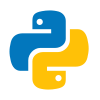
******

Ipek, Atilla & Roser, Kevin

02.06.2025| GEWERBLICH-TECHNISCHE SCHULE OFFENBURG, E1FI1 2024/2025, HUG CHRISTOPHER

Internet der Dinge

MIT WEMOS D1 MINI ESP8266

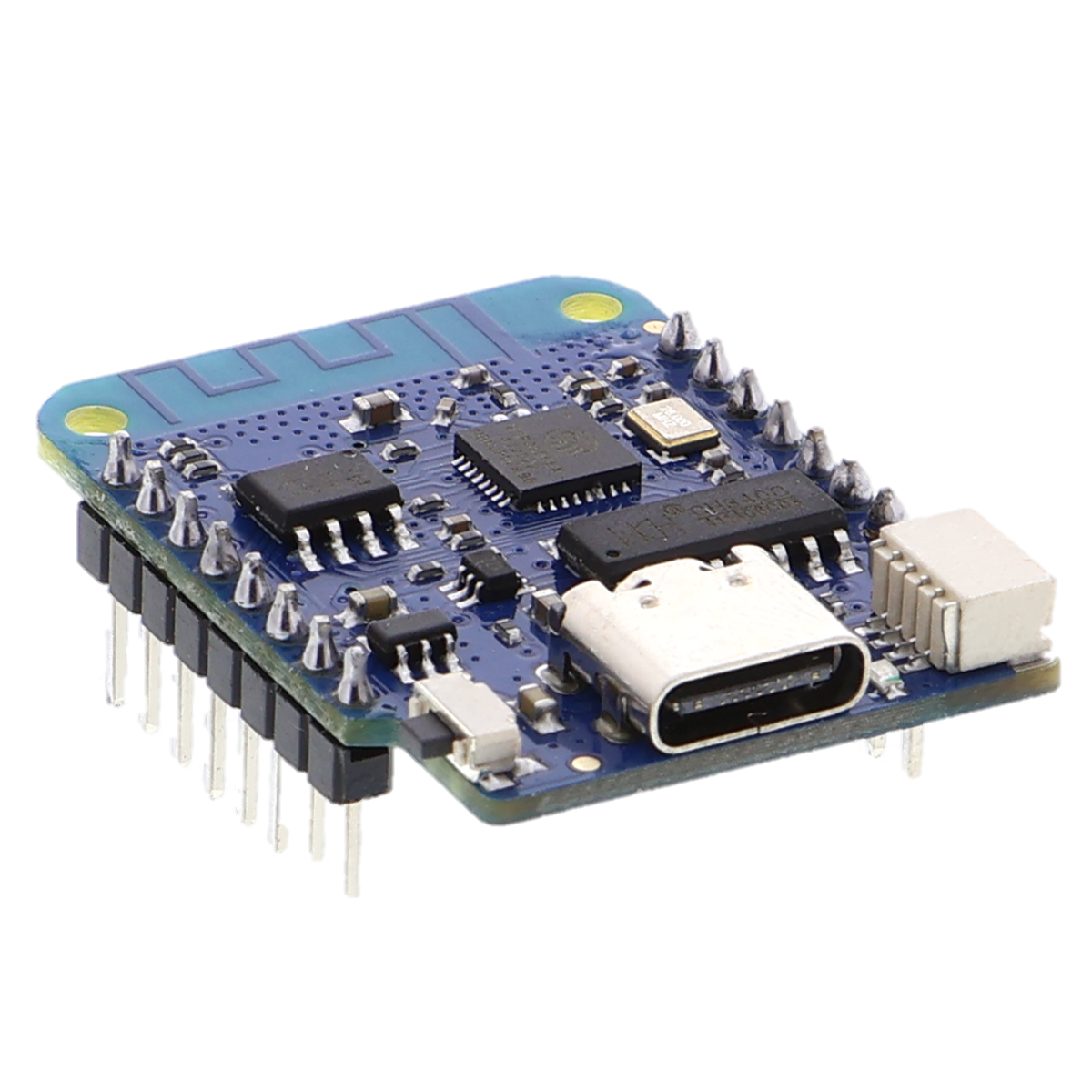
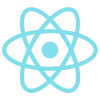
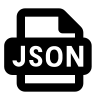
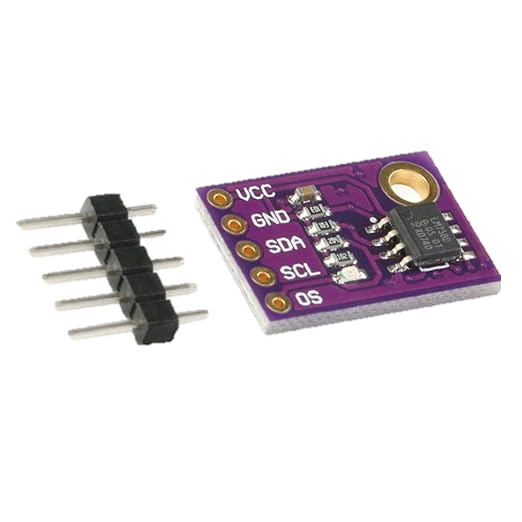
****

Abbildung 1

Abbildung 2

Inhaltsverzeichnis

[I. Abbildungsverzeichnis 0](#_Toc200466866)

[II. Abkürzungsverzeichnis 0](#_Toc200466867)

[III. Literaturverzeichnis 0](#_Toc200466868)

[1 Einleitung 1](#_Toc200466869)

[1.1 Motivation/Problemstellung 1](#_Toc200466870)

[1.2 Zielgruppe 1](#_Toc200466871)

[2 Projektbeschreibung 2](#_Toc200466872)

[2.1 Projektidee & Ziele 2](#_Toc200466873)

[2.2 Technische Spezifikationen 2](#_Toc200466874)

[2.3 Hardwarearchitektur 3](#_Toc200466875)

[2.4 Softwarearchitektur 3](#_Toc200466876)

[2.4.1 IDE und verwendete Libraries 3](#_Toc200466877)

[2.4.2 Backend 4](#_Toc200466878)

[2.4.3 Frontend 7](#_Toc200466879)

[2.5 Verwendete Web-Technologien 8](#_Toc200466880)

[2.5.1 HTML (HyperText Markup Language) 8](#_Toc200466881)

[2.5.2 JSON (JavaScript Object Notation) 8](#_Toc200466882)

[2.5.3 XML (eXtensible Markup Language) 8](#_Toc200466883)

[2.5.4 JavaScript-Framework React 9](#_Toc200466884)

[3 Schluss 9](#_Toc200466885)

[3.1 Zusammenfassung des Projekts 9](#_Toc200466886)

[3.2 Mögliche Erweiterungen 11](#_Toc200466887)

[4 Kommunikationsplan 12](#_Toc200466888)

[4.1.1 Teammitglieder 12](#_Toc200466889)

[4.1.2 Aufgabenverteilung 12](#_Toc200466890)

[4.1.3 Genutzte Kommunikationskanäle 13](#_Toc200466891)

[A. Anhang 14](#_Toc200466892)

[A.1 Flussdiagramm des Datamanagers 14](#_Toc200466893)

[A.2 Blockschaltbild des ganzen Systems 15](#_Toc200466894)

[A.3 Struktogramm von Temperatursensorauswertung 15](#_Toc200466896)

[A.4 Flussdiagramm von Main-Klasse 16](#_Toc200466897)

[A.5 Klassendiagramm des ganzen Systems 17](#_Toc200466898)

[A.6 Konfigurationsdatei *Config.py* 18](#_Toc200466899)

[A.7 Minimalistische HTML-Struktur des ESP 18](#_Toc200466900)

[A.8 Visualisierung von Temperaturdaten 18](#_Toc200466901)

[A.9 Funktionsweise Frontend 19](#_Toc200466902)

1. Abbildungsverzeichnis

[Abbildung 1 1](file:///C:\Users\it014\Desktop\ESP8266DokumentationAtillaIpek.docx#_Toc200464203)

<https://www.amazon.de/HiLetgo-Temperatursensor-Interface-Precision-Temperature/dp/B082M65D8P>

[Abbildung 2 1](file:///C:\Users\it014\Desktop\ESP8266DokumentationAtillaIpek.docx#_Toc200464204)

https://funduinoshop.com/elektronische-module/sonstige/mikrocontroller/wemos-d1-mini-v4.0-esp8266-mikrocontroller-lolin-d1-mini-v4.0-kompatibel

[Abbildung 3 11](file:///C:\Users\it014\Desktop\ESP8266DokumentationAtillaIpek.docx#_Toc200464205)

<https://www.amazon.de/DollaTek-LCUS-Relais-intelligente-Schaltersteuerung/dp/B07DJ549LX/>

1. 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](https://de.wikipedia.org/wiki/System).

**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)

1. Literaturverzeichnis

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

# Einleitung

## Motivation/Problemstellung

Warum ist IoT wichtig? Warum der Wemos D1 Mini? @Kevin

## Zielgruppe

Schüler:innen, Lehrer, mit der Erweiterung jeder der im Office arbeitet @Kevin

# Projektbeschreibung

## Projektidee & Ziele

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

* ***Kurzzeitansicht***: die letzten 50 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.

## 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

## 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]](#footnote-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]](#footnote-2)

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

## 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

### 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]](#footnote-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*“

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

### 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]](#footnote-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.

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

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

* API-Endpoints
* Response-Format (Beispiel Response):

*// JSON Response*

[

["2024-01-15 14:30:25", 23.5],

["2024-01-15 14:30:30", 23.7]

]

*// JSON Response*

[

["2024-01-15 14:30:25", 23.5],

["2024-01-15 14:30:30", 23.7]

]

*// JSON Response*

[

["2024-01-15 14:30:25", 23.5],

["2024-01-15 14:30:30", 23.7]

]

*// JSON Response*

[

["2024-01-15 14:30:25", 23.5],

["2024-01-15 14:30:30", 23.7]

]

*// JSON Response*

[

["2024-01-15 14:30:25", 23.5],

["2024-01-15 14:30:30", 23.7]

]

*// JSON Response*

[

["2024-01-15 14:30:25", 23.5],

["2024-01-15 14:30:30", 23.7]

]

*// JSON Response*

[

["2024-01-15 14:30:25", 23.5],

["2024-01-15 14:30:30", 23.7]

]

*// 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>

*// XML Response*

<temperaturen>

<eintrag>

<zeitpunkt>2024-01-15 14:30:25</zeitpunkt>

<wert>23.5</wert>

</eintrag>

</temperaturen>

*// XML Response*

<temperaturen>

<eintrag>

<zeitpunkt>2024-01-15 14:30:25</zeitpunkt>

<wert>23.5</wert>

</eintrag>

</temperaturen>

*// XML Response*

<temperaturen>

<eintrag>

<zeitpunkt>2024-01-15 14:30:25</zeitpunkt>

<wert>23.5</wert>

</eintrag>

</temperaturen>

*// XML Response*

<temperaturen>

<eintrag>

<zeitpunkt>2024-01-15 14:30:25</zeitpunkt>

<wert>23.5</wert>

</eintrag>

</temperaturen>

*// XML Response*

<temperaturen>

<eintrag>

<zeitpunkt>2024-01-15 14:30:25</zeitpunkt>

<wert>23.5</wert>

</eintrag>

</temperaturen>

*// XML Response*

<temperaturen>

<eintrag>

<zeitpunkt>2024-01-15 14:30:25</zeitpunkt>

<wert>23.5</wert>

</eintrag>

</temperaturen>

*// 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.

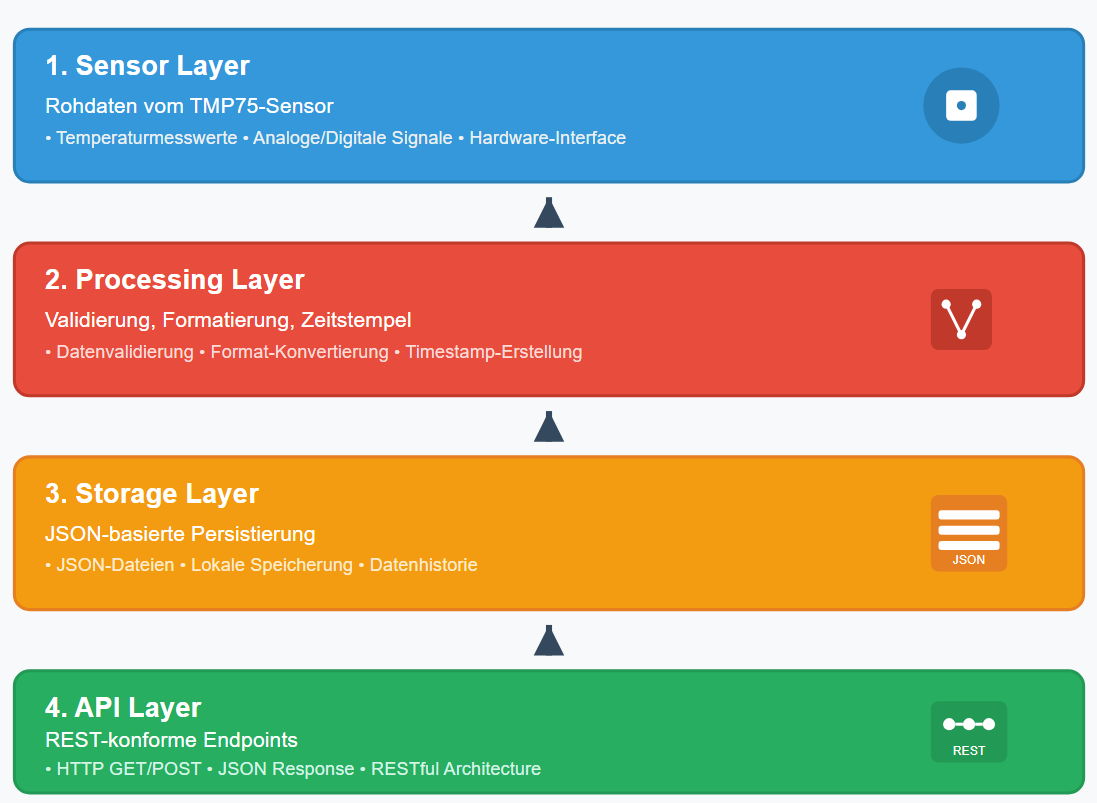
* Für die Verwaltung der Netzwerkverbindung mit Verbindungsüberwachung ist die WifiManager-Klasse (*WifiManager.py*) zuständig. Zusätzlich stellt sie sicher, dass bei einer erfolgreichen Verbindung die aktuelle IP-Adresse des ESP automatisch in einer *launchsettings.json*-Datei gespeichert wird. Dies ermöglicht eine automatische Kopplung von Backend und Frontend über das lokale Netzwerk.
* Mit der SoftwareWatchdog-Klasse (*SoftwareWatchdog.py*) wurde eine Überwachungsfunktion zur Resilienzimplementiert:
* 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.

Daten

### 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*

## Verwendete Web-Technologien

### HTML (HyperText Markup Language)

(Wird noch gemacht) @Kevin

### JSON (JavaScript Object Notation)

(Wird noch gemacht) @Kevin

### XML (eXtensible Markup Language)

(Wird noch gemacht) @Kevin

### 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. [[6]](#footnote-6)

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*.

# Schluss

## 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 50 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 somit auch nicht wirklich uns erklären woran er lag und 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.

## Mögliche Erweiterungen

* HTTP-Client in Python @Kevin
* Client-PC steuert über Temperatur-Abfrage (M) Ventilator. Beispielsweise im Sommer wenn es zu heiß wird, geht der Ventilator von alleine an am Arbeitsplatz

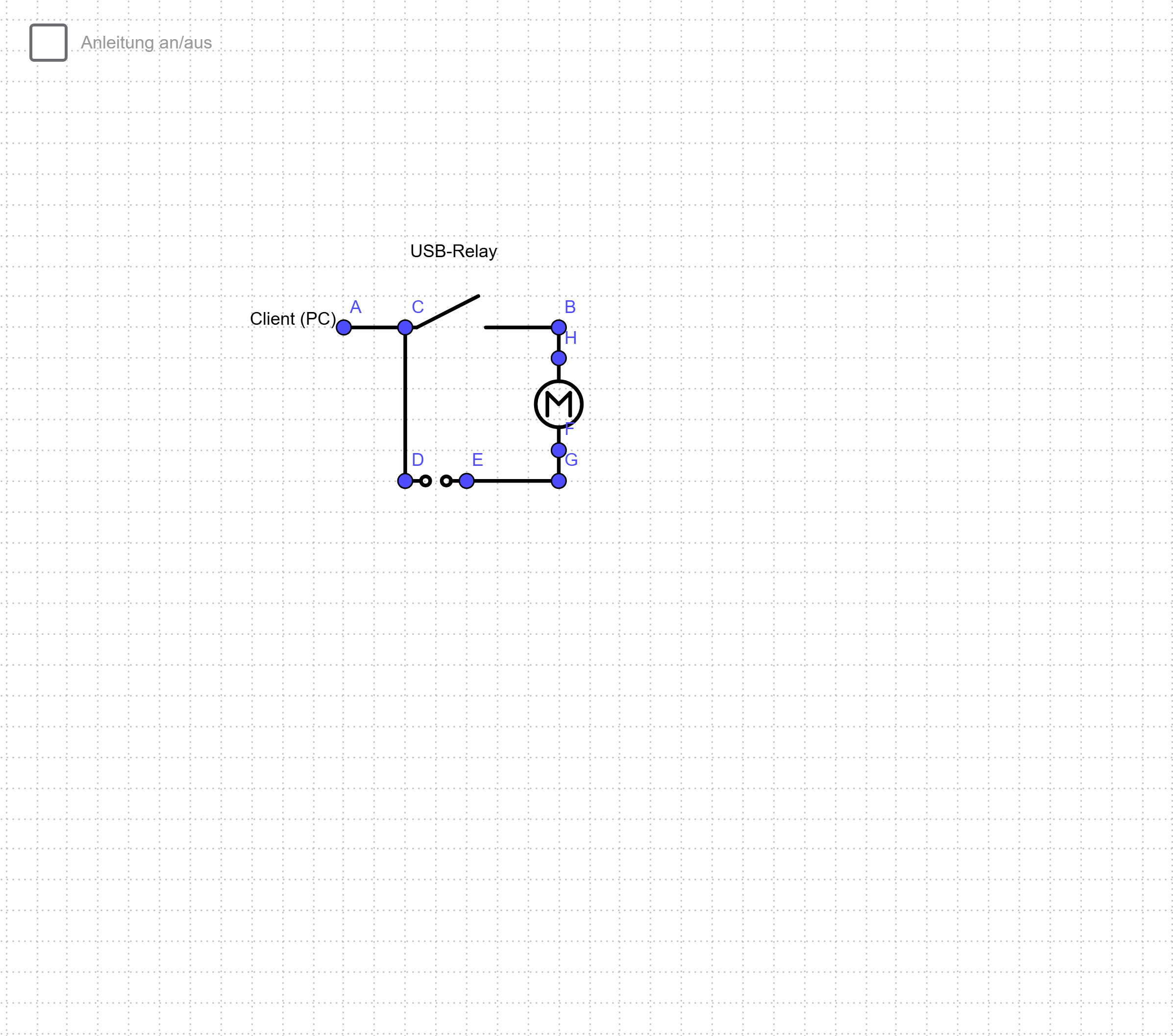
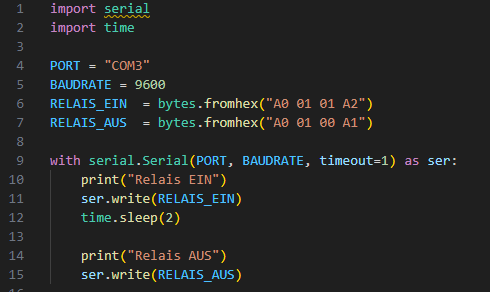


Abbildung 3

ex

über eine serielle Schnittstelle, in Python:



# Kommunikationsplan

### Teammitglieder

* **Atilla Ipek** – atilla.ipek94@gmail.com
* **Kevin Roser** - [E-Mail wird im Original ergänzt]

### 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

#### ****Gemeinsame Verantwortlichkeiten:****

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

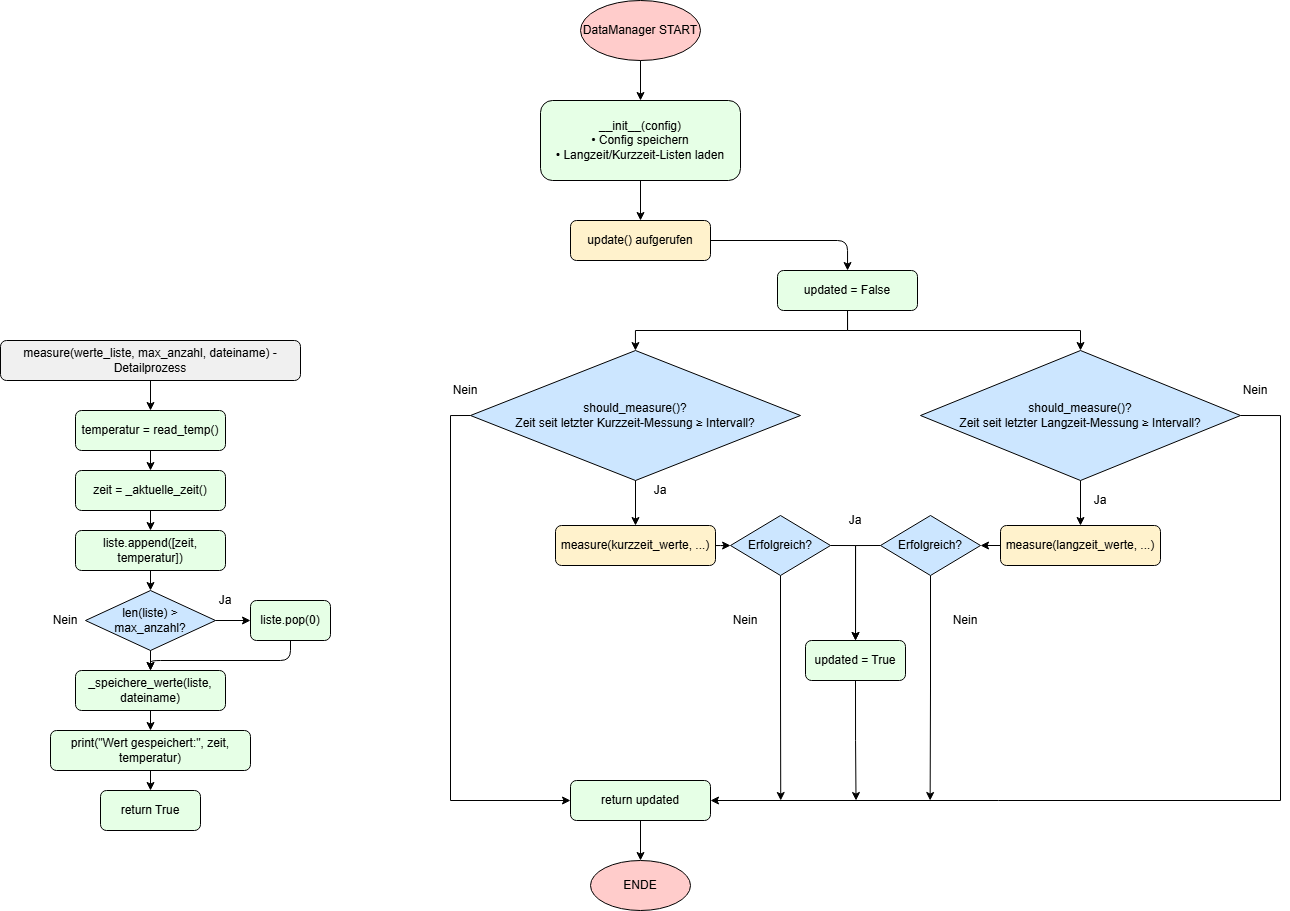
### Genutzte Kommunikationskanäle

**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

1. Anhang
   1. Flussdiagramm des Datamanagers

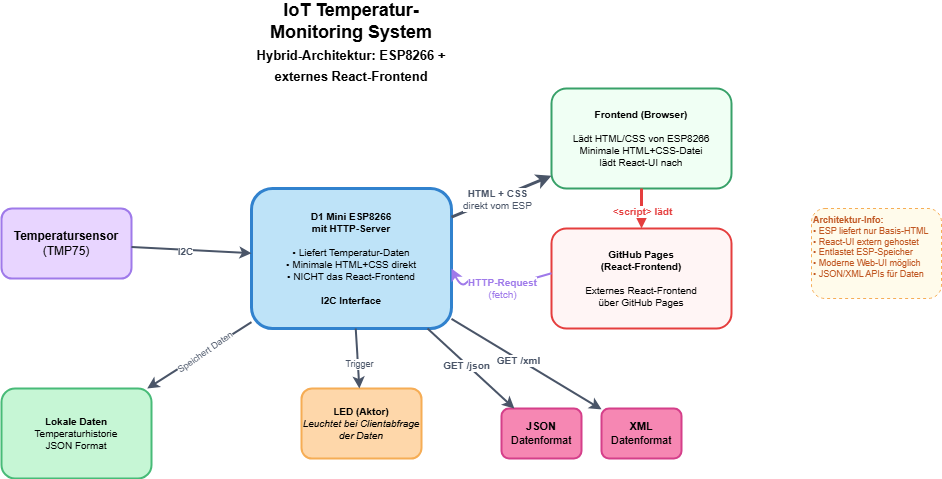


Speichervorgang 1

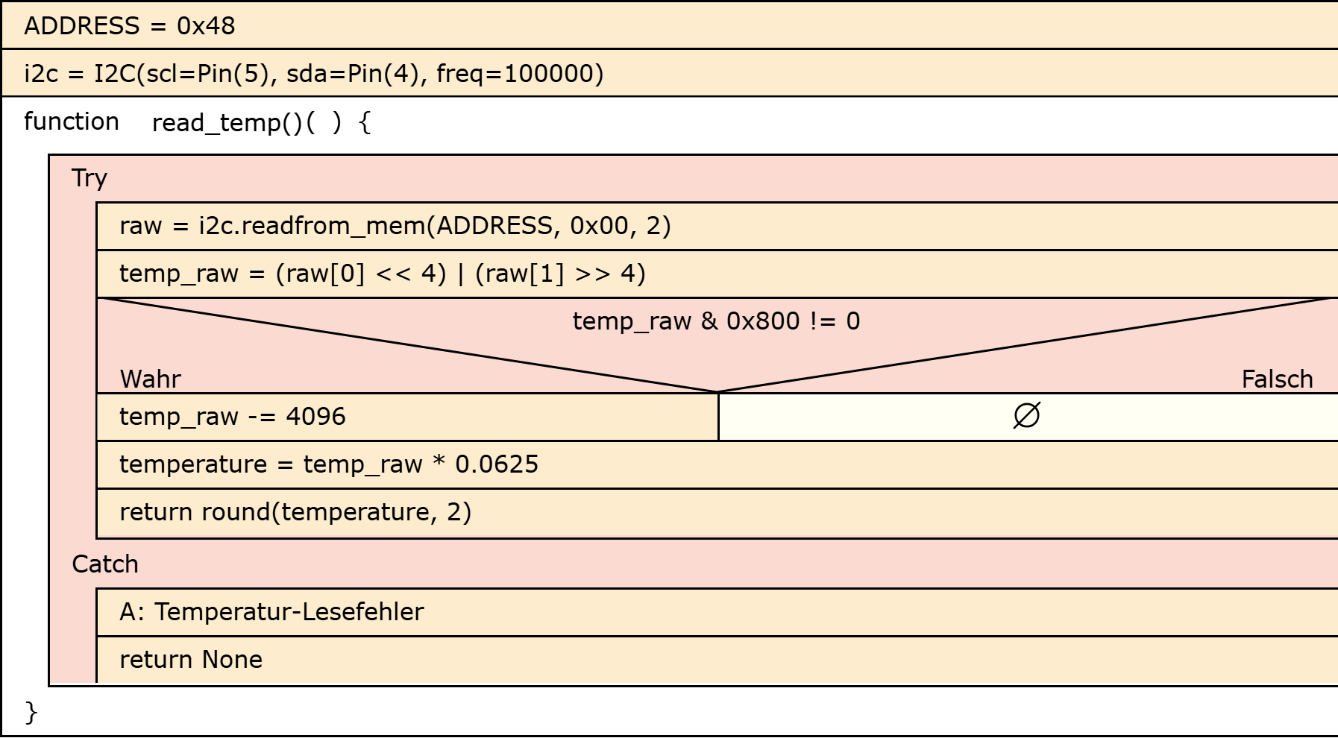
Flussdiagramm 1

* 1. Blockschaltbild des ganzen Systems

Blockschaltbild 1



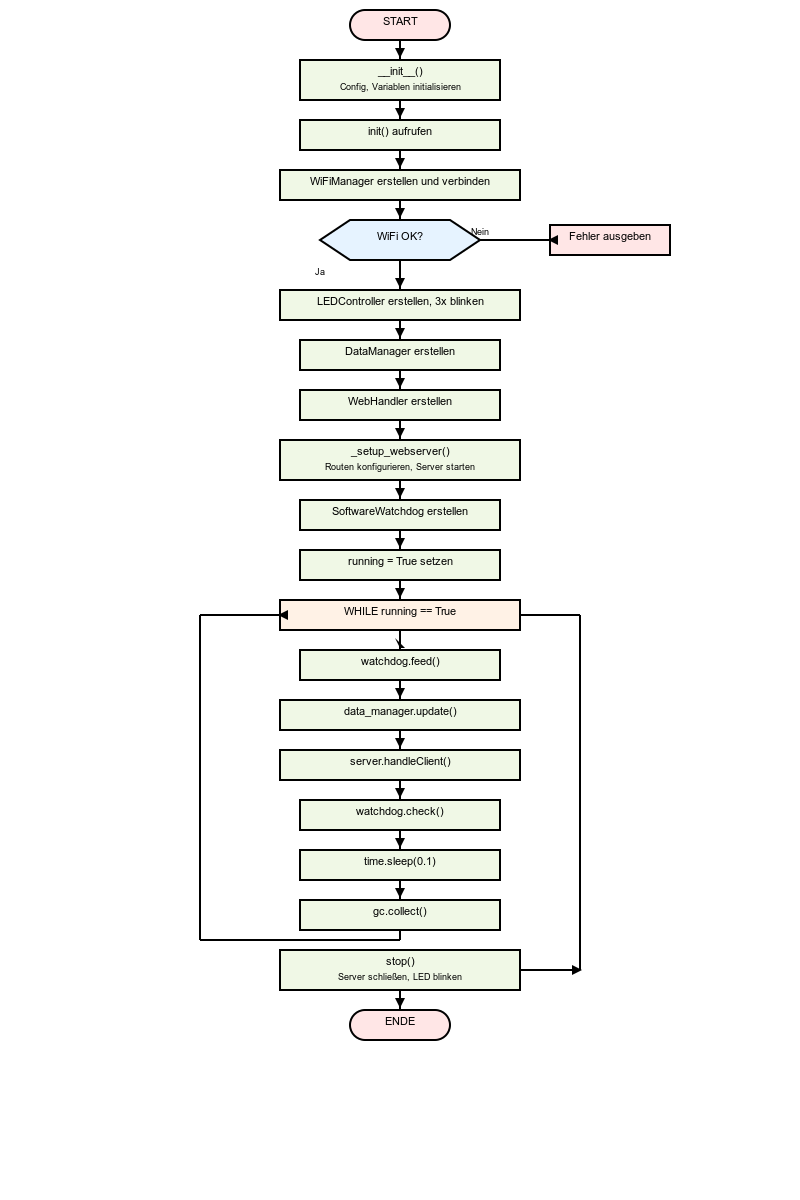
* 1. Struktogramm von Temperatursensorauswertung



Struktogramm 1

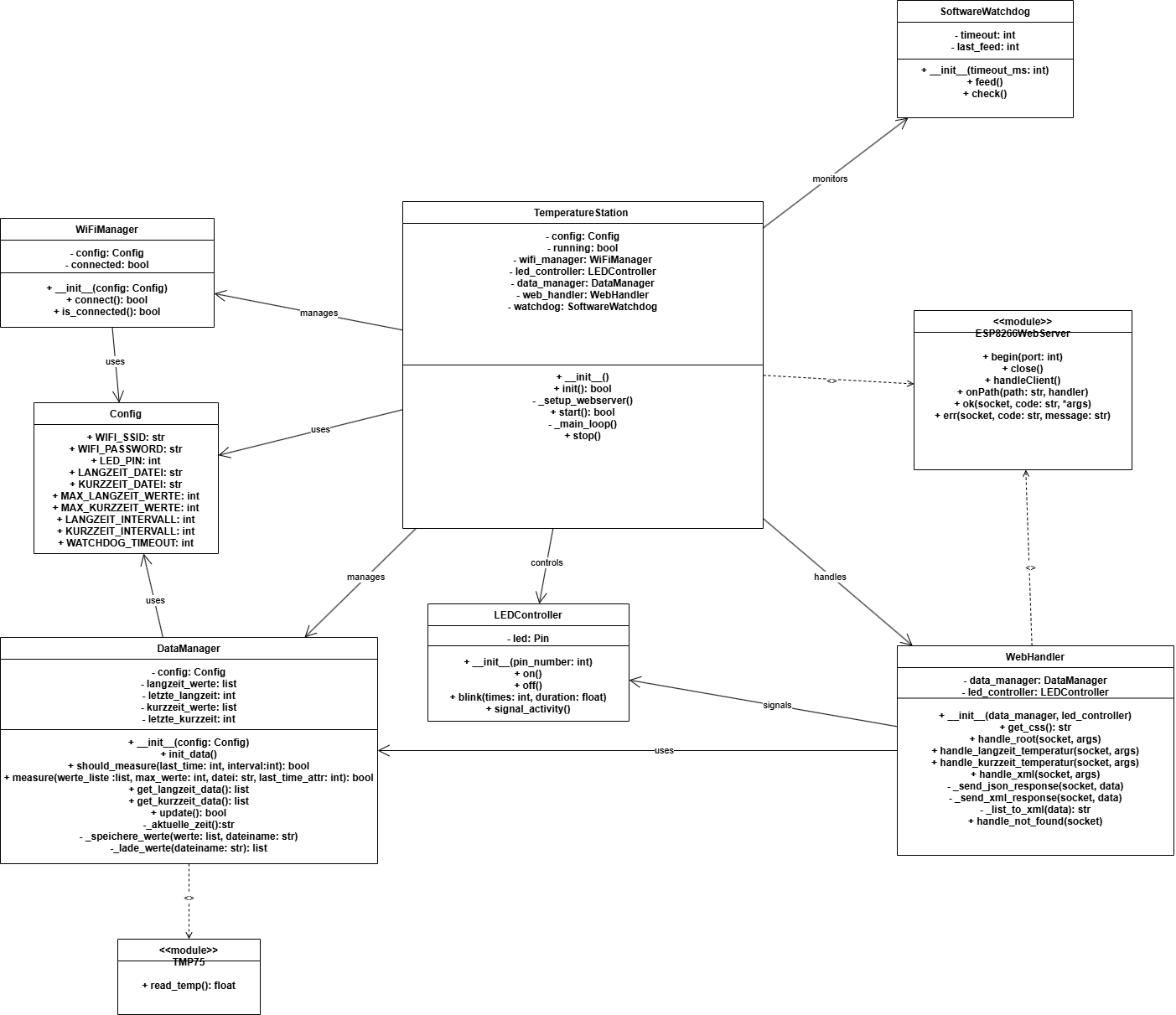
* 1. Flussdiagramm von Main-Klasse

Flussdiagramm 2



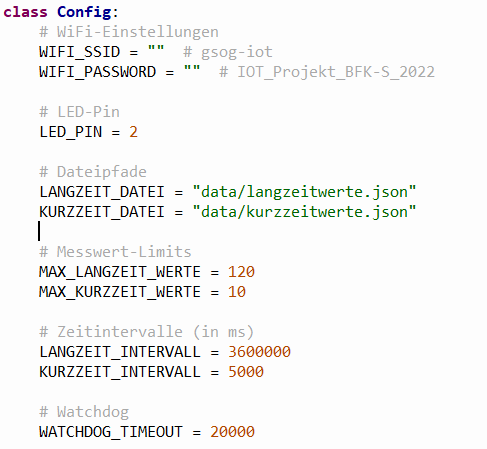
* 1. Klassendiagramm des ganzen Systems

Klassendiagramm 1



* 1. Konfigurationsdatei *Config.py*

Konfiguration 1

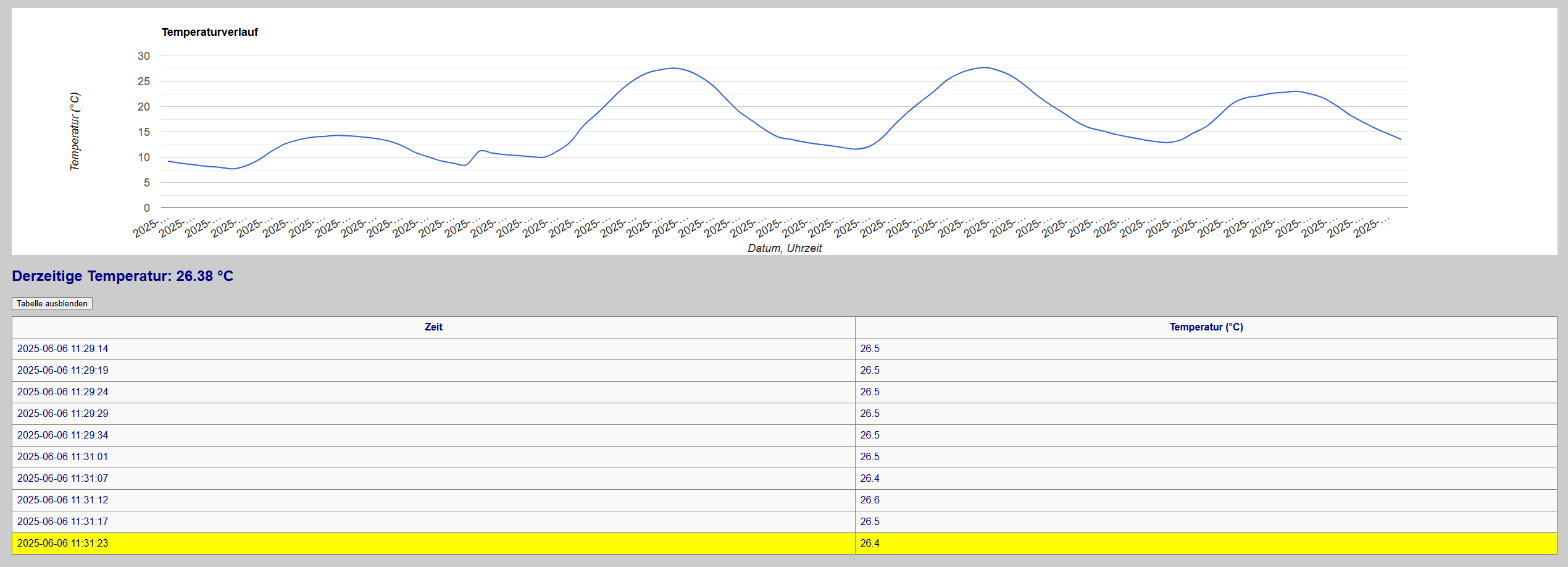


* 1. Minimalistische HTML-Struktur des ESP

minimalistische HTML-Struktur 1

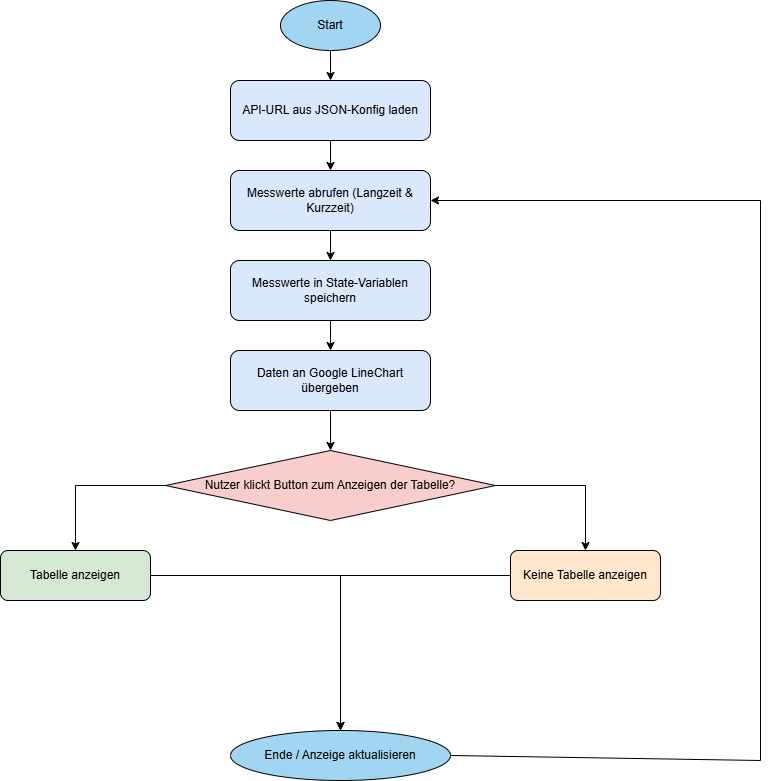


* 1. Visualisierung von Temperaturdaten



Visualisierung von Temperaturdaten 1

* 1. Funktionsweise Frontend



Flussdiagramm 3

1. (Wemos D1 Mini) [↑](#footnote-ref-1)
2. (CJMCU LM75 Temperatursensor) [↑](#footnote-ref-2)
3. (Thonny) [↑](#footnote-ref-3)
4. (React Google Charts) [↑](#footnote-ref-4)
5. (Wikipedia) [↑](#footnote-ref-5)
6. (React-js) [↑](#footnote-ref-6)