Hochschule München
Fakultät für Elektrotechnik und Informationstechnik
Studiengang: Elektro- und Informationstechnik

Bachelor-Thesis

# Entwicklung eines automatisierten Monitoring-Systems für Energiedaten und PDF-Dokumente in einer Schule

verfasst von:
*Uwe Buchert*

Bearbeitungsbeginn: *01.10.2020*
Abgabetermin: *02.03.2021*
Lfd. Nr.: *2021*

Hochschule München
Fakultät für Elektrotechnik und Informationstechnik
Studiengang: *Elektro- und Informationstechnik*

Bachelor-Thesis

# Entwurf und Implementierung eines automatisierten Monitoring-Systems inklusive Datenbank für die Energiedaten und PDF-Dokumente in einer Schule

# Design and implementation of a monitoring system including a database to visualize energy data and PDF documents in a school

verfasst von: *Uwe Buchert*
betreut von: *Professor Dr. Simon Schramm*

Bearbeitungsbeginn: *01.10.2020*
Abgabetermin: *02.03.2021*
Lfd. Nr.: *2021*

**Hochschule München**
**Fakultät für Elektrotechnik und Informationstechnik**

# Erklärung des Bearbeiters:

| Buchert | Uwe |
|---|---|
| Name | Vorname |

1) Ich erkläre hiermit, dass ich die vorliegende Bachelorarbeit selbständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe.

   Sämtliche benutzte Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate sind als solche gekennzeichnet.

| Waldkraiburg, 02.03.2021 | |
|---|---|
| Ort, Datum | Unterschrift |

2) Ich erkläre mein Einverständnis, dass die von mir erstellte Bachelorarbeit in die Bibliothek der Hochschule München eingestellt wird. Ich wurde darauf hingewiesen, dass die Hochschule in keiner Weise für die missbräuchliche Verwendung von Inhalten durch Dritte infolge der Lektüre der Arbeit haftet. Insbesondere ist mir bewusst, dass ich für die Anmeldung von Patenten, Warenzeichen oder Geschmacksmustern selbst verantwortlich bin und daraus resultierende Ansprüche selbst verfolgen muss.

| Waldkraiburg, 02.03.2021 | |
|---|---|
| Ort, Datum | Unterschrift |

## Kurzzusammenfassung

Für eine Schule soll ein Konzept entwickelt werden, um den Energiehaushalt des Gebäudes für die Schüler geeignet aufzubereiten und darzustellen. Dabei bieten zwei installierte PV-Anlagen sowie ein Blockheizkraftwerk großes Potential zur Einsparung von Energie. Die Energie- und Leistungsdaten aus den Anlagen sollen gemessen, ausgewertet und auf einem Dashboard für die Schüler dargestellt werden. Dafür wird ein Gesamtkonzept inklusive der notwendigen Hardware, Software sowie einer graphischen Benutzeroberfläche für den Administrator erstellt. Das Dashboard soll für die Schüler leicht verständlich sein und sie zu einem bewussten Umgang mit Energie und Ressourcen heranführen. Das langfristige Speichern der Energiewerte wird von einer Datenbank übernommen. Zuletzt wird der Nutzen des Systems durch eine Möglichkeit erweitert, PDF-Dokumente für die Schüler einzublenden. Als Methodik zur ordentlichen Planung und Durchführung wird eine Variante des Wasserfallmodells angewendet. Die Phasen gliedern sich demnach in Analyse, Entwurf, Implementierung und Tests.



## Abstract

For a school, a concept shall be developed, to prepare and visualize the buildings energy balance suitable for its pupils. There are two photovoltaic systems and a block-type thermal power station installed, offering a large potential for saving energy. The energy and power data of these systems shall be measured, evaluated, and presented to the pupils on a dashboard. Considering that a general concept is created, including hardware, software as well as a graphical user interface for the administrator. The dashboard shall be easy for the pupils to understand and guide them to a better awareness for energy and resources. The long-term storage of energy data is handled by a proper database. Last the systems usage is expanded by the possibility to fade in PDF documents for the pupils. As a method to allow proper planning and execution, a variant of the waterfall model is used. Therefore, the phases can be divided into analysis, design, implementation, and testing.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Hintergrund und Motivation

Die Stichwörter Klimaschutz und Energiewende beschäftigen die moderne Welt nun schon seit einigen Jahren. Fachkräfte suchen deswegen stets neue Wege, um den Energieverbrauch im Alltag weiter zu senken und Energie effizienter zu nutzen. Mit etwa 35% fällt ein wesentlicher Teil des Energieverbrauchs in Deutschland auf den Gebäudebereich zurück, ebenso etwa ein Drittel der Treibhausgasemissionen. [1] Im Beruf, der Schule oder auch im privaten Umfeld ist ein gewisser Komfort mittlerweile unabdingbar geworden. Er steigert nicht nur unsere Arbeitsleistung, sondern sorgt natürlich auch für ein besseres Wohlbefinden. Um eine angenehme Raumtemperatur, Luftqualität und sogar -feuchtigkeit zu gewährleisten, müssen Heizungs-, Lüftungs-, und Klimaanlagen gerade in öffentlichen Gebäudekomplexen meistens viele Stunden am Tag arbeiten. Schließlich verbringen Menschen in Europa, laut Untersuchungen der Welt Gesundheitsorganisation (WHO), ca. 90% ihrer Lebenszeit in Gebäuden. [2]

An diesen Zahlen wird schnell deutlich, welches Potenzial in der Gebäudetechnik steckt, wenn es um Maßnahmen zum Klimaschutz geht. Bis 2050 sollen Gebäude in Deutschland nahezu klimaneutral werden – so lautet das Ziel der Bundesregierung. [1] Aus diesem Grund wurde auch mehr im Bereich der Energieeffizienz investiert:

| | 2010 | 2017 | Veränderung ggü. 2010 |
|---|---|---|---|
| Investitionen zur Steigerung der Energieeffizienz im Gebäudebestand (in Mrd. Euro) | 36,1 | 46,3 | +28,1 % |
| Umsätze durch energetische Sanierung im Gebäudebestand (in Mrd. Euro) | 62,9 | 79,0 | +25,6 % |
| Beschäftigte durch energetische Gebäudesanierung im Bestand (in 1.000 Pers.) | 521,9 | 573,1 | +9,8 % |

**Abbildung 1.1 Effekte durch Energieeffizienzmaßnahmen [3, p. 9]**

Die Abbildung 1.1 des Bundesministeriums für Wirtschaft und Energie zeigt, dass die Investitionen zur Steigerung der Energieeffizienz von 2010 auf 2017 um 28,1% gestiegen sind. Damit sind auch die Umsätze der Branche gestiegen, was den Weg für eine höhere Beschäftigungsrate im Bereich der energetischen Gebäudesanierung ebnete. Die Anzahl der Beschäftigten in diesem Bereich stieg im selben Zeitraum um 51.200 auf insgesamt 573.100. An dieser Grafik kann man gut erkennen, dass der notwendige Handlungsbedarf in der Gebäudetechnik auch durch den Staat erkannt und gefördert wurde.

Um die Klimaziele für 2050 zu erreichen, muss der absolute Verbrauch an Primärenergie gegenüber 2008 um die Hälfte gesenkt werden. [3, p. 15] Als Primärenergie bezeichnet man Energie aus Quellen, die in natürlich vorkommender Form, wie z.B. Kohle oder Erdgas, aber auch durch Energieträger wie Sonne und Wind oder Kernbrennstoffe zur Verfügung stehen. Diese Primärenergie wird dann – mit Umwandlungs- und Übertragungsverlusten – in Nutzenergie, wie z.B. Wärme, für den Verbraucher gewandelt. [4] Das Verhältnis von Wirtschaftsleistung zu Energieverbrauch wird als Energieproduktivität bezeichnet. Um aber das Wirtschaftswachstum zu halten und gleichzeitig die Klimaziele zu erreichen, ist es zwingend notwendig die Energieproduktivität zu steigern. [3, p. 11]

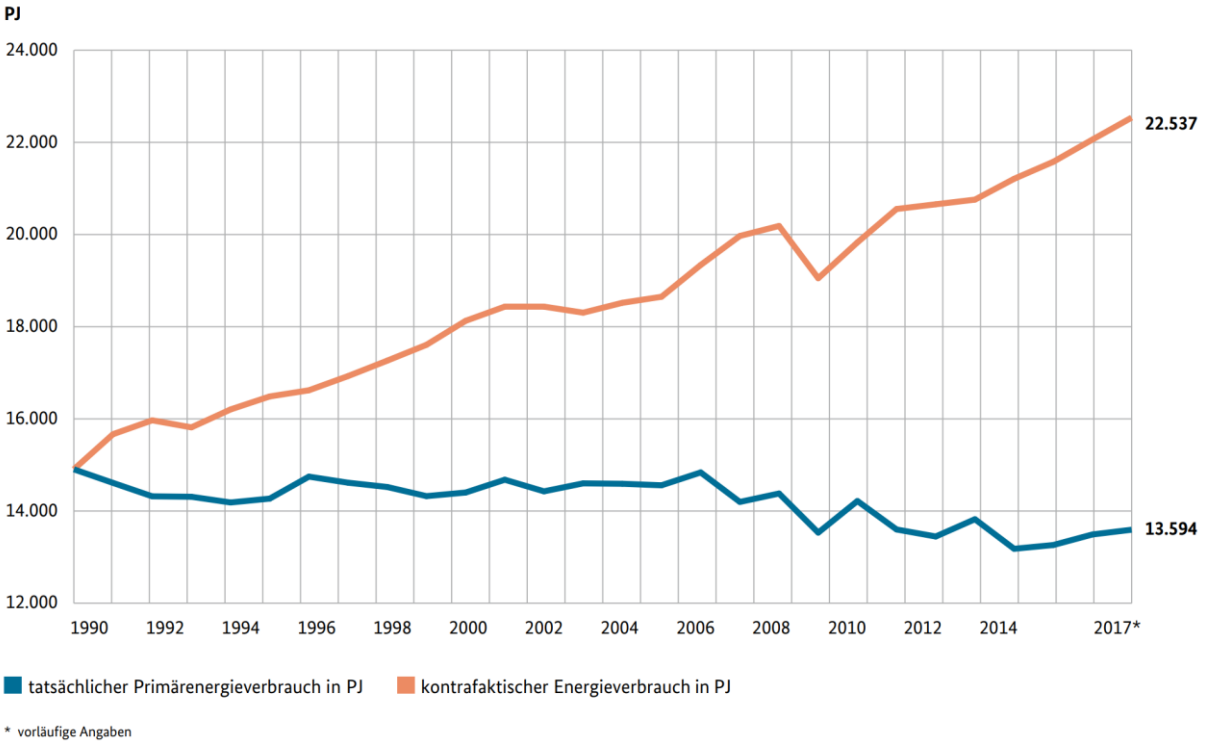


**Abbildung 1.2 Hypothetischer Primärenergieverbrauch bei gleicher Energieproduktivität [3, p. 11]**

Abbildung 1.2 zeigt den Verlauf des tatsächlichen Energieverbrauchs gegenüber dem Energieverbrauch, wenn die oben erwähnte Energieproduktivität seit 1990 gleichgeblieben wäre, beide in Petajoule (1 PJ = $10^{15}$ J). Das Ergebnis wäre fatal: Der Primärenergieverbrauch hätte sich bis 2017 auf diese Weise um fast 50% gesteigert. Der blaue Graf zeigt hingegen, was in dieser kurzen Zeit bereits erreicht wurde: Trotz stetigem Wirtschaftswachstum(!) über die Jahre konnte man den tatsächlichen Primärenergieverbrauch seit 1990 um fast 10% senken. Dies wird vor allem dadurch möglich, dass die gewonnene Energie immer besser verwertet werden kann. Ebenfalls wichtig ist die Rolle, die regenerative Energien in diesem Prozess spielen. Durch sie ist es möglich, Energie für den Menschen nutzbar zu machen, die die Natur nahezu unbegrenzt bereitstellt, ohne dabei wertvolle und stark begrenzt vorkommende Rohstoffe, wie z.B. Erdöl oder Braunkohle zu verwenden. Immer mehr Gebäude, ob privat, öffentlich oder gewerblich genutzt, nutzen zum Beispiel Photovoltaikanlagen oder Blockheizkraftwerke (BHKW), um Energie nutzbar zu machen, die sonst verloren wäre. Im Rahmen dieser Arbeit wird ein Energiemonitoring-System für eine Schule entwickelt, die solche Anlagen nutzt.

## 1.2 Aufgabenstellung

Für die Montessori Schule in Niederseeon (Bayern) soll ein Konzept entwickelt werden, um den Energiehaushalt des Gebäudes für die Schüler geeignet aufzubereiten und darzustellen. Dabei bieten zwei installierte PV-Anlagen sowie ein Blockheizkraftwerk (BHKW) großes Potential zur Einsparung von Energie. Die Daten aus den PV-Anlagen und aus dem BHKW werden zu Beginn dieses Projekts bereits von Sensoren an verschiedenen Messpunkten erfasst und in einem Notebook zu sekündlich aktualisierten csv-Dateien verarbeitet. Diese csv-Dateien sollen nun ausgewertet, und auf einem Dashboard für die Schüler dargestellt werden.

Dabei kommt es nicht auf detaillierte wissenschaftliche Diagramme an, da auf dieser Schule auch noch junge Altersstufen ab der ersten Klasse vorhanden sind, sondern eher auf eine leicht verständliche Visualisierung. Für die Schüler soll klar werden, dass es wichtig ist, in unserer heutigen modernen Zeit auf den Ressourcenverbrauch zu achten und ihn, wo nur möglich, zu minimieren. Mit einer geeigneten visuellen Darstellung des Energiehaushalts können die Schüler für diese Problematik bereits in jungen Jahren ein Bewusstsein entwickeln und mit dem Heranwachsen einen verantwortungsvollen Umgang mit Ressourcen verinnerlichen. Für den ein oder anderen kann es bestimmt auch ein Antrieb für Innovation und technologischen Fortschritt sein, zu sehen, wie ein oder zwei kleine Anlagen bereits einen merkbaren positiven Beitrag zum Energiehaushalt leisten. Auf diese Weise kann das Interesse der Kinder und Jugendlichen geweckt und ihr Erfindergeist gefördert werden.

Dafür wird ein Gesamtkonzept inklusive der notwendigen Hardware sowie einer graphischen Bedienoberfläche (GUI) für den Administrator erstellt. Ein Raspberry Pi 4 verarbeitet über Python Skripte neue Daten automatisch im Hintergrund und bietet eine GUI für den Benutzer. Das langfristige Speichern der Energiewerte wird von einer Datenbank übernommen. Die Zeitreihen-Datenbank InfluxDB erweist sich hier als sehr passend. Grafana übernimmt die eigentliche Visualisierung der Energiebilanz auf einer Weboberfläche. Ein wesentlicher Teil der Arbeit befasst sich damit, Hardware und Software für den Dauerbetrieb zu optimieren und das Management der Datenbank effizient zu gestalten. Darunter fallen auch Themen wie Kühlung der Hardware, Datensicherung und -wiederherstellung, Fehlerbehandlung und Alarmweiterleitung. Zuletzt wird der Nutzen des Systems durch eine Möglichkeit erweitert PDF-Dokumente für die Schüler einzublenden, etwa für einen Essensplan der Mensa, Busfahrpläne etc. Um diese Dokumente nicht auf derselben Seite wie das Dashboard anzuzeigen und so die Lesbarkeit zu reduzieren, wird eine Lösung implementiert, um zwischen den Dokumenten und dem Dashboard nach einer einstellbaren Zeit automatisch zu wechseln.

Diese Dokumentation wird parallel zur eigentlichen Entwicklung der Software erstellt und spiegelt alle Phasen wider. Als Methode zur ordentlichen Planung und Durchführung wird eine Variante des Wasserfallmodells angewendet. Die Phasen gliedern sich demnach wie folgt:

Analyse: Das Projekt startet mit einer genauen Analyse der derzeitigen Situation an der Schule. Außerdem werden Anforderungen an die Nutzung und an das System selbst festgehalten. Eine Recherche zu vergleichbaren Lösungen auf dem Markt wird ebenfalls in die Analyse miteinbezogen.

Entwurf: In der Entwurf-Phase wird ein Konzept für das System als Ganzes erstellt. Die passenden Hardwarekomponenten werden recherchiert und zusammengetragen. Zusätzlich wird ein erster Überblick über mögliche Software Lösungen geschaffen. Abschließend werden die einzelnen Arbeitspakete erstellt, zeitlich geplant und priorisiert.

Implementierung: Die umfangreichste Phase stellt die Implementation der genauen Funktionen dar. Zu diesem Zeitpunkt ist die benötigte Hardware zur Programmierung geliefert worden und wird nun eingerichtet. Nötige Software, wie Datenbanken, Skript Editoren und Clients werden installiert und der Programmcode geschrieben. Diese Phase geht Hand in Hand mit einer Testphase, denn Änderungen im Programmcode werden sofort auf richtige Funktion überprüft. Die Software Erstellung geschieht also direkt auf der Rechnerplattform, die die Anwendung letzten Endes auch vor Ort erledigen soll.

# 2 Analyse

Zu Beginn des Projekts wird der aktuelle Stand vor Ort festgehalten. Ist- und Soll-Zustand werden analysiert bzw. definiert und daraus die Problemstellung abgeleitet.

## 2.1 Erfassung der aktuellen Lage

In diesem Fall bietet es sich an mit einer groben Beschreibung der Schule selbst zu beginnen: Es handelt sich um die Montessori-Schule Niederseeon in 85665 Moosbach (Bayern). Aktuell lernen hier ca. 200 Schülerinnen und Schüler nach dem namensgebenden Montessori Konzept. [5] Demnach sollen die Kinder ihren eigenen Willen entwickeln und frei in ihren Entscheidungen sein. Selbstständiges Denken und Handeln wird bereits in frühen Jahren gefördert und die Lerninhalte zeitlich an die Bedürfnisse der Schüler angepasst, indem sie selbst entscheiden, wann und was sie lernen. [6] Mehr als 30 Lehrerinnen und Lehrer unterstützen die Schüler dabei. Von der ersten bis zur zehnten Klasse sind alle Jahrgänge vertreten, sodass auch die entsprechenden Schulabschlüsse erzielt werden können. [5]

In der Schule sind zwei Photovoltaikanlagen mit einer Gesamtleistung von 10kW sowie ein gasbetriebenes Blockheizkraftwerk mit einer Leistung von 6kW installiert. Photovoltaikanlagen nutzen den Photoelektrischen Effekt, indem sie eintreffendes Licht in elektrische Ladung umwandeln, die dann in Form von Gleichstrom abgegriffen werden kann. Ein eingebauter Wechselrichter wandelt diesen Gleichstrom in Wechselstrom um, wie er in unserem gewöhnlichen Stromnetz vorkommt. Ein Blockheizkraftwerk arbeitet mit einem Verbrennungsmotor ähnlich wie in den meisten Kraftfahrzeugen. Als Energiequelle wird in diesem Fall Erdgas verwendet, das ein mechanisches Gestänge antreibt. Die mechanische Bewegung des Gestänges, wird über einen Generator direkt in elektrischen Wechselstrom umgewandelt. Das besondere an einem Blockheizkraftwerk ist, dass die bei der Bewegung entstehende Abwärme leicht durch einen Wärmetauscher nutzbar gemacht werden kann und es somit sehr energieeffizient arbeitet. Nutzt man also gleichzeitig den produzierten Strom und die Wärme so sind Nutzungsgrade von bis zu ca. 90% möglich. Es sei an dieser Stelle ausdrücklich erwähnt, dass für dieses Projekt nur die elektrischen, nicht aber die wärmebezogenen Energiedaten vorliegen. Hier bestehen also noch Möglichkeiten zur Erweiterung für die Zukunft.

Als Messstation vor Ort dient ein Notebook, das die Daten aller angebrachten Sensoren sammelt und sekündlich in csv-Dateien schreibt. Für jeden neuen Tag wird eine neue csv-Datei angelegt. Diese Dateien sind lokal auf dem Notebook gespeichert, aber können für den Zugriff im Netzwerk freigegeben werden. Das Notebook erfüllt diese Funktion im Dauerbetrieb und benutzt für die Erstellung der Dateien das „OpenMUC“ Framework. Dies ist ein kostenloses Tool, das es erlaubt Sensordaten in verschiedenen Kanälen zu erfassen und in einstellbaren Intervallen in csv-Dateien zu schreiben. Auftretende Fehler, wie z.B. das Ausfallen von Messwerten, werden mit verschiedenen Fehlercodes in der Datei vermerkt. Insgesamt werden hier 34 Messgrößen ermittelt, welche bei der Implementierung dieses Projekts simuliert werden müssen; mehr dazu in Abschnitt 4.1. Eine richtige Visualisierung der erfassten Daten vor Ort ist noch nicht vorhanden, sondern wurde nur testweise erstellt.

## 2.2 Anforderungsdefinition

Aufgrund des Montessori Konzeptes kam von Seiten der Schule der Wunsch auf, etwas pädagogisch Wertvolles zum Thema der Energiewende beizutragen. Die Daten der zwei installierten Photovoltaikanlagen und des Blockheizkraftwerkes sollen geeignet visualisiert werden und den Schülern vor Augen führen, dass Energieeinsparung immer wichtiger wird. Der Einblick der Schüler in dieses Thema soll erleichtert werden, sodass sie ein Gefühl dafür entwickeln können, wie viel die vorhandenen Anlagen wirklich an Energie einsparen können und wie viel ihre Schule tatsächlich an Energie verbraucht.

### 2.2.1 Nutzungsanforderungen

Die Anforderungen an den Nutzen lassen sich grob in zwei Perspektiven unterteilen: Schüler und Administrator.

Die Schüler müssen am System in erster Linie ablesen können, wie viel Energie in einem gewissen Zeitraum erzeugt und verbraucht worden ist. Es ist besonders wichtig, auf eine gut leserliche Darstellung zu achten. Die Diagramme der Energiedaten dürfen nicht zu kompliziert aussehen und auf den ersten Blick Ergebnisse liefern. Es bietet sich an verschiedene Dashboards bereitzustellen, um das System durch die Abwechslung interessanter zu gestalten, und verschiedene Daten groß genug anzeigen zu können. Eine Möglichkeit Energiedaten greifbarer zu machen wäre es, passende Vergleiche einzubinden, die für die Schüler leichter zu verstehen sind und die unbekannten physikalischen Größen von Energie und Leistung in einen Kontext setzen. Zuletzt sollen Schüler mit dem System Einsicht in verschiedene Pläne erhalten. Es soll sich zum Beispiel ein Busfahrplan oder ein Essensplan der Schulmensa ansehen lassen.

Um welche Pläne oder Dokumente es sich genau handelt, soll der Administrator auswählen können. Da sich Dashboards und Dokumente nur verkleinert und unleserlich auf einer einzigen Anzeige darstellen lassen, soll hierfür eine Art Abwechslung oder Rotation implementiert werden. Ob und wie viele Pläne dann zusätzlich zum Dashboard angezeigt werden, soll für den Administrator einstellbar sein, ebenso die Anzeigedauer pro Dokument oder Dashboard.

Ein Beispiel: Für die Anzeige ist ein Energie Dashboard, ein Busfahrplan und ein Essensplan der Mensa vom Administrator ausgewählt. Als Dauer hat er jeweils 30, 10 und 15 Sekunden eingestellt. Das System wird dann auf einem Bildschirm im Vollbildmodus für die ersten 30 Sekunden ein Energie Dashboard anzeigen. Danach wechselt die Anzeige für 10 Sekunden auf den Busfahrplan und letztlich nochmal für 15 Sekunden auf den Essensplan. Nach diesen insgesamt 55 Sekunden beginnt die Rotation automatisch wieder von vorn, in diesem Fall also bei dem Energie Dashboard.

Der Administrator soll während dieses Vorgangs stets die komplette Kontrolle über die Rotation haben. Er muss sie stoppen, ggf. neu konfigurieren und wieder starten können. Zudem muss er in der Lage sein, den automatisierten Prozess bis zu einem gewissen Grad manuell zu bedienen: Das Starten der beiden Hauptprogramme soll ermöglicht werden, einzelne csv-Dateien müssen zu beliebigen Zeitpunkten eingelesen werden können und der Administrator soll die Möglichkeit haben manuell Backups oder

Wiederherstellungen der Datenbank durchzuführen. Die entworfenen Dashboard Layouts sollen außerdem in Form und Inhalt anpassbar bleiben.

### 2.2.2 Systemanforderungen

Die fertige Lösung soll automatisch und ohne notwendiges Eingreifen des Bedieners jeden Tag die Messwerte aus den neuen csv-Dateien einlesen, wichtige Daten für die Energiebilanz herausfiltern, eventuell bearbeiten und in die Datenbank speichern. Die Visualisierung soll über ein einfach gehaltenes Dashboard realisiert werden, basierend auf den Werten in der Datenbank.

Dies soll tagsüber im Dauerbetrieb geschehen und nachts pausiert werden; es ist also eine Art Ruhebetrieb zu implementieren. Im Dauerbetrieb müssen Änderungen der Messwerte der letzten Minuten auch auf dem Dashboard sichtbar werden, wobei eine Aktualisierung im Minutentakt ausreicht. Zudem sollen Einstellungen durch den Bediener, also zum Beispiel das Ändern der Konfiguration für die Anzeige (PDF-Pläne, Anzeigedauer, etc.), programmtechnisch umgesetzt werden. Für diese Einstellungen soll eine Grafische Benutzeroberfläche implementiert werden, die bei Bedarf vom Bediener aufgerufen werden kann. Nachts soll das System sämtliche Prozesse beenden und sie morgens wieder starten.

Im Schritt der Datenbearbeitung sollen fehlerhafte Werte bereinigt und wenn möglich durch plausible Werte korrigiert werden, sodass einzelne Fehler die Ergebnisse im Dashboard nicht beeinflussen. Treten Fehler hingegen für einen längeren Zeitraum am Stück auf, so bietet es sich an eine Benachrichtigung, z.B. per E-Mail an den Administrator zu senden, um dem Fehler auf den Grund zu gehen.

Da das System auf unbegrenzte Zeit eingesetzt werden soll, ist es nötig den Speicherplatz der Datenbank zu bedenken. Wird diese mit Minutenwerten gefüllt, so ist es leicht vorstellbar, dass nach einigen Monaten oder Jahren die Speicherkapazität mit zu vielen Datensätzen erreicht wurde oder die Performance der Datenbank darunter leidet. Es muss also dafür gesorgt werden, dass Minutenwerte, die zu weit in der Vergangenheit liegen, z.B. zu Stunden- oder Tagesdurchschnittswerten komprimiert werden.

Sollte es zu Programmabstürzen kommen oder unvorhersehbare Fehler eintreten, so müssen diese zumindest in Log-Dateien festgehalten werden, um einen Anhaltspunkt für die Fehlersuche zu liefern.

## 2.3 Lösungen auf dem Markt

In Deutschland gibt es bereits einige professionelle Lösungen zum Monitoring von Energiedaten, meistens finden Sie Anwendung in Unternehmen und Firmen, dieser Abschnitt beschränkt sich aber auf die Anwendung in Schulen und soll einige Beispiele aufgreifen. Dabei werden Gemeinsamkeiten und Unterschiede zu diesem Projekt für die Montessori Schule betrachtet und ein kurzes Resümee gezogen.

Als erstes Beispiel soll das Energie Monitoring an der Grundschule Prüfening in Regensburg dienen: Dabei werden verschiedene Messwerte in der Schule detailliert erfasst und auf einem Server gespeichert. [7] Sämtliche Energieströme im Gebäude lassen sich dann analysieren und die Auswirkung der zuvor überlegten Energiekonzepte überprüfen. Neben der HLK-Technik, die dabei maßgeblich optimiert werden kann, betont die Schule auf ihrer Website Folgendes: „Ganz wichtig ist auch die Einbindung der Schüler, Lehrer und Hausbetreuer. Bewusster Umgang mit Energie und der damit verbundene

Ressourceneinsatz trägt maßgeblich zum Erfolg des Projekts bei.“ [7] Neben Befragungen geht aus der Website der Schule jedoch nicht hervor, wie genau Schüler in den Prozess miteinbezogen werden und ob die vorhandenen Daten aus der Analyse für sie entsprechend aufbereitet werden.

Eine weitere Anwendung findet sich in Mecklenburg-Vorpommern im Landkreis Ludwigslust-Parchim: Dort wurden sogar 30 Gebäude im Rahmen eines Förderprogramms mit einem Energie Monitoringsystem ausgestattet. [8] Dieses ermöglicht es alle Energieverbrauchswerte gebündelt an einer Stelle zugänglich zu machen, z.B. über Diagramme wie Abbildung 2.1. Weiter heißt es auf der Website: „Die Schülerinnen und Schüler haben so eine Möglichkeit, sich über die Verbräuche ihrer Schule zu Informieren und können beispielsweise in Unterrichtseinheiten innerhalb der Wahlpflichtmodule, Auswertungen zu dem Thema durchführen.“ [8] Die gewonnen Daten können also tatsächlich konkrete Anwendung im Unterrichtsumfeld der Schüler finden, jedoch legt Abbildung 2.1 nahe, dass es sich hier hauptsächlich um höhere Klassen der Mittel- oder Oberstufe handeln muss, da die Diagramme detailliert und wissenschaftlich ausfallen. Das Hauptaugenmerk in der Analyse liegt hier in der Auswertung und Dokumentation der Verbräuche. Damit sollen Fehlentwicklungen rechtzeitig erkannt und geeignete Gegenmaßnahmen ergriffen werden.

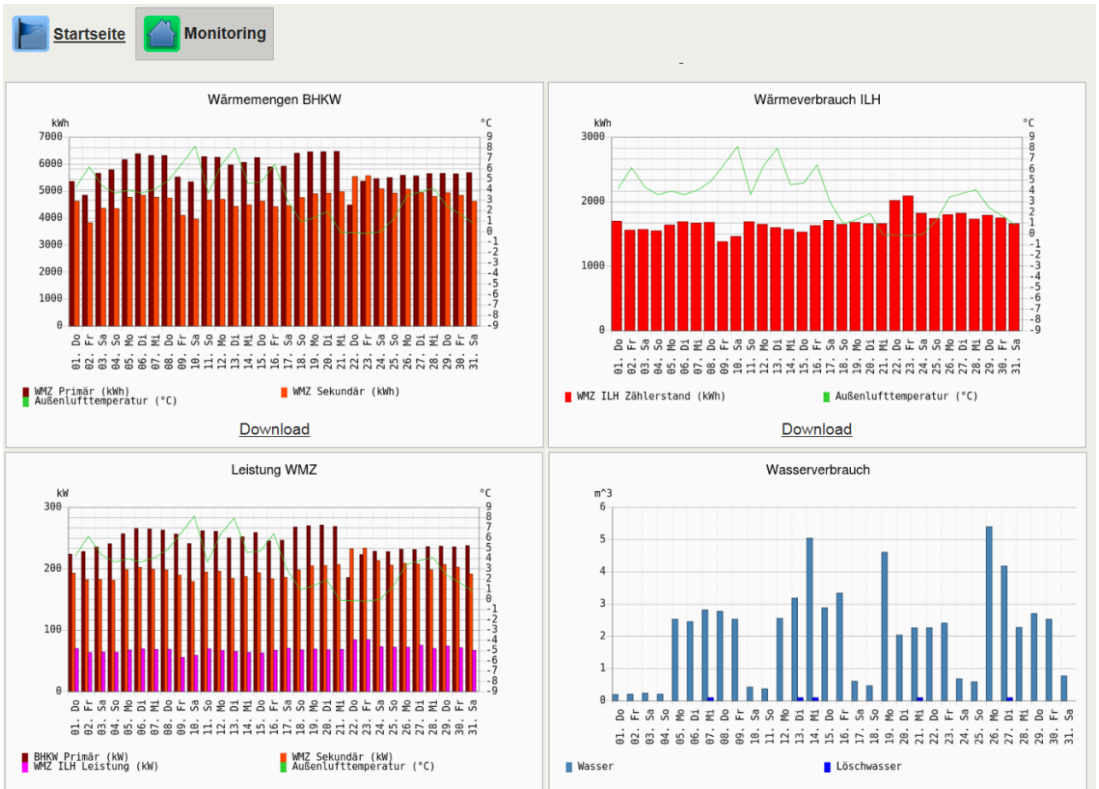


**Abbildung 2.1 Energie Monitoringsystem des Landkreises Ludwigslust-Parchim [8]**

Die letzten beiden Beispiele zeigen, dass der Fokus in Energiemonitoring Lösungen die Anwendung für den Betreiber ist. Das Ziel ist die Analyse der gesammelten Daten zum Zweck der Optimierung und der Fehlersuche in der Anlagentechnik. Außerdem sollen große Verbraucher ausfindig gemacht und so Energie Einsparmöglichkeiten gefunden werden. Dieses Ziel unterscheidet sich grundsätzlich von dem des Projektes für die Montessori Schule: Energiedaten für Schüler greifbar zu machen und ein Bewusst-

sein für den Umgang mit Energie zu schaffen. Zu erwähnen ist auch, dass die oben aufgeführten Monitoring Systeme bereits Messdaten aus der HLK-Technik miteinbeziehen. Im Projekt für die Montessori Schule liegen zurzeit nur Messdaten für den elektrischen Stromverbrauch bzw. -erzeugung vor. Eine Erweiterung für Bereiche der Heizung, Warmwasser, usw. ist sicherlich sinnvoll für die Zukunft. Daraus können dann sehr gut Pläne entwickelt werden, um vorhandene Anlagen effizienter laufen zu lassen.

Als abschließendes Beispiel sei nun noch das Energiemonitoring-System am Gymnasium Rahden in Nordrhein-Westfalen erwähnt: Es besteht aus einem Monitor und einem Stromzähler, der den elektrischen Energieverbrauch sowie Spitzen- und Grundlast misst. [9] Ein Abiturient programmierte 2014 eine Visualisierung der Daten. Seitdem veranschaulicht der Monitor im Foyer den gesamten Energieverbrauch der Schule und macht Schülern und Lehrern gleichermaßen jeden Tag bewusst, wie viel Energie im Schulalltag benötigt wird. Interessant ist hier, dass sich der Fokus erweitert hat: Es geht nun auch darum die Beteiligten visuell zu konfrontieren. Die gesammelten Daten können zudem als Grundlage für die weitere Behandlung im Unterricht dienen. [9] Die Intention dahinter deckt sich gut mit dem Ziel des Projekts in der Montessori Schule.

Zusammenfassend lässt sich sagen, dass es natürlich einige sehr gute und qualitativ hochwertige Lösungen bereits auf dem Markt gibt. Das Hauptaugenmerk liegt dabei aber oft auf der Optimierung der Anlagentechnik mit dem Ziel der Energie- und Kosteneinsparung des Betreibers. Mit dem Projekt an der Montessori Schule sollen andere Wege eingeschlagen werden: Der Fokus des Systems ist die Bildung der Schüler mit dem Ziel ein Bewusstsein für Energie zu schaffen. Die aufgebaute Datenbank kann aber auch als Grundlage für Erweiterungen dienen. Es wird großer Wert darauf gelegt zu veranschaulichen, wie wichtig regenerative Energieformen für unseren Energiehaushalt sind und wie groß ihre Auswirkungen konkret sein können. Als Ergänzung ist hier noch die Einbindung von Plänen in Form von PDF-Dokumenten vorgesehen, was somit eine Art Komplettlösung für einen Monitor in der Schulaula darstellt. Verglichen mit den oben aufgeführten Lösungen ist bei diesem Monitoring-System also mehr Interaktion von Seiten des Benutzers möglich.

# 3 Design und Entwurf

## 3.1 System Entwurf

Beim Entwurf des Systems müssen Schnittstellen der aktuellen Konfiguration vor Ort berücksichtigt werden. Wie bereits in Abschnitt 2.1 erwähnt, wurde die Datenerfassung nämlich bereits im Vorfeld implementiert. Ein Notebook vor Ort liest die Messdaten der Sensoren aus und schreibt diese im Sekundentakt in dat-Dateien, ein Dateiformat, das mit „comma separated values“ (kurz csv) arbeitet. Eine Zeile in der Datei entspricht demnach einer Sekunde, die einzelnen Spalteneinträge (insgesamt 34) entsprechen den verschiedenen Messgrößen. Welche davon wichtig werden, wird im Abschnitt 4.2 behandelt.

Als Lösung wird nun zusätzlich ein Mikrocontroller verwendet, der folgende Aufgaben übernehmen soll:

- Einlesen, Filtern und Verarbeiten der dat-Dateien
- Aufbau und Speicherung einer Datenbank aus den errechneten Werten
- Visualisierung von Daten und PDF-Plänen auf einem Monitor in der Schulaula
- Versenden von Benachrichtigungen bei ernsten Fehlern

Die Bearbeitung der Daten und Steuerung der Visualisierung wird mit Python Skripten (Version 3.7) durchgeführt. Der wesentliche Grund dafür ist, dass die Pandas Bibliothek in Python mächtige und effiziente Werkzeuge zur Verfügung stellt, um große Sammlungen von Daten zu behandeln. So können auch die zahlreichen Zeilen in den .dat-Dateien noch effizient eingelesen und zum Beispiel Aggregationen, mathematische Berechnungen und Gruppierungen durchgeführt werden. Einstellungen zum Programmablauf durch den Benutzer sind optional, die Schnittstelle dafür bildet die Konfigurationsdatei monitoring-config.ini auf dem Mikrocontroller, die durch die GUI angepasst werden kann.

Den nächsten Schritt in der Datenbearbeitung stellt die Speicherung dar. Da es sich um zeitabhängige Messwerte handelt, wird hierfür die Zeitreihen-Datenbank InfluxDB aufgesetzt. Im Vergleich zu relationalen Datenbanken haben Zeitreihen-Datenbanken den Vorteil, dass sie nur eine Dimension haben, in der Daten angeordnet werden, nämlich Zeit. Sie bieten dadurch eine hohe Effizienz im Einlesen von Daten und behalten diese auch mit voranschreitender Zeit und Größe bei. [10] Des Weiteren muss darauf geachtet werden, dass Daten nicht ewig gespeichert werden können, um dem begrenzten Speicherplatz auf dem Mikrocontroller gerecht zu werden. Minutenwerte, die z.B. Jahre in der Vergangenheit liegen, sind nicht von Interesse und sollten bzw. müssen entweder aggregiert oder gelöscht werden. Zeitreihen-Datenbanken verwenden dafür sogenannte „Retention Policies“, die – einmal eingestellt – diese Aufgabe automatisch übernehmen. [10] InfluxDB ist eine Open Source Datenbank, die aufgrund ihrer Einfachheit und starken Performance sehr beliebt bei vielen Anwendern ist. Außerdem wird sie von vielen Schnittstellen und Drittanbietern unterstützt, da sie seit 2013 auf dem Markt ist. So gibt es beispielsweise entsprechende Python Bibliotheken, die eine Benutzung erleichtern. InfluxDB wird als Client lokal auf dem Mikrocontroller gehostet und ist unter dem Port 8086 erreichbar.

Um die Daten den Schülern auch präsentieren zu können wird Grafana verwendet. Es handelt sich hierbei um ein kostenloses Visualisierungs-Tool, das mit einer Datenbank (hier InfluxDB) verknüpft wird und aus deren Daten automatisch verschiedenste Diagramme und andere Darstellungen, wie z.B. HTML Elemente, auf einem Dashboard anzeigen kann. Zusätzlich gibt es die Grafana Query Language, die es ermöglicht präzise Abfragen zu erstellen, und viele Möglichkeiten zur Automatisierung, beispielsweise Rotationen zwischen verschiedenen Dashboards oder das Senden von Berichten. Grafana wird ebenfalls lokal auf dem Mikrocontroller gehostet und ist über den Port 3000 auf einer Web-Oberfläche aufrufbar.

Für den Benutzer ist der Eingriff in Form einer grafischen Benutzeroberfläche (engl. graphical user interface = GUI) realisiert. Diese läuft allerdings nicht auf dem Mikrocontroller selbst, sondern auf einem anderen Rechner im Netzwerk, z.B. ein Rechner im Sekretariat, der oft in Benutzung ist und so einen komfortablen Eingriff in das System bietet. Die GUI kann als Python Programm aufgerufen werden, ist aber für den Betrieb nicht zwingend notwendig. Wie oben bereits beschrieben sollen hier nur PDF-Dokumente zur Anzeige eingestellt werden oder oberflächliche Eingriffe in die Datenbank erfolgen. Diese Einstellungen werden über das Python Programm in die oben erwähnte Konfigurationsdatei auf dem Mikrocontroller geschrieben, wofür also eine Netzwerkverbindung zwischen dem Mikrocontroller und dem benutzten Rechner bestehen muss. Die Hauptanwendung auf dem Mikrocontroller läuft also autark und bedarf im Regelfall keines Eingriffs. Dennoch sollte eine Fernsteuerung im verwendeten Rechner implementiert werden, alternativ ist natürlich auch die direkte Bedienung per Maus und Tastatur unter Verwendung eines Monitors am Mikrocontroller möglich.

## 3.2 Hardware Architektur

Als Mikrocontroller Board wird ein Raspberry Pi 4 Modell B in der Variante mit 4GB Arbeitsspeicher eingesetzt. Die Anwendung erfordert aufgrund der durchgehenden Datenverarbeitung, dem lokalen hosten der Datenbank InfluxDB und des Grafana Servers sowie dem Netzwerkeingriff durch den Benutzer einiges an Rechenleistung und soll dabei auch im Dauerbetrieb laufen. Der Raspberry Pi 4 ist deswegen eine naheliegende Lösung, weil er mit einem – für Mikrocontroller Boards – großzügigen Arbeitsspeicher und einer vergleichsweise hohen Taktfrequenz von 1,5GHz fast die Rechenleistung eines vollwertigen PCs bei der ungefähren Größe einer Kreditkarte besitzt. [11] Für die Anwendung im Dauerbetrieb spielt außerdem auch die Leistungsaufnahme eine wichtige Rolle; diese liegt bei Ein-Platinen-Rechnern typischerweise zwischen 2W und 8W. [12] Ein Testbericht des Online Magazins CHIP zeigt eine Leistungsaufnahme von 3W im Leerlauf und 11W „unter mutwillig erzeugter Volllast“. [13] Diese ist vergleichsweise hoch, allerdings immer noch stromsparend genug, um im Dauerbetrieb laufen zu können.

Zusätzlich bieten zwei Mikro HDMI Ausgänge die Möglichkeit bei Bedarf gleich mehrere Bildschirme zu verbinden. Es besteht somit die Möglichkeit, neben dem Monitor in der Schulaula einen weiteren, z.B. im Sekretariat, anzuschließen. Für eventuelle Erweiterungen in der Zukunft sind mit zwei USB 3.0 Ports, zwei USB 2.0 Ports, einem Gigabit LAN-Anschluss, Bluetooth 5.0 sowie 40 belegbaren GPIO Pins ausreichend Schnittstellen vorhanden. [11]

Ein Problem, das hohe Rechenleistung auf engem Raum mit sich bringt, ist die hohe Wärmeentwicklung im Bereich des Prozessors. Martin Rowan testete und protokollierte einige Szenarien auf seiner Website mit folgendem Ergebnis:

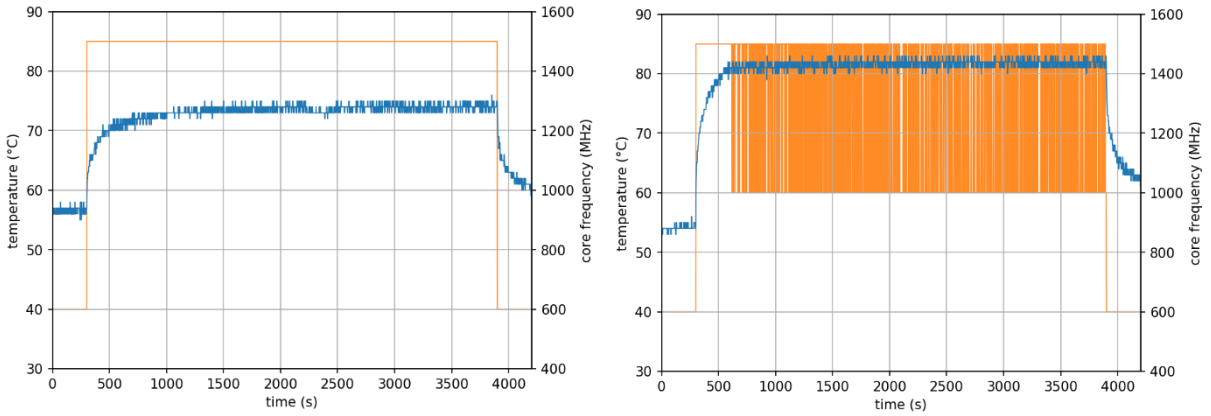


**Abbildung 3.1 Temperaturentwicklung eines Raspberry Pi 4 ohne Gehäuse, Benutzung von 2 Kernen (links) bzw. 4 Kernen (rechts) [14]**

Der Test zeigt, dass die Temperatur der CPU bei Volllast auf über 80°C steigen kann. Dies hat zur Folge, dass die Taktfrequenz des Raspberry Pi intern heruntergefahren wird, um eine Überhitzung zu vermeiden, wie in Abbildung 3.1 rechts zu sehen. Die Wärmeentwicklung ist also beachtlich und kann zu Performance Problemen führen. Daher ist es nötig, das Board und vor allem den Prozessor zu kühlen. Es gibt verschiedene Hüllen auf dem Markt, die speziell für den Raspberry Pi 4 passgenau entworfen wurden und entweder passiv mit Kühlkörpern oder aktiv mit kleinen Ventilatoren arbeiten, um die Temperaturen auf dem Board möglichst gering zu halten.

Eine sehr gute Kombination aus passiver und aktiver Kühlung stellt das „Argon One“ Gehäuse dar, das für dieses Projekt verwendet wird. Es handelt sich um ein Gehäuse aus Aluminium, das einen integrierten 30mm Lüfter verbaut hat. [15] Turmartige Blöcke im Aluminium Gehäuse sind so angeordnet, dass sie direkt an die CPU- und die RAM-Einheit des Boards andocken und werden dort mit Wärme leitenden Klebestreifen verbunden. Diese Anordnung ermöglicht auch eine effiziente passive Wärmeabfuhr. Für den Ventilator stellt die Firma Argon ein kleines Skript zum Download bereit, das dessen Drehzahl abhängig von der CPU-Temperatur schrittweise regelt. Der Download erfolgt über Raspbian in der Kommandozeile mit folgendem Befehl:

```
curl https://download.argon40.com/argon1.sh | bash
```

Die Standard Einstellungen starten den Lüfter mit 10% bei 55°C, die nächsten Stufen entsprechen 55% bei 60°C und 100% bei 65°C. Diese Standard Einstellungen erwiesen sich auch in der Testphase als nützlich und werden deshalb beibehalten. Änderungen in der Konfiguration startet man ebenfalls in der Kommandozeile mit

```
argonone-config
```

und folgt dort den simplen Anleitungen. Der Befehl

```
argonone-uninstall
```

entfernt das Skript von dem Raspberry Pi. Ein weiteres Merkmal des Argon One Gehäuses ist, dass es alle Anschlüsse des Boards auf einer Seite bündelt: Die USB-C Stromversorgung und die beiden Micro HDMI Anschlüsse werden in einer kleinen Platine im Gehäuse abgegriffen und auf die gleiche Seite wie die LAN und USB-Anschlüsse geführt, GPIO Pins bleiben unter einer magnetischen Verschlusskappe ebenfalls zugänglich und sind beschriftet. Optisch wird das Board so durch ein sauberes Erscheinungsbild aufgewertet und Kabelzuführungen erfolgen nun alle einheitlich an der Rückseite des Mikrocontrollers. Dort befindet sich außerdem ein Power Button, der ebenfalls mit dem zuvor erwähnten Skript gesteuert wird. Ein einfacher Tastendruck schaltet den Raspberry Pi ein, längeres Drücken (über drei Sekunden) fährt ihn herunter und ein zweifaches Drücken führt einen Reboot durch. Zudem kann für ein hartes, erzwungenes Ausschalten der Taster über fünf Sekunden lang gedrückt werden. Die Auswirkungen der gesamten Hülle auf den Temperaturverlauf zeigt folgende Messung von Martin Rowan:

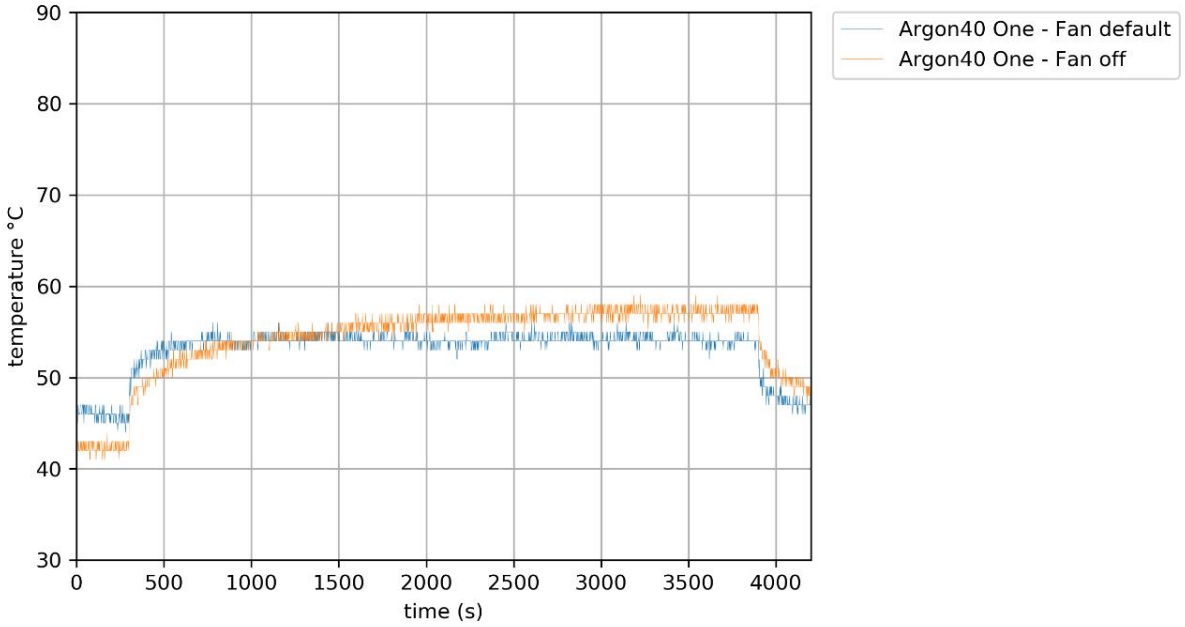


**Abbildung 3.2 Temperaturentwicklung eines Raspberry Pi 4 mit dem Argon One Gehäuse [15]**

Sogar bei ausgeschaltetem Lüfter reicht die passive Kühlwirkung durch das großflächige Aluminium Gehäuse aus, um die Temperatur der CPU bei einem Stresstest unter Volllast unterhalb von 60°C zu halten – zumindest für die Dauer von einer Stunde. Der Einsatz des Lüfters kann also im Normalbetrieb weitestgehend vermieden werden, um den Geräuschpegel nicht unnötig zu erhöhen.

Ein wesentlicher Nachteil eines Aluminium Gehäuses ist allerdings, dass es nicht nur als Kühlfläche, sondern gleichzeitig als Form eines Faraday'schen Käfigs fungiert. Das bedeutet, dass elektromagnetische Wellen von außen nur sehr schwach eindringen können. Die WLAN- und Bluetooth-Verbindung zum Raspberry Pi leiden deshalb stark darunter. Während dem Betrieb in der Schule, sollte also eine kabelgebundene LAN-Verbindung verwendet werden. Eine Bluetooth-Verbindung wird für dieses Projekt zwar nicht benötigt, kann aber für spätere Erweiterungen am besten mit USB-Bluetooth Adaptern realisiert werden.

# 4 Implementierung

## 4.1 Vorbereitung der Entwicklungsumgebung

Die Entwicklung und Implementierung der Monitoring Lösung wird im Home-Office durchgeführt. Die reale (Netzwerk-) Umgebung vor Ort muss daher möglichst gut nachempfunden werden. Der Aufbau besteht daher aus einem privaten Notebook, das jenes in der Montessori Schule nachbildet und als Datenquelle für die .dat-Dateien dient und von dem aus sich die GUI öffnen lässt, dem Raspberry Pi, der die Hauptaufgabe im System tragen wird, sowie einem Monitor, der direkt an den Mikrocontroller angeschlossen ist und für die Visualisierung sorgen wird. Der Raspberry Pi wird zusammen mit dem Notebook in einem Heimnetzwerk betrieben, das die Verbindung zwischen beiden sichert.

### 4.1.1 OpenMUC Framework

Die richtigen Messdaten von dem Notebook in der Montessori Schule liegen also nicht vor, lediglich einzelne .dat-Dateien aus dem Mai 2020 dienen als Beispielvorlage, um die genaue Struktur und das Format der Datei nachbilden zu können. Wie bereits erwähnt wird die Erzeugung dieser Dateien vor Ort durch das OpenMUC Framework realisiert. Dies ist ein kostenloses Tool des Fraunhofer Instituts für Solarenergie-Systeme in Freiburg, Deutschland. [16] Für dieses Projekt wird es in der Version v0.18.1 ausschließlich für das Schreiben von vorkonfigurierten Daten in .dat-Dateien verwendet. Reale Messinstrumente werden also nicht eingebunden.

Einige Messwerte (für etwa 150 Sekunden) werden im Vorhinein in einer Excel Tabelle per Zufallszahl aus einem plausiblen Bereich erzeugt und mit passenden Spaltenüberschriften versehen. Diese Spaltenüberschriften entsprechen jenen der Beispielvorlagen aus dem Mai 2020. Diese .csv-Datei enthält die gleichen 34 Kanäle, die in der Montessori Schule erzeugt bzw. durch Sensoren erfasst werden.

Aus diesen 34 Kanälen werden nicht alle in die Verarbeitung und die Visualisierung miteinbezogen, denn nicht alle Schüler – vor allem aus jüngeren Jahrgangsstufen – sind vertraut mit Größen wie z.B. elektrischer Spannung oder Blindleistung. Manche Größen spielen aber auch keine Rolle für die energetische Auswertung der Anlagen, wie z.B. die Oberflächentemperatur einer Siliziumzelle. Aus Gründen der Vollständigkeit werden aber dennoch alle Messgrößen in Anhang A4 aufgeführt.

Startet man das Framework so liest es aus der Datei channels.xml die Konfiguration aus, welche der oben genannten Kanäle unter welchem Namen in die .dat-Datei geschrieben werden soll und in welchem zeitlichen Abstand. Channels.xml ist nach dem Download bereits vorkonfiguriert und muss nur entsprechend angepasst werden, wie in Ausschnitt 4.1 zu sehen:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <configuration>
3    <driver id="csv">
4      <samplingTimeout>10s</samplingTimeout>
5      <connectRetryInterval>1m</connectRetryInterval>
6      <device id="home1">
7        <description/>
8        <deviceAddress>./csv-driver/energiedaten_neu.csv</deviceAddress>
9        <settings>samplingmode=line;rewind=true</settings>
10       <channel id="hhmmss">
11         <description>hhmmss</description>
12         <channelAddress>hhmmss</channelAddress>
13         <samplingInterval>1s</samplingInterval>
14       </channel>
15       <channel id="U1_L1">
16         <description>-</description>
17         <channelAddress>u1_l1</channelAddress>
18         <unit>V</unit>
19         <samplingInterval>1s</samplingInterval>
20         <loggingInterval>1s</loggingInterval>
21       </channel>
22       <channel id="U1_L2">
23         <description>-</description>
24         <channelAddress>u1_l2</channelAddress>
25         <unit>V</unit>
26         <samplingInterval>1s</samplingInterval>
27         <loggingInterval>1s</loggingInterval>
28       </channel>
```

**Ausschnitt 4.1 Kanal Einstellungen**
**- \openmuc\framework\conf \channels.xml**

In Zeile 8 wird die zuvor erwähnte Datenquelle eingetragen, in Zeile 9 erfolgt die Angabe Zeile für Zeile auszulesen und nach Erreichen des Endes wieder von vorne zu beginnen. Im Anschluss müssen für jeden der 34 Kanäle Angaben gemacht werden. Die Kanal-ID, z.B. in Zeile 15, enthält den genauen Namen, unter dem dieser Kanal in der .dat-Datei gespeichert werden soll. Hier ist es wichtig den exakt gleichen Wortlaut zu verwenden wie die Beispielvorlagen aus dem Mai 2020, damit die Entwicklungsumgebung auch mit der realen Umgebung vor Ort übereinstimmt. Die Kanal Adresse, z.B. in Zeile 17, entspricht der Spaltenüberschrift in der Datenquelle. Zudem erfolgt die Angabe des Abtast-Intervalls bei jedem einzelnen Kanal von einer Sekunde, z.B. in Zeile 19. Dieses Intervall gibt an in welchem zeitlichen Abstand Werte aus der Datenquelle erfasst werden. Mit dem Logging-Intervall, z.B. in Zeile 20, muss dann auch die Angabe erfolgen, in welchem zeitlichen Abstand die erfassten Werte in die .dat-Datei geschrieben werden. Das Logging Intervall von einer Sekunde sorgt außerdem dafür, dass die .dat-Dateien die Endung „_1000.dat“, also 1000 Millisekunden, erhalten.

Nun muss noch der Speicherort für die .dat-Dateien eingetragen werden, wie in Code Ausschnitt 4.2 in Zeile 45, zu sehen. In der Entwicklungsumgebung wird hierfür der Ordner „BA_Netzwerk“ auf dem Notebook mit dem Namen „SURFACE“ verwendet, der über eine Windows Freigabe im Netzwerk erreichbar ist und somit als Schnittstelle zum Raspberry Pi fungiert.

```
42 ##################### Ascii Logger
43 # enable file filling instead of renaming asciidata files in *.old
44 org.openmuc.framework.datalogger.ascii.fillUpFiles = true
45 org.openmuc.framework.datalogger.ascii.directory = //SURFACE/BA_Netzwerk/
```

**Ausschnitt 4.2 Speicherort der .dat-Dateien**
**- \openmuc\framework\conf\system.properties**

Das Framework kann auf dem Notebook über die Windows Batch-Datei

```
\openmuc\framework\bin\openmuc.bat
```

gestartet werden und führt im Hintergrund eine Java Applikation über die Kommandozeile aus. Näheres zur genauen Funktionsweise wird im Rahmen dieses Projekts nicht behandelt. Solange das Fenster nicht geschlossen wird und die Applikation läuft, werden über den eingebauten ASCII-Logger sekündlich Daten in die .dat-Datei geschrieben. Der resultierende Zeitstempel entspricht dem wirklichen Zeitpunkt des Schreibens, nicht dem aus der Datenquelle, sodass Quasi-Echtzeitdaten simuliert werden. Die Kanäle mit der Nr. 1 bis einschließlich 3 aus der Datenquelle (vgl. Anhang A4) werden also im Grunde genommen durch den ASCII-Logger überschrieben und sind nur für die richtige Formatierung von Bedeutung.

Es wird für jeden Tag genau eine .dat-Datei erstellt, die das entsprechende Datum im Namen enthält. Die Datei „20210212_1000.dat“ enthält z.B. alle Messwerte für den 12. Februar 2021 in einem Takt von 1000 Millisekunden. Wird der ASCII-Logger zwischendurch beendet und später neu gestartet, so wird in dieselbe Datei weitergeschrieben, wobei ausgelassene Zeitpunkte zwischen dem letzten Eintrag und dem ersten Zeitpunkt seit Neustart mit einem Fehlercode versehen werden.

### 4.1.2 Netzwerk und Schnittstellen

Der Ordner „BA_Netzwerk“ auf dem Windows Notebook enthält also die benötigten .dat-Dateien und wird später auch als Pfad für mögliche Datenbank Sicherungen dienen. Um ihn im Netzwerk verfügbar zu machen wird unter Windows 10 in den Eigenschaften des Ordners der Zugriff für jeden gewährt. Im eigenen Heimnetzwerk ist hier kein besonderer Passwortschutz nötig, beim richtigen Betrieb in der Montessori Schule sollte die Netzwerkverbindung aber zur Sicherheit mit einem Passwort geschützt werden. Anschließend wird im Netzwerk- und Freigabecenter unter erweiterte Freigabeeinstellungen die Netzwerkerkennung für das Notebook selbst eingeschaltet, erst dann ist es im Netzwerk sichtbar.

Nun muss dieser Ordner auf dem Raspberry Pi eingebunden werden. Auf dem Pi läuft das Betriebssystem Raspbian, eine Linux Variante. Netzwerkkomponenten werden unter diesem Betriebssystem über sogenannte „Mounts“ eingebunden, man sagt auch: Sie werden „gemountet“. Dazu muss zuerst ein neuer Ordner auf dem Pi angelegt werden, der als Mount-Punkt dient. Im Grunde genommen ist der Mount-Punkt also nur ein Verzeichnis im lokalen Dateisystem, in dem ein externes Dateisystem (in diesem Fall ein Windows Ordner) eingegliedert wird. Unter dem Verzeichnis /home/pi/ wird deshalb ein Ordner mit dem Namen „win_share“ erstellt.

Nun muss die Verbindung zwischen diesem Mount-Punkt und dem Ordner „BA_Netzwerk“ auf dem Notebook hergestellt werden. Dies lässt sich mit einem einzigen Befehl in der Kommandozeile des Raspberry Pi ausführen, allerdings soll dieser Mount-Prozess sogleich automatisiert werden, weshalb es sich anbietet ihn in einem kurzen Shell Skript zu implementieren. Das Shell Skript „mount_befehl.sh“ besteht dann aus folgenden zwei Zeilen:

```
#!/bin/bash
sudo mount -t cifs -o username=pi,guest
//<IP_des_Notebooks>/BA_Netzwerk /home/pi/win_share
```

Die erste Zeile ist das sogenannte „Shebang“ und gibt an mit welchem Interpreter das Skript ausgeführt werden soll, in diesem Fall als Bash Skript. Da in der Windows Freigabe zuvor kein Passwortschutz eingestellt wurde, reicht hier die Angabe als Gast für den Benutzer „pi“. Außerdem wird die IP-Adresse

des Notebooks benötigt, die also in den Netzwerkeinstellungen des Routers starr vergeben werden sollte. Der letzte Teil des Befehls enthält dann die Angabe des Ordnerpfads auf dem Raspberry Pi, der als Mount-Punkt dienen soll. Um dieses Shell Skript nun ausführbar zu machen braucht es folgenden Befehl in der Pi Kommandozeile:

```
chmod +x mount_befehl.sh
```

Nun lässt sich das Skript mit einem Doppelklick ausführen und der Ordner „BA_Netzwerk“ ist auch auf dem Raspberry Pi unter /home/pi/win_share erreichbar. Um diesen Mount Vorgang nun zu automatisieren wird eine kleine System Datei geschrieben, die unter /etc/systemd/system/ abgelegt werden sollte. Folgender Befehl öffnet einen Texteditor in der Kommandozeile und erstellt bei erstmaliger Benutzung gleichzeitig die Datei „win_connection.service“:

```
sudo nano /etc/systemd/system/win_connection.service
```

Diese Service Datei dient dazu das zuvor erstellte Shell Skript „mount_befehl.sh“ automatisch und wenn nötig wiederholt auszuführen. Bricht die Verbindung zum Notebook also zwischendurch aufgrund eines Fehlers ab und ist nach danach wieder verfügbar, so ist kein Eingreifen des Benutzers für ein erneutes Mounten nötig. Die Service Datei ist wie folgt aufgebaut:

```
1  [Unit]
2  Description=Connection to Windows Folder
3  After=multi-user.target
4  Requires=network.target
5
6  [Service]
7  Type=idle
8
9  User=pi
10 ExecStart=/home/pi/Desktop/mount_befehl.sh
11
12 Restart=always
13 RestartSec=60
14
15 [Install]
16 WantedBy=multi-user.target
```

**Ausschnitt 4.3 Automatisierung des Mount-Befehls - /etc/systemd/system/win_connection.service**

Um die Service Datei auch lesbar zu machen, müssen nun noch die nötigen Rechte dafür vergeben werden:

```
sudo chmod 644 /etc/systemd/system/win_sonnection.service
```

Da es sich um eine Service Datei handelt, die vom Betriebssystem auch als solche erkannt werden soll, muss nun abschließend der sogenannte „daemon“, der im Hintergrund Services ausführt neu geladen und der eben erstellte Service dann einmalig aktiviert werden:

```
sudo systemctl daemon-reload
sudo systemctl enable win_sonnection.service
```

Damit ist die dauerhafte Verbindung zu dem Ordner „BA_Netzwerk“, der die .dat-Dateien enthält, auch nach einem Neustart zu jedem möglichen Zeitpunkt gewährleistet. Nun muss aber beachtet werden, dass das Hauptprogramm autark auf dem Raspberry Pi laufen soll, auch wenn das Notebook nicht erreicht werden kann. In diesem Fall können dann zwar keine neuen Messwerte in die Datenbank geschrieben werden, aber die Visualisierung bereits vorhandener Daten bzw. die Steuerung der Anzeige

für PDF-Dokumente durch den Benutzer soll immer noch möglich sein. Aus diesem Grund ist es notwendig einen zweiten Ordner im Netzwerk verfügbar zu machen, der als Schnittstelle zur Programmsteuerung zwischen Raspberry Pi und dem GUI-Anwender fungiert. Dieser Ordner muss sich aber auf dem Raspberry Pi selbst befinden, damit Unterbrechungen in der Netzwerkverbindung den Programmablauf nicht stören. Der Ordner erhält den Namen „pi-share“ und wird im Verzeichnis /home/pi/ angelegt. Es bietet sich an, einen Unterordner „Programm-Daten“ zu erstellen, der rein dem Programmablauf dient und zu später die zentrale Konfigurationsdatei „monitoring_config.ini“ enthalten wird, sodass andere Dateien, wie z.B. anzuzeigende PDF-Dokumente, im Hauptordner „pi-share“ hinterlegt werden können. Der Benutzer läuft damit nicht Gefahr, versehentlich in die Programm Daten einzugreifen.

Um diesen Ordner nun im Netzwerk freizugeben benötigt man ein Softwarepaket namens „Samba“. Nach der Installation in der Kommandozeile mit

```
sudo apt-get install samba samba-common smbclient
```

ist unter dem Pfad /etc/samba/ die Konfigurationsdatei smb.conf zu finden. Diese ist nun mit folgendem Block zu ergänzen:

```
1  [Pi-Share]
2  comment=Raspberry Pi Share
3  path=/home/pi/pi-share
4  browseable=Yes
5  writeable=Yes
6  only guest=no
7  create mask=0777
8  directory mask=0777
9  public=no
```

**Ausschnitt 4.4 Samba Konfiguration - /etc/samba/smb.conf**

In Zeile 3 ist der freizugebende Ordnerpfad anzugeben, Zeile 4 und 5 vergeben Lese- und Schreibrechte für Benutzer, die auf diese Samba Freigabe zugreifen. Diese Samba Benutzer müssen ebenfalls als Nutzer auf dem Betriebssystem selbst vorhanden sein, daher bietet es sich an den Benutzer „pi“ nun auch als Samba Benutzer zu erstellen und ein Passwort zu vergeben:

```
sudo smbpasswd -a pi
```

Nach Eingabe eines Passworts und Bestätigen desselben ist der Nutzer angelegt und kann durch das Notebook verwendet werden, um Zugriff auf den freigegebenen Ordner zu erhalten. Unter Windows 10 kann man diesen Ordner nämlich unter „Dieser PC“ als Netzlaufwerk verbinden. Nach Wahl eines beliebigen Laufwerksbuchstaben ist der Ordner nun unter „Durchsuchen“ -> „RASPBERRYPI“ -> „Pi-Share“ zu finden. Mit einem Klick auf „Fertig stellen“ erscheint der Aufruf für die Eingabe des zuvor erstellten Samba Nutzers „pi“ samt Passwort. Damit ist die Grundlage der Netzwerkverbindung geschaffen.

Für den reinen Programmablauf gibt es die zentrale Konfigurationsdatei „monitoring_config.ini“, die u.a. auch Angaben zu Netzwerkeinstellungen in Form von Ordnerpfaden enthält. Diese müssen vor Ort richtig angepasst werden. Aus diesem Grund wird hier in einer Übersicht dargestellt, welche Angaben wozu dienen.

Anpassungen in der Konfigurationsdatei „monitoring_config.ini“:

`[grafana][user]` bzw. `[grafana][password]`

- Benutzername und Passwort, die von dem Browser Controller für den Login bei Grafana verwendet werden

`[grafana][dashboard1]` bzw. `[grafana][dashboard2]`

- Die URLs auf dem Raspberry Pi unter der die zwei gewünschten Grafana Dashboards aufgerufen werden können, z.B. für die heutigen Energiedaten und eine Historie der letzten Woche.

`[mail][address]`

- E-Mail-Adresse, die im Fall eines Fehlers bei den Messwerten durch den Influx Converter benachrichtigt wird

`[paths][dat_files]`

- Ordnerpfad auf dem Raspberry Pi, in dem die .dat-Dateien zu finden sind
- entspricht dem Mount- Ordner, vgl. Abschnitt 4.1.2

`[paths][usb_stick]`

- Ordnerpfad auf dem Raspberry Pi, in dem der USB-Stick zu finden ist
- Änderung nur nötig, falls dieser ausgetauscht und ein anderer Name verwendet wird

`[paths][win_dat_folder]`

- Ordnerpfad auf dem GUI-Rechner, unter dem die .dat-Dateien zu finden sind
- Man bedenke: Sowohl der Raspberry Pi als auch der GUI Anwender greifen auf diese Konfigurationsdatei zu, daher ist es nötig die Ordnerpfade für beide Systeme zu hinterlegen.

`[paths][win_prog_folder]`

- Ordnerpfad auf dem GUI-Rechner, unter dem die Programm Daten zu finden sind, also auch die Datei „monitoring_config.ini“ selbst
- Hier werden später auch Dateien durch die GUI erzeugt, die den Programm Ablauf steuern.

In dem nächsten Schritt beginnt die Entwicklung der eigentlichen Python Programme auf dem Raspberry Pi. Dabei gibt es zwei große Hauptprogramme: Der InfluxDB Converter („converter.py“) liest .dat-Dateien ein, filtert und verarbeitet die benötigten Werte und schreibt sie in die Datenbank InfluxDB, der Browser Controller („controller.py“) ist dann für die Anzeige und Steuerung eines Browserfensters zuständig, das das Grafana Dashboard und PDF-Dokumente anzeigt. Ergänzend zu diesen beiden Hauptprogrammen gibt es fünf kleinere Skripte, die für die Kommunikation mit dem Windows Rechner, die Übersichtlichkeit oder für die direkte Ausführung mancher Befehle benötigt werden.

## 4.2 InfluxDB Converter

Der InfluxDB Converter ist als Python Programm *converter.py* in dem Ordner *\home\pi\Desktop\Energiemonitoring* auf dem Raspberry Pi implementiert. Hier sind folgende Schritte enthalten:

- Einlesen und Filtern von Messdaten aus einer .dat-Datei
- Konvertierung in ein Pandas Dataframe

- Durchschnittsbildung über 60 Sekunden zu einem Minutenwert für jede Messgröße
- Errechnen von wichtigen Werten aus den einzelnen Messungen: Residuallast, erzeugte und verbrauchte Leistung, Umgebungstemperatur und Einstrahlung
- Behandlung und Markierung von fehlerhaften Messwerten
- Schreiben der errechneten Daten in die Datenbank InfluxDB
- E-Mail-Benachrichtigung bei längerem Ausfall von Messwerten oder der Netzwerkverbindung

Abbildung 4.1 veranschaulicht diese Prozesskette. In den grünen Pfeilen sind jeweils die entsprechenden Python Funktionen dargestellt, die nun grob erläutert werden, für detaillierte Erklärungen sind Kommentare im Programmcode enthalten.

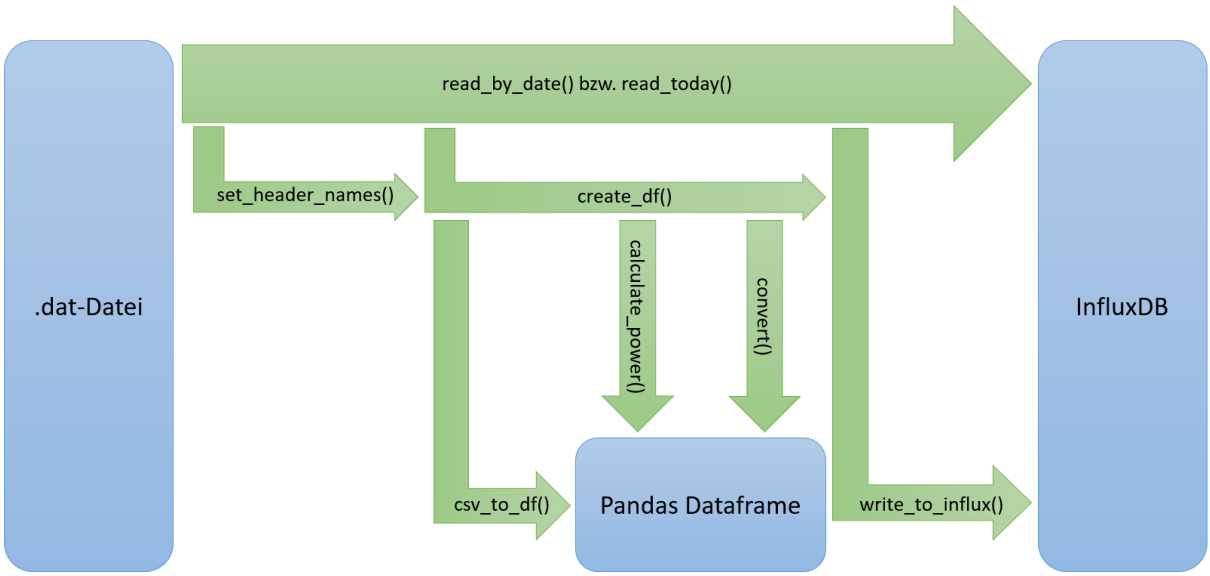


**Abbildung 4.1 converter.py – Prozesskette**

Alle aufgeführten Funktionen sind genauer gesagt Methoden der Klasse Converter. Wird das Skript „converter.py“ ausgeführt, so erstellt es zu Beginn gleich ein Objekt der Klasse Converter. Im Konstruktor werden Attribute des Objektes erstellt und definiert. Hier findet sich z.B. das Attribut *programm_folder*, das angibt unter welchem Dateipfad sich die zentrale Konfigurationsdatei „monitoring_config.ini“ befindet. Mit Hilfe der Klasse *ConfigParser* aus der gleichnamigen Python Bibliothek wird diese Konfigurationsdatei eingelesen, wichtige Angaben zu der Konfigurationsdatei sind in Abschnitt 4.1.2 beschrieben.

Die Methoden *read_by_date()* bzw. *read_today()* sind die beiden Hauptmethoden, die bei Ausführung des Skriptes aufgerufen werden. Sie rufen dann jeweils untergeordnete Methoden auf, wie in Abbildung 4.1 zu sehen. Da im richtigen Betrieb die Programme abends angehalten und morgens neu gestartet werden, müssen an jedem Tag die Abendstunden des Vortages noch eingelesen werden. Dafür liest zuerst die Methode *read_by_date()* standardmäßig immer den Vortag ein, bevor *read_today()* den heutigen Tag seit 0:00 Uhr bis zum jetzigen Augenblick einliest. Ist dies getan, so geht *read_today()* in einen Dauerbetrieb über.

Die Methode *set_header_names()* liest lediglich die Kopfzeile (Zeile 40) aus der .dat-Datei aus und speichert die Namen als Liste im Attribut *header_names* ab. Zum Lesen der .dat-Datei wird die Methode *csv.reader()* aus der importierten Bibliothek csv verwendet. Dabei kann ein eigener Dialekt als Übergabeparameter verwendet werden, der genaue Angaben enthält, wie mit dem csv Format umgegangen werden soll. Dieser Dialekt ist in einer kurzen Python Datei „my_dialect.py“ als Klasse hinterlegt. Stehen die Spaltenüberschriften fest, so wird die Methode *create_df()* und deren untergeordnete Methoden aufgerufen. Das große Einlesen der vorhandenen Messwerte geschieht innerhalb der Methode *csv_to_df()*. Sie erzeugt mithilfe der Pandas Bibliothek ein sogenanntes Dataframe, mit dem das Programm ab jetzt arbeiten und Berechnungen durchführen wird. Ein Dataframe ist eine tabellarische Datenstruktur: Die Zeilen können mit einem Index referenziert werden, die Spalten über die jeweilige Spaltenüberschrift. In dieser Anwendung werden als Indizes die Unix Zeitstempel aus den .dat-Dateien verwendet und in das Python Datetime Format konvertiert. Die Spaltenüberschriften entsprechen den zuvor erwähnten *header_names*. Nachdem so die gesamte bestehende .dat-Datei eingelesen wurde, werden in einem ersten Schritt alle Spalten aus dem Dataframe entfernt, die für die Auswertung der Energiedaten nicht relevant sind. Das Ergebnis enthält dann nur noch acht Spalten (Wirkleistungen, Umgebungstemperatur und Einstrahlung) und als Indizes die einzelnen Sekundenwerte des Tages. Befinden sich in der .dat-Datei Einträge mit einem Fehlercode, so werden diese als *np.nan* – also „not a number“ aus der Numpy Bibliothek – in dem Dataframe hinterlegt, kurz: nan-Werte. Diese können dann im Anschluss leichter verarbeitet werden als die eigentlichen textuellen Fehlercodes.

Nach *csv_to_df()* liegt also das rohe Dataframe vor und wird im nächsten Schritt von *create_df()* weiter bearbeitet: Als erstes werden die Daten einem Downsampling unterzogen, sodass für jede Spalte immer 60 Sekundenwerte zu einem einzigen Minutenwert als Durchschnitt zusammengefasst werden. Allein diese Durchschnittsbildung macht kleine Ausfälle in den Messdaten ungeschehen, da ausschließlich numerische Werte dafür verwendet werden, einzelne nan-Werte werden ignoriert. Fehlen Messdaten aber für alle 60 Sekunden, so ist das Ergebnis wieder ein nan-Wert, der auch im endgültigen Dataframe als solcher vorliegen wird.

Als nächstes wird die Methode *calculate_power()* aufgerufen. Diese berechnet aus der Summe der drei Phasen (vgl. Kanal #12 – 14 in Anhang A4) die Residuallast am Netzanschlusspunkt. Diese sagt aus, wie viel elektrische Leistung aus dem Netz angefordert wird, abhängig von der erzeugten und benötigten Leistung. Aus den gemessen Leistungswerten des BHKWs und der beiden Photovoltaikanlagen (vgl. Kanal #15, #21 und #26 in Anhang A4) wird die gesamte erzeugte Leistung als Summe berechnet. Die gesamte verbrauchte Leistung ergibt sich demnach aus der Differenz der Residuallast und der erzeugten Leistung. Abschließend werden mit Aufruf der Methode *convert()* noch die reinen Zahlenwerte für Umgebungstemperatur und Einstrahlung (vgl. Kanal #31 und #34 in Anhang A4) in die korrekten physikalischen Werte umgerechnet.

Zu erwähnen ist hier noch, dass in *create_df()* an dieser Stelle eine zusätzliche Spalte zur Fehlerkontrolle in das Dataframe eingefügt wird: Sie enthält einen binären Wert mit der Angabe, ob in der entsprechenden Zeile ein oder mehrere nan-Werte vorhanden sind. Kommt mindestens ein solcher fehlerhafter Wert vor, so wird die Zeile mit *True* markiert, fehlerfreie Zeilen erhalten ein *False*.

Für das spätere Übertragen des Dataframes in die Datenbank wird der Dataframe Client aus der Python Bibliothek InfluxDB verwendet. Bei dem eigentlichen Schreibprozess in die Datenbank haben Tests folgendes Problem ergeben: Normalerweise können nan-Werte in die Datenbank geschrieben werden, diese werden dann als leer bzw. nicht vorhanden interpretiert und liefern später z.B. in der Grafana Darstellung einfach keinen Datenpunkt. Allerdings tritt bei dem Übertragen des Dataframes in die Datenbank ein Fehler auf,

1) wenn ein Wert in der ersten Spalte des Dataframes ein nan-Wert ist.
2) wenn ein Wert in der ersten Zeile des Dataframes ein nan-Wert ist.

Aus diesem Grund sind zwei zusätzliche Eingriffe nötig, die im Programmcode deutlich kommentiert sind. Abbildung 4.2 veranschaulicht das Problem beispielhaft:

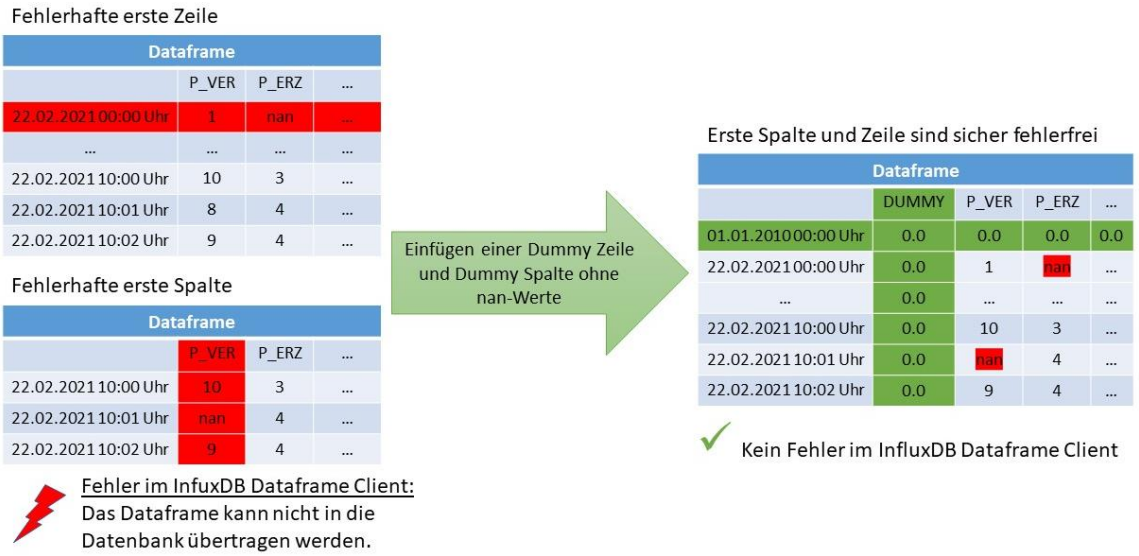


**Abbildung 4.2 Probleme im Dataframe Client**

Für die erste Spalte und erste Zeile im Dataframe werden jeweils Dummys eingefügt, die an jeder Position den numerischen Wert 0.0 enthalten und damit sicherstellen, dass oben genannte Probleme nicht auftreten können. Problem 1) wird am Ende von *create_df()* behandelt, indem also eine Dummy-Spalte an die erste Position eingesetzt wird. Im Dauerbetrieb von *read_today()* wird im Regelfall pro Minute immer ein neues kleines Dataframe erstellt, das an das bereits existierende gesamte Dataframe angehängt wird. Aus diesem Grund ist es nötig, das Einfügen der Dummy-Spalte nach jedem Downsampling in *create_df()* erneut durchzuführen, um die Dataframes aneinander anzugleichen. Damit ist also Problem 1) von oben gelöst.

Danach wird in der Hauptmethode *read_by_date()* bzw. *read_today()* die Methode w*rite_to_influx()* aufgerufen, die nun dazu dient, das erstellte Dataframe in die Datenbank zu übertragen. Zuerst wird das Problem 2) von oben behandelt, indem eine Dummy-Zeile eingefügt wird. Um diese als erste Zeile einfügen zu können, muss der Index der Zeile einem früheren Datum entsprechen als die Werte aus der .dat-Datei und das Dataframe erneut nach den Indizes sortiert werden. Damit auf diese Weise nicht unnötig viele Datenpunkte verfälscht werden, wird immer dasselbe Datum, der 01.01.2010, verwendet.

Bevor nun das eigentliche Übertragen stattfinden kann, muss noch kurz erläutert werden, wie mit den verschiedenen Retention Policies für die Datenbank verfahren werden soll. Retention Policies sind einstellbare Regelungen in Zeitreihen-Datenbanken, die bestimmen, wie lange Daten in einer bestimmten Genauigkeit gespeichert werden sollen. Wie bereits in Abschnitt 3.1 erwähnt müssen die Speicherkapazitäten des Raspberry Pi berücksichtigt werden, sodass es nicht sinnvoll ist für jedes Datum sämtliche Minutenwerte zu behalten. Zur Übersicht wird für jede Retention Policy ein eigenes *Measurement*, eine Art Datentabelle in InfluxDB, verwendet. Genaueres zum Aufbau der Datenbank ist in Abschnitt 4.5.2 beschrieben. Für diesen Teil in der Programmierung ist nur wichtig zu verstehen, dass insgesamt drei *Retention Policies* für drei verschiedene *Measurements* verwendet werden:

- Die Retention Policy *week* speichert Minutenwerte im Measurement *Messwerte* für einen Zeitraum von einer Woche.
- Die Retention Policy *year* speichert den Durchschnittswert für jeweils 15 Minuten im Measurement *Messwerte_15* für einen Zeitraum von einem Jahr.
- Die Retention Policy *forever* speichert den Durchschnittswert für jeweils 24 Stunden im Measurement *Messwerte_d* für einen unbegrenzten Zeitraum.

Da manuell auch ältere .dat-Dateien eingelesen werden können, wird also je nach zu verwendender Retention Policy eine von drei If-Anweisungen durchgeführt, die im Fall von *year* oder *forever* nochmal ein Downsampling für das Dataframe durchlaufen muss, bevor die Dummy Zeile eingefügt werden kann. Geschieht das Downsampling danach, so werden dabei auch Werte für alle Zeitpunkte (also alle 15 Minuten oder alle 24 Stunden) seit dem Zeitpunkt der Dummy-Zeile berechnet, was die Größe des Dataframes dann explodieren lässt. Das Schreiben in die Datenbank geschieht dann innerhalb der If-Anweisungen mithilfe der Methode *client.write_points()* unter Angabe der Retention Policy und des entsprechenden *Measurements* als Parameter. Eine Überprüfung auf plausible Werte ist noch nicht implementiert. Abhängig von einigen Erfahrungswerten vor Ort, wäre es durchaus sinnvoll einen gültigen Bereich für jeden Messwert anzugeben. Wird dieser Bereich dann über- oder unterschritten, so könnte man die entsprechenden Werte zu nan-Werten wandeln.

Die beiden Hauptmethoden *read_by_date()* und *read_today()* unterscheiden sich vor allem darin, dass in *read_today()* zusätzlich ein Dauerbetrieb implementiert ist. Das bedeutet das Programm wird tagsüber ununterbrochen ausgeführt, wobei alle 20 Sekunden geprüft wird, ob Werte einer neuen vollen Minute in der heutigen .dat-Datei vorhanden sind. Diese werden dann gelesen und an das bereits existierende Dataframe angehängt und in die Datenbank übertragen. Dadurch, dass das gesamte Dataframe noch gespeichert ist, kann in *csv_to_df()* die ungefähre Anzahl bereits gelesener Zeilen aus dem Dataframe bestimmt werden, sodass nur noch neue Zeilen in der .dat-Datei beachtet werden können.

Als Ergänzung für die Benutzerfreundlichkeit wird eine Fehler- bzw. Warnungs-Mail versendet, falls die Netzwerkverbindung zu dem Ordner mit den .dat-Dateien abbricht oder einige Messwerte über einen Zeitraum von 10 Minuten fehlerhaft sind. Abhängig davon, ob diese Fehler bei Programmstart oder im Dauerbetrieb auftreten, wird das Programm entweder gleich zu Beginn beendet oder läuft im Hintergrund weiter und wartet auf ein Wiederherstellen der Verbindung.

## 4.3 Browser Controller

Auf dem Raspberry Pi findet sich unter *\home\pi\Desktop\Energiemonitoring* das Python Programm *controller.py*, das den Browser Controller realisiert. Folgende Schritte werden von diesem Programm übernommen:

- Einmalig bei Programmbeginn: Setzen der Standard Konfiguration in *monitoring_config.ini*
- Regelmäßiges Prüfen der Konfigurationsdatei auf Änderungen
- Öffnen und Steuern eines Browser Fensters im Vollbild mithilfe des Selenium Webdrivers
- Öffnen von ausgewählten Grafana Dashboards und/ oder PDF-Dokumenten als einzelne Tabs
- Durchwechseln des angezeigten Tabs nach einstellbarer Zeitdauer

➔ Die Auswahl und Anzeigedauer der zu öffnenden Tabs kann über die GUI in die Konfigurationsdatei eingetragen werden.

Dies ist neben dem InfluxDB Converter das zweite große Hauptprogramm auf dem Raspberry Pi. Beide laufen normalerweise im Dauerbetrieb und separat voneinander. Abbildung 4.3 veranschaulicht das Programm in einem Schaubild:

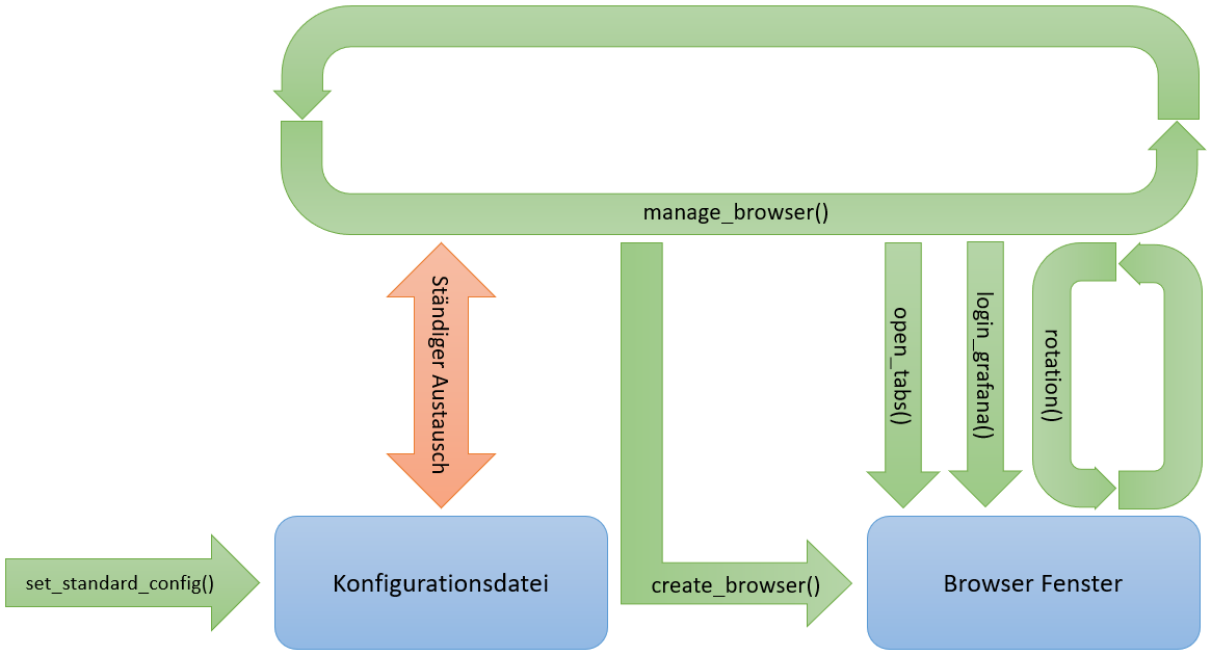


**Abbildung 4.3 controller.py – Prozesskette**

Für das Lesen und Schreiben in die Konfigurationsdatei wird die importierte Klasse ConfigParser verwendet. Da diese Datei von zentraler Bedeutung für die Ablaufsteuerung ist, werden anhand der Standardkonfiguration zuerst alle Einstellungen erläutert. Die Methode *set_start_config()* setzt bei Ausführung des Programms die folgende Konfiguration zum Starten in *monitoring_config.ini* fest:

```
cp['checkboxes'] = {'grafana1': 1,
                    'grafana2': 1,
                    'pdf1': 0,
                    'pdf2': 0,
                    'pdf3': 0}
cp['files'] = {'pdf1': "",
               'pdf2': "",
               'pdf3': "",
               'dat': "",
               'backup': ""}
cp['seconds'] = {'grafana1': "10",
                 'grafana2': "10",
                 'pdf1': "",
                 'pdf2': "",
                 'pdf3': "",}
cp['variables'] = {'rotation': 'no',
                   'restart': 'yes',
                   'browser_active': 'no',
                   'influx_timestamp': "-"}
```

**Ausschnitt 4.5 Browser Standard Konfiguration**
**- /home/pi/Desktop/Energiemonitoring/controller.py**

Unter *checkboxes* sind die boolschen Werte für die Auswahl verschiedener Tabs zu finden (1: ausgewählt, 0: nicht ausgewählt). Es werden also zu Beginn nur Grafana Dashboard 1 und Grafana Dashboard 2 ausgewählt, PDF-Dokumente sind abgewählt, da ja zu diesem Zeitpunkt auch noch keine .pdf-Datei durch den Benutzer vorgegeben wird. Unter *files* werden leere Strings als Platzhalter angelegt, an deren Stelle später u.a. Namen von bis zu drei .pdf-Dateien eingetragen werden können. An die Stelle von *dat* und *backup* rücken später Namen einzelner .dat-Dateien für das manuelle Einlesen bzw. der Ordnername eines Backups, der zur manuellen Wiederherstellung dienen soll. In der Rubrik *seconds* wird für jeden der maximal fünf anzuzeigenden Tabs die jeweilige Anzeigedauer in Sekunden angegeben. Ein leerer Wert bedeutet, dass dieser Tab nicht angezeigt werden soll. Abschließend sind unter *variables* vier Variablen aufgeführt, die zur Programmsteuerung dienen. Der Wert *influx_timestamp* ist der Zeitpunkt des letzten übertragenen Datenpunkts in die Datenbank und wird durch den InfluxDB Converter gesetzt. Die anderen drei Variablen können als boolsch interpretiert werden und ermöglichen eine gemeinsame Programmsteuerung zwischen dem Raspberry Pi und dem GUI-Anwender.

Die Hauptmethode, die als Dauerschleife ausgelegt ist, ist *manage_browser()*. Sie liest regelmäßig über den ConfigParser die oben beschriebene Konfiguration ein und kann so entweder ein erneutes Öffnen des Browser Fensters einleiten oder etwa die Rotation der Tabs in einer Schleife solange aufrechterhalten, bis sie über die GUI gestoppt wird. In der Standard Konfiguration aus Ausschnitt 4.5 wird die Variable *restart = yes* gesetzt, damit in dieser Methode zuerst ein Browser Fenster geöffnet wird. Diese Funktion ist in die untergeordnete Methode *create_browser()* ausgelagert. Dort wird der *Webdriver* aus der Selenium Bibliothek erstellt, welcher sofort ein Browser Fenster öffnet, das er später auch steuern kann. Die Variablen werden aktualisiert mit *browser_active = yes* und *restart = no*. Das Browser Fenster enthält zu diesem Zeitpunkt nur einen blanken Tab.

Ist das Browser Fenster geöffnet, so findet in *manage_browser()* die Abfrage statt, welcher Tab denn zuerst geöffnet werden soll (entsprechend der Werte in *checkboxes* aus Ausschnitt 4.5). Die Reihenfolge der Abfrage beginnt bei Grafana Dashboard 1 und endet mit PDF-Dokument 3. Ohne Änderung in der Konfiguration startet dann also direkt der erste Tab mit dem Grafana Dashboard 1. Da Grafana

Dashboards immer nur für bestimmte Nutzer zugänglich sind, muss hier zuerst der Login erfolgen. Das erledigt die kurze Methode *login_grafana()*. Sie verwendet ebenfalls den *Webdriver*, um die nötigen HTML Elemente (Benutzerfeld, Passwortfeld und Button für das Einloggen) auf der Login Seite ausfindig zu machen. Benutzer und Passwort werden dabei aus der Konfigurationsdatei ausgelesen (siehe Abschnitt 4.1.2).

Der nächste Schritt in der Hauptmethode ruft sogleich *open_tabs()* auf. Dort wird nun nacheinander überprüft welche verbleibenden Elemente, also Grafana Dashboard 1 und die drei möglichen PDF-Dokumente, ebenfalls ausgewählt wurden. Wichtig ist hier der Übergabeparameter *as_first_tab*, der folgenden Sinn hat:

- *as_first_tab = False* bedeutet: Es wurde bereits mindestens ein Tab mit einer URL geöffnet.
  - ➔ Es muss zuerst ein neuer Tab geöffnet und zu diesem gewechselt werden, bevor die gewünschte URL darin aufgerufen wird.
  - ➔ Die Variable *rotation* muss auf *yes* gesetzt werden, denn nun sind mindestens zwei Tabs vorhanden, zwischen denen gewechselt werden muss.
- *as_first_tab = True* bedeutet: Im ersten (blanken) Tab seit Öffnen des Browser Fensters wurde noch keine URL geöffnet.
  - ➔ Die gewünschte URL kann sofort in diesem Tab geöffnet werden.
  - ➔ Die Variable *as_first_tab* muss für das nächste Element auf *False* gesetzt werden, da ja soeben der erste Tab bereits benutzt wurde.

Für die PDF-Dokumente sei hier noch erwähnt, dass sie im Browser nur unter folgender URL angezeigt werden können und keine Umlaute wie „ä, ö, ü“ im Namen enthalten sollten:

```
file:///home/pi/pi-share/<Name_der_Datei>.pdf”
```

Nun sind also alle ausgewählten Tabs geöffnet und für das Abwechseln auf dem Bildschirm bereit. Eine Schleife in *manage_browser()* prüft dann die Konfigurationsdatei in jedem Durchlauf auf Änderungen durch den GUI-Anwender und startet dann die Methode *rotation()*. Innerhalb einer For-Schleife kann hier über das Attribut *webdriver.window_handles* iteriert werden, wobei jedes Objekt in *window_handles* einem Tab im geöffneten Browser entspricht. In jeder Iteration findet ein Vergleich der aktuell geöffneten URL im Tab mit allen ausgewählten URLs statt. Bei einer Übereinstimmung wird für die entsprechende Anzeigedauer gewartet, bevor der Tab am Ende der For-Schleife gewechselt wird. Es ist aber möglich, dass der GUI-Anwender während dieser Wartezeit die Rotation manuell stoppen möchte (*rotation = no*). Aus diesem Grund ist es nötig nach jedem Warten die Konfigurationsdatei auf Änderungen zu prüfen und gegebenenfalls die For-Schleife zu beenden, bevor der Tab gewechselt wird. Mit dem Ende dieser For-Schleife ist auch die Methode *rotation()* beendet und man befindet sich wieder in der inneren While-Schleife der Methode *manage_browser()*. Diese wird auch nur solange durchlaufen, wie die Variable *rotation = yes* erkannt wird. Ein Stopp-Befehl durch den GUI-Anwender bewirkt somit, dass die Dauerschleife in der Hauptmethode also wieder von vorne beginnt.

## 4.4 Grafische Benutzeroberfläche (GUI)

Die Grafische Benutzeroberfläche wird mit dem Python Programm *monitoring_gui.py* auf dem Notebook geöffnet. Der genaue Aufbau der GUI wird in Abschnitt 4.4.1 beschrieben und illustriert. Sie wird erstellt mit Tkinter, ein Python Toolkit, das einfache, aber dennoch umfangreiche Möglichkeiten für das Umsetzen von Benutzeroberflächen bietet. Es ist bereits im Lieferumfang von Python selbst enthalten und außerdem kostenlos, da keinerlei Gebühren für Lizenzen oder Verträge für die Nutzung anfallen. Abbildung 4.4 zeigt die GUI in der Standard Konfiguration (vgl. Ausschnitt 4.5):

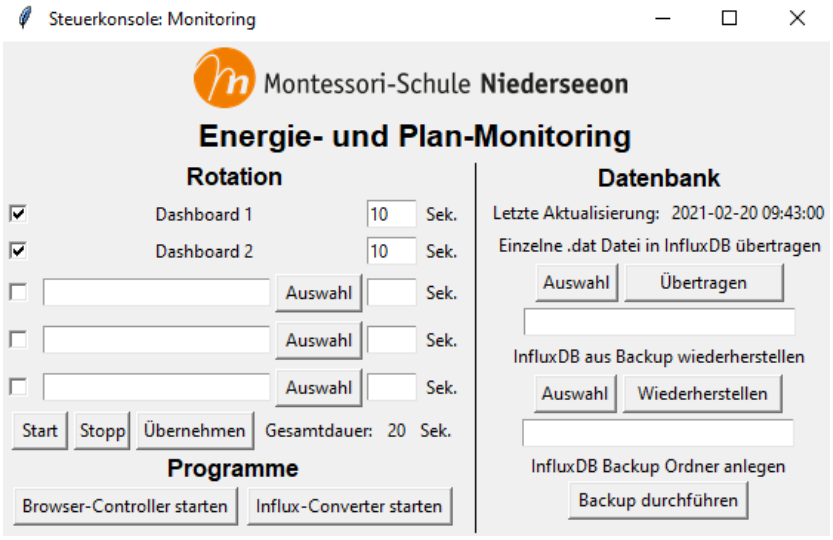


**Abbildung 4.4 Grafische Benutzeroberfläche (GUI)**

Auf der linken Seite unter „Rotation“ sind die folgenden Standard Einstellungen aus Ausschnitt 4.5 wiederzuerkennen:

- *checkboxes*: Die Haken für die beiden Dashboards sind gesetzt. Für die Anzeige von PDF-Dokumenten ist keiner der drei möglichen Haken gesetzt.
- *seconds*: Die Anzeigedauer jedes Dashboards ist auf 10 Sekunden gestellt. Für die Anzeigedauer von PDF-Dokumenten ist noch nichts vorgegeben.
- *files*: Die großen weißen Entry Felder in der GUI sind nicht befüllt, da noch keine .pdf-Datei ausgewählt wurde. Erfolgt eine Auswahl, so wird der gesamte Dateipfad unter *files* in der Konfigurationsdatei gespeichert und (aus Platzgründen) nur der Dateiname in dem Entry Feld der GUI angezeigt. Ähnlich verhält es sich bei den Feldern auf der rechten Seite unter „Datenbank“.

Dieses Programm ist optional und dient lediglich dem Benutzer dazu, folgende Einstellungen für den Browser Controller vorzunehmen und Interaktionen mit der Datenbank InfluxDB auf dem Raspberry Pi durchzuführen:

- Auswahl der anzuzeigenden Elemente (Dashboards und PDF-Dokumente) im Browser Controller unter Angabe einer Anzeigedauer
- Hinterlegen von PDF-Dateien für die Anzeige
- Stoppen und Neustarten der Rotation im Browser
- Manuelles Starten des InfluxDB Converters und des Browser Controllers

- Einlesen einer einzelnen .dat-Datei in die Datenbank
- Durchführen eines InfluxDB Backups
- Wiederherstellen der Datenbank aus einem vorhandenen Backup

Die Realisierung dieser Funktionen und das Zusammenspiel von Raspberry Pi und GUI werden in Abschnitt 4.4.2 erläutert.

### 4.4.1 Aufbau

Tkinter arbeitet mit verschiedenen Frames, die eine Art Rahmen bilden, in den dann Textfelder, Buttons usw. platziert werden können. Außerdem lassen sich einzelne Frames ineinander schachteln, was die Formatierung insgesamt deutlich erleichtert. Für die Implementierung ist es wichtig von beliebigen Stellen im Skript auf Variablen in jedem einzelnen Frame zuzugreifen. Aus diesem Grund wird die Klasse *SubFrame* erstellt, die von der Tkinter Klasse *Frame* erbt:

```
1  class SubFrame(Frame):
2      def __init__(self, master):
3          Frame.__init__(self,master)
4          self.entries = {}
5          self.labels = {}
6          self.buttons = {}
7          self.checkButtons = {}
8          self.files = ["","",""]
9
10     def add_entry(self, e_name , e_obj):
11         self.entries[e_name] = e_obj
12
13     def add_label(self, l_name, l_obj):
14         self.labels[l_name] = l_obj
15
16     def add_button(self, b_name, b_obj):
17         self.buttons[b_name] = b_obj
18
19     def add_checkButton(self, cb_name, cb_var):
20         self.checkButtons[cb_name] = cb_var
21
22     def set_file(self, f_nr, f_name):
23         self.files[f_nr-1] = f_name
```

**Ausschnitt 4.6 Implementierung der Klasse *Subframe* – */Energiemonitoring/monitoring_gui.py***

Jedes Subframe erhält also vier Python Dictionarys, die alle Elemente dieses Subframes enthalten, sowie eine Python Liste mit den Dateipfaden der übergebenen PDF-Dokumente. Bei der Erstellung muss außerdem ein *master* angegeben werden, also ein anderes Frame oder Subframe, in das es eingebettet wird (Verschachtelung). Bei Aufruf dieses Skriptes wird ein Objekt für jedes Subframe erstellt. Dafür gibt es für jedes Subframe die entsprechende Funktion *create_<frame_name>()*, in der das Subframe mitsamt Inhalt erstellt und als Rückgabewert der Funktion als Objekt zurückgegeben wird. Anhand des Beispiels für das Subframe *f_rota* wird der konkrete Nutzen im Folgenden erklärt.

```
1  def create_f_rota():
2      # ===== Frame für die Konfiguration der Rotation ==========
3      f_rota = SubFrame(root)
4      f_rota.grid(row=1)
5
6      #   ===== Checkboxes Spalte 0 ==========
7      var1 = IntVar()
8      c1 = Checkbutton(f_rota, variable=var1)
9      f_rota.add_checkButton("c1", var1)
10     c1.grid(row=1, column=0)
11
12     #   ===== Label Spalte 1 ==========
13     l_grafana1 = Label(f_rota, text="Dashboard 1")
14     f_rota.add_label("l_grafana1", l_grafana1)
15     l_grafana1.grid(row=1, column=1, columnspan=2)
16
17     #   ===== Entries Spalte 1 ==========
18     e_pdf1 = Entry(f_rota, text="", bg="#FFFFFF", width=25)
19     f_rota.add_entry("e_pdf1", e_pdf1)
20     e_pdf1.grid(row=3, column=1)
21
22     #   ===== Buttons Spalte 2 ==========
23     b_aus1 = Button(f_rota, text="Auswahl", width=7,
24                     command= lambda: click_choose(1))
25     f_rota.add_button("b_aus1", b_aus1)
26     b_aus1.grid(row=3, column=2, padx=3, pady=3)
27
28     return f_rota
```

**Ausschnitt 4.7 Erstellen eines Subframes mit verschiedenen Elementen – /Energiemonitoring/monitoring_gui.py**

Bei Programmaufruf wird die Funktion mit *f_rota = create_f_rota()* aufgerufen und erstellt in Zeile 3 sogleich das gewollte Subframe *f_rota*. Als *master* wird das Hauptfenster von Tkinter übergeben (*root*). Nachfolgend werden nun nacheinander die Elemente erstellt: Eine Checkbox ist ein Auswahlkästchen, Labels sind einfache Schriftzüge und Entries stellen veränderbare Textfelder dar. Buttons können bei einem Klick andere Funktionen aufrufen, so wird z.B. die Funktion *click_choose(1)* für den Button *b_aus1* übergeben (vgl. Zeile 23, 24). Mehr zu diesen Funktionalitäten ist in Abschnitt 4.4.2 beschrieben. Das Erstellen jedes dieser Elemente erfordert nun wiederum die Angabe eines *masters* (vgl. Zeile 8, 13, 18, 23), in diesem Fall also *f_rota*. An welche genaue Position das Element eingefügt wird bestimmt der Befehl *<element>.grid()*. Auf diese Weise werden also z.B. Buttons oder Labels einer Position im Subframe zugeordnet und das Subframe als Ganzes wiederum einer Position im Hauptfenster. Die so entstehende Verschachtelung macht es leichter, die GUI im Programmcode übersichtlich zu halten, sodass sie auch für zukünftige Erweiterungen angepasst werden kann. Ein Schema, wie die Anordnung der einzelnen Frames und Subframes realisiert wurde, ist nachfolgend in Abbildung 4.5 zu sehen.

Um nun die einzelnen Elemente zugänglich zu machen werden die zuvor erwähnten Dictionarys verwendet, die bei Erstellen der Elemente gefüllt werden. Als Beispiel betrachte man das Entry Objekt *e_pdf1*: In Ausschnitt 4.7 Zeile 19 wird dieses Entry mit der Methode f_rota.*add_entry()* in dem Dictionary *f_rota.entries* eingetragen. Als Dictionary Key wird der Name des Entrys übergeben, der Value hingegen ist das Objekt *e_pdf1* selbst. Somit lässt sich später an beliebiger Stelle im Programm auf dieses Entry Objekt einfach mit *f_rota.entries["e_pdf1"]* zugreifen, sodass dieses zunächst leere Textfeld mit dem Dateinamen ausgefüllt werden kann, nachdem ein PDF-Dokument ausgewählt wurde. Das Skript *monitoring_gui.py* ist also im Sinne einer objektorientierten Programmierung realisiert.

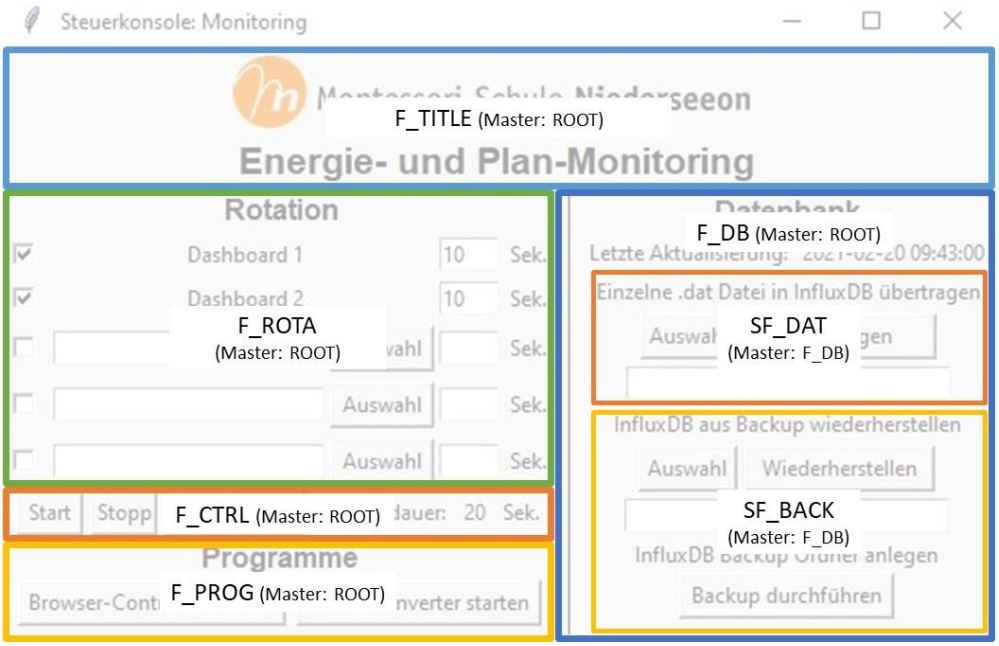


**Abbildung 4.5 Anordnung der Subframes in der GUI**

### 4.4.2 Funktionalität

Nachdem alle Frames und Subframes erstellt wurden, wird mit der Funktion *import_config_to_gui()* als erstes die aktuelle Einstellung aus der Konfigurationsdatei eingelesen und in die Felder der GUI übertragen. Dies betrifft die Auswahlkästchen, PDF-Namen und Anzeigedauer in *f_rota*, sowie den Zeitstempel der letzten Aktualisierung in *f_db*. Somit muss der Benutzer nicht bei jedem Öffnen der GUI alle gewollten Einstellungen nochmal vornehmen, sondern kann auch nur kleine Änderungen durchführen und den Rest beibehalten. Solange die GUI dann geöffnet ist, läuft ein Thread des Programms im Hintergrund, der ein Mal pro Sekunde die Zeitdauer der gesamten Rotation berechnet (vgl. Abbildung 4.5 in *F_CTRL*). Dieser wird mit Hilfe der Python Bibliothek Threading erstellt und führt die Funktion *calc_seconds()* aus, die als Dauerschleife realisiert ist. Der Thread beendet sich selbst, sobald die GUI geschlossen wird.

Auswahlkästchen und die Anzeigedauer in Sekunden lassen sich natürlich direkt in der GUI auswählen bzw. eintragen. Alle weiteren Funktionen werden über die Buttons der GUI aufgerufen. Dabei fällt auf, dass es den Auswahl-Button mehrmals in verschiedenen Frames gibt, er aber grundsätzlich überall die gleiche Funktion erfüllen soll. Aus diesem Grund werden alle Auswahl-Buttons in einer einzigen Funktion *click_choose(x)* gebündelt, wobei der Parameter x abhängig vom gewählten Button ist. Für die drei Auswahl-Buttons in *f_rota* wird jeweils entweder der Integer Wert 1, 2 oder 3 für x übergeben. Für *sf_dat* ist x=4 und für *sf_back* ist x=5. In Ausschnitt 4.7 Zeile 24 ist zu sehen, wie der Lambda Ausdruck in Python dazu verwendet wird, um jeden Button die Funktion *click_choose(x)* mit einem eigenen Parameter aufrufen zu lassen. Ausschnitt 4.8 zeigt die Funktion *click_choose(x)* und macht klar, wie genau der Parameter für die unterschiedlichen Fälle verwendet wird.

```
def click_choose(x):
    if x<4:
        # x<4 --> click_choose wurde durch einen Auswahl Button
        # für das Anzeigen von PDF Dateien aufgerufen aufgerufen
        file_path = filedialog.askopenfilename(initialdir=raspi_folder)
        file_dir, file_name = os.path.split(file_path)
        f_rota.set_file(x, file_path)
        f_rota.entries["e_pdf" + str(x)].delete(0,END)
        f_rota.entries["e_pdf" + str(x)].insert(0, file_name)
    if x==4:
        # x==4 --> click_choose wurde durch den Auswahl Button
        # für das Einlesen einer einzelnen .dat-Datei aufgerufen
        file_path = filedialog.askopenfilename(initialdir=dat_folder )
        file_dir, file_name = os.path.split(file_path)
        f_db.set_file(1,file_path)
        f_db.entries["e_dat"].delete(0,END)
        f_db.entries["e_dat"].insert(0, file_name)
    if x==5:
        # x==5 --> click_choose wurde durch den Auswahl Button
        # für die Wiederherstellung aufgerufen
        folder_path = filedialog.askdirectory(initialdir=dat_folder)
        f_db.set_file(2,folder_path)
        f_db.entries["e_restore"].delete(0,END)
        f_db.entries["e_restore"].insert(0, folder_path)
    return
```

**Ausschnitt 4.8 Funktion eines Auswahl Buttons: *click_choose(x)***
**- /Energiemonitoring/monitoring_gui.py**

Die Grundsätzliche Funktion ist also für alle Fälle gleich: Es wird ein Explorer Fenster geöffnet, in dem man einzelne Dateien oder im Fall von *sf_back* einen Ordner auswählen kann. Die Datei- bzw. Ordnerpfade werden dann in der Liste *files* eines jeden Frames gespeichert. Außerdem werden die zuvor leeren Entry Felder nun mit dem Dateinamen bzw. Ordnerpfad gefüllt, damit in der GUI ersichtlich wird, dass eine Auswahl getätigt wurde.

In dem Frame *f_ctrl* gibt es drei Buttons, mit denen die Rotation im Browser gesteuert werden kann. Der Stopp-Button pausiert die Rotation in dem momentan ausgewählten Tab mit der Funktion *click_stop()*. Nachdem Änderungen vorgenommen wurden und die Rotation gestoppt wurde, kann mit dem Übernehmen-Button die neue Einstellung über die Funktion *click_uebernehmen()* in die Konfigurationsdatei geschrieben werden. Mit dem Start-Button und der Funktion *click_start()* wird dann das existierende Browser Fenster geschlossen und ein neues geöffnet, das nun wiederum alle neu eingestellten Tabs aus der Konfigurationsdatei öffnet. Wichtig zu verstehen ist, dass nicht die GUI selbst Einfluss auf den Browser hat, sondern lediglich die nötigen Variablen in der Konfigurationsdatei setzt, auf deren Änderung dann der Browser Controller aus Abschnitt 4.3 reagiert. Auf diese Weise ist eine gemeinsame Programmsteuerung möglich.

Das Frame *f_prog* bietet zwei Buttons, mit denen die beiden Hauptprogramme auf dem Raspberry Pi gestartet werden können, falls z.B. aufgrund eines unerwarteten Fehlers ein Programm abbrechen sollte oder kein zeitgesteuerter, sondern manueller Programmstart gewünscht ist. Da sich Programme auf dem Raspberry Pi aus der Ferne nur umständlich starten lassen (z.B. über einen SSH Zugriff), wird hier folgende Methode verwendet: Über die Buttons werden lediglich Textdateien auf dem Raspberry Pi erzeugt. Dieser führt dann minütlich ein kurzes Shell-Skript namens „starter.sh“ aus, in dem dann

geprüft wird, ob bestimmte Textdateien vorhanden sind. Falls ja wird sogleich das entsprechende Programm gestartet und die vorhandene Textdatei wieder gelöscht. Zur Erinnerung: In Abschnitt 4.1.2 wurde u.a. für diesen Zweck extra der Ordner

```
/home/pi/pi-share/Programm-Daten
```

auf dem Raspberry Pi angelegt, der sowohl die Konfigurationsdatei als auch die von der GUI erstellten Textdateien enthält. Für die beiden Buttons in *f_ctrl* wird eine einzige Funktion *click_programm(x)* benutzt, die also eine Textdatei auf dem Raspberry Pi erzeugt, wobei der genaue Name der Datei als Parameter *x* übergeben wird. Der Button *Browser-Controller starten* ruft die Funktion z.B. mit dem Parameter *x="browser_start.txt"* auf.

In dem Subframe *sf_dat* wird der Übertragen-Button dazu verwendet, eine zuvor ausgewählte .dat-Datei in die Datenbank InfluxDB zu übertragen. Dafür wird mit Hilfe der Funktion *read_single_dat()* wieder eine Textdatei auf dem Raspberry Pi erzeugt, die den Pfad der gewählten .dat-Datei enthält. Der Raspberry Pi führt dann (wieder gestartet über *starter.sh*) ein eigenständiges Python Skript mit dem Namen „read_single_dat.py" aus. Aus dem Namen der .dat-Datei wird zunächst das Datum extrahiert und daraus die benötigte Retention Policy für das Schreiben in InfluxDB festgelegt. Anschließend muss der Pfad der .dat-Datei an das Dateisystem des Raspberry Pis angeglichen werden. Als erkennbare Pfade kommen entweder der Netzwerkordner, der in der Konfigurationsdatei unter

```
[paths][win_dat_folder]
```

angegeben ist, oder der vorhandene USB-Stick in Frage. Sobald diese beiden Schritte erfolgreich waren kann die Methode *read_by_date()* des InfluxDB Converters aus Abschnitt 4.2 aufgerufen werden, um die Messdaten in die Datenbank zu übertragen. Als Übergabeparameter werden der Pfad der .dat-Datei sowie die zu verwendende Retention Policy übergeben.

Das Subframe *sf_back* bietet mit dem Button *Backup durchführen* die Möglichkeit ein Backup der Datenbank InfluxDB in vordefinierte Ordner abzuspeichern. Dazu wird erneut die Funktion *click_programm(x)* mit dem Parameter *x="do_backup.txt"* verwendet. Mit dem Wiederherstellen-Button kann aus einem zuvor ausgewählten Backup Ordner die Datenbank wiederhergestellt werden. Er verwendet die Funktion *click_restore()*, um wieder eine Textdatei auf dem Raspberry Pi zu erzeugen, die den ausgewählten Backup Ordner enthält. Das eigentliche Backup und die Wiederherstellung werden wieder über *starter.sh* auf dem Raspberry Pi eingeleitet. Dazu gibt es zwei eigene Python Skripte, die beide in Abschnitt 4.5.3 genauer erklärt werden.

## 4.5 Automatisierung und Datenbank-Management

### 4.5.1 Crontab und Cronjobs

Für die zeitgesteuerte Ausführung bestimmter Befehle oder Skripte auf dem Raspberry Pi werden sogenannte „Cronjobs" verwendet. Im Hintergrund wird auf Linux Betriebssystemen permanent in einer Tabelle überprüft, welche Cronjobs als nächstes ausgeführt werden müssen. Diese Tabelle wir „Crontab" genannt und enthält die einzelnen Cronjobs. In diesem Fall wird die systemweite Crontab

des Raspberry Pis verwendet, die für alle Benutzer gilt. In dieser wurden neun Cronjobs für dieses Projekt eingetragen, sie ist zu finden unter folgendem Pfad:

```
/etc/crontab
```

Drei der neun Cronjobs sind hier als Beispiel aufgeführt, die Übrigen können in der Crontab selbst angesehen und angepasst werden:

```
00 08 * * *       root  /usr/bin/vcgencmd display_power 1
00 18 * * *       root  /usr/bin/vcgencmd display_power 0
 *  * * * *       pi    /home/pi/Desktop/starter.sh
```

Die ersten beiden Cronjobs sind dazu da, das Ausgangssignal am HDMI-Port um 08:00 Uhr morgens einzuschalten bzw. um 18:00 Uhr abends wieder auszuschalten. Der angeschlossene Bildschirm erkennt dann, ob ein Videosignal vorhanden ist oder nicht und schaltet sich abends aus und morgens wieder an. Somit wird ein großer Teil des Strombedarfs im System über die Nacht reduziert. Der Raspberry Pi selbst bleibt aber eingeschalten, um am nächsten Morgen automatisch mit dem Monitoring weiterzumachen. Der dritte Cronjob wird zu jeder vollen Minute ausgeführt, sodass das Shell-Skript *starter.sh* ein Mal pro Minute ausgeführt wird. Wie in Abschnitt 4.4.2 beschrieben, werden dadurch Programme und Skripte auf dem Raspberry Pi gestartet, die über die GUI initiiert wurden. Fünf weitere Cronjobs starten und beenden die beiden Hauptprogramme bzw. den Browser. Der letzte Cronjob führt jeden Sonntag um 19:00 Uhr ein Backup der Datenbank InfluxDB durch.

### 4.5.2 Retention Policies und Continuous Queries

*Retention Policies* in InfluxDB sind Regelungen, die bestimmen für wie lange Daten in einer Datenbank gespeichert bleiben. Es muss immer mindestens eine *Retention Policy* pro Datenbank existieren, die als Standard definiert ist. Im Fall von InfluxDB wird daher immer die Standard *Retention Policy autogen* angelegt, die Daten für eine unbegrenzte Zeitdauer speichert. Da der Speicherplatz der SD-Karte im Raspberry Pi allerdings begrenzt ist, müssen hierfür eigene Regelungen implementiert werden. Jedes Mal, wenn Daten in die Datenbank geschrieben werden geschieht dies über eine bestimmte *Retention Policy*. Wird keine angegeben, so wird diejenige verwendet, die als Standard eingestellt ist.

*Continuous Queries* sind Abfragen, genauer gesagt Kopierbefehle, die es ermöglichen große Datenmengen, wie z.B. Minutenwerte eines Tages zu einzelnen Durchschnittswerten zusammenzufassen und an anderer Stelle abzuspeichern. Sie laufen im Hintergrund und erfolgen automatisch, z.B. ein Mal pro Stunde.

Die Datenbank, in die die Messdaten geschrieben werden, heißt *energy*. Sie enthält drei einzelne Tabellen, also drei sogenannte *Measurements* mit den Namen *„Messwerte“, „Messwerte_15“* und *„Messwerte_d“*. Jedes Schreiben von Messdaten in eines dieser Measurements geschieht über eine bestimmte Retention Policy. Die Minutendaten im Measurement *Messwerte* werden regelmäßig über die zwei *Continuous Queries* einem Downsampling unterzogen und in die beiden anderen Measurements übertragen. Das Zusammenspiel von *Retention Policies* und *Continuous Queries* wird in Abbildung 4.6 veranschaulicht.

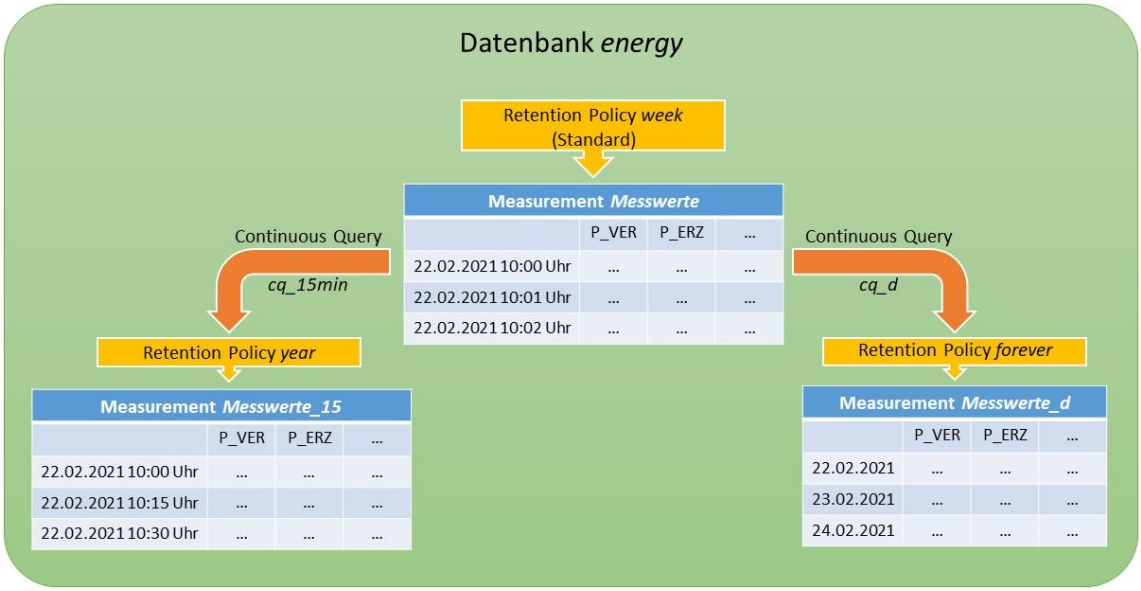


**Abbildung 4.6 Datenbank Management**

Als Beispiel zur Veranschaulichung: Wenn der InfluxDB Converter den heutigen Tag einliest und in die Datenbank schreibt, dann verwendet er dafür die Standard Retention Policy *week*, die Minutenwerte für einen zurückliegenden Zeitraum von einer Woche (week) in dem Measurement *Messwerte* speichert. Um diese Daten nicht nach einer Woche zu verlieren, wird die Continuous Query *cq_15min* erstellt. Sie kopiert jeweils einen Durchschnittswert für 15 Minuten über die Retention Policy *year* in das Measurement *Messwerte_15*. Diese Werte werden dann für einen zurückliegenden Zeitraum von einem Jahr (*year*) behalten. Die zweite Continuous Query namens *cq_d* kopiert die Durchschnittswerte für einen ganzen Tag über die Retention Policy *forever* in das Measurement *Messwerte_d*. Diese Tages Durchschnittswerte werden niemals gelöscht (*forever*), da mit nur einem einzigen Datensatz pro Tag die Speicherkapazität der SD-Karte auch nach vielen Jahren nicht erschöpft ist. Wie genau eine Continuos Query erstellt wird, ist in Ausschnitt 4.9 am Beispiel von *cq_15min* zu sehen:

```
1  CREATE CONTINUOUS QUERY cq_15min ON energy RESAMPLE EVERY 1h FOR 1h
2  BEGIN
3        SELECT
4              mean(P_L1) AS P_L1,
5              mean(P_L2) AS P_L2,
6              mean(P_L3) AS P_L3,
7              mean(IRR) AS IRR,
8              mean(P_BHKW) AS P_BHKW,
9              mean(P_ERZ) AS P_ERZ,
10             mean(P_GES) AS P_GES,
11             mean(P_PV1) AS P_PV1,
12             mean(P_PV2) AS P_PV2,
13             mean(P_VER) AS P_VER,
14             mean(T_AMB) AS T_AMB,
15             spread(ERR) AS ERR
16       INTO energy.year.Messwerte_15
17       FROM energy.week.Messwerte
18       GROUP BY time(15m)
19 END
```

**Ausschnitt 4.9 Continuous Query *cq_15min***

In Zeile 1 ist zu erkennen, dass *cq_15min* ein Mal pro Stunde ausgeführt wird und dabei alle Werte innerhalb der letzten Stunde berücksichtigt werden. Analog dazu wird *cq_d* ein Mal pro Tag für die

Werte des ganzen Tages durchgeführt. Alle Measurements, Retention Policies und Continuous Queries können über die Konsole des Raspberry Pi überprüft werden. Dazu begibt man sich mit dem Befehl

```
influx
```

zuerst in das Terminal von InfluxDB, wo man dann wiederum folgende Befehle ausführen kann:

```
use energy
show measurements
show retention policies
show continuous queries
```

### 4.5.3 Backup und Wiederherstellung der Datenbank

Das Erstellen eines Backups für InfluxDB läuft auf dem Raspberry Pi über das Python Skript „influx_backup.py“. Darin werden lediglich die folgenden zwei Befehle an die Kommandozeile des Raspberry Pi übermittelt, die die neuen Ordner für das Backup erstellen und dieses dann ausführen:

```
mkdir <Backup_Ordnerpfad>
influxd backup -portable <Backup_Ordnerpfad>
```

Das Skript wird entweder über einen Cronjob jeden Sonntag um 19:00 Uhr durchgeführt oder kann manuell über die GUI ausgeführt werden. Dabei werden zwei Backup Ordner angelegt: Einer auf dem USB-Stick im Raspberry Pi und ein weiterer in demselben Ordner auf dem Notebook, unter dem die .dat-Dateien gespeichert sind. Das sind also die Pfade, die in der Konfigurationsdatei *monitoring_config.ini* unter

```
[paths][usb_stick] und [paths][dat_files]
```

angegeben sind. Jeder Ordner enthält ein Backup der gesamten Datenbank InfluxDB, d.h. sämtliche Messwerte, Retention Policies und Continuous Queries. Der Name des Ordners trägt auch die Angabe, zu welchem Zeitpunkt das Backup getätigt wurde und setzt sich wie folgt zusammen:

```
Influx_Backup_YYYY_MM_DD_hh_mm
```

Z.B. ergibt sich für ein Backup am 23.02.2021 um 10:40 Uhr also dieser Ordnername:

```
Influx_Backup_2021_02_23_10_40
```

Das Wiederherstellen von InfluxDB aus einem Backup Ordner wird mit dem Python Skript „influx_restore.py“ realisiert. Dieses kann über die GUI mit dem Button *Wiederherstellen* ausgeführt werden, nachdem zuvor ein Backup Ordner ausgewählt wurde. Dadurch wird eine Textdatei auf dem Raspberry Pi erstellt, die den Pfad des Backup Ordners beinhaltet. Nun muss dieser Pfad zuerst an das Dateisystem des Raspberry Pi angepasst werden. Mögliche Pfade dafür sind entweder der Netzwerkordner oder der USB-Stick. Zur Veranschaulichung: Bei einem Klick auf *Wiederherstellen* schreibt der GUI-Anwender den Backup Pfad

```
C:/Users/ … /Desktop/BA_Netzwerk/Influx_Backup_2021_02_23_10_40
```

in die Textdatei. Dieser Pfad ist jedoch aus Sicht des Raspberry Pi nicht erreichbar. Allerdings wurde in Abschnitt 4.1.2 ein Mount Punkt auf dem Raspberry Pi erstellt, der genau den obigen Ordner enthält. Der Pfad dieses Mount Punktes ist folgender:

```
/home/pi/win_share
```

Die beiden Pfade sind also inhaltlich identisch und unterscheiden sich nur darin, auf welchem Dateisystem sie verwendet werden. Es muss daher überprüft werden, ob der obige Windows Pfad in der Textdatei vorkommt und falls ja, der Pfad des Mount Punktes dafür verwendet werden.

Als nächstes muss eine Unterscheidung getroffen werden, ob die Datenbank *energy* auf dem Raspberry Pi noch existiert oder nicht. Ist dies nicht der Fall, so kann sofort die Wiederherstellung mit dem Befehl

```
influxd restore -portable <Backup_Ordner>
```

erfolgen. Ist die Datenbank *energy* allerdings noch erhalten, so erlaubt es InfluxDB nicht, diese in einem Zug zu überschreiben. Es muss zuerst eine zweite temporäre Datenbank (*energy_back)* erzeugt werden, in die alle Daten aus dem Backup wiederhergestellt werden. Aus dieser werden dann mit einem Kopierbefehl alle Daten in die ursprüngliche Datenbank *energy* übertragen. Für den Kopierbefehl wird in Python der Client aus der Bibliothek *InfluxDBClient* verwendet. Dieser erlaubt es, das Kopieren in Form einer Query direkt in der Datenbank durchzuführen:

```
client.switch_database("energy_back")
client.query(select * into energy..:measurement from /.*/ group by *)
```

Damit sind nun alle Daten aus dem Backup wieder in *energy* enthalten und die temporäre Datenbank kann wieder gelöscht werden.

## 4.6 Visualisierung mit Grafana

Für die letztliche Darstellung der Energiedaten in dem Browser Fenster wird das Webtool Grafana in der Version 7.2.1 verwendet. Unter der Adresse *localhost:3000* kann die Weboberfläche nach dem Einloggen aufgerufen werden. Benutzername und Passwort sind der Konfigurationsdatei *monitoring_config.ini* zu entnehmen. Zuerst wird die Datenbank InfluxDB in den Einstellungen hinzugefügt:

```
Configuration -> Data Sources -> Add data source
```

Die Datenbank ist, wie auch Grafana, ebenfalls lokal auf dem Raspberry Pi gehostet und somit unter *localhost:8086* erreichbar. Um den Überblick über mehrere Dashboards zu behalten werden in Grafana drei Ordner angelegt, deren Dashboards sich in den Measurements unterscheiden:

- Der Ordner *Minutenwerte* enthält Dashboards, die das Measurement *Messwerte* verwenden.
- Der Ordner *15 Min. Durchschnittswerte* enthält Dashboards, die das Measurement *Messwerte_15* verwenden.
- Der Ordner *Tages-Durchschnittswerte* enthält Dashboards, die das Measurement *Messwerte_d* verwenden.

In diesem Abschnitt wird nun der Aufbau von Dashboard 2 erläutert. Dabei handelt es sich um einen Rückblick von zwei Wochen, auf dem Leistungswerte, gesamte Energie und die prozentuale Energieeinsparung dargestellt sind. Hierfür werden 15 Minuten Durchschnittswerte verwendet. Dashboard 1 hingegen ist ähnlich aufgebaut, enthält aber lediglich die Daten für den heutigen Tag und verwendet dazu Minutenwerte. Abbildung 4.7 zeigt Grafana Dashboard 2 zur Übersicht. In der rechten oberen Ecke ist zu erkennen, dass dieses Dashboard alle fünf Sekunden die Werte aus der Datenbank neu einliest und sich so aktualisiert. Die dargestellten Messdaten sind simulierte Daten des OpenMUC Frameworks.

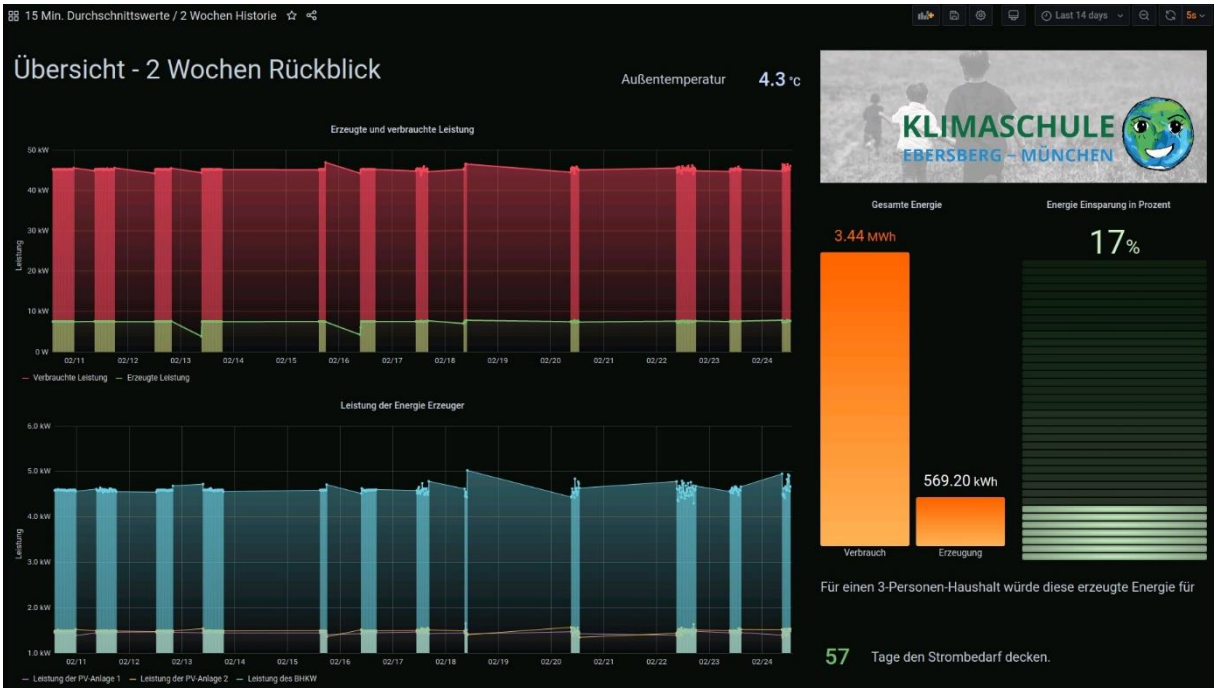


**Abbildung 4.7 Grafana Dashboard 2 – Rückblick über zwei Wochen**

Jedes Grafana Dashboard besteht aus einer schwarzen Fläche, die mit einzelnen Panels befüllt wird. Ein Panel ist ein vielseitig anpassbares Objekt, das entweder Messdaten in Form von Graphen darstellt oder mit einem Bild oder Text in Form von HTML-Code das Dashboard gliedert und übersichtlich hält. Für jedes Panel, das Messdaten darstellen soll, muss zuerst die Datenquelle InfluxDB ausgewählt werden. Es lassen sich dann eine oder mehrere Queries anlegen, um verschiedene Messreihen in dem Panel anzuzeigen. Ein Beispiel zeigt Abbildung 4.8 für die verbrauchte und erzeugte Leistung:


**Abbildung 4.8 Konfiguration einer Influx Query in einem Grafana Panel**

In dem unteren Teil von Abbildung 4.8 (Query A) ist zu sehen, wie sich vorkonfigurierte InfluxDB Queries einstellen lassen, um die gewünschten Daten zu erhalten. In diesem Fall wird aus der Retention Policy *year* und dem Measurement *Messwerte_15* das Feld *P_VER* ausgewählt, das die verbrauchte Leistung enthält (im Graph rot). *P_VER* wurde durch den InfluxDB Converter berechnet und in die Datenbank geschrieben. Ohne weitere Angabe unter *Group By* werden alle einzelnen Datenpunkte aus dem oben rechts ausgewählten Zeitraum im Graphen angezeigt. Die Query B wurde auf dieselbe Weise wie A erstellt aber nach der Fertigstellung minimiert, sodass die reine InfluxDB Query in Textform sichtbar wird. Sie enthält die erzeugte Leistung *P_ERZ.* Die Variable *$timeFilter* wird dabei automatisch erstellt. Die weiteren Einstellungen für das gesamte Panel lassen sich weiter rechts vornehmen (in Abbildung 4.8 nicht mehr zu sehen). Sie betreffen z.B. Achsenbeschriftungen, Panel-Titel, Einheiten, Auswahl der Diagrammform und einige weitere designspezifische Punkte. Wichtig für die korrekte Darstellung ist, für die Messdaten die Einheit Watt auszuwählen, da der InfluxDB Converter alle Leistungswerte in der Grundeinheit Watt berechnet und so in die Datenbank schreibt. Das Panel für die Leistung der einzelnen Energieerzeuger lässt sich völlig analog erstellen.

Als Anmerkung: Die großen Lücken in den Graphen sind dadurch bedingt, dass es sich immer noch um simulierte Werte handelt und das Notebook mit dem OpenMUC Framework im Home-Office nicht durchgehend eingeschaltet war. Es werden also nur Datenpunkte für die Zeit erzeugt, in der das OpenMUC Framework auf dem Notebook läuft. Wird das Framework z.B. morgens gestartet, mittags pausiert und am Nachmittag erneut gestartet, so enthält die erzeugte .dat-Datei Fehlercodes für die Mittagszeit, die durch den InfluxDB Converter zu nan-Werten (also *not a number*) umgewandelt und in Grafana nicht dargestellt werden.

Das Panel für die Außentemperatur bildet insofern eine Ausnahme als dass es Minutenwerte aus dem Measurement *Messwerte* und der Retention Policy *week* verwendet. Der Grund dafür ist, dass es wenig Sinn macht die durchschnittliche Temperatur der letzten beiden Wochen anzuzeigen. Viel mehr interessiert die aktuelle Außentemperatur. Deswegen wird in den Panel Einstellungen unter

```
Panel -> Display -> Calculation
```

die Option *Last (not null)* ausgewählt, die den letzten aktuellen Datenpunkt anzeigt. Zudem werden einige Grenzwerte gesetzt, die die Farbe der Anzeige mit steigender Temperatur von einem kalten dunkelblau zu einem heißen rot ändern. Das Titelbild in der rechten oberen Ecke von Abbildung 4.7 wird in einem extra Panel als HTML-Code eingebunden. Dazu wird nur die URL des Bildes benötigt.

Für komplexere Abfragen reicht die simple vorkonfigurierte InfluxDB Query in Abbildung 4.8 nicht mehr aus bzw. ist nicht ganz eindeutig, sodass die Query besser manuell geschrieben wird. Dies ist u.a. der Fall bei dem Panel für die verbrauchte und erzeugte Energie, das in Abbildung 4.9 betrachtet wird.

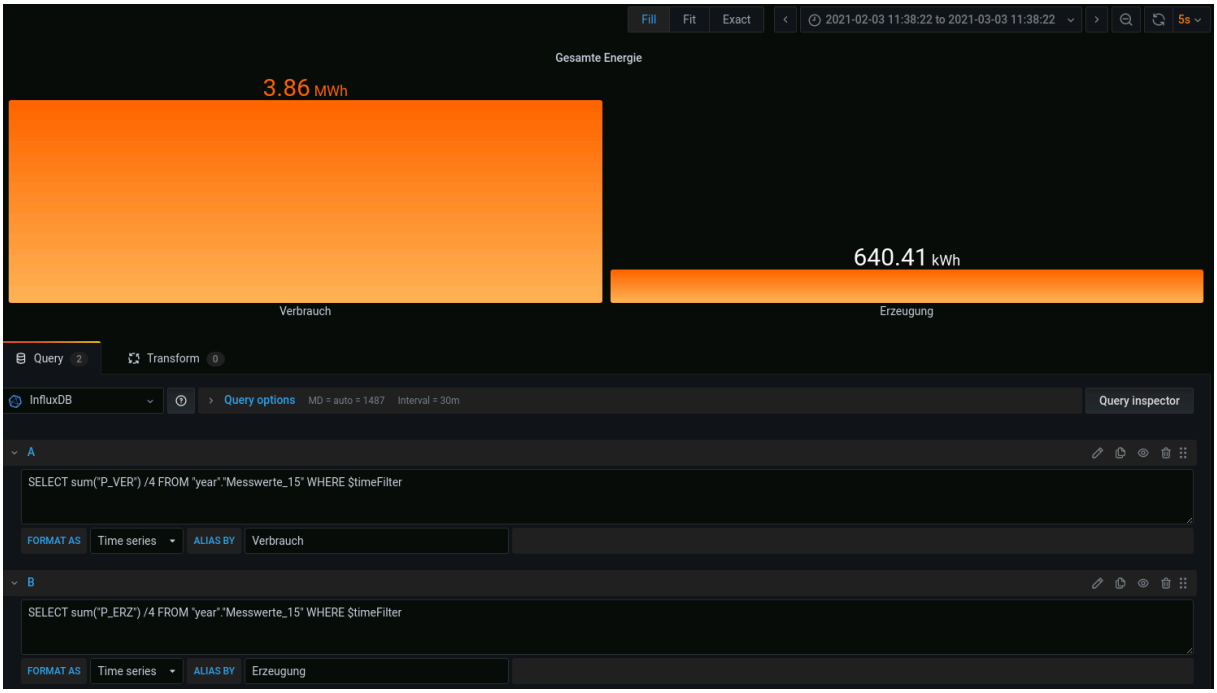


**Abbildung 4.9 Erstellen einer komplexen Influx Query in einem Grafana Panel**

Für die Berechnung der gesamten verbrauchten und erzeugten Energie müssen die Leistungswerte über die Zeit integriert werden, denn Energie entspricht einer Leistung über einer bestimmten Zeit. Dieses Integrieren erfolgt für diskrete Zeitpunkte in Form einer Multiplikation von Leistungswert und Zeitraum für jeden Datenpunkt und anschließendes aufsummieren der Ergebnisse. Im Measurement *Messwerte_15* stehen nun die durchschnittlichen Leistungswerte für jeweils 15 Minuten zur Verfügung. Das bedeutet jeder Datenpunkt muss mit der Zeiteinheit von 0,25 Stunden multipliziert werden, um als Ergebnis eine Energie in Wattstunden zu erhalten. Summiert man also alle Leistungswerte auf und multipliziert das Ergebnis mit den 0,25 Stunden, so erhält man die gesamte Energie für den betrachteten Zeitraum von zwei Wochen. Dieses Vorgehen wird in Query A für die verbrauchte Energie und in Query B für die erzeugte Energie durchgeführt.

Die prozentuale Einsparung ist als Balkendiagramm von 0 – 100% in einem weiteren Panel implementiert. Sie errechnet sich aus dem gesamten Verhältnis der erzeugten Energie zu der verbrauchten Energie. Da jedoch beide denselben Zeitraum von zwei Wochen abdecken, kann auch direkt das Verhältnis der aufsummierten Leistungswerte verwendet werden. Auch hier werden, ähnlich wie bei der Außentemperatur, Grenzwerte bei 25%, 50% und 75% gesetzt, die das Diagramm farblich etwas aufwerten.

Um für die Schüler diese Energiewerte in einen konkreten Kontext zu setzen, wird in Abbildung 4.7 unten rechts eine Anzahl an Tagen berechnet, für die diese erzeugte Energie bei einem durchschnittlichen Haushalt von drei Personen den gesamten Strombedarf decken würde. Der Abschnitt besteht aus zwei Panels, die einen HTML-Text enthalten, und einem weiteren, das einen einzigen Wert anzeigt. Auf der Internetplattform Statista ist die folgende Grafik für 2019 zu finden, die den durchschnittlichen jährlichen Stromverbrauch von Haushalten mit drei Personen abhängig von der genauen Wohnsituation und einer vorhandenen elektrischen Warmwasserbereitung angibt:

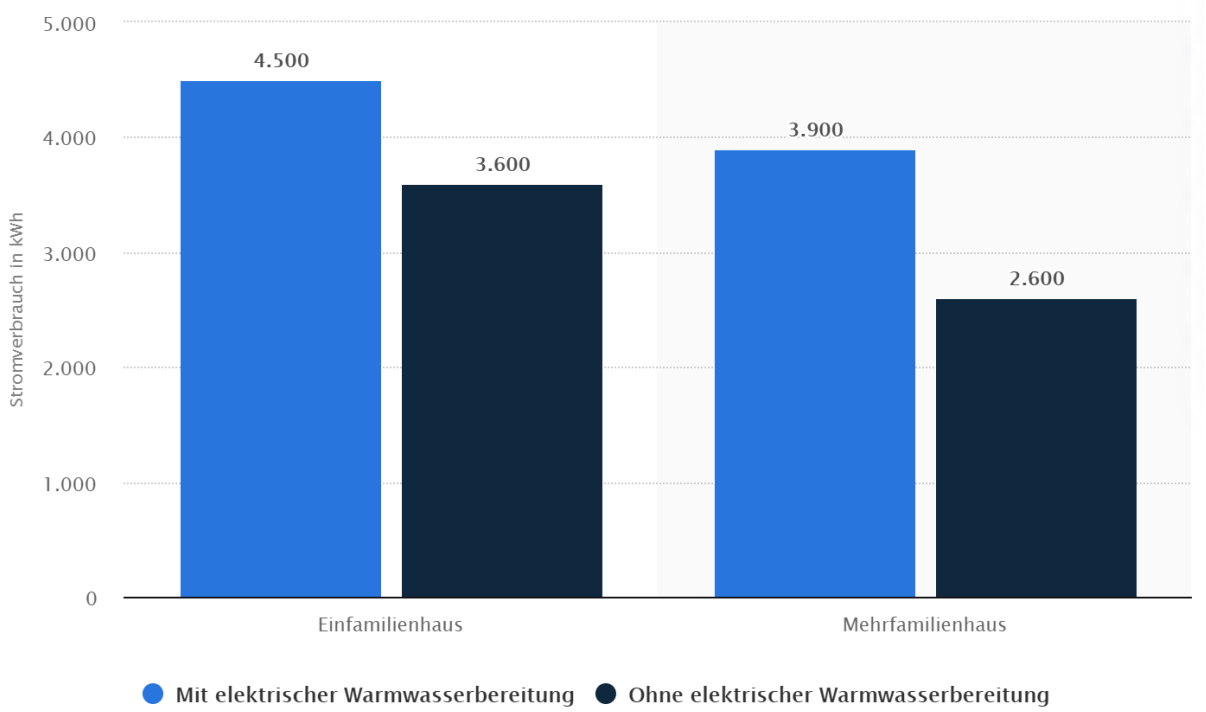


**Abbildung 4.10 Jährlicher Stromverbrauch eines 3-Personen-Haushalts in Deutschland nach Gebäudetyp im Jahr 2019 [17]**

Aus Abbildung 4.10 wird nun ein einziger Durchschnittswert berechnet:

$$\frac{(4500\ +\ 3600\ +\ 3900\ +\ 2600)\ kWh}{4} = 3650\ kWh = 3650000\ Wh$$

Das Verhältnis zwischen den vorhandenen Werten ergibt sich wie folgt:

$$\frac{erzeugte\ Energie}{Anzahl\ Tage} = \frac{durchschnittlicher\ Stromverbrauch}{Jahr}$$

$$\frac{sum(\text{"P\_ERZ"})/4}{Anzahl\ Tage} = \frac{3650000\ Wh}{365\ Tage}$$

Die gesamte erzeugte Energie muss daher mit dem Faktor $\frac{365}{3650000\ Wh}$ skaliert werden, um die Anzahl an Tagen zu erhalten. Daraus ergibt sich folgende Influx Query:

```
SELECT sum("P_ERZ")/4/3650000 * 365
FROM "year"."Messwerte_15"
WHERE $timeFilter
```

Damit sind alle Panels für das Dashboard 2 erstellt. Es können allerdings nur Werte angezeigt werden, die innerhalb der verwendeten Retention Policy *year* liegen, also nur ein Jahr. Die Energiedaten des heutigen Tages werden von Grafana Dashboard 1 visualisiert, das aber vollkommen analog zu Dashboard 2 aufgebaut ist. Es verwendet allerdings das Measurement *Messwerte* und die Retention Policy *week*, um minutengenaue Werte anzuzeigen, die öfter aktualisiert werden. Damit lassen sich wiederum nur Werte anzeigen, die maximal eine Woche zurückliegen. Für noch ältere Werte müsste in einem separaten Grafana Dashboard das Measurement *Messwerte_d* mit der Retention Policy *forever* verwendet werden. Eine passende Implementierung ist zu diesem Zeitpunkt aber noch nicht sinnvoll.

# 5 Zusammenfassung und Ausblick

Das im Rahmen dieser Bachelorarbeit entwickelte Monitoring System kann als erster großer Schritt eines vollumfänglichen Visualisierungskonzeptes für die Montessori Schule Niederseeon oder auch andere Schulen mit ähnlichen Zielen gesehen werden. Es enthält die elementaren Grundbausteine der Datenverarbeitung und -speicherung und bietet einige manuelle Eingriffsmöglichkeiten für den Benutzer. Vergleicht man das Endergebnis mit den in Abschnitt 2.2 gestellten Anforderungen, so stellt man Folgendes fest:

Für den Administrator, der die Programme in gewissem Maß steuern kann, sind alle Anforderungen erfüllt. Ein Problem vor Ort könnte aber womöglich darin bestehen, an welchem Ort der Raspberry Pi genau montiert wird. Zunächst ist die Montage auf der Rückseite eines Monitors in der Schulaula vorgesehen und nicht etwa im Sekretariat. Das macht eine direkte Bedienung des Raspberry Pi schwer, da sich Maus und Tastatur nur bedingt oder provisorisch anschließen lassen. Sollte sich ein Programm auf dem Raspberry Pi also z.B. aufhängen, so ist man auf die Möglichkeiten einer Remoteverbindung angewiesen.

Auch für die Schüler, die durch die Visualisierung sensibilisiert werden sollen und im besten Fall auch etwas lernen können, sind die fast alle Anforderungen erfüllt worden. Die Dashboards sind minimalistisch gestaltet und es kommen abgesehen von der Außentemperatur lediglich die physikalischen Größen von Leistung und Energie vor. Womöglich sollte der Zusammenhang dieser beiden Größen vor allem für die unteren Jahrgangsstufen aber genauer erklärt werden. Die Balkendiagramme für die erzeugte und verbrauchte Energie sowie die Einsparung in Prozent sind klar und unmissverständlich, auch wenn die Einheit einer Kilowattstunde für manche jüngeren Schüler womöglich noch kein Begriff ist. Allerdings sind greifbare Vergleiche oder eine wirklich kinderfreundliche Darstellung nur bedingt vorhanden. Der Grund dafür ist, dass Grafana zwar viele Möglichkeiten zur Visualisierung bietet aber alle sich auf einem wissenschaftlichen Niveau befinden. Als Beispiel: Die Energieeinsparung in Prozent war anfangs als eine Art Batterie-Anzeige gedacht, wobei sich die Umsetzung in Grafana nur mit einem Balkendiagramm und farblich unterschiedlichen Grenzwerten realisieren ließ. Auch die verbrauchte und erzeugte Energie könnte gut in einem waagrechten Balken als eine Art Tauziehen dargestellt werden, das für die Schüler eher nach einem interessanten Wettkampf zwischen Verbrauch und Erzeugung aussehen würde. Stattdessen mussten zwei einzelne Balken verwendet werden, die sich für eine gemeinsame Skalierung auch in einem gemeinsamen Panel befinden und deswegen leider nur mit den gleichen Farben angezeigt werden können.

Die Anforderungen an das System wurden in den Programmen gut und vollumfänglich umgesetzt. Mit Hilfe von Cronjobs gelingt auch die zeitgesteuerte Automatisierung zuverlässig. Der in Abschnitt 2.2.2 angesprochene Ruhemodus wurde allerdings nur als ein Ausschalten des Bildschirmes und der beiden Hauptprogramme auf dem Raspberry Pi realisiert. Dieser bleibt also auch nachts eingeschaltet, was aber aufgrund seines sehr geringen Stromverbrauchs kein großes Problem darstellen sollte. Wenn die Schule am Wochenende nicht besucht ist bzw. das Monitoring zu dieser Zeit nicht gewünscht ist, sollten die Cronjobs so angepasst werden, dass auch an Wochenenden diese Abschaltung erfolgt.

Das entwickelte Monitoring System ist an sich also voll funktionsfähig, kann aber in Zukunft noch sinnvoll ausgebaut und erweitert werden. Folgende Möglichkeiten bieten sich dafür noch an:

- Einbinden von Wärmedaten
  Sobald alle nötigen Sensoren bzw. Zähler in der Schule angebracht worden sind und auch in das OpenMUC Framework des Notebooks vor Ort eingelesen werden, lassen sich ebenso die Wärmedaten der Schule visualisieren. Hierbei wird sich erst recht der Nutzen des BHKWs zeigen und es kann wirklich der gesamte Energiehaushalt der Schule analysiert werden.
- Eingriffsmöglichkeiten für die Schüler
  Im Moment können die Energiedaten nur abgelesen und betrachtet werden. Es würde bestimmt noch mehr Interesse wecken, wenn die Schüler mit einigen Tasten am Raspberry Pi zwischen Dashboards oder einzelnen Graphen hin- und herschalten könnten. Außerdem könnten die Energiedaten in Lerneinheiten verwendet werden und als Beispiel für manche Aufgaben oder Erklärungen dienen. Einige Computer mit einer Remote Verbindung zu dem Raspberry Pi zu verknüpfen, könnte auch eine Lösung sein, um die Schüler selbst die Daten erforschen zu lassen. Dazu würde sich die Einrichtung von Gast Accounts in Grafana empfehlen.
- Ausbauen und Verbessern der GUI
  Die graphische Benutzeroberfläche ist noch eher funktional und simpel gestaltet. Sie könnte an einigen Stellen noch ausgebessert werden, z.B. ein rot umrandeter Stopp-Button, falls die Rotation im Browser gerade läuft. Generell kann hier noch mehr Rückmeldung an den Anwender implementiert werden, wenn das Einlesen einer .dat-Datei oder ein Backup erfolgreich war. Fehler könnten dann ebenfalls in einem extra Fenster angezeigt werden.
- Einbinden der Grafana Applikation in die Homepage der Montessori Schule
  Auf diese Wiese könnten nicht nur die Schüler und Lehrbeauftragten die Messdaten sehen, sondern auch andere Interessierte, wie z.B. Eltern, Freunde und Verwandte. Es würde den bewussten Umgang der Schule mit Energie und Ressourcen zum Ausdruck bringen und könnte so auch mehr Interesse im Umkreis wecken.
- Aufstellen einer Bilanz für die Energiekosten in Euro
  Ein Aspekt, der bislang in diesem Monitoring System noch außen vorgelassen wurde, ist die Einsparung an monetären Kosten, die mit der Verwendung der regenerativen Energiequellen einhergeht. Geld ist für sowohl für Erwachsene als auch für Schüler etwas greifbares und kann eine ganz neue Perspektive auf das Thema der regenerativen Energieerzeugung kreieren.

# A1 Literaturverzeichnis

[1] „Bundesstelle für Energieeffizienz," 2019. [Online]. Available: https://www.bfee-online.de/BfEE/DE/Effizienzpolitik/EnergieeffizienzGebaeude/energieeffizienzgebaeude_node.html. [Zugriff am 18 Oktober 2020].

[2] „World Health Organization - Regional Office for Europe," 16-18 Oktober 2013. [Online]. Available: https://www.euro.who.int/en/health-topics/environment-and-health/air-quality/publications/2014/combined-or-multiple-exposure-to-health-stressors-in-indoor-built-environments. [Zugriff am 18 Oktober 2020].

[3] „Energieeffizienz in Zahlen - Entwicklungen und Trends in Deutschland," Bundesministerium für Wirtschaft und Energie (BMWi), Berlin, 2019.

[4] R. Dr. Paschotta, „RP-Energie-Lexikon," 14 März 2020. [Online]. Available: https://www.energie-lexikon.info/primaerenergie.html. [Zugriff am 18 Oktober 2020].

[5] „Besser Bilden," [Online]. Available: https://www.besser-bilden.de/privatschulen-internate-suche/deutschland/bayern/moosach/montessori-schule-niederseeon/. [Zugriff am 29 10 2020].

[6] „Montessori Pädagogik," [Online]. Available: http://www.montessori.de/montpaed.php. [Zugriff am 29 10 2020].

[7] „Grundschule Prüfening - Energiekonzept," [Online]. Available: https://gsp.schulen2.regensburg.de/energiekonzept/. [Zugriff am 10 Januar 2021].

[8] „Landkreis Ludwigslust-Parchim - Energiemonitoring," [Online]. Available: https://www.kreis-lup.de/buergerservice-verwaltung/kreisverwaltung/verwaltung-mit-zukunft/energiemonitoring/. [Zugriff am 10 Januar 2021].

[9] „Gymnasium Rahden - Energiemonitoring (seit 2013)," 2015. [Online]. Available: https://www.gymnasium-rahden.de/schwerpunkte/umweltbildung/energiemonitoring.html. [Zugriff am 10 Januar 2021].

[10] „dev connected," 2019. [Online]. Available: https://devconnected.com/the-definitive-guide-to-influxdb-in-2019/. [Zugriff am 14 Januar 2021].

[11] „Raspberry Pi," [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/. [Zugriff am 16 Januar 2021].

[12] „PC Welt," 13 März 2019. [Online]. Available: https://www.pcwelt.de/ratgeber/Die_besten_Ein-Platinen-PCs_im_Vergleich-Kleine_Helfer-8998742.html. [Zugriff am 16 Januar 2021].

[13] „CHIP,“ 03 Juli 2019. [Online]. Available: https://www.chip.de/artikel/Raspberry-Pi-4-im-Test-Das-kann-der-Rechenzwerg_170606939.html. [Zugriff am 16 Januar 2021].

[14] M. Rowan, „Martin Rowan Pi Cases,“ 4 September 2019. [Online]. Available: https://www.martinrowan.co.uk/2019/09/raspberry-pi-4-cases-temperature-and-cpu-throttling-under-load/. [Zugriff am 16 Januar 2021].

[15] M. Rowan, „Martin Rowan Argon One Case,“ 03 Dezember 2019. [Online]. Available: https://www.martinrowan.co.uk/2019/12/argon-one-pi-4/. [Zugriff am 16 Januar 2021].

[16] OpenMUC, „OpenMUC - About Us,“ Fraunhofer Institut, 2021. [Online]. Available: https://www.openmuc.org/about-us/. [Zugriff am 12 Februar 2021].

[17] „Statista,“ 2021. [Online]. Available: https://de.statista.com/statistik/daten/studie/558280/umfrage/stromverbrauch-einen-3-personen-haushalts-in-deutschland/. [Zugriff am 24 Februar 2021].

# A2 Abbildungsverzeichnis

# A3 Ausschnitts-Verzeichnis

# A4 Messgrößen Übersicht

Anmerkung zu BHKW: Der Anschluss ist 3-Phasig, die Messungen erfolgen aber 1-phasig, da es sich um einen symmetrischen Erzeuger handelt und somit alle 3 Phasen den gleichen Strom bzw. die gleiche Spannung und Leistung führen.

#1 YYYYMMDD: Datum
#2 hhmmss: Uhrzeit
#3 unixtimestamp: Zeit seit 1970 in Sekunden
#4 U1_L1: Spannung am Netzanschlusspunkt, Phase 1, in Volt
#5 U1_L2: Spannung am Netzanschlusspunkt, Phase 2, in Volt
#6 U1_L3: Spannung am Netzanschlusspunkt, Phase 3, in Volt
#7 U1_L4: Spannung am BHKW-Anschlusspunkt, Phase 1 (entspricht U1_L1), in Volt
#8 I1_L1: Strom am Netzanschlusspunkt, Phase 1, in Ampere
#9 I1_L2: Strom am Netzanschlusspunkt, Phase 2, in Ampere
#10 I1_L3: Strom am Netzanschlusspunkt, Phase 3, in Ampere
#11 I1_L4: Strom am BHKW-Anschlusspunkt, Phase 1, in Ampere
#12 P1_L1: Wirkleistung am Netzanschlusspunkt, Phase 1, in Kilowatt
#13 P1_L2: Wirkleistung am Netzanschlusspunkt, Phase 2, in Kilowatt
#14 P1_L3: Wirkleistung am Netzanschlusspunkt, Phase 3, in Kilowatt
#15 P1_L4: Wirkleistung am BHKW-Anschlusspunkt, Phase 1, in Kilowatt
#16 Q1_L1: Blindleistung am Netzanschlusspunkt, Phase 1, in Kilo Volt-Ampere-Reaktiv
#17 Q1_L2: Blindleistung am Netzanschlusspunkt, Phase 2, in Kilo Volt-Ampere-Reaktiv
#18 Q1_L3: Blindleistung am Netzanschlusspunkt, Phase 3, in Kilo Volt-Ampere-Reaktiv
#19 Q1_L4: Blindleistung am BHKW-Anschlusspunkt, Phase 1, in Kilo Volt-Ampere-Reaktiv
#20 1frequency: Netzfrequenz, in Hertz
#21 PAC_I1: Wirkleistung (Wechselstrom) der Photovoltaikanlage 1, in Watt
#22 PDC_I1A: Wirkleistung (Gleichstrom) Kanal 1/2 der Photovoltaikanlage 1, in Watt
#23 PDC_I1B: Wirkleistung (Gleichstrom) Kanal 2/2 der Photovoltaikanlage 1, in Watt
#24 UDC_I1A: Gleichspannung Kanal 1/2 der Photovoltaikanlage 1, in Volt
#25 UDC_I1B: Gleichspannung Kanal 2/2 der Photovoltaikanlage 1, in Volt
#26 PAC_I2: Wirkleistung (Wechselstrom) der Photovoltaikanlage 2, in Watt
#27 PDC_I2A: Wirkleistung (Gleichstrom) Kanal 1/2 der Photovoltaikanlage 2, in Watt
#28 PDC_I2B: Wirkleistung (Gleichstrom) Kanal 2/2 der Photovoltaikanlage 2, in Watt
#29 UDC_I2A: Gleichspannung Kanal 1/2 der Photovoltaikanlage 2, in Volt
#30 UDC_I2B: Gleichspannung Kanal 2/2 der Photovoltaikanlage 2, in Volt
#31 T_AMB: Umgebungstemperatur, Wertebereich von 0 bis $2^{16}$ entspricht -40°C bis 105°C
#32 T_IBC: Zelltemperatur Silizium, Wertebereich von 0 bis $2^{16}$ entspricht -40°C bis 160°C
#33 T_AV: Zelltemperatur Dünnschicht, Wertebereich von 0 bis $2^{16}$ entspricht -40°C bis 160°C
#34 IRR: Einstrahlung, Wertebereich von $2^{15}$ bis $2^{16}$ entspricht 0 bis 1300 Watt pro Quadratmeter

# A5 USB-Stick