

BACHELORARBEIT

zur Erlangung des akademischen Grades

„Bachelor of Science in Engineering“ im Studiengang
Maschinenbau

MINDfactoree

**Entwicklung eines Monitoring-Dashboards zur Analyse von
Umweltbedingungen, Leistungsindikatoren und Ertragsprognosen**

Ausgeführt von: Philipp Gasser

Personenkennzeichen: 2010779092

BegutachterIn: PD DI Dr. Maximilian Lackner, MBA

Wien, 04.05.2023

Eidesstattliche Erklärung

„Ich, als Autor / als Autorin und Urheber / Urheberin der vorliegenden Arbeit, bestätige mit meiner Unterschrift die Kenntnisnahme der einschlägigen urheber- und hochschulrechtlichen Bestimmungen (vgl. etwa §§ 21, 46 und 57 UrhG idgF sowie § 14 Satzungsteil Studienrechtliche Bestimmungen / Prüfungsordnung der FH Technikum Wien).

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt und Gedankengut jeglicher Art aus fremden sowie selbst verfassten Quellen zur Gänze zitiert habe. Ich bin mir bei Nachweis fehlender Eigen- und Selbstständigkeit sowie dem Nachweis eines Vorsatzes zur Erschleichung einer positiven Beurteilung dieser Arbeit der Konsequenzen bewusst, die von der Studiengangsleitung ausgesprochen werden können (vgl. § 14 Abs. 1 Satzungsteil Studienrechtliche Bestimmungen / Prüfungsordnung der FH Technikum Wien).

Weiters bestätige ich, dass ich die vorliegende Arbeit bis dato nicht veröffentlicht und weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt habe. Ich versichere, dass die abgegebene Version jener im Uploadtool entspricht.“

Ort, Datum

Unterschrift

Abstract

Nowadays, the use of renewable energies is becoming increasingly relevant in the industrial context. Especially the use of photovoltaic systems is extremely popular due to the rapid and inexpensive installation. Often factories are already equipped with energy meters and PV systems, but a comprehensible, central retrievability of the measured values is usually missing. Influences of geographical and technical parameters on the yield are also often neglected.

These problems are to be solved by integrating an energy meter, a photovoltaic module, various sensors and by developing a dashboard based on the MINDfactory created by Fraunhofer Research Austria. So far, the MINDfactory has enabled best production planning and a reduction in operating costs, energy costs and investment costs by integrating algorithms. But soon, the currently available energy resources will also be included in the planning process to make planning even more efficient.

The power consumption of the MINDfactory is measured with an SDM230 Modbus energy meter, the power production of the PV module with a voltage and ACS712 current sensor and the battery temperature with a DHT22 temperature sensor. The already installed BME680 environmental sensor and LDR03 photoresistor enables the evaluation of factory temperature, humidity, air quality and brightness. The measured values are then transferred to a TXT controller by using practical communication protocols, such as MQTT or ModBus, and saved in a NoSQL MongoDB database. Finally, all datasets are represented graphically and complemented by an experimental PV power prediction based on global radiation, area covered by PV modules and efficiency. The graphs are created using Plotly and the dashboard is based on the Python Django web framework. To allow all applications to run in parallel without relying on enormous resources, Docker container virtualization was used.

The results show that significant time savings in data acquisition, data retrieval, and compression can be achieved by moving to a time series database, such as InfluxDB. In addition, a temperature sensor with a lower sampling time and separate data collection of voltage or current can be used to prevent data gaps in power production. Finally, Django Channels can be used to display the measured values (excluding the graphs) in real-time on the dashboard.

Keywords: MINDfactory, PV, Raspberry Pi, Python, Django, Docker, ModBus, MQTT, MongoDB, ACS712, DHT22, SDM230, BME680, LDR03

Kurzfassung

Heutzutage gewinnt die Nutzung von erneuerbaren Energien im industriellen Kontext immer mehr an Relevanz. Hier ist vor allem der Einsatz von Photovoltaikanlagen aufgrund der rapiden und preiswerten Installation sehr populär. Häufig sind Fabriken zwar bereits mit Energiemessgeräten und PV-Anlagen ausgestattet, doch eine nachvollziehbare, zentrale Abrufbarkeit der Messwerte bleibt meist aus. Einflüsse von geographischen und technischen Parametern auf den Ertrag werden ebenso oftmals vernachlässigt.

Dieses Problem soll durch die Integration eines Energiemessgerätes, eines Photovoltaikmoduls bzw. dazugehöriger Sensorik und durch die Entwicklung eines Dashboards auf Basis, der von Fraunhofer Research Austria entwickelten MINDfactory, gelöst werden. Bislang ermöglichte die MINDfactory durch Einbindung von Algorithmen eine optimale Produktionsplanung und somit eine Reduzierung der operativen Kosten, der Energiekosten und der Investitionskosten. Doch bald schon, sollen in einem weiteren Forschungsprojekt auch die derzeit zur Verfügung stehenden Energieressourcen in die Planung eingebunden werden.

Die Messung der Leistungsaufnahme der MINDfactory erfolgt mit einem SDM230 Modbus Energiemessgerät, die Leistungsproduktion des PV-Moduls mit einem Spannungs- und ACS712-Stromsensors und die Batterietemperatur mit einem DHT22-Temperatursensor. Der bereits verbaute BME680-Umweltsensor und LDR03-Fotowiderstand ermöglicht die Auswertung der Fabriktemperatur, der Luftfeuchtigkeit, der Luftqualität und der Helligkeit. Die ermittelten Messwerte werden durch die Nutzung praxisnaher Kommunikationsprotokolle, wie MQTT bzw. ModBus, zu einem bereits im Netzwerk integrierten TXT-Controller transferiert und zur Aufbewahrung in einer NoSQL-MongoDB-Datenbank gesichert. Letztlich werden sämtliche Datensätze graphisch dargestellt und durch eine experimentelle PV-Leistungsprognose auf Basis der Globalstrahlung, der durch die PV-Module abgedeckten Fläche und des Wirkungsgrads ergänzt. Die Erstellung der Diagramme erfolgt mit Plotly und das Dashboard basiert auf dem Python Django Webframework. Damit sämtliche Applikationen parallel ausgeführt werden können, ohne auf enorme Ressourcen angewiesen zu sein, wurde die Containervirtualisierung Docker genutzt.

Die Ergebnisse weisen auf, dass durch einen Umstieg auf eine Zeitreihendatenbank, wie InfluxDB, erhebliche Zeitersparnisse bei der Datenaufnahme, der Datenabfrage und der Komprimierung erzielt werden können. Darüber hinaus kann ein Temperatursensor mit geringerer Abtastzeit und separater Datenerhebung von Spannung bzw. Stromstärke verwendet werden, um Datenlücken in der Leistungsproduktion vorzubeugen. Letztlich können Django Channels verwendet werden, um die Messwerte (exklusive der Diagramme) in Echtzeit am Dashboard anzeigen zu können.

Schlagwörter: MINDfactory, PV, Raspberry Pi, Python, Django, Docker, ModBus, MQTT, MongoDB, ACS712, DHT22, SDM230, BME680, LDR03

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelorarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Herrn PD DI Dr. Maximilian Lackner, MBA, der meine Bachelorarbeit betreut und begutachtet hat. Ebenso möchte ich mich bei den Herrn Dipl.-Ing. Nikolaus Kremslehner, Dipl.-Ing. Wolfgang Dummer und Gerhard Scheucher BSc. bedanken, welche mich mit ihrer Expertise unterstützten.

Für die hilfreichen Anregungen, die schnelle Abrufbarkeit und die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken. Ich konnte mich sehr glücklich schätzen, eine Betreuung genießen zu dürfen, die wesentlich am Fortschritt meiner Arbeit interessiert war.

Abschließend möchte ich mich bei meiner Mutter und meiner Partnerin bedanken, die mir mein Studium durch ihre Unterstützung ermöglicht haben und stets ein offenes Ohr für mich hatten.

Inhaltsverzeichnis

Abkürzungsverzeichnis	8
Abbildungsverzeichnis	9
Tabellenverzeichnis	11
Variablenverzeichnis.....	12
1 Problemstellung und Zielsetzung	13
1.1 Problemstellung	13
1.2 Forschungsfragen	14
1.3 Zielsetzung	14
1.4 Nicht-Ziele.....	14
1.5 Lösungsansatz.....	15
2 Theoretische Grundlagen.....	16
2.1 MINDfactory	16
2.2 Energiemessgerät.....	21
2.3 Photovoltaikanlagen.....	22
2.3.1 Allgemeines	22
2.3.2 Marktentwicklung in Österreich	23
2.3.3 Komponenten	24
2.3.4 Einflussfaktoren auf Energieerträge	26
3 Methodik	35
3.1 Datenerfassung.....	36
3.1.1 Programmierung des Grundgerüsts	37
3.1.2 Integration des Energiemessgerätes.....	39
3.1.3 Integration des Photovoltaikmoduls.....	44
3.2 Datensicherung.....	50
3.3 Visualisierung	52
4 Ergebnis und Diskussion.....	54
4.1 Ergebnis	54
4.2 Diskussion	56
4.3 Weitere Vorgehensweise	58

Literaturverzeichnis	60
Anhang A: 01_dummy_consumption.py	65
Anhang B: 01_dummy_production.py	66
Anhang C: 01_real_consumption.py	67
Anhang D: 01_real_production.py	68
Anhang E: 02_consumption_to_mongodb.py	70
Anhang F: 02_production_to_mongodb.py	71
Anhang G: 02_bme_to_mongodb.py	72
Anhang H: 02_ldr_to_mongodb.py	73
Anhang I: views.py	74
Anhang J: index_pv_modul.html	80
Anhang K: update_chart.js	84

Abkürzungsverzeichnis

ADC	Analog-Digital-Converter
API	Application Programming Interface
BSON	Binary Java Script Object Notation
DC	Direct Current
EU	Europäische Union
HTTP	Hypertext Transfer Protocol
IAQ	Indoor air quality
IoT	Internet of Things
JSON	Java Script Object Notation
MINDfactory	MINiature Digital Factory
MQTT	Message Queuing Telemetry Transport
NUC	Next Unit of Computing
OPC UA	Open Platform Communications United Architecture
PC	Personal Computer
PV	Photovoltaik
RTU	Remote Terminal Unit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Abbildungsverzeichnis

Abbildung 2.1: Lernfabrik 4.0 24V – Fischertechnik [1]	16
Abbildung 2.2: Schematische Darstellung der Lernfabrik 4.0 24V – Fischertechnik [2]	17
Abbildung 2.3: Systemarchitektur der MINDfactory [4]	18
Abbildung 2.4: Schematischer Aufbau von Django-Web-Applikationen [18]	20
Abbildung 2.5: Eastron SDM230-Modbus [21]	21
Abbildung 2.6: Strahlungsbilanz der Erde [26]	22
Abbildung 2.7: Die Marktentwicklung der Photovoltaik in Österreich bis 2021 [27]	23
Abbildung 2.8: PV-Batteriespeicherkapazität in MWh von 2014 bis 2021 [27]	24
Abbildung 2.9: Blockschaltbild eines photovoltaischen Inselsystems [28]	24
Abbildung 2.10: Einfluss der Umgebungstemperatur auf den Wirkungsgrad [29]	26
Abbildung 2.11: Einfluss der relativen Luftfeuchtigkeit auf die Ausgangsleistung [30]	27
Abbildung 2.12: Einfluss des Windes auf die Ausgangsleistung [31]	27
Abbildung 2.13: Einfluss des Wolkenstandes auf die Sonneneinstrahlung [32]	28
Abbildung 2.14: Erläuterung von Azimut- und Elevationswinkel [26]	29
Abbildung 2.15: Sonnenstandsdiagramm zur Verschattungsanalyse [26]	29
Abbildung 2.16: Spezifische Erträge von Standard- und Bifazialen PV-Systemen [35]	31
Abbildung 2.17: Weltmarktanteile von mono- und bifazialen PV-Modulen [36]	31
Abbildung 2.18: Spezifischer jährlicher Energieertrag bei variierendem Modulreihenabstand [26]	32
Abbildung 2.19: Energieerträge von verschiedenen Aufständern [37]	33
Abbildung 2.20: Arten von gebäudebezogenen Anlagen nach [38]	34
Abbildung 3.1: Blockschaltbild zu Illustration der Herangehensweise dieser Arbeit	35
Abbildung 3.2: Blockschaltbild der Datenerfassung	36
Abbildung 3.3: Blockschaltbild des Grundgerüsts der Datenerfassung	37
Abbildung 3.4: Optimierte Blockschaltbild des Grundgerüsts der Datenerfassung	37
Abbildung 3.5: Datenübertragung des Energieverbrauchs via S0-Schnittstelle [22, 23]	39
Abbildung 3.6: Datenübertragung des Energieverbrauchs via USB [22, 23, 24]	40
Abbildung 3.7: Datenübertragung des Energieverbrauchs via Ethernet [22, 23, 25]	41
Abbildung 3.8: Schaltungsaufbau des Energiemessgerätes	42
Abbildung 3.9: Blockschaltbild von Datenerfassung des Energiemessgerätes	43
Abbildung 3.10: Verwendetes PV-Modul [39]	44
Abbildung 3.11: Raspberry Pi 4 Model B [40]	45
Abbildung 3.12: Spannungssensor 0-25V DC [42]	45
Abbildung 3.13: Stromsensor ACS712-30A [43]	46
Abbildung 3.14: Analog-Digital-Converter ADS1115 [44]	46
Abbildung 3.15: Temperatursensor DHT22 [45]	47
Abbildung 3.16: Schaltungsaufbau des Photovoltaikmoduls mit Sensorik	48

Abbildung 3.17: Optionaler Schaltplan für Schaltungsaufbau des Photovoltaikmoduls mit Sensorik	49
Abbildung 3.18: Blockschaltbild von Datenerfassung des Photovoltaikmoduls	49
Abbildung 3.19: Blockschaltbild der Datensicherung	50
Abbildung 3.20: Blockschaltbild der Visualisierung	52
Abbildung 4.1: Dashboard	54
Abbildung 4.2: Air-Quality-Index (IAQ = Indoor Air Quality)	55

Tabellenverzeichnis

Tabelle 2.1: Erläuterung der Stationskürzel	17
Tabelle 2.2: Leistung und Temperatur bei Verschmutzungen auf PV-Panel [33]	28
Tabelle 3.1: Stückliste von Konzept 1 (S0-Schnittstelle)	39
Tabelle 3.2: Vor- und Nachteile der Datenerfassung via S0-Schnittstelle	39
Tabelle 3.3:Stückliste von Konzept 2 (ModBus via USB).....	40
Tabelle 3.4: Vor- und Nachteile der Datenerfassung via USB-Schnittstelle	40
Tabelle 3.5: Stückliste von Konzept 3 (ModBus via Ethernet).....	41
Tabelle 3.6: Vor- und Nachteile der Datenerfassung via Ethernet-Schnittstelle	41
Tabelle 3.7: Nutzwertanalyse für Konzepte zur Integration des Energiemessgerätes	42
Tabelle 3.8: Anschlusstabelle für Schaltungsaufbau des Photovoltaikmoduls mit Sensorik ..	48

Variablenverzeichnis

Zeichen	Bedeutung	Einheit
I	Stromstärke	A
R	Widerstand	Ω
U	Spannung	V
V_{cc}	Voltage at the common collector	V

1 Problemstellung und Zielsetzung

1.1 Problemstellung

Heutzutage gewinnt die Nutzung von erneuerbaren Energien, nicht nur im privaten, sondern auch im industriellen Kontext immer mehr an Relevanz. Sowohl der rapide Anstieg an Energiekosten, welcher durch unvorhersehbare geopolitische Restriktionen verursacht wurde, als auch die Reduktion von Schadstoffausstoßen und die langfristige Einsparung an Kapitalmitteln, rücken erneuerbare Energien immer mehr in den Vordergrund. Viele Unternehmen sind gerade im Gange die Nutzung von Sonnen-, Wind- oder Wasserkraft in deren Fabrikplanung einfließen zu lassen. Vor allem in dieser Branche wird sich der zukunftsorientierte Trend der erneuerbaren Energie stetig durchsetzen. Hierbei ist vor allem der Einsatz von Photovoltaikanlagen beliebt, da eine spontane Aufrüstung schnell realisierbar ist. Häufig sind zwar bereits Energiezähler und PV-Anlagen in Fabriken integriert, aber eine sinnvoll nachvollziehbare, schnell abrufbare Visualisierung bleibt aus. Hinsichtlich der Erreichung der EU-Klimaschutzpakete ist das auch ein erster Schritt in die richtige Richtung, aber der wichtigste Aspekt, nämlich die Gegenüberstellung von Leistungsproduktion zu Leistungsaufnahme, wird leider dennoch oftmals vergessen.

Durch den Fortschritt in ein digitales Zeitalter, ist vielen Unternehmen der Begriff „Digitaler Zwilling“ kein Fremdwort mehr. Mit Hilfe der Fischertechnik „Fabrik Simulation 24V“-Trainingsmodells, ist es der Fraunhofer Austria Research GmbH erstmalig gelungen, einen vollständig digitalen Zwilling durch Planungssteuerung via Dashboard in eine Produktion zu implementieren. Vom automatisierten Hochregal über eine Multi-Bearbeitungsstation mit Brennofen, einer Sortierstrecke mit Farberkennung bis hin zu einem Vakuum-Sauggreifer lässt die Miniaturfabrik nichts zu wünschen übrig. Doch die Smart-Factory von Morgen hat immer mehr Ansprüche an die Hersteller, weshalb sich die Fraunhofer Research Austria GmbH dazu entschlossen hat, die MINiature Digital Factory, auch MINDfactory genannt, als Demonstrator für zukunftsorientierte Produktionslösungen ständig weiterzuentwickeln. Bislang war es möglich durch die Kombination von Algorithmen, Applikationen und damit verbundenen Dashboards die operativen Kosten, die Energiekosten und die Investitionskosten bestmöglich zu reduzieren.

Bald jedoch soll der Demonstrator erstmalig auch mit erneuerbarer Energie, um genau zu sein, mit Photovoltaik in Kontakt treten. Unter Zuhilfenahme einer Photovoltaikanlage beziehungsweise eines Energiemessgerätes, soll die von der MINDfactory produzierte und aufgenommene Leistung gemessen werden. Um dem Endkunden eine unkomplizierte und bequeme Möglichkeit zum jederzeitigen Abruf der Parameter bieten zu können, sollen die Messdaten über ein Dashboard visualisiert werden.

1.2 Forschungsfragen

Um nun aus der oben angeführten Problemstellung einen nutzbaren Mehrwert ziehen zu können, ist die Beantwortung folgender Forschungsfragen essenziell:

1. Wie kann ein Modbus Energiemessgerät in die MINDfactory integriert und die damit verbundene Leistungsaufnahme ermittelt werden?
2. Wie kann ein Photovoltaik-Modul in die MINDfactory integriert und die damit verbundene Leistungsproduktion ermittelt werden?
3. Wie können genannte Leistungen per MQTT veröffentlicht, in MongoDB abgesichert und anhand eines interaktiven Diagramms im Dashboard gegenübergestellt werden?
4. Wie können externe Schnittstellen genutzt werden, um Prognosen über künftige Erträge der Photovoltaikanlage treffen zu können?

1.3 Zielsetzung

Das Ziel der Arbeit ist es, offenzulegen, wie die MINDfactory mit einem Photovoltaikmodul und einem Energiemessgerät erweitert werden muss, um die Datenerfassung von Leistungsaufnahme bzw. Leistungsproduktion realisieren zu können. Darauf aufbauend, werden die ermittelten Messwerte einerseits durch die Nutzung von Kommunikationsprotokollen, wie MQ Telemetry Transport (MQTT) oder ModBus, zu einem bereits im Netzwerk integrierten TXT-Controller transferiert und andererseits zur Aufbewahrung in einer NoSQL-Datenbank gesichert. Letztlich sollen beide Datensätze sowohl graphisch gegenübergestellt als auch in einem Dashboard abrufbar sein.

1.4 Nicht-Ziele

Das Ziel dieser Arbeit ist es nicht, jede bisherig implementierte Komponente der MINDfactory in Bezug auf Hardware und Software zu optimieren oder zu ersetzen.

Außerdem sollen keine elektrischen Komponenten von Grund auf selbst entwickelt werden., beispielweise die Entwicklung von Sensorik oder Ähnlichem.

Der Fokus dieser Arbeit liegt außerdem nicht auf der funktionalen elektrotechnischen Entwicklung einer Photovoltaikanlage. Die Schaltungselektronik wurde in Zusammenarbeit mit der Fraunhofer Research Austria GmbH erstellt und wird im Rahmen dieser Arbeit nicht behandelt. Lediglich der Aufbau zur Datenerhebung wird vollständigshalber erläutert.

1.5 Lösungsansatz

Um nun aus den oben angeführten Forschungsfragen die grundlegenden Ideen erläutern zu können, ist die Einteilung der Arbeit in folgende Lösungsansätze unumgänglich:

INTEGRATION DES PHOTOVOLTAIK-MODULS

1. Wahl eines 50W Peak PV-Moduls, eines Spannungs-, eines Strom- und eines Temperatursensors
2. Verkabelung der Komponenten
3. Datenerhebung mittels Spannungs-, Strom- und Temperatursensoren
4. Datentransfer von Raspberry Pi zu TXT-Controller via MQTT
5. Datensicherung in No-SQL-Datenbank MongoDB

INTEGRATION DES ENERGIEMESSGERÄTES

1. Wahl eines Modbus Energiemessgerätes und eines RS485-Converters
2. Verkabelung der Komponenten
3. Datenerhebung mittels Modbus Energiemessgerät via RJ45
4. Datentransfer von TCP/IP-Converter zu TXT-Controller via MQTT
5. Datensicherung auf No-SQL-Datenbank MongoDB

VISUALISIERUNG DER MESSDATEN IN DASHBOARD

1. Datenempfang von Messwerten via MQTT
2. Verarbeitung von Messwerten in Python-Applikation
3. Erstellung bzw. Erweiterung des Dashboards

EINBINDUNG VON WETTERPROGNOSEN ZUR VORHERSAGE VON ENERGIEERTRÄGEN EINES PV-SYSTEMS

1. Wahl einer vertrauenswürdigen Wetter- oder PV-API-Schnittstelle
2. Empfang der Prognosedaten via HTTP-Requests
3. Darstellung der Kennwerte im Dashboard

2 Theoretische Grundlagen

2.1 MINDfactory

Das Fundament der MINDfactory ist das Fischertechnik „Fabrik Simulation 24V“-Trainingsmodell. Es gilt nicht nur als Lernumgebung für Bildungseinrichtungen, sondern auch als Demonstrator von künftigen Industrie-4.0-Anwendungen im Bereich der Forschung und der Entwicklung. Die Fabrik erlaubt es die Bestell-, Produktion- und Lieferprozesse besser nachvollziehen und letztendlich besser planen zu können. Der Produktionsablauf umfasst in der Regel eine Anlieferung der Rohware aus dem Hochregallager, eine Bearbeitung des Werkstücks im Brennofen beziehungsweise der Bearbeitungsstation und eine farbabhängige Einlagerung von der Sortierstrecke zurück in das Hochregallager. Der Transport zwischen den diversen Stationen wird durch einen Vakuum-Sauggreifer gewährleistet. Darüber hinaus ist die Fabrik sowohl mit einem Umweltsensor BME680, welcher Temperatur, Luftdruck, Luftqualität und Luftfeuchtigkeit messen kann, als auch einem Fotowiderstand LDR03, welcher eine Helligkeitsmessung ermöglicht, ausgestattet. Zur Nachvollziehbarkeit der Prozessschritte sind die Werkstücke mit einem NFC-Chip gekennzeichnet. Mit Abmessungen von 1.16 x 0.77 x 0.42m und einem Gewicht von gerade einmal 24kg bietet diese Lernfabrik also durchaus eine realistische Möglichkeit zur sinnhaften Realisierung von Optimierungen beziehungsweise Erweiterungen im Bereich der Fabrikplanung und des Produktionsmanagements. [1]

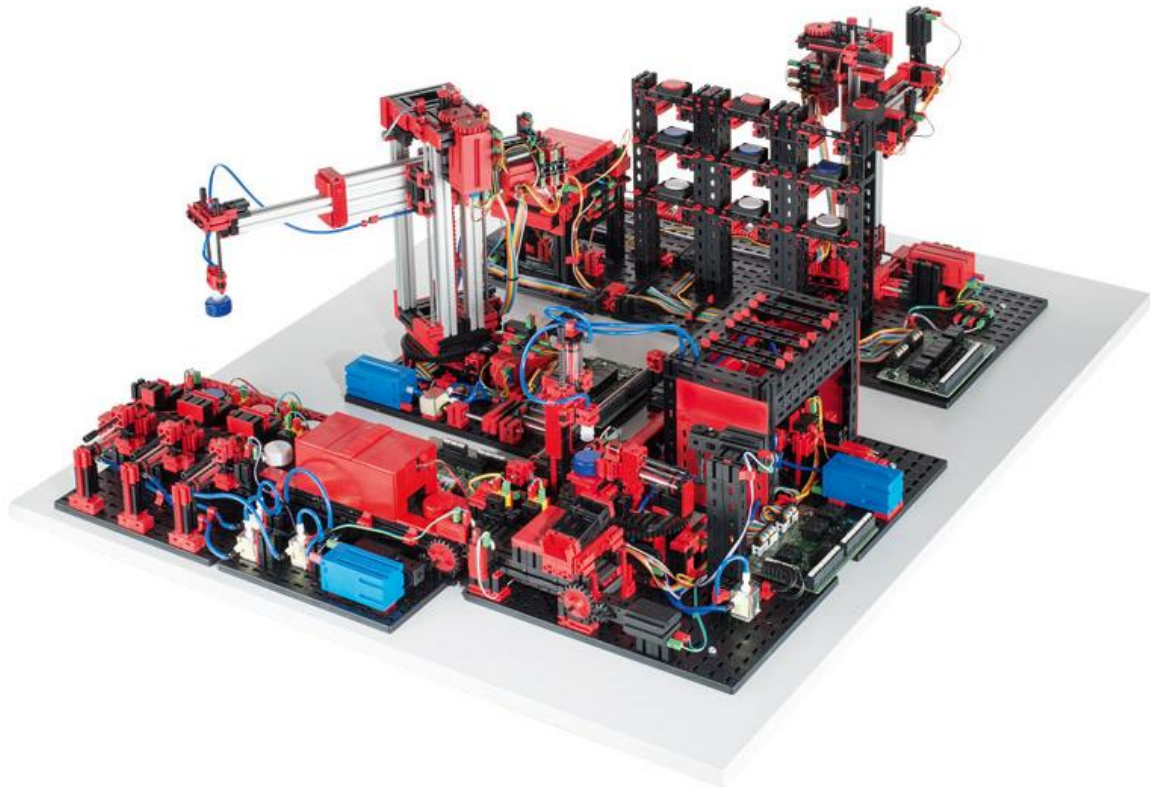


Abbildung 2.1: Lernfabrik 4.0 24V – Fischertechnik [1]

Um die jeweiligen Komponenten beziehungsweise deren Kommunikation besser nachvollziehen zu können, ist folgende Grafik notwendig:

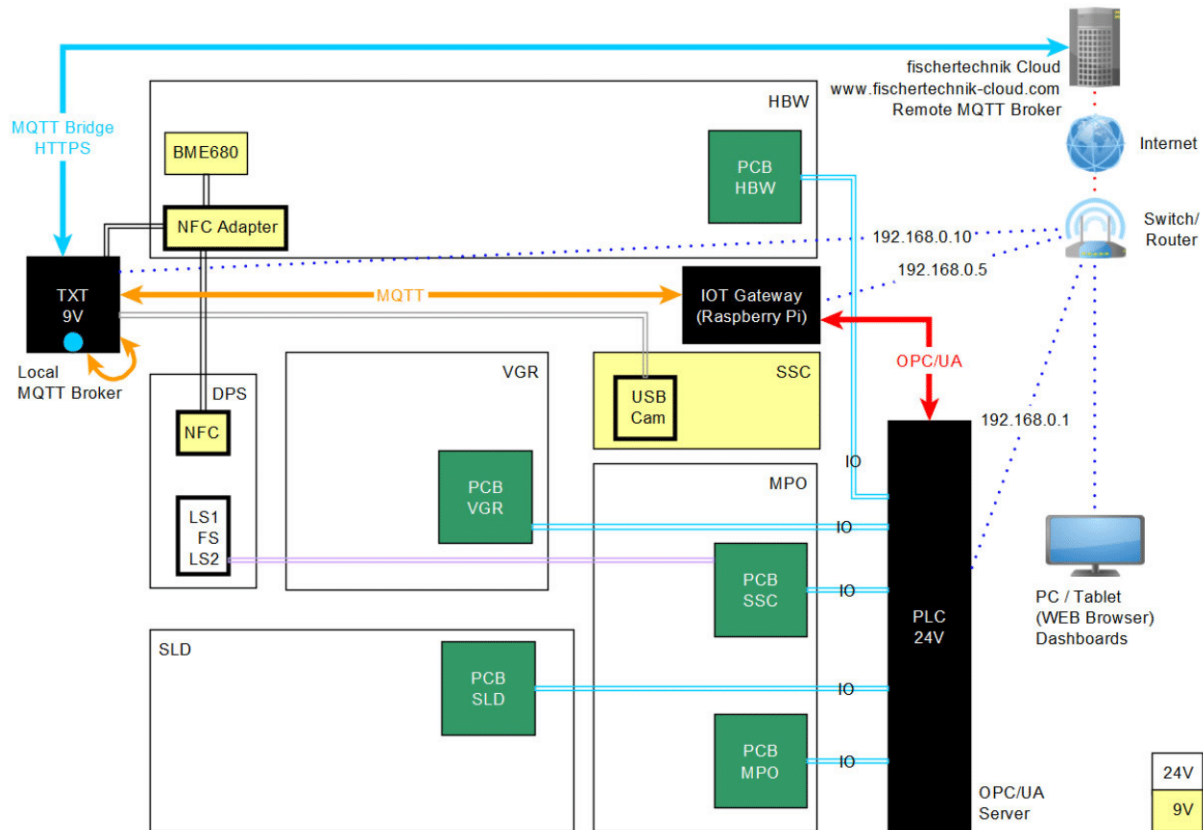


Abbildung 2.2: Schematische Darstellung der Lernfabrik 4.0 24V – Fischertechnik [2]

Kürzel	Bezeichnung
SSC	S ensor S tation with C amera
HBW	H igh- B ay W arehouse
VGR	V acuum G ripper R obot
DPS	D elivery and P ickup S tation
MPO	M ulti- P rocessing S tation with O ven
SLD	S orting L ine with C olor D etection

Tabelle 2.1: Erläuterung der Stationskürzel

Die Kommunikation des Systems wird ab Werk durch eine Siemens S7-1500 SPS, einem TXT-Controller, einem Raspberry Pi und einer aufrechten Ethernet-Verbindung gewährleistet. Die SPS ist für die Signalverarbeitung und Steuerung der Anlagenkomponenten der Lernfabrik zuständig. Der TXT-Controller fungiert als MQTT-Broker, welcher für einkommende MQTT-Nachrichten verantwortlich ist. Diese Nachrichten werden dann per MQTT an das IoT Gateway bzw. den Raspberry gesendet und anschließend wiederum per OPC UA an die SPS gesendet. Durch diesen Datenfluss lassen sich somit auch 9V-Geräte, also die USB-Kamera, der NFC-Reader und der Umweltsensor BME680, ansprechen und letztlich von der SPS auslesen. Für

eine ausführliche Erklärung der einzelnen Bauteile, der dazugehörigen Belegungspläne und der Inbetriebnahme der Fabrik (inklusive Router) siehe [1] ► *Dokumente zum Download* ► *Begleitheft*. [3]

Im vergangenen Forschungsprojekt MINDfactory hat die Fraunhofer Research Austria GmbH bereits Anpassungen an der Software der SPS getroffen, um notwendige Informationen zur Auftragsabwicklung zu generieren und darauf aufbauend eine passende Produktionsplanung zu entwerfen. Dies inkludiert demnach auch die Integration eines leistungsfähigen Mini-PCs, ein sogenannter NUC, auf welchem die notwendigen Microservices laufen und eines Switches für den einfachen Anschluss an das Netzwerk via Ethernet. Die Softwareerweiterung umfasst einen OPC UA Client, eine MongoDB Instanz, eine Mongo Express Webapp, einen Digitalen Zwilling, ein Zustandsüberwachungsprogramm (OEE-App), ein Produktionsplanungsprogramm und ein Dashboard zur Auftragserteilung beziehungsweise Visualisierung der Aufträge, der Lager, der Produktionsplanung, der Planungsqualität und des dazugehörigen Gantt Charts. Mit Hilfe nachstehender Abbildung kann die aktuelle Systemarchitektur der MINDfactory visuell besser aufgegriffen werden: [4]

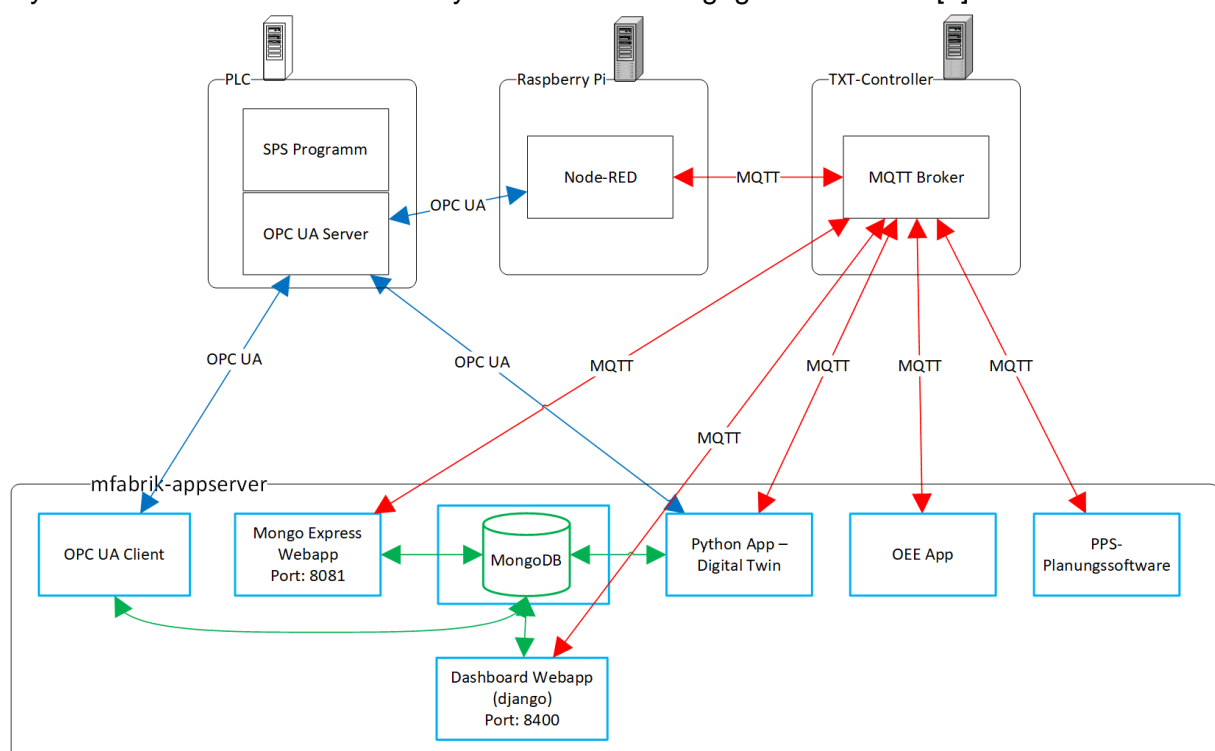


Abbildung 2.3: Systemarchitektur der MINDfactory [4]

Sämtliche Kommunikationsprotokolle und Technologien werden zur Nachvollziehbarkeit kurz erläutert:

CONTAINERVIRTUALISIERUNG

Containervirtualisierung bietet die Möglichkeit mehrere Instanzen eines beliebigen Betriebssystems zeitgleich laufen zu lassen, ohne immense Ressourcen verwenden zu

müssen. Dadurch können einzelne Softwaredienste, auch Microservices genannt, schnell und unkompliziert erstellt werden. Die Kommunikation der jeweiligen Microservices untereinander erfolgt über standardisierte Schnittstellen. Bei der MINDfactory wurde die Virtualisierungstechnologie Docker genutzt. [4,5,6]

OPC UA

OPC UA ist eine herstellerunabhängige Schnittstelle, die in der Industrie für den Datenaustausch zwischen Maschine und Maschine beziehungsweise PC und Maschine zuständig ist. Im Wesentlichen ist OPC UA verantwortlich für die Synchronisierung des Produktionsplans, der Anlagenzustände beziehungsweise der Zustände zwischen SPS und der restlichen Infrastruktur. Somit ermöglicht der OPC UA Client, unter Hinzunahme von MQTT, die Bereitstellung von SPS gesammelten Informationen im Netzwerk. Bei der MINDfactory wurde das Python-Softwarepaket opcua-asyncio verwendet. [7,8]

MQTT

MQTT ist ein Client Server-Publish/Subscribe-Messaging-Transport Protokoll, dass sich durch seine Leichtigkeit und seine unkomplizierte Nutzung auszeichnet. Durch seine rasche Implementierung findet es seinen Einsatz deshalb in Maschine-to-Maschine- und vor allem IoT-Anwendungen. [9] In der Programmierumgebung der MINDfactory wurde Paho-MQTT [10] für die Einrichtung des Clients und Mosquitto [11] für die Einrichtung des Brokers importiert.

MONGODB

Bei MongoDB handelt es sich um eine dokumentorientierte Open-Source No-SQL Datenbank, die im Unterschied zu einer SQL-Datenbank keine gewohnte tabellenartige relationale Datenstruktur aufweist. Die Speicherung der Daten erfolgt im BSON-Format, welche nichts anderes als eine binär codierte Serialisierung von JSON-Dokumenten ist. [12] Im Rahmen dieser Arbeit wurde das Python-Softwarepaket pymongo verwendet. [13]

ModBus

ModBus ist ein Kommunikationsprotokoll für Austausch von Prozessdaten zwischen einem Master und beliebig vielen Slaves. Üblicherweise handelt es sich beim Master um eine Recheneinheit und bei den Slaves um Mess-, Steuer- oder Regelungseinheiten, die per Ethernet-Kabel miteinander verbunden sind. Je nach Anwendungsfall können die Daten per RTU, TCP oder UDP ausgelesen werden, dazu aber in späteren Kapiteln mehr. [14] Im Rahmen dieser Arbeit wurde das Python-Softwarepaket sdm_modbus verwendet. [15]

PLOTLY

Plotly ist ein Open-Source Softwarepaket, das auf der JavaScript-Bibliothek plotly.js basiert und dadurch eine rasche Erstellung von über 40 interaktiven webbasierten Diagrammtypen ermöglicht. Es kommt nicht nur in der Wissenschaft, sondern auch in der Statistik, Geografie, Künstlichen Intelligenz, Bioinformatik oder im Finanzwesen zum Einsatz. [16]

DJANGO

Django ist ein Open-Source Web-Framework in Python, das eine rasche und einfache Umsetzung der Programmierung von Webseiten bietet. Im Gegensatz zu konventionellen Web-Applikationen wird der Programmcode, sprich die Ausführung und Verarbeitung von bestimmten Prozessen, in verschiedene Dateien unterteilt. Diese können anhand von Abbildung 2.4. nachvollzogen werden. [17]

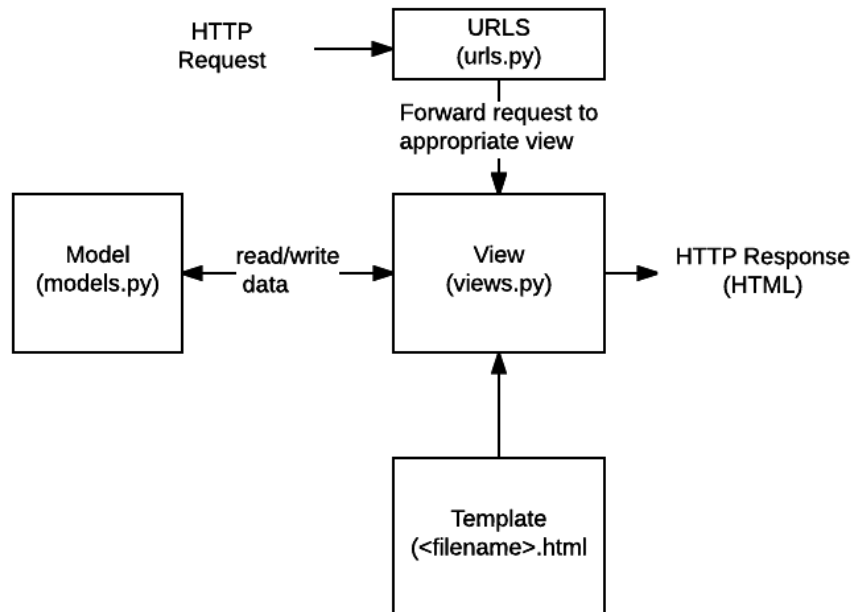


Abbildung 2.4: Schematischer Aufbau von Django-Web-Applikationen [18]

HTTP(-REQUESTS)

HTTP ist ein Client-Server-Protokoll, welches den Aufruf von Ressourcen beziehungsweise den Austausch von Daten im Internet ermöglicht. Nicht nur der Zugriff auf Hypertext-Dokumente, Bilder, Videos und Audio-Dateien, sondern auch das Senden von Daten eines HTML-Formulars an den Server und die Aktualisierung von Webseiten wird gewährleistet. Es bildet damit das Fundament für den Datenaustausch der heutigen Welt. [19]

WEBSOCKETS

Bei WebSockets handelt es sich um ein auf TCP basierendes bidirektionales und interaktives Echtzeitkommunikationsprotokoll, welches eine Verbindung von Webanwendungen und Webservern ermöglicht. Da der Client nur eine Verbindung zum Server benötigt und nicht, wie bei HTTP, auf eine Antwort seiner Anfrage warten muss, werden WebSockets bei Webapplikationen, beispielsweise bei Dashboards zur Echtzeit-Datenvisualisierung, Live-Chats, sozialen Medien oder in Onlinespielen eingesetzt, wo schnelle Ladezeiten unabdingbar sind. Darüber hinaus ist die Größe der einzelnen Nachrichten um hunderte Bytes kleiner, da keine Header definiert werden müssen. Die Vorteile gegenüber HTTP-Requests sind zwar klar ersichtlich, dennoch sind WebSockets kein Ersatz, sondern lediglich eine effizientere und schnellere Methode Daten abrufen zu können. [20]

2.2 Energiemessgerät

Der Eastron SDM230-Modbus ist ein einphasiges Energiemessgerät, welches nach Datenblatt [21] folgende Parameter ermitteln kann:

- Spannung
- Stromstärke
 - aktuelle Stromstärke
 - Strombedarf
- Netzfrequenz
- Leistung
 - aufgenommene und abgegebene Wirkleistung,
 - aufgenommene und abgegebene Blindleistung
 - momentane Wirkleistung
 - momentane Blindleistung
 - Scheinleistung
 - momentane Leistung
 - maximaler Leistungsbedarf
 - Leistungsfaktor

Die gemessenen Werte können sowohl über das integrierte Display als auch über eine RS485-Schnittstelle an einem externen Gerät ausgelesen werden. Doch zu den expliziten Konzepten später in der Methodik mehr.

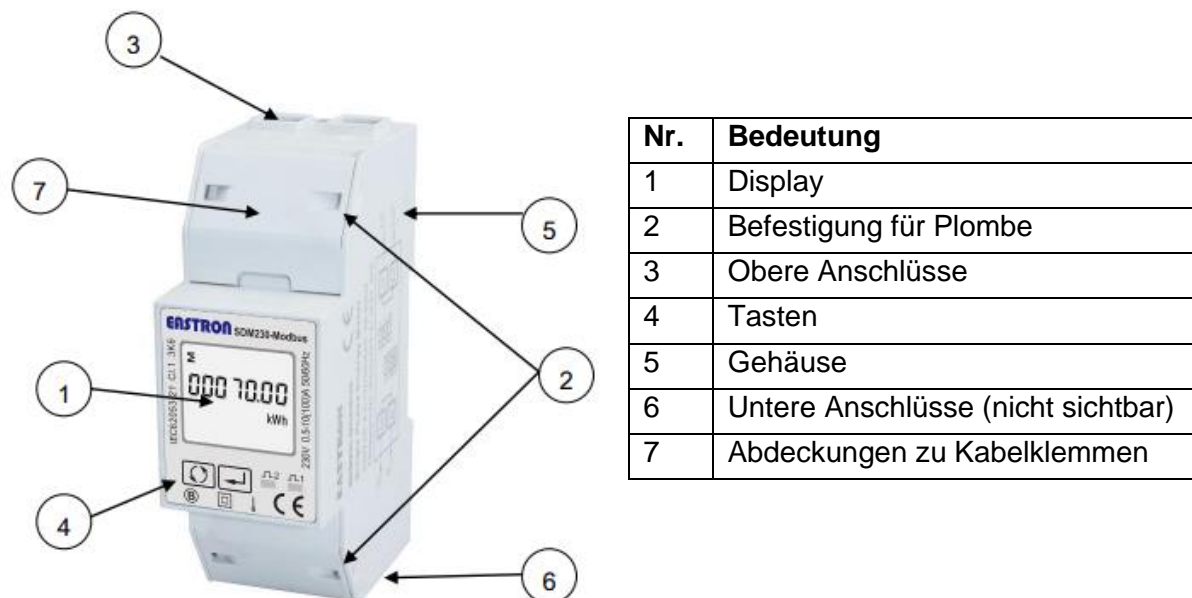


Abbildung 2.5: Eastron SDM230-Modbus [21]

2.3 Photovoltaikanlagen

2.3.1 Allgemeines

Der Einsatz von Photovoltaikanlagen hat sich in den letzten 40 Jahren stark verändert und ist von kleinen bis mittelgroßen autarken Anwendungen im Milliwattbereich, wie Taschenrechnern oder Satelliten, hin zu großen industriellen teils auch netzgekoppelten Kraftwerken im Megawattbereich gewandert. Die Bauformen reichen von 50W-Anlagen für Hobby-Projekte, bis hin zu gebäudeintegrierten Anlagen in Einfamilienhäusern oder gar PV-Freiflächenanlagen zur industriellen Energieerzeugung. Photovoltaikanlagen spielen somit heutzutage eine sehr wichtige Rolle im Bereich der weltweiten Energieerzeugung. In dieser Arbeit liegt der Fokus aber auf sogenannten Inselanlagen, welche den überschüssigen Strom nicht in das elektrische Versorgungsnetz rückspeisen. Inselanlagen werden in mobile, beispielsweise Solarfahrzeuge, und stationäre Systeme, wie Trinkwasserpumpen oder Signalanlagen, unterteilt. Größere Inselanlagen beinhalten in der Regel auch eine Batterie zum Ausgleich von Angebot und Nachfrage der Energie. [26]

In der Photovoltaik werden Solarzellen genutzt, um Anteile der Strahlungsenergie in elektrische Energie umzuwandeln. Obwohl es teilweise für Testzwecke speziell entwickelte PV-Strahlungsgeneratoren gibt, wird im Regelfall die Strahlung der Sonne gemeint. Durch Fusionsprozesse im inneren der Sonne wird Energie erzeugt, welche grob vereinfacht als Abstrahlung in das Weltall freigesetzt wird. Sobald diese dann auf die Erdatmosphäre trifft, bewirken Streu- und Reflexionsprozesse eine Teilung der Globalstrahlung in gerichtete Direktstrahlung und ungerichtete Diffusivstrahlung. Ein Blick auf die Strahlungsbilanz der Erde zeigt, dass nur knapp 47% der Solareinstrahlung bei Eintritt der Erdatmosphäre wirklich auf der Erde ankommt. [26]

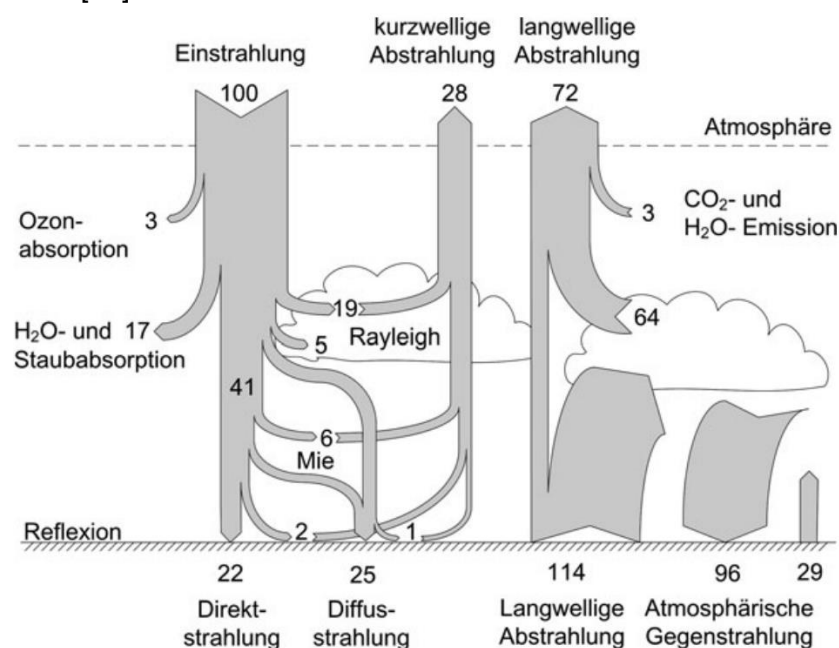


Abbildung 2.6: Strahlungsbilanz der Erde [26]

2.3.2 Marktentwicklung in Österreich

Die Erhebung und Auswertung der Marktentwicklung österreichischer Photovoltaiksysteme erfolgt bereits seit dem Jahr 1990. Anfangs wurden die Daten stichprobenartig mit Hilfe von Befragungen verschiedener Anlageplaner und Produktionsfirmen erhoben. Doch seit dem Jahr 2009 ist dies aufgrund einer signifikanten Marktdiffusion nicht mehr möglich, weshalb zusätzlich eine jährliche Befragung der Abwicklungsstelle für Ökostrom (OeMAG), der Klima- und Energiefonds (KLIEN), der Kommunalkredit Public Consulting (KPC) und der Landesförderstellen abgehalten werden muss. [27]

Der Trend der erneuerbaren Energien im Bereich der Photovoltaik hat sich in Österreich erst im Jahre 2008 etablieren können. In der Zeitspanne von 2000 bis 2013 hat das Interesse stetig zugenommen. Gefolgt von einigen Jahren Seitwärtsbewegung konnte Österreich im Jahre 2020 erstmals seit der rapiden Abnahme im Jahr 2013 einen Anstieg an jährlich installierter PV-Leistung verzeichnen. Doch die zuletzt dokumentierte Zunahme im Jahre 2021 von 340 auf 760 MW Peak jährlich installierter PV-Leistung, welche sich auf knapp +95% belief, steht kaum in Relation dazu. [27]

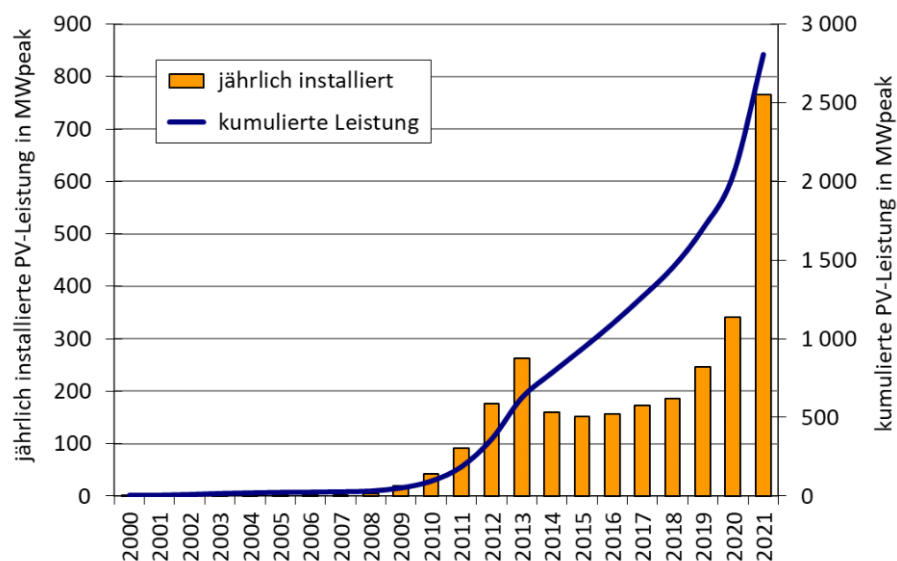


Abbildung 2.7: Die Marktentwicklung der Photovoltaik in Österreich bis 2021 [27]

Ein Blick auf die Entwicklung der Speicherkapazität von in Photovoltaikanlagen verbauten Batterien zeigt, dass das Interesse an sogenannten Inselsystemen, welche unabhängig vom Netzstrom betrieben werden können, seit dem Jahre 2014 stetig zu nimmt. Obwohl im Jahr 2020 neue prozentuale Wachstumsrekorde verzeichnet werden konnten, hat das Jahr 2021 einen stärkeren Anstieg an jährlichem Zubau darlegen können. Da die kumulierte jährlich neu installierte nutzbare Speicherkapazität in nachstehender Grafik beinahe einer Exponentialfunktion ähnelt, kann durchaus behauptet werden, dass neben netzgekoppelten Systemen auch Inselsystemen, relevanter denn je sind. [27]

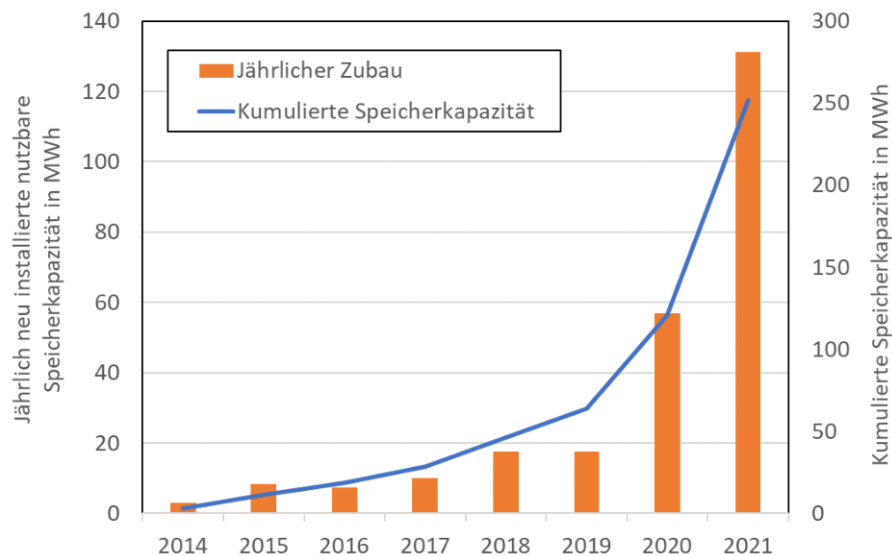


Abbildung 2.8: PV-Batteriespeicherkapazität in MWh von 2014 bis 2021 [27]

2.3.3 Komponenten

Grundsätzlich besteht eine Photovoltaik immer aus mindestens den folgenden sechs Komponenten:

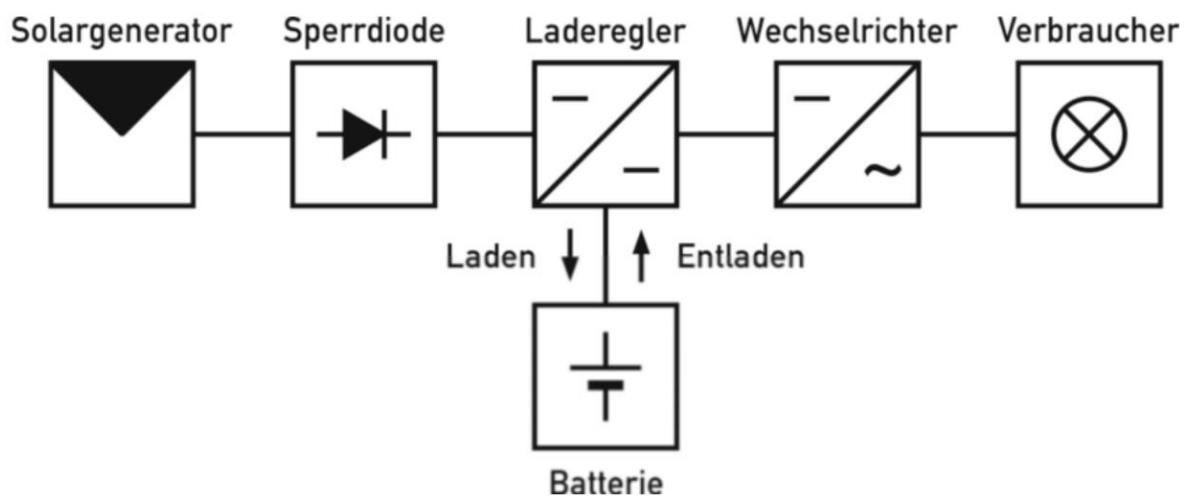


Abbildung 2.9: Blockschaltbild eines photovoltaischen Inselsystems [28]

SOLARGENERATOR

Der Solargenerator, auch Photovoltaikgenerator genannt, bildet das Fundament einer Photovoltaikanlage. Er besteht aus dem Solar-Panel, also mehreren in Reihe geschalteten Solarzellen, der dazugehörigen Aufständerung zur optimalen Ausrichtung des Einstrahlwinkels und der Gleichstromverkabelung. Solarzellen kommen üblicherweise ohne aufwendige Wartung aus, da sie keine verschleißbaren Teile besitzen und ihre Lebensdauer meist sehr hoch ist. Dies spiegelt sich auch in der Herstellergarantie wider, welche sich oftmals auf mindestens 20 Jahre beläuft. Die bekanntesten Bauformen reichen von monokristallinen, hin zu polykristallinen Silizium-Solarzellen, Dünnschicht-Solarzellen aus amorphem Silizium

und CIGS-Solarzellen. Die Aufständerung kann entweder festmontiert sein oder ein beziehungsweise zwei Achsen zur automatischen Verfolgung der Sonne haben. Diese sogenannten Tracker-System, also jene Systeme mit Achsen, haben zwar eine Ertragssteigerung und verursachen eine geringere Flächenversiegelung, sind aber wesentlich teurer und wartungsintensiver als fixe Systeme. [26]

SPERRDIODE

Die Aufgabe der Sperrdiode ist es, die Rückspeisung der elektrischen Energie beispielsweise bei Verschattung oder Verschmutzung zu vermeiden. Sie befindet sich in den heutigen Systemen meist bereits in den Verbindungssteckern der Gleichstromkabel. [28]

LADEREGLER

Wie der Name dieses Bauteils schon vermuten lässt, ist der Laderegler für die Überwachung beziehungsweise die Regelung des Batterieladezustandes verantwortlich und schützt sie damit vor Tiefentladungen und Überladungen. [28]

BATTERIE

Die Batterie wird genutzt, um den überschüssigen, nicht benötigten Strom für sonnenarme Tagesabschnitte, beispielsweise in der Nacht, nutzen zu können und somit Angebot und Nachfrage der Energie auszugleichen. Die derzeit meistverbreiteten Batterien sind Bleisäure-Batterien, Blei-Gel-Batterien und Lithium-Ionen-Batterien, welche jedoch so ausgewählt werden müssen, dass diese mit den Herstellerangaben des Ladereglers kompatibel sind. [28]

WECHSELRICHTER

Um schlussendlich aber Verbraucher an das Photovoltaiksystem anschließen und vor allem ordnungsgemäß betreiben zu können, muss die Gleichspannung des Solargenerators beziehungsweise der Batterie auf die Nenn-Wechselspannung des Verbrauchers gebracht werden. [28]

VERBRAUCHER

Unter einem Verbraucher wird das von der Photovoltaikanlage zu betreibende Endgerät verstanden. Hier ist besonders auf die erforderliche Geräteleistung zu achten, da bei unzureichendem Energieertrag des Solargenerators oder insuffizienten Batterieladezustand kein ordentlicher Betrieb gewährleistet werden kann. [28]

2.3.4 Einflussfaktoren auf Energieerträge

Neben einer übersichtlichen Visualisierung ist eine ordnungsgemäße Interpretation der Ergebnisse essenziell, um die Ursache von möglichen Spitzen (Maxima) oder Einbrüchen (Minima) erkennen zu können. Hierzu ist es notwendig von Grund auf eine Darlegung der verschiedenen Einflussfaktoren auf Energieerträge durchzuführen. Die Einflüsse können einerseits in geographische und andererseits in technische Faktoren unterteilt werden:

2.3.4.1 Geographische Faktoren

WITTERUNGSVERHÄLTNISSE

Der Ertrag einer Photovoltaikanlage ist, wie bereits in der Einleitung erwähnt, grundsätzlich durch die Einstrahlung des Sonnenlichts limitiert. Doch weitere Witterungsverhältnisse werden leider oftmals nicht berücksichtigt, weshalb deren Einfluss hier kurz erläutert wird:

- **Umgebungstemperatur:**

Der Wirkungsgrad einer Photovoltaikanlage hängt nicht nur von der Abstimmung der einzelnen Komponenten untereinander, sondern auch von der jeweiligen Empfindlichkeit auf extreme Temperaturen ab. Mit steigender Umgebungstemperatur nimmt der Wirkungsgrad zu und demnach auch der Ertrag. Besonders in sehr heißen Gebieten empfiehlt es sich dennoch den Wirkungsgrad zu überwachen und anfällige Bauteile gegebenenfalls zu kühlen. Dasselbe gilt auch in sehr kalten Arealen, aber nur mit dem Unterschied einer Heizung statt einer Kühlung. Achtung: Eine Kühlung/Beheizung der Komponenten empfiehlt sich nur, wenn auch die Wirtschaftlichkeit gegeben ist. Dies kann beispielweise durch eine Nutzwertanalyse garantiert werden.

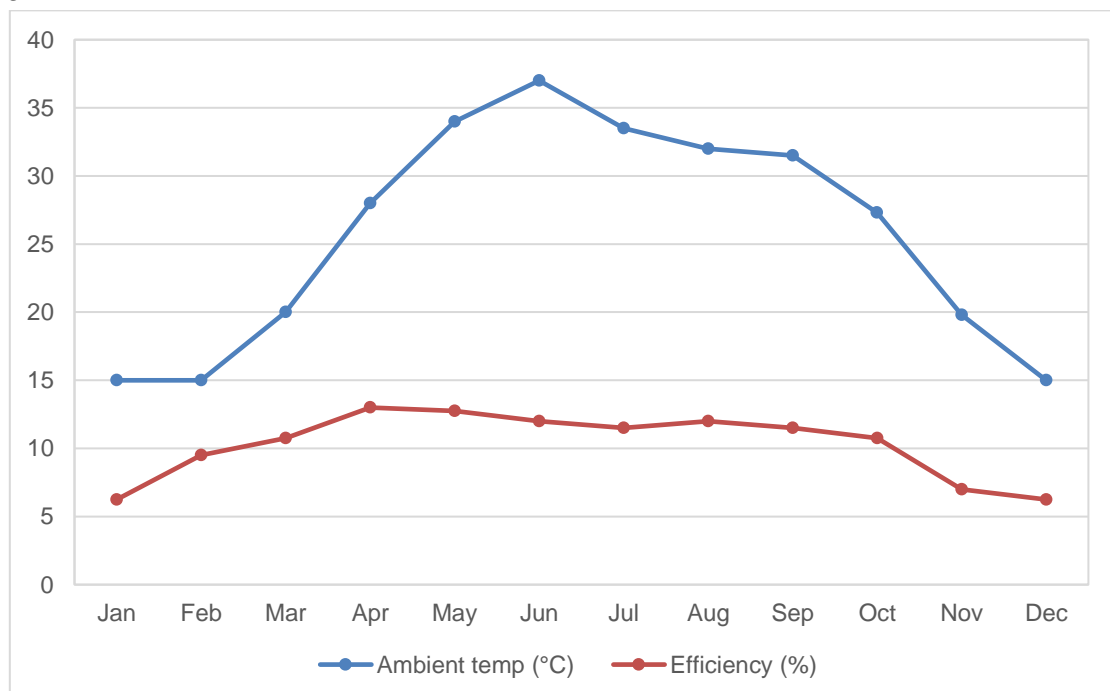


Abbildung 2.10: Einfluss der Umgebungstemperatur auf den Wirkungsgrad [29]

- **Relative Luftfeuchtigkeit**

Die Luftfeuchtigkeit hat im Vergleich zu anderen Witterungsverhältnissen meist einen vernachlässigbaren Einfluss auf den Energieertrag. Bei hohen Messwerten kann der Wasserdampfgehalt der Luft jedoch zu einer Streuung der Strahlung und demnach zu einer Reduzierung des Wirkungsgrades beziehungsweise des Ertrags führen.

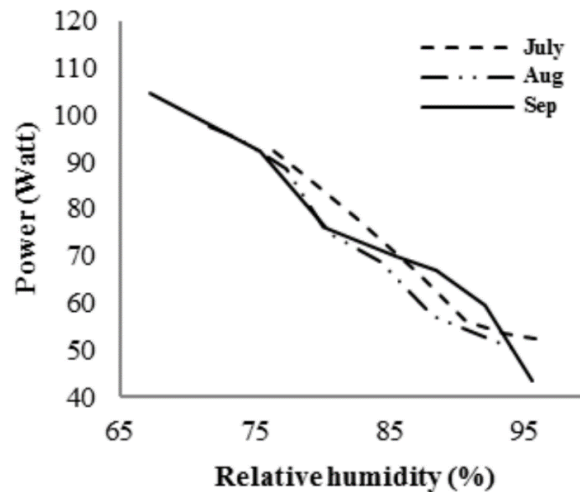


Abbildung 2.11: Einfluss der relativen Luftfeuchtigkeit auf die Ausgangsleistung [30]

- **Windstärke**

Eine gemäßigte Windstärke kann als Kühlung der Komponenten, vor allem der Solarzellen, dienen und eine Steigerung des Wirkungsgrads beziehungsweise der Leistung mit sich ziehen. In Regionen mit sehr hohen Windstärken sollte wiederum auf die korrekte Befestigung der Komponenten geachtet werden, um Defekte oder zeitbeziehungsweise kostenintensive Wartungen vorzubeugen.

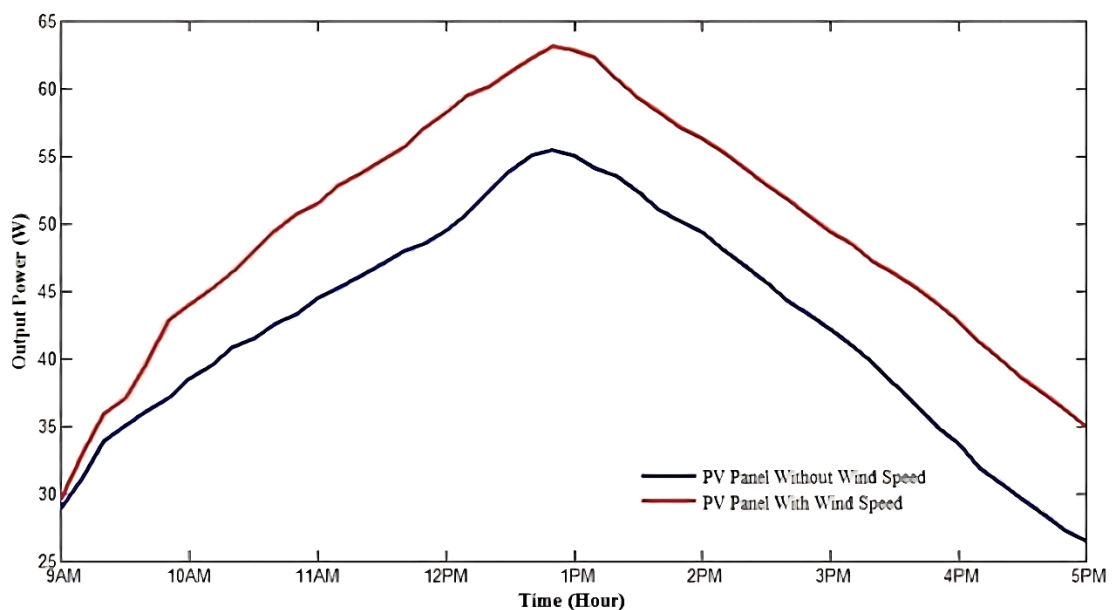


Abbildung 2.12: Einfluss des Windes auf die Ausgangsleistung [31]

- **Wolkenbildung**

Während hochstehende Wolken als Linse wirken und so das Licht in bestimmten Bereichen konzentrieren, verhindern tiefhängende Wolken eine direkte Sonneneinstrahlung. Je nach Wolkentyp kann demnach der Ertrag erhöht, aber typischerweise eher minimiert werden.

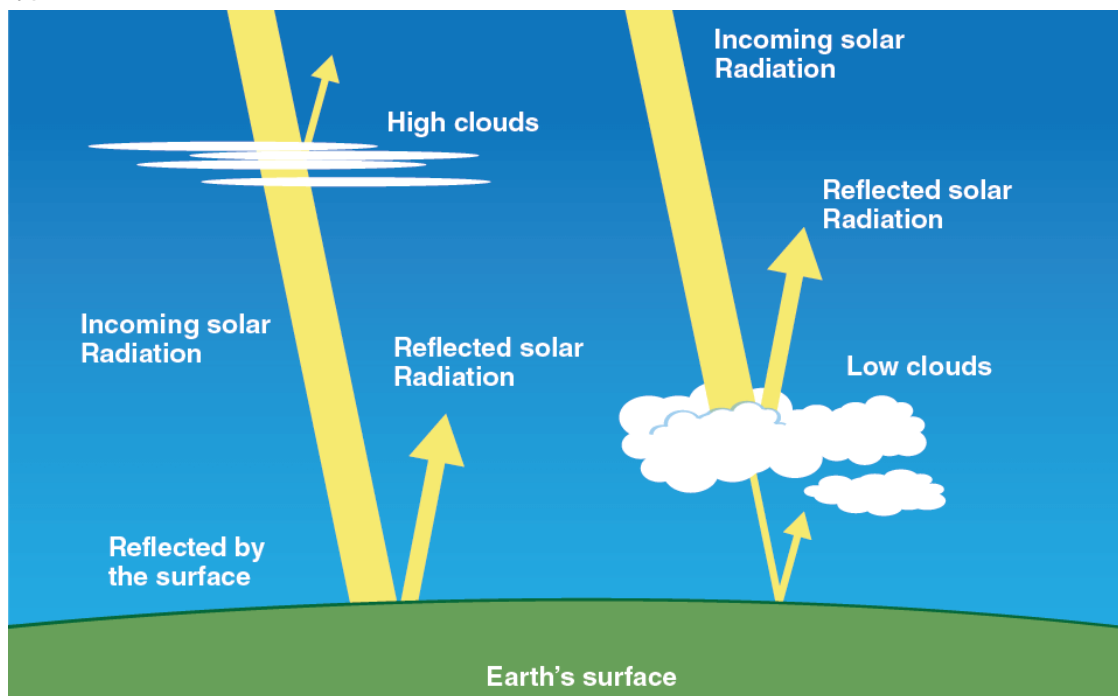


Abbildung 2.13: Einfluss des Wolkenstandes auf die Sonneneinstrahlung [32]

- **Verschmutzung**

Häufig lagert sich Laub, Staub, Sand oder etwaiger Schmutz auf Solarzellen ab und verursacht dadurch eine Ertragsverminderung. Teilweise werden Verunreinigungen zwar von Regen und Wind beseitigt, aber hinterlassen dennoch spürbare Einbußen:

Art der Verschmutzung	Maximale Leistung	Temperatur des PV-Panels
Sauber	40,96 W	44,8 °C
Vogelkot	36,65 W	40,4 °C
Sand	37,42 W	40,6 °C
Zementstaub	34,16 W	40,2 °C
Kohlenstaub	27,46 W	39,7 °C

Tabelle 2.2: Leistung und Temperatur bei Verschmutzungen auf PV-Panel [33]

LAGE

In der Regel ist die Aufständerung der Solarzelle fest, insofern es sich nicht um ein Nachführsystem handelt. Hierbei wird zwischen dem Winkel α , welcher die Verdrehung gegen Südrichtung beschreibt, und dem Winkel β , welcher gegenüber der Horizontalen angestellt ist, unterschieden.

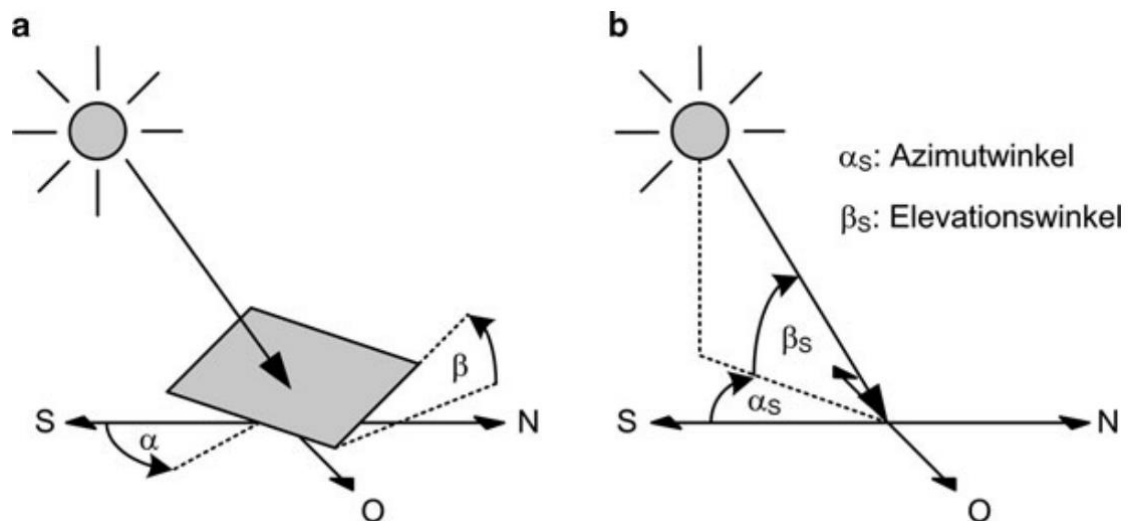


Abbildung 2.14: Erläuterung von Azimut- und Elevationswinkel [26]

Im rechten Teil der obigen Abbildung ist der Azimutwinkel und der Elevationswinkel der einfallenden Sonnenstrahlung gekennzeichnet. Nachstehende Grafik stellt den Zusammenhang dieser Parameter mit dem Tagesverlauf des Sonnenstandes visuell dar:

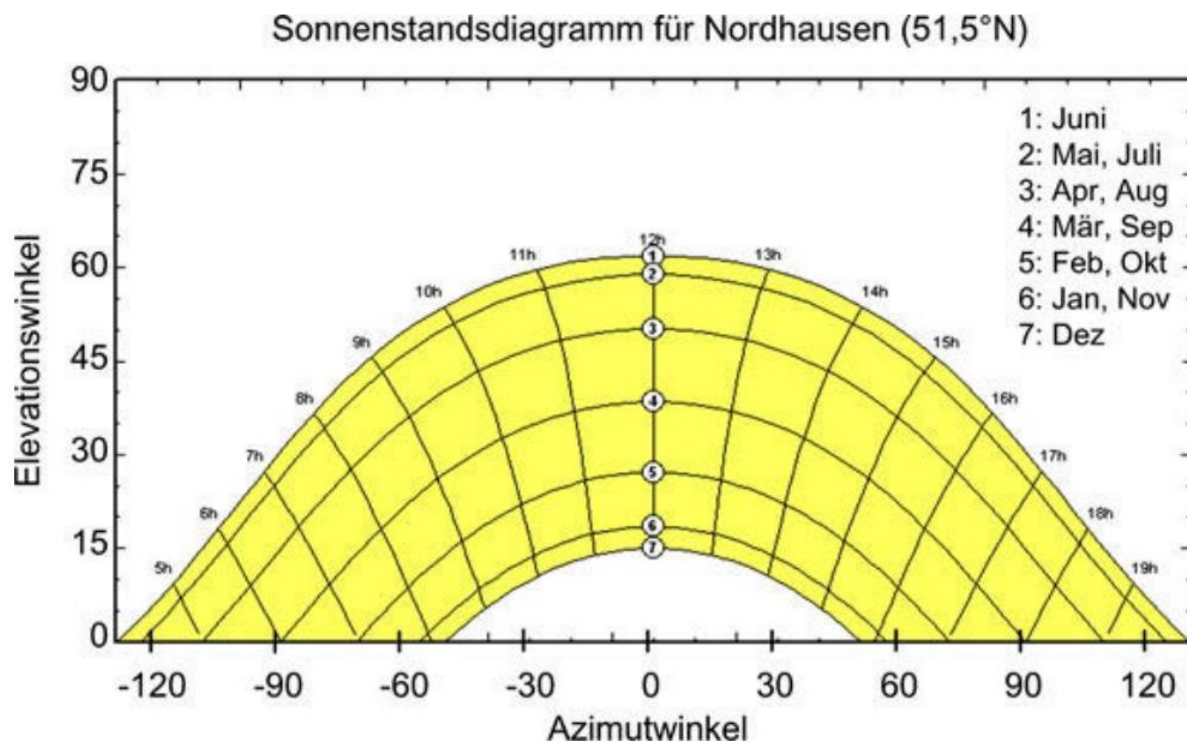


Abbildung 2.15: Sonnenstandsdiagramm zur Verschattungsanalyse [26]

Das Sonnenstandsdiagramm wurde für den Standort Nordhausen mit den Koordinaten 51,5°N, 10,8°O erstellt und zeigt den Verlauf des Sonnenstandes des dafür maßgeblichen Breitengrades. Auf der Abszisse ist der Azimutwinkel in Grad [°] und auf der Ordinate der Elevationswinkel in Grad [°] angegeben. Bei den repräsentativen Tageskurven, welche rechts oben beschreiben sind, handelt es sich um den 21. des jeweiligen Monats. In Sonnenstandsdiagrammen können auch Verschattungen von Objekten, wie Gebäuden oder Bäumen, berücksichtigt werden, unter der Voraussetzung, dass sie vor Ort in Bezug auf Elevationswinkel und Azimutwinkel vermessen wurden.

Um eine korrekte Interpretation des Diagramms zu garantieren, wird der Monat Februar exemplarisch erklärt:

- **Sonnenaufgang:**
Um 07:00 Uhr bei einem Azimutwinkel von -70° und einem Elevationswinkel von 0°.
- **Höchststand:**
Um 12:00 Uhr bei einem Azimutwinkel von 0° und einem Elevationswinkel von 25°.
- **Sonnenuntergang:**
Um 17:00 Uhr bei einem Azimutwinkel von 70° und einem Elevationswinkel von 0°.

Zusammengefasst kann also gesagt werden, dass eine fundierte Ertragsoptimierung viele ortsabhängige Parameter beinhaltet. Da einige Faktoren des Witterungsverhältnisses und der Lage einander beeinflussen beziehungsweise implizieren, dürfen die einzelnen Kennwerte nicht getrennt voneinander, sondern müssen als Ganzes betrachtet werden. Deshalb ist in jedem Fall eine individuelle Evaluierung der geographischen Faktoren des lokalen Standortes zu empfehlen.

2.3.4.2 Technische Faktoren

BAUFORM

Der Wirkungsgrad einer Photovoltaikanlage hängt von allen Komponenten des Systems ab, wird aber maßgeblich vom Solargenerator beziehungsweise vom PV-Modul beeinflusst:

- **Peripherie**
Nicht nur der Wirkungsgrad des PV-Moduls, sondern auch der des Wechselrichters, des Ladereglers, des Speichers beziehungsweise der Verkabelung ist zu beachten. Während der Wirkungsgrad eines kristallinen PV-Moduls bei ca. 16% bis 18% ist, ist er Vergleich dazu für Wechselrichter, Speicher und Verkabelung meist vernachlässigbar. Meistens liegt der Wirkungsgrad einer PV-Anlage deshalb sehr nahe unter dem des PV-Moduls.[34]

Deshalb sollte bei der Nachrüstung oder Optimierung eines PV-Moduls auch eine Analyse des gesamten Systems durchgeführt werden. Bei der Verkabelung ist grundsätzlich darauf zu achten, den richtigen Querschnitt zu wählen und die Kabellänge als kurz als möglich zu halten.

- **Art des PV-Moduls**

Prinzipiell kann zwischen den standardmäßigen einseitigen, sogenannten monofazialen, und den beidseitigen beziehungsweise bifazialen PV-Modulen unterschieden werden. Heutzutage werden aufgrund der geringeren Anschaffungskosten und dem einfachen Aufbau vorrangig monofaziale Ausführungen eingesetzt. Doch unter der Annahme, dass bifaziale Module einer ähnlichen Preisentwicklung, wie der von monofazialen Modulen des letzten Jahrzehnts folgen, könnte die Ertragssteigerung von in etwa 30% bald an Relevanz gewinnen.

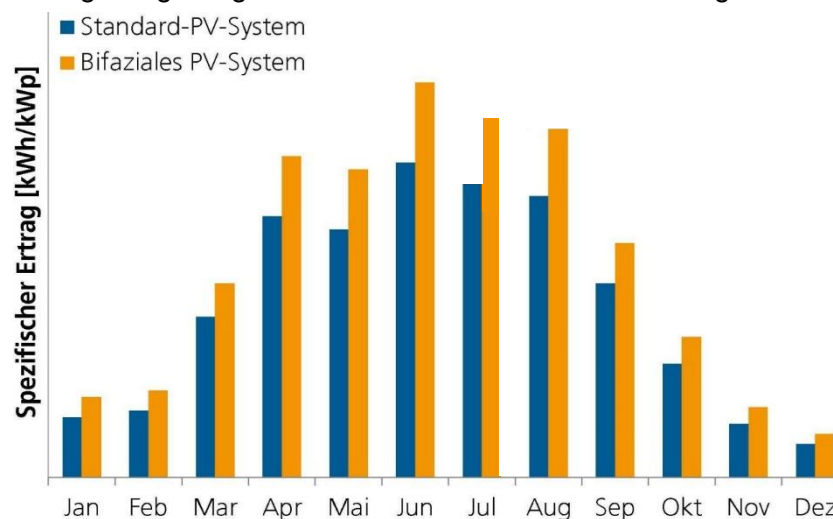


Abbildung 2.16: Spezifische Erträge von Standard- und Bifazialen PV-Systemen [35]

Dies bestätigt sich auch beim Vergleich der Weltmarktanteile von mono- und bifazialen Modulen. Anzumerken ist, dass es sich ab dem Jahr 2023 zwar um Prognosen handelt, aber der Anstieg der bifazialen Module von knapp 18% im Jahr 2020 auf ca. 28% im Jahr 2021 guten Grund für solch eine Zuversicht bietet.

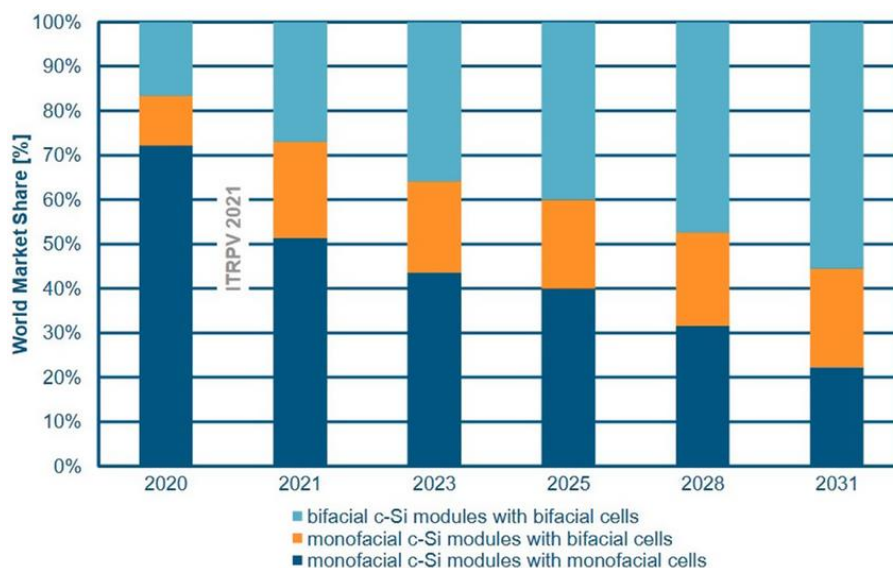


Abbildung 2.17: Weltmarktanteile von mono- und bifazialen PV-Modulen [36]

- **Größe des PV-Moduls**

Wie die Einheit der Global-, Direkt- beziehungsweise Diffusivstrahlung, nämlich Watt pro Quadratmeter, schon vermuten lässt, trägt die abgedeckte Fläche des PV-Moduls einen essenziellen Beitrag zur Steigerung des Ertrages bei. Das heißt, desto mehr Fläche durch die verbauten PV-Module abgedeckt wird, desto höher der Ertrag.

Achtung: Größer heißt nicht unbedingt besser! Neben der Größe sollte unbedingt Augenmerk auf die Anschaffungskosten und die maximale Leistung gelegt werden.

ADJUSTIERUNG

- **Anordnung der PV-Module**

Vor allem bei Freifeldanlagen ist die Berücksichtigung der Verschattung der PV-Module, erzeugt durch Neigungswinkel und Reihenabstand, bedeutsam. Ein marginaler Neigungswinkel führt unmittelbar zu einem kleinerem Reihenabstand, wodurch der Flächennutzungsfaktor zwar steigt, aber die Globalstrahlungssumme aufgrund von suboptimaler Ausrichtung sinkt. Um den Zusammenhang zwischen Neigungswinkel und Reihenabstand aufweisen zu können, wurden für eine Freifeldanlage in Oberhausen die spezifischen Energieerträge bei unterschiedlichen Neigungswinkeln simuliert. Als Referenzwert wurde ein Winkel von 30° gewählt und der Reihenabstand beträgt in sämtlichen Simulationen der dreifachen Modulbreite:

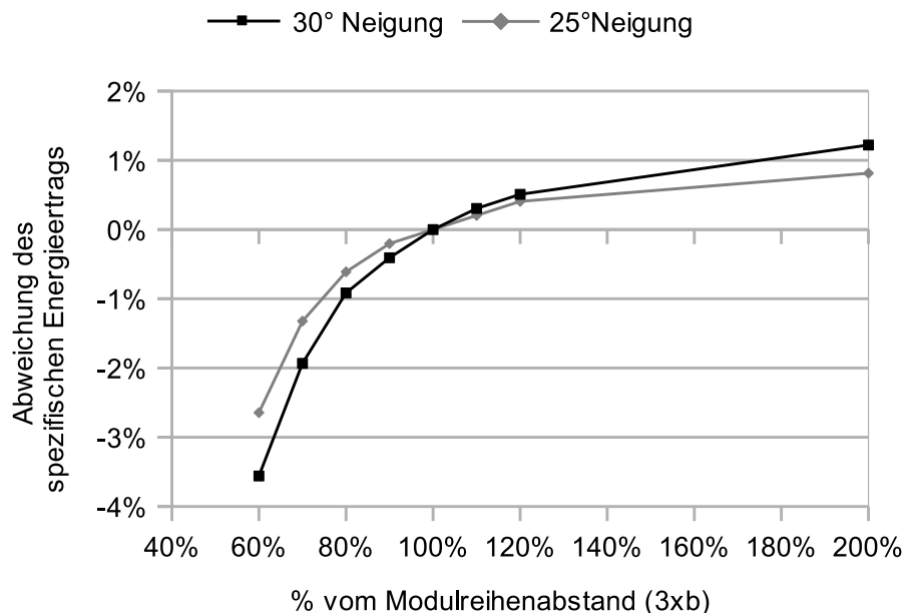


Abbildung 2.18: Spezifischer jährlicher Energieertrag bei variierendem Modulreihenabstand [26]

Während eine Vergrößerung des Reihenabstandes kaum einen Einfluss auf den Energieertrag hat, verursacht eine Verminderung eine verschattungsbedingte Ertragsreduktion. Diese ist jedoch bei kleinerem Neigungswinkel minimaler, weshalb bei Verringerung des Modulreihenabstands auch der Neigungswinkel verringert werden sollte. [26]

- **Art der Aufständerung**

Sowohl der Vergleich von mono-, als auch bifazialen PV-Modulen mit beziehungsweise ohne Nachführsystemen zeigt, dass ein Nachführsystem zu einer dauerhaften Ertragssteigerung führt. Für $\frac{3}{4}$ des Jahres, um genauer zu sein in den Monaten März, Mai, Juni, Juli, August, September, Oktober und November schneidet das einseitige Modul mit Nachführsystem besser ab als ein beidseitiges Modul ohne Nachführsystem.

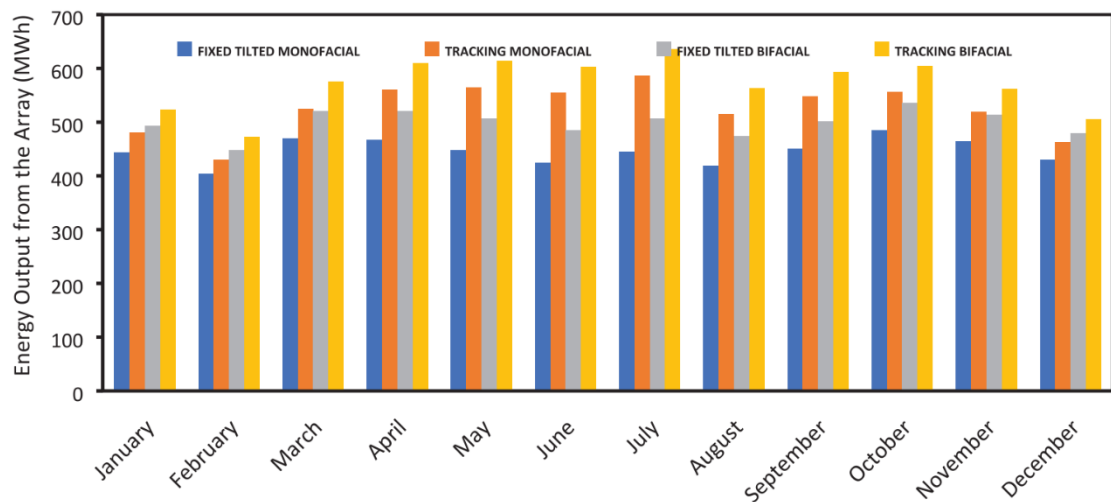


Table 6. Fixed inclined monofacial and bifacial module geometry.

Parameter	Value
Albedo	0.4
Ground clearance height	1.5
Module width	3.04 m
Module length	1.8 m
Tilt angle	15°
Module azimuth	0

Table 7. Horizontal axis tracking monofacial and bifacial module geometry.

Parameter	Value
Albedo	0.4
Array height	1.3–1.8 m
Module width	3.04 m
Axis Azimuth	0
Tilt Angle	Vary
Tracking range of motion	± 60 °
Ground cover ratio	0.338

Abbildung 2.19: Energieerträge von verschiedenen Aufständerungen [37]

Demnach kann gesagt werden, dass ertragstechnisch die Integration eines automatischen Trackingsystems einem Umstieg von mono- auf bifazialen Modulen vorzuziehen ist. Die anfallenden Kosten und Umrüstungszeiten für jeglichen Umstieg sollten jedenfalls individuell berücksichtigt werden.

Neben Freifeldanlagen sollten, vor allem in Hinsicht auf den Einsatz von Photovoltaikanlagen in der Fabrikplanung, auch gebäudeintegrierte Anlagen berücksichtigt werden. Anhand folgender Grafik können die verschiedenen Ausführungen visuell aufgefasst und deren jeweilige Vor- beziehungsweise Nachteile nachvollzogen werden:



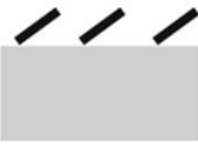



	Schrägdach Aufdach	<ul style="list-style-type: none"> + keine Eigenverschattung + vollständige Flächennutzung - Ausrichtung nicht immer optimal
	Schrägdach In-Dach	<ul style="list-style-type: none"> + keine Eigenverschattung + vollständige Flächennutzung - nur für Dachneubau geeignet - schlechte Hinterlüftung
	Flachdach Aufdach	<ul style="list-style-type: none"> + optimale Neigung + optimale Ausrichtung + gute Hinterlüftung - Eigenverschattung
	Flachdach In-Dach	<ul style="list-style-type: none"> - keine Selbstreinigung - ungünstige Ausrichtung - schlechte Hinterlüftung
	Fassade	<ul style="list-style-type: none"> + Nutzung als Sonnenschutz mit teiltransparenten Modulen + keine Eigenverschattung - ungünstige Ausrichtung - schlechte Hinterlüftung
	Sonnenschutz- elemente	<ul style="list-style-type: none"> + Nutzung als Sonnenschutz + optimale Ausrichtung + gute Hinterlüftung - Eigenverschattung

Abbildung 2.20: Arten von gebäudebezogenen Anlagen nach [38]

3 Methodik

Obwohl der Fokus dieser Arbeit eigentlich auf der Entwicklung beziehungsweise der Implementierung einer passenden Python-Software liegen soll, wird verständnishalber auch oberflächlich auf die Elektronik eingegangen. Die Einrichtung der MINDfactory, der SPS, des Servers (NUC) und des Raspberry Pis wird im Rahmen dieser Arbeit nicht erläutert.

Um die grundlegende Vorgehensweise dieser Arbeit nachvollziehen zu können, ist folgende Abbildung essenziell:

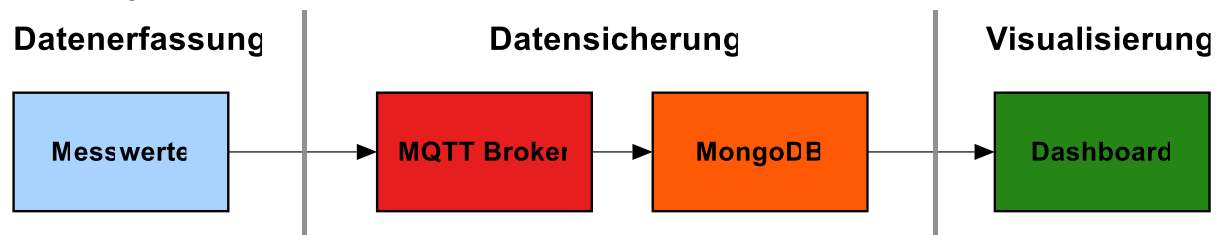


Abbildung 3.1: Blockschaltbild zu Illustration der Herangehensweise dieser Arbeit

Um den roten Faden dieser Arbeit gewährleisten zu können, wird die Einteilung der Methodik deshalb kurz erläutert. Das Kapitel wird in folgende drei Kategorien gegliedert:

1. DATENERFASSUNG

Grundlegend für die Datenerfassung ist die Auswahl der elektronischen Komponenten des Energiemessgerätes und der Photovoltaikanlage beziehungsweise die damit verbundene Verkabelung/Montage. Danach kann einerseits die Programmierung des Raspberry Pis zur Evaluierung der von der Photovoltaikanlage erzeugten Leistung anhand von Spannung und Stromstärke ermittelt und andererseits die verbrauchte Leistung über das Netzwerk ausgelesen werden. Diese erhobenen Daten werden dann mittels MQTT an den TXT-Controller, welcher in der MINDfactory als MQTT-Broker fungiert, gepublished.

2. DATENSICHERUNG

Bevor die Daten dann aber im Dashboard visualisiert oder abgerufen werden können, müssen sie vom MQTT-Broker bezogen beziehungsweise subscribed werden. Danach können die Messwerte gegebenenfalls aufbereitet und zur späteren Einsicht oder nachträglichen Interpretation in der MongoDB-Datenbank gesichert werden.

3. VISUALISIERUNG

Der letzte Schritt vor der Visualisierung im Dashboard ist das Einlesen der Daten aus MongoDB und der externen APIs. Die Daten werden hier nicht direkt vom MQTT-Broker bezogen, da ansonsten Datenreihen der Vergangenheit nicht aufgefasst werden können.

3.1 Datenerfassung

Die Datenerfassung teilt sich in zwei zentrale Punkte auf. Einerseits handelt sich um die PV-Anlage und andererseits um die MINDfactory. Die Abgrenzung dieser beiden Einrichtungen ist notwendig, da die Bereitstellung der Daten unterschiedlich ist. Während bei der PV-Anlage noch keinerlei Sensorik verbaut ist, sind Umweltsensor BME 680 und Fotowiderstand LDR03 bereits in der MINDfactory integriert. Das Veröffentlichen der Daten dieser zwei Sensoren zu dem MQTT-Broker erfolgt bereits werkseitig via Node-Red und muss deshalb lediglich kontrolliert werden.

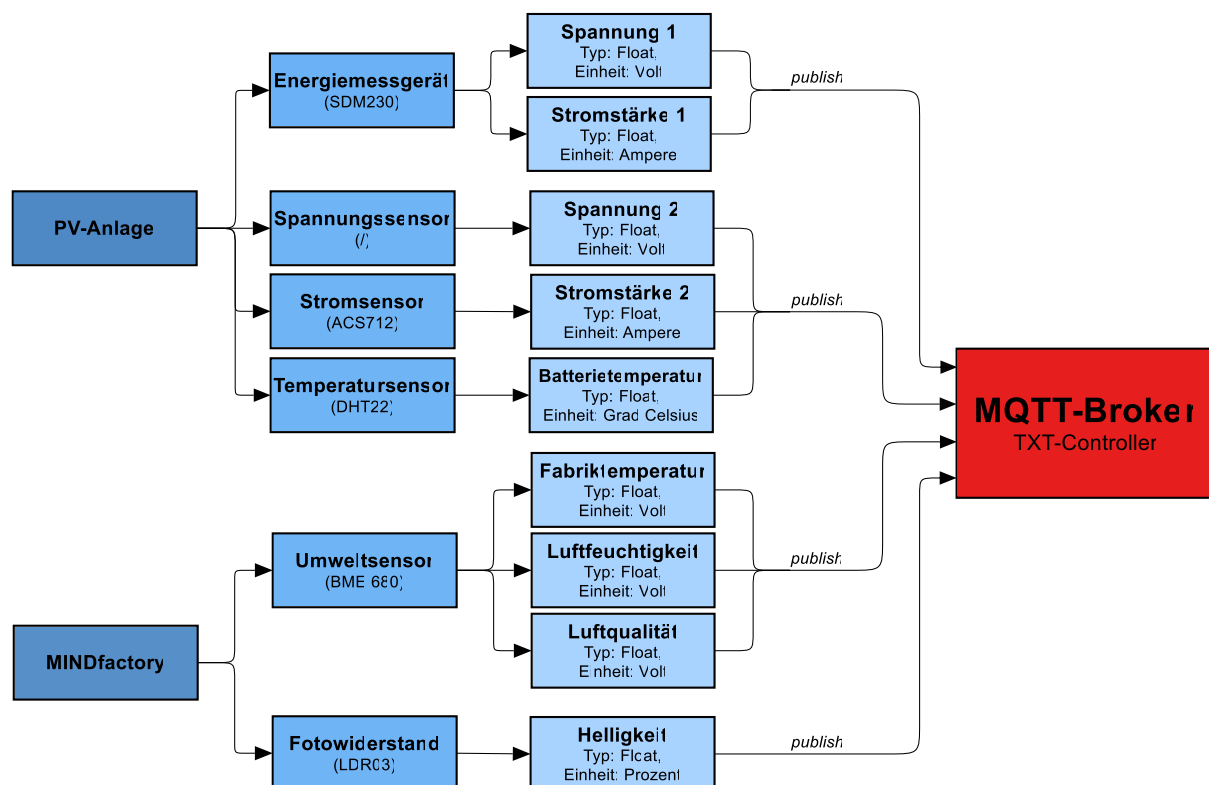


Abbildung 3.2: Blockschaftbild der Datenerfassung

Da zur Erfassung der realen Messwerte der PV-Anlage die gesamte Elektronik konzipiert beziehungsweise verkabelt werden muss und dadurch eine immense Verzögerung der weiteren Arbeitsschritte verursachen würde, wird hier eine Möglichkeit zur parallelen Bearbeitung der Programmierung berücksichtigt. Deshalb wird hier anstelle der Sensorik ein Python-Programm als Placeholder entwickelt, welches beliebig definierbare imaginäre Datenreihen ausgibt und so die reale Datenerfassung vorerst ersetzt. Für einen späteren und vor allem komplikationsarmen Austausch der imaginären durch reale Messwerte ist der Datentyp von hoher Relevanz. Aus diesem Grund sollte, vor der Programmierung des Grundgerüsts, unbedingt das Datenblatt des jeweiligen Sensors sorgfältig durchgelesen werden. Der Datentyp eines Messwertes kann nämlich je nach Sensor neben einer Gleitkommazahl (Float) auch eine Ganzzahl (Integer), ein boolescher Wert (Bool) oder eine Zeichenkette (String) sein.

3.1.1 Programmierung des Grundgerüsts

Wenn nun die PV-Anlage beziehungsweise die dazugehörige Sensorik durch ein Python Script ersetzt und für ein besseres Verständnis die Kategorien Verbrauch und Produktion eingefügt werden, ergibt sich folgende Grafik:

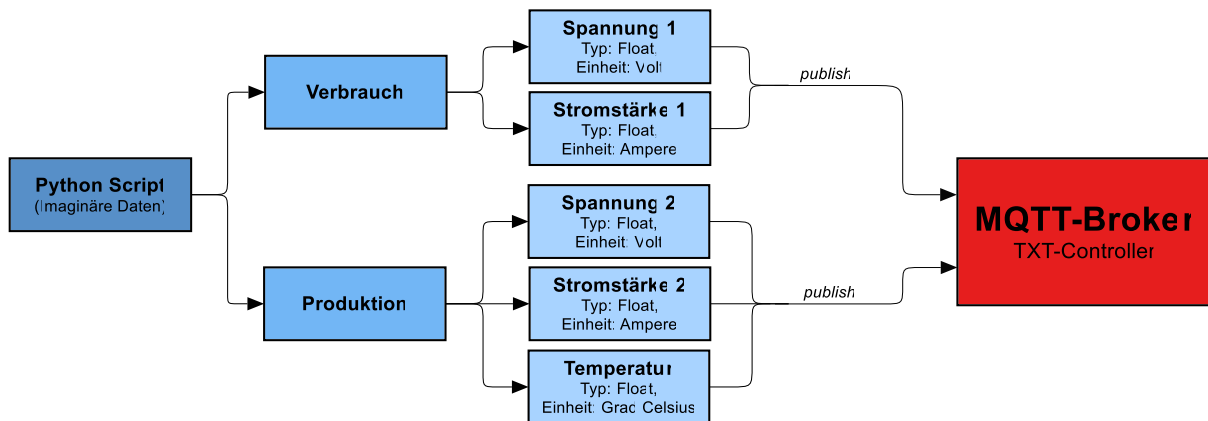


Abbildung 3.3: Blockschaltbild des Grundgerüsts der Datenerfassung

Doch, wie im rechten Teil obiger Grafik ersichtlich, müssen zwei verschiedene Datensätze an den MQTT-Broker veröffentlicht werden. Wenn dies innerhalb eines Programmes geschehen würde, könnte die Datenübermittlung einerseits nicht zeitgleich erfolgen und andererseits könnten beide Programmteile einander stören. Wenn es beispielsweise nach dem Ablauf der Veröffentlichung des Verbrauches zu Verzögerungen oder gar Unterbrechungen beziehungsweise Fehlermeldungen kommen sollte, hat dies unmittelbare Auswirkungen auf die Veröffentlichung der Produktion. Aus genannten Gründen, einer besseren Organisation, einer einfacheren Wiederverwendbarkeit und der im realen Szenario getrennt voneinander erfolgenden Datenerhebung, bietet es sich deshalb an, statt einem Programm zwei zu verwenden. Diese Entscheidung vereinfacht obige Abbildung dann folgendermaßen:

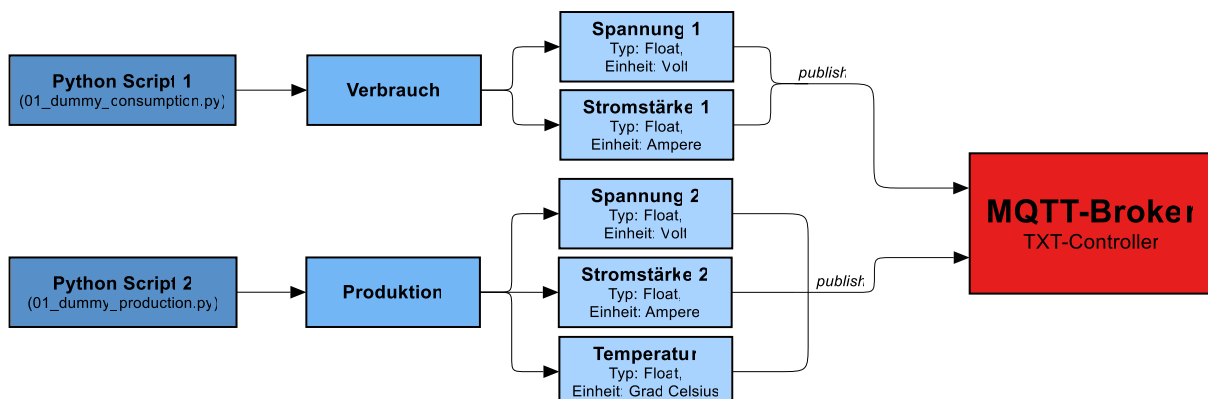


Abbildung 3.4: Optimierte Blockschaltbild des Grundgerüsts der Datenerfassung

Wie in obiger Abbildung ersichtlich werden also zwei separate Python-Programme erstellt, auf welche nur oberflächlich eingegangen wird, da grundlegende Programmierkenntnisse vorausgesetzt werden.

PYTHON SCRIPT 1 (ANHANG A)

Anfangs wird die Adresse, der Port und das Timeout des MQTT-Brokers eingelesen und eine Verbindung aufgebaut. Anschließend werden zufällige Werte im jeweilig vorgegebenen Wertebereich für Spannung, Stromstärke bzw. Batterietemperatur erzeugt, mit einem Zeitstempel versehen und im Topic `pv_modul/energy_consumption` veröffentlicht. Die Veröffentlichung erfolgt dauerhaft mit einem Delay von 1s, um eine Auslastung bzw. einen möglichen Ausfall des MQTT-Brokers zu vermeiden.

PYTHON SCRIPT 2 (ANHANG B)

Analoges gilt für dieses Programm, mit dem Unterschied, dass hier keine Temperatur erzeugt bzw. veröffentlicht wird und die Daten im Topic `pv_modul/energy_production` aufzufinden sind.

Nun kann sich beispielweise mit der Bearbeitung der Datensicherung und der Visualisierung beschäftigt werden, ohne auf einen korrekten Schaltungsaufbau warten zu müssen. Dies war im Rahmen dieser Arbeit ein sehr wichtiger Aspekt, weshalb das Grundgerüst als eigenes Kapitel angeführt wurde. Außerdem kann bei Problemen mit der Sensorik auf das Grundgerüst ausgewichen werden und somit eine weitere problemlose Bearbeitung gewährleistet werden.

3.1.2 Integration des Energiemessgerätes

3.1.2.1 Komponenten

Wie die realen Messdaten ausgelesen werden, bleibt dem Anwender überlassen. Je nach Anwendungsfall hat jedes der folgenden Konzepte seine Vor- aber auch Nachteile:

KONZEPT 1: S0-SCHNITTSTELLE

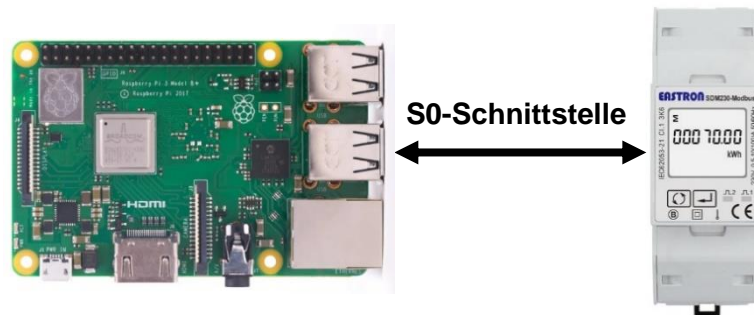


Abbildung 3.5: Datenübertragung des Energieverbrauchs via S0-Schnittstelle [22, 23]

Stückliste Konzept 1 (Stand: 02.03.2022)		
Produkt	Kosten	Verfügbarkeit
1x Energiemessgerät SDM230	41,09€	1 Woche
1x Raspberry Pi (bereits vorhanden)	/	/
	= 41,09€	1 Woche

Tabelle 3.1: Stückliste von Konzept 1 (S0-Schnittstelle)

Bei der S0-Schnittstelle werden die Impulse gemessen, welche pro verbrauchtes Watt erzeugt werden. Die Impulsgröße ist unbestimmt und die Impulszeit beträgt in der Regel meist zwischen 10 und 100 Millisekunden. Die Verbindung zu dem Einplatinencomputer kann hier beispielweise über einen digitalen Interrupt-Pin erfolgen. Zur Verbesserung der Messbarkeit des Signals kann ebenso ein Pull Down-Widerstand eingesetzt werden. Die Versorgung der des Einplatinencomputers könnte direkt über den +5V Anschluss am Energiemessgerät erfolgen und würde somit die Verwendungen eines externen Netzteils vermeiden. Im Programmcode wird dann der Pegel des Impulses überwacht und bei einem Wechsel von 0 auf 1 beziehungsweise von Low auf High der Counter um eins erhöht.

Vorteil	Nachteil
Kostengünstig	Unbestimmte Impulsgröße bzw. -zeit
	Keine zusätzlichen Datensätze
	Erhöhter Arbeitsaufwand

Tabelle 3.2: Vor- und Nachteile der Datenerfassung via S0-Schnittstelle

Achtung: Der SDM230 ist bei diesem Konzept lediglich ein Beispiel und kann durch jedes beliebige Energiemessgerät, welches über eine S0-Schnittstelle verfügt, ersetzt werden.

KONZEPT 2: MODBUS VIA USB



Abbildung 3.6: Datenübertragung des Energieverbrauchs via USB [22, 23, 24]

Stückliste Konzept 2 (Stand: 02.03.2022)		
Produkt	Kosten	Verfügbarkeit
1x Energiemessgerät SDM230	41,09€	1 Woche
1x USB to RS485 Converter	26,40€	1 Woche
1x Raspberry Pi (bereits vorhanden)	/	/
	= 67,49€	1 Woche

Tabelle 3.3: Stückliste von Konzept 2 (ModBus via USB)

Der Datenfluss vom Energiemessgerät zum Einplatinencomputer wird hier durch eine USB-Schnittstelle bzw. dem dazu benötigten Converter gewährleistet. Dieses Konzept bietet sich vor allem dann an, wenn es kein bestehendes Netzwerk gibt beziehungsweise die Kommunikation mit anderen Teilnehmern nicht gewährleistet werden muss. Natürlich könnte der Einplatinencomputer mit einer dementsprechenden Erweiterung per Ethernet-Verbindung in das Netzwerk integriert werden, um den Zugriff auf die Daten für andere Teilnehmer zu ermöglichen, wäre dann aber im Vergleich zum nächsten Konzept wesentlich aufwendiger und unübersichtlicher.

Vorteil	Nachteil
Einfache Nachvollziehbarkeit	Kein Direkter Datenabruf für Netzwerkteilnehmer
Perfekt für Point-to-Point-Verbindung zwischen 2 Geräten	Mittlerer Arbeitsaufwand

Tabelle 3.4: Vor- und Nachteile der Datenerfassung via USB-Schnittstelle

KONZEPT 3: MODBUS VIA ETHERNET

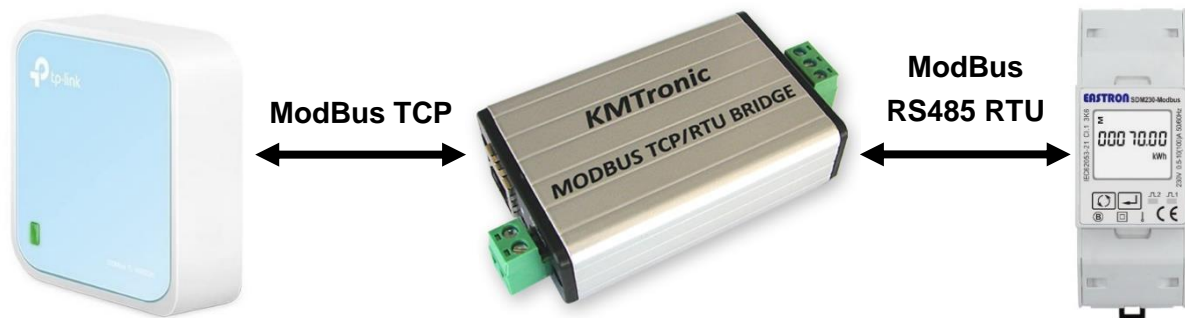


Abbildung 3.7: Datenübertragung des Energieverbrauchs via Ethernet [22, 23, 25]

Stückliste Konzept 3 (Stand: 02.03.2022)		
Produkt	Kosten	Verfügbarkeit
1x Energiemessgerät SDM230	41,09€	1 Woche
1x RS485 Converter to TCP/IP	74,40€	1 Woche
1x TP-Link WR802N Router (bereits vorhanden)	/	/
	= 115,49€	1 Woche

Tabelle 3.5: Stückliste von Konzept 3 (ModBus via Ethernet)

Der Datentransfer vom Energiemessgerät zum Einplatinencomputer wird hier durch eine Ethernet-Schnittstelle bzw. dem dazu benötigten Converter gewährleistet. Damit die Messdaten von jedem Teilnehmer über das interne Netzwerk abgerufen werden können, ohne eine direkte Verbindung zu dem Converter haben zu müssen, ist der Einsatz eines Routers unabdingbar. Grundsätzlich wäre aber die Verwendung eines Einplatinencomputer mit entsprechender Ethernet-Schnittstellen Erweiterung, ähnlich wie bei Konzept 2, ebenso möglich gewesen. Die Einstellungen des Converters, wie Baud Rate, Parity oder Stop Bits erfolgen nicht, wie gewohnt über den Programmcode, sondern über eine seitens des Herstellers vordefinierten Maske. Damit diese Maske aber sichtbar ist, kann die Anpassung des Standardgateways und demnach die Überarbeitung des gesamten Netzwerks notwendig sein.

Vorteil	Nachteil
Geringer Arbeitsaufwand	Überarbeitung des gesamten Netzwerks
Einfache Nachvollziehbarkeit	
Direkter Datenabruf für Netzwerkteilnehmer	

Tabelle 3.6: Vor- und Nachteile der Datenerfassung via Ethernet-Schnittstelle

Um die Wahl des durchzuführenden Konzeptes einerseits neutral und andererseits in Abhängigkeit der Relevanz des jeweiligen Kriteriums zu stellen, wurde eine Nutzwertanalyse durchgeführt. Hier wird zuerst das Kriterium mit dazugehöriger Gewichtung festgelegt und anschließend die einzelnen Konzepte bewertet. Danach werden die Produkte aus Beurteilung beziehungsweise Gewichtung des jeweiligen Kriteriums gebildet, aufaddiert und letztlich verglichen, wobei die Alternative mit den meisten Punkten die bestmögliche Wahl ist.

Nutzwertanalyse							
Kriterium	Gewichtung	Konzept 1		Konzept 2		Konzept 3	
		Bew.	Wert	Bew.	Wert	Bew.	Wert
Kostengünstig	20 %	5	1,0	4	0,8	1	0,2
Aufwandsarm	20 %	1	0,2	3	0,6	4	0,8
Verfügbarkeit	30 %	1	0,3	4	1,2	4	1,2
Datenabrufbarkeit	30 %	2	0,6	3	0,9	5	1,5
SUMME	100%	9	3,1	14	3,5	14	3,7
		3. Platz		2. Platz		1. Platz	

Tabelle 3.7: Nutzwertanalyse für Konzepte zur Integration des Energiemessgerätes

Da die Summen der Bewertungen von Konzept 2 und Konzept 3 mit einem Wert von 14 identisch sind, ist klar ersichtlich, dass die Gewichtung in dieser Nutzwertanalyse den Unterschied ausgemacht hat. Nichtsdestotrotz wird Konzept 3 umgesetzt, weil es durch die Gewichtung besser zu den individuellen Anforderungen dieser Arbeit passt.

3.1.2.2 Schaltungsaufbau

Um ein Verständnis für den Schaltungsaufbau des Energiemessgerätes entwickeln zu können, ist folgende Grafik essenziell:

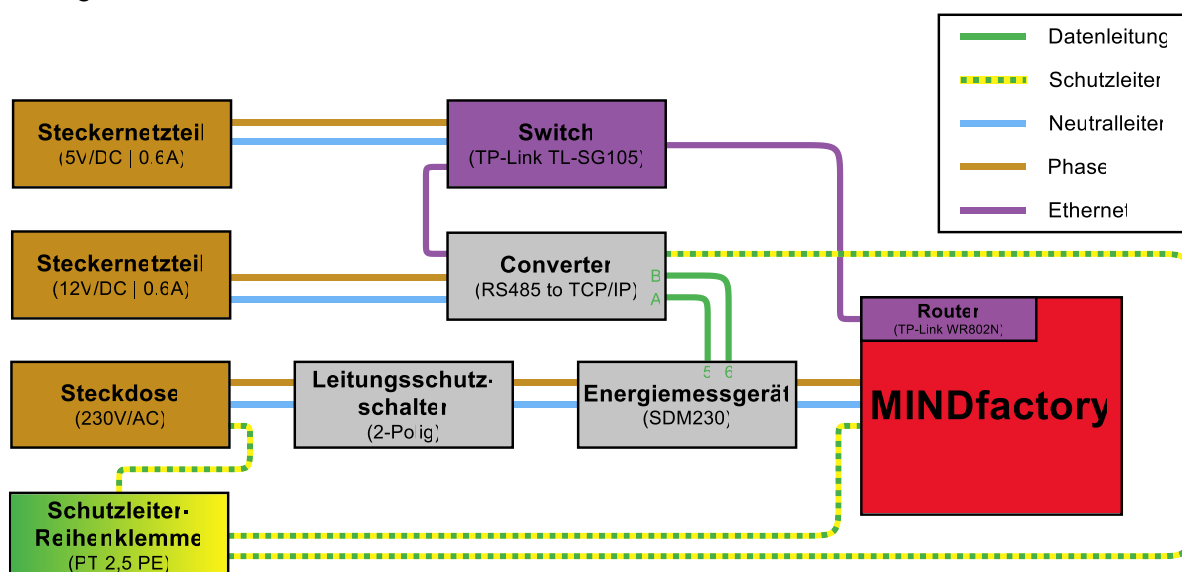


Abbildung 3.8: Schaltungsaufbau des Energiemessgerätes

Bisher unbekannte Komponenten, wie Steckernetzteile, Leistungsschutzschalter, Switches oder Schutzleiterreihenklemmen wurden in den Stücklisten der Konzepte nicht angeführt, da sie einerseits keinen Mehrwert zur Nutzwertanalyse beitragen und andererseits den Fokus von den grundlegenden Bauteilen ziehen. Darüber hinaus waren beschriebene Komponenten entweder im Lieferumfang enthalten oder bereits vorhanden.

3.1.2.3 Programmierung

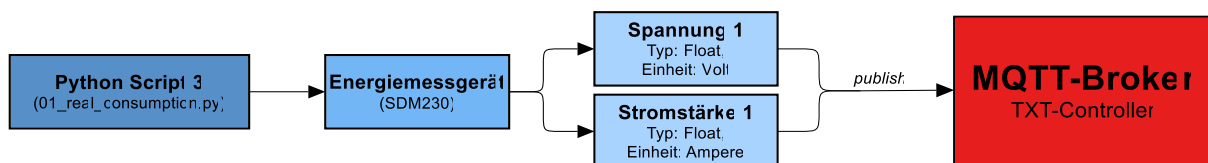


Abbildung 3.9: Blockschaltbild von Datenerfassung des Energiesmessgerätes

PYTHON SCRIPT 3 (ANHANG C)

Anfangs wird die Adresse, der Port und das Timeout des MQTT-Brokers eingelesen und eine Verbindung aufgebaut. Anschließend wird eine Verbindung mit dem SDM230 Energiesmessgerät via TCP erstellt. Wenn die Verbindung erfolgreich aufgebaut werden konnte, werden die Messwerte für Spannung und Stromstärke ausgelesen, mit einem Zeitstempel versehen und im Topic `pv_modul/energy_consumption` veröffentlicht. Die Veröffentlichung erfolgt dauerhaft mit einem Delay von 0.4s, um einerseits eine Auslastung bzw. einen möglichen Ausfall des MQTT-Brokers zu vermeiden und andererseits großen Datenlücken entgegenzuwirken.

3.1.3 Integration des Photovoltaikmoduls

3.1.3.1 Komponenten

Im Gegensatz zur Integration des Energiemessgerätes ist eine Erstellung mehrerer Konzepte und eine damit verbundene Nutzwertanalyse nicht notwendig, da die Produktvielfalt eingeschränkt ist und sich die Konzepte einander maßgeblich überschneiden würden.

PHOTOVOLTAIK-MODUL

Wie bereits beschrieben, ist das Photovoltaik-Modul für die Umwandlung der Strahlungsenergie in elektrische Energie verantwortlich. Da es sich bei der MINDfactory um einen Demonstrator handelt, empfiehlt es sich einerseits ein überschaubar großes und andererseits ein kostengünstiges Modul auszuwählen. Deshalb wurde ein monofaziales monokristallines Modul mit einer maximalen Nennleistung von 50W, einer Abmessung von 700 x 435 x 35mm und einem Gewicht von 3,96kg der Firma Enjoy Solar verwendet, welches für den Betrieb eines 12V-Systems ausgelegt ist.

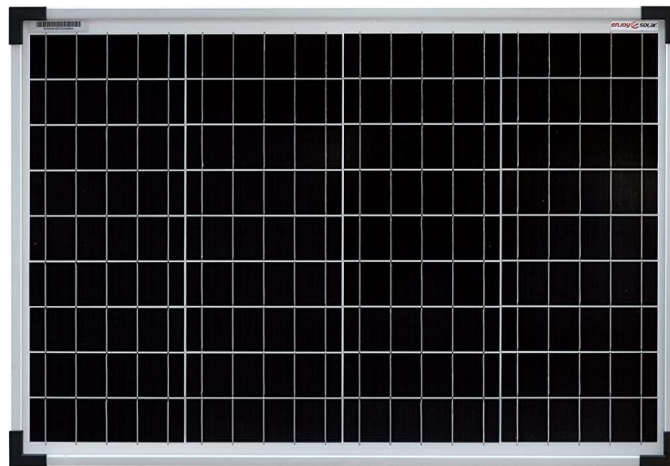


Abbildung 3.10: Verwendetes PV-Modul [39]

Für weitere technische Daten und I-V Kennlinien bei verschiedenen Temperaturen und Bestrahlungen siehe [39].

RASPBERRY PI

Das Herzstück der Datenerfassung von Spannung und Stromstärke des PV-Moduls ist der Einplatinencomputer Raspberry Pi. In dieser Arbeit wurde der Raspberry Pi 4 Modell B verwendet, kann aber prinzipiell auch durch ein älteres Modell ersetzt werden.

Alternative Computer, wie Arduino o.Ä. wurden nicht in Betracht gezogen, da der Raspberry Pi leistungsfähiger, vielseitiger, flexibler und praxisnaher ist. Darüber hinaus liegt der Fokus dieser Arbeit auf der Entwicklung von Python-Applikationen und da das bestehende System der MINDfactory auch in Python programmiert wurde, ist die Umsetzung letztendlich am besten mit einem Raspberry Pi zu realisieren.

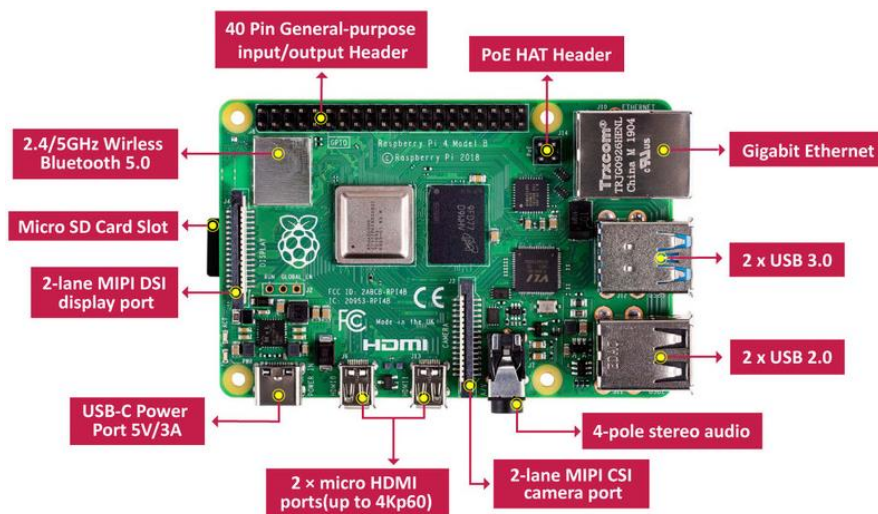
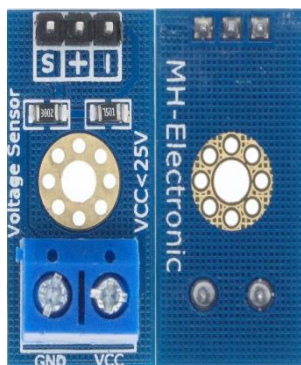


Abbildung 3.11: Raspberry Pi 4 Model B [40]

Im Rahmen dieser Arbeit wurde der USB-C-Anschluss zur Spannungsversorgung, der HDMI-Anschluss zur Anzeige der Nutzeroberfläche, der Ethernet-Anschluss zur Verbindung mit dem vorhandenen Netzwerk, die GPIO-Pins zur Evaluierung bzw. Versorgung der Sensorik und der Micro SD Card Slot zur Nutzung eines primären Speichers für das Sichern von Betriebssystem, Daten, Programmen. Die Micro SD-Karte wurde mithilfe eines USB Card Adapter mit dem Rechner verbunden, mit dem SD Card Formatter formatiert und letztlich mit dem Raspberry Pi Imager mit dem Betriebssystem Raspberry Pi OS (32-Bit) beschrieben. Für weitere technische Daten und Spezifikationen siehe [41].

SPANNUNGSSENSOR

Der Spannungssensor basiert auf einem resistiven Spannungsteiler, welcher aus zwei in Serie geschalteten Widerständen besteht. Die Widerstände $R_1 = 30\text{k}\Omega$ (links) und $R_2 = 7.5\text{k}\Omega$ (rechts) sind so ausgelegt, dass die max. Spannung dem fünffachen der Betriebsspannung entspricht. Die gemessene Spannung wird als analoges Signal am S-Pin in der Ausgangsschnittstelle ausgegeben, durch einen ADC in ein digitales Signal umgewandelt und von einem Einplatinencomputer ausgelesen.



Betriebsspannung	3,3/5V DC
Messbereich	0-16,5V/25V DC
Eingangsschnittstelle	VCC = Phase GND = Neutraleiter
Ausgangsschnittstelle	+ = 5V DC - = GND S = Analoges Ausgangssignal (ADC)
Größe	2.7 x 1.4 x 1.4cm

Abbildung 3.12: Spannungssensor 0-25V DC [42]

STROMSENSOR ACS712

Der ACS712-30A ist ein Stromsensor, der auf dem Hall-Effekt-Prinzip basiert. Er misst die magnetische Feldstärke des durchfließenden Stroms und erzeugt anhand dessen eine proportionale Hall-Spannung. Dieser Messwert wird als analoges Signal am OUT-Pin in der Ausgangsschnittstelle ausgegeben, durch einen ADC in ein digitales Signal umgewandelt und von einem Einplatinencomputer weiterverarbeitet. Doch um von der Hall-Spannung sinnvolle Rückschlüsse auf die eigentliche Stromstärke ziehen zu können, muss die Nullspannung abgezogen und anschließend durch die Sensorempfindlichkeit dividiert werden. Das Endergebnis entspricht dann der gemessenen Stromstärke.

$$I = \frac{U - \frac{V_{CC}}{2}}{\text{Sensitivity}} = \frac{U - 2.5V}{0.066 \frac{V}{A}} \quad (1)$$

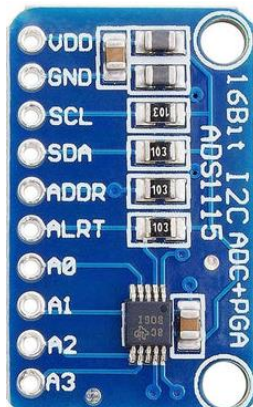


Betriebsspannung	5V DC
Messbereich	± 30 A
Empfindlichkeit	66mV/A
Eingangsschnittstelle	Pin_1, Pin_2 = Phase, Phase
Ausgangsschnittstelle	VCC = 5V DC OUT = Analoges Ausgangssignal (ADC) GND = GND
Größe	3.2 x 1.4 x 1.3cm

Abbildung 3.13: Stromsensor ACS712-30A [43]

ANALOG-DIGITAL-CONVERTER ADS1115

Der ADS1115 ist ein Analog-Digital-Converter mit einer Auflösung von 16-Bit und der Anschlussmöglichkeit von 4 Kanälen. Er nutzt eine I2C-Schnittstelle, um mit dem Einplatinencomputer interagieren zu können. Bei der Nutzung von mehreren ADC-Modulen am gleichen I2C-Bus muss die I2C-Adresse am ADDR-Pin festgelegt und daher entweder mit GND (0x48), VDD (0x49), SDA (0x4A) oder SDL (0x4B) verbunden werden. Bei unbelegtem Anschluss ist die Adresse 0x48. Der ALRT-Pin kann in Form einer booleschen Variable genutzt werden, um zu sehen, ob ein Vergleichswert über- oder unterschritten wird.



Betriebsspannung	3.3V DC
Eingangsschnittstelle	A0, A1, A2, A3 = Analogeingänge
Ausgangsschnittstelle	VDD = 3.3V DC GND = GND SCL = Serial Clock (I2C) SDA = Serial Data (I2C) ADDR = I2C-Adressauswahl ALRT = Alarm
Größe	2.7 x 1.7 x 0.3cm

Abbildung 3.14: Analog-Digital-Converter ADS1115 [44]

TEMPERATURSENSOR DHT22

Der DHT22, auch als AM2302 bekannt, ist ein Temperatur- bzw. Feuchtigkeitssensor, der Temperaturen im Bereich von -40 °C bis +80 °C und relative Luftfeuchtwerte von 0% bis 100% messen kann. Die Messwerte werden am OUT-Pin in Form eines digitalen Signals ausgegeben und erfordert daher keinen zusätzlichen Analog-Digital Converter. Der Abtastzeitraum beträgt durchschnittlich zwei Sekunden, das heißt, dass zwischen aufeinanderfolgenden Messwertänderungen etwa zwei Sekunden liegen.



Betriebsspannung	3,3-6V DC
Messbereich	Temperatur: -40 bis 80°C Luftfeuchtigkeit: 0-100% RH
Ausgangsschnittstelle	+ = 3,3V DC OUT = Digitales Ausgangssignal - = GND
Genauigkeit	Temperatur: $\pm 0,5^{\circ}\text{C}$ Relative Luftfeuchtigkeit: $\pm 2\text{-}5\%$
Abtastzeitraum	~ 2s
Größe	2.7 x 1.7 x 0.3cm

Abbildung 3.15: Temperatursensor DHT22 [45]

3.1.3.2 Schaltungsaufbau

Um ein Verständnis für den Schaltungsaufbau des Photovoltaikmoduls entwickeln zu können, ist folgende Grafik essenziell:

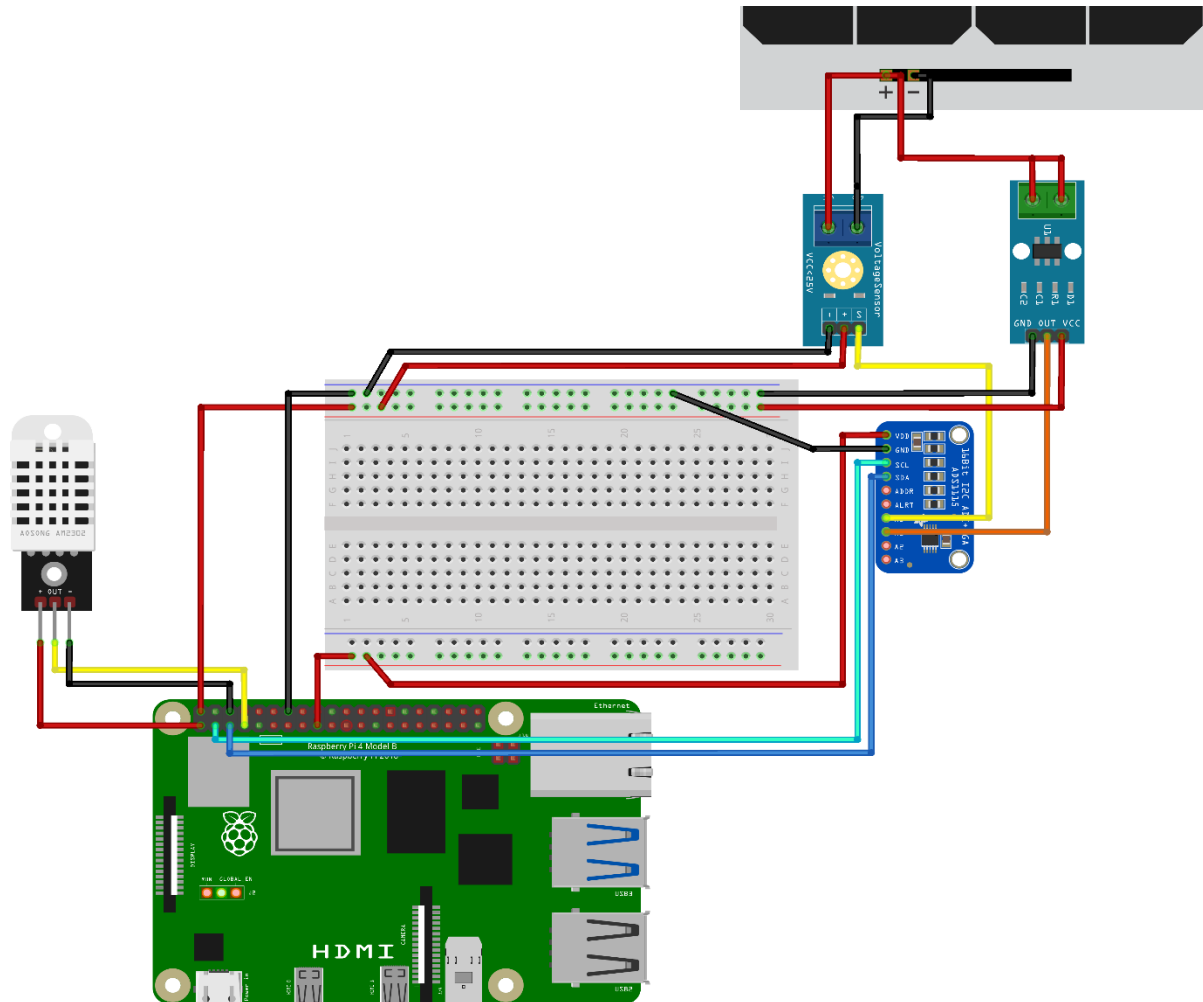


Abbildung 3.16: Schaltungsaufbau des Photovoltaikmoduls mit Sensorik

Pin_1 (3.3V DC)	+ (DHT22)
Pin_2 (5V DC)	+ (Spannungssensor) VCC (Stromsensor)
Pin_3 (SCL)	SCL (ADC)
Pin_5 (SDA)	SDA (ADC)
Pin_6 (GND)	- (DHT22)
Pin_7 (GPIO4)	OUT (DHT22)
Pin_14 (GND)	- (Spannungssensor) GND (Stromsensor) GND (ADC)
Pin_17 (3.3V DC)	VDD (ADC)

Tabelle 3.8: Anschlussabelle für Schaltungsaufbau des Photovoltaikmoduls mit Sensorik

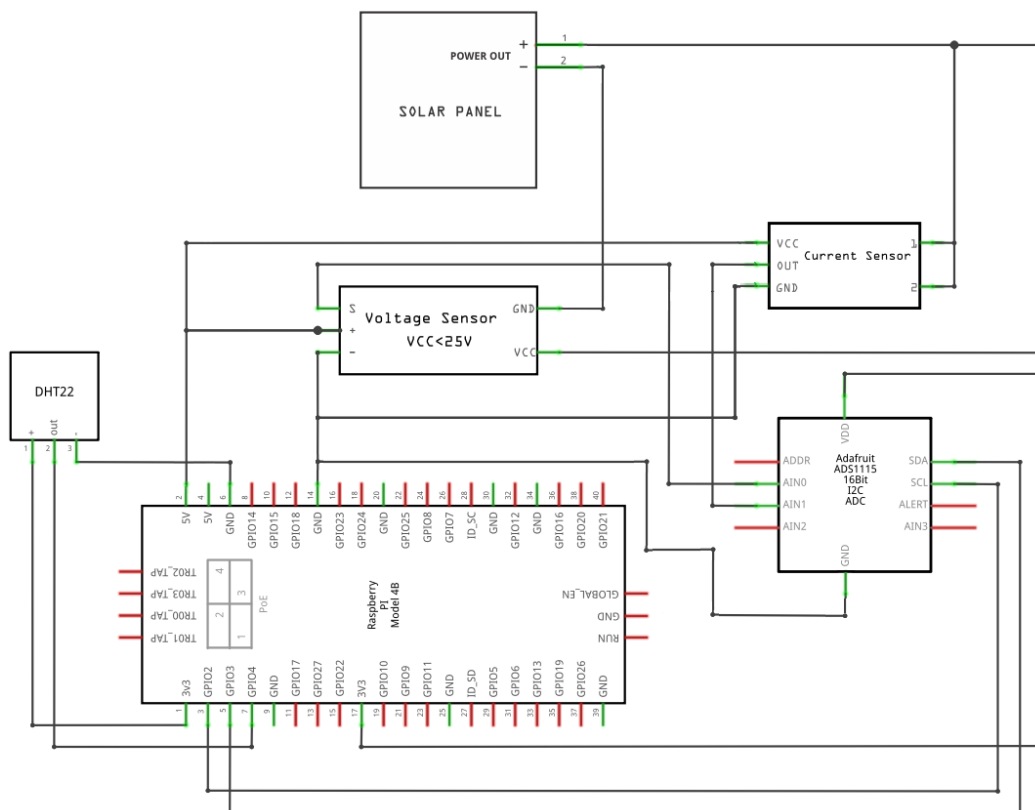


Abbildung 3.17: Optionaler Schaltplan für Schaltungsaufbau des Photovoltaikmoduls mit Sensorik

Da die einzelnen Komponenten bereits erläutert wurden, wird nicht genauer auf den Schaltungsaufbau eingegangen.

3.1.3.3 Programmierung

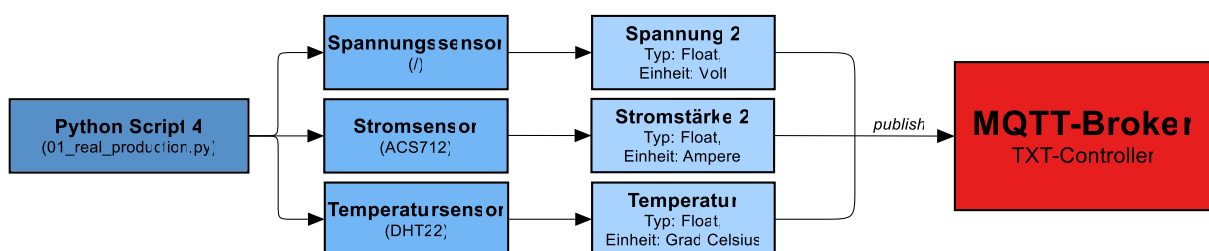


Abbildung 3.18: Blockschaltbild von Datenerfassung des Photovoltaikmoduls

PYTHON SCRIPT 4 (ANHANG D)

Anfangs wird die Adresse, der Port und das Timeout des MQTT-Brokers eingelesen und eine Verbindung aufgebaut. Anschließend wird die Spannung anhand des Spannungssensor und die Stromstärke mithilfe des ACS712-30A Stromsensors ausgelesen, mit einem Zeitstempel versehen und im Topic `pv_modul/energy_production` veröffentlicht. Die Veröffentlichung erfolgt dauerhaft mit einem Delay von 0.4s, um einerseits eine Auslastung bzw. einen möglichen Ausfall des MQTT-Brokers zu vermeiden und andererseits großen Datenlücken entgegenzuwirken.

3.2 Datensicherung

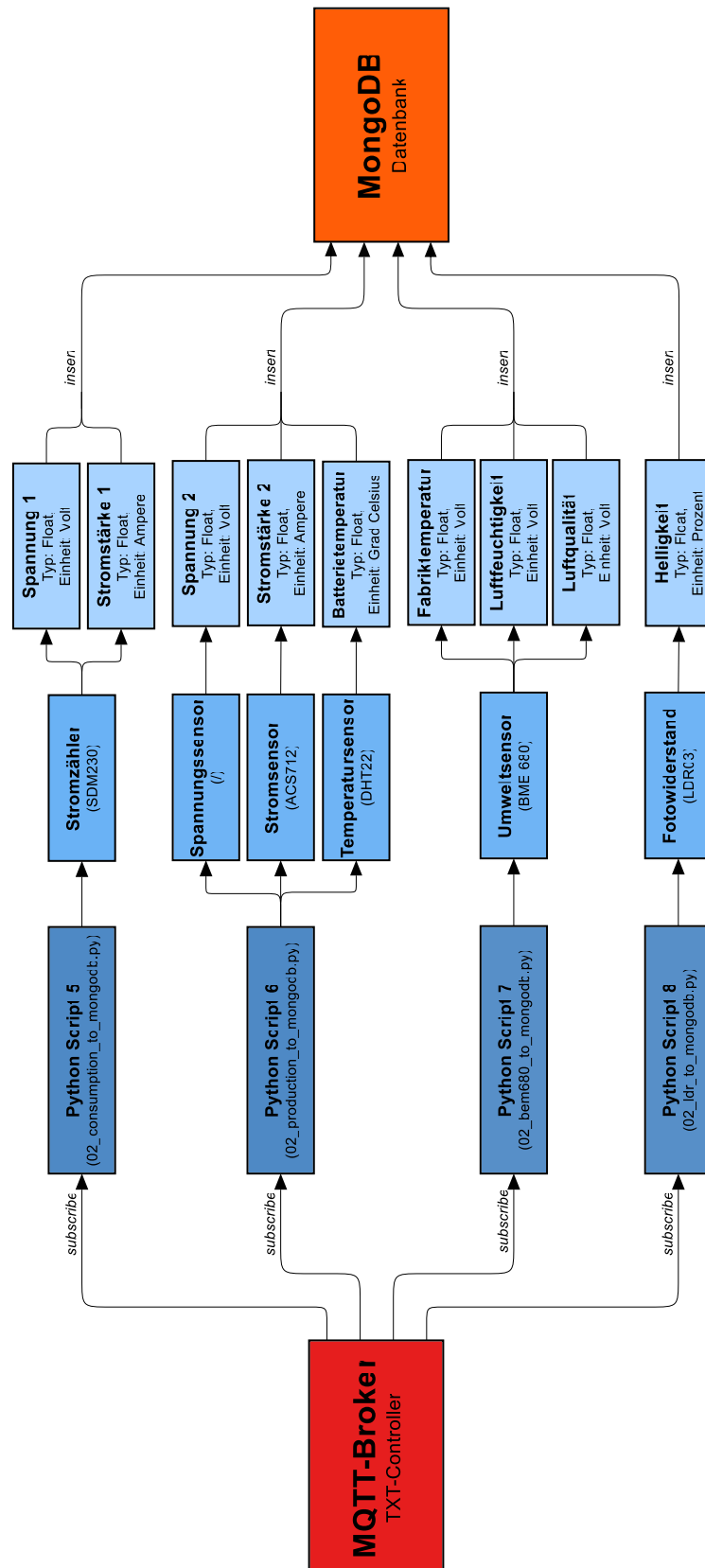


Abbildung 3.19: Blockschaftbild der Datensicherung

PYTHON SCRIPT 5 (ANHANG E)

Anfangs wird die Adresse, der Port und das Timeout des MQTT-Brokers eingelesen und eine Verbindung aufgebaut. Anschließend wird im Topic `pv_modul/energy_consumption` auf Nachrichten des Energiemessgerätes gewartet und diese dann in MongoDB in der Collection `pv_modul/energy_consumption` gesichert.

PYTHON SCRIPT 6 (ANHANG F)

Analoges gilt hier, mit dem Unterschied, dass dieses Programm im Topic `pv_modul/energy_production` auf Nachrichten des Spannungs-, Strom- bzw. Temperatursensors wartet und in MongoDB in der Collection `pv_modul/energy_production` sichert.

PYTHON SCRIPT 7 (ANHANG G)

Analoges gilt hier, mit dem Unterschied, dass dieses Programm im Topic `i/bme680` auf Nachrichten des Umweltsensors wartet und in MongoDB in der Collection `pv_modul/bme60` sichert. Informationen bezüglich Umweltsensor BME680 sind dem Datenblatt [46] zu entnehmen.

PYTHON SCRIPT 8 (ANHANG H)

Analoges gilt hier, mit dem Unterschied, dass dieses Programm im Topic `i/ltr` auf Nachrichten des Umweltsensors wartet und in MongoDB in der Collection `pv_modul/ltr` sichert. Informationen bezüglich Fotowiderstand LDR03 sind dem Datenblatt [47] zu entnehmen.

3.3 Visualisierung

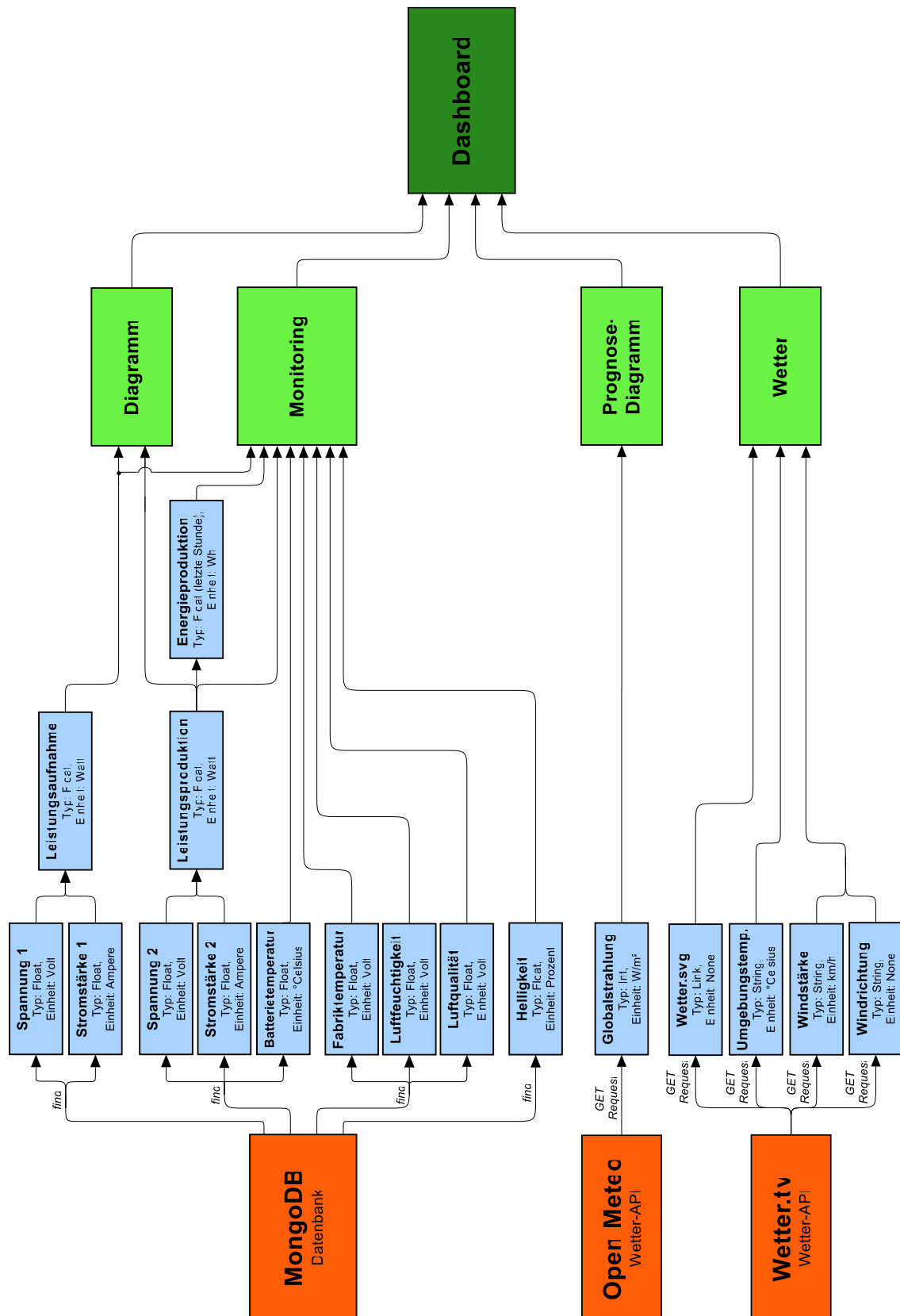


Abbildung 3.20: Blockschaltbild der Visualisierung

Zur besseren Veranschaulichung wurden die jeweiligen Programmcodes nicht in vorheriger Grafik eingebunden, sondern werden lediglich kurz erläutert. Um den Rahmen dieser Arbeit nicht zu sprengen und unnötig in die Tiefe zu gehen werden Grundkenntnisse im Bereich der Programmierung mit Django vorausgesetzt. Darüber hinaus ist es nicht Ziel dieser Arbeit bestehende Komponenten der MINDfactory zu verbessern, weshalb auf bereits bestehende Einstellungen nicht eingegangen wird. Bei Auffrischungsbedarf können die Quellen [17, 18] zur Hilfe gezogen werden.

VIEWS.PY (ANHANG I)

Diese Python-Applikation ist das Herzstück des Dashboards, da hier die Messwerte und alle externen Daten verarbeitet und endgültig definiert werden. Prinzipiell wird eine Django TemplateView Klasse mit dem Namen PVBaseView definiert, die später die HTML-Vorlage `index_pv_modul.html` rendert. Bevor dies aber geschieht, werden einerseits die aktuellen Messwerte aus der MongoDB-Datenbank abgerufen und andererseits die Wetterdaten der nächsten 4 Stunden von Wetter.tv bzw. die Globalstrahlung der nächsten Woche von der Open-Meteo-API bezogen. Anschließend kommt es zu einer grafischen Darstellung der Leistungsproduktion und der Leistungsaufnahme mit jeweiliger Glättung mit Plotly. Die Wochenprognose der theoretischen Maximalleistung basiert auf der Glättung der Globalstrahlung anhand der sieben umgebenden Datenpunkte und wurde ebenso mit Plotly realisiert. Der Wirkungsgrad wurde mit 18% angenommen und die durch die PV-Module abgedeckte Fläche wurde den Herstellerangaben entnommen. Die Glättung wurde in beiden Fällen mit dem Savitzky-Golay-Filter bzw. einer Polynomfunktion 2. Grades erstellt. Die Fensterlänge der Leistungsproduktion bzw. der Leistungsaufnahme beträgt 61 und der Wochenprognose 7.

INDEX_PV_MODUL.HTML (ANHANG J)

Das HTML-File ist die Vorlage, welche die Webapplikation, also in diesem Fall das Dashboard, hat. Damit die Base, welche unter anderem Header, Footer und Logo festlegt, für alle Endpoints des Dashboards demselben Schema und Format folgen, wurde diese aus `base.html` importiert. Darüber hinaus werden relevante Daten mithilfe von `get_context_data()` aus `views.py` eingebettet.

UPDATE_CHART.JS (ANHANG K)

Damit der Einfluss der durch die PV-Module abgedeckten Fläche auf die theoretische Maximalleistung interaktiv gestaltet werden kann, ist diese JavaScript-Datei notwendig. Grob vereinfacht aktualisiert sie den HTML-Code, je nachdem welchen Wert der Nutzer für die Modulfläche angibt. Dadurch kann ein echtzeitbasierter Vergleich mit der derzeit verbauten Modulgröße erfolgen.

Obwohl die HTML- und JavaScript-Dateien keine Python-Applikationen sind, sind sie für das Dashboard unabdingbar und wurden deshalb aus Gründen der besseren Nachvollziehbarkeit ebenso erwähnt.

4 Ergebnis und Diskussion

4.1 Ergebnis



Abbildung 4.1: Dashboard

Für eine hochauflösende Darstellung folgende PDF- oder HTML-Datei öffnen:



MINDfactoree_Dashboard.pdf



MINDfactoree_Dashboard.zip

Mit einem Klick auf die Luftqualität öffnet sich folgende Wertetabelle:

Table 4: Index for Air Quality (IAQ) classification and color-coding⁹

IAQ Index	Air Quality	Impact (long-term exposure)	Suggested action
0 – 50	Excellent	Pure air; best for well-being	No measures needed
51 – 100	Good	No irritation or impact on well-being	No measures needed
101 – 150	Lightly polluted	Reduction of well-being possible	Ventilation suggested
151 – 200	Moderately polluted	More significant irritation possible	Increase ventilation with clean air
201 – 250 ⁹	Heavily polluted	Exposition might lead to effects like headache depending on type of VOCs	optimize ventilation
251 – 350	Severely polluted	More severe health issue possible if harmful VOC present	Contamination should be identified if level is reached even w/o presence of people; maximize ventilation & reduce attendance
> 351	Extremely polluted	Headaches, additional neurotoxic effects possible	Contamination needs to be identified; avoid presence in room and maximize ventilation

Close

Abbildung 4.2: Air-Quality-Index (IAQ = Indoor Air Quality)

4.2 Diskussion

DATENERFASSUNG

Der von fischertechnik ab Werk verbaute TXT-Controller, welcher als MQTT-Broker fungiert, verursacht durch einen Systemabsturz oftmals Kommunikationsprobleme. Vermutlich ist er für den zusätzlichen Datenverkehr nicht ausgelegt, weshalb er nach unbestimmter Zeit nicht mehr erreichbar ist, und eine Datenerhebung verhindert. Deshalb ist die Entwicklung eines MQTT-Brokers, beispielsweise in Form eines Raspberry Pis, empfehlenswert. Grundlegend hierfür ist aber eine ordnungsgemäße Validierung der Ursache des Systemabsturzes. Als vorrübergehende Lösung könnte eine Python-Applikation entwickelt werden, welche die Erreichbarkeit des MQTT-Brokers anhand einer roten bzw. grünen LED repräsentiert und damit die Fehlersuche wesentlich vereinfacht.

Bei der Abfrage der Batterietemperatur kann es passieren, dass der DHT22 Temperatursensor nicht antwortet. Da die Erhebung der Spannung und der Stromstärke des PV-Moduls mit der Batterietemperatur gekoppelt ist, kann es zu Datenlücken kommen. Hier könnte einerseits ein anderer Temperatursensor verwendet werden oder andererseits eine getrennte Datenerhebung erfolgen. Sollte sich für eine getrennte Datenerhebung entschieden werden, ist die Verwendung von Docker empfehlenswert.

Derzeit aktualisieren sich die Messwerte nur bei erneutem manuellem Aufruf der Website. Dies ist für diese Arbeit völlig ausreichend, aber könnte mithilfe von Django Channels per Websocket in Echtzeit basieren. Zu beachten ist jedoch, dass die Graphen der Performance und der Prognose davon ausgenommen werden sollten, damit eine sinnvolle Interaktion gewährleistet werden kann. Ansonsten würden die Graphen nämlich ständig aktualisiert werden und könnten direkt durch ein statisches Bild ersetzt werden.

DATENSICHERUNG

Die Datensicherung erfolgt derzeit in der No-SQL Datenbank MongoDB und funktioniert einwandfrei. In der industriellen Anwendung könnte der Wechsel auf eine Zeitreihendatenbank, wie InfluxDB, von Vorteil sein. Ein Vergleich der beiden Datenbanken zeigte, dass InfluxDB bei der Datenaufnahme um das 1.9-fache schneller, bei der Datenabfrage um das 5-fache schneller und bei der Komprimierung um das 7.3-fache besser war. [48]

Dem ist hinzuzufügen, dass nicht alle Datenpunkte im Graph der Performance dargestellt werden sollten, um eine schnellere Abrufbarkeit des Dashboards zu ermöglichen. Ebenso wäre es empfehlenswert veraltete Datenpunkte automatisch aus der Datenbank zu löschen, um Ressourcen einzusparen und Ladezeiten zu minimieren.

VISUALISIERUNG

Über die Visualisierung kann gesagt werden, dass das Dashboard tendenziell eher modern, aber dennoch sehr nutzerfreundlich gestaltet wurde. Obwohl die Verwendung von Emojis im industriellen Kontext in Vergangenheit häufig verpönt war, sind einige Unternehmen bereits auf den Zug der Modernisierung aufgesprungen. Ebenso wurde es auch in dieser Arbeit handgehabt, weshalb durch den gezielten Einsatz von Emojis die jeweiligen Messwerte hervorgehoben wurden. Dies fördert nicht nur das Engagement des Nutzers, sondern vermittelt die Messwerte auch auf ansprechende Art und Weise. Ergänzend dazu, ist die Angabe der Uhrzeit des letzten Updates hilfreich, um zu erkennen, ob die angezeigten Daten aktuell sind. Die Uhrzeit wird aber unabhängig von der Aktualität der Messwerte geändert. Hier wäre es sinnvoll entweder den Timer an die Messdaten zu koppeln oder für jeden Messwert einen eigenen Timer zu verlinken.

Die Einbindung des Wetters der nächsten 4 Stunden ergänzt die entnommenen Messwerte, um den allgemeinen Wetterzustand, die Umgebungstemperatur, die Windstärke und die Windrichtung. Dadurch wird eine schnelle bzw. oberflächliche Wahrnehmung der Wettervorhersage ermöglicht.

Sowohl die Performance, also die Gegenüberstellung von Leistungsproduktion der PV-Anlage und Leistungsaufnahme der Fabrik, aber auch die Wochenprognose der theoretischen Maximalleistung sind gut erkenntlich und interaktiv gestaltet. Folglich gibt es die Möglichkeit des Skalierens und des Ausblendens der Datenreihen, weswegen eine Analyse der Messwerte maßgeblich vereinfacht wird. Die vordefinierten Buttons helfen dabei schnell und akkurat zwischen Zeitfenstern wechseln zu können. Auch die Implementierung der Glättungen ermöglicht eine Reduzierung von Schwankungen bzw. Rauschen und gestaltet damit die Identifikation eines Trends wesentlich einfacher. Die ordnungsgemäße Achsenbeschriftung, Diagrammbeschreibung und Quellenangabe extern bezogener Daten erlauben eine korrekte Nachvollziehbarkeit und Interpretation der Graphen.

Der Fokus dieser Arbeit lag zwar auf der Umsetzung des Dashboards und den damit verbundenen Python-Applikationen, aber im nächsten Kapitel werden dennoch verschiedene weitere Vorgehensweisen erläutert. Diese beinhalten sowohl informatische, mechanische und elektrotechnische Ansätze.

4.3 Weitere Vorgehensweise

AKKURATERE ERTRAGSPROGNOSE DURCH ENTWICKLUNG KÜNSTLICHER INTELLIGENZ

Es kann gesagt werden, dass die in dieser Arbeit prognostizierten Leistungs- bzw. daraus ableitbaren Energieerträge anhand der Globalstrahlung von Open-Meteo, der durch die PV-Module abgedeckten Fläche und des Wirkungsgrades eher experimentell und daher sehr vage sind. Dies reicht aber im Rahmen dieser Arbeit völlig aus und dient lediglich zur Demonstration der Anwendung von Daten externer Schnittstellen. Der Grundgedanke ist aber, wie es bereits das PVGIS (Photovoltaic Geographical Information System) der europäischen Kommission macht, wesentlich mehr Faktoren bzw. Methoden in die Prognose einzubinden. Um diesen wissenschaftlich fundierten Methodiken dann den letzten Feinschliff zu geben, könnten diese in eine künstliche Intelligenz einspeist werden. Dies ist vor allem dann vorteilhaft, wenn ein langfristiger Betrieb bei gleichbleibendem Standort angestrebt wird, da die individuelle Prognose mit steigenden Datensätzen immer akkurater wird.

Zusammengefasst kann also gesagt werden, dass der Einsatz einer künstlichen Intelligenz die Leistungs- bzw. Energieertragsprognose aussagekräftiger machen und damit beispielweise eine

- Abhängigkeitsanalyse der (Energie-)Lieferanten
- Vermeidung von Überproduktion bzw. Kostenersparnis durch gezielten Einsatz von Solarstrom für energieintensive Prozesse (z.B. Brennofen)
- Wartung- und Reinigungsplanung
- Abweichung von untypischen Energieertragsverläufen bzw. Ertragseinbrüchen

wesentlich effizienter gestalten würde.

Für weitere Informationen bezüglich PGIS, siehe:

https://joint-research-centre.ec.europa.eu/pvgis-online-tool/getting-started-pvgis/pvgis-user-manual_en

AUTOMATISCHE AUSRICHTUNG DES PV-MODULS

Interessant wäre es zudem ein Gestell zu entwickeln, welches eine Ertragsoptimierung durch automatische Ausrichtung des PV-Moduls nach Einstrahlungsrichtung ermöglicht. Womöglich könnte auch direkt ein fertiges 2-Achsen-System zugekauft werden und mit Sensorik erweitert werden. Mit Hilfe mehrerer Helligkeits- bzw. GPS-Sensoren und der Prognose des Sonnenstandes könnte der Azimut- und Neigungswinkel automatisch angepasst werden. Derzeit gibt es zwar, ähnlich wie bei der Ertragsprognose, Empfehlungen bzw. Richtwerte auf Monatsbasis aber nicht auf Tages- oder Stundenbasis. Für eine vernünftige Umsetzung wäre es deshalb essenziell eine fundierte Prognose bereits parat zu haben.

ERWEITERUNG DES SYSTEMS MIT MEHREREN PV-MODULEN

Ebenso könnte es empfehlenswert sein mehrere PV-Module zu nutzen, um den Leistungs- und infolge den Energieertrag zu erhöhen. Hierbei wäre es vor allem interessant jedes Modul mit einem Spannungs- und Stromsensor auszustatten, um den Ertrag der einzelnen Module beobachten und vergleichen zu können. Diese könnten dann in tabellarischer Ansicht im Dashboard eingesehen werden.

ERSATZ VON SPANNUNGS-, STROMSENSOR UND ADC DURCH INA219-SENSOR

Wenn eine Erweiterung des Systems erfolgen soll, ist eine Optimierung der Bauteile, insbesondere der Spannungs- und Stromsensoren relevant. Statt der simultanen Verwendung von Spannungs-, Stromsensor und ADC könnte auf einen INA219-Sensor zurückgegriffen werden. Dieser kann sowohl Spannung als auch Stromstärke messen und direkt per integrierter I2C-Schnittstelle ausgeben, wodurch der Schaltungsaufbau wesentlich vereinfacht werden kann. Dennoch gilt zu beachten, dass der I2C-Bus des Raspberry Pis nur 4 Adressen unterstützt. Falls es also zu einem Anwendungsfall von mehr als 4 PV-Modulen kommen sollte, ist der derzeitige Aufbau mit separaten ADCs passender, da er wesentlich mehr Analogeingänge ermöglicht.

Literaturverzeichnis

- [1] Fischertechnik, *Industrie 4.0-Anwendungen und Prozessabläufe als modulares Trainings- und Simulationsmodell*. [Online] Verfügbar unter: <https://www.fischertechnik.de/de-de/produkte/simulieren/simulationsmodelle/560840-lernfabrik-4-0-24v-inkl-s7-1500> (Zugriff am: 14. Februar 2023)
- [2] Fischertechnik, *Industrie 4.0-Anwendungen und Prozessabläufe als modulares Trainings- und Simulationsmodell*. [Online] Verfügbar unter: https://content.ugfischer.com/cbfiles/fischer/Zulassungen/ft/Blockschaltbild_554868_Lernfabrik_4_0_24V.pdf (Zugriff am: 14. Februar 2023)
- [3] Fischertechnik, *Industrie 4.0-Anwendungen und Prozessabläufe als modulares Trainings- und Simulationsmodell*. [Online] Verfügbar unter: https://content.ugfischer.com/cbfiles/fischer/Zulassungen/ft/Industriekatalog_2021_deutsch-Poster-Lernfabrik-german-poster-learning-factory.pdf (Zugriff am: 14. Februar 2023)
- [4] Wolfgang Dummer, *Entwicklung eines digitalen Zwillings für einen Fabrikdemonstrator in Miniaturgröße*. (Zugriff am: 14. Februar 2023)
- [5] J. Ratliff, „Home - Docker“, *Docker*, 10. Mai 2022, 2022. [Online]. Verfügbar unter: <https://www.docker.com/>. (Zugriff am: 15. Februar 2023)
- [6] Docker, *Docker: lightweight linux containers for consistent development and deployment*, 2014. [Online]. Verfügbar unter: <https://www.seltzer.com/margo/teaching/cs508.19/papers/merkel14.pdf> (Zugriff am: 15. Februar 2023)
- [7] PCFoundation, *OPC UA Reference*. [Online]. Verfügbar unter: <https://reference.opcfoundation.org/> (Zugriff am: 14. Februar 2023)
- [8] GitHub, *FreeOpcUa/opcua-asyncio: OPC UA library for python >= 3.7*. [Online]. Verfügbar unter: <https://github.com/FreeOpcUa/opcua-asyncio> (Zugriff am: 15. Februar 2023)
- [9] MQTT Version 5.0. [Online]. Verfügbar unter: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html> (Zugriff am: 16. Februar 2023).
- [10] I. Craggs, Eclipse Paho | *The Eclipse Foundation*. [Online]. Verfügbar unter: <https://www.eclipse.org/paho/index.php?page=clients/python/index.php> (Zugriff am: 16. Februar 2023)
- [11] Eclipse Mosquitto, *Eclipse Mosquitto*. [Online]. Verfügbar unter: <https://mosquitto.org/> (Zugriff am: 16. Februar 2023).

- [12] MongoDB, *MongoDB: The Developer Data Platform*. [Online]. Verfügbar unter: <https://www.mongodb.com/> (Zugriff am: 16. Februar 2023).
- [13] GitHub, *PyMongo*. [Online]. Verfügbar unter: <https://github.com/mongodb/mongo-python-driver> (Zugriff am: 16. Februar 2023).
- [14] ModBus, *ModBus*. [Online]. <https://modbus.org/> (Zugriff am: 16. Februar 2023).
- [15] GitHub, *sdm_modbus*. [Online]. Verfügbar unter: https://github.com/nmakel/sdm_modbus (Zugriff am: 16. Februar 2023).
- [16] Plotly, *Plotly Open Source Graphing Library for Python*. [Online]. Verfügbar unter: <https://plotly.com/python/> (Zugriff am: 16. Februar 2023).
- [17] Django, *Django documentation* [Online]. Verfügbar unter: <https://docs.djangoproject.com/en/4.1/> (Zugriff am: 16. Februar 2023).
- [18] Developer Mozilla, *Django introduction* [Online]. Verfügbar unter: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction> (Zugriff am: 16. Februar 2023).
- [19] Developer Mozilla, *An overview of HTTP* [Online]. Verfügbar unter: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview> (Zugriff am: 16. Februar 2023).
- [20] Developer Mozilla, *The WebSocket API (WebSockets)* [Online]. Verfügbar unter: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (Zugriff am: 16. Februar 2023).
- [21] BG-Etech: *Energiemessgerät SDM230-Modbus, Deutsche Original Montage- und Gebrauchsanleitung 08/2017*. Verfügbar unter: <https://bg-etech.de/download/manual/SDM230-Modbus.pdf> (Zugriff am: 24. Februar 2023).
- [22] TP-Link Corporation Limited: *TL-WR802N*. Verfügbar unter: <https://www.tp-link.com/de/home-networking/wifi-router/tl-wr802n/> (Zugriff am: 24. Februar 2023).
- [23] Energibutiken.Se: *Eastron SDM230 modbus MID Electricity meter single phase*. Verfügbar unter: <https://www.energibutiken.se/en/electricity-meters/210-electricity-meter-single-phase-eastron-sdm230-modbus-mid-01006.html> (Zugriff am: 24. Februar 2023).
- [24] KMtronic LTD: *USB to RS485 FTDI Interface Converter*. Verfügbar unter: https://www.kmtronic.com/all-products&product_id=71 (Zugriff am: 24. Februar 2023).

- [25] KMtronic LTD: *Modbus LAN TCP/IP to Modbus RS485 RTU Serial Converter*. Verfügbar unter: https://www.kmtronic.com/all-products?product_id=114 (Zugriff am: 24. Februar 2023).
- [26] V. Wesselak, S. Voswinckel, *Photovoltaik – Wie Sonne zu Strom wird*. Springer-Verlag Berlin Heidelberg 2016, ISBN: 978-3-662-48906-2, DOI: <https://doi-10.100344sj01d9.han.technikum-wien.at/10.1007/978-3-662-48906-2> (Zugriff am: 23. Februar 2023).
- [27] Bundesministerium für Klimaschutz, Umwelt, Energie, Mobilität, Innovation und Technologie (BMK), *Energie in Österreich 2021 - Zahlen, Daten, Fakten*. [Online]. Verfügbar unter: https://pvaustria.at/wp-content/uploads/Marktstatistik_2021.pdf (Zugriff am: 21. Februar 2023).
- [28] A. Wagner, *Photovoltaik Engineering: Handbuch für Planung, Entwicklung und Anwendung*. Springer Vieweg Berlin, Heidelberg, ISBN: 978-3-662-58455-2, DOI: <https://doi-10.100342sj01d1.han.technikum-wien.at/10.1007/978-3-662-58455-2> (Zugriff am: 23. Februar 2023).
- [29] Z. Abbas, K. Harijan, P.H. Shaikh, G. Valasai, F. Ali: *Effect of Ambient Temperature and Relative Humidity on Solar PV System Performance: A Case Study of Quaid-e-Azam Solar Park, Pakistan*. [Online]. Verfügbar unter: <https://doi.org/10.26692/sujo/2017.12.0047> (Zugriff am 27. Februar 2023)
- [30] H. Kazem, M. Chaichan: *Effect of humidity on photovoltaic performance based on experimental study*. [Online]. Verfügbar unter: https://www.researchgate.net/publication/289546072_Effect_of_humidity_on_photovoltaic_performance_based_on_experimental_study (Zugriff am 27. Februar 2023)
- [31] L. Wai Zhe, M. Yusoff, M. Irwanto, A.R. Amelia, S. Ibrahim: *Influence of wind speed on the performance of photovoltaic panel*. [Online]. Verfügbar unter: <https://doi.org/10.11591/ijeecs.v15.i1.pp62-70> Zugriff am 27. Februar 2023)
- [32] NASA/JPL-Caltech, *How Do Clouds Affect Solar Energy?* [Online]. Verfügbar unter: <https://scijinks.gov/solar-energy-and-clouds/> (Zugriff am: 18. Februar 2023)
- [33] Nazligül, Yavuzdeger, Esenboga, Ekinci, Dogru Mert, Demirdelen: *Investigation of the effect of dust and dirt accumulation on the electrical performance of solar PV panels*. Adana Alparslan Türkeş Science and Technology University, Turkey. [Online]. Verfügbar unter: https://www.researchgate.net/publication/358235136_investigation_of_the_effect_of_dust_and_dirt_accumulation_on_the_electrical_performance_of_solar_pv_panels (Zugriff am 27. Februar 2023)

- [34] A. Singh, V. Umakanth, N. Tyagi, A. K. Baghel, S. Kumar: *Comparative study of commercial crystalline solar cells*. [Online]. Verfügbar unter: <https://doi.org/10.1016/j.rio.2023.100379> (Zugriff am 27. Februar 2023)
- [35] Fraunhofer ISE: *Ertragsvergleich von bifazialen Systemen und Standardsystemen*. [Online]. Verfügbar unter: <https://www.ise.fraunhofer.de/de/geschaeftsfelder/photovoltaik/photovoltaische-module-und-kraftwerke/photovoltaische-kraftwerke/ertragsgutachten-auch-bifaziale-module.html> (Zugriff am 27. Februar 2023)
- [36] M. Cecile, J. Oh, F.I. Mahmood, M. Li, P. Hacke, F. Li, R. Smith, D. Colvin, M. Matam, C. DiRubio, G. Tamizhmani, G. Seigneur: *Review of Potential-Induced Degradation in Bifacial PV Modules*. [Online]. Verfügbar unter: <https://doi.org/10.1002/ente.202200943> (Zugriff am 27. Februar 2023)
- [37] R. Yakubu, L. Mensah, D. Quansah, M. Adaramola: *Improving solar photovoltaic installation energy yield using bifacial modules and tracking systems: An analytical approach*. [Online]. Verfügbar unter: <https://doi.org/10.1177/168781322211397> (Zugriff am 27. Februar 2023)
- [38] Konrad, F.: *Planung von Photovoltaik-Anlagen*. Vieweg 2007.
- [39] SolarV GmbH: *enjoysolar® Monokristallines Solarmodul 50W 12V*. [Online]. <https://solarv.de/product/monokristalline-solarmodul-50w/> (Zugriff am 15. März 2023)
- [40] A2O Distribution - Robot-Advance: *Raspberry Pi 4 model B 4Go technical specifications*. [Online]. <https://www.robot-advance.com/userfiles/www.robot-advance.com/images/raspberry-4-modele-b-4go.jpg> (Zugriff am 15. März 2023)
- [41] Raspberry Pi Foundation: *Raspberry Pi 4*. [Online]. <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/> (Zugriff am 15. März 2023)
- [42] Alibaba Group: *Voltage Sensor*. [Online]. <https://de.aliexpress.com/item/32884941176.html> (Zugriff am 15. März 2023)
- [43] Eckstein GmbH: *30A Stromsensor ACS712-30 Current Sensor mit Analogausgang*. [Online]. <https://eckstein-shop.de/30AStromsensorACS712-30CurrentSensormitAnalogausgang> (Zugriff am 15. März 2023)
- [44] AZ-Delivery Vertriebs GmbH: *ADS1115 ADC Module 16Bit 4 channels for Raspberry Pi*. [Online]. <https://www.az-delivery.de/en/products/analog-digitalwandler-ads1115-mit-i2c-interface> (Zugriff am 15. März 2023)

- [45] AZ-Delivery Vertriebs GmbH: *DHT22 AM2302 Temperature sensor and air humidity sensor with a circuit board and cable compatible with Arduino and Raspberry Pi*. [Online]. https://cdn.shopify.com/s/files/1/1509/1638/files/DHT_22_-_AM2302_Temperatur_und_Luftfeuchtigkeitssensor_Datenblatt.pdf (Zugriff am 15. März 2023)
- [46] fischertechnik GmbH: *Datenblatt Umweltsensor BME680*. [Online]. https://content.ugfischer.com/cbfiles/fischer/Zulassungen/ft/167358_Environmental_sensor.pdf (Zugriff am 20. März 2023)
- [47] fischertechnik GmbH: *Datenblatt Fotowiderstand LDR03*. [Online]. <https://content.ugfischer.com/cbfiles/fischer/Zulassungen/ft/32698-Photoresistor-LDR03.pdf> (Zugriff am 20. März 2023)
- [48] InfluxData Inc: *InfluxDB is 5x Faster vs. MongoDB for Time Series Workloads*. [Online]. <https://www.influxdata.com/blog/influxdb-is-27x-faster-vs-mongodb-for-time-series-workloads/> (Zugriff am 22. März 2023)

Anhang A: 01_dummy_consumption.py

```
import sys
import time
import json
import random
import logging
import datetime
import paho.mqtt.client as mqtt

def read_config():
    """Read configuration data from JSON file"""
    f = open (
        "/usr/src/app/mqtt_base/mqtt_config.json",
        "r",
        encoding="utf-8",
    )
    config_data = json.load(f)
    address, port, timeout = (
        config_data["address"],
        config_data["port"],
        config_data["timeout"],
    )
    return address, port, timeout

def on_connect(client, userdata, flags, rc):
    """Subscribe to "pv_modul/energy_consumption" topic when client connects to MQTT broker"""
    logging.info("Connected with result code " + str(rc))
    client.subscribe("pv_modul/energy_consumption")

if __name__ == "__main__":
    try:
        client = mqtt.Client()
        client.on_connect = on_connect
        address, port, timeout = read_config()
        client.connect(address, port, timeout)

        # Publish data to MQTT broker
        while True:
            voltage, current, temperature = (
                random.uniform(229, 231),
                random.uniform(0.18, 0.19),
                random.uniform(25, 27),
            )
            ts = datetime.datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ")
            datapoint = json.dumps(
                {
                    "voltage": voltage,
                    "current": current,
                    "temperature": temperature,
                    "ts": ts
                }
            )
            client.publish("pv_modul/energy_consumption", datapoint)
            time.sleep(1)

    except:
        exc_obj, tb = sys.exc_info()[1:]
        lineno = tb.tb_lineno
        filename = tb.tb_frame.f_code.co_filename
        logging.error(f"({filename}, LINE {lineno}): {exc_obj}")
```

Anhang B: 01_dummy_production.py

```
import sys
import time
import json
import random
import logging
import datetime
import paho.mqtt.client as mqtt

def read_config():
    """Read configuration data from JSON file"""
    f = open(
        "/usr/src/app/mqtt_base/mqtt_config.json",
        "r",
        encoding="utf-8",
    )
    config_data = json.load(f)
    address, port, timeout = (
        config_data["address"],
        config_data["port"],
        config_data["timeout"],
    )
    return address, port, timeout

def on_connect(client, userdata, flags, rc):
    """Subscribe to "pv_modul/energy_production" topic when client connects to MQTT broker"""
    logging.info("Connected with result code " + str(rc))
    client.subscribe("pv_modul/energy_production")

if __name__ == "__main__":
    try:
        client = mqtt.Client()
        client.on_connect = on_connect
        address, port, timeout = read_config()
        client.connect(address, port, timeout)

        # Publish data to MQTT broker
        while True:
            voltage, current = random.uniform(11, 12), random.uniform(1, 2)
            ts = datetime.datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ")
            datapoint = json.dumps({"voltage": voltage, "current": current, "ts": ts})
            client.publish("pv_modul/energy_production", datapoint)
            time.sleep(1)

    except:
        exc_obj, tb = sys.exc_info()[1:]
        lineno = tb.tb_lineno
        filename = tb.tb_frame.f_code.co_filename
        logging.error(f"({filename}, LINE {lineno}): {exc_obj}")
```

Anhang C: 01_real_consumption.py

```
import sys
import time
import json
import logging
import datetime
import sdm_modbus
import paho.mqtt.client as mqtt

def read_config():
    """Read configuration data from JSON file"""
    f = open(
        "/usr/src/app/mqtt_base/mqtt_config.json",
        "r",
        encoding="utf-8",
    )
    config_data = json.load(f)
    address, port, timeout = (
        config_data["address"],
        config_data["port"],
        config_data["timeout"],
    )
    return address, port, timeout

def on_connect(client, userdata, flags, rc):
    """Subscribe to "pv_modul/energy_consumption" topic when client connects to MQTT broker"""
    logging.info("Connected with result code " + str(rc))
    client.subscribe("pv_modul/energy_consumption")

if __name__ == "__main__":
    try:
        client = mqtt.Client()
        client.on_connect = on_connect
        address, port, timeout = read_config()
        client.connect(address, port, timeout)

        # Publish energy meter SDM230 data to MQTT Broker
        device = sdm_modbus.SDM230(host="192.168.0.199", port=502)
        while True:
            if device.connected() == True:
                data = device.read_all(sdm_modbus.registerType.INPUT)
                voltage, current = device.read("voltage"), device.read("current")
                ts = datetime.datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ")
                datapoint = json.dumps(
                    {"voltage": voltage, "current": current, "ts": ts}
                )
                client.publish("pv_modul/energy_consumption", datapoint)
                time.sleep(0.4)
            else:
                device = sdm_modbus.SDM230(host="192.168.0.199", port=502)
                logging.error("Device not connected, retrying...")

    except:
        exc_obj, tb = sys.exc_info()[1:]
        lineno = tb.tb_lineno
        filename = tb.tb_frame.f_code.co_filename
        logging.error(f"({filename}, LINE {lineno}): {exc_obj}")
```

Anhang D: 01_real_production.py

```
import sys
import time
import json
import board
import busio
import logging
import datetime
import Adafruit_DHT
import paho.mqtt.client as mqtt
import adafruit_ads1x15.ads1115 as ADS
from adafruit_ads1x15.analog_in import AnalogIn

def read_config():
    """Read configuration data from JSON file"""
    f = open(
        "/home/admin/MINDfactoree/mqtt_config.json",
        "r",
        encoding="utf-8",
    )
    config_data = json.load(f)
    address, port, timeout = (
        config_data["address"],
        config_data["port"],
        config_data["timeout"],
    )
    return address, port, timeout

def on_connect(client, userdata, flags, rc):
    """Subscribe to "pv_modul/energy_production" topic when client connects to MQTT broker"""
    logging.info("Connected with result code " + str(rc))
    client.subscribe("pv_modul/energy_production")

# Set DHT Sensor and Pin
DHT_SENSOR, DHT_PIN = Adafruit_DHT.DHT22, 4

# Resistors Voltage Sensor [Ohm]
R1, R2 = 30000, 7500

# Initialize I2C bus, ADC
i2c = busio.I2C(board.SCL, board.SDA)
ads = ADS.ADS1115(i2c)
adc_0, adc_1 = AnalogIn(ads, ADS.P0), AnalogIn(ads, ADS.P1)

if __name__ == "__main__":
    while True:
        try:
            client = mqtt.Client()
            client.on_connect = on_connect
            address, port, timeout = read_config()
            client.connect(address, port, timeout)
            humidity, temperature = Adafruit_DHT.read_retry(
                DHT_SENSOR, DHT_PIN)
            channel_0_voltage = adc_0.voltage * ((R1 + R2) / R2)
            channel_1_current = (adc_1.voltage - 2.5) / 0.066
            ts_info = datetime.datetime.now().strftime("%H:%M:%S")
            ts = datetime.datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ")
            datapoint = json.dumps(
                {
                    "voltage": channel_0_voltage,
                    "current": channel_1_current,
                    "temperature": temperature,
                    "ts": ts
                }
            )
```

```
)
client.publish("pv_modul/energy_production", datapoint)
time.sleep(0.5)

except:
    exc_obj, tb = sys.exc_info()[1:]
    lineno = tb.tb_lineno
    filename = tb.tb_frame.f_code.co_filename
    logging.error(f"({filename}, LINE {lineno}): {exc_obj}")
```

Anhang E: 02_consumption_to_mongodb.py

```
import sys
import time
import json
import logging
from pymongo import MongoClient
import paho.mqtt.client as mqtt

def read_config():
    """Read configuration data from JSON file"""
    f = open(
        "/usr/src/app/mqtt_base/mqtt_config.json",
        "r",
        encoding="utf-8",
    )
    config_data = json.load(f)
    address, port, timeout = (
        config_data["address"],
        config_data["port"],
        config_data["timeout"],
    )
    return address, port, timeout

def on_connect(client, userdata, flags, rc):
    """Subscribe to "pv_modul/energy_consumption" topic when client connects to MQTT broker"""
    logging.info("Connected with result code " + str(rc))
    client.subscribe("pv_modul/energy_consumption")

def on_message(client, userdata, msg):
    """Insert JSON payload in "pv_modul/energy_consumption" mongodb collection when new message from
    MQTT is received"""
    received_message = json.loads((msg.payload).decode("utf-8"))
    client = MongoClient("mongodb://root:root@mongo:27017/")
    collection = client["pv_modul"]["energy_consumption"]
    collection.insert_one(received_message)
    time.sleep(0.2)

if __name__ == "__main__":
    try:
        client = mqtt.Client()
        client.on_connect = on_connect
        client.on_message = on_message
        address, port, timeout = read_config()
        client.connect(address, port, timeout)
        client.loop_forever()
    except:
        exc_obj, tb = sys.exc_info()[1:]
        lineno = tb.tb_lineno
        filename = tb.tb_frame.f_code.co_filename
        logging.error(f"({filename}, LINE {lineno}): {exc_obj}")
```

Anhang F: 02_production_to_mongodb.py

```
import sys
import time
import json
import logging
from pymongo import MongoClient
import paho.mqtt.client as mqtt

def read_config():
    """Read configuration data from JSON file"""
    f = open(
        "/usr/src/app/mqtt_base/mqtt_config.json",
        "r",
        encoding="utf-8",
    )
    config_data = json.load(f)
    address, port, timeout = (
        config_data["address"],
        config_data["port"],
        config_data["timeout"],
    )
    return address, port, timeout

def on_connect(client, userdata, flags, rc):
    """Subscribe to "pv_modul/energy_production" topic when client connects to MQTT broker"""
    logging.info("Connected with result code " + str(rc))
    client.subscribe("pv_modul/energy_production")

def on_message(client, userdata, msg):
    """Insert JSON payload in "pv_modul/energy_production" mongodb collection when new message from
    MQTT is received"""
    received_message = json.loads((msg.payload).decode("utf-8"))
    client = MongoClient("mongodb://root:root@mongo:27017/")
    collection = client["pv_modul"]["energy_production"]
    collection.insert_one(received_message)
    time.sleep(1)

if __name__ == "__main__":
    try:
        client = mqtt.Client()
        client.on_connect = on_connect
        client.on_message = on_message
        address, port, timeout = read_config()
        client.connect(address, port, timeout)
        client.loop_forever()
    except:
        exc_obj, tb = sys.exc_info()[1:]
        lineno = tb.tb_lineno
        filename = tb.tb_frame.f_code.co_filename
        logging.error(f"({filename}, LINE {lineno}): {exc_obj}")
```

Anhang G: 02_bme_to_mongodb.py

```
import sys
import time
import json
import logging
import datetime
from pymongo import MongoClient
import paho.mqtt.client as mqtt

def read_config():
    """Read configuration data from JSON file"""
    f = open(
        "/usr/src/app/mqtt_base/mqtt_config.json",
        "r",
        encoding="utf-8",
    )
    config_data = json.load(f)
    address, port, timeout = (
        config_data["address"],
        config_data["port"],
        config_data["timeout"],
    )
    return address, port, timeout

def on_connect(client, userdata, flags, rc):
    """Subscribe to "i/bme680" topic when client connects to MQTT broker"""
    logging.info("Connected with result code " + str(rc))
    client.subscribe("i/bme680")

def on_message(client, userdata, msg):
    """Insert JSON payload in "pv_modul/bme680" mongodb collection when new message from MQTT is received"""
    received_message = json.loads((msg.payload).decode("utf-8"))
    temperature, humidity, air_quality = (
        received_message["t"],
        received_message["h"],
        received_message["iaq"],
    )
    ts = datetime.datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ")
    datapoint = {
        "temperature": temperature,
        "humidity": humidity,
        "air_quality": air_quality,
        "ts": ts,
    }
    client = MongoClient("mongodb://root:root@mongo:27017/")
    collection = client["pv_modul"]["bme680"]
    collection.insert_one(datapoint)
    time.sleep(1)

if __name__ == "__main__":
    try:
        client = mqtt.Client()
        client.on_connect = on_connect
        client.on_message = on_message
        address, port, timeout = read_config()
        client.connect(address, port, timeout)
        client.loop_forever()
    except:
        exc_obj, tb = sys.exc_info()[1:]
        lineno = tb.tb_lineno
        filename = tb.tb_frame.f_code.co_filename
        logging.error(f"({filename}, LINE {lineno}): {exc_obj}")
```


Anhang H: 02_ldr_to_mongodb.py

```
import sys
import time
import json
import logging
import datetime
from pymongo import MongoClient
import paho.mqtt.client as mqtt

def read_config():
    """Read configuration data from JSON file"""
    f = open(
        "/usr/src/app/mqtt_base/mqtt_config.json",
        "r",
        encoding="utf-8",
    )
    config_data = json.load(f)
    address, port, timeout = (
        config_data["address"],
        config_data["port"],
        config_data["timeout"],
    )
    return address, port, timeout

def on_connect(client, userdata, flags, rc):
    """Subscribe to "i/ldr" topic when client connects to MQTT broker"""
    logging.info("Connected with result code " + str(rc))
    client.subscribe("i/ldr")

def on_message(client, userdata, msg):
    """Insert JSON payload in "pv_modul/ldr" mongodb collection when new message from MQTT is received"""
    received_message = json.loads((msg.payload).decode("utf-8"))
    brightness = received_message["br"]
    ts = datetime.datetime.now().strftime("%Y-%m-%dT%H:%M:%SZ")
    datapoint = {"brightness": brightness, "ts": ts}
    client = MongoClient("mongodb://root:root@mongo:27017/")
    collection = client["pv_modul"]["ldr"]
    collection.insert_one(datapoint)
    time.sleep(1)

if __name__ == "__main__":
    try:
        client = mqtt.Client()
        client.on_connect = on_connect
        client.on_message = on_message
        address, port, timeout = read_config()
        client.connect(address, port, timeout)
        client.loop_forever()
    except:
        exc_obj, tb = sys.exc_info()[1:]
        lineno = tb.tb_lineno
        filename = tb.tb_frame.f_code.co_filename
        logging.error(f"({filename}, LINE {lineno}): {exc_obj}")
```

Anhang I: views.py

```
import json
import plotly
import requests
from typing import Any
import plotly.graph_objects as go
from bs4 import BeautifulSoup as bs
from scipy.signal import savgol_filter
from datetime import datetime, timedelta
from pymongo import MongoClient, DESCENDING
from django.views.generic import TemplateView

class PVBaseView(TemplateView):
    """Django TemplateView to display mindfactoree performance and weather forecast data"""

    template_name = "index_pv_modul.html"

    def get_context_data(self, **kwargs: Any) -> dict[str, Any]:
        """Get context data for rendering the template"""
        data = super().get_context_data(**kwargs)
        (
            shortwave_radiation_time,
            shortwave_radiation_list_smoothed,
        ) = self.get_shortwave_radiation()
        data["forecast_chart"] = self.forecast_chart_html(
            shortwave_radiation_time, shortwave_radiation_list_smoothed
        )
        data["weather"] = self.get_weather()
        (
            recent_energy_consumption,
            recent_energy_production,
            recent_battery_temperature,
            recent_brightness,
            recent_temperature,
            recent_humidity,
            recent_air_quality,
            energy_consumption_ts,
            energy_consumption_powers,
            energy_production_ts,
            energy_production_powers,
            energy_consumption_moving_average,
            energy_production_moving_average,
            average_energy_production_power_last_hour,
        ) = self.get_performance()
        data["recent_energy_consumption"] = f"{recent_energy_consumption}W"
        data["recent_energy_production"] = f"{recent_energy_production}W"
        data[
            "average_energy_production"
        ] = f"{average_energy_production_power_last_hour}Wh"
        data["recent_battery_temperature"] = f"{recent_battery_temperature}°C"
        data["recent_temperature"] = f"{recent_temperature}°C"
        data["recent_humidity"] = f"{recent_humidity}% r.H."
        data["recent_air_quality"] = f"{recent_air_quality}"
        data["recent_brightness"] = f"{recent_brightness}%"
        data["performance_chart"] = self.performance_chart_html(
            energy_consumption_ts,
            energy_consumption_powers,
            energy_consumption_moving_average,
            energy_production_ts,
            energy_production_powers,
            energy_production_moving_average,
        )
        data["current_time"] = datetime.now().strftime("%H:%M:%S")
        return data

    def get_shortwave_radiation(self):
        """Get shortwave radiation from open-meteo and smooth it with Savitzky-Golay filter"""
        response = requests.get(
```

```

        "https://api.open-
meteo.com/v1/forecast?latitude=48.192440&longitude=16.37736&timezone=Europe/Vienna&hourly=shortwave_rad
iation",
        timeout=5,
    ).json()
    efficiency, pv_panel_area = 0.18, 0.3082
    shortwave_radiation_time = response["hourly"]["time"]
    shortwave_radiation_raw = response["hourly"]["shortwave_radiation"]
    shortwave_radiation_list = [
        value * efficiency * pv_panel_area for value in shortwave_radiation_raw
    ]
    shortwave_radiation_list_smoothed = savgol_filter(
        shortwave_radiation_list, window_length=7, polyorder=2
    )
    shortwave_radiation_list_smoothed = [
        0 if val < 0 else val for val in shortwave_radiation_list_smoothed
    ]
    return shortwave_radiation_time, shortwave_radiation_list_smoothed

def forecast_chart_html(self, time, value):
    """Generate HTML containing forecast chart using Plotly"""
    fig = go.Figure()
    fig.add_trace(
        go.Scatter(
            x=time,
            y=value,
            mode="lines",
            name="0.31 m²",
            marker_color="blue",
            fill="tozeroy",
        )
    )
    fig.update_layout(margin=dict(l=0, r=0, b=0, t=0)) # left, right, bottom, top
    fig.update_layout({"paper_bgcolor": "rgba(0,0,0,0)"}, showlegend=True)
    fig.update_layout(
        legend=dict(
            x=1,
            y=1,
            traceorder="reversed",
            font=dict(
                size=12,
            ),
            bgcolor="White",
            bordercolor="Grey",
            borderwidth=1,
        ),
        xaxis=dict(
            range=[datetime.now(), datetime.now() + timedelta(hours=12)],
            showgrid=True,
            gridcolor="white",
            showline=True,
            linecolor="grey",
            rangeselector=dict(
                buttons=list(
                    [
                        dict(count=12, label="12h", step="hour", stepmode="todate"),
                        dict(count=24, label="1d", step="hour", stepmode="todate"),
                        dict(count=7, label="1w", step="all", stepmode="todate"),
                    ]
                )
            ),
            rangeslider=dict(visible=True),
        ),
        yaxis=dict(
            title="Leistung [W]",
            rangemode="tozero",
            showgrid=True,
            gridcolor="white",
            showline=True,
            linecolor="grey",
        ),
    ),

```

```

    )
    html = fig.to_html(include_plotlyjs=False, default_height=300)
    chart_data = json.dumps(fig.data, cls=plotly.utils.PlotlyJSONEncoder)
    chart_layout = json.dumps(fig.layout, cls=plotly.utils.PlotlyJSONEncoder)
    html = f"<div id=\"chart-div\" data-chart-data='{chart_data}' data-chart-
layout='{chart_layout}'>{html}</div>"
    return html

def get_weather(self):
    """Get weather data for the next 4 hours from wetter.tv"""
    weather = []
    response = requests.get("https://wetter.tv/de-AT/p14/wien", timeout=5)
    if response.status_code == 200:
        soup = bs(response.text, features="html.parser")
        hours = soup.find("tbody").find_all("tr")
        for hour in hours:
            time = (
                hour.find(
                    "td",
                    {"class": "hourly-overview-td hourly-overview-td--timepoint"},
                )
                .getText()
                .replace("\n", "")
                .replace(" ", "")
            )
            image = hour.find("img", {"class": "hourly-overview-weather-icon"})[
                "lazy-src"
            ]
            temperature = (
                hour.find(
                    "td", {"class": "hourly-overview-td hourly-overview-td--temp"}
                )
                .getText()
                .replace("\n", "")
                .replace(" ", "")
                + "C"
            )
            wind = (
                hour.find(
                    "td", {"class": "hourly-overview-td hourly-overview-td--wind"}
                )
                .find("span", {"class": "font-weight-semi-bold"})
                .getText()
                .replace("\n", "")
                .replace(" ", "")
            )
            wind_direction = (
                hour.find(
                    "td", {"class": "hourly-overview-td hourly-overview-td--wind"}
                )
                .find(
                    "span",
                    {
                        "class": "font-weight-light secondary d-none d-sm-inline d-md-none d-lg-
inline"
                    }
                ),
            )
            .getText()
            .replace("\n", "")
            .replace(" ", "")
        )

        weather.append(
            {
                "time": time,
                "image": image,
                "temperature": temperature,
                "wind": f"{wind}km/h",
                "wind_direction": wind_direction,
            }
        )

```

```

weather_trimmed = weather[:4]
return weather_trimmed

def get_performance(self):
    """Get energy_consumption, energy_production, bme680 and ldr data from MongoDB database"""
    db_client = MongoClient("mongodb://root:root@mongo:27017/")
    energy_consumption_db = db_client.pv_modul.energy_consumption
    energy_production_db = db_client.pv_modul.energy_production
    bme680_db = db_client.pv_modul.bme680
    ldr_db = db_client.pv_modul.ldr

    energy_consumption_list = list(
        energy_consumption_db.find({}).sort([("ts", DESCENDING)])
    )
    energy_production_list = list(
        energy_production_db.find({}).sort([("ts", DESCENDING)])
    )
    bme680_list = list(bme680_db.find({}).sort([("ts", DESCENDING)]))
    ldr_list = list(ldr_db.find({}).sort([("ts", DESCENDING)]))

    recent_energy_consumption = round(
        energy_consumption_list[0]["voltage"]
        * energy_consumption_list[0]["current"],
        2,
    )
    recent_energy_production = round(
        energy_production_list[0]["voltage"] * energy_production_list[0]["current"],
        2,
    )
    recent_battery_temperature = round(energy_production_list[0]["temperature"], 2)
    recent_brightness = ldr_list[0]["brightness"]
    recent_temperature = bme680_list[0]["temperature"]
    recent_humidity = bme680_list[0]["humidity"]
    recent_air_quality = bme680_list[0]["air_quality"]

    energy_consumption_ts = [i["ts"] for i in energy_consumption_list]
    energy_consumption_powers = [
        i["current"] * i["voltage"] for i in energy_consumption_list
    ]
    energy_production_ts = [i["ts"] for i in energy_production_list]
    energy_production_powers = [
        i["current"] * i["voltage"] for i in energy_production_list
    ]

    energy_consumption_moving_average = savgol_filter(
        energy_consumption_powers, window_length=61, polyorder=2
    )
    energy_production_moving_average = savgol_filter(
        energy_production_powers, window_length=61, polyorder=2
    )

    energy_production_ts = [
        datetime.fromisoformat(t.replace("Z", "+00:00"))
        for t in energy_production_ts
    ]
    last_time = energy_production_ts[0]
    start_time = last_time - timedelta(hours=1)
    energy_production_powers_last_hour = [
        energy_production_powers[i]
        for i, t in enumerate(energy_production_ts)
        if t >= start_time
    ]
    average_energy_production_power_last_hour = round(
        (
            sum(energy_production_powers_last_hour)
            / len(energy_production_powers_last_hour)
        ),
        2,
    )

    return (

```

```

        recent_energy_consumption,
        recent_energy_production,
        recent_battery_temperature,
        recent_brightness,
        recent_temperature,
        recent_humidity,
        recent_air_quality,
        energy_consumption_ts,
        energy_consumption_powers,
        energy_production_ts,
        energy_production_powers,
        energy_consumption_moving_average,
        energy_production_moving_average,
        average_energy_production_power_last_hour,
    )

def performance_chart_html(
    self, time_1, value_1, average_1, time_2, value_2, average_2
):
    """Generate HTML containing performance_chart using Plotly"""
    fig = go.Figure()
    fig.add_trace(
        go.Scatter(
            x=time_1,
            y=value_1,
            mode="lines",
            name="Aufnahme",
            marker_color="red",
        )
    )
    fig.add_trace(
        go.Scatter(
            x=time_1,
            y=average_1,
            mode="lines",
            name="Aufnahme geglättet",
            marker_color="darkred",
        )
    )
    fig.add_trace(
        go.Scatter(
            x=time_2,
            y=value_2,
            mode="lines",
            name="Produktion",
            marker_color="green",
        )
    )
    fig.add_trace(
        go.Scatter(
            x=time_2,
            y=average_2,
            mode="lines",
            name="Produktion geglättet",
            marker_color="darkgreen",
        )
    )
    fig.update_layout(margin=dict(l=0, r=0, b=0, t=0))
    fig.update_layout({"paper_bgcolor": "rgba(0,0,0,0)"})
    fig.update_layout(
        legend=dict(
            x=1,
            y=1,
            traceorder="reversed",
            font=dict(
                size=12,
            ),
            bgcolor="White",
            bordercolor="Grey",
            borderwidth=1,
        ),
    ),

```

```

xaxis=dict(
    range=[datetime.now() - timedelta(minutes=15), datetime.now()],
    showgrid=True,
    gridcolor="white",
    showline=True,
    linecolor="grey",
    rangeselector=dict(
        buttons=list(
            [
                dict(
                    count=15,
                    label="15min",
                    step="minute",
                    stepmode="backward",
                ),
                dict(count=1, label="1h", step="hour", stepmode="backward"),
                dict(
                    count=12, label="12h", step="hour", stepmode="backward"
                ),
                dict(count=1, label="1d", step="day", stepmode="backward"),
                dict(count=7, label="1w", step="day", stepmode="backward"),
                dict(step="all"),
            ]
        )
    ),
),
yaxis=dict(
    title="Leistung [W]",
    showgrid=True,
    gridcolor="white",
    showline=True,
    linecolor="grey",
),
)
html = fig.to_html(include_plotlyjs=False, default_height=300)
return html

```

Anhang J: index_pv_modul.html

```
<!DOCTYPE html>
{% extends "base.html" %}
{% load static %}
{% block body_block %}

<body>
  <div class="row">
    <div class="col">
      <div class="card mt-2 border-dark">
        <div class="card-header">
          <div class="row mt-2">
            <div class="col-9 d-flex align-items-stretch">
              <div class="card mb-3 border-dark flex-fill">
                <div class="card-header">
                  <h3 class="text-left">Monitoring</h3>
                  <p class="card-text"><small class="text-muted">Last update:
                    {{ current_time }}</small></p>
                </div>
                <div class="card-body bg-silvergrey border-dark">
                  <div class="row">
                    <div class="col-3 border-dark">
                      <div class="card border-grey mb-2" style="max-width: 18rem;">
                        <div class="card-header text-center" style="font-size:
15px;">Leistungsproduktion &#9889</div>
                        <div class="card-body">
                          <h5 class="card-title text-center">{{ recent_energy_production }}</h5>
                        </div>
                      </div>
                    </div>
                    <div class="col-3 border-dark">
                      <div class="card border-grey" style="max-width: 18rem;">
                        <div class="card-header text-center" style="font-size: 15px;">Energieproduktion
(1h) &#128267
                        <div class="card-body">
                          <h5 class="card-title text-center">{{ average_energy_production }}</h5>
                        </div>
                      </div>
                    </div>
                    <div class="col-3 border-dark">
                      <div class="card border-grey" style="max-width: 18rem;">
                        <div class="card-header text-center" style="font-size: 15px;">Fabriktemperatur
&#127777</div>
                        <div class="card-body">
                          <h5 class="card-title text-center">{{ recent_temperature }}</h5>
                        </div>
                      </div>
                    </div>
                    <div class="col-3 border-dark">
                      <div class="card border-grey" style="max-width: 18rem;">
                        <div class="card-header text-center" style="font-size: 15px;">Luftfeuchtigkeit
&#128167</div>
                        <div class="card-body">
                          <h5 class="card-title text-center">{{ recent_humidity }}</h5>
                        </div>
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
    <div class="col">
      <div class="card mt-2 border-dark">
        <div class="card-header">
          <div class="row mt-2">
            <div class="col-9 d-flex align-items-stretch">
              <div class="card mb-3 border-dark flex-fill">
                <div class="card-header">
                  <h3 class="text-left">Monitoring</h3>
                  <p class="card-text"><small class="text-muted">Last update:
                    {{ current_time }}</small></p>
                </div>
                <div class="card-body bg-silvergrey border-dark">
                  <div class="row">
                    <div class="col-3 border-dark">
                      <div class="card border-grey mb-2" style="max-width: 18rem;">
                        <div class="card-header text-center" style="font-size:
15px;">Leistungsaufnahme
&#127981</div>
                        <div class="card-body">
                          <h5 class="card-title text-center">{{ recent_energy_consumption }}</h5>
                        </div>
                      </div>
                    </div>
                    <div class="col-3 border-dark">
                      <div class="card border-grey" style="max-width: 18rem;">
                        <div class="card-header text-center" style="font-size: 15px;">Energieverbrauch
(1h) &#128267
                        <div class="card-body">
                          <h5 class="card-title text-center">{{ average_energy_consumption }}</h5>
                        </div>
                      </div>
                    </div>
                    <div class="col-3 border-dark">
                      <div class="card border-grey" style="max-width: 18rem;">
                        <div class="card-header text-center" style="font-size: 15px;">Temperatur
&#128167</div>
                        <div class="card-body">
                          <h5 class="card-title text-center">{{ recent_temperature }}</h5>
                        </div>
                      </div>
                    </div>
                  </div>
                </div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</body>

{% endblock %}
```



```

        <div class="col-3 border-dark">
            <div class="card border-grey" style="max-width: 18rem;">
                <div class="card-header text-center" style="font-size:
15px;">Batterietemperatur &#127777</div>
                <div class="card-body">
                    <h5 class="card-title text-center">{{ recent_battery_temperature }}</h5>
                </div>
            </div>
        <div class="col-3 border-dark">
            <div class="card border-grey" style="max-width: 18rem;">
                <div class="card-header text-center" style="font-size: 15px;">Helligkeit
&#128161</div>
                <div class="card-body">
                    <h5 class="card-title text-center">{{ recent_brightness }}</h5>
                </div>
            </div>
        <div class="col-3 border-dark" data-toggle="modal" data-target="#exampleModal">
            <div class="card border-grey" style="max-width: 18rem;">
                <div class="card-header text-center" style="font-size: 15px;">Luftqualität
&#128168</div>
                <div class="card-body">
                    <h5 class="card-title text-center">{{ recent_air_quality }}</h5>
                    <div class="modal fade" id="exampleModal" tabindex="-1" role="dialog"
                        aria-labelledby="exampleModalLabel" aria-hidden="true">
                        <div class="modal-dialog modal-xl text-center" role="document">
                            <div class="modal-content">
                                <div class="modal-body">
                                    
                                </div>
                                <div class="modal-footer">
                                    <button type="button" class="btn btn-secondary" data-
dismiss="modal">Close</button>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
    <div class="col-3 d-flex" style="max-height: 100%;">
        <div class="card mb-3 border-dark flex-fill">
            <div class="card-header">
                <div class="row">
                    <div class="col text-center">
                        <h3 class="text-center">Wetter</h3>
                        <p class="card-text"><small class="text-muted">Quelle: <a
                            href="https://wetter.tv/de-AT/p14/wien">Wetter.tv</a></small></p>
                    </div>
                </div>
            </div>
            <div class="card-body bg-light" style="max-height: 100%; padding-top: 8px; padding-
bottom: 0px;">
                <div class="container d-flex justify-content-center">
                    <div class="center">
                        <table style="font-size: 16px;">
                            <tbody>
                                <tr>
                                    <th class="text-center" style="padding-left: 6px; padding-right:
6px;">Zeit</th>
                                    <th class="text-center" style="padding-left: 6px; padding-right:
6px;">Wetter</th>
                                    <th class="text-center" style="padding-left: 6px; padding-right:
6px;">Temp.</th>

```

```

        <th class="text-center" style="padding-left: 6px; padding-right:
6px;">Wind</th>
    </tr>
    {% for item in weather %}
    <tr>
        <td class="align-middle text-center" style="padding-left: 6px; padding-
right: 6px;">
            <div class="time">
                <span style="height: 40px !important;">{{ item.time }}</span>
            </div>
        </td>
        <td class="align-middle text-center" style="padding-left: 10px; padding-
right: 10px;">
            
        </td>
        <td class="align-middle text-center" style="padding-left: 10px; padding-
right: 10px;">
            <span class="temperature">{{ item.temperature }}</span><br>
        </td>
        <td class="align-middle text-center" style="padding-left: 10px; padding-
right: 10px;">
            <span class="wind">{{ item.wind }}<br><small>{{ item.wind_direction
}}</small></span><br>
        </td>
    </tr>
    {% endfor %}
</tbody>
</table>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
<div class="row">
    <div class="col d-flex align-items-stretch">
        <div class="card mb-3 border-dark flex-fill">
            <div class="card-header">
                <h3 class="text-left">Performance</h3>
                <p class="card-text"><small class="text-muted">Gegenüberstellung von
Leistungsproduktion der PV-Anlage und Leistungsaufnahme der Fabrik.</small>
                <br>
                <small><small class="text-muted">Anmerkung: Die Funktion wurde mit dem Savitzky-
Golay-Filter
                    (Fensterlänge=61, Polynomordnung=2) geglättet.</small></small>
                </p>
            </div>
            <div class="card-body bg-light id="chart1">
                {% autoescape off %}
                {{ performance_chart }}
                {% endautoescape %}
            </div>
        </div>
    </div>
    <div class="col">
        <div class="card mb-2 border-dark">
            <div class="card-header">
                <div class="row">
                    <div class="col-9 text-left">
                        <h3 class="text-left">Wochenprognose</h3>
                        <p class="card-text"><small class="text-muted">Theoretische Maximalleistung der
PV-Anlage basierend auf Globalstrahlung, der durch PV-Module abgedeckten Fläche
und
                            Wirkungsgrad (18%).</small>
                        <br>
                        <small class="text-muted">Quelle: <a href="https://open-meteo.com/">Open-
Meteo</small></a>
                        <br>
                        <small><small class="text-muted">Anmerkung: Die Funktion wurde mit dem Savitzky-
Golay-Filter

```

```

(Fensterlänge=7, Polynomordnung=2) geglättet.</small></small>
    <br>
  </p>
</div>
<div class="col text-right">
  <div>
    <div>
      <label for="basic-url">PV-Modulfäche in m²</label>
      <div class="input-group">
        <input type="text" id="multiplier" name="multiplier" class="form-
control"
        value="0.3082" />
        <button class="btn btn-success text-white border-dark" name="color"
type="button"
        onclick="update_chart(this)">Submit</button>
      </div>
    </div>
  </div>
</div>
<div class="card-body bg-light" id="forecast_chart">
  {% autoescape off %}
  {{ forecast_chart }}
  {% endautoescape %}
</div>
</div>
</div>
</div>
</div>
<script src="https://cdn.plot.ly/plotly-2.3.0.min.js"></script>
<script src="{% static 'js/update_chart.js' %}"></script>
</body>
{% endblock %}

```

Anhang K: update_chart.js

```
function update_chart() {
  // Get input field value
  var multiplicator = document.getElementById("multiplicator").value;

  // Get existing chart data and layout
  var chartData = document.getElementById("chart-div").getAttribute("data-chart-data");
  var chartLayout = document.getElementById("chart-div").getAttribute("data-chart-layout");

  // Parse chart data and layout as JSON strings
  var data = JSON.parse(chartData);
  var layout = JSON.parse(chartLayout);

  // Check if trace with same name already exists
  var traceExists = false;
  for (var i = 0; i < data.length; i++) {
    if (data[i].name === Number(multiplicator).toFixed(2) + ' m²') {
      traceExists = true;
      break;
    }
  }

  // If trace does not exist, create new one and add to data
  if (!traceExists) {
    var newTrace = {
      x: data[0].x,
      y: data[0].y.map(function (val) { return val * multiplicator * (1 / 0.3082); }),
      type: 'scatter',
      name: Number(multiplicator).toFixed(2) + ' m²',
      fill: 'tozeroy',
      marker: { color: 'dodgerblue' }
    };
    data.push(newTrace);
    // Sort traces by name in descending order
    data.sort(function (a, b) { return b.name.localeCompare(a.name); });
  }

  // Redraw chart
  Plotly.newPlot('chart-div', data, layout);
}
```