UNIVERSIDADE DO ESTADO DE SANTA CATARINA 2025/5° FASE
CENTRO DE EDUCAÇÃO DO PLANALTO NORTE/DEPARTAMENTO DE SISTEMAS DE INFORMAÇÃO AUTOMAÇÃO DE SISTEMAS
Aluno(s): LISTA DE NOMES DOS ALUNOS
Matrícula(s): LISTA DE MATRÍCULAS DOS ALUNOS

ROTEIRO DE ATIVIDADES ESP-32 e IoT

1. Introdução ao ESP32 e Conceitos de IoT

1. O que é o microcontrolador?

   Um microcontrolador é um circuito integrado que contém um processador, memória e periféricos de entrada/saída em um único chip, sendo utilizado para controlar dispositivos eletrônicos de forma autônoma.

2. Diferença entre ESP8266, ESP32 e Arduino Uno.

   O ESP8266 é um microcontrolador com conectividade Wi-Fi, enquanto o ESP32 é uma versão mais avançada com Wi-Fi e Bluetooth integrados, oferecendo mais recursos e potência de processamento. O Arduino Uno é uma placa de prototipagem baseada no microcontrolador ATmega328, que não possui conectividade Wi-Fi ou Bluetooth nativa, mas é amplamente utilizado para projetos de eletrônica e programação básica.

3. Conceitos básicos de Internet das Coisas (IoT).

   Internet das Coisas (IoT) refere-se à interconexão de dispositivos físicos à internet, permitindo que eles coletem e compartilhem dados, além de serem controlados remotamente. Isso possibilita a automação, monitoramento e otimização de processos em diversas áreas, como residências, indústrias e cidades inteligentes.

4. Exemplos de aplicações com ESP32 e IoT. O ESP32 pode ser utilizado em diversas aplicações de IoT, como:

   - Monitoramento ambiental (temperatura, umidade, qualidade do ar);
   - Automação residencial (controle de luzes, eletrodomésticos, sistemas de segurança);
   - Sistemas de irrigação inteligente;
   - Dispositivos vestíveis (wearables) para monitoramento de saúde;
   - Controle remoto de robôs e drones.

2. Instalação da IDE Arduino e Configuração do ESP32.

1. Instalação da IDE Arduino:

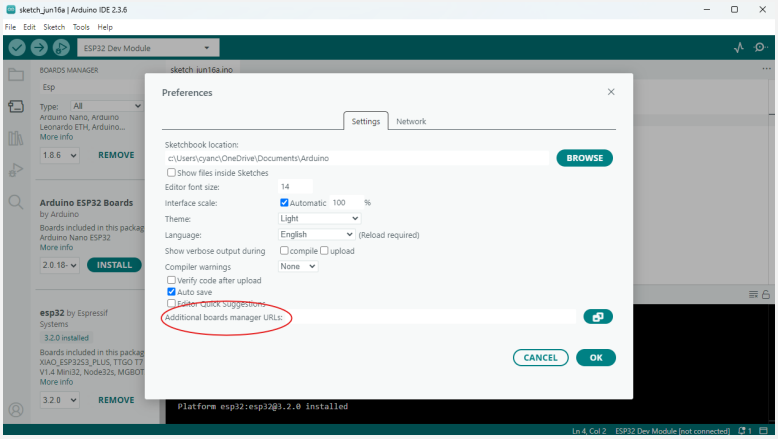
   A IDE Arduino é uma plataforma de desenvolvimento integrada que permite programar microcontroladores, incluindo o ESP32. Para instalar a IDE, basta baixar o instalador do site oficial do Arduino e seguir as instruções de instalação.

   - Acesse o site oficial do Arduino: `https://www.arduino.cc/en/software`.
   - Baixe a versão adequada para o seu sistema operacional (Windows, macOS, Linux).
   - Execute o instalador e siga as instruções na tela.
   - Após a instalação, abra a IDE Arduino.

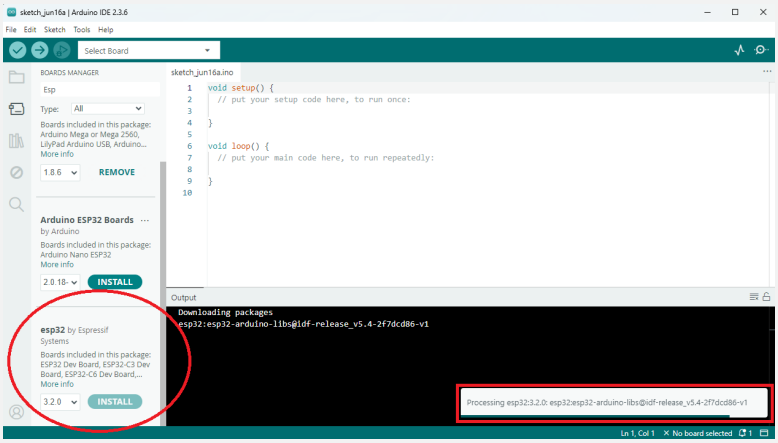
2. Configuração do ESP32 na IDE Arduino:

   Para programar o ESP32 na IDE Arduino, é necessário instalar o suporte ao ESP32. Siga os passos abaixo:

- Abra a IDE Arduino.
- Vá para "File" > "Preferences".
- Na seção "Additional Boards Manager URLs", adicione a seguinte URL: `https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json`.
- Para facilitar, veja a imagem abaixo mostrando onde adicionar a URL:



- Clique em "OK" para salvar as preferências.
- Vá para "Tools" > "Board" > "Boards Manager".
- Pesquise por "ESP32" e instale o pacote "esp32 by Espressif".
- Veja a imagem abaixo mostrando a seleção da placa:



- Após a instalação, selecione a sua placa ESP32 em "Tools" > "Board".
- Após instalar o pacote e selecionr a placa, conecte sua placa ESP32 ao computador.
- Selecione a porta correta em "Tools" > "Port".

3. Teste de conexão com código "Blink":

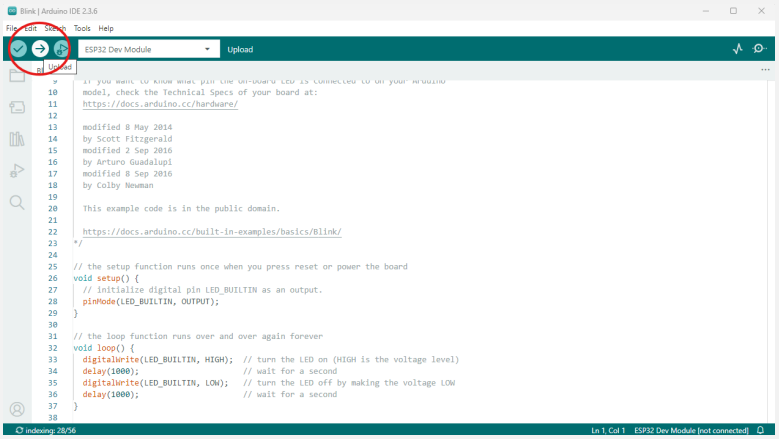
   Para verificar se a configuração está correta, você pode carregar o exemplo "Blink" na IDE Arduino. Siga os passos abaixo:

   - Vá para "File" > "Examples" > "01.Basics" > "Blink".
   - O código do exemplo deve ser semelhante ao seguinte:

```
void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    digitalWrite(LED_BUILTIN, HIGH); // Liga LED
    delay(1000); // Espera por 1 segundo
    digitalWrite(LED_BUILTIN, LOW); // Desliga LED
    delay(1000); // Espera por 1 segundo
}
```

- Carregue o código na placa ESP32 clicando no botão de upload (seta para a direita).



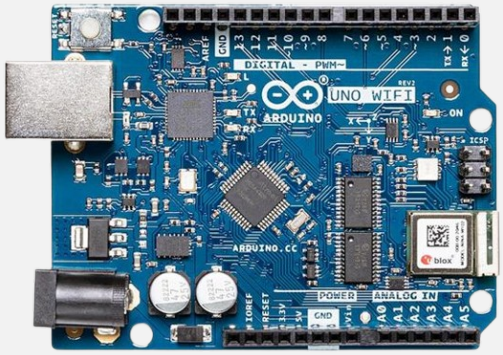
- Após o upload, o LED integrado da placa deve piscar a cada segundo.

3. Comparação com ESP8266 e Arduino Uno.

- **ESP32:** Microcontrolador avançado da Espressif, possui processador dual-core, conectividade Wi-Fi e Bluetooth integrados, maior quantidade de pinos de entrada/saída, suporte a múltiplos periféricos, ADCs de maior resolução e maior capacidade de processamento e memória. Ideal para aplicações IoT mais complexas e que exigem conectividade sem fio diversificada.

- **ESP8266:** Também da Espressif, é mais simples que o ESP32, com processador single-core, conectividade Wi-Fi integrada, menos pinos e recursos. É indicado para projetos IoT básicos que demandam apenas Wi-Fi e menor consumo de recursos.

- **Arduino Uno:** Baseado no microcontrolador ATmega328P, não possui conectividade Wi-Fi ou Bluetooth nativa, mas é muito utilizado em projetos de eletrônica básica, prototipagem e ensino. Possui menos memória e processamento em relação aos ESPs, mas conta com vasta documentação e comunidade.



Resumo: O ESP32 é o mais completo em termos de recursos e conectividade, seguido pelo ESP8266 (mais simples e barato), enquanto o Arduino Uno é ideal para projetos básicos sem necessidade de conexão sem fio.

4. Leitura de sensores analógicos e digitais

Obetivo: Aprender a ler dados de sensores analógicos e digitais utilizando o ESP32.

1. Leitura de sensores analógicos:

   Para ler dados de sensores analógicos, como um potenciômetro ou sensor de temperatura, você pode usar a função 'analogRead()'. O ESP32 possui vários pinos ADC (Conversores Analógico-Digital) que podem ser utilizados para essa finalidade.

   Exemplo de código para ler um sensor analógico:

```
int sensorPin = 34; // Pino ADC
int sensorValue = 0;

void setup() {
    Serial.begin(115200);
}
```

```
void loop() {
    sensorValue = analogRead(sensorPin); // Lê o valor do sensor
    Serial.println(sensorValue); // Imprime o valor no monitor serial
    delay(1000); // Espera 1 segundo
}
```

2. Leitura de sensores digitais:

   Para ler dados de sensores digitais, como um botão ou sensor de movimento, você pode usar a função 'digitalRead()'. O ESP32 possui vários pinos digitais que podem ser utilizados para essa finalidade.

   Exemplo de código para ler um sensor digital:

```
int buttonPin = 2; // Pino digital
int buttonState = 0;

void setup() {
    pinMode(buttonPin, INPUT); // Configura o pino como entrada
    Serial.begin(115200);
}

void loop() {
    buttonState = digitalRead(buttonPin); // Lê o estado do botão
    Serial.println(buttonState); // Imprime o estado no monitor serial
    delay(1000); // Espera 1 segundo
}
```

5. Controle de atuadores (EX: LED e buzzer).

1. Controle de LED:

   Para controlar um LED, você pode usar a função 'digitalWrite()'. O ESP32 possui vários pinos digitais que podem ser utilizados para essa finalidade.

   Exemplo de código para controlar um LED:

```
int ledPin = 2; // Pino do LED

void setup() {
    pinMode(ledPin, OUTPUT); // Configura o pino como saída
}

void loop() {
    digitalWrite(ledPin, HIGH); // Liga o LED
    delay(1000); // Espera 1 segundo
    digitalWrite(ledPin, LOW); // Desliga o LED
    delay(1000); // Espera 1 segundo
}
```

   O resultado será um LED piscando a cada segundo (Conforme a imagem abaixo).

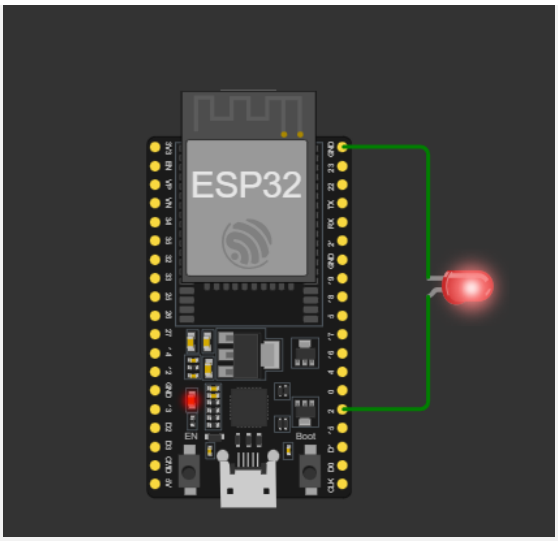


Figura: Imagem retirada do simulador ESP32 wokwi.com.

2. Controle de buzzer:

   Para controlar um buzzer, você também pode usar a função ‘digitalWrite()‘.

   Exemplo de código para controlar um buzzer:

```
int buzzerPin = 2; // Pino do buzzer

void setup() {
    pinMode(buzzerPin, OUTPUT); // Configura o pino como saída
}

void loop() {
    // Aumenta a frequência gradualmente (efeito de sirene)
    for (int freq = 800; freq <= 2000; freq += 10) {
        tone(buzzerPin, freq); // Gera tom com a frequência atual
        delay(10); // Pequeno atraso para suavizar a transição
    }

    // Diminui a frequência gradualmente
    for (int freq = 2000; freq >= 800; freq -= 10) {
        tone(buzzerPin, freq); // Gera tom com a frequência atual
        delay(10); // Pequeno atraso para suavizar a transição
    }

    noTone(buzzerPin); // Para o som
    delay(200); // Pausa breve antes de reiniciar
    }
```

   O resultado será um buzzer emitindo um som de sirene (visualmente conforme a imagem abaixo).

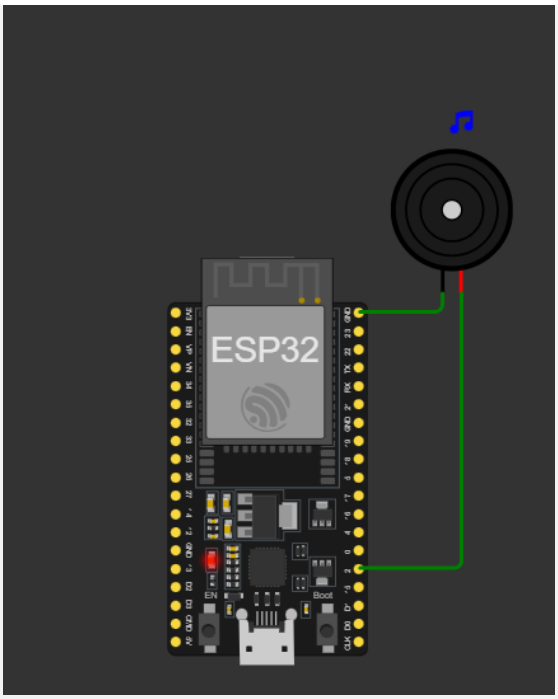


Figura: Imagem retirada do simulador ESP32 wokwi.com.

6. Conectando o ESP32 a uma rede Wi-Fi.

1. Biblioteca WiFih:

   Para conectar o ESP32 a uma rede Wi-Fi, utilizamos a biblioteca WiFi.h, que já vem incluída no pacote ESP32 da IDE Arduino. Esta biblioteca fornece todas as funções necessárias para gerenciar conexões Wi-Fi..

```
#include <WiFi.h>
```

2. Configuração básica de conexão Wi-Fi:

   Exemplo básico de como conectar o ESP32 a uma rede Wi-Fi fornecendo o SSID e senha:

```
#include <WiFi.h>

const char* ssid = "NOME_DA_SUA_REDE";
const char* password = "SENHA_DA_SUA_REDE";

void setup() {
    Serial.begin(115200);

    WiFi.begin(ssid, password);
    Serial.print("Conectando à rede Wi-Fi");

    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.print(".");
    }
```

```
    Serial.println();
    Serial.println("Conectado com sucesso!");
    Serial.print("Endereço IP: ");
    Serial.println(WiFi.localIP());
}

void loop() {
// Código principal aqui
}
```

3. Verificação do status da conexão:

   Código para verificar o status da conexão e exibir informações sobre a rede Wi-Fi:

```
#include <WiFi.h>

const char* ssid = "NOME_DA_SUA_REDE";
const char* password = "SENHA_DA_SUA_REDE";

void setup() {
Serial.begin(115200);

WiFi.begin(ssid, password);
Serial.print("Conectando à rede Wi-Fi");

int tentativas = 0;
while (WiFi.status() != WL_CONNECTED && tentativas < 20) {
    delay(1000);
    Serial.print(".");
    tentativas++;
}

if (WiFi.status() == WL_CONNECTED) {
    Serial.println();
    Serial.println("Conectado com sucesso!");
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());
    Serial.print("Endereço IP: ");
    Serial.println(WiFi.localIP());
    Serial.print("Intensidade do sinal (RSSI): ");
    Serial.println(WiFi.RSSI());
} else {
    Serial.println();
    Serial.println("Falha na conexão Wi-Fi!");
    Serial.println("Verifique as credenciais da rede.");
}
}

void loop() {
if (WiFi.status() == WL_CONNECTED) {
    Serial.println("Wi-Fi conectado");
} else {
    Serial.println("Wi-Fi desconectado - tentando reconectar...");
    WiFi.begin(ssid, password);
```

```
}

delay(10000);
}
```

4. Códigos de status Wi-Fi:

   Principais códigos de status retornados por WiFi.status():

   - `WL_CONNECTED` (3): Conectado com sucesso.
   - `WL_NO_SSID_AVAIL` (1): SSID não encontrado.
   - `WL_CONNECT_FAILED`: (4) Falha na conexão por senha incorreta.
   - `WL_CONNECTION_LOST`: A conexão com a rede Wi-Fi foi perdida.
   - `WL_DISCONNECTED`: O ESP32 está desconectado da rede Wi-Fi.

5. Exemplo completo com reconexão automática:

   Código completo com reconexão automática à rede Wi-Fi:

```
#include <WiFi.h>

const char* ssid = "NOME_DA_SUA_REDE";
const char* password = "SENHA_DA_SUA_REDE";

void conectarWiFi() {
WiFi.begin(ssid, password);

Serial.print("Conectando à rede Wi-Fi");

int tentativas = 0;
while (WiFi.status() != WL_CONNECTED && tentativas < 20) {
    delay(1000);
    Serial.print(".");
    tentativas++;
}

if (WiFi.status() == WL_CONNECTED) {
    Serial.println();
    Serial.println("OK Conectado com sucesso!");
    exibirInformacoesRede();
} else {
    Serial.println();
    Serial.println("X Falha na conexão Wi-Fi!");
}
}

void exibirInformacoesRede() {
    Serial.println("=== Informações da Rede ===");
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());
    Serial.print("Endereço IP: ");
    Serial.println(WiFi.localIP());
    Serial.print("Gateway: ");
    Serial.println(WiFi.gatewayIP());
    Serial.print("Máscara de sub-rede: ");
```

```
        Serial.println(WiFi.subnetMask());
        Serial.print("DNS: ");
        Serial.println(WiFi.dnsIP());
        Serial.print("Intensidade do sinal (RSSI): ");
        Serial.print(WiFi.RSSI());
        Serial.println(" dBm");
        Serial.println("============================");
    }

    void setup() {
        Serial.begin(115200);
        WiFi.mode(WIFI_STA);
        conectarWiFi();
    }

    void loop() {
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("Conexão Wi-Fi perdida. Tentando reconectar...");
        conectarWiFi();
    } else {
        Serial.println("Wi-Fi conectado - Sistema funcionando normalmente");
    }

    delay(30000);
    }
```

6. Explicação do código:

   Dicas importantes:

   - Substitua as credenciais da rede pelos seus dados reais.
   - RSSI próximo de 0 indica sinal forte. Mais negativo = mais fraco.
   - `WiFi.mode(WIFI_STA)` configura o ESP32 como cliente Wi-Fi.
   - Outros modos: `WIFI_AP` (Access Point), `WIFI_AP_STA` (modo misto).
   - Definir limite de tentativas evita travamento se a rede estiver indisponível.

7. Testando a conexão:

   - Após carregar o código, abra o Monitor Serial (Ctrl+Shift+M).
   - Configure para 115200 baud.
   - Veja as mensagens de conexão e o IP atribuído ao ESP32.

   Com o ESP32 conectado, é possível comunicar com outros dispositivos e com a Internet usando protocolos como MQTT.

7. Introdução ao protocolo MQTT.

O protocolo MQTT é o protocolo padrão pra comunicação entre dispositivos IoT. Ele é planejado pra ser extremamente leve, com o padrão de publicador / assinante que é ideal pra conectar dispositivos com o mínimo de recursos possível. Ele é usado em diversas indústrias, como automotiva, manufatura, telecomunicação, entre outras.

1. O padrão da indústria:

Ele é o protocolo mais usado na indústria, porque é o que melhor funciona em dispositivos com recursos limitados, com baixa largura de banda e conectividade limitada. Ele usa por baixo dos panos o TCP/IP, com um formato de mensagem binária, em contraste com outros protocolos que usam um formato de mensagens de texto, como o HTTP, saindo na frente no transporte de dados.

2. Modelo Publicador / Assinante:

   - `Publicadores (publishers) enviam mensagens em tópicos.`
   - `Assinantes (subscribers) recebem mensagens do tópico que assinaram.`
   - `Um broker (servidor intermediário) gerencia o envio das mensagens aos clientes.`

   É assim que funciona a comunicação entre os dispositivos com o protocolo MQTT. Ele também tem outras funcionalidades interessantes, como a de sessões persistentes: se a sessão entre um publicador / assinante for desconectada, a sessão vai continuar ativa mesmo assim. Uma vez que a internet for conectada novamente, eles vão se conectar de novo automaticamente e continuar a sessão.

   Também existem outras opções que o MQTT oferece:

   - `Retenção de mensagens:` É possível configurar mensagens como "retidas", assim quando novos assinantes assinam um tópico, eles recebem imediatamente a última mensagem publicada.
   - `Qualidade de serviço (QoS):` Oferece três níveis de entrega de mensagens:
     - QoS 0: Entrega no mínimo uma vez (sem garantia de entrega).
     - QoS 1: Entrega pelo menos uma vez (pode haver duplicação).
     - QoS 2: Entrega exatamente uma vez (garantia de entrega sem duplicação).
   - `Mensagens com última vontade (Last will and testament):` Permite que o cliente defina uma mensagem que será enviada se ele se desconectar de forma inesperada. É útil pra detectar falhas em dispositivos.

8. Publicando dados do sensor em um broker MQTT.

Como Publicar Dados de um Sensor em um Broker MQTT Usando um ESP32:

O protocolo MQTT (Message Queuing Telemetry Transport) é ideal para aplicações de IoT devido à sua leveza e eficiência. Com um ESP32, é possível coletar dados de um sensor e publicá-los em um broker MQTT de forma simples. Abaixo estão os passos para configurar e implementar essa tarefa.

1. Escolha e Configure o Broker MQTT:

   O broker MQTT é o servidor que gerencia as mensagens entre dispositivos. Você pode usar:

   - **Broker local:** Instale o Mosquitto em um computador ou servidor:

     ```
     sudo apt-get install mosquitto mosquitto-clients
     ```

   - **Broker na nuvem:** Serviços como HiveMQ Cloud ou Adafruit IO são boas opções. Crie uma conta para obter o endereço do broker, porta (geralmente 1883 para conexões não seguras ou 8883 para TLS), usuário e senha (se necessário).
   - Certifique-se de que o broker está acessível e as portas estão abertas.

2. Conecte o Sensor ao ESP32: Conecte o sensor ao ESP32. Neste exemplo, usaremos um sensor de temperatura e umidade DHT11:

- **Pino de dados do DHT11:** Conecte a um pino GPIO, como o GPIO 4.
- **Alimentação:** Conecte VCC ao 3,3V e GND ao GND do ESP32.
- Adicione um resistor pull-up de 4.7 kΩ a 10 kΩ entre o pino de dados e o VCC, se necessário.
- Instale a biblioteca DHT sensor library no Arduino IDE para facilitar a leitura do sensor.

3. Configure o Ambiente de Programação:

   - **Arduino IDE:** Instale o suporte ao ESP32 no Arduino IDE (em "Placas" > "Gerenciador de Placas", procure por "ESP32").
   - **Bibliotecas:** Instale as bibliotecas PubSubClient (para MQTT) e WiFi (já inclusa no suporte ao ESP32).
   - Configure o Wi-Fi do ESP32 com o SSID e a senha da sua rede.

4. Escreva o Código para o ESP32:

   O código abaixo lê a temperatura do sensor DHT11 e a publica em um tópico MQTT. Substitua as variáveis com suas configurações específicas:

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

// Configurações de Wi-Fi
const char* ssid = "SUA_REDE_WIFI";
const char* password = "SUA_SENHA_WIFI";

// Configurações do broker MQTT
const char* mqtt_server = "SEU_BROKER_MQTT"; // ex: broker.hivemq.com
const int mqtt_port = 1883;
const char* mqtt_user = "SEU_USUARIO"; // Opcional
const char* mqtt_password = "SUA_SENHA"; // Opcional
const char* mqtt_topic = "esp32/temperatura";

// Configurações do sensor DHT
#define DHTPIN 4 // Pino GPIO conectado ao DHT11
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
    Serial.begin(115200);
    dht.begin();

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

```
    // Configurar o broker MQTT
    client.setServer(mqtt_server, mqtt_port);
    }

void reconnect() {
    while (!client.connected()) {
        Serial.println("Conectando ao broker MQTT...");
        String clientId = "ESP32Client-" + String(random(0xffff), HEX);
        if (client.connect(clientId.c_str(), mqtt_user, mqtt_password)) {
            Serial.println("Conectado ao broker MQTT");
        } else {
            Serial.print("Falha na conexão, rc=");
            Serial.print(client.state());
            delay(5000);
        }
    }
}

void loop() {
    if (!client.connected()) {
        reconnect();
    }
    client.loop();

    // Ler dados do sensor
    float temperatura = dht.readTemperature();
    if (isnan(temperatura)) {
        Serial.println("Erro ao ler o sensor DHT11!");
        delay(2000);
        return;
    }

    // Publicar dados no tópico
    String payload = String(temperatura);
    client.publish(mqtt_topic, payload.c_str());
    Serial.println("Temperatura publicada: " + payload + " °C");

    delay(5000); // Publica a cada 5 segundos
}
```

5. Escolha do Tópico:

   Escolha um tópico descritivo, como esp32/temperatura. Tópicos são organizados hierarquicamente (ex.: dispositivo/local/variável) para facilitar o gerenciamento.

6. Teste a Publicação:

   - **Carregue o código:** Use o Arduino IDE para compilar e carregar o código no ESP32.
   - **Monitore via Serial:** Abra o Monitor Serial (115200 baud) para verificar a conexão Wi-Fi, a conexão com o broker e os dados do sensor.
   - **Verifique os dados no broker:** Use um cliente MQTT (como MQTT Explorer ou o comando `mosquitto_sub`) para assinar o tópico e confirmar a recepção dos dados:

     ```
     mosquitto_sub -h SEU_BROKER -t esp32/temperatura
     ```

7. Segurança (Opcional):

   Para Maior segurança:

   - Use MQTT com TLS/SSL (porta 8883) e configure certificados no ESP32.
   - Adicione autenticação com usuário e senha no broker.
   - Considere criptografar os dados do sensor, se necessário.

8. Dicas Adicionais:

   - **Qualidade de Serviço (QoS):** No método client.publish, você pode especificar o nível de QoS (0, 1 ou 2) como um parâmetro adicional, dependendo da confiabilidade desejada.
   - **Reconexão automática:** O código inclui uma função reconnect() para lidar com desconexões.
   - **Outros sensores:** Para sensores diferentes (como BMP280 ou DS18B20), ajuste a leitura no código, mantendo a estrutura de publicação MQTT.

9. Conclusão:

   Com um ESP32, é possível publicar dados de um sensor em um broker MQTT de forma eficiente usando a biblioteca PubSubClient. O exemplo acima usa um sensor DHT11, mas pode ser adaptado para outros sensores. Teste a conexão, monitore os dados e, se necessário, implemente segurança para aplicações reais.

9. Recebendo comandos MQTT para controle de atuadores.

Recebendo Comandos MQTT para Controle de Atuadores com ESP32

O protocolo MQTT, descrito na página 11 do documento, opera no modelo publicador/assinante, permitindo que dispositivos como o ESP32 publiquem dados em tópicos e assinem tópicos para receber mensagens. Para controlar atuadores (como LEDs, relés ou motores), o ESP32 pode assinar um tópico MQTT, receber comandos enviados por outro dispositivo ou aplicação e atuar sobre um pino GPIO com base nesses comandos. Este texto explica como configurar o ESP32 para receber comandos MQTT e controlar um atuador, usando como exemplo um LED conectado a um pino GPIO.

1. Pré-requisitos

   Antes de começar, certifique-se de que:

   - O ambiente do Arduino IDE está configurado para o ESP32.
   - O ESP32 está conectado a um sensor e a um atuador (ex.: um LED conectado ao GPIO 5).
   - Um broker MQTT está configurado (local, como Mosquitto, ou na nuvem, como HiveMQ Cloud), com endereço, porta (1883 ou 8883 para TLS), usuário e senha, se necessário.
   - As bibliotecas WiFi, PubSubClient e DHT estão instaladas no Arduino IDE.

2. Conexão do Atuador ao ESP32

   Conecte o atuador ao ESP32. Para este exemplo, usaremos um LED:

   - **Pino do LED:** Conecte o ânodo (perna longa) ao GPIO 5 e o cátodo (perna curta) ao GND, com um resistor de 220Ω a 330Ω em série para limitar a corrente.
   - Alternativamente, use um relé ou outro atuador compatível com os níveis de tensão do ESP32 (3,3V para sinais lógicos).

- Verifique se o pino escolhido (GPIO 5) suporta saída digital, conforme descrito na página 4 do documento sobre leitura de sensores analógicos e digitais.

3. Configuração do Código

   O código a seguir adapta o exemplo anterior (publicação de dados do sensor DHT11) para incluir a assinatura de um tópico MQTT e o controle de um LED com base em comandos recebidos. O ESP32 assina o tópico `esp32/comando` para receber mensagens como "ON" ou "OFF" e controla o LED no GPIO 5. A reconexão automática ao Wi-Fi e ao broker MQTT, também é implementada.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

// Configurações de Wi-Fi
const char* ssid = "NOME_DA_SUA_REDE";
const char* password = "SENHA_DA_SUA_REDE";

// Configurações do broker MQTT
const char* mqtt_server = "SEU_BROKER_MQTT"; // ex.: broker.hivemq.com
const int mqtt_port = 1883;
const char* mqtt_user = "SEU_USUARIO"; // Opcional
const char* mqtt_password = "SUA_SENHA"; // Opcional
const char* mqtt_topic_sensor = "esp32/temperatura";
const char* mqtt_topic_comando = "esp32/comando";

// Configurações do sensor DHT e do atuador
#define DHTPIN 4 // Pino GPIO conectado ao DHT11
#define DHTTYPE DHT11
#define LED_PIN 5 // Pino GPIO conectado ao LED
DHT dht(DHTPIN, DHTTYPE);

WiFiClient espClient;
PubSubClient client(espClient);

void connectarWiFi() {
WiFi.begin(ssid, password);
Serial.print("Conectando à rede Wi-Fi");
int tentativas = 0;
while (WiFi.status() != WL_CONNECTED && tentativas < 20) {
    delay(1000);
    Serial.print(".");
    tentativas++;
}
if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nConectado ao Wi-Fi");
} else {
    Serial.println("\nFalha na conexão Wi-Fi");
}
}

void callback(char* topic, byte* payload, unsigned int length) {
// Processar mensagem recebida
String mensagem;
for (unsigned int i = 0; i < length; i++) {
```

```
    mensagem += (char)payload[i];
}
Serial.println("Mensagem recebida no tópico [" + String(topic) + "]: " + mensa

// Controlar o LED com base na mensagem
if (mensagem == "ON") {
    digitalWrite(LED_PIN, HIGH);
    Serial.println("LED ligado");
} else if (mensagem == "OFF") {
    digitalWrite(LED_PIN, LOW);
    Serial.println("LED desligado");
}
}

void reconnectMQTT() {
while (!client.connected()) {
    Serial.println("Conectando ao broker MQTT...");
    String clientId = "ESP32Client-" + String(random(0xffff), HEX);
    if (client.connect(clientId.c_str(), mqtt_user, mqtt_password)) {
    Serial.println("Conectado ao broker MQTT");
    // Assinar o tópico de comandos
    client.subscribe(mqtt_topic_comando, 1); // QoS 1
    } else {
    Serial.print("Falha na conexão, rc=");
    Serial.print(client.state());
    delay(5000);
    }
}
}

void setup() {
Serial.begin(115200);
dht.begin();
pinMode(LED_PIN, OUTPUT); // Configurar o pino do LED como saída
digitalWrite(LED_PIN, LOW); // Iniciar com LED desligado
connectarWiFi();
client.setServer(mqtt_server, mqtt_port);
client.setCallback(callback); // Definir a função de callback para mensagens r
}

void loop() {
if (WiFi.status() != WL_CONNECTED) {
    connectarWiFi();
}
if (!client.connected()) {
    reconnectMQTT();
}
client.loop();

// Ler e publicar dados do sensor
float temperatura = dht.readTemperature();
if (!isnan(temperatura)) {
    String payload = String(temperatura);
    client.publish(mqtt_topic_sensor, payload.c_str(), true); // QoS 1, mensag
    Serial.println("Temperatura publicada: " + payload + " °C");
```

```
    } else {
        Serial.println("Erro ao ler o sensor DHT11!");
    }

    delay(5000); // Publica e verifica mensagens a cada 5 segundos
    }
```

4. Explicação do Código

   - **Configuração inicial:**
     - O pino do LED (GPIO 5) é configurado como saída com `pinMode(LED_PIN, OUTPUT)`.
     - A função `connectarWiFi()` implementa a reconexão automática ao Wi-Fi, com até 20 tentativas.
   - **Conexão MQTT:**
     - A função `reconnectMQTT()` conecta ao broker e assina o tópico `esp32/comando` com QoS 1.
     - A função `client.setCallback(callback)` define a função que processa mensagens recebidas.
   - **Leitura do sensor:**
     - O sensor DHT11 é lido a cada 5 segundos, e a temperatura é publicada no tópico `esp32/temperatura` com QoS 1 e mensagem retida.
   - **Processamento de comandos:**
     - A função `callback` é chamada quando uma mensagem é recebida no tópico `esp32/comando`.
     - A mensagem é convertida em uma string e comparada com "ON" ou "OFF" para ligar ou desligar o LED usando `digitalWrite`.
   - **Tópicos:**
     - `esp32/temperatura`: Para publicar dados do sensor.
     - `esp32/comando`: Para receber comandos do atuador.

5. Teste do Controle do Atuador

   - **Carregue o código:** Compile e carregue o código no ESP32 usando o Arduino IDE.
   - **Monitore via Serial:** Abra o Monitor Serial (115200 baud) para verificar conexões Wi-Fi e MQTT, leituras do sensor e comandos recebidos.
   - **Envie comandos MQTT:** Use um cliente MQTT (ex.: MQTT Explorer ou `mosquitto_pub`) para publicar comandos no tópico `esp32/comando`:
     - `mosquitto_pub -h SEU_BROKER -t esp32/comando -m "ON"`
     - `mosquitto_pub -h SEU_BROKER -t esp32/comando -m "OFF"`
   - **Verifique o atuador:** Confirme se o LED liga ou desliga conforme os comandos enviados. No Monitor Serial, você verá mensagens como "LED ligado" ou "LED desligado".
   - **Debug:** Verifique erros no Monitor Serial, como falhas de conexão (`WL_NO_SSID_AVAIL`, `WL_CONNECT_FAILED`) ou problemas com o broker MQTT (`client.state()`).

6. Protocolo MQTT e Controle de Atuadores

   O MQTT é ideal para controle de atuadores devido ao seu modelo publicador/assinante:

- **Publicadores:** Um dispositivo ou aplicação (ex.: um aplicativo web ou celular) publica comandos como "ON" ou "OFF" no tópico `esp32/comando`.
- **Assinantes:** O ESP32 assina o tópico e recebe os comandos, atuando sobre o GPIO.
- **Broker:** Gerencia a entrega das mensagens, suportando QoS (neste caso, QoS 1 para entrega garantida) e mensagens retidas (a última mensagem é armazenada para novos assinantes).
- **Vantagens:** Baixo consumo de banda, reconexão automática em caso de falhas de rede e suporte a dispositivos com recursos limitados, como o ESP32.

7. Segurança (Opcional)

   - Use MQTT com TLS/SSL (porta 8883) para proteger os comandos, configurando certificados no ESP32.
   - Adicione autenticação com usuário e senha no broker.
   - Considere validar os comandos recebidos (ex.: aceitar apenas "ON" ou "OFF") para evitar ações indesejadas.

8. Vantagens do ESP32

   O ESP32 é ideal para este projeto devido ao seu processador dual-core, Wi-Fi integrado e múltiplos pinos GPIO, que permitem controlar atuadores e ler sensores simultaneamente. Comparado ao ESP8266 (single-core, menos pinos) ou Arduino Uno (sem Wi-Fi nativo), o ESP32 é mais versátil para IoT.

9. Conclusão

   Este texto detalha como configurar um ESP32 para receber comandos MQTT e controlar um atuador (LED no GPIO 5), mantendo a funcionalidade de publicar dados de um sensor (DHT11). O código inclui reconexão automática ao Wi-Fi e MQTT, assinatura de tópicos com QoS 1 e processamento de comandos. Para testar, envie comandos via um cliente MQTT e verifique o comportamento do atuador.