htlperg
innovativ | kreativ | praxisnah

Implementierung eines Software-Oszilloskops in Angular

DIPLOMARBEIT

Höhere Abteilung für Informatik

01/10/2024 – 04/04/2025

Projektmitglieder: Jakob Bruckner
Julia Gillhofer
Alexander Nader

Betreuer: Prof. Mag. Michael Holzmann

Eidesstattliche Erklärung

Die unterfertigten Kandidaten / Kandidatin haben gemäß § 34 (3) SchUG in Verbindung mit § 22 (1) Zi. 3 lit. b der Verordnung über die abschließenden Prüfungen in den berufsbildenden mittleren und höheren Schulen, BGBl. II Nr. 70 vom 24.02.2000 (Prüfungsordnung BMHS), die Ausarbeitung einer Diplomarbeit mit der umseitig angeführten Aufgabenstellung gewählt.

Die Kandidaten / Kandidatin nehmen zur Kenntnis, dass die Diplomarbeit in eigenständiger Weise und außerhalb des Unterrichts zu bearbeiten und anzufertigen ist, wobei Ergebnisse des Unterrichts mit einbezogen werden können.

Die Abgabe der Diplomarbeit hat bis spätestens 04.04.2025 beim zuständigen Betreuer zu erfolgen.

Die Kandidaten / Kandidatin nehmen weiters zur Kenntnis, dass gemäß § 9 (6) der Prüfungsordnung BMHS nur der Schulleiter bis spätestens Ende des vorletzten Semesters den Abbruch einer Diplomarbeit anordnen kann, wenn diese aus nicht bei den Prüfungskandidaten / bei der Prüfungskandidatin gelegenen Gründen nicht fertiggestellt werden kann.

Danksagung

Wir bedanken uns bei sämtlichen Personen der HTL-Perg und KEBA, die diese Diplomarbeit ermöglicht haben.

Dabei möchten wir uns besonders bei unserem Diplomarbeitsbetreuer Herrn Professor Michael Holzmann bedanken, welcher uns tatkräftig beim theoretischen und praktischen Teil dieser Arbeit unterstützt hat.

Wir bedanken uns auch bei Fabian Schöppl und Markus Lengauer, welche diese Diplomarbeit und die Zusammenarbeit mit KEBA ermöglicht haben.

Ebenfalls bedanken wir uns bei unseren Eltern für das Korrekturlesen und die mentale Unterstützung.

Abstract

The aim of KEBA ChartView is to create a Software-Oscilloscope which fetches data from a programmable logic controller (PLC) for analytical purposes. The data is then displayed neatly in an individually configurable chart. The fetched data is captured by sensors on the machine itself. The Software-Oscilloscope is embedded in an already existing web application made by KEBA. This diploma thesis is split up into three different parts.

The first part is the middleware, which is responsible for the communication between the PLC and our project. Two different interfaces are used for communication. The first interface is a REST-API, which is responsible for the configuration of the PLC. The other interface is a WebSocket-API, which transmits the fetched data to our application. The middleware is, contrary to the other parts of the project, a pure TypeScript library and does not use any framework.

The data is displayed in a diagram - a chart. As part of this diploma thesis, six JavaScript chart libraries were compared and evaluated. The selected JavaScript chart library, µPlot, is used to display the data. The chart and the display of individual lines within the chart can be customized. Operation via mouse clicks and touch gestures allows for use in a browser as well as directly on an PLC.

The Software-Oscilloscope combines the middleware and the chart. The variable data which is fetched from the PLC is transformed and passed on to the chart. In the Software-Oscilloscope, it is possible to configure variables and data recorders. The configuration influences how the variables are displayed in the chart. Due to the use of a responsive design, the Software-Oscilloscope dynamically adjusts to the screen size. Therefore, a broad spectrum of devices can be used.

Zusammenfassung

Das Ziel von KEBA ChartView ist es, ein Software-Oszilloskop zu schaffen, welches Daten zu Analysezwecken aus einer Industriesteuerung (PLC) ausliest und übersichtlich in einem Chart anzeigt. Es ist möglich, die Anzeige der Daten individuell anzupassen. Die ausgelesenen Daten werden in der Maschine von Sensoren erfasst. Dieses Software-Oszilloskop ist in eine bestehende Web-Anwendung von KEBA eingebunden. KEBA ChartView ist in drei Teile gegliedert.

Der erste Teil ist die Middleware, welche dafür zuständig ist, die Kommunikation mit dem PLC zu verwalten. Es werden zwei verschiedene Schnittstellen zur Kommunikation verwendet. Eine ist eine REST-API, welche für die Konfiguration der PLC zuständig ist. Die andere Schnittstelle ist eine Websocket-Anbindung, über die die ausgelesenen Daten an unsere Anwendung übertragen werden. Die Middleware ist im Gegensatz zu den anderen Teilen eine reine TypeScript-Bibliothek und verwendet kein weiteres Framework.

Die Daten der PLC werden in einem Diagramm, einem Chart, angezeigt. Im Rahmen der Diplomarbeit wurden sechs JavaScript-Chart-Bibliotheken verglichen und evaluiert. Die ausgewählte JavaScript-Chart-Bibliothek µPlot wird verwendet, um die Daten anzuzeigen. Das Chart und die Anzeige der einzelnen Linien im Chart können individuell angepasst werden. Die Bedienung über Mausklicks und Touch-Gesten ermöglicht die Benutzung in einem Browser sowie direkt auf einer Industriesteuerung.

Im Software-Oszilloskop wird die Middleware mit dem Chart verbunden. Die Variablen-Daten, die von dem PLC bezogen werden, werden umformatiert und an das Chart weitergegeben. Im Software-Oszilloskop ist es außerdem möglich, Variablen und Datenrekorder zu konfigurieren. Diese Konfiguration beeinflusst, wie die Variablen im Chart angezeigt werden. Durch ein responsives Design passt sich das Software-Oszilloskop dynamisch an die Bildschirmgröße an. So können verschiedenste Geräte verwendet werden.

Inhaltsverzeichnis

1 Einleitung

In folgendem Kapitel wird die Entstehung und der Hintergrund dieser Diplomarbeit beschrieben und erklärt. Die Ziele und Erwartungen werden dargestellt.

1.1 Motivation

Zur Analyse von Variablen auf PLCs (siehe Abschnitt 2.1.3) gibt es bereits ein Software-Oszilloskop von KEBA. Dieses Software-Oszilloskop verwendet eine alte Schnittstelle zur Abfrage von Daten von einem PLC. Für verbesserte Sicherheit wurden von KEBA neue APIs zur Abfrage von Variablendaten der PLCs entwickelt. Diese APIs sollen in einem erneuerten Software-Oszilloskop zum Einsatz kommen. Dieses neue Software-Oszilloskop soll webbasiert sein und sich nahtlos in den DevAdmin (siehe Abschnitt 2.1.1) integrieren, um die Steuerung und Überwachung der PLCs sowie die Analyse der Variablen dieser PLCs in einem Werkzeug zusammenzufassen.

1.2 Zielsetzung

Ziel ist es, eine Anwendung in den DevAdmin (siehe Abschnitt 2.1.1) zu integrieren, welche dabei hilft, Daten, welche von Sensoren aufgenommen werden einfacher zu überwachen. Diese Anwendung soll aus drei Teilen bestehen.

Ein Teil ist die Middleware (siehe Abschnitt 3.5), welche eine TypeScript-Anbindung an den Datenrekorder (siehe Abschnitt 2.1.2) bietet.

Das Software-Oszilloskop (siehe Abschnitt 1.3.1) soll nahtlos in den DevAdmin eingebunden werden. Es soll dem Stil des DevAdmins entsprechen und alle Möglichkeiten zur Konfiguration der Datenrekorder zur Verfügung stellen.

Die Chart-Komponente (siehe Abschnitt 1.3.2) soll eine eigenständige Angular-Komponente sein, welche auch an anderen Stellen des DevAdmins benutzt werden kann. Diese Chart-Komponente soll es ermöglichen, mehrere Daten auf einmal in einem Diagramm anzuzeigen.

1.3 Projektinhalt

Der Projektinhalt kann in drei wichtige Komponenten unterteilt werden:

1.3.1 Software-Oszilloskop

Das Software-Oszilloskop ist eine Angular-Anwendung zur Visualisierung von Maschinendaten. Das Software-Oszilloskop ist in eine bestehende Anwendung von KEBA eingebunden. Daten, die von Sensoren von Maschinensteuerungen aufgezeichnet werden, können im Software-Oszilloskop in einem Chart angezeigt werden. Die Daten werden von der Steuerung über eine Middleware bezogen. Da das Format der Daten von der Middleware und das Format, welches das Chart benötigt, verschieden sind, werden die Daten zuerst transformiert.

Im Software-Oszilloskop ist es weiterhin möglich, diverse Konfigurationen für das Chart, die Variablen und die Datenrekorder vorzunehmen. Es wird ermöglicht, mehrere Datenrekorder zu erstellen. Bei jedem Datenrekorder können mehrere Variablen hinzugefügt werden. Auch ist es möglich, allgemeine Einstellungen wie zum Beispiel den Aufzeichnungsmodus oder die Aufzeichnungsgeschwindigkeit zu hinterlegen. Bei den Variablen ist es möglich, die Skalierung, Farbe, Linienstärke und Linienstile anzupassen. Welche Variablen angezeigt werden, ändert sich dynamisch, je nachdem, welcher Datenrekorder ausgewählt ist.

Durch ein responsives Design passt sich das Software-Oszilloskop automatisch an die Bildschirmgröße des Geräts, auf dem es zurzeit betrieben wird, an. Um auch auf kleineren Bildschirmen arbeiten zu können, ändert sich das Layout des Software-Oszilloskops bei definierten Bildschirmbreiten. Damit wird es Nutzern und Nutzerinnen ermöglicht, auf einem breiten Spektrum von Geräten arbeiten zu können.

1.3.2 Chart-Komponente

Die Chart-Komponente ist eine Angular-Komponente (siehe 2.1.6), die ein Diagramm (ein Chart) darstellt. Dem angezeigten Chart liegt eine JavaScript-Chart-Bibliothek zugrunde. Diese wurde über eine Evaluierung verschiedener JavaScript-Chart-Bibliotheken ausgewählt. Im Chart werden Variablen angezeigt und dem Benutzer zur Analyse zur Verfügung gestellt. Der Komponente können verschiedene Parameter übergeben werden, damit kann das Chart konfiguriert werden. Über die individuelle Gestaltung der angezeigten Variablen sowie eine Skalierung auf bestimmte Wertebereiche können Variablen im Chart im Kontext zueinander angezeigt und analysiert werden.

Die Chart-Komponente stellt das Modell für die Einstellungen der im Chart angezeigten Linien, die Konfiguration des Charts und die verschiedenen Anzeigemodi zur Verfügung. Um die Bedienung über den DevAdmin (siehe 2.1.1) sowie auf einem Steuerungsdisplay (siehe 2.6.2) zu ermöglichen, kann mit der Chart-Komponente per Mausklick sowie über Touch-Gesten interagiert werden.

Das Chart bietet mehrere Anzeigemodi, die das Verhalten des Charts bestimmen. Über eine Zoom-Funktionalität können Teilbereiche der angezeigten Variablen genauer betrachtet werden. Durch die Skalierung der einzelnen Variablen ist es möglich, auch Variablen mit sehr unterschiedlichen Wertebereichen nebeneinander darzustellen.

Die Chart-Komponente ist im Software-Oszilloskop eingebunden (siehe 1.3.1) und wird zur Anzeige von Variablen auf den Steuerungen genutzt, die über den DevAdmin verwaltet und gesteuert werden.

1.3.3 Middleware

Die Middleware ist eine TypeScript-Klasse, welche für die Kommunikation zwischen dem Software-Oszilloskop und der KEBA-Steuerung verantwortlich ist. Bei dieser Kommunikation wird auf der Seite des Software-Oszilloskops ein TypeScript-Interface verwendet. Die Konfiguration der verschiedenen Datenrekorder und Variablen wird über normale TypeScript-Methoden durchgeführt. Das Weiterleiten der Sensordaten wird mithilfe eines RxJS-Observables (siehe Abschnitt 2.4.1) realisiert. Auf der Seite der KEBA-Steuerung wird eine REST-API und eine Websocket-API benutzt. Dabei wird die REST-API für das Konfigurieren der Datenrekorder und der verschiedenen Variablen genutzt. Für die Übertragung der Sensordaten wird die Websocket-API verwendet.

Das grundlegende Modell der Datenrekorder, der Variablen und der Trigger wird ebenfalls von der Middleware bereitgestellt. Dieses Modell übernimmt das Datenmodell der verschiedenen APIs, wobei es einige grundlegende Änderungen gibt. Das Modell trennt die Auslöser von den Datenrekordern, um diese einfacher konfigurieren zu können. Die Middleware verändert außerdem das Format der verschiedenen Parameter, um es an den Standard in TypeScript anzupassen. Um mit der Websocket-API kommunizieren zu können, implementiert die Middleware das Schema einer Async-API-Definition. Die Middleware kümmert sich um den Verbindungsaufbau sowie um mögliche Probleme, die bei der Verbindung auftreten könnten. Wenn die Verbindung abreißt, baut die Middleware diese automatisch wieder auf. Es wird auch sichergestellt, dass die Verbindung richtig geschlossen wird, nachdem das Programm beendet wird.

1.4 Projektumfeld

1.4.1 Projektteam

Das Projektteam setzt sich aus Jakob Bruckner, Julia Gillhofer und Alexander Nader zusammen (siehe Abbildung 1). Alle Projektmitglieder sind Schüler und Schülerinnen der höheren technischen Bundeslehranstalt Perg. Jakob Bruckner nutzte sein vorhandenes Wissen über TypeScript und Websockets, um die Middleware zu gestalten. Julia Gillhofer übernahm die Implementierung der Chart-Komponente und nutzte dabei ihre Erfahrungen mit Angular. Alexander Nader setzte sein Wissen über User Experience und Design für die Implementierung des Software-Oszilloskop ein.

Abbildung 1: Jakob Bruckner, Julia Gillhofer, Alexander Nader (von links nach rechts)

1.4.2 Betreuung

Die Diplomarbeit wurde im technischen und theoretischen Teil durch Michael Holzmann, mit seiner detaillierten und aufmerksamen Betreuung, tatkräftig unterstützt.

1.4.3 Auftraggeber KEBA

Auftraggeber dieser Arbeit ist die Firma KEBA (Logo siehe Abbildung 2) mit Hauptsitz in Linz. KEBA ist ein international tätiges Technologieunternehmen mit 28 Niederlassungen in 16 Ländern und über 2000 Mitarbeitern und Mitarbeiterinnen. Ein Tätigkeitsbereich ist die Herstellung von Industriesteuerungen und der dazugehörigen Software. [1]

KEBA®
Automation by innovation.

Abbildung 2: KEBA Logo [1]

Die Betreuung bei KEBA wurde von Fabian Schöppl durchgeführt. Während der technischen Umsetzung wurde die Arbeit auch durch Markus Lengauer, der sich hauptsächlich mit dem DevAdmin (siehe Abschnitt 2.1.1) auseinandersetzt, unterstützt.

2 Grundlagen und Methoden

2.1 Grundlegende Fachbegriffe

Anschließend werden einige Fachbegriffe erklärt, die für diese Arbeit bedeutend sind.

2.1.1 DevAdmin

Der DevAdmin ist eine Webanwendung von KEBA, die den Fernzugriff auf Hardwaregeräte von KEBA ermöglicht und einen genauen Einblick in den Zustand des Geräts, aber auch die Möglichkeit zur Analyse dessen bietet. Dabei werden Informationen über den Zustand des Gerätes, wie etwa Performance-Daten zu Prozessoren und Arbeitsspeicherauslastung sowie generelle Daten wie die Seriennummer angezeigt. Der DevAdmin bietet auch eine Benutzerverwaltung. Aktuelle Lizenzen und Softwareversionen können eingesehen und erneuert werden. [2]

Der DevAdmin ist für verschiedene, Linux-basierte Hardwaregeräte von KEBA verfügbar. Um den DevAdmin zu erreichen, kann in einem Browser die IP-Adresse eines Geräts eingegeben werden. Dieses Gerät kann über den DevAdmin gesteuert sowie manuell gestartet und gestoppt werden. Zudem können weitere Geräte im selben Netzwerk angezeigt werden. [2]

Weitere Informationen zum Status der Hardware, wie die aktuellen Konfigurationen, verwendete Schnittstellen oder die am Gerät vorhandenen Variablen, können eingesehen und in einem Status-Report exportiert werden. Aktuelle Geräteinformationen helfen vor allem im Fehlerfall bei der Erkennung des Problems und bei der Analyse des Zustands des Geräts.[2]

2.1.2 Datenrekorder

Ein Datenrekorder ist eine Sammlung von Variablen, die augezeichnet werden können. Die Werte der Variablen werden von verschiedenen Sensoren aufgenommen und danach von allen Datenrekordern, welche diese Variablen beinhalten, aufgezeichnet. Diese Werte können danach gesammelt mittels einer Websocket-API (siehe Abschnitt 2.7.1) oder einer REST-API (siehe Abschnitt 2.7.2) abgerufen werden. Über diese Schnittstelle ist es ebenfalls möglich, die Datenrekorder zu konfigurieren.

Die Konfiguration bietet die Möglichkeit, neue Variablen bei einem Datenrekorder hinzuzufügen. Diese Variablen können auch gelöscht oder verändert werden. Zu jeder dieser Variablen können außerdem eigene Attribute gespeichert werden. Auch der Datenrekorder selbst kann konfiguriert werden. Es kann verändert werden, wie oft ein Datenrekorder neue Werte aufnimmt und wann er diese aufzeichnet. Datenrekorder können manuell oder durch einen Auslöser gestartet werden.

2.1.3 Programmable Logic Controller

Ein PLC repräsentiert industrielle Steuerungssysteme zur automatisierten Kontrolle von Fertigungsprozessen. Um sich als PLC zu qualifizieren, muss ein Steuerungssystem gewisse Voraussetzungen erfüllen. Es muss eine zentrale Prozesseinheit besitzen. Weiters müssen Eingangsmodule, welche digital oder analog sein können, und Ausgangsmodule vorhanden sein. Eine optionale Eigenschaft, die weit verbreitet ist, sind Kommunikationsschnittstellen sowie Speichermodule. [3]

Ein PLC kann auf verschiedene Weisen mit anderen PLCs verglichen werden. Beispielsweise kann die Zykluszeit der Programmausführung verglichen werden, um festzustellen, welcher PLC performanter ist. Es kann auch ein Vergleich der Speicherkapazität hergestellt werden, um herauszufinden, welcher PLC mehr Programme und Daten speichern kann. Weitere Leistungsindikatoren sind die Anzahl der Ein- oder Ausgangssignale, die Kommunikationsgeschwindigkeit und die Redundanzfähigkeit. [3]

2.1.4 HMI

Ein (HMI) ist eine Schnittstelle zwischen Mensch und Maschine, die es dem Menschen ermöglicht, mit einer Maschine zu interagieren. Dabei umfasst ein HMI alle Teile, die für die Interaktion zwischen Mensch und Maschine benötigt werden. Hardware für die Mensch-Maschinen-Interaktion umfasst unter anderem Touchscreens und Tastaturen für Eingaben, Displays für die Anzeige sowie Sensoren für die Erfassung von Gestik oder Sprache. Software für HMIs ermöglicht die Steuerung einer Maschine und unterstützt den Bediener und die Bedienerin, den Zustand der Maschine zu erkennen, um Fehler aufzuzeigen und auf diese reagieren zu können. HMIs werden für die Steuerung von Industriemaschinen genutzt und verbinden oft einen Menschen mit einer PLC (siehe Abschnitt 2.1.3). [4]

2.1.5 DOM

Das DOM ist eine Repräsentation für die verschiedenen HTML-Elemente (siehe Abschnitt 2.2.6). Diese Elemente sind in einer Baumstruktur angeordnet. Es ist möglich, durch diese Baumstruktur zu navigieren und eigene Elemente hinzuzufügen oder zu entfernen. Dadurch kann die Webseite dynamisch angepasst werden. Es kann auch die Formatierung der HTML-Elemente angepasst werden. Diese Manipulation wird bei dieser Arbeit mithilfe des Frameworks Angular (siehe Abschnitt 2.2.3) durchgeführt. [5]

2.1.6 Angular-Komponente

Eine Angular-Komponente ist ein Teil eines Angular-Programmes. Ein Bestandteil einer Angular-Komponente ist eine TypeScript-Klasse (siehe Abschnitt 2.2.4), welche die Logik und das Verhalten der Komponente bestimmt. Ein weiterer Bestandteil ist eine HTML-Schablone (siehe Abschnitt 2.2.6). Diese Schablone kontrolliert, welche HTML-Elemente im DOM (siehe Abschnitt 2.1.5) gerendert werden. Der letzte Bestandteil einer Komponente ist eine SCSS (siehe Abschnitt 2.2.5) Formatvorlage, welche den HTML-Elementen ein strukturiertes Aussehen gibt. [6]

Jede Angular-Komponente hat einen eindeutigen CSS-Kennzeichnung. Diese CSS-Kennzeichnung kann in anderen Angular-Komponenten verwendet werden, um an diese Stelle die Angular-Komponente einzufügen. Jede dieser CSS-Kennzeichnungen muss im ganzen Angular-Projekt eindeutig sein. [6]

Eine Angular-Komponente geht durch mehrere Lebenszyklus-Methoden. Die erste dieser Methoden ist die 'constructor'-Methode. Diese wird bei der Initialisierung der Komponente aufgerufen. Diese Methode ist der Standard-TypeScript-Konstruktor (siehe Abschnitt 2.2.4) und wird deshalb vor allen anderen Methoden aufgerufen. [6]

Die nächste Methode des Lebenszyklus ist die 'ngOnInit'-Methode. Diese wird aufgerufen, nachdem Angular alle Eingaben der Komponente initialisiert hat. In dieser Methode kann das Verhalten implementiert werden, welches für die Initialisierung der Komponente notwendig ist. Das können zum Beispiel RxJS (siehe Abschnitt 2.4.1) Abonnements sein. Um diese Methode implementieren zu können, muss das 'OnInit'-Interface von der Angular-Komponente implementiert werden. [6]

2.1.7 Angular-Service

Ein Angular-Service ist eine TypeScript (siehe Abschnitt 2.2.4) Klasse, welche in andere Angular-Services oder in Angular-Komponenten eingefügt werden kann. Diese Services sollten immer so klein wie möglich gehalten werden und nur eine Aufgabe erfüllen. Da ein Angular-Service in mehreren Komponenten eingefügt werden kann, ist es möglich, Code-Duplikation zu verringern. [7]

Ein typischer Anwendungsbereich für Angular-Services ist das Abrufen von REST-APIs (siehe Abschnitt 2.2.7). Der zweite wichtige Anwendungsbereich in unserer Anwendung sind RxJS (siehe Abschnitt 2.4.1) Observables, die benutzt werden, um andere Angular-Komponenten zu informieren, dass ein Ereignis geschehen ist. Diese Komponenten können dann auf diese Ereignisse reagieren, indem sie zum Beispiel die Anzeige aktualisieren (siehe Abschnitt 3.2.3). [7]

2.1.8 Module Federation

Module Federation ist ein Konzept, mit dem verschiedene Module zur Laufzeit zu einer Anwendung zusammengebaut werden. Module Federation wurde mit Webpack 5 vorgestellt und wird oft im Zusammenhang mit Micro-Frontends verwendet. Bei den Modulen wird zwischen lokal zur Verfügung stehenden Modulen und remote abgefragten Modulen unterschieden. [8]

Um ein Modul zu laden, wird ein Container benötigt, von dem das Modul geladen werden kann. Ein Container erlaubt den Zugriff auf ein Modul von außen. Dabei können die Container Module von anderen Containern verwenden und sich gegenseitig referenzieren. So können schrittweise immer größere Teile der Anwendung zusammengefasst werden. [8]

Module Federation wird unter anderem dafür verwendet, Teile einer SPA einzeln zu laden. Dabei werden die einzelnen Seiten einer SPA jeweils von einem Container zur Verfügung gestellt. Das ermöglicht eine flexible Zusammensetzung der einzelnen Teile einer SPA. Zudem werden mehrfach genutzte Bibliotheken als eigene Module angelegt, damit diese von mehreren Modulen gemeinsam genutzt werden können und in einer gebauten Anwendung nur einmal vorkommen. [8]

Die gemeinsame Nutzung von Bibliotheken kann auch dafür verwendet werden, um nur einen Teil der Anwendung neu bauen zu müssen, wenn die Version einer Bibliothek erneuert wird. Stehen die Bibliotheken als Module zur Verfügung, müssen nur diese Module neu gebaut werden.

Listing 1: Einbindung von Modulen

```
plugins: [
    new ModuleFederationPlugin({
      name: 'host',
      remotes: {
        app1: 'app1@http://localhost:3001/remoteEntry.js',
      },
    }),
  ]
```

Das Listing 1 zeigt die Konfiguration für das Einbinden von Modulen mit einem Plugin. Mit diesem wird der Name der Webanwendung als 'host' definiert. Diese Webanwendung bindet die Anwendung 'app1' als Remote-Modul ein. Dabei gibt die URL an, dass die Anwendung 'app1' über die Datei 'remoteEntry.js' über die angegebene URL ihren Code zur Verfügung stellt, der dann dynamisch von der 'host'-Anwendung geladen werden kann. [8]

2.1.9 Unit-Tests

Unit Tests sind Tests, welche nur einen bestimmten Teil der Anwendung testen. Alle anderen Aspekte der Anwendung werden simuliert. Dadurch kann ohne die Funktion von einem bestimmten Teil getestet werden. Durch diese Abkapselung sind Unit Tests nicht von anderen Teilen des Projekts abhängig. [9]

Wenn ein Teil des Programms verändert wird, helfen Unit-Tests dabei, Änderungen an der bestehenden Funktionalität zu finden. Wenn diese Unit-Tests bereits während der Entwicklung oder schon vor der Entwicklung geschrieben werden, können Fehler bereits während der Entwicklung gefunden werden. Dadurch, dass Fehler früher gefunden und beseitigt werden, wird die Entwicklung am Ende einfacher und schneller. [9]

Unit Tests können um schneller ausgeführt werden als normale Tests, da nur ein Teil des Programms getestet wird. Außerdem helfen Unit Tests dabei, die Funktionen und Reaktionen des Projektes zu dokumentieren. [9]

2.1.10 RESTful

Damit APIs als RESTful bezeichnet werden können, müssen diese bestimmte Kriterien erfüllen. Das erste Kriterium ist, dass die API eine Architektur mit Clients und Servern aufweist. Diese Server müssen verschiedene Ressourcen haben. Um diese Ressourcen zu verwalten, muss HTTP verwendet werden. [10] [11]

Das zweite Kriterium ist, dass die Kommunikation zwischen Server und Client zustandslos ist. Das heißt, es werden keine Informationen über die Verbindung oder über den Client beim

Server zwischen Anfragen gespeichert. Es müssen dadurch alle Anfragen separat erfolgen. Diese Anfragen können dadurch nicht verbunden sein. [10] [11]

Das dritte Kriterium ist, dass es möglich ist, die Daten beim Client zwischenzuspeichern. Dadurch wird die Kommunikation zwischen Server und Client optimiert, da ein Client nicht mehrmals dieselben Informationen abfragen muss. [10] [11]

Das letzte Kriterium ist, dass eine einheitliche Schnittstelle zwischen den verschiedenen Komponenten entsteht, welche Informationen in einem Standardformat überträgt. Damit das möglich ist, müssen vier Voraussetzungen gegeben sein. Die angeforderten Ressourcen müssen identifizierbar und von den an den Client gesendeten Darstellungen getrennt sein. Diese Darstellung muss genügend Informationen enthalten, um es dem Client zu ermöglichen, die verschiedenen Ressourcen zu manipulieren. Die Antworten an den Client müssen ebenfalls genügend Informationen enthalten, um es dem Client zu ermöglichen, diese selbst zu verarbeiten. Die letzte Voraussetzung ist, dass Hypertext oder Hypermedia verfügbar ist, um es dem Client zu ermöglichen, herauszufinden, welche möglichen Aktionen es gibt. [10] [11]

2.2 Verwendete Technologien

Folgend sind die Technologien, welche bei unserer Arbeit verwendet wurden, um die Anwendung zu erstellen.

2.2.1 Node.js

Node.js (kurz Node) (Logo siehe Abbildung 3) ist eine Umgebung, welche es ermöglicht, serverseitigen Code in JavaScript zu schreiben. Node läuft standardmäßig nur in einem Thread. Wenn eine blockierende Operation wie ein Netzwerkaufruf Zeit beansprucht, können in dieser Zeit Anfragen vom Server entgegengenommen werden, da Node diese Aufrufe asynchron durchführt und die Anfrage zu diesem Zeitpunkt unterbricht, um mit einer anderen zu beginnen oder fortzusetzen. [13]

Abbildung 3: Node.js Logo [12]

2.2.2 npm

NPM (Logo siehe Abbildung 4) dient dazu, Node-Pakete zu installieren, deinstallieren und deren Versionen zu verwalten. Diese Pakete werden in einer Registrierdatenbank, die von NPM verwaltet wird, gespeichert. Zugriff auf diese Datenbank besteht im Regelfall über ein Kommandozeilenprogramm.[15]

Abbildung 4: NPM Logo [14]

Die Pakete werden nach dem Installieren in einem eigenen Ordner im Projekt gespeichert, dem 'node_modules' Verzeichnis. Außerdem werden alle installierten Pakete mit deren Version in einer 'package-lock.json' Datei festgehalten. Dadurch ist es sehr einfach, dieselben Pakete auf verschiedenen Rechnern zu installieren. Für das Installieren der selben Pakete wird nur die 'package-lock.json' Datei benötigt.

Es gibt auch die Möglichkeit, selbst Pakete für diese Registrierdatenbank zu schreiben. Der Zugriff auf diese Pakete kann ebenfalls eingeschränkt werden, um es zu ermöglichen, zum Beispiel firmeninterne Pakete zu erstellen und einfach zu verbreiten. Bei Bedarf an Zugriffskontrolle muss jedoch ein monatlicher Betrag bezahlt werden. Für öffentliche Pakete ist kein Betrag zu bezahlen. Jedes dieser Pakete wird auf der Website von NPM dokumentiert (http://www.npmjs.com/). [15]

2.2.3 Angular

Angular (Logo siehe Abbildung 5) ist ein webbasiertes Framework, also eine Sammlung von Werkzeugen, APIs und Bibliotheken, welches dabei hilft, Webclients zu programmieren. Angular ist besonders für SPA geeignet. Das sind Applikationen, welche die Seite nicht neu laden. Es ist jedoch auch bei Angular möglich, auf einen anderen Pfad zu wechseln, dies funktioniert intern über einen eigenen Router. Der Angular-Router ermöglicht das dynamische Laden der verschiedenen Seiten, auch wenn es sich um eine SPA handelt. [17]

Abbildung 5: Angular Logo [16]

Ein weiterer Vorteil von Angular besteht in der Gliederung des Projektes. Das Angular-Projekt kann nicht nur in verschiedene Seiten wie bei einer Multi-Page-Applikation gegliedert werden, sondern auch in Komponenten und Services. Angular-Komponenten (siehe Abschnitt 2.1.6) sind abgekapselte Programmteile, welche ihre eigenen HTML- (siehe Abschnitt 2.2.6), SCSS- (siehe Abschnitt 2.2.5) und TypeScript- (siehe Abschnitt 2.2.4) Dateien haben. Diese Programmteile können in anderen Komponenten sowie der Hauptkomponente eingefügt und dupliziert werden. Angular-Services (siehe Abschnitt 2.1.7) werden nicht für die Visualisierung verwendet, sie haben auch keine HTML und SCSS Dateien. Ein Beispiel für so einen Service ist ein Datenbank-Service, welcher alle Abfragen an die Datenbank abhandelt. [6]

Um ein Angular-Projekt zu erstellen, wird das Angular CLI benötigt. Dieses CLI ermöglicht es auch, Komponenten und Services einfach in das Projekt einzubinden. Dieses CLI wird auch benutzt, um das Projekt zu kompilieren und zu testen. Angular verwendet NPM (siehe Abschnitt 2.2.2) für das Paketmanagement. [18]

Angular ist ein Open-Source-Projekt, welches von Google betreut wird. Google entscheidet, welche Änderungen in das Git-Repository (siehe Abschnitt 2.3.1) übernommen werden. Google benutzt Angular in einigen internen Projekten, wodurch sichergestellt wird, dass die Entwicklung von Angular nicht unterbrochen oder abgebrochen wird. Google selbst trägt auch zur Entwicklung bei, wobei der gesamte Programmcode Open-Source ist. [19] [20]

2.2.4 TypeScript

Für die technische Umsetzung dieser Arbeit wurde die Programmiersprache TypeScript (Logo siehe Abbildung 6) gewählt. TypeScript ist eine typisierte Web-Programmiersprache. TypeScript wird vor allem in webbasierten Projekten benutzt, da es die Standard-Programmiersprache des populären Frameworks Angular ist. [22]

Abbildung 6: TypeScript Logo [21]

In Angular (siehe Abschnitt 2.2.3) wird TypeScript verwendet, um die zentrale Logik zu erstellen. TypeScript wird in Angular-Komponenten (siehe Abschnitt 2.1.6) für die Logik benutzt und in Angular-Services (siehe Abschnitt 2.1.7) verwendet. RxJS (siehe Abschnitt 2.4.1) baut ebenfalls auf TypeScript auf.

TypeScript wird in JavaScript Code transpiliert, da alle gängigen Browser nur JavaScript unterstützen. Bei dieser Transpilierung werden auch die Typen geprüft, da diese Typen später nicht mehr vorhanden sind. Durch diese Eigenheit von TypeScript ist jeder gültige JavaScript Code auch gültiger TypeScript Code. Die Dateien, die nach dem Transpilieren entstehen, sind zwar in einer leserlichen Programmiersprache geschrieben, aber der TypeScript-Transpiler schreibt aus Optimierungsgründen keinen gut menschenleserlichen Code.

2.2.5 SCSS

SCSS (Logo siehe Abbildung 7) ist eine Erweiterungssprache, welche auf CSS aufbaut. Diese Programmiersprache ist, genauso wie CSS, optimiert für das Gestalten von Webanwendungen. SCSS wird in webbasierten Projekten genutzt. Im Gegensatz zu CSS ist es in SCSS möglich, Code zu verschachteln. SCSS verbessert CSS, indem die Lesbarkeit und die Schreibgeschwindigkeit der Entwickler und der Entwicklerinnen optimiert werden. [24]

Abbildung 7: SCSS Logo [23]

In Angular (siehe Abschnitt 2.2.3) kann SCSS verwendet werden, um in Angular-Komponenten (siehe Abschnitt 2.1.6) das Design festzulegen. Eine Besonderheit von SCSS-Code in Angular besteht darin, dass das Design an die Komponenten, die in anderen Komponenten referenziert werden, weitergegeben wird.

2.2.6 HTML

HTML (Logo siehe Abbildung 8) ist eine Auszeichnungssprache, welche die Struktur von Webseiten festlegt. HTML besteht aus verschiedenen Elementen. Diese sind, genau so wie in anderen Auszeichnungssprachen, in einem Baumsystem, dem DOM (siehe Abschnitt 2.1.5) aufgebaut. HTML-Elemente können als Struktur der verschiedenen Komponenten verstanden werden. Diese Elemente können dann im TypeScript (siehe Abschnitt 2.2.4) und SCSS (siehe Abschnitt 2.2.5) Code referenziert werden. [26]

Abbildung 8: HTML Logo [25]

In Angular (siehe Abschnitt 2.2.3) wird HTML gemeinsam mit TypeScript verwendet, um wiederholende Elemente einfacher implementieren zu können. Jede Angular-Komponente (siehe Abschnitt 2.1.6) bekommt ebenfalls ein eigenes CSS-Kennzeichen, dieses Custom CSS-Kennzeichen kann dann in anderen Komponenten verwendet werden. So wird die Verschachtelung der verschiedenen Komponenten vorgenommen. HTML wird direkt vom Browser interpretiert, was jedoch nicht bedeutet, dass diese Dateien nicht von Angular bearbeitet werden, bevor die Anwendung ausgeführt wird. Die Speichergröße der HTML-Dateien kann verringert werden, indem Leerzeilen und Absätze entfernt werden.

2.2.7 REST

REST ist eine Sammlung von Architekturbeschränkungen. Diese werden bei einer Web-API angewendet. Über eine API können Programme untereinander interagieren, da APIs menschenlesbar sind, ist es auch als Mensch möglich, mit ihnen zu interagieren. Diese API muss genau definiert werden, um sicherzustellen, dass die Kommunikation reibungslos verläuft. REST hilft dabei, indem es Beschränkungen erstellt, wodurch beide Kommunikationspartner von ein paar Standardverhalten ausgehen können. [10][11]

Eine API implementiert REST, wenn sie bei jeder Anfrage an die API unabhängig von vorherigen Anfragen antwortet. Das bedeutet, dass bei jeder Anfrage der aktuelle Status der Daten mitgegeben werden muss, da die API jede Anfrage einzeln behandelt. Durch diese Isolation ist es möglich, dass verschiedene Programme gleichzeitig mit der API kommunizieren. Wenn eine API die REST Architekturbeschränkungen implementiert, wird diese API als RESTful (siehe Abschnitt 2.1.10) bezeichnet. [10][11]

APIs kommunizieren standardmäßig mit HTTP, da durch das HTTP-Protokoll einfach der Zustand der Daten mitgegeben werden kann. HTTP ermöglicht es durch die HTTP-Methoden ebenfalls, einfach den Typ der Anfrage herauszufinden (GET, POST, PUT, DELETE, PATCH, ...). Meist wird bei einer REST-API JSON als Übertragungsformat gewählt, da dieses Format sowohl von Menschen als auch von Maschinen gelesen werden kann. [10][11]

2.2.8 WebSocket

WebSocket ist ein Kommunikationsprotokoll, mit welchem Daten übertragen werden können. Diese Daten werden über eine TCP-Verbindung übertragen. Da TCP eine Verbindung zwischen zwei Kommunikationsendpunkten herstellt, welche auch Sockets genannt werden, ist es möglich, in beide Richtungen zu kommunizieren. Es wird eine dauerhafte Verbindung aufgebaut, welche nicht nach jeder Anfrage geschlossen wird. [27]

Durch die dauerhafte Verbindung werden Wartezeiten verhindert, die Kommunikation beschleunigt und es ist möglich, Daten in Ëchtzeitäuszutauschen. Aus diesem Grund wird bei besonders performanten Anwendungen oft das WebSocket-Protokoll gewählt. Es können auch sehr viele Daten übertragen werden, wodurch ein weiterer Punkt für dieses Protokoll geschaffen wird. [27]

Im Vergleich zu einer REST-API (siehe Abschnitt 2.2.7) ist sichtbar, dass sich der Zustand des Servers nach einer Anfrage ändern kann. Dadurch muss ein besonderes Protokoll eingehalten werden, um sicherzustellen, dass sich der Server nie im falschen Zustand befindet. Deswegen ist das WebSocket-Protokoll nicht geeignet, um einzelne Daten zu übertragen. Es wird meistens verwendet, um viele Daten in kurzer Zeit zu übertragen. Wenn als Beispiel eine Social-Media-Anwendung gewählt wird, werden über WebSocket Bilddaten sowie einzelne Posts übertragen. Für den Login wird jedoch ein anderes Protokoll gewählt.

2.3 Verwendete Entwicklungssysteme

Um ein Softwareprojekt mit mehreren Personen effizient umzusetzen, bedarf es mehrerer Entwicklungssysteme. Diese Systeme bieten verschiedene Funktionen, die es ermöglichen, Code effizient zu entwickeln, auszuführen, zu testen und auszurollen.

2.3.1 Git

Als Quellcodeverwaltungssystem wurde Git (Logo siehe Abbildung 9) in Form eines GitLab-Monorepos (siehe 3.1) verwendet. Git ermöglicht es, dass mehrere Entwickler und Entwicklerinnen an einem Projekt arbeiten können, ohne sich gegenseitig in die Quere zu kommen.

Abbildung 9: Git Logo [28]

Git basiert auf einem verteilten System, bei dem Dateien auf einem Server, dem Repository, gespeichert werden. Alle haben einen aktuellen Stand des Repositories auf dem lokalen Computer. Wenn Dateien hinzugefügt, geändert oder gelöscht werden, muss dies mit dem Repository synchronisiert werden. Dadurch haben alle Zugriff auf diese Änderungen. Wenn zwei Personen dieselbe Datei geändert haben, müssen diese Änderungen manuell geprüft und eine Kombination beider übernommen werden. Dieser Prozess, lokale Änderungen mit dem Server zu synchronisieren, nennt sich Merging. Dies ist in Abbildung 10 zu sehen.

Zuerst muss das zentrale Repository einmalig auf den lokalen Rechner geklont werden. Dann können Änderungen in Dateien vorgenommen werden. Um diese Änderungen auch in das zentrale Repository zu bringen, müssen die gewünschten Änderungen in den Dateien ausgewählt werden. Diese werden dann mit einem Commit fixiert und danach mit dem Server synchronisiert (siehe Listing 2). Ein Commit ist ein Status des Repositories, auf den jederzeit zugegriffen werden kann. Das heißt, dass man auch auf ältere Versionen des Quellcodes Zugriff hat. Damit können zum Beispiel Fehler leichter gefunden werden.

Listing 2: Beispielhafter Auszug von GIT-Kommandos

```
1  git clone "<repository>"
2  git checkout -b feature
3  git add file1.txt
4  git commit -m "commit1"
5  git add directory1
6  git commit -m "commit2"
7  git push origin feature
8  git checkout master
9  git pull origin master
10 git merge feature
11 git push origin master
```

Damit die Programmierung besser verwaltet werden kann, bietet Git die Funktion von Branches (siehe Abbildung 10). Es kann von einzelnen Punkten im Repository, den sogenannten Commits, abgezweigt werden. Hier kann zum Beispiel experimentiert werden, ohne Einfluss auf die restlichen Entwickelnden zu haben. [29] [30]

Branches werden in Software-Teams auch im Zuge verschiedener Workflows eingesetzt. Ein Workflow ist eine vom Team definierte Prozedur, die beim Arbeiten mit Git eingehalten werden sollte. Bei vielen Workflows wird nicht direkt im Master-Branch des Repositories gearbeitet, sondern es werden eigene Branches erstellt. Danach wird der Branch meist von einer anderen Person, die sich zuvor die Änderungen überprüft und Feedback gibt, mit dem Master-Branch des Repositories synchronisiert (siehe Abbildung 10). Da die Feature-Branches nur eine kurze Lebensdauer haben, wird das Risiko von zeitaufwändigen Konflikten beim Synchronisieren minimiert. [31]

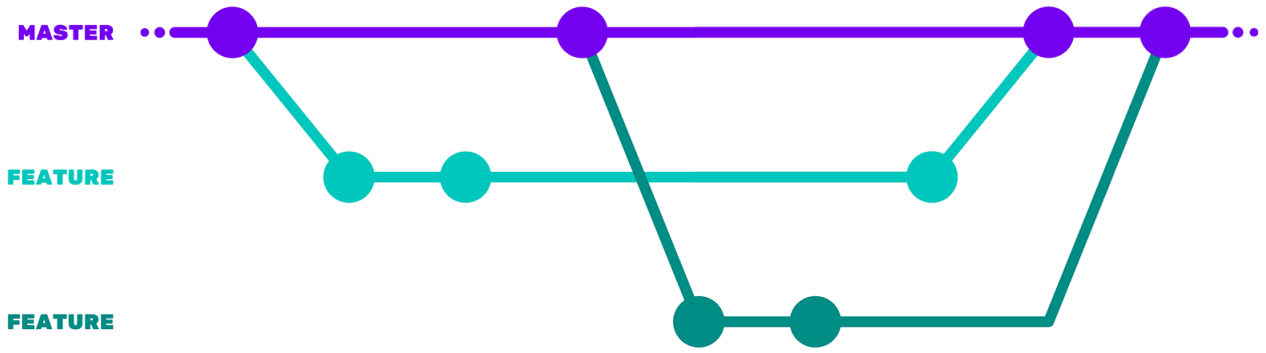

Abbildung 10: Beispielhafter Auszug eines Git-Repositories

2.3.2 Visual Studio Code

Visual Studio Code (Logo siehe Abbildung 11) ist eine sowohl für Privatpersonen als auch für Unternehmen kostenlose Programmierumgebung, mit der es möglich ist, Softwareprojekte in diversen Sprachen umzusetzen. Im Gegensatz zu traditionellen IDE fokussiert sich Visual Studio Code auf den Bearbeitungsteil des Programmierens. Sprachspezifische Werkzeuge, wie Debugger oder Generatoren, die in gewöhnlichen IDE direkt integriert sind, müssen mittels Erweiterungen nachinstalliert werden. Visual Studio Code ist sowohl für Windows als auch für Linux, macOS und im Web verfügbar. Visual Studio Code ist mit Electron programmiert. Damit ist es möglich, dass nur in einer Codebase, mithilfe von Web-Technologien, für alle Plattformen gleichzeitig entwickelt wird. [33] [34] [35]

Abbildung 11: VS Code Logo [32]

2.3.3 Docker

Abbildung 12: Docker Logo [36]

Docker (Logo siehe Abbildung 12) ist eine Plattform, die es Entwicklern ermöglicht, Anwendungen gekapselt auszuführen. Die Anwendungen werden in sogenannten Containern ausgeführt. Container sind isolierte Prozesse, das heißt, dass sie nur einen geringen Einfluss auf das Betriebssystem oder andere Container haben (siehe Abbildung 13). Dadurch werden potenzielle Konflikte, die zum Beispiel durch die Installation von mehreren Datenbanken entstehen können, umgangen. Im Gegensatz zu virtuellen Maschinen, die ein ganzes Betriebssystem virtualisieren müssen, läuft Docker meist auf einem Linux-Betriebssystem. Die verschiedenen Container greifen auf denselben Linux-Kernel zu. Dadurch sind Docker-Container ressourcensparender als virtuelle Maschinen. [37]

Da Linux-Container in Docker nur unter Linux verfügbar sind, muss, wenn diese Container unter Windows verwendet werden sollen, ein Linux-Kernel virtualisiert werden. Um diesen Prozess zu erleichtern, gibt es mit Docker Desktop unter Windows eine für Privatpersonen kostenlose Software, die alle nötigen Komponenten mitbringt, um Container auszuführen. Die kostenlose Version bietet jedoch nur eingeschränkte Funktionalitäten und ist für den nicht kommerziellen Nutzen ausgelegt. Unternehmen müssen die kostenpflichtige Version ($24 pro Monat, pro Nutzer, Stand Nov. 2024) kaufen. Die kostenpflichtigen Versionen bieten Funktionen, welche besonders für größere Teams und Unternehmen relevant sind. Beispiele dafür sind: GitHub-Integration, Single-Sign-On und private Marktplätze für Erweiterungen. [38]

Daten, die bei Docker-Containern anfallen, werden standardmäßig nicht persistent gespeichert. Wenn ein Container gestoppt wird, gehen alle Daten darin verloren. Besonders bei Datenbanken ist dies nicht gewünscht. Um zum Beispiel Daten von Datenbanken trotzdem zu persistieren, gibt es Docker-Volumes. Docker-Volumes sind vom Host-Betriebssystem isolierte Verzeichnisse, auf die Docker-Container zugreifen können. Bei der Erstellung von Containern können die zu verwendenden Volumes definiert werden. Dasselbe Volume kann in mehreren Containern verwendet werden, daher ist es möglich, Daten zwischen mehreren Containern zu teilen.[39]

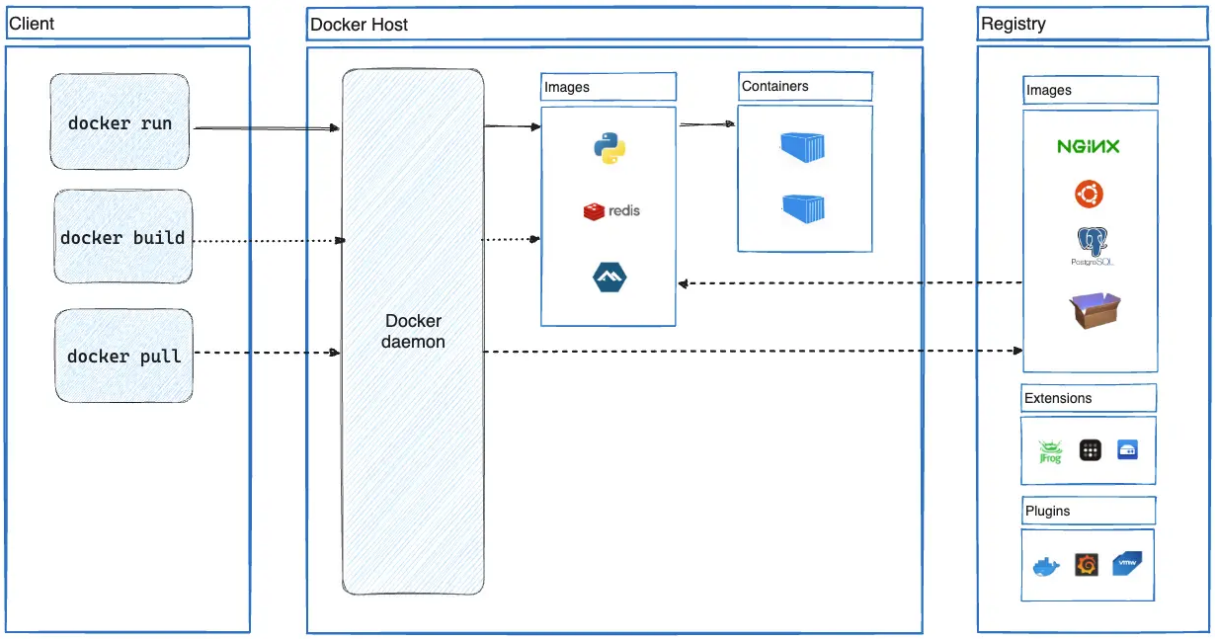

Abbildung 13: Architekturediagramm von Docker [40]

Docker wird in der Software-Entwicklung oft verwendet, um Entwicklungs-Container aufzusetzen. Entwicklungs-Container ermöglichen es, allen Mitgliedern dieselbe Entwicklungsumgebung zur Verfügung zu stellen. Damit wird sichergestellt, dass keine umgebungsspezifischen Fehler bei einzelnen Teammitgliedern auftreten. Wenn neue Projektteilnehmer am Projekt mitarbeiten, muss die Umgebung nicht von Anfang an neu aufgesetzt werden. Hier kann auf das bestehende Docker-Image zurückgegriffen werden (siehe Abbildung 13). Visual Studio Code (siehe Abschnitt 2.3.2) bietet direkt eine Funktion, die es ermöglicht, nahtlos in einem Entwicklungs-Container zu entwickeln, sofern einer konfiguriert ist.

2.3.4 WSL

Abbildung 14: WSL Logo [41]

WSL (Logo siehe Abbildung 14) ist eine Funktion von Windows, die es ermöglicht, Linux-Distributionen parallel zu Windows verwenden zu können. Diese Funktion ist besonders wichtig für Entwickler und Entwicklerinnen, die in Windows programmieren, aber trotzdem Programme wie Docker oder grep nutzen wollen, die nur unter Linux verfügbar sind. Nachdem WSL in Windows aktiviert worden ist, können Linux-Anwendungen in der Konsole verwendet werden. Mit WSL ist es möglich, unter Windows und Linux mit denselben Dateien zu arbeiten. [42] [43]

Damit Windows und Linux gleichzeitig auf einem Rechner laufen können, müssen die Betriebssysteme virtualisiert werden. Dafür wird der Typ-1-Hypervisor Hyper-V verwendet. Hyper-V läuft direkt auf der Hardware des Systems und virtualisiert einen Linux-Kernel (siehe Abb. 15). Da direkt auf der Hardware virtualisiert wird, ist die Linux-Installation performanter, als wenn sie durch einen Typ-2-Hypervisor, der unter Windows läuft, virtualisiert werden würde.

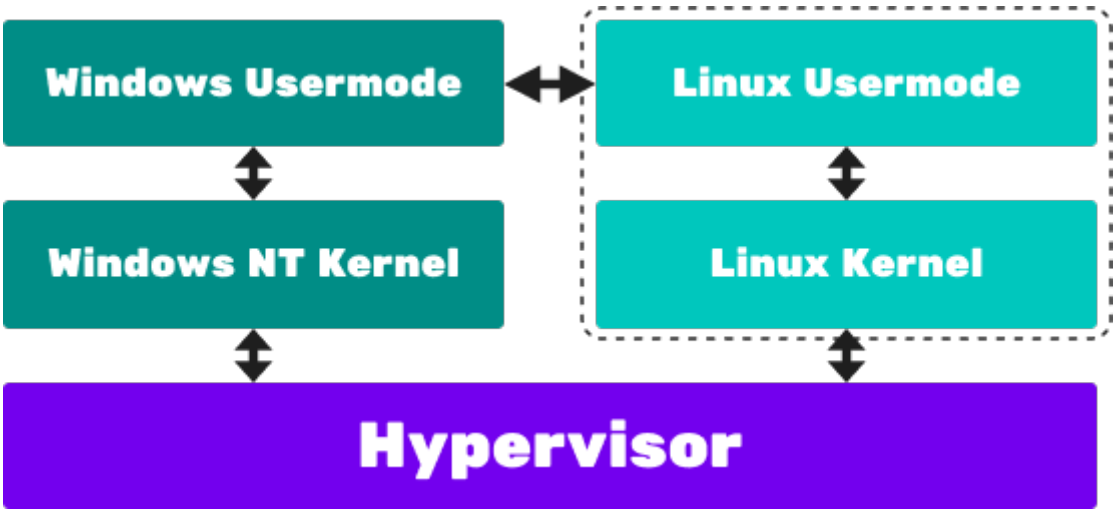

Abbildung 15: Architekturediagramm von WSL

2.3.5 Chrome DevTools

Die Chrome DevTools (Logo siehe Abbildung 16) sind eine Reihe an Werkzeugen für die Webentwicklung, die im Chrome-Browser integriert sind. Mit diesen DevTools ist es möglich, HTML, CSS und den JavaScript-Code einer Webseite direkt im Browser zu ändern. Das ermöglicht ein schnelles Ausprobieren von Ideen. Zudem bieten die DevTools hier Künstliche Intelligenz als Unterstützung an. [44]

Abbildung 16: Chrome DevTools Logo [44]

Neben dem Ändern einer Webseite bieten die Chrome DevTools weitere Möglichkeiten für Entwickler, das Verhalten und die Leistung der Webseite zu untersuchen. Über den Netzwerk-Tab können ausgesendete und empfangene Pakete angesehen und inspiziert werden. Für ein Paket können die gesetzten Header, der Inhalt des Pakets, die eingetroffene Antwort, die benötigte Zeit, der Auslöser für das Abschicken des Pakets und die gespeicherten Cookies angesehen werden. Außerdem kann nach bestimmten Paketen gefiltert und sortiert werden. [45]

Über den Applikationen-Tab können die Cookies, der lokale Speicher und der Speicher der aktuellen Session angesehen werden. So kann genau überprüft werden, was die Webseite gerade

wo speichert und welchen Wert die gespeicherten Variablen annehmen. Über den Konsolen-Tab können Ausgaben in der Konsole angesehen werden. Hier werden auch Fehler angezeigt, wenn diese ausgegeben werden. Neben der Ausgabe von Informationen kann auch JavaScript-Code im Konsolen-Tab ausgeführt werden, um das Verhalten der Webseite zu ändern. [46] [47]

Im Speicher-Tab kann ein Einblick in die Speichernutzung der JavaScript-Objekte der Webseite gewonnen werden. Über Snapshots wird der aktuelle Speicherstand angezeigt und welche Objekte wie viel Speicher benutzen. Zudem kann der Speicherverbrauch über einen Zeitraum aufgezeichnet und analysiert werden. Über eine Stichprobe kann der voraussichtliche Speicherverbrauch abgeschätzt werden. [48]

Im Leistungs-Tab werden die LCP, die CLS und die INP angezeigt. LCP gibt die benötigte Zeit an, um das größte für den Benutzer sichtbare Element zu laden, wie etwa ein Bild. Dabei kommt es nicht auf die eigentliche Bildgröße an, sondern darauf, welches Element den größten Teil des für den Benutzer sichtbaren Bereichs abdeckt, da LCP die Ladezeit des Hauptinhalts einer Webseite bestimmt. Dieser Zeitwert wird mit einer Bewertung angegeben. Der Zielwert für diese Zeitwerte liegt bei 2.5 Sekunden. CLS gibt die Anzahl der plötzlichen Veränderungen der Struktur einer Webseite an. Das kann passieren, wenn ein Text auf der Webseite die Position wechselt oder wenn ein zusätzlich eingeblendetes Element die anderen Elemente verschiebt. Diese CLS können den Benutzer verwirren. Der INP gibt die längste benötigte Zeit an, um auf eine Benutzeraktion wie etwa einen Klick zu reagieren. [49]

Neben den direkt angezeigten Analysen im Leistungs-Tab können Aufnahmen während der Benutzung der Webseite erstellt und analysiert werden. Für eine Aufnahme kann unter anderem eingestellt werden, ob neben der INP auch Ladezeiten weiterer Inhalte aufgezeichnet und angezeigt werden sollen. Zudem kann das Verhalten der Webseite mit simulierten CPU- und Netzwerk-Ressourcen getestet werden. So kann man etwa die CPU verlangsamen, die Ladezeiten innerhalb eines 3G-Netzwerks beobachten oder den Internetzugang verhindern. [49]

Im Rendering-Tab können Einstellungen vorgenommen werden, um etwa Bereiche der Webseite hervorzuheben, die neu gezeichnet werden müssen, Werbungen hervorzuheben, automatisch in den Dark-Mode zu wechseln oder die FPS-Rate anzuzeigen. [50]

2.3.6 Storybook

Storybook (Logo siehe Abbildung 17) ist ein Werkzeug, mit dem einzelne Komponenten einer Webanwendung getestet und dokumentiert werden können. Mit Storybook muss nicht eine ganze Anwendung geladen werden, um einzelne Komponenten zu rendern und das Verhalten der Komponente über ihre Eingabeparameter zu testen. [52]

Abbildung 17: Storybook Logo [51]

Für jede Komponente können verschiedene Stories erstellt werden. Eine Story ist ein gerenderter Zustand einer Komponente, der einen vorgesehenen Stand dieser Komponente darstellt. Abbildung 18 zeigt eine Story für einen Button.

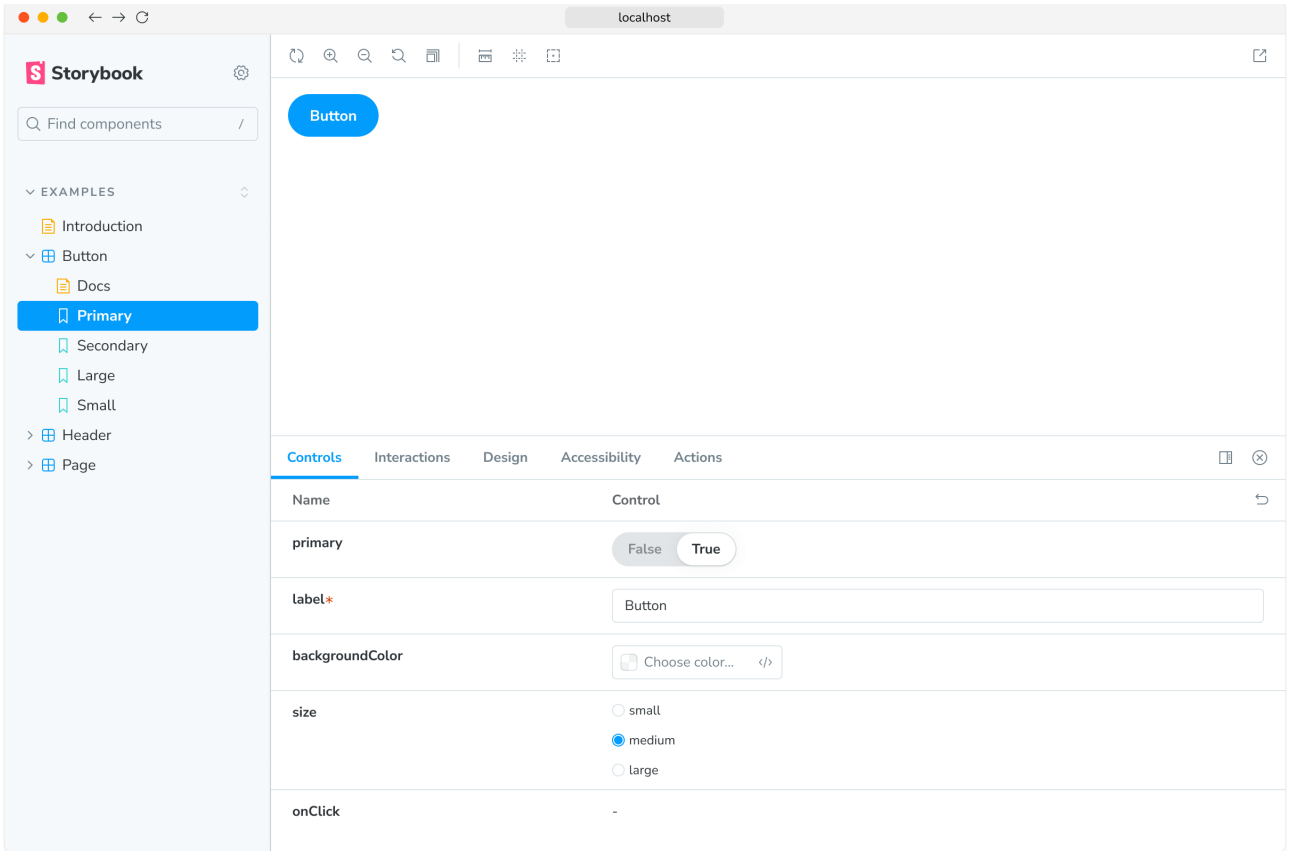

Abbildung 18: Story für einen Button [53]

Für diesen Button wurden vier Stories erstellt. Die gezeigte Story hat den Namen 'Primary' und zeigt einen Button, der im Primärzustand gerendert wurde. Für diesen Button können weitere Parameter geändert werden, um das Verhalten des Buttons zu testen, wie etwa den im Button angezeigten Text.

Mit Stories können Entwickler und Entwicklerinnen Funktionalitäten und Verhaltensweisen von Komponenten getrennt von der gesamten Anwendung testen. Gleichzeitig werden die möglichen Ausprägungen einer Komponente, die benötigten oder möglichen einstellbaren Parameter und deren Wertebereiche dokumentiert.

2.3.7 NX

Abbildung 19: NX Logo [14]

NX (Logo siehe Abbildung 19)ist ein Entwicklungs-Werkzeug, welches die ständige Integration eines Monorepos (siehe Abschnitt 3.1) vereinfacht. Da bei einem Monorepo sehr viel Code anfällt, welcher nicht mehr verändert wird und darum auch nicht mehr getestet werden muss, hilft NX dabei, diesen nicht mehr bauen und testen zu müssen. NX speichert den letzten Build in einem Cache. Dieser Cache wird dann bei etwaigen Builds aufgerufen. [54]

NX baut auf einer Versionsverwaltungssoftware wie Git (siehe Abschnitt 2.3.1) auf, um festzustellen, wo es Änderungen gibt und wo der Cache benutzt werden kann. NX ist auch kompatibel mit der Git-Pipeline, welche oft zum Testen von Monorepos verwendet wird. Es ist auch möglich, mittels NX die Last auf mehrere Geräte zu verteilen.

2.4 Verwendete Bibliotheken und Plug-Ins

Um die Entwicklung von Softwareprojekten zu vereinfachen, werden diverse Bibliotheken verwendet. Um möglichst effizient zu programmieren, kann es Sinn machen, Funktionen, die schon jemand zuvor implementiert hat, zu verwenden. Plug-Ins erweitern die Funktionalitäten von bestehenden Systemen. Damit können die bestehenden Systeme an die Bedürfnisse des Nutzers angepasst werden.

2.4.1 RxJS

RxJS (Logo siehe Abbildung 20) ist eine Bibliothek, die es ermöglicht, asynchron und ereignisorientiert zu programmieren. Das heißt, dass die Ausführung dieser Codeblöcke den Hauptfluss des Programms nicht unterbricht. Angestoßen werden diese durch diverse Ereignisse. RxJS bietet Funktionalitäten, mit denen es möglich ist, effizient mit asynchronen Datenströmen zu arbeiten. Um RxJS zu verwenden, muss diese Bibliothek mittels NPM (siehe Abschnitt 2.2.2) mit 'npm install rxjs' zuerst installiert werden. In Angular-Projekten (siehe Abschnitt 2.2.3) ist RxJS bereits standardmäßig vorinstalliert. [56] [57]

Abbildung 20: RxJS Logo [55]

RxJS-Observables sind Collections, in welche mehrere Werte gegeben werden können. Auf Änderungen in dieser Collection kann reagiert werden. JavaScript bietet standardmäßig die Funktion von Promises, diese funktionieren ähnlich wie Observables. Der Unterschied ist, dass Observables mehrere Werte speichern können. Observables und Promises verwenden das Push-Protokoll. Das heißt, dass der Produzent der Daten entscheidet, wann Daten gesendet werden. Der Konsument der Daten kann dann auf diese reagieren. Das Gegenstück zum Push-Protokoll ist das Pull-Protokoll. Beim Pull-Protokoll entscheidet der Konsument, wann Daten gesendet werden. Der Konsument sendet eine Anfrage an den Produzenten, welcher dann eine Antwort zurücksendet. Ein Beispiel für das Pull-Protokoll, bei dem ein Wert gesendet wird, ist eine normale JavaScript-Funktion. Ein Beispiel für das Pull-Protokoll, bei dem mehrere Werte gesendet werden können, ist ein Iterator. Mit Iteratoren kann man über mehrere Werte iterieren. Dadurch kann man mehrere Werte aus einer Funktion bekommen. [58] [59] [60]

Observer werden verwendet, um Daten zu konsumieren. In RxJS sind Observer Objekte mit drei Callback-Funktionen: 'next', 'error' und 'complete'. Mit 'next' kann auf den nächsten Wert reagiert werden. Mit 'error' kann auf potenzielle Fehler, die auftreten, reagiert werden.

Und mit 'complete' wird signalisiert, dass die Datenreihe fertig ist. Das Observer-Objekt kann jede Kombination dieser Callback-Funktionen enthalten. Wenn ein Observable abonniert wird, kann ein Observer-Objekt mitgegeben werden. Dieses Objekt definiert, wie auf Werte des Observables reagiert werden soll. Alternativ können auch direkt die benötigten Callback-Funktionen mitgegeben werden. [61]

Um mit Datenreihen arbeiten zu können, bietet RxJS eine Reihe von Operatoren, mit welchen Observables erstellt und manipuliert werden können. Mit der pipe-Funktion können manipulierende Operatoren zu einem Observable hinzugefügt werden. Die pipe-Funktion gibt ein neues Observable zurück. Auf alle Werte im ursprünglichen Observable wird der Operator angewandt. Das resultierende Observable kann wieder abonniert werden. Um aus den vielen Operatoren genau den Richtigen zu finden, gibt es unter "https://rxjs.dev/operator-decision-tree"(siehe Abb. 21) einen vom RxJS-Team erstellten Entscheidungsbaum. Wenn auf dieser Seite einige Fragen beantwortet werden, kommt am Ende ein Operator heraus, mit dem die gewünschte Funktionalität implementiert werden kann. [62] [63]

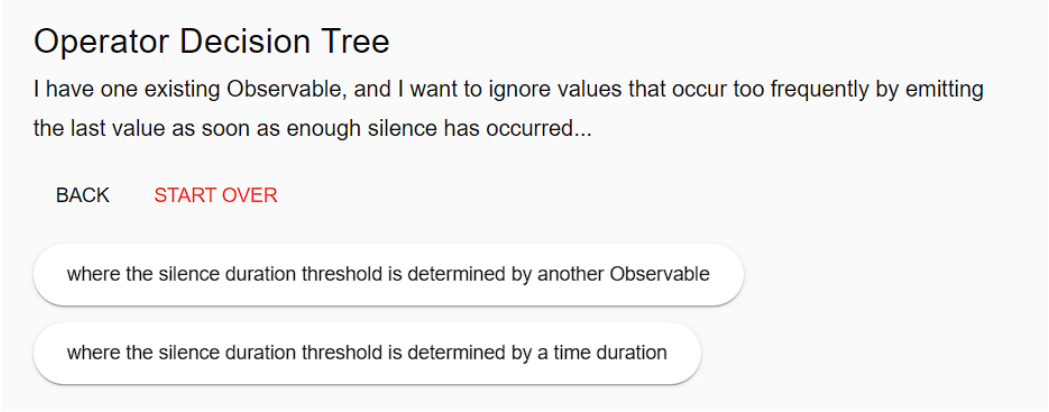

Abbildung 21: Screenshot vom RxJS-Operator-Entscheidungsbaum [63]

Eine weitere Form von Produzenten von Daten sind RxJS-Subjects. RxJS-Subjects sind sehr ähnlich zu Observables, der einzige Unterschied ist, dass ein Subject von mehreren Observern abonniert werden kann. Daher können mehrere Observer auf dieselben Ereignisse reagieren. Eine besondere Form von Subjects sind BehaviourSubjects. BehaviourSubjects speichern keine Reihen an Daten, sondern immer nur einen einzelnen Status. [64]

RxJS wird zum Beispiel bei REST-API-Aufrufen (siehe Abschnitt 2.2.7) verwendet. Nach einem API-Aufruf ist es nicht klar, wie lange die Antwort dauern wird. Daher ist eine asynchrone Programmierung notwendig. Der HTTP-Client von Angular (siehe Abschnitt 2.2.3) gibt zum Beispiel bei einem HTTP-Aufruf nicht das direkte Resultat, sondern ein RxJS-Observable zurück. Dieses Observable muss abonniert werden, damit der HTTP-Aufruf zustande kommt. [65]

2.4.2 Reactive Forms

Reactive Forms sind ein Angular-spezifischer Ansatz, um Eingabefelder in Anwendungen, die mit Angular erstellt werden, kontrollieren zu können. Mit Reactive Forms können diese Eingabefelder überprüft, Werte umgesetzt und das Verhalten eines gesamten Formulars gesteuert werden. Der aktuelle Stand eines Formulars - mit den Werten aller Eingabefelder, die ein Teil des Formulars sind - kann über ein RxJS-Observable (siehe Abschnitt 2.4.1) abgefragt werden. Dieses Observable kann synchron abgefragt werden. Verändert sich der Zustand des Formulars, wird das bestehende Objekt, welches das Formular repräsentiert, nicht verändert. Stattdessen wird ein neues Objekt erstellt. [66]

Listing 3: FormControl

```
name = new FormControl('');

updateName() {
    this.name.setValue('Max');
}
```

Im Listing 3 wird ein neues FormControl-Objekt angelegt und eine leere Zeichenkette als Initialwert gesetzt. In der 'updateName'-Methode wird der Wert von 'name' auf 'Max' gesetzt. [66]

Listing 4: Eingabelfelder mit FormControl

```
<label for="name">Name: </label>
<input id="name" type="text" [formControl]="name">
```

Im Listing 4 wird ein Eingabefeld für Texteingaben in HTML definiert und das im Listing 3 angelegte FormControl-Objekt zugewiesen. So kann nun in TypeScript (siehe Abschnitt 2.2.4) der Wert des Eingabefeldes gelesen und verändert werden. Zudem kann über das FormControl-Objekt auf Veränderungen im Eingabefeld reagiert werden. [66]

Mehrere FormControl-Objekte können zu einer Gruppe, einer FormGroup, zusammengefasst werden. Dadurch kann ein Formular abgebildet werden. FormControl- und FormGroup-Objekte können jeweils ineinander verschachtelt werden.

Listing 5: Verschachtelungen

```
profileForm = new FormGroup({
    firstName: new FormControl(''),
    lastName: new FormControl(''),
    address: new FormGroup({
      street: new FormControl(''),
      city: new FormControl(''),
      state: new FormControl(''),
      zip: new FormControl(''),
    }),
  });
```

Im Listing 5 wird ein FormGroup-Objekt angelegt. Dieses besitzt die Attribute 'firstName' und 'lastName'. Innerhalb des FormGroup-Objekts ist ein weiteres FormGroup-Objekt verschachtelt, welches die Adresse darstellt. So können über mehrere Ebenen auch verschachtelte Formulare abgebildet werden. [66]

2.4.3 KEBA Webcomponents

Die KEBA-Webcomponents sind eine KEBA-interne UI-Bibliothek, welche oft verwendete UI-Komponenten standardisieren. KEBA-Webcomponents sind als Angular-Komponenten (siehe Abschnitt 2.1.6) implementiert. Um diese verwenden zu können, muss nur das Modul der KEBA-Webcomponents importiert werden. Mittels Angular-Content-Projection und diversen Directives ist es möglich, die Komponenten den Ansprüchen entsprechend anzupassen. Mit Content-Projection können die Stellen, an denen diverse Texte eingefügt werden, angepasst werden. Dies geschieht über die Slot-Directive. Mit anderen Directives können auch zum Beispiel die Größe oder das Verhalten der Komponenten gesteuert werden. Damit wird es Entwickelnden ermöglicht, zeiteffizient Benutzeroberflächen zu bauen. Weiters ist damit auch ein einheitliches Design über die ganze Anwendung hinweg gewährleistet. [67]

2.4.4 µPlot

µPlot ist eine 2D-Chart-Bibliothek, die sich auf Leistung und Arbeitsspeichereffizienz konzentriert. [68]

In der Abbildung 22 wird ein einfacher µPlot-Chart mit einer Linie angezeigt. Die Legende zeigt für die Position des Mauszeigers den Punkt auf der x-Achse und den dazugehörigen y-Wert an. Der angezeigte Datenpunkt wird auf der Linie durch einen Punkt angezeigt. Wenn nicht weiter angegeben, legt µPlot eine Linie durch alle Datenpunkte und verbindet diese. Optional können Sprünge in der Linie eingestellt werden. Dadurch werden zwei Datenpunkte nicht durch eine Linie miteinander verbunden.

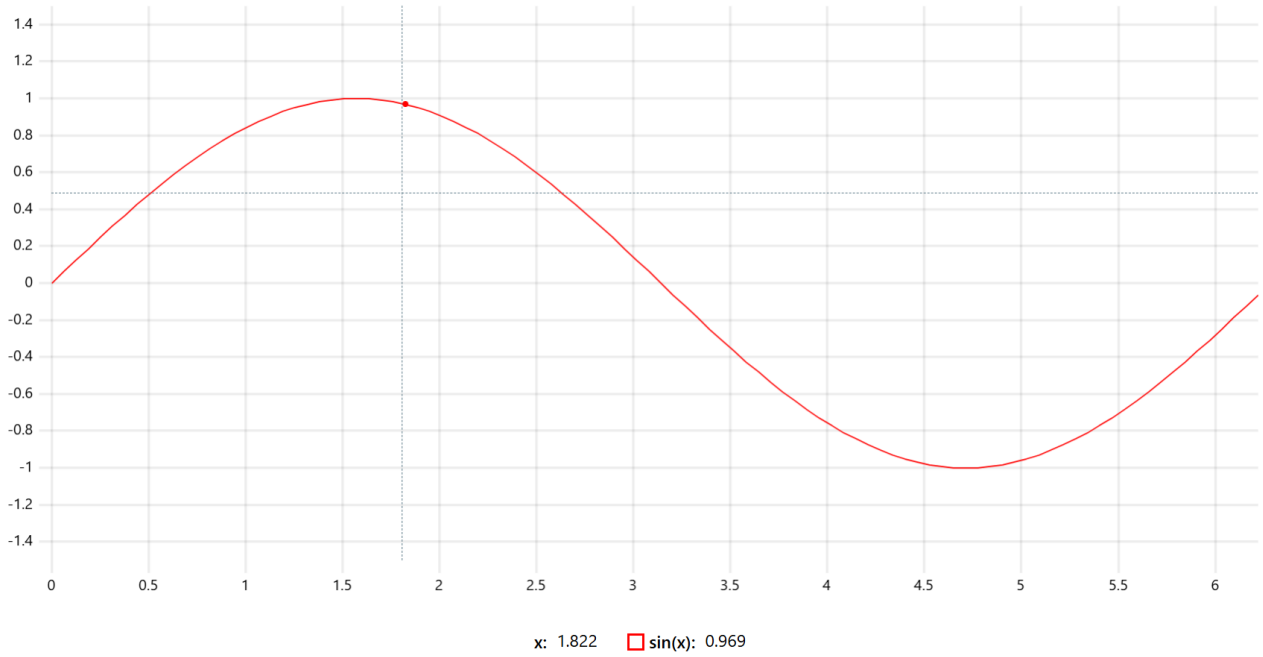

Abbildung 22: µPlot-Chart mit einer Linie [69]

Listing 6: Erstellen eines µPlot-Charts

```
let u = new uPlot(opts, data, document.body);
```

Listing 6 zeigt das Erstellen eines Charts mit µPlot. An µPlot werden drei Parameter übergeben. 'opts' sind die Chart-Konfigurationen, 'data' ist ein Array mit Daten und document.body ist das HTML-Element, in welches das Chart eingebettet wird. [70]

Listing 7: Datenformat von µPlot

```
let data = [
  [1546300800, 1546387200],    // x-values (timestamps)
  [        35,         71],    // y-values (series 1)
  [        90,         15],    // y-values (series 2)
];
```

Listing 7 zeigt das Datenformat von µPlot für zwei angezeigte Linien im Chart. Im Datenarray gibt es drei Unterarrays: Das erste Array beinhaltet die Daten für die x-Achse. Im Listing 7 werden die Werte für die x-Achse in Millisekunden seit 1970 angegeben, weil µPlot die Werte der x-Achse standardmäßig als Unix-Timestamp interpretiert und für die Anzeige in menschenlesbare Zeiten umwandelt. Nach dem Array für die Werte der x-Achse können mehrere Arrays angegeben werden. Jedes dieser Arrays speichert jeweils die zu der x-Achse passenden y-Werte für eine Linie. [70]

Listing 8: Auszug der Chart-Optionen

```
let opts = {
  title: "My Chart",
  width: 800,
  height: 600,
  series: [
    {},
    {
      [...]
      label: "RAM",
      value: (self, rawValue) => rawValue == undefined ? "" : rawValue.toFixed(2),
      stroke: "red",
      width: 1,
      dash: [10, 5],
    }
  ],
};
```

Im Listing 8 wird ein Auszug der Einstellungen für ein µPlot-Chart angegeben. In den Optionen wird hier festgelegt, dass das Chart den Titel "My Chartänzeigt sowie 800 Pixel breit und 600 Pixel hoch ist. Die Höhe des Charts berücksichtigt den benötigten Platz für den Titel sowie für eine Legende nicht, weil diese Größen mit CSS angepasst werden können. Nach den Grundeinstellungen für das gesamte Chart können Einstellungen für einzelne Linien angegeben werden. Jede Linie ist eine sogenannte Serie. Im Listing 8 wird einer Serie der Name "RAM"gegeben. Dieser wird in der Legende angezeigt. Über "value"können Daten für die Anzeige bearbeitet werden. Im Listing wird, wenn für einen Wert auf der x-Achse kein Wert für die y-Achse vorhanden ist, der Wert genommen. Das sorgt dafür, dass in der Legende kein ündefinedängezeigt wird, sondern der Wert leer ist. Gibt es hingegen einen Wert, wird dieser auf zwei Nachkommastellen gerundet. Weiters wird für diese Serie die Farbe Rot, eine Linienstärke von eins und ein Linienstil ausgewählt. µPlot nutzt keine fixen Linienstile, stattdessen können diese individuell eingestellt werden. Im Listing 8 wird mit "dash: [10, 5]ëin strichlierter Stil ausgewählt, bei dem ein Strich zehn Pixel lang ist und zwischen den Strichen jeweils fünf Pixel Abstand sind. [70]

2.4.5 OpenAPI-Generator-CLI

Das OpenAPI-Generator-CLI ist ein Open-Source-Tool, mit dem es möglich ist, clientseitige Anbindungen an REST-APIs, welche mit dem OpenAPI-Standard verfasst wurden, zu generieren. Es ist ebenfalls möglich, die Dokumentation und Konfiguration des OpenAPI-Standards zu generieren. [71]

Im Projekt wurde das OpenAPI-Generator-CLI zur Generierung der Anbindung an die REST-API des Datenrekorders (siehe Abschnitt 2.7.2) verwendet.

2.5 Sonstige verwendete Software

Für das Design, zur Planung und zur Kommunikation wurden auch noch diverse andere Softwareprodukte verwendet.

2.5.1 Figma

Figma (Logo siehe Abbildung 23 ist ein Werkzeug, mit dem es möglich ist, kollaborativ Designs für Softwareprodukte zu erstellen. Mit Figma ist es möglich, sogenannte Wireframes zu bauen. Ein Wireframe ist ein erster Prototyp eines Softwareprodukts. Damit werden das Layout und die Benutzerinteraktionen festgelegt. Mit einem fertigen Wireframe ist es möglich, durch alle Seiten der Anwendung zu klicken. Die dahinterliegenden Funktionen sind jedoch noch nicht implementiert. Mithilfe von Wireframes können Designs schnell iteriert werden. Weiters ist es möglich, schnell mit Kunden abzustimmen, in welche Richtung sich das Software-Produkt entwickeln soll. [72]

Abbildung 23: Figma Logo

2.5.2 Chromium

Chromium (Logo siehe Abbildung 24) ist ein Open-Source-Browser von Google, welcher eine Code-Basis für andere Browser liefert, wie etwa Google Chrome oder Microsoft Edge. Dabei basiert zum Beispiel Google Chrome zwar auf Chromium, erweitert diesen aber noch um einige weitere Funktionen und Google-Dienste. Chromium bietet alle notwendigen Funktionen, die bei einem Browser auf einer Steuerung (siehe Abschnitt 2.6.1) benötigt werden. Chromium bietet keine Unterstützung für viele zusätzliche Funktionalitäten, die andere Browser zur Verfügung stellen, die aber nicht auf einer Steuerung genutzt werden können. Daher eignet sich Chromium sehr gut für Anwendungen, die auf einer Steuerung in einem Browser ausgeführt werden, weil diese nicht genutzten zusätzlichen Funktionalitäten keine Leistung benötigen. [73]

Abbildung 24: Chromium Logo [73]

2.5.3 Clockify

Clockify (Logo siehe Abbildung 25) ist ein Zeitverwaltungswerkzeug, in dem Nutzer und Nutzerinnen die benötigte Zeit für Aufgaben eintragen und die aufgewendete Zeit für Projekte messen und analysieren können. Neben dem Zeitmanagement bietet Clockify auch die Möglichkeit, Projekte und Klienten für diese Projekte zu verwalten. Über eintragbare Stundenlöhne bietet Clockify zudem Funktionen im Bereich Budgetierung und Kostenberechnung. [74]

Abbildung 25: Clockify Logo [74]

2.5.4 Microsoft Teams

Microsoft Teams (Logo siehe Abbildung 26) ist eine Plattform von Microsoft, über welche Online-Meetings durchgeführt werden können. Die Plattform bietet eine Aufteilung in Teams (Gruppen). Innerhalb eines Teams können über Nachrichtenkanäle Informationen und Dateien ausgetauscht werden. Dabei fokussiert sich ein Kanal in einem Team jeweils auf ein bestimmtes Thema oder ein Projekt. Microsoft Teams ermöglicht die Einbindung verschiedener Anwendungen, um den Nutzern von Microsoft Teams mehr Möglichkeiten zu bieten. Zu dieser Auswahl an Anwendungen gehören unter anderem Planungsanwendungen, Unterstützung durch Künstliche Intelligenz oder Notizanwendungen. Über einen Kalender werden alle geplanten Meetings angezeigt. [76]

Abbildung 26: Microsoft Teams Logo [75]

2.5.5 Overleaf

Overleaf (Logo siehe Abbildung 27) ist ein LaTeX-Editor, mit dem wissenschaftliche Arbeiten mittels LaTeX geschrieben werden können. LaTeX ist ein Werkzeug zum Erstellen von professionellen Texten. Im Gegensatz zu anderen Anwendungen, mit denen Texte geschrieben und formatiert werden können, nutzt LaTeX Plaintext. Formatierungen wie Schriftstärke, Überschriften und eingefügte Grafiken, Code-Listings, Tabellen und weitere Elemente der Arbeit werden über Befehle innerhalb der Arbeit eingebunden. Die TeX-Engine verarbeitet dieses Text-Dokument

Abbildung 27: Overleaf Logo [77]

und erstellt ein formatiertes PDF. Dabei übernimmt die TEX-Engine die genaue Platzierung von Grafiken und weitere Ausrichtungen. Über Befehle kann angegeben werden, wie nahe an der Platzierung im Text das Element eingebunden werden soll und wie viel Spielraum die TEX-Engine bei der Strukturierung der Arbeit hat. Overleaf kann lokal oder webbasiert verwendet werden. Für eine Arbeit kann ein Projekt erstellt werden. Alle Mitglieder des Projekts können gleichzeitig auf die Arbeit zugreifen und gemeinsam schreiben. [78]

2.6 Verwendete Hardware

Um das Software-Oszilloskop testen zu können, hat KEBA uns eine Testumgebung bereitgestellt. Diese Testumgebung besteht aus einer Steuerung (siehe Abschnitt 2.6.1) und einem Display (siehe Abschnitt 2.6.2). Diese Steuerung befindet sich im selben Netzwerk wie unsere Arbeitsgeräte.

2.6.1 KeControl C5

Die KeControl C5 (siehe Abbildung 28) ist eine Industriesteuerung der Firma KEBA. Diese Steuerung kann bei vielen verschiedenen Industriemaschinen eingesetzt werden. Die Steuerung zeichnet mittels Sensoren verschiedene Daten auf. Diese Daten werden danach über eine Websocket-API (siehe Abschnitt 2.7.1) an unsere Anwendung weitergeleitet. Die KeControl C5 ist ein PLC (siehe Abschnitt 2.1.3).

Abbildung 28: KeControl C5 [79]

Da diese Steuerung nur begrenzte Rechenkraft hat, war es bei unserer Anwendung besonders wichtig, darauf zu achten, dass diese auch auf dieser Steuerung performant ist.

2.6.2 KeTop AP500

Das KeTop AP500 (siehe Abbildung 29) ist ein Display, welches von der Firma KEBA hergestellt wird. Das KeTop AP500 ist auch für Industrieumgebungen zugelassen und wird vor allem in solchen Umgebungen benutzt. Dieses Display wird dazu genutzt, um mit einer Industriemaschine, welche von einer Industriesteuerung von KEBA gesteuert wird, zu bedienen.

Abbildung 29: KeTop AP500 [80]

Dieses Display bietet eine Touchfunktion. Darum haben wir sichergestellt, dass unsere Anwendung auch mit reinen Touchgesten bedienbar ist.

2.7 Verwendete Schnittstellen

Bei unserer Anwendung wurden verschiedene Schnittstellen des PLCs verwendet, um mit diesem zu kommunizieren.

2.7.1 Datenrekorder Websocket-API

Der PLC (siehe Abschnitt 2.1.3) bietet eine WebSocket-Anbindung (siehe Abschnitt 2.2.8) für die Konfiguration der Datenrekorder (siehe Abschnitt 2.1.2). Mit dieser Anbindung können neue Datenrekorder angelegt werden. Diese können auch wieder gelöscht oder bearbeitet werden. Es ist ebenfalls möglich, Auslöser zu diesen Datenrekordern hinzuzufügen. Das Entfernen und Hinzufügen von Variablen zu einem Datenrekorder wird auch von dieser Schnittstelle bereitgestellt. Die Daten, die von den verschiedenen Variablen aufgezeichnet werden, können über diese API empfangen werden.

Diese Anbindung entspricht dem AsyncAPI-Standard der Version 3.0. Dieser Standard hilft dabei, asynchrone APIs zu definieren und ist protokollunabhängig. Es werden einige Tools für diesen Standard angeboten, welche das Schreiben von Anbindungen an die API und das Dokumentieren der API erleichtern. [81]

2.7.2 Datenrekorder REST-API

Der PLC (siehe Abschnitt 2.1.3) stellt eine REST-API (siehe Abschnitt 2.2.7) zur Konfiguration der Datenrekorder (siehe Abschnitt 2.1.2) bereit. Über diese Schnittstelle können neue Datenrekorder erstellt, bestehende gelöscht oder angepasst werden. Zudem lassen sich Auslöser für einzelne Datenrekorder konfigurieren. Die API ermöglicht auch das Hinzufügen und Entfernen von Variablen zu den Datenrekordern. Darüber hinaus können die aufgezeichneten Variablendaten der Datenrekorder über die Schnittstelle abgerufen werden.

Diese REST-API hält sich an den OpenAPI-Standard und wurde mithilfe des OpenAPI-Plugins (siehe Abschnitt 2.4.5) an unsere Anwendung angebunden.

3 Implementierung

3.1 Monorepo

Ein Monorepo ist ein Repository (siehe Abschnitt 2.3.1) welches aus mehreren kleineren Anwendungen besteht. Diese Anwendungen müssen definierte Schnittstellen haben. Es ist ebenfalls wichtig, dass eine klare Trennung zwischen den verschiedenen Anwendungen besteht. Ein weiterer wichtiger Punkt ist, dass alle Teams in diesem Monorepo arbeiten. Dadurch wird die Zusammenarbeit zwischen unterschiedlichen Teams gefördert. Bei einem Monorepo werden anwendungsübergreifende Strukturen aufgebaut, die beim Testen, Versionieren und Veröffentlichen helfen. [82]

Ein Monorepo ermöglicht es, einfach neue Anwendungen zu erstellen, da die gesamte umliegende Struktur bereits aufgebaut ist. Durch gemeinsame Tests ist es immer möglich zu sicherzustellen, dass keine Änderungen eingebunden werden, welche in anderen Anwendungen zu Problemen führen. Es wird auch sichergestellt, dass alle Projekte auf demselben Stand sind. Da die Struktur und die Arbeitsweise immer gleich sind, können Entwickler und Entwicklerinnen einfacher zwischen verschiedenen Teams wechseln. [82]

3.1.1 Arbeitsweise im Monorepo

Die Arbeitsweise in einem Monorepo ist meist über alle Anwendungen hinweg einheitlich geregelt, geregelt. Es gibt einige Vorteile, welche ein Monorepo im Gegensatz zu anderen Strukturen im Bezug zur Arbeitsweise hat. Es ist möglich, das Resultat einzelner Anwendungen, sofern diese unverändert sind, lokal zu speichern. Dadurch muss dieselbe Anwendung nicht mehrmals gebaut werden. Es ist ebenfalls möglich, einige Aufgaben parallel auszuführen, was dazu führt, dass die Zeit, die zum Testen benötigt wird, reduziert wird. Wenn eine Anwendung gebaut wird, wird sie auf einen lokalen Server hochgeladen, um für alle zur Verfügung zu stehen. Das bedeutet, dass ein und dasselbe Programm nur einmal gebaut werden muss und zwischen den verschiedenen Computern der Entwickler und der Entwicklerinnen ausgetauscht werden kann. [82]

Ein weiterer Vorteil von Monorepos ist die Möglichkeit, Aufgaben auf mehrere Geräte aufzuteilen. Dadurch wird die Last dieser Aufgaben aufgeteilt und die Dauer dieser Aufgaben reduziert. Durch

die klare Struktur und die festgelegten Schnittstellen kann in einem Monorepo schnell festgestellt werden, welche Projekte von einer Veränderung betroffen sind. Viele Monorepo-Werkzeuge bieten außerdem die Möglichkeit, einen Graphen des ganzen Monorepos zu zeichnen, welcher dabei hilft, den Überblick über das Monorepo zu behalten. Dieser Graph hilft außerdem dabei, neue Entwickler und Entwicklerinnen in die Struktur des Monorepos einzuführen. [82]

3.1.2 Nachteile eines Monorepos

Ein Monorepo hat auch einige Nachteile. Es wird der benötigte Speicherplatz erhöht, da jeder Entwickler und jede Entwicklerin immer die gesamte Codebasis herunterladen muss. Da alles in einem Git-Repository (siehe Abschnitt 2.3.1) gespeichert wird, ist dies der einzige Weg, um zum Monorepo beizutragen. Außerdem wird die Build-Pipeline komplizierter und das Testen kann länger dauern. Das ist der Fall, da immer jeder Teil des Monorepos getestet wird, auch die Teile, welche nicht verändert wurden und auf die das veränderte Programm keine Auswirkung hat. Wenn die Last dieser Tests nicht aufgeteilt wird, sondern zentral von einem Server übernommen wird, kann es dazu kommen, dass einzelne Entwickler und Entwicklerinnen auf andere warten müssen, da die Tests so lange dauern oder nicht paralell ausgeführt werden können. Ein weiterer Nachteil ist, dass jeder Entwickler und jede Entwicklerin Zugriff auf die gesamte Codebasis hat. Dies kann zu Sicherheitslücken führen. Es ist dadurch auch schwieriger, Teile des Projektes als Open-Source zur Verfügung zu stellen. [83]

3.2 Architektur

Zur besseren Strukturierung ist das Software-Oszilloskop in mehrere Komponenten aufgeteilt. Die KEBA-Steuerung (siehe Abschnitt 2.6.1) zeichnet Daten von zahlreichen Sensoren auf. Diese Daten werden durch eine REST-API sowie durch eine Websocket-API von der Steuerung zur Verfügung gestellt (siehe Abbildung 30).

Die Steuerung bietet zudem die Möglichkeiten zur Konfiguration von Datenrekordern. Damit kann eingestellt werden, welche Daten aufgezeichnet werden sollen und wie sich die Datenaufzeichnung verhalten soll. Diese Konfigurationen sind ebenfalls über eine REST-API sowie über eine Websocket-API vorzunehmen (siehe Abbildung 30).

Die Middleware (siehe Abschnitt 3.5) verwendet die Datenrekorder-REST-API sowie die Datenrekorder-Websocket-API (siehe Abschnitt 3.5.3 und 3.5.4). Die Middleware ist im DevAdmin (siehe Abschnitt 2.1.1) in Form von zwei TypeScript-Bibliotheken (siehe Abschnitt 2.2.4) eingebunden (siehe Abbildung 30). Die Datenrekorder-REST-API wird zur Konfiguration von Datenrekordern und Variablen verwendet. Die Datenrekorder-Websocket-API wird für die Abfrage von Sensordaten verwendet.

Das Software-Oszilloskop ist als Remote in den DevAdmin integriert (siehe Abschnitt 3.2.2). Über die Benutzeroberfläche des Software-Oszilloskops ist es möglich, mithilfe der beiden TypeScript-Bibliotheken der Middleware, Konfigurationen an Variablen und Datenrekordern vorzunehmen (siehe Abbildung 30). Ebenfalls ist es die Aufgabe des Software-Oszilloskops, die Chart-Komponente zu konfigurieren.

Die Chart-Komponente ist als Angular-Komponente (siehe Abschnitt 2.1.6) implementiert und in das Software-Oszilloskop integriert. Die Chart-Komponente ist so allgemein implementiert, dass sie auch in anderen Teilen des DevAdmin (siehe Abschnitt 2.1.1) verwendet werden kann. Die Chart-Komponente ist dafür zuständig, die ihr gegebenen Daten anzuzeigen. Durch einen Angular-Service (siehe Abschnitt 2.1.7) ist es möglich, Konfigurationen am Chart vorzunehmen.

3.2.1 Datenfluss

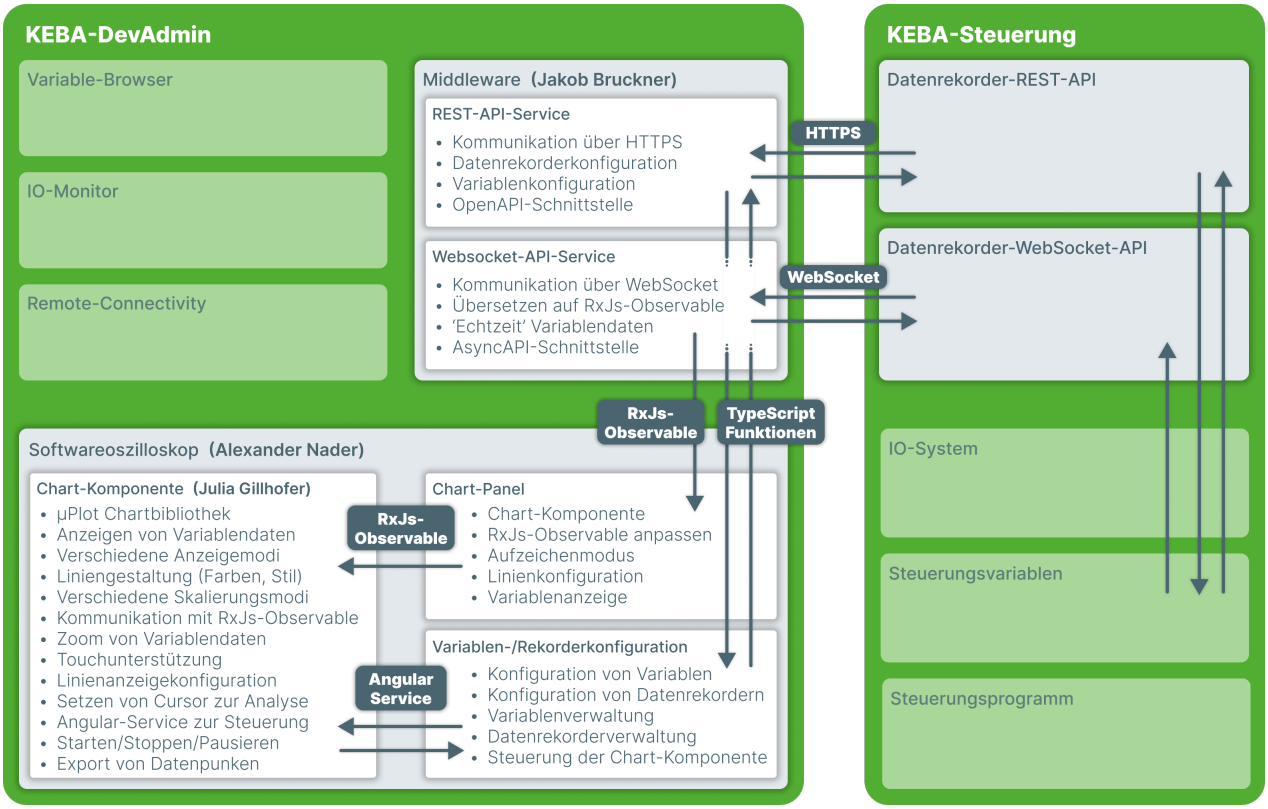

Abbildung 30: Architektur und Datenfluss des Softwareoszilloskops

Die Kommunikation zwischen der KEBA-Steuerung (siehe Abschnitt 2.6.1) und der Middleware (siehe Abschnitt 3.5) erfolgt zum einen durch HTTPS und zum anderen durch WebSockets (siehe Abschnitt 2.2.8). Die HTTPS-Schnittstelle wird zur Konfiguration und Verwaltung von Variablen und Datenrekordern verwendet (siehe Abbildung 30). Durch REST ist es möglich, diese Daten effizient auszutauschen. Diese Funktionen werden über eine TypeScript-Bibliothek weitergegeben. Bei der Übertragung der Echtzeit-Variablendaten werden WebSockets (siehe Abschnitt 2.2.8) genutzt. WebSockets ermöglichen eine effiziente Kommunikation, ohne einen HTTP-Request absetzen zu müssen. Mit WebSockets ist es der Steuerung möglich, Daten zu senden, ohne zuvor einen Aufruf erhalten zu haben. Es muss nur einmalig eine Verbindung zwischen Middleware und Steuerung aufgebaut werden. Die Echtzeit-Variablendaten werden in einer TypeScript-Bibliothek in ein RxJS-Observable (siehe Abschnitt 2.4.1) übertragen.

Die Datenübertragung zwischen Middleware und Software-Oszilloskop erfolgt zum einen durch normale TypeScript-Funktionen (siehe Abschnitt 2.2.4) und zum anderen über RxJS-Observables (siehe Abschnitt 2.4.1). Die Konfiguration und Verwaltung von Variablen und Datenrekordern ist über normale TypeScript-Funktionen realisiert. Um die Echtzeit-Variablendaten auslesen zu können, stellt die Middleware RxJS-Observables zur Verfügung. In diese Datenreihen werden immer die aktuellsten Werte der Variablen, die über den WebSocket zur Verfügung gestellt

werden, hineingegeben (siehe Abbildung 30). Wenn diese Observables abonniert werden, können immer die aktuellsten Daten in das Chart übergeben werden. Im Software-Oszilloskop werden die Daten vom Observable der Middleware in das Format der Chart-Komponente übersetzt.

Die Chart-Komponente bezieht über die Angular-Input-Direktive (siehe Abschnitt 2.2.3) ihre Eingabeparameter. Mithilfe eines RxJS-Observables (siehe Abschnitt 2.4.1) werden die angezeigten Daten im Diagramm immer aktuell gehalten. Mit einem Konfigurationsobjekt, welches dem Chart übergeben wird, kann die Gestaltung der einzelnen Linien definiert werden. Über einen von der Chart-Komponente zur Verfügung gestellten Angular-Service (siehe Abschnitt 2.1.7) können auch noch mehr Funktionen wie das Pausieren und der Datenpunkt-Export verwendet werden (siehe Abschnitt 30).

3.2.2 Integration in den DevAdmin

Das Software-Oszilloskop ist mittels Module Federation (siehe Abschnitt 2.1.8) als Remote in den DevAdmin (siehe Abschnitt 2.1.1) integriert. Das Software-Oszilloskop ist nur ein kleiner Teil des DevAdmins. Die anderen Teile, wie zum Beispiel der Variablen-Browser oder der IO-Monitor, sind ebenfalls als Remote in den DevAdmin eingebunden. Dadurch wäre es möglich, für die verschiedenen Teile des DevAdmin verschiedene Frameworks oder zumindest verschiedene Versionen von einem Framework zu verwenden. Zurzeit wird im DevAdmin jedoch nur Angular 17 verwendet.

3.2.3 Datenaktualisierung

An einigen Stellen im Projekt ist es notwendig, Daten über mehrere Komponenten (siehe Abschnitt 2.1.6) hinweg zu aktualisieren. Dies war zum Beispiel beim Wechseln von Datenrekordern oder beim Aktivieren und Deaktivieren von Variablen nötig. Es gibt mehrere Ansätze, diese Problemstellung zu lösen. In dieser Anwendung wurde ein Angular-Service (siehe Abschnitt 2.1.7) mit einem RxJS-Subject (siehe Abschnitt 2.4.1) verwendet, um die Datenaktualisierung umzusetzen. Im RxJS-Subject wird immer der aktuelle Status eines Ereignisses geteilt. Um diese Ergebnisse auszulösen, gibt es im Angular-Service eine Funktion, die einen neuen Wert in den Stream des Observables schreibt. Um von außen eine Aktualisierung zu forcieren, muss nur die 'triggerReload()'-Funktion in diesem Service aufgerufen werden (siehe Listing 9, Z. 5-7).

Listing 9: Beispielhafte Implementierung eines Angular-Reload-Service

```
1  export class ReloadService {
2    private reloadSubject = new Subject<void>();
3    public reload$ = this.reloadSubject.asObservable();
4
5    public triggerReload(): void {
6      this.reloadSubject.next();
7    }
8  }
```

Außenstehende können das aus dem Subject resultierende Observable mittels der 'subscribe()'-Funktion abonnieren (siehe Listing 10, Z. 5-13), um auf Ereignisse, wie das Aktualisieren von Daten, reagieren zu können (siehe Abschnitt 2.4.1). Damit keine Speicherlecks auftreten, wird die Subscription gespeichert. Dadurch kann die Subscription auch wieder deabonniert werden, wenn diese nicht mehr benötigt wird (siehe Listing 10, Z. 16). Um die Subscription automatisch nach dem Lebensende der Komponente zu beenden, wird die 'OnDestroy'-Lebenszyklus-Methode von Angular verwendet.

Listing 10: Beispielhafte Verwendung eines Angular-Reload-Services

```
1  export class DataComponent implements OnInit, OnDestroy {
2    private reloadService: ReloadService = inject(ReloadService);
3    private reloadSubscription!: Subscription;
4
5    ngOnInit() {
6      this.loadData();
7
8      this.reloadSubscription = this.reloadService.reload$.subscribe(
9        () => {
10         this.loadData();
11       }
12     );
13   }
14
15   ngOnDestroy() {
16     this.reloadSubscription.unsubscribe();
17   }
18   [...]
19 }
```

3.3 Evaluierung von diversen Chart-Bibliotheken

In den nachfolgenden Abschnitten wird der Evaluierungsprozess für die in der Chart-Komponente (siehe Abschnitt 1.3.2) verwendete JavaScript-Chart-Bibliothek beschrieben.

3.3.1 Auswahl der Chart-Bibliotheken

Für die Evaluierung wurden JavaScript-Chart-Bibliotheken ausgewählt, die folgende Anforderungen erfüllen:

Zuallererst muss es möglich sein, mehrere Linien in einem Chart anzuzeigen und verschiedene Farben einzustellen. Zumindest als erweiternde Funktionen werden eine Skalierungsfunktion (siehe Abschnitt 3.6.6) und ein Cursor (siehe Abschnitt 3.6.9) benötigt. Zusätzlich wurde bei der Auswahl der Chart-Bibliotheken für die Evaluierung auf weitere Punkte geachtet, darunter die mögliche Einstellung verschiedener Linienarten, der Zoom, welche Funktionalitäten sich bereits im Basis-Chart befinden und welche hinzugefügt werden müssen, wie lange der Code für das Basis-Chart ist und wie ausführlich die Dokumentation der Bibliothek ist.

Ein weiterer wichtiger Punkt in der Vorauswahl sind Lizenzen, Kosten für die Nutzung der Bibliotheken sowie das Aussehen der Charts, da das Chart in den DevAdmin (siehe Abschnitt 2.1.1) integriert wird. Alle Chart-Bibliotheken, die in die Vorauswahl aufgenommen wurden, sind kostenlos und verfügen über eine MIT-Lizenz.

Folgende Chart-Bibliotheken wurden für die Evaluierung ausgewählt (Reihung alphabetisch):

1. APEXCHARTS.JS
2. Chart.js
3. CHARTIST.JS
4. dygraphs
5. TOAST UI
6. µPlot

3.3.2 Testbedingungen

Für die Evaluierung wurde ein Angular-Projekt (siehe Abschnitt 2.2.3) erstellt. In diesem liefert ein Service (siehe Abschnitt 2.1.7) zufällige Daten für die Charts. Die Charts werden jeweils als

eine Komponente (siehe Abschnitt 2.1.6) in das Angular-Projekt eingebunden. Die Evaluierung teilt sich in drei Evaluierungsrunden.

3.3.3 Erste Evaluierungsrunde

Der Fokus der ersten Runde liegt darin, einen ersten Einblick in das Arbeiten mit diesen Bibliotheken zu erlangen und die Bibliotheken nach Performance sowie nach der Einfachheit und dem Volumen des Codes einzuordnen. Dafür werden in jedem Chart in der ersten Evaluierungsrunde zehn Linien angezeigt. Pro Linie werden die ersten 300 Datenpunkte angezeigt. Kommt danach ein neuer Datenpunkt hinzu, wird der erste Datenpunkt der Linie entfernt. Die erste Evaluierungsrunde ist bestanden, wenn zehn Linien mit je 300 Datenpunkten und sich ändernden Daten ohne Stocken angezeigt werden können. Für alle Chart-Bibliotheken aus der Vorauswahl kann die Farbe der Linien geändert und mehrere Linien gleichzeitig angezeigt werden.

APEXCHARTS.JS

Die Stärken von APEXCHARTS.JS sind eine gute Dokumentation, viele Beispielprojekte und Anwendungen für Angular sowie ein eingebautes Tool-Menü, mit dem der Zoom- und Verschiebungs-Gesten sowie ein Download des Charts als Grafik ermöglicht werden. Allerdings ist APEXCHARTS.JS in der ersten Evaluierungsrunde ausgeschieden, weil bei nur einer Linie und einem neuen Datenpunkt alle zehn Millisekunden das Chart für das Auge sichtbar stockte und somit die geforderte Performance nicht bietet. [84]

Chart.js

Chart.js hat die erste Evaluierungsrunde nicht bestanden. Bei zehn Linien und neuen Datenpunkten alle zehn Millisekunden werden 300 Datenpunkte pro Linie nur noch mit etwa 20-25 FPS aktualisiert und angezeigt. Da diese Evaluierungsrunde nicht die Obergrenze der benötigten Performance darstellt, reichen 20-25 FPS nicht aus. Die Entwicklung der Angular-Komponente für Chart.js konnte aufgrund der Dokumentation schnell abgewickelt werden. [85]

CHARTIST.JS

CHARTIST.JS hat die erste Evaluierungsrunde bestanden. Bei zehn Linien und einem neuen Datenpunkt für jede Linie alle zehn Millisekunden wird in den Chrome DevTools (siehe Abschnitt 2.3.5) eine FPS-Rate von circa 40 angezeigt. Wie bei APEXCHARTS.JS gibt es viele Beispielprojekte und der Graph skaliert automatisch mit, wenn sich der Wertebereich der Daten ändert. Zoom-Funktionalitäten sind im Basis-Chart nicht vorhanden. [86]

dygraphs

dygraphs hat die erste Evaluierungsrunde bestanden. Bei zehn Linien und einer Daten-Update-Rate von zehn Millisekunden werden in den Chrome DevTools durchgehend 60 FPS angezeigt. Im Basis-Chart sind Zoom- und Verschiebungs-Funktionalitäten eingebaut und es kann nach dem Wertebereich der angezeigten Daten skaliert werden. [87]

TOAST UI

Die Stärke von TOAST UI ist einfacher, übersichtlicher Code, allerdings benötigt TOAST UI als erste Chart-Bibliothek einen kurzen Moment, bis die Daten im Chart angezeigt werden. Da bei nur zwei Linien und neuen Datenpunkten alle 100 Millisekunden das Chart mit der Anzeige nicht hinterherkommt, scheidet TOAST UI aufgrund der Performance in der ersten Evaluierungsrunde aus. [88]

µPlot

µPlot hat die erste Evaluierungsrunde bestanden. Genauso wie auch dygraphs wurden für das µPlot-Chart bei zehn Linien und neuen Datenpunkten alle zehn Millisekunden in den Chrome DevTools durchgehend 60 FPS angezeigt. Das Basis-Chart bietet einen Zoom und skaliert den angezeigten Datenbereich automatisch auf die angezeigten Daten. [68]

3.3.4 Zweite Evaluierungsrunde

Nach der ersten Evaluierungsrunde bleiben die drei Chart-Bibliotheken CHARTIST.JS, dygraphs und µPlot, deren Performance in der zweiten Evaluierungsrunde weiter getestet wird. Dafür werden in den Charts wieder 10 Linien angezeigt. Alle zehn Millisekunden wird ein neuer Datenpunkt für die Linien hinzugefügt. Bis zu 2000 Datenpunkte werden pro Linie angezeigt, erst danach wird ein Datenpunkt gelöscht, wenn ein weiterer hinzukommt. Die Performance wird wieder mit den Chrome DevTools (siehe Abschnitt 2.3.5) gemessen.

CHARTIST.JS

Die FPS von dem CHARTIST.JS Chart gehen bei zehn Linien mit maximal 2000 Datenpunkten und neuen Datenpunkten alle zehn Millisekunden auf etwa sieben FPS zurück. Damit scheidet CHARTIST.JS in der zweiten Evaluierungsrunde aufgrund der Performance aus.

dygraphs

dygraphs hält auch bei 2000 Datenpunkten, zehn Linien und neuen Daten alle zehn Millisekunden 60 FPS. Um die Performance weiter zu testen, wurde bei den Chrome DevTools eine sechsmal langsamere CPU ausgewählt. Hier kommt das Chart auf 35 FPS. Allerdings werden bei einer

sechsmal langsameren CPU die Daten nicht mehr alle zehn Millisekunden hinzugefügt, sondern nur alle 30 Millisekunden. dygraphs hat die zweite Evaluierungsrunde bestanden.

µPlot

Genau so wie auch dygraphs bleibt µPlot in der zweiten Evaluierungsrunde bei 60 FPS. Bei sechsmal langsamer CPU bietet µPlot 45 FPS, neue Daten werden nicht mehr alle zehn Millisekunden aktualisiert, sondern nach 20 Millisekunden eingefügt. Damit hat µPlot die zweite Evaluierungsrunde bestanden.

3.3.5 Dritte Evaluierungsrunde

dygraphs und µPlot haben die zweite Evaluierungsrunde bestanden. Beide Chart-Bibliotheken erfüllen die Performance-Anforderungen sowie alle Anforderungen an Funktionalität. Um eine Entscheidung zu treffen, wird in der dritten Evaluierungsrunde das Verhalten der beiden Bibliotheken auf einer KEBA-Steuerung (siehe Abschnitt 2.6.1) evaluiert. Dafür wird das Beispiel-Projekt über einen SSH-Tunnel und Chromium (siehe Abschnitt 2.5.2) auf einer KEBA-Steuerung ausgeführt.

Der Test besteht aus zehn Linien, es werden alle zehn Millisekunden neue Datenpunkte hinzugefügt und maximal 2000 Datenpunkte pro Linie angezeigt. Danach wird für jeden hinzukommenden Datenpunkt ein Datenpunkt am Linienanfang entfernt. Insgesamt werden 3000 Datenpunkte eingefügt. Daher sollte der Test, wenn es keine zeitliche Verzögerung gibt, 30 Sekunden benötigen.

dygraphs

dygraphs benötigt mit der Hardware der Steuerung 130 Sekunden für den Test. Das liegt an der Hardware der Steuerung, die im Punkt Leistung der eines normalen Computers weit unterliegt, vor allem, weil es in der Steuerung wenig Platz für Kühlsysteme gibt und die CPU daher nicht zu viel Leistung bringen darf. Die Bedienbarkeit des dygraphs-Charts auf der Steuerung ist sehr gut. Zoom über Touch-Bedienung und Verschiebungs-Gestik über Touch sind bereits im Basis-Chart möglich. Allerdings sind diese zu sensibel und die Bedienung auf dem KEBA Display (siehe Abschnitt 2.6.2) ist wenig intuitiv.

µPlot

µPlot benötigt für denselben Test nur 55 Sekunden. Damit braucht µPlot zwar fast doppelt so lange, wie der Test bei optimalen Bedingungen benötigen sollte, ist aber mehr als doppelt so schnell wie dygraphs. Touch-Gesten werden vom Basis-Chart nicht erkannt, dafür benötigt µPlot

ein Plugin. Der Zoom mit diesem Plugin ist - verglichen von mehreren Mitarbeitern von KEBA, die mit der Bedienung von KEBA-Displays arbeiten - auf der KEBA-Steuerung intuitiver.

3.3.6 Entscheidungsfindung

Da dygraphs und µPlot beide die geforderte Performance liefern, werden für die Entscheidungsfindung neben der Performance auch die Vor- und Nachteile der beiden Chart-Bibliotheken verglichen und gewichtet.

	dygraphs	µPlot	Gewicht
Performance	-	++	5
Dokumentation	++	+	2
Touch-Bedienung	+-	+	3
Code	++	+	2

Tabelle 1: Vergleich von dygraphs und µPlot

Auf der Steuerung ist die Performance am wichtigsten, um dem Benutzer / der Benutzerin ein Chart anzeigen zu können, das weder stockt und flackert, noch mit der Anzeige der Daten nicht nachkommt. Daher hat die Performance die größte Gewichtung. Dokumentation und Code haben jeweils dieselbe Gewichtung und sind auch ähnlich einzuordnen. Dokumentation umfasst die Größe, Auffindbarkeit und Qualität der Dokumentation und wie viel Aufwand damit verbunden ist, Beispiele aus der Dokumentation im Chart umzusetzen. Code beschreibt, wie viel Aufwand damit verbunden ist, sich in den Code vom Chart einzulesen und diesen anzuwenden. Die Touch-Bedienung umfasst die Möglichkeiten, die die Charts in Bezug auf Touch-Gesten zur Verfügung stellen, sowie das Bedienverhalten.

dygraphs Stärken sind Dokumentation und Code, weil die Chart-Bibliothek in beiden Punkten besser abschneidet als µPlot. Die Touch-Bedienung muss nicht zusätzlich eingebunden werden, dafür schlägt der Zoom über Touch-Gestik zu schnell an. µPlot punktet vor allem bei der Performance, aber auch bei der Touch-Bedienung. Diese muss zwar über ein Plugin eingebunden werden, allerdings überwiegt hier der Vorteil der besseren Bedienbarkeit. Dokumentation und Code sind beide gut, können aber nicht mit der Qualität von Dokumentation und Code von dygraphs mithalten.

Aufgrund der Evaluierung wird µPlot als Chart-Bibliothek ausgewählt. dygraphs und µPlot haben beide ihre Stärken, aber die Stärken von µPlot sind höher gewichtet.

3.4 Design

Zu Beginn des Projekts wurde in der Designphase zunächst mit Figma (siehe Abschnitt 2.5.1) ein detailliertes Design in Form von Wireframes erstellt. Durch die frühe Festlegung von Design und Layout konnte sich im weiteren Verlauf der Diplomarbeit auf die Implementierung der Funktionalität konzentriert werden. Während des Implementierens mussten sich keine weiteren Gedanken über das Layout und die Gestaltung des Software-Oszilloskops gemacht werden. In der finalen Implementierung haben sich nur einige Kleinigkeiten im Vergleich zum vorherigen Design geändert.

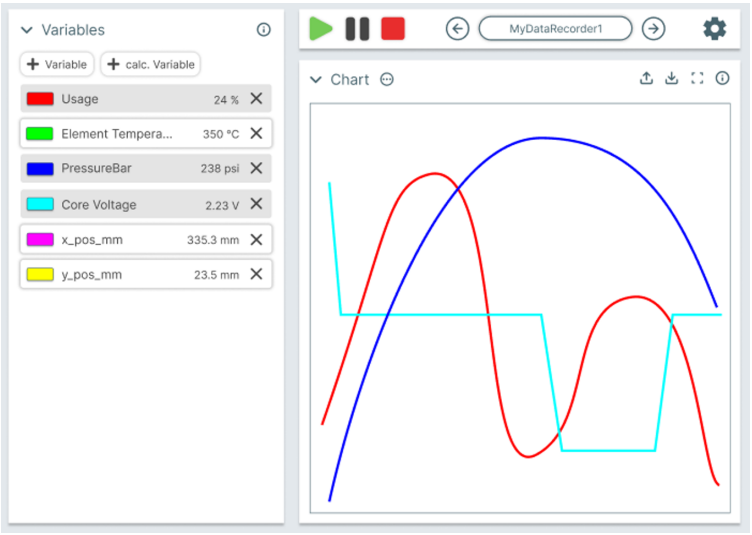

Abbildung 31: Screenshot des größten Layouts des Software-Oszilloskops

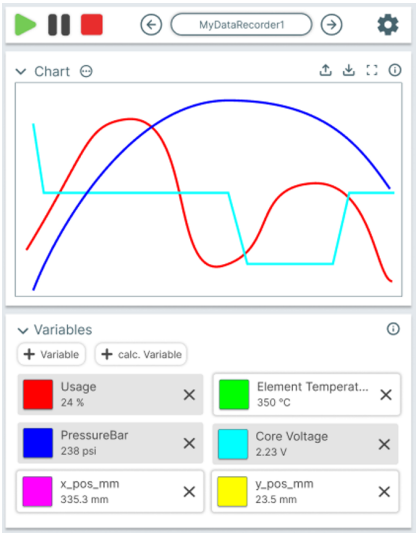

Abbildung 32: Screenshot des mittleren Layouts

3.4.1 Layout

Das Layout des Software-Oszilloskops teilt sich in drei Bereiche auf: die Variablenliste, die Interaktionsleiste und das Chart (siehe Abbildung 31). Hier finden sich alle Funktionen, die das Software-Oszilloskop zu bieten hat. Die Variablenliste ist aufgrund der Vielzahl an Variablen, die möglicherweise hinzugefügt werden, als vertikale Liste gestaltet. Diese Liste streckt sich beim vollen Layout auf der linken Seite über den gesamten Bildschirm. Um dem Chart, welches die ausgewählten Variablendaten anzeigt, so viel Platz wie möglich zu geben, ist die Interaktionsleiste als horizontales Element aufgebaut. In dieser Leiste werden das Chart gesteuert und die Datenrekorder verwaltet. Die wichtigsten und voraussichtlich am meisten genutzten Steuerelemente und Einstellungen befinden sich im linken Teil der Interaktionsleiste. Die weniger oft verwendeten Optionen, wie das Erstellen und Editieren der Datenrekorder, befinden sich im rechten Teil dieser Leiste.

Die speziellen Einstellungen für Datenrekorder und Variablen sind in ein gesondertes Side-Panel ausgelagert (siehe Abbildung 33 und 34). Beim Editieren von Datenrekordern können zum Beispiel nicht alle Parameter angepasst werden, diese sind dann ausgegraut. Beim Editieren der Variablen und Datenrekordern wird auch ein Knopf zum Löschen dieser angezeigt. Das Side-Panel öffnet sich immer von der rechten Seite des Bildschirms. Die Einstellungen sind in diesem Panel zur besseren Übersichtlichkeit in einzelne Bereiche aufgeteilt. Die einzelnen Teile sind in Akkordeon-Abschnitte eingebettet. Das bedeutet, dass einzelne logisch zusammengehörige Einstellungen gemeinsam ein- und ausgeklappt werden können. Dadurch können mehr Einstellungsmöglichkeiten auf der limitierten Bildschirmfläche untergebracht werden.

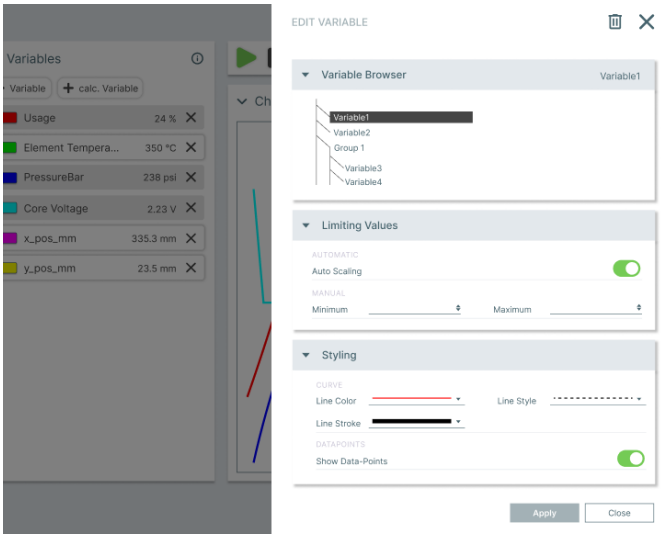

Abbildung 33: Screenshot vom großen Layout der Einstellungen

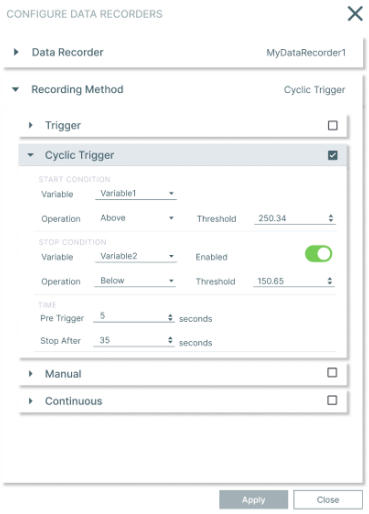

Abbildung 34: Screenshot vom kleinen Layout

3.4.2 Responsives Design

Da die Anwendung auf vielen verschiedenen Geräten läuft, ist es wichtig, dass die Anwendung bei allen möglichen Bildschirmgrößen immer noch bedienbar ist. Darum gibt es mehrere Abstufungen der Bildschirmgrößen, bei denen sich die Struktur der Elemente des Software-Oszilloskops ändert.

In der ersten Abstufung (siehe Abbildung 32) ändert sich die Position und der Aufbau der Variablenliste. Diese befindet sich nun nicht mehr auf der linken Seite der Anwendung, sondern unter dem Chart. Die Variablen werden in dieser Abstufung in zwei Spalten angezeigt. Die aktuellen Werte der Variablen werden auch nicht mehr neben den Variablennamen angezeigt. Zur besseren Sichtbarkeit werden die Werte hier unter den Namen der Variablen gezeigt. Dadurch kann der limitierte Platz bei schmaleren Bildschirmen effizienter genutzt werden. Außerdem sind die Größen der einzelnen Elemente so angepasst, dass alle Teilkomponenten noch im Blick

behalten werden können. In dieser Abstufung wird auch das Side-Panel, in welchem sich die Einstellungen zu Variablen und Datenrekordern finden, angepasst. Statt der Anzeige am rechten Rand werden die Einstellungen über die gesamte Bildschirmbreite hinweg angezeigt (siehe Abbildung 34).

In der nächsten Abstufung ändert sich die Struktur der Variablenliste und der Interaktionsleiste. Die nebeneinander platzierten Elemente der Interaktionsleiste haben ab einer bestimmten Bildschirmgröße keinen Platz mehr. Daher werden sie hier in zwei Zeilen aufgeteilt. In der oberen Zeile befinden sich die Steuerelemente des Charts sowie die Konfigurationsmöglichkeiten der Datenrekorder. Die Steuerelemente sind hierbei zentriert, die Datenrekorderkonfiguration befindet sich rechts. In der zweiten Zeile befindet sich die Datenrekorderauswahl. Auch die Struktur der Variablenliste ändert sich. Bei dieser Größe werden die einzelnen Variablen nur noch in einer Spalte angezeigt. Wenn die Variablen weiterhin in zwei Spalten angezeigt würden, könnte nicht mehr garantiert werden, dass der Variablenname und der Knopf zum Löschen sich nicht überschneiden. Die Variablenwerte werden weiterhin unter dem jeweiligen Variablennamen angezeigt.

In der letzten Abstufung wird noch einmal das Layout der Interaktionsleiste angepasst. Hierbei muss der Bildschirm so klein sein, dass die zentrierte Anzeige der Chart-Steuerelemente mit der Datenrekorderkonfiguration überschneidet. Um das zu verhindern, werden die Steuerelemente auf die linke Seite der ersten Zeile der Interaktionsleiste verschoben. Die zweite Zeile der Interaktionsleiste und auch das restliche Software-Oszilloskop ändern sich bei dieser Abstufung nicht weiter.

3.4.3 Farbpalette

Die Farbpalette, die für das Software-Oszilloskop zur Anwendung kommt, richtet sich stark nach dem bereits bestehenden Farbschema des DevAdmins (siehe Abschnitt 2.1.1). Die Farbpalette enthält die Farben, die bereits im DevAdmin verwendet wurden (siehe Abbildung 35). Im hellen Farbmodus wird als Primärfarbe ein bläuliches Grau verwendet. Diese Farbe kommt bei Text, Symbolen, Umrandungen und Knöpfen zum Einsatz. Im Gegensatz zu Schwarz wirkt diese Farbe ansprechender. Als Sekundärfarbe wird Weiß verwendet. Die Sekundärfarbe wird als Hintergrundfarbe der Interaktionselemente wie der Variablenliste, der Interaktionsleiste, des Charts und der Konfigurationen genutzt. Für den Hintergrund der Anwendung kommt ein helles, leicht bläuliches Grau zum Einsatz. Damit heben sich die weißen Interaktionselemente vom Hintergrund ab. Als Akzentfarbe wird das KEBA-Grün verwendet. Diese Farbe kommt nur bei kleineren Elementen wie dem Abspielknopf oder dem Schalterelement vor.

Das Software-Oszilloskop bietet auch die Funktionalität eines dunklen Farbmodus. Bei diesem ändern sich die Farben, um ein dunkleres Erscheinungsbild zu erreichen (siehe Abbildung 35). Die Primärfarbe ändert sich zu einem hellen Grau. Die dunkleren Elemente werden damit heller. Die Sekundär- und die Hintergrundfarbe ändern sich von einem hellen zu einem dunklen Design. Die Sekundärfarbe ist in diesem Modus ein gräuliches Navyblau, die Hintergrundfarbe ist ein dunkleres gräuliches Navyblau. Die Akzentfarbe ändert sich im dunklen Farbmodus nicht und bleibt beim KEBA-Grün.

Abbildung 35: Farbpalette des DevAdmins im hellen und dunklen Farbmodus

3.4.4 Look and Feel

Das Software-Oszilloskop passt sich stark an den bestehenden DevAdmin (siehe Abschnitt 2.1.1) an. Um das zu erreichen, wurde beim Design und der Implementierung auf die KEBA-Webcomponents (siehe Abschnitt 2.4.3) zurückgegriffen. Durch die KEBA-Webcomponents sehen alle Eingabe-, Steuer- und Strukturierungselemente im DevAdmin gleich aus. Für typische Funktionen wie Schließen oder Hinzufügen wurden dieselben Symbole verwendet, die im DevAdmin schon vorhanden sind. Auch die Aufteilung verschiedener Teile einer Seite in mehrere Paneele orientiert sich an anderen Teilen des DevAdmins. Die Konfigurationseinstellungen sind auch in anderen Teilen des DevAdmins in ein Side-Panel ausgelagert. Damit sich die einzelnen Paneele besser vom Hintergrund abheben, wurde ein Schatten verwendet. Dieser verleiht der Anwendung auch mehr Tiefe. Schatten wurden auch bei der Abgrenzung der Variablen in der Variablenliste und bei den Akkordeons in den Einstellungen verwendet. Die Anwendung reagiert sehr responsiv: wenn zum Beispiel eine Variable konfiguriert oder neu hinzugefügt wurde, schließt sich das Side-Panel automatisch und die neue oder editierte Variable wird zeitgleich in der Liste angezeigt.

Der helle und dunkle Farbmodus erlaubt es Nutzern und Nutzerinnen, die Anwendungen in einem breiten Spektrum von Umgebungen, Tageszeiten und Endgeräten zu bedienen. Durch das responsive Design passt sich das Software-Oszilloskop dynamisch an die Größe des Bildschirms, auf dem es verwendet wird, an. Damit wird sichergestellt, dass die Anwendung unabhängig vom Gerät reibungslos verwendet werden kann. Um dies zu erreichen, wurde ebenfalls darauf geachtet, dass sich das Software-Oszilloskop mit Touch-Displays wie KeTop (siehe Abschnitt 2.6.2) gut bedienen lässt.

3.4.5 Animationen

Damit die Anwendung flüssiger und dynamischer wirkt, wurden an einigen Stellen im Software-Oszilloskop Animationen verwendet. Meist sind diese Animationen direkt in die KEBA-Web-components (siehe Abschnitt 2.4.3) integriert. Animationen sind aber auch in anderen Teilen des Software-Oszilloskops vorhanden. Beim Aktivieren und Deaktivieren der Variablen wird zwischen dem normalen und dem ausgegrauten Design flüssig übergegangen. Weiters werden die Akkordeon-Menüs sowie die Side-Panels beim Ein- und Ausklappen animiert. Auch die Interaktionen mit diversen Knöpfen werden durch einen Klick-Effekt animiert. Um bei nicht so leistungsstarken Geräten die Verwendbarkeit sicherzustellen, können die Animationen deaktiviert werden. Diese Option wird vom Browser geregelt und verwaltet.

3.5 Middleware

Die Middleware ist die Schnittstelle zwischen dem Software-Oszilloskop (siehe Abschnitt 3.7) und den Datenrekordern (siehe Abschnitt 2.1.2). Die Implementierung dieser Schnittstelle ist mittels einer TypeScript-Bibliothek (siehe Abschnitt 2.2.4) realisiert.

3.5.1 Datenmodell der Middleware

Die Middleware stellt das grundlegende Datenmodell für das Software-Oszilloskop zur Verfügung. Dieses Modell ist in mehrere Klassen aufgeteilt.

Listing 11: Klassen die einen Datenrekorder darstellen

```
export type DataRecorder = Readonly<{
  name: string;
  startOptions: StartOptions;
  configOptions: ConfigOptions;
}>;

export type StartOptions = Readonly<{
  bufferSize: number;
  maxVariableCount: number;
  autostart: boolean;
  priority: number;
}>;

export type ConfigOptions = Readonly<{
  sampleRateHz: number;
  recordingSeconds?: number;
}>;
```

Die Datenrekorder-Klasse (siehe Listing 11) hat einen Namen, welcher den Datenrekorder eindeutig identifiziert, sowie Startoptionen, welche nur beim Erstellen des Datenrekorders verändert werden können. Ein weiteres Attribut der Datenrekorder-Klasse sind die Konfigurationsoptionen, welche beim Erstellen sowie beim Bearbeiten des Datenrekorders verändert werden können.

Die Startoptionen (siehe Listing 11) bestehen aus einer Puffergröße, welche für die Größe des internen Ringpuffers steht. Mit der Angabe dieses Puffers ist es möglich, die Anzahl der Datenpunkte zu erhöhen, die sofort nach dem Aufrufen von 'getSamples$' (siehe Abschnitt 3.5.2) übergeben werden. Ein Datenrekorder hat eine maximale Anzahl an Variablen, welche bei diesem Datenrekorder aufgenommen werden. Der Datenrekorder hat eine Priorität, mit 'priority' kann die Priorität im Prozessor verändert werden, um sicherzustellen, dass gewisse Datenrekorder genauere Daten liefern als andere. Es gibt außerdem die Option, den Datenrekorder nach dem Erstellen oder nach einem Neustart der Steuerung automatisch zu starten.

Die Konfigurationsoptionen (siehe Listing 11) bestehen aus der Rate, mit welcher Datenpunkte aufgenommen werden. Diese Rate wird in Hertz angegeben. Zuletzt gibt es noch den optionalen Parameter 'recordingSeconds' welcher die Aufzeichnungsdauer in Sekunden definiert. Diese

Angabe ergibt natürlich nur Sinn, wenn der Datenrekorder nur für eine begrenzte Dauer aufnehmen soll.

Listing 12: Klasse die eine Variable darstellt

```
export type Variable = Readonly<{
  variablePath: string;
  type: Type;
  color: string;
  strokeWidth: number;
  lineStyle: "Full" | "Dashed" | "Dotted" | "DotDash";
}>;
```

Die Klasse der Variablen (siehe Listing 12) besteht aus dem Pfad der Variable, mit welchem die Variable eindeutig identifiziert wird. Die Variable hat einen Typ, der angibt, in welchem Format die Daten aufgenommen werden. Dieser Typ kann ein Nummernformat wie 'uint8' oder 'sint64' sein, es sind aber auch Werte wie 'boolean' oder 'byte' möglich. Es ist ebenfalls möglich, Zeitformate wie 'date' oder 'time' anzugeben. Die restlichen Bestandteile der Variablenklasse geben an, wie die Linie der Variable im Diagramm aussieht. Es kann die Farbe, Linienstärke und Linienart angegeben werden.Dabei steht die Farbe für den CSS-Namen der Farbe und Linienstärke für die Dicke der Linie, welche ein numerischer Wert ist. Bei der Linienart gibt es einige vorgegebene Optionen.

Listing 13: Klassen die einen Trigger darstellen

```
export type Trigger = Readonly<{
    startCondition: Condition,
    recordBefore: boolean,
    recordAfter: boolean,
    stopCondition?: Condition,
    recordingSecondsAfter?: number
}>;


export type Condition = Readonly<{
    operator: Operator,
    variable: string,
    threshold: number | string
}>;

export type Operator = "none" | "pos_slope" | "neg_slope" | "pos_neg_slope" | "any_change";
```

Im Modell gibt es ebenfalls Klassen, die den Auslöser für Datenrekorder definieren (siehe Listing 13). Ein Auslöser hat eine Startkondition, eine Stopkondition und die Option, vor und nach dem Auslöser aufzunehmen.

Die Startkondition und die Stopkondition haben eigene Klassen (siehe Listing 13). Diese Klassen haben einen Schwellenwert, welcher angibt, ab welchem Wert der Auslöser aktiviert wird. Ebenfalls muss die Variable, auf die der Auslöser hört, angegeben werden. Als letztes muss noch der Operator angegeben werden, welcher bestimmt, unter welchen Bedingungen die Schwellenwertprüfung erfolgt.

Es gibt fünf verschiedene Bedingungen, unter welchen die Schwellenwertprüfung erfolgt. Die erste ist 'none' und deaktiviert den Auslöser. Die zweite ist 'pos_slope' und wird aktiviert, sobald sich der Wert erhöht. Die dritte ist 'neg_slope' und wird aktiviert, sobald sich der Wert verringert. Die vierte ist 'pos_neg_slope' und wird bei jeder numerischen Veränderung aktiviert. Die letzte ist 'any_change' und wird aktiviert, sobald eine Veränderung festgestellt wird. Im Gegensatz zu 'pos_neg_slope' wird bei 'any_change' auch die Veränderung eines nicht numerischen Wertes erkannt.

Die Stopkondition ist optional, da der Auslöser nicht unbedingt aufhören muss aufzunehmen. Es macht jedoch keinen Sinn, einen Auslöser zu definieren, welcher keine Startkondition hat.

Listing 14: Klasse die ein Sample darstellt

```
1  export type Sample = Readonly<{
2      time: number,
3      values: ReadonlyArray<number | boolean>
4  }>;
```

Die letzte Klasse des Modells ist die Klasse der Datenpunkte (siehe Listing 14). Diese Klasse ist für eine Aufnahme gedacht und hat zwei Attribute. Es wird die Zeit der Aufnahme des Datenpunkts und ein Wert für jede Variable des Datenrekorders gesendet.

3.5.2 Schnittstelle zum Software Oszilloskop

Die Middleware ist über ein TypeScript-Interface mit dem Software-Oszilloskop verbunden. Zum Testen dieses Interfaces wurden Unit-Tests (siehe Abschnitt 2.1.9) verwendet. Das Interface definiert 16 verschiedene Methoden.

Listing 15: Methoden um Datenrekorder zu verändern

```
addDataRecorder(args: {
  name: string;
  bufferSize: number;
  maxVariableCount: number;
  autostart: boolean;
  priority: number;
  sampleRateHz: number;
  recordingSeconds?: number;
}): Promise<void>;

getDataRecorder(name: string): Promise<DataRecorder | undefined>;

getAllDataRecorders(): Promise<ReadonlyArray<DataRecorder> | undefined>;

configDataRecorder(args: {
  name: string;
  sampleRateHz: number;
  priority: number;
  recordingSeconds?: number;
}): Promise<void>;

removeDataRecorder(name: string): Promise<void>;

startDataRecorder(name: string): Promise<void>;

stopDataRecorder(name: string): Promise<void>;
```

Mit der Methode 'addDataRecorder()' (siehe Listing 15) ist es möglich, einen neuen Datenrekorder hinzuzufügen. Bei dieser Funktion müssen alle Startoptionen und Konfigurationsoptionen übergeben werden. Wenn dabei die Anzahl an aufzunehmenden Sekunden mitgegeben wird, wird automatisch der Aufnahmemodus (siehe Abschnitt 3.7.5) 'Manual' gewählt. Wenn diese Option weggelassen wird und es keinen Auslöser für diesen Datenrekorder gibt, wird der Aufnahmemodus 'Continuous' gewählt. Besitzt dieser Datenrekorder einen Auslöser, wird der Aufnahmemodus 'Trigger' gewählt.

Mittels der Methode 'getDataRecorder()' (siehe Listing 15) kann ein einzelner Datenrekorder mit all seinen Attributen abgefragt werden. Dabei muss nur der Name des Datenrekorders angegeben werden, um den Datenrekorder identifizieren zu können. Wenn alle Datenrekorder benötigt werden, kann die Methode 'getAllDataRecorder()' (siehe Listing 15) verwendet werden.

Um einen Datenrekorder zu verändern, gibt es die Methode 'configDataRecorder()' (siehe Listing 15). Hier ist es nur möglich, die Konfigurationsoptionen mitzugeben und nicht die Startoptionen, da diese nach dem Erstellen des Datenrekorders nicht mehr verändert werden können.

Mit der Methode 'removeDataRecorder()' (siehe Listing 15) kann ein Datenrekorder entfernt werden. Es werden der dazugehörige Trigger und die dazugehörigen Variablen natürlich ebenfalls gelöscht. Beim Löschen muss nur der Name des Datenrekorders angegeben werden, um ihn eindeutig zu identifizieren.

Die Methoden 'startDataRecorder()' und 'stopDataRecorder()' (siehe Listing 15) sind vorhanden, um die Aufnahme der Daten stoppen und starten zu können.

Diese Methoden sind mit der REST-API des Datenrekorders (siehe Abschnitt 2.7.2) implementiert.

Listing 16: Methoden um Variablen zu verändern

```
addVariable(args: {
  dataRecorderName: string;
  variablePath: string;
  color: string;
  strokeWidth: number;
  lineStyle: string;
}): Promise<void>;

getVariable(dataRecorderName: string, variablePath: string): Promise<Variable | undefined>;

getAllVariables(dataRecorderName: string): Promise<ReadonlyArray<Variable> | undefined>;

setVariable(args: {
  dataRecorderName: string;
  variablePath: string;
  color: string;
  strokeWidth: number;
  lineStyle: string;
 }): Promise<void>;

removeVariable(dataRecorderName: string, variablePath: string): Promise<void>;
```

Mit der Methode 'addVariable()' (siehe Listing 16) kann eine neue Variable zu einem Datenrekorder hinzugefügt werden. Dabei müssen alle Attribute der Variable sowie der dazugehörige Datenrekorder angegeben werden.

Mittels der Methode 'getVariable()' (siehe Listing 16) kann eine einzige Variable abgerufen werden. Dabei muss der Datenrekorder und der Variablenpfad angegeben werden. Ist es notwendig, alle Variablen eines Datenrekorders abzurufen, kann die Funktion 'getAllVariables()' (siehe Listing 16) verwendet werden.

Um eine Variable zu verändern, kann die Methode 'setVariable()' (siehe Listing 16) verwendet werden. Bei dieser Methode müssen alle Attribute der Variable übergeben werden. Bei Variablen gibt es keinen Unterschied zwischen Startoptionen und Konfigurationsoptionen.

Zuletzt gibt es noch die Methode 'removeVariable()' (siehe Listing 16) mit welcher es möglich ist, Variablen von Datenrekordern zu entfernen. Die Variablen existieren trotzdem noch auf der Industriesteuerung (siehe Abschnitt 2.6.1) und es werden auch weiterhin Werte aufgenommen, diese werden jedoch nicht mehr vom Datenrekorder erfasst.

Diese Methoden sind ebenfalls mit der REST-API des Datenrekorders (siehe Abschnitt 2.7.2) implementiert.

Listing 17: Methoden um Trigger zu verändern

```
getTrigger(dataRecorderName: string): Promise<Trigger | undefined>;

setTrigger(args: {
  dataRecorderName: string;
  startOperator: Operator;
  startVariablePath: string;
  startThreshold: number | string;
  recordBefore: boolean;
  recordAfter: boolean;
  stopOperator?: Operator;
  stopVariablePath?: string;
  stopThreshold?: number | string;
  recordingSecondsAfter?: number;
}): Promise<void>;

removeTrigger(dataRecorderName: string): Promise<void>;
```

Mit der Methode 'setTrigger()' (siehe Listing 17) kann ein Auslöser für einen Datenrekorder konfiguriert werden. Sobald dies passiert ist, wird der Aufnahmemodus (siehe Abschnitt 3.7.5) auf 'Trigger' gesetzt. Diese Methode ist eine erstellende und konfigurierende Methode, das bedeutet, dass ein weiterer Aufruf dieser Methode keinen weiteren Trigger erstellt, sondern den vorherigen ersetzt. Es müssen alle Optionen mitgegeben werden.

Mittels der Methode 'getTrigger()' (siehe Listing 17) kann der Trigger eines Datenrekorders abgefragt werden. Wenn dieser Trigger gelöscht werden soll, kann die Methode 'removeTrigger()' (siehe Listing 17) aufgerufen werden.

Diese Methoden sind mit der REST-API des Datenrekorders (siehe Abschnitt 2.7.2) implementiert.

Listing 18: Methode um Variablendaten zu beziehen

```
getSamples$(dataRecorderName: string): Observable<Sample>;
```

Die Methode 'getSapmles$()' (siehe Listing 18) wird verwendet, um die Daten an das Software-Oszilloskop (siehe Abschnitt 1.3.1) weiterzugeben. Die Daten werden über ein RxJS-Observable (siehe Abschnitt 2.4.1) weitergegeben. Die Daten werden über die Websocket-API des Datenrekorders (siehe Abschnitt 2.7.1) übertragen und danach an alle Abonnenten weitergeleitet.

3.5.3 KEBA Datenrekorder REST-API

Die Middleware benutzt die REST-API des Datenrekorders (siehe Abschnitt 2.7.2). Dabei gibt es einige Unterschiede zur Schnittstelle vom Software-Oszilloskop. Beispielsweise gibt es keine eigenen Trigger-Methoden, sondern nur ein Attribut des Datenrekorders, welches 'trigger' heißt. Außerdem heißen alle Datenrekorder nicht 'Datenrekorder' sondern 'profile'. Es sind alle Methoden außer die Methode zur Beschaffung der Daten mit dieser REST-API implementiert, da die Beschaffung der Daten mit so wenig Verzögerung wie möglich geschehen sollte. Alle

anderen Methoden können diese minimale Verzögerung ignorieren, da diese Methoden nicht so oft aufgerufen werden.

3.5.4 KEBA Datenrekorder WebSocket-API

Es wird auch die Websocket-API des Datenrekorders (siehe Abschnitt 2.7.1) bei der Middleware verwendet. Dieses wird für die Übertragung der Daten genutzt, da bei diesem Protokoll im Gegensatz zur REST-API des Datenrekorders (siehe Abschnitt 2.7.2) nur eine Anfrage gesendet werden muss, um danach alle Daten empfangen zu können.

3.5.5 Speicherung von Attributen

Die Variablen eines Datenrekorders haben Attribute, welche nicht von den Schnittstellen des Datenrekorders vorgesehen waren. Diese Attribute sind die Farbe, Linienstärke und Linienart. Diese werden in der vorgesehenen Liste 'attributes' der Variablen gespeichert, in welcher jeder beliebige Wert gespeichert werden kann. Auf diesem Weg könnten auch noch weitere Attribute gespeichert werden.

3.6 Chart Komponente

Die Chart-Komponente beinhaltet ein µPlot-Chart (siehe Abschnitt 2.4.4) und zeigt Daten in diesem an.

3.6.1 Schnittstelle zum Software Oszilloskop

Die Chart-Komponente ist als eigene Angular-Komponente (siehe Abschnitt 2.1.6) in das Software-Oszilloskop eingebunden. Das Chart (siehe Abschnitt 2.4.4) in der Chart-Komponente kann über Eingabeparameter konfiguriert werden.

Listing 19: Einbindung der Chart-Komponente

```
<section class="chart-wrapper" slot="body">
    @if (chartOptionsService.lineConfigs.length > 0) {
      <swo-chart [lineConfigs]="chartOptionsService.lineConfigs" [newData]="sampleData$" />
    }
    @else {
      <p class="loading-message">Loading...</p>
    }
  </section>
```

Im Listing 19 ist die Einbindung der Chart-Komponente in das Chart-Panel abgebildet. Bis die Linienkonfigurationen geladen werden, wird nur der Text 'Loading...' angezeigt. Erst bei vorhandener Linienkonfiguration wird die Chart-Komponente verwendet. Zwei der drei möglichen Eingabeparameter (siehe 3.6.2) werden dem Chart übergeben. Zum einen die Linienkonfigurationen und zum anderen ein RxJS-Observable (siehe Abschnitt 2.4.1), welches die Daten liefert.

3.6.2 Eingabeparameter

Der Komponente können drei Eingabeparameter mitgegeben werden. Erforderlich sind eine Linienkonfiguration sowie ein Observable (siehe Abschnitt 2.4.1), welches die Daten für das Chart liefert. Optional kann der Komponente auch eine Konfiguration für das Chart selbst mitgegeben werden.

Listing 20: Schnittstellenbeschreibung der Linienkonfiguration

```
export type LineConfig = Readonly<{
    color: string,
    lineStyle: LineStyle,
    width: number,
    label: string,
    show: boolean,
    scale?: boolean,
    min?: number,
    max?: number
}>;
```

Im Listing 20 ist die Linienkonfiguration abgebildet. Damit wird das Design der Linien, die im Chart angezeigt werden, definiert. Eingestellt werden können Farbe, der Stil der Linie und die Dicke. Für den Linienstil stehen die Optionen 'Full', 'Dashed', 'Dotted' und 'DotDash' zur Verfügung. Neben dem Aussehen der Linie kann auch eingestellt werden, ob die Linie überhaupt angezeigt werden soll und ob die Daten der Linie auf einen bestimmten Wertebereich auf der y-Achse skaliert werden sollen (siehe Abschnitt 3.6.6).

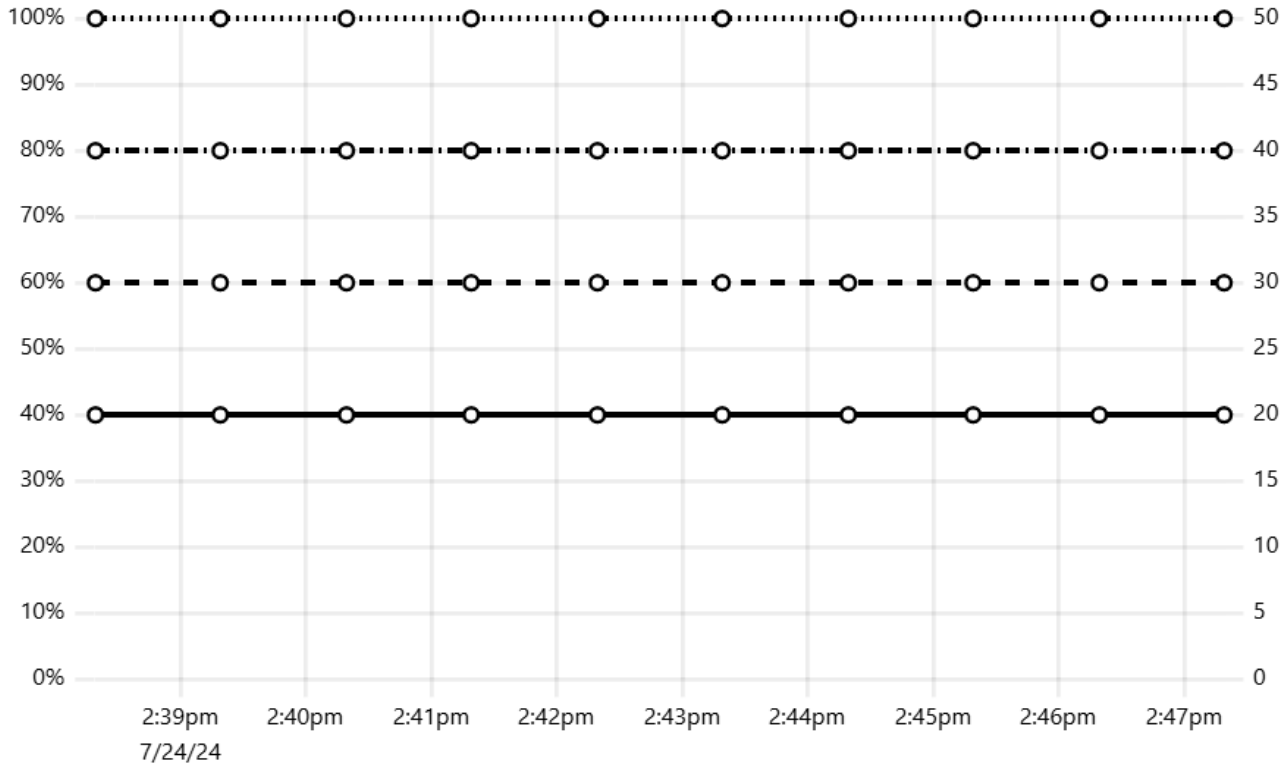

Abbildung 36: Linienstile

Abbildung 36 zeigt die verschiedenen Linienstile in einem Chart. Die unterste Linie hat den Linienstil 'Full', die Linien darüber haben 'Dashed', 'DotDash' und 'Dotted'. Die Linienstile werden jeweils als ein Array definiert.

Listing 21: Definition der Linienstile

```
1 LineStyle.Full: dash = [0,0];
2 LineStyle.Dashed: dash = [10];
3 LineStyle.Dotted: dash = [2,4];
4 LineStyle.DotDash: dash = [10, 4, 2, 4];
```

Das Listing 21 zeigt die Einstellungen für die einzelnen Linienstile. Eine durchgehende Linie wird durch '[0,0]' dargestellt. Die erste Zahl gibt hierbei (wie bei einer Linie in einem HTML5-Canvas) die Länge des gezeichneten Striches in Pixel an, die zweite Zahl den Abstand. Diese Zahl ist optional. Da beide 0 sind, wird eine durchgehende Linie gezeichnet. Die Einstellungen für

'DotDash' sind vier Zahlen, da hierbei die zwei Linienarten 'Dashed' und 'Dotted' abwechselnd verwendet werden.

Der zweite erforderliche Eingabeparameter ist ein RxJS-Observable (siehe Abschnitt 2.4.1), welches die Daten liefert. Ein Datenpunkt besteht immer aus einem x-Wert und einem Array an y-Werten. Jeder y-Wert steht für den Wert, den eine Variable, die im Chart angezeigt wird, an diesem x-Wert hat.

Der optionale Eingabeparameter für die Chart-Komponente ist die Konfiguration für das Chart.

Listing 22: Konfiguration für das Chart

```
export type ChartConfig = Readonly<{
    modus: ChartModus,
    maxDataPoints?: number,
    start?: number,
    end?: number
}>;
```

Im Listing 22 ist diese Konfiguration angegeben. Zum einen kann der Aufnahmemodus des Charts eingestellt werden:

1. **Continuous**: Es werden alle Daten angezeigt.
2. **Wandering**: Bis zu einem Limit an Datenpunkten werden alle Daten angezeigt, wird dieses Limit überschritten werden Datenpunkte am Beginn herausgenommen und nicht mehr angezeigt.
3. **Limited**: Es werden nur die Datenpunkte angezeigt, bei denen der Wert auf der x-Achse in einen bestimmten Bereich liegt.

Weiters können optionale Chart-Einstellungen in der Chartkonfiguration angegeben werden. Ob diese Parameter benötigt werden, hängt vom Anzeigemodus ab. Im 'Continuous'-Modus werden alle optionalen Einstellungsmöglichkeiten ignoriert. Im 'Wandering'-Modus muss 'maxDataPoints' gesetzt werden. Mit 'maxDataPoints' wird festgelegt, wie viele Datenpunkte höchstens im Chart angezeigt werden können. Im 'Limited'-Modus werden nur Datenpunkte angezeigt, bei denen der x-Wert zwischen 'start' und 'end' liegt. Beide Parameter müssen für diesen Anzeigemodus gegeben sein. Da nur der 'Continuous'-Modus keine optionalen Einstellungsmöglichkeiten verwendet, wird das Chart, wenn der Chart-Komponente keine Chartkonfiguration übergeben wird, automatisch im 'Continuous'-Modus gestartet.

3.6.3 Schnittstelle zum Datenexport

Um auf die Daten, die im Chart angezeigt werden, auch außerhalb des Charts Zugriff zu haben, werden zwei Arrays mit Datenpunkten über Getter-Funktionen zur Verfügung gestellt. Einerseits die Originaldaten, die dem Chart übergeben wurden. Andererseits können auch die skalierten Datenpunkte (siehe Abschnitt 3.6.6) über eine Getter-Funktion abgefragt werden.

3.6.4 Datenaktualisierung

Das Chart erhält Daten über ein RxJS-Observable (siehe Abschnitt 2.4.1). In der Chart-Komponente wird das Observable abonniert. Wenn jemand nun neue Daten zum Observable hinzufügt, wird die Chart-Komponente darüber informiert und fügt die Daten in das Chart ein (siehe Abschnitt 3.2.3).

3.6.5 Storybook

Für die Chart-Komponente wurden mehrere Storybooks (siehe Abschnitt 2.3.6) erstellt. Diese dienen zur Veranschaulichung der Funktionalitäten der Komponente und sind gleichzeitig eine Dokumentation der Chart-Komponente. Insgesamt beinhaltet das Storybook der Chart-Komponente sieben verschiedene Stories.

Die erste Story zeigt ein einfaches Chart mit nur einer Linie. Für diese Linie kann die Linienkonfiguration geändert werden. Die zweite Story (siehe Abbildung 37) zeigt ein Chart mit mehreren Linien. Da die Linienkonfiguration hier im Vordergrund steht, ändern sich die angezeigten Daten bei diesen Stories nicht. Bei allen weiteren ändern sich die angezeigten Daten dynamisch.

Eine weitere Story zeigt die dynamische Datenänderung über ein Observable. Für jede der drei Chart-Modi gibt es eine Story, um die unterschiedlichen Verhaltensweisen des Charts bei unterschiedlichen Modi darzustellen. Die letzte Story zeigt die Skalierung. Dabei kann die Skalierung unterschiedlicher Variablen dafür genutzt werden, Variablen mit stark variierenden Wertebereichen in einem Chart nebeneinander anzuzeigen.

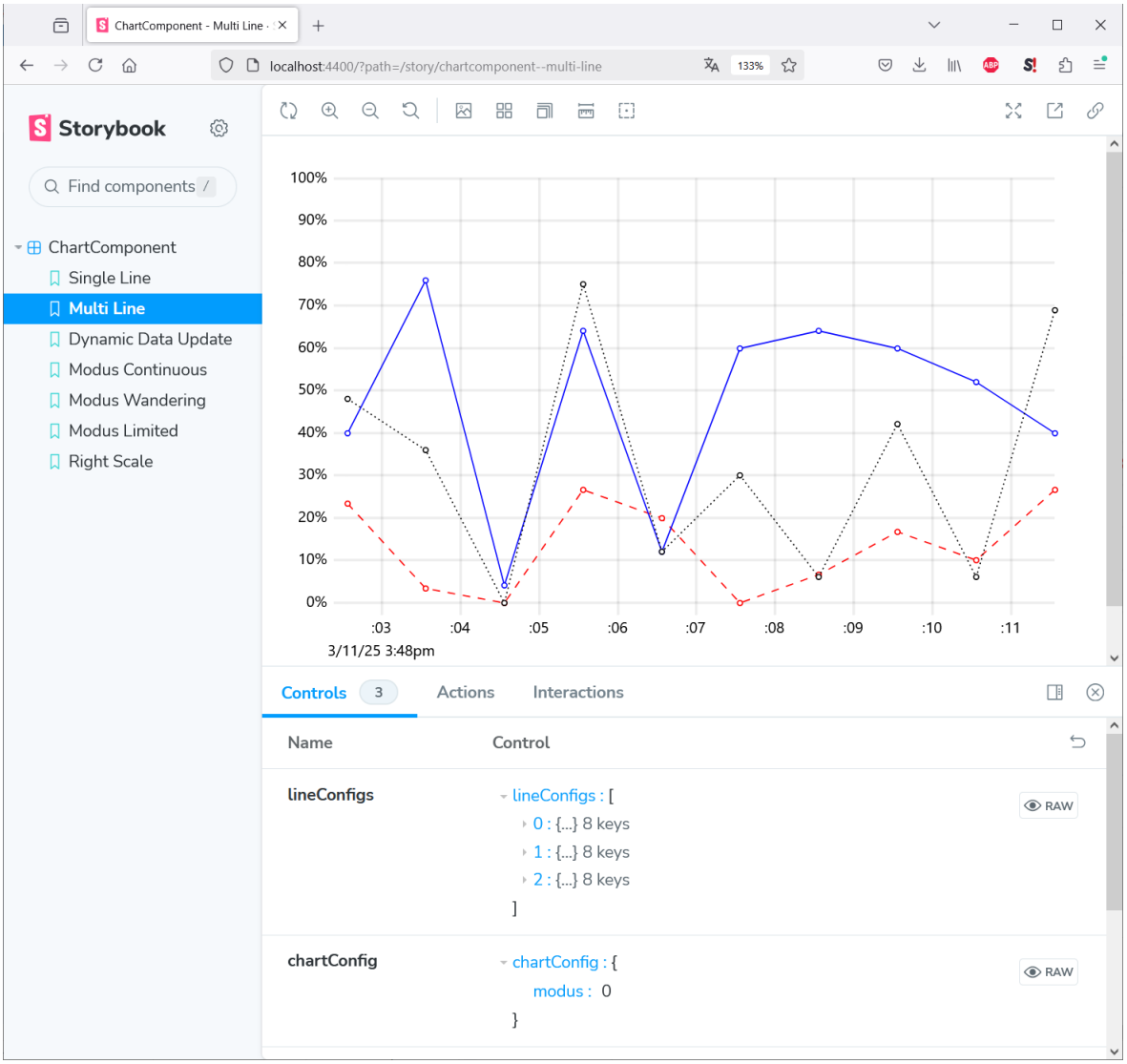

Abbildung 37: Story für die Chart-Komponente mit mehreren Linien

3.6.6 Skalierung

In der Linienkonfiguration (siehe Abschnitt 3.6.2) kann für jede Linie ein Bereich angegeben werden, auf den der tatsächliche Wertebereich dieser Linie auf der y-Achse skaliert werden soll. Werden für eine Linie keine Ober- und Untergrenze angegeben, so wird auf den jeweils höchsten und niedrigsten Datenwert skaliert, der gerade auf dieser Linie angezeigt wird.

Werden eine Ober- und Untergrenze für eine Linie angegeben, so skaliert das Chart alle Datenpunkte für diese Linie auf diesen Bereich. Der höchste Wert, der gerade auf dieser Linie angezeigt wird, skaliert zur angegebenen Obergrenze, der niedrigste zur angegebenen Untergrenze, und alle Werte zwischen dem höchsten und niedrigsten Wert auf der Linie skalieren zu dem proportionalen Wert zwischen Ober- und Untergrenze.

Wenn eine Linie skaliert wird, soll trotzdem nach außen hin der echte Wert abrufbar sein. Für den Cursor (siehe Abschnitt 3.6.9) werden Linienwerte direkt aus dem Chart abgefragt. Dafür ist es wichtig, dass das Chart die unskalierten Werte zurückgibt, weil es möglich sein soll, über den Cursor die tatsächlichen und nicht die skalierten Daten abzufragen.

Listing 23: Serie hinzufügen

```
this._chart.addSeries(
    {
      stroke: this._lineConfig[i].color,
      width: this._lineConfig[i].width,
      label: this._lineConfig[i].label,
      dash: dash,
      show: this._lineConfig[i].show,
      scale: '%',
      value: (self, rawValue) => {
        if (rawValue === null) {
          return '';
        }

        let index = (self.data[i + 1] as Array<number>).findIndex((element: number) =>
            element === rawValue);
        return this.originalData[i + 1][index];
      },
    },
    i + 1,
  );
```

Im Listing 23 ist das Hinzufügen einer Linie zum Chart dargestellt. Dafür wird eine sogenannte Serie erstellt. Für die 'value'-Eigenschaft dieser Serie wird eine Funktion gesetzt, die statt des skalierten Werts den Originalwert zurückgibt. Dadurch wird ermöglicht, dass im Chart zwar skalierte Werte angezeigt werden, der Cursor jedoch auf die Originalwerte zugreifen kann.

3.6.7 Zoom

Um sich gewisse Teile der angezeigten Daten genauer ansehen zu können, ermöglicht die Chart-Komponente verschiedene Zoom-Möglichkeiten. Zum einen kann per Maus ein Teilbereich des Charts ausgewählt werden. Dieser Teilbereich wird dann über das gesamte Chart angezeigt und kann auf verschiedene Arten ausgewählt werden.

Die Abbildungen 38, 39 und 40 zeigen die verschiedenen Zoom-Möglichkeiten.

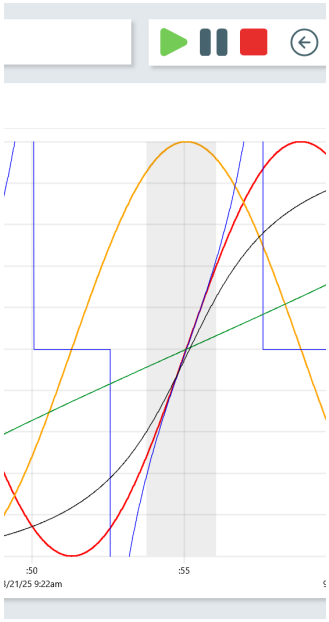

Abbildung 38: Zoom x-Achse

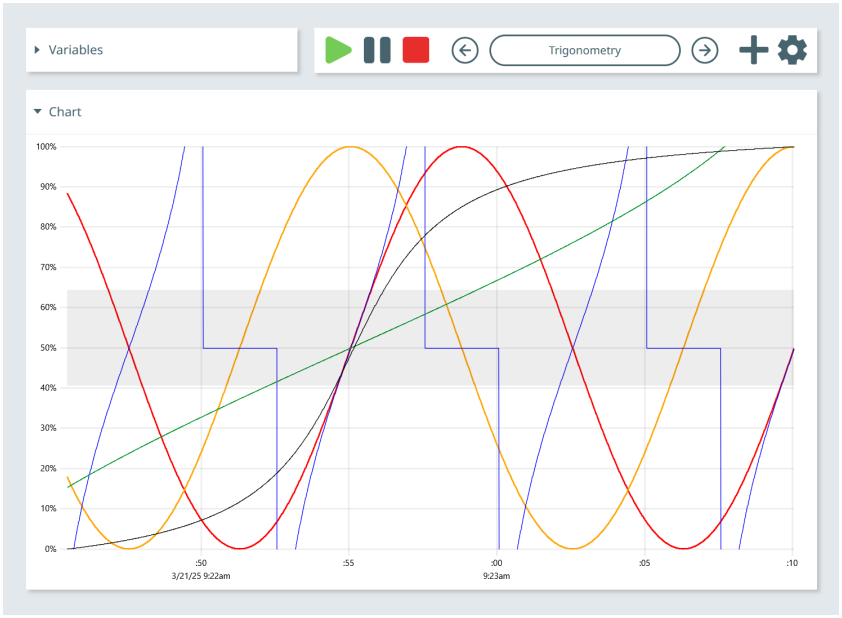

Abbildung 39: Zoom auf der y-Achse, auswählbar per Mausklick oder Touch-Gestik

Um nur einen Teil der x-Achse anzuzeigen, aber alle Werte auf der y-Achse zu behalten, kann der Benutzer im Chart mit gedrückter rechter Maustaste nach links oder rechts fahren. Wird die rechte Maustaste dann losgelassen, wird auf der x-Achse der so ausgewählte Bereich angezeigt. Um die Auswahl nur auf die x-Achse zu beschränken, darf die Maus während des Auswählens nicht zu stark in y-Richtung schwanken.

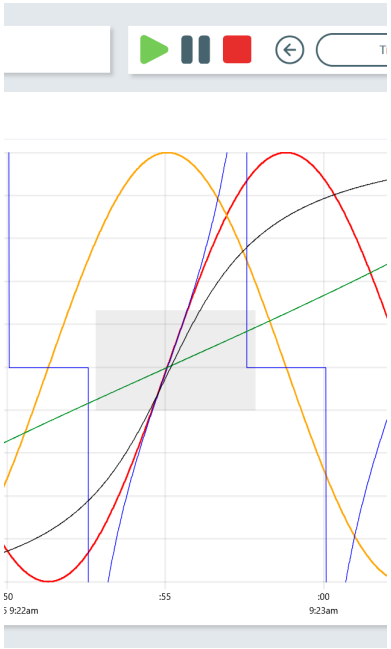

Abbildung 40: Zoom in einen Teilbereich

Mit einer ähnlichen Auswahl kann auch nur ein Teil der y-Achse angezeigt werden, während sich die x-Achse nicht ändert. Hier ist es bei der Auswahl wichtig, die Maus beim Auswählen nicht zu weit auf der x-Achse zu bewegen.

Die dritte Zoom-Möglichkeit ist die Einschränkung der Anzeige auf der x- und y-Achse. Bei den vorherigen Zoom-Möglichkeiten wird die Maus jeweils nur in eine Richtung bewegt. Die dritte Zoom-Möglichkeit wird durch eine Bewegung in beide Richtungen ausgewählt. Sobald die Auswahl in x- und y-Richtung eine Breite und Höhe von je 50 Pixel erlangt, wird ein Rechteck ausgewählt. Dieser Teilbereich wird dann wieder angezeigt.

Die Abbildung 38 zeigt den Zoom auf der x-Achse. Damit können Werte in einem Teilbereich der x-Achse genauer angesehen werden. Die Abbildung 39 zeigt den Zoom auf der y-Achse. Hiermit können Datenpunkte in einem Wertebereich genauer angesehen werden. Die Abbildung 40 zeigt den kombinierten Zoom, bei dem zugleich in die x- und y-Achse hineingezoomt wird.

3.6.8 Touch-Unterstützung

Da der DevAdmin (siehe Abschnitt 2.1.1) auch auf KEBA Displays (siehe Abschnitt 2.6.2) aufrufbar ist und viele dieser Displays über Touch-Bedienung verfügen, unterstützt die Chart-Komponente auch verschiedene Bedienungsmöglichkeiten per Touch.

Zum einen wird ein Zoom über Touch ermöglicht. Dafür wird eine Zwei-Finger-Gestensteuerung genutzt. Durch Auseinanderziehen der zwei Finger auf dem Display wird ein Teil des Charts vergrößert dargestellt, durch das Zusammenziehen wird ein größerer Bereich angezeigt.

Mit einer Ein-Finger-Geste kann der angezeigte Bereich verschoben werden. Durch einen Doppeltipp wird wieder das gesamte Chart angezeigt. Auch das Setzen des Cursors wird über Touch-Bedienung unterstützt.

3.6.9 Cursor

Der Cursor ermöglicht es, sich die Werte der einzelnen Linien für einen bestimmten Punkt auf der x-Achse anzeigen zu lassen. Mit einem einzelnen Klick oder einer Touch-Geste wird eine senkrechte Linie an einen bestimmten Punkt auf der x-Achse gesetzt. Alle y-Werte des Datenpunktes, der sich gerade am nächsten zum Cursor befindet, werden über einen Angular-Service (siehe Abschnitt 2.1.7) zur Verfügung gestellt. In der rechten oberen Ecke der Chart-Komponente wird eine Schaltfläche zum Entfernen des Cursors angezeigt. Mit einem Klick oder einer Touch-Geste auf diese Schaltfläche wird der Cursor wieder entfernt.

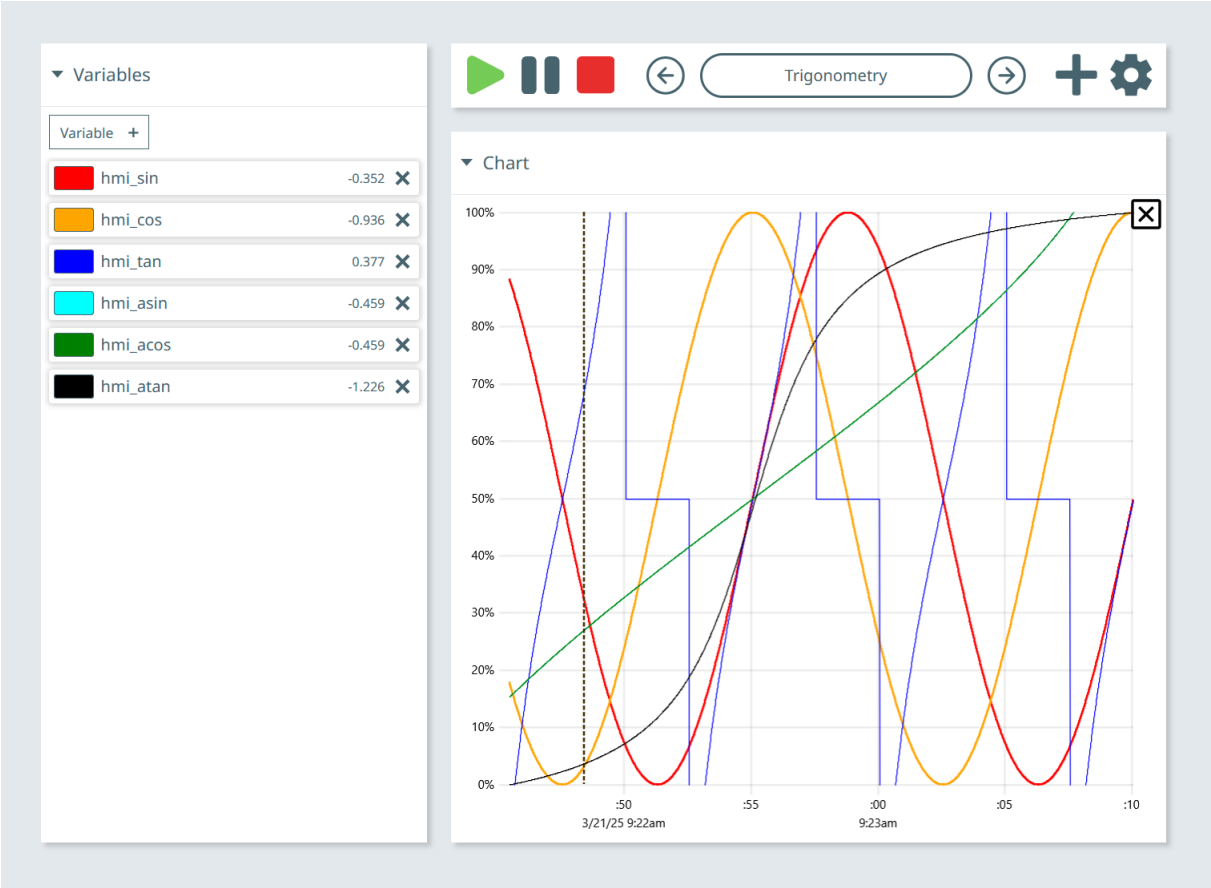

Abbildung 41: Im Chart gesetzter Cursor

Abbildung 41 zeigt den Cursor der Chart-Komponente. Dieser ist gerade im ersten Drittel der x-Achse gesetzt und liefert die Originalwerte für alle Variablen in einem Angular-Service (siehe Abschnitt 2.1.7). Dieser Service wird vom Software-Oszilloskop genutzt. Die Originalwerte der Variablen werden in der Variablenliste (siehe Abschnitt 3.7.3) auf der Seite angezeigt.

3.7 Software-Oszilloskop

Das Software-Oszilloskop führt die Middleware, die für die Datenbeschaffung verantwortlich ist, und die Chart-Komponente, welche für die Datenvisualisierung zuständig ist, zusammen. Das Software-Oszilloskop ist ein Teil des KEBA-DevAdmin (siehe Abschnitt 2.1.1).

3.7.1 Struktur des Software-Oszilloskops

Das Software-Oszilloskop ist in drei Teile unterteilt (siehe Abbildung 42).

1. **Variablen-Box**: In der Variablen-Box werden dynamisch die aktuellen Variablen und ihre Daten angezeigt. Es können neue Variablen hinzugefügt oder bestehende editiert oder entfernt werden.
2. **Chart-Box**: In der Chart-Box ist die Chart-Komponente (siehe Abschnitt 3.6) integriert. Mit dieser ist es möglich, Daten von Maschinensteuerungen (siehe Abschnitt 2.6.1) zu visualisieren.
3. **Interaktionsleiste**: Mit der Interaktionsleiste interagieren Nutzer und Nutzerinnen mit dem Software-Oszilloskop. In der Interaktionsleiste finden sich Möglichkeiten, Aufzeichnungen zu starten, zu pausieren und zu stoppen. Weiters kann auch der aktive Datenrekorder geändert werden. Hier können auch neue Datenrekorder hinzugefügt und bestehende editiert oder gelöscht werden.

Die oben genannten Elemente sind mit einem SCSS-Grid strukturiert. Dadurch werden die einzelnen Elemente geordnet, mittig auf dem Bildschirm angezeigt. Mit SCSS-Grid ist es möglich, tabellenähnliche, dynamische Layouts zu definieren. Bei Änderungen der Bildschirmgröße werden die Positionen der einzelnen Felder geändert. Damit ist es bei allen Bildschirmgrößen immer möglich, alle Funktionalitäten des Software-Oszilloskops reibungslos zu verwenden. Die Variablen- und die Chart-Box sind als Akkordeon der KEBA-Webcomponents implementiert. Das heißt, dass diese beiden Elemente der Benutzeroberfläche ein- und ausgeklappt werden können. Besonders bei kleineren Bildschirmgrößen ist dies nützlich, um alles im Blick behalten zu können.

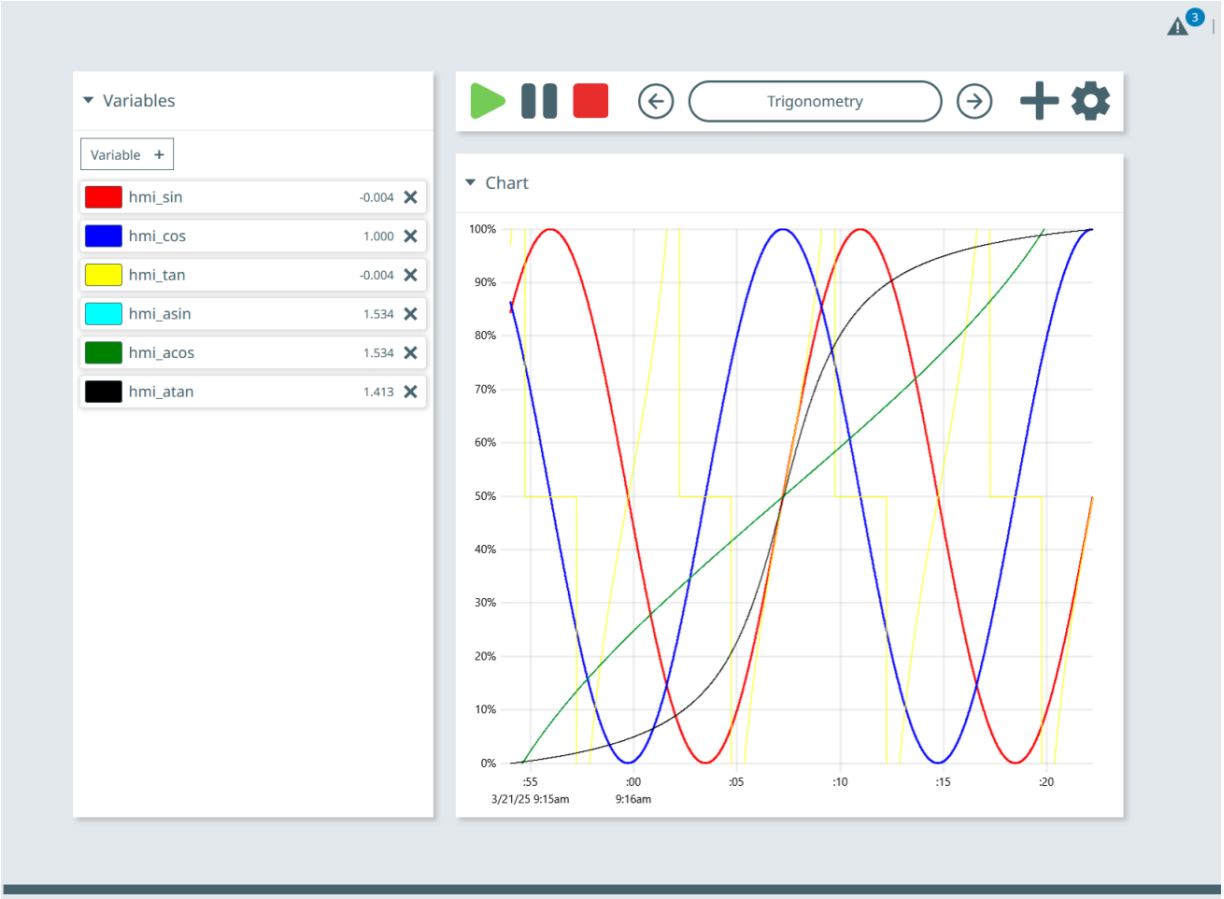

Abbildung 42: Screenshot vom Layout des Software-Oszilloskops

3.7.2 Implementierung des responsiven Designs

Das Responsive Design (siehe Abschnitt 3.4.2) wurde mittels SCSS (siehe Abschnitt 2.2.5) implementiert. SCSS bietet mit Media Queries und Container Queries eine Funktion, mit der es möglich ist, basierend auf der Bildschirmgröße oder Containergröße verschiedene Designoptionen zu verwenden. Als Breakpoints wurden Größen von 450 Pixel bis 1000 Pixel verwendet. Breakpoints sind die Viewportgrößen, ab denen sich das Design ändert. Hier wurden die im DevAdmin (siehe Abschnitt 2.1.1) als Variablen hinterlegten Größen verwendet (siehe Listing 24).

Listing 24: SCSS-Breakpoint-Variablen im DevAdmin

```
$large-screen: 1000px;
$medium-screen: 800px;
$small-screen: 600px;
$extra-small-screen: 450px;
```

Zur responsiven Gestaltung wurden größtenteils SCSS Media Queries verwendet. Durch die Verwendung der im DevAdmin (siehe Abschnitt 2.1.1) definierten Breakpoints (siehe Listing 24) verhalten sich Elemente der Benutzeroberfläche im DevAdmin immer gleich. Bei der Variablen-

Liste wurden beispielsweise die Breakpoints '$large-screen' und '$small-screen' verwendet. Mit SCSS ist es möglich, Media Queries zu verschachteln. Dadurch werden bei einer SCSS-Klasse mehrere Media Queries angegeben (siehe Listing 25). Bei der Variablen-Liste werden bei einer Bildschirmgröße von über 1000 Pixel alle Variablen untereinander angezeigt (siehe Abbildung 43). Wenn die Bildschirmgröße zwischen 1000 Pixel und 600 Pixel liegt, werden die Variablen in zwei Spalten angezeigt. Bei dieser Größe werden die aktuellen Werte der Variablen nach unten verschoben(siehe Abbildung 44). Wenn der Bildschirm kleiner als 600 Pixel ist, werden die Variablen wieder in nur einer Spalte angezeigt. Die Werte der Variablen bleiben weiterhin unter dem Namen (siehe Abbildung 45).

Listing 25: Responsives Design der Variablenliste

```
1  @import 'variables.scss';
2
3  .variables {
4    overflow-y: auto;
5    overflow-x: hidden;
6    height: min(calc(100vh - 19em), 44em);
7
8    @media (max-width: $large-screen) {
9      display: grid;
10     grid-template-columns: 50% 50%;
11     height: auto;
12   }
13
14   @media (max-width: $small-screen) {
15     display: grid;
16     grid-template-columns: 100%;
17     height: auto;
18   }
19 }
```

Dadurch ist es bei allen üblichen Bildschirmgrößen möglich, die Benutzeroberfläche intuitiv zu bedienen. Dies ist beim DevAdmin besonders wichtig, um auch mobile Geräte wie KeTop (siehe Abschnitt 2.6.2) oder Tablets zu unterstützen.

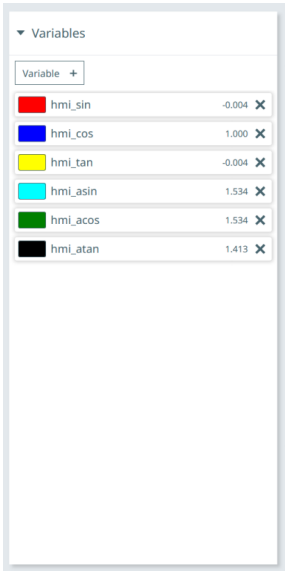

Abbildung 43: Variablen wenn Viewport größer 1000 Pixel

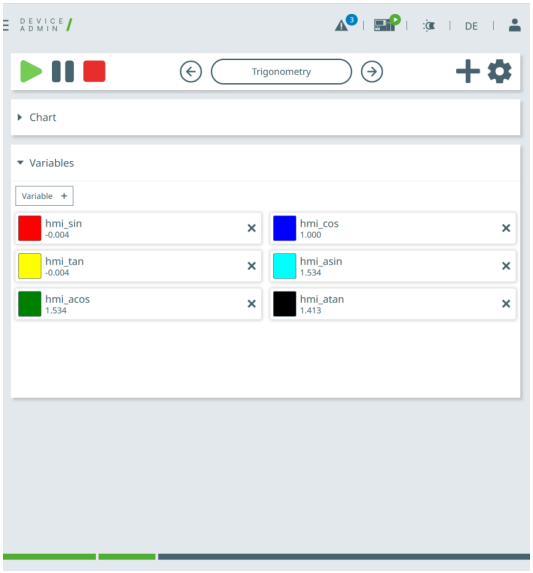

Abbildung 44: Variablen wenn Viewport größer 600 Pixel und kleiner 1000 Pixel groß ist

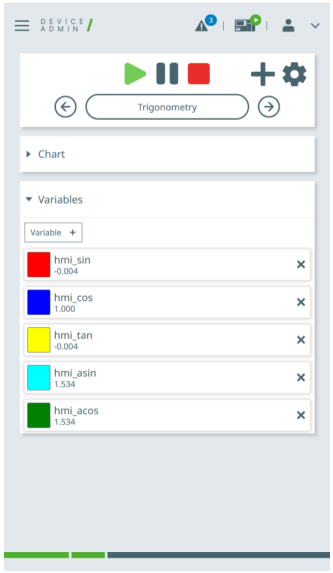

Abbildung 45: Variablen wenn Viewport kleiner 600 Pixel

3.7.3 Variablen

In der Variablen-Box können Variablen hinzugefügt und editiert werden. Je nachdem, welcher Datenrekorder eingestellt ist, wechseln die konfigurierten Variablen in der Liste. Um der Variablenliste mitzuteilen, wenn sich die Auswahl des Datenrekorders geändert hat, wird ein Angular-Reload-Service verwendet (siehe Abschnitt 3.2.3).

Beim Hinzufügen von Variablen können der Pfad der Variable, die Skalierung der Variable, eine Farbe, die Linienbreite und ein Linienstil definiert werden. Der Pfad der Variable bestimmt, welche Variable zur Visualisierung herangezogen werden soll. Variablen werden auf der Steuerung (siehe Abschnitt 2.6.1) in einer Baumstruktur klar strukturiert gespeichert. Um zu wissen, welche Variablen auf der Steuerung zur Verfügung stehen, können mit dem Variablen-Browser, einem anderen Modul des DevAdmin (siehe Abschnitt 2.1.1), alle Variablen durchsucht werden. Die Skalierungsoptionen sind in einem abgesonderten Bereich der Variablen-Einstellungen. Hier kann eingestellt werden, wie sich die Variable im Chart verhält. Mit einer Ober- und Untergrenze werden die Daten auf den so definierten Bereich skaliert. Wenn diese Eingabe leer bleibt, wird die Variable automatisch skaliert. Das bedeutet, dass der höchste Wert der Variable bei 100% und der niedrigste Wert bei 0% angezeigt werden, diese Limits ändern sich dynamisch (siehe Abschnitt 3.6.6). Zur besseren Erkennbarkeit und zur besseren Unterscheidung können Variablen-Linien durch mehrere Möglichkeiten gestaltet werden. Es ist möglich, eine aus elf vordefinierten Farben auszuwählen, mögliche Eingaben sind: 'White', 'Black', 'Red', 'Green', 'Blue', 'Cyan',

'Magenta', 'Yellow', 'Orange', 'Purple' und 'Lime Green'. Ebenfalls kann der Linienstil eingestellt werden, die möglichen Eingaben sind hier 'Full', 'Dashed', 'Dotted' und 'Dot-Dash'. Weiters kann auch die Linienbreite numerisch konfiguriert werden. Die Einstellungsmöglichkeiten der Farbe und des Linienstils sind über ein Drop-Down-Menü realisiert (siehe Abbildung 46). Hierbei wurden die KEBA-Webcomponents (siehe Abschnitt 2.4.3) verwendet. Wenn eine neue Variable hinzugefügt wird, wird sie automatisch zum aktiven Datenrekorder angefügt.

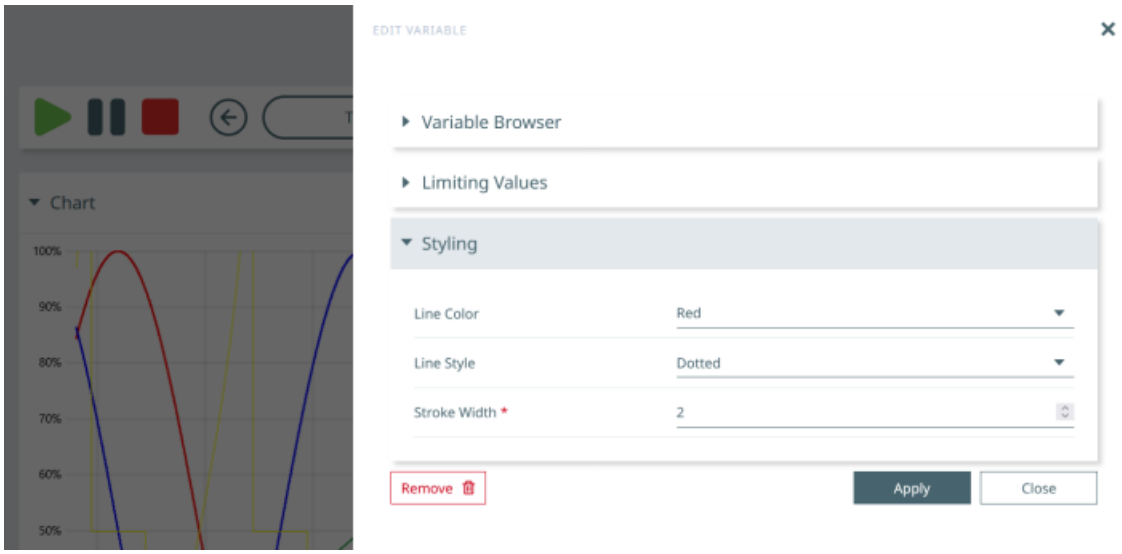

Abbildung 46: Screenshot der Design-Konfiguration der Variablen

Beim Editieren kann ebenfalls die Skalierung der Variable, die Farbe, die Linienbreite und der Linienstil angepasst werden. Der Variablenpfad kann nur beim erstmaligen Erstellen definiert werden. Um eine Variable zu editieren, wird auf die farbige Fläche neben dem Variablennamen geklickt (siehe Abbildung 47). Hierbei öffnet sich dasselbe Konfigurationsfenster wie beim Hinzufügen. Nur ist es in diesem Fall im Editierungsmodus. Das bedeutet, dass der Variablenname ausgegraut ist und auch eine Option zum Löschen der Variable angezeigt wird.

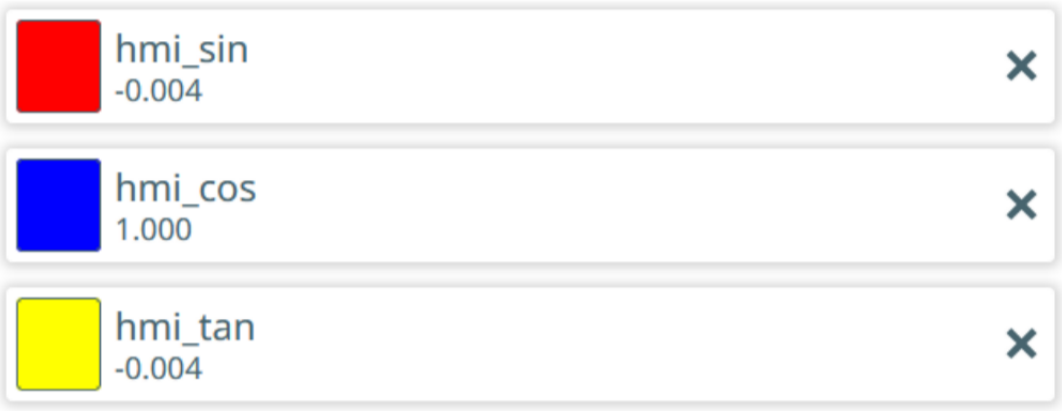

Abbildung 47: Screenshot von drei Variablen

Das Formular, welches zur Implementierung der Einstellungsmöglichkeiten der Variablen verwendet wird, ist mit Reactive-Forms (siehe Abschnitt 2.4.2) und den KEBA-Webcomponents (siehe Abschnitt 2.4.3) implementiert. Da die einzelnen Einstellungen zur Übersichtlichkeit und besseren Wartbarkeit auf mehrere Angular-Komponenten (siehe Abschnitt 2.1.6) aufgeteilt sind, wird ein Angular-Service (siehe Abschnitt 2.1.7) zur Datenhaltung der Form-Group-Elemente verwendet. Um die Einstellungen in der Benutzeroberfläche besser zu strukturieren, wurden die Pfadeinstellungen, die Skalierungseinstellungen und die Gestaltungseinstellungen jeweils in ein Akkordeon gegeben (siehe Abbildung 46). Diese Akkordeons befinden sich in einer Akkordeon-Group (siehe Listing 26). Die KEBA-Webcomponents ermöglichen es so, dass jeweils nur eine Einstellungsgruppe angezeigt wird. Wenn ein anderes Akkordeon der Akkordeon-Group geöffnet wird, wird das zuvor geöffnete Akkordeon automatisch geschlossen. Es ist bei einer Akkordeon-Group auch möglich, ein standardmäßig geöffnetes Akkordeon zu definieren. Im Fall der Variablen-Einstellungen ist der Pfad der Variable automatisch geöffnet.

Listing 26: Auszug der Verwendung der KEBA-Akkordeons der Variablenkonfiguration

```
<keba-accordion-group [openOnInit]="0">
    <keba-accordion
      [accordionTitle]="'Variable Browser'"
      [openHeaderBackgroundColor]="'var(--color-background)'"
      [closedHeaderBackgroundColor]="'var(--color-secondary)'"
      [id]="'VarBrowserID'">
      <p slot="body"><swo-variable-selection-settings
          [editMode]="editMode"></swo-variable-selection-settings></p>
    </keba-accordion>
    <keba-accordion
      [accordionTitle]="'Limiting Values'"
      [openHeaderBackgroundColor]="'var(--color-background)'"
      [closedHeaderBackgroundColor]="'var(--color-secondary)'"
      [id]="'LimitingValuesID'">
      <p slot="body"><swo-variable-value-settings></swo-variable-value-settings></p>
    </keba-accordion>
    [...]
</keba-accordion-group>
```

Neben dem Variablennamen wird immer der aktuelle Wert der Variable aus dem Chart angezeigt. Dieser wird aufgrund der Lesbarkeit nur einmal pro Sekunde aktualisiert. Wenn ein Cursor (siehe Abschnitt 3.6.9) gesetzt ist, wird hier auch der Wert der Variable an der Stelle des Cursors angezeigt. Diese Daten werden vom Chart über einen Angular-Service (siehe Abschnitt 2.1.7) vom Chart zur Verfügung gestellt. Mit dem 'X'-Symbol daneben ist es möglich, Variablen wieder vom Datenrekorder zu entfernen. Wenn eine Variable entfernt wird, ist diese direkt danach auch nicht mehr im Chart zu sehen. Die Implementierung dieses Knopfs erfolgte über die KEBA-Webcomponents (siehe Abbildung 47).

3.7.4 Starten, Stoppen, Pausieren

Die Aufzeichnung kann in der Interaktionsleiste gestartet, gestoppt und pausiert werden (siehe Abbildung 48). Diese Aufrufe werden an die Middleware (siehe Abschnitt 3.5) weitergegeben, diese gibt dann die Aufrufe an die Datenrekorder-REST-API weiter, welche dann die Aktion auf der Steuerung ausführt. Mit dem ersten Knopf wird die Aufnahme gestartet, je nach Konfiguration des Datenrekorders werden Daten im Chart dann entweder sofort, oder nachdem eine Bedingung eingetreten ist, angezeigt. Mit dem zweiten Knopf kann das Anzeigen der Daten pausiert werden. Die Daten werden hier weiterhin aufgezeichnet, aber nicht im Chart aktualisiert. Dies kann zum Beispiel hilfreich sein, wenn während einer Aufzeichnung bestimmte Datenpunkte durch Zoom und Cursor (siehe Abschnitt 3.6.7 und 3.6.9) analysiert werden sollen. Wenn danach wieder der Startknopf betätigt wird, werden die aktuellen Daten wieder angezeigt. Daten, die in der Zwischenzeit aufgezeichnet wurden, werden dann auf einen Schlag alle im Chart angezeigt. Mit dem dritten Knopf kann die Aufzeichnung gestoppt werden, hierbei hört im Gegensatz zum Pausieren auch die Aufzeichnung der Daten auf. Die Daten werden zur Analyse noch im Chart behalten. Erst beim erneuten Starten der Aufzeichnung werden die Daten entfernt.

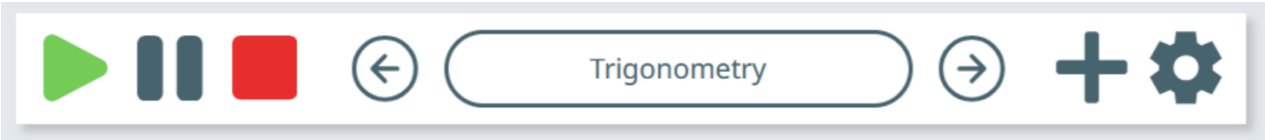

Abbildung 48: Screenshot der Interaktionsleiste

Zur Implementierung der Knöpfe wurden die KEBA-Webcomponents (siehe Abschnitt 2.4.3) verwendet. Insbesondere der 'IconButton', mit dieser Komponente ist es möglich, einen Knopf mit einem integrierten Bild zu implementieren.

3.7.5 Datenrekorder

Datenrekorder (siehe Abschnitt 2.1.2) können in der Interaktionsleiste (siehe Abschnitt 3.4.1) hinzugefügt und konfiguriert werden. Auch ist es möglich, zwischen den angelegten Datenrekordern zu wechseln (siehe Abbildung 48). Damit ist es möglich, mehrere Einstellungsprofile zu speichern. In Datenrekordern werden Variablen mit ihren Einstellungen sowie der Aufzeichnungsmodus gespeichert. Die drei Aufzeichnungsmodi, welche die Anwendung zur Verfügung stellt, sind 'Manual', 'Continuous' und 'Trigger'. Mit diesen Modi kann das Verhalten der Aufzeichnung konfiguriert werden (siehe Abbildung 54).

1. **Manual**: Im 'Manual'-Modus ist es möglich, einen Zeitdauer zu definieren, in dem Daten aufgezeichnet werden sollen. Dieser Modus muss manuell gestartet werden. Danach werden

die Daten von der Middleware (siehe Abschnitt 3.5) in das Chart übertragen. Wenn die Aufzeichnungsdauer abgelaufen ist, bleiben die Daten im Chart stehen.

2. **Continuous**: Im 'Continuous'-Modus werden Daten ununterbrochen aufgezeichnet und in das Chart geschrieben. Dieser Modus muss manuell gestartet werden. Nach dem Start werden laufend die aktuellen Datenpunkte im Chart angezeigt. Dies endet erst, wenn die Aufzeichnung gestoppt wird.

3. **Trigger**: Im 'Trigger'-Modus ist es möglich, Bedingungen zu definieren, bei denen die Aufzeichnung gestartet und gestoppt wird. Dieser Modus muss manuell gestartet werden. Nach dem Starten werden erst Daten angezeigt, wenn die Startbedingung eintritt. Die Aufzeichnung läuft je nach Konfiguration für eine zuvor definierte Dauer oder bis eine Stoppbedingung eintritt. Bei der Konfiguration der Bedingungen können die Variablen, die überwacht werden sollen, die Änderungsmodi ('pos_slope', 'neg_slope', 'pos_neg_slope', 'any_change') und die Schwellwerte eingestellt werden. Mit den Änderungsmodi 'pos_slope' und 'neg_slope' kann angegeben werden, ob aufgezeichnet werden soll, wenn der Wert steigt oder sinkt. Mit dem Änderungsmodus 'pos_neg_slope' ist es egal, ob der Wert steigt oder sinkt, es wird aufgezeichnet, sobald der Schwellwert erreicht wird. Mit 'any_change' können auch nicht numerische Werte überprüft werden, hierbei ist es nicht möglich, einen Schwellwert zu definieren. Damit ist es zum Beispiel möglich, die Aufzeichnung zu starten, wenn der Druck unter 25 Bar sinkt. Dies kann bei der Fehlersuche bei Maschinen helfen.

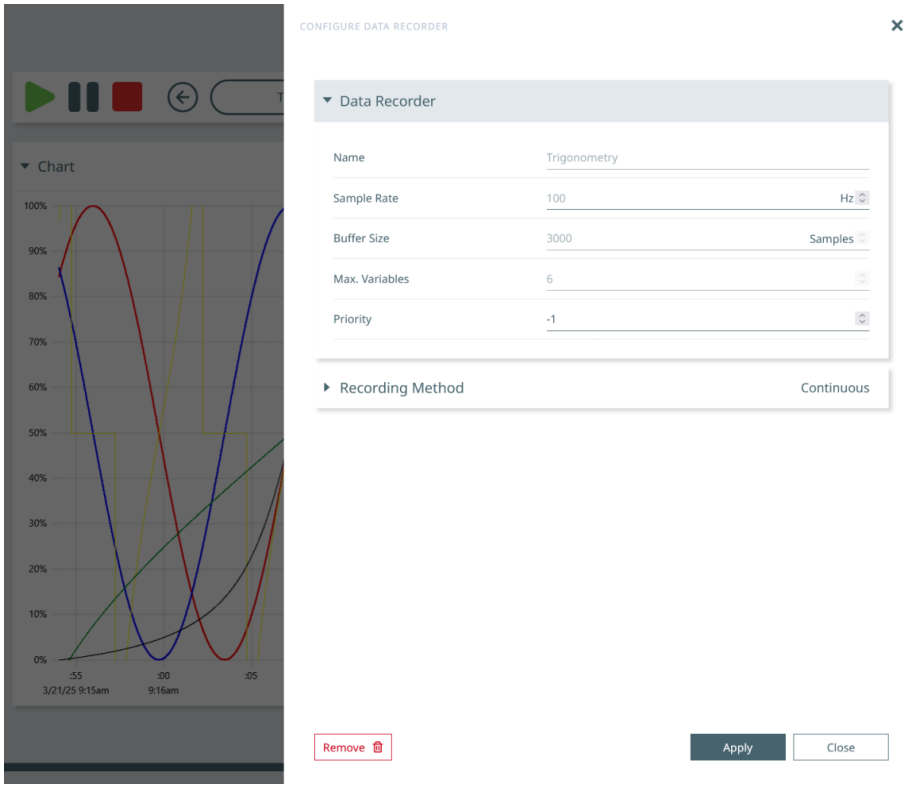

Abbildung 49: Screenshot der grundlegenden Datenrekorder-Konfigurationsmöglichkeiten

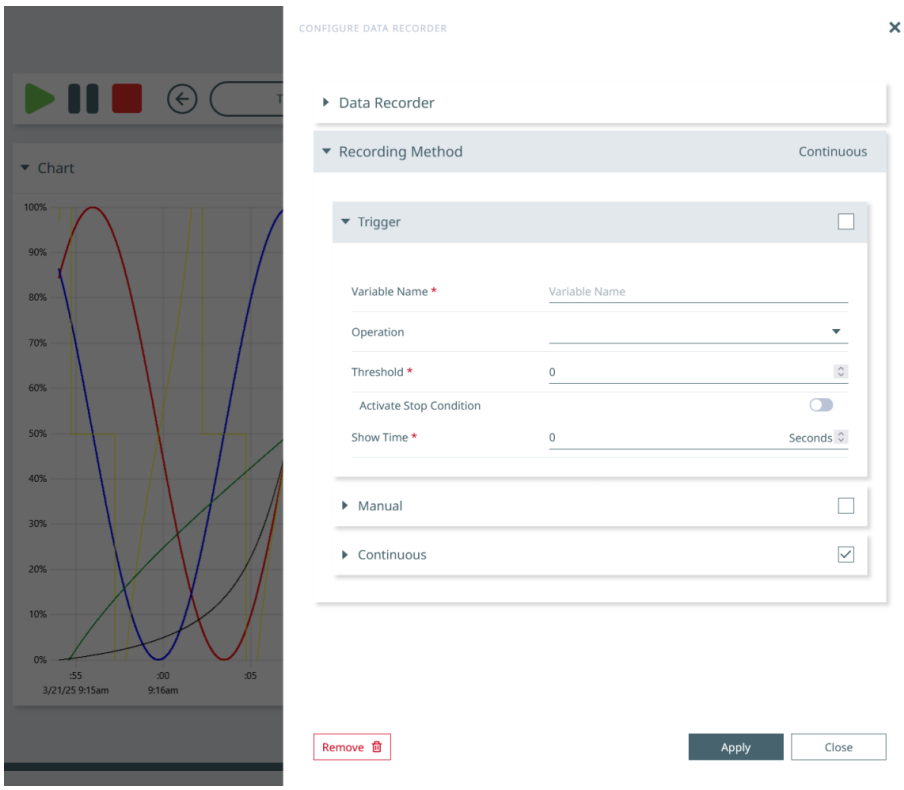

Abbildung 50: Screenshot der Trigger-Spezifischen Datenrekorder-Konfigurationsmöglichkeiten

Beim Anlegen von Datenrekordern können ein Name, die Buffergröße, die maximale Variablenanzahl, die CPU-Priorität und die Aufzeichnungsrate in Hertz definiert werden. Beim Editieren von Datenrekordern können der Name, die CPU-Priorität und die Aufzeichnungsrate aktualisiert werden (siehe Listing 27). Parameter wie die Buffergröße und die maximale Variablenanzahl können nicht aktualisiert werden, da die unterliegende Datenrekorder-REST-API (siehe Abschnitt 2.7.2) es nicht unterstützt, diese Parameter zu editieren. Das ist so, weil beim Anlegen, basierend auf diesen Einstellungen, Speicher reserviert werden muss, und die Größe des zugewiesenen Speichers nicht dynamisch angepasst werden kann. Beim Einstellungs-Panel wird zwischen einem Modus zum Hinzufügen und einem Modus zum Editieren unterschieden. Wenn der Editierungsmodus aktiv ist, werden die nicht möglichen Parameter ausgegraut (siehe Listing 27 Zeile 4-5). Außerdem wird der Knopf zum Löschen eines Datenrekorders nur im Editierungsmodus angezeigt.

Für die Implementierung der Konfigurationsmöglichkeiten der Datenrekorder wurden Reactive Forms (siehe Abschnitt 2.4.2 verwendet. Da sich die Konfiguration zur besseren Wartbarkeit und Übersichtlichkeit in mehrere kleinere Angular-Komponenten (siehe Abschnitt 2.1.6) aufgeteilt hat, wurde ein Service verwendet, der sich um die Handhabung des Reactive-Forms kümmert. In diesem Service werden die Form-Group-Elemente gehalten und die Validierung implementiert.

Listing 27: Auszug des Einstellungsformulars der Datenrekorder

```
<form [formGroup]="dataRecorderSettings.dataRecorderForm" slot="body" novalidate>
  <keba-input
    id="DataRecorderName"
    [required]="!dataRecorderSettings.editMode"
    [disabled]="dataRecorderSettings.editMode"
    [type]="'text'"
    [control]="dataRecorderSettings.dataRecorderForm.controls.dataRecorderName"
    [placeholder]="'newDataRecorder'"
    [label]="'Name'"
    [validationMessage]="'Validation Name'"
  ></keba-input>
  <keba-input
    id="SampleRate"
    [required]="!dataRecorderSettings.editMode"
    [type]="'number'"
    [control]="dataRecorderSettings.dataRecorderForm.controls.sampleRate"
    [placeholder]="'100'"
    [unit]="'Hz'"
    [label]="'Sample Rate'"
    [validationMessage]="'Validation Sample Rate'"
  ></keba-input>
  [...]
</form>
```

Die einzelnen Elemente der Formulare wurden mit den KEBA-Webcomponents (siehe Abschnitt 2.4.3) implementiert. Die KEBA-Webcomponents bieten für die Implementierung von Formularen diverse Eingabefelder, Knöpfe und UI-Komponenten. Da diese Komponenten im gesamten DevAdmin angewendet werden, zieht sich eine geschlossene Designphilosophie durch die gesamte Anwendung. Wie in Listing 27 zu sehen ist, wurden die KEBA-Webcomponents über die Verwendung von Angular-Direktiven an den Anwendungsfall angepasst. Über diese Direktiven wurden ein Typ, ein Platzhalter, eine Beschriftung sowie die Form-Control-Elemente definiert. Dadurch passt sich die Komponente genau an den Anwendungsfall an. Durch den Typ der 'keba-input'-Komponente wird definiert, um welchen Datentyp es sich bei diesem Eingabefeld handelt. Beim Datentyp 'number' ist es zum Beispiel nicht möglich, Text einzugeben.

3.7.6 Chart-Komponente

Die Chart-Komponente (siehe Abschnitt 3.6) ist wie die Variablen ebenfalls in Form eines Akkordeons der KEBA-Webcomponents (siehe Abschnitt 2.4.3) in das Software-Oszilloskop eingebunden. Eine Besonderheit des Akkordeons, in dem die Chart-Komponente integriert ist, besteht darin, dass sich das Akkordeon an das Akkordeon, in dem die Variablen liegen, dynamisch anpasst. Das heißt, dass, wenn das Variablen-Akkordeon eingeklappt ist, das Chart-Akkordeon den frei gewordenen Platz einnimmt und die ganze verfügbare Bildschirmbreite ausfüllt (siehe Abbildung 51 und 52). Wenn der Bildschirm zu klein wird und das Variablen- sowie das Chart-Akkordeon untereinander platziert werden, ändert sich beim Einklappen dieser nichts mehr. Die Chart-Komponente befindet sich in einem Akkordeon, um gewährleisten zu können, dass auch auf kleineren Bildschirmen alles im Blick behalten werden kann.

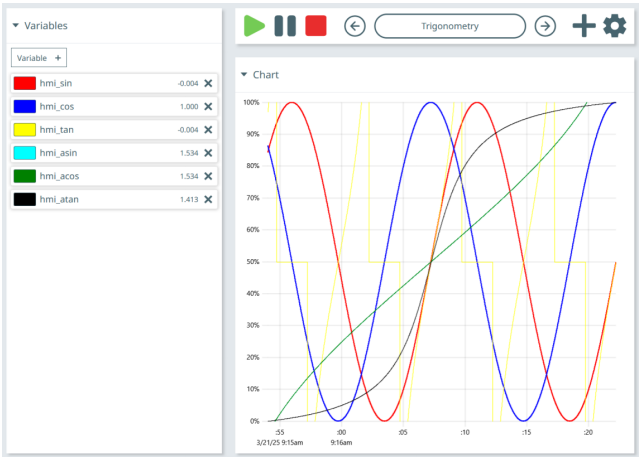

Abbildung 51: Screenshot der ausgeklappten Variablenliste

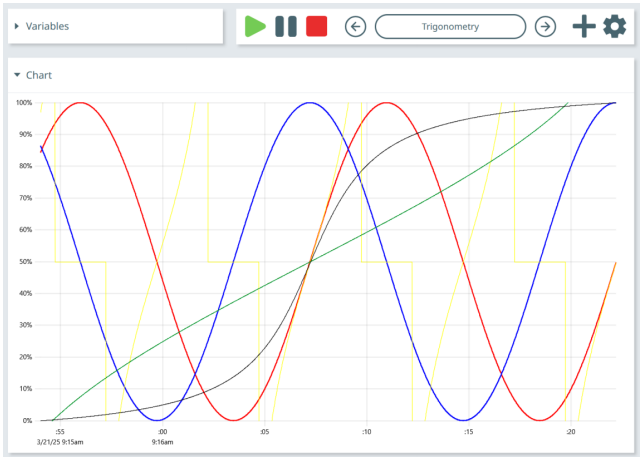

Abbildung 52: Screenshot der eingeklappten Variablenliste

4 Ergebnis

Nachfolgend sind die Ergebnisse dieser Arbeit aufgelistet und beschrieben. Diese bestehen aus fünf Teilen: der Middleware, dem Software-Oszilloskop, dem Design des Software-Oszilloskops, der Chart-Komponente und einem Vergleich von verschiedenen Charting-Libraries.

4.1 Middleware

Die Middleware (siehe Abschnitt 3.5) ist eine TypeScript-Bibliothek (siehe Abschnitt 2.2.4), welche unabhängig von dem Software-Oszilloskop entwickelt wurde. Die Middleware bietet eine TypeScript-Schnittstelle für Datenrekorder (siehe Abschnitt 2.1.2). Sie greift dazu auf zwei verschiedene APIs zu. Die REST-API des Datenrecorders (siehe Abschnitt 2.7.2) wird verwendet, um die Konfiguration der Datenrekorder durchzuführen. Es ist möglich, neue Datenrekorder zu erstellen, diesen Datenrekordern verschiedene Variablen zuzuordnen und diese Datenrekorder in verschiedene Aufnahmemodi zu setzen. Die zweite API ist die Websocket-API (siehe Abschnitt 2.7.1), welche verwendet wird, um die Sensordaten zu empfangen. Diese Sensordaten werden mithilfe eines RxJS-Observables (siehe Abschnitt 2.4.1) weitergegeben.

4.2 Software-Oszilloskop

Das Software-Oszilloskop ist ein Bestandteil des DevAdmin (siehe Abschnitt 2.1.1). Die Integration in den DevAdmin erfolgte mittels Module Federation (siehe Abschnitt 2.1.8). Das Software-Oszilloskop verbindet die Middleware (siehe Abschnitt 3.5) und die Chart-Komponente (siehe Abschnitt 1.3.2). Um die Chart-Komponente zu steuern und zu konfigurieren, welche Variablen in der Chart-Komponente angezeigt werden, bietet das Software-Oszilloskop Steuerelemente und Konfigurationseinstellungen. Mit diesen Einstellungen können diverse Konfigurationsarbeiten vorgenommen werden, die über die Middleware an die Steuerung weitergeleitet werden. Weiters können mehrere Datenrekorder (siehe Abschnitt 2.1.2) angelegt werden. Diesen Datenrekordern können Variablen zugewiesen werden. Beim Durchschalten der Datenrekorder werden auch die Variablen dynamisch angepasst.

4.3 Design

Das Design des Software-Oszilloskops wurde vor der Implementierung detailliert erstellt. Durch die frühe Festlegung eines Designs konnte bei der Implementierung das Hauptaugenmerk auf der Implementierung der Funktionen liegen. Das Design hat sich während der Implementierung nur in einigen Details leicht geändert. Durch die Verwendung der KEBA-Webcomponents (siehe Abschnitt 2.4.3) gliedert sich das Software-Oszilloskop nahtlos in den DevAdmin (siehe Abschnitt 2.1.1) ein. Auch die generelle Strukturierung, Gestaltung und Verwendung des Software-Oszilloskops orientiert sich an anderen Teilen des DevAdmin. Durch ein responsives Design kann das Software-Oszilloskop auf einem breiten Spektrum von Geräten verwendet werden.

4.4 Chart-Komponente

Die Chart-Komponente ist eine eigenständige Angular-Komponente (siehe Abschnitt 2.1.6) und daher weder vom Software-Oszilloskop noch von der Middleware abhängig. Für die Konfiguration der Chart-Komponente können drei Parameter als Eingabeparameter übergeben werden, einer davon ist optional. Notwendig ist einerseits ein RxJS-Observable (siehe Abschnitt 2.4.1), welches dem Chart die Daten liefert, die angezeigt werden. Der zweite benötigte Parameter ist eine Konfiguration für die Linien, die im Chart angezeigt werden. Jede Variable wird als eine Linie im Chart dargestellt. Mit dieser Konfiguration wird dem Chart angegeben, wie viele Linien gleichzeitig angezeigt werden und wie diese Linien aussehen. Der optionale Parameter ist eine Einstellung für das Chart selbst. Hier können verschiedene Anzeigemodi ausgewählt und konfiguriert werden.

In das Chart kann hineingezoomt werden, um einen Teil der angezeigten Datenpunkte genauer zu betrachten. Um Variablen mit unterschiedlichen Wertebereichen in einem Chart anzeigen zu können, können Variablen auf andere Wertebereiche skaliert werden. Um sich den Originalwert einer skalierten Variable ansehen zu können, kann ein Cursor gesetzt werden. Über ein Angular-Service (siehe Abschnitt 2.1.7) können die Originalwerte der Variablen an der Stelle des Cursors abgefragt werden. Da der DevAdmin (siehe Abschnitt 2.1.1) in einem Browser auf einem Computer sowie auch auf einem Steuerungsdisplay (siehe Abschnitt 2.6.2 aufgerufen werden kann, ist es möglich, das Chart auch über Touch-Gesten zu bedienen.

4.5 Evaluierung verschiedener Chart-Bibliotheken

Vor der Erstellung der Chart-Komponente wurden verschiedene JavaScript-Chart-Bibliotheken verglichen. Die Evaluierung teilt sich in drei Runden auf. Grundvoraussetzung für die Vorauswahl der Chart-Bibliotheken ist einerseits die Möglichkeit, mehrere Linien gleichzeitig anzuzeigen. Weiters benötigen die Chart-Bibliotheken einen grafischen Stil, der sich in den DevAdmin (siehe Abschnitt 2.1.1) integrieren lässt, sowie Funktionalitäten wie stilistische Anpassungsmöglichkeiten für einzelne Linien, Zoom- und Skalierungsfunktionen und die Möglichkeit, einen Cursor zu setzen. Für die Evaluierung wurden folgende JavaScript-Chart-Bibliotheken ausgewählt.

1. APEXCHARTS.JS
2. Chart.js
3. CHARTIST.JS
4. dygraphs
5. TOAST UI
6. µPlot

In den drei Evaluierungsrunden lag der Fokus vor allem auf der Performance, da das Chart auch bei zehn Linien mit je 2000 Datenpunkten und neuen Daten im Millisekundenbereich funktionieren und möglichst flüssig sein soll. Weitere wichtige Punkte sind die Dokumentation und der Code, der für das Chart benötigt wird. Hier lag der Fokus auf Verständlichkeit, Wartbarkeit und Entwicklungsdauer.

Die Chart-Bibliotheken dygraphs und µPlot haben es bis in die finale Evaluierungsrunde geschafft. Beide liegen in der Performance weit über den anderen getesteten Chart-Bibliotheken. Am Ende der Evaluierung wurden die Vor- und Nachteile der beiden Bibliotheken verglichen. µPlot wurde ausgewählt und in der Chart-Komponente verwendet.

5 Resümee

5.1 Fazit

Unsere Diplomarbeit, KEBA ChartView, wurde in Zusammenarbeit mit KEBA entwickelt, um Variablendaten von Industriesteuerungen in einem Software-Oszilloskop für Analysezwecke anzuzeigen. Dabei ist das Software-Oszilloskop nahtlos in den DevAdmin (siehe Abschnitt 2.1.1) eingebunden.

Zuallererst möchten wir die gute Zusammenarbeit mit unserem Auftraggeber hervorheben. Durch tatkräftige Unterstützung und regelmäßige Kommunikation konnte unsere Diplomarbeit verwirklicht werden. Wir möchten uns hier noch einmal für die gelungene Zusammenarbeit bedanken.

Da KEBA ChartView sich in drei konkrete Teile aufteilt, gab es eine klare Aufgabentrennung. Die Chart-Komponente (siehe Abschnitt 1.3.2) ist nur über wenige Parameter mit anderen Teilen der Anwendung verbunden und dadurch am weitesten von den anderen beiden Teilen von KEBA ChartView entkoppelt. Das Software-Oszilloskop (siehe Abschnitt 1.3.1) und die Middleware (siehe Abschnitt 3.5) kommunizieren stärker miteinander. Da alle Teile zu Beginn einzeln entwickelt wurden, mussten nach der Zusammenfügung der Einzelteile kleine Anpassungen vorgenommen werden. Durch vorab definierte Schnittstellen konnten große Probleme vermieden werden. Für die Zukunft nehmen wir mit, Einzelteile früher zusammenzufügen, um den Zusammenbau effizienter durchführen zu können.

5.2 Zukunftsausblick

Momentan werden Einstellungen für die Anzeige der einzelnen Linien im Chart direkt im Browser gespeichert, da diese Daten zum Zeitpunkt des Entwickelns unserer Diplomarbeit nicht direkt über die APIs auf der Steuerung gespeichert werden können. Wir haben aber für eine Änderung diesbezüglich den Grundstein gelegt.

Zudem sind die Middleware und die Chart-Komponente eigenständig und können auch außerhalb vom Software-Oszilloskop verwendet werden.

Glossar

API Application programmable Interface

CLI Command Line Interface

CLS Cumulative Layout Shift

CPU Central Processing Unit

CSS Cascading Style Sheets

DOM Document Object Model

FPS Frames per Second

HMI Human Machine Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protcol

IDE Integrated Development Environment

INP Interaction to Next Paint

JSON JavaScript Object Notation

LCP Largest Contentful Paint

MIT Massachusetts Institute of Technology

NPM Node Paket Manager

PLC Programmable Logic Controller

REST Representional State Transfer

RxJS Reactive Extensions for JavaScript

SCSS Sassy Cascading Style Sheets

SPA Single Page Application

SSH Secure Shell

TCP Transfer Control Protcol

UI User Interface

WSL Windows Subsystem for Linux

Literaturverzeichnis

[1] KEBA, „Willkommen bei KEBA," letzter Zugriff am 29.03.2025. Online verfügbar: https://www.keba.com/de/home

[2] ——, „KEBA DevAdmin," letzter Zugriff am 25.03.2025. Online verfügbar: https://www.keba.com/download/x/c4d1a731e1/devadmin_productflyer_en.pdf

[3] Symestic, „Programmable Logic Controllers (PLC) - Definition & Anwendung," letzter Zugriff am 25.03.2025. Online verfügbar: https://www.symestic.com/de-de/was-ist/programmable-logic-controllers

[4] erp information, „Human Machine Interface (HMI) – Components, Examples, and Trends," letzter Zugriff am 26.03.2025. Online verfügbar: https://www.erp-information.com/human-machine-interface

[5] Mozilla, „Document Object Model (DOM)," letzter Zugriff am 14.03.2025. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

[6] Angular, „Components," letzter Zugriff am 24.2.2025. Online verfügbar: https://angular.dev/guide/components

[7] ——, „Dependency Injection," letzter Zugriff am 24.2.2025. Online verfügbar: https://angular.dev/guide/di

[8] webpack, „Module Federation," letzter Zugriff am 26.03.2025. Online verfügbar: https://webpack.js.org/concepts/module-federation

[9] S. Salimi, „Unit Tests," letzter Zugriff am 22.03.2025. Online verfügbar: https://www.agile-academy.com/en/agile-dictionary/unit-tests/

[10] Red Hat, „Was ist eine REST API?" letzter Zugriff am 30.11.2024. Online verfügbar: https://www.redhat.com/de/topics/api/what-is-a-rest-api

[11] R. T. Fielding, „Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvene, 2000.

[12] Node, „Run JavaScript Everywhere," letzter Zugriff am 29.03.2025. Online verfügbar: https://nodejs.org/en

[13] C. Flavio, Potch *et al.*, „Introduction to Node.js," letzter Zugriff am 24.11.2024. Online verfügbar: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

[14] NPM, „Build amazing things," letzter Zugriff am 29.03.2025. Online verfügbar: https://www.npmjs.com/

[15] K. Luke, R. Michael *et al.*, „About npm," letzter Zugriff am 24.11.2024. Online verfügbar: https://docs.npmjs.com/about-npm

[16] Angular, „Angular," letzter Zugriff am 29.03.2025. Online verfügbar: https://angular.dev/

[17] Google, „What is Angular?" letzter Zugriff am 25.11.2024. Online verfügbar: https://angular.dev/overview

[18] ——, „The Angular CLI," letzter Zugriff am 25.11.2024. Online verfügbar: https://angular.dev/tools/cli

[19] ——, „Angular," letzter Zugriff am 27.11.2024. Online verfügbar: https://opensource.google/projects/angular

[20] Angular, „Submission Guidelines," letzter Zugriff am 30.1.2025. Online verfügbar: https://github.com/angular/angular/blob/main/CONTRIBUTING.md#submit

[21] TypeScript, „TypeScript is JavaScript with syntax for types," letzter Zugriff am 29.03.2025. Online verfügbar: https://www.typescriptlang.org/

[22] P. Vogel, „TypeScript-Grundlagen," letzter Zugriff am 27.11.2024. Online verfügbar: https://learn.microsoft.com/de-de/archive/msdn-magazine/2015/january/typescript-understanding-typescript

[23] SASS, „CSS with superpowers," letzter Zugriff am 29.03.2025. Online verfügbar: https://sass-lang.com/

[24] W3Schools, „Introduction to SCSS," letzter Zugriff am 27.11.2024. Online verfügbar: https://www.w3schools.in/scss/introduction

[25] w3Scools, „Take Control - your web, your logo," letzter Zugriff am 29.03.2025. Online verfügbar: https://www.w3.org/html/logo/

[26] W3Schools, „HTML Introduction," letzter Zugriff am 30.11.2024. Online verfügbar: https://www.w3schools.com/html/html_intro.asp

[27] IONOS Redaktion, „Was ist WebSocket?" letzter Zugriff am 30.11.2024. Online verfügbar: https://www.ionos.at/digitalguide/websites/web-entwicklung/was-ist-websocket/

[28] G. SCM, „Git Logo," letzter Zugriff am 30.03.2025. Online verfügbar: https://git-scm.com/downloads/logos

[29] GIT SCM, „About Git," letzter Zugriff am 23.11.2024. Online verfügbar: https://git-scm.com/about

[30] ——, „Git Documentation," letzter Zugriff am 24.11.2024. Online verfügbar: https://git-scm.com/docs/gittutorial

[31] GitLab, „Was ist ein Git-Workflow," letzter Zugriff am 23.11.2024. Online verfügbar: https://about.gitlab.com/de-de/topics/version-control/what-is-git-workflow/

[32] V. S. Code, „Visual Studio Code Logo," letzter Zugriff am 30.03.2025. Online verfügbar: https://code.visualstudio.com/brand

[33] Visual Studio Code, „Visual Studio Code documentation," letzter Zugriff am 23.11.2024. Online verfügbar: https://code.visualstudio.com/docs

[34] ——, „Visual Studio Code download," letzter Zugriff am 08.02.2025. Online verfügbar: https://code.visualstudio.com/download

[35] ——, „Visual Studio Code electron," letzter Zugriff am 08.02.2025. Online verfügbar: https://code.visualstudio.com/blogs/2022/11/28/vscode-sandbox

[36] Docker, „Docker Brand Guidelines," letzter Zugriff am 30.03.2025. Online verfügbar: https://www.docker.com/company/newsroom/media-resources/

[37] Docker, „Was ist ein Container," letzter Zugriff am 27.11.2024. Online verfügbar: https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-container/

[38] ——, „Docker Desktop," letzter Zugriff am 27.11.2024. Online verfügbar: https://www.docker.com/products/docker-desktop/

[39] ——, „Docker Storage," letzter Zugriff am 28.11.2024. Online verfügbar: https://docs.docker.com/engine/storage/

[40] ——, „Docker Architecture," letzter Zugriff am 28.11.2024. Online verfügbar: https://docs.docker.com/get-started/docker-overview/

[41] WSL, „Microsoft," letzter Zugriff am 30.03.2025. Online verfügbar: https://www.elevenforum.com/proxy.php?image=https%3A%2F%2Fstore-images.s-microsoft.com%2Fimage%2Fapps.61786.14131597032361940.38d2a067-3798-455f-934a-f69935156b3d.eb49d3ac-e311-4e6f-b89b-f1fe8db9d73b&hash=8434b2f9376ccaaa6dce990eb7763f6a&return_error=1

[42] Microsoft, „Was ist das Windows-Subsystem für Linux?" letzter Zugriff am 24.11.2024. Online verfügbar: https://learn.microsoft.com/de-de/windows/wsl/about

[43] ——, „Häufig gestellte Fragen zum Windows-Subsystem für Linux," letzter Zugriff am 24.11.2024. Online verfügbar: https://learn.microsoft.com/de-de/windows/wsl/faq

[44] Google, „Chrome DevTools," letzter Zugriff am 16.03.2025. Online verfügbar: https://developer.chrome.com/docs/devtools

[45] D. S. Marthe, „Network panel: Analyze network load and resources," letzter Zugriff am 16.03.2025. Online verfügbar: https://developer.chrome.com/docs/devtools/network/overview

[46] S. E. Kayce Basques, „Debug Progressive Web Apps," letzter Zugriff am 16.03.2025. Online verfügbar: https://developer.chrome.com/docs/devtools/progressive-web-apps

[47] K. Basques, „Console overview," letzter Zugriff am 16.03.2025. Online verfügbar: https://developer.chrome.com/docs/devtools/console

[48] D. S. Marthe, „Memory panel overview," letzter Zugriff am 16.03.2025. Online verfügbar: https://developer.chrome.com/docs/devtools/memory

[49] S. E. Dale St. Marthe, „Performance panel: Analyze your website's performance," letzter Zugriff am 16.03.2025. Online verfügbar: https://developer.chrome.com/docs/devtools/performance/overview

[50] S. Emelianova, „Rendering tab overview," letzter Zugriff am 16.03.2025. Online verfügbar: https://developer.chrome.com/docs/devtools/rendering

[51] Storybook, „StorybookLogo," letzter Zugriff am 09.03.2025. Online verfügbar: https://github.com/storybookjs

[52] ——, „Storybook," letzter Zugriff am 09.03.2025. Online verfügbar: https://storybook.js.org/docs

[53] ——, „StorybookButton," letzter Zugriff am 09.03.2025. Online verfügbar: https://storybook.js.org/docs/get-started/whats-a-story

[54] NX, „Continuous Integration with Nx," letzter Zugriff am 30.11.2024. Online verfügbar: https://nx.dev/ci/intro/ci-with-nx

[55] S. Ashwin, T. Brian *et al.*, „RxJS Logo," letzter Zugriff am 30.03.2025. Online verfügbar: https://rxjs.dev

[56] ——, „RxJS Inroduction," letzter Zugriff am 30.11.2024. Online verfügbar: https://rxjs.dev/guide/overview

[57] ——, „RxJS Installation," letzter Zugriff am 30.11.2024. Online verfügbar: https://rxjs.dev/guide/installation

[58] ——, „Observable," letzter Zugriff am 30.11.2024. Online verfügbar: https://rxjs.dev/guide/observable

[59] Mozilla, „Promise," letzter Zugriff am 30.11.2024. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[60] ——, „Iteration Protocols," letzter Zugriff am 30.11.2024. Online verfügbar: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Iteration_protocols

[61] S. Ashwin, T. Brian *et al.*, „Observer," letzter Zugriff am 30.11.2024. Online verfügbar: https://rxjs.dev/guide/observer

[62] ——, „RxJS Operator," letzter Zugriff am 30.11.2024. Online verfügbar: https://rxjs.dev/guide/operators

[63] ——, „Operator Desicion Tree," letzter Zugriff am 30.11.2024. Online verfügbar: https://rxjs.dev/operator-decision-tree

[64] ——, „Subject," letzter Zugriff am 30.11.2024. Online verfügbar: https://rxjs.dev/guide/subject

[65] Angular, „Making HTTP Requests," letzter Zugriff am 30.11.2024. Online verfügbar: https://angular.dev/guide/http/making-requests

[66] ——, „Reactive forms," letzter Zugriff am 26.03.2025. Online verfügbar: https://v17.angular.io/guide/reactive-forms

[67] Google, „Angular," letzter Zugriff am 16.02.2025. Online verfügbar: https://angular.dev/guide/components/content-projection

[68] L. Sorokin, „µPlot," letzter Zugriff am 09.03.2025. Online verfügbar: https://github.com/leeoniya/uPlot

[69] ——, „µPlot Demos," letzter Zugriff am 16.03.2025. Online verfügbar: https://leeoniya.github.io/uPlot/demos

[70] ——, „µPlot Docs," letzter Zugriff am 16.03.2025. Online verfügbar: https://github.com/leeoniya/uPlot/tree/master/docs

[71] K. Schecker, „@openapitools/openapi-generator-cli," letzter Zugriff am 21.2.2025. Online verfügbar: https://www.npmjs.com/package/@openapitools/openapi-generator-cli

[72] Figma, „Figma Dokumentation," letzter Zugriff am 08.2.2025. Online verfügbar: https://help.figma.com

[73] Chromium, „Chromium," letzter Zugriff am 16.03.2025. Online verfügbar: https://www.chromium.org/Home/

[74] Clockify, „The most popular free time tracker for teams," letzter Zugriff am 16.03.2025. Online verfügbar: https://clockify.me

[75] Microsoft, „Microsoft Teams Logo," letzter Zugriff am 1.04.2025. Online verfügbar: https://www.microsoft.com/en-us/microsoft-teams/group-chat-software?icid=SSM_AS_MicrosoftTeams#Solutions

[76] ——, „Microsoft Teams help & learning," letzter Zugriff am 16.03.2025. Online verfügbar: https://support.microsoft.com/en-us/teams

[77] Overleaf, „Overleaf Projekte," letzter Zugriff am 16.03.2025. Online verfügbar: https://www.overleaf.com/project

[78] ——, „Overleaf Documentation," letzter Zugriff am 16.03.2025. Online verfügbar: https://www.overleaf.com/learn

[79] KEBA, „KeControl C5," letzter Zugriff am 29.03.2025. Online verfügbar: https://www.keba.com/de/industrial-automation/products/controls-ipc/kecontrol-c5-detail

[80] ——, „KeTop AP500," letzter Zugriff am 29.03.2025. Online verfügbar: https://www.keba.com/de/industrial-automation/products/hmi/stationary/ketop-ap-500-detail

[81] AsyncAPI, „Warum AsyncAPI?" letzter Zugriff am 19.2.2025. Online verfügbar: https://www.asyncapi.com/de

[82] Nx, „Monorepo Explained," letzter Zugriff am 16.03.2025. Online verfügbar: https://monorepo.tools/

[83] S. Woltmann, „Was sind Nachteile eines Monorepos," letzter Zugriff am 16.03.2025. Online verfügbar: https://www.happycoders.eu/de/software-craftsmanship/monorepo-vor-und-nachteile/

[84] J. Chhipa und B. Lagunas, „APEXCHARTS.JS," letzter Zugriff am 09.03.2025. Online verfügbar: https://apexcharts.com/

[85] Chart.js, „Chart.js," letzter Zugriff am 09.03.2025. Online verfügbar: https://www.chartjs.org/

[86] CHARTIST.JS, „CHARTIST.JS," letzter Zugriff am 09.03.2025. Online verfügbar: https://gionkunz.github.io/chartist-js/index.html

[87] dygraphs, „dygraphs," letzter Zugriff am 09.03.2025. Online verfügbar: https://dygraphs.com/

[88] NHN, „TOAST UI," letzter Zugriff am 09.03.2025. Online verfügbar: https://ui.toast.com/

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

A Aufgabenverteilung

Nachfolgend wird aufgezählt, welches Teammitglied welche Kapitel der Diplomarbeit verfasst hat.

A.1 Jakob Bruckner

- Zielsetzung (1.2)
- Projektinhalt (1.3)
  - Middleware (1.3.3)
- Projektumfeld (1.4)
- Grundlegende Fachbegriffe (2.1)
  - Datenrecorder (2.1.2)
  - PLC (2.1.3)
  - DOM (2.1.5)
  - Angular Komponente (2.1.6)
  - Angular Service (2.1.7)
  - Unit Testing (2.1.9)
  - Restful (2.1.10)
- Verwendete Technologien (2.2)
- Verwendete Entwicklungssysteme (2.3)
  - NX (2.3.7)
- Verwendete Bibliotheken und Plug-Ins (2.4)
  - OpenAPI-Generator-CLI (2.4.5)
- Verwendete Hardware (2.6)
- Verwendete Schnittstellen (2.7)
- Middleware (3.5)
- Ergebnis (4)
  - Middleware (4.1)

A.2 Alexander Nader

- Verwendete Entwicklungssysteme (2.3)
  - Git (2.3.1)
  - Visual Studio Code (2.3.2)
  - Docker (2.3.3)
  - WSL (2.3.4)
- Verwendete Bibliotheken und Plug-Ins (2.4)
  - RxJS (2.4.1)
  - KEBA Webcomponents (2.4.3)
- Sonstige verwendete Software (2.5)
  - Figma (2.5.1)
- Architektur (3.2)
- Design (3.4)
- Software Oszilloskop (1.3.1)
- Ergebnis (4)
  - Software Oszilloskop (4.4)
  - Design (4.5)

A.3 Julia Gillhofer

- Projektinhalt - Überblick
  - Chart-Komponente (1.3.2)
- Grundlegende Fachbegriffe
  - DevAdmin (2.1.1)
  - HMI (2.1.4)
  - Module Federation (2.1.8)
- Verwendete Entwicklungssysteme (2.3)
  - Chrome DevTools (2.3.5)
  - Storybook (2.3.6)
- Verwendete Bibliotheken und Plug-Ins (2.4)
  - Reactive Forms (2.4.2)
- Sonstige verwendete Software (2.5)
  - Chromium (2.5.2)
  - Clockify (2.5.3)
  - Teams (2.5.4)

  - Overleaf (2.5.5)
- Evaluierung von diversen Chart-Bibliotheken (3.3)
- Chart-Komponente (3.6)
- Ergebnis (4)
  - Chart-Komponente (4.4)
  - Evaluierung verschiedener Chart-Bibliotheken (4.5)
- Resümee (5)

B Meilensteine

Folgende Meilensteine wurden für die Diplomarbeit definiert:

Meilenstein	**Abgabedatum**
Basiskommunikation mit Websocket PLC ist möglich	12.07.2024
Chart-Libraries sind ausgewählt und evaluiert	12.07.2024
Design von Software-Oszilloskop ist fixiert	12.07.2024
Schnittstelle von der Middleware zu Datenrekorder-Services ist definiert	19.07.2024
Chart-View zeigt Daten in der Komponente an	19.07.2024
Design von Software-Oszilloskop ist implementiert	19.07.2024
Datenbasis und eventorientierte Anbindung steht zur Verfügung	26.07.2024
Chart-View bietet Quality-of-Life-Features	26.07.2024
Software-Oszilloskop bietet alle Funktionalitäten	26.07.2024
Dokumentation (Storybook, Schnittstelle) ist vollständig	02.08.2024
Tests sind implementiert	02.08.2024
Schriftliche Arbeit ist fertiggestellt	31.03.2025

Tabelle 2: Zeitplan der Meilensteine für die Entwicklung im Rahmen der Diplomarbeit

Alle Meilensteine wurden nach dem definierten Ziel erreicht und KEBA präsentiert und vorgelegt.

C Logo

Abbildung 53: KEBA ChartView Logo

D Diplomarbeitsplakat

Abbildung 54: KEBA ChartView Plakat