

**Fachrichtung Informatik**

**Schuljahr 2017/2018**

# **DIPLOMARBEIT**

**Gesamtprojekt**

# **AEMS – Advanced Energy Monitoring System**

**Ausgeführt von:**

Niklas Graf, 5BHIF-2

Lukas Knoll, 5BHIF-8

Sebastian Mandl, 5BHIF-11

**Betreuer:**

DI Josef Doppelbauer

Grieskirchen, am 04.04.2018

---

**Abgabevermerk:**

Datum: 04.04.2018

Betreuer: DI Josef Doppelbauer

## **Erklärung gemäß Prüfungsordnung**

„Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und alle den benutzten Quellen wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.“

Grieskirchen, 04.04.2018

Verfasser/innen:

Niklas Graf

Lukas Knoll

Sebastian Mandl

## **DIPLOMARBEIT DOKUMENTATION**

<b>Namen der Verfasser/innen</b>	Niklas Graf Lukas Knoll Sebastian Mandl
<b>Jahrgang Schuljahr</b>	2017/18
<b>Thema der Diplomarbeit</b>	AEMS – Advanced Energy Monitoring System
<b>Kooperationspartner</b>	Energiegenossenschaft Region Eferding

<b>Aufgabenstellung</b>	<p>Entwicklung eines Bots, welcher die AMIS-Zählerdaten ausliest und diese in einer Datenbank speichert.</p> <p>Eigene Zähler für z.B. Strom und Wärmemenge auf Raspberry Pis aufsetzen.</p> <p>Alle gesammelten Daten werden in Form von Statistiken grafisch aufbereitet.</p> <p>Weiters sollen in festgelegten Zeitabständen Berichte über die Verbrauchswerte generiert werden.</p> <p>Downloadfunktion für Statistiken und Berichte.</p> <p>Entwicklung einer Anomalierkennung bei Abweichung der Verbrauchswerte.</p> <p>Benachrichtigungssystem um bei Verbrauchsabweichungen benachrichtigt zu werden.</p> <p>Entwicklung einer Android-App, um sich am Smartphone Statistiken anzusehen, die Möglichkeit zu haben diese herunterzuladen und bei Verbrauchsabweichungen eine Benachrichtigung zu erhalten.</p> <p>Administrationstool für die Verwaltung der Zugriffsrechte.</p>
-------------------------	--

<b>Realisierung</b>	<p>Entwicklung der Android-App mithilfe von Androidstudio V2.3. Für die Statistiken wird die Library MPAndroidCharts verwendet. Zur Kommunikation mit dem Server werden die eigens entwickelte Library aems-api-lib und der Java REST Dienst verwendet. Alle weiteren Funktionen der Android-App wurden selbst entwickelt.</p> <p>Erstellung des Webinterfaces unter Zuhilfenahme der CSS-Library Bootstrap und Erstellung der Modals mithilfe von clean-modal-login-form. Für das Webinterface wurde die Sprache xhtml verwendet und in Kombination damit Java Server Faces. Für weitere Funktionalität wurden jQuery und Javascript verwendet. Für das Anzeigen der Statistiken wurde die Chart-Library Chart.js genutzt und der Download als PDF wurde mit der Library jsPDF ermöglicht.</p> <p>Das Webinterface bietet die Möglichkeit sich auf der Startseite seine gewünschten Statistiken anzeigen zu lassen, sich Schnellauswertungen anlegen und herunterladen zu können, Statistiken anzulegen, Statistiken für die Startseite und Android-App zu konfigurieren, Berichte für gewünschte Perioden einzurichten und sich Richtlinien für Benachrichtigungen zu erstellen.</p>
---------------------	--

	<b>HTBLA Grieskirchen</b>	
Fachrichtung:	Informatik	

	<p>Das Administrationstool wurde auch als xhtml-Webinterface mit Hilfe von Bootstrap, clean-modal-login-form, jQuery, Javascript und JSF erstellt. Hier gibt es die Möglichkeit die Freigaben für das System zu betreuen, Administratoren zu verwalten und sich seine Zuständigkeitsbereiche als Administrator zu definieren.</p> <p>PostgreSQL-Datenbank für die Speicherung der Daten.</p> <p>Der Report-Bot, welcher die Daten aus dem netzonline Portal herunterlädt und in die Datenbank speichert, wurde mit Java realisiert. Dazu wurde die Klassenbibliothek „HTMLUnit“ verwendet, um die Interaktion mit dem netzonline Portal zu ermöglichen, da es dafür keine API oder ähnliches gibt. Des Weiteren wurde die Bibliothek „Apache POI“ verwendet, um die benötigten Daten aus den heruntergeladenen MS-Excel Dateien zu extrahieren.</p> <p>Über eine Java REST Schnittstelle wurde die Kommunikation zwischen der Datenbank und den einzelnen Clients realisiert. Diese Schnittstelle bildet eine Zwischenschicht, welche ein immer gleichbleibendes Kommunikationsprotokoll garantiert.</p> <p>Mit Hilfe eines Raspberry Pis werden verschiedene Typen von Zählern ausgelesen, diese Daten werden dann über eine REST-Schnittstelle an die Datenbank gesendet und dort gespeichert. Das Erfassen der Zählerdaten wird über ein Plugin-System bewältigt, welches dynamisch Plugins in den Verarbeitungsprozess miteinbindet.</p> <p>In Verbindung mit Home-Automation wurde eine Schnittstelle zum OpenHAB errichtet. Über diese Schnittstelle ist es möglich die Daten der Zähler, also z.B.: den aktuellen Verbrauch, einzusehen.</p>
--	---

<b>Ergebnisse</b>	<p>Funktionierender Bot, welcher die Daten ausliest und in die Datenbank einspeist.</p> <p>Funktionierende Client/Server Infrastruktur zwischen den verwendeten Diensten.</p> <p>Funktionierende Dienste Webinterface, Administration-Webinterface, Android-App, Raspberry PI Zähler, aems-apilib, Datenbank und Java REST Service.</p> <p>Jegliche Anforderungen des Auftraggebers wurden erfüllt.</p>
-------------------	---

	<b>HTBLA Grieskirchen</b>
Fachrichtung:	<b>Informatik</b>

<p><b>Typische Grafik, Foto etc. (mit Erläuterung)</b></p>  <p>The screenshot shows a weekly energy consumption bar chart titled "Wochenstatistik". The Y-axis represents energy usage in kWh, ranging from 0 to 290 in increments of 50. The X-axis lists the days of the week: Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, and Sonntag. For each day, there are two bars: a yellow bar for "Verbrauch" and an orange bar for "Verbrauch (Vorperiode)". The chart shows varying levels of energy usage throughout the week, with a notable peak on Friday.</p>	<p>Übersicht der Startseite. Oben befindet sich die Navigation-Bar mit den Menüs, dem Benutzer/der Benutzerin und dem Benachrichtigungssystem. Der Seiteninhalt selbst besteht aus ausgewählten Statistiken, welche für den Download bereitstehen.</p>  <p>The screenshot shows a dual-axis chart titled "Stromstatistik 2" displayed on a smartphone. The chart features three data series: "Aktueller Verbrauch" (blue bars), "Verbrauch Vorperiode" (green bars), and "Außentemperatur" (red line with dots). The X-axis represents time periods, and the Y-axis represents numerical values. An anomalous data point is highlighted with a red circle and labeled "Anomalie".</p>
--	---

	<b>HTBLA Grieskirchen</b>
Fachrichtung:	<b>Informatik</b>

	 <p>The screenshot shows the AEMS-Admin interface. At the top, there's a navigation bar with 'Administration', 'Zuständigkeitsbereich', a user dropdown ('admin'), and a notification bell icon. Below the navigation is a section titled 'Neue Anfragen (3)' which lists three new requests: 'Graf' (sent recently), 'Knoll' (sent 17 days ago), and 'Mandl' (sent over a month ago). Further down is a section titled 'AEMS - Benutzer' which shows a search bar and a list entry for 'Josef Doppelbauer' with the email 'betreuungslehrer@gmx.net'. At the bottom right of the interface, there are links for 'Impressum' and '© Ein Dienst der Energiegenossenschaft Eferding'.</p> <p>Übersicht der Administrationswebsite. In der Navigation-Bar sind die verfügbaren Menüs eingeblendet, der verwendete Nutzer/Nutzerin und das Benachrichtigungssystem. Im Seiteninhalt werden die neuen Anfragen und die bereits von einem Administrator verwalteten Benutzer angezeigt. Diese können durch eine Suchfunktion einfacher gefunden werden.</p>
--	---

<b>Teilnahme an Wettbewerben, Auszeichnungen</b>	Bosch – Technik fürs Leben Preis 2018 Weitblick – Champions Jugend Innovativ – Sonderpreis Sustainability  Da die Wettbewerbe erst nach Beendigung der Diplomarbeit ausgewertet werden, liegen noch keine Informationen zu Auszeichnungen vor.
--	--

<b>Möglichkeiten der Einsichtnahme in die Arbeit</b>	Der Quellcode dieser Diplomarbeit unterliegt auf Wunsch des Projektteams der Geheimhaltung und steht nur der Prüfungskommission zur Einsicht zur Verfügung. Eine Version der Diplomarbeit wird im Archiv verwahrt und ist für die öffentliche Einsichtnahme gesperrt.
--	---

<b>Approbation (Datum/Unterschrift)</b>	Prüfer/Prüferin	Direktor/Direktorin
---	-----------------	---------------------

	<b>HTBLA Grieskirchen</b>	
Department:	Informatik	

## DIPLOMA THESIS Documentation

<b>Author(s)</b>	Niklas Graf Lukas Knoll Sebastian Mandl
<b>Form</b> <b>Academic year</b>	2017/18
<b>Topic</b>	AEMS – Advanced Energy Monitoring System
<b>Co-operation partners</b>	Energiegenossenschaft Region Eferding

<b>Assignment of tasks</b>	Development of a bot which extracts data from AMIS-meters and stores it in the database of the AEMS-System.
	Define customized meters for instance electricity-meters, heat-meters and many more and set them up on your personal Raspberry-PI.
	The entirety of data will be processed and displayed as statistics. Furthermore, the generation of reports which specify time constraints are possible as well. The aforementioned functionalities also provide download capability.
	Additionally, the detection of anomalies is provided by an integrated scripting language called AEMS-scripting-language (AEMSSL). The main aim of the scripting language is the detection of deviation of consumption values either caused by meters themselves or sensors affecting the meters values (day-time, light-intensity).
	Moreover, an integrated notification system supports easy recognition of abnormal meter data.
	Development of an android-app to display priorly defined and generated statistics. The android-app also comes along with the download capability of statistics and notifies the user if abnormal meter data is noticed by the system.
	Administrative tool for the management of access rights to the system.

<b>Realisation</b>	Android app development was supported by the IDE Android Studio V.2.3. For the generation of statistics the MPAndroidCharts library was consulted. The communication between the server and the client is managed by a REST-API as well as by a self-created library AEMS-API-LIB. Every further functionality found in the android-app was entirely self-created.
	Creation of the web-interface was supported by several libraries namely CSS, Bootstrap. Additionally, modals were created by the help of clean-modal-

	<b>HTBLA Grieskirchen</b>	
Department:	Informatik	

	<p>login-form.</p> <p>The language in which the web-interface was designed was chosen to be XHTML and was utilized in conjunction with Java-Server-Faces (JSF). Moreover, jQuery and JavaScript were used to implement further functionality.</p> <p>The library which grants the display of statistics is named Chart.js And the one which provides the capabilities for downloading PDFs is named jsPDF.</p> <p>The web-interfaces supplies the user with the possibility to display the desired statistics which the user wants to have immediate access to as well as to define quick statistics. These statistics can furthermore be subject to download or configuration. Possible configuration would be to only let the statistic appear on the smartphone app or to declare time constraints on reports or statistics to only receive reviews of periods that the user demands. In Addition, guidelines for the detection of anomalies can be specified which will be taken under advisement by the unit which is accountable for processing the data received from the meters and sensors (AEMSSL).</p> <p>The administrative tool was also chosen to be designed with XHTML, jQuery, JavaScript, JSF and Bootstrap. This tool permits the administration of access rights hence the declaration of new administrators and moreover to specify the exact geographical realms where the administrators operate in.</p> <p>PostgreSQL-Database as a means of data storage.</p> <p>The report-bot is in charge of downloading the data from the NETZONLINE-page which then gets stowed in the database of the system. This program was entirely implemented with the JAVA programming language. In order to had implemented the communication with the NETZONLINE-page a special library named HTMLUnit was consulted. For the extraction of the data from the excel files which were downloaded by the report-bot another library named Apache POI was utilized.</p> <p>A Java REST-API enables the communication between the database and the several clients. The aforementioned REST-API provides a logical layer for all database requests performed via GraphQL to guarantee a never changing communication protocol.</p> <p>With the utilization of a Raspberry PI the data of various different meter types is able to be read from. After the read the data is transmitted to the database via the REST-API which then is responsible for creating the proper SQL statements to store the data properly in the database. The acquisition of meter data is performed by a Plugin-System which is capable of dynamically including user-defined Plugins into the processing environment.</p> <p>In conjunction with home automation an interface to Open-HAB was established. Via this interface it is possible for the user to retrieve the current meter value from the server and to display it.</p>
--	---

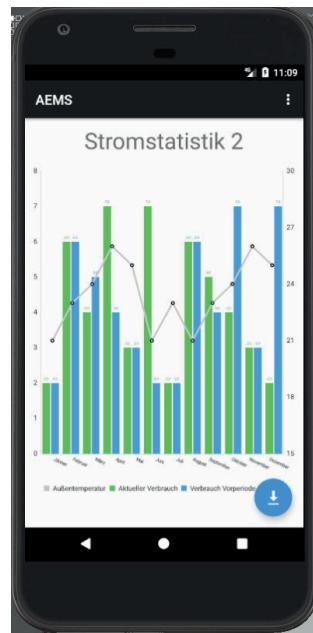
<b>Results</b>	<p>Functional report-bot, which retrieves data and stores it in the database.</p> <p>Functional client/server infrastructure between the utilized services.</p> <p>Functional services are web-interface, administrative tool, android-app, Raspberry PI meter data extraction, aems-apilib, database and Java REST-API.</p> <p>Every functionality demanded by the customer was satisfied.</p>
----------------	---

	<b>HTBLA Grieskirchen</b>
Department:	<b>Informatik</b>



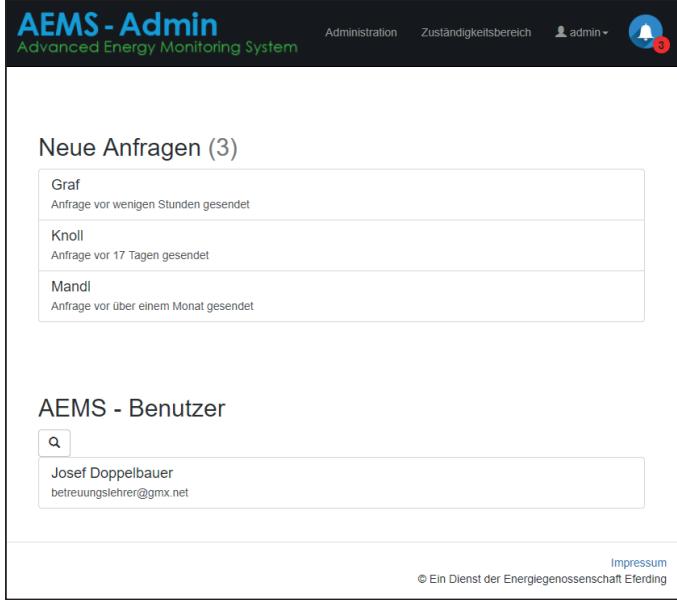
Dashboard. On the top one can find the navigation bar with the menus, the user and the notification system. The content of the page itself is comprised of user selected statistics which are available for download.

#### Illustrative graph, photo (incl. explanation)



View of a statistic in the android-app. On the top one can find the navigation bar which provides the functionalities of displaying the current notifications accessible by the user as well as layout changes. The view depicts a statistic with current consumption values and consumption values of the pre-period and the corresponding anomaly. Furthermore, there is the possibility to download the statistics.

	<b>HTBLA Grieskirchen</b>
Department:	<b>Informatik</b>

	 <p>The screenshot shows the AEMS-Admin interface. At the top, there's a navigation bar with links for Administration, Zuständigkeitsbereich, and a user dropdown. A notification bell icon indicates three unread notifications. Below the navigation, there are two main sections: "Neue Anfragen (3)" (New Requests) and "AEMS - Benutzer". The "Neue Anfragen" section lists three requests from users Graf, Knoll, and Mandl, each with a timestamp. The "AEMS - Benutzer" section shows a search input and a result for Josef Doppelbauer with his email address. At the bottom right of the interface, there's a link to the Impressum.</p> <p>The view of the administrative tool web-page. The navigation bar displays the menus accessible by the user as well as the currently logged-in user and the notification system. The content of this picture illustrates incoming request for usership. These have to accepted in order to be viewed as a user of the system. Previously added users can be easily found by utilizing the provided search functionality.</p>
--	--

<b>Participation in competitions</b> <b>Awards</b>	Bosch – Technik fürs Leben Preis 2018 Weitblick – Champions Jugend Innovativ – Sonderpreis Sustainability  Since the evaluation of the diploma is performed after the conclusion and submission of the project work, to competitions no information regarding awards is present.
---	--

<b>Accessibility of</b> <b>diploma thesis</b>	The source code of this diploma is subject to strict secrecy, at the wish of the project team, and is merely accessible by the commission responsible for the grading of this diploma. One version of this diploma will be stored in the archive and is subject to prohibition for public insight.
--	--

<b>Approval</b> <b>(date/signature)</b>	Examiner	Head of College/Department
--	----------	----------------------------

# Inhaltsverzeichnis

<b>1 Einleitung (Aufgabenstellung des Gesamtprojekts und Überblick über die Funktionen).....</b>	<b>1</b>
1.1 Aufgabenstellung .....	1
1.2 Überblick über die Funktionen .....	1
<b>2 Individuelle Zielsetzungen des Projektteams .....</b>	<b>3</b>
2.1 Niklas Graf.....	3
2.2 Lukas Knoll .....	3
2.3 Sebastian Mandl .....	3
<b>3 Umsetzung .....</b>	<b>4</b>
3.1 Umsetzung Web-Application .....	4
3.1.1 Frontend .....	4
3.1.2 Schnittstelle XHTML.....	12
3.1.3 Backend .....	13
3.2 Umsetzung Report-Bot.....	22
3.2.1 Allgemein.....	22
3.2.2 Umsetzung.....	22
3.2.3 Konfiguration.....	23
3.2.4 Logging .....	24
3.2.5 FileDownloader .....	24
3.2.6 ExcelDataExtractor.....	26
3.2.7 Installation.....	26
3.2.8 Abfragen der Wetterdaten .....	29
3.3 Umsetzung Android-App.....	30
3.3.1 Allgemein.....	30
3.3.2 Main Activity .....	31

3.3.3	Statistik Tab Activity.....	35
3.3.4	Login Activity .....	38
3.3.5	Menü .....	41
3.3.6	Background Task Notifications.....	42
3.4	Umsetzung WebUI .....	44
3.4.1	Umsetzung Scripting-Language-Interpreter .....	45
3.5	Umsetzung REST-Service .....	54
3.5.1	Authentifizierung.....	54
3.5.2	Abfrage von Daten.....	55
3.5.3	Authorization.....	56
3.5.4	Änderungsoperation via REST .....	56
3.6	Umsetzung Datenbank.....	58
3.6.1	Allgemein.....	58
3.6.2	ERD .....	59
3.7	Umsetzung Raspberry PI Konfigurationstool .....	60
3.8	Umsetzung openHAB .....	63
3.9	Weitere Dienste .....	65
3.9.1	aems-apilib .....	65
3.9.2	Diffie-Hellman Library.....	66
3.9.3	AEMS-Database Library .....	67
<b>4</b>	<b>Verwendete Libraries .....</b>	<b>69</b>
4.1	Externe Quellen .....	69
4.1.1	Apache POI .....	69
4.1.2	Apache HttpClient .....	69
4.1.3	Bootstrap.....	69
4.1.4	Bootsnipp-Modal.....	69
4.1.5	Firebase .....	69
4.1.6	Graphql-Java.....	69

4.1.7	Gson .....	70
4.1.8	HTMLUnit .....	70
4.1.9	html2Canvas.....	70
4.1.10	JSON .....	70
4.1.11	jsPDF.....	70
4.1.12	MPAndroidChart.....	70
4.1.13	Notify.js .....	70
4.1.14	Noty.....	71
4.1.15	PrettyFaces .....	71
4.1.16	Pyserial .....	71
4.2	Selbst entwickelt.....	71
4.2.1	aems-apilib .....	71
4.2.2	Diffie-Hellman Library.....	71
4.2.3	AEMS-Database Library .....	71
<b>5</b>	<b>Ergebnisse.....</b>	<b>72</b>
5.1	Erreichte Ziele .....	72
5.2	Auswirkung .....	72
5.3	Abschluss .....	72
5.4	Statement des Auftraggebers .....	73
<b>6</b>	<b>Danksagung .....</b>	<b>74</b>
<b>7</b>	<b>Anhang .....</b>	<b>82</b>
7.1	Projektdokumentation (Soll-Ist-Vergleich, Kostendarstellung) .....	82
7.2	Arbeitsbericht Niklas Graf - Ausschnitt .....	83
7.3	Arbeitsbericht Lukas Knoll - Ausschnitt.....	84
7.4	Arbeitsbericht Sebastian Mandl - Ausschnitt.....	85
7.5	Begleitprotokoll.....	86

7.6	Bildschirmmasken .....	91
7.6.1	AEMS .....	91
7.6.2	AEMS - Admin.....	93
7.6.3	App .....	94

# **1 Einleitung (Aufgabenstellung des Gesamtprojekts und Überblick über die Funktionen)**

## **1.1 Aufgabenstellung**

Die Aufgabe ist es, in öffentlichen Gebäuden einen einfacheren Überblick über die Verbrauchswerte zu liefern. Hierzu soll es die Möglichkeit geben die Verbrauchswerte der AMIS-Stromzähler auszulesen. Für eine gute Übersichtlichkeit sollen diese Verbrauchswerte als Statistiken und Berichte aufbereitet werden.

Um den Energieverbrauch besser steuern zu können soll ein Benachrichtigungssystem entwickelt werden, über welches man bei abweichendem Verbrauch im Vergleich zu den Vorperioden benachrichtigt wird.

Eine weitere Aufgabe ist die Erstellung einer mobilen Application für Android Smartphones. Hier soll es die Möglichkeit geben, sich vordefinierte Statistiken in der App anzeigen zu lassen und diese herunterladen zu können. Des Weiteren soll man auch am Smartphone bei abweichendem Verbrauch benachrichtigt werden.

Um nicht nur von den AMIS-Zählern abhängig zu sein, soll ein modulares System entwickelt werden, mit welchem man sich auf Raspberry PIs selbst eigene Zähler anlegen und konfigurieren kann. Als Beispiel Strom-, Wärmemenge-, Gaszähler.

Als Erweiterung wird eine Anomalierkennung umgesetzt. Diese Anomalierkennung können sich AEMS-Nutzer-, und Nutzerinnen selbst konfigurieren, um sich eigene Abhängigkeitsfaktoren wie z.B. die Außentemperatur einzustellen. Diese Anomalien können dazu verwendet werden das Benachrichtigungssystem intelligent auf Abhängigkeitsfaktoren reagieren zu lassen.

## **1.2 Überblick über die Funktionen**

AEMS bietet die Möglichkeit sich selbst Statistiken über selbst gewählte Zähler und Perioden erstellen zu lassen. Diese Statistiken werden in der Datenbank gespeichert und stehen jederzeit auf der Webpage zur Verfügung.

Es gibt die Möglichkeit zur Erstellung von Berichten. Diese Berichte können durch eigene Wünsche frei konfiguriert werden. Man kann sich die zu verwendenden Statistiken und die Berichtsperiode auswählen. Weiters gibt es die Möglichkeit sich diese Berichte in periodischen Abständen automatisch generieren zu lassen. Zum Beispiel 1 Bericht pro Monat.

Erstellung von Benachrichtigungen und Anomalien. Man kann sich selbst Benachrichtigungen konfigurieren, bei welchen man die zu kontrollierenden Zähler und die höchst zulässige Abweichung angibt. Bei größerer Abweichung wird man benachrichtigt. Um das Benachrichtigungssystem dynamischer zu gestalten gibt es die Möglichkeit sich selbst Anomalien zu erstellen, welche

für die Benachrichtigerstellung berücksichtigt werden.

Erstellung von eigenen Zählern auf Raspberry PIs. Auf den Mini-Computern können eigene Zähler für beispielsweise Strom oder Wärmemenge installiert werden. Diese Verbrauchswerte werden auch in die Datenbank eingepflegt und können für Auswertungen und Benachrichtigungen verwendet werden.

Die Android-App bietet die Möglichkeit die zuvor in der Webpage erstellten Statistiken anzuzeigen. Hier gibt es auch die Möglichkeit die Statistik als Bild am Smartphone zu speichern. Für die bessere Kontrolle der Verbrauchswerte, werden die Benachrichtigungen mit allen relevanten Informationen auch am Smartphone angezeigt.

Im Administrationstool können neue Administratoren und Administratorinnen ernannt und konfiguriert werden. Anhand von Postleitzahlen werden diesen ihre Zuständigkeitsbereiche zugewiesen. Diese Systembetreuer und Systembetreuerinnen managen die Freigaben für das AEMS-System.

## **2 Individuelle Zielsetzungen des Projektteams**

### **2.1 Niklas Graf**

- Erstellung des BOTs, welcher die Verbrauchswerte der AMIS-Stromzähler von der NetzOnline Website abruft und in der Datenbank speichert
- Programmierung des Backend für die beiden Webpages „AEMS- Advanced Energy Monitoring System“ und „AEMS - Admin“
- Entwicklung der aems-apilib für die Kommunikation zwischen den Clients und dem Java REST Service
- Grafische Umsetzung der Statistiken und Berichte inkl. Downloadfunktion dieser im Browser

### **2.2 Lukas Knoll**

- Entwicklung der Webpages „AEMS - Advanced Energy Monitoring System“ und „AEMS - Admin“
- Erweiterung und Adaption der Webpages auf XHTML für die Verwendung von JSF
- Programmierung diverser Funktionen für die Webpages mit Javascript und jQuery
- Entwicklung der gesamten Android-Application (Frontend und Backend)
- Konfiguration und Installation des Servers
- Projektmanagement und Controlling

### **2.3 Sebastian Mandl**

- Entwicklung des Datenmodells und der Datenbank
- Entwicklung eines Plugin-Systems für das Auslesen von Zählern via Raspberry PI
- Erstellung des Java REST Dienstes unter Verwendung von GraphQL
- Programmierung der Kommunikations-, und Verschlüsselungsstruktur
- Implementierung einer OPENHAB Anbindung
- Implementieren einer Script-Sprache zur Anomalierkennung
- Entwicklung einer graphischen Lösung zur Anomaliedefinition

## 3 Umsetzung

### 3.1 Umsetzung Web-Application

#### 3.1.1 Frontend

##### 3.1.1.1 Allgemein

Die HTML Seiten entsprechen dem HTML 5 Standard. Für die Stylesheets wurde CSS in der Version 3 verwendet und als Design-Library Bootstrap in der Version 3.3.7. Es wurden 2 Websites programmiert. Die erste Website ist das AEMS-Tool für die Verwaltung des Energieverbrauchs und die zweite Website ist das Administrationstool für die Verwaltung der Zugriffsrechte.

##### 3.1.1.2 Verwendete Tools

Für die Erstellung der HTML-Webpages wurde das Programm Adobe Dreamweaver in der Version CC 2017 verwendet. Damit wurde der Aufbau der HTML-Seiten und die dazugehörigen Stylesheets gemacht. Für die Entwicklung diverser Javascript-Funktionen wurde das Notepad++ in der Version 7.5 verwendet.

##### 3.1.1.3 HTML Pages Struktur

Das Webtool AEMS dient der einfachen Übersicht der Verbrauchswerte, Erstellung von Berichten und Konfiguration von Benachrichtigungen. Diese Funktionalität wurde auf mehrere Seiten aufgeteilt. Der Aufbau dieser Webpages ist auf allen Seiten gleich.

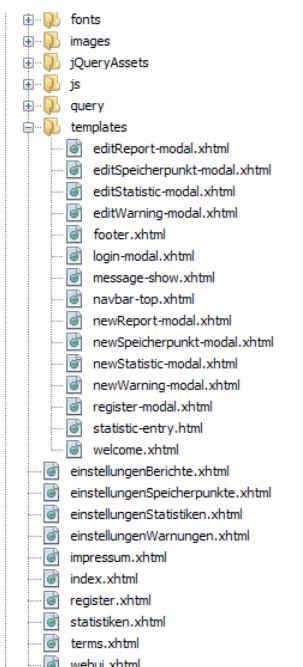


Abb. 1: Übersicht Frontend Dateien

### 3.1.1.4 HTML-Pages Grundgerüst

Die einzelnen Webpages bestehen nur aus dem Grundgerüst der für die Seite notwendigen Elemente. Alle komplexeren Elemente wurden in andere Dateien ausgelagert und werden mit dem Tag <ui:include> eingebunden. Dies sorgt für eine gute Übersichtlichkeit und eine einfache Verwendung der Elemente.

```
<head>
    <meta charset="utf-8"/>
    <meta http-equiv="Content-Type" content="text/html"/>
    <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
    <meta name="viewport" content="width=device-width, initial-scale=1"/>

    <title>AEMS - Advanced Energy Monitoring System</title>

    <link href="css/bootstrap.css" rel="stylesheet"/>
    <link href="css/main.css" rel="stylesheet" type="text/css"/>
    <link href="css/index.css" rel="stylesheet" type="text/css"/>
    <link href="css/login-modal.css" rel="stylesheet" type="text/css"/>
</head>

<body style="padding-top: 70px">
    <f:event type="preRenderView" listener="#{notificationBean.init}" />
    <ui:include src="/templates/navbar-top.xhtml"></ui:include>

    <c:if test="#{!user.loggedIn}">
        <ui:include src="/templates/welcome.xhtml"></ui:include>
    </c:if>
    <!-- Seiteninhalt -->
    <c:if test="#{user.loggedIn}">
        <article class="statistics">
            <ul class="listStyle">
                <c:forEach items="#{statisticDisplayBean.statistics}" var="statistic">
                    <li class="statistikElement">
                        <div class="statistic-container">
                            <h2 id="h_#{statistic.name}" class="stat-title">#{statistic.name}</h2>
                            <canvas id="#{statistic.id}" width="600" height="400"></canvas>
                            <button class="btn btn-default download-pdf"><span class="glyphicon glyphicon-download-alt"></span> PDF Herunterladen</button>
                        </div>
                    </li>
                </c:forEach>
            </ul>
        </article>
    </c:if>
</body>
```

Abb. 2: Startseite AEMS

Die Grundstruktur der Seiten beinhaltet den HTML-Aufbau, die Imports der einzelnen Elemente und die Verweise auf die zu verwendenden Libraries, CSS-, und Javascript-Dateien.

Durch die Verwendung von <ui:include> werden die Elemente mit dem Tag <ui:composition> an die entsprechende Stelle geladen.

```
<ui:composition
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    xmlns:sp="http://xmlns.jcp.org/jsf/pstl/core"
    xmlns:pt="http://xmlns.jcp.org/jsf/passthrough">

    <c:if test="#{register.isRegistering()}">
        <div class="welcome">
            <h1>Willkommen bei AEMS</h1>
            <p>
                Wir freuen uns, dass Sie unser neues System verwenden wollen.<br />
                Mit AEMS - Advanced Energy Monitoring System ist es Ihnen möglich, Ihre Verbrauchswerte grafisch aufbereiten und auswerten zu lassen.<br />
                Auch die Möglichkeit sich Benachrichtigungen und Warnungen bei abweichenden Verbrauchswerten anzeigen zu lassen ist durch dieses System möglich.<br />
                Für fortgeschrittenen Benutzer gibt es auch noch die Möglichkeit sich selbst Zähler auf Raspberry-Pi's einzurichten und diese in das AEMS-System einzubinden.<br />
                Besuchen Sie uns auch im Android-App Store. Dort gibt es unter dem Titel "AEMS" auch noch eine App für Ihr Smartphone oder Tablet.
            </p>
        <br />
        <a href="#" data-toggle="modal" data-target="#login-modal" data-backdrop="false" data-dismiss="modal" class="btn btn-default">Login</a>
        <a href="#" data-toggle="modal" data-target="#register-modal" data-backdrop="false" data-dismiss="modal" class="btn btn-default">Registrieren</a>
    </c:if>
</ui:composition>
```

Abb. 3: UI-Composition

Diese UI-Compositions fungieren als ein Set von Elementen. Inhalte, wie der Seiteninhalt, oder ganze Formulare wurden in solche UI-Compositions ausgelagert.

### 3.1.1.5 Design

Für das Design einzelner Elemente der Webpages wurde die Designlibrary Bootstrap verwendet.

<https://getbootstrap.com/docs/3.3/getting-started/> [22.07.2017]

Weiters wurden eigene Designs und Designerweiterungen entworfen. Die Webpages wurden somit dynamisch gestaltet, um eine gute Übersichtlichkeit auf allen Geräten (PC, Notebook, Tablet und Smartphone) gewährleisten zu können.

```
.statistics .listStyle {
    list-style-type: none;
    margin-left: 3%;
}

@media screen and (min-width:760px) and (max-width:1200px)
{
    .statistics {
        margin-top: 4em;
        margin-left: 3%;
    }
    .statistic-container {
        width: 650px;
        margin-bottom: 5%;
        display: inline-block;
    }

    .statistics .listStyle {
        list-style-type: none;
        margin-left: 3%;
    }
}

.actionBox {
    border-radius: 5px;
    background-color: orange;
    color: black;
    padding: 15px;
    text-align: center;
    font-size: 2em;
    width: 100px;
    text-decoration: none;
}
```

Abb. 4: CSS

Für eine gute Verwendbarkeit auf allen Gerätetypen, wurde darauf geachtet den Elementen und Abständen zwischen den Elementen, keine festen Größen zu geben, sondern diese durch die Verwendung von prozentuellen Verhältnissen besser zu strukturieren.

Um das Design auf den verschiedenen Bildschirmgrößen weiter zu verbessern, wurden die Eigenschaften @media (min-width) and (max-width) verwendet. Damit ist es möglich das selbe Element für verschiedene Displaygrößen anzupassen.

Der CSS-Code für das Design wurde nur in die dafür vorgesehenen CSS-Dateien und nicht in die HTML-Pages geschrieben.

Als Beispiel für ein Element, für welches Bootstrap in Kombination mit eigenen CSS Styles verwendet wurde, dient die Navigation-Bar.

```
<nav class="navbar navbar-default navbar-inverse navbar-fixed-top navbarMainFormat">
    <div class="container-fluid">
        <!-- Brand and toggle get grouped for better mobile display -->
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed"
                data-toggle="collapse" data-target="#topFixedNavbar"
                aria-expanded="false"><span class="sr-only">
                Toggle navigation</span>
            <span class="icon-bar"></span><span class="icon-bar">
            </span><span class="icon-bar"></span></button>
            <h1 outcome="index" class="navbar-brand titleFormat">
                
            </h1>
        </div>
        <!-- Collect the nav links, forms, and other content for toggling -->
        <c:if test="#{user.loggedIn}">
            <div class="collapse navbar-collapse" id="topFixedNavbar">

                <ul class="nav navbar-nav navbar-right">

                    <!-- Menü Statistiken -->
                    <li class="navBarItems"><h2>Statistik</h2></li>

                    <!-- Menü Einstellungen -->
                    <li class="dropdown"><a href="#" class="dropdown-toggle navBarItems"
                        data-toggle="dropdown" role="button"
                        aria-haspopup="true" aria-expanded="false">
                        Einstellungen<span class="caret"></span></a>
                        <ul class="dropdown-menu">
                            <li><h2>Einstellungen Speicherpunkte</h2></li>
                            <li role="separator" class="divider"></li>
                        </ul>
                    </li>
                </ul>
            </div>
        </c:if>
    </div>

```

Abb. 5: Navigation-Bar

Hier ist ein Ausschnitt aus der Navigation-Bar zu sehen.

Für das Grunddesign wurde hier Bootstrap verwendet. Für eine andere Anordnung, andere Größe der Elemente und das Verhalten auf verschiedenen Displaygrößen, wurden eigene Styles entworfen. Dies gilt für den Grundaufbau der Navigation-Bar, wie auch für die darin enthaltenen Elemente.

### 3.1.1.6 Modals

Um eine gute Übersichtlichkeit auf den Websites zu gewährleisten, wurden die Eingabefelder nicht in den Websites selbst integriert, sondern als Dialoge (Modals) ausgelagert. Diese Modals öffnen sich beim Klick auf die Buttons für die Erstellung oder Bearbeitung von Elementen, wie Statistiken, Berichte, oder Benachrichtigungen.

```
<button id="createStatistic" class="btn btn-default ownButton"
    data-toggle="modal" data-target="#newStatistic-modal"
    data-backdrop="false" data-dismiss="modal">
    Neue Statistik anlegen</button>
```

Abb. 6: Modal aufruf

Der Aufruf eines Modals funktioniert wie folgt:

Man muss einen Data-Toggle angeben. Das Element, welches den Data-Toggle bekommt, wird somit fokussiert. Durch die Eingabe von „modal“ bekommt somit das Modal welches aufgerufen wird den Fokus.

Unter Data-Target muss man anschließend das Element angeben, welches aufgerufen werden soll. Also die Id des Modals.

Mit Data-Backdrop gibt man an, ob der Hintergrund auf Grau gesetzt werden soll oder nicht. Diese Auswahl führt zu einer besseren Übersicht, ob das Modal fokussiert ist, oder nicht.

Mit dem Befehl Data-Dismiss fügt man dem Modal einen Button mit einem „X“ hinzu, um das Modal zu schließen.

```
<div class="modal fade" id="newStatistic-modal"
    role="dialog" aria-labelledby="myModalLabel"
    aria-hidden="true" style="display: none;">
    <div class="modal-dialog">
        <div class="loginmodal-container">
            <h1>Neue Statistik erstellen</h1><br />
            <h2>Statistikname</h2>
            <input type="text" id="statistikNameInput" value="Statistikname">
            <h3>Werte</h3>
            <table border="1">
                <thead>
                    <tr>
                        <th>Wert</th>
                        <th>Wert</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>1</td>
                        <td>1</td>
                    </tr>
                    <tr>
                        <td>2</td>
                        <td>2</td>
                    </tr>
                    <tr>
                        <td>3</td>
                        <td>3</td>
                    </tr>
                    <tr>
                        <td>4</td>
                        <td>4</td>
                    </tr>
                    <tr>
                        <td>5</td>
                        <td>5</td>
                    </tr>
                </tbody>
            </table>
            <div style="text-align: right; margin-top: 10px;">
                <button type="button" class="btn btn-primary" data-dismiss="modal">Abbrechen
                <button type="button" class="btn btn-primary">Erstellen
            </div>
        </div>
    </div>

```

Abb. 7: Modal

Hier ist nun das Modal selbst zu sehen. Dieses besitzt die Id, welche beim Modalaufruf als Data-Target angegeben wurde.

Durch den Befehl role= "dialog" wird das Modal in einem neuen Fenster (Dialog) geöffnet und nicht in die Website selbst eingebettet.

Falls das Modal nicht aufgerufen werden kann, wird stattdessen der Text, welcher in aria-labelledby steht angezeigt.

Der Befehl `aria-hidden` sorgt dafür, dass der Inhalt nicht von einem Screen-Reader vorgelesen wird.

Für das Design der Modals wurde eine Library aus dem Internet verwendet. Weitere Designanpassungen wurden aber auch selbst vorgenommen.

<https://bootsnipp.com/snippets/featured/clean-modal-login-form> [25.07.2017]

### 3.1.1.7 Startseite

Zu Beginn werden auf der Startseite nur ein Text über die Funktionen des AEMS-Systems und die Funktionen für Login und Registrieren bereitgestellt.

Der Seiteninhalt der Startseite wird erst nach der Anmeldung oder Registrierung angezeigt. Diese Funktionalität wird wieder mit ui:include und den entsprechenden ui:compositions gelöst.

```
<c:if test="#{!user.loggedIn}">
    <ui:include src="/templates/welcome.xhtml"></ui:include>
</c:if>
<!-- Seiteninhalt --&gt;
&lt;c:if test="#{user.loggedIn}"&gt;
    &lt;article class="statistics"&gt;

        &lt;ul class="listStyle"&gt;

            &lt;c:forEach items="#{statisticDisplayBean.statistics}" var="statistic"&gt;
                &lt;li class="statistikElement"&gt;
                    &lt;div class="statistic-container"&gt;
                        &lt;h2 id="h_#{statistic.name}" class="stat-title"&gt;#{statistic.name}&lt;/h2&gt;
                        &lt;canvas id="#{statistic.id}" width="600" height="400"&gt;&lt;/canvas&gt;
                        &lt;button class="btn btn-default download-pdf"&gt;
                            &lt;span class="glyphicon glyphicon-download-alt"&gt;&lt;/span&gt;
                            PDF Herunterladen
                        &lt;/button&gt;
                    &lt;/div&gt;
                &lt;/li&gt;
            &lt;/c:forEach&gt;
        &lt;/ul&gt;
    &lt;/article&gt;
&lt;/c:if&gt;</pre>
```

Abb. 8: Anzeige Startseite

Hier ist zu sehen, dass es eine Abfrage gibt, ob der Benutzer/die Benutzerin angemeldet ist. Wenn nicht, wird durch die Einbindung des welcome-modals die Startseite mit den Infos über das AEMS-System angezeigt. Wenn der Nutzer/die Nutzerin angemeldet ist, erscheinen direkt seine für die Startseite konfigurierten Statistiken.

### 3.1.1.8 Einstellungen - Funktionen

Für die Erstellung und Bearbeitung von Statistiken, Berichten und Benachrichtigungen gibt es jeweils entsprechende Seiten. Für eine einfache Verständlichkeit der Funktionen (Bearbeiten, Löschen, Zur App hinzufügen, Zur Startseite hinzufügen) - Funktionen abhängig von den verwendeten Tools - werden Icons mit Tooltips verwendet.

```

<h:form>
    <div id="allList" class="list-group statisticsList">
        <c:forEach items="#{statisticBean.statistics}" var="item">
            <div class="list-group-item">
                <h4 class="list-group-item-heading">#{item.name}</h4>
                <p class="list-group-item-text">#{item.annotation}</p>
                <div class="btn-glyph">
                    <div class="edit-statistic btn btn-default"
                        data-id="#{item.id}" title="Statistik bearbeiten"
                        data-toggle="modal" data-target="#editStatistic-modal"
                        data-backdrop="false" data-dismiss="modal">
                        <span class="glyphicon glyphicon-pencil"></span></div>
                    <h:commandLink action="#{statisticActionBean.addToAndroid(item.id)}">
                        <button class="btn btn-default" title="Statistik zu Android-App hinzufügen">
                            <span class="glyphicon glyphicon-phone"></span></button>
                    </h:commandLink>
                    <h:commandLink action="#{statisticActionBean.addToIndex(item.id)}">
                        <button class="btn btn-default" title="Statistik zu Startseite hinzufügen">
                            <span class="glyphicon glyphicon-home"></span></button>
                    </h:commandLink>
                    <h:commandLink action="#{statisticActionBean.remove(item.id)}">
                        <button class="btn btn-default" title="Statistik löschen">
                            <span class="glyphicon glyphicon-trash"></span></button>
                    </h:commandLink>
                </div>
            </div>
        </c:forEach>
    </div>

```

Abb. 9: Funktionen Webpages

Für die Funktionen werden Buttons verwendet, mit den entsprechenden Icons. Je nachdem, welche Funktion aufgerufen wird, öffnet sich ein Modal zur Bearbeitung des Eintrags, oder es wird direkt eine Aktion ausgeführt (zum Beispiel Löschen).

Für die Icons wird die Library Bootstrap Glyphicon verwendet.

[https://www.w3schools.com/bootstrap/bootstrap\\_ref\\_comp\\_glyphs.asp](https://www.w3schools.com/bootstrap/bootstrap_ref_comp_glyphs.asp) [11.12.2017]

### 3.1.1.9 Mobile Functions

Für eine leichtere Verwendung auf mobilen Geräten, wurden einige Elemente speziell für mobile Browser angepasst. Als Beispiel die Fußzeile.

```

$(document).load($(window).bind("resize", startFormat));

function startFormat(){
    footerFormat();
    navbarFormat();
}

function footerFormat(){

    if (/Mobi/.test(navigator.userAgent)) {
        $('footer').removeClass("footer");
        $('footer').removeClass("navbar-fixed-bottom");
    }
}

```

Abb. 10: Mobile Function

In diesem Javascript-Code wird überprüft, ob die Website von einem mobilen Browser aufgerufen wird. Wenn der Browser eine mobile Version ist, wird die Fußzeile nicht mehr am unteren Rand des Displays fixiert. Das sorgt dafür, dass am Smartphone mehr inhaltsrelevante Daten angezeigt werden können und der Platz nicht durch Standardelemente verschwendet wird.

### **3.1.1.11 Notifications**

In der Navigation-Bar wird man über Funktionen des AEMS-Systems benachrichtigt. Zum Beispiel bei einer Warnung über zu hohen Verbrauch, oder wenn ein neuer Bericht zum Download zur Verfügung steht.

Für die Erstellung von Benachrichtigungen wird die Library Noty.js verwendet. Diese ist für das Frontend der Notifications, wie auch das Backend zuständig. Funktionalität und aussehen der Notifications wurden mit eigenen Funktionen angepasst.

<https://ned.im/noty/#/> [05.01.2018]

### **3.1.1.10 Vergleich AEMS und AEMS-Admin**

Für die designtechnische Umsetzung der beiden Webseiten AEMS und AEMS-Admin wurden die selben Technologien verwendet. Der Unterschied der beiden Tools liegt nur in ihrer Funktionalität, nicht aber in ihren Designs.

### **3.1.2 Schnittstelle XHTML**

Um für die Webpages eine Funktionalität zu erhalten, wird mit Java Server Faces, kurz JSF gearbeitet. Die Voraussetzung für die Verwendung von JSF ist XHTML. XHTML ist eine Abwandlung von HTML-Code.

Im eigentlichen Sinne ist XHTML, HTML-Code im XML-Format.

Während die Richtlinien bei HTML nicht so streng sind, wird der XHTML-Code strikter auf Korrektheit überprüft. Während HTML Tags nicht zwingend geschlossen werden müssen, ist dies bei XHTML zwingend erforderlich. Auch müssen Attributwerte zwingend unter Hochkommas gestellt werden.

Im Grunde kann HTML-Code leicht in XHTML-Code umgewandelt werden. Jedoch ist darauf zu achten, dass nicht alle Tag-Attribute, welche es in HTML gibt, auch in XHTML unterstützt werden.

Im Kapitel 3.1 gezeigter Code ist bereits XHTML-Code. Dort wurde bereits mit Java Beans gearbeitet, welche im nächsten Kapitel erklärt werden.

Würde HTML-, anstatt XHTML-Code für die Programmierung verwendet werden, wäre der Code für den Interpreter unleserlich.

[https://www.w3schools.com/html/html\\_xhtml.asp](https://www.w3schools.com/html/html_xhtml.asp) [23.09.2017]

### 3.1.3 Backend

#### 3.1.3.1 Tools und Umgebung

Um die benötigten war-Dateien zu generieren, welche anschließend auf einem Webserver laufen, wurde die Programmierumgebung NetBeans IDE 8.2 (Build 201609300101) verwendet. Zum Ausführen dieser Dateien wird mindestens Java 7 benötigt. Es wurde in NetBeans ein neues „Web Application“ Projekt erstellt, welches Maven als Build-Management-Tool verwendet. Als Framework wurde das JSF Framework gewählt, da das Konzept der Managed Beans für die zu bewältigende Aufgabe passend erschien.

#### 3.1.3.2 Vorbereitungen

Die fertigen HTML-Dateien mussten zu XHTML Dateien umgeschrieben werden, um darin Programmierlogik ausführen zu können. Dabei war vor allem darauf zu achten, dass die XHTML-Dateien dem XML-Standard entsprechen, da sie ansonsten für den Interpreter unlesbar sind.

HTML-Komponenten, welche im gesamten Webinterface verwendet werden, wurden in eigene Dateien ausgelagert, um sie danach schnell und einfach an gewünschten Stellen einzufügen zu können.

#### 3.1.3.3 Implementierung der Web-Logik

Die Web-Logik wurde mit Managed Beans und Servlets umgesetzt. Dabei werden die Beans in zwei verschiedene Kategorien unterteilt.

Die Kategorie der **DisplayBeans** ist für das Anzeigen von verschiedenen Benutzerdaten zuständig. Diese Benutzerdaten müssen zuvor von der AEMS-API abgefragt werden. Dieses Abfragen der Daten passiert einmalig am Beginn der Benutzersession. Danach werden die Werte periodisch erneuert, es sei denn die update Methode der Bean wird aufgerufen.

Die Art von Daten, welche von der Bean gespeichert werden, lässt sich bereits am Namen selber erkennen. So werden Daten über die Zähler des Benutzers/ der Benutzerin beispielsweise in der UserMeterBean abgespeichert. Daten über Benachrichtigungen finden sich in der NotificationBean.

Diese Logik für den Webserver und den Webserver der Admin-Page ist identisch.

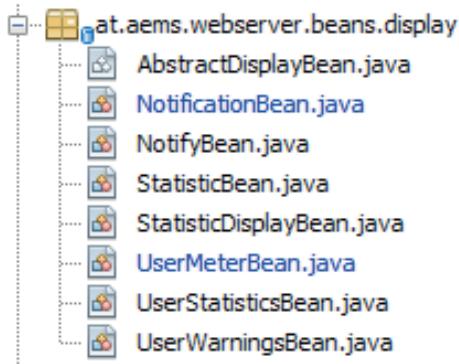


Abb. 11: Übersicht über verschiedene DisplayBeans

Jede DisplayBean erbt von der abstrakten Basisklasse **AbstractDisplayBean**. Diese definiert im Wesentlichen eine Methode, die überschrieben werden muss: Die Methode „update()“. In dieser Methode sollen die benötigten Daten vom REST-Service abgefragt werden. Die Methode wird nach dem erstmaligen Aufruf etwa alle fünf Minuten erneut aufgerufen, um den Informationsstand vom Webinterface wieder auf jenen der Datenbank zu bringen. Die „update()“ Methode kann auch vom Programmierer aufgerufen werden. Dies ist sinnvoll wenn der Benutzer/ die Benutzerin am Webinterface Aktionen ausführt, welche die angezeigten Informationen beeinflussen.

```

private Set<String> meterTypes;
private Map<String, String> meters;

@Override
public void update() {
    // Initialize to empty
    meterTypes = new HashSet<>();
    meters = new HashMap<>();

    // retrieve GraphQL-Query from text file and send to REST
    AemsQueryAction query = new AemsQueryAction
        (userBean.getAemsUser(), EncryptionType.SSL);
    query.setQuery(AemsUtils.getQuery("meters_simple", NewMap.of("USER_"
    JSONArray data = getJsonResponse(query);

    // iterate over response and add all meters
    for(int i = 0; i < data.size(); i++) {
        JSONObject j = data.get(i).getAsJsonObject();
        String id = j.get("id").getAsString();
        String type = j.get("metertype").getAsJsonObject().get("display");
        meters.put(id, type);
    }

    // Since meterTypes is a HashSet, duplicate values will be erased
    meterTypes.addAll(meters.values());
}

```

Abb. 12: Update-Methode der UserMeterBean,

Die Kategorie der **ActionBeans** ist für Benutzeraktionen zuständig. Benutzeraktionen sind beispielsweise das Ausfüllen eines Formulars oder das Klicken einer Schaltfläche. Beans dieser Art stellen die Schnittstelle zwischen den Eingabeelementen auf der Clientseite und den benötigten Informationen auf der Serverseite dar.

```

private String name;

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

<h:form id="newStatistic">
    <h4>Statistikname</h4>
    <h:inputText value="#{newStatistic.name}"
        maxlength="64" pt:placeholder="Meine Statistik"

    <h4>Zählertyp</h4>
    <div class="btn-group">
        <button type="button" class="btn btn-default dro

```

Abb. 13: Anwendungsbeispiel einer ActionBean nach dem MVC-Prinzip von JSF.

Bei einfachen Textfeldern konnte das Binding ohne weitere Umstände mithilfe des value Attributes ermöglicht werden. Jedoch können nicht alle Eingabeelemente von HTML auf XHTML umgeschrieben werden, da der JSF HTML Namespace (<http://xmlns.jcp.org/jsf/html>) nicht alle HTML Elemente enthält und das Value-Binding nur bei Elementen aus diesem Namespace in beide Richtungen funktioniert. Deswegen musste für komplexere Elemente eine andere Lösung gefunden werden.

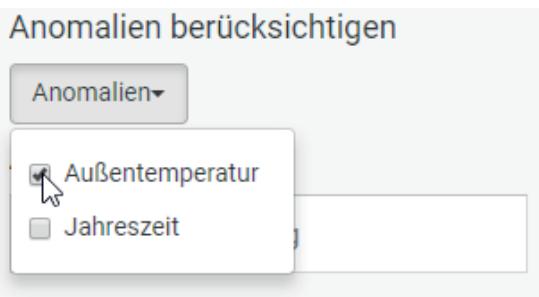


Abb. 14: Beispiel für ein komplexeres Eingabeelement

Dieses Problem wurde mit versteckten Eingabefeldern und JavaScript gelöst. Die ausgewählten Werte werden beim Klicken des „Submit-Buttons“ mittels JavaScript in das versteckte Eingabefeld geschrieben. Als Trennzeichen wurde der Strichpunkt verwendet. Das eigentliche Binding geschieht beim Value-Attribut des versteckten Eingabefeldes.

Jede ActionBean stellt eine oder mehrere Methoden zum Ausführen verschiedener Aktionen bereit. Diese Methoden sind mit dem Prefix „do“ gekennzeichnet. Das bedeutet, dass eine Aktion zum Bearbeiten beispielsweise mit der Methode „doEdit()“ aufgerufen werden kann.

In diesen Methoden geschieht die nicht-abfragende Interaktion mit dem REST-Service. Elemente wie Statistiken, Berichte oder Warnungen werden hier angelegt, bearbeitet oder gelöscht.

Da jede ActionBean von der abstrakten Basisklasse AbstractActionBean erbt, haben alle ActionBeans Zugriff auf andere wichtige Beans, welche etwa Informationen über den eingeloggten Benutzer bereitstellen.

Außerdem implementiert die Basisklasse auch eine Methode „callUpdateOn(beanName)“, welche es ermöglicht, die „update()“ Methode einer DisplayBean aufzurufen.

```
public class WarningActionBean extends AbstractActionBean {  
  
    public String doDelete(Integer id) {  
        AemsDeleteAction delete = new AemsDeleteAction(userBean.get  
        delete.setTable("Notifications");  
        delete.setIdColumn("id", id);  
        try {  
            AemsAPI.call0(delete, null);  
            notify.setMessage("Benachrichtigung wurde entfernt!");  
            callUpdateOn("userWarningsBean");  
        } catch(IOException e) {  
            notify.setMessage("Ein Fehler ist aufgetreten!");  
        }  
        return "einstellungenWarnungen";  
    }  
}
```

Abb. 15: Implementierung der „doDelete“ Methode in der WarningActionBean.

### 3.1.3.4 UserBean

Diese Bean stellt Informationen über den eingeloggten Benutzer/Benutzerin zur Verfügung. Diese Infomationen sind die Benutzer-ID, der Benutzername und das Passwort. Diese Elemente werden für die Authentifizierung bei der API benötigt.

### 3.1.3.5 LoginBean

Diese Bean übernimmt den Login- und Logout Prozess. Im Erfolgsfall wird der Benutzer/ die Benutzerin auf die Startseite weitergeleitet. Falls das Login fehlschlagen sollte, erhält der Benutzer/ die Benutzerin über die NotifyBean eine entsprechende Benachrichtigung.

### 3.1.3.6 NotifyBean

Die NotifyBean wird verwendet, um Statusnachrichten an den Benutzer/ die Benutzerin zu übermitteln. Diese Nachrichten erscheinen am rechten oberen Bildschirmrand als Pop-Up Benachrichtigung. Dies wurde mithilfe der JavaScript-Library notify.js verwirklicht.

<https://notifyjs.com/> [20.02.2018]

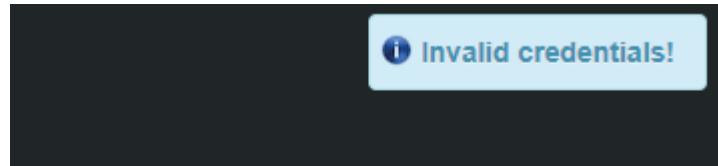


Abb. 16: Pop-Up bei fehlgeschlagenen Login-Versuch.

### 3.1.3.7 NotificationBean

Die NotificationBeans zeigt im Gegensatz zu der NotifyBean keine Statusmeldungen, sondern Benachrichtigungen über den Energieverbrauch an. Diese werden nicht sofort angezeigt, sondern müssen mit einem Klick auf die Benachrichtigungs-Glocke am rechten oberen Rand des Bildschirms eingeblendet werden.

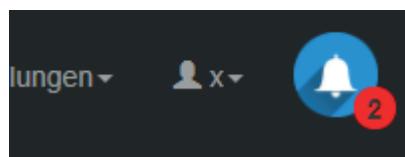


Abb. 17: Glocken-Symbol mit zwei neuen Benachrichtigungen

### 3.1.3.8 AemsUtils

Zur Verbesserung der Lesbarkeit des Programmcodes, wird häufig gebrauchter Code oftmals in sogenannten Utils-Klassen eingebunden. Die AemsUtils Klasse bietet eine Methode zum Auslesen von Text in Dateien, welche in einem spezifischen Verzeichnis liegen. Die Inhalte dieser Textdateien repräsentieren GraphQL Abfragen.

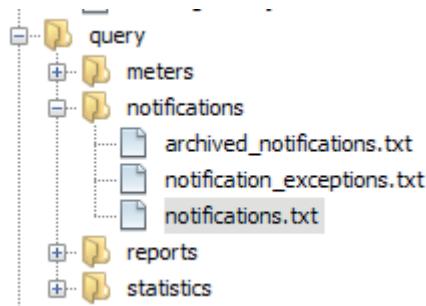


Abb. 18: Verzeichnisstruktur der Abfrage-Dateien

Damit der Text von der „AemsUtils.getQuery(name, platzhalter)“ Methode gelesen werden kann, muss sich die Datei um Unterverzeichnis „query“ befinden. Weitere Unterverzeichnisse dienen nur der Übersicht. Sobald diese Methode aufgerufen wird, wird das gesamte „query“ Verzeichnis rekursiv nach Dateien abgesucht. Wird eine Datei mit dem gewünschten Namen gefunden, so wird ihr Inhalt ausgelesen und dem Aufrufer zurückgegeben.

Der zweite Parameter „platzhalter“ stellt eine Liste von Platzhaltern in der gesuchten Datei dar, und legt fest, durch was diese Platzhalter ersetzt werden sollen. So kann eine Abfrage in der Datei statisch definiert werden, und trotzdem durch Platzhalter flexibel angepasst werden.

```
archived_meter_notifications ( meter : #METER_ID# ) {  
    id  
    name  
    type  
    min_positive_deviation  
    max_positive_deviation  
}
```

Abb. 19: Beispiel einer GraphQL Abfrage mit Platzhalter für die Zähler ID - #METER\_ID#

```
AemsUtils.getQuery("archived_notifications",  
    ...  
    NewMap.of("METER_ID", "AT00000123"));
```

Abb. 20: Einsatz der AemsUtils.getQuery() Methode

### 3.1.3.9 NewMap

Die NewMap Klasse ist ebenfalls eine Utils Klasse, welche zum schnellen Erstellen von (Hash) Maps verwendet wird. Die „of(...)“ Methode akzeptiert eine beliebige Anzahl von Parametern, wobei immer die zwei aufeinanderfolgenden Argumente als Key und Value betrachtet und in die HashMap eingefügt werden.

### 3.1.3.10 EditServlet

Da sie im Webinterface nur selten benötigt werden, werden umfassende Daten über Statistiken, Berichte und Benachrichtigungen standardmäßig nicht geladen. Da sie beim Bearbeiten jedoch benötigt werden, müssen sie über Ajax nachgeladen werden. Dazu wird ein HTTP Request an das EditServlet gesendet, welches die Authentifikation mit der API übernimmt. Damit ist gewährleistet, dass die Authentifikation nur serverseitig passiert und nicht vom Client beeinflusst werden kann.

Um die Daten nachladen zu können, muss eine aktive Session am Webserver bestehen. Aus dieser Session werden die Anmeldedaten des Benutzers / der Benutzerin ausgelesen und die Authentifikation durchgeführt.

Durch Übergabe der ID des benötigten Objekts (Statistik, Bericht, etc.) können dessen Daten von der API gelesen und in Form von JSON zurückgegeben werden.

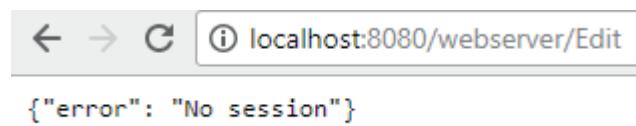


Abb. 21: Fehlermeldung bei Servlet-Aufruf ohne aktive Session

### 3.1.3.11 Anzeige der Statistiken

Das Anzeigen der Statistiken auf der Startseite wurde mit der JavaScript Bibliothek „chart.js“ ermöglicht. Diese ermöglicht es, auf Canvas-Elementen Statistiken darzustellen.

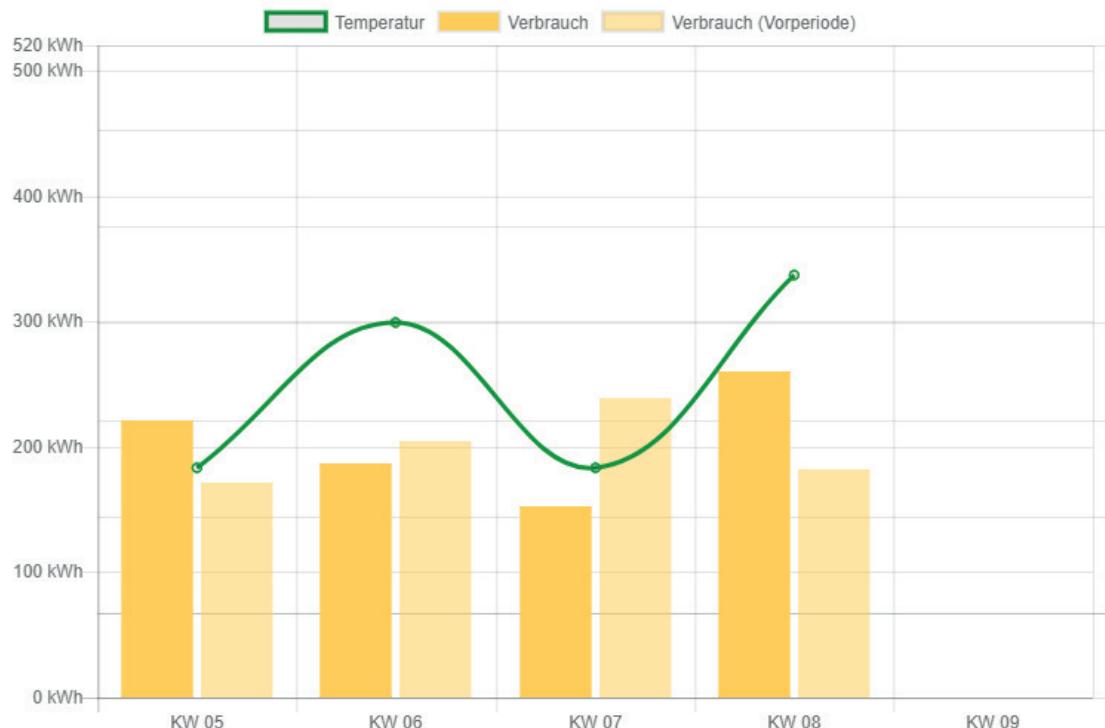


Abb. 22: Typische Ansicht einer Monatsstatistik

Dafür wurden zwei verschiedene Datasets definiert. Ein Dataset repräsentiert die in der Statistik abgebildete Anomalie. Das können verschiedenste Werte sein, von Temperatur bis zur Lichteinwirkung. Das andere Dataset repräsentiert die Energieverbrauchswerte und eventuell die Verbrauchswerte aus der Vorperiode.

Da sich Temperaturwerte eventuell sehr stark von Verbrauchswerten unterscheiden, wurden diese Datasets auf verschiedenen Achsen angezeigt, um zu verhindern, dass eines der beiden Sets unter- bzw. überdimensional dargestellt wird.

```
var mixedChart = new Chart(ctx, {
    type: 'bar',
    data: {
        datasets: dataSets,
        labels: labels
    },
    options: {
        scales: {
            yAxes: [{
                id: 'primary',
                type: 'linear',
                position: 'left',
                ticks: {
                    max: highestValueToDisplay + valueOffset,
                    min: 0,
                    callback: function (value, index, values) {
                        return value + " kWh";
                    }
                }
            }, {
                id: 'secondary',
                type: 'linear',
                position: 'right',
                ticks: {
                    max: highestAnomaly + averageAnomaly, //highest
                    min: lowestAnomaly - averageAnomaly,
                    callback: function (value, index, values) {
                        return value;
                    }
                }
            }
        }
    }
});
```

Abb. 23: Erstellung des Chart Objekts in JavaScript

Für Werte, die keine Energieverbrauchswerte darstellen, gibt es eine eigene Achse. Diese Werte werden als Liniendiagramm dargestellt. Damit die Extrempunkte dieser Kurven nicht mit dem oberen bzw. unteren Rand der Statistik abschließen wird zu den **min** und **max** Optionen des Diagramms noch ein Durchschnittswert addiert bzw. subtrahiert, wie im obigen Screenshot erkennbar ist.

### 3.1.3.12 Download der Statistiken

Um das Herunterladen der Statistiken zu realisieren, wurden die Bibliotheken jsPDF (<https://parall.ax/products/jspdf> [23.02.2018]) und html2canvas (<https://github.com/niklasvh/html2canvas/releases> [23.02.2018]) verwendet.

Mit html2canvas wurde aus dem Canvas eine Data-URL erzeugt. Diese URL enthält den gesamten Inhalt des Bildes, welches danach in das generierte PDF eingefügt wird.

Nach dem Erzeugen eines Bildes aus dem Canvas wird das PDF generiert. Die Statistik wird, zusammen mit einigen anderen Informationen, in das PDF eingefügt. Danach wird es vom Client heruntergeladen.

```
html2canvas($('#statistic'), {
    onrendered: function (canvas) {
        var imageData = canvas.toDataURL('image/png');
        var pdf = new jsPDF('l', 'mm');

        pdf.setFontSize(25);
        pdf.text(title, pdf_xaxis_indent, 35);
        pdf.setFontSize(15);
        pdf.text(dateStr, pdf_xaxis_indent, 40);

        // Add graph image
        pdf.addImage(imageData, 'PNG', pdf_xaxis_indent, 50);

        // Add Logo @ Top Left
        imageToBase64("images/logo_schrift.png", function (base64) {
            var idx = base64.indexOf(',');
            var imgData = "data:image/png;base64" + base64.substring(idx + 1);
            pdf.addImage(imgData, 'PNG', pdf_xaxis_indent - 5, 0);
            pdf.save(title + ".pdf");
        });
    }
});
```

Abb. 24: Auszug aus der Implementation des Downloads von Statistiken

## 3.2 Umsetzung Report-Bot

### 3.2.1 Allgemein

Der „Report-Bot“ ist ein Java Programm, welches mehrmals täglich Zähler- und Wetterdaten erfasst. Die Aufgabe des Report-Bots ist es, regelmäßig die neuesten Zählerdaten aus dem **Netzonline-Portal** (<https://netz-online.netzbmbh.at/> [23.02.2018]) herunterzuladen und diese an die Datenbank zu senden. Dazu werden die Anmeldedaten aller Benutzer/Benutzerinnen benötigt, welche einen Account bei dem Netzonline-Portal besitzen. Mit diesen Anmeldedaten ist es möglich, sich im Namen des Benutzers/der Benutzerin beim Netzonline-Portal anzumelden und die geforderten Daten herunterzuladen.

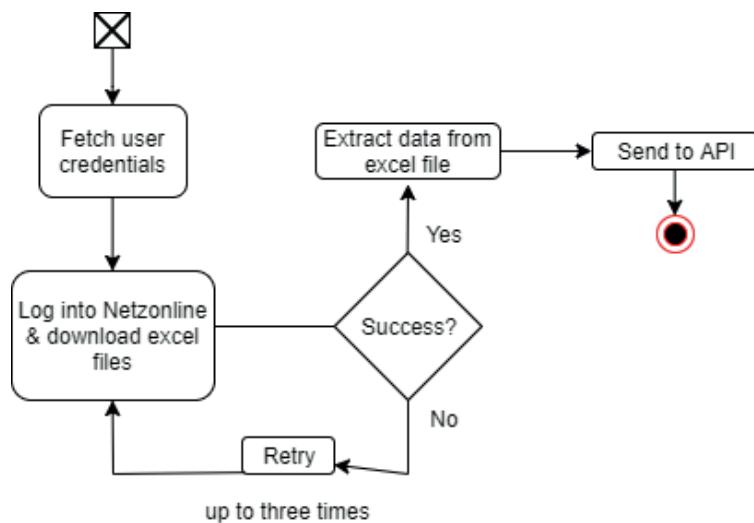


Abb. 25: Ablaufdiagramm des Report-Bots

### 3.2.2 Umsetzung

Mit der Java-Bibliothek HTMLUnit wird ein Web-Browser simuliert. Zunächst wird das Login-Formular ausgefüllt. Sollte der Username und das Passwort nicht richtig sein, wird die Datenabfrage für diesen Benutzer/ diese Benutzerin eingestellt. Sollten die Daten richtig sein, so beginnt der Bot mit dem Herunterladen der benötigten MS-Excel Dateien.

Die heruntergeladene Excel-Datei enthält drei Spalten. In der ersten Spalte ist ein Zeitstempel angegeben. Dieser gibt den Zeitpunkt an, an dem der Stromverbrauch gemessen wurde. Die zweite Spalte liefert den eigentlichen Wert, der an dem Zeitpunkt gemessen wurde. Die dritte Spalte weist auf, ob der angegebene Wert richtig ist oder einfach noch nicht gemessen wurde und deswegen Null ist.

<http://htmlunit.sourceforge.net/> [23.08.2018]

2109	22.02.2018 23:00	0,512	VALID
2110	22.02.2018 23:15	0,628	VALID
2111	22.02.2018 23:30	0,648	VALID
2112	22.02.2018 23:45	0,5	VALID
2113	23.02.2018 00:00	0,548	VALID
2114	23.02.2018 00:15	0,52	VALID
2115	23.02.2018 00:30	0	VALID
2116	23.02.2018 00:45	0	MISSING
2117	23.02.2018 01:00	0	MISSING
2118	23.02.2018 01:15	0	MISSING
2119	23.02.2018 01:30	0	MISSING
2120	23.02.2018 01:45	0	MISSING
2121	23.02.2018 02:00	0	MISSING

Abb. 26: Auszug aus einer Excel-Datei

Die heruntergeladenen Dateien sind Binärdateien. Deshalb wurde zur Extraktion der Daten eine weitere Klassenbibliothek verwendet. Apache POI ist eine Bibliothek zum Arbeiten mit Microsoft-Dokumenten. Mithilfe dieser Klassenbibliothek wurden die Werte aus den verschiedenen Spalten ausgelesen und, sollte die Spalte „VALID“ sein, an die REST-API gesendet.

<https://poi.apache.org/> [23.02.2018]

### 3.2.3 Konfiguration

Beim erstmaligen Ausführen werden vom Programm einige Dateien erstellt. Eine davon ist die **bot.properties** Datei, in welcher Einstellungen vorgenommen werden können.

```
#AEMS Bot Konfigurationsdatei
#Thu Jan 25 10:54:00 CET 2018
aems-api-url=http\://api.aems.at
xls-storage=Exceldateien
max-users-at-once=50
super-pwd=password of superuser
log-level=DEBUG
super-id=0
max-retries=1
super-name=superuser
logs-folder=Logs
logging-enabled=true
openweathermap-apikey=65a4ce346594c
```

Abb. 27: bot.properties Konfigurationsdatei

Die **aems-api-url** wird benötigt, um die Anmeldedaten von allen registrierten Benutzern/ Benutzerinnen zu erlangen. Die Werte **super-id**, **super-name** und **super-pwd** sind die Anmeldedaten des „Super-Users“. Dieser virtuelle Benutzer/Benutzerin wurde speziell für den Bot angelegt, denn nur dieser ist dazu berechtigt, Daten über alle Benutzer / Benutzerinnen abzufragen. Um Wetterdaten abfragen zu können, wird der **openweathermap-apikey** benötigt.

Um die Konfigurationsdaten aus dieser Datei auszulesen, wurde die Klasse **BotConfiguration** erstellt. Diese Klasse definiert Konstanten für den Zugriff auf die Properties. Außerdem wird die **bot.properties** Datei von dieser Klasse erstellt und mit Standardwerten befüllt, sollte die Datei nicht existieren.

```

public BotConfiguration() {
    this.configFile = new File("bot.properties");
    this.props = new Properties();
    init();
}

private void init() {
    try {
        if(!configFile.exists()) {
            configFile.createNewFile();
        }
        InputStream stream = new FileInputStream(configFile);
        this.props.load(stream);

        if(props.isEmpty()) { // Put default values
            this.props.setProperty(BotConfiguration.API_URL, "ht

```

Abb. 28: Code Ausschnitt aus der BotConfiguration Klasse

### 3.2.4 Logging

Sollte in der Konfiguration das Logging aktiviert sein, so werden vom Programm Statusmeldungen ausgegeben. Diese Meldungen erscheinen in der Konsole und werden außerdem in eine Logdatei geschrieben. So können beispielsweise eventuelle Fehler nachgewiesen werden. Der Umfang des Loggings wird von der Einstellung **log-level** in der Konfigurationsdatei bestimmt. Beim Level DEBUG werden alle Meldungen geloggt. Für den normalen Gebrauch ist dies eher ungeeignet, da sehr viele Daten geschrieben werden, was zu unnötig verbrauchtem Speicherplatz führt. Die restlichen Level sind INFO, WARN und ERROR.

Um die Speicherbelastung zu reduzieren entfernt der Bot Log-Dateien, welche älter als 30 Tage sind, selbstständig.

### 3.2.5 FileDownloader

Die FileDownloader Klasse ist die Hauptkomponente des Report-Bots, weil darin die Logik zum Herunterladen der Excel Dateien implementiert wird. Da diese Klasse das **Runnable** Interface implementiert, können mehrere FileDownloader Klassen zeitlich parallel ausgeführt werden.

Da nicht alle HTML Tags auf der Netzonline Seite mit IDs versehen sind, müssen die benötigten HTML Elemente auf andere Weise gefunden werden. Die meisten Elemente können mit XPATH gefunden werden.

```

private List<HtmlElement> getConsumptionButtons(HtmlPage page) {
    return page.getByXPath("//footer/nav/ul/li/form/input[@id='overview~submit']");
}

```

Abb. 29: Anwendung von XPATH zum Finden von Buttons

Leider ist diese Herangehensweise äußerst verwundbar gegenüber Änderungen der HTML-Struktur der Netzonline Seite. Jedoch lies die Aufgabenstellung andere Vorgehensweisen nicht oder nur teilweise zu.

### **3.2.5.1 Ablauf des Dateidownloads**

Der Programmablauf beginnt mit dem erstellen des WebClient Objekts, welchen den Web-Browser repräsentiert. Zu Beginn begibt sich der WebClient auf die Login Seite des Netzonline Portals. Dort wird das Login Formular mit den Benutzerdaten ausgefüllt.

<https://netz-online.netzgmbh.at/eServiceWeb/main.html> [25.02.2018]

## Anmeldung

Benutzername

Passwort

Abb. 30: Anmeldeforumlar

Nach erfolgreichem Login findet man sich auf einer Übersichtsseite wieder. Hier sind bis zu fünf Zähler aufgelistet. Um an die Details eines Zählers zu kommen, muss zuerst der Pfeil auf der linken Seite gedrückt werden. Danach werden die Daten für diesen Zähler nachgeladen. Das bedeutet, dass eine Verzögerung eingebaut werden musste. Erst danach sind die Schaltflächen zur Detailansicht verfügbar.

## Ihre Vertragskonten

In der Vertragsübersicht werden Ihre Vertragskonten angezeigt.

1 - 5 von 30 gefundenen (30 Vertragskonten insgesamt)

- Energiegenossenschaft Region Eferdi 4070 Eferding,  
aktives Vertragskonto (24151096)**

Abb. 31: Übersichtsseite

Nach dem Klick auf die Schaltfläche „Verbräuche“ erfolgt eine Weiterleitung auf die Detailansicht für diesen Zähler. Hier ist es durch das Klicken auf eine weitere Schaltfläche möglich, die gewünschten Exceldateien herunterzuladen.

Nach diesem Herunterladen navigiert der Bot zur Übersichtsseite zurück. Dort wird der eben beschriebene Vorgang so lange wiederholt, bis alle Zähler abgehendelt wurden.

### 3.2.6 ExcelDataExtractor

Die Excel Dateien, welche vom FileDownloader heruntergeladen werden, werden in das Verzeichnis `Exceldateien/Username/ZählerNr.xls` gespeichert. Sobald alle FileDownloader Threads ihre Aufgabe erledigt haben, wird mit der Extraktion der Daten begonnen. Die `ExcelDataExtractor` Klasse implementiert ebenfalls das Interface **Runnable** und behandelt alle Dateien für einen Benutzer/ eine Benutzerin.

```
private void init() {
    try {
        InputStream is = new FileInputStream(file);
        POIFSFileSystem fs = new POIFSFileSystem(is);
        is.close();

        HSSFWorkbook wb = new HSSFWorkbook(fs);
        this.excelSheet = wb.getSheetAt(0);

    } catch(IOException e) {
        e.printStackTrace();
    }
}
```

Abb. 32: Verwendung der Apache POI Klassenbibliothek

Die Daten werden in einer geeigneten Datenstruktur zwischengespeichert. Sobald alle `ExcelDataExtractor` Threads ihre Arbeit erledigt haben, werden diese Daten an die AEMS-API versendet. Anschließend werden alle generierten Verzeichnisse und Dateien wieder gelöscht.

### 3.2.7 Installation

Für den Report-Bot gibt es zwei Tasks. Einen für das Abrufen der Zählerdaten von der Netz-Online Website und einen für das Abrufen der Wetterdaten von Open Wheater Map.

Als Server wird ein Windows Server in der Version 2012 verwendet. Darauf wurden im Task-Scheduler die beiden Tasks für den Bot gemacht.

Die Zählerwerte sollen 1 mal täglich abgerufen werden. Die Wetterdaten stündlich. Hierfür wurden Trigger definiert.

Aufgabenstatus					
Status von Aufgaben, die im folgenden Zeitraum gestartet wurden:				Letzte 24 Stunden	
Zusammenfassung: 49 insgesamt -- 2 ausgeführt, 46 erfolgreich, 0 beendet, 1 fehlerhaft					
Aufgabenname	Ausführ...	Anfang der Au...	Ende der Ausf...	Ausgelöst durch	
☐ AemsBot (letzte Ausführung erfolgreich am 26.02.2018 01:00:04)	Erfolgreich	26.02.2018 01:0...	26.02.2018 01:0...		
AemsBot					
☐ AemsBotTemperatur (letzte Ausführung erfolgreich am 26.02.20...	Erfolgreich	26.02.2018 10:0...	26.02.2018 10:0...	Zeitplan	
AemsBotTemperatur					
AemsBotTemperatur	Erfolgreich	26.02.2018 09:0...	26.02.2018 09:0...	Zeitplan	
AemsBotTemperatur	Erfolgreich	26.02.2018 08:0...	26.02.2018 08:0...	Zeitplan	
AemsBotTemperatur	Erfolgreich	26.02.2018 07:0...	26.02.2018 07:0...	Zeitplan	

Abb. 33: Bot Tasks

Die Grundlegenden Einstellungen der beiden Tasks sind gleich. Jedoch unterscheiden sich die Trigger für die zeitliche Ausführung der beiden Tasks.

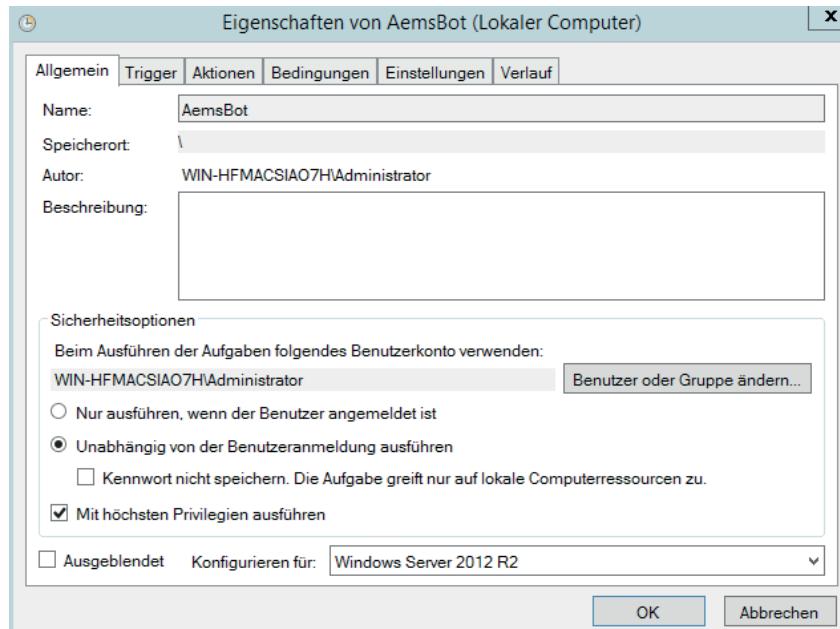


Abb. 34: Task-Allgemein

In den Grundeinstellungen wird festgelegt, dass der Task auch ausgeführt werden soll, wenn gerade niemand am Server angemeldet ist. Weiters wurde definiert, dass der Task mit den höchstmöglichen Privilegien ausgeführt wird. Das hat den Grund, dass nur Administratoren/Administratorinnen Programme für den AEMS-Dienst ausführen dürfen. Somit wird dem Task die Berechtigung „Administrator“ gegeben.

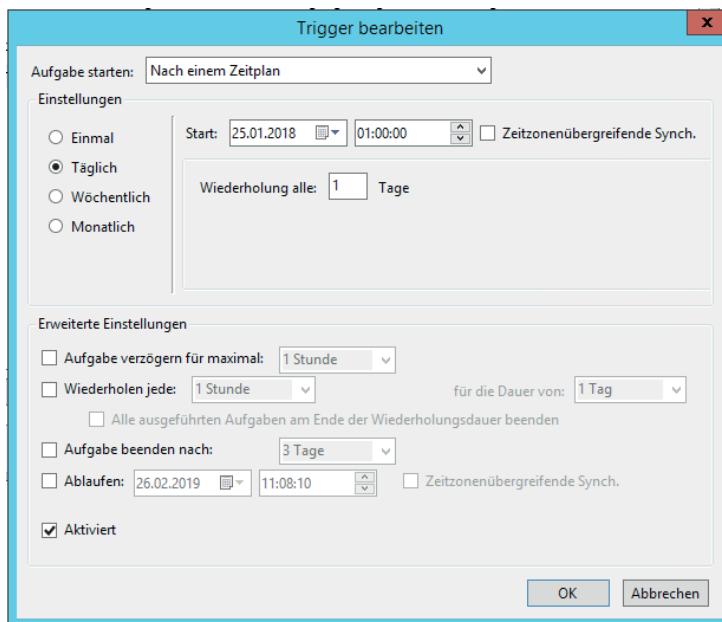


Abb. 35: Task-Trigger

Anschließend wird der Trigger erstellt. Hier werden die Ausführungszyklen definiert. Der Bot soll einmal täglich ausgeführt werden um 1 Uhr in der Früh. Für die Temperatur wurde in den „Erweiterten Einstellungen“ definiert, dass dieser auch jede Stunde läuft.

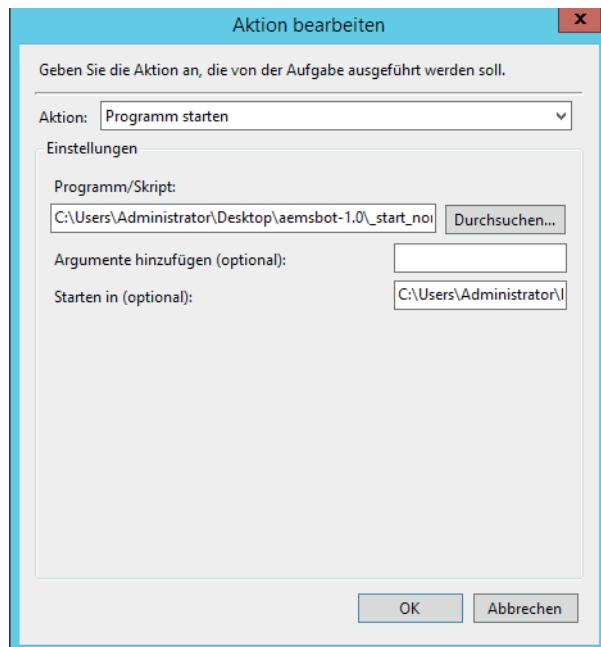


Abb. 36: Task-Ausführung

Auch die Tasks, welche ausgeführt werden sollen, müssen definiert werden. Hierzu muss bei „Aktion“ ausgewählt werden, dass ein Programm gestartet werden soll. Anschließend muss auch der Pfad zur auszuführenden Datei angegeben werden.

Zusätzlich muss auch noch der Pfad, in welchem diese .bat-Datei liegt extra angegeben werden.

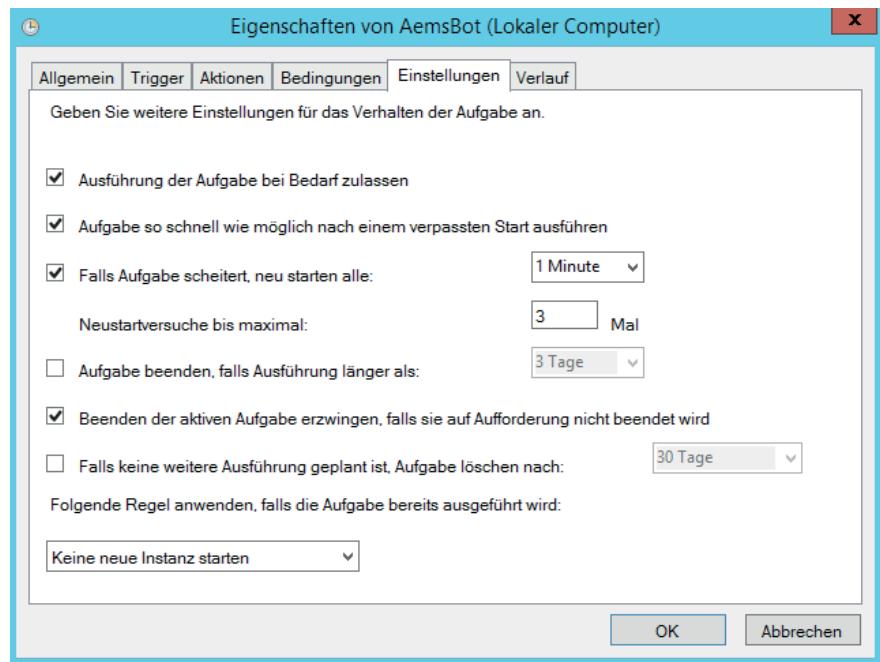


Abb. 37: Task-Eigenschaften

Zum Schluss müssen noch Grundeinstellungen gemacht werden. Zum Beispiel ob der Task bei einem Fehler neu gestartet werden soll, wie oft und in welchem Zeitabstand. Wenn der Task für die Ausführung zu lange braucht, wird eine aktive Aufgabe beendet.

### 3.2.8 Abfragen der Wetterdaten

Wird der Report-Bot ohne Argumente ausgeführt, so startet er im normalen Modus und sammelt Zählerdaten. Wird er hingegen mit dem Argument **-temp** gestartet, so werden nur Temperaturdaten ermittelt. Hierfür wird die API von OpenWeatherMap (<https://openweathermap.org/> [25.02.2018]) verwendet.

In der Datenbank kann für jeden Zähler auch eine geografische Position abgespeichert werden. Sollte eine solche Position für einen Zähler vorhanden sein, dann wird die Temperatur an dieser Position ermittelt.

#### 3.2.8.1 Aufrufen der API

Das Aufrufen der OpenWeahterMap API erfolgt über einen GET Request. Die erforderlichen Daten werden also in der URL übertragen. Es gibt verschiedene Möglichkeiten, die geografische Position anzugeben. Sollte der Ortsname der Position verfügbar sein, wird dieser verwendet. Ansonsten wird auf den Längen- und Breitengrad zurückgegriffen.

```
private double getTemperature(AemsMeter meter) throws Exception {
    String urlString = BASE_URL + meter.getLocation().getLocationAsQueryString();
    urlString += "&units=metric&APPID=";
    urlString += Main.config.get(BotConfiguration.API_KEY);

    URL url = new URL(urlString);
    HttpURLConnection connection = (HttpURLConnection) url.openConnection();

    if(connection.getResponseCode() == HttpURLConnection.HTTP_OK) {
        String rawData = readDataFromStream((InputStream) connection.getContent());
        JSONObject response = new JSONObject(rawData);
        JSONObject data = response.getJSONObject("main");
        return data.getDouble("temp");
    } else {
        return ABSOLUTE_ZERO;
    }
}
```

Abb. 38: Code zum Abrufen der Temperatur an der Position eines Zählers

### 3.3 Umsetzung Android-App

#### 3.3.1 Allgemein

Die Android-App ist dafür gemacht, die Benutzer/Benutzerinnen bei abweichendem Energieverbrauch darüber zu benachrichtigen. Weiters werden die zuvor bereits in der Website angelegten und für die App freigegebenen Statistiken angezeigt. Diese können auch direkt heruntergeladen werden. Entwickelt wurde die App im Androidstudio mit der Version 2.3.3. Die Kompatibilität der App wird ab der Androidversion 6.0 gewährleistet. Funktioniert aber auch mit früheren Versionen.

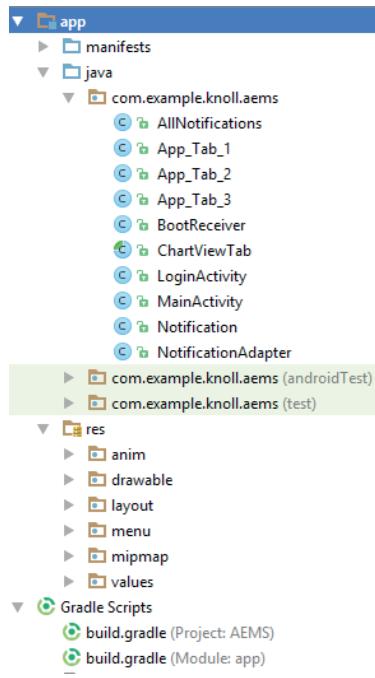


Abb. 39: Android-App Struktur

Der Aufbau der App-Struktur ist wie im obigen Bild zu sehen.

Im Manifest müssen alle verwendeten Activities und Permissions eingetragen werden. Permissions dienen dazu, der App nur die Rechte, welche sie auch wirklich braucht zu geben. Als Beispiel das Recht Daten auf dem Telefonspeicher abzulegen.

Im Ordner „Java“ sind alle Klassen und Activities zu finden. Hier wird die eigentliche Logik der App programmiert.

Im Verzeichnis „res“ befindet sich der grafische Teil der App. Unter Layouts werden die Designs der einzelnen Seiten und Fragments (Teilelemente) definiert.

Unter „menu“ werden Design und Inhalt der Menüs programmiert und unter values können weitere Dinge wie Farben, oder Styles definiert werden.

Gradle ist ein Build-System. Hier müssen alle Libraries, welche verwendet werden, eingetragen werden.

### 3.3.2 Main Activity

Die Main-Activity einer Android-App ist die erste Klasse, welche beim Ausführen der App gestartet wird. Darin wird die Hauptfunktionalität des Programms definiert. Der Aufbau der App, die Menüstruktur, wie auch der Aufruf weiterer Activities wird hier gemacht.

#### 3.3.2.1 Tabbed Activities

Für die Anzeige der Statistiken wird die Technologie der Tabbed Activities verwendet. Diese Technologie ermöglicht es in waagrechte Richtung zwischen den Statistiken hin und her zu wischen.

Hierfür müssen die einzelnen Statistik Tabs als eigene Activity-Klassen definiert und in der Main-Activity aufgerufen werden. Das heißt es müssen Instanzen der „ChartViewTabs“, also der einzelnen Statistik-Activities in der Main-Activity gemacht werden.

Weiters müssen ein „SectionPagerAdapter“ und ein „ViewPager“ angelegt werden. Diese werden für den Vorgang des Wechsels der Statistiken benötigt. Diese Tabs müssen anschließend auch ihren Activity Klassen zugewiesen werden.

```
private SectionsPagerAdapter mSectionsPagerAdapter;
private ViewPager mViewPager;

private ChartViewTab tab1;
private ChartViewTab tab2;
private ChartViewTab tab3;

tab1 = new App_Tab_1();
tab2 = new App_Tab_2();
tab3 = new App_Tab_3();
```

Abb. 40: Anlegen der Statistik Tabs

Als nächstes muss der App gesagt werden, welche Statistik bei einem Wischen als nächstes angezeigt werden soll und welche die aktuelle Statistik ist. Dies funktioniert mit einem sogenannten SectionPagerAdapter. Dieser wählt das benötigte Fragment (in unserem Fall die benötigte Statistik) und zeigt dieses an.

Im nächsten Schritt merkt sich der ViewPager das gerade aktuelle Fragment (aktuelle Statistik).

In getCount() wird festgelegt, wie viele verschiedene Tabs es geben darf.

```
public class SectionsPagerAdapter extends FragmentPagerAdapter {  
  
    public SectionsPagerAdapter(FragmentManager fm) {  
        super(fm);  
    }  
  
    @Override  
    public Fragment getItem(int position) {  
  
        switch (position){  
            case 0:  
                return tab1;  
            case 1:  
                return tab2;  
            case 2:  
                return tab3;  
            default:  
                return null;  
        }  
    }  
  
    public ChartViewTab getCurrentTab() {  
        return (ChartViewTab) getItem(mViewPager.getCurrentItem());  
    }  
  
    @Override  
    public int getCount() {  
        // Show 3 total pages.  
        return 3;  
    }  
}
```

Abb. 41: SectionPagerAdapter

<https://developer.android.com/training/implementing-navigation/lateral.html> [13.08.2017]

### 3.3.2.2 Download von Statistiken

Um die in den Tabs angelegten Statistiken herunterladen zu können, wird die zurzeit angezeigte Statistik benötigt. Diese haben wir uns wie unter 3.2.2.1 beschrieben in einem ViewPager gespeichert. Über diesen ViewPager wissen wir jetzt, welche Statistik heruntergeladen werden soll.

```

mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());

// Set up the ViewPager with the sections adapter.
mViewPager = (ViewPager) findViewById(R.id.container);
mViewPager.setAdapter(mSectionsPagerAdapter);

FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);

fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        ChartViewTab currentTab = mSectionsPagerAdapter.getCurrentTab();
        Snackbar.make(view, currentTab.getStatisticTitle() + " wird heruntergeladen!", Snackbar.LENGTH_LONG)
            .setAction("Action", null).show();

        switch (mViewPager.getCurrentItem()) {
            case 0:
                downloadStatistic(tab1);
                break;
            case 1:
                downloadStatistic(tab2);
                break;
            case 2:
                downloadStatistic(tab3);
                break;
        }
    }
});

```

Abb. 42: Selektion der Statistik für den Download

Zuerst wird ein SectionPagerAdapter angelegt. Dieser Adapter muss anschließend dem Pager zugewiesen werden.

Anschließend wird ein FloatingActionButton angelegt. Beim Klick auf diesen Button soll die entsprechende Statistik heruntergeladen werden. Wenn man auf diesen Button klickt, wird dessen onClick-Methode aufgerufen.

In der onClick-Methode wird zuerst der gerade ausgewählte Tab bestimmt. Anschließend wird mit dem Befehl „Snackbar.make()“ ein Balken am unteren Displayrand erzeugt, welcher anzeigt das eine bestimmte Statistik heruntergeladen wird.

Im Anschluss wird mit einem „switch“ festgelegt, welche Statistik heruntergeladen werden soll. Hier wird dann die downloadStatistic-Methode mit dem entsprechenden Statistik-Parameter aufgerufen.

```

private void downloadStatistic(ChartViewTab tab) {
    boolean saveToSd = false;
    boolean canSaveImage = false;
    Chart chart = tab.getChart();
    Bitmap image = chart.getChartBitmap();
    String filename = tab.getStatisticTitle();
    File filePath = null;

    if(saveToSd) {
        ActivityCompat.requestPermissions(this, new String[] {Manifest.permission.WRITE_EXTERNAL_STORAGE}, 1);
        filePath = new File(Environment.getExternalStorageDirectory(), "/AEMS/");
        filePath.mkdirs();

        File imageFile = new File(filePath, filename + ".jpg");
        OutputStream stream = null;
        try {
            stream = new FileOutputStream(imageFile);
            image.compress(Bitmap.CompressFormat.JPEG, 100, stream);
            stream.flush();
            stream.close();
        } catch(IOException e) {
            e.printStackTrace();
        }
    } else {
        if (hasStoragePermission()) {
            String url = MediaStore.Images.Media.insertImage(getApplicationContext(), image, filename, "Hello");
            if(url != null){
                canSaveImage = true;
            }
            else if(url == null) {
                canSaveImage = false;
            }
        } else {
            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                    12);
        }
    }
    doGenerateNotification(canSaveImage, filename);
}

```

Abb. 43: Statistik Download

In der Methode „downloadStatistic“ wird nun die Statistik heruntergeladen. Zuerst muss der App die Berechtigung gegeben werden, Daten auf den Smartphonespeicher zu speichern. Hierfür muss eine Permission für „WRITE\_EXTERNAL\_STORAGE“ gesetzt werden. Anschließend muss noch ein Pfad bestimmt werden, unterm welchem die Statistiken gespeichert werden sollen.

Dem imageFile werden der Pfad unter welchem das Bild gespeichert werden soll, der Dateiname und die Dateiendung mitübergeben.

Nun wird über einen FileStream die Statistik zu einem JPEG-Bild konvertiert.

Wenn der Benutzer/die Benutzerin der App die Berechtigung zum Speichern von Dateien auf den Smartphonespeicher erteilt hat, wird die Statistik unter dem angegebenen Pfad gespeichert.

Im Anschluss wird die Methode doGenerateNotification() aufgerufen. Diese bekommt die Attribute canSaveImage und den Dateinamen mit. Hier wird im Anschluss an das Speichern der Statistik eine Benachrichtigung erzeugt, ob das Speichern der Statistik erfolgreich verlaufen ist.

<https://stackoverflow.com/questions/649154/save-bitmap-to-location> [01.09.2017]

### 3.3.3 Statistik Tab Activity

#### 3.3.3.1 Allgemein

In den Statistic Tab Activities werden die Statistiken erstellt, welche in der Main Activity aufgerufen und in die entsprechenden Tabs eingefügt werden.

Hier gibt es zwei Arten von Statistiken. Die einfachen Charts, welche nur aus einer Statistik bestehen und die CombinedCharts, bei welchen mehrere Statistiken in einer Grafik kombiniert werden können.

Als Library wird MPAndroidChart verwendet.

<https://github.com/PhilJay/MPAndroidChart> [24.08.2017]

#### 3.3.3.2 Einfache Bar Charts

Einfache Charts bestehen aus nur einem Set von Daten, wie zum Beispiel dem Stromverbrauch.

```
public class App_Tab_1 extends ChartViewTab {  
  
    public App_Tab_1() {  
        super(R.layout.app_tab_1, R.id.chart1);  
    }  
  
    @Override  
    public void onCreateChart(Chart chart) {  
        List<BarEntry> entries = new ArrayList<BarEntry>();  
        entries.add(new BarEntry(1f, 2));  
        entries.add(new BarEntry(2f, 1));  
        entries.add(new BarEntry(4f, 15));  
        entries.add(new BarEntry(5f, 13));  
        entries.add(new BarEntry(7f, 4));  
        entries.add(new BarEntry(8f, 5));  
        entries.add(new BarEntry(10f, 7));  
        entries.add(new BarEntry(11f, 8));  
  
        BarDataSet dataSet = new BarDataSet(entries, "Stromverbrauch");  
        dataSet.setColor(Color.RED);  
        dataSet.setValueTextColor(Color.BLUE);  
  
        BarData barData = new BarData(dataSet);  
        chart.setData(barData);  
        chart.invalidate();  
    }  
  
    @Override  
    public String getStatisticTitle() {  
        return "Strom1";  
    }  
}
```

Abb. 44: Einfache Bar Charts

Bei einfachen Bar Charts müssen in einer Liste die Werte aller Balken angegeben werden. Hierfür wird der Liste ein neues Bar-Element hinzugefügt, welchem dann die Werte für die X-, und Y-Koordinaten gegeben werden.

Anschließend wird aus den Entries ein Data-Set mit einem Namen erzeugt. Es können auch noch weitere Eigenschaften, wie die Farbe der Bars vergeben werden. Mit chart.setData() wird der Inhalt für die Statistik gesetzt und mit chart.invalidate() wird die Statistik neu geladen.

### 3.3.3.3 Combined Charts

Für die Verwendung von mehreren Statistiken in einer Grafik ist es notwendig sogenannte Data-Sets zu erstellen. Mehrere Sets können zu einer Grafik zusammengestöpselt werden.

```
private BarData generateBarData() {  
  
    ArrayList<BarEntry> entries1 = new ArrayList<BarEntry>();  
    ArrayList<BarEntry> entries2 = new ArrayList<BarEntry>();  
  
    entries1.add(new BarEntry(lf, 2));  
    entries1.add(new BarEntry(lf, 6));  
    entries1.add(new BarEntry(lf, 4));  
  
    entries2.add(new BarEntry(lf, 2));  
    entries2.add(new BarEntry(lf, 6));  
    entries2.add(new BarEntry(lf, 5));  
  
    BarDataSet set1 = new BarDataSet(entries1, "Aktuelle Werte");  
    set1.setColor(Color.rgb(60, 220, 78));  
    set1.setValueTextColor(Color.rgb(60, 220, 78));  
    set1.setValueTextSize(8f);  
    set1.setAxisDependency(YAxis.AxisDependency.LEFT);  
  
    BarDataSet set2 = new BarDataSet(entries2, "Vorperiode");  
    set2.setColor(Color.rgb(61, 165, 255));  
    set2.setValueTextColor(Color.rgb(61, 165, 255));  
    set2.setValueTextSize(8f);  
  
    float groupSpace = 0.5f; // Space between Bar Groups  
    float barSpace = 0.1f;  
    float barWidth = 0.7f;  
  
    BarData barD = new BarData(set1, set2);  
    barD.setBarWidth(barWidth);  
  
    // Makes BarData object grouped  
    barD.groupBars(0, groupSpace, barSpace); // Start at x = 0
```

Abb. 45: Combined Chart - Bar Set

Für Charts vom selben Typ muss immer ein Set erstellt werden. Wie im obigen Code zu sehen, gibt es zwei Bar-Charts, welche zusammengefügt werden.

Es ist für jeden Chart eine Liste zu erstellen, welche mit den entsprechenden Daten befüllt werden muss.

Anschließend werden die Einstellungen für ein DataSet vorgenommen (Farbe der Balken, Ausrichtung der Legende, ...).

Zum Schluss muss ein BarData Object gemacht werden, welchem die beiden DataSets zugewiesen werden.

Mit dem Befehl barD.groupBars werden die Balken nach einem definierten Schema gruppiert.

```

private LineData generateLineData() {

    LineData lineD = new LineData();

    ArrayList<Entry> entries = new ArrayList<Entry>();

    entries.add(new Entry(1f, 21));
    entries.add(new Entry(3f, 23));
    entries.add(new Entry(5f, 24));

    LineDataSet dataSet = new LineDataSet(entries, "Außentemperatur");
    dataSet.setColor(Color.rgb(240, 140, 70));
    dataSet.setLineWidth(2.5f);
    dataSet.setCircleColor(Color.rgb(240, 238, 70));
    dataSet.setCircleRadius(5f);
    dataSet.setFillColor(Color.BLUE);
    dataSet.setMode(LineDataSet.Mode.CUBIC_BEZIER);
    dataSet.setDrawValues(false);
    dataSet.setValueTextSize(10f);
    dataSet.setValueTextColor(Color.BLACK);
    dataSet.setAxisDependency(YAxis.AxisDependency.RIGHT);

    lineD.addDataSet(dataSet);
}

```

Abb. 46: Combined Chart - Line Graph

Um in eine Grafik nicht nur den Energieverbrauch eines Zählertyps und dessen Werte der Vorperiode als Balkendiagramm einfließen zu lassen, kann auch noch ein Liniendiagramm für Anomalien miteingebunden werden.

Das System ist hier wieder das Selbe. Es muss eine Liste für alle Werte angelegt werden. Diese Liste wird dann einem LineDataSet-Objekt zugewiesen. Im Anschluss können wieder Einstellungen für den Diagramm-Typ gewählt werden. Zum Beispiel die Strichdicke, Farbe, Legende und ob die Werte direkt im Diagramm dabeistehen sollen.

```

CombinedData data = new CombinedData();

data.setData(generateBarData());
data.setData(generateLineData());

xAxis.setAxisMaximum(data.getXMax() + 0.2f);

chart.setData(data);
chart.invalidate();

```

Abb. 47: Combined Data

Im Hauptprogramm werden beide Methoden aufgerufen. Die des Balkendiagramms und die des Liniendiagramms. Diese verschiedenen Chart-Typen werden dann einem CombinedData-Objekt mitgegeben. Anschließend müssen die Daten dieses CombinedData-Objekts noch dem Diagramm selbst mitgegeben werden.

Mit dem Befehl chart.invalidate() wird die Statistik wieder neu geladen.

### 3.3.4 Login Activity

Für das Login wird ein von Google vorgefertigtes Template verwendet, welches jedoch sehr stark angepasst wurde. Beim Start der App wird man auf die Loginseite gelotzt. Auf dieser kann man sich mit den selben Zugangsdaten, wie auf der Website anmelden.

Die Zugangsdaten werden über SharedPreferences gespeichert, damit man sich bei der Verwendung der App nicht jedes mal neu anmelden muss.

```
public boolean validate() {
    boolean valid = true;

    String user = _inputUsername.getText().toString();
    String password = _passwordText.getText().toString();

    if (user.isEmpty()) {
        _inputUsername.setError("Geben Sie einen Benutzernamen ein");
        valid = false;
    } else {
        _inputUsername.setError(null);
    }

    if (password.isEmpty()) {
        _passwordText.setError("Geben Sie ein Passwort ein");
        valid = false;
    } else {
        _passwordText.setError(null);
    }

    return valid;
}
```

Abb. 48: Login Validate

Damit man überhaupt einen Anmeldeversuch machen kann, wird überprüft, ob der Benutzer/die Benutzerin überhaupt einen Benutzernamen und ein Passwort eingegeben hat. Wenn nicht, wird gar keine Login-Anfrage an den Server geschickt und man bekommt den Hinweis, Nutzerdaten einzugeben.

Da der Benutzername keinem festen Schema (z.B. E-Mail) entspricht ist eine Validierung auf den Inhalt der Textfelder nicht notwendig.

```
public void onLoginFailed() {
    Toast.makeText(getApplicationContext(), "Login fehlgeschlagen - Überprüfen Sie Ihre Logindaten", Toast.LENGTH_LONG).show();
    _loginButton.setEnabled(true);
}
```

Abb. 49: Login-Failed

Bei der Eingabe falscher Nutzerdaten wird man darauf aufmerksam gemacht, dass die Logindaten falsch eingegeben wurden.

```

public void login() {
    Log.d(TAG, "Login");

    if (!validate()) {
        onLoginFailed();
        return;
    }

    _loginButton.setEnabled(false);

    final ProgressDialog progressDialog = new ProgressDialog(LoginActivity.this,
        R.style.Theme_AppCompat_Light_Dialog);
    progressDialog.setIndeterminate(true);
    progressDialog.setMessage("Authentifizierung...");
    progressDialog.show();

    final String email = _inputUsername.getText().toString();
    final String password = _passwordText.getText().toString();

    //Login to AEMS
    BigDecimal key = null;
    try {
        DiffieHellmanProcedure.sendKeyInfos(new Socket(InetAddress.getByName("localhost"), 9950));
        key = KeyUtils.salt(new BigDecimal(new String(DiffieHellmanProcedure.confirmKey()))), email, password);
    } catch(Exception e) {
        e.printStackTrace();
    }

    BigInteger keyInt = key.unscaledValue();
    byte[] sharedSecretKey = keyInt.toByteArray();

```

Abb. 50: Login in AEMS

Beim Loginversuch wird zuerst die Methode validate() aufgerufen, welche überprüft, ob Nutzerdaten eingegeben wurden.

Anschließend erscheint am Display ein Progressdialog, welcher darüber informiert, dass gerade ein Anmeldeversuch gemacht wird.

Um die Zugangsdaten zu verschlüsseln, wird die selbst entwickelte DiffieHellman-Library verwendet. Hier werden der Nutzernname, das Passwort und ein zufällig generierter Salt dazu verwendet, die Sicherheit zu garantieren.

Anschließend wird der Key, welchen man erhält auf ein byte[] gecastet, um ihn für das Login verwenden zu können.

```

AemsAPI.setUrl("https://api.aems.at");

AemsLoginAction loginAction = new AemsLoginAction(EncryptionType.AES);
loginAction.setUsername(email);
loginAction.setPassword(password);

AemsResponse response = null;
try {
    response = AemsAPI.call0(loginAction, sharedSecretKey);
    httpCode = response.getResponseCode();
    responseText = response.getResponseText();
    responseDecryptedText = response.getDecryptedResponse();
} catch (IOException e) {
    e.printStackTrace();
}

new android.os.Handler().postDelayed(
    new Runnable() {
        public void run() {
            // On complete call either onLoginSuccess or onLoginFailed
            if(httpCode == 200){
                onLoginSuccess(email, password);
            }
            else{
                onLoginFailed();
            }
            progressDialog.dismiss();
        }
    }, 500);

```

Abb. 51: Loginprozess

Mit dem oben erhaltenen Key wird man dann beim AEMS-Dienst angemeldet. Diese Kommunikation geschieht über die eigens entwickelte aems-apilib.

Wenn als responseCode 200 zurück kommt, war das Login erfolgreich und die Methode onLoginSuccess() wird aufgerufen. Wenn der Responsecode nicht 200 ist, kommt man über die Methode onLoginFailed() zur Loginseite zurück und wird benachrichtigt, dass das Login nicht erfolgreich war.

```

public void onLoginSuccess(String email, String password) {
    _loginButton.setEnabled(true);

    //Save Logindata in SharedPreference
    sharedPreferences = getSharedPreferences(PREFERENCE_KEY, MODE_PRIVATE);
    String user = sharedPreferences.getString("EMAIL", null);
    String passw = sharedPreferences.getString("PASSWORD", null);

    if(user == null && passw == null){
        CheckBox checkBoxRememberMe = (CheckBox) findViewById(R.id.checkBoxRememberLogin);

        if(checkBoxRememberMe.isChecked()){
            sharedPreferences = getSharedPreferences(PREFERENCE_KEY, MODE_PRIVATE);
            sharedPreferences.edit().putString("EMAIL", email).commit();
            sharedPreferences.edit().putString("PASSWORD", password).commit();
            sharedPreferences.edit().putBoolean("REMEMBERLOGIN", true).commit();
        }
    }

    finish();
}

```

Abb. 52: onLoginSuccess()

Wenn der Loginvorgang erfolgreich war, wird onLoginSuccess() aufgerufen. Hier wird abgefragt, ob der Benutzer/die Benutzerin das Feld „Benutzerdaten merken“ angehakt hat. Wenn ja, werden die Benutzerdaten in einer SharedPreference gespeichert. Somit muss man sich nicht bei jedem Appstart neu anmelden.

In der Main-Activity wird beim Start der App überprüft, ob die Nutzerdaten in einer SharedPreference gespeichert wurden. Wenn ja, wird der Aufruf der Loginactivity übersprungen.

Außerdem gibt es im Menü die Möglichkeit sich abzumelden. Beim Klick auf abmelden, wird die SharedPreference gelöscht und man gelangt wieder zum Loginbildschirm.

<https://developer.android.com/studio/projects/templates.html> [03.10.2017]

<https://sourcey.com/beautiful-android-login-and-signup-screens-with-material-design/> [03.10.2018]

### 3.3.5 Menü

Im Menü gibt es die Möglichkeit sich abzumelden und sich alle Benachrichtigungen anzeigen zu lassen.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

    int id = item.getItemId();
    if (id == R.id.action_logout) {
        //Delete Logininformation
        sharedpreferences = this.getSharedPreferences(PREFERENCE_KEY, MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedpreferences.edit();
        editor.clear();
        editor.commit();

        Intent intent = new Intent(this, LoginActivity.class);
        startActivity(intent);
    }
    if(id == R.id.action_showNotifications){
        Intent intent = new Intent(this, AllNotifications.class);
        startActivity(intent);
    }
    return super.onOptionsItemSelected(item);
}
```

Abb. 53: Optionsmenü

Im Menü wird überprüft, welcher Eintrag ausgewählt wurde. Beim Klick auf das Element mit der Id action\_logout, wird die SharedPreference mit den Nutzerdaten gelöscht und man gelangt zum Loginbildschirm.

Beim Klick auf das Element mit der Id action\_showNotifications, wird eine Activity aufgerufen, in welcher alle Benachrichtigungen in einer Liste angezeigt werden. Beim Klick auf einen Listeneintrag, bekommt man eine Ansicht mit den Details einer Benachrichtigung (betroffene Zähler, Grund der Benachrichtigung, Anmerkung,...).

### 3.3.6 Background Task Notifications

Da man auch ohne geöffneter App benachrichtigt werden will, wenn der Energieverbrauch abweicht, musste ein Background Task entwickelt werden, welcher im Hintergrund, ab dem Start des Smartphones mitläuft und überprüft, ob es Benachrichtigungen gibt.

Hierfür musste im Manifest die Permission für RECEIVE\_BOOT\_COMPLETED gesetzt werden. Dies ist notwendig, da die App ansonsten nicht ab dem Start des Smartphones im Hintergrund laufen darf.

Für die Notifications wurde ein NotificationAdapter entwickelt.

Der Adapter ist für das Aussehen und den Inhalt der Notifications zuständig.

```
@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
  
    if(convertView == null){  
        convertView = View.inflate(context, R.layout.notification_list_style, null);  
    }  
  
    ImageView images = (ImageView) convertView.findViewById(R.id.imageView);  
    TextView title = (TextView) convertView.findViewById(R.id.textViewTitle);  
    TextView info = (TextView) convertView.findViewById(R.id.textViewInformation);  
    TextView notType = (TextView) convertView.findViewById(R.id.textViewNotificationType);  
  
    images.setImageResource(elemPicture.get(position));  
    title.setText(elemTitle.get(position));  
    info.setText(elemInfo.get(position));  
    notType.setText(elemType.get(position));  
  
    return convertView;  
}
```

Abb. 54: NotificationAdapter

Hier wird das AEMS Logo gesetzt, der Titel der Benachrichtigung bestimmt, die Zusatzinfo, welche mit angezeigt werden soll und der Benachrichtigungstyp.

Für die Kommunikation mit dem Server wurde der von Google angebotene Firebase-Dienst verwendet. Für jeden Client wird ein Token erzeugt. Dieser wird durch die Kommunikation mit der apilib an die Datenbank gesendet und dort gespeichert.

Für den Firebase-Dienst sind am Client 2 Klassen notwendig, ein IdService und ein Messaging Service.

<https://developer.android.com/guide/topics/ui/notifiers/notifications.html> [07.11.2017]

```

public class MyFirebaseInstanceIdService extends FirebaseInstanceIdService {

    private static final String TAG = "MyFirebaseIIDService";
    private static final String PREFERENCE_KEY = "AemsLoginPreferenceKey";
    SharedPreferences sharedpreferences;
    int errorCount = 0;

    @Override
    public void onTokenRefresh() {

        String refreshedToken = FirebaseInstanceId.getInstance().getToken();
        Log.d(TAG, "Refreshed token: " + refreshedToken);

        saveTokenInDatabase(refreshedToken);
    }

    private void saveTokenInDatabase(String refreshedToken) {

```

Abb. 55: MyFirebaseInstanceIdService

Im MyFirebaseInstanceIdService wird der Token erzeugt. dieser wird wenn die App neu installiert wird für das Gerät neu definiert.

In der Methode saveTokenInDatabase() wird der Token an die Datenbank geschickt und beim entsprechenden User/Userin gespeichert.

```

public class MyFirebaseMessagingService extends FirebaseMessagingService {

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {

        Intent intent = new Intent(this, AllNotifications.class);
        intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_ONE_SHOT);

        NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(this);
        notificationBuilder.setContentTitle(remoteMessage.getNotification().getTitle());
        notificationBuilder.setContentText(remoteMessage.getNotification().getBody());
        notificationBuilder.setAutoCancel(true);
        notificationBuilder.setSmallIcon(R.drawable.Logo_icon);
        notificationBuilder.setContentIntent(pendingIntent);
        NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(0, notificationBuilder.build());
    }
}

```

Abb. 56: MyFirebaseMessagingService

Im MyFirebaseMessagingService wird die grundsätzliche Notification definiert. Also welche Werte angezeigt werden sollen, oder welche Seite der App beim Klick auf die Notification geöffnet wird.

Im Manifest musste eingetragen werden, welches Standardlogo und welche Standardfarbe die Notification besitzt, wenn dies am Server nicht definiert wird. Weiters müssen die beiden Firebase Klassen als Service im Manifest registriert werden.

In der Datenbank wurde ein Trigger definiert, welcher bei einer neuen Notification Code ausführt, welcher die Benachrichtigung an den Firebase Dienst und somit ans Androidgerät sendet.  
<https://firebase.google.com/docs/cloud-messaging/android/client> [07.11.2017]

<https://firebase.google.com/docs/cloud-messaging/send-message> [07.11.2017]

## 3.4 Umsetzung WebUI

Für die Verwirklichung einer fundierten und benutzerdefinierten Anomalien-Erkennung wurde eigens eine interpretierende Script-Sprache entworfen. Diese Script-Sprache ermöglicht es dem Benutzer/ der Benutzerin Anomalien genau nach deren Bedürfnisse zu definieren.

Das Konzept der Anomalien-Erkennung besteht im Großen und Ganzen aus drei Bestandteilen nämlich der Script-Sprache selbst, dem Interpreter der Script-Sprache und der WebUI, welche eine grafische Benutzeroberfläche für die grafische Deklaration von Anomalien bereitstellt.

Als erstes wird das Konzept des WebUIs vorgestellt. Wie Eingangs schon erwähnt, ist die WebUI eine grafische Oberfläche, welche dazu dient Anomalien zu deklarieren.

Zähler	AT0000000000000000000000003333			
Sensor	Sensor 1			
<b>Script</b>				
	<pre>\$y := \$sensor of period from \$today until \$today + 1 \$avg := \$y : avg raise notice "Anomaly found (\$avg)" on \$avg &lt; 1024</pre>			
Wiederholungszeit [Minuten]	15			
<input type="button" value="Verlinken"/>				
Zähler	Sensor	Script	Script Fehler	
AT0000000000000000000000003333	Sensor 1	<pre>\$y := \$sensor of period from \$today until \$today + 1 \$avg := \$y : avg raise notice "Anomaly found (\$avg)" on \$avg &lt; 1024</pre>	keine	<input type="button" value="Löschen"/> <input type="button" value="Editieren"/>

Abb. 57: WebUI

In der oberen Abbildung sind die Bestandteile des WebUIs sichtbar. Der erste Schritt besteht darin, den Zähler und den Sensor auszuwählen. Es werden nur jene Sensoren und Zähler angezeigt, welche der Benutzer/ die Benutzerin auch sehen darf. Die Daten, über welche der Benutzer / die Benutzerin Zugangsrechte besitzt, werden über die AEMS-RestAPI bereitgestellt. Anschließend gibt man sein Script in der AEMS-Scripting-Language ein, welche für die Anomalien-Erkennung konsultiert wird und stellt ganz unten unter Wiederholungszeit die Zeit in Minuten ein, nach welcher das Script periodisch ablaufen soll. Das Wiederholungszeiteingabefeld ist mit einem javax.faces.Validator versehen, welcher nur Eingaben erlaubt, welche durch Fünfzehn ohne Rest teilbar sind. Im Bild gezeigten Beispiel wird das Script alle fünfzehn Minuten vom Interpreter ausgeführt. Die Eingabe wird mit einem Klick auf „Verlinken“ bestätigt und in die Datenbank als Anomalie eingetragen.

Im unteren Teil der Abbildung ist die Anzeige der bereits deklarierten Anomalien zu sehen. Dabei werden auch die vom Interpreter erkannten Script-Fehler, welche während der Exekution aufgetreten sind, angezeigt. Rechts neben der jeweiligen Anomalie sind zwei Schaltflächen vorzufinden. Der Erste ist mit „Löschen“ betitelt und der Zweite mit „Editieren“. Der „Löschen“ Schaltknopf ist zum Entfernen der Anomalie aus der Datenbank, was mitunter auch zur Folge hat, dass das Anomalie-Script nicht mehr vom Interpreter der AEMS-Scripting-Language exekutiert wird. Der „Editieren“ Schaltknopf ist zum Editieren der jeweiligen Anomalie. Sobald auf den Schaltknopf gedrückt wird, werden alle Parameter in die oberen Felder eingetragen.

Anschließen kann man diese bearbeiten und mit einem erneuten Klick auf „Verlinken“ werden die Änderungen in die Datenbank übernommen.

Wie zuvor schon mehrmals erwähnt, werden die Anomalien, welche mit Hilfe des WebUIs deklariert wurden in die Datenbank gespeichert. In nachfolgendem Bild ist ein Ausschnitt der Tabelle „Anomalien“ dargestellt:

<b>id</b>	<b>PK</b>	<b>numeric(16,0)</b>	<b>script</b>	<b>character varying(1500)</b>	<b>exec_intermediate_time</b>	<b>last_execution_time</b>	<b>timestamp without time zone</b>	<b>meter</b>	<b>character varying(150)</b>	<b>sensor</b>	<b>character varying(150)</b>	<b>script_errors</b>	<b>character varying(500)</b>
2			Sy :=	Sensor of period from Stoday until Stoday + 1	15	2018-02-11	17:36:27	AT000000000000000000000000003333	Sensor 1				
22			Shbb :=	Sensor of period from Stoday - 1 until Stoday	15	2018-02-22	12:16:22	7580140	AT000000000000000000000000005555	Sensor 2			

Abb. 58: AnomalieN Table

Die Tabelle besitzt alle notwendigen Spalten, um eine Anomalie zu beschreiben.

## Spalten:

- script: Das Skript, welches vom Interpreter ausgeführt wird
  - exec\_intermediate\_time: Die Zeit in Minuten, bis wann das Skript erneut ausgeführt wird
  - last\_execution: Diese Spalte wird für den Interpreter benötigt um festzustellen, ob es schon wieder an der Zeit ist, das Skript auszuführen. Dieser Wert wird nach jeder Exekution durch den Interpreter auf die Exekutionszeit gesetzt.
  - meter: Eine Referenz auf die Meters Tabelle. Diese Spalte beschreibt, welcher Zähler an der Anomalien-Erkennung beteiligt ist.
  - sensor: Eine Referenz auf die Meters Tabelle. Diese Spalte beschreibt, welcher Sensor an der Anomalien-Erkennung beteiligt ist.
  - script\_errors: Diese Spalte ist sehr essentiell, da es für den Endbenutzer/ die Endbenutzerin die einzige Möglichkeit der Fehlererkennung darstellt. Ohne diese Information könnte man keine Einsicht auf Skript-Fehler erhalten. Diese Spalte wird vom Interpreter befüllt, sobald Skript-Fehler während der Ausführung auftauchen.

### 3.4.1 Umsetzung Scripting-Language-Interpreter

Nachdem die Anomalien vom WebUI deklariert worden sind und in die Datenbank eingefügt wurden, kann der Interpreter sich die Anomalien über die AEMS-RestAPI holen. Doch bevor im Detail auf die Verarbeitung der Anomalien eingegangen wird, wird zuvor noch die grundlegende Funktionsweise des AEMS-Scripting-Language-Interpreters erklärt. Der Interpreter ist ein Java8-Programm, welches auf dem Server läuft. Alle fünfzehn Minuten werden die Anomalien neu von der Datenbank akquiriert und deren Skripts exekutiert, falls die Wiederholungszeit oder exec\_intermediate\_time das zulässt. Der Grund dafür, dass der Interpreter nur alle fünfzehn Minuten die Anomalien aktualisiert ist, da die kleinste Zeitscheibe welche die Stromzähler, Gaszähler, etc. darstellen, fünfzehn Minuten ist. Das heißt nur alle fünfzehn Minuten kommen neue Zählerwerte in der Datenbank an. Somit ist es nur logisch, die kleinste Einheit der Exekution auf fünfzehn Minuten zu fixieren.

Nun da das Grundkonzept des Interpreters erläutert wurde, geht es ein bisschen in die Tiefe. In nachfolgendem Bild ist ein Teil der Interpreters zu sehen, welcher dafür zuständig ist, die Anomalien aus der Datenbank zu holen und diese dann in Anomaly-Objekte zu verwandeln. Diese Anomaly-Objekte sind eine interne Repräsentation, welche zur einfacheren Verarbeitung der Anomalien dient. Die Daten werden mithilfe der AEMS-RestAPI akquiriert und sind AES verschlüsselt. Bevor der Datenaustausch stattfinden kann, muss ein Schlüsselaustausch passieren.

```
DiffieHellmanProcedure.sendKeyInfos(new Socket(InetAddress.getByName("127.0.0.1"), 9950));
key = KeyUtils.salt(new BigDecimal(new String(DiffieHellmanProcedure.confirmKey())), "master", pwd);
```

Abb. 59: Key-Exchange

```
private static void fetchAnomalies() {
    final StringBuilder BUILDER = new StringBuilder();
    try {
        Files.readAllLines(Paths.get("src/assets/anomaly_query.txt")).stream().forEach(x -> {
            if (!x.startsWith("#"))
                BUILDER.append(x).append("\n");
        });
        String raw = BUILDER.toString();

        HttpURLConnection con = (HttpURLConnection) new URL(REST_ADDRESS + "encryption=AES&action=QUERY&user=215&data="
                + Base64.getURLEncoder().encodeToString(Encrypter.requestEncryption(Main.key, raw.getBytes()))).openConnection();
        con.setRequestMethod("POST");
        con.setDoOutput(true);
        con.setDoInput(true);
        BufferedReader reader = new BufferedReader(new InputStreamReader(con.getInputStream()));

        StringBuilder builder = new StringBuilder();
        for (String line = reader.readLine(); line != null; line = reader.readLine()) {
            builder.append(line);
        }

        String response = new String(Decrypter.requestDecryption(key, Base64.getUrlDecoder().decode(builder.toString().getBytes())));

        JSONObject root = new JSONObject(response);
        JSONArray array = root.getJSONArray("anomalies");
        for (int i = 0; i < array.length(); i++) {
            JSONObject object = array.getJSONObject(i);
            Anomaly a = new Anomaly();
            System.out.println(object);
            a.meter = object.getJSONObject("meter").getString("id");
            a.sensor = object.getJSONObject("sensor").getString("id");
            a.execIntermediateTime = object.getInt("exec_intermediate_time");
            a.lastExecution = format.parse(object.getString("last_execution"));
            a.script = object.getString("script");
            a.id = object.getInt("id");
            ANOMALIES.add(a);
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Abb. 60: Funktion fetchAnomalies

Im nächsten Bild ist der Algorithmus abgebildet, welcher für die periodische Exekution der Skripts verantwortlich ist. Wie im Bild zu erkennen ist, wird vor jedem Durchgang `fetchAnomalies` aufgerufen, was die Methode ist, welche oben näher beschrieben wurde. Anschließend wird in `startTime` die Startzeit vermerkt, bei welcher der Durchgang gestartet wurde. Dies ist wichtig für später, da das für die Berechnung, der noch zu wartenden Zeit ausschlaggebend ist. Um das mit der noch zu wartenden Zeit besser zu verstehen, hier ein kleines Beispiel: Die Exekution aller Skripts benötigt insgesamt sieben Minuten. Damit nun die Skripts in einem fünfzehn Minuten Zyklus ablaufen, muss die Differenz zwischen der Dauer der Exekution und der fünfzehn Minuten errechnet werden. Diese Differenz ist dann die Wartezeit. Nach dieser Wartezeit beginnt der nächste Durchgang. Am Ende jedes Durchgangs wird `updateAnomalies` aufgerufen, welche die Exekutionszeit der jeweiligen Anomalien aktualisiert. Wenn ein Skript-Fehler auftritt, wird die Methode `uploadScriptError` aufgerufen. Diese schreibt dann in die dafür vorgesehene Spalte `script_errors` von der Anomalie in der Datenbank den aufgetretenen Fehler. In der ganz innersten Schleife des Algorithmus, wird dem Parser Zeile für Zeile des Skripts übergeben. Dies ist notwendig, da der Parser nach dem **One-Statement-Per-Line Prinzip** funktioniert. Dies bringt Vorteile mit sich, zum Beispiel das kein besonderes Trennzeichen, wie ein Strichpunkt erforderlich ist, um das Statement abzugrenzen. Des Weiteren sorgt dieses Prinzip für eine bessere Lesbarkeit, da die Statements eher einfach gehalten werden.

```

while(true) {
    fetchAnomalies();
    long startTime = System.currentTimeMillis();
    Logger.logDebug("start of iteration");
    for(Anomaly anomaly : ANOMALIES) {
        long minutes = (System.currentTimeMillis() - anomaly.lastExecution.getTime()) / 1000 / 60;
        if(minutes < anomaly.execIntermediateTime)
            continue; // script should not be executed in this cycle
        anomaly.changed = true;
        anomaly.lastExecution = new Date(System.currentTimeMillis());
        try {
            System.out.println(anomaly.script);
            Tokenizer tokenizer = new Tokenizer(anomaly.script);
            Parser parser = new Parser(anomaly.meter, anomaly.sensor);
            Token token = null;
            ArrayList<Token> tokens = new ArrayList<>();
            while(!token = tokenizer.nextToken()).getType().is(TokenTypes.EOF)) {
                if(token.getType().is(TokenTypes.NEW_LINE)) {
                    if(tokens.size() == 0)
                        continue;

                    parser.parse(tokens.toArray(new Token[tokens.size()]));
                    tokens.clear();
                    continue;
                }
                tokens.add(token);
            }
            if(!tokens.isEmpty())
                parser.parse(tokens.toArray(new Token[tokens.size()]));
        } catch(Exception e) {
            e.printStackTrace();
            uploadScriptError(anomaly, e.getMessage());
        }
    }
    Logger.logDebug("end of iteration");
    updateAnomalies();
    try {
        //1000 * 60 * 15
        long wait = 1000 * 60 * 15 - (System.currentTimeMillis() - startTime); // 15 minutes cycle
        Thread.sleep(wait < 0 ? 0 : wait);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

```

Abb. 61: Scripting-Language-Interpreter Main-Loop

Nachdem die Zeile dem Parser übergeben wurde, muss überprüft werden welches Statement diese Zeile repräsentiert. Dazu wurden die Patterns für die Statements wie folgt definiert:

```

Object[] DEF_VARIABLE = toArray(TokenType.VARIABLE, TokenType.COLON, TokenType.EQUALS, TokenType.ANY_END);
Object[] PERIODIFY = toArray(TokenType.VARIABLE, "of", "period", "from", TokenType.ANY_IN_BETWEEN, "until", TokenType.ANY_END);
Object[] RAISE_NOTICE_SIMPLE = toArray("raise", "notice", TokenType.STRING, "on", TokenType.ANY_END);
Object[] FUNCTION_APPLICATION = toArray(TokenType.VARIABLE, TokenType.COLON, TokenType.WORD);
Object[] EXCEPTION_ON = toArray("except", "on", TokenType.ANY_END);
Object[] IF_ON = toArray("if", "on", TokenType.ANY_END);

```

Abb. 62: Statement Deklarationen

Die Klasse TokenType beinhaltet jede Art von Token, welche der Tokenizer als ein gültiges Token erkennt. Jedes Statement besteht aus einer Aneinanderreihung von Tokens, welche wie im oberen Bild gezeigt als Object[] definiert werden. Es gibt auch ein paar spezielle Tokens, welche sogenannte Auxilary-Tokens sind, nämlich ANY\_IN\_BETWEEN und ANY\_END. Das ANY\_IN\_BETWEEN Token ignoriert solange alle Tokens dazwischen, bis das nächste Token in der Statement-Definition (siehe Abbildung 62) gefunden wurde. Das ANY\_END Token ignoriert alles was nach diesem Token folgt.

```

public static boolean is(Object[] stm, Token[] input) {
    if(input.length < stm.length)
        return false;

    int stmIndex = 0;
    for(int i = 0; i < input.length; i++, stmIndex++) {
        Token token = input[i];
        if(stm[stmIndex] instanceof String) {
            if(!token.getRawToken().equals(stm[stmIndex]))
                return false;
        } else if(stm[stmIndex] instanceof TokenTypes) {
            TokenTypes type = (TokenTypes) stm[stmIndex];
            if(type.is(TokenTypes.ANY_END)) {
                return true;
            } else if(type.is(TokenTypes.ANY_IN_BETWEEN)) {
                stmIndex++;
                for(;;) {
                    if(i >= input.length - 1)
                        return false;
                    if(input[i++].getRawToken().equals(stm[stmIndex]))
                        break;
                }
                i--;
            } else if(!token.getType().is(type))
                return false;
        }
    }
    return true;
}

```

Abb. 63: Statement Detections Algorithmus

In der oberen Abbildung ist der Algorithmus zum Erkennen der Statements abgebildet. Die Methode erhält als Eingangsparameter zum einen die Statement-Definition und zum anderen die Zeile, welche dem Parser als Input übergeben wurden.

**Um numerische Ausdrücke auswerten zu können, wurde ein Algorithmus von Jack W. Crenshaw aus Compiler Building Tutorial vom 10. Oktober 2012 konzeptionell aufgegriffen.**

```

private Token evaluate() {
    Token f = product();
    for(;;) {
        if(next(TokenTypes.PLUS))
            f = add(f);
        else if(next(TokenTypes_MINUS))
            f = sub(f);
        else
            return f;
    }
}

public Token product() {
    Token result = factor();
    for(;;) {
        if(next(TokenTypes_MUL))
            result = mul(result);
        else if (next(TokenTypes_DIV))
            result = div(result);
        else
            return result;
    }
}

public Token factor() {
    if(next(TokenTypes.OPENING_PARENTHES)) {
        evaluate();
        next(TokenTypes.CLOSING_PARENTHES);
    }
    return next();
}

```

Abb. 64: Numerischer Evaluator

In den jeweiligen Methoden wie add, mul, sub und div werden die Datentypen für die jeweiligen mathematischen Operationen berücksichtigt. Das heißt, dass Daten anders als numerische Werte addiert, subtrahiert, multipliziert oder dividiert werden.

```

-----+
public void evaluate() {
    boolean tempResult = false;
    Token[] tempInput;
    int loopAvoidance = 0;

    loops : {
        for (;;) {
            ArrayList<Token[]> temp = split(input, "or");
            //System.out.println(Arrays.toString(temp.get(0)));

            if (temp.get(1).length == 0) {
                tempResult = true;
                tempInput = input;
                for (;;) {
                    // System.out.println("~~~" + Arrays.toString(input));
                    temp = split(tempInput, "and");

                    if(!tempResult) {
                        tempInput = temp.get(1);
                        continue;
                    }

                    // System.out.println(Arrays.toString(temp.get(0)));
                    ArrayList<Token[]> expr = splitAtComparisonSign(temp.get(0));
                    NumericalExpressionParser leftSide = new NumericalExpressionParser(expr.get(0), symbolTable);
                    NumericalExpressionParser rightSide = new NumericalExpressionParser(expr.get(1), symbolTable);
                    boolean leftSideIsAlist = false;
                    ArrayList<Float> values = null;
                    if(leftSide.getResult() == null && expr.get(0).length == 1) {
                        Token token = expr.get(0)[0];
                        if(token.getType().is(TokenTypes.VARIABLE)) {
                            SymbolTableEntry var = symbolTable.getSymbol(token.getRawToken());
                            if(var.getType().is(DataTypes.LIST)) {
                                // left side of the expression is a list
                                leftSideIsAlist = true;
                                values = var.getValue(ArrayList.class);
                                if(!rightSide.getDataType().is(DataTypes.NUMBER))
                                    Logger.logError("list can only be compared to a numerical expression!");
                            }
                        } else {
                            Logger.logError("left side of expression could not be resolved!");
                        }
                    } else if (!(leftSide.getDataType().is(DataTypes.NUMBER) && rightSide.getDataType().is(DataTypes.NUMBER))) {
                        // one of the both did not resolve to a number
                        Logger.logError("can only compare numbers!!!!");
                    }

                    float leftSideValue = leftSideIsAlist ? 0f : (Float) leftSide.getResult();
                    float rightSideValue = (Float) rightSide.getResult();

                    switch (getComparisonSign(temp.get(0)).getType()) {
                        case LEFT_ARROW:
                            if(leftSideIsAlist) {
                                loop: {
                                    for(float f : values) {
                                        if(f < rightSideValue) {

```

Abb. 65: Teil des Algorithmus des booleschen Expression-Parsers

In der Abbildung 65 ist ein Teil des Algorithmus für die Evaluation von booleschen Ausdrücken abgebildet. Anders als der numerische Parser ist dieser Parser zu hundert Prozent selbst programmiert. Da der AND Operator stärker bindet als der OR Operator muss zuerst der Token-Input beim OR Operator gesplittet werden und dann beim AND Operator. Nachdem bei einem AND gesplittet wurde, wird nun auch beim Vergleichsoperator gesplittet. Danach wird die linke und die rechte Seite dem numerischen Parser übergeben. Falls das Ergebnis für die linke Seite nicht definiert sein sollte, könnte es sich um eine Liste handeln, falls nicht, wird ein Fehler geworfen. Nachdem die linke und die rechte Seite ausgewertet wurden und einen Wert haben, werden über ein switch-Statement die entsprechenden Funktionen angewandt. Die Auswertung des booleschen Ausdrucks ist erst dann zu Ende, wenn entweder ein Fehler auftritt oder keine ORs und ANDs mehr gefunden werden können.

### **3.4.1.1 AEMS-Scripting-Language Syntax**

Die Syntax besteht im Grundlegenden aus sechs Statement-Typen:

- DEF\_STATEMENT
- PERIODIFY
- RAISE\_NOTICE\_SIMPLE
- EXCEPTION\_ON
- IF\_ON
- FUNCTION\_APPLICATION

### **3.4.1.2 DEF\_STATEMENT**

Das DEF\_STATEMENT ist das DEFINE-STATEMENT. Es ist das Grundstatement, welches eine Variablen Deklaration beschreibt. Der Aufbau dieses Statements ist wie folgt:

`$<var_name> := <any>`

Auf der linken Seite des Ausdrucks befindet sich die Variable, also `$<var_name>`, welcher man den Wert auf der rechten Seite zuweisen möchte, also `<any>`, wobei der Variablenname nur aus kleinen und großen Buchstaben von A bis Z bestehen darf. `<any>` stellt eine beliebige Reihe von Tokens dar.

Jedes DEFINE-STATEMENT hat einen Eintrag in den SymbolTable zur Folge, welcher für die Verwaltung von „Symbolen“, also in diesem Fall sind es nur Variablen, verantwortlich ist. Über den SymbolTable kann der aktuelle Wert der Variable abgerufen werden. Variablen werden aber als immutable angesehen, was heißt, dass sie nur einmal einen Wert zugewiesen bekommen können. Sie sind also Konstanten.

### **3.4.1.3 PERIODIFY**

Das PERIODIFY-STATEMENT ist ein DEFINE-STATEMENT. Nachdem ein DEFINE-STATEMENT detektiert wurde, wird überprüft, ob es sich nicht um ein PERIODIFY-STATEMENT handelt. Der Aufbau des Statements ist wie folgt:

`$<var_name> := ($meter|$sensor) of period from <from_time> until <to_time>`

Gewisse Werte lassen sich periodisieren. Diese Werte sind in den zwei Variablen `$meter` und `$sensor` beinhaltet. Diese zwei Variablen gehören zu den vordefinierten Variablen der Script-Sprache, auf welche später noch genauer eingegangen wird. Charakteristisch für dieses Statement ist die Periodisierung des Wertes, welcher vor `of period from` steht. Sowohl `<from_time>` als auch `<to_time>` sind Platzhalter für Datumswerte oder Ausdrücke, welche in Datumswerten resultieren.

In der Script-Sprache gibt es nur eine Möglichkeit Datumswerte zu spezifizieren und zwar in dem man von dem Referenzwert \$today ausgeht. Dies ist eine weitere vordefinierte Variable, welche das Datum des aktuellen Tages gespeichert hat. Somit wäre ein mögliches Statement:

```
$values := $meter of period from $today - $week * 2 until $today
```

Beim oben gezeigten Statement würde der Interpreter die Zählerwerte innerhalb der zwei Datumsgrenzen und zwar von \$today - \$week \* 2 (vom heutigen Datum subtrahiere zwei Wochen) bis heute über die AEMS-RestAPI abfragen und diese dann als Liste in \$values abspeichern. Alle Werte, welche innerhalb dieser Periode liegen befinden sich nun in \$values. Diese Variable hat nun den Datentyp LIST.

#### 3.4.1.4 RAISE\_NOTICE\_SIMPLE

Dieses Statement ist eines der Kernelemente der Scripting-Language. Der Aufbau sieht wie folgt aus:

```
raise notice "<msg>" on <any_conditions>
```

Wenn die Bedingungen, welche angegeben wurden sich auf true auflösen, dann wird ein Eintrag in der Datenbank getätigkt, dass eine Anomalie entdeckt wurde. Die Nachricht, welche in die Datenbank gespeichert wird, steht unter Hochkomma. Am nachfolgenden Beispiel ist zu erkennen, dass die Nachricht auch dynamisch zusammengestellt werden kann:

```
$x := $meter of period from $today - 1 until $today
```

```
$avg := x : avg
```

```
raise notice "Anomaly found! Exceeded average consumption of {$avg}" on $meter > $avg
```

Mit den geschweiften Klammern ist es möglich Variablenwerte in die Zeichenkette zu integrieren. Die Bedingungen sind Boolesche-Ausdrücke (siehe Boolesche Ausdrücke).

#### 3.4.1.5 EXCEPTION\_ON und IF\_ON

Die EXCEPT-ON und IF-ON Klauseln werden an das RAISE-NOTICE-SIMPLE-STATMENT angehängt. Wenn man möchte, dass eine Notice nur an bestimmten Tagen berücksichtigt wird, kann man dies mit diesen Klauseln tun, zum Beispiel:

```
raise notice "Anomaly found!" on $meter > $avg if on $monday, $tuesday
```

```
raise notice "Anomaly found!" on $meter > $avg except on $monday, $tuesday
```

Die IF-ON Klausel bewirkt, dass eine Notice nur an Montagen und Dienstagen erhoben wird, wobei die EXCEPT-ON Klausel bewirkt, dass seine Notice nur dann erhoben wird, wenn der aktuelle Tag weder ein Montag noch ein Dienstag ist.

### **3.4.1.6 FUNCTION\_APPLICATION**

Wie der Name schon verrät handelt es sich hier um die Anwendung einer Funktion, genauer gesagt um eine Aggregatsfunktion. Vier Funktionen stehen zur Verfügung, nämlich MIN, MAX, AVG, MEAN.

MIN gibt aus einer Liste von periodisierten Werten den kleinsten Wert zurück und MAX den Größten. AVG berechnet das arithmetische Mittel, wohingegen MEAN den Median berechnet. In der Anwendung würde das Ganze so aussehen:

```
$values := $meter of period from $today - $week * 2 until $today
$avg := $values : avg
$mean := $values : mean
$min := $values : min
$max := $values : max
```

Die Syntax des FUNCTION-APPLICATION-STATEMENTS wurde aus dem DEFINE-STATEMENT abgeleitet. Somit ist der Aufbau:

```
$<var_name> := $<var_name> : <function_name>
```

### **3.4.1.7 Vordefinierte Variablen**

Wie zuvor schon angekündigt enthält die Script-Sprache unter anderem auch vordefinierte Variablen. Die zwei wichtigsten Variablen im gesamten Script sind \$meter und \$sensor (beide vom Type NUMBER), wobei \$meter den aktuellen Wert des Zählers und \$sensor den aktuellen Wert des Sensors beinhaltet. Eine weitere vordefinierte Variable ist \$today, diese ist vom Typ DATE und beinhaltet das aktuelle Datum. Weitere Variablen sind \$monday, \$tuesday, \$wednesday, \$thursday, \$friday, \$saturday, \$sunday sowie \$day, \$week, \$month, \$year. Alle diese Variablen sind vom Typ NUMBER.

### **3.4.1.8 Datentypen**

Wie fast jede Script-Sprache hat auch diese Script-Sprache eine Einteilung der Daten in sogenannte Datentypen. Diese Datentypen werden jedoch nur intern für die Evaluation der Statements verwendet und müssen nicht von dem Benutzer/ der Benutzerin deklariert werden. In dieser Script-Sprache wird der Datentyp anhand des Ausdruckes abgeleitet. Es wird zwischen drei Datentypen unterschieden, nämlich DATE, NUMBER und LIST. Der Datentyp DATE stellt unschwer zu erkennen einen Datumswert dar und der Datentyp NUMBER einen numerischen Wert. Der Datentyp LIST deutet dem Interpreter an, dass diese Variable mehr als nur einen Wert repräsentiert und somit auch Aggregatsfunktionen (MIN, MAX, AVG, MEAN) sowie spezielle List-Operationen vorgenommen werden können. Spezielle List-Operationen sind zum Beispiel das Vergleichen einer Liste mit einem numerischen Wert: \$list < 5. In diesem Beispiel ist \$list eine Liste von Werten. Der Vergleich mit der numerischen Zahl fünf bedeutet, sobald eine Zahl in der Liste kleiner ist als fünf, wird die Bedingung in true resultieren.

### **3.4.1.9 Ausdrücke**

Es gibt zwei Arten von Ausdrücken, welche evaluierbar sind, nämlich numerische Ausdrücke und boolesche Ausdrücke.

### **3.4.1.10 Numerische Ausdrücke**

Wie in den meisten Script-Sprachen ist es auch in dieser Script-Sprache möglich numerische Ausdrücke zu deklarieren, zum Beispiel:  $5 + 3 * 7 + 6$ . Dies funktioniert genauso wie mit jedem handelsüblichen Taschenrechner, sprich mit Operator-Precedence (Punkt vor Strich). Numerische Ausdrücke können nicht nur Zahlen beinhalten, sondern auch Variablen, welche sich in Zahlen auflösen. Es gibt auch eine Ausnahme bei dieser Regel. Die Variable `$today` kann auch in einem numerischen Ausdruck vorkommen, sie muss jedoch den Start des Ausdrucks bilden, d. h. `$today + $week` ist ein valider Ausdruck, wobei `$week + $today` ein ungültiger Ausdruck ist. Das beruht auf dem Konzept des Interpreters, welcher als Regel definiert bekommen hat, dass jener Datentyp, welcher am Beginn eines Ausdruckes steht, auch der Datentyp ist, in welchen der Ausdruck resultiert. Dies ist insbesondere wichtig, da man mit Daten anders rechnen muss, als mit Zahlen.

### **3.4.1.11 Boolesche Ausdrücke**

Boolesche Ausdrücke sind Ausdrücke, welche eine wahre oder eine falsche Aussage als Ergebnis haben. Ein beispielhafter boolescher Ausdruck ist: `$list < 5 + 3 * 5 and $x > $y or $x < $z`. Das Konzept birgt wieder eine Operator-Precedence. Das AND bindet stärker als das OR. Es können so viele ANDs oder ORs verwendet werden, wie man möchte, solange der Ausdruck in einer wahren oder einer falschen Aussage resultieren kann. Bei Variablen vom Datentyp LIST haben Vergleichsoperatoren einen besonderen Effekt. Es ist möglich eine Liste mit dem `>` oder `<` Operator mit einer numerischen Konstante oder Variable zu vergleichen. Dabei wird jeder Wert verglichen und kontrolliert, ob dieser entweder kleiner oder größer als der angeführte numerische Wert ist. Wie man auch an dem angeführten Beispiel erkennen kann, ist der numerische Parser ein Teil des booleschen Parsers. Es können also numerische Ausdrücke in boolesche Ausdrücke geschachtelt werden.

## 3.5 Umsetzung REST-Service

Die AEMS-RestAPI wurde mit Java umgesetzt und ist Teil der Server Applikation. Im grundlegenden bestehen die Aufgaben der Rest-API in einer Schnittstelle für Abfragen für die Daten der Datenbank und einer Schnittstelle zum Ändern der Daten in der Datenbank. Die Änderungen belaufen sich auf Einfüge-, Lösch-, und Update Operationen.

### 3.5.1 Authentifizierung

Bevor eine Einfüge-, Lösch- oder Update Operation sowie eine Abfrage via GraphQL getätigter werden kann, wird vom Server überprüft, ob der Client, welcher die Operation tätigen möchte, auch dazu berechtigt ist. Dies geschieht allerdings nur bei der Verschlüsselung über SSL, da bei AES sowieso nur jener Client die Nachricht entschlüsseln kann, welcher den richtigen Schlüssel besitzt.

```
private boolean doLogin(String username, String authStr, String salt, String encryption, HttpServletRequest req, HttpServletResponse resp, boolean returnId) throws IOException {
    //JSONObject root = new JSONObject(query);
    String username = root.getString("user");
    String authStr = root.getString("auth_str");
    String salt = root.getString("salt");

    if(!isHashEqual(username, authStr, salt, resp))
        return false;

    // certainty that the username exists and ...
    // ... certainty that the credentials are valid

    try {
        if(!NUMBER_PATTERN.matcher(username).find()) { // conversion from username to user id
            ArrayList<String> projection = new ArrayList<>();
            projection.add(new String[]{ AEMSDatabase.Users.ID });
            HashMap<String, String> selection = new HashMap<>();
            selection.put(AEMSDatabase.Users.USERNAME, username);
            ResultSet set = DatabaseConnectionManager.getDatabaseConnection().select("aems", AEMSDatabase.USERS, projection, selection);
            username = set.getString(0, 0);
        }

        IPtoID.registerIPtoIDMapping(req.getRemoteAddr(), username);

        if(returnId) {
            String response = "{ id : " + username + " }";
            response = getEncryptedResponse(response, encryption, username, req, resp);
            response = Base64.getUrlEncoder().encodeToString(response.getBytes());

            resp.getWriter().write(response);
            resp.getWriter().flush();
        }
    } catch (SQLException ex) {
        resp.getWriter().write(encodeBase64("{ error : \"Invalid credentials!\" }"));
        resp.getWriter().flush();
        return false;
    }
    return true;
}
```

Abb. 66: Überprüfung der Autorität

Im oberen Bild ist der Algorithmus für die Authentifizierung via SSL zu sehen. Die Funktion `isHashEqual` prüft ob die beiden Authentication-Strings gleich sind. Der Authentication-String wird von dem Server aus Benutzername und Passwort zusammengebaut und danach mit dem Authentication-String, welcher vom Client mitgesendet wird, verglichen. Stimmen diese nicht überein, erhält der Client eine Fehlermeldung. Stimmen diese überein, so findet die geforderte Operation statt.

Bei der AES Verschlüsselung muss zuvor ein Schlüsselaustausch mit dem Server stattfinden. Dies passiert über den unten dargestellten Code. Dieser Thread wartet darauf, dass Clients einen Schlüsselaustausch anfragen. Ist der Schlüssel ausgetauscht worden, so wird der Schlüssel im AESKeyManager zwischengespeichert, sodass nicht bei jeder Abfrage ein neuer Schlüssel ausgetauscht werden muss.

```

new Thread(new Runnable() {
    public void run() {
        byte[] key;
        try {
            key = DiffieHellmanProcedure.receiveKeyInfos();
            AESKeyManager.addKey(req.getLocalAddr(), key);
        } catch (IOException ex) {
            Logger.getLogger(AAA.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}).start();

```

Abb. 67: AEMS Key Service Thread

### 3.5.2 Abfrage von Daten

Die Abfrage von Daten aus der Datenbank wird durch die AEMS-Rest API gewährleistet. Im Wesentlichen sind alle Abfragen GraphQL-Abfragen. GraphQL ist eine Abfrage Sprache, welche von Facebook entwickelt wurde. Damit die Daten mithilfe von GraphQL abgefragt werden können, musste zuerst GraphQL auf dem Server so eingerichtet werden, dass Abfragen so getätigter werden konnten, wie es im Rahmen dieser Diplomarbeit notwendig war.

<http://graphql-java.readthedocs.io/en/v7/> [03-03-2018]

```

public class Anomaly extends GraphQLObjectType {
    private static Anomaly instance;

    private static final GraphQLFieldDefinition ID = Query.getFieldDefinition("id", AEMSDatabase.ANALOGIES, AEMSDatabase.Anomalies.ID, Scalars.GraphQLInt);
    private static final GraphQLFieldDefinition METER = Query.getFieldDefinition("meter", Meter.getInstance());
    private static final GraphQLFieldDefinition SENSOR = Query.getFieldDefinition("sensor", Meter.getInstance());
    private static final GraphQLFieldDefinition SCRIPT = Query.getFieldDefinition("script", AEMSDatabase.ANALOGIES, AEMSDatabase.Anomalies.SCRIPT, Scalars.GraphQLString);
    private static final GraphQLFieldDefinition EXEC_INTERMEDIATE_TIME = Query.getFieldDefinition("exec_intermediate_time", AEMSDatabase.ANALOGIES, AEMSDatabase.Anomalies.EXEC_INTERMEDIATE_TIME, Scalars.GraphQLFloat);
    private static final GraphQLFieldDefinition LAST_EXECUTION = Query.getFieldDefinition("last_execution", AEMSDatabase.ANALOGIES, AEMSDatabase.Anomalies.LAST_EXECUTION, Scalars.GraphQLString);
    private static final GraphQLFieldDefinition SCRIPT_ERRORS = Query.getFieldDefinition("script_errors", AEMSDatabase.ANALOGIES, AEMSDatabase.Anomalies.SCRIPT_ERRORS, Scalars.GraphQLList.of(Scalars.GraphQLString));
    public Anomaly(String name, String description, List<GraphQLFieldDefinition> fieldDefinitions, List<GraphQLOutputType> interfaces) {
        super(name, description, fieldDefinitions, interfaces);
    }

    public static Anomaly getInstance() {
        if (instance != null)
            return instance;

        ArrayList<GraphQLFieldDefinition> defs = new ArrayList<>();
        defs.add(ID);
        defs.add(METER);
        defs.add(SENSOR);
        defs.add(SCRIPT);
        defs.add(LAST_EXECUTION);
        defs.add(EXEC_INTERMEDIATE_TIME);
        defs.add(SCRIPT_ERRORS);

        instance = new Anomaly("anomaly", "", defs, new ArrayList<GraphQLOutputType>());
        return instance;
    }
}

```

Abb. 68: Abbildung einer Datebanktabelle in GraphQL

Im oben angeführten Bild ist eine Definition der Klasse Anomaly zu erkennen, welche in der Datenbank die Tabelle Anomalies darstellt. Die selbst geschriebenen Utility-Funktionen wie Query.getFieldDefinition helfen bei der Deklaration der einzelnen Felder, welche dann über GraphQL angesprochen werden können. Die Utility-Funktionen übernehmen die Aufgabe der Abfragen und definieren, welche Daten in der Datenbank zu diesem Feld in GraphQL gehören.

In nachfolgendem Bild ist eine Überladung der Utility-Funktion Query.getFieldDefinition zu sehen, welche als Parameter den GraphQL-Feldnamen hat. Anschließend wird dieser in den GraphQL Abfragen benutzt, um das Feld, den Tabellennamen und die Spalte der Datenbanktabelle anzusprechen. Diese Funktion fragt die Daten aus der angegebenen Tabelle und Spalte in der Datenbank ab und gibt abhängig von dem spezifizierten Typ das Ergebnis als Zahl oder Zeichenkette zurück.

```

public static GraphQLFieldDefinition getFieldDefinition(final String FIELD_NAME, final String TABLE, final String COLUMN, final GraphQLScalarType TYPE) {
    return GraphQLFieldDefinition.newBuilder().name(FIELD_NAME).type(TYPE).dataFetcher(new DataFetcher<Object>() {
        @Override
        public Object get(DataFetchingEnvironment environment) {
            JSONObject obj = new JSONObject(environment.getSource().toString());
            if(TYPE.getName().equals(Scalars.GraphQLString.getName())) {
                return Query.execQuery(obj, TABLE, COLUMN);
            } else if(TYPE.getName().equals(Scalars.GraphQLInt.getName())) {
                return Integer.parseInt(Query.execQuery(obj, TABLE, COLUMN));
            } else if(TYPE.getName().equals(Scalars.GraphQLFloat.getName())) {
                return Float.parseFloat(Query.execQuery(obj, TABLE, COLUMN));
            }
            return null;
        }
    }).build();
}

```

Abb. 69: Funktion getFieldDefinition

Damit die Daten auch richtig abgefragt werden können, muss die GraphQL Abfrage einer gewissen Konvention folgen. Nachfolgend ist eine Beispielabfrage zu sehen, welche vom WebUI dazu benutzt wurde, die ID von Sensoren, welche auf einen Benutzer registriert sind, abzufragen.

```
"{ meters {is_sensor : \" + isSensor + "\"} { id } }"
```

Abb. 70: GraphQL Query

Die oben abgebildete Abfrage beinhaltet auch ein sogenanntes Argument. Es ist in einen Klammerausdruck verpackt und ist betitelt als `is_sensor`. Mit diesem Argument wird das SQL-Statement, welches die Daten abfragt in der WHERE-Klausel ergänzt.

### 3.5.3 Authorization

Ein wichtiger Punkt ist auch noch, welche Daten der Benutzer berechtigt ist zu sehen. Dies geschieht bei dieser Diplomarbeit auf Daten-Ebene. Es wird für jeden Datensatz überprüft, ob der Client die Daten dieses Datensatzes lesen darf.

```

boolean authorized = false;

if(instance.authorizationId.equals("215")) {
    authorized = true;
} else {
    if(AEMSDatabase.doesTablePossessColumn(table, "user")) {
        authorized = checkAuthorityLevel(obj, table);
    } else if(AEMSDatabase.doesTablePossessColumn(table, "meter")) {
        authorized = checkAuthorityLevel(obj, table);
    } else if(table.equals(AEMSDatabase.USERS) || !table.equals(AEMSDatabase.NOTIFICATION_METERS) || table.equals(AEMSDatabase.REPORT_STATISTICS) || table.equals(AEMSDatabase.USERS)) {
        authorized = checkAuthorityLevel(obj);
    } else {
        authorized = true;
    }
}

if(!authorized) {
    throw new graphql.GraphQLEException("Insufficient authorization!");
}

```

Abb. 71: Authorization Dispatch-Algorithmus

In der oben gezeigten Abbildung ist ein Teil des Algorithmus zu sehen, welcher bestimmt, welches Level an Authorization zutrifft. Dies ist hauptsächlich für die SQL-Abfragen von Notwendigkeit, da der Datensatz bis zu seinem dazugehörigen User verfolgt werden muss. Für den Benutzer/ die Benutzerin mit der ID 215, welche die ID des MASTER-Users ist, ist eine Ausnahme vermerkt. Dieser Benutzer/ diese Benutzerin wird in erster Linie vom AEMS-ReportBot benutzt und ist berechtigt alle Daten zu lesen. Zum Schluss lässt sich sagen, dass der Client nur jene Daten zu Gesicht bekommt, auf welche er auch Einsicht haben darf.

### 3.5.4 Änderungsoperation via REST

Als Datenformat für die Kommunikation mit dem REST-Service, in Bezug auf Änderungsoperationen, wurde JSON verwendet.

<https://www.mkyong.com/java/how-do-convert-java-object-to-from-json-format-gson-api/>  
[03-03-2018]

#### 3.5.4.1 Einfüge Operationen

Die Einfüge Operation wurde in der HTTP PUT-Methode des Java Rest Services implementiert. Eine beispielhafte Einfüge Operation ist:

```
{ meters : [{ id: "AT0000000000000003333", user: 185}]} 
```

Das oben angeführte Beispiel würde in die Tabelle Meters (= Name des Arrays) einen neuen Meter einfügen mit der ID AT0000000000000003333. Des Weiteren würde der Benutzer/die Benutzerin zu dem dieser Meter gehört, auf jenen Benutzer/ jene Benutzerin gesetzt werden, welche/r die ID 185 hat.

#### 3.5.4.2 Update Operationen

Die Update Operation wurde in der HTTP PUT-Methode des Java Rest Services implementiert.

Die Update Operation benötigt noch einen zusätzlichen Parameter neben dem Namen der Tabelle. Es wird auch eine ID Spalte benötigt. Die ID Spalte trägt jenen Namen, welche auch die Spalte in der Datenbank trägt. In untenstehenden Fall ist der Name der Spalte in der Datenbank id. Es wird nur jener Datensatz verändert, welcher in der Datenbank die id AT0000000000000003333 aufweist.

```
{ id:"AT0000000000000003333", meters : [{ id: "AT0000000000000003333", user: 185}]} 
```

#### 3.5.4.3 Delete Operationen

Die Delete Operation wurde in der HTTP DELETE-Methode des Java Rest Services implementiert.

Das gleiche Beispiel, welches schon bei der Update Operation vorgestellt wurde, ist auch für die Delete Operation anwendbar. Im Gegensatz zur Update Operation wird hier der Datensatz, welcher die id AT0000000000000003333 aufweist, aus der Datenbanktabelle gelöscht. Des Weiteren ist es auch nicht notwendig in das Array von meters etwas hineinzuschreiben, da für den Parser nur der Tabellenname interessant ist.

```
{ id:"AT0000000000000003333", meters :[] } 
```

## 3.6 Umsetzung Datenbank

### 3.6.1 Allgemein

Als Datenbank wurde für dieses Projekt eine PostgreSQL-Datenbank in der Version 9.5 benutzt. Wir haben uns für diese Datenbank entschieden, da es mit Abstand die beste, kostenlose Alternative zu einer Oracle-Datenbank ist und einen ähnlichen Aufbau besitzt.

Für die Entwicklung von Datenbank-Funktionen wurde auch eine Extension der PostgreSQL-Datenbank namens PLJAVA verwendet. Diese Extension ermöglicht es Java-Code innerhalb der Datenbank auszuführen.

<https://github.com/tada/pljava> [02-03-2018]

<https://tada.github.io/pljava/> [02-03-2018]

```
CREATE OR REPLACE FUNCTION aems.get_user_infos()
RETURNS bytea AS
$BODY$
declare
    decrypted_password_bytes bytea;
    aes_key varchar(40);
    aes_key_bytes bytea;
    user_meter_record record;
    json varchar(500000) default '';
    transferKey varchar(100);
    finalKeyNum numeric(100, 0) default 1;
begin
    aes_key := pg_read_file(datafile, 0, 16);
    aes_key_bytes := convert_to(aes_key, 'UTF-8');

    json := '[';
    for user_record in (select id, username, "password"
                        from aems."User") loop
        decrypted_password_bytes := aems.request_decryption(aes_key_bytes, user_record."password");
        decrypted_password := convert_from(decrypted_password_bytes, 'UTF-8');
        raise notice 'is: %', decrypted_password;

        json := json || ('{"id":' || user_record.id || ", "username":"' || user_record.username || '", "password":"' || decrypted_password || '", "meters":[';

        json := json || '"id":"' || user_meter_record.id || '", "type":"' || user_meter_record.metertype || '", ';
        json := json || '"location": {"city":"' || user_meter_record.city || '", "lat":"' || user_meter_record.latitude || '", "long":"' || user_meter_record.longitude
        end loop;
        select substr(json, 0, char_length(json)) into json; -- remove last comma
        json := json || ']';
    end loop;
    raise notice 'json: %', json;
    select substr(json, 0, char_length(json)) into json;
    json := json || ']';

    -- convert_to("key", 'UTF-8')
    select "key" into transferKey
    from aems.TransferInfos
    where id = 0;

    finalKeyNum := aems.salt(aems.hash_code(transferKey), 'master', aems.get_user_password('master'));

    --raise notice 'final key num: %', finalKeyNum;
    select substr(cast(finalKeyNum as varchar), 0, 17) into transferKey; -- key cannot exceed 16 digits
    select convert_to(transferKey, 'UTF-8') into aes_key_bytes;

    raise notice 'key: %', transferKey;

    delete from aems."DebugInfo";
    insert into aems."DebugInfo" (pk, "string") values (0, transferKey);

    return aems.request_encryption(aes_key_bytes, convert_to(json, 'UTF-8'));
end;
$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
ALTER FUNCTION aems.get_user_infos()
OWNER TO aems;
```

Abb. 72: Algorithmus zum Abfragen aller Benutzer für die BOT-Action im REST-Service

Der oben abgebildete Algorithmus wird von der BOT-Action der REST-API verwendet. Damit erhält der BOT alle registrierten Benutzer/Benutzerinnen. Wie man auch erkennen kann, wird das Userpasswort mit den Java Funktionen aus der eigenen Java-Library zuvor entschlüsselt. Das liegt daran, dass die Passwörter verschlüsselt in der Datenbank gespeichert werden. Nachdem alle Benutzerdaten von der Datenbank in eine JSON-Struktur überführt wurden, wird das Ergebnis mit den Benutzerdaten des Master-Users verschlüsselt. Nur wer jetzt die Benutzerdaten des Master-Users kennt, kann die Nachricht auch wieder entschlüsseln.

```

CREATE FUNCTION request_decryption(bytea, bytea) RETURNS bytea
LANGUAGE java
AS $$at.htlgkr.aems.util.crypto.Decrypter.requestDecryption$$;

```

Abb. 73: PLJAVA Funktionsdeklaration

In der oben gezeigten Abbildung ist eine externe Funktion zu erkennen. Diese Funktion stammt aus einer JAR-Datei, welche die Funktion `requestDecryption` in der Klasse `Decrypter` beinhaltet.

### 3.6.2 ERD

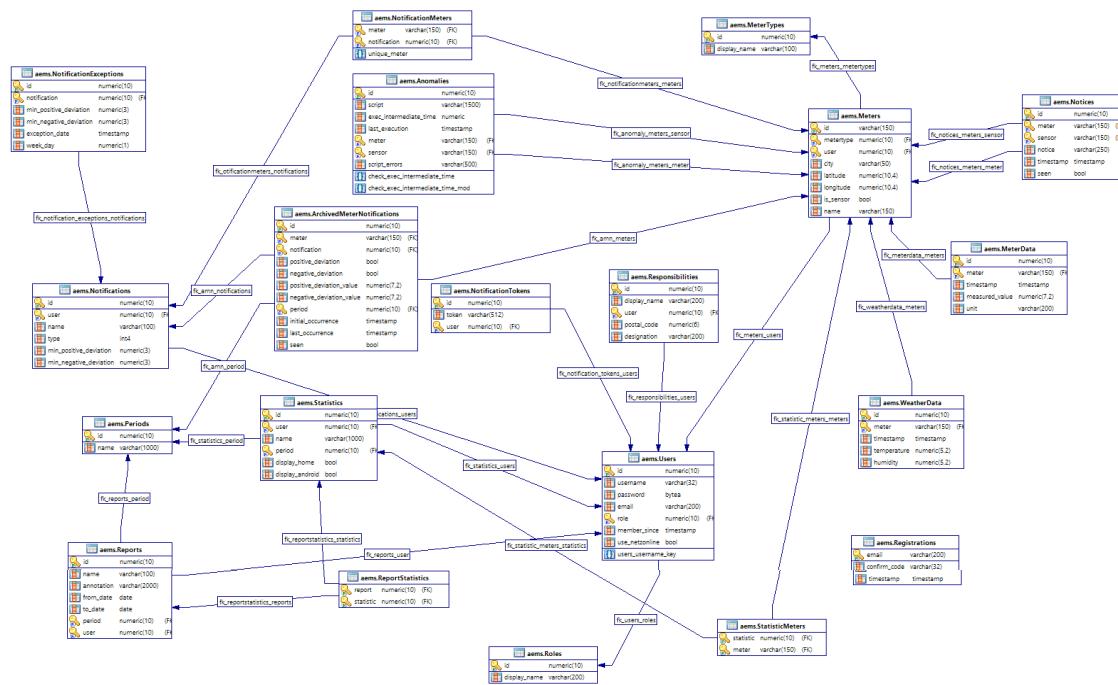


Abb. 74: ERD

### 3.7 Umsetzung Raspberry PI Konfigurationstool

Eine weitere Anwendung, welche im Rahmen der Diplomarbeit entstanden ist, ist das AEMS-Konfigurationstool. Im Grunde genommen handelt es sich bei diesem Tool um ein Werkzeug, welches das Auslesen von Sensoren und Zählern, wie Stromzähler, Gaszähler, Wärmemengenzähler und Wasserzählern mit Hilfe von Plugins ermöglicht. Das Plugin-System ist so umgesetzt, dass jederzeit neue Plugins das bestehende System erweitern können.

**Teile des Plugin-Systems wurden konzeptionell aus der vorherigen Diplomarbeit Smart-Meter-Integration übernommen.**

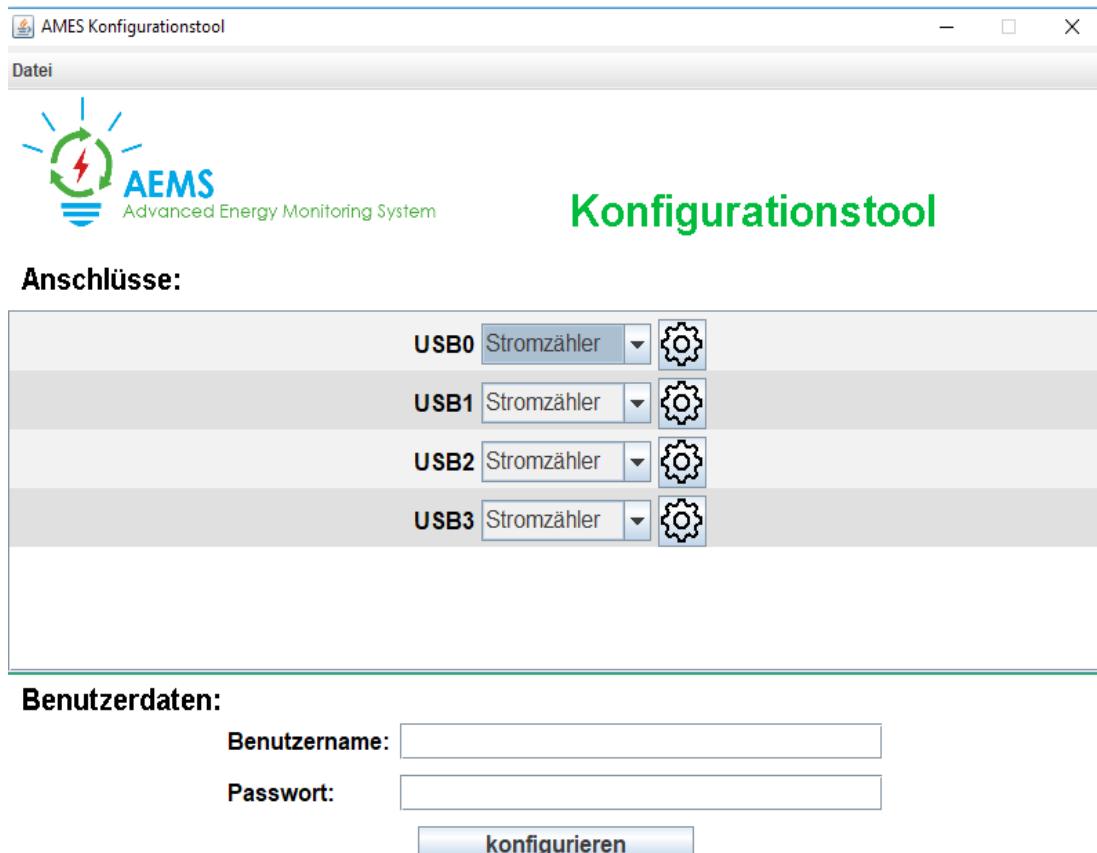


Abb. 75: UI des Konfigurationstools für den Raspberry PI

In der oben gezeigten Abbildung ist ein Menüband mit dem Reiter Datei zu erkennen. Dieser Reiter stellt die Funktionalität des Speicherns und Ladens der Konfiguration zur Verfügung. Unter der Kategorie Anschlüsse werden alle USB Anschlüsse, welche am Raspberry PI erkannt wurden, aufgelistet. Für jeden USB Anschluss ist es nun möglich einen Typ festzulegen. Dieser Typ ist ein Zählertyp oder ein Sensor. Unter der Kategorie Benutzerdaten wird der Benutzer/ die Benutzerin dazu aufgefordert, die Benutzerdaten einzugeben. Diese werden für die Übertragung der Daten mit dem AEMS-Rest-Service benötigt.

In der unten gezeigten Abbildung ist die Konfigurationsansicht eines Stromzählers zu sehen. Unter dem Punkt Zählernummer gibt man die Nummer des Zählers an, welcher in der Datenbank vorhanden ist. Unter dem Punkt Plugin wird man dazu aufgefordert, das Plugin, welches für die Verarbeitung der Daten verantwortlich ist, anzugeben.

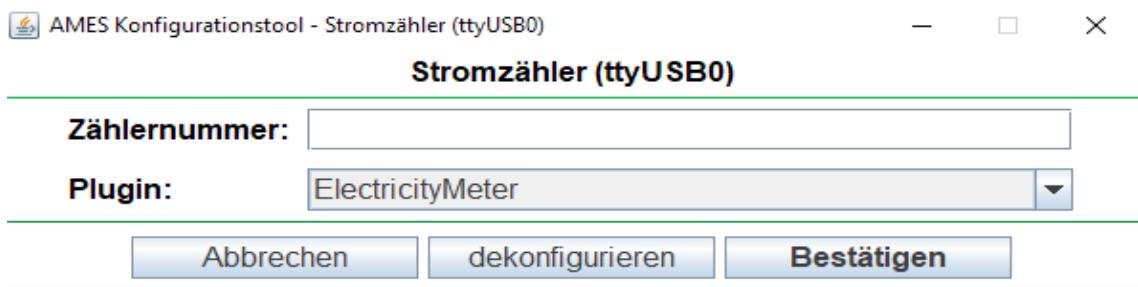


Abb. 76: Portkonfiguration am Beispiel Stromzähler

Damit unter dem Punkt Plugin auch Plugins angezeigt werden können, müssen diese zuvor dem System bekannt gegeben werden. Grundsätzlich werden Plugins von dem Verzeichnis „plugins“ geladen. Der unten abgebildete Algorithmus lädt die JAR Dateien in das System und speichert diese als sogenanntes PREFAB.

```
public static void loadPlugIns() {
    File[] files = new File(DIRECTORY).listFiles();
    for(File file : files) {
        if(file.isFile() && file.getName().endsWith(".jar")) {
            try {
                URLClassLoader loader = new URLClassLoader(new URL[]{new URL("file://" + file.getAbsolutePath())}, PlugInManager.class.getClassLoader());
                ZipInputStream is = new ZipInputStream(new FileInputStream(file));
                for(ZipEntry entry = is.getNextEntry(); entry != null; entry = is.getNextEntry()) {
                    if(!entry.isDirectory() && entry.getName().endsWith(".class")) {
                        @SuppressWarnings("unchecked")
                        Class<PlugIn> clazz = (Class<PlugIn>) Class.forName(entry.getName().substring(0, entry.getName().length() - 6), true, loader);
                        PLUGINS_PREFABS.add(clazz.newInstance());
                    }
                }
                is.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

Abb. 77: Algorithmus zum Laden der Plugins

Nachdem ein Plugin einem Anschluss zugeteilt wurde und dieser mit der Zählernummer konfiguriert wurde, wird das entsprechende PREFAB geklont und in ein lauffähiges Plugin verwandelt, welches die Daten der graphischen Eingabe beinhaltet (Zählernummer, Anschlussnummer). Jedes Plugin besitzt zudem ein Script, mit welchem die Daten von den USB Anschlüssen gelesen werden.

<https://stackoverflow.com/questions/26330234/how-can-i-load-a-class-file-from-a-running-jar-into-a-java-object-in-order-to-r> [03-03-2018]

```

public void run() {
    while(!shouldTerminate) {
        // start process
        InputStream is = null;
        Process process = null;
        try {
            process = Runtime.getRuntime().exec(file.getExecutingProgramName() + " " + file.getScriptFile().getPath(), null, new File(PlugInManager.DIRECTORY));
            Logger.log(LogType.INFO, "Process for plugin \"%s\" was started!", plugin.getName());
            // log error to console in case the process fails to execute properly!
            if(process.waitFor(1, TimeUnit.SECONDS)) {
                Logger.log(LogType.ERROR, "Process for plugin \"%s\" has terminated", plugin.getName());
                try(BufferedReader reader = new BufferedReader(new InputStreamReader(process.getErrorStream()))) {
                    for(String line = reader.readLine(); line != null; line = reader.readLine()) {
                        Logger.log(LogType.ERROR, line);
                    }
                } catch(Exception e) {
                    e.printStackTrace();
                }
                break;
            }
            // send init parameters to process hence to the "script"
            // json format: transmit port
            try (BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(process.getOutputStream()))) {
                writer.write(" port : \"" + setting.getPort() + "\"\r\n");
                writer.flush();
            } catch(Exception e) {
                e.printStackTrace();
            }
            is = process.getInputStream();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        // check for continuation and invoke plugin-function
        if(!plugin.readCyclic(plugin, is))
            break;
        try {
            if(process.isAlive())
                process.destroy();

            if(!process.isAlive())
                Logger.log(LogType.INFO, "Process for plugin \"%s\" was killed!", plugin.getName());
            is.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Abb. 79: Alorithmus für die zyklische Ausführung

In der oben gezeigten Abbildung ist der Algorithmus abgebildet, welcher den Prozess startet, welcher das Script ausführt. Anschließend wird der Input-Stream des Prozesses an die Methode readCyclic an das PlugIn weitergegeben. Von dort an kann der Programmierer/ die Programmiererin des PlugIn bestimmen, was mit den Daten passiert.

Das PlugIn bietet zudem auch noch einen Uploader, welcher sich um die korrekte Speicherung der erfassten Daten kümmert. Den Uploader kann man bei der PlugIn-Programmierung verwenden.

```

public class AEMSElectricityPlugIn extends PlugIn {

    public AEMSElectricityPlugIn() {
        super("ElectricityMeter PLUGIN", Setting.getSetting(MeterTypes.ELECTRICITY, new ScriptFile("py", "run.py")));
        super.getSetting().setMillisUntilRepetition(1000 * 60 * 15);
        super.setUploader(new AEMSUploader(this));
    }

    final Pattern NUMBER = Pattern.compile("(?<number>\d+)");
    final SimpleDateFormat FORMAT = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    @Override
    public boolean readCyclic(PlugIn plugin, InputStream inputStream) {
        getUploader().setPlugIn(plugin);

        try(BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream))) {
            for(String line = reader.readLine(); line != null; line = reader.readLine()) {
                Matcher matcher = NUMBER.matcher(line);
                if(matcher.find()) {
                    String value = matcher.group("number");
                    super.getUploader().upload(new UploadPackage().addData(new TableData("meter_data")
                        .addData("meter", plugin.getSetting().getMeterId()).addData("measured_value", value).addData("timestamp", FORMAT.format(new Date()))));
                }
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
        return true;
    }
}

```

Abb. 78: Electricity Meter Plugin

In der oben gezeigten Abbildung ist der Aufbau des ElectricityMeter Plugins zu erkennen. Mit setMillisUntilRepetition wird die Wiederholungszeit, also die Zeit nachdem die Methode readCyclic wieder aufgerufen werden sollte, auf fünfzehn Minuten gesetzt. Als Uploader wird der AEMSUploader verwendet, welcher die Daten entsprechend der Struktur in der Datenbank hochlädt. In dem oben gezeigten Beispiel wird der Wert, welcher über den OutputStream des Prozesses gesendet wurde, mithilfe des AEMSUploaders in die Datenbank gespeichert. Der Wert wurde zuvor mit einem optischen Infrarot Lesekopf eingelesen.

### 3.8 Umsetzung openHAB

Für die Visualisierung der aktuellen Verbrauchswerte wurde auch noch ein Binding für OPENHAB entwickelt. OPENHAB ist eine Open-Source Software, welche sich mit dem Thema Home-Automation beschäftigt. OPENHAB stellt verschiedene Typen zur Verfügung, für diese Diplomarbeit sind jedoch nur die THINGS und die CHANNELS von Interesse.

<https://www.openhab.org/> [02-03-2018]

<https://docs.openhab.org/> [03-03-2018]

```
<thing-type id="aems-meter-interface">
    <label>AEMS Meter Interface</label>
    <description>This interface grants you an insight into the entirety of data your Raspberry-PI has collected about your smart-meter!</description>
    <channels>
        <channel id="aems-temperature-info-channel" typeId="temperature-info-channel"/>
        <channel id="aems-humidity-info-channel" typeId="humidity-info-channel"/>
        <channel id="aems-current-consumption-channel" typeId="current-consumption-channel"/>
    </channels>
    <config-description>
        <parameter name="username" type="text" required="true">
            <label>Username</label>
            <description>Your AEMS-Username</description>
        </parameter>
        <parameter name="password" type="text" required="true">
            <context>password</context>
            <label>Password</label>
            <description>Your AEMS-Password</description>
        </parameter>
        <parameter name="meterId" type="text" required="true">
            <label>Meter-ID</label>
            <description>The id of your smart-meter!</description>
        </parameter>
    </config-description>
</thing-type>
```

Abb. 80: Basiskonfigurations XML für das openHAB Binding

In der oben angeführten Abbildung ist ein Teil der Basiskonfiguration des THINGS aems-meter-interface zu sehen. In dieser Konfigurationsdatei werden zum einen die verschiedenen CHANNELS angelegt. CHANNELS sind Kanäle, durch welche Daten an das THING gesendet werden, welche somit dem THING seine Funktionalität verleihen. Zum anderen kann auch, wie im Bild zu sehen ist, eine config-description erstellt werden. Diese dient dem Erstellen von Eingabefeldern, welche den Benutzer/ die Benutzerin bei der Erstellung des THINGS auffordert, gewisse Daten einzugeben. In diesem Fall muss der Benutzername und das Benutzerpasswort sowie die Zählernummer eingegeben werden. Wurden alle diese Daten bereitgestellt, werden die Daten im fünf Minuten Takt von der Datenbank abgefragt, wie es im Bild unterhalb an dem Thread.sleep(5000) zu erkennen ist. Die Datenübertragung ist mit AES verschlüsselt. Dafür muss zuvor wieder ein geheimer Schlüssel ausgetauscht werden, welches innerhalb von exchangeAESKey passiert. Nachdem die Initialisierung abgeschlossen wurde, wird der THING Status auf Online gesetzt, um zu benachrichtigen, dass die Initialisierung abgeschlossen wurde.

```

@Override
public void initialize() {
    // TODO: Initialize the thing. If done set status to ONLINE to indicate proper working.
    // Long running initialization should be done asynchronously in background.

    config = getConfigAs(AEMSInterfaceConfiguration.class);

    exchangeAESKey(config.username, config.password);

    getThing().setLabel(getThing().getLabel() + " (" + config.meterId + ")");

    (thread = new Thread(() -> {
        while (shouldUpdate) {
            updateData();
            try {
                Thread.sleep(5000); // update every 5 minutes
            } catch (InterruptedException e) {
                logger.info("update thread for meter \"{}\" has stopped executing!", config.meterId);
                return;
            }
        }
    })).start();

    updateStatus(ThingStatus.ONLINE);
}

```

Abb. 81: Zyklischer Update-Mechanismus

Alle fünf Minuten werden die Daten von der Datenbank abgefragt. Dadurch muss auch alle fünf Minuten der State des jeweiligen CHANNELS aktualisiert werden. Dies wird mit updateState bewerkstelligt. Der State eines CHANNELS ist im Grunde genommen der Wert, welcher angezeigt wird.

```

BufferedReader reader = new BufferedReader(new InputStreamReader(con.getInputStream()));
String jsonLine = reader.readLine();
jsonLine = new String(Decrypter.requestDecryption(key, Base64.getUrlDecoder().decode(jsonLine.getBytes())));

String humidity = "not set";
String temperature = "not set";
String measured_value = "not set";
String unit = "not set";

Matcher matcher = JSON_PATTERN.matcher(jsonLine);
while (matcher.find()) {
    humidity = matcher.group("humidity");
    temperature = matcher.group("temperature");
    measured_value = matcher.group("measuredvalue");
    unit = matcher.group("unit");
}

updateState(getThing().getChannel(AemsInterfaceBindingConstants.HUMIDITY_CHANNEL).getUID(),
    new StringType(humidity + " %"));
updateState(getThing().getChannel(AemsInterfaceBindingConstants.TEMPERATURE_CHANNEL).getUID(),
    new StringType(temperature + " °n"));
updateState(getThing().getChannel(AemsInterfaceBindingConstants.CURRENT_CONSUMPTION_CHANNEL).getUID(),
    new StringType(measured_value + " " + unit));

updateThing(getThing());

logger.info("humidity: {}, temperature: {}, measured_value: {}, unit: {}", humidity, temperature,
    measured_value, unit);
logger.info("AEMS Interface for meter {} has been updated!", config.meterId);

```

Abb. 82: Extraktion der Daten und Aktualisierung der Channels

## 3.9 Weitere Dienste

### 3.9.1 aems-apilib

Die AEMS-APILIB ermöglicht das Abfragen, Einfügen, Verändern und Löschen von Daten. Die erforderlichen Informationen, welche dabei zwischen Client und Server ausgetauscht werden, werden im JSON Format übermittelt. Da diese JSON-Konstrukte vor allem bei Einfügeoperationen sehr groß und komplex werden können, ist das händische erstellen des JSON Strings langwierig und fehleranfällig. Durch die **aems-apilib** wird der JSON String mithilfe der GSON Bibliothek erstellt. Außerdem übernimmt die aems-apilib die Kommunikation mit dem REST-Service.

#### 3.9.1.1 Verwendung - Abfragen

Abfragen werden durch die Klasse **AemsQueryAction** ermöglicht. Es müssen die Daten zur Authentifikation und die eigentliche GraphQL-Abfrage mitgegeben werden.

```
AemsQueryAction q = new AemsQueryAction(testUser, EncryptionType.SSL);
q.setQuery("{ meters { id is_sensor user } }");

{
    "user": 0,
    "salt": "5Um73QJ4ADRkXsx1",
    "auth_str": "f726bf21577b84ecd745eb1a5074ff0b7bff2fe5dl
    "action": "QUERY",
    "data": "{ meters { id is_sensor user } }",
    "encryption": "SSL"
}
```

Abb. 83: Verwendung der AemsQueryAction und generiertes JSON

#### 3.9.1.2 Verwendung - Einfügen

Das Einfügen von Daten wird durch die Klasse **AemsInsertAction** gehandhabt. Hier muss der Name der Tabelle angegeben werden. Zusätzlich müssen mit der „write“ Methode die Namen der Spalten in der Datenbanktabelle und deren gewünschter Wert angegeben werden.

```
AemsInsertAction i = new AemsInsertAction(testUser, Encry
i.setTable("Meters");
i.write("id", "AT0000001");
i.write("user", 0);
i.write("is_sensor", true);
i.endWrite();
i.write("id", "AT0000002");
i.write("user", 0);
i.write("is_sensor", false);
i.endWrite();
```

Abb. 84: Verwendung der AemsInsertAction

```

    "user": 0,
    "salt": "1b3ZqAnW5mWDYsOR",
    "auth_str": "706e50c6795c8ad484f71039",
    "action": "INSERT",
    "data": [
        "Meters": [
            {
                "is_sensor": true,
                "id": "AT0000001",
                "user": 0
            },
            {
                "is_sensor": false,
                "id": "AT0000002",
                "user": 0
            }
        ]
    ]
}

```

Abb. 85: Generiertes JSON bei AemsInsertAction

### 3.9.1.3 Verwendung - Bearbeiten

Das Bearbeiten von Datensätzen wird von der Klasse **AemsUpdateAction** geregelt. Neben den Daten, welche aktualisiert werden sollen, wird auch noch der Name und Wert der Spalte benötigt, welche die Selektion der Datenbankzeilen einschränkt. Außerdem wird wieder der Name der Datenbanktabelle benötigt.

### 3.9.1.4 Verwendung - Löschen

Das Löschen von Datensätzen wird von der Klasse **AemsDeleteAction** geregelt. Es wird lediglich der Name der Tabelle und der Name und Wert der Spalte benötigt, welcher die Selektion einschränkt.

## 3.9.2 Diffie-Hellman Library

Mit Hilfe der Diffie-Hellman Library ist es möglich AES verschlüsselte Nachrichten zu entschlüsseln und genauso ist es möglich PLAIN-TEXT AES zu verschlüsseln. Des weiteren stellt diese Library auch die Funktionalität eines Schlüsselaustauschs zur Verfügung (E2EE-Prinzip).

```

public static byte[] requestDecryption(byte[] key, byte[] encrypted) throws IllegalBlockSizeException, BadPaddingException,
    InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException {
    Key keyBytes = new SecretKeySpec(key, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.DECRYPT_MODE, keyBytes);
    byte[] decrypted = cipher.doFinal(encrypted);
    return decrypted;
}

```

Abb. 86: Function requestDecryption

Die oben gezeigte Funktion entschlüsselt zuvor verschlüsselte AES Nachrichten, indem der Key für die Entschlüsselung gemeinsam mit der Nachricht übergeben wird.

<https://stackoverflow.com/questions/15554296/simple-java-aes-encrypt-decrypt-example>  
[03-03-2018]

```

public static BigDecimal salt(BigDecimal key, String username, String password) {
    key = key.multiply(new BigDecimal(String.valueOf(HashCode.getHashCode(username)))); 
    key = key.multiply(new BigDecimal(String.valueOf(HashCode.getHashCode(password)))); 
    return new BigDecimal(key.abs().toString().substring(0, DiffieHellmanProcedure.KEY_LENGTH));
}

```

Abb. 87: Function salt

Die oben gezeigte Funktion saltet den bestehenden Key mit den jeweiligen Benutzerdaten.

```

public static String getKeyDetails() {
    final Random RANDOM = new Random();
    StringBuilder builder = new StringBuilder();
    builder.append("(");
    BigDecimal baseNumber = createRandomNumber(BASE_NUMBER_LENGTH);
    builder.append("base:").append(baseNumber);
    //DiffieHellmanProcedure.baseNumber = baseNumber;

    BigDecimal modNumber = new BigDecimal(RANDOM.nextInt());
    modNumber = modNumber.multiply(createRandomNumber(MOD_NUMBER_LENGTH));
    builder.append(", mod:").append(modNumber);
    DiffieHellmanProcedure.modNumber = modNumber;

    BigDecimal secretNumber = new BigDecimal(RANDOM.nextInt(SECRET_TOP_LIMIT) + SECRET_BOTTOM_LIMIT);
    secretNumberClient = secretNumber;
    BigDecimal combination = compute(baseNumber, modNumber, secretNumber);
    builder.append(", combination:").append(combination.toString());
    builder.append(")");
}

return builder.toString();
}

```

Abb. 88: Function getKeyDetails

Diese Funktion berechnet den ersten Teil, welcher dazu benötigt wird, am anderen Ende den Schlüssel zu generieren. Die berechneten Werte werden anschließend an das andere Ende geschickt.

```

public static void sendKeyInfos(Socket socket) throws IOException {
    try(BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()))) {
        writer.write(getKeyDetails());
        writer.write("\r\n");
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        socket.close();
    }
}

```

Abb. 89: Übermittlung der berechneten Daten an den anderen Client

Danach berechnet der andere Client seinen Teil des Schlüssels, welcher dann auch schon der finale Schlüssel ist. Die Daten werden abermals übermittelt und ab diesem Zeitpunkt besitzen beide Clients den gleichen Schlüssel.

<https://www.youtube.com/watch?v=M-0qt6tdHzk> [03-03-2018]

### 3.9.3 AEMS-Database Library

Diese Library bietet die Möglichkeit SQL-Statements auf eine einfachere Art und Weise zusammenzubauen. Für die Ausführung der generierten SQL-Statements wird die Java Datenbank Schnittstelle JDBC verwendet. Für die Verbindung mit der PostgreSQL Datenbank wird ein Treiber benötigt, welcher von der offiziellen Seite von PostgreSQL akquiriert wurde.

<https://jdbc.postgresql.org/> [03-03-2018]

```

public void open(String username, String password) throws SQLException {
    Properties p = new Properties();
    p.setProperty("user", username);
    p.setProperty("password", password);

    connectionHandle = DriverManager.getConnection("jdbc:postgresql://localhost:5432/AEMSData", p);
    connectionHandle.setAutoCommit(true);
    statementHandle = connectionHandle.createStatement();
}

```

**Abb. 91: Function open**

Um sich mit der PostgreSQL Datenbank verbinden zu können muss ein sogenannter Connection-String verwendet werden. Nachdem man mit der Datenbank verbunden wurde, repräsentiert das statementHandle die Transaktion.

```

public void delete(String schema, String tableName, HashMap<String, String> selection, boolean and, String customSelection) throws SQLException {
    StringBuffer buffer = new StringBuffer();
    buffer.append("DELETE FROM ");

    schema = (schema == null) ? defaultSchema : schema;

    if(schema != null)
        buffer.append("\\"").append(schema).append("\\");

    buffer.append("\\"").append(tableName).append("\\");

    if(customSelection == null && selection != null) {
        buffer.append(" WHERE ");
        for(String key : selection.keySet()) {
            buffer.append(formatEqualOrLike(key, selection.get(key), false));
            if(and)
                buffer.append(" AND ");
            else
                buffer.append(" OR ");
        }

        if(and)
            buffer.setLength(buffer.length() - 5);
        else
            buffer.setLength(buffer.length() - 4);
    } else if(customSelection != null && selection == null) {
        buffer.append(" WHERE ");
        buffer.append(customSelection);
    } /* else {
        throw new IllegalArgumentException("Either the custom selection or the default map selection can be initialized! Not both simultaneously!");
    }*/

    buffer.append(";");
    String sql = buffer.toString();
    executeSQL(sql);
}

```

**Abb. 90: Funktion delete**

Die oben gezeigte Funktion baut einen String zusammen, welcher im Endeffekt ein SQL-DELETE Statement ist. Während der Konstruktion des Strings werden auch in der WHERE Klausel die Java-Datentypen berücksichtigt und Vergleiche mit LIKE oder EQUALS dementsprechend generiert.

Ähnliches wurde für die Funktionen SELECT, INSERT und UPDATE implementiert. Diese werden jedoch aus Platzgründen nicht in diese Dokumentation aufgenommen, da die Algorithmen zu umfangreich sind.

## **4        Verwendete Libraries**

### **4.1        Externe Quellen**

#### **4.1.1      Apache POI**

Apache POI wurde verwendet, um Daten aus den binären Exceldateien zu lesen.

Website: <https://poi.apache.org/>

#### **4.1.2      Apache HttpClient**

Die Apache HttpClient Klassenbibliothek wurde verwendet, um die Benutzerdaten bei der Registrierung auf Richtigkeit zu überprüfen, indem ein Login Request an das Netzonline-Portal gesendet wird.

Website: <https://hc.apache.org/httpcomponents-client-5.0.x/index.html>

#### **4.1.3      Bootstrap**

Bootstrap ist eine CSS-Library mit vordefinierten Styles für HTML-Elemente.

Website: <https://getbootstrap.com/>

#### **4.1.4      Bootsnipp-Modal**

Dient als Designgrundlage der Modals.

Website: <https://bootsnipp.com/snippets/featured/clean-modal-login-form>

#### **4.1.5      Firebase**

Wird für die Kommunikation der Notifications der Android-App verwendet.

Website: <https://firebase.google.com/>

#### **4.1.6      Graphql-Java**

Wird für die Abfrage der Daten aus der Datenbank verwendet.

Website: <https://github.com/graphql-java/graphql-java>

#### **4.1.7     Gson**

Wird für das Erzeugen von JSON Strukturen verwendet.

Website: <https://github.com/google/gson>

#### **4.1.8     HTMLUnit**

HTMLUnit wurde dazu verwendet einen Web-Browser zu simulieren, um es dem Report Bot zu ermöglichen, die erforderlichen Dateien herunterzuladen.

Website: <https://htmlunit.sourceforge.net>

#### **4.1.9     html2Canvas**

Wird verwendet um die Canvas Elemente der Website(Statistiken) zu bekommen und diese in ein PDF integrieren zu können.

Website: <https://github.com/niklasvh/html2canvas/releases>

#### **4.1.10    JSON**

Wird für das Erzeugen von JSON Strukturen verwendet.

Website: <https://github.com/stleary/JSON-java>

#### **4.1.11    jsPDF**

Diente für die Umsetzung der Downloadfunktion von Statistiken.

Website: <https://parall.ax/products/jspdf>

#### **4.1.12    MPAndroidChart**

MPAndroidchart wurde für die Erstellung der Statistiken in der Android-App benötigt.

Website: <https://github.com/PhilJay/MPAndroidChart>

#### **4.1.13    Notify.js**

Wird für das Anzeigen von Statusmeldungen beim Webinterface und Admin Webinterface angewendet.

Website: <https://notifyjs.com/>

#### **4.1.14 Noty**

Wurde für das Grundgerüst der Notifications im Webbereich genutzt.

Website: <https://ned.im/noty/#/>

#### **4.1.15 PrettyFaces**

PrettyFaces wurde verwendet, um die URLs für das Webinterface anzupassen.

Website: <https://www.ocpsoft.org/prettyfaces/>

#### **4.1.16 Pyserial**

Wurde dazu verwendet, die Raspberry PI Scripts vom USB-Port zu lesen.

Website: <https://pypi.python.org/pypi/pyserial>

### **4.2 Selbst entwickelt**

#### **4.2.1 aems-apilib**

Die aems-apilib dient der Kommunikation zwischen Client und Server. Mit der Library wird der für die Kommunikation nötige JSON-String erzeugt.

#### **4.2.2 Diffie-Hellman Library**

Dienst für die Verschlüsselung und den Datenaustausch. Als Grundgerüst wird das Diffie-Hellman Protokoll verwendet.

#### **4.2.3 AEMS-Database Library**

Eine Library, welche die Kommunikation mit der Datenbank erleichtert. Mit dieser Library ist es möglich SQL-Statements zusammenzubauen.

## **5        Ergebnisse**

### **5.1      Erreichte Ziele**

- Erstellung einer Website mit einer einfachen Übersicht über seine Energieverbrauchswerte.
- Benachrichtigungssystem bei Verfügbarkeit neuer Berichte und für abweichende Verbrauchswerte.
- Entwicklung eines Administrationstools für die Benutzerverwaltung.
- Entwicklung eines Plugins, welches die Möglichkeit bietet eigene Zähler auf Raspberry PIs zu definieren.
- Entwicklung einer Android-App für Benachrichtigungen am Smartphone und Einsicht in die Statistiken.
- Entwicklung einer modularen Anomalierkennung, welche durch Erstellung von Scripts erweitert werden kann.

### **5.2      Auswirkung**

Durch das einfache Monitoring der Verbrauchswerte und die Benachrichtigungen bei abweichendem Verbrauch, soll Energie eingespart werden können. Dies bringt mehrere positive Aspekte mit sich:

- Einsparung von Energiekosten
- Vorteil für die Umwelt durch sinkenden Energieverbrauch

### **5.3      Abschluss**

Mit Stand 28. März 2018 wurde das Projekt abgeschlossen.

## **5.4 Statement des Auftraggebers**

Als Auftraggeber möchte ich dem Projektteam zu ihrer Leistung gratulieren.

Während der Projektdauer waren die Maturanten hoch motiviert. Die Kommunikation, vor allem über E-Mail funktionierte perfekt und zeitnah. Dringende Fragen wurden unverzüglich beantwortet.

Alle 3 Maturanten wirken sehr kompetent, hatten sich die Aufgaben für mich sehr schlüssig verteilt.

Die Software lässt hinsichtlich der geplanten Funktionalität keine Wünsche übrig. Meiner Überzeugung nach ist das entwickelte Tool für die Praxis bestens geeignet. Nach einer inhaltlichen Vorstellung beabsichtigen 3 weitere öö. Klima-und Energiemodell-Regionen einen Einsatz mit ca. 1.500 Zählern durchzuführen.

Jedes Unternehmen kann sich glücklich schätzen, einen dieser 3 Absolventen für sich als Mitarbeiter gewinnen zu können.

(Pölzlberger Herbert 2018, E-Mail)

## **6 Danksagung**

Ein ausdrücklicher Dank gebührt unserem Auftraggeber Herrn Ing. Herbert Pölzlberger, MSc für die Bereitstellung dieses Projekts als Diplomarbeit. Wir möchten uns sehr herzlich dafür bedanken, dass wir dieses Projekt in Kooperation mit der Energiegenossenschaft Eferding realisieren durften.

Vielen Dank für das tolle Projekt und die sehr gute Zusammenarbeit!

Ein Dank gebührt auch unserem Betreuungslehrer, Herrn DI Josef Doppelbauer, welcher uns bei Fragen immer mit Rat und Tat zur Seite stand.

## Quellen und Literaturverzeichnis

### Online, aus dem Internet

Android Developers: <https://developer.android.com/index.html>

Android-Login: <https://sourcey.com/beautiful-android-login-and-signup-screens-with-material-design/>

Apache POI: <https://poi.apache.org/>

Apache.org: <https://www.apache.org/>

Bootsnipp-Designs: <https://bootsnipp.com/>

Bootstrap: <https://getbootstrap.com/>

Compiler Building - Crenshaw: <https://compilers.iecc.com/crenshaw/>

GraphQL Java: <http://graphql-java.readthedocs.io/en/v7/>

Gson: <https://github.com/google/gson>

html2canvas: <https://github.com/niklasvh/html2canvas/releases>

HtmlUnit: <http://htmlunit.sourceforge.net/>

JDBC-PostgreSQL: <https://jdbc.postgresql.org/>

jsPDF: <https://parall.ax/products/jspdf>

MPAndroidChart: <https://github.com/PhilJay/MPAndroidChart>

Mykong JSON: <https://www.mkyong.com/java/how-do-convert-java-object-to-from-json-for-format-gson-api/>

Netz-Online: <https://netz-online.netzgmbh.at/eServiceWeb/main.html>

Notify.js: <https://notifyjs.com/>

Noty: <https://ned.im/noty/#/>

OpenHab Docs: <https://docs.openhab.org/>

OpenHab: <https://www.openhab.org/>

OpenWeatherMap: <https://openweathermap.org/>

PL-Java Github: <https://github.com/tada/pljava>

PL-Java Tada: <https://tada.github.io/pljava/>

PrettyFaces: <https://www.ocpsoft.org/prettyfaces/>

Stackoverflow: <https://stackoverflow.com/>

W3-Schools: <https://www.w3schools.com/>

Youtube- Schlüsselaustausch: <https://www.youtube.com/watch?v=M-0qt6tdHzk>

### **Sonstige Quellen**

Statement des Auftraggebers: (Pöhlberger Herbert, E-Mail an Lukas Knoll, 08.02.2018)

# Verzeichnis der Abbildungen, Tabellen und Abkürzungen

Abb. 1: Übersicht Frontend Dateien.....	4
Abb. 2: Startseite AEMS.....	5
Abb. 3: UI-Composition .....	5
Abb. 4: CSS .....	6
Abb. 5: Navigation-Bar .....	7
Abb. 6: Modal aufruf .....	7
Abb. 7: Modal.....	8
Abb. 8: Anzeige Startseite .....	9
Abb. 9: Funktionen Webpages .....	10
Abb. 10: Mobile Function .....	10
Abb. 11: Übersicht über verschiedene DisplayBeans .....	14
Abb. 12: Update-Methode der UserMeterBean,.....	14
Abb. 13: Anwendungsbeispiel einer ActionBean nach dem MVC-Prinzip von JSF.....	15
Abb. 14: Beispiel für ein komplexeres Eingabeelement .....	15
Abb. 15: Implementierung der „doDelete“ Methode in der WarningActionBean. ....	16
Abb. 16: Pop-Up bei fehlgeschlagenen Login-Versuch.....	17
Abb. 17: Glocken-Symbol mit zwei neuen Benachrichtigungen.....	17
Abb. 18: Verzeichnisstruktur der Abfrage-Dateien.....	18
Abb. 19: Beispiel einer GraphQL Abfrage mit Platzhalter für die Zähler ID - #METER_ID# .....	18
Abb. 20: Einsatz der AemsUtils.getQuery() Methode .....	18
Abb. 21: Fehlermeldung bei Servlet-Aufruf ohne aktive Session .....	19
Abb. 22: Typische Ansicht einer Monatsstatistik.....	19
Abb. 23: Erstellung des Chart Objekts in JavaScript .....	20

Abb. 24: Auszug aus der Implementation des Downloads von Statistiken.....	21
Abb. 25: Ablaufdiagramm des Report-Bots .....	22
Abb. 26: Auszug aus einer Excel-Datei.....	23
Abb. 27: bot.properties Konfigurationsdatei.....	23
Abb. 28: Code Ausschnitt aus der BotConfiguration Klasse .....	24
Abb. 29: Anwendung von XPATH zum Finden von Buttons .....	24
Abb. 30: Anmeldeforumlar.....	25
Abb. 31: Übersichtsseite.....	25
Abb. 32: Verwendung der Apache POI Klassenbibliothek .....	26
Abb. 33: Bot Tasks .....	26
Abb. 34: Task-Allgemein .....	27
Abb. 35: Task-Trigger .....	27
Abb. 36: Task-Ausführung .....	28
Abb. 37: Task-Eigenschaften .....	28
Abb. 38: Code zum Abrufen der Temperatur an der Position eines Zählers.....	29
Abb. 39: Android-App Struktur.....	30
Abb. 40: Anlegen der Statistik Tabs .....	31
Abb. 41: SectionPagerAdapter .....	32
Abb. 42: Selektion der Statistik für den Download.....	33
Abb. 43: Statistik Download .....	34
Abb. 44: Einfache Bar Charts .....	35
Abb. 45: Combined Chart - Bar Set.....	36
Abb. 46: Combined Chart - Line Graph.....	37
Abb. 47: Combined Data .....	37
Abb. 48: Login Validate.....	38

Abb. 49: Login-Failed .....	38
Abb. 50: Login in AEMS .....	39
Abb. 51: Loginprozess.....	40
Abb. 52: onLoginSuccess().....	40
Abb. 53: Optionsmenü .....	41
Abb. 54: NotificationAdapter .....	42
Abb. 55: MyFirebaseInstanceIdService .....	43
Abb. 56: MyFirebaseMessagingService .....	43
Abb. 57: WebUI .....	44
Abb. 58: AnomalieN Table .....	45
Abb. 59: Key-Exchange .....	46
Abb. 60: Funktion fetchAnomalies .....	46
Abb. 61: Scripting-Language-Interpreter Main-Loop.....	47
Abb. 62: Statement Deklarationen .....	47
Abb. 63: Statement Detections Algorithmus.....	48
Abb. 64: Numerischer Evaluator.....	48
Abb. 65: Teil des Algorithmus des booleschen Expression-Parsers .....	49
Abb. 66: Überprüfung der Autorität.....	54
Abb. 67: AEMS Key Service Thread.....	55
Abb. 68: Abbildung einer Datebanktabelle in GraphQL .....	55
Abb. 69: Funktion getFieldDefinition.....	56
Abb. 70: GraphQL Query .....	56
Abb. 71: Authorization Dispatch-Algorithmus.....	56
Abb. 72: Algorithmus zum Abfragen aller Benutzer für die BOT-Action im REST-Service .....	58
Abb. 73: PLJAVA Funktionsdeklaration .....	59

Abb. 74: ERD.....	59
Abb. 75: UI des Konfigurationstools für den Raspberry PI .....	60
Abb. 76: Portkonfiguration am Beispiel Stromzähler .....	61
Abb. 77: Algorithmus zum Laden der Plugins.....	61
Abb. 78: Electricity Meter Plugin.....	62
Abb. 79: Alorithmus für die zyklische Ausführung .....	62
Abb. 80: Basiskonfigurations XML für das openHAB Binding .....	63
Abb. 81: Zyklischer Update-Mechanismus .....	64
Abb. 82: Extraktion der Daten und Aktualisierung der Channels .....	64
Abb. 83: Verwendung der AemsQueryAction und generiertes JSON .....	65
Abb. 84: Verwendung der AemsInsertAction .....	65
Abb. 85: Generiertes JSON bei AemsInsertAction.....	66
Abb. 86: Function requestDecryption .....	66
Abb. 87: Function salt.....	67
Abb. 88: Function getKeyDetails.....	67
Abb. 89: Übermittlung der berechneten Daten an den anderen Client .....	67
Abb. 90: Funktion delete .....	68
Abb. 91: Function open .....	68
Abb. 92: Bildschirmmaske Startseite .....	91
Abb. 93: Bildschirmmaske Startseite nach dem Login.....	91
Abb. 94: Bildschirmmaske Notifications.....	92
Abb. 95: Bildschirmmaske Statistiken.....	92
Abb. 96: Bildschirmmaske Modal.....	92
Abb. 97: Bildschirmmaske Administration Startseite .....	93
Abb. 98: Bildschirmmaske Administratoren Verwaltung.....	93

Abb. 99: Bildschirmmaske Definition der Zuständigkeitsbereiche .....	94
Abb. 100: Bildschirmmaske App Login .....	94
Abb. 101: Bildschirmmaske App Anzeige der Statistiken.....	95
Abb. 102: Bildschirmmaske App Liste der Benachrichtigungen .....	95
Abb. 103: Bildschirmmaske App Benachrichtigungs Informationen .....	96
Abb. 104: Bildschirmmaske Notifications.....	96

# 7 Anhang

## 7.1 Projektdokumentation (Soll-Ist-Vergleich, Kostendarstellung)

Nr.	PSP-Code	Bezeichnung	Fortschr.	Status	Verantwortlichkeit	Basis Termine	Aktuelle Termine		Personal		Personalkosten		
							Start	Ende	Start	Ende	Basis	Ist	
Termine												Ressourcen (PT)	
1	1 AEMS	1.1 Projektmanagement	100% grün	Lukas Knoll	07.07.17 20:01:18	25.01.17 07:07:17	18.03.18 07:01:18	198	418	624,0	624,0	62.400	62.400
2	3 1.1.1 Projekt gestartet	100% grün	Lukas Knoll	07.07.17 07:07:17	07:07:17	07:07:17	0	0	46,0	46,0	4.600	4.600	
3	4 1.1.2 Projektstart	100% grün	Lukas Knoll	07.07.17 07:07:17	07:07:17	07:07:17	1	1	1,0	1,0	0	0	
4	5 1.1.3 Projektkoordination	100% grün	Lukas Knoll	07.07.17 20:01:18	07:07:17	20:01:18	198	22,0	22,0	100	100	100	100
5	6 1.1.4 Projektcontrolling	100% grün	Lukas Knoll	07.07.17 20:01:18	07:07:17	20:01:18	198	20,0	20,0	2,000	2,000	2,200	2,200
6	7 1.1.5 Projektabschluss	100% grün	Lukas Knoll	15.01.18 20:01:18	15.01.18	20:01:18	6	3,0	3,0	300	300	4.600	4.600
7	8 1.1.6 Projekt abgeschlossen	100% grün	Lukas Knoll	20:01:18 20:01:18	0	20:01:18	0	0	0	0	0	0	0
8	9 1.2 Einarbeitungsphase	100% grün	Lukas Knoll	07.07.17 21:07:17	07:07:17	21:07:17	15	50,0	50,0	5.000	5.000	5.000	5.000
9	10 1.2.1 Webdevelopment	100% grün	Lukas Knoll	07.07.17 14:07:17	07:07:17	14:07:17	8	10,0	10,0	1.000	1.000	1.000	1.000
10	11 1.2.2 Android	100% grün	Niklas Graf	14.07.17 20:07:17	14:07:17	20:07:17	7	10,0	10,0	1.000	1.000	1.000	1.000
11	12 1.2.3 Datenvisualisierung	100% grün	Sebastian Mandl	07.07.17 21:07:17	07:07:17	21:07:17	15	15,0	15,0	1.500	1.500	1.500	1.500
12	13 1.2.4 OpenHAB	100% grün	Sebastian Mandl	12.07.17 20:07:17	12:07:17	20:07:17	9	10,0	10,0	1.000	1.000	1.000	1.000
13	14 1.2.5 Application-Server Java REST	100% grün	Lukas Knoll	21:07:17 21:07:17	0	21:07:17	0	0	0	0	0	0	0
14	15 1.2.6 Einarbeitung abgeschlossen	100% grün	Lukas Knoll	13.07.17 31:01:18	25:01:17	18:03:18	418	476,0	474,0	492,0	47.400	49.200	5.550
15	16 1.3 Implementierung	100% grün	Lukas Knoll	21:07:17 13:08:17	21:07:17	13:08:17	24	35,5	35,5	3.550	3.550	3.550	3.550
16	17 1.3.1 Aufbau Websitestruktur	100% grün	Lukas Knoll	07:08:17 07:10:17	07:08:17	10:10:17	62	25,5	25,5	2.550	2.550	2.550	2.550
17	18 1.3.2 Android-App Struktur	100% grün	Lukas Knoll	29:08:17 31:08:17	29:08:17	31:08:17	3	7,0	7,0	700	700	700	700
18	19 1.3.3 Webinterface Administration	100% grün	Lukas Knoll	07:10:17 07:10:17	07:10:17	07:10:17	0	0	0	0	0	0	0
19	20 1.3.4 Fertiges User Interface Website und App	100% grün	Lukas Knoll	15:09:17 17:12:17	15:09:17	17:12:17	128	55,0	60,0	60,0	5.500	6.000	6.000
20	21 1.3.5 Interagieren der Webstruktur	100% grün	Lukas Knoll	18:11:17 10:12:17	18:11:17	10:12:17	26	14,0	14,0	1.400	1.400	1.400	1.400
21	22 1.3.6 Notifications Android	100% grün	Lukas Knoll	17:12:17 10:01:18	12:01:18	15:03:18	63	30,0	30,0	3.000	3.000	3.000	3.000
22	23 1.3.7 Android-App Server-, Datenbankkommunikation	100% grün	Lukas Knoll	10:01:18 14:01:18	22:02:18	22:02:18	7	5,0	10,0	500	1.000	1.000	1.000
23	24 1.3.8 Bugfixes und Testen	100% grün	Niklas Graf	21:07:17 08:08:17	21:07:17	08:08:17	19	19,0	19,0	1.900	1.900	1.900	1.900
24	25 1.3.9 Fertige Kommunikationsstruktur Website und Java REST	100% grün	Niklas Graf	09:08:17 09:12:17	09:08:17	22:02:18	198	70,0	55,0	55,0	7.000	5.500	5.500
25	26 1.3.10 Report-Bot	100% grün	Niklas Graf	20:12:17 13:01:18	20:02:18	07:03:18	16	30,0	20,0	2.000	2.000	2.000	2.000
26	27 1.3.11 Statistiken	100% grün	Niklas Graf	13:01:18 13:01:18	07:03:18	07:03:18	0	0	0	0	0	0	0
27	28 1.3.12 Reports	100% grün	Niklas Graf	07:08:17 07:10:17	07:08:17	10:10:17	62	25,5	25,5	2.550	2.550	2.550	2.550
28	29 1.3.13 Erstellen von Statistiken und Berichten fertig	100% grün	Sebastian Mandl	13:07:17 20:01:18	13:07:17	20:01:18	231	55,0	55,0	5.500	5.500	5.500	5.500
29	30 1.3.14 Datenbank	100% grün	Sebastian Mandl	20:07:17 21:12:17	20:07:17	21:12:17	216	60,0	65,0	6.500	6.500	6.500	6.500
30	31 1.3.15 Kommunikation und Datenstruktur (Raspberry)	100% grün	Sebastian Mandl	16:08:17 15:11:17	16:08:17	15:11:17	92	25,0	25,0	2.500	2.500	2.500	2.500
31	32 1.3.16 Java REST	100% grün	Sebastian Mandl	15:10:17 14:01:18	15:10:17	14:01:18	131	25,0	31,0	3.100	3.100	3.100	3.100
32	33 1.3.17 Web-UI	100% grün	Niklas Graf	15:01:18 15:02:18	15:02:18	32	20,0	20,0	20,0	2.000	2.000	2.000	2.000
33	34 1.3.18 Funktionalität Admin-Webpage	100% grün	Niklas Graf	20:01:18 28:02:18	28:02:18	0	0	0	0	0	0	0	0
34	35 1.3.19 Fertigstellung Projekt	100% grün	Lukas Knoll	22:01:18 25:01:18	25:01:18	18:03:18	418	20,0	20,0	2.000	2.000	2.000	2.000
35	36 1.3.20 Deployment	100% grün	Niklas Graf	12:01:18 14:01:18	14:01:18	14:01:18	2	2,0	2,0	200	200	200	200
36	37 1.4 Testen	100% grün	Niklas Graf	13:01:18 14:01:18	13:01:18	14:01:18	2	2,0	2,0	200	200	200	200
37	38 1.4.1 BOT-Tests	100% grün	Niklas Graf	12:01:18 12:01:18	14:01:18	14:01:18	0	0	0	0	0	0	0
38	39 1.4.2 Fertigstellung Testen	100% grün	Lukas Knoll	07:07:17 20:01:18	07:07:17	20:01:18	252	50,0	47,0	4.700	4.700	4.700	4.700
39	40 1.5 Dokumentation	100% grün	Lukas Knoll	10:12:17 14:01:18	01:02:18	17:02:18	17	15,0	13,0	1.300	1.300	1.300	1.300
40	41 1.5.1 Dokumentation Knoll	100% grün	Niklas Graf	07:07:17 14:01:18	07:07:17	15:03:18	252	15,0	9,0	900	900	900	900
41	42 1.5.2 Dokumentation Graf	100% grün	Sebastian Mandl	10:12:17 14:01:18	10:12:17	15:03:18	96	15,0	12,0	1.200	1.200	1.200	1.200
42	43 1.5.3 Dokumentation Mandl	100% grün	Lukas Knoll	01:02:18 01:03:18	01:02:18	01:03:18	5	8,0	8,0	800	800	800	800
43	44 1.5.4 Formulierung der Dokumentation	100% grün	Lukas Knoll	01:01:18 01:03:18	01:03:18	01:03:18	15	5,0	5,0	500	500	500	500
44	45 1.5.5 Zusammenführen der Dokumentation	100% grün	Lukas Knoll	20:01:18 05:03:18	20:01:18	05:03:18	0	0	0	0	0	0	0
45	46 1.5.6 Fertige Dokumentation	100% grün	Lukas Knoll	20:01:18 20:01:18	05:03:18	05:03:18	24	5,0	5,0	500	500	500	500
46	47 1.6 Einreichungen	100% grün	Lukas Knoll	20:02:18 20:02:18	05:03:18	05:03:18	24	5,0	5,0	500	500	500	500

## 7.2 Arbeitsbericht Niklas Graf - Ausschnitt

Arbeitsbericht		Gruppe:	AEMS - Advanced Energy Monitoring System		Name:	Niklas Graf
Datum dd.mm.jjjj	Einheiten für Projekt sonst	PSP	Tätigkeiten	PSP	geplant für nächste Woche	
SUMME	216,8	0			1.3.10	Report-Bot
01.08.2017	2	1.3.8	Report-Bot			
02.08.2017	2	1.3.1	Interagieren der Webstruktur			
04.08.2017	2	1.3.8	Report-Bot			
08.08.2017	2	1.3.8	Report-Bot			
10.08.2017	2	1.1.3	Projektkoordination (Treffen mit Auftraggeber)			
11.08.2017	2	1.3.8	Report-Bot			
12.08.2017	2	1.3.11	Statistiken - Erste Tests mit der Visualisierung der Daten mit Javascript und PDF Export			
13.08.2017	2	1.3.11	Statistiken - Wechsel auf Chart.js -> Visualisierung der Vorperiode			
15.08.2017	2	1.3.11	Statistiken -> Einfügen von Bildern in die Statistik (im PDF) ist möglich			
17.08.2017	4	1.3.11	Statistiken			
20.08.2017	3	1.3.11	Statistiken			
21.08.2017	4	1.3.11	Statistiken ERD Overhaul + Report & Warning ERD Erstellung + Übersetzung in pgSQL Code			
22.08.2017	3	1.3.11	Statistiken Reports Warnings ERD			
23.08.2017	1	1.1.3	Projektkoordination (Treffen mit Projektteam)			
25.08.2017	3	1.3.11	Statistiken			
26.08.2017	4	1.3.11	Statistiken			
07.09.2017	4	1.3.11	Statistiken Download - App			
13.07.2017	4	1.3.11	Statistiken			
15.09.2017	2	1.3.5	Anlegen des WebServers für das Aems Web Interface			
23.09.2017	2	1.3.5	Konvertierung der HTML zu XHTML Dateien - Fehlerbehebung:			
03.10.2017	5	1.3.5	Einarbeiten in XHTML für den Webserver			
06.10.2017	6	1.3.5	Erstellen der Managed Beans für das Login am Webserver			
07.10.2017	6	1.3.5	Umsetzen der Userdatenvalidation für den Webserver und URL-Rewrite mit PrettyFaces			
08.10.2017	3	1.3.5	Zählerdaten mit Ajax			
08.10.2017	1	1.1.3	Projektkoordination mit Projektleiter			
28.10.2017	3	1.3.11	Website			
04.10.2017	2	1.3.11	Website			
05.10.2017	4	1.3.11	Website			
08.10.2017	1	1.3.11	Website			
15.11.2017	1,5	1.3.11	Website			
16.11.2017	3	1.3.11	Website			
17.11.2017	3	1.3.11	Website			
22.11.2017	4	1.3.11	Website (JSF)			

### 7.3 Arbeitsbericht Lukas Knoll - Ausschnitt

Arbeitsbericht			Gruppe:	AEMS - Advanced Energy Monitoring System	Name:	Lukas Knoll
Datum	Einheiten	PSP	Tätigkeiten	PSP geplant für nächste Woche		
dd.mm.iii	für Projekt	Sonst				
<b>SUMME</b>	<b>222,57</b>	<b>0</b>				
08.07.2017	2,5		1.1.4 Projektcontrolling (Spezifikation)	1.2.1 Einarbeitung Webdevelopment		
10.07.2017	1,5		1.1.4 Projektcontrolling (Spezifikation)			
20.07.2017	0,67		1.1.3 Projektkoordination (Treffen mit Auftraggeber)			
21.07.2017	1		1.1.4 Projektcontrolling (Veraffassung Gesprächsprotokoll)			
22.07.2017	4		1.1.4 Aufbau Websitestruktur	1.3.1 Aufbau Websitestruktur		
25.07.2017	2		1.1.4 Aufbau Websitestruktur (Login, Logout, Register)			
26.07.2017	3		1.1.4 Aufbau Websitestruktur (notify.js)			
27.07.2017	2,5		1.1.4 Aufbau Websitestruktur (Login, Logout, Register)			
29.07.2017	6,5		1.1.3 Aufbau Websitestruktur (Benachrichtigungen, Einstellungen Berichte)	1.3.1 Aufbau Websitestruktur		
30.07.2017	3		1.1.3 Aufbau Websitestruktur (Einstellungen Startseite + Warnungen)			
01.08.2017	2		1.1.3 Aufbau Websitestruktur (AGB, Login, Einstellungen Berichte)			
02.08.2017	2		1.1.3 Aufbau Websitestruktur (statistiken.html und Datepicker)			
03.08.2017	4		1.1.3 Aufbau Websitestruktur (Adminpage und Bugfixes)			
03.08.2017	2		1.1.3 Aufbau Websitestruktur (Bugfixes im dynamischen Layout)			
06.08.2017	2		1.1.4 Projectcontrolling (Einrichtung Testserver mit Subdomain- Limacity)	1.3.1, 1.3.2 Webinterface, Androidapp-Struktur		
07.08.2017	2		1.1.3.2 Android App (Vertraut machen mit neuem Android Studio)			
10.08.2017	2		1.1.3 Projektkoordination (Treffen mit Auftraggeber)			
10.08.2017	1,5		1.1.3 Projectcontrolling (Verfassung Gesprächsprotokoll + Statusbericht)			
11.08.2017	1		1.1.3 Aufbau Websitestruktur (Adminpage Zuständigkeitsbereiche)			
12.08.2017	1		1.1.3 Aufbau Websitestruktur (Register - E-Mail und PLZ)			
13.08.2017	0,5		1.1.3 Aufbau Websitestruktur (Layout Bugfixes)			
13.08.2017	1		1.1.3.2 Androidapp-Struktur (Swipeen zwischen Statistiken)	1.3.2 Androidapp-Struktur		
15.08.2017	1		1.1.3.2 Androidapp-Struktur (Tabbed Layout)			
16.08.2017	2		1.1.3.9 Statistiken (Startseiten Formatierung)	1.3.2 Androidapp-Struktur		
22.08.2017	2		1.1.3.2 Androidapp-Struktur (Konfiguration der App-Inhalte)			
22.08.2017	2		1.1.3.2 Androidapp-Struktur (Tab Layout - Fragments)			
23.08.2017	1		1.1.3 Projektkoordination (Treffen des Projektteams)			
24.08.2017	2		1.1.3.2 Androidapp-Struktur (Chart Library)			
26.08.2017	4		1.1.3.2 Androidapp-Struktur (MPAndroidChartLibrary)	1.3.2 Androidapp-Struktur		
29.08.2017	4		1.1.3.2 Androidapp-Struktur (MPAndroidChart Combined Chart)			
30.08.2017	7		1.1.3.3 Webinterface Administration			
01.09.2017	1,5		1.1.3.2 Androidapp-Struktur (Chart Download)	1.3.2 Androidapp-Struktur		
03.10.2017	4		1.1.3.2 Androidapp (Login und Bugfixes)			

## 7.4 Arbeitsbericht Sebastian Mandl - Ausschnitt

Arbeitsbericht				Gruppe: AEMS - Advanced Energy Monitoring System				Name: Sebastian Mandl	
Datum	Einheiten	PSP	Tätigkeiten	PSP	geplant für nächste Woche				
dd.mm.jjjj	für Projekt								
SUMME	215,75	0							
08.09.2017	4	1.3.16	Erste Implementierungen von Abfragen mit GraphQL						Weitere Implementierungen von Abfragen mit GraphQL
09.09.2017	1	1.3.16	Bugfixing von bisherigen Abfragen mit GraphQL						Weitere Implementierungen von Abfragen mit GraphQL
14.09.2017	1,5	1.3.16	GraphQL - weitere Abfragen						Raspberry PI GUI: GraphQL
14.09.2017	0,5	1.3.14	Adaptierung der AEMSDatabase Lib						Weitere GraphQL Abfragen + Raspberry PI GUI
15.09.2017	1	1.3.16	Weitere GraphQL Abfragen						REST PUT, DELETE
20.09.2017	2	1.3.16	GraphQL - weitere Abfragen (fertig)						Adaptierung der AEMSDatabase Lib
20.09.2017	0,5	1.3.14	Datenbankmodell erweiterung + Implementierung neuer Relationen						User Validation + Multi-Thread Read
23.09.2017	5	1.3.15	GUI Raspberry PI						Bug Fixing POST Request
04.10.2017	0,5	1.1.3	Projektkoordination (Besprechung mit Betreuungslehrer)						
06.10.2017	1,5	1.3.14	Encryption Implementation (hoffentlich) vollendet						
06.10.2017	0,5	1.3.16	Testing of POST Request						
11.10.2017	2	1.3.16	RestAPI PUT Methode						
15.10.2017	2	1.3.17	Benachrichtigungsalgorithmus						
16.10.2017	1,5	1.3.17	Benachrichtigungsalgorithmus fast fertig						
16.10.2017	1	1.3.14	Adaption der Datenbankstruktur						
18.10.2017	1	1.3.14	Benachrichtigungsalgorithmus fertig + Kaskadierende Übertritte						
29.10.2017	2	1.3.17	Benachrichtigungsalgorithmus um Archivierung Erweitert + Konzeptüberlegung						
03.11.2017	1	1.3.16	GraphQL						
03.11.2017	0,25	1.3.15	OpenHAB						OpenHAB Queries
17.11.2017	2	1.3.15	OpenHAB						Plugin-System
18.11.2017	3,5	1.3.15	Umsetzung Plugin-System fertig						Uploader
19.11.2017	2	1.3.15	Basic Uploader ohne Security						Security für Uploader + GUI Adaptionen bei Raspberry Software
21.11.2017	3	1.3.15	Raspberry PI Update on GUI						Save and Load Configurations
22.11.2017	2,5	1.3.15	Save and Load Configurations						ERD
25.11.2017	1	1.3.14							
06.12.2017	1,5	1.3.15							OPENHAB Fertig
13.12.2017	2	1.3.15							Raspberry PI Sensor Plugins + UI
27.12.2017	1	1.1.3							Projektkoordination mit Projektteam
28.12.2017	4	1.3.14							Encryption bei REST + DB Sequences and Enhancements
29.12.2017	4	1.3.14							Encryption bei REST fertig
02.01.2018	1,5	1.3.14							Login Packet Rest
03.01.2018	5	1.3.14							Authorization Rest
05.01.2018	2	1.3.14							Authorization Rest

## 7.5 Begleitprotokoll

# Begleitprotokoll

## BEGLEITPROTKOLL DER SCHÜLERIN/DES SCHÜLERS

Schuljahr: 2017/18

Klasse: 5. BHIF

Thema des Projekts:	<b>AEMS – Advanced Energy Monitoring System</b>
Name der Betreuerin/des Betreuers:	<b>DI Josef Doppelbauer</b>
Auftraggeber:	<b>Ing. Herbert Pölzberger, MSc, Energiegenossenschaft Eferding</b>
Teammitglieder:	<b>Lukas Knoll, Niklas Graf, Sebastian Mandl</b>
Protokollführer:	<b>Lukas Knoll</b>

Datum	Teilnehmer	Besprochenes
20.07.2017	Knoll Lukas Mandl Sebastian Graf Niklas Pölzberger Herbert	<p>Besprechung, Adaption und Abnahme der Spezifikation: Im Großen und Ganzen war die Spezifikation in Ordnung. Änderung des Speicherintervalls der Zählerdaten in der Datenbank von halbstündlichen auf viertelstündliche Werte. Neben Strom-, Wasser-, und Gaszählern soll es auch möglich sein die Daten von Wärmemengenzählern auszulesen.</p> <p>Zählertausch: Die Zählpunktnummer des neuen Zählers bleibt gleich. Daher kann der Zähler einfach getauscht werden.</p> <p>Klärung von Fragen, wie z.B: Userverwaltung für Login, Verwendung eines Wetterdienstes für Anomalieerkennung, Verbrauchswerte abhängig von Tages-, und Jahreszeit.</p> <p>Der Punkt „Speicherpunkte“ blieb noch offen, da dieser etwas unklar ist.</p> <p>Anmerkung von Herrn Pölzberger, dass das Projekt eingereicht werden soll, um das System österreichweit verwenden zu können.</p> <p>Gesprächsdauer: ca. 40 Minuten</p>
10.08.2017	Knoll Lukas, Graf Niklas Pölzberger Herbert	<b>Vorführung des Website Layouts</b> Im Großen und Ganzen alles ok. Entfernen der Anomalie für die Tageszeit.

		<p>Möglichkeit zur Konfiguration von verschiedenen Zählertypen in einem Gebäude (in einer Statistik)</p> <p><b>Besprechung diverser Fragen</b></p> <p>Was soll alles in der Statistik stehen? Statistik, mit herausgehobenen Feldern bei gewählter Anomalie, oder Warnung.</p> <p>Welche Arten von Statistiken soll es geben? Balken und Verlaufsstatistiken.</p> <p>Was soll ein Bericht alles beinhalten? Verschiedene Statistiken zu verschiedenen Zählertypen. Bsp... Gas, Wasser und Strom.</p> <p>Wie geschieht die Zuteilung, welcher Administrator welche Nutzungsanfragen erhält? Überprüfung anhand der Postleitzahl des Antragstellers.</p> <p>Welchen Inhalt soll die Android-App besitzen? Nur die Möglichkeit sich seine Statistiken und Warnungen anzeigen zu lassen. Keine Konfiguration von Berichten, Statistiken oder Warnungen.</p> <p>Von wem und wie werden die Nutzungsbedingungen erstellt? Von dem Projektteam in Anlehnung an die Nutzungsbedingungen der Netz-Online Website.</p> <p>Gesprächsdauer ca. 40 Minuten</p>
4.10.2017	Knoll Lukas Mandl Sebastian Graf Niklas Doppelbauer Josef	<p>Gespräch über den Fortschritt des Projekts, Klärung von Unklarheiten, Vorführung des Weblayouts und Besprechung der Funktionalität.</p> <p>Der Fortschritt des Projekts ist gut.</p> <p>Klärung der Frage, wie die Daten der Zähler (Strom, Gas, Wasser, Wärmemenge,...) über Raspberry Pi's ausgelesen werden können, da diese verschlüsselt aus den Zählern heraus kommen – Code zum Entschlüsseln kann bei NetzOnline bzw. EnergieAG angefragt werden → Vorlage zum Daten auslesen ist die Diplomarbeit „Smart Meter Integration“.</p> <p>Besprechung des Themas Speicherpunkte. Es soll möglich sein Speicherpunkte zu erstellen. Das heißt, dass auch Zählertypen angelegt werden können, welche noch nicht existieren bzw. programmiertechnisch umgesetzt wurden. Hier soll es möglich sein den Speicherpunkten bzw. „virtuellen Zählern“ bereits Daten wie Name, Standort, Einheit, Typ,... zu geben. Dies hat den Sinn, dass das System modular erweiterbar ist. Dieser Punkt ist als Erweiterung geplant und kein Pflichtziel.</p> <p>Möglichkeit zur Erstellung von eigenen „Anomalien“. Bsp. Helligkeitssensor für Raspberry Pi, anhand einer Skriptsprache. Dieser Punkt ist als Erweiterung geplant und kein Pflichtziel.</p> <p>Benachrichtigungen/Warnungen: Hier soll Rücksicht auf Anomalien (Außentemperatur, Jahreszeit, ...) genommen werden können. Zählerbezeichnung soll als Benachrichtigung auf der Website mit angezeigt werden, wenn der Stromverbrauch den festgelegten Rahmen übersteigt. Abweichung des Verbrauchs soll auch bei zu</p>

		niedrigem Verbrauch an den User gemeldet werden.  Gesprächsdauer: ca. 30 Minuten
25.10.2017	Knoll Lukas Mandl Sebastian Graf Niklas Doppelbauer Josef	Gespräch über den Fortschritt des Projekts. Next-Project muss überarbeitet und auf neuesten Stand gebracht werden. Mandl muss sich das Raspberry Pi-Image bei Herrn Doppelbauer abholen, um an der Diplomarbeit weiterarbeiten zu können. Termin für nächstes Treffen wurde für Donnerstag 9. November festgelegt. Ansonsten keine Fragen oder Unklarheiten.
09.11.2017	Knoll Lukas Mandl Sebastian Graf Niklas Doppelbauer Josef	Gespräch über den Fortschritt des Projekts. Kurzer Informationsaustausch zwischen Mandl und Prof. Doppelbauer über das Raspberry PI Image. Gemeinsames Durchsehen der NextProject-Planung. Ergebnis: Planung ist in Ordnung. Kurzes Gespräch, ob wir bei der Umsetzung Probleme haben und wie gut der Projektfortschritt ist. Ergebnis: Projektfortschritt liegt im Plan und es gibt keine Unklarheiten oder Probleme.
30.11.2017	Knoll Lukas Mandl Sebastian Graf Niklas Doppelbauer Josef	Nächster Gesprächstermin wurde für den 23.November um ca. 11:40 vereinbart.  Gesprächsdauer: ca. 10 Minuten
11.12.2017	Knoll Lukas Mandl Sebastian	Gespräch über den Fortschritt und die Funktionen des Projekts. Alle bereits vorhandenen Komponenten wurden dem Auftraggeber

	Graf Niklas Pözlberger Herbert	<p>vorgeführt und erklärt:</p> <ul style="list-style-type: none"> <li>• Webinterface Endbenutzer</li> <li>• Webinterface Administration</li> <li>• Android-Application</li> <li>• Userinterface für Raspberry-PI-Zähler</li> </ul> <p>Es wurde die Sinnhaftigkeit eines 3stufigen Administrationsverfahrens besprochen. Es macht jedoch keinen Sinn das Administrationsstool 3-stufig auszubauen (Administrator, Bundeslandadministrator, Administrator für Enduser). Deshalb wird das Administrationsstool zweistufig bleiben.</p> <p>Es wurde ausgemacht, dass der Auftraggeber, dem Projektteam die AGB's für die Nutzung des AEMS-Systems zukommen lässt.</p> <p>Dem Auftraggeber wurden Vorabversionen des Diplomarbeitsfolders und des Plakats übergeben.</p> <p>Der Deploymentprozess wurde kurz besprochen. Ergebnis: Der Auftraggeber lässt dem Projektteam Daten (Nutzeranzahl, Anzahl der Speicherpunkte,...) zukommen, damit diese den bestmöglichen Server für das System finden können.</p> <p>Gesprächsdauer: ca. 40 Minuten</p>
14.12.2017	Knoll Lukas Doppelbauer Josef	<p>Besprechung über den Fortschritt und die Funktionen des Projekts. Die Diplomarbeit liegt gut in der Zeit und es sind ca. 2/3 der Arbeit erledigt.</p> <p>Kurze Vorführung der Androidapp-Funktionen (Statistiken, Statistik-Download, Notifications).</p> <p>Besprechung über den Inhalt der „Speicherpunkte“. Speicherpunkte sollen am Raspberry PI angelegt werden können und es soll keine Beschränkung auf von uns umgesetzte Zählertypen geben.</p> <p>Niklas Graf und Sebastian Mandl besuchten das Gespräch aus unbekannten Gründen nicht, obwohl diese in der Schule anwesend waren.</p> <p>Gesprächsdauer: ca. 20 Minuten</p>
27.12.2017	Knoll Lukas, Graf Niklas, Mandl Sebastian	<p>Besprechung der Schnittstelle und Kommunikation zwischen Android-App, Webinterface und Datenbank/Server (Java REST API, JSON, GraphQL)</p> <p>Besprechung der Verschlüsselung der Daten und des Datenaustausches – Aus Nutzerdaten wird am Client und Server ein Hash gebildet und Verglichen – zur Verbesserung der Sicherheit mit Salt.</p> <p>Besprechung des Algorithmus für die Erkennung der Verbrauchsabweichungen – Folglich Besprechung des Notification-Systems.</p> <p>Diskussion über die Sinnhaftigkeit, wenn Statistiken bereits am Server erstellt werden.</p>

		<p>Besprechung der Funktionsweise der Funktion „Mit Vorperiode vergleichen“ im Menü Statistiken. Es soll die Möglichkeit geben sich die vorhergehende Periode (z.B. Vorwoche) oder den Verbrauchswert von diesem Datumsbereich aus dem Vorjahr einbinden zu lassen.</p> <p>Erstellen des Anforderungsprofils für den Dependency-Editor: Auf Startseite soll es die Möglichkeit geben sich aussuchen zu können, welche Anomalien (Temperatur, Helligkeit,...) in der Statistik angezeigt werden sollen.</p> <p>Gesprächsdauer: ca. 1 Stunde</p>
11.01.2018	Knoll Lukas, Graf Niklas, Mandl Sebastian, Doppelbauer Josef	<p>Besprechung über den Fortschritt und die Funktionen des Projekts.</p> <p>Alle bereits vorhandenen Dienste (Java REST, Datenbank, ...) sollen auf dem Schulserver probeweise installiert werden, um deren Funktionalität zu testen. Dafür soll Herrn Doppelbauer eine entsprechende VM übergeben werden.</p> <p>Wenn fertig, soll auch noch der Raspberry getestet werden.</p> <p>Das nächste Treffen wurde für in zwei Wochen (25.1.2018) vereinbart.</p> <p>Gesprächsdauer: ca. 20 Minuten</p>
11.01.2018	Knoll Lukas, Graf Niklas, Mandl Sebastian, Doppelbauer Josef	<p>Gespräch über den Projektfortschritt.</p> <p>Besprechung der Dokumentationsrichtlinien (Gendern, Seitenanzahl, Anzahl der Druckexemplare).</p> <p>Information darüber, dass eine vorläufige VM vorbereitet wurde und am Schulserver installiert werden kann.</p> <p>Nach Installation der VM sollen über den BOT Daten von der Netzonline Website abgerufen werden und man die Kommunikation testen können.</p> <p>Nächste Gespräch wurde für den 8.2.2018 vereinbart.</p> <p>Gesprächsdauer: ca. 20 Minuten</p>
08.02.2018	Knoll Lukas, Graf Niklas, Mandl Sebastian	Gespräch wurde von Betreuungslehrer Herrn Doppelbauer abgesagt und auf einen späteren Zeitpunkt verschoben.
*	Knoll Lukas, Graf Niklas, Mandl Sebastian, Doppelbauer Josef	Weitere kurze Gespräche, bei welchen die Fertigstellung der Diplomarbeit besprochen wurde.
<u>25.03.2018</u>		<u>Lukas Knoll</u>
Datum		Schriftführer

## 7.6 Bildschirmmasken

### 7.6.1 AEMS

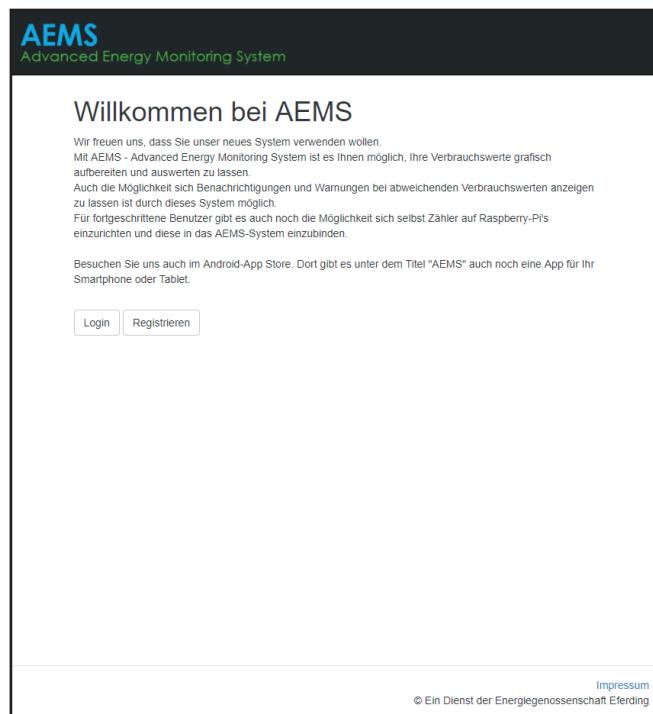


Abb. 92: Bildschirmmaske Startseite

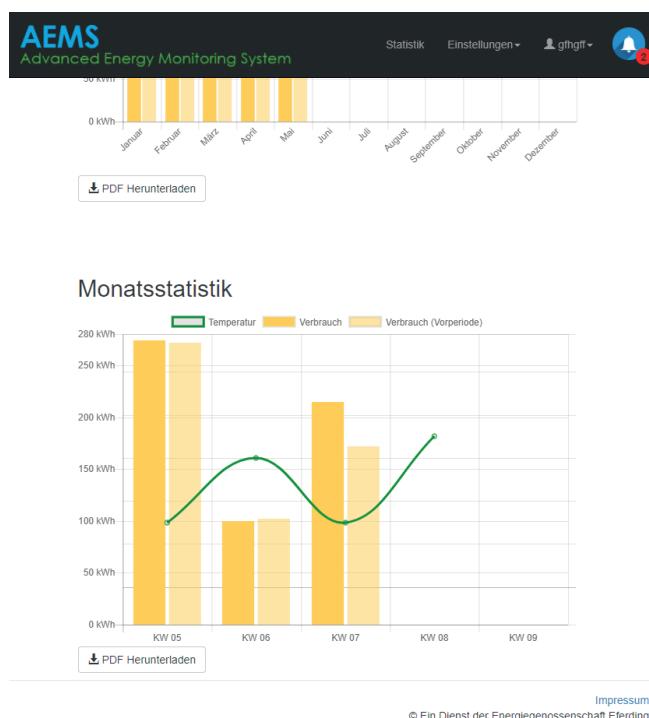


Abb. 93: Bildschirmmaske Startseite nach dem Login

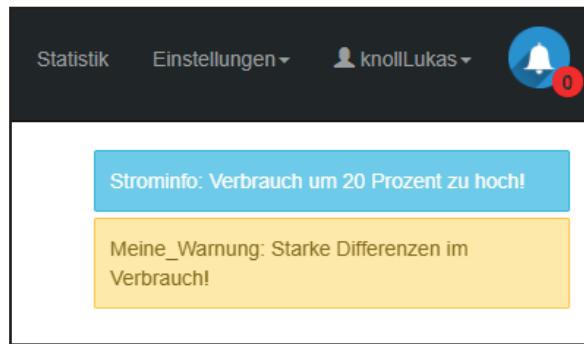


Abb. 94: Bildschirmmaske Notifications

Abb. 95: Bildschirmmaske Statistiken

Abb. 96: Bildschirmmaske Modal

## 7.6.2 AEMS - Admin

The screenshot shows the AEMS-Admin interface. At the top, there's a navigation bar with 'Administration', 'Zuständigkeitsbereich', a user icon 'admin', and a notification bell icon with a red '3' badge. Below the header, the main content area has two sections: 'Neue Anfragen (3)' (New Requests) and 'AEMS - Benutzer' (AEMS - User). The 'Neue Anfragen' section lists three items: 'Graf' (Anfrage vor wenigen Stunden gesendet), 'Knolli' (Anfrage vor 17 Tagen gesendet), and 'Mandl' (Anfrage vor über einem Monat gesendet). The 'AEMS - Benutzer' section shows a search input and a result for 'Josef Doppelbauer' with the email 'doppelbauer@gmx.net'. At the bottom right, there are links for 'Impressum' and '© Ein Dienst der Energiegenossenschaft Eferding'.

Abb. 97: Bildschirmmaske Administration Startseite

The screenshot shows the 'Administratoren' (Administrators) and 'Sub - Administratoren' (Sub-Administrators) sections of the AEMS-Admin interface. The 'Administratoren' section lists four users: 'Max Mustermann' (Zusatzinfo), 'Lukas Knoll' (Zusatzinfo), 'Niklas Graf' (Zusatzinfo), and 'Sebastian Mandl' (Zusatzinfo). The 'Sub - Administratoren' section lists three users: 'Josef Doppelbauer' (Zusatzinfo), 'Andreas Baumgartner' (Zusatzinfo), and 'Verena Moser' (Zusatzinfo). Both sections have a 'Zusatzinfo' link next to each user entry. At the bottom right, there are links for 'Impressum' and '© Ein Dienst der Energiegenossenschaft Eferding'.

Abb. 98: Bildschirmmaske Administratoren Verwaltung

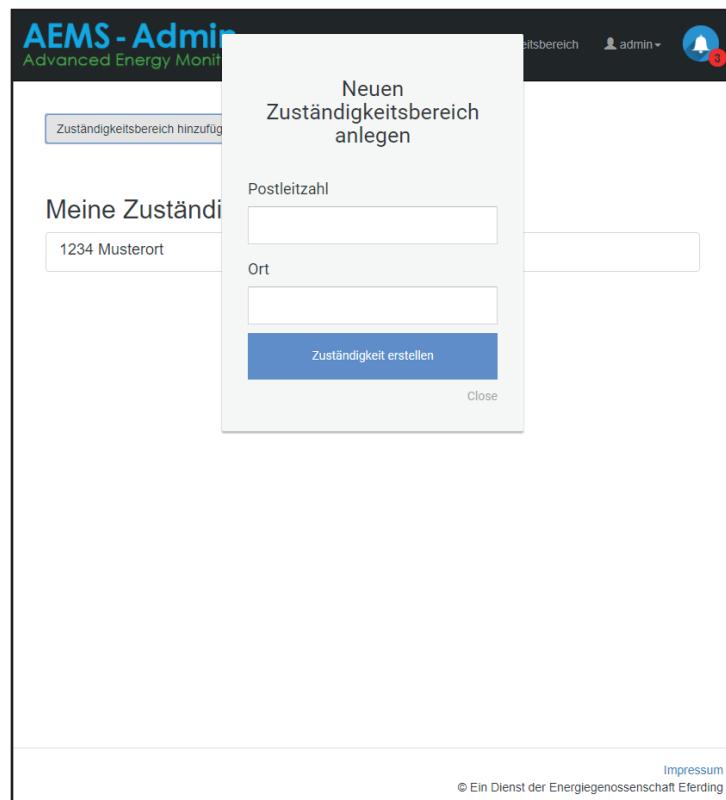


Abb. 99: Bildschirmmaske Definition der Zuständigkeitsbereiche

### 7.6.3 App

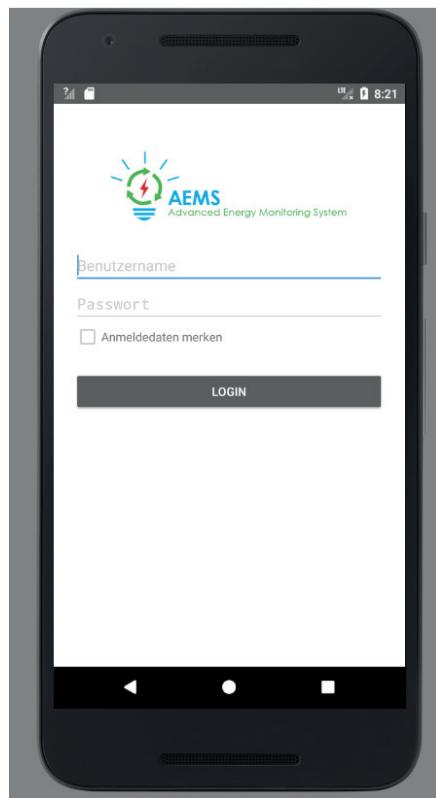


Abb. 100: BildschirmmaskeAppLogin

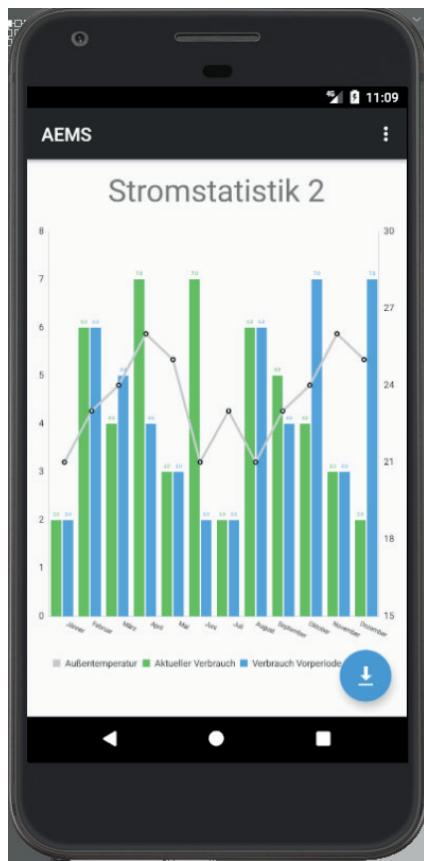


Abb. 101: Bildschirmmaske App Anzeige der Statistiken

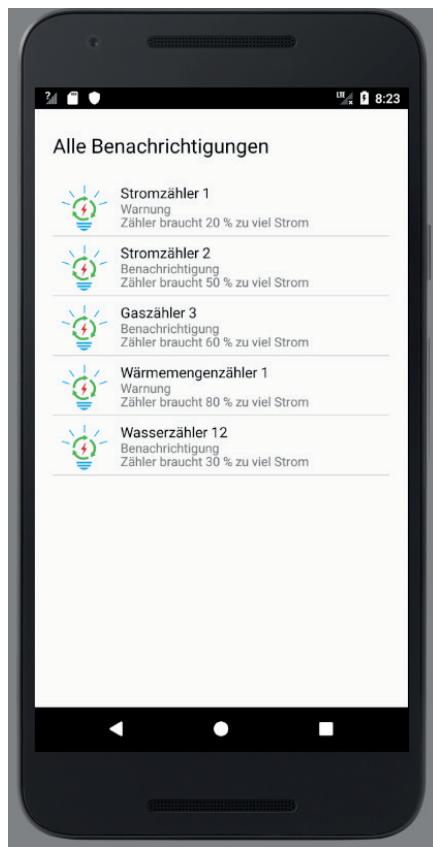


Abb. 102: Bildschirmmaske App Liste der Benachrichtigungen

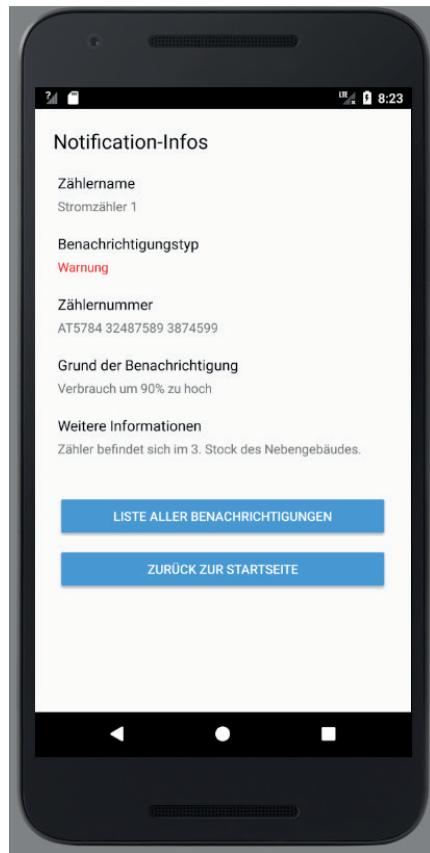


Abb. 103: Bildschirmmaske App BenachrichtigungsInformationen

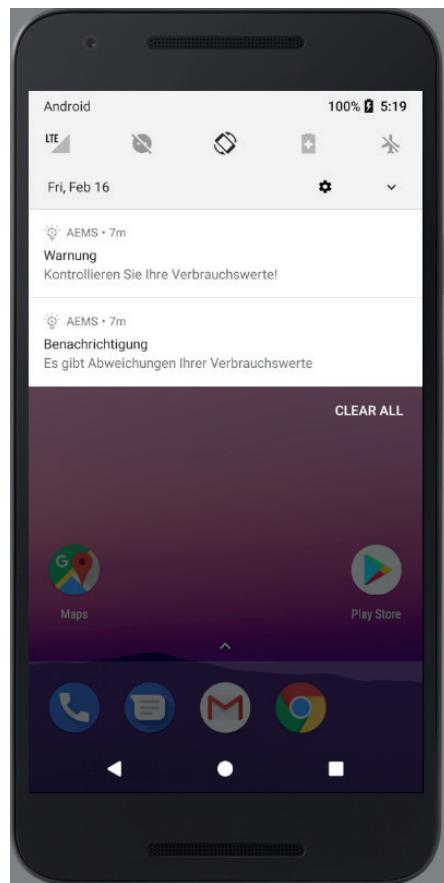


Abb. 104: BildschirmmaskeNotifications