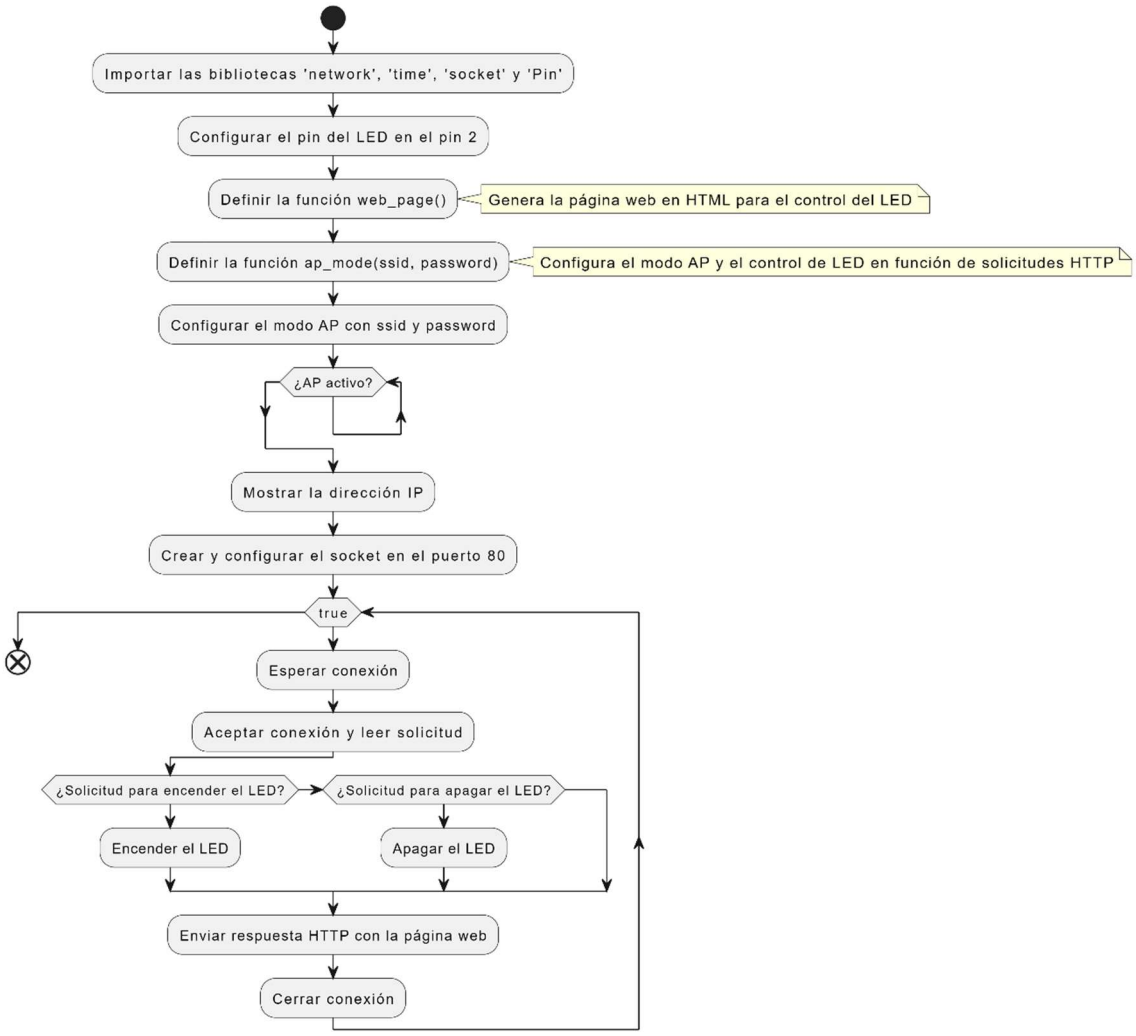
APServer

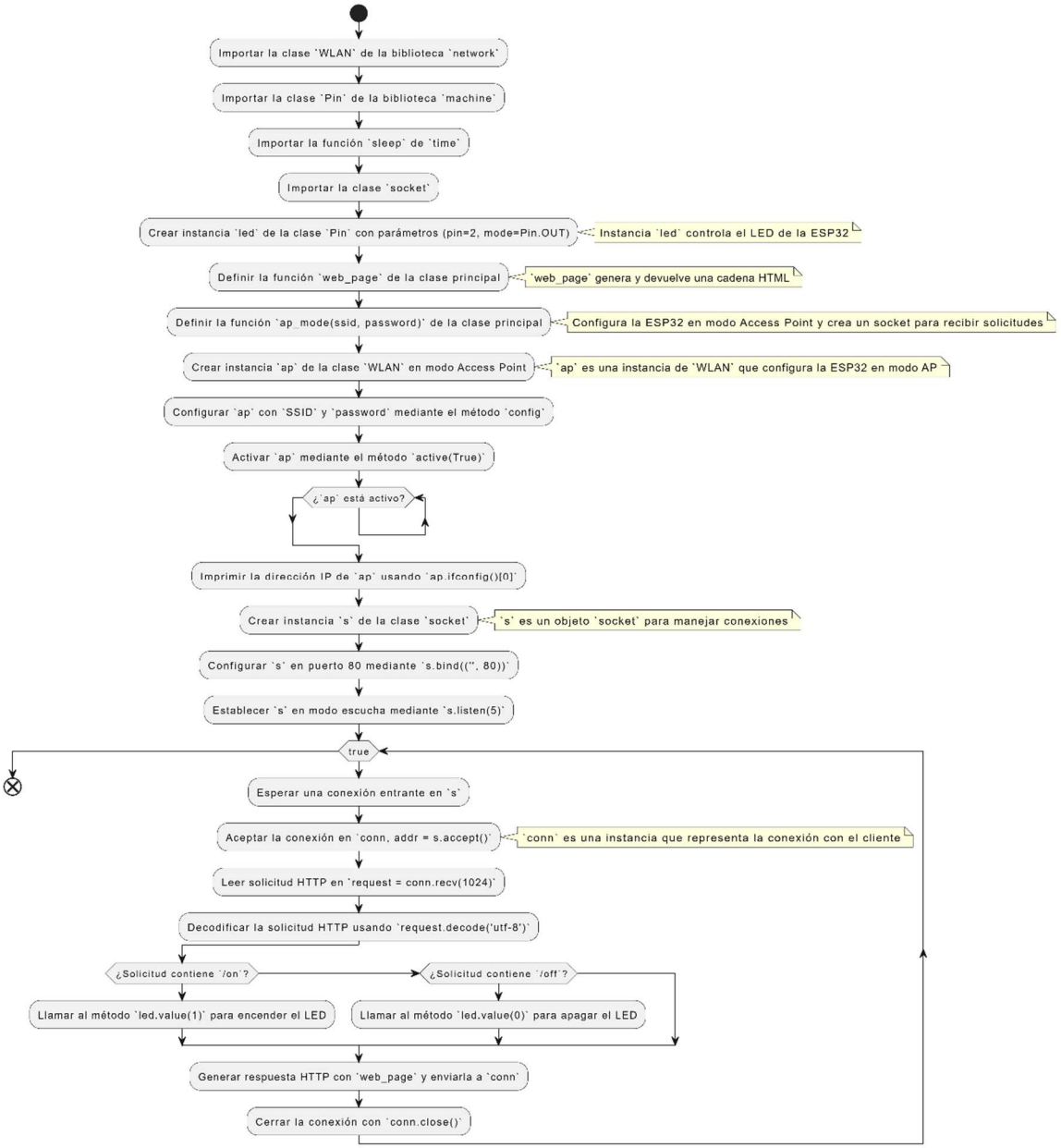
- **ESP32**

```
##
# 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')
```



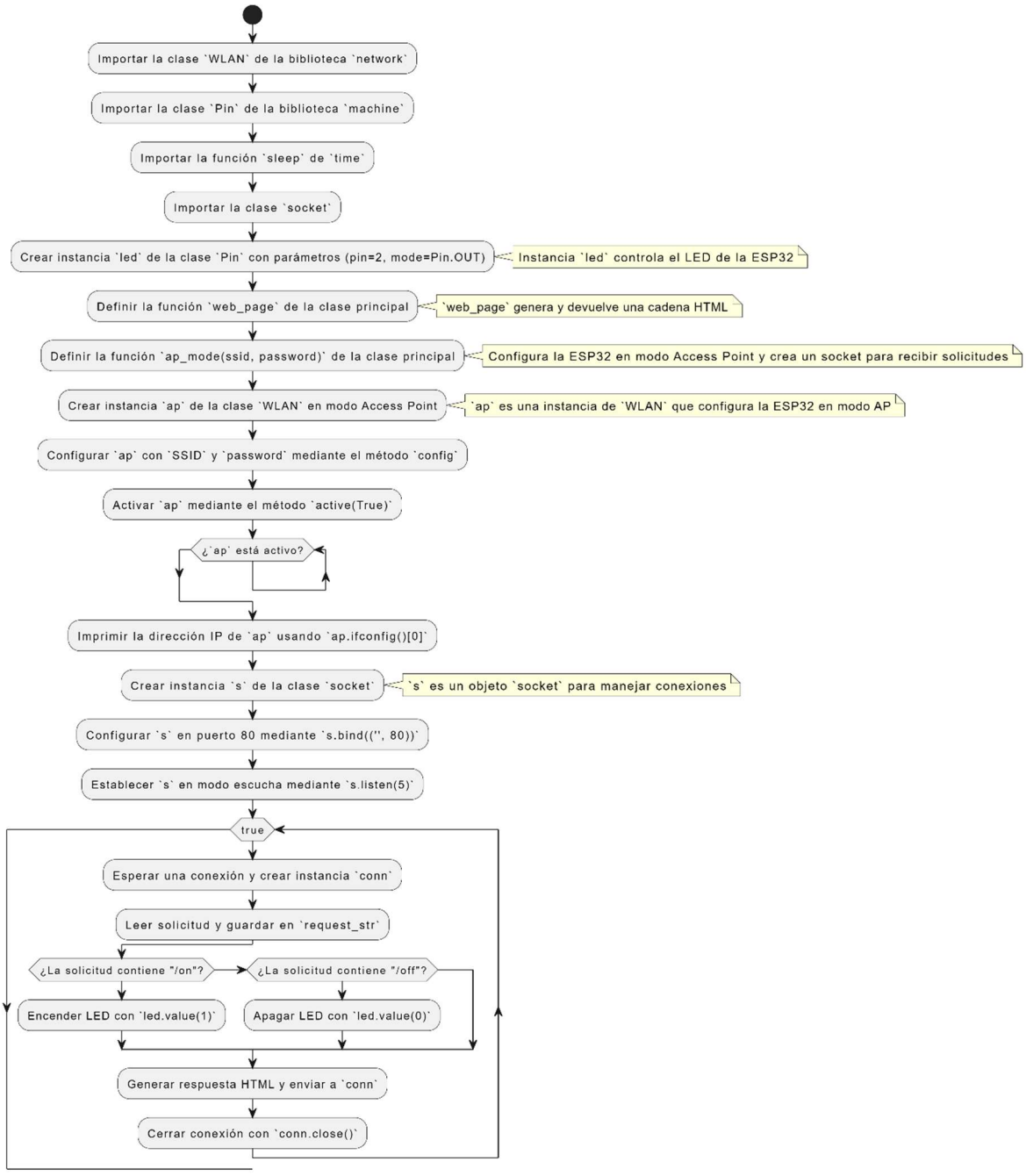

Importar la clase `WLAN` de la biblioteca `network`
Importar la clase `Pin` de la biblioteca `machine`
Importar la función `sleep` de `time`
Importar la clase `socket`
Crear instancia `led` de la clase `Pin` con parámetros (pin=2, mode=Pin.OUT)
Instancia `led` controla el LED de la ESP32
Definir la función `web_page` de la clase principal
`web_page` genera y devuelve una cadena HTML
Definir la función `ap_mode(ssid, password)` de la clase principal
Configura la ESP32 en modo Access Point y crea un socket para recibir solicitudes
Crear instancia `ap` de la clase `WLAN` en modo Access Point
`ap` es una instancia de `WLAN` que configura la ESP32 en modo AP
Configurar `ap` con `SSID` y `password` mediante el método `config`
Activar `ap` mediante el método `active(True)`
¿`ap` está activo?
Imprimir la dirección IP de `ap` usando `ap.ifconfig()[0]`
Crear instancia `s` de la clase `socket`
`s` es un objeto `socket` para manejar conexiones
Configurar `s` en puerto 80 mediante `s.bind(('', 80))`
Establecer `s` en modo escucha mediante `s.listen(5)`
true
Esperar una conexión entrante en `s`
Aceptar la conexión en `conn, addr = s.accept()`
`conn` es una instancia que representa la conexión con el cliente
Leer solicitud HTTP en `request = conn.recv(1024)`
Decodificar la solicitud HTTP usando `request.decode('utf-8')`
¿Solicitud contiene `/on`?
¿Solicitud contiene `/off`?
Llamar al método `led.value(1)` para encender el LED
Llamar al método `led.value(0)` para apagar el LED
Generar respuesta HTTP con `web_page` y enviarla a `conn`
Cerrar la conexión con `conn.close()`

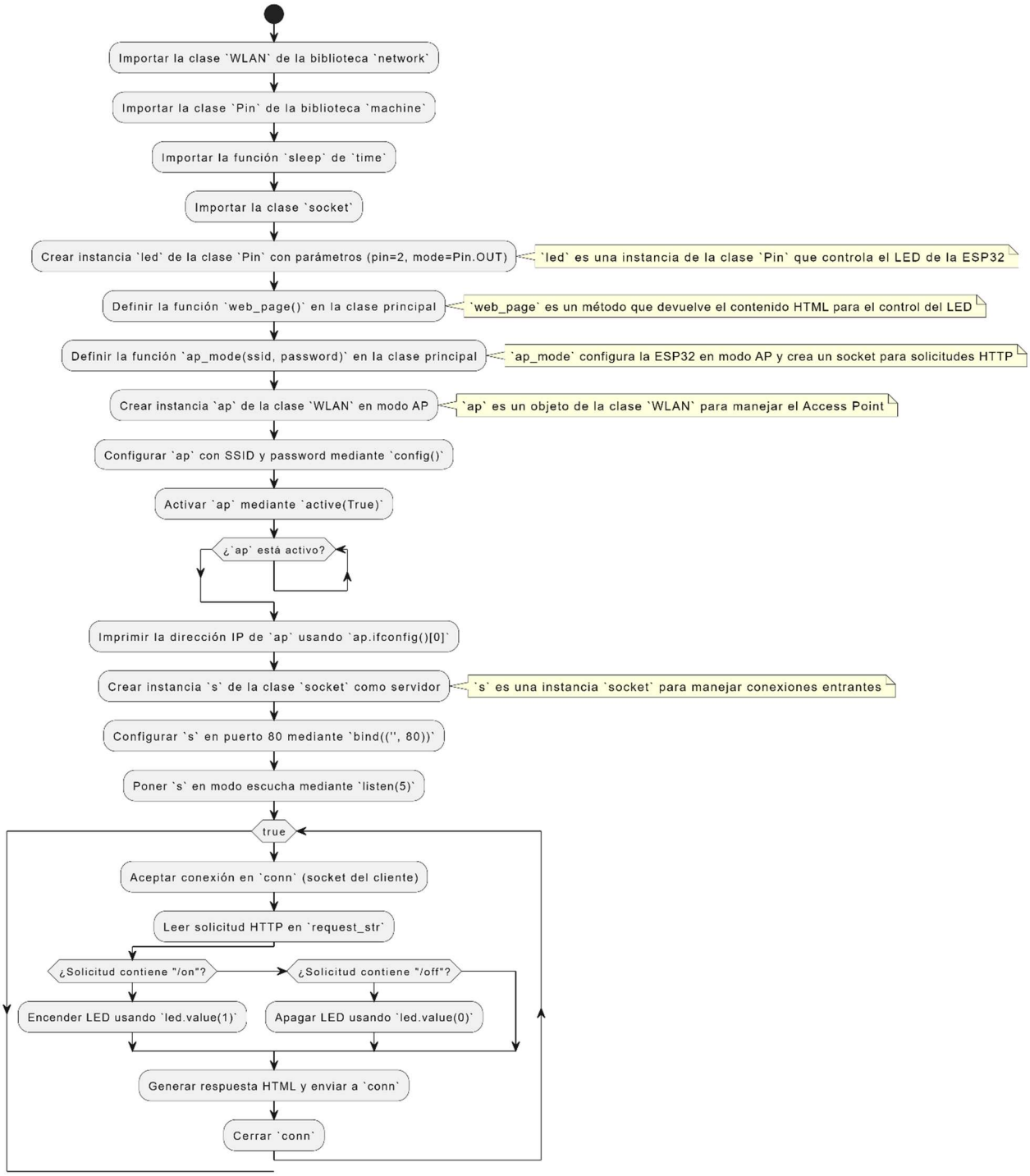
- **Raspberry pi pico W**

```
##
# 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')
```


Importar la clase `WLAN` de la biblioteca `network`
Importar la clase `Pin` de la biblioteca `machine`
Importar la función `sleep` de `time`
Importar la clase `socket`
Crear instancia `led` de la clase `Pin` con parámetros (pin=2, mode=Pin.OUT)
Instancia `led` controla el LED de la ESP32
Definir la función `web_page` de la clase principal
`web_page` genera y devuelve una cadena HTML
Definir la función `ap_mode(ssid, password)` de la clase principal
Configura la ESP32 en modo Access Point y crea un socket para recibir solicitudes
Crear instancia `ap` de la clase `WLAN` en modo Access Point
`ap` es una instancia de `WLAN` que configura la ESP32 en modo AP
Configurar `ap` con `SSID` y `password` mediante el método `config`
Activar `ap` mediante el método `active(True)`
¿`ap` está activo?
Imprimir la dirección IP de `ap` usando `ap.ifconfig()[0]`
Crear instancia `s` de la clase `socket`
`s` es un objeto `socket` para manejar conexiones
Configurar `s` en puerto 80 mediante `s.bind(('', 80))`
Establecer `s` en modo escucha mediante `s.listen(5)`
true
Esperar una conexión y crear instancia `conn`
Leer solicitud y guardar en `request_str`
¿La solicitud contiene "/on"?
¿La solicitud contiene "/off"?
Encender LED con `led.value(1)`
Apagar LED con `led.value(0)`
Generar respuesta HTML y enviar a `conn`
Cerrar conexión con `conn.close()`


Importar la clase `WLAN` de la biblioteca `network`
Importar la clase `Pin` de la biblioteca `machine`
Importar la función `sleep` de `time`
Importar la clase `socket`
Crear instancia `led` de la clase `Pin` con parámetros (pin=2, mode=Pin.OUT)
`led` es una instancia de la clase `Pin` que controla el LED de la ESP32
Definir la función `web_page()` en la clase principal
`web_page` es un método que devuelve el contenido HTML para el control del LED
Definir la función `ap_mode(ssid, password)` en la clase principal
`ap_mode` configura la ESP32 en modo AP y crea un socket para solicitudes HTTP
Crear instancia `ap` de la clase `WLAN` en modo AP
`ap` es un objeto de la clase `WLAN` para manejar el Access Point
Configurar `ap` con SSID y password mediante `config()`
Activar `ap` mediante `active(True)`
¿`ap` está activo?
Imprimir la dirección IP de `ap` usando `ap.ifconfig()[0]`
Crear instancia `s` de la clase `socket` como servidor
`s` es una instancia `socket` para manejar conexiones entrantes
Configurar `s` en puerto 80 mediante `bind(('', 80))`
Poner `s` en modo escucha mediante `listen(5)`
true
Aceptar conexión en `conn` (socket del cliente)
Leer solicitud HTTP en `request_str`
¿Solicitud contiene "/on"?
¿Solicitud contiene "/off"?
Encender LED usando `led.value(1)`
Apagar LED usando `led.value(0)`
Generar respuesta HTML y enviar a `conn`
Cerrar `conn`