**serverSerialRTC**

* **server**

// Carga el módulo Express para manejar la lógica del servidor web.

const express = require('express');

// Crea una instancia de una aplicación Express.

const app = express();

// Carga el módulo HTTP y crea un servidor HTTP usando la instancia de Express.

const http = require('http');

const server = http.createServer(app);

// Carga el módulo Socket.IO y crea un servidor de WebSocket enlazado al servidor HTTP.

const { Server } = require("socket.io");

const io = new Server(server);

// Carga el módulo SerialPort y configura la conexión al puerto serial para recibir datos.

const { SerialPort, ReadlineParser } = require('serialport');

const port = new SerialPort({

path: '/dev/ttyACM0',

baudRate: 9600 });

// Crea un parser que procesará los datos del puerto serial basándose en delimitadores específicos.

const parser = port.pipe(new ReadlineParser({ delimiter: '\r\n' }));

// Configura la aplicación Express para servir archivos estáticos desde el directorio 'public'.

app.use(express.static('public'));

// Define una ruta para la raíz que simplemente envía "Hello World!" al navegador.

app.get('/', (req, res) => {

res.send('Hello World!');

});

// Maneja eventos de conexión WebSocket.

io.on('connection', (socket) => {

console.log('a user connected');

// Establece un manejador para los datos recibidos a través del puerto serial.

parser.on('data', (data) => {

// Escribe los datos recibidos en la consola del servidor y los transmite a los clientes conectados.

process.stdout.write(data + "\r");

socket.emit('time', data); // Envia el estado recibido al cliente conectado.

});

});

// Inicia el servidor en el puerto 8080 y registra un mensaje en la consola.

server.listen(8080, () => {

console.log('listening on \*:8080');

});

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

* **rtc**

# Importación del módulo Timer y machine para controlar los GPIOs.

from machine import Timer

import machine

# Inicialización de una variable global 'count' que se usa como contador.

count = 0

# Configuración del pin del LED como salida.

# 'led\_pin' es una instancia de la clase 'Pin' con el pin 25 configurado como salida.

led\_pin = machine.Pin(25, machine.Pin.OUT)

# Definición de la función 'blink' que actúa como callback para el temporizador.

# Esta función toma un objeto 'timer' y cambia el estado del LED.

def blink(timer):

# Cambia el valor del pin del LED al valor opuesto (encendido/apagado).

led\_pin.value(not led\_pin.value())

# Definición de la función 'counter' que también actúa como callback para el temporizador.

# Incrementa la variable global 'count' y la imprime.

def counter(timer):

global count

# Incrementa 'count' y muestra su valor en la consola.

print(count)

count += 1

# Creación de una instancia de la clase 'Timer'.

# 'timer' es un objeto de la clase 'Timer' que se utiliza para crear temporizadores.

timer = Timer(-1)

# Configuración del temporizador para llamar a la función 'blink' cada 1000 milisegundos (1 segundo).

timer.init(period=1000, mode=Timer.PERIODIC, callback=blink)

# Configuración del temporizador para llamar a la función 'counter' cada 1000 milisegundos (1 segundo).

timer.init(period=1000, mode=Timer.PERIODIC, callback=counter)

# Comentarios sobre el modo de suspensión profunda no implementado en el código proporcionado.

**Diagrama

Descripción generada automáticamente**

**Imagen que contiene Diagrama

Descripción generada automáticamente**

* **rtc2**

# Importa la clase Pin desde el módulo machine para control de GPIO.

from machine import Pin

import machine

# Inicializa una variable global para contar las pulsaciones del botón.

count = 0

# Configura el pin del botón como entrada con resistencia de pull-up.

# 'button\_pin' es una instancia de la clase 'Pin', configurada para leer entradas.

button\_pin = Pin(14, Pin.IN, Pin.PULL\_UP)

# Define la función que se ejecutará cuando se detecte una pulsación del botón.

# Esta función modifica la variable global 'count' y muestra su valor.

def button\_pressed(pin):

global count

count += 1 # Incrementa 'count' cada vez que se presiona el botón.

print(count) # Muestra el valor actual de 'count'.

# Configura una interrupción en el pin del botón para llamar a 'button\_pressed' cuando el botón se presione.

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

# Ciclo infinito para mantener el programa en ejecución.

# Mantiene el programa activo para que pueda responder a interrupciones.

while True:

pass # Este lugar es adecuado para añadir más lógica si es necesario.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* **rtcoled1**

# Importa las clases Timer, I2C y Pin del módulo machine.

from machine import Timer, I2C, Pin

# Importa la biblioteca sh1106 para interactuar con la pantalla OLED.

import sh1106

# Importa la biblioteca time para manejar funciones relacionadas con el tiempo.

import time

# Inicializa una variable global para llevar un conteo.

count = 0

# Configura el bus I2C para la pantalla OLED.

# 'i2c' es una instancia de la clase 'I2C' configurada con pines específicos y frecuencia.

i2c = I2C(0, scl=Pin(1), sda=Pin(0), freq=400000)

# Crea una instancia de la pantalla OLED utilizando el bus I2C.

# 'oled' es una instancia de la clase 'SH1106\_I2C' que representa la pantalla OLED.

oled = sh1106.SH1106\_I2C(128, 64, i2c)

# Define una función para actualizar el contenido mostrado en la pantalla OLED.

def update\_display():

oled.fill(0) # Limpia la pantalla.

oled.text('Conteo:', 0, 0) # Muestra el texto "Conteo:".

oled.text(str(count), 0, 10) # Muestra el valor actual del conteo.

oled.show() # Envía el buffer gráfico a la pantalla para actualizarla.

# Define una función que actúa como callback para un temporizador.

# Incrementa el contador y actualiza la pantalla.

def counter(timer):

global count

count += 1 # Incrementa el contador.

print(count) # Imprime el valor actual del contador en la consola.

update\_display() # Llama a la función para actualizar la pantalla.

# Crea y configura un temporizador.

# 'timer' es un objeto de la clase 'Timer' que se utiliza para ejecutar periódicamente el callback 'counter'.

timer = Timer(-1)

timer.init(period=1000, mode=Timer.PERIODIC, callback=counter)

# El programa mantiene el temporizador activo para actualizar continuamente la pantalla OLED.

**Interfaz de usuario gráfica, Texto

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

* **rtcoled2**

# Importa las clases necesarias para el control de pines e I2C.

from machine import Pin, I2C

# Importa la biblioteca sh1106 para la gestión de la pantalla OLED.

import sh1106

# Importa la biblioteca time para pausas y temporización.

import time

# Inicializa un contador global para rastrear la cantidad de presiones de botón.

count = 0

# Crea una instancia de la clase 'I2C' para comunicarse con la pantalla OLED.

i2c = I2C(0, scl=Pin(22), sda=Pin(21)) # Configuración de los pines SCL y SDA.

# Crea una instancia de la pantalla OLED a través de la clase 'SH1106\_I2C'.

oled = sh1106.SH1106\_I2C(128, 64, i2c) # Dimensiones de la pantalla OLED.

# Configura un pin para el botón como una entrada con resistencia de pull-up.

button\_pin = Pin(14, Pin.IN, Pin.PULL\_UP)

# Define una función que se llama cuando el botón es presionado.

def button\_pressed(pin):

global count

count += 1 # Incrementa el contador global.

print(count) # Imprime el contador en la consola.

update\_display() # Actualiza la pantalla OLED con el nuevo contador.

# Define una función para actualizar el contenido mostrado en la pantalla OLED.

def update\_display():

oled.fill(0) # Borra el contenido de la pantalla.

oled.text('Count:', 0, 0) # Muestra el texto "Count".

oled.text(str(count), 0, 20) # Muestra el contador.

oled.show() # Envía el contenido al buffer de la pantalla para su visualización.

# Configura una interrupción para detectar pulsaciones del botón.

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

# Actualiza la pantalla OLED con el estado inicial del contador.

update\_display()

# Ciclo infinito para mantener el programa en ejecución y permitir actualizaciones continuas.

while True:

time.sleep(1) # Pausa de un segundo para reducir el uso del CPU.

Texto

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* **rtcoled11**

# Importación de clases necesarias para manejar el temporizador, I2C, y pines GPIO.

from machine import Timer, I2C, Pin

# Importación de la biblioteca para controlar la pantalla SSD1306.

import ssd1306

# Importación de time para controlar el tiempo de espera en el bucle.

import time

# Inicialización de una variable global para llevar el conteo.

count = 0

# Creación de una instancia de I2C para la comunicación con la pantalla OLED.

i2c = I2C(0, scl=Pin(1), sda=Pin(0), freq=400000) # Configuración de pines y frecuencia.

# Instanciación de la pantalla OLED utilizando la clase SSD1306\_I2C.

oled = ssd1306.SSD1306\_I2C(128, 64, i2c)

# Definición de una función para actualizar la pantalla OLED.

def update\_display():

oled.fill(0) # Borra el contenido de la pantalla.

oled.text('Conteo:', 0, 0) # Muestra texto estático.

oled.text(str(count), 0, 10) # Muestra el valor de 'count'.

oled.show() # Envía los datos al buffer de la pantalla para visualización.

# Definición de la función callback para el temporizador que incrementa y muestra el conteo.

def counter(timer):

global count

count += 1 # Incrementa el contador.

print(count) # Imprime el valor actualizado del contador.

update\_display() # Llama a la función de actualización de la pantalla.

# Creación y configuración de un objeto temporizador.

timer = Timer(-1)

timer.init(period=1000, mode=Timer.PERIODIC, callback=counter) # Configura para ejecutar 'counter' cada segundo.

# Actualización inicial de la pantalla para mostrar el estado inicial.

update\_display()

# Bucle infinito para mantener el programa funcionando y permitir actualizaciones continuas.

while True:

time.sleep(1) # Reducción del uso de CPU.

Diagrama

Descripción generada automáticamente con confianza media

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* **rtcoled22**

# Importa las clases necesarias para la gestión de pines e I2C.

from machine import Pin, I2C

# Importa la biblioteca SSD1306 para controlar la pantalla OLED.

import ssd1306

# Importa la biblioteca time para funciones relacionadas con el tiempo.

import time

# Inicializa un contador global para rastrear la cantidad de veces que se presiona el botón.

count = 0

# Crea una instancia de la clase 'I2C' para la comunicación con la pantalla OLED.

i2c = I2C(0, scl=Pin(22), sda=Pin(21)) # Configuración de los pines SCL y SDA.

# Crea una instancia de la pantalla OLED utilizando la clase 'SSD1306\_I2C'.

oled = ssd1306.SSD1306\_I2C(128, 64, i2c) # Especificaciones de la pantalla OLED.

# Configura un pin para el botón como una entrada con resistencia de pull-up.

button\_pin = Pin(14, Pin.IN, Pin.PULL\_UP)

# Define una función que se ejecuta cuando el botón es presionado.

def button\_pressed(pin):

global count

count += 1 # Incrementa el contador global.

print(count) # Imprime el contador actualizado en la consola.

update\_display() # Actualiza la pantalla OLED con el nuevo contador.

# Define una función para actualizar el contenido mostrado en la pantalla OLED.

def update\_display():

oled.fill(0) # Borra el contenido de la pantalla.

oled.text('Count:', 0, 0) # Muestra texto estático.

oled.text(str(count), 0, 20) # Muestra el valor actual de 'count'.

oled.show() # Envía los datos al buffer de la pantalla para su visualización.

# Configura una interrupción para detectar pulsaciones del botón.

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

# Realiza una actualización inicial de la pantalla para mostrar el estado inicial del contador.

update\_display()

# Bucle infinito para mantener el programa funcionando y permitir actualizaciones continuas.

while True:

time.sleep(1) # Pausa para reducir el uso de CPU.

**Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente con confianza media**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**