UMTEC INSTITUT FÜR UMWELT- UND VERFAHRENSTECHNIK

OST Ostschweizer Fachhochschule

Visualisierung von Messdaten mittels Raspberry Pi

Studienarbeit

Studiengang Informatik
OST – Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Herbstsemester 2020

Autoren:
Fabian Thurnheer, Marco Gartmann

Betreuer:
Silvan Gehrig

Projektpartner:
Institut für Umwelt- und Verfahrenstechnik UMTEC

Abstract

Trinkwasser ist ein nicht selbstverständliches Gut. Besonders in Entwicklungsländern ist die Sensibilisierung für das Thema der Trinkwasserqualität von grosser Bedeutung. Für diese Sensibilisierung braucht es Wissen und die Möglichkeit, dieses Wissen auch in technisch schlecht erschlossene Gebiete bringen zu können.

Mit einem Raspberry Pi's und darauf installierter Moodle-Lernplattform wurde als Teil des COFER WASH[1] Projekts eine Lösung gefunden, um die für die Sensibilisierung nötige Theorie in Hochschulen solcher Gebiete bereitzustellen. Für das Institut für Umwelt- und Verfahrenstechnik UMTEC der OST stellte sich die Frage, wie diese Theorie nun mit Praxisbeispielen veranschaulicht werden kann, um das Verständnis zu diesem Thema weiter zu vertiefen.

Die mit dieser Studienarbeit entwickelte Anwendung befähigt das im genannten Projekt bereits eingesetzte Raspberry Pi, Messdaten von Sensoren zu erheben, welche per USB ans Gerät angeschlossen werden. Die erhobenen Messdaten werden dem Benutzer in einer Web-Applikation, welche auch in einen Moodle-Kurs integriert werden kann, graphisch dargestellt. Mit den angeschlossenen Sensoren kann somit die Wasserqualität anhand verschiedener Eigenschaften gemessen und analysiert werden.

Das Raspberry Pi agiert dabei als WLAN-Accesspoint, auf welchen sich die Benutzer mit ihrem Smartphone oder Laptop verbinden, um die Messwerte in der Web-App zu analysieren. Da die Web-App ebenfalls vom Raspberry Pi ausgeliefert wird, ist der Einsatz auch ohne Internetverbindung gewährleistet.

Für die Realisierung wurde die Programmiersprache TypeScript mit Einsatz von Node.js und Vue.js verwendet. Die Anwendung wurde mit Blick in die Zukunft so entwickelt, dass neue Sensor-Typen mit möglichst wenig Aufwand integriert werden können.

[1] Water, sanitation and hygiene, Projekt des swissuniversities Development and Cooperation Network

Lay Summary

Visualisierung von Messdaten mittels Raspberry Pi

Eine kostengünstige Messstation zur Vermittlung von Wissen über die Trinkwasserqualität.

Diplomanden	Fabian Thurnheer, Marco Gartmann
Betreuer	Silvan Gehrig
Themengebiet	Software Engineering – Core Systems
Industriepartner	Institut für Umwelt- und Verfahrenstechnik UMTEC, OST

Ausgangslage:

Die Sensibilisierung für die Sauberkeit von Trinkwasser erfordert Wissen und die Möglichkeit, dieses Wissen in technisch schlecht erschlossene Gebiete zu bringen. In einem vorangegangenen Projekt installierte das UMTEC mit weiteren Hochschulen zur Schulung eine Moodle-Box (Raspberry Pi inkl. Moodle) mit Kursen zu diesem Thema in afrikanischen Partner-Hochschulen. Um zur vermittelten Theorie auch Praxis-Übungen bieten zu können, soll mittels des Raspberry Pi und angeschlossen USB-Sensoren die Messung der Qualität von Wasserproben ermöglicht werden. Diese Studienarbeit hatte die Entwicklung einer Software zum Ziel, mit welcher Messdaten von USB-Sensoren erhoben und diese Werte in einer Web-App live dargestellt werden können.

Vorgehen/Technologien:

Nach einer Anforderungserhebung wurde zuerst analysiert, wie die Kommunikation mit den zur Verfügung gestellten Sensoren implementiert werden kann. Bei der anschliessenden Ausarbeitung der Software-Architektur galt es insbesondere darauf zu achten, dass künftig neue Sensor-Typen mit wenig Aufwand integriert werden können. Es entstand eine Anwendung, welche aus einer Single-Page Web-App, einem Backend und einer Hardware-Abstraktionsschicht besteht. Letztere ist dabei für die Kommunikation mit den Sensoren verantwortlich. Zwischen den Komponenten wurden Schnittstellen implementiert, über welche die Messdaten in Echtzeit bis zur Web-App gelangen, wo sie graphisch dargestellt werden.

Ergebnisse:

Mit Abschluss dieser SA wird eine Software abgeliefert, welche auf einem Raspberry Pi betrieben werden und Messwerte der daran angeschlossenen USB-Sensoren erheben kann. Über die Frontend Web-App können die Studierenden die laufenden Messungen in Echtzeit beobachten und so z.B. die Wasserqualität einer Probe analysieren. Dabei können sie zwischen einer Live-Value Ansicht und einer Graph-Ansicht wechseln. Mit letzterer können auch Messwerte dargestellt werden, welche vor dem Aufruf der Web-App erhoben wurden, was einen flexiblen Einsatz des Produkts in unterschiedlichen Anwendungsfällen ermöglicht.

Gesamtinhaltsverzeichnis

1. Aufgabenstellung

OST Ostschweizer Fachhochschule

Semesterarbeit

Thema: Visualisierung von Messdaten mittels Raspberry Pi

Autor:	Silvan Gehrig
Version:	1.0
Erstellt am:	13.09.2020
Letzte Änderung am:	22.09.2020

OST Ostschweizer Fachhochschule

Änderungsnachweis

Version	Änderungsgrund	Kurz-Z.	Datum
1.0	Initial	sgeh	13.09.20
1.1	Aufgabenbestellung	sgeh	19.09.20
1.2	Termine konkretisieren	sgeh	22.09.20

Inhaltsverzeichnis

OST Ostschweizer Fachhochschule

1. Einführung

Trinkwasser ist nicht selbstverständlich. Gerade in Entwicklungsländern ist das Thema «Trinkwasserversorgung» von grosser Bedeutung. Die Sensibilisierung für die Sauberkeit von Trinkwasser erfordert Wissen und die Möglichkeit, dieses Wissen in technisch schlecht erschlossene Gebiete zu bringen.

Wir wollen mittels Raspberry Pi die Messdaten von Sonden zur Wasserqualitätsmessung visualisieren, um die Bewertung von Wasser zu sensibilisieren.

2. Aufgabe

Das Ziel der Arbeit besteht darin, eine Live-Visualisierung von Messdaten verschiedener Sonden/Sensoren zu realisieren. Diese Live-Visualisierungen sollen in eine bestehende Moodle-Umgebung als externe Applikation integriert werden.

Die Aufgabenstellung kann bei Bedarf mutiert werden. Die Arbeit besteht aus den folgenden Hauptaufgaben:

2.1 Requirements Analysis

- Ausarbeiten der Anforderungen an die Messstation in Zusammenarbeit mit dem Auftraggeber.
- Festlegen der Messbereiche, Einheiten, Signifikante Grössen und Messintervalle für die jeweiligen Sensoren.

2.2 Backend und Hardware

- Erstellen einer Steuersoftware für die verschiedenen Sensoren.
  Sicherstellen des Systemstatus der Sensoren (connected / disconnected / …).
  Einführen eines HAL (Hardware Abstraction Layer) um die Wartbarkeit und Entwicklung sicherzustellen.
  Auslesen der Sensor-Werte mittels zugehöriger Low-Level Drivers.
- Anbieten der Messdaten über eine API (REST / GraphQL).

2.3 Frontend

- Visualisierung der Messdaten auf einer graphischen Oberfläche.
  Kategorisieren der Sensoren (PH, Redox, Light, Sound, Temperatur, Soil Moisture, Hygrometer) in verschiedene Typen sowie Ausarbeiten von passenden Visualisierungsvarianten.
- Integration des Frontend als externe Applikation in die Lernumgebung «MoodleBox».
- Ausarbeiten der Benutzerschnittstelle anhand Mobile First (Support von Handies und Tablets).

OST Ostschweizer Fachhochschule

3. Hinweise

3.1 Themen und Technologien

- Frontend
  - SPA, Vue mit Typescript
- Backend
  - Node.js, Node express, node-usb, napi
- Operation System
  - RaspbianOS (MoodleBox) oder alternativ Debian mit Moodle und MoodleBox Plugin

3.2 Hardware

Die folgende Hardware wird vom Auftraggeber UMTEC zur Verfügung gestellt und bildet die Integrationsplattform der Semesterarbeit.

- Basis Platine
  - Raspberry Pi 3 / 4
- Sensoren
  - Pico Technologies DrDAQ
    - Temperature sensor (built-in)
    - Light sensor (built-in)
    - Sound sensor (built-in)
    - PH sensor
    - Redox sensor
  - Tinovi USB Sensor
    - Soil Moisture, Temperature Sensor
  - Temperatur & Humidity Measurement Sensor TEMPer1F_H1
    - Hygrometer (Humidity) sensor, Temperature sensor

4. Erwartete Resultate

Als Resultat soll eine dokumentierte und auf dem Raspberry Pi lauffähige Applikation abgegeben werden. Eine gute Wartbarkeit der Software soll die Weiterentwicklung sicherstellen. Weiteres gilt zu beachten:

- Vorgehen nach den Regeln des Software-Engineering (Scrum und UP), Arbeit nach Projektplan. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten.
- Erfassen des tatsächlichen Arbeitsaufwands. Es muss ersichtlich sein, wer für welchen Teil der Arbeit und des Berichts verantwortlich ist.
- Alle Dokumente sind nachzuführen, d.h. sie sollen den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren.

OST Ostschweizer Fachhochschule

5. Termine

Die verbindlichen Termine für die Semesterarbeit finden sich in den Studiengangrichtlinien unter: \\hsr.ch\root\alg\skripte\Informatik\Fachbereich\Studienarbeit_Informatik\SAI14\Termine .

Die zeitliche Planung der weiteren Meetings wird von den Studenten durchgeführt:

- Ende der Elaboration (ca. in 1/3 des Projekts) findet ein Meeting statt, bei welchem von den Studenten der «Durchstich» /demonstriert wird.
- Mittels einer Abschlusspräsentation soll das Erreichte dem Auftraggeber demonstriert werden.
- Es findet jede Woche ein (oder alternativ alle 2 Wochen ein längeres) Statusmeeting mit dem organisatorischen Betreuer statt. Zusätzliche Besprechungen sind nach Bedarf zu veranlassen.

  Während des Status-Meeting werden anhand des Projektplanes folgende Punkte erläutert:
  - Was wurde erreicht?
  - Wie viele Stunden wurden geleistet?
  - Wo gab es Probleme, werden Hilfestellungen benötigt?
  - Wie ist das weitere Vorgehen?
- Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und Beschlüsse in einem Protokoll zu dokumentieren, das den Betreuern per E-Mail zugestellt wird.

6. Betreuung und Ansprechpartner

6.1 Auftraggeber

- Product Owner: Institut für Umwelt- und Verfahrenstechnik, Mirko Rohr
- Stake Holder: Institut für Umwelt- und Verfahrenstechnik, Michael Burkhardt, Institutsleiter UMTEC

6.2 Technische Betreuung

- USB Driver Programming: Institut für Software, Felix Morgener
- Node.js, vue.js, TypeScript: Institut für Software, Silvan Gehrig

6.3 Organisatorische Betreuung

- Stake Holder: IFS, Institut für Software, Silvan Gehrig

7. Beurteilung

Eine erfolgreiche Studienarbeit zählt 8 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von ca. 25 bis 30 Stunden budgetiert (vgl. M_SAI14 für die Modulbeschreibung der Studienarbeiten). Für die Beurteilung ist der organisatorische Betreuer verantwortlich unter Einbezug des Feedbacks des Auftraggebers.

OST Ostschweizer Fachhochschule

Die Beurteilung der Arbeiten erfolgt nach einem einheitlichen Bewertungsschema:

Gesichtspunkt	Gewicht
1. Organisation, Durchführung (Projektplanung u. Nachführung Arbeit gemäss Projektplan, Selbständigkeit, Einsatz, Zusammenarbeit mit Auftraggeber, Betreuerin oder Betreuer)	1/5
2. Bericht (Inhalt des Projektschlussberichts, Gliederung, Darstellung, Sprache der gesamten Dokumentation)	1/5
3. Resultat der Arbeit	
3.1 Problemanalyse (Vorstudie, Literaturstudium, Anforderungsspezifikation, Anforderungsanalyse, Domainanalyse)	1/5
3.2 Lösungsentwurf (Lösungsvarianten und deren Beurteilung, Variantenentscheid, Konzept, Entwurf)	1/5
3.3 Realisierung und Test	1/5

Diese Aufstellung basiert auf den Regelungen Studien- und Bachelorarbeiten im Studiengang Informatik \\hsr.ch\root\alg\skripte\Informatik\Fachbereich\Studienarbeit_Informatik\SAI14\ . Es gelten die Bestimmungen der Abteilung Informatik zur Durchführung von Studienarbeiten.

Rapperswil-Jona, 22. September 2020

Silvan Gehrig

Institut für Software

OST – Ostschweizer Fachhochschule

2. Technischer Bericht

2.1 Ausgangslage & Problembeschreibung

Als Mitglied des COFER WASH[2] Clusters von swissuniversities, der Dachorganisation der Schweizer Hochschulen, beteiligte sich das Institut für Umwelt- und Verfahrenstechnik UMTEC der OST (folgend UMTEC genannt) aktiv an Projekten zum Thema Trinkwasser. Das dem Cluster übergeordnete Organ CLOC[3] West Africa definiert sein Ziel wie folgt:

"Das übergeordnete Ziel des CLOC West Africa – "Network for Water and Life" (NEWAL) – ist die Förderung und Stärkung des Verständnisses, der Vernetzung und des wissenschaftlichen Austausches zwischen schweizer und westafrikanischen Institutionen an der Schnittstelle von Wasser und Leben." [1] (Übersetzung des Autors).

Als Teil des COFER WASH Clusters beteiligte sich das UMTEC in der Vergangenheit an einer Lösung, um Wissen und Expertise zu den genannten Themen auch in technisch schlecht erschlossene Gebiete bringen zu können. Dabei lieferte man sogenannte Moodle-Boxen an Hochschulen und Universitäten der afrikanischen Partner-Institutionen. Eine Moodle-Box ist im eigentlichen Sinn ein portabler Raspberry Pi Minicomputer, auf welchem eine Instanz der Lernplattform Moodle installiert ist. Für diese Lernplattform wurden Kurse zum Thema der Wasserqualität vorbereitet, mit welchen die Moodle-Boxen dann ausgeliefert werden konnten. Da diese als WLAN-Access Point agieren, können Studierende auch ohne Internetverbindung mit ihrem Smartphone oder Laptop über das Moodle-Box WLAN auf die Moodle-Webseite und die Kursinhalte zugreifen.

Aus eigener Erfahrung wissen wir, dass gelernte Theorie durch die praktische Anwendung besser verstanden und auch weiter vertieft werden kann. Wo Studierende der Ostschweizer Fachhochschule am Standort Rapperswil von modernen Labor-Einrichtungen profitieren können, um das Gelernte praktisch anzuwenden, verfügt hingegen nicht jede Hochschule oder Universität über solche Labors. [2]

2.2 Vision

Als Auftraggeber und Projektpartner dieser Studienarbeit sah das UMTEC die Möglichkeit, die bereits eingesetzten Raspberry Pi's als Hub für USB-Sensoren einzusetzen. Dabei kann mit dieser Lösung ein kostengünstiges, portables Labor bereitgestellt werden. Die Stärken dieser Lösung sehen wir als Entwicklerteam vor allem darin, dass dieses "Labor" auch von Institutionen mit geringerem Budget angeschafft und betrieben werden kann. Durch die Portabilität des Raspberry Pi's, welches in etwa die Grösse einer handelsüblichen Zigarettenschachtel hat, kann es zusammen mit den Sensoren zudem einfach verschoben werden. Sofern die Stromzufuhr gewährleistet ist, könnte es somit auch für Messungen und Experimente ausserhalb der Hochschul-Gebäude eingesetzt werden.

In dieser Studienarbeit geht es in erster Linie darum, eine funktionierende Applikation für die oben genannten Szenarien zu erstellen, welche vom UMTEC genutzt und anderen Bildungsinstitutionen vorgestellt werden kann. Dafür soll, wie in der Aufgabenstellung beschrieben, neben dem Backend der Applikation auch eine Frontend Web-Applikation entwickelt werden, welche direkt in einen Moodle-Kurs integriert werden kann.

Die Vision ist, dass die entwickelte Applikation in einem zukünftigen Projekt des COFER WASH Clusters effektiv an ausländischen Hochschulen eingesetzt werden könnte und dabei Studierenden die Analyse von Wasser- oder anderen Proben ermöglicht.

[2] Consortia for Education and Research (COFER); Water, Sanitation and Hygiene (WASH)
[3] Cluster of Cooperation (CLOC)

2.3 Lösungskonzept

2.3.1 Systemumfeld

Die folgende Abbildung soll aufzeigen, in welchem Kontext die konzipierte Anwendung VMRP am Ende der Arbeit zu anderen Systemen steht. Ebenfalls werden dabei die Systemgrenzen aufgezeigt. Was ausserhalb des mit VMRP gekennzeichneten Bereichs steht, liegt nicht im Einflussbereich dieser Arbeit.

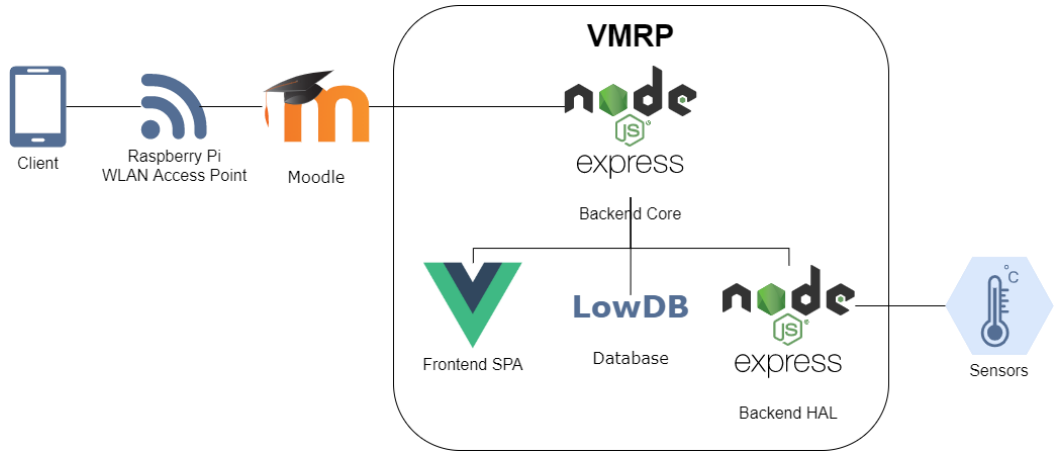

Abbildung 1: Systemübersicht der entwickelten Anwendung

In der obigen Abbildung ist ersichtlich, dass sich die Clients über das WLAN des Raspberry Pi's auf die Moodle-Lernplattform verbinden. Die in dieser SA entwickelte Anwendung soll im Moodle integriert werden, damit Benutzer/-innen die Messdaten in der ihnen bekannten Moodle-Oberfläche analysieren können.

2.3.2 Endgeräte

In der mit Herr Mirko Rohr, Product Owner seitens UMTEC, durchgeführten Anforderungsanalyse stellte sich heraus, dass die Mehrheit der Studierenden in den afrikanischen Partner-Hochschulen die Applikation über ihr Smartphone abrufen würde. Die Entwicklung der Single Page Applikation wurde daher anhand des "Mobile First" Ansatzes vollzogen und somit sichergestellt, dass sie Applikation auf mobilen Geräten bedienbar ist.

2.3.3 Hardware

Für die Realisierung der Anwendung wurden von Seiten des UMTEC ein Raspberry Pi 4B, wie auch drei Sensoren der folgenden Typen zur Verfügung gestellt:

Geräte-Modell	Integrierte Sensoren
Temper1F_H1	Hygrometer, Temperatur, Luftfeuchtigkeit
Tinovi	Bodenfeuchtigkeit, Temperatur, elektrische Leitfähigkeit
Pico Technologies DrDAQ	Temperatur, Lichtstärke, Sound Schallwellenfrequenz, Sound Lautstärke, pH Sensor. Zudem können über drei externe Eingänge weitere Sensoren angeschlossen werden.

Tabelle 1: zur Verfügung gestellte Sensor-Geräte

Am Anfang der Realisierung der Kommunikation mit den Sensoren stand die Analyse dazu, wie die verschiedenen Sensor-Typen angesprochen werden können, was für APIs sie anbieten und mit was für Libraries die Kommunikation realisiert werden kann.

Die Analyse ergab, dass sich die Sensoren nicht einheitlich ansprechen lassen, wie folgende Abschnitte zeigen.

Temper1F_H1: Dieser Sensor implementiert das HID USB-Profil und tritt somit als tastaturähnliches Eingabegerät auf, wenn es an einen USB-Eingang angeschlossen wird. Da es sich bei diesem Sensor um den nominell ungenausten der drei zur Verfügung gestellten handelt, wurde seine Integration in die Anwendung tiefer priorisiert als die Integration der beiden anderen Sensoren.

Tinovi: Für die Kommunikation mit dem Tinovi Sensor bot sich der Einsatz einer externen Library an, mit welcher sich der Sensor direkt aus der zu entwickelnden Applikation ansprechen lässt. Da dieses Gerät als serielles USB-Gerät auftritt und keine eigenen Treiber voraussetzt, konnte die Kommunikation ohne grosse Schwierigkeiten realisiert werden.

Pico Tech. DrDAQ: Schwieriger gestaltete sich die Kommunikation mit dem DrDAQ Sensor-Board. Da dafür die Treiber des Herstellers auf dem System installiert werden mussten, konnte das Gerät nicht über den generischen, universellen USB Device Treiber angesprochen werden. Für die Kommunikation wurde nach einer Analyse mehrerer möglicher Varianten als Lösung der Einsatz der herstellereigenen Python-Wrapper Library evaluiert, mit welcher die C-Funktionen des Treibers ausgeführt werden konnten.

2.3.4 Software Architektur

Wie in der Aufgabenstellung beschrieben, soll sowohl eine Frontend Web-Applikation wie auch ein Backend implementiert werden. Aufgrund der Hardware-Nähe des Backend wurde entschieden, den hardwareabhängigen Teil in eine eigene Software Komponente, einen Hardware Abstraction Layer (HAL), auszulagern.

Die daraus resultierenden drei Softwarekomponenten sind:

- Frontend Single Page Applikation: Webseite, mit welcher der Benutzer die Messdaten analysiert.
- Backend Core: soll für die Persistenz der Messdaten verantwortlich sein.
- Backend HAL: übernimmt komplette Kommunikation mit der Hardware, resp. den Sensoren.

Die einzelnen Softwarekomponenten sollten über APIs miteinander verbunden werden. Statische Daten, wie z.B. Informationen über die zurzeit angeschlossenen Sensoren oder historische Messdaten, werden dabei über ein HTTP API abgefragt. Für die Übermittlung der Echtzeitdaten, wie Werte laufender Messungen oder Hardware-Events, wurde ein Websocket API konzipiert. Dabei registriert sich das Frontend beim Core als Websocket Client, um diese Echtzeitdaten zu erhalten, und der Core Teil des Backend sich seinerseits beim HAL, welcher die Messwerte der laufenden Sensoren sendet.

Wie weiter oben erwähnt, wurden drei verschiedene Sensor-Typen zur Verfügung gestellt. Eine Anforderung an die Architektur war, dass sie es ermöglicht, neue und bislang nicht in die Anwendung eingebundene Sensoren in die Architektur zu integrieren und betreiben zu können.

2.4 Umsetzung

Nachdem im vorangegangenen Kapitel die grundlegenden Konzepte und die grobe Architektur bereits beschrieben wurden, liefert dieses Kapitel Erklärungen für ausgewählte Implementations-Details.

2.4.1 Typisierung des Frontend

Wie die Aufgabenstellung festhält, ist für das Frontend Vue.js mit TypeScript verwendet worden. Die zum Startpunkt des Projektes zur Verfügung stehende Version von Vue.js verfügte aber nur über einen schwachen TypeScript Support.

Der Einsatz von geeigneten Dependencies konnte dem entgegengewirken. Diese erlaubten es, die ursprünglich aus JavaScript-Objekten bestehende Logik der Vue.js Components mit entsprechenden TypeScript Klassen zu ersetzen. Mittels Decorators ist es nun möglich, Vue.js spezifische Funktionen und Eigenschaften auch mit den neuen TypeScript Klassen weiterhin zu verwenden. Dies erhöhte die Leserlichkeit, die Wartbarkeit und korrekte Typisierung des Codes.

Die Datenhaltung und Business Logic wurde im Frontend mittels Vuex Store implementiert. Vuex ist ein Plugin für Vue.js, welches einem ermöglicht, ein State Management Pattern in Form einer Library zu verwenden. So kann ein zentraler Store erstellt werden, welcher den State der Applikation mit beliebigen Vue.js Components teilt.

Dieser besteht normalerweise aus JavaScript Objekten, welche wiederum in Modulen gegliedert sind. Da Vue.js Components mit dem Store interagieren, musste auch hier eine Lösung gefunden werden, um die Module des Stores in TypeScript Klassen umzuwandeln und so zu typisieren. Dies gelang mit einer entsprechenden Dependency, welche ebenfalls mit Decorators arbeitet. Die daraus resultierende Vue.js Applikation ist zur Compilezeit vollständig typisiert, was ohne den Einsatz der Dependencies nicht möglich gewesen wäre.

2.4.2 Umsetzung DrDAQ Anbindung

Um das Pico Technologies DrDAQ Sensor-Board in die Anwendung einzubinden, wurde, wie bereits im vorangegangenen Kapitel beschrieben, nach einer Analyse der verschiedenen Möglichkeiten die Verwendung der herstellereigenen Python Wrapper Library als beste Lösung evaluiert.

Gegen die direkte Verwendung der C-Funktionen des Gerätetreibers und für die Lösung mit Python sprach, dass die Sprache Python näher an den von uns bereits häufiger eingesetzten Programmiersprachen ist. Dadurch konnten die für die Kommunikation nötigen Funktionalitäten schneller implementiert werden.

Im Folgenden wird die Funktionsweise und die Implementation zusammengefasst erklärt:

- Für die Kommunikation mit dem DrDAQ Board wurde ein eigenes Python Skript entwickelt, welches in einem konfigurierbaren Intervall die Messwerte sämtlicher aktiven Sensoren auf dem Board erhebt.
- Für die parallele Ausführung des Skripts und der eigentlichen HAL Node.js Applikation wird in letzterer ein Child-Prozess erzeugt, welcher das Python Skript nebenläufig ausführt, sobald das Board per USB ans Raspberry Pi angeschlossen wird.
- Um die Messwerte vom Python Skript in die Node.js Applikation weiterzuleiten, kommunizieren die beiden Prozesse über den STDOUT und den STDIN File-Deskriptor des Child-Prozesses.
- Dabei gibt der Child-Prozess, welcher das Python Skript ausführt, die erhobenen Messwerte über seinen STDOUT File-Deskriptor aus. In der Node.js Applikation kann mit der eingesetzten Library auf solche STDOUT-Events reagiert, die Daten von dort aufgegriffen und diese dann weiterverarbeitet werden. Es handelt sich dabei also um eine klassische File-basierte Interprozess-Kommunikation mit einem Producer und einem Consumer.

Genauere Implementationsdetails des Python Skripts finden sich im Softwaredesign Kapitel im folgenden Teil des Dokuments.

2.4.3 Moodle Integration

Die Frontend Web-Applikation konnte unkompliziert als iframe in einen Kurs oder eine Inhaltsseite eines Kurses integriert werden. Moodle bietet von Haus aus die Möglichkeit, den Inhalt eines Kurses oder einer Seite mit einem HTML-Editor zu bearbeiten. In diesen konnte das iframe eingefügt werden. Da dieser Prozess von künftigen Benutzern der Anwendung durchgeführt wird, wurde er in einem User Guide schrittweise erklärt.

2.4.4 Deployment via Docker

Das Deployment und das Updaten der Anwendung stellte eine Herausforderung dar, da die Anwendung auf Raspberry Pi's verteilt werden musste, welche nur unregelmässig eine Verbindung ins Internet haben. Obschon das Erarbeiten einer Deployment-Strategie nicht Teil der Aufgabenstellung war, wurde mit der Anwendung von Docker und Docker Compose in dieser Arbeit die Grundlage für ein künftiges Deployment gelegt. Mittels Docker können mehrere Software Komponenten in einem Image gebündelt und dank Docker Compose mit einem einzelnen Befehl gestartet werden.

Das Bilden der Docker Images wurde in die GitLab CI Pipeline integriert und die erstellten Images in der Container Registry des IFS[4]-eigenen GitLab veröffentlicht. Beim Build gilt zu erwähnen, dass die CPUs auf den Raspberry Pi Geräten eine ARM Architektur besitzen, der Runner des GitLabs jedoch auf einem x86 Prozessor betrieben wird. Docker Images können nur auf jener Prozessor-Architektur ausgeführt werden, für welche sie erstellt wurden. Folglich musste im GitLab Runner ein Docker-in-Docker Image mit installiertem Docker BuildX Plugin angewendet werden, um die Backend Images für die ARM Architektur bilden und auf einem Raspberry Pi ausführen zu können.

Durch die Verwendung der Docker Images ergeben sich für den Kunden, das UMTEC, folgende Vorteile:

- Auf den Raspberry Pi Geräten wird lediglich das Docker Compose File benötigt, um die aktuellen Images von der IFS GitLab Registry herunterzuladen und damit die komplette Anwendung zu starten.
- Die für die Sensoren nötigen Treiber sind Teil der Images und müssen nicht auf jedem Raspberry Pi neu installiert werden.
- Damit die künftigen Benutzer die Docker Images nicht bei jeder Verwendung über eine SSH Verbindung starten müssen, wurden die Images im Docker Compose File so konfiguriert, dass sie automatisch gestartet werden, wenn das Raspberry Pi an den Strom angeschlossen wird. Die gesamte Anwendung steht damit ohne Zutun der Benutzer zur Verfügung.
- Ein automatisiertes Update der Docker Images könnte in Zukunft noch realisiert werden, sodass die Raspberry Pi Geräte im Hintergrund die neuen Images von der Registry herunterladen, sobald sie eine Verbindung ins Internet haben. So könnte man bei Updates neue Images veröffentlichen, welche von sämtlichen vom UMTEC verteilten Raspberry Pi's selbstständig gepullt würden.

2.5 Ergebnisdiskussion & Ausblick

2.5.1 Ergebnis

Mit Abschluss dieser SA wird eine Software abgeliefert, welche auf einem Raspberry Pi betrieben werden und Messwerte der daran angeschlossenen USB-Sensoren erheben kann. Über die Frontend Web-App können die Studierenden die laufenden Messungen der angeschlossenen Sensoren in Echtzeit beobachten und so z.B. die Wasserqualität einer Probe analysieren. Dabei können sie zwischen einer Live-Value Ansicht und einer Graph-Ansicht wechseln. Mit letzterer können auch Messwerte dargestellt werden, welche vor dem Aufruf der Web-App erhoben wurden, was einen flexiblen Einsatz des Produkts in unterschiedlichen Anwendungsfällen ermöglicht.

Die Web-Applikation kann zudem direkt in einen Moodle-Kurs integriert werden, was eine einfache Einbindung der Anwendung in die bestehenden Unterrichtsformen ermöglicht.

[4] Institute for Software, OST

2.5.2 Erreichte Ziele

In der Aufgabenstellung werden die Ziele der Arbeit wie folgt definiert: "Das Ziel der Arbeit besteht darin, eine Live-Visualisierung von Messdaten verschiedener Sonden/Sensoren zu realisieren. Diese Live-Visualisierungen sollen in eine bestehende Moodle-Umgebung als externe Applikation integriert werden." [3]

Anhand der im Kapitel Anforderungsanalyse beschriebenen Funktionalitäten werden die in der Aufgabenstellung definierten Ziele als erfüllt erachtet.

Sämtliche als Kernfunktionalität eingestufte Funktionen konnten erfolgreich implementiert werden.

2.5.3 Mögliche Verbesserungen

Obschon sämtliche Kernfunktionalitäten implementiert werden konnten, reichte die verbleibende Zeit nicht aus, um die als Erweiterungsfunktionen definierten Features im Rahmen dieser Studienarbeit zu realisieren. Namentlich sind dies folgende Funktionen:

- Anpassung des Messintervalls durch den Benutzer: Der Benutzer kann steuern, in welchem Intervall ein Sensor Messungen vornehmen soll.
- CSV Export: Die gespeicherten Messdaten eines Sensors lassen sich als CSV exportieren, um sie in diesem Format weiter bearbeiten zu können.
- Threshold Alarmierung: Der Benutzer kann für einen Sensor einen oberen und unteren Grenzwert definieren. Wird dieser Grenzwert unter-/überschritten, wird eine visuelle Meldung in der Web-App angezeigt.

Zudem existiert bei folgenden Punkten noch Verbesserungspotenzial:

- Performance beim Laden von aufgezeichneten Messungen: Bei einer getesteten Langzeitmessung, in welcher 24h lang von acht Sensoren alle 2s eine Messung erfasst wurde, dauerte das Laden der Frontend Web-App überdurchschnittlich lange. Eine vertiefte Analyse der Ursache konnte aus zeitlichen Gründen nicht mehr durchgeführt werden.
- Rendern der Charts optimieren: Momentan wird bei jeder Messung im Frontend die gesamte Chart-Komponente neu gerendert. Dies liesse sich noch dahin optimieren, dass nur noch der Graph neu gezeichnet wird. Eventuell bedürfte es dazu einem Wechsel der eingesetzten Chart-Library.
- Historische Daten könnten beim Anschliessen eines Sensors direkt über Websocket mitgesendet werden, um den HTTP Request einzusparen.
- Der Zeitpunkt der Messdatenerhebung wird momentan im UTC Zeitformat angegeben und gespeichert. In einer weiteren Entwicklung der Anwendung könnte implementiert werden, dass dafür die lokale Zeitzone des Benutzers verwendet wird.
- Automatisiertes Update der Docker Container: Zurzeit werden die Docker-Container auf dem Raspberry Pi lediglich gestartet. Dies könnte mit einer Update-Strategie erweitert werden, sodass ein Raspberry Pi im Hintergrund die neuen Images von der Registry herunterlädt, sobald er mit dem Internet verbunden ist.

2.5.4 Ausblick

Im Anschluss an die Studienarbeit wird das fertige Produkt dem UMTEC präsentiert und näher erklärt. Danach ist für das UMTEC eines der folgenden Zukunftsszenarien für die Anwendung denkbar [4]:

- Einsatz der Anwendung für die Lehre an Orten mit schwacher Infrastruktur. Beispielsweise können in einem Land wie der Elfenbeinküste im Unterricht kleinere Experimente mit dem Ziel durchgeführt werden, Lerninhalte zu verifizieren.
- Ebenfalls denkbar ist die Weiterentwicklung des Produkts für eine konkrete oder umfangreichere Anwendung.

2.6 Abbildungsverzeichnis

2.7 Tabellenverzeichnis

2.8 Literaturverzeichnis

[1] swissuniversities, «CLOC West Africa,» [Online]. Available: https://www.swissuniversities.ch/themen/entwicklung-und-zusammenarbeit/p-6-swissuniversities-development-and-cooperation-network-sudac/cloc-west-africa. [Zugriff am 10 Dezember 2020].

[2] M. Rohr, Interviewee, *Sitzung zur Anforderungsanalyse.* [Interview]. 07 September 2020.

[3] S. Gehrig, «Aufgabenstellung Semesterarbeit "Visualisierung von Messdaten mittels Raspberry Pi",» Rapperswil, 2020.

[4] R. Stalder, Interviewee, *E-Mail Verkehr zur weiteren Verwendung der Anwendung.* [Interview]. 13 Dezember 2020.

3. Software Engineering Dokumente

3.1. Domainanalyse

SA - VMRP

Autoren: Marco Gartmann, Fabian Thurnheer
Version: 1.0

Erstellt am: 25.09.2020
Letzte Änderung am: 18.12.2020

Inhaltsverzeichnis

1. Einführung

1.1 Zweck

Dieses Dokument beschreibt die Domäne der Arbeit und beinhaltet das Domain Modell als Abbildung der realen Umgebung.

1.2 Gültigkeitsbereich

Die Gültigkeit des Projektplans beschränkt sich auf die Laufzeit des Moduls "Studienarbeit" im Herbstsemester 2020.

2. Domain Modell

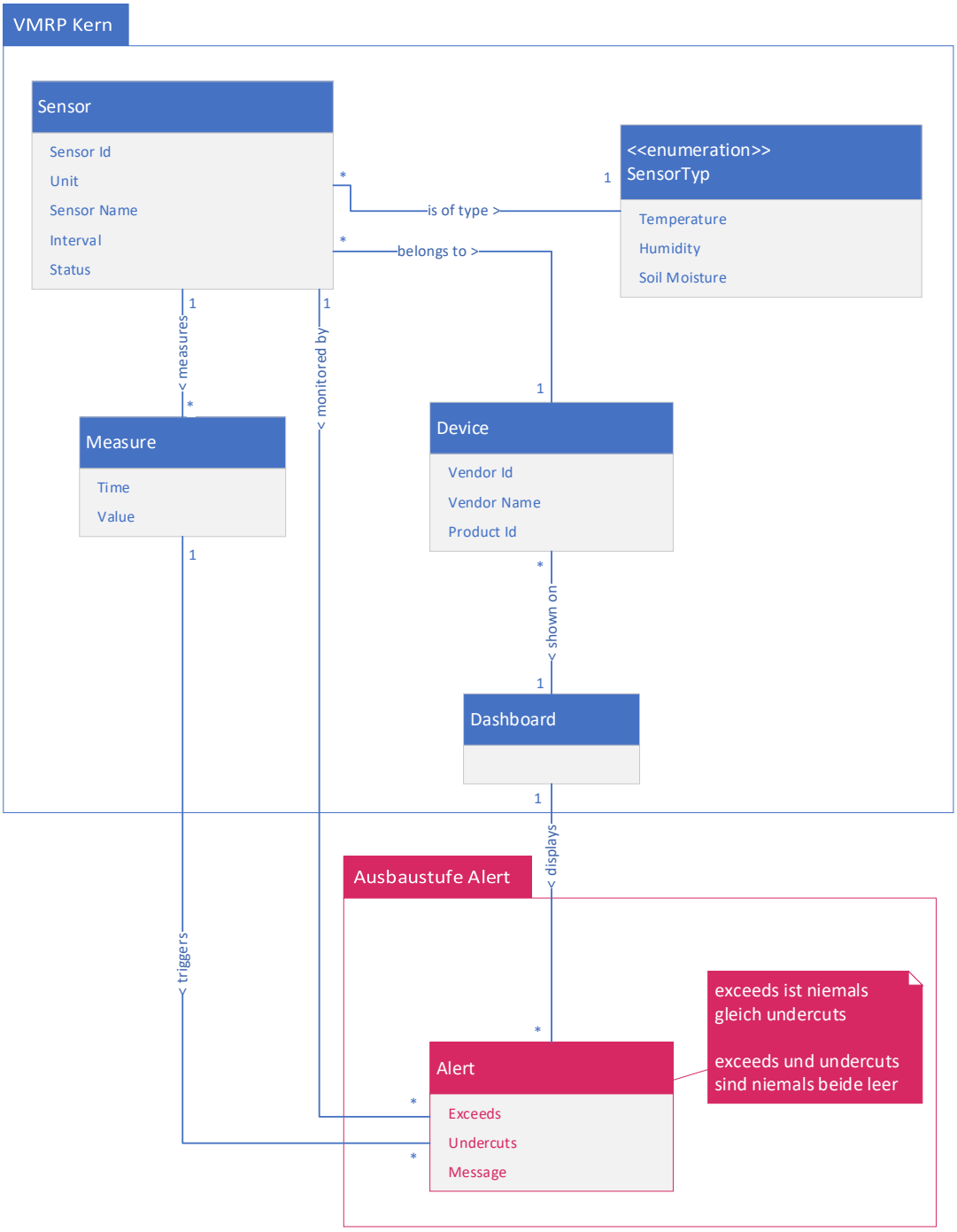

Abbildung 2: Domain Modell

Das obenstehende Domain Modell zeigt, welche Komponenten der realen Welt als Teil der Problem Domain zählen und wie diese zusammenhängen. Dieses Modell dient als Grundlage für die Realisierung der Entitätsklassen in der Software. Die zentralste Entität befindet sich oben links, gegen rechts unten nehmen die Entitätsklassen eine weniger zentrale Rolle ein. Zuunterst steht demzufolge die Ausbaustufe "Alert".

2.1 Wichtige Konzepte

2.1.1 Sensor

Feld	Bedeutung	Beispiel
Sensor Id	Vergebene SensorId	[1, ∞)
Unit	Physikalische Einheit, mit welcher die Werte dargestellt werden sollen.	°C, %, mS
Sensor Name	Name des Sensors	Temperature Sensor
Interval	Angabe des Abfrageintervalls in Sekunden	[1, ∞)

Tabelle 2: Sensor

Ein Sensor repräsentiert einen physischen Sensor, welcher auf einem USB-Gerät installiert ist oder an ein solches als externer Sensor angeschlossen wird. Er beinhaltet alle Metadaten des physischen Sensors.

2.1.2 Measurement

Feld	Bedeutung	Beispiel
Time	Zeit, an welchem eine Messung vorgenommen wurde.	2020-10-02 12:12:43
Value	Gemessener Wert eines Sensors	24.55 (°C)

Tabelle 3: Measurement

Eine Measurement repräsentiert einen einzelnen Messwert, welcher die Durchführungszeit der Messung und den gemessenen Wert beinhaltet.

2.1.3 Device

Feld	Bedeutung	Beispiel
Vendor Id	Die auf dem Device vom Hersteller eingebrannte Hersteller-ID zur Identifikation des Herstellers.	43289
Vendor Name	Name des Herstellers	"Pico Technologies"
Product Id	ID, um angeschlossenes Gerät identifizieren zu können. Diese Hardware ID stammt ebenfalls vom Hersteller.	2349

Tabelle 4: Device

Die Klasse Device stellt ein physisches USB-Device dar, welches an den Raspberry Pi angeschlossen wird. Ein Device kann physisch mehrere Sensoren beherbergen.

3. Tabellenverzeichnis

4. Abbildungsverzeichnis

3.2. Anforderungsanalyse

SA - VMRP

Autoren: Marco Gartmann, Fabian Thurnheer
Version: 1.0

Erstellt am: 16.09.2020
Letzte Änderung am: 18.12.2020

Inhaltsverzeichnis

1. Einführung

1.1 Zweck

Dieses Dokument dient der Spezifikation der Anforderungen, welche an die in der Studienarbeit zu entwickelnde Software gestellt werden.

1.2 Gültigkeitsbereich

Dieses Dokument ist gültig im Zusammenhang mit der Studienarbeit 2020 VMRP.

2. Allgemeine Beschreibung

2.1 Produktfunktionen

Im Hochschulen-Verbund swissuniversities führte das Institut für Umwelt- und Verfahrenstechnik UMTEC der OST zusammen mit Partneruniversitäten in Äthiopien und der Elfenbeinküste das Projekt COFER WASH durch, welches die Vermittlung von (Anwendungs-)Wissen zur Wasserhygiene zum Ziel hatte [1]. Um in den afrikanischen Partneruniversitäten Schulungen zum genannten Thema internetunabhängig anbieten zu können, wurden Kurse für eine MoodleBox entwickelt, welche in einem Schulzimmer platziert werden kann. Die MoodleBox agiert dabei als WLAN Access Point, auf welchen sich Studenten verbinden und so auf die Moodle Lernplattform zugreifen und die Kursinhalte abrufen können [2].

Das Ziel der Arbeit besteht darin, eine Live-Visualisierung von Messdaten verschiedener Sensoren zu realisieren. Diese Live-Visualisierungen sollen in eine bestehende Moodle-Umgebung als externe Applikation integriert werden.

Das Produkt soll ein Frontend als Single Page Applikation (SPA) zur Verfügung stellen, welche Messdaten von den an die MoodleBox angeschlossenen Sensoren visualisiert. Ein Backend soll die Sensoren kategorisieren, die Messdaten der Sensoren abfragen und diese Daten dem Frontend über eine API-Schnittstelle zur Verfügung stellen.

Da der Zugriff auf die Applikation vorwiegend mit Smartphones stattfinden wird, soll die SPA für mobile Geräte optimiert werden.

2.2 Benutzer Charakteristik

Als Zielgruppe gelten Dozierende und Studierende sowohl in europäischen wie auch in afrikanischen Hochschulen.

2.3 Einschränkungen

Authentication
- Im Zuge dieser Arbeit wird keine Authentifizierung von Benutzenden vorgenommen. Eine Weitergabe des Session-Tokens von Moodle an unsere Applikation ist nicht gefordert.

Sensoren
- Die Erkennung der Sensoren ist auf USB-Geräte beschränkt. Dies soll die Handhabung für den Benutzer erleichtern.

3. Use Cases

3.1 Use Case Diagramm

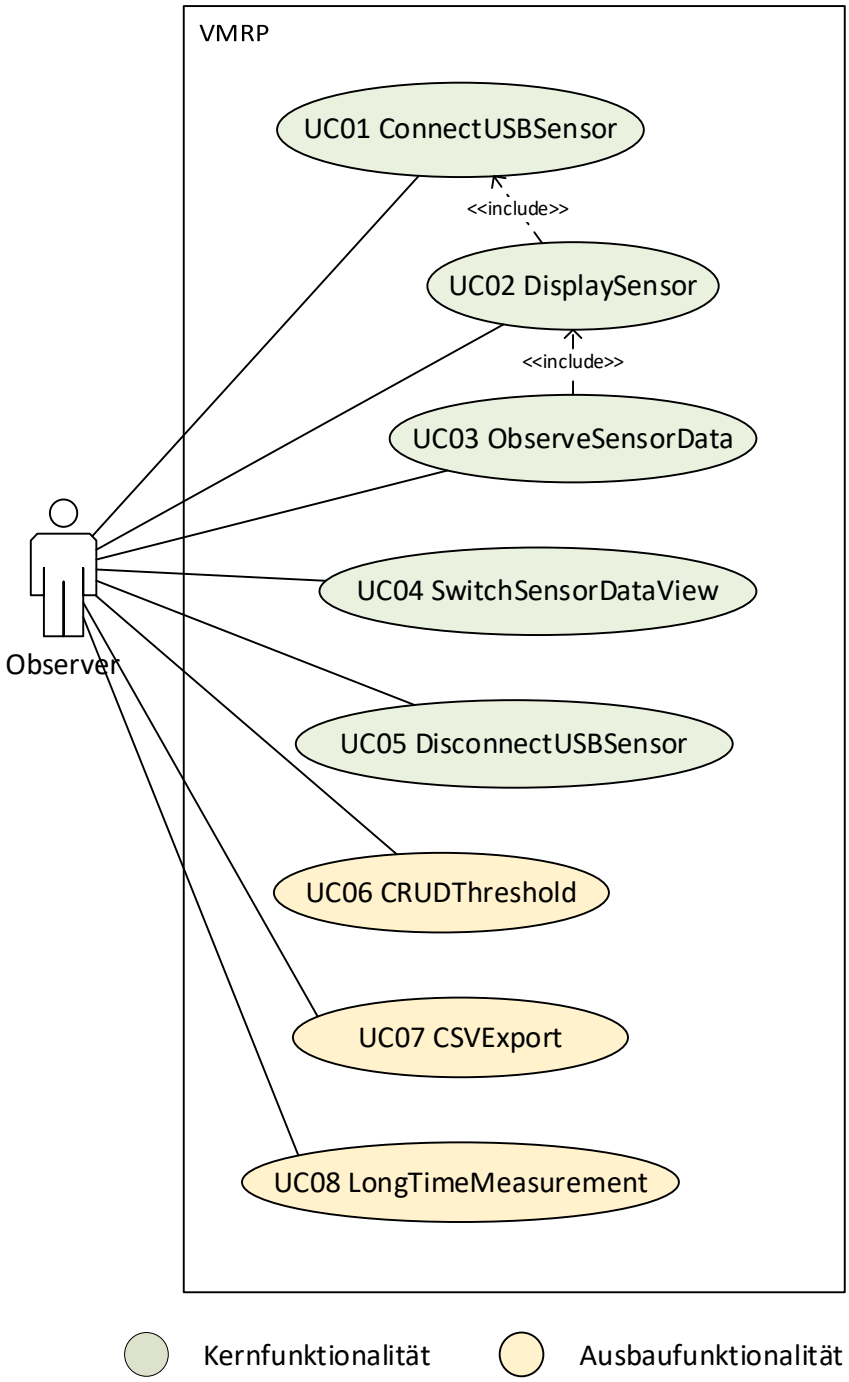

Abbildung 3: Use Case Diagramm

3.2 Aktoren

3.2.1 Observer

Der Observer startet den Raspberry Pi und verbindet per USB einen Sensor. Er führt damit Messungen durch.

Danach will er die Werte, welche mit einem angeschlossenen Sensor gemessen werden, graphisch angezeigt bekommen.

3.3 Beschreibungen

3.3.1 UC01 ConnectUSBSensor

User Story	Als Observer möchte ich ein USB-Sensor mit einem Raspberry Pi verbinden, damit gemessene Sensordaten verarbeitet werden können.
Main Success Scenario	1. Ein Observer möchte mit Hilfe eines Raspberry Pi's und einem USB-Sensor physikalische oder chemische Werte messen. 2. Dazu startet er den Raspberry Pi… 3. …und schliesst den Sensor an. 4. Das zu entwickelnde System erkennt den Sensor automatisch.
Alternate Scenarios	3a. Ein Observer verbindet ein dem Backend unbekanntes USB-Gerät. Das System zeigt dem Observer eine passende Meldung an.

3.3.2 UC02 DisplaySensor

User Story	Als Observer möchte ich auf dem Dashboard einen Sensor anzeigen können, um mir dessen Messdaten anzeigen zu lassen.
Main Success Scenario	1. Ein Observer möchte sich Sensordaten, welche durch den verbundenen USB-Sensor gemessen werden, grafisch anzeigen lassen. 2. Dazu öffnet er eine bestimmte Moodle-Seite auf dem Raspberry Pi oder er hat diese bereits offen. 3. Er selektiert den gewünschten Sensor. 4. Der ausgewählte Sensor wird auf dem Dashboard angezeigt. 5. Die Sensorauswahl eines Benutzers wird lokal gespeichert.
Alternate Scenarios	3a. Sollte noch kein Sensor am Raspberry Pi angeschlossen sein, erhält der Observer eine entsprechende Meldung. 3b. Der gesuchte Sensor wird bereits auf dem Dashboard angezeigt und muss nicht erneut ausgewählt werden.

3.3.3 UC03 ObserveSensorData

User Story	Als Observer möchte ich mir in Echtzeit Sensordaten anzeigen lassen, um gemessene Werte direkt betrachten zu können.
Main Success Scenario	1. Hat ein Observer einen Sensor ausgewählt, … 2. … werden ihm die in Echtzeit gemessenen Daten des Sensors angezeigt.
Alternate Scenarios	Trennt der User die Verbindung zur MoodleBox und verbindet sich einen Tag (24 Stunden) später erneut, sollten die aufgezeichneten Daten ersichtlich sein. Die Daten sollen auf dem Gerät gespeichert werden, solange dieses läuft.

3.3.4 UC04 SwitchSensorDataView

User Story	Als Observer möchte ich zwischen zwei Datenansichten wechseln können, um mir einen besseren Überblick über die gemessenen Werte zu ermöglichen.
Main Success Scenario	1. Ein Observer möchte für die Sensordatenansichten zwischen "Graph" und "Aktueller Wert" hin und her wechseln können. 2. Der Observer verbindet sich mit der Moodle-Seite. 3. Er schaltet die globale Datenansicht auf die gewünschte Einstellung. 4. Das Dashboard zeigt nun, je nach getätigter Auswahl, für alle Sensoren immer nur den aktuellen Messwert oder stellt die Messwerte als Graph dar.
Alternate Scenarios	3a. Wählt der Observer keine Ansicht aus, wird ihm standardmässig die Ansicht "Graph" angezeigt.

3.3.5 UC05 DisconnectUSBSensor

User Story	Als Observer möchte ich einen bereits verbundenen USB-Sensor vom Raspberry Pi trennen, damit eine Messung beendet wird.
Main Success Scenario	1. Ein Observer möchte eine Messung beenden und trennt den USB-Sensor vom Raspberry Pi. 2. Die zum USB-Sensor passenden Einträge im Frontend werden entfernt. Dazu zählen die Darstellung des Sensors und seine Messwerte, sowie gespeicherte Thresholds.
Alternate Scenarios	1a. Ein Observer trennt ein dem Backend unbekanntes USB-Gerät vom Raspberry Pi. Da durch das unbekannte Gerät im Frontend keine Sensoren angezeigt werden, muss das System nichts weiter unternehmen.

3.3.6 UC06 CRUDThreshold

User Story

Als Observer möchte ich für jeden Sensor einen oberen und unteren Alarm-Schwellwert (engl. Threshold) festlegen, bearbeiten und wieder entfernen können, um beim Über-/Unterschreiten dieser Werte schnell reagieren zu können

Main Success Scenario

1. Ein Observer verbindet sich mit der Frontend-Webseite, …
2. … öffnet die Ansicht eines Sensors …
3. … und klickt auf einen Button, um einen Threshold für diesen Sensor zu erstellen.
4. Er gibt den oberen und unteren Schwellwert ein.
5. Wird der Schwellwert über-/unterschritten, zeigt die Webseite ein Pop-Up Fenster an, welches über dieses Ereignis informiert.

Alternate Scenarios

4a. Gibt der Observer einen oberen Schwellwert ein, welcher kleiner als der untere Schwellwert ist (und umgekehrt), wird ihm ein Fehler angezeigt und der Alarm nicht erstellt.

3.3.7 UC07 CSVExport

User Story

Als Observer will ich die aktuellen Messdaten eines Sensors als CSV exportieren können, um mit den Daten in anderen Programmen weiterzuarbeiten.

Main Success Scenario

1. Ein Observer verbindet sich mit der Frontend-Webseite…
2. … und öffnet die Ansicht eines Sensors.
3. Dort sieht er die bisherigen Messdaten. Er klickt auf einen Export-Button.
4. Das System liefert ihm als Download eine CSV-Datei, welche die Daten dieses einen Sensors enthält.

Alternate Scenarios

2a. Sind am Raspberry Pi noch keine Sensoren angeschlossen, werden auf der Front end-Webseite keine Sensoren gelistet. Dem Observer sollen in diesem Fall keine Download-Buttons zur Verfügung stehen.

3.3.8 UC08 LongTimeMeasurement

User Story

Als Observer will ich Daten über einen längeren Zeitraum von einer Woche erheben können, damit ich neben den schulischen Messungen für andere Zwecke auch Langzeitmessungen durchführen kann.

Main Success Scenario

1. Ein Observer entscheidet sich, eine Langzeitmessung durchzuführen.
2. Er schaltet die Option dazu im Frontend ein.
3. Die Sensoren werden vom System automatisch mit einem höheren Intervall (z.B. minütlich) zur Messdatenerhebung konfiguriert.
4. Der Observer platziert die Sensoren und kehrt nach einer Woche zurück.
5. Er öffnet die Frontend-Webseite…
6. … und sieht die historischen Messdaten in der entsprechenden Granularität.

Alternate Scenario

3a. Wenn ein Sensor keine Möglichkeit zur Konfiguration des Messintervalls bietet, werden die Messdaten nur in der gewünschten Granularität gespeichert und die restlichen verworfen.

4. Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen werden im Zuge dieser Studienarbeit nach FURPS+ erhoben und nach dem von IBM vorgeschlagenen Schema [3] dokumentiert.

4.1 Usability

Requirement	Beschlüsse
Accessability	• Die Skala für die darzustellenden Messwerte muss anhand der Messwerte skaliert werden. D.h., die Y-Achse soll immer vom kleinsten Messwert bis zum grössten Messwert reichen.
Learnability	• Der User (ein/e Student/in einer Fachhochschule oder einer Universität) soll in der Lage sein, ohne Einführung in die Software einen Sensorwert anzeigen zu können.
Aesthetics	• Das UI soll im UMTEC Farb-Design gestaltet sein sowie das Logo des Instituts beinhalten (Farben von der Webseite https://umtec.ch).

Tabelle 5: Usability Anforderungen

4.2 Reliability

Requirement	Beschlüsse
Availability	• Die Software muss mit einem messenden Sensor 24 Stunden ohne Unterbrechung laufen können. Dies bedeutet, dass während dieser Laufzeit im System keine Fehler auftreten dürfen, welche verhindern, dass die Messdaten auf der Webseite dargestellt werden. Nach 24 Stunden muss ein Graph mit regelmässigen Messpunkten des Sensors dargestellt werden können.

Tabelle 6: Reliability Anforderungen

4.3 Performance

Requirement	Beschlüsse
Response Time	• Das Frontend soll beim Öffnen für den DOMContentLoaded Event im Chrome Browser (Desktop) nicht mehr als fünf Sekunden beanspruchen.
Start-up Time	• Sobald die Moodle-Webseite verfügbar ist, soll dies auch die Webanwendung sein (Messung von Hand: Moodle-Login durchführen und einen Kurs mit eingebundener Applikation öffnen).
Throughput	• Für einen beliebigen Sensor muss das System in der Lage sein, pro Sekunde einen Messwert vom Sensor auszulesen und im Frontend darzustellen, sofern der Sensor einen solch tiefes Intervall unterstützt.

Tabelle 7: Performance Anforderungen

4.4 Supportability

Requirement	Beschlüsse
Adaptability	• Die Softwarearchitektur muss so gestaltet sein, dass in der Zukunft neue Sensortreiber eingebunden werden können.
Compatibility	• Software muss auf Raspberry Pi 4 mit 8GB RAM funktionieren und dabei die restlichen nichtfunktionalen Anforderungen erfüllen.
Scalability	• Bei einem angeschlossenen Sensor müssen 20 User-Sessions gleichzeitig die Live-Daten dieses Sensors ansehen können. Als gut gilt, wenn die Daten ohne merkliche Verzögerung im Verhältnis zu einem einzelnen verbundenen User angezeigt werden. Akzeptabel ist eine Verlangsamung um 2 Sekunden.
Localizability	• Die Sprache des Frontend der Software muss Englisch sein. Die Software könnte auch in französischsprechenden Ländern zum Einsatz kommen. Bei der Entwicklung soll darauf geachtet werden, dass eine Nachimplementierung von Sprachpakten nicht verhindert wird.

Tabelle 8: Supportability Anforderungen

4.5 Design Requirements

Requirement	Beschlüsse
Persistence	• Messwerte sollen bis zum Herunterfahren des Raspberry Pi gespeichert werden.
Third Party Components	• Der Betrieb der zu entwickelnden Software soll gleichzeitig neben einer MoodleBox Instanz auf einem Raspberry Pi 4 möglich sein.
Implementation Languages	• Frontend: Vue.js mit TypeScript • Backend: Node.js, Node Express mit TypeScript
Plattform Support	• Betriebssystem: Anwendung muss auf RaspbianOS (MoodleBox) oder alternativ Debian mit Moodle und MoodleBox Plugin funktionieren. • Das Frontend wird vorwiegend auf Android Smartphones (Google Chrome) genutzt. • Es werden nur USB-Sensoren unterstützt.
Resource Limits	• Raspberry Pi 4: 8GB RAM. Dabei gilt zu beachten, dass eine In-Memory Datenbank ebenfalls das RAM auch beansprucht.

External Systems	• Es soll versucht werden folgende Sensoren zu unterstützen:
	  • Pico Technologies DrDAQ (Temperature / Light / Sound Sensor built in, PH Sensor, Redox Sensor).
	  • Tinovi USB-Sensor (Soil Moisture / Temperature Sensor).
	  • Temperature & Humidity Sensor TEMPer1F_H1 (Hygrometer / Temperature / Humidity Sensor.

Tabelle 9: Software Design Anforderungen

5. Tabellenverzeichnis

6. Abbildungsverzeichnis

7. Literaturverzeichnis

[1] swissuniversities, «COFER-WASH: Consortium for Education and Research in Water, Sanitation and Hygiene,» [Online]. Available: https://www.swissuniversities.ch/themen/entwicklung-und-zusammenarbeit/p-6-swissuniversities-development-and-cooperation-network-sudac/cofer-wash. [Zugriff am 17 September 2020].

[2] M. Rohr, Interviewee, *Sitzung zur Anforderungsanalyse.* [Interview]. 07 September 2020.

[3] P. Eeles, „Sample Architectural Requirements Questionnaire," IBM, 04 April 2004. [Online]. Available: https://www.ibm.com/developerworks/rational/library/4710.html. [Zugriff am 16 September 2020].

3.3. Softwarearchitektur

SA - VMRP

Autoren: Marco Gartmann, Fabian Thurnheer
Version: 1.0

Erstellt am: 01.10.2020
Letzte Änderung am: 18.12.2020

Inhaltsverzeichnis

1. Einführung

1.1 Zweck

In diesem Dokument wird die Architektur der einzelnen Applikationsteile, die physikalische Verteilung dieser und alle getroffenen Architekturentscheide dokumentiert. Es soll Personen, welche sich in Zukunft mit der Weiterentwicklung der Applikation befassen, das Verstehen der Anwendung vereinfachen.

1.2 Gültigkeitsbereich

Die Gültigkeit des Projektplans beschränkt sich auf die Laufzeit des Moduls "Studienarbeit" im Herbstsemester 2020.

2. Systemübersicht

Die Applikation VMRP, welche im Zuge der Studienarbeit entwickelt wird, integriert sich wie folgt in die umliegenden Systeme:

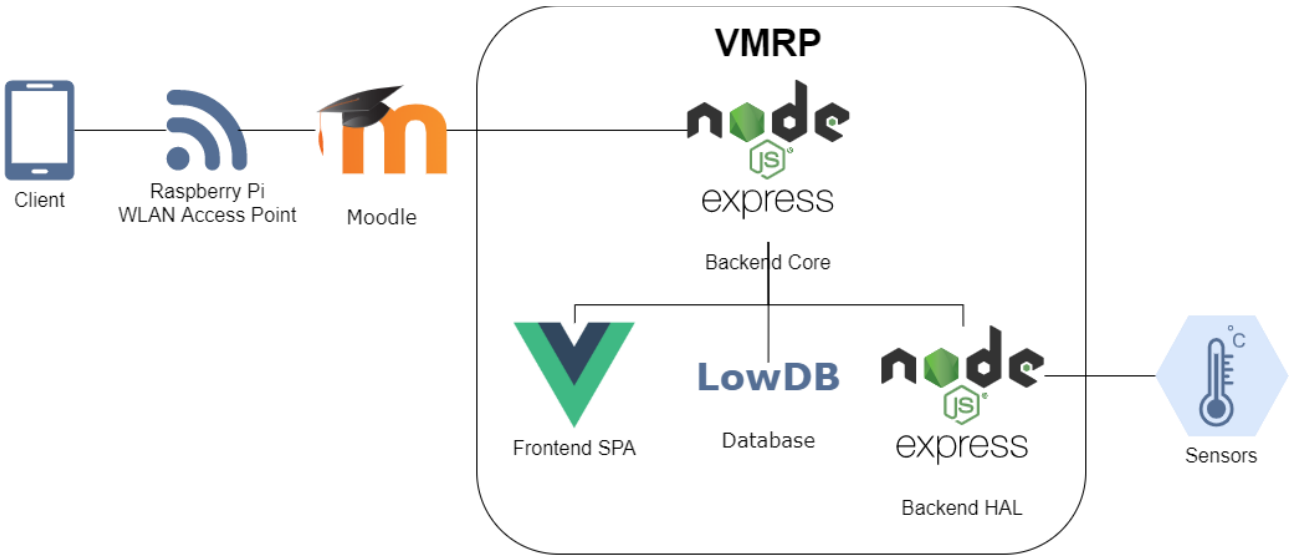

Abbildung 4: Systemübersicht Diagramm

Der mit VMRP gekennzeichnete Rahmen bezeichnet den Einflussbereich dieser Studienarbeit. Alle Komponenten ausserhalb dieses Bereichs werden als externe Systeme betrachtet.

Die gesamte VMRP Anwendung besteht aus den drei folgenden Komponenten, dargestellt als CRC Karten:

Frontend	
Zuständigkeit - Liest Sensordaten aus dem Core aus. Stellt Sensordaten grafisch dar.	**Benötigt**: Core

Tabelle 10: CRC Frontend

Core	
Zuständigkeit - Liest Sensordaten aus dem Hardware Abstraction Layer (HAL) aus. - Persistiert Sensordaten. Stellt über eine Web API die Sensordaten zur Verfügung.	**Benötigt**: HAL **Wird benötigt von:** Frontend

Tabelle 11: CRC Backend Core

HAL	
Zuständigkeit - Stellt Verbindung mit USB-Geräten her. - Verarbeitet Sensordaten. Stellt über eine Web API die Sensordaten zur Verfügung.	**Wird benötigt von**: Core

Tabelle 12: CRC Backend HAL

3. Architekturentscheide

3.1 Aufteilung des Backend

Das Backend wurde in eine Core Komponente und in eine HAL Komponente unterteilt. Die HAL Komponente übernimmt dabei die Kommunikation mit den Hardware Sensoren.

Die beiden Komponenten werden über ein API miteinander verbunden, über welches die Sensordaten und die Messwerte ausgetauscht werden.

Dieser Ansatz wurde gewählt, damit der hardwareabhängige Teil der Applikation vom Rest des Backend abgetrennt werden kann. Als Folge müssen sich die HAL und die Komponente des Cores nicht auf demselben physikalischen Gerät befinden. Weiter können mit einer Core Instanz theoretisch mehrere verteilte HALs angesprochen werden.

Dieser Entscheid hat ebenfalls Auswirkungen auf die nichtfunktionale Anforderung "Adaptability", wonach die Software Architektur ein einfaches Erweitern von neuen Sensortreibern ermöglichen soll. Durch die Aufteilung entstehen kleinere Software Container ("high-level technical building blocks", nach C4 [1]), wodurch sich künftige Programmierer für die Erweiterung eines Containers – z.B. die Implementierung neuer Sensortreibern im HAL – in eine kleinere Codebasis einarbeiten müssen. Aus eigener Erfahrung vereinfacht diese Massnahme die Weiterentwicklung der Software aus Programmierersicht und trägt zur Adaptierbarkeit des Systems bei.

3.2 Mobile First Frontend

Für die Entwicklung der Single Page Applikation zur Darstellung der Messdaten wurde ein Mobile First Ansatz gewählt. Dies bedeutet, dass die Webseite für die Bedienung auf mobilen Geräten wie Smartphones optimiert sein soll. Dieser gewählte Ansatz lässt sich damit begründen, dass die Anwendung wie in der Aufgabenstellung geschildert in Hochschulen auf dem Afrikanischen Kontinent zum Einsatz kommen kann, in welchen der Grossteil der Studierenden über ein Smartphone, aber nicht über einen eigenen Laptop verfügt. [2]

Der Mobile First Ansatz garantiert, dass von Beginn der Entwicklung an auf die Bedienbarkeit der Webseiten-Elemente auf einem Smartphone geachtet wird. Somit wird ein Teil der nichtfunktionalen Anforderung "Plattform Support" abgedeckt.

3.3 Gestaltung nach Material Design

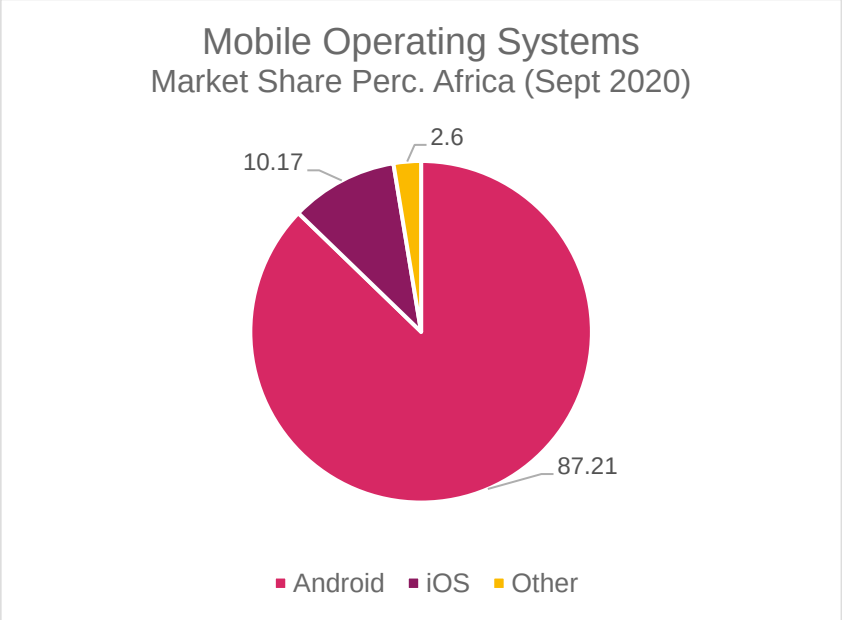

Abbildung 5: Marktanteil mobiler Betriebssysteme in Afrikanischen Ländern [3]

Als Teil der Usability Requirements der nichtfunktionalen Anforderungen wurde der Punkt "Learnability" genannt. Demnach soll ein User der genannten Benutzergruppe in der Lage sein, die Software ohne Einführung bedienen zu können. Um diese Anforderung abzudecken, soll das User Interface so gestaltet werden, dass ein User bereits mit dem Aussehen dieses vertraut ist und sich darin direkt zurechtfindet. Dieses Kapitel dient dazu, die Evaluation der eingesetzten Design Sprache zu beschreiben.

Material Design ist ein von Google entwickeltes Design Framework, welches u.a. im Android Betriebssystem und in den Applikationen von Google angewendet wird und dort für ein einheitliches Aussehen sorgt. Alternativen dazu wären etwa das von Alibaba entwickelte AntDesign oder Grommet. Dazu gibt es weitere Design Sprachen, welche auf Material Design aufbauen. [4]

Wie die Abbildung 5 zeigt, verwendet der Grossteil der Afrikanischen Smartphone-Benutzer ein Gerät mit dem Android Betriebssystem [3]. Für die Gestaltung der Android Benutzeroberfläche und Android Apps hält sich Google an die Empfehlungen aus der Material Design Sprache. Damit das Aussehen der zu entwickelnden Webseite dem Grossteil der Benutzer möglichst vertraut vorkommt, halten wir uns bei der Entwicklung ebenfalls an die Empfehlungen des Material Designs.

Um die Umsetzung der Gestaltung nach den Material Design Prinzipien zu unterstützen, wird im Frontend die Library Vuetify eingesetzt.

3.4 Einsatz einer Datenbank

Um beim Use Case "UC03 ObserveSensorData" das Szenario zu unterstützen, in welchem ein Benutzer die Messwerte eines Sensors im Nachhinein auslesen möchte, müssen die erhobenen Messwerte zwischengespeichert werden. Wie in den nichtfunktionalen Anforderungen erhoben wurde, müssen diese Daten bis zum Herunterfahren des Host-Geräts und nicht darüber hinaus persistiert werden.

Für die Zwischenspeicherung dieser Daten bieten sich folgende zwei Varianten an:

- In-Memory, im Code: als Teil einer Datenstruktur, welche die Programmiersprache anbieten, z.B. in einer Map (Schlüssel-Werte Paare).
- in einer Datenbank: in Memory oder auf einem Speichermedium persistiert.

Variante	Pro	Contra
Code Datenstruktur (Map)	• Einfache Implementation. • keine zusätzlichen Komponenten.	• Bietet keine Möglichkeit, Daten in Zukunft über einen Neustart des Geräts zu persistieren. • Für unterschiedliche Entities (z.B. Sensordaten, Geräteinformationen) müssen mehrere Instanzen einer Map angelegt werden. • Verfügbares Memory auf den Raspberry Pi's ist beschränkt.
Datenbank	• Kann verschiedene Daten In-Memory und auf Disk persistiert speichern. • Daten sind in verschiedenen Formaten speicherbar, z.B. auch JSON. • Eine Datenbank bildet von sich aus eine einheitliche Datenschnittstelle (Single Source of Truth).	• Zusätzliche Abhängigkeit.

Tabelle 13: Evaluation Speichermöglichkeiten

Anhand der in Tabelle 13 evaluierten Vor- und Nachteile fiel die Entscheidung auf den Einsatz einer Datenbanklösung. Obwohl mit einer Datenstruktur wie einer Map die Grundanforderungen erfüllt werden könnten, ist es damit nicht möglich, Daten zu persistieren. Dies wäre für den Use Case 03 zwar nicht erforderlich, aber in Zukunft möglicherweise erwünscht.

Auch in Bezug auf die Ausbaufunktionalität "UC08 LongTimeMeasure" bietet sich der Einsatz einer Datenbank an. Sollte während einer Langzeitmessung die Stromzufuhr zum Raspberry Pi unterbrochen werden, könnte mit Hilfe einer Datenbank bereits verarbeitete Messwerte wieder verfügbar gemacht werden. Bei der Verwendung einer In-Memory Datenbank besteht zudem die Gefahr, dass das System durch die grössere Datenmenge einer Langzeitmessung stark verlangsamt und unbenutzbar wird.

3.4.1 Evaluation der Datenbanklösung

Nach dem Entscheid zum Einsatz einer Datenbank geht es in diesem Kapitel um die Evaluation der passendsten Datenbanklösung.

Bei der Evaluation wurden die folgenden Punkte berücksichtigt:

- Möglichst einfache Integration in die Software
- Es existieren keine komplexen Beziehungen zwischen Messdaten und den Sensorinformationen.
- Einfachheit der API
- Qualität und Umfang der Dokumentation
- Abdeckung der benötigten Funktionen

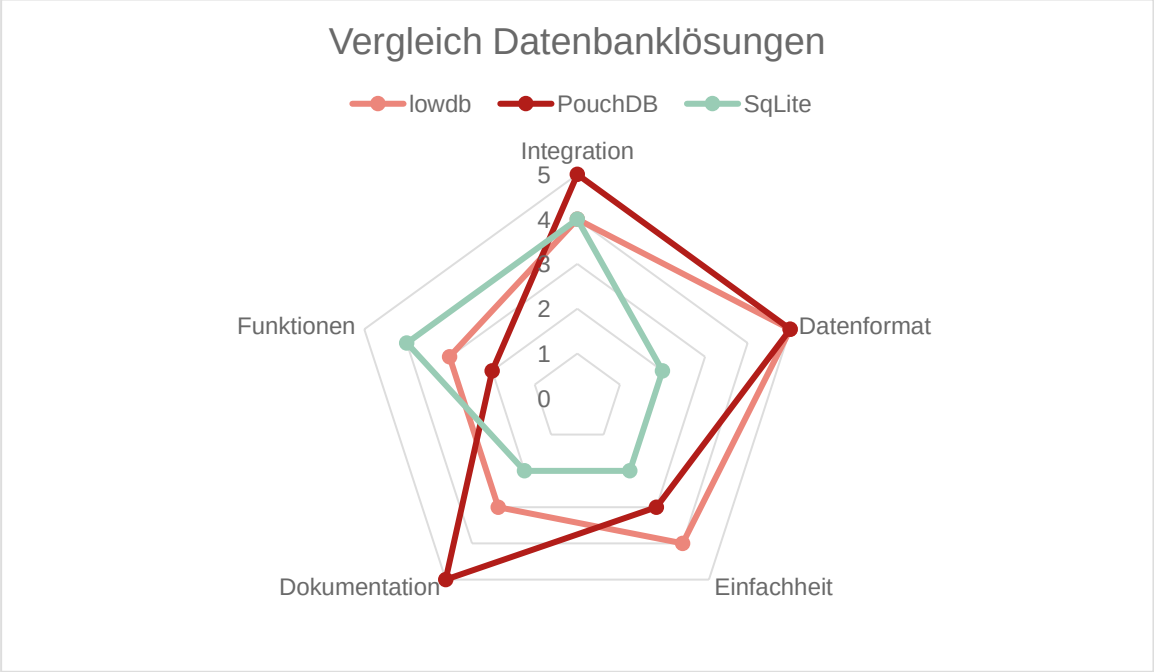

Abbildung 6: Vergleich Datenbanklösungen

Anhand einer Recherche über verbreitet eingesetzte Lösungen ergaben sich die folgenden drei Datenbank Produkte:

lowdb JSON DB, welche aus einem einzigen db.json File besteht.

- Integration:
  - \+ Installation via npm Paket Manager, einfaches require im Code.
  - \+ Durch Mitgabe eines Schema-Typs erhält man gute TypeScript Unterstützung für die DB Entitäten.
- Datenformat:
  - \+ Nicht-relational
  - \+ Speichert Daten direkt im für den Menschen leserlichen JSON Format, welches schlussendlich auch für die Auslieferung der Daten über die APIs verwendet wird.
  - \+ Dadurch ist ein einfaches Debugging der Datenbank möglich.
- Einfachheit:
  - \+ Verwendet einfache und bekannte Syntax.
  - \+ Ist nicht mit Funktionen überladen und damit einfach zu erlernen.
  - \+ Setup und Erstellung der DB ist in vier Zeilen Code möglich.
- Dokumentation:
  - \+ Grundlegende Funktionen sind mit vielen Code Beispielen dokumentiert.
  - – Dokumentation ausreichend, bei vertieften Fragen müsste aber auf Community zurückgegriffen werden.
- Funktionen:
  - \+ Bietet die zwingend benötigten Funktionen an.
  - – Erweitertes Filtern der Daten ist nicht direkt bei der DB-Abfrage möglich. Man muss sich ein grosses Datenset liefern lassen und dann im Programm selbst filtern.

PouchDB

Dokumentenbasierte DB, aufgebaut auf Apache CouchDB.

- Integration:
  - \+ Installation via npm Paket Manager, einfaches require im Code.
  - \+ TypeScript Types werden vom Hersteller mitgeliefert.
- Datenformat:
  - \+ Nicht-relational
  - \+ Speichert Dokumente ebenfalls im JSON Format, wodurch die bei lowdb bereits genannten Vorteile auch hier gelten.
- Einfachheit:
  - \+ Grundlegende Funktionen sind einfach anzuwenden.
  - – Durch die grosse Anzahl an angebotenen Funktionen braucht es aber wohl eine längere Einarbeitungszeit als diese bei lowdb der Fall ist.
- Dokumentation:
  - \+ Die Produktwebseite enthält eine sehr umfangreiche Dokumentation.
  - \+ Alle API-Funktionen sind mit Code Beispielen versehen.
  - \+ Doku gibt Tipps zur korrekten Verwendung.
- Funktionen:
  - – Bietet mehr Funktionen an, als in diesem Projekt benötigt werden.
  - – Das wichtigste Feature zur Synchronisation von mehreren DBs wird in diesem Projekt nicht benötigt.

SqLite

Serverlose SQL-Datenbank-Engine ohne fixes Schema,

- Integration:
  - \+ Einbindung über Node Modul sqlite3 möglich.
  - \+ Typen für TypeScript können mit npm installiert werden.
- Datenformat:
  - – Bei SqLite handelt es sich um eine relationale Datenbank
  - – Die Daten in diesem Projekt verfügen über keine Komplexen Beziehungen, die relationale Abbildung der Daten wird nicht benötigt und führt zu unnötiger Komplexität.
  - – Durch die relationale Natur der Datenbank müsste ein OR-Mapper eingesetzt werden.
- Einfachheit:
  - \+ Als Abfragesprache wird SQL verwendet, welche aus dem Studium bekannt ist. Dadurch können komplexe Abfragen getätigt werden.
  - – Die Ausführung von Abfragen auf der Datenbank scheint auf den ersten Blick komplexer und ist weniger intuitiv, als es bei den anderen beiden Lösungen der Fall ist.
- Dokumentation:
  - \+ Dokumentation ist auf GitHub vorhanden.
  - – Viele API-Funktionen sind ohne Code Beispiele dokumentiert, was das erlernen erschwert.
- Funktionen:
  - \+ Im Vergleich zu lowdb können bei dieser Lösung Abfragen mit komplexen Filter Anweisungen direkt in der DB durchgeführt werden, welche für solche Operationen optimiert ist.

Als Lösung, welche unsere Bedürfnisse am besten abdeckt, wurde die Datenbank lowdb evaluiert. Für diese Datenbank spricht, dass…

- … eine gepflegte Dokumentation der Funktionalitäten vorhanden ist.
- … ein Schema definiert werden kann, was Typensicherheit im Programm garantiert.
- … die Daten im JSON Format gespeichert werden, was ein einfaches Debugging erlaubt und die Verwendung eines OR-Mappers ausschliesst, was die Integration weniger aufwändig macht.
- … es sich dabei um die Lösung handelt, welche aus unserer Sicht am einfachsten in der Anwendung und am schnellsten zu erlernen ist.

3.5 Trennung HTTP und Websocket API

In der gewählten Architektur werden die Echtzeit-Messdaten und die Benachrichtigungen über das Anschliessen oder Trennen eines Geräts über ein Websocket API übermittelt. Die statischen Daten, wie z.B. eine Liste mit allen angeschlossenen Geräten oder die historischen Daten, werden über ein HTTP API abgefragt und versendet.

Alternativ hätte auf das HTTP API verzichtet und sämtliche Daten über ein Websocket API versendet werden können.

Folgende Punkte führten zum Entscheid, die statischen Daten über ein separates HTTP API zu übermitteln:

- Gelegentliche Kommunikation: da das Frontend die Daten nur sporadisch abfragt und diese nicht live sein müssen, ist ein HTTP API besser geeignet.
- Eine bidirektionale Kommunikation wird nicht benötigt, da das Frontend die historischen Daten nur anfragen muss, wenn die Seite neu geladen wird.
- Durch die Trennung der beiden APIs wird die Websocket API-Logik übersichtlich gehalten. Somit werden nur Nachrichten über den Websocket übertragen, welche eventbasiert sind.

4. Logische Architektur Frontend

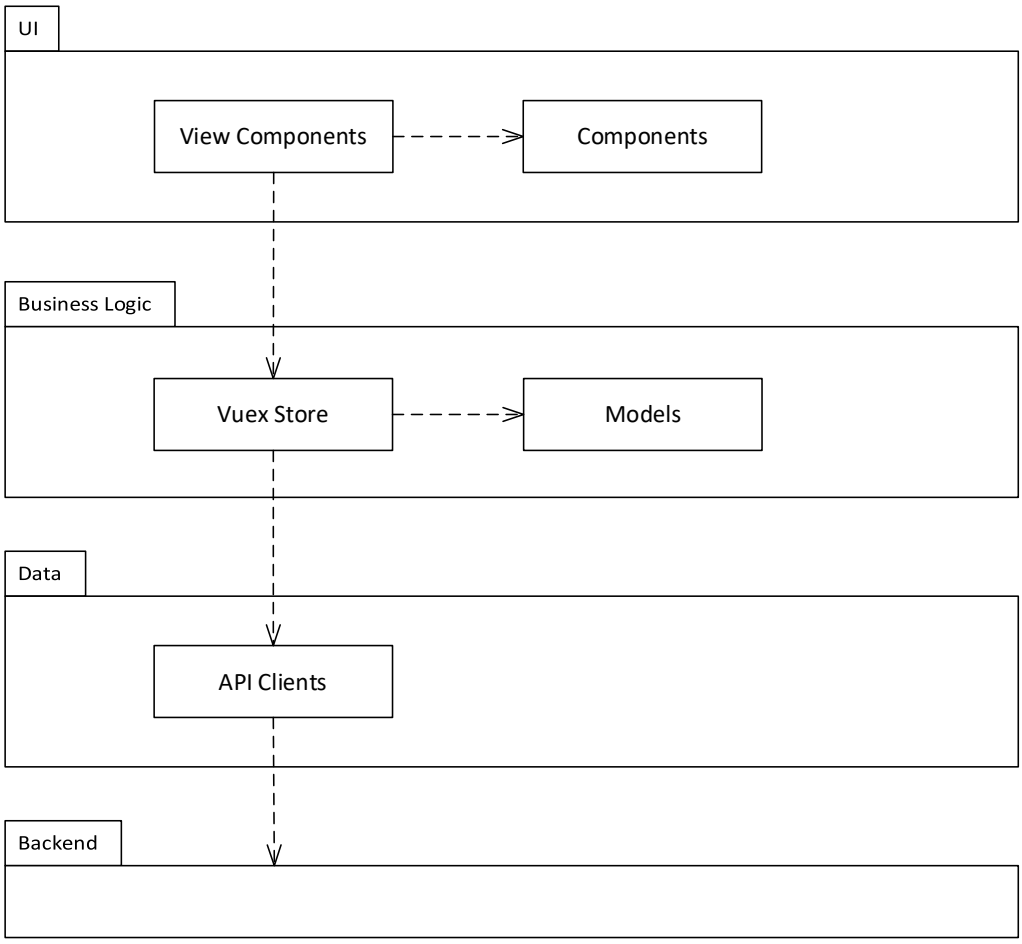

Abbildung 7: Frontend Layer-Diagramm

4.1 UI

Im UI Layer befinden sich die Vue.js Components. Sie definieren die grafische Benutzeroberfläche der Applikation.

4.1.1 Komponentenstruktur

Komponente	Beschreibung
View Components	Bilden das Grundgerüst des UIs. Sie können View Components oder Components beinhalten.
Components	Sind einzelne austauschbare Bausteine des UIs. Meist sind sie von anderen Components unabhängig oder benötigen nur Properties, welche beim Aufruf des Components mitgegeben werden können.

Tabelle 14: Komponentenübersicht UI Frontend

4.1.2 Schnittstellen

- Fordert Daten über die Getter-Funktionen der einzelnen Vuex Store Module an.
- Initialisiert über Action-Funktionen des Vuex Stores das Laden von Daten. (z.B. Messdaten, Geräteinformationen).

4.1.3 Wichtige interne Abläufe

- Nimmt Daten aus dem Vuex Store entgegen und stellt diese dar.

4.2 Business Logic

Die Business Logic im Frontend bestehet aus dem Vuex Store. Darin befindet sich der State der Applikation. Sie stellt Funktionen zur Verfügung, um den State zu lesen und zu verändern.

4.2.1 Komponentenstruktur

Komponente	Beschreibung
Vuex Store	Besteht aus einzelnen Modulen und bildet mit ihnen den State der Applikation.
Models	Models des Frontend

Tabelle 15: Komponentenübersicht Business Logic Frontend

4.2.2 Schnittstellen

- Bietet der UI Schicht Getter und Action Funktionen an.
- Nutzt Services der Data Schicht, um Daten aus dem Backend zu laden.

4.2.3 Wichtige interne Abläufe

- Der Message Store beliefert die anderen Store Komponenten mit Websocket Messages.

4.3 Data

Die Data-Schicht bietet dem Business Logic Layer Services zum Laden von Daten an.

4.3.1 Komponentenstruktur

Komponente	Beschreibung
API-Clients	Services, welche es ermöglichen, über Websocket oder HTTP mit dem Backend zu kommunizieren.

Tabelle 16: Komponentenübersicht Data Frontend

4.3.2 Schnittstellen

- Kommuniziert mit dem Backend über Websocket und HTTP.

5. Logische Architektur Core

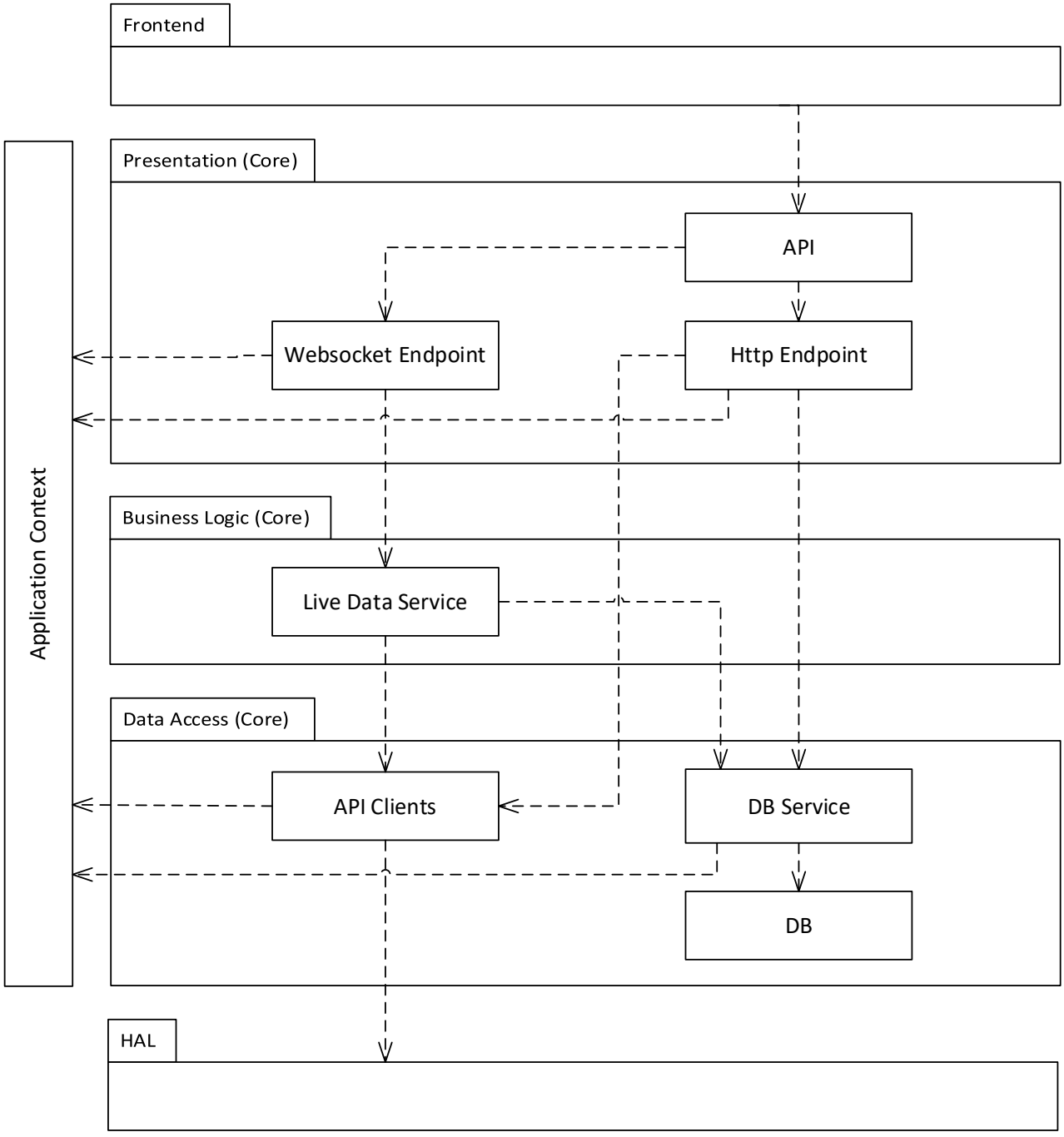

Abbildung 8: Layer-Diagramm Core

5.1 Presentation

Die Klassen im Presentation Layer haben die Aufgabe, die vom Sensor gemessenen Daten sowie die Informationen über die Sensoren der Aussenwelt über ein API zur Verfügung zu stellen. Sie entscheiden, ob einem Client die aktuellen Live-Daten der Sensoren oder die seit dem Anschliessen eines Sensors aufgezeichneten Messungen geliefert werden.

5.1.1 Komponentenstruktur

Komponente	Beschreibung
API	Stellt dem Frontend eine HTTP und eine Websocket Schnittstelle zur Verfügung. Einstiegspunkt der Applikation. Wird aufgerufen, sobald das Backend gestartet wird. Öffnet die TCP Ports, welche für die APIs benötigt werden.
Websocket Endpoint	Aktiviert eine Websocket Instanz und registriert die für die Kommunikation nötigen Handlers.
HTTP Endpoint	Stellt die Express.js Applikation bereit. Enthält einen HTTP Controller, welcher die Abfragen abarbeitet und einen HTTP Router, der die Anfragen einem Controller zuweist.

Tabelle 17: Komponentenübersicht Presentation Core

5.1.2 Schnittstellen

Der Presentation Layer bietet dem Frontend zwei Schnittstellen an:

- API: über diese Schnittstelle kann ein Frontend abfragen, welche Sensoren momentan angeschlossen sind. Es können zusätzlich Informationen über den Typ des Sensors, den Hersteller usw. mitgeliefert werden.
- Websocket API: für die Auslieferung der Live-Messdaten an den Client. Sobald ein Sensor eine neue Messung vermeldet, wird diese ohne Anfrage des Clients an diesen weitergeleitet.

5.1.3 Wichtige interne Abläufe

- Die Funktionen in der API-Klasse werden als erstes ausgeführt, wenn die Backend Applikation gestartet wird.
- Frontend HTTP-Anfragen werden von der API-Klasse entgegengenommen, von der Router Klasse geroutet und von den Controllern bearbeitet.

5.2 Business Logic

Der Business Layer übernimmt die Verantwortung, die Datenquellen im Auftrag des Presentation Layers abzufragen und die Daten zurückzuliefern.

5.2.1 Komponentenstruktur

Komponente	Beschreibung
Live Data Service	Wartet auf neue Messdaten vom HAL API, leitet Speicherung dieser Daten ein und leitet sie zum Core API weiter.

Tabelle 18: Komponentenübersicht Business Logic Core

5.2.2 Schnittstellen

Die Komponenten stehen mit dem darunterliegenden Data Access Layer im Kontakt, um Daten zu erhalten und zu speichern.

5.2.3 Wichtige interne Abläufe

Erhält der Live Data Service neue Live-Daten, werden diese dem Data Access Layer zur Speicherung in die DB übergeben.

5.3 Data Access

Der Data Access Layer ist für die Kommunikation mit der eingesetzten Datenbank und das Erhalten der Live-Daten des HALs zuständig.

5.3.1 Komponentenstruktur

Komponente	Beschreibung
DB Service	Komponente, welche das CRUD der Daten in der Datenbank übernimmt.
API Clients	Agieren als Clients des HAL APIs, über welches dieser Service die Live-Daten und Informationen über die Sensoren erhält.

Tabelle 19: Komponentenübersicht Data Access Core

5.3.2 Schnittstellen

- Der DB Service greift auf die Methoden zu, welche ihm das API der eingesetzten Datenbank bietet.
- Greift auf die vom HAL angebotenen APIs zu.

6. Logische Architektur HAL

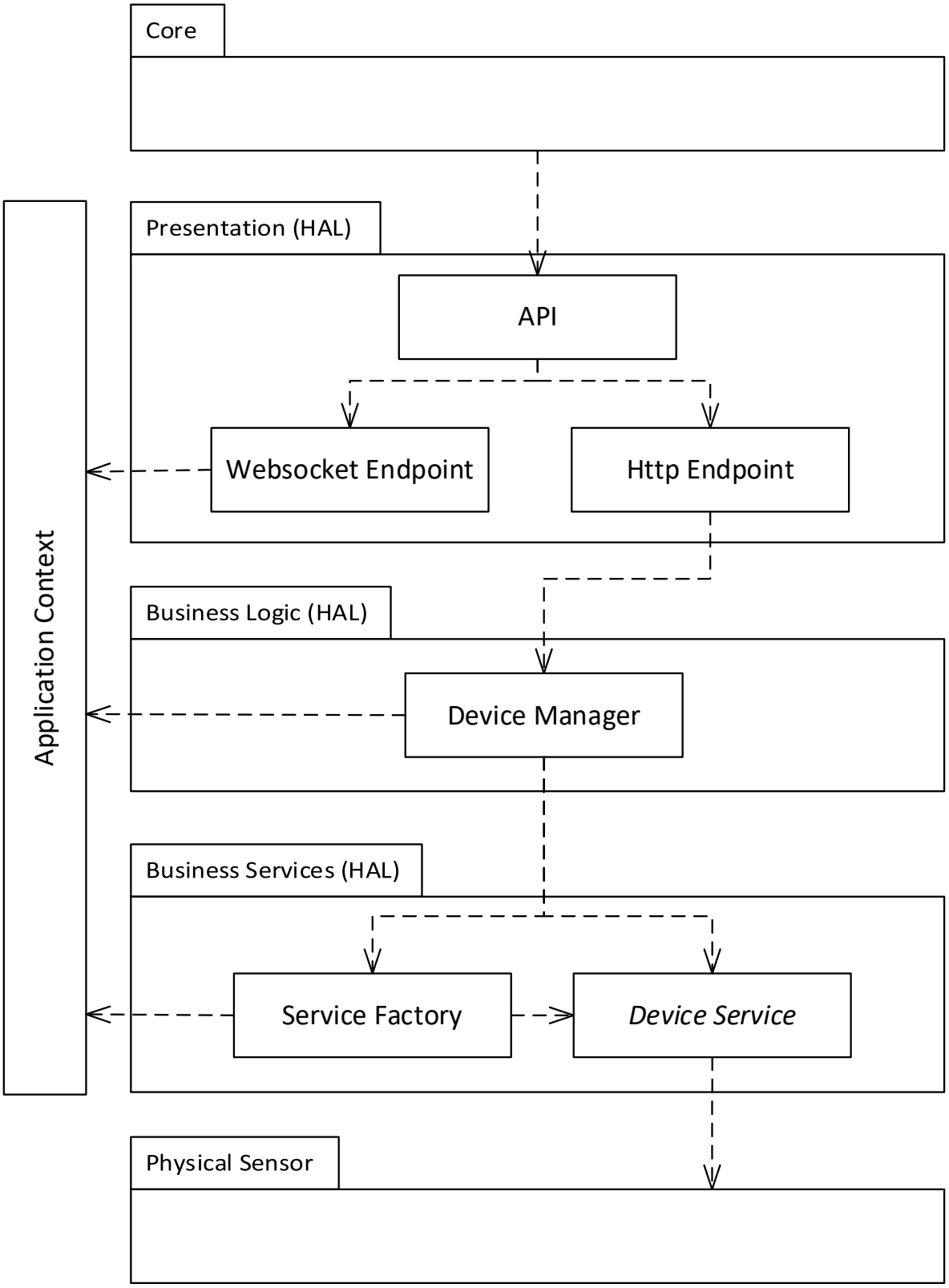

Abbildung 9: Layer-Diagramm HAL

Der Hardware Abstraction Layer (HAL) dient zur Abstraktion aller hardwareabhängigen Komponenten des Systems. Der Core soll damit komplett unabhängig von den eingesetzten Sensoren betrieben werden können. Zudem soll es der HAL ermöglichen, im Nachhinein neue Sensortypen hinzuzufügen, ohne Änderungen am Code in darüberliegenden Schichten anpassen zu müssen. Diese Schicht wird so implementiert, dass man sie auf einen vom Backend separierten Tier laufen lassen könnte.

Die wichtigsten internen Abläufe des HALs sind im Kapitel "7. Wichtige Abläufe" dokumentiert.

6.1 Presentation

Im Presentation Layer werden die Messdaten dem Core über eine API zur Verfügung gestellt.

6.1.1 Komponentenstruktur

Komponente	Beschreibung
API	Stellt dem Core eine HTTP und eine Websocket Schnittstelle zur Verfügung. Einstiegspunkt der Applikation, wird aufgerufen, sobald das Backend gestartet wird. Öffnet die TCP Ports, welche für die APIs benötigt werden.
Websocket Endpoint	Aktiviert eine Websocket Instanz und registriert die für die Kommunikation nötigen Handlers.
HTTP Endpoint	Stellt die Express.js Applikation bereit. Enthält einen HTTP Controller, welcher die Abfragen abarbeitet und einen HTTP Router, der die Anfragen einem Controller zuweist.

Tabelle 20: Komponentenübersicht Präsentation HAL

6.1.2 Schnittstellen

- Dem Core wird eine API angeboten, über welche die Messdaten und Informationen über Sensoren abgefragt werden können.
- Bezieht die Messdaten und Informationen über Sensoren vom DeviceManager aus der Business Logic.

6.2 Business Logic

Die Business Logic bildet den Kern des HALs. Er verwaltet die angeschlossenen USB-Geräte und startet Services, um diese Geräte anzusteuern.

6.2.1 Komponentenstruktur

Komponente	Beschreibung
Device Manager	Ist zuständig für das Erkennen von USB-Geräten und das Starten von Device Services, um diese Geräte anzusteuern. Verwaltet die Informationen darüber, welche Sensoren aktuell angeschlossen sind.

Tabelle 21: Komponentenübersicht Business Logic HAL

6.2.2 Schnittstellen

- Wird von der Presentation Schicht nach Messdaten und Sensorinformationen gefragt.
- Bezieht die Messdaten und Informationen über Sensoren von gestarteten Device Services aus dem Business Services Layer.

6.3 Business Services

Der Business Services Layer beinhaltet die Device Services, welche verwendet werden, um USB-Geräte anzusteuern und deren Sensoren auszulesen.

6.3.1 Komponentenstruktur

Komponente	Beschreibung
Device Service	Pro unterstütztem USB-Gerät existiert ein Device Service. Er steuert das jeweilige Gerät an und liest die Messdaten sowie Geräteinformationen aus. Wird gestartet, sobald das passende Gerät angeschlossen wird.
Service Factory	Retourniert anhand des Namens eines Device Services die dazu passende Klasseninstanz.

Tabelle 22: Komponentenübersicht Präsentation HAL

6.3.2 Schnittstellen

- Bezieht Messdaten und Informationen über Sensoren von USB-Geräten.

7. Wichtige Abläufe

7.1 Systemsequenzdiagramm "Anschliessen eines Sensors im HAL"

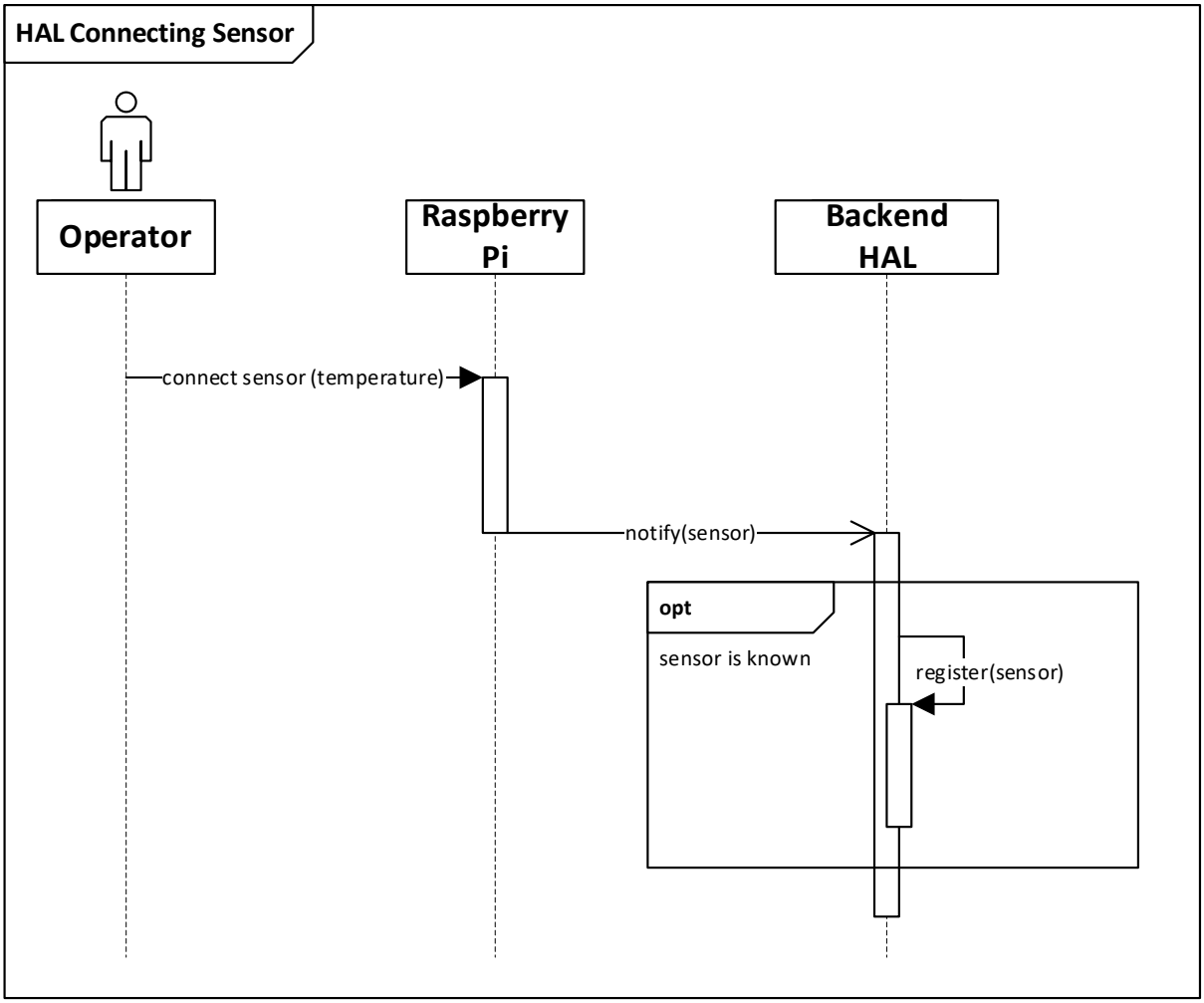

Abbildung 10: Sequenzdiagramm zum Anschliessen eines Sensors im HAL des Backend

Das oben dargestellte Diagramm abstrahiert und beschreibt das Zusammenspiel der Klassen im Hardware-abstraktionslayer (HAL) des Backend beim Anschliessen eines physikalischen Sensor-Geräts (im Diagramm 'Device' genannt) an den Raspberry Pi Minicomputer.

7.2 Systemsequenzdiagramm "Verarbeitung einer Messung im HAL"

Folgendes Diagramm dient zur Erklärung der Abfolge der Bearbeitung eines Messwerts, welcher von einem physikalischen Sensor stammt.

Vorbedingung dafür ist, dass zuvor ein Sensor so angeschlossen wurde, wie es im Diagramm "HAL Connecting Sensor" dargestellt ist.

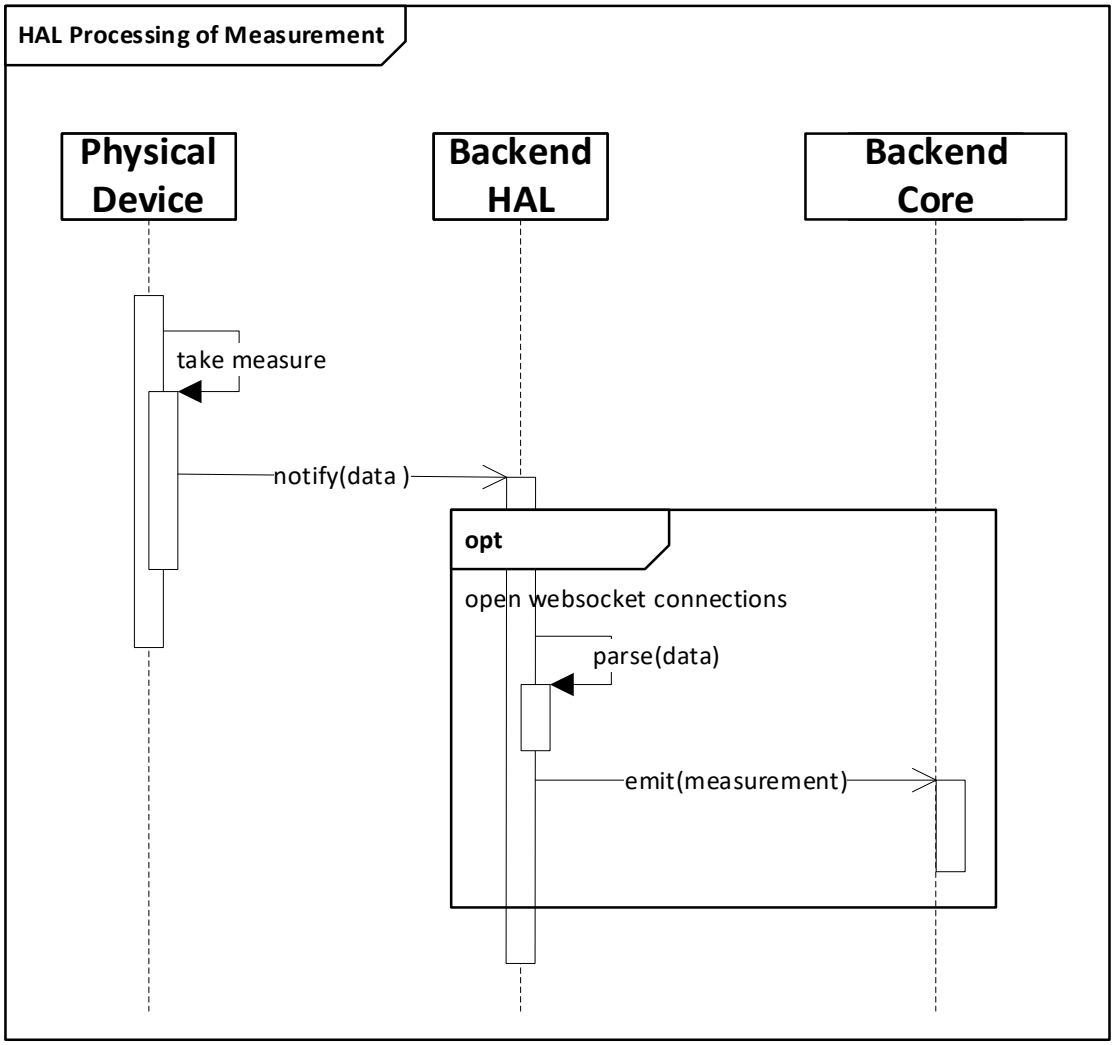

Abbildung 11: Sequenzdiagramm zur Verarbeitung eines Messwerts im Backend HAL

7.3 Systemsequenzdiagramm "Verarbeitung einer Messung im Core"

Sobald der HAL des Backend dem Core eine neue Messung sendet, wird diese dort verarbeitet, in der Datenbank persistiert und danach an das Frontend oder einen anderen API-Client weitergegeben.

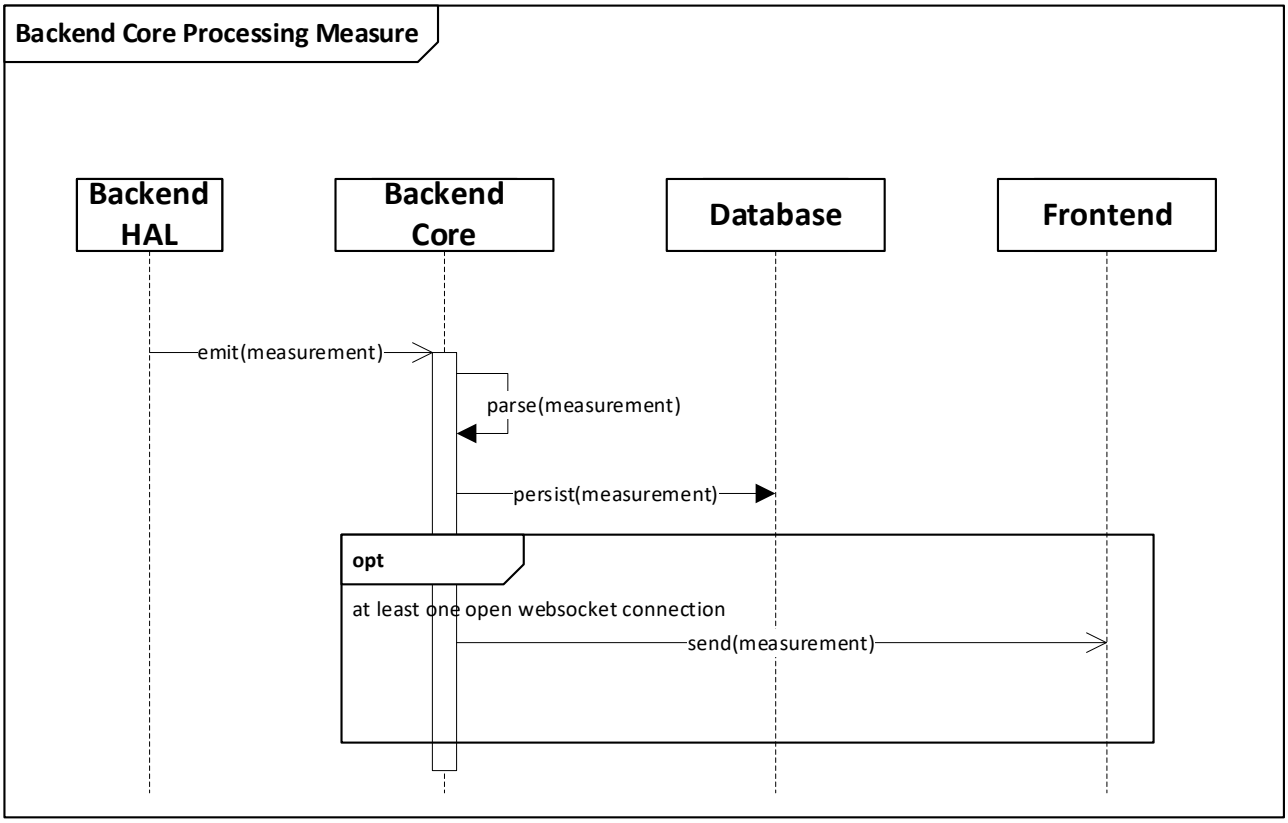

Abbildung 12: Sequenzdiagramm zur Verarbeitung eines Messwerts im Backend Core

8. Schnittstellen

8.1 Backend Websocket API

Verfügbar auf folgenden Hosts:

- **Host HAL**: ws://moodlebox.home:**4**000
- **Host Core**: ws://moodlebox.home:**3**000

8.1.1 Data Event

Event	Format	Beschreibung
data	JSON	Übermittelt gemessene Sensorwerte mit Messzeit und USB-Gerät-ID.

Beispielantwort:

```
{
    "deviceId":"8888:99999",
    "time":"2020-12-12T13:26:01.999Z",
    "measurements": [
        { "sensorId":"S1", "value":10.97 },
        { "sensorId":"S2", "value":1.11  }
    ]
}
```

Tabelle 23: Beschreibung Data Event

8.1.2 Control Event

Event	Format	Beschreibung
control	JSON	Übermittelt Status- und Fehlerinformationen.

Beispielantworten:

```
{
    "messageType":"ERROR",
    // ErrorDto
    "message": {
        "errorType":"HAL_API_ERROR",
        "errorMessage":"/devices/connected"
    }
}
```

```
{
    "messageType":"ATTACHED",
    // DeviceDto
    "message": {
        "id":"8888:99999",
```

```
        "vendorName":"Fake Device - Tinovi",
        "serviceName":"FakeService",
        "sensors": [
            {
                "id":"S1",
                "name":"temperature sensor",
                "unit":"°C",
                "unitName":"Celsius",
                "type":"Temperature Sensor",
                "parseAttribute":"3"
            },
            {
                "id":"S2",
                "name":"soil moisture sensor",
                "unit":"Pa","unitName":"Pascal",
                "type":"Soil Moisture Sensor",
                "parseAttribute":"2"
            }
        ]
    }
}
```


Tabelle 24: Beschreibung Control Event

Anmerkungen:

- Folgende messageType's mit dazugehörigen message's sind möglich:
  - **ATTACHED**: DeviceDto
  - **DETACHED**: DeviceDto
  - **ERROR**: ErrorDto

8.2 Backend HTTP API

8.2.1 Verbundene Geräte

Verfügbar auf folgenden Hosts:

- **Host HAL**: HTTP://moodlebox.home:**4**000
- **Host Core**: HTTP://moodlebox.home:**3**000

Route	Methode	Format	Beschreibung
/devices/connected	GET	JSON	Gibt ein Array mit allen erkannten und verbunden USB-Geräten zurück.

Beispielantwort:

```
[
    {
      "id": "8888:99999",
      "vendorName": "Fake Device - Tinovi",
      "serviceName": "FakeService",
      "sensors": [
        {
          "id": "S1",
          "name": "temperature sensor",
          "unit": "°C",
          "unitName": "Celsius",
          "type": "Temperature Sensor",
          "parseAttribute": "3"
        }
      ]
    }
]
```

Tabelle 25: Beschreibung API verbundene Geräte

8.2.2 Gespeicherte Messdaten

Verfügbar auf folgenden Hosts:

- **Host Core**: HTTP://moodlebox.home:**3**000

Route	Methode	Format	Beschreibung
/historydata	GET	JSON	Gibt ein Array mit allen im Core gespeicherten Messdaten in Form von MeasurementDto's zurück (kann Messdaten von unterschiedlichen USB-Geräten beinhalten).

Beispielantwort:

```
[
    {
      "deviceId": "8888:99999",
      "time": "2020-12-12T13:23:21.931Z",
      "measurements": [
        { "sensorId": "S1", "value": 17.97 },
        { "sensorId": "S2", "value": 1.11 }
      ]
    },
    {
      "deviceId": "8888:99999",
      "time": "2020-12-12T13:23:24.431Z",
      "measurements": [
        { "sensorId": "S1", "value": 28.97 },
        { "sensorId": "S2", "value": 1.11 }
      ]
    }
]
```

Tabelle 26: Beschreibung API gespeicherte Messdaten

8.3 Sensoren

8.3.1 Pico Technologies DrDAQ Board

Das Pico Technologies DrDAQ Sensor-Board bietet zur Konfiguration der Sensoren und zur Abfrage derer Daten API Funktionen in der Programmiersprache C an. Dieses API muss vom HAL des Backend und somit mit der Programmiersprache TypeScript angesprochen werden.

Es standen die folgenden beiden Möglichkeiten zur Auswahl, um die Kommunikation mit dem Gerät zu ermöglichen:

- Angebotene C-Funktionen des Gerätetreibers direkt nutzen.
- Vom Hersteller angebotene Python-Library Funktionen einsetzen, welche die C Funktionen umhüllen, damit Python Skripte erstellen und diese mit JavaScript ausführen.

Für die Umsetzung der ersten Variante existiert die Compiler Toolchain emscripten, welche C-Quellcode in JavaScript Module kompilieren kann. Die Funktionen der JS Module könnten dann direkt in anderen JS Dateien aufgerufen werden. Der Nachteil dieser Lösung ist, dass jede C-Quellcode-Datei, welche der Hersteller als API in einem SDK anbietet, einzeln kompiliert werden müsste. Dabei müsste zudem jede Funktion explizit angegeben werden, welche zu JS kompiliert werden soll. Um dies zu umgehen, können die benötigten C-Funktionen im Quellcode zwar speziell annotiert werden, dies wäre aber ebenfalls mit einem Mehraufwand verbunden. [5] Als Variation dazu bietet es sich an, ein C-Skript zu erstellen, welches die API-Funktionen verwendet und mit emscripten in ein JS-Modul kompiliert werden kann. Aufgrund der besseren Machbarkeit wird im folgenden Vergleich diese Variante weiterverfolgt.

Bei der Verwendung der Python-Wrapper Library muss das SDK des Herstellers ebenfalls installiert werden, danach lässt sich das C-API jedoch über die im SDK enthaltenen Python Funktionen ansprechen. In der Node.js Applikation kann dann ein Child-Prozess erstellt werden, welcher ein erstelltes Python Skript ausführt. Hierbei ist die Übertragung der Rückgabewerte vom Python Skript in die Node.js Applikation jedoch weniger intuitiv. [6]

Mit untenstehendem Vergleich werden die Stärken und Schwächen beider Möglichkeiten einander gegenübergestellt und dabei die passendere Lösung evaluiert.

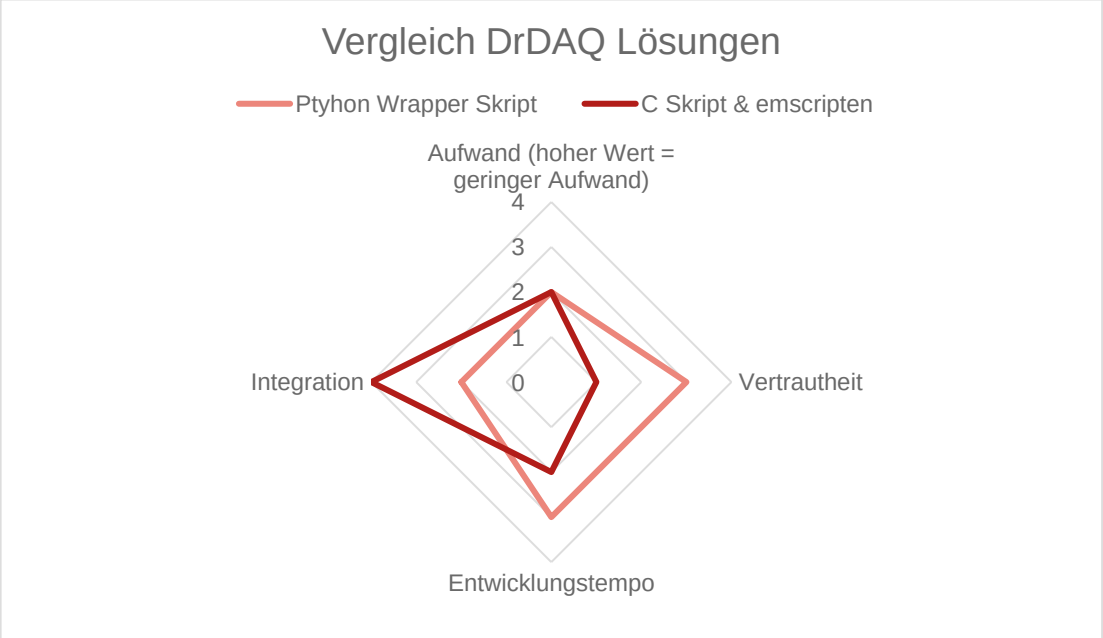

Abbildung 13: Vergleich Python Skript und C Skript für die DrDAQ Integration

Erklärungen zum Diagramm:

Python Wrapper Skript:

- Aufwand:
  - – Der Aufwand wird als gross eingestuft, da sowohl ein eigenes Skript geschrieben sowie der Einsatz einer Library für das Ausführen von Child-Prozessen in Node.js studiert werden muss.
- Integration:
  - – Die Integration ins Node.js ist möglich, aber umständlicher. Da das Skript in einem eigenen Prozess läuft, braucht es für die Rückgabe der Daten in den Node.js Prozess eine Interprozesskommunikation.
  - \+ Diese Lösung bietet jedoch den Vorteil, dass das Skript ebenfalls objektorientiert programmiert werden kann und damit dem Programmierstil der restlichen Applikation entspricht. Zudem kann das Skript dadurch besser strukturiert werden, was die Wartbarkeit verbessert.
- Vertrautheit:
  - \+ Die Hochsprache Python ist nach unserem Empfinden näher an Sprachen wie TypeScript oder C# als dies C ist. Dies erleichtert die Entwicklung und ein allfälliges Troubleshooting.
- Entwicklungstempo:
  - \+ Obwohl im Entwicklerteam nur wenig Erfahrung vorhanden ist, ist die Sprache wie bereits beschrieben näher an TypeScript als C. Daher wird davon ausgegangen, dass mit diesem Ansatz schneller Resultate erreicht werden können.
  - \+ Eine Einarbeitungsphase in die Sprache ist nötig, die zeitliche Dauer dieser wird aber als gering eingestuft.

C Skript & emscripten:

- Aufwand:
  - – Der Aufwand wird ebenfalls als hoch eingestuft. Auch bei dieser Variante muss ein Skript, hier in der Sprache C, geschrieben und die Funktionalität der emscripten Library erlernt werden.

- Integration:
  - \+ Nach der Kompilation der C Funktionen nach JavaScript können aus der Node.js Applikation normale JS Funktionen aufgerufen werden. Dabei können der Funktion die Argumente direkt als Parameter übergeben werden.
  - \+ Soll das Skript parallel zum Node.js Prozess laufen, kann das Node.js Modul "worker_threads" verwendet werden, welches Memory Sharing zwischen den Threads erlaubt [7]. Es bedarf keiner Interprozesskommunikation.
- Vertrautheit:
  - – Das Entwicklerteam konnte bislang nur wenig Erfahrung mit der Sprache C erlangen.
  - – Da es sich bei C anders als bei Python nicht um eine Hochsprache handelt, sind uns auch ihre Konzepte weniger vertraut.
- Entwicklungstempo:
  - – Aufgrund der fehlenden Kenntnisse zur Sprache wird damit gerechnet, dass mehr Zeit in die Einarbeitung fliesst. Aufgrund dessen können Resultate weniger schnell erzielt werden.
  - – Ein allfälliges Troubleshooting wird als aufwändiger erachtet.

Die Attribute "Vertrautheit" und "Entwicklungstempo" werden im Vergleich höher gewichtet als die "Integration". Dies ist dadurch zu begründen, dass mit diesen Eigenschaften bei Problemen eine schnellere und bessere Lösung gefunden werden kann. Dafür wird die umständlichere Integration in die Node.js Applikation in Kauf genommen. Anhand des obenstehenden Vergleichs fiel der Entscheid auf die Variante "Python Skript mit Child-Prozess".

8.4 Integration in Moodle

Ein Teil der Anforderungen an unsere Studienarbeit ist die Integration der erstellten Single Page Applikation in die Lernplattform Moodle, welche auf dem Raspberry Pi installiert ist. Auf dieser Lernplattform werden schlussendlich die Kurse angeboten, welche das Experimentieren mit den Sensoren beinhalten.

8.4.1 HTML iframe

Die Integration der Single Page Applikation ist mittels einem HTML iframe umgesetzt. Dies bedeutet, dass die Webapplikation unabhängig von Moodle als normale Webseite programmiert werden kann und kein Plugin für Moodle erstellt werden muss. Zudem kann die Webapplikation somit auch ausserhalb von Moodle aufgerufen werden, was einen flexiblen Einsatz dieser ermöglicht.

8.4.2 Umsetzung

Moodle bietet von Haus aus die Möglichkeit, den Inhalt eines erstellten Kurses über einen HTML Editor anzupassen. [8] In diesem HTML Editor kann ein iframe HTML-Element eingefügt und eine beliebige Webseiten-URL als Quelle angegeben werden. Als Quelle kann die URL der VMRP Web-App angegeben werden, welche dadurch direkt in einem Moodle-Kurs dargestellt wird.

Die Breite des iframe wird vom Moodle anhand der Bildschirmbreite der Geräte angepasst. Die Höhe wird beim iframe fix eingegeben und müsste über ein Skript abhängig von der Bildschirmhöhe angepasst und skaliert werden. Da solch ein Skript als Erweiterung für Moodle programmiert werden müsste, fällt dies nicht in den Rahmen dieser Studienarbeit und wird folglich nicht angedacht und umgesetzt.

9. Deployment

Das nachfolgende Diagramm soll aufzeigen, auf welche Hardwarekomponenten die verschiedenen Applikationsteile bei der Verwendung verteilt sind.

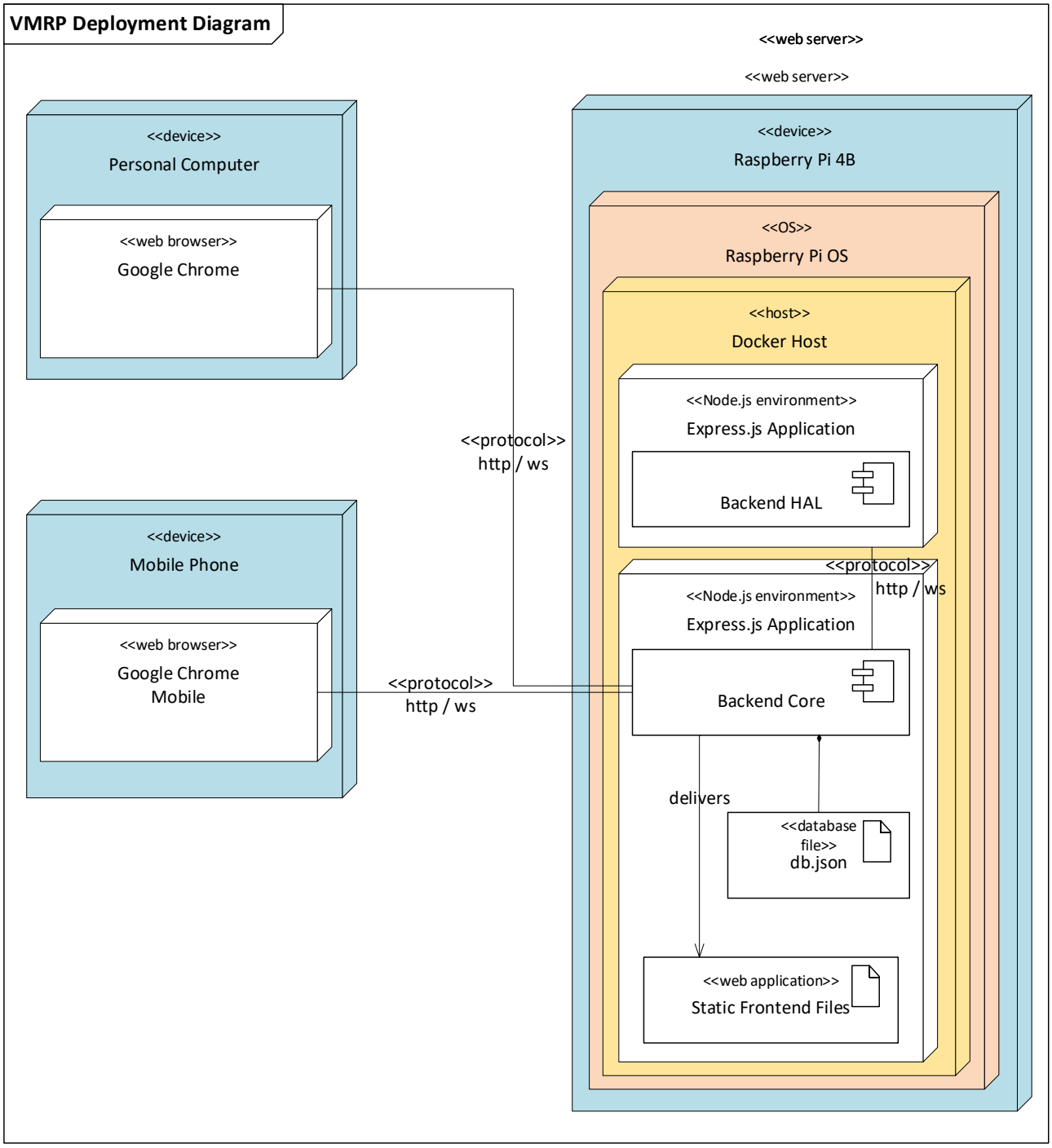

Abbildung 14: Deploymentdiagramm VMRP

Erklärung zum Diagramm:

- Es handelt sich um eine Two-Tier Architektur, bestehend aus dem persönlichen Gerät des Benutzers und einem Raspberry Pi Minicomputer, an welchen die Sensoren angeschlossen sind.
- Clients, Personal Computer oder Mobile Phones greifen über ein Websocket- oder ein HTTP API auf den Core Teil des Backend zu.
- Der Hardware Abstraction Layer des Backend läuft separat vom Core in einer eigenen Express.js Applikation.
- Der Core Teil des Backend agiert als Webserver und liefert den Clients die statischen Files der Single Page Applikation des Frontend aus.

9.1 Docker & Docker Compose

Das Deployment der Applikation VMRP wurde mit Docker und Docker Compose realisiert. Nachfolgende Vorteile führten zu dieser Entscheidung.

9.1.1 Vorteile

- **Treiberinstallation**: Gewissen USB-Geräte benötigen einen zusätzlichen Treiber, damit Applikationen mit ihnen kommunizieren können. Der Treiber kann mittels Docker bereits ins Image integriert werden. Somit entfällt die erneute Installation auf jedem Zielsystem.
- **Installation npm-Dependencies**: Um eine Applikation, welche mit Node.js und Vue.js erstellt wurde, betreiben zu können, müssen meist Software Dependencies, in Form von npm-Paketen installiert werden. Je nach Paket muss dieses für die zugrunde liegende CPU-Architektur gebuildet sein. Der Einsatz von Docker hilft dabei, diese plattformabhängigen Installationen zu abstrahieren und das Bereitstellen der Software auf dem Zielsystem zu vereinfachen.
- **Update der Applikation**: Wenn neue Features für die Applikation im GitLab bereit sind, kann durch einen einfachen Mausklick in der GitLab-Pipeline neue Docker Images erstellt werden. Diese können über Docker Compose auf dem Zielsystem heruntergeladen werden. Dies ermöglicht es, neue Features über das Internet auszuliefern. Das "Pullen" der neuen Features eignet sich zudem als Deployment Strategie, da die Raspberry Pi's nur in unterschiedlichen Intervallen an ein Netzwerk angeschlossen werden.
- **Buildx**: Dank dem Einsatz des CLI Plugins Buildx von Docker in der CI-Pipeline ist es möglich, Docker Images für eine Vielzahl von CPU-Architekturen zu erstellen. Damit lässt sich die Applikation für die unterschiedlichsten Zielsysteme paketieren und auf diesen betreiben.

Anmerkung:

Das Basisimage des HALs besteht aus einem Ubuntu-Node Image und die Basis des Cores bildet ein Alpine-Node Image. Der Grund für den Unterschied besteht in der nötigen Kommunikation mit den USB-Geräten, welche für den HAL notwendig ist. Aus Gründen von Best Practices ist zu Beginn für beide Docker-Images die Alpine-Node Version verwendet worden. Alpine Images sind bekannt dafür, eine möglichst kleine Dateigrösse aufzuweisen und nur den allernötigsten Funktionsumfang zu beinhalten. Es lag höchstwahrscheinlich an dem geringen Funktionsumfang, dass mit der Verwendung eines Alpine-Images die Kommunikation vom HAL zu den USB-Geräten nicht mehr richtig funktionierte. Z.B. konnte der HAL nicht mehr erkennen, ob ein USB-Gerät ausgesteckt wurde. Dies hat dazu geführt, dass als Basisimage für den HAL ein Ubuntu-Node Image verwendet wird, mit welchem die USB-Kommunikation ohne weitere Konfigurationen funktioniert.

10. Tabellenverzeichnis

11. Abbildungsverzeichnis

12. Literaturverzeichnis

[1] S. Brown, «The C4 model for visualising software architecture,» [Online]. Available: https://c4model.com. [Zugriff am 19 Novemeber 2020].

[2] M. Rohr, Interviewee, *SA VMRP: Anforderungsanalyse.* [Interview]. 17 September 2020.

[3] statcounter, «Mobile Operating System Market Share Africa,» [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/africa/#monthly-202009-202009-bar. [Zugriff am 27 Oktober 2020].

[4] S. McGowan, «5 Alternatives To Material Design,» [Online]. Available: https://usabilitygeek.com/alternatives-to-material-design/. [Zugriff am 13 November 2020].

[5] E. Maino, «Webassembly: calling C functions from Javascript with emscripten,» 27 Juli 2017. [Online]. Available: https://medium.com/@eliamaino/calling-c-functions-from-javascript-with-emscripten-first-part-e99fb6eedb22. [Zugriff am 29 Oktober 2020].

[6] P. Koulianos, «How to Run a Python script from Node.js,» 16 Januar 2020. [Online]. Available: https://medium.com/swlh/run-python-script-from-node-js-and-send-data-to-browser-15677fcf199f. [Zugriff am 29 Oktober 2020].

[7] nodejs.org, «Worker threads,» [Online]. Available: https://nodejs.org/api/worker_threads.html#worker_threads_worker_threads. [Zugriff am 12 Dezember 2020].

[8] Moodle, «iframe,» 15 Januar 2018. [Online]. Available: https://docs.moodle.org/39/en/Iframe#Code_example. [Zugriff am 18 Oktober 2020].

[9] Mozilla, «iframe,» 23 März 2019. [Online]. Available: https://developer.mozilla.org/de/docs/Web/HTML/Element/iframe. [Zugriff am 17 Oktober 2020].

3.4. Softwaredesign

SA – VMRP

Autoren: Marco Gartmann, Fabian Thurnheer
Version: 1.0

Erstellt am: 13.11.2020
Letzte Änderung am: 18.12.2020

Inhaltsverzeichnis

1. Einführung

1.1 Zweck

In diesem Dokument wird beschrieben, wie die geplante Softwarearchitektur in der Software umgesetzt wurde. Es soll einen Einblick in die Strukturen und Abläufe der Softwarekomponenten gewähren. Zudem werden spezielle Implementationsdetails erläutert.

1.2 Gültigkeitsbereich

Die Gültigkeit dieses Dokuments beschränkt sich auf die Laufzeit des Moduls "Studienarbeit" im Herbstsemester 2020.

2. Design Frontend

2.1 Libraries Userinterface

Im Vue.js-Projekt kamen folgende Dependencies zum Einsatz, um das Userinterface zu gestalten:

- **Highcharts**: Um gemessene Sensordaten in einem Graphen darstellen zu können, wurde der offizielle Highcharts Wrapper für Vue.js verwendet. Highcharts ist eine bekannte JavaScript Chart Library, welche eine breite Community besitzt und diverse vorgefertigte Grafiken zur Verfügung stellt.
- **Vuetify**: Die komplette Struktur des Userinterfaces und ihr Design wurde mit Vuetify umgesetzt. Vuetify ist eine Material Design Library und bietet von Haus aus eine Vielzahl von Vue.js Components, welche zur Gestaltung einer Webapplikation verwendet werden können. Wie in der Dokumentation der Softwarearchitektur beschrieben, soll das zu entwickelnde UI anhand der Material Design Prinzipen gestaltet werden. Da sich Vuetify bei der Gestaltung der Components an diese Prinzipien hält, eignet es sich bestens, um diese Anforderung zu erfüllen.

2.2 Vuex Store

2.2.1 Aufbau

Die eigentliche Komplexität des Frontend befindet sich, wie im Abschnitt "Softwarearchitektur" beschrieben, im Layer der Business Logic. Deshalb wird in diesem Kapitel dessen Aufbau nochmals genauer erläutert.

Die Business Logic besteht hauptsächlich aus dem Vuex Store. Vuex ist ein Plugin, mit welchem einer Vue.js Applikation ein State Management Pattern in Form einer Library hinzugefügt werden kann.

Der Store ist in drei Module aufgeteilt:

- **device:** Beinhaltet den State über die aktuell angeschlossenen USB-Geräte. Bietet Funktionen, um diese abzufragen, hinzuzufügen und zu entfernen.
- **measurement:** Alle übertragenen Messungen sind in diesem Modul abgespeichert. Dies beinhaltet bereits aufgezeichnete Messungen, welche im Backend (Core) über eine gewisse Zeit gespeichert wurden, sowie Messungen, welche live übertragen werden. Bietet Funktionen, um Messungen hinzuzufügen und zu entfernen.
- **message:** Bildet den Kern des Stores. Das message-Modul registriert sich über die Klasse WsApiClient auf Websocket-Events und entscheidet je nach Art der Message, wie die erhaltene Nachricht zu verarbeiten ist. Die Message wird entweder über die Reaktivität des message-Moduls direkt an einen Vue.js Component oder an die anderen Store-Module weitergereicht. Diese sorgen mit ihrer eigenen Reaktivität dafür, dass die Messages automatisch in den entsprechenden Vue.js Components dargestellt werden.

  Der Vorteil dieser Variante besteht in der "Single Responsibility" des message-Moduls. Dieses konzentriert sich darauf, Messages zu empfangen und weiter zu reichen, wodurch die anderen Module den Fokus auf die Verwaltung ihres eigenen States legen können.

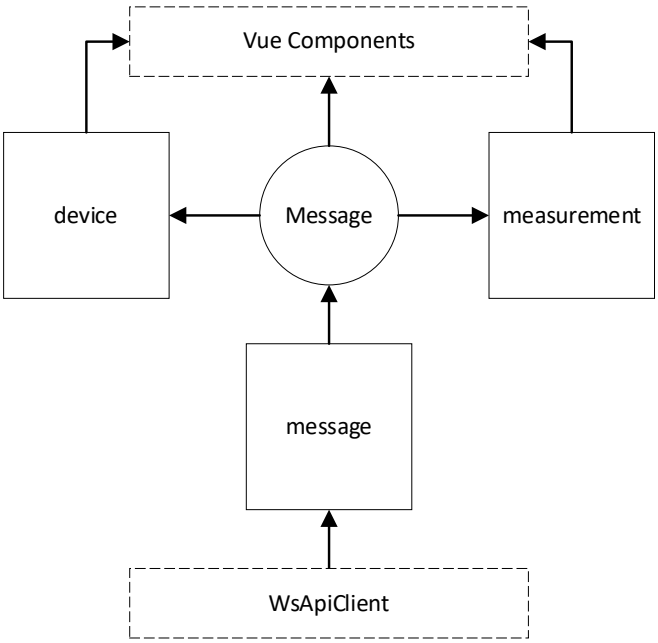

Abbildung 15: Möglicher Fluss einer Message durch den Vuex Store

2.2.2 Typisierung

Wie in der Aufgabenstellung beschrieben, ist für das Frontend Vue.js mit TypeScript verwendet worden. Die zum Startpunkt des Projektes zur Verfügung stehende Version von Vue.js verfügte aber nur über einen schwachen TypeScript Support. Der Einsatz von geeigneten Dependencies konnte diesem Umstand entgegenwirken. Die Abhängigkeiten "vue-property-decorator" und "vue-class-component" ermöglichten es, in Vue.js Components TypeScript Klassen zu verwenden. Die Umsetzung erfolgte, wie die Namen bereits verraten, über Decorators auf oder in den entsprechenden TypeScript Klassen. Dies erhöht die Leserlichkeit, die Wartbarkeit und korrekte Typisierung des Codes.

Die Datenhaltung und die Business Logik wurde im Frontend mittels Vuex Store implementiert. Dieser besteht normalerweise aus JavaScript Objekten, welche wiederum in Module gegliedert sind. Da Vue.js Components mit dem Store interagieren, musste auch hier eine Lösung gefunden werden, um die Module des Stores in TypeScript Klassen umzuwandeln und so zu typisieren. Dies gelang mit der Dependency "vuex-module-decorators". Die Umsetzung erfolgte ebenfalls über Decorators der Dependency. Daraus resultierte eine Vue.js Applikation, welche zur Compilezeit eine vollständige Typisierung bietet, was ohne den Einsatz der Dependencies nicht möglich gewesen wäre.

3. Logische Architektur Core

Dieses Kapitel ergänzt das gleichnamige Kapitel des Abschnitts "Softwarearchitektur", wobei in diesem Diagramm ein Zoom-In auf Klassen-Ebene geschieht. Dabei soll der Zusammenhang der einzelnen Klassen ersichtlich und verständlich werden.

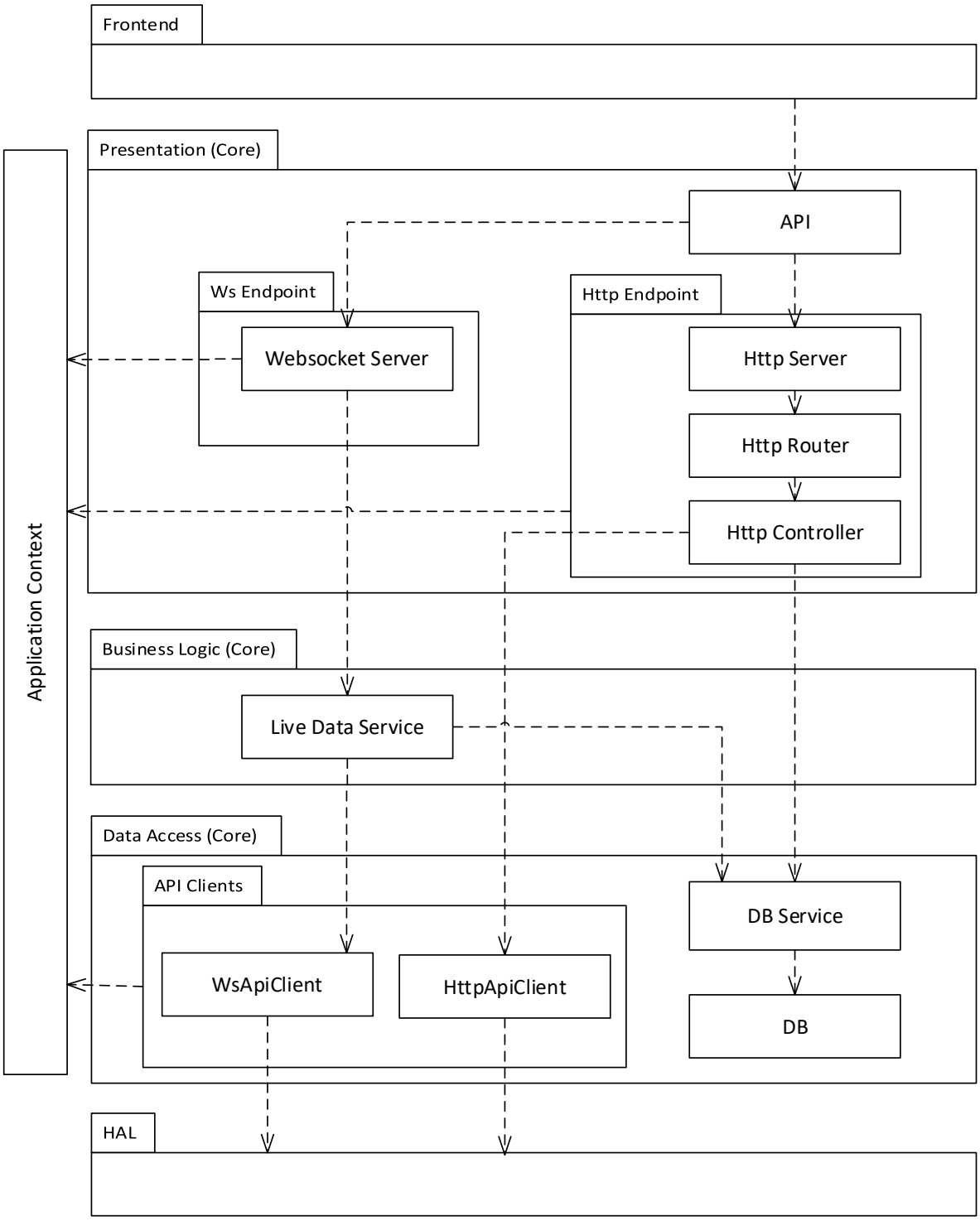

Abbildung 16: Logische Klassenarchitektur Core

4. Logische Architektur HAL

Wie das vorangegangene Kapitel ergänzt auch dieses das gleichnamige Kapitel des Abschnitts "Softwarearchitektur", wobei in diesem Diagramm ein Zoom-In auf Klassen-Ebene geschieht. Dabei soll der Zusammenhang der einzelnen Klassen ersichtlich und verständlich werden.

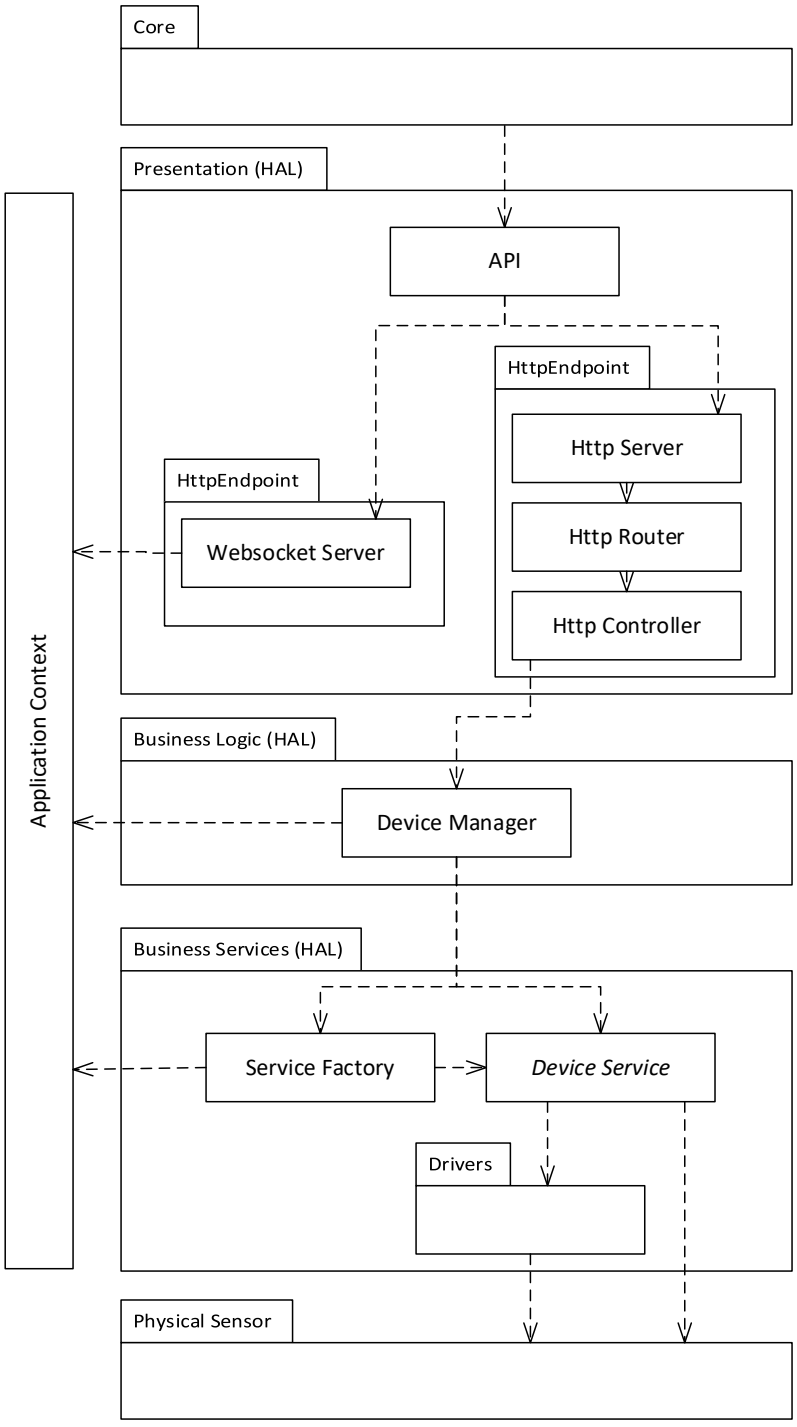

Abbildung 17: Logische Architektur HAL

5. Umsetzung Application Context

Um Objekte wie Logger, Error Handler und globale Settings in Klassen verfügbar zu machen, welche sich in unterschiedlichen Layern des Backend befinden und dabei das Layering der Softwarearchitektur nicht zu verletzen, wurde ein Application Context implementiert. Dieser erstreckt sich als vertikaler Layer über sämtliche andere Layer der Architektur.

Der Application Context beinhaltet folgende Komponenten:

- Error Handler
- Logger
- Settings

Beim Start des Cores und des HALs wird je eine Instanz des Application Context erstellt und dieser den oben erwähnten Komponenten zugewiesen. Dieser Application Context wird vom Entrypoint des Cores und des HALs nach dem "Parametrize from Above" Prinzip den anderen Klassen über den Konstruktor mitgegeben.

Auf diese Art und Weise konnte eine vereinfachte Art der Dependecy Injection erreicht und global benötigte Komponenten in einer einheitlichen Klasse gekapselt werden.

Die Environment Variablen, welche von ausserhalb des Programms gesetzt werden können, wurden ebenfalls in die Settings des Application Context aufgenommen. Der Vorteil davon ist, dass bei einer Umbenennung der Umgebungsvariablen im Programm nur eine Klasse angepasst werden muss. Würde man in den Klassen direkt auf die Umgebungsvariablen zugreifen, wäre der Aufwand einer solchen Änderung erheblich höher.

6. Wichtige Abläufe

6.1 Sequenzdiagramm "Anschliessen eines Sensors im HAL"

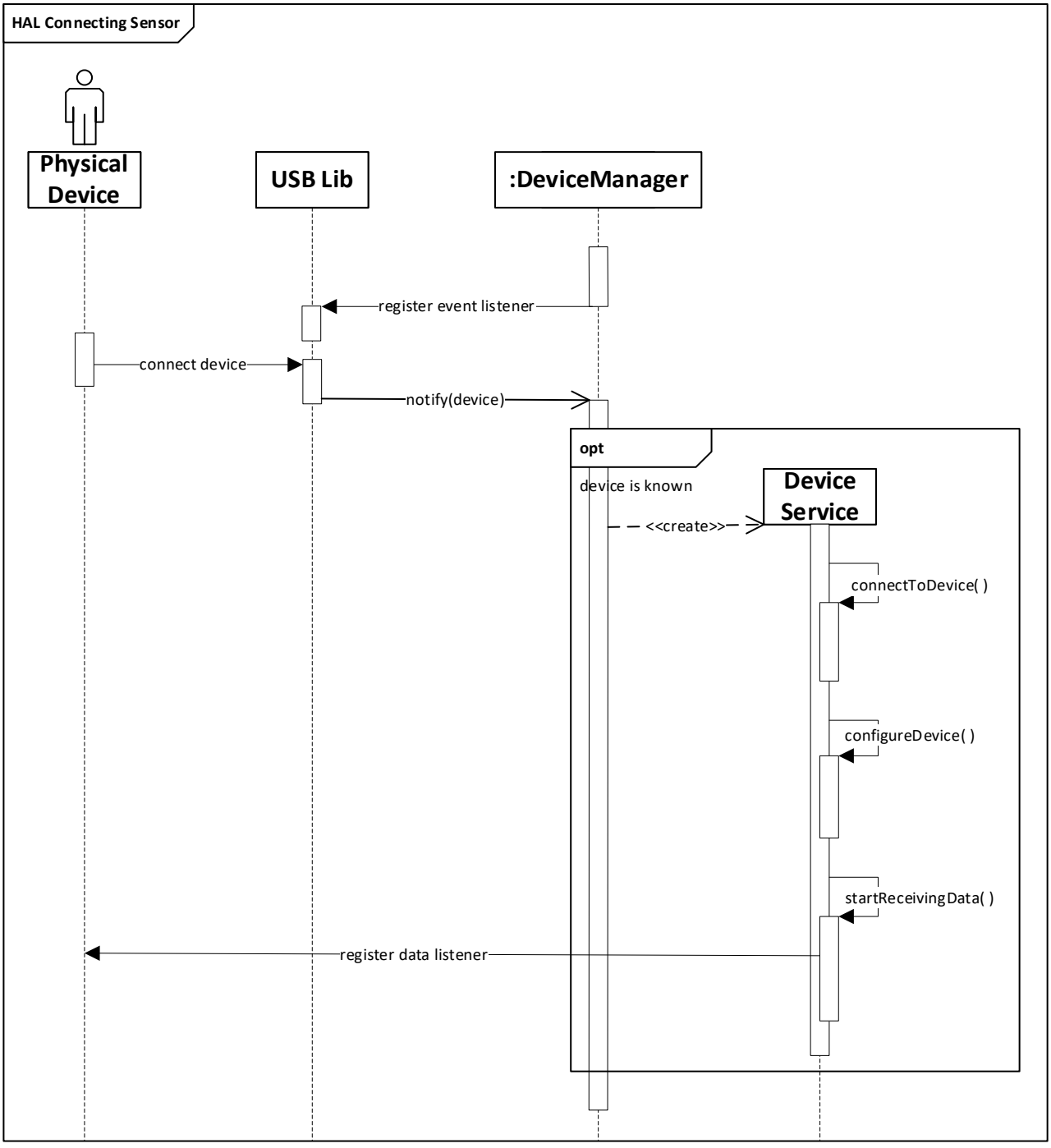

Abbildung 18: Sequenzdiagramm zum Anschliessen eines Sensors im HAL des Backend

Das oben dargestellte Diagramm abstrahiert und beschreibt das Zusammenspiel der Klassen im HAL des Backend beim Anschliessen eines physikalischen Sensor-Geräts (im Diagramm 'Device' genannt) an den Raspberry Pi Minicomputer.

6.2 Sequenzdiagramm "Verarbeitung einer Messung im HAL"

Folgendes Diagramm dient zur Erklärung der Abfolge der Bearbeitung eines Messwerts, welcher von einem physikalischen Sensor stammt.

Vorbedingung dafür ist, dass zuvor ein Sensor, wie im Diagramm "HAL Connecting Sensor" dargestellt, angeschlossen wurde.

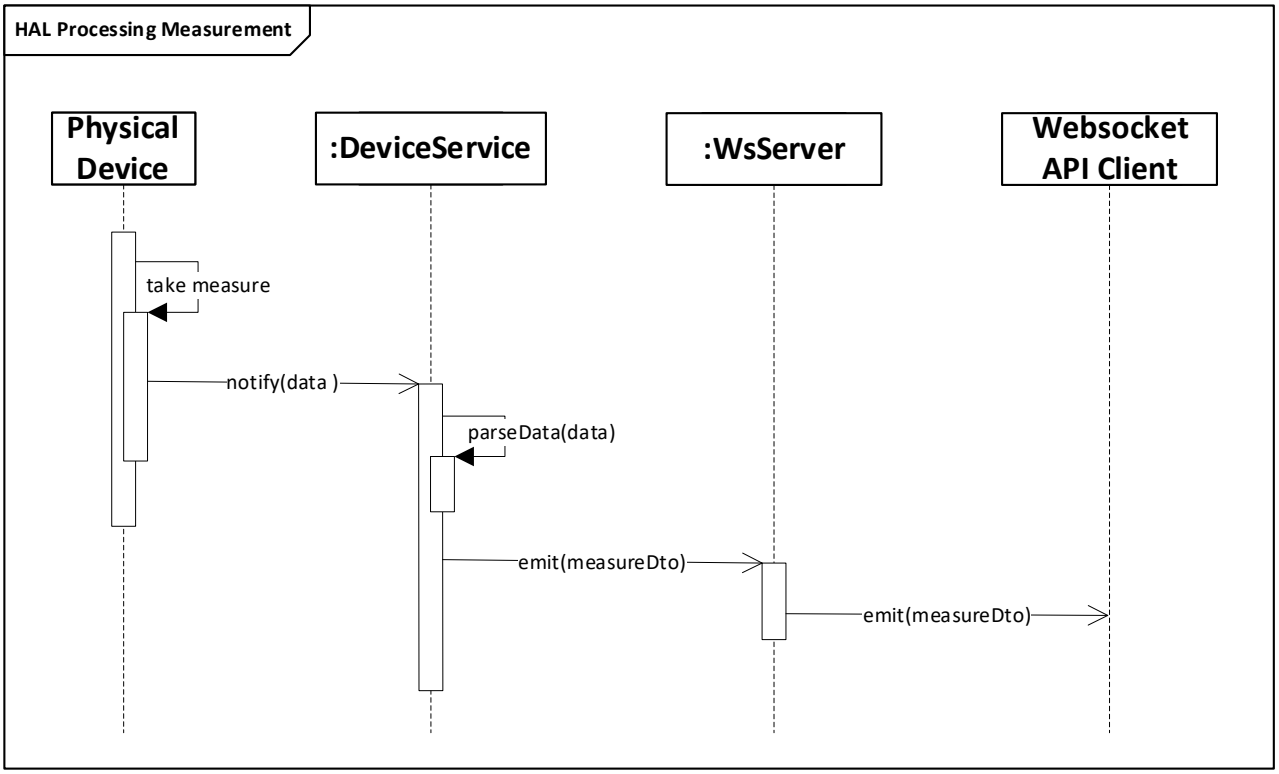

Abbildung 19: Sequenzdiagramm zur Verarbeitung eines Messwerts im Backend HAL

6.3 Sequenzdiagramm "Verarbeitung einer Messung im Backend Core"

Sobald der HAL des Backend dem Core eine neue Messung sendet, wird diese dort verarbeitet, in der Datenbank persistiert und danach an das Frontend oder einen anderen API-Client weitergegeben.

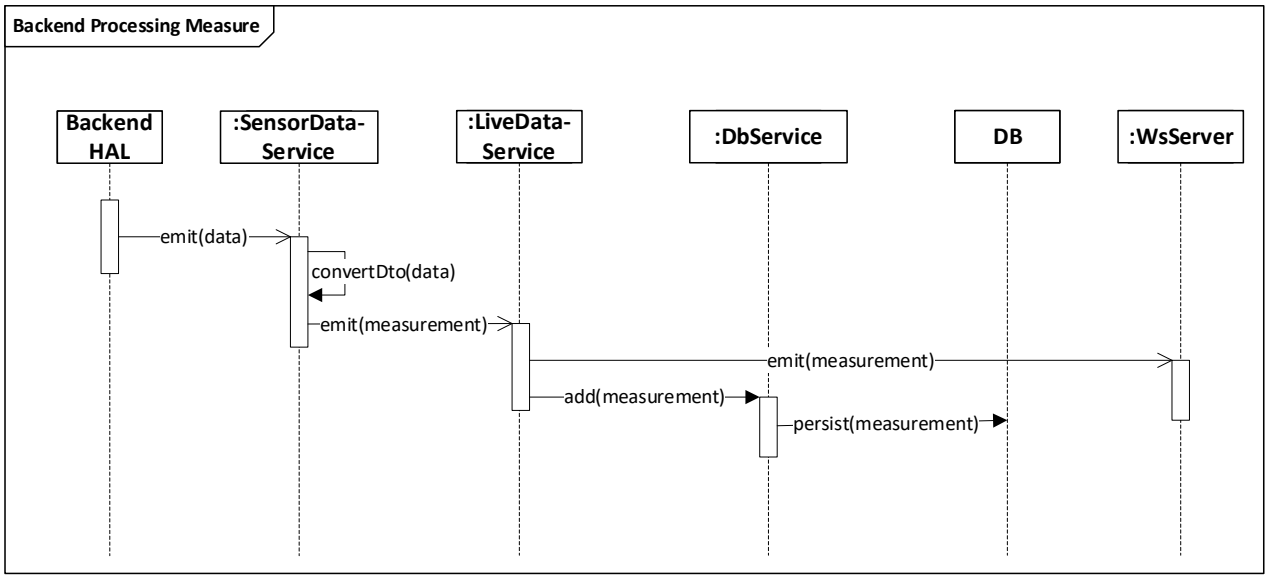

Abbildung 20: Sequenzdiagramm zur Verarbeitung eines Messwerts im Backend Core

7. Klassendiagramm

Das folgende Diagramm dient zur Visualisierung der Klassen in der Software.

7.1 Klassendiagramm Backend HAL

Abbildung 21: Klassendiagramm Backend HAL

8. Umsetzung Fehlerbehandlung

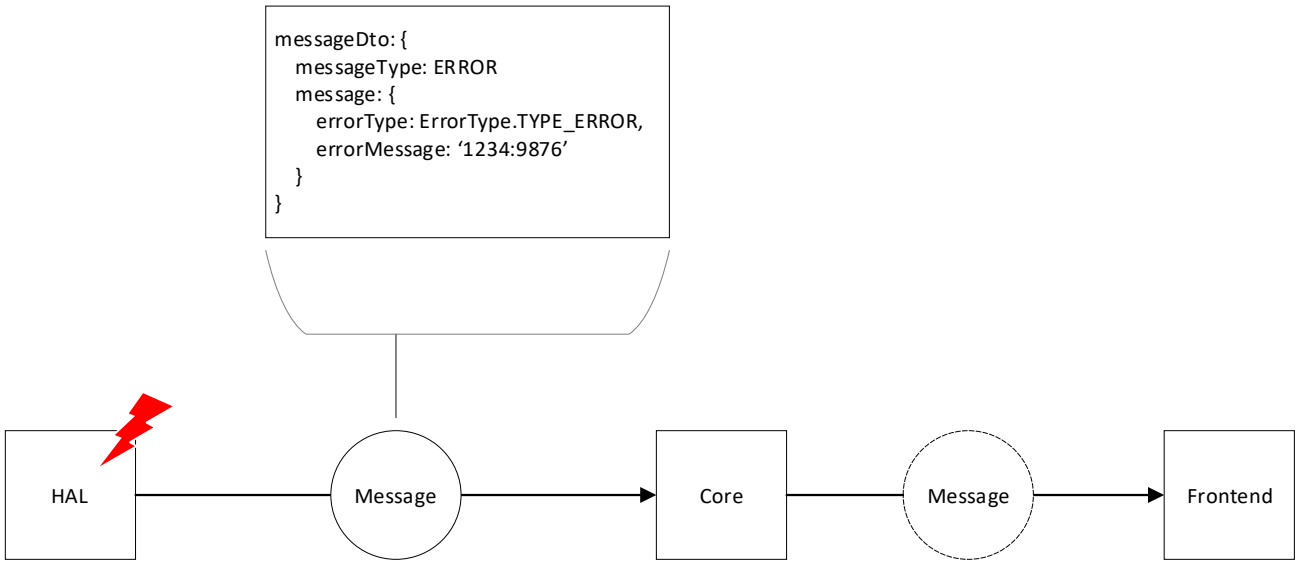

Abbildung 22: Darstellung des Error Message Fluss über Websocket

Das Error Handling wurde mit einer Error Handler Klasse, welche sowohl im HAL wie auch im Core existiert, implementiert. Tritt in einem der beiden Softwarekomponenten ein Fehler auf, wird er dem dafür verantwortlichen Error Handler überreicht.

Je nach Typ des Fehlers leitet der Error Handler einen der folgenden Schritte ein:

- Logging des Errors in die Konsole.
- Weitergabe des Errors über die Websocket Verbindung. Dazu wird der Error in ein messageDto mit dem messageType "ERROR" verpackt, welche den eigentlichen Error als message Property beinhaltet (in Abbildung 22 dargestellt).
- Bei schwerwiegenden Fehlern leitet der Error Handler den geordneten Shutdown der Software über die Klasse GlobalErrorHandler ein.

In einer Error Message werden bewusst nur der errorType und eine optionale message, welche z.B. die Geräte-ID des fehlerhaften Geräts beinhaltet, versendet. Das Generieren der Fehlermeldung, welche in der Web-App angezeigt wird, wird mit folgenden Überlegungen dem Frontend überlassen:

- Trennung der Komponenten: Eine künftige Anpassung der anzuzeigenden Fehlermeldung soll keinen Einfluss auf die Implementation des Backend haben.
- Internationalisierung: In einem zukünftigen Szenario, in welchem ein Benutzer die UI-Sprache seinen Bedürfnissen anpassen möchte, kann das Frontend die Fehlermeldung in dieser Sprache erzeugen. Auf Grund des Layerings hat der HAL keine Kenntnis über solche Einstellungen. Das Generieren der Meldung in verschiedenen Sprachen fällt demzufolge nicht in seinen Aufgabenbereich.

9. Integration Pico Tech. DrDAQ

9.1 Umsetzung DrDAQ Python Skript

Wie im Kapitel Softwarearchitektur beschrieben wurde für die Integration des Pico Technologies DrDAQ Sensor-Boards eine Lösung evaluiert, bei welcher in der Node.js Applikation des HALs ein Child-Prozess gestartet wird. Dieser Child-Prozess führt ein Python Skript aus, mit welchem mit dem Gerät kommuniziert wird.

Das Abfragen der aktuellen Messwerte sollte dabei in einem steuerbaren Intervall geschehen.

In einer ersten Version des Skripts wurde diese Abfrage in einer Schlaufe getätigt, an deren Ende mit einem Timeout auf das Verstreichen der mitgegebenen Intervallzeit gewartet wurde:

```
#...
def __start_capturing(self):
     try:
        while self.isRunning:
            measure_set = {}
            for sensor in drDaq.USB_DRDAQ_INPUTS:
                #get measurement value from board and add it to measure_set

            sys.stdout.write(json.dumps(measure_set))
            sys.stdout.flush()
            time.sleep(self.INTERVAL_IN_SECONDS)
        #...
```

Abbildung 23: Code-Ausschnitt DrDAQ Python Skript

Um die Genauigkeit des Intervalls zu prüfen, wurde die Ausgabe der JSON-formatierten Messwerte durch die Ausgabe der Systemzeit ersetzt und das Skript mit einem 1s Intervall gestartet:

```
moodlebox@moodlebox:~/mg/sa/backend/hal/business_services/drivers
$ python3 drDaqDriver.py 1
2020-12-06 18:26:15.423527
2020-12-06 18:26:17.558037
2020-12-06 18:26:19.691740
2020-12-06 18:26:21.825218
2020-12-06 18:26:23.959542
2020-12-06 18:26:26.093323
2020-12-06 18:26:28.227342
2020-12-06 18:26:30.360609
```

Abbildung 24: Python-Skript mit Drifting

In der resultierenden Ausgabe fallen zwei Punkte auf:

- Die Zeit zwischen den Ausgaben ist nicht ein ganzzahliger Sekundenbetrag. Zusätzlich erhöhen sich die Nachkommastellen bei jeder Ausgabe. Dieses Verhalten wird in diesem Kontext auch "Drifting" genannt.
- Die Zeit zwischen den Ausgaben liegt mit ca. 2s stark über dem angegebenen Intervall von einer Sekunde.

Der Fehler der ersten Version des Abfrage-Algorithmus war, dass nach einer Messdatenerhebung auf jeden Fall die definierte Intervall-Zeit abgewartet wurde, auch wenn die Erhebung die Zeitdauer des Intervalls bereits überschritten hat.

9.1.1 Verbesserung des Skripts

Als Massnahme wurde in einer zweiten Version des Skripts ein Algorithmus verwendet, welcher das Delta zwischen der nächsten geplanten Ausführung des Loops und der jetzigen Systemzeit errechnet und nur diese Zeit abwartet. Ist diese Zeit negativ – was bedeutet, dass die Ausführung des Algorithmus länger dauerte als das festgelegte Intervall – so wird ohne zu warten mit der Ausführung der nächsten Loop-Iteration fortgefahren [1].

```
def __do_every(self,period,func):
    def calculate_waiting_time():
        next_iteration = time.time()
        while True:
            next_iteration += period
            yield max(next_iteration - time.time(),0)
    time_to_wait = calculate_waiting_time()
    while True:
        time.sleep(next(time_to_wait))
        func()
```

Abbildung 25: verbesserte Version des Iterations-Algorithmus. Quelle: [1]

Diesem Iterations-Algorithmus kann die Funktion, welche die Messdaten der Hardware abfragt, als Parameter mitgegeben werden.

Bei der Überprüfung der Genauigkeit des zeitlichen Intervalls von einer Sekunde wurde festgestellt, dass diese Zeit nach wie vor nicht genau eingehalten wird und ebenfalls ein Drifting entsteht. Man beachte die Zahlen hinter dem Dezimalpunkt:

```
$ python3 drDaqDriver.py 1
2020-12-06 20:37:17.915508
2020-12-06 20:37:19.047337
2020-12-06 20:37:20.179501
2020-12-06 20:37:21.311319
2020-12-06 20:37:22.443153
2020-12-06 20:37:23.574706
```

Abbildung 26: Resultat des überarbeiteten Skripts mit Drifting

Der Grund dafür ist, dass das Abfragen aller Sensor-Channels des Boards länger als eine Sekunde dauert. Im Vergleich zur ersten Version des Algorithmus ist das zeitliche Delta zwischen den Ausführungen nun zwar bei ca. 1.13s anstelle von ca. 2s – bei der Vorgabe eines 1s-Intervals bei beiden Varianten.

Setzt man das Intervall auf zwei Sekunden, ist ersichtlich, dass das Drifting nicht mehr auftritt und die Zeit zwischen den Messungen auf die Hundertstelsekunde genau konstant bleibt:

```
$ python3 drDaqDriver.py 2
2020-12-06 20:37:44.005560
2020-12-06 20:37:46.004335
2020-12-06 20:37:48.004307
2020-12-06 20:37:50.004419
2020-12-06 20:37:52.005331
2020-12-06 20:37:54.004134
```

Abbildung 27: Resultat des überarbeiteten Skripts mit erhöhtem Intervall

Somit wurde die gewünschte Verbesserung der Intervall-Genauigkeit erreicht.

9.1.2 Hohe CPU-Prozentzahlen mit top

Im Zuge der Tests der nichtfunktionalen Anforderungen wurde bemerkt, dass das Python Skript überraschend hohe CPU-Anteile (bis zu 80%) für sich beansprucht. Dies konnte sowohl in der neuen wie auch in der ersten Version des Skripts beobachtet werden.

Dafür fand sich folgende Erklärung: Die CPU-Auslastung wurde mit dem Linux-eigenen Tool top angezeigt, welches für die Berechnung der CPU-Last eines Prozesses die Last jedes einzelnen CPU-Kerns zusammenzählt. Ist in einem 4-Kern Prozessor z.B. ein Kern zu 70% und ein anderer zu 50% ausgelastet, ergibt sich eine Gesamtauslastung von 120%. Die vollständige Auslastung wäre mit 4 Kernen bei 400% erreicht [2]. Dieser Modus nennt sich "Irix Mode". Mit Shift + i innerhalb von top kann der Modus zu "Solaris Mode" gewechselt werden, sodass die CPU-Beanspruchung einzelner Prozesse in Prozent der gesamthaft verfügbaren Leistung aller Kerne angezeigt wird.

Im Solaris Mode weist die alte Version des Skripts mit einem Intervall von einer Sekunde eine Auslastung von durchschnittlich 15% auf. Das Skript der neuen Version benötigt durchschnittlich 14% aller CPU-Kerne. Da eine Messdatenerhebung wie weiter oben beschrieben ca. 1.13s in Anspruch nimmt, wurde das Intervall bei der neuen Version des Skripts auf 2s gesetzt, was eine Sleeping-Zeit des Prozesses von 0.87s (2s – 1.13s) bedeutet. Somit haben die Prozesse beider Varianten ähnliche Sleeping-Zeiten, was einen qualitativen Vergleich beider CPU-Auslastungen ermöglicht.

Der Intervall-Algorithmus des überarbeiteten Skripts ist also zudem leicht ressourcenschonender als jener der ersten Variante.

9.2 Treiberinstallation in Docker Containern

Wie im Dokument der Softwarearchitektur bereits beschrieben, wird das Deployment mit Docker umgesetzt.

Da die Applikation schlussendlich in einem Docker Container betrieben wird, mussten die Pico Technologies Treiber für das DrDAQ Board ebenfalls im Dockerfile des HALs installiert werden.

Bei dieser Installation trat das Problem auf, dass die Treiberinstallation im Postinstall Skript des Softwarepakets fehlschlug, wie in untenstehendem Code-Ausschnitt ersichtlich ist.

```
# apt-get install -y libusbdrdaq
...
E: Post install script exited with code 2
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

Abbildung 28: Fehler bei der Installation der Treiber-Library

Der Exit Code 2 bedeutet dabei: "All builtins return an exit status of 2 to indicate incorrect usage, generally invalid options or missing arguments." [3]

Dies hatte zur Folge, dass das für die Kommunikation mit dem DrDAQ Board erstellte Python Skript die Treiber Library des DrDAQ Boards nicht finden konnte und fehlschlug.

9.2.1 Troubleshooting

Während dem Troubleshooting konnte festgestellt werden, dass die Installation auch im WSL fehlschlug. Es wird davon ausgegangen, dass das Skript generell ein Problem mit virtuellen Umgebungen hat.

Um herauszufinden, welcher Teil des Postinstall Skripts fehlschlug, wurde mit dem Befehl `find . -cmin -<time>` ein Diff des Dateisystems vor und nach der Installation des libusbdrdaq Pakets ausgegeben. Unter anderem werden bei der Installation folgende Dateien und Ordner angepasst oder erstellt:

```
./etc/udev/rules.d/95-pico.rules
./etc/ld.so.conf.d/picoscope.conf
./opt/picoscope/include/libusbdrdaq
./opt/picoscope/lib/libusbdrdaq.so        (1)
./var/lib/dpkg/info/libusbdrdaq.postrm
./var/lib/dpkg/info/libusbdrdaq.list
./var/lib/dpkg/info/libusbdrdaq.preinst
./var/lib/dpkg/info/libusbdrdaq.md5sums
./var/lib/dpkg/info/libusbdrdaq.postinst  (2)
./var/lib/dpkg/info/libusbdrdaq.config
```

Abbildung 29: Analyse der geänderten Files bei der Treiberinstallation

Wie zu sehen ist, wird die benötigte Treiber Library installiert (1). Trotzdem wurde sie vom Python Skript nicht gefunden.

Das Postinstall Skript (2) erstellt diverse Dateien und befüllt diese mit Inhalt. Eine Kontrolle dieser Files nach der Installation ergab, dass all diese Files erstellt werden konnten und den korrekten Inhalt hatten.

Einzig die letzte Zeile des Skripts…

```
udevadm control --reload-rules && udevadm trigger --attr-match=idVendor='0ce9'
```

Abbildung 30: Befehl zum Reload der udev Rules

… schlug beim manuellen Ausführen fehl. Wurde beim manuellen Ausführen der Match Attribute Teil weggelassen, schlug der Befehl nicht fehl. Der Debian Package Manager dpkg listete die Installation des Pakets jedoch weiterhin als fehlerhaft, weshalb die Treiber Library nach wie vor nicht gefunden wurde:

```
# dpkg -l libusbdrdaq
||/ Name               Version         Architecture   Description
+++-==================-===============-==============-==============================
iF  libusbdrdaq        2.0.40-1r2131   amd64          PicoScope USBDrDAQ Linux driver
```

Abbildung 31: Fehlerhafter Installationsstatus der Treiber-Library

9.2.2 Workaround

Beim gefundenen Workaround wird im Dockerfile der Exit Code der fehlschlagenden Installationen übersteuert, damit die Ausführung des Dockerfiles weiterläuft.

Nach der Installation des Pakets libusbdrdaq werden die udev Regeln manuell neu geladen.

Um die Installation des fehlerhaften libusbdrdaq Pakets fertigstellen zu können, wird dem problematischen Postinstall Skript ein "exit 0" eingefügt. Das Laden der udev Regeln im Skript schlägt weiterhin fehl, was jedoch missachtet werden kann, da diese zuvor bereits manuell geladen wurden. Da das Skript nun aber keinen Error mehr zurückgibt, kann die Installation des Pakets mit dem Debian Package Manager vervollständigt werden.

Mit dem nun vollständig installierten libusbdrdaq Paket wird die Treiber Library vom Python Skript wieder aufgefunden.

Der erklärte Workaround ist im folgenden Auszug des HAL Dockerfiles ersichtlich:

```
# install DrDaq drivers
# This installation is known to fail in virtual environments due to a buggy post-in-
stall script.
# Ref: https://www.picotech.com/support/topic40771.html
# The demanded drivers are being installed nevertheless.
# To continue the build of this Dockerfile, status 0 is returned manually.
RUN apt-get install libusbdrdaq -y; exit 0

# bug in picotech's post install script requires manual reload of udev rules
RUN udevadm control --reload-rules && udevadm trigger; exit 0

# workaround for failing post-install script of libusbdrdaq: enforce exit 0 and re-
configure package
RUN [ "bash", "-c", "echo 'exit 0;' >> /var/lib/dpkg/info/libusbdrdaq.postinst
&& dpkg --configure libusbdrdaq"]
```

Abbildung 32: Workaround im HAL Dockerfile

10. Abbildungsverzeichnis

11. Literaturverzeichnis

[1] watsonic, „Executing periodic actions in Python,“ 19 Januar 2015. [Online]. Available: https://stackoverflow.com/a/28034554. [Zugriff am 06 Dezember 2020].

[2] E. Chen, „Irix Mode vs. Solaris Mode in Top Command,“ 15 Dezember 2012. [Online]. Available: https://logic.edchen.org/irix-mode-vs-solaris-mode-in-top-command/. [Zugriff am 06 Dezember 2020].

[3] GNU.org, „3.7.5 Exit Status,“ [Online]. Available: https://www.gnu.org/software/bash/manual/html_node/Exit-Status.html. [Zugriff am 13 Dezember 2020].

4. Administrative Anhänge

A. Glossar

A

API: Application Programming Interface, Schnittstelle

B

Branch: Einer in Git verwendeter Begriff um einen unabhängigen Codestand zu definieren

C

CI/CD: Continuous Integration and Deployment
CRC: Class-Responsibility-Collaboration
CRUD: Create, Read, Update, Delete

E

ETCS: European Credit Transfer and Accumulation System

F

FURPS: Functionality, Usability, Reliability, Performance, Supportability

H

HAL: Hardware Abstraction Layer

I

IDE: Integrierte Entwicklungsumgebung (Software)
IFS: Institut für Software an der OST
Issue: Auch Arbeitspaket, ein plan- und kontrollierbares Element in einem Softwareprojekt

J

JS: JavaScript Programmiersprache

M

MVP: Minimal Viable Product

N

NFR: Nonfunctional Requirements, dt. nichtfunktionale Anforderungen
npm: Node Package Manager

O

OR-Mapper: Objektorientiert-Relational Mapper
OST: Ostschweizer Fachhochschule

P

Pipeline: Eine Pipeline bezeichnet die Durchführung einer Abfolge von Schritten im CI/CD
Pull Request: Eine in Git verwendete Möglichkeit, um Softwareänderungen an ein Code Repository bekannt zu geben

S

SA: Studienarbeit, 8 ECTS Modul an der OST
Scrum Framework: Ein Kollaborationsframwork um Produkte zu entwickeln
SDK: Software Development Kit
SPA: Single Page Application

U

UMTEC: Institut für Umwelt- und Verfahrenstechnik der Ostschweizer Fachhochschule
Unified Process: Ein iteratives und inkrementelles Software Entwicklungsframework
UTC: Coordinated Universal Time

V

VMRP: Wird als Abkürzung für die Semesterarbeit "Visualisierung von Messdaten mittels Raspberry Pi" verwendet
VS Code: Microsoft Visual Studio Code

W

WSL: Windows Subsystem for Linux