Staatliche Technikerschule Berlin

Überwachungssystem für autarke Anlagen

Projektarbeit von:

Steinmann, Ole
TEA16192
Wächter, Matthias
TEE16626

betreut durch:

Dipl.-Ing. Kurt Jankowski-Tepe

Fachrichtung : Elektrotechnik
Schwerpunkt : Energietechnik

Berlin, den 28.05.18

Kurzfassung

Gegenstand der hier vorgestellten Arbeit ist ein Dokument für eine Abschlussarbeit für den Staatlich geprüften Techniker im Bereich Elektrotechnik.
In dieser Dokumentation ist beschrieben, wie ein Überwachungssystem von Grund auf durch einen Mikrocontroller realisiert wird.
Die Bedienung und Ausgabe finden an einem direkt angeschlossenen Touch Panel statt.
Dem Bediener werden die wichtigsten Informationen wie Energieerzeugung, dem Verbrauch und die Wetterdaten angezeigt. Zudem wird eine Prognose der nächsten zwei Tage gestellt, um möglichst effizient mit der vorhandenen Energie auszukommen.

Die Energieerzeugung kann auf verschieden Wegen stattfinden. Wir haben uns für den Weg der Erzeugung mittels Solarpanel entschieden. Zu Vorführungszwecken wird ein geschlossenes System vorgeführt.
Zum Speichern der Energie wird unsererseits eine Batterie mit 12V / 7Ah verwendet.

Der Akku ist der zentrale Knotenpunkt der gesamten Energieversorgung. Die Spannungsversorgung der zentralen Steuerung wird ebenfalls über diesen Akku versorgt. Strom- und Spannungswerte von der Energieerzeugung und dem Verbrauch werden im programmierten Mikrocontroller ausgewertet und analysiert. Um ein zusammenbrechen der Versorgung zu verhindern, wird der Benutzer rechtzeitig gewarnt. Es werden Warnungen ausgegeben, um möglichst lange mit der Akkukapazität auszukommen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Vorwort

Nachdem wir uns für ein Projekt geeinigt haben, mussten wir schnell feststellen, dass das Projekt sehr Umfangreich werden kann. Daher mussten wir uns besonders bei dem Thema der Prognose für den Verbrauch auf eine einfache Variante beschränken. Es werden nur der aktuelle Verbrauch und die Erzeugung ermittelt. Diese Werte werden verarbeitet und der Batterie gegenübergestellt. Auch der Einfluss des Wetters wird mit in diese Prognose einfließen. Denn für den Benutzer ist eine Vorhersage der Erzeugung vom Vorteil. Jedoch kann man unter der Wettervorhersage sehen, ob Sonne bzw. Wind vorhanden sein wird. Dadurch kann festgestellt werden, ob es bsp. Bewölkt, leicht bedeckt oder sonnig ist.

Im Laufe des Projektes wurden auch mehrere Umbauarbeiten an einem Gehäuse für das Display unternommen. Der Anbieter, oder auch sonst keine weiteren im Internet bieten für unser Model etwas an. Da wir die kosten jedoch im Überblick halten mussten, entschieden wir uns für eine kostengünstigere Variante des Touchpanels. Leider mussten wir feststellen wie ungenau dieses arbeitet. Aus diesem Grund wurden die Icons groß, damit es leicht bedienbar wird. Jedoch ist im Laufe der Zeit die Funktion des „Touch“ kaputt gegangen und die Zeit für ein Austausch ist nicht da. Aus diesem Grund wird die Präsentation mit einer Maus stattfinden.

Das Projekt ist für uns ausbaufähig und im privaten Bereich vom Nutzen.

1 Überblick

Ein Überwachungssystem soll geschaffen werden. Dieses findet beispielsweise in einem Campingwagen oder einer Ferienhütte, ohne Anschluss an das öffentliche Netz statt.

Für die Realisierung haben wir uns für einen aktuellen Raspberry Pi 3 entschieden. Da dieser über genügend Leistung und ein integriertes Wlan Modul verfügt. Dieses wird für die Aktualisierung der Wetterdaten über einen Hotspot eines Mobiltelefons benötigt. Die visuelle Darstellung und die Bedienung des Panels erfolgt über ein 7 Zoll Touchscreen.

Für die Programmierung haben wir uns für Linux entschieden, da der Raspberry unter diesem Betriebssystem läuft.
Als Editor für die Programmierung verwenden wir „Sublime Text“, aufgrund sehr einfacher Strukturen und Übersichtlichkeit.
Bei der Programmiersprache haben wir uns für Python entschieden. Da sich viele Programme und einzelne Funktionen wesentlich schneller programmieren lassen als beispielsweise in C(++). Zudem ist diese eine Plattformunabhängige Sprache und lässt sich auf verschiedenen Betriebssystemen Anwenden.
Python bringt mit „QT Designer“ ein mächtiges und leicht zu bedienendes Werkzeug zur Programmierung grafischer Benutzeroberflächen mit, die ebenfalls Plattformunabhängig sind.
Es lassen sich Aufgrund der großen Internetgemeinschaft viele Informationen oder Anstöße für die Programmierung im Internet und Büchern finden.
Hinzu kommt das Dokumentationsprogramm Doxygen. *Dies* ist ein Open-Source-Dokumentationswerkzeug, um innerhalb von Quelltexten zu dokumentieren.
Zur Kontrolle wird „PyLint“ verwendet. Dieses Programm überprüft den Python-Code auf mögliche Fehler und zeigt diese an.

Die einzelne Software muss unter Linux jedoch noch zusätzlich installiert werden.

Sämtliche Bauteile stehen zur freien Verfügung. Lediglich die Software und die Bedieneroberfläche wurden eigenständig erstellt.

Folgende Komponenten werden für das Projekt benötigt:

Hardware

Raspberry Pi 3
SD- Speicherkarte
7 Zoll Touchscreen
Batterie 12V/6Ah
Gehäuse
Relais
Stepdown 5V→12V
Strommesssensoren
Widerstände 22KΩ und 4,7KΩ
Platine
Sicherungen 500mA,3A und 1,25A (Schmelzsicherung-Flink)
Laderegler

Software

Linux
Sublime Text
QT Designer
Doxygen
PyLint
Visio (Office)
Word

Die Zusammenführung der Hardware Komponenten findet in einer Kunstoffbox statt. Diese ist ausschließlich für den Prototypen vorgesehen und entspricht nicht den gesetzlichen Vorschriften.

2 Ziel

Ziel dieses Projekts ist es, ein Überwachungssystem zu schaffen, welches vielseitig verwendbar ist. Die Benutzeroberfläche soll dabei einfach und klar strukturiert sein. Der Anwender soll dabei nichts von dem im Hintergrund laufenden Prozessen mitbekommen.

Das System soll auf einem Blick die Verbrauchs-, Erzeugungs- und die Wetterdaten anzeigen. Zusätzlich wird das System bei einem kritischen Zustand der Batterie eine Warnmeldung ausgeben und das System herunter fahren.
Durch Auswertung der Wetterdaten und einer groben Prognose wird eine mögliche Energieerzeugung berechnet.
Über eine ermittelte Entlade kurve der Batterie und dem aktuellen Verbrauch ist es möglich, eine Auskunft über die Restlaufzeit des Systems zu geben.

3 Komponenten

3.1 Raspberry Pi 3

Der Raspberry Pi 3 ist ein Einplatinencomputer. Dieser verfügt über ausreichend Leistung und digitale Ein- und Ausgänge. Die benötigte Spannung beträgt 5 V/2 A. Die Versorgung des Gerätes findet über den Akku statt.
Zudem wird eine Sd-Karte benötigt um Linux zu installieren und die Software, um das Programm auszuführen.

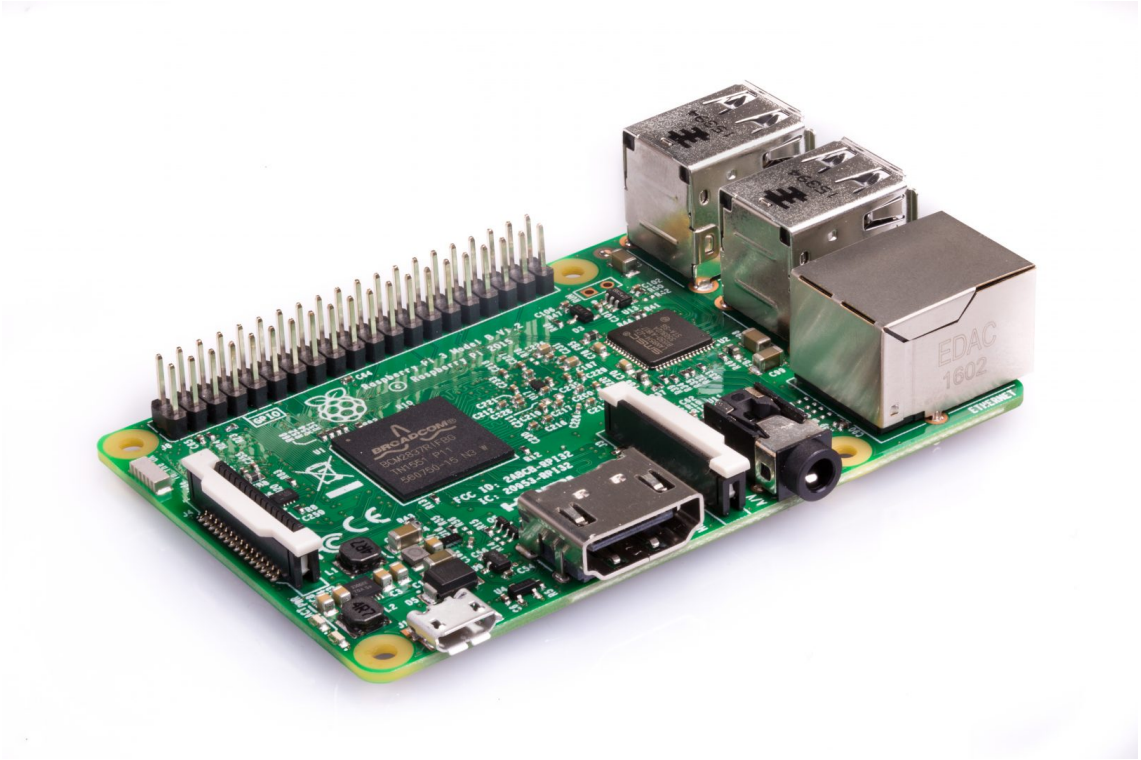


Abbildung 3-1: Der Mikrocontroller Raspberry Pi 3 (Quelle: www.raspberry.org)

3.2 7 Zoll Touchscreen

Für die Ausgabe und Bedienung der Software wird ein 7 Zoll Touchscreen verwendet. Dieser muss vorher mit entsprechender Software installiert und kalibriert werden. Das 7" (17,78 cm) Display-Set mit Touchscreen LS-7T verfügt über HDMI/DVI/VGA und CVBS Ausgänge. Die Auflösung beträgt 1024x(RGB)x600 Pixel. Die Betriebsspannung beträgt 9-18V und eine maximale Stromaufnahme von 460 mA.

Abbildung 3-2: Touchscreen 7 Zoll (Quelle: www.bastard-fdb.blogspot.de)

3.3 Batterie

Für das Projekt wird ein handelsüblicher Akku verwendet. Dieser verfügt über 12V/7Ah.

Abbildung 3-3: 12 V/7Ah Akku

3.4 Platine

Da das System eine Verbindung zwischen Erzeugung und Mikrocontroller erfordert, wird eine Platine benötigt. Diese Platine ist das Bindeglied zwischen den einzelnen Komponenten, Träger der Sicherungen, sowie sämtlichen Bauteilen.
Sie wurde eigens ausschließlich für dieses Projekt angefertigt und befindet sich im unteren Teil der Kunststoffbox.

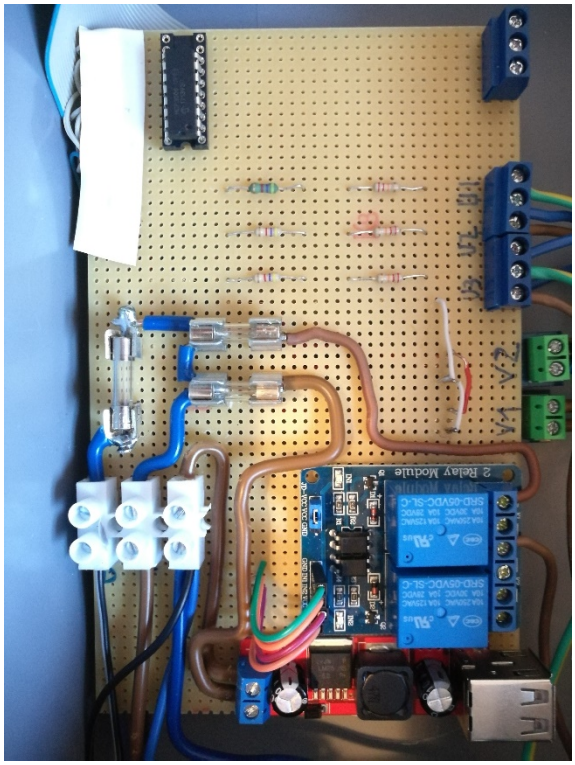


Abbildung 3-4 Platine

3.5 Hauptbild Erklärung

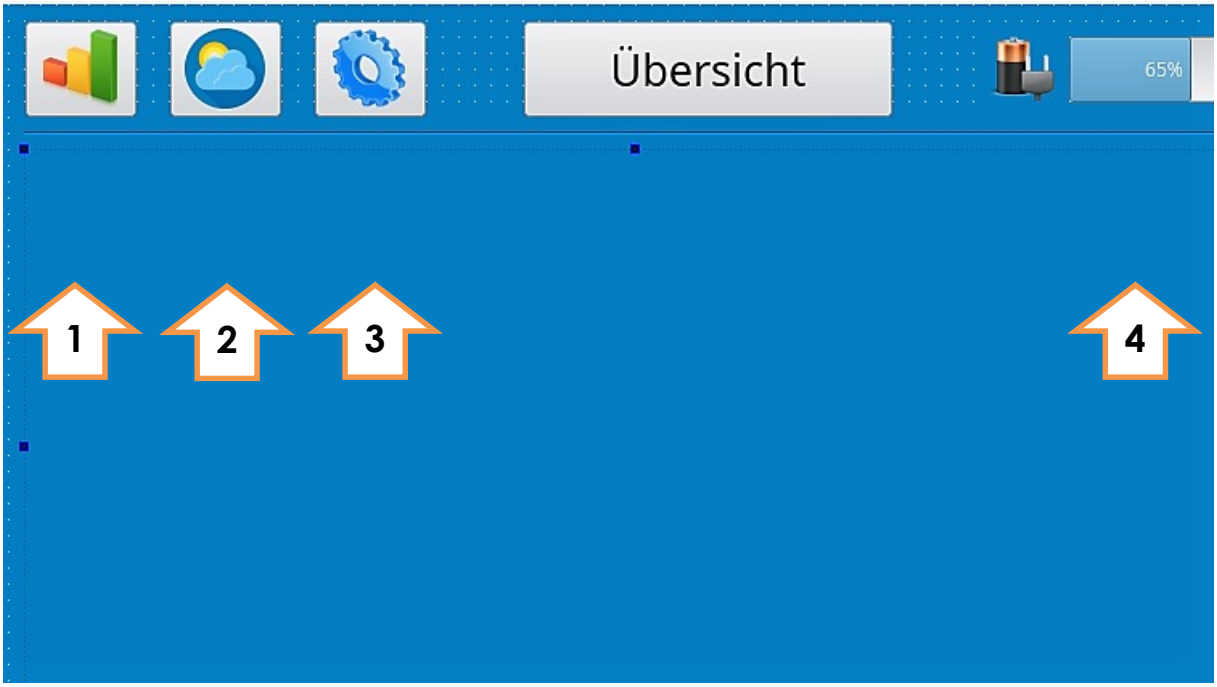


Abbildung 3-5: Darstellung Hauptbild

1. Diese Schaltfläche zeigt den Hauptbildschirm an. Dem Benutzer sollen auf einem Blick die aktuellen Verbrauchs-, Erzeugungs- und Prognosedaten gezeigt werden. Die Verbrauchs- und Erzeugungswerte werden in Echtzeit angegeben. Bei den Wetterdaten findet eine Aktualisierung bei bestehender Internetverbindung alle 8 Stunden statt. Um eine möglichst genaue Prognose zu erstellen, ist der Datenabruf von äußerster Wichtigkeit.

2. Durch klicken auf das Wettersymbol werden die aktuellen Wetterdaten angezeigt. Zudem gibt es auch eine Vorhersage für die nächsten 2 Tage. Es werden die Temperaturen, Luftfeuchte, Windgeschwindigkeit, Regenwahrscheinlichkeit, Sonnenauf- und Sonnenuntergang sowie die Bewölkung als Icon.

3. Die Einstellungen enthalten nur wenige Option für den Nutzer. Hierbei kann der Benutzer des Systems die Daten des Solarpaneels in Watt eingeben. Zudem gibt es die Möglichkeit mit dem Display in ein Energiesparmodus zu gehen oder das System Herunter zu fahren.

4. Das Batteriesymbol zeigt den aktuellen Akkuzustand in Prozent an. Dieser Wert wird über eine Entladekurve ermittelt. Siehe auch unter Kapazitätsprognose.

3.6 Stromrelais

Übersicht:
Das IC von Allegro ermöglicht es mittels integriertem Hall Effect-Based Linear Stromsensor bis zu 20A zu messen und Analog - Linear auszugeben. Einfache Anbindung an Microcontroller Schaltungen oder DMM:
Ladegeräte;
Batteriestromüberwachung;
Zustandsanzeige von Geräten;
Überwachnung von elektrischen und elektronischen Geräten.

Technische Daten:
Chip ACS712-20 20A 5V
5V DC Betriebsspannung z.B. von der MCU
Betriebsspannung mit LED "Power Good Anzeige"
Messbereich: ±20A/DC
Analog Ausgang:100 mV/A
fließt kein Strom - Ausgangsspannung ist ~ VCC / 2
Maße PCB: L x B x H ca. 27,5 x 11,6 x 14 mm

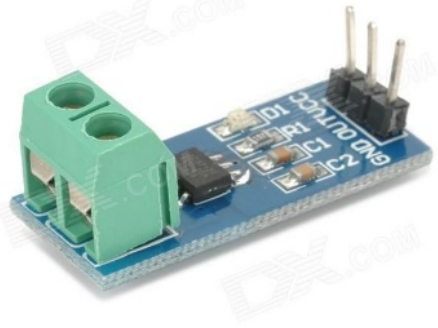


Abbildung 3-6Abbildung 3 1: Stromsenor ACS712-20 20A (Quelle: www.reichelt.de)

3.7 IC-MCP3008.I/P

Technische Daten

10BIT ADC,2.7V,8CH,SPI,16DIP
Auflösung, Bit: 10bit
Abtastrate: 200kSPS
Supply Voltage Type: Einfach
Versorgungsspannung, min.: 2.7V
Versorgungsspannung, max.: 5.5V
Stromversorgung: 425µA
Bauform: DIP
Anzahl der Pins: 16
Eingangskanaltyp: Pseudo-Differenz, unsymmetrisch
Dateninterface: Seriell, SPI
Analog-Versorgungsspannung: 2.7V bis 5.5V
Anschlussart: Durchsteckmontage
Betriebstemperatur: -40°C bis +85°C
Funktions-Nr.: 3008
Linearitätsfehler, ADC/DAC, +: 1GWB
Versorgungsspannung: 2.7V bis 5.5V
Zahl der Kanäle: 8
Zeit, Umwandlung: 10µs

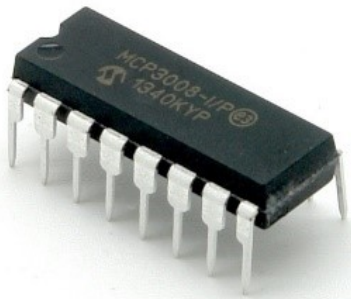


Abbildung 3-7 AD Wandler IC-MCP3008.I/P (Quelle: www.Ptricoma-netzwerk.de)

3.8 Energiesparmodus

Um eine Tiefenentladung zu verhindern, wird das Relais K2 bei 15% des Batteriestatus (zuständig für die Endverbraucher), bei einer Spannung von 11,995 Volt, abgeschalten. Der Verbraucher K1, welcher in unserem Fall das Licht schaltet, wird zusätzlich zum Raspberry bei 5% des Akkus deaktiviert.
Durch diese Abschaltung wird ein defekt der Batterie durch das Überwachungssystem verhindert.
Aufgrund von zeitlichem Mangel, wird das Display manuell durch einen Schalter deaktiviert.

Dem Bediener bleibt nur die Möglichkeit sein System zu kontrollieren. Waren beispielsweise der Verbraucher zu Groß und die erzeugte Energie im kleineren Verhältnis, so kann der Bediener sehen, dass das Gerät wieder geladen werden kann. Sollte jedoch der Fall eintreten, dass auch der Verbrauch des Überwachungssystem zu groß ist, so wird auch dieses abgeschalten um eine Tiefenentladung bzw. ein defekt der Batterie zu verhindern.

Oftmals will man auch Module mit höheren Spannung mit dem Raspberry Pi steuern. Für diesen Zweck können am Raspberry Pi Relais verwendet werden: Mittels eines Impulses wird der Relais-„Schalter" umgelegt. Da der Pi nur maximal 5V (die GPIOs sogar nur 3.3V) verträgt bleibt ohne Relais das Risiko vorhanden, den Pi durchbrennen zu lassen. Hat man jedoch zwei voneinander getrennte Stromkreise kann das nicht passieren.

Relais-Schalter

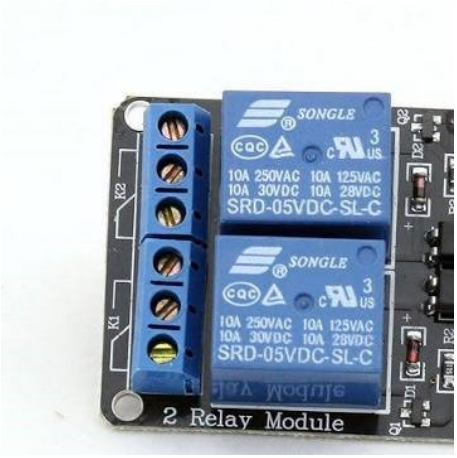


Abbildung 3-8GPIO-Pin (Quelle:www.encrypted-tbn2.gstatic.com)

4 Messungen

Das Überwachungssystem ist im gesamten Umfang nur Einsatzfähig, sobald Messwerte vorliegen.
Für dieses Projekt werden die Spannungswerte im Eingangs- und Ausgangsbereich, sowie am Akku benötigt. Der Strom wird nur im eingehenden und ausgehenden Bereich gemessen.

4.1 Strommessung

Von hoher Wichtigkeit ist, die Auswertung der ein- und ausgehenden Ströme bzw. Spannung. Um einen genauen Abgleich durchzuführen, werden die Werte der erzeugten und verbrauchten Energie ermittelt.
Dem Benutzer wird hierdurch eine Übersicht angezeigt, um sich daran orientieren zu können. Nur durch die exakten Angaben, kann das Überwachungssystem aufrechterhalten werden. Sollten dabei die Werte des Akkus auf 11,8 Volt sinken, so wird das System deaktiviert. Dabei wird eine Tiefenentladung des Akkus verhindert. Zudem werden die Verbraucher bei einem Restwert von 15% über das Relais K2 abgeschalten. Dem Benutzer stehen nun die Werte einer möglichen Erzeugung über dem Panel zur Verfügung.
Um die Erzeugte bzw. die verbrauchte Leistung zu ermitteln, werden 3 Spannungsmesser sowie 2 Stromsensoren benötigt.
Um auf dem Microcontroller die Messdaten auswerten zu können, wurden handelsübliche Stromsensoren vom Typ ACS 712 verwendet.

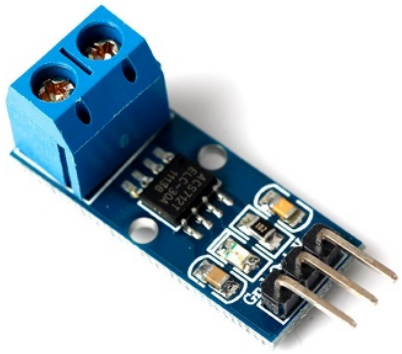


Abbildung 4-1Stromsenor ACS712-20 20A (Quelle: www.amazon.de)

Um die abgegebene Leistung bestimmen zu können wird an jedem der drei Energieerzeugern
und auf der Verbraucherseite jeweils ein Stromsensor integriert.
Um auf dem Microcontroller die Messdaten auswerten zu können, wurden handelsübliche Stromsensoren vom Typ ACS 712 verwendet.

Der Sensor kann einen Strom von bis zu 20A messen. Dadurch sind wir mit dem Projekt durch den ausreichenden Messbereich gut abgedeckt. Auch die Spannungsversorgung findet gleich wie beim Raspberry über den 5V Stepdown statt.
Die Sensoren sind direkt mit dem Mikrocontroller verbunden und können den Wert entsprechend anzeigen.
Mit Hilfe eines Hallsensors und der Versorgungsspannung liefert der Sensor einen Spannungswert. Jedoch zeigte sich ein Problem. Die Referenz zur schwankenden Versorgungsspannung verfälschen die Messergebnisse.
Um die Genauigkeit der Messung zu erhöhen wurde mit Hilfe eines Messversuches der möglichst genaue Nullpunkt ermittelt.
Die Messung wurde im Labor der STB durchgeführt. Mittels Generator und einem Vielfachmessgerät, sowie einem 50Ω Potentiometer wird der Strom auf 100mA eingestellt. Diese Schaltung wurde nun direkt mit dem Mikrocontroller verbunden und es wurden über eine Schleife 100 Messwerte erfasst. Der daraus entstandene Mittelwert wird später in der Berechnung subtrahiert. Diese Vorgehensweise bringt einen relativen guten Wert.

Als nächstes wird überprüft, ob bei einer Stromstärke von 0 A am Analogeingang eine Spannung von 2,5 V angezeigt wird. Mittels einer Rechnung wird diese Spannung ermittelt.

$\frac{Messwert}{1023} * 5V$ = 2,5016 V Offsetspannung → 2,5016 V – 2,5V = 0,016 V

Durch diese Formel wird die Offsetspannung des Sensors ermittelt. Um dem Nullpunkt möglichst nahe zu kommen, wird der Offset um – 0,016 V nach unten korrigiert.
Der nun ermittelte Spannungswert muss nun in einen entsprechenden Stromwert umgerechnet werden. Um nun die Schrittweite bestimmen zu können, muss ein vorher festgelegter Wert eingestellt werden.
Auf den Eingang des Sensors werden nun 1A gegeben. Der Mikrocontroller zeigt eine Spannung von 2,77 V an. Anhand des Wertes kann man einen Spannungswert pro Ampere von 0,064 V festlegen.
Als nächstes folgt die Skalierung. Beim Nullpunkt von 2,5 V ist es wichtig, dass ein Strom von 0 A angezeigt wird. Der Spannungswert wird also um -2,5 V nach unten korrigiert.

Der Code sieht wie folgt aus:

```
def ConvertAmpere(channel, places):
SUMamp = 0
SUMdata = 0
for i in range(0, 100):
data = ReadChannel(channel)
ampOffset = 0.016 #Offset abweichung
mVproAmp = 0.064  #Ermittelte Volt pro Ampere
ampere = (data * 3.3) / float(1023)+ampOffset
ampere = (ampere-1.65) / mVproAmp
SUMamp = SUMamp + ampere
SUMdata = SUMdata + data
ampere = SUMamp / 100
data = SUMdata / 100
ampere = round(ampere, places)
return (data, ampere)
```

4.2 Spannungsmessung

Neben dem Strom soll auch die Spannung mit dem Raspberry ausgelesen werden. Hierzu verwenden wir ein Potentiometer mit 10 KΩ.
Schließt man ihn zwischen der Masse und 3,3 V an, so erhält man zwischen Masse und dem Anschluss für den regelbaren Widerstand eine Spannung zwischen 0 V und 3,3 V, je nachdem wie weit das Potentiometer eingestellt ist. Mit Hilfe des A/D-Wandlers können nun Werte zwischen 0 und 1023 erfasst werden.
Diese müssen jedoch in einen realen Spannungswert umgewandelt werden.
Neben dem Mikrocontroller nutzen wir die einfach Methode des Spannungsteilers und einem IC1713V3U

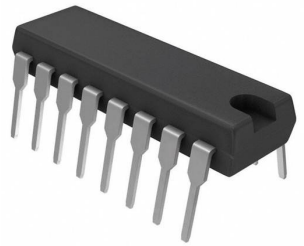


Abbildung 4-2ic 1713V3U (Quelle: www.conrad.de)

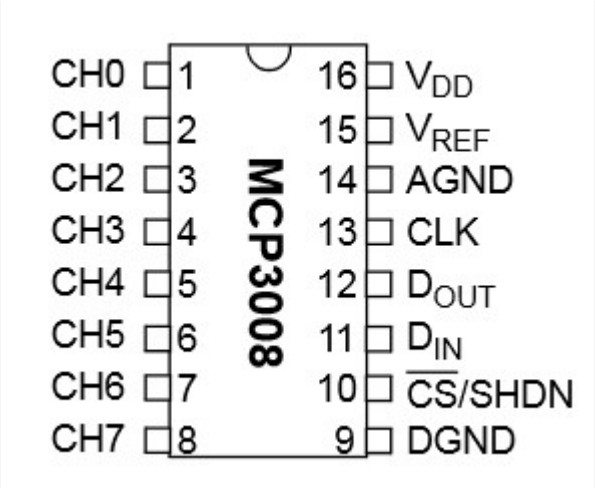
Bei der Spannungsmessung im Labor, konnten wir ein Offset von 0,028V feststellen und somit in den Code einbetten. Dadurch konnten wir ein genaues Ergebnis erzielen, welches für den Batteriestatus von Wichtigkeit ist.

Der Code sieht wie folgt aus:

```
def ConvertVolt(channel, places):
SUMvolt = 0
SUMdata = 0
ampOffset = 0.028
 for i in range(0, 100):
data = ReadChannel(channel)
volt = (data * 3.3) / float(1023)+ampOffset
volt = (volt * (R1+R2)) / R2
SUMvolt = SUMvolt + volt
SUMdata = SUMdata + data
volt = SUMvolt / 100
data = SUMdata / 100
volt = round(volt, places)
return (data, volt)
```

4.2.1 Verbindung zwischen Raspberry Pi und MCP3008

Die Verbindung des Raspberry Pi und dem des IC´s wird nachfolgend erklärt.



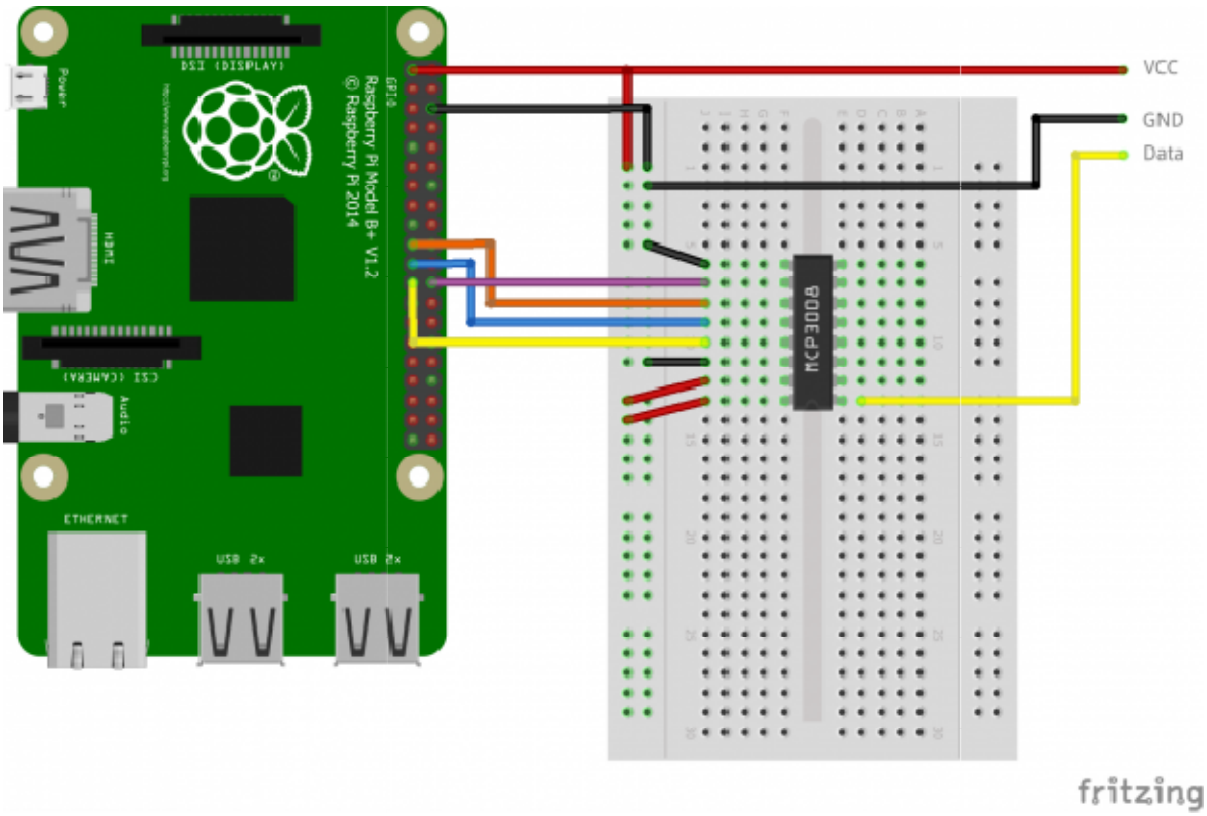
Auf der linken Seite des MCP3008 befinden sich die 8 analog auslesbaren Kanälen.

Die einfachste Methode einen Analog-Digital Konverter anzusprechen ist den SPI Bus zu verwenden. Hierbei können alle 8 anliegenden Signale mit einer Abfrage ausgelesen werden und umgewandelt werden können.
Wie im Datenblatt zu sehen ist, verträgt der ADC eine Eingangsspannung zwischen 2.7V und 5V. Da die GPIOs mit 3.3V arbeiten und bei höherer Spannung kaputt gehen könnten, sollte der MCP3008 auch nur mit 3.3V betrieben werden. Solltest du ein analoges Modul verwenden, welches eine höhere Eingangsspannung als 3.3V hat (und diese auch weitergeben kann) musst du unbedingt sichergehen, dass diese den ADC nicht erreichen. Dafür können z.B. Vorwiderstände verwendet werden und dies ggf. im weiteren Verlauf (Berechnungen etc.) beachtet werden.
Die Verbindung zum Raspberry Pi ist recht einfach, da nur jene GPIOs verwendet werden, welche auch für den SPI Bus sind:

RaspberryPi	MCP3008
Pin 1 (3.3V)	Pin 16 (VDD)
Pin 1 (3.3V)	Pin 15 (VREF)
Pin 6 (GND)	Pin 14 (AGND)
Pin 23 (SCLK)	Pin 13 (CLK)
Pin 21 (MISO)	Pin 12 (DOUT)
Pin 19 (MOSI)	Pin 11 (DIN)
Pin 24 (CE0)	Pin 10 (CS/SHDN)
Pin 6 (GND)	Pin 9 (DGND)

Tabelle 5.2.1: Pin Belegung Rapberry zu MCP3008

Schematisch sieht die Verbindung dann wie folgt aus, wobei ich explizit keinen Sensor auf die rechte Seite gestellt habe, da alle digitalen Signale so ausgelesen werden können:



Quelle:
https://tutorials-raspberrypi.de/raspberry-pi-mcp3008-analoge-signale-auslesen/

5 Übersichten

Das folgende Bild soll eine grobe Übersicht der Bauteile und dazwischenliegenden Strom- und Spannungsmessungen anzeigen. Es wird davon ausgegangen, dass das vollständige System miteinander verbunden ist.

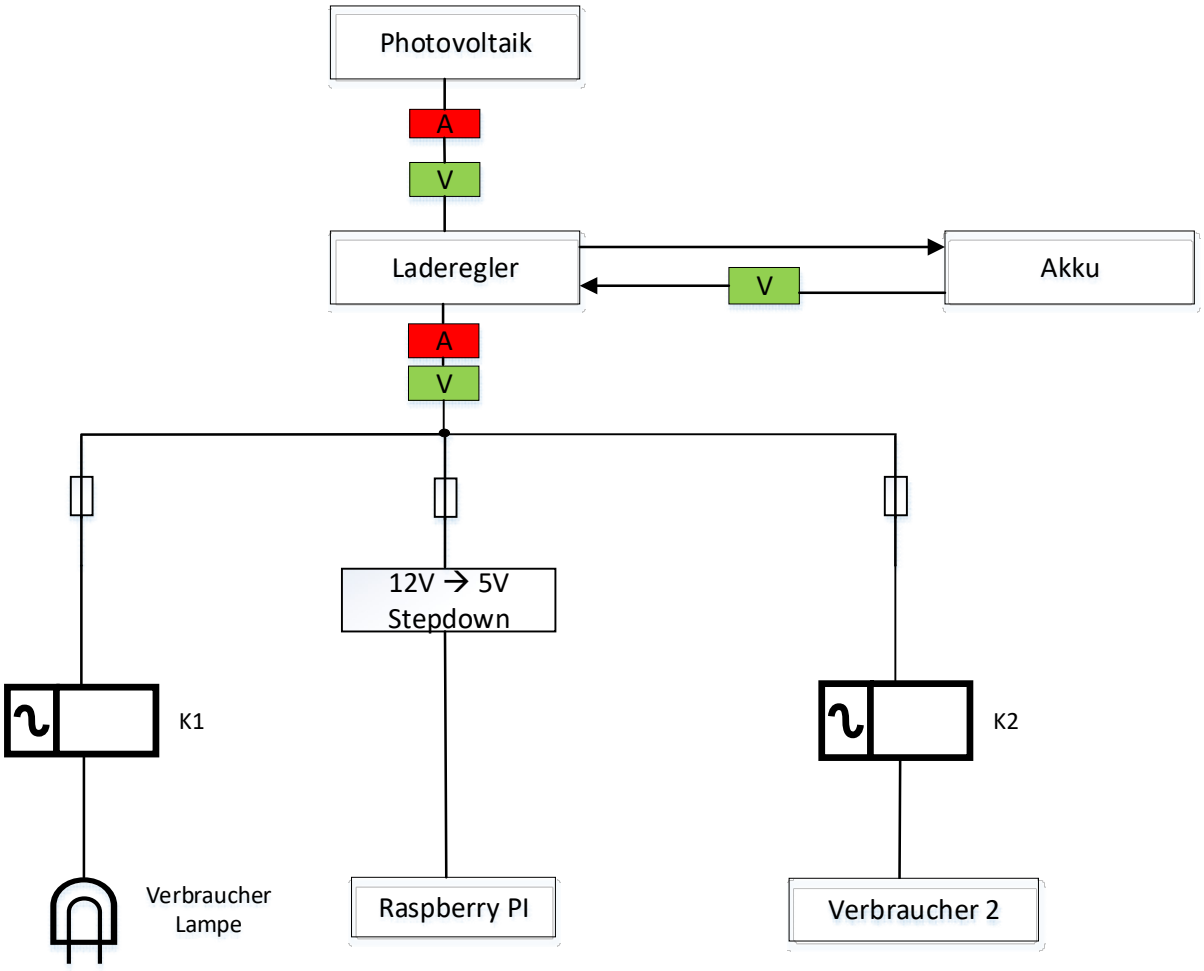


Abbildung 5-1 Übersichtsplan

5.1.1 Funktionsweise der Hardware

I

Für die Energieerzeugung dient eine Photovoltaikanlage, die über einen Laderegler zugeschalten werden kann. Der Laderegler hat die Aufgabe, die Batteriespannung zu überwachen und bei Bedarf zu laden.
Um die eingehende Spannung bzw. den Strom zu messen, werden Messgeräte vor und hinter den Laderegler geschalten. Hierdurch wird ein Vergleich erzeugt, mit dem es möglich den Verbrauch zu ermitteln. Zudem wird direkt hinter dem Akku eine Spannungsmessung durchgeführt, mit der der Batteriestatus auf dem Display angezeigt werden kann.
Die jeweiligen Komponenten sind mit einzelnen Schmelzsicherungen abgesichert.

- Display mit 500mA Schmelzsicherung Flink
- Raspberry mit 3 A Schmelzsicherung Flink
- Verbraucher mit 1,25 A Schmelzsicherung Flink

Da der Raspberry PI nur mit 5 Volt betrieben werden kann, war die installation eines 12V zu 5V Stepdown nötig. Dieser brachte jedoch den Vorteil, dass das Gerät direkt mit einem USB-Ladekabel angeschlossen werden kann.
Das Display wird mit einem Relais (K1) angesteuert. Das ermöglicht die Abschaltung über das System. Hierbei kann es auch unter dem Punkt der Einstellungen direkt in den Energiespar-Modus geschickt werden. Das Display verbraucht nun keine weitere Energie mehr.
Für die Simulation der Verbraucher wird eine LED angesteuert. Diese wird durch das weitere Relais K2 angesteuert und dient nur zur Präsentation.

5.2 Bildübersichten

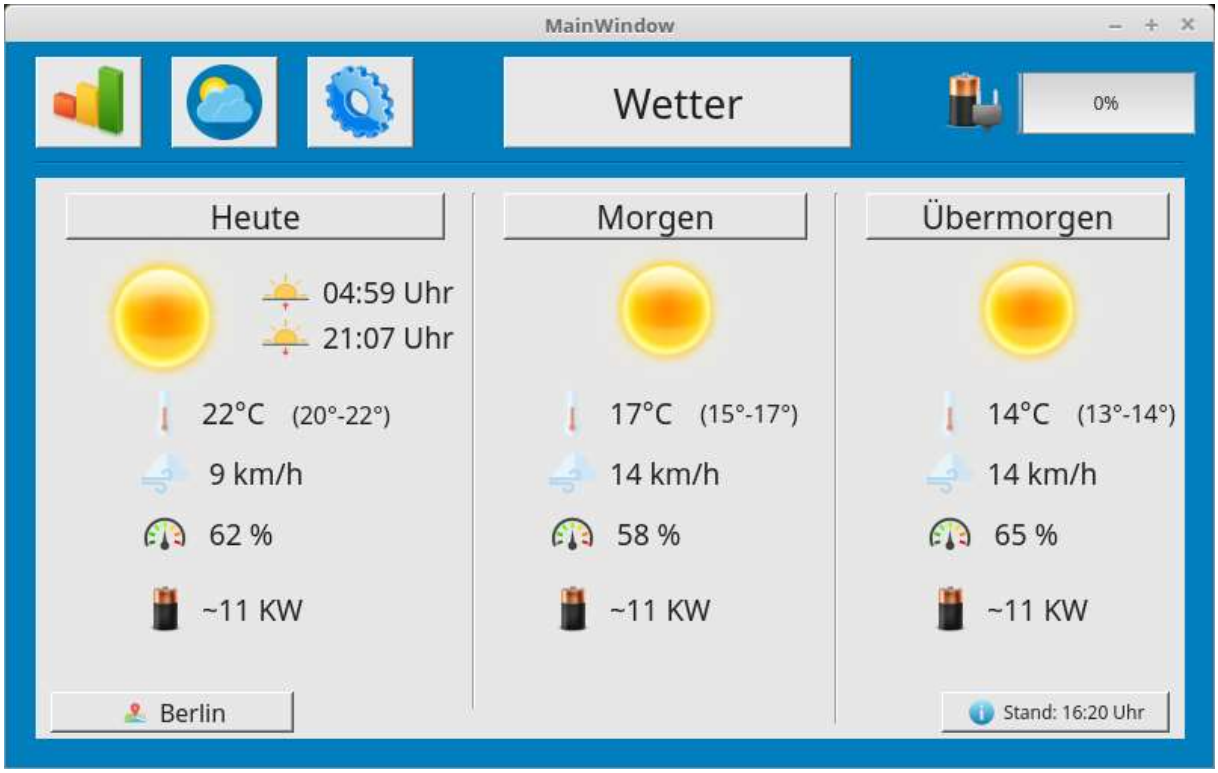
Hier werden die einzelnen Untermenüs in der Grafik dargestellt und erläutert.S

5.2.1 Startbild

5-2 Menüpunkt Startbild

Aktuelle Daten werden von Sensoren in Echtzeit ausgelesen.
Die Daten unter den Punkten Stunden, Tag, Woche und Monat sind der Durchschnittswert aus der Datenbank.
In regelmäßigen Abständen werden Verbrauch- und Erzeugungsdaten in die Datenbank eingetragen.
Die Akkuanzeige in Prozent werden ermittelt durch die aktuelle Spannung der und der Mindest- bzw. Maximal Spannung der Batterie.

5.2.2 Wetter



5-3 Menüpunkt Wetterübersicht

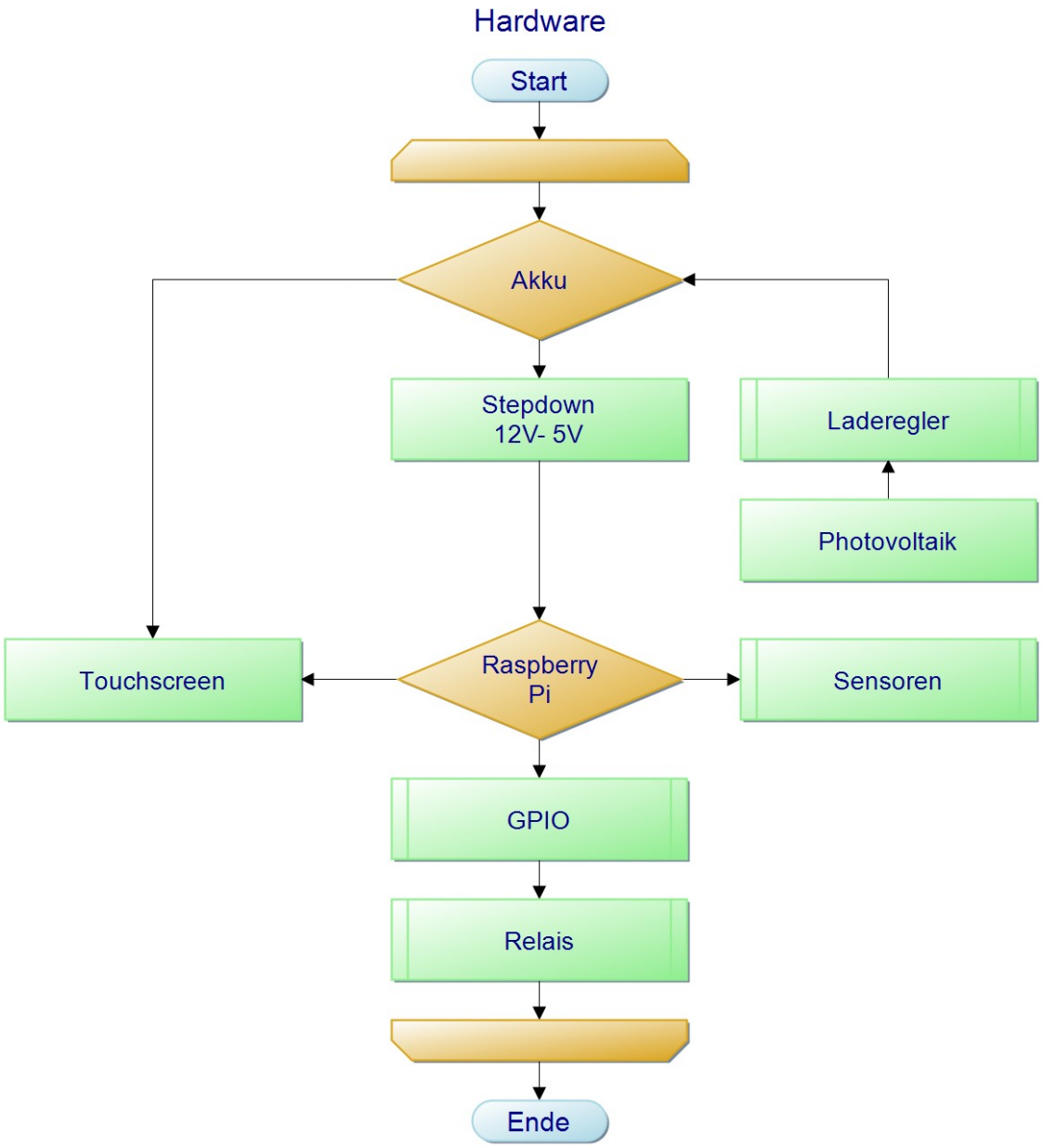
Temperatur, Windgeschwindigkeit, Luftfeuchtigkeit und Energieprognose sind Bestandteil dieses Menüs.
Die Prognose der Energieerzeugung ergibt sich aus den Wetterverhältnissen.

5.2.3 Einstellungen

5-4 Menüpunkt Einstellungen

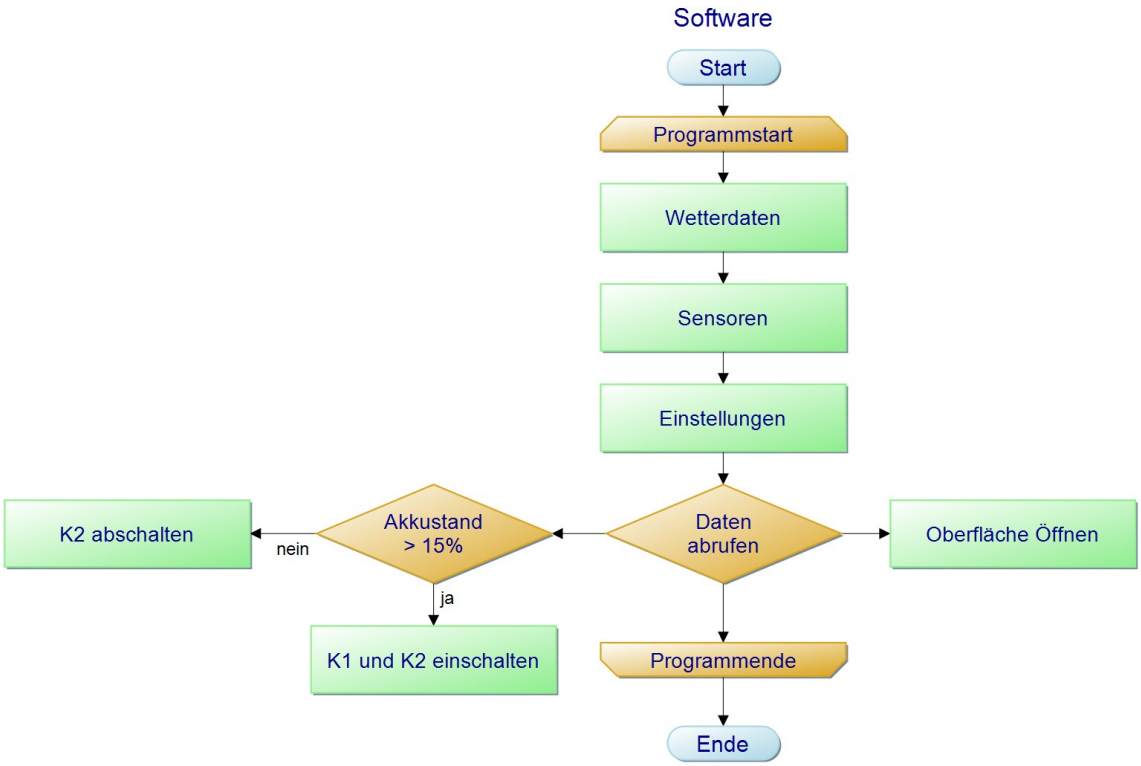
In diesem Menü kann der Benutzer die Einstellungen der Batteriekapazität, die maximale Leistung der Photovoltaikanlage oder des Windrades eingeben.
Zusätzlich kann das System Heruntergefahren, Neugestartet oder das Display in den Ruhezustand versetzt werden.
Wenn der Energiesparmodus deaktiviert wird, kann das System weiter laufen. Die Abschaltung bei einer Kapazität unter 15% wird dabei ignoriert.

5.3 PAP Hardware



5-5PAP Hardware

5.4 PAP Software



5-6 PAP Software

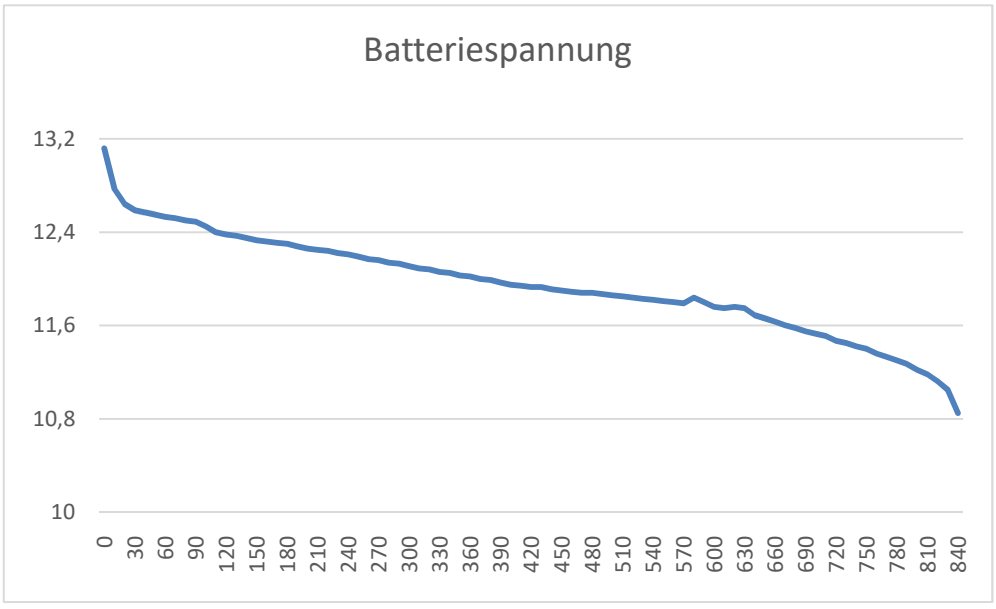
6 Prognosen

6.1 Kapazitätsprognose

Um eine geeignete Aussage treffen zu können, wie lange ein Akku bei einer konstanten Last noch in Betrieb sein kann, sind mehrere Faktoren zu berücksichtigen.
In diesem Projekt liegt das Hauptaugenmerk auf die Überwachung. Da es Zeitlich nicht möglich ist, sich intensiver mit dem Thema der Entladung eines Akkus auseinanderzusetzen, wird unsererseits eine grobe Annäherung erarbeitet.

Für die Erstellung einer Entladekurve wählten wir eine Stromstärke von 500mA. Diese wurde durch ein Widerstand mit 24Ω umgesetzt.
Über einen Zeitraum von 14 Stunden war es möglich, eine Kurve zu erstellen.
Diese Kurve ist durch den konstanten Verbrauch nicht Praxisnah. Da der Einfluss durch schwankende Verbraucher, Temperatur und Zustand der Batterie nicht berücksichtigt wurden.
Sie gibt lediglich eine grobe Aussage darüber, wie lange die Batterie mit dem aktuellen Verbrauchswert noch auskommen würde.
Das nachfolgende Diagramm stellt den Spannungsfall in einem Zeitverlauf dar.

Diagramm 1: Spannungsfall gegenüber Stromverbrauch (Zeitverlauf in Minuten)



6-1 Entladekurve

6.2 Wetterprognose

Das Abrufen der Wetterdaten ist von großer Bedeutung. Nicht nur dem Benutzer werden auf einem Blick die aktuellen Daten sowie der kommenden zwei Tage angezeigt. Auch für das Programm und der Berechnung für die Prognose sind sie von großer Wichtigkeit. Doch wie gelang man an diese Daten und wie werden sie eingebettet?

6.2.1 Wetterdaten online abrufen

Bei dem Wetterdienst www.openweathermap.com wird eine Funktion kostenlos zur Verfügung gestellt. Diese ermöglicht das Abrufen der Daten für die nächsten 16 Tage. Der Dienst ist kostenfrei, jedoch mit einigen Einschränkungen. Lediglich eine einmalige Anmeldung war nötig, um an den API-key zu gelangen. Der Code wird in der Programmierung eingepflegt. Zusätzlich werden die Ortsdaten benötigt. Dadurch ist das Abrufen der Daten Ortsgebunden. Es gibt eine Möglichkeit, die es ermöglicht den Ort manuell im System einträgt. Jedoch muss dieser auch bei „Openweathermap“ zur Verfügung stehen.
Die Wetterdaten von Openweathermap bekommt man als JSON Datentyp, diese müssen verarbeitet und formatiert werden, damit sie mit Python verwendet werden können.

Der Code sieht wie folgt aus:

```
def url_builder():
"""URL von Openweathermap erzeugen.
Es werden Daten der Tagesprognose und Vorhersage benötigt.
Daten übergeben an data_fetch Funktion"""
user_api = "ec5f8cf7b573ec7ed472d1e04cfa5a37"
city_id = "Berlin,de"
unit = "metric"  # Fahrenheit = imperial, Celsius = metric, Kelvin = default.
# Für Koordinaten api.openweathermap.org/data/2.5/weather?lat=35&lon=139
api_current = "http://api.openweathermap.org/data/2.5/weather?q="
api_forecast = "http://api.openweathermap.org/data/2.5/forecast?q="
# City ID hier: http://bulk.openweathermap.org/sample/city.list.json.g
api_url_current = api_current + str(city_id) + "&mode=json&units=" + unit + "&APPID=" + user_api
api_url_forecast = api_forecast + str(city_id) + "&mode=json&units=" + unit + "&APPID=" + user_api + "&cnt=3"
full_api_url = (api_url_current, api_url_forecast)
return full_api_url
```

7 Fazit

In diesem Projekt wurde ein Versuchsaufbau geschaffen, indem ein Überwachungssystem für autarke Anlagen erstellt wurde. Die Umsetzung erfolgte mittels Mikrocontroller und der Programmiersprache Python. Durch dieses Zusammenspiel wurde für uns als „Anfänger“ in diesem Gebiet eine Interessante Herausforderung geschaffen. Jedoch erwies es sich teilweise als große Herausforderung, um bestimmte Programmcodes zu erstellen. Da sich im Internet kein vergleichsweises Projekt finden ließ, war es schon eine Hürde die Oberfläche zu schaffen und beispielsweise die abgerufenen Daten über www.openweathermap.com einzubinden. Im Laufe des Projektes wurde klar, dass wir uns nicht nur auf die „reine“ Software orientieren konnten, da das Auslesen der Strom und Spannungen eine wichtige Rolle spielen. Diese werden in Echtzeit benötigt und somit kamen immer mehr Komponenten wie Sensoren, Spannungswandler, Widerstände, das Erstellen einer Entladekurve und einen AD Wandler dazu. Zusätzlich musste eine Platine geschaffen werden, um die Komponenten zu verbinden. Dies hat viel Zeit in Anspruch genommen. Auch das herstellen einer Box für alle Bauteile Zwang zum Umstrukturieren.
Zum Schluss aber können wir ein System vorstellen, dass in der Lage ist, sämtliche Informationen grafisch, sowie in Textform anzuzeigen. Auch die Umsetzung eine Energiesparmodus und die Abschaltung der Endverbraucher ist uns gelungen. Leider hat die Funktion des Touchpanels ein defekt bekommen. Somit ist die Vorführung nur über eine Maus möglich.
Für uns war dieses Projekt ein voller Erfolg und ein Einblick in die Umsetzung zwischen Soft- und Hardware. Zudem ist dieses Projekt für uns zum privaten nutzen gedacht.

9 Anhang

9.1 Tabellenverzeichnis

9.2 Programmcode

```
'''Werte der Aufzeichnung von der Entladekurve
eines 7Ah Bleigel Akku,
mit 500mA konstant Verbrauch
'''
global AkkuAmpere

#Listen
AkkuZeitlist = [
    0,
    10,
    20,
    30,
    40,
    50,
    60,
    70,
    80,
    90,
    100,
    110,
    120,
    130,
    140,
    150,
    160,
    170,
    180,
    190,
    200,
    210,
    220,
    230,
    240,
    250,
    260,
    270,
```

280,
290,
300,
310,
320,
330,
340,
350,
360,
370,
380,
390,
400,
410,
420,
430,
440,
450,
460,
470,
480,
490,
500,
510,
520,
530,
540,
550,
560,
570,
580,
590,
600,
610,
620,
630,
640,
650,
660,
670,
680,
690,
700,
710,
720,
730,

```
    740,
    750,
    760,
    770,
    780,
    790,
    800,
    810,
    820,
    830,
    840
]
AkkuVoltlist = [
    13.12,
    12.77,
    12.64,
    12.59,
    12.57,
    12.55,
    12.53,
    12.52,
    12.5,
    12.49,
    12.45,
    12.4,
    12.38,
    12.37,
    12.35,
    12.33,
    12.32,
    12.31,
    12.3,
    12.28,
    12.26,
    12.25,
    12.24,
    12.22,
    12.21,
    12.19,
    12.17,
    12.16,
    12.14,
    12.13,
    12.11,
    12.09,
    12.08,
```

12.06,
12.05,
12.03,
12.02,
12.0,
11.99,
11.97,
11.95,
11.94,
11.93,
11.93,
11.91,
11.9,
11.89,
11.88,
11.88,
11.87,
11.86,
11.85,
11.84,
11.83,
11.82,
11.81,
11.8,
11.79,
11.84,
11.8,
11.76,
11.75,
11.76,
11.75,
11.69,
11.66,
11.63,
11.6,
11.58,
11.55,
11.53,
11.51,
11.47,
11.45,
11.42,
11.4,
11.36,
11.33,
11.3,

```
    11.27,
    11.22,
    11.18,
    11.12,
    11.05,
    10.85
]
```

```
def test():
    lokalvar = "Lokal Variable!"
    lokalvar2 = "Die zweite"
    lokalvar3 = "NUMMER3"
    Werte = [lokalvar, lokalvar2,
    lokalvar3]
    return Werte

RestzeitStunden = (28+0.6) / 0.5
print(RestzeitStunden)
RestzeitTage = int(RestzeitStunden / 24)
print(RestzeitTage)
RestzeitStunden = RestzeitStunden - (RestzeitTage * 24)
print(RestzeitStunden)
```

```
'''@package docstring
Hauptprogramm
'''
import sys #System lib
import os #Shell Lib
import PyQt5 #PyQt Lib
from PyQt5.QtWidgets import *
from PyQt5.QtCore import QTimer

#Eigene Module
import mainwindow
import overview
import weather
from weather import QtGui
import settings
import owm
import sensors
import mysql
import qdialog
import Akkuwerte


#--------------Relais-Initialisieren----------------
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)    #RPI Layout setzten
GPIO.setup(38, GPIO.OUT)    #Relais 1
GPIO.setup(40, GPIO.OUT)    #Relais 2
```

```
#---------------------------------------------------

class MainWindow(QMainWindow, mainwindow.Ui_MainWindow):
    # FENSTERNAME(QMainWindow, dateiname.Objektname):
    '''Class zum Erzeugung des Hauptfensters'''

    def __init__(self):
        '''Constructor'''

        super(self.__class__, self).__init__()
        '''Fenster und Widgets initialisieren'''
        self.MainLayout(self)

        self.overview_widget = QWidget()
        self.overview_manager = overview.Ui_Overview()
        self.overview_manager.setupUi(self.overview_widget)

        self.weather_widget = QWidget()
        self.weather_manager = weather.Ui_Weather()
        self.weather_manager.setupUi(self.weather_widget)

        self.settings_widget = QWidget()
        self.settings_manager = settings.Ui_Settings()
        self.settings_manager.setupUi(self.settings_widget)

        self.content.setLayout(QStackedLayout())
        self.content.layout().addWidget(self.overview_widget)
        self.content.layout().addWidget(self.weather_widget)
        self.content.layout().addWidget(self.settings_widget)
        self.setContent(self.overview_widget)

        '''Menübutton aktionen'''
        self.m_overview.clicked.connect(lambda: self.setContent(self.overview_widget))
        self.m_overview.clicked.connect(lambda: self.title_bg.setText("Übersicht"))
        self.m_weather.clicked.connect(lambda: self.setContent(self.weather_widget))
        self.m_weather.clicked.connect(lambda: self.title_bg.setText("Wetter"))
        self.m_settings.clicked.connect(lambda: self.setContent(self.settings_widget))
        self.m_settings.clicked.connect(lambda: self.title_bg.setText("Einstellung"))

        '''Button aktionen festlegen'''
        self.settings_manager.b_displayoff.clicked.connect(lambda: os.system("xset dpms force off"))
        self.settings_manager.b_shutdown.clicked.connect(lambda: os.system("shutdown -h now"))
        self.settings_manager.b_reboot.clicked.connect(lambda: os.system("shutdown -r now"))

        '''QTimer Funktion um Daten konstant zu aktualisieren (Multithreads)'''
        CountMinute = 60*1000
        CountStunde = 60*60*1000
        DialogBatteryLow = 0
        DialogBatteryEmpty = 0

        #Akkuanzeige
        global Dialog1
        global Dialog2
        Dialog1 = 0 #LowBattery bereits Dialog gesehen?
        Dialog2 = 0 #EmptyBattery bereits Dialog gesehen?
        self.BatterieMonitor()
```

```
        overviewTimer1 = QTimer(self)
        overviewTimer1.timeout.connect(lambda: self.BatterieMonitor())
        overviewTimer1.start(3000)

        #Overview anzeige
        self.UpdateOverview()
        overviewTimer2 = QTimer(self)
        overviewTimer2.timeout.connect(lambda: self.UpdateOverview())
        overviewTimer2.start(CountStunde)

        #Overview anzeige
        self.UpdateCurrentSensors()
        overviewTimer3 = QTimer(self)
        overviewTimer3.timeout.connect(lambda: self.UpdateCurrentSensors())
        overviewTimer3.start(500)

        #Wetter
        self.UpdateWeather()
        WeatherTimer = QTimer(self)
        WeatherTimer.timeout.connect(lambda: self.UpdateWeather())
        WeatherTimer.start(CountStunde)

        #Settings
        self.UpdateSettings()
        SettingsTimer = QTimer(self)
        SettingsTimer.timeout.connect(lambda: self.UpdateSettings())
        SettingsTimer.start(100)

        #MYSQL
        global erzSUMamp
        global erzSUMvolt
        global verbSUMamp
        global verbSUMvolt
        global counter
        erzSUMamp = 0
        erzSUMvolt = 0
        verbSUMamp = 0
        verbSUMvolt = 0
        counter = 0
        MYSQLperiode = 10
        self.UpdateMYSQL(MYSQLperiode)
        MYSQLtimer = QTimer(self)
        MYSQLtimer.timeout.connect(lambda: self.UpdateMYSQL(MYSQLperiode))
        MYSQLtimer.start(CountMinute)

    def setContent(self, widget):
        '''Content Widget ändern bei Buttonklick'''
        self.content.layout().setCurrentWidget(widget)

    def BatterieMonitor(self):
        '''Batterieüberwachung
        Prozentanzeige und Warnmeldungen
        sowie Relaissteuerung bei kritischem Zustand'''
        global Dialog1
        global Dialog2
        SensorData = sensors.UpdateSensors()
```

```
        self.capacity.setProperty("value", SensorData[5])
        if(SensorData[5] <= 5) and (Dialog1 == 0):
            GPIO.output(38, GPIO.HIGH) #Licht
            GPIO.output(40, GPIO.HIGH) #Allgemein
            qdialog.BatteryEmpty()
            Dialog1 = 1 #Warnung gesehen
        elif(SensorData[5] <= 15) and (SensorData[5] > 5) and (Dialog2 == 0):
            GPIO.output(38, GPIO.LOW) #Licht
            energysafer = self.settings_manager.b_energysafer
            if energysafer.isChecked():
                GPIO.output(40, GPIO.LOW) #Allgemein
            else:
                qdialog.BatteryLow()
                GPIO.output(40, GPIO.HIGH)
                Dialog2 = 1 #Warnung gesehen
        elif(SensorData[5] > 15):
            GPIO.output(38, GPIO.LOW) #Licht
            GPIO.output(40, GPIO.LOW) #Allgemein
            Dialog1 = 0
            Dialog2 = 0

    def UpdateOverview(self):
        mysql.askMYSQL()
        """Erzeugung darstellen"""
        self.overview_manager.erz_stunde.setText("%.2f W / Stunde" % mysql.e_leistung_stunde)
        self.overview_manager.erz_tag.setText("%.2f kW / Tag" % mysql.e_leistung_tag)
        self.overview_manager.erz_woche.setText("%.2f kW / Woche" % mysql.e_leistung_woche)
        self.overview_manager.erz_monat.setText("%.2f kW / Monat" % mysql.e_leistung_monat)

        """Verbrauch darstellen"""
        self.overview_manager.ver_stunde.setText("%.2f W / Stunde" % mysql.v_leistung_stunde)
        self.overview_manager.ver_tag.setText("%.2f kW / Tag" % mysql.v_leistung_tag)
        self.overview_manager.ver_woche.setText("%.2f kW / Woche" % mysql.v_leistung_woche)
        self.overview_manager.ver_monat.setText("%.2f kW / Monat" % mysql.v_leistung_monat)


    def UpdateCurrentSensors(self):
        """Aktuelle Erzeugung und Vebrauch anzeigen"""
        SensorData = sensors.UpdateSensors()
        #Sensor DICT [PV_Volt, PV_Ampere, Verb_Volt, Verb_Ampere, AkkuVolt,
        #AkkuProzent, Erz_Leistung, Verb_Leistung]
        self.overview_manager.ver_jetzt.setText("%.3f W / Aktuell" % SensorData[7])
        self.overview_manager.erz_jetzt.setText("%.3f W / Aktuell" % SensorData[6])
        """Berechnung Akkukapazität"""
        AkkuAmpere = "na"
        while(AkkuAmpere == "na"):
            for position, item in enumerate(Akkuwerte.AkkuVoltlist):
                if item == SensorData[4]:
                    AkkuAmpere = 7 - (Akkuwerte.AkkuZeitlist[position] / 60) * 0.5
                    AkkuAmpere = round(AkkuAmpere,3)
            if AkkuAmpere == "na":
                SensorData[4] = SensorData[4] + 0.01
                SensorData[4] = round(SensorData[4], 2)
        #------------------(Akkuampere + Aktuelle Erz) / Aktuellen Verbrauch
        RestzeitStunden = (AkkuAmpere + SensorData[1]) / SensorData[3]
        RestzeitTage = int(RestzeitStunden / 24)
```

```
        RestzeitStunden = RestzeitStunden - (RestzeitTage * 24)
        self.overview_manager.prognose.setText("{:2.0f} Tage {:2.0f} Stunden".format(RestzeitTage, Restzeit-
Stunden))


    def UpdateWeather(self):
        """Neue Wetterdaten einbinden"""
        try:
            owm.data_organizer(owm.data_fetch(owm.url_builder()))
            self.weather_manager.b_refresh_status.setText("Stand: {} Uhr".format(owm.daytime))
            self.weather_manager.b_location.setText(owm.city)
            self.weather_manager.temp.setText("{:2.0f}°C".format(owm.temp[0]))
            self.weather_manager.temp_2.setText("{:2.0f}°C".format(owm.temp[1]))
            self.weather_manager.temp_3.setText("{:2.0f}°C".format(owm.temp[2]))
            self.weather_manager.temp_minmax.setText("({0:2.0f}°-{1:2.0f}°)".format(owm.temp_min[0],
owm.temp_max[0]))
            self.weather_manager.temp_minmax_2.setText("({0:2.0f}°-{1:2.0f}°)".format(owm.temp_min[1],
owm.temp_max[1]))
            self.weather_manager.temp_minmax_3.setText("({0:2.0f}°-{1:2.0f}°)".format(owm.temp_min[2],
owm.temp_max[2]))
            self.weather_manager.wind.setText("{:2.0f} km/h".format(owm.wind[0]))
            self.weather_manager.wind_2.setText("{:2.0f} km/h".format(owm.wind[1]))
            self.weather_manager.wind_3.setText("{:2.0f} km/h".format(owm.wind[2]))
            self.weather_manager.humidity.setText(" {:2.0f} %".format(owm.humidity[0]))
            self.weather_manager.humidity_2.setText(" {:2.0f} %".format(owm.humidity[1]))
            self.weather_manager.humidity_3.setText(" {:2.0f} %".format(owm.humidity[2]))
            self.weather_manager.sunrise.setText("{} Uhr".format(owm.sunrise))
            self.weather_manager.sunset.setText("{} Uhr".format(owm.sunset))

            """Wolken darstellung & Prognose berechnen"""
            #------------Prognose Einstellung--------------"""""""
            ProgPanels = []
            ProgWind = []
            ProgErz = []
            sonnenstunden = 9 #Prognose nicht fertig
            windstunden = 20 #Prognose nicht fertig
            SensorData = sensors.UpdateSensors()
            """Ampere aus Settings * die aktuelle Akkuspannung"""
            panels = float(self.settings_manager.panels.text())
            wind = float(self.settings_manager.wind.text())
            PanelErzTag = panels * sonnenstunden
            WindErzTag = wind * windstunden
            #-----------------------------------------------
            clouds = [self.weather_manager.clouds, self.weather_manager.clouds_2,
self.weather_manager.clouds_3]
            for i in range(0, 3):
                if((owm.condition[i] >= 200) and (owm.condition[i] < 300)): #Gewitter
                    clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/0.png"))
                    ProgPanels.append(PanelErzTag - (PanelErzTag * 0.9))
                elif((owm.condition[i] <= 300) and (owm.condition[i] < 600)): #Regen
                    clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/11.png"))
                    ProgPanels.append(PanelErzTag - (PanelErzTag * 0.85))
                elif((owm.condition[i] <= 600) and (owm.condition[i] < 700)): #Schnee
                    clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/14.png"))
                    ProgPanels.append(PanelErzTag - (PanelErzTag * 0.6))
                elif((owm.condition[i] <= 700) and (owm.condition[i] < 800)): #Nebel, Rauch, Tornado
```

```
                clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/20.png"))
                ProgPanels.append(PanelErzTag - (PanelErzTag * 0.92))
            elif(owm.condition[i] >= 800): #Wolken
                if(owm.condition[i] == 800): #clear
                    clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/32.png"))
                    ProgPanels.append(PanelErzTag - (PanelErzTag * 0.1))
                elif(owm.condition[i] == 801): #wenig
                    clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/34.png"))
                    ProgPanels.append(PanelErzTag - (PanelErzTag * 0.3))
                elif(owm.condition[i] == 802): #leicht
                    clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/30.png"))
                    ProgPanels.append(PanelErzTag - (PanelErzTag * 0.5))
                elif(owm.condition[i] == 803): #mittel
                    clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/28.png"))
                    ProgPanels.append(PanelErzTag - (PanelErzTag * 0.6))
                elif(owm.condition[i] == 801): #stark
                    clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/26.png"))
                    ProgPanels.append(PanelErzTag - (PanelErzTag * 0.9))
                else: #Nicht Verfügbar
                    clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/na.png"))
                    ProgPanels.append(0)
            else: #Nicht Verfügbar
                clouds[i].setPixmap(QtGui.QPixmap(":/weather/img/weather/na.png"))
                ProgPanels.append(0)

        ProgErz1 = ProgPanels[0] / 24# + ProgWind[0]
        ProgErz2 = ProgPanels[1] / 24# + ProgWind[1]
        ProgErz3 = ProgPanels[2] / 24# + ProgWind[2]
        self.weather_manager.prog_erz.setText("~{:1.1f} KW/h".format(ProgErz1))
        self.weather_manager.prog_erz_2.setText("~{:1.1f} KW/h".format(ProgErz2))
        self.weather_manager.prog_erz_3.setText("~{:1.1f} KW/h".format(ProgErz3))

    except IOError:
        print("Kein internet [WETTER]")
        qdialog.NoInternet()

def UpdateSettings(self):
    """Energiesparmodus verwalten"""
    energysafer = self.settings_manager.b_energysafer
    if energysafer.isChecked():
        energysafer.setStyleSheet("color: rgb(0, 121, 16);") #Text Grün
        energysafer.setText("Aktivieren")
    elif not energysafer.isChecked():
        energysafer.setStyleSheet("color: rgb(199, 0, 0);") #Text Rot
        energysafer.setText("Deaktivieren")

def UpdateMYSQL(self, periode):
    global counter
    global erzSUMamp
    global erzSUMvolt
    global verbSUMamp
    global verbSUMvolt
    if counter < periode:
        SensorData = sensors.UpdateSensors()
        #Sensor DICT [PV_Volt, PV_Ampere, Verb_Volt, Verb_Ampere, AkkuVolt,
        #AkkuProzent, Erz_Leistung, Verb_Leistung]
```

```
            erzSUMvolt = erzSUMvolt + SensorData[0]
            erzSUMamp = erzSUMamp + SensorData[1]
            verbSUMvolt = verbSUMvolt + SensorData[2]
            verbSUMamp = verbSUMamp + SensorData[3]
            counter = counter + 1
            print(counter)
        elif counter == periode:
            mysql.SensorEntry(periode, erzSUMvolt, erzSUMamp, verbSUMvolt, verbSUMamp)
            erzSUMvolt = 0
            erzSUMamp = 0
            verbSUMamp = 0
            verbSUMvolt = 0
            counter = 0

def main():
    '''App Instanz erzeugen'''
    app = QApplication(sys.argv)
    form = MainWindow()
    form.show()
    sys.exit(app.exec_())


if __name__ == '__main__':
    '''MainProgramm abruf'''
    main()
```

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'mainwindow.ui'
#
# Created by: PyQt5 UI code generator 5.5.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def MainLayout(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 480)
        MainWindow.setStyleSheet("background-color: rgb(0, 126, 189);")
        MainWindow.showFullScreen()
        self.centralWidget = QtWidgets.QWidget(MainWindow)
        self.centralWidget.setObjectName("centralWidget")
        self.capacity = QtWidgets.QProgressBar(self.centralWidget)
        self.capacity.setGeometry(QtCore.QRect(670, 20, 118, 41))
        self.capacity.setProperty("value", 0)
        self.capacity.setObjectName("capacity")
        self.img_akku = QtWidgets.QLabel(self.centralWidget)
        self.img_akku.setGeometry(QtCore.QRect(620, 20, 41, 41))
        self.img_akku.setText("")
        self.img_akku.setPixmap(QtGui.QPixmap(":/system/img/system/if_energy_6031.png"))
        self.img_akku.setScaledContents(True)
        self.img_akku.setObjectName("img_akku")
        self.m_overview = QtWidgets.QPushButton(self.centralWidget)
```

```
        self.m_overview.setEnabled(True)
        self.m_overview.setGeometry(QtCore.QRect(20, 10, 71, 61))
        self.m_overview.setAutoFillBackground(False)
        self.m_overview.setStyleSheet("background-color: rgb(232, 232, 232);")
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap(":/system/img/system/if_Stats_32681.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        self.m_overview.setIcon(icon)
        self.m_overview.setIconSize(QtCore.QSize(50, 70))
        self.m_overview.setCheckable(False)
        self.m_overview.setChecked(False)
        self.m_overview.setAutoRepeat(False)
        self.m_overview.setObjectName("m_overview")
        self.m_weather = QtWidgets.QPushButton(self.centralWidget)
        self.m_weather.setGeometry(QtCore.QRect(110, 10, 71, 61))
        self.m_weather.setAutoFillBackground(False)
        self.m_weather.setStyleSheet("background-color: rgb(232, 232, 232);")
        self.m_weather.setText("")
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap(":/system/img/system/if_Weather_669958.png"),
QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.m_weather.setIcon(icon1)
        self.m_weather.setIconSize(QtCore.QSize(50, 50))
        self.m_weather.setCheckable(False)
        self.m_weather.setChecked(False)
        self.m_weather.setAutoRepeat(False)
        self.m_weather.setAutoExclusive(False)
        self.m_weather.setObjectName("m_weather")
        self.m_settings = QtWidgets.QPushButton(self.centralWidget)
        self.m_settings.setGeometry(QtCore.QRect(200, 10, 71, 61))
        self.m_settings.setStyleSheet("background-color: rgb(232, 232, 232);")
        self.m_settings.setText("")
        icon2 = QtGui.QIcon()
        icon2.addPixmap(QtGui.QPixmap(":/system/img/system/if_Settings_105244.png"),
QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.m_settings.setIcon(icon2)
        self.m_settings.setIconSize(QtCore.QSize(50, 70))
        self.m_settings.setCheckable(False)
        self.m_settings.setChecked(False)
        self.m_settings.setAutoRepeat(False)
        self.m_settings.setObjectName("m_settings")
        self.title_bg = QtWidgets.QPushButton(self.centralWidget)
        self.title_bg.setGeometry(QtCore.QRect(330, 10, 231, 61))
        self.title_bg.setStyleSheet("background-color: rgb(232, 232, 232);\n"
"font: 75 20pt \"Noto Sans\";")
        self.title_bg.setObjectName("title_bg")
        self.line = QtWidgets.QFrame(self.centralWidget)
        self.line.setGeometry(QtCore.QRect(20, 70, 761, 21))
        self.line.setFrameShape(QtWidgets.QFrame.HLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.content = QtWidgets.QWidget(self.centralWidget)
        self.content.setGeometry(QtCore.QRect(20, 90, 761, 371))
        self.content.setObjectName("content")
        self.content.setStyleSheet("background-color: rgb(231, 231, 231);")
        self.img_akku.raise_()
```

```
        self.m_overview.raise_()
        self.capacity.raise_()
        self.m_weather.raise_()
        self.m_settings.raise_()
        self.title_bg.raise_()
        self.line.raise_()
        self.content.raise_()
        MainWindow.setCentralWidget(self.centralWidget)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.title_bg.setText(_translate("MainWindow", "Übersicht"))

import resources_rc
```

```
import MySQLdb

def askMYSQL():
    """Mit Datenbank verbinden"""
    db = MySQLdb.connect("localhost", "Ole", "IchmagBrot!", "project")
    # Mysqldb.connect("HOST, USER, PW, DATABASE")
    curs = db.cursor()

    """Datenbankabfrage der Mittelwerte"""
    try:
        #---------------------ERZEUGUNG----------------------#
        curs.execute("SELECT AVG(leistung) FROM erzeugung WHERE date >= CURRENT_TIMESTAMP -
INTERVAL 1 HOUR")
        for leistung in curs.fetchone():
            global e_leistung_stunde
            e_leistung_stunde = leistung
            if e_leistung_stunde == None:
                e_leistung_stunde = 0

        curs.execute("SELECT AVG(leistung) FROM erzeugung WHERE date >= CURRENT_TIMESTAMP -
INTERVAL 24 HOUR")
        for leistung in curs.fetchone():
            global e_leistung_tag
            e_leistung_tag = leistung
            if e_leistung_tag == None:
                e_leistung_tag = 0
            elif e_leistung_tag > 0:
                e_leistung_tag = e_leistung_tag/1000

        curs.execute ("SELECT AVG(leistung) FROM erzeugung WHERE date >= CURRENT_TIMESTAMP -
INTERVAL 7 DAY")
        for leistung in curs.fetchone():
            global e_leistung_woche
            e_leistung_woche = leistung
            if e_leistung_woche == None:
                e_leistung_woche = 0
```

```
        elif e_leistung_woche > 0:
            e_leistung_woche = e_leistung_woche/1000

    curs.execute ("SELECT AVG(leistung) FROM erzeugung WHERE date >= CURRENT_TIMESTAMP -
INTERVAL 30 DAY")
    for leistung in curs.fetchone():
        global e_leistung_monat
        e_leistung_monat = leistung
        if e_leistung_monat == None:
            e_leistung_monat = 0
        elif e_leistung_monat > 0:
            e_leistung_monat = e_leistung_monat/1000

    #---------------------VERBRAUCH----------------------#
    curs.execute("SELECT AVG(leistung) FROM verbrauch WHERE date >= CURRENT_TIMESTAMP -
INTERVAL 1 HOUR")
    for leistung in curs.fetchone():
        global v_leistung_stunde
        v_leistung_stunde = leistung
        if v_leistung_stunde == None:
            v_leistung_stunde = 0

    curs.execute("SELECT AVG(leistung) FROM verbrauch WHERE date >= CURRENT_TIMESTAMP -
INTERVAL 24 HOUR")
    for leistung in curs.fetchone():
        global v_leistung_tag
        v_leistung_tag = leistung
        if v_leistung_tag == None:
            v_leistung_tag = 0
        elif v_leistung_tag > 0:
            v_leistung_tag = v_leistung_tag/1000

    curs.execute ("SELECT AVG(leistung) FROM verbrauch WHERE date >= CURRENT_TIMESTAMP -
INTERVAL 7 DAY")
    for leistung in curs.fetchone():
        global v_leistung_woche
        v_leistung_woche = leistung
        if v_leistung_woche == None:
            v_leistung_woche = 0
        elif v_leistung_woche > 0:
            v_leistung_woche = v_leistung_woche/1000

    curs.execute ("SELECT AVG(leistung) FROM verbrauch WHERE date >= CURRENT_TIMESTAMP -
INTERVAL 30 DAY")
    for leistung in curs.fetchone():
        global v_leistung_monat
        v_leistung_monat = leistung
        if v_leistung_monat == None:
            v_leistung_monat = 0
        elif v_leistung_monat > 0:
            v_leistung_monat = v_leistung_monat/1000

    #-----Ampere/Tag------#
    curs.execute("SELECT AVG(leistung) FROM verbrauch WHERE date >= CURRENT_TIMESTAMP -
INTERVAL 24 HOUR")
    for strom in curs.fetchone():
```

```
            global v_ampere_tag
            v_ampere_tag = strom
            if v_ampere_tag == None:
                v_ampere_tag = 0
    except:
        print("MYSQL ERROR! [AskMYSQL]")
        db.rollback()

def SensorEntry(periode, erzSUMvolt, erzSUMamp, verbSUMvolt, verbSUMamp):
    '''Mit Datenbank verbinden'''
    db = MySQLdb.connect("localhost", "Ole", "IchmagBrot!", "project")
    # Mysqldb.connect("HOST, USER, PW, DATABASE")
    curs = db.cursor()

    '''Addieren der Sensorwerte im Minutentakt ausgelöst durch QTimer->UpdateMYSQL im Main.py
    Periodischer Eintrag des Mittelwert aller Daten in die MYSQL Datenbank'''
    try:
        erzVolt = erzSUMvolt / periode
        erzAmp = erzSUMamp / periode
        erzStrom = erzVolt * erzAmp
        verbVolt = verbSUMvolt / periode
        verbAmp = verbSUMvolt / periode
        verbStrom = verbVolt * verbAmp
        print("erzVolt: {}, erzAmp {}, erzStrom {}".format(erzVolt, erzAmp, erzStrom))
        curs = db.cursor()
        curs.execute("INSERT INTO erzeugung (date, volt, ampere, leistung) VALUES (CURRENT_TIMESTAMP,
%.2f, %.2f, %.2f);" % (erzVolt, erzAmp, erzStrom))
        curs.execute("INSERT INTO verbrauch (date, volt, ampere, leistung) VALUES (CURRENT_TIMESTAMP,
%.2f, %.2f, %.2f);" % (verbVolt, verbAmp, verbStrom))
        db.commit()
        print("Eintrag erstellt! [Erzeugung -> MYSQL]\n")
    except:
        print("MYSQL ERROR! [SensorEntry]")
        db.rollback()

# #----------------------Beispiel Eintrag erstellen---------------------#
# try:
#     spannung = 12
#     strom = 9
#     leistung = (spannung * strom)
#     curs.execute("INSERT INTO erzeugung (date, volt, ampere, leistung) VALUES (CURRENT_TIMESTAMP,
%.2f, %.2f, %.2f);" % (spannung, strom, leistung))
#     db.commit()
#     print("Eintrag erstellt [MYSQL.py]!\n")
# except:
#     print("Error. Rolling back.\n!")
#     db.rollback()

#z.B SELECT spalte1, spalte2 FROM tabelle ORDER BY vornamen
# #-----------------------------------------------------------------------#

# --------------------------ABFRAGE BEISPIEL--------------------------#
# curs.execute("SELECT * FROM erzeugung")
# for spalte in curs.fetchall():
# print("Ergebnisse am {0} um {1}Uhr war {2}Volt".format(spalte[1], spalte[2], spalte[3]))
```

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'overview.ui'
#
# Created by: PyQt5 UI code generator 5.5.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Overview(object):
    def setupUi(self, Overview):
        Overview.setObjectName("Overview")
        Overview.resize(761, 371)
        Overview.setAutoFillBackground(False)
        self.erz_stunde = QtWidgets.QLabel(Overview)
        self.erz_stunde.setGeometry(QtCore.QRect(20, 140, 201, 41))
        self.erz_stunde.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.erz_stunde.setObjectName("erz_stunde")
        self.line_4 = QtWidgets.QFrame(Overview)
        self.line_4.setGeometry(QtCore.QRect(500, 10, 20, 341))
        self.line_4.setFrameShape(QtWidgets.QFrame.VLine)
        self.line_4.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line_4.setObjectName("line_4")
        self.erz_monat = QtWidgets.QLabel(Overview)
        self.erz_monat.setGeometry(QtCore.QRect(20, 290, 201, 41))
        self.erz_monat.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.erz_monat.setObjectName("erz_monat")
        self.ver_stunde = QtWidgets.QLabel(Overview)
        self.ver_stunde.setGeometry(QtCore.QRect(300, 140, 201, 41))
        self.ver_stunde.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.ver_stunde.setObjectName("ver_stunde")
        self.ver_monat = QtWidgets.QLabel(Overview)
        self.ver_monat.setGeometry(QtCore.QRect(300, 290, 201, 41))
        self.ver_monat.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.ver_monat.setObjectName("ver_monat")
        self.line_5 = QtWidgets.QFrame(Overview)
        self.line_5.setGeometry(QtCore.QRect(240, 10, 20, 341))
        self.line_5.setFrameShape(QtWidgets.QFrame.VLine)
        self.line_5.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line_5.setObjectName("line_5")
        self.erz_woche = QtWidgets.QLabel(Overview)
        self.erz_woche.setGeometry(QtCore.QRect(20, 240, 201, 41))
        self.erz_woche.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.erz_woche.setObjectName("erz_woche")
        self.b_erzeugung = QtWidgets.QPushButton(Overview)
        self.b_erzeugung.setGeometry(QtCore.QRect(20, 10, 211, 51))
        self.b_erzeugung.setStyleSheet("font: 75 18pt \"Noto Sans\";")
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap(":/system/img/system/if_solar_64682.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        self.b_erzeugung.setIcon(icon)
        self.b_erzeugung.setIconSize(QtCore.QSize(30, 30))
        self.b_erzeugung.setObjectName("b_erzeugung")
        self.b_prognose = QtWidgets.QPushButton(Overview)
        self.b_prognose.setGeometry(QtCore.QRect(530, 10, 211, 51))
```

```
        self.b_prognose.setStyleSheet("font: 75 18pt \"Noto Sans\";")
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap(":/system/img/system/if_gnome-session-reboot_28664.png"),
QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.b_prognose.setIcon(icon1)
        self.b_prognose.setIconSize(QtCore.QSize(30, 30))
        self.b_prognose.setObjectName("b_prognose")
        self.b_verbrauch = QtWidgets.QPushButton(Overview)
        self.b_verbrauch.setGeometry(QtCore.QRect(270, 10, 221, 51))
        self.b_verbrauch.setStyleSheet("font: 75 18pt \"Noto Sans\";")
        icon2 = QtGui.QIcon()
        icon2.addPixmap(QtGui.QPixmap(":/system/img/system/if_Light_bulb_653262.png"),
QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.b_verbrauch.setIcon(icon2)
        self.b_verbrauch.setIconSize(QtCore.QSize(30, 30))
        self.b_verbrauch.setObjectName("b_verbrauch")
        self.ver_tag = QtWidgets.QLabel(Overview)
        self.ver_tag.setGeometry(QtCore.QRect(300, 190, 201, 41))
        self.ver_tag.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.ver_tag.setObjectName("ver_tag")
        self.ver_woche = QtWidgets.QLabel(Overview)
        self.ver_woche.setGeometry(QtCore.QRect(300, 240, 201, 41))
        self.ver_woche.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.ver_woche.setObjectName("ver_woche")
        self.erz_tag = QtWidgets.QLabel(Overview)
        self.erz_tag.setGeometry(QtCore.QRect(20, 190, 201, 41))
        self.erz_tag.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.erz_tag.setObjectName("erz_tag")
        self.prognose = QtWidgets.QLabel(Overview)
        self.prognose.setGeometry(QtCore.QRect(540, 90, 201, 41))
        self.prognose.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.prognose.setObjectName("prognose")
        self.erz_jetzt = QtWidgets.QLabel(Overview)
        self.erz_jetzt.setGeometry(QtCore.QRect(20, 90, 201, 41))
        self.erz_jetzt.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.erz_jetzt.setObjectName("erz_jetzt")
        self.ver_jetzt = QtWidgets.QLabel(Overview)
        self.ver_jetzt.setGeometry(QtCore.QRect(300, 90, 201, 41))
        self.ver_jetzt.setStyleSheet("font: 75 16pt \"Noto Sans\";")
        self.ver_jetzt.setObjectName("ver_jetzt")

        self.retranslateUi(Overview)
        QtCore.QMetaObject.connectSlotsByName(Overview)

    def retranslateUi(self, Overview):
        _translate = QtCore.QCoreApplication.translate
        Overview.setWindowTitle(_translate("Overview", "Form"))
        self.erz_stunde.setText(_translate("Overview", "000 W / Stunde"))
        self.erz_monat.setText(_translate("Overview", "000 kW / Monat"))
        self.ver_stunde.setText(_translate("Overview", "000 W / Stunde"))
        self.ver_monat.setText(_translate("Overview", "000 kW / Monat"))
        self.erz_woche.setText(_translate("Overview", "000 kW / Woche"))
        self.b_erzeugung.setText(_translate("Overview", "Erzeugung"))
        self.b_prognose.setText(_translate("Overview", "Prognose"))
        self.b_verbrauch.setText(_translate("Overview", "Verbrauch"))
        self.ver_tag.setText(_translate("Overview", "000 kW / Tag"))
```

```
        self.ver_woche.setText(_translate("Overview", "000 kW / Woche"))
        self.erz_tag.setText(_translate("Overview", "000 kW / Tag"))
        self.prognose.setText(_translate("Overview", "1 Tag 8 Stunden"))
        self.erz_jetzt.setText(_translate("Overview", "000 W / Aktuell"))
        self.ver_jetzt.setText(_translate("Overview", "000 W / Aktuell"))

import resources_rc
```

```
import json
import datetime
import urllib.request

def ConvertSpeed(speed):
        speed = speed * 3.6  # m/s in km/
        speed = round(speed,1)
        #speed = str(speed)
        return speed

def ConvertTime(time):
        """Zeitstempel formatieren"""
        converted_time = datetime.datetime.fromtimestamp(
                int(time)
        ).strftime('%H:%M')
        return converted_time

def url_builder():
        """URL von Openweathermap erzeugen.
        Es werden Daten der Tagesprognose und Vorhersage benötigt.
        Daten übergeben an data_fetch Funktion"""
        user_api = "ec5f8cf7b573ec7ed472d1e04cfa5a37"
        city_id = "Berlin,de"
        unit = "metric"  # Fahrenheit = imperial, Celsius = metric, Kelvin = default.
        # Für Koordinaten api.openweathermap.org/data/2.5/weather?lat=35&lon=139
        api_current = "http://api.openweathermap.org/data/2.5/weather?q="
        api_forecast = "http://api.openweathermap.org/data/2.5/forecast?q="
        # City ID hier: http://bulk.openweathermap.org/sample/city.list.json.gz

        api_url_current = api_current + str(city_id) + "&mode=json&units=" + unit + "&APPID=" + user_api
        api_url_forecast = api_forecast + str(city_id) + "&mode=json&units=" + unit + "&APPID=" + user_api
+ "&cnt=3"
        full_api_url = (api_url_current, api_url_forecast)
        return full_api_url


def data_fetch(full_api_url):
        """Daten aus URL abfrufen in UTF-8 dekodieren
        und an data_organizer übergeben"""
        #Current Weather
        current_url = urllib.request.urlopen(full_api_url[0])
        current_output = current_url.read().decode("utf-8")
        current_data = json.loads(current_output)
        current_url.close()

        #forecast Weather
        forecast_url = urllib.request.urlopen(full_api_url[1])
        forecast_output = forecast_url.read().decode("utf-8")
        forecast_data = json.loads(forecast_output)
```

```
        forecast_url.close()
        weatherData = (current_data, forecast_data)
        return weatherData


def data_organizer(weatherData):
        """Wetterdaten aus JSON Liste formatieren
        und an einzelne Variablen übergeben.
        """
        global daytime
        global temp
        global temp_min
        global temp_max
        global humidity
        global pressure
        global wind
        global condition
        global sunrise
        global sunset
        global city
        temp = []
        temp_min = []
        temp_max = []
        humidity = []
        clouds = []
        condition =[]
        pressure = []
        wind = []

        daytime = ConvertTime(weatherData[0].get("dt"))
        sunrise = ConvertTime(weatherData[0].get("sys").get("sunrise"))
        sunset = ConvertTime(weatherData[0].get("sys").get("sunset"))
        city = weatherData[1].get("city").get("name")

        for forecast in weatherData[1].get("list"):
                temp.append(forecast.get("main").get("temp"))
                temp_min.append(forecast.get("main").get("temp_min"))
                temp_max.append(forecast.get("main").get("temp_max"))
                humidity.append(forecast.get("main").get("humidity"))
                pressure.append(forecast.get("main").get("pressure"))
                condition.append(forecast.get("weather")[0].get("id"))
                wind.append(ConvertSpeed(forecast.get("wind").get("speed")))


        print("---------------------------------------")
        print("Letzter Stand: {}".format(daytime))
        print("Temp: {}".format(temp[0]))
        print("Temp: {}".format(temp[1]))
        print("Temp: {}".format(temp[2]))
        print("Max: {}, Min: {}".format(temp_max[0], temp_min[0]))
        print("Max: {}, Min: {}".format(temp_max[1], temp_min[1]))
        print("Max: {}, Min: {}".format(temp_max[2], temp_min[2]))
        print("Humidity: {}".format(humidity[0]))
        print("ID1: {}".format(condition[0]))
        print("ID2: {}".format(condition[1]))
        print("ID3: {}".format(condition[2]))
```

```
            # print("sky: {}".format(sky[0]))
            # print("sky: {}".format(sky[1]))
            # print("sky: {}".format(sky[2]))
            # print("Pressure: {}".format(pressure[0]))
            print("Wind: {} km/h".format(wind[0]))
            print("City: {}".format(city))
            print("----------------------------------------")


if __name__ == "__main__":
            try:
                        data_organizer(data_fetch(url_builder()))
            except IOError:
                        print("no internet")
```

```
import sys
#import os
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *

def Sample():
    '''Beispiel eines Dialogfensters'''
    app = QApplication(sys.argv)
    msg = QMessageBox()
    msg.setIcon(QMessageBox.Critical)
    # Warning, Critical, Question, Information
    msg.setText("This is a message box")
    msg.setInformativeText("This is additional information")
    msg.setDetailedText("Detials in message bereich:")
    msg.setStandardButtons(QMessageBox.Ok | QMessageBox.Cancel)
    msg.setWindowFlags(Qt.FramelessWindowHint) #Titlebar verbergen
    retval = msg.exec_() #Pflicht für Ausführung

def BatteryLow():
    '''Warnmeldung bei niedrigen Batteriezustand'''
    msg = QMessageBox()
    msg.setIcon(QMessageBox.Warning)
    msg.setText("Batterie kapazität ist gering!")
    msg.setInformativeText("Energiesparmodus wurde aktiviert.")
    msg.setStandardButtons(QMessageBox.Ok)
    msg.setWindowFlags(Qt.FramelessWindowHint) #Titlebar verbergen
    retval = msg.exec_() #Pflicht für Ausführung

def BatteryEmpty():
    '''Warnmeldung bei leerem Batteriezustand'''
    msg = QMessageBox()
    msg.setIcon(QMessageBox.Critical)
    msg.setText("Die Batterie Leer!")
    msg.setInformativeText("Die Relais wurden abgeschaltet.")
    msg.setStandardButtons(QMessageBox.Ok)
    msg.setWindowFlags(Qt.FramelessWindowHint) #Titlebar verbergen
    retval = msg.exec_() #Pflicht für Ausführung

def NoInternet():
    '''Warnmeldung wenn kein Internet vorhanden'''
    msg = QMessageBox()
```

```
    msg.setIcon(QMessageBox.Warning)
    msg.setText("Wetterdaten konnten nicht abgerufen werden.")
    msg.setInformativeText("Bitte Internetverbindung überprüfen!")
    msg.setStandardButtons(QMessageBox.Ok)
    msg.setWindowFlags(Qt.FramelessWindowHint) #Titlebar verbergen
    retval = msg.exec_() #Pflicht für Ausführung

if __name__ == '__main__':
   Sample()
```

```
import spidev
import time
import os
import MySQLdb

global R1
global R2
#------------Settings------------#
R1 = 22000
R2 = 4700
VoltMax = 13.1  #Max Akkuspannung
VoltMin = 11.8  #Min Zul. Akkuspannung
channel1 = 0    #PV_Spannung
channel2 = 1    #PV_Ampere
channel3 = 2    #Verb_Spannung
channel4 = 3    #Verb_Ampere
channel5 = 4    #Akku_Spannung
#--------------------------------#

"""SPI Bus initialisieren"""
spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 1000000

def ReadChannel(channel):
    """Werte der Analog Channels auslesen"""
    adc = spi.xfer2([1, (8+channel)<<4, 0])
    data = ((adc[1]&3) << 8) + adc[2]
    return data

def ConvertVolt(channel, places):
    """Analogwerte in Spannungs umwandeln"""
    SUMvolt = 0
    SUMdata = 0
    for i in range(0, 100):
        data = ReadChannel(channel)
        voltOffset =  0.028
        volt = (data * 3.3) / float(1023)+voltOffset
        volt = (volt * (R1+R2)) / R2
        SUMvolt = SUMvolt + volt
        SUMdata = SUMdata + data
    volt = SUMvolt / 100
    data = SUMdata / 100
    volt = round(volt, places)
    return (data, volt)

def ConvertAmpere(channel, places):
```

```
    '''Analogwerte in Ampere umwandeln'''
    SUMamp = 0
    SUMdata = 0
    for i in range(0, 100):
        data = ReadChannel(channel)
        ampOffset = 0.016 #Offset abweichung
        mVproAmp = 0.064  #Ermittelte Volt pro Ampere
        ampere = (data * 3.3) / float(1023)+ampOffset
        #ampere = (ampere-1.65) / mVproAmp
        SUMamp = SUMamp + ampere
        SUMdata = SUMdata + data
    ampere = SUMamp / 100
    data = SUMdata / 100
    #ampere = round(ampere, places)
    return (data, ampere)

def UpdateSensors():
    '''Aktuellen Sensorwerte abrufen'''
    voltData, PV_Volt = ConvertVolt(channel1, 2)
    ampereData1, PV_Ampere = ConvertAmpere(channel2, 3)
    voltData2, Verb_Volt = ConvertVolt(channel3, 2)
    ampereData2, Verb_Ampere = ConvertAmpere(channel4, 3)
    voltData3, AkkuVolt = ConvertVolt(channel5, 2)

    Erz_Leistung = PV_Volt * PV_Ampere
    Verb_Leistung = Verb_Volt * Verb_Ampere

    '''Spannung der Batterie in Prozent umwandeln'''
    deltaVolt = (VoltMax - VoltMin)
    AkkuProzent = ((AkkuVolt - VoltMin) / deltaVolt) * 100
    AkkuProzent = round(AkkuProzent, 2)
    if (AkkuVolt <= VoltMin):
        AkkuProzent = 0
    if (AkkuVolt >= VoltMax):
        AkkuProzent = 100
##
##    print("-----------------------------------------")
##    print("PV Spannung: {} ({}V)".format(voltData, PV_Volt))
##    print("PV Strom: {} ({}A)".format(ampereData1, PV_Ampere))
##    print("Verb Spannung: {} ({}V)".format(voltData2, Verb_Volt))
##    print("Verb Strom: {} ({}A)".format(ampereData2, Verb_Ampere))
##    print("Akku Spannung: {} ({}V)".format(voltData3, AkkuVolt))
##    print("Akku: {}%".format(AkkuProzent))

    SensorData = [PV_Volt, PV_Ampere, Verb_Volt, Verb_Ampere, AkkuVolt,
    AkkuProzent, Erz_Leistung, Verb_Leistung]
    return SensorData


# ADW (mcp3008) auslesen mit RPi
# https://www.raspberrypi-spy.co.uk/2013/10/analogue-sensors-on-the-raspberry-pi-using-an-mcp3008

if __name__ == '__main__':
    UpdateSensors()
```

```
# -*- coding: utf-8 -*-
```

```
# Form implementation generated from reading ui file 'settings.ui'
#
# Created by: PyQt5 UI code generator 5.5.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Settings(object):
    def setupUi(self, Settings):
        Settings.setObjectName("Settings")
        Settings.resize(761, 371)
        Settings.setAutoFillBackground(False)
        self.wind = QtWidgets.QLineEdit(Settings)
        self.wind.setGeometry(QtCore.QRect(190, 116, 91, 31))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.wind.setFont(font)
        self.wind.setAlignment(QtCore.Qt.AlignRight | QtCore.Qt.AlignTrailing | QtCore.Qt.AlignVCenter)
        self.wind.setObjectName("wind")
        self.label_3 = QtWidgets.QLabel(Settings)
        self.label_3.setGeometry(QtCore.QRect(90, 110, 81, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label_3.setFont(font)
        self.label_3.setObjectName("label_3")
        self.b_shutdown = QtWidgets.QPushButton(Settings)
        self.b_shutdown.setGeometry(QtCore.QRect(40, 310, 221, 51))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.b_shutdown.setFont(font)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap(":/system/img/system/if_Turn_off_105237.png"),
QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.b_shutdown.setIcon(icon)
        self.b_shutdown.setIconSize(QtCore.QSize(40, 40))
        self.b_shutdown.setObjectName("b_shutdown")
        self.label_4 = QtWidgets.QLabel(Settings)
        self.label_4.setGeometry(QtCore.QRect(290, 110, 81, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label_4.setFont(font)
        self.label_4.setObjectName("label_4")
        self.b_displayoff = QtWidgets.QPushButton(Settings)
        self.b_displayoff.setGeometry(QtCore.QRect(540, 310, 181, 51))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.b_displayoff.setFont(font)
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap(":/system/img/system/display.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        self.b_displayoff.setIcon(icon1)
        self.b_displayoff.setIconSize(QtCore.QSize(40, 40))
        self.b_displayoff.setObjectName("b_displayoff")
        self.icon_wind = QtWidgets.QLabel(Settings)
        self.icon_wind.setGeometry(QtCore.QRect(30, 110, 41, 51))
```

```
        self.icon_wind.setText("")
        self.icon_wind.setPixmap(QtGui.QPixmap(":/system/img/system/if_wind-turbine-01_2140005.png"))
        self.icon_wind.setScaledContents(True)
        self.icon_wind.setObjectName("icon_wind")
        self.icon_panels = QtWidgets.QLabel(Settings)
        self.icon_panels.setGeometry(QtCore.QRect(30, 60, 41, 41))
        self.icon_panels.setText("")
        self.icon_panels.setPixmap(QtGui.QPixmap(":/system/img/system/if_solar_64682.png"))
        self.icon_panels.setScaledContents(True)
        self.icon_panels.setObjectName("icon_panels")
        self.panels = QtWidgets.QLineEdit(Settings)
        self.panels.setGeometry(QtCore.QRect(190, 66, 91, 31))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.panels.setFont(font)
        self.panels.setAlignment(QtCore.Qt.AlignRight | QtCore.Qt.AlignTrailing | QtCore.Qt.AlignVCenter)
        self.panels.setObjectName("panels")
        self.label_2 = QtWidgets.QLabel(Settings)
        self.label_2.setGeometry(QtCore.QRect(290, 60, 81, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
        self.label = QtWidgets.QLabel(Settings)
        self.label.setGeometry(QtCore.QRect(90, 60, 81, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.b_energysafer = QtWidgets.QPushButton(Settings)
        self.b_energysafer.setGeometry(QtCore.QRect(490, 50, 191, 41))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.b_energysafer.setFont(font)
        self.b_energysafer.setAutoFillBackground(False)
        self.b_energysafer.setStyleSheet("color: rgb(199, 0, 0);")
        self.b_energysafer.setIconSize(QtCore.QSize(40, 40))
        self.b_energysafer.setCheckable(True)
        self.b_energysafer.setObjectName("b_energysafer")
        self.label_5 = QtWidgets.QLabel(Settings)
        self.label_5.setGeometry(QtCore.QRect(490, 10, 201, 31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label_5.setFont(font)
        self.label_5.setObjectName("label_5")
        self.b_reboot = QtWidgets.QPushButton(Settings)
        self.b_reboot.setGeometry(QtCore.QRect(310, 310, 181, 51))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.b_reboot.setFont(font)
        icon2 = QtGui.QIcon()
        icon2.addPixmap(QtGui.QPixmap(":/system/img/system/if_gnome-session-reboot_28664.png"),
QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.b_reboot.setIcon(icon2)
        self.b_reboot.setIconSize(QtCore.QSize(40, 40))
        self.b_reboot.setObjectName("b_reboot")
```

```
        self.label_6 = QtWidgets.QLabel(Settings)
        self.label_6.setGeometry(QtCore.QRect(90, 10, 91, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label_6.setFont(font)
        self.label_6.setObjectName("label_6")
        self.Battery = QtWidgets.QLineEdit(Settings)
        self.Battery.setGeometry(QtCore.QRect(190, 16, 91, 31))
        font = QtGui.QFont()
        font.setPointSize(14)
        self.Battery.setFont(font)
        self.Battery.setAlignment(QtCore.Qt.AlignRight | QtCore.Qt.AlignTrailing | QtCore.Qt.AlignVCenter)
        self.Battery.setObjectName("Battery")
        self.label_7 = QtWidgets.QLabel(Settings)
        self.label_7.setGeometry(QtCore.QRect(290, 10, 81, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label_7.setFont(font)
        self.label_7.setObjectName("label_7")
        self.icon_battery = QtWidgets.QLabel(Settings)
        self.icon_battery.setGeometry(QtCore.QRect(30, 10, 41, 41))
        self.icon_battery.setText("")
        self.icon_battery.setPixmap(QtGui.QPixmap(":/system/img/system/blue_energy.svg"))
        self.icon_battery.setScaledContents(True)
        self.icon_battery.setObjectName("icon_battery")

        self.retranslateUi(Settings)
        QtCore.QMetaObject.connectSlotsByName(Settings)

    def retranslateUi(self, Settings):
        _translate = QtCore.QCoreApplication.translate
        Settings.setWindowTitle(_translate("Settings", "Form"))
        self.wind.setText(_translate("Settings", "120"))
        self.label_3.setText(_translate("Settings", "Wind:"))
        self.b_shutdown.setText(_translate("Settings", "Herunterfahren"))
        self.label_4.setText(_translate("Settings", "Watt"))
        self.b_displayoff.setText(_translate("Settings", "Display aus"))
        self.panels.setText(_translate("Settings", "10"))
        self.label_2.setText(_translate("Settings", "Watt"))
        self.label.setText(_translate("Settings", "Panels:"))
        self.b_energysafer.setText(_translate("Settings", "Deaktivieren"))
        self.label_5.setText(_translate("Settings", "Energiesparmodus"))
        self.b_reboot.setText(_translate("Settings", "Neustart"))
        self.label_6.setText(_translate("Settings", "Batterie:"))
        self.Battery.setText(_translate("Settings", "7"))
        self.label_7.setText(_translate("Settings", "Ah"))

import resources_rc
```

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'weather.ui'
#
# Created by: PyQt5 UI code generator 5.5.1
#
# WARNING! All changes made in this file will be lost!
```

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Weather(object):
    def setupUi(self, Weather):
        Weather.setObjectName("Weather")
        Weather.resize(761, 371)
        self.clouds = QtWidgets.QLabel(Weather)
        self.clouds.setGeometry(QtCore.QRect(0, 40, 141, 101))
        self.clouds.setText("")
        self.clouds.setPixmap(QtGui.QPixmap(":/weather/img/weather/na.png"))
        self.clouds.setScaledContents(True)
        self.clouds.setObjectName("clouds")
        self.line = QtWidgets.QFrame(Weather)
        self.line.setGeometry(QtCore.QRect(280, 10, 20, 341))
        self.line.setFrameShape(QtWidgets.QFrame.VLine)
        self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line.setObjectName("line")
        self.wind = QtWidgets.QLabel(Weather)
        self.wind.setGeometry(QtCore.QRect(110, 180, 91, 31))
        self.wind.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.wind.setObjectName("wind")
        self.temp = QtWidgets.QLabel(Weather)
        self.temp.setGeometry(QtCore.QRect(110, 140, 61, 31))
        self.temp.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.temp.setObjectName("temp")
        self.sunrise = QtWidgets.QLabel(Weather)
        self.sunrise.setGeometry(QtCore.QRect(190, 60, 91, 31))
        self.sunrise.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.sunrise.setObjectName("sunrise")
        self.sunset = QtWidgets.QLabel(Weather)
        self.sunset.setGeometry(QtCore.QRect(190, 90, 91, 31))
        self.sunset.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.sunset.setObjectName("sunset")
        self.icon_sunrise = QtWidgets.QLabel(Weather)
        self.icon_sunrise.setGeometry(QtCore.QRect(150, 60, 31, 31))
        self.icon_sunrise.setText("")
        self.icon_sunrise.setPixmap(QtGui.QPixmap(":/weather/img/weather/sunrise.png"))
        self.icon_sunrise.setScaledContents(True)
        self.icon_sunrise.setObjectName("icon_sunrise")
        self.icon_sunset = QtWidgets.QLabel(Weather)
        self.icon_sunset.setGeometry(QtCore.QRect(150, 90, 31, 31))
        self.icon_sunset.setText("")
        self.icon_sunset.setPixmap(QtGui.QPixmap(":/weather/img/weather/sunset.png"))
        self.icon_sunset.setScaledContents(True)
        self.icon_sunset.setObjectName("icon_sunset")
        self.icon_pressure = QtWidgets.QLabel(Weather)
        self.icon_pressure.setGeometry(QtCore.QRect(70, 220, 31, 31))
        self.icon_pressure.setText("")
        self.icon_pressure.setPixmap(QtGui.QPixmap(":/weather/img/weather/pressure.png"))
        self.icon_pressure.setScaledContents(True)
        self.icon_pressure.setObjectName("icon_pressure")
        self.icon_temperatur = QtWidgets.QLabel(Weather)
        self.icon_temperatur.setGeometry(QtCore.QRect(70, 140, 31, 31))
        self.icon_temperatur.setText("")
        self.icon_temperatur.setPixmap(QtGui.QPixmap(":/weather/img/weather/thermometer-2.png"))
        self.icon_temperatur.setScaledContents(True)
```

```
self.icon_temperatur.setObjectName("icon_temperatur")
self.icon_temperatur_2 = QtWidgets.QLabel(Weather)
self.icon_temperatur_2.setGeometry(QtCore.QRect(70, 180, 31, 31))
self.icon_temperatur_2.setFrameShape(QtWidgets.QFrame.NoFrame)
self.icon_temperatur_2.setText("")
self.icon_temperatur_2.setPixmap(QtGui.QPixmap(":/weather/img/weather/wind-2.png"))
self.icon_temperatur_2.setScaledContents(True)
self.icon_temperatur_2.setObjectName("icon_temperatur_2")
self.clouds_2 = QtWidgets.QLabel(Weather)
self.clouds_2.setGeometry(QtCore.QRect(340, 40, 131, 91))
self.clouds_2.setText("")
self.clouds_2.setPixmap(QtGui.QPixmap(":/weather/img/weather/na.png"))
self.clouds_2.setScaledContents(True)
self.clouds_2.setObjectName("clouds_2")
self.icon_temperatur_3 = QtWidgets.QLabel(Weather)
self.icon_temperatur_3.setGeometry(QtCore.QRect(340, 140, 31, 31))
self.icon_temperatur_3.setText("")
self.icon_temperatur_3.setPixmap(QtGui.QPixmap(":/weather/img/weather/thermometer-2.png"))
self.icon_temperatur_3.setScaledContents(True)
self.icon_temperatur_3.setObjectName("icon_temperatur_3")
self.icon_pressure_2 = QtWidgets.QLabel(Weather)
self.icon_pressure_2.setGeometry(QtCore.QRect(340, 220, 31, 31))
self.icon_pressure_2.setText("")
self.icon_pressure_2.setPixmap(QtGui.QPixmap(":/weather/img/weather/pressure.png"))
self.icon_pressure_2.setScaledContents(True)
self.icon_pressure_2.setObjectName("icon_pressure_2")
self.temp_2 = QtWidgets.QLabel(Weather)
self.temp_2.setGeometry(QtCore.QRect(380, 140, 61, 31))
self.temp_2.setStyleSheet("font: 75 14pt \"Noto Sans\";")
self.temp_2.setObjectName("temp_2")
self.wind_2 = QtWidgets.QLabel(Weather)
self.wind_2.setGeometry(QtCore.QRect(380, 180, 131, 31))
self.wind_2.setStyleSheet("font: 75 14pt \"Noto Sans\";")
self.wind_2.setObjectName("wind_2")
self.icon_temperatur_4 = QtWidgets.QLabel(Weather)
self.icon_temperatur_4.setGeometry(QtCore.QRect(340, 180, 31, 31))
self.icon_temperatur_4.setFrameShape(QtWidgets.QFrame.NoFrame)
self.icon_temperatur_4.setText("")
self.icon_temperatur_4.setPixmap(QtGui.QPixmap(":/weather/img/weather/wind-2.png"))
self.icon_temperatur_4.setScaledContents(True)
self.icon_temperatur_4.setObjectName("icon_temperatur_4")
self.clouds_3 = QtWidgets.QLabel(Weather)
self.clouds_3.setGeometry(QtCore.QRect(580, 40, 131, 91))
self.clouds_3.setText("")
self.clouds_3.setPixmap(QtGui.QPixmap(":/weather/img/weather/na.png"))
self.clouds_3.setScaledContents(True)
self.clouds_3.setObjectName("clouds_3")
self.icon_temperatur_5 = QtWidgets.QLabel(Weather)
self.icon_temperatur_5.setGeometry(QtCore.QRect(590, 140, 31, 31))
self.icon_temperatur_5.setText("")
self.icon_temperatur_5.setPixmap(QtGui.QPixmap(":/weather/img/weather/thermometer-2.png"))
self.icon_temperatur_5.setScaledContents(True)
self.icon_temperatur_5.setObjectName("icon_temperatur_5")
self.icon_pressure_3 = QtWidgets.QLabel(Weather)
self.icon_pressure_3.setGeometry(QtCore.QRect(590, 220, 31, 31))
self.icon_pressure_3.setText("")
```

```
        self.icon_pressure_3.setPixmap(QtGui.QPixmap(":/weather/img/weather/pressure.png"))
        self.icon_pressure_3.setScaledContents(True)
        self.icon_pressure_3.setObjectName("icon_pressure_3")
        self.temp_3 = QtWidgets.QLabel(Weather)
        self.temp_3.setGeometry(QtCore.QRect(630, 140, 61, 31))
        self.temp_3.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.temp_3.setObjectName("temp_3")
        self.wind_3 = QtWidgets.QLabel(Weather)
        self.wind_3.setGeometry(QtCore.QRect(630, 180, 131, 31))
        self.wind_3.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.wind_3.setObjectName("wind_3")
        self.icon_temperatur_6 = QtWidgets.QLabel(Weather)
        self.icon_temperatur_6.setGeometry(QtCore.QRect(590, 180, 31, 31))
        self.icon_temperatur_6.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.icon_temperatur_6.setText("")
        self.icon_temperatur_6.setPixmap(QtGui.QPixmap(":/weather/img/weather/wind-2.png"))
        self.icon_temperatur_6.setScaledContents(True)
        self.icon_temperatur_6.setObjectName("icon_temperatur_6")
        self.prog_erz_2 = QtWidgets.QLabel(Weather)
        self.prog_erz_2.setGeometry(QtCore.QRect(380, 270, 131, 31))
        self.prog_erz_2.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.prog_erz_2.setObjectName("prog_erz_2")
        self.icon_erzeugung2 = QtWidgets.QLabel(Weather)
        self.icon_erzeugung2.setGeometry(QtCore.QRect(340, 270, 31, 31))
        self.icon_erzeugung2.setText("")

self.icon_erzeugung2.setPixmap(QtGui.QPixmap(":/system/img/system/if_klaptopdaemon_18023.png"))
        self.icon_erzeugung2.setScaledContents(True)
        self.icon_erzeugung2.setObjectName("icon_erzeugung2")
        self.line_2 = QtWidgets.QFrame(Weather)
        self.line_2.setGeometry(QtCore.QRect(520, 10, 20, 351))
        self.line_2.setFrameShape(QtWidgets.QFrame.VLine)
        self.line_2.setFrameShadow(QtWidgets.QFrame.Sunken)
        self.line_2.setObjectName("line_2")
        self.icon_erzeugung3 = QtWidgets.QLabel(Weather)
        self.icon_erzeugung3.setGeometry(QtCore.QRect(590, 270, 31, 31))
        self.icon_erzeugung3.setText("")

self.icon_erzeugung3.setPixmap(QtGui.QPixmap(":/system/img/system/if_klaptopdaemon_18023.png"))
        self.icon_erzeugung3.setScaledContents(True)
        self.icon_erzeugung3.setObjectName("icon_erzeugung3")
        self.prog_erz_3 = QtWidgets.QLabel(Weather)
        self.prog_erz_3.setGeometry(QtCore.QRect(630, 270, 131, 31))
        self.prog_erz_3.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.prog_erz_3.setObjectName("prog_erz_3")
        self.icon_erzeugung1 = QtWidgets.QLabel(Weather)
        self.icon_erzeugung1.setGeometry(QtCore.QRect(70, 270, 31, 31))
        self.icon_erzeugung1.setText("")

self.icon_erzeugung1.setPixmap(QtGui.QPixmap(":/system/img/system/if_klaptopdaemon_18023.png"))
        self.icon_erzeugung1.setScaledContents(True)
        self.icon_erzeugung1.setObjectName("icon_erzeugung1")
        self.prog_erz = QtWidgets.QLabel(Weather)
        self.prog_erz.setGeometry(QtCore.QRect(110, 270, 131, 31))
        self.prog_erz.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.prog_erz.setObjectName("prog_erz")
```

```
        self.b_today = QtWidgets.QPushButton(Weather)
        self.b_today.setGeometry(QtCore.QRect(20, 10, 251, 31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.b_today.setFont(font)
        self.b_today.setObjectName("b_today")
        self.b_today_2 = QtWidgets.QPushButton(Weather)
        self.b_today_2.setGeometry(QtCore.QRect(310, 10, 201, 31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.b_today_2.setFont(font)
        self.b_today_2.setObjectName("b_today_2")
        self.b_today_3 = QtWidgets.QPushButton(Weather)
        self.b_today_3.setGeometry(QtCore.QRect(550, 10, 201, 31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.b_today_3.setFont(font)
        self.b_today_3.setObjectName("b_today_3")
        self.temp_minmax = QtWidgets.QLabel(Weather)
        self.temp_minmax.setGeometry(QtCore.QRect(170, 146, 81, 21))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.temp_minmax.setFont(font)
        self.temp_minmax.setObjectName("temp_minmax")
        self.b_location = QtWidgets.QPushButton(Weather)
        self.b_location.setGeometry(QtCore.QRect(10, 340, 161, 27))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.b_location.setFont(font)
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap(":/weather/img/weather/location.png"), QtGui.QIcon.Normal,
QtGui.QIcon.Off)
        self.b_location.setIcon(icon)
        self.b_location.setObjectName("b_location")
        self.b_refresh_status = QtWidgets.QPushButton(Weather)
        self.b_refresh_status.setGeometry(QtCore.QRect(600, 340, 151, 27))
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap(":/system/img/system/if_info_blue_40801.png"),
QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.b_refresh_status.setIcon(icon1)
        self.b_refresh_status.setIconSize(QtCore.QSize(21, 21))
        self.b_refresh_status.setObjectName("b_refresh_status")
        self.temp_minmax_2 = QtWidgets.QLabel(Weather)
        self.temp_minmax_2.setGeometry(QtCore.QRect(440, 140, 81, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.temp_minmax_2.setFont(font)
        self.temp_minmax_2.setObjectName("temp_minmax_2")
        self.temp_minmax_3 = QtWidgets.QLabel(Weather)
        self.temp_minmax_3.setGeometry(QtCore.QRect(690, 140, 71, 31))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.temp_minmax_3.setFont(font)
        self.temp_minmax_3.setObjectName("temp_minmax_3")
        self.humidity = QtWidgets.QLabel(Weather)
        self.humidity.setGeometry(QtCore.QRect(110, 220, 91, 31))
```

```
        self.humidity.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.humidity.setObjectName("humidity")
        self.humidity_2 = QtWidgets.QLabel(Weather)
        self.humidity_2.setGeometry(QtCore.QRect(380, 220, 91, 31))
        self.humidity_2.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.humidity_2.setObjectName("humidity_2")
        self.humidity_3 = QtWidgets.QLabel(Weather)
        self.humidity_3.setGeometry(QtCore.QRect(630, 220, 91, 31))
        self.humidity_3.setStyleSheet("font: 75 14pt \"Noto Sans\";")
        self.humidity_3.setObjectName("humidity_3")
        self.clouds.raise_()
        self.line.raise_()
        self.wind.raise_()
        self.temp.raise_()
        self.sunrise.raise_()
        self.sunset.raise_()
        self.icon_sunrise.raise_()
        self.icon_sunset.raise_()
        self.icon_pressure.raise_()
        self.icon_temperatur_2.raise_()
        self.icon_temperatur.raise_()
        self.clouds_2.raise_()
        self.icon_temperatur_3.raise_()
        self.icon_pressure_2.raise_()
        self.temp_2.raise_()
        self.wind_2.raise_()
        self.icon_temperatur_4.raise_()
        self.clouds_3.raise_()
        self.icon_temperatur_5.raise_()
        self.icon_pressure_3.raise_()
        self.temp_3.raise_()
        self.wind_3.raise_()
        self.icon_temperatur_6.raise_()
        self.prog_erz_2.raise_()
        self.icon_erzeugung2.raise_()
        self.line_2.raise_()
        self.icon_erzeugung3.raise_()
        self.prog_erz_3.raise_()
        self.icon_erzeugung1.raise_()
        self.prog_erz.raise_()
        self.b_today.raise_()
        self.b_today_2.raise_()
        self.b_today_3.raise_()
        self.temp_minmax.raise_()
        self.b_location.raise_()
        self.b_refresh_status.raise_()
        self.temp_minmax_2.raise_()
        self.temp_minmax_3.raise_()
        self.humidity.raise_()
        self.humidity_2.raise_()
        self.humidity_3.raise_()

        self.retranslateUi(Weather)
        QtCore.QMetaObject.connectSlotsByName(Weather)

    def retranslateUi(self, Weather):
```

```
        _translate = QtCore.QCoreApplication.translate
        Weather.setWindowTitle(_translate("Weather", "Form"))
        self.wind.setText(_translate("Weather", "N/A"))
        self.temp.setText(_translate("Weather", "N/A"))
        self.sunrise.setText(_translate("Weather", "N/A"))
        self.sunset.setText(_translate("Weather", "N/A"))
        self.temp_2.setText(_translate("Weather", "N/A"))
        self.wind_2.setText(_translate("Weather", "N/A"))
        self.temp_3.setText(_translate("Weather", "N/A"))
        self.wind_3.setText(_translate("Weather", "N/A"))
        self.prog_erz_2.setText(_translate("Weather", "0"))
        self.prog_erz_3.setText(_translate("Weather", "0"))
        self.prog_erz.setText(_translate("Weather", "0" ))
        self.b_today.setText(_translate("Weather", "Heute"))
        self.b_today_2.setText(_translate("Weather", "Morgen"))
        self.b_today_3.setText(_translate("Weather", "Übermorgen"))
        self.temp_minmax.setText(_translate("Weather", ""))
        self.b_location.setText(_translate("Weather", "N/A"))
        self.b_refresh_status.setText(_translate("Weather", "Nicht aktuell"))
        self.temp_minmax_2.setText(_translate("Weather", ""))
        self.temp_minmax_3.setText(_translate("Weather", ""))
        self.humidity.setText(_translate("Weather", "N/A"))
        self.humidity_2.setText(_translate("Weather", "N/A"))
        self.humidity_3.setText(_translate("Weather", "N/A"))

import resources_rc
```

9 Selbständigkeitserklärung

Wir erklären hiermit, dass wir diese Projektarbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt haben. Alle Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, haben wir als solche gekennzeichnet.

Einer Veröffentlichung/Verwertung durch die Staatliche Technikerschule Berlin für den regulären Betrieb der Schule, insbesondere für den Lehrbetrieb, stimme ich zu.

Berlin, 28. 05. 2018 .....
Studierende(r)

Berlin, 28. 05. 2018 .....
Studierende(r)

Selbständigkeitserklärung

Wir erklären hiermit, dass wir diese Projektarbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt haben. Alle Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, haben wir als solche gekennzeichnet.

Einer Veröffentlichung/Verwertung durch die Staatliche Technikerschule Berlin für den regulären Betrieb der Schule, insbesondere für den Lehrbetrieb, stimmen wir zu.

Berlin, 28. 05. 2018 .....
Studierende(r)

Berlin, 28. 05. 2018 .....
Studierende(r)