



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:

25. November 2020

Inhaltsverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
Listingverzeichnis	VI
1 Installation	1
1.1 ESP8266	1
1.2 Backend und Frontend	2
2 Fachkonzept	3
2.1 ESP8266	3
2.2 Backend	4
2.3 Frontend	4
3 ER-Diagramm	5
4 Projekt-Architektur	7
4.1 Frontend	7
4.2 Backend	7
4.3 NodeMCU	8
5 Abläufe	9
5.1 Backend	9
5.2 NodeMCU	9
6 Schnittstellen	10
7 Security	12
7.1 ESP8266	12
7.2 Backend	12
7.3 Frontend	13
7.4 Github	13
7.5 Deployment	13
8 Tests	15
8.1 Automatische Tests	15
8.2 Manuelle Klicktests	16
9 Software-Engineering	17

10 Anwendungsfälle	19
11 GUI-Konzept	20
12 Projektplanung	22
12.1 Verwendete Tools und Methodik	22
12.2 Projektmanagement	23
12.3 Soll-Ist-Vergleich	23
13 Schlussbetrachtung	24
13.1 Bewertung	24
13.2 Ausblick	25
13.3 Fazit	25
Anhang	26
Quellenverzeichnis	44

Abbildungsverzeichnis

Abbildung 1: Komponenten des Gesamtsystems	3
Abbildung 2: ER-Diagramm des Projekts	5
Abbildung 3: Root-Ordnerstruktur	7
Abbildung 4: Testabdeckung Frontend	15
Abbildung 5: Testabdeckung Backend	16
Abbildung 6: Use Case Diagramm des Projekts	19
Abbildung 7: GUI-Konzept	20
Abbildung 8: Farben-Konzept	21
Abbildung 9: Finale GUI	31
Abbildung 10: Datumsauswahl Popup (finale GUI)	32
Abbildung 11: Admininterface (finale GUI)	33
Abbildung 12: Informationsmeldung (finale GUI)	34
Abbildung 13: Frontend-Ordnerstruktur	36
Abbildung 14: Backend-Ordnerstruktur	37
Abbildung 15: NodeMCU-Ordnerstruktur	37
Abbildung 16: Positionierung Trend	39
Abbildung 17: Zustandsdiagramm setup-Funktion	40
Abbildung 18: Zustandsdiagramm loop-Funktion	41
Abbildung 19: Zustandsdiagramm sendCachedData-Funktion	43

Tabellenverzeichnis

Tabelle 1: Verwendete Arduino-Bibliotheken	1
Tabelle 2: Pin Layout für Verbindung ESP mit BME	1
Tabelle 3: Bewertung der Anwendung	24
Tabelle 4: Anforderungen	35

Listingverzeichnis

Listing 1: Antwort GET-Request /weatherData	27
Listing 2: Antwort GET-Request /sensorData/id/1	28
Listing 3: Antwort GET-Request /sensors	29
Listing 4: Antwort GET-Request /sensor/id/1	29
Listing 5: Beispielhafter Logauszug Backend	30
Listing 6: Beispielhafter Logauszug Frontend	30
Listing 7: Pseudocode Regressionsgerade	38

1 Installation

1.1 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
EasyNTPCClient (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

1.2 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 entschieden 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 gegebenenmaßen 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.

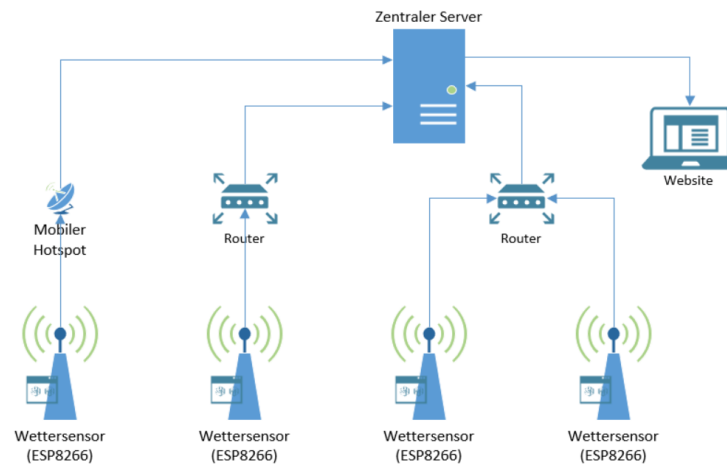


Abbildung 1: Komponenten des Gesamtsystems (eigene Darstellung)

Der „zentrale Server“ besteht aus zwei Modulen, dem „Backend“ sowie dem „Frontend“ welche im Folgenden vorgestellt werden.

2.1 ESP8266

Wie in Unterabschnitt 1.1 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.

2.2 Backend

Das Backend-Modul des Projekts, wird auf dem Zentralen Server bereitgestellt. Das Backend basiert auf der „Nodes.js“¹ Runtime in Version 14.4, ist somit in Javascript geschrieben und verwendet eine SQLite-Datenbank in Version 3 für die Speicherung der Daten. Für die Kommunikation zu den Sensoren und dem Frontend ist eine REST-API vorhanden, die in Abschnitt 6 näher erläutert wird.

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 Frontend

Das Frontend-Modul, welches die Weboberfläche bereitstellt, basiert ebenfalls auf „Node.js“ in Version 14.4. Aufgrund des Projektaufbaus ist es wie oben genannt möglich, das Hosting von Backend und Frontend zu trennen.

Für die Bereitstellung des Frontends nutzen wir die durch „Express“² bereitgestellte „Router“-Library. Die Darstellung der Messdaten auf der Website erfolgt mit „Chart.js“³. Die Auswahl eines Zeitraums für die Darstellung eines Intervalls ist mit der Library „daterangepicker“⁴ umgesetzt. Zusätzlich wurde für das Projekt „Browserify“⁵ verwendet, um zu verhindern, dass mehrere Dateien im Frontend geserved werden müssen, indem die einzelnen Dateien zu je einer Datei für die Haupt- und Adminseite zusammengefasst werden. Die HTML- und CSS-Komponenten der Website sind mit Bootstrap 4 erstellt worden. Auf die Gestaltung wird in Abschnitt 11 näher eingegangen.

¹nodejs

²express.2020

³chartjs.2020

⁴daterangepicker.2020

⁵browserify.2020

3 ER-Diagramm

In diesem Kapitel wird die Datenbankstruktur des Projekts dargestellt.

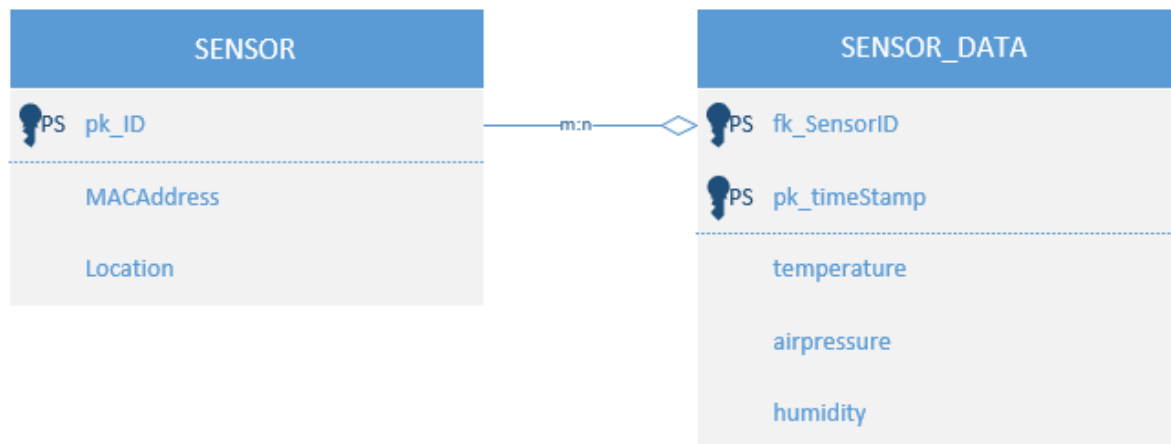


Abbildung 2: ER-Diagramm des Projekts (eigene Darstellung)

In Abbildung 2 ist das ER-Diagramm des Projekts dargestellt. Die Datenbankstruktur ist bewusst simpel aber erweiterbar gehalten.

SENSOR

Die Tabelle Sensor beinhaltet alle Informationen über die Sensoren des Systems. Neben einer vom System vergebenen ID sind Informationen über die MAC-Adresse und den Standort enthalten. Die MAC-Adresse wird bei jeder Request vom System zur Identifikation mitgeschickt und dient im Backend der Zuordnung von Messwerten zu Sensoren. Die MAC-Adresse wird als String gespeichert, es findet jedoch im Backend eine Überprüfung statt, die sicherstellt dass keine fehlerhafte Formatierung möglich ist. Die Spalte „Location“ ermöglicht die Zuordnung eines Standorts zu einer Messstation. Dieser Wert kann im Frontend nur durch den Administrator festgelegt werden, weshalb hier keine besondere Datenvalidierung notwendig ist und die Speicherung als String die größtmögliche Flexibilität bietet.

SENSOR_DATA

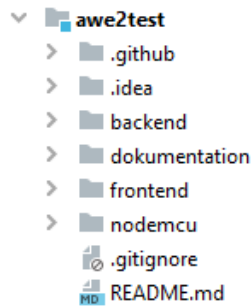
In dieser Tabelle werden die einzelnen Datensätze gespeichert. Ein Datensatz besteht standardmäßig aus einem Zeitstempel, und je einem Messwert für Temperatur, Luftdruck und

-Feuchtigkeit. Die Einzigartigkeit eines Datensatzes wird durch die Unique-Property von Sensor-ID und Zeitstempel sichergestellt. Der Sensor überträgt die oben genannten Daten zusätzlich zu seiner eigenen MAC-Adresse. Im Backend wird vor dem Speichern des Messwerts anhand der MAC-Adresse die Sensor-ID ermittelt (wenn der Sensor bereits bekannt ist) oder neu vergeben und dem Datensatz hinzugefügt, mit dem die Daten dann in die Datenbank eingefügt werden. Zeitstempel und die drei Messwerte sind hierbei numerisch gewählt, wobei integer als Datentyp ausreichend ist, um die möglichen Zeitstempel abzudecken und beispielsweise float genügt, um die möglichen Messwerte mit Komma zu speichern. Im Backend erfolgt eine Validierung der Daten, die verhindert dass verfälschte Messwerte oder Zeitstempel (zum Beispiel in Folge eines Messfehlers) in die Datenbank geraten.

4 Projekt-Architektur

Die Ordnerstruktur des Projekts ist in Abbildung 3 zu sehen. Die Strukturen für das Frontend, Backend sowie den NodeMCU Code sind im Anhang 5 zu finden.

Abbildung 3: Root-Ordnerstruktur



Quelle: Eigene Darstellung

Die Quelltexte der drei Komponenten NodeMCU, Backend und Frontend sind voneinander getrennt, werden aber als ein Gesamtprojekt ausgeliefert.

4.1 Frontend

Die Struktur des Frontend-Quellcodes ist eine Mischung aus klassischer Web- und Nodejs-Projektstruktur. In dem Ordner *resources* sind Ressourcen zu finden, welche der Webserver den Clients bereitstellt. Diese sind in *css*, *img* und *js* gegliedert. Verwendete Libraries, wie z.B. *sir.js* befinden sich in eigenen Unterordnern, während der eigenentwickelte Quellcode sich direkt in den Ordnern befindet.

Im Root-Ordner des Frontends befinden sich, für die Komponente, globale Dateien, wie z.B. das Buildscript für Docker, die *package.json* für Node.js und die *.eslintrc.json* zur Konfiguration des JavaScript-Linters.

4.2 Backend

Die Struktur des Backend-Quellcodes orientiert sich an der üblichen Struktur für Node.js Projekte, ist jedoch absichtlich simpel und flach, anstatt tief hierarchisch, gestaltet. Dies

ermöglicht bei der kleinen Größe der Anwendung eine effizientere Navigation durch die Dateien. In dem Unterordner *src* befinden die Quelltexte des Backends, in *test* die Quelltextdatei *testMain.js* sowie die *testData.js*, welche nur zum Generieren von Testdaten verwendet wird.

4.3 NodeMCU

In dieser Komponente befindet sich im Unterordner *src/nocemcu* die aktuell einzige Quelltextdatei für den NodeMCU. Im Ordner *ress* befindet sich zusätzlich eine Tabelle des Wirings von dem NodeMCU und dem BME280-Sensor.

hyperref

5 Ablaeufe

In diesem Kapitel werden die Abläufe der jeweiligen Anwendungen (vereinfacht) dargestellt. Für die Komponenten Backend und NodeMCU wurde ein UML-Ablaufdiagramm erstellt. Für das Frontend wurde darauf verzichtet, da bereits ein Use-Case-Diagramm besteht und der technische Ablauf sehr schnuckelig ist. Die Zustandsdiagramme befinden sich in Abschnitt 7.

5.1 Backend

5.2 NodeMCU

Das Zustandsdiagramm ist in Abbildung 17, Abbildung 18 und Abbildung 19 zu sehen. Es wurde zu Gunsten der Übersichtlichkeit in drei Teile aufgeteilt. Es folgt eine grobe Erläuterung. Auf eine detaillierte Erläuterung wird verzichtet, da hierzu das Diagramm sowie der Quelltext analysiert werden kann. Nach dem Boot wird in die setup-Funktion gesprungen. In dieser wird nach einem BME280 Sensor gesucht. Insofern einer gefunden, wird die WLAN-Verbindung aufgebaut. Anschließend wird die loop-Funktion endlos in einer Schleife aufgerufen. Diese beginnt mit dem Auslesen der Sensordaten, welche in eine Liste gespeichert werden. Wenn die Liste voll ist, wird das erste Element zuvor entfernt. Nach Prüfung, ob die WLAN-Verbindung vorhanden ist, wird die sendCachedData-Funktion aufgerufen. In dieser wird über alle Listenelemente iteriert. In jeder Iteration wird eine HTTP-Verbindung zum Backend aufgebaut. Anschließend wird ein eventuell inkorrektter Timestamp (mit Wert = 0) korrigiert. Danach wird der JSON-String für den HTTP-Body aus den Daten erstellt. Nach anschließendem versenden wird der HTTP-Code ausgewertet. Bei einem erwarteten Wert wird das Element aus der Liste entfernt. Zuletzt wird die HTTP-Verbindung geschlossen und in die nächste Iteration gegangen.

6 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.⁶

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

⁶Für beinhaltete Datentypen siehe Abschnitt 3

- *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.⁷

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.⁸

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

⁷Für beinhaltete Daten siehe Abschnitt 3

⁸Dieser steht nur dem Admin zur Verfügung und ist genauer unter Abschnitt 7 spezifiziert.

7 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.

7.1 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.⁹ 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.

7.2 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 6), 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

⁹Darüber 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.

Abwegung des möglichen Schaden gegenüber den Kosten der Implementierung haben wir von einer weiteren Absicherung abgesehen.

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.

7.3 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.

7.4 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.

7.5 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 realisieren¹⁰. Dadurch ergibt sich eine zusätzliche Instanz welche zum Schutz der Anwendung beiträgt. Mit Hilfe des Reverse-Proxy-Servers können

¹⁰Hierzu wurde während der Entwicklung ein `swag` Docker Container genutzt, alternativ kann `traefik` als Docker Container genutzt werden

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** realisieren.¹¹

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 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**, ebenfalls als Docker Container, an.

¹¹Dieses ist in **swag** bereits inkludiert

8 Tests

8.1 Automatische Tests

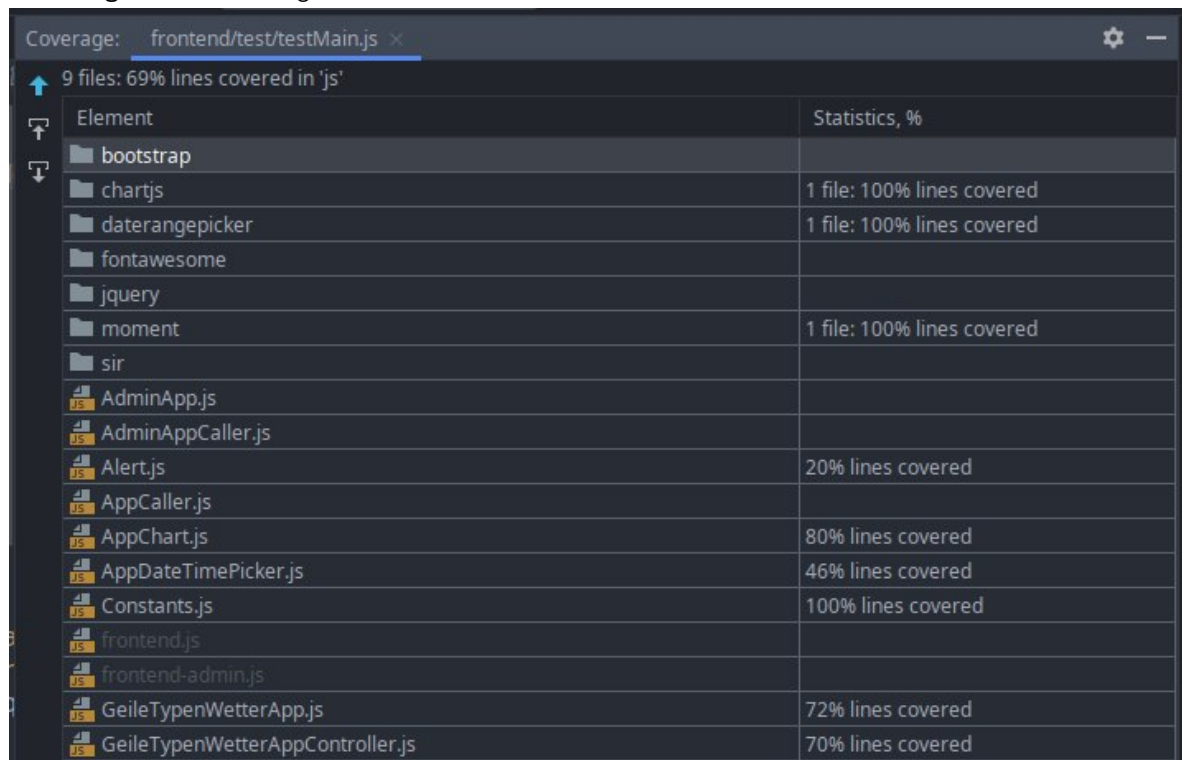
Für die relevantesten Funktionen der Anwendung wurden Unit-Tests geschrieben. Im Backend sowie Frontend wurden die Frameworks Chai und Mocha verwendet. Chai ist eine Assertion Library, während Mocha ein komplettes Test-Framework ist. Die Struktur der Tests wird somit durch Mocha vorgegeben. Durch Chai werden lediglich die Assertions evaluiert.

Die Testklassen heißen jeweils testMain.js und befinden sich in den src/test Ordnern des Front- und Backends.

Testabdeckung

Die erreichte Testabdeckungen ist in Abbildung 4 und Abbildung 5 zu sehen.

Abbildung 4: Testabdeckung Frontend



Coverage: frontend/test/testMain.js x

9 files: 69% lines covered in 'js'

Element	Statistics, %
bootstrap	
chartjs	1 file: 100% lines covered
daterangepicker	1 file: 100% lines covered
fontawesome	
jquery	
moment	1 file: 100% lines covered
sir	
AdminApp.js	
AdminAppCaller.js	
Alert.js	20% lines covered
AppCaller.js	
AppChart.js	80% lines covered
AppDateTimePicker.js	46% lines covered
Constants.js	100% lines covered
frontend.js	
frontend-admin.js	
GeileTypenWetterApp.js	72% lines covered
GeileTypenWetterAppController.js	70% lines covered

Quelle: Eigene Darstellung

Abbildung 5: Testabdeckung Backend

Element	Statistics, %
databaseConnection.js	39% lines covered
helper.js	78% lines covered
persistanceService.js	100% lines covered
restController.js	96% lines covered

Quelle: Eigene Darstellung

8.2 Manuelle Klicktests

Es wurden manuelle Tests der Benutzeroberfläche ausgeführt, um etwaige Fehler dort zu entdecken. Diese wurden dokumentiert.

Inhalte

Es sollen folgende Inhalte dokumentiert werden:

- Git Commit Hash (Programmversion)
- Git Branch
- Verwendetes Betriebssystem
- Verwendeter Browser inkl. Build
- Screenshots bei Darstellungsfehlern
- Bildschirmauflösung, insbesondere bei mobiler Ansicht

Darüber hinaus wurde die Anwendung in der mobilen (responsiven) Ansicht getestet, was ebenfalls dokumentiert wurde.

9 Software-Engineering

In diesem Kapitel wird auf den Software-Engineering-Aspekt des Projekts eingegangen. Anhand von drei Beispielen wird dargelegt, wie wir die Code-Qualität durch Verwendung dieser Prinzipien sichergestellt und verbessert haben.

Code-Kommentierung

Im Projekt wurde auf die Kommentierung des Quellcodes weitestgehend verzichtet. In der Industrie ist heute der verbreitete Standard, dass eine tiefgehende Kommentierung des Codes nicht notwendig und nicht sinnvoll ist, um ein Verständnis für die Funktionalität zu schaffen. Stattdessen wird strikt darauf geachtet, gut lesbaren und verständlichen Code zu schreiben, der eine zügige Einarbeitung auch ohne Kommentare ermöglicht. Im Projekt wurde in Absprache mit Florian Wortmann¹² zum Großteil darauf verzichtet, den Code zu kommentieren. Ausnahmen bilden hier technisch komplexe Funktionen. Hierzu gehören zum einen beispielhaft die Verwendung von regulären Ausdrücken. Oder zum anderen innerhalb des Node-MCU Codes, die Erklärung unserer Funktion zur Zeitstempelkorrektur, da diese anhand einer „FIFO“-Queue die Werte berechnet. Des weiteren wurden Kommentare genutzt um den Code in inhaltlich getrennte Abschnitte zu unterteilen, beispielsweise die Unterteilung in Vorbereitungsfunktionen und tatsächliche Tests in den Front- und Backendtests. Eine gute Änderbarkeit des Codes wird somit primär durch die Verwendung von sprechenden Variablen- und Funktionsbezeichnungen und die Übersichtlichkeit durch Aufteilung in atomare Funktionen sichergestellt.

Continuous Integration

Um die durchgehende Funktionalität des Codes zu sichern und zu gewährleisten, dass nur funktionierender Code gepusht wird, wurde die CI-Funktionalität von GitHub eingebunden. Hierbei wurde für Backend sowie Frontend jeweils getrennt eine eigene Pipeline eingerichtet. Die CI stellt sicher, dass alle Tests erfolgreich durchlaufen und ermöglicht somit eine schnelle Fehlererkennung und gibt Ansätze zur Behebung. Durch die Kopplung von Tests im Zuge der Entwicklung von Funktionalität, sowie der automatischen Ausführung dieser auf einem unabhängigen „cleanen“ System, wird der Code direkt nach dem Push validiert. Die Verwendung der CI stellt somit sicher, dass der Code qualitativ hochwertig und fehlerfrei ist.

¹²vgl. BesprNotiz.2020

Linten

Für die Durchsetzung der Code-Konventionen wird der Linter ESLint¹³ als Plugin in IntelliJ genutzt. Der Linter stellt durch invasive Meldungen die Qualität und Lesbarkeit des geschriebenen Codes sicher und ist besonders bei der o.g. kommentar-armen Entwicklung wichtig, damit der Code nach dem Projekt auch durch andere Entwickler erweiterbar ist. Die Einstellung von ESLint wurden anhand der Code-Konventionen von Google vorgenommen, die in der Industrie weit verbreitet sind.

¹³ESLint.2020

10 Anwendungsfälle

In diesem Kapitel wird das Use Case Diagramm dargestellt, welches die Anwendungsfälle der Website darstellt. Da das Backend und der Mikrocontroller keine direkte Interaktion mit dem User ermöglichen, werden sie hier nicht erfasst.

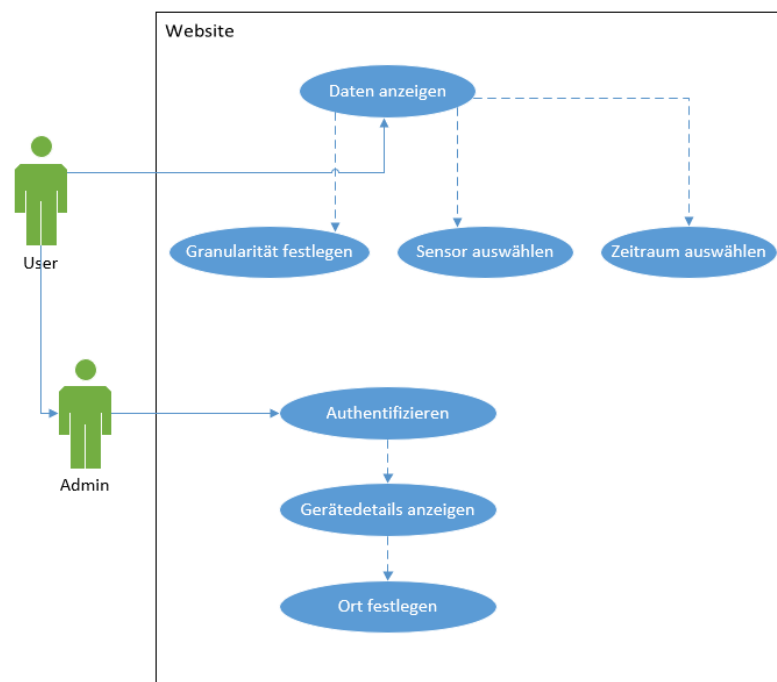


Abbildung 6: Use Case Diagramm des Projekts (eigene Darstellung)

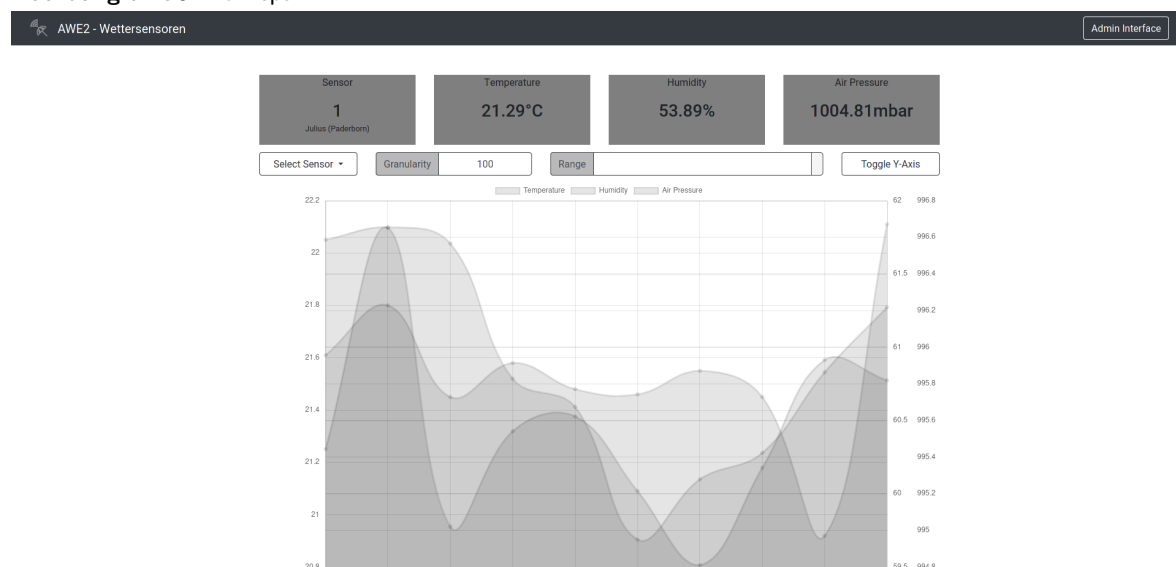
Der User hat auf der Website einen primären Anwendungsfall mit einigen Erweiterungen. Neben der Anzeige der Daten kann der User die Granularität festlegen, also ändern wie viele Datenpunkte im Diagramm angezeigt werden. Außerdem kann ein anderer Sensor aus dem Dropdown ausgewählt werden, um dessen Messwerte darzustellen. Über den sogenannten „Daterangepicker“ kann der Zeitraum festgelegt werden, in dem die Messdaten angezeigt werden.

Der Administrator hat neben den Anwendungsfällen des Users zusätzlich die Möglichkeit sich mit Nutzernamen und Passwort zu authentifizieren um das Administrator-Interface anzuzeigen. Dort sind Gerätedetails aufgeführt und der Administrator kann den Standort des Sensors festlegen.

11 GUI-Konzept

Dieses GUI-Konzept entspricht dem Stand vor dem Beginn der Entwicklung unseres Prototypen. Hilfreich war hierbei, dass zu diesem Zeitpunkt bereits die zu nutzenden Technologien festgelegt waren. Für das Konzept, abgebildet in Abb. 7, wurde „Bootstrap 4“¹⁴ genutzt. Hierdurch sind alle Buttons sowie UI-Elemente grundlegend einheitlich. Dies findet sich insbesondere in der Navigationsleiste sowie den Buttons und Input-Feldern wieder. Die Sprache ist auf Englisch gehalten, da Dies für uns am meisten gewohnt war. Innerhalb der Navigationsleiste findet sich zur linken der Name unserer Anwendung sowie unser Logo. Letzteres stellt stilisiert einen Regenschirm mit Funkverbindung da. Eine humoristische Anspielung auf den Einsatzzweck unserer Anwendung.

Abbildung 7: GUI-Konzept



Quelle: Eigene Darstellung

Ziel unseres Konzeptes war es eine konsistente, einheitliche und leicht verständliche Benutzeroberfläche zu generieren. Der Fokus unserer Anwendung ist die Darstellung und Auswertung von Wettersensoren. Unter diesen Aspekten wurde die Oberfläche entwickelt. Informationen sind zentral gehalten, direkt ersichtlich sind vier Blöcke mittig auf dem Bildschirm platziert. Durch diese ist einerseits der zurzeit ausgewählte Sensor andererseits alle drei aktuellen Sensorwerte im direkten Fokus des Anwenders. Alle Einstellungsmöglichkeiten sind innerhalb einer darunterliegenden Reihe platziert. Die Auswahl des darzustellenden Sensors erfolgt direkt unter der

¹⁴bootstrap

Ansicht des Aktuellen, auch hierdurch wird die Bedienung klar und für den Anwender einfach gehalten. Daneben finden sich alle weiteren Eingabemöglichkeiten für Parameter der Graphen. Das Hauptelement in Form der Graphen wird durch „chart.js“¹⁵ bereitgestellt. Hier haben wir uns entschieden alle Sensorverläufe innerhalb eines Graphen zu plotten um Diese zentral und sichtbar zu halten. Hier ist insbesondere anzumerken, dass Achsenbeschriftungen sowie Einfärbungen noch nicht vorhanden sind, da diese innerhalb des Codes durch Konfigurationen angepasst werden.

Durch die frühe Festlegung der Positionierung sowie der gesamten Oberfläche konnte sich während der Entwicklung auf die technische Implementierung konzentriert werden, da das fertige Design nur umgesetzt werden musste. So war es im folgenden nur noch notwendig geringfügige Anpassungen vorzunehmen um der Anwendung den letzten Schliff zu geben. Nach der Entwicklung des ersten Prototypen haben wir Dies einerseits in der Entwicklung eines einheitlichen Farbenkonzeptes umgesetzt. Die in Abb. 8 abgebildeten Farben basieren in den hellen sowie dunklen Tönen auf Bootstrap Standard-Farben. Für die weiteren Farbtöne haben wir uns eine Pastell-Farb-Palette ausgesucht. Diese wurden im folgenden einheitlich genutzt. Der Gelbton wurde ausschließlich für Informationen benutzt, die restlichen Farben wurden jeweils für Temperatur, Luftfeuchte und Luftdruck genutzt. Dies findet sich innerhalb der Blockelemente in der oberen Reihe aber auch insbesondere im Chart innerhalb der Graphen wieder, wo die Farben hervorhebend eingesetzt wurden. Andererseits wurden die Informationen um passende „Font Awesome“-Icons¹⁶ ergänzt. Hierdurch wird eine Orientierung auf der Oberfläche nochmals erleichtert und verbessert. Darüber hinaus wurden die Block Elemente innerhalb der GUI um runde Kanten ergänzt, hierdurch bekam die Anwendung eine modernere zeitgemäßere Oberfläche. Die insbesondere konsistenter und passender zu den in Bootstrap vorhandenen Buttons ist.

Abbildung 8: Farben-Konzept



Quelle: Eigene Darstellung

Die fertige GUI findet sich im Anhang 3.1. Weitere GUI-Elemente werden hier ebenfalls beschrieben. Dazu gehören die Datumsauswahl in Anhang 3.2, das Admininterface im Anhang 3.3, sowie die Informationsmeldungen im Anhang 3.4. Als netter Nebeneffekt ergab sich durch die Nutzung von Bootstrap, mit Hilfe geringer Anpassungen, eine nahezu vollständige Kompatibilität zu Mobilgeräten, auch wenn Diese nicht gefordert war.

¹⁵ **chart.js.2020**

¹⁶ **fontawesome**

12 Projektplanung

Für das Projekt werden zahlreiche Projektmanagementtools eingesetzt. Diese werden in diesem Kapitel erläutert.

12.1 Verwendete Tools und Methodik

Für das Projektmanagement wird Trello genutzt. Trello ist die führende Online-Plattform für Kanban¹⁷. Aufgrund der Struktur der Gruppe und des Projektumfangs bietet sich Kanban als Projektmanagementmethode an. Die Features werden als Arbeitspakete in Karten aufgeteilt und zugewiesen. Zusätzliche Karten werden angelegt, um organisatorische Features, Karten für Bugfixes und Randaufgaben (wie Refactoring) erweitert. Der Vorteil von Trello und Kanban für das Projekt sind besonders aufgrund der Corona-Situation gegeben, da es auch ohne ausführliche Besprechungen ermöglicht einzusehen, welches Gruppenmitglied aktuell an welchem Arbeitspaket arbeitet und welche Arbeitspakete vor der Vervollständigung stehen. Die drei üblichen Kanban-Bereiche „Neu“, „In Arbeit“ und „Fertig“ werden hierbei in Anlehnung an agile Softwareentwicklung in die folgenden Bereiche erweitert:

- **Nicht im Backlog:** In diesen Bereich kann auch außerhalb von einem Meeting jedes Gruppenmitglied Karten hinzufügen, die für sinnvoll erachtet werden. Karten in diesem Bereich werden beim folgenden Meeting diskutiert und gegebenenfalls ins Backlog geschoben.
- **Backlog:** Dieser Bereich beinhaltet alle Karten, die zur Bearbeitung anstehen. Karten in diesem Bereich werden in der Regel bei einem Meeting einem oder mehreren Gruppenmitgliedern zugewiesen.
- **In Arbeit:** In diesen Bereich werden Karten geschoben, die aktuell bearbeitet werden.
- **In Prüfung:** Nach der Fertigstellung einer Karte werden die entwickelten Features vom gesamten Team im nächsten Meeting überprüft. Dadurch stellen wir eine hohe Code-Qualität sicher und können bei Bedarf Karten für Bugfixes oder Refactoring in das Backlog schieben. Arbeitspakete bleiben hier bis zur Fertigstellung aller verwandten Bugfixes.
- **Fertig:** Nach der erfolgreichen Review werden alle fertiggestellten und auf ihre korrekte Funktionsweise überprüften Arbeitspakete in diesen Bereich geschoben.

¹⁷vgl. [atlassian](#)

12.2 Projektmanagement

Durch diese Struktur wurde sich explizit auch für an agile Entwicklung angelehnte Sprints entschieden. Auf eine strikte Festlegung auf eine agiles Umfeld wie beispielsweise Scrum wurde jedoch bewusst verzichtet. Der Grund hierfür ist die kurze Entwicklungsdauer im Semester und die geringe Gruppengröße. Die Sprints waren dadurch in der Regel nur wenige Tage lang. Für das Projekt wurde zunächst ein Prototyp nach dem Rapid-Prototyping-Prinzip¹⁸ entwickelt. Hierfür wurden die essentiellen Funktionen bereits in der ersten Woche umgesetzt, und damit eine rudimentäre Basis geschaffen, die in den folgenden Sprints um die zusätzlichen Features erweitert wurde. In der nächsten Projektphase wurde der Code refactored und Fehler die teilweise noch aus dem Prototyp stammten behoben. Durch diese Projektplanung konnte das Projekt bereits einen Monat nach Beginn für feature-complete erklärt werden. Danach wurden nur noch später entdeckte Fehler behoben und die Dokumentation begonnen.

12.3 Soll-Ist-Vergleich

Die einzelnen Features, deren Einstufung und der Status sind im Anhang in Abschnitt 4 aufgeführt. Die Darstellung basiert auf **Wortmann.2020** und wurde in Muss-, Kann und Bonus-Anforderungen gegliedert. Bonusanforderungen waren hierbei zunächst nicht genannt, kamen jedoch in Besprechungen als mögliche Erweiterung hinzu. Das Projekt wurde von Anfang an so konzipiert, dass eine hohe Unabhängigkeit der einzelnen Komponenten gegeben ist. Die Anwendung kann beliebig auf die Gegebenheiten angepasst werden, örtliche Trennung sind genauso möglich wie Fernzugriff auf die Daten und Verbindung des Sensors mit einem mobilen Hotspot. Auch während des Betriebs können einzelne Bestandteile des Projekts bei Bedarf in den Konfigurationsdateien neu eingestellt werden. Alle Anforderungen wurden so umgesetzt, dass auch eine Veränderung oder Erweiterung der Software im späteren Verlauf problemlos möglich ist.

¹⁸vgl. **krypczyk.2019**

13 Schlussbetrachtung

13.1 Bewertung

Alle Muss-Features wurden erfolgreich implementiert. Auch die meisten Kann-Features wurden umgesetzt. Lediglich die Wetterprognose wurde nicht entwickelt, da sie nach weiterer Recherche aus den gewonnenen Daten zu ungenau gewesen wäre. Alle Features wurden gemäß der Projektplanung fristgerecht fertiggestellt. Sämtliche Deadlines konnten insbesondere durch den frühen Projektstart eingehalten werden. Dies bezieht sich auf einzelne Teilaufgaben sowie das Gesamtprojekt. Durch erste Erfahrungen der Entwickler mit JavaScript verlief der Einstieg in eine produktive Entwicklung zügig.

Leider wurde das Team durch die exklusiv digitale Kommunikation - bedingt durch die anhaltende Corona-Pandemie - in der Zusammenarbeit etwas gebremst. Auch die - sonst übliche - hervorragende Arbeitsatmosphäre wurde dadurch verschlechtert.

Die letzten Releases der Anwendung liefen über einen Zeitraum von über vier Wochen stabil auf dem Produktivsystem mit einer Uptime von 100%. Darüber hinaus gab es keine Fehler in den automatischen sowie manuellen Tests des Front- und Backends.

Die Qualität der erstellten Anwendung wird von den Entwicklern selbst folgendermaßen eingeschätzt (1-10 Punkte):

Tabelle 3: Bewertung der Anwendung

Änderbarkeit	Benutzbarkeit	Effizienz	Funktionalität	Übertragbarkeit	Zuverlässigkeit
7	8	9	8	9	9

Die Bewertung richtet sich nach ISO/IEC 9126. Es folgt eine kurze Erläuterung der einzelnen Bewertungen. Das am schlechtesten bewertete Kriterium ist die Änderbarkeit. Diese ist auf die verwendete JavaScript Programmiersprache zurückzuführen. Diese ist aufgrund ihrer schwachen Typisierung und inkonsistenten Implementierung anfällig für Unübersichtlichkeit bei wachsenden Anwendungen. Sie wird jedoch für mehr als ausreichend für das Projekt in der aktuellen Größe angesehen. Im Gegensatz dazu bietet sie zusammen mit Node.js, dem standardmäßigen Non-Blocking-IO sowie Asynchronität eine sehr gute Effizienz für parallele Aufgaben, wie das - in diesem Fall - bereitstellen eines Webservers. Die Fehleranfälligkeit von JavaScript wurde durch automatisierte Tests möglichst ausgeglichen, wodurch die hohe Zuverlässigkeit entstanden ist. Die Übertragbarkeit wird als sehr hoch eingeschätzt, da die Anwendung als Docker Image auf sämtlichen modernen Serverumgebungen problemlos installiert werden kann. Die Funktionalität und Benutzbarkeit entstammen unserer subjektiven Einschätzung durch manuelle Tests sowie der von dritten Testpersonen.

13.2 Ausblick

Über die von uns implementierte Funktionalität hinaus, können wir uns die Ergänzung um eine Trendanzeige vorstellen. Diese war im Rahmen unseres Projektes zeitlich nicht umsetzbar, könnte aber im Anschluss ergänzend hinzugefügt werden. Sinn dieser wäre es, anhand von Regression¹⁹ eine „Wetterprognose“ aufzustellen. Hierzu sollte für die jeweiligen Sensorwerte ein Trend in der Entwicklung berechnet werden. Das liesse sich anhand einer Regressionsgeraden der „n“-letzten²⁰ Datenpunkte umsetzen (Siehe Anhang 6). Hierdurch würde unsere Anwendung um sinnvolle Funktionalität erweitert welche dem Nutzer durch klar erkennbare Prognosen einen Mehrwert bieten würde.

13.3 Fazit

Im Soll-ist Vergleich ist zu sehen, dass die Produktivität innerhalb des Projekts hervorragend war. Das Team bietet somit eine gute Grundlage für weitere Zusammenarbeiten. Es wird gehofft, dass zukünftig wieder eine persönliches Zusammentreffen möglich ist, um auch mal auf den gemeinsamen Erfolg einen Bolkstoff zischen zu können.

¹⁹Vgl. **regression**

²⁰Dieser Wert ist zum einen abhängig vom Sendeintervall, zum anderen sollte dieser im Zuge der Implementierung evaluiert werden, um eine sinnvolle Prognose zu erhalten.

Anhang

Anhangsverzeichnis

Anhang 1:	Schnittstellen	27
Anhang 1.1:	Antwort GET-Request auf „/weatherData“	27
Anhang 1.2:	Antwort GET-Request auf „/sensorData/id/1“	28
Anhang 1.3:	Antwort GET-Request auf „/sensors“	29
Anhang 1.4:	Antwort GET-Request auf „/sensor/id/1“	29
Anhang 2:	Logdaten	30
Anhang 2.1:	Beispielhafter Logauszug Backend	30
Anhang 2.2:	Beispielhafter Logauszug Frontend	30
Anhang 3:	GUI	30
Anhang 3.1:	Finales GUI Design	31
Anhang 3.2:	Datumsauswahl	32
Anhang 3.3:	Admininterface	33
Anhang 3.4:	Informationsmeldungen	34
Anhang 4:	Soll-Ist-Vergleich	35
Anhang 5:	Projekt-Architektur	36
Anhang 5.1:	Frontend	36
Anhang 5.2:	Backend	37
Anhang 5.3:	NodeMCU	37
Anhang 6:	Ausblick - Wetterprognose	38
Anhang 6.1:	Pseudocode Umsetzung	38
Anhang 6.2:	GUI Positionierung Konzeptuell	39
Anhang 7:	Ablaeufe	39
Anhang 7.1:	Setup	40
Anhang 7.2:	Loop	41
Anhang 7.3:	SendCachedData	42

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]

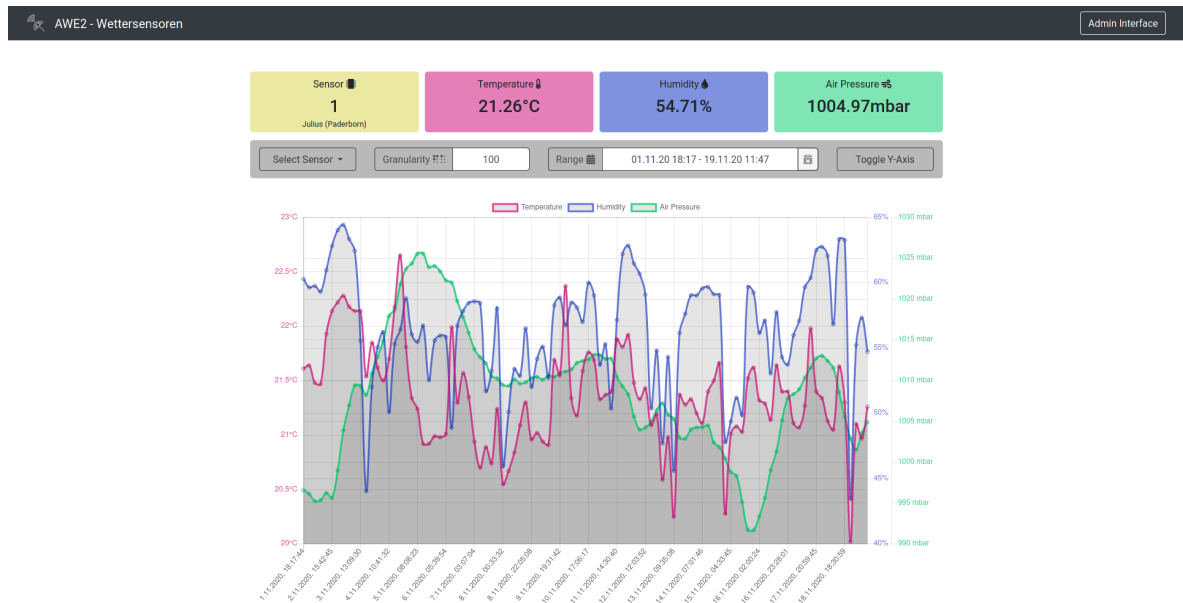
16.11.2020, 12:50:18 - INFO :
  GOT REQUEST TO [/robots.txt] FROM [$ZENSIERTE_IP_ADRESSE]
```

Anhang 3 GUI

Anhang 3.1 Finales GUI Design

Hier wird mit Blick auf den Prototypen sichtbar, wie geringfügig die Änderungen gegenüber diesem ausgefallen sind.

Abbildung 9: Finale GUI

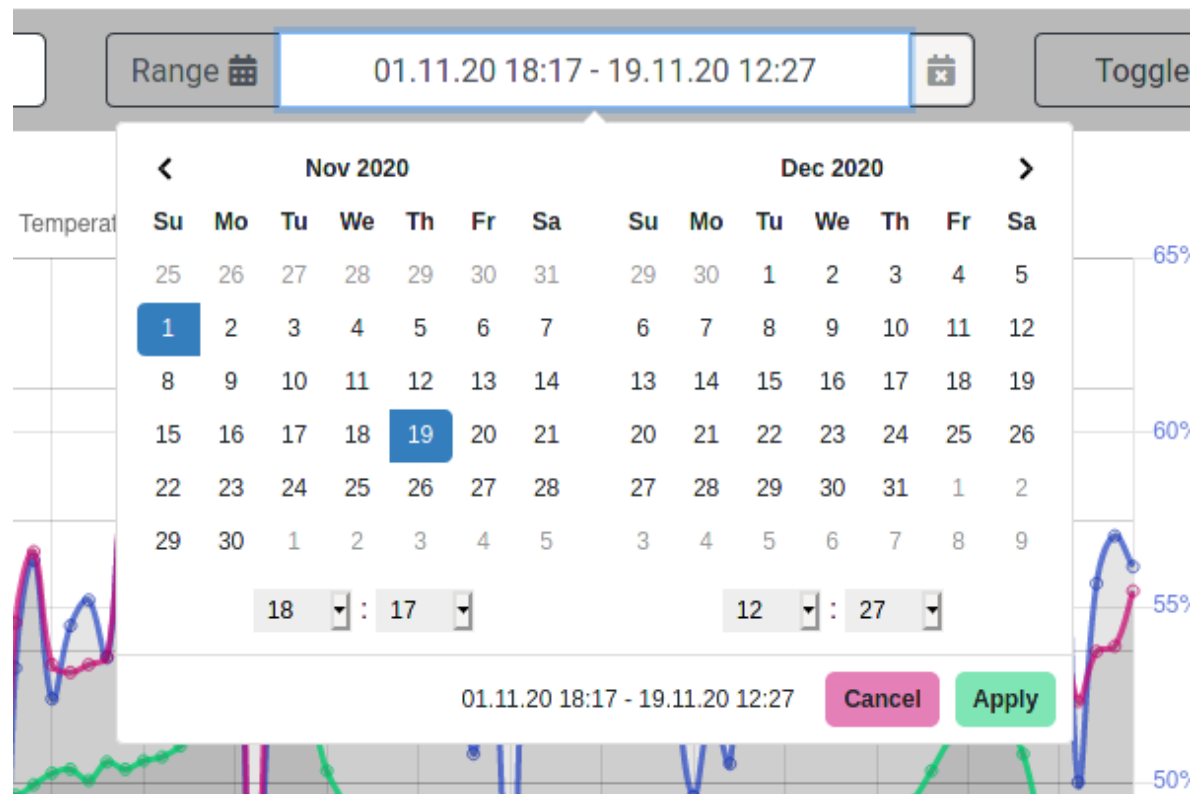


Quelle: Eigene Darstellung

Anhang 3.2 Datumsauswahl

Die Datums- sowie Zeitauswahl beruht auf der Library „Date Range Picker“²¹. Diese ermöglicht eine einfache, ohne Anleitung verständliche, Auswahl des anzuzeigenden Zeitbereiches. Auch hier wurden die Farben durch unsere Pastell-Farb-Palette angepasst.

Abbildung 10: Datumsauswahl Popup (finale GUI)



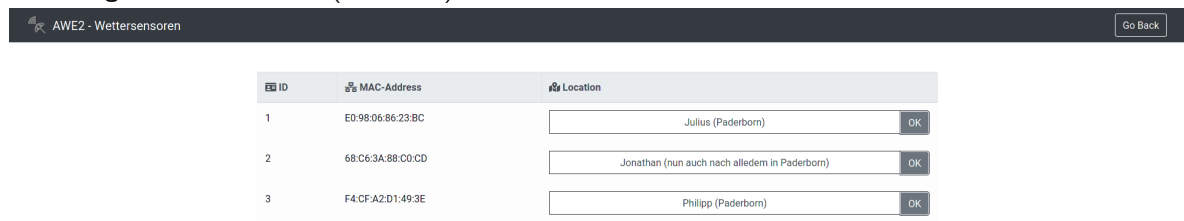
Quelle: Eigene Darstellung

²¹ daterangepicker.2020

Anhang 3.3 Admininterface

Das Admininterface wurde einfach gehalten und greift wie die Hauptseite auf „Font-Awesome“²² für die Icons zurück. Alle zur Verfügung stehenden Sensoren werden hier innerhalb einer Tabelle aufgelistet, der Standort des Sensors lässt sich einfach durch ein Inputfeld mit zugehörigem Button anpassen. Bei erfolgreicher Eingabe wird eine positive Rückmeldung eingeblendet. Diese findet sich beispielhaft in Anhang 3.4.

Abbildung 11: Admininterface (finale GUI)



The screenshot shows a web interface titled "AWE2 - Wettersensoren" with a "Go Back" button. Below the header is a table with three columns: "ID", "MAC-Address", and "Location". The table contains three rows of sensor data. Each row has an input field for the location and an "OK" button.

ID	MAC-Address	Location
1	E0:98:06:86:23:BC	<input type="text" value="Julius (Paderborn)"/> <input type="button" value="OK"/>
2	68:C6:3A:88:C0:CD	<input type="text" value="Jonathan (nun auch nach alledem in Paderborn)"/> <input type="button" value="OK"/>
3	F4:CF:A2:D1:49:3E	<input type="text" value="Philipp (Paderborn)"/> <input type="button" value="OK"/>

Quelle: Eigene Darstellung

²²fontawesome

Anhang 3.4 Informationsmeldungen

Diese Einblendung wird zur Information des Nutzers angezeigt. Neben dieser Darstellung welche nach erfolgreicher Änderung des Sensorstandortes eingeblendet wird, existiert eine „Error“-Version. Diese ist in Rot gehalten und zeigt Fehler auf.

Abbildung 12: Informationsmeldung (finale GUI)

The screenshot shows a dark header bar with the text 'E2 - Wettersensoren'. Below it is a green success message box that says 'Saved Successfully!'. Underneath is a table with three columns: 'ID', 'MAC-Address', and 'Location'. The table contains two rows of sensor data. Each row has a text input field for the location and an 'OK' button.

ID	MAC-Address	Location
1	E0:98:06:86:23:BC	Julius
2	68:C6:3A:88:C0:CD	Jonathan (nun auch nach alledem in Paderborn)

Quelle: Eigene Darstellung

Anhang 4 Soll-Ist-Vergleich

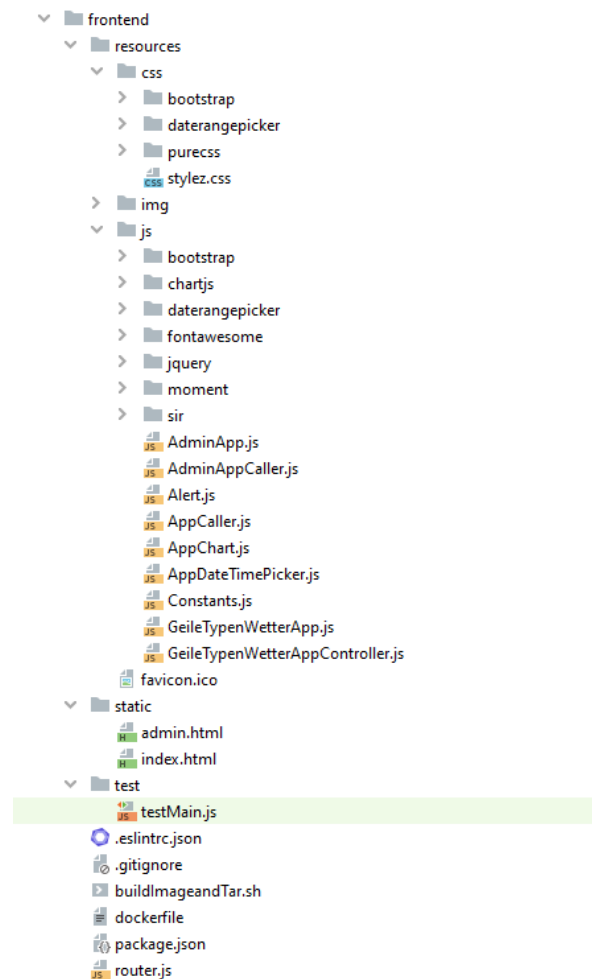
Anforderung	Einstufung	Status
Es soll ein System zur dezentralen Erfassung von Daten erstellt werden Folgende Daten sollen erfasst werden: Temperatur, Luftdruck, Luftfeuchtigkeit	Muss	Erledigt
Jedes Erfassungssystem ist ein Ort oder Raum fest verbunden	Muss	Erledigt
Es gibt einen Server auf dem die Daten abgelegt werden Server und Erfassungssystem müssen nicht im selben Intranet sein	Muss	Erledigt
Im Falle einer Kommunikationsunterbrechung werden die Daten lokal gespeichert Die Daten werden in diesem Fall übertragen sobald die Verbindung wieder aufgebaut wird	Kann	Erledigt
Die Kommunikation soll mit minimalem Energieaufwand durchgeführt werden	Kann	Erledigt
Die Erfassungsgeräte können auch über einen mobilen Hotspot Daten an den Server senden	Kann	Erledigt
Die Daten sollen von einem beliebigen Ort mit Internetanschluss abgerufen werden	Muss	Erledigt
Die Daten sollen grafisch dargestellt werden	Muss	Erledigt
In der Darstellung der Daten können bestimmte Datentypen oder einzelne Erfassungssysteme ausgewählt werden	Kann	Erledigt
Die Granularität der Daten ist variabel einstellbar	Muss	Erledigt
Alle angeschlossenen Geräte sollen für einen Administrator auf einer Plattform sichtbar sein	Muss	Erledigt
Mehrere Erfassungssystem pro Server sind möglich	Muss	Erledigt
Ein Erfassungssystem kann mit verschiedenen Servern kommunizieren	Kann	Erledigt
Es ist möglich zu sehen, wann der Sensor das letzte Mal Daten gesendet hat	Bonus	Erledigt
Es ist möglich eine Wetterprognose anzuzeigen	Bonus	Nicht erledigt

Tabelle 4: Anforderungen (nach Wortmann.2020, eigene Darstellung)

Anhang 5 Projekt-Architektur

Anhang 5.1 Frontend

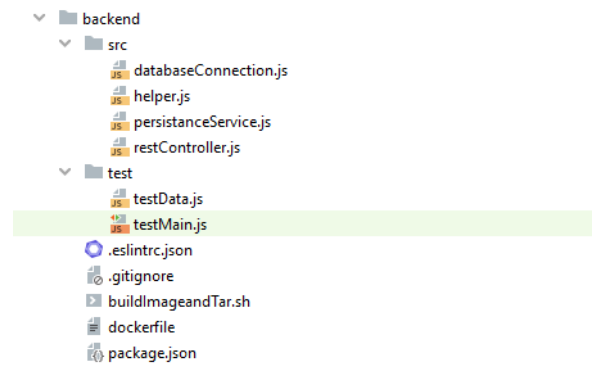
Abbildung 13: Frontend-Ordnerstruktur



Quelle: Eigene Darstellung

Anhang 5.2 Backend

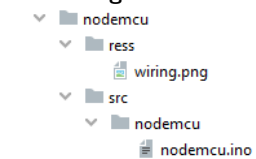
Abbildung 14: Backend-Ordnerstruktur



Quelle: Eigene Darstellung

Anhang 5.3 NodeMCU

Abbildung 15: NodeMCU-Ordnerstruktur



Quelle: Eigene Darstellung

Anhang 6 Ausblick - Wetterprognose

In diesem Abschnitt findet sich eine Prototypische Implementierung die zu Pseudocode abgeändert wurde. Für den spezifischen gewählten Sensor *specificSensor* werden die „n“-letzten Datenpunkte benutzt um mit Hilfe der Funktion *calculateTrends* eine Regressionsgerade über diese zu erzeugen. Hierzu werden zwei Hilfsfunktionen herangezogen. Die Funktion *createDataPointTuples* erzeugt aus den Sensordaten (*sensorData*) einen mehrdimensionalen Array welcher von der Library (hier beispielhaft anhand *regressionLibrary*) zur Erzeugung der linearen Regressionsgeraden verwendet wird. Des weiteren wird die Funktion *getLowerBoundGranularityOrSensordataLength* herangezogen um sicherzustellen dass keine „Out-of-Bounds“-Exceptions auftreten, indem der geringere Wert, anhand der Menge der vorhandenen Daten sowie den „n“-gewünschten Datenpunkten erzeugt wird.

Anhang 6.1 Pseudocode Umsetzung

Listing 7: Pseudocode Regressionsgerade

```

    calculateTrends(sensorData,specificSensor,granularity){
    let dataLength = this.
    getLowerBoundGranularityOrSensordataLength(sensorData, granularity)
    let trendData = this.
    createDataPointTuples(
        sensorData.timestamps.slice(-dataLength),
        sensorData.specificSensor.slice(-dataLength),
        dataLength
    );
    console.log(trendData);
    let trendSpecificSensor =
        regressionLibrary.linearRegression(trendData);
    console.log(trendSpecificSensor);
    }

    createDataPointTuples(timestamps,specificSensor, length){
    return Array.from({length: length}, (x, i) => {
        return [
            moment(timestamps[i], "DD.MM.YYYY, HH.mm.ss").unix(),
            specificSensor[i]
        ];
    });
    }

    getLowerBoundGranularityOrSensordataLength(sensorData, granularity) {
    return sensorData.timestamps.length < granularity ?
        sensorData.timestamps.length : granularity;
    }

```

Anhang 6.2 GUI Positionierung Konzeptuell

In Abb. 16 findet sich die Visualisierung der in Listing 7 errechneten Daten. Mit Hilfe der Steigung der Regressionsgerade wird das Icon eines Pfeils anhand von CSS im passenden Winkel rotiert. Hierdurch ist direkt unter den aktuellen Werten der Trend in der Wertentwicklung sichtbar. Idealerweise liesse sich dies noch durch einen Tooltip ergänzen, damit die Bedeutung des Pfeils für jeden intuitiv ersichtlich wird.

Abbildung 16: Positionierung Trend

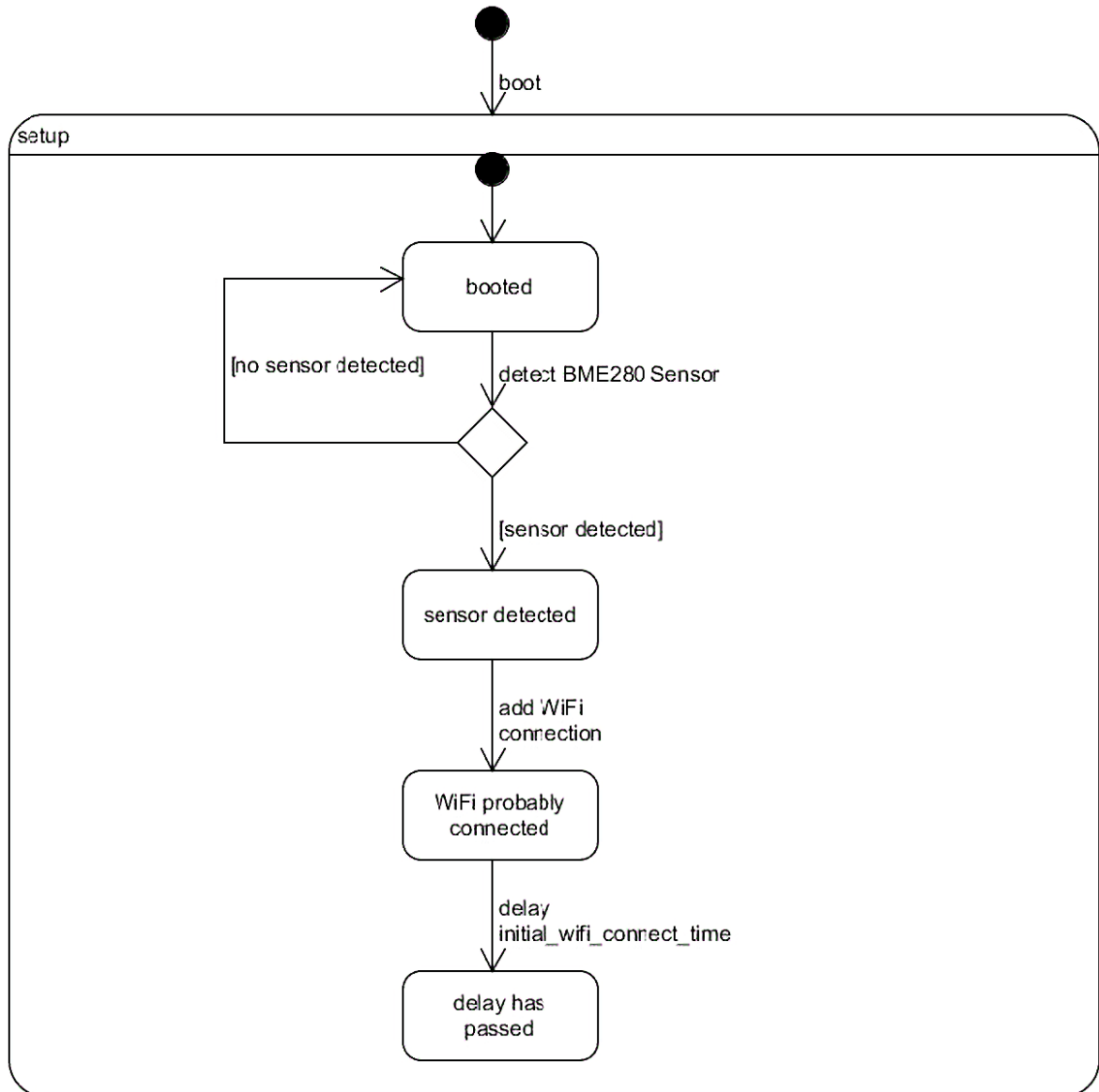


Quelle: Eigene Darstellung

Anhang 7 Abläufe

Anhang 7.1 Setup

Abbildung 17: Zustandsdiagramm setup-Funktion



Quelle: Eigene Darstellung

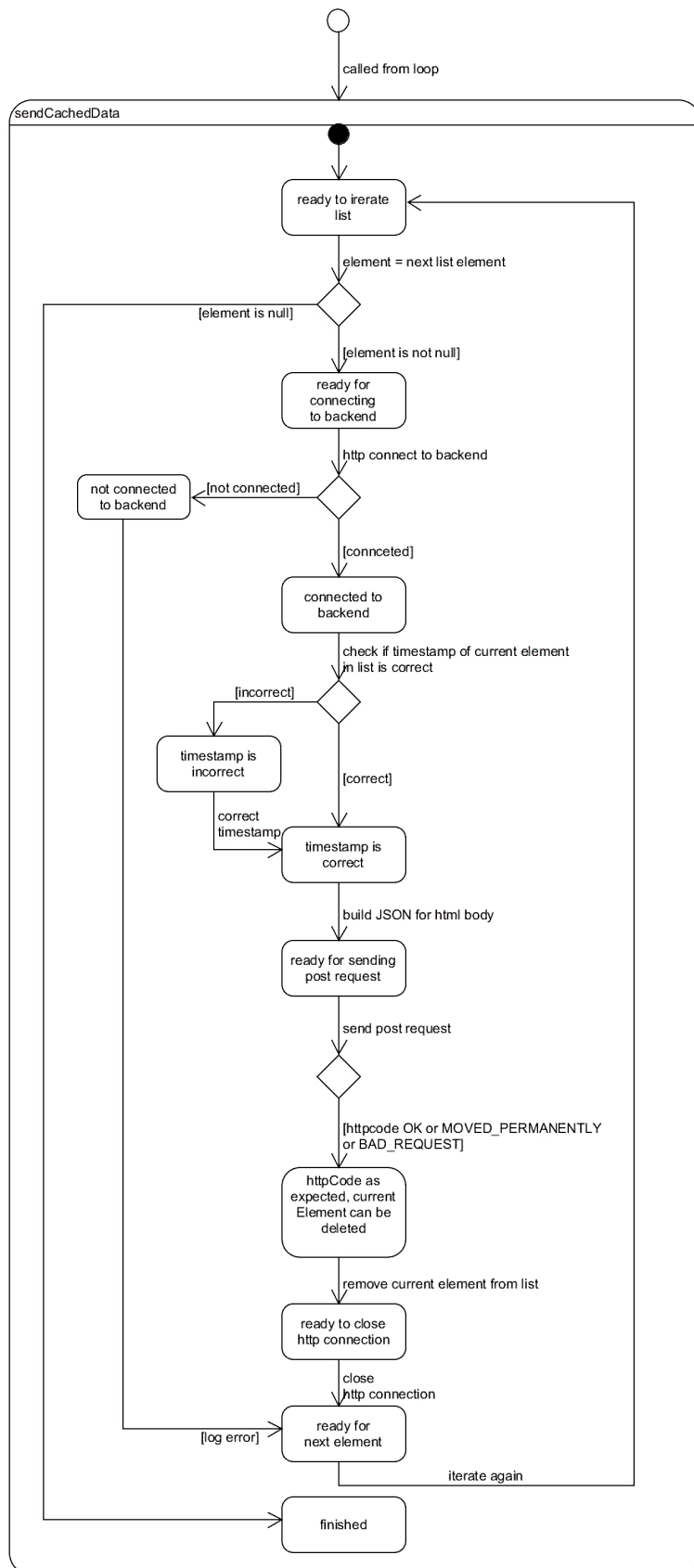
Anhang 7.2 Loop

Abbildung 18: Zustandsdiagramm loop-Funktion

Quelle: Eigene Darstellung

Anhang 7.3 SendCachedData

Abbildung 19: Zustandsdiagramm sendCachedData-Funktion



Quellenverzeichnis