PAW IoT-Cube

SoSe 19

Dokumentation
für Node-Red, die Weboberfläche
und die Firmware

Stand: 07.06.2019

Version: 3

Autor: AR

Version	Datum	Änderungen	Autor
1	06.06.19	Dokument erstellt	AR
2	07.06.19	Hinzufügen Kapitel Allgemeine Struktur	AR
3	07.06.19	Hinzufügen Dokumentation Firmware	AR, WD

1. Inhaltsverzeichnis

2. Node-Red & Weboberfläche

2.1. Allgemeine Struktur

Der Softwareteil des Projekts auf dem Raspberry Pi umfasst drei Bereiche: die Logik, durch Node-Red erstellt, die Benutzeroberfläche im Browser und die dahinterliegende Datenbank.

Die Datenbank wird von Node-Red genutzt, um eine Referenz der bekannten Sensoren, Aktoren und einige Konfigurationen für den Programmablauf zu hinterlegen. Gleichzeitig dient sie als Aufzeichnung der variablen Website und zum Loggen der aufgenommenen Sensordaten. Die Weboberfläche erhält alle benötigten Daten über Node-Red aus der Datenbank.

2.1.1. Aufbau Node-Red-Projekt

Das Projekt ist in vier Flows unterteilt, in denen verschiedene Bereiche der Logik ablaufen.

Discovery: Dieser Tab ist für das automatische Hinzufügen einer neuen Box zuständig, wobei dabei noch nicht das Generieren der neuen Dashboardbausteine inbegriffen ist.

MQTT pubsub: Hier wird die standardmäßige MQTT-Kommunikation abgewickelt. Es werden alle Sensortopics beobachtet und einkommende Nachrichten für die Verarbeitung vorbereitet. Zusätzlich werden von hier die Topics der Aktoren und für die Sensoreinstellungen gesetzt. Außerdem befindet sich hier der Watchdog, der den Verbindungsstatus der Sensoren überwacht.

Databases: In diesem Tab werden allgemeine Interaktionen mit den Datenbanken durchgeführt. Dazu gehören das Abspeichern der neuen Sensordaten, der Einstellungen, die von der Webseite gewünscht werden und das Übernehmen verschobener Bausteine auf dem Dashboard sowie das Verarbeiten der Regeln beim Verbinden von Aktor und Sensor.

Website: Hier befinden sich die Nodes, die aus den Quelldateien der Weboberfläche die Webseite hosten, und alle zugehörigen Funktionen,

um Daten an die einzelnen Tabs der Seite liefern oder entgegennehmen zu können.

CSV generation: Hier werden die gesammelten Daten eines gewünschten Sensors ausgelesen und pro Messgröße in eine CSV-Datei umgewandelt und anschließend als Zip-Datei abgespeichert, um auf der Webseite als Download zur Verfügung zu stehen.

2.1.2. Abhängigkeiten Node-Red

Um das Projekt in Node-Red nutzen zu können, werden folgende Erweiterungen benötigt:

- *node-red-contrib-uibuilder*
- *node-red-node-sqlite*
- *node-red-contrib-moment*
- *node-red-contrib-zip*

Auf dem Raspberry Pi werden neben Mosquitto und anderen Grundabhängigkeiten noch die Programme *git* und *zip* benötigt.

2.1.3. Abhängigkeiten Code Weboberfläche

Damit die Weboberfläche wie gewünscht funktioniert und dargestellt wird, werden neben den Quelldateien der Webseiten folgende Skripte / Verlinkungen benötigt:

- *Bootstrap v4.3.1 (JS- & CSS-Datei)*
- *Chart.js v2.8.0 (JS-Datei)*
- *Feather v4.21.0 (JS-Datei)*
- *Bootstrap Confimation v4.0.3 (JS-Datei)*
- *jQuery v3.3.1 (JS-Datei)*
- *Moment v2.24.0 (JS-Datei)*
- *Popper v1.14.7 (JS-Datei)*

Diese müssen in einem Ordner „cdn-fallback“ bei jeder einzelnen Webseite im selben Ordner wie die index.*-Dateien liegen.

2.1.4. Aufbau Code Weboberfläche

Die verschiedenen Tabs der Webseite sind in einzelnen Dateien und Ordnern angeordnet. Die Quelldateien umfassen dabei jeweils mindestens eine HTML-, eine CSS- und eine JS-Datei. Hier wird nur der grobe Aufbau der verschiedenen Skriptdateien aufgezeigt, da sich dort der Hauptteil der Logik abspielt.

2.1.4.1. Dashboard

Folgende Funktionen existieren für das Dashboard:

- **onChange des msg-Objektes:** Prüft bei Eingang einer Nachricht von Node-Red, welche Art Nachricht es ist und wie damit verfahren werden soll. Quasi als „main“-Funktion zu betrachten.
- **on shown.bs.modal:** Wird aufgerufen, wenn das Modal zum Bearbeiten des Gruppennamens aufgerufen wird und speichert, von welcher Gruppe der Aufruf kommt.
- **on submit der formEditGroup:** Sendet den neuen Gruppennamen an Node-Red, wenn das Modal speichernd geschlossen wird.
- **on confirmed.bs.confirmation:** Wird aufgerufen, wenn der Nutzer bestätigt, die Gruppe entfernen zu wollen. Ruft wiederum die Funktion *removeGroupAndItems* auf.
- **removeGroupAndItems:** Sendet den Befehl zum Löschen der entsprechenden Gruppe und der darin enthaltenen Items an Node-Red.
- **generateGrid:** Generiert den HTML-Code für die Platzhalter der Dashboardgruppen und

Items. Ruft *generateGridGroupWithItems* auf.

- **generateGridGroupWithItems:** Erstellt den HTML-Code einer Gruppe und der darin enthaltenen Itemplatzhalter. Ruft dafür *generateGridItems* auf.
- **generateGridItems:** Erstellt den HTML-Code aller Itemplatzhalter einer Gruppe. Ruft dafür *generateGridItem* auf.
- **generateGridItem:** Erstellt den HTML-Code für einen Platzhalter eines Items.
- **fillDashboardWithData:** Füllt die generierten Platzhalter mit Inhalt entsprechend den Daten aus Node-Red.
- **fillEditModal:** Füllt das Modal zum Umbenennen einer Gruppe mit den derzeitigen Werten, wenn es aufgerufen wird.
- **makeChart:** Erstellt aus den gegebenen Datenpunkten einen Graphen.
- **drag:** Wird aufgerufen, wenn ein Item zum Verschieben angefasst wird. Speichert das aufrufende Item.
- **allowDrop:** Überprüft, ob ein Item in ein Gebiet geschoben werden darf.
- **clearDrop:** Macht den Effekt von *allowDrop* rückgängig.
- **drop:** Hauptfunktion des Drag-and-Drop. Verschiebt das Item letztendlich und sendet Node-Red die neue Positionsverteilung zu.
- **updateDropzones:** Lädt die Bereiche zwischen Items, die ein

Ablegen der Items ermöglichen, neu.

2.1.4.2. Device Overview

Folgende Funktionen existieren für diesen Tab:

- **onChange des msg-Objektes:** Prüft bei Eingang einer Nachricht von Node-Red, welche Art Nachricht es ist und wie damit verfahren werden soll. Quasi als „main“-Funktion zu betrachten.
- **fillTable:** Generiert den HTML-Code für die Übersichtstabelle und füllt diesen mit den übergebenen Werten.

2.1.4.3. Device Settings

Folgende Funktionen werden für diesen Tab genutzt:

- **onChange des msg-Objektes:** Prüft bei Eingang einer Nachricht von Node-Red, welche Art Nachricht es ist und wie damit verfahren werden soll. Quasi als „main“-Funktion zu betrachten.
- **on shown.bs.modal:** Wird je aufgerufen, wenn eins der Modals zum Bearbeiten von Sensoren oder Aktoren aufgerufen wird und speichert, von welcher Gruppe der Aufruf kommt.
- **on submit der formSensor / formActuator:** Sendet die neuen Einstellungen an Node-Red, wenn das Modal speichernd geschlossen wird.
- **change des radioManual-Selects:** Sendet die Einstellung zum manuellen Steuern bei einer Änderung direkt an

	Node-Red.
• **click der btnGenerateCSV:**	Sendet Node-Red den Befehl zum Generieren der CSV zu.
• **click der aDownload:**	Verändert den Status des Buttons, dass er ohne erneutes Generieren der CSV nicht noch einmal betätigt werden kann.
• **on confirmed.bs.confirmation:**	Wird aufgerufen, wenn der Nutzer bestätigt hat, den Client zu löschen. Speichert den aufrufenden Client, der entfernt werden soll und ruft *removeDevice* auf.
• **on change selectSensorLink:**	Generiert automatisch die Messgrößen für den ausgewählten Sensor als Dropdown. Ruft dazu *addOptionsSelectMeasurand* auf.
• **generateSensorCardsTemplate:**	Generiert den HTML-Code für einen Sensor als Item.
• **generateActuatorCardsTemplate:**	Generiert den HTML-Code für einen Aktor als Item.
• **fillItems:**	Füllt die generierten Items mit den gegebenen Daten.
• **fillSensorModal:**	Befüllt das Modal für die Sensorkonfiguration, wenn es aufgerufen wird.
• **fillActuatorModal:**	Befüllt das Modal für die Aktorkonfiguration, wenn es aufgerufen wird. Ruft *addOptionsSelectMeasurand* auf.

- **addOptionsSelectMeasurand:** Generiert je nach gegebenem Sensortyp die entsprechenden Einträge in einem Dropdown.
- **removeDevice:** Sendet den Befehl, einen Client aus dem System zu löschen, an Node-Red.

2.1.4.4. MQTT Information

Folgende Funktionen existieren für diesen Tab:

- **onChange des msg-Objektes:** Prüft bei Eingang einer Nachricht von Node-Red, welche Art Nachricht es ist und wie damit verfahren werden soll. Quasi als „main“-Funktion zu betrachten.
- **fillTable:** Generiert den HTML-Code für die Übersichtstabelle der Topics und füllt diesen mit den übergebenen Werten.

2.2. Datenbanken

Alle Daten und Einstellungen, die von der Webseite oder Node-Red benötigt werden, sind in SQLite-Datenbanken abgespeichert.

In den folgenden Kapiteln sind neben den Bezeichnungen der einzelnen Spalten und dem jeweiligen Datentyp ebenfalls beispielhafte Werte in einer dritten Zeile zu finden.

2.2.1. Datenbank reference.db

In dieser Datenbank ist die Referenz für alle bekannten Sensoren und Aktoren hinterlegt.

2.2.1.1. Tabelle sensactref

id	**name**	**types**	**category**	**defaultunits**	**defaultinterval**
INTEGER	*TEXT*	*TEXT*	*TEXT*	*TEXT*	*REAL*
1	DHT22	TH	sensor	Cp	10.0

id: automatisch inkrementierter Zähler, um die Tabelle zu sortieren / Reihen zu identifizieren

name: Name des Sensors / Aktors, der bei einem Discovery vom Client gesendet werden würde

types: codierte Auflistung der Messarten. Bedeutung der Codierung siehe 2.2.3

category: „sensor“ oder „actuator“

defaultunits: codierte Auflistung der Standardeinheiten des Sensors. Bedeutung der Codierung siehe 2.2.4

defaultinterval: Standardintervall, in dem ein Sensor Daten bereitstellen soll, in Sekunden

2.2.2. Datenbank data.db

In dieser Datenbank ist der restliche Teil der gespeicherten Daten abgelegt. Dazu gehören die Tabelle mit den aktuellen bekannten Verbindungen (connections), die Tabellen, in

denen das Aussehen und Anordnung des Dashboards gespeichert sind (uigroups, uilayout), und die einzelnen Log-Tabellen.

2.2.2.1. Tabelle connections

id	IP	refid	category	types	curunit	curinterval
INTEGER	*TEXT*	*INTEGER*	*TEXT*	*TEXT*	*TEXT*	*INTEGER*
1	192.168.1.2	10	sensor	DHT22	Cp	10

Weiter in nächster Zeile:

dispname	status	islogged	isnotifying	showndatapoints	isvisible
TEXT	*TEXT*	*BOOLEAN*	*BOOLEAN*	*INTEGER*	*BOOLEAN*
Bath	connected	1	0	5	1

Weiter in nächster Zeile:

lastmsgtime	mancontrol	linkedto	threshold	linkrule	linkedMeas
INTEGER	*BOOLEAN*	*INTEGER*	*INTEGER*	*TEXT*	*TEXT*
1559309254221	0	10	19	GT	C

Vorsicht: beim Beispiel ab **mancontrol** sind die Werte unabhängig dessen zuvor, da nur für Aktor wichtig!

id: automatisch inkrementierter Zähler, um die Tabelle zu sortieren / Reihen zu identifizieren

IP: IP des Clients

refid: vom System zugewiesene ID, die für die meisten Aktionen benötigt wird

category: „sensor“ oder „actuator“

types: aus reference.db übernommener Name des Sensors

curunit: Derzeitig zum Anzeigen verwendete Einheiten

curinterval: Derzeitiges Sendeintervall des Sensors in Sekunden

dispname: Anzeigename des Clients auf der Weboberfläche

status: „connected“ oder „disconnected“, zeigt an, ob ein Sensor noch erreichbar ist

islogged: momentan ohne Funktion

isnotifying: momentan ohne Funktion

showndatapoints: Legt fest, wie viele Datenpunkte für diesen Sensor auf dem Dashboard pro Messtyp im Verlauf dargestellt werden

isvisible: legt fest, ob ein Sensor auf dem Dashboard angezeigt wird

lastmsgtime: Unixzeit in Millisekunden, wann die letzte Nachricht von dem Sensor empfangen wurde

mancontrol: Zustand der manuellen Steuerung eines Aktors

linkedto: **refid** des verknüpften Sensors, ansonsten leer oder 0

threshold: Schwellwert, bei welchem der Aktor aktiviert werden soll

linkrule: Aktivierung, wenn über dem Schwellwert (GT) oder unterhalb (LT)

linkedMeas: Auswahl, welche Messart eines Sensors mit dem Schwellwert verglichen werden

2.2.2.2. Tabelle uigroups

id	htmlid	name	items
INTEGER	*TEXT*	*TEXT*	*TEXT*
1	1	Erste Gruppe	VC

id: automatisch inkrementierter Zähler, um die Tabelle zu sortieren / Reihen zu identifizieren

htmlid: Nummer der Gruppe auf der Weboberfläche

name: Anzeigename / Titel der Gruppe, wird bei Discovery automatisch mit Name bzw. Typ des Sensors initialisiert

items: codierte Angabe der enthaltenen Items. Anzahl Buchstaben = Anzahl Items, Buchstaben können dabei V (Value) und C (Chart) sein

2.2.2.3. Tabelle uilayout

id	htmlid	numbergroup	numberitem
INTEGER	*TEXT*	*INTEGER*	*INTEGER*
1	val-Temp-10	1	1

id: automatisch inkrementierter Zähler, um die Tabelle zu sortieren / Reihen zu identifizieren

htmlid: Angabe verschiedener Parameter, mit - getrennt:

1. val / chart
2. Code für Messtyp, siehe 2.2.5
3. **refid** des Sensors

numbergroup: Angabe, in welche Gruppe das Item platziert werden soll

numberitem: Angabe, an welche Position das Item in der Gruppe platziert werden soll

2.2.2.4. Tabellen mit Logwerten

Die einzelnen Tabellen folgen dabei dem Namensschema:

- codierter Messwerttyp, siehe 2.2.3
- Trennzeichen **_**
- **refid** des Sensors

Beispiel: *T_15*

Beim Discovery werden für jeden benötigten Messwerttyp eine solche Tabelle erstellt.

id	**timehr**	**timeunix**	**value**
INTEGER	*TEXT*	*INTEGER*	*REAL*
1	2019-05-31T15:27:34	1559309254221	20.3

id: automatisch inkrementierter Zähler, um die Tabelle zu sortieren / Reihen zu identifizieren

timehr: Messzeitpunkt in lesbarer Form

timeunix: Messzeitpunkt in Unixzeit in Millisekunden

value: aufgenommener Messwert

2.2.3. Codierung der Messtypen

Je ein Buchstabe entspricht einer Art Messwert.

Abkürzung	Bedeutung
A	Beschleunigung
B	Taster
C	eCO2
D	Distanz
E	E-Feld / Berührung
G	Gyroskop
H	Feuchtigkeit
L	Lichtstärke
M	Bewegung
O	Spannung (Analogmessung)
P	Luftdruck
T	Temperatur
U	UV
V	VOC
W	Wind

2.2.4. Codierung der Messeinheiten

Je ein Buchstabe entspricht einer Einheit bzw. Suffix für den Messwert.

Abkürzung	**Bedeutung**
a	m/s^2
C	°C
c	Lux
d	°/s
g	ppm
m	cm
N	*Keine Einheit*
P	Pa
p	%
q	ppb
s	m/s
u	Auf UV-Index
V	V
v	Codierte Anzahl erkannter Berührungen

2.2.5. Codierung der Messtypen für Webseite

Hier sind die Typen als Abkürzung codiert.

Abkürzung	**Bedeutung**
Acc	Beschleunigung
Ana	Spannung (Analogmessung)
But	Taster
Co2	eCO2
Dist	Distanz
Efi	E-Feld / Berührung
Gyro	Gyroskop
Hum	Feuchtigkeit
Lum	Lichtstärke
Mot	Bewegung
Press	Luftdruck
Temp	Temperatur
Uv	UV
Voc	VOC
Wind	Wind

2.3. Hinzufügen eines neuen Sensors

Um einen dem System unbekannten Sensor hinzuzufügen, müssen an einigen Stellen im Code der Weboberfläche und von Node-Red Änderungen vorgenommen werden.

2.3.1. Allgemeine Änderungen

2.3.1.1. Änderungen in Datenbanken

In der **reference.db** in der Tabelle **sensactref** muss ein neuer Eintrag für den Sensor erstellt werden. Der dort vergebene Name wird nachfolgend als Sensortyp bezeichnet.

Gemäß den Tabellen zur Codierung der Messarten werden in den weiteren Kapiteln verschiedene Begriffe genutzt: Typen-Abkürzung (z.B. „Temp“) und Zeichen (z. B. „T“).

2.3.2. Änderungen im Quellcode der Weboberfläche

2.3.2.1. Dashboard

Die Änderungen hier müssen in der Datei **index.js** erledigt werden. Suchen Sie die Funktion **fillDashboardWithData**.

Dort muss in dem switch-case am Ende der Funktion ein neues case hinzugefügt werden. In diesem müssen entsprechend die Typen-Abkürzung und die Texte wie in den anderen cases angelegt / geändert werden.

2.3.2.2. Device Settings

Die Änderungen hier müssen in der Datei **index.js** erledigt werden. Suchen Sie die Funktion **addOptionsSelectMeasurand**.

Dort muss in dem switch-case der Funktion ein neues case hinzugefügt werden. In diesem müssen entsprechend der Sensortyp und die Codierung der Messart sowie die ausgeschriebene Form der Messart angegeben werden.

2.3.3. Änderungen bei Node-Red

2.3.3.1. Flow MQTT pubsub

Falls der Sensor nicht über eins der vorhandenen Topics kommuniziert, muss dafür ein Template wie folgt angelegt werden:

In der Funktionsnode wird dabei der gleiche Code verwendet, wie in den bereits vorhandenen Nodes dieses Namens, allerdings mit dem abgeänderten Topic. Der Link am Ende führt an denselben Link wie die bereits vorhandenen Links.

2.3.3.2. Flow Website

Die einzelnen Bereiche sind durch die uibuilder-Nodenamen und Links gekennzeichnet.

Bereich Dashboard

Im Ablauf bei Laden der Seite müssen folgende Funktionsnodes verändert werden:

- **forEach send check if table exists**: Hier muss ein neues else-if hinzugefügt werden. Dieses prüft auf die Typen-Abkürzung und setzt im SQLite-Befehl das Zeichen.

- **forEach send select datapoints**: Hier müssen zweimal ein neues else-if hinzugefügt werden. Diese prüfen jeweils auf die Typen-Abkürzung und setzen im SQLite-Befehl das Zeichen.

Im Bereich, der für das Aktualisieren der Seite zuständig ist, muss Folgendes verändert werden:

- **set measurement type**: Hier bitte ein neues case hinzufügen. Dort wird auf das Zeichen mit folgendem Unterstrich geprüft und dann die Typen-Abkürzung gesetzt.

- **forEach send select datapoints**: Hier müssen zweimal ein neues else-if hinzugefügt werden. Diese prüfen jeweils auf die Typen-Abkürzung und setzen im SQLite-Befehl das Zeichen.

Im Ablauf bei einem Discovery:

- **calc itemCount to add to dashboard**: Sollte der Sensor pro Messgröße mehr als einen Wert ausgeben, muss hier ein else if vor das else gesetzt werden, in dem wie beim vorhandenen if mit dem MPU6050 vorgegangen wird. Statt dem Faktor 3 dort angeben, wie viele einzelne Werte pro Größe existieren.

- **insert new items into uilayout**: Hier bitte ein neues case hinzufügen, welches auf den Sensortyp prüft und die Typen-Abkürzung setzt.

- **send check if table exists**: Hier muss ein neues else-if hinzugefügt werden. Dieses prüft auf die Typen-Abkürzung und setzt im SQLite-Befehl das Zeichen.

- **send select datapoints**: Hier müssen zweimal ein neues else-if hinzugefügt werden. Diese prüfen jeweils auf die Typen-Abkürzung und setzen im SQLite-Befehl das Zeichen.

Bereich MQTT Information

- **generate topic list in mqttData**: Hier bitte ein neues case hinzufügen, welches auf den Clienttyp prüft und die Topics des Clients setzt.

2.3.3.3. Flow CSV generation

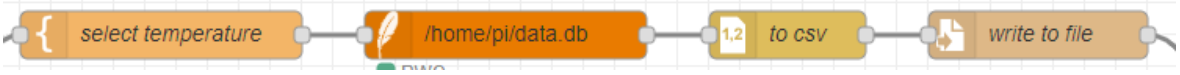
Je nach Anzahl Messgrößen des neuen Sensors muss hier unterschiedlich viel abgeändert werden. Momentan ist der Ablauf für maximal vier verschiedene Messgrößen pro Sensor ausgelegt, kann aber erweitert werden.

Beispiel:
Es soll ein neuer Sensor hinzugefügt werden, der vom Typ „NEU“ ist und Temperatur und Ultraschall messen kann. Die für das System relevanten Kenndaten wären dann der Typ „NEU“, die codierte Form der Messgrößen „TS“ (siehe Tabelle zur Codierung der Messtypen, hier S für Ultraschall angenommen) und die Anzahl der Messgrößen, hier 2.

Es müssen nun folgende Dinge verändert werden:

In den großen „**switch sensorType**“-Nodes muss jeweils der Typ als neue Bedingung eingetragen werden, also „NEU“. Es müssen, solange der Sensor weniger als vier Messgrößen hat, immer in den *ersten n+1 Switchen* der Sensortyp hinzugefügt werden, in diesem Fall also in den ersten drei.

Ist die erste Messgröße dem System bisher noch nicht bekannt, haben also noch keine anderen Sensoren diese Größe, muss eine neue Reihe der den Switchen folgenden Nodes erstellt werden.



Dies kopieren und darunter einfügen und dann folgende Anpassungen in den Nodes vornehmen:

- **select ...**: Namen der Messgröße anpassen und wichtiger: Das Präfix vor {{sensorId}} zu dem codierten Kürzel der Messgröße ändern
- **write to file**: Im Pfad oben den Namen der CSV-Datei entsprechend anpassen

Nun wird die neue Zeile mit dem neuen Ausgang der ersten Switchnode und dem Link am Ende der Zeile verbunden.

Ist die erste Messgröße schon bekannt (wie im Beispiel die Temperatur), kann der neue Ausgang der ersten Switchnode einfach mit der entsprechenden Reihe verbunden werden.

Für die zweite Messgröße wird nun genauso verfahren, allerdings hinter der zweiten Switchnode. Im Beispiel wird hier eine neue Reihe eingefügt, um Ultraschall mit dem Kürzel S hinzuzufügen. Hat der Sensor keine weiteren Messgrößen, wird der neue Ausgang des zweiten Switches mit dem Link direkt hinter dem Switch verbunden.

Hat ein Sensor drei Messgrößen, wird beim letzten Switch der Ausgang direkt mit dem Link am Ende der Zeile verbunden.

2.4. Hinzufügen eines neuen Aktors

Um einen dem System unbekannten Aktor hinzuzufügen, müssen an einigen Stellen im Code von Node-Red Änderungen vorgenommen werden.

2.4.1. Allgemeine Änderungen

2.4.1.1. Änderungen in Datenbanken

In der **reference.db** in der Tabelle **sensactref** muss ein neuer Eintrag für den Aktor erstellt werden. Der dort vergebene Name wird nachfolgend als Aktortyp bezeichnet.

2.4.2. Änderungen bei Node-Red

2.4.2.1. Flow Website

Die einzelnen Bereiche sind durch die uibuilder-Nodenamen und Links gekennzeichnet.

Bereich MQTT Information

- **generate topic list in mqttData**: Hier bitte ein neues case hinzufügen, welches auf den Aktortyp prüft und die Topics des Clients setzt.

3. Firmware

3.1. Aufbau der Firmware

3.1.1. Allgemeiner Aufbau der Firmware

Der Hauptteil der Firmware besteht aus den folgenden Dateien:

main.cpp	WLAN-Verbindung MQTT-Verbindung MQTT-Handshake Hauptroutine
readSensors.h/cpp	Funktionen zum Auslesen der einzelnen Sensoren und Schicken der Werte via MQTT
setActuators.h/cpp	Funktionen zum Ansteuern der Aktoren
controlLed.h/cpp	Funktionen zum Ansteuern der Status-LED

Dazu kommt noch eine Konfigurationsdatei settings.h, in der WLAN-Zugangsdaten und Broker-IP spezifiziert werden können sowie zwischen der Ausgabe von Debug-Nachrichten über die serielle Schnittstelle oder Statusanzeigen via LED umgeschaltet werden kann.

3.1.2. Genutzte externe Bibliotheken

Allgemeine Bibliotheken	
Bibliothek	**Link**
PubSubClient	https://www.arduinolibraries.info/libraries/pub-sub-client
ESP8266Wifi	https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi

Sensorbibliotheken	
Bibliothek	**Link**

3.2. Hinzufügen eines neuen Sensors

Um einen neuen Sensor in die bestehende Firmware zu integrieren sind mehrere Schritte notwendig.

3.2.1. Änderungen in readSensorValues.h

- Festlegen eines neuen Sensornamens und eines entsprechenden #define
    - Achtung: für diesen #define muss ein Wert zwischen 0 und 19 gewählt werden, der noch nicht vorhanden ist
    - Hinweis: Die Nummerierung der DIP-Schalter ist wie folgt zu interpretieren

Nr.	Wert
1	16
2	8
3	4
4	2
5	1

    - Die Summe aller Werte der aktiven DIP-Schalter ist der Wert der in den defines angegeben ist
- Gegebenenfalls vorhandene Bibliothek zum Auslesen des Sensors einbinden

- Anlegen eines Funktionsprototyps zum Auslesen des entsprechenden Sensors mit entsprechenden Übergabeparametern

3.2.2. Änderungen in readSensorValues.cpp

- Implementierung einer Funktion
    - Auslesen des Sensors
    - Ggf. Formatierung des Messwertes zu einem String
    - MQTT-Publish
- Orientierung bieten bereits implementierten Funktionen für andere Sensoren

3.2.3. Änderungen in main.cpp

- Abhängig von der Bibliothek, die zum Auslesen des Sensors verfügbar ist, ist gegebenenfalls eine Initialisierung von Variablen vor der setup() Funktion notwendig
- Änderungen in der setup() Funktion
    - Erweiterung des ersten switch-case um das zusätzlich in readSensorValues.h definierte #define
    - Gegebenenfalls Initialisierung von Variablen (abhängig von gegebener Bibliothek zum Auslesen des Sensors)
    - Generieren der discovery message
        - Achtung: Der übergebene Name muss mit dem in Nodered spezifizierten Discovery-Namen übereinstimmen, siehe hierzu die entsprechende Dokumentation zum Implementieren eines neuen Sensors in Nodered
    - Orientierung bieten die bereits implementierten Sensoren
- Änderung in der loop() Funktion
    - Erweiterung des switch-case um das zusätzlich in readSensorValues.h definierte #define

- Generieren eines Topics pro aufzunehmenden Messwerttyp
  - Achtung: die Topics müssen hierbei mit den in Nodered spezifizierten Topics übereinstimmen. Existiert der gewünschte Messwerttyp noch nicht, so muss hierfür ein neuer Zweig in Nodered angelegt werden. Siehe hierzu die entsprechende Dokumentation
- Kombinieren der vom Broker erhaltenen ID mit zuvor definierten Topics (Orientierung bieten die bereits implementierten Sensoren)
- Aufrufen der Funktion, die in readSensorValues.cpp zum Auslesen des Sensors definiert wurde

3.3. Hinzufügen eines neuen Aktors

Auch für die Implementierung eines neuen Aktors sind einige Schritte notwendig.

3.3.1. Änderungen in setActuators.h

- Festlegen eines neuen Aktornamens und eines entsprechenden #define
  - Achtung: für diesen #define muss ein Wert zwischen 20 und 31 gewählt werden, der noch nicht vorhanden ist
- Gegebenenfalls vorhandene Bibliothek zum Ansteuern des Aktors einbinden
- Anlegen eines Funktionsprototyps zum Ansteuern des entsprechenden Aktors mit entsprechenden Übergabeparametern

3.3.2. Änderungen in setActuators.cpp

- Implementierung einer Funktion zur Verarbeitung des übergebenen Wertes und Ansteuern des Aktors

3.3.3. Änderungen in main.cpp

- Änderungen in der setup() Funktion
  - Erweiterung des ersten und zweiten switch-case um das zusätzlich in setActuators.h definierte #define

- Generieren der discovery message im ersten switch-case
  - Achtung: Der übergebene Name muss mit dem in Nodered spezifizierten Discovery-Namen übereinstimmen, siehe hierzu die entsprechende Dokumentation zum Implementieren eines neuen Sensors in Nodered
- Generieren eines Topics für die Aktoren-Befehle im zweiten switch-case und Abonnieren dieses Topics
  - Achtung: die Topics müssen hierbei mit den in Nodered spezifizierten Topics übereinstimmen. Existiert der gewünschte Messwerttyp noch nicht, so muss hierfür ein neuer Zweig in Nodered angelegt werden. Siehe hierzu die entsprechende Dokumentation
- Orientierung bieten die bereits Implementierten Aktoren

- Änderungen in der commandReceive() Funktion
  - Erweiterung des switch-case um das zusätzlich in readSensorValues.h definierte #define
  - Konvertierung des empfangenen Command-Strings und Aufruf der in setActuators.h/cpp neu hinzugefügten Funktion