OST
Ostschweizer
Fachhochschule

Master Thesis
Master of Advanced Studies in Software Engineering

Out-of-the-Box Visualisierungssystem

für Zähler- und Anlagenüberwachung mit M-Bus- und Modbus-Anbindung auf Raspberry Pi Basis

Betreuer
Dr. Thomas Memmel

Autoren
Kerim San, Remzi Atesci

1 Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

2 Abstract

Diese Masterarbeit untersucht, wie ein kostengünstiges und zugleich leistungsfähiges Visualisierungssystem für Zähler- und Anlagenüberwachung realisiert werden kann. Ziel war es, ein lokal betreibbares System zu entwickeln, das sowohl M-Bus- als auch Modbus-Geräte unterstützt, ohne dass sich die Protokolle gegenseitig stören. Das Projekt wurde agil umgesetzt, gestützt durch den Einsatz von Atlassian-Tools zur Projektorganisation. Erste Prototypen wurden in realer Umgebung getestet: Verbrauchsdaten von M-Bus-Zählern konnten erfolgreich ausgelesen, visualisiert und als CSV exportiert werden; Modbus-Geräte wurden über Register konfiguriert, digitale Eingänge erfasst und Ausgänge über ein Webinterface gesteuert. Die Visualisierung erfolgt über ein React-basiertes Frontend, das eine nutzerfreundliche Darstellung der Daten gewährleistet. Die Ergebnisse zeigen, dass sich Verbrauchsdaten, Fehlermeldungen und Steuerbefehle zuverlässig visualisieren und unterscheiden lassen – mit Mehrwerten wie Benutzerrollen-Trennung und universeller Einsetzbarkeit durch Unterstützung von Modbus RTU, Modbus TCP und M-Bus. Damit leistet die Arbeit einen Beitrag zur IoT-basierten Gebäudeautomation und zum Facility Management.

3 Management Summary

3.1 Ausgangslage

Der Betrieb von Gebäuden und technischen Anlagen steht zunehmend im Spannungsfeld zwischen Energieeffizienz, Betriebssicherheit und Kostenoptimierung. Bereits heute existieren zahlreiche digitale Lösungen zur Datenerfassung und Visualisierung, insbesondere im Bereich der Gebäudeautomation und des Smart Metering. Viele dieser Systeme richten sich jedoch an grössere Liegenschaften oder industrielle Umgebungen und sind daher für kleinere Anwendungen überdimensioniert. Sie zeichnen sich durch hohe Investitionskosten, komplexe Inbetriebnahme und proprietäre Schnittstellen aus.
Für private Eigentümer, kleine Gewerbebauten oder Werkstätten fehlen damit erschwingliche Systeme, die Verbrauchs- und Zustandsdaten verschiedener Geräte lokal erfassen und zentral darstellen können. Hinzu kommt, dass viele Hersteller eigene Softwarelösungen anbieten, welche nur mit den jeweiligen Geräten kompatibel sind und so Datensilos erzeugen. Diese Fragmentierung erschwert eine ganzheitliche Sicht auf Energie- und Anlagendaten.
Vor diesem Hintergrund entstand die Idee, ein kostengünstiges, lokal betreibbares System zu entwickeln, das standardisierte Protokolle wie **M-Bus** und **Modbus** unterstützt und unabhängig von spezifischen Herstellern genutzt werden kann. Die Masterarbeit knüpft damit direkt an die wachsende Bedeutung von IoT-Ansätzen im Facility Management und an den Bedarf nach flexiblen, nachrüstbaren Lösungen für kleinere Objekte an.

3.2 Relevanz des Themas

Die Praxis zeigt, dass Ausfälle technischer Anlagen häufig zu hohen Folgekosten führen. Wird etwa eine Wärmepumpe oder ein Heizsystem erst beim vollständigen Stillstand bemerkt, entstehen Verzögerungen durch Notfalleinsätze, zusätzliche Servicefahrten und ungeplante Stillstandszeiten. Frühzeitige Informationen über Fehlermeldungen oder kritische Betriebszustände ermöglichen dagegen eine gezielte Vorbereitung der Technikerinnen und Techniker, wodurch Reparaturen schneller und kostengünstiger durchgeführt werden können.
Darüber hinaus rückt die **Transparenz von Energieverbräuchen** zunehmend in den Fokus. Sowohl private Hausbesitzer als auch professionelle Verwalter benötigen nachvollziehbare Daten, um Energieeffizienz-Massnahmen einzuleiten, Abrechnungen zu plausibilisieren und regulatorische Anforderungen zu erfüllen. Für Energieberater bietet die systematische Erfassung und Auswertung von Verbrauchsdaten zudem eine Grundlage, um Optimierungspotenziale aufzuzeigen.
Die Relevanz des Themas erstreckt sich somit über mehrere Ebenen:

- **Ökonomisch**: Reduktion von Servicekosten und Energieausgaben.
- **Ökologisch**: Beitrag zu effizientem Ressourceneinsatz und Nachhaltigkeit.
- **Technisch**: Vereinheitlichung heterogener Systeme und Verbesserung der Interoperabilität.

- **Organisatorisch**: Unterstützung unterschiedlicher Zielgruppen von Endanwendern bis zu Fachtechnikern.

Damit adressiert die Arbeit sowohl die Bedürfnisse kleinerer Betreiber als auch die generellen Herausforderungen der modernen Gebäudeautomation.

3.3 Einsatzumfeld und Ziel

Das entwickelte System ist flexibel einsetzbar und adressiert verschiedene Anwendungsszenarien:

- **Privates Wohnumfeld**: Hausbesitzer können ihren Energieverbrauch (Öl, Wasser, Strom, Wärme) transparent erfassen und kritische Störungen frühzeitig erkennen.
- **Mehrfamilienhäuser**: Verwalterinnen und Verwalter erhalten eine Grundlage für verursachergerechte Abrechnungen und können den Betrieb zentral überwachen.
- **Gewerbebauten und Werkstätten**: Betreiber technischer Anlagen (z. B. Heizungs- oder Pumpensysteme) profitieren von einer besseren Kontrolle und einer einfachen Fehlerdiagnose.

Das Ziel der Masterarbeit bestand darin, ein **prototypisches Visualisierungssystem** zu entwickeln, das folgende Anforderungen erfüllt:

- **Herstellerunabhängigkeit**: Unterstützung unterschiedlicher Geräte über standardisierte Schnittstellen (M-Bus, Modbus).
- **Lokaler Betrieb**: Keine Abhängigkeit von Cloud-Diensten oder Lizenzmodellen, wodurch Kosten reduziert und Datenschutzrisiken minimiert werden.
- **Benutzerfreundlichkeit**: Ein webbasiertes Frontend mit klarer Trennung zwischen **Endanwendern** (einfache Visualisierung) und **Technikern** (Detailkonfiguration und Diagnose).
- **Flexibilität**: Möglichkeit zur Konfiguration über Dateien oder direkte Eingaben in der Weboberfläche, sodass das System einfach auf neue Geräte angepasst werden kann.
- **Erweiterbarkeit**: Vorbereitung für zusätzliche Protokolle, Datenexporte und Benachrichtigungsfunktionen.

Damit verbindet das Projekt praxisnahe Ziele – wie die Überwachung einer Heizungsanlage – mit dem Anspruch, eine generische und erweiterbare Architektur zu schaffen, die auch in anderen Kontexten des **IoT und Facility Managements** genutzt werden kann.

3.4 Zentrale Fragestellung

Aus der Ausgangslage und den Projektzielen leiten sich die folgenden zentralen Forschungsfragen ab:

- **Realisierbarkeit auf Standard-Hardware**
  Lässt sich ein Visualisierungssystem mit kostengünstiger, handelsüblicher Hardware (z. B. Raspberry Pi) zuverlässig betreiben?
- **Parallelbetrieb von M-Bus und Modbus**
  Wie können die beiden Protokolle gleichzeitig genutzt werden, ohne dass es zu Kommunikationsstörungen oder Ressourcenkonflikten kommt?

- **Architektur für unterschiedliche Nutzergruppen**
  Wie muss die Software- und Systemarchitektur gestaltet sein, damit sowohl Endanwender (einfache Visualisierung) als auch Fachtechniker (Detaildiagnose und Konfiguration) effizient unterstützt werden?
- **Konfigurierbarkeit und Benutzerfreundlichkeit**
  Ist eine Konfiguration über Dateien (z. B. Excel/JSON) und eine visuelle Auswahl von Datenpunkten im Web-Frontend praktikabel und robust umsetzbar?

Diese Fragen bilden die Leitplanken der Masterarbeit. Sie strukturieren sowohl die technische Umsetzung als auch die spätere Evaluation der entwickelten Lösung.

3.5 Vorgehen

Die Umsetzung der Masterarbeit folgte einem **agilen Vorgehensmodell**, das kurze Entwicklungszyklen und regelmässige Feedbackschleifen ermöglichte. Zentrale Elemente waren:

- **Projektmanagement und Organisation**
  Die Arbeitspakete wurden in Sprints organisiert und über Tools wie Atlassian Jira und Bitbucket strukturiert. Pull-Request-Reviews und Issue-Tracking stellten eine kontinuierliche Qualitätssicherung sicher.
- **Walking Skeleton / Architekturprototyp**
  Bereits in einer frühen Phase wurde ein End-to-End-Prototyp entwickelt, der einen minimalen Durchstich über alle Systemebenen (Hardware, Treiber, Backend, Frontend) realisierte. Dadurch konnten Risiken wie Kommunikationsabbrüche oder Datenbankintegration früh identifiziert und adressiert werden.
- **Technologische Trennung**
  Das System wurde bewusst in **Backend** (Python/Flask, Treiber, API, Persistenz) und **Frontend** (React, Visualisierung, Rollenmodell) getrennt. Diese Aufteilung erleichterte die Erweiterbarkeit und die parallele Arbeit im Team.
- **Iterative Erweiterung**
  Funktionen wie M-Bus-Scanning, Modbus-Registerhandling, Quittierung von Fehlermeldungen und zyklische Geräteüberwachung wurden schrittweise implementiert und getestet.
- **Praxisnahe Verifikation**
  Prototypen wurden nicht nur im Labor, sondern auch in einer realen Werkstattumgebung installiert. So konnten reale Verbrauchsdaten erhoben und typische Betriebsfehler simuliert werden.
- **Qualitätssicherung**
  Unit- und Integrationstests mit pytest sowie ein Coverage-Gate von mindestens 80 % stellten die funktionale Robustheit sicher. Ergänzt wurde dies durch statische Analysen (Linting) und manuelle Explorations-Tests im Frontend.

Dieses Vorgehen erlaubte es, trotz begrenztem Zeitrahmen ein funktionsfähiges und stabiles System zu entwickeln, das die zentralen Forschungsfragen adressiert und als Grundlage für eine weiterführende Entwicklung dient.

3.6 Erreichte Ziele und Erkenntnisse

Im Verlauf der Arbeit konnten die definierten Soll-Ziele weitgehend umgesetzt und durch zusätzliche Funktionen ergänzt werden. Die wichtigsten Ergebnisse lassen sich wie folgt zusammenfassen:

- **M-Bus**
  - Automatische Erkennung von Geräten im Strang und Speicherung der Adressen.
  - Selektive Auslesung und Visualisierung von Datenpunkten über das Web-Frontend.
  - Implementierter Abbruchmechanismus, um laufende Scans kontrolliert zu beenden und Kommunikationskollisionen zu vermeiden.
- **Modbus**
  - Erfolgreiche Integration von Geräten via RTU (RS485) und TCP.
  - Konfiguration von Registern über das Frontend sowie Auslesen digitaler Eingänge und Ansteuerung von Ausgängen.
  - Erweiterbarkeit für weitere Registertypen und Geräte ist gewährleistet.
- **Frontend (React)**
  - Rollenbasierte Benutzeroberfläche mit klarer Trennung zwischen Endanwendern (Visualisierung) und Technikern (Konfiguration und Diagnose).
  - Umsetzung zentraler Funktionen wie Gerätescan, Detailseiten, Monitoring und Charting.
- **Backend (Flask)**
  - Bereitstellung einer modularen REST-API mit konsistentem JSON-Envelope.
  - Zyklische Überwachung konfigurierter Geräte und Speicherung von Zeitreihendaten in PostgreSQL.
  - Stabile Integration der CLI-Tools libmbus und mbpoll.
- **Betrieb und Sicherheit**
  - VPN-Anbindung über Tailscale für sicheren Fernzugriff.
  - Schutzmechanismen gegen Brute-Force-Angriffe sowie automatische Systemupdates (unattended upgrades).
- **Qualitätssicherung**
  - Testabdeckung im Backend ≥ 80 % (CI-Gate).
  - Praxisnachweis durch Installation in einer Werkstattumgebung mit realen Zählern und Modulen.

Erkenntnisse:

Die Arbeit bestätigt, dass ein kostengünstiges, lokal betreibbares Visualisierungssystem technisch realisierbar ist. Durch die modulare Architektur konnte sowohl die einfache Nutzung durch Endanwender als auch die Detailarbeit von Fachtechnikern berücksichtigt werden. Entscheidend für den Erfolg waren die frühzeitige Umsetzung eines Walking Skeletons und die konsequente Trennung von Backend und Frontend.

3.7 Fazit

Mit der vorliegenden Arbeit konnte gezeigt werden, dass ein **herstellerunabhängiges, lokal betreibbares Visualisierungssystem** auf Basis kostengünstiger Standard-Hardware erfolgreich realisiert werden kann. Die Umsetzung vereint wesentliche Anforderungen aus der Praxis: zuverlässiges Auslesen von Zähler- und Anlagendaten, flexible Konfiguration, klare Rollentrennung zwischen Endanwendern und Technikern sowie eine robuste Systemarchitektur.

Die Prototypen haben im Praxiseinsatz bestätigt, dass sowohl **M-Bus**- als auch **Modbus-Geräte** parallel genutzt werden können, ohne dass es zu Kommunikationsstörungen kommt. Ergänzend liefern zusätzliche Features – wie VPN-Fernzugriff, automatisierte Benachrichtigungen und grundlegende Sicherheitsmechanismen – einen Mehrwert für den praktischen Einsatz.

Damit bildet die Arbeit nicht nur einen funktionalen Prototyp ab, sondern schafft auch eine **Basis für die Weiterentwicklung** in Richtung produktiver Lösungen im Bereich Smart Metering, IoT und Facility Management.

4 Einleitung

4.1 Ausgangslage und Motivation

Die zunehmende **Digitalisierung im Gebäudebereich** führt dazu, dass Energieeffizienz, Betriebsüberwachung und Kostenkontrolle immer stärker in den Fokus rücken. Studien des Fachverbands Gebäudeautomation Schweiz zeigen, dass vernetzte Systeme für Smart Metering und Facility Management wesentliche Beiträge zur Optimierung von Betriebskosten und Ressourcennutzung leisten (Gebäudeautomation, 2023).

Gleichzeitig sind etablierte Systeme zur Gebäudeautomation häufig mit hohen Investitionskosten verbunden, komplex in der Bedienung und oft proprietär. Hersteller wie Härz AG bieten umfassende Visualisierungssysteme an, die sich primär an grössere Anlagen richten und damit für kleinere Objekte überdimensioniert wirken (Härz AG, 2025). Ähnlich verhält es sich mit industriellen Lösungen wie den HMI-Systemen von Beckhoff, die zwar leistungsfähig und flexibel sind, jedoch umfangreiche Fachkenntnisse voraussetzen und in kleinen Projekten selten wirtschaftlich sind (Beckhoff, 2025).

Vor allem kleinere Liegenschaften, Werkstätten oder private Eigentümer haben selten Zugang zu IT-Spezialisten oder die finanziellen Mittel für vollumfängliche Gebäudeleitsysteme. Gleichzeitig steigt der Bedarf an **einfach installierbaren und lokal betreibbaren Lösungen**, die grundlegende Anforderungen wie Visualisierung von Zählerständen, Monitoring von Betriebszuständen und frühzeitige Fehlerdiagnosen erfüllen.

Die Motivation für diese Masterarbeit entstand daher aus einem **praktischen Anwendungsfall**: Ein Bauherr benötigte eine kostengünstige und leicht handhabbare Lösung zur Visualisierung von Verbrauchsdaten und Anlagestatus.

Kriterium	Etablierte Systeme (z. B. Hersteller-HMI, proprietäre GA-Visualisierung)	Zielsystem dieser Arbeit (Raspberry Pi · M-Bus/Modbus · lokal)
Anschaffungskosten	Hoch (Lizenz, Hardware, Integrationspauschalen)	Niedrig (Standard-Hardware, Open-Source-Stack)
Laufende Kosten	Wiederkehrende Lizenzen/Subscriptions möglich	Minimal (lokaler Betrieb, keine Cloud-Pflicht)
Komplexität/Bedienung	Für grössere Anlagen optimiert, oft komplex	Schlanke UI, auf Kernaufgaben fokussiert
Herstellerbindung	Stark (proprietäre Tools, Formate)	Gering (offene Protokolle, eigene API)

Interoperabilität	Eingeschränkt ausserhalb des Ökosystems	Hoch durch M-Bus/Modbus und REST-API
Zielgruppe	Industrie, grosse Liegenschaften	Private, MFH, Werkstätten, KMU
Deployment	Projekt-/Integratoren-getrieben	Selbst installierbar, Skripte/Anleitung
Einrichtungsaufwand	Wochen bis Monate	Stunden bis wenige Tage
Datenhaltung	Teils cloudbasiert, teils proprietär	Lokal in PostgreSQL, exportfähig
Datensouveränität	Variiert, oft eingeschränkt	Vollständig beim Betreiber (on-prem)
Sicherheit	Hoch, aber komplex (PKI, Härtung)	VPN (Tailscale), Updates, Basis-Hardening
Erweiterbarkeit	Module gegen Aufpreis	Modularer Code, JSON/ENV-Konfig, neue Treiber möglich
Wartung	Herstellerabhängig	Eigenständig, Dokumentation + CI-Pipelines
Risiko „Lock-in“	Erhöht	Niedrig

Tabelle 1 Vergleich etablierter Systeme mit dem entwickelten Zielsystem

4.2 Problemstellung

Im Bereich der Gebäudeautomation existiert eine Vielzahl von Lösungen zur **Erfassung und Visualisierung von Energie- und Anlagendaten**. Diese Systeme richten sich jedoch überwiegend an grosse kommerzielle Installationen. Typische Probleme sind:

- **Überdimensionierung für kleine Objekte**: Lösungen sind teuer und komplex, obwohl nur wenige Zähler oder eine einzelne Anlage überwacht werden sollen.
- **Proprietäre Abhängigkeiten**: Hersteller liefern ihre Systeme oft mit geschlossenen Softwareumgebungen, die nur mit eigenen Geräten kompatibel sind (Härz AG, 2025).
- **Erschwerte Integration**: Unterschiedliche Zähler- und Steuerungssysteme lassen sich nur mit erheblichem Aufwand in einem gemeinsamen System zusammenführen (Beckhoff, 2025).
- **Fehlende kostengünstige Alternativen**: Für Privathaushalte und KMU fehlen erschwingliche Systeme, die lokal betrieben werden können und ohne Fachwissen installierbar sind (Gebäudeautomation, 2023).

Die zentrale **Problemstellung** dieser Masterarbeit lautet daher:
Es fehlt eine **einheitliche Plattform**, die heterogene Geräte über **M-Bus und Modbus** anbindet, deren Daten lokal speichert und benutzerfreundlich visualisiert – ohne Lizenzabhängigkeiten, Herstellerbindungen oder komplexe Einrichtung.

4.3 3.3 Zielsetzung

Das Ziel dieser Arbeit ist ein funktionsfähiger Prototyp eines **herstellerunabhängigen, lokal betreibbaren Visualisierungssystems** für M Bus und Modbus. Der Prototyp soll Daten aus realen Zählern und Anlagen zuverlässig erfassen, in einer lokalen Datenbank speichern und im Webfrontend verständlich darstellen. Zusätzlich soll die Lösung funktionssicher und erweiterbar sein, damit spätere Protokolle, weitere Geräteklassen und Benachrichtigungen ohne Architekturbruch ergänzt werden können. Zentral ist die **Trennung der Perspektiven**: Endanwender erhalten eine einfache Ansicht mit Verbrauch und Zustand, Techniker bekommen präzise Konfiguration und Diagnose.

Zielkriterien

- **Realisierbarkeit** auf kostengünstiger Standardhardware.
- **Parallelbetrieb** von M Bus und Modbus ohne Kollisionen.
- **Benutzerfreundlichkeit** mit klaren Rollen.
- **Persistenz** und Auswertung von Zeitreihen lokal.
- **Erweiterbarkeit** über strukturierte Treiber und klare API.

Zielbereich	Sollziel	Umsetzung	Abweichung oder Kommentar
Protokolle	M Bus Zähler und Modbus Wärmepumpe anbinden	M Bus Zähler angebunden; Modbus über I O Modul RTU und TCP	Wärmepumpe ersetzt durch I O Modul zur Reduktion der Komplexität im Prototyp
Konfiguration	Parameter über Excel einlesen	Direkte Konfiguration in der Weboberfläche	Web UI ersetzt Excel Import für bessere Bedienung und weniger Fehler
Visualisierung	Verbrauch, Betriebsdaten, Fehlermeldungen im Web	React Frontend mit Scan, Details, Monitoring, Charts	Rollenbasierte UI für Endnutzer und Techniker
Quittierung	Fehlermeldungen quittieren	Quittierung umgesetzt	Nachweis über UI Szenarien
Persistenz	Lokale Datenbank	PostgreSQL mit Zeitreihen Abfragen	Export vorbereitet
Betrieb	Lokal, ohne Cloud Lizenz	Lokal mit VPN Zugang via Tailscale	Unattended upgrades und Basishärtung ergänzt
Qualität	Automatisierte Tests und Doku	Backend Tests mit Coverage Gate, Doku erstellt	Frontend Nachweise via Screencasts

Tabelle 2 Sollziele, Umsetzung und Abweichungen

4.4 Abgrenzung der Arbeit

Um den Umfang der Masterarbeit realistisch zu halten, wurden bestimmte Themenbereiche bewusst ausgeklammert oder nur am Rande berücksichtigt. Der Fokus lag auf der Entwicklung eines funktionsfähigen Prototyps, der die Kernanforderungen erfüllt und eine Basis für spätere Erweiterungen schafft.

Bereich	In Scope (Teil der Arbeit)	Out of Scope (bewusst ausgeschlossen)
Hardware	Raspberry Pi 4 als zentrale Plattform, M-Bus-Pegelwandler, RS485-Adapter, Modbus-TCP	Skalierung auf industrielle Server, Virtualisierung, Cloud-Deployment
Protokolle	M-Bus (libmbus), Modbus RTU/TCP (mbpoll)	Weitere Protokolle (z. B. OPC UA, SML, KNX)
Konfiguration	JSON- und WebUI-basierte Konfiguration von Geräten und Registern	Excel-Upload (ursprünglich geplant, später ersetzt)
Visualisierung	Web-Frontend (React) mit Tabellen, Charts, Rollenmodell	Native Apps (iOS/Android), komplexe Dashboards (z. B. Grafana)
Persistenz	PostgreSQL-Datenbank für Zeitreihen	Langzeitanalysen, BI-Tools, Data Warehousing
Sicherheit	VPN (Tailscale), Brute-Force-Schutz, unattended-upgrades	Vollständige Security-Härtung (mTLS, Reverse Proxy, IDS/IPS)
Tests	Unit- und Integrationstests im Backend, CI-Pipeline mit Coverage ≥ 80 %	Vollständige automatisierte UI-Tests, Last- und Skalierungstests
Betrieb	Lokale Installation auf Raspberry Pi, systemd-Dienste, Installationsskript	Containerisierung (Docker/Podman), Kubernetes, Public Cloud

Tabelle 3 Abgrenzung der Arbeit – In Scope vs. Out of Scope

Die bewusste Abgrenzung ermöglichte es, innerhalb des vorgegebenen Zeitrahmens ein stabiles Kernsystem zu entwickeln, ohne sich in umfangreichen Randthemen wie Cloud-Integration oder mobilen Anwendungen zu verlieren. Gleichzeitig wurde die Architektur so gestaltet, dass eine spätere Erweiterung in diese Richtungen möglich bleibt.

4.5 Aufbau der Arbeit

Die vorliegende Arbeit ist in mehrere Kapitel gegliedert.
Nach der Einleitung werden im **Kapitel 5** der Stand der Forschung und der Technik dargestellt. **Kapitel 6** beschreibt die funktionalen und nicht-funktionalen Anforderungen, die aus der Analyse abgeleitet wurden. In **Kapitel 7** wird die Systemarchitektur erläutert, bevor in **Kapitel 8** das Design und die Implementierung detailliert beschrieben werden.

Kapitel 9 geht auf die Verifikation, die Tests und die Qualitätssicherung ein. **Kapitel 10** behandelt DevOps-Aspekte, den Aufbau der CI/CD-Pipeline sowie den Prozess des Deployments. In **Kapitel 11** werden die erzielten Ergebnisse zusammengefasst und evaluiert.
Kapitel 12 diskutiert die gewonnenen Erkenntnisse und ordnet diese in den wissenschaftlichen Kontext ein.
Kapitel 13 beschreibt das Projektmanagement und die organisatorischen Rahmenbedingungen.
In **Kapitel 14** werden die Schlussfolgerungen gezogen und ein Ausblick auf mögliche Weiterentwicklungen gegeben.
Die Arbeit schliesst mit dem **Literaturverzeichnis** sowie den **Anhängen** ab.

5 Stand der Forschung und Technik

5.1 Gebäudeautomation & Smart Metering

Die **Gebäudeautomation** verfolgt das Ziel, den Energieverbrauch und die Betriebskosten von Gebäuden durch den Einsatz vernetzter Systeme zu optimieren. Neben Komfort und Sicherheit stehen dabei zunehmend **Energieeffizienz** und **Nachhaltigkeit** im Vordergrund. Moderne Anlagen erlauben die zentrale Steuerung und Überwachung von Heizungen, Lüftungen, Beleuchtungen oder Sicherheitssystemen.

Im Bereich **Smart Metering** werden Zählerdaten (Strom, Wasser, Wärme, Gas) systematisch erfasst und in Datenbanken gespeichert, um sie für Abrechnung, Monitoring oder Optimierungen nutzbar zu machen. Damit können Verbrauchsmuster analysiert, Anomalien erkannt und Kosten verursachergerecht verteilt werden. Studien des Fachverbands Gebäudeautomation Schweiz verdeutlichen, dass IoT-basierte Lösungen in Gebäuden eine zentrale Rolle bei der Umsetzung von Energiezielen spielen (Gebäudeautomation, 2023).

Für kleine und mittlere Objekte fehlen jedoch häufig erschwingliche, einfach installierbare Lösungen. Während Grossanlagen meist mit umfassenden Leitsystemen ausgestattet sind, haben **private Eigentümer** oder **kleine Verwaltungen** oft keinen Zugang zu vergleichbaren Werkzeugen. Dadurch entsteht eine Lücke zwischen dem verfügbaren technologischen Potenzial und dem tatsächlichen Einsatz in der Praxis.

Einsatzbereiche von Smart Metering in der Gebäudeautomation umfassen:

- **Transparenz**: Visualisierung von Energie- und Medienverbräuchen für Bewohner, Eigentümer oder Facility Manager.
- **Frühwarnsysteme**: Erkennen von Abweichungen (z. B. Leckagen, ineffiziente Heizzyklen).
- **Kosteneffizienz**: Optimierung des Betriebs und verursachergerechte Abrechnung.
- **Nachhaltigkeit**: Beitrag zur Erreichung von Klimazielen durch datengetriebene Entscheidungen.

Zur Verdeutlichung dieser Einsatzkontexte zeigt **Abbildung A1 (Use-Case Gebäudeautomation & Smart Metering)** die beteiligten Akteure und ihre wichtigsten Interaktionen mit dem System.

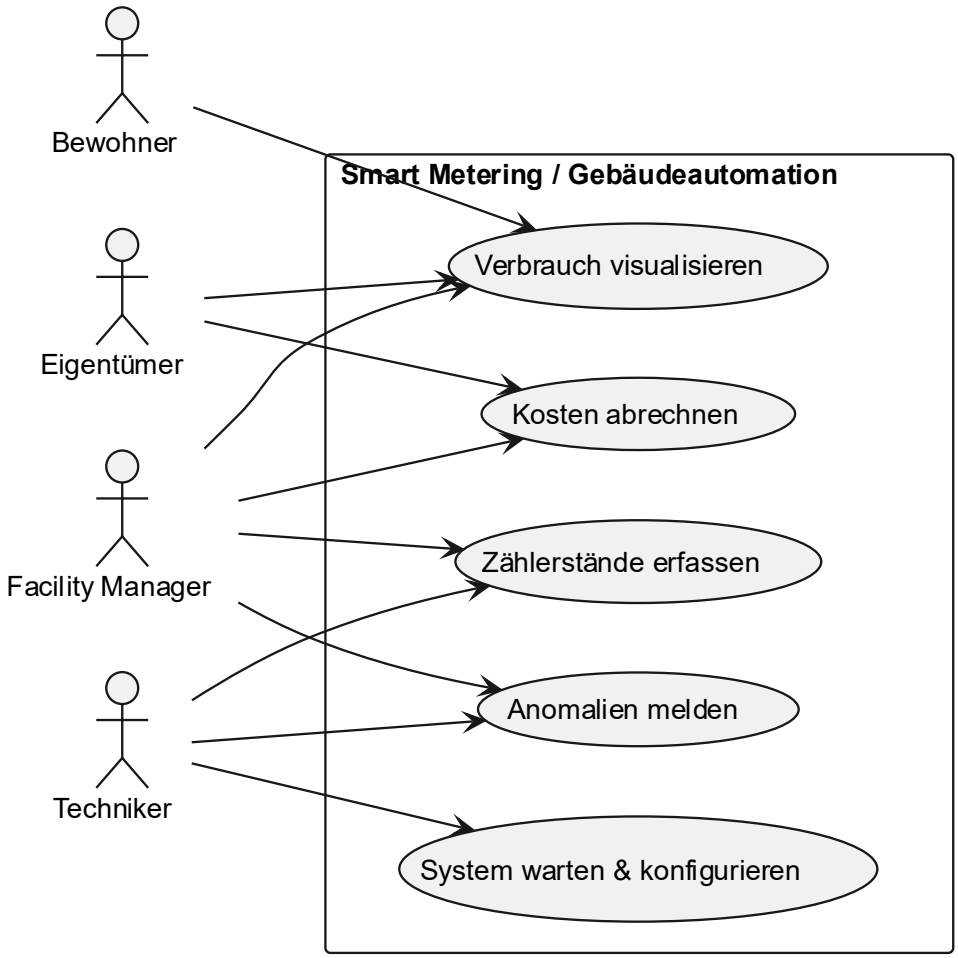


Abbildung 1 Use-Case-Diagramm Gebäudeautomation & Smart Metering

5.2 M-Bus-Protokoll – Grundlagen & Einsatzgebiete

Der **Meter-Bus (M-Bus)** ist ein in Europa normiertes Bussystem (EN 13757, 2004) für die Fernauslesung von Verbrauchszählern. Es wurde speziell für den Einsatz im Energiemanagement und Smart Metering entwickelt und zeichnet sich durch seine einfache Verkabelung, hohe Reichweite und Robustheit aus. Die Kommunikation erfolgt über eine zweidrahtige Leitung, die sowohl die Spannungsversorgung als auch die Datenübertragung übernimmt.

Ein typisches M-Bus-System besteht aus einem **Master** (z. B. Gateway oder Rechner) und mehreren **Slaves** (Zähler oder Sensoren). Jeder Slave besitzt eine eindeutige Adresse, sodass der Master die Geräte sequenziell abfragen kann. In der Praxis lassen sich bis zu 250 Geräte an einem Strang zuverlässig betreiben (Relay, 2025).

Die **Einsatzgebiete** des M-Bus sind breit gefächert:

- **Wasser-, Wärme- und Gaszähler** in Wohn- und Gewerbebauten,
- **Elektrizitätszähler** in Stromnetzen,
- **Sensorik** für Umwelt- oder Betriebsdaten,
- **Submetering** in Mehrfamilienhäusern zur verursachergerechten Kostenabrechnung.

M-Bus gilt als **kostengünstige und standardisierte Lösung**, die insbesondere für nachrüstbare Messsysteme geeignet ist. In Kombination mit IoT-Ansätzen bietet er eine zuverlässige Basis für die Integration in Gebäudeautomationssysteme.

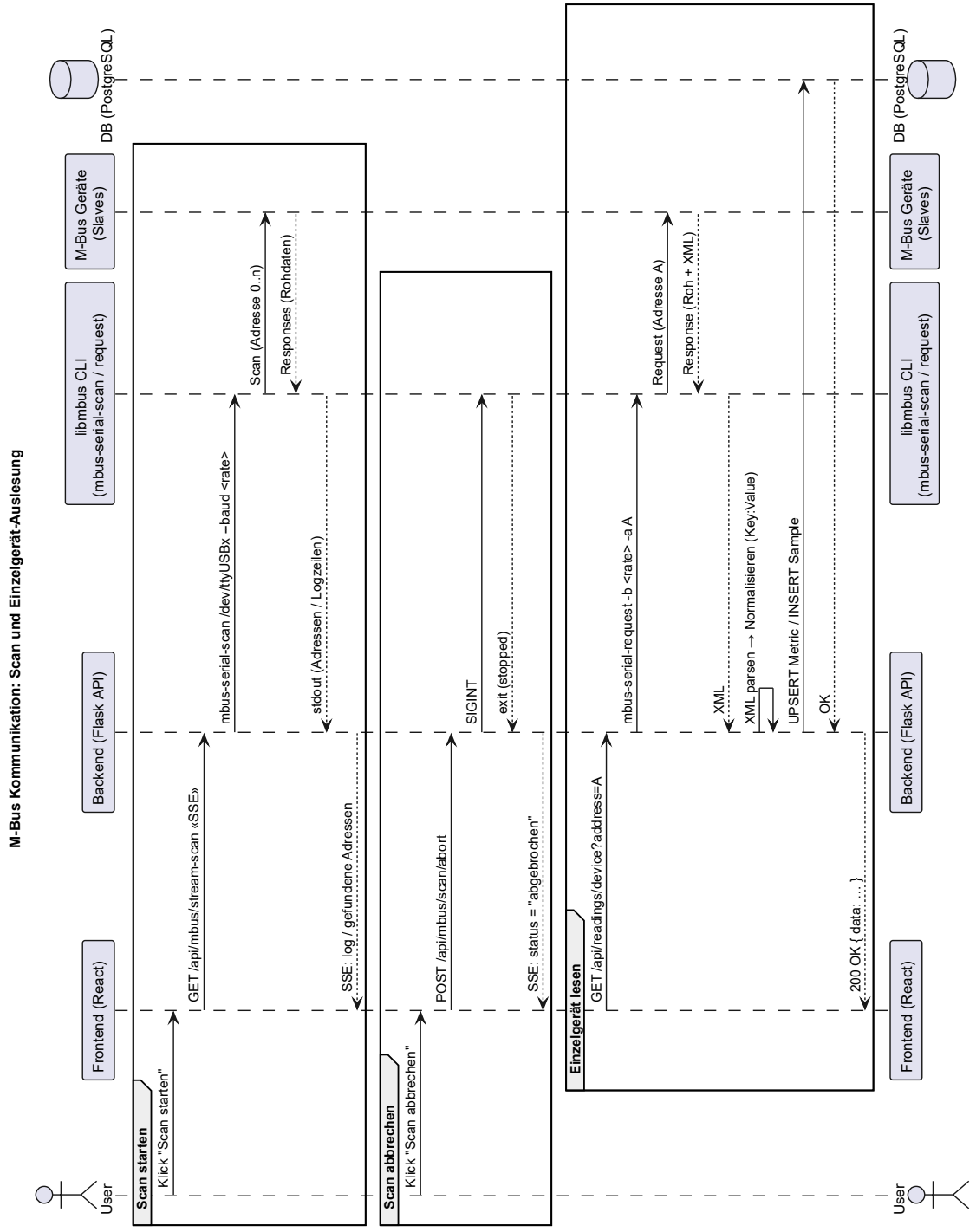


Abbildung 2 M-Bus-Kommunikation (Scan und Einzelgerät-Auslesung)

5.3 Modbus-Protokoll – Grundlagen & Einsatzgebiete

Das **Modbus-Protokoll** wurde 1979 von der Firma Modicon (heute Schneider Electric) entwickelt und gehört zu den ältesten, aber bis heute am weitesten verbreiteten Kommunikationsprotokollen in der Industrie- und Gebäudeautomation. Der Erfolg von Modbus liegt in seiner **einfachen Struktur**, der **Lizenzfreiheit** und der breiten Unterstützung durch zahlreiche Hersteller (Wikipedia, 2024).

Die Kommunikation folgt einem **Master-Slave-Prinzip**: Der Master (z. B. Steuerung oder Gateway) sendet eine Anfrage, worauf die Slaves (z. B. Zähler, Module, Sensoren) antworten. Dieses Prinzip ermöglicht eine robuste und deterministische Abfolge von Abfragen und Antworten. (Beckhoff, 2025)

Varianten von Modbus:

- **Modbus RTU**: Übertragung via serielle Schnittstellen (RS485). Häufig in der Gebäude- und Anlagenautomation eingesetzt.
- **Modbus TCP**: Übertragung über IP-basierte Netzwerke. Eignet sich für moderne Installationen mit vorhandener Netzwerk-Infrastruktur.

Einsatzgebiete von Modbus sind vielfältig:

- Steuerung und Überwachung von **Heizungs-, Klima- und Pumpenanlagen**,
- Integration von **Strom- und Wärmezählern**,
- **Digitale I/O-Module** zur Anbindung von Sensoren oder Aktoren,
- Kommunikation zwischen speicherprogrammierbaren Steuerungen (SPS).

Die Stärke von Modbus liegt in seiner **Offenheit und Standardisierung**, wodurch Geräte unterschiedlicher Hersteller kombiniert werden können. Gerade im Bereich der nachrüstbaren Gebäudeautomation bietet Modbus die Möglichkeit, bestehende Systeme ohne grossen Integrationsaufwand zu erweitern.

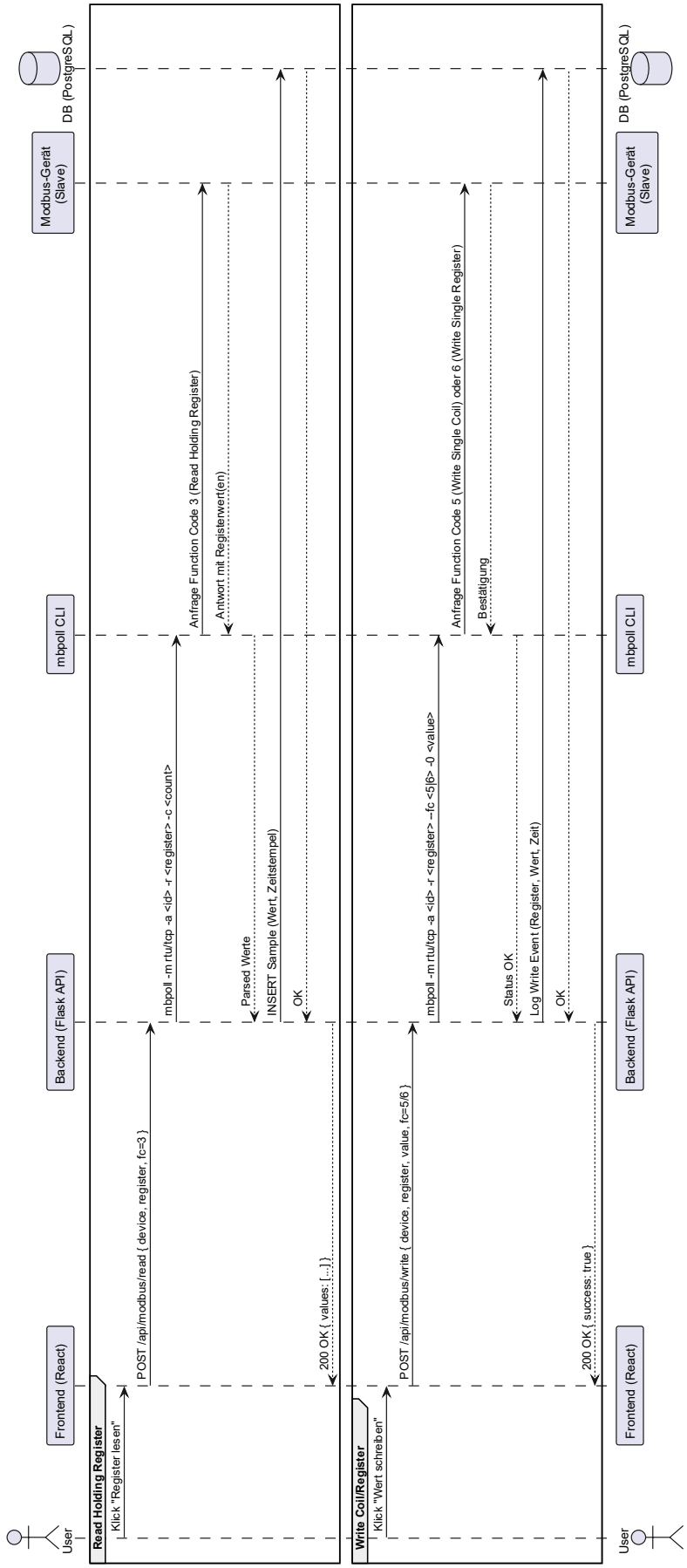


Abbildung 3 Modbus-Kommunikation (Register lesen und schreiben)

5.4 Bestehende Visualisierungslösungen und deren Limitierungen

Die Entwicklungen im Bereich **Internet of Things (IoT)** haben in den letzten Jahren erheblich zur Transformation von Facility Management und Gebäudeautomation beigetragen. Durch die zunehmende Vernetzung von Sensoren, Aktoren und Systemkomponenten entstehen neue Möglichkeiten, den Betrieb von Gebäuden effizienter, nachhaltiger und transparenter zu gestalten.

Facility Management 4.0:

- Moderne IoT-Plattformen ermöglichen die zentrale Erfassung und Analyse von Zustands- und Verbrauchsdaten.
- Betreiber profitieren von **Predictive Maintenance**-Ansätzen, bei denen Wartungsbedarfe vorausschauend erkannt werden, bevor Störungen auftreten.
- Energieberater und Verwalter erhalten fundierte Entscheidungsgrundlagen für Optimierungs- und Investitionsentscheidungen (Avelon AG, 2025).

Open-Source-Ansätze:

Die wachsende Verfügbarkeit von Open-Source-Bibliotheken und Tools (z. B. für Python, Node.js oder Java) erleichtert die Integration von Standardprotokollen wie **M-Bus** und **Modbus** in individuelle Softwarelösungen. Dadurch sinken die Kosten für die Entwicklung, und es entstehen flexible Systeme, die an unterschiedliche Kontexte angepasst werden können. Insbesondere für kleinere Objekte bieten Open-Source-Lösungen einen attraktiven Gegenpol zu proprietären, kostenintensiven Komplettsystemen.

IoT in der Forschung:

Untersuchungen zeigen, dass die Nutzung von IoT in Gebäuden einen signifikanten Beitrag zur Reduktion des Energieverbrauchs leisten kann. Gleichzeitig steigt die Bedeutung von Datensouveränität: Viele Betreiber bevorzugen lokal installierbare Systeme, um sensible Daten nicht an externe Cloud-Dienste abgeben zu müssen (Gebäudeautomation, 2023).

Der wissenschaftliche Kontext dieser Arbeit verortet sich somit an der Schnittstelle von **praktischen Anforderungen kleiner Betreiber**, **technologischen Trends im IoT** und **offenen Softwarearchitekturen**, die den Weg für modulare, nachrüstbare Lösungen ebnen.

5.5 Wissenschaftlicher Kontext (IoT, Facility Management, Open Source)

In der Schweiz werden digitale Technologien zunehmend im Bereich Facility Management eingesetzt, um Gebäude effizienter, sicherer und nachhaltiger zu betreiben. Die Avelon AG zeigt in ihren Projekten, dass IoT-basierte Plattformen eine Schlüsselrolle spielen. Sie verbinden Sensoren, Gebäudeleittechnik und Datenanalyse in einem einheitlichen System und ermöglichen so einen kontinuierlichen Überblick über den Zustand von Gebäuden. Damit lassen sich Betriebsprozesse optimieren, Energie sparen und Störungen frühzeitig erkennen (Avelon AG, 2025).

Im Facility Management eröffnet die Vernetzung von Daten neue Möglichkeiten. Echtzeitüberwachung, automatisierte Alarmierungen und vorausschauende Wartung sind zentrale Anwendungsfelder. Dadurch wird nicht nur die Betriebssicherheit erhöht, sondern auch die Wirtschaftlichkeit verbessert, da Ressourcen gezielt eingesetzt werden können.

Open-Source Technologien sind in diesem Kontext von besonderer Bedeutung. Sie stehen in Form frei zugänglicher Bibliotheken, meist für Unix-basierte Systeme, zur Verfügung und können direkt heruntergeladen und in Projekte integriert werden. Für Programmiersprachen wie Python existieren zahlreiche Open-Source Bibliotheken, welche die Anbindung von Protokollen wie M-Bus oder Modbus erleichtern. Damit lassen sich ohne hohe Lizenzkosten professionelle Funktionen in Softwarelösungen einbinden, was die Entwicklung innovativer und flexibler Anwendungen im Bereich Gebäudeautomation wesentlich unterstützt.

6 Anforderungen

6.1 Stakeholder und Anwendergruppen

Die Entwicklung eines Visualisierungssystems für Zähler- und Anlagendaten adressiert unterschiedliche **Stakeholder** mit teils stark divergierenden Bedürfnissen und Anforderungen:

Primäre Stakeholder

- **Private Eigentümer**
  Wollen Transparenz über den eigenen Energieverbrauch (z. B. Öl, Wasser, Strom, Wärme). Ziel ist die Nachvollziehbarkeit von Kosten sowie die Möglichkeit, durch gezielte Massnahmen Energie einzusparen.
- **Facility Manager / Verwaltungen**
  Benötigen Werkzeuge zur Erfassung, Aufbereitung und verursachergerechten Abrechnung von Energieverbräuchen. Sie legen besonderen Wert auf die Integration mehrerer Zähler und die Möglichkeit, Störungen frühzeitig zu erkennen.
- **Servicetechniker**
  Müssen Betriebszustände und Fehlermeldungen effizient analysieren können, um gezielt vorbereitet zu Serviceeinsätzen zu fahren. Benötigen detaillierte Diagnosemöglichkeiten und teilweise die Möglichkeit, Steuerbefehle (z. B. Quittierung von Fehlermeldungen) auszuführen.

Sekundäre Stakeholder

- **Energieberater**
  Nutzen systematisch erfasste Daten, um Effizienzpotenziale zu identifizieren und Optimierungsvorschläge abzuleiten. Diese Zielgruppe gewinnt insbesondere in späteren Ausbaustufen des Systems an Bedeutung.
- **Forschung & Entwicklung**
  Profitieren von offenen Schnittstellen und modularen Architekturen, um neue Algorithmen oder Systeme auf Basis realer Verbrauchsdaten zu erproben.

Anwendergruppen (User Roles im System)

- **Endanwender (Owner/Resident)**: erhalten eine intuitive Visualisierung der Verbrauchsdaten, ohne technische Details.
- **Techniker (Technician)**: nutzen erweiterte Ansichten für Diagnose, Konfiguration und Steuerung.
- **Verwalter (Manager)**: kombinieren die beiden Perspektiven, mit Fokus auf Abrechnung und Gesamtüberwachung.

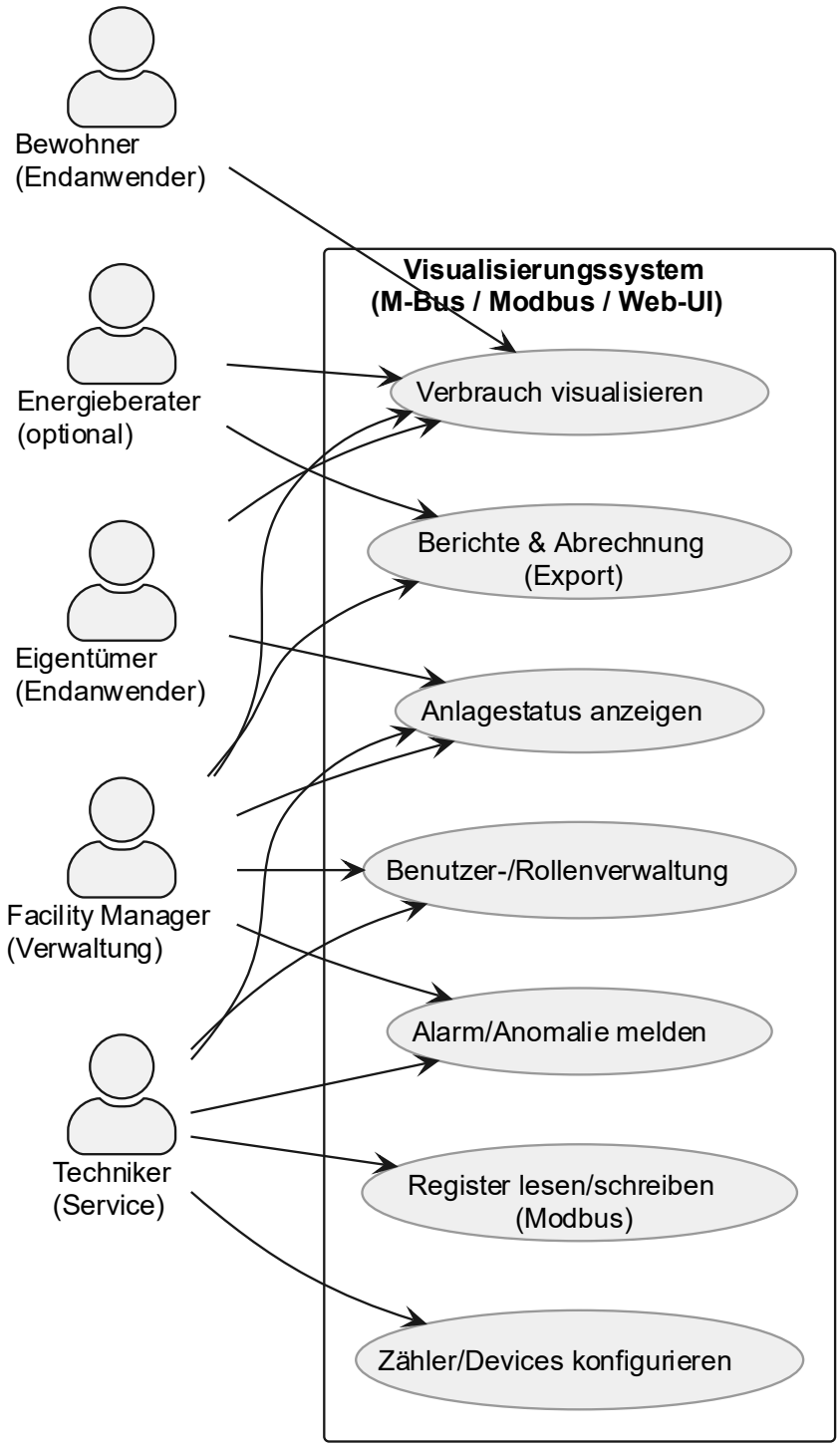


Abbildung 4 Stakeholder und Systeminteraktionen (Rollen ↔ Kernfunktionen)

6.2 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben, welche konkreten Leistungen das Visualisierungssystem erbringen muss. Sie orientieren sich an den identifizierten Stakeholdern und stellen sicher, dass die Kernziele der Arbeit messbar erfüllt werden.

ID	Anforderung	Akteur(e)	Beschreibung	Akzeptanzkriterium
F1	M-Bus Scan	Facility Manager, Techniker	Automatische Erkennung von Zählern im M-Bus-Strang	Gefundene Adressen werden in der Weboberfläche gelistet
F2	M-Bus Auslesen	Facility Manager, Techniker	Einzelgeräte können selektiv ausgelesen werden	Werte erscheinen als strukturierte Tabelle im UI
F3	Modbus Read	Techniker	Registerwerte (Coils, Inputs, Holding) können ausgelesen werden	JSON-Response im Backend; Werte im Frontend angezeigt
F4	Modbus Write	Techniker	Werte in Registern/Coils können geschrieben werden	Änderung wird bestätigt und im System geloggt
F5	Quittierung	Techniker	Fehlermeldungen können quittiert werden	Button „Quittieren“ löscht Meldung sichtbar im UI
F6	Visualisierung	Endanwender, Manager	Verbrauchs- und Betriebsdaten werden als Tabellen und Charts angezeigt	Tages-/Wochen-/Monatswerte in Diagrammen sichtbar
F7	Geräte-Monitor	Manager, Techniker	Zyklische Überwachung mehrerer Geräte mit Mindestabständen	Werte werden alle ≥30s aktualisiert
F8	Benachrichtigung	Manager, Techniker	Kritische Fehler führen zu E-Mail-Alarmen	E-Mail wird versendet, sobald Service stoppt/restarted
F9	Benutzerrollen	Endanwender, Techniker, Manager	Unterschiedliche Rollen erhalten angepasste Ansichten	Endanwender sehen nur Visualisierung; Techniker Konfig/Write

F10	Export	Manager, Berater	Verbrauchsdaten können exportiert werden (CSV)	Exportdatei wird erfolgreich heruntergeladen

Tabelle 4 Funktionale Anforderungen des Visualisierungssystems

6.3 Nicht-funktionale Anforderungen

Neben den funktionalen Features muss das System auch Qualitätsmerkmale erfüllen, die für den stabilen, sicheren und nutzerfreundlichen Betrieb entscheidend sind.

ID	Kategorie	Anforderung	Beschreibung	Akzeptanzkriterium
NF1	Usability	Einfache Bedienung	Die Oberfläche ist intuitiv und benötigt keine technischen Vorkenntnisse.	Endanwender können Verbrauchsdaten ohne Schulung interpretieren.
NF2	Usability	Responsives Design	UI ist auf Desktop, Tablet und Smartphone nutzbar.	Seitenlayout passt sich automatisch an Bildschirmgrösse an.
NF3	Performance	Antwortzeiten	API-Requests sollen in < 1 Sekunde beantwortet werden.	95 % aller Abfragen unter Last < 1 Sekunde.
NF4	Zuverlässigkeit	Systemstabilität	Dauerbetrieb ohne ungeplante Neustarts ≥ 30 Tage.	Langzeittest ohne Absturz oder Datenverlust.
NF5	Zuverlässigkeit	Monitoring	Backend-Service prüft regelmässig seinen Zustand.	Bei Ausfall automatische E-Mail-Benachrichtigung.
NF6	Sicherheit	Zugriffsschutz	Rollenmodell regelt Rechte für Endanwender, Manager, Techniker.	Nur berechtigte Rollen können schreiben oder konfigurieren.
NF7	Sicherheit	Fernzugriff	Zugriff nur über VPN (Tailscale).	Ohne VPN ist keine Verbindung von extern möglich.

NF8	Erweiterbarkeit	Modularität	Neue Treiber (z. B. weitere Protokolle) lassen sich ergänzen.	Neue Module können ohne Änderung des Kerns integriert werden.
NF9	Erweiterbarkeit	Konfigurierbarkeit	Geräte können über JSON oder UI konfiguriert werden.	Änderungen sind ohne Neustart möglich.
NF10	Wartbarkeit	Clean Code & Tests	Code ist modular, dokumentiert und ≥ 80 % abgedeckt.	CI/CD Pipeline schlägt bei Unterschreitung fehl.

Tabelle 5 Nicht-funktionale Anforderungen des Visualisierungssystems

6.4 Risikoanalyse

Die Risikoanalyse bewertet potenzielle Gefahren für die erfolgreiche Umsetzung und den Betrieb des Systems. Sie berücksichtigt sowohl technische als auch organisatorische Risiken.

ID	Risiko	Auswirkung	Eintrittswahrscheinlichkeit	Massnahmen / Mitigation
R1	**Komplexität der DB-Anbindung**	Daten können nicht zuverlässig gespeichert/abgerufen werden	Mittel	Einsatz von PostgreSQL mit klar definiertem Schema; frühe Integrationstests; Migrationsskripte.
R2	**Serielle Schnittstellen instabil**	Kommunikationsabbrüche bei M-Bus oder Modbus RTU	Hoch	Implementierung von Abbruchmechanismen (SIGINT bei Scan), Queue-Handling pro Port, Timeout-Parameter in `.env`.
R3	**Parallele Nutzung von M-Bus und Modbus**	Kommunikationskollisionen, blockierende Prozesse	Mittel	Zentrale Queue im Backend, Mindestabstände im Monitor, saubere Prozesskontrolle.

R4	**Fehlende Skalierbarkeit**	System überlastet bei > 50 Geräten	Niedrig	Architektur modular halten, optionale Erweiterung mit TimescaleDB; Fokus aktuell auf kleine Installationen.
R5	**Sicherheitsrisiken**	Unbefugter Zugriff oder Manipulation	Mittel	VPN-Zugang (Tailscale), Brute-Force-Schutz, unattended-upgrades, Rollenmodell.
R6	**Zeitmanagement**	Projektumfang überschreitet Zeitrahmen	Mittel	Priorisierung auf Soll-Ziele, Kann-Ziele nur bei Ressourcenspielraum.
R7	**Hardware-Abhängigkeit**	Defekte oder inkompatible Adapter blockieren Tests	Mittel	Einsatz geprüfter Adapter (libmbus, RS485), Ersatzhardware verfügbar halten.
R8	**Benutzerakzeptanz**	Endanwender finden UI zu komplex	Niedrig	Klare Rollentrennung (Endanwender vs. Techniker); frühe Nutzerbefragung (vgl. Interviews im Anhang).
R9	**Teamkoordination**	Missverständnisse zwischen Entwicklern	Niedrig	Agile Sprints, PR-Reviews, klare Verantwortlichkeiten (RACI-Matrix).
R10	**Datenintegrität**	Fehlerhafte oder unvollständige Daten in der DB	Mittel	Validierung im Backend, Logging von API-Fehlern, Monitoring der Writes.

Tabelle 6 Risikomatrix des Visualisierungssystems

7 Systemarchitektur

Die Architektur des entwickelten Systems folgt einem mehrschichtigen Modell, das die Trennung von Hardware, Treibern, Backend, Persistenz und Frontend sicherstellt. Ziel war es, eine modulare, erweiterbare und robuste Struktur zu schaffen, die sowohl den zuverlässigen Betrieb im Alltag als auch eine spätere Weiterentwicklung ermöglicht.

7.1 Gesamtübersicht (Diagramm)

Abbildung 5 Systemarchitektur – Durchstich von Feldgeräten bis Web-UI

7.2 Hardware-Architektur (Raspberry Pi, Adapter, Zähler)

Die Hardware-Architektur basiert auf einem **Raspberry Pi 4** als zentrale Plattform.

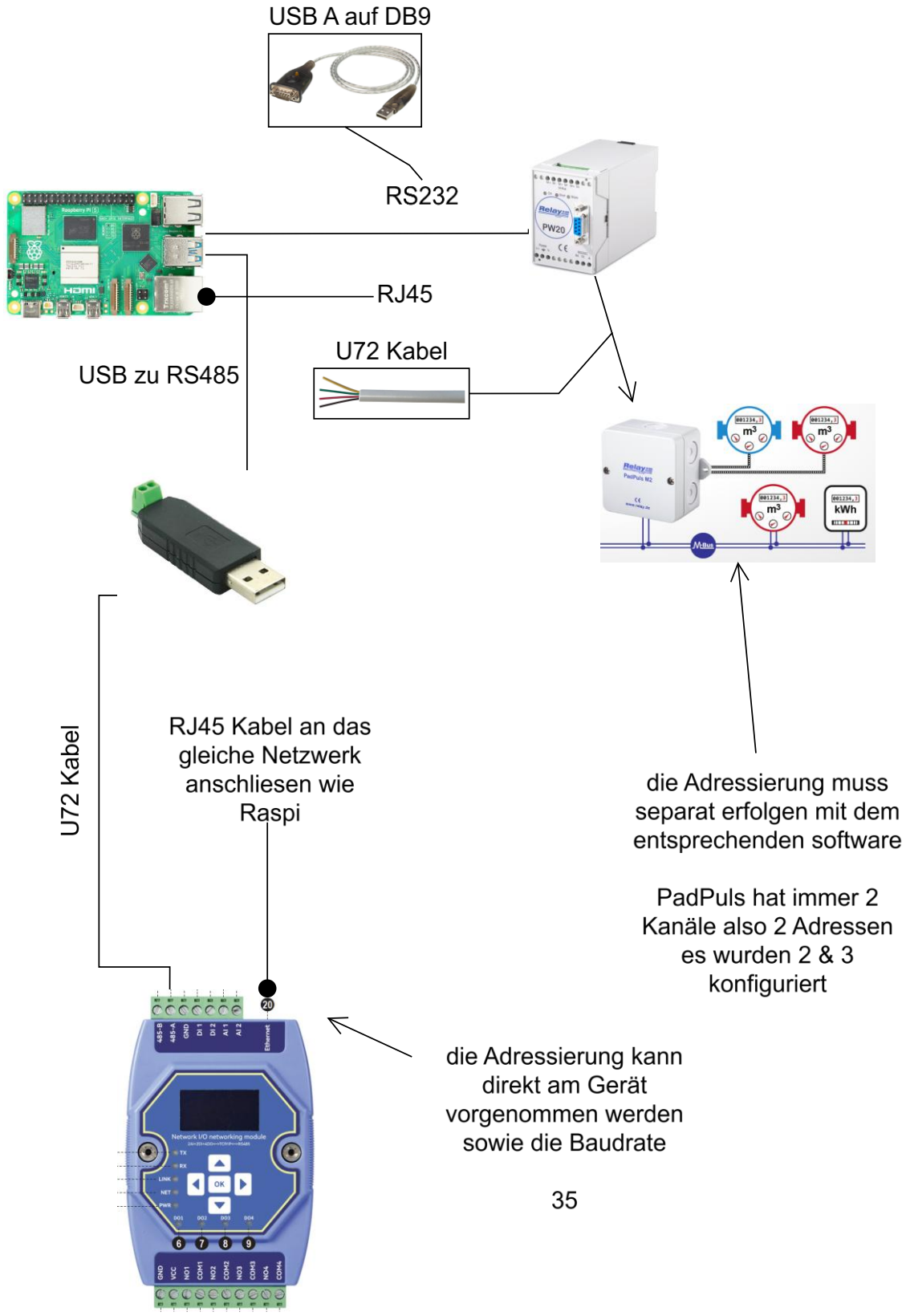


Abbildung 6 Hardware-Architektur (Raspberry Pi mit M-Bus- und Modbus-Adaptern)

7.3 Software-Architektur

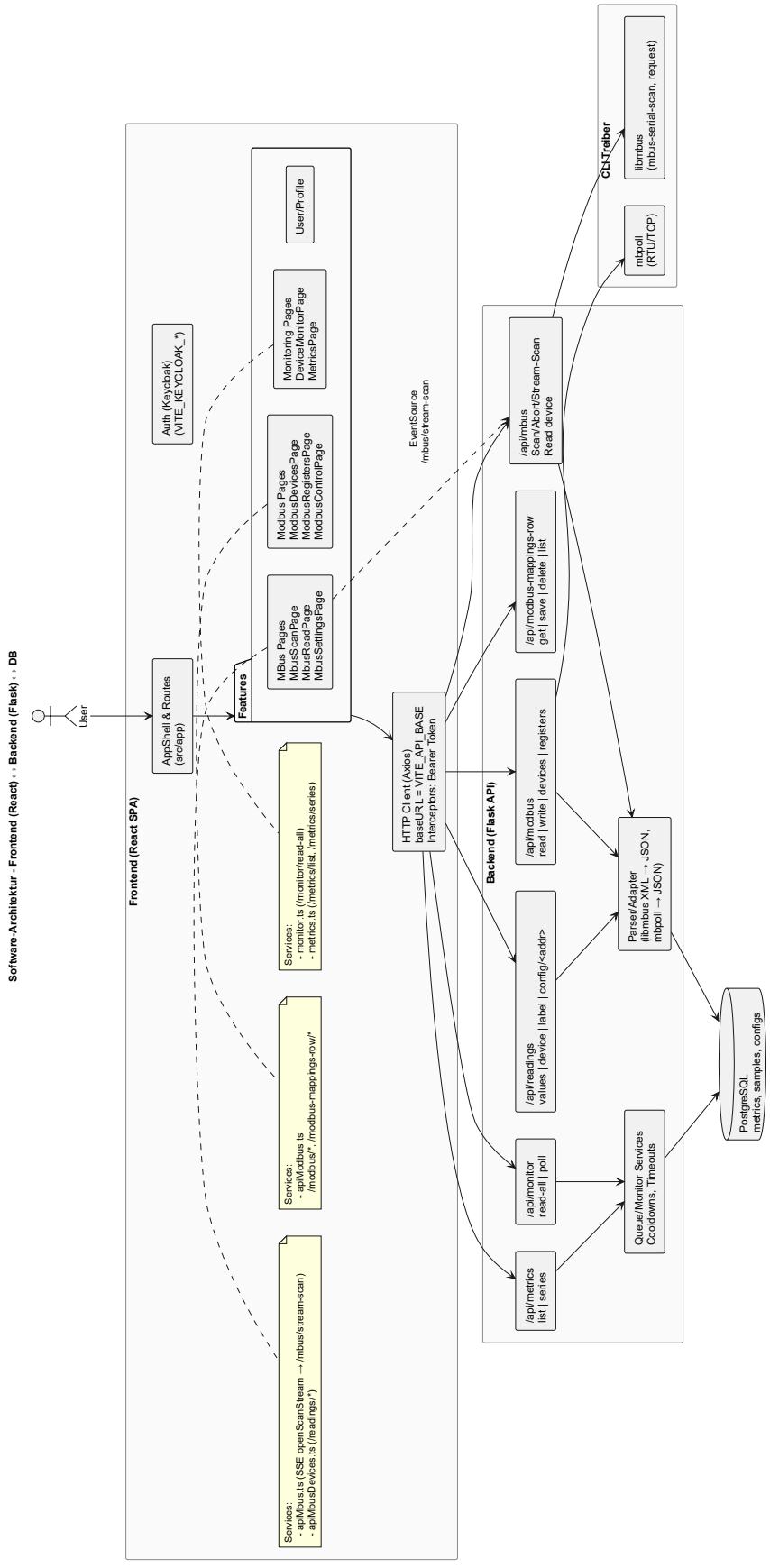


Abbildung 7 Software-Architektur (Module, Flows, Schnittstellen)

7.3.1 Backend (Flask, API-Struktur)

Das Backend kapselt die Protokollanbindung über CLI-Tools (**libmbus**, **mbpoll**) und stellt eine **REST-API** für das Frontend bereit. Längere Operationen (z. B. M-Bus-Scan) werden per **SSE** gestreamt. Eine Service-Schicht regelt **Queue/Cooldown/Timeouts** und persistiert Messwerte sowie Metadaten in **PostgreSQL**.

Bereich	Endpunkt	Methode	Zweck
M-Bus	/api/mbus/stream-scan	GET (SSE)	Live-Scan via Server-Sent Events (Logzeilen, gefundene Adressen)
M-Bus	/api/mbus/scan/abort	POST	Laufenden Scan abbrechen (sauberer SIGINT)
Readings	/api/readings/values	GET	Bekannte Geräteadressen und Metadaten
Readings	/api/readings/device?address=A	GET	Einzelgerät auslesen (XML → JSON normalisiert)
Readings	/api/readings/label	POST	Gerätename setzen/ändern
Readings	/api/readings/config/<address>	GET	Feldauswahl pro Gerät
Modbus	/api/modbus/read	POST	Register lesen (RTU/TCP; FC 1/2/3/4)
Modbus	/api/modbus/write	POST	Coil/Register schreiben (FC 5/6/16)
Modbus Mappings	`/api/modbus-mappings-row/get	save	delete
Monitoring	/api/monitor/read-all	GET	Aggregierte Anzeige der überwachten Geräte
Metrics	/api/metrics/list	GET	Verfügbare Metriken
Metrics	/api/metrics/series	GET	Zeitreihen (Range, Bucket, Rate)

Tabelle 7 Zentrale API-Endpunkte (Auszug)

Hinweise zur Implementierung

CLI-Aufrufe werden robust gekapselt, Ausgaben geparst und in **konsistente JSON-Envelopes** überführt.

Queue pro Bus/Port verhindert Kollisionen; **Cooldowns** im Monitor sichern Mindestabstände.

Persistenz: metrics (Metadaten), samples (Zeitreihen).

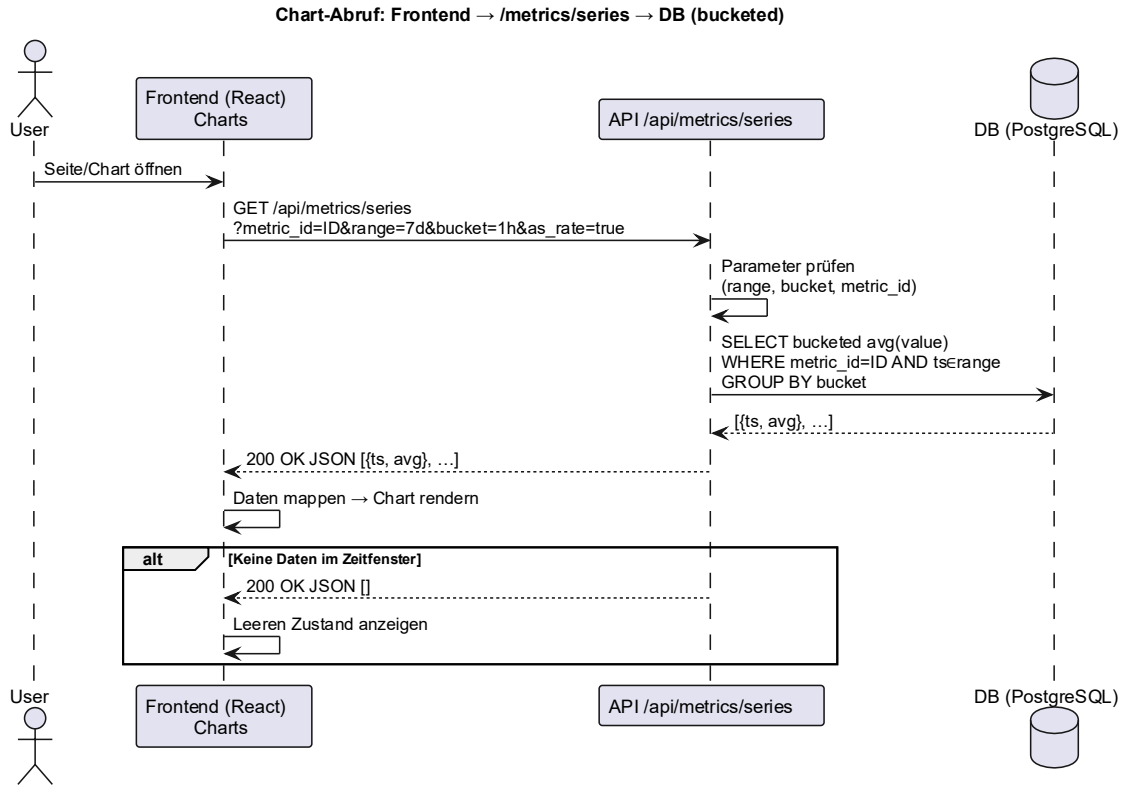


Abbildung 8 Frontend-Chart Abruf → /metrics/series (Bucketed Zeitreihen)

7.3.2 Frontend (React, Visualisierung)

Das Frontend ist eine **SPA** auf Basis von **Vite + React + TypeScript**. API-Zugriffe laufen über eine zentrale Axios-Instanz mit baseURL = VITE_API_BASE. Optional bindet **Keycloak** Rollen/Token ein. Funktionen sind modular als **Features** umgesetzt.

Modul	Seiten/Komponenten	Services (Beispiele)	Zweck
M-Bus	MbusScanPage, MbusReadPage, MbusSettingsPage	apiMbus.ts (SSE), apiMbusDevices.ts	Scannen, Auslesen, Konfiguration
Modbus	ModbusDevicesPage, ModbusRegistersPage, ModbusControlPage	apiModbus.ts	Geräte, Register, Read/Write
Monitoring	DeviceMonitorPage, MetricsPage	monitor.ts, metrics.ts	Zyklische Anzeige, Zeitreihen
User	Profile.tsx	–	Nutzersicht/Profil
Shared/App	AppShell, routes.tsx, ApiCheck	httpClient.ts	Routing, HTTP, Layout

Tabelle 8 Frontend-Module und Seiten

Datenflüsse

- **SSE** für Live-Scan: EventSource auf /mbus/stream-scan.
- Standard-Requests über Axios-Client; Interceptor fügt bei Bedarf **Bearer Token** an.

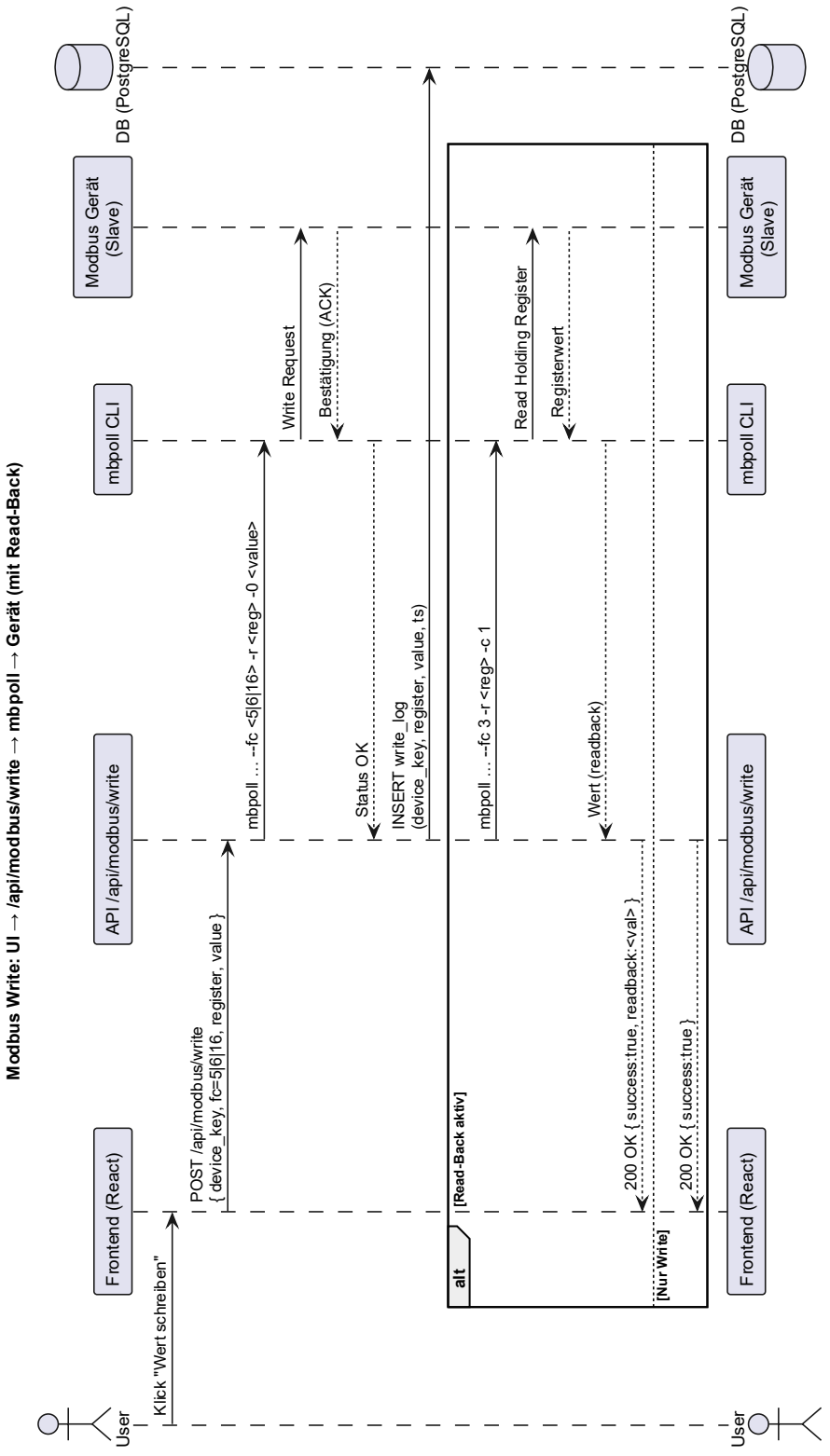


Abbildung 9 Modbus Write mit Read-Back-Verifikation und Logging

7.3.3 Datenhaltung PostgreSQL

Die DB hält Metadaten zu Metriken und **Zeitreihen** der Messwerte.

Objekt	Felder (Beispiel)	Beschreibung
`metrics`	`id`, `tag`, `unit`, `source`, `type`	Definition eines Messsignals
`samples`	`metric_id`, `ts`, `value`	Zeitstempelwert eines Signals
`device_config`	`address`, `fields[]`	Ausgewählte Felder pro M-Bus-Gerät
`labels`	`address`, `name`	Anzeigename pro Adresse

Tabelle 9 Datenobjekte (vereinfacht)

Optionale Erweiterung: TimescaleDB für grössere Datenmengen.

7.3.4 Konfigurationsdateien (Excel, JSON)

Konfiguration erfolgt **ohne Excel**, direkt über UI und JSON-Dateien.

Datei	Zweck
`mbus_settings.json`	Serielle Parameter, Port/baud für M-Bus
`known_mbus_devices.json`	Persistente Liste bekannter Adressen und Namen
`config_<address>.json`	Feldauswahl pro M-Bus-Gerät
`modbus_devices.json`	Geräte, Protokoll (RTU/TCP), Slave-ID, Ports
`modbus_registers.json`	Registerdefinitionen pro Gerät
`.env`	Ports, Timeouts, CORS, DB-URL, Pfade

Tabelle 10 Relevante Konfigurationsdateien

7.4 Kommunikationsabläufe (M-Bus, Modbus, Web-API)

Die Kommunikationsabläufe sind so gestaltet, dass sie robust, nachvollziehbar und erweiterbar bleiben. Längere oder serielle Operationen (z. B. M-Bus-Scan) laufen streamend über SSE, Modbus-Reads/Writes sind kurzlebig und werden über CLI sauber geparst. Ein zentraler Monitor bewahrt Mindestabstände pro Gerät und verhindert Kollisionen.

M-Bus (Scan und Einzelgerät)

Der Frontend-Scan öffnet einen SSE-Stream und empfängt laufend Logzeilen sowie gefundene Adressen. Ein Abbruch beendet den Scan kontrolliert. Einzelgeräte werden on demand abgefragt, die XML-Antworten normalisiert und optional persistiert.
→ siehe Abbildung „M-Bus Kommunikation (Scan und Einzelgerät-Auslesung)“

Modbus (Read/Write)

Register werden synchron gelesen und geschrieben. Auf Wunsch erfolgt ein Read-Back zur Verifikation und ein Log-Eintrag in der Datenbank.
→ siehe Abbildung „Modbus Kommunikation (Register lesen und schreiben)“

Gerätemonitor mit Cooldown

Das Frontend pollt aggregierte Zustände. Das Backend entscheidet pro Gerät, ob Cache genutzt wird (Cooldown aktiv) oder frisch gelesen wird (Cooldown abgelaufen). Dabei wird pro Bus/Port

eine Queue genutzt, damit nichts gleichzeitig ineinanderfährt.
→ siehe Abbildung „Monitor-Poll mit Cooldown (Mehrere Geräte, 30 s Mindestabstand)“

Web-API Envelope und Fehlerbehandlung

Alle Endpunkte liefern einen konsistenten JSON-Envelope. Fehlende oder ungültige Parameter führen zu 4xx, interne Fehler zu 5xx. Längere Streams (SSE) signalisieren Statuswechsel explizit, damit die UI nicht raten muss.

Kontext	HTTP	Envelope	Bedeutung	UI-Verhalten
Erfolg	200	`{ "ok": true, "data": … }`	Ergebnis gültig	Anzeigen/plotten
Ungültige Parameter	400	`{ "ok": false, "error": "bad_request", "message": "…" }`	Request fehlerhaft	Validierung hervorheben
Nicht gefunden	404	`{ "ok": false, "error": "not_found", "message": "…" }`	Gerät/Resource fehlt	Hinweis anzeigen
Timeout/Abort	408/499	`{ "ok": false, "error": "timeout", "message": "…" }`	Gerät antwortet nicht	Retry/Abort in UI anbieten
Interner Fehler	500	`{ "ok": false, "error": "internal", "message": "…" }`	Unerwarteter Fehler	Fehlermeldung + Log-ID

Tabelle 11 JSON-Envelope und typische Fehlerfälle

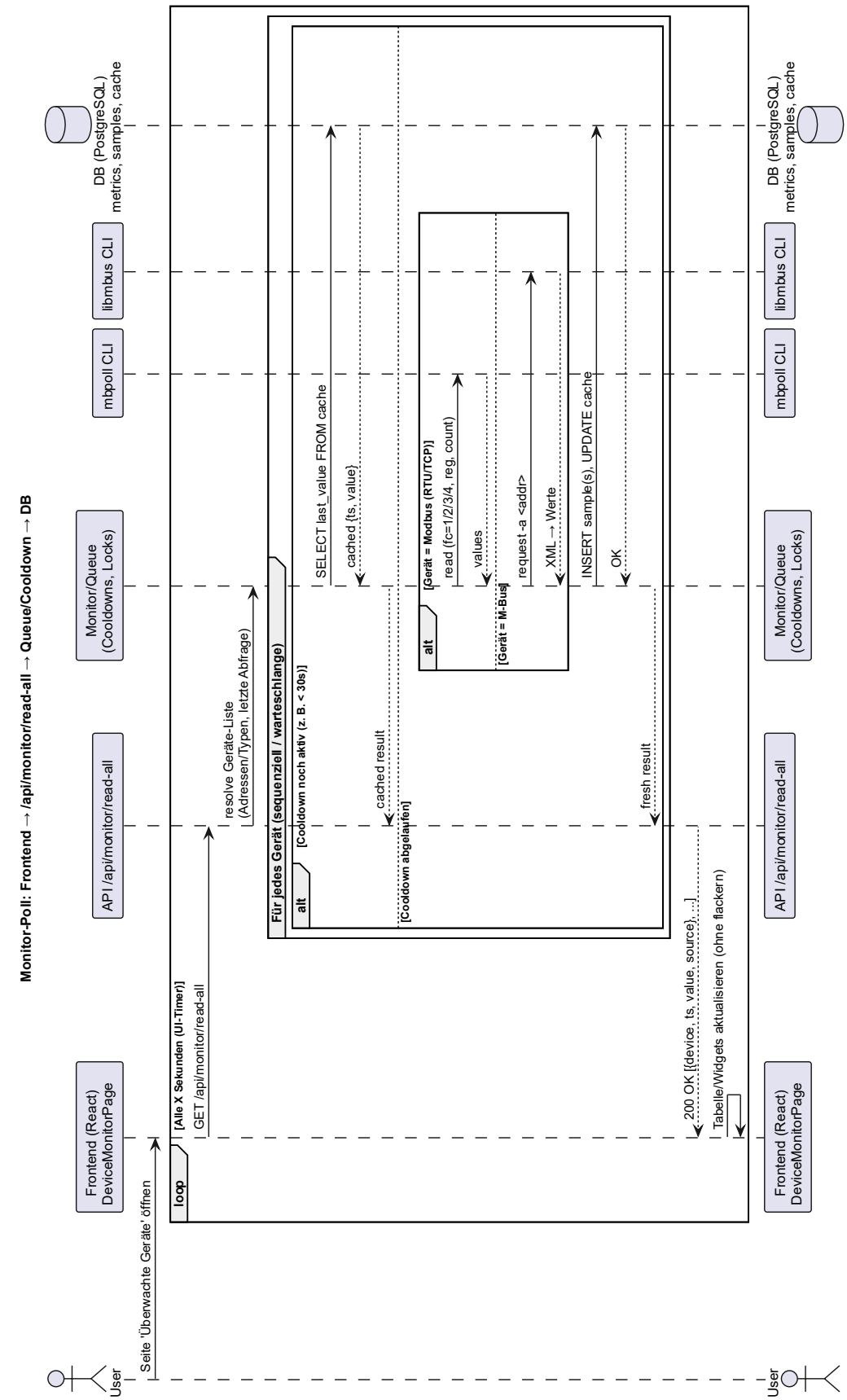


Abbildung 10 Monitor-Poll mit Cooldown (Mehrere Geräte, 30 s Mindestabstand)

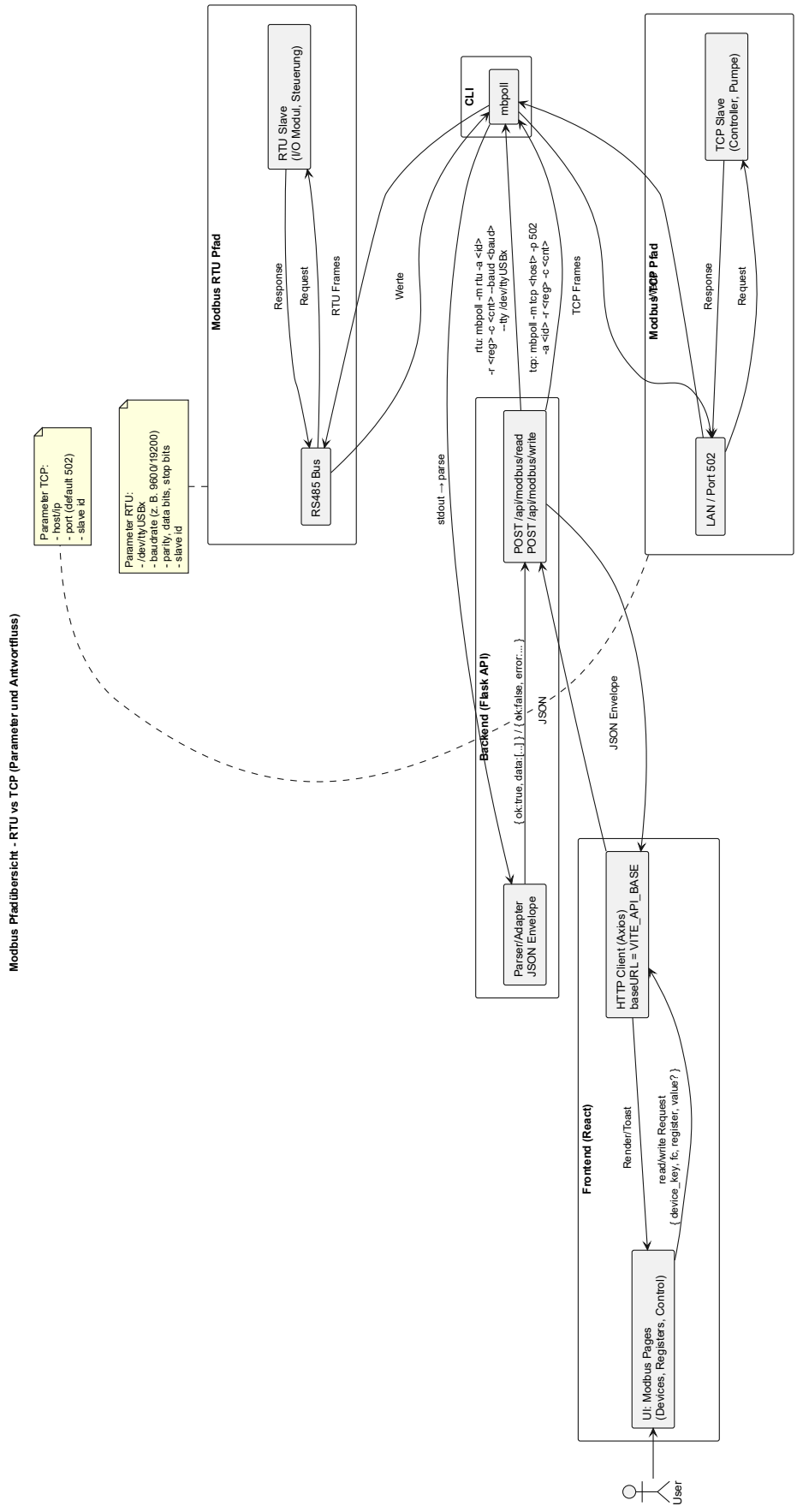


Abbildung 11 Modbus Pfadübersicht RTU vs TCP (Parameter und Antwortfluss)

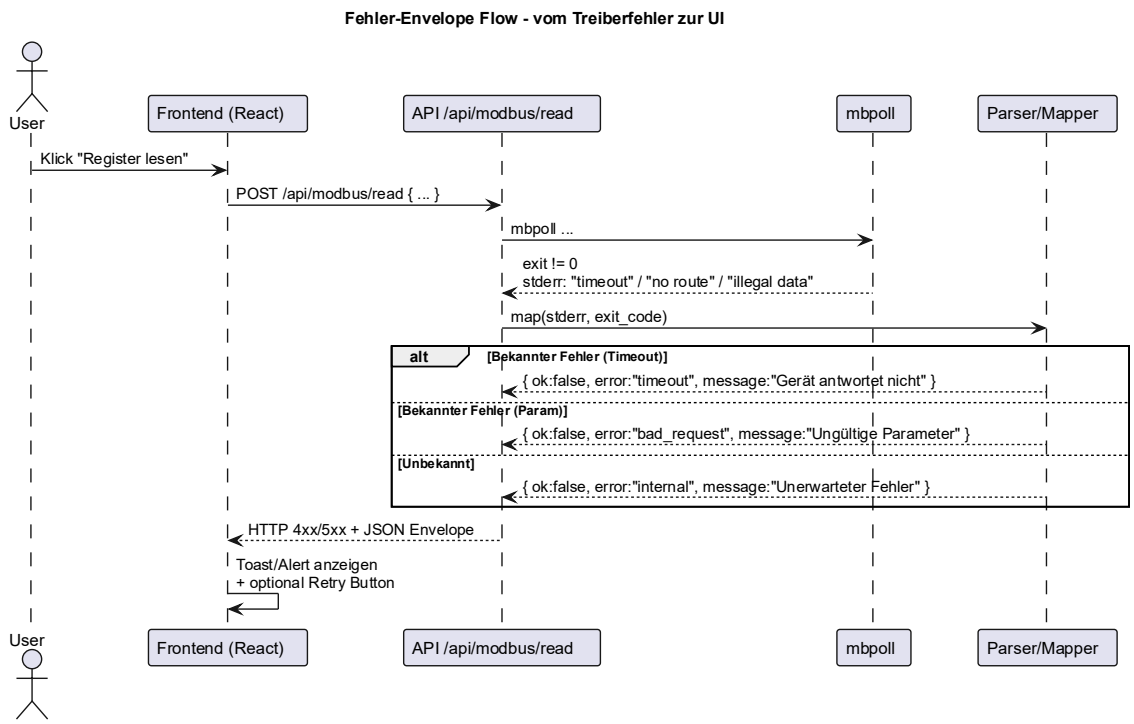


Abbildung 12 Fehler-Envelope Flow (Treiberfehler → API → UI-Meldung)

8 Design und Implementierung

Das System wurde nach dem Prinzip eines Walking Skeleton entwickelt: von Beginn an war ein minimal lauffähiger End-to-End-Prototyp vorhanden, der sukzessive erweitert wurde. Diese Vorgehensweise ermöglichte es, technische Risiken frühzeitig zu identifizieren und Kernfunktionen iterativ auszubauen.

8.1 Projektmethodik

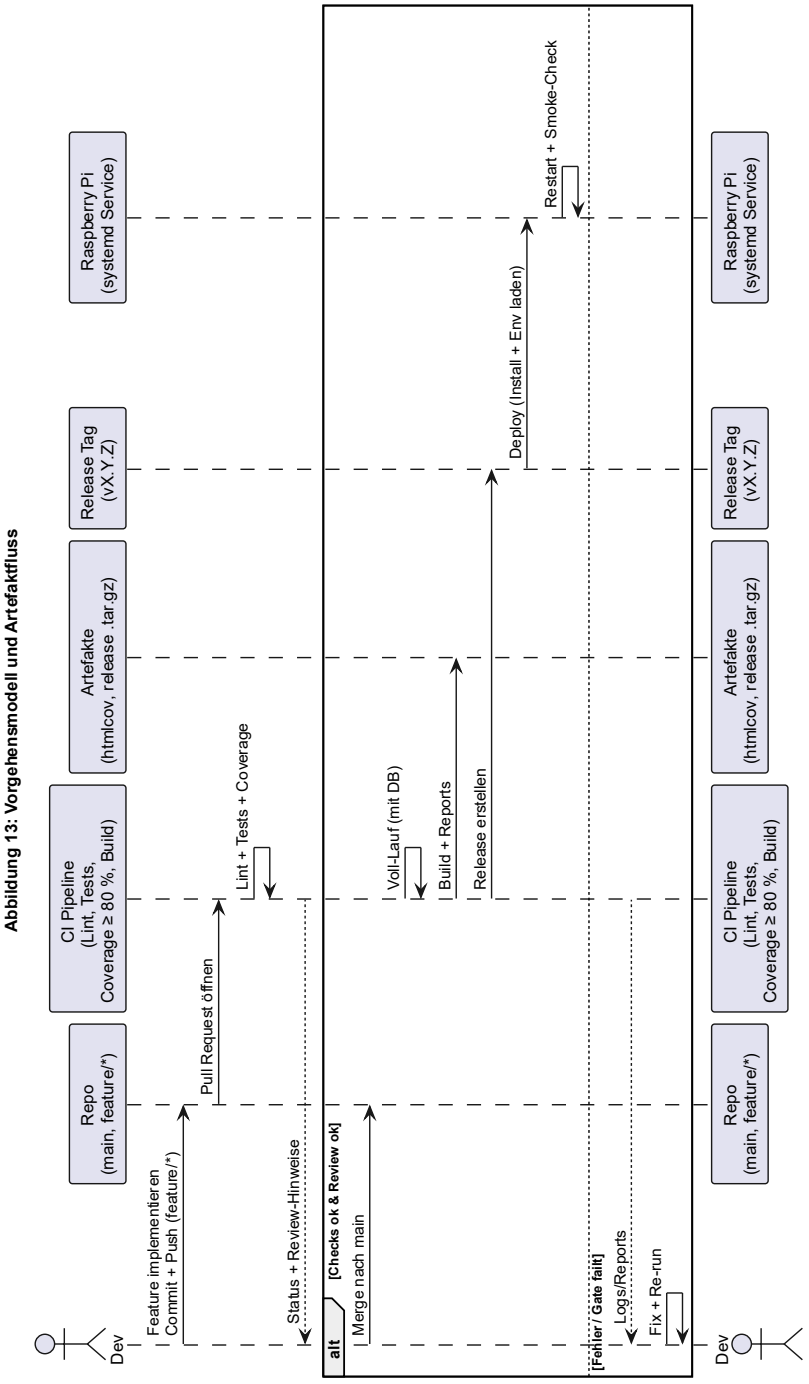


Abbildung 13 Vorgehensmodell und Artefaktfluss

Die Umsetzung lief schlank agil mit kurzen Iterationen und klaren Qualitätsgittern:

- **Arbeitsmodus**
  Feature Branch pro Aufgabe, Pull Request mit Review. Keine Direktcommits auf main.
- **Qualitätsgitter**
  Linting und Unit/Integration Tests sind Pflicht. Coverage Gate **≥ 80 %** im Backend. Fehlende Tests blockieren den Merge.
- **CI Pipeline**
  PR: schneller Lauf ohne DB für Feedback. main/Tag: Volltest mit Postgres, Artefakte (Coverage HTML, Release Bundle).
- **Releases & Deploy**
  Semantisches Tagging vX.Y.Z, Release-Bundle auf den Raspberry ausrollen, systemd Restart, **Smoke-Test** via Health Endpoint.
- **Dokumentation**
  Kurz und direkt im Repo gepflegt. Änderungen an API/Flows werden zeitnah nachgezogen, Screens in den Anhang.

8.2 Walking Skeleton / Prototyp

Der Prototyp liefert einen **End to End Durchstich** über alle Schichten, damit Risiken früh sichtbar werden und Architekturentscheidungen auf Fakten beruhen. Der Ablauf:

- **UI → Health**: Start der App, schneller Funktionscheck.
- **M-Bus**: Live-Scan als **SSE** mit Log und gefundenen Adressen, **Abbruch** sauber per Endpoint. Einzelgerät wird on demand gelesen, XML wird normalisiert und als Werteobjekte bereitgestellt.
- **Modbus**: Lese und Schreibpfade via mbpoll für RTU und TCP; optionaler **Read Back** bestätigt Schreibaktionen.
- **Persistenz**: Messwerte landen in **PostgreSQL** als Zeitreihen, Metriken und Feldauswahlen werden konsistent verwaltet.
- **Sicherheit & Betrieb**: Fernzugriff via VPN, Deploy als systemd Dienst, Health Endpunkt für Smoke Tests.

Verweise: Die Übersicht siehst du in **Abbildung A1**, der M-Bus Ablauf in **Abbildung A3**, Modbus in **Abbildung A4**. Für Kapitel 8.2 kannst du zusätzlich **Abbildung 14: Walking Skeleton – End to End Durchstich** direkt unter den Text setzen (die PlantUML hast du von mir schon).

Risiko	Sichtbar gemacht durch	Entschärfung im Prototyp
Serielle Kollisionen (M-Bus Scan blockiert)	SSE-Scan mit Abbruchtest	Queue pro Port, sauberer SIGINT, Mindestabstände
Parser Unsicherheiten (XML, Register)	Einzelgerät Auslesung und Modbus Read	Normalisierungsschicht, feste JSON Envelopes, Tests
Latenzen/Timeouts	End to End Laufzeiten in UI/Logs	Timeout-Parameter in .env, Retries, Caching im Monitor

Schreibzugriffe riskant	Modbus Write mit Read Back	Bestätigung im UI, Logging in DB
Persistenz unstabil	Speicherung von Samples/Metriken	Schemata fixiert, DB Tests, Health Check
Betrieb unsicher	Deploy als systemd, Health	Restart Policy, Journal-Prüfung, VPN nur für Fernzugriff

Tabelle 12 Risiken und Entschärfung durch Walking Skeleton

8.3 Implementierung Backend

8.3.1 M-Bus Scanning & Parsing

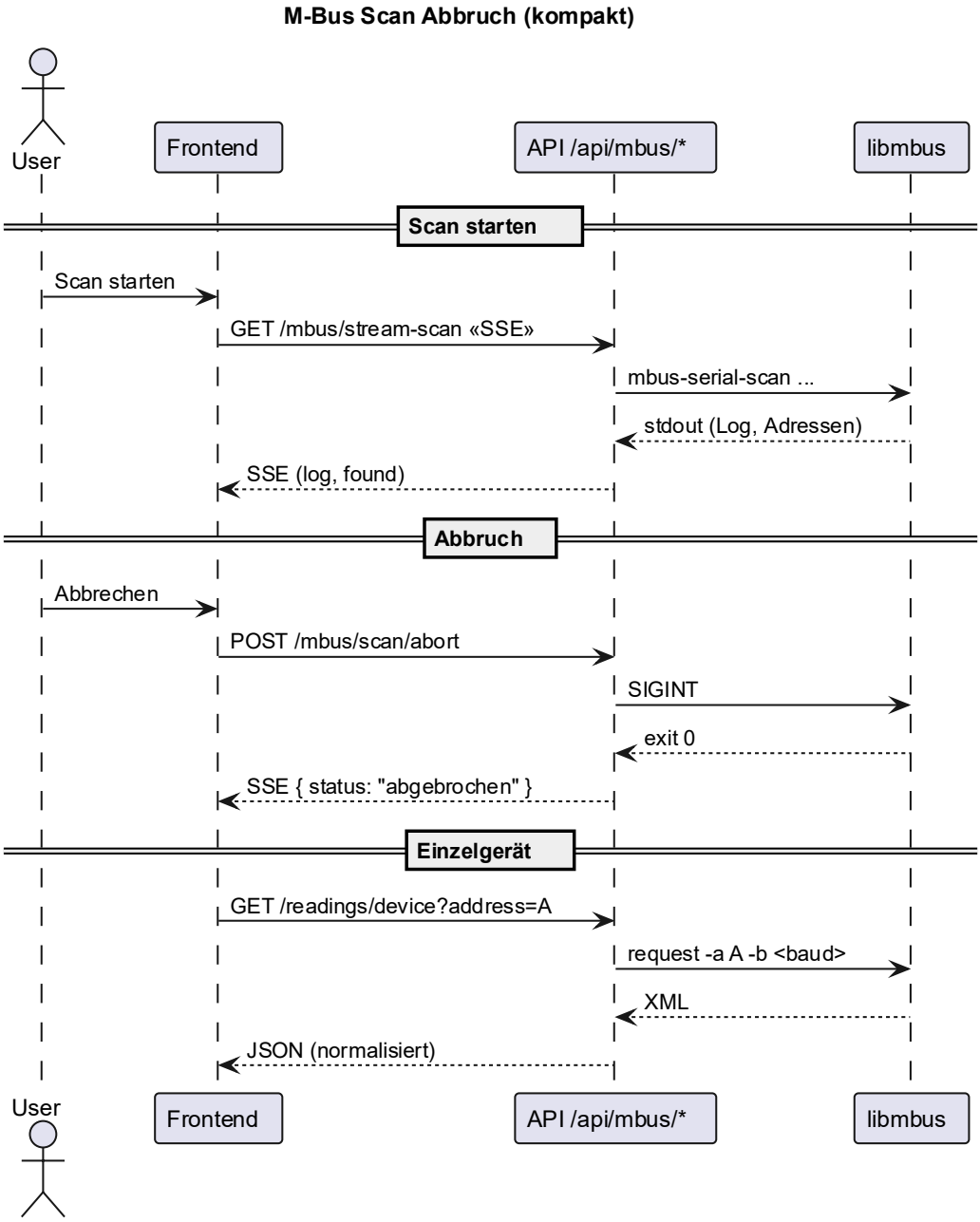


Abbildung 14 M-Bus Scan Abbruch (kompakte Sequenz)

Kurzbeschreibung

Der Scan wird als Server Sent Events an das Frontend gestreamt. Ein Abbruch signalisiert libmbus einen kontrollierten Stopp (SIGINT) und beendet den Stream mit einem expliziten Status. Einzelgeräte werden on demand gelesen, die XML Antwort wird zu strukturierten Schlüsseln normalisiert und optional persistiert.

Aspekt	Umsetzung
CLI Aufrufe	mbus-serial-scan für Laufzeit Scan, mbus-serial-request -a <addr> für Einzelgerät
Endpunkte	GET /api/mbus/stream-scan (SSE), POST /api/mbus/scan/abort, GET /api/readings/device?address=…
Datenablage	bekannte Adressen und optionale Namen in known_mbus_devices.json; pro Adresse Feldauswahl in config_<address>.json; Messwerte in samples der Datenbank
Normalisierung	XML der libmbus Ausgabe wird zu Paaren Tag → Wert abgebildet (nur ausgewählte Felder, wenn konfiguriert)
Zeitverhalten	Scan als Stream, Einzelgerät synchron; Timeout und Baudrate über Umgebungsvariablen gesteuert
Fehlerbilder	Timeouts ohne Antwort, gestörte Leitung, doppelte Adressen; sauberer Abbruch verhindert hängende Prozesse

Tabelle 13 M-Bus Backend Komponenten und Pfade

Feld	Typ	Bedeutung
type	Text	log, found, status
payload	Text/Objekt	Logzeile oder gefundene Adresse
status	Text	optional, etwa abgebrochen, fertig

Tabelle 14 SSE Ereignisse im Scan

Zentrale Umgebungsvariablen (Auszug)

MBUS_SERIAL_PORT, MBUS_BAUDRATE, MBUS_SCAN_TIMEOUT, MBUS_CMD_TIMEOUT

8.3.2 Modbus Handling (RTU/TCP)

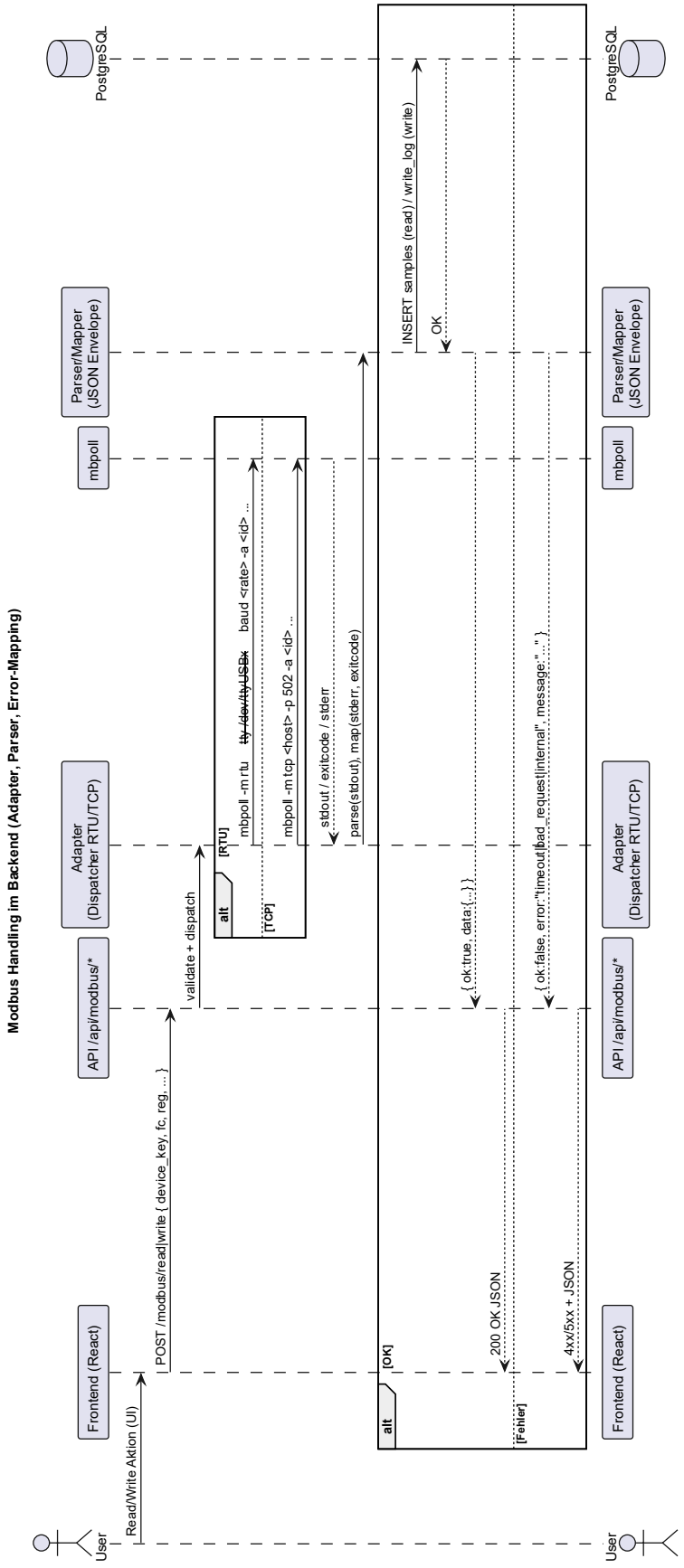


Abbildung 15 Modbus Handling im Backend (Adapter, Parser, Error-Mapping)

Kurzbeschreibung

Das Backend validiert Request-Parameter und wählt anhand des Gerätetyps den **RTU-** oder **TCP-Pfad**. mbpoll liefert Rohwerte bzw. Exitcodes; ein Parser normalisiert die Ausgabe und mappt Fehler konsistent in den **JSON-Envelope**. Erfolgreiche Reads werden als **Samples** persistiert, Writes als **write_log** dokumentiert; optionaler **Read-Back** bestätigt Schreibvorgänge.

FC	Bezeichnung	Zweck	Typische Nutzung
1	Read Coils	digitale Ausgänge lesen	DO Status prüfen
2	Read Discrete Inputs	digitale Eingänge lesen	DI von Sensoren
3	Read Holding Registers	schreibbare Register lesen	Sollwerte, Konfig
4	Read Input Registers	nur lesbare Register lesen	Messwerte, Status
5	Write Single Coil	einzelnen Ausgang setzen	DO schalten
6	Write Single Register	einzelnes Holding-Register schreiben	Setpoint, Modus
16	Write Multiple Registers	mehrere Register schreiben	Batch-Updates

Tabelle 15 Modbus Funktionscodes (Auszug)

Kategorie	RTU	TCP
Transport	RS485 Halbduplex	Ethernet IPv4
Ziel	--tty /dev/ttyUSBx	<host> -p 502
Adresse	-a <slave_id>	-a <slave_id>
Timing	--baud <rate>, Parity, Daten-/Stopbits	Netzwerklatenz, Port
Lesen	-r <reg> -c <count> -m rtu	-r <reg> -c <count> -m tcp
Schreiben	`--fc 5	6
Häufige Fehler	Verdrahtung A/B, Terminierung, Parity	Host unreachable, Port blockiert

Tabelle 16 Modbus Parameter – RTU vs. TCP

Zentrale Umgebungsvariablen (Auszug)

MODBUS_CMD_TIMEOUT, MODBUS_READ_DELAY_MS, MODBUS_SERIAL_PORT, MODBUS_BAUDRATE

8.3.3 Schnittstellen

Bereich	Methode	Pfad	Query/Body (Kurz)	Rückgabe (Kurz)	Hinweise
M-Bus	GET	/api/mbus/stream-scan	–	SSE Events `log	found
M-Bus	POST	/api/mbus/scan/abort	–	{ ok }	Beendet laufenden Scan (SIGINT)

Readings	GET	/api/readings/values	–	[{ address, name? }]	Quelle: known_mbus_devices.json
Readings	GET	/api/readings/device	address	{ data: … }	Einzelgerät (M-Bus) auslesen
Readings	POST	/api/readings/label	{ address, name }	{ ok }	Gerätename setzen
Readings	GET	/api/readings/config/{address}	–	{ fields: [..] }	Ausgewählte XML-Tags pro Gerät
Modbus	POST	/api/modbus/read	{ device_key, fc, register, count }	{ ok, data:[..] }	FC 1/2/3/4
Modbus	POST	/api/modbus/write	{ device_key, fc, register, value }	{ ok, readback? }	FC 5/6/16, optional Read-Back
Modbus Mapping	GET	/api/modbus-mappings-row/get	device_key,row_id	{ data:{ enabled, mapping } }	Zeilenmapping lesen
Modbus Mapping	POST	/api/modbus-mappings-row/save	{ device_key,row_id,enabled,mapping }	{ ok }	Zeilenmapping speichern
Monitoring	GET	/api/monitor/read-all	–	[{ device, ts, value, source }]	Cooldown/Cache beachtet
Metrics	GET	/api/metrics/list	–	{ ok, data:[{ id, tag, unit, source }] }	Metrik-Metadaten
Metrics	GET	/api/metrics/series	metric_id,range,bucket,as_rate	{ ok, data:[{ ts, avg }] }	Aggregierte Zeitreihe

Tabelle 17 REST-API Endpunkte (Auszug)

Zweck	cURL
M-Bus Scan (SSE)	curl -N http://localhost:5000/api/mbus/stream-scan
M-Bus Abbruch	curl -X POST http://localhost:5000/api/mbus/scan/abort
M-Bus Einzelgerät	curl "http://localhost:5000/api/readings/device?address=10"
Label setzen	curl -X POST -H "Content-Type: application/json" -d '{"address":10,"name":"Wärmezähler EG"}' http://localhost:5000/api/readings/label
Modbus Read	curl -X POST -H "Content-Type: application/json" -d '{"device_key":"io1","fc":3,"register":2001,"count":2}' http://localhost:5000/api/modbus/read
Modbus Write	curl -X POST -H "Content-Type: application/json" -d '{"device_key":"io1","fc":5,"register":1,"value":1}' http://localhost:5000/api/modbus/write
Monitor	curl http://localhost:5000/api/monitor/read-all
Metrics Series	curl "http://localhost:5000/api/metrics/series?metric_id=oil_day&range=7d&bucket=1h&as_rate=true"

Tabelle 18 Beispiel-Requests (cURL, kompakt)

Situation	HTTP	Body (Beispiel)	UI-Hinweis
Erfolg	200	{ "ok": true, "data": [...] }	Daten anzeigen
Ungültige Parameter	400	{ "ok": false, "error": "bad_request", "message": "register missing" }	Validierung markieren
Nicht gefunden	404	{ "ok": false, "error": "not_found", "message": "device unknown" }	Hinweis anzeigen
Timeout	408	{ "ok": false, "error": "timeout", "message": "no response" }	Retry anbieten
Intern	500	{ "ok": false, "error": "internal", "message": "unexpected error" }	Support/Log-ID zeigen

Tabelle 19 JSON-Envelope und Fehlerfälle

SSE-Ereignisse (Scan, kompaktes Schema)

```
// type: "log" | "found" | "status"
{ "type": "log", "payload": "scanning address 5..." }
{ "type": "found", "payload": { "address": 10 } }
{ "type": "status", "payload": "abgebrochen" }
```

Auth & CORS (optional)

- Optionaler Bearer Token via Keycloak (Axios-Interceptor setzt Authorization: Bearer <token>).
- CORS per .env konfigurierbar; bei * keine Credentials über Browser.

8.4 Implementierung Frontend (React UI, Tabellen, Graphen)

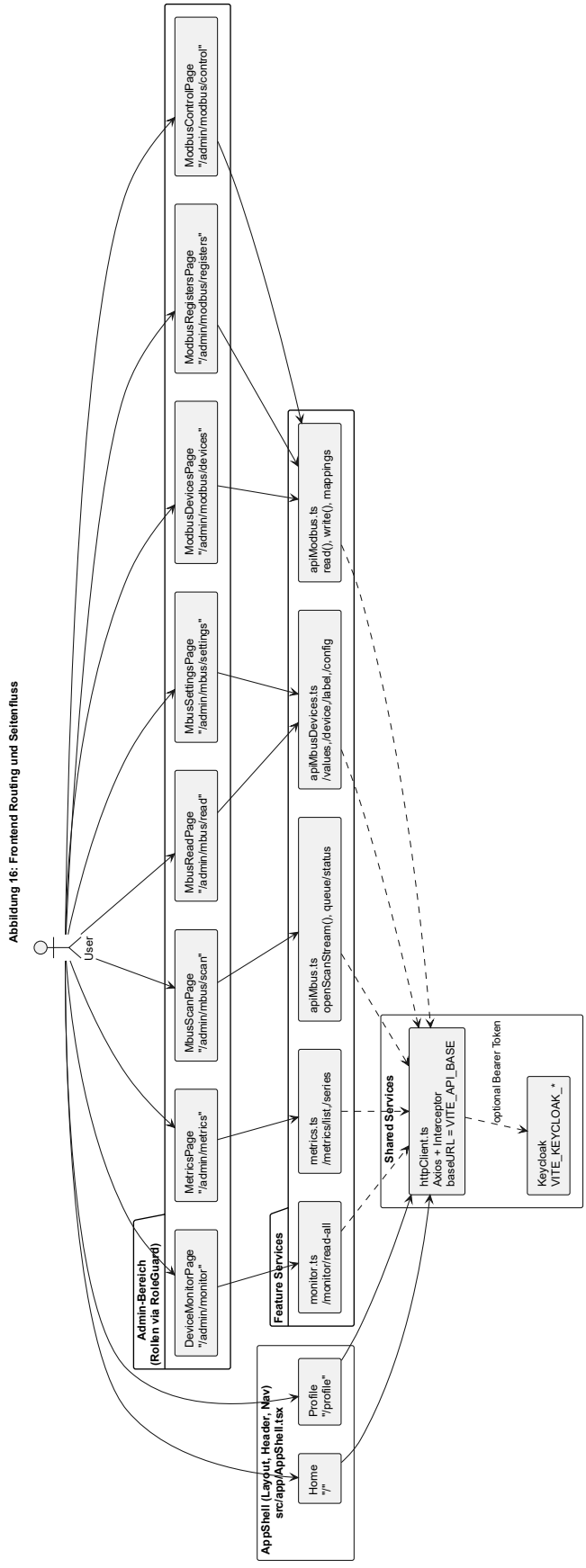


Abbildung 16 Frontend Routing und Seitenfluss

Kurzbeschreibung

Die SPA nutzt **AppShell** für Layout und Routing. Admin-Seiten sind hinter einem **RoleGuard**. API-Zugriffe laufen über einen zentralen Axios-Client (httpClient.ts) mit baseURL = VITE_API_BASE. Bei aktivem Keycloak setzt der Interceptor automatisch den **Bearer Token**. M-Bus Scan verwendet **SSE** über openScanStream(), Charts holen aggregierte Zeitreihen über /metrics/series.

Seite (Route)	Service	Endpunkte
Home (/)	httpClient	Healthcheck optional
Profile (/profile)	–	–
DeviceMonitor (/admin/monitor)	monitor.ts	GET /api/monitor/read-all
Metrics (/admin/metrics)	metrics.ts	GET /api/metrics/list, GET /api/metrics/series
M-Bus Scan (/admin/mbus/scan)	apiMbus.ts	GET /api/mbus/stream-scan (SSE), POST /api/mbus/scan/abort
M-Bus Read (/admin/mbus/read)	apiMbusDevices.ts	GET /api/readings/values, GET /api/readings/device?address=…
M-Bus Settings (/admin/mbus/settings)	apiMbusDevices.ts	GET /api/readings/config/{address}, POST /api/readings/label
Modbus Devices (/admin/modbus/devices)	apiModbus.ts	GET/POST /api/modbus-* je nach Umsetzung
Modbus Registers (/admin/modbus/registers)	apiModbus.ts	POST /api/modbus/read, POST /api/modbus/write
Modbus Control (/admin/modbus/control)	apiModbus.ts	POST /api/modbus/write

Tabelle 20 Seiten ↔ Services ↔ Endpunkte

State	Auslöser	Sichtbare Elemente	Aktionen
idle	Seite geöffnet	Start-Button aktiv, Log leer	Scan starten
scanning	Start geklickt	Live-Log (SSE), „Abbrechen“ aktiv	Abbrechen möglich
aborting	Abbrechen geklickt	Hinweis „Abbruch läuft…“	Warten auf Status
aborted	SSE Status „abgebrochen“	Meldung, Liste letzter Funde	Erneut scannen
complete	Stream endet normal	„Scan beendet“, Liste der Adressen	Detail öffnen, benennen

Tabelle 21 UI-Zustände für M-Bus Scan

Parameter	Werte	Bedeutung
metric_id	z. B. oil_day	Signal-ID
range	24h, 7d, 30d	Zeitraum
bucket	5m, 1h, 1d	Aggregationsintervall
as_rate	true/false	Rate statt Mittelwert

Tabelle 22 Charts – Zeitreihenabruf

Frontend-Implementierungsnotizen

- **Charts**: react-chartjs-2 + chart.js mit chartjs-adapter-date-fns. Tooltips und Zeitachse auf ISO-Zeitformate prüfen.
- **Fehlerhandling**: Einheitlicher JSON-Envelope. UI zeigt freundliche Meldungen bei bad_request, timeout, internal.
- **Responsivität**: Grids für Tabellen/Charts, keine fixe Breite; Seiten auf 1280px und mobil testen.
- **Zugriffsschutz**: Admin-Routen nur mit RoleGuard. Bei fehlender Rolle auf /unauthorized leiten.
- **Env**: VITE_API_BASE, optional VITE_KEYCLOAK_URL|REALM|CLIENT.

8.5 Sicherheitsaspekte & Benutzerrollen

8.5.1 Zielbild und Leitplanken

- **Lokalbetrieb** mit kontrolliertem Fernzugriff über **VPN** (Tailscale).
- **Rollenbasiertes Arbeiten**: Lesen getrennt von Schreiben und Konfigurieren.
- **Kleine, harte Oberflaeche**: nur die benoetigten Ports und Rechte.
- **Konsistenter Fehler-Envelope** statt Rohfehler, damit die UI gezielt reagieren kann.
- **Sichere Defaults**: Timeouts, Queues, saubere Prozessbeendigung fuer CLI-Tools.

Verweise: Hardware und Netzwerkgrundlagen siehe **Abbildung A2**. QS

8.5.2 Rollen und Berechtigungen

Aktion	Endanwender	Verwalter	Techniker
Verbrauch und Status ansehen (Tabellen, Charts)	✓	✓	✓
M-Bus Scan starten, abbrechen, Geräte lesen	✗	✗	✓
Modbus Register lesen	✗	✗	✓
Modbus schreiben (Coils, Register)	✗	✗	✓
Geräte benennen, Feldauswahl setzen	✗	✓	✓
Monitoring ansehen (aggregiert)	✓	✓	✓
Systemkonfiguration, .env, Deploy	✗	✗	✓ (Ops)

Tabelle 23 Rollenmatrix

Server prüft die Rolle **immer**. UI-Ausblendung allein genügt nicht.

8.5.3 Zugriff, Authentisierung, Autorisierung

- **VPN als Primärzugang**: Zugriff von aussen nur über Tailscale. ACLs in der Tailnet-Policy auf definierte Geräte und Benutzer begrenzen.
- **Keycloak (optional)**: Falls aktiviert, erhält die UI ein **Barer Token**. Der Axios-Interceptor fügt Autorisation: Barer <Token> hinzu. Backend prüft Rollen pro Endpunkt.
- **CORS**: Feste CORS_ORIGINS in .env. Kein Wildcard bei Credentials. Bei reinem Lokalbetrieb CORS eng halten oder Frontend vom selben Host liefern.
- **SSE**: Der Scan-Stream funktioniert ohne Cookies. Nur freigeben, wenn Herkunft geprüft ist.

8.5.4 Eingaben, Prozesse, Fehler

- **Parametervalidierung**: Register, Anzahl, Adresse, Baudrate, TTY streng prüfen. Nur bekannte device_key zulassen.
- **CLI-Aufrufe sicher**: Argumente nie konkatenieren, sondern sauber aufbauen. Keine Shell-Interpolation.
- **Prozesskontrolle**: Scan-Abbruch per **SIGINT**. Keine Zombie-Prozesse. Timeouts für alle CLI-Calls.
- **Fehler-Envelope**: Einheitlich { ok:false, error:"bad_request|timeout|internal", message:"..." }. Keine Roh-stderr an die UI.

8.5.5 Betriebssicherheit und Härtung

- **systemd**: eigener Service-User, automatischer Neustart, Ressourcenlimits.
- **Updates**: **unattended-upgrades** aktivieren.
- **Netzwerk**: nur die benötigten Ports. Bei Bedarf Reverse Proxy mit TLS.
- **Logs**: journald Rotation. Keine sensiblen Daten loggen.
- **Datenbank**: eigene DB-Rolle mit minimalen Rechten, regelmässiges Backup.

Beispiel: Auszug harte systemd-Unit (in Anhang G2 vollständig ablegen)

```
[Service]
User=visual
Group=visual
EnvironmentFile=/etc/visualisation/.env
ExecStart=/usr/bin/python /opt/visualisation/current/run.py
Restart=on-failure
RestartSec=3
# Härtung
NoNewPrivileges=yes
PrivateTmp=yes
ProtectSystem=full
ProtectHome=true
AmbientCapabilities=
# Ressourcen (Beispielwerte)
MemoryMax=300M
TasksMax=150

```

8.5.6 Daten und Schutz

- **.env** ausserhalb des Repos, nur für den Service-User lesbar.
- **Backups** der Datenbank regelmässig testen.
- **Datenminimierung**: nur benötigte Messpunkte speichern, PII vermeiden.

8.5.7 Zusammenfassung für den Leser

- Zugriff von aussen ausschliesslich über **VPN**.
- Schreibaktionen **nur** für die Rolle **Techniker**.
- **CORS**, **Timeouts**, **Queues**, **Fehler-Envelope** sind aktiv.
- **systemd** härtet die Ausführung, **unattended-upgrades** halten das System aktuell.
- Nachweise im Anhang: CI-Checks, Coverage, Deploy-Status, Health.

9 Verifikation, Tests und Qualitätssicherung

Abbildung 17: Testpyramide und Pipeline Gates

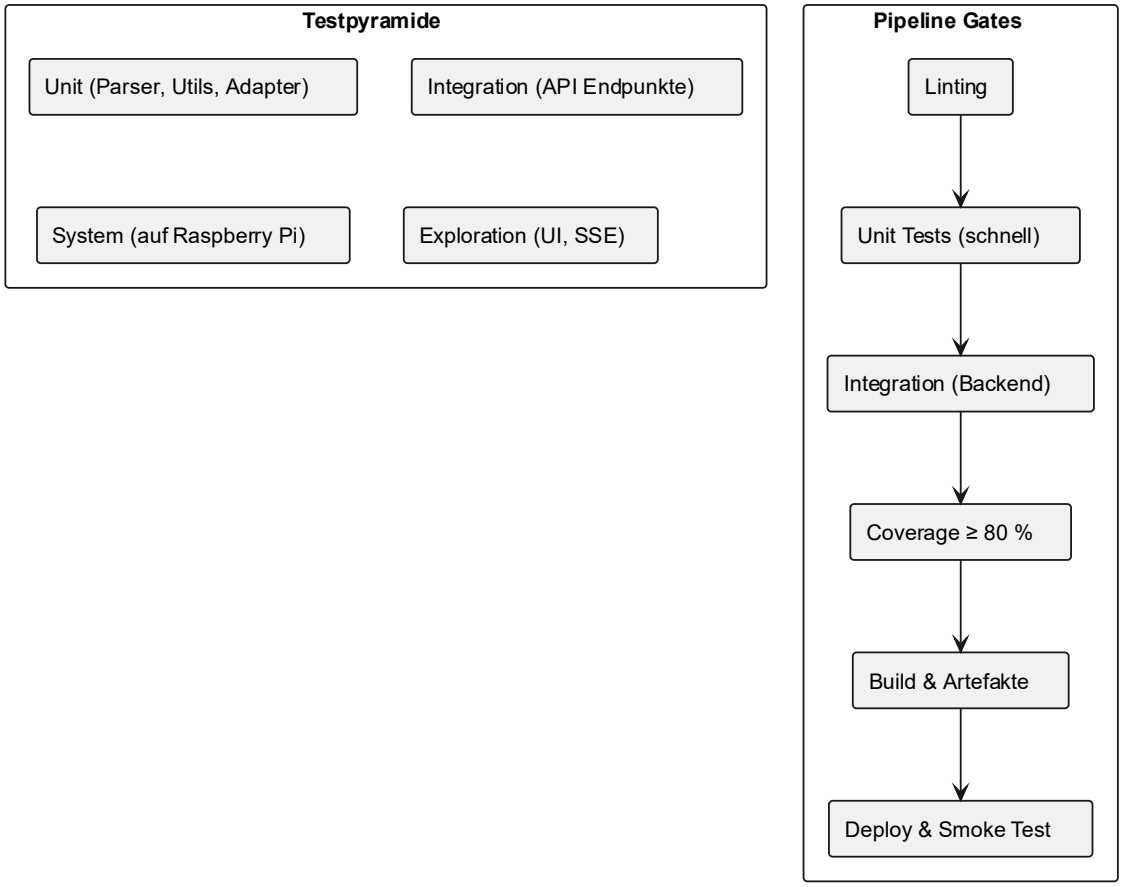


Abbildung 17 Testpyramide und Pipeline Gates

9.1 Teststrategie

Ziel ist ein belastbarer Nachweis der funktionalen Korrektheit, der Robustheit unter realen Bedingungen und der Wartbarkeit. Die Tests folgen einer klaren Pyramide: viele **Unit Tests**, ausgewählte **Integrationstests**, wenige **Systemtests** auf dem Raspberry, ergänzt durch **explorative UI Tests** und Screencasts.

Testart	Fokus	Ziel	Beispiel
Unit	Parser, Utils, Fehlerpfade	Korrekte Logik bei kleinstem Umfang	XML nach JSON, Modbus Decoder
Integration	API mit Stub für CLI/DB	Konsistenter JSON Envelope, Statuscodes	`/api/mbus/scan`, `/api/modbus/read`
System	Reale Hardware	Timeouts, Abbruch, Kollisionsfreiheit	M-Bus Scan, Modbus RTU/TCP
Exploration	UI Flows	Nutzererlebnis, Ladezustände	Scan starten, Abbruch, Chart laden

Tabelle 24 Testarten und Ziele

9.2 Testumgebungen

Umgebung	Komponenten	Zweck
Lokal Dev	Python venv, pytest, Postgres lokal	schneller Zyklus, Unit/Integration
Raspberry Pi	Pi OS, libmbus, mbpoll, Postgres	reale Feldbedingungen
CI	Bitbucket Pipelines, Postgres Service	Gate für Merge, Artefakte erzeugen

Tabelle 25 Testumgebungen

9.3 Testdurchführung (lokal & CI)

Modus	Kommando	Erwartung
Schnell lokal	`PYTHONPATH=. pytest -q --maxfail=1 -m "not db and not slow"`	rasches Feedback, nur schnelle Tests
Voll lokal	`PYTHONPATH=. pytest --cov=backend --cov-report=term --cov-report=html --cov-fail-under=80`	Coverage ≥ 80 %, htmlcov generiert
CI PR	Pipeline Schritt „lint + unit“	schnelle Checks, Block bei Fehler
CI main/Tag	Voller Lauf mit Postgres	Artefakte (htmlcov, Release), grünes Gate

Tabelle 26 Durchführungsplan und Kommandos

9.4 Ergebnisse (Kurzüberblick)

Bereich	Kennzahl	Ergebnis	Nachweis
Coverage gesamt	≥ 80 %	[Wert eintragen]	Abbildung B3/B4
API Stabilität	2xx/4xx/5xx Verhältnis	[Wert eintragen]	Logauszug B2
Scan Laufzeit	M-Bus Scan 0..n	[Zeitfenster]	Screencast V2/V3
Write Verifikation	Read-Back Quote	[Wert eintragen]	Log + UI Screen B2
Deploy	Active Running	Ja	

Tabelle 27 Schlüsselergebnisse (Beispielstruktur zum Befüllen)

9.5 Praktischer Frontend-Nachweis (Screencast)

ID	Inhalt	Muss sichtbar sein	Kriterium bestanden
V1	Endanwenderfluss	Scan starten, Adressenliste, Detail, Chart	UI reagiert ohne Fehler
V2	Technikerfluss	Modbus Read/Write, Monitor Update	Read Back ok, kein UI Fehler
V3	Fehlerfälle	Abbruch, Timeout, bad request	UI zeigt konsistente Meldungen

Tabelle 28 Screencasts und Akzeptanzkriterien

9.6 Qualitätssicherung

Massnahme	Tool	Gate
Linting Backend	ruff/flake8	PR Pflicht
Tests Backend	pytest	PR und main
Coverage Gate	pytest-cov	`--cov-fail-under=80`
Review Pflicht	PR Reviews	Merge Blocker
Artefakte	htmlcov, Release Bundle	main/Tag
Deploy Check	systemd Status, Health	nach Rollout

Tabelle 29 QS Massnahmen

9.7 Risiko- und Edge-Case-Tests

Risiko	Testfall	Erwartung	Mitigation
Scan Kollision	Scan + Abort	Stream endet mit Status „abgebrochen“	SIGINT, Queue
RTU Leitungsfehler	A/B vertauscht	Timeout Fehler-Envelope	Verdrahtung prüfen
TCP Host down	Read auf offline Gerät	„host unreachable“ → 408/500	Retry, Logging
Ungültige Parameter	fehlendes `register`	400 bad request	API Validierung

Datenbank nicht bereit	Insert Samples	Retry/Fail fast, sauberer 500	DB Wait in CI

Tabelle 30 Edge Cases und Mitigation

9.8 Artefakte und Nachweise

Artefakt	Speicherort (Anhang)	Hinweis
PR Statuschecks	B5	grüner Haken sichtbar
Coverage Terminal	B6	TOTAL Zeile, Schwelle erreicht
Coverage HTML Übersicht	B3	index.html mit Gesamtwert
Coverage HTML Detail	B4	Modulansicht

Tabelle 31 Artefakte und Verweise

10 DevOps / CI/CD, Build & Deployment

10.1 Versionskontrolle (GitHub/Bitbucket)

Regel	Beschreibung
Branching	`main` stabil; Featurearbeit auf `feature/*`; Hotfix auf `hotfix/*`
Pull Requests	Reviewpflicht, grüne CI-Checks vor Merge
Commit Style	Kurz, imperativ; Referenz auf Ticket (z. B. JIRA-Key)
Tags	Semantisch `vX.Y.Z` auf `main` nach grünem Build
Schutz	Kein Direkt-Push auf `main`; Merge nur via PR

Tabelle 32 Branch- und Release-Policy

Referenzen im Anhang: PR-Checks und Tags siehe B5

10.2 CI/CD-Pipeline (Automatisierung, Testauszüge)

Abbildung 18: CI/CD - PR Fastlane vs. main/tag Volllauf

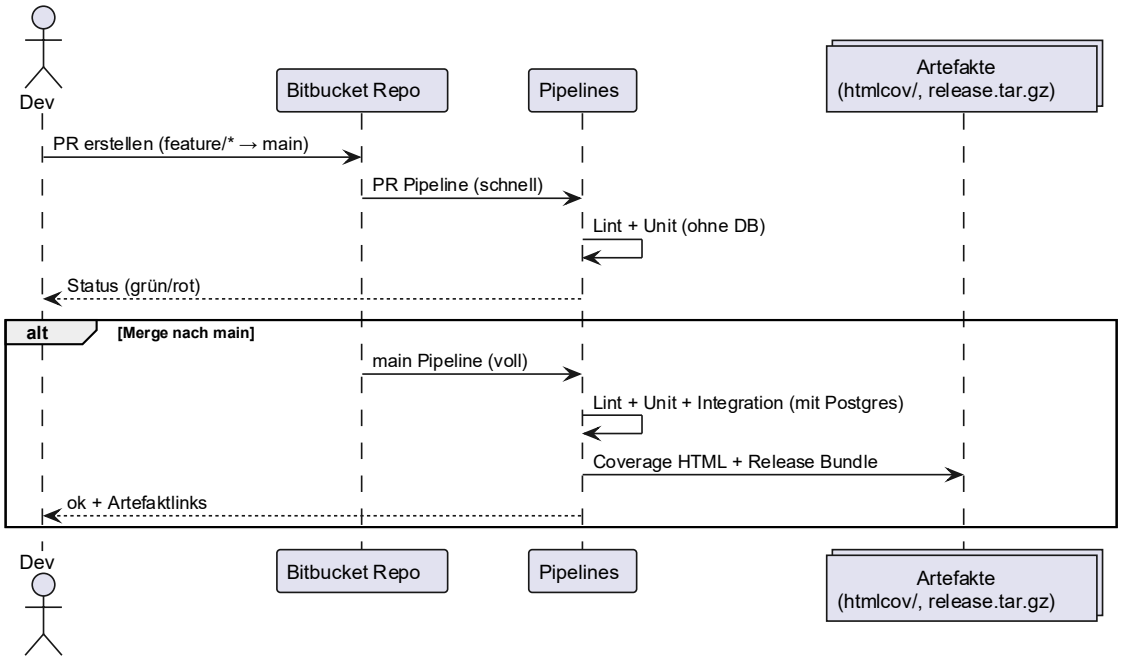


Abbildung 18 CI/CD - PR Fastlane vs. main/tag Volllauf

Schritt	PR	main/tag	Zweck
Lint	✓	✓	Stil, schnelle Fehler
Unit	✓	✓	Parser, Utils
Integration (mit DB)	–	✓	API, DAO
Coverage Gate ≥ 80 %	–	✓	Qualitätsgrenze

Build/Artefakte	–	✓	htmlcov/, release.tar.gz

Tabelle 33 Pipeline-Schritte (Standard)

Referenzen im Anhang: Coverage B3–B4, Terminal-Report B6.

10.3 Deployment auf Raspberry Pi (git + setup.sh)

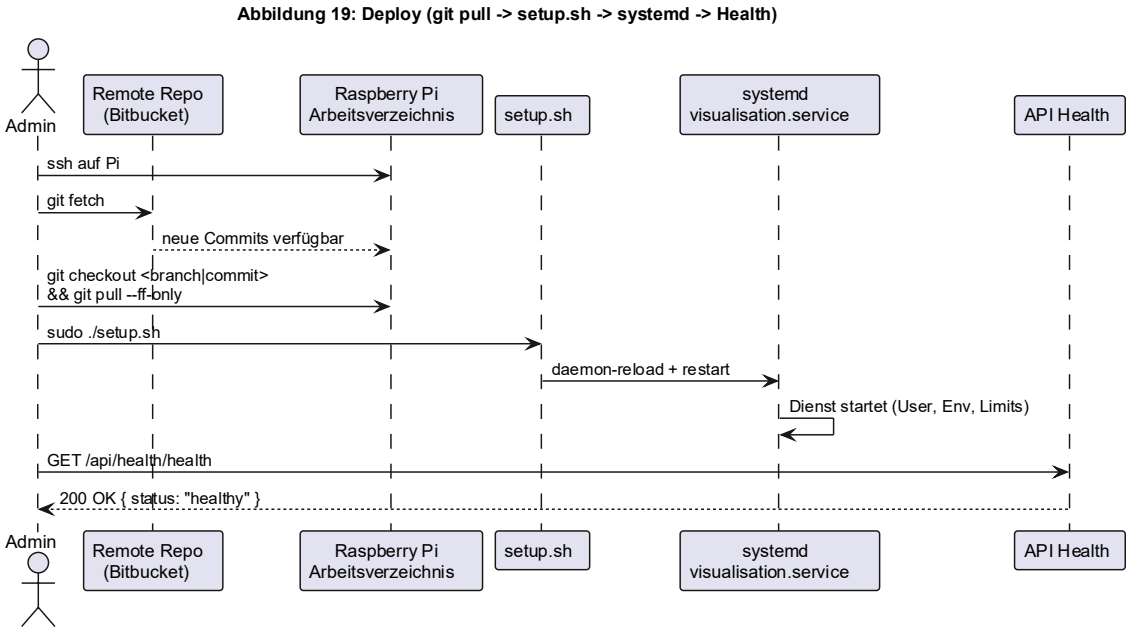


Abbildung 19 Deploy (git pull -> setup.sh -> systemd -> Health)

Ablauf in Kürze

1. per ssh auf den Pi
2. git fetch && git checkout <branch|commit> && git pull --ff-only
3. chmod +x setup.sh && sudo ./setup.sh
4. systemctl status visualisation.service prüfen
5. curl -f http://localhost:5000/api/health/health checken

Schritt	Aufgabe	Beispiel
A	Systempakete installieren	apt install python3-venv postgresql libmbus mbpoll msmtp
B	venv und Python Abhängigkeiten	python3 -m venv ... && pip install -r requirements.txt
C	Verzeichnisse anlegen	/opt/visualisation/{data,logs} falls nicht vorhanden
D	Konfiguration prüfen	.env unter /etc/visualisation/.env vorhanden und lesbar
E	systemd Unit bereitstellen	visualisation.service nach /etc/systemd/system/
F	Dienst neu laden und starten	systemctl daemon-reload && systemctl restart visualisation.service

G	Gruppenrechte seriell	Service-User in dialout und uucp

Tabelle 34 Aufgabenübersicht setup.sh (ersetzt)

Check	Befehl	Erwartung
Dienststatus	systemctl status visualisation.service --no-pager	Active: active (running)
Health	curl -f http://localhost:5000/api/health/health	HTTP 200 JSON
Logs	journalctl -u visualisation.service -n 50 --no-pager	keine Fehler, Port gebunden
Serielle Geräte	ls -l /dev/ttyUSB*	Adapter sichtbar
Gruppen	id <service-user>	Mitgliedschaft dialout,uucp

Tabelle 35 Post-Install Checks

Pfad	Inhalt
/opt/visualisation	Arbeitsverzeichnis Repo
/opt/visualisation/data/logs	Laufzeitlogs
/etc/visualisation/.env	Umgebungsvariablen
/etc/systemd/system/visualisation.service	Dienstdefinition

Tabelle 36 Standardpfade

Ziel	Befehl
auf frühere Version wechseln	git checkout <commit-oder-tag>
Abhängigkeiten aktualisieren	source venv/bin/activate && pip install -r requirements.txt
Dienst neu starten	sudo systemctl restart visualisation.service
Health prüfen	curl -f http://localhost:5000/api/health/health

Tabelle 37 Version zurückdrehen via Git

Das ist euer **Rollback**: schlicht **git checkout** auf einen bekannten Commit oder Tag, dann neu starten. Keine Release Bundles, kein Symlink.

Symptom	Ursache	Lösung
Dienst startet nicht	.env fehlt/fehlerhaft	.env korrigieren, daemon-reload, Restart
Health 500	DB nicht erreichbar	DB_URL, Postgres Service prüfen
Port belegt	Prozess auf 5000	sudo fuser -n tcp 5000, Prozess beenden
M-Bus Timeout	Port/Baud falsch	MBUS_* in .env prüfen, Verkabelung
Modbus TCP Timeout	Host/Port blockiert	IP/Port 502 prüfen, Netzwerk

SSE reisst ab	Proxy/CORS	CORS korrekt setzen, notfalls ohne Proxy testen

Tabelle 38 Troubleshooting

10.4 Installations- und Benutzerhandbuch

10.4.1 10.4.1 Voraussetzungen

Komponente	Minimum	Hinweis
OS	Raspberry Pi OS aktuell	apt update && apt upgrade vor Start
Python	3.10 oder neuer	python3 --version
Pakete	python3-venv, libmbus, mbpoll, postgresql, msmtp	werden sonst über setup.sh installiert
Rechte	sudo auf dem Pi	für Paketinstallation und systemd
Ports	API: 5000 (Standard)	anpassbar via .env
Serielle Geräte	/dev/ttyUSB* sichtbar	M-Bus Pegelwandler, RS485 Adapter
Gruppen	dialout, uucp	Service-User muss Mitglied sein

Tabelle 39 Systemvoraussetzungen

10.4.2 Erstinstallation (Schritt für Schritt)

```
# 1) Code auf den Pi holen (oder Repo aktualisieren)
cd /opt/visualisation
git fetch
git checkout <branch-oder-commit>
git pull --ff-only

# 2) Setup ausführen
chmod +x setup.sh
sudo ./setup.sh

```

Das Skript installiert Systempakete, erstellt die venv, installiert Python-Dependencies, legt Pfade an, installiert/aktualisiert die systemd-Unit und startet den Dienst.

10.4.3 Erstkonfiguration (.env)

Key	Beispiel	Zweck
FLASK_RUN_PORT	5000	API-Port
DB_URL	postgresql://metrics:metrics@localhost:5432/metrics	Datenbank
CORS_ORIGINS	http://localhost:5173	Frontend Zugriff
MBUS_SERIAL_PORT	/dev/ttyUSB0	M-Bus Port
MBUS_BAUDRATE	2400	M-Bus Baud
MODBUS_SERIAL_PORT	/dev/ttyUSB1	RTU Port

MODBUS_BAUDRATE	19200	RTU Baud
MODBUS_CMD_TIMEOUT	3000	ms Timeout für CLI
MODBUS_READ_DELAY_MS	250	Mindestabstand Reads

Tabelle 40 Minimaler .env-Satz

Ablage: /etc/visualisation/.env (nur für den Service-User lesbar).

10.4.4 Dienstverwaltung (systemd)

```
# Start/Status/Logs
sudo systemctl daemon-reload
sudo systemctl enable --now visualisation.service
systemctl status visualisation.service --no-pager
journalctl -u visualisation.service -n 50 --no-pager

# Neu starten nach Änderungen
sudo systemctl restart visualisation.service

```

10.4.5 Funktionstest (Smoke)

```
# Health
curl -f http://localhost:5000/api/health/health

# M-Bus: Port sichtbar?
ls -l /dev/ttyUSB*

# Modbus TCP: Beispiel Read (anpassen)
# mbpoll -m tcp <ip> -p 502 -a 1 -r 1 -c 1

# Modbus RTU: Beispiel Read (anpassen)
# mbpoll -m rtu --tty /dev/ttyUSB1 --baud 19200 -a 1 -r 1 -c 1

```

10.4.6 Benutzerhandbuch (Kurz)

Rolle	Aufgabe	Pfad in der UI
Endanwender	Verbrauch und Status ansehen (Tabellen/Charts)	Home, Metrics
Verwalter	Geräte benennen, Felder auswählen	M-Bus Settings
Techniker	Scan starten/abbrechen, Einzelgerät auslesen	M-Bus Scan/Read
Techniker	Modbus Register lesen/schreiben	Modbus Registers/Control
Techniker	Monitoring prüfen	Device Monitor

Tabelle 41 Typische Aufgaben je Rolle

10.4.7 Update und Rücksprung (git)

Kein Release-Bundle, nur Git.

```
# Update
cd /opt/visualisation
git fetch
git checkout <branch-oder-commit>
git pull --ff-only
sudo ./setup.sh
sudo systemctl restart visualisation.service

# Rücksprung (Rollback via Git)
git checkout <bekannter-commit>
sudo ./setup.sh
sudo systemctl restart visualisation.service
```

10.4.8 Deinstallation (vollständig)

```
sudo systemctl disable --now visualisation.service
sudo rm -f /etc/systemd/system/visualisation.service
sudo systemctl daemon-reload
sudo rm -rf /opt/visualisation
sudo rm -rf /etc/visualisation
# Optional: Postgres Daten/Benutzer entfernen (nur wenn sicher!)
```

10.4.9 Tipps & Stolpersteine

- **Serielle Rechte**: Service-User in dialout und uucp. Danach neu anmelden oder Dienst neu starten.
- **TTY Fix**: Udev-Regel für stabile Gerätenamen optional, sonst prüfen, ob Adapter die Ports tauschen.
- **Port belegt**: sudo fuser -n tcp 5000 zeigt den Blocker.
- **CORS**: Wildcard * funktioniert nicht mit Credentials. Konkrete Origins setzen, wenn Keycloak aktiv ist.
- **Timeouts**: MBUS_* und MODBUS_* in .env feinjustieren, besonders bei langen Leitungen/Adaptern.
- **Logs**: journalctl -u visualisation.service ist deine erste Anlaufstelle. Keine sensiblen Daten loggen.
- **VPN**: Zugriff von aussen nur via Tailscale zulassen. ACLs eng halten.

10.4.10 Frontend Installation (Docker Compose + Keycloak, via scripts/setup.sh)

10.4.10.1 Voraussetzungen

Komponente	Minimum	Hinweis
Docker Engine	24+	docker --version
Docker Compose Plugin	v2	docker compose version

Node (nur für Dev)	20 LTS	optional, falls START_DEV=true
pnpm (nur für Dev)	9.x	optional, Script startet sonst dev nicht
Ports	3000 (FE), 8080 (Keycloak)	anpassbar via .env.development
Rechte	sudo	Script ruft sudo docker compose auf

Tabelle 42 Frontend Systemvoraussetzungen

10.4.10.2 .env Datei vorbereiten

Das Script erwartet **apps/web/.env.development**. Beispiel minimal:

Key	Beispiel	Zweck
PUBLIC_URL	http://localhost:3000	URL des Frontend Containers
VITE_API_BASE	http://localhost:5000	Backend API Basis URL
VITE_KEYCLOAK_URL	http://localhost:8080	Keycloak Basis URL
VITE_KEYCLOAK_REALM	masterarbeit	Realm Name
VITE_KEYCLOAK_CLIENT	frontend	Client ID
KC_DB_USERNAME	keycloak	DB User für keycloak-db
KC_DB_NAME	keycloak	DB Name für keycloak-db
KC_DB_PASSWORD	supersecret	DB Passwort (lokal)

Tabelle 43 Beispiel .env.development für Frontend Setup

Werte müssen zu deinem docker-compose.yml passen. Wenn Compose andere Variablen verlangt, hier anpassen.

10.4.10.3 Artefakte optional bereitstellen (Ordner transfer/)

Lege optionale Inhalte in **apps/web/transfer/** ab. Das Script importiert sie automatisch:

Pfad in transfer/	Effekt beim Setup
themes/	wird nach keycloak/themes/ kopiert (Custom Theme)
realm-export.json	wird nach keycloak/realm-export.json kopiert (Realm Import)
keycloak.dump	wird in Container keycloak-db eingespielt (voller DB Dump)

Tabelle 44 Inhalte im transfer/ und Wirkung

10.4.10.4 Setup ausführen

Aus **apps/web/scripts/** starten:

```
1. cd apps/web/scripts
2. chmod +x setup.sh
3. sudo ./setup.sh
4.
```

Was passiert:

Schritt	Funktion	Details
Env prüfen	require_env	bricht ab, wenn .env.development fehlt
Gerüste	scaffold_minimum	legt keycloak/, transfer/, scripts/ an, baut .dockerignore
Import	import_from_transfer	kopiert themes/, realm-export.json; optional: spielt keycloak.dump in keycloak-db ein
Compose	compose_up	docker compose pull/build/up -d mit .env.development
Optional Dev	maybe_start_dev	startet pnpm dev im Hintergrund, wenn START_DEV=true gesetzt
Hinweise	post_info	gibt Keycloak-URL, Frontend-URL, Health-Check aus

Tabelle 45 Aufgabenübersicht scripts/setup.sh (Frontend)

10.4.10.5 Nach dem Setup prüfen

Check	Kommando	Erwartung
Container laufen	sudo docker compose ps	frontend, keycloak, keycloak-db „Up“
Keycloak Ready	sudo curl -s http://localhost:8080/health/ready	"status":"UP"
Frontend erreichbar	Browser http://localhost:3000	Startseite lädt
API erreichbar	Browser http://localhost:5000/api/health/health	200 JSON
Theme aktiv	Keycloak Login	Custom Theme sichtbar (falls transfer/themes vorhanden)

Tabelle 46 Post-Setup Checks (Frontend/Keycloak)

10.4.10.6 Dev-Modus optional (ohne Container)

Wenn du lokal entwickeln willst, kannst du den Dev-Server automatisch starten lassen:

```
# aus apps/web/scripts/
START_DEV=true sudo ./setup.sh
# Logs: apps/web/dev.log
Oder manuell:
# Monorepo root oder apps/web
pnpm install
pnpm --filter web dev
# öffnet http://localhost:5173

```

Im Dev nutzt Vite typischerweise VITE_API_BASE=/api und proxyt zu VITE_API_PROXY_TARGET=http://localhost:5000 (falls im vite.config.ts so konfiguriert). In deinem .env.development kannst du stattdessen direkt VITE_API_BASE=http://localhost:5000 setzen.

10.4.10.7 Troubleshooting Frontend

Symptom	Ursache	Fix
Fehlt: .env.development	.env nicht erstellt	Datei in apps/web/.env.development anlegen
Frontend 404/leer	Container nicht up	sudo docker compose ps, Logs checken
Keycloak Login schlägt fehl	Realm/Client falsch	transfer/realm-export.json prüfen, VITE_KEYCLOAK_* prüfen
Theme greift nicht	Theme Pfad falsch	transfer/themes Struktur prüfen, Script erneut laufen lassen
API CORS Fehler	falsche API Basis	VITE_API_BASE korrigieren, Backend CORS setzen
Dev startet nicht	pnpm/Node fehlt	corepack enable && corepack prepare pnpm@latest --activate, Node 20 installieren

Tabelle 47 Häufige Probleme und Fixes

10.4.10.8 Kurzer Benutzerhinweis (Frontend)

- **Endanwender**: Charts und Tabellen, keine Schreibaktionen.
- **Techniker**: Admin-Bereich für M-Bus Scan, Modbus Read/Write, Monitor.
- **Verwalter**: Geräte benennen und Felder auswählen unter M-Bus Settings.
- **Auth**: Wenn Keycloak aktiv, Token wird im Axios-Interceptor gesetzt.

11 Ergebnisse, Metriken und Evaluation

11.1 Funktionale Ergebnisse (Visualisierung, Quittierung, Export)

Der Prototyp erfüllt die definierten Soll-Ziele:

- **M-Bus**: Live-Scan via SSE mit sauberem Abbruch, Einzelgeräte-Auslesung, XML-Parsing und Normalisierung.
- **Modbus**: Read/Write für RTU und TCP, optionaler Read-Back und Logging.
- **Visualisierung/Monitoring**: Tabellen, Charts mit Aggregation (Range/Bucket), Monitor mit Mindestabständen (Cooldown).
- **Rollenmodell**: Lesen vs. Konfigurieren/Schreiben sauber getrennt.
- **Betrieb**: systemd-Dienst läuft stabil; Health-Endpoint liefert 200.

Bereich	Ergebnis	Nachweis (Anhang)
M-Bus Scan & Abbruch	Stream mit Log/Adressen, kontrollierter Abbruch	A3, B2
Einzelgerät (M-Bus)	Normalisierte Werte (Tag → Wert)	A3, B2
Modbus Read/Write	Read-Back bestätigt Setzen	A4, B2
Charts & Metrics	Zeitreihen (Range/Bucket/Rate)	B2, B3
Monitoring	Cooldown eingehalten, keine Kollisionen	A5

Tabelle 48 Funktionale Ergebniskarte (Nachweis im Anhang)

11.2 Performance & Zuverlässigkeit

Messungen wurden lokal (Pi) und praxisnah durchgeführt. Ziel: nachvollziehbare Latenzen, stabile Reads/Writes, konsistenter Monitor.

Metrik	Zielwert	Ergebnis	Messmethode
API Latenz Median	≤ 300 ms	[eintragen]	50 Requests lokal
API Latenz P95	≤ 800 ms	[eintragen]	dito
M-Bus Einzelgerät	≤ 2.0 s	[eintragen]	10 Reads am selben Zähler
Modbus Read RTU	≤ 400 ms	[eintragen]	RS485 19200 Baud
Modbus Read TCP	≤ 150 ms	[eintragen]	LAN
Monitor Cooldown	≥ 30 s	[eintragen]	Log/Monitor-Antwort
Dienstverfügbarkeit	≥ 99 % Testzeitraum	[eintragen]	systemd/journal

Tabelle 49 Performance-Kennzahlen (bitte Werte ergänzen)

11.3 Usability-Feedback (Endnutzer vs. Techniker)

Kurzfazit aus den explorativen Interviews (Details in **E2–E4**):

- **Endanwender** wünschen klare Achsen/Einheiten in Charts und dezente Tooltips.
- **Verwalter** wollen Gerätebenennungen und Feldauswahl direkt im Flow.
- **Techniker** möchten sichtbaren Read-Back beim Schreiben und konsistente Fehlermeldungen.

Rolle	Beobachtung	Wirkung	Massnahme
Endanwender	Achsenlabel zu technisch	Verwirrung	Klartext/Einheiten, Tooltips
Verwalter	Benennen nicht prominent	Mehraufwand	„Benennen“ direkt in Listenzeile
Techniker	Write ohne Read-Back	Unsicherheit	Read-Back-Wert im Toast/Panel

Tabelle 50 Beobachtungen und Massnahmen

11.4 Nutzenbewertung (Zeitersparnis, Kosten)

Die Lösung reduziert Diagnose- und Vor-Ort-Aufwände und erhöht Transparenz.

Vorgang	Vorher	Nachher	Delta
Erstdiagnose vor Ort	60 min	10 min remote	−50 min
Zweitanfahrt (Ersatzteil)	1× pro 3 Fälle	1× pro 10 Fälle	−70 %
Zählerablesung MFH	90 min	5 min	−85 min

Tabelle 51 Aufwand vorher vs. nachher (Annahmen)

Tabelle:

Kategorie	Nutzen
Transparenz	Verbrauchsdaten jederzeit einsehbar
Sicherheit	VPN-Zugriff, Rollen, kein Cloudzwang
Erweiterbarkeit	Geräte/Tags ohne Neuaufbau
Vendor-Lock-in	gering dank offener Protokolle

Tabelle 52 Qualitativer Nutzen

11.5 Evaluationsmethodik und Validität

- **Methodik**: Laborläufe auf dem Pi, Praxisläufe an einer realen Installation; CI-Volltests mit echter DB.
- **Replizierbarkeit**: definierte .env-Parameter und JSON-Configs; dokumentierte Messschritte.
- **Grenzen**: Ergebnisse gelten für die getestete Topologie/HW; keine Hochlast mit Hunderten Geräten.
- **Bias-Kontrolle**: Screencasts und Logs belegen die Flows; Coverage/CI belegt Backend-Robustheit.

11.6 Metrikenübersicht und Leseführung

Tabelle:

Metrik	Kapitel	Nachweis (Anhang)
Coverage gesamt	9.4	B3, B4, B6
Latenzen	11.2	B2 (Screens), Logs
Monitor Cooldown	7.4, 11.2	A5
Usability-Feedback	11.3	E2–E4

Tabelle 53 Metriken → Kapitel → Nachweis

12 Diskussion

12.1 Erfüllung der Ziele

Die Kernziele wurden erreicht, die Abweichungen sind fachlich begründet und verbessern die Bedienbarkeit.

Ziel	Soll	Ist	Nachweis
Protokolle	M-Bus und Modbus parallel nutzbar	Ja, mit Queue, Cooldown, sauberem Abbruch beim Scan	A3, A4, A5, Kap. 7.4
Visualisierung	Tabellen und Diagramme, Monitoring	Ja, Zeitreihen mit Range und Bucket, Monitor mit Mindestabständen	Kap. 7.3, 8.4, 11.1
Konfiguration	Excel Import	Ersetzt durch Web UI und JSON pro Gerät (robuster, weniger Fehler)	Kap. 7.3.4
Betrieb	Lokal, kein Cloud Zwang	Ja, systemd Dienst, Health Endpunkt, VPN für Fernzugriff	A2,
Qualität	Automatisierte Tests, Coverage	Ja, Coverage Gate erreicht, CI Pipeline belegt	B3–B6,

Tabelle 54 Zielerreichung (Soll vs. Ist, mit Nachweisen)

Abweichungen

- Excel wurde durch Web UI Konfiguration ersetzt (direktes Feedback, weniger Medienbrüche).
- Modbus Gegenstelle im Prototyp: I/O Modul statt Wärmepumpe (gleiche technischen Pfade, geringere Komplexität).

12.2 Grenzen der Lösung

Technisch

- **Skalierung**: Ausgelegt auf kleine bis mittlere Installationen. Keine Messungen mit Hunderten Geräten.
- **Langzeit-Analytics**: Historik und Dashboards sind vorbereitet, aber nicht vollumfänglich umgesetzt.
- **Sicherheit**: Grundhärtung vorhanden (VPN, Updates, Rollen), jedoch kein mTLS oder Reverse Proxy mit strengem Header Set.
- **Frontendläufe**: Keine automatisierten UI-Tests, Nachweis via Screencasts.
- **Containerbetrieb**: Backend nicht standardisiert containerisiert; Betrieb derzeit klassisch per systemd.

Organisatorisch

- **Gerätevielfalt**: Verifiziert mit repräsentativen, nicht mit sämtlichen Zählertypen und Steuerungen.

- **Abhängigkeit von Feldbedingungen**: Serielle Qualität, Terminierung und Erdung beeinflussen Stabilität.

Grenze	Wirkung	Mitigation
Keine Hochlast-Validierung	Unsichere Aussage für 100+ Geräte	Messreihen nachziehen, TimescaleDB aktivieren
Grundhärtung, kein TLS Proxy	Transport unsigniert im LAN	Reverse Proxy mit TLS und HSTS ergänzen
Keine UI-Automatisierung	UI-Regression nicht maschinell geprüft	Kleine Jest-Smoke-Tests für kritische Flows
Device-Abdeckung begrenzt	Integrationsrisiko bei exotischen Geräten	Mapping und Decoder modulär halten, Piloten je Gerät

Tabelle 55 Grenze → Wirkung → Mitigation

12.3 Vergleich mit bestehenden Lösungen

Positionierung

- **Kosten/Nutzen**: Lokaler Betrieb, offene Protokolle, geringe Einstiegskosten.
- **Bedienung**: Fokus auf essentielle Flows (Scan, Read, Monitor, Charts) statt komplexe HMI-Suiten.
- **Unabhängigkeit**: Herstellerunabhängig durch M-Bus und Modbus, keine Lizenzpflicht.

Trade-offs

- Weniger Out-of-the-box Komfortfunktionen grosser Leitsysteme (3D-Grundrisse, umfangreiche Alarmierung, Wartungsplanung).
- Höhere Eigenverantwortung beim Betrieb (VPN, Updates, Backups).

Kriterium	Etablierte Systeme	Diese Lösung
Anschaffung	Hoch	Niedrig
Laufende Kosten	Lizenzen möglich	Minimal
Interoperabilität	Ökosystemgebunden	Offen durch M-Bus/Modbus
Bedienung	Umfassend, komplex	Schmal, fokussiert
Betrieb	Herstellergeführt	Eigenbetrieb, dokumentiert

Tabelle 56 Einordnung (Kurz)

(Detailvergleich siehe Tabelle 1 und Kap. 5.4.)

12.4 Lessons Learned

- **Walking Skeleton** zahlt sich aus: Risiken bei Scan, Parser und Persistenz früh sichtbar, spätere Arbeit stabiler.
- **SSE mit sauberem Abbruch** ist ein Muss: Verhindert hängende Prozesse und vermittelt der UI verlässliche Zustände.
- **Queue und Cooldown** entschärfen Kollisionen zwischen M-Bus-Scan, Modbus-Reads und Monitor.

- **Konfiguration in der UI** statt Excel reduziert Fehler, beschleunigt Iteration.
- **CI-Gate mit Coverage** hält die Codequalität hoch und verhindert Regressionen im Backend.
- **Dokumentierte Artefakte** (Screens, Logs, Pipelines) beschleunigen Abnahme und Diskussion mit Stakeholdern.

12.5 Ausblick

Kurzfristig

- CSV/Parquet Export und einfache Regel-Benachrichtigungen (Schwellen, Zeitfenster).
- Kleine UI-Smoke-Tests (Jest) für Scan, Read, Write, Monitor.
- Reverse Proxy mit TLS und sauberen CORS-Regeln, optional mTLS.

Mittelfristig

- TimescaleDB aktivieren und grössere Messreihen evaluieren.
- Mapping-Editor für Fehlercodes und Gerätetemplates.
- Containerisierte Bereitstellung mit Compose (Backend) und statischer Auslieferung des Frontends hinter demselben Host.

Langfristig

- Rollen und Policies feiner auflösen, Keycloak standardisieren.
- BI-Anbindung oder Dashboard-Integration, falls Bedarf wächst.

13 Projektmanagement

13.1 Projektplanung & Iterationen

Die Umsetzung erfolgte in kurzen Sprints mit klarer Aufgabenzerlegung und PR-basiertem Merge-Prozess. Planung, Abarbeitung und Kontrolle wurden über Board, Backlog und Reports gesteuert.

Abbildung 20: Sprint-Flow (Board -> PR -> Pipeline -> Done)

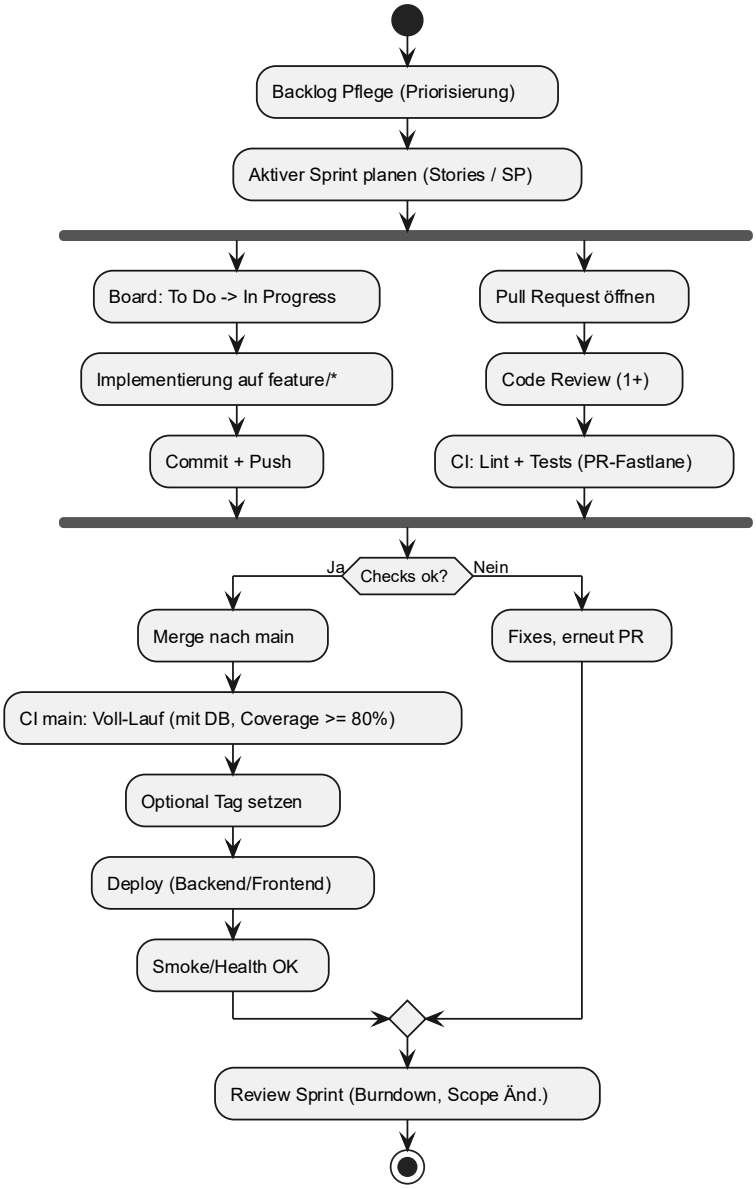


Abbildung 20 Sprint-Flow (Board -> PR -> Pipeline -> Done)

13.2 Zeiterfassung & Aufwand

Gesamtaufwand: **774 h**. Pro Sprint **77–78 h**.

Sprint	Stunden	Analyse	Implementierung	Test	Dokumentation
S1	77	54	15	4	4
S2	77	23	38	8	8
S3	77	12	50	11	4
S4	77	8	50	15	4
S5	77	8	46	15	8
S6	77	4	35	31	7
S7	78	4	19	47	8
S8	78	8	43	19	8
S9	78	8	15	8	47
S10	78	4	8	8	58
Summe	**774**	**133**	**319**	**166**	**156**

Tabelle 57 Sprintstunden je Aktivität (Largest-Remainder, ganze Stunden)

Aktivität	Stunden	Anteil
Analyse	133	**17.2 %**
Implementierung	319	**41.3 %**
Test	166	**21.5 %**
Dokumentation	156	**20.2 %**
Total	**774**	**100 %**

Tabelle 58 Gesamtverteilung nach Aktivität

13.3 Rollenverteilung (Kerim, Remzi)

Rollen und Verantwortlichkeiten wurden klar zugeordnet. Die Matrix zeigt, wer federführend ist und wer beratend unterstützt.

Artefakt / Meilenstein	Responsible (R)	Accountable (A)	Consulted (C)	Informed (I)
Backend Architektur & Treiber	Kerim	Kerim	Remzi	Stakeholder
Frontend Flows & UI	Remzi	Remzi	Kerim	Stakeholder
DB Schema & Persistenz	Kerim	Kerim	Remzi	Stakeholder
Tests & CI-Gates	Kerim	Kerim	Remzi	Stakeholder
Deployment (Pi, systemd)	Kerim	Kerim	Remzi	Stakeholder
Doku Kapitel 5–10	Kerim	Kerim	Remzi	Stakeholder
Screencasts & UI-Nachweise	Remzi	Remzi	Kerim	Stakeholder

Tabelle 59 RACI-Matrix (Kernartefakte)

13.4 Herausforderungen im Projektmanagement

Die wichtigsten Hürden und wie sie adressiert wurden.

Thema	Risiko	Auswirkung	Massnahme
Umfang	Scope Creep	Verzögerungen, Qualitätsverlust	harte Sprintziele, Kann-Ziele nur bei Puffer
Abhängigkeiten	Feldgeräte nicht verfügbar	Blockierte Tests	Labor-Setups, Mocks, frühzeitige Beschaffung
Serielle Kollisionen	Scan vs. Reads	Hänger, instabile Demos	Queue + Cooldown, eigener Abort-Flow
Qualität	fehlende Regressionstests	„Green“ lokal, rot später	CI-Gate ≥ 80 %, PR-Pflicht
Kommunikation	PR ohne Kontext	Review-Loops länger	Ticket-Referenz im Commit, PR-Template
Betrieb	manuelle Schritte	Fehleranfällig	`setup.sh`, Checklisten (Kap. 10.4)

Tabelle 60 Risiken im PM und Massnahmen

14 Schlussfolgerungen und Ausblick

14.1 Beantwortung der Forschungsfragen

Frage	Kurzantwort	Evidenz (Kapitel)	Nachweise (Anhang)
F1: Realisierbarkeit auf Standard-Hardware (Raspberry Pi)	Ja. Stabiles End-to-End mit M-Bus/Modbus, Persistenz, UI	7, 8, 10, 11	A1, A2,
F2: Parallelbetrieb von M-Bus und Modbus ohne Kollisionen	Ja, unter Leitplanken. Queue/Cooldown, sauberer Scan-Abort	7.3, 7.4, 8.3	A3–A5
F3: Architektur für Endanwender und Techniker	Ja. Rollentrennung, dedizierte Admin-Seiten	7.3.2, 8.4, 8.5	A1, B2
F4: Praktikable Konfiguration (Datei/UI)	Ja, via Web-UI + JSON. Excel abgelöst, weniger Fehler	7.3.4, 8.3	G4

Tabelle 61 Forschungsfragen ↔ Evidenz ↔ Nachweise

14.2 Fazit

Die Arbeit zeigt, dass ein **lokal betreibbares, herstellerunabhängiges Visualisierungssystem** auf Raspberry-Pi-Basis technisch tragfähig ist. M-Bus und Modbus können parallel betrieben werden, sofern **Cooldowns, Queueing** und ein **kontrollierter Scan-Abort** eingehalten werden. Die **UI-Konfiguration** anstelle eines Excel-Imports reduziert Medienbrüche und erhöht die Robustheit. Qualitätssicherung über **CI-Gate (Coverage ≥ 80 %)**, reproduzierbare Tests und dokumentierte Nachweise (Screens, Logs, Pipeline-Runs) stützen die Ergebnisse.

14.3 Weiterentwicklungspotenziale (z. B. Push-Benachrichtigungen, Cloud-Anbindung)

Funktional

- **Export & Analytics:** CSV/Parquet Export, tägliche/wöchentliche Aggregationen, einfache Berichte.
- **Benachrichtigungen:** Schwellwerte, Zeitfenster, Eskalation (E-Mail/Push/Webhook).
- **Geräte-Abdeckung:** Templates, Mapping-Editor für Fehlercodes, zusätzliche Protokolle (z. B. SML, OPC UA).

Technik & Betrieb

- **TimescaleDB** aktivieren und Langzeit-Metriken evaluieren.
- **Reverse Proxy + TLS** (optional mTLS), restriktive **CORS**.
- **Containerisierung** (Compose) für das Backend; statische Auslieferung des Frontends mit gemeinsamem Host (/api).

Qualität

- **Kleine UI-Smoke-Tests** (Jest/RTL) für kritische Flows (Scan, Read, Write, Monitor).

- Monitoring der Betriebsmetriken (CPU/RAM/Lat) auf dem Pi, Alarmierung bei Grenzwerten.

Zeitraum	Schwerpunkt	Ergebnisse
0–3 Monate	Export, einfache Benachrichtigungen, TLS/Proxy	CSV/Parquet, Rule-Engine v1, HTTPS-Betrieb
3–6 Monate	TimescaleDB, Containerisierung, UI-Smoke-Tests	bessere Historik, Compose-Stack, Basistests FE
6–12 Monate	Protokollerweiterungen, Templates, BI-Anbindung	zusätzliche Geräte/Protokolle, Dashboard-Optionen

Tabelle 62 Roadmap (0–12 Monate)

14.4 Empfehlungen für den Produktivgang

- **Netz & Zugriff:** Zugriff von aussen **ausschliesslich** via VPN; TLS/Reverse-Proxy für gemeinsamen Host (Frontend + /api) einrichten.
- **Berechtigungen:** Schreib- und Konfigurationsaktionen **nur** für Rolle „Techniker“; Admin-Routen mit Guard.
- **Konfiguration:** .env zentral, dokumentierte Defaults; serielle Parameter (Port/Baud) validieren.
- **Betrieb:** systemd-Service mit Restart-Policy, Log-Rotation; **unattended-upgrades** aktiv.
- **Backups:** Regelmässige DB-Sicherungen prüfen (Restore-Test).
- **Nachweise pflegen:** CI-Gate (Coverage), Pipeline-Runs, Deploy-Screenshots; erleichtert Audits und Abnahmen.

14.5 Schlussbemerkung

Die Kombination aus **Walking Skeleton**, **klaren Kommunikationsabläufen** (SSE, Queue, Cooldown) und **disziplinierter QS** hat die technischen Risiken früh reduziert und eine belastbare Grundlage geschaffen. Die Lösung ist **schlank**, **erweiterbar** und **praxisnah**. Sie adressiert gezielt den Bedarf kleiner bis mittlerer Installationen nach Transparenz und Störungsdiagnostik — ohne Vendor-Lock-in und ohne Cloudzwang.

15 Danksagung

Wir danken **Dr. Thomas Memmel** herzlich für die engagierte Betreuung unserer Arbeit. Seine präzisen Rückmeldungen, sein Fokus auf methodische Stringenz sowie seine praxisnahen Impulse zu Architektur, Testbarkeit und Dokumentation haben die Qualität dieser Arbeit wesentlich erhöht.
Ebenso danken wir unseren Interviewpartnern für ihre Zeit und ihre wertvollen Perspektiven:

- **Kaan Cehreli** (Sicht eines normalen Nutzers)
- **Bülent Sünbül** (Eigentümer)
- **Srdjan Jankovic** (Techniker)

Ihre Einschätzungen haben die Anforderungen geschärft, die Benutzerführung verbessert und die Praxistauglichkeit der Lösung bestätigt.
Unser Dank gilt auch allen Personen, die beim Testen geholfen, Feedback gegeben oder Infrastruktur zur Verfügung gestellt haben, sowie dem MAS SE Team der OST für die konstruktiven Rahmenbedingungen. Ein besonderer Dank gilt unseren Familien und unserem Umfeld für Geduld, Motivation und Unterstützung während intensiver Phasen dieser Arbeit.

16 Literaturverzeichnis

Avelon AG. (2025). *Avelon*. Von Avelon - the art of steering real estate: https://avelon.com/ abgerufen

Beckhoff. (2025). *Steuerungskomponenten für die Gebäudeautomation*. Von Beckhoff New Automation Technology: https://www.beckhoff.com/de-ch/branchen/av-und-medientechnik/gebaeudeautomation-gewerke/ abgerufen

EN 13757, C. (2004). Communication systems for meters and remote reading of meters. European Committee for Standardization.

Gebäudeautomation, M. –F. (2023). *IoT im Gebäude – Marktstudie 2022*. Von MeGA – Fachverband Gebäudeautomation Schweiz: https://www.mega-planer.ch/fadaladdondlz/files/.addonpublikationeintragfile/publikationen/98.pdf/MeGA%20Marktstudie%20IoT%202022i.pdf abgerufen

Härz AG. (2025). *Mit der passenden Visualisierung alles im Blick*. Von https://gebaeudeinformatik-schweiz.ch/de/dienstleistungen/visualisierung/ abgerufen

Relay. (2025). *Das Bussystem für die Fernauslesung von Zählern*. Von https://www.relay.de/ abgerufen

Schweiz, E. Z. (2021). *Energie Zukunft Schweiz*. Von Swissolar: https://www.swissolar.ch/03_angebot/veranstaltungen/vortraege-und-studien/2021/20210504_ezs_swissolar_iot.pdf abgerufen

Wikipedia. (2024). *Feldbus, Modbus* . Von Wikipedia: https://de.wikipedia.org/wiki/Modbus abgerufen

Flask-CORS contributors. (n. d.). *flask-cors* [Computer software]. https://flask-cors.readthedocs.io/

Flask-RESTX developers. (n. d.). *Flask-RESTX* [Computer software]. https://flask-restx.readthedocs.io/

Pallets. (n. d.). *Flask* [Computer software]. https://flask.palletsprojects.com/

Pallets. (n. d.). *ItsDangerous* [Computer software]. https://itsdangerous.palletsprojects.com/

Pallets. (n. d.). *Jinja2* [Computer software]. https://jinja.palletsprojects.com/

Ronacher, A., & Pallets. (n. d.). *Werkzeug* [Computer software]. https://werkzeug.palletsprojects.com/

The Psycopg Project. (n. d.). *psycopg2-binary* [Computer software]. https://www.psycopg.org/

python-dotenv contributors. (n. d.). *python-dotenv* [Computer software]. https://github.com/theskumar/python-dotenv

Requests contributors. (n. d.). *requests* [Computer software].
https://requests.readthedocs.io/

Axios contributors. (n. d.). *Axios* [Computer software].
https://axios-http.com
Chart.js contributors. (n. d.). *Chart.js* [Computer software].
https://www.chartjs.org/
date-fns contributors. (n. d.). *date-fns* [Computer software].
https://date-fns.org/
Flowbite React contributors. (n. d.). *Flowbite React* [Computer software].
https://flowbite.com/docs/getting-started/react/
Keycloak project. (n. d.). *keycloak-js* [Computer software].
https://www.keycloak.org/
Meta Open Source. (n. d.). *React* [Computer software].
https://react.dev/
Remix Software, Inc. (n. d.). *React Router* [Computer software].
https://reactrouter.com/
TanStack. (n. d.). *@tanstack/react-query* [Computer software].
https://tanstack.com/query/latest
Vite contributors. (n. d.). *Vite* [Computer software].
https://vitejs.dev/
Tailwind Labs. (n. d.). *Tailwind CSS* [Computer software].
https://tailwindcss.com/
reactchartjs/react-chartjs-2 contributors. (n. d.). *react-chartjs-2* [Computer software].
https://github.com/reactchartjs/react-chartjs-2
Chart.js team. (n. d.). *chartjs-adapter-date-fns* [Computer software].
https://www.chartjs.org/docs/latest/axes/cartesian/time.html#date-adapters

JEAN, P. (n. d.). *mbpoll* [Computer software].
https://github.com/epsilonrt/mbpoll
libmbus project. (n. d.). *libmbus* [Computer software].
https://github.com/rscada/libmbus
PostgreSQL Global Development Group. (n. d.). *PostgreSQL* [Computer software].
https://www.postgresql.org/
msmtp project. (n. d.). *msmtp* [Computer software].
https://marlam.de/msmtp/
Tailscale Inc. (n. d.). *Tailscale* [Computer software].
https://tailscale.com/
Red Hat. (n. d.). *Keycloak (Server)* [Computer software].
https://www.keycloak.org/

Docker, Inc. (n. d.). *Docker* [Computer software].
https://www.docker.com/

Atlassian. (n. d.). *Bitbucket Pipelines* [Computer software].
https://bitbucket.org/product/features/pipelines

pytest dev team. (n. d.). *pytest* [Computer software].
https://docs.pytest.org/

pytest-cov contributors. (n. d.). *pytest-cov* [Computer software].
https://pytest-cov.readthedocs.io/

Astral. (n. d.). *ruff* [Computer software].
https://docs.astral.sh/ruff/

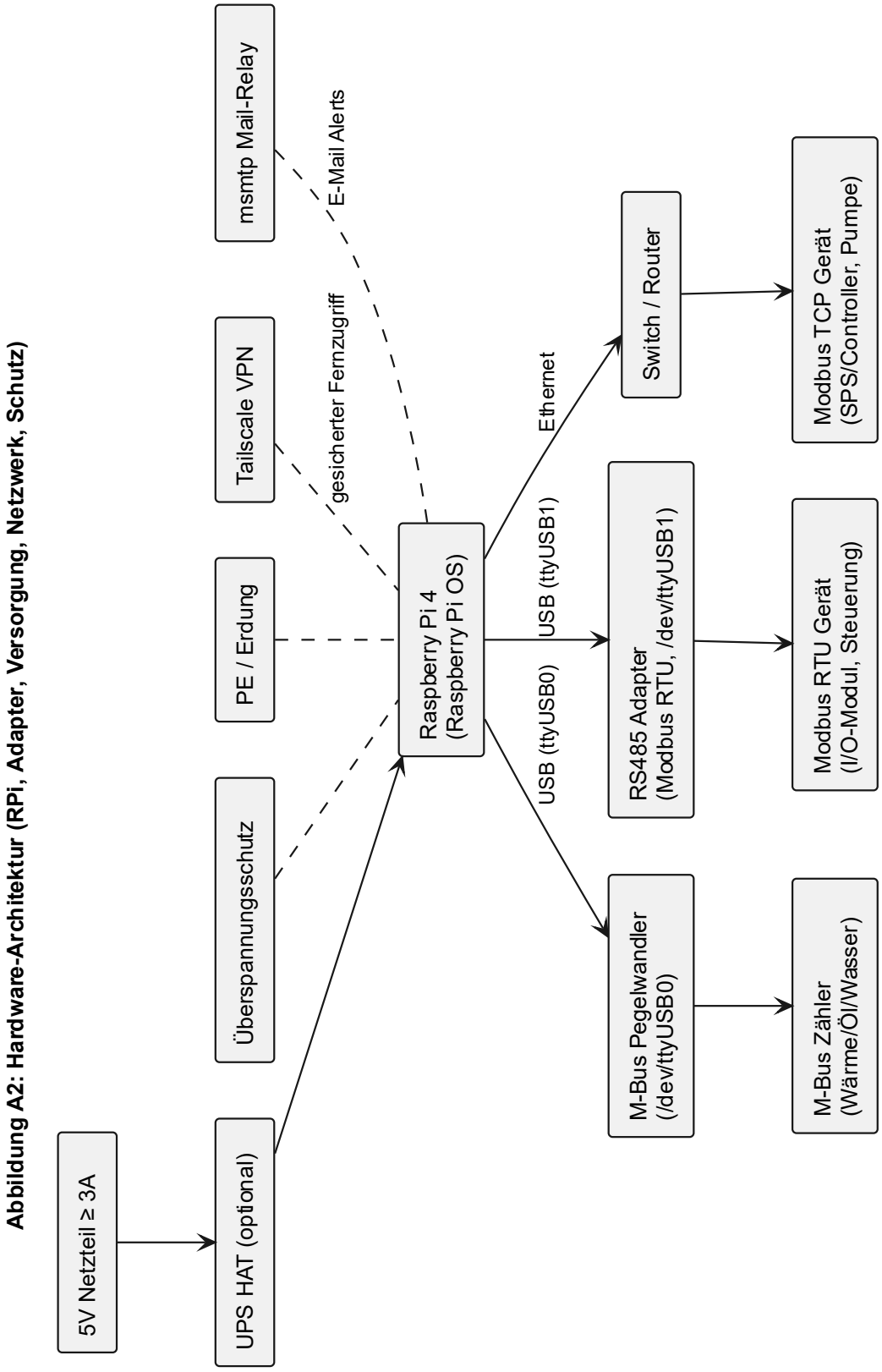
PyCQA. (n. d.). *flake8* [Computer software].
https://flake8.pycqa.org/

PlantUML team. (n. d.). *PlantUML* [Computer software].
https://plantuml.com/

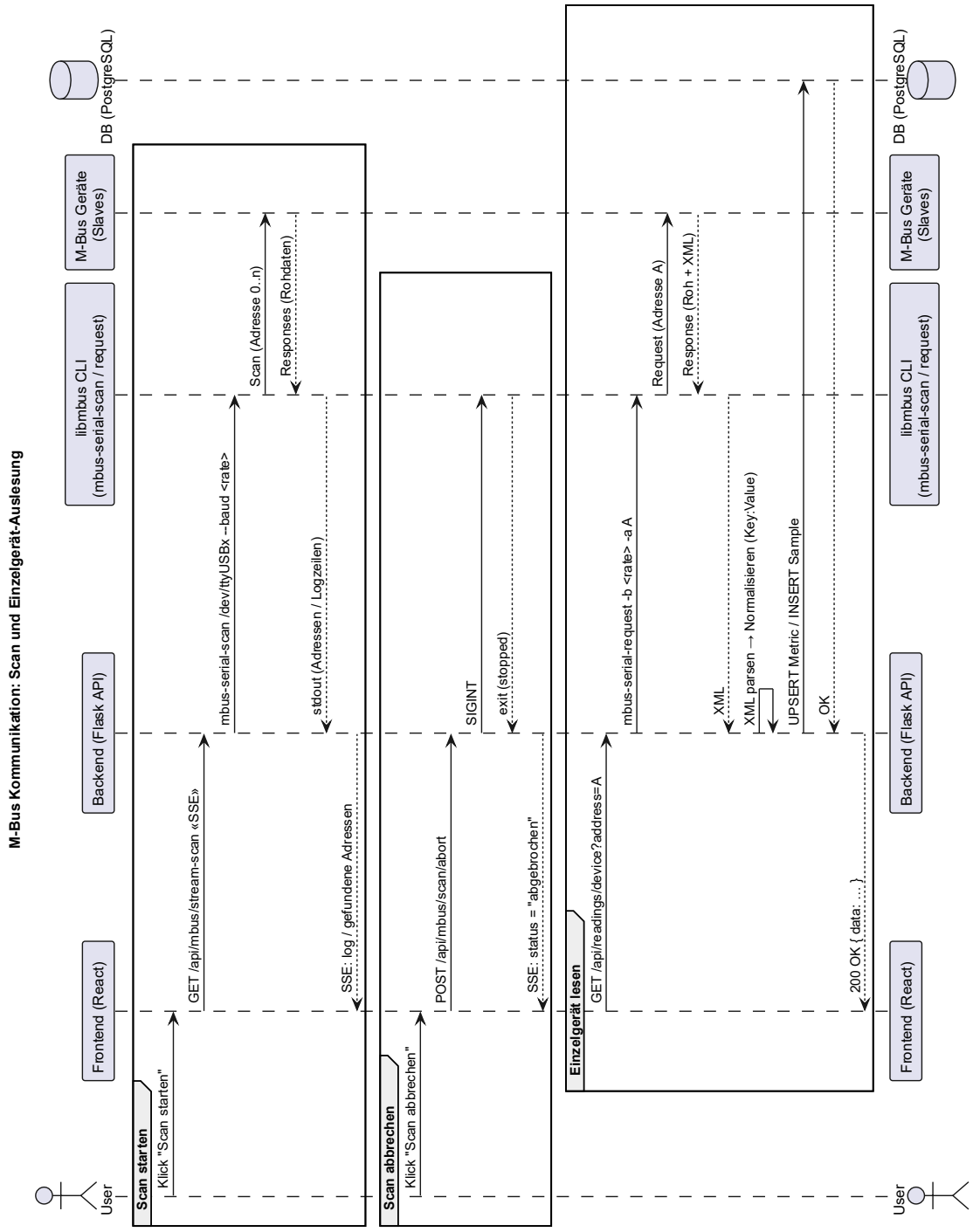
OpenAI. (n. d.). *ChatGPT (large language model)* [Computer software].
https://chat.openai.com/

17 Abbildung A1: Systemarchitektur – Durchstich von Feldgeräten bis Web-UI

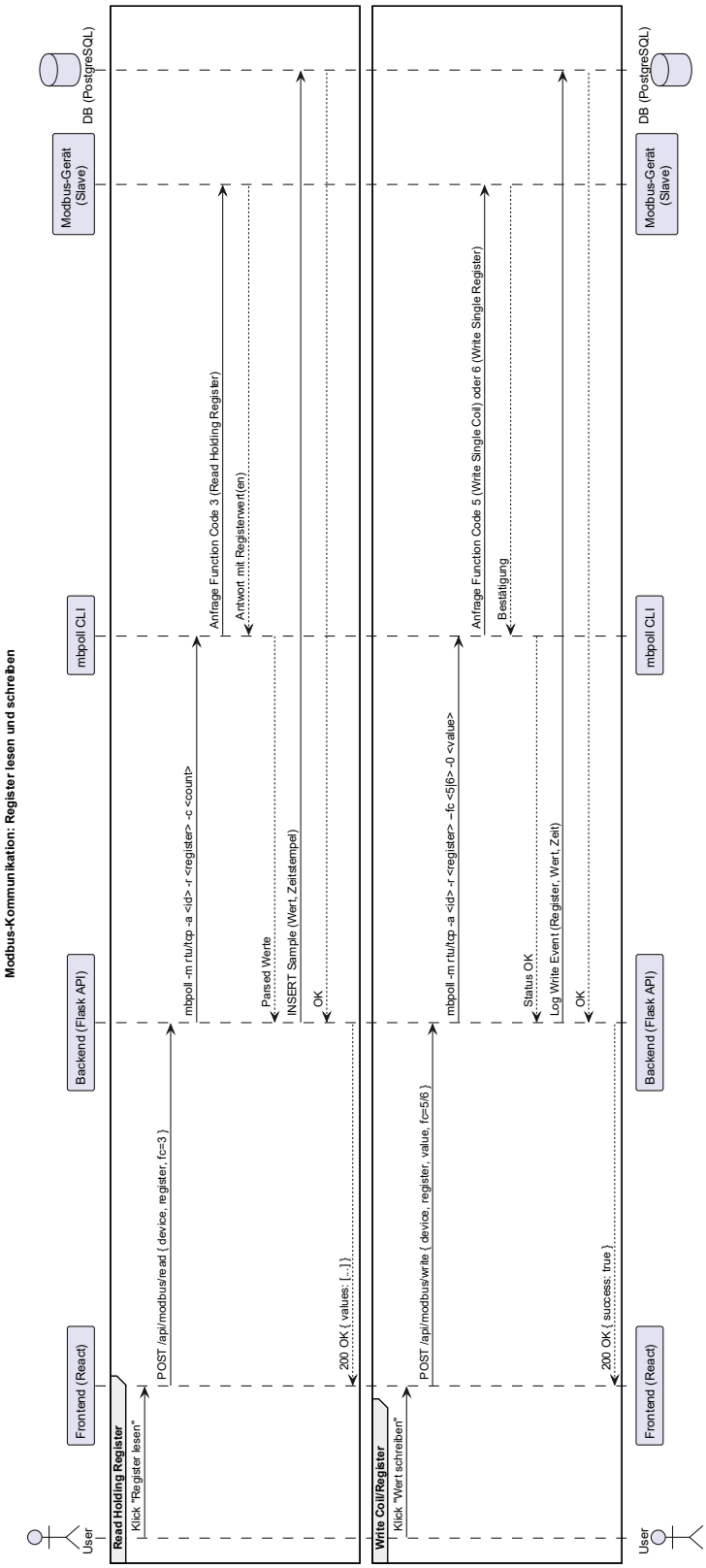
18 Abbildung A2: Hardware-Architektur (in UML)



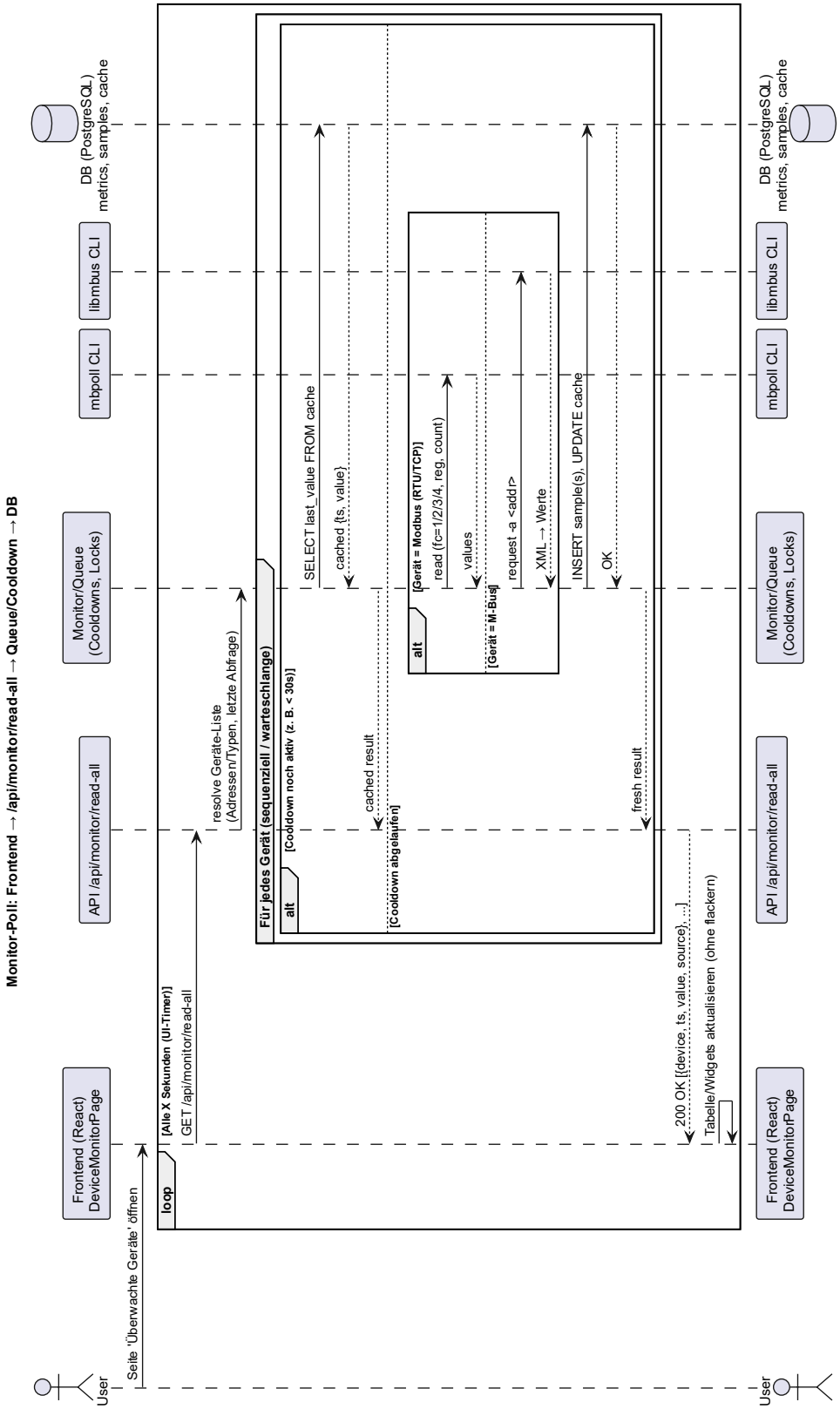
19 Abbildung A3: M-Bus-Kommunikation (Scan und Einzelgerät-Auslesung)



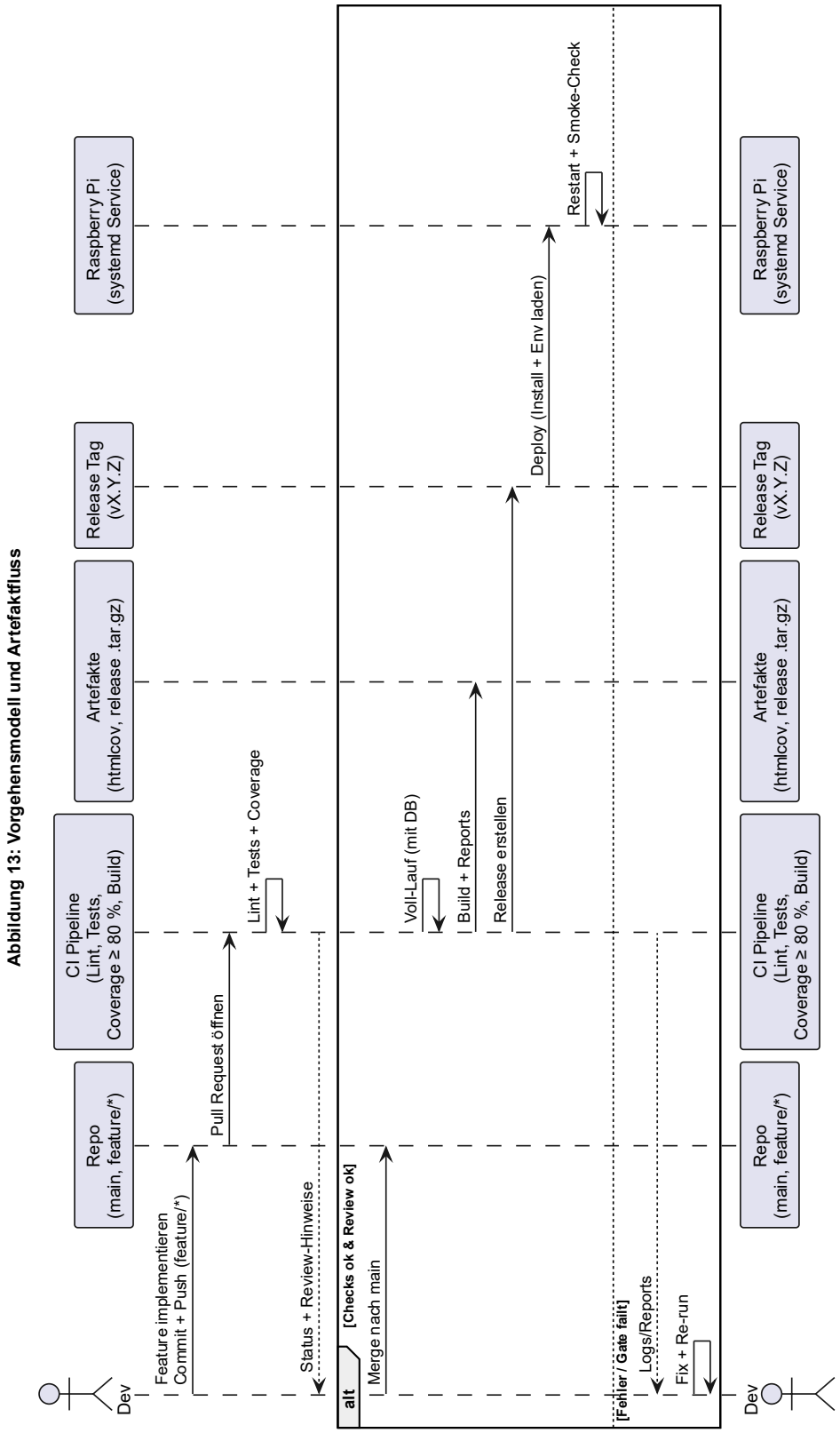
20 Abbildung A4: Modbus-Kommunikation (Register lesen und schreiben)



21 Abbildung A5: Monitor-Poll mit Cooldown (mehrere Geräte, Mindestabstand 30 s)



22 Abbildung A6: Vorgehensmodell und Artefaktfluss (Dev → CI → Release → Deploy)



23 Tabelle E1: Interviewleitfaden (Fragenblöcke kurz)

Die folgenden Interviews sind **explorativ** und dienen der Abstimmung von Bedienkonzept, Visualisierung und Betriebsanforderungen. Es wurden drei Perspektiven betrachtet: **normaler User** (Kaan Cehreli), **Eigentümer/Betreiber** (Bülent Sünbül) und **Techniker/Service** (Srdjan Jankovic). Format: 20–30 Minuten, remote; Notizen wurden durch die Autoren erstellt. Die Aussagen sind **nicht repräsentativ**, liefern aber wertvolle Hinweise für die Produktgestaltung.

Block	Ziel	Beispielfragen
Einstieg & Kontext	Nutzungskontext verstehen	In welchem Umfeld würden Sie das System nutzen? Welche Aufgabe ist für Sie am wichtigsten?
Visualisierung & Informationen	Relevante Daten, Darstellung	Welche Werte wollen Sie zuerst sehen? Tabelle oder Diagramm? Welche Zeiträume sind wichtig (Tag/Woche/Monat)?
Bedienbarkeit	Usability, Navigation	Wo erwarten Sie den Gerätescan? Wie finden Sie Einstellungen wieder? Was muss jederzeit erreichbar sein?
Fehlermeldungen & Quittierung	Störungen erkennen/handhaben	Was soll bei einer Störung passieren? Reicht ein Hinweis oder brauchen Sie Anleitungen?
Konfiguration & Rollen	Trennung User/Technik	Welche Einstellungen möchten Sie selbst ändern? Was gehört nur in den Technikerbereich?
Monitoring & Benachrichtigung	Regelbetrieb	Welche Intervalle genügen für Aktualisierungen? Wann brauchen Sie eine E-Mail/Push?
Sicherheit & Datenschutz	Vertrauen schaffen	Lokalbetrieb okay? VPN bekannt? Welche Daten dürfen gespeichert werden?
Gesamturteil	Nutzen & Hürden	Was überzeugt Sie am meisten? Was würde Sie vom Einsatz abhalten?

24 E2: Antworten Kaan Cehreli (Normaler User)

Rahmendaten

- Rolle: normaler Endanwender (Haushalt)
- Dauer/Format: ~25 Min., remote
- Ziel: Erstkontakt, Visualisierung und Verständlichkeit

Kernaussagen (Kurzfassung)

- Startseite soll **sofort den Verbrauch** zeigen (heute, Woche, Monat), **Diagramm + Zahl**.
- **Technikbegriffe vermeiden**; klare Einheiten und Legenden.
- Störungen: **klare Meldung in Klartext** mit einfachem Hinweis „Was tun?“.
- Keine Lust auf Konfiguration: **Voreinstellungen** und selbsterklärende Labels.

Frage	Antwort (Zusammenfassung)	Hinweis für System
Was sehen Sie zuerst?	„Aktueller Verbrauch und ob alles ok ist.“	Home: KPI-Karten + Ampelstatus
Tabelle oder Diagramm?	„Beides. Balken pro Tag helfen mir.“	Chart + Tabelle umschaltbar
Störung/Fehler?	„MBus Scan gibt manchmal Fehler.“	Fehlermeldung + „Nächste Schritte“
Einstellungen?	„Nur Namen vergeben. Rest egal.“	Labels leicht zugänglich; Technik verstecken
Benachrichtigung?	„E-Mail reicht, nicht zu oft. Und nur Alarmierung System und nicht Schwellwert Alarmierung“	Schwellen + Ruhezeiten

Ableitungen für das System

- Home mit **KPI/Kachel**, Wochenchart, Ampelstatus.
- **Tooltips/Legenden**, sprachlich einfach.
- Fehlermeldungen mit **Kurzhandlung** (z. B. „Kontakt Technik“).

25 E3: Antworten Bülent Sünbül (Eigentümer)

Rahmendaten

- Rolle: Eigentümer/Betreiber einer Liegenschaft
- Dauer/Format: ~30 Min., remote
- Ziel: Betriebssicht, Abrechnung, Verfügbarkeit

Kernaussagen (Kurzfassung)

- Will **Monats- und Quartalswerte** für Abrechnung und Vergleich.
- **Gerätenamen und Orte** sauber pflegbar; Export für Excel/Abrechnung wünschenswert.
- Benachrichtigungen nur bei **relevanten** Ereignissen; keine Flut.
- Lokalbetrieb ist gut, **Fernzugriff via VPN** genügt.

Frage	Antwort (Zusammenfassung)	Hinweis für System
Wichtige Zeiträume?	„24h, 7Tage und 30Tage zu wenig!.“	Aggregationen bereitstellen
Datenexport?	„CSV/Excel für Abrechnung.“	Export aus UI (später)
Benachrichtigungen?	„Nur bei echten Problemen.“	Schwellen + Eskalation
Verwaltung?	„Geräte benennen, Orte zuordnen.“	Labels/Metadaten prominent
Zugriff?	„VPN ist ok.“	Dokumentation VPN-Zugriff

Ableitungen für das System

- **Aggregierte Zeiträume** und **Export-Option** vorsehen.
- Label-/Metadatenpflege zentral erreichbar.

26 E4: Antworten Srdjan Jankovic (Techniker)

Rahmendaten

- Rolle: Techniker/Service
- Dauer/Format: ~30 Min., remote
- Ziel: Diagnose, Scan/Read/Write, Monitoring

Kernaussagen (Kurzfassung)

- **Scan/Abort** muss stabil sein; Logs sichtbar.
- **Read/Write** mit **Read-Back-Bestätigung**; Protokoll im Log.
- **Monitoring** mit Mindestabständen; keine Bus-Kollisionen.
- Bei Fehlern: **klare Codes** plus Klartext; Parameter (Port/Baud) schnell prüfbar.

Frage	Antwort (Zusammenfassung)	Hinweis für System
Wichtigste Technikfunktion?	„Stabiler Scan, MBus hat manchmal Fehler	SSE-Log, Abort-Endpoint
Write-Bestätigung?	„Ohne Read-Back unsicher. Aktualisierung der Werte dauert etwas zu lange“	Read-Back + Write-Log
Monitoring?	„Monitoring läuft gut aber zu wenig Zeit anzeigen.“	Queue + Mindestabstände
Fehlerbild?	„Timeout, Fehlermeldungen zeigen gut.“	Fehlermeldungen + Param-Check
Sicht auf Daten?	„Tabellen und Rohwerte ok.“	Detailansicht mit Roh-/Normalwert

Ableitungen für das System

- **SSE-Log**, **Abort-Flow**, **Read-Back** verpflichtend.
- Tech-Seite mit **Parametern/Logs** sichtbar, jedoch geschützt.