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 S. Schwammal & S. Schwammal

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

1.3.4 Filezilla

1.3.5 Linux Terminal

1.3.6 Discord

1.3.7 Google drive

1.3.8 vs code

2 Umfeldanalyse

2.1 Algemeiner Technologie-Part

2.1.1 Java

2.1.2 JSON

2.1.3 HTML/CSS + Javascript

2.1.4 Typescript

2.1.5 Beckhoff

2.1.6 Raspberry

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 Ï-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.

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. Jede 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.

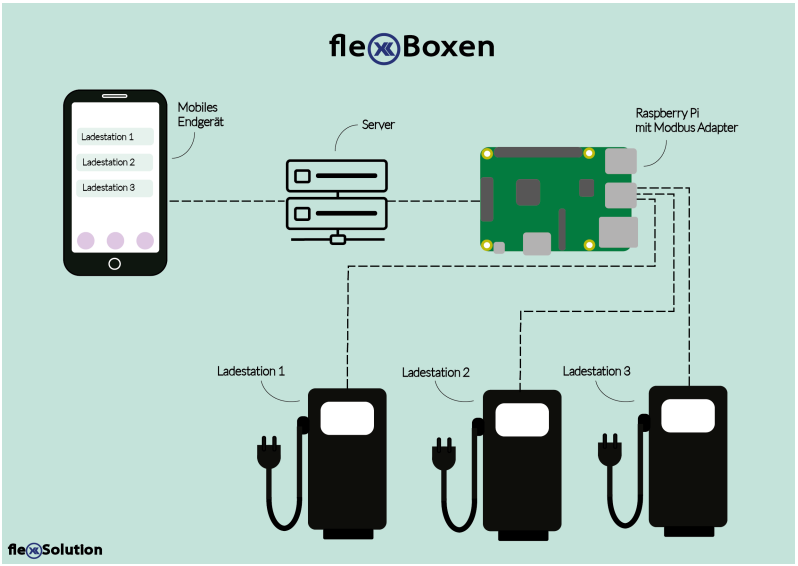

Abbildung 1: Darstellung des Aufbaues

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 ausgeschalten 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.BaudRate57600) -> 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 keinen signed, als kein Überwachungsbit gibt
- setStopBits(2) -> Auch dieser Parameter wurde vom Hersteller vorgeschrieben. Die Stopbits werden in einem späteren Kapitel (1.2) noch genauer erklärt

Der erste Teil des Tasks besteht aus einer for-Schleife, und einem Code Block, 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 Gerätsteht 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 der

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 Wertdirekt 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:

- 153 -> ist das Register, in welchem die Ansteckdauer gespeichert wird.
- 151 -> ist das Register, in welchem die Ladedauer gespeichert wird
- 126 -> ist das Register, in welchem der aktuelle Ladestromwert in Ampere 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 Datenpunkte, denn diese sind dafür da, um Values von anderen Tasks zu bekommen).

Der Dritte Part benützt, nicht so wie die ersten zwei Abschnitten, 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 Arte 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 aufrecht zu erhalten
- 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 kann. 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, das 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.

- dpsetChargingActive -> 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.
- dpChargingSlider -> Wenn die Wallbox eingeschalten 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.
- wallboxStatus -> 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 “Blau” und “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.

- wallboxLädtMitKW -> Dieses Array speichert den aktuellen Wert, mit welchem die Wallbox gerade lädt. Da bei den E-Autos meist die Kapazität des Akkus in kW/h angegeben sind, 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 "dpChargingSlider" Array, da bei letzterem der Wert direkt wieder zur UI geschickt wird, während "wallboxLädtMitKW" der Tatsächliche Wert der Wallbox ist!
- wallboxLadestromvorgabe -> 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 "ladedauerWb" einen 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 werden:

- getDataFromModbuswb -> 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 "Ansteckdauer" an der Stelle [x] (Device ID) gespeichert.
- ladedauerWb -> 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.

- aktuellerLadestromWertWb -> 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 “wallboxLädtMitKW” gepublished.
- cctGetLadestromvorgabeFromHmi -> Sobald die Wallbox eingeschalten ist (die Valie des Icons ist 2) bekommt der Task die Vorgabe des Sliders auf von 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) und an den Modbus Task geschickt. Der Wert wird außerdem in einem Array zwischengespeichert, um ihn beim Einschalten direkt an den Modbus-task mitzuschicken.
- cctGetStartChargingcommandWb -> 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 “wallboxLadestromvorgabe” mitgegeben. Außerdem wird die Variable “pressedwb” auf true gesetzt, da die Wallbox nun lädt. Sollte das Icon schon grün sein und somit die Wallbox schon eingeschaltet, wird bei einem weiteren Button-press das Icon auf Rot (aus) gestellt, und “wallboxLadestromvorgabe” 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 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 Main Thread 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

Die letzte und für den User die Wichtigste Komponente ist das Graphical user interface (kurz: GUI). Um den Nutzern in der Firma das Verwalten der Oberfläche so einfach wie möglich zu machen, hab wurde die Oberfläche mit den Firmeneigenen Systemen entwickelt. Die von der den Mitarbeitern entwickelte HMI (HMI steht für Human Machine Interface) funktioniert nur auf den internen Servern. Das Besondere ist, dass alle Oberflächen der Firma via Sockets verbunden sind, und somit alle Anzeigen zu jeder Zeit gleich sind. Der große Vorteil ist dadurch, dass sich alles Systeme miteinander verbinden lassen. Dies geschieht mit dem Flextask "HMI-Machine", der einzig dafür verantwortlich ist, sich auf neue Oberflächen zu subscriben, und ihr alle für sie bestimmten Daten zu schicken. Dies geschieht auch wieder über Datenpunkte, die auf die einzelnen Elemente der Oberfläche gemappt sind. Diese Elemente werden in einem großen JSON-Objekt zusammengestellt, konfiguriert, benannt und ausgerichtet.

Listing 1: 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    },
```

Man kann nun für diesen Button einen Datenpunkt anlegen, welcher in die Flexcloud published, wann jener Button gedrückt wird. Man kann dabei verschiedene Attribute hinzufügen bzw verändern. Beispiele sind:

- Type: gibt an, welchen Typ das Element besitzt. In diesem Fall ist es ein Button, der sich drücken lässt
- 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 "Sprüngen"
- Icon: dort kann man Icons von librarys wie zum Beispiel Font-Awesome anbinden. Die Icons können mit setzten von verschiedenen Werten die Farbe ändern.

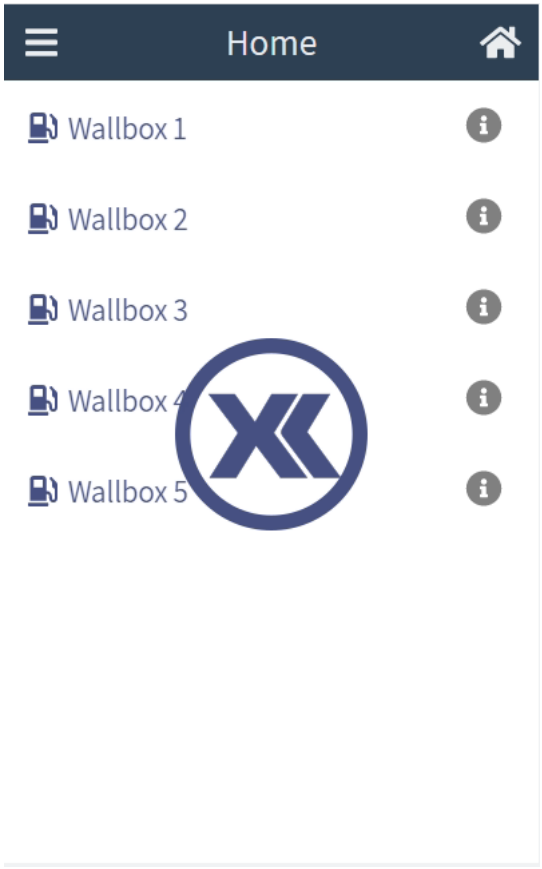

Abbildung 2: Übersicht über die fünf Wallboxen auf der HMI:

Die Oberfläche der App besteht aus einer Hauptseite, wo man eine Übersicht die 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

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 Status Element, 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

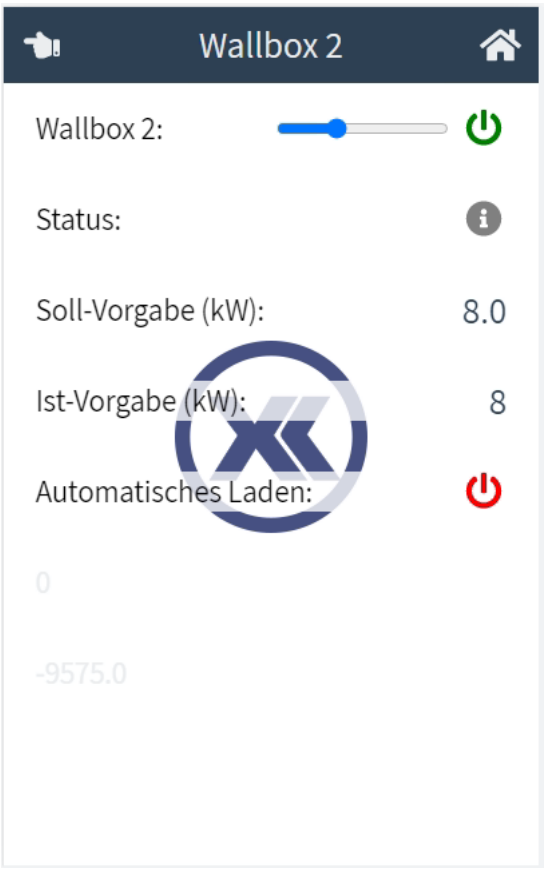

Abbildung 3: Detailansicht der Oberfläche für die Einzelnen Wallboxen

eingestellt hat. Er ändert sich dynamisch, und gibt dem Benutzer die Möglichkeit, die gewünschten kW/h genau einzustellen. Bei Verlassen bzw. Neu laden 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 welcher das Auto dann wirklich aufgeladen wird. Dieser ändert sich meist mit einer kleinen Verzögerung, da die Wallbox den neuen Wert erst nach wenigen Augenblicke übernimmt. Das Element mit dem Namen “Automatisches Laden” ist 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 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.

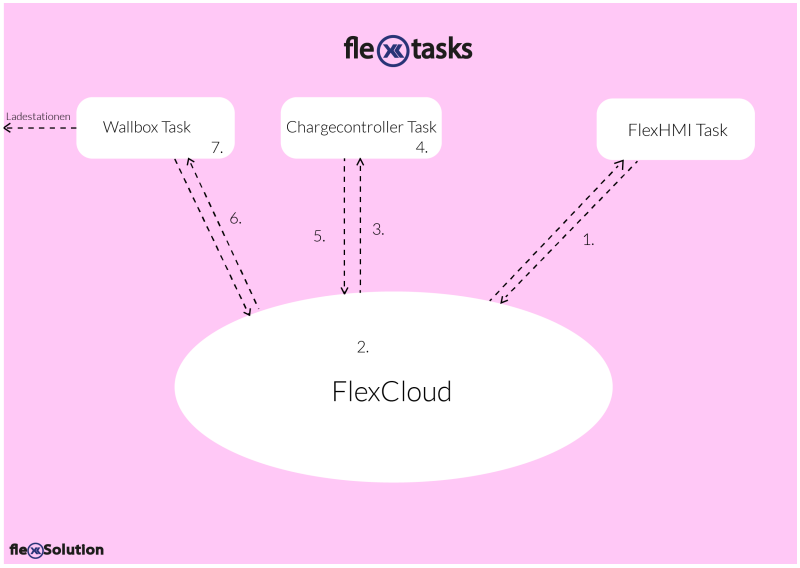

Abbildung 4: Darstellung des Aufbaues

3.4 FlexTasks in der Flexcloud

3.4.1 Was ist die Flexcloud?

Siehe Aufbau der Flexcloud: 4.

Die Flexcloud, auch FlexcommunicationCoud 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 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-Coud ist auch für viele kleinere Anwendungen nützlich, da sich solche Microservices sehr schnell aufsetzten lassen. Man kann einzelne Tasks aufsetzten, oder mehrere miteinander verbinden und kommunizieren lassen. Das hat den Vorteil, dass man wiederverwendbare Anwendungen entwickeln kann, welche nur grundliegende 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 einen oder mehreren Tasks Informationen oder Werte auszutau-

schen. Auch Funktionen-Aufrufe können durch solche Datapoints ermöglicht werden. Um durch Datenpunkte miteinander zu kommunizieren, müssen besagte FlexTasks mit den ProjectID´s verbunden werden. Wie man die Datenpunkte anlegen kann, was dabei wichtig ist und wie die Daten gemapped werden, wird im Kapitel (Datapoints) beschrieben.

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 “eu.flexsolution.task.” zu finden sein. Die GroupID ist dieselbe wie das Package der MainClass. In dem Fall ist es wieder “eu.flexsolution.task”. In der “artifactId” steht der Namen des Tasks. Bei den Dependencies wird der “Flexsolution core”, “log4j”, “org.json”. Eine weiter wichtige Sache ist das Einbinden des “Internal Snapshot Repository”. Das ist der Ort, an dem der Flexsolution core und die Flexlib gepublished werden. Maven lädt sich 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 “FlexTask” zu erweitern.

Listing 2: 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 die ein neues Objekt der Main Klasse erstellt.

Listing 3: Example Element

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

In der “initTask()” Methode kann dann die Hauptfunktion des Tasks implementiert werden. In den meisten Fällen wird hierfür Datenpunkte und andere Kommunikations-Methoden eingebunden. Mach 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 diesen 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) Der nächste Schritt ist im Verzeichnis “ /.local/share/systemd/user/”

ein File mit dem Namen des Services anzulegen: ‘gateway.service’. Dieses File muss wie folgt aufgebaut sein

Listing 4: Example Element

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

4. Da jeder Flextask eine Datenbank mit dem 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 Tables hinzugefügt:
   a. Device: besteht aus einer “ID” und einem Label. Im Falle des Gateways sind es 5 Devices, 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)” ausgelesen werden können. Die Tabelle besteht aus einer ID, einer DeviceID, einer Value und einem Label.
   c. taskParameter: hier können wichtige Parameter für das System gesetzt. Meistens wird hier nur der Standart-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. datapointValueMapping: Besteht aus einer Datapoint1Id und Datapoint2Id, einer Value und einer Spalte für neue Werte (newValue). Dies ist dafür da, um zwischen zwei Datenpunkte die Werte zu synchronisieren oder gegebenenfalls mit newValue zu überschreiben
   e. Datapoint: Herzstück der Datenbank. Hier werden die Datenpunkte angelegt (mehr dazu siehe 3.5). Die Tabelle besteht aus einer ID, einer deviceId, einem Label, einer “specificAddress”, einem “specificDatatype”, einem “specificStruct”, 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. datapointMap: Diese Tabelle besteht aus einer datapoint1Id, einer datapoint2Id und einer Funktion. Dieser Table 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, man kann das keepAlive Signal einstellen. Außerdem gibt es die PROJECTID, welche für das Verbinden mehrerer Tasks notwendig ist. Denn es können sich nur FlexTasks finden, welche dieselbe ProjectID besitzen. Im Falle des Gateways werden die Tasks mit der ID "AUT, HMI, chargecontroller" verbunden. Das ist wichtig, um den einzelnen Tasks die benötigten Parameter zu übergeben. AUT ist in dem Falle 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 "systemctl –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

==========

starting package maintenance... installation directory: /home/marcel/.miktex/texmfs/install package repository: https://mirror.lyrahosting.com/CTAN/systems/win32/miktex/tm/packages/ visiting repository https://mirror.lyrahosting.com/CTAN/systems/win32/miktex/tm/packages/... repository type: remote package repository loading package repository manifest... downloading https://mirror.lyrahosting.com/CTAN/systems/win32/miktex/tm/packages/miktex-zzdb1-2.9.tar.lzma... 0.31 MB, 102.88 Mbit/s package repository digest: 6edfdaad7fdb845c308709d75 going to download 69965 bytes going to install 43 file(s) (1 package(s)) downloading htt-

ps://mirror.lyrahosting.com/CTAN/systems/win32/miktex/tm/packages/latexindent.tar.lzma...
0.07 MB, 394.45 Mbit/s extracting files from latexindent.tar.lzma... ==========

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.
- deviceId: 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 deviceId 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 Datenpunkte, 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 “INT”, “REAL”, “BOOL” und “CMD” wählen.
- Intervall: Hier kann man ein Intervall, angeben, mit welchen der Datenpunkt in die Flexcloud gepushed 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 DB gespeichert. Man kann in der conf.db Datei einen Standard-Wert 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 5: 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 “SPECIFICADDRESS” angeben:

Listing 6: 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 7: 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 die 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()” aufrufen. Mit dieser Methode bekommt man immer den jetzigen Wert des Datenpunkt zurück.
3. Wie schon in den vorhergingen 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 8: 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 “datapointMap” 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 “datapoint1Id” zuerst das Label des Devices, zu welchem der Datenpunkt gehört, und anschließend die ID des ersten

Datenpunkt angibt. “datapoint2Id” 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 “ladedauerWb1” angelegt werden. Wird nun im Code auf diesen Datenpunkt was gepublished, sieht man “modbuswallbox.wb1.ladedauerWb1”. Damit die Daten jetzt auch ankommen, muss im Task für den Charge-controller ein Datenpunkt mit der ID “ladedauerWb” angelegt werden. In die “datapointMap” Tabelle muss dann bei der “datapoint1Id” das Device und der Datenpunkt angegeben werden, und bei der “datapoint2Id” zuerst der Task Name, das Device und dann der Datenpunkt, der die Daten pulished.

Listing 9: Example Element

```
1       dev1.ladedauer\textunderscore wb
            modbuswallbox.wb1.ladedauer\textunderscore wb1
```

3.6 Modbus

3.7 FlexHMI

3.8 Energiemanagement

3.9 Wallboxen

3.9.1 Java

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 eine 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 einen DataJDBCAdapter, einer Hauptklasse namens FlexTaskFsLogger. Weiteres befindet sich in dem Programm eine Modelklasse namens LogEntry, welche die Attribute der eingetragenen LogEntries definiert. Um die 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 abzuspei-

chern (insert). Um diese Verbindung herzustellen, verwendet der JDBCAdapter eine URL, welche in einem String abgespeichert ist, sowie einen Usernamen 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.

Das Haupmethode des FlexTaskFsLogger ist die initTask-Methode. In ihr wird eine BlockingQueue mit dem generic Type LogEntry erstellt. Mithilfe dieser Blocking-Queue wird 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 die 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 DataJDBCAdapter ausgeführt. Dabei wird der StringBuilder mit dem SQL-Statement, welche mithilfe der toString-Methode in einen String umgewandelt wird, übergeben.

Anschließend wird der eben verwendetet 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 man das Programm, also den Consumer und Producer aus- und einschalten kann. Dazu werden zuerst zwei Datenpunkte initialisiert. Diese werden durch das Setzen einer spezifischen Adresse und dem Namen eines Datenpunkts erstellt. Dabei ist der Datenpunkt `logger_nav_status_icon_Click` dazu verantwortlich, darauf zu hören, ob sich ein Wert, welcher in diesen 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-Statement

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 den genauen 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 Datenpunkte 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 zurückgeben 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 Übeprüfung herangezogen. Wenn dieses IF-Statement feststellt, dass das File nicht existiert, wird eine RuntimeExcpetiion 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 Resonse 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 wird eine Verbindung zur Datenbank aufgebaut und zurückgegeben, welche in den folgenden Methoden verwendet werden kann.

Listing 10: 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 LogEntrys in einem bestimmten Datums-Bereich 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 startMilllis 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 Erflog 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 Mal 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, erstellt. Danach wird eine Verbindung zur Datenbank mithilfe der connect-Methode hergestellt. Durch ein PreparedStatement, welches aus SQL-String geformt wird, könne 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 Ansicht

Die erste Seite, welche man betritt, wenn man das Programm startet, ist die Hauptseite 5. Auf ihr findet man 3 verschiedene Formulare. Jedes dieser Formulare hat eine andere Funktion. Das erste ist dafür zuständig, alle Diagramme in einem bestimmten Zeitraum anzuzeigen. Wenn man den Button dieses Formulars betätigt, wird man auf eine Seite weitergeleitet. Auf dieser kann nun zwischen allen Diagrammen mithilfe von Buttons wechseln 6. Das zweite Formular hat eine ähnliche Funktion. Lediglich kann man hier das Diagramm, welches man angezeigt bekommen will, im Formular mitübergeben. Durch den Button wird man nun direkt zum gewünschten Diagramm weitergeleitet.

Eine ganz andere Funktion hat das letzte Formular. In ihm wird zwar genauso der Wunschzeitraum angegeben, allerdings wird nach Betätigen des Buttons ein CSV-File

Abbildung 5: Website Hauptseitenansicht

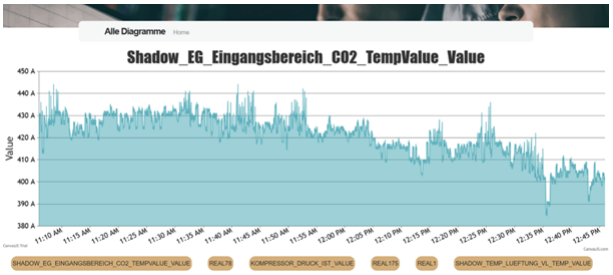

Abbildung 6: Website Diagrammansicht

generiert und anschließend gedownloadet. In diesem werden alle Daten übersichtlich angezeigt.

App-Component

In der TypeScript Klasse der App-Komponente wird ansich 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 angeben, 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 man eine Browser-app starten, sowie laufen lassen 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 verwenden.

Home-Component

Die ist die Haupt Komponente 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 Reactive 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 werden bei den Variablen jeweils ein Standard-Wert gesetzt. Die gesetzten Validatoren überprüfen dabei, ob die Daten korrekte Eingaben in den Formularen sind.

Weiteres findet man 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 Diagramm Optionen. 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 Diagramm Optionen. 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.

Ion 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 wenn die Länge des leeren Arrays 0 ist, wird der erste name hinzugefügt. Sonst wird eine weitere for-Schleife betreten, welche alle Elemente der List 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 die Diagramm Optionen gespeichert. Um dies umzusetzen, wird als Parameter ein String namens filterString übergeben. Mithilfe diesem 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 nicht undefiniert ist. Wenn dies der Fall ist, wird der erste Datenpunkt, sowie der Titel in den Diagramm Optionen gesetzt. Danach werden die restlichen Datenpunkte mithilfe einer For-Schleife in den Diagramm Optionen gespeichert.
- setChartOptions() Beim Aufrufen dieser Methode werden die Diagramm Optionen neu gesetzt. In diesem Fall wird 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 Backend 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 man sagen, dass man mit dem HttpClient jeweils einen GET-Request absetzt, welcher jeweils ein anderes Ergebnis liefert, je nachdem welche URL man als Parameter übergibt.

Die ersten beiden haben jeweils als Rückgabe-Paramater ein LogEntry Array. Beide geben die gesuchten Daten in einem bestimmten Zeitraum zurück, lediglich kann man bei der zweiten Methode noch einen Namen hinzufügen. 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 gedownloadet 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, um 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 Streams

4.4.2 Lambda

4.4.3 Serialisierung

4.4.4 Concurrency/Gleichzeitigkeit

4.4.5 Jenkins queue

4.4.6 Service executer

4.4.7 Java Futures

4.4.8 Java Completablefutures

4.4.9 First come first serve (Möglichkeiten Prozesse abzuarbeiten)

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 H2

http://www.h2database.com/html/performance.html

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

4.6.2 SQLite

https://www.sqlite.org/index.html

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

4.6.3 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

4.6.4 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 autcommit ausgeschalten wird. Die Perfomance 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 man die Derby Datenbank in jede JAVA-Applikation einbinden kann
- Supportet den Client/Server Modus
- Einfach zu installieren, einzurichten, sowie zu benutzen

Implementiert in Java.

4.7 Visuelle Darestellung

4.7.1 Angular

https://angular.io/

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 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 de 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 definiert.

Listing 11: Beipsiel für eine minimierte Angular Komponente

```
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'hello-world',
5     template: `
6       <h2>Hello World</h2>
7       <p>This is my first component!</p>
8     `
9   })
10  export class HelloWorldComponent {
11    // The code in this class drives the component's behavior.
12  }
```

4.7.2 Bootstrap

4.7.3 CanvasJS

4.8 Abspeicherung von Daten

4.8.1 Vergleich Vor und Nachteile JSON vs CSV vs Datenbank

4.9 Quarkus

https://quarkus.io

4.9.1 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 Serverlosen Umgebungen zu machen. Quarkus ist Open Source.

Die größte Challenge von Mikro Service Architekturen 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 Ressourcen Verwertung erweitert werden kann. Die Systeme regeln die zeitliche Planung und das Management der Mikro Services in einer dynamischen Weise. Dadurch kann man auch je nach Bedarf an dem System arbeiten, ohne das 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 PROPAGATION OF CHANGE zu verbinden.

4.9.2 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 7. 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 Klassen Hierarchien für ein Set von Klassen in eine Speicher-effizienten Repräsentation zu indexen.
- Gizmo: Gizmo ist eine Bytecode-Generation 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, welches die ahead-of-time (AOT) Kompilation von Java Applikationen von Java Programmen zu eigenständigen ausführbaren Programmen erlaubt.

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

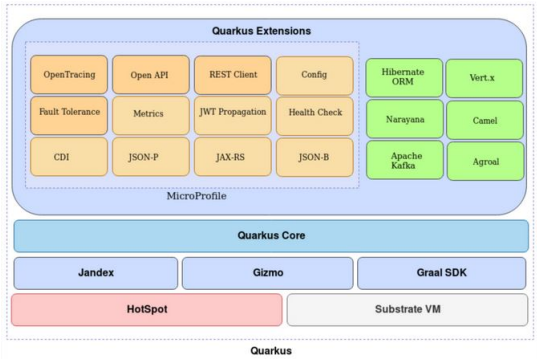

Abbildung 7: Quarkus Architektur

4.9.3 Funktionen

4.9.4 Vorteile gegenüber anderen Tools

4.9.5 NodeJS im Vergleich

4.9.6 http GET/POST/PUT Funktionen

4.9.7 JDBC + JPA

5 test

Siehe tolle Daten in Tab. 1.

Siehe und staune in Abb. 8.

Dann betrachte den Code in Listing 12.

Listing 12: 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

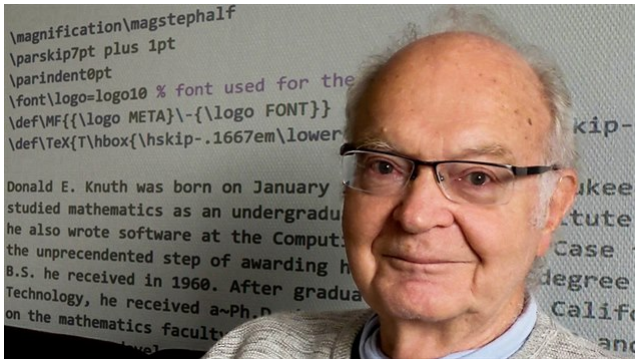

Abbildung 8: Don Knuth – CS Allfather

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

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang