ESP32SmartBoard_HttpSensors

Dieses Arduino-Projekt ermöglicht es dem *ESP32SmartBoard* (siehe Hardware Projekt *<ESP32SmartBoard_PCB>)*, seine Sensordaten über den Embedded WebServer zu präsentieren. Die Werte des Temperatur- und Luftfeuchtesensors (DHT22), des CO2-Sensors (MH-Z19) sowie die übrige Peripherie des Boards werden über HTTP-Seiten dargestellt. Damit lassen sich diese Seiten auf allen gängigen Endgeräten wie PC, Laptop, Tablet oder Smartphone ganz einfach in einem Browser anzeigen. Die Embedded WebServer Firmware basiert auf der Library *ESPAsyncWebServer*, die Zeigerinstrumente werden durch die Open Source Bibliothek *"gauge.min.js"* realisiert (siehe Download-Links am Ende des Dokuments).

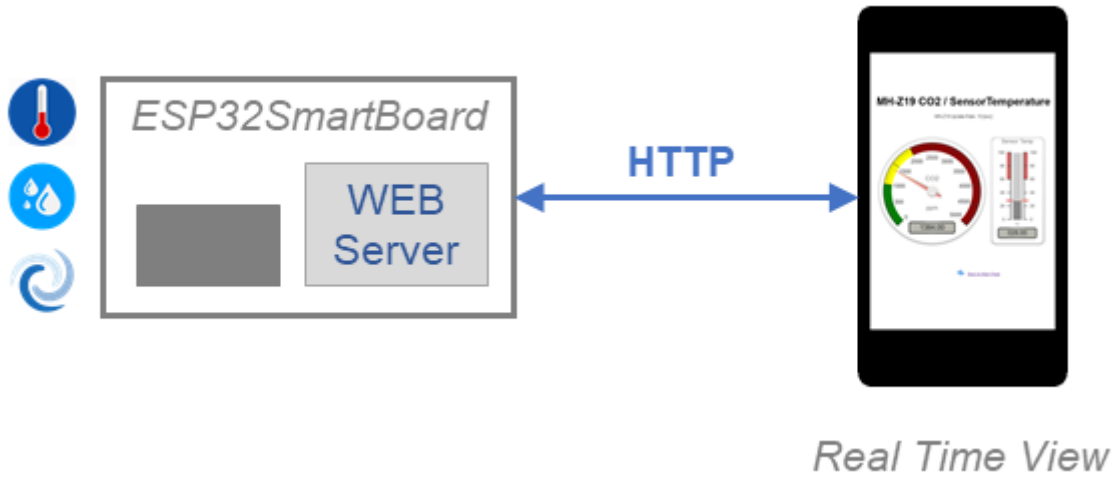
Neben diesem Arduino-Projekt gibt es mit dem Projekt <ESP32SmartBoard_MqttSensors> eine alternative Firmware für das *ESP32SmartBoard*, die eine in Grundzügen ähnliche Funktionalität realisiert. Vom hier beschrieben Projekt unterscheidet sie sich dadurch, dass sie die Werte der Sensoren als MQTT Nachrichten versendet. Die beiden Projekte wurden für folgende, unterschiedlichen Use-Cases entworfen:

- Dieses Projekt: ESP32SmartBoard_HttpSensors
  Einfach zu nutzen, übersichtliche Anzeige aktueller Sensorwerte
  Stand-alone Lösung mit Embedded WebServer, Präsentation der momentanen Sensorwerte über HTTP-Seiten in Echtzeit, direkter Zugriff auf das *ESP32SmartBoard* von Endgeräten wie PC, Laptop, Tablet oder Smartphone

- Alternative: ESP32SmartBoard_MqttSensors
  Komplexeres Setup, unterstützt dafür aber Auswertung und Analyse historischer Daten
  Teil eines Gesamtsystems aus mehreren *ESP32SmartBoards*, jedes Board übermittelt seine Daten per MQTT an einen (gemeinsamen) Broker, der die Daten in eine Datenbank schreibt (z.B. InfluxDB), von da aus lassen sie sich über grafische Dashboards anzeigen und auswerten (z.B. Grafana)

Projekt-Überblick

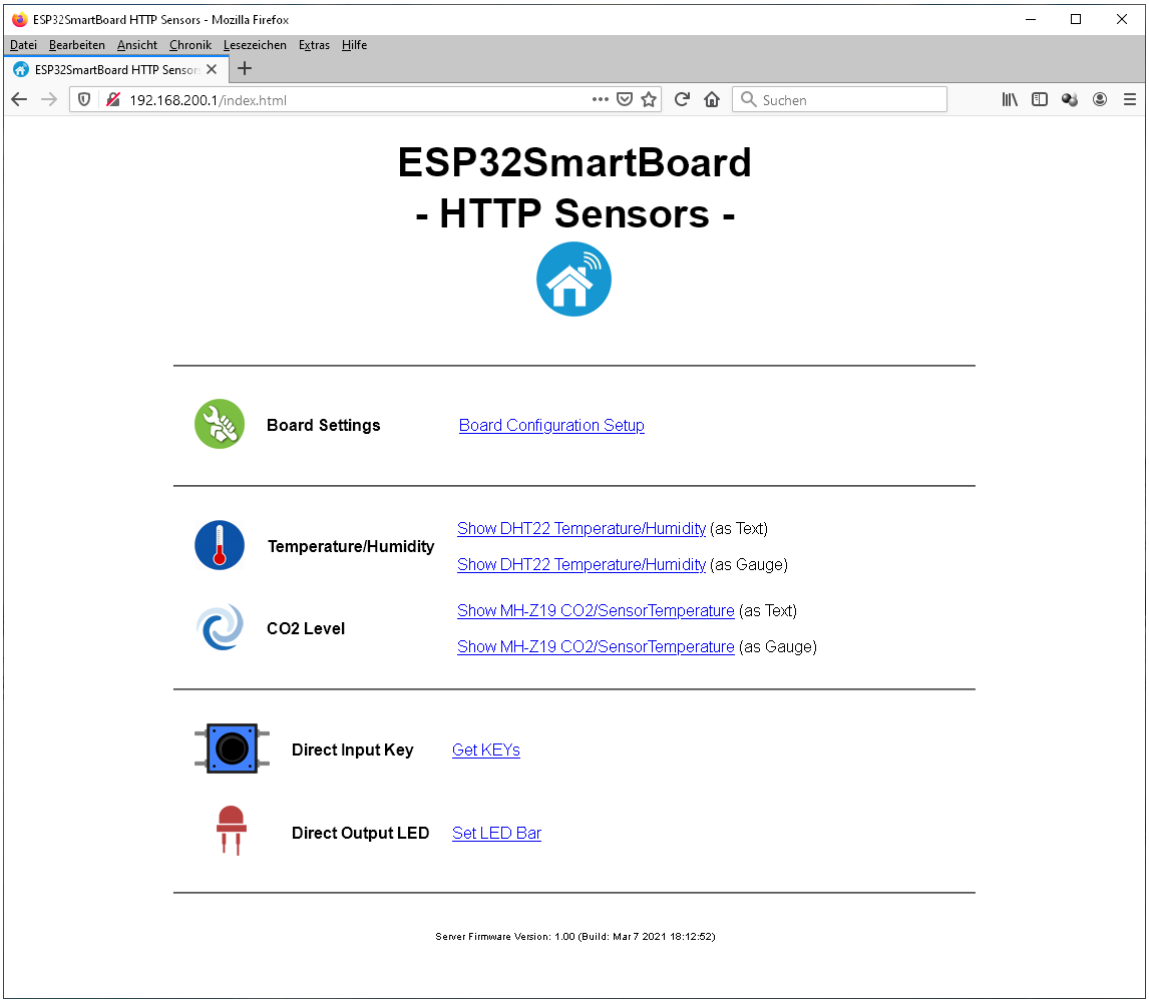
Das *ESP32SmartBoard* etabliert einen Embedded WebServer und ermöglicht darüber Clients im lokalen Netzwerk Zugriff auf die Werte des Temperatur- und Luftfeuchtesensors (DHT22), des CO2-Sensors (MH-Z19) sowie auf die übrige Peripherie des Boards. Die dafür notwendigen Netzwerk-Einstellungen beschreibt der Abschnitt "WLAN-Konfiguration" weiter unten.



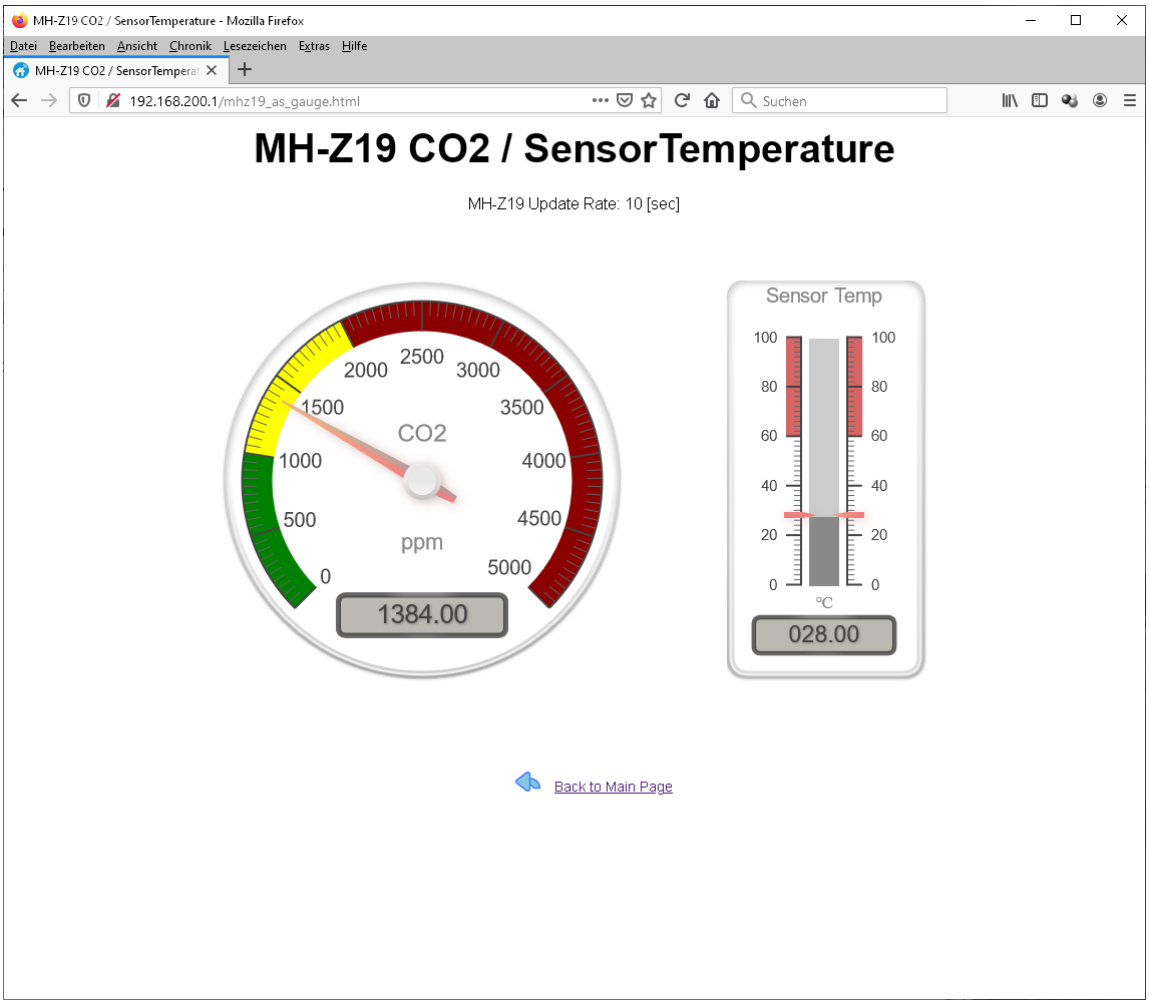
Der Embedded WebServer des *ESP32SmartBoard* interagiert mit folgenden HTTP-Seiten:

index.html	Hauptmenu
board_settings.html	Konfiguration des Boards
dht22_as_gauge.html	Temperatur- und Luftfeuchtesensors (DHT22) als grafische Messinstrumente
dht22_as_text.html	Temperatur- und Luftfeuchtesensors (DHT22) in textueller Form
mhz19_as_gauge.html	CO2 Level und Sensor Temperatur (MH-Z19) als grafische Messinstrumente
mhz19_as_text.html	CO2 Level und Sensor Temperatur (MH-Z19) in textueller Form
board_input_keys.html	Aktueller Zustand der Taster KEY0 und KEY1
board_output_leds.html	Setzen der LEDs

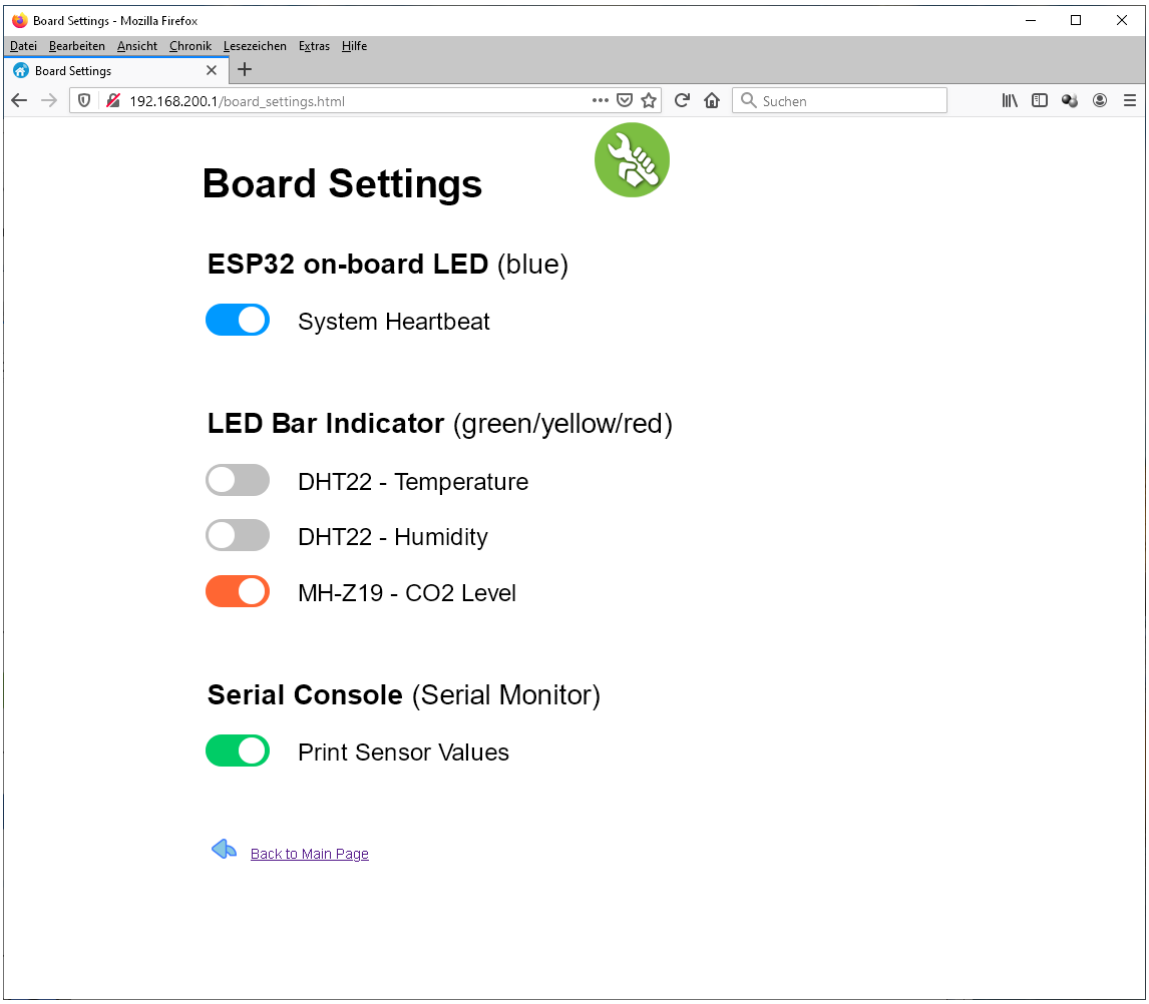
Um die Startseite `index.html` aufzurufen, muss wie üblich nur die IP-Adresse des *ESP32SmartBoard* im Browser eingegeben werden, z.B. `192.168.200.1` (die Default-Adresse des *ESP32SmartBoard*)



Über das Menü der Startseite lassen sich alle oben gelisteten Funktionen des Boards aufrufen.



Das Bild zeigt beispielhaft die Anzeige des CO2 Levels mittels grafischem Zeigerinstrument.



Über die Seite *"board_settings.html"* lässt sich unter anderem konfigurieren, welcher Messwert an der LED Bar des Boards dargestellt werden soll. Folgende LED Bar Quellen können ausgewählt werden:

- Temperatur (DHT22)
- Luftfeuchte (DHT22)
- CO2 Level (MH-Z19)

Die Default-Quelle wird definiert mit der Konstanten:

```
DEFAULT_LED_BAR_INDICATOR = kLedBarMhz19Co2Level;
```

WLAN-Konfiguration Sektion

Das *ESP32SmartBoard* kann sowohl als AccessPoint (AP) ein eigenes WLAN aufspannen, als auch im Client Mode (CM, Station) arbeiten und sich in ein existierendes WLAN einbuchen. Die Auswahl zwischen AP und CM erfolgt mitfolgender Konfigurations-Einstellung:

```
const int  CFG_WIFI_AP_MODE = 0;  // 0 = Device is Station/Client, 1 = Device is AP
```

Die Laufzeit-Konfiguration für den Client Modus (CM, Station) erfolgt über den folgenden Konfigurations-Abschnitt:

```
// WIFI Station/Client Configuration
const char*  WIFI_STA_SSID            = "YourNetworkName";
const char*  WIFI_STA_PASSWORD        = "YourNetworkPassword";
IPAddress    WIFI_STA_LOCAL_IP(192,168,30,100);   // DHCP: IP(0,0,0,0)
```

Wird das Board als Access Point (AP) betrieben, definiert folgender Abschnitt die Laufzeit-Konfiguration:

```
// WIFI AccessPoint Configuration
const char*  WIFI_AP_SSID             = "ESP32SmartBoard";
const char*  WIFI_AP_PASSWORD         = "HttpSensors";
IPAddress    WIFI_AP_LOCAL_IP(192,168,200,1);     // IPAddr for Server if running
in AP Mode
const int    WIFI_AP_CHANNEL          = 1;        // WIFI Channel, from 1 to 13
const bool   WIFI_AP_HIDDE_SSID       = false;    // true = Hide SSID
const int    WIFI_AP_MAX_CONNECTIONS  = 4;        // max. simultaneous connected
stations
```

Wenn das *ESP32SmartBoard* als AccessPoint (AP) ein eigenes WLAN aufspannt, muss das Endgerät auf dem die HTML-Seiten angezeigt werden sollen (Laptop, Tablet, Smartphone), in dieses WLAN eingebucht werden. Dazu ist auf dem Gerät das unter `WIFI_AP_SSID` definierte WLAN auszuwählen. Als Passwort ist das unter `WIFI_AP_PASSWORD` festgelegte Passwort zu verwenden.

Applikations-Konfiguration Sektion

Am Anfang des Sketches befindet sich folgender Konfigurations-Abschnitt:

```
const int  CFG_ENABLE_NETWORK_SCAN  = 1;
const int  CFG_ENABLE_DI_DO         = 1;
const int  CFG_ENABLE_DHT_SENSOR    = 1;
const int  CFG_ENABLE_MHZ_SENSOR    = 1;
const int  CFG_ENABLE_STATUS_LED    = 1;
```

Hiermit lässt sich die Laufzeit-Ausführung der zugehörigen Code Abschnitte freischalten (=1) bzw. sperren (=0). Das vermeidet das Auftreten von Laufzeitfehlern bei Boards, auf denen nicht alle Komponenten bestückt sind (insbesondere, wenn der DHT22 oder der MH-Z19 nicht vorhanden sind).

Dateien für den ESPAsyncWebServer auf dem ESP32SmartBoard

Der Embedded WebServer *ESPAsyncWebServer* benötigt zur Laufzeit Zugriff auf verschiedene HTML-Dateien, Bilder und JavaScript Dateien. Alle diese Dateien befinden sich im Ordner *'data'* des aktuellen Sketches. Um diese Dateien auf das *ESP32SmartBoard* zu übertragen, ist das *"Arduino ESP32 filesystem uploader"* Plugin (https://github.com/me-no-dev/arduino-esp32fs-plugin/releases) für die Arduino IDE erforderlich. Das Plugin kopiert alle Dateien aus dem Ordner *'data'* des aktuellen Sketches in das SPIFFS Filesystem im ESP32 Flash Speichers.

Zur Installation des Plugins ist lediglich der Inhalt des ZIP Files *'ESP32FS'* in das Verzeichnis *'Arduino\Tools'* der IDE zu entpacken.

Der Filetransfer wird aus der IDE heraus über *"Tools -> ESP32 Sketch Data Upload"* gestartet.

Kalibrierung des CO2 Sensors MH-Z19

Eine falsch ausgeführte Kalibrierung kann dazu führen, dass der Sensor unbrauchbar wird. Es ist daher wichtig, die Funktionsweise der Kalibrierung zu verstehen.

Der Sensor ist für einen Einsatz im 24/7 Dauerbetrieb konzipiert. Er unterstützt die Modi AutoCalibration, Kalibrierung per Hardware-Signal (manuell ausgelöst) und Kalibrierun per Software-Kommando (ebenfalls manuell ausgelöst). Das *ESP32SmartBoard* nutzt davon die Modi AutoCalibration und manuelle Kalibrierun per Software-Kommando. Unabhängig von der jeweiligen Methode wird durch die Kalibrierung sensor-intern der Referenzwert von 400 ppm CO2 gesetzt. Eine Konzentration von 400 ppm gilt als CO2-Normalwert in der Erdatmosphäre, also als typischer Wert der Außenluft im ländlichen Raum.

(1) AutoCalibration:

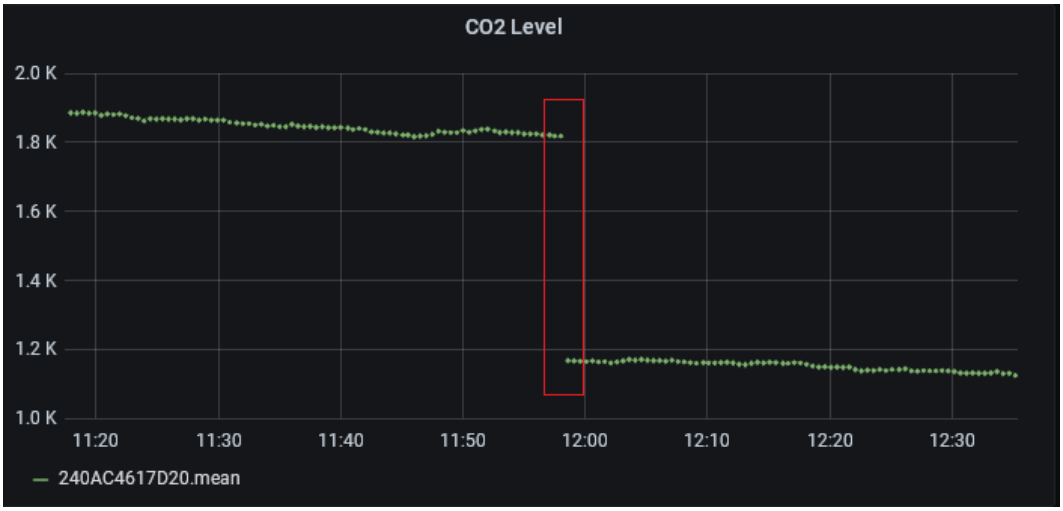
Bei der AutoCalibration überwacht der Sensor permanent die gemessenen CO2 Werte. Der jeweils niedrigste innerhalb von 24 Stunden gemessene Wert wird dabei als Referenzwert von 400 ppm CO2 interpretiert. Diese Methode ist vom Sensor-Hersteller empfohlen für den Einsatz des Sensors in normalen Wohn- oder Büro-Räumen, die regelmäßig gut gelüftet werden. Dabei wird implizit davon ausgegangen, dass beim Lüften die Raumluft komplett getauscht wird und somit die CO2-Konzentration im Raum bis auf den Normalwert der Erdatmosphäre / Außenluft abfällt.

In seinem Datasheet weist der Sensor-Hersteller jedoch explizit darauf hin, dass die AutoCalibration Methode nicht für den Einsatz in landwirtschaftlichen Gewächshäusern, Ställen, Kühlschränken usw. genutzt werden kann. Hier sollte die AutoCalibration disabled werden.

Im *ESP32SmartBoard* Sketch dient die Konstante `DEFAULT_MHZ19_AUTO_CALIBRATION` zum aktivieren (*true*) bzw. deaktivieren (*false*) der AutoCalibration Methode. Die AutoCalibration Funktion wird einmalig beim Systemstart gesetzt. Bei Bedarf kann die AutoCalibration Methode per Taster invertiert werden. Dazu sind folgende Schritte notwendig:

1. KEY0 drücken und durchgehend bis Schritt 3 gedrückt halten
2. Reset am ESP32DevKit drücken und wieder loslassen
3. KEY0 noch weitere 2 Sekunden gedrückt halten
4. KEY0 loslassen

Hinweis: Der AutoCalibration Modus kann insbesondere in den ersten Tagen nach Inbetriebnahme des Sensors zu stark sprunghaften Veränderungen des CO2 Wertes führen. Nach einiger Zeit im 24/7 Dauerbetrieb nimmt dieser Effekt immer mehr ab.



(Dieser Grafana-Screenshot stammt aus dem Projekt <ESP32SmartBoard_MqttSensors>. Die Sensor-Routinen sind jedoch in beiden Projekten identisch.)

Die durch den AutoCalibration Modus verursachten spontanen Unstetigkeiten lassen sich durch eine manuelle Kalibrierung des Sensors vermeiden.

(2) Manuelle Kalibrierung:

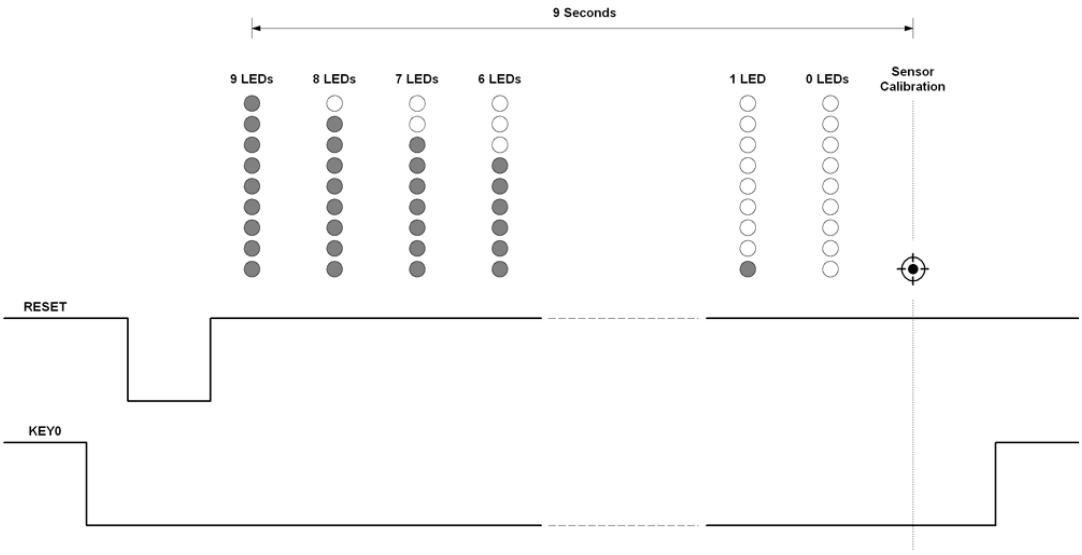
Vor einer manuellen Kalibrierung muss der Sensor für mindestens 20 Minuten in einer stabilen Referenzumgebung mit 400 ppm CO2 betrieben werden. Diese Forderung ist im Amateur- und Hobby-Bereich ohne definierte Kalibrierumgebung nur näherungsweise realisierbar. Dazu kann das *ESP32SmartBoard* im Freien an einem schattigen Platz oder in einem Raum in der Nähe eines geöffneten Fensters ebenfalls im Schatten betrieben werden. In dieser Umgebung muss das *ESP32SmartBoard* für mindestens 20 Minuten arbeiten, bevor die Kalibrierung ausgelöst werden kann.

(2a) Direkte Kalibrierung:

Arbeitet das *ESP32SmartBoard* bereits seit mindestens 20 Minuten in der 400 ppm Referenzumgebung (im Freien, am geöffneten Fenster), kann der Sensor direkt kalibriert werden. Dazu sind folgende Schritte notwendig:

1. KEY0 drücken und durchgehend bis Schritt 4 gedrückt halten
2. Reset am ESP32DevKit drücken und wieder loslassen
3. das *ESP32SmartBoard* startet einen Countdown von 9 Sekunden, dabei blinkt die LED Bar im Sekundentakt und zeigt die noch verbleibende Zeit in Sekunden an
4. KEY0 die gesamte Zeit über gedrückt halten bis der Countdown abgeschlossen ist und die LED Bar die abgeschlossene Kalibrierung mit 3x schnellen Flashen quittiert
5. KEY0 loslassen

Wird KEY0 während des Countdowns losgelassen, bricht das *ESP32SmartBoard* die Prozedur ab, ohne den Sensor zu kalibrieren.



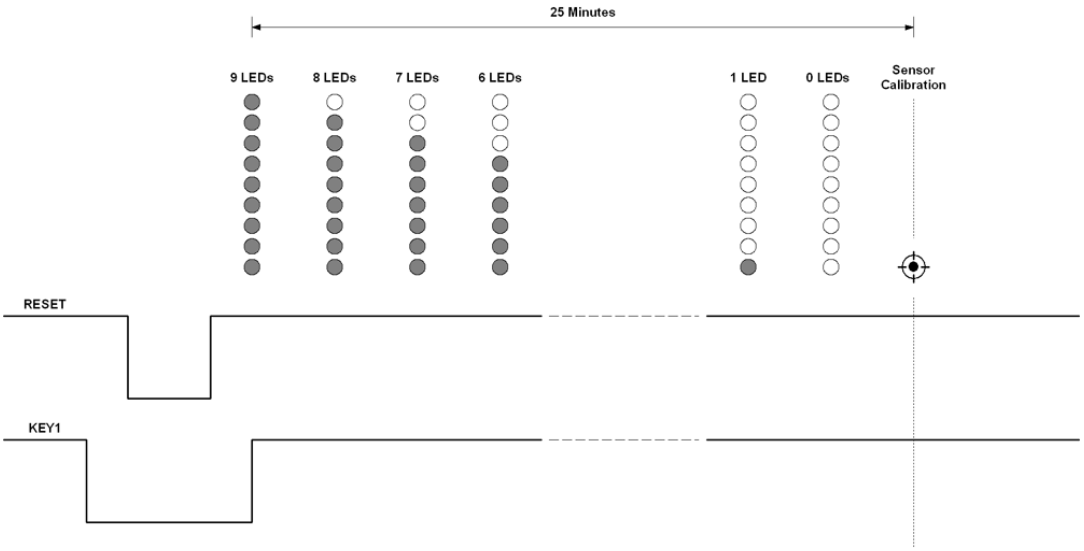
(2b) Unbeaufsichtigte, zeitverzögerte Kalibrierung:

Nachdem das *ESP32SmartBoard* in der 400 ppm Referenzumgebung (im Freien, am geöffneten Fenster) platziert wurde, kann die Kalibrierung unbeaufsichtigt und zeitverzögert durchgeführt werden. Dazu sind folgende Schritte notwendig:

1. KEY1 drücken und durchgehend bis Schritt 2 gedrückt halten
2. Reset am ESP32DevKit drücken und wieder loslassen
3. KEY1 loslassen

Das *ESP32SmartBoard* startet einen Countdown von 25 Minuten. Während des Countdowns blinkt die LED Bar im Sekundentakt und zeigt die noch verbleibende Zeit an (25 Minuten / 9 LEDs = 2:46 Minuten / LED). Nach Ablauf des Countdowns wird der Kalibrievorgang ausgelöst und mit 3x

schnellen Flashen der LED Bar quittiert. Anschließend wird das *ESP32SmartBoard* per Software-Reset neu gestartet.



Laufzeitausgaben im seriellen Terminal-Fenster

Zur Laufzeit werden alle relevanten Informationen im seriellen Terminal-Fenster (115200Bd) ausgegeben. Insbesondere während des Systemstarts (Sketch Funktion *setup()*) werden hier auch Fehlermeldungen angezeigt, die ggf. auf eine fehlerhafte Softwarekonfiguration zurückzuführen sind. Diese Meldungen sollten insbesondere während der Erstinbetriebnahme auf jeden Fall beachtet werden.

In der Hauptschleife des Programms (Sketch Funktion *main()*) werden zyklisch die Werte des DHT22 (Temperatur und Luftfeuchtigkeit) und des MH-Z19 (CO2 Level und Sensor Temperatur) angezeigt. Darüber hinaus informieren auch hier wieder Meldungen über evtl. Probleme beim Zugriff auf die Sensoren.

Durch aktivieren der Zeile *#define DEBUG* am Anfang des Sketches werden weitere, sehr detaillierte Laufzeitmeldungen angezeigt. Diese sind insbesondere während der Programmentwicklung bzw. zur Fehlersuche sehr hilfreich. Durch auskommentieren der Zeile *#define DEBUG* werden die Ausgaben wieder unterdrückt.

ESPAsyncWebServer Basics

Der *ESP32SmartBoard* Sketch nutz die Library *ESPAsyncWebServer* (https://github.com/me-no-dev/ESPAsyncWebServer). Diese Library wurde speziell für Embedded Devices designend. Sie unterstützt ein breites Spektrum an Funktionalitäten, verschieden Datenquellen und REST Schnittstellen. Der ESP32SmartBoard Sketch nutzt davon folgende Funktionalitäten:

(1) Aufbereitung und Auslieferung von HTML-Dateien mit dynamischen Inhalten

Dazu wird die URL der dynamisch aufzubereitenden HTML-Dateien in der *"on"* Methode des *AsyncWebServer* Objektes für die Operation *"GET"* registriert und mit einem Request-Handler verknüpft:

```
AsyncWebServer.on(<file_url>, HTTP_GET, <RequestHandler>(AsyncWebServerRequest*
pServerRequest_p));
```

Bei Auslieferung einer Seite via *"GET"* Operation werden die in der Seite enthaltenen symbolische Platzhalter (*"%SENSOR_VALUE%"*) mit Hilfe des Request-Handlers dynamisch durch die aktuellen

Werte der Status- und Sensordaten ersetzt. Somit werden die Daten direkt in den HTML-Code der Seite eingefügt. Aus Sicht des Browsers stellt sich die dynamisch aufbereitete Seite damit wie eine normale HTML-Seite dar.

(2) Auslieferung statischer Dateien

Für alle Dateien, deren URL nicht dediziert über die *"on"* Methode des *AsyncWebServers* registriert wurde, ruft der *AsyncWebServer* die allgemeine Methode *"onNotFound"* auf. Hierrüber liefert der *ESP32SmartBoard* Sketch alle übrigen Dateien wie statische HTML-Seiten (ohne dynamische Inhalte), Bilder und ähnliche Files aus:

```
WebServerSync.onNotFound([](AsyncWebServerRequest* pServerRequest_p)
{
    String strPath = pServerRequest_p->url();
    pServerRequest_p->send(SPIFFS, strPath.c_str());
});
```

(3) Realisierung von REST-Services

Ähnlich zur Aufbereitung und Auslieferung von HTML-Dateien mit dynamischen Inhalten werden auch die URLs für REST-Services mittels *"GET"* und *"POST"* Operationen in der *"on"* Methode des *AsyncWebServer* Objektes registriert und mit einem Request-Handler verknüpft:

```
AsyncWebServer.on(<rest_url>, HTTP_GET, <RequestHandler>(AsyncWebServerRequest*
pServerRequest_p));
```

Mit der *"GET"* Operation werden Daten vom *ESP32SmartBoard* abgefragt. Dies wird insbesondere zum Abfragen aktueller Status- und Sensorwerte genutzt.

```
AsyncWebServer.on(<rest_url>, HTTP_POST, <RequestHandler>(AsyncWebServerRequest*
pServerRequest_p));
```

Mit der *"POST"* Operation werden Daten an das *ESP32SmartBoard* übergeben. Dies wird insbesondere zum Setzen von Konfigurationseinstellungen genutzt.

AsyncWebServer auf dem ESP32SmartBoard

Der Hauptzweck der HTML-Seiten ist die Präsentation von dynamischen Status- und Sensordaten sowie das Senden von Steuer- oder Konfigurationsdaten an das *ESP32SmartBoard*. Der Sketch nutz dazu folgende Mechanismen:

1. Bei Auslieferung einer Seite mit Hilfe der *"GET"* Operation werden die in der Seite enthaltenen symbolische Platzhalter (*"%SENSOR_VALUE%"*) durch die aktuellen Werte der Status- und Sensordaten ersetzt. Somit werden die Daten direkt in den HTML-Code der Seite eingefügt. Aus Sicht des Browsers stellen sie sich damit wie statische HTML-Werte dar.

2. Während der Laufzeit werden Status- und Sensordaten zyklisch per *"GET"* Operation erneut abgefragt. Das Abfragen der Werte vom *ESP32SmartBoard* sowie das Aktualisieren der Anzeigewerte in den HTML-Elementen erfolgt durch in die Seite eingebetteten JavaScript Code.

3. Das Senden von Steuer- oder Konfigurationsdaten an das *ESP32SmartBoard* erfolgt mit Hilfe von *"POST"* Operationen. Das Senden der entsprechenden Requests erfolgt ebenfalls durch in die Seite eingebetteten JavaScript Code.

Im Detail sind die oben aufgeführten Mechanismen wie folgt umgesetzt:

(1) Auflösen symbolischer Platzhalter (*"%SENSOR_VALUE%"*)

Um dynamisch Status- und Sensorwerte in eine HTML-Seite einzubetten, werden die betreffenden Stellen im HTML-Code mit symbolischen Platzhaltern gekennzeichnet (*"%SENSOR_VALUE%"*). Am Beispiel der HTML-Datei *"/dht22_as_text.html"* ist der Platzhalter für den Temperaturwert des DHT22-Sensors wie folgt in den HTML-Code eingebettet:

```
<span id="dht22_temperature">%DHT22_TEMPERATURE%</span>
```

Den Einstiegspunkt zur Auflösung symbolischer Platzhalter auf dem *ESP32SmartBoard* bildet die *"on"* Methode des *AsyncWebServer* Objektes. Dabei wird die zu modifizierende HTML-Datei (*"/dht22_as_text.html"*) für die Operation *"GET"* registriert und der Request-Handler (*"AsyncWebServerRequest*"*) als anonyme Funktion implementiert:

```
pWebServer_g->on("/dht22_as_text.html", HTTP_GET, [](AsyncWebServerRequest*
pServerRequest_p)
{
    pServerRequest_p->send(SPIFFS, "/dht22_as_text.html", String(), false,
AppCbHdlrHtmlResolveSymbolDht22Value);
});
```

Die *"send"* Methode liest die angeforderte HTML-Datei vom SPIFFS Filesystem des ESP32 Flash Speichers und nutzt die angegebene Callback Funktion (hier *"AppCbHdlrHtmlResolveSymbolDht22Value"*) zum Auflösen der im HTML-Code eingefügten symbolischen Platzhalter (*"%SENSOR_VALUE%"*). Vom *AsyncWebServer* wird die Callback Funktion für jedes aufzulösende Symbol separat aufgerufen. Das Symbol wird dabei als Aufrufparameter übergeben. Als Rückgabewert liefert die Callback Funktion den in die HTML-Seite einzubettenden Status- und Sensorwert.

```
// (vereinfachte Version, reale Implementierung siehe ESP32SmartBoard_HttpSensors.ino)
String  AppCbHdlrHtmlResolveSymbolDht22Value (const String& strSymbol_p)
{

String  strValue;

    if (strSymbol_p == "DHT22_TEMPERATURE")
    {
        strValue = strDht22Temperature_g;
    }
    else if (strSymbol_p == "DHT22_HUMIDITY")
    {
        strValue = strDht22Humidity_g;
    }
    else
    {
        strValue = "?";
    }

    return (strValue);

}
```

(2) Zyklische Abfrage der Status- und Sensordaten zur Laufzeit

Die zyklische Abfrage der Status- und Sensordaten zur Laufzeit erfolgt per REST Service mittels *"GET"* Operation. Dazu werden in die HTML-Seite eingebettete JavaScript Funktionen per Intervall-Timer zyklisch aufgerufen. Beispielsweise ist die JavaScript Funktion zur Abfrage des aktuellen Temperaturwertes vom DHT22-Sensor wie folgt implementiert:

```
function FunTimerHandlerGetDht22Temperature()
{
    var XHttpReq = new XMLHttpRequest();
    XHttpReq.onreadystatechange = function()
    {
        if (this.readyState == 4 && this.status == 200)
        {
            document.getElementById("dht22_temperature").innerHTML =
this.responseText;
        }
    };
    XHttpReq.open("GET", "/get_dht22?temperature", true);
    XHttpReq.send();
}
```

Durch die *"GET"* Operation wird auf dem *ESP32SmartBoard* die URL *"/get_dht22?temperature"* aufgerufen. Der Teil vor dem "?" Trennzeichen bildet die Basis-URL (*"/get_dht22"*). Der Teil nach dem Trennzeichen ist die Parameter-Liste (*"temperature"*). Den Einstiegspunkt zum Auflösen der URL bildet die *"on"* Methode des *AsyncWebServer* Objektes. Dabei wird die Basis-URL (*"/get_dht22"*) für die Operation *"GET"* registriert und dem Request-Handler *"AppCbHdlrHtmlOnRequestDht22"* zugeordnet:

```
// Handle a GET request to <ESP_IP>/get_dht22?<ValueType>
pWebServer_g->on("/get_dht22", HTTP_GET, AppCbHdlrHtmlOnRequestDht22);
```

Der Request-Handler *"AppCbHdlrHtmlOnRequestDht22"* ist in *ESP32SmartBoard_HttpSensors.ino* wie folgt implementiert:

```
void  AppCbHdlrHtmlOnRequestDht22 (AsyncWebServerRequest* pServerRequest_p)
{

#define PARAM_NAME_TEMPERATURE      "temperature"
#define PARAM_NAME_HUMIDITY         "humidity"

String  strValue;

    if (pServerRequest_p != NULL)
    {
        // get values on <ESP_IP>/get_dht22?<ValueType>
        if (pServerRequest_p->hasParam(PARAM_NAME_TEMPERATURE))
        {
            strValue = AppGetDht22Value(PARAM_NAME_TEMPERATURE);
        }
        else if (pServerRequest_p->hasParam(PARAM_NAME_HUMIDITY))
        {
            strValue = AppGetDht22Value(PARAM_NAME_HUMIDITY);
        }
        else
        {
            strValue = "{unknown}";
        }
    }

    pServerRequest_p->send(200, "text/plain", strValue);
}
```

(3) Setzen von Ausgängen und Schreiben von Konfigurationseinstellungen

Das Setzen von Ausgängen und Schreiben von Konfigurationseinstellungen zur Laufzeit erfolgt per REST Service mittels *"POST"* Operation. In der HTML-Seite wird das mit Hilfe von eingebetteten JavaScript Funktionen realisiert. Beispielsweise ist die JavaScript Funktion zum Setzen des Heartbeat-Modus (Blinken der blauen LED auf dem ESP32DevKit) wie folgt implementiert:

```
function FunSliderSysLedHeartbeat(HtmlElement_p)
{
    var Value;

    if ( HtmlElement_p.checked )
    {
        Value = "1";
    }
    else
    {
        Value = "0";
    }

    var XHttpReq = new XMLHttpRequest();
    XHttpReq.open("POST", "/settings?option=" + HtmlElement_p.id + "&value=" +
Value, true);
    XHttpReq.send();
}
```

Durch die *"POST"* Operation wird auf dem *ESP32SmartBoard* die URL *"/settings?option=1"* aufgerufen. Der Teil vor dem "?" Trennzeichen bildet die Basis-URL (*"/settings"*). Der Teil nach dem Trennzeichen ist die Parameter-Liste (*"option", value=1*). Den Einstiegspunkt zum Auflösen der URL bildet die *"on"* Methode des *AsyncWebServer* Objektes. Dabei wird die Basis-URL (*"/settings"*) für die Operation *"POST"* registriert und dem Request-Handler *"AppCbHdlrHtmlOnRequestDht22"* zugeordnet:

```
// Handle a POST request to <ESP_IP>/settings?option=<OptionType>&value=<SetValue>
pWebServer_g-> on("/settings", HTTP_POST, AppCbHdlrHtmlOnTellBoardSettings);
```

Der Request-Handler *"AppCbHdlrHtmlOnTellBoardSettings"* ist in *ESP32SmartBoard_HttpSensors.ino* wie folgt implementiert:

// (vereinfachte Version, reale Implementierung siehe ESP32SmartBoard_HttpSensors.ino)

```
void  AppCbHdlrHtmlOnTellBoardSettings (AsyncWebServerRequest* pServerRequest_p)
{

#define PARAM_NAME_OPTION   "option"
#define PARAM_NAME_VALUE    "value"

String  strOptionType;
String  strValue;
int     iValue;

    if (pServerRequest_p != NULL)
    {
        // get values on <ESP_IP>/settings?option=<OptionType>&value=<SetValue>
        if (pServerRequest_p->hasParam(PARAM_NAME_OPTION) &&
            pServerRequest_p->hasParam(PARAM_NAME_VALUE))
        {
            strOptionType = pServerRequest_p->getParam(PARAM_NAME_OPTION)->value();
            strOptionType.toUpperCase();
            strValue = pServerRequest_p->getParam(PARAM_NAME_VALUE)->value();
            iValue = strValue.toInt();

            if (strOptionType == String("SYSLED_HEARTBEAT"))
            {
                fStatusLedHeartbeat_g = (iValue == 0) ? false : true;
            }
            else if (strOptionType == String("PRINT_SENSOR_VALUES"))
            {
                fPrintSensorValues_g = (iValue == 0) ? false : true;
            }
        }
    }

    pServerRequest_p->send(200);
}
```

(4) Redirect zu "index.html"

Um die Startseite *index.html* aufzurufen, muss wie üblich nur die IP-Adresse des *ESP32SmartBoard* im Browser eingegeben werden, z.B. *"192.168.200.1"*. Dieser Aufruf wir von Sketch intern zur Startseite *index.html* weitergeleitet.

Der Redirect erfolgt in der Methode *"onNotFound"* des *AsyncWebServers.* Dazu wird die URL *"/"* ausgewertet und durch ein HTML Redirect beantwortet:

```
<meta http-equiv="refresh" content="0; url='/index.html'" />
```

Dieser HTML-Code ist Bestandteil des Feldes *const char root_html[]*, das fest im Sketch *ESP32SmartBoard_HttpSensors.ino* definiert ist. Die zur Auslieferung notwendige Funktionalität in der Methode *"onNotFound"* ist wie folgt implementiert:

```
// (vereinfachte Version, reale Implementierung siehe ESP32SmartBoard_HttpSensors.ino)
pWebServer_g->onNotFound([](AsyncWebServerRequest* pServerRequest_p)
{
    String strPath = pServerRequest_p->url();
    if (strPath == String("/"))
    {
        // Resolve <ESP32SmartBoard_IP> to 'index.html'
        pServerRequest_p->send_P(200, "text/html", root_html);
    }
    else
    {
        // ...
    }
});
```

Das Redirect der IP-Adresse zur Startseite *index.html* hat den Vorteil, dass auch für die Startseite eine dedizierte *"on"* Methode des *AsyncWebServer* Objektes definiert werden kann, um auch für diese Seite symbolischen Platzhalter auflösen zu können. Die Startseite *index.html* besitzt beispielsweise symbolische Platzhalter für Versionsnummer und Build-Timestamp, die auf diese Art aufgelöst werden.

Verwendete Drittanbieter Komponenten

1. **Embedded WebServer**

   Für den Embedded WebServer Firmware wird die Bibliothek *ESPAsyncWebServer* verwendet: https://github.com/me-no-dev/ESPAsyncWebServer

2. **Grafische Zeigerinstrumente**

   Die in den HTML-Seiten verwendeten Zeigerinstrumente für Temperatur, Luftfeuchtigkeit und CO2-Level werden durch die Open Source Bibliothek *"gauge.min.js"* realisiert. Eine ausführliche Dokumentation, Beispiele sowie der Quellcode sind auf https://canvas-gauges.com/ verfügbar.

   Die Bibliothek *"gauge.min.js"* liegt zusammen mit den HTML-Dateien und Bildern im Ordner *'data'* des Sketches und wird von da aus in das SPIFFS Filesystem im ESP32 Flash Speicher übertragen.

3. **MH-Z19 CO2 Sensor**

   Für den MH-Z19 CO2 Sensor werden folgende Treiberbibliotheken verwendet:
   https://www.arduino.cc/reference/en/libraries/mh-z19/
   https://www.arduino.cc/en/Reference/SoftwareSerial

   Die Installation erfolgt mit dem Library Manager der Arduino IDE.

4. **DHT Sensor**

   Für den DHT Sensor (Temperatur, Luftfeuchtigkeit) wird die Treiberbibliothek von Adafruit verwendet. Die Installation erfolgt mit dem Library Manager der Arduino IDE.