Hector Seminar

wbk Institut für Produktionstechnik | KIT Karlsruher Institut für Technologie

Web-Entwicklung

Abb. 1 Bosch Rexroth Nexo Akkuschrauber

Abschlussbericht der Kooperationsphase 2018/19

Durchgeführt am Institut für Produktionstechnik (wbk),
Karlsruher Institut für Technologie (KIT)

Betreuer: Tom Stähr, Constantin Hofmann

Johannes Huber

Milena Lara Seeburger

Inhaltsverzeichnis

Abstract

Industry 4.0 – Automated solutions for the production of electric motors.

From the individual parts to the finished product – with the help of automated machines, which can be controlled easily. What still sounds like a vision is already reality in the learning factory ‘Global Production’ at the Institute of Production Science (wbk), part of the Karlsruhe Institute of Technology (KIT).

The task was to integrate a cordless screwdriver into an existing production line at wbk. To collect the data of the screwdriver, a server with a database was needed and its connection to the screwdriver had to be established via HTTP.

Therefore, important parameter such as the rotational angle and the torque were measured, put into graphs and represented on a website, compiled by Angular.

In order to complete the project successfully, the screwdriver must still be integrated into the production line and the program must be adapted to the requirements of the learning factory.

1. Einleitung

Industrie 4.0 – zunehmende Automatisierung bestimmt die vierte industrielle Revolution. In der Lernfabrik Globale Produktion (Abb. 2) des Instituts für Produktionstechnik (wbk), Karlsruher Institut für Technologie (KIT), arbeiten Studierende und Doktoranden an der Entwicklung von Automatisierungslösungen. In einer Industriemontagelinie für die Produktion von Elektromotoren können dort verschiedene Industrie 4.0-Anwendungen gesehen und ausprobiert werden. Basierend auf dem Lean Management, einem Ansatz zur kontinuierlichen Prozessoptimierung für ein verschwendungsfreies Produktionssystem, werden in der Lernfabrik Schulungen für Fach- und Führungskräfte, Projektleiter sowie Produkt- und Prozessverantwortliche aus allen Unternehmensbereichen angeboten. Am Beispiel der vorhandenen Montagelinie kann die Produktion einer variantenreichen Kleinserie erprobt werden. Die eingesetzten Werkzeuge erfassen dabei Daten, welche in einem Produktionsleitsystem gesammelt und ausgewertet werden. In so genannten Shopfloor-Meetings werden anhand der gewonnenen Daten die implementierten Methoden und Werkzeuge analysiert und der Prozess reflektiert.

Abb. 2 Lernfabrik Globale Produktion

Ziel des Projektes ist es, den Funk-Akkuschrauber Nexo der Firma Bosch-Rexroth in die Produktionsline für Elektromotoren zu integrieren. Die gewonnenen Daten, z. B. der Verlauf von Drehwinkel und Drehmoment sollen gesammelt, grafisch aufbereitet und auf einer Website ausgegeben werden, um sie den Schulungsteilnehmern für Untersuchungen zur Verfügung zu stellen. Die Datenverwaltung dazu wird in Java programmiert, zur Datenspeicherung wird mongoDB benutzt. Zur Umsetzung der Website wird mit Angular, einem TypeScript-basiertem Frontend-Webapplikationsframework, gearbeitet.

2. Material und Methoden

2.1 Der Bosch-Rexroth Nexo Funk-Akkuschrauber

Beim eingesetzten Funk-Akkuschrauber handelt es sich um den Bosch-Rexroth Nexo, einen Mittelgriffschrauber, dessen gesamte Steuerung samt Display sowie ein Barcode-Scanner im Schrauber integriert ist (Abb. 1 und Abb. 3). Durch drahtlose Netzwerkfähigkeit lässt er sich an übergeordnete Steuerungssysteme anbinden. Das Gerät ist einfach zu bedienen: ein Mitarbeiter kann die zu verschraubenden Komponenten mit dem Barcode-Scanner erfassen und mittels vordefinierter Programme verschrauben.

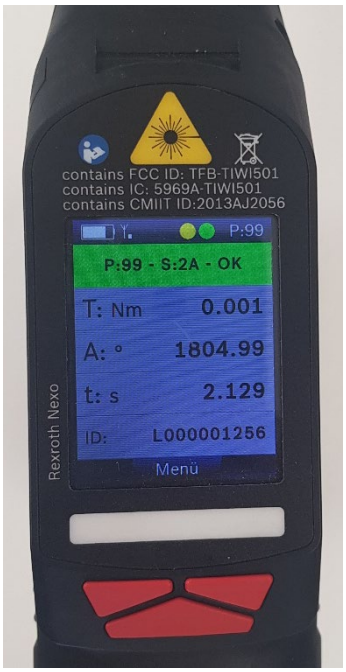

Abb. 3 Display des Akkuschraubers

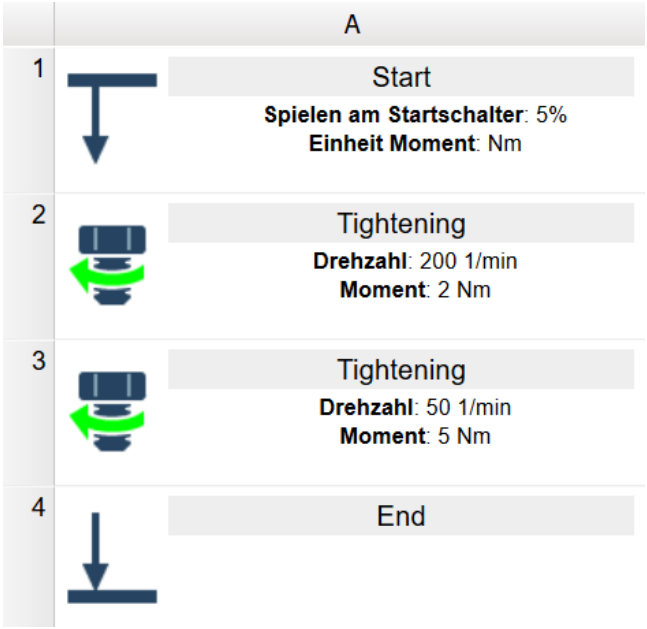

Abb. 4 Schraubstufen in einem Schraubprozess

Ein solches Programm besteht aus verschiedenen Schraubstufen (Abb. 4). Innerhalb derer werden Ziel-, Überwachungs- und Zusatzfunktionen unterschieden. Die Zielfunktion ist die innerhalb einer Schraubstufe relevante Steuerungsfunktion, z. B. das Drehmoment bei drehmomentgesteuerten Schraubverfahren. Wird der Zielparameter (= Sollwert) erreicht, endet die Schraubstufe. Zusätzlich zur ständig aktivierten ersten Zielfunktion ist es möglich, eine zweite Zielfunktion zu aktivieren. Dann beendet die zuerst erreichte Zielfunktion die Schraubstufe.

Innerhalb einer Schraubstufe „beobachten“ Überwachungsfunktionen den Schraubprozess. Überwachungsparameter, also Grenzwerte einer Überwachungsfunktion, dienen dabei als Grundlage für eine Schraubstufenbewertung. Damit können die erzielten Schraubergebnisse (z. B. Drehmoment, Drehwinkel) am Ende einer Schraubstufe als „OK“ bzw. „NOK“ (Not-OK)

bewertet werden. Wird ein Überwachungsparameter verletzt, kann eine Schraubstufe sogar vorzeitig abgebrochen werden. Zusatzfunktionen, wie z. B. eine Anlaufunterdrückung, beeinflussen zwar den Schraubverlauf, bewerten ihn aber nicht. Informationen, wie etwa wann eine Verschraubung durchgeführt oder abgeschlossen wurde, sowie mögliche Probleme können somit vom Gerät erfasst und an Produktionsleitsysteme weitergeleitet werden. Dies kann über einen HTTP-Server geschehen. Dabei versendet der Schrauber nach jeder Verschraubung ein JSON-File.

Die JavaScript Object Notation, kurz JSON, ist ein kompaktes Datenformat, welches für Menschen einfach zu lesen und für Maschinen leicht zu verarbeiten ist. Es ist ein von Programmiersprachen unabhängiges Textformat, basierend auf Schlüssel/Wert-Paarungen und geordneten Listen von Werten.
Abb. 5 zeigt einen gekürzten Datensatz des Schraubers. Neben Resultat („result“) und Zeitpunkt („date“) der Verschraubung finden sich Name des Verschraubungsprogrammes („prg name“) und ein ID-Code zur eindeutigen Identifikation („id code“). Der Qualitätscode („quality code“) macht eine Aussage über das Ergebnis einer Verschraubung. Bei einer NOK-Bewertung wird außerdem die Ursache hierfür beschrieben. Durch die Zuordnung der Ursache von NOK-Bewertungen von Schraubabläufen zu bestimmten Qualitätscodes ist eine Einteilung in einzelne Bereiche möglich. Dies kann z. B. bei einer Auswertung von ausgegebenen Schraubergebnissen hilfreich sein. Durch das letzte Schraubprogramm-Kommando („last cmd“) wird bei einer fehlerhaften Verschraubung der Grund für einen vorzeitigen Abbruch deutlich. Zu jeder Schraubstufe („tightening steps“) werden neben dem Resultat der einzelnen Stufe („result“) deren End-Drehmoment und -winkel („torque“, „angle“) sowie Zeitbedarf („duration“) und Einzelwerte mit zugehörigen Zeitwerten - jede Millisekunde entspricht dabei einem Messwert („graph“) - aufgelistet. Zusätzlich werden Ziel-, Überwachungs- und Zusatzfunktionen des Programmes („tightening functions“) mit jeweiligen Grenz- („nom“) und tatsächlichen Werten („act“) ausgegeben. Anhand dieser Daten kann die Qualität des Vorgangs bewertet werden.

```
{
   "result":  "OK",
   "prg name":    "X009",
   "date":    "2000-01-16 07:05:36",
   "id code": "L000147953",
   "torque unit": "Nm",
   "last cmd":    "TF Torque",
   "quality code":    "1",
   "total time":  "1.275000",
   "tightening steps":    [{
         "result":  "OK",
         "name":    "Verschrauben",
         "last cmd":    "TF Torque",
         "torque":  4.01300,
         "angle":   2708,
         "duration":    1.17500,
         "quality code":    "1",
         "speed":   400,
         "angle threshold nom": 0,
         "angle threshold act": 0.06900,
         "tightening functions":    [...],
         "graph":   {
            "angle values":    [1631, 1634, [...], 2706, 2708],
            "torque values":   [0.06900, 0.06900, [...], 3.81700, 4.01300],
            "time values": [0.70900, 0.71000, [...], 1.17400, 1.17500]
         }
      }, {
         "result":  "OK",
         "name":    "Verschrauben",
         "last cmd":    "TF Torque",
         "torque":  9.26700,
         "angle":   36,
         "duration":    0.10000,
         "quality code":    "1",
         "speed":   50,
         "angle threshold nom": 0,
         "angle threshold act": 4.08800,
         "tightening functions":    [{
               "name":    "TF Torque",
               "nom": 9,
               "act": 9.04900
            }, {
               "name":    "MFs TimeMax",
               "nom": 10,
               "act": 0.10000
            }, {
               "name":    "MF TorqueMin",
               "nom": 7,
               "act": 9.26700
            }, {
               "name":    "MF TorqueMax",
               "nom": 11,
               "act": 9.26700
            }],
         "graph": [...]
      }]
}
```


Abb. 5 Gekürzter JSON-Datensatz des Akkuschraubers

2.2 Programmierung

2.2.1 Backend

Als Backend wird der Teil eines IT-Systems bezeichnet, der sich mit der Datenverarbeitung im Hintergrund beschäftigt. Die vom Akkuschrauber gesammelten Daten über beispielsweise Drehmoment, Drehwinkel und Winkelgeschwindigkeit (Abb. 5) müssen empfangen und in einer Datenbank abgespeichert werden. In der Lernfabrik wird dazu mit mongoDB[1] gearbeitet, einer NoSQL-Datenbank mit dokumentorientiertem Speichersystem. Ein Dokument ist dabei eine Sammlung von Schlüssel/Werte-Paaren im BSON Format, einer binären Darstellung von JSON-ähnlichen Dokumenten. Um die im JSON-Format vorliegenden Datensätze des Akkuschraubers in der mongoDB speichern zu können, muss eine Verbindung zwischen Backend-Server und Akkuschrauber hergestellt werden. Dies wird über einen REST-Controller realisiert. REST steht für REpresentional State Transfer und bezeichnet einen Ansatz für die Kommunikation zwischen verteilten Systemen. Dies wird durch HTTP realisiert. Services werden per URL/URI angesprochen und HTTP-Methoden geben an, welche Operation ein Dienst ausführen soll (vgl. Abb. 6).

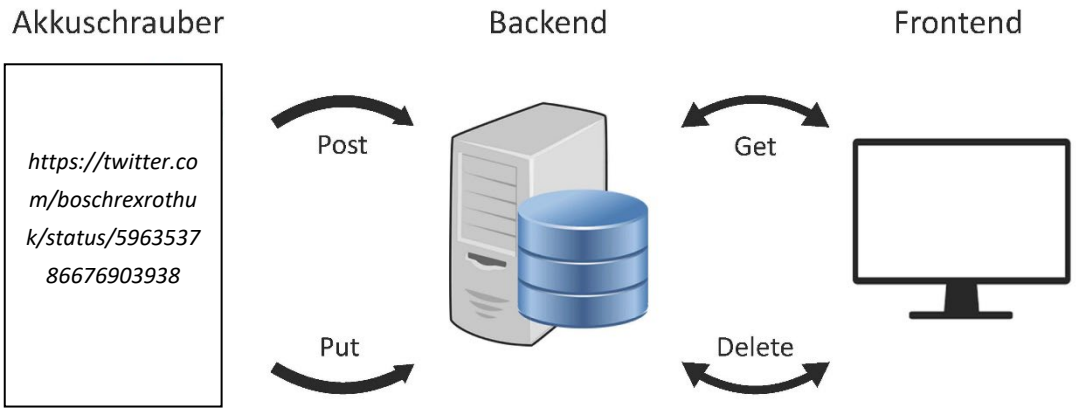

Abb. 6 Visualisierung des Datenverkehrs

In der Entwicklungsumgebung Intellij IDEA[2] wurde - basierend auf Java Version 8, Build 201[3] - mittels Spring Boot[4], einem quelloffenen Framework zur Vereinfachung der Entwicklung mit Java, ein einfacher RESTful Web Service aufgesetzt, welcher über HTTP kommunizieren kann.

[1] https://www.mongodb.com
[2] https://www.jetbrains.com/idea/
[3] https://www.java.com
[4] https://spring.io/projects/spring-boot

Dieser ist auf drei Ebenen aufgeteilt. Der Controller (vgl. 6.2.1.1 NexoController) nimmt HTTP-Requests entgegen und gibt das JSON-File des Akkuschraubers als DTO, zu Deutsch „Datenübertragungsobjekt“ an den Service Layer weiter (vgl. 6.2.1.2 NexoService). Dieser kann mögliche Arbeiten an dem Datensatz vornehmen und gibt ihn dann als Entity (vgl. 6.2.1.4 TighteningProcess), welches ein Abbild des Datensatzes darstellt, an den DAO Layer weiter. DAO steht für „Datenzugriffsobjekt“ und übernimmt den Zugriff auf die Datenbank (vgl. 6.2.1.3 NexoDAO). Damit können Datensätze in die Datenbank geschrieben und ausgelesen werden. Abb. 7 zeigt den Ablauf der Datenverwaltung als Diagramm.

Der Akkuschrauber versendet für jede Verschraubung automatisch einen Datensatz an einen HTTP-Server im JSON-Format. Dieser wird abgespeichert und von dort an im Frontend weiterverwendet.

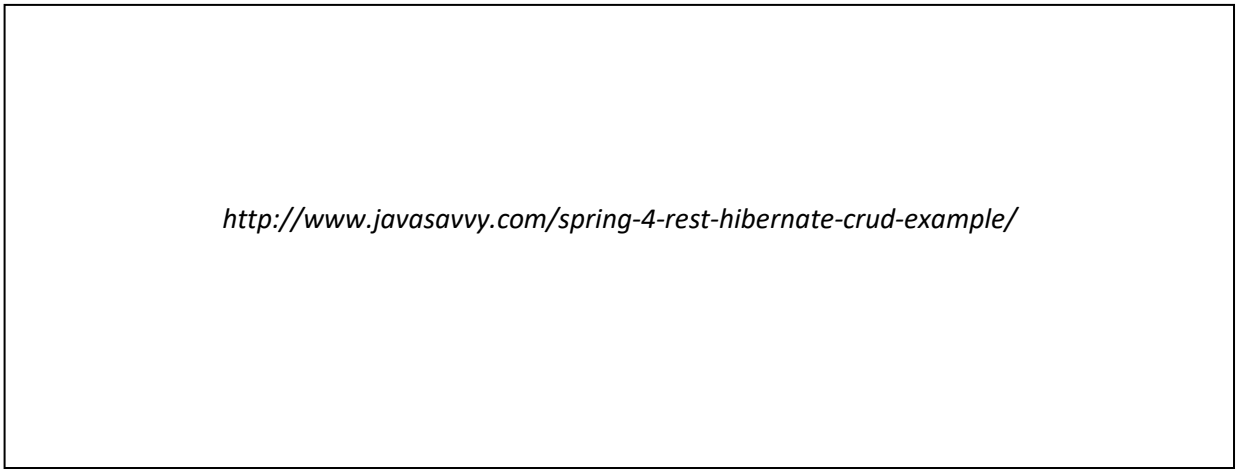

Abb. 7 Visualisierung des Backend-Servers (Pokuri, 2017)

2.2.2 Frontend

Als Frontend bezeichnet man den Teil einer Applikation, den der Benutzer sieht, z. B. in Form einer grafischen Benutzeroberfläche. Die Verschraubungsdaten werden auf einer Website dargestellt. Dies wird mittels Angular[5] (Version 8), einem TypeScript-basiertem Frontend-Webapplikationsframework realisiert. Die wichtigsten Bausteine einer Angular-App sind die Komponenten. Über diese werden die einzelnen Teile einer Website strukturiert, eine App gleicht im Aufbau einer Baumstruktur. Eine Komponente besteht dabei aus einem Template, einer Komponentenklasse und einem optionalen Stylesheet. Das Template spiegelt zusammen mit dem Stylesheet die Benutzeroberfläche wider, während die Komponentenklasse die dahinterstehende Logik zur Verfügung stellt. Die eigentliche Business-Logik wiederum wird in separaten Services ausgelagert (vgl. Abb. 8).

Abb. 8 Angular Architektur (Veits, 2017)

Mit dem HTTP-Client von Angular wird ein HTTP-GET-Request an den Backend-Server geschickt. Der Controller nimmt die Anfrage entgegen und gibt sie an den Service-Layer weiter. Dieser sucht die nach Datum sortiert letzte Verschraubung, die sich in der Datenbank befindet, und gibt sie dem Frontend-Client als JSON-Datensatz zurück (vgl. 6.2.1 Backend). Dieser Datensatz wird auf ein Klassen-Objekt übertragen.

[5] https://angular.io/

Pro Verschraubungsschritt wird nun mittels Chart.js[6] ein Graph erstellt. Chart.js ist eine JavaScript Libary, welche mittels des HTML5 <canvas> Elementes aus Daten Diagramme erstellen kann. Der Graph (Abb. 9) stellt den zeitlichen Verlauf von Drehmoment und Drehwinkel dar. Neben den Informationen und Graphen zu jedem Verschraubungsschritt werden auch Resultat, Zeitpunkt und -dauer der Verschraubung sowie Name des Verschraubungs- programmes angezeigt (vgl. Anhang Abb. 14). So lässt sich schnell und einfach ein Überblick über die Verschraubung gewinnen. Die Anwendung zeigt die jeweils letzte Verschraubung in der Datenbank. Da allerdings sämtliche Daten gesammelt und verfügbar sind, ist die Darstellung und Verarbeitung der Daten auf unterschiedlichste Art und Weisen möglich.

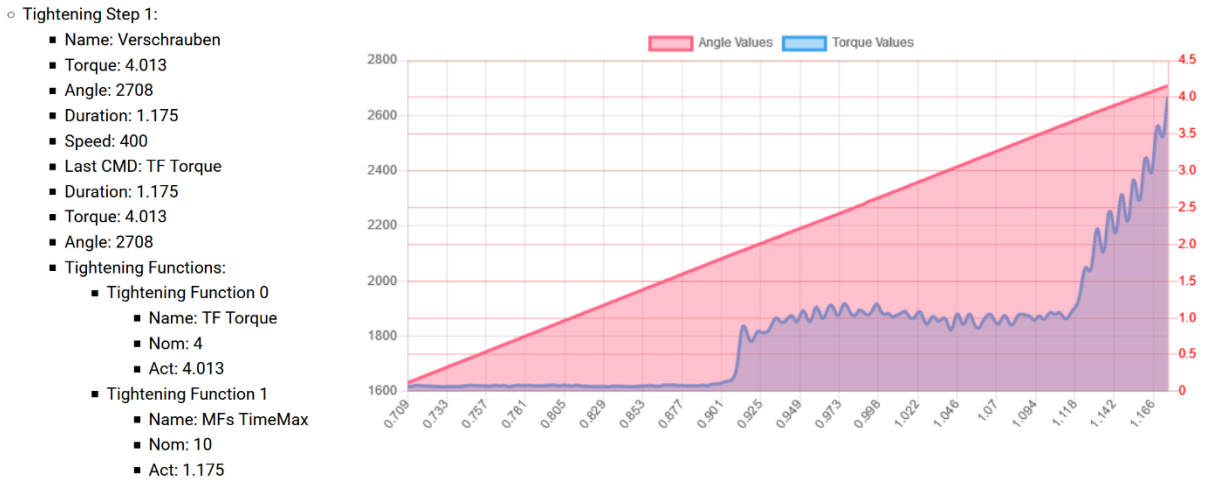

Abb. 9 Beispielausgabe eines Graphen

In Abb. 10 beispielsweise werden die letzten zehn Verschraubungen in einer Liste dargestellt. Dies erlaubt einen schnellen Überblick über Resultat, IdCode, Name und Zeitpunkt. Die jeweilige Verschraubung kann über den IdCode per Click aufgerufen werden. Über den „Load More“-Button werden die nächsten zehn Verschraubungen aus der Datenbank geladen und der Liste hinzugefügt.

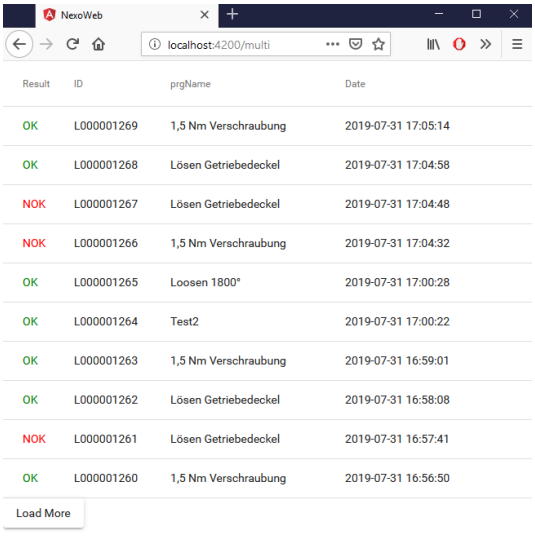

Abb. 10 Listenansicht der letzten Verschraubungen

[6] https://www.chartjs.org/

3. Diskussion

Das Ziel, Daten über Drehmoment- und Drehwinkelverlauf für eine Verschraubung grafisch darzustellen, wurde erreicht. Zusätzlich dazu werden zur Analyse der Verschraubung wichtige Daten wie Zeitpunkt und Name angezeigt. Dies ist sowohl für die letzte als auch über die Listenansicht für jede andere, in der Datenbank gespeicherte, Verschraubung möglich. Über den IdCode können die jeweiligen Detailseiten aufgerufen werden, um eine Verschraubung genau zu beurteilen.

Beim Programmieren dieser Funktionen sind häufig Fehler aufgetreten, die sich verschiedenen Bereichen zuordnen lassen. Syntaktische Fehler sind Verstöße gegen die syntaktischen Regeln einer Programmiersprache, daher kann der Compiler sie während der Analysephase erkennen. Sie treten meist zur Übersetzungszeit, also dem Zeitpunkt, an dem der Compiler den Quellcode in ein ausführbares Programm übersetzt, auf. Der Compiler identifiziert diese und gibt dabei oftmals sogar die entsprechende Zeilennummer mit kurzer Beschreibung aus, sodass diese in der Regel leicht behoben werden können (vgl. Abb. 11).

```
this.dataSource.push({
  result: tempProcess.result,
  prgName: tempProcess.prgName,
  date: tempProcess.date,
});
```

```
ERROR in src/app/multiview/multiview.component.ts(46,32): error TS2345:
Argument of type '{ result: string; prgName: string; date: string; }' is
not assignable to parameter of type 'DataInterface'.

Property 'idCode' is missing in type '{ result: string; prgName: string;
date: string; }' but required in type 'DataInterface'.
```

```
this.dataSource.push({
  result: tempProcess.result,
  prgName: tempProcess.prgName,
  date: tempProcess.date,
  idCode: tempProcess.idCode //Fehlt oben
});
```

Abb. 11 Beispiel eines Syntaxfehlers mit Compilerausgabe im Frontend (vgl. 6.2.2.3 Multiview.component.ts)

Semantische Fehler sind Fehler in der Programmlogik; diese werden vom Compiler nicht zwingend entdeckt. Dadurch kann es zu einem Laufzeitfehler kommen, gemeinhin bekannt als Bug. Die Fehlersuche kann sich dabei als schwierig erweisen, gerade wenn man mit externen Geräten arbeitet, da der Fehler nicht zwingend im Quellcode liegen muss. In unserem Projekt erhielten wir vom Akkuschrauber lange Zeit scheinbar leere Datensätze. Da das Programm mit Fake-Datensätzen einwandfrei funktionierte, beschränkte sich die Fehlersuche im Folgenden

auf den Akkuschrauber. Trotz intensiver Suche führte dies nicht zum Erfolg. Wie sich herausstellte, verbarg sich die Ursache des Fehlers im Programm, da der von Akkuschrauber gelieferte Datensatz nicht einwandfrei ausgelesen und weiterverarbeitet wurde. Die einzelnen Schlüssel des JSON-Files folgen, im Gegensatz zu den Variablennamen des Java-Objectes, nicht der camelCase-Notation. Somit ist eine simple Übertragung nicht möglich, stattdessen muss jede Variable dem dazugehörigen JSON-Schlüssel zugewiesen werden (vgl. Abb. 12).

```
/*
  Funktioniert nicht, da die Schlüssel im JSON-File nicht der
  camelCase-Notation folgen und somit nicht übereinstimmen
*/
private int prgNr;
private String prgName;
private String prgDate;
```

```
@JsonProperty(value = "prg nr")
private int prgNr;
@JsonProperty(value = "prg name")
private String prgName;
@JsonProperty(value = "prg date")
private String prgDate;
```

Abb. 12 Beispiel eines semantischen Fehlers im Backend (vgl. 6.2.1.4)

Als bei der späteren Programmierung des Frontend-Clients ein ähnliches Problem auftrat, konnte der Fehler schnell gefunden werden. Hier schlug die Übertragung der Antwort des Backend-Servers auf ein Objekt, mit dem weitergearbeitet werden kann, fehl (vgl. Abb. 13).

```
constructor(data) {
  /*
    Die Werte des data Parameters (der Antwort des Backends) können
    so nicht zugewiesen werden
  */
  Object.assign(this, data);
}
```

```
constructor(data) {
  this.prgNr = data['prg nr'];
  this.prgName = data['prg name'];
  this.prgDate = data['prg date'];
}
```

Abb. 13 Beispiel eines semantischen Fehlers im Frontend (vgl. 6.2.2.6 Tightening-process.ts)

Mit dem Projekt wurden die grundlegenden Strukturen, um den Funk-Akkuschrauber in die Lernfabrik zu einzubinden, gelegt. Für den konkreten Einsatz in der Produktionslinie für Elektromotoren muss das Programm in das Produktionsleitsystem der Lernfabrik eingefügt werden. Die erfassten Verschraubungsdaten sollen in der Umgebung des Servers der Lernfabrik verwendet und angezeigt werden können. Dazu müssen einzelne Verschraubungen Endprodukten zugeordnet werden können, um in der Datenaufbereitungsphase von einer Verschraubung auf ein Produkt verweisen zu können. Hierfür wäre es beispielsweise sinnvoll, die Listenansicht um eine Suchfunktion nach Datum und IdCode zu erweitern.

Die Integration des Funk-Akkuschraubers in die Montagelinie ist ein weiterer Schritt zur Automatisierung von Produktionsprozessen in der Lernfabrik. Das gewählte Verfahren dazu kann auf weitere Werkzeuge übertragen werden, damit leistet das Projekt einen kleinen Beitrag zur vierten industriellen Revolution.

4. Danksagung

Wir möchten uns bei allen bedanken, die uns während unseres Projektes unterstützt haben.

Unser Dank geht vor allem an Tom Stähr und Constantin Hofmann und das gesamte wbk-Team, das uns in allen Arbeitsphasen mit Rat und Tat zur Seite stand.

Josephine und Dr. Hans-Werner Hector danken wir dafür, dass sie uns die Teilnahme am Hector-Seminar ermöglicht haben und uns somit die Möglichkeit gegeben haben, wissenschaftliches Arbeiten aus allen Perspektiven zu erleben.

Schlussendlich bedanken wir uns auch noch ganz herzlich bei unseren Kursleitern Anke Richert und Dietmar Gruber sowie Paul Bischof für die tolle und ausdauernde Unterstützung.

5. Quellen

Bosch-Rexroth Ltd, 2015. *Twitter.* [Online]
Available at: https://twitter.com/boschrexrothuk/status/596353786676903938
[Zugriff am 24.07.2019].

Pokuri, J., 2017. *spring 4 rest hibernate crud example.* [Online]
Available at: http://www.javasavvy.com/spring-4-rest-hibernate-crud-example/
[Zugriff am 22.07.2019].

Veits, O., 2017. *Consuming a REST API with Angular 6 - HttpClientModule.* [Online]
Available at: https://vocon-it.com/2017/06/24/consuming-a-restful-web-service-with-angular/
[Zugriff am 30.07.2019].

Im Projekt wurden folgende Programme, Frameworks und Erweiterungen verwendet:

- mongoDB - https://www.mongodb.com
- Entwicklungsumgebung Intellij IDEA - https://www.jetbrains.com/idea/
- Java Version 8 Build 201 - https://www.java.com
- Spring Boot - https://spring.io/projects/spring-boot
- Angular - https://angular.io/
- Chart.js - https://www.chartjs.org/

6. Anhang

6.1 Abbildungen

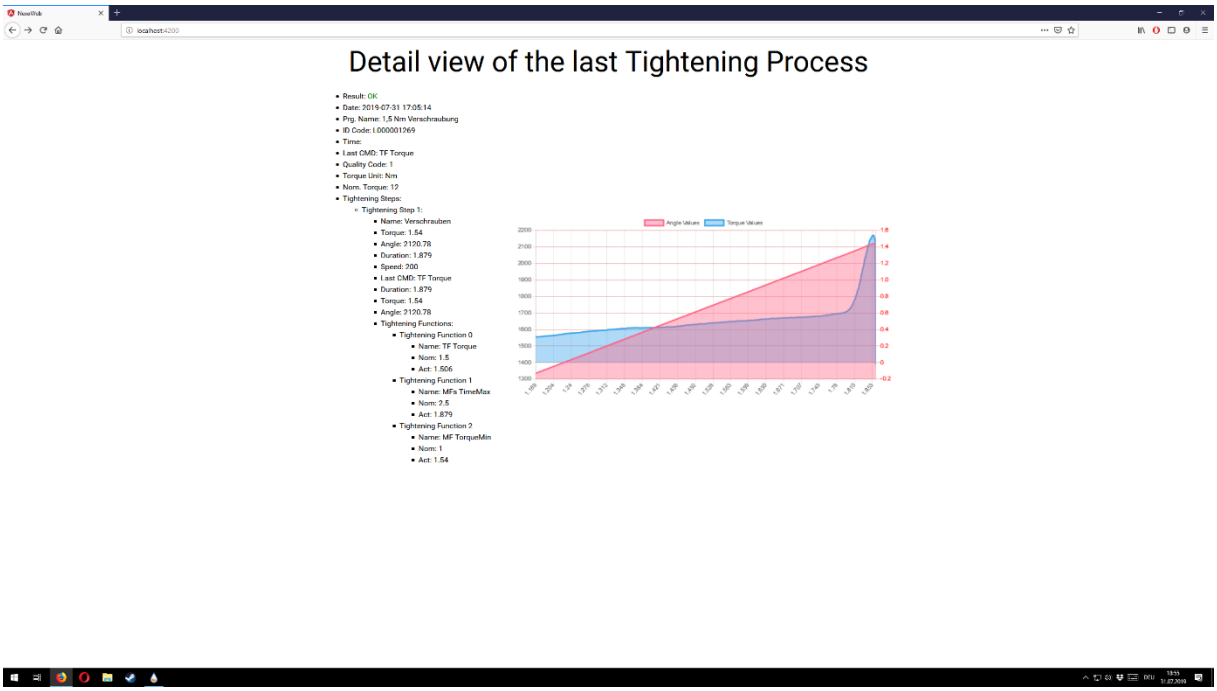

Abb. 14 Detailansicht der letzten Verschraubung

6.2 Quellcode

6.2.1 Backend

6.2.1.1 NexoController

```
package com.nexo.server.Controller;

import com.nexo.server.Entity.TighteningProcess;
import com.nexo.server.Service.NexoService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@CrossOrigin
public class NexoController {

    @Autowired
    private NexoService nexoService;

    @RequestMapping(method = RequestMethod.GET, value = "/nexo")
    public List<TighteningProcess> getXResults(@RequestParam(defaultValue
= "0") int index) {
        return nexoService.getXResults(index);
    }

    @RequestMapping(method = RequestMethod.GET, value = "/nexo/{idCode}")
    public TighteningProcess getByIdCode(@PathVariable String idCode) {
        return nexoService.getByIdCode(idCode);
    }

    @RequestMapping(method = RequestMethod.GET, value = "/nexoAll")
    public List<TighteningProcess> getAll() {
        return nexoService.getAll();
    }

    @RequestMapping(method = RequestMethod.GET, value = "/nexoLast")
    public TighteningProcess getLastProcess() { return
nexoService.getLastProcess(); }

    @RequestMapping(
            method = RequestMethod.POST,
            value = "/nexo",
            consumes = MediaType.APPLICATION_JSON_VALUE)
    public void nexo(@RequestBody TighteningProcess tighteningProcess) {
        nexoService.insertResult(tighteningProcess);
    }

}
```

6.2.1.2 NexoService

```
package com.nexo.server.Service;

import com.nexo.server.DAO.NexoDAO;
import com.nexo.server.Entity.TighteningProcess;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.*;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.List;

@Service
public class NexoService {

    @Autowired
    private NexoDAO nexoDAO;

    public void insertResult(TighteningProcess tighteningProcess) {
        nexoDAO.insert(tighteningProcess);
    }

    public List<TighteningProcess> getXResults(int index) {
        Slice<TighteningProcess> slice;
        Pageable pageable = PageRequest.of(index * 10, 1,
Sort.Direction.DESC, "date");

        List<TighteningProcess> content = new ArrayList<>();

        for (int i = 0; i < 10; i++) {
            slice = nexoDAO.findAll(pageable);
            if (!slice.hasContent()) {
                return content;
            }
            content.add(slice.getContent().get(0));
            if (slice.isLast()) {
                return content;
            }
            pageable = slice.nextPageable();
        }
        return content;
    }


    public TighteningProcess getLastProcess() {
        Pageable pageable = PageRequest.of(0, 1, Sort.Direction.DESC,
"date");
        Page<TighteningProcess> processPage = nexoDAO.findAll(pageable);

        return processPage.getContent().get(0);
    }

    public List<TighteningProcess> getAll() {
        return nexoDAO.findAll();
    }

    public TighteningProcess getByIdCode(String idCode) {
        return nexoDAO.findByIdCode(idCode);
    }
}
```

6.2.1.3 NexoDAO

```
package com.nexo.server.DAO;

import com.nexo.server.Entity.TighteningProcess;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface NexoDAO extends MongoRepository<TighteningProcess,
String> {

    TighteningProcess findByIdCode(String idCode);
}
```

6.2.1.4 TighteningProcess

```java
package com.nexo.server.Entity;

import com.fasterxml.jackson.annotation.JsonProperty;

import java.util.ArrayList;
import java.util.List;

/**
 * TighteningProcess
 */
public class TighteningProcess {

    private int nr;
    private String result;
    private String channel;
    @JsonProperty(value = "prg nr")
    private int prgNr;
    @JsonProperty(value = "prg name")
    private String prgName;
    @JsonProperty(value = "prg date")
    private String prgDate;
    private int cycle;
    @JsonProperty(value = "nominal torque")
    private double nominalTorque;
    private String date;
    @JsonProperty(value = "id code")
    private String idCode;
    @JsonProperty(value = "torque unit")
    private String torqueUnit;
    @JsonProperty(value = "last cmd")
    private String lastCmd;

    [...]

    @JsonProperty(value = "tightening steps")
    private List<TighteningStep> tighteningSteps = new ArrayList<>();

    // GETTERS AND SETTERS
    public int getNr() {
        return nr;
    }

    public void setNr(int nr) {
        this.nr = nr;
    }

    [...]

}
```

6.2.2 Frontend

6.2.2.1 Nexo.service.ts

```
import {Injectable} from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {TighteningProcess} from './Entities/tightening-process';

const BASE_URL = 'http://localhost:8080/nexo?index=';

@Injectable({
  providedIn: 'root'
})
export class NexoService {

  public processes: TighteningProcess[] = [];

  constructor(private http: HttpClient) {
  }

  getLastData() {
    return this.http.get('http://localhost:8080/nexoLast');
  }

  getData(index) {
    const url = BASE_URL.concat(index);
    return this.http.get(url);
  }

  addData(t: TighteningProcess) {
    this.processes.push(t);
  }

  getByIdCode(idCode: string) {
    return this.http.get('http://localhost:8080/nexo/'.concat(idCode));
  }
}
```

6.2.2.2 Multiview.component.html

```
<div>
  <table [dataSource]="dataSource" mat-table multiTemplateDataRows>
    <!-- Result Column -->
    <ng-container matColumnDef="result">
      <th *matHeaderCellDef mat-header-cell> Result</th>
      <td *matCellDef="let process" mat-cell>
        <span [ngClass]="{'color-red': process.result === 'NOK', 'color-
green': process.result === 'OK'}">
        {{process.result}}
      </span>
      </td>
    </ng-container>

    <!-- ID Column -->
    <ng-container matColumnDef="idCode">
      <th *matHeaderCellDef mat-header-cell> ID</th>
      <td *matCellDef="let process" mat-cell> {{process.idCode}} </td>
    </ng-container>

    <!-- Name Column -->
    <ng-container matColumnDef="prgName">
      <th *matHeaderCellDef mat-header-cell> prgName</th>
      <td *matCellDef="let process" mat-cell> {{process.prgName}} </td>
    </ng-container>

    <!-- Date Column -->
    <ng-container matColumnDef="date">
      <th *matHeaderCellDef mat-header-cell> Date</th>
      <td *matCellDef="let process" mat-cell> {{process.date}} </td>
    </ng-container>


    <tr *matHeaderRowDef="displayedColumns" mat-header-row></tr>
    <tr (click)="navigateTo(row)" *matRowDef="let row; columns:
displayedColumns" class="row-hover" mat-row></tr>
  </table>

  <button (click)="loadMore()" mat-raised-button>Load More</button>
</div>
```

6.2.2.3 Multiview.component.ts

```
import {AfterViewInit, Component, OnInit, ViewChild} from
'@angular/core';
import {NexoService} from '../nexo.service';
import {TighteningProcess} from '../Entities/tightening-process';
import {MatSnackBar, MatTable} from '@angular/material';
import {Router} from '@angular/router';


export interface DataInterface {
  result: string;
  prgName: string;
  date: string;
  idCode: string;
}

@Component({
  selector: 'app-multiview',
  templateUrl: './multiview.component.html',
  styleUrls: ['./multiview.component.css'],
})
export class MultiviewComponent implements OnInit {

  displayedColumns: string[] = ['result', 'idCode', 'prgName', 'date'];

  @ViewChild(MatTable, {static: false}) table: MatTable<any>;

  constructor(private nexoService: NexoService, private snackBar:
MatSnackBar, private router: Router) {
  }
  dataSource: DataInterface[] = [];
  public processes: TighteningProcess[] = [];
  dataIndex = 0;

  ngOnInit() {
    this.data();
  }
```

```
  data() {
    this.nexoService.getData(this.dataIndex)
      .subscribe((data: []) => {
        if (data.length === 0) {
          const snackBarRef = this.snackBar.open('No more Data
available');
        }
        // @ts-ignore
        for (const d of data) {
          const tempProcess = new TighteningProcess(d);
          this.nexoService.addData(tempProcess);
          this.dataSource.push({
            result: tempProcess.result,
            prgName: tempProcess.prgName,
            date: tempProcess.date,
            idCode: tempProcess.idCode
          });
          this.processes.push(tempProcess);
        }
        this.table.renderRows();
      });
  }

  loadMore() {
    this.dataIndex++;
    this.data();
  }

  navigateTo(row: DataInterface) {
    this.router.navigate(['process', row.idCode]);
  }

}
```

6.2.2.4 Detailview.component.html

```
<div id="headline">Detail view of the last Tightening Process</div>
<div id="content">
  <ul>
    <li>Result:
      <span [ngClass]="{'color-red': tiProcess.result === 'NOK', 'color-
green': tiProcess.result === 'OK'}">{{tiProcess.result}}      </span>
    </li>
    <li>Date: {{tiProcess.date}}</li>
    <li>Prg. Name: {{tiProcess.prgName}}</li>
    <li>ID Code: {{tiProcess.idCode}}</li>
    <li>Time: {{tiProcess.totalTime}}</li>
    <li>Last CMD: {{tiProcess.lastCmd}}</li>
    <li>Quality Code: {{tiProcess.qualityCode}}</li>
    <li>Torque Unit: {{tiProcess.torqueUnit}}</li>
    <li>Nom. Torque: {{tiProcess.nominalTorque}}</li>
    <li>Tightening Steps:
      <ul>
        <li *ngFor="let t of tiProcess.tighteningSteps; let i = index">
          Tightening Step {{i + 1}}:
          <div style="position: relative; overflow: hidden">
            <ul class="listing">
              <li>Name: {{t.name}}</li>
              <li>Torque: {{t.torque}}</li>
              <li>Angle: {{t.angle}}</li>
              <li>Duration: {{t.duration}}</li>
              <li>Speed: {{t.speed}}</li>
              <li>Last CMD: {{t.lastCmd}}</li>
              <li>Duration: {{t.duration}}</li>
              <li>Torque: {{t.torque}}</li>
              <li>Angle: {{t.angle}}</li>
              <li>Tightening Functions:
                <ul>
                  <li *ngFor="let f of t.tighteningFunctions; let j =
index">
                    Tightening Function {{j}}
                    <ul>
                      <li>Name: {{f.name}}</li>
                      <li>Nom: {{f.nom}}</li>
                      <li>Act: {{f.act}}</li>
                    </ul>
                  </li>
                </ul>
              </li>
            </ul>
            <div class="chart">
              <canvas id="{{i}}" baseChart
                      [datasets]="lineChartData"
                      [labels]="lineChartLabels"
                      [options]="lineChartOptions"
                      [legend]="lineChartLegend"
                      [chartType]="lineChartType">
              </canvas>
            </div>
          </div>       </li>   </ul>   </ul>   </div>
```

6.2.2.5 Detailview.component.ts

```
import {AfterViewInit, Component, OnInit, QueryList, ViewChildren} from
'@angular/core';
import {ActivatedRoute, Router} from '@angular/router';
import {TighteningProcess} from '../Entities/tightening-process';
import {NexoService} from '../nexo.service';
import {ChartOptions} from 'chart.js';
import {BaseChartDirective} from 'ng2-charts';
import {MatSnackBar} from '@angular/material';

@Component({
  selector: 'app-detailview',
  templateUrl: './detailview.component.html',
  styleUrls: ['./detailview.component.css']
})
export class DetailviewComponent implements OnInit, AfterViewInit {
  public lineChartData = [
    {data: [], label: 'Angle Values', yAxisID: 'A'},
    {data: [], label: 'Torque Values', yAxisID: 'B'}
  ];
  public lineChartLabels: number[] = [];
  public lineChartOptions: (ChartOptions & { annotation: any }) = {
    responsive: true,
    elements: {
      point: { radius: 0 }
    },
    scales: {
      xAxes: [{
        ticks: {
          maxTicksLimit: 20,
        }
      }],
      yAxes: [
        {
          id: 'A',
          position: 'left',
        },
        {
          id: 'B',
          position: 'right',
          gridLines: {
            color: 'rgba(255,0,0,.3)',
          },
          ticks: {
            fontColor: 'red',
          }
        }
      ]
    },
    annotation: {
      annotations: [
        {
          type: 'line',
          mode: 'vertical',
          scaleID: 'x-axis-number,',
          value: 'March',
          borderColor: 'orange',
          borderWidth: 2,
```

```
          label: {
            enabled: true,
            fontColor: 'orange',
            content: 'LineAnno'
          },
        },      ],    },  };
  public lineChartLegend = true;
  public lineChartType = 'line';

  @ViewChildren(BaseChartDirective) c: QueryList<any>;

  processID: string = null;
  public tiProcess: TighteningProcess;
  constructor(private route: ActivatedRoute, private nexoService:
NexoService, private snackBar: MatSnackBar, private router: Router) {
  }
  ngOnInit() {
    this.route.params.forEach((urlParameters) => {
      this.processID = urlParameters['idCode'];
    });
    this.tiProcess = this.nexoService.processes.find(p => p.idCode ===
this.processID);
    if (this.tiProcess === undefined) {
      this.nexoService.getByIdCode(this.processID)
        .subscribe((data: TighteningProcess) => {
          if (data === null) {
            this.snackBar.open('No Process with this ID found!
Redirecting to multiview...');
            setTimeout(() => {
              this.router.navigate(['multi']);
            }, 3000);
            this.snackBar.dismiss();
          }
          this.tiProcess = new TighteningProcess(data);
        });
    }
  }
  ngAfterViewInit() {
    const d: BaseChartDirective[] = this.c.toArray();
    const lineChartDatas = [];

    for (let i = 0; i < this.tiProcess.tighteningSteps.length; i++) {

      lineChartDatas[i] = [];
      lineChartDatas[i][0] = {...this.lineChartData[0]};
      lineChartDatas[i][1] = {...this.lineChartData[1]};
      d[i].chart.data.datasets = lineChartDatas[i];
      d[i].chart.update();
      d[i].chart.data.datasets[0].data =
this.tiProcess.tighteningSteps[i].graph.angleValues;
      d[i].chart.data.datasets[1].data =
this.tiProcess.tighteningSteps[i].graph.torqueValues;
      // @ts-ignore
      d[i].chart.data.labels =
this.tiProcess.tighteningSteps[i].graph.timeValues;
      d[i].chart.update();
    }
  }
}
```

6.2.2.6 Tightening-process.ts

```
import {TighteningStep} from './tightening-step';

export class TighteningProcess {
  public nr: number;
  public result: string;
  public channel: string;
  public cycle: number;
  public date: string;
  public prgNr: number;
  public prgName: string;
  public prgDate: string;
  public nominalTorque: number;
  public idCode: string;
  public torqueUnit: string;
  public lastCmd: string;
  public qualityCode: string;
  public totalTime: number;
  public toolSerial: string;
  public reworkCode: number;
  public reworkText: string;
  public cellId: string;
  public jobNr: number;
  public MCEFactor: number;
  public batchNr: string;
  public batchCanceled: number;
  public batchDirectionOK: number;
  public batchDirectionNOK: number;
  public batchMaxOK: number;
  public batchMaxNOK: number;
  public batchOK: number;
  public batchNOK: number;
  public tighteningSteps: TighteningStep[];

constructor(data) {
    this.nr = data['nr'];
    this.result = data['result'];
    this.channel = data['channel'];
    this.cycle = data['cycle'];
    this.date = data['date'];
    this.prgNr = data['prg nr'];
    this.prgName = data['prg name'];
    this.prgDate = data['prg date'];
    this.nominalTorque = data['nominal torque'];
    this.idCode = data['id code'];
    this.torqueUnit = data['torque unit'];
    this.lastCmd = data['last cmd'];

    [...]

    for (let i = 0; i < data['tightening steps'].length; i++) {
      this.tighteningSteps[i] = new TighteningStep(
        data['tightening steps'][i]);
    }
  }
}
```

Hinweis:

Automatisch generierte Dateien sind hier nicht gelistet, diese sind allerdings u. U. für ein funktionstüchtiges Programm notwendig!

Der Kürze der Abhandlung wegen wurden einige Dateien gekürzt, dies wurde entsprechend gekennzeichnet „[...]“. Nachfolgende Dateien, die für das grundlegende Verständnis der Arbeit nicht relevant sind, wurden nicht näher aufgeführt. Sie liegen der Arbeit in digitaler Form bei.

Backend:

- Main
- TightteningStep
- TighteningFunction
- Graph
- Build.gradle

Frontend:

- Index.html
- Styles.css
- App.component.html
- App.component.ts
- App.module.ts
- App-routing.module.ts
- Multiview.component.css
- Linechart.component.html
- Linechart.component.css
- Linechart.component.ts
- Detailview.component.css
- Tightening-Step.ts
- Tightening-Function.ts
- Graph.ts

Selbstständigkeitserklärung

Hiermit versichern wir, dass wir diese Arbeit unter der Beratung durch Tom Stähr und Constantin Hofmann selbstständig verfasst haben und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht haben.

Ort, Datum

Johannes Huber

Milena Seeburger