1 Desenvolvimento

1.1 Lâmpada

Este projeto tem como objetivo o desenvolvimento de uma lâmpada inteligente com controle de intensidade luminosa (*dimmer*) e acionamento automático baseado em sensores de luminosidade e presença. Para isso, foram utilizados um microcontrolador, o sensor de luminosidade BH1750 e o sensor de presença por radar LD2410C, além da integração com o sistema de automação residencial Home Assistant OS (HAOS) via protocolo MQTT.

O desenvolvimento iniciou-se com uma prova de conceito, validando a comunicação MQTT entre o microcontrolador e o *addon MQTT Broker* no HAOS. A partir dessa etapa, foram realizadas as integrações dos sensores e a criação de automações no Home Assistant.

1.1.1 Conexão Wi-Fi

A conexão Wi-Fi foi implementada com base no exemplo oficial disponível na biblioteca Arduino WiFi.[1]

O SSID e a senha da rede são armazenados em variáveis, e a função `WiFi.begin()` é utilizada para iniciar a conexão.

O status da conexão é exibido no monitor serial, permitindo a verificação do sucesso da conexão e facilitando a depuração.

```
const char* ssid = "HA AP";
const char* password = "12345678";

void connectToWiFi() {
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Conectando ao Wi-Fi...");
  }
  Serial.println("Conectado ao Wi-Fi");
}
```

[1] https://github.com/arduino-libraries/WiFi/blob/master/examples/ConnectWithWPA/ConnectWithWPA.ino

1.1.2 Conexão ao MQTT

A conexão ao *MQTT Broker* do HAOS foi desenvolvida com base no exemplo oficial da biblioteca PubSubClient.[2]

O endereço IPv4 do HAOS, bem como o nome de usuário e a senha de um usuário secundário, são armazenados em variáveis. A função `client.setServer()` é utilizada para definir o endereço IP e a porta do broker MQTT. A seguir, a função `client.setCallback()` registra o método que será executado sempre que uma mensagem for recebida. A conexão é então estabelecida por meio da função `client.connect()`, que recebe como parâmetros o identificador do cliente (no caso, o ESP) e as credenciais. O status da conexão é exibido no monitor serial, auxiliando na depuração e no monitoramento da comunicação MQTT.Após a conexão bem-sucedida, o ESP se inscreve nos tópicos de interesse utilizando a função `client.subscribe()`.

```
const char* mqtt_server = "192.168.88.253";
const char* mqtt_user = "mosquito";
const char* mqtt_password = "12345";

WiFiClient espClient;
PubSubClient client(espClient);

void setupMQTT() {
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);

  while (!client.connected()) {
    Serial.println("Conectando ao MQTT...");
    if (client.connect("ESP32Client", mqtt_user, mqtt_password)) {
      Serial.println("Conectado ao MQTT");
      client.subscribe("topico/teste");
    } else {
      Serial.print("Falha, rc=");
      Serial.print(client.state());
      delay(5000);
    }
  }
}

void mqttLoop() {
  client.loop();
}
```

[2] https://github.com/knolleary/pubsubclient/blob/master/examples/mqtt_basic/mqtt_basic.ino

1.1.3 *Callback* MQTT

A função `callback`, mencionada anteriormente, tem a função de processar as mensagens recebidas nos tópicos em que o ESP está inscrito.

O conteúdo da mensagem, inicialmente recebido como um vetor de bytes, é convertido para uma string, armazenado na variável `message` e, em seguida, exibido no monitor serial.

```
void callback(char* topic, byte* payload, unsigned int length){
        String message = "";
        for (unsigned int i = 0; i < length; i++) {
                message += (char)payload[i];
        }

        Serial.print("Mensagem recebida [");
        Serial.print(topic);
        Serial.print("]: ");
        Serial.println(message);
}
```

A comunicação MQTT foi testada pelo menu do add-on Mosquitto MQTT, publicando mensagens nos tópicos e verificando a recepção no monitor serial.

1.2 Teste dos Sensores

Com a conexão Wi-Fi e o protocolo de comunicação MQTT funcionando corretamente, prosseguiu-se com a implementação da leitura dos sensores no microcontrolador, utilizando como base os exemplos disponibilizados nas bibliotecas dos respectivos sensores.

1.2.1 Sensor BH1750

O código para leitura do sensor de luminosidade BH1750 foi baseado no exemplo oficial dabibliotecado sensor. [3]

Define-se um tópico MQTT para envio dos dados e uma variável para controle do intervalo de leitura. A comunicação I2C é inicializada com `Wire.begin()`, e o sensor é configurado com `lightMeter.begin()`. A cada intervalo, a função `lightMeter.readLightLevel()` obtém o valor em lux, que é convertido de `float` para `string` com `dtostrf()` e publicado via `client.publish()` no tópico MQTT correspondente.

[3] `https://github.com/claws/BH1750/blob/master/examples/BH1750test/BH1750test.ino`

```
BH1750 lightMeter;
const char* topic_lux = "interruptor/lux";
uint32_t lastLuxReading = 0;

void initLuxSensor() {
  Wire.begin(2, 1);
  lightMeter.begin();
}

void readAndPublishLux() {
  if (millis() - lastLuxReading > 5000) {
    lastLuxReading = millis();
    float lux = lightMeter.readLightLevel();
    char luxStr[8];
    dtostrf(lux, 1, 2, luxStr);
    client.publish(topic_lux, luxStr);
  }
}
```

1.2.2 Sensor de Radar LD2410

O código para integração do sensor de radar LD2410 foi desenvolvido com base no exemplo oficial da biblioteca do sensor. [4]

Inicialmente, define-se os pinos RX e TX utilizados na comunicação serial com o radar, além da porta serial secundária (`Serial1`). A função `initRadar()` realiza a inicialização da comunicação serial com o radar e verifica se a conexão foi bem-sucedida. A função `readAndPublishRadar()` é executada periodicamente e realiza a leitura dos dados do sensor. Quando é detectada uma mudança no estado de presença (presença detectada ou não detectada), essa informação é publicada em um tópico MQTT específico. Caso haja presença, também são coletadas e publicadas as informações de distância e energia dos alvos, tanto estacionários quanto em movimento, nos respectivos tópicos.

[4] https://github.com/ncmreynolds/ld2410/blob/main/examples/basicSensor/basicSensor.ino

```
#define MONITOR_SERIAL Serial
#define RADAR_SERIAL Serial1
#define RADAR_RX_PIN 20
#define RADAR_TX_PIN 21

ld2410 radar;
uint32_t lastReading = 0;
bool presenceState = false;

void initRadar() {
  RADAR_SERIAL.begin(256000, SERIAL_8N1, RADAR_RX_PIN, RADAR_TX_PIN);

  if (radar.begin(RADAR_SERIAL)) {
    MONITOR_SERIAL.println(F("OK"));
  } else {
    MONITOR_SERIAL.println(F("NÃO CONECTADO"));
  }
}

void readAndPublishRadar() {
  if (millis() - lastReading > 1000) {
    lastReading = millis();
    radar.read();

    bool presenceDetected = radar.presenceDetected();

    if (presenceDetected && !presenceState) {
      client.publish("interruptor/radar", "Presença Detectada");
      presenceState = true;
    }
    else if (!presenceDetected && presenceState) {
      client.publish("interruptor/radar", "Presença Não Detectada");
      presenceState = false;
    }

    if (presenceDetected) {
      if (radar.stationaryTargetDetected()) {
        char distStr[8], energyStr[8];
        itoa(radar.stationaryTargetDistance(), distStr, 10);
        itoa(radar.stationaryTargetEnergy(), energyStr, 10);
        client.publish("interruptor/radar/estacionario/distancia", distStr);
        client.publish("interruptor/radar/estacionario/energia", energyStr);
      }
```

```
      if (radar.movingTargetDetected()) {
        char distStr[8], energyStr[8];
        itoa(radar.movingTargetDistance(), distStr, 10);
        itoa(radar.movingTargetEnergy(), energyStr, 10);
        client.publish("interruptor/radar/movendo/distancia", distStr);
        client.publish("interruptor/radar/movendo/energia", energyStr);
      }
    }
  }
}
```

1.3 Juntando todas as funções

Com a estrutura modular do código pronta e todas as funções definidas nos cabeçalhos dos arquivos c++ foi feito o código que faz a chamada de todas as funções usando o Arduino IDE.

```
#include "wifi_manager.h"
#include "mqtt_manager.h"
#include "radar_sensor.h"
#include "lux_sensor.h"

void setup() {
  Serial.begin(115200);
  connectToWiFi();
  setupMQTT();
  initLuxSensor();
  initRadar();
}

void loop() {
  mqttLoop();
  readAndPublishLux();
  readAndPublishRadar();
}
```

1.4 Recebendo os dados no HA

Com o *addon* mosquito MQTT instalado no HA, serão adicionadas entidades no HA editando o arquivo `configuration.yaml` as entidades de sensores são definidas de acordo com a documentação oficial do HA. [5]

1.4.1 Definindo os sensores em yaml.

Sensores MQTT são definidos em uma tabela yaml que recebe um nome amigável para o sensor, um identificador para a entidade do HA e o tópico onde o estado do sensor será publicado.

[5] `https://www.home-assistant.io/integrations/sensor.mqtt/`

Será declarado um sensor para cada tópico MQTT definido anteriormente no ESP.

```yaml
mqtt:
  sensor:
    - name: "Lux"
      unique_id: sensor_lux
      state_topic: "interruptor/lux"
      device_class: illuminance
    - name: "Presença Estacionária Distância"
      unique_id: sensor_est_dis
      state_topic: "interruptor/radar/estacionario/distancia"
    - name: "Presença Estacionária Energia"
      unique_id: sensor_est_ene
      state_topic: "interruptor/radar/estacionario/energia"
    - name: "Presença Em Movimento Distância"
      unique_id: sensor_mov_dis
      state_topic: "interruptor/radar/movendo/distancia"
    - name: "Presença Em Movimento Energia"
      unique_id: sensor_mov_ene
      state_topic: "interruptor/radar/movendo/energia"
    - name: "Presença Detectada"
      unique_id: sensor_mov_pre
      state_topic: "interruptor/radar"
```

Com os dados armazenados nas entidades elas pode ser usadas de diversas formas no HA, para os nossos propósitos vamos focar em automações no HA.

1.5 Automações

Automações no HA podem ser definidas no arquivo `automations.yaml` ou pelo editor de blocos visual do HA.

1.5.1 *Swith* para a lâmpada

Foi criada uma automação com o objetivo de ligar e desligar a lampada com base no sensor de presença, para isso é necessário configurar um *switch* MQTT que é declarado no arquivo `configuration.yaml` da seguinte forma

```yaml
mqtt:
  switch:
    - name: "Lampada"
      unique_id: lampada
      state_topic: "interruptor/switch"
      command_topic: "interruptor/switch"
      payload_on: "1"
      payload_off: "0"
```

Com isso foi criada uma entidade *swith* que posta (1) no tópico `interruptor/switch` quando o *switch* é ligado e (0) quando desligado. Devido a natureza do MQTT caso esse estado seja alterado por outro dispositivo ou de outra forma no HA a entidade também será atualizado de acordo. Com a entidade definida foram criadas duas automações uma para ligar o *switch* caso presença seja detectada e desligar o *switch* após uma quantidade arbitraria de tempo sem presença detectada.

```yaml
id: '1743357707824'
  alias: Detector de Presença (Ligar)
  description: Liga a lampada se presença detectada
  triggers:
  - entity_id:
    - sensor.presenca_detectada
    from: Presença Não Detectada
    to: Precença Detectada
    trigger: state
  conditions: []
  actions:
  - action: switch.turn_on
    metadata: {}
    data: {}
    target:
      entity_id: switch.lampada
  mode: single
- id: '1745710625710'
  alias: Sensor de Presença (Desligar)
  description: Desliga a lampada se presença não detectada
  triggers:
  - entity_id:
    - sensor.presenca_detectada
    to: Presença Não Detectada
    trigger: state
    from: Presença Detectada
    for:
      hours: 0
      minutes: 3
      seconds: 0
```

```yaml
conditions: []
actions:
- action: switch.turn_off
  metadata: {}
  data: {}
  target:
    entity_id: switch.lampada
mode: single
```

Inscrevendo o ESP nesse tópico torna-se possível acessar essa condição.

Na função `setupMQTT()`

```
void setupMQTT() {
client.subscribe("interruptor/switch");
}
```

Para controlar a lâmpada por meio de um relé, pode-se utilizar uma estrutura condicional `if`, inserida na função `callback()`. Nessa função, a comparação entre a `string` da variável `topic` e o tópico previamente definido no HA é realizada por meio da função `strcmp()`. Em seguida, o valor da mensagem contido na variável `message` é verificado e, com base nele, a função `digitalWrite()` é utilizada para alterar o estado lógico do pino conectado ao relé.

Na função `callback()`

```
const int pinoRELE = 0;
void callback(char* topic, byte* payload, unsigned int length) {
        if (strcmp(topic, "interruptor/switch") == 0) {
                if (message == "1") {
                        digitalWrite(pinoRELE, HIGH);
                } else if (message == "0") {
                        digitalWrite(pinoRELE, LOW);
                }
        }
}
```

1.5.2 Controle de brilho da lâmpada

Em seguida usando os valores lidos e publicados no tópico do sensor lux para automatizar o controle do brilho da lâmpada.

Adicionada uma entidade `number` no arquivo `configuration.yaml` que também é postada em seu respectivo tópico.

```
mqtt:
  number:
    - name: "Brilho"
      unique_id: mqtt_slider_Brilho
      state_topic: "lampada/brilho"
      command_topic: "lampada/brilho"
      min: 0
      max: 100
      step: 1
      retain: false
      unit_of_measurement: "%"
```

E criando uma automação usando a formula sugerida nos forums do HA [6] e usando como base o `blueprint` [7]

$$Brilo = (Declive \times lux) + constante$$

No arquivo `automations.yaml` onde teremos as variáveis `maxB` Que seria o valor de lux onde o brilho é configurado em 0 como definido na variável `light_value_1`, `minB` o valor de lux onde o brilho é configurado em 100 como definido na variável `ligh_value_2`. O declive é calculado com a formula:

$$slope = \frac{ligh1 - light2}{maxB - minB}$$

e a constante com a formula:

$$constant = light1 - (slope \times maxB)$$

```
- id: '1745186830191'
  alias: Dimmer
  description: Configura Brilho baseado em um valor alvo
  triggers:
  - entity_id: sensor.lux
    trigger: state
  conditions:
  - condition: numeric_state
    entity_id: sensor.lux
    above: 0
  actions:
  - target:
      entity_id: number.brilho
    data:
      value: "{% if states(light_sensor)|int > maxB %}\n  0\n{% else %}\n  {{ (( slope
        * states(light_sensor)|int ) + constant)|round }}\n{% endif %}\n"
    action: number.set_value
  variables:
    light_sensor: sensor.lux
    maxB: 400
    minB: 0
    light_value_1: 0
    light_value_2: 100
    light1: '{{ light_value_1 * 1 }}'
    light2: '{{ light_value_2 * 1 }}'
    slope: '{{ ( light1 - light2 ) / ( maxB - minB ) }}'
    constant: '{{ light1 - ( slope * maxB ) }}'
  mode: single
```

Novamente o ESP é inscrito nesse tópico.

Na função `setupMQTT()`

[6] https://community.home-assistant.io/t/dim-lights-as-lumens-increases/182065/15

[7] https://community.home-assistant.io/t/smart-lux-dimmer-adjust-light-brightness-depending-on-light-sensor-value/403646

```
void setupMQTT() {
        client.subscribe("lampada/brilho");
}
```

Na função `callback()`, são definidas variáveis para o sinal PWM, incluindo frequência, resolução e o pino correspondente. A função `ledcAttach()` do ESP é utilizada para configurar o canal PWM com esses parâmetros. O valor recebido é restringido entre 0 e 100 e, em seguida, convertido proporcionalmente para o intervalo de 0 a 255, que representa o *Duty Cycle* do PWM do ESP. Por fim, a função `ledcWrite()` ajusta o pino com o *Duty Cycle* calculado, baseado no valor do brilho recebido no tópico `lampada/brilho`.

```
const int pwmFreq = 20000;  // Frequência de 20kHz
const int pwmResolution = 8; // Resolução de 8 bits (0-255)
const int pinoPWM = 10;
void setupPWM() {
        ledcAttach(pinoPWM, pwmFreq, pwmResolution);
}
void callback(char* topic, byte* payload, unsigned int length) {
        if (strcmp(topic, "lampada/brilho") == 0) {
                int brilho = constrain(message.toInt(), 0, 100);
                int pwm = (255.0 / 100.0) * brilho;
                ledcWrite(pinoPWM, pwm);
        }
}
```

1.6 Modificando uma lâmpada

Lâmpadas de LED comum não podem ter seu brilho ajustado dinamicamente como lampadas de outras tecnologias podiam.

Para modificar o brilho de lampadas modernas de LED, que usam um simples circuito integrado para controlar a potência da lâmpada foi usado o principio exemplificado nesse vídeo [8], mas ao invéz de modificar o valor dos resistores, o que reduziria a potência da lâmpada o brilho pode ser controlado dinamicamente usando um transistor que recebe o sinal PWM do ESP.

[8] https://www.youtube.com/watch?v=5HTa2jVi_rc