Inhalt

03.10.2024 – Protokoll 2
Motorsteuerung mit Webserver 2
09.01.2025 - Protokoll 11
Motoransteuerung über API-Requests 11
Ziel 11
Login mit Postman 11
API-Calls mit Python 11
Login-Funktion in Python 11
Funktionen durch Browse entdecken 12
Motor starten 14
Motor stoppen 15
Erste Applikation 15
Abbildungsverzeichnis 15

03.10.2024 – Protokoll

Motorsteuerung mit Webserver

Aufbau:

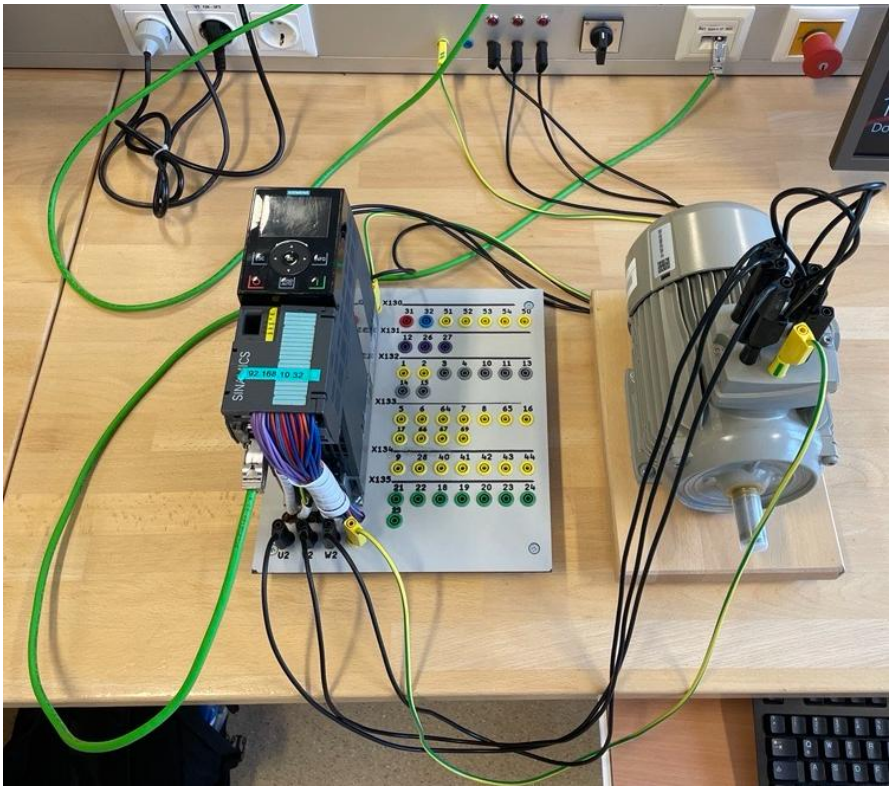

Figure 1 Aufbau

Zuerst Namen der CPU und des FU's (Frequenzumrichter) ändern → aufgrund der IPs die auf den Geräten stehen.

IP von FU eingeben und Subnetz auf 255.255.254.0 stellen

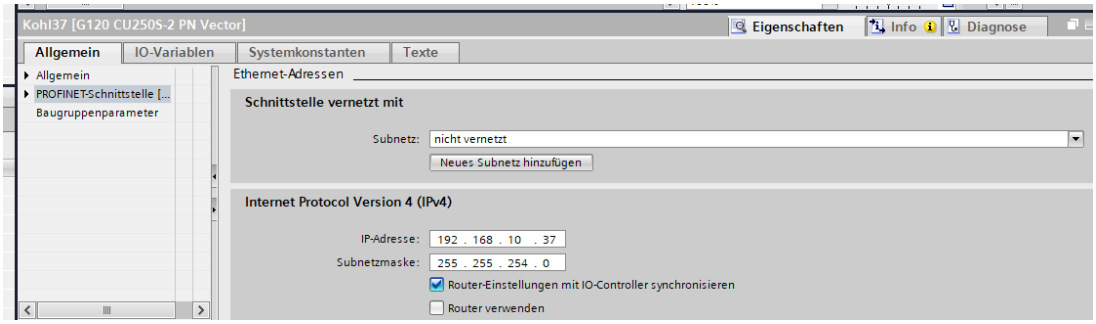

Figure 2 IP Eingabe & Subnetz

Danach auf „Online Verbinden“ klicken und hier die richtigen Schnittstellen einstellen

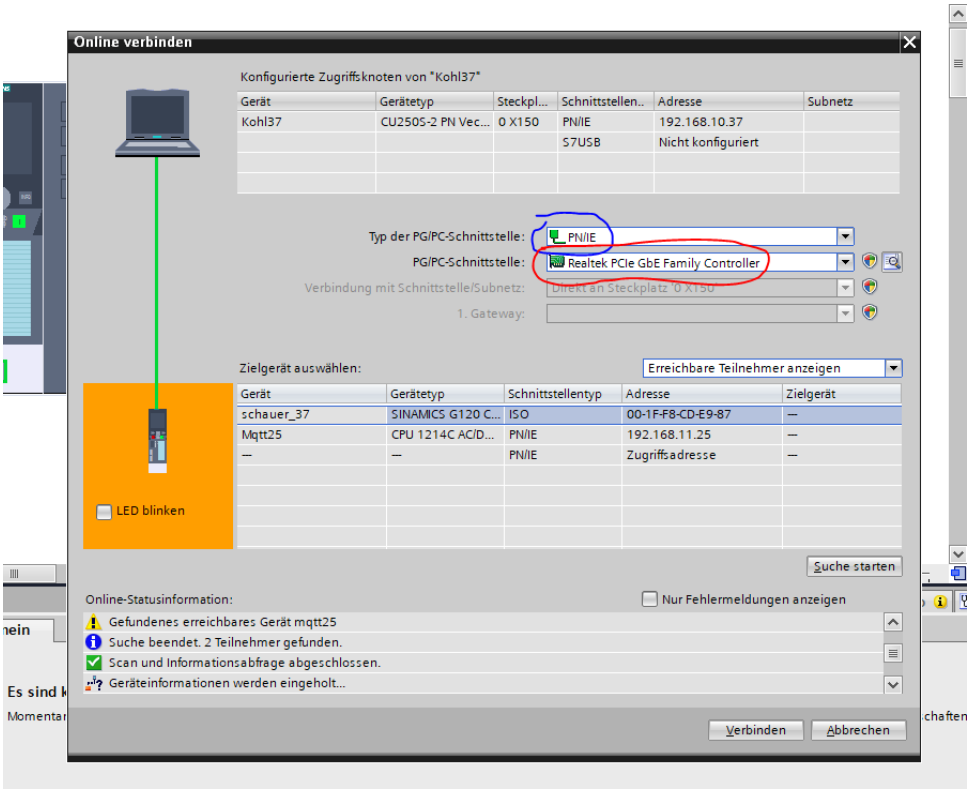

Figure 3 verbinden

- Dann auf „Suche starten“ klicken, danach auf „Verbinden“
- Soll die IP-Adresse hinzugefügt werden?: JA
- Danach Reiter „Inbetriebnahme auswählen und durchklicken bis zu den Motordaten
- Motordaten anhand von der Plakette auf dem Motor eingeben und weiter durchklicken
- Motor mit Steuertafel in Betrieb nehmen

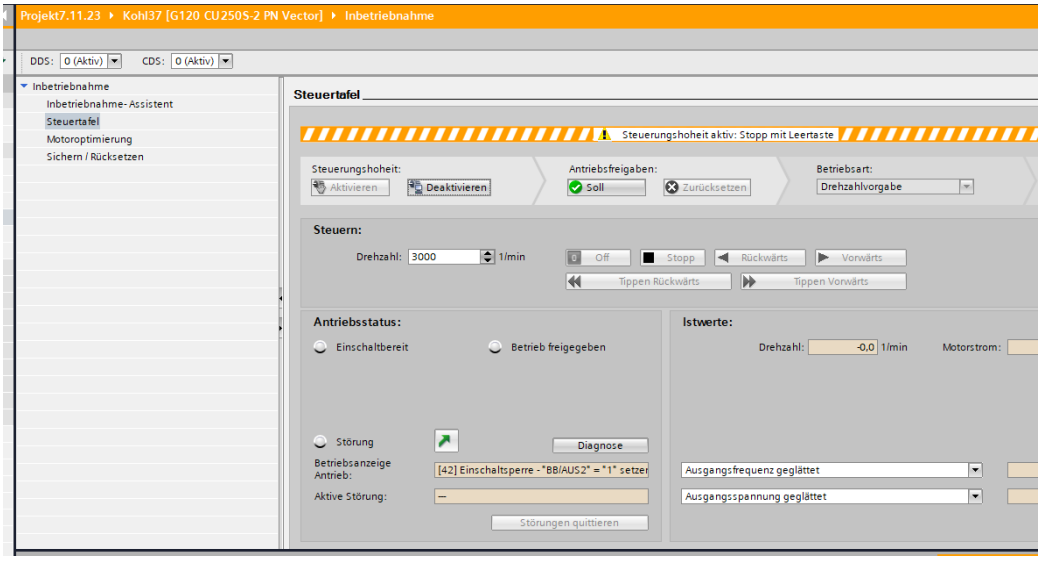

Figure 4 Motor in Betrieb

- FU mit CPUverbinden
- Rechtsklick auf FU und Gerätenamen zuweisen
- Liste aktualisieren, dann Gerät auswählen und Namen zuweisen
- Gerätekonfiguration der CPU öffnen

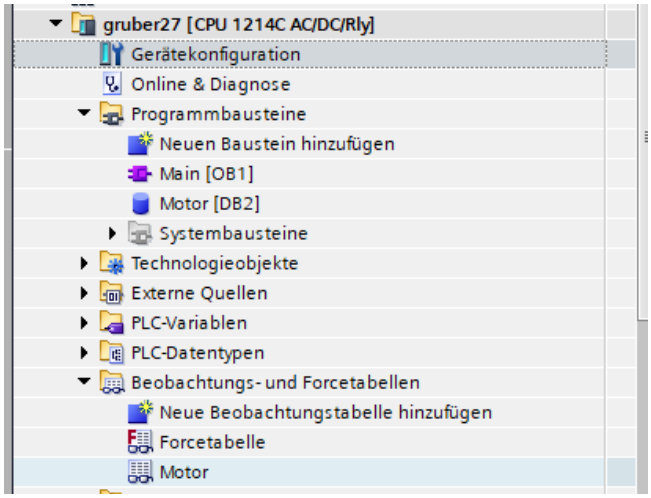

Figure 5 Gerätekonfiguration - Motor

- Neue Beobachtungstabelle „Motor“ erstellen
- Programmbausteine Motor[DB2] auswählen

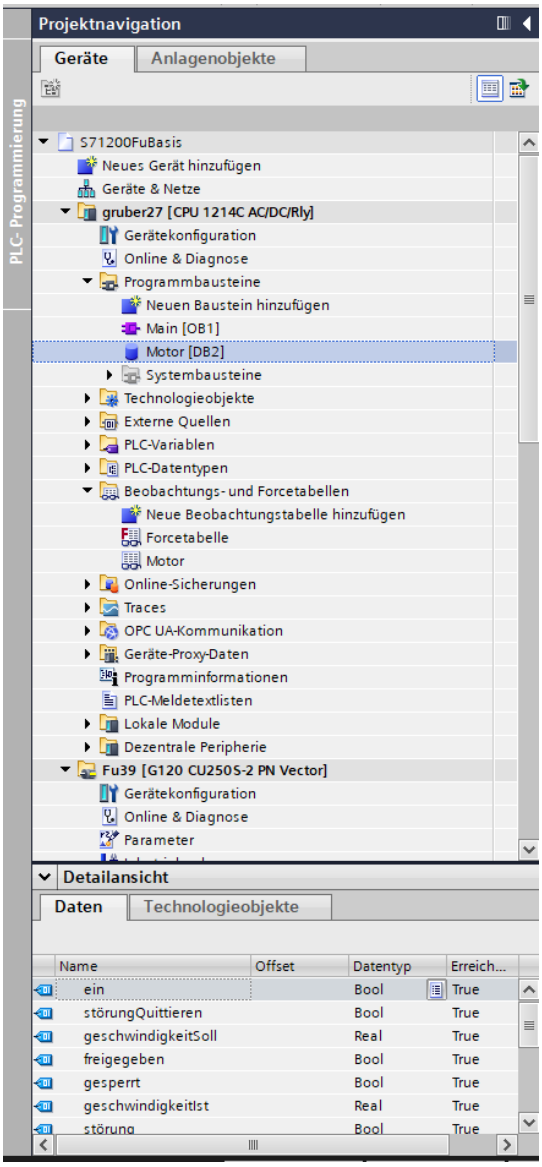

Figure 6 Programmbaustein - Motor

- Felder „**ein**“, „**störungQuittieren**“, „**geschwindigkeitSoll**“ und „**geschwindigkeitIst**“ in die Beobachtungstabelle reinziehen

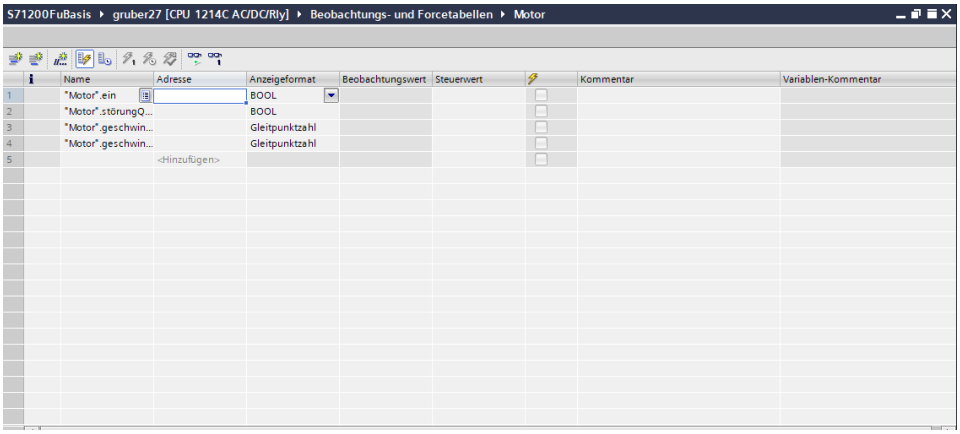

Figure 7 Beobachtungstabelle

- Über „Gerätekonfiguration“ → „Eigenschaften“ → „Webserver“ → „Allgemein“ den Webserver auf allen Modulen dieses Geräts aktivieren

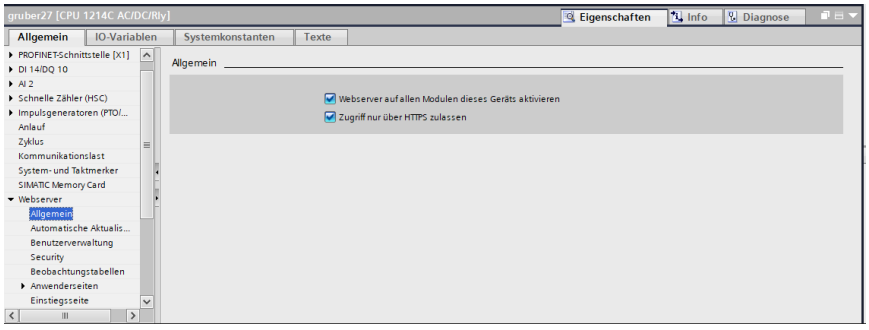

Figure 8 Webserver aktivieren

- In der Benutzerverwaltung einen neuen Benutzer mit administrativen Rechten erstellen

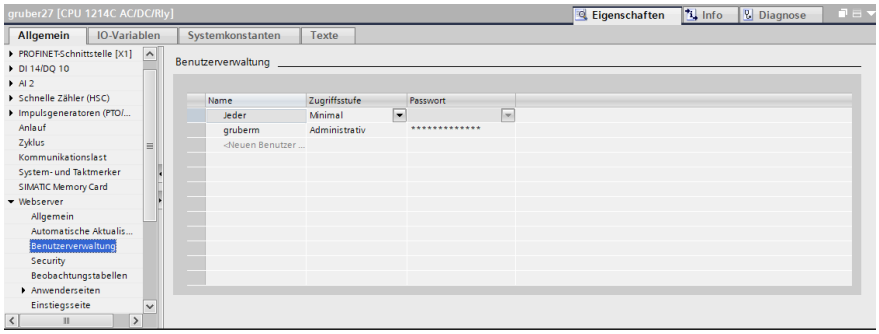

Figure 9 Benutzer erstellen

- Bei „Security“ den Zertifikatstyp auf „Hardwaregeneriert“ umstellen

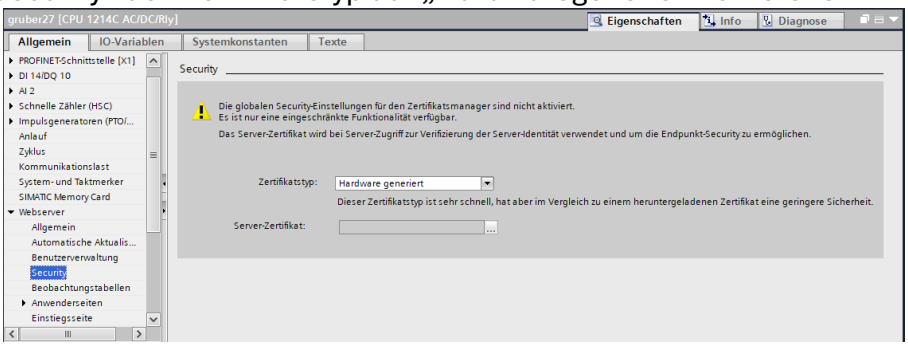

Figure 10 Zertifikat

- Beobachtungstabelle „Motor“ unter „Beobachtungstabellen“ mit Lese/Schreib-Rechten hinzufügen

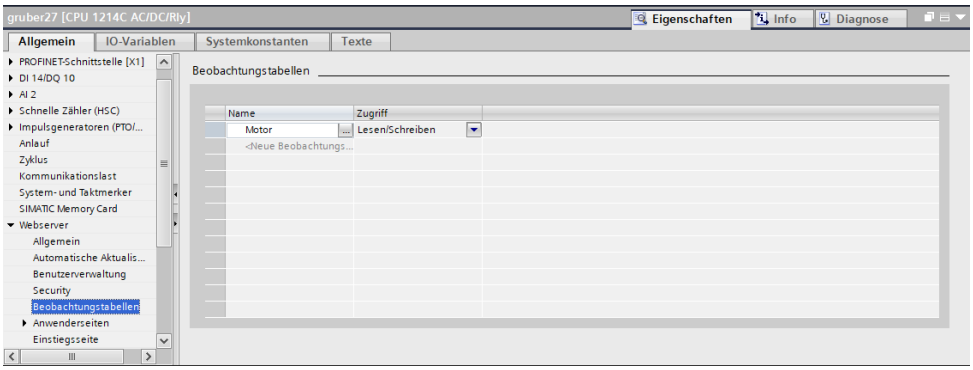

Figure 11 Lese- Schreibrechte

- Unter der IP-Adresse den Webserver aufrufen

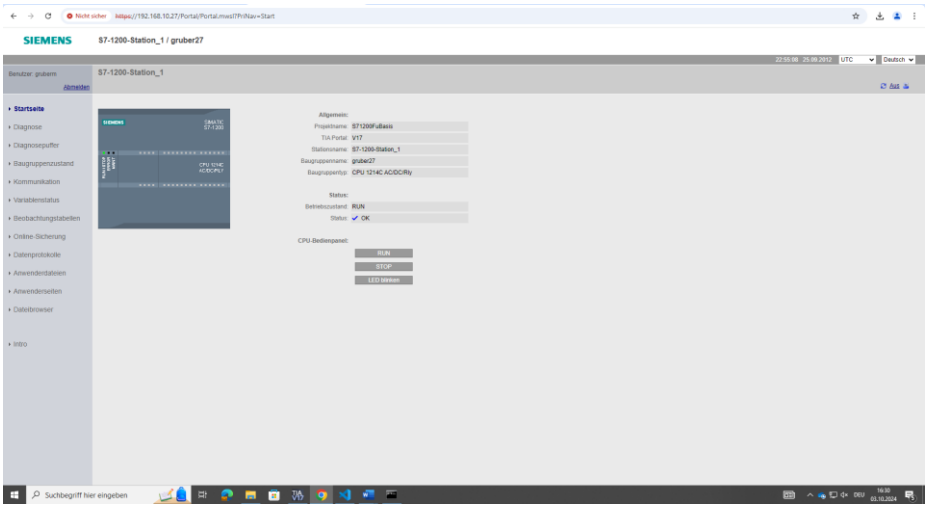

Figure 12 Webserver aufrufen

- Unter „Systembausteine/Webserver“ „DB333“ auswählen

- Rechts unter „Kommunikation/Webserver“ das Modul „WWW“ reinziehen
- RET_VAL als retval setzen und die Variable mit Rechtsklick definieren

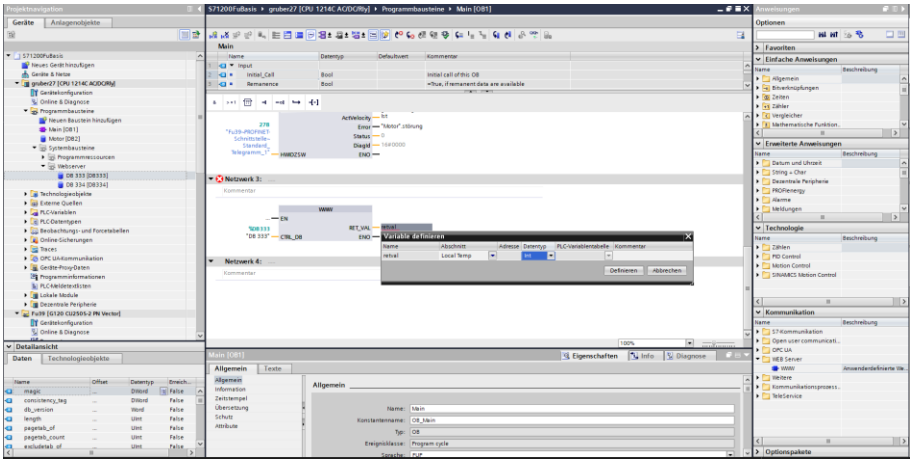

Figure 13 Variablen definieren

- Im Project ornder unter UserFiles den Ordner „Web“ erstellen und die Startseite die wir vom Fachlehrer bekommen haben reinkopieren

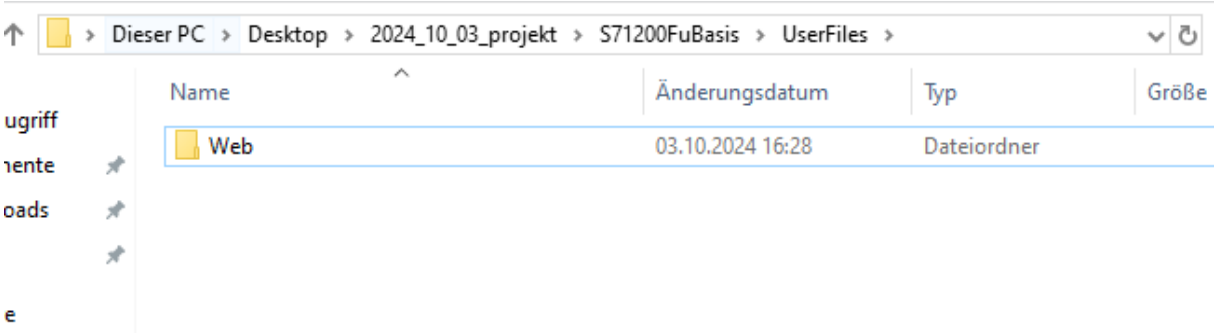

Figure 14 Startseite erstellen

- Unter „Webserver/Anwenderseiten“ das HTML Verzeichnis und den Applikationsnamen definieren

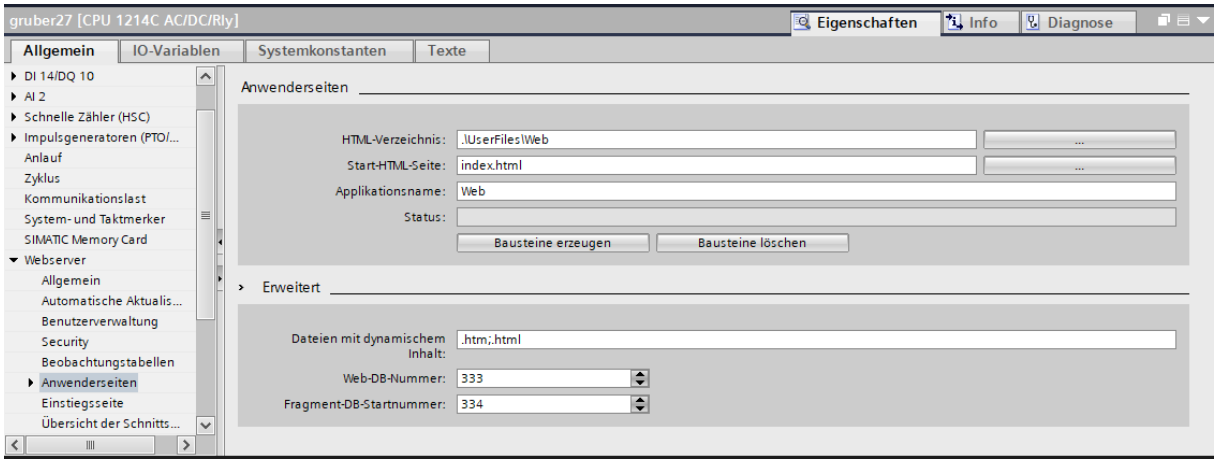

Figure 15 Verzeichnis und Namen definieren

- Folgende Ausgabe erscheint unter Allgemein wenn alles funktioniert hat

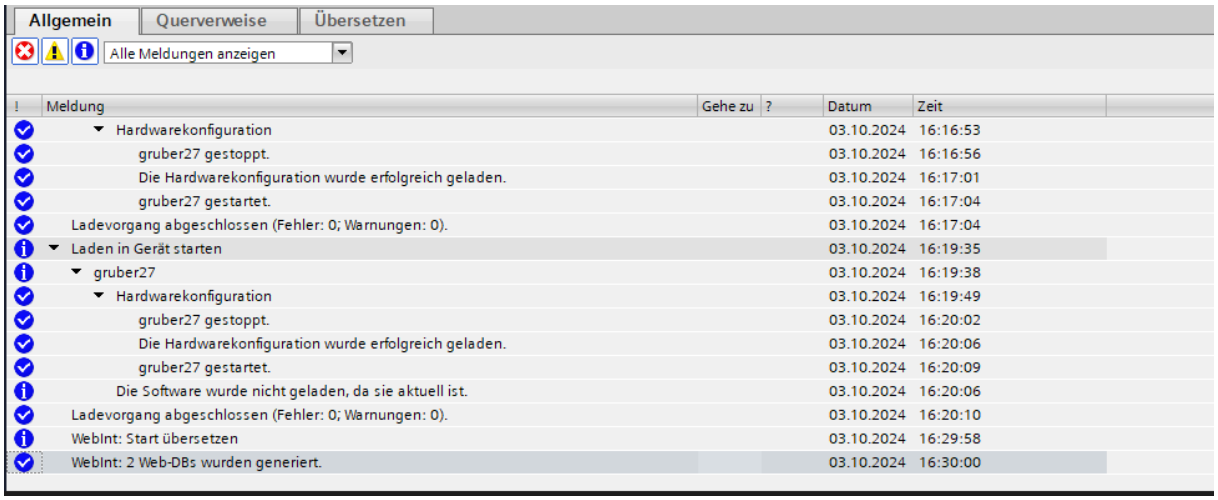

Figure 16 Output

- Auf der vorher generierten Seite unter Anwenderseiten kann man nun auf die erstellte Website zugreifen und den Motor über ebendiese steuern.

Figure 17 Webserver

09.01.2025 - Protokoll

Motoransteuerung über API-Requests

Ziel

Das Ziel dieser Übung ist es, einen Motor über API-Requests zu steuern. Dies wird zunächst mit Postman getestet, um die grundlegenden API-Funktionen zu verstehen, und anschließend in einem Python-Skript umgesetzt.

Login mit Postman

- In Postman wird ein POST-Request an die `/jsonrpc`-Route gesendet.
- Das JSON-Body enthält folgende Struktur:

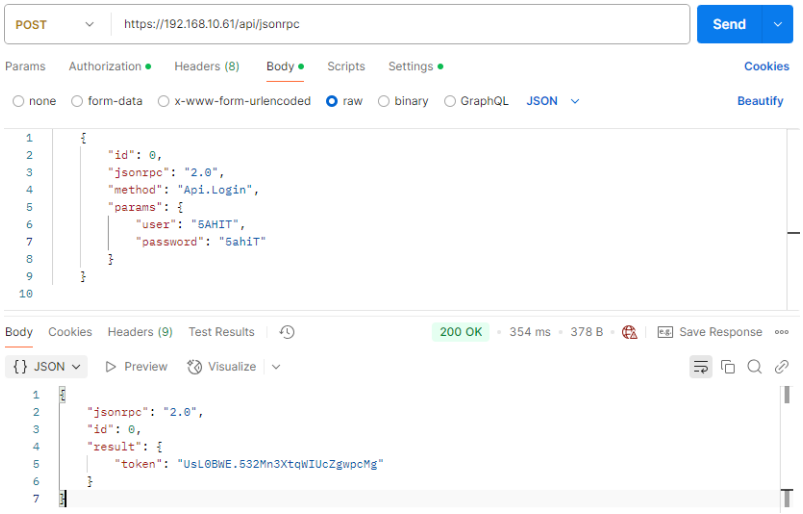

Figure 18 JSON-Body in Postman

- Als Antwort wird ein Token zurückgegeben, das für alle weiteren API-Requests erforderlich ist.

API-Calls mit Python

Um die API einfacher zu verwenden, wird der Login in einer Funktion gekapselt.

Login-Funktion in Python

Die folgende Funktion sendet den Login-Request und gibt den Token zurück:

```python
import requests

URL = "https://192.168.10.61/api/jsonrpc"


def login(username, password) -> str:

    payload = {
    "id": 0,
    "jsonrpc": "2.0",
    "method": "Api.Login",
    "params": {
        "user": username,
        "password": password
    }
    }

    headers = {
        "Content-Type": "application/json"
    }

    try:
        response = requests.post(URL, json=payload, headers=headers, verify=False)
        response.raise_for_status()

        response_json = response.json()

        token = response_json.get("result", {}).get("token")
        return token

    except requests.exceptions.RequestException as e:
        print("Error:", e)


if __name__ == "__main__":
    token = login("5AHIT", "5ahiT")
```

Figure 19 Login API-Call Python

Funktionen durch Browse entdecken

Um verfügbare Funktionen für den Motor herauszufinden, kann die `Browse`-Route verwendet werden.

Diese Route wird durch folgenden Code aufgerufen. In diesem Fall wird geschaut, welche Funktionen man für die Komponente „Motor“ ausführen kann.

Der vorher erhaltene Token muss angegeben werden.

```python
def browseMotor(token: str):
    payload = {
        "jsonrpc": "2.0",
        "id": 4,
        "method": "PlcProgram.Browse",
            "params": {
            "var": "\"Motor\"",
            "mode": "children"
        }
    }

    headers = {
        "Content-Type": "application/json",
        "X-Auth-token": f"{token}"
    }


    try:
        response = requests.post(URL, json=payload, headers=headers, verify=False)
        response.raise_for_status()

        response_json = response.json()
        return response_json

    except requests.exceptions.RequestException as e:
        print("Error:", e)
```

Figure 20 Browse Python

Folgende Ausgabe wird erzeugt, wenn man den Return-Wert dieser Funktion ausgibt.

{'jsonrpc': '2.0', 'id': 4, 'result': [{'name': 'ein', 'db_number': 3, 'datatype': 'bool'}, {'name': 'störungQuittieren', 'db_number': 3, 'datatype': 'bool'}, {'name': 'drehrichtungInvertieren', 'db_number': 3, 'datatype': 'bool'}, {'name': 'Sollgeschwindigkeit', 'db_number': 3, 'datatype': 'real'}, {'name': 'Istgeschwindigkeit', 'db_number': 3, 'datatype': 'real'}, {'name': 'freigegeben', 'db_number': 3, 'datatype': 'bool'}, {'name': 'lokalBetrieb', 'db_number': 3, 'datatype': 'bool'}, {'name': 'störung', 'db_number': 3, 'datatype': 'bool'}]}

Figure 21 Browse Output

Motor starten

Durch das vorherige Browsen wurde festgestellt, dass der Motor ein „.ein“ Attribut hat, welches den Motor startet.

Setzt man dieses Attribut hier auf True wird der Motor beim Aufrufen dieser Route gestartet.

```
payload = {
    "jsonrpc": "2.0",
    "method": "PlcProgram.Write",
    "id": 1,
    "params": {
        "var": "\"Motor\".ein",
        "value": True
    }
}
```

Figure 22 JSON für Motor starten

Folgendes Bild zeigt die ganze Funktion, die den Motor startet.

```
def startMotor(token: str):
    payload = {
        "jsonrpc": "2.0",
        "method": "PlcProgram.Write",
        "id": 1,
        "params": {
            "var": "\"Motor\".ein",
            "value": True
        }
    }

    headers = {
        "Content-Type": "application/json",
        "X-Auth-token": f"{token}"
    }


    try:
        response = requests.post(URL, json=payload, headers=headers, verify=False)
        response.raise_for_status()

        response_json = response.json()
        return response_json

    except requests.exceptions.RequestException as e:
        print("Error:", e)
```

Figure 23 Motor starten Funktion

Motor stoppen

Auf ähnlichem Weg kann der Motor auch wieder gestoppt werden.

Das Motor.ein Attribut muss auf False gesetzt werden.

```
payload = {
    "jsonrpc": "2.0",
    "method": "PlcProgram.Write",
    "id": 1,
    "params": {
        "var": "\"Motor\".ein",
        "value": False
    }
}
```

Figure 24 JSON für Motor stoppen

Der restliche Code ist gleich wie beim Starten des Motors.

Erste Applikation

Durch folgenden Code wird der Motor für 5 Sekunden eingeschaltet und dann wieder aus.

```
if __name__ == "__main__":
    token = login("5AHIT", "5ahiT")
    print(browseMotor(token))
    startMotor(token)
    time.sleep(5)
    stopMotor(token)
```

Figure 25 Main.py

Abbildungsverzeichnis

Figure 1 Aufbau 2
Figure 2 IP Eingabe & Subnetz 2
Figure 3 verbinden 3
Figure 4 Motor in Betrieb 4
Figure 5 Gerätekonfiguration - Motor 4
Figure 6 Programmbaustein - Motor 5
Figure 7 Beobachtungstabelle 6
Figure 8 Webserver aktivieren 6

Figure 9 Benutzer erstellen 6
Figure 10 Zertifikat 7
Figure 11 Lese- Schreibrechte 7
Figure 12 Webserver aufrufen 7
Figure 13 Variablen definieren 8
Figure 14 Startseite erstellen 8
Figure 15 Verzeichnis und Namen definieren 9
Figure 16 Output 9
Figure 17 Webserver 10
Figure 18 JSON-Body in Postman 11
Figure 19 Login API-Call Python 12
Figure 20 Browse Python 13
Figure 21 Browse Output 13
Figure 22 JSON für Motor starten 14
Figure 23 Motor starten Funktion 14
Figure 24 JSON für Motor stoppen 15
Figure 25 Main.py 15