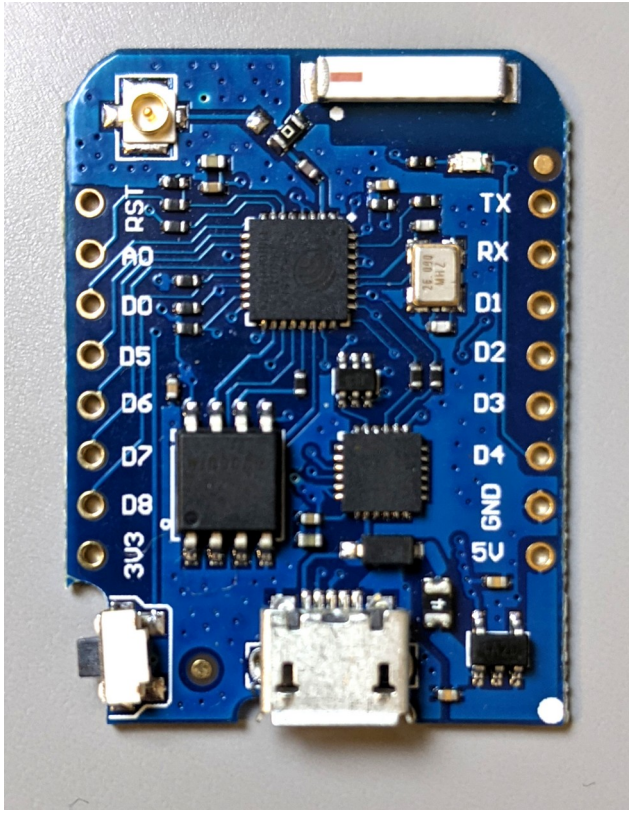
Sketching with Hardware

07: ESP8266 & Kommunikationsprotokolle

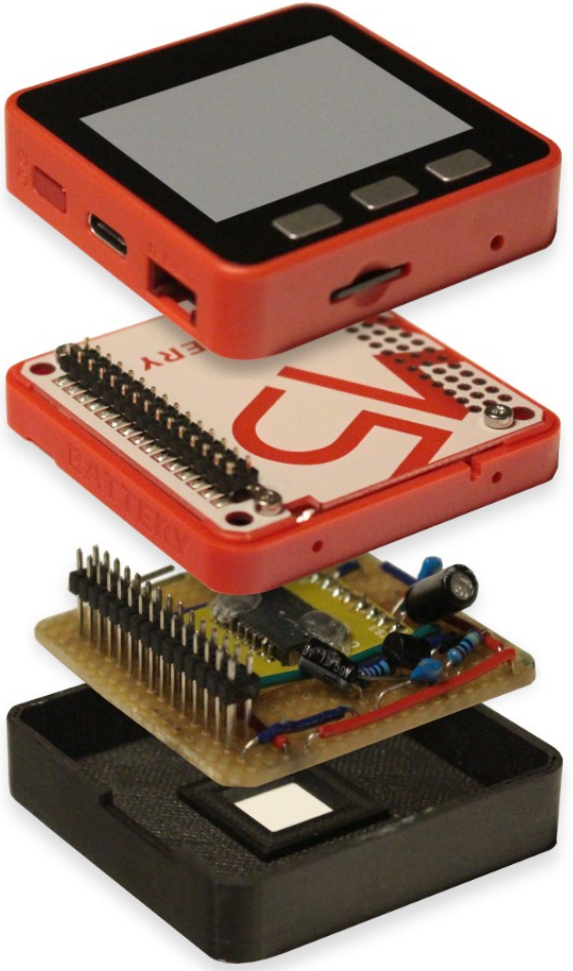
Wemos D1 mini (ESP8266)

- WiFi-fähiger Mikrocontroller
- Arduino-Kompatibel
- Arbeitet mit 3.3 Volt!
- Shield-System
- Sehr klein
- Extrem günstig (1 – 6€)



M5Stack

- ESP32 mit allerhand Zuberhör:
  - LCD, Buttons, Lautsprecher, WiFi, SD-Slot, 9-DoF-Sensor, …
- Erweiterbar über Shields und Grove-System
- Arduino-Kompatibel
- 3.3 Volt!
- Sehr teuer (60-80€)



Fragen?

Netzwerkprogrammierung

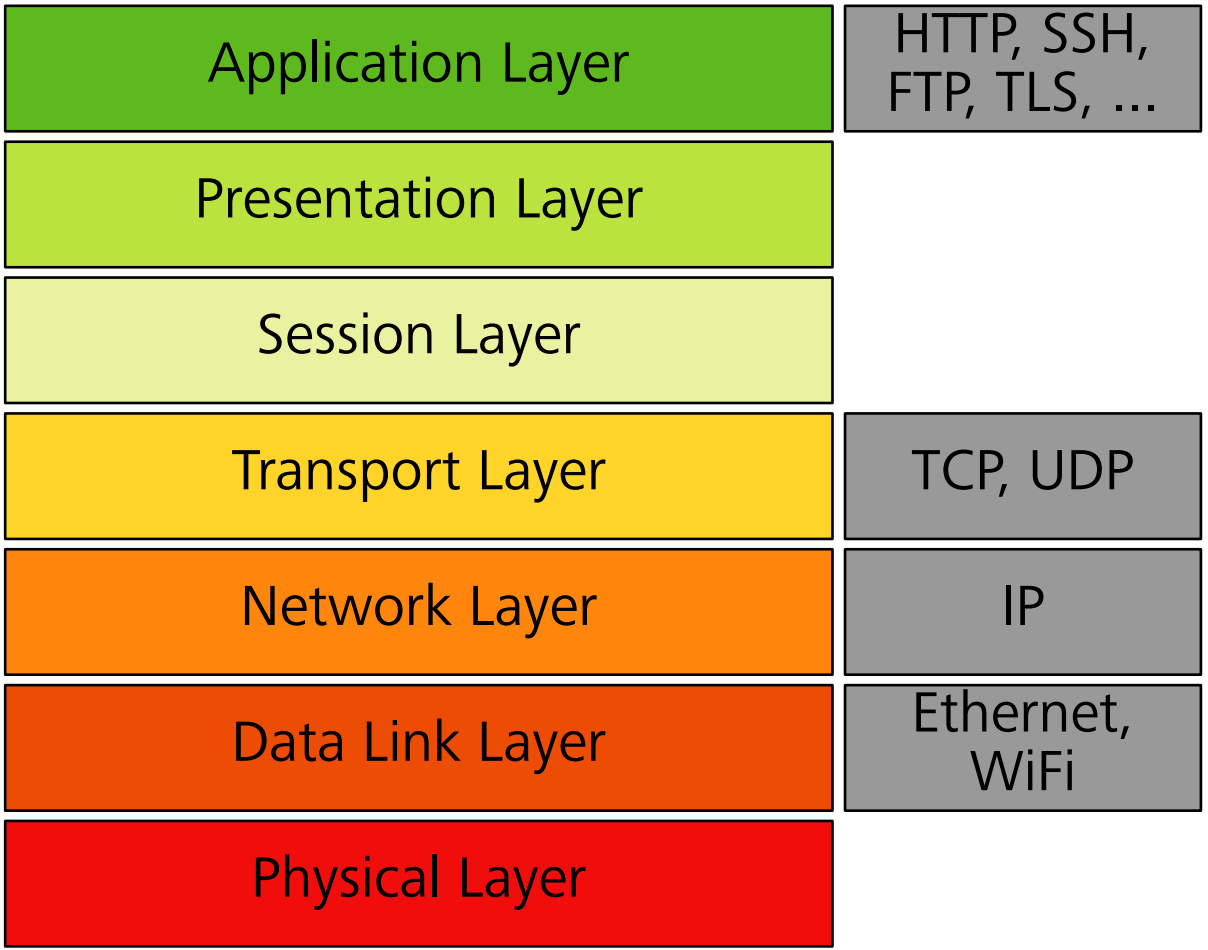
- Wir arbeiten mit Arduino-Bibliotheken am Mikrocontroller und Python am PC
- Wir sehen uns Sockets, HTTP und MQTT an
- Codesamples, auf die ihr Projekte aufbauen könnt

Disclaimer / Entwarnung

- Praktischer Ansatz: Wir wollen Drahtloskommunikation **benutzen**
  → wir müssen nicht jedes Detail verstehen
- Aber: Wenn man die Hintergründe kennt, fällt die praktische Arbeit leichter (z.B. Debugging)
- Drahtlose Verbindungen sind von Natur aus fehleranfällig

  → eine weitere potentielle Fehlerquelle

OSI Model & Protocol Stack



Nach: https://en.wikipedia.org/wiki/Protocol_stack, https://en.wikipedia.org/wiki/OSI_model

Transport Layer: TCP vs UDP

- TCP und UDP sind die wichtigsten Übertragungsprotokolle
- **TCP** (Transmission Control Protocol):
  - Zuverlässig
  - Geordnet
  - Robust gegenüber Fehlern
  - Verwendet man, wenn die Daten sicher ankommen sollen
- **UDP** (User Datagram Protocol):
  - Schnell
  - Fire & Forget
  - vergleichsweise einfach implementierbar
  - Gut für Echtzeitdaten, die sowieso ständig aktualisiert werden

Client-Server Model

- Zentraler Server wartet auf Verbindungen
- Client verbindet sich
- Client fordert etwas an, Server antwortet
- Oder: Server schickt Nachricht an verbundene Clients

- Es können auch mehrere Clients gleichzeitig verbunden sein
- Der Server verwaltet verbundene Clients, trennt ggf. Verbindungen

Sockets

- „Endpunkt“ zum Senden und Empfangen von Daten (Wikipedia)
- Beide Enden einer Verbindung haben einen Socket, schreibe ich auf der einen Seite, kommt es auf der anderen Seite an
- Für uns: Objekt, das Funktionen zur Kommunikation bereitstellt
  - `available()`
  - `read()`
  - `write()`
- WiFiServer und WiFiClient sind Sockets (TCP)
- Die Arduino-Dokumentation erklärt alle verfügbaren Funktionen:

  https://www.arduino.cc/en/Reference/WiFi

Server aufsetzen

- WiFi-Bibliothek einbinden

```
#include <WiFi.h>
```

- Zugangsdaten definieren

```
#define PORT 1515
#define SSID "MeinNetzwerk"
#define PASSWORD "geheim123"
```

- Server einrichten

```
WiFiServer server(PORT);
// in setup():
WiFi.mode(WIFI_AP);
WiFi.softAP(SSID, PASSWORD);
```

- Access Point einrichten

- Server und AP starten

```
WiFi.begin();
server.begin();
```

Serverlogik

- Objekt für Client erstellen
- Warten, bis sich ein Client verbindet
- Solange der Client Daten sendet, empfange Sie und gib sie aus

```
WiFiClient client; // ganz oben


// in loop():
if(!client.connected())
{
    client = server.available();
}
else
{
    while(client.available())
    {
        byte data = client.read();
        Serial.print(data);
    }
}
```

Daten strukturieren

```
// loop():
byte temperature = SuperSensor.read();
client.write(temperature);

byte humidity = OtherSensor.read();
client.write(humidity); // ooops!
```

Daten strukturieren

```
#define DATA_LENGTH 2
#define INDEX_TEMPERATURE 0
#define INDEX_HUMIDITY 1


// loop():
byte data[DATA_LENGTH];
data[INDEX_TEMPERATURE] = SuperSensor.read();
data[INDEX_HUMIDITY] = OtherSensor.read();
client.write(data, DATA_LENGTH);
```

Daten strukturieren

```
// loop():

int temperature = SuperSensor.read();

int humidity = OtherSensor.read();

String message = “temp:” + String(temperature) + “;”;

message += “humi:” + String(humidity) + “;”;

client.print(message);
```

Paketheader

- Wird jedem Paket vorangestellt
- Bei komplexeren Projekten sinnvoll
- Enthält z.B.:
  - Client ID (bei mehreren Clients)
  - Art und Länge der Daten
  - Checksumme
  - Timestamp

Fragen?