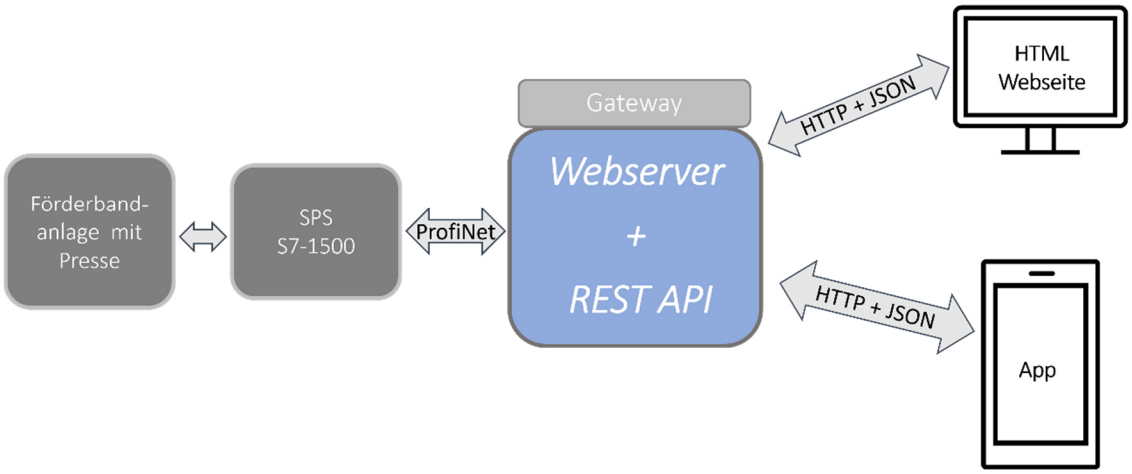
FRITZ-HOPF-
TECHNIKERSCHULE
NÖRDLINGEN

Fachbereich Elektrotechnik



Bereitstellen eines Webservers mit JSON REST API zur Erfassung steuerungstechnischer Parameter einer S7-1500 mittels IIOT-Gateway

vorgelegt von:

Kißlinger Thomas

Schmutterer Christian

(Klasse TE_2 – Schuljahr 2021/2022)

am 17.01.2022

I) Worum geht es genau?

In dieser Arbeit wird eine bestehende Maschinenanlage, die über eine Siemens-SPS gesteuert wird, in einem normalen PC-Netzwerk erreichbar gemacht. Damit ergibt sich die Möglichkeit, mit einem netzwerkfähigen Gerät (bspw. Laptop oder Smartphone) sämtliche Anlagenparameter ortsunabhängig auszulesen und zu verändern, also die Anlage auch fernzusteuern.

Realisiert wird das Ganze über das Einpflegen eines sog. „IIoT-Gateway“ (Industrial-Internet-of-Things-Gateway) über ProfiNet an die SPS. Dieses Gateway fungiert als Schnittstelle zwischen der SPS-Seite und dem Internet und benutzt für die Kommunikation bzw. den Datenaustausch das JSON-Format (Java Script Object Notation).

Als Anwendungsbeispiele wurden weiterhin eine eigene Webseite mit Fokus auf der übersichtlichen Darstellung der JSON-Zeichenketten und eine Smartphone-App mit Fokus auf der Visualisierung der Anlage erstellt. Beide ermöglichen das Auslesen und Steuern der Anlage.

II) Versuchsanlagenaufbau

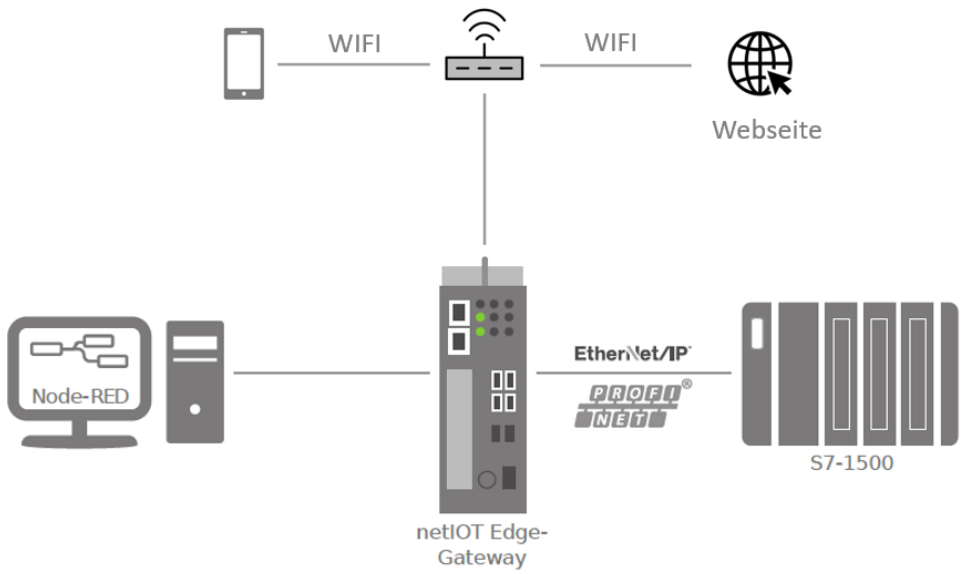


Abbildung 1: schematischer Versuchsaufbau

1.1) Elabo Förderband

Als Grundlage und Modell für eine reale Industriemaschine wird ein Förderband vom Lehrmittelhersteller ETS DIDACTIC GMBH angesteuert. Die Anlage setzt sich aus zwei Hauptbestandteilen zusammen. Zum einen ist in der Mitte des Bandes eine Press-Station mit zwei doppelwirkenden Zylindern befestigt. Ein Zylinder dient zum Schließen der Dose und ein anderer zum Öffnen der Dose. Der zweite Hauptbestandteil ist ein 24 Volt DC Motor, der zwei gespannte Gummibänder antreibt und es damit ermöglicht die Dosen auf ihrem Werkzeugschlitten hin und her zu bewegen. Zusätzlich sind zur Positionsüberwachung an den Zylindern Reed-Kontakte verbaut. An den „Stopper"-Zylindern jeweils einer, an den doppelwirkenden Zylindern der Press-Station jeweils zwei.

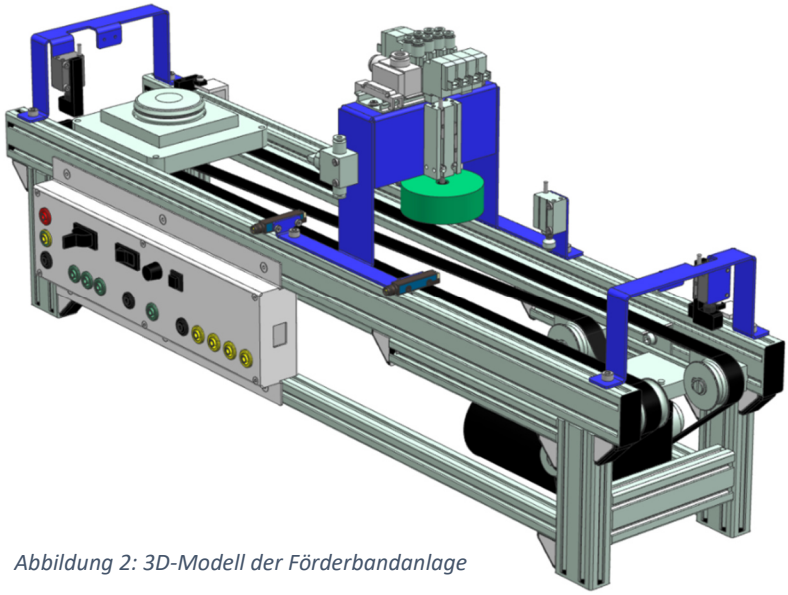


Abbildung 2: 3D-Modell der Förderbandanlage

1.2) IIOT-Gateway

Unter dem Begriff IoT (Internet of Things) versteht man den Informationsaustausch zwischen physischen Objekten untereinander und dem Benutzer über das Internet. IIoT (Industrial Internet of Things) ist für den Datenaustausch und Datenerfassung im produzierenden und industriellen Umfeld gedacht, um die Wertschöpfungskette zu optimieren. Diese Aufgabe übernimmt als physisches Gerät das Gateway. In den allermeisten Fällen ist dies ein PC mit mehreren Netzwerk-schnittstellen, der als Server fungiert. (z.B. ein Raspberry Pi).

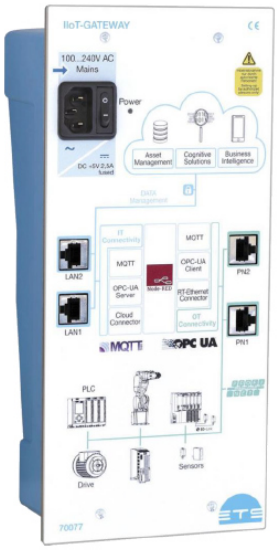


Abbildung 3: IIoT Gateway

III) Aufbau und Funktion eines JSON Strings

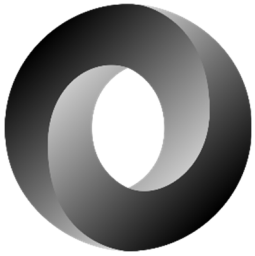


Abbildung 4: JSON Logo

JSON ist ein Datenaustauschformat, dass es auf einfache und lesbare Weise schafft, Daten darzustellen. Da viele Notationsregeln von JSON auf den bekannten Programmiersprachen wie C oder auch Java aufbauen, ist es für fast alle Programmiersprachen sehr leicht den Inhalt zu lesen und daraus verwertbare Daten wieder abzuleiten (sog. Parsen).

Beim Aufbau gibt es grundsätzlich zwei Möglichkeiten:

- Name (sog. „keys“) und in Kombination ein Wert. Getrennt durch einen Doppelpunkt.

  z.B. `'{"age" : 18}'`

oder

- eine Liste bzw. ein Array an Werten

  z.B. `'["raspberry", "strawberry", 42, "blackberry"]'`

Dabei muss der „key“ immer ein String bzw. eine Zeichenkette sein. Der Wert kann ein Boolean, Zahlenwert, String, Array, Objekt oder ein Null-Pointer sein. Paare werden durch ein Komma getrennt. Hierbei gilt es zu beachten, dass wirklich nur Werte und keine Funktionen übergeben werden dürfen. Des Weiteren sind auch Kombinationen oder Verschachtelungen denkbar:

z.B.
```
'{   "name" : "Hans",
     "age"    :  26 ,
     "favourite movies" :   [
                "Lord of the Rings: The Return of the King",
                "Avatar"
     ]
}'
```

Ein gutes, praxisnahes Beispiel sind unter anderem Lesezeichen aus dem Firefox-Browser, welche beim Exportieren als .json-Datei gespeichert werden. Öffnet man diese versuchsweise mit dem Windows-Editor, so erhält man die gesamten, eigenen Lesezeichen in der typischen JSON-Notation.

Dieses Format ist neben „.xml“ das am weitesten verbreitete im Internet und wird auch im Folgenden verwendet, um den Datenaustausch der Anlage zu realisieren.

IV) Programmaufbau & Kommunikation

1) Verwendete Software

TIA-Portal	Programmierung der Ablauffolge auf dem Förderband
Node-Red	Programmierung des IIoT-Gateways und Verarbeitung der JSON-Zeichenketten
Android Studio	Programmierung des Smartphone-App
Inkscape	Erstellung von sogenannten Vektorgrafiken zur Visualisierung

2) SPS-Programm

Die Förderbandanlage kann automatisch kleine Metalldosen öffnen und schließen und besitzt einen dritten Modus für den Handbetrieb. Für die Realisierung eines solchen Modus wird ein Funktionsbaustein (FB) benutzt. Durch die drei benötigten Modi werden im Organisationsbaustein (OB) also auch drei verschiedene Funktionsbausteine erstellt, um die verschiedenen Funktionen voneinander zu trennen.

Um die Daten zum Gateway zu senden, werden die überwachten Ein- oder Ausgänge auf das Ausgangsbyte 12 bis 43 weitergeleitet. In Abbildung 3 wird beispielsweise der Sensor B1 am Eingang 16.4 vom Förderband überwacht. Wenn der Eingang „HIGH“ wird, wird das Signal an den Ausgang A16.2 geschickt. Das Eingangssignal wird also dupliziert und über die Profinet-Verbindung ebenfalls dem IIot-Gateway zugesendet.

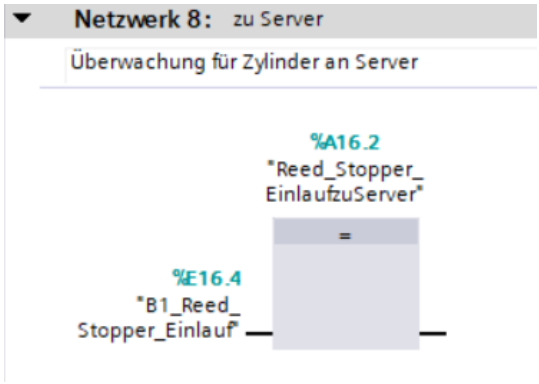


Abbildung 5: Weiterleiten der Sensorzustände

Um Daten vom IIot-Gateway zu empfangen, werden die Eingangsbytes 20-51 verwendet. Die benötigten Eingangsbytes werden über Profinet mit den vorher definierten Bytes vom Gateway synchronisiert und ermöglichen dadurch die Kommunikation. Die über Profinet erhaltenen Signalzustände können dann im weiteren Verlauf genauso verwendet werden, als ob sie über die digitalen Eingänge der SPS kommen würden. Innerhalb des FB-Bausteines werden die Eingänge wieder auf die verwendeten Ausgänge weitergeleitet.

3) Node-RED

Der Programmaufbau dieses Projekts in Node-RED (Roter Knoten) kann grundsätzlich in 2 Teile unterschieden werden:

- Bandsteuerung und Statusausgabe
- Ausgabe html-Webseite

Da viele unterschiedliche Knoten mitunter auf die gleiche Information zugreifen, wird der Status der Sensoren des Förderbands in sogenannte „flow“ -Variablen gespeichert. Diese sind von überall innerhalb eines „flows“ aus zugreifbar und können z.B. mit Instanzvariablen in Java verglichen werden.

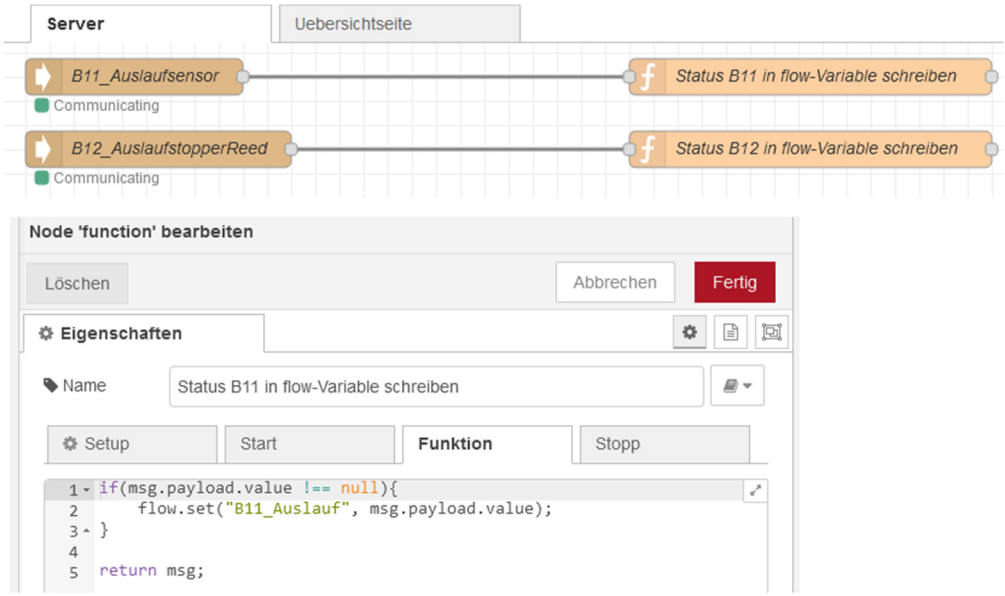


Abbildung 6: Speichern von Feldbus-Werten in „flow“-Variablen

Für die Ausgabe greift man auf diese „flow“-Variablen wieder lesend zu und verpackt das Ganze leserlich und geordnet in einer JSON-Zeichenkette. Laut unserer eigens definierten API-Tabelle wird ein Eingangsknoten unter dem Adresszusatz „*api/sensoren*“ zur Verfügung gestellt, welcher als Rückgabewert den genannten JSON-String wieder ausgibt. D.h. jeglicher http-Request an die URL (Uniform Resource Locator) des Gateways plus den Adresszusatz, der in der API definiert ist, bekommt als Antwort eine JSON-Zeichenkette mit den aktuellen Sensorwerten zurück.

Die anderen GET-Methoden funktionieren dementsprechend auf ähnliche Weise, aber mit anderen Rückgabewerten.

Für die Steuerung des Förderbands über das Gateway werden die POST-Methoden benutzt. Diese erwarten unter der URL des Gateways und dem Adresszusatz einen zusätzlichen JSON-String und setzen dann die entsprechende Variable im Feldbusknoten ebenfalls auf den übermittelten Wert. (z.B. '{"value" : 1}' an die Adresse < *http://192.168.178.49:1880/api/motor/hand/rechts* >)

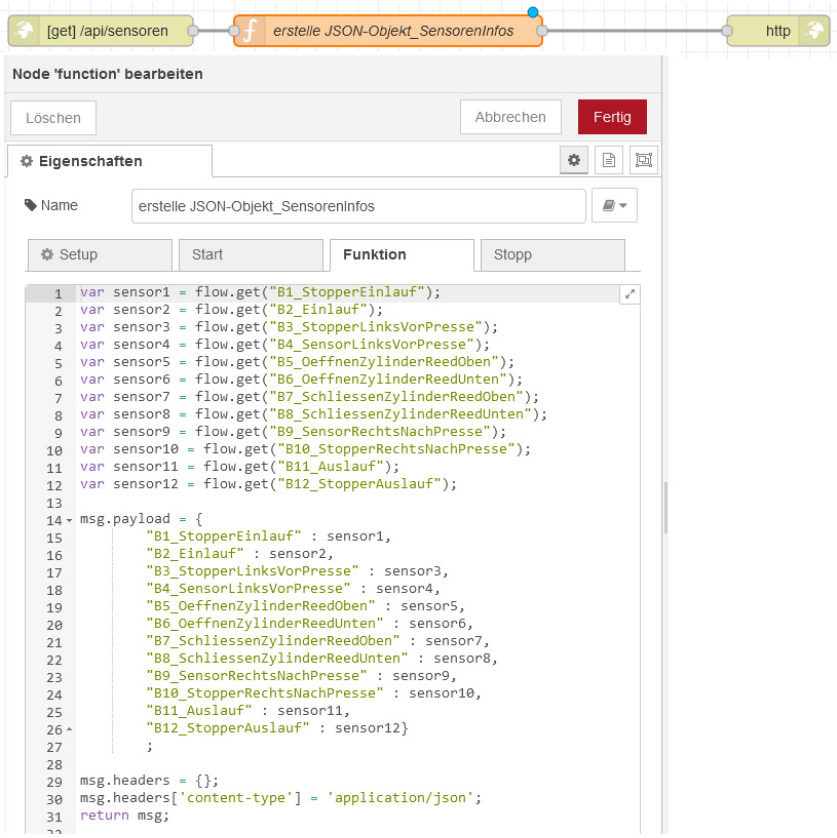


Abbildung 7: Erstellen des JSON-Strings für den Sensorstatus

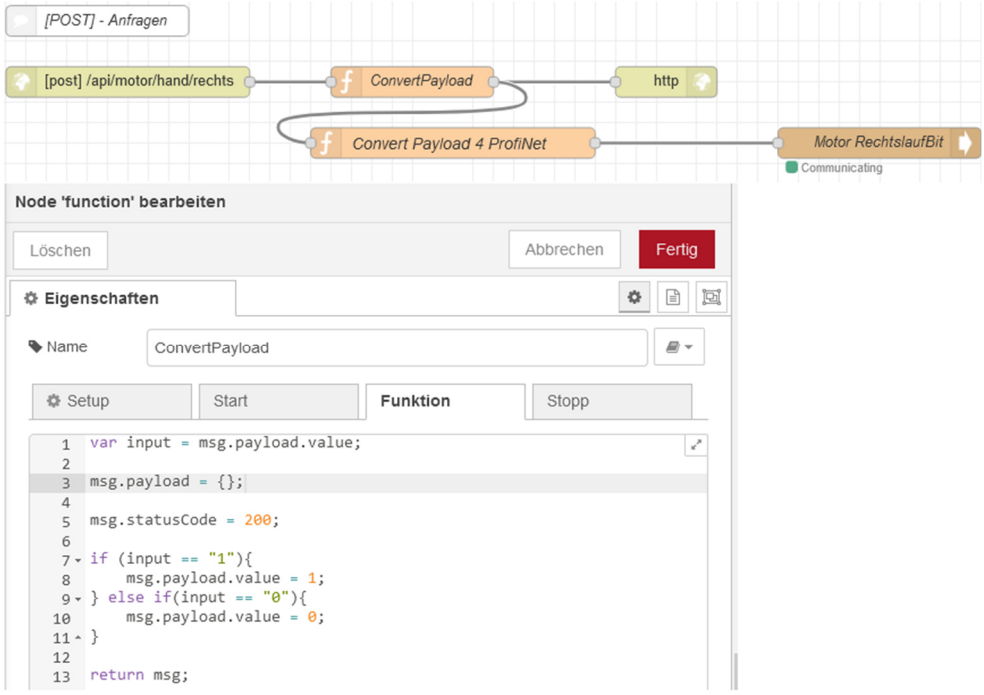


Abbildung 8: POST-Anfrage für Motor Rechtslauf

In Abbildung 8 wird beispielsweise die Antwort einmal zurück an den Client und ebenfalls an den entsprechenden Profinet-Knoten gesendet. Bei bestehender Verbindung wird der Motor also in Rechtslauf geschaltet.

4) html-Webseite

Als Anwendungsbeispiel wurde zusätzlich eine einfache Webseite erstellt, die über verschiedene Buttons alle Request-Methoden implementiert.

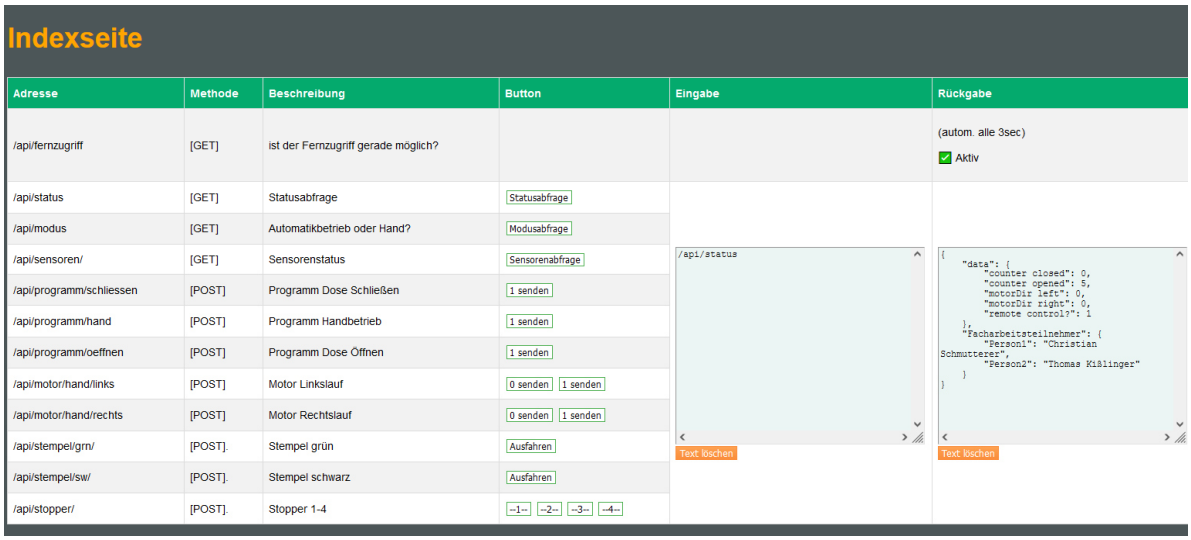


Abbildung 9: Bildschirmfoto der html-Webseite

Die Webseite für dieses Projekt beinhaltet aus Gründen der Übersichtlichkeit eine einfache Tabelle, die alle Befehle von oben nach unten über verschiedene Buttons ausführen kann. Die dabei übermittelten und auch vom Host wieder zurückgesendeten JSON-Strings werden jeweils auf Textfeldern ausgegeben, die auch in die Tabelle mitintegriert sind.

Im JavaScript-Teil sollen zum einen die Benutzereingaben beim Klicken der Buttons verarbeitet werden und die Methoden zum Kommunizieren mit dem Host bereitgestellt werden.

Das Abfangen von sogenannten „click-events“ wird dabei folgendermaßen realisiert:

```
3  //###Buttons
4  var button0 = document.getElementById("btnRechtslauf1");
5      button0.addEventListener("click", function(event){
6          let url     = '/api/motor/hand/rechts';
7          let wert    = {"value":1};
8          let element1 = document.getElementById("textInput");
9          let element2 = document.getElementById("textStatus");
10         //console.log("Button geklickt")
11       doPostData(url, wert, element1, element2);
12
13 });
```

Abbildung 10: JavaScript-Code für Buttonaktion

Das heißt beim Klicken dieses Buttons wird einmalig die Funktion „doPostData()“ mit den entsprechend definierten Werten aufgerufen.

Die Funktion selbst sieht dann wie folgt aus:

```
//###Post
async function doPostData(url, data, inputElement, outputElement) {


fetch(url, {
  method: 'POST', // or 'PUT'
  headers: {
    'Content-Type': 'application/json',
  },
  //body: JSON.stringify(data),
  body: JSON.stringify(data),
  //console.log(data);
})
.then(response => response.json())
.then(back => {
  console.log('Success:', back);
  inputElement.innerHTML = JSON.stringify(data, null, 4);
  outputElement.innerHTML = JSON.stringify(back, null, 4);
})
.catch((error) => {
  console.error('Error:', error);
});

}
```

Abbildung 11: JavaScript-Code für eine POST-Request-Methode

Hierbei ist besonders die Asynchronität der benutzten fetch()-Methode erwähnenswert. Bei der normalen Programmierung wird Code normalerweise Schritt für Schritt von „oben nach unten“ abgearbeitet. In diesem Fall würde aber der Computer nach dem Senden einer Anfrage so lange blockieren bzw. warten, bis sie beantwortet wurde bzw. eine Zeitüberschreitung erfolgt ist. Um dies zu verhindern, wird bei sogenannten asynchronen Funktionen automatisch ein weiterer „Thread“ für solche Anfragen erzeugt. Der normale Programmablauf kann unterdessen weitergehen.

5) Android App

Wie schon bei der html-Webseite lässt sich auch hier zwischen Grafik und Funktionalität unterscheiden.

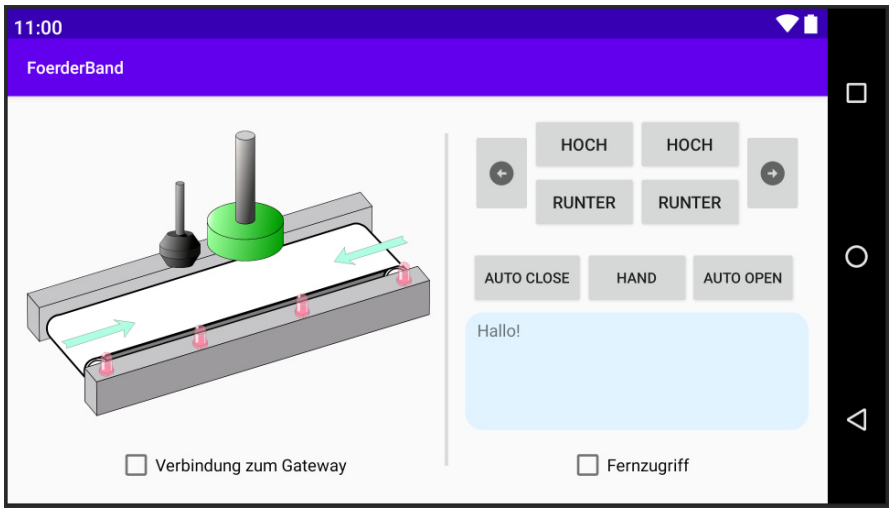


Abbildung 12: Bildschirmfoto der Smartphone-App

Beim Verfahren der Zylinder bewegen sich diese auf bzw. ab, ein aktives Förderband animiert einen Pfeil in die jeweilige Richtung, die Stopper erscheinen bei deren Betätigung und die Sensoren werden bei Erkennung eines Bauteils durch eine leuchtende LED dargestellt. Zusätzlich gibt es noch ein Textfeld für die Informationsausgabe.

Für den programmiertechnischen Part lohnt es, sich zuerst einen Überblick zu verschaffen:

```
32  public class MainActivity extends AppCompatActivity {
33
34      private final String url = "http://192.168.178.49:1880/";
35
36      private TextView txtBox;
37      private ImageView stmplOeffnenBLK;
38      private ImageView stmplSchliessenGRN;
39      private ImageView ledBandEinlauf;
40      private ImageView ledVorZyl;
41      private ImageView ledNachZyl;
42      private ImageView ledBandAuslauf;
43      private ImageView PfeilRechts;
44      private ImageView PfeilLinks;
45      private CheckBox checkBoxConnection;
46      private CheckBox checkBoxFernzugriff;
47      private RequestQueue mQueue;
48      private Context context;
49
50      private Button btn_modeClose;
51      private Button btn_modeHand;
52      private Button btn_modeOpen;
53
54      private JSONObject jsonSensors;
55      private JSONObject jsonMotor;
56      private JSONObject mode;
57
58      private boolean fernZugriff;
59
60      private ObjectAnimator animArrowLeft;
61      private ObjectAnimator animArrowRight;
62      private ObjectAnimator animStmplGrn;
63      private ObjectAnimator animStmplSw;
64
```

Abbildung 13: Überblick App-Code_Instanzvariablen

```
67      @Override
68      ////OnCreate
69      protected void onCreate(Bundle savedInstanceState) {...}
275     ////###OnCreate
276
277     ////Methoden
278     //Http[GET]-Requests mit JSON-Objekt
279     private void checkFernzugriff() throws JSONException{...}
319     private void checkSensors() throws JSONException{...}
344     private void checkMode() throws JSONException{...}
367     private void checkMotor() throws JSONException{...}
390     //###Http[GET]-Requests mit JSON-Objekt
391
392     //Http[POST]-Requests mit JSON-Objekt
393     private void bandRechtslauf(boolean OnOff) throws JSONException {...}
423     private void bandLinkslauf(boolean OnOff) throws JSONException {...}
453     private void stmplSchwrz(boolean HochRunter) throws JSONException {...}
486     private void stmplGrn(boolean HochRunter) throws JSONException {...}
517     private void selectMode(int modus) throws JSONException {...}
552     //###Http[POST]-Requests mit JSON-Objekt
553
554     //Grafik-Animationen
555     private void moveStmplGrnUpwards(ImageView v){...}
565     private void moveStmplGrnDownwards(ImageView v){...}
573     private void moveStmplSwUpwards(ImageView v){...}
583     private void moveStmplSwDownwards(ImageView v){...}
591     private void animateArrowRight(){...}
618     private void animateArrowLeft(){...}
645     //##Grafik-Animationen
646 @   private JSONObject updateData(JSONObject gotThis, JSONObject updateThat) throws JSONException{...}
668     private void updateGraphics() throws JSONException {...}
738     ////##Methoden
739 }
```

Abbildung 14: Überblick App-Code_Funktionen

In Abbildung 13 findet sich ein Überblick über alle Instanzvariablen der App. Diese werden benötigt, um die Verbindung zwischen grafischen Elementen und dem Java-Code herzustellen und um Variablen zu definieren, die von allen Methoden gleichermaßen benutzt werden können.

In Abbildung 14 finden sich alle Funktionen, die über Buttons in der App gestartet werden können bzw. automatisch beim Laden der App oder regelmäßig bei der Programmausführung benutzt werden.

6) Anmerkung

Zur besseren Veranschaulichung sind im Anhang noch kurze Videoclips, die die Fernsteuerungsmöglichkeiten und Visualisierung der Smartphone-App zeigen.