15 Ein einfacher Website-Server

Um es gleich klar zu stellen: Hier geht es nicht um den Server für eine einzige Web*seite* ; dies haben wir im Prinzip in den letzten Kapiteln in verschiedenen Varianten schon behandelt. Vielmehr soll unser ESP32 als Server einer **Web*site*** mit mehreren untereinander verlinkten Webseiten (inkl. Bildern) dienen. Die zugehörigen HTML-Dokumente und Bilder sollen sich dabei in einem Ordner `web` auf dem ESP32 befinden.

Im Wesentlichen kann unser Programm auf die in den letzten Kapiteln behandelten Vorgehensweisen zurückgreifen. Der entscheidende Unterschied ist: Bislang hatten wir das "HTLM-Dokument", welches der Server an den Client senden sollte, in das Server-Programm integriert, z. B. in Form einer Funktion `html()`. Nun liegen die HTML-Dokumente in Form von einzelnen HTML-Dateien im Speicher des ESP32 vor.

Die Idee ist nun: Den Dateinamen der vom Client angeforderten Webseite bestimmen wir aus dem GET-Parameter des HTTP-Request. Damit öffnen wir über die **`open`**-Funktion diese Datei und weisen sie der Variablen `f` zu. Anschließend lesen wir mit der **`read`**-Methode die Datei Block für Block und senden diese über den Socket zum Client. Der entsprechende Programmteil lautet:

```
f = open(filename, 'rb') # HTML-Datei oder Bild-Datei zum Lesen öffnen
block_size = 512
while True:
  data = f.read(block_size)
  connection.send(data)
  if len(data) < block_size:
    break
  f.close()
```

Dabei ist es egal, ob es sich um HTML- oder Bild-Dateien handelt. Letztlich wird hier sowohl beim Lesen als auch beim Senden nur mit Bytes gearbeitet.

Zuvor senden wir allerdings erst noch den HTTP-Header. Dabei sollte auch der Datei-Typ (Content-Type) angegeben werden. Dieser wird der Funktion `http_header` als Parameter übergeben:

```
def http_header(extension):
  c_type = ''
  if extension == 'jpg':
    c_type = 'image/jpeg'
  if extension == 'htm':
    c_type = 'text/html'
  return '''HTTP/1.1 200 OK\r
Content-Type: ''' + c_type + '''\r
Connection: close\r
\r
'''
```

Welcher der beiden Header gesendet wird, muss das Programm an Hand der Extension der angeforderten Datei entscheiden.

Das gesamte Programm findet man unter dem Dateinamen `sta_webserver_2.py`. Bevor wir es starten, wollen wir aber noch unseren Ordner `web`, in welchem sich die HTML- und jpeg-Dateien unserer kleinen Test-Website befinden, auf den ESP32 hochladen. Wie dies geschieht, ist im Einführungs-Kapitel beschrieben.

Nun starten wir das Programm; im Terminal wird wie immer die IP-Adresse unseres Servers angezeigt; diese geben wir mitsamt der gewünschten Startseite `web/index.htm` ein. In meinem Fall ist die IP-Adresse `192.168.2.117`; hier lautet der Eintrag in die Adresszeile des Browsers also:

```
192.168.2.117/web/index.htm
```

Im Browser-Fenster wird angezeigt:

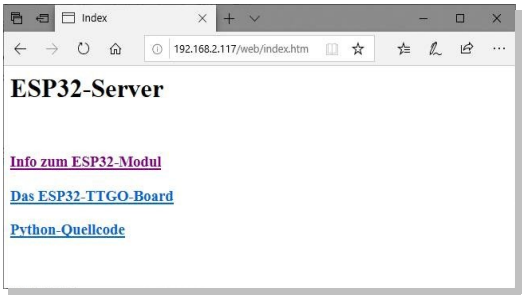

Abb. 1

Nun klicken wir auf den 2. Link; der Browser zeigt dann:

Abb. 2

Schauen wir auf das Terminal-Fenster der Thonny-IDE, stellen wir fest, dass dabei gleich zwei Requests beim Server eingegangen sind:

```
GET /web/ESP32_TTGO.htm HTTP/1.1
Referer: http://192.168.2.117/web/index.htm
...
Host: 192.168.2.117
Connection: Keep-Alive
```

und

```
GET /web/TTGO.jpg HTTP/1.1
Referer: http://192.168.2.117/web/ESP32_TTGO.htm
...
Host: 192.168.2.117
Connection: Keep-Alive
```

Dies können wir so erklären: Durch den ersten Request erhält der Client das HTML-Dokument ESP32_TTGO.htm. Um dieses Dokument anzuzeigen, muss der Browser sämtliche Inhalte des Dokuments auswerten; dabei stößt er unter anderem auf die folgende Zeile:

```
<img src="TTGO.jpg">
```

Diese Zeile bedeutet, dass das Bild mit dem Dateinamen `TTGO.jpg` im Browser-Fenster angezeigt werden soll. Dieses Bild muss sich nun der Browser beim Server besorgen. Dazu sendet er jetzt den zweiten Request. Erst, wenn er die zugehörige Antwort erhalten hat, kann er das Bild wiedergeben und damit die Anzeige der Webseite vervollständigen.

Es ist also nicht der Server, der dafür sorgt, dass alle Bestandteile der Webseite beim Client ankommen. Vielmehr ist hier der Client (Browser) in der Pflicht: Er muss kontrollieren, welche Dateien er noch zur vollständigen Darstellung der Webseite benötigt, und dementsprechend ggf. weitere Requests vornehmen.

Natürlich können wir noch einiges an diesem Webserver verbessern. Das merken Sie sofort, wenn Sie einen falschen Dateinamen eingeben. In diesem Fall bricht unser Programm mit einer Fehlermeldung ab (Es kann ja die Datei nicht öffnen!) und der Browser wartet vergeblich auf eine Antwort. Sie können selbst einmal versuchen, hier für Abhilfe zu sorgen. Dabei sollten Sie u. A. beachten:

- Vor dem Öffnen der angeforderten Datei überprüfen, ob die Datei überhaupt existiert
- Falls eine Datei nicht existiert, auch einen entsprechenden Header (Error 404) an den Client senden

Viel Erfolg dabei!