MicroWebSrv

Zu den Modulen in der Firmware von M5Stack gehören auch die für den **MicroWebSrv**. Damit kann in Micropython ein Webserver erzeugt werden.

https://github.com/jczic/MicroWebSrv

https://github.com/loboris/MicroPython_ESP32_psRAM_LoBo/wiki/microWebSrv

https://github.com/loboris/MicroPython_ESP32_psRAM_LoBo/blob/master/MicroPython_BUILD/components/micropython/esp32/modules_examples/webserver/webserver_example.py

https://www.youtube.com/watch?v=Y7jhgPtclFE

MicroWebSrv

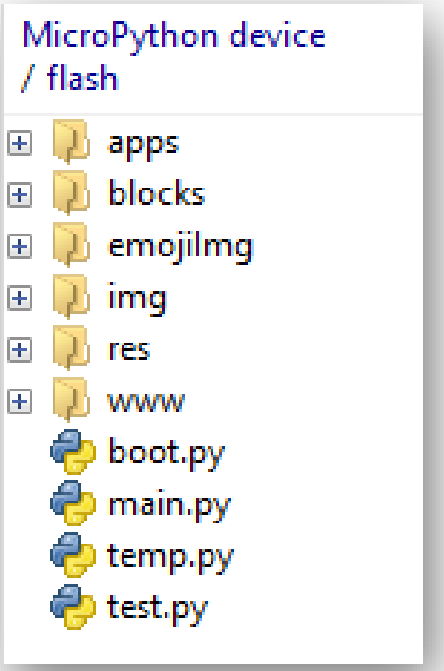
Ins Wlan gehen:

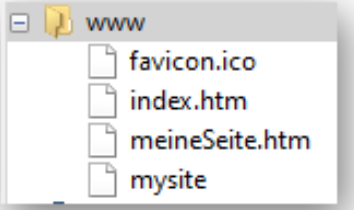
Damit ein Webserver seine Informationen an den Mann bringen kann, muss er sich in einem Netzwerk befinden. Beim ESP32 bietet sich dazu das Wlan an:

```
# websrv_01.py

>>> import network
    from wlansecrets import SSID, PW

>>> wlan = network.WLAN(network.STA_IF)
>>> wlan.active(True)
True
>>> wlan.connect(SSID, PW)
>>> while wlan.isconnected() != True:
        time.sleep(1)
    else:
        print(wlan.ifconfig()[0])
192.168.5.105
>>>
```

MicroWebSrv

Was der Webserver so braucht:
Der Webserver benötigt, wie andere Webserver auch ein Verzeichnis in dem er die Dateien findet, die er ausgeben soll. Dieses Verzeichnis heißt per default **www**. Man kann auch einen anderen Namen wählen, muss es dem Webserver aber mitteilen.





MicroWebSrv

Wie funktioniert HTTP? (stark vereinfacht)

HTTP (Hypertext Transfer Protocol) wird zur Übertragung von Webseiten verwendet. Es ist ein verbindungsorientiertes, zustandsloses Protokoll. Dh. Es wird zuerst eine **Verbindung aufgebaut** bevor Daten übertragen werden. (Wie Telefon)

Dann schickt der **Client (Browser)** eine **Anfrage (Request)** zum Server. Dieser schickt eine **Antwort (Respont)** zurück.
Dann wird die Verbindung beendet.

HTTP-Verbindungen können nur vom Client initiiert werden.

Es gibt Neuerungen, die von MicroWebSrv aber nicht unterstützt werden.

MicroWebSrv

Die Struktur des Modules MicroWebSrv ist etwas verworren:

```
>>> import MicroWebSrv
>>> dir(MicroWebSrv)
['__class__', '__name__', '__file__', '__path__', 'microWebSrv']
>>> dir(MicroWebSrv.microWebSrv)
['__class__', '__name__', '__file__', 'dumps', 'gc', 'loads', 'socket',
'start_new_thread', 'stat', 'MicroWebSrv', 're', 'MicroWebSrvRoute']
>>> dir(MicroWebSrv.microWebSrv.MicroWebSrv)
['__class__', '__init__', '__module__', '__name__', '__qualname__',
'__bases__', '__dict__', 'Start', 'route', '_indexPages', '_mimeTypes',
'_html_escape_chars', '_pyhtmlPagesExt', '_docoratedRouteHandlers',
'HTMLEscape', '_startThread', '_unquote', '_unquote_plus', '_fileExists',
'_isPyHTMLFile', '_serverProcess', 'Stop', 'IsStarted', 'SetNotFoundPageUrl',
'GetMimeTypeFromFilename', 'GetRouteHandler', '_physPathFromURLPath',
'_client', '_response']
>>>
```

MicroWebSrv

Webserver starten

Nachdem die Verbindung zum **Wlan** hergestellt und das Verzeichnis **www** eingerichtet ist, kann der Webserver gestartet werden.

```
>>> import MicroWebSrv.microWebSrv as mws
    srv = mws.MicroWebSrv()
    srv.Start(threaded=True)
>>>

# websrv02.py
```

http://192.168.5.105/

MicroWebSrv

Webserver für statische Webseiten

Nun haben wir einen einfachen Webserver für statische Webseiten. D.h. wir können keine Inhalte zwischen den Webseiten und dem Programm austauschen.

Aber dafür gibt es beim **MicroWebSrv** eine Lösung:

Route Handler

MicroWebSrv

URL’s (Uniform Resource Locator)

Damit wird eine Adresse bezeichnet, die einen Pfad zu einer bestimmten Datei auf einem Server angibt

Z.B:
192.168.5.105/mysite

Dabei ist
192.168.5.105 die Geräteadresse und
/mysite der **relative Pfad** unterhalb von **/flash/www**.

MicroWebSrv

Route Handler

Route Handler sind **Pythonfunktionen** die bei bestimmten **relativen Pfad** aufgerufen werden. Sie können auf 2 Arten organisiert werden:

1. **Als Liste**
2. **Als Decorator**

MicroWebSrv

RouteHandler mit Liste anlegen

```
routeHandlers = [
   ( "relative_url_route_1", "METHOD", handlerFunc_1 ),
   ( "relative_url_route_2", "METHOD", handlerFunc_2 ),
   ( ... )
]

def handlerFunc_1(httpClient, httpResponse, routeArgs=None) :
  print("In HTTP handler 1")

def handlerFunc_2(httpClient, httpResponse, routeArgs=None) :
  print("In HTTP handler 2")
```

MicroWebSrv

RouteHandler mit Liste anlegen

Ein Beispiel:

```
route_handler_list = [
    ( "/ledon", "GET", handler_led_on ),
    ( "/ledoff", "GET", handler_led_off )
]
```

Dann muss dem **MicroWebSrv** diese Datei beim Start bekanntgemacht werden:

```
srv = MicroWebSrv(routeHandlers=None, port=80, bindIP='0.0.0.0',
webPath="/flash/www") # routeHandlers=route_handler_list
```

MicroWebSrv

Beispiel für einen Routehandler der eine HTML-Seite erzeugt:

```
def _httpHandlerTestGet(httpClient, httpResponse):
    global var1, var2
    content = """\
    <!DOCTYPE html>
    <html lang=de>
        <head>
        ...
        </head>
        <body>
        ...
        </body>
    </html>
    """ .format(var1, var2)
    httpResponse.WriteResponseOk( headers = None, contentType = "text/html",
                                  contentCharset = "UTF-8", content = content)
```

MicroWebSrv

Handler für /ledon:

1. LED einschalten
2. Response senden

```
def ledon(httpClient, httpResponse):

    M5Led.on()

    content = """ LED eingeschaltet
    """ .format()
    httpResponse.WriteResponseOk( headers = None,
                                  contentType = "text/html",
                                  contentCharset = "UTF-8",
                                  content = content)
```

MicroWebSrv

Handler für /ledoff:

1. LED ausschalten
2. Response senden

```
def ledon(httpClient, httpResponse):

    M5Led.off()

    content = """ LED ausgeschaltet
    """ .format()
    httpResponse.WriteResponseOk( headers = None,
                                  contentType = "text/html",
                                  contentCharset = "UTF-8",
                                  content = content)
```

MicroWebSrv

Die Route Handler Liste anlegen:

```
handler_liste = [
    ("/ledon", "GET", ledon),
    ("/ledoff", "GET", ledoff)
    ]
```

Dem Server muss nun der Name der Routehandlerliste bekannt gemacht werden:

```
srv = mws.MicroWebSrv(routeHandlers=handler_liste)
```

MicroWebSrv

Nun ein Beispiel mit der **Routehandler Liste** zum Ein- und Ausschalten der internen LED:

`websrv_03.py`

MicroWebSrv

Datenübermittlung zum Server mit /?

Die Methode

`httpClient.GetRequestQueryString()`

ermöglicht es Daten aus der URL-Zeile an den Server zu übergeben. So wie es z.B. bei Anfragen an Google gemacht wird:

`www.google.com/search?q=Peter+3%2C14++3.14+42`

Der Server bekommt dann den String:

`"q=Peter+3%2C14++3.14+42"`

MicroWebSrv

Um Daten vom Client entgegen nehmen zu können brauchen wir eine **globale Variable** die im Hauptprogramm erzeugt wird:

```
data_vom_client = None
```

Im Handler müssen wir sie bekanntmachen:

```
global data_vom_client
```

Anschließend können wir den **Querystring** hinein tun:

```
data_vom_client = httpClient.GetRequestQueryString()
```

Nun können wir außerhalb des Handlers darauf zugreifen:

```
>>> data_vom_client
'Hallo%20Teilnehmer'
```

MicroWebSrv

Dieses Konzept findet sich in

`websrv_04.py`

wieder.

Der Handler

`data_giving(httpClient, httpResponse)`

realisiert eine solche Abfrage.

Sie reagiert auf den relativen Pfad “**`/?`**“

MicroWebSrv

LED mit Querystring steuern

Nun können wir die LED auch mit einem Querystring steuern:
Wir benötigen nur noch einen Handler zum Ein- und Ausschalten der LED. Dieser Handler ist in

`websrv_05.py`

Implementiert.

Die SteuerURL’s sind:

`/LED?on`
`/LED?off`

MicroWebSrv

Werte in Webseiten ausgeben:

Wir bei dem LED- Beispiel schon gesehen, dass wir in der Variablen **content** Text eintragen können, der an den **Browser** geschickt wird. Dieser Text kann **HTML-Code** sein, so das der Browser daraus eine entsprechende Seite aufbauen kann. Da es sich um eine **Python String Variable** handelt, kann diese auch mit den **String Methoden** bearbeitet werden. Mit der **format** Methode können wir also in eine vorbereitete HTML Seite Werte formatiert einfügen.
Diese Variable muss mit dreifachen Anführungszeichen erstellt werden, weil sie sehr lang ist und Zeilenumbrüche enthält.

MicroWebSrv

Werte in eine Webseite einsetzen:

```
def bat_info(httpClient, httpResponse):
    content = """\
    Batteriespannung: {} Volt
    <br>
    Ladestrom       : {} Milliampere
    """.format(axp.getBatVoltage(), axp.getBatCurrent())
    httpResponse.WriteResponseOk( headers = None,
                                  contentType = "text/html",
                                  contentCharset = "UTF-8",
                                  content = content)
```

MicroWebSrv

Aufbau einer HTML-Datei:

```
<!DOCTYPE html>
  <html>
    <head>
      <meta charset="UTF-8" />
      <title>TEST POST</title>
    </head>
    <body>
      <h1>TEST POST</h1>
      Text zur Darstellung
    </body>
  </html>
```

MicroWebSrv

RouteHandler als Decorator anlegen

Während bei der Liste alles an einer Stelle liegt, können Decorators dort eingesetzt werden, wo der entsprechende Handler definiert wird.

```
@MicroWebSrv.route('/get-test')
def handlerFuncGet(httpClient, httpResponse) :
    print("In GET-TEST HTTP")

@MicroWebSrv.route('/post-test', 'POST')
def handlerFuncPost(httpClient, httpResponse) :
    print("In POST-TEST HTTP")
```

MicroWebSrv

Die Decorator-Methode im LED-Beispiel:

```
@mws.MicroWebSrv.route('/ledon')
def ledon(httpClient, httpResponse):
    M5Led.on()
    content = """ LED eingeschaltet
    """ .format()
    httpResponse.WriteResponseOk( headers = None,
                                  contentType = "text/html",
                                  contentCharset = "UTF-8",
                                  content = content)

@mws.MicroWebSrv.route('/ledoff')
def ledoff(httpClient, httpResponse):
    …
srv = mws.MicroWebSrv()
```

MicroWebSrv

ENDE des Kurses