfortable indoor environment.
* **Energy Savings:** Helps in natural ventilation and utilizes external conditions for energy efficiency.

**Fire Alert System:**

* **Quick Response:** Immediate detection of smoke or fire triggers an emergency protocol, ensuring rapid response to potential hazards.
* **User Notification:** Notifies users through the mobile app, enhancing safety measures.

**Mobile App Control:**

* **Remote Accessibility:** Users can control and monitor the system remotely via the mobile app, providing convenience and flexibility.
* **Real-Time Data**: Displays real-time data from sensors, allowing users to stay informed about the room environment.

**Energy Management:**

* **Smart Plugs Scheduling:** Enables users to schedule the operation of electrical appliances for energy efficiency and cost savings.
* **Usage Insights:** Provides insights into energy consumption patterns through the mobile app.

**Security Measures:**

* **User Authentication:** Ensures secure access to the system through the mobile app.
* **Encrypted Communication:** Utilizes encryption protocols for secure communication between devices.

**Advantages:**

1. **Enhanced Comfort:**

The system optimizes temperature, lighting, and window conditions to enhance the overall comfort of the room.

1. **Improved Energy Efficiency:**

Energy-efficient features, such as motion-triggered lighting and smart plug scheduling, contribute to reduced energy consumption.

1. **User-Friendly Interface:**

The mobile app provides an intuitive and user-friendly interface for convenient control and monitoring.

1. **Safety and Security:**

The fire alert system and emergency protocols enhance safety measures, providing users with a secure environment.

1. **Remote Accessibility:**

Users can remotely control and monitor the automation system, providing convenience and flexibility.

1. **Scalability:**

The system is designed to be easily scalable, allowing for the integration of additional devices and features in the future.

1. **Adaptive Automation:**

The automation system adapts to changing environmental conditions, promoting efficiency and resource optimization.

1. **Data-Driven Insights:**

Real-time data and energy consumption insights empower users with information for better decision-making.

1. **Intelligent Integration:**

Integration with external systems, such as weather forecasts and utility monitoring, allows for intelligent decision-making.

1. **Continuous Improvement:**

The design includes mechanisms for collecting user feedback and implementing iterative improvements for sustained efficiency and relevance.

**Justification for each automation choice:**

**1. Central Controller: Raspberry Pi**

**Justification:**

* **Versatility:** Raspberry Pi is a versatile and cost-effective platform suitable for small-scale automation projects.
* **Community Support:** Wide community support and extensive documentation make it easier to find solutions and troubleshoot issues.
* **GPIO Support:** Raspberry Pi's GPIO pins facilitate easy interfacing with sensors and actuators.

**2. Communication Protocol: MQTT (Message Queuing Telemetry Transport)**

**Justification:**

* **Lightweight:** MQTT is a lightweight protocol, reducing data overhead and ensuring efficient communication between devices.
* **Reliability**: It offers reliable message delivery, making it suitable for scenarios where data integrity is crucial.
* **Publish-Subscribe Model**: The publish-subscribe model allows for flexible and scalable communication between devices.

**3. Sensors: DHT22, PIR, Window/Door Contacts, MQ-2**

**Justification:**

* **Accuracy:** Chosen sensors have good accuracy for temperature, motion, and fire detection.
* **Versatility:** The combination of sensors provides a comprehensive view of the room environment.
* **Cost-Effectiveness:** These sensors offer a balance between performance and cost, suitable for a home automation project.

**4. Actuators: Relay Modules, Motorized Window Control**

**Justification:**

* **Compatibility:** Relay modules are compatible with various devices, allowing control of lights, fans, and the AC.
* **Precision Control:** Motorized window control enables precise and automated adjustments based on environmental conditions.

**5. Smart Plugs for Electrical Appliances**

**Justification:**

* **Ease of Integration:** Smart plugs are easy to integrate and provide a convenient way to control electrical appliances.
* **Energy Management:** Smart plugs allow for scheduling, contributing to energy management and cost savings.

**6. Mobile App Development using React Native**

**Justification:**

* **Cross-Platform Compatibility:** React Native enables the development of a single codebase for both iOS and Android platforms, reducing development effort.
* **Community Support:** React Native has a large and active community, ensuring ongoing support and updates.
* **User Interface:** React Native allows the creation of a visually appealing and responsive user interface.

**7. Security Measures: User Authentication, Encrypted Communication**

**Justification:**

* **User Privacy:** User authentication ensures that only authorized users can control the automation system, enhancing privacy and security.
* **Data Integrity:** Encrypted communication safeguards data integrity and protects against potential security threats.

**8**. **Emergency Protocols: Fire Alerts and Emergency Shutdown**

**Justification:**

* **Safety:** Immediate fire alerts and emergency shutdown mechanisms prioritize user safety and property protection.
* **Rapid Response:** Automation enables a rapid response to potential hazards, minimizing damage.

**9. Energy Management: Smart Plugs Scheduling**

**Justification:**

* **Efficiency:** Scheduling smart plugs helps optimize energy consumption by turning off appliances during specific hours.
* **Cost Savings:** Energy management contributes to cost savings over time.

**10. Integration with External Systems: Weather Forecasts, Utility Monitoring**

**Justification:**

* **Adaptability:** Integration with weather forecasts allows the system to adapt based on external conditions, enhancing comfort and energy efficiency.
* **Insights:** Utility monitoring provides insights into energy consumption patterns, supporting informed decision-making.

**11. Scalability and Modularity in Design**

**Justification:**

* **Future-Proofing:** A scalable and modular design allows for easy integration of additional devices or features in the future.
* **Flexibility:** Modularity supports flexibility in adapting the system to evolving user needs and technological advancements.

**12. Continuous Improvement Mechanisms**

**Justification:**

* **User-Centric Design:** Collecting user feedback and implementing continuous improvements ensure that the system remains user-centric.
* **Adaptation:** Regular updates based on feedback and technological advancements keep the system relevant and efficient.

**Potential Benefits and Impacts:**

**1. Enhanced Comfort:**

* **Benefit:** Users experience a more comfortable living environment with automated temperature control and adaptive features.
* **Impact:** Improved comfort contributes to a better quality of life and user satisfaction.

**2. Energy Efficiency:**

* **Benefit:** Automation features like motion-triggered lighting, smart plugs scheduling, and adaptive window control lead to reduced energy consumption.
* **Impact:** Lower energy usage contributes to cost savings and a smaller environmental footprint.

**3. Cost Savings:**

* **Benefit:** Efficient energy management and optimized appliance usage result in reduced utility bills.
* **Impact:** Users experience financial savings over time.

**4. User Convenience:**

* **Benefit:** Remote control via the mobile app provides users with convenient access to system controls.
* **Impact:** Increased convenience and flexibility in managing the room environment.

**5. Safety and Security:**

* **Benefit:** Fire alerts and emergency shutdown mechanisms enhance safety measures.
* **Impact:** Rapid response to potential hazards minimizes risks and property damage.

**6. Data-Driven Decision Making:**

* **Benefit:** Real-time data and insights into energy consumption patterns empower users to make informed decisions.
* **Impact:** Users can optimize their living environment based on data-driven insights.

**7. Adaptive Automation:**

* **Benefit:** Adaptive features, such as window automation based on environmental conditions, improve overall system efficiency.
* **Impact:** The system adapts to changing conditions, providing an intelligent and responsive automation experience.

**8. Cross-Platform Accessibility:**

* **Benefit:** The mobile app's cross-platform compatibility (iOS/Android) ensures accessibility for a wide range of users.
* **Impact:** Increased user adoption and inclusivity.

**9. Scalability and Future-Proofing:**

* **Benefit:** A modular and scalable design allows for easy integration of new devices and features in the future.
* **Impact:** The system remains adaptable to evolving user needs and technological advancements.

**10. Continuous Improvement:**

* **Benefit:** Collection of user feedback and continuous improvements ensure the system stays user-centric and relevant.
* **Impact:** Ongoing enhancements enhance user satisfaction and system performance.

**11. Environmental Impact:**

* **Benefit:** Lower energy consumption and adaptive features contribute to a reduced environmental impact.
* **Impact:** Users contribute to sustainability and environmental conservation.

**12. Safety in Emergency Situations:**

* **Benefit:** Fire alerts and emergency shutdown mechanisms prioritize safety in critical situations.
* **Impact:** Enhanced safety measures protect lives and property.

**13. User Empowerment:**

* **Benefit:** Users have control over their living environment and receive insights to make informed decisions.
* **Impact:** Increased user empowerment and engagement.

**14. Adoption of Smart Technologies:**

* **Benefit:** Users embrace smart technologies that simplify their daily lives.
* **Impact:** Encourages the adoption of home automation and smart living practices.

**15. Regulatory Compliance:**

* **Benefit:** Adhering to safety standards and regulations ensures the system meets legal requirements.
* **Impact:** Users can trust in the system's safety and compliance with industry standards.

**Codes:**

**Mqtt.py**

import RPi.GPIO as GPIO

import dht22

import time

import paho.mqtt.client as mqtt

# Initialize GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

# Initialize DHT22 sensor

sensor = dht22.DHT22(pin=4)

# GPIO pins for actuators

light\_relay\_pin = 17

fan\_relay\_pin = 18

# Set up MQTT parameters

mqtt\_broker = "your\_mqtt\_broker\_address"

mqtt\_port = 1883

mqtt\_topic\_temp = "room/temperature"

mqtt\_topic\_motion = "room/motion"

mqtt\_topic\_light = "room/light"

# Set up GPIO pins

GPIO.setup(light\_relay\_pin, GPIO.OUT)

GPIO.setup(fan\_relay\_pin, GPIO.OUT)

# MQTT on\_connect callback

def on\_connect(client, userdata, flags, rc):

print("Connected with result code "+str(rc))

client.subscribe(mqtt\_topic\_temp)

client.subscribe(mqtt\_topic\_motion)

# MQTT on\_message callback

def on\_message(client, userdata, msg):

if msg.topic == mqtt\_topic\_temp:

process\_temperature(float(msg.payload))

elif msg.topic == mqtt\_topic\_motion:

process\_motion(int(msg.payload))

# Function to control lights based on motion

def process\_motion(motion\_status):

if motion\_status == 1:

GPIO.output(light\_relay\_pin, GPIO.HIGH) # Turn on lights

else:

GPIO.output(light\_relay\_pin, GPIO.LOW) # Turn off lights

# Function to adjust temperature control

def process\_temperature(temperature):

# Add logic to control temperature (e.g., turn on fan or AC based on temperature)

# Set up MQTT client

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

# Connect to MQTT broker

client.connect(mqtt\_broker, mqtt\_port, 60)

# Start MQTT loop

client.loop\_start()

try:

while True:

# Read temperature from DHT22 sensor

result = sensor.read()

if result.is\_valid():

temperature = result.temperature

# Publish temperature to MQTT broker

client.publish(mqtt\_topic\_temp, temperature)

time.sleep(5) # Adjust the sleep duration based on your requirements

except KeyboardInterrupt:

print("Program terminated by user.")

GPIO.cleanup()

client.disconnect()

# Import necessary libraries

import RPi.GPIO as GPIO

import dht22

import time

import paho.mqtt.client as mqtt

# Initialize GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

# Initialize DHT22 sensor

sensor = dht22.DHT22(pin=4)

# GPIO pins for actuators

light\_relay\_pin = 17

fan\_relay\_pin = 18

# Set up MQTT parameters

mqtt\_broker = "your\_mqtt\_broker\_address"

mqtt\_port = 1883

mqtt\_topic\_temp = "room/temperature"

mqtt\_topic\_motion = "room/motion"

# Set up GPIO pins

GPIO.setup(light\_relay\_pin, GPIO.OUT)

GPIO.setup(fan\_relay\_pin, GPIO.OUT)

# MQTT on\_connect callback

def on\_connect(client, userdata, flags, rc):

print("Connected with result code " + str(rc))

client.subscribe(mqtt\_topic\_temp)

client.subscribe(mqtt\_topic\_motion)

# MQTT on\_message callback

def on\_message(client, userdata, msg):

if msg.topic == mqtt\_topic\_temp:

process\_temperature(float(msg.payload))

elif msg.topic == mqtt\_topic\_motion:

process\_motion(int(msg.payload))

# Function to control lights based on motion

def process\_motion(motion\_status):

if motion\_status == 1:

GPIO.output(light\_relay\_pin, GPIO.HIGH) # Turn on lights

else:

GPIO.output(light\_relay\_pin, GPIO.LOW) # Turn off lights

# Function to adjust temperature control

def process\_temperature(temperature):

# Add logic to control temperature (e.g., turn on fan or AC based on temperature)

# Set up MQTT client

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

# Connect to MQTT broker

client.connect(mqtt\_broker, mqtt\_port, 60)

# Start MQTT loop

client.loop\_start()

try:

while True:

# Read temperature from DHT22 sensor

result = sensor.read()

if result.is\_valid():

temperature = result.temperature

# Publish temperature to MQTT broker

client.publish(mqtt\_topic\_temp, temperature)

time.sleep(5) # Adjust the sleep duration based on your requirements

except KeyboardInterrupt:

print("Program terminated by user.")

GPIO.cleanup()

client.disconnect()# Import necessary libraries

**control.py**

import RPi.GPIO as GPIO

import dht22

import time

import paho.mqtt.client as mqtt

# Initialize GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

# Initialize DHT22 sensor

sensor = dht22.DHT22(pin=4)

# GPIO pins for actuators

light\_relay\_pin = 17

fan\_relay\_pin = 18

# Set up MQTT parameters

mqtt\_broker = "your\_mqtt\_broker\_address"

mqtt\_port = 1883

mqtt\_topic\_temp = "room/temperature"

mqtt\_topic\_motion = "room/motion"

# Set up GPIO pins

GPIO.setup(light\_relay\_pin, GPIO.OUT)

GPIO.setup(fan\_relay\_pin, GPIO.OUT)

# MQTT on\_connect callback

def on\_connect(client, userdata, flags, rc):

print("Connected with result code " + str(rc))

client.subscribe(mqtt\_topic\_temp)

client.subscribe(mqtt\_topic\_motion)

# MQTT on\_message callback

def on\_message(client, userdata, msg):

if msg.topic == mqtt\_topic\_temp:

process\_temperature(float(msg.payload))

elif msg.topic == mqtt\_topic\_motion:

process\_motion(int(msg.payload))

# Function to control lights based on motion

def process\_motion(motion\_status):

if motion\_status == 1:

GPIO.output(light\_relay\_pin, GPIO.HIGH) # Turn on lights

else:

GPIO.output(light\_relay\_pin, GPIO.LOW) # Turn off lights

# Function to adjust temperature control

def process\_temperature(temperature):

# Add logic to control temperature (e.g., turn on fan or AC based on temperature)

# Set up MQTT client

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

# Connect to MQTT broker

client.connect(mqtt\_broker, mqtt\_port, 60)

# Start MQTT loop

client.loop\_start()

try:

while True:

# Read temperature from DHT22 sensor

result = sensor.read()

if result.is\_valid():

temperature = result.temperature

# Publish temperature to MQTT broker

client.publish(mqtt\_topic\_temp, temperature)

time.sleep(5) # Adjust the sleep duration based on your requirements

except KeyboardInterrupt:

print("Program terminated by user.")

GPIO.cleanup()

client.disconnect()

**script.js**

import React, { useState, useEffect } from 'react';

import { View, Text, Button } from 'react-native';

import Mqtt from 'react-native-mqtt';

const App = () => {

const [temperature, setTemperature] = useState(0);

const [motionStatus, setMotionStatus] = useState(0);

useEffect(() => {

// Set up MQTT client

const client = new Mqtt.Client({ uri: 'mqtt://your\_mqtt\_broker\_address:1883', clientId: 'clientId' });

client.connect();

// Subscribe to MQTT topics

client.on('connect', () => {

client.subscribe('room/temperature');

client.subscribe('room/motion');

});

// Handle incoming MQTT messages

client.on('message', (topic, message) => {

if (topic === 'room/temperature') {

setTemperature(parseFloat(message.toString()));

} else if (topic === 'room/motion') {

setMotionStatus(parseInt(message.toString()));

}

});

// Clean up on component unmount

return () => {

client.disconnect();

};

}, []);

return (

<View>

<Text>Temperature: {temperature}°C</Text>

<Text>Motion Status: {motionStatus === 1 ? 'Motion Detected' : 'No Motion'}</Text>

<Button title="Toggle Lights" onPress={() => {/\* Add logic to publish MQTT message to control lights \*/}} />

</View>

);

};

export default App; import React, { useState, useEffect } from 'react';

import { View, Text, Button } from 'react-native';

import Mqtt from 'react-native-mqtt';

const App = () => {

const [temperature, setTemperature] = useState(0);

const [motionStatus, setMotionStatus] = useState(0);

useEffect(() => {

// Set up MQTT client

const client = new Mqtt.Client({ uri: 'mqtt://your\_mqtt\_broker\_address:1883', clientId: 'clientId' });

client.connect();

// Subscribe to MQTT topics

client.on('connect', () => {

client.subscribe('room/temperature');

client.subscribe('room/motion');

});

// Handle incoming MQTT messages

client.on('message', (topic, message) => {

if (topic === 'room/temperature') {

setTemperature(parseFloat(message.toString()));

} else if (topic === 'room/motion') {

setMotionStatus(parseInt(message.toString()));

}

});

// Clean up on component unmount

return () => {

client.disconnect();

};

}, []);

return (

<View>

<Text>Temperature: {temperature}°C</Text>

<Text>Motion Status: {motionStatus === 1 ? 'Motion Detected' : 'No Motion'}</Text>

<Button title="Toggle Lights" onPress={() => {/\* Add logic to publish MQTT message to control lights \*/}} />

</View>

);

};

export default App;

**Conclusion**

In conclusion, the automation system design presented offers a comprehensive solution for a smart room, incorporating various functionalities to enhance comfort, energy efficiency, and safety. The integration of a central controller (Raspberry Pi), sensors, actuators, and a mobile app provides a versatile and user-friendly platform for home automation.