Roteiro Nº 04	**Utilização do ESP32 como Servidor Web em uma LAN**	
Curso Engenharia Elétrica	**Disciplina** ELT 1119 – Redes e Aplicações IoT	**Professor** Carlos Alberto Vasconcelos Bezerra
Nome do Estudante		**Data**

1. Objetivos da Aprendizagem

- Compreender o funcionamento do ESP32 como servidor web.
- Configurar uma rede Wi-Fi local para o ESP32.
- Implementar um servidor web dinâmico que exibe dados de um sensor.
- Utilizar requisições HTTP para atualizar dados em uma página web.
- Desenvolver habilidades práticas na programação do ESP32 utilizando MicroPython.

2. Material Necessário

- Placa de desenvolvimento ESP32-DevKitC-V4;
- Computador com ambiente de desenvolvimento Thonny instalado;
- Cabo USB para comunicação entre o ESP32 e o computador;
- Sensor de temperatura LM35;
- Resistores 240Ω, 5kΩ;
- LED 5mm.

3. Fundamentação Teórica

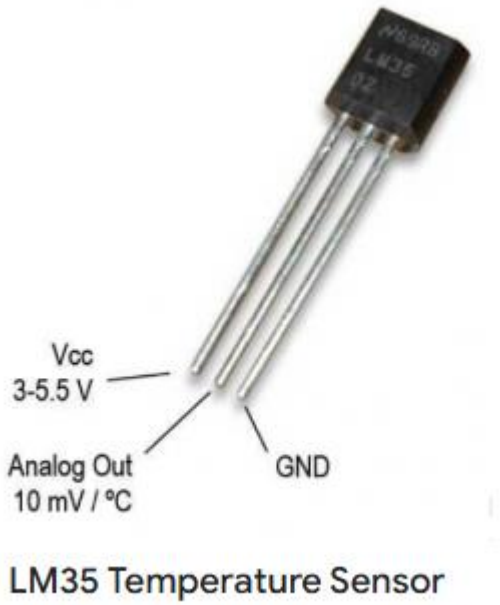
O ESP32 é um microcontrolador poderoso, equipado com conectividade Wi-Fi e Bluetooth, permitindo sua utilização em aplicações de IoT. Um dos principais usos é a implementação de servidores web para monitoramento e controle remoto de dispositivos.
Em um servidor web, o ESP32 atua como um ponto de acesso (AP) ou como cliente de uma rede Wi-Fi (STA), recebendo requisições HTTP de dispositivos na mesma rede e respondendo com páginas HTML. No experimento, utilizaremos o ESP32 para coletar dados do sensor de temperatura LM35 e exibi-los em uma página web.
A comunicação entre o ESP32 e os dispositivos na rede ocorre por meio do protocolo HTTP. O cliente (navegador) envia requisições para o servidor (ESP32), que responde com o conteúdo HTML, possibilitando a visualização dos dados em tempo real.

4. Procedimento Experimental

4.1 Montagem do circuito:

Conecte o sensor LM35 ao ESP32:

VCC → 3.3V
GND → GND
Saída do LM35 → Pino 34 (ADC) – (Conecte um resistor de 5kΩ entre este pino e o GND)



LM35 Temperature Sensor

4.2 Conecte um LED em série com um resistor de 240Ω ao pino 2 do ESP32 para indicar a atualização da página web.

5. Configuração do ESP32:

5.1 Conecte-se ao ESP32 via Thonny IDE.
5.2 Copie e execute o código abaixo para inicializar o servidor web.

```
import network
import socket
import machine
import time

# Configuração do Wi-Fi
SSID = "Alencar Oi 2.4G"
PASSWORD = "Carloshenrique"

# Conectar ao Wi-Fi
station = network.WLAN(network.STA_IF)
station.active(True)
station.connect(SSID, PASSWORD)

while not station.isconnected():
    pass

print("Conectado ao Wi-Fi")
print("Endereço IP:", station.ifconfig()[0])

# Configuração do sensor LM35
adc = machine.ADC(machine.Pin(34))
adc.atten(machine.ADC.ATTN_11DB)  # Configura a atenuação para leitura de 0 a 3,3V
```

```
# Configuração do LED
led = machine.Pin(2, machine.Pin.OUT)

# Função para ler temperatura com média de 10 amostras
def read_temperature():
    total = 0
    samples = 10  # Número de leituras para a média
    for _ in range(samples):
        time.sleep(0.05)  # Pequeno atraso entre leituras
        voltage = adc.read() * (3.3 / 4095)  # Converte leitura ADC para tensão
        temperature = voltage * 100  # Conversão para temperatura em Celsius (LM35: 10mV/°C)
        total += temperature
    return round(total / samples, 2)  # Retorna a média das leituras

# Página HTML com AJAX para atualização dinâmica
def web_page():
    html = """
    <!DOCTYPE html>
    <html>
    <head>
        <title>Monitoramento de Temperatura</title>
        <meta charset="UTF-8">
        <script>
            function updateTemperature() {
                var xhr = new XMLHttpRequest();
                xhr.onreadystatechange = function() {
                    if (xhr.readyState == 4 && xhr.status == 200) {
                        document.getElementById("temp_value").innerHTML = xhr.responseText;
                    }
                };
                xhr.open("GET", "/temp", true);
                xhr.send();
            }
            setInterval(updateTemperature, 1000);
        </script>
        <style>
            body { font-family: Arial, sans-serif; text-align: center; }
            .temp { font-size: 50px; color: #ff6600; }
        </style>
    </head>
    <body>
        <h1>Monitoramento de Temperatura - ESP32</h1>
        <p class="temp" id="temp_value">-- °C</p>
    </body>
    </html>
    """
    return html

# Criar servidor web
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind(('', 80))
s.listen(5)

print("Servidor Web Iniciado...")
```

```
# Laço principal
while True:
    led.on()  # Liga o LED indicando a atualização da página
    conn, addr = s.accept()
    print("Conexão de:", addr)
    request = conn.recv(1024).decode()

    if request.startswith("GET /temp"):
        temp = read_temperature()
        response = "HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nConnection: close\r\n\r\n" + str(temp) + " °C"
    else:
        response = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\nConnection: close\r\n\r\n" + web_page()

    conn.sendall(response.encode())
    conn.close()
    led.off()  # Desliga o LED após a atualização da página
    time.sleep(1)  # Aguarda 1 segundo antes da próxima atualização
```

6. Execução do servidor web:

6.1 Configure o ESP32 para conectar-se a uma rede Wi-Fi local.
6.2 Observe o endereço IP do ESP32 exibido no terminal.
6.3 Acesse esse endereço em um navegador web na mesma rede.
6.4 Observe a exibição da temperatura e o LED piscando a cada atualização.

7. Exercícios Propostos:

7.1 Explique como o ESP32 manipula requisições HTTP para atualizar a página web.
7.2 Modifique o intervalo de atualização da página de 1 segundo para 2 segundos. O que acontece?
7.3 Adicione um segundo sensor (exemplo: DHT11) e exiba tanto a temperatura quanto a umidade na página web.
7.4 Implemente um botão virtual na página web para acionar um LED remotamente.
7.5 Como a implementação de um servidor web pode ser aplicada em sistemas IoT?

6 Simulação

Simular os exercícios propostos utilizando o simulador wokwi no link:

https://wokwi.com/projects/new/esp32

Bibliografia

ASCHER, D.; LUTZ, M. Aprendendo Python. Porto Alegre: Bookman, 2007.

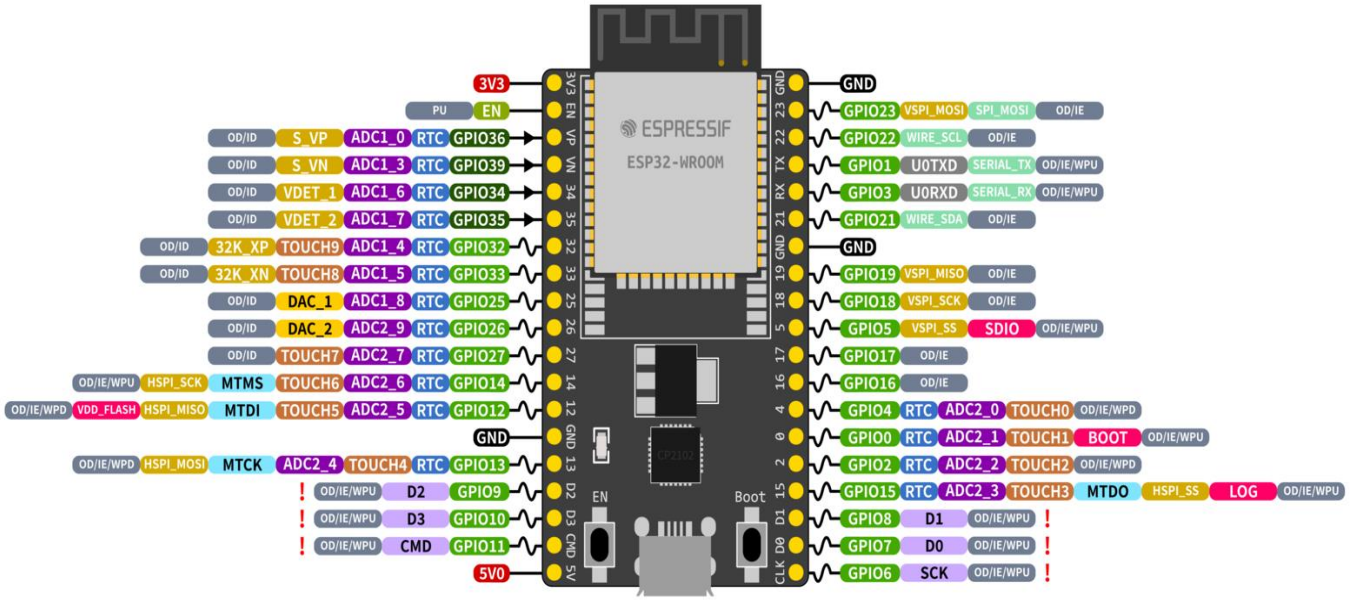
MENEZES, N. N. C. Introdução à programação com Python. São Paulo: Ed. Novatec, 2014.

OLIVEIRA, S. Internet das coisas com ESP8266, Arduino e Raspberry Pi. São Paulo: Ed. Novatec, 2017.

OLIVEIRA, C. L. V. IoT com Micropython e NodeMCU. São Paulo: Ed. Novatec, 2022.

ESP32-DevKitC

ESPRESSIF



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
DEBUG JTAG for Debugging
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions

RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory *Can't be used as regular GPIO*

GPIO STATE
WPU: *Weak Pull-up (Internal)*
WPD: *Weak Pull-down (Internal)*
PU: *Pull-up (External)*
IE: *Input Enable (After Reset)*
ID: *Input Disabled (After Reset)*
OE: *Output Enable (After Reset)*
OD: *Output Disabled (After Reset)*