HTL Leonding
next level

HÖHERE TECHNISCHE
BUNDESLEHRANSTALT
LEONDING

Elektroauto Lademanager für die Firma Flexsolution

DIPLOMARBEIT

verfasst im Rahmen der

Reife- und Diplomprüfung

an der

Höheren Abteilung für Medientechnik

Eingereicht von:
Teresa Holzer
Marcel Pouget

Betreuer:
Professor Martin Huemer

Projektpartner:
Alfred Pimminger, Flexsolution GMBH

Leonding, April 2023

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, April 2022 T. Holzer & M. Pouget

Abstract

This is the description of the journey to succeed this project. It took way longer than I thought, it was a lot of work and struggled, but in the end we did it. And if you are reading this right now it means, that everything went exactly as planned. And this is great, because it means that I have my A-Levels done, and don't have to go to this shitty school. Don't worry, I will correct this summary, but right now I don't know what's important, so I write just what comes in my mind. Enjoy it, and pls write me, if you find some mistakes: smalldickenergy@grethat.tu"

Zusammenfassung

Zusammenfassung unserer genialen Arbeit. Auf Deutsch. Das ist das einzige Mal, dass eine Grafik in den Textfluss eingebunden wird. Die gewählte Grafik soll irgendwie eure Arbeit repräsentieren. Das ist ungewöhnlich für eine wissenschaftliche Arbeit aber eine Anforderung der Obrigkeit. *Bitte auf keinen Fall mit der Zusammenfassung verwechseln, die den Abschluss der Arbeit bildet!*

Inhaltsverzeichnis

1 Einleitung

1.1 Herangehensweise

1.2 Zeitplan

1.3 Verwendete Tools

1.3.1 Latex

1.3.2 Intellij

1.3.3 Webstorm

WebStorm wird als Entwicklungsumgebung für JavaScript und JavaScript-ähnliche Sprachen definiert. Es wurde von JetBrains entwickelt und liefert eine Reihe von Hilfestellungen, um JavaScript optimal programmieren zu können. WebStorm beinhaltet eine automatische Code-Überprüfung sowie die automatische Komplettierung von einzelnen Codezeilen. WebStrom beinhaltet außerdem einen bereits eingebauten Run- sowie Debug-Button, durch welchen das Starten in der Konsole wegfällt. Zusätzlich können in WebStorm verschiedenste Plugins hinzugefügt werden, welche das Erscheinungsbild verändern, andere Sprachen überprüfen können und vieles mehr.

1.3.4 Filezilla

1.3.5 Linux Terminal

1.3.6 Discord

1.3.7 Google drive

Google Drive ist ein Cloud-Speicher, welcher für das Online-Speichern von Google Docs, Google Tabellen, Google Präsentationen und Google Formularen verwendet werden kann. Außerdem können Bild- und andere Formate hochgeladen und somit gesichert werden.

Google Drive kann kostenlos durch das Erstellen eines Google-Accounts verwendet werden. Google stellt dabei 15GB Speicher frei zur Verfügung, danach kann durch ein monatliches Abo-Modell mehr Speicherplatz erlangt werden.

1.3.8 vs code

1.3.9 Github

GitHub ist ein Online-Tool, um zusammen mit Teammitgliedern einfach an Projekten zusammenzuarbeiten. Projekte können dabei einfach neu heruntergeladen werden, es kann auf neue Versionen aktualisiert werden, und es können Versionen mit verschiedenen Versionsnummern zusammengefügt werden. GitHub kann über das Terminal oder GitHub Desktop verwendet werden.

2 Umfeldanalyse

2.1 Allgemeiner Technologie-Part

2.1.1 Java

2.1.2 CSV

CSV-Files werden dazu verwendet, eine große Menge von Daten in einem File abzuspeichern. Diese Daten werden mithilfe von Semikolons, Beistrichen oder anderen Trennzeichen geteilt. Durch diese Trennzeichen kann eine CSV-Datei jederzeit übersichtlich in Excel geöffnet werden. Mithilfe von CSV-Files können Daten sehr einfach von einer Anwendung zu einer anderen Anwendung transferiert werden.

2.1.3 HTML/CSS und Javascript

HTML

HTML gilt als Grundgerüst einer Website. HTML besitzt verschiedenste Tags, welche alle über ihren eigenen Nutzen verfügen. Durch diese sogenannten Tags kann der Titel einer Website festgelegt werden, verschiedenste Bilder, Texte und anderer Content hinzugefügt werden. Außerdem bietet HTML die Möglichkeit einer Webseite die passende Struktur zu verleihen.

Um das Strukturieren einer Website zu vereinfachen, gibt es fest zugewiesene Tags. Um die wichtigsten kurz aufzulisten:

<title></title>

- Definiert den Titel eines HTML Dokuments

 <nav></nav>

- Definieren jeweils einen Abschnitt in einem HTML Dokument, nav ist dabei speziell für die Navigationsleiste zuständig

 <h1></h1>

- Sind dafür verantwortlich, dem HTML Dokument Text hinzuzufügen

<img> <video> <audio>

- Mithilfe dieser Tags kann ein/e Bild/Video/Audio-Spur eingefügt werden

Ein Beispiel für ein kurzes HTML-File kann im nächsten Abschnitt gefunden werden. 2

CSS

Mithilfe von CSS können Farben, Formen, Abstände und vieles andere in einer Webseite geändert werden. CSS geht Hand in Hand mit HTML und kann jede Klasse oder auch jeden Tag, der in einem HTML-File vorkommt, ansprechen. Ein paar der wichtigsten Attribute, welche man mit CSS ansprechen und folglich verändern kann, sind: color (Farbe), width und height (Breite und Höhe eines Elements), sowie margin und padding (Abstand eines Elements zu anderen Elementen).

CSS kann wie JavaScript entweder direkt im HTML eingebettet werden 1, oder als externes File angegeben werden. Dies dient wiederum der Übersichtlichkeit und sollte je nach Größe des Projekts entschieden werden.

Listing 1: CSS Einbettung

```
1       <link rel="stylesheet" href="style.css" type="type/css" />
```

JavaScript

JavaScript wurde ursprünglich dafür entwickelt, um als einfache Skriptsprache für kurze Code-Snippets im Browser eingesetzt zu werden. Nach einiger Zeit wurde es allerdings populärer und wurde immer öfters und für längere Codestellen eingesetzt. Webbrowser reagierten darauf sehr positiv und optimierten die Ausführung von JavaScript. Nach dem Erfolg, den JavaScript bei Browsern feierte, wurde es unter anderem auch für node.js und JS-Server verwendet.

JavaScript kann verwendet werden, indem es in ein HTML-Dokument eingebettet wird 2, allerdings ist es ab einer bestimmten Größe des Programms empfehlenswert HTML und JavaScript in verschiedene Files zu verlagern, um die Lesbarkeit zu unterstützen.

Listing 2: HTML mit eingebettetem JavaScript

```
1       <!DOCTYPE html>
2       <html>
3           <head>
4               <title>Hello World!</title>
5           </head>
6           <script>
```

```
7               alert('Hello World!'); // wenn die Seite aufgerufen wird, wird
                    mithilfe von JavaScript ein Alert-Fenster mit 'Hello World!'
                    ausgegeben
8           </script>
9           <body>
10          </body>
11      </html>
```

2.1.4 Typescript

TypeScript ist eine Weiterentwicklung von Javascript, welche einige Vorteile mit sich bringt. TypeScript fügt zum Beispiel zusätzliche Syntax zu JavaScript hinzu, um eine engere Integration mit dem Editor der Wahl zu ermöglichen. Somit können Errors früher erkannt werden. TypeScript Code wird zu JavaScript Code umgewandelt und als dieser ausgeführt, somit läuft es überall, wo auch JavaScript läuft: Dies beinhaltet einen beliebigen Browser, Node.js oder in einer App. Zusätzlich kann JavaScript auch in TypeScript verwendet werden. Ein weiterer großer Vorteil von TypeScript ist, dass es eine automatische Typendeklaration besitzt. Das heißt, dass bei der Erstellung einer Variable kein Typ mitangegeben werden muss, sondern dieser beim Kompilieren des Programms (aufgrund des Inhalts der Variable) selbst zugewiesen wird. 3

Listing 3: TypeScript automatische Zuweisung

```
1       // Der Type String wird hier bei der Kompilation automatisch zugewiesen
2       let helloWorld = "Hello World";
```

2.1.5 Beckhoff

2.1.6 Raspberry

2.1.7 SQL

2.2 Firmenstruktur

2.2.1 Beschreibung der Tätigkeiten der Firma

2.2.2 Flex Tasks

2.2.3 Datenpunkte

3 Wallbox

3.1 Untersuchungsanliegen

Das ist ein Test, um zu schaun, ob es so funktioniert, wie ich mir das vorstelle

3.2 Ist-Stand

Zu der Zeit vor dem Projekt ist es der Firma nicht möglich, Elektro-Fahrzeuge aufzuladen. Um das erreichen zu können, soll auf dem Dach des Firmengeländes eine Photovoltaik Anlage installiert werden, welche mit bis zu 170 kWh die Firma und die neuen E-Autos mit Strom versorgen kann. Die eigens dafür angeschafften Elektroautos sollen mit 5 Wallboxen der Marke 'I-CHARGE CION' beladen werden. Da es bis zu dem Zeitpunkt des Startes des Projekts keine Möglichkeit gab, jene Wallboxen anzusteuern und überwachen zu können, beschäftigt sich dieser Teil der Diplomarbeit mit diesen Themen.

3.3 Aufbau des Projektes

3.3.1 Beschreibung der Funktionen

Das Projekt besteht hauptsächlich aus drei Teilen. Einer Website, auf welcher die Statusanzeigen der einzelnen Wallboxen und andere Funktionen angezeigt werden, einen Charge Controller, welcher dafür verantwortlich ist, die Befehle der GUI entgegenzunehmen und auszuwerten, und aus einem Gateway, welches zwischen den Controllern und dem Modbus-Adapter liegt. Wenn also jemand eine Ladestation einschaltet, schickt die Website (HMI (Human Machine Interface)) über die Flexcloud einen Befehl zu dem Charge Controller. Siehe Aufbau des Projektes: 1.

Dieser ändert dann die Farbe des ön/officons auf der Website, speichert den Wert des Sliders, und schickt diesen dann an das Gateway. Er ist nur dafür verantwortlich, um die jeweiligen Befehle entgegenzunehmen, und an die richtige Wallbox, an das richtige Register mit der richtigen Value zu schreiben.

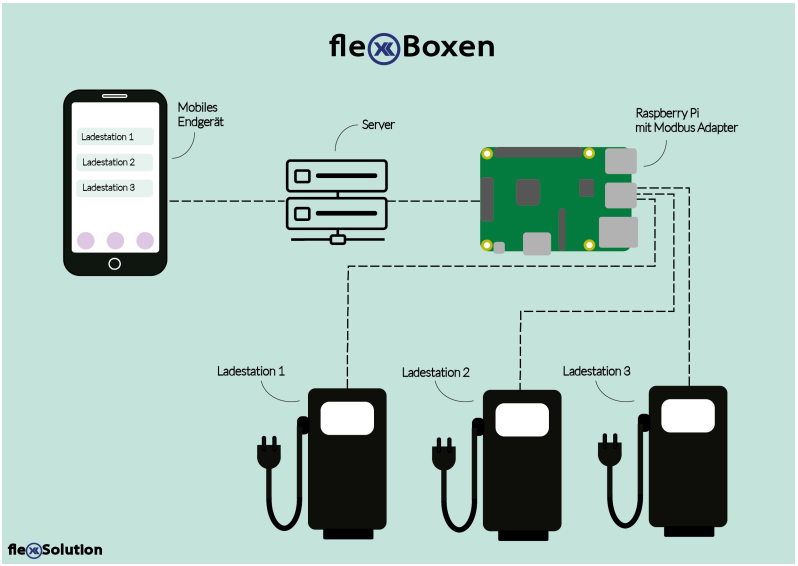


Abbildung 1: Darstellung des Aufbaues

3.3.2 Beschreibung der Wallboxen

Die Wallboxen sind 5 Ladestationen der Marke Ï-Charge". Sie wurden im August 2021 an einer Außenwand der Firma montiert. Jeder Einzelne ist mit 400 Volt an das Stromnetz der Firma gebunden, und kann mit bis zu 32 A bzw. 22 kWh Leistung das Auto laden. Um den Status der Wallbox überwachen zu können, gibt es an der Vorderseite des Panels eine LED-Anzeige mit 5 Punkten, welche in verschiedenen Farben leuchten können.

- grün durchgehend: Station ist frei
- blau blinkend: Station ist besetzt, das Auto lädt aber nicht, weil es entweder voll oder die Wallbox ausgeschaltet ist
- blau durchgehend: Station ist besetzt, das Auto lädt
- rot blinkend: Station ist defekt.

Weitere wichtige Anschlüsse der Stationen sind eine Modbus485-Schnittstelle und einen 5 Volt-Pin. Erstere ist dazu da, um der Box Befehle zu geben und Register, also laufende Werte auszulesen. Mit Zweiterem kann man jede einzelne Wallbox ausschalten, indem man auf den 5 Volt Pin eine Spannung anlegt, und einschalten, wenn die Spannung wieder weggenommen wird. Die Wallboxen sind alle seriell miteinander verbunden, um die Modbus-Kommunikation zu ermöglichen. Dafür wurden jeweils zwei Drähte genutzt, die in eine Wallbox reingehen, an das Modbus-Interface angesteckt wurden und dann wieder aus der einen Wallbox in die nächste Ladestation gelegt wurden. Am Ende der seriellen Leitung befindet sich ein Abschlusswiderstand, um ein möglichst genaues und sauberes Signal zu ermöglichen. Die Kabel der 5 Volt-Leitungen sind alle

parallel geschalten und gehen am anderen Ende in eine Beckhoff-Steuerung. Jene ist dafür ausgelegt, um die Spannung in den Drähten zu verändern. Das Kabel, welches für die Modbus-Kommunikation zuständig ist, ist am anderen Ende mit einem Modbus-zu-USB-Adapter verlötet. Dieser Adapter steckt in einem Raspberry Pi 4, welcher in demselben Schaltschrank montiert ist wie die Beckhoff-Steuerung.

3.3.3 Beschreibung des Gateways

Das Gateway ist ein sogenannter "FlexTask"(später dazu mehr, siehe 1.1), welcher auf dem Raspberry mit dem USB-Adapter läuft. Der Task ist in drei Abschnitte unterteilt. Wenn er gestartet wird, initialisiert dieser zuerst mehrere Arrays, welche die benötigten Datenpunkte beinhalten. Dafür wurden in der Tabelle für jeden Befehl jeweils 5 Datenpunkte gespeichert. Dies ist notwendig, da jeder Befehl auch an jede Wallbox geschickt werden kann, das Gateway aber die einzelnen IDs zuordnen muss. Der Datenpunkt an der Stelle "3ïn einem Befehlsarray repräsentiert den Befehl, welcher an die dritte Wallbox gesendet werden soll. Sobald alle Datenpunkte ohne Fehler angelegt und angemeldet wurden, beginnt der Task damit, eine serielle Verbindung zu dem Adapter aufzubauen. Es werden die Parameter für die Modbus-Verbindung gesetzt und danach wird die Verbindung geöffnet. Für diese Verbindung wird die Java Library "jlibmodbus"verwendet, bei welcher folgende Parameter gesetzt werden müssen.

Wichtige Parameter

- setDevice(/dev/ttyUSB0") -> hier wird angegeben, an welchem Port der Adapter liegt.
- setBaudRate(SerialPort.BaudRate.BAUD_RATE_57600) -> Die Baut-Rate ist ein von der Wallbox vorgegebener Parameter, welcher beschreibt, wie hoch die Baut-Rate ist. Hier wird der Wert 57600 gesetzt.
- setDataBits(8) -> Die Databits müssen 8 Bits betragen, da auch hier der Wallbox-hersteller sich für diese Konfiguration entschieden hat.
- setParity (SerialPort.Parity.NONE) -> Das bedeutet so viel wie, dass es kein signed, also kein Überwachungsbit gibt.
- setStopBits(2) -> Auch dieser Parameter wurde vom Hersteller vorgeschrieben. Die Stopbits werden in einem späteren Kapitel 3.5.4 noch genauer erklärt.

Der erste Teil des Tasks besteht aus einer for-Schleife und einem Codeblock, welcher auf Veränderungen bei den Datenpunkten hört. Als erster Schritt geht die for-Schleife alle

vorhandenen "Devices"(siehe 1.3) durch. In diesem Fall sind es 5, jedes Gerät steht für eine Wallbox. Danach werden für jedes Device die Datenpunkte rausgesucht, und wenn der in der Datenbank gesetzte SSpecificDataType"(siehe 1.4) nicht "nullïst, wird ein sogenannter "DatapointCommandängehängt. Dieser Command ist dafür zuständig, dass Änderungen bei den Werten erkannt und dann in dem oben beschriebenen Codeblock ausgeführt werden. Dort werden dann über die Deviceid, die Specificaddress und den Wert jene Daten entnommen, welche man für das Beschreiben der Modbus-Register benötigt. Dadurch wird ermöglicht, dass der Task, sobald die Value eines Datenpunktes sich ändert, dieser Wert direkt an die Wallbox mit der ID des Geräts gesendet wird.

Der zweite Teil des Tasks ist ein Timer, welcher alle 300 Millisekunden drei verschiedene Register ausliest.

Dazu wird bei jedem Durchlauf an alle Wallboxen ein read-Command geschickt, um folgende Adressen auszulesen:

- 126 -> ist das Register, in welchem der aktuelle Ladestromwert in Ampere gespeichert wird.
- 151 -> ist das Register, in welchem die Ladedauer gespeichert wird.
- 153 -> ist das Register, in welchem die Ansteckdauer gespeichert wird.

Die Werte, welche das Modbus-Protokoll zurückliefert, werden an Datenpunkte übergeben, welche dafür zuständig sind, Werte vom Gateway weiterzuschicken (im Gegensatz zu den vorhererwähnten Datenpunkten, denn diese sind dafür da, um Values von anderen Tasks zu bekommen).

Der dritte Teil benützt nicht so wie die ersten zwei Abschnitte Modbus TCP statt Modbus RTU. Dieser ist wiederum in weitere 2 Teile aufgeteilt. Der eine Part baut eine Verbindung zu dem Controller der PV-Anlage auf, der zweite Part baut auf dieselbe Art eine Verbindung zu den Janitza Messklemmen auf (siehe 1.6). Folgende Beschreibung gilt für beide Verbindungen, es ändern sich nur die Parameter und die Art, mit den Rückgabewerten umzugehen.

Folgende Parameter sind für die Fronius-Verbindung einzustellen:

- tcpParameters.setHost(InetAddress.getByName("10.50.30.200")) -> Hier wird die IP des Modbus-Masters eingegeben
- tcpParameters.setKeepAlive(true) -> Hier wird ein Signal gesetzt, um die Verbindung aufrechtzuerhalten

- tcpParameters.setPort(Modbus.TCP_PORT) -> Standardparameter für den TCP-Port des Modbus (502)
- int slaveId = 1 -> SlaveID des Modbus-Masters
- int offset = 499; -> die Startadresse des Registers, in welchem der aktuelle Stromwert in Watt gespeichert wird
- int quantity = 2; die Anzahl der Register, welche ausgelesen werden sollen. Hier sind es zwei, da in jedem Register nur bis zu 65536 gespeichert werden können. Da aber die PV-Anlage mehr als 65 kW erzeugen kann, müssen hierfür 2 Register zum Speichern verwendet werden. Das erste Register zeigt dabei an, wie oft das zweite Register schon befüllt wurde. Ist also in dem ersten Register eine 1 und im zweiten Register 30000, bedeutet das, dass einmal 65536 + 30000 W produziert werden.

Da die Janitza Klemmen eine andere Art der Persistierung nutzen, musste in dieser Klasse anders mit den zurückgelieferten Werten umgegangen werden. Die Werte der Janitza Klemmen sind als Float abgespeichert, und können aus diesem Grund nicht einfach abgelesen werden. Um trotzdem die richtigen Daten zu bekommen, wurde der zurückgegebenen Value mithilfe der “Integer.toBinaryString(value)” in binäre Zeichen umgewandelt. Mit der Hilfe eines Stringbuilders wurden dann die 2 Register aneinandergehängt. “str.append(Integer.toBinaryString(value));”. Um nun den Binären Wert in eine echte Zahl zurück zu verwandeln, wurde die Funktion “Float.intBitsToFloat” verwendet. Der Wert aus dem String aus dem Stringbuilder wird zu einem UnsignedInt geparsed, und der Wert dann einer temporären Variablen zugewiesen. “tmp = Float.intBitsToFloat(Integer.parseUnsignedInt(str.toString(), 2));”. Da der zurückgegebene Wert aufgrund der Übertragung Fehlern anfällig ist, wird noch überprüft, ob der Wert nicht 0 ist, und kleiner als 200.000, da bei manchen Abfragen die Value nicht stimmt. Die Werte der zwei Modbus TCP-Verbindungen werden alle 300 Millisekunden ausgelesen, und in die Flexcloud gepushed. (Link zu den Fehlern. Beschreiben, wie durch Forum die Art der Umrechnung gefunden wurde. PV und Janitza Klemmen!!!)

Controller

Der sogenannte Chargecontroller ist die Verbindung zwischen dem Graphical User Interface und dem Gateway. Dieser ist für den Logikteil der Anwendung zuständig. In ihm werden die Inputs des Benutzers auf Richtigkeit überprüft, Einheiten umgewandelt, States von Buttons geändert, und User-Feedback generiert. Der Task läuft, genau wie das Gateway, in einem sogenannten Flextask und wird auf einem weiteren Raspberry

initialisiert. Wenn der Task gestartet wird, werden Arrays für die einzelnen Daten angelegt. Jedes Array hat dabei die Länge 0-5, um damit die Wallbox ID zu simulieren. Die Daten der ersten Wallbox, ist somit an der Stelle [1], und nicht [0], so wie es normalerweise der Fall ist. Das ist notwendig, um im späteren Verlauf der Applikation das Ansprechen der Datenpunkte zu vereinfachen.

- Ladedauer -> ist ein Zwischenspeicher, um die Veränderung der Ladedauer zu überprüfen.
- Ansteckdauer -> ist ein Zwischenspeicher, um die Veränderung der Ansteckdauer zu überprüfen.
- wallboxTurnOff -> ermöglicht das Ein- und Ausschalten der Wallboxen.
- Wallboxpriority -> ermöglicht das Wechseln zwischen priorisiertem und normalem Laden.
- PressdWB -> Ein Array aus Booleans, welche alle auf false gesetzt werden. Sobald eine Wallbox eingeschaltet wurde, wird der Wert an der richtigen Stelle auf true gesetzt
- ChangePriority -> Array, in welchem die Variable an der Stelle [x] auf true gesetzt wird, sobald eine Wallbox auf automatisches Laden gestellt wird

Nachdem der Task erfolgreich gestartet ist, werden im Main Thread die Datenpunkte zur Datenübertragung angelegt. Diese bestehen wieder aus einem Array von jeweils 5 Datenpunkte, da man für jede einzelne Wallbox jeden Datenpunkt braucht.

- dpset_charging_active -> ist für das Userfeedback zuständig. Wenn jemand auf einen Button drückt und sich der Status der Wallbox erfolgreich geändert hat, wird mit diesem Datenpunkt in der HMI die Farbe des Buttons geändert.
- DpChangePriorityOfCharching -> Dieser Datenpunkt macht genau dasselbe, nur ist er für die Farbe des Buttons zuständig, welcher das priorisierte Laden aktiviert.
- dpCharging_slider -> Wenn die Wallbox eingeschaltet ist und jemand den Slider für die Stromvorgabe ändert, wird auf diesem Datenpunkt der vom User eingestellte Wert gepublished. Das ist notwendig, um dem Benutzer ein direktes Feedback in der UI zu geben. Er kann dadurch sehen, wie viel Strom er der Wallbox vorgegeben hat.
- wallbox_status -> Dieser Datenpunkt ist für die Farbe des Status-Symbols da. Es gibt drei verschiedene Status:
  - Rot: -> Ladesäule ist besetzt, aber das Auto lädt nicht
  - Blau: -> Ladesäule ist besetzt, und das Auto lädt mit den vorgegebenen Ampere
  - Grau: -> Ladesäule ist frei und kann jederzeit benutzt werden

Diese werden je nach Status der Wallbox aktualisiert. Achtung: Vor allem der Wechsel zwischen "Blauünd "Rot"(in genau dieser Reihenfolge) kann etwas länger dauern, da die Wallbox selbst erst nach einigen Sekunden den Stromfluss zum Auto unterbricht und somit das Laden stoppt.

- wallbox_lädt_mit_kW -> Dieses Array speichert den aktuellen Wert, mit dem die Wallbox gerade lädt. Da bei den E-Autos meist die Kapazität des Akkus in kW/h angegeben ist, wird im Chargecontroller der Wert der Wallbox in Ampere umgerechnet, und mit diesem Datenpunkt zur HMI geschickt. Dient dazu, um dem Benutzer eine Möglichkeit zur Kontrolle zu geben. Achtung! Es ist nicht derselbe Wert wie im "dpCharging_sliderÄrray, da bei letzterem der Wert direkt wieder zur UI geschickt wird, während "wallbox_lädt_mit_kW"der tatsächliche Wert der Wallbox ist!
- wallbox_ladestromvorgabe -> Dieser Datenpunkt ist auch wieder für die Vorgabe des Ladestroms zuständig. Jedoch wird mit diesem Array die Position des Sliders angepasst, sodass selbst nach Verlassen der Oberfläche der Slider immer die zuletzt eingestellte Position besitzt und der Wert wird an das Gateway weitergeleitet. Dafür wird der Wert des Sliders von kW/h in Ampere umgerechnet und gepubilshed
- AktuellerStromWertVonJanitzaForHMI -> Dieser Datenpunkt ist nur zur Kontrolle da. Er dient der Veranschaulichung des Stromverbrauchs in der Firma. Ist der Wert unter 50.000 W, kann das priorisierte Laden nicht aktiviert werden.

Der nächste Abschnitt ist vom Aufbau her ähnlich wie das Gateway. Zuerst wird mit einer For-Schleife über alle Datenpunkte, deren spezifischer Datentyp nicht leer ist, iteriert. An jeden gefundenen Datenpunkt wird dann wieder der "DatapointChangedCommand"gehängt. Das ist dafür da, um auf Änderungen der Werte zu hören. Der nächste Abschnitt der App besteht aus einem "DatapointCommand“, welches ausgeführt wird, sollte die Flexlib eine Änderung der Werte erkennen. Darin ist ein Switch-Statement, welches auf die ID der einzelnen Datenpunkte hört. Sollte also der Datenpunkt mit der ID "ladedauer_wbëinen neuen Wert zugewiesen bekommen, wird der darunterliegende Codeblock ausgeführt. Mit Hilde der Device-ID, welche 1-5 betragen kann, bekommt man die ID der Wallbox, für welche die Änderungen bestimmt sind. Es gibt folgende Datenpunkte-IDs, auf welche gehört wird:

- get_data_from_modbuswb -> Dieser Wert verändert sich, wenn das Gateway eine Veränderung der Ansteckdauer feststellt und diese dem Controller sendet. Wenn

sich die Ansteckdauer nun verändert, wird der Wert in das Array Ästeckdauerän der Stelle [x] (Device ID) gespeichert.

- ladedauer_wb -> Sollte sich der Wert des Datenpunktes mit der Ladedauer verändern, wird der Wert genau wie im oberen Datenpunkt in das Array für die "Ladedauer"gespeichert.
- aktueller_ladestrom_wert_wb -> Das ist der Wert, der sich ändert, wenn das Gateway den aktuellen Ladestrom misst und dem Controller sendet. Dieser wird wieder in W/h umgerechnet und an den Datenpunkt "wallbox_lädt_mit_kW"gepublished.
- cct_get_Ladestromvorgabe_from_hmi -> Sobald die Wallbox eingeschalten ist (die Valie des Icons ist 2) bekommt der Task die Vorgabe des Sliders von der HMI. Da der Slider auf kW eingestellt ist (1-22), wird der Wert des Datenpunktes in Ampere umgerechnet und dann sowohl zur HMI (zur Kontrolle) als auch an den Modbus-Task geschickt. Der Wert wird außerdem in einem Array zwischengespeichert, um ihn beim Einschalten direkt an den Modbus-Task mitzuschicken.
- cct_get_start_chargingcommand_wb -> Dieser Datenpunkt wird dann ausgelöst, wenn ein User den Button zum Starten des Ladevorganges drückt. Bei jedem zweiten Event (ein Klick löst zwei Events aus) wird der Status in dem Array "pressedwb"überprüft. Ist die Variable auf "false", wird das Icon des Buttons auf grün (Value 2) gesetzt, und der aktuelle Wert des Sliders wird "wallbox_ladestromvorgabe"mitgegeben. Außerdem wird die Variable "pressedwbäuf true gesetzt, da die Wallbox nun lädt. Sollte das Icon schon grün sein und somit die Wallbox schon eingeschaltet sein, wird bei einem weiteren Button-Press das Icon auf Rot (aus) gestellt, und "wallbox_ladestromvorgabe"wird auf 0 gesetzt. Das Boolean wird auf false gesetzt.
- ChangePriorityOfCharching -> Hier wird das automatische Laden geregelt. Zuerst wird geschaut, ob das Icon ein bzw. ausgeschaltet ist, dann, ob die Wallbox eingeschaltet ist. Und wenn dann auch noch das Array von true auf true ist, welches überprüft, ob die Janitza Klemmen mehr als 10 kW zurückgeben, dann wird das Auto automatisch geladen. In meinem Fall wird einfach an den Modbus-Task der Wert 32 (A) geschickt, was die maximale Ladegeschwindigkeit ist. Wenn er nicht priorisiert lädt, bekommt der Modbus-task den Wert 10 (A)
- AktuellerStromWertVonPv -> Dieser Datenpunkt ist nur dafür da, um den Wert von dem Gateway, welches den aktuellen Wert der PV schickt, an die HMI weiterzuleiten.
- AktuellerStromWertVonJanitza -> Macht genau dasselbe, nur mit dem Wert der Janitza Messklemmen

Der Controller besteht neben dem Mainthread noch aus einem Timer, welcher alle 1,2 Sekunden einen Codeblock ausführt. In diesem wird geschaut, ob sich die Ladedauer oder die Ansteckdauer im Vergleich zum letzten Durchgang verändert hat. Sollte dies der Fall sein, werden die Farben der Icons angepasst. Außerdem wird überprüft, ob das priorisierte Laden aktiviert werden darf oder nicht. Sollte der Wert der Janitza-Klemmen innerhalb von 30 Wiederholungen nicht unter 10.000 gefallen sein, darf man das Auto priorisiert laden.

HMI / GUI

Der Controller besteht neben dem Mainthread noch aus einem Timer, welcher alle 1,2 Sekunden einen Codeblock ausführt. In diesem wird geschaut, ob sich die Ladedauer oder die Ansteckdauer im Vergleich zum letzten Durchgang verändert hat. Sollte dies der Fall sein, werden die Farben der Icons angepasst. Außerdem wird überprüft, ob das priorisierte Laden aktiviert werden darf oder nicht. Sollte der Wert der Janitza-Klemmen innerhalb von 30 Wiederholungen nicht unter 10.000 gefallen sein, darf man das Auto priorisiert laden.

Listing 4: Example Element

```
1    Test_DynButtonRGBSliderOnOff: {
2      type: "dynButton",
3      items: {
4        label: { id: "name", textContent: "DynButton RGB Slider Ein/Aus"},
5        color: { id: "farbe" },
6        slider: { id: "range" },
7        icon: { id: "symbol" },
8      },
9    },
```

Der Controller besteht neben dem Mainthread noch aus einem Timer, welcher alle 1,2 Sekunden einen Codeblock ausführt. In diesem wird geschaut, ob sich die Ladedauer oder die Ansteckdauer im Vergleich zum letzten Durchgang verändert hat. Sollte dies der Fall sein, werden die Farben der Icons angepasst. Außerdem wird überprüft, ob das priorisierte Laden aktiviert werden darf oder nicht. Sollte der Wert der Janitza-Klemmen innerhalb von 30 Wiederholungen nicht unter 10.000 gefallen sein, darf man das Auto priorisiert laden.

- Der Controller besteht neben dem Mainthread noch aus einem Timer, welcher alle 1,2 Sekunden einen Codeblock ausführt. In diesem wird geschaut, ob sich die Ladedauer oder die Ansteckdauer im Vergleich zum letzten Durchgang verändert hat. Sollte dies der Fall sein, werden die Farben der Icons angepasst. Außerdem wird überprüft, ob das priorisierte Laden aktiviert werden darf oder nicht. Sollte der

Wert der Janitza-Klemmen innerhalb von 30 Wiederholungen nicht unter 10.000 gefallen sein, darf man das Auto priorisiert laden.

- Label: gibt an, welche ID und welchen Content der Button haben soll. Man kann den Content über den Namen des Elements und dem Label ansprechen und verändern.
- Color: Dadurch lässt sich die Farbe des Buttons / Elements verändern.
- Slider: erstellt einen Slider mit einer gewissen Range und mit einstellbaren SSprüngen"
- Icon: Dort kann man Icons von Librarys wie zum Beispiel Font-Awesome anbinden. Die Icons können mit dem Setzen von verschiedenen Werten die Farbe ändern.

Die Oberfläche der App besteht aus einer Hauptseite, wo man eine Übersicht der verschiedenen Ladestationen bekommt. Die Startseite besteht aus einem Icon als Zeichen für die Wallbox, einem Label mit dem Namen der Wallbox, und einem zweiten, dynamischen Icon. Letzteres ist dafür da, um über den aktuellen Status der Wallbox zu informieren.

- Grau -> Station ist frei
- Rot -> Station ist besetzt, aber das Auto lädt nicht
- Blau -> Station ist besetzt, und das Auto lädt

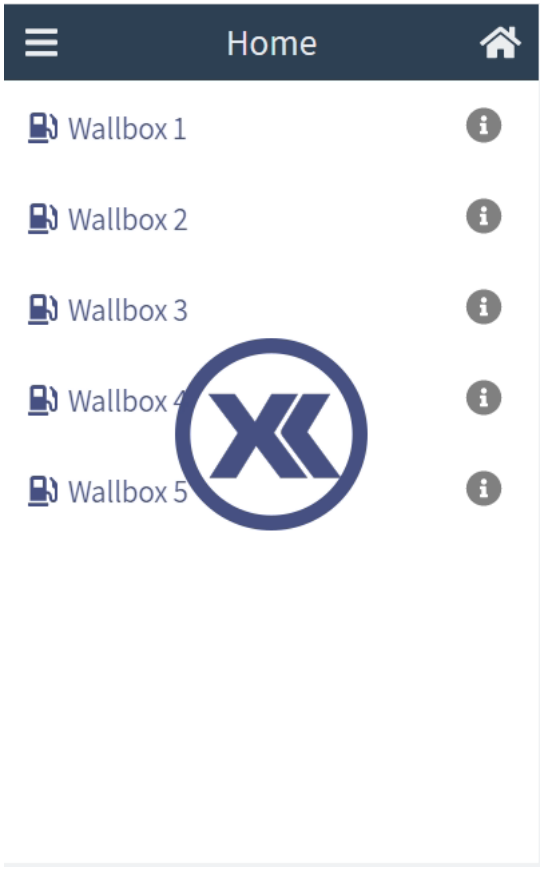


Abbildung 2: Übersicht über die fünf Wallboxen auf der HMI:

Siehe Übersicht über die einzelnen Wallboxen: 2.

Jedes dieser 5 Label ist ein Link zu der Unterseite der jeweiligen Wallbox. Jede der Unterseiten besteht dabei aus einem dynamischen Header mit der ID der ausgewählten Wallbox. Das nächste Element besteht aus einem Label, einem dynamischen Slider und einem Icon, welches zum Ausschalten der jeweiligen Wallbox verwendet wird. Der Slider hat eine Range von 1 - 22, was die möglichen kW veranschaulichen soll. Darunter ist noch ein Statuselement, welches mit dem Icon auf der Startseite gekoppelt ist. Die Soll-Vorgabe, welche sich darunter befindet, ist der Wert, den der User mit dem Slider eingestellt hat. Er ändert sich dynamisch und gibt dem Benutzer die Möglichkeit, die gewünschten kW/h genau einzustellen. Bei Verlassen bzw. Neuladen der Seite wird der Wert natürlich gespeichert und auch der Slider behält seine Position. Die Ist-Vorgabe ist der Wert, welcher bei der Wallbox eingestellt ist und mit welchem das Auto dann wirklich aufgeladen wird. Dieser ändert sich meist mit einer kleinen Verzögerung, da die Wallbox den neuen Wert erst nach wenigen Augenblicken übernimmt. Das Element mit dem Namen Äutomatisches Ladenïst dafür da, um zwischen Solar- und Netzstrom zu schalten. Die Funktion "priorisiertes Laden“, welche in den vorherigen Kapiteln beschrieben wurde, kommt hier zum Einsatz. Die Oberflächen sind für jede Wallbox gleich, nur wird auf jeder Unterseite eine andere Ladestation angesteuert. Die GUI überreicht man nur im Firmen internen Netzwerk über eine URL. Mit der Raute #walli kommt man auf das mobile Template der Anwendung. Die Buttons lassen sich nur mit einem Touch-Screen oder den Entwicklungs-Tools von Chrome betätigen.

Siehe Deteilansicht der Wallbox: 3.

3.4 FlexTasks in der Flexcloud

3.4.1 Was ist die Flexcloud?

Siehe Aufbau der Flexcloud: 4.

Die Flexcloud, auch FlexcommunicationCloud genannt, ist eine Erfindung der Firma FlexSolution. Die Anwendung findet im Firmeneigenen Netzwerk statt, und ermöglicht es, viele kleine Microservices miteinander kommunizieren zu lassen. Die Hauptkomponente, der sogenannte FlexCore, wurde von den Mitarbeitern in der Sprache Java entwickelt. Mit der sogenannten Flexlib lassen sich die Anwendungen mit vielen weiteren kleinen Dependencies erweitern. Das wird vor allem dann benötigt, wenn z.B. eine Datenbankanbindung notwendig ist. Die Flexcloud besteht also aus lauter sogenannten

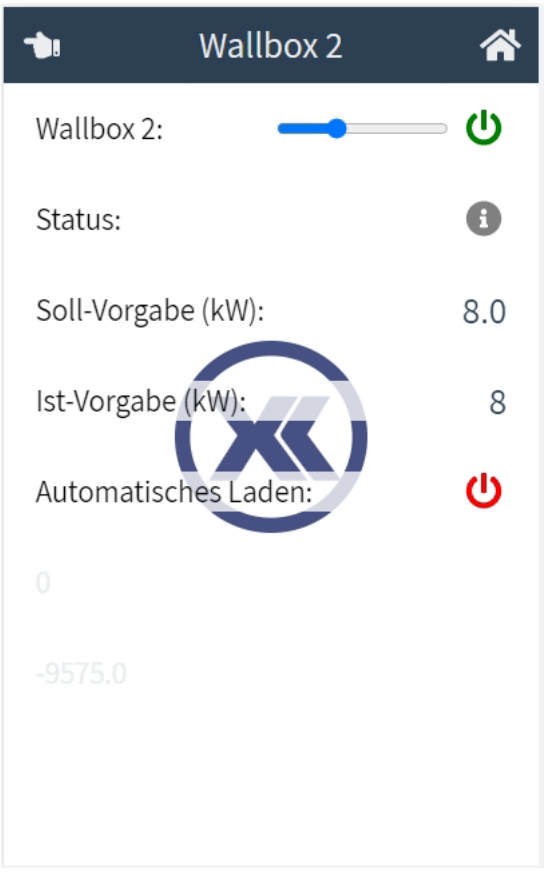


Abbildung 3: Detailansicht der Oberfläche für die Einzelnen Wallboxen

Flextasks, welche verschiedene Aufgaben ausführen. Es zum Beispiel einen Task, der für die Ansteuerung der Lampen und Rollos zuständig ist. Ein weiterer Task, der für die HMI zuständig ist, verbindet die Flexcloud mit dem Browser. Die Flexcommunication-Cloud ist auch für viele kleinere Anwendungen nützlich, da sich solche Microservices sehr schnell aufsetzen lassen. Man kann einzelne Tasks aufsetzen oder mehrere miteinander verbinden und kommunizieren lassen. Das hat den Vorteil, dass man wiederverwendbare Anwendungen entwickeln kann, welche nur grundlegende Funktionen erfüllen, aber von vielen Tasks gebraucht werden. So ist es z.B. in Planung, einen eigenen Modbus-Task zu implementieren, welcher nur dafür da ist, Befehle von anderen Tasks zu übernehmen und an eine USB-Schnittstelle weiterzuleiten. Damit die Tasks untereinander kommunizieren können, hat die Firma die sogenannten Datenpunkte (Datapoints) entwickelt. Sie dienen dazu, um zwischen einem oder mehreren Tasks Informationen oder Werte auszutauschen. Auch Funktionen-Aufrufe können durch solche Datapoints ermöglicht werden. Um durch Datenpunkte miteinander zu kommunizieren, müssen besagte Flex-Tasks mit den Project_ID's verbunden werden. Wie man die Datenpunkte anlegen kann, was dabei wichtig ist und wie die Daten gemapped werden, wird im Kapitel 3.5.1 beschrieben.

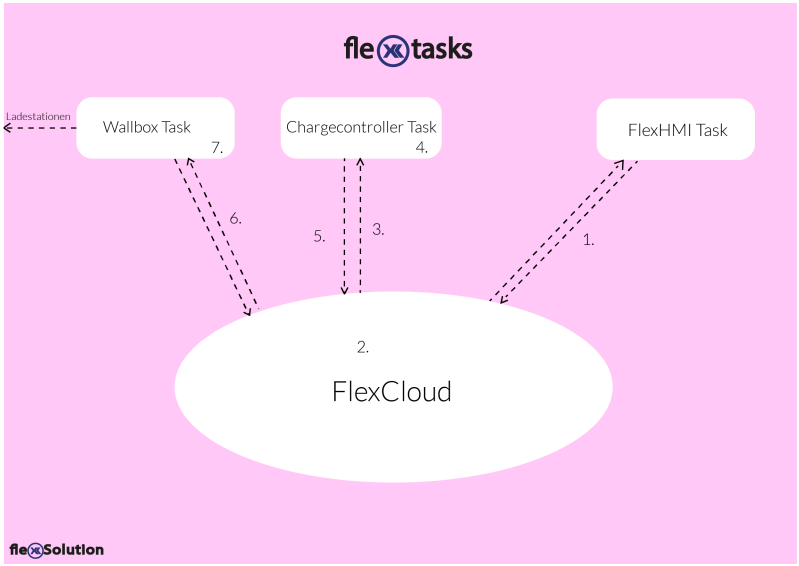


Abbildung 4: Darstellung des Aufbaues

3.4.2 Grundaufbau der Tasks

1. Pom.xml des Tasks editieren: Als erstes sollte die MainClass gesetzt werden. Dies folgt immer einem gewissen Schema. Die Klasse sollte im Package ëu.flexsolution.task.ßu finden sein. Die GroupID ist dieselbe wie das Package der MainClass. In dem Fall ist es wieder ëu.flexsolution.task". In der ärtifactIdßteht der Name des Tasks. Bei den Dependencies wird der "Flexsolution core", "log4j", örg.json". Eine weiter wichtige Sache ist das Einbinden des İnternal Snapshot Repository". Das ist der Ort, an dem der Flexsolution core und die Flexlib gepublished werden. Maven lädt dann beim Starten der Applikation die Dependencies von dem von der Firma bereitgestellten Server herunter, und fügt sie zur Applikation hinzu.
2. Das Anlegen der Main Klasse. Der erste Schritt ist, die Klasse "FlexTaskßu erweitern.

Listing 5: Example Element

```
1          public class FlexTaskModbuswallbox extends FlexTask
```

Dadurch werden einige Funktionen überschrieben, bei denen die meisten jedoch nicht verwendet werden. In der "public static void main"wird ein neues Objekt der Main Klasse erstellt.

Listing 6: Example Element

```
1          new FlexTaskModbuswallbox();
```

In der ïnitTask()"Methode kann dann die Hauptfunktion des Tasks implementiert werden. In den meisten Fällen werden hierfür Datenpunkte und andere Kommunikations-Methoden eingebunden. Mit Fertigstellung der App muss der Code mithilfe von Maven zu einem JAR-File kompiliert werden.

3. Linux-Maschine vorbereiten: Als erster Schritt ist es wichtig, ein Verzeichnis für den Task zu erstellen. In diese wird dann das .jar File kopiert. Da in der Firma mit "log4j"gearbeitet wird, muss hierfür noch ein Konfigurations-File angelegt werden. (siehe log4j File) Da jeder Flextask eine Datenbank mit den für ihn zugehörigen Datenpunkten besitzt, muss dieses File auch erst angelegt und konfiguriert werden: im Verzeichnis '/var/flex/tasks/' wird ein neuer Ordner mit dem Namen des Tasks angelegt. In diesem wird die Datei conf.db erstellt und folgende Tabellen hinzugefügt:

Listing 7: Example Element

```
1          [Service]
2          Type=simple
3          ExecStart=/usr/bin/java -Xmx32m -Dlog4j.configurationFile=./log4j2.xml
               -jar /srv/tasks/CURRENT/modbuswallbox/modbuswallbox.jar
4          Environment="TASKNAME=modbuswallbox"
5          WorkingDirectory=/srv/tasks/CURRENT/modbuswallbox
6          TimeoutStopSec=3
7          Restart=always
8          RestartSec=10
9          Slice=flexTasks.slice
10         [Install]
11         WantedBy=default.target
```

4. Da jeder Flextask eine Datenbank mit den für ihn zugehörigen Datenpunkten besitzt, muss dieses File auch erst angelegt und konfiguriert werden: im Verzeichnis '/var/flex/tasks/' wird ein neuer Ordner mit dem Namen des Tasks angelegt. In diesem wird die Datei conf.db erstellt und folgende Tabellen hinzugefügt:
   a. Device: besteht aus einer ÏDünd einem Label. Im Falle des Gateways sind es 5 Geräte, für jede Wallbox eines. Das ist dafür gut, um Datenpunkte mit denselben Namen nutzen zu können, da man sie mithilfe der Deviceid unterschieden kann
   b. DeviceParameter: Hier können wichtige Parameter gesetzt werden, die dann beim Starten eines Tasks mit der Methode "getNeededDeviceParameters(Map<String, ValueCheck> map)äusgelesen werden können. Die Tabelle besteht aus einer ID, einer DeviceID, einem Wert und einem Label.
   c. taskParameter: hier können wichtige Parameter für das System gesetzt werden. Meistens wird hier nur der Standard-Port 8150 gesetzt. Die Tabelle besteht aus einer Spalte für die ID, für Values und einer für Labels. Auch hier können beim Starten eines Tasks die Werte der DB ausgelesen werden. Hierfür verwendet man die Funktion "getNeededTaskParameters(Map<String, ValueCheck> map)".
   d. datapoint_valueMapping: Besteht aus einer Datapoint1_id und Datapoint2_id, einem Wert und einer Spalte für neue Werte (new_Value). Dies ist dafür da, um zwischen zwei Datenpunkten die Werte zu synchronisieren oder gegebenenfalls mit new_Value zu überschreiben

e. Datapoint: Herzstück der Datenbank. Hier werden die Datenpunkte angelegt 3.5.1. Die Tabelle besteht aus einer ID, einer device_id, einem Label, einer ßpecificAddress", einem ßpecificDatatype", einem ßpecificStruct", einem Datentyp (datatype), Flags, einem Intervall, einem Threshold und einer Value. Für das Starten des Tasks ist diese Tabelle zu befüllen, jedoch ist sie unumgänglich, wenn man die Hauptfunktionen der Flexlib verwenden möchte

f. datapoint_map: Diese Tabelle besteht aus einer datapoint1_id, einer datapoint2_id und einer Funktion. Diese Tabelle wird dazu verwendet, um Zwei Datenpunkte zu mappen (verknüpfen).

g. SystemParameter: Hier werden die wichtigsten Einstellungen für den Task selbst getroffen. Es gibt Einstellungen für den Namen des Tasks, den Port und man kann das keep_Alive Signal einstellen. Außerdem gibt es die PROJECT_ID, welche für das Verbinden mehrerer Tasks notwendig ist. Denn es können sich nur FlexTasks finden, welche dieselbe Project_ID besitzen. Im Falle des Gateways werden die Tasks mit der ID ÄUT, HMI, chargecontroller"verbunden. Das ist wichtig, um den einzelnen Tasks die benötigten Parameter zu übergeben. AUT ist in dem Fall die Haussteuerung, HMI ist der oben kurz beschriebene Flextask zur Veranschaulichung der Elemente und chargecontroller ist der auch schon beschriebene Controller für dieses Projekt.

5. Wenn alle Pfade richtig benannt wurden, kann man im Verzeichnis einen symbolischen Link zu dem .jar-File machen. Das hilft bei der Entwicklung, da man bei einer Namensänderung nur den Link anpassen muss und nicht die angegebenen Pfade in den Konfigurations-Dateien.
6. Starten, stoppen bzw. überwachen kann man den Task mit ßystemctl –user start / stop / restart / status FlextasName.service
7. Mit dem Command "tail -f /tmp/log/ FlextaskName.log"kann man in Echtzeit mitansehen, was Log4j in das File schreibt. Die Files werden alle in dem Verzeichnis /tmp/log/"gespeichert, wenn sie nach einiger Zeit nicht mehr benötigt werden, werden sie komprimiert und in ein anderes Verzeichnis verschoben, um Platz zu sparen.

3.5 Datenpunkte

3.5.1 Struktur eines Datenpunktes

Die Datenpunkte sind im Grunde alle gleich aufgebaut. Ein Datenpunkt (dp) besteht immer aus einer:

- ID: kann nach Belieben benannt werden. Meist ein String, um den Nutzen zu veranschaulichen und die DP´s voneinander unterscheiden zu können. Die ID muss für jedes Device eindeutig sein.
- device_id: gibt an, zu welchem Device der Datenpunt gehört. Ein Device kann in dem config.db-File hinzugefügt werden. Das dient dazu, um mehrere Datenpunkte mit derselben ID für mehrere Devices verwenden zu können. Die device_id besteht immer aus einem INT. Um Datenpunkte nutzen zu können, muss mindestens ein Device angelegt werden. Die ID wird automatisch immer um eins erhöht. Im Normalfall ist schon ein Device mit der ID 1 und dem Label dev1 angelegt.
- Label: Hier kann man den Datenpunkt kurz beschreiben, um seine Funktion zu dokumentieren. Das wird dafür verwendet, um den Entwicklern das Verwalten der Datenpunkte zu vereinfachen.
- SpecificAddress: Ist meistens derselbe String wie die ID, kann jedoch auch etwas anderes sein. Wird dazu benötigt, um den Datenpunkt im Task anzulegen.
- SpecificDatatype: Hier kann ein String mitgegeben werden, um in einem Task einen Datentyp für den Datenpunkt festzulegen, bzw. zwischen Gruppen von Dp zu unterscheiden. Im Task für das Gateway wird der specificDatatype dafür genutzt, um zwischen Datenpunkten, die nur für Modbus zuständig sind, und den anderen Datenpunkten zu unterscheiden.
- Datatype: hier kann man angeben, welchen Datentyp die Value haben soll. Man kann hier zwischen ÏNT", "REAL", "BOOLünd "CMD"wählen.
- Intervall: Hier kann man ein Intervall angeben, mit welchem der Datenpunkt in die Flexcloud gepusht werden soll. Standardmäßig ist –1 eingestellt, was bedeutet, dass der Datenpunkt nur nach Anfrage des Tasks bzw. bei Änderung der Value gepublished wird.
- Value: Hier werden die Werte eines Datenpunktes gespeichert. Man kann in der conf.db Datei einen Standardwert angeben, doch meistens werden diese Werte in einem Task gesetzt bzw. ausgelesen.

3.5.2 Anlegen eines Datenpunktes

Um einen neuen Datenpunk in einem Task anzulegen, muss man sich zuerst auf die VM (Virtual machine), auf welcher der Flextask sich befindet, verbinden. In dem Verzeichnis mit der conf.db Datei muss man mit SQLITE3 die Datei öffnen. Dort kann man dann mit einem insert bzw. Update Statement einen neuen Datenpunkt hinzufügen, aktualisieren oder löschen.

Listing 8: Example Element

```
1       INSERT INTO datapoint VALUES ('Wallbox\textunderscore 1\textunderscore
            startCharching\textunderscore icon\textunderscore
            state',1,'','state\textunderscore [Wallbox\textunderscore 1\textunderscore
            startCharching\textunderscore icon]','','','INT',-1,-1,0.0,'');
```

3.5.3 Nutzung eines Datenpunktes:

Somit ist der Datenpunkt in der Datenbank angelegt und ready to use. Um nun in einem Flextask auf so einen Datenpunkt Zugriff zu haben, muss man ihn erstmal anlegen und die “SPECIFIC_ADDRESS” angeben:

Listing 9: Example Datapoint

```
1       Datapoint datapoint =
            StaticData.datapoints.getDatapoint(Datapoints.DatapointField.SPECIFIC\textunderscore
            ADDRESS, "SPECIFIC\textunderscore ADDRESS\textunderscore
            DES\textunderscore DATENPUNKTES");
```

Nun hat man mehrere Möglichkeiten, wie man Daten aus dem Datenpunkt lesen kann:

1. Man ruft die Methode “setDatapointChangedCommand()” auf. Diese Methode überschreibt eine in der Klasse Flextask, und ist dafür da, um auf Änderungen des Datenpunktes zu hören.

Listing 10: Example catapoint usage

```
1           datapoint.setDatapointChangedCommand(new DatapointCommand() {
2               @Override
3               public DatapointResponse execute(DatapointResult result) {
4
5                   System.out.println(datapoint.getValue());
6                    return new
                         DatapointResponse(DatapointResponse.ResponseCode.OK);
7               }
8           });
```

Diese Methode gibt bei jeder Änderung des Wertes den neuen Wert in der Konsole aus. Hier kann man den Wert speichern, andere Methoden ausführen, oder sonst alles machen, was man in einer Methode machen könnte.

2. Man kann jederzeit im Code "datapoint.getValue()äufrufen. Mit dieser Methode bekommt man immer den jetzigen Wert des Datenpunkts zurück.
3. Wie schon in den vorhergegangenen Kapiteln erwähnt, kann man über alle Datenpunkte jedes Device iterieren, und je nach Bedingung ein DatapointCahngedCommand anhängen. Dies bewirkt, dass eine Methode immer dann aufgerufen wird, wenn sich einer der Datenpunkte ändert. Mit der ID der einzelnen Devices kann man dann auf die SpecificAddress zugreifen, und die einzelnen Fälle mit einem switch Statement abdecken.

Listing 11: Example multible datapoint usage

```
1           for (Device dev : StaticData.devices.getDevices()) {
2       for (Datapoint dp : StaticData.datapoints.getDatapoints(dev.getId())) {
3           if (!dp.getSpecificDataType().isEmpty()) {
4               dp.setDatapointChangedCommand(dpCmd);                  //!!!
5           }
6       }
7   }
8
9
10
11  DatapointCommand dpCmd = new DatapointCommand() {
12          @Override
13      public DatapointResponse execute(DatapointResult datapointResult) {
14
15          int deviceID= datapointResult.getDeviceId();
16          switch (datapointResult.getDpId()) {
17              case "datapoint\textunderscore example\textunderscore id":
18                  System.out.println(datapointResult.getValue() + " " + deviceID
                        );
19                   break;
20                      }
21          return new DatapointResponse(DatapointResponse.ResponseCode.OK);
22      }
23  };
```

In diesem Beispiel werden die DeviceID und der aktuelle Wert des Datenpunktes, welcher sich geändert hat, ausgegeben.

Mit “datapoint.setValue(value)” kann man einem Datenpunkt einen neuen Wert zuweisen. Dieser wird dann automatisch in die FLexcloud gepublished, und kann von einem anderen Task abgefangen werden.

3.5.4 Datapoint mapping

Um zwei Datenpunkte von zwei verschiedenen Tasks zu mappen, und somit Daten zu übertragen, muss man einen Eintrag in der “datapoint_map” hinzufügen. Um die Daten zu empfangen, muss man die conf.db des zweiten Tasks aktualisieren. Zu beachten ist, dass das Mapping nur bei dem Task gemacht werden muss, welcher die Daten bekommen soll. Man verbindet also den Datenpunkt eines anderen Tasks auf einen eigenen Datenpunkt. Wichtig dabei ist, dass man bei der “datapoint1_id” zuerst das Label des Devices, zu welchem der Datenpunkt gehört, und anschließend die ID des

ersten Datenpunkt angibt. “datapoint2_id” wird zuerst der Name des anderen Tasks angegeben, dann das Label des Devices, und anschließend die ID des Datenpunktes.

Beispiel: Ich möchte von dem Flextask “modbuswallbox”, welcher in dem Projekt als das Gateway bezeichnet wurde, den Wert der Ladedauer der ersten Wallbox zum “chargecontroller” schicken. In dem Task “modbuswallbox” muss dafür ein Device mit dem Label “wb1” und ein Datenpunkt mit der ID “ladedauer_wb1” angelegt werden. Wird nun im Code auf diesen Datenpunkt was gepublished, sieht man “modbuswallbox.wb1.ladedauer_wb1”. Damit die Daten jetzt auch ankommen, muss im Task für den Charge-controller ein Datenpunkt mit der ID “ladedauer_wb” angelegt werden. In die “datapoint_map” Tabelle muss dann bei der “datapoint1_id” das Device und der Datenpunkt angegeben werden, und bei der “datapoint2_id” zuerst der Task Name, das Device und dann der Datenpunkt, der die Daten pulished.

Listing 12: Example Element

```
1      dev1.ladedauer_wb                    modbuswallbox.wb1.ladedauer_wb1
```

3.6 Modbus

3.6.1 7 Schichten OSI-Modell

Um die Art der Kommunikation zwischen den Wallboxen, der PV-Anlagen, den Janitza Klemmen und dem Raspberry PI zu verstehen, muss man sich erstmals mit der Übertragung an sich beschäftigen. Denn mit dem Open System Interconnection Reference Model (kurz OSI-Referenzmodell) lässt sich am besten beschreiben, wie und auf welchen Ebenen die Daten übertragen werden. Kurz zusammengefasst ist es ein Modell, das die Kommunikation in einem Netzwerk in sieben Schichten unterteilt. Die Entwicklung, welche seit 1983 als Standard veröffentlicht wird, begann bereits 1977. Der Fokus der Entwicklung war es, ein Modell zu erschaffen, mit welchem sich die Kommunikation zwischen Systemen, auch innerhalb eines Netzwerkes, beschreiben lassen. Es gibt dabei sieben Schichten, auf welche sich unterschiedliche Übertragungsarten abspielen.

Die sieben Schichten des OSI-Modells sind:

1. Die Bitübertragungsschicht ist die unterste Schicht. Die physische Schicht umfasst die Hardware, die zur Übertragung von Daten über das Netzwerk verwendet wird, wie zum Beispiel verschiedene Kabel (wie Netzwerkkabel, Serielle Verbindungen), Hubs, Switches und Router. Außerhalb eines Netzwerkes könnte die physische Schicht

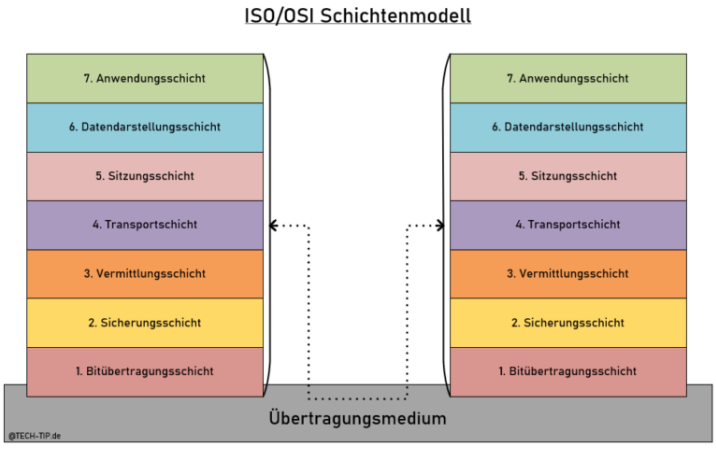
auch der Rauch sein, mit dem Rauchzeichen übermittelt werden, oder die Luft, wenn es um Morsezeichen geht. Im Grunde alles, durch was sich verschlüsselte Nachrichten übermitteln lassen.

2. Die Datenübertragungsschicht ist für die Übertragung von Datenpaketen von einem Gerät zu einem anderen verantwortlich und verwendet MAC-Adressen, um Geräte im Netzwerk zu identifizieren. [1] Die MAC-Adresse besteht aus 48 Bit, was sechs Bytes entspricht. Im Unterschied zur IP-Adresse wird die MAC-Adresse in hexadezimaler Form dargestellt. Ein Beispiel dafür wäre: 00-1D-60-4A-8C-CB. Es werden immer zwei Bytes zusammengeschrieben. Die jeweiligen Paare werden durch einen Bindestrich oder in manchen Fällen durch einen Doppelpunkt getrennt. Hinter den ersten drei Bytes verbirgt sich die Kennung des Herstellers (OUI). Die verbliebenen drei Bytes werden vom Hersteller beliebig vergeben. Auf diese Weise wird sichergestellt, dass sich die MAC-Adressen der verschiedenen Hersteller voneinander unterscheiden. Es bedeutet aber auch, dass sich jeder Hersteller ein entsprechendes Schema überlegen muss, damit die letzten drei Bytes einer MAC-Adresse nicht versehentlich doppelt vergeben werden.
3. Die Netzwerkschicht ist für die Vermittlung von Datenpaketen zwischen Netzwerken verantwortlich und verwendet IP-Adressen, um Geräte im Internet zu identifizieren. [2] Die klassischen IP4 Adressen bestehen, wie der Name vermuten lässt, aus 4 Bytes und einer Subnetmask. Diese ist dafür da, um die Adresse in zwei Teile zu unterteilen. Der vordere Teil repräsentiert dabei immer den Netzwerkteil und der hintere die Computer ID. Für private Netzwerke gibt es reservierte Bereiche. 10.0.0.0/8, was vor allem in großen Firmen benötigt wird, da es 16.777.216 mögliche Adressen gibt. Im Heimnetz sieht man vor allem 172.16.0.0/12 und 192.168.0.0/16. Bei diesen IPS kann man in einem Netzwerk viel weniger Geräte benutzen. Bei der 172.16.0.0/12 sind es immerhin noch 1.048.576 mögliche Geräte, und bei der letzten Möglichkeit nur noch 65.536. Deswegen findet man in privaten Haushalten diese am häufigsten. Ein klassisches Beispiel wäre: 192.168.0.100/16.
4. Die Transportschicht ist für die Übertragung von Daten zwischen Endgeräten verantwortlich und sorgt dafür, dass die Übertragung fehlerfrei erfolgt. Auf dieser Schicht kommen vor allem die Protokolle TCP und UDP zum Einsatz. TCP ist dafür da, um mit Hilfe einer Prüfsumme zu gewährleisten, dass alle Daten richtig bei dem Empfänger ankommen. UDP hingegen ist ein auf Geschwindigkeit angelegtes Protokoll, bei dem es auch vorkommen darf, wenn ein paar Bits verloren gehen,

5. Die Sitzungsschicht ist für die Verbindungsaufnahme und -beendigung zwischen Endgeräten verantwortlich und sorgt dafür, dass die Übertragung ordnungsgemäß abläuft.
6. Die Darstellungsschicht ist für die Umwandlung von Daten zuständig. Die Daten werden in ein für die Übertragung geeignetes Format umgewandelt, und sorgt dafür, dass die Daten in der richtigen Form, also richtig formatiert, an ihr Ziel gelangen. Auch Kompression und Verschlüsslungen gehören zu dieser Schicht.
7. Die Anwendungsschicht ist die oberste Schicht und ist dafür verantwortlich, um mit den Anwendungen kommunizieren zu können. Diese werden auf den Endgeräten ausgeführt. Über diese Schicht wird die Verbindung zu den unteren Schichten hergestellt. Hier werden vor allem Applikationen verwendet, mit denen man Daten austauschen kann. Typische Beispiele sind Webbrowser, E-Mail-Programme oder Nachrichtendienste.

[3]

Das OSI-Referenzmodell hilft, die Kommunikation in einem Netzwerk zu beschreiben und zu verstehen, indem es die verschiedenen Aspekte der Netzwerkkommunikation in logische Schichten unterteilt. Dies ermöglicht es, Netzwerkprotokolle zu standardisieren und die Kompatibilität von Geräten und Software in verschiedenen Netzwerken zu gewährleisten.



[4]

Abbildung 5: Das 7 Schichten OSI/ISO Modell

3.6.2 Modbus Protokoll

Modbus ist ein Protokoll, welches 1979 entwickelt wurde, und vor allem in der Industrie große Bedeutung hat. Es ermöglicht eine Kommunikation zwischen automatisierten

Maschinen, und wurde damals als Protokoll auf der Anwendungsebene implementiert, das Daten über die serielle Schicht übertragen soll. Das Protokoll gehört mittlerweile zu den Industriestandards, und wird vor allem in der Automatisierungstechnik gerne benützt. Mittlerweile hat sich Modbus weiterentwickelt, so dass es mehrere Implementierungen gibt. Der Datenaustausch ist mittlerweile über TCP/IP, seriell oder über das "User-datagram-protocol", kurz UDP, möglich. [5] [6]

3.6.3 Was genau ist jetzt das Modbus-Protokol?

Modbus ist ein sogenanntes request-response Protokoll, welches eine "Master-slave" Beziehung nutzt. In so einer "Master-slave" Beziehung funktioniert die Kommunikation immer paarweise – ein Gerät schickt eine Anfrage an einen "Slave", und wartet auf eine Antwort. Der Master ist dabei für jede Interaktion verantwortlich. Der Master ist normalerweise ein einfaches HMI (human machine interface), oder ein "Supervisory Control and Data Acquisition (SCADA) system". Der Slave ist meistens ein Sensor, eine Messklemme, ein programmable logic controller (PLC) oder ein Interface. Der Inhalt der Anfragen und Antworten sowie die Netzwerkschichten, über die diese Nachrichten gesendet werden, werden von den verschiedenen Schichten des Protokolls definiert. [7]

3.6.4 Die verschiedenen Layer des Modbus Protokoll

In der ersten Implementierung war Modbus ein einzelnes Protokoll, welches auf Serielle Protokolle aufbaut, es konnte also nicht in mehreren Schichten aufgeteilt werden. (Modbus RTU) Über einen längeren Zeitraum wurden neue Implementierungen vorgestellt, um entweder das serielle Paketformat zu ändern oder die Verwendung von TCP/IP- und UDP-Netzwerken (User Datagram Protocol) zu ermöglichen. TCP, Remote Terminal Unit (RTU) und ASCII sind die drei am häufigsten verwendeten ADU-Formate. RTU- und ASCII-ADUs werden traditionell über eine serielle Verbindung verwendet, während TCP über zeitgenössische TCP/IP- oder UDP-IP-Netzwerke verwendet wird.

3.6.5 Wie funktioniert Modbus RTU, und wie ist das Protokoll aufgebaut?

Am Anfang ist es wichtig zu erwähnen, dass jeder einzelne Teilnehmer (Slave) eine sogenannte Slave Id besitzt. Diese muss in einem "Netzwerk" eindeutig identifizierbar

sein, und kann je nach Hersteller fest vergeben, automatisch konfiguriert oder frei wählbar sein. Die Adresse 0 ist für den Master reserviert. Die einzelnen Teilnehmer können die Adressen von 1 bis 247 annehmen, da die Adressen von 248 bis 255 reserviert sind. Die Daten von den Slaves sind in sogenannte Register gespeichert. Diese sind immer 16 Bit groß.

Das Protokoll Modbus überträgt die Daten in binärer Form. Das führt dazu, dass man die Daten nicht direkt auswerden kann, sondern eine Möglichkeit des Parsens braucht, um auf die ursprünglichen Werte zu kommen. Vorteil ist jedoch, dass dadurch eine großer Datendurchsatz ermöglicht wird.

Soll nun ein Paket über Modbus geschickt werden, muss man zuerst ein paar variablen Einstellungen treffen. So kann die Baudrate (Bitrate) oft frei gewählt werden (solange sie nicht von dem Hersteller festgelegt wurde, wie es bei den Wallboxen der Fall war). Außerdem gibt es sogenannte “Stopbits”, welche bei jedem Hersteller anders konfiguriert sind. Die Defaultwerte sind dafür 1 oder 2 Bits.

Vor und nach jedem gesendeten Paket gibt es eine Sendepause (Wartezeit) von mindestens 3.5 Zeichen. Da ein Zeichen eine Länge von 11 Bit besitzt, hängt die Wartezeit von der Bitrate ab. Dabei ist zu beachten, dass vor allem bei einer niedrigen Übertragungsrate es sehr wichtig ist, diese Zeit genau einzuhalten, da es sonst zu Überschneidungen und zu Datenverlusten kommen würde.

Nach der Pause fängt das Packet mit der Adresse des jeweiligen Slaves an. So kann jeder Teilnehmer direkt auf sein Packt reagieren. Bei jeder Antwort wird die Adresse zurückgesendet, damit der Master das Packet zuordnen kann. Die Adresse ist immer 8 Bits lange, und kann somit 256 Zustände einnehmen (ausgeschlossen sind die oben beschriebenen vordefinierten Adressen)

Das nächste Byte enthält die Information über die Funktion. Folgende sind in der Produktion wichtig: **??**

[8]

Bei der Kommunikation mit den Wallboxen wurde nur die Funktion 4 und 6 benutzt, um die Register auszulesen und neu zu beschreiben.

Nach der Funktion kommen die eigentlichen Daten des Paketes. Diese bestehen immer aus einem Register. Je nach Funktion kommen noch andere Werte hinzu, zum Beispiel der Wert, den man auf ein Register schreiben möchte. Ein Register lässt sich hier sehr

gut mit einer Adresse in dem jeweiligen Device vergleichen. Dort stehen nämlich die Daten des Gerätes bzw. dort können Werte gesetzt werden. Die Daten können dabei beliebig groß sein.

Der CR-Check am Ende ist eine Prüfsumme des Paketes, um die Gültigkeit der Daten zu überprüfen. Diese Überprüfung ist vor allem bei der seriellen Kommunikation sehr wichtig, da es immer wieder zu Differenzen kommen kann, wo Daten verloren gehen und das Paket unvollständig ankommt. **??**

3.6.6 Wie funktioniert Modbus TCP/IP, und wie ist das Protokoll aufgebaut? Was sind die Unterschiede zu RTU?

Die Modbus-Kommunikation über TCP ist der über RTU sehr ähnlich. Der größte Unterschied ist, wie der Name schon erraten lässt, hier wird das Packet über TCP verschickt. Die ganze Datenübertragung läuft über den Port 502, und der Master ist immer über eine IP-Adresse erreichbar. Es gibt noch folgende Unterschiede am Packet selber:

Jedes Paket hat eine Transaktionsnummer, die immer 16 Bits lang ist. Danach kommt, mit derselben Länge, ein Protokollkennzeichen, welches immer gleich ist (0x0000). Der nächste Abschnitt beschreibt, wie viele Bytes noch folgen werden. Dieser Bereich ist immer um 2 Byte größer als die tatsächliche Anzahl der Bytes. Die Länge der Adresse, der Funktion und der Daten ist dieselbe wie bei Modbus TCP. **??**

3.6.7 Modbus RS485 vs RS232

Es gibt bei Modbus RTU noch weitere Unterteilungen, welche sich in der physikalischen Ebene unterscheiden. Die größten Unterschiede sind dabei die Länge der Kabel, welche verwendet werden, die Art, wie die Teilnehmer mit dem Master verbunden sind, und der Pegel, welcher in den Leitungen herrscht.

Bei Modbus 485 ist aufgrund der physischen Bauweise eine viel größere Länge der Kabel möglich. Während bei Modbus 232 meist nur eine Kabellänge von 10-15 Metern wirklich gut funktioniert, kann bei RS485 eine Länge von bis zu 1200 Metern erreicht werden.

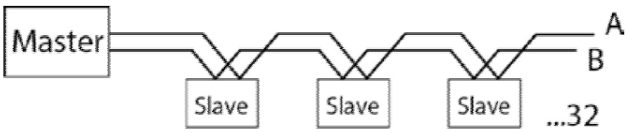
Code	Description
1	Read coils
2	Read discrete Inputs
3	Read discrete Inputs
4	Read Input Registers
5	Write single Coil
6	Wirte single Register
7	Read Exception Status (nur für serielle Übertragung)

Start	Adresse	Funktion	Daten	CR-Check	Ende
Wartezeit (min. 3.5 Zeichen)	1 Byte	1 Byte	n Byte	2 Byte	Wartezeit (min 3.5 Ze

Transaktionsnummer	Protokollkennzeichen	Zahl der noch folgenden Bytes	Adresse	Fun
2 Byte	2 Byte (immer 0x0000)	2 Byte (n + 2)	1 Byte	1 B

Auch der Signalpegel ist ein großer und wichtiger Unterschied. Während RS485 differentielle Signale nutzt, die zwischen positiven und negativen Spannungen schwanken, nutzt RS232 eine Spannung von 3V.

Ein weiterer wichtiger Punkt ist die Verkabelung der einzelnen Teilnehmer. Während bei Modbus RS232 eine Punkt zu Punkt Verkabelung verwendet wird, sind die Slaves bei RS484 alle an denselben zwei Kabeln angeschlossen (siehe Abbildung Bild). Ein weiterer, wichtiger Punkt ist der Abschlusswiderstand, den man am Ende jeder Leitung braucht. Dieser dient dazu, um Interferenzen zu verringern, und so Störungen im System zu eliminieren. [9]



[10]

Abbildung 6: Aufau eines Modbus-Netzwerkes

3.6.8 Modbus tools + libraries

Es gibt viele tools, mit denen man über einen USB zu Modbus Konvertern Modbus-Register auslesen kann. Für diese Arbeit wurden hauptsächlich die Programme “QModMaster” und “Hercules” verwendet. Es handelt sich bei beiden um kostenlose Programme. 7

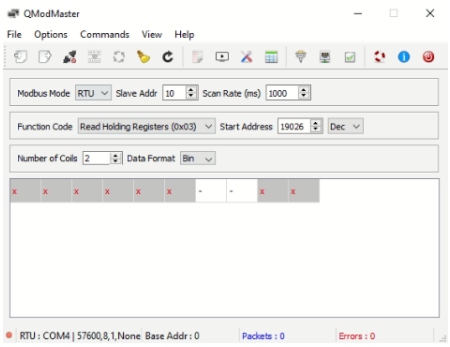


Abbildung 7: Overview von QModMaster

In QModMaster kann man beim Modbus-Modus zwischen dem Modus RTU und TCP wechseln, da das Programm beides unterstützt. Unter der Slave Adresse kann man den gewünschten Teilnehmer auswählen. Hier ist es in dem Beispiel der Slave mit der ID 10. Mit der Scan-Rate kann man ein Intervall einstellen, mit welchem das Tool den Port abfragt.

Bei dem Abschnitt "Function Code"kann man auswählen, welche Funktion [rev Funktionstabelle] ausgeführt werden soll. Die Start-Adresse ist das Register, welches beschrieben oder ausgelesen werden soll. Hier kann man auch noch angeben, auf welche Art die Daten übermittelt werden sollen. Es gibt die Möglichkeit für Binär-, Dezimal- und Hexadecimal-Zahlen.

Die "Number of Coilsßagt an, wie viele Register auf einmal ausgelesen / beschrieben werden sollen, da bei manchen Geräten die Werte für ein Register zu groß ist (über 65536). In solchen Fällen werden die Daten je nach Hersteller in 2 oder mehreren Registern abgespeichert. Sollte dies der Fall sein, ist es aber immer vom Hersteller dokumentiert.

Der untere Bereich ist das Ergebnis, welches der Slave zurücksendet. Hier kann man auch auswählen, in welcher Form die Daten angezeigt werden sollen. Ganz unten sieht man noch eine Zusammenfassung der Einstellungen, wie viele Packages gesendet wurden und wie viele davon fehlerhaft waren.

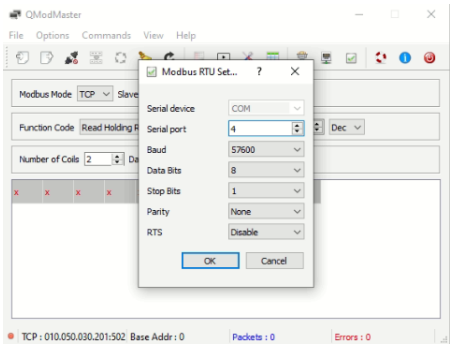


Abbildung 8: Einstellungen für Modbus RTU

8 Hier kann man die Werte der Parameter einstellen. Das “Serial-device” ist in diesem Fall der USB-Port des Computers, und der “Serial-port” ist der Steckplatz, in welchem der Modbus-Konverter steckt. Falls man den Port des Adapters nicht kennt, findet man diesen in Windows unter den verbundenen Geräten im Gerätemanager.

“Baud” ist die Konfiguration für die Baudrate. Diese muss mit den Angaben des Herstellers übereinstimmen.

Data-Bits beschreibt die Länge des Protokolls. Dieser Wert ist auch im Datenblatt des jeweiligen Gerätes zu finden.

Die Stop-bits findet man auch in den Angaben des Herstellers. Der Wert beträgt meist 1 oder 2 Bits. Parity und RTS sind meist disabled, können aber bei Bedarf auch konfiguriert werden.

Außerdem gibt es einen Monitor, auf welchem man genau die einzelnen Abfragen des Tools nachvollziehen, und abspeichern kann. Auf der 9 findet man noch weitere Tools, die man in Verbindung mit Modbus verwenden kann.

Wenn man mit QModbusMaster eine Modbus TCP Verbindung aufbauen möchte, muss man den Mode auf TCP stellen. Die Einstellmöglichkeiten sind fast identisch zu der Modbus RTU Verbindung, hier muss man lediglich die Unit ID anstatt der SlaveID einstellen, jedoch ist es derselbe Wert. 10

3.7 FlexHMI

3.8 Energiemanagement

3.9 Wallboxen

3.9.1 Java

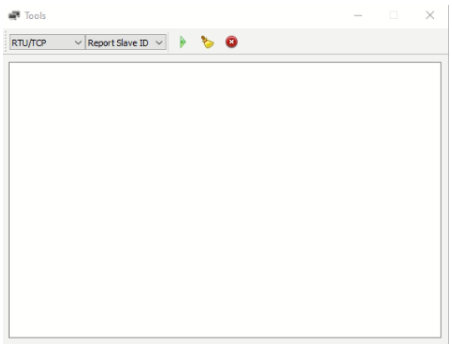


Abbildung 9: Einstellungen für Modbus RTU

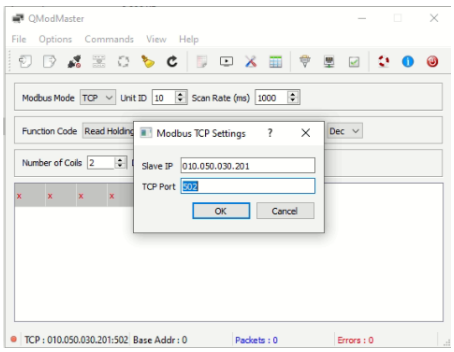


Abbildung 10: Einstellungen für Modbus RTU

4 Flexlogger

4.1 Untersuchungsanliegen

4.2 IST-Stand

Die Firma FlexSolution befindet sich in der Lage, die benötigten Daten zu loggen (abzuspeichern) mithilfe von Log4J. Allerdings werden dabei die Daten zuerst in einer Datei gespeichert, darauffolgend komprimiert und in einem Ordner abgelegt. Nachdem diese Daten komprimiert wurden, gibt es nur sehr umständliche Möglichkeiten, die Daten auszulesen, da diese nicht zentral abgespeichert werden und ein Zugriff auf die Daten sich somit als sehr umständlich erweist. Das Ziel dieses Teils der Diplomarbeit ist, die Daten in Echtzeit zu loggen und in einer zusätzlichen Datenbank abzuspeichern, um diese anschließend grafisch ansprechend darzustellen.

4.2.1 Log4J

Framework, um Anwendungsmeldungen von JAVA zu loggen.

4.3 Aufbau des Projektes

4.3.1 Beschreibung der Funktionen

4.3.2 Logger

Das sogenannte FS-Logger-Programm setzt sich aus mehreren Klassen zusammen, wie etwa einem DataJDBCAdapter und einer Hauptklasse namens FlexTaskFsLogger. Weiteres befindet sich in dem Programm eine Modelklasse namens LogEntry, welche die Attribute der eingetragenen LogEntries definiert. Um einen reibungslosen Ablauf zu garantieren, wird eine Blocking-Queue verwendet, welche Multithreading unterstützt. Das ganze Programm baut auf dem Producer-Consumer-Pattern auf.

DataJDBCAdapter

Der DataJDBCAdapter ist dafür verantwortlich, eine Verbindung zu der PostgreSQL-Datenbank aufzubauen und somit die benötigten Objekte in einer Datenbank abzu-

speichern (insert). Um diese Verbindung herzustellen, verwendet der JDBCAdapter eine URL, welche in einem String abgespeichert ist sowie einen Benutzernamen und ein Passwort. Innerhalb der connect-Methode werden diese Daten verwendet, sie gibt ein Connection-Objekt zurück, welches angibt, ob die Verbindung zur Datenbank erfolgreich war. Die connect-Methode wird innerhalb der darunterliegenden insertLogEntry-Methode aufgerufen. Wenn die Anbindung erfolgreich war, wird ein PreparedStatement mithilfe des vorher übergebenen SQL-String erstellt und darauffolgend ausgeführt. Somit wird erfolgreich ein neuer Datenbankeintrag in der Datenbank gespeichert. Falls während des ganzen Ablaufes ein unerwarteter Fehler auftauchen sollte, wird dieser mittels einer Java-Exception in die Konsole geschrieben.

Main Klasse FlexTaskFsLogger

Die Klasse FlexTaskFsLogger ist die Main-Klasse des Programms. Sie führt das Programm aus. Dieser Klasse werden auch folgende Methoden vererbt, da sie eine Subklasse der Klasse FlexTask ist:

- getNeededTaskParameters
- getNeededDeviceParameters
- isSingleDeviceTask
- closing
- initTask

Zusätzlich befindet sich noch in der Klasse eine Getter- und Setter-Methode, diese sind allerdings lediglich für eine Zählvariable zuständig, um bestimmte Daten weniger oft in die Datenbank einzuspeichern.

Die Hauptmethode des FlexTaskFsLogger ist die initTask-Methode. In ihr wird eine BlockingQueue mit dem generic Type LogEntry erstellt. Mithilfe dieser Blocking-Queue werden darauffolgend ein Producer und ein Consumer erstellt, welche die Blocking-Queue als Parameter übergeben bekommen. Nachfolgend werden jeweils zwei Producer-Threads und zwei Consumer-Threads erstellt und gestartet. Um eine zeitliche Einteilung zu haben, wird ein sogenannter TimerTask erstellt, welcher wiederum auch ein neuer, unabhängiger Thread ist. Er wird dazu genutzt, um in einem gewissen Intervall Methoden oder Code-Blöcke aufzurufen. In diesem Fall wird der Task dazu verwendet, über den DataJDBCAdapter einen neuen Eintrag in der Datenbank zu speichern. Dazu wird zuallererst überprüft, ob eine Variable, welche für das Starten und Stoppen des Tasks zuständig ist, auf true gesetzt wurde. Anschließend wird ein sogenannter StringBuilder,

welcher aus einer Kette von Strings besteht, aus dem vorher erstellten Consumer geholt. Dieser StringBuilder besteht aus einem aneinandergeketteten Insert-Statement. Um die Ausführbarkeit des Stringbuilders zu gewährleisten, muss die letzte Stelle (an welcher sich ein Beistrich befindet) gelöscht werden und durch einen Strichpunkt ersetzt werden.

Um das korrekt erstellte SQL-Statement nun ausführen zu können, wird in einem try and catch Statement die insertLogEntry-Methode des DataJDBCAdapters ausgeführt. Dabei wird der StringBuilder mit dem SQL-Statement, welches mithilfe der toString-Methode in einen String umgewandelt wird, übergeben.

Anschließend wird der eben verwendete StringBuilder geleert, indem die delete-Methode aufgerufen wird. Um einen neuen Insert-Aufruf zu ermöglichen, wird ein insert-Header (siehe Abb.1) über die setStringBuilder-Methode aus dem Consumer gesetzt.

Nun wird der vorher beschriebene TimerTask in einem eine Zeile davor erstellten Timer als Parameter übergeben.

Der nächste Abschnitt sorgt dafür, dass das Programm, also der Consumer und der Producer aus- und eingeschaltet werden kann. Dazu werden zuerst zwei Datenpunkte initialisiert. Diese werden durch das Setzen einer spezifischen Adresse und den Namen eines Datenpunkts erstellt. Dabei ist der Datenpunkt `logger_nav_status_icon_Click` dazu verantwortlich, darauf zu hören, ob sich ein Wert, welcher in diesem Datenpunkt gespeichert ist, geändert hat. Das heißt, wenn auf der eigens dafür erstellten Website der Ein- und Ausschaltknopf gedrückt wird, wird der Timer pausiert und das Programm schreibt keine Werte mehr in die Datenbank. Im Gegensatz dazu ist der `logger_nav_status_icon_State` dafür da, einen Wert für die Website zu übergeben. Damit wird die Farbe des Icons verändert, um dem User ein Feedback zu seiner Aktion zu geben. Nachdem die Methode setDatapointChangedCommand mit dem `logger_nav_status_icon_Click` verbunden wird, wird aktiv auf den Datenpunkt gehört. Ein neuer DatapointCommand wird dabei übergeben, in ihm wird die execute-Methode überschrieben. In dieser Methode wird zuallererst eine temporäre Variable erhöht, sie ist dafür da, jeden zweiten Dateneintrag zu überspringen, da die HMI jedes Mal zwei Werte schickt, allerdings nur einer benötigt wird. In Folge darauf wird in einem If-Statement der Wert des `logger_nav_status_icon_State` Datenpunkts geändert, um die Farbe des Einschalt Icons zu verändern. Zusätzlich wird die startOrStop Variable auf false gesetzt und der Producer und der Consumer werden mithilfe der Übergabe von ‚false‘ in der setRun-Methode gestoppt. Im else-Zweig des If-Statements

wird genau das Gegenteilige bewirkt, d.h. Producer und Consumer werden gestartet und die `logger_nav_status_icon_State` wird wieder auf den ursprünglichen Wert geändert.

LogEntry Klasse

Dies ist die Model-Klasse des Programms. In ihr werden die Attribute des LogEntries festgelegt. Diese beinhalten die dp-id, d.h. die Id des Datenpunktes, welcher gespeichert werden soll. Außerdem wird der Wert des Datenpunktes, die Einheit und der Timestamp, also der genaue Zeitpunkt, an welchem der LogEntry erstellt wurde, angelegt. In der Klasse befinden sich zusätzlich Getter und Setter für die Attribute und ein Konstruktor.

Blocking Queue - Producer

Diese Klasse ist der Producer der Blocking Queue, d.h. in dieser Klasse werden die Objekte in die Blocking Queue hinzugefügt.

Es lassen sich zwei Attribute im Producer finden, die Blocking Queue, in welche die Objekte später hinzugefügt werden und eine Boolean Variable, welche den Namen "run"trägt. Sie ist dafür verantwortlich, den Producer zu deaktivieren bzw. zu aktivieren. Unterhalb der Attribute befinden sich ein Konstruktor, um die Blocking-Queue zu übergeben und ein Getter und ein Setter für die run-Variable.

Die Hauptmethode der Klasse ist die run-Methode. Hier wird ein neuer Datapoint-Command erstellt, in welchem die execute-Methode überschrieben wird. Innerhalb der execute-Methode wird ganz oben eine Counter Variable initialisiert, sie ist dafür verantwortlich, bestimmte Daten von Datenpunkten nur jedes zweite Mal in die Blocking Queue hinzuzufügen. Danach befindet sich ein if-Statement, welches überprüft, ob der Code im Consumer ausgeführt werden soll oder nicht. In diesem if-Statement befindet sich ein Switch für den specificDataType, welcher je nach DataType einen bestimmten Codeteil ausführt. In dem einen Codeteil wird die counter-Variable auf true oder false überprüft (bei einem false wird der Datenteil nicht in die Blocking-Queue hinzugefügt), in dem anderen wird dies ignoriert.

Allerdings kümmern sich beide Codeteile darum, die jeweiligen Daten als LogEntry zu erstellen und diesen anschließend in die Blocking Queue zu pushen.

Mithilfe des Codes, welcher sich unterhalb des erstellten DataPointCommands befindet, wird über die Datenpunkte des jeweiligen Geräts iteriert und sorgt dafür, dass der oben liegende Codeteil bei einer Änderung ausgeführt wird.

Blocking Queue - Consumer

In dieser Klasse geht es um den Consumer der Blocking Queue, d.h. vom Consumer werden Objekte aus der Blocking Queue genommen und diese weiterverarbeitet. Die Attribute in dieser Klasse sind so wie in der Producer Klasse eine Blocking-Queue, sowie die run-Variable, um den Consumer zu aktivieren bzw. zu deaktivieren. Im Konstruktor wird wiederum die Blocking-Queue übergeben.

In der run-Methode der Klasse wird in einem while-True ein if-Statement ausgeführt, in welchem sich ein Try-and-Catch-Statement befindet. In diesem wird das benötigte Objekt aus der Blocking-Queue geholt und anschließend in einen StringBuilder als gültiges SQL-Statement hinzugefügt.

Unterhalb befindet sich noch eine Getter- und Setter-Methode für den StringBuilder sowie die run-Variable.

4.3.3 Quarkus Backend (Beschreibend)

Das Backend besteht aus Java und dem Java-Framework Quarkus.

LogEntry Klasse

Hier werden die Attribute des LogEntries festgelegt. Diese beinhalten wie auch im Programm FlexTaskFsLogger die dp-id, den Wert, die Einheit und einen aktuellen Timestamp des Eintrags. Auch beinhaltet ist ein Konstruktor, Getter- und Setter-Methoden für die Attribute sowie eine toString-Methode, um die Attribute formatiert zurückgeben zu können.

LogEntry Ressource

Mithilfe dieser Klasse werden bestimmte Daten an bestimmte vorher definierte Adressen geschickt. Der vorher definierte Path ‚/logEntry‘ wird dabei bei jeder Anfrage am Anfang der Adresse verwendet. Um das davor erstellte LogEntry-Repository zu verwenden, wird es am Anfang der Methode mit dem Schlüsselwort new initialisiert.

In der Ressource befinden sich verschiedenste GET-Methoden, welche alle einen anderen Nutzen haben:

- getAll(): Hier wird durch die Übergabe der Parameter definiert, in welchem Zeitraum die benötigten Daten zurückgegeben werden. Diese Parameter werden mithilfe von PathParam übergeben, d.h. die Daten werden über die URL übergeben. Zusätzlich steht über der Methode ein `@Produces(MediaType.APPLICATION_JSON)`, um festzulegen, in welchem Format der Rückgabewert zurückgegeben wird. In diesem Fall ist es das JSON-Format.
- getByName(): Diese Methode ist sehr ähnlich zur getAll()-Methode. Lediglich wird ein zusätzlicher Name übergeben und somit die getByName-Methode des Repositories verwendet.
- getCSV(): In dieser Methode wird im Pfad neben den Daten ein FilePath angegeben, in welchem die CSV-Datei gespeichert werden soll. Mithilfe der getCSVAll-Methode aus dem Repository wird die CSV-Datei anschließend im richtigen Pfad gespeichert.
- getCSVByName(): Die Methode hat eine ähnliche Funktion wie die getCSV()-Methode. Der entscheidende Punkt dabei ist, dass ein Name übergeben werden kann, welcher die Rückgabedaten beeinflusst, da so nur die Daten mit dem richtigen Namen als CSV-Datei erstellt wird.
- insert(): Mithilfe dieser Methode kann ein neuer LogEntry in die Datenbank eingefügt werden. Als Parameter in der insertLogEntry-Methode des Repositories wird dabei ein neu erstellter LogEntry übergeben.
- downloadFile(): Um das vorher erstellte CSV-File zu downloaden, wird diese Methode genutzt. In diesem Fall ist der Rückgabewert der Methode mit einem `@Produces({"text/csv"})`definiert, da es sich dabei um eine CSV-Datei handelt. Zuallererst wird in der Methode ein File-Name, ein Pfad, sowie ein File erstellt. Der Pfad und der File-Name ist dabei jeweils vom Datentyp String, das File ist vom Datentyp File und wird mithilfe des Pfads als Parameter erstellt. Um zu überprüfen, dass der Pfad existiert, wird mithilfe eines IF-Statements die Methode exists() beim vorher erstellten File als Überprüfung herangezogen. Wenn dieses IF-Statement feststellt, dass das File nicht existiert, wird eine RuntimeException geworfen. Diese enthält die Nachricht "File not found: ünd den zugehörigen File-Namen. Bei einem erfolgreichen Erstellen des Files wird ein ResponseBuilder namens res erstellt. Dieser enthält die Response OK sowie das File. Zusätzlich wird bei dem ResponseBuilder ein Header gesetzt, in welchem sich unter anderem der FileName befindet. Schlussendlich wird der ResponseBuilder mit einem Build-Statement zurückgegeben. Somit

wurde das vorher erstellte File erfolgreich heruntergeladen und kann nun mithilfe von einem passenden Programm geöffnet und gesichtet werden.

LogEntry Repository

Das Repository des Backends ist dafür verantwortlich, eine Verbindung zur Datenbank herzustellen und die richtigen Daten an die Ressource weiterzugeben. Zuallererst werden drei verschiedene Strings definiert, um Zugriff auf die Datenbank zu erlangen. Der erste ist dabei die URL, um sich zur Postgres-Datenbank zu verbinden. Der zweite und dritte String definiert den User sowie das Passwort, um Zugriff zur Datenbank zu erlangen.

Die erste Methode in der Klasse trägt den Namen connect. Wie der Name schon sagt, wird in der Methode mithilfe eines DriverManagers eine Verbindung auf die Datenbank aufgebaut und zurückgegeben, welche in den folgenden Methoden verwendet werden kann.

Listing 13: Connect to SQL Database

```
1       /**
2        * Connect to the PostgreSQL database
3        *
4        * @return a Connection object
5        */
6       public Connection connect() throws SQLException {
7           return DriverManager.getConnection(url, user, password);
8       }
```

Jede der nachfolgenden Methoden ist für eine bestimmte Aktion auf der Datenbank zuständig. Um alle vorhandenen LogEntries in einem bestimmten Datumsbereich zu erlangen, kann die getAll-Methode verwendet werden. Die übergebenen Parameter sind dabei das Anfangs- und das Enddatum, sowie die Anfangs- und die Endzeit. Anfangs wird nun ein Set von LogEntries erstellt, in welchem nachfolgend die Daten hinzugefügt werden. Um das Datum und die Zeit in Millisekunden umwandeln zu können, da nur diese später in einem SQL-Statement verwendet werden können, wird eine convertToMillis-Methode verwendet. So wird eine Start- und Endzeit in Millisekunden erstellt. Um die Daten, welche zwischen dem Start- und dem Enddatum liegen, zu bekommen, wird in einem SQL-Statement definiert, dass der Timestamp des jeweiligen LogEntries größer als die Startmillisekunden, und kleiner als die Endmillisekunden ist. Außerdem werden die Daten nach dem timestamp geordnet. Anschließend wird eine Verbindung zur Datenbank aufgebaut. Um das vorher erstellte SQL-Statement nutzen zu können, wird ein PreparedStatement genutzt. In diesem werden die beiden Parameter startMillis

und endMillis gesetzt. Aus diesem PreparedStatement wird nun ein ResultSet gewonnen, durch die Methode executeQuery. Um vollständige LogEntries zu erhalten, werden alle ResultSets mithilfe einer while-Schleife durchgegangen. Jeder der LogEntries wird zu dem Set namens LogEntries hinzugefügt. Um die richtigen Spalten der LogEntries zu bekommen, wird der Name der Spalte verwendet.

Bei einem Fehler in der Verbindung zur Datenbank wird eine Fehlermeldung ausgegeben. Bei einem Erfolg wird das Set von LogEntries zurückgegeben und kann somit in der Ressource verwendet werden.

Die getByName-Methode ist sehr ähnlich zur getAll-Methode. Der einzige Unterschied ist, dass als Parameter zusätzlich ein Name mitübergeben wird. Dieser wird zusätzlich im SQL-Statement gesetzt und somit werden alle LogEntries, welche einen anderen Namen tragen, aussortiert. Zurückgegeben wird erneut ein Set aus LogEntries.

Die bereits verwendete convertToMillis-Methode befindet sich direkt unter der vorigen Methode. In ihr wird das Datum und die Zeit als gemeinsamer String erstellt. Dieser String wird nun verwendet, um eine Variable des Typs LocalDateTime zu erstellen. Dieses wird nach dem richtigen Formatieren bzw. Umwandeln zurückgegeben.

Der nächste Abschnitt des Repositories ist für alle Methoden rund um das Erstellen von CSV-Dateien zuständig. Die ersten beiden Methoden unterteilen jeweils, ob ein Name mitübergeben wurde, oder nicht. Die Parameter sind dabei das Anfangs- und Enddatum, sowie die Anfangs- und Endzeit. Bei der zweiten Methode wird nun ein Name zusätzlich übergeben. Der meiste Teil der Methoden ist relativ ähnlich, zuerst wird ein Set von LogEntries erstellt. Je nach Methode werden mithilfe der Parameter die richtigen LogEntries mit der getByName-Methode bzw. der getAll-Methode in dem Set gespeichert. Nachfolgend wird aus dem Set ein Stream erstellt, dabei wird jedes logEntry-Objekt zu einem String umgewandelt. Anschließend wird jeweils die Methode writeToCSVFile aufgerufen, welche sich direkt unter den anderen zwei Methoden befindet. Übergeben wird dabei beide Male der zuvor erstellte Stream mit den Strings, sowie ein neu erstelltes File mit einem vorher erstellen FilePath.

Die erste Zeile in der writeToCSVFile-Methode konvertiert das eben übergebene Set zu einer Liste. In diesem Prozess wird eine weitere Methode angewendet, welche den Namen convertToCSVFormat trägt. Mithilfe eines Map-Befehls wird die Methode auf jeder Zeile des Sets angewendet. Die convertToCSVFormat-Methode gibt dabei die übergebene Zeile als Stream zurück, wobei zwischen den einzelnen Werten in einer Zeile

ein Semikolon eingefügt wird. Als nächstes wird ein BufferedWriter verwendet. Dieser wird als neue Instanz erstellt, mit dem Übergabeparameter eines neuen FileWriters, in welchem das am Kopf der Methode übergebene File übergeben wird. Der BufferedWriter ist von einem Try-and-Catch umgeben. Dies ist erforderlich, um bei einem eventuell auftretenden Fehler, wie etwa einem falschen Filepath oder fehlenden Berechtigungen, mit einer Fehlermeldung richtig reagieren zu können. In diesem Try-and-Catch wird nun jede Zeile der vorher erstellten Liste mithilfe einer For-Schleife durchgegangen. Mit den Methoden write() und newLine() wird somit die CSV-Datei Zeile für Zeile erstellt.

Die letzte Methode im Repository ist die insertLogEntry-Methode. Wie der Name bereits verrät, ist sie dafür zuständig, neue LogEntries in der Datenbank zu speichern. Um einen neuen LogEntry zu speichern, wird zuallererst ein String erstellt, welcher ein Insert-Statement enthält. Danach wird eine Verbindung zur Datenbank mithilfe der connect-Methode hergestellt. Durch ein PreparedStatement, welches aus einem SQL-String geformt wird, können alle Parameter gesetzt werden. Diese beinhalten die dpId (der Name des Datenpunkts), die value (den Wert des Datenpunkts), die unit (die Einheit des Datenpunkts), sowie den timestamp (wann der Datenpunkt erstellt wurde). Nachdem alle Parameter gesetzt wurden, wird ein executeUpdate durchgeführt, mithilfe dessen der neue LogEntry in die Datenbank geschrieben wird.

4.3.4 Angular Frontend (Beschreibend)

Website - User/Anwender Ansicht

Die erste Seite, welche zu sehen ist, ist die Hauptseite 15. Auf ihr ist zuerst ein großes Bild zu finden, gleich darunter kann die Schrift "Lassen Sie sich ihre gewünschten Diagramme anzeigenërkannt werden. Durch drei verschiedene Formulare, welche sich direkt unter der großen Überschrift befinden, können verschiedenste Funktionen genutzt werden. Jedes dieser Formulare besitzt dabei einen anderen Zweck.

- Formular 'Alle Diagramme': Das erste ist dafür zuständig, alle Diagramme in einem bestimmten Zeitraum anzuzeigen. Wenn der Button 12 dieses Formulars betätigt wird, erfolgt eine Weiterleitung auf eine weitere Seite. Auf dieser kann nun zwischen allen Diagrammen mithilfe von automatisch generierten Buttons wechseln 16.
- Formular 'Diagramm per Name': Das zweite Formular hat eine ähnliche Funktion. Lediglich kann hier das Diagramm, welches angezeigt werden soll, im Formular

Abbildung 11: Formular Zeitraum

Abbildung 12: Formular Button

mitübergeben werden. Durch den Button 12 erfolgt nun eine direkte Weiterleitung zum gewünschten Diagramm.

- Formular 'CSV File': Eine ganz andere Funktion hat das letzte Formular. In ihm wird zwar genauso der Wunschzeitraum angegeben, allerdings wird nach Betätigen des Buttons 12 ein CSV-File generiert und anschließend heruntergeladen. In diesem werden alle Daten übersichtlich angezeigt. 13

Die Formulare können dadurch genutzt werden, dass zuerst jeweils die richtigen Von-, Bis-Daten angegeben werden müssen. 11 Wenn die Daten falsch eingegeben wurden (d.h. es wurde ein Datum eingegeben, welches hinter dem Anfangsdatum liegt bzw. bei einem gleichen Anfangs- und Enddatum muss die Endzeit hinter der Anfangszeit liegen), wird eine Fehlermeldung ausgegeben, um den User auf seine fehlerhafte Eingabe aufmerksam zu machen. Bei dem Formular 'Diagramm per Name' muss zusätzlich ein Name angegeben werden, sonst wird erneut eine Fehlermeldung angezeigt. Wenn allerdings das Formular 'CSV-File' betrachtet wird, gibt es eine zusätzliche Fehlerüberprüfung, welche direkt nach dem Betätigen des Buttons ausgeführt wird. Werden keine Daten aus der Datenbank für den eingegebenen Zeitraum gefunden, wird eine passende Fehlermeldung angezeigt. 14

1	REAL78	430	A	1,6603E+12
2	Shadow_EG_Eingangsbereich	430	A	1,6603E+12
3	Kompressor_Druck_IST	07.Mär	A	1,6603E+12
4	REAL1	07.Mär	A	1,6603E+12
5	REAL173	1300	A	1,6603E+12
6	REAL178	1900	A	1,6603E+12
7	REAL173	1200	A	1,6603E+12
8	REAL173	1300	A	1,6603E+12
9	REAL173	1200	A	1,6603E+12
10	REAL178	1800	A	1,6603E+12
11	REAL178	1900	A	1,6603E+12
12	REAL178	1800	A	1,6603E+12

Abbildung 13: CSV-Datei Beispiel Ausgabe

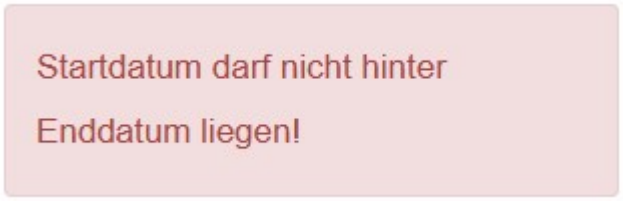


Abbildung 14: Webseite Fehlermeldung

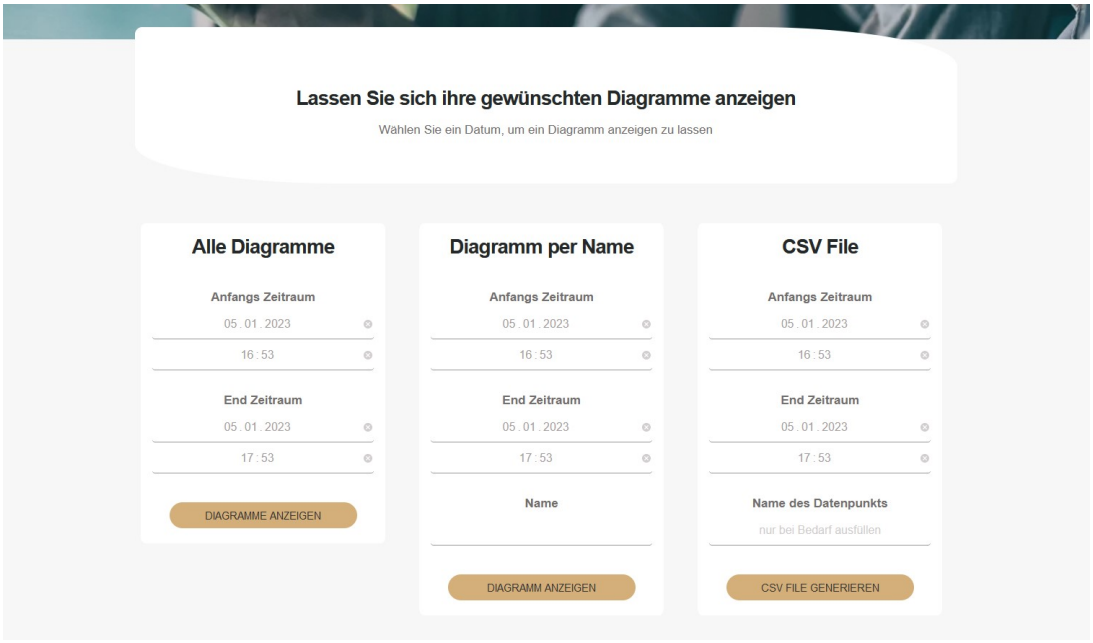


Abbildung 15: Website Hauptseitenansicht

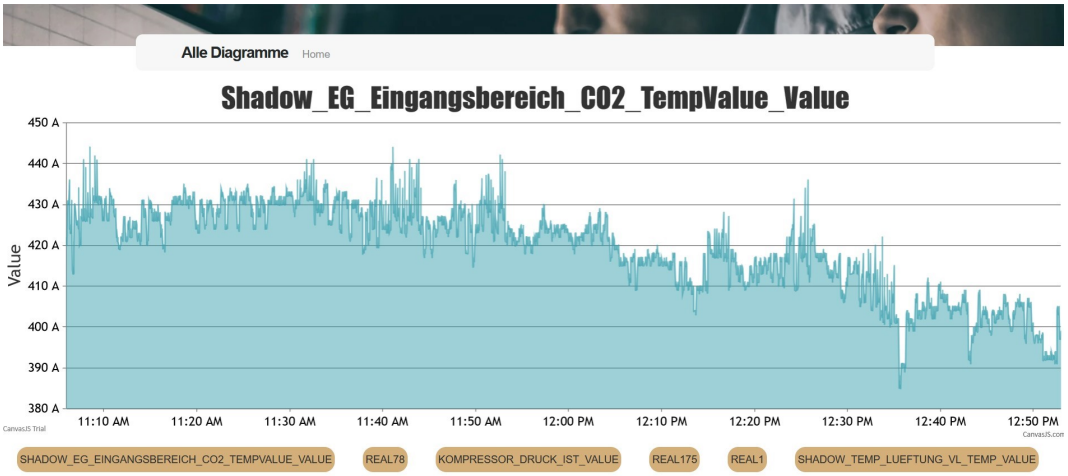


Abbildung 16: Website Diagrammansicht

App-Component

In der TypeScript Klasse der App-Komponente wird selbst ein Titel definiert, dieser trägt in diesem Fall den Namen „flexloggerFE2". Im HTML-File ist dabei ein Router-Outlet definiert. Durch dieses wird das Routing im Projekt ermöglicht. Das heißt, jede Komponente wird praktisch in der App-Komponente angezeigt. In der app-routing-module Klasse werden alle Pfade des Projekts definiert. Dabei wird jeweils ein String als Pfad angegeben sowie eine Komponente definiert, zu welcher der Pfad führen soll. Zusätzlich wird bei den Modulen des Projekts ein RouterModule hinzugefügt, welches zusätzlich die Methode forRoot verwendet, in welchem die vorher definierten Routen als Parameter übergeben werden. In der App-Module-Klasse werden noch weitere Module hinzugefügt:

- BrowserModule: Stellt einen Service zur Verfügung, mit welchem eine Browser-App gestartet, sowie laufen gelassen werden kann.
- AppRoutingModule: Ermöglicht das Navigieren zwischen verschiedenen Komponenten.
- HttpClientModule: Mithilfe dieses Moduls können Netzwerk Requests abgesetzt werden. Diese inkludieren GET, POST, PUT, PATCH und DELETE.
- FormsModule: Durch dieses Modul können template-driven Forms erstellt werden.
- ReactiveFormsModule: Mithilfe dieses Moduls kann ein ReactiveForm verwendet werden.

Home-Component

Die ist die Hauptkomponente des Programms. In ihr werden 3 Formulare definiert, jedes davon hat eine andere Funktion. Im Konstruktor der Komponente werden verschiedenste Parameter übergeben:

- HttpService: Dies ist der Service, mithilfe dessen eine Server-Verbindung zum Backend aufgebaut werden kann.
- Router: Mithilfe dieses in Angular eingebauten Features kann auf der Website zwischen den Komponenten gewechselt werden.
- ActivatedRoute: Mithilfe dieses Parameters können Daten über Komponenten hinweg übergeben werden.
- FormBuilder: Durch diesen Parameter können reaktive Formulare erstellt werden
- ValidatorService: Wird später verwendet, um die Richtigkeit der Eingabe bei den Formularen zu garantieren.

Im Konstruktor selbst, wird das heutige Datum sowie das heutige Datum plus einer Stunde gesetzt. Um die Komponente zu initialisieren, wird in der ngOnInit()-Methode die initForms-Methode aufgerufen. Diese initialisiert alle 3 Formulare, indem sie die Variablen in den Formularen, sowie Validatoren setzt. Dabei wird bei den Variablen jeweils ein Standardwert gesetzt. Die gesetzten Validatoren überprüfen dabei, ob die Daten korrekte Eingaben in den Formularen sind.

Weiteres finden sich in der Komponente zwei Methoden, welche sich um das Routing kümmern. Diese kommen beim Klicken der Buttons der Formulare zum Einsatz, um die Komponente, welche angezeigt wird, zu wechseln. Die benötigten Daten werden dabei in der URL übergeben.

Die nachfolgende Methode ist für das Generieren sowie das Downloaden einer CSV-Datei zuständig. Zuallererst wird die checkCSV-Methode aufgerufen, in welcher überprüft wird, ob in dem ausgewählten Zeitraum Daten verfügbar sind.

Wenn die checkCSV-Variable als true gesetzt wurde, wird überprüft, ob der Namenparameter im Formular nicht leer ist. Trifft dies ein, wird mithilfe eines Timers und des Http-Service eine CSV-Datei generiert, welche alle Datenpunkte in dem richtigen Datumsbereich generiert. Nachdem das Generieren erfolgreich abgeschlossen wurde, wird der Download der Datei gestartet. Dieser Download wird mithilfe der window.open() Methode verwirklicht. Wenn der Namenparameter in dem Formular einen Namen enthält, so wird ein CSV-File generiert, welches nur die Daten eines Datenpunkts beinhält.

Wenn der Fall eintrifft, dass die checkCSV-Variable auf false gesetzt wurde, dann werden die Errors des Namenparameters auf true gesetzt. Dadurch wird unter dem Formular eine Fehlermeldung ausgegeben, um dem User eine Rückmeldung der Formulareingabe zu geben.

Canvas-Chart-Component

In dieser Komponente wird ein Diagramm erstellt, in welchem anschließend mithilfe von verschiedenen Buttons die gewünschten Daten angezeigt werden.

Der erste wichtige Teil der Komponente ist das Erstellen der Diagrammoptionen. In diesen kann man verschiedenste Attribute eines Diagramms definieren:

- animationEnabled: Stellt ein, ob beim ersten Anzeigen eines Diagramms jeder Punkt des Diagramms flüssig geladen wird, um ein dynamischeres Ergebnis zu erhalten.
- title: Setzt den Titel fest, welcher als Überschrift über dem Diagramm stehen soll.
- axisY: Legt den Titel der Y-Achse fest, diese Einstellung kann genauso auf der X-Achse getätigt werden.
- data: Dies ist die wichtigste Einstellung der Diagrammoptionen. Sie legt den Typ des Diagramms fest (in diesem Fall ist es ein Liniendiagramm), die Farbe des Diagramms und setzt die Datenpunkte fest. Beim ersten Laden der Seite werden die Datenpunkte der X- und Y- Achse auf 0 bzw. den 01.01.1970 gesetzt.

Beim Laden der Seite wird zuerst im Konstruktor der Komponente eine Funktion namens onload() ausgeführt. Diese ist dafür zuständig, die erforderlichen Daten mithilfe des http-Service aus dem Backend zu bekommen. Zuerst werden hierfür die übergebenen Werte aus dem Formular mithilfe von Route-Snapshots übergeben. Anschließend wird durch den http-Service eine getLogEntries-Methode aufgerufen, diese gibt die Werte zurück, welche das erforderliche Datum besitzen. Diese werden nun in einem Array gespeichert, um weiter verwendet zu werden.

In der nächsten Zeile des Konstruktors befindet sich ein Timer, welcher nach 1 Sekunde den darauffolgenden Code durchführt. In diesem Code wird zuallererst eine getFiles-Methode ausgeführt, diese gibt das vorher erstellte Array zurück. Das IF-Statement eine Zeile darunter garantiert, dass die Länge des Arrays nicht 0 beträgt, ansonsten wird die Fehlermeldung SZu Ihrem ausgewählten Zeitpunkt wurden keine Daten gefunden.äusgegeben. Bei einem positiven Ergebnis des IF-Statements werden nachfolgend vier Methoden aufgerufen:

- getListOfDatapointNames(): Diese Methode kümmert sich darum, eine Liste der Namen für die Buttons zu erstellen. Diese Buttons sind später dafür zuständig, zwischen den angezeigten Daten zu wechseln. Um zu verhindern, dass der Name eines Datenpunkts öfters in der Liste vorkommt, wird am Beginn eine Boolean-Variable namens nameInList erstellt. Diese wird vorerst auf False gesetzt. Anschließend wird ein vorher initialisiertes Array auf ein leeres Array gesetzt, in welches danach die Namen hinzugefügt werden. Um alle Namen aus den Daten zu erlangen, wird eine for-Schleife verwendet, welche alle vorher erhaltenen Daten durchgeht. Der erste Name wird immer hinzugefügt, daher wird, wenn die Länge des leeren Arrays 0 ist, der erste Name hinzugefügt. Sonst wird eine weitere for-Schleife betreten, welche alle Elemente der Liste der Namen durchgeht. Wenn ein Element bereits vorhanden ist, wird die nameInList Variable auf true gesetzt. Später wird in einem weiteren If-Statement überprüft, ob diese Variable auf True oder False gesetzt ist. Bei einem False wird dabei der Name in die Liste hinzugefügt. Durch dieses Verfahren wird sichergestellt, dass kein Name doppelt in der Liste vorkommt und somit Buttons nicht doppelt angezeigt werden.
- changeData() Hier werden die geänderten Daten in den Diagrammoptionen gespeichert. Um dies umzusetzen, wird als Parameter ein String namens filterString übergeben. Mithilfe dessen und einer For-Schleife werden alle Datenpunkte herausgefiltert, welche nicht den gewünschten Namen besitzen. Die richtigen Daten werden nun im Array dynamicLogLines gespeichert. Nach dem Aussortieren der Daten wird mit einem IF-Statement überprüft, ob die erste Stelle des dynamicLogLines-Arrays nicht undefiniert ist. Wenn dies der Fall ist, werden der erste Datenpunkt, sowie der Titel in den Diagrammoptionen gesetzt. Danach werden die restlichen Datenpunkte mithilfe einer For-Schleife in den Diagrammoptionen gespeichert.
- setChartOptions() Beim Aufrufen dieser Methode werden die Diagrammoptionen neu gesetzt. In diesem Fall werden der Titel, die Einheit und die Datenpunkte des Diagramms erneuert.

Anschließend wird eine Boolean-Variable namens showChart auf True gesetzt, wenn dieser Fall eintritt, wird auf der Website ein Diagramm angezeigt.

Canvas-Chart-Single-Component

Die Komponente ist vom Aufbau her sehr ähnlich wie die Canvas-Chart-Komponente. Genau wie in der anderen Komponente werden zuerst einige Methoden im Konstruktor

aufgerufen. Der größte Unterschied dabei ist, dass keine Buttonnamen erstellt werden, bzw. auch keine Buttons angezeigt werden.

HttpService

Der Service ist dafür zuständig, die jeweiligen Daten aus dem Backend zu beschaffen. Zuerst wird ein String definiert, in welchem die URL des Backends gespeichert ist. Im Konstruktor wird der sogenannte HttpClient als Parameter übergeben, er ist der Hauptakteur in der Klasse. Mithilfe von ihm kann eine Verbindung zum Server hergestellt werden.

In dem Service befinden sich mehrere Methoden. Allgemein kann gesagt werden, dass mit dem HttpClient jeweils einen GET-Request abgesetzt wird, welcher jeweils ein anderes Ergebnis liefert, je nachdem, welche URL als Parameter übergeben wird.

Die ersten beiden haben jeweils als Rückgabe-Parameter ein LogEntry Array. Beide geben die gesuchten Daten in einem bestimmten Zeitraum zurück, lediglich kann bei der zweiten Methode noch einen Namen hinzugefügt werden. Die Daten, welche die Zeiträume definieren, werden als Parameter in den Methoden übergeben. Die nächsten Methoden sind allesamt für das Downloaden eines CSV-Files verantwortlich. Dabei kümmern sich die ersten beiden um das Erstellen der CSV-Datei, die dritte ist für den eigentlichen Download verantwortlich. Um die CSV-Datei zu erstellen, werden wiederum die gewünschten Daten übergeben und anschließend wird daraus eine URL gebaut und ein GET-Request abgesetzt. Der einzige Unterschied zwischen den Methoden ist abermals ein zusätzlicher Name-Parameter. Die downloadCSV-Methode verwendet wie die anderen Methoden einen GET-Request, allerdings hat sie den weiteren Parameter responseType. Dieser ist notwendig, da innerhalb des Requests eine CSV-Datei heruntergeladen wird und somit der Response-Type Array-Buffer definiert werden muss.

ValidatorService

Der ValidatorService ist dafür zuständig, die Richtigkeit der Eingabe im Formular zu überprüfen. Wenn diese als nicht akzeptabel erkannt wurden, werden die Errors der Parameter auf true gesetzt, und somit eine Fehlermeldung ausgegeben.

Die match-Methode überprüft, ob jedes eingegebene Datum als valide Eingabe akzeptiert werden kann. Dabei werden zuerst alle Controls des Formulars übergeben. Bei der ersten

Überprüfung handelt es sich darum, ob das Startdatum hinter dem Enddatum liegt. Bei Bestätigung dieser Überprüfung werden die Errors mit dem Namen dateMustBeBigger aktiviert. Anschließend wird der Fall überprüft, wenn die beiden Daten gleich sind, die Zeiten sich allerdings unterscheiden, d.h. der Zeitraum am gleichen Tag stattfindet. Dies ist grundsätzlich erlaubt, allerdings nur, wenn die Startzeit kleiner ist als die Endzeit. Wenn dies nicht der Fall ist, wird der Error timeMustBeBigger aktiviert.

LogEntry Model

Hier wird ein Model erstellt, welches den Namen LogEntry trägt, in diesem werden die Parameter dpId, value, unit und timeStamp definiert. Das Model wird dazu verwendet, die vorher geloggten Daten aus der Datenbank weiterzuverwenden.

4.3.5 CanvasJS

Mithilfe von CanvasJS, welche eine HTML5- und Javascript-Charting-Library ist, wird das Anzeigen der Daten in Diagrammen ermöglicht.

4.4 Threads

4.4.1 Serialisierung (Nebenläufigkeit und Parallelität)

Wenn in Java über Nebenläufigkeit gesprochen wird, sind dabei Threads gemeint. Diese sorgen für eine gleichzeitige Abarbeitung von Programmen bzw. der Ressourcen. Dies kann umgesetzt werden, indem die Hardware mehrere Prozessoren oder Kerne besitzt und diese parallel Prozesse abarbeiten können.

Bei einem modernen Ein-Prozessor-Betriebssystem wirkt es oft, als wären die Prozesse parallel, doch dies wird einem nur vorgespielt, indem der Prozessor alle paar Sekunden auf einen anderen Prozess wechselt. Bei einem Betriebssystem mit mehreren Kernen, werden die Prozesse wirklich parallel bearbeitet.

Nebenläufigkeit von Programmen steigert Geschwindigkeit

Wie nebenläufige Abarbeitung die Performance bei einem Einprozessorsystem beschleunigt, kann an folgendem Beispiel-Programm betrachtet werden:

- Programm führt eine Reihe von Befehlen aus
- Programm soll Datenbank-Report erstellen/visualisieren

- Dabei können einige Prozesse nebenläufig abgearbeitet werden
- Diese Prozesse sind zum Beispiel: Öffnen der Datenbank, Lesen neuer Datensätze gleichzeitig mit dem Analysieren alter Daten, alte Daten können in eine Report-Datei geschrieben werden während neue Daten analysiert werden
- Wenn diese Prozesse parallel ausgeführt werden, kann sehr viel Zeit eingespart werden beziehungsweise die Performance sehr erhöht werden.
- Dieses Beispiel zeigt allerdings auch, dass Nebenläufigkeit sehr gut geplant werden muss.

4.4.2 Concurrency/Gleichzeitigkeit

Die Ausführung des Programmcodes wird bei einem modernen Betriebssystem von Prozessen, welche jeweils mindestens einen Thread beherbergen, ausgeführt. Das Interessante daran ist, dass nicht etwa die Prozesse nebenläufig ausgeführt werden, sondern die Threads. Dabei können die Threads auf den gleichen Adressraum zugreifen.

Zustände eines Threads

- Noch nicht erzeugt: Lebenslauf beginnt mit Schlüsselwort `new`
- Running (Laufend) und Not Running (Nicht laufend); Thread kommt in den Zustand Running, wenn die Methode `run()` aufgerufen wird. Wenn ein anderer Thread den Prozessor des aktuellen Threads übernimmt, kommt der Thread in den Zustand Not Running.
- Waiting (Wartend): Thread befindet sich in einem Wartezustand
- Beendet: Aktivität des Threads wurde beendet

4.4.3 Aufbau eines Threads

Interface Runnable

Dem Thread muss ein Befehlsobjekt des Typs Runnable übergeben werden 14, um wissen zu können, welcher Code ausgeführt werden soll. Wenn der Thread gestartet wird, werden die Codezeilen im Befehlsobjekt nebenläufig zum anderen Code in dem Programm ab.

Listing 14: Java Runnable

```
1    interface java.lang.Runnable
```

Durch dieses Interface kann nun die run()-Methode verwendet werden. 15

Listing 15: Einfaches Thread Beispiel

```
1       public class CommandThread implements Runnable {
2           @Override public void run() {
3               ...
4               }
5           }
```

Dieser Thread kann nun in einer anderen Klasse verwendet werden, indem er zum Beispiel gestartet (`thread.start()`) oder gestoppt (`thread.stop()`) wird. 16

Listing 16: Thread erstellen/starten

```
1       Thread t1 = new Thread( new CommandThread() );
2       t1.start();
```

Zusätzlich ist es auch noch möglich, den Thread mit (`thread.sleep(ms)`) zu pausieren.

4.4.4 BlockingQueue

Eine BlockingQueue hat typischerweise einen Thread, welcher Objekte produziert und einen anderen, welcher die Objekte konsumiert. 17

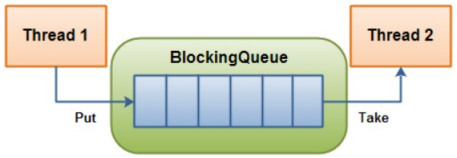


Abbildung 17: Blocking Queue

Der produzierende Thread wird weiterhin Objekte produzieren und sie zur Blocking-Queue hinzufügen, bis das Limit der Queue erreicht wird. Wenn dieses Limit erreicht wird, wird der Producer-Thread blockiert, während er versucht ein neues Objekt hinzuzufügen. Der Producer Thread bleibt dabei blockiert, bis ein Consumer Thread ein Objekt aus der Queue nimmt.

Auf der anderen Seite nimmt der Consumer Thread Objekte aus der Queue, um sie weiter zu verarbeiten. Wenn der Thread versucht ein Objekt aus einer leeren Queue zu nehmen, wird er blockiert, bis der Producer-Thread ein Objekt in die Queue hinzufügt.

BlockingQueue Methoden

Um die Blocking Queue mit Daten zu bespielen, gibt es spezielle Methoden.

put(o)	fügt Objekte in die BlockingQueue hinzu
take()	nimmt Objekte aus der BlockingQueue

Wenn die Operation bei einer der Methoden erfolglos war, blockiert die Methode, bis die Operation erfolgreich war.

Java Blocking Queue Beispiel

Das Beispiel unten angeführt verwendet eine ArrayBlockingQueue als Implementation.

Listing 17: Java BlockingQueue Beispiel

```
1       public class BlockingQueueClass {
2       public static void main(String[] args) throws Exception {
3           BlockingQueue blockingQueue = new ArrayBlockingQueue(2048);
4           Producer producer = new Producer(queue);
5           Consumer consumer = new Consumer(queue);
6           new Thread(producer).start();
7           new Thread(consumer).start();
8           Thread.sleep(4000);
9       }
10      ##################################################
11
12      public class Producer implements Runnable{
13          protected BlockingQueue queue = null;
14          public Producer(BlockingQueue queue) {
15              this.queue = queue;
16          }
17          public void run() {
18              try {
19                  queue.put("object1");
20                  Thread.sleep(1000);
21                  queue.put("2");
22              } catch (InterruptedException e) { e.printStackTrace();}
23      }}
24      ##################################################
25      public class Consumer implements Runnable{
26          protected BlockingQueue queue = null;
27          public Consumer(BlockingQueue queue) {
28              this.queue = queue;
29          }
30          public void run() {
31              try {
32                  System.out.println(queue.take());
33                  System.out.println(queue.take());
34              } catch (InterruptedException e) { e.printStackTrace();}
35      }}
36  }
```

4.4.5 Service executer

Der Service Executer hilft dabei, den eigentlichen arbeitenden Thread von dem Runnable zu trennen. Denn, wenn ein Thread erzeugt wird, muss das Runnable-Objekt im Konstruktor übergeben werden, dies kann zu Problemen führen, da das Thread-Objekt somit nicht vorher erstellt bzw. aufgebaut werden kann. Ein weiterer Grund, das Runnable und den Thread zu teilen, ist, dass ein Thread nicht einfach so ein anderes Runnable bearbeiten kann, da dieses vorher schon zugewiesen wurde.

4.5 Performance

4.5.1 Grenzen (Wie viel Daten gleichzeitig kommen können)

4.5.2 Bandbreite

4.5.3 Menge der Datenpunkte

4.6 Datenbanken

4.6.1 Datenbanken allgemein

Eine Datenbank ist gemeingültig dafür zuständig, viele Daten übersichtlich abzuspeichern, um diese später weiterzuverwenden. Daten werden immer und immer wichtiger und wichtiger in unserer Gesellschaft und diese verlässlich und zugänglich abspeichern zu können ist nicht mehr wegzudenken.

Der Grundaufbau einer relationalen Datenbank (welche in den folgenden Beispielen behandelt wird) besteht aus Zeilen und Spalten, welche zusammen eine Sammlung an Daten halten. Dabei erfüllen die Zeilen die Rolle der einzelnen Elemente und die Spalten teilen die Elemente in seine Unterelemente ein.

4.6.2 H2

Performance (Version 2.0.202)

Die Datenbank ist langsamer bei größeren ResultSets, da sie ab einer bestimmten Anzahl von zurückgegebenen Records zwischengespeichert werden.

Hauptmerkmale

- Sehr schnell, open source, JDBC API
- Embedded und Server Modus, in-memory databases
- Konsolen Anwendung für den Browser
- Das Jar-File hat nur eine Größe von 2.5 MB

4.6.3 SQLite

Performance (SQLite 3.36.0.3)

Performt etwa 2-5x schlechter bei einfachen Arbeiten auf der Datenbank. Dies führt zu einer niedrigen Arbeit-pro-Transaktion Ratio. Allerdings kann SQLite, wenn die Datenbankzugriffe komplexer werden, eine bessere Leistung erbringen. Ein wichtiger Zusatz ist allerdings, dass die Ergebnisse je nach Maschine sehr variieren.

Hauptmerkmale

- Klein
- Schnell
- Sehr verlässlich
- Stand-allone
- Full-featured SQL-Implementierung

Meistgenutzte Datenbank der Welt, benötigt keine Administration. Die Datenbank eignet sich dadurch sehr gut für Mobiltelefone, Kameras, TV-Geräte etc., da diese ohne fachlichen Support funktionieren müssen. Auch für Websites, auf denen weniger bis mittelviele Datenbankzugriffe stattfinden, ist die SQLite Datenbank eine gute Wahl. Bei einer sehr großen Anzahl an Datenbankzugriffen, ist allerdings von einer SQLite Datenbank eher abzuraten.

4.6.4 PostgreSQL

Performance (Version 13.4)

Die Schnelligkeit der Datenbank liegt ziemlich mittig zwischen der Derby- und der H2-Datenbank, wobei sie teilweise etwas schneller als H2 abschneiden kann.

Hauptmerkmale

- Open source
- Objektrelationale Datenbank
- Sehr mächtig
- Verlässlich
- Datenintegrität
- Viele Features/Add-Ons

4.6.5 Derby

Performance (Version 10.14.2.0)

Von all den angeführten Datenbanken die Langsamste. Die Operationen auf der Datenbank werden dabei sehr schleppend ausgeführt. Ein besseres Ergebnis für Derby kann allerdings erzielt werden, wenn Autocommit ausgeschaltet wird. Die Performance wird dabei um 20 % besser. Um einen besseren Vergleich der Schnelligkeit heranzuziehen: Die Datenbank ist nicht einmal halb so schnell wie das erste Beispiel dieser Auflistung, die H2 Datenbank.

Hauptmerkmale

- Open source
- Nur 3.5 MB Größe für die Basis Engine sowie den JDBC-Driver
- Basiert auf JAVA, JDBS sowie SQL Standards
- Derby stellt einen embedded JDBS Driver zur Verfügung, mithilfe dessen die Derby Datenbank in jede JAVA-Applikation eingebunden werden kann
- Supportet den Client/Server Modus
- Einfach zu installieren, einzurichten, sowie zu benutzen

Implementiert in Java.

4.7 Visuelle Darestellung

4.7.1 Angular

Angular ist eine Plattform, um Web-Applikationen zu erstellen, welche für die Desktop- sowie für die mobile Anwendung gleichfalls funktionieren sollen. Gebaut wurde Angular in der Programmiersprache TypeScript und es inkludiert folgende Fähigkeiten:

- Ein Komponenten-basiertes Framework, um eine skalierbare Web-Applikation zu erstellen.
- Eine Sammlung von Bibliotheken, welche eine große Varietät von Features beinhaltet, Beispiele dafür sind Routing, das Management von Formularen sowie eine Client-Server-Kommunikation. Diese Bibliotheken sind laut Angular gut in die Plattform eingebunden.
- Eine Auswahl von Entwickler-Tools, welche hilfreich sind, um den Code zu entwickeln, zu testen, zu bauen und upzudaten.

Komponenten

Komponenten sind die Baublöcke, um eine Applikation zusammenzustellen. In einer Komponente sind folgende Segmente:

- ein HTML-Template
- ein CSS-Style Template
- einem @Component-Part, in welchem folgende Informationen definiert werden:

  Ein CSS-Selektor, welcher definiert, wie die Komponente in einem Template verwendet wird. Dieser Selektor kann anschließend in ein HTML-File eingebunden werden. Passiert dies, wird dieser Selektor eine Instanz der Komponente des HTML-Files.

  Ein HTML-Template, welches Angular anleitet, wie es diese Komponente zu rendern hat.

  Ein optionales Set von CSS-Styles, welche das Aussehen des HTML-Elements definieren.

Listing 18: Beispiel für eine minimierte Angular Komponente

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-world',
  template: `
    <h2>Hello World</h2>
    <p>This is my first component!</p>
  `
})
export class HelloWorldComponent {
  // The code in this class drives the component's behavior.
}
```

Anbindung einer Angular-Applikation an einen Webserver

Um eine Anbindung an einen Webserver zu ermöglichen, stellt Angular ein Modul, welches den Namen HttpClientModule trägt, zur Verfügung. Nachdem dieses in die Klasse AppModule importiert wurde, kann es verwendet werden.

Wenn nun zum Beispiel die Daten eines GET-Requests aus dem Backend ausgelesen werden sollen, muss zuerst der HttpClient aufgerufen werden, definieren, ob ein GET, POST oder ein anderer Request benutzt werden soll und diesem muss anschließend eine passende URL übergeben werden. 19

Listing 19: Beispiel für einen GET-Request

```
this.http.get<Animal[]>(`http://localhost:8080/api/animals`)
  .subscribe(animals => {
    this.animalArray = animals;
  });
```

Mit den spitzen Klammern wird dabei der Type der übergebenen Daten definiert, in diesem Fall wäre es eine Liste des Typs Animal. Die erhaltenen Daten werden dann auf die Variable this.animalArray gesetzt.

Angular Forms

Um in einem Angular-Projekt ein Formular verwenden zu können, muss zuerst die Forms-API hinzugefügt werden. Dies kann mit dem Befehl `npm install @angular/forms -save` umgesetzt werden. Nachdem die Module in der gewünschten Komponente eingebaut wurden, kann ein reaktives Formular erstellt werden.

Um auf die Eingabewerte eines Formulars zugreifen zu können, kann eine FormGroup verwendet werden. Bei dieser muss lediglich ein FormGroup-Element angelegt werden, welches die Namen der Eingabefelder trägt. 20 Der linke Parameter ist dabei dazu da, einen Default-Wert festzulegen, der rechte ist dazu da, eine Überprüfung der Eingabewerte zu aktivieren.

Listing 20: Beispiel für FormGroup eines Angular Formulars

```
1    this.animalForm = this.fb.group({
2      animalName: [null, [Validators.required]],
3      animalAge: [0, [Validators.required]],
4    });
```

Das zugehörige HTML-Form hat in der Praxis diese Form: 21

Listing 21: Beispiel für ein reaktives Formular

```
1    <form [formGroup]="animalForm">
2          <label>Tier Name</label>
3          <input formControlName="animalName" type="text">
4          <label>Tier Alter</label>
5          <input formControlName="animalAge" type="number">
6        </form>
```

4.7.2 CanvasJS

CanvasJS ist ein Framework, mit welchem sich einfach Diagramme erstellen lassen können. Es besitzt eine hervorragende Integration in Angular, dies wird bestätigt durch eine eigene Unterseite auf der CanvasJS-Seite, welche der Integration mit Angular gewidmet ist. Auf dieser können unterschiedlichste Diagramme mit dem passenden Komponenten-, Modul- sowie HTML-Code gefunden werden.

Um CanvasChart in Angular zu verwenden, müssen verschiedene Codeteile eingefügt werden. Diese beinhalten einen HTML-Teil, welcher das Chart anzeigt. 22

Listing 22: CanvasJS HTML

```
<canvasjs-chart [options]="chartOptions" [styles]="{width: '100%', height:
    '360px'}"></canvasjs-chart>
```

Anschließend müssen Dateien, welche auf der canvasJS-Seite heruntergeladen werden können, in die Dateiordner des gewünschten Angular-Projekts kopiert werden. Diese tragen die Namen `canvasjs.min.js` und `canvasjs.angular.component.ts`.

Der vorletzte Schritt beinhaltet das Importieren des CanvasJSChart-Moduls. Im letzten Schritt werden im TypeScript-File der Komponente, in der das Diagramm angezeigt werden soll, die Diagrammoptionen für das Diagramm festgelegt. Diese beinhalten in der einfachsten Form den Titel und die Daten, dies kann aber erweitert werden. 24

Listing 23: CanvasJS chartOptiones

```
chartOptions = {
  title: {
    text: "Basic Column Chart in Angular"
  },
  data: [{
    type: "line",
    dataPoints: [
      { label: "Apple",  y: 10  },
      { label: "Orange", y: 15  }
    ]}]};
```

4.8 Quarkus

4.8.1 Allgemeines

Quarkus wurde kreiert, um Applikationen zu erstellen, welche in einer modernen, cloud-nativen Welt funktionieren sollen. Quarkus ist ein kubernetes-natives Java-Framework, auf GraalVM und HotSpot zugeschnitten. Das Ziel von Quarkus ist es, JAVA zur führenden Plattform für Kubernetes und serverlose Umgebungen zu machen. Quarkus ist Open Source.

Die größte Challenge von Mikroservicearchitekturen ist, dass die Vermehrung von Services die Komplexität des Systems erhöht. Diese kann mithilfe von Kubernetes-basierenden orchestrierenden Systemen1 gelöst werden, da somit die Effizienz sowie die Ressourcenverwertung erweitert werden können. Die Systeme regeln die zeitliche Planung und das Management der Mikroservices in einer dynamischen Weise. Dadurch kann man auch je nach Bedarf an dem System arbeiten, ohne dass die Gefahr besteht, dass ein Container ausfällt. Um nun die gesamten Komponenten zusammenzufügen, wurde das Framework Quarkus entwickelt.

Quarkus funktioniert ausgezeichnet, wenn es darum geht, Cloud-Native Applikationen von Unternehmen zu managen. Es ist in der Lage kurzen nativen Code aus Java Klassen zu bauen, sowie Container Images daraus zu erstellen. Diese Container können darauffolgend auf Kubernetes laufen. Außerdem unterstützt Quarkus die bekanntesten Java Libraries wie etwa RESTEasy, Hibernate, Apache Kafka, Vert.x, usw.

Wie nun schon vorher erwähnt, ist eines der vielversprechendsten Features von Quarkus die Fähigkeit aus Applikationen automatisch Container Images zu generieren. Durch das Generieren von Container Images aus nativen Applikationen wird außerdem eine Gefahr zunichte gemacht. Diese hat mit der nativen Ausführung des Programms zu tun, es handelt sich dabei um potentielle Konfliktfehler von Errors, wenn der Build auf einem anderen Operating System stattfand. Quarkus sorgt außerdem dafür, imperative und reaktive Modelle zu verbinden. Reaktives Programmieren wird immer beliebter aus dem Grund, dass es in der Lage ist asynchrones Programmieren mit Daten Streams und der Veränderung von Daten zu verbinden.

4.8.2 Architektur

Im Zentrum von Quarkus liegt die Kern Komponente, welche die Aufgabe hat, die Applikation in der Build-Phase umzuschreiben, um sie perfekt zu optimieren 18. Daraus entsteht eine native ausführbare und Java-runnable Applikation. Damit der Quarkus-Kern diese Arbeit erledigen kann, müssen einige verschiedene Komponenten zusammenarbeiten:

- Jandex: Ein platzsparender Java Annotation Indexer, sowie eine offline Reflexions Library. Diese Bibliothek ist in der Lage alle runtime sichtbaren Java Annotationen und Klassenhierarchien für ein Set von Klassen in einer speichereffizienten Repräsentation zu indexen.
- Gizmo: Gizmo ist eine Bytecode-Generations-Library, welche von Quarkus verwendet wird, um Java Bytecode zu produzieren.
- GraalVM: Ein Set von Komponenten, in welchem jede eine bestimmte Funktion hat. Beispiele dafür sind: ein Compiler, ein SDK API für die Integration von Graal Sprachen und der Konfiguration von native images, runtime Umgebung für JVM-basierte Sprachen
- SubstrateVM: Unterkomponente von GraalVM, welche die ahead-of-time (AOT) Kompilation von Java Applikationen von Java Programmen zu eigenständigen ausführbaren Programmen erlaubt.

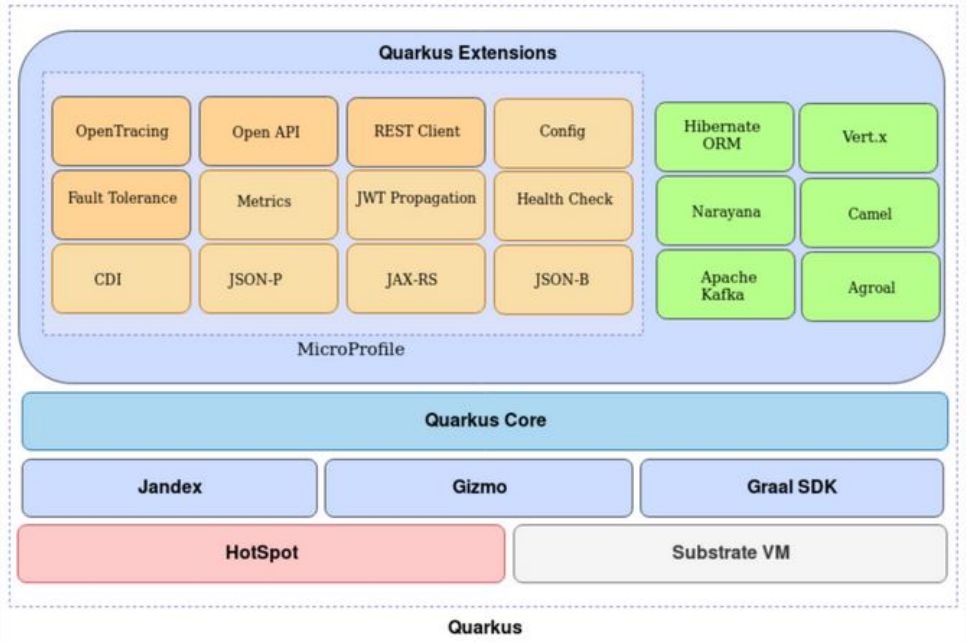


Abbildung 18: Quarkus Architektur

Des Weiteren gibt es noch einige Quarkus Extensions. Dazu gehören die MicroProfile Spezifikationen, sowie ein Set von Extensions für Hibernate ORM, ein Transaktionsmanager (Narayana), ein Verbindungs-Pool-Manager und viele mehr.

4.8.3 Funktionen

4.8.4 Vergleich zu anderen Tools

Drei sehr vergleichbare Frameworks sind Quarkus, Micronout und Spring Boot. Sie besitzen alle ähnliche Features und Fähigkeiten. Am wichtigsten ist dabei, welches Framework ist am schnellsten, welches am performantesten, welches benötigt den wenigsten Speicher?

Quarkus

- Kubernetes-native, für Java designed, für GraalVM und OpenJDK Hotspot, besitzt die besten Java-Bibliotheken und Standards, schnell beim Starten

Micronout

- Cloud-native, JVM-basiert, full-stack Framework für Mikro-Services und serverlose Anwendungen, geringer Speicherverbrauch

Spring Boot

- Open-Source Java Framework, einfach um Stand-alone Anwendungen und Mikroservices zu kreieren, benötigt sehr wenig Konfiguration, um starten zu können

OpenJDK 14 on 2019 iMac Pro Xeon 8 Core. Winner in Red.

METRIC	MICRONAUT 2.0 M2	QUARKUS 1.3.1	SPRING 2.3 M3
Compile Time ./mvn clean compile	1.48s	1.45s	1.33s
Test Time ./mvn test	4.3s	5.8s	7.2s
Start Time Dev Mode	420ms	866ms (1)	920ms
Start Time Production java -jar myjar.jar	510ms	655ms	1.24s
Time to First Response	960ms	890ms	1.85s
Requests Per Second (2)	79k req/sec	75k req/sec	??? (3)
Request Per Second -Xmx18m	50k req/sec	46k req/sec	??? (3)
Memory Consumption After Load Test (-Xmx128m) (4)	290MB	390MB	480MB
Memory Consumption After Load Test (-Xmx18m) (4)	249MB	340MB	430MB

(1) Verifier Disabled
(2) Measured with: ab -k -c 20 -n 10000 http://localhost:8080/hello/John
(3) Spring WebFlux doesn't seem to support keep alive?
(4) Measured with: ps x -o rss,vsz,command | grep java

Abbildung 19: Quarkus Benchmark Test

Für einen näheren Vergleich zwischen den Frameworks siehe 19. Dabei kann erkannt werden, dass Quarkus Response Zeit die bestbewertete ist, Spring trotzt dabei mit einer Compilation Zeit von nur 1.33s, wenn der Befehl `./mnv clean compile` verwendet wird. Insgesamt kann allerdings aus dem Benchmark Vergleich gezogen werden, dass Micronout in den meisten Kategorien als Sieger hervorgeht. Vor allem, wenn einem ein kleiner Speicherverbrauch wichtig ist, sollte somit zu Micronout gegriffen werden.

4.8.5 NodeJS im Vergleich

4.8.6 http GET/POST/PUT Funktionen

Mithilfe von GET/POST/PUT können Daten an einen Webserver geschickt werden, um diese später z.B. in einer Webapplikation nutzen zu können.

Um diese Annotations verwenden zu können, muss eine REST Ressource erstellt werden. Bei dieser Ressource muss zuallererst ein Pfad definiert werden. Dieser gilt für alle darauffolgenden Funktionen in der Ressource Klasse.

Die Funktionen in der Klasse sind alle für verschiedene GET- bzw. POST-Requests mit verschiedenen Pfaden verantwortlich. Bei jedem von ihnen wird definiert, ob ein GET, POST oder PUT Request abgesetzt werden soll. In der darauffolgenden Zeile wird ein Pfad angegeben, welcher zusätzlich nach dem über der Klasse angegebenen Pfad verwendet wird. Als letztes Attribut wird definiert, je nachdem ob der Request

etwas auslesen soll oder etwas anzeigen lassen soll, ein @Produces bzw. ein @Consumes. In diesen wird jeweils der Typ von dem auszulesenden bzw. anzuzeigenden angeführt. In dem Beispiel unterhalb ist es ein JSON-Format.

Listing 24: Quarkus POST-Request

```
1       @GET
2       @Produces(MediaType.APPLICATION_JSON)
3       @Path("/all/animal")
4       public Set<Animal> getAll() {
5           return animalRepository.getAll();
6       }
```

4.8.7 JDBC + JPA

4.9 Abspeicherung von Daten

4.9.1 Vergleich Vor- und Nachteile JSON vs CSV vs Datenbank

Vor allem wenn es darum geht, große Datenmengen abzuspeichern, ist es besonders wichtig, das richtige Datenformat auszuwählen. In diesem Abschnitt wird das CSV, das JSON, und eine herkömmliche Datenbank verglichen.

Daten können intern oder extern generiert werden und dabei gibt es verschiedene Möglichkeiten, die Daten passend abzuspeichern. Es ist sehr wichtig, das richtige Format auszuwählen, da davon einige Faktoren abhängen. Dazu gehören die Verarbeitungsgeschwindigkeit, sowie die Speichergröße. Außerdem kann das Format die Skalierbarkeit, die Kompatibilität, die Cloud-Speicherkosten und die Performance-Geschwindigkeit beeinflussen. Konkrete Beispiele, warum bestimmte Dateiformate ausgewählt werden sollten, sind: Geld zu sparen, indem zu einem CSV-Dateiformat gewechselt wird, wenn große Datensets im Cloud-Speicher verarbeitet werden. JSON ist eine bessere Wahl, wenn es darum geht, kleinere Datensets mit einer komplexen Hierarchie zu speichern. Im Allgemeinen bedeutet dies, dass das richtige Datenformat Geld und Zeit spart.

Ein genauerer Vergleich (exaktere Daten kommen noch)			
Features	CSV	JSON	Datenbank
Speicheranforderung	weniger Platz	mehr Platz	?
Verarbeitungsgeschwindigkeit	schnell	langsam	?
Security	sicherer	unsicherer	sicherer
Große oder komplexe Datensets	große Datensets	komplexe Datensets	Beides

4.9.2 CSV

CSV ist ein Text-Dateiformat, mit der Besonderheit, dass die Werte durch Semikolons unterteilt werden. Dadurch eignet es sich sehr gut für eine Speicherung von sehr vielen Daten. Jede Reihe der CSV-Datei repräsentiert dabei eine Zeile von Daten, die Spalten werden dabei von den Strichpunkten unterteilt. Das CSV-Format kann die Nutzung von Speicherplatz, sowie das Austauschen von Daten maximieren. Strukturierte CSV-Files können außerdem einen Header enthalten, welcher jede Spalte einordnet. CSV-Dateien können außerdem nicht nur durch Semikolons, sondern auch durch Kommas, Tabs und Abstände unterteilt werden. CSV wird am meisten in Entwicklungseinrichtungen und technischen Konsumenten-Anwendungen verwendet. Ein weiterer Vorteil von dem CSV-Dateiformat ist, dass es von der meisten Datenverarbeitungs-Software importiert, konvertiert und exportiert werden kann. Mithilfe dieser Softwares kann die CSV-Dateien auch sehr einfach serialisiert oder deserialisiert werden. CSV ist ein sehr einfach aufgebautes Format und kann somit von fast jedem Datenanalysator ausgewertet werden. Als Nachteile von CSV kann aufgezählt werden, dass es im Rohformat schwer zu lesen ist, und es anfällig für menschliche Fehler ist. Der größte Nachteil gegenüber den anderen zwei Datenformaten sind auf jeden Fall die limitierten Möglichkeiten, die Daten komplex aufzubereiten.

4.9.3 JSON

JSON ist im Vergleich zu CSV leichter verständlich für Menschen. Die Daten werden als semi-strukturiert angezeigt. 25 JSON ist sehr weit kompatibel und wird somit von vielen Software-Entwicklern verwendet, um configs und APIs zu designen. Da JSON von JavaScript entwickelt wurde, kann es sehr einfach in eine Java-basierte Umgebung integriert werden. Somit wird JSON sehr oft für die Datenverarbeitung von Front- und Backend verwendet. Mithilfe von JSON kann sehr einfach auf neue Daten zugegriffen werden. Gerade wenn es um rationales und hierarchisches Datenmanagement geht, eignet sich JSON sehr gut, da es dies sehr unterstützt. JSON-Dateien sind selbsterklärend und es ist für Systeme sehr leicht, eine JSON-Datei zu erkennen und zu verarbeiten. JSON ist somit auch sehr viel einfacher zu lesen als CSV-Dateien.

Listing 25: JSON Beispiel

```
1       {   "pupil": {
2               "id":       "1"
3               "name":     "Margaret"
4           }
5       }
```

4.9.4 Datenbank

4.10 SQL

4.10.1 Allgemein

SQL ist eine Programmiersprache, welche entwickelt wurde, um die Daten in einer relationalen Datenbank zu bearbeiten. Außerdem kann damit die Struktur der Datenbank abgeändert werden.

Es gibt verschiedenste SQL-Befehlsgruppen, welche jeweils einen anderen Zweck erfüllen.

SQL-Befehlsgruppen		
Database Manipulation (DML)	Database Definition (DDL)	Database Operation
DELETE	CREATE	SERVERERROR
INSERT	ALTER	LOGON/LOGOFF
UPDATE	DROP	STARTUP/SHUTDOWN

Mithilfe all dieser verschiedenen Gruppen ist es möglich, eine Datenbank zu erstellen, zu verwalten und zu löschen.

Um beispielsweise eine Tabelle in einer Datenbank zu erstellen, wird das Schlüsselwort `CREATE` verwendet, welches in der Data Definition Language gefunden werden kann. 26

Listing 26: CREATE table

```
1       CREATE TABLE animal (
2           animalName VARCHAR(20),
3           animalAge NUMBER(3)
4           );
```

Um nun eine Zeile in die Tabelle hinzuzufügen, wird ein INSERT-Statement benötigt. 27

Listing 27: CREATE table

```
1           INSERT INTO animal (animalName, animalAge) VALUES ('snake', 3);
```

Ein anderer wichtiger Befehl in SQL ist der SELECT-Befehl. Mit ihm kann aus jeder beliebigen Tabelle ein bestimmter Teil ausgegeben werden. Dabei können verschiedenste Kriterien übergeben werden, wie z.B. dass die auszugebenden Daten einen bestimmten Wert haben müssen.

4.10.2 PLSQL

4.10.3 Kommunikation mit Datenbanken

Um SQL in einer Datenbank zu verwenden, muss dazu aller erst eine Datenbank angelegt werden. Dabei gibt es verschiedenste Datenbanken, welche je nach Performance, bzw. benötigter Größe ausgewählt werden sollte. Siehe dazu Kapitel 4.6.

5 test

Siehe tolle Daten in Tab. 1.

Siehe und staune in Abb. 20.

Dann betrachte den Code in Listing 28.

Listing 28: Some code

```
# Program to find the sum of all numbers stored in a list (the not-Pythonic-way)

# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
    sum = sum+val

print("The sum is", sum)
```

	Regular Customers	**Random Customers**
Age	20-40	>60
Education	university	high school

Tabelle 1: Ein paar tabellarische Daten

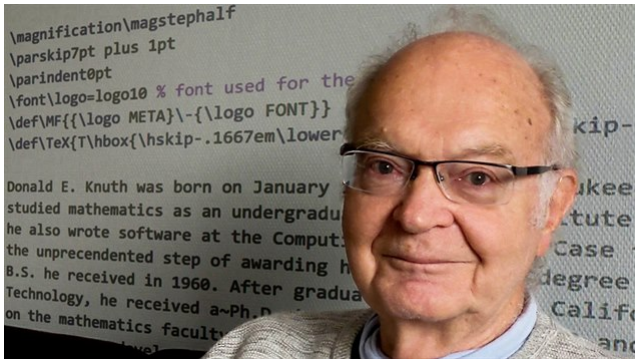


Abbildung 20: Don Knuth – CS Allfather

6 Zusammenfassung

Aufzählungen:

- Itemize Level 1
  - Itemize Level 2
    - Itemize Level 3 (vermeiden)

1. Enumerate Level 1
   a. Enumerate Level 2
      i. Enumerate Level 3 (vermeiden)

Desc Level 1

Desc Level 2 (vermeiden)

Desc Level 3 (vermeiden)

Literaturverzeichnis

[1] M. Mertens, „MAC Adresse: Definition, Funktion Und Einsatz.” 10 2021. Online verfügbar: https://www.placetel.de/ratgeber/mac-adresse

[2] „What is a Private IP Address?” Online verfügbar: https://whatismyipaddress.com/private-ip

[3] F. Becher und J. Steitz, „ISO/OSI-Referenzmodell,” S. 10–15, 1 2007. Online verfügbar: https://lernarchiv.bildung.hessen.de/sek/informatik/technisch/rechnernetz/allgemein/OSI_Ausarbeitung.pdf

[4] B. Grossmann, „ISO/OSI Schichtenmodell,” 2 2022. Online verfügbar: https://tech-tip.de/iso-osi-schichtenmodell/

[5] K. GmbH., „Modbus RTU Grundlagen.” Online verfügbar: https://www.kunbus.de/modbus-rtu-grundlagen

[6] ——, „Modbus.” Online verfügbar: https://www.kunbus.de/modbus

[7] „MODBUS over Serial Line Specification and Implementation Guide,” 12 2006. Online verfügbar: https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

[8] „Modbus – Wikipedia.” Online verfügbar: https://de.wikipedia.org/wiki/Modbus

[9] O. Weis, „ RS232 und RS485 Unterschiede | Serielle Protokolle im Vergleich,” 10 2020. Online verfügbar: https://www.virtual-serial-port.org/de/article/what-is-serial-port/rs232-vs-rs485.html#RS232

[10] „Modbus Illustration.” Online verfügbar: https://www.janitza.de/files/topics/wissen/Wissensdatenbank-Technischer-Anhang-Bilder/RS485-Schnittstelle-11.png

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang