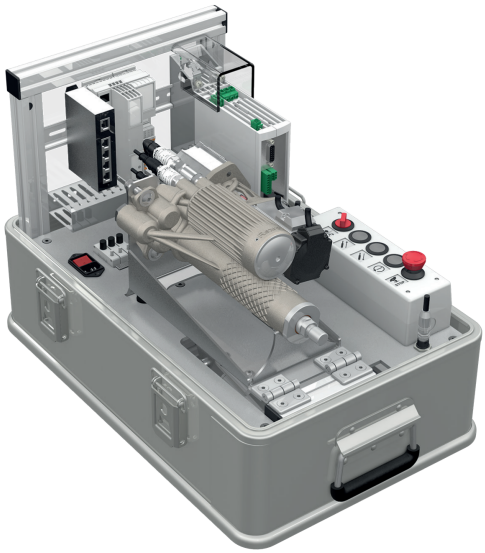
Maschinenautomatisierung für Hydraulikpressen

Bachelorarbeit

Studiengang Informatik

Frühlingssemester 2022



Autoren: Diluxion Marku
Lukas Dätwyler

Betreuer: Prof. Dr. Daniel Politze

OST – Ostschweizer Fachhochschule

Campus Rapperswil-Jona

Abstract

Ausgangslage

Die Plattform ctrlX AUTOMATION kann bereits für die Automatisierung verschiedener Prozesse verwendet werden. In dieser Arbeit soll das Proof-of-Concept einer Applikation zur Automatisierung von Hydraulikanwendungen realisiert werden. Die Anwendungen sollen dafür mit der visuellen Entwicklungsumgebung TIGER-IDE programmiert werden können.

Vorgehen

Mit einem Endkunden wurde eine Vision für das entsprechende Produkt erstellt, von welchem anschliessend die wichtigsten Elemente umgesetzt wurden. Die Implementierung geschah mit dem **ASP.NET Framework** und der **ctrlX AUTOMATION SDK**.

Ergebnis

Die gewünschte Funktionalität der Webapp konnte nicht komplett umgesetzt werden. Dies liegt daran, dass die TIGER-IDE in Entwicklung ist, und noch nicht den gesamten, geplanten Funktionsumfang anbietet. So müsste insbesondere die REST API der TIGER-IDE erweitert werden, damit die erstellte Webapp autonom von der TIGER-IDE App verwendet werden könnte. Ausserdem ist die Verbindung zum Antrieb einer Hydraulikanwendung im aktuellen Zustand nicht genügend zuverlässig und nicht immer möglich.

Inhaltsverzeichnis

Teil I

Management Summary

Management Summary

Beteiligte Personen:

Diluxion Marku	Studierender
Lukas Dätwyler	Studierender
Daniel Politze	Examinator
Ramon Schildknecht	Experte
Markus Stolze	Gegenleser
Josef Müller	Ansprechperson Bosch Rexroth AG

Industriepartner: Bosch Rexroth AG

Problembeschreibung

Die Bosch Rexroth AG möchte die Automatisierung von Hydraulikanwendungen mittels ihrer neuen Plattform ctrlX AUTOMATION vereinfachen. Bisher ist die Programmierung von Hydraulikanwendungen nur über komplizierte Programme und mit Kenntnissen in der SPS Programmierung möglich. Neu soll dies direkt über die visuelle Entwicklungsumgebung TIGER-IDE möglich sein. Diese stellt auch die Schnittstelle zur Presse.

Das Ziel der Arbeit besteht darin, unter Verwendung der TIGER-IDE, ein Proof-of-Concept als Webapp zu implementieren. Dabei sollen mögliche Schwierigkeiten und Fehler erkannt und dokumentiert werden. Die Applikation soll innerhalb der ctrlX AUTOMATION Plattform auf einer ctrlX-Core lauffähig sein.

Abbildung 1: ctrlX Core [1]

Vorgehen

In einem ersten Schritt wurde die Vision des Produktes mit einem Endkunden dokumentiert und die notwendigen Softwarekomponenten daraus extrahiert. Anschliessend wurde der Projektplan erstellt und mit der Entwicklung einer Webapp angefangen. Dies geschah aufgrund der vielen Unbekannten, im agilen Prozessverfahren. Folgende Frameworks, Bibliotheken und Tools wurden dafür eingesetzt.

- **ASP.NET MVC:** Das Grundgerüst der Webapp.
- **Bootstrap:** Für die Gestaltung der einzelnen Elemente des GUI.
- **ctrlX AUTOMATION SDK:** Für die Kommunikation mit dem Datenbroker ctrlX Datalayer.
- **TIGER-IDE:** Für die visuelle Programmierung der Presse innerhalb der Webapp und die Kommunikation mit der Presse selbst.
- **LiteDB:** Für die Speicherung der einzelnen Parametrisierungen.
- **Snapcraft:** Für das Deployment der Webapp auf der ctrlX-Core, welche die Webapp hostet.

Ergebnis

Während ein Teil der Vision auf der Webapp umgesetzt werden konnte, war es nicht möglich, alle Funktionen ohne separate TIGER-IDE App auszuführen, da die API davon noch nicht vollständig vorhanden ist und die App selbst noch Bugs aufweist.

Ausblick

Wenn die TIGER-IDE fertiggestellt ist und Zugriff auf den Quellcode anderer ctrlX AUTOMATION Projekte vorhanden ist, kann mit der Entwicklung einer endgültigen Lösung begonnen werden.

Teil II

Product Documentation

Kapitel 1

Aufgabenstellung

1.1 Ausgangssituation seitens Industriepartner

Mit der neuen Automatisierungs-Plattform ctrlX AUTOMATION [2] soll für einen Pressenhersteller (Lindenberg Technics in Altendorf) eine Maschinenautomatisierung erstellt werden. Die auf Google Blockly [3] basierende TIGER-IDE [4] stellt das Herzstück der Automatisierungslösung dar. Damit soll ermöglicht werden, dass die Bedienperson den Ablauf einer Automatisierung selbständig programmieren kann. Jede Maschine ist anders aufgebaut. Die Anzahl Funktionen und die Parameter der Funktionen variieren. Trotzdem soll jede Maschine möglichst einfach und effizient konfiguriert werden können.

1.2 Konkrete Aufgabenstellung

Die Aufgabe der Bachelorarbeit ist es, eine Webapp zu entwickeln, die als ein vereinfachtes Frontend für die Bedienung einer hydraulischen Presse dienen soll. Zuerst soll eine Vision erarbeitet werden, welche mit der Bosch Rexroth AG und einem Endkunden (Lindenberg Technics AG) abgesprochen wird. Im nächsten Schritt erfolgt die eigentliche Umsetzung. Dafür müssen maschinenspezifische Befehle für die TIGER-IDE entwickelt werden, welche für hydraulischen Pressen funktionieren sollen. Ebenso ein minimales GUI, sowie eine Funktion für die Erstinbetriebnahme der Maschine. Wichtig zu erwähnen ist, dass die Arbeit hauptsächlich als Forschungsarbeit betrachtet werden soll, welche seitens Industriepartner als Proof-of-Concept für eine reale Lösung dient.

1.3 Szenario

Die nachfolgenden Rollen und Szenarien wurden mit Josef Müller abgesprochen.

1.3.1 Rollen

Einrichtende Person	Die einrichtende Person ist ein Mitarbeiter des Kunden, der die Pressen einrichtet. Dies besteht aus der Konfiguration, der Parametrisierung und der Programmierung respektive Testen des Programms.
Bedienende Person	Die bedienende Person steuert die Presse in der automatischen Umgebung und überwacht deren korrekten Ablauf.

1.3.2 Hauptszenario aus Sicht des Einrichters

1. Die einrichtende Person meldet sich mit Benutzernamen und Passwort an.

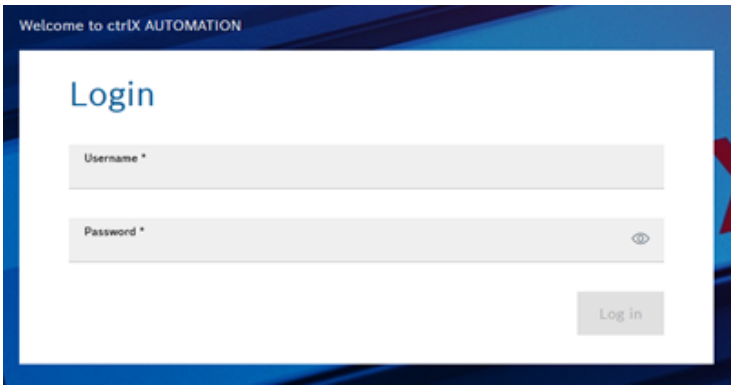


Abbildung 1.1: Screenshot vom Login des ctrlX-Core

2. Die einrichtende Person verbindet die Presse mit der ctrlX-Core.
3. Die einrichtende Person erstellt eine neue Konfiguration für die Presse, die programmiert werden soll.

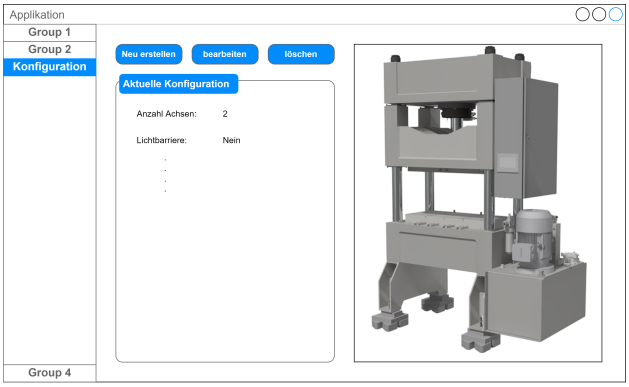


Abbildung 1.2: Konfiguration Übersicht

Beim Erstellen einer neuen Konfiguration oder wenn die bestehende bearbeitet wird, erscheint ein neues Fenster mit dem Formular, um die Daten einzutragen.

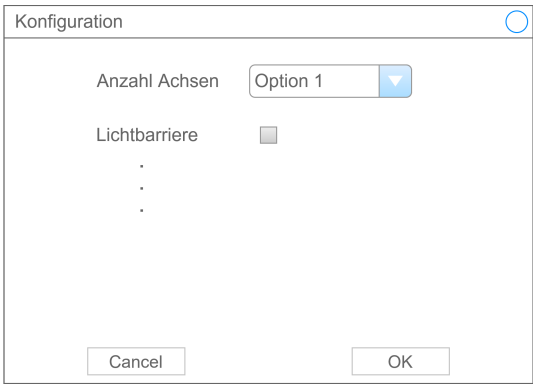


Abbildung 1.3: Konfiguration Formular

4. Die einrichtende Person definiert die Grundparameter für die aktuell gewählte Presse. Dafür steht eine Visualisierung der Presse zur Verfügung, auf dem die Werte eingetragen werden können.

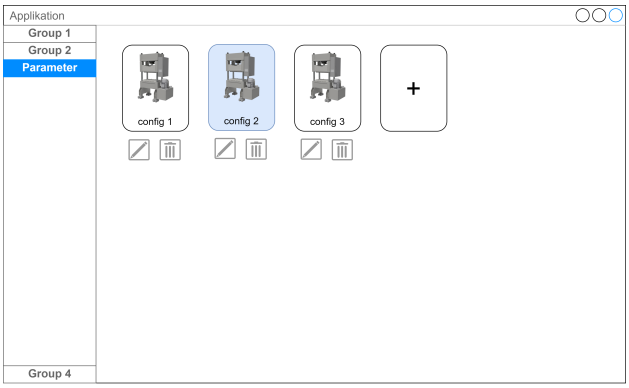


Abbildung 1.4: Parametrisierung Übersicht

Das Editieren oder Erstellen von einer neuen Parametrisierung geschieht in einem separaten Fenster.

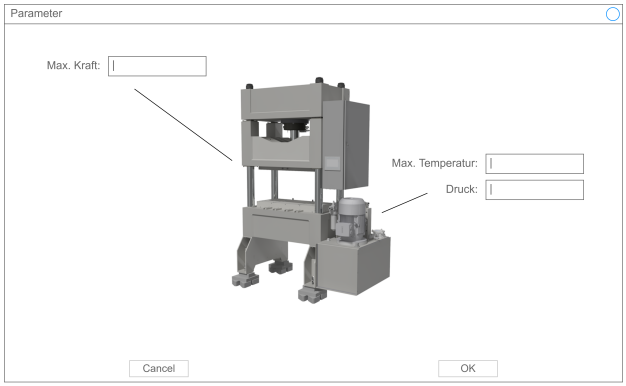


Abbildung 1.5: Parametrisierung Formular

5. Die einrichtende Person erstellt ein neues Programm, wählt dafür die gewünschte Parametervorlage und bearbeitet anschliessend das Programm.

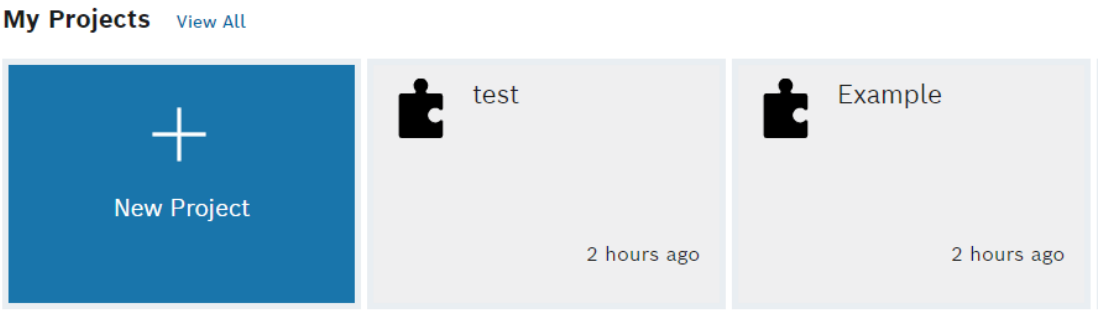


Abbildung 1.6: Screenshot von der Programmwahl der TIGER-IDE

Die visuelle Programmierung findet mit der TIGER-IDE statt.

Abbildung 1.7: Screenshot von der TIGER-IDE

6. Die einrichtende Person testet das erstellte Programm mit der Simulationssoftware und sieht an einem digitalen Abbild, ob sich die Presse gemäss den Vorstellungen verhält.

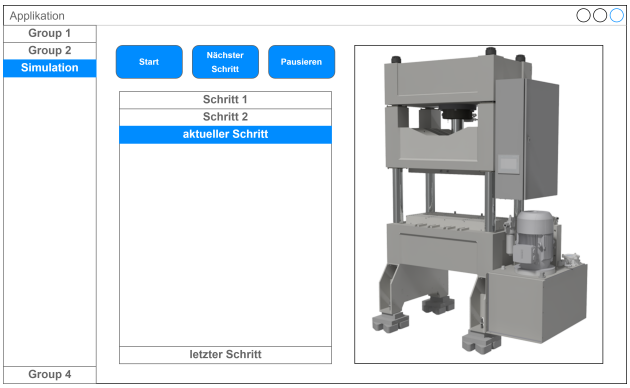


Abbildung 1.8: Simulation

7. Die einrichtende Person speichert das Programm.

1.3.3 Hauptszenario aus Sicht der bedienenden Person

1. Die bedienende Person meldet sich mit Benutzernamen und Passwort an.
2. Die bedienende Person wählt in einer Übersicht das gewünschte Programm aus und startet es.
3. Die bedienende Person überwacht die Presse mittels des Übersichtsbildschirms, auf welchem aktuelle Informationen zur Presse angezeigt werden.

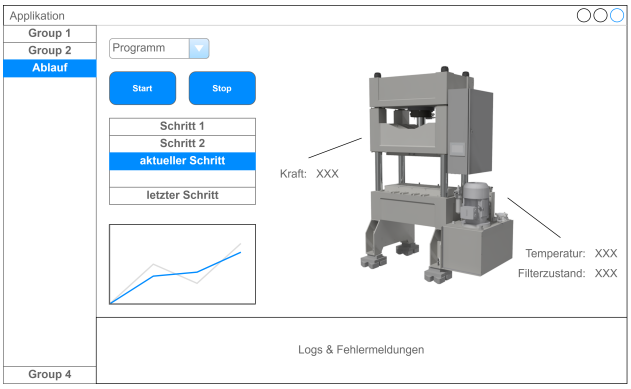


Abbildung 1.9: Ablauf

4. Die bedienende Person beendet den Prozess.

Kapitel 2

Anforderungen

2.1 Identifikation Use Cases

Nachfolgend werden die beiden Szenarios analysiert und daraus die Use Cases (Fett gekennzeichnet) extrahiert, welche nötig sind, um die Funktionalität umzusetzen.

Da die Szenarien verschiedene Rollen beschreiben, liegt es nahe, dass ein **Login**, sowie eine **Nutzerverwaltung** nötig sind. In einem zweiten Schritt wird die Presse an die ctrlX-Core angeschlossen und in der Webapp die **Konfiguration erstellt**. Diese entspricht dem digitalen Abbild der Presse und muss somit nicht veränderbar sein. Die physische Presse wird ebenfalls kaum verändert. Die Konfiguration kann allerdings **gelöscht** werden. Als Nächstes müssen bestimmte Parameter definiert werden. Diese werden in einer veränderbaren **Parametrisierung** gespeichert. Es können jeweils mehrere Parametrisierungen gleichzeitig vorhanden sein, weshalb die gesamte CRUD-Funktionalität (Create, Read, Update, Delete) vorhanden sein muss. Ausserdem muss eine **Parametrisierung ausgewählt** werden können, die definiert, welche Parameter aktuell angewendet werden. Anschliessend wird die Presse **visuell programmiert**. Da ebenfalls immer mehrere Programme vorhanden sein können, ist auch an dieser Stelle die gesamte CRUD-Funktionalität nötig. Anschliessend kann der Ablauf der erstellten Software an einer **Simulation** der Presse dargestellt werden. Wenn das Verhalten wie erwartet ist, kann das Programm nun auf eine reale Presse geladen und dort **ausgeführt** werden. Die Überwachung dieser Ausführung soll mit einer **Anzeige des aktuellen Zustands** vereinfacht werden.

2.2 Priorisierung Use Cases

Die wichtigste Funktion des Produkts ist die visuelle Programmierung einer hydraulischen Presse, sowie die Ausführung eines Programmes. Dies wurde gemeinsam mit Josef Müller von der Bosch Rexroth AG definiert. Folgende Use Cases sind dafür nötig:

1. Erstellen der Konfiguration

2. Löschen der Konfiguration
3. CRUD Parametrisierung
4. Auswählen einer Parametrisierung
5. Visuelle Programmierung
6. CRUD Programmierung
7. Ausführung auf physischer Presse
8. Anzeige des aktuellen Zustands

Die **Nutzerverwaltung** und das **Login** wurden weggelassen, da beides zwar die Usability verbessert, allerdings keine zusätzliche Funktionalität bietet. Ausserdem bietet die ctrlX Plattform bereits beide Funktionen. Im Idealfall würden diese deshalb in die Pressesteuerung eingebunden werden. Ausserdem wurde die **Simulation** zu den möglichen Erweiterungen hinzugefügt, da sie ein praktisches Feature für die einrichtende Person darstellt, jedoch nicht notwendig ist, um eine Presse zu programmieren.
Diese Use Cases, welche Teil des MVPs (Minimal Viable Products) sind, werden anschliessend genauer beschrieben.

Use Case 1: Erstellen der Konfiguration

Nutzer: Die einrichtende Person

Hauptszenario

1. Die einrichtende Person sieht, dass keine Konfiguration vorhanden ist.
2. Die einrichtende Person beginnt mit dem Erstellen einer neuen Konfiguration.
3. Die einrichtende Person definiert, wie viele Achsen die Presse hat.
4. Die einrichtende Person speichert die Konfiguration.

Alternativszenarien

- *a: Die Konfiguration kann auch innerhalb der TIGER-IDE erstellt werden.

Use Case 2: Löschen der Konfiguration

Nutzer: Die einrichtende Person

Hauptszenario

1. Die einrichtende Person sieht, dass eine Konfiguration vorhanden ist.
2. Die einrichtende Person löscht die aktuelle Konfiguration.
3. Die einrichtende Person bestätigt, dass die aktuelle Konfiguration gelöscht werden soll.
4. Die einrichtende Person sieht, dass die Konfiguration nicht mehr vorhanden ist.

Alternativszenarien

- 3a: Die einrichtende Person kann das Löschen auch abbrechen.

Use Case 3: CRUD Parametrisierung

Nutzer: Die einrichtende Person

Hauptszenario

1. Die einrichtende Person sieht alle gespeicherten Parametrisierungen.
2. Die einrichtende Person beginnt mit dem Erstellen einer neuen Parametrisierung.
3. Die einrichtende Person definiert die Punkte: *Maximalposition, Minimal- und Maximalgeschwindigkeit, Minimal- und Maximalbeschleunigung, Minimal- und Maximalkraft* und speichert die Parametrisierung.
4. Die einrichtende Person sieht alle Parametrisierungen, inklusive der neu erstellten.
5. Die einrichtende Person wählt eine Parametrisierung aus und sieht alle ihre Parameter.
6. Die einrichtende Person ändert einen oder mehrere Parameter und speichert die Parametrisierung.
7. Die einrichtende Person löscht eine Parametrisierung.
8. Die einrichtende Person bestätigt, dass er die Parametrisierung löschen möchte.

Alternativszenarien

- 1a: Falls keine Parametrisierungen vorhanden sind, wird dies angezeigt.
- 3a: Das Erstellen kann abgebrochen werden.
- 3b: Falls die Parameter ungültige Werte haben, kann die Parametrisierung nicht gespeichert werden.

- 6a: Das Ändern kann abgebrochen werden.
- 6b: Falls die Parameter ungültige Werte haben, kann die Parametrisierung nicht gespeichert werden.
- 8a: Das Löschen kann abgebrochen werden.

Use Case 4: Auswählen der Parametrisierung

Nutzer: Die einrichtende Person

Hauptszenario

1. Die einrichtende Person sieht alle Parametrisierungen, welche gespeichert sind.
2. Die einrichtende Person wählt eine Parametrisierung aus und erkennt, dass diese aktiviert wurde.

Alternativszenarien

- 1a: Falls keine Parametrisierungen vorhanden sind, wird dies angezeigt.

Use Case 5: Visuelle Programmierung

Nutzer: Die einrichtende Person

Hauptszenario

1. Die einrichtende Person nutzt die TIGER-IDE, um ein Programm für die konfigurierte Presse zu erstellen.

Use Case 6: CRUD Programmierungen

Nutzer: Die einrichtende Person

Hauptszenario

1. Die einrichtende Person sieht alle Programme, welche gespeichert sind.
2. Die einrichtende Person erstellt ein neues Programm. (Gemäss UC 5)
3. Die einrichtende Person wählt ein bestehendes Programm aus, bearbeitet es und speichert die Änderungen.
4. Die einrichtende Person löscht ein bestehendes Programm und bestätigt die Löschung.

Alternativszenarien

- 1a: Falls keine Programme vorhanden sind, wird dies angezeigt.
- 3a: Das Bearbeiten kann abgebrochen werden.
- 4a: Das Löschen kann abgebrochen werden.

Use Case 7: Ausführung auf physischer Presse

Nutzer: Die bedienende und die einrichtende Person

Hauptszenario

1. Die bedienende oder einrichtende Person wählt ein Programm, welches ausgeführt werden soll.
2. Die bedienende oder einrichtende Person startet das Programm, und die Presse führt das Programm aus.

Alternativszenarien

- 1a: Falls das Programm fehlerhaft ist, wird der Fehler angezeigt und das Programm nicht gestartet.

Use Case 8: Anzeige des aktuellen Zustands

Nutzer: Die bedienende und einrichtende Person

Hauptszenario

1. Die bedienende oder einrichtende Person sieht den aktuellen Arbeitsschritt des Programmes.
2. Die bedienende oder einrichtende Person sieht die aktuellen Werte des *Tankdrucks, Öltemperatur, Position, Geschwindigkeit, Beschleunigung und Kraft.*

2.3 Nicht funktionale Anforderungen

Die nicht funktionalen Anforderungen wurden gemäss dem ISO 25010 (der neueren Variante von ISO 9126) Standard [5] dokumentiert. Dieser Standard verschafft einen Überblick über die möglichen Anforderungen, von welchen jene genauer betrachtet wurden, die für das Projekt relevant sind. Nicht genauer definiert wurden deshalb die *Funktionalität* (Bereits in den Use Cases aufgeführt), die *Kompatibilität* (Durch die Containerisierung der ctrlX bereits sichergestellt) und die *Portierbarkeit* (Wird von der ctrlX-Plattform zur Verfügung gestellt).

2.3.1 Performance, Effizienz

Ziel: Alle Aktionen des Nutzers werden in angemessener Geschwindigkeit ausgeführt. Dafür können alle Ressourcen der Hardware verwendet werden.
Massnahmen: Bei ungenügender Performance werden Funktionen asynchron und evtl. auf mehreren Threads verteilt ausgeführt.
Überprüfung: Beim Testen der kompletten Funktionalität durch Josef Müller wird überprüft, ob die Performance den Erwartungen entspricht.

2.3.2 Usability

Ziel: Da die Applikation hauptsächlich als Proof-of-Concept zu verstehen ist, hat die Usability eine sehr tiefe Priorität. Dennoch soll die Applikation mit minimaler Anleitung bedienbar sein.
Massnahmen: Die grafische Oberfläche wird in Anlehnung an das Alternativsystem für elektrische Pressen [6] gestaltet. Somit kann die neue Applikation von allen Nutzenden, welche das SFK kennen, ohne grossen Lernaufwand bedient werden.
Überprüfung: Mit Josef Müller wird ein Acceptance Test durchgeführt.

2.3.3 Zuverlässigkeit

Ziel: Falsch erfasste Werte sollen nicht zu einem Systemunterbruch führen. Falls die Kommunikation mit anderen Softwarekomponenten (TIGER-IDE) fehlschlägt, soll dies ebenfalls nicht zu einem Absturz der Applikation führen.
Massnahmen: Im ASP.NET Framework ist die Überprüfung von Eingabefeldern bereits implementiert [7]. Diese Funktionalität wird genutzt. Die externen Softwarekomponenten werden so eingebunden, dass ein Verbindungsunterbruch nicht zu einem Systemunterbruch führt.
Überprüfung: Mithilfe von Integrationstests werden fehlerhafte Eingaben und Verbindungen getestet.

2.3.4 Sicherheit

Ziel: Die gängigen Sicherheitsrisiken einer Webapp mitigieren.
Massnahmen: Die Nutzerverwaltung wird von der ctrlX-Plattform übernommen, weshalb dafür keine eigenen Sicherheitsmassnahmen nötig sind. Ausserdem wird die Webapp nur lokal auf der ctrlX-Core gehostet, weshalb die meisten Risiken wie XSS oder CSRF nicht relevant sind.
Überprüfung: Die Sicherheitsaspekte der Webapp werden nicht weiter überprüft.

2.3.5 Wartbarkeit

Ziel: Der Quellcode soll für Drittpersonen verständlich sein. Ausserdem soll er modular gestaltet sein, sodass einzelne Komponenten ausgewechselt werden können.

Massnahmen: Es werden die offiziellen Microsoft Coding Conventions [8] eingesetzt. Ausserdem wird Dependency Injection verwendet, um einzelne Services modular einzusetzen.
Überprüfung: Die Coding Conventions werden manuell mittels Tools überprüft und forciert.

2.4 Domänenanalyse

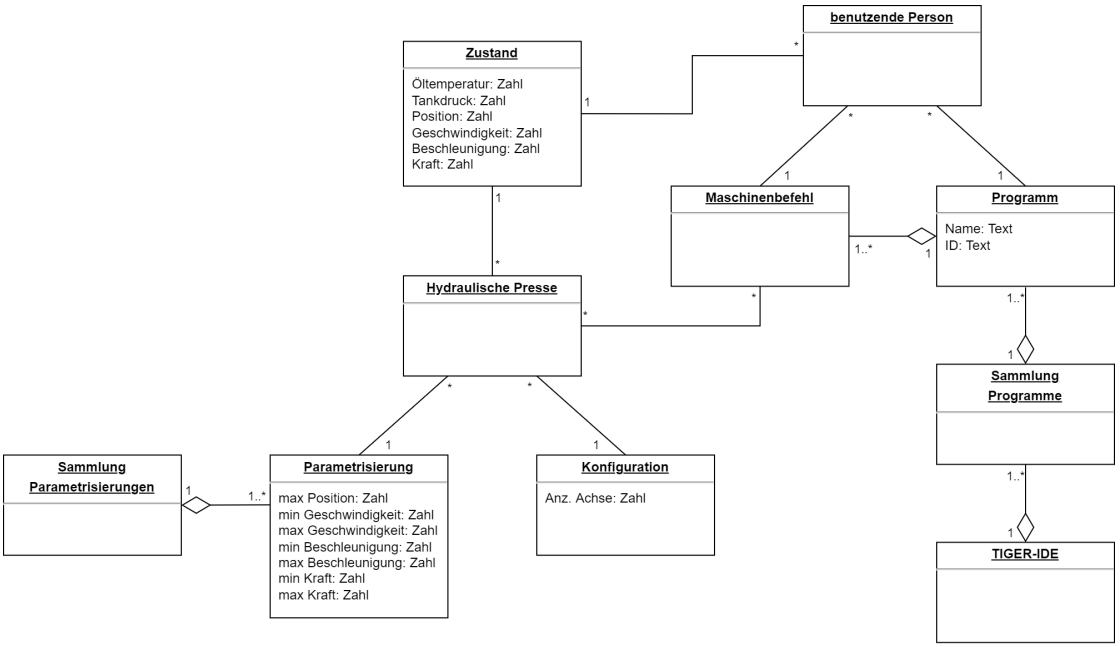


Abbildung 2.1: Domänenmodell

Eine hydraulische Presse hat eine Konfiguration, eine Parametrisierung und einen aktuellen Zustand. Die Parametrisierung hängt von der Konfiguration ab. Dabei kann aus einer Sammlung von Parametrisierungen eine ausgewählt werden, die für die aktuelle Konfiguration der Presse gelten soll. Wichtige, sich verändernde Werte der Presse werden als Zustand beschrieben.
Die Presse kann durch Maschinenbefehle bedient und deren Zustand überprüft werden. Die Befehle hängen von der Konfiguration und der aktuellen Parametrisierung der Presse ab. Es können sowohl die einzelnen Maschinenbefehle eines Programms als auch das Programm selbst bearbeitet werden.
Zu jedem Zeitpunkt können mehrere Programme gespeichert sein, wobei aber immer nur ein ausgewähltes Programm laufen gelassen werden kann. Die Programme befinden sich in der TIGER-IDE.

Begriffserklärung zum Domänenmodell

Hydraulische Presse Ein Antrieb mit einer bzw. mehreren hydraulischen Achsen, die für Pressarbeiten genutzt wird.

Konfiguration Die Konfiguration beschreibt die Anzahl der beweglichen Achsen, die in der Presse vorhanden sind.

Parametrisierung In der Parametrisierung werden die Grenzwerte der Presse für die Position, Beschleunigung, Geschwindigkeit und Kraft festgelegt.

Zustand Der Zustand beschreibt die aktuellen Werte der Presse. Diese sind die Öltemperatur, der Tankdruck, die Position, die Geschwindigkeit, die Beschleunigung und die Kraft.

benutzende Person Person, welche die Presse bedient, unabhängig von ihrer Rolle.

Maschinenbefehl Befehle, die in der TIGER-IDE vorhanden sind und für die Kommandierung der Presse genutzt werden.

Programm Ein Programm enthält mehrere Maschinenbefehle, die nacheinander ausgeführt werden. Im Kontext der TIGER-IDE wird ein Programm als ein Projekt bezeichnet.

TIGER-IDE Eine browserbasierte Entwicklungsumgebung auf der ctrlX-Core Steuerung. Ermöglicht das Erstellen von Programmabläufen sowohl mit reinem Code als auch mit visuellen Elementen.

2.5 Einschränkungen, Abhängigkeiten und Annahmen

Die grösste Abhängigkeit der Webapp besteht zu der TIGER-IDE, welche sich noch in der Entwicklungsphase befindet. Es wird davon ausgegangen, dass die TIGER-IDE zumindest benutzbar ist und die nötige API für die Grundfunktionalitäten bietet.
Im konkreten werden folgende Annahmen getroffen:

1. Die TIGER-IDE kann mit einer hydraulischen Presse kommunizieren.
2. Die TIGER-IDE ermöglicht die externe Benutzung der visuellen Programmierelemente.
3. Die API der TIGER-IDE bietet die Möglichkeit für Datenaustausch.

Je nachdem welche Annahmen nicht erfüllt werden, kann dies zur starken Einschränkung der Funktionalitäten der Webapp führen.

Kapitel 3

Technologien-Stack und Architektur

An dieser Stelle werden die Umgebung des Projekts und die Schnittstellen zu aussenstehenden Softwarekomponenten dokumentiert.

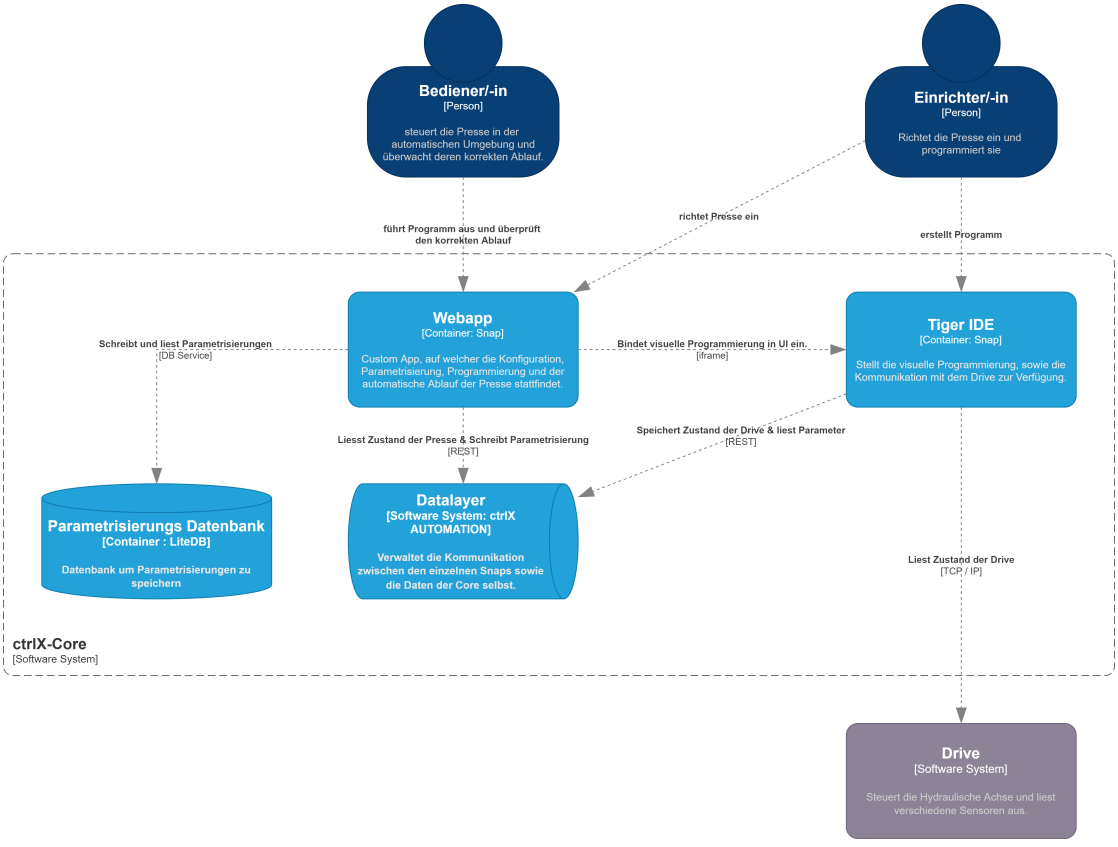


Abbildung 3.1: Containerdiagramm Custom App

3.1 Webapp

Die Webapp wird mit dem ASP.NET Framework [9] implementiert und als Snap [10] innerhalb der ctrlX AUTOMATION Plattform ausgeführt. Intern wird sie als MVC Applikation erstellt, die mit unterschiedlichen Controllern die Funktionalität realisieren.

3.2 TIGER-IDE

Die Webapp kommuniziert auf zwei Arten mit der TIGER-IDE. Zum einen wird die TIGER-IDE in das Frontend eingebunden, sodass die Bearbeitung von Programmen direkt in der Webapp möglich ist. Zum anderen werden die Daten, welche die TIGER-IDE von der Drive erhält über den Datalayer in die Webapp geliefert.

3.3 Datalayer

Der Datalayer [11] ist eine von der ctrlX AUTOMATION Plattform zur Verfügung gestellte Softwarekomponente, welche es ermöglicht, Werte in Nodes zu schreiben. Alle Applikationen auf der Plattform können diese Daten lesen und je nach Berechtigungen auch bearbeiten. Die Kommunikation geschieht über eine REST-Schnittstelle, wobei in der ctrlX AUTOMATION SDK [12] bereits eine Utility Klasse dafür vorhanden ist.

3.4 Parametrisierungs-Datenbank

Die Parametrisierungen werden in einer Datenbank persistent gespeichert. Der Zugriff darauf geschieht von der Webapp, über einen Datenbank Service innerhalb der Webapp.

Kapitel 4

Projektplan

4.1 Arbeitspakete

Der Vollständigkeit halber werden an erster Stelle die Arbeitspakete, welche Teil der Projektplanung sind, aufgeführt. Nachfolgend sind die Use Cases aufgeführt, woraus die Arbeitspakete für die Implementation definiert werden.

4.1.1 Projektplanung

Abklärungen mit der Lindenberg Technics AG

Gemeinsam mit Josef Müller und der Lindenberg Technics AG werden die Wünsche an eine Webapp zur Pressesteuerung aufgenommen und dokumentiert.

Vision

Die Vision beschreibt das gewünschte Endprodukt aus Sicht des Kunden Lindenberg Technics AG, sowie Josef Müller von der Bosch Rexroth AG. Dazu gehören die gewünschte Funktionalität sowie Mockups der Screens, welche angezeigt werden sollen.

Use Cases definieren

Aus der Vision werden die Use Cases definiert und anschliessend mit Josef Müller zusammen priorisiert. Die Use Cases mit hoher Priorität bilden anschliessend das MVP.

Domänenanalyse

Um die Arbeit und Machbarkeit der Use Cases einzuschätzen, wird die Problemdomäne analysiert.

Projekt- und Zeitplanung

Aus den Use Cases des MVP werden die Arbeitspakete extrahiert und somit ein Zeitplan erstellt.

Einrichtung der Infrastruktur

Die für die Entwicklung benötigten Geräte (ctrlX-Core, Demo-Presse, ...) werden von Bosch Rexroth verteilt, sowie die nötige Software (Linux-VM, virtual-Core) bei den Studierenden installiert und aufgesetzt.

Konzeptionierung Grafische Oberfläche

Die Rolle der grafischen Oberfläche der Applikation muss mit dem Kunden, Josef Müller, besprochen werden. Idealerweise sollte gemeinsam, bereits ein grobes Konzept definiert werden.

Integrationstest

Nachdem die Webapp vollumfänglich vorhanden und funktionsfähig ist, soll die korrekte Zusammenarbeit der einzelnen Komponenten in einem Integrationstest überprüft werden.

Abnahme

Das fertige Produkt muss dem Kunden, Josef Müller, präsentiert werden. Mit ihm zusammen wird besprochen, ob die Anforderungen zufriedenstellend erfüllt worden sind.

4.1.2 UC 1: Erstellen der Konfiguration & UC 2: Löschen der Konfiguration

Grundgerüst Webapp

Vor der Implementierung des ersten Use Cases, muss das Grundgerüst für die Webapp erstellt werden. Dafür werden die unterschiedlichen Technologien zum Hosten einer Webapp verglichen und eine passende ausgesucht. Anschliessend muss damit das Grundgerüst einer Webapp erstellt werden, sodass die nachfolgenden Softwarekomponenten eingefügt werden können. In diesem Grundgerüst ist der MVC-Teil bereits enthalten.

Konfiguration der Presse

Eine neue Presse muss konfiguriert werden können. Es ist nötig, abzuklären, ob dies innerhalb einer eigenen View möglich ist, oder ob sie innerhalb der TIGER-IDE konfiguriert werden muss.

4.1.3 UC 3: CRUD Parametrisierung & UC 4: Auswählen der Parametrisierung

Datenbank für Parametrisierung

Es müssen verschiedene Parametrisierungen erstellt, gespeichert und ausgewählt werden können. Damit diese persistent vorhanden sind, ist eine Datenbank nötig, in welcher die Parametrisierungen gespeichert werden.

View: Parametrisierung

In einer View werden die einzelnen Parameter eingetragen. Zudem wird ausgewählt, welche Parametrisierung momentan verwendet wird.

Datalayer Service

Um die Parameter von der TIGER-IDE abzurufen, müssen sie auf dem Datalayer gespeichert werden. Dafür müssen die nötigen Nodes erstellt werden und die Funktionalität muss zur Verfügung stehen, um diese anzupassen.

4.1.4 UC 5: Visuelle Programmierung & UC 6: CRUD Programmierungen

TIGER-IDE Blöcke

Für die TIGER-IDE müssen eigene Programmierblöcke erstellt werden, mit welchen die hydraulischen Achsen bewegt werden können. Diese Blöcke müssen bei neuen Projekten automatisch eingebunden werden.

Auswahl Programmierung

Es müssen die vorhandenen Programme angezeigt werden, sodass eines ausgewählt werden kann. Ausserdem sollen neue erstellt und bestehende gelöscht werden können.

View: Visuelle Programmierung

Die visuelle Programmierung mit der TIGER-IDE soll direkt in die Webapp inkludiert werden, sodass es nicht nötig ist, die TIGER-IDE als separate Webapp zu starten.

4.1.5 UC 7: Ausführung auf physischer Presse & UC 8: Anzeige des aktuellen Zustands

Programm starten

Die Programme müssen gestartet werden können. Während dem Ablauf soll der aktuelle Arbeitsschritt aufgezeigt werden.

View: Programmablauf

Während dem Programmablauf müssen ausserdem die verschiedenen Werte (Tankdruck, Öltemperatur, Position, Geschwindigkeit, Kraft und Beschleunigung) angezeigt werden.

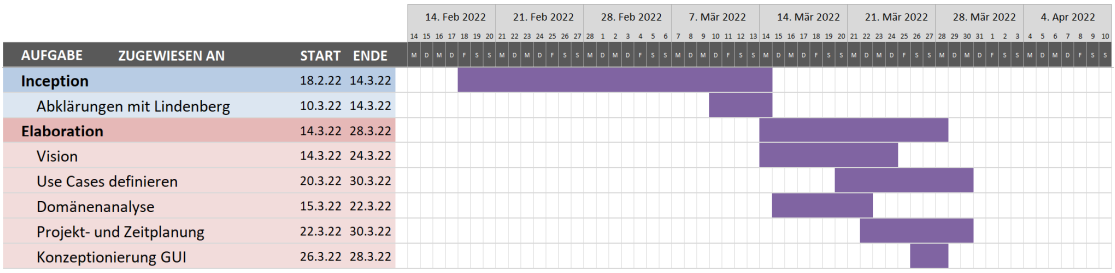
4.2 Phasen

Nachfolgend werden die Arbeitspakete zur Übersicht, den vier Phasen des klassischen Rational Unified Process zugeordnet.

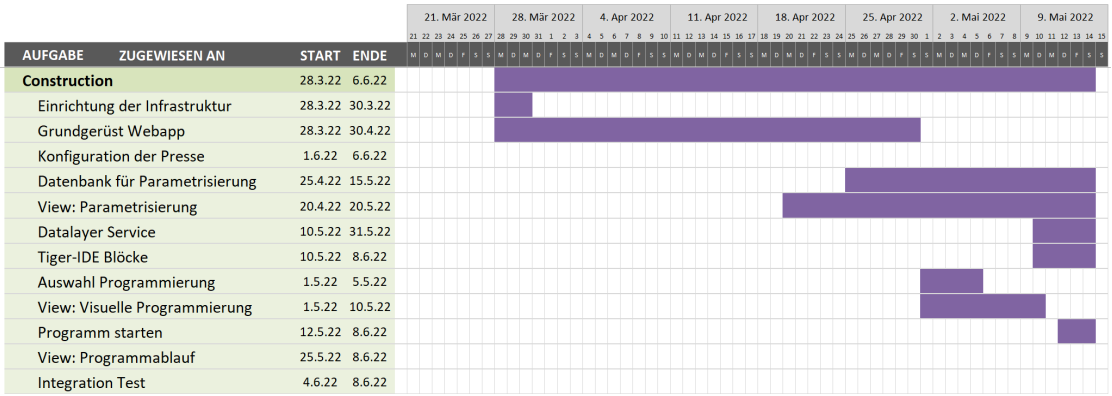
- 18.02.22 - 14.03.22 Inception
  - Abklärungen mit der Lindenberg Technics AG
- 14.03.22 - 28.03.22 Elaboration
  - Vision
  - Use Cases definieren
  - Domänenanalyse
  - Projekt- und Zeitplanung
  - Konzeptionierung Grafische Oberfläche
- 28.03.22 - 06.06.22 Construction
  - Einrichtung der Infrastruktur
  - Grundgerüst Webapp
  - Konfiguration der Presse
  - Datenbank für Parametrisierung
  - View: Parametrisierung
  - Datalayer Service
  - TIGER-IDE Blöcke
  - Auswahl Programmierung
  - View: Visuelle Programmierung
  - Programm starten
  - View: Programmablauf
  - Integrationstest
- 06.06.22 - 13.06.22 Transition
  - Abnahme

4.3 Gantt Diagramm

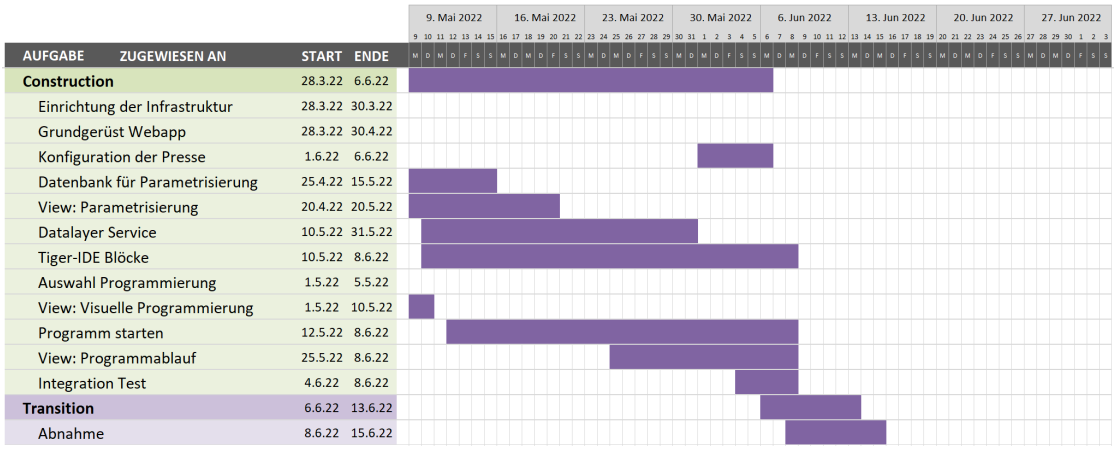
Der Zeitplan der Phasen wurde zu Beginn festgelegt. Die Daten, welche bei den einzelnen Arbeitspaketen stehen, sind die realen Daten. Die groben Abweichungen werden nachfolgend erklärt.



AUFGABE	ZUGEWIESEN AN	START	ENDE
Inception		18.2.22	14.3.22
Abklärungen mit Lindenberg		10.3.22	14.3.22
Elaboration		14.3.22	28.3.22
Vision		14.3.22	24.3.22
Use Cases definieren		20.3.22	30.3.22
Domänenanalyse		15.3.22	22.3.22
Projekt- und Zeitplanung		22.3.22	30.3.22
Konzeptionierung GUI		26.3.22	28.3.22



AUFGABE	ZUGEWIESEN AN	START	ENDE
Construction		28.3.22	6.6.22
Einrichtung der Infrastruktur		28.3.22	30.3.22
Grundgerüst Webapp		28.3.22	30.4.22
Konfiguration der Presse		1.6.22	6.6.22
Datenbank für Parametrisierung		25.4.22	15.5.22
View: Parametrisierung		20.4.22	20.5.22
Datalayer Service		10.5.22	31.5.22
Tiger-IDE Blöcke		10.5.22	8.6.22
Auswahl Programmierung		1.5.22	5.5.22
View: Visuelle Programmierung		1.5.22	10.5.22
Programm starten		12.5.22	8.6.22
View: Programmablauf		25.5.22	8.6.22
Integration Test		4.6.22	8.6.22



AUFGABE	ZUGEWIESEN AN	START	ENDE
Construction		28.3.22	6.6.22
Einrichtung der Infrastruktur		28.3.22	30.3.22
Grundgerüst Webapp		28.3.22	30.4.22
Konfiguration der Presse		1.6.22	6.6.22
Datenbank für Parametrisierung		25.4.22	15.5.22
View: Parametrisierung		20.4.22	20.5.22
Datalayer Service		10.5.22	31.5.22
Tiger-IDE Blöcke		10.5.22	8.6.22
Auswahl Programmierung		1.5.22	5.5.22
View: Visuelle Programmierung		1.5.22	10.5.22
Programm starten		12.5.22	8.6.22
View: Programmablauf		25.5.22	8.6.22
Integration Test		4.6.22	8.6.22
Transition		6.6.22	13.6.22
Abnahme		8.6.22	15.6.22

- Die Konfiguration der Presse wurde an den Schluss der Implementierung verschoben, da zu Beginn unklar war, wie dies direkt in der Webapp gelöst werden kann und somit erst Abklärungen mit dem Entwicklungsteam der TIGER-IDE nötig waren. Ausserdem konnte notfalls dafür direkt die TIGER-IDE verwendet werden.

- Generell, nahmen einige Teile der Implementation mehr Zeit in Anspruch als geplant, was vor allem mit der starken Abhängigkeit zur TIGER-IDE und der damit verbundenen Kommunikation mit den Entwicklern zusammenhing.

4.4 Meilensteine

Aus den Phasen wurden folgende Meilensteine hergeleitet:

Nr.	Titel	gepl. Datum	eff. Datum
1	Vision fertiggestellt	14. 3. 2022	24. 3. 2022
2	Projekt- und Zeitplanung fertig	21. 3. 2022	30. 3. 2022
3	Grundgerüst Webapp auf Core implementiert	24. 4. 2022	30. 4. 2022
4	Webapp komplett funktionsfähig	6. 6. 2022	8. 6. 2022
5	Produkt vom Kunden abgenommen	13. 6. 2022	15. 6. 2022

Die Verzögerungen, vor allem in den ersten zwei Meilensteinen, lassen sich dadurch erklären, dass zu Beginn mehr Abklärungen als erwartet nötig waren, um die genaue Problemlage zu analysieren. Die Komplexität der ctrlX AUTOMATION Plattform wurde etwas unterschätzt.

4.5 Risikomanagement

Die Risikoanalyse zeigt identifizierte Risiken der Bachelorarbeit auf und dokumentiert Einschätzungen und Mitigationsstrategien.

Nr	Titel	Beschreibung	Eintritts-wahrschein-lichkeit	Gewichteter Schaden
R1	Probleme TIGER-IDE	Da die Tiger-IDE von einem externen Entwicklungsteam bereitgestellt wird, führen Bugs und deren Fehlerbehebung zu langen Wartezeiten.	Mittel	5
R2	Kommunikation Presse	Die eigentliche Kommunikation mit der Presse wird über die Tiger-IDE geregelt. Da versucht wird, mit der Webapp extern auf diese Daten zuzugreifen und diese zu modifizieren, können unerwartete Aufwände entstehen.	Mittel	3
R3	Kommunikation ctrlX-Core	Das Betriebssystem der ctrlX-Core basiert auf Ubuntu Core und setzt dementsprechend auf Container-basierte Applikationen. Durch die Kapselung der Applikationen können unerwartete Zusatzaufwände entstehen.	Klein	3
R4	.NET / C#	Beiden Studierenden fehlt Erfahrung in der Entwicklung mit .NET Core, wodurch natürlicherweise Zusatzaufwände entstehen können.	Klein	1
R5	Snapcraft	Dies ist eine komplett neue Technologie für die Studierenden und könnte dementsprechend unerwartete Aufwände und Verzögerungen verursachen.	Klein	2
R6	Datalayer	Dies ist eine spezifische Bibliothek der ctrlX Umgebung, die von der Bosch Rexroth zur Verfügung gestellt wird. Deshalb ist der Wissensaustausch mit anderen Programmierern nur bedingt möglich.	Mittel	2

4.5.1 Umgang mit den Risiken

R1: Probleme TIGER-IDE

Um die Wartezeiten zu minimieren, werden gefundene Bugs und Fehlverhalten der TIGER-IDE sofort dem Entwicklungsteam gemeldet. Während der Wartezeit auf eine Lösung wird an anderen Stellen der Webapp weiterentwickelt, wo die Tiger-IDE nicht benötigt wird.

R2: Kommunikation Presse

Falls die Aufwände zu gross sind oder es keine Möglichkeiten gibt, die Daten extern zu modifizieren, wird die Kommunikation mit der Presse, nur über die TIGER-IDE geregelt.

R3: Kommunikation ctrlX-Core

Hier kann entweder selbständig nach anderen Lösungsmöglichkeiten für das spezifische Problem gesucht oder im Notfall das Entwicklungsteam um Hilfe gebeten werden.

R4: .NET / C#

Gegenseitige Unterstützung der Studierenden und Recherche in der .NET Dokumentation. Ausserdem können Dozenten der Fachhochschule OST bei spezifischen Problemen gefragt werden.

R5: Snapcraft

Die zur Verfügung stehenden Sample Apps des Entwicklungsteams als Hilfestellung und Recherche in der Snapcraft Dokumentation.

R6: Datalayer

Die zur Verfügung stehenden Sample Apps des Entwicklungsteams als Hilfestellung und möglicherweise die ctrlX Developer Community um Hilfe bitten.

4.5.2 Eingetretene Risiken

Das Risiko R1 ist eingetreten und beeinflusste das Projekt an mehreren Stellen. Mehr dazu im Kapitel 6.2.

4.6 Infrastruktur

Die folgende Infrastruktur wurde verwendet:

Hardware

- PCs und Notebooks der Studierenden mit Windows 10
- ctrlX-Core Steuerung von Bosch Rexroth
- Demo-Koffer mit hydraulischer Presse von Bosch Rexroth

Software

- **LaTeX:** Die Dokumentation wird in der Sprache LaTeX geschrieben.
- **Gitlab:** Auf dem Gitlab der Fachhochschule OST wird die Dokumentation gespeichert. Mittels der CI/CD Pipeline werden direkt lesbare PDFs erstellt.
- **Clockify:** Mittels Clockify wird die verwendete Arbeitszeit notiert und ausgewertet.
- **Visual Studio Code:** Zur Entwicklung der Webapp verwendet.
- **ASP.NET Core:** Framework zur Entwicklung der Webapp.
- **xUnit:** Zum Schreiben von Unit Tests.

- **Moq:** Zum Erstellen von Mockups bei Unit Tests.
- **Snapcraft:** Zum Generieren der Snaps aus dem Quellcode.
- **Ubuntu 20 VM:** Wird benötigt, da Snapcraft nur auf Linux Betriebssystemen funktioniert.
- **ctrlX SDK:** Bibliothek für die proprietären Technologien von ctrlX AUTOMATION.
- **ctrlX VirtualCore:** Zum Testen der Webapp auf einer virtuellen Umgebung.
- **TIGER-IDE:** Tool für die visuelle Programmierung der Presse und für die Kommunikation damit.
- **IndraWorks Ds:** Bisheriges Tool für die Konfiguration, Parametrisierung und Inbetriebnahme der Presse.
- **Postman:** Tool zum Testen der REST API von der TIGER-IDE wie auch der ctrlX-Core.
- **signalR:** Zur Echtzeitübertragung von Daten an die Views.
- **SonarQube:** Zur Analyse des C# Codes.

Kapitel 5

Softwareimplementierung

Die Webapp setzt sich aus mehreren Komponenten zusammen. Die Controller spiegeln die verschiedenen funktionalen Anforderungen wider. Die vier Services sind für die Kommunikation mit jeweils einer externen Schnittstelle zuständig.

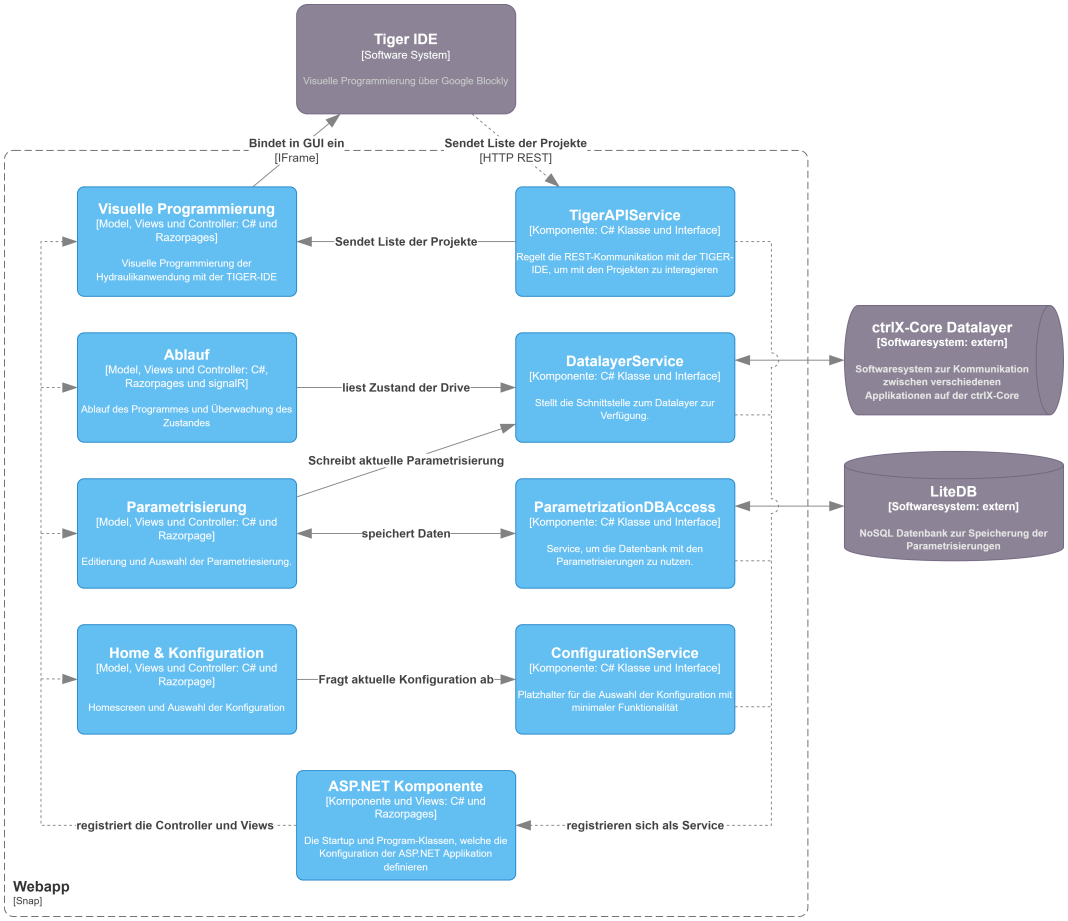


Abbildung 5.1: Komponentendiagramm der Webapp App

Es werden Model-Klassen für Konfiguration, Parametrisierung, Programm und Zustand erstellt. Die Komponenten und ihre Verbindungen sind auf der nächsten Seite als Komponentendiagramm des C4-Models [13] dokumentiert. Die meisten Komponenten bestehen aus mehreren Klassen.

Die Softwarekomponente namens *ASP.NET-Komponente* steht stellvertretend für die Funktionalität, welche durch das ASP.NET Framework zur Verfügung gestellt wird. Die vier Komponenten *Home*, *Parametrisierung*, *Visuelle Programmierung* und *Ablauf* erfüllen jeweils einen Teil der funktionalen Anforderungen und enthalten immer einen Controller, ein Model, sowie eine oder mehrere Views in Form von Razor Pages.

5.1 ASP.NET Web App

Die Webapp wurde mit dem ASP.NET Framework von .NET realisiert. Dieses wurde aus mehreren Gründen gewählt. Zum einen, da für die ctrlX-Plattform Libraries in C# vorhanden sind. So kann ein Grossteil der Entwicklung in derselben Sprache geschehen. Ansonsten bietet nur JavaScript mit dem *Node.js* Framework den vollen Funktionsumfang der ctrlX-Plattform. Ausserdem haben beide Studierenden Erfahrung in der Entwicklung von klassisch objektorientierten Sprachen wie Java oder C#. Als dritter Grund ist die ausführliche Dokumentation von .NET im Generellen zu erwähnen.

5.1.1 MVC

Im ASP.NET Umfeld gibt es drei verschiedene Lösungen für das UI einer Webapp. *ASP.NET Core Razor Pages*, *ASP.NET Core MVC* und *Blazor* [14]. Die Razor Pages und MVC erstellen das UI serverseitig, während es bei Blazor direkt beim Client generiert wird. Für die Bachelorarbeit wurde die Variante MVC gewählt. Der Grund dafür war, dass die Studierenden bereits mit dem MVC Pattern vertraut sind und dass die Struktur der Files damit verständlicher war als bei den anderen Varianten. Ausserdem konnte die URL in der MVC Variante einfacher angepasst werden, sodass das Routing auch auf der ctrlX-Plattform funktioniert. Anzumerken ist, dass auch die MVC Variante Razor Pages für die Views verwendet, jedoch keine Code-Behind Files, wie dies bei *ASP.NET Core Razor Pages* eingesetzt wird.

- Das Model besteht aus einer C# Datei, welche die Businesslogik der Komponente enthält.
- Die View wird als .cshtml Datei umgesetzt. Diese beinhaltet html, welches das UI definiert, sowie Logik, welche beispielsweise verwendet wird, um alle Elemente einer Liste darzustellen und dynamisch zu aktualisieren.
- Der Controller ist eine C# Datei, in der die Interaktion mit dem User definiert ist. Der Controller wählt aus, welche View dargestellt wird.

5.1.2 Routing

Die ASP.NET Implementation des MVC-Patterns übernimmt sehr viele Aufgaben des Entwicklers. Das Routing funktioniert beispielsweise mit dem Namen der Methode im Controller. Dies wird in der Startup Datei einmal definiert und kann anschliessend implizit verwendet werden.

```
app.UseEndpoints(endpoints => {
    endpoints.MapControllerRoute(
        name: "default",
        pattern:
          "pressesteuerung/{controller=Home}/{action=Index}/{id?}");
});
```

Wenn die URL nur aus einem „/“ besteht, wird standardmässig die Methode Index des HomeController ausgeführt. Um weitere Views hinzuzufügen, muss lediglich ein neuer Controller erstellt und für alle nötigen Aktionen eine Methode definiert werden. Die UseRouting-Middleware sorgt anschliessend dafür, dass bei einem Aufruf die korrekte Methode ausgeführt wird.

5.1.3 Dependency Injection

Ebenfalls Teil von ASP.NET ist ein Dependency Injection Framework. Benötigte Services werden als *Transient*, *Scoped* oder *Singleton* definiert und anschliessend automatisch in die benötigten Klassen (meistens Controller) injiziert. Dies geschieht per Konstruktor dieser Klassen. Das heisst, dass die Startup Klasse, in der die Konfiguration der Webapp geschieht, nur von den Interfaces der Services abhängt. Die konkreten Services und auch die Controller sind komplett austauschbar. In der Webapp sind alle Services als Singleton initialisiert, da sie genau einmal definiert werden sollen, und ihre Lebenszeit während der gesamten Nutzungsdauer der App bestehen soll.

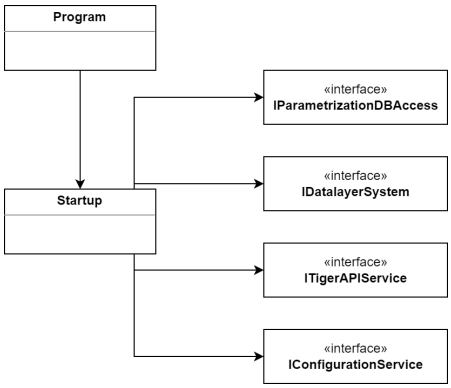


Abbildung 5.2: Klassendiagramm aus Sicht Startup

```
services.AddSingleton<IParametrizationDBAccess>(sp => new
    ParametrizationDBAccess());
services.AddSingleton<IDatalayerSystem>(sp => new
    DatalayerSystem());
services.AddSingleton<IDataLayerService>(sp =>
{
    var parametrizationDBAccess =
        sp.GetService<IParametrizationDBAccess>();
    var datalayerSystem = sp.GetService<IDatalayerSystem>();
    return new DataLayerService(parametrizationDBAccess,
        datalayerSystem);
});
// weitere Services deklarieren
```

Der zweite Service, *DataLayerService* hängt selbst von dem *ParametrizationDBAccess* ab. Dieser kann ebenfalls per Konstruktor eingefügt werden, was beispielsweise folgendermassen geschieht:

```
public class DataLayerService : IDataLayerService
{
    private readonly IParametrizationDBAccess
        _parametrizationDBAccess;
    private readonly IDataLayerService _dataLayerService;

    public DataLayerService(IParametrizationDBAccess
        parametrizationDBAccess, IDatalayerSystem datalayerSystem)
    {
        _parametrizationDBAccess = parametrizationDBAccess;
        _dataLayerService = dataLayerService;
    }
}
```

5.2 Home

Von der *Home* View aus kann mit dem *Start* Button, auf den Configuration-Screen gewechselt und mit der Neukonfiguration einer Presse begonnen werden. Für das digitale Abbild der Hydraulikanwendung muss vorerst nur die Anzahl der Achsen definiert werden. Dementsprechend enthält das *Configuration* Model nur eine Property namens *NrOfAxes*. Da diese Webapp als Proof-of-Concept dienen soll und gemäss den Entwicklern die TIGER-IDE aktuell nur mit einer hydraulischen Achse getestet wurde, wird auch hier nur das 1-Achsensystem unterstützt. Allerdings, kann die mögliche Anzahl der Achsen, durch die nun vorhandenen Klassen für die Configuration-Komponente der Webapp, einfach erweitert werden.

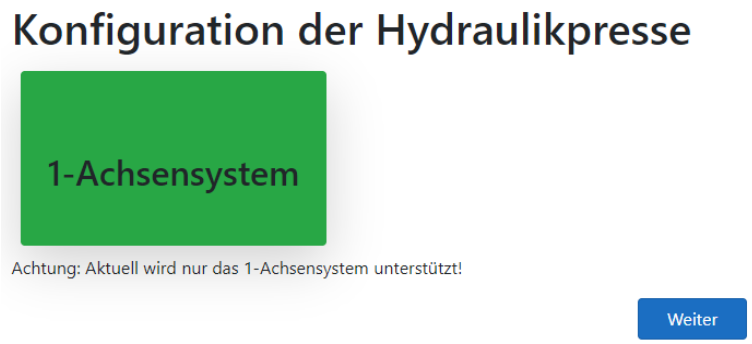


Abbildung 5.3: Configuration Screen

5.3 Parametrisierung

Die Auswahl und das Editieren von Parametrisierungen finden im gleichnamigen Controller statt. Dieser hängt deshalb von den Interfaces der beiden Services, der Datenbank und des Datalayers ab.

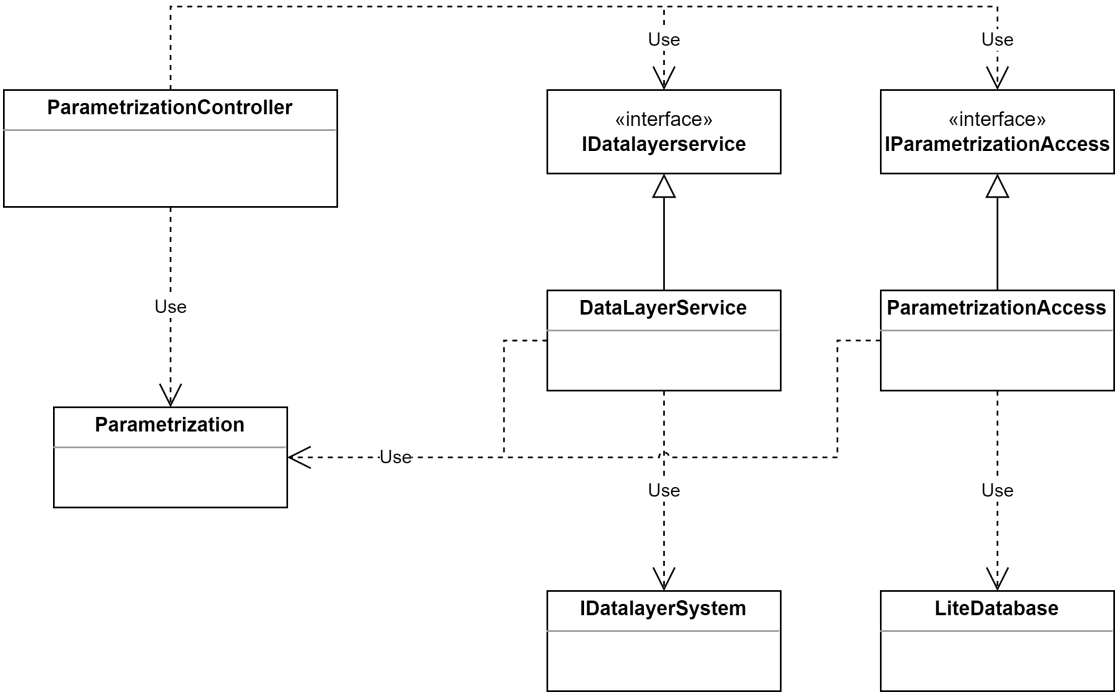


Abbildung 5.4: Klassendiagramm aus Sicht des ParametrizationController

5.3.1 Controller

Der Controller hat je eine Methode für jede Action, welche auf dem Parametrisierungs-Screen ausgeführt werden kann. Dies sind die CRUD Funktionen, hier *Create*, *Details*, *Edit* und *Delete* genannt, sowie die Action um eine Parametrisierung auszuwählen. Beim *Create* und *Edit* wird eine Parametrisierung übergeben, was per Value-Binding geschieht.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult>
    Create([Bind("Id,Name,MaxPosition,MinSpeed," +
    "MaxSpeed,MinAcceleration,MaxAcceleration," +
    "MinTorque,MaxTorque")] Parametrization parametrization)
{
     parametrization.CreateDate = DateTime.Now;
     if (ModelState.IsValid)
     {
          await Task.Run(() => _parametrizationDBAccess
              .addParametrization(parametrization));
          return RedirectToAction(nameof(Index));
     }
     return View(parametrization);
}
```

Mit dem Bind-Attribut im Parameter der Methode, kann aus den angegebenen Werten der View, automatisch ein Parametrisierungsobjekt erstellt werden.

Bei der Choose Action wird hingegen nicht eine Parametrisierung, sondern nur die Id einer Parametrisierung mitgegeben. Dafür muss die Id als Parameter hinzugefügt werden.

```
public async Task<IActionResult> Choose(int? id)
{
     if (id == null)
     {
          return NotFound();
     }

     // Choose Logik

     return RedirectToAction(nameof(Index));
}
```

Am Schluss wird eine Redirection zur Index-Action zurückgegeben, da die Choose-Action zur einfacheren Nutzbarkeit die neue Parametrisierung wählt, ohne zuerst auf eine andere View zu wechseln.

5.3.2 Model

Die Parametrization-Klasse dient als Model für den Controller. Diese enthält sowohl alle nötigen Parameter (MaximalPosition, -Geschwindigkeit usw.), als auch eine Id, ein Timestamp und ein Boolscher Wert, der definiert, ob diese Parametrisierung aktuell aktiv ist. Die Id wird für die Datenbank verwendet, was nachfolgend beschrieben ist. Auffallend ist, dass die Parametrisierung zwar eine maximale aber keine minimale Position aufweist. Das kommt daher, dass „0“ als die minimale Position einer Achse angenommen wird.

5.3.3 Datenbank

Die Parameter müssen zwingend persistent gespeichert werden, wofür eine Form von Datenbank nötig ist. Es gab die Möglichkeit einer relationalen (z.B. MySQL, SQLite) oder einer Dokumenten-basierten (MongoDB, LiteDB) Datenbank. Da die verschiedenen Parametrisierungen eigenständig sind und untereinander keine Verknüpfungen bestehen, ist eine relationale Datenbank überflüssig komplex. Die weit verbreitete MongoDB hat einen sehr grossen Feature-Umfang, muss allerdings in einer separaten Instanz gestartet werden. Aus diesem Grund wurde sich für das leichtgewichtigere Open-Source-Projekt LiteDB [15] entschieden. Diese Datenbank erstellt lokal eine Datei, in welcher die Dokumente im binären JSON-Format (BSON) gespeichert werden.
Der Zugriff auf die DB geschieht mit LINQ oder einigen Komofortmethoden, welche die API von LiteDB zur Verfügung stellt. Für die Datenbankzugriffe wurde der Service ParametrizationDBAccess erstellt.

```
1 public class ParametrizationDBAccess : IParametrizationDBAccess
2 {
3     private readonly LiteDatabase _db;
4
5     public ParametrizationDBAccess(string path =
          "Parametrizations.db")
6     {
7         _db = new LiteDatabase(path);
8     }
9 }
```

Bei der Konstruktion des ParametrizationDBAccess Service kann optional der Pfad zum gewünschten Speicherort der DB mitgegeben werden. Falls dieses File noch nicht existiert, wird es bei der Konstruktion der LiteDB erstellt. Die einzelnen Objekte werden in der LiteDB in Form von Collections gespeichert. Alle Elemente derselben Collection müssen dieselben Parameter beinhalten.

```
1 public List<Parametrization> GetParametrizations ()
2 {
```

```
3       var allParametrizations =
            _db.GetCollection<Parametrization>("parametrizations");
4       return allParametrizations.Query()
5           .ToList();
6   }
```

So werden beispielsweise alle Elemente der parametrizationsCollection ausgegeben. In der ParametrizationDBAcces Klasse ist dies auch die einzige Collection, welche verwendet wird. Auf einer solchen Collection können anschliessend entweder Query-Operationen (wie im vorherigen Code Beispiel) oder auch direkt Hilfsmethoden wie FindOne() ausgeführt werden.

```
1   public Parametrization? GetParametrization (int? id)
2   {
3       var allParametrizations =
            _db.GetCollection<Parametrization>("parametrizations");
4       var param = allParametrizations.FindOne(x => x.Id == id);
5       return param;
6   }
```

5.3.4 Datalayer

Der Datalayer ist ein Softwaresystem der CtrlX-Plattform um zwischen unterschiedlichen Applikationen zu kommunizieren. Dies geschieht über ein geteiltes Filesystem, auf welchem Werte in Form von Nodes gespeichert werden können. Diese Nodes können unterschiedliche Datentypen (Double, String, ...) haben und nur lesbar oder schreib- und lesbar sein. Für C# gibt es eine Bibliothek zur Nutzung des Datalayers [12]. Diese wird innerhalb des *DatalayerService* verwendet.

```
1   public DataLayerService(IParametrizationDBAccess
        parametrizationDBAccess, IDatalayerSystem datalayerSystem)
2   {
3       _parametrizationDBAccess = parametrizationDBAccess;
4       _datalayerSystem = datalayerSystem;
5
6       _datalayerSystem.Start(startBroker: false);
7       Console.WriteLine("ctrlX Data Layer System started.");
8
9       Client = createClient(ip, user, password);
10      Provider = createProvider(ip, user, password);
11
12      createParameterNodes();
13      createCurrentConditionNodes();
```

```
14      createLimitNodes();
15
16      Provider.Start();
17  }
```

Am Konstruktor ist erkennbar, dass der Service die beiden weiteren Services, *ParametrizationDBAccess* und *DatalayerService* benötigt, um einwandfrei zu funktionieren. Dies liegt daran, dass auf dem DatalayerService direkt Funktionalität vorhanden ist, um Parametrisierungen von der DB zu lesen und auf das Datalayer zu schreiben. Dies vereinfacht die Programmierung erheblich, führt aber aufgrund der Dependency-Injection im Konstruktor nicht zu schlechterer Testbarkeit.

Anschliessend wird ein Client und ein Provider erstellt. Diese sind nötig, um neue Nodes zu erstellen, respektive die Werte auf existierenden zu ändern. Der Benutzername und das Passwort, welche den Konstruktoren mitgegeben werden sind da, um ein TCP-Client zu erstellen, falls die Applikation nicht in einer Snap ausgeführt wird. Dies ist nur für die Testbarkeit nötig. Denn die fertige Applikation wird immer als Snap ausgeführt. Aus diesem Grund ist es auch nicht problematisch, dass das Passwort direkt als Konstante im Quellcode steht.

Der Client wird folgendermassen erstellt:

```
1   private IClient createClient(IPAddress ip, string user, string
        password)
2   {
3       // Check if the process is running inside a snap
4       var isSnapped =
            !string.IsNullOrEmpty(Environment.GetEnvironmentVariable("SNAP"));
5       Console.WriteLine($"Running inside snap: {isSnapped}");
6
7       // Create the client with inter-process communication (ipc)
            protocol if running in snap, otherwise tcp
8       var client = isSnapped
9           ? _datalayerSystem.Factory.CreateIpcClient()
10          : _datalayerSystem.Factory.CreateTcpClient(ip,
11              DatalayerSystem.DefaultClientPort,
12              user,
13              password);
14
15      Console.WriteLine("ctrlX Data Layer client created.");
16
17      return client;
18  }
```

Die Erstellung des Providers geschieht beinahe identisch und wird deshalb nicht weiter beschrieben. Nach der Erstellung des Clients und Providers werden die *ParameterNodes*, sowie die *CurrentConditionNodes* erstellt. Dies ist beim Start der Applikation nötig, da ab diesem Zeitpunkt die TIGER-IDE Daten darauf speichern können soll. Dafür wird für jeden Parameter eine Node erstellt. Diese benötigt die Parameter *address*, *type*, *description* und *value*.
Als *address* kann ein beliebiger Pfad übergeben werden. Zur Übersichtlichkeit wurde der Name des Parameters innerhalb eines Ordners für die aktive Parametrisierung verwendet. Der *type* ist eine Auswahl verschiedener Typen, welche von der CtrlX-Plattform definiert sind. Diese Typen sind allerdings über einen Pfad definiert, der auf den gewünschten Datentyp zeigt. Ein Double wird beispielsweise am besten als *types/datalayer/float64* repräsentiert. Die *description* kann bei jeder Node angezeigt werden. Der Initialwert der Node muss als IVariant übergeben werden. Die Methode zur Erstellung einer ReadWriteNode wird zum Beispiel folgendermassen aufgerufen.

```
1 CreateReadWriteNode(PARAMETRIZATION_PATH + "maxPosition",
    "types/datalayer/float64", "maximal position", new
    Variant((double)param.MaxPosition))
```

5.3.5 View

Index

In der Indexseite der Parametrization-View werden alle gespeicherten Parametrisierungen angezeigt. Die aktuell Ausgewählte ist dabei gekennzeichnet.

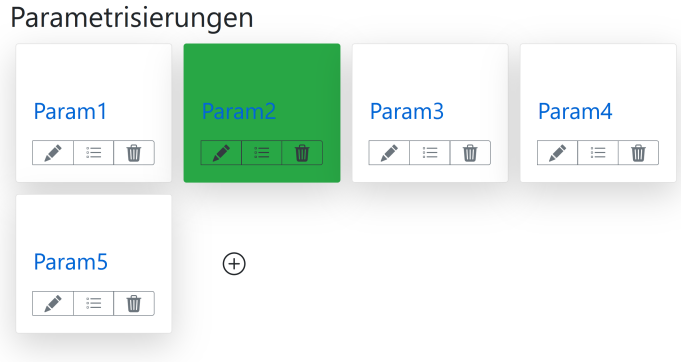


Abbildung 5.5: Index der Parametrisierungen

Das Kennzeichnen der ausgewählten Parametrisierung geschieht mit der Razor Markup Syntax. So kann bei jeder Parametrisierung im übergebenen Model der *Active* Bool überprüft werden, und falls dieser *True* ist, die Parametrisierung hervorgehoben werden.

```
@foreach (var item in Model)
{
    @if (item.Active)
    {
        <!- Darstellung der aktiven Parametrisierung -->
    }
    else
    {
        <!- Darstellung der uebrigen Parametrisierung -->
    }
}
```

Mit einem Klick auf eine Parametrisierung wird diese ausgewählt. Die drei Buttons unterhalb des Namens der Parametrisierung führen zum Edit-, Detail- oder Delete-Bildschirm.

Edit

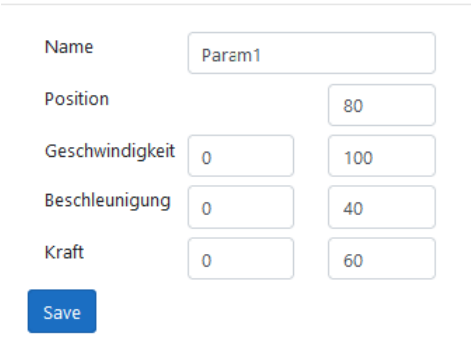


Abbildung 5.6: Edit Screen

Die einzelnen Parameter werden vom ASP.NET Framework bereits bei der Eingabe auf den korrekten Typ überprüft. Im Controller wird zusätzlich überprüft, ob die Werte innerhalb der definierten Limiten liegen. Das Bestätigen der Parametrisierung ist erst möglich, wenn alle Parameter einen gültigen Zustand haben.

```
<div class="col-sm">
    <label class="control-label">Geschwindigkeit</label>
</div>
<div class="col-sm">
    <input asp-for="MinSpeed" class="form-control"/>
    <span asp-validation-for="MinSpeed"
       class="text-danger"></span>
</div>
<div class="col-sm">
```

```
9       <input asp-for="MaxSpeed" class="form-control"/>
10      <span asp-validation-for="MaxSpeed"
            class="text-danger"></span>
11  </div>
12
13  <!- restliche Parameter -->
```

Wie beim vorherigen Code auf Zeile 6 und 10 beschrieben, erscheint eine Nachricht, falls der Nutzer versucht, einen ungültigen Wert in den Parameter zu schreiben. Um sicherzustellen, dass keine ungültigen Werte gespeichert werden, wird vor dem Speichern auch im Controller überprüft, ob das Model valid ist.

Detailansicht

Die Detailansicht ist stark am Edit-Screen angelehnt. Aus diesem Grund ist auch ein Wechsel zum Edit Screen direkt möglich.

Abbildung 5.7: Detail Screen

Löschen

Um unbeabsichtigtes Löschen einer Parametrisierung zu verhindern, muss das Löschen zuerst bestätigt werden, wobei zur Identifikation nebst dem Namen auch das Erstelldatum der Parametrisierung angezeigt wird.

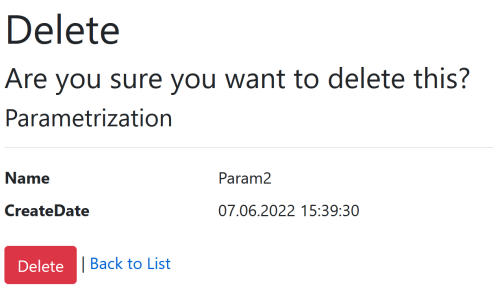


Abbildung 5.8: Delete Screen

5.4 Visuelle Programmierung

Die TIGER-IDE ist das Herzstück der visuellen Programmierung und somit ein sehr wichtiger Bestandteil der Webapp. Sie enthält die visuellen Programmierblöcke, die für das einfache Programmieren der physischen Presse eingesetzt werden. Mehr dazu in Kapitel 5.6. Für die Einbindung der TIGER-IDE werden verschiedene Schnittstellen beansprucht. Der Programming-Controller muss mit der **REST API** [16] der TIGER-IDE kommunizieren, um eine Liste der Projekte zu erhalten, die sich auf dem Gerät befinden. Diese enthält die Projekt-IDs, welche den Zugriff auf die Projekte über das **iframe** Element [17] in der Programming-View ermöglichen.

5.4.1 REST API

Die ganze Kommunikation mit der REST API der TIGER-IDE wird über den *TigerAPIService* erledigt. Um überhaupt mit der REST Schnittstelle der TIGER-IDE kommunizieren zu können, muss der *TigerAPIService* jedoch zuerst einen validen Access-Token erhalten. Für die einfache Handhabung der HTTP Anfragen verwendet der Service einen HTTP Client. Dieser wird im Konstruktor initialisiert und wie unten aufgeführt mit den notwendigen Daten beschrieben. Wichtig zu erwähnen ist, dass in Zeile 6 die Validierung der SSL Zertifikate umgangen wird. Aktuell gibt es keine andere Möglichkeit, um die REST API der ctrlX-Core zu nutzen. Dies ist ein bekanntes Problem, welches noch nicht gelöst wurde. [18]

```
1  private readonly HttpClient _httpClient;
2  private string accessToken;
3  public TigerAPIService()
4  {
5      var httpClientHandler = new HttpClientHandler();
6      httpClientHandler.ServerCertificateCustomValidationCallback =
           (message, cert, chain, errors) => { return true; };
7      _httpClient = new HttpClient(httpClientHandler);
8      _httpClient.BaseAddress = new Uri(BASE_ADDRESS);
9      _httpClient.DefaultRequestHeaders.Accept.Clear();
10     _httpClient.DefaultRequestHeaders.Accept.Add(
11         new System.Net.Http.Headers
12             .MediaTypeWithQualityHeaderValue("application/json"));
13     accessToken = "";
14 }
```

Den Access-Token erhält der *TigerAPIService*, indem er eine HTTP-POST Anfrage an die Identity-Manager API der ctrlX-Core stellt. Das wird unten in der asynchronen Methode *renewToken* erledigt. Zusammen mit der Anfrage muss im Request-Body ein

Benutzername und ein Password für ein Benutzerkonto mitgegeben werden, welches sich schon auf der ctrlX-Core befindet. Hier wird das Standard-Benutzerkonto verwendet, welches sich üblicherweise auf allen ctrlX-Core Steuerungen befindet. Der Request-Body muss als JSON formatierten String übergeben werden.

```
//...
var response = await _httpClient.PostAsync(
    IDENTITY_MANAGER_API,
    new StringContent("{\"name\": \"boschrexroth\", \"password\":
       \"boschrexroth\"}", Encoding.UTF8, "application/json"));
response.EnsureSuccessStatusCode();
var reader = new StreamReader(await
   response.Content.ReadAsStreamAsync());
var responseJson = JObject.Parse(reader.ReadToEnd());
return (string)responseJson.GetValue("access_token");
//...
```

Falls die Anfrage erfolgreich war, wird der Access-Token im Response-Body der Antwort übermittelt. Der Token muss dann aus dem String extrahiert werden. Mit der nachfolgenden *GetTigerProjects* Methode kann eine HTTP-GET Anfrage an die REST API der TIGER-IDE gestellt werden, um eine Liste der Projekte zu erhalten, die sich auf der ctrlX-Core befinden. Dafür muss zuerst sichergestellt werden, dass ein gültiger Access-Token vorhanden ist. Auch diese Methode wurde asynchron implementiert.

```
if (accessToken == "")
{
    accessToken = await renewToken();
}
var request = new HttpRequestMessage(HttpMethod.Get, TIGER_API);
request.Headers.Add("authorization", "Bearer " + accessToken);
var response = await _httpClient.SendAsync(request);
response.EnsureSuccessStatusCode();
// ...
```

Die Response-Body enthält mehr als nur die IDs und die Namen der Projekte. Aus diesem Grund werden die ID und der Name jedes Projekts extrahiert und mit Hilfe des TigerProject-Models jeweils zusammen in einem Objekt verwaltet. Die *GetTigerProjects* Methode wird allerdings erst im *ProgrammingController* aufgerufen. Mit dem Controller werden dann nur noch die Daten an die View weitergeleitet, wo sie für das Einbinden des iframe-Elements benötigt werden. Die REST API der TIGER-IDE hat momentan nur die Funktionalität, alle Projekte aufzulisten.

```
// ...
```

```
var reader = new StreamReader(await
    response.Content.ReadAsStreamAsync());
var responseJson = JObject.Parse(reader.ReadToEnd());
var projectsJArr = (JArray)responseJson["pkgs"];
var projectsList = new List<TigerProject>();
for (int i = 0; i < projectsJArr.Count; i++)
{
    JObject item = (JObject)projectsJArr[i];
    TigerProject project = new TigerProject()
    {
        Id = (string)item["path"],
        Name = (string)item["config"]["name"]
    };
    projectsList.Add(project);
}
return projectsList;
//...
```

5.4.2 iframe

Das iframe ist ein HTML Element, welches genutzt wird, um eine Webseite oder ein HTML-Dokument innerhalb einer anderen Webseite einzubinden und anzuzeigen. Mit Hilfe des iframe-Elements kann somit die Programmieroberfläche der TIGER-IDE direkt in der Programming-View der Webapp genutzt werden.

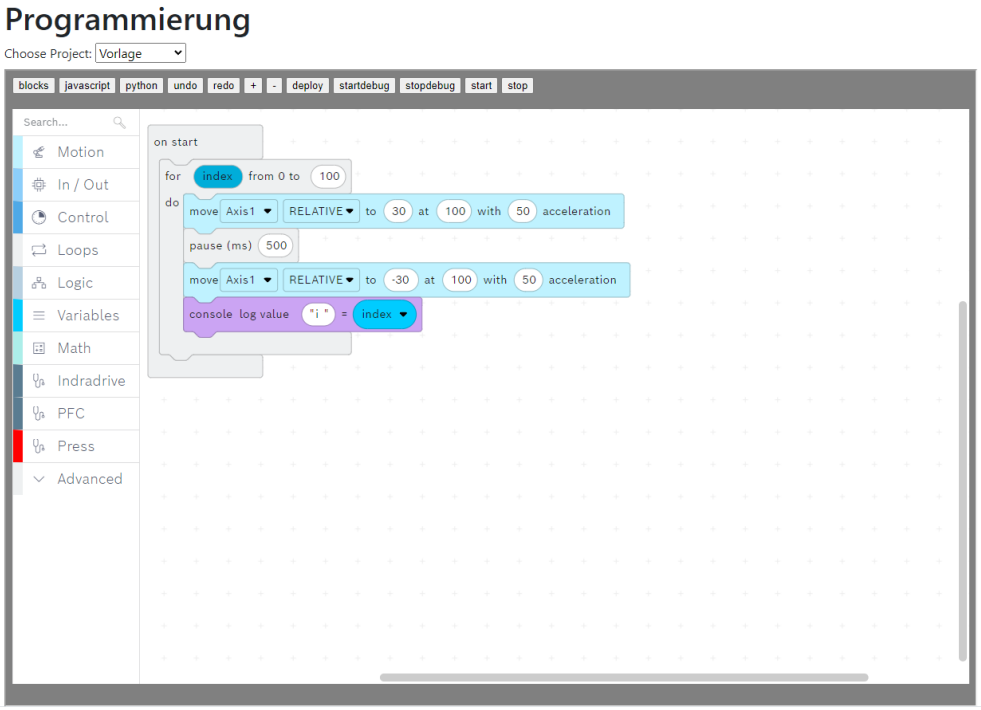


Abbildung 5.9: Programming Screen

Weil das iframe ein HTML-Element ist, wird es in der *Index.cshtml* Datei der Programming-View erstellt.

```
<iframe id="myiFrame" src="" title="Tiger Project" width="100%"
    height="800" style="display:block;"></iframe>
```

Damit die Programmieroberfläche der TIGER-IDE eingebunden werden kann, muss diese korrekt verlinkt werden. Dafür werden die weitergeleiteten Daten des Programming-Controllers so verarbeitet, dass mit der Auswahl des Projektnamens im Dropdown-Menu das entsprechende Projekt der TIGER-IDE mit der richtigen Projekt-ID verlinkt und im iframe angezeigt wird.

```
<select id="projects" onchange="changeFunc();">
    @foreach (var item in Model)
    {
        <option value="@item.Id">@item.Name</option>
    }
</select>
```

Unten auf der Zeile 8 im JavaScript Code ist zu sehen, dass die Projekt-ID des gewählten Projekts im lokalen Speicher des Browsers gesichert wird. Somit kann beim Neuladen der Seite die Auswahl im Dropdown-Menu wiederhergestellt und das iframe mit dem gleichen Projekt angezeigt werden. Zudem wird die ID des aktuellen Projekts für das iframe im Execution-View benötigt. Mehr dazu weiter unten.

```
function changeFunc() {
    const selectedValue =
        selectBox.options[selectBox.selectedIndex].value;
    iFrame.src = currentAddress + tigerIframeAddress +
        selectedValue;
    if (selectedValue === "0") {
        iFrame.src = "";
    }
    reloadIframe();
    localStorage.setItem("projectId", selectedValue);
}

if (localStorage.getItem("projectId")) {
  for (const option of optionList) {
    const projectId = option.value;
    if (projectId === localStorage.getItem("projectId") &&
        projectId !== "0") {
```

```
15          option.setAttribute("selected", "");
16          iFrame.src = currentAddress + tigerIframeAddress +
                projectId;
17          break;
18      }
19    }
20  }
```

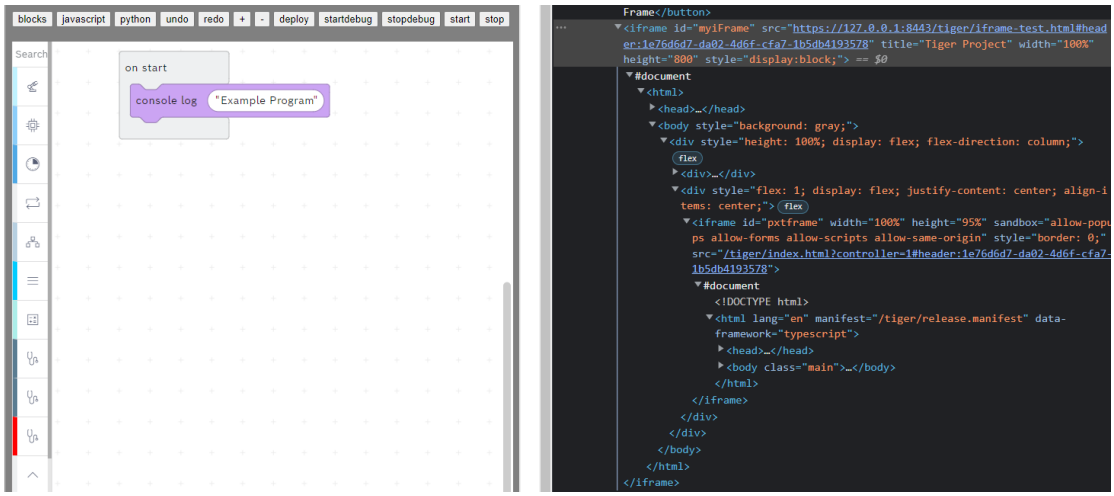


Abbildung 5.10: iframe in iframe

Das Spezielle bei der Verlinkung der TIGER-IDE ist, dass innerhalb des iframe-Elements ein weiteres, verschachteltes iframe-Element geladen wird. Das äussere iframe lädt die Interaktionsbuttons zu der TIGER-IDE zusammen mit einem div-Container für das innere iframe. Wobei das innere iframe die Programmieroberfläche der TIGER-IDE lädt. Dies wurde festgestellt als versucht wurde die Grösse des iframes anzupassen. Dementsprechend musste die Grösse des inneren iframes auch angepasst werden. Zudem wurde zum *start* Button ein zusätzlicher *EventListener* hinzugefügt. Dieser veranlasst das Laden der Execution-View.

```
1   iFrame.onload = function () {
2     try {
3       const iDocument = iFrame.contentWindow.document;
4       const pxtFrame = iDocument.getElementById("pxtframe");
5       pxtFrame.setAttribute("width", "100%");
6       pxtFrame.setAttribute("height", "95%");
7
8       const buttonCollection =
            iDocument.getElementsByTagName("button");
9       for (let item of buttonCollection) {
```

```
10          if (item.innerText === "start") {
11            item.addEventListener("click", function () {
                  window.top.location.href = currentAddress +
                  "/pressesteuerung/Execution"; });
12          }
13        }
14      } catch (err) {
15        console.log(err.message);
16      }
17  }
```

5.5 Durchführung

5.5.1 Programmablauf

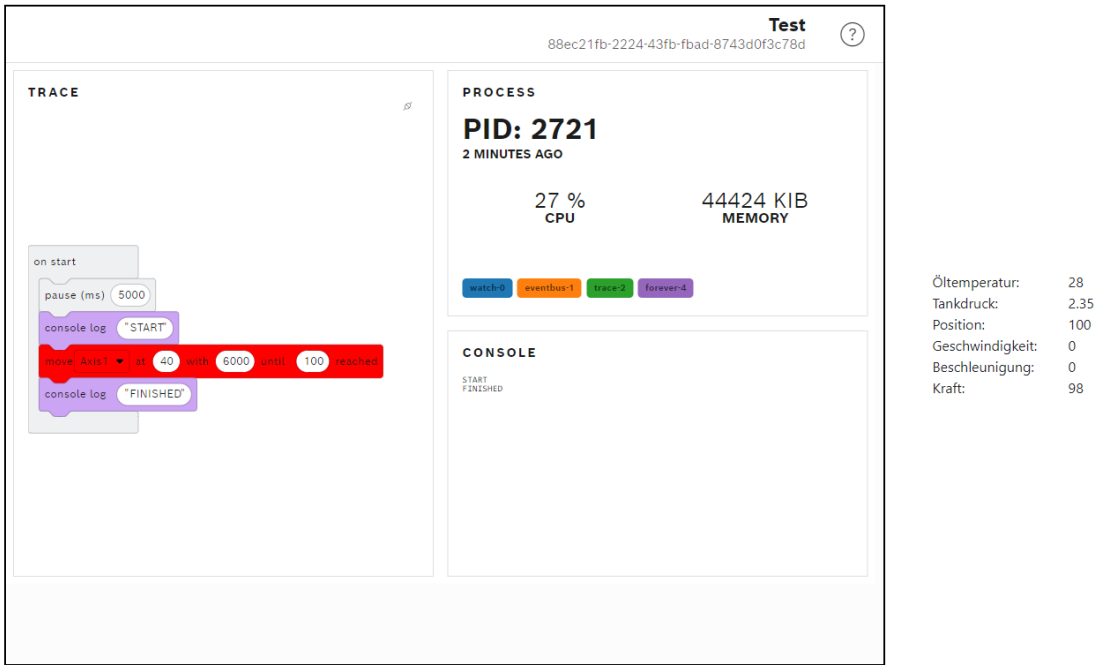


Abbildung 5.11: Execution Screen

In der Execution-View wird wiederum ein iframe benutzt, um den Programmablauf des gewählten Projekts anzuzeigen. Um das iframe korrekt zu verlinken kann die lokal gespeicherte Projekt-ID vom vorherigen Screen wiederverwendet werden. Somit ist es nicht nötig, im Execution-Controller den *TigerAPIService* zu nutzen, um die Liste der Projekte per REST Interface anzufragen. Das iframe hier wurde im Vergleich zum Programming-View leicht verändert. Auf diesem Screen können keine Projekte bearbeitet werden. Die

ursprünglichen Buttons innerhalb des iframes wurden auch entfernt. Dieser Screen soll nur als Anzeige dienen und bietet somit keine Interaktionsmöglichkeiten an.

5.5.2 Zustandsanzeige

Zusätzlich zum Programmablauf sollten auch die Werte: *Öltemperatur, Tankdruck, Position, Geschwindigkeit, Beschleunigung und Kraft* der Presse auf diesem Screen angezeigt werden. Dies dient der Zustandskontrolle der Presse. Diese aktuellen Werte können über die TIGER-IDE gelesen und direkt ins Datalayer geschrieben werden. Aber aufgrund eines bekannten Fehlers der TIGER-IDE, funktioniert dies aktuell nicht. Denn solange die Verbindung zur Presse besteht, kann die Datalayer-Bibliothek der TIGER-IDE nicht genutzt werden. Somit können die aktuellen Werte dem Datalayer nicht mitgeteilt werden. Trotzdem werden im Execution-Controller mit Hilfe des *DatalayerServices* die nötigen Nodes auf dem Datalayer erstellt. Diese werden direkt im *ExecutionController* in Sekundentakt ausgelesen und auf dem Screen aktualisiert.

Das Auslesen der Werte vom Datalayer geschieht innerhalb des ExecutionControllers mittels asynchroner Schleife. In dieser Schleife werden die Werte mit dem signalR-Framework an einen Hub gesendet, welcher die View notifiziert, die neuen Werte darzustellen. Dies ist der von Microsoft empfohlene Ansatz, um Echtzeitdaten an die View zu übermitteln. [19]

```
1  public IActionResult Index()
2  {
3      Task.Run(async () => {
4          while (!currentStateReadingCancellationToken
5              .IsCancellationRequested)
6          {
7              currentState =
                   _dataLayerService.readCurrentStateNodes();
8              await
                   _hubContext.Clients.All.SendAsync("ReceiveMessage",
9                    currentState.OilTemperature.ToString(),
10                   currentState.OilPressure.ToString(),
11                   currentState.Position.ToString(),
12                   currentState.Speed.ToString(),
13                   currentState.Acceleration.ToString(),
14                   currentState.Force.ToString());
15             await Task.Delay(CURRENT_STATE_READ_DELAY_MS,
                   currentStateReadingCancellationToken);
16         }
17     }, currentStateReadingCancellationToken);
18     return View(currentState);
19 }
```

In der View wird sowohl ein signalR-Script, als auch ein eigenes JS-Script eingebunden, welches die neuen Daten des Hubs ohne Reload der Seite anzeigt.

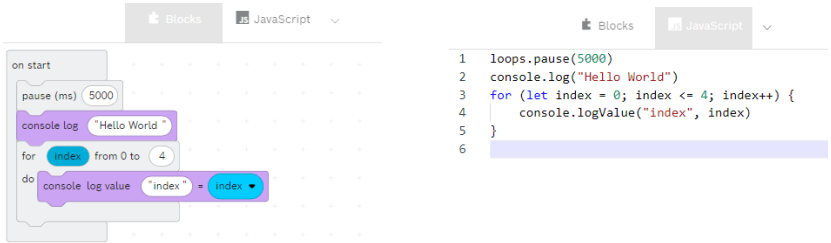
```
var connection = new signalR.HubConnectionBuilder()
    .withUrl("/pressesteuerung/current_state_hub").build();

connection.on("ReceiveMessage", function (oilTemperature,
    oilPressure, position, speed, acceleration, force) {
    document.getElementById("oilTemperature").textContent =
        oilTemperature;
    // Text bei weiteren Elementen einfuegen
});

connection.start().then(function () {
    console.log("client connection to signalR hub started.")
});
```

5.6 TIGER-IDE

Die IDE App [4] unterscheidet zwischen den beiden Entwicklungsumgebungen Visual Coding und Textual Coding. Der Visual Coding Teil der IDE App wird als TIGER-IDE (Target Independent Graphical Experience fRamework - Integrated Development Environment) bezeichnet. Dieser Teil der App ist zwar weniger umfangreich, setzt aber keine Programmierkenntnisse voraus. Der Textual Coding Teil ist die vollumfängliche Entwicklungsumgebung mit weniger Restriktionen. Weil der Textual Coding Teil der IDE App nicht relevant für diese Webapp ist, wird im Folgenden nur noch Bezug auf die TIGER-IDE genommen. Es kann auch innerhalb der TIGER-IDE mit JavaScript oder Python programmiert werden. Im Unterschied zum Textual Coding Teil der IDE App, ist das Programmieren mit JavaScript und Python innerhalb der TIGER-IDE sehr viel eingeschränkter.



(a) Blöcke (b) JavaScript

Abbildung 5.12: TIGER-IDE Programmbeispiel

5.6.1 Konfiguration einer hydraulischen Presse

Die TIGER-IDE enthält bereits die meisten Standardkonstrukte wie if-else Statements und while Loops. Zudem bietet sie aber auch schon komplexere Programmierblöcke, wie das Bedienen einer virtuellen oder physischen Achse an. Diese Blöcke funktionieren zwar auch für hydraulische Achsen, allerdings kann die TIGER-IDE standardmässig nicht mit einer hydraulischen Presse kommunizieren und somit auch nicht deren Achse bedienen. Aus diesem Grund hat das Entwicklungsteam der TIGER-IDE eine Beta-Version der App mit drei neuen Extensions zur Verfügung gestellt. Damit soll eine hydraulische Presse konfiguriert und bedient werden können.

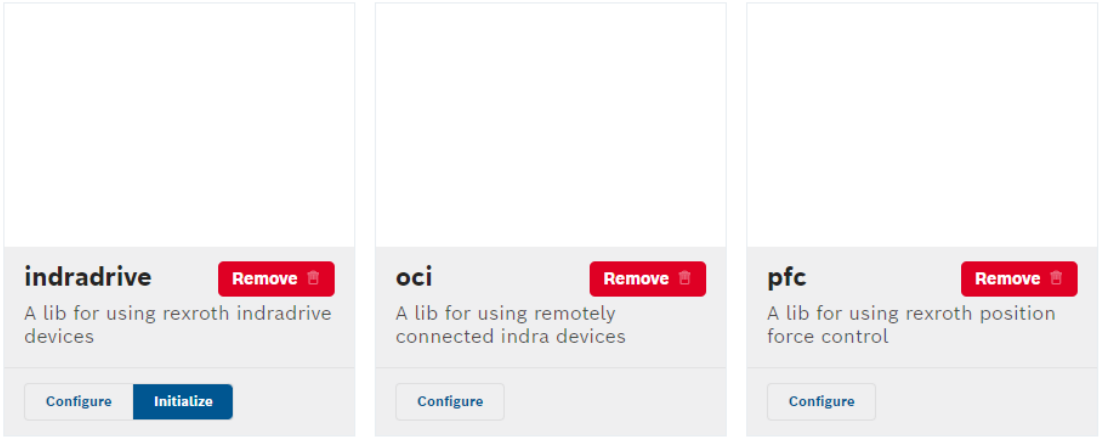


Abbildung 5.13: TIGER-IDE Extenstions

Im Folgenden wird gezeigt, wie eine Konfiguration erstellt wird. Dank des iframe-Elements in der Programming-View der Webapp, können die Schritte der Konfiguration genau gleich auch dort ausgeführt werden. Es ist jedoch aktuell nicht möglich, ein neues TIGER-IDE Projekt mit dem iframe zu erstellen. Deswegen muss dieser Schritt noch innerhalb der TIGER-IDE selbst durchgeführt werden. Da in diesem Fall mit dem ctrlX Virtual-Core gearbeitet wird, ist es wichtig, dass im nachfolgenden Screen namens *Boards*, die Motion Simulation Option gewählt wird. Die Option *ctrlx* wird für eine reale ctrlX-Core und die Option *iec* für alle anderen Steuerungen verwendet. An dieser Stelle sollte auch sichergestellt werden, dass die Presse korrekt mit einem Ethernet Kabel an dem Rechner angeschlossen ist, auf welchem die VirtualCore läuft.

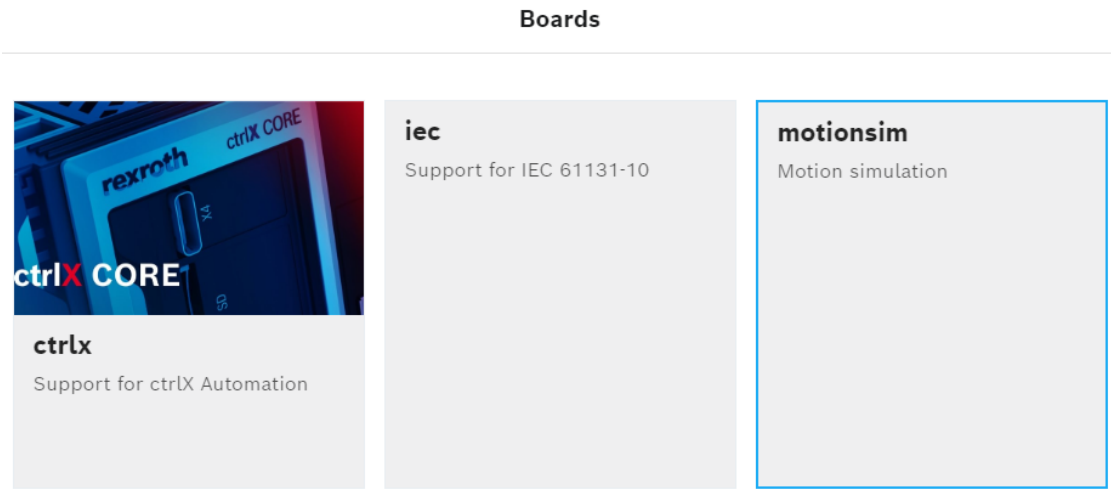


Abbildung 5.14: TIGER-IDE Boards

Extension: oci

Mit der *oci* Extension kann die TIGER-IDE eine Verbindungskonfiguration erstellen, welches im nächsten Schritt benötigt wird. Dafür muss ein neuer Eintrag mit einer eindeutigen Id, der IPv4-Adresse der Presse, sowie dem Typ *EAL* erstellt werden.

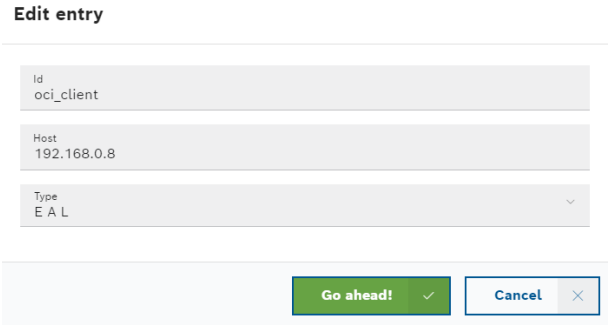


Abbildung 5.15: oci Extension Eintrag

Extension: indradrive

Zusammen mit der Verbindungskonfiguration von vorher kann die *indradrive* Extension anschliessend eine Konfiguration für den Antrieb der hydraulischen Presse erstellen. Auch hier muss dafür ein neuer Eintrag erstellt werden. In der nachfolgenden Abbildung wird der Eintrag mit den darauf ersichtlichen Standardwerten für die unteren beiden Felder erstellt. Das Feld *Standstill Window* gibt die Schwelle für die Stillstanderkennung der Presse in [mm/s] an. Mit dem Feld *Pfc Relative Switching Window* „wird ein prozentuales Fenster um den Kraft-Sollwert eingestellt, in dem sich der Kraft-Istwert befinden muss, damit die ablösende Regelung von Lageregelung in Kraftregelung

überführt“. [20, S. 240] Nach dem Speichern des Eintrags wird nach einer Initialisierung des Projekts gefragt, was zu diesem Zeitpunkt noch nicht nötig ist. Das Pop-up Fenster kann deswegen geschlossen werden.

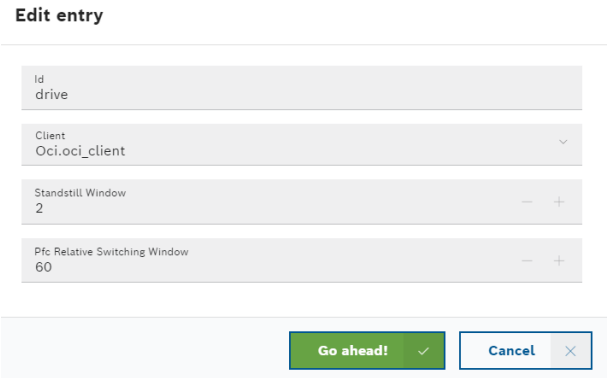


Abbildung 5.16: indradrive Extension Eintrag

Extension: pfc

Bei der *pfc* Extension muss nichts konfiguriert werden, da diese lediglich einen Programmierblock zur Verfügung stellt. Dieser Block ermöglicht die Bewegung einer hydraulischen Achse mit der Nutzung der Position Force Controll Software im Speicher des Antriebs. Bevor dieser Block benutzt werden kann, muss eine Achse konfiguriert und der Presse hinzugefügt werden. Mit der *motion* Extension kann folgendermassen eine neue Achse konfiguriert werden.

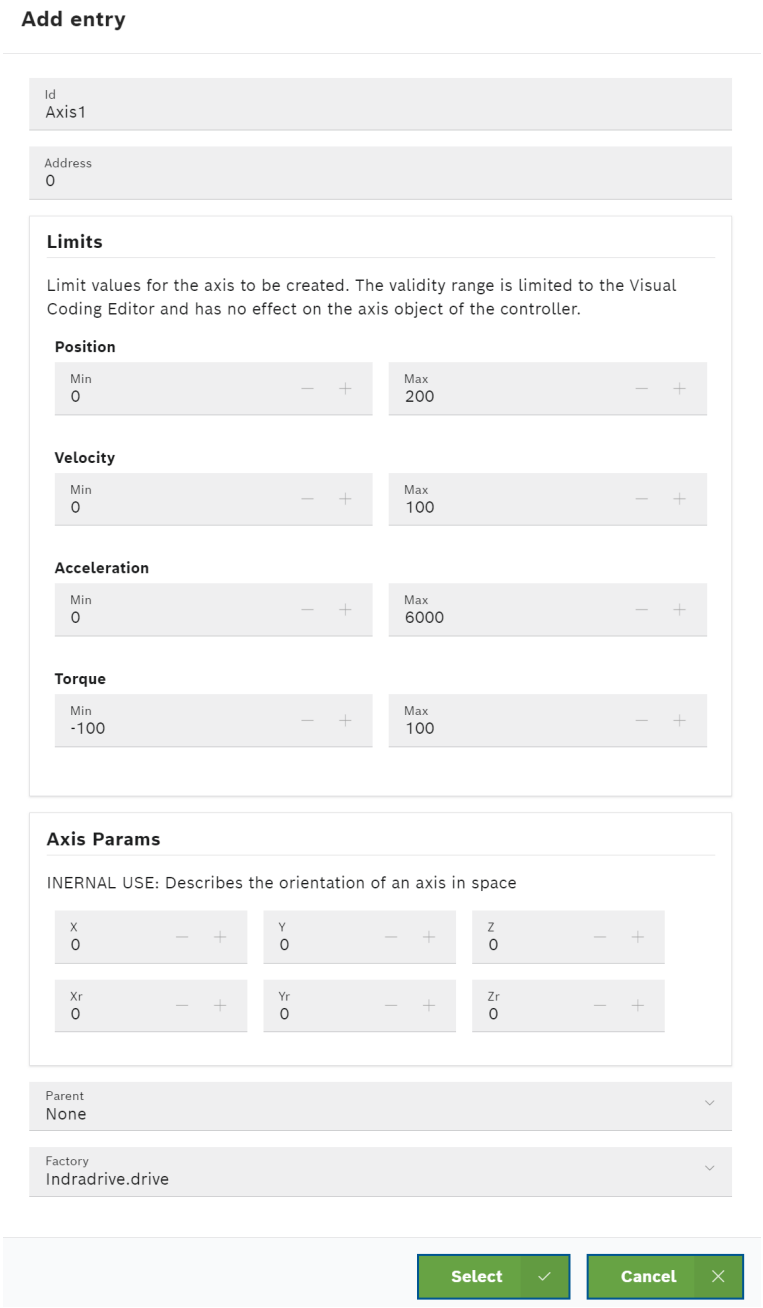


Abbildung 5.17: TIGER-IDE Achsen Konfiguration

Das Adressenfeld wird üblicherweise mit „0“ beschrieben, falls die Presse nur eine Achse hat. Die Grenzwerte, die hier definiert werden, haben keinen Einfluss auf die tatsächlichen Grenzwerte der Achse. Diese dienen lediglich zur Eingrenzung der Werte die im Programmierblock eingeben werden können. Die maximale Beschleunigung muss auf 6000 mm/s^2 gesetzt werden, da sich die Achse bei einer niedrigeren Beschleunigung nicht be-

wegt. Dies ist ein Fehler der TIGER-IDE und hat nichts mit der Achse zu tun. Zum Schluss muss noch im Feld Factory die vorher konfigurierte indradrive-Extension ausgewählt werden. Erst jetzt, nachdem die Achsenkonfiguration gespeichert wurde, sollte im Pop-up Fenster, die Initialisierungsoption *Run on Target* ausgewählt werden.

Abbildung 5.18: PFC Programmierblock

Nun sollte es möglich sein, mit dem PFC Programmierblock die Achse zu bewegen. Dazu müssen die Werte für Geschwindigkeit, Beschleunigung und Kraft eingegeben werden. Allerdings hat die TIGER-IDE noch Probleme sich mit der hydraulischen Presse zu verbinden. Mehr dazu in Kapitel 6.2. Dieser Block dient als Beispiel, wie eigene Programmierblöcke kreiert werden können. Der Block wurde vom Entwicklungsteam zur Verfügung gestellt.

5.6.2 Neue Programmierblöcke

Wie schon oben erwähnt, bietet die TIGER-IDE die Möglichkeit, eigene Programmierblöcke mit JavaScript zu erstellen. Die erstellten Blöcke befinden sich dann aber nur in dem Projekt, in welchem sie erstellt wurden und können nicht automatisch mit anderen Projekten geteilt werden. Nur das Entwicklungsteam der TIGER-IDE kann Programmierblöcke erstellen, die zumindest als Extension in jedem Projekt angezeigt werden. Ansonsten muss der Code, mit welchem die Programmierblöcke erstellt wurde, manuell in jedes Projekt kopiert werden.

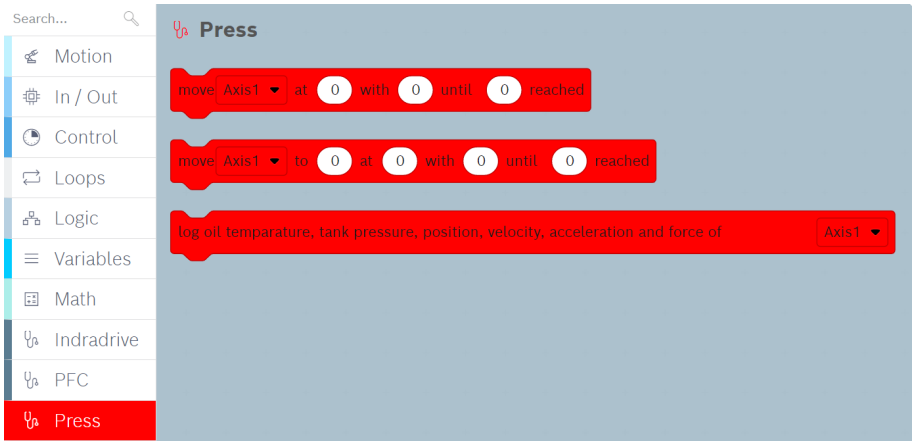


Abbildung 5.19: PFC Programmierblock

Die neuen Programmierblöcke werden im Scope *Press* angezeigt und können wie die anderen verwendet werden. Es ist aber zu beachten, dass diese Blöcke nur bei hydraulischen Achsen mit PFC benutzt werden sollten. Bei anderen Achsen könnten Parameter fehlen, die intern in den Programmierblöcken eingelesen werden. Wie unten ersichtlich, können Funktionen im gleichen Namespace als Programmierblöcke im gleichen Scope definiert werden.

```
/**
 * Hydraulic Press Commands and Helper Functions
 */
//% icon="\ue268" block="Press" color=#FF0000
namespace press {
    /**
     * Moves axis to the specified position.
     * @param axis the axis to be moved
     * @param position the target position
     * @param velocity the target velocity
     * @param acceleration the target acceleration
     * @param force the maximum force to be reaced
     */
    //% block="move $axis to $position at $velocity with
       $acceleration until $force reached"
    //% axis.fieldEditor="configInstance"
    //% axis.fieldOptions.pkg="motion"
    //% axis.fieldOptions.property="axes"
    //% position.shadow="axisValuePicker"
    //% velocity.shadow="axisVelocityPicker"
    //% acceleration.shadow="axisAccelerationPicker"
    //% inlineInputMode=inline
    //% promise
    export function moveToPosition(axis: oci.Axis, position:
       number, velocity: number, acceleration: number, force:
       number = 10000): void {
        // ...
    }
//...
}
```

Mit Annotationen oberhalb der jeweiligen Funktion kann das Aussehen sowie die Parameteroptionen der Programmierblöcke gestaltet werden.

Achse zur Zielposition bewegen

Im Unterschied zur vorhandenen *move* Bewegung des Motion Scopes wird hier die Achse nur solange in Richtung der Position bewegt, bis die festgelegte Kraft erreicht wird. Falls die Zielkraft schon vor der Zielposition erreicht wird, stoppt die Funktion die Bewegung

dank des PFC. Mehr dazu weiter unten.

```
1         export function moveToPosition(axis: oci.Axis, position:
              number, velocity: number, acceleration: number, force:
              number = 10000): void {
2             axis.client.writeParameter('S-0-0553.0.0', 20000); //
                  setpoint ramp
3
4             const maxPosition =
                  axis.client.readParameter('S-0-0278.0.0') - 0.1;
5             const currentPosition =
                  axis.client.readParameter('S-0-0386.0.0');
6             const inputPosition = position < 0 ? 0 : position; //
                  no negative value
7             const targetPosition = maxPosition < position ?
                  maxPosition : position;
8
9             const maxForce =
                  axis.client.readParameter('S-0-0082.0.0');
10            const targetForce = maxForce < force ? maxForce :
                  force;
11            axis.client.writeParameter('S-0-0550.0.0',
                  targetForce); // set target force
12        // ...
```

Die benötigten Daten werden hier direkt von der Presse abgelesen. Mit dem Tool *IndraWorks Ds* konnten die jeweiligen Parameter-IDs herausgefunden werden. In Zeile 2 wird die Rampensteigung für die Zielkraft festgelegt. Diese bewirkt, dass die Zielkraft in einer stetigen Bewegung erreicht wird und nicht sprunghaft. Danach werden einige Grenzwerte und die aktuelle Position der Achse ausgelesen. Falls die vom Benutzer eingegebene Zielkraft die Maximalkraft der Presse übersteigt, wird die Zielkraft dementsprechend geändert.

```
1         //...
2         const relTargetPosition = inputPosition < currentPosition
              ? (inputPosition - currentPosition) : inputPosition;
3
4         axis.client.writeParameter('P-0-1382.0.0', 1); // switch
              PFC on
5         axis.move(MoveType.RELATIVE, relTargetPosition, velocity,
              acceleration);
6
7         while (Math.abs(axis.client.readParameter('S-0-0552.0.0')
              - targetForce) > 10 ) {
8             loops.pause(100);
9         }
```

```
10          loops.pause(100);
11
12          axis.stop();
13          axis.client.writeParameter('S-0-0550.0.0', 0);
14          loops.pause(100);
15          axis.client.writeParameter('P-0-1382.0.0', 0); // switch
                PFC off
16      }
```

In Zeile 1 wird die absolute Zielposition in die relative Zielposition umgerechnet, da die absolute Variante der vorgegebenen move-Funktion unerklärliche Fehler ausgibt. Das Entwicklungsteam der TIGER-IDE konnte das Problem nicht im Zeitraum der Bachelorarbeit lösen. Bevor die Achse bewegt werden kann, muss noch die PFC eingeschaltet werden. Diese ist für das parallele Kontrollieren der Achsbewegung in Bezug auf Position und Kraft zuständig. Je nachdem welcher Ziel- oder Grenzwert zuerst erreicht werden würde, stoppt das PFC die Achsbewegung mit einem Puffer. Mit der while-Schleife wartet das Programm bis die Bewegung vollständig durchgeführt wurde oder die maximale Zielkraft erreicht wurde. Zum Schluss wird die Zielkraft zurück auf 0 gesetzt und das PFC ausgeschaltet.

Achse ausfahren bis Zielkraft erreicht wird

```
1           //...
2           axis.move(MoveType.RELATIVE, maxPosition, velocity,
                acceleration);
3
4           while (Math.abs(axis.client.readParameter('S-0-0552.0.0')
                - targetForce) > 10) {
5               loops.pause(100);
6           }
7           //...
```

Diese Funktion ist fast identisch zur Vorherigen. Der Unterschied ist lediglich, dass die Zielposition nicht vom Benutzer gewählt werden kann, da die Achse so lange ausfahren soll, bis die Zielkraft erreicht wird. Wenn die Zielkraft nie erreicht wird, wird die Achse ganz ausgefahren.

Aktuelle Zustandswerte der Achse ausgeben

```
1       export function logOilTemparatureAndTankPressure(axis:
            oci.Axis): void {
2           const oilTemp = axis.client.readParameter('P-0-1284.0.0');
```

```
3           const tankPress =
                axis.client.readParameter('S-0-0814.0.0');
4           const position =
                axis.client.readParameter("S-0-0386.0.0");
5           const velocity =
                axis.client.readParameter("S-0-0040.0.0");
6           const acceleration = axis.client.readParameter("");
7           const force = axis.client.readParameter("S-0-0552.0.0");
8           console.log("Oil Temparature: " + tools.toString(oilTemp)
                + " C" );
9           console.log("Position: " + tools.toString(position) + "
                mm");
10          console.log("Velocity: " + tools.toString(velocity) + "
                mm/s" );
11          console.log("Acceleration: " +
                tools.toString(acceleration) + " mm/s^2");
12          console.log("Force: " + tools.toString(force) + " N" );
13      }
```

Mit dieser Funktion werden die aktuellen Zustandswerte der Achse auf der Konsole ausgegeben.

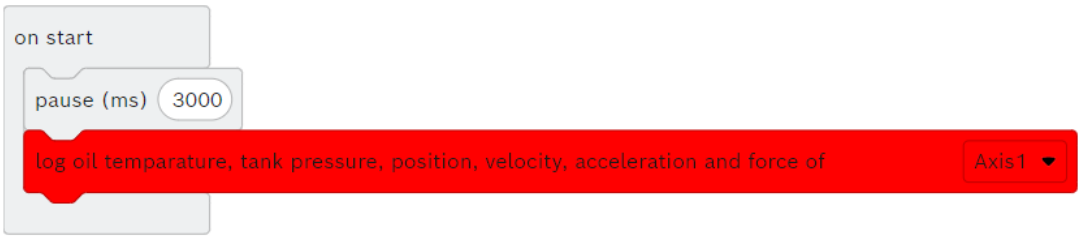


Abbildung 5.20: Neuer Programmierblock Ausführung

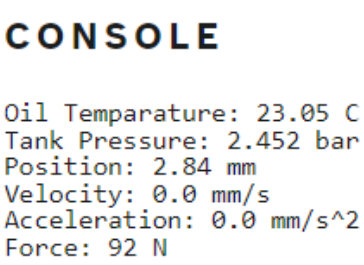


Abbildung 5.21: Neuer Programmierblock Auswertung

Weitere Programmierblöcke

Es wurde von Josef Müller bestätigt, dass diese Blöcke als Vorlage genügen. Wenn weitere Blöcke benötigt werden, können diese leicht aus den vorhandenen Blöcken erstellt werden.

Kapitel 6

Validierung

6.1 Qualitätsmassnahmen

6.1.1 Dokumentation

Die Dokumentation der Arbeit wird mit LaTeX in einem GitLab Repository erstellt.

6.1.2 Entwicklung

Der Source Code der Arbeit wird ebenfalls in einem GitLab Repository verwaltet. Dabei werden Feature Branches und Merge Requests verwendet. Zudem wird GitLab CI als Continuous Integration Tool verwendet.

6.1.3 Code Style Guidelines

Als Code Style Guidelines werden die C# Coding Conventions [8] angewendet.

6.1.4 Code Reviews

Code Reviews werden von beiden Studierenden durchgeführt. Bei einem Merge Request überprüfen die Studierenden gegenseitig den Code, bevor dieser auf den Master Branch gemergt werden kann.

6.1.5 Unit Tests

xUnit

Vom .NET Standard wird empfohlen, die Unit Tests mit dem xUnit-Framework zu schreiben. [21] Dafür wird am einfachsten ein eigenes Projekt mit dem xUnit Template erstellt. Dieses kann der bestehenden Solution hinzugefügt werden. Das Testprojekt benötigt eine Verlinkung auf das Pressesteuerungsprojekt. Ein beispielhafter Unit Test sieht folgendermassen aus:

```
public class UnitTests
{
    [Fact]
    public void TestFoo()
    {
        var result = foo();
        Assert.Equal(result, 42);
    }
}
```

Es können beliebig viele Unit Tests als Funktionen mit dem *Fact* Attribut erstellt werden. Dies definiert, dass die Methode vom Test Runner ausgeführt werden soll. Mit dem Befehl *dotnet test* führt der Test Runner alle Tests aus und schreibt das Resultat auf die Konsole.
Das xUnit Framework unterstützt auch den Test von asynchroner Logik. Dafür muss lediglich die Testmethode als *async* definiert werden. Innerhalb der Methode können asynchrone Funktionen dann problemlos mit await aufgerufen werden.

Moq

Als Mocking Framework wird moq [22] empfohlen. [21] Dies ist ein sehr mächtiges und intuitiv zu bedienendes Framework für das Erstellen von Mocks, wie sie bei Unit Tests benötigt werden. Da im Pressesteuerungsprojekt die Controller ihre benötigten Services mit Constructor Injection erhalten, können an diesen Stellen sehr gut Mocks der Services eingefügt werden. Dies sieht zum Beispiel so aus:

```
[Fact]
public void Index_ReturnsViewWithAllParametrizations()
{
    var mockLogger = new
       Mock<ILogger<ParametrizationController>>();
    var mockDataLayerService = new Mock<IDataLayerService>();
    var mockParametrizationDBAccess = new
       Mock<IParametrizationDBAccess>();
    var testList = new List<Parametrization>()
    {
        new Parametrization(){Id=0},
        new Parametrization(){Id=1},
        new Parametrization(){Id=2}
    };
    mockParametrizationDBAccess.Setup(pa =>
       pa.GetParametrizations())
        .Returns(testList);

    var paramController = new
       ParametrizationController(mockLogger.Object,
```

```
17              mockDataLayerService.Object,
18              mockParametrizationDBAccess.Object);
19
20      var result = paramController.Index();
21
22      var viewResult = Assert.IsType<ViewResult>(result);
23      var model =
            Assert.IsAssignableFrom<IEnumerable<Parametrization>>
24          (viewResult.ViewData.Model);
25      Assert.Equal(testList, model);
26  }
```

In den Zeilen 4-6 werden die Mock-Objekte basierend auf den mitgegebenen Interfaces erstellt. Bei 7-14 wird die gewünschte Logik gemockt. Dies geschieht indem definiert wird, was beim Aufruf gewisser Methoden auf dem Mock-Objekt zurückgegeben wird. Anschliessend wir der Controller mit den Mock-Objekten konstruiert und die Index() Methode ausgeführt. Die statischen IsType() und IsAssignableFrom() Methoden auf Zeile 22 und 23 dienen nebst dem Test dazu, das erhaltene Objekt zum gewünschten Typen zu casten, sodass es auf der Zeile 24 mit dem ursprünglich übergebenen Objekt verglichen werden kann.
Auf den Mocks kann ebenfalls überprüft werden, wie oft und mit welchen Parametern eine Methode aufgerufen wurde. Dafür wird im Setup eine leere Liste der Parameter mitgegeben, welche während der Ausführung des Tests von Moq befüllt wird.

```
1  var args = new List<Parametrization>();
2  mockParametrizationDBAccess.Setup(pa =>
       pa.editParametrization(Capture.In(args)));
```

Anschliessend kann überprüft werden, ob die Methode richtig oft und mit den korrekten Parametern ausgeführt wurde.

```
1  mockParametrizationDBAccess.Verify(pa =>
       pa.editParametrization(), Times.AtLeastOnce());
2  Assert.Equal(args[0], oldParams);
3  Assert.Equal(args[1], newParams);
```

Ergebnis

Da das Schreiben von Unit Tests mit mehreren Mock Objekten recht aufwändig ist, wurde darauf verzichtet, dies mit allen Klassen zu machen. Stattdessen wurden nur jene getestet, welche grössere Mengen an eigener Funktionalität aufweisen. Ebenfalls weggelassen wurden die Klassen: *MetadataBuilder, MetadataProvider, Node und Read-*

WriteNodeHandler da diese direkt von Samples der ctrlX-SDK kopiert wurden. Getestet wurden schliesslich folgende Klassen:

- ParametrizationController
- DatalayerService

Beide Klassen nutzen nicht nur APIs sondern verarbeiten die Daten auch.

Name	Zeilen total	Zeilen getestet	Coverage
ParametrizationController	128	110	85.9%
DataLayerService	160	136	85%

6.1.6 Manuelle Integrationstest

Zur Bestimmung der Testfälle wurde das Komponentendiagramm im Kapitel 5 als Hilfestellung genommen. Dabei soll die korrekte Zusammenarbeit der Komponenten überprüft werden. Die im jeweiligen Testfall involvierten Komponenten sind im Titel des Testfalls aufgelistet. Zudem wird hier die Verbindung zu einer hydraulischen Achse getestet.

Testfall 1: Parametrisierung - Datalayer

- **Ziel:** Prüfen, dass die Parametrisierungskomponente imstande ist dem Datalayer Daten zu übermitteln.
- **Vorgehen:** Eine neue Parametrisierung erstellen und auswählen.
- **Erwartetes Ergebnis:** Die Werte der erstellten Parametrisierung werden auf die Datalayer-Nodes im Parametrization Ordner übertragen.
- **Abweichungen vom erwarteten Ergebnis:** Keine.

Testfall 2: Parametrisierung - Datalayer

- **Ziel:** Prüfen, dass die Parametrisierungskomponente imstande ist Daten vom Datalayer zu lesen.
- **Vorgehen:** Die Grenzwerte in den Datalayer-Nodes im Limits Ordner definieren und danach eine Parametrisierung ausserhalb der Grenzwerte erstellen.
- **Erwartetes Ergebnis:** Die Parametrisierung kann mit den Werten ausserhalb der Grenzwerte nicht gespeichert werden.
- **Abweichungen vom erwarteten Ergebnis:** Keine.

Testfall 3: Parametrisierung - ParametrizationDB

- **Ziel:** Prüfen, dass die Parametrisierungskomponente imstande ist Daten auf der ParametrizationDB persistent zu speichern.
- **Vorgehen:** Eine neue Parametrisierung erstellen, diese speichern, die Webapp schliessen, die ctrlX-Core neu starten und die Webapp wieder öffnen.
- **Erwartetes Ergebnis:** Die Werte der erstellten Parametrisierung sind noch in der Webapp vorhanden.
- **Abweichungen vom erwarteten Ergebnis:** Keine.

Testfall 4: Programmierung - TIGER-IDE

- **Ziel:** Prüfen, dass die Programmierungskomponente erfolgreich mit der REST API der TIGER-IDE kommunizieren kann.
- **Vorgehen:** Ein neues Projekt in der TIGER-IDE erstellen und danach die Webapp öffnen und auf die Programmierungsseite wechseln.
- **Erwartetes Ergebnis:** Im Dropdown-Menu der Programmierungsseite wird das neu erstellte TIGER-IDE Projekt mit dem richtigen Namen aufgelistet.
- **Abweichungen vom erwarteten Ergebnis:** Keine.

Testfall 5: Programmierung - TIGER-IDE

- **Ziel:** Prüfen, dass die TIGER-IDE erfolgreich mit dem iframe-Element der Programmierungskomponente verlinkt werden kann.
- **Vorgehen:** Ein vorhandenes Projekt im Dropdown-Menu der Programmierungsseite auswählen.
- **Erwartetes Ergebnis:** Das entsprechende TIGER-IDE Projekt wird korrekt im iframe angezeigt und kann auch bearbeitet werden.
- **Abweichungen vom erwarteten Ergebnis:** Keine.

Testfall 6: Programmierung - Durchführung

- **Ziel:** Prüfen, dass die Programmierungskomponente und die Durchführungskomponente jeweils das gleiche Projekt im iframe-Element verlinken.
- **Vorgehen:** Ein vorhandenes Projekt im Dropdown-Menu der Programmierungsseite auswählen und danach den *start* Button im iframe der Programmierungsseite drücken.

- **Erwartetes Ergebnis:** Das entsprechende Programm wird gestartet und dessen Ablauf auf der Durchführungsseite angezeigt.
- **Abweichungen vom erwarteten Ergebnis:** Falls das Programm Fehler aufweist, wird die Webapp trotzdem auf die Durchführungsseite wechseln. Aber anstelle des Programmablaufs, wird wieder die Programmieroberfläche angezeigt. Wichtig zu erwähnen ist, dass mögliche Compile Error der TIGER-IDE Projekte nicht im iframe angezeigt werden.

Testfall 7: Datalayer - Durchführung

- **Ziel:** Prüfen, dass die Durchführungskomponente imstande ist Daten vom Datalayer zu lesen.
- **Vorgehen:** Das Datalayer öffnen und neue Werte für die Nodes im Condition Ordner schreiben.
- **Erwartetes Ergebnis:** Die festgelegten Werte werden auf der Durchführungsseite korrekt angezeigt.
- **Abweichungen vom erwarteten Ergebnis:** Keine.

Testfall 8: Datalayer - TIGER-IDE

- **Ziel:** Prüfen, dass die TIGER-IDE imstande ist der Datalayer-Komponente Daten zu übermitteln.
- **Vorgehen:** In der TIGER-IDE ein neues Projekt erstellen, die Datalayer-Extension benutzen und neue Werte auf Nodes in Ordner Condition schreiben.
- **Erwartetes Ergebnis:** Die geschriebenen Werte werden korrekt auf den Datalayer-Nodes übernommen.
- **Abweichungen vom erwarteten Ergebnis:** Die Datalayer-Extension von der TIGER-IDE schreibt die Werte nur dann erfolgreich, wenn bei der Erstellung des Projekts, das Board *ctrlx* ausgewählt wurde.

Testfall 9: Datalayer - TIGER-IDE

- **Ziel:** Prüfen, dass die TIGER-IDE imstande ist Daten vom Datalayer zu lesen.
- **Vorgehen:** In der TIGER-IDE ein neues Projekt erstellen, die Datalayer-Extension benutzen um die Werte der Nodes im Parametrization Ordner zu lesen.
- **Erwartetes Ergebnis:** Die vorhandenen Werte der Datalayer-Nodes können erfolgreich gelesen werden.

- **Abweichungen vom erwarteten Ergebnis:** Die Datalayer-Extension von der TIGER-IDE liest die Werte nur dann erfolgreich, wenn bei der Erstellung des Projekts, das Board *ctrlx* ausgewählt wurde.

Testfall 10: TIGER-IDE - Hydraulische Achse

- **Ziel:** Prüfen, dass die TIGER-IDE imstande ist sich mit einer hydraulischen Achse zu verbinden und aktuelle Zustandswerte der Achse auszulesen.
- **Vorgehen:** Die Konfiguration einer hydraulischen Presse gemäss Kapitel 5.6.1 durchführen. Danach ein Programm mit dem Programmierblock *log oil temp...* aus dem Scope *Press* erstellen und ausführen.
- **Erwartetes Ergebnis:** Die aktuellen Zustandswerte der Achse werden auf der Konsole der TIGER-IDE ausgegeben.
- **Abweichungen vom erwarteten Ergebnis:** Keine.

Testfall 11: TIGER-IDE - Hydraulische Achse

- **Ziel:** Prüfen, dass die TIGER-IDE im Stande ist eine hydraulischen Achse zu kommandieren.
- **Vorgehen:** Die Konfiguration einer hydraulischen Presse gemäss Kapitel 5.6.1 durchführen. Danach ein Programm mit einem Programmierblock *move ...* aus dem Scope *Press* erstellen und ausführen.
- **Erwartetes Ergebnis:** Die Achse bewegt sich gemäss der Eingaben des move-Blocks.
- **Abweichungen vom erwarteten Ergebnis:** Die Achse schaltet sich ein, aber sie bewegt sich nicht. Es wird ein Compile Error angezeigt.

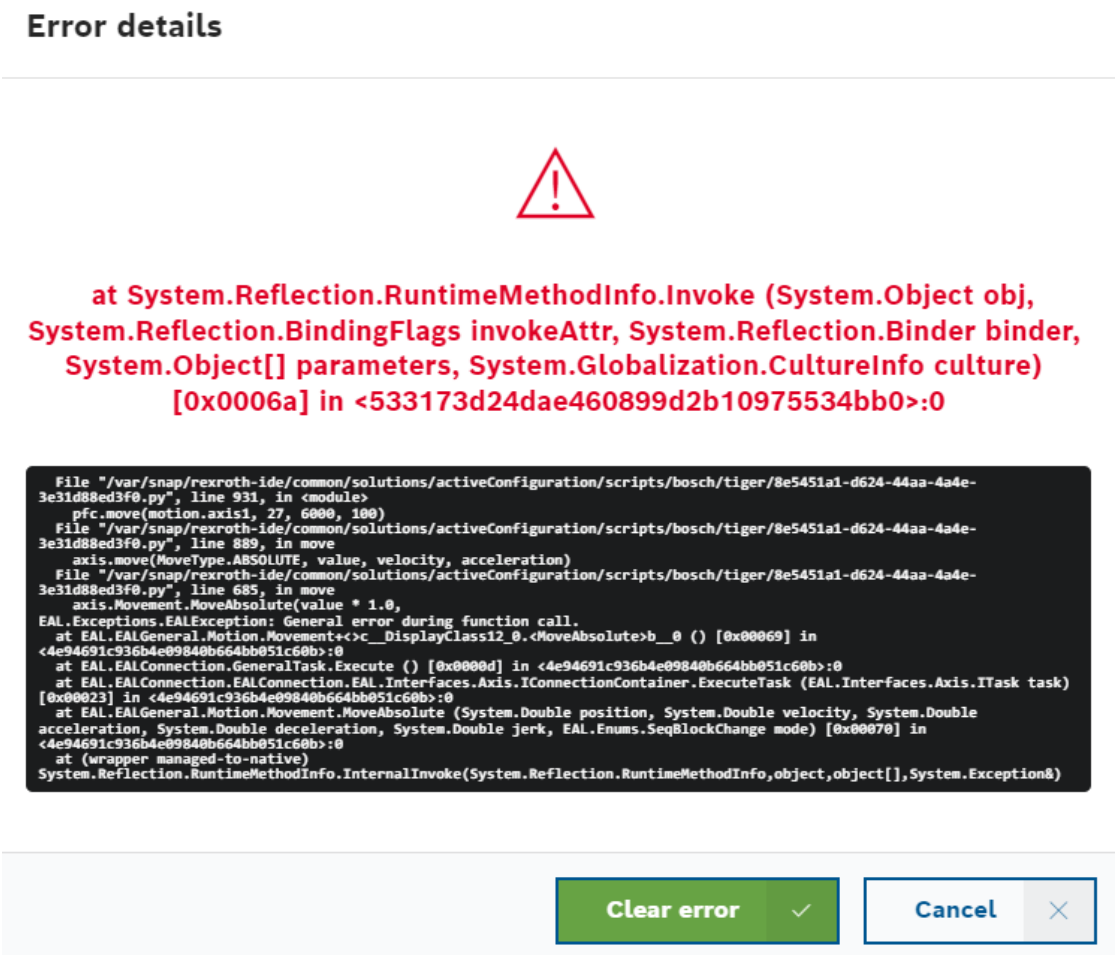


Abbildung 6.1: Error bei Versuch Achse zu bewegen.

Erkenntnisse

Abgesehen von der TIGER-IDE funktioniert die Verbindung zwischen den einzelnen Komponenten problemlos. Die Probleme der TIGER-IDE werden im nächsten Kapitel erläutert.

6.2 Validierung funktionale Anforderungen

Nachfolgend werden alle Use Cases des MVPs aufgeführt und beschrieben, ob sie erfüllt werden konnten.

Nr.	Name	erfüllt	Grund
1	Erstellen der Konfiguration	nein	fehlende API TIGER-IDE
2	Löschen der Konfiguration	nein	fehlende API TIGER-IDE
3	CRUD Parametrisierung	ja	
4	Auswählen der Parametrisierung	ja	
5	Visuelle Programmierung	ja	
6	CRUD Programmierung	nein	fehlende API TIGER-IDE
7	Ausführung auf physischer Presse	nein	
8	Anzeige des Aktuellen Zustands	ja	

Einschränkungen TIGER-IDE

Da die TIGER-IDE zum Zeitpunkt der Bachelorarbeit noch in Entwicklung ist, existiert nicht die gesamte notwendige Funktionalität. So funktioniert die Kommunikation mit der physischen Presse nur, wenn die TIGER-IDE auf der ctrlX Virtual-Core ausgeführt wird. Diese dient lediglich als Entwicklungstool, an welchem physische Geräte angeschlossen werden können. Auch auf dem Virtual-Core funktioniert die Kommunikation mit der Presse jedoch nicht zuverlässig. Die meiste Zeit wird ein Compile Error in der TIGER-IDE angezeigt, wenn versucht wird die physische Achse zu bewegen. Die angezeigten Error sind wenig aussagekräftig, weswegen auch das Entwicklungsteam der TIGER-IDE Mühe damit hat. Aber wenn versucht wird die aktuellen Zustandswerte der Achse zu lesen, ohne sie zu bewegen, funktioniert das problemlos.

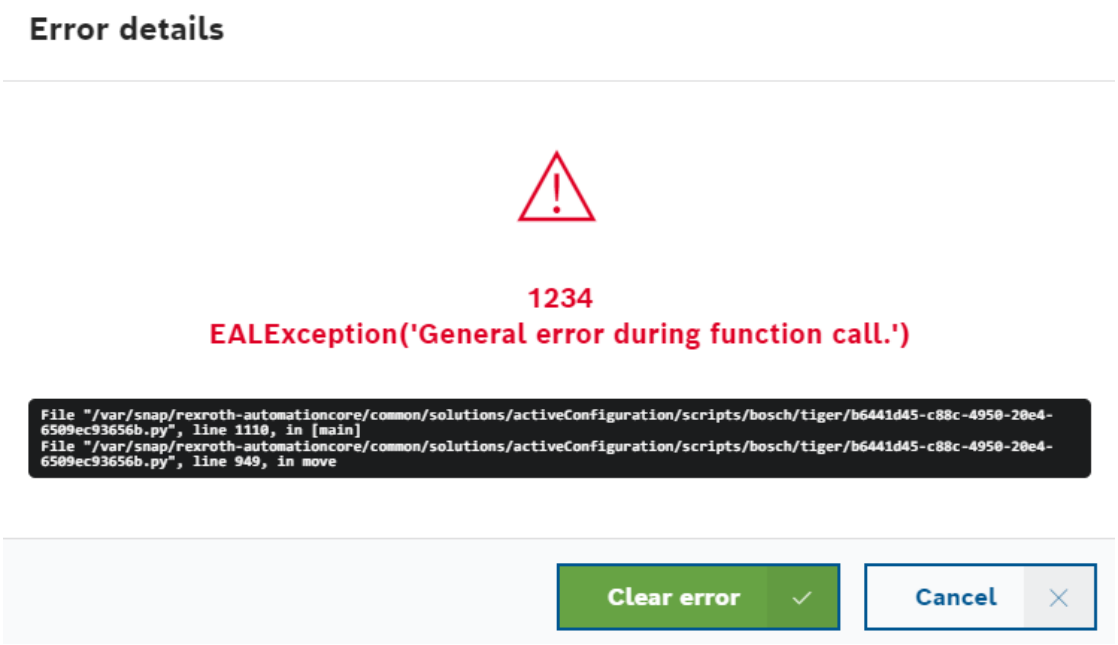


Abbildung 6.2: Weiterer Error in Zusammenhang mit einem move-Block

Zum Teil wird auch nichts angezeigt, aber die Achse bewegt sich trotzdem nicht. Wichtig zu erwähnen ist, dass die Programmierblöcke der TIGER-IDE sich bei verschiedenen Geräte mit derselben Softwareversion unterschiedlich verhalten, was vom Entwicklungsteam der TIGER-IDE nicht nachvollzogen werden konnte.
Ein weiteres Problem besteht darin, dass die TIGER-IDE noch keine umfassende API zur Verfügung stellt. So ist es nicht möglich, von der Webapp aus ein Projekt in der TIGER-IDE zu erstellen/löschen. Dies muss direkt in der TIGER-IDE Applikation geschehen. Es ist auch nicht möglich die Konfiguration auf einem Projekt über die API zu ändern. Sie kann aber über das iframe verändert werden. Somit kann die Webapp nicht autonom von der separaten TIGER-IDE Applikation verwendet werden. Zusätzlich müssen beim Erstellen eines neuen TIGER-IDE Projekts die auch die Blöcke für Hydraulikanwendungen neu erstellt werden. Dies würde idealerweise von einem Standardprojekt übernommen werden, was in der momentanen Version nicht implementiert ist.

Ein internes Problem der TIGER-IDE ist ebenfalls, dass sie entweder mit der Presse kommunizieren oder die Datalayer Extension nutzen kann, um Knoten auf dem Datalayer zu verändern. Beides gleichzeitig ist nicht möglich. Dies bedeutet, dass es zwar möglich ist, aktuelle Zustandswerte der physischen Presse über die TIGER-IDE zu lesen, diese jedoch nicht an die Webapp übermittelt werden können.

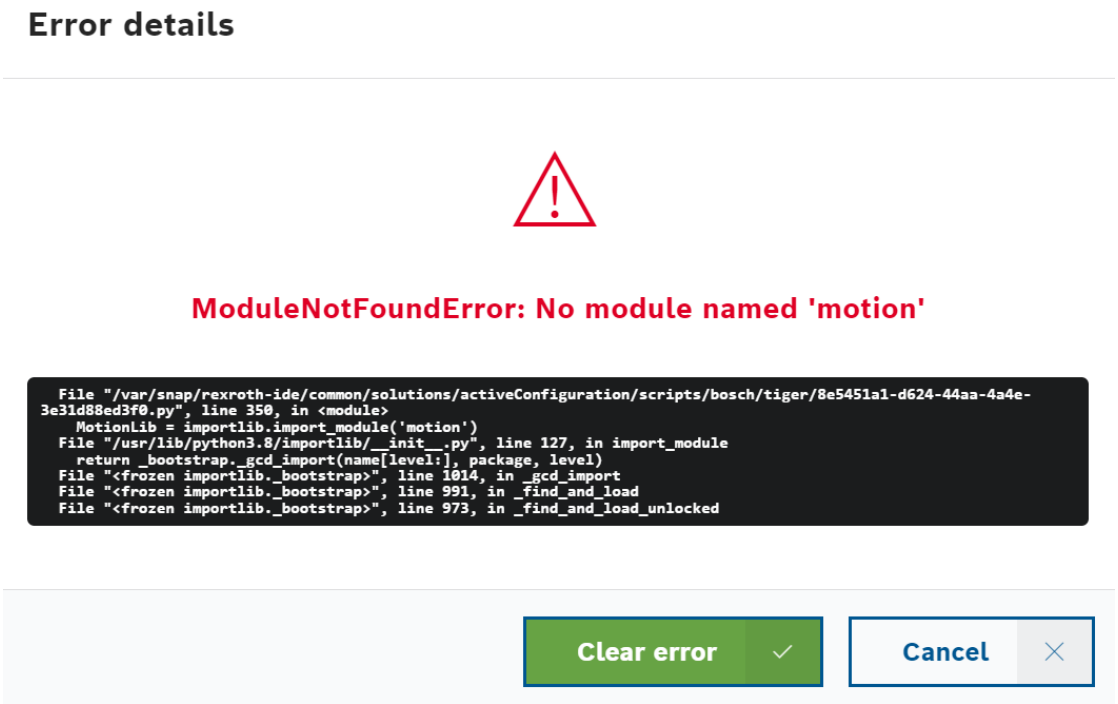


Abbildung 6.3: Error wenn Verbindung zu Achse und Datalayer gleichzeitig

Um dieses Problem zu umgehen, wurde auch versucht, an Stelle der Datalayer Extension die REST Extension der TIGER-IDE zu nutzen, um so über das Datalayer API der ctrlX

VirtualCORE Zugriff auf das Datalayer zu bekommen. Die REST Extension kann zwar benutzt werden, allerdings muss der Access-Token für die HTTP Anfragen manuell in der Konfiguration der REST-Extension eingegeben werden. Wenn der Token dann nicht mehr gültig ist, verweigert die Datalayer API die Kommunikation, bis der Token manuell erneuert wird. Die Zustandswerte können aber mit dem neuen Programmierblock *log oil temp..* auf die Konsole der TIGER-IDE ausgegeben werden.

6.3 Validierung Nicht funktionale Anforderungen

- **Performance, Effizienz:** Mit Ausnahme des ersten Ladens des iframes für die visuelle Programmierung passieren alle Interaktionen mit der Webapp ohne spürbare Verzögerung. Getestet wurde dies auf einer ctrlX-Core, dem Zielsystem für die Webapp. Der limitierende Faktor beim Laden des iframes sind allerdings die vielen Ressourcen, welche für die TIGER-IDE benötigt werden und somit kann dies in der Webapp nicht weiter optimiert werden.
- **Usability:** Bei der Abnahme durch Josef Müller wurde die Usability als angemessen empfunden. Genauere Beschreibungen sind im Kapitel 6.4 aufgeführt.
- **Zuverlässigkeit:** Bei fehlgeschlagener Verbindung mit der TIGER-IDE wird dies auf dem iframe angezeigt. Die restliche Funktionalität der Webapp kann jedoch weiterhin verwendet werden.
- **Sicherheit:** Dies wurde nicht weiter überprüft.
- **Wartbarkeit:** Die Coding Conventions von C# wurden gemäss Dokumentation [8] überprüft.

6.4 Abnahme

Die Abnahme wurde am 15. 06. 2022 von Josef Müller vorgenommen. Da es sich um eine Forschungsarbeit handelt, wurden dafür keine Abnahmekriterien definiert, sondern die umgesetzte Funktionalität (Im Kapitel 5 beschrieben), sowie die erlangten Erkenntnisse (Im Kapitel 6.2 beschrieben) präsentiert. Er nahm das Produkt mit folgendem offenen Punkt dankend entgegen:

- Beim Durchführscreen sind die dargestellten Parameter nicht sichtbar genug.

Dies wird für den Proof-of-Concept nicht mehr umgesetzt.

Kapitel 7

Fazit und Ausblick

Die Webapp beinhaltet in der schlussendlichen Version nur einen Teil der gewünschten Funktionalität. Dies liegt am momentanen Zustand der ctrlX AUTOMATION Plattform und der TIGER-IDE. So ermöglicht die erstellte Webapp nur bedingt ein einfacheres Interface zum Programmieren, als dies direkt in der TIGER-IDE möglich ist. Um dies zu verbessern sind insbesondere folgende Punkte wichtig, welche bei 6.2 genauer definiert sind.

- **Erweiterung TIGER-IDE API:** Damit die nutzende Person die gesamte Funktionalität ohne Starten der separaten TIGER-IDE Applikation einsetzen kann, muss deren API so erweitert werden, dass damit Programme erstellt und konfiguriert werden können.
- **Produktionsreife TIGER-IDE für Hydraulikanwendungen:** Die TIGER-IDE muss aktualisiert werden, sodass damit auch eine physische ctrlX-Core mit der Hydraulikanwendung kommunizieren kann. Momentan ist dies nur mit der virtual-Core möglich. Ausserdem muss ermöglicht werden, gleichzeitig den Kontakt mit der Anwendung und dem Datalayer zu haben, was momentan nicht geht. Schlussendlich gibt es im momentanen Zustand Bugs, welche die Verbindung mit der Hydraulikanwendung abbrechen und die Nutzung stark erschweren.
- **Automatisches Einfügen neuer Blöcke:** Um das Erstellen neuer TIGER-IDE Projekte zu erleichtern, sollte eine Möglichkeit zur Verfügung gestellt werden, die neu erstellten Blöcke für Hydraulikanwendungen direkt in neue Projekte einzufügen, sodass der Code daraus nicht manuell kopiert und mittels GUI der TIGER-IDE eingefügt werden muss.

Wenn diese Punkte umgesetzt sind, kann die Webapp erweitert werden, sodass sie schliesslich der Vision entspricht.

Teil III

Anhang

Kapitel 8

Zeiterfassung

Nachfolgend ist die aufgewendete Arbeitszeit der beiden Studierenden für die Bachelorarbeit dokumentiert. Die Zeit wurde mit dem Tool *clockify* erfasst und ausgewertet. Auf einige Punkte wird speziell eingegangen.

8.1 Gesamtzeit pro Teammitglied

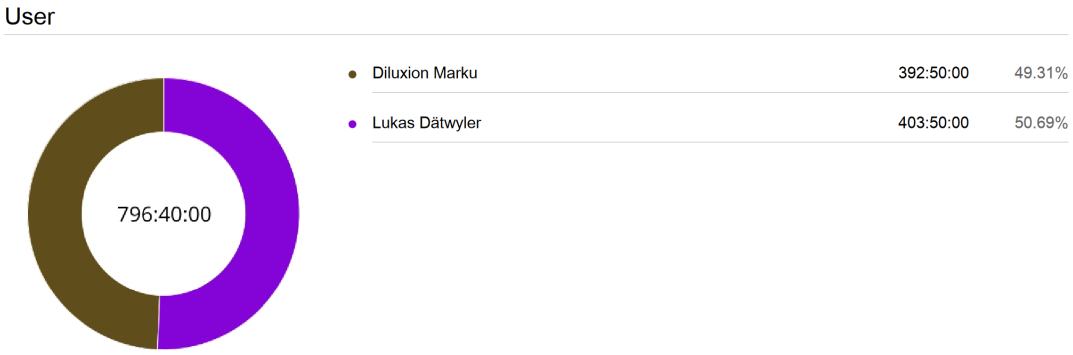


Abbildung 8.1: Gesamte Arbeitszeit pro Teammitglied

Diese Ansicht zeigt, dass etwas mehr als die erwarteten 360 Arbeitsstunden pro Student aufgewendet wurden. Der Aufwand ist allerdings sehr gleichmässig unter den Studierenden aufgeteilt.

8.2 Aufwand pro Kategorie

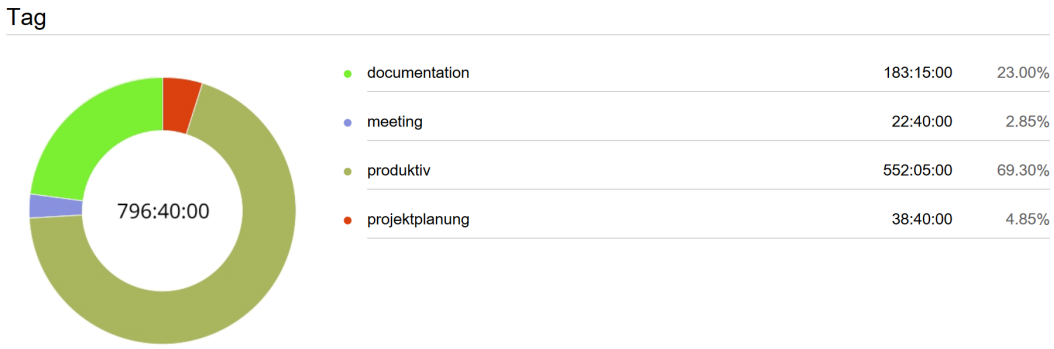


Abbildung 8.2: Gesamte Arbeitszeit pro Teammitglied

Diese Darstellung zeigt, dass die Projektplanung mit weniger als 5% des Aufwands recht klein ausfiel. Dies ist dadurch zu erklären, dass sie sich in vielen Teilen mit der Dokumentation überschneidet und dort aufgeführt wurde.

8.3 Verteilung unter den Studierenden

Kategorie	Diluxion Marku	Lukas Dätwyler
Dokumentation	78h	105h
Meetings	11h	11h
Produktiv	282h	269h
Projektplanung	21h	18h

Es ist ersichtlich, dass Diluxion Marku etwas mehr Zeit in produktive Arbeiten aufwendete während Lukas Dätwyler etwas mehr an der Dokumentation arbeitete. Dies liegt unter anderem daran, dass Diluxion Marku für die Einbindung der TIGER-IDE zuständig war. Für diese Arbeit war viel Kommunikation mit dem zuständigen Entwicklungsteam nötig.

Kapitel 9

Erfahrungsbericht

9.1 Lukas Dätwyler

Bei der Arbeit schlug ich mich grösstenteils mit ASP.NET MVC und dem ganzen .NET Umfeld herum. Dies war für mich auch der spannendste Teil. Das betrifft insbesondere den Aufbau der MVC Applikation, die Schnittstelle mit der LiteDB und das Übertragen von Echtzeitdaten mittels signalR. All diese Technologien konnten auch recht gut eingesetzt werden und funktionierten zufriedenstellend. Positiv zu erwähnen ist die sehr ausführliche Dokumentation der Microsoft Technologien, welche das Leben an einigen Stellen stark erleichterten.

Mit den Technologien von ctrlX AUTOMATION kam ich vor allem bei der Implementation des Datalayer-Services in Kontakt. Dabei konnte natürlich nicht mit einer vergleichbaren Dokumentation zu Microsoft gerechnet werden. Es standen jedoch mehrere Sample-Projekte zur Verfügung, mit welchen die API grob hergeleitet werden konnte. Frustrierend war lediglich, dass die API augenscheinlich nicht für alle Versionen der ctrlX Core Plattform gleich gut funktioniert und die Software so auf verschiedenen Geräten unterschiedliche Resultate lieferte. Diese Probleme sind vermutlich dadurch zu Erklären, dass die gesamte Plattform noch recht jung ist.

Nachträglich muss ich feststellen, dass die Priorisierung der Arbeitspakete im Projektplan vermutlich hätte sinnvoller gewählt werden können. So stand zwar recht früh ein Grundgerüst der Webapp mit einigen Funktionen, doch der wichtigste Teil der Applikation, das Steuern der hydraulischen Achse, wurde erst später angegangen. Gerade wegen der starken Abhängigkeit zur TIGER-IDE wäre es vermutlich besser gewesen, wenn dies früher geschehen wäre.

9.2 Diluxion Marku

In dieser Arbeit habe ich mich nicht nur mit den .NET Technologien beschäftigt, sondern auch viel mit der ctrlX AUTOMATION Umgebung. Zu Beginn der Arbeit wurde

die meiste Zeit darin investiert die ctrlX-Plattform kennenzulernen und zu verstehen. Dabei war es spannend zu sehen, dass auch in der Maschinenindustrie vermehrt auf IT-Technologien wie Container-basierte Architekturen und Webapps gesetzt wird. Sehr spannend fand ich auch die Arbeit mit dem Tool Snapcraft, welche die Kapselung der Webapps überhaupt ermöglicht.

Da wir für den visuellen Programmierungs-Teil der Arbeit die TIGER-IDE verwenden sollten, habe ich mich mit den vorhandenen Funktionen dieser Komponente beschäftigt. Leider verlief nicht alles so, wie wir uns das vorgestellt hatten. Vor allem aber, was den Funktionsumfang der API der TIGER-IDE betrifft. Wenn wir das zu Beginn der Arbeit gewusst hätten, hätten wir die Arbeit vielleicht anders angehen und strukturieren können. Nichtsdestotrotz, war die Arbeit sehr spannend und lehrreich. Sowohl der Teil mit den .NET Technologien als auch das Arbeiten mit Industriemaschinen.

Literaturverzeichnis

[1] "ctrlx core," Bosch Rexroth AG, 2022, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://apps.boschrexroth.com/microsites/ctrlx-automation/files/ctrlx/Produkte/CORE/DC-AE_ctrlX_CORE_PST3315_02_432x750px.jpg

[2] "ctrlx automation – two steps ahead," Bosch Rexroth AG, 2022, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://apps.boschrexroth.com/microsites/ctrlx-automation/en/

[3] "Google blocky," Google Developers, 2022, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://developers.google.com/blockly

[4] "Ide app, integrated development environment, anwendungsbeschreibung," Bosch Rexroth AG, 2022, zuletzt am 10. Juni 2022 besucht. [Online]. Available: https://docs.automation.boschrexroth.com/doc/376945218/ide-app-integrated-development-environment-anwendungsbeschreibung/latest/iv/

[5] I. 25000, "Iso/iec 25010," 2022, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[6] "Sfk4p software anleitung de," Bosch Rexroth AG, 2022, zuletzt am 10. Juni 2022 besucht. [Online]. Available: https://www.boschrexroth.com/en/xc/myrexroth/document-library?p_p_id=20&p_p_lifecycle=0&p_p_mode=view&p_p_state=maximized&_20_struts_action=%2Fdocument_library%2Fview_file_entry&_20_redirect=./&_20_fileEntryId=43558701

[7] M. Docs, "Model validation in asp.net core mvc and razor pages," 2022, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-6.0

[8] "Coding conventions," Microsoft Docs, 2022, zuletzt am 24. Mai 2022 besucht. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions

[9] M. Docs, "Asp.net overview," 2022, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/overview

[10] “Snap documentation,” Canonical, 2022, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://snapcraft.io/docs

[11] CodeShepherd, “Faq for ctrlx data layer,” 2021, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://developer.community.boschrexroth.com/t5/Store-and-How-to/FAQ-for-ctrlX-Data-Layer/ba-p/21236

[12] “ctrlx automation sdk,” Bosch Rexroth AG, 2022, zuletzt am 15. Juni 2022 besucht. [Online]. Available: https://boschrexroth.github.io/ctrlx-automation-sdk/overview.html

[13] S. Brown, “The c4 model for visualising software architecture,” 2022, zuletzt am 15. April 2022 besucht. [Online]. Available: https://c4model.com/

[14] “Choose an aps.net core web ui,” Microsoft Docs, 2022, zuletzt am 15. April 2022 besucht. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/tutorials/choose-web-ui?view=aspnetcore-6.0

[15] “Litedb,” LiteDB, 2022, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://www.litedb.org/

[16] CodeShepherd, “Using rest api of ctrlx core,” 2020, zuletzt am 7. Juni 2022 besucht. [Online]. Available: https://developer.community.boschrexroth.com/t5/Store-and-How-to/Using-REST-API-of-ctrlX-CORE/ba-p/12490

[17] M. contributors, “¡iframe¿: The inline frame element,” 2022, zuletzt am 7. Juni 2022 besucht. [Online]. Available: https://restfulapi.net/

[18] bostroemc, “Rest access using python,” 2022, zuletzt am 10. Juni 2022 besucht. [Online]. Available: https://developer.community.boschrexroth.com/t5/Communication/REST-Access-using-Python/m-p/14842

[19] “Overview of asp.net core signalr,” Microsoft Docs, 2022, zuletzt am 14. Juni 2022 besucht. [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction?view=aspnetcore-6.0

[20] “Rexroth sytronix svp 7020 pfc, drehzahlvariables positionieren hydraulischer achsen,” Bosch Rexroth AG, 2020, zuletzt am 10. Juni 2022 besucht. [Online]. Available: https://www.boschrexroth.com/en/xc/myrexroth/document-library?p_p_id=20&p_p_lifecycle=0&p_p_mode=view&p_p_state=maximized&_20_struts_action=%2Fdocument_library%2Fview_file_entry&_20_redirect=./&_20_fileEntryId=29761649

[21] “Unit testing in cs .net core using dotnet test and xunit,” Microsoft Docs, 2022, zuletzt am 7. Juni 2022 besucht. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-dotnet-test

[22] D. Cazzulino, A. Alonso, D. Schuppli, and B. Conrad, “moq v5,” 2022, zuletzt am 15. Juni 2022 besucht. [Online]. Available: https://github.com/moq/moq