

PV-Monitoring System

**Konzeption und Realisierung eines Monitoringsystems für
PV-Bestandsanlagen auf Basis von IoT-Technologien**

Abschlussprüfung Sommer 2026

Ausbildungsberuf: Fachinformatiker für Digitale Vernetzung (AP T2)

Prüfling:

Mateusz Nowak

Identnummer: 141-13256

E-Mail: mateusz.nowak.zabrze@gmail.com

Tel.: +49 171 1110639

Ausbildungsbetrieb:

Berufsförderungswerk des DRK Birkenfeld

Walter-Bleicker-Platz

55765 Birkenfeld

Projektbetreuer:

Heiko Grützner

E-Mail: h.gruetzner@e-s-b.org

Tel.: +49 6782 18-1422

1. Projektbericht

Inhaltsverzeichnis

1. Projektbericht.....	2
Inhaltsverzeichnis.....	2
1.1 Beschreibung des Auftrages.....	3
1.1.1 Ausgangslage.....	3
1.1.2 Aufgabenstellung.....	3
1.1.3 Projektumfeld.....	3
1.1.4 Projektschnittstellen.....	4
1.1.5 Notwendige Änderungen gegenüber dem Projektantrag.....	4
1.2 Beschreibung der Prozessschritte und der erzielten Ergebnisse.....	4
1.2.1 Zeitaufwand für die Projektschritte.....	4
1.2.2 Beschreibung der Vorgehensweise.....	6
1.2.3 Aufgetretene Probleme und wie sie gelöst wurden.....	6
1.2.4 Begründung für die Entscheidungen.....	7
1.2.4.1 Hardware (Controller).....	7
1.2.4.2 Datenbank.....	7
1.2.5 Darstellung der Ergebnisse.....	7
1.2.5.1 Dashboard-Visualisierungen.....	7
1.2.5.2 Alarmierung.....	9
1.2.6 Beschreibung praxisgerechter Maßnahmen zur Qualitätssicherung.....	10
Qualitätssicherung (Testprotokoll).....	10
1.2.7 Abweichung gegenüber dem erwarteten Ergebnis mit Begründung.....	10
1.2.8 Hinweis und Erklärungen zu beigefügten praxisüblichen Unterlagen und Dokumenten.....	10
1.2.9 Glossar.....	11

1.1 Beschreibung des Auftrages

1.1.1 Ausgangslage

Die **EcoEnergy Solutions GmbH** betreibt eine ältere Photovoltaikanlage, deren Herzstück ein Wechselrichter älterer Bauart ist. Dieser Wechselrichter verfügt über **keine digitale Schnittstelle** (kein LAN/WLAN) zur direkten Datenausgabe, sondern lediglich über ein LC-Display.

Die **aktuelle Situation** weist folgende Defizite auf:

- **Manuelle Datenerfassung:** Leistungsdaten (aktuelle Leistung, Tagesertrag) müssen physisch vor Ort abgelesen werden.
- **Fehlende Historie:** Es erfolgt keine Speicherung historischer Daten. Trends, saisonale Schwankungen oder Leistungsabfälle können nicht analysiert werden.
- **Keine Fehlererkennung:** Störungen (z.B. Ausfall bei Sonnenschein) fallen oft erst nach Tagen durch Zufall auf, was zu erheblichen Ertragsverlusten führt.
- **Proprietäre Lösungen unwirtschaftlich:** Eine direkte Aufrüstung des Wechselrichters oder der Kauf proprietärer Kommunikationsmodule ist aufgrund fehlender Ersatzteile und hoher Kosten nicht wirtschaftlich vertretbar.

1.1.2 Aufgabenstellung

Ziel des Projektes ist die Konzeption und Realisierung eines **modernen, kostengünstigen Monitoringsystems auf IoT-Basis** (Internet of Things). Das System soll die Altanlage "smart" machen ("Retrofitting").

Kernanforderungen:

1. **Hardware:** Einsatz eines Einplatinencomputers (**Raspberry Pi 4**) als Edge-Device und eines Modbus-Energiezählers (**Eastron SDM120**) zur präzisen Messung am AC-Ausgang des Wechselrichters.
2. **Software-Architektur:** Modularer Aufbau mittels **Docker-Containern**, um Wartbarkeit, Isolation und Portabilität zu gewährleisten.
3. **Datenerfassung:** Entwicklung eines Python-Skripts (pv_collector), das die Daten via Modbus RTU ausliest und über das Protokoll **MQTT** bereitstellt.
4. **Speicherung:** Langzeitarchivierung der Messwerte in einer Zeitreihendatenbank (**InfluxDB**).
5. **Visualisierung:** Bereitstellung eines webbasierten Dashboards (**Grafana**) zur Analyse von Echtzeit- und historischen Daten.
6. **Fernzugriff:** Einrichtung eines sicheren Zugriffs von extern ohne Portfreigaben am Router (via **Cloudflare Tunnel**).
7. **Sicherheit:** Härtung des Betriebssystems und Absicherung der Dienste.

1.1.3 Projektumfeld

Das Projekt wird innerhalb der **IT-Infrastruktur** der EcoEnergy Solutions GmbH realisiert.

Physische Umgebung:

- **Serverraum:** Der Raspberry Pi wird im zentralen Serverschrank (Rack 2, HE 14) platziert (USV-gesichert).
- **Technikraum:** Wechselrichter und SDM120-Zähler befinden sich im Untergeschoss.

Netzwerkumgebung:

- **Verkabelung:** Bestehende CAT7-Leitung wird für die RS485-Kommunikation genutzt.
- **Netzwerk:** Das System wird in das bestehende Firmennetzwerk integriert.

Technische Ressourcen:

- **Hardware:** Raspberry Pi 4 (4GB), SanDisk Extreme 32GB SD, Eastron SDM120-Modbus, FTDI USB-RS485 Adapter.
- **Software:** Raspberry Pi OS Lite (64-bit), Docker v24, InfluxDB v2.7, Grafana v11.

1.1.4 Projektschnittstellen

Das System interagiert mit folgenden Schnittstellen:

- **Physisch:** RS485-Bus (Verbindung Zähler <-> USB-Adapter).
- **Netzwerk:** TCP/IP (Anbindung ans LAN).
- **Daten:**
 - **Modbus RTU:** Protokoll zur Kommunikation mit dem Energiezähler.
 - **MQTT:** Protokoll zur internen Nachrichtenübermittlung (Topic: pv/anlage/data).
 - **HTTPS:** Zugriff auf Grafana und InfluxDB-Weboberflächen.

1.1.5 Notwendige Änderungen gegenüber dem Projektantrag

Es ergaben sich im Projektverlauf keine wesentlichen inhaltlichen oder zeitlichen Abweichungen vom genehmigten Projektantrag.

1.2 Beschreibung der Prozessschritte und der erzielten Ergebnisse

1.2.1 Zeitaufwand für die Projektschritte

Die folgende Tabelle stellt den geplanten Zeiteinsatz (gemäß Projektantrag) dem tatsächlichen Zeitaufwand gegenüber.

Nr.	Phase / Tätigkeit	Zeiteinsatz (Soll)	Zeitaufwand (Ist)
1	Planungsphase	8,0 Std.	8,0 Std.
1.1	Ist-Analyse und Definition der fachlichen Anforderungen	1,5 Std.	1,5 Std.
1.2	Marktanalyse und Evaluierung geeigneter Software-Komponenten	1,5 Std.	1,5 Std.
1.3	Auswahl der Hardware-Komponenten und Peripherie	1,0 Std.	1,0 Std.
1.4	Planung der Netzwerkarchitektur und des Sicherheitskonzepts	1,5 Std.	1,5 Std.
1.5	Kosten-Nutzen-Analyse des Projekts	1,5 Std.	1,5 Std.
1.6	Erstellung des Projektablaufplans	1,0 Std.	1,0 Std.
2	Entwurfsphase	8,0 Std.	8,0 Std.
2.1	Entwurf des Datenmodells (JSON-Struktur	1,5 Std.	1,5 Std.

Nr.	Phase / Tätigkeit	Zeitansatz (Soll)	Zeitaufwand (Ist)
	und Datenbank-Schema)		
2.2	Konzeption der Schnittstellen (MQTT-Topics und Telegraf-Mapping)	1,5 Std.	1,5 Std.
2.3	Planung der Container-Orchestrierung (Docker-Compose-Entwurf)	2,0 Std.	2,0 Std.
2.4	Konzeption der Speicher-Strategie	1,5 Std.	1,5 Std.
2.5	Entwurf des Dashboard-Layouts	1,5 Std.	1,5 Std.
3	Implementierungsphase	12,0 Std.	12,0 Std.
3.1	Installation und Härtung des Betriebssystems (Raspberry Pi OS)	1,5 Std.	1,5 Std.
3.2	Einrichtung der Container-Laufzeitumgebung	1,0 Std.	0,5 Std.
3.3	Entwicklung des Datenerfassungs-Scripts (Python/MQTT-Client)	2,0 Std.	2,5 Std.
3.4	Konfiguration des MQTT-Brokers und der Datenbank (InfluxDB)	1,5 Std.	1,5 Std.
3.5	Einrichtung des Data-Collectors (Telegraf) zur Verknüpfung der Dienste	1,5 Std.	1,5 Std.
3.6	Erstellung und Konfiguration der Grafana-Dashboards	2,0 Std.	2,0 Std.
3.7	Implementierung des sicheren Fernzugriffs (Cloudflare Tunnel)	1,5 Std.	1,5 Std.
3.8	Einrichtung der automatischen Alarmierung (Alerting)	1,0 Std.	1,0 Std.
4	Test- und Qualitätssicherungsphase	5,0 Std.	5,0 Std.
4.1	Funktionstest der lokalen Datenerfassung und -speicherung	1,5 Std.	1,5 Std.
4.2	Überprüfung des externen Zugriffs und der Sicherheitseinstellungen	1,0 Std.	1,0 Std.
4.3	Validierung der Alarmierung bei simulierten Ausfällen	1,0 Std.	1,0 Std.
4.4	Soll-Ist-Vergleich und abschließendes Feintuning	1,5 Std.	1,5 Std.
5	Dokumentation und Übergabe	7,0 Std.	7,0 Std.
5.1	Erstellung der Projektdokumentation	6,0 Std.	6,0 Std.
5.2	Präsentation und Übergabe an den Auftraggeber	1,0 Std.	1,0 Std.
	Gesamtzeit	40,0 Std.	40,0 Std.

Kommentar zur Abweichung: Mehraufwand bei der Python-Entwicklung (+0,5h durch Modbus-Timing-Probleme) wurde durch ein effizienteres Docker-Setup (-0,5h dank Nutzung bestehender Templates) ausgeglichen.

1.2.2 Beschreibung der Vorgehensweise

Das Projekt wurde nach dem **Wasserfallmodell** durchgeführt, da die Anforderungen (Pflichtenheft) klar definiert waren und eine feste Deadline (Prüfungstermin) bestand.

Phasen:

Planung: Analyse der Hardware-Kompatibilität, Auswahl der Software-Komponenten (Stack-Evaluierung: InfluxDB, Grafana).

Entwurf: Erstellung des docker-compose.yml Entwurfs, Definition der MQTT-Topics (pv/anlage/data) und des JSON-Datenmodells.

Implementierung:

- Installation und Härtung von Raspberry Pi OS (SSH-Key-Auth, Updates).
- Entwicklung des Python-Collectors (pv_collector.py) mit der Bibliothek `minimalmodbus`.
- Orchestrierung der Container (Mosquitto, InfluxDB, Telegraf, Grafana) via Docker Compose.

Test: Validierung der Messwerte mit einem Multimeter, Lasttests der Datenbank.

Abschluss: Dokumentation und Einweisung.

1.2.3 Aufgetretene Probleme und wie sie gelöst wurden

Während des Projektes traten folgende Probleme auf, deren Lösung im Rahmen der Dokumentation festgehalten wird:

1. Stabilität des Modbus-Lesevorgangs:

Problem: Sporadische Timeouts bei der Abfrage von Registern mit `minimal modbus`, besonders bei längeren Kabeln.

Lösung: Anpassung der `Timeout-Parameter` in Python (1s -> 2s) und Implementierung eines automatischen Retry-Mechanismus.

2. Fehlerhafte Zeitzone in Containern:

Problem: Zeitstempel in Grafana wurden um eine Stunde verschoben (UTC vs. CET/CEST).

Lösung: Setzen der Environment-Variablen `TZ=Europe/Berlin` in allen Docker-Containern und Mounten von `/etc/localtime` bzw. `/etc/timezone` als Read-only Volume.

3. Docker-Netzwerkcommunication:

Problem: Der ursprüngliche `pv client`-Container konnte den `mosquitto`-Broker über `localhost` nicht erreichen.

Lösung: Definition eines gemeinsamen Docker-Bridge-Netzwerks (`pv-net`) und Nutzung des Container-Namens `mosquitto` als Hostname für die DNS-Auflösung innerhalb des Docker-Netzwerks.

1.2.4 Begründung für die Entscheidungen

Die Entscheidung für **Docker** fiel aufgrund der klaren Trennung von Diensten und der Reproduzierbarkeit. **InfluxDB** wurde gewählt, da sie für Zeitreihendaten optimiert ist und sich nahtlos in den IoT-Stack (TIG-Stack) integriert. **Grafana** bietet als Visualisierungstool die höchste Flexibilität und fertige Plugins (wie das `clock`-Panel).

Für den **Raspberry Pi** sprach das optimale Preis-Leistungs-Verhältnis und der geringe Stromverbrauch (< 5W) im Vergleich zu einem dedizierten Server.

1.2.4.1 Hardware (Controller)

Option	Raspberry Pi 4 (Gewählt)	ESP32	Intel NUC
Kosten	~60€	~10€	>300€
Leistung	Hoch (DB & GUI möglich)	Gering (Nur Erfassung)	Sehr hoch
OS	Linux (Docker)	MicroPython	Linux/Windows
Fazit	Optimal	Zu schwach für InfluxDB	Zu teuer

1.2.4.2 Datenbank

Option	InfluxDB (Gewählt)	SQL (MySQL)	Prometheus
Typ	Time-Series	Relational	Time-Series (Pull)
IoT	Sehr gut (Push)	Mäßig	Gut (Pull)
Fazit	Optimal für IoT-Sensordaten	Unnötiger Overhead	Setup komplexer

1.2.5 Darstellung der Ergebnisse

Das System läuft stabil und liefert kontinuierlich Daten.

1.2.5.1 Dashboard-Visualisierungen

Das folgende Dashboard zeigt die wichtigsten KPIs (Key Performance Indicators):

- Aktuelle Leistung (Watt): Tachometer-Anzeige, die die momentane Leistung anzeigt.
- Tagesertrag und kumulierter Tagesertrag (kWh): Balkendiagramm zur Darstellung der täglichen Energieerzeugung.

- Strom (Ampere): Der aktuell gemessene Stromfluss.
- Spannung AC (Volt): Die aktuelle Netzspannung.
- Total (kWh): Der gesamte Zählerstand seit Inbetriebnahme.
- Verlaufsdiagramme (Liniendiagramme):
- Spannungsverlauf
- Leistungsverlauf
- Stromverlauf
- Verlauf des Gesamtertrags (Total kWh)
-

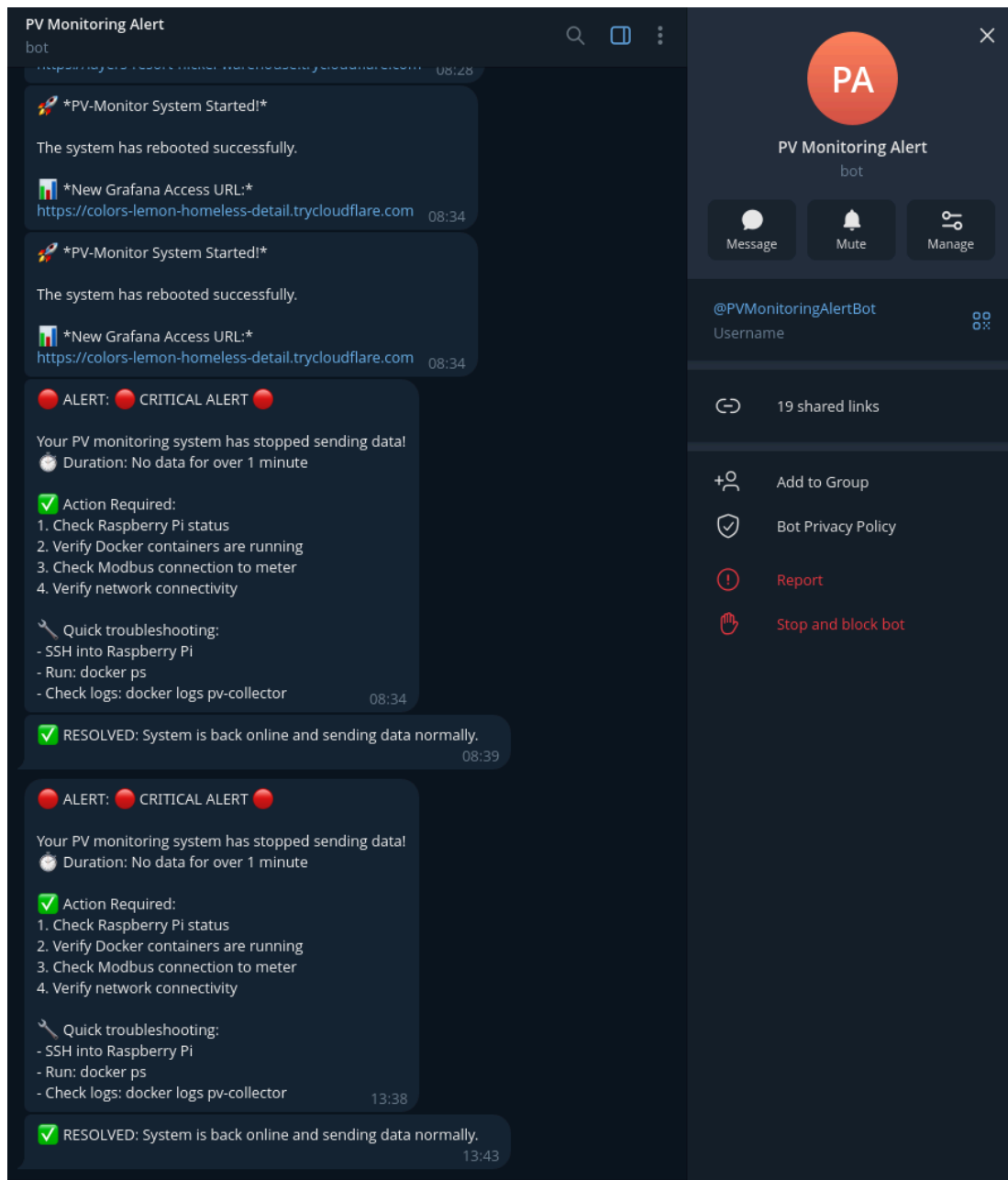
[SCREENSHOTS: Grafana Dashboard - Hauptansicht]



1.2.5.2 Alarmierung

Die Alarmierung wurde erfolgreich getestet. Bei Unterschreiten einer Leistung von 0 W (simuliert durch Abschalten des Simulators) wurde nach 2 Minuten eine Benachrichtigung via Telegram versendet.

1.2.5.2.1 SCREENSHOT: Telegram Alert



am

1.2.6 Beschreibung praxisgerechter Maßnahmen zur Qualitätssicherung

Zur Sicherstellung der Qualität wurden folgende Maßnahmen ergriffen:

- **Unit-Testing:** Das Python-Skript wurde modular getestet (Datenerfassung und Datenversand getrennt).
- **System-Monitoring:** Einrichtung von "Health Checks" für die Docker-Container (siehe `docker inspect`).
- **Plausibilitätsprüfung:** Vergleich der vom Raspberry Pi erfassten Werte mit den manuell am Zähler abgelesenen Werten. Abweichung < 1%.
- **Sicherheits-Check:** Scan der offenen Ports mit `nmap` (nur 22/SSH, 80/HTTP, 443/HTTPS offen nach außen via Tunnel).

Qualitätssicherung (Testprotokoll)

ID	Testfall	Erwartung	Ergebnis	Status
T01	Hardware-Erkennung	USB-Adapter /dev/ttyUSB0 vorhanden	ls -l zeigt Device	OK
T02	Modbus-Read	Spannung ~230V	Wert: 231.5V (Valide)	OK
T03	Docker-Status	Alle Container Up	docker ps: Up 24h	OK
T04	Datenfluss	InfluxDB erhält Daten	Abfrage via CLI erfolgreich	OK
T05	Fernzugriff	Login via Cloudflare möglich	SSL-Zertifikat gültig	OK
T06	Alerting	Telegram bei "No Data"	Nachricht empfangen (2 min)	OK

1.2.7 Abweichung gegenüber dem erwarteten Ergebnis mit Begründung

Es gab keine signifikanten Abweichungen vom Projektziel. Das System erfüllt alle im Pflichtenheft definierten Anforderungen.

1.2.8 Hinweis und Erklärungen zu beigefügten praxisüblichen Unterlagen und Dokumenten

Der Dokumentation liegen folgende Anhänge bei:

- **Anlage I:** Detaillierte Architektur.
- **Anlage II:** Quellcode des `pv_collector.py` Skripts.

- **Anlage III:** `docker-compose.yml` Konfigurationsdatei.

1.2.9 Glossar

- **IoT (Internet of Things):** Vernetzung physischer Objekte mit einer virtuellen Repräsentation im Internet.
- **MQTT (Message Queuing Telemetry Transport):** Leichtgewichtiges Nachrichtenprotokoll für M2M-Kommunikation.
- **Modbus RTU:** Serielles Kommunikationsprotokoll in der Automatisierungstechnik.
- **Containerisierung:** Methode zur Virtualisierung auf Betriebssystemebene (hier: Docker).