M5Stack mit WiFi und JSON-Auswertung

Wir brauchen nur das M5Stack-Modul. Erstellt einen neuen Projektordner, wählt diesen für VSC aus und erstellt mit PlatformIO ein neues Projekt „M5Stack-Wifi-JSON“.

Wählt wieder das M5Stack-Core-ESP32-Board aus und wählt die Arduino-Einstellung. Deselektiert den Haken bei dem Default-Ordner und wählt euer angelegtes Verzeichnis für das Projekt (den Ordner, der in VSC geöffnet wurde).

Zusätzlich sind die Libraries

- WiFi.h
- HTTPClient.h
- ArduinoJson.h

notwendig. Bindet diese in das platformio.ini ein.

Eure platformio.ini sollte so aussehen:

```
[env:m5stack-core-esp32]
platform = espressif32
board = m5stack-core-esp32
framework = arduino
upload_speed = 921600
targets = upload
upload_protocol = esptool
monitor_speed = 115200
lib_deps =
        M5Stack
    HTTPClient
    ArduinoJson
```

Die Aufgabe spricht einen im Netz verfügbaren Joke-Server an (URL siehe Code). Dieser Server schickt eine Antwort in JSON-Struktur verpackt zurück.

Zum Test könnt ihr die URL in einen Browser eingeben, um die zurückgelieferte Datenstruktur zu sehen. Die Antworten können entweder nur ein Satz sein (type=“single“) oder in Form von Frage-Antwort (type=“twopart“). Im ersten Fall steht der Text in „joke“, im zweiten Fall setzen sich die beiden Sätze aus „setup“ und „delivery“ zusammen.

Für das HWS-WLAN reicht eine ‚normale‘ Konfiguration nicht aus.

```
#define EAP_IDENTITY "yourname"  // your username for WLAN-Access
#define EAP_USERNAME "yourname"  // repeat of the identity
#define EAP_PASSWORD "password"  //your Eduroam password
const char *ssid = "Heinrich-Wieland-Schule";    // HWS SSID

...
const String url = "https://v2.jokeapi.dev/joke/Programming";
...

// in setup
WiFi.disconnect(true);  // disconnect form wifi to set new wifi connection
WiFi.mode(WIFI_STA);    // init wifi mode
WiFi.begin(ssid, WPA2_AUTH_PEAP, EAP_IDENTITY, EAP_USERNAME, EAP_PASSWORD);

int count = 0;
while (WiFi.status() != WL_CONNECTED && count < 10) {
    delay(1000);
}
if (WiFi.status() != WL_CONNECTED && count >= 10) { // no connection after 10 sek
    M5.Lcd.setTextColor(TFT_RED, TFT_BLACK);
    M5.Lcd.setCursor(0, 0);
    M5.Lcd.print('!');
    M5.Lcd.print("WiFi STATION Failed:\n configure Correctly");
}
M5.Lcd.setTextColor(TFT_GREEN, TFT_DARKGREY);
M5.Lcd.fillScreen(0);
M5.Lcd.setCursor(0, 0);
M5.Lcd.print("\nConnected to " + WiFi.SSID() + "   IP address: " + WiFi.localIP().toString());
...
```

Alternativ (um später mit dem kleinen Raspi-MQTT-Broker zu arbeiten):

```
const char *ssid = "Sng_Mobile";  // Sng_Mobile_5G
const char *password = %Sng-R@uter4IOT_";

...
const String url = "https://v2.jokeapi.dev/joke/Programming";
...

// in setup
WiFi.disconnect(true);  // disconnect form wifi to set new wifi connection
WiFi.begin(ssid, password);

// sonst wie oben
...
```

Der Code (auszugsweise), besteht aus den Definitionen und den beiden geforderten Funktionen setup() und loop(). Zu den Definitionen gehören auch die Funktionsdefinitionen, mit denen nachfolgende eigene Funktionen definiert werden.

Durch den Aufbau kann der Code unterschiedlicher Projekte besser nachvollzogen werden.

Der http-Client holt sich den JSON-Content als String (so kann auch eine http-Seite geholt werden). Da wir wissen, dass der Content ein JSON-String ist, deserialisieren wir ihn.

Anschließend holen wir uns die einzelnen Teile aus dem JSON-Objekt und zeigen es an.

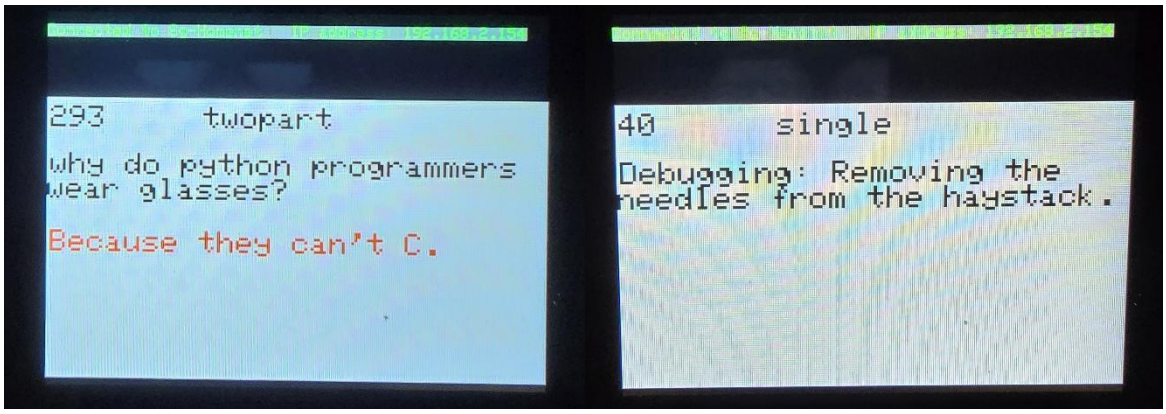
```
.. Definitionen

// Funktions-Definitionen ohne Rumpf
void getJoke();
String formatString(String str);


void setup() {
    ..
}


void loop() {
   getJoke();
   delay(10000); // ca. 10 sek
}


// Funktions-Implementierungen
void getJoke() {
    ..
}

String formatString(String str) {
    ..
}
```

```
String getJoke() {
  HTTPClient http;
  http.useHTTP10(true);
  http.begin(url);
  http.GET();
  String result = http.getString();

  JsonDocument doc;
  DeserializationError error = deserializeJson(doc, result);

  if (error)  {
    Serial.print("deserializeJson() failed: ");
    Serial.println(error.c_str());
  }

  int yLine = 60;  // Positionierung auf dem Screen
  M5.Lcd.setTextColor(BLACK);
  M5.Lcd.fillRect(0, yLine - 5, 320, 180, LIGHTGREY);

  String type = doc["type"].as<String>();
  String joke = doc["joke"].as<String>();
  // ::TODO:: alle anderen auch ..
  http.end();

  M5.Lcd.setCursor(2, yLine);
  M5.Lcd.print(id);
  // ::TODO:: wenn single, dann Ausgabe joke, sonst setup
  // und delivery, mit formatString Text lesbar umbrechen
}

String formatString(String str) {
  // den Text so zerlegen, dass die einzelnen Worte vollständig
  // in eine Zeile passen.
}
```

Erläuterung:

Im Code oben (Funktion `getJoke`) ist ein Beispiel zur Extraktion eines Attributs aus dem JSON-Antwortstring gezeigt (Attribute „type“ und „joke“).

Wertet die Antworten zu den Attributen id, type, joke, setup und delivery aus. Gebt sie passend zum type aus (siehe Anweisungen im Quelltext). Nachfolgend sind zwei Bilder zu einer möglichen Ausgabe zu sehen (in der ersten Zeile steht die IP-Adresse und das Netz in Schriftgröße 1).



Mehr Informationen zum ArduinoJason gibt’s unter

https://arduinojson.org/news/2024/01/03/arduinojson-7/

Aufgabe 1:

Setze alles zusammen. Stelle alle Informationen auf dem TFT-Display dar.

Aufgabe 2:

Erarbeitet eine JSON-Struktur, in der die Messdaten des MPU versendet werden können. Die Daten von der MPU sollen sekündlich erhoben werden und minütlich verschickt werden. Die JSON-Struktur soll alle Informationen pro Sekunde sowie ein Mittelwert pro Messwert enthalten.

Informationen zu den Daten und dem Range siehe

https://shop.m5stack.com/products/env-iii-unit-with-temperature-humidity-air-pressure-sensor-sht30-qmp6988?srsltid=AfmBOop6jtteo-xxyQ2Sk8REqRDBKOVO7WOe2YY6BL12u0bqbPx6MQ4E

Achtet auf eine kompakte Datenübertragung (möglichst wenig unnötige Informationen).