**APServer**

* **ESP32**

##

# Importa la clase `WLAN` de la biblioteca `network`.

# `WLAN` permite gestionar las conexiones de red en la ESP32.

##

import network

##

# Importa la biblioteca `time` para funciones de temporización y retrasos.

##

import time

##

# Importa la clase `socket` de la biblioteca `socket`.

# `socket` permite crear y gestionar conexiones de red mediante sockets.

##

import socket

##

# Importa la clase `Pin` de la biblioteca `machine`.

# `Pin` permite controlar los pines de entrada/salida en el microcontrolador.

##

from machine import Pin

##

# Crea una instancia de la clase `Pin` para controlar el LED en el pin 2 de la ESP32.

# `led` es un objeto que representa el pin 2 configurado como salida, permitiendo encender y apagar el LED.

##

led = Pin(2, Pin.OUT)

##

# Define la función `web\_page` para generar la página web HTML que controla el LED.

# Esta función devuelve una cadena de texto con el HTML, que contiene los botones de control para el LED.

##

def web\_page():

html = """

<html>

<head>

<title>LED SWITCH</title>

<style>

body { font-family: Arial, sans-serif; background-color: #f4f4f4; text-align: center; padding-top: 50px; }

h1 { color: #333; font-size: 36px; margin-bottom: 20px; }

p { margin: 20px; }

button { padding: 15px 30px; font-size: 18px; border: none; border-radius: 5px; cursor: pointer; transition: background-color 0.3s; }

button:hover { opacity: 0.9; }

.on-button { background-color: #28a745; color: white; }

.off-button { background-color: #dc3545; color: white; }

.container { display: inline-block; padding: 20px; border-radius: 10px; background-color: white; box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.1); }

</style>

</head>

<body>

<div class="container">

<h1>LED SWITCH</h1>

<p><a href="/on"><button class="on-button">Encender</button></a></p>

<p><a href="/off"><button class="off-button">Apagar</button></a></p>

</div>

</body>

</html>

"""

return html

##

# Define la función `ap\_mode`, que configura la ESP32 en modo Access Point (AP) para controlar el LED mediante solicitudes HTTP.

# Recibe dos parámetros: `ssid` (nombre de la red) y `password` (contraseña).

##

def ap\_mode(ssid, password):

##

# Crea una instancia de la clase `WLAN` en modo Access Point.

# `ap` representa la conexión en modo AP.

##

ap = network.WLAN(network.AP\_IF)

##

# Configura el AP con el nombre de red (SSID) y la contraseña.

##

ap.config(essid=ssid, password=password)

##

# Activa el modo AP.

##

ap.active(True)

##

# Espera hasta que el AP esté activo.

##

while not ap.active():

pass

print('Modo AP está activo, puedes conectarte ahora.')

print('Dirección IP: ' + ap.ifconfig()[0])

##

# Crea una instancia de `socket` para manejar conexiones de red.

# `s` es un socket TCP configurado para escuchar en el puerto 80.

##

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

##

# Configura el socket para escuchar en el puerto 80.

##

s.bind(('', 80))

##

# Pone el socket en modo de escucha, permitiendo hasta 5 conexiones en espera.

##

s.listen(5)

##

# Bucle infinito que acepta conexiones y procesa las solicitudes HTTP para controlar el LED.

##

while True:

##

# Espera una conexión y crea una instancia `conn` para manejar la conexión con el cliente.

##

conn, addr = s.accept()

print('Conexión por parte de (IP): %s' % str(addr))

##

# Lee la solicitud HTTP recibida y decodifica el contenido.

##

request = conn.recv(1024)

request\_str = request.decode('utf-8')

print('Contenido = %s' % str(request))

##

# Verifica si la solicitud contiene "/on" para encender el LED.

##

if '/on' in request\_str:

print('Encendiendo LED')

led.value(1) # Enciende el LED llamando al método `value(1)` en la instancia `led`.

##

# Verifica si la solicitud contiene "/off" para apagar el LED.

##

elif '/off' in request\_str:

print('Apagando LED')

led.value(0) # Apaga el LED llamando al método `value(0)` en la instancia `led`.

##

# Genera la respuesta HTTP con el HTML de `web\_page` y la envía al cliente.

##

response = "HTTP/1.1 200 OK\nContent-Type: text/html\nConnection: close\n\n" + web\_page()

conn.sendall(response)

##

# Cierra la conexión con el cliente.

##

conn.close()

##

# Llama a la función `ap\_mode` para activar el modo AP en la ESP32 con el SSID y la contraseña especificados.

##

ap\_mode('ESP32\_AP', '12345678')

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

* **Raspberry pi pico W**

##

# Importa la clase `WLAN` de la biblioteca `network`.

# Esta clase permite gestionar las conexiones de red, y en este caso se usará para configurar la ESP32 en modo Access Point.

##

import network

##

# Importa la biblioteca `time`, que proporciona funciones de temporización y retrasos.

##

import time

##

# Importa la clase `socket`, que permite crear y gestionar conexiones de red mediante sockets.

##

import socket

##

# Importa la clase `Pin` de la biblioteca `machine`.

# `Pin` permite controlar los pines de entrada/salida en el microcontrolador.

##

from machine import Pin

##

# Crea una instancia de la clase `Pin` para controlar el LED en el pin 2 de la ESP32.

# `led` es un objeto que representa el pin 2 configurado como salida, permitiendo encender y apagar el LED.

##

led = Pin(2, Pin.OUT)

##

# Define la función `web\_page`, que genera el contenido de la página web en HTML para controlar el LED.

# Esta función devuelve una cadena de texto con el HTML, que contiene botones de control para encender y apagar el LED.

##

def web\_page():

html = """

<html>

<head>

<title>LED SWITCH</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

text-align: center;

padding-top: 50px;

}

h1 {

color: #333;

font-size: 36px;

margin-bottom: 20px;

}

p {

margin: 20px;

}

button {

padding: 15px 30px;

font-size: 18px;

border: none;

border-radius: 5px;

cursor: pointer;

transition: background-color 0.3s;

}

button:hover {

opacity: 0.9;

}

.on-button {

background-color: #28a745;

color: white;

}

.off-button {

background-color: #dc3545;

color: white;

}

.container {

display: inline-block;

padding: 20px;

border-radius: 10px;

background-color: white;

box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.1);

}

</style>

</head>

<body>

<div class="container">

<h1>LED SWITCH</h1>

<p><a href="/on"><button class="on-button">Encender</button></a></p>

<p><a href="/off"><button class="off-button">Apagar</button></a></p>

</div>

</body>

</html>

"""

return html

##

# Define la función `ap\_mode`, que configura la ESP32 en modo Access Point (AP) para controlar el LED mediante solicitudes HTTP.

# Recibe dos parámetros: `ssid` (nombre de la red) y `password` (contraseña).

##

def ap\_mode(ssid, password):

##

# Crea una instancia de la clase `WLAN` en modo Access Point.

# `ap` representa la conexión en modo AP.

##

ap = network.WLAN(network.AP\_IF)

##

# Configura el AP con el nombre de red (SSID) y la contraseña.

##

ap.config(essid=ssid, password=password)

##

# Activa el modo AP.

##

ap.active(True)

##

# Espera hasta que el AP esté activo.

##

while not ap.active():

pass

print('Modo AP está activo, puedes conectarte ahora.')

print('Dirección IP: ' + ap.ifconfig()[0])

##

# Crea una instancia de `socket` para manejar conexiones de red.

# `s` es un socket TCP configurado para escuchar en el puerto 80.

##

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

##

# Configura el socket para escuchar en el puerto 80.

##

s.bind(('', 80))

##

# Pone el socket en modo de escucha, permitiendo hasta 5 conexiones en espera.

##

s.listen(5)

##

# Bucle infinito que acepta conexiones y procesa las solicitudes HTTP para controlar el LED.

##

while True:

##

# Espera una conexión y crea una instancia `conn` para manejar la conexión con el cliente.

##

conn, addr = s.accept()

print('Conexión por parte de (IP): %s' % str(addr))

##

# Lee la solicitud HTTP recibida y decodifica el contenido.

##

request = conn.recv(1024)

request\_str = request.decode('utf-8')

print('Contenido = %s' % str(request))

##

# Verifica si la solicitud contiene "/on" para encender el LED.

##

if '/on' in request\_str:

print('Encendiendo LED')

led.value(1) # Enciende el LED llamando al método `value(1)` en la instancia `led`.

##

# Verifica si la solicitud contiene "/off" para apagar el LED.

##

elif '/off' in request\_str:

print('Apagando LED')

led.value(0) # Apaga el LED llamando al método `value(0)` en la instancia `led`.

##

# Genera la respuesta HTTP con el HTML de `web\_page` y la envía al cliente.

##

response = "HTTP/1.1 200 OK\nContent-Type: text/html\nConnection: close\n\n" + web\_page()

conn.sendall(response)

##

# Cierra la conexión con el cliente.

##

conn.close()

##

# Llama a la función `ap\_mode` para activar el modo AP en la ESP32 con el SSID y la contraseña especificados.

##

ap\_mode('PICO\_W\_AP', '12345678')

Tabla

Descripción generada automáticamente

Tabla

Descripción generada automáticamente