Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

József Gergely

Entwicklung eines WiFi-Sensor-Knotens zur Temperatur- und Feuchtigkeitserfassung

Fakultät Technik und Informatik *Faculty of Engineering and Computer Science*

József Gergely

Entwicklung eines WiFi-Sensor-Knotens zur Temperatur- und Feuchtigkeitserfassung

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Mechatronik
an der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr.Ing. Hans Peter Kölzer

Abgegeben am 7. August 2019

József Gergely

Thema der Bachelorthesis

Entwicklung eines WiFi-Sensor-Knotens zur Temperatur- und Feuchtigkeitserfassung

Stichworte

Entwicklung, Messtechnik, Monitoring, Optimierung, Prototypenbau, Softwareentwicklung, Cloud, Server, REST API, WiFi, Sensoren

Kurzzusammenfassung

Diese Arbeit umfasst die Entwicklung und Prototyperstellung eines WiFi-Sensor-Knotens, welcher zur Temperatur- und Feuchtigkeitserfassung eingesetzt wird. Dabei soll ein WiFi-Modul ausgewählt und dafür ein Schaltplan für den Einsatz in Einzelhaushalten und in industriellen Anwendungen erstellt werden.

József Gergely

Title of the paper

Development of WiFi Sensor node for sensing temperature and humidity

Keywords

Development, Measurement, Monitoring, Optimization, Prototyping, Software Development, Cloud, Server, REST API, WiFi, Sensors

Abstract

This work involves the development and prototyping of a WiFi sensor node used for temperature and humidity sensing. A WiFi module is to be selected and a circuit diagram draw up for use in individual households and for industrial applications.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Fertigstellung der Arbeit unterstützt haben. Im erster Linie bei meinem Betreuer Herrn Prof. Dr.Ing. Karl-Ragmar Riemschneider, der immer gute Ratschläge für mich hatte. Aber auch bei meinen Kollegen der Firma “Energiezentrale Nord“ in Norderstedt, die mir durch ihre Unterstützung das Ganze erst ermöglicht haben. Ein besonderer Dank gilt hierbei meinem Betreuer Timo Krei, der mir jederzeit bei Problemen helfen konnte. Besonderen Dank auch an die gesamte Arbeitsgruppe für die freundschaftliche Arbeitsatmosphäre, viele wertvolle Anregungen und stete Hilfsbereitschaft. Auch für die mühevolle Arbeit des Korrekturlesens möchte ich mich herzlich bedanken. Außerdem möchte ich mich bei meiner Familie und Freunden bedanken, die mir in dieser stressigen Zeit Rückhalt gegeben und sich die Zeit genommen haben, diese Arbeit Probe zu lesen.

Inhaltsverzeichnis

1. Einführung

Die Firma Energiezentrale Nord führt Messungen zur Inbetriebnahme und Optimierung von Heizungsanlagen durch. Mit Hilfe von Messtechnik wird die aktuelle Anlagensituation überprüft und maßgeschneiderte Lösungen für einen energieeffizienten Betrieb entwickelt. Die Komplexität der Wärmeerzeugungsanlagen steigt schneller als der Ausbildungsstand. Es werden häufig Wärmeerzeugungstechnologien wie Solar, Blockheizkraftwerk (BHKW), Brennwertkessel oder Wärmepumpen kombiniert, um den Bedarf an Wärme optimal zu erzeugen. Damit die Wärmeerzeugung und -verteilung effizient und reibungslos funktionieren, ist das Monitoring von Anlagenparametern wie Temperaturen, Volumenströmen und Wärmemengen ein elementarer Baustein im Optimierungskonzept. Die moderneren Anlagen sind mit verschiedenen Schnittstellen wie zum Beispiel Mod-Bus, M-Bus, BACNet, KNX und eBus ausgestattet. Dadurch besteht die Möglichkeit, viele Informationen über den Heizungsprozess auszulesen. Für die Optimierung werden noch zusätzliche Temperaturdaten gebraucht, die nicht direkt über die Regelung der Heizungsanlage ausgelesen werden können.

Um diese zusätzlichen Temperaturdaten aufzeichnen zu können, wird eine zusätzliche Messtechnik in die Heizungsanlage eingebaut. Derzeit wird von der Firma Ingenieurbüro H.Lertes GmbH & Co. KG ein Multi-Protokoll Datenlogger/Gateway RmCU 4.0 verwendet [1]. Dieser Datenlogger verfügt über sehr viele Funktionen und Schnittstellen, die nicht benötigt werden. Wenn verschiedene Schnittstellen benötigt werden, ist die RmCU eine gute Lösung. Die meisten Anlagendaten können damit ausgelesen und für die Auswertung bereitgestellt werden. Aber wenn nur zusätzliche Temperaturfühler benötigt werden, werden die Kosten deutlich höher, da der Datenlogger sehr teuer ist.

Aus diesen Gründen stellt sich die erste Frage, ob eine günstige Messtechnik für Anlagen, die kleiner als 100kW sind und das Einsparvolumen sehr gering ist, entwickelt werden kann. Weiterhin ist das Ziel, mit einem selbst entwickelten Datenlogger die zusätzlichen Daten für die Optimierung kostengünstiger und effizienter zu erfassen. Ein weitere sehr wichtiger Aspekt ist wie gut sich das System in eine bestehende Infrastruktur integrieren lässt.

Neben Temperaturen sollen Feuchtigkeitswerte und Gaszählerimpulse erfasst werden. Dazu muss eine S0 Schnittstelle implementiert werden. In dem ersten Teil der Arbeit werden die Anforderungen an die Messtechnik festgelegt. Anhand dieser Anforderungen werden die Komponenten wie Mikrocontroller beziehungsweise Entwicklungsboard, Temperatur- und Feuchtigkeitssensor ausgesucht. Nachdem alle Komponenten ausgewählt wurden, wird ein

Schaltplan entworfen und dazu ein Platinenlayout erstellt. Nach dem Zusammenbau der Hardware wird die Software entwickelt.

Anschließend werden verschiedene Testabläufe geplant, womit alle Funktionen des Datenloggers getestet werden können. Die komplette Entwicklung wird zum Schluss kurz analysiert, die gesetzten Rahmenbedingungen, die Grundkonzeption, auftretende Probleme und wesentliche Folgerungen werden beschrieben. Verbesserungsvorschläge und Ansätze für die Weiterentwicklung werden genannt.

2. Analyse

2.1. Anforderungsanalyse

In diesem Kapitel werden als Erstes die Anforderungen für die Messtechnik definiert und festgelegt. Die bereits verwendete Messtechnik und der Datenausleseprozess werden analysiert. Mit dem Datenlogger werden die Daten für das Anlagemonitoring gesammelt und mit Hilfe eines LTE/UMTS (Long Term Evolution, Universal Mobile Telecommunications System) fähigen Routers, der auch über eine WLAN-Schnittstelle (Wireless Local Area Network) verfügt, an den Datenbankserver gesendet. Alle Daten werden am Server mit einem Zeitstempel versehen und für die weitere Verwendung bereitgestellt. Die Abbildung 2.1 stellt das Gesamtbild von der Datenerfassung bis zur Visualisierung der Energiezentrale Nord GmbH dar.

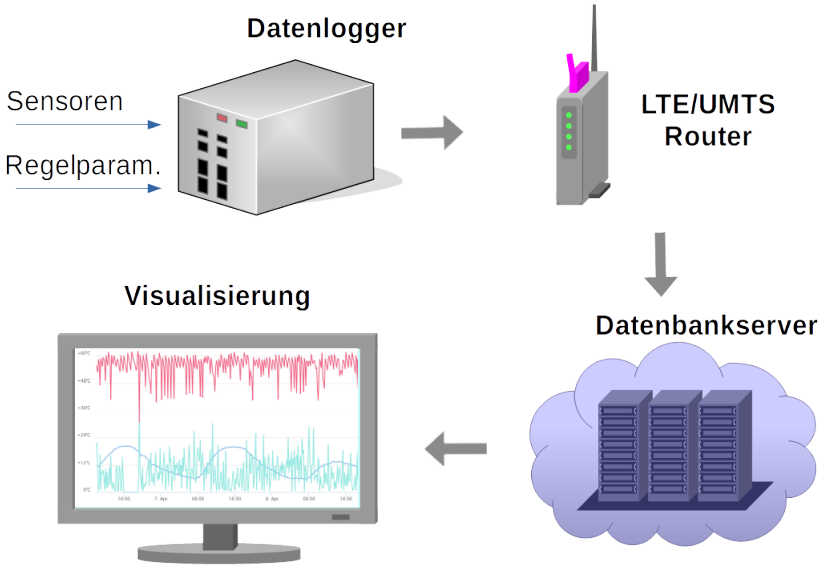


Abbildung 2.1.: Schematische Darstellung der Datenerfassung, Datenübertragung und Visualisierung

Bei dem bisher verwendeten Datenlogger für die Temperaturerfassung wurden analoge Temperaturfühler (PT1000, Platin-Messwiderstand, $R_0 = 1k\Omega$) verwendet. Dazu wurde ein Analog-Input-Modul über die Mod-Bus Schnittstelle des Datenloggers angeschlossen. Es können bis zu 16 Temperatursensoren ausgelesen werden. Falls mehr als 16 Temperatursensoren benötigt werden, können weitere Analog-Input-Module parallel geschaltet werden, was natürlich zu einem erheblichen Kostenanstieg führen würde. Der Datenlogger (RmCU 4.0) hat noch weitere Funktionen und andere Schnittstellen, welche fast nie gebraucht wurden.

Für die flächendeckende Anlagenoptimierung ist es sinnvol eine günstigere Messtechnik für kleine Anlagen mit geringem Einsparvolumen zu entwickeln. Mit der Entwicklung eines günstigeren Datenloggers soll eine wesentliche Kostenreduktion und einfache Bedienbarkeit erreicht werden. Nach der Berücksichtigung dieser Aspekten, wurden durch die Energiezentrale folgende Anforderungen festgelegt:

- Der Datenlogger soll über Funk mit dem LTE/UMTS Router verbunden werden können
- Es sollen bis zu 20 Temperatursensoren angeschlossen werden
- 10m Kabellänge für die Busleitung soll möglich sein
- Ein Feuchtigkeitssensor soll ebenfalls angeschlossen werden können
- Impulse sollen gezählt werden können
- Der Datenlogger soll via Representational State Transfer Application Programming Interface (REST API) die Daten an Server versenden können
- Eine Weboberfläche soll zur einfachen Konfiguration zur Verfügung gestellt werden
- Es soll kostengünstig sein (max 50€)
- Die Weboberfläche soll eine Übersicht der Einstellungen und der angeschlossenen Sensoren geben
- Der Datenlogger soll zuverlässig funktionieren
- Der Datenlogger soll die Sensorwerte bei einem Verbindungsausfall zum Server bis zu 48 Stunden speichern können
- Die gespeicherten Werte sollen mit Zeitstempel gespeichert werden
- Bei der Wiederverbindung soll der Datenlogger gespeicherte Sensorwerte an den Server senden und den Speicher freigeben
- Ein Sensorausfall darf keine Beeinträchtigungen der Funktionalität verursachen
- Der Zusammenbau soll per Hand erfolgen können

- Die Montage und Inbetriebnahme soll möglichst einfach realisiert werden können
- Die verwendeten Komponenten sollen für mehrere Jahre lieferbar sein
- Erfassung der Messdaten im Minutentakt
- Over-The-Air Firmware Update
- 24V Spannungsversorgung
- Einhaltung der EMVG und CE Prüfung erfolgreich bestehen

Bei der Entwicklung soll darauf geachtet werden, dass das Gerät keine Gefahr für den Benutzer birgt. Außerdem darf der Dataprovider nicht durch elektrische oder elektromagnetische Effekte stören oder von anderen Geräten gestört werden. Das bedeutet, dass die elektromagnetische- Ausstrahlung und Störfestigkeit der Dataprovider die im Elektromagnetische Verträglichkeit Gesetz (EMVG) festgelegten Werte nicht überschreiten darf [2]. In dem nächsten Kapitel wird nach geeigneten Komponenten gesucht, die mit der Anforderungsliste übereinstimmen.

2.2. Marktrecherche

Für die Entwicklung eines Datenloggers wird im ersten Schritt ein Mikrocontroller mit Funkmodul gesucht. Da sich der Datenlogger direkt mit dem LTE/UMTS Router über WLAN verbinden soll, soll ein Entwicklungsboard mit WLAN Modul und mit WiFi Funkschnittstelle eingesetzt werden. In der Tabelle 2.1 sind Entwicklungsboards und deren wichtigste Informationen für den Vergleich eingetragen.

Aus der Marktrecherche haben sich die in Tabelle 2.1 aufgelisteten Entwicklungsboards ergeben. Diese Boards erfüllen alle Punkte, die im Anforderungskatalog festgelegt wurden.

Als nächstes werden auf dem Markt befindliche Sensoren betrachtet. Es sind zwei Arten von Sensoren zu unterscheiden. Um physikalische Größen zu erfassen, werden analoge und digitale Sensoren verwendet. Analoge Sensoren übermitteln Strom- oder Spannungssignale. Dabei gibt es sehr viele Einflussfaktoren, die die Messergebnisse beeinflussen können (zum Beispiel Leitungslänge, Störsignale, EMV-Störungen, Genauigkeit der A/D-Auswertemodule, etc). Durch diese Fehlerquellen und Messungenauigkeiten sind in Anwendungsfällen, in denen eine hohe Anforderung an die Genauigkeit gestellt wird, analoge Sensoren nicht gut geeignet. Für die Auswertung analoger Sensoren wird eine recht komplizierte Schaltung (A/D Wandler) benötigt. Bei digitalen Sensoren werden gleich die Temperaturwerte über den Bus nach Anfrage gesendet.

Tabelle 2.1.: Die für die Entwicklung ausgesuchten Entwicklungsboards mit WLAN Modul

Board	WLAN Modul	Takt	Speicher	Besonderheit	Preis
NodeMCU[3, 4]	ESP8266	160 MHz	4 MB	OTA, ADC,...	2.80 €[5]
NodeMCU[6, 7]	ESP32	240 MHz	4 MB	OTA, ADC, 2 Cores, DAC,BLE...	4.55 €[8]
Adafruit Feather M0 WiFi[9, 10]	ATSAMD21 + ATWINC1500	12 MHz	256 KB	No EEPROM	36.40 €[11]
Arduino MRK1000 WiFi[12]	WINC1500	48 MHz	256 KB	Encryption	34.50 €[12]
LinkIt Smart 7688[13]	MT7688AN	580 MHz	32 MB, 128 MB DDR2(RAM)	OpenWRT, C/C++, Python, Node.js	17.73 €[14]

Vorteil der digitalen Sensoren ist, dass alle komplexeren Komponenten, wie A/D Wandler, Look-Up-Table und Kalibrierung direkt in einem Chip vereint sind. Es gibt keine Abhängigkeit der Genauigkeit von der Leitungslänge. Damit kann eine Genauigkeit bis zu 0,05 %, ohne größeren Kostenaufwand erreicht werden [15]. Aus diesen Gründen werden digitale Sensoren ausgewählt und in der Tabelle 2.2 zum Vergleich eingetragen.

Tabelle 2.2.: Die für die Entwicklung ausgesuchten Temperatursensoren

Sensor	Schnittstelle	Bereich	Genauigkeit	Besonderheit	Preis
LMT01[16]	Conter Pulse	$-50°C$ bis $150°C$	$\pm 0.5°C$	-	2.24 €[17]
DS18B20[18]	ONE-WIRE	$-55°C$ bis $125°C$	$\pm 0.5°C$	Wasserdicht	1.99 €[19]
DS1621[20]	2-WIRE	$-55°C$ bis $125°C$	$\pm 0.5°C$	-	3.99 €[19]

In der Tabelle 2.3 werden Sensoren, die sowohl für Temperaturerfassung als auch für die Feuchtigkeitsmessung geeignet sind, zusammengefasst.

Im nächsten Kapitel werden die ausgesuchten Sensoren und Entwicklungsboards im Detail betrachtet. Es werden alle in der Anforderungsliste festgelegten Aspekte betrachtet und es werden die bestmöglichen Komponenten ausgewählt.

Tabelle 2.3.: Die für die Entwicklung ausgesuchten Temperatur- und Feuchtigkeitssensoren

Sensor	Schnittstelle	Bereich	Genauigkeit	Besonderheit	Preis
AM2301/ DHT21[21]	ONE-WIRE	$-40°C$ bis $80°C$ 0% bis 99.9%	$\pm 0.3°C$	Temp & Humidity	6.95 €[22]
DHT11[23]	Single-Bus	$0°C$ bis $50°C$ 20% bis 90%	$\pm 2°C$	Temp & Humidity	2.00 €[24]
DHT22[25]	Single-Bus	$-40°C$ bis $80°C$ 0% bis 100%	$\pm 0.5°C$	Temp & Humidity	4.95 €[26]

2.3. Komponentenauswahl

Für den Prototyp eines Datenloggers wurden verschiedene Sensoren und Entwicklungsboards mit WLAN Modul ausgesucht. Anhand der Anforderungsliste werden die am besten geeigneten Komponenten ausgewählt. Nachdem die Komponenten festgelegt worden sind, kann die Entwicklung des Hardwaredesigns und der Softwareentwurfs beginnen.

Entwicklungsboard

Viele Hersteller wie zum Beispiel Microchip, Silicon Laboratories, Texas Instrument und Espressif bieten zahlreiche WLAN-Module an. Für die Prototyp-Entwicklung wird ein fertiges Entwicklungsboard ausgewählt, da ohne viel Aufwand das Hardwaredesign und die Softwareentwicklung beginnen soll. Alle ausgewählten Boards verfügen über die Standard Hardwareschnittstellen (GPIO's, PWM, I^2C, SPI, etc) und über einen TCP/IP Stack.

Nach der Recherche wurden fünf Entwicklungsboards in Tabelle 2.1 zum Vergleich eingetragen. Das Adafruit Feather M0 WiFi[9] und der Arduino MRK1000 WiFi[12] haben sehr ähnliche Eigenschaften. Das Board von Adafruit hat aber nur 12MHz Taktfrequenz und verfügt über keinen EEPROM. Im Vergleich ist das Bord von Arduino mit 48MHz getaktet. Beide Boards kosten um die 35.00€, welches für die Anwendung zu teuer ist.

Ein weiteres Entwicklungsboard, das LinkIT Smart 7688[13], wurde auch zum Vergleich ausgewählt. Das Board verfügt über 32MB Flash Speicher und zusätzlich über 128MB DDR2 RAM Speicher. Ein minimales Betriebssystem, OpenWRT, ist von Werk aus installiert. Dieses Betriebssystem kann mit den Programmiersprachen C/C++, Python oder Node.js erweitert werden. Es ist ein sehr leistungstarkes Board und bietet sehr viele Möglichkeiten zu einem sehr gutem Preis von rund 18€[14], an. Für den geplanten Anwendungsfall ist dieses aber überdimensioniert.

Die Firma NodeMcu bietet zwei Entwicklungsboards an, das ESP8266[3] WLAN- und das NodeMCU ESP32[6] WLAN-Modulen. Das Board mit dem ESP32 WLAN Modul hat die Besonderheit, dass es zwei Kerne hat und neben WiFi über eine Bluetooth Schnittstelle verfügt. Beide Boards sind Remote-Update-Fähig (Over-the-Air, OTA), welches auch ein sehr wichtiger Aspekt für den neuen Datenlogger ist. Das NodeMCU mit dem ESP8266 WLAN-Modul hat eine Taktfrequenz von 160MHz und verfügt über einen 4MB Flash Speicher. Das Board kostet 2.80€[5] und ist somit am besten für die Anwendung geeignet. Der NodeMCU mit ESP32 WLAN-Modulen wäre die nächstbeste Alternative für 4.55€[8].

Die Firma Espressif ist ein chinesisches, multinationales Halbleiterunternehmen, welches sich auf die Entwicklung von stromsparenden WiFi-Bluetooth-Modulen konzentriert. Der ESP8266 war der erste Schritt auf dem IoT-Markt (Internet of Things). Das NodeMCU ist eine frei verfügbare Firmware für den 32-Bit-Mikrocontroller ESP8266 WLAN-SoC(System on Chip). Es sind mehrere Bauformen und Varianten des ESP8266-SoC auf dem Markt erhältlich. Auf dem NodeMCU-Board ist der ESP-12E verbaut wurden. Das Modul hat eine Taktfrequenz von 80 MHz (maximal 160MHz), verfügt über mehrere Schnittstellen, wie ADC, SPI, I2C, UART, I2S und GPIO-Ports, die für PWM-Signale genutzt werden können. Die Abbildung 2.2 bildet das NodeMCU Entwicklungsboard ab.

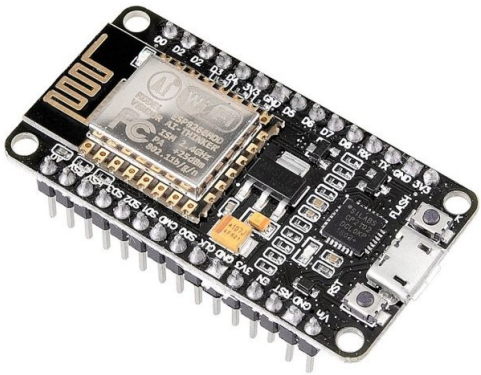


Abbildung 2.2.: ESP8266 NodeMCU Entwicklungsboard [27]

Das besondere an dem ERSP8266 ist, dass es über einen 4MB Flash-Speicher verfügt. Dieser Speicher kann in zwei Bereiche aufgeteilt werden. Ein Teil kann für das Programm und der andere Teil kann als Flash File System (FFS) genutzt werden. Dies ermöglicht es, Konfigurationsdateien oder Inhalte für den Webserver zu speichern[28]. Die nachfolgende Abbildung zeigt das in der Adruino-Umgebung benutzte Flash-Layout von dem ESP8266 WLAN-Modul.

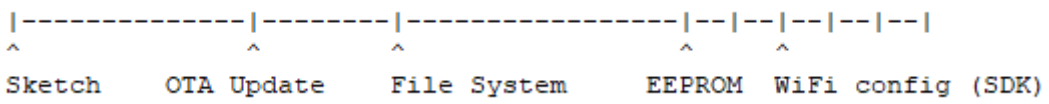


Abbildung 2.3.: Das Speicherlayout von dem ESP8266 Modul[28]

Temperatursensor

In diesem Abschnitt wird ein Temperatursensor für den Datenlogger ausgewählt. Dadurch, dass das Entwicklungsboard nicht über sehr viele GPIO's verfügt, bietet sich ein Bussensor gut an. Der DS18B20 Temperaturfühler passt am besten zu der Anwendung. Es handelt sich um einen One-Wire Bussensor. Dieser Sensor ist in eine Edelstahlhülse eingeschweißt und mit Schrumpfschlauch isoliert. Er wird mit vier verschiedenen Kabellängen ausgeliefert (1m, 3m, 5m, und 10m Kabellänge).

Dieser Temperatursensor kann direkt an dem Datenlogger angeschlossen und am Messpunkt befestigt werden. Die anderen Sensoren sind nicht mit Kabel und wasserdichter Hülse ausgestattet. Aus diesem Grund wird der DS18B20 Temperatursensor ausgewählt, da dieser am besten zur Anforderungsliste passt und auch der günstigste von allen ist. Die Auflösung dieses Sensors kann im Bereich von 9-12Bit eingestellt werden. Die Messgenauigkeit dieser Sensoren liegt bei $\pm 0.5°$C innerhalb von -$10°$C bis $+85°$C[18]. In der Abbildung 2.4 wird der ausgewählte Temperatursensor dargestellt.

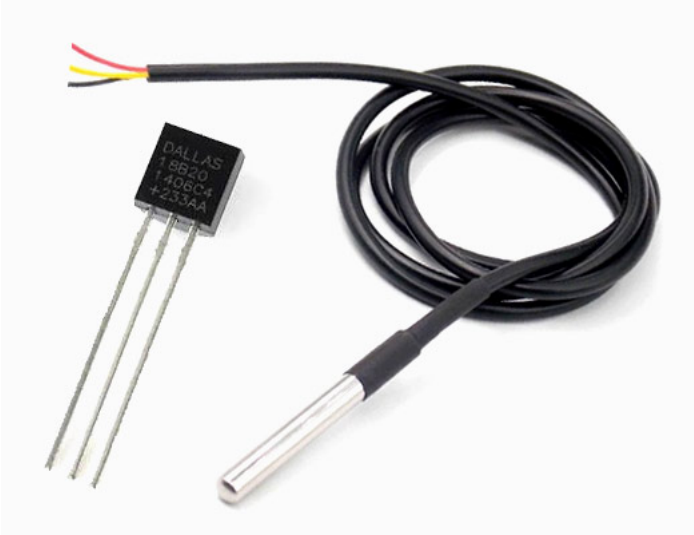


Abbildung 2.4.: DS18B20 Temperatursensor[29](Links der Chip, rechts der komplette Sensor)

Temperatur- und Feuchtigkeitssensor

Bei dem Feuchtigkeitssensor gibt es nicht die Anforderung, dass an den Datenlogger mehrere Sensoren angeschlossen werden sollen. Für alle Anwendungsfälle reicht nur ein Sensor aus, um die Feuchtigkeit der Umgebung zu erfassen. Deswegen wird hier kein One-Wire-Sensor benötigt. Es reicht ein Single-Wire-Sensor vollkommen aus. Der DHT11[23] Feuchtigkeit- und Temperatursensor ist zwar sehr günstig(2.00€[24]), hat aber eine zu geringe Genauigkeit ($\pm 2^{\circ}$C). Als Alternative stehen die Sensoren AM2301/DHT21[21] oder DHT22[25] zur Verfügung. Da der AM2301/DHT21 Sensor rund 7€[22] kostet, wird der DHT22 Feuchtigkeit- und Temperatursensor ausgewählt. In Abbildung 2.5 wird dieser Sensor dargestellt.

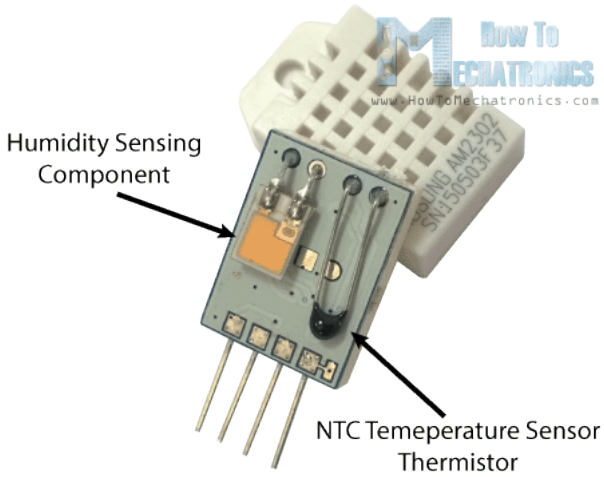


Abbildung 2.5.: DHT22 Temperatur- und Feuchtigkeitssensor[30]

Der DHT22 Sensor verwendet einen NTC-Temperatursensor (Negativer Temperaturkoeffizient (NTC))(oder Thermistor). Auf der Rückseite des Sensors befindet sich ein IC, der die Sensorwerte in digitaler Form bereitstellt[30].

3. Konzept und Design

Im diesem Teil der Arbeit wird die Hardware und das Softwaredesign für den Datenlogger entworfen. Für die praktische Benutzung des Sensorknotens sollte ein einfacher Schaltplan und ein Platinenlayout erstellt werden. Zum besseren Verständnis des Programmablaufs wird eine Programmstruktur erstellt. Diese Schritte werden in den nächsten Unterkapiteln detailliert beschrieben.

3.1. Hardwaredesign

Bis zu 20 Temperatursensoren sollen an dem Datenlogger angeschlossen werden können. Zur verbesserten Ausfallsicherheit werden die 20 Sensoren nicht an einem One-Wire-Bus, sondern auf zwei Busse aufgeteilt. So werden an jedem One-Wire-Bus zehn Temperatursensoren angeschlossen. Die Spannungsversorgung soll über einen 24V Hutschienennetzteil[31] erfolgen. Es wird ein DC-DC Wandler von Traco Power[32] verwendet, um die 5V Versorgungsspannung für den Mikrocontroller bereitstellen zu können. Der Traco Power Spannungswandler hat eine stabile Ausgangsspannung mit einem sehr hohen Wirkungsgrad von 94%. Aus diesen Gründen ist das Bauteil mit seiner kompakten Bauform und seinen Eigenschaften gut für die Anwendung geeignet.

In einem Open Source Programm (KiCad) wird der Schaltplan für den Datenlogger erstellt. Anhand der Datenblätter der Komponenten werden verschiedene Pull-Up Widerstände für die Sensoren benötigt. Für den DHT22 Temperatur- und Feuchtigkeitssensor ist ein $4.7k\Omega$ Pull-Up Widerstand zwischen Spannungsversorgung und Datenleitung erforderlich. Für die One-Wire-Bus Temperatursensoren sind auch $4.7k\Omega$ Pull-Up Widerstände einzusetzen. Wenn aber mehr als 5 Temperatursensoren an einem Bus angeschlossen werden, ist der $4.7k\Omega$ Widerstand zu groß und es bricht die Spannung auf der Busleitung zusammen (und die Sensorwerte können nicht ausgelesen werden). Es wurden ein Reihe von Tests durchgeführt, bei denen sich ergeben hat, dass bei zehn Sensoren und bei einer Buslänge von 10m ein Pull-Up Widerstand von $1k\Omega$ erforderlich ist. Es wird meistens eine Kabellänge von 5m bis 10m für die Busleitung benötigt, da der Datenlogger an sich nicht direkt am Messpunkt verbaut wird. Die Busleitungen werden mittels lötbarer Schraubklemmen an der Leiterplatte befestigt.

Für den S0 Impulseingang wird ein digitaler Input-Pin definiert und ein Anschluss mit lötbarer Schraubklemme zur Verfügung gestellt. Zum Entprellen des S0 Signals ist ein 10pF Kondensator zwischen den S0 Leitungen vorgesehen. Im ECAD Programmpaket KiCad wird der Schaltplan erstellt und in der nachfolgenden Abbildung dargestellt.

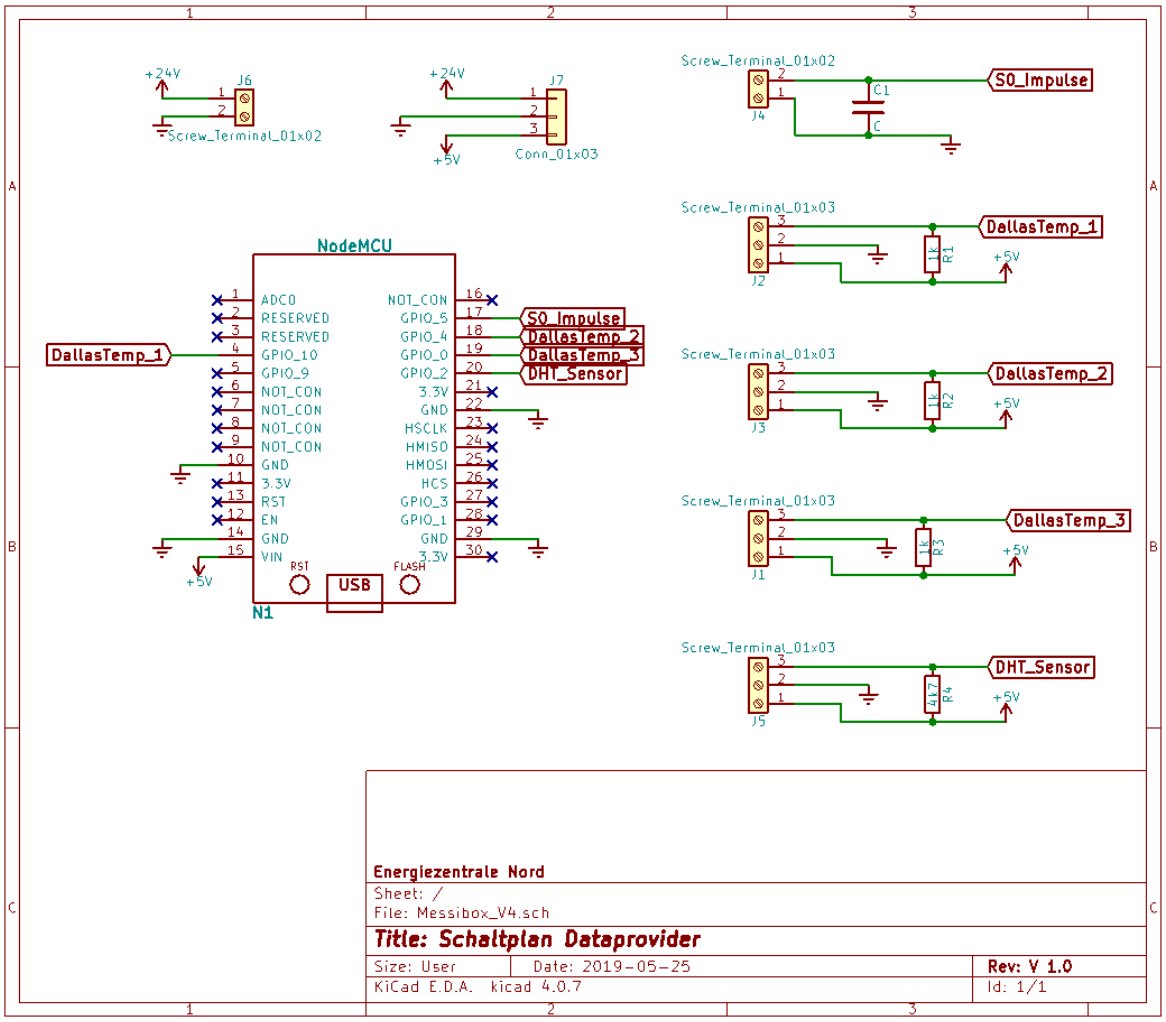


Abbildung 3.1.: Schaltplan des entwickelten Datenloggers mit dem NodeMCU Entwicklungsboard, vier Sensoren, einer S0-Schnittstelle und Spannungsversorgung

Für den Schaltplan in KiCad wurde das NodeMCU Entwicklungsboard als Bauteil abgelegt. Der Footprint des Bauteils wurde ebenfalls als Bibliothek angelegt. Nachdem der Schaltplan entwickelt wurde, wird das Platinenlayout erstellt. Das Platinenlayout wird so dimensioniert, dass es in ein Hutschienengehäuse eingebaut werden kann. In KiCad besteht die Möglichkeit, dass das 3D Modell des Platinenlayouts dargestellt werden kann (siehe Abbildung 3.2).

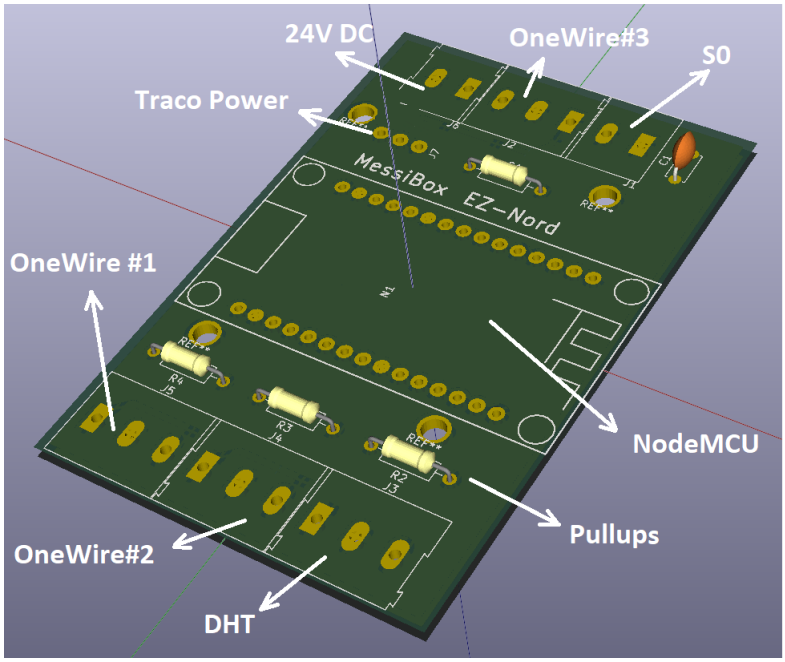


Abbildung 3.2.: Das Platinenlayout des entwickelten Datenloggers mit Beschriftung von Komponenten

3.2. Softwaredesign

In diesem Kapitel soll ein Programm in C/C++ für der Mikrocontroller entwickelt werden. Die Funktionsweise und der Programmablauf werden in diesen Abschnitt festgelegt. Neben Datenerfassung, Aufbereitung und Weiterleitung von Daten soll das Modul über eine Weboberfläche verfügen, worüber die wichtigsten Parameter eingestellt werden können. Die Firmware des Datenloggers soll aus der Ferne aktualisiert werden können, wobei alle Einstellungen erhalten bleiben sollen.

Nach dem Einschalten des Moduls soll die eingestellte Konfiguration geladen und anhand dieser Einstellungen die Parameter gesetzt werden. Weitere Konfigurationen, die zum Programmablauf benötigt werden, werden ebenfalls ausgeführt. Das Modul soll sich in dem vorkonfigurierten WiFi Netz anmelden. Wenn keine Verbindung zu dem Router hergestellt werden kann, soll der Datenlogger trotzdem Messwerte erfassen und speichern können. Das gleiche soll bei einem Verbindungsabbruch passieren.

Wenn eine Verbindung zu dem Router besteht und der Datenbankserver des Unternehmens erreichbar ist, soll zuerst überprüft werden, ob schon Messdaten gespeichert wurden. Falls ja, sollen diese an den Server gesendet werden. Wenn keine gespeicherten Daten vorliegen oder alle gespeicherten Daten gesendet wurden, sollen minütlich die aktuellen Sensorwerte

ausgelesen und an den Server gesendet werden. Außerdem soll der Request der Weboberfläche jederzeit bedient und verarbeitet werden können. Die Abbildung 3.3 zeigt den Ablauf des Programms.

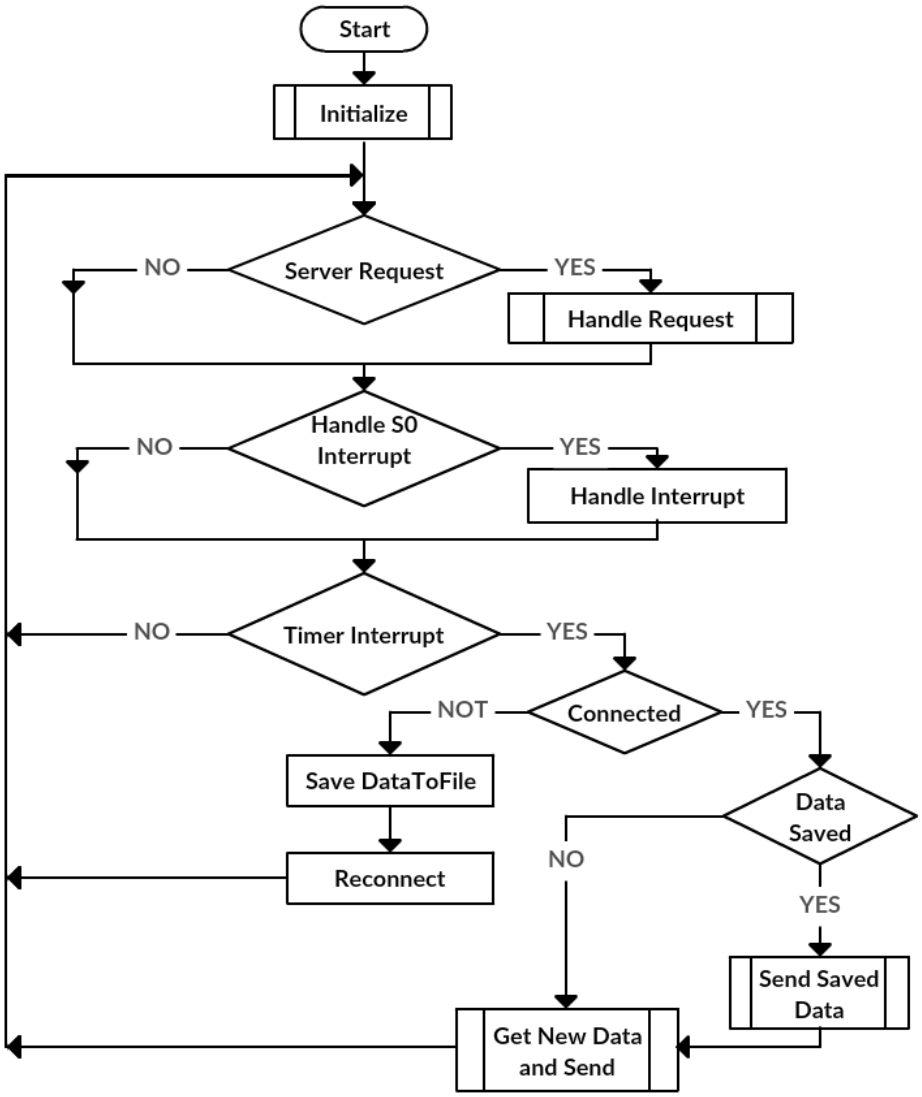


Abbildung 3.3.: Der Programmablauf des Datenloggers

3.3. Webinterface Design

Auf dem ausgewählten WLAN-Modul kann ein Web-Server eingerichtet werden. Dieser Webserver stellt HTML Webseiten für Clients bereit. Mit Hilfe von Webformularen ist es möglich, Benutzereingaben zu erfassen und zu verarbeiten. Diese Daten werden mit den HTML-GET oder HTML-POST Methoden abgesendet[33]. Es wird eine Weboberfläche konzipiert, worüber die Einstellparameter des Mikrocontrollers und die grundlegenden Konfigurationen verändert werden können.

Die Konfigurationsoberfläche wird in vier Hauptgruppen aufgeteilt. Mit einer Tab-Leiste kann zwischen den Hauptgruppen gewechselt werden. In der ersten Hauptgruppe soll eine Übersicht über die aktuellen Einstellungen dargestellt werden. Diese Darstellung soll in drei Kategorien aufgeteilt werden. In der ersten Kategorie werden Geräte-spezifische Informationen dargestellt. In der zweiten Kategorie werden die Netzwerkeinstellungen und als letztes die Sensoreinstellungen aufgelistet. Am Ende der Auflistung besteht die Möglichkeit mit Hilfe einer Schaltfläche den Mikrocontroller neu zu starten.

Unter dem Netzwerk-Tab sollen die Geräte- und die Netzwerkeinstellungen verändert und gespeichert werden können. Die Sensoreinstellungen sollen unter dem Tab Sensoren einstellbar sein. Hier soll zusätzlich zu jedem Sensor eine Bezeichnung hinzugefügt werden können. Unter den letzten Tab soll eine Tabelle mit allen Sensoren, Sensorwerten und zusätzlichen Informationen aufgelistet werden. In der Abbildung 3.4 wird ein Konzept dargestellt, wie die Webseite zum Konfigurieren aussehen soll. Für die einfachere Darstellung sind die Tab-Inhalte unter jedem Tab dargestellt.

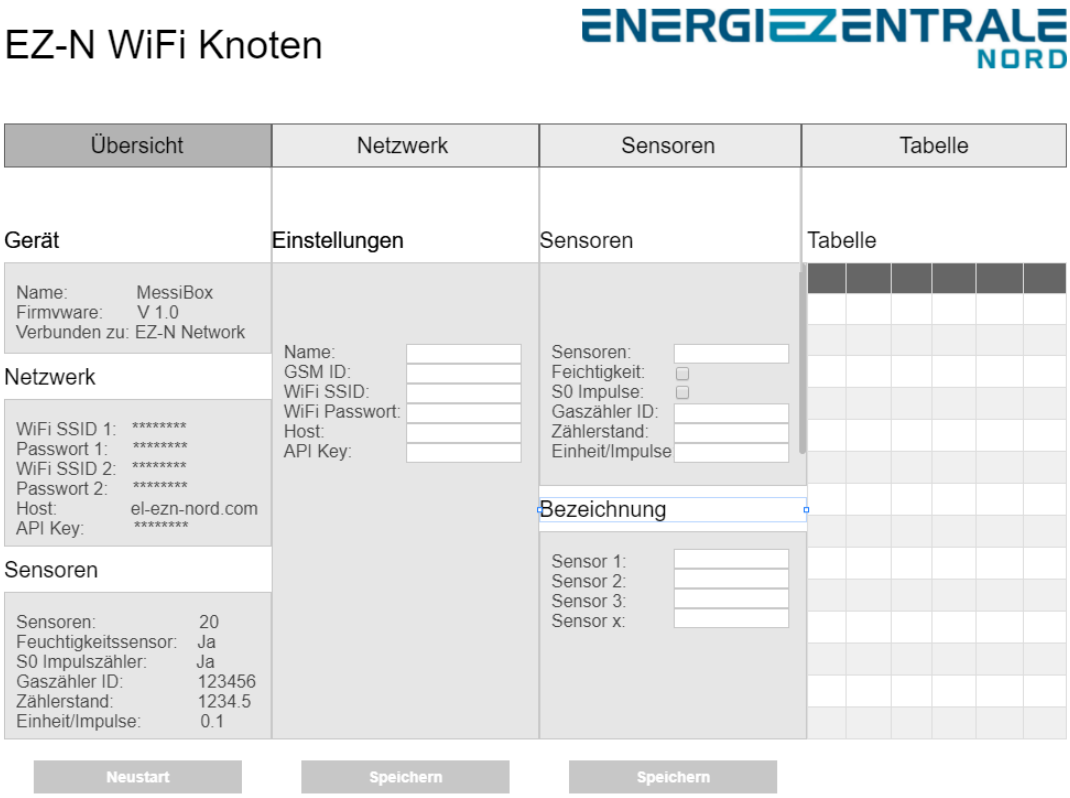


Abbildung 3.4.: Konzept der Weboberfläche des Datenloggers

4. Umsetzung des Konzepts

Im vorherigen Kapitel wurde das Design ausgearbeitet. Das weitere Vorgehen zur Umsetzung wird in diesem Kapitel vorgestellt. Zuerst werden die Hardwarekomponenten bestellt und zusammengebaut. Danach wird eine integrierte Entwicklungsumgebung zum Programmieren des Mikrocontrollers ausgewählt und eingerichtet. Das Programm für den Mikrocontroller wird in Programmiersprache C/C++ und die Weboberfläche in HTML/JavaScript geschrieben.

4.1. Hardware

Die in der Software KiCad erstellten Platinenlayouts wurden bei dem deutschen Unternehmen Aisler [34] bestellt. Aisler bietet komplette Lösungen für Elektronikprojekte von Platinenherstellung, Bauteilen, bis zu SMD-Lötschablonen an. In diesem Projekt wurde das Entwicklungsboard aufgrund des einfacheren Zusammenbaus nur mit wenigen Bauteilen erweitert, deshalb wurden die Komponenten bei anderen Anbietern bestellt. Die fertig bestückte Leiterplatte sieht folgendermaßen aus:

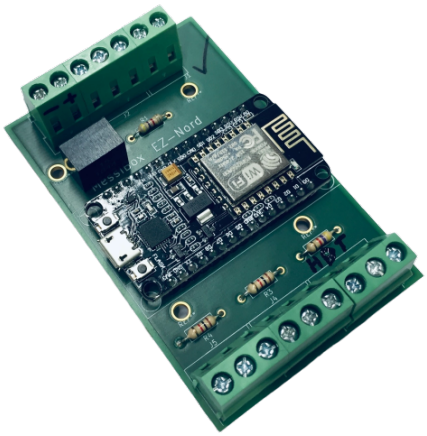


Abbildung 4.1.: Fertig bestückte Leiterplatte

Das Entwicklungsboard, die Widerstände, der DC/DC Konverter und die Schraubenklemmen wurden auf die Platine gelötet. Abschließend wurden alle Kontakte überprüft, ob alle Lötstellen korrekt durchgeführt worden.

4.2. Software für Mikrocontroller

Das NodeMCU Entwicklungsboard ist bereits mit einer Firmware in LUA Programmiersprache von Werk aus programmiert. Das Bord verfügt über eine USB Schnittstelle, Bootloader und kann mit der Arduino Integrated Developmant Environment (IDE) programmiert werden. Da LUA eine sehr seltene Programmiersprache [35] ist, wird das Modul in C/C++ mit der Verwendung des Arduino Frameworks programmiert. Für das ESP8266-Modul sind bereits sehr viele Arduino-Bibliotheken erstellt worden [36]. In der ESP8266 Arduino Core's Dokumentation [37] gibt es eine detaillierte Beschreibung, wie das ESP8266 Modul in die Arduino Umgebung eingebunden und programmiert werden kann.

Die Arduino Umgebung ist ein sehr einfaches und praktisches Tool, welches für einfachere Projekte sehr gut geeignet ist. Aufgrund der Komplexität des Programms wird hier eine andere Entwicklungsumgebung für eine bequeme Entwicklung gewählt. Es gibt mehrere Möglichkeiten, um eine Entwicklungsumgebung für das ESP8266-Modul aufzusetzen. PlatformIO ist ein Open Source, plattformübergreifender Code Builder und Library Manager für Platformen wie Arduino und MBED. Außerdem stellt PlatformIO Toolchains, Debugger und Frameworks bereit und es werden mehr als 600 Boards unterstützt [38].

PlatformIO wurde ursprünglich für die Anwendung über die Kommandozeile entwickelt, welche mittlerweile die Verwendung anderer IDEs wie Eclipse, Visual Studio Code, CLion, NetBeans, etc. ermöglicht. Für die Entwicklung wurde Eclipse CDT (C/C++ Development Tooling) verwendet. Um PlatformIO verwenden zu können, wird Python 2.7 benötigt. Nach der Installation von PlatformIO und Eclipse CDT wurde ein Projekt in PlatformIO erstellt. Dafür musste festgelegt werden, welches Entwicklungsboard beziehungsweise Plattform, welches Framework und welche IDE verwendet werden soll. Ein Projekt wird unter Windows 10 mit dem folgenden Befehl erstellt:

```
C:\User\pathToProjektFolder> platformio init --ide eclipse
--board nodemcuv2
```

Das erstellte Projekt wird in Eclipse als "Existing Project" importiert, das Entwicklungsboard wird via USB am Rechner angeschlossen und die Entwicklung kann beginnen. Im Projektordner ist automatisch folgende Datenstruktur vorhanden:

```
\--data
\--include
\--lib
\--src
     \-main.cpp
     \-...
\--test
\-platformio.ini
```

Im Ordner `data` sind die Dateien, die auf das File System des Mikrocontrollers hochgeladen werden sollen, aufgeführt. Das Serial Peripheral Interface Flash File System (SPIFFS) unterstützt keine Verzeichnisse, sondern speichert nur eine Liste von Dateien. Der Schrägstrich `"/"` ist in Dateinamen erlaubt, im wesentlichen werden nur die Dateinamen gefiltert und beibehalten, die mit dem Schrägstrich beginnen (zum Beispiel: `/config.txt`). Die Dateinamen dürfen nicht länger als 32 Zeichen sein, zusammen mit dem `'\0'` Zeichen, das für die Terminierung von C-Strings reserviert ist [37].

Unter dem Verzeichnis `src` werden `main.cpp` und alle Sourcecode-Files gespeichert. Die Verzeichnisse `test` und `lib` sind für verschiedene Testdefinitionen und “Library Dependencies“ vorgesehen.

Die `platform.ini` Datei beinhaltet die Einstellungen zum Entwicklungsboard [38]. Hier können sehr viele Optionen zum Build, Board, Upload, Monitor, Library, Test und Debug vorgenommen werden. Die drei wichtigsten Einstellungen sind Platform, Framework und Board. In der `platform.ini` Datei sind für das Projekt folgende Einstellungen vorgenommen worden:

```
[env:nodemcuv2]

platform = espressif8266
board = nodemcuv2
framework = arduino

monitor_speed = 115200; Serial Monitor Baud
board_build.f_cpu = 8000000L; MCU frequency
build_flags = -Wl,-Teagle.flash.4m3m.ld; 3MB File System
size
```

Zum Programmieren des NodeMCU-Boards wird das Arduino Framework benutzt und in der `platform.ini` Datei eingetragen. Die Baudrate des seriellen Monitors wird auf 115200 [Symbol/Sekunde] und die Taktfrequenz der Mikrocontroller auf 80MHz eingestellt.

Das ESP8266-Modul kann mit einer maximalen Taktfrequenz von 160MHz betrieben werden, welches für das Projekt nicht unbedingt nötig ist. Mit dem letzten Eintrag in der `platform.ini` Datei kann die Speichergröße des SPIFFS bestimmt werden. In diesem Fall wurden die gesamten 4MB des Speichers auf 1MB für das Programm und 3MB für das File System aufgeteilt [39].

Der in dem Softwaredesign festgelegte Ablauf wird in den nächsten Schritten umgesetzt. Dazu wurden folgende Softwaremodule herausgearbeitet:

- Globale Variablen und Hilfsfunktionen
- Lesen und Speichern von Dateien
- WiFi Verbindung
- NTP Time Server
- Konfiguration und Auslesen von Sensoren
- S0 Schnittstelle
- Webserver
- Main

Als nächstes werden alle Softwaremodule detailliert beschrieben. Das Modul “Webserver“ wird in Kapitel 4.3 separat behandelt. Der Sourcecode für den Mikrocontroller wird im Anhang B und der HTML-Code im Anhang C aufgelistet.

4.2.1. Globale Variablen und Hilfsfunktionen

Viele Variablen werden von mehreren Funktionen und Softwaremodulen verwendet. Der Webserver soll die Eingaben für die Konfiguration weitergeben und speichern können. Für die WiFi-Verbindung werden die Zugangsdaten benötigt und für das Lesen und Speichern von Daten soll auf die Temperaturdaten und auf die Konfigurationen zugegriffen werden können. Deshalb werden solche Variablen global in einem Header File `globals.h` als `extern` deklariert. Alle deklarierten Variablen werden in `globals.cpp` definiert und allen wird ein Defaultwert zugewiesen.

Die Makros und Dateinamen werden als `"const char"` Variablen ebenfalls unter `globals.h` deklariert. Ein Makro für die serielle Ausgabefunktion wird erstellt und wird mit der Definition von `#defien DEBUG` eingeblendet. Ansonsten erfolgt keine serielle Ausgabe im laufenden Programm. Weiterhin werden die Variablen für One-Wire-Bus-, Feuchtigkeitssensoren, Speichern von Daten, NTP Time-Server und für Konfigurationen definiert.

Unter `helpers.h` werden Funktionen für grundlegende Operationen wie Konvertieren einer Reihe von `Strings` in JavaScript Object Notation (JSON) Format, Filenamen anhand der Zählernummer generieren, Taupunktberechnung anhand des Feuchtigkeitswerts oder Lesen und Schreiben von Dateien von/an Serial Peripheral Interface Flash File System (SPIFFS) definiert. Für das Filesystem wird die Bibliothek `#include <FS.h>` verwendet. Dies ermöglicht es, Dateien auf dem Mikrocontroller zu speichern, zu öffnen, zu löschen oder alle gespeicherten Dateien aufzulisten.

Weitere Hilfsfunktionen wie Tabellenerzeugung mit Sensordaten für die Weboberfläche werden ebenfalls hier definiert. Es werden dafür zwei Funktionen geschrieben. Eine für die Erstellung des Tabellenkopfs und eine für die Tabelleneinträge.

4.2.2. Lesen und Schreiben von Dateien

In dem Header File `fileSaver.h` werden die Lese- und Schreibfunktionen für Konfigurationsdatei und für Sensorwerte definiert. Mit der Funktionen `readConfigFile()` oder `writeConfigFile()` wird das File `/my_config_file.json` zum Lesen oder zum Schreiben geöffnet. Der Inhalt kann entweder in Variablen gespeichert oder der Wert der Variablen ins File geschrieben werden. Für das Konfigurationsfile wurde das Dateiformat JSON verwendet. Das Format ist für Datenaustausch zwischen Applikationen entwickelt wurden. Es ist ein schlankes Datenaustauschformat und es ist sowohl für Menschen als auch für Maschinen praktisch zu nutzen. Es ist komplett unabhängig von Programmiersprachen und baut auf zwei Strukturen auf[40]:

- Namen/Wert Paare
- Liste von Werten

In JSON gibt es Objekte, die aus einer ungeordneten Liste von Name/Wert Paaren bestehen. Ein JSON Objekt sieht wie folgt aus:

```
{"Name_1":"Wert_1", "Name_2":"Wert_2", ...}
```

Diese Name/Wert Paare sind einfach zu parsen und zu generieren. Ein JSON Wert kann ein Objekt, ein Array, eine Zeichenkette oder einer der Ausdrücke “true“, “false“ oder “null“ sein. In diesem Projekt wird zum Parsen und Generieren die Bibliothek `<ArduinoJson.h>` verwendet. Die Bibliothek ist sehr gut dokumentiert und ist einfach zu nutzen [41]. Dieses Format wird auch für die Datenübertragung an die Datenbank verwendet.

Der Datenlogger soll die Funktionalität beinhalten, Temperaturwerte zu speichern, wenn der Datenlogger keine Verbindung zum Server hat. Mit der Funktion

`void saveDataToFile(String a_fileName, String a_data)` werden Daten in einem File gespeichert. Als Argumente sollen eine Zeichenkette mit Temperaturdaten übergeben werden und der Dateiname, unter welchem die Datei gespeichert werden soll. Mit der Funktion `String readDataFromFile(String a_fileName)` kann die abgespeicherten Temperaturwerte ausgelesen werden. Sie werden als eine Zeichenkette zurückgegeben. Diese Reihe von Daten können danach zusammen mit anderen Informationen in das JSON Format konvertiert werden.

Die Temperaturdaten werden zusammen mit anderen Dateien wie Konfiguration- und HTML-Dateien für die Weboberfläche im zur Verfügung stehenden Speicherplatz des Filesystems abgelegt. Damit es nicht zu einer vollen Speicherbelegung kommt, werden Temperaturdaten mit 20 Temperatursensoren maximal für 48 Stunden gespeichert. Mit der Funktion `void deletTemperatureFiles()` gibt es die Möglichkeit, alle bisher gespeicherten Temperaturdaten zu löschen. Dies kann über die Weboberfläche mit dem Knopf “Neustart“ ausgeführt werden.

Eine Übersicht aller Temperatursensoren und deren Eigenschaften wird mit der Funktion `createTable()` erstellt. Abhängig davon, wie viele Sensoren an dem Datenlogger angeschlossen sind, wird eine `tabelle.csv` Datei erzeugt und gespeichert. Bei einem Seitenaufruf über die Weboberfläche wird diese Datei direkt als HTML/JavaScript geöffnet und aus dem Inhalt eine HTML Tabelle generiert.

4.2.3. WiFi-Verbindung

Die Internetverbindung wird mit Hilfe der `<ESP8266WiFiMulti.h>` Bibliothek realisiert. Diese Bibliothek hat den Vorteil, dass mehrere Verbindungsstellen angegeben werden können. Der Mikrocontroller wird sich mit der Verbindungsstelle verbinden, bei der das WLAN Signal am stärksten ist. Standardmäßig wird immer der SSID und Passwort von der UMTS/LTE und der Energiezentrale Nord GmbH Router angegeben. Die Zugangsdaten von der Energiezentrale-Router können überschrieben werden. Eine Instanz von `ESP8266WiFiMulti` wird erstellt und mit der Methode `ESp8266WiFiMulti->addAP(const char* ssid, const char* passphrase)` können die Verbindungsstellen angegeben werden.

Mit dem Aufruf der Funktion `bool wifiConnection()` wird die Datei `config_file.json` geöffnet und die dort gespeicherten Parameter in Variablen kopiert. Wenn diese Operation erfolgreich durchgeführt werden konnte, wird der WiFi-Chip in dem Station-Mode gesetzt und die Verbindung zu einem Router aufgebaut. Nach der erfolgreichen Verbindung zu einem Router wird die Verbindungsstelle und die IP Adresse auf dem Seriellen Monitor ausgegeben. Wenn keine Verbindung aufgebaut werden konnte, sollen die Temperaturwerte

in einen Datei gespeichert werden. Mit der Funktion `void wifiReconnect()` soll versucht werden, die Verbindung zu einem Router erneut herzustellen.

Die Funktion `bool isConnected()` überprüft, ob die Datenbank erreichbar ist oder nicht. Es kann vorkommen, dass die Internetverbindung vorhanden ist, aber der Server oder die Datenbank ist nicht erreichbar. In dem Fall sollen die Temperaturwerte ebenfalls gespeichert werden, bis die Verbindung wieder da ist.

4.2.4. NTP Time Server

Das Network Time Protocol (NTP) wird häufig verwendet, um einen Computer, Funkmodul oder Satellitenempfänger mit Internet-Zeitservern zu synchronisieren. Dafür wird ein verbindungsloses Transportprotokoll UDP verwendet [42]. Für das Projekt wird der Internet-Zeitserver benötigt, um gegebenenfalls die Temperaturdaten mit einem Zeitstempel zu versehen.

Die Temperaturdaten sollen minütlich erfasst werden. Wenn sie gleich versendet werden können, wird der Server diese mit dem aktuellen Zeitstempel versehen. Wenn aber alte Daten hochgeladen werden, dann muss der Zeitstempel, an dem sie erfasst wurden, hinzugefügt werden. Der Mikrocontroller verfügt über keine Real Time Clock (RTC) und wenn keine Internetverbindung besteht, ist die aktuelle Zeit nicht abrufbar. Deswegen wird die Zeit in jeder Minute abgefragt, gespeichert und gleichzeitig die Sensordaten erfasst. Wenn bei der nächsten Abfrage die Verbindung schon abgebrochen ist, werden die Sensordaten wie zuvor in jeder Minute erfasst und abgespeichert.

Bei einer Wiederverbindung werden die gespeicherten Sensordaten genommen, der letzte gespeicherte Zeitpunkt um eine Minute erhöht, zu den Sensordaten hinzugefügt und versendet. Der Zeitstempel für jeden Sensorwert muss im richtigen Format am Datenbank-Server ankommen. Der Server akzeptiert folgende Zeitformate:

```
JJJJ-MM-DD HH:MM:SS
```

Der Zeitstempel wird mit der Funktion `String getTimeStemp(time_t a_timeInMS)` in des richtige Format konvertiert.

4.2.5. Konfiguration und Auslesen von Sensoren

Die One-Wire Sensoren-Instanzen wurden bereits in der `globales.h` Datei deklariert. Dafür wurden die Bibliotheken `<OneWire.h>` und `<DallasTemperature.h>` verwendet. Die Funktion `void sensorBegin()` initialisiert alle Temperatursensoren, die an Busleitungen angeschlossen sind. Die Anzahl der Sensoren wird mit der Methode `getDeviceCount()` erfasst und gespeichert.

Wenn die Sensoren bereits initialisiert sind, können die Werte mit dieser Funktion abgefragt werden:

```
String getTemperatures(bool a_connected,
                       uint8_t a_nrSensor,
                       DallasTemperature a_sensor,
                       DeviceAddress a_sensorAddress);
```

Zuerst wird die Sensor-Identifikation ermittelt, die Genauigkeit wird auf $\pm 0.5°$C eingestellt und dann der Sensorwert ausgelesen. Wenn der erste Temperaturwert nach dem Einschalten bei $85.00°$C liegt, wird der Sensor erneut ausgelesen. Wenn die Verbindung zur Datenbank nicht vorhanden ist, werden alle Temperaturwerte in eine Zeichenkette mit dem Komma getrennt zurückgegeben. Wenn aber die Verbindung steht, ist der Rückgabewert der Funktion ein JSON-String, ein Format, dass von der Datenbank angenommen werden kann. Der JSON-String soll für jeden Sensorwert folgende Einträge beinhalten:

```
{
  "TP": "TEMP_1",
  "Tag": "28eff47e17",
  "timestmp": "2019-04-19 16:21:00",
  "Val": "22.5"
}
```

Der Eintrag `"timestmp"` ist optional. Wenn er nicht mitgesendet wird, fügt der Server dies mit der Ankunftszeit hinzu. Der zurückgegebene JSON-String kann mit dem `PUT request` an die Datenbank gesendet werden. Der HTTPS Request wird mit der Funktion `bool request(String a_data)` durchgeführt. Dies ist eine verschlüsselte Verbindung über den PORT 443 zu dem Server. Weitere Authentifizierung wie GSMID und APIKEY sind erforderlich. Diese Zugangsdaten werden von der Visualisierungsplattform EZN Visio bei jeder Anlage vom Dataprovider automatisch erstellt.

Wenn der Server nicht erreichbar ist, werden die Sensorwerte mit den Funktionen `void saveDataToSD()` und `void readDataFromSD()` gespeichert. Da weniger als 3MB Speicher zur Verfügung steht, werden nur die Temperaturwerte ohne Zusatzinformationen abgespeichert, da die sich nicht ändern.Der Zeitstempel kann im Nachhinein hinzugefügt werden. In einer Datei werden 60 Einträge gespeichert. Im jedem Eintrag stehen die Werte aller vorhandenen Sensoren (in der ersten Zeile die Werte aus dem ersten One-Wire-Bus, in der zweiten Zeile die Werte aus dem zweiten One-Wire-Bus).

Der Feuchtigkeitssensor kann mit der Funktion `String getHunidity(bool a_connected)` erfasst werden. Die relative Feuchtigkeit wird mit dem DHT22 Sensor gemessen und daraus kann zusammen mit der Umgebungstemperatur die absolute Feuchtigkeit und die Taupunkttemperatur errechnet werden. Diese Berechnungen werden mit der Funktion `void calcTaupunkt(float* a_humVector)` durchgeführt.

4.2.6. S0 Schnittstelle

Als S0 Schnittstelle wird ein digitaler Input PIN mit Interrupt verwendet. Der Input-Pin wird per Software auf 3.3V gezogen (Internal PULL UP). Mit dem Schließen des Reed-Kontakt Schalters wird der Input-Pin auf Masse gezogen. Jeder Impuls entspricht einer gewissen Anzahl an Energie. In diesem Fall wird meistens ein Impulsnehmer an Gaszählern angeschlossen. Die Impulse pro Einheit sind je nach Zählergröße unterschiedlich.

Um den Zählerwert aktualisieren zu können, sollten die `config_file.json` und die `counter.txt` geöffnet und der Inhalt muss als JSON-String geparst werden. Der Zählerwert kann erst dann erhöht und wieder in die Dateien geschrieben werden. Da dieser Prozess zu lange dauert, wird im “interrupt handler“ nur der `lowEwent`Flag gesetzt und der `eventCounter` erhöht. In einer separaten Funktion wird der `lowEvent` Flag abgefragt, ob sich Impulse ergeben haben und dort die Aktualisierung ausgeführt.

Mit der Funktion `String returnImpulse();` wird der Zählerwert auf das JSON Format konvertiert. Dann kann der Wert an den Server gesendet werden, wie zum Beispiel:

```
{
  "TP": "GAS",
  "Tag": "2347659",
  "timestmp": "2019-04-19 16:21:00",
  "Val": "54389.3"
}
```

Als “Tag“ wird die Gaszähler-ID an den Server gesendet.

4.2.7. Main

Im `main.cpp` File wird der im Softwaredesign festgelegte Ablauf durchgeführt. Nach dem Einschalten wird versucht, die Verbindung zu einem in der `config_file.json` File festgelegten Netzwerk herzustellen. Danach werden die Netzwerkinformationen ausgegeben und alle Initialisierungen durchgeführt. Ein Interrupt mit eine Minute für das Messintervall wird ebenfalls initialisiert.

In der `loop()` Funktion wird als erstes der "Client Handle" für den Webserver ausgeführt. Dabei wird überprüft, ob der Webserver-Request von einem Client ansteht oder nicht. Wenn ja, wird die angeforderte Webseite aus dem File System geöffnet und geladen. Dazu werden die Eingaben mit der HTML-Methoden verarbeitet. Als nächstes wird der Impulszähler aktualisiert, wenn sich in der Zwischenzeit ein oder mehrere Impulse ergeben haben (`updateS0()`).

Nach einer Minute, wenn der `interruptFlag` auf `true` gesetzt wird, wird überprüft, ob Sensordaten bereits gespeichert wurden oder nicht. Wenn eine Verbindung zu dem Server hergestellt werden kann und keine Sensordaten gespeichert sind, werden die aktuellen Temperatursensoren ausgelesen. Die ermittelten Werte werden mit der Funktion `bool request(String a_data)` an den Server gesendet. Falls alte Sensorwerte in der Datei gespeichert sind, werden diese erstmal in der gespeicherten Reichenfolge geöffnet, mit einem Zeitstempel versehen und in dem richtigen Format an den Server geschickt. Falls die Verarbeitung der Daten länger als eine Minute dauert, wird der Prozess unterbrochen und die aktuellen Sensordaten ermittelt und versendet.

Wenn keine Verbindung zu dem Server aufgebaut werden kann, werden die Sensorwerte in der Datei gespeichert. In einer Datei werden 60 Einträge gespeichert, danach wird eine neue Datei erstellt. Die Anzahl der Dateien werden in einer Variablen vermerkt. Die Dateinamen werden anhand der Dateinummer generiert. So können die Dateinummern dynamisch erstellt werden, was den Aufwand zur Speicherung jedes einzelnes Dateinamens erspart.

4.3. Software für Webinterface

Auf dem ausgewählten WLAN-Modul kann ein minimaler Webserver implementiert werden. Der Webserver wird benutzt, um Informationen an den Benutzer über den Datenlogger zu geben und bestimmte Konfigurationen einstellen zu können. Das Webinterface wird in Programmiersprachen Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) und JavaScript erstellt. Die Codedateien werden in dem File System von dem Mikrocontroller hochgeladen. Wenn eine HTML Seite angefordert wird, wird diese Datei geöffnet und der

Inhalt an den Browser gestreamt. In den nächsten Unterkapiteln werden die Funktionalitäten und die Funktionsweise des Webservers beschrieben. Weiterhin wird beschrieben, wie der Informationsaustausch zwischen Weboberfläche und Mikrocontroller, bzw. HTML und C-Code ermöglicht wird.

4.3.1. Aufbau der Webinterface

Die zum Webserver gehörenden Funktionalitäten sind in der `espServer.h` Header Datei festgelegt. Die Weboberfläche besteht aus einer Hauptseite und mehreren Nebenseiten. Sie ist aus folgenden HTML-Files zusammengesetzt:

```
/PAGE_main.html
/PAGE_restart.html
/PAGE_saveSensorConf.html
/PAGE_saveNetworkConf.html
/PAGE_saveDescription.html
/upload.html
/success.html
```

Die `PAGE_main.html` ist die Hauptseite, auf der alle Informationen über den Datenlogger dargestellt sind und alle möglichen Einstellungen vorgenommen werden können. Der Style der Webseiten wird mit CSS am Anfang des Dokuments festgelegt. Die Hauptseite wird in vier Tabs eingeteilt. Unter dem Tab “Übersicht“, siehe Ziffer eins in der Abbildung 4.2, sind alle Informationen über den Datelnogger aufgelistet. In der erste Gruppe werden gerätespezifische Informationen wie Gerätename, eingestellte GSM ID, Firmware-Version, aktuelle Netzwerkverbindung und WLAN-Signal-Stärke angezeigt. In der zweite Gruppe sind die Netzwerkinformationen dargestellt, die WiFi-Verbindungsstellen und die Server Zugangsdaten. In der letzte Gruppe sind Informationen über die angeschlossenen Sensoren dargestellt.

Unter dem Netzwerk-Tab, zwei in der Abbildung 4.2, sind Geräte- und Netzwerkinformationen zu ändern. Unter Sensoren, drei in der Abbildung 4.2, können Feuchtigkeitssensoren und S0-Schnittstellen hinzugefügt werden. Die S0-Schnittstelle kann darüber hinaus noch weiter mit Zähler-ID, aktuellem Zählerstand und Impulsen pro Einheit konfiguriert werden. Eine Sensorübersicht, unter vier in der Abbildung 4.2, wird in dem Tab “Tabelle“ ausführlich dargestellt. Die nachfolgende Abbildung zeigt, wie das Webinterface “Übersicht“-Tab aussieht. Die weiteren Inhalte des Webinterfaces, Nummer zwei, drei und vier werden im Anhang A dargestellt.

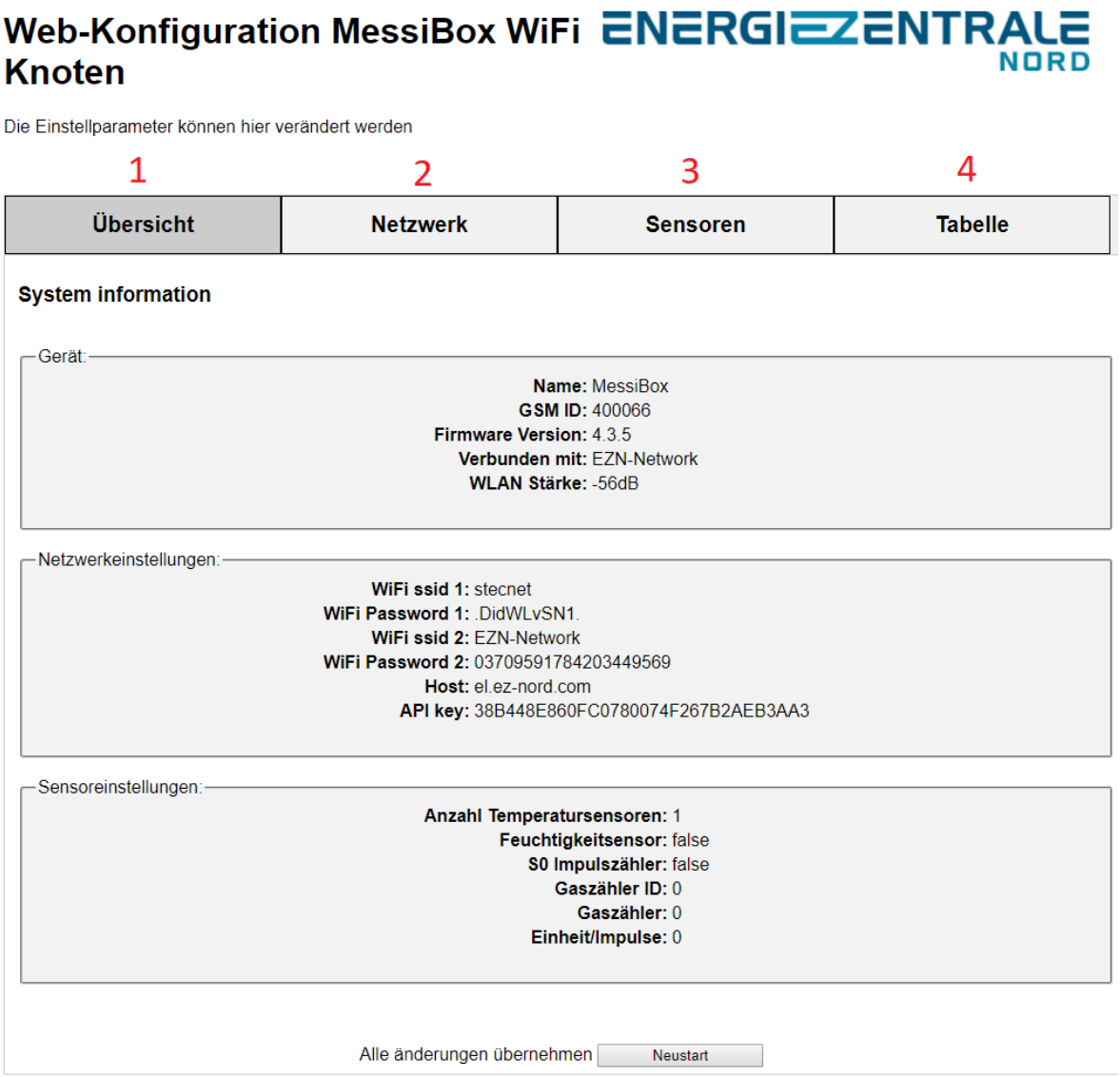


Abbildung 4.2.: Die Übersichtsseite des realisierten Webinterfaces

4.3.2. Handle Requests

Die Funktion `serverHandleClient()` bedient alle Abfragen der Webbrowsern und führt die richtigen Operationen aus. Für das Webinterface auf dem Entwicklungsboard werden die Bibliotheken `<ESP8266WebServer.h>` und `<ESP8266HTTPUpdateServer.h>` verwendet. Bei der Initialisierung soll festgelegt werden, welche HTTP Request Methoden akzeptiert werden und bei welchem Pfad eine Seite geöffnet werden soll. In der Funktion `void initHandleClient()` sind diese

Vorgehen eingestellt. Mit der IP-Adresse von dem Datenlogger, die er bei der Verbindung zugeordnet bekommen hat, wird die Hauptseite geöffnet.

Mit einem Pfad hinter der IP-Adresse können alle Seiten, die im File-System abgelegt sind, direkt aufgerufen werden. Das Remote-Update über den Webbrowser wird ebenfalls hier eingestellt. Es wird ein Benutzername und Passwort wegen der Sicherheit eingerichtet. Mit der Funktion `void handleFileUpload()` können zu dem File-System verschiedene Files hochgeladen werden. Ein Update von dem Webinterface aus ist durch das Hochladen der neuen HTML-Datei möglich.

Mit der Funktion `bool handleFileRead()` werden alle Aufrufe bedient und die eingestellten Parameter gespeichert. Wenn die Hauptseite `/PAGEmain.html` aufgerufen wird, soll die Konfiguration aus der Datei `config_file.json` gelesen werden. Die Sensortabelle wird erstellt und dann wird die Seite mit den aktuellen Informationen gefüllt und geöffnet. Diese Informationen werden vom Mikrocontroller gesammelt und ins File abgespeichert. Der Webserver holt diese Daten direkt aus den Files.

Bei einer Veränderung der Konfigurationsparameter über den Webbrowser, werden die geänderten Daten mittels der HTTP-Methode GET als Server-Argumente an den Mikrocontroller weitergegeben. Diese Argumente werden nach Input-Identifikation sortiert und gefiltert. Bei den Netzwerkeinstellungen werden die zum Netzwerk gehörenden Server-Argumente abgefragt und falls, ein Eintrag vorhanden ist, in Variablen abgespeichert. Das `config_file.json` wird dementsprechend ebenfalls angepasst.

Von der Hauptseite aus kann der Mikrocontroller mit dem Button “Neustart“ neu gestartet werden. Dadurch werden alle gespeicherten Sensordaten und alle zugehörigen Informationen aus dem File-System gelöscht. Ansonsten werden die Daten so lange erhalten, bis sie an den Server gesendet werden.

4.3.3. Speichern von Serverargumenten

Die Kommunikation zwischen Mikrocontroller und Webinterface erfolgt über die HTTP Methode “GET“. In dem HTML File wird als `FORM` mit der Methode “GET“ die eingegebenen Daten weitergegeben und der Nachfolgebefehl eingestellt. Für den Bereich “Netzwerkeinstellungen“ wird die Action `PAGE_saveNetworkConf.html` eingestellt. Die eingetragenen Informationen können mit dem Knopf “Speichern“ an den Mikrocontroller weitergeleitet werden und es wird zur Seite `PAGE_saveNetworkConf.html` gewechselt. Von dort aus kann zurück zur Hauptseite gewechselt werden und weitere Einstellungen vorgenommen werden.

Innerhalb einer HTML `form` werden die Einträge als `input` mit den folgenden Eigenschaften festgelegt:

```
<input type = "text"
       id = "name"
       name = "name"
       size = "20"
       maxlength = "50"
       placeholder = "MessiBox" ></input>
```

Die Methode “GET“ leitet die Input-Einträge weiter. Anhand der `ID` können die Einträge identifiziert werden. Mit der Eigenschaft `size` wird die Input-Feldgröße im Webbrowser festgelegt. Die Länge`maxlength` legt die maximale Länge der Information fest, die der Benutzer eingeben darf. Das ist aus Sicherheitsgründen auf 50 Charakter limitiert. Unter `placeholder` kann ein Beispiel für die Eingabe zum Anzeigen festgelegt werden.

Bei dem Aufruf der Seite `PAGE_saveNetworkConf.html` wird die Funktion `void saveNetworkConfig()` ebenfalls aufgerufen und die Einträge werden gespeichert. Bei den Sensoreinstellungen wird der Seite `PAGE_saveNetworkConf.html` aufgerufen, damit diese hier eingegebenen Einträge mit der Funktion `void saveSensorConfig()` gespeichert werden können. Für die anderen Parameter wird die dazugehörige HTML-Seite und Funktion zum Abspeichern von Daten ausgeführt. Durch die Aufteilung wird nur das veränderte Segment geprüft und wenn Änderungen vorliegen, gespeichert.

4.3.4. Handle File Upload

Die Funktion `bool handleFileUpload()` verwaltet die Files, die hochgeladen werden. Der Filename wird als erstes ermittelt, wenn ein File zum Hochladen ausgewählt ist. Das Zeichen Schrägstrich “/“ wird am Anfang des Filenamens hinzugefügt, falls der nicht vorhanden ist. Ein File im File-System wird mit diesem Filenamen erstellt und die empfangenen Bytes werden ins File geschrieben. Wenn der Prozess erfolgreich abgeschlossen werden konnte, wird der Seite `success.html` aufgerufen. Auf dem seriellen Monitor wird der hochgeladene Filename und die Größe angezeigt.

4.3.5. OTA Software Update

Da die Datenlogger an verschiedenen Orten aufgebaut werden, ist es sehr wichtig, Fernupdates durchführen zu können. Bei dem Modul ESP8266 besteht die Möglichkeit, einen so-

genannten Over-The-Air (OTA) durchzuführen. Für diese Funktionalität wird die Bibliothek `<ESP8266HTTPUpdateServer.h>` [43] verwendet. Dadurch kann ein sicheres Webupdate mit Authentifizierung durchgeführt werden. Benutzername, Passwort und ein Pfad kann eingestellt werden, welcher aufgerufen werden soll, um die kompilierte Binärdatei hochzuladen. Wenn die Datei erfolgreich übertragen werden konnte, wird sie auf dem Mikrocontroller gespielt. Nach dem Neustart wird die neue Firmware gestartet und damit das Update erfolgreich durchgeführt.

5. Testplan und Durchführung

In diesem Kapitel handelt es sich um die Erstellung und Durchführung von Tests für den entwickelten Datenlogger. Ein Test kann auf zwei unterschiedliche Weisen betrachtet werden. Als erstes mit dem Ziel, Fehler im System zu finden und zum zweiten um zu zeigen, dass das System tut was es soll [44]. So wird sichergestellt, dass die Systemimplementierung den Systemspezifikationen entspricht.

Der Datenlogger wird als Hauptsystem betrachtet. Zu dem Hauptsystem gehören mehrere Subsysteme, die mit dem Hauptsystem agieren. Die Funktionalitäten und die Zusammenhänge zwischen Haupt- und einzelnen Subsystemen sollen getestet werden. Der Test soll beweisen, dass das System alle Anforderungen erfüllt und reibungslos funktioniert. Dabei sollen Sondersituationen und kritische Stellen mit Sorgfalt behandelt werden. Fehler sollen entdeckt, dokumentiert und korrigiert werden. Zum Schluss werden alle Testergebnisse analysiert und die daraus folgenden Erkenntnisse dargestellt.

Ein systematischer Test besteht aus [44]:

- Randbedingungen definieren
- Eingaben systematisch auswählen
- Ergebnisse dokumentieren und nach Kriterien beurteilen

Die ISO/IEC/IEEE 29119 definiert einen international vereinbarten Satz von Normen für Softwaretest. Die Norm liefert eine Einführung und Beispiele für die Praxisanwendung. Konzepte und Definitionen im Bereich Softwaretest sind definiert [45]. In dem Testvorgang sollen folgende Kriterien betrachtet werden [46]:

- Erkennen von Analyse- und Designfehlern
- Erkennen von Implementierungsfehlern
- Bestimmen des Produktstatus
- Abnahmevortest

Der Zweck des Testplans ist ein strukturiertes Vorgehen, um Fehler so früh wie möglich im System zu erkennen und die verfügbare Zeit sowie Kapazitäten optimal zu nutzen [47]. Das System und die Systemgrenzen sollen definiert werden. Abbildung 5.1 stellt das betrachtete System dar.

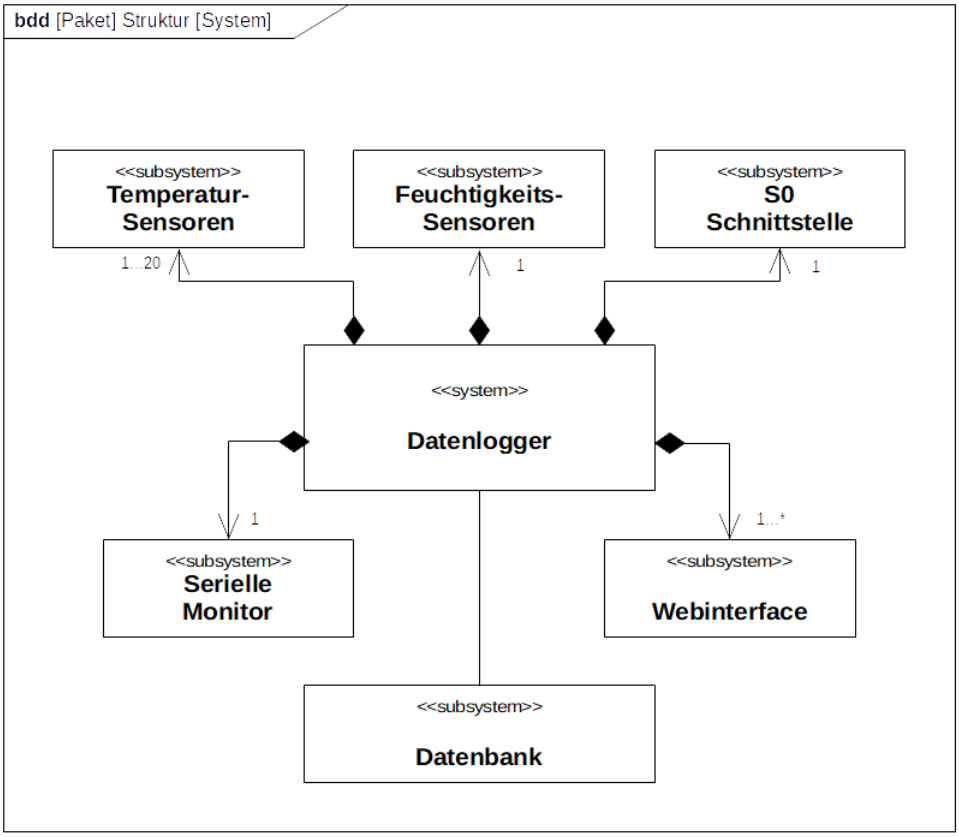


Abbildung 5.1.: Zu testende Systemübersicht des Datenloggers mit den dazu gehörenden Funktionsgruppen

Bei einer äußeren Betrachtung des Systems wird die Benutzeroberfläche unter die Lupe genommen. Fehlerhafte Bedienungsabläufe oder Bedienungsprobleme weisen auf einen Design- oder Analysefehler hin. Die Bedienungsabläufe sollen dem in Anforderungen festgelegten Kriterien entsprechen, ansonsten werden sie als Fehler betrachtet [46].

Programmabstürze, zum Beispiel durch eine Benutzereingabe, deuten auf einen Implementierungsfehler hin. Es sollen möglichst viele Variationen ausprobiert werden [46]. Die Kompatibilität zwischen HTML Code und C Programm soll hierunter überprüft werden, da Daten zwischen den Programmiersprachen ausgetauscht werden.

Der Produktstatus wird durch zwei Arten von Testen bestimmt. Dem sogenannten “Alphatest“ werden die einzelnen Funktionalitäten des System unabhängig voneinander getestet. Für

den Alphatest soll das komplette Produkt nicht fertiggestellt werden. Durch diese Art von Test, werden in Abschnitt 5.2 die funktionalen Anforderungen getestet.

Durch einen “Betatest“ wird das gesamte System unter Realbedingungen geprüft [46]. In dem Abschnitt 5.3 soll die volle Funktionalität des System vorhanden sein und getestet werden.

Der Abnahmevortest ist der letzte Test. Damit soll die korrekte Funktionalität des Systems nachgewiesen werden. Unter anderem werden die Bereiche Installationsumgebung, Inbetriebnahme, Vollständigkeit und Korrektheit der Dokumentation ebenfalls überprüft [46].

Als nächstes wird für jeden Testabschnitt ein Testplan erstellt. Zu jedem Testabschnitt wird eine passende Testtechnik ausgewählt und durchgeführt.

5.1. Komponententests

In diesem Kapitel werden die Hardwarekomponente und die dazu gehörigen Schnittstellen getestet. Hauptsächlich die One-Wire-Busleitung für die Temperatursensoren wird in mehreren Hinsichten getestet. Der Feuchtigkeitssensor ist mit Single-Wire-Bus an dem Datenlogger, mit einem maximal 1m langen Kabel, angeschlossen.

5.1.1. Länge der Busleitung

An dem Datenlogger sollen bis zu 20 Temperatursensoren angeschlossen werden. Im Heizungsanlagen liegen häufig die Messpunkten weit auseinander. Aus diesem Grund wurden die 20 Sensoren auf zwei Schnittstellen aufgeteilt.

Im nächsten Schritt wird ein Testplan für die Ermittlung der maximale Leitungslänge für die Bussensoren erstellt. Es wird systematisch vorgegangen, erstmal fünf und dann zehn Temperatursensoren, mit unterschiedlichen Leitungslängen an dem Datenlogger angeschlossen. Die zum überprüfende Längen sind 5m, 10m, und 15m.

Die Angaben und die Messergebnisse werden in einer Tabelle Zusammengefasst. Es wird überprüft, ab welche Anzahl von Sensoren und ab welcher Leitungslänge die Temperatursensoren auf dem One-Wire-Bus nicht mehr auslesbar sind. Die nachfolgende Tabelle zeigt

die Testparameter und die daraus resultierenden Ergebnisse.

Tabelle 5.1.: Leitungslängentest für die One-Wire-Bus Sensoren

Anzahl der Sensoren	Leitungslänge	Ermittelbar
5	5m	JA
5	10m	JA
5	15m	JA
10	5m	JA
10	10m	JA
10	15m	NEIN

Wie in der Tabelle 5.1 zusammengefasst wurde, sind bis auf 10m Leitungslänge zehn Temperatursensoren auslesbar. Bis auf fünf Sensoren kann die Leitungslänge auf 15m erhöht werden. Bei der Länge sind die Sensoren auslesbar. Ab zehn Temperatursensoren auf 15m Busleitung sind zwar die Sensoren erkennbar, aber nicht auslesbar. Ein Grund dafür könnte eine Datenkollision auf der Busleitung sein.

5.1.2. Anzahl der Temperatursensoren

In diesem Abschnitt wird die maximale Anzahl der Temperatursensoren, die an einer Busleitung angeschlossen werden können, getestet. Da die Anforderungen für die Leitungslänge 10m entspricht, wird ein 10m lange Leitung genommen und durch Ausprobieren immer mehr Sensoren angeschlossen.

Der Test wird mit fünf Sensoren angefangen. Die Anzahl der Sensoren werden einzeln erhöht. Nach jedem Temperatursensor, der zu der Busleitung hinzugefügt wird, soll überprüft werden, ob die Sensordaten ausgelesen werden können. In der Tabelle 5.2 werden diese Messschritte und die Messergebnisse eingetragen.

Aus der Tabelle 5.2 wird ersichtlich, dass bei einer Leitungslänge von 10m nicht mehr als zwölf Temperatursensoren auslesbar sind. Aus Sicherheitsgründen wird der Anzahl von Sensoren auf zehn festgelegt. Die Software wird dementsprechend so implementiert, dass pro One-Wire-Bus bis zu zehn Sensoren angeschlossen und dynamisch ausgelesen werden können. Der PULL-UP Widerstand zwischen Versorgungsspannung und Datenleitung wurde auf 1kΩ reduziert.

Tabelle 5.2.: Ermittlung der maximalen Anzahl von Temperatursensoren auf 10m Leitungslänge

Leitungslänge	Anzahl der Sensoren	Auslesbar
10m	5	JA
10m	6	JA
10m	7	JA
10m	8	JA
10m	9	JA
10m	10	JA
10m	11	JA
10m	12	JA
10m	13	NEIN
10m	14	NEIN

Mit der Anpassung des PULL-UP Widerstands kann die Anzahl noch eventuell erhöht werden. Dabei soll aber darauf geachtet werden, dass bei dem Temperatursensor nicht zu hohe Spannungsverluste auftreten. Dafür müssen weitere Tests durchgeführt und die Software an mehreren Stellen angepasst werden. In dieser Arbeit ist das nicht vorgesehen.

5.1.3. Sensortest

Mit dem Sensortest ist die Genauigkeit des Temperatursensors zu überprüfen. In diesem Teil der Arbeit werden die ermittelten Temperaturwerte mit anderen Messgeräten verglichen. Zum Vergleich wird der Datenlogger von der Firma Trotec GmbH and Co.KG [48], das Modell DL200P eingesetzt (Siehe Abbildung 5.2). Das Messgerät kann in vielen Bereichen zum Monitoring eingesetzt werden. Neben Temperatur und Luftfeuchtigkeit kann das Gerät mit externen Messgrößenaufnehmern erweitert werden [48].

Für den Vergleich wird kein externer Aufnehmer benötigt. Der Temperatur- und Feuchtigkeitswert wird mit dem gemessenen Wert der entwickelten Datenlogger verglichen. Die Ergebnisse werden in Tabelle 5.3 eingetragen.

Der Trotec Datenlogger hat eine Auflösung von 0,1°C. Der ausgewählte Temperatursensor DS18B20 hat zwar eine Auflösung von 0,1°C, aber die Genauigkeit ist 0,5°C. Aus diesen Gründen wurde die Auflösung vom Temperatursensor auf 0,5°C eingestellt.

Aus dem Vergleich ist ersichtlich, dass alle drei Temperatursensoren nahezu den gleichen Wert anzeigen. Bei der Luftfeuchtigkeit ist der Unterschied ein wenig größer, liegt aber

Abbildung 5.2.: Der für den Temperaturvergleich verwendete Datenlogger

Tabelle 5.3.: Vergleich der Temperatursensoren

Thermometer/Sensor	Temperatur	Feuchtigkeit
Trotec DL200P	24.9 °C	53.4 %
DHT22	24.6 °C	57.7 %
DS18B20	24.5 °C	-

trotzdem innerhalb der Toleranz.

Die Abbildung 5.3 stellt die Tabellenübersicht des Datenloggers mit zehn Temperatursensoren und mit einem Temperatur- und Feuchtigkeitssensor dar. Alle Sensoren zeichnen die Raumtemperatur und Raumfeuchtigkeit auf. So kann festgestellt werden, ob alle Temperatursensoren den gleichen Wert anzeigen. Von zehn angeschlossenen Sensoren zeigen zwei 25.00°C an. Ein Sensor zeigt 24.00°C und alle anderen zeigen 24.50°C an. Diese Messung entspricht der in den Sensordatenblättern versprochenen ± 0.5°C Toleranz.

Die Messung wurde öfters durchgeführt, die Unterschiede zwischen den Sensorwerten waren bei jeder Messung gleich. Dies erfüllt die am Anfang festgelegten Spezifikationen.

Im Anhang befinden sich mehrere Tabellenübersichten des Webinterfaces.

NR	Knoten	Sensor-ID	Wert	Einheit	TP	Bezeichnung
1	400066	183373b0ff	24.50	°C	TEMP_1	
2	400066	18337988ff	24.50	°C	TEMP_1	
3	400066	18335534ff	25.00	°C	TEMP_1	
4	400066	1833c1ecff	24.00	°C	TEMP_1	
5	400066	1833bf79ff	24.50	°C	TEMP_1	
6	400066	16c23da5ff	25.00	°C	TEMP_1	
7	400066	1833bf03ff	24.50	°C	TEMP_1	
8	400066	18337433ff	24.50	°C	TEMP_1	
9	400066	18330a4bff	24.50	°C	TEMP_1	
10	400066	16c02c6bff	24.50	°C	TEMP_1	
11	400066	DHT22AM220	57.70	%	TEMP_1	Relative Luftfeuchte
12	400066	DTT22AM230	24.50	°C	TEMP_1	Temperatur
13	400066	DHT22AM240	15.44	°C	TEMP_1	Taupunkttemperatur
14	400066	DHT22AM250	13.11	g/m^3	TEMP_1	Absolute Luftfeuchte

Abbildung 5.3.: Mit dem Datenlogger erfasste Temperaturdaten und Feuchtigkeiten

5.2. Funktionstest

Der entwickelte Datenlogger besteht aus mehrere Softwarekomponenten. Die Funktionalitäten der einzelnen Module werden nacheinander getestet. Als erstes wird der Verbindungsaufbau vom Datenlogger zu dem Router überprüft. Danach werden die Serverfunktionen und das Webinterface getestet. Zum Schluss werden Erfassung, Speicherung und Senden von Daten überprüft.

Ein Testplan wird für jeder Funktion erstellt und durchgeführt. Der Ablauf der zu behandelnden Funktion wird kurz beschrieben. Es wird überprüft, ob das getestete Teil des System das macht, was gemacht werden soll und es wird nach Fehler gesucht. Nach der Durchführung von Tests wurden die Ergebnisse dargestellt. Die Folgen der Ergebnisse werden analysiert und bewertet.

5.2.1. Internetverbindung

Nach der Inbetriebnahme der Datenlogger wird als erstes die WLAN Verbindung hergestellt. Die Bibliothek `<ESP8266WiFiMulti>` wird benutzt, um gleichzeitig mehrere Verbin-

dungsstellen angeben zu können. Der Datenlogger baut die Verbindung zu dem Router auf, bei dem das WLAN Signal am stärksten ist. Mit dem nachfolgenden Programmabschnitt wird dieser Test ausgeführt:

```
WiFi.mode(WIFI_STA);
multi.addAP("ssid_1", "passwort_1");
multi.addAP("ssid_2", "passwort_2");

uint8_t i;
for (i = 0; (i < 20 && multi.run() != WL_CONNECTED); i++) {
 delay(1000);
}
```

An der Stelle der "ssid_1/2" und "passwort_1/2" sind die Verbindungsdaten anzugeben. In der for-Schleife wird entweder 20 Sekunde lang oder bis die Verbindung aufgebaut ist, gewartet. Wenn die WLAN Reichweite ausreichend ist und die Authentifizierung erfolgreich durchführt wurde, wird die Verbindung aufgebaut. Bis auf eine Reichweite von -95dBm ist die Verbindung stabil. Wenn die Verbindung steht, werden die Temperaturdaten erfasst und an die Datenbank gesendet.

Die Internetverbindung wird minütlich überprüft. Wenn sie nicht besteht, speichert der Datenlogger die Temperaturdaten in das File System. Danach wird versucht, die Verbindung wieder herzustellen. Falls immer noch keine Internetverbindung vorhanden ist, werden die Sensorwerte weiterhin in das File System gespeichert.

Bei der Durchführung des Tests werden folgende Schritte betrachtet:

- Router ist Offline, keine Verbindung ist möglich
- Router ist Online, Internet vorhanden
- Der Datenlogger verbindet sich mit dem Router
- Router geht Offline, Verbindung geht verloren
- Router geht wieder Online,

In der Tabelle 5.4 werden die Testschritte zusammengefasst und die Testergebnisse eingetragen.

Im ersten Fall gibt es keine Internetverbindung. Da der Datenlogger über keine Hardware Real Time Clock (RTC) verfügt, kann die Uhrzeit nicht ermittelt werden. So werden die

Tabelle 5.4.: Testdurchführung zur Internetverbindung des Datenloggers

Nummer	Testschritt	Aktion	Testergebnis
1.	Router ist Offline	Datenlogger geht in Speichermodus	Erfüllt
2.	Router ist Online	Datenlogger baut Verbindung auf	Erfüllt
3.	Router geht Offline	Datenlogger geht in Speichermodus	Erfüllt
4.	Router geht Online	Datenlogger baut Verbindung auf	Erfüllt

Temperaturdaten mit einem Zeitstempel von 01.01.1900 versehen.

Es wird davon ausgegangen, dass bei der Inbetriebnahme des Datenloggers eine Internetverbindung aufgebaut werden kann. Wenn einmal die Internetverbindung steht, wird die Unix Zeit abgefragt. So kann bei weiteren Internetausfällen der Zeitstempel sekundenweise hochgezählt werden.

In den Fällen, dass wenn keine Internetverbindung vorhanden ist und der Datenlogger neu gestartet wird oder die Spannungsversorgung für kurze Zeit unterbrochen wird, werden die Temperaturwerte mit falschem Zeitstempel versehen und gespeichert. Nach einer Wiederverbindung werden diese Werte an die Datenbank gesendet, wird aber unbrauchbar. Die Wahrscheinlichkeit, dass dieser Fall auftritt ist, sehr gering. Alle anderen Szenarien erfüllen die Anforderungen.

5.2.2. Verbindung zum Datenbankserver

Wenn die Verbindung zu dem Router aufgebaut ist und die Internetverbindung steht, kann überprüft werden, ob eine sichere Verbindung über den Port 443 zu dem Datenbankserver hergestellt werden kann. Für die Verbindung wird eine sogenannte GSM ID und API KEY benötigt. Dies wird über die Visualisierungsplattform (EZN Visio) durch Erstellung eines neuen Dataproviders zur Verfügung gestellt.

Vorausgesetzt, dass eine Internetverbindung vorhanden ist, werden dafür folgende Szenarien überprüft:

- Verbunden über UMTS/LTE Router
- Verbunden über das Netzwerk der Energiezentrale
- Client meldet sich bei dem Server an

- Client stellt einen HTTP PUT Request an Server
- Warten auf die Serverantwort

Die Durchführung und die Testergebnisse werden in die Tabellen 5.5 und 5.7 zusammengefasst.

Tabelle 5.5.: Testdurchführung, Verbindung zum Server über UMTS/LTE Router

Nummer	Testschritt	Aktion	Testergebnis
1.	Client meldet sich an	Verbindung angenommen	Erfüllt
2.	HTTP PUT Request	-	-
3.	Server antwortet	Request angenommen	Erfüllt
4.	Datenformat akzeptiert	Returncode 200	Erfüllt

Über den UMTS/LTE Router kann der Datenbankserver problemlos erreicht werden. Es kann vorkommen, dass aus unterschiedlichen Gründen der Server nicht erreichbar ist. In dem Fall reicht die Überprüfung der Internetverbindung nicht aus um sicherzustellen, dass die Temperaturdaten an die Datenbank gesendet werden können.

Am Anfang jedes Zyklus muss die Verbindung zum Datenbankserver anstatt der Internetverbindung geprüft werden. Für Verbindung zum Server ist eine Internetverbindung Voraussetzung. Das hat Einfluss auf die zuverlässige Funktionalität des Datenloggers und der Quellcode wird dementsprechend angepasst.

Tabelle 5.6.: Testdurchführung, Verbindung zum Server über das Netzwerk der Energiezentrale

Nummer	Testschritt	Aktion	Testergebnis
1.	Client meldet sich an	Verbindung angenommen	Nicht erfüllt
2.	HTTP PUT Request	-	-
3.	Server antwortet	Reuest angenommen	Nicht erfüllt
4.	Datenformat akzeptiert	Returncode 200 (OK)	Nicht erfüllt

Die Kommunikation mit dem Server über das Netzwerk der Energiezentrale Nord funktioniert grundsätzlich nicht. Es kommt mal vor, dass die Verbindung für kurze Zeit mit dem Server hergestellt werden kann. Die Verbindung kommt unregelmäßig zustande und ist nicht reproduzierbar. Nach kurzer Analyse konnte nicht festgestellt werden, was die Ursache für diesen Fehler ist. Da des EZ-Network nur während der Entwicklung benutzt wird, wird in dieser Arbeit keine weitere Fehleranalyse erfolgen.

5.2.3. File System

Das File System auf dem Entwicklungsboard ist ein sehr wichtiges Element des Datenloggers. Die Konfigurationsfiles, HTML-Files und historische Temperaturdaten werden darauf abgelegt. Es ist sehr wichtig, dass das Schreiben und Lesen von Daten zuverlässig funktioniert. Dazu müssen folgende Schritte überprüft werden:

- Files zum Schreiben und Lesen öffnen
- Konfigurationsfile lesen
- Konfigurationsfile schreiben
- Temperaturtabelle zur Übersicht schreiben
- Temperaturtabelle zur Übersicht lesen
- Temperaturdaten schreiben
- Temperaturdaten lesen
- S0 Zählerstand schreiben
- S0 Zählerstand lesen
- HTML-Files zum Lesen öffnen

In der Tabelle 5.7 werden die Durchführung und die Ergebnisse zusammengefasst.

Tabelle 5.7.: Testdurchführung, Lesen und Schreiben von Daten in das File System des Datenloggers

Nummer	Testschritt	Aktion	Testergebnis
1.	Files zum schreiben öffnen	Datei erfolgreich geöffnet	Erfüllt
2.	Files zum lesen öffnen	Datei erfolgreich geöffnet	Erfüllt
3.	Konfigurationsfile lesen	Datei erfolgreich geöffnet	Erfüllt
4.	Konfigurationsfile schreiben	Datei erfolgreich geöffnet	Erfüllt
5.	Temperaturtabelle schreiben	Datei erfolgreich geöffnet	Erfüllt
6.	Temperaturtabelle lesen	Datei erfolgreich geöffnet	Erfüllt
7.	Temperaturdaten schreiben	Datei erfolgreich geöffnet	Erfüllt
8.	Temperaturdaten lesen	Datei erfolgreich geöffnet	Erfüllt
9.	S0 Zählerstand schreiben	Datei erfolgreich geöffnet	Erfüllt
10.	S0 Zählerstand lesen	Datei erfolgreich geöffnet	Erfüllt

Das Lesen und Schreiben von Dateien auf das File System wurde getestet. Während des Tests hat alles funktioniert, es konnten keine Fehler festgestellt werden. Diese Funktionalitäten wurden bereits während der Entwicklung kontinuierlich getestet und die aufgetauchten Fehler korrigiert.

5.2.4. Webinterface

Das Webinterface des Datenloggers dient in erster Linie für eine Übersicht über die Einstellungen und über die angeschlossenen Sensoren sowie Schnittstellen. Als zweites lassen sich die Einstellungen über die Menüpunkte “Netzwerk“ und “Sensoren“ verändern. Weitere, sehr wichtige Funktionen des Webinterface sind die Firmware Update-Funktion und das Hochladen von verschiedenen Dateien in das File System des Mikrocontrollers.

Alle Funktionalitäten des Webinterface werden systematisch getestet. Im ersten Schritt wird den Aufruf der Hauptseite überprüft. Danach werden die Tab-Funktionen und die Input-Einträge getestet. Die Einträge werden in die Konfigurationsdatei geschrieben.

Der Upload, Firmware Update und Neustart der Mikrocontroller wird als nächstes überprüft. In der Tabelle 5.8 sind die Testschritte und die Ergebnisse zusammengefasst.

Tabelle 5.8.: Testdurchführung zum Testen des Webinterface

Nummer	Testschritt	Aktion	Testergebnis
1.	Aufruf der Hauptseite	Darstellung im Webbrowser	Erfüllt
2.	Wechsel zwischen Tabs	Darstellung im Webbrowser	Erfüllt
3.	Input Netzwerk	Input in Variablen speichern	Erfüllt
4.	Eintrag speichern	Konfigurationsdatei aktualisieren	Erfüllt
5.	Input Sensoren	Input in Variablen speichern	Erfüllt
6.	Eintrag speichern	Konfigurationsdatei aktualisieren	Erfüllt
7.	Input Bezeichnung	Input in Variablen speichern	Erfüllt
8.	Eintrag speichern	Datei für Bezeichnung erstellen	Erfüllt
9.	Aktualisierte Übersicht	Darstellung im Webbrowser	Erfüllt
10.	Tabellendarstellung	Darstellung alle Sensoren	Erfüllt
11.	Upload Dateien	Upload Success	Erfüllt
12.	Firmware Update	Update Success	Erfüllt
13.	Neustart	Gespeicherte Sensordaten löschen	Erfüllt

Während der Entwicklung des Webservers wurden viele Tests durchgeführt. Der Datenaustausch zwischen den Anwendungen funktioniert reibungslos. Die eingestellten Parameter

werden in JSON-Dateien im File System abgespeichert. Diese Dateien beinhalten die Benutzerinformationen für die Authentifizierung zum Router, Datenbankserver und die Freischaltung der Schnittstellen (Feuchtigkeitssensor und S0 Schnittstelle). Das Einlesen der Konfigurationsdatei und das Parsen von JSON-Dateien wurde erfolgreich durchgeführt.

Die oben genannten Funktionen wurden getestet. Das Hochladen von Dateien in das File System wurde öfter durchgeführt und war erfolgreich. Bei einem Firmware Update ist eine zusätzliche Authentifizierung erforderlich. Nach einer erfolgreichen Authentifizierung können die kompilierten Binärdateien hochgeladen werden. Sobald die Binärdatei vollständig hochgeladen ist, wird der Update-Prozess durchgeführt. Der Datenlogger wird neu gestartet und die aktualisierte Software wird geladen. Der Test hat gezeigt, dass ein Firmware Update erfolgreich durchgeführt werden kann.

5.2.5. Auslesen von Temperatur- und Feuchtigkeitswerte

Der Datenlogger verfügt über zwei One-Wire-Bus Eingänge für digitale Temperatursensoren und eine Single-Wire-Bus für Feuchtigkeit- und Temperatursensoren. An jedem One-Wire-Bus können bis zu zehn Temperatursensoren angeschlossen werden. Im Kapitel 5.1 wurde teilweise diese Funktion mit anderen Kriterien getestet.

In diesem Abschnitt wird überprüft, ob alle 20 Sensoren und der Feuchtigkeitssensor ausgelesen und in das richtige JSON-Format (siehe in Kapitel 4.2) konvertiert werden kann. Die Sensordaten werden nur in einem bestimmten Format vom Datenbankserver angenommen. In der Tabelle 5.11 sind die Testschritte und die Ergebnisse zusammengefasst.

Tabelle 5.9.: Testdurchführung zum Auslesen von Sensoren

Nummer	Testschritt	Aktion	Testergebnis
1.	Erste Bus, 10 Sensoren	alle 10 Sensorwerte erfassbar	Erfüllt
2.	Zweite Bus, 10 Sensoren	alle 10 Sensorwerte erfassbar	Erfüllt
3.	Beide Busse, 20 Sensoren	alle 20 Sensorwerte erfassbar	Erfüllt
4.	Feuchtigkeitssensor	Feuchtigkeit erfassbar	Erfüllt
5.	JSON-Format	Ausgabe der Werte in JSON	Erfüllt

Die Testergebnisse haben gezeigt, dass alle Sensoren, die am Datenlogger angeschlossen wurden, erfassbar sind. Die Ausgaben auf dem Seriellen Monitor haben gezeigt, dass die Sensorwerte, zusammen mit den zusätzlichen Informationen, in das richtige Format konvertiert werden. An dem Punkt wurde bei der Entwicklung viel getestet, damit bei einer späteren

Fehlersuche die Komplexität vereinfacht wird. Dadurch kann diese Stelle als Fehlerquelle ausgeschlossen werden.

5.2.6. S0 Schnittstelle

In den Heizungsanlagen sind die Gaszähler mit Impulsausgang zur digitalen Erfassung der Zählerstände ausgestattet. Bei den Datenloggern wurde eine S0-Schnittstelle implementiert, um diese Impulse erfassen zu können. Der Zählerstand soll so gespeichert werden, dass er bei einem eventuellen Stromausfall, Softwareupdate oder bei einem Neustart des Mikrocontroller erhalten bleibt. Aus diesen Gründen wird der Zählerstand in das File System als Datei abgelegt.

Die Impulse werden mit einem GPIO-PIN durch Interrupt erfasst. Da der Zählerstand im Datei abgespeichert ist, dauert es lange, bis der Wert aktualisiert werden kann. In dem Interrupt-Händler wird nur die Anzahl der Interrupts gezählt. Mit einen Flag wird gemerkt (`true`), dass sich ein oder mehrere Interrupts ergeben haben. Dieses Flag wird periodisch abgefragt. Wenn Impulse entstanden sind, wird der Zählerstand aktualisiert.

Bei einem Impuls sollen folgende Aktionen ausgeführt werden:

- Der Datei soll geöffnet werden
- Alter Wert soll in Variablen übertragen werden
- Die Anzahl der Impulse werden mit der Wertigkeit multipliziert (Einheit/Impulse, m^3/imp)
- Es wird auf dem nlten Wert drauf addiert
- Der aktuelle Zählerstand wird in der Datei aktualisiert

Die oben beschriebenen Schritte wurden getestet und die Ergebnisse in der Tabelle 5.10 zusammengefasst. Im Testfall soll ein Impuls erkannt und der Zählerstand aktualisiert werden. Der Prozess der Aktualisierung soll überprüft werden.

Der Test hat gezeigt, dass die Aktualisierungsschritte erfolgreich durchgeführt werden konnten. Ein weiterer Test wurde in der Heizungsanlage der Energiezentrale Nord eingerichtet. Ein Reed-Kontakt Schalter wurde am Gaszähler mit einer 3m langen Leitung montiert und das Signal am Datenlogger angeschlossen.

Tabelle 5.10.: Testdurchführung, Funktionalität der S0-Schnittstellen

Nummer	Aktion	Testergebnis
1.	Datei wird geöffnet	Erfüllt
2.	Wert wird in Variablen geschrieben	Erfüllt
3.	Impulse mal Wertigkeit	Erfüllt
4.	Variable aktualisiert	Erfüllt
5.	Zählerstand in Datei aktualisiert	Erfüllt

Die Gaszählerimpulse wurden eine Woche lang erfasst. Nach einer Woche wurde der digital erfasste Verbrauch mit dem Zählerstand der Gaszähler verglichen. Beide Zähler haben den gleichen Verbrauchswert angezeigt.

5.2.7. Sensordaten an Datenbankserver versenden

Die Verbindung zum Datenbankserver wurde bereits mit einen Dummy-Wert getestet. Jetzt wird versucht, erfasste Sensordaten an den Datenbankserver zu senden. Dazu werden alle Sensoren angeschlossen und die S0-Schnittstelle freigeschaltet. Der Request zum Datenbankserver wird gestellt und die Sensordaten gebündelt in JSON Format versendet. Die Serverantwort wird gespeichert und analysiert. Die durchgeführten Schritte und die Ergebnisse werden in der Tabelle 5.11 zusammengefasst.

Tabelle 5.11.: Testdurchführung, Sensordaten vorbereiten und an den Datenbankserver versenden

Nummer	Testschritt	Aktion	Testergebnis
1.	Sensordaten vorbereiten	JSON String	Erfüllt
2.	Sichere Verbindung zum Server	Über den Port 443	Erfüllt
3.	Autentifikation	GSM ID und API KEY	Erfüllt
4.	Sensordaten versenden	Request absenden	Erfüllt
5.	Antwort der Server	200 (OK) ?	Erfüllt

Falls der entwickelte Datenlogger mit dem UMTS/LTE Router verbunden ist, kann eine Verbindung aufgebaut und die Sensordaten können versendet werden. Wie schon bei dem Test im Kapitel 5.2.2 festgestellt wurde, kann keine Verbindung mit dem Datenbankserver von dem EZ-Netzwerk hergestellt werden. Die Ursache des Problems wird in dieser Arbeit nicht weiter analysiert. Über den UMTS/LTE Router, der für die Anwendungen vorgesehen ist, können Daten an den Server versendet werden.

Wenn die Daten vom Datenbankserver angenommen wurden, sieht die Antwort wie folgt aus:

```
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, must-revalidate
Content-Type: application/json
Date: Fri, 14 Jun 2019 13:22:36 GMT
Pragma: no-cache
Server: Apache/2.4.39 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 1113
Connection: Close

[{"status":"ok","GSM_ID":400368,"ArrivalTime":"2019-06-14
13:22:36","TP":"TEMP_1","Tag":"DHT22AM220","Val":52.1},...
```

Somit sind alle Teilfunktionen getestet worden. Manche Fehler wurden schon bei der Entwicklung entdeckt und behoben, da während der Entwicklung sehr viel getestet wurde. Die Fehler, die bei dem Funktionstest entdeckt wurden, wurden analysiert und die, die Funktionalität des Datanloggers beeinträchtigt haben, wurden behoben. In dem nächsten Kapitel werden weitere Tests, die die Funktionalität des Gesamtsystems überprüfen, durchgeführt.

5.3. Systemtest

Bis jetzt wurden nur die Teilfunktionen des entwickelten Systems getestet. Nachdem die einzelnen Komponenten den Test bestanden haben, soll das Zusammenspiel dieser Teile im Rahmen von Integrationstests überprüft werden.

5.3.1. Integrationstest

Der Integrationstest ist einer der letzten Schritte die bei dem entwickelten Datenlogger durchgeführt werden müssen. Es wird überprüft, wie die einzelnen Module zusammenpassen und die Anforderungen erfüllen. Der im Kapitel 3.2 festgelegte Programmablauf wird in der Abbildung 3.3 überprüft und alle möglichen Ablaufzweige ausprobiert.

In der Tabelle 5.12 werden alle Schritte und die möglichen Wege beschrieben. Die Ergebnisse werden in Form “Erfolgt“ oder “Nicht Erfolgt“ ebenfalls in der Tabelle eingetragen.

Tabelle 5.12.: Testdurchführung, aktuelle und historische Sensordaten an den Datenbankserver versenden

Nummer	Schritt	Aktion	Folgeschritt	Testergebnis
1.	Initialisierung	-	Server Request	Erfüllt
2.	Server Request	JA	Handle Request	Erfüllt
3.	Handle Request	-	Handle S0 Interrupt	Erfüllt
4.	Server Request	NEIN	Handle S0 Interrupt	Erfüllt
5.	Handle S0 Interrupt	JA	Handle Interrupt	Erfüllt
6.	Handle Interrupt	-	Timer Interrupt	Erfüllt
7.	Handle S0 INterrupt	NEIN	Timer Interrupt	Erfüllt
8.	Timer Interrupt	NEIN	Server Request	Erfüllt
9.	Timer Interrupt	JA	Connected	Erfüllt
10.	Connected	NEIN	Save Data To File	Erfüllt
11.	Save Data To File	-	Reconnect	Erfüllt
12.	Reconnect	-	Server Request	Erfüllt
13.	Connected	JA	DataSaved	Erfüllt
14.	Data Saved	NEIN	Get New Data To Send	Erfüllt
15.	Get New Data To Send	-	Server Request	Erfüllt
16.	Data saved	JA	Send Saved Data	Erfüllt
17.	Send Saved Data	-	Get New Data To Send	Erfüllt
18.	Get New Data To Send	-	Server Request	Erfüllt

Der Integrationstest wurde durchgeführt und es konnten alle möglichen Zweige des Ablaufs erreicht werden. Bei dem Test wurde nicht nur auf die Abläufe, sondern auch auf die Ergebnisse der einzelnen Module geachtet. Wie die Funktionstests ergeben haben, funktionieren die Module ohne weitere Probleme. Der ausgearbeitete Ablauf kann in dem Integrationstest wiedergefunden und verfolgt werden. Der entwickelte Datenlogger kann insofern entnommen werden.

An dieser Stelle werden die Datenlogger im realen Umfeld angewendet um weitere Tests durchzuführen und die Funktionalitäten weiterhin zu beobachten.

5.3.2. Fehlerfälle und Sondersituationen

In diesem Abschnitt der Arbeit werden die möglichen Sondersituationen und Fehlerfälle behandelt. Zuerst werden die Sondersituationen analysiert und die Folgen beschrieben.

- Bei einer S0 Schnittstellenkonfiguration muss die S0-Schnittstelle frei geschaltet werden, indem der Harken bei S0 gesetzt ist, ansonsten werden die Änderungen nicht gespeichert
- Die gespeicherten Sensordaten, wenn sie nicht versendet werden sollen, können nur in der Weboberfläche durch einen "Neustart"gelöscht werden. Der Speicher wird freigegeben und die dazu gehörigen Variablen, wie Anzahl der Zeilen oder die Anzahl der Dateien werden zurückgesetzt
- Während eines Softwareupdates erfolgt ein Aufruf des Webinterfaces, von einem anderen Klient wird das normal durchgeführt aber der Request des zweiten Klients kommt nicht mehr an dem Datenlogger an. Der zweite Klient wird nicht bedient, solange das Update nicht vollständig durchgeführt wird
- Wenn zwei Klients gleichzeitig die Weboberfläche der Datenlogger aufrufen, werden beide bedient. Für den Fall einer gleichzeitigen Konfigurationsänderung wird der letzte gespeicherte Wert behalten.
- Bei einem Internetausfall wird der Feuchtigkeitswert nicht in das Filesystem gespeichert, nur die Temperaturwerte
- Bei einem Stromausfall kann der Gasverbrauch nicht weiter gezählt werden. Es wird davon ausgegangen, dass in dem Fall die Heizungsanlage auch nicht funktioniert.

Als Zweitens werden die Fehlerfälle betrachtet, welche bis jetzt identifiziert wurden. Eine mögliche Fehlerquelle ist der Ausfall von Sensoren oder sogar der kompletten Busleitung. Solche Fälle können von dem Datenlogger erkannt werden, aber es wird keine Benachrichtigung für den Anwender geben. Die Funktionalität der Datenlogger wird nicht beeinträchtigt. Wenn einer oder mehrere Sensoren ausfallen, werden die restlichen Sensoren ausgelesen und verarbeitet, ohne dass die Funktionalität des Datenloggers beeinflusst wird.

Bei Ausfall einer Busleitung können keine Sensoren ausgelesen werden, dementsprechend werden keine Daten oder Nachrichten an den Datenbankserver gesendet. Der Datenlogger ist weiterhin über das Webinterface erreichbar aber die Sensoren nicht. In dem Fall muss die Ursache gesucht und beseitigt werden.

Ein zweiten Fehler könnte während eines Internetausfalls und gleichzeitigem Stromausfall entstehen. Dann werden die Sensordaten mit falschem Zeitstempel versehen, wenn bei der Neustart der Datenlogger immer noch kein Internet vorhanden ist. Die Daten werden zwar an den Datenbankserver gesendet, aber durch den falschen Zeitstempel (Unix Zeit 0) werden sie in EZN-Visio nicht angezeigt. Die Wahrscheinlichkeit, dass dieser Fall eintritt, ist sehr gering. In dieser Arbeit werden dafür keine Maßnahmen vorgenommen.

5.3.3. Sicherheit

Die Anforderungen an die Datensicherheit bekommen eine immer größere Bedeutung. Deswegen ist es besonders wichtig, dass die im Anwendungsbereich erfassten Daten sicher und verschlüsselt in der Datenbank ankommen.

Der Datenlogger wird mit einem verschlüsselten WLAN an dem UMTS/LTE Router verbunden und sendet im Normalfall aktuelle Sensordaten an den Datenbankserver. Der Router stellt ein verschlüsseltes Open-VPN Netzwerk zur Verfügung, welches von außerhalb des Netzwerkes nicht erreichbar ist. Durch dieses Netzwerk ist ein sicherer Datenverkehr gesichert.

Der Datenlogger speichert Sensordaten auf dem internen File System ab, wenn keine Internetverbindung zur Verfügung steht. Andere Konfigurationsdaten sind ebenfalls hier abgelegt. Die hier abgelegten Daten sind nicht mit einfachen Mitteln auslesbar. Außerdem werden die Temperaturwerte in einer Reihe nacheinander, ohne weitere Angaben gespeichert. Bei dem Auslesen werden die zusätzlichen Informationen rekonstruiert und an den Datenbankserver gesendet.

Da es sich um ein verschlüsseltes Netzwerk handelt, wurde eine Authentifizierung des Webinterface und der Datenlogger nicht vorgesehen. In der “Übersicht“ Seite des Datenloggers sind die Verbindungsinformationen zu den Routern sichtbar. Da der Datenlogger nur innerhalb des Netzwerks erreichbar ist, müssen trotzdem diese Informationen ausgeblendet werden. Die Änderungen dafür wurden durchgeführt.

5.4. EMV Test

Elektromagnetische Verträglichkeit (EMV) ist die Fähigkeit der elektrischen und elektronischen Geräter, zufriedenstellend zu funktionieren, ohne andere Einrichtungen durch ungewollte elektrische oder elektromagnetische Effekte zu stören oder selbst durch andere Geräte gestört zu werden [49]. EMV umfasst sowohl die Störfestigkeit als auch die Begrenzung der Störaussendung. Die Auswirkungen, die durch Änderungen oder Ausfall von Netzgrößen verursacht werden, werden im Rahmen von EMV nicht behandelt [50].

Zur Sicherstellung der elektromagnetisch verträglichen Funktion müssen die elektrischen und elektronischen Geräte sachgerecht aufgebaut und gestaltet werden. Die Mindeststörfestigkeit und die Begrenzung der Störaussendungen von Geräten sind in den EMV-Richtlinien der Europäischen Union, in Deutschland im Elektromagnetische Verträglichkeit

Gesetz (EMVG) festgelegt [49]. Um die Konformität der Europäischen EMV-Richtlinien nachweisen zu können, müssen die Elektrogeräte verschiedene EMV-Tests bestehen. Die Geräte, die den Test bestanden haben, erhalten die CE-Kennzeichnung [50].

5.4.1. EMV Messverfahren

Die Messungen werden in sogenannten geschirmten EMV-Räumen durchgeführt. Diese Räume dienen zur Fernhaltung von äußeren elektromagnetischen Beeinflussungen und der Begrenzung störender Emissionen. Zur Vermeidung des störenden Einflusses werden die geschirmten EMV-Messräume mit Ferritkacheln und Absorbern ausgekleidet. Die niedrigen Frequenzen werden durch die Ferritkacheln und die hohe Frequenzen durch Schaumstoffabsorber abgedeckt [51]. Die Schirmung funktioniert nach dem Prinzip des Faradayschen Käfigs, in dem weder von außen noch aus dem Raum hinaus elektromagnetische Strahlung gelangen kann.

Für die Messungen werden häufig zwei Typen von EMV-Räumen verwendet: vollständig reflexionsfreie Räume, Fully-anechoic Rooms (FAR), und Absorberräume mit reflektierendem Boden, Semi-anechoic Chamber (SAC). In der Hochschule Angewandte Wissenschaften Hamburg wurde ein FAR-Raum mit einer Messdistanz von 3m für solche Zwecke eingerichtet, wo die EMV-Messung der entwickelten Datenlogger durchgeführt wurde.

5.4.2. Durchführung der EMV-Messung

In dem Messlabor der Hochschule wurde eine Vorabmessung für eine Selbsterklärung der CE Zertifizierung der Datenlogger durchgeführt. Die entwickelte Steuerplatine wurde ohne Gehäuse auf einen Drehtisch gelegt. Somit ist der zu prüfende Teil auf halber Raumhöhe gebracht. Die Elektromagnetische Abstrahlung des Gerätes wurde im Niederfrequenzbereich bis 1GHz gemessen. Da der Signalgenerator nicht alle Frequenzen gleichzeitig abbilden kann, wird die Messung in zwei Teilen durchgeführt. Die Platine wurde in vier Ausrichtungen (Nord, Süd, Ost, West), in horizontaler und vertikaler Antennenrichtung gemessen. Das gleiche wurde in Hochfrequenzbereich, von 1GHz bis 6GHz ebenfalls durchgeführt.

Die Messergebnisse im Niederfrequenzbereich haben gezeigt, dass an einigen Stellen kleinere Störsignale vorhanden sind. Diese Signale liegen unterhalb der erlaubten Grenzwerte. Bei einer Frequenz von 853,32MHz ist das Störsignal minimal, mit 47,622$dB\mu V/m$, über dem Grenzwert hinausgeragt, siehe Abbildung 5.4.

Der Datenlogger wurde über ein 1m langes USB Kabel mit Spannung versorgt. Das USB Kabel dient gleichzeitig für Datenkommunikation, welche im Debug-Modus benutzt wird. Die

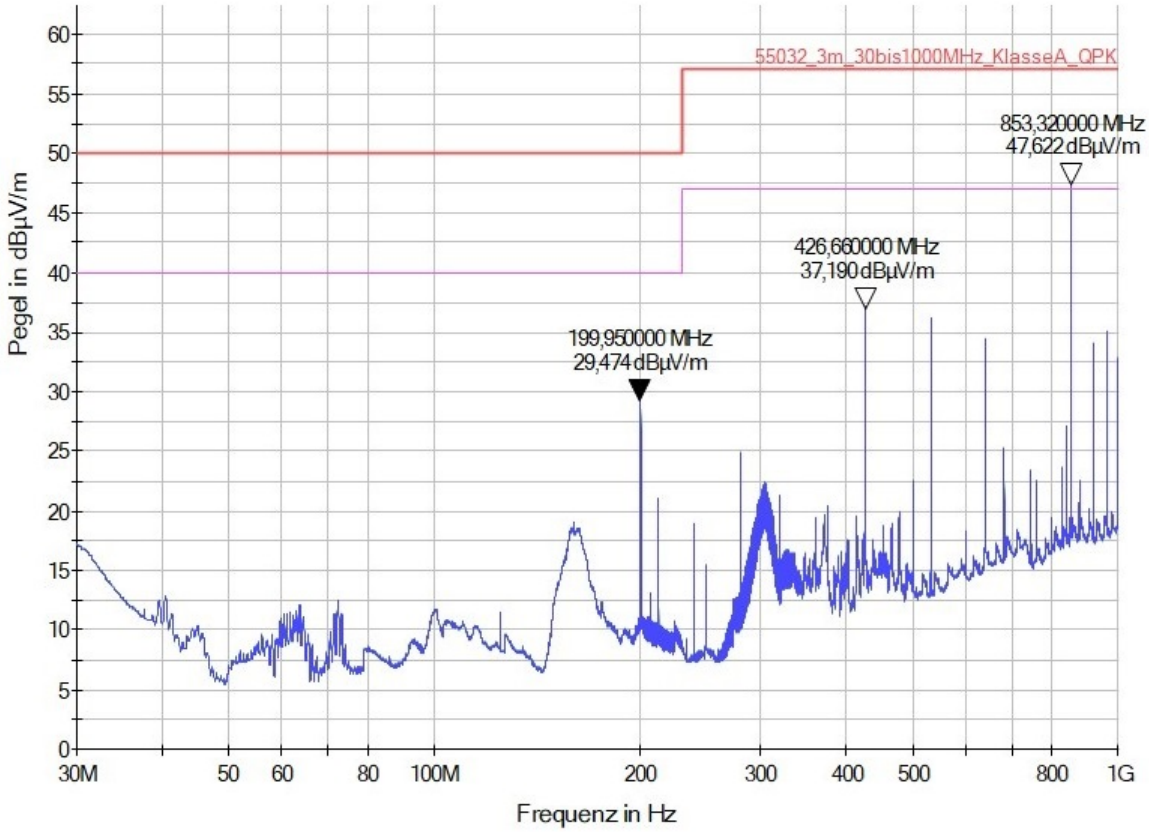


Abbildung 5.4.: EMV-Messung im Niederfrequenzbereich, mit einem hohen Störsignal bei 853,3MHz

Ursache von Störsignalen im Niederfrequenzbereich könnte dieses ungeschirmte USB Kabel sein.

Im Hochfrequenzbereich, von 1GHz bis 6GHz ist ein minimales Grundrauschlevel immer vorhanden. Die 2,4GHz WiFi Kommunikationsfrequenz auf der Aufzeichnung ist deutlich zu erkennen. Neben dem Grundrauschen wurden nur wenige Störsignale gemessen, die sich weit unter der Toleranzlinie befinden. Bei der doppelten WLAN-Frequenz, 4,82GHz ist ein Signalpegel immer fast mit der gleiche Höhe wiederzufinden. Wie in der Abbildung 5.5 erkennbar ist, ist das Störsignal teilweise ein wenig höher als der im Haushalt zugelassene Grenzwert. Im Anhang D sind alle Messergebnisse aufgelistet.

Da das entwickelte Modul nicht in dem Metallgehäuse, welches für die Anwendung vorgesehen ist, montiert und durchgemessen wurde, können an der offenen Platine Signale gemessen werden, die durch eine Schirmung nicht mehr messbar werden. Zusätzlich wird der Datenlogger über einen Netzteil mit Spannung anstatt über USB Kabel versorgt. Dadurch werden die Störsignalquellen wesentlich reduziert.

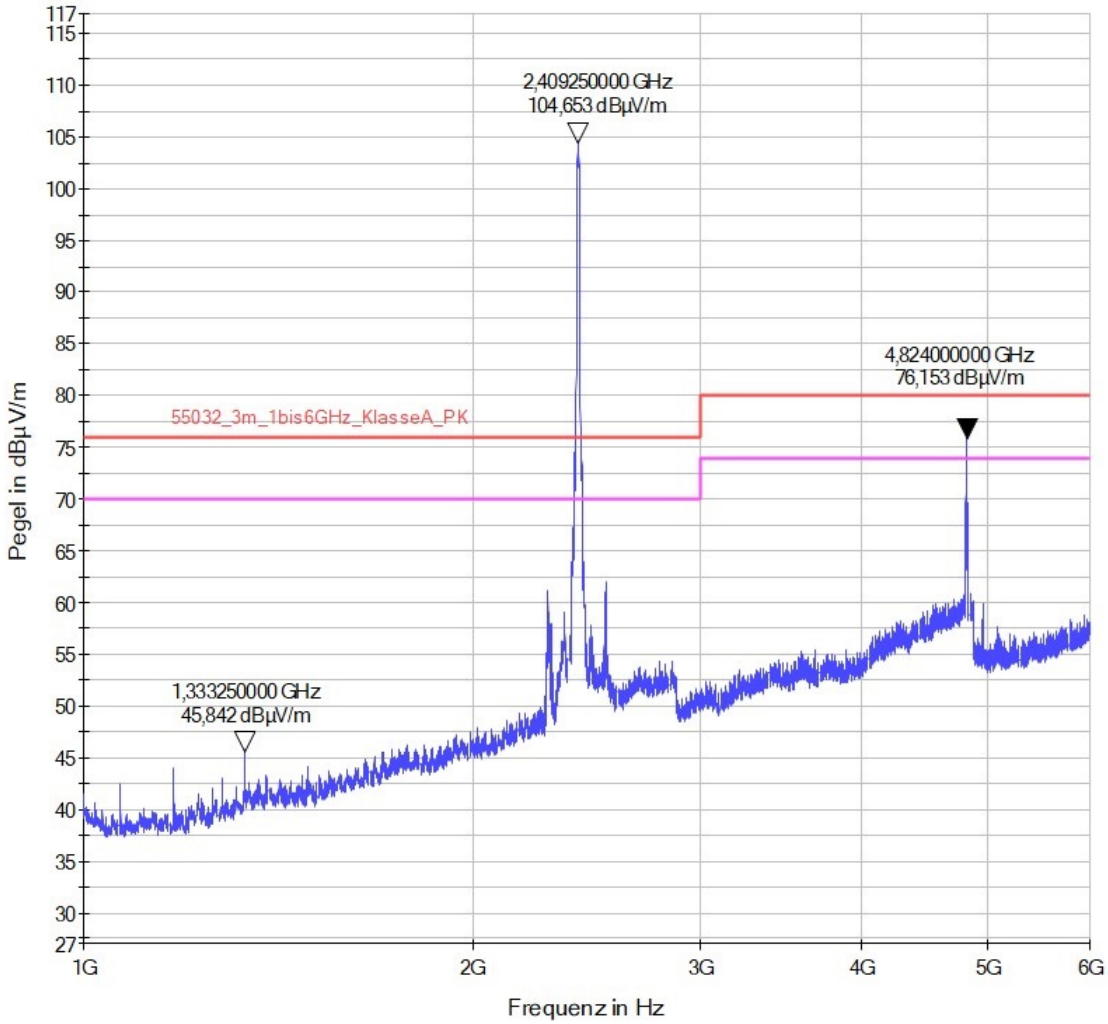


Abbildung 5.5.: EMV-Messung im 1 GHz bis 6GHz Bereich, mit dem Störsignal bei 4,82MHz

In der ersten Linie wurde die Störaussendung der entwickelten Platine gemessen. Um eine Selbsterklärung für die CE Zertifizierung ausstellen zu können, müssen weitere EMV-Messungen durchgeführt werden. Dafür muss der Datenlogger genauso wie in der Anwendung aufgebaut werden. Neben Störaussendungen muss die Störfestigkeit des Gerätes geprüft werden. Dazu gehören die leitungsgebundenen Prüfungen, wie Netzspannungseinbrüche, Einwirkungen von Schaltstörungen, elektrostatische Entladungen und Einkopplungen von Störungen von benachbarten Geräten. Diese Messungen werden nicht mehr im Rahmen dieser Arbeit durchgeführt.

5.5. Zusammenfassung der Testergebnisse

Für den entwickelten Datenlogger wurde ein Testplan erstellt und Tests wurden durchgeführt. Außer der Testphase wurden während der Entwicklung die Softwarekomponenten kontinuierlich getestet, da eine agile Entwicklungsmethode verfolgt wurde. Die in der Testphase durchgeführten Überprüfungen lassen sich in vier Kategorien einteilen:

- Komponententest
- Funktionstest
- Systemtest
- EMV Vortest

Die Fehler, die bei der Testdurchführung gefunden wurden und die Funktionalität des Datenloggers beeinträchtigten könnten, wurden beseitigt. Andere Fehler oder Stellen, die die Funktionalitäten des Datenloggers nicht beeinträchtigen, wurden dokumentiert und Hinweise darauf gegeben. Bei einer Weiterentwicklung sollten diese Hinweise beachtet werden.

Nach der Testdurchführung kann festgestellt werden, dass der entwickelte Datenlogger den Anforderungen entspricht und einsatzbereit ist.

6. Bewertung und Ausblick

Die Komplexität der Wärmeerzeugungsanlagen wird immer höher. Dadurch steigt die Fehleranfälligkeit von der Planung bis hin zur Inbetriebnahme. Durch die technologische Entwicklung stehen sehr viele Werkzeuge zur Verfügung, mit denen diese Fehler identifiziert und für Effizienzerhöhung und Optimierung eingesetzt werden können. Laut Bundesverband der Deutschen Heizungsindustrie (BDH-Köln) sind rund 63% der industriellen Heizungsanlagen unzureichend effizient [52].

Unter Berücksichtigung dieser Aspekte hat das Unternehmen Energiezentrale Nord eine Anforderungsliste für eine Messtechnik erstellt. Dazu wurde eine Analyse durchgeführt, um die richtige Komponente zu finden. Die Entwicklung eines Datenloggers für 20 Temperatursensoren, einen Feuchtigkeitssensor und einen S0-Schnittstelle wurde durchgeführt. Der Datenlogger eignet sich für kleine Anlagen, bei denen das Einsparvolumen gering ist, sehr gut.

6.1. Erreichtes

Die Anforderungen konnten erfüllt werden. Der Umstieg von analogen auf digitalen Bussensoren wurde umgesetzt. Das hat die Installation der Messtechnik erheblich erleichtert. Die grundlegenden Funktionalitäten, wie Temperaturdaten im Minutentakt zu erfassen und an den Datenbankserver zu senden, könnten in die bestehenden Infrastruktur ohne größeren Aufwand integriert werden.

Die Weboberfläche, die zur Übersicht und zur einfachen Konfigurationen dient, ist übersichtlich und kann sehr einfach bedient werden. Softwareaktualisierungen können über dieses Webinterface aus der Ferne schnell und bequem durchgeführt werden. Durch Hochladen von Dateien über den Webbrowser können die Struktur der Weboberfläche angepasst oder geändert und die Konfigurationsdateien ausgetauscht werden.

Durch ein geplantes Testen wurden die einzelnen Funktionen getestet. Dadurch wurden die Fehler leichter identifizierbar und der Aufwand für Korrekturen war geringer. Nachdem alle entdeckten Fehler beseitigt wurden, wurde der Systemtest durchgeführt. Die Erkenntnisse,

die aus dem Systemtest gewonnen wurden, sollten für die Weiterentwicklung berücksichtigt werden.

6.2. Offene Punkte

Ein Programm kann immer verbessert werden. In der Zwischenzeit wurden mehrere Datenlogger in die Heizungsanlagen zur Übertragung von Temperaturdaten eingesetzt. Durch die Anwendung entstehen neue Anforderungen. Zusammen mit den Testergebnissen haben sich folgende Vorschläge für den Weiterentwicklung ergeben:

- Die gespeicherten Sensordaten und die Konfigurationsdatei im File System des Mikrocontrollers verschlüsseln
- Authentifizierung für das Webinterface
- Benachrichtigung im Falle des Sensorausfalls
- Die Sensorbezeichnungen den Sensoren zuordnen
- Dynamische Anpassung des Zeitraums der gespeicherten Sensordaten anhand der Anzahl von angeschlossenen Sensoren
- Feuchtigkeitswert in das File System speichern, wenn keine Internetverbindung vorhanden ist
- Softwareerweiterung für ein weitere One-Wire-Bus, Hardwareseitig ist schon vorhanden
- Einheitliche Seiten für das Webinterface, bei `Upload` und `Update`
- Durch das Webinterface das Löschen von Dateien ermöglichen

Für kleine Wärmeerzeugungsanlagen ist es sehr nützlich, eine kompakte und kostengünstige Messtechnik zu verwenden. Der entwickelte Datenlogger kann mit geringem Aufwand installiert werden. Laut der Monteure ist die Inbetriebnahme und die Bedienung der Weboberfläche wesentlich einfacher als bei den bisher verwendeten Datenloggern. Die Anschaffung von Bauteilen ist wesentlich günstiger und der Zusammenbau kann per Hand gemacht werden. Insgesamt können die Daten für Optimierung mit geringerem Aufwand und Kosten bereitgestellt werden und somit sind die Ziele diese Arbeit erreicht worden.

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

BHKW Blockheizkraftwerk 7

EMVG Elektromagnetische-Verträglichkeit-Gesetz 11

EMV Elektromagnetische Verträglichkeiten 56

WLAN Wireless Local Area Network

LTE Long Term Evolution

UMTS Universal Mobile Telecommunications System

PT1000 Platin-Messwiderstand, $R_0 = 1k\Omega$

REST API Representational State Transfer Application Programming Interface 10

IoT Internet of Things

SoC System on Chip

FFS Flash File System 14

NTC Negativer Temperaturkoeffizient 16

IC Integrated Circuit

Literaturverzeichnis

[1] Ingenieurbüro H.Lertes GmbH & Co. KG. Home, (Zugegriffen am 11.04.2019).

[2] Bundesrecht. Emvg elektromagnetische-verträglichkeit-gesetz, (Zugegriffen am 11.04.2019).

[3] Michel Deslierres. Another esp8266 nodemcu development board, (Zugegriffen am 12.04.2019).

[4] LastMinuteEngineers. 0a-esp8266ex_datasheet_en.pages, (Zugegriffen am 12.04.2019).

[5] ebay. Esp8266 esp-12e ch340 wifi netzwerk entwicklung board modul für nodemcu fbb ebay, (Zugegriffen am 12.04.2019).

[6] ESPRESSIF. Esp32-devkitc v4 getting started guide esp-idf programming guide v4.0-dev-311-g70eda3d22 documentation, (Zugegriffen am 12.04.2019).

[7] ESPRESSIF. esp32_datasheet_en.pdf, (Zugegriffen am 12.04.2019).

[8] ebay. Esp32 esp-32s nodemcu-entwicklungsboard 2.4 ghz wifi + bluetooth dual mode neu ebay, (Zugegriffen am 12.04.2019).

[9] Adafruit. Adafruit feather m0 wifi - atsamd21 + atwinc1500 id: 3010 - $34.95 : Adafruit industries, unique & fun diy electronics and kits, (Zugegriffen am 12.04.2019).

[10] Lady Ada. adafruit-feather-m0-wifi-atwinc1500.pdf, (Zugegriffen am 12.04.2019).

[11] Reichelt. Ada fb m0 wifi: Cartes de developpement - adafruit feather m0 wifi chez reichelt elektronik, (Zugegriffen am 12.04.2019).

[12] Arduino. Arduino mkr1000 wifi, (Zugegriffen am 12.04.2019).

[13] Mediatek labs. Mediatek linkit smart 7688 iot development platform mediatek labs, (Zugegriffen am 12.04.2019).

[14] Eckstein Komponente. Seeed studio linkit smart 7688 - raspberry pi, arduino, robote, motor etc. beim eckstein komponente, 17,73, (Zugegriffen am 12.04.2019).

[15] Stefan Heusel Maurice Bildstein. document2360470226777143247.indd, (Zugegriffen am 12.04.2019).

[16] Texas Instrument. Lmt01 0.5Â°c accurate 2-pin digital output temperature sensor with pulse count datasheet (rev. d), (Zugegriffen am 15.04.2019).

[17] Mouser. Lmt01 texas instruments, mouser germany, (Zugegriffen am 15.04.2019).

[18] Sparkfun. Programmable resolution 1-wire digital thermometer, (Zugegriffen am 15.04.2019).

[19] AZ-Delivery. Ds18b20 mit 1m kabel - az-delivery, (Zugegriffen am 15.04.2019).

[20] maximintegrated. Ds1621.pdf, (Zugegriffen am 15.04.2019).

[21] aosong. Am2301.pdf, (Zugegriffen am 15.04.2019).

[22] Eckstein komponente. Am2301 humidity capacitor module - raspberry pi, arduino, robote, motor etc. beim eckstein komponente, 6,95, (Zugegriffen am 15.04.2019).

[23] Mouser. Dht11 humidity & temperature sensor, (Zugegriffen am 15.04.2019).

[24] Eckstein komponente. Dht11 digitaler temperatur- und feuchtigkeitssensor - raspberry pi, arduino, robote, motor etc. beim eckstein komponente, 2,00, (Zugegriffen am 15.04.2019).

[25] Sparkfun. Dht22.pdf, (Zugegriffen am 15.04.2019).

[26] Eckstein komponente. Dht22 am2302 digital temperatur feuchtigkeit sensor mit kabel und 4,7 kohm widerstand - raspberry pi, arduino, robote, motor etc. beim eckstein komponente, 4,95, (Zugegriffen am 15.04.2019).

[27] Raspberry PI Tutorials. EinfÃ¼hrung & programmierung des esp8266 nodemcu boards, (Zugegriffen am 15.04.2019).

[28] ESP8266 Community. Welcome to esp8266 arduino core s documentation! - esp8266 arduino core documentation, (Zugegriffen am 15.04.2019).

[29] Components 101. Ds18b20 temperature sensor pinout, specifications, equivalents & datasheet, (Zugegriffen am 15.04.2019).

[30] How to Mechatronics. Dht11 & dht22 sensor temperature and humidity tutorial, (Zugegriffen am 15.04.2019).

[31] Mean Well. Mdr-20-20170707.cdr, (Zugegriffen am 15.04.2019).

[32] Connrad. 156673-da-01-de-dc_dc_wandler_1a_tsr_1_2450.pdf, (Zugegriffen am 15.04.2019).

[33] Tutorialpoints. Html forms, 2015 (Zugegriffen am 25.04.2019).

[34] Aisler. Aisler, 2015 (Zugegriffen am 25.04.2019).

[35] LUA Community. Lua-dokumentation, 2015 (Zugegriffen am 01.05.2019).

[36] ESP8266 Community Forum. Arduino/libraries at master esp8266/arduino, (Zugegriffen am 14.06.2019).

[37] Ivan Grokhotkov. Welcome to esp8266 arduino core s documentation! esp8266 arduino core documentation, 2015 (Zugegriffen am 01.05.2019).

[38] PlatformIO. An open source ecosystem for iot development platformio, 2015 (Zugegriffen am 01.05.2019).

[39] ESP8266 Community. Arduino/tools/sdk/ld at master esp8266/arduino, 2015 (Zugegriffen am 01.05.2019).

[40] Douglas Crockford. Json, 2015 (Zugegriffen am 04.05.2019).

[41] Benoit Blanchon. Documentation arduinojson 6, 2015 (Zugegriffen am 06.05.2019).

[42] Walt Kelly Pogo. The network time protocol (ntp) distribution, 2014 (Zugegriffen am 06.05.2019).

[43] ESP8266 Community. Arduino/libraries/esp8266httpupdateserver at master esp8266/arduino, 2015 (Zugegriffen am 17.05.2019).

[44] Horst Lichter Jochen Ludewig. *Software Engineering*. dpunkt.verlag, 2013.

[45] IEEE. 29119-1-2013 - iso/iec/ieee international standard - software and systems engineering -software testing -part 1:concepts and definitions - ieee standard, 2019 (Zugegriffen am 23.05.2019).

[46] Herwig Mayr. *Projekt Engineering*. Carl Hanser Verlag, 2001.

[47] Sogeti. Tmap next downloads german tmap, 2019 (Zugegriffen am 20.05.2019).

[48] Trotec GmbH and Co.KG. Dl200p-datenlogger - trotec, 2019 (Zugegriffen am 06.06.2019).

[49] Mobilfunk. Was bedeutet elektromagnetische vertraeglichkeit (emv) informationszentrum mobilfunk, 2019 (Zugegriffen am 05.07.2019).

[50] Baunetzwissen. Elektromagnetische vertraeglichkeit (emv) sicherheitstechnik glossar baunetz-wissen, 2019 (Zugegriffen am 05.07.2019).

[51] Wolfgang Kürner Adolf J. Schwab. *Elektromagnetische Verträglichkeit*. Springer-Verlag Berlin Heidelberg, 2011.

[52] BDH. Infografik effizienzstruktur 2016 de.indd, 2019 (Zugegriffen am 13.06.2019).

[53] Edwin Notenboom Bart Broekman. *Testing Embedded Software*. Addison-Wesley, 2003.

[54] Karsten Günther. *LaTeX GE-PACKT*. mitp, 2002.

[55] Leslie Lamport. *Das LaTeX Handbuch*. Addison-Wesley, 1995.

[56] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *Der LaTeX -Begleiter*. Addison-Wesley, 2000.

[57] Markus Kohm and Jens-Uwe Morawksi. *KOMA-Script*. dante, 2003.

[58] Andrew Hunt and David Thomas. *Der Pragmatische Programmierer*. Hanser, 2003.

[59] Oliver Böhm. *Java Software Engineering unter Linux*. SuSE Press, 2002.

[60] Klaus-Dieter Schmatz. *Java 2 Micro Edition - Entwicklung mobiler Anwendungen mit CLDC und MIDP*. dpunkt.verlag, 2004.

[61] Norbert Streitz and Paddy Nixon. The disappearing computer. *Communications of the ACM*, 48(3):33–35, March 2005.

[62] Hewlett-Packard. Cooltown (overview), 2004.

[63] Rolf von Lüde, Daniel Moldt, and Rüdiger Valk. *Sozionik - Modellierung soziologischer Theorie*. LIT, 2004.

[64] Tom DeMarco and Tomothy Lister. *Wien wartet auf Dich - Der Faktor Mensch im DV-Management*. Carl Hanser Verlag München Wien, 1999.

[65] Malte Kollakowski. Mobile aktivitäten. *Der Entwickler*, (4.04):15–20, April 2004.

[66] Christian W. Dawson. *Computerprojekte im Klartext*. Pearson Studium, 2003.

[67] Klaus Poenicke. *Wie verfaßt man wissenschaftliche Arbeiten?* Duden, 1988.

[68] Otto Kruse. *Keine Angst vor dem leeren Blatt*. campus concret, 2000.

[69] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, 1998.

[70] Jochen Heinsohn and Rolf Socher-Ambrosius. *Wissensverarbeitung*. Spektrum Akademischer Verlag, 1999.

[71] George F. Luger. *Künstliche Intelligenz*. Pearson Studium, 2001.

[72] Ralf Kühnel. *Agentenbasierte Softwareentwicklung*. Addison-Wesley, 2001.

[73] Joseph Bigus and Jennifer Bigus. *Intelligente Agenten mit Java programmieren*. Addison-Wesley, 2001.

[74] Jacques Ferber. *Multiagentensysteme*. Addison-Wesley, 2001.

[75] Michael Wooldridge. *MultiAgent Systems*. Wiley, 2002.

[76] James F. Kurose and Keith W. Ross. *Computernetze*. Pearson Studium, 2002.

[77] Carsten Vogt. *Betriebssysteme*. Spektrum Akademischer Verlag, 2001.

[78] Florian Rötzer. Ein neuer standard soll kundendaten zusammenführen, 1999.

[79] W3C. Platform for privacy preferences (p3p) project, 2004.

[80] IDEAlliance. Cp exchange, 2004.

[81] Prof. Dr. Dr. h.c. Günther Drosdowski, Dr. Werner Scholze-Stubenrecht, and Dr. Matthias Wermke, editors. *Das Fremdörterbuch*. Duden, 1997.

[82] Julia Hörauf. Hewlett-packard cooltown project. Universität Karlsruhe - Institut für Telematik, Seminararbeit zum Seminar Ubiquitäre Systeme, 2001.

[83] Rafael Schirru. Trust, reputation, privacy. Universität Karlsruhe - Lehrgebiet Datenverwaltungssysteme, Seminararbeit zum Seminar Grundlagen webbasierter Informationssysteme, 2004.

[84] Alexander Babic. Entwicklung einer profilverarbeitenden ubiquitären anwendung. Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg, 2003.

[85] Andru Lüpke. Entwurf einer sicherheitsarchitektur für den einsatz mobiler endgeräte. Diplomarbeit, Hochschule für Angewandte Wissenschaften Hamburg, 2004.

A. Grafiken

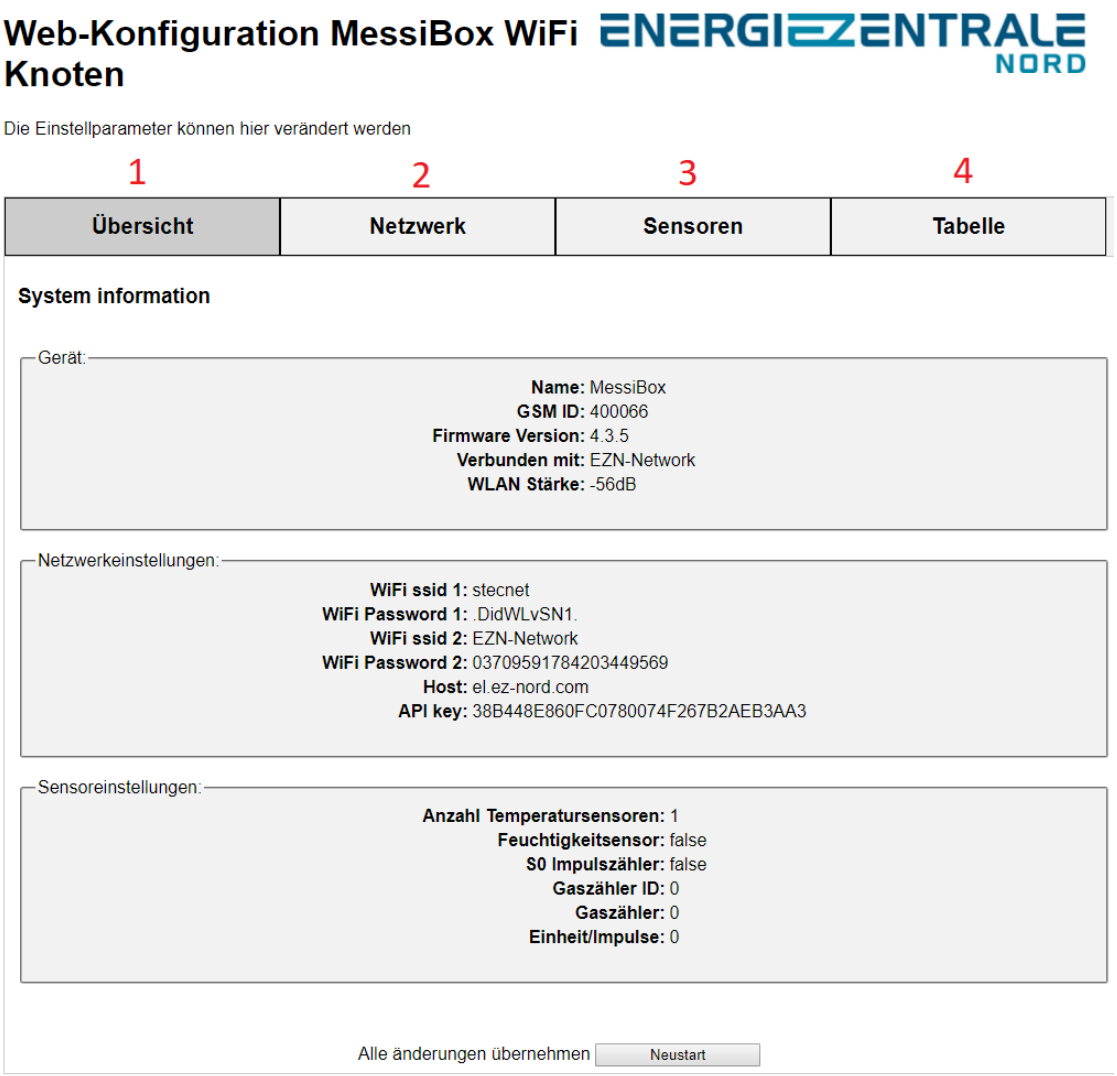


Abbildung A.1.: Tab Uebersicht

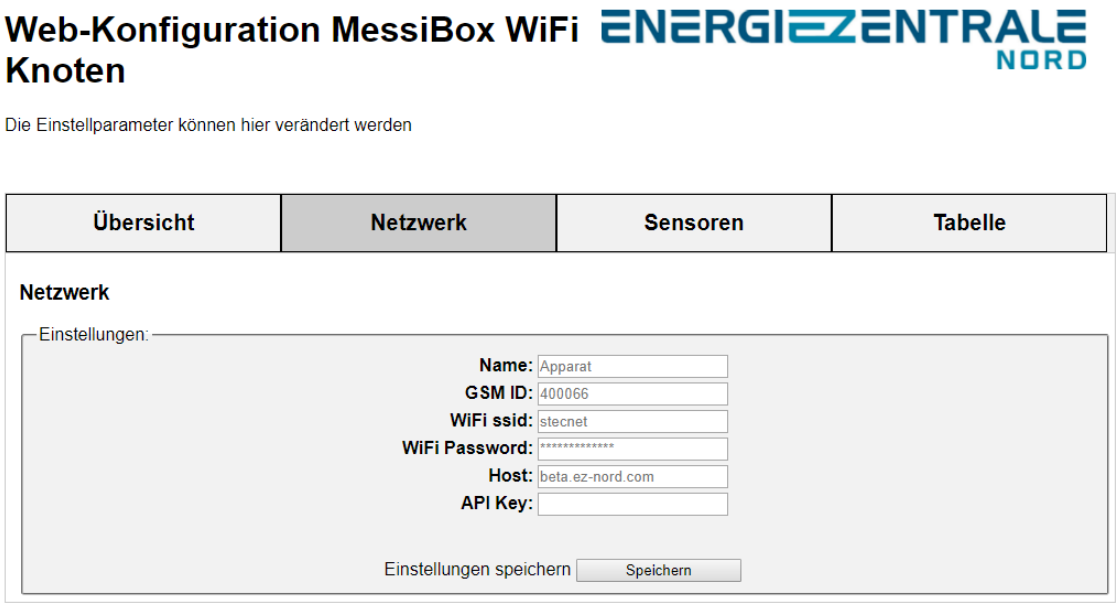


Abbildung A.2.: Tab Netzwerk

Abbildung A.3.: Tab Sensoren

Web-Konfiguration MessiBox WiFi Knoten

ENERGIEZENTRALE NORD

Die Einstellparameter können hier verändert werden

Übersicht	Netzwerk	Sensoren	Tabelle

Temperaturtabelle

undefined

NR	Knoten	Sensor-ID	Wert	Einheit	TP	Bezeichnung
1	400066	183373b0ff	23.50	°C	TEMP_1	
2	400066	18337988ff	23.50	°C	TEMP_1	
3	400066	18335534ff	23.50	°C	TEMP_1	
4	400066	1833c1ecff	23.00	°C	TEMP_1	
5	400066	1833bf79ff	23.00	°C	TEMP_1	
6	400066	16c23da5ff	24.00	°C	TEMP_1	
7	400066	1833bf03ff	23.50	°C	TEMP_1	
8	400066	18337433ff	23.50	°C	TEMP_1	
9	400066	18330a4bff	23.50	°C	TEMP_1	
10	400066	16c02c6bff	23.50	°C	TEMP_1	
11	400066	DHT22AM220	51.90	%	TEMP_1	Relative Luftfeuchte
12	400066	DTT22AM230	23.80	°C	TEMP_1	Temperatur
13	400066	DHT22AM240	13.15	°C	TEMP_1	Taupunkttemperatur
14	400066	DHT22AM250	11.31	g/m^3	TEMP_1	Absolute Luftfeuchte
15	400066	123456	654587.44	m^3	GAS	

Abbildung A.4.: Tab Sensortabelle

B. Source Code

main.cpp

```
/*
 * main.cpp
 *
 *  Created on: 31.07.2018
 *      Author: jgergely
 */

#include <Ticker.h>
#include "wifiConnection.h"
#include "timeServer.h"
#include "temperatureSensors.h"
#include "impulseCounter.h"
#include "fileSaver.h"
#include "espServer.h"

const char* FIRMWARE = "4.3.5";

void setup() {

  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, LOW);
  Serial.begin(112500);
  PRINT("\nESP8266 is ready\n");
  config.firmware = FIRMWARE;
  wifiConnection();
  printWiFiInfo();
  init_FS();
  sensorBegin();
  initUDP();
  writeConfigFile();
  initHandleClient();
  InterruptHandling();
  interrupt.attach(60, interruptHandle);
}

void loop() {
```

```
37
38    serverHandleClient();
39    updateS0();
40
41    if (interruptFlag) {
42      interruptFlag = false;
43      if (isConnected()) {
44        PRINTLN("Mesibox is connected to EZ-N server");
45  //        SAVE_TEST();
46
47        uint8_t savedValues = readFromFile("/saved.txt").toInt();
48        if (savedValues) {
49          readDataFromSD();
50        }
51        timeNow = getNTPTime();
52        if (timeNow != 0) {
53          preTime = timeNow;
54        } else {
55          timeNow = preTime + 60;
56          preTime = timeNow;
57        }
58        timestmp = getTimeStemp(timeNow);
59        PRINT("Time request: ");
60        PRINTLN(timestmp);
61
62        String temp = getTemperatures(true, sensor_1, sensors_1, sensorAddress_1);
63        if (temp != "NULL") {
64          request(temp);
65          PRINTF("temp_1:\n %s \n", temp.c_str());
66        }
67        yield();
68        temp = "";
69        temp = getTemperatures(true, sensor_2, sensors_2, sensorAddress_2);
70        if (temp != "NULL") {
71          request(temp);
72          PRINTF("temp_2:\n %s \n", temp.c_str());
73        }
74        yield();
75        temp = "";
76        temp = getHumidity(true);
77        if (temp != "NULL") {
78          request(temp);
79          PRINTF("hum:\n %s \n", temp.c_str());
80        }
81        temp = "";
82        yield();
83        temp = returnImpulse();
84        if (temp != "NULL") {
```

```
85            request(temp);
86            PRINTF("imp:\n %s \n", temp.c_str());
87          }
88          yield();
89        } else {
90          PRINTLN("Mesibox cannot connect to EZ-N server");
91          saveDataToSD();
92          wifiReconnect();
93        }
94      }
95    }
```

globals.h

```
/*
 * globals.h
 *
 *  Created on: 04.02.2019
 *      Author: JozsefGergely
 *
 * example for global variables to use in any file
 *
 * global.h:
 *    #ifndef SRC_GLOBALS_H_
 *    #define SRC_GLOBALS_H_
 *    extern int global_variable;  // define
 *      #endif // SRC_GLOBALS_H_
 *
 * global.cpp
 *    #include "global.h"
 *    int global_variable;    // declare
 *    global_variable  = 0;   // and initialize (option)
 *
 * otherFiles.h
 *    #include "global.h"     //
 *    // use variable
 *
 *
 * Usage of DHT.h Library:
 *
 * 8
 */

#ifndef SRC_GLOBALS_H_
#define SRC_GLOBALS_H_

#include <Arduino.h>
#include <ArduinoJson.h>
#include <DallasTemperature.h>
#include <OneWire.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <WiFiUdp.h>
#include <TimeLib.h>
#include <ESP8266HTTPUpdateServer.h>

/*
 * Define Serial debugging with macro
 * Uncomment "#define DEBUG" for production
 * */
```

```
47 #define DEBUG
48
49 #ifdef DEBUG
50   #define PRINT(x) Serial.print(x)
51   #define PRINTLN(x) Serial.println(x)
52   #define PRINTF(x, y) Serial.printf(x, y)
53   #define SAVE_TEST() saveTest()
54
55 #else
56   #define PRINT(x)
57   #define PRINTLN(x)
58   #define PRINTF(x, y)
59   #define SAVE_TEST()
60 #endif
61
62 #define PATH_FILE_COUNTER    "/fileCounter.txt"
63 #define PATH_ROW_COUNTER     "/rowCounter.txt"
64 #define PATH_SAVED_DATA    "/saved.txt"
65
66 /*
67  * Define global variables for temperature sensors
68  * */
69 extern const uint8_t ONE_WIRE_BUS_1;
70 extern const uint8_t ONE_WIRE_BUS_2;
71 extern const uint8_t PRECISION;
72
73 extern OneWire oneWire_1;
74 extern OneWire oneWire_2;
75 extern DallasTemperature sensors_1;
76 extern DallasTemperature sensors_2;
77 extern DeviceAddress sensorAddress_1;
78 extern DeviceAddress sensorAddress_2;
79
80 extern uint8_t sensor_1;
81 extern uint8_t sensor_2;
82 extern bool humidity;
83
84 extern const uint8_t DHT_PIN;
85 extern const uint8_t DHT_TYPE;
86
87 /*
88  * Define global variables for timestemp and
89  * data saving in case of no connection
90  * */
91
92 extern bool interruptFlag;
93
94 extern const uint8_t MAX_ROW;
```

```
95  extern const uint8_t MAX_FILE;
96
97  extern uint8_t rowCounter;
98  extern uint8_t fileCounter;
99  extern uint32_t filePionter;
100
101 extern String timestmp;
102 extern time_t timeNow;
103 extern time_t preTime;
104
105 /*
106  * Define HTTPS Port for server requests
107  * */
108 extern const int PORT;
109
110 /*
111  * Define variables for NTP time server packets
112  * */
113 extern WiFiUDP udp;
114 extern const int NTP_PACKET_SIZE;
115 extern const int LOCALE_PORT;
116 extern const int TIME_ZONE;
117 extern const char* NTP_SERVER_NAME;
118
119 extern int m_timeZone;
120 extern String ntpServerName;
121 extern uint8_t packetBuffer[];
122
123 /*
124  * Define global configuration structure
125  * */
126 typedef struct {
127
128   bool conf = false;
129
130   String name = "";
131   uint32_t gsmId = 0;
132   String firmware = "";
133
134   String connectedTo = "";
135   String wifiStrength = "";
136
137   String ssid0 = "";
138   String ssid1 = "";
139   String pass0 = "";
140   String pass1 = "";
141
142   String apiUrl = "";
```

```
143   String apiKey = "";
144
145   uint8_t nrSensor = 0;
146   bool configDHT = false;
147   bool configS0 = false;
148
149   String impulseID = "";
150   String impulseVal = "";
151   float unitPImp = 0.0;
152
153 }messi;
154 extern messi config;
155
156 /*
157  * Define a String array to hold remarks for temperature sensors
158  * */
159 extern String descrioption[20];
160
161 #endif /* SRC_GLOBALS_H_ */
```

globals.cpp

```
/*
 * globals.cpp
 *
 *  Created on: 04.02.2019
 *      Author: JozsefGergely
 */

#include "globals.h"

/*
 * Define global variables for temperature sensors
 * */

const uint8_t ONE_WIRE_BUS_1 = 2;
const uint8_t ONE_WIRE_BUS_2 = 0;

const uint8_t PRECISION = 9;

OneWire oneWire_1(ONE_WIRE_BUS_1);
OneWire oneWire_2(ONE_WIRE_BUS_2);
DallasTemperature sensors_1(&oneWire_1);
DallasTemperature sensors_2(&oneWire_2);
DeviceAddress sensorAddress_1;
DeviceAddress sensorAddress_2;

uint8_t sensor_1;
uint8_t sensor_2;
bool humidity = false;

const uint8_t DHT_PIN = 4;
const uint8_t DHT_TYPE = DHT22;

/*
 * Declare and initialize global variables for measuring interval, timestemp and
 * data saving in case of no connection
 * */

bool interruptFlag = false;

const uint8_t MAX_ROW = 60;
const uint8_t MAX_FILE = 48;

uint8_t rowCounter = 0;
uint8_t fileCounter = 0;
uint32_t filePionter = 0;

```

```
47 String timestmp = "";
48 time_t timeNow = 0;
49 time_t preTime = 0;
50
51 /*
52  * Declare HTTPS Port for server requests
53  * */
54 const int PORT = 443;
55
56 /*
57  * Declare and initialize variables for NTP time server packets
58  * */
59 WiFiUDP udp;
60 const int NTP_PACKET_SIZE = 48;
61 const int LOCALE_PORT = 8888;
62 const int TIME_ZONE = 0;
63 const char* NTP_SERVER_NAME = (char*)"us.pool.ntp.org";
64
65 int m_timeZone = TIME_ZONE;
66 String ntpServerName = NTP_SERVER_NAME;
67 uint8_t packetBuffer[NTP_PACKET_SIZE]{0};
68
69 /*
70  * Declare global configuration variable (structure)
71  * */
72 messi config;
73
74 /*
75  * Declare a String array to hold remarks for temperature sensors
76  * */
77 String descrioption[20] = "";
```

helpers.h

```
/*
 * functions.h
 *
 *  Created on: 04.02.2019
 *      Author: JozsefGergely
 */

#ifndef SRC_HELPERS_H_
#define SRC_HELPERS_H_

#include <Arduino.h>
#include <ArduinoJson.h>
#include <FS.h>
#include <EEPROM.h>
#include"globals.h"

/*********************************************************
 *  It writes configurations data String in a file to
 *  SPIFFS. Data string and path need to be specified
 *    Path:   /data.csv
 *        /config_file.json
 * *******************************************************/
bool writeToFile(String a_data, String a_path);

/*********************************************************
 *  Read configurations file from SPIFFS and return as a
 *  String. Path need to be specified
 *    Path:   /data.csv
 *        /config_file.json
 * *******************************************************/
String readFromFile(String a_path);

/*********************************************************
 *  Create table head of all the sensors for overview
 * *******************************************************/
void createTableHead();
/*********************************************************
 *  Create table details of all the sensors for overview
 *  The function creates one row of the table
 * *******************************************************/
void createTableDetails(uint8_t a_number, uint32_t a_gsmID, String a_sensorID,
    float a_value, String a_unit, String a_notation, String a_description);

/*********************************************************
 * Save S0 value to the EEPROM
 * *******************************************************/
```

```
47 void putCounterValueEEPROM(String a_value);
48
49 /*******************************************************
50  * get S0 counter value from EEPROM
51  * ****************************************************/
52 String getCounterValueEEPROM();
53
54 /*******************************************************
55  * Check if there is a saved value, otherwise the EEPROM
56  * can't be read
57  * ****************************************************/
58 bool EEPROMReady();
59
60 /*******************************************************
61  * Calculate the absolute humidity and the "Taupunkt"
62  * temperature if there is a humidity sensor
63  *    Argument: a_humVector[4]
64  * ****************************************************/
65 void calcTaupunkt(float* a_humVector);
66
67 /*******************************************************
68  * Create and convert a path for save temperature values
69  * in file
70  * ****************************************************/
71 String fileNameToPath(uint8_t a_fileIndex);
72
73 /*******************************************************
74  * Convert file data to a right  JSON String format, with
75  * time stamps for PUT request
76  * ****************************************************/
77 String convertStringToJSON(uint8_t a_nrSensor, String a_timestmp, String a_addr,
    String a_temp);
78
79 void removeFile(String a_filename);
80
81 #endif /* SRC_HELPERS_H_ */
```

helpers.cpp

```
/*
 * functions.cpp
 *
 *  Created on: 04.02.2019
 *      Author: JozsefGergely
 */

#include <helpers.h>

bool writeToFile(String a_data, String a_path) {

  SPIFFS.begin();
  SPIFFS.remove(a_path);
  File file = SPIFFS.open(a_path, "w");

  if (!file) {
    file.close();
    PRINTF("File %s can not be write\n", a_path.c_str());
    return false;
  } else {
    a_data += "\0";
    file.println(a_data);
    file.close();
    return true;
  }
}

String readFromFile(String a_path) {

  SPIFFS.begin();
  File file = SPIFFS.open(a_path, "r");
  String line = "";
  if (!file) {
    file.close();
    PRINTF("File %s can not be open\n", a_path.c_str());
    return "NULL";
  } else {
    if (a_path == "/counter.txt") {
      if (file.available()) {
        line = file.readStringUntil('\r');
      }
    } else {
      if (file.available()) {
        line = file.readString();
      }
    }
```

```
47    }
48    file.close();
49    SPIFFS.end();
50    return line;
51  }
52
53  void createTableHead() {
54
55    SPIFFS.begin();
56    File table_head = SPIFFS.open("/data.csv", "w");
57    table_head.print("NR");
58    table_head.print(";");
59    table_head.print("Knoten");
60    table_head.print(";");
61    table_head.print("Sensor-ID");
62    table_head.print(";");
63    table_head.print("Wert");
64    table_head.print(";");
65    table_head.print("Einheit");
66    table_head.print(";");
67    table_head.print("TP");
68    table_head.print(";");
69    table_head.println("Bezeichnung");
70    table_head.close();
71
72    SPIFFS.end();
73  }
74
75  void createTableDetails(uint8_t a_number, uint32_t a_gsmID, String a_sensorID,
76      float a_value, String a_unit, String a_notation, String a_description) {
77
78    SPIFFS.begin();
79    File table_data = SPIFFS.open("/data.csv", "a");
80    table_data.print(a_number);
81    table_data.print(";");
82    table_data.print(a_gsmID);
83    table_data.print(";");
84    table_data.print(a_sensorID);
85    table_data.print(";");
86    table_data.print(a_value);
87    table_data.print(";");
88    table_data.print(a_unit);
89    table_data.print(";");
90    table_data.print(a_notation);
91    table_data.print(";");
92    table_data.println(a_description);
93    table_data.close();
94
```

```
95    SPIFFS.end();
96  }
97
98  void putCounterValueEEPROM(String a_value) {
99
100   EEPROM.begin(512);
101   EEPROM.write(0, 1);
102   byte dataSize = a_value.length();
103   EEPROM.write(1, dataSize);
104
105   for (byte i = 2; i < dataSize + 1; i++) {
106     EEPROM.write(i, a_value[i - 2]);
107   }
108   EEPROM.commit();
109   EEPROM.end();
110 }
111
112 String getCounterValueEEPROM() {
113   EEPROM.begin(512);
114   byte size = EEPROM.read(1);
115   char buffer[size];
116
117   for (byte i = 2; i < size + 1; i++) {
118     buffer[i - 2] = EEPROM.read(i);
119   }
120   buffer[size] = '\0';
121   EEPROM.commit();
122   EEPROM.end();
123   return String(buffer);
124 }
125
126 bool EEPROMReady(){
127   EEPROM.begin(512);
128   if (EEPROM.read(0) == 1){
129     return true;
130   } else {
131     return false;
132   }
133 }
134
135 void calcTaupunkt(float* a_humVector) {
136
137   double a = a_humVector[0];
138   double b = a_humVector[1];
139
140   double ssd = 6.1078 * pow(10, ((7.5 * b) / (237.3 + b)));
141   double dd = (a / 100) * ssd;
142   double v = log10(dd / 6.178);
```

```
143
144   a_humVector[2] = 237.3 * v / (7.5 - v);                    // Taupunkt Temperature
145   a_humVector[3] = pow(10, 5) * (18.016 / 8314.3) * (dd / 293.15); // Absolute
        humidity
146 }
147
148 String fileNameToPath(uint8_t a_fileIndex){
149   return "/" + (String)a_fileIndex + ".csv";
150 }
151
152 String convertStringToJSON(uint8_t a_nrSensor, String a_timestmp, String a_addr,
        String a_temp) {
153
154   if (a_nrSensor == 0) {
155     return "NULL";
156   }
157
158   uint8_t addrIndex = 0;
159   uint8_t tempIndex = 0;
160
161   StaticJsonBuffer<2024> jsonBuffer; // create a  static JSON datai format to
        sawe temperatur data
162   JsonArray& root = jsonBuffer.createArray();
163
164   for (uint8_t i = 0; i < a_nrSensor; i++) {
165     JsonObject& nestedObject = root.createNestedObject();
166     char addrBuffer[17] = { 0 };
167     char tempBuffer[6] = { 0 };
168     uint8_t addrI = 0;
169     while (a_addr[addrIndex] != ';') {
170       addrBuffer[addrI] = a_addr[addrIndex];
171       ++addrI;
172       ++addrIndex;
173     }
174 //     Serial.println(addrBuffer);
175
176     uint8_t tempI = 0;
177     while (a_temp[tempIndex] != ';') {
178       tempBuffer[tempI] = a_temp[tempIndex];
179       ++tempI;
180       ++tempIndex;
181     }
182 //     Serial.println(tempBuffer);
183
184     nestedObject["ArrivalTime"] = a_timestmp;
185     nestedObject["TP"] = "TEMP_1";
186     nestedObject["Tag"] = addrBuffer;
187     nestedObject["Val"] = tempBuffer;
188
```

```
189
190     ++tempIndex;
191     ++addrIndex;
192   }
193   String dataJSON = "";
194   root.printTo(dataJSON);
195   return dataJSON;
196 }
197
198 void removeFile(String a_filename){
199   SPIFFS.begin();
200   SPIFFS.remove(a_filename);
201   SPIFFS.end();
202
203 }
```

fileSaver.h

```
/*
 * fileSaver.h
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#ifndef SRC_FILESAVER_H_
#define SRC_FILESAVER_H_

#include <Arduino.h>
#include <FS.h>
#include <ArduinoJson.h>
#include "globals.h"
#include "helpers.h"

/*
 * Initialize the SPIFFS, only for debugging required
 * */
void init_FS();

/*************************************************************
 * After startup delete all the saved files to avoid
 * overwriting. (For cases when files was not read and
 * deleted
 * ***********************************************************
 * */
void deletTemperatureFiles();

/*************************************************************
 * check the size of the uploaded files (Used for init_FS();)
 * ***********************************************************
 * */
String formatBytes(size_t bytes);

/*************************************************************
 * The function saves String data to file. If file existing
 * data will append, if not file will be created
 * ***********************************************************
 * */
void saveDataToFile(String a_fileName, String a_data);

/*************************************************************
 * Return data from File as String
 * ***********************************************************
 * */
```

```
47 String readDataFromFile(String a_fileName);
48
49 /*************************************************************
50  * Read configuration file and save to config structure
51  * variable
52  * ***********************************************************
53  * */
54 bool readConfigFile();
55
56 /*************************************************************
57  * save config structure variables to file
58  * ***********************************************************
59  * */
60 bool writeConfigFile();
61
62 /*************************************************************
63  * Create review table file for the web configuration
64  * ***********************************************************
65  * */
66 void createTable();
67
68 #endif /* SRC_FILESAVER_H_ */
```

fileSaver.cpp

```
/*
 * fileSaver.cpp
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#include "fileSaver.h"

void init_FS(){

  SPIFFS.begin();  // Start the SPI Flash File System (SPIFFS)
  PRINTLN("SPIFFS started. Contents:");

  Dir dir = SPIFFS.openDir("/");
  while (dir.next()) {                      // List the file system contents
    String fileName = dir.fileName();
    size_t fileSize = dir.fileSize();
    PRINTF("\tFS File: %s,  ", fileName.c_str());
    PRINTF("size: %s\r\n", formatBytes(fileSize).c_str());
  }
  PRINT("\n");
}

void deletTemperatureFiles(){

  SPIFFS.begin();  // Start the SPI Flash File System (SPIFFS)
  for(uint8_t i = 0; i < 48; i++){
    String file = fileNameToPath(i);
    if (SPIFFS.exists(file)) {
      SPIFFS.remove(file);
    }
  }
}

String formatBytes(size_t bytes) {
  if (bytes < 1024) {
    return String(bytes) + "B";
  } else if (bytes < (1024 * 1024)) {
    return String(bytes / 1024.0) + "KB";
  } else if (bytes < (1024 * 1024 * 1024)) {
    return String(bytes / 1024.0 / 1024.0) + "MB";
  } else {
    return String(bytes / 1024.0 / 1024.0 / 1024.0) + "GB";
  }
}
```

```
47
48 void saveDataToFile(String a_fileName, String a_data){
49
50   SPIFFS.begin();
51   if (SPIFFS.exists(a_fileName)) {
52     File file = SPIFFS.open(a_fileName, "a");
53     file.println(a_data);
54     file.close();
55   } else {
56     File file = SPIFFS.open(a_fileName, "w");
57     file.println(a_data);
58     file.close();
59   }
60   yield();
61
62 }
63
64 String readDataFromFile(String a_fileName){
65
66   SPIFFS.begin();
67   if (!SPIFFS.exists(a_fileName)){
68     PRINTLN("file dosn't exist");
69     return "";
70   }
71   File file = SPIFFS.open(a_fileName, "r");
72
73   String dataFile = "";
74   while(file.available()){
75     dataFile = file.readStringUntil('\0');
76   }
77   file.close();
78   SPIFFS.remove(a_fileName);
79   yield();
80
81   return dataFile;
82 }
83
84 bool readConfigFile() {
85
86   PRINTLN("Read configuration file...");
87   String configFile = readFromFile("/my_config_file.json");
88
89   if (configFile.length() < 100) {
90     configFile = readFromFile("/configBackup.json");
91   }
92
93   if (configFile != "NULL") {
94     StaticJsonBuffer<1024> jsonBuffer;
```

```
95      JsonObject& configRoot = jsonBuffer.parseObject(configFile);
96
97      if (!configRoot.success()) {
98        return false;
99      } else if (configRoot["config"]) {
100 //       configRoot.prettyPrintTo(Serial);
101
102       config.conf = configRoot["config"];
103       config.name = (const char*) configRoot["name"];
104       config.gsmId = configRoot["gsm_id"];
105
106       config.connectedTo = (const char*) configRoot["connected_to"];
107       config.wifiStrength = (const char*) configRoot["wifi_strength"];
108
109       config.ssid0 = (const char*) configRoot["ssid_0"];
110       config.pass0 = (const char*) configRoot["password_0"];
111       config.ssid1 = (const char*) configRoot["ssid_1"];
112       config.pass1 = (const char*) configRoot["password_1"];
113
114       config.apiUrl = (const char*) configRoot["api_url"];
115       config.apiKey = (const char*) configRoot["api_key"];
116
117       config.configDHT = configRoot["hum"];
118       config.configS0 = configRoot["s0"];
119       config.impulseID = (const char*) configRoot["s0_id"];
120       config.unitPImp = configRoot["unit_per_pulse"];
121
122       config.impulseVal = readFromFile("/counter.txt");
123       return true;
124     }
125   }
126   return false;
127 }
128
129 bool writeConfigFile() {
130
131   PRINTLN("Write configuration file...");
132   if (config.conf) {
133     StaticJsonBuffer<1024> jsonBuffer;
134     JsonObject& root = jsonBuffer.createObject();
135
136     root["config"] = config.conf;
137
138     root["name"] = config.name;
139     root["gsm_id"] = config.gsmId;
140     root["firmware_version"] = config.firmware;
141     root["connected_to"] = config.connectedTo;
142     root["wifi_strength"] = config.wifiStrength;
```

```
143
144     root["ssid_0"] = config.ssid0;
145     root["password_0"] = config.pass0;
146     root["ssid_1"] = config.ssid1;
147     root["password_1"] = config.pass1;
148     root["ssid_2"] = "";
149     root["password_2"] = "";
150
151     root["api_url"] = config.apiUrl;
152     root["api_key"] = config.apiKey;
153
154     root["number_of_temp_sensor"] = config.nrSensor;
155     root["hum"] = config.configDHT;
156     root["s0"] = config.configS0;
157     root["s0_id"] = config.impulseID;
158     root["s0_counter"] = config.impulseVal;
159     root["unit_per_pulse"] = config.unitPImp;
160
161     String configFile = "";
162     root.prettyPrintTo(configFile);
163
164     if (config.configS0) {
165       writeToFile(config.impulseVal, "/counter.txt");
166     }
167     return writeToFile(configFile, "/my_config_file.json");
168   } else {
169     return false;
170   }
171 }
172
173 void createTable() {
174
175   PRINTLN("Creating review table...");
176   createTableHead();
177
178   // read description file
179   uint32_t filePionter = 0;
180   SPIFFS.begin();
181   if (SPIFFS.exists("/description.txt")) {
182     File file = SPIFFS.open("/description.txt", "r");
183     String dataFile = "";
184     while (file.available()) {
185       dataFile = file.readString();
186     }
187     uint8_t l = 0;
188     while (dataFile[filePionter] != '\0') {
189       char buffer[30] = {0};
190       uint8_t m = 0;
```

```
191       while (dataFile[filePionter] != ';') {
192         buffer[m] = dataFile[filePionter];
193         ++filePionter;
194         ++m;
195       }
196       descrioption[l] = (String) buffer;
197       ++filePionter;
198       ++l;
199     }
200     file.close();
201   } else {
202     PRINTLN("file \"/description.txt\" dosn't exist");
203   }
204   // end read description file
205
206   uint8_t i = 0;
207
208   for (i = 0; i < sensor_1; i++) {
209     if (!sensors_1.getAddress(sensorAddress_1, i))
210       Serial.println("Unable to find address for Device");
211     float temp = 0;
212     String addr = "";
213     for (uint8_t j = 5; j > 0; j--) {
214       if (sensorAddress_1[j] < 16) {
215         addr = addr + '0';
216       }
217       addr = addr + String(sensorAddress_1[j], HEX);
218     }
219     sensors_1.setResolution(sensorAddress_1, PRECISION);
220     sensors_1.requestTemperaturesByAddress(sensorAddress_1);
221
222     String tp = "TEMP_1";
223     temp = sensors_1.getTempC(sensorAddress_1);
224
225     while (temp == 85.000000) {
226       temp = sensors_1.getTempC(sensorAddress_1);
227     }
228     createTableDetails(i + 1, config.gsmId, addr, temp, " Â°C ", tp,
229         descrioption[i]);
230   }
231
232   for (i = 0; i < sensor_2; i++) {
233     if (!sensors_2.getAddress(sensorAddress_2, i))
234       PRINTLN("Unable to find address for Device");
235     float temp = 0;
236     String addr = "";
237     for (uint8_t j = 5; j > 0; j--) {
238       if (sensorAddress_2[j] < 16) {
```

```
239         addr = addr + '0';
240       }
241       addr = addr + String(sensorAddress_2[j], HEX);
242     }
243     sensors_2.setResolution(sensorAddress_2, PRECISION);
244     sensors_2.requestTemperaturesByAddress(sensorAddress_2);
245
246     String tp = "TEMP_1";
247     temp = sensors_2.getTempC(sensorAddress_2);
248
249     while (temp == 85.000000) {
250       temp = sensors_2.getTempC(sensorAddress_2);
251     }
252     createTableDetails((i + 1 + sensor_1), config.gsmId, addr, temp, " Â°C ",
253         tp, descrioption[i + sensor_1]);
254   }
255
256   if (config.configDHT) {
257     DHT dht(DHT_PIN, DHT22);
258     dht.begin();
259     float humVector[4] = {0};
260
261     humVector[0] = dht.readHumidity();
262     humVector[1] = dht.readTemperature();
263
264     if(!isnan(humVector[0])){
265       calcTaupunkt(humVector);
266
267       createTableDetails(++i + sensor_1, config.gsmId, "DHT22AM220", humVector
    [0], " %",
268           "TEMP_1", "Relative Luftfeuchte");
269       createTableDetails(++i + sensor_1, config.gsmId, "DTT22AM230", humVector
    [1], " Â°C",
270           "TEMP_1", "Temperatur");
271       createTableDetails(++i + sensor_1, config.gsmId, "DHT22AM240", humVector
    [2], " Â°C",
272           "TEMP_1", "Taupunkttemperatur");
273       createTableDetails(++i + sensor_1, config.gsmId, "DHT22AM250", humVector
    [3],
274           " g/m^3", "TEMP_1", "Absolute Luftfeuchte");
275     }
276   }
277
278   if (config.configS0) {
279     float val = config.impulseVal.toFloat() * config.unitPImp;
280     createTableDetails(i + 1 + sensor_1, config.gsmId, config.impulseID, val,
281         "m^3", "GAS", "");
282   }
```

```
283 }
```

espServer.h

```
/*
 * espServer.h
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#ifndef SRC_ESPSERVER_H_
#define SRC_ESPSERVER_H_

#include <Arduino.h>
#include <ESP8266WebServer.h>
#include <ESP8266HTTPUpdateServer.h>
#include <WiFiClientSecure.h>
#include <FS.h>
#include "globals.h"
#include "fileSaver.h"

/************************************************************
 * Locale variables for the ESP8266 web server and
 * authentication for remote firmware update
 * ***********************************************************
 * */
extern ESP8266HTTPUpdateServer httpUpdater;
extern const char* updateUser;
extern const char* upcatePassword;
extern File file;

/************************************************************
 * Get the content type of a file
 * ***********************************************************
 * */
String getContentType(String filename);

/************************************************************
 * Send the requested file/web-page back to the client
 * ***********************************************************
 * */
bool handleFileRead(String a_path);

/************************************************************
 * Initialize the client requests
 * ***********************************************************
 * */
void initHandleClient();

```

```
47 /**************************************************************
48  * server handler
49  * ************************************************************
50  * */
51 void serverHandleClient();
52
53 /**************************************************************
54  * Upload handler, show the uploaded file name and size
55  * at serial monitor
56  * ************************************************************
57  * */
58 bool handleFileUpload();
59
60 /**************************************************************
61  * Save server arguments of sensor configuration entered in
62  * web configuration to config variables
63  * ************************************************************
64  * */
65 void saveSensorConfig();
66
67 /**************************************************************
68  * Save server arguments of network configuration entered in
69  * web configuration to config variables
70  * ************************************************************
71  * */
72 void saveNetworkConfig();
73
74 /**************************************************************
75  * Save server arguments of sensor description entered in web
76  * configuration to config variables
77  * ************************************************************
78  * */
79 void saveDescriptions();
80
81 #endif /* SRC_ESPSERVER_H_ */
```

espServer.cpp

```
/*
 * espServer.cpp
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#include "espServer.h"

ESP8266WebServer server(80);
ESP8266HTTPUpdateServer httpUpdater;
const char* updateUser  = "Messibox";
const char* upcatePassword = "messi_V3.0";
File file;

String getContentType(String filename) {
  if (server.hasArg("download"))
    return "application/octet-stream";
  else if (filename.endsWith(".htm"))
    return "text/html";
  else if (filename.endsWith(".html"))
    return "text/html";
  else if (filename.endsWith(".css"))
    return "text/css";
  else if (filename.endsWith(".js"))
    return "application/javascript";
  else if (filename.endsWith(".png"))
    return "image/png";
  else if (filename.endsWith(".gif"))
    return "image/gif";
  else if (filename.endsWith(".jpg"))
    return "image/jpeg";
  else if (filename.endsWith(".ico"))
    return "image/x-icon";
  else if (filename.endsWith(".xml"))
    return "text/xml";
  else if (filename.endsWith(".pdf"))
    return "application/x-pdf";
  else if (filename.endsWith(".zip"))
    return "application/x-zip";
  else if (filename.endsWith(".gz"))
    return "application/x-gzip";
  return "text/plain";
}

bool handleFileRead(String a_path) {
```

```
47
48    if (a_path.endsWith("/"))
49      a_path = "/PAGE_main.html"; // If a folder is requested, send the main file
50    String contentType = getContentType(a_path);       // Get the MIME type
51    if (a_path == "/PAGE_main.html") {
52      readConfigFile();    // get config data and save to file
53      createTable();         // create table to file
54    }
55    SPIFFS.begin();
56    if (SPIFFS.exists(a_path)) {                            // If the file exists
57      File file = SPIFFS.open(a_path, "r");                 // Open it
58
59
60      server.streamFile(file, contentType); // And send it to the client
61      file.close();                                 // Then close the file again
62
63      if (a_path == "/PAGE_saveNetworkConf.html") {
64        saveNetworkConfig();
65        writeConfigFile();
66        PRINTLN("Save and write network configurations");
67      }
68      if (a_path == "/PAGE_saveSensorConf.html") {
69        saveSensorConfig();
70        writeConfigFile();
71        PRINTLN("Save and write sensor configurations");
72      }
73      if (a_path == "/PAGE_saveDescription.html") {
74        saveDescriptions();
75        writeConfigFile();
76        PRINTLN("Save and write sensor descriptions");
77      }
78      if (a_path == "/PAGE_restart.html") {
79        deletTemperatureFiles();
80        writeToFile((String)0, PATH_FILE_COUNTER);
81        writeToFile((String)0, PATH_ROW_COUNTER);
82        writeToFile((String)0, PATH_SAVED_DATA);
83        PRINTLN("Restart ESP8266");
84        ESP.restart();
85      }
86      return true;
87    }
88    return false;                     //If the file doesn't exist, return false
89  }
90
91  void initHandleClient() {
92
93    SPIFFS.begin();
94
```

```
95    server.on("/", HTTP_GET, []() {
96      if(!handleFileRead("/PAGE_main.html")) {
97        server.send(404, "text/plain", "FileNotFound");
98      }
99    });
100
101   server.on("/upload", HTTP_GET, [&]() { // if the client requests the upload
        page
102       if (!handleFileRead("/upload.html"))// send it if it exists
103       server.send(404, "text/plain", "404: Not Found");// otherwise, respond
      with a 404 (Not Found) error
104     });
105   server.on("/upload", HTTP_POST,     // if the client posts to the upload page
106       []() {server.send(200);}, // Send status 200 (OK) to tell the client we are
       ready to receive
107       handleFileUpload                          // Receive and save the file
108       );
109
110   // handle any pages
111   server.onNotFound([&]() {
112     if(!handleFileRead(server.uri())) {
113       server.send(404, "text/plain", "FileNotFound");
114     }
115   });
116
117   httpUpdater.setup(&server, updateUser, upcatePassword);
118   server.begin();
119 }
120
121 void serverHandleClient() {
122   server.handleClient();
123 }
124
125 bool handleFileUpload() {
126
127   HTTPUpload& upload = server.upload();
128
129   if (upload.status == UPLOAD_FILE_START) {
130     String filename = upload.filename;
131     if (!filename.startsWith("/"))
132       filename = "/" + filename;
133
134     PRINT("handleFileUpload Name: ");
135     PRINTLN(filename);
136
137     file = SPIFFS.open(filename, "w"); // Open the file for writing in SPIFFS (
      create if it doesn't exist)
138     filename = String();
```

```
139   } else if (upload.status == UPLOAD_FILE_WRITE) {
140     if (file)
141       file.write(upload.buf, upload.currentSize); // Write the received bytes to
      the file
142   } else if (upload.status == UPLOAD_FILE_END) {
143     if (file) {                    // If the file was successfully created
144       file.close();                          // Close the file again
145
146       PRINT("handleFileUpload Size: ");
147       PRINTLN(upload.totalSize);
148
149       server.sendHeader("Location", "/success.html"); // Redirect the client to
      the success page
150       server.send(303);
151       return true;
152     } else {
153       server.send(500, "text/plain", "500: couldn't create file");
154       return false;
155     }
156   }
157   init_FS();
158   return true;
159 }
160
161 void saveSensorConfig() {
162   if (server.args() > 0) {
163     config.conf = true;
164
165     if (server.arg("humidity") == "hum") {
166       config.configDHT = true;
167     } else {
168       config.configDHT = false;
169     }
170     if (server.arg("s0") == "s0") {
171       config.configS0 = true;
172     } else {
173       config.configS0 = false;
174     }
175     if (server.arg("gaszaehler_id") != "") {
176       config.impulseID = (server.arg("gaszaehler_id"));
177     }
178     if (server.arg("gaszaehler") != "") {
179       config.impulseVal = server.arg("gaszaehler");
180     }
181     if (server.arg("einheit_impulse") != "") {
182       config.unitPImp = (server.arg("einheit_impulse")).toFloat();
183     }
184     for (uint8_t i = 0; i < 20; i++) {
```

```
185       if (config.name[i] == '+') {
186         config.name[i] = ' ';
187       }
188     }
189   } else {
190     config.conf = false;
191   }
192 }
193
194 void saveNetworkConfig() {
195   if (server.args() > 0) {
196     config.conf = true;
197
198     if (server.arg("name") != "") {
199       config.name = server.arg("name");
200       PRINTF("Name: %s \n", config.name.c_str());
201     }
202     if (server.arg("gsm_id") != "") {
203       config.gsmId = (server.arg("gsm_id")).toInt();
204     }
205     if (server.arg("ssid") != "") {
206       Serial.println(server.arg("ssid"));
207       config.ssid0 = (server.arg("ssid"));
208     }
209     if (server.arg("password") != "") {
210       config.pass0 = (server.arg("password"));
211       Serial.println(server.arg("password"));
212     }
213     if (server.arg("host") != "") {
214       config.apiUrl = server.arg("host");
215     }
216     if (server.arg("api_key") != "") {
217       config.apiKey = server.arg("api_key");
218     }
219     for (uint8_t i = 0; i < 20; i++) {
220       if (config.name[i] == '+') {
221         config.name[i] = ' ';
222       }
223     }
224   } else {
225     config.conf = false;
226   }
227 }
228
229 void saveDescriptions() {
230   if (server.args() > 0) {
231     config.conf = true;
232
```

```
233     if (server.arg("1_id") != "") {
234       descrioption[0] = (server.arg("1_id"));
235     }
236     if (server.arg("2_id") != "") {
237       descrioption[1] = (server.arg("2_id"));
238     }
239     if (server.arg("3_id") != "") {
240       descrioption[2] = (server.arg("3_id"));
241     }
242     if (server.arg("4_id") != "") {
243       descrioption[3] = (server.arg("4_id"));
244     }
245     if (server.arg("5_id") != "") {
246       descrioption[4] = (server.arg("5_id"));
247     }
248     if (server.arg("6_id") != "") {
249       descrioption[5] = (server.arg("6_id"));
250     }
251     if (server.arg("7_id") != "") {
252       descrioption[6] = (server.arg("7_id"));
253     }
254     if (server.arg("8_id") != "") {
255       descrioption[7] = (server.arg("8_id"));
256     }
257     if (server.arg("9_id") != "") {
258       descrioption[8] = (server.arg("9_id"));
259     }
260     if (server.arg("10_id") != "") {
261       descrioption[9] = (server.arg("10_id"));
262     }
263     if (server.arg("11_id") != "") {
264       descrioption[10] = (server.arg("11_id"));
265     }
266     if (server.arg("12_id") != "") {
267       descrioption[11] = (server.arg("12_id"));
268     }
269     if (server.arg("13_id") != "") {
270       descrioption[12] = (server.arg("13_id"));
271     }
272     if (server.arg("14_id") != "") {
273       descrioption[13] = (server.arg("14_id"));
274     }
275     if (server.arg("15_id") != "") {
276       descrioption[14] = (server.arg("15_id"));
277     }
278     if (server.arg("16_id") != "") {
279       descrioption[15] = (server.arg("16_id"));
280     }
```

```
281     if (server.arg("17_id") != "") {
282       descrioption[16] = (server.arg("17_id"));
283     }
284     if (server.arg("18_id") != "") {
285       descrioption[17] = (server.arg("18_id"));
286     }
287     if (server.arg("19_id") != "") {
288       descrioption[18] = (server.arg("19_id"));
289     }
290     if (server.arg("20_id") != "") {
291       descrioption[19] = (server.arg("20_id"));
292     }
293
294   }
295   SPIFFS.begin();
296   if (SPIFFS.exists("/description.txt")) {
297     SPIFFS.remove("/description.txt");
298   }
299   File file = SPIFFS.open("/description.txt", "w");
300   for (uint8_t i = 0; i < (sensor_1 + sensor_2); i++) {
301     file.print(descrioption[i]);
302     file.print(';');
303   }
304   file.println('\0');
305   file.close();
306 }
```

wifiConnection.h

```
/*
 * wifiConnection.h
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#ifndef SRC_WIFICONNECTION_H_
#define SRC_WIFICONNECTION_H_

#include <Arduino.h>
#include <ESP8266WiFiMulti.h>

#include "globals.h"
#include "fileSaver.h"

/*************************************************************
 * Define locale instance for WiFi multi connection
 * ***********************************************************
 * */
extern ESP8266WiFiMulti multi;

/*************************************************************
 * Get configuration details from file and save at first,
 * then try to connect to one of the saved WiFi networks
 * ***********************************************************
 * */
bool wifiConnection();

/*************************************************************
 * Check if server request is reachable
 * ***********************************************************
 * */
bool isConnected();

/*************************************************************
 * Reconnect WiFi in case of connection is lost
 * ***********************************************************
 * */
void wifiReconnect();

/*************************************************************
 * Print to Serial monitor the IP address and the WiFi
 * strength
 * ***********************************************************
 * */
```

```
47 void printWiFiInfo();
48
49 #endif /* SRC_WIFICONNECTION_H_ */
```

wifiConnection.cpp

```
/*
 * wifi.connection.cpp
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#include "wifiConnection.h"

ESP8266WiFiMulti multi;

bool wifiConnection() {

  if (readConfigFile()) {
    WiFi.mode(WIFI_STA);

    multi.addAP(config.ssid0.c_str(), config.pass0.c_str());
    multi.addAP(config.ssid1.c_str(), config.pass1.c_str());

    // try to connect, after 20 seconds break and continue without connection
    for (uint8_t i = 0; (i < 20 && multi.run() != WL_CONNECTED); i++) {
      delay(1000);
      Serial.print(".");
    }
    if (multi.run() == WL_CONNECTED) {
      config.connectedTo = WiFi.SSID();
      config.wifiStrength = (String) WiFi.RSSI() + "dB";
      Serial.print("\n\nConnected to: ");
      Serial.println(config.connectedTo);
      Serial.print("IP Address: ");
      Serial.println(WiFi.localIP());
      return true;
    } else {
      PRINTLN("Unable to connect!!!");
      return false;
    }
  }
  return false;
}

void wifiReconnect() {

  WiFi.mode(WIFI_STA);
  multi.addAP(config.ssid0.c_str(), config.pass0.c_str());
  multi.addAP(config.ssid1.c_str(), config.pass1.c_str());
  // try to connect, after 30 seconds break and continue without connection
```

```
47    for (uint8_t i = 0; (i < 20 && multi.run() != WL_CONNECTED); i++) {
48      delay(1000);
49      PRINT(".");
50    }
51  }
52
53  bool isConnected() {
54
55    WiFiClientSecure client;
56    if (client.connect(config.apiUrl, PORT)) {
57      client.stop();
58      return true;
59    } else {
60      return false;
61    }
62  }
63
64  void printWiFiInfo() {
65
66    PRINT("\nconnected to: ");
67    PRINTLN(config.connectedTo);
68    PRINT("IP Address: ");
69    PRINTLN(WiFi.localIP());
70    PRINT("Wifi Strenght: ");
71    PRINTLN(config.wifiStrength);
72  }
```

timeServer.h

```
/*
 * timeServer.h
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#ifndef SRC_TIMESERVER_H_
#define SRC_TIMESERVER_H_

#include <Arduino.h>
#include <WiFiUdp.h>
#include <TimeLib.h>
#include <ESP8266WiFiMulti.h>
#include "globals.h"

/*-------------------------------------------------------------
 * Start UDP at locale port 8888
 *-------------------------------------------------------------
 * */
void initUDP();

/*-------------------------------------------------------------
 * Send NTP packet to the UDP server for request time
 *-------------------------------------------------------------
 * */
void sendNTPpacket(IPAddress &address);

/*-------------------------------------------------------------
 * Add leading zeros to converted time and date
 *-------------------------------------------------------------
 * */
String digits(int digits);

/*-------------------------------------------------------------
 * Catch the UDP server response and convert it to a second
 * (seconds since 1900)
 *-------------------------------------------------------------
 * */
time_t getNTPTime();

/*-------------------------------------------------------------
 * Convert the seconds in to a right date format
 * JJJJ-MM-DD HH:MM:SS
 *-------------------------------------------------------------
 * */
```

```
47 String getTimeStemp(time_t a_timeInMS);
48
49 #endif /* SRC_TIMESERVER_H_ */
```

timeServer.cpp

```
/*
 * timeServer.cpp
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#include "timeServer.h"

void initUDP() {
  udp.begin(LOCALE_PORT);
}

void sendNTPpacket(IPAddress &address) {
  memset(packetBuffer, 0, NTP_PACKET_SIZE);
  // Initialize values needed to form NTP request
  // (see URL above for details on the packets)
  packetBuffer[0] = 0b11100011;   // LI, Version, Mode
  packetBuffer[1] = 0;     // Stratum, or type of clock
  packetBuffer[2] = 6;     // Polling Interval
  packetBuffer[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  packetBuffer[12] = 49;
  packetBuffer[13] = 0x4E;
  packetBuffer[14] = 49;
  packetBuffer[15] = 52;
  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:
  udp.beginPacket(address, 123); //NTP requests are to port 123
  udp.write(packetBuffer, NTP_PACKET_SIZE);
  udp.endPacket();
}

time_t getNTPTime() {

  IPAddress ntpServerIP; // NTP server's ip address

  while (udp.parsePacket() > 0)
    ; // discard any previously received packets
  PRINTLN("Transmit NTP Request");
    // get a random server from the pool
  WiFi.hostByName(ntpServerName.c_str(), ntpServerIP);
//  Serial.print(ntpServerName);
//  Serial.print(": ");
//  Serial.println(ntpServerIP);
  sendNTPpacket(ntpServerIP);
```

```
47   uint32_t beginWait = millis();
48
49   while (millis() - beginWait < 1500) {
50     int size = udp.parsePacket();
51
52     if (size >= NTP_PACKET_SIZE) {
53       PRINTLN("Receive NTP Response");
54       udp.read(packetBuffer, NTP_PACKET_SIZE); // read packet into the buffer
55       unsigned long secsSince1900;
56       // convert four bytes starting at location 40 to a long integer
57       secsSince1900 = (unsigned long) packetBuffer[40] << 24;
58       secsSince1900 |= (unsigned long) packetBuffer[41] << 16;
59       secsSince1900 |= (unsigned long) packetBuffer[42] << 8;
60       secsSince1900 |= (unsigned long) packetBuffer[43];
61       time_t m_timeInMS = secsSince1900 - 2208988800UL;
62       return m_timeInMS;
63     }
64   }
65   PRINTLN("No NTP Response :-(");
66   return 0; // return 0 if unable to get the time
67 }
68
69 String digits(int digits) {
70   String _digits = "";
71   if (digits < 10) {
72     _digits += "0";
73   }
74   _digits += String(digits);
75
76   return _digits;
77 }
78
79 String getTimeStemp(time_t a_timeInMS) {
80
81   String stmp = "";
82   stmp = String(year(a_timeInMS));
83   stmp += "-";
84   stmp += digits(month(a_timeInMS));
85   stmp += "-";
86   stmp += digits(day(a_timeInMS));
87   stmp += " ";
88   stmp += digits(hour(a_timeInMS));
89   stmp += ":";
90   stmp += digits(minute(a_timeInMS));
91   stmp += ":";
92   stmp += digits(second(a_timeInMS));
93
94   return stmp;
```

```
95 }
```

temperatureSensors.h

```
/*
 * temperatureSensors.h
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#ifndef SRC_TEMPERATURESENSORS_H_
#define SRC_TEMPERATURESENSORS_H_

#include <Arduino.h>
#include <Ticker.h>
#include <WiFiClientSecure.h>
#include "globals.h"
#include "helpers.h"
#include "fileSaver.h"
#include "timeServer.h"

/*************************************************************
 * Timer instance for timer interrupt
 * ***********************************************************
 * */
extern Ticker interrupt;

/*************************************************************
 * Setup temperature sensors and save the number of sensors
 * for both busses
 * ***********************************************************
 * */
void sensorBegin();

/*************************************************************
 * Get temperature values, convert it in JSON format and
 * return as String
 * ***********************************************************
 * */
String getTemperatures(bool a_connected, uint8_t a_nrSensor, DallasTemperature
    a_sensors, DeviceAddress a_sensorAddress);

/*************************************************************
 * Get temperature addresses for saved values to recreate
 * JSON String
 * ***********************************************************
 * */
String getAddresses(uint8_t a_nrSensor, DallasTemperature a_sensors,
    DeviceAddress a_sensorAddress);
```

```
45
46 /*************************************************************
47  * Get humidity values, convert it in JSON format and
48  * return as String
49  * ***********************************************************
50  * */
51 String getHumidity(bool a_connected);
52
53 /*************************************************************
54  * Make server request and send the JSON string to the
55  * EZ-N server
56  * ***********************************************************
57  * */
58 bool request(String a_data);
59
60 void saveTest();
61
62 /*************************************************************
63  *  If no Internet connection available, get temperature
64  *  values only, without tag's and ID's and save it in to the
65  *  SPIFFS. Create 48 data files, each file has 60 rows.
66  *  Save data for 24 hours.
67  * ***********************************************************
68  * */
69 void saveDataToSD();
70
71 /*************************************************************
72  * Open saved data files, recreate the right JSON Format and
73  * make a request to send data to the server
74  * ***********************************************************
75  * */
76 void readDataFromSD();
77
78 /*************************************************************
79  *  Send saved data may takes longer then one minute. In one
80  *  minute interval make an interrupt to get new temperature
81  *  data and send it to the server as well
82  * ***********************************************************
83  * */
84 void interruptHandle();
85
86 #endif /* SRC_TEMPERATURESENSORS_H_ */
```

temperatureSensors.cpp

```
/*
 * temperatureSensore.cpp
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#include "temperatureSensors.h"

Ticker interrupt;

void sensorBegin() {

  PRINTLN("Sensor 1 Begin");
  sensors_1.setWaitForConversion(false); // Don't block the program while the
    temperature sensor is reading
  sensors_1.begin();
  sensor_1 = sensors_1.getDeviceCount();
  config.nrSensor = sensor_1;

  PRINTLN("Sensor 2 Begin");
  sensors_2.setWaitForConversion(false); // Don't block the program while the
    temperature sensor is reading
  sensors_2.begin();
  sensor_2 = sensors_2.getDeviceCount();
  config.nrSensor = sensor_1 + sensor_2;
}

String getTemperatures(bool a_connected, uint8_t a_nrSensor,
    DallasTemperature a_sensors, DeviceAddress a_sensorAddress) {

  if (a_nrSensor == 0) {
    return "NULL";
  }
  StaticJsonBuffer < 2024 > jsonBuffer; // create a  static JSON datai format to
    sawe temperatur data
  JsonArray& root = jsonBuffer.createArray();
  String dataJSON = "";

  for (uint8_t i = 0; i < a_nrSensor; i++) {
    JsonObject& nestedObject = root.createNestedObject();
    if (!a_sensors.getAddress(a_sensorAddress, i)) {
      PRINTLN("Unable to find address for Device");
      ESP.restart();
    }
    float temp = 0;
```

```
44      String addr = "";
45
46      for (uint8_t j = 5; j > 0; j--) {
47        if (a_sensorAddress[j] < 16) {
48          addr = addr + '0';
49        }
50        addr = addr + String(a_sensorAddress[j], HEX);
51      }
52      a_sensors.setResolution(a_sensorAddress, PRECISION);
53      a_sensors.requestTemperaturesByAddress(a_sensorAddress);
54
55      temp = a_sensors.getTempC(a_sensorAddress);
56      while (temp == 85.000000) {
57        temp = a_sensors.getTempC(a_sensorAddress);
58      }
59      if (!a_connected) {
60        dataJSON += (String) temp + ";";
61      } else {
62        nestedObject["Val"] = temp;
63        nestedObject["TP"] = "TEMP_1";
64        nestedObject["Tag"] = addr;
65      }
66    }
67    if (a_connected) {
68      root.printTo(dataJSON);
69    }
70    return dataJSON;
71  }
72
73  String getAddresses(uint8_t a_nrSensor, DallasTemperature a_sensors,
74      DeviceAddress a_sensorAddress) {
75
76    if (a_nrSensor == 0) {
77      return "NULL";
78    }
79
80    String dataAddr;
81    for (uint8_t i = 0; i < a_nrSensor; i++) {
82
83      if (!a_sensors.getAddress(a_sensorAddress, i)) {
84        PRINTLN("Unable to find address for Device");
85        ESP.restart();
86      }
87
88      String addr = "";
89
90      for (uint8_t j = 5; j > 0; j--) {
91        if (a_sensorAddress[j] < 16) {
```

```
92          addr = addr + '0';
93        }
94        addr = addr + String(a_sensorAddress[j], HEX);
95      }
96      dataAddr += addr + ";";
97    }
98    return dataAddr;
99  }
100
101 String getHumidity(bool a_connected) {
102
103   DHT dht(DHT_PIN, DHT22);
104   dht.begin();
105   float humVector[4] = { 0 };
106
107   humVector[0] = dht.readHumidity();
108   humVector[1] = dht.readTemperature();
109
110   if (isnan(humVector[0]) || !config.configDHT) {
111     return "NULL";
112   }
113   calcTaupunkt(humVector);
114   String dataJSON = "";
115
116   if (a_connected) {
117     StaticJsonBuffer < 512 > jsonBuffer;
118     JsonArray& root = jsonBuffer.createArray();
119
120     String humTempID[4] = { "DHT22AM220", "DTT22AM230", "DHT22AM240",
121         "DHT22AM250" };
122     for (uint8_t i = 0; i < 4; i++) {
123       JsonObject& nestedObject = root.createNestedObject();
124       nestedObject["Val"] = humVector[i];
125       nestedObject["TP"] = "TEMP_1";
126       nestedObject["Tag"] = humTempID[i];
127     }
128     root.printTo(dataJSON);
129     return dataJSON;
130   } else {
131     dataJSON = (String) humVector[0] + ";" + (String) humVector[1] + ";"
132         + (String) humVector[2] + ";" + (String) humVector[3] + ";";
133     return dataJSON;
134   }
135 }
136
137 bool request(String a_data) {
138
139 //  PRINTLN("request server...");
```

```
140   uint16_t len = a_data.length();
141
142   WiFiClientSecure client;
143   if (client.connect(config.apiUrl, PORT)) {
144     String request = "PUT /api/v1/datapoints/";
145     request += (String) config.gsmId;
146     request += "?APIKEY=";
147     request += config.apiKey;
148     request += "&count=1000 HTTP/1.1\r\n";
149     request += "HOST: ";
150     request += config.apiUrl;
151     request += "\r\n";
152     request += "Connection: close\r\n";
153     request += "content-length: ";
154     request += len;
155     request += "\r\n";
156     request += "content-type: application/json\r\n";
157     request += "cookie: PHPSESSID=bjuu56sed21qgtqvottbtscoc7\r\n\r\n";
158     request += a_data;
159     request += "\n";
160
161     client.print(request);
162
163 //     while (client.connected()) {
164 //       if (client.available()) {
165 //         String line = client.readStringUntil('\n');
166 //         Serial.println(line);
167 //       }
168 //     }
169     PRINTLN("request OK ");
170     return true;
171
172   } else {
173     PRINTLN("request is NOT available ");
174     return false;
175   }
176 }
177
178 void saveTest() {
179   PRINTLN("save data test...");
180   writeToFile((String)1, PATH_SAVED_DATA);
181   String temp_1 = getTemperatures(false, sensor_1, sensors_1,
182       sensorAddress_1);
183   yield();
184   String temp_2 = getTemperatures(false, sensor_2, sensors_2,
185       sensorAddress_2);
186   yield();
187   String hum = getHumidity(false);
```

```
188   yield();
189
190   String filename = "";
191   for (uint8_t i = 0; i <= 1; i++) {
192     filename = fileNameToPath(i);
193     PRINTF("File: %s \n", filename.c_str());
194     for (uint8_t j = 0; j <= 5; j++) {
195       saveDataToFile(filename, temp_1);
196       yield();
197       saveDataToFile(filename, temp_2);
198       yield();
199     }
200     ++fileCounter;
201   }
202 }
203
204 void saveDataToSD() {
205
206   writeToFile((String)1, PATH_SAVED_DATA);
207
208   // get file counter
209   fileCounter = readFromFile(PATH_FILE_COUNTER).toInt();
210   rowCounter = readFromFile(PATH_ROW_COUNTER).toInt();
211
212   if (fileCounter > MAX_FILE) {
213     PRINTLN("Max file reached");
214     return;
215   }
216   PRINTLN("Request is NOT available!!!");
217
218   // each file contains max 60 rows (if more then 10 sensor, 120 rows)
219   // then open up an other file and start to count the rows again
220   String filename = "";
221   if (rowCounter < MAX_ROW) {
222     filename = fileNameToPath(fileCounter);
223   } else {
224     rowCounter = 0;
225     ++fileCounter;
226     writeToFile((String)fileCounter, PATH_FILE_COUNTER);
227     filename = fileNameToPath(fileCounter);
228   }
229   // get temperature values in a format to save in file
230   String temp = getTemperatures(false, sensor_1, sensors_1,
231       sensorAddress_1);
232   if (temp != "NULL") {
233     saveDataToFile(filename, temp);
234     ++rowCounter;
235     writeToFile((String)rowCounter, PATH_ROW_COUNTER);
```

```
236
237     PRINTF("Save temp data to file: %s \n", filename.c_str());
238     PRINTF("Row number: %d \n", rowCounter);
239     PRINTF("temp_1: %s \n", temp.c_str());
240     yield();
241   }
242   temp = "";
243   temp = getTemperatures(false, sensor_2, sensors_2,
244       sensorAddress_2);
245   if (temp != "NULL") {
246     saveDataToFile(filename, temp);
247     PRINTF("temp_2: %s \n", temp.c_str());
248     yield();
249   }
250
251 //   **********Not suported yet **********
252 //   readDataFromSD() and saveDataToSD() need to be changed
253 //   for getting humidity value also saved
254 //
255 //   String hum = getHumidity(false);
256 //   if (hum != "NULL") {
257 //     humidity = true;
258 //     saveDataToFile(filename, hum);
259 //     PRINT("hum: ");
260 //     PRINTLN(hum);
261 //     yield();
262 //   }
263
264
265 }
266
267 void readDataFromSD() {
268
269   PRINTLN("Read sensore data from file");
270
271   String addr_1 = getAddresses(sensor_1, sensors_1, sensorAddress_1);
272   yield();
273   String addr_2 = getAddresses(sensor_2, sensors_2, sensorAddress_2);
274   yield();
275   String filename = "";
276
277   fileCounter = readFromFile(PATH_FILE_COUNTER).toInt();
278   PRINTF("Saved files: %d \n", fileCounter);
279
280   // for loop to go trough all the file
281   for (uint8_t j = 0; j <= fileCounter; j++) {
282     filename = fileNameToPath(j);
283     String dataFile = readDataFromFile(filename);
```

```
284     uint8_t counter = 0;
285     filePionter = 0;
286 
287     PRINTF("Get temp data: %s \n", filename.c_str());
288 //    PRINTLN(dataFile);
289 
290     while (dataFile[filePionter] != '\0') {
291       ++counter;
292       char lineBuffer[100] = { 0 };
293       uint8_t i = 0;
294       timestmp = getTimeStemp(preTime += 60);
295       yield();
296       while (dataFile[filePionter] != '\n') {
297         lineBuffer[i] = dataFile[filePionter];
298         ++i;
299         ++filePionter;
300       }
301 
302 //      PRINTLN(lineBuffer);
303       PRINTLN(counter);
304 
305       // Make a decision if there are more then 10 sensors
306       // If there is more then 10 sensors, every second line is for
307       // the second BUS (temp_2)
308       if (sensor_2) {
309 
310         if (counter % 2) {
311           String data = convertStringToJSON(sensor_1, timestmp,
312               addr_1, lineBuffer);
313           request(data);
314           yield();
315           PRINTLN("request for temp_1");
316         } else {
317           String data = convertStringToJSON(sensor_2, timestmp,
318               addr_2, lineBuffer);
319           request(data);
320           yield();
321           PRINTLN("request for temp_2");
322         }
323       } else if (sensor_1) {
324         String data = convertStringToJSON(sensor_1, timestmp, addr_1,
325             lineBuffer);
326         request(data);
327         yield();
328         PRINTLN("request for temp_1");
329       }
330 //      Interrupt request temperature sensors if reading data took more then one
    minute
```

```
331       if (interruptFlag) {
332
333         PRINTLN("Interrupt request between reading out data!");
334         interruptFlag = false;
335         String temp = getTemperatures(true, sensor_1, sensors_1,
336             sensorAddress_1);
337         if (temp != "NULL") {
338           request(temp);
339         }
340         temp = "";
341         temp = getTemperatures(true, sensor_2, sensors_2,
342             sensorAddress_2);
343         if (temp != "NULL") {
344           request(temp);
345         }
346         yield();
347       }
348       ++filePionter;
349     }
350   }
351
352   writeToFile((String)0, PATH_FILE_COUNTER);
353   writeToFile((String)0, PATH_ROW_COUNTER);
354   writeToFile((String)0, PATH_SAVED_DATA);
355 }
356
357 void interruptHandle(void) {
358   interruptFlag = true;
359
360 }
```

impulseCounter.h

```
/*
 * impulseCounter.h
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#ifndef SRC_IMPULSECOUNTER_H_
#define SRC_IMPULSECOUNTER_H_

#include <Arduino.h>
#include "globals.h"
#include "fileSaver.h"

/*************************************************************
 * Declare locale static variables for count the pulses
 * ***********************************************************
 * */
static constexpr uint8_t INTERRUPT_PIN = 5;
extern bool highEvent;
extern bool lowEvent;
extern uint8_t eventCounter;

/*************************************************************
 * This is the counter handler for interrupt
 * Check for the change of Pin states and count the impulse
 * ***********************************************************
 * */
void counterHandler();

/*************************************************************
 * Setup the PIN and the handler for Interrupt
 * ***********************************************************
 * */
void InterruptHandling();

/*************************************************************
 * If interrupt occurred, update the counter value
 * Counter value is saved in config JSON and in a separate
 * txt files
 * ***********************************************************
 * */
void updateS0();

/*************************************************************
 * return impulse value in JSON format for PUT request
```

```
47  * ************************************************************
48  * */
49 String returnImpulse();
50 
51 /* ************************************************************
52  * return impulse value in JSON format for PUT request
53  * ************************************************************
54  * */
55 
56 #endif /* SRC_IMPULSECOUNTER_H_ */
```

impulseCounter.cpp

```
/*
 * impulseCounter.cpp
 *
 *  Created on: 05.02.2019
 *      Author: JozsefGergely
 */

#include "impulseCounter.h"

bool highEvent;
bool lowEvent;
uint8_t eventCounter;

void counterHandler() {

  detachInterrupt(INTERRUPT_PIN);

  lowEvent = true; // this two variable must be set back after updating values
  eventCounter++;
  delay(5);
  Serial.printf("low event, eventCounter: %d\n", eventCounter);

  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), counterHandler,
  FALLING);
}


void InterruptHandling() {

  pinMode(INTERRUPT_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), counterHandler,
  FALLING);
}

void updateS0() {
  if (config.configS0) {
    if (lowEvent) {
      lowEvent = false;
      readConfigFile();
      uint32_t s0Int = config.impulseVal.toInt();
      s0Int = s0Int + eventCounter;
      config.impulseVal = (String) s0Int;
      PRINTF(" update S0: %d\n", s0Int);
      config.conf = true;
      writeConfigFile();
      eventCounter = 0;
```

```
47      }
48    }
49  }
50
51  String returnImpulse() {
52
53    if (!config.configS0) {
54      return "NULL";
55    }
56    config.impulseVal = readFromFile("/counter.txt");
57    float val = config.impulseVal.toFloat() * config.unitPImp;
58    StaticJsonBuffer<512> jsonBuffer;
59    JsonArray& root = jsonBuffer.createArray();
60    JsonObject& nestedObject = root.createNestedObject();
61
62    nestedObject["Val"] = val;
63    nestedObject["TP"] = "GAS";
64    nestedObject["Tag"] = config.impulseID;
65
66    String data = "";
67    root.printTo(data);
68
69    return data;
70  }
```

C. HTML Source

PAGE_main.html

```
1
2  <!DOCTYPE html>
3  <html>
4    <head>
5      <meta charset="UTF-8">
6
7      <script type="text/javascript" src="https://
           ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js
           "></script>
8      <title> MessibBox </title>
9      <figure>
10       <img src = "Energiezentrale_logo.png" alt = "logo" align
             ="right">
11     </figure>
12     <style type = "text/css">
13       body {
14         font-family: "Lato", sans-serif;
15         <--!background-color: #e6e6e6;-->
16
17       }
18
19       fieldset {
20         width: 960px;
21         background-color:  #f2f2f2;
22         border-radius: 3px;
23       }
24       button.tablinks {
25         font-size:20px;
26         font-weight: bold;
27       }
28
29       table.sensor{
```

```
30          width: 100%;
31          border-collapse:collapse;
32
33        }
34
35        table.sensor td, table.sensor th{
36          border: 1px solid #dddddd;
37          text-align: left;
38          padding: 3px;;
39        }
40
41        table.sensor tr:nth-child(even) {
42          background-color: #ffffff;
43        }
44        .wrapper{
45          max-width:1009px;
46          margin:0 auto;
47        }
48
49        /* Style the tab */
50        div.tab {
51          overflow: hidden;
52          border: 1px solid #ccc;
53          background-color: #f1f1f1;
54        }
55
56        /* Style the buttons inside the tab */
57        div.tab button {
58          background-color: inherit;
59          float: left;
60          border: 1px solid #000000;
61          outline: none;
62          cursor: pointer;
63          padding: 14px 16px;
64          transition: 0.3s;
65          width: 250px;
66        }
67
68        /* Change background color of buttons on hover */
69        div.tab button:hover {
70          background-color: #ddd;
71        }
72
73        /* Create an active/current tablink class */
```

```
74        div.tab button.active {
75          background-color: #ccc;
76        }
77
78        /* Style the tab content */
79        .tabcontent {
80          display: none;
81          padding: 6px 12px;
82          border: 1px solid #ccc;
83          border-top: none;
84        }
85
86        /* Fade in tabs */
87        .tabcontent {
88          -webkit-animation: fadeEffect 1s;
89          animation: fadeEffect 1s; /* Fading effect takes 1
               second */
90        }
91        @-webkit-keyframes fadeEffect {
92          from {opacity: 0;}
93          to {opacity: 1;}
94        }
95
96        @keyframes fadeEffect {
97          from {opacity: 0;}
98          to {opacity: 1;}
99        }
100     </style>
101   </head>
102
103   <div class="wrapper">
104
105     <body>
106       <h1> Web-Konfiguration MessiBox WiFi Knoten  </h1>
107       <p id = "id_p01"> Die Einstellparameter konnen hier
             verandert werden </p>
108       <br />
109       <br />
110
111     <div class="tab">
112       <button class="tablinks" onclick="open_tab(event, '
             Ubersicht')"id="defaultOpen">Ubersicht</button>
113       <button class="tablinks" onclick="open_tab(event, '
             Netzwerk')">Netzwerk</b></button>
```

```
114       <button class="tablinks" onclick="open_tab(event, '
              Sensoren')">Sensoren</button>
115       <button class="tablinks" onclick="open_tab(event, '
              Tabelle')">Tabelle</button>
116     </div>
117
118
119     <div id = "Ubersicht" class="tabcontent">
120       <h3> System information </h3>
121       <br />
122       <form action="PAGE_restart.html" method = "GET">
123       <fieldset>
124         <legend> Gerat: </legend>
125           <center>
126           <table>
127             <tr>
128               <th scope = "row", align = "right"> Name: </th>
129               <td id="_name"> </td>
130             </tr>
131             <tr>
132               <th scope = "row", align = "right"> GSM ID: </th>
133               <td id="_gsm_id"> </td>
134             </tr>
135             <tr>
136               <th scope = "row", align = "right"> Firmware
                      Version: </th>
137               <td id="_firmware_version"> </td>
138             </tr>
139             <tr>
140               <th scope = "row", align = "right"> Verbunden
                      mit: </th>
141               <td id="_connected_to"> </td>
142             </tr>
143             <tr>
144               <th scope = "row", align = "right"> WLAN Starke:
                      </th>
145               <td id="_wifi_strength"> </td>
146             </tr>
147           </table>
148           <br />
149           </center>
150       </fieldset>
151       <br />
152       <fieldset>
```

```
153         <legend> Netzwerkeinstellungen: </legend>
154           <center>
155           <table>
156             <tr>
157               <th scope = "row", align = "right"> WiFi ssid 1:
                     </th>
158                 <td id = "_ssid_0"> </td>
159             </tr>
160             <tr>
161               <th scope = "row", align = "right"> WiFi Password
                      1: </th>
162                 <td id = "_password_0"> </td>
163             </tr>
164             <tr>
165               <th scope = "row", align = "right"> WiFi ssid 2:
                     </th>
166                 <td id = "_ssid_1"> </td>
167             </tr>
168             <tr>
169               <th scope = "row", align = "right"> WiFi Password
                      2: </th>
170                 <td id = "_password_1"> </td>
171             </tr>
172             <tr>
173               <th scope = "row", align = "right"> Host: </th>
174                 <td id = "_api_url"> </td>
175             </tr>
176             <tr>
177               <th scope = "row", align = "right"> API key: </th
                    >
178                 <td id = "_api_key"> </td>
179             </tr>
180           </table>
181           </center>
182           <br />
183       </fieldset>
184       <br />
185       <fieldset>
186         <legend> Sensoreinstellungen: </legend>
187           <center>
188           <table>
189             <tr>
190               <th scope = "row", align = "right"> Anzahl
                     Temperatursensoren: </th>
```

```
191                 <td id = "_number_of_temp_sensor"></td>
192               </tr>
193               <tr>
194                 <th scope = "row", align = "right">
                      Feuchtigkeitsensor: </th>
195                 <td id = "_hum"></td>
196               </tr>
197               <tr>
198                 <th scope = "row", align = "right"> S0
                      Impulszahler: </th>
199                 <td id = "_s0"></td>
200               </tr>
201               <tr>
202                 <th scope = "row", align = "right"> Gaszahler ID:
                       </th>
203                 <td id = "_s0_id"></td>
204               </tr>
205               <tr>
206                 <th scope = "row", align = "right"> Gaszahler: </
                      th>
207                 <td id = "_s0_counter"></td>
208               </tr>
209               <tr>
210                 <th scope = "row", align = "right"> Einheit/
                      Impulse: </th>
211                 <td id = "_unit_per_pulse"></td>
212               </tr>
213             </table>
214             </center>
215             <br />
216         </fieldset>
217         <br />
218         <br />
219         <br />
220         <center>
221         <label> Alle anderungen ubernehmen </label>
222         <button id = "save_all" type = "submit" style="
              width:150px"> Neustart </button>
223         </form>
224         </center>
225       </div>
226
227
228       <div id="Netzwerk" class="tabcontent">
```

```
229         <h3>Netzwerk</h3>
230         <fieldset>
231           <legend> Einstellungen: </legend>
232             <center>
233             <form action = "PAGE_saveNetworkConf.html" method = "
                    GET">
234             <table>
235               <tr>
236                 <th scope = "row", align = "right"> Name: </th>
237                 <td>
238                   <input  type = "text"
239                       id = "name"
240                       name = "name"
241                       size = "20"
242                       maxlength = "50"
243                       placeholder = "Apparat" ></input>
244                 </td>
245               </tr>
246               <tr>
247                 <th scope = "row", align = "right"> GSM ID: </th>
248                 <td>
249                   <input  type = "text"
250                       id = "gsm_id"
251                       name = "gsm_id"
252                       size = "20"
253                       maxlength = "50"
254                       placeholder = "400066" ></input>
255                 </td>
256               </tr>
257               <tr>
258                 <th scope = "row", align = "right"> WiFi ssid: </
                      th>
259                 <td>
260                   <input  type = "text"
261                       id = "ssid"
262                       name = "ssid"
263                       size = "20"
264                       maxlength = "50"
265                       placeholder = "stecnet"></input>
266                 </td>
267               </tr>
268               <tr>
269                 <th scope = "row", align = "right"> WiFi
                      Password: </th>
```

```
270                 <td>
271                   <input  type = "text"
272                       id = "password"
273                       name = "password"
274                       value = ""
275                       size = "20"
276                       maxlength = "50"
277                       placeholder = "************"></input>
278                 </td>
279               </tr>
280               <tr>
281                 <th scope = "row", align = "right"> Host: </th>
282                 <td>
283                   <input  type = "text"
284                       id = "host"
285                       name = "host"
286                       value = ""
287                       size = "20"
288                       maxlength = "50"
289                       placeholder = "beta.ez-nord.com" ></input>
290                 </td>
291               </tr>
292               <tr>
293                 <th scope = "row", align = "right"> API Key: </th
                      >
294                 <td>
295                   <input  type = "text"
296                       id = "api_key"
297                       name = "api_key"
298                       value = ""
299                       size = "20"
300                       maxlength = "50"
301                       placeholder = "" ></input>
302                 </td>
303               </tr>
304             </table>
305             <br />
306             <br />
307           <center>
308           <label> Einstellungen speichern  </label>
309           <button id = "save_networkConf" type = "submit" style="
                  width:150px"> Speichern </button>
310           <br />
311           </form>
```

```
312         </center>
313       </fieldset>
314     </div>
315
316     <div id="Sensoren" class="tabcontent">
317       <h3>Sensoren</h3>
318       <fieldset>
319         <legend> Einstellungen: </legend>
320           <center>
321           <form action = "PAGE_saveSensorConf.html" method = "
                GET">
322           <table>
323             <tr>
324               <th scope = "row", align = "right"> Feuchtigkeit:
                    </th>
325               <td>
326                 <input  type = "checkbox"
327                     id = "humidity"
328                     name = "humidity"
329                     value = "hum"></input>
330               </td>
331             </tr>
332             <tr>
333               <th scope = "row", align = "right"> S0: </th>
334               <td>
335                 <input  type = "checkbox"
336                     id = "s0"
337                     name = "s0"
338                     value = "s0" ></input>
339               </td>
340             </tr>
341             <tr>
342               <th scope = "row", align = "right"> Gaszahler ID:
                    </th>
343               <td>
344                 <input  type = "number"
345                     id = "gaszaehler_id"
346                     name = "gaszaehler_id"
347                     value = ""
348                     size = "20"
349                     maxlength = "50"
350                     placeholder = "1245878"></input>
351               </td>
352             </tr>
```

```
353             <tr>
354               <th scope = "row", align = "right"> Gaszahler: </
                     th>
355               <td>
356                 <input  type = "number"
357                     id = "gaszaehler"
358                     name = "gaszaehler"
359                     value = ""
360                     step = "0.01"
361                     size = "20"
362                     maxlength = "50"
363                     placeholder = "1245873.8"></input>
364               </td>
365             </tr>
366             <tr>
367               <th scope = "row", align = "right"> Einheit/
                     Impulse: </th>
368               <td>
369                 <input  type = "number"
370                     id = "einheit_impulse"
371                     name = "einheit_impulse"
372                     value = ""
373                     step = "0.01"
374                     size = "20"
375                     maxlength = "50"
376                     placeholder = "0.1"></input>
377               </td>
378             </tr>
379           </table>
380           <br />
381           <br />
382         <center>
383         <label> Einstellungen speichern  </label>
384         <button id = "save_sensorConf" type = "submit" style="
              width:150px"> Speichern </button>
385         <br />
386         </form>
387         </center>
388       </fieldset>
389       <br />
390       <fieldset>
391         <legend> Bezeichnung: </legend>
392           <center>
```

```
393             <form action = "PAGE_saveDescription.html" method = "
                    GET">
394             <table>
395               <tr>
396               <th scope = "row", align = "right">Sensor  1: </th>
397                 <td>
398                   <input  type = "text"
399                       id = "1_id"
400                       name = "1_id"
401                       value = ""
402                       size = "20"
403                       maxlength = "50"
404                       placeholder = ""></input>
405                 </td>
406               </tr>
407               <tr>
408               <th scope = "row", align = "right">Sensor  2: </th>
409                 <td>
410                   <input  type = "text"
411                       id = "2_id"
412                       name = "2_id"
413                       value = ""
414                       size = "20"
415                       maxlength = "50"
416                       placeholder = ""></input>
417                 </td>
418               </tr>
419               <tr>
420               <th scope = "row", align = "right">Sensor  3: </th>
421                 <td>
422                   <input  type = "text"
423                       id = "3_id"
424                       name = "3_id"
425                       value = ""
426                       size = "20"
427                       maxlength = "50"
428                       placeholder = ""></input>
429                 </td>
430               </tr>
431               <tr>
432               <th scope = "row", align = "right">Sensor  4: </th>
433                 <td>
434                   <input  type = "text"
435                       id = "4_id"
```

```
436                     name = "4_id"
437                     value = ""
438                     size = "20"
439                     maxlength = "50"
440                     placeholder = ""></input>
441               </td>
442             </tr>
443             <tr>
444             <th scope = "row", align = "right">Sensor  5: </th>
445               <td>
446                 <input  type = "text"
447                     id = "5_id"
448                     name = "5_id"
449                     value = ""
450                     size = "20"
451                     maxlength = "50"
452                     placeholder = ""></input>
453               </td>
454             </tr>
455             <tr>
456             <th scope = "row", align = "right">Sensor  6: </th>
457               <td>
458                 <input  type = "text"
459                     id = " 6_id"
460                     name = "6_id"
461                     value = ""
462                     size = "20"
463                     maxlength = "50"
464                     placeholder = ""></input>
465               </td>
466             </tr>
467             <tr>
468             <th scope = "row", align = "right">Sensor  7: </th>
469               <td>
470                 <input  type = "text"
471                     id = "7_id"
472                     name = "7_id"
473                     value = ""
474                     size = "20"
475                     maxlength = "50"
476                     placeholder = ""></input>
477               </td>
478             </tr>
479             <tr>
```

```
480             <th scope = "row", align = "right">Sensor  8: </th>
481               <td>
482                 <input  type = "text"
483                     id = "8_id"
484                     name = "8_id"
485                     value = ""
486                     size = "20"
487                     maxlength = "50"
488                     placeholder = ""></input>
489               </td>
490             </tr>
491             <tr>
492             <th scope = "row", align = "right">Sensor  9: </th>
493               <td>
494                 <input  type = "text"
495                     id = "9_id"
496                     name = "9_id"
497                     value = ""
498                     size = "20"
499                     maxlength = "50"
500                     placeholder = ""></input>
501               </td>
502             </tr>
503             <tr>
504             <th scope = "row", align = "right">Sensor 10: </th>
505               <td>
506                 <input  type = "text"
507                     id = "10_id"
508                     name = "10_id"
509                     value = ""
510                     size = "20"
511                     maxlength = "50"
512                     placeholder = ""></input>
513               </td>
514             </tr>
515             <tr>
516             <th scope = "row", align = "right">Sensor 11: </th>
517               <td>
518                 <input  type = "text"
519                     id = "11_id"
520                     name = "11_id"
521                     value = ""
522                     size = "20"
523                     maxlength = "50"
```

```
524                     placeholder = ""></input>
525               </td>
526             </tr>
527             <tr>
528             <th scope = "row", align = "right">Sensor 12: </th>
529               <td>
530                 <input  type = "text"
531                     id = "12_id"
532                     name = "12_id"
533                     value = ""
534                     size = "20"
535                     maxlength = "50"
536                     placeholder = ""></input>
537               </td>
538             </tr>
539             <tr>
540             <th scope = "row", align = "right">Sensor 13: </th>
541               <td>
542                 <input  type = "text"
543                     id = "13_id"
544                     name = "13_id"
545                     value = ""
546                     size = "20"
547                     maxlength = "50"
548                     placeholder = ""></input>
549               </td>
550             </tr>
551             <tr>
552             <th scope = "row", align = "right">Sensor 14: </th>
553               <td>
554                 <input  type = "text"
555                     id = "14_id"
556                     name = "14_id"
557                     value = ""
558                     size = "20"
559                     maxlength = "50"
560                     placeholder = ""></input>
561               </td>
562             </tr>
563             <tr>
564             <th scope = "row", align = "right">Sensor 15: </th>
565               <td>
566                 <input  type = "text"
567                     id = "15_id"
```

```
568                     name = "15_id"
569                     value = ""
570                     size = "20"
571                     maxlength = "50"
572                     placeholder = ""></input>
573               </td>
574             </tr>
575             <tr>
576             <th scope = "row", align = "right">Sensor 16: </th>
577               <td>
578                 <input  type = "text"
579                     id = "16_id"
580                     name = "16_id"
581                     value = ""
582                     size = "20"
583                     maxlength = "50"
584                     placeholder = ""></input>
585               </td>
586             </tr>
587             <tr>
588             <th scope = "row", align = "right">Sensor 17: </th>
589               <td>
590                 <input  type = "text"
591                     id = "17_id"
592                     name = "17_id"
593                     value = ""
594                     size = "20"
595                     maxlength = "50"
596                     placeholder = ""></input>
597               </td>
598             </tr>
599             <tr>
600             <th scope = "row", align = "right">Sensor 18: </th>
601               <td>
602                 <input  type = "text"
603                     id = "18_id"
604                     name = "18_id"
605                     value = ""
606                     size = "20"
607                     maxlength = "50"
608                     placeholder = ""></input>
609               </td>
610             </tr>
611             <tr>
```

```
612             <th scope = "row", align = "right">Sensor 19: </th>
613               <td>
614                 <input  type = "text"
615                     id = "19_id"
616                     name = "19_id"
617                     value = ""
618                     size = "20"
619                     maxlength = "50"
620                     placeholder = ""></input>
621               </td>
622             </tr>
623             <tr>
624             <th scope = "row", align = "right">Sensor 20: </th>
625               <td>
626                 <input  type = "text"
627                     id = "20_id"
628                     name = "20_id"
629                     value = ""
630                     size = "20"
631                     maxlength = "50"
632                     placeholder = ""></input>
633               </td>
634             </tr>
635           </table>
636           <br />
637         <br />
638         <center>
639         <label> Bezeichnung speichern  </label>
640         <button id = "save_descriptionConf" type = "submit"
              style="width:150px"> Speichern </button>
641         <br />
642         </form>
643         </center>
644       </fieldset>
645     </div>
646
647     <div id="Tabelle" class="tabcontent">
648     <h3>Temperaturtabelle</h3>
649       <table id = "sensortabelle" class="sensor">
650
651       </table>
652       <br />
653       <br />
654     </div>
```

```
655   </div>
656
657 <script>
658
659
660   function csv_to_table(csv_source, id){
661     $.ajax({
662     url: csv_source,
663     dataType: 'text',
664     }).done(successFunction);
665
666     function successFunction(data) {
667       var allRows = data.split(/\r?\n|\r/);
668       var table = '<table>';
669
670       for (var singleRow = 0; singleRow < allRows.length;
             singleRow++) {
671
672       if (singleRow === 0) {
673         table += '<thead>';
674         table += '<tr>';
675       } else {
676         table += '<tr>';
677       }
678       var rowCells = allRows[singleRow].split(';');
679       for (var rowCell = 0; rowCell < rowCells.length; rowCell
            ++) {
680         if (singleRow === 0) {
681         table += '<th>';
682         table += rowCells[rowCell];
683         table += '</th>';
684         } else {
685         table += '<td>';
686         table += rowCells[rowCell];
687         table += '</td>';
688         }
689       }
690       if (singleRow === 0) {
691         table += '</tr>';
692         table += '</thead>';
693         table += '<tbody>';
694       } else {
695         table += '</tr>';
696       }
```

```
697         }
698         table += '</tbody>';
699         table += '</table>';
700        $(id).append(table);
701       }
702     }
703     document.getElementById("sensortabelle").innerHTML =
          csv_to_table("data.csv", "#sensortabelle");
704
705     var xmlhttp = new XMLHttpRequest();
706     xmlhttp.onreadystatechange = function() {
707       if (this.readyState == 4 && this.status == 200) {
708         var myObj = JSON.parse(this.responseText);
709
710         document.getElementById("_name").innerHTML = myObj.name;
711         document.getElementById("_gsm_id").innerHTML =
              myObj.gsm_id;
712         document.getElementById("_firmware_version").innerHTML =
              myObj.firmware_version;
713         document.getElementById("_connected_to").innerHTML =
              myObj.connected_to;
714         document.getElementById("_wifi_strength").innerHTML =
              myObj.wifi_strength;
715
716         document.getElementById("_ssid_0").innerHTML =
              myObj.ssid_0;
717         document.getElementById("_password_0").innerHTML =
              myObj.password_0;
718         document.getElementById("_ssid_1").innerHTML =
              myObj.ssid_1;
719         document.getElementById("_password_1").innerHTML =
              myObj.password_1;
720
721         document.getElementById("_api_url").innerHTML =
              myObj.api_url;
722         document.getElementById("_api_key").innerHTML =
              myObj.api_key;
723
724         document.getElementById("_number_of_temp_sensor").
              innerHTML = myObj.number_of_temp_sensor;
725         document.getElementById("_hum").innerHTML = myObj.hum;
726         document.getElementById("_s0").innerHTML = myObj.s0;
727         document.getElementById("_s0_id").innerHTML =
              myObj.s0_id;
```

```
728         document.getElementById("_s0_counter").innerHTML =
                myObj.s0_counter;
729         document.getElementById("_unit_per_pulse").innerHTML =
                myObj.unit_per_pulse;
730
731       }
732     };
733     xmlhttp.open("GET", "my_config_file.json", true);
734     xmlhttp.send();
735
736
737
738     function open_tab(evt, tab_name) {
739       var i, tabcontent, tablinks;
740       tabcontent = document.getElementsByClassName("tabcontent");
741       for (i = 0; i < tabcontent.length; i++) {
742         tabcontent[i].style.display = "none";
743       }
744       tablinks = document.getElementsByClassName("tablinks");
745       for (i = 0; i < tablinks.length; i++) {
746         tablinks[i].className = tablinks[i].className.replace("
                active", "");
747       }
748       document.getElementById(tab_name).style.display = "block";
749       evt.currentTarget.className += " active";
750     }
751
752 // Get the element with id="defaultOpen" and click on it
753     document.getElementById("defaultOpen").click();
754
755 </script>
756 </body>
757 </html>
```

PAGE_restart.html

```
1
2 <!DOCTYPE html>
3 <html>
4   <head>
5     <meta charset="UTF-8">
6       <title> ESP Configuration </title>
7       <figure>
8         <img src = "Energiezentrale_logo.png" alt = "logo" align
              ="right">
9       </figure>
10      <style type = "text/css">
11        body {
12
13          font-family: "Lato", sans-serif;
14          margin-left:20px;
15        }
16      </style>
17    </head>
18    <body>
19    <div align="center">
20      <br />
21      <br />
22      <br />
23      <br />
24      <h1> Alle Einstellungen wurden ubernommen  </h2>
25      <br />
26      <br />
27        <br />
28        <form action = "PAGE_main.html" method = "GET">
29          <label style = "margin-left: 30px;"> Zuruck zum
                Ubersicht </label>
30          <input type = "submit" style = "width: 50px" value = "
                OK">
31        <br />
32        </form>
33        <br />
34    </div>
35    </body>
36 </html>
```

PAGE_saveDescription.html

```
1
2  <!DOCTYPE html>
3  <html>
4    <head>
5      <meta charset="UTF-8">
6      <title> ESP Configuration </title>
7      <figure>
8        <img src = "Energiezentrale_logo.png" alt = "logo" align
             ="right">
9      </figure>
10     <style type = "text/css">
11       body {
12
13         font-family: "Lato", sans-serif;
14         margin-left:20px;
15       }
16     </style>
17   </head>
18   <body>
19   <div align="center">
20     <br />
21     <br />
22     <br />
23     <br />
24     <h1> Bezeichnungen wurden gespeichert </h2>
25     <br />
26     <br />
27
28       <br />
29       <form action = "PAGE_main.html" method = "GET">
30         <label style = "margin-left: 30px;"> Zuruck zum
               Ubersicht </label>
31         <input type = "submit" style = "width: 50px" value = "
               OK">
32       <br />
33       </form>
34       <br />
35   </div>
36   </body>
37 </html>
```

PAGE_saveNetworkConf.html

```
1
2  <!DOCTYPE html>
3  <html>
4    <head>
5      <meta charset="UTF-8">
6      <title> ESP Configuration </title>
7      <figure>
8        <img src = "Energiezentrale_logo.png" alt = "logo" align
             ="right">
9      </figure>
10     <style type = "text/css">
11       body {
12         background-color: #e6e6e6;
13         font-family: "Lato", sans-serif;
14         margin-left:20px;
15       }
16     </style>
17   </head>
18   <body>
19   <div align="center">
20     <br />
21     <br />
22     <br />
23     <br />
24     <h1> Netzwerkeinstellungen wurden gespeichert </h2>
25     <br />
26     <br />
27
28       <br />
29       <form action = "PAGE_main.html" method = "GET">
30         <label style = "margin-left: 30px;"> Zuruck zum
               Ubersicht </label>
31         <input type = "submit" style = "width: 50px" value = "
               OK">
32       <br />
33       </form>
34       <br />
35   </div>
36   </body>
37 </html>
```

PAGE_saveSensorConf.html

```
1
2  <!DOCTYPE html>
3  <html>
4    <head>
5      <meta charset="UTF-8">
6      <title> ESP Configuration </title>
7      <figure>
8        <img src = "Energiezentrale_logo.png" alt = "logo" align
             ="right">
9      </figure>
10     <style type = "text/css">
11       body {
12
13         font-family: "Lato", sans-serif;
14         margin-left:20px;
15       }
16     </style>
17   </head>
18   <body>
19   <div align="center">
20     <br />
21     <br />
22     <br />
23     <br />
24     <h1> Einstellungen wurden gespeichert </h2>
25     <br />
26     <br />
27
28       <br />
29       <form action = "PAGE_main.html" method = "GET">
30         <label style = "margin-left: 30px;"> Zuruck zum
               Ubersicht </label>
31         <input type = "submit" style = "width: 50px" value = "
               OK">
32       <br />
33       </form>
34       <br />
35   </div>
36   </body>
37 </html>
```

success.html

```
1
2  <!DOCTYPE html>
3  <html>
4    <head>
5      <meta charset="UTF-8">
6      <title> ESP Configuration </title>
7      <figure>
8        <img src = "Energiezentrale_logo.png" alt = "logo" align
             ="right">
9      </figure>
10     <style type = "text/css">
11       body {
12
13         font-family: "Lato", sans-serif;
14         margin-left:20px;
15       }
16     </style>
17   </head>
18   <body>
19   <div align="center">
20     <br />
21     <br />
22     <br />
23     <br />
24     <h1> Datei hochladen war erfolgreich </h2>
25     <br />
26     <br />
27
28       <br />
29       <form action = "PAGE_main.html" method = "GET">
30         <label style = "margin-left: 30px;"> Zuruck zum
               Ubersicht </label>
31         <input type = "submit" style = "width: 50px" value = "
               OK">
32       <br />
33       </form>
34       <br />
35   </div>
36   </body>
37 </html>
```

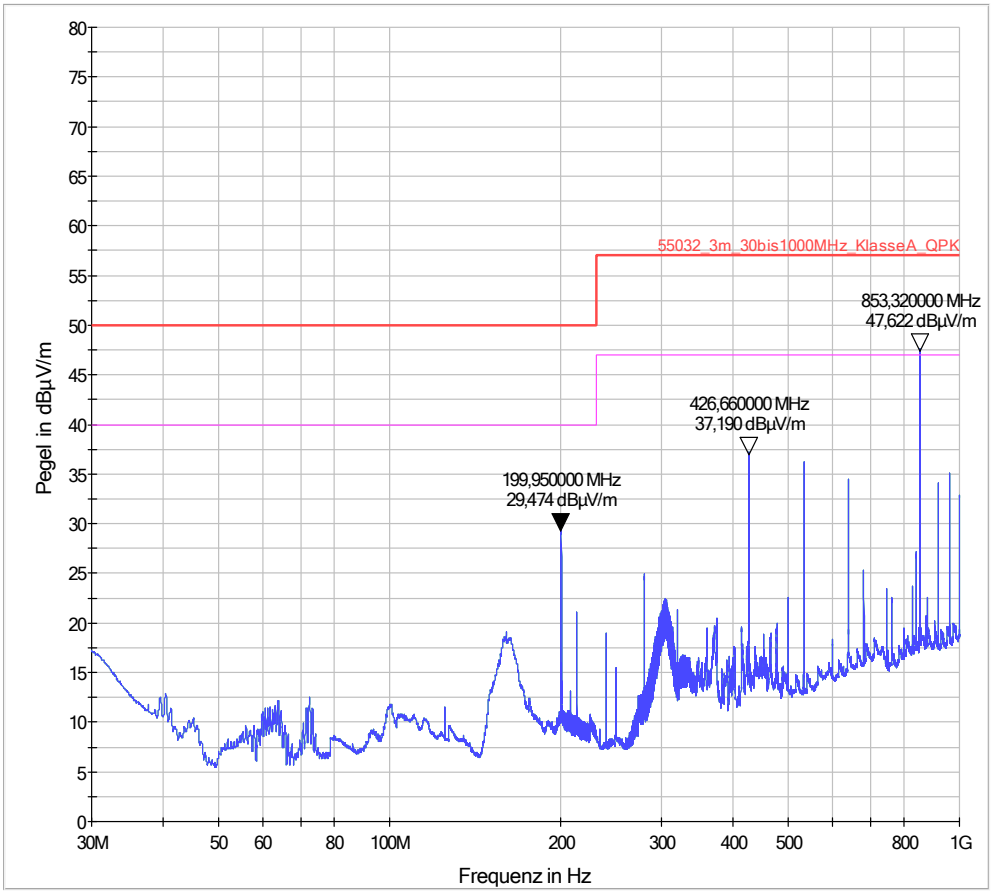
upload.html

```
1 <form method="post" enctype="multipart/form-data">
2     <input type="file" name="name">
3     <input class="button" type="submit" value="Upload">
4 </form>
```

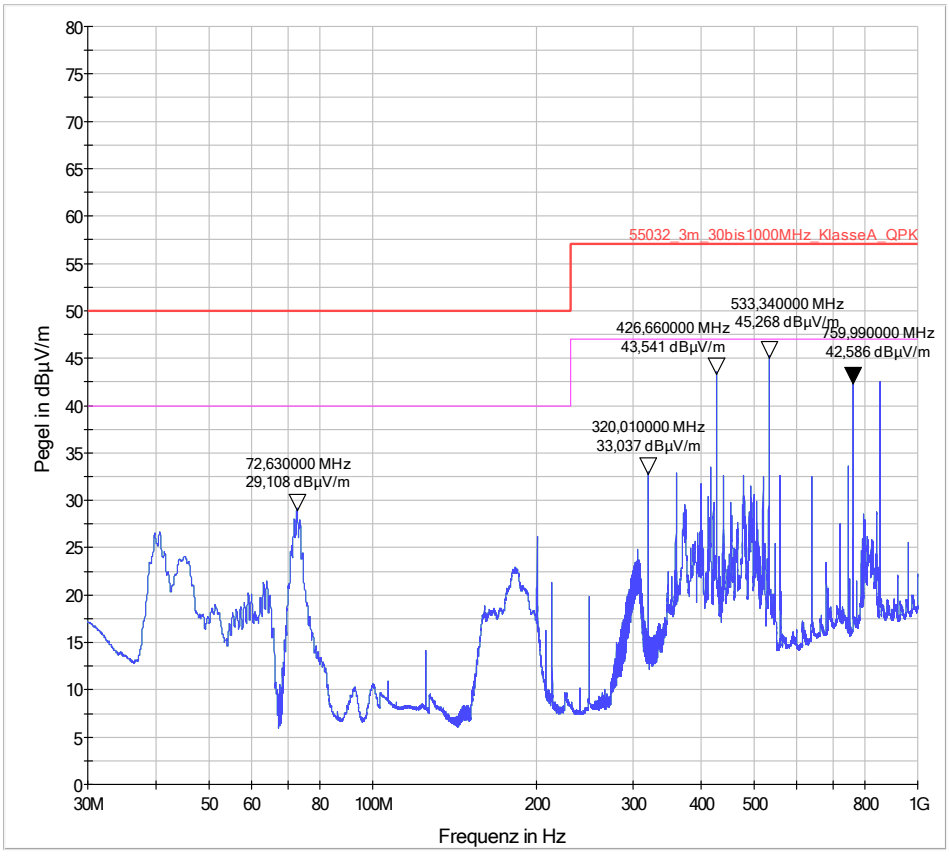
D. EMV Messergebnisse

Prüfobjekt Steuerplatine

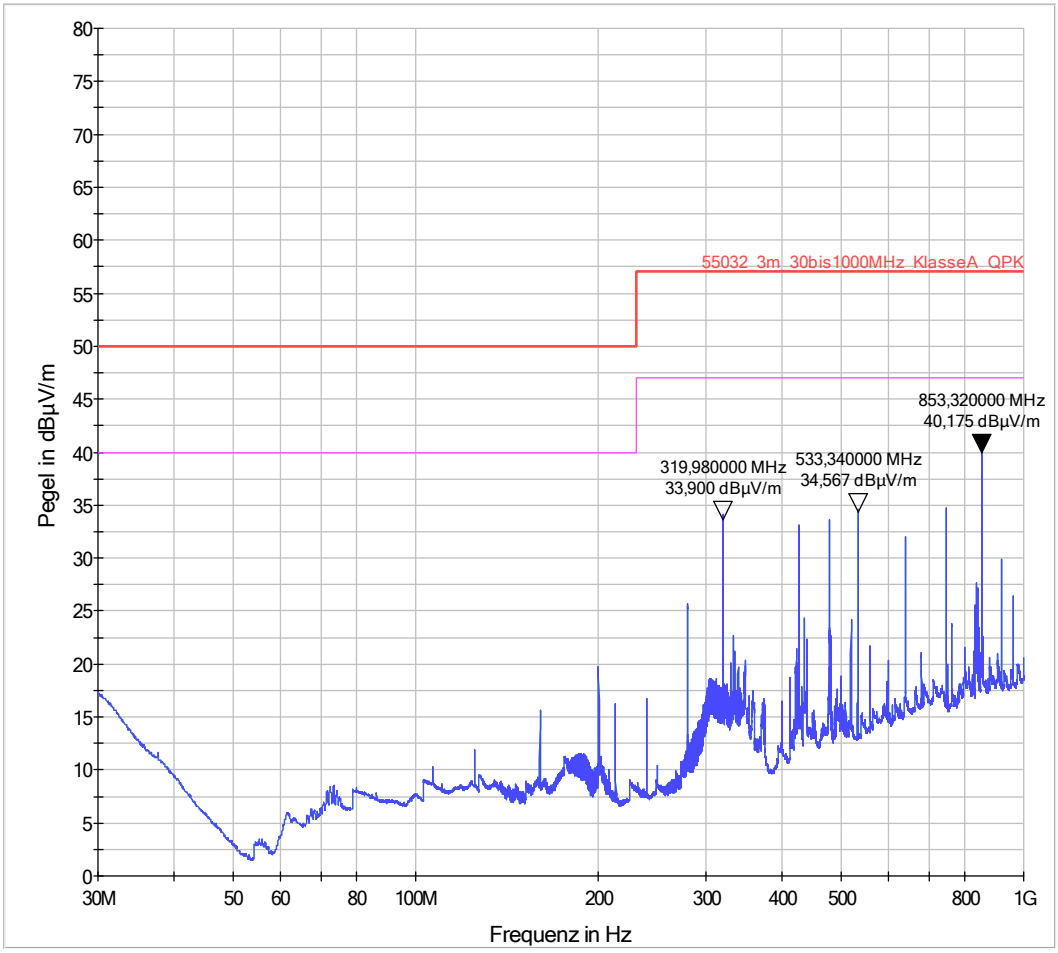
Horizontal / 0°



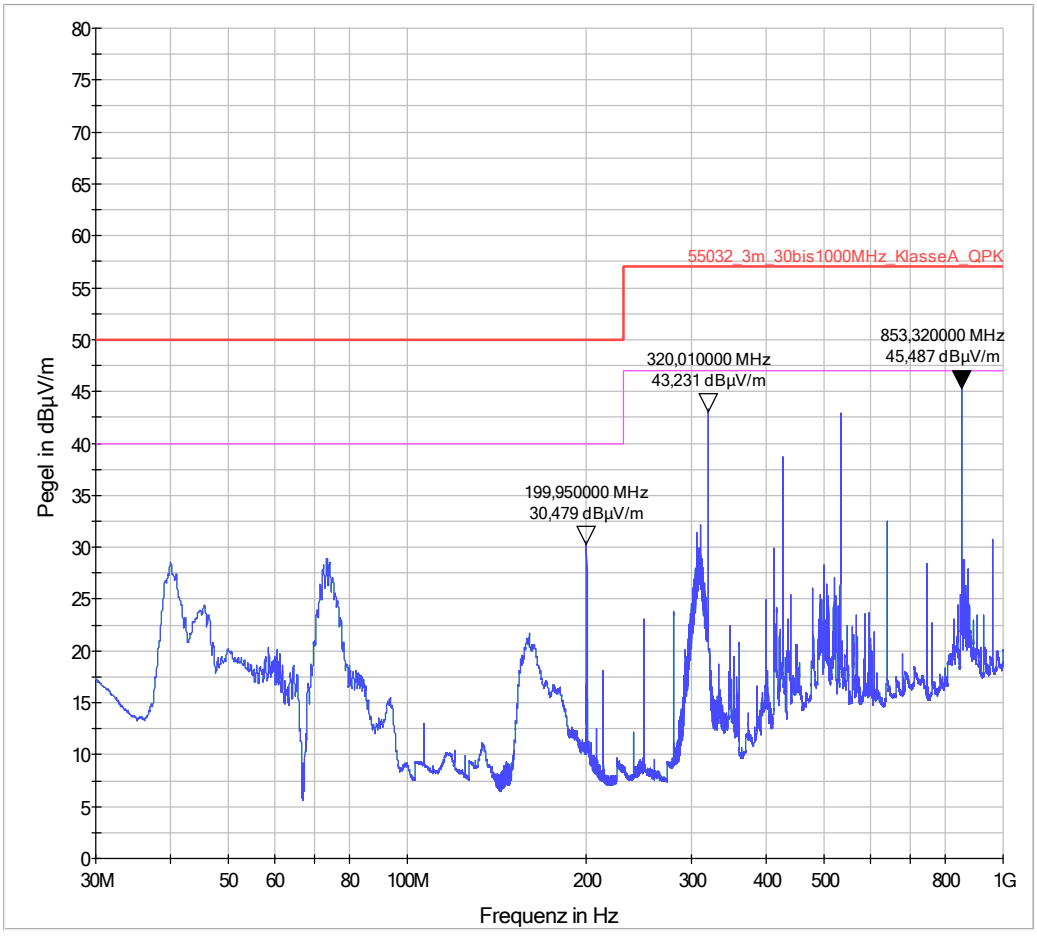
Vertikal / 0°



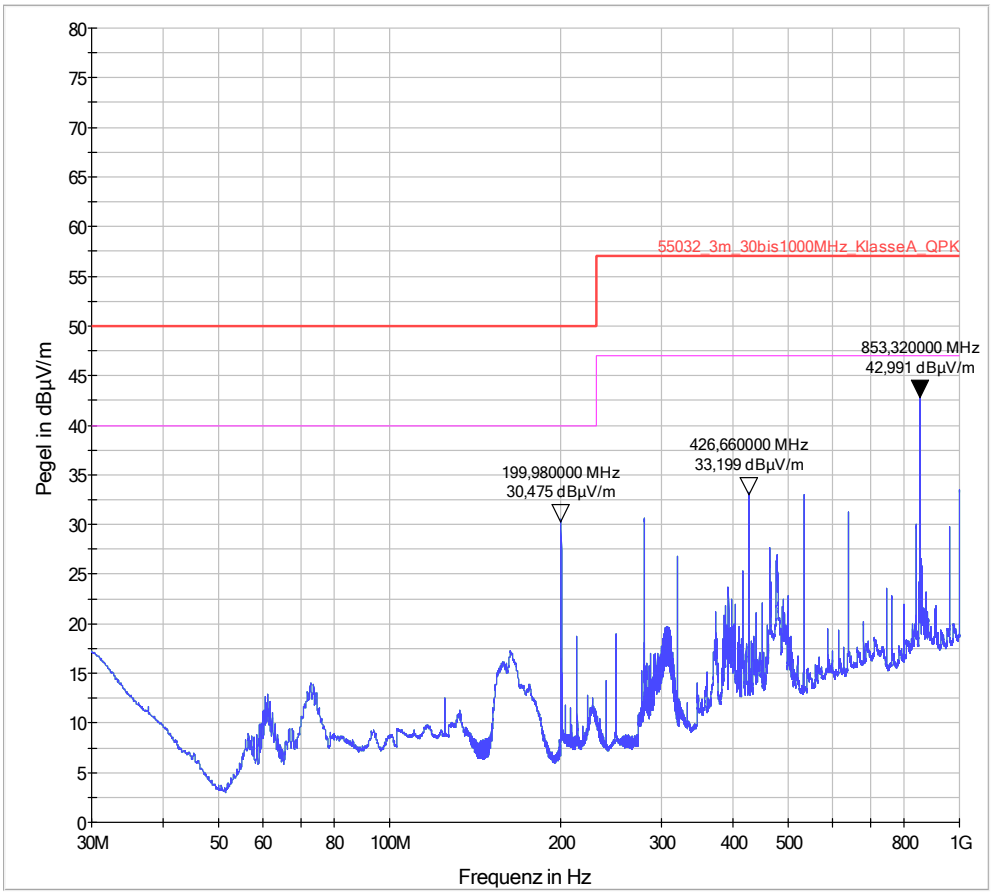
Horizontal / 90°



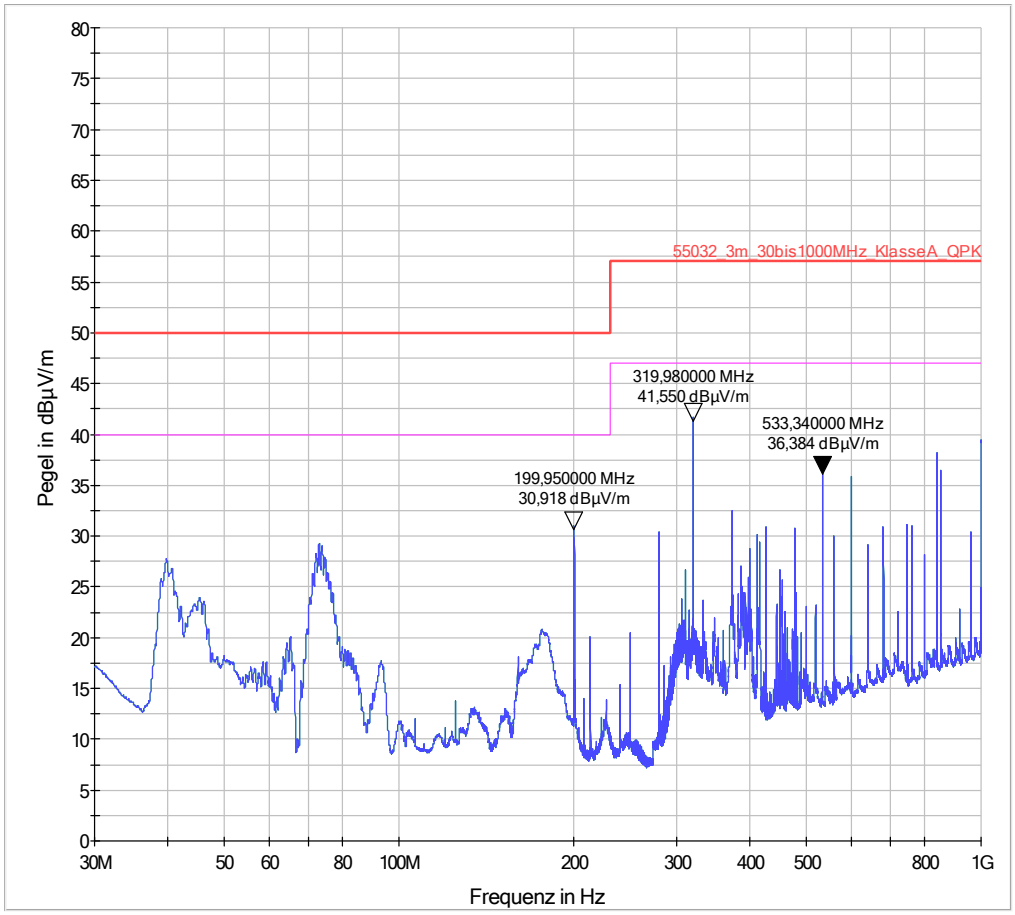
Vertikal / 90°



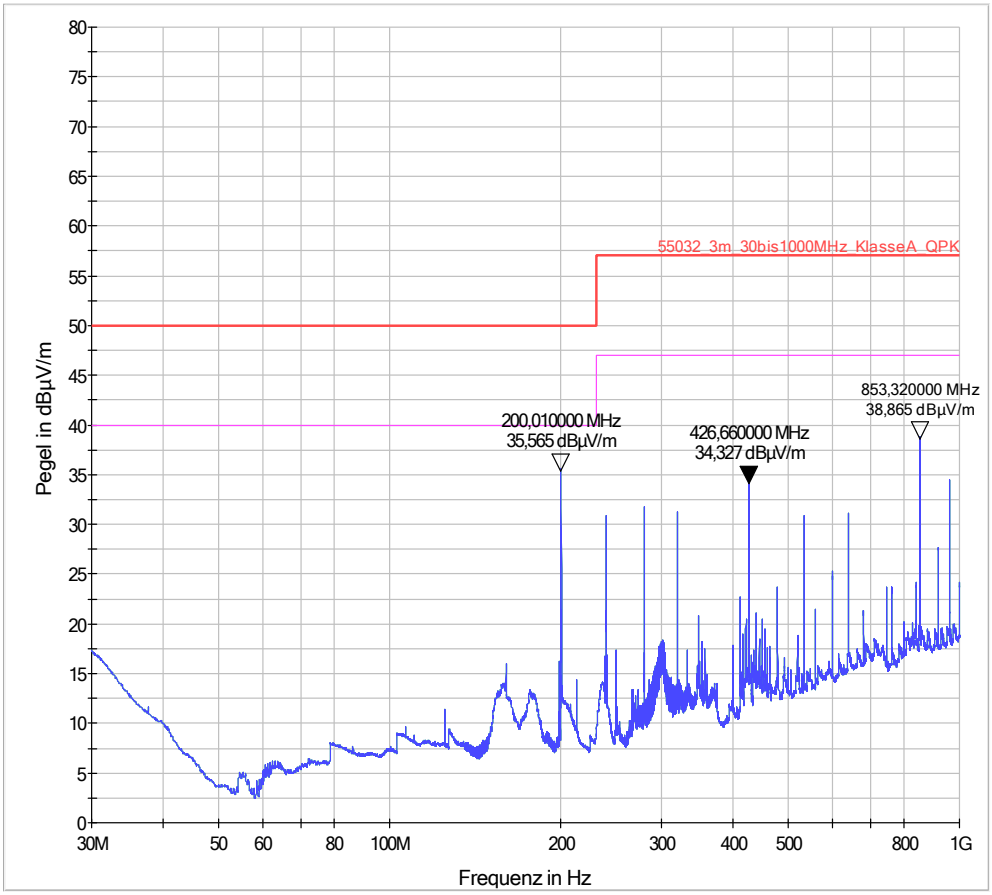
Horizontal / 180°



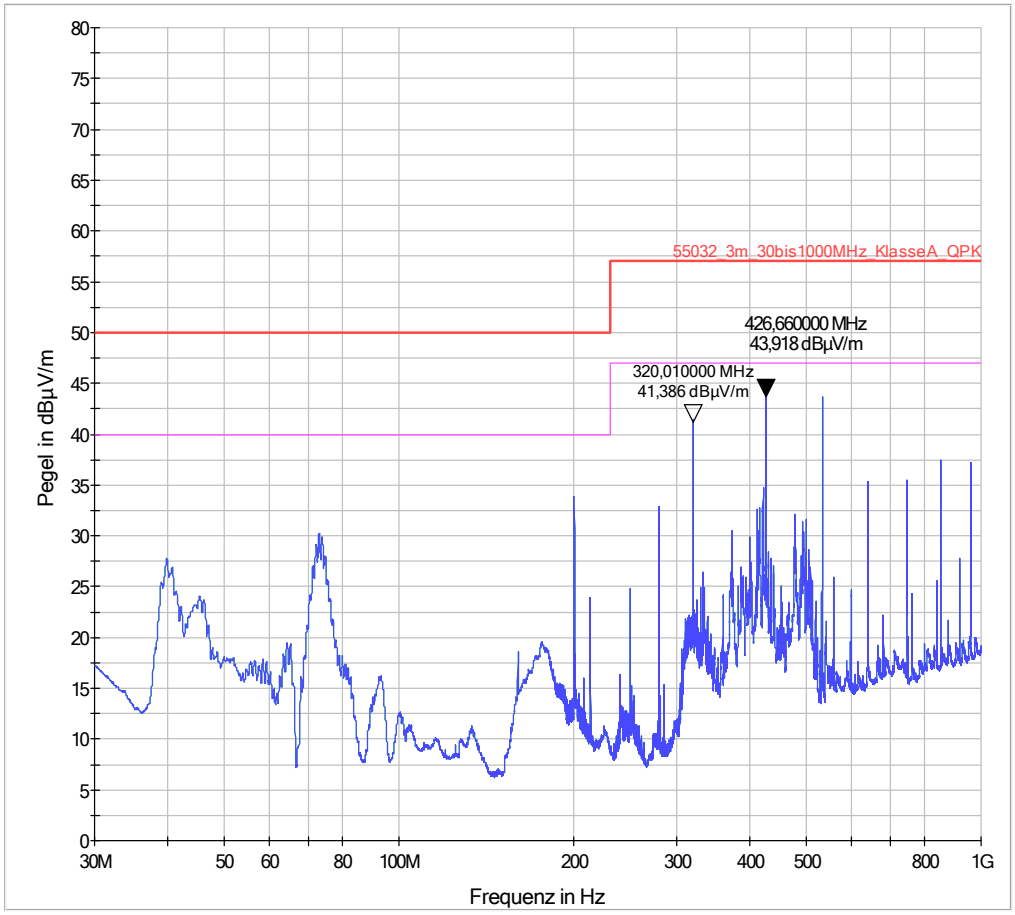
Vertikal / 180°



Horizontal / 270°

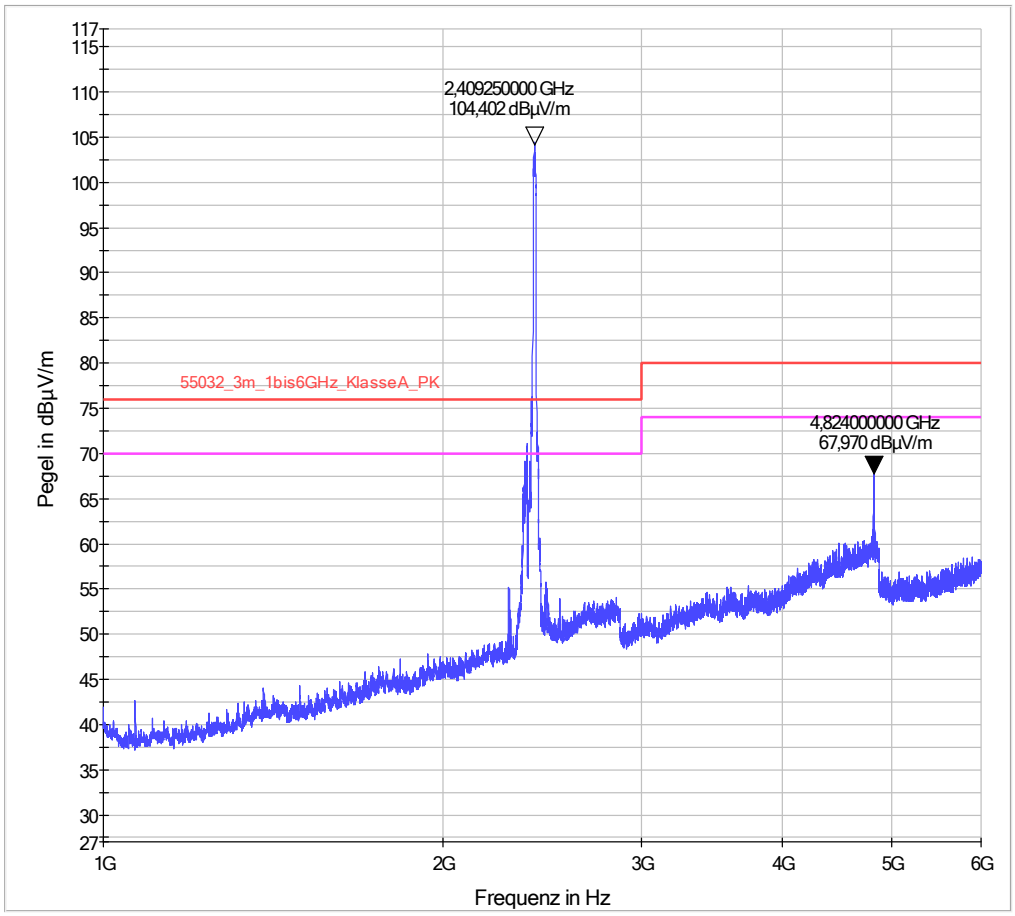


Vertikal / 270°

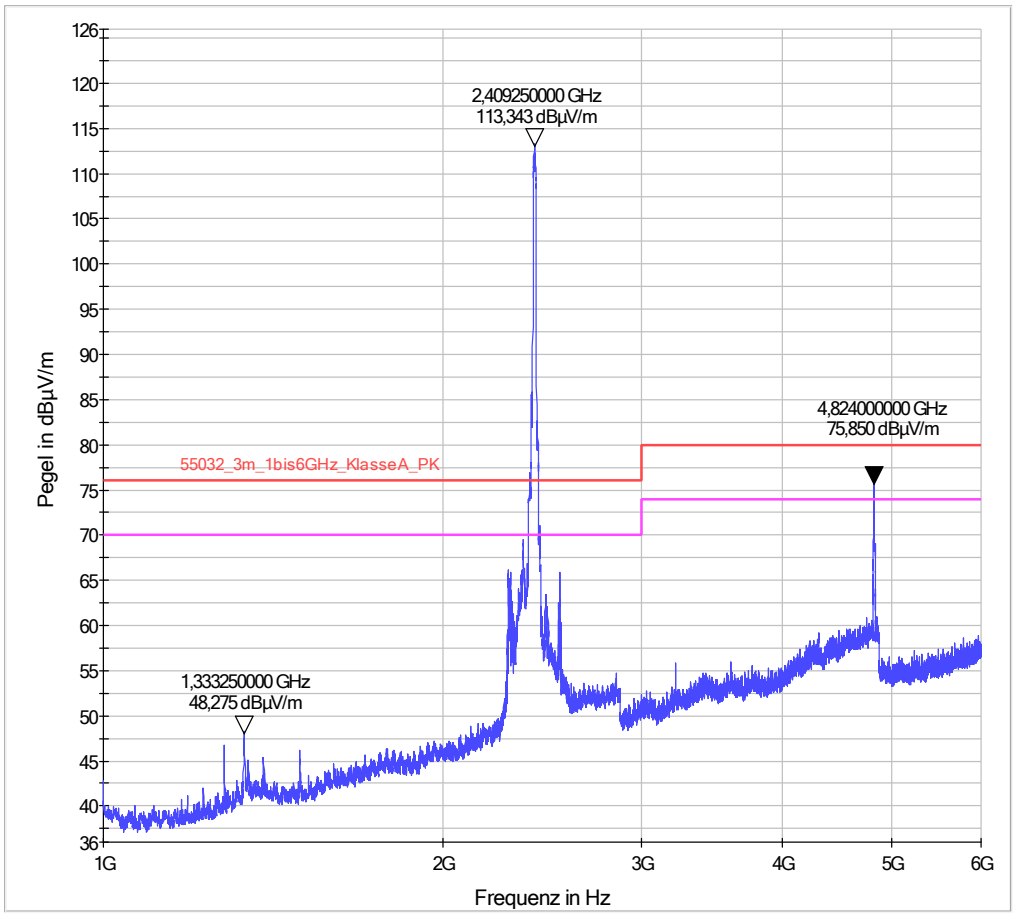


Prüfobjekt Steuerplatine

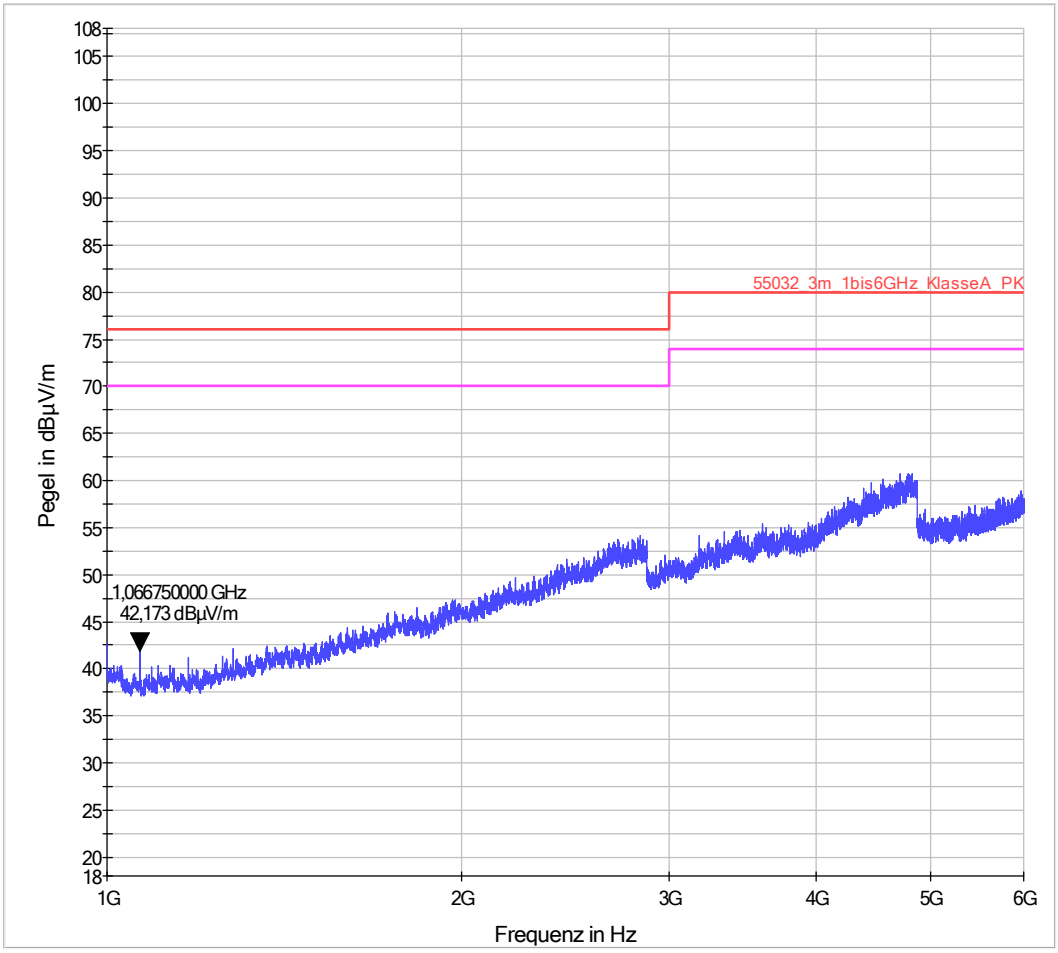
Horizontal / 0°



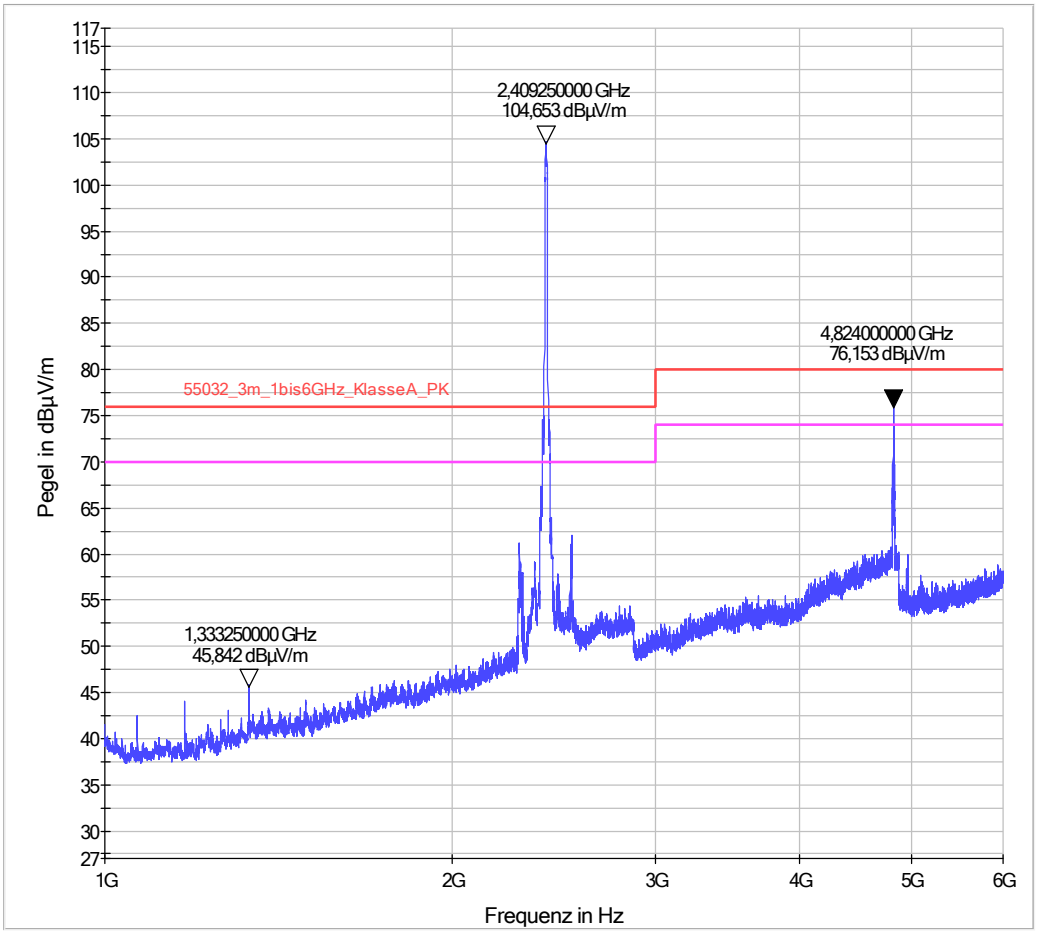
Vertikal / 0°



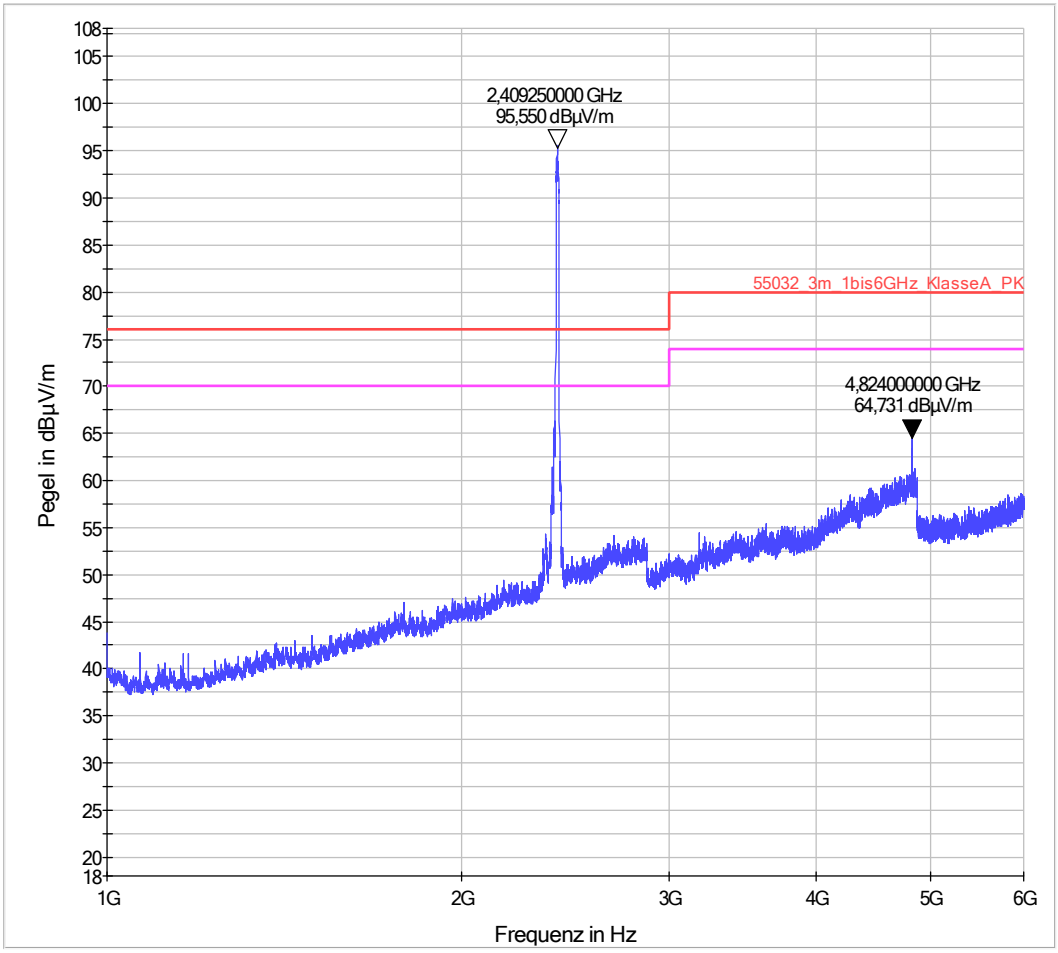
Horizontal / 90°



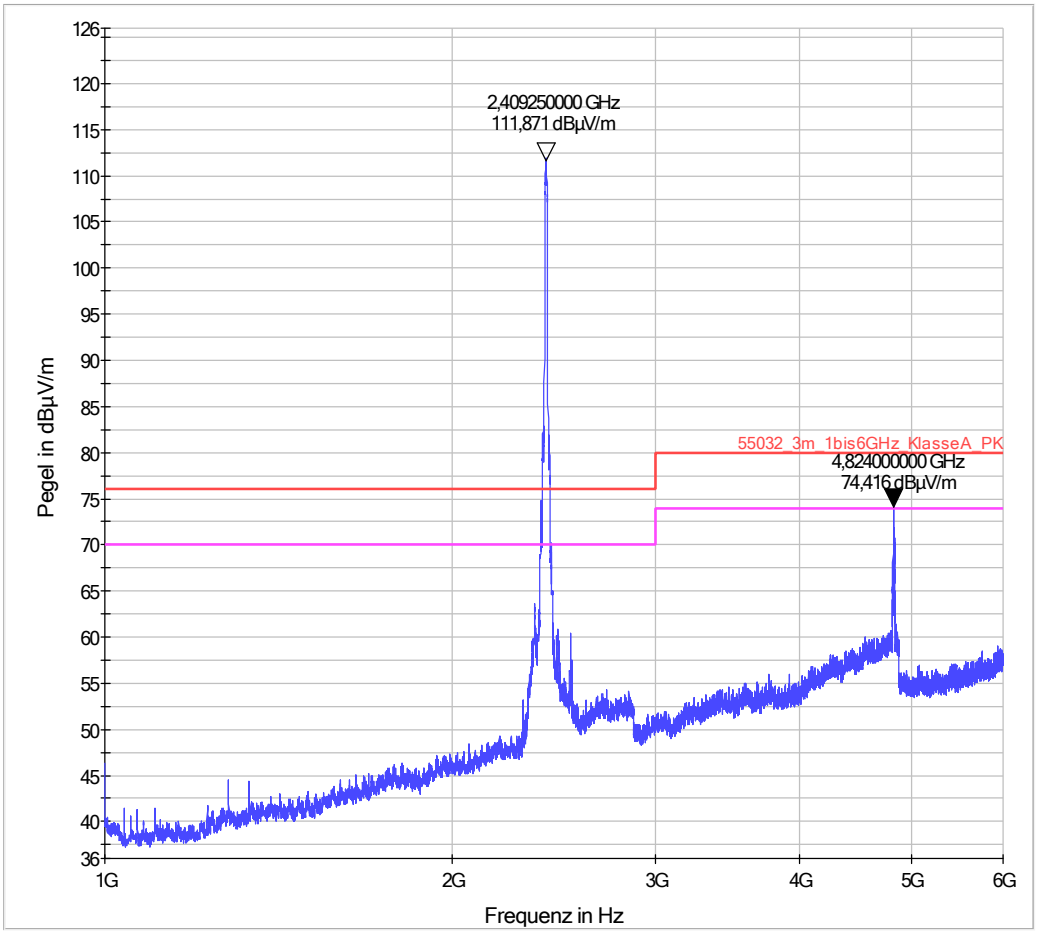
Vertikal / 90°



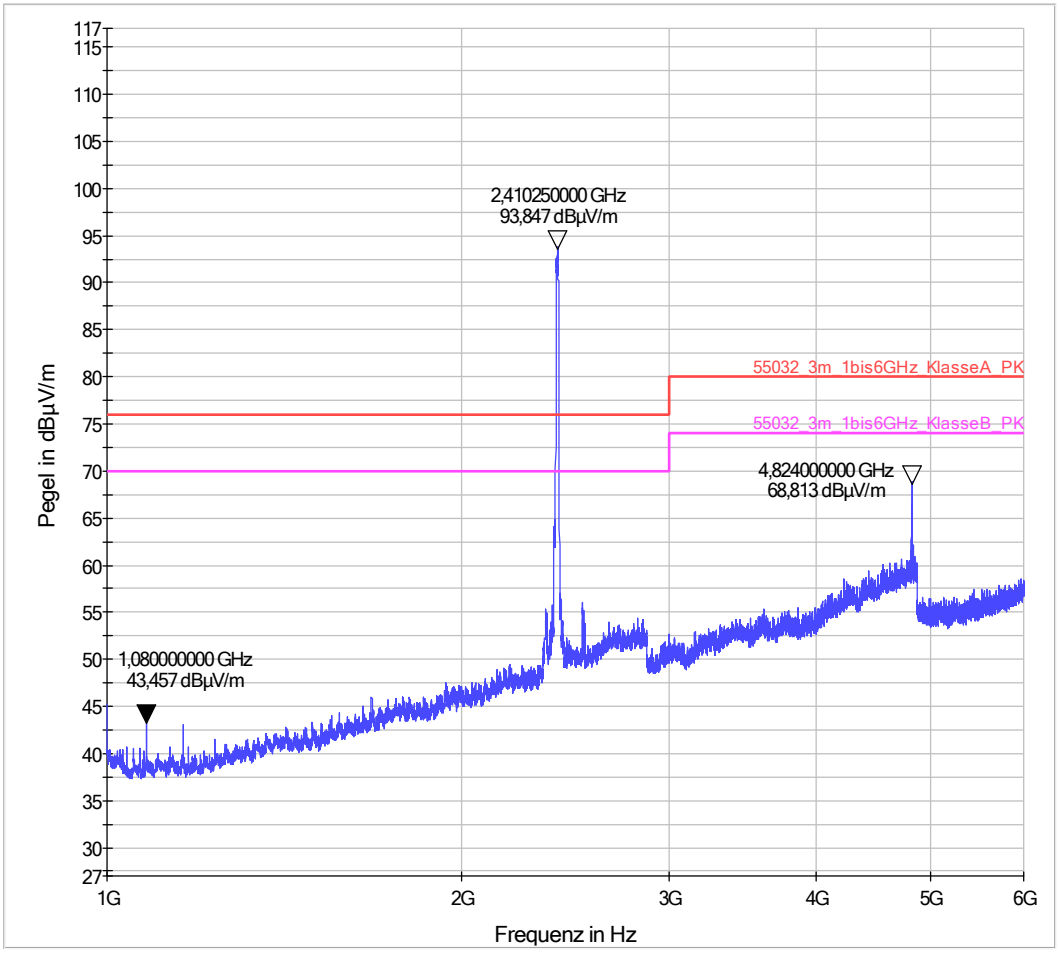
Horizontal / 180°



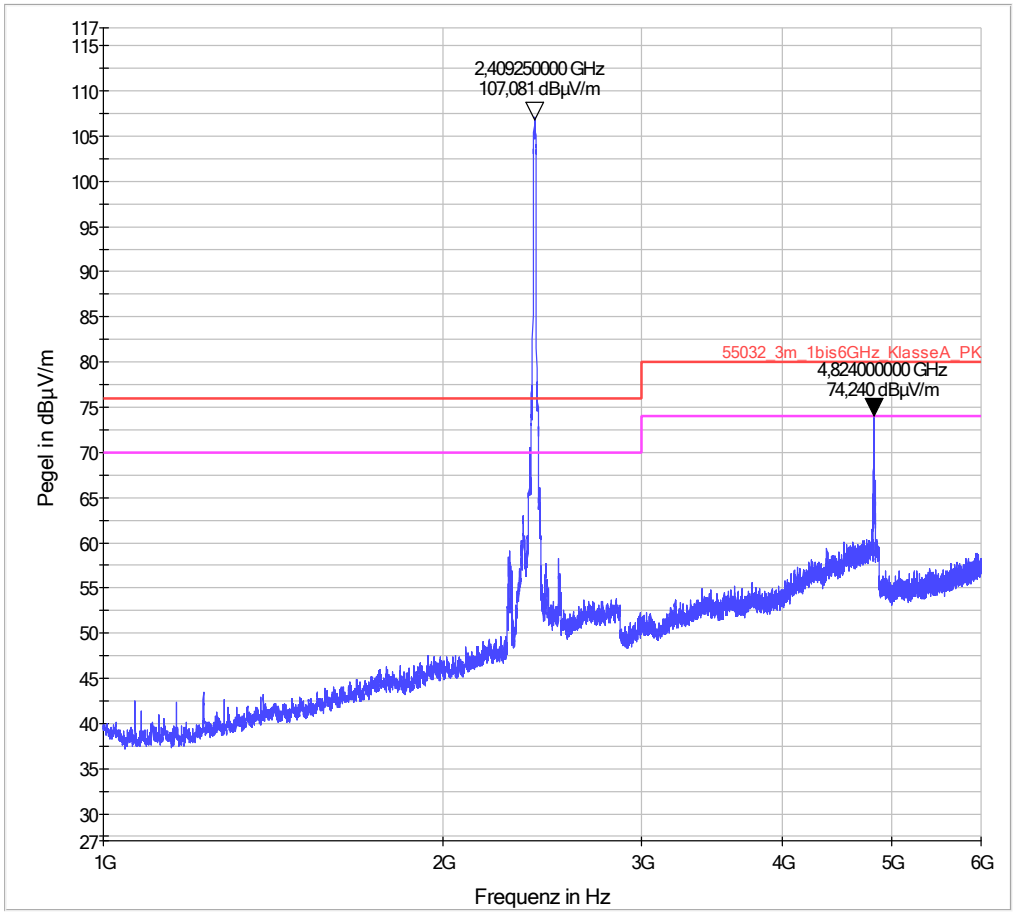
Vertikal / 180°



Horizontal / 270°



Vertikal / 270°



Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 07.08.2019
Ort, Datum

Unterschrift