Gewerblich-Technische
Schule Offenburg
Friedrich-August-Haselwander-Schule

Internet der Dinge

MIT WEMOS D1 MINI ESP8266

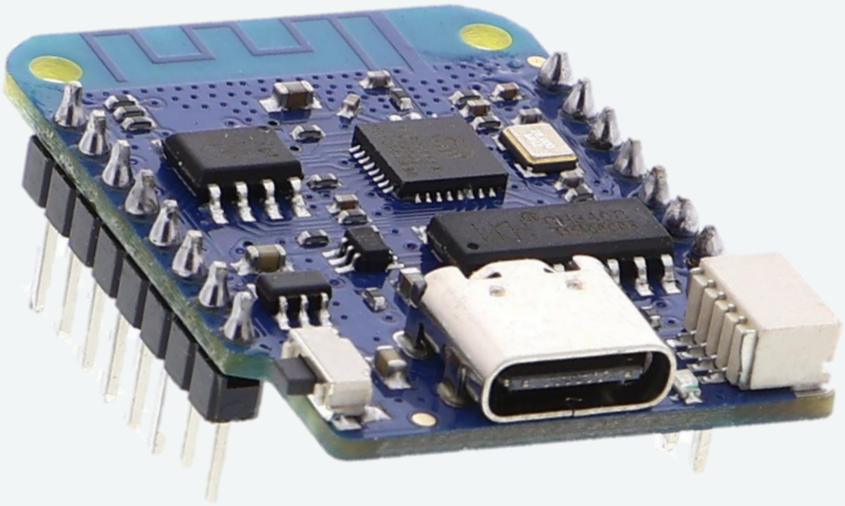


ABBILDUNG 2

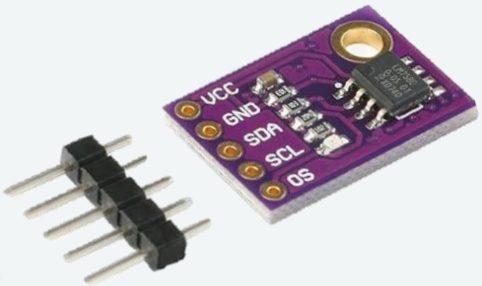


ABBILDUNG 1

Ipek, Atilla & Roser, Kevin
27.06.2025| GEWERBLICH-TECHNISCHE SCHULE OFFENBURG, E1FI1 2024/2025, HUG CHRISTOPHER

INHALTSVERZEICHNIS

I. ABBILDUNGSVERZEICHNIS

II. ABKÜRZUNGSVERZEICHNIS

Frontend - grafische Benutzeroberfläche (GUI), mit der Ihre Benutzer direkt interagieren können.

Backend - bezeichnet in der Regel den funktionalen Teil eines digitalen Produkts wie beispielsweise Webseiten oder Apps und bezieht sich meist auf Client Server.

REST-API - Eine REST-API (Representational State Transfer Application Programming Interface) ist eine Art von API, die den REST-Architekturstil befolgt

IDE - "Integrated Development Environment" (Integrierte Entwicklungsumgebung).

Watchdog - bezeichnet eine Funktion zur Ausfallerkennung eines Systems.

React-SPA – React- Single Page Application ist eine Webanwendung, die auf einem einzigen HTML-Dokument basiert, das beim ersten Laden der Seite komplett vom Server geladen wird.

CDN - Content Delivery Network

UI - interaktive Benutzer Oberfläche (User Interface)

III. LITERATURVERZEICHNIS

CJMCU LM75 Temperatursensor. (2025). Von https://www.analog.com/media/en/technical-documentation/data-sheets/lm75.pdf abgerufen

JSON.org. (2025). Von https://www.json.org/json-de.html abgerufen

opc-router.de. (2025). Von https://www.opc-router.de/was-ist-json/ abgerufen

React Google Charts. (2025). Von https://react-google-charts.com/ abgerufen

React-js. (2025). Von https://reactjs.org/ abgerufen

REST api. (2025). Von https://restfulapi.net/ abgerufen

Thonny. (2025). Von https://thonny.org/ abgerufen

w3schools CSS. (2025). Von https://www.w3schools.com/css/css_intro.asp abgerufen

w3schools HTML. (2025). Von https://www.w3schools.com/html/html_intro.asp abgerufen

w3schools JSON. (2025). Von https://www.w3schools.com/js/js_json.asp abgerufen

w3schools XML. (2025). Von https://www.w3schools.com/xml/xml_whatis.asp abgerufen

Wemos D1 Mini. (2025). Von https://www.wemos.cc/en/latest/d1/d1_mini.html abgerufen

Wikipedia. (kein Datum). Von https://de.wikipedia.org/wiki/Prinzipien_objektorientierten_Designs#cite_note-1 abgerufen

1 EINLEITUNG

1.1 MOTIVATION/PROBLEMSTELLUNG

In Zeiten der Digitalisierung und steigender Nachfrage von Automatisierung wird es immer wichtiger Daten mithilfe von Sensoren nicht nur Aufzunehmen, sondern diese den Nutzern auch in Echtzeit und übersichtlich zu Visualisieren. Dies ist gut umsetzbar mit einem Microcontroller-Board wie dem Wemos D1 Mini ESP8266, kombiniert mit einem kompatiblen Temperatursensor Modul. Durch das integrierte WLAN, den niedrigen Stromverbrauch und seiner Kompakten Bauweise, eignet er sich besonders gut für das Projekt.

Das Ziel war es mithilfe der oben genannten Komponenten und Micro Python eine kompakte Lösung zur Temperaturüberwachung mithilfe eines Microcontrollers zu entwickeln. Diese Aufgabe lag kein konkretes Problem zugrunde, es handelte sich vielmehr um eine realitätsnahe Übung, welche uns das Zusammenspiel von Hardware, Software und Webtechnologie näherbringen sollte.

Auch wenn in diesem Fall kein konkretes Problem festgelegt wurde, wäre unser Projekt eine gute Lösung für zum Beispiel, generelle Raumklimaüberwachung oder die Temperaturüberwachung eines Gewächshauses.

1.2 ZIELGRUPPE

Die Zielgruppe des Projekts können Schüler oder Lehrer sein, welche sich für die Thematik Internet of Things oder Mikrocontroller-Programmierung interessieren. Auch könnte man es einsetzten, wenn man beispielsweise ein Hobby hat, bei welchem man Temperaturangaben benötigt wie zum Beispiel Botanik. Hierbei muss man jedoch bedenken, dass der Sensor eine gewisse Abweichung hat, weswegen der Einsatz lediglich für Projekte einsetzbar ist, welche keine sehr genaue Messung voraussetzen.

2 PROJEKTBESCHREIBUNG

2.1 PROJEKTIDEE & ZIELE

Ziel unseres Projekts ist es, Temperaturdaten in zwei unterschiedlichen Darstellungsformen bereitzustellen und zu speichern:

- ***Kurzzeitansicht***: die letzten 45 Sekunden, in 5 Sekunden Abständen
- ***Langzeitansicht***: die letzten 4 Tage, in 1 Stunde abständen

Die Daten werden durch einen HTTPS-Server zur Verfügung gestellt, der direkt auf dem Wemos D1 Mini gehostet wird.

Die Kurzzeitansicht präsentiert die Temperaturwerte in tabellarischer, kompakter Form. Die Langzeitansicht stellt die Daten grafisch dar, wodurch sie so eine übersichtliche und verständliche Auswertung der Temperaturentwicklung über einen längeren Zeitraum ermöglicht.

Der Zugriff der Daten erfolgt plattformunabhängig über das lokale Netzwerk per Web-Server.

Zusätzlich ist das Projekt robust mit der Watchdog-Überwachung und skalierbar durch die gewählte Softwarearchitektur mit dem modularen Aufbau.

Im *Blockschaltbild 1* wird das Gesamtsystem inklusive Sensor, Aktor und Netzwerkverbindung übersichtlich dargestellt.

2.2 TECHNISCHE SPEZIFIKATIONEN

- **Mikrocontroller**: ESP8266 (Wemos D1 Mini)
- **Temperatursensor**: TMP75 (I2C-Interface), +- 0.5° Genauigkeit
- **Programmiersprache**: MicroPython
- **Webserver**: Integrierter HTTP-Server
- **Frontend**: React-basierte Single-Page-Application
- **Datenformat**: JSON und XML für API-Endpoints

2.3 HARDWAREARCHITEKTUR

Uns wurde als zentrale Steuereinheit der Wemos D1 mini zur Verfügung gestellt. Er ist ein kompakter Mikrocontroller auf Basis des ESP8266EX-Chips mit integriertem WLAN. Das Board verfügt über 4MB Flash-Speicher sowie einen 32 Bit-Prozessor und bietet eine Vielzahl an I/O Pins für externe Komponenten.[1]

In unserem Projekt nutzen wir die Serial-Data- (SDA) und Serial-Clock-Leitung (SCL) des I2C Busses, um den CJMCU-75 Temperatursensor auszulesen und die aktuelle Umgebungstemperatur zu erfassen.[2]

Als Beispiels Aktor dient die OnBoard-LED des Wemos D1 Mini, welche über den GPIO-Pin 2 angesteuert werden kann.

2.4 SOFTWAREARCHITEKTUR

Unser Programm folgt einer klaren Trennung zwischen Frontend und Backend:

- **Frontend**: React-SPA gehostet auf GitHub Pages CDN
- **Backend**: ESP8266 mit minimaler HTML-Struktur und REST-API

2.4.1 IDE UND VERWENDETE LIBRARIES

Für die Entwicklung des Backend unseres Projekts wurde die Thonny IDE verwendet, da sie sich besonders gut für MicroPython eignet.[3] Visual Studio Code kam für das Frontend zum Einsatz, um die JSX-Dateien und die Weboberfläche komfortabel zu gestalten.

Das Backend des Projektes nutzt ausschließlich Micro-Python-kompatible Bibliotheken:

- „*network, socket, time, gc*“ – System und Netzwerkfunktionen
- eigenentwickelte Module: „*SoftwareWatchdog, LEDController, DataManager, WebHandler, WiFiManager, ESP8266WebServer*“

[1] (Wemos D1 Mini)
[2] (CJMCU LM75 Temperatursensor)
[3] (Thonny)

Zur Visualisierung der Daten haben wir uns entschieden, die Bibliothek *„react-google-charts“* zu nutzen.[4] Mit ihr realisieren wir einfache und flexible Diagramme, mit denen der Client interagieren kann.

2.4.2 Backend

Die Entwicklung unseres Programms erfolgte entlang eines klaren objektorientierten Konzepts unter Beachtung des SRP-Prinzips (Single Responsibility Principle). Die Klassen sind so strukturiert, dass jede eine klare abgegrenzte Verantwortung innerhalb des Systems übernimmt.[5]

Das *Klassendiagramm 1* zeigt die modulare Aufteilung der Backend Komponenten.

- Die TemperatureStation-Klasse (*main.py*) ist die zentrale Koordinationseinheit des gesamten Systems und somit zuständig für:
  - **Systeminitialisierung**
  - **Lebenszyklusverwaltung**
  - **Hauptschleife**

  Der Ablauf dieser zentralen Einheit ist in *Flussdiagramm 2* veranschaulicht.

- Die DataManager-Klasse (*DataManager.py*) ist verantwortlich für die gesamte Datenverwaltung:
  - **Datenpersistierung**
  - **Datenerfassung**
  - **Ringpuffer-Logik**

  Der grundlegende Prozessablauf ist in *Flussdiagramm 1* dargestellt.

[4] (React Google Charts)
[5] (Wikipedia)

➢ Die WebHandler-Klasse (*WebHandler.py*) verarbeitet HTTP-Anfragen und stellt sie in verschiedenen Datenformaten bereit:

- API-Endpoints

Endpoint	Method	Response	Zweck
/	GET	HTML	Web-Interface
/api/kurzzeit	GET	JSON	JSON-Daten der letzten 10 Messwerte (5s Intervall)
/api/langzeit	GET	JSON	JSON-Daten der letzten 96 Messwerte (1h Intervall)
/api/xml	GET	XML	XML-Format der Kurzzeit-Daten

- Response-Format (Beispiel Response):

```
// JSON Response
[
["2024-01-15 14:30:25", 23.5],
["2024-01-15 14:30:30", 23.7]
]
```

```
// XML Response
<temperaturen>
        <eintrag>
                <zeitpunkt>2024-01-15 14:30:25</zeitpunkt>
                <wert>23.5</wert>
        </eintrag>
</temperaturen>
```

Sobald eine HTTP-Anfrage an das Backend gestellt wird, aktiviert sich die OnBoard LED. Sie dient dabei als visuelles Signal und zeigt an, dass eine aktive Kommunikation mit dem System erfolgt.

- Für die Verwaltung der Netzwerkverbindung mit Verbindungsüberwachung ist die WifiManager-Klasse (*WifiManager.py*) zuständig. Zusätzlich stellt sie sicher, dass bei einer erfolgreichen Verbindung die aktuelle IP-Adresse des ESP automatisch in einer *launchsettings.json*-Datei gespeichert wird. Dies ermöglicht eine automatische Kopplung von Backend und Frontend über das lokale Netzwerk.
- Mit der SoftwareWatchdog-Klasse (*SoftwareWatchdog.py*) wurde eine Überwachungsfunktion zur Resilienz implementiert:
    - Überwacht regelmäßige "Lebenszeichen" des Systems
    - Automatischer Neustart bei Systemhängern
    - Konfigurierbares Timeout (Standard: 20 Sekunden)
- Mit der LEDController-Klasse (*LED-Controller.py*) lassen sich einfach neue LEDs modular hinzufügen.
- Das Hilfs-Modul *TMP75.py* ist für die Temperatursensor-Integration da und liest mithilfe der I2C-Kommunikation an der Adresse 0x48 den Temperatur Wert aus. Das *Struktogramm 1* zeigt die Verarbeitung des TMP75-Sensors.
- Die *Config.py* ist für die *Konfiguration 1* des Systems da.

DATENARCHITEKTUR

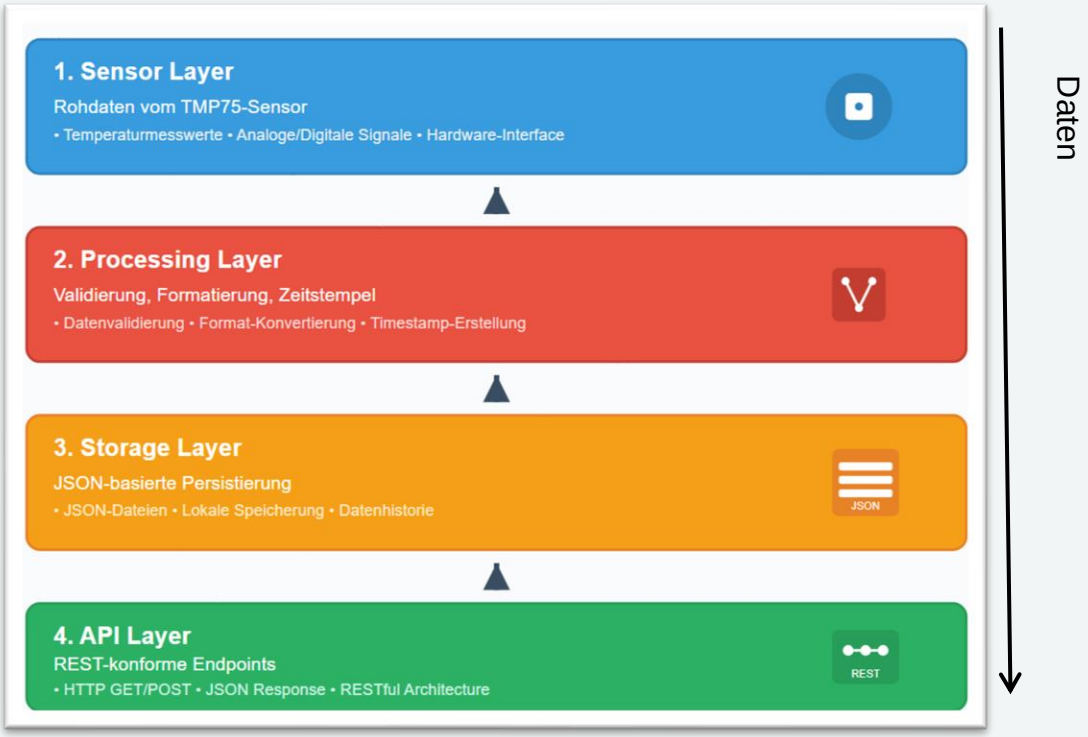
Technisch werden die beiden Datenansichten Langzeitansicht und Kurzzeitansicht so realisiert, dass es zwei JSON-Dateien gibt, in denen die jeweiligen Messwerte persistiert werden. Die Entscheidung, wann neue Messwerte in die jeweilige Datei geschrieben werden, erfolgt durch die Methode:

- *should_measure()*

Sie prüft, ob seit der letzten Speicherung genügend Zeit vergangen ist, basierend auf zwei separat konfigurierbaren Zeitintervallen in der *Config.py* - Datei.

Bei Programmstart werden die vorhandenen JSON-Dateien initial eingelesen und die darin enthaltenen Messwerte in Listen geladen. Diese Listen sind die Datenspeicher während der Laufzeit des Programmes. Bei jedem neuen Messwert im *Speichervorgang 1*, wird überprüft ob die maximal Listen Größe erreicht wurde und falls wird der älteste Eintrag jeweils entfernt und dann der neueste Messwert

hinzugefügt. Ist die maximale Größe der Liste noch nicht erreicht, wird der neue Messwert angehängt. Hierdurch entsteht eine einfache, aber effektive Ringpuffer-logik, welche eine kontinuierliche Datenaktualisierung ermöglicht die speicherressourcenschonend für den Mikrocontrollers ist.



2.4.3 Frontend

Das Frontend basiert auf einer React-SPA, die über GitHub Pages CDN geladen wird. Dieses Vorgehen wurde gewählt, um den begrenzten Ressourcen des WEMOS D1 Mini ESP8266 entgegenzusteuern und gleichzeitig eine stabile Systemarchitektur zu gewährleisten ist:

- **Speicherplatz-Effizienz**: Da der Speicherplatz auf dem ESP ohnehin schon begrenzt ist, ist nur eine *minimalistische HTML-Struktur 1* auf dem ESP.
- **RAM-Optimierung**: Das vollständige Laden der JavaScript-Datei direkt auf dem ESP hätte zu Speicherüberläufen (Out-of-Memory) geführt, weshalb die Auslagerung für unser Vorhaben nötig war.
- **Systemstabilität**: Durch diese Auslagerung des Frontend erreichen wir, dass der ESP nur eine statische HTML-Datei und API-Endpunkte bereitstellen muss. Das reduziert die Komplexität und erhöht die Laufzeitstabilität.

Nachteil dieser Auslagerung ist, der ESP benötigt eine Anbindung an das Internet, bzw. an das GitHub Pages CDN, ohne solchen Funktioniert die Weboberfläche nicht.

Die eigentliche Anwendungslogik wird dann clientseitig ausgeführt und kommuniziert über REST-APIs mit dem ESP.

FUNKTIONSWEISE DES FRONTENDS

Die Anwendung besteht im Wesentlichen aus einer React-Komponente, sie übernimmt die wesentlichen Funktionen:

- Interaktive *Visualisierung von Temperaturdaten 1* in einem Liniendiagramm mit Hilfe von der Bibliothek *„react-google-charts“.*
- Kontinuierlicher Datenabruf aktueller und Langzeit Messwerte vom ESP über die beiden REST-API-Endpoints (*/api/langzeit und /api/kurzzeit)*, im 5 Sekunden Takt.

Technische Funktionsweise: *Flussdiagramm 3*

2.5 VERWENDETE WEB-TECHNOLOGIEN

2.5.1 HTML (HYPERTEXT MARKUP LANGUAGE)

HTML ist die Abkürzung für HyperText Markup Language. Sie ist eine textbasierte Auszeichnungssprache und der grundlegende Baustein für Webseiten. Mit ihr wird die Struktur von Webinhalten definiert. Browser wie Google Chrome nutzen HTML, um Websites anzuzeigen und zu verknüpfen. Da HTML-Dateien von jedem Betriebssystem mit einem Webbrowser dargestellt werden können, bezeichnet man HTML als eine plattformunabhängige Sprache

Jedes Element auf der Website, zum Beispiel ein Text oder ein Bild, wird zwischen zwei Tags geschrieben, einem Starttag und einem Endtag. Der einzige Unterschied zwischen den beiden Tags ist, dass vor dem Endtag noch ein Schrägstrich steht. Ein Beispiel hierfür ist der Tag <*b*>. Er dient als Starttag mit dem man Text fett darstellen kann, der passende Endtag hierfür ist </*b*>. Der Text, den man zwischen diesen beiden Tags schreibt, wird später auf der Website fett angezeigt.

Ein HTML-Code besteht aus mehreren Tags, welche den Text, Bilder, Tabellen und andere Elemente der Seite strukturieren.

Das Grundgerüst eines HTML-Codes besteht aus einem head und einem body. Im head befinden sich Metainformationen, welche nicht direkt auf der Seite angezeigt werden, ein Beispiel hierfür wäre ein meta-Tag wie <*meta charset="utf-8"*>. Er sorgt dafür, dass die Seite Sonderzeichen wie zum Beispiel die Umlaute ä,ö und ü korrekt darstellen kann. Im body sind alle Elemente enthalten, die man letztendlich auf der Seite sieht, also der Text oder die Bilder.[6]

Beispiel für die Struktur eines HTML-Codes

```
<!DOCTYPE html>
<html>
	<head>
		<title>Titel </title>
	</head>
	<body>
		<h1>Überschrift 1</h1>
		<p>Das ist ein Absatz.</p>
	</body>
</html>
```

2.5.2 CSS (Cascading Style Sheets)

CSS ist die Abkürzung für Cascading Style Sheets. Während HTML dazu genutzt wird, die Struktur von Webseiten festzulegen, wird CSS genutzt um festzulegen wie diese auf der Website angezeigt werden. Man kann mit CSS zum Beispiel, Hintergrundfarbe, Textfarbe oder Größe von einzelnen Textabschnitten definieren. Diese Style Informationen werden in einer externen .css Datei gespeichert und später auf die Website angewandt.

In dieser .css Datei wird mit sogenannten selectors angegeben, welches HTML Element verändert werden soll. Als Beispiel mit dem selector h1 würde man die

[6] (w3schools HTML, 2025)

erste Überschrift der Seite verändern. Dahinter wird dann zum einen die Eigenschaft, welche verändert werden soll und der Wert der Eigenschaft angegeben. Wenn mehrere Eigenschaften geändert werden sollen müssen diese in geschweifte Klammern gesetzt und mit Semikolons getrennt werden.[7]

Ein Beispiel hierfür wäre die Zeile:

```
h1 {color:red; font-size:14px;}
```

mit dieser Zeile würde man die erste Überschrift rot machen und die Schriftgröße dieser auf 14 setzen.

2.5.3 JSON (JavaScript Object Notation)

JSON ist die Abkürzung für JavaScript Object Notation. Es ist ein Datenformat zur Darstellung strukturierter Daten. Das Format von JSON ist komplett unabhängig von Programmiersprachen, verwendet allerdings eine Teilmenge der JavaScript-Syntax. Da es nur aus Text besteht, kann es von nahezu jeder Programmiersprache genutzt werden. [8]

Aufgrund der Kompaktheit und der guten Leserlichkeit wird JSON oft genutzt um Daten auszutauschen.[9]

Ein JSON Dokument beginnt und endet mit geschweiften Klammern ({}), zwischen diesen Klammern werden die Inhalte geschrieben. Im Dokument selber können dann Objekte definiert werden, diese bestehen aus einem oder mehreren Schlüssel-Wert-Paaren. Jedem Schlüssel ist ein Wert zugeordnet, die beiden Komponenten werden mit einem Doppelpunkt getrennt.[10]

[7] (w3schools CSS, 2025)
[8] (JSON.org, 2025)
[9] (w3schools JSON, 2025)
[10] (opc-router.de, 2025)

Ein einfaches Beispiel für die Struktur einer JSON-Datei:

```
{
  "name": "Anna",
  "alter": 25,
  "verheiratet": false
}
```

2.5.4 XML (eXTENSIBLE MARKUP LANGUAGE)

XML ist die Abkürzung für Extensible Markup Language. Sie ist, ähnlich wie HTML, auch eine Markup Language und wurde, wie JSON dazu entwickelt, Daten zu speichern und zu transportieren. Anders als HTML wird XML nicht dazu benutzt, etwas darzustellen, sondern dient lediglich dazu, Daten zu strukturieren und zu beschreiben. Dies bietet eine software- und hardwareunabhängige Methode, Daten zu transportieren und zu teilen.

Die Syntax besteht, wie bei HTML, aus verschiedenen Tags, die geöffnet und mit einem Schrägstrich geschlossen werden müssen. Diese Tags sind allerdings frei wählbar und nicht vordefiniert.

Im Vergleich zu JSON ist XML weniger kompakt, bietet dafür jedoch umfangreichere Möglichkeiten zur Darstellung komplexer und stark verschachtelter Datenstrukturen. Aus diesem Grund wird XML öfter eingesetzt, wenn die zu verarbeitenden Daten besonders komplex sind.[11]

Ein einfaches Beispiel für die Struktur einer XML-Datei:

```
<bibliothek>
        <buch>
                <titel>Der kleine Prinz</titel>
                <autor>Antoine de Saint-Exupéry</autor>
                <erscheinungsjahr>1943</erscheinungsjahr>
        </buch>
</bibliothek>
```

[11] (w3schools XML, 2025)

2.5.5 JavaScript-Framework React

React ist ein JavaScript-Framework das von Facebook entwickelt wurde und zur Erstellung von UIs verwendet wird. React basiert auf dem Konzept von wiederverwendbaren Komponenten und ermöglicht dynamische SPAs. [12]

Grundlegende Konzepte von React:

- **Komponenten-basierte Architektur:** UI wird in kleine wiederverwendbare Bausteine aufgeteilt.
- **Virtual DOM:** Effiziente Aktualisierung der UI durch virtuelle *Document Object Model*, (tatsächliche veränderte Elemente werden aktualisiert).
- **Deklarative Programmierung:** Der Fokus liegt auf der Beschreibung des gewünschten Ergebnisses, ohne die einzelnen Schritte zur Umsetzung explizit anzugeben.
- **State-Management:** Verwaltung von Anwendungszuständen über „Hooks“ wie *useState oder useEffect*.

[12] (React-js)

3 SCHLUSS

3.1 ZUSAMMENFASSUNG DES PROJEKTS

Unser entwickeltes System verbindet moderne Web-Technologien mit hardwarenaher Programmierung zu einer stabilen und erweiterbaren Lösung des Projekts. Unser ESP dient als zentraler HTTP-Server, der dauerhaft mit dem CJMCU-Sensor Temperaturdaten erfasst und diese in zwei Zeitformaten bereitstellt:

- **Kurzzeitspeicher**: 10 Messwerte der letzten 45 Sekunden (5 Sekunden-Intervall)
- **Langzeitspeicher**: 96 Messwerte der letzten 4 Tage (1 Stunde-Intervall)

Wir stellen diese Messwerte bereit über eine REST-API in Form von drei verschiedenen Formaten (JSON, XML und die HTML-Oberfläche).

Unsere Backend-Implementierung folgt den objektorientierten Designprinzipien und ist klar strukturiert, wodurch wir immer eine einfache Weiterentwicklung im Projekt hatten.

Besonders wird unser Projekt durch die Frontend Architektur. Der ESP hostet nur eine minimalistische HTML-Struktur und die komplexere React-Anwendung wird über GitHub Pages CDN (*cross-origin*) geliefert.
Durch diese Lösung überwinden wir die Speicher- und RAM- Limitierungen des Mikrocontrollers, ohne an Funktionalität zu verlieren.

Auch hervorzuheben ist die Ausrichtung auf Systemstabilität und Robustheit. Dies wird durch den Einsatz eines Software-Watchdogs gewährleistet, der das System kontinuierlich überwacht. Zusätzlich sorgt die Implementierung von Ringpuffern für eine durchgängig Overflow-freie Datenspeicherung. Ergänzend dazu ermöglicht die automatische Wifi-Verwaltung mit der IP-Adressen Speicherung in der *launchsettings.json*-Datei eine Verbindung zwischen Backend und Frontend selbst nach Systemneustarts.

Ein bestehendes Problem konnten wir jedoch nicht ganz lösen. Der Temperatursensor bzw. die Kabel haben einen Wackelkontakt, wodurch der Sensor manchmal keinen Wert liefert. Wir konnten diesen Fehler jedoch schwer reproduzieren und uns auch nicht wirklich erklären warum er auftritt bzw. ihn lösen.

Zusammenfassend kann man sagen, unser Projekt stellt eine erfolgreiche Integration verschiedener Technologiebereiche, von Mikrocontroller-Programmierung über Web-Entwicklung bis hin zu modernen Frontend-Frameworks dar. Wir haben einige Herausforderungen damit überwunden und eine Basis für verschiedene weiterführende IoT-Projekte geschaffen.

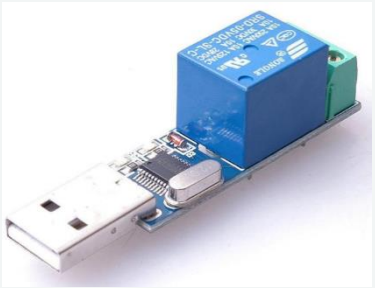
3.2 ERWEITERUNGEN

- HTTP-Client in Python

  Der HTTP-Client sendet, mithilfe des *http.Client* Moduls wiederholt eine http-GET-Anfrage an den Server. Die GET-Anfrage wird alle 5 Sekunden an die Adresse *http://10.1.226.191/api/kurzzeit* gesendet, die Antwort, welche JSON-Daten der letzten 10 Messwerte (5s Intervall) enthält, wird ausgelesen und in *message* gespeichert. Danach wird die message decoded und wieder in ein JSON-Objekt umgewandelt, dieses wird dann ausgegeben. Dieser Prozess wird in einem 5 Sekunden Abstand wiederholt, bis man mithilfe der Tastenkombination „Strg+C“ das Skript stoppt.

  Mithilfe dieses HTTP-Clients, einer Temperatur-Abfrage und eines Relais am Client PCs könnte man dann einen (M) Ventilator steuern. Beispielsweise im Sommer, wenn es zu heiß wird, geht der Ventilator von alleine an am Arbeitsplatz.

  Dies würde über eine serielle Schnittstelle, in Python gehen:



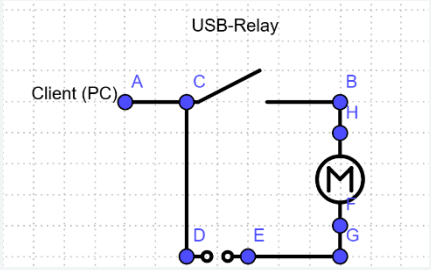


ABBILDUNG 3

```python
import serial
import time

PORT = "COM3"
BAUDRATE = 9600
RELAIS_EIN  = bytes.fromhex("A0 01 01 A2")
RELAIS_AUS  = bytes.fromhex("A0 01 00 A1")

with serial.Serial(PORT, BAUDRATE, timeout=1) as ser:
    print("Relais EIN")
    ser.write(RELAIS_EIN)
    time.sleep(2)

    print("Relais AUS")
    ser.write(RELAIS_AUS)
```

4 KOMMUNIKATIONSPLAN

4.1 TEAMMITGLIEDER

- **Atilla Ipek** – atilla.ipek94@gmail.com
- **Kevin Roser** – kevin.roser.2003@gmail.com

4.2 AUFGABENVERTEILUNG

ATILLA IPEK - SCHWERPUNKT BACKEND & FRONTEND:

- Integration der Datenvisualisierung mit react-google-charts
- Implementierung der REST-API und Webserver-Funktionalität
- Entwicklung der React-SPA und UI-Komponenten
- Entwicklung der objektorientierten Softwarearchitektur
- Implementierung des Software-Watchdogs und der Systemüberwachung
- Dokumentation der Backend-Komponenten und Systemarchitektur
- Implementierung der Frontend-Backend-Kommunikation
- Dokumentation der Frontend-Architektur

KEVIN ROSER – HARDWARE, WEB-TECHNOLOGIEN & HTTP CLIENT:

- Dokumentation der Web-Technologien
- Erstellung der projektbegleitenden Materialien
- Schreiben und Dokumentieren des http-Clients
- Produktion des Projekt-Vorstellungsvideos

GEMEINSAME VERANTWORTLICHKEITEN:

- Projektplanung und Koordination
- Systemintegration und Testing
- Dokumentationserstellung
- Problemanalyse und -lösung

4.3 Genutzte Kommunikationskanäle

GitHub: Zentrale Versionsverwaltung für Code und Dokumentation, Issue-Tracking für Aufgabenverteilung und Fehlerverfolgung

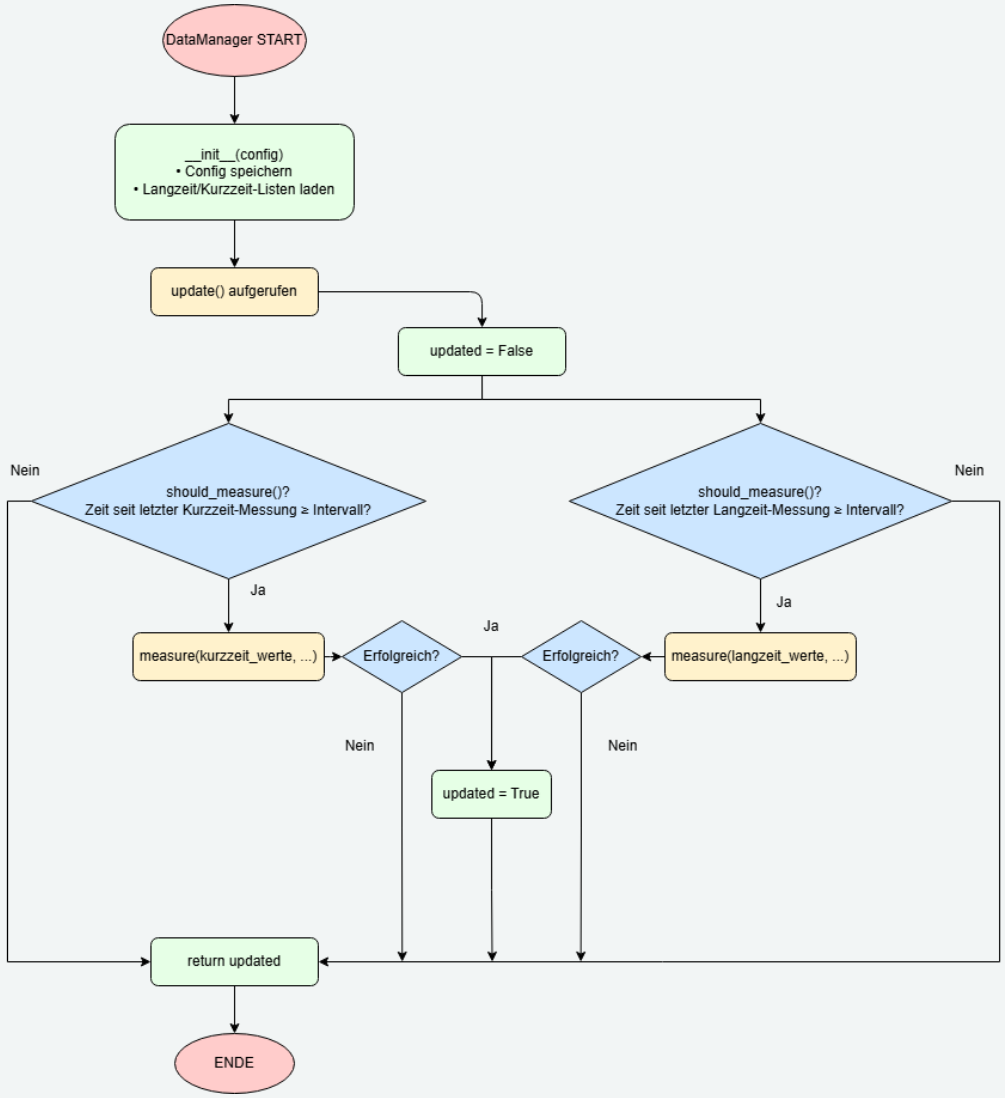
WhatsApp: Schnelle Abstimmung für organisatorische Fragen, Terminkoordination und kurze technische Rückfragen

Persönliche Treffen (Schultage): gemeinsames Debugging und Systemintegration

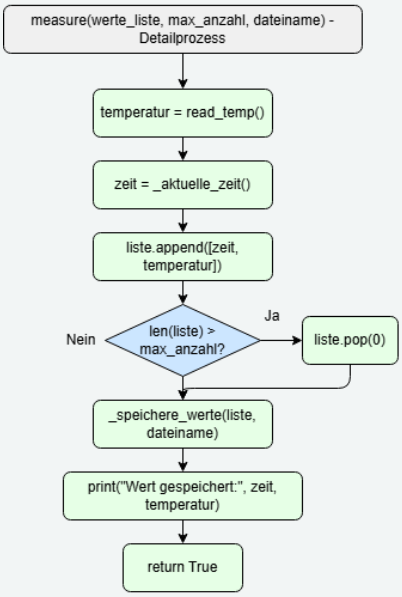
A. ANHANG

A.1 FLUSSDIAGRAMM DES DATAMANAGERS

FLUSSDIAGRAMM 1

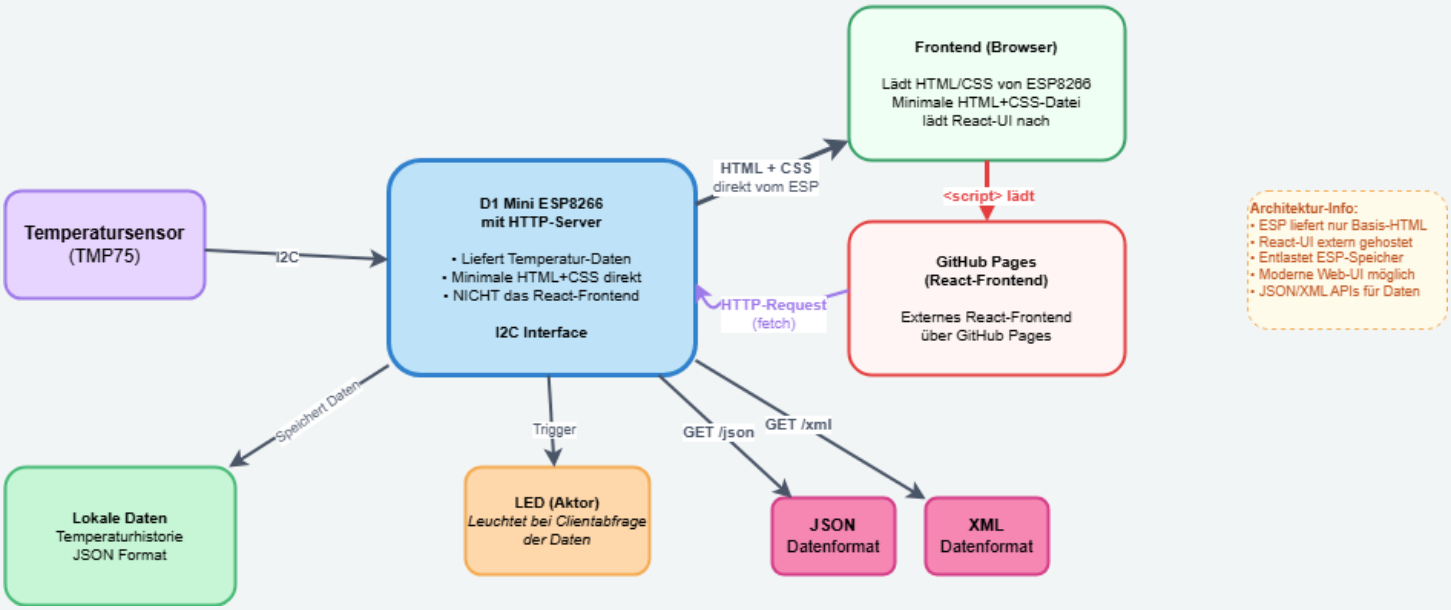


SPEICHERVORGANG 1



A.2 BLOCKSCHALTBILD DES GANZEN SYSTEMS

BLOCKSCHALTBILD 1



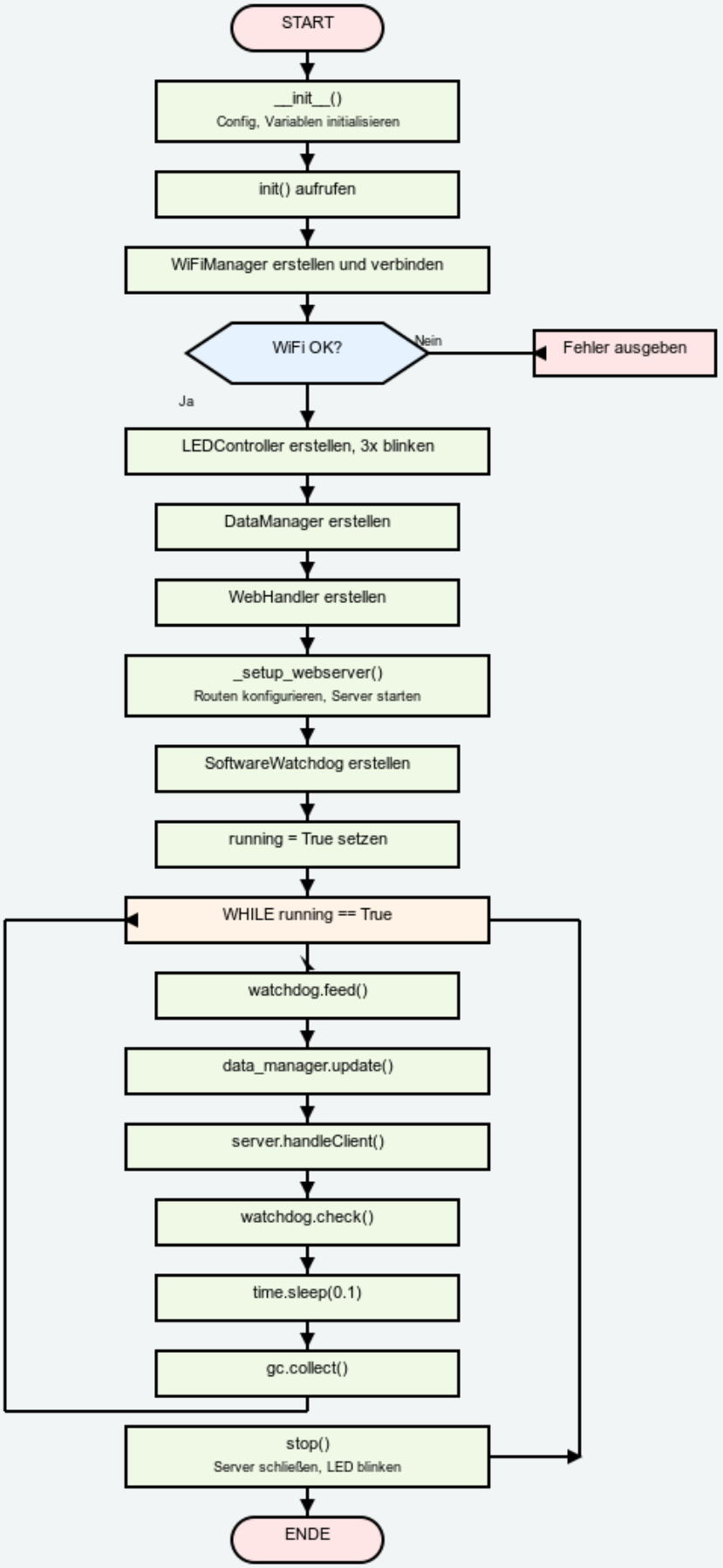
A.3 STRUKTOGRAMM VON TEMPERATURSENSORAUSWERTUNG

STRUKTOGRAMM 1

```
ADDRESS = 0x48
i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000)
function  read_temp()( ) {
    Try
        raw = i2c.readfrom_mem(ADDRESS, 0x00, 2)
        temp_raw = (raw[0] << 4) | (raw[1] >> 4)
        temp_raw & 0x800 != 0
        Wahr                          Falsch
        temp_raw -= 4096              ∅
        temperature = temp_raw * 0.0625
        return round(temperature, 2)
    Catch
        A: Temperatur-Lesefehler
        return None
}
```

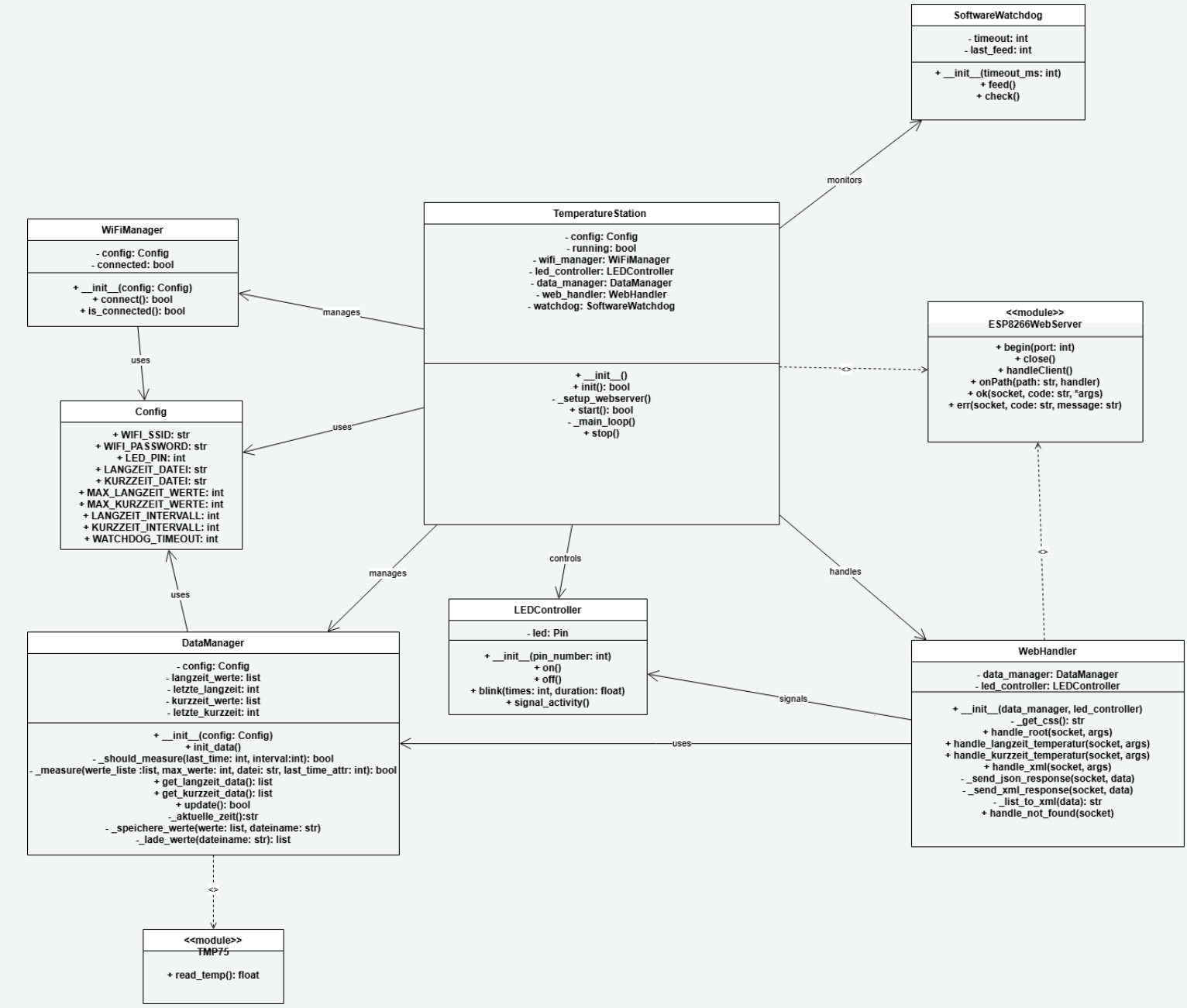
A.4 FLUSSDIAGRAMM VON MAIN-KLASSE

FLUSSDIAGRAMM 2



A.5 KLASSENDIAGRAMM DES GANZEN SYSTEMS

KLASSENDIAGRAMM 1



A.6 KONFIGURATIONSDATEI *CONFIG.PY*

KONFIGURATION 1

```
class Config:
    # WiFi-Einstellungen
    WIFI_SSID = ""  # gsog-iot
    WIFI_PASSWORD = ""  # IOT_Projekt_BFK-S_2022

    # LED-Pin
    LED_PIN = 2

    # Dateipfade
    LANGZEIT_DATEI = "data/langzeitwerte.json"
    KURZZEIT_DATEI = "data/kurzzeitwerte.json"

    # Messwert-Limits
    MAX_LANGZEIT_WERTE = 120
    MAX_KURZZEIT_WERTE = 10

    # Zeitintervalle (in ms)
    LANGZEIT_INTERVALL = 3600000
    KURZZEIT_INTERVALL = 5000

    # Watchdog
    WATCHDOG_TIMEOUT = 20000
```

A.7 MINIMALISTISCHE HTML-STRUKTUR DES ESP

MINIMALISTISCHE HTML-STRUKTUR 1

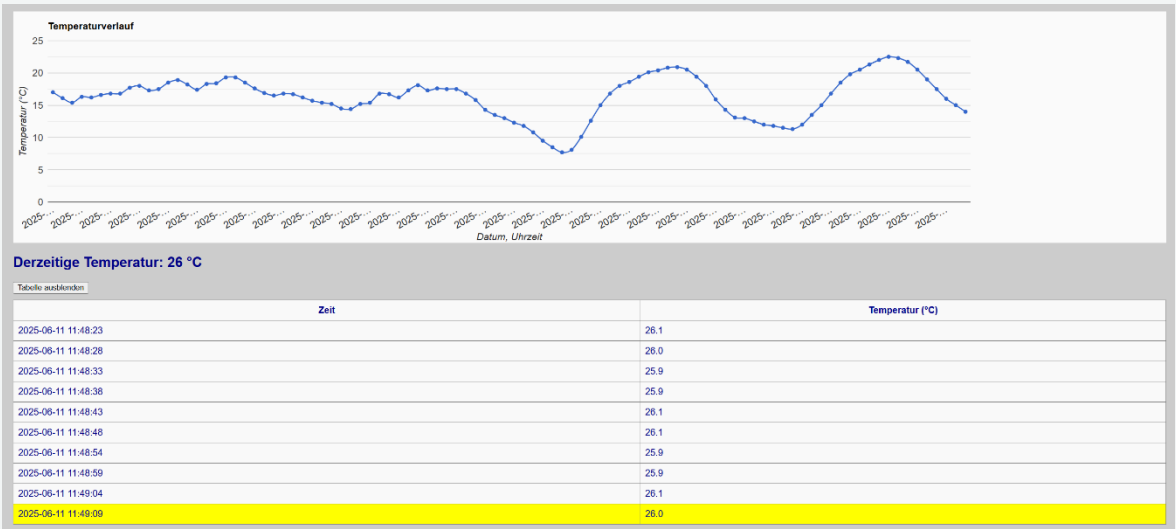
```
def handle_root(self, socket, args):
    """Root-Handler für Hauptseite"""


    page = """
    <!doctype html>
    <html lang="en">
      <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>ESP-FrontEnd</title>
        <script type="module" crossorigin src="https://ati567-ray.github.io/espfrontend/esp-frontend/dist/assets/index-98SST5nH.js"></script>
        {css}
      </head>
      <body>
          <div id="root"></div>
      </body>
    </html>
    """.format(css=self.get_css())


    server.ok(socket, "200", "text/html", page)
```

A.8 VISUALISIERUNG VON TEMPERATURDATEN

VISUALISIERUNG VON TEMPERATURDATEN 1



A.9 FUNKTIONSWEISE FRONTEND

FLUSSDIAGRAMM 3