**ServidorRPiPICOW**

1. **Server Blink**

* **client**

# Programa para conectar un dispositivo a una red Wi-Fi y controlar un LED mediante comandos TCP.

# Importa la biblioteca 'network' para gestionar la conexión a la red Wi-Fi.

import network

# Importa la biblioteca 'socket' para la comunicación TCP con el servidor.

import socket

# Importa la biblioteca 'machine' para manejar el hardware, como el pin del LED.

import machine

# Importa la biblioteca 'time' para controlar pausas en el código.

import time

class WiFiClient:

"""

Clase para manejar la conexión Wi-Fi y el control de un LED por comandos TCP.

"""

def \_\_init\_\_(self, ssid, password, server\_ip, port=3001):

"""

Inicializa el cliente Wi-Fi con los datos de conexión y configura el LED.

Parámetros:

ssid (str): Nombre de la red Wi-Fi.

password (str): Contraseña de la red Wi-Fi.

server\_ip (str): Dirección IP del servidor TCP.

port (int): Puerto del servidor TCP, por defecto es 3001.

"""

self.ssid = ssid # Guarda el nombre de la red Wi-Fi

self.password = password # Guarda la contraseña de la red Wi-Fi

self.server\_ip = server\_ip # Guarda la IP del servidor

self.port = port # Guarda el puerto del servidor TCP

# Configura el Wi-Fi en modo cliente (STA) y el pin LED como salida

self.wlan = network.WLAN(network.STA\_IF)

self.led = machine.Pin("LED", machine.Pin.OUT)

def connect\_wifi(self):

"""

Conecta el dispositivo a la red Wi-Fi.

Activa el Wi-Fi y se conecta a la red usando el SSID y contraseña dados.

"""

self.wlan.active(True) # Activa el Wi-Fi en modo cliente

self.wlan.connect(self.ssid, self.password) # Conecta a la red Wi-Fi

# Espera hasta que el dispositivo esté conectado a la red Wi-Fi

while not self.wlan.isconnected():

time.sleep(1) # Pausa de 1 segundo para evitar intentos rápidos

print('Wi-Fi conectado:', self.wlan.ifconfig()) # Muestra la configuración de red

def start\_client(self):

"""

Inicia la conexión TCP con el servidor y controla el LED según los comandos recibidos.

"""

# Crea un socket TCP y conecta al servidor

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((self.server\_ip, self.port))

print('Conectado al servidor') # Indica que la conexión fue exitosa

# Bucle principal para recibir y procesar comandos del servidor

while True:

# Espera recibir un comando del servidor

command = client\_socket.recv(1024).decode()

# Enciende el LED si el comando es "on"

if command == 'on':

self.led.value(1)

# Apaga el LED si el comando es "off"

elif command == 'off':

self.led.value(0)

def main():

"""

Función principal que configura y ejecuta el cliente Wi-Fi.

"""

ssid = 'SSID' # Nombre de la red Wi-Fi

password = 'PASS' # Contraseña de la red Wi-Fi

server\_ip = 'IP' # Dirección IP del servidor TCP

# Crea la instancia del cliente Wi-Fi y conecta a la red

wifi\_client = WiFiClient(ssid, password, server\_ip)

wifi\_client.connect\_wifi() # Conecta a la red Wi-Fi

wifi\_client.start\_client() # Inicia la conexión TCP y controla el LED

# Ejecuta la función principal si el archivo es ejecutado directamente

if \_\_name\_\_ == "\_\_main\_\_":

main()

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

* **Server**

// Importa el módulo HTTP de Node.js para crear un servidor HTTP.

const http = require('http');

// Importa el módulo del sistema de archivos para interactuar con archivos del sistema.

const fs = require('fs');

// Importa la biblioteca WebSocket para manejar conexiones WebSocket.

const WebSocket = require('ws');

// Importa el módulo de red de Node.js para crear un servidor TCP.

const net = require('net');

// Importa el módulo de rutas para manejar rutas de archivos.

const path = require('path');

// Define la dirección IP y los puertos del servidor.

const hostname = 'IP'; // Dirección IP del servidor.

const httpPort = 3000; // Puerto para el servidor HTTP.

const tcpPort = 3001; // Puerto para el servidor TCP.

// Crea un servidor HTTP para servir un archivo HTML o mostrar un error 404 si la URL no existe.

const server = http.createServer((req, res) => {

if (req.url === '/') { // Si la URL solicitada es la raíz

fs.readFile(path.join(\_\_dirname, 'index.html'), (err, data) => { // Lee el archivo index.html

res.writeHead(err ? 500 : 200, { 'Content-Type': 'text/html' }); // Establece el código de estado y tipo de contenido

res.end(err ? 'ERROR' : data); // Envía el contenido del archivo o un mensaje de error

});

} else { // Si la URL no es la raíz

res.writeHead(404).end('404 NOT FOUND'); // Responde con un error 404

}

});

// Crea un servidor WebSocket que reutiliza el servidor HTTP

const wss = new WebSocket.Server({ server }); // Inicializa el servidor WebSocket

const tcpClients = []; // Array para almacenar los clientes TCP conectados

// Maneja conexiones WebSocket y reenvía mensajes a los clientes TCP

wss.on('connection', ws => { // Cuando un cliente WebSocket se conecta

ws.on('message', message => { // Cuando se recibe un mensaje de WebSocket

tcpClients.forEach(client => client.write(message)); // Envía el mensaje a todos los clientes TCP conectados

});

});

// Crea un servidor TCP para manejar conexiones y comandos de clientes TCP

net.createServer(socket => { // Inicializa el servidor TCP

tcpClients.push(socket); // Agrega el nuevo cliente TCP al array de clientes

socket.on('data', data => { // Cuando se recibe datos del cliente TCP

const command = data.toString().trim(); // Convierte los datos a cadena y elimina espacios en blanco

// Responde según el comando recibido

socket.write(command === 'on' ? 'LED encendido' : command === 'off' ? 'LED apagado' : 'Comando no reconocido');

});

}).listen(tcpPort); // Escucha en el puerto TCP definido

// Inicia el servidor HTTP y muestra un mensaje en la consola

server.listen(httpPort, hostname, () => console.log(`Servidor HTTP en http://${hostname}:${httpPort}/`));

**Diagrama

Descripción generada automáticamente**

**Diagrama

Descripción generada automáticamente**

* **Index**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Control</title>

<style>

body {

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

font-family: Arial, sans-serif;

background-color: #f9f9f9;

height: 100vh;

margin: 0;

color: #333;

}

.buttonon {

background-color: green;

color: #fff;

border: none;

padding: 20px 50px;

margin: 5px;

font-size: 1em;

cursor: pointer;

}

.buttonoff {

background-color: red;

color: #fff;

border: none;

padding: 20px 50px;

margin: 5px;

font-size: 1em;

cursor: pointer;

}

#response {

margin-top: 15px;

font-size: 1em;

}

</style>

</head>

<body>

<p style="color:blue">Control de luces</p>

<button class="buttonon" onclick="sendMessage('on')">Encender</button>

<button class="buttonoff" onclick="sendMessage('off')">Apagar</button>

<p id="response">Presiona un boton.</p>

<script>

const socket = new WebSocket(`ws://${window.location.hostname}:3000`);

socket.onopen = () => {

console.log('Conexiï¿½n WebSocket establecida');

};

socket.onmessage = (event) => {

document.getElementById('response').innerText = event.data;

};

socket.onerror = (error) => {

console.error('Error de WebSocket:', error);

};

function sendMessage(message) {

if (socket.readyState === WebSocket.OPEN) {

socket.send(message);

}

}

</script>

</body>

</html>

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

1. **Server RGB**

* **Client**

# Importación de módulos

import network # Para la configuración y conexión a la red Wi-Fi

import socket # Para la comunicación TCP

import machine # Para el control de hardware (pines PWM del LED RGB)

import time # Para funciones de temporización

import json # Para manejar datos en formato JSON

# Configuración de la red Wi-Fi

SSID = 'Tenda\_73E5E0' # Nombre de la red Wi-Fi

PASSWORD = 'b6ksnu2YNd' # Contraseña de la red Wi-Fi

# Conectar a la red Wi-Fi

wlan = network.WLAN(network.STA\_IF) # Configura el Wi-Fi en modo cliente

wlan.active(True) # Activa la interfaz Wi-Fi

wlan.connect(SSID, PASSWORD) # Conecta a la red Wi-Fi usando SSID y contraseña

# Esperar hasta que el dispositivo esté conectado a la red

while not wlan.isconnected():

time.sleep(1) # Pausa de 1 segundo antes de volver a comprobar

# Imprimir la configuración de red del dispositivo

print('Conectado a la red:', wlan.ifconfig())

# Configuración de los pines del LED RGB como PWM

red\_pin = machine.PWM(machine.Pin(0)) # Configura el pin rojo para PWM

green\_pin = machine.PWM(machine.Pin(1)) # Configura el pin verde para PWM

blue\_pin = machine.PWM(machine.Pin(2)) # Configura el pin azul para PWM

# Establecer la frecuencia de PWM en 1000 Hz para los pines del LED RGB

red\_pin.freq(1000)

green\_pin.freq(1000)

blue\_pin.freq(1000)

# Configuración del cliente TCP

TCP\_IP = '192.168.0.223' # Dirección IP del servidor TCP (por ejemplo, una Raspberry Pi)

TCP\_PORT = 3001 # Puerto del servidor TCP

# Función para ajustar el color del LED RGB

# Los valores r, g y b están en el rango 0-255

def set\_rgb(r, g, b):

red\_pin.duty\_u16(int(r \* 65535 / 255)) # Convierte el valor de rojo a un rango de 0 a 65535 y ajusta el pin rojo

green\_pin.duty\_u16(int(g \* 65535 / 255)) # Convierte el valor de verde a un rango de 0 a 65535 y ajusta el pin verde

blue\_pin.duty\_u16(int(b \* 65535 / 255)) # Convierte el valor de azul a un rango de 0 a 65535 y ajusta el pin azul

# Función para conectarse al servidor TCP y recibir valores RGB

def receive\_rgb():

# Crear un socket TCP

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((TCP\_IP, TCP\_PORT)) # Conectar al servidor TCP usando IP y puerto

# Bucle principal para recibir y procesar datos

while True:

data = client\_socket.recv(1024) # Recibe datos (hasta 1024 bytes) del servidor

if data:

try:

# Decodifica los datos recibidos en formato JSON y ajusta el LED RGB

rgb = json.loads(data.decode())

set\_rgb(rgb['r'], rgb['g'], rgb['b']) # Ajusta el LED RGB usando los valores recibidos

print('Valores RGB recibidos:', rgb) # Muestra en consola los valores recibidos

except Exception as e:

# Si ocurre un error al procesar los datos, muestra un mensaje de error

print('Error al procesar datos:', e)

# Cierra el socket al finalizar (aunque en este caso no se llega a esta línea por el bucle infinito)

client\_socket.close()

# Iniciar la recepción de valores RGB llamando a la función

receive\_rgb()

Diagrama

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

* **Server**

// Importación de módulos necesarios

const http = require('http'); // Para crear el servidor HTTP

const fs = require('fs'); // Para leer archivos del sistema

const WebSocket = require('ws'); // Para manejar conexiones WebSocket

const net = require('net'); // Para crear el servidor TCP

const path = require('path'); // Para manejar rutas de archivos

// Configuración de la dirección y puertos del servidor

const hostname = '0.0.0.0'; // Dirección IP del servidor

const httpPort = 3000; // Puerto para el servidor HTTP

const tcpPort = 3001; // Puerto para el servidor TCP

// Objeto que almacena los valores RGB iniciales

let rgbValues = { r: 0, g: 0, b: 0 };

// Crear servidor HTTP

const server = http.createServer((req, res) => {

// Manejo de solicitudes HTTP a la raíz "/"

if (req.url === '/') {

// Leer y servir el archivo `index.html`

fs.readFile(path.join(\_\_dirname, 'index.html'), (err, data) => {

res.writeHead(err ? 500 : 200, { 'Content-Type': 'text/html' });

res.end(err ? 'ERROR' : data);

});

} else {

// Responder con un error 404 si la URL no es "/"

res.writeHead(404).end('404 NOT FOUND');

}

});

// Crear servidor WebSocket asociado al servidor HTTP

const wss = new WebSocket.Server({ server });

// Manejo de conexiones WebSocket

wss.on('connection', ws => {

// Enviar los valores RGB actuales al cliente WebSocket

ws.send(JSON.stringify(rgbValues));

// Recibir nuevos valores RGB desde el cliente WebSocket

ws.on('message', message => {

rgbValues = JSON.parse(message); // Actualizar valores RGB

console.log('Nuevos valores RGB:', rgbValues);

// Enviar los valores actualizados a todos los clientes TCP conectados

tcpClients.forEach(client => {

client.write(JSON.stringify(rgbValues));

});

});

});

// Almacena los clientes TCP conectados

const tcpClients = new Set();

// Crear servidor TCP para la conexión con dispositivos como Pico W

net.createServer(socket => {

console.log('Cliente TCP (Pico W) conectado');

tcpClients.add(socket); // Agregar el nuevo cliente TCP al conjunto

// Manejo de desconexión del cliente TCP

socket.on('end', () => {

console.log('Cliente TCP (Pico W) desconectado');

tcpClients.delete(socket); // Eliminar el cliente desconectado del conjunto

});

}).listen(tcpPort); // Escuchar en el puerto TCP definido

// Iniciar el servidor HTTP

server.listen(httpPort, hostname, () => console.log(`Servidor HTTP en http://${hostname}:${httpPort}/`));

Tabla

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* **Index**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Control LED RGB</title>

<style>

body {

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

font-family: Arial, sans-serif;

background-color: #f9f9f9;

height: 100vh;

margin: 0;

color: #333;

}

.color-picker, .slider-container {

margin: 20px 0;

}

.slider {

width: 200px;

}

</style>

</head>

<body>

<h1>Control LED RGB</h1>

<input type="color" id="colorPicker" class="color-picker">

<div class="slider-container">

<input type="range" id="redSlider" class="slider" min="0" max="255" value="0">

<label for="redSlider">R: <span id="redValue">0</span></label>

</div>

<div class="slider-container">

<input type="range" id="greenSlider" class="slider" min="0" max="255" value="0">

<label for="greenSlider">G: <span id="greenValue">0</span></label>

</div>

<div class="slider-container">

<input type="range" id="blueSlider" class="slider" min="0" max="255" value="0">

<label for="blueSlider">B: <span id="blueValue">0</span></label>

</div>

<div id="rgbValues"></div>

<script>

const socket = new WebSocket(`ws://${window.location.hostname}:3000`);

const colorPicker = document.getElementById('colorPicker');

const redSlider = document.getElementById('redSlider');

const greenSlider = document.getElementById('greenSlider');

const blueSlider = document.getElementById('blueSlider');

const redValue = document.getElementById('redValue');

const greenValue = document.getElementById('greenValue');

const blueValue = document.getElementById('blueValue');

const rgbValues = document.getElementById('rgbValues');

socket.onopen = () => {

console.log('Conexiï¿½n WebSocket establecida');

};

socket.onmessage = (event) => {

const data = JSON.parse(event.data);

updateColorPicker(data);

updateSliders(data);

};

colorPicker.addEventListener('input', updateFromColorPicker);

redSlider.addEventListener('input', updateFromSliders);

greenSlider.addEventListener('input', updateFromSliders);

blueSlider.addEventListener('input', updateFromSliders);

function updateFromColorPicker() {

const color = colorPicker.value;

const rgb = hexToRgb(color);

updateSliders(rgb);

sendRgbValues(rgb);

}

function updateFromSliders() {

const rgb = {

r: parseInt(redSlider.value),

g: parseInt(greenSlider.value),

b: parseInt(blueSlider.value)

};

updateColorPicker(rgb);

sendRgbValues(rgb);

}

function hexToRgb(hex) {

const r = parseInt(hex.slice(1, 3), 16);

const g = parseInt(hex.slice(3, 5), 16);

const b = parseInt(hex.slice(5, 7), 16);

return { r, g, b };

}

function updateColorPicker(rgb) {

const hex = `#${rgb.r.toString(16).padStart(2, '0')}${rgb.g.toString(16).padStart(2, '0')}${rgb.b.toString(16).padStart(2, '0')}`;

colorPicker.value = hex;

updateRgbValues(rgb);

}

function updateSliders(rgb) {

redSlider.value = rgb.r;

greenSlider.value = rgb.g;

blueSlider.value = rgb.b;

redValue.textContent = rgb.r;

greenValue.textContent = rgb.g;

blueValue.textContent = rgb.b;

}

function updateRgbValues(rgb) {

rgbValues.textContent = `R: ${rgb.r}, G: ${rgb.g}, B: ${rgb.b}`;

}

function sendRgbValues(rgb) {

socket.send(JSON.stringify(rgb));

}

</script>

</body>

</html>

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

1. **Server RTC**

* **Client**

# Importación de bibliotecas necesarias

import network # Para manejar la conexión Wi-Fi

import socket # Para la comunicación TCP

import machine # Para el manejo de hardware

import time # Para funciones de espera

from sh1106 import SH1106\_I2C # Para manejar la pantalla OLED SH1106

# Configuración de la red Wi-Fi

SSID = 'Tenda\_73E5E0' # Nombre de la red Wi-Fi

PASSWORD = 'b6ksnu2YNd' # Contraseña de la red Wi-Fi

# Configuración y conexión a la red Wi-Fi

wlan = network.WLAN(network.STA\_IF)

wlan.active(True)

wlan.connect(SSID, PASSWORD)

# Esperar a que se conecte

while not wlan.isconnected():

time.sleep(1)

print('Conectado a la red:', wlan.ifconfig())

# Configuración del botón

button\_pin = machine.Pin(15, machine.Pin.IN, machine.Pin.PULL\_UP) # Pin configurado como entrada con resistencia de pull-up

count = 0 # Inicialización del contador de pulsaciones

# Configuración del cliente TCP

TCP\_IP = '192.168.0.223' # Dirección IP del servidor

TCP\_PORT = 3001 # Puerto del servidor

# Configuración de la pantalla I2C

i2c = machine.I2C(0, scl=machine.Pin(1), sda=machine.Pin(0)) # Configuración del bus I2C

oled = SH1106\_I2C(128, 64, i2c) # Inicialización de la pantalla OLED

# Función de interrupción para el botón

def button\_handler(pin):

global count

count += 1 # Incrementar el conteo

print('Botón presionado, conteo:', count)

# Configuración de la interrupción en el botón

button\_pin.irq(trigger=machine.Pin.IRQ\_FALLING, handler=button\_handler)

# Función para enviar el conteo al servidor TCP y mostrarlo en pantalla

def send\_count():

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # Creación del socket

client\_socket.connect((TCP\_IP, TCP\_PORT)) # Conectar al servidor

while True:

client\_socket.send(str(count).encode()) # Enviar el conteo

# Actualizar el conteo en la pantalla

oled.fill(0) # Limpiar pantalla

oled.text('Conteo:', 0, 0)

oled.text(str(count), 0, 10)

oled.show() # Actualizar pantalla

time.sleep(1) # Enviar cada segundo

client\_socket.close() # Cerrar el socket (no se alcanza en este caso)

send\_count() # Iniciar la función de envío de conteo

**Diagrama

Descripción generada automáticamente**

**Diagrama

Descripción generada automáticamente**

* **Server**

// Importación de módulos necesarios

const http = require('http'); // Módulo HTTP para crear el servidor

const fs = require('fs'); // Módulo de sistema de archivos para leer archivos

const WebSocket = require('ws'); // Biblioteca WebSocket para manejar conexiones WebSocket

const net = require('net'); // Módulo de red para crear un servidor TCP

const path = require('path'); // Módulo para manejar rutas de archivos

// Configuración del servidor

const hostname = '0.0.0.0'; // IP en todas las interfaces de red

const httpPort = 3000; // Puerto para el servidor HTTP

const tcpPort = 3001; // Puerto para el servidor TCP

// Contador para almacenar el valor actual

let count = 0;

// Creación del servidor HTTP

const server = http.createServer((req, res) => {

if (req.url === '/') { // Verifica si la solicitud es la raíz

fs.readFile(path.join(\_\_dirname, 'index.html'), (err, data) => { // Lee el archivo index.html

res.writeHead(err ? 500 : 200, { 'Content-Type': 'text/html' }); // Envía código de estado y tipo de contenido

res.end(err ? 'ERROR' : data); // Envía el contenido o el error

});

} else {

res.writeHead(404).end('404 NOT FOUND'); // Respuesta 404 si no es la raíz

}

});

// Creación del servidor WebSocket

const wss = new WebSocket.Server({ server });

// Manejo de conexiones WebSocket

wss.on('connection', ws => {

ws.send(JSON.stringify({ count })); // Envía el valor inicial del contador

// Envía actualizaciones del contador cada segundo

const interval = setInterval(() => {

ws.send(JSON.stringify({ count }));

}, 1000);

// Limpia el intervalo al cerrar la conexión

ws.on('close', () => {

clearInterval(interval);

});

});

// Creación del servidor TCP

net.createServer(socket => {

console.log('Cliente TCP conectado');

socket.on('data', data => { // Recibe datos del cliente TCP

count = parseInt(data.toString().trim(), 10); // Actualiza el contador

console.log('Conteo recibido:', count);

// Notifica a todos los clientes WebSocket el nuevo valor del contador

wss.clients.forEach(client => {

if (client.readyState === WebSocket.OPEN) {

client.send(JSON.stringify({ count }));

}

});

});

// Mensaje al desconectarse el cliente TCP

socket.on('end', () => {

console.log('Cliente TCP desconectado');

});

}).listen(tcpPort);

// Inicia el servidor HTTP

server.listen(httpPort, hostname, () => console.log(`Servidor HTTP en http://${hostname}:${httpPort}/`));

Diagrama

Descripción generada automáticamente Diagrama

Descripción generada automáticamente

* **Index**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Conteo de Interrupciones</title>

<style>

body {

display: flex;

align-items: center;

justify-content: center;

flex-direction: column;

font-family: Arial, sans-serif;

background-color: #f9f9f9;

height: 100vh;

margin: 0;

color: #333;

}

#count {

font-size: 2em;

margin-top: 20px;

}

</style>

</head>

<body>

<h1>Conteo de Interrupciones</h1>

<div id="count">0</div> <!-- Mostrar el conteo aquí -->

<script>

const socket = new WebSocket(`ws://${window.location.hostname}:3000`);

socket.onopen = () => {

console.log('Conexión WebSocket establecida');

};

socket.onmessage = (event) => {

const data = JSON.parse(event.data);

document.getElementById('count').innerText = data.count; // Actualiza el conteo en la página

};

socket.onerror = (error) => {

console.error('Error de WebSocket:', error);

};

</script>

</body>

</html>

Tabla

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente