Schriftliche Ausarbeitung

Dokumentation Anwendung „Wettersensoren“
im Rahmen des Moduls „AWE2“

Prüfer:
Florian Wortmann

Erstellt von:
Jonathan Brockhausen, Philipp Röring, Julius Figge

Studiengang:
Angewandte Informatik B.Sc.

Eingereicht am:
18. November 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

1 Installation

ESP8266

Zur Erfassung der Wetterdaten wird ein BME280 Sensor verwendet, welcher von einem Board mit ESP8266 Mikrocontroller und NodeMCU 1.0 Firmware angesteuert wird. Auf diesem Mikrocontroller kann der Sourcecode *nodemcu.ino* ausgeführt werden. Zum Kompilieren sollte die Arduino IDE verwendet werden, in der zuvor die Treiber für den ESP8266 installiert werden müssen. Dazu muss in den Voreinstellungen folgende Boardverwalter-URL hinzugefügt werden: *Boardverwalter-URL*. Darüber hinaus müssen folgende Bibliotheken über die integrierte Bibliotheksverwaltung installiert werden:

Tabelle 1: Verwendete Arduino-Bibliotheken

Bibliothek	**Version**
WiFi (by Arduino)	1.2.7
Adafruit BME280 Library (by Adafruit)	2.1.1
Adafruit Unified Sensor (by Adafruit)	1.1.4
EasyNTPClient (by Harsha Alva)	1.1.0
LinkedList (by Ivan Seidel)	1.2.3

Quelle: Eigene Darstellung

Bevor der Quellcode kompiliert wird, müssen die folgenden Konstanten auf die lokalen Gegebenheiten angepasst werden:

- *SERVER_TO_CONNECT*
- *SSID*
- *WIFI_PASSWORD*

Der BME280 Sensor und der ESP8266 müssen folgendermaßen verbunden werden:

Tabelle 2: Pin Layout für Verbindung ESP mit BME

ESP8266 Pin	**BME280**
3.3V	VIN
G	GND
D1	SCL
D2	SDA

Quelle: Eigene Darstellung

Die exakten Geräte sind:

- AZDelivery NodeMCU Lua Lolin V3 Module ESP8266 ESP-12F WIFI

- AZDelivery GY-BME280

Backend und Frontend

Backend sowie Frontend werden mithilfe von Docker deployed. Im Frontend ist hierzu die Backend Url in der Klasse *Constants.js* die Variable *SERVER_URI* anzupassen. Die Images dafür lassen sich in den jeweiligen Modulen mit Hilfe der *buildImageandTar.sh* Skripts bauen. Diese bauen die Images und stellen diese in der lokalen Dockerumgebung zum Start bereit. Darüber hinaus werden im Projekt-Root-Ordner tar-Bälle mit den jeweiligen Images hinterlegt. Für diesen Schritt haben wir uns entscheiden um die Images einfach auf einem Server verfügbar zu machen ohne Docker Registries (z.B. Docker.io) in Anspruch nehmen zu müssen.

Der Standard Admin-Zugang ist *admin:$PASSWORD* Das Passwort für den Admin Zugang - in Form der Variable *$PASSWORD* in *router.js* - ist gegebenermaßen zu ersetzen.

Deployment Voraussetzungen unter Windows

Das Projekt kann mithilfe der WSL 2 und Docker for Windows deployed werden. Für die Vorbereitung muss zunächst eine WSL 2 eingerichtet werden (Link zur Anleitung: hier). Danach kann Docker for Windows mit den WSL 2-Komponenten installiert werden (Link zur Anleitung: hier). Nachdem Docker for Windows bereitgestellt wurde kann die ausgewählte Linux Distribution in der WSL gestartet werden. Danach sind in der WSL die Schritte für Linux auszuführen.

Deployment Voraussetzungen unter Linux

In Linux sind Docker sowie Node und npm durch den Distribution-spezifischen Package Manager zu installieren.

Backend Start-Command:

docker run -p 3000:3000 -v $PATH_TO_DATABASE:/usr/src/app/db –name awe2-backend -it awe2/backend:abgabe

$PATH_TO_DATABASE ist zu ersetzen mit dem Ordner, in welchem die Datenbank auf dem Host-System gespeichert werden soll.

Frontend Start-Command:

docker run -p 3344:3344 –name awe2-frontend -it awe2/frontend:abgabe

2 Fachkonzept

In diesem Kapitel werden die im Projekt eingesetzten Technologien beschrieben.

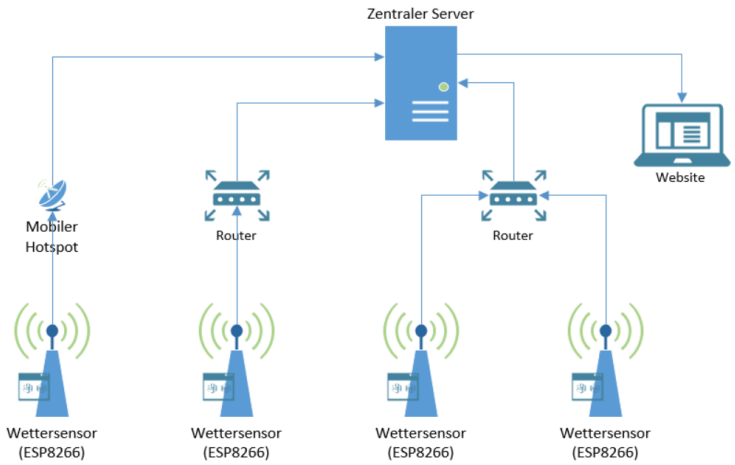

Abbildung 1: Komponenten des Gesamtsystems (eigene Darstellung)

In Abbildung 1 ist ein möglicher Aufbau der Projektkomponenten gemäß den Projektanforderungen[1] dargestellt.

2.1 Mikrocontroller

Wie in Abschnitt 1 beschrieben, kommt im Projekt ein ESP8266 Mikrocontroller zum Einsatz. Auf dem Mikrocontroller erfolgt die Erfassung der Messdaten mit einem BME280-Sensor und der Versand an das Backend. Bei der Auswahl der Libraries wurde hierbei beachtet, möglichst auf den Anwendungsfall spezifische Libraries zu wählen um die Auslastung des Mikrocontrollers zu minimieren. Ein geringer Energieverbrauch wird in der Programmierung besonders beachtet. Dadurch ist die Funktionalität auf das wesentliche beschränkt während trotzdem eine zuverlässige Funktionsweise gewährleistet wird. Wenn der Mikrocontroller keine Verbindung zum angegebenen WLAN-Netzwerk herstellen kann oder der Server nicht erreichbar ist, werden die Daten gecached und versandt, sobald eine Verbindung hergestellt werden kann. Der Mikrocontroller wird in der Arduino IDE in C++ programmiert.

[1] Wortmann, Florian (2020)

2.2 Zentraler Server

Auf dem zentralen Server wird das Backend des Projekts bereitgestellt. Das Backend ist in JavaScript geschrieben und verwendet eine SQLite-Datenbank für die Speicherung der Daten. Für die Kommunikation zu den Sensoren und dem Frontend ist eine REST-API vorhanden, die in Abschnitt 3 näher erläutert wird.
Das Projekt nutzt node.js Version 14.4 und SQLite-Version 3.
Sowohl das Backend als auch das Frontend sind für das Deployment in Docker vorgesehen. Hierbei ist eine örtliche Trennung möglich, das Frontend kann auf jeden Server zugreifen, der die API dieses Projekts bereitstellt.

2.3 Website

Die Website basiert ebenfalls auf JavaScript für die Logik. Aufgrund des Projektaufbaus ist es wie oben genannt möglich, das Hosting von Backend und Frontend zu trennen.
Für die Bereitstellung des Frontends kommen einige Node-Libraries zum Einsatz. Die Darstellung der Messdaten auf der Website erfolgt mit Chart.js[2]. Die Auswahl eines Zeitraums für die Darstellung eines Intervalls ist mit der Library daterangepicker[3] umgesetzt. Die HTML- und CSS-Komponenten der Website sind mit Bootstrap 4 erstellt worden. Auf die Gestaltung wird in **??** näher eingegangen.

[2] *ChartJS* (2020)
[3] *Date Range Picker* (2020)

3 Schnittstellen

Das Backend unserer Anwendung verfügt über eine REST-API. Diese wird zur Kommunikation des Backends mit den jeweiligen Wettersensoren, sowie zur Kommunikation des Frontends mit dem Backend verwendet.

Schnittstellenbeschreibung REST-API

Die Schnittstelle stellt folgende Services bereit:

- *HTTP-Methode*: GET

  Relativer Pfad: **/weatherData**

  Antwort: JSON-Object mit zwei Arrays. Das erste beinhaltet alle Sensoren, das zweite alle verfügbaren Sensordaten.[4]

  Beispielantwort: siehe Anhang 1

- *HTTP-Methode*: GET

  Relativer Pfad: **/sensorData/id/:SENSOR_ID**

  Antwort: JSON-Object mit Wetterdaten für gewählten Sensor

  Parameter:

  – URL-Encoded: *ID* - ID des gewünschten Sensor

  – (optional) Query: *timerange_start* - Mindestzeitstempel der Sensordaten.

  – (optional) Query: *timerange_end* - Höchstzeitstempel der Sensordaten.

  – (optional) Query: *granularity* - Menge der zurückzugebenden Datenpunkte

  Beispielantwort: siehe Anhang 1

- *HTTP-Methode*: GET

  Relativer Pfad: **/sensors**

  Antwort: JSON-Object welches alle Sensoren beinhaltet.

  Beispielantwort: siehe Anhang 1

[4] Für beinhaltete Datentypen siehe **??**

- *HTTP-Methode*: GET

  Relativer Pfad: **/sensor/id/:SENSOR_ID**

  Antwort: JSON-Object welches die Informationen über einen ausgewählten Sensor beinhaltet.

  Parameter:

  – URL-Encoded: *ID* - ID des gewünschten Sensor

  Beispielantwort: siehe Anhang 1

- *HTTP-Methode*: POST

  Relativer Pfad: **/weatherData/**

  Inhalt: JSON-Object welches Sensordaten beinhaltet.[5]

  Antwort bei Erfolg: HTTP-Status 200, String der erfolgreiche Speicherung bestätigt

  Antwort bei Fehlern:

  – *Duplizierter Inhalt*: HTTP-Status 400, Fehlermeldungen und Errors

  – *Fehler beim Parsen des Bodies*: HTTP-Status 400, Fehlermeldungen und Errors

- *HTTP-Methode*: POST

  Relativer Pfad: **/updateSensorLocation/**

  Inhalt: JSON-Object welches aktualisierten Ort für über ID identifizierten Sensor beinhaltet.

  Absicherung: Dieser Endpoint kann nur durch mitsenden eines API-Tokens genutzt werden.[6]

  Antwort bei Erfolg: HTTP-Status 200, String der erfolgreiche Speicherung bestätigt

  Antwort bei Fehlern:

  – *Fehler beim Parsen des Bodies*: HTTP-Status 400, Fehlermeldungen und Errors

[5] Für beinhaltete Daten siehe **??**
[6] Dieser steht nur dem Admin zur Verfügung und ist genauer unter Abschnitt 4 spezifiziert.

4 Security

Im folgenden werden getroffene (Design)-Entscheidungen in Bezug auf die Sicherheit unserer Anwendung erläutert. Dabei werden besonders relevante Stellen hervorgehoben und erklärt. Die Erläuterung ist aufgeteilt in die Bestandteile unserer Anwendung.

ESP8266

Die *Wlan Zugangsdaten* werden beim flashen unserer Anwendung auf den ESP übertragen und sind ab diesem Punkt fest / „hardcoded“. Diese Entscheidung wurde getroffen unter der Abwegung, dass die ESPs, und damit die Wettersensoren, einen festen Standort besitzen.[7] Dieser wird wie bereits erläutert im Admininterface der jeweiligen MAC-Adresse des Gerätes zugeordnet. Hierdurch bedingt war eine mögliche dynamische Festlegung der Zugangsdaten für uns nicht verhältnismäßig. Insbesondere da ohne WLAN-Verbindung keine Kommunikation mit dem Gerät erfolgen kann.
Der zweite relevante Punkt ist die *Absicherung von REST-Calls*. Wir haben uns im Hinblick auf die geringe Leistung der ESPs gegen eine weitere Absicherung der REST-Calls entschieden. Um aufgrund der Corona-Situation eine Zusammenarbeit zu ermöglichen wurde die Anwendung öffentlich erreichbar gemacht. Grundsätzlich ist dies aber kein notwendigerweise geplanter Einsatzzweck. Diese Entscheidung wird im folgenden Absatz weiter erläutert.

Backend

Durch die Ausrichtung auf eine interne und nicht öffentliche Nutzung (beispielhaft in einem internen Unternehmensnetzwerk) haben wir entschieden, dass eine grundlegende Absicherung des *Sensordaten Endpoints* ausreichend ist. Diese besteht bei uns konkret in der Prüfung der empfangenen Sensordaten (Siehe Abschnitt 3), hier werden die Werte auf Korrektheit im Bezug auf Datentypen, sowie insbesondere sinnvolle Werte (z.B. Zeitstempel nicht vor Veröffentlichungsdatum oder Temperaturen außerhalb von festgelegten Grenzwerten) geprüft. Des weiteren werden die Daten vor dem einfügen in die Datenbank von uns parametrisiert um SQL-Injections zu verhindern.
Die Nutzung des *Sensorstandort Endpoint* ist dem Administrator vorbehalten. Deswegen ist zur erfolgreichen Änderung des Standorts ein „API-Token“ notwendig. Dieser wird bei Änderungen des Standorts im Frontend mitgesendet. Diese Absicherung ist grundlegend da nur Administratoren diesen „API-Token“ durch einen erfolgreichen Login erlangen können. Nach Abwegung des möglichen Schaden gegenüber den Kosten der Implementierung haben wir von einer weiteren Absicherung abgesehen.

[7] Darüber hinaus ist anzumerken, dass sowieso die letzte WLAN-Verbindung im ESP gehalten wird, auch wenn neuer Code geflasht wird. Somit dürfen die Geräte nicht durch Unbefugte benutzt werden.

Darüber hinaus nutzt unsere Anwendung *Logging*. Eingehende Anfragen und insbesondere Fehler werden durch das Node-Modul `rotating-file-stream` geloggt. Dieses ist so konfiguriert, dass Events mit Zeitstempel und Dringlichkeit (beispielhaft „INFO“ oder „ERROR“)in die Konsole geloggt werden. Beim auftreten von Errors werden die fehlerhaften Requests in Gänze geloggt, da dies notwendig ist um die Quelle dieser nachzuvollziehen. Darüber hinaus werden alle Log-Nachrichten in einer Datei gespeichert. Diese ist auf 10 Megabyte Größe begrenzt und wird täglich gewechselt. Alte Dateien werden komprimiert gespeichert. Auszüge hiervon finden sich im Anhang 2.

Frontend

Das Frontend-Modul verwendet die Selbe *Logging Konfiguration* wie das Backend. Auszüge hiervon finden sich im Anhang 2.
Erwähnenswert ist hier insbesondere das *Admin-Interface*. Der Zugang hierzu kann erst nach erfolgreichem Login mit Hilfe von Basic-Authentication erfolgen. Da im Admin-Interface nur die Funktionalität der Festlegung des Sensorstandortes verfügbar ist, ist diese Lösung die effektivste und sinnvollste zur Gewährleistung der Sicherheit.

Github

Im Zuge der Versionsverwaltung unseres Projektes werden alle Module, wie von uns explizit konfiguriert, *automatisiert auf Sicherheitslücken* überprüft. Sollte durch Github eine Sicherheitslücke in Libraries oder Code gefunden werden, so werden wir als Entwickler benachrichtigt. Mit dieser Maßnahme lassen sich potentielle Sicherheitslücken schnell und automatisiert finden und einfach beheben.

Deployment

Das Deployment unserer Anwendung ist mit Hilfe von Docker realisiert. Wie während der Entwicklung umgesetzt, ist ein etwaiges Live-Deployment im besten Fall hinter einem *zusätzlichen Reverse-Proxy-Server* zu realisieren[8]. Dadurch ergibt sich eine zusätzliche Instanz welche zum Schutz der Anwendung beiträgt. Mit Hilfe des Reverse-Proxy-Servers können Anfragen an Frontend sowie Backend überwacht und gefiltert werden. Darüber hinaus lassen sich IP-Adressen böswilliger Anfragen sperren. Dies lässt sich mit einem System wie *Fail2Ban* (2020) realisieren.[9]
Ebenso sollte auf dem Deployment-Server im Zuge der Dockerverwaltung ein *automatisches Update der Container* konfiguriert werden, wie während der Entwicklung geschehen. Hierbei werden mit Hilfe der Dockerfiles die Images regelmäßig neu gebaut, um Abhängigkeiten auf

[8] Hierzu wurde während der Entwicklung ein *linuxserver/docker-swag* (2020) Docker Container genutzt, alternativ kann *traefik* (2020) als Docker Container genutzt werden
[9] Dieses ist in *linuxserver/docker-swag* (2020) bereits inkludiert

dem neuesten Stand zu halten. Anschliessend werden die verwendeten Images der Container ausgetauscht. Zum automatisierten Bau der Images, sollte ein „Cron-Job“ genutzt werden. Um die Nutzung der neuesten Images zu gewährleisten bietet sich *watchtower* (2020), ebenfalls als Docker Container, an.

Anhang

Anhangsverzeichnis

Anhang 1 Schnittstellen

Anhang 1.1 Antwort GET-Request auf „/weatherData“

Listing 1: Antwort GET-Request /weatherData


```
{"sensors":[
  {"ID":1,
    "MAC_ADDRESS":"e0:98:06:86:23:bc",
    "LOCATION":"Julius (Paderborn)",
    "LAST_UPDATE":1605628770000
  },
  {"ID":2,
    "MAC_ADDRESS":"68:c6:3a:88:c0:cd",
    "LOCATION":"Jonathan (nun auch nach alledem in Paderborn)",
    "LAST_UPDATE":1605357215000
  },
  {"ID":3,
    "MAC_ADDRESS":"f4:cf:a2:d1:49:3e",
    "LOCATION":"Philipp (Paderborn)",
    "LAST_UPDATE":1605628927000
  }
  ],
"sensorData":[
  {"SENSOR_ID":1,
    "TIMESTAMP":1604251064000,
    "TEMPERATURE":21.610001,
    "AIRPRESSURE":996.586426,
    "HUMIDITY":60.304688
  },
  {"SENSOR_ID":1,
    "TIMESTAMP":1604251364000,
    "TEMPERATURE":21.610001,
    "AIRPRESSURE":996.608887,
    "HUMIDITY":60.868164
  }
  ]
}
```

Anhang 1.2 Antwort GET-Request auf „/sensorData/id/1“

Listing 2: Antwort GET-Request /sensorData/id/1


```
{"sensorData":[
  {"SENSOR_ID":1,
    "TIMESTAMP":1604251064000,
    "TEMPERATURE":21.610001,
    "AIRPRESSURE":996.586426,
    "HUMIDITY":60.304688
  },
  {"SENSOR_ID":1,
    "TIMESTAMP":1604251364000,
    "TEMPERATURE":21.610001,
    "AIRPRESSURE":996.608887,
    "HUMIDITY":60.868164
  },
  {"SENSOR_ID":1,
    "TIMESTAMP":1604251666000,
    "TEMPERATURE":21.610001,
    "AIRPRESSURE":996.680603,
    "HUMIDITY":60.949219
  }
  ]
}
```

Anhang 1.3 Antwort GET-Request auf „/sensors“

Listing 3: Antwort GET-Request /sensors


```
{"sensors":[
  {"ID":1,
    "MAC_ADDRESS":"e0:98:06:86:23:bc",
    "LOCATION":"Julius (Paderborn)",
    "LAST_UPDATE":1605629371000
  },
  {"ID":2,
    "MAC_ADDRESS":"68:c6:3a:88:c0:cd",
    "LOCATION":"Jonathan (nun auch nach alledem in Paderborn)",
    "LAST_UPDATE":1605357215000
  },
  {"ID":3,
    "MAC_ADDRESS":"f4:cf:a2:d1:49:3e",
    "LOCATION":"Philipp (Paderborn)",
    "LAST_UPDATE":1605629652000
  }
]
}
```


Anhang 1.4 Antwort GET-Request auf „/sensor/id/1“

Listing 4: Antwort GET-Request /sensor/id/1


```
{
  "sensor": {
    "ID": 1,
    "MAC_ADDRESS": "e0:98:06:86:23:bc",
    "LOCATION": "Julius (Paderborn)"
  }
}
```

Anhang 2 Logdaten

Anmerkung: Aus Datenschutzgründen sind alle IP-Adressen im Folgenden zensiert.

Anhang 2.1 Beispielhafter Logauszug Backend

Listing 5: Beispielhafter Logauszug Backend

```
18.11.2020, 09:41:09 - ERROR :
  POST REQUEST PARSING BODY FAILED FROM [$ZENSIERTE_IP_ADRESSE], REQUEST BODY: {
    "MACADDRESS":"68:c6:3a:88:c0:cd",
    "TIMESTAMP":"1605692160",
    "TEMPERATURE":-143.25,
    "AIRPRESSURE":1185.736206,
    "HUMIDITY":100
  }

INSERT INTO SENSOR (MAC_ADDRESS, LOCATION)
  VALUES (?, "") EXCEPT
    SELECT MAC_ADDRESS, LOCATION FROM SENSOR WHERE MAC_ADDRESS = ?

18.11.2020, 09:41:25 - INFO :
  GOT REQUEST TO [/weatherData] FROM [$ZENSIERTE_IP_ADRESSE]

18.11.2020, 09:41:42 - INFO :
  GOT REQUEST TO [/sensorData/id/1?granularity=100] FROM [$ZENSIERTE_IP_ADRESSE]

18.11.2020, 09:41:43 - INFO :
  GOT REQUEST TO [/sensors/] FROM [$ZENSIERTE_IP_ADRESSE]

18.11.2020, 09:41:44 - INFO :
  GOT REQUEST TO [/sensor/id/1] FROM [$ZENSIERTE_IP_ADRESSE]
```

Anhang 2.2 Beispielhafter Logauszug Frontend

Listing 6: Beispielhafter Logauszug Frontend

```
16.11.2020, 11:00:34 - INFO :
  FRONTEND STARTED

16.11.2020, 11:08:58 - INFO :
  GOT REQUEST TO [/] FROM [$ZENSIERTE_IP_ADRESSE]

16.11.2020, 11:09:54 - INFO :
  GOT REQUEST TO [/] FROM [$ZENSIERTE_IP_ADRESSE]

16.11.2020, 11:09:54 - INFO :
  GOT REQUEST TO [/favicon.ico] FROM [$ZENSIERTE_IP_ADRESSE]

16.11.2020, 12:50:18 - INFO :
  GOT REQUEST TO [/robots.txt] FROM [$ZENSIERTE_IP_ADRESSE]
```

Quellenverzeichnis

Literatur

ChartJS (2020). URL: https://www.chartjs.org/.

Date Range Picker (2020). URL: https://www.daterangepicker.com/.

Fail2Ban (2020). URL: https://www.fail2ban.org/wiki/index.php/Main_Page.

linuxserver/docker-swag (2020). URL: https://github.com/linuxserver/docker-swag.

traefik (2020). URL: https://github.com/traefik/traefik.

watchtower (2020). URL: https://github.com/containrrr/watchtower.

Wortmann, Florian (2020). *Anwendungsentwicklung II Q4/2020*.