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:
17. November 2020

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

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
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

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 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 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.

2.1 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.[1]

  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

[1] 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.[2]

  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.[3]

  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

[2] Für beinhaltete Daten siehe **??**
[3] Dieser steht nur dem Admin zur Verfügung und ist genauer unter Abschnitt 3 spezifiziert.

3 Security

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)"
  }
}
```

Quellenverzeichnis