BACHELOR THESIS

Test Data and Goal-Setting in Mjölnir

Manuel Wascher

matriculation number: 01061478

e-mail: mawasche@edu.aau.at

supervisor: Univ.Prof. Dipl.-Ing.Dr. Wilfried Elmenreich

Alpen-Adria-Universität Klagenfurt

Institute of Networked and Embedded Systems
Smart Grid Group

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

Klagenfurt, April 2021

Abstract

Today it is impossible for the western society to live without electrical energy. It is needed nearly everywhere (e.g., household, working place, ...). First of all, this lifestyle produces a lot of costs. Furthermore, if a lot of electrical energy is needed, this energy must be reproduced. Today, most of this energy will be produced by coal and atom power plants, which are harmful to the environment. To reduce these costs and the electrical energy production, it's beneficial to save this energy. Energy advisory systems are giving an overview, how much electrical energy is consumed in a household or an office, and how consumption could be reduced. An example for such a system is *Mjölnir*, which has been developed by the institute of *Networked and Embedded Systems* at the *Alpen-Adria-Universität Klagenfurt*. It is adapted especially for smart homes or offices. Among other information, energy consumption, production, and costs, will be shown on a screen for every household, or office. To get this information, specific measurement devices are used, which will measure the energy consumption, or production, of the electrical, or electronically used devices. This system is an open-source software, and can be expanded by every person, which will be desirable for the future. The needed measurement devices are also cost-efficient. If *Mjölnir* will be expanded in the future, it is important to test the new developed module beforehand. The following sections are describing, how test data can be inserted into this system through the *Test-Data-Input* module.

Contents

List of Figures

1. Introduction

The *Mjölnir* system is an energy advisory system, which stores consumed, or produced energy data. The analysed data will be shown on screen, through different program modules. This system should advice the users, how they can save electrical energy in their own households, or offices. [9]

Currently, *Mjölnir* consists of a couple of basic modules, but it is desirable for the future that this system can be expanded repeatedly. Because *Mjölnir* is an open-source project, every person has the possibility to change, and expand Mjölnir by themselves. This is relatively easy, because the system is organized in modules. A newly developed module, which is called a *widget*, can be simply added to *Mjölnir*.

1.1. Background

One current disadvantage of *Mjölnir* is that it is missing the possibility to test the entire system. At the moment, a developer can only test the system functionality, if measurement devices are installed inside a household, or an office. So a *widget* is beneficial, when real recorded test data is inserted into the *Mjölnir* system. Such modules provide the additional effect, that potential users are able to get an overview of the system functionality, without any hardware technical effort.

Furthermore, *Mjölnir* should support the user possibility, for constructive energy saving, through challenges. This can be achieved through the *Challenge-/Goal-setting* widget.

1.2. Scope

For this work the following two questions are important:

- How is it possible to minimize the hardware technical effort, when a system test is needed?
- Which possibilities can be used to motivate modern society to save energy constructively?

1.3. Outline

This report is structured into the following main sections, to answer the previous questions:

- Energy advisory system definition - gives a short overview about the whole system structure.

- Developing methods - shows the basic *Test-Data-Input widget* structure in figures. Furthermore, the *Test-Data-Input widget* program structure will be described in detail.
- Evaluation - shows the test results of the *Test-Data-Input widget* and also depicts the weak points of *Mjölnir*.
- Possible future widget - describes a process possibility of the *challenge-/goal-setting widget*. The whole widget implementation was beyond the scope of this *widget*. This section describes also, how good challenges are defined.
- Conclusion - gives a short summary of the *Test-Data-Input widget*, and also discusses the weak points of *Mjölnir* which should be approached in the future.

2. Energy Advisor System Definition

The function of such systems is to advise companies or single households about their own energy management. People shouldn't consume electrical energy unnecessarily, because today it has become more expensive. Furthermore, the most of the today's electrical energy is produced by coal and atom power plants, which are harmful to the environment. So, the main goal of these systems is to assist consumers in saving energy. This goal will be reached through web applications and the corresponding needed measurement devices. [6, p. 3-5]

The next few sections are describing this application structure which is assessed through the energy advisory system *Mjölnir*. It consists of a web-based graphical user interface, which is divided into different modules, a database in the background, which stores all important data, and multiple measurement devices. They are sending their energy information to the database with fixed timestamps.

2.1. Database

The *Mjölnir* database includes a list of tables, which represent altogether a relational database model. Through this model type, it's possible that all needed user actions can be recorded during a session. The most important data is the energy information, which is necessary for the *Mjölnir* system execution. Additionally, this database stores, all needed energy devices, circuits, rooms, and buildings. Through this data, the system is able to add energy information, mapping to a widget which is user-related. [6, p. 9-13]

To get an overview of the data model, and the table relations, the whole schema is displayed in *Figure 1*.

If the *Mjölnir* system is extended through one, or more widget modules, the data base will also become larger, this depends on if more data will be needed. The *Challenge/Goal-setting* widget needs additional tables, because the basic data base doesn't provide the possibility to store the needed user's session data for this module. A schematic overview, of this *Mjölnir* data base adaption, supplies the following table. It contains the added table names, and describes in short sentence their functionality. In section *Possible Future Widget: Challenge-/Goal-setting Widget* a more detailed description of this *widget* will be given.

Table name	Description
challenge_message	provides important needed messages, during a challenge session (e.g. challenge win message)

challenge_user_message	stores all related user messages, on a challenge day
challenge_energy_level	records all energy saving levels, one user can reach
challenge_level_user	gives a summary of the energy level of the current user, in that moment
challenge_incentive	lists all possible challenge incentives, related to the corresponding energy level (at the next level, the user can win more expensive incentives)
challenge	all basic challenges, that the administrator added in relation to that particular level to the system, are stored in this table (challenges mean, that these challenges will adapt to that particular user)
challenge_user	stores the whole challenge information, which is related to a user in a specific challenge level.
challenge_equal_devices	stores information about electrical device types, which are used in the *Challenge-/Goal-setting* widget

Table 1: List of the tables, which must be added to the *Mjölnir* data base, if the *Challenge-/Goal-setting* widget will be developed.

The section *Developing Methods*, specifies the correlation between the system data base and the widget *Test-Data-Input.*

2.2. The Mjölnir Graphic User Interface (GUI)

To handle the data inside the *Mjölnir's* data base, a *GUI* is needed. This graphical interface is a web application, and can be used by every common browser (e.g. *Chrome*, *Firefox*, ...). [6, p. 7]

It has the following structure:

- Every user can add one or more **buildings** to the default building, which will be applied in the database after his registration. One building represents a house abstraction (e.g. single family house, a multi-family house, or a big office) and consists of one or more rooms, so that this model can become more realistic for users. Every building is stored in the database table *user_buildings*. [6, p. 16]

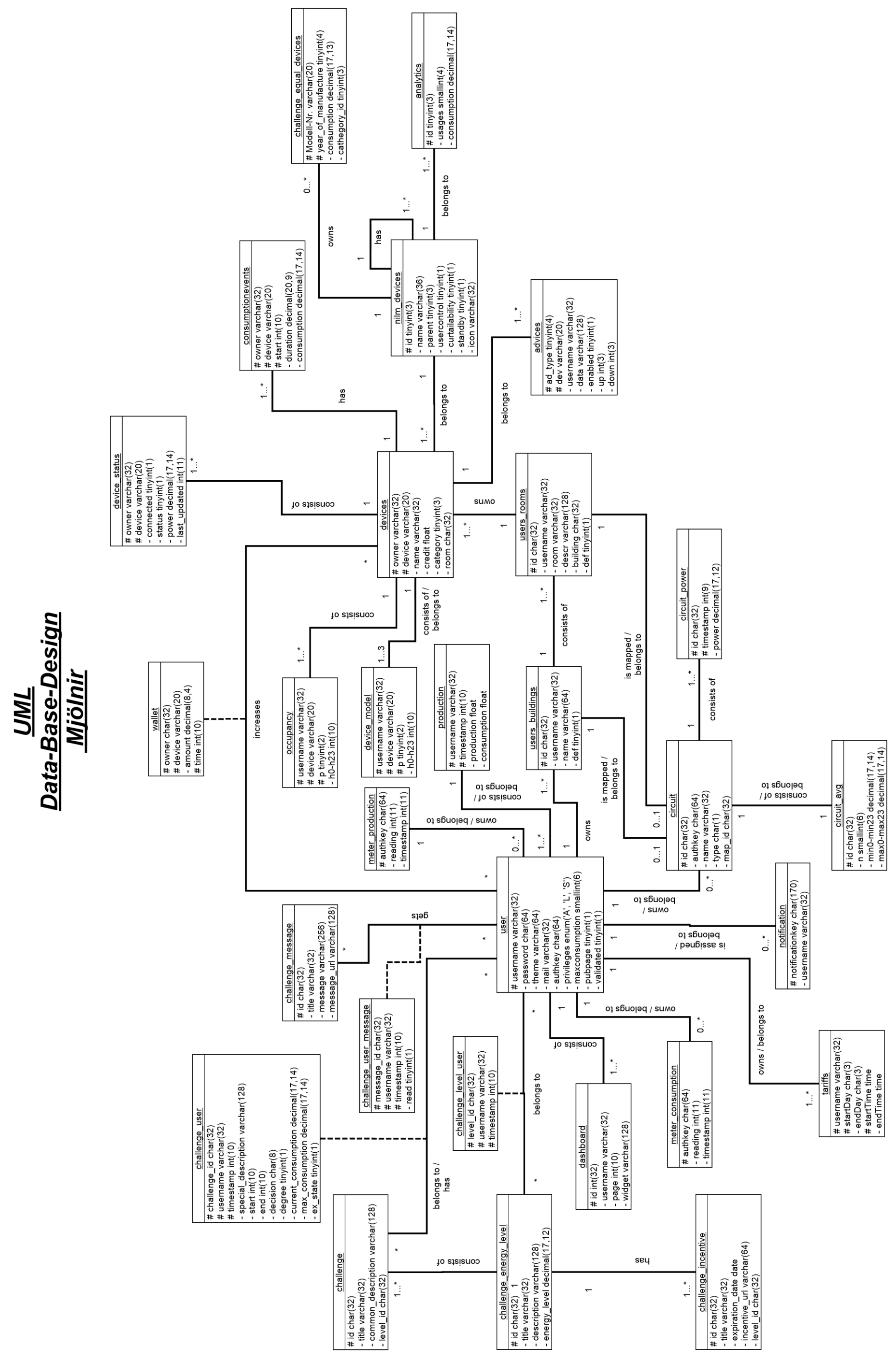


Figure 1: UML Mjölnir data base schema

- One **room** can store a group of devices, which are operating inside the same area. A good example is the living room, in which the most households have their TV, hi-fi unit, or computer. Users can group these together into one room. All rooms are stored in the table: *user_rooms*. [6, p. 16]

- Another way to group devices together are **circuits**, which have an unique *ID*. The column *map_id* contains either a user building, or room ID. This ID gives the member information about the rooms, or buildings inside a circuit. This data is stored in the table: *circuits*. [6, p. 16]

- The lowest GUI entity is the **device**. Every room consists of one, or more devices. They represent all the installed electronic devices inside a building, or household. The energy consumption, or production can be determined through special measurement devices, which will be described in more detail in the subsection *Used Measurement Devices*. [6, p. 16]

To handle these stored measurement device data, the *Mjölnir* system uses small program modules, called *widgets*. Some of these modules are using additional, tariff data, because they are computing, e.g. the monthly energy costs. If the user signs in for the first time, he has to enter his weekly household, or office tariff data. The next few paragraphs are describing some important *widgets*, to get a summary of the energy advice this application provides:

Production Report Widget

This widget gives a summary, of how much energy all current user buildings are producing every day. The energy display format is a histogram over thirty days with the physical unit *kWh*. This histogram displays an aggregated energy production, as shown in *Figure 2*. [6, p. 16-21]

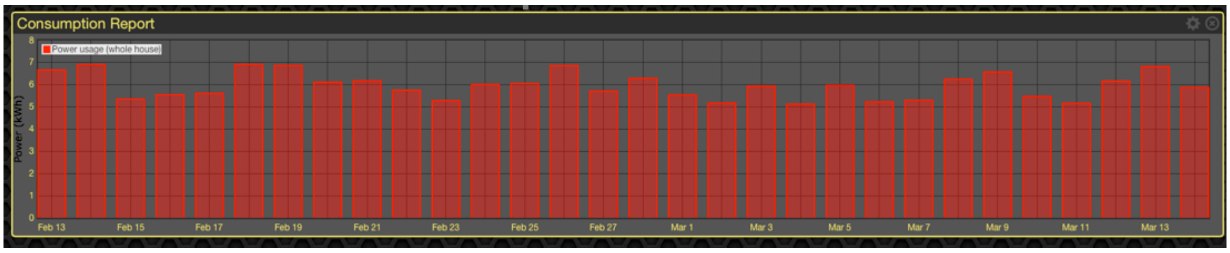


Figure 2: Consumption Report

Consumption Report Widget

The consumption report also shows a histogram, but only of the daily user energy consumption. This widget provides the aggregated, and disaggregated option.

This additional option approaches a real value energy consumption, because every new consumption event (e.g. every second, or minute), which is stored inside the database table *consumptionevents*, will be added to this histogram (q.v. *Figure 3*). [6, p. 16-21]

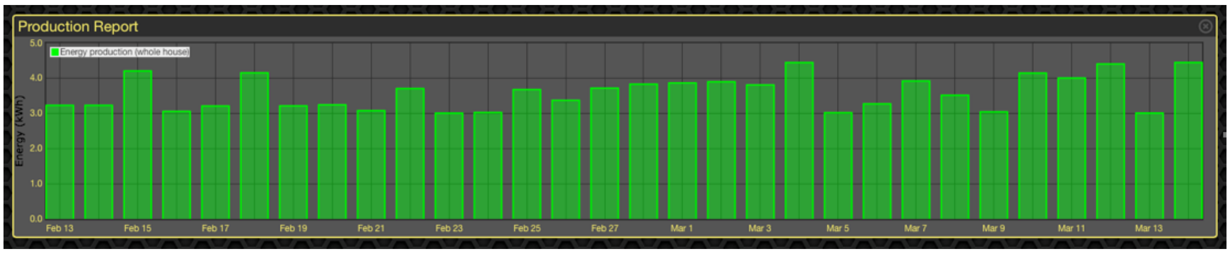


Figure 3: Production Report

Cost Report Widget

Also important for a household, or an office, are the device costs. This program module computes the daily energy device costs in euros, based on the tariff, the current user indicated at the first log-in. Every histogram bar shows the daily costs for the current month (q.v. *Figure 4*). [6, p. 16-21]

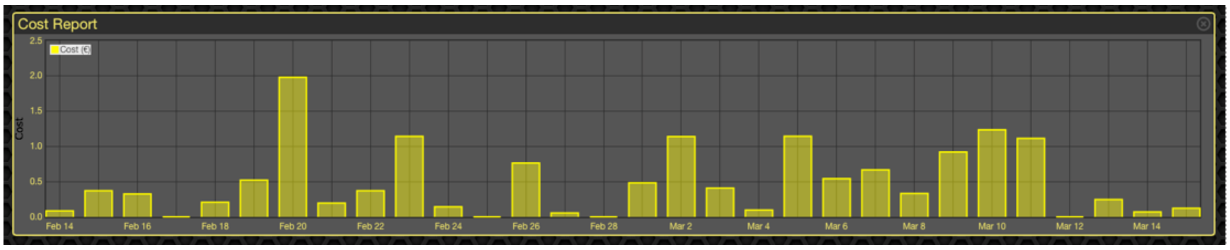
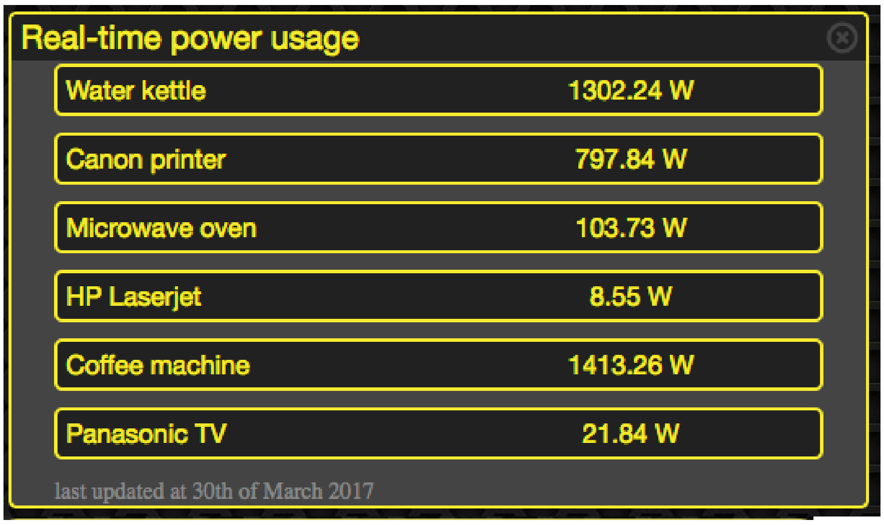


Figure 4: Cost Report

Real-time Power Usage Widget

Every day household members are using electronic devices like TVs, PCs, Notebooks, etc., but they don't know how much energy a certain device needs at that moment. This widget gives a real-time power overview for all added devices. If the user doesn't need one of these devices currently, he can switch it off, to save energy (q.v. *Figure 5*). [6, p. 16-21]


Real-time power usage

Water kettle	1302.24 W
Canon printer	797.84 W
Microwave oven	103.73 W
HP Laserjet	8.55 W
Coffee machine	1413.26 W
Panasonic TV	21.84 W

last updated at 30th of March 2017

Figure 5: Real-time power usage

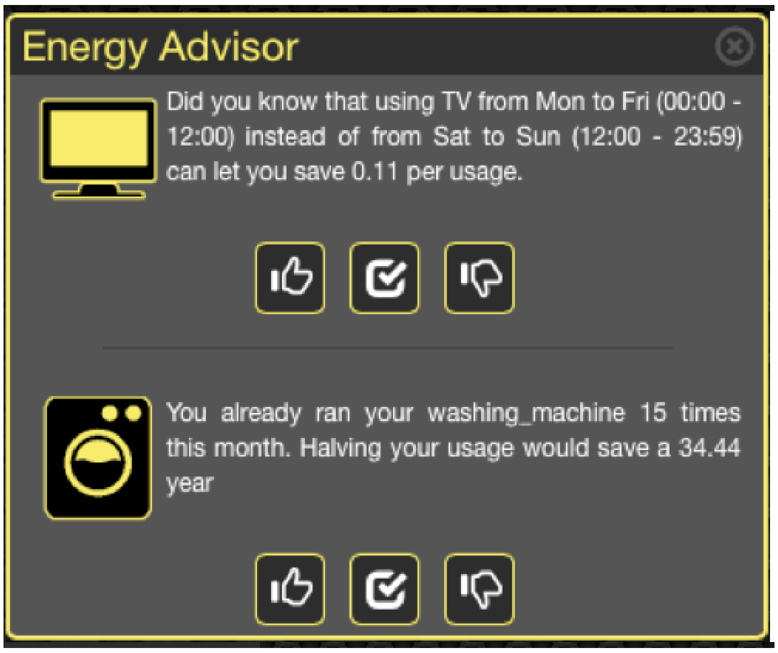
Energy Advisor Widget

The *Mjölnir* system analyzes all stored energy data in the background. It has to find out, if the current household has bad patterns for energy usage. This system generates advice which may help the users to save energy. *Mjölnir* uses different advice categories, and the customer must to choose which are the most helpful tips. The current user can evaluate every form of advice through three different answers, but he can only choose one:

- *OK, thanks!*,
- *I'm already doing it!*,
- or *No, thanks!*

If customers choose the last answer option, they can give *Mjölnir* feedback, whether they want to see the current device tip or not, or advice only this kind in general (q.v. *Figure 6*). [6, p. 16-21]

The above mentioned widgets are only a few, but there are many more. If users want to extend this system with more widgets, it won't be a problem to do this, because the whole source code is for free. Customers only have to purchase the measurement devices, which will be described in the next subsection. How users can add self-made widgets to *Mjölnir*, is explained in section *Developing Methods*.


Energy Advisor

Did you know that using TV from Mon to Fri (00:00 - 12:00) instead of from Sat to Sun (12:00 - 23:59) can let you save 0.11 per usage.

You already ran your washing_machine 15 times this month. Halving your usage would save a 34.44 year

Figure 6: Energy Advisor

2.3. Used Measurement Devices

Every electronic device, which the user wants to add in *Mjölnir*, has to be connected with a measurement device, which sends, via a network, the current energy data, with every fixed timestamp to the advisor system server. The server stores this data into the *Mjölnir* data base. There are different measurement device types [3], that an operator can use, but in general, they should be smart plugs, which support the *zigbee* communication protocol. [2, p. 3]

Potential measurement devices are the *Plug* of the company *plugwise*. How the gauge measure the current energy consumption, will be described in the next few lines.

The *Plug* works with the *zigbee* communication protocol, which is used for low data wireless networks (e.g. sensor networks). This device has three different application functions:

- electrical device switching,
- energy consumption measuring,
- and network communication. [1, p. 8-9]

Through these different kinds of modules, it's possible to create an own wireless *zigbee* network, because a *Plug* behaves like a *zigbee* router, which means that these devices are always reachable by other *zigbee* modules. Another operation possibility is, to use the *Plug* as a network gateway. This makes it possible, to reach other *zigbee* networks, and also the *Internet.* So the energy consumption data, can be stored at a server. These data sets will be stored in the *consumptionevents* table, of the *Mjölnir* data base. [1, p. 8-9]

Every technical device has a limit, for example the *Plug* must not exceed the power limit of 3680 Watt, or a current of 16A. [1, p. 8-9]

A total building current can be determined through an electric meter. In previous setups, *Mjölnir* has been used with the *Carlo Gavazzi EM24* [2]. The measured electrical energy data of the gauge can be inserted into the *meter_consumption* table.

3. Developing Methods

Every person can download the *Mjölnir* source code for free. The programming languages, which the development group used are *PHP*, and *Javascript*. This web application is only a basic program, which contains the most important widgets, but everyone can expand *Mjölnir* by themselves, if they are familiar with the programming languages. The next few subsections are describing the extension *Test-Data-Input* in detail.

If system operators are adding one or more widgets to *Mjölnir*, they have to test these modules as well. The test costs are significantly lower, if they can use test data instead of real measurement devices. That's the reason for developing the *Test-Data-Input* widget, which stores already recorded energy data into the data base table *consumptionevents*. Additional information about this test data is described in paragraph **??**.

3.1. Test-Data-Input Widget Process

Through this module, the *Mjölnir* system operators should be able to test their own developed modules, or the whole system, without using the needed measurement devices.

The next few paragraphs will describe, how the user can insert the test data sets through this widget. This module can only be executed, if the current user is the system administrator. In every other case, the following message will be shown on screen:

"This widget can be only used by the Mjölnir-Administrator!"

Through the *next-*, and *back-button* (q.v. *Figure 7*), the user can move forward, and backward inside the widget process. If this module is not needed anymore, it can be closed top right, by the *closed-button* (q.v. *Figure 8*).

Users

The widget user must be able to insert test data for multiple users, to test the *Mjölnir* system boundaries. Therefore, an user can be chosen. Then the energy data will be inserted for this specific user. Of course, a couple of users have to be added to this system beforehand.

Test Building

The whole test data set consists of multiple buildings, which are representing real households. The section, *The Mjölnir Graphic User Interface (GUI)* describes this in more detail.

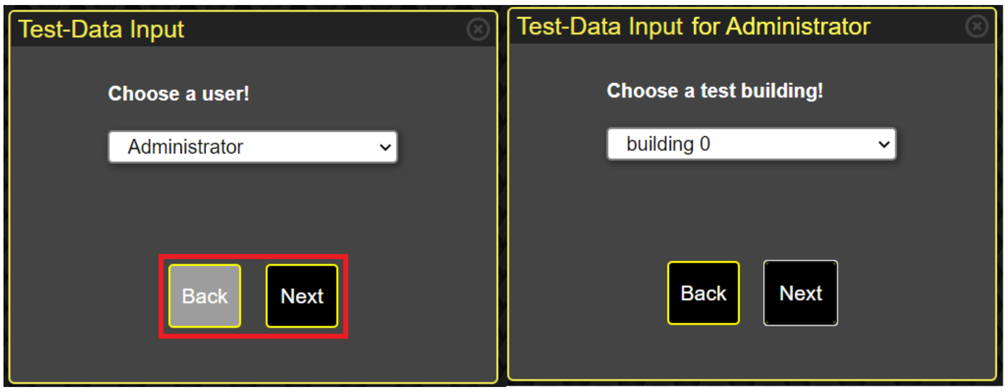
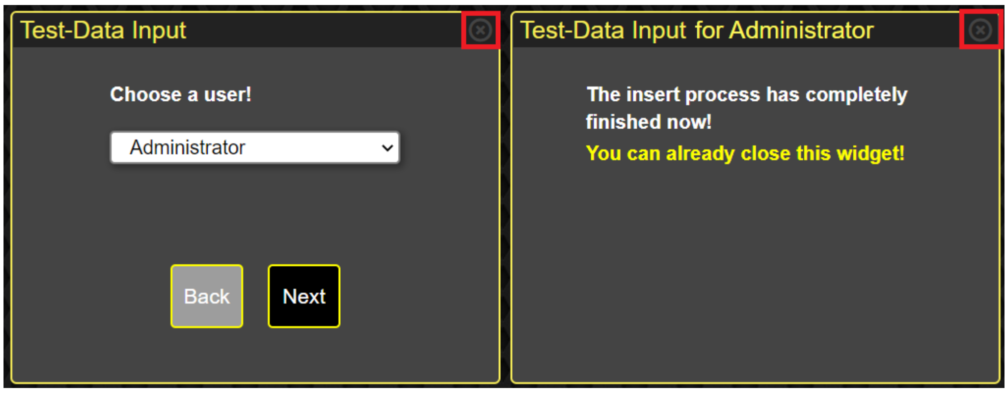


Figure 7: next and back; choose building



The insert process has completely finished now!
You can already close this widget!

Figure 8: close widget; widget is finished

Test File

If one test building has been chosen, two different cases can occur:

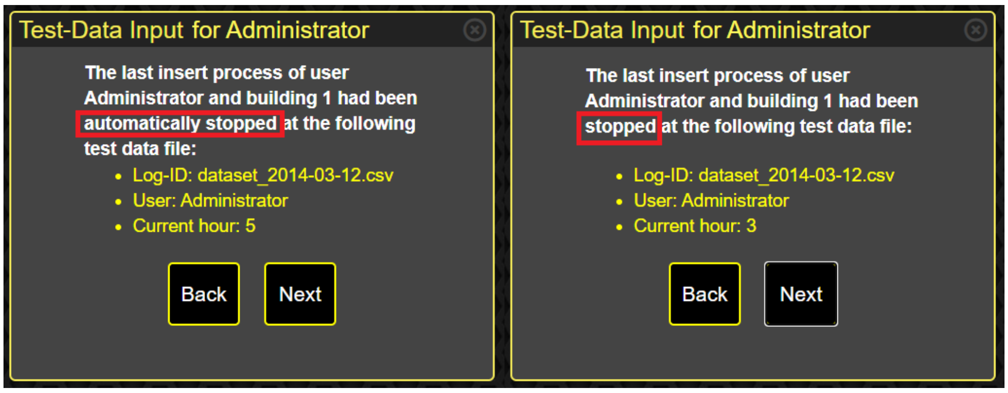
- A new test data file can be chosen (it represents one day),
- or the system can send information, about the last insert process of the current building, which had been stopped. (q.v. *Figure 9*)

The last insert process can be stopped, through two different ways:

- automatically - if the user doesn't want to insert the whole data file, he can then only chose a certain amount of hours, and after all the chosen hours have been inserted, the system will stop this process automatically.
- the *stop-button* - the user can press the *stop-button*, if he doesn't have enough time.

The stop message (means that the insert process has been terminated through the *stop-button*), contains three additional pieces of information:

- the last used data file,
- the last used user,
- and the next hour, where the continuing process will start.


Test-Data Input for Administrator

The last insert process of user Administrator and building 1 had been automatically stopped at the following test data file:

- Log-ID: dataset_2014-03-12.csv
- User: Administrator
- Current hour: 5

Test-Data Input for Administrator

The last insert process of user Administrator and building 1 had been stopped at the following test data file:

- Log-ID: dataset_2014-03-12.csv
- User: Administrator
- Current hour: 3

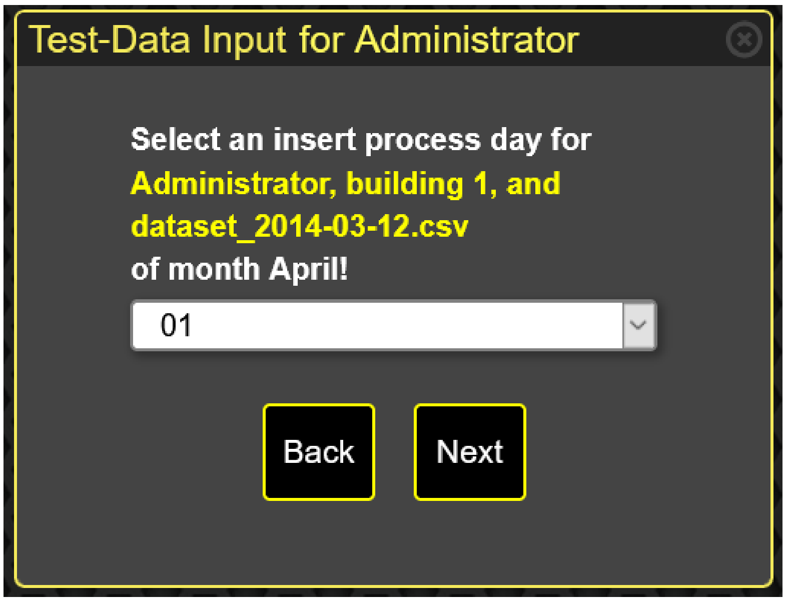
Figure 9: last insert process automatically stopped; last insert process stopped by hard

Day of Current Month

The admin can choose a day of the current month. For that particular day, the energy data of the chosen test data file will be inserted. (q.v. *Figure 10*)

List of Hours

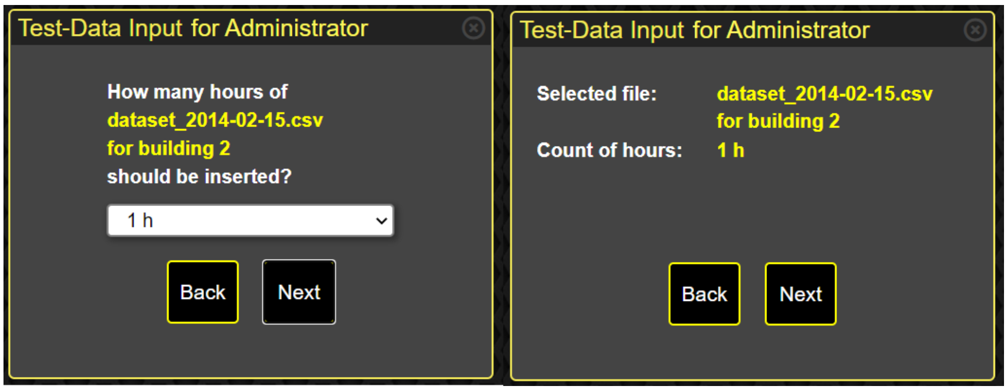
This view will be selected, if a data file has been chosen before. Every building data file represents one day, and this is noted in the subsection *Test file*. The administrator can select one hour, or up to a maximum of twenty four hours, that the *Mjölnir* system should insert into the data base immediately (q.v. *Figure 11*; the selected hour is called as ***process hour***). If the last insert process has been automatically stopped, the data file can be selected again, but the administrator can only continue from that given hour, which hadn't been inserted before.


Test-Data Input for Administrator

Select an insert process day for
Administrator, building 1, and
dataset_2014-03-12.csv
of month April!

01

Figure 10: insert process day selection


Test-Data Input for Administrator

How many hours of
dataset_2014-02-15.csv
for building 2
should be inserted?

1 h

Test-Data Input for Administrator

Selected file: dataset_2014-02-15.csv
for building 2
Count of hours: 1 h

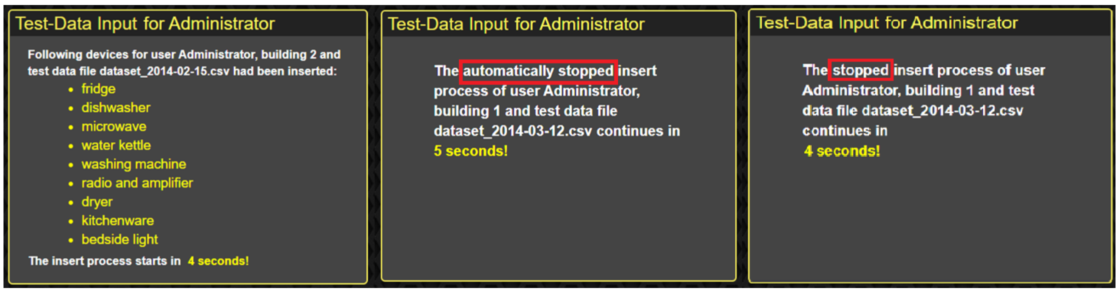
Figure 11: list of hours; selected data

Insert Process Preparation

If all needed insert process information is available, the *Mjölnir* client will wait for five seconds, and then the test data insert process will begin, but only if the selected building doesn't exist already, the *Mjölnir* system creates a new building, one room, and the related test devices automatically. Then all the current added devices are shown on this widget (q.v. *Figure 12*).

The Insert Process

During the data insert process, the following information will be displayed:



Test-Data Input for Administrator

Following devices for user Administrator, building 2 and test data file dataset_2014-02-15.csv had been inserted:

- fridge
- dishwasher
- microwave
- water kettle
- washing machine
- radio and amplifier
- dryer
- kitchenware
- bedside light

The insert process starts in 4 seconds!

Test-Data Input for Administrator

The automatically stopped insert process of user Administrator, building 1 and test data file dataset_2014-03-12.csv continues in 5 seconds!

Test-Data Input for Administrator

The stopped insert process of user Administrator, building 1 and test data file dataset_2014-03-12.csv continues in 4 seconds!

Figure 12: insert process prep. for the first time; insert process prep. after automatically stop; insert process prep. after hard stop

- the current chosen *building*, and the *test data file*,
- the last inserted device *consumptionevents* in *Watt*, at a special timestamp, which are embedded into a table (the first line relates to the previous *consumptionevents*, the second line to the current),
- a progress bar, which displays the whole data file insert process progress in percentage.

The whole process can be interrupted through the *BREAK-button*. During this time, the insert process screen is shown greyed out, and the following message will be displayed (q.v. *Figure 13*):

"The insert process have been interrupted! This process can be reactivated, if you press CONTINUE!"

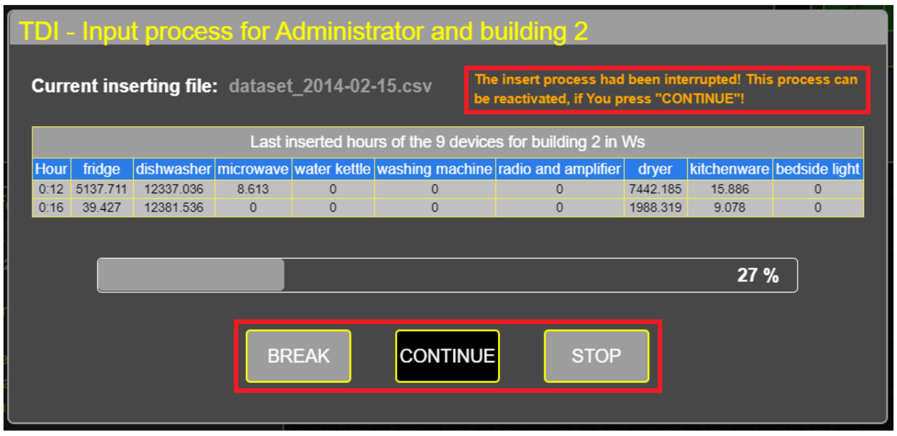
The *BREAK-* , and *STOP-button*, are disabled.

Through the *STOP-button*, this insert process will also be interrupted, but the view will be closed, and it has to be started all over again, as is described in the subsections before. After this button has been pressed, a message will be shown (q.v. *Figure 14*):

"Attention: The insert process had been stopped, and will be closed in 5 seconds!"

All the buttons are greyed out, and disabled.

No problems will show up, if the *insert process* is ended ahead of time, by closing the *web-browser*, or pressing the *refresh-button* inside the browser. At the next session, this *insert process* will start again, where it had been interrupted before.



TDI - Input process for Administrator and building 2

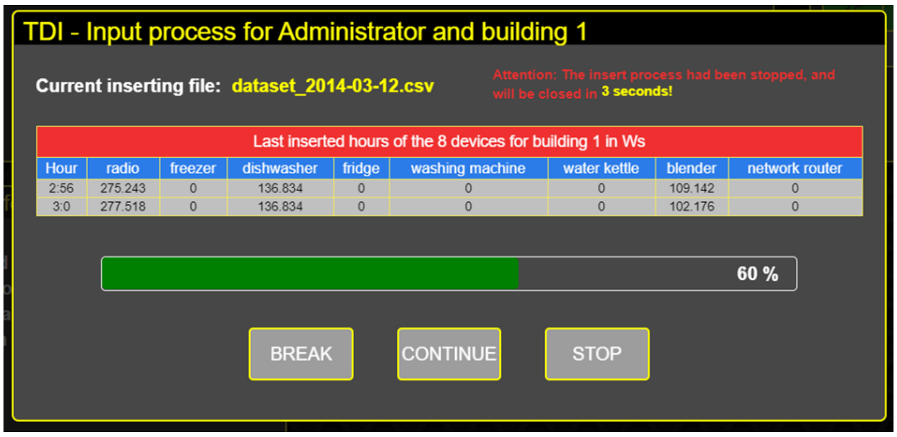
Current inserting file: dataset_2014-02-15.csv

The insert process had been interrupted! This process can be reactivated, if You press "CONTINUE"!

Last inserted hours of the 9 devices for building 2 in Ws

Hour	fridge	dishwasher	microwave	water kettle	washing machine	radio and amplifier	dryer	kitchenware	bedside light
0:12	5137.711	12337.036	8.613	0	0	0	7442.185	15.886	0
0:16	39.427	12381.536	0	0	0	0	1988.319	9.078	0

Figure 13: continue input process



TDI - Input process for Administrator and building 1

Current inserting file: dataset_2014-03-12.csv

Attention: The insert process had been stopped, and will be closed in 3 seconds!

Last inserted hours of the 8 devices for building 1 in Ws

Hour	radio	freezer	dishwasher	fridge	washing machine	water kettle	blender	network router
2:56	275.243	0	136.834	0	0	0	109.142	0
3:0	277.518	0	136.834	0	0	0	102.176	0

Figure 14: stop input process by hard

Widget end

If the insert process had finished normally, the widget will display the message:

"... Would you like to continue?"

The current user has the following options:

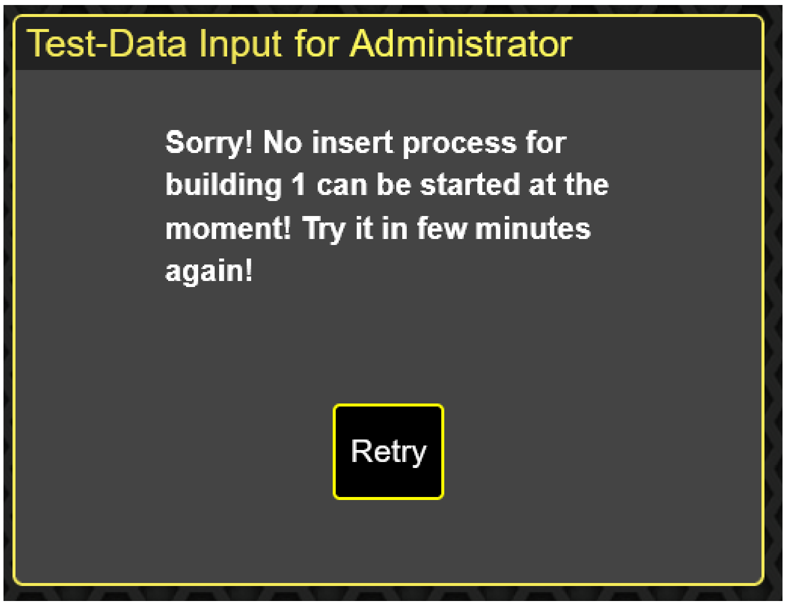
- *YES* - the widget starts at the beginning again
- *NO* - this process has finished completely, and the widget can be closed (q.v. *Figure 8*)

The Trash-Button

If one, ore more test data files, of a particular user, and building have been inserted completely, it is possible to delete all stored information (inserted energy data, devices, ...) for this selected user, and building, by clicking on the *trash-button*, on the *Test File* view. (q.v. *Figure 16*) Every time an *insert process* is executed, it isn't

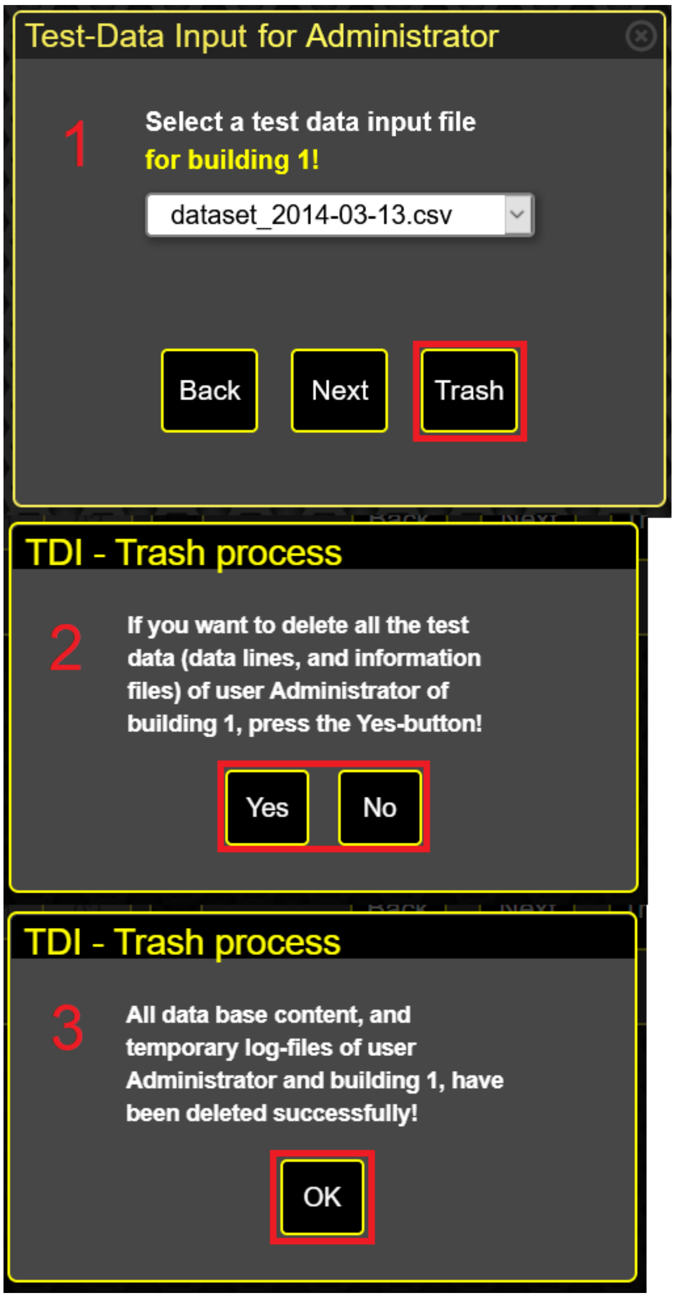
possible for the admin user to start a new one by any other browser, or browser tab, for the previous selected user, and building. This avoids that the needed insert information (e.g. the last inserted energy data line) will be mutilated. (q.v. *Figure 15*)

If there is enough electrical energy data inside the *Mjölnir* data base, it is possible to use most of the other widgets normally, the *The Mjölnir Graphic User Interface (GUI)* describes this.


Test-Data Input for Administrator

Sorry! No insert process for building 1 can be started at the moment! Try it in few minutes again!

Figure 15: no insert process is available



If you want to delete all the test data (data lines, and information files) of user Administrator of building 1, press the Yes-button!

All data base content, and temporary log-files of user Administrator and building 1, have been deleted successfully!

Figure 16: deletion of the inserted data

3.2. Widget Programming

Every time a browser shows a website on the screen, a request is sent to the desired web server, beforehand. The request answer can include a simple HTML-site, or come with more data (e.g. data base content). With this received data, the current used browser is able to change website views, or content. For request and data handling, the server usually uses the programming language PHP, and the client (=browser) Javascript and HTML. In the *Mjölnir*-Application the data base stores important user data, subsection *Database* describes this. Every time, the current user opens a widget, the browser sends a data base request to the web server. The server gets the demanded data, through the data base, and sends this back to the client. If the request is successful, the browser shows all desired data on the screen (e.g. a new challenge message). Every widget contains at least the following three files:

- *ajax_db.php* - the web server uses this script to get the requested data.
- *include.php* - contains *Javascript* code, that the client needs in order to display the received data on screen.
- *config.php* - some application configurations are stored in this file (e.g. widget size).

The next few subsections are describing how server and client are working together through these files, by means of the *Test Data Input* widget.

3.2.1. Test-Data-Input Widget: Javascript Methods

This section gives a program process summary of this widget, through little data flow diagrams, and important information. The widget process itself consists of four important methods, which will be described in the following paragraphs. To get a summary of the context between the *client-,* and the *server-scripts*, this section refers sometimes to the *php-scripts* in section *Test-Data-Input Widget: PHP-Scripts.* One widget procedure consists of the following parts:

- Selection of the needed data for the *input process* itself - user, building, test data file, *process hour* (q.v. *List of Hours*).
- Input process preparation - new buildings, devices, ... will be entered into the data base; this is a description of the next process steps.
- *Input process* - test data inserting.
- *Input process* end.

Before the *Test-Data-Input* widget is ready for a new procedure, it has to be checked, if the logged on user owns admin rights. If this user doesn't own admin rights, following message will be shown on the screen:

"This widget can be only used by the Mjölnir-Administrator!"

If everything is okay, the method *reloadWidget()* will be executed. All server requests in this widget are *ajax-requests.* If the *client* calls up any of these requests, the following info is necessary:

- Type - which request kind is used (e.g. *POST*, or *GET*)
- url - which script the server has to execute
- data - a data list, the server needs for the script execution

The server sends the required data back to the client. If the data return has been finished successfully, the client executes a function with the related data.

Method: reloadWidget()

All available data base users have to be loaded into a *drop-down-list.* Therefore, the client sends an *ajax-request* to the server. Then the *php-script tdi_start_users.php* (q.v. *Start Users Script)* will be executed by the server. All data, the server is sending back to the client will be stored in the variable *data_array.* If this array contains the info *USERS*, all available users can be loaded into the *drop-down-list.* A new *Test-Data-Input* process is ready to start. The variable *current_state* stores the actual program clause of the *client-script.* So the client knows what has to be done next. This variable is related to the methods *btnNext()*, and *btnBack()*, which can be executed, if the buttons *Next*, and *Back* are clicked. According to which program state is stored in *current_state*, one clause of the two previous noted methods will be executed. These program clauses are separated through the *switch-statement.* The buttons *Next*, and *Back* are generated by the *reloadWidget()* execution.

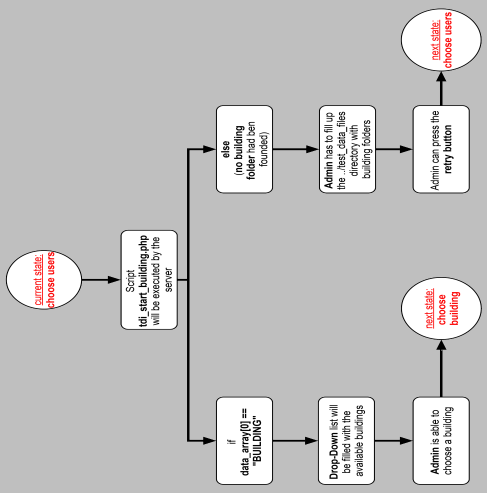
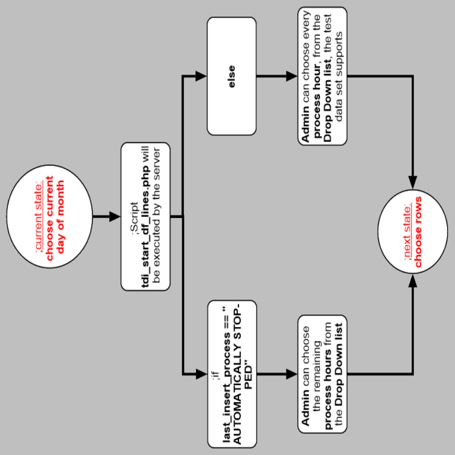
Method: btnNext()

The table below, describes the widget program procedure through little *data-flow-diagrams.* The column *Description* contains the following information:

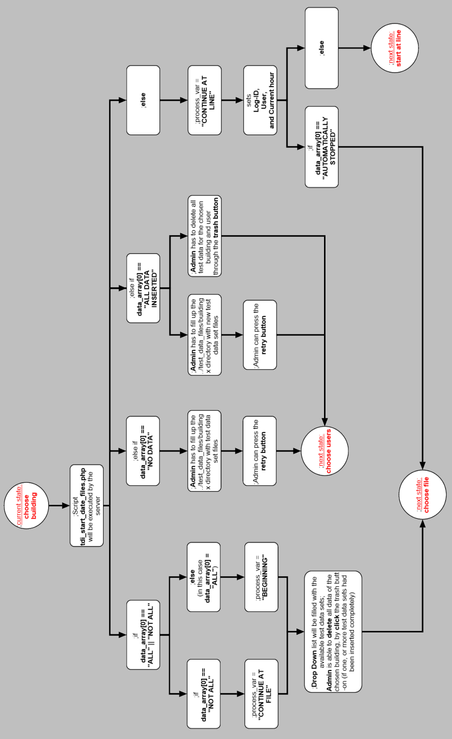
- reference to the next *php-script* description, which will be executed,
- important information about the *data-flow-diagram* in the *Figures* column.

Every time, the button *Next* is clicked, the *current_state* value will be changed. Other important script-variables are *process_var*, and *last_insert_process.* These are responsible, for how the new *input process* starts.

The method *btnBack()* isn't so important to be able to understand the widget procedure, because this method only changes the *current_state* value. The widget process state has to be changed to the previous one. So this method won't be described in the following sections.

Description	Figures
• *Building Script* will be executed. • *BUILDING* means, that building folders are existing on the *web-server* file system. • The sent folder names are filled also in a *drop-down-list.* • The button *Retry* executes the method *dashboard.getData()*, which loads the whole *Mjölnir*-application.	 current state: choose users Script tdi_start_building.php will be executed by the server else (no building folder had ben founded) Admin has to fill up the ../test_data_files directory with building folders Admin can press the retry button next state: choose users if data_array[0] == "BUILDING" Drop-Down list will be filled with the available buildings Admin is able to choose a building next state: choose building
• *Start Data File Lines Script* will be executed. • If the client receives *AUTOMATICALLY STOPPED*, only the remaining *process hours* of the selected test data file will be filled into the *drop-down-list.* • The *process hours* meaning, is described in section List of Hours. • The *process hour* determining is described in section Process Hours Determining.	 ;current state: choose current day of month ;Script tdi_start_df_lines.php will be executed by the server else Admin can choose every process hour, from the Drop Down list, the test data set supports ;if last_insert_process == " AUTOMATICALLY STOPPED" Admin can choose the remaining process hours from the Drop Down list ;next state: choose rows

- *Data Files Script* will be executed.
- Through *ALL*, every available test data file, of the selected building is loaded into a *drop-downl-list.*
- If the client gets the keyword *NOT ALL* back, all remaining test data files will be inserted into the *drop-down-list.*
- *NO DATA* means, that the selected building folder doesn't contain test data files.
- If *ALL DATA INSERTED* is sent back to the client, no test data files remain.
- At *AUTOMATICALLY STOPPED*, or *STOP*, no *drop-down-list* will be shown on screen, only an *unsorted list*, with the values *LOG-ID*, *user*, and *current process hour.*
- If the variable *process_var* contains the value *STOP*, the client changes immediately into the state *start at line.*
- The variable *process_var* will be described in more detail in the section *Method: inputProcessPreparation()*



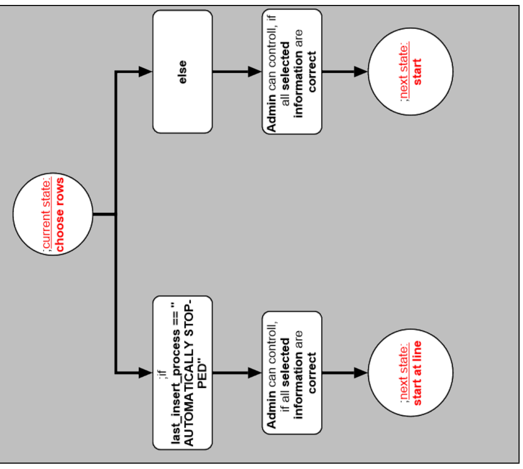
• Before a new *input process* can be started, the admin must check, if all selected information is correct. • If everything is okay, the method *insertProcessPreparation()* will be called up, which will be described in more detail, in the following section.	 ;current state: choose rows else ;if last_insert_process == " AUTOMATICALLY STOPPED" Admin can controll, if all selected information are correct Admin can controll, if all selected information are correct ;next state: start ;next state: start at line

Table 2: Process description of the btnNext() method.

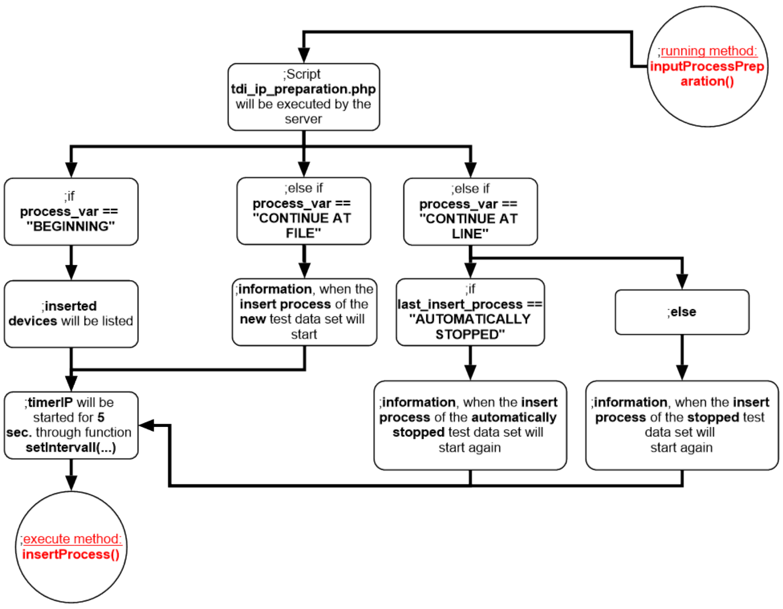
Method: inputProcessPreparation()

The variable *process_var* is very important, if this method is called up, because the variable values are responsible, for how the next *input process* will be started. The following three values can be stored in *process_var*:

- *BEGINNING* - for the selected user, and building, no test data file has been inserted until now.
- *CONTINUE AT FILE* - one, or more test data files, of the selected user, and building, have been already inserted. At the last *insert process*, the last selected test data file has been totally inserted, this means, the new *input process* starts by using a new test data file.
- *CONTINUE AT LINE* - at the last *input process*, the selected energy data file has not been totally inserted into the data base. It has been interrupted at a specific *process hour*. These can be following reasons:
  - *AUTOMATICALLY STOPPED* - the admin selected a specific *process hour*. The *input process* has been interrupted through the system itself, if the process reached the data line for the chosen *process hour*.
  - *STOP* - the last *input process* has been interrupted, by leaving this *web-application*, or clicking on the *STOP*-button. That means, that this process has not reached the chosen *process hour* line.

The *server-script* description of this section, can be read in section *Input Process Preparation Script*.

Another important point is the method *timerIP()* (q.v. *Method: timerIP()*), which is responsible, for the *input process* itself, which will start after *five seconds*. So a *count-down* is produced by the *widget*. Very important *timer* variables are *timer_index*, and *current_timer_id_counter*. *timer_index* is to be initialized every time with the value *5*. *current_timer_id_counter* stores the *timer-id*, which will be returned, after the *Javascript* function *setInterval(...)* has been called up. This function produces a *timer-event*, which calls up the method *timerIP()* (downgrades *timer_index every second*), that has a fixed timestamp of *one second*, until *timer_index* is *zero*. Then the *timer-event* will be deleted through the *Javascript* function *clearInterval(...)*, by the given variable *current_timer_id_counter*. So, *timerIP()* won't be called up after *5 seconds*. The remaining *seconds* will be shown on screen. The method *timerIP()* will also be used, if the *input process* is ending. (q.v. *Figure 17*)



;running method: inputProcessPreparation()
;Script tdi_ip_preparation.php will be executed by the server
;if process_var == "BEGINNING"
;else if process_var == "CONTINUE AT FILE"
;else if process_var == "CONTINUE AT LINE"
;inserted devices will be listed
;information, when the insert process of the new test data set will start
;if last_insert_process == "AUTOMATICALLY STOPPED"
;else
;timerIP will be started for 5 sec. through function setIntervall(...)
;information, when the insert process of the automatically stopped test data set will start again
;information, when the insert process of the stopped test data set will start again
;execute method: insertProcess()

Figure 17: program process: input process preparation

Method: inputProcess()

During an *input process*, the variable value of *process_var* is by default *NEXT LINE*. As long as this is the case, the method *insertProcess()* will be called up, every second. The function *setInterval(...)* is also used for the *insert process*. If the method *inputProcess()* should be executed permanently, multiple factors must be satisfied. The variables, which will be used, to reach this goal, are storing the following values:

- *current_row_expected* - This variable will be upgraded to 240, which correlates to *four minutes*. To allow more energy data to be inserted, *current_row_expected* has to be lower than *current_count_of_rows*.
- *current_count_of_rows* - up to this row, energy data will be collected, and stored. (can be the whole file, or only 5 hours)
- *current_row* - when used, the energy data is then inserted one after another. So, after every successful executed *ajax-request* for the *php-script Input Process Script*, *current_row* is allocated to the *current_row_expected* value. Then four more *minutes* can be inserted, because the following applies:

  if(... && (current_row == current_row_expected)) (q.v. *Method: insertProcess()*)

The *progress-bar* shows the *input process* progress, of the energy data. The *progress-bar* length depends on the variable *current_count_of_rows*, as seen by the below offered formula:

$$l_{pb,new} = \frac{cr_{inserted}}{cor} * 100[\%] \qquad (1)$$

By clicking on the *STOP-button*, the *process_var* value is set on *STOP*. So, *tdi_ip_stop.php* will be executed by the server (q.v. *Input Process Stop Script*), and also the *Javascript-function clearInterval(...)*, which deletes the *timer-event* of the *insert process*. Through *timerIP()* a process ending *count-down* starts.

The *input process* ends automatically, only if *current_row_expected* is greater than *current_count_of_rows*. *process_var* will be set on *FINISHED SUCCESSFUL*, if all energy data of the selected data file has been inserted into the *Mjölnir* data base, in every other case *AUTOMATICALLY STOPPED*. The *client* sends a request to the server, so that *tdi_ip_as_fs.php* is executed. (q.v. *Figure 18*)

3.2.2. Test-Data-Input Widget: PHP-Scripts

As is displayed in *Table 2*, this widget is divided into different process states, which is why every state is related to a server script. The next few paragraphs are describing this script's functionality in short sentences, in relation to the state diagram.

Start Users Script

This script returns a list, of all currently signed in *Mjölnir* users, so that the test data can be inserted for multiple users, the section *Test-Data-Input Widget Process* describes this.

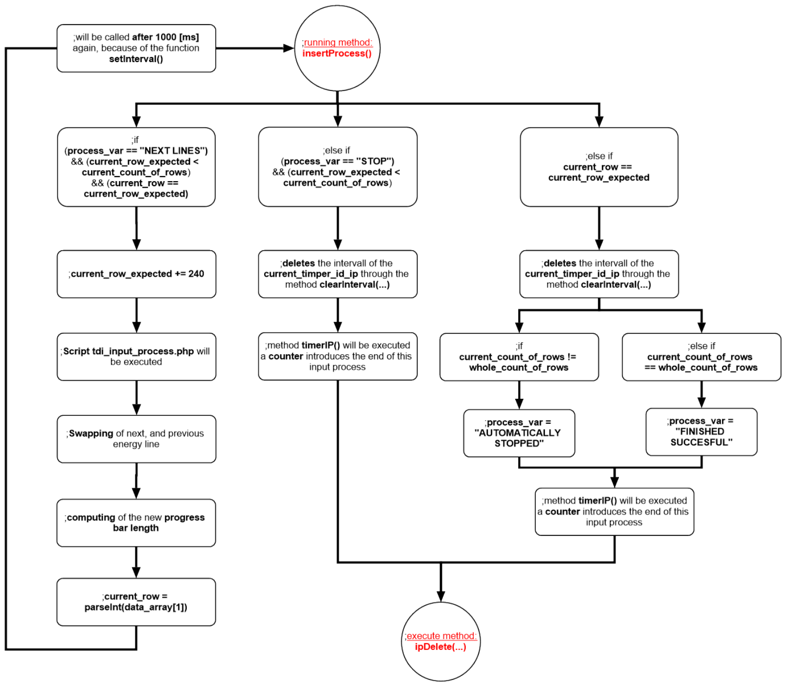

;will be called after 1000 [ms] again, because of the function setInterval()
;running method: insertProcess()
;if (process_var == "NEXT LINES") && (current_row_expected < current_count_of_rows) && (current_row == current_row_expected)
;else if (process_var == "STOP") && (current_row_expected < current_count_of_rows)
;else if current_row == current_row_expected
;current_row_expected += 240
;deletes the intervall of the current_timper_id_ip through the method clearInterval(...)
;deletes the intervall of the current_timper_id_ip through the method clearInterval(...)
;Script tdi_input_process.php will be executed
;method timerIP() will be executed a counter introduces the end of this input process
;if current_count_of_rows != whole_count_of_rows
;else if current_count_of_rows == whole_count_of_rows
;Swapping of next, and previous energy line
;process_var = "AUTOMATICALLY STOPPED"
;process_var = "FINISHED SUCCESFUL"
;computing of the new progress bar length
;method timerIP() will be executed a counter introduces the end of this input process
;current_row = parseInt(data_array[1])
;execute method: ipDelete(...)

Figure 18: program process: input process

Building Script

The test data set, this widget uses, is divided into eight test *buildings*. Every test *building* folder contains multiple *.csv* test data files. If the php-script *tdi_start_building.php* is executed through the server, it will check, if one, or more *test building folders* are existing. After this testing, the following cases can be entered:

- *test building folders* are existing,
- or no *test building folders* are existing.

If they are existing, the data

- current state (in that case *BUILDING*),
- *test building array*,
- and count of *test buildings*,

will be sent back to the client. Otherwise,

- also the current state (in that case *NO BUILDING*),

- and a *no test building folder* message,

will be sent back.
These folders are included by default, but if the system administrator deletes them by an oversight, the current widget user will be informed about this unexpected fault.

Data Files Script

An input process interruption, and a test data input history for the selected user, and building, is reached through the following temporary files:

- ***cif__username__buildingname.php*** - saves the test data files of the selected building, which have already been inserted, and also the current *input process state*. (q.v. *Temporary File: cif_user_building x.php*)
- ***le__username__buildingname.php*** - saves, how many process hours of the test data file should be inserted, and which minute has been inserted last. This info is stored as data lines. (q.v. *Temporary File: le_user_building x.php*)

The script checks at the beginning, if these temporary files are already existing. If it's so, the *input process state* of the last test data file will be checked ***cif__username__buildingname.php***. The following cases can be entered:

- *END* - a list of all not inserted test data files will be returned. If no files are available, the system sends the following message back to the client:

  "All test data files have already been inserted! If you have some more, you have to load them into ../test__data__files/building 0/! Otherwise delete all test data for building 0 through the *trash-button*!"

- *STOP*, or *AUTOMATICALLY STOPPED* (q.v. *Test-Data-Input Widget: Javascript Methods*) - the stopped input process of the selected user, and building, can be continued, at the *process minute*, where it had been interrupted, from the previous time.

If no test data files in the folder of the selected building are existing, *Mjölnir* sends the following message back to the client:

"No test data files for user Administrator and building 0 have been found! You have to load them into ../test__data__files/building 0/!"

This case enters, if the test data files have been deleted on purpose, or accidentally by the system admin.

If no temporary files are existing, all the inserted test data in the *Mjölnir* data base, which are related to the selected user, and building, will be deleted. This is a security measure, because it's also possible, that the system admin deleted these files accidentally.

Start Data File Lines Script

The following information will be sent back to the client through this script:

- *STOP*:
  - the last inserted line of the selected user, and building, which are stored in the *last entry* file (q.v. Temporary File: le_user_building x.php)
  - the total line count of the current chosen file, which is related to the selected building and user
- *AUTOMATICALLY STOPPED*:
  - the same information as before
  - the *next client state* for the *Method: btnNext()*
- *END*:
  - the last inserted line of the selected user, and building, which are stored in the *last entry* file (q.v. Temporary File: le_user_building x.php)
  - the total line count of the current chosen file, which is related to the selected building and user
  - the *next client state* for the *Method: btnNext()*

Count of Lines Script

The script functionality is the count of lines supply, which the current chosen test data file has, through the function *countOfLines(...)* (it has been described in the subsection *Data file script*). Another important piece of information is the *current line*, where the last insert process was stopped. Every test data file line represents one second, where the energy measurement of different devices has been taken. Through this information, the client can determine the count of hours, a user is able to insert.

Delete Data Script

Following data will be deleted from the *Mjölnir* data base, through the function *delSpecificTDIDataContent(...)*:

- the selected building,
- all rooms, devices, and circuits, which are related to the chosen building
- every energy data, which is related to the selected building, and circuit

This function gives the effect, that the *Mjölnir* system is ready for use, because there is no more test data is inside the *Mjölnir* data base anymore.

Furthermore, all temporary files for the selected user and building, will also be deleted through the php-function *unlink(...)*. (q.v. Data Files Script)

If the script has been finished, the following message will be sent back to the client:

"All data base content, and temporary log-files of user Administrator and building 0, have been deleted successfully!"

Input Process Stop Script

Through this script, the *input state* in the temporary file *Temporary File: cif_user_building x.php* of the running input process, will be recorded to *STOP*. Thereby the system knows, how the *insert process* of the selected user, and building has been ended, from the previous time.

Input process automatically stopped/finished successful script

Like in the subsection before, the *input state* of the temporary file *Temporary File: cif_user_building x.php*, will also be recorded to *AUTOMATICALLY STOPPED*, or *END*. During the input process, the *input state* value is *NEXT LINE*. Through the messages below,

- ***"The insert process of user Administrator, building 0 and test data file dataset_2013-12-07 has been stopped automatically!"***, and
- ***"The insert process of user Administrator, building 0 and test data file dataset_2013-12-07 has been finished successfully!"***,

the client receives the info, of how this process has been ended.

Input Process Preparation Script

Before an *input process* is ready for start, some important things have to be done.
If the *input process* of the selected user, and building starts for the first time,

- a test building,
- a test room,
- all electronic devices, which are related to the test building (q.v. Test Devices: building x.php),
- and a circuit, which is also related to the test building,

will be inserted into the *Mjölnir* data base, automatically (the function *insertingDevices* is used). Additionally also the needed temporary files,

- *cif_username_buildingname.php*,
- and *le_username_buildingname.php*,

will be added in the *Mjölnir-web-application* file system.
If a new *input process* starts at a specific line, the *input state* in the temporary file *cif_username_buildingname.php*, will be changed to *NEXT LINES*. In this case, the previous *insert process* was stopped *automatically*, or stopped *by hard* (q.v. The Insert Process). A new *process hour* (q.v. The Insert Process) will be entered at *current line*, in the file *le_username_buildingname.php*, if the last *insert process* was stopped *automatically.*
A new *array-line* will be inserted in *cif_username_buildingname.php*, if the previous test data file, was totally recorded into the *Mjölnir* data base. The new line content consists of:

- the *test data file name*,
- the *selected insert process date* (q.v. ...),
- and the *input state.*

The temporary file *le_username_buildingname.php* will also be adapted for the new test data file.

Input Process Script

Every time this script is executed, the function *writingConsumptionevents(...)* will be called up. The *consumptionevents* table will be filled with the energy data for every test device of the selected test data file. One *consumptionevent* stores the consumed energy value of a specific device, for a fixed timestamp in *kWh* (e.g. four minutes).

This script also calls up the method *addingEvents(...)*, so that it is possible to use the most *Mjölnir* widgets with the inserted test data. The main task of this method is, to fill other data base tables with the the selected test data file content (e.g. *circuit_avg. addingEvents(...)* which can be expanded, every time a new table is added to the data base, which also needs energy data (e.g. Challenge-Goal/-Setting widget).

The following two paragraphs are describing these methods in more detail.

Function: writingConsumptionevents(. . .)

Every test data file is stored as a *.csv* file. It is described in the beginning of the section Developing Methods, every test data file represents the energy consumption of a whole day, for different electronic devices. One row contains the energy consumption of one second, for every used device. Figure 13 shows the input process, and gives a short impression, of how these files can look like.

If the whole test data file should be inserted into the *Mjölnir* data base at the same time, the used *web-server* interrupts the input process after a short time. So, this widget uses a fixed time period (e.g. four minutes). All energy data inside this time period for each electronic device will be added, on extra. Every summed up energy value is stored into the *consumptionevents* table (q.v. SQL-Statement: cons_event_stream). The value *rows_count* of the *sql-statement* is between seven, and nine. It depends on, which devices of the selected building have been used. For the energy consumption of the k-th device, this function uses the following formula:

$$E_k = \frac{1}{3600000} \sum_{i=l}^{n} e_k[i][kWh] \tag{2}$$

Every energy consumption entry for a specific device is specified in *Watt*, but all *consumptionevents* entries must be specified in *[kWh]*. The sum over the fixed time period returns a result in *[Ws]*. The division of *3600000* converts this result into *[kWh]* (q.v. [6, p. 9-13], or Function: writingConsumptionevents(...).

The Input Process Script describes it already, most of the other *widgets* can also be used, if enough test data has been inserted into the table *consumptionevents*. Therefore, the function *writingConsumptionevents(...)* computes the needed energy data for the *circuit_power* table. In contrast to the energy data of the *consumptionevents* table, the *circuit_power* table stores the entries of every second, which are related to the whole *circuit.* That means, that every entry in this table is a power sum of all related devices for a specific second of a day. In this case a *circuit* is a group of devices, which have been added to the selected building. Therefore, the following formula is the result for every *circuit_power* entry:

$$P_i = \sum_{k=0}^{m} p_i[k][W] \tag{3}$$

All entries in this table are recorded in the unit *Watt*, the previous formula shows this. These computed values will be used for the next method *addingEvents(...).*

Method: addingEvents(. . .)

This method adds test energy entries into other needed tables in the background, during the *insert process.* Therefore, most of the *Mjölnir* widgets can be used for test cases, without using real measurement devices. If additional tables must be added into the data base in the future, *addingEvents(...)* can be expanded anytime, because every routine is proceeded separately through a *switch-clause.* One routine will be described in more detail in the following paragraph (for all the others, go to Function: addingEvents(...).

For the recorded *circuits* the table *circuit_avg* exists. This table stores the *min*-, and *max*-values, of a whole day for every *circuit.* That means, for every hour a *min*-, and *max*-column exists. This can also be applied to the test data, with every fixed time period with the computed *circuit_power* data, which will be checked by the following things:

- The hour of the current time period (00:00 to 23:00).
- Is the actual entered *max*-value of the current hour less than one of the *circuit_power*-values of the current time period? If so, then the current data base entry will be replaced by the new *max*-value.
- Is the actual entered *min*-value of the current hour greater than one of the *circuit_power*-values of the current time period? If so, then the current data base entry will be replaced by the new *min*-value.

The actual *circuit_power*-values will be assumed from the return values of the function *writingConsumptionevents(...).*

4. Evaluation

The following paragraphs are giving an evaluation for the *Test-Data-Input* possibilities.

4.1. Test-Data-Input Possibilities

Through this widget, it's possible, to load recorded energy data into the *Mjölnir* data base. This data can be stored in the data base for every signed in user. All the needed *buildings*, *rooms*, *devices*, and *circuits* will also be inserted by this *widget* (q.v. *Figure 19*). The following widgets can be tested, if enough energy data has been inserted:

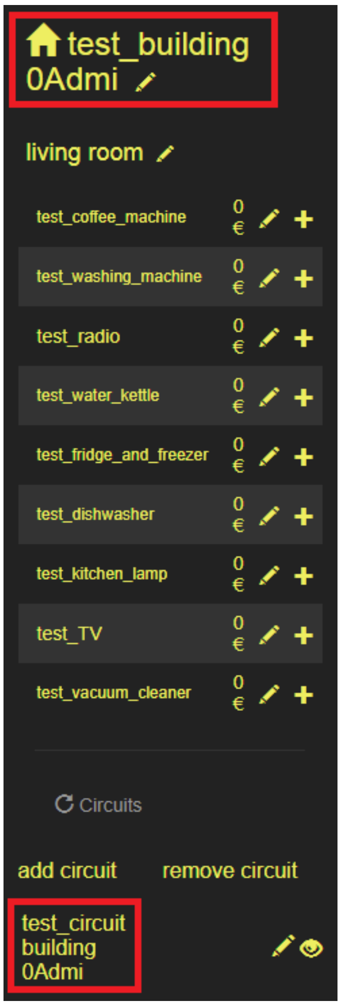
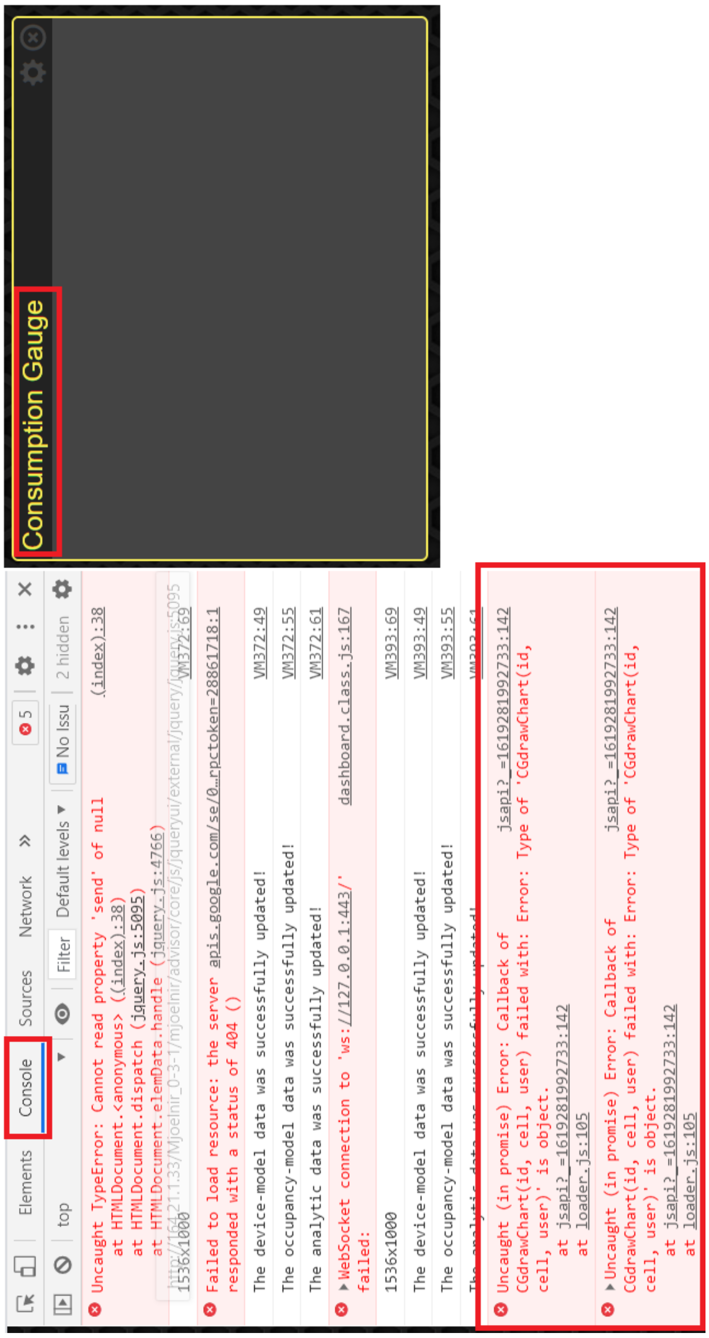


Figure 19: inserted test building, and test circuit

- *Consumption, and Production Calendar* (q.v. *Figure 22*),
- *Consumption Report, Production Report, and Cost Report* (q.v. *Figure 23*), *Energy Advisor* (q.v. *Figure 29*),
- *Estimated Energy* (q.v. *Figure 28*),
- *Timeline* (q.v. *Figure 27*),
- *Device Usage Model* (q.v. *Figure 27*),
- *Occupancy* - no test result for this *widget* is available,
- *Consumption per Room* (q.v. *Figure 25*),
- *Tarif Switch* (q.v. *Figure 24*),
- and *Timeseries* (q.v. *Figure 26*).

The figures below are displaying the test results. Beforehand, six *test data files* of *building 0* (this *building* is related to the *greend-sourceforge.net* data set, which is described in the following paragraphs) for the user *Administrator* have been inserted into the *Mjölnir* data base. Every inserted *test data file* contains a particular day in April 2021. *documentation_0-3-1.pdf* describes the previous listed widgets in detail on the pages *16 to 21.*

The widgets *Consumption Gauge*, *Power Gauge*, and *Device Itemization*, should also be usable, in accordance with *documentation_0-3-1.pdf*. Every time these widgets are called up, failures are displayed in the *web-browser console* (q.v. *Figure 20*). It is more than likely a problem, if the present-day *web-browsers* call up the functions *CGdrawChart(...)*, *PGdrawChart(...)*, and *drawCake(...).*



```
Uncaught TypeError: Cannot read property 'send' of null          (index):38
    at HTMLDocument.<anonymous> ((index):38)
    at HTMLDocument.dispatch (jquery.js:5095)
    at HTMLDocument.elemData.handle (jquery.js:4766)
http://164.21.1.33/Mjoelnir_0-3-1/mjoelnir/advisor/core/js/jqueryui/external/jquery/jquery.js:5095
1536x1000                                                         VM372:69
Failed to load resource: the server   apis.google.com/se/0…rpctoken=28861718:1
responded with a status of 404 ()
The device-model data was successfully updated!                   VM372:49
The occupancy-model data was successfully updated!                VM372:55
The analytic data was successfully updated!                       VM372:61
▸ WebSocket connection to 'ws://127.0.0.1:443/'        dashboard.class.js:167
failed:
1536x1000                                                         VM393:69
The device-model data was successfully updated!                   VM393:49
The occupancy-model data was successfully updated!                VM393:55
The analytic data was successfully updated!                       VM393:61
Uncaught (in promise) Error: Callback of        jsapi?_=1619281992733:142
CGdrawChart(id, cell, user) failed with: Error: Type of 'CGdrawChart(id,
cell, user)' is object.
    at jsapi?_=1619281992733:142
    at loader.js:105
▸ Uncaught (in promise) Error: Callback of      jsapi?_=1619281992733:142
CGdrawChart(id, cell, user) failed with: Error: Type of 'CGdrawChart(id,
cell, user)' is object.
    at jsapi?_=1619281992733:142
    at loader.js:105
```

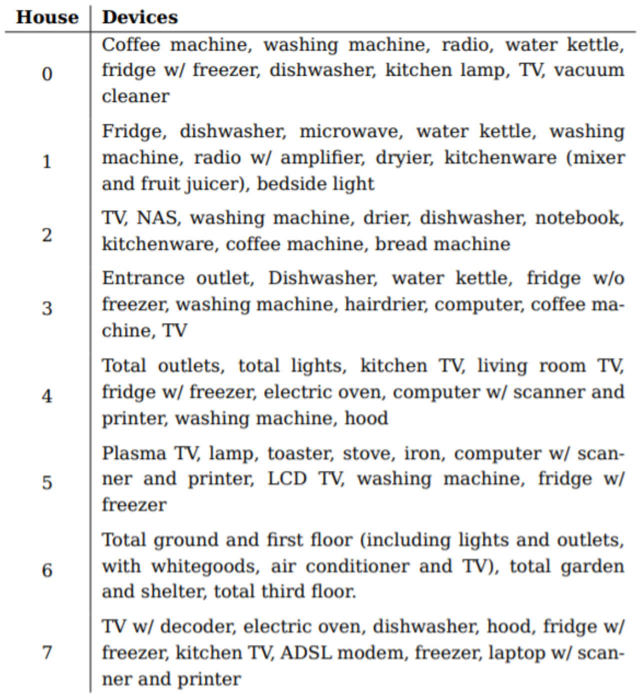
Figure 20: Consumption Gauge widget failure

The test data, that this widget uses, is stored in *.csv-files*, which have been recorded in different households, of Italy and Austria, between 2013 and 2014. All data has been anonymized. The recorded period is between nine and ten months. For every household, energy data, of 7 to 9 household appliance exists (q.v. *Figure 21*). A single *.csv-file* contains the energy data, of a whole day. Every row in one of these files, represents one recorded second. The used *physical unit* is *Watt*, corresponding to the average consumed power in the measured duration. The whole test data set, can be downloaded by *greend-sourceforge.net* [8].

The *Test-Data-Input widget* can be used by the following screen resolutions in *Google-Chrome*:

- ***Full-HD*** - 1920 x 1080,
- ***max-width: 1536px***,
- ***max-width: 1366px***,
- ***max-width: 800px*** - standard tablet resolution in panel format (this widget can be used on a tablet, if the panel format is used),

but this widget is most suitable, by using the *Full-HD* resolution, or the *window width* of *1536px*.

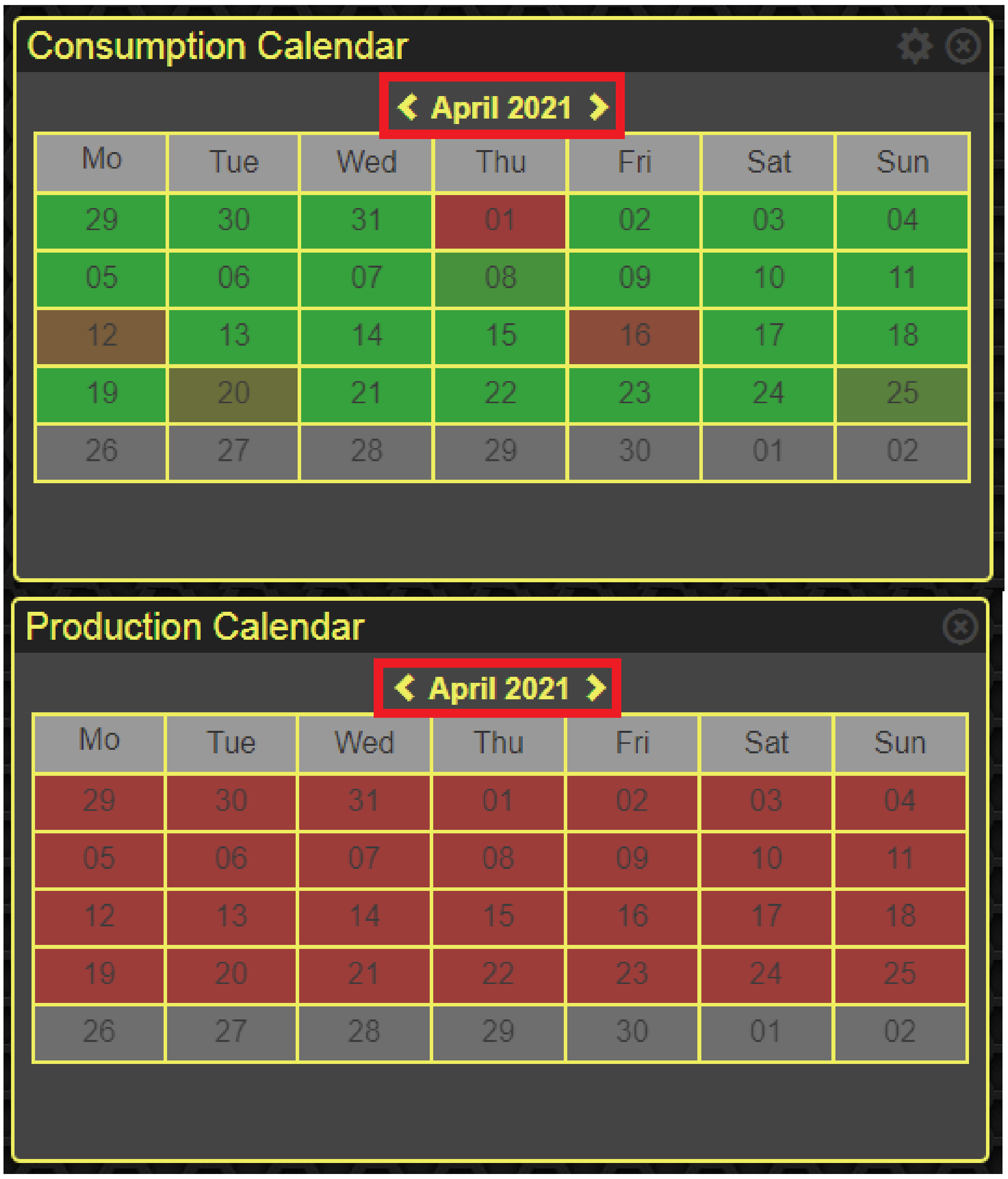


House	Devices
0	Coffee machine, washing machine, radio, water kettle, fridge w/ freezer, dishwasher, kitchen lamp, TV, vacuum cleaner
1	Fridge, dishwasher, microwave, water kettle, washing machine, radio w/ amplifier, dryier, kitchenware (mixer and fruit juicer), bedside light
2	TV, NAS, washing machine, drier, dishwasher, notebook, kitchenware, coffee machine, bread machine
3	Entrance outlet, Dishwasher, water kettle, fridge w/o freezer, washing machine, hairdrier, computer, coffee machine, TV
4	Total outlets, total lights, kitchen TV, living room TV, fridge w/ freezer, electric oven, computer w/ scanner and printer, washing machine, hood
5	Plasma TV, lamp, toaster, stove, iron, computer w/ scanner and printer, LCD TV, washing machine, fridge w/ freezer
6	Total ground and first floor (including lights and outlets, with whitegoods, air conditioner and TV), total garden and shelter, total third floor.
7	TV w/ decoder, electric oven, dishwasher, hood, fridge w/ freezer, kitchen TV, ADSL modem, freezer, laptop w/ scanner and printer

Figure 21: Table of the recorded devices [4, p. 6].

4.2. Test-Data-Input Widget Testing Results

The following figures are displaying all the results of the *Test-Data-Input* widget, which are described in the previous paragraph.



Consumption Calendar

April 2021

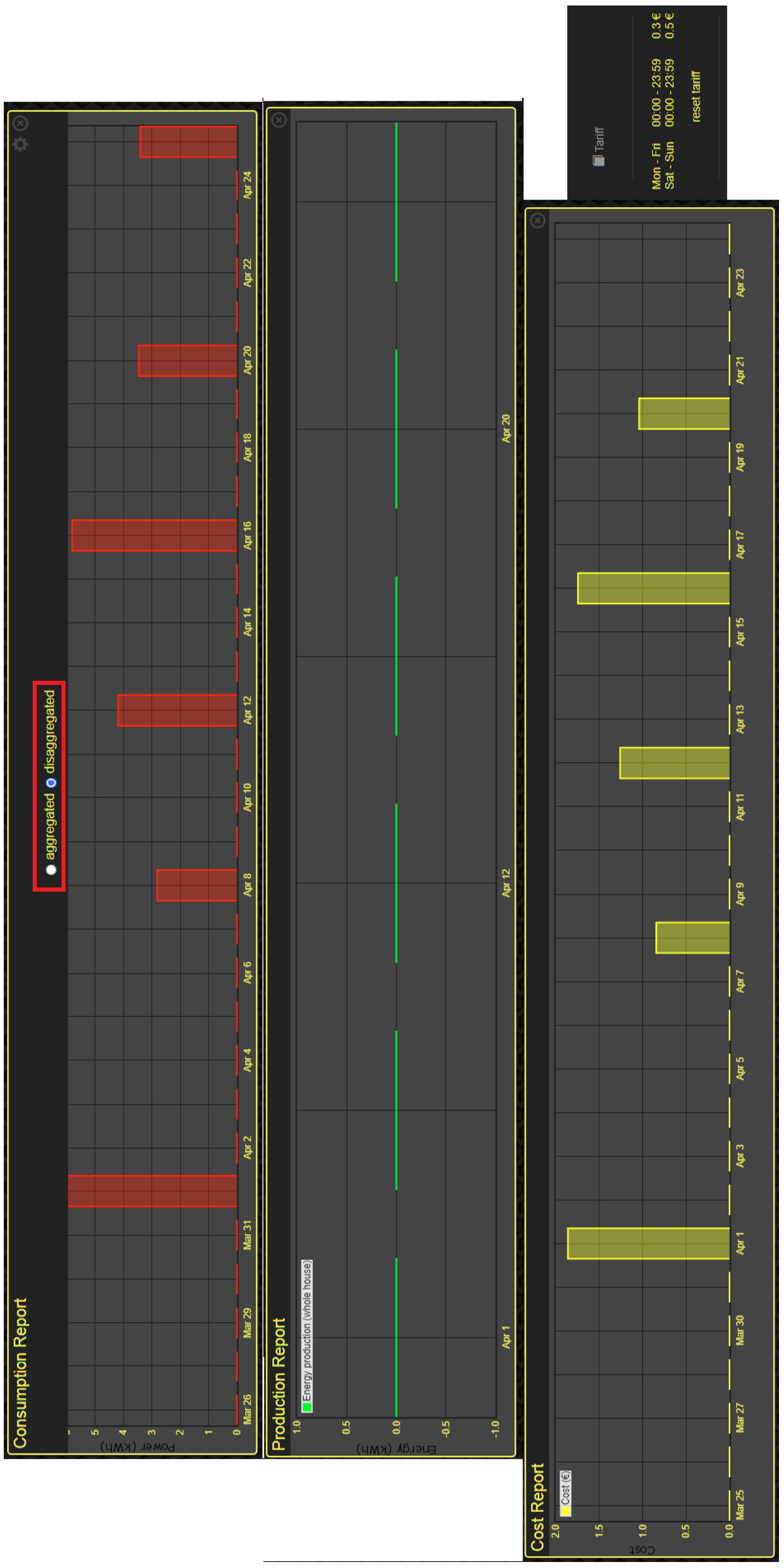
Mo	Tue	Wed	Thu	Fri	Sat	Sun
29	30	31	01	02	03	04
05	06	07	08	09	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	01	02

Production Calendar

April 2021

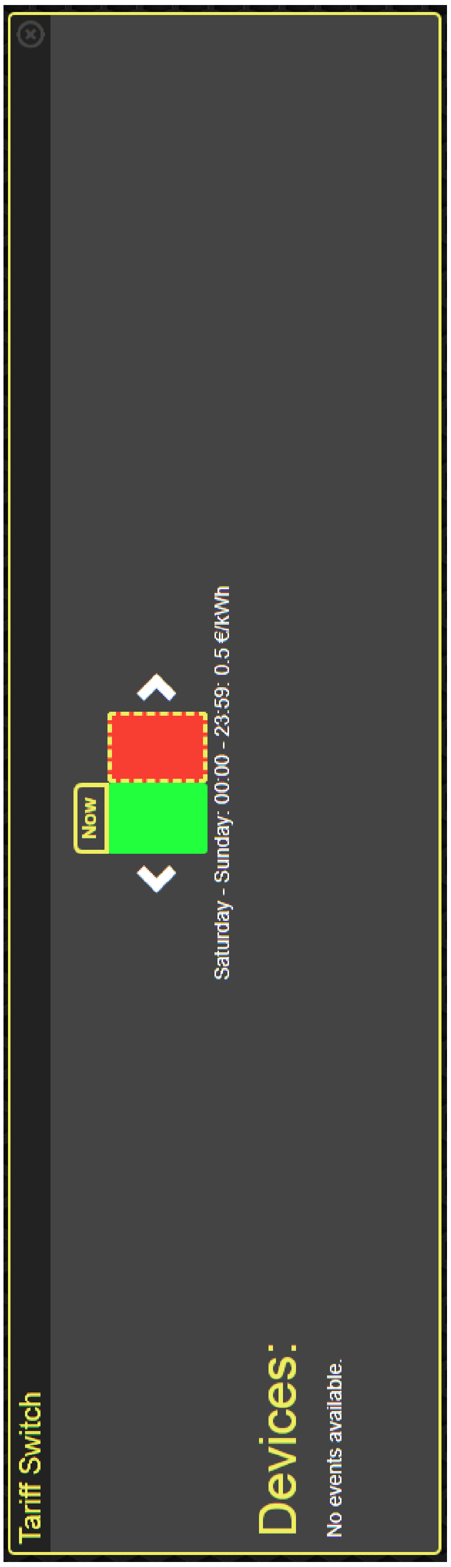
Mo	Tue	Wed	Thu	Fri	Sat	Sun
29	30	31	01	02	03	04
05	06	07	08	09	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	01	02

Figure 22: Consumption, and Procuction Calendar



Mon - Fri 00:00 - 23:59 0.3 €
Sat - Sun 00:00 - 23:59 0.5 €

Figure 23: Consumption, Procuction, and Cost Report



Saturday - Sunday: 00:00 - 23:59: 0.5 €/kWh

Devices:

No events available.

Figure 24: Tariff Switch

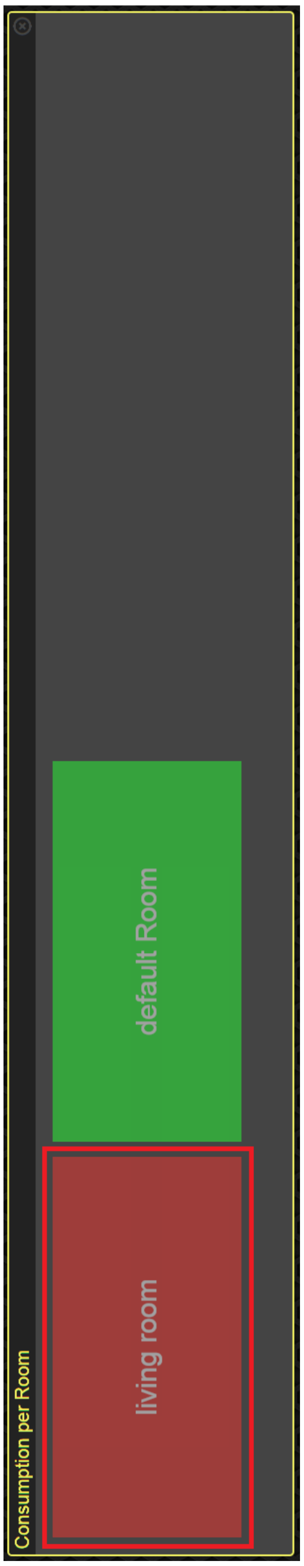


Figure 25: Consumption per Room

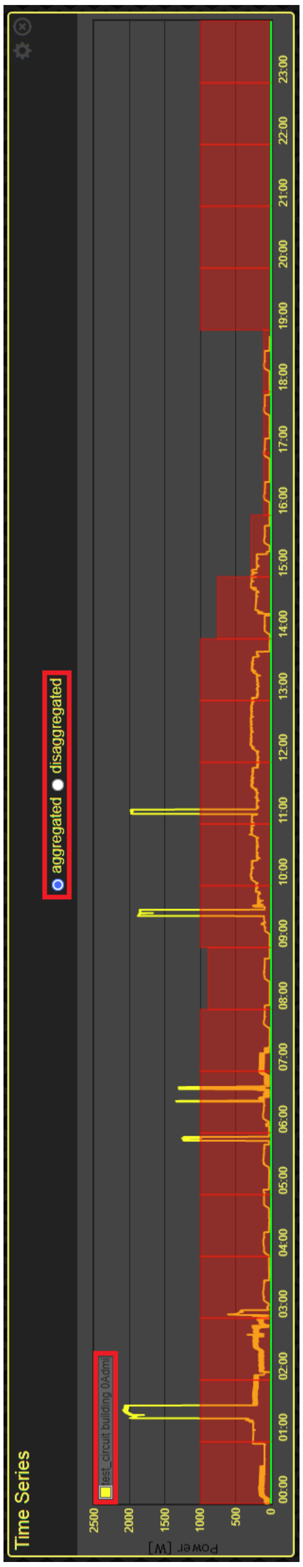


Figure 26: Timeseries

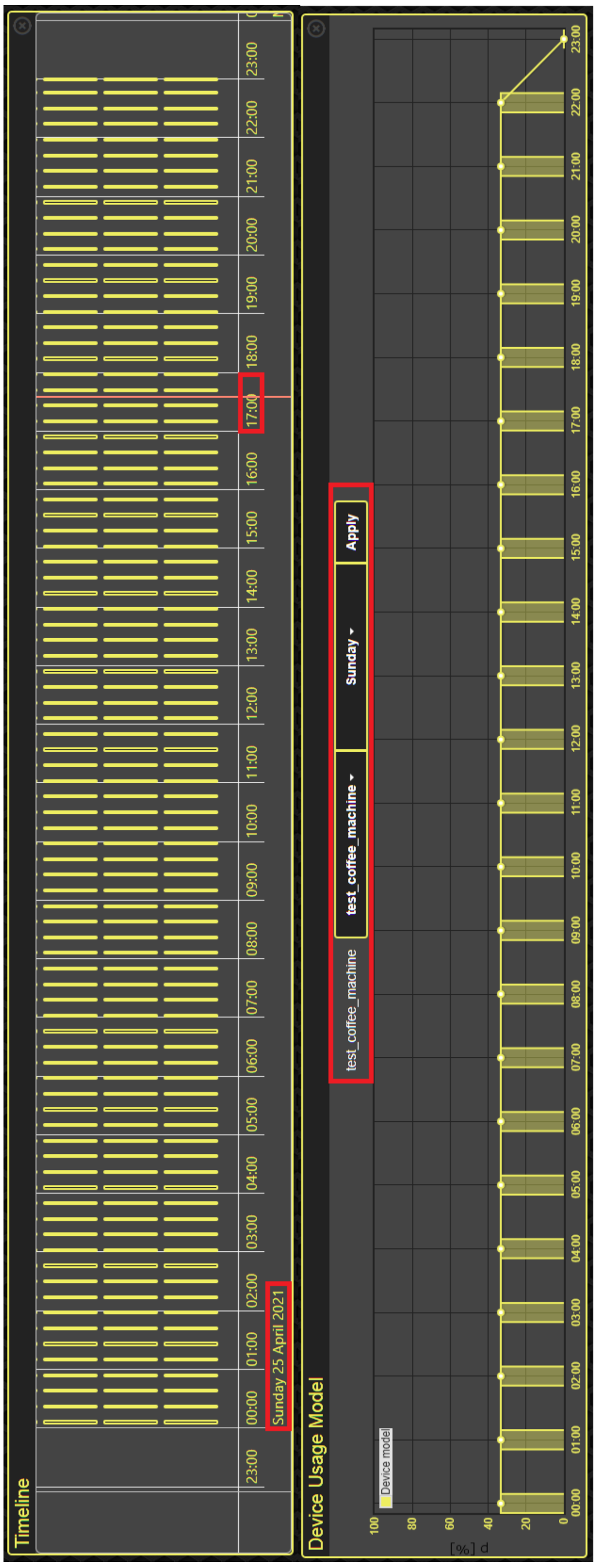


Figure 27: Timeline, and Device Usage Model

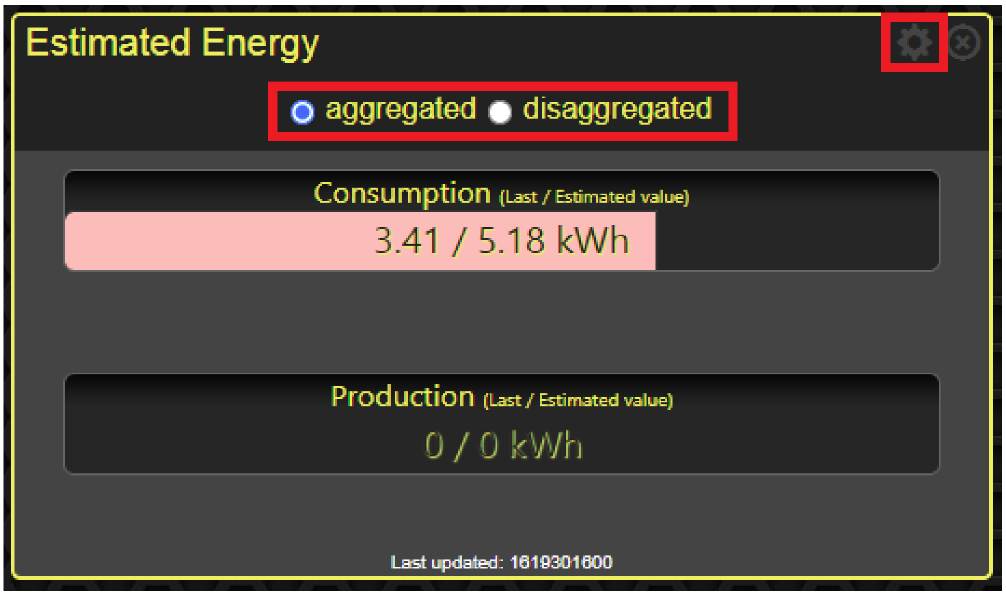
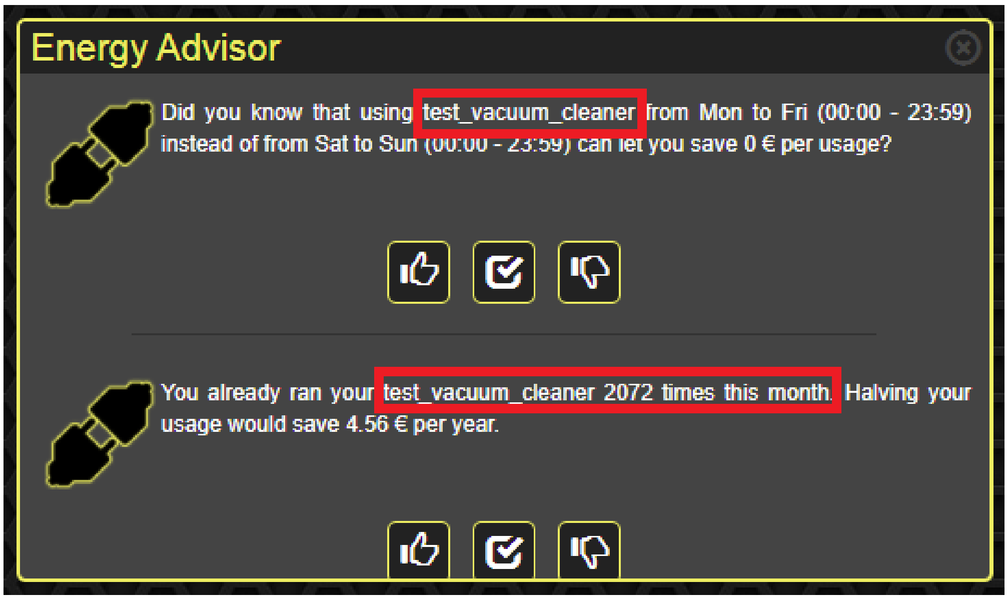


Figure 28: Estimated Energy


Energy Advisor

Did you know that using test_vacuum_cleaner from Mon to Fri (00:00 - 23:59) instead of from Sat to Sun (00:00 - 23:59) can let you save 0 € per usage?

You already ran your test_vacuum_cleaner 2072 times this month. Halving your usage would save 4.56 € per year.

Figure 29: Energy Advisor

5. Possible Future Widget: Challenge-/Goal-setting Widget

The goal of this widget is, that system users are challenged, so that they can save energy in their environment (e.g. flat, office, ...). This will be reached through challenges, which are divided in different levels (*level 1 to level n*). If a user wins a challenge, they will be upgraded a level. If the user loses, they will be downgraded a level. Such challenges will be executed once a month (in Figure 30 it's the last day of a month). Every time this widget is executed, the current user gets messages, which are dependent on the the. These are stored in the *Mjölnir* data base table *messages* (q.v. Figure 1). The *Challeng-/Goal-setting* widget is supervised by an admin, and so this widget process is divided into an user, and admin part. These will be described in more detail, in the next two subsections.

5.1. Widget Process: Admin

Before this widget can be executed, the admin has to insert the following things into the data base:

- minimum 3 challenge levels
- 1 challenge for level 1, and minimum 2 for the other levels
- min. 3 incentives per level
- min. 1 general message, which contains a reason

If this data has been inserted, the challenge relation for the users can be started. Every user should be informed, the day before, that a challenge will be executed (can be done by e-mail, or directly through the widget itself). The admin must justify, why this challenge is executed. The challenges are selected randomly, but in relation to the current user level. This procedure is executed for every user. On the admin screen, a table for every user challenge is shown, with the following content:

- username
- title
- common-/special-description
- current level

- start
- end
- decision
- current degree
- current energy consumption
- incentive

Because of *current energy consumption*, every challenge row is related to a colour. The admin can use these colours as control, during the challenge execution. Through the *refresh-button*, the admin is able to update the current *challenge-table*. The admin is also responsible, for every *challenge-winner* getting their win incentive.

5.2. Widget Process: User

Every *challenge-user* will be informed about his own challenge. They are able to *accept*, or *decline* this challenge. If the user declines this challenge, the user will be downgraded a level, automatically. Every graphic user interface consists of the following tab:

- *New* - gives an overview about the upcoming challenge. [7, p. 6]
- *Pending* - if the actual challenge has been accepted, the following data will be shown on this tab [7, p. 6]:
  - title (only *detail-view*)
  - common-/special-description
  - current level (only *detail-view*)
  - start
  - end
  - decision (only *detail-view*)
  - current degree (only *detail-view*)
  - current energy consumption (on *detail-view* with *progress-bar*; detailed consumption disaggregation)
  - incentive (only *detail-view*)

The *detail-view* gives additional, and important information on the pending challenge. It can be activated, by clicking on the *detail-button*.

The *normal-view* determines the *current degree* by fixed colours. *Current degree* determines, if the current user is able to win, or lose the pending challenge

- *Won* - a list of all won challenges is shown. [7, p. 6]
- *Declined* - a list of all declined challenges is shown. [7, p. 6]
- *Missed* - a list of all lost challenges is shown. [7, p. 6]

5.3. Description of Good Challenges

These paragraphs are important for the *Challenge-/Goal-setting* widget, because the provided challenges have to be good. The following list gives information, about which content a challenge should have:

- A challenge should take place on a fixed date, for every member (e.g. last day of a month) -> a friendly stress occurs. [10]
- Every challenge must be expressed clearly, and in short. [10]
- Time factor -> shouldn't take too long time (a big expenditure of time occurs). [10]
- The members have to know, why this challenge is executed. [5]
- Which goal shall be reached. [5]
- How can this goal be reached. [5]

5.3.1. Challenge Example for *Mjölnir*

- Every *challenge-user* will be informed through a message, the day before. This message contains a short reason about the challenge execution.
- The three devices, which have consumed the most energy in the last month, will be selected.
- It is not allowed, that the whole energy consumption of these devices, pass a given energy level.

It's important, that a lot of challenges exist, because otherwise it will become boring for the *challenge-users* over a short period of time.

A summary of, how the *Challenge-/Goal-setting* program procedure can look like, is given in *Figure 30*, and *Figure 31*. The *uml-diagram* in *Figure 1* gives one more summary of, how the *Mjölnir* data base should be expanded, if this widget is implemented. To determine the maximum energy consumption of particular challenges, the table *challenge_equal_devices* can be used. This table stores the energy consumption, a used household device, should normally consume for one day. The *challenge_equal_devices* table must be filled, by the system admin.

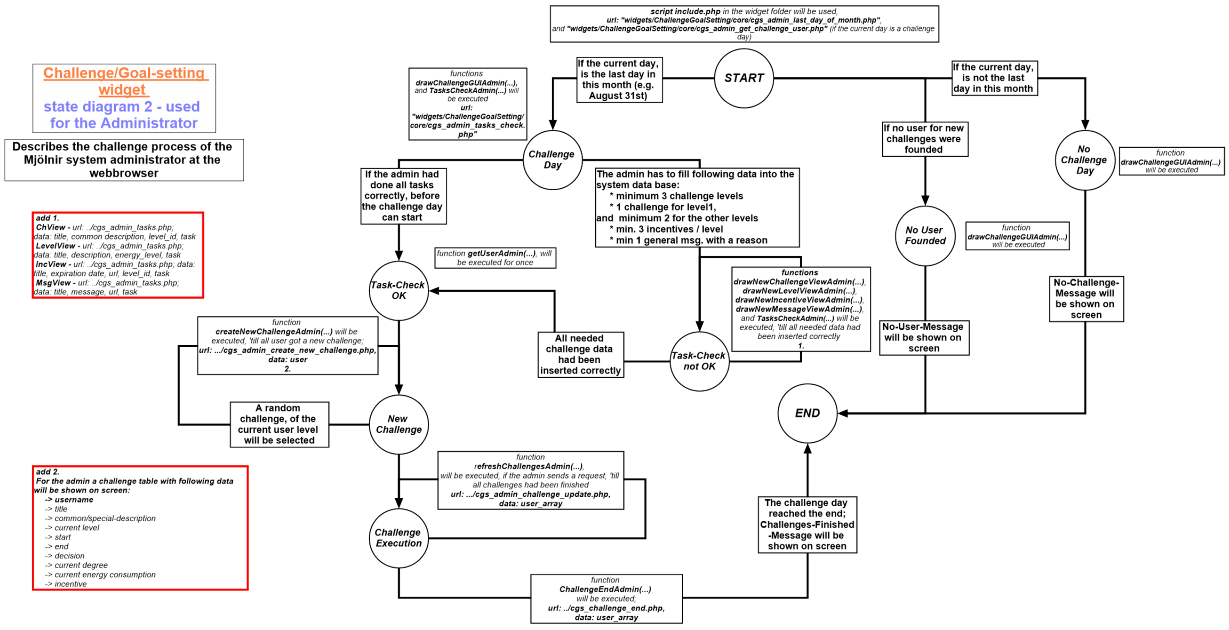

Challenge/Goal-setting widget
state diagram 2 - used for the Administrator
Describes the challenge process of the Mjölnir system administrator at the webbrowser
add 1.
ChView - url: ../cgs_admin_tasks.php; data: title, common description, level_id, task
LevelView - url: ../cgs_admin_tasks.php; data: title, description, energy_level, task
IncView - url: ../cgs_admin_tasks.php; data: title, expiration date, url, level_id, task
MsgView - url: ../cgs_admin_tasks.php; data: title, message, url, task
add 2.
For the admin a challenge table with following data will be shown on screen:
-> username
-> title
-> common/special-description
-> current level
-> start
-> end
-> decision
-> current degree
-> current energy consumption
-> incentive
script include.php in the widget folder will be used, url: "widgets/ChallengeGoalSetting/core/cgs_admin_last_day_of_month.php", and "widgets/ChallengeGoalSetting/core/cgs_admin_get_challenge_user.php" (if the current day is a challenge day)
START
If the current day, is the last day in this month (e.g. August 31st)
If the current day, is not the last day in this month
functions drawChallengeGUIAdmin(...), and TasksCheckAdmin(...) will be executed url: "widgets/ChallengeGoalSetting/core/cgs_admin_tasks_check.php"
Challenge Day
If the admin had done all tasks correctly, before the challenge day can start
The admin has to fill following data into the system data base:
* minimum 3 challenge levels
* 1 challenge for level1,
and minimum 2 for the other levels
* min. 3 incentives / level
* min 1 general msg. with a reason
function getUserAdmin(...), will be executed for once
Task-Check OK
functions drawNewChallengeViewAdmin(...), drawNewLevelViewAdmin(...), drawNewIncentiveViewAdmin(...), drawNewMessageViewAdmin(...), and TasksCheckAdmin(...) will be executed, 'till all needed data had been inserted correctly 1.
All needed challenge data had been inserted correctly
Task-Check not OK
function createNewChallengeAdmin(...) will be executed, 'till all user got a new challenge; url: .../cgs_admin_create_new_challenge.php, data: user 2.
A random challenge, of the current user level will be selected
New Challenge
function refreshChallengesAdmin(...), will be executed, if the admin sends a request, 'till all challenges had been finished url: .../cgs_admin_challenge_update.php, data: user_array
Challenge Execution
function ChallengeEndAdmin(...) will be executed; url: ../cgs_challenge_end.php, data: user_array
The challenge day reached the end; Challenges-Finished-Message will be shown on screen
END
If no user for new challenges were founded
No User Founded
function drawChallengeGUIAdmin(...) will be executed
No-User-Message will be shown on screen
No Challenge Day
function drawChallengeGUIAdmin(...) will be executed
No-Challenge-Message will be shown on screen

Figure 30: program process challenge-/goal-setting widget: javascript overview for the admin

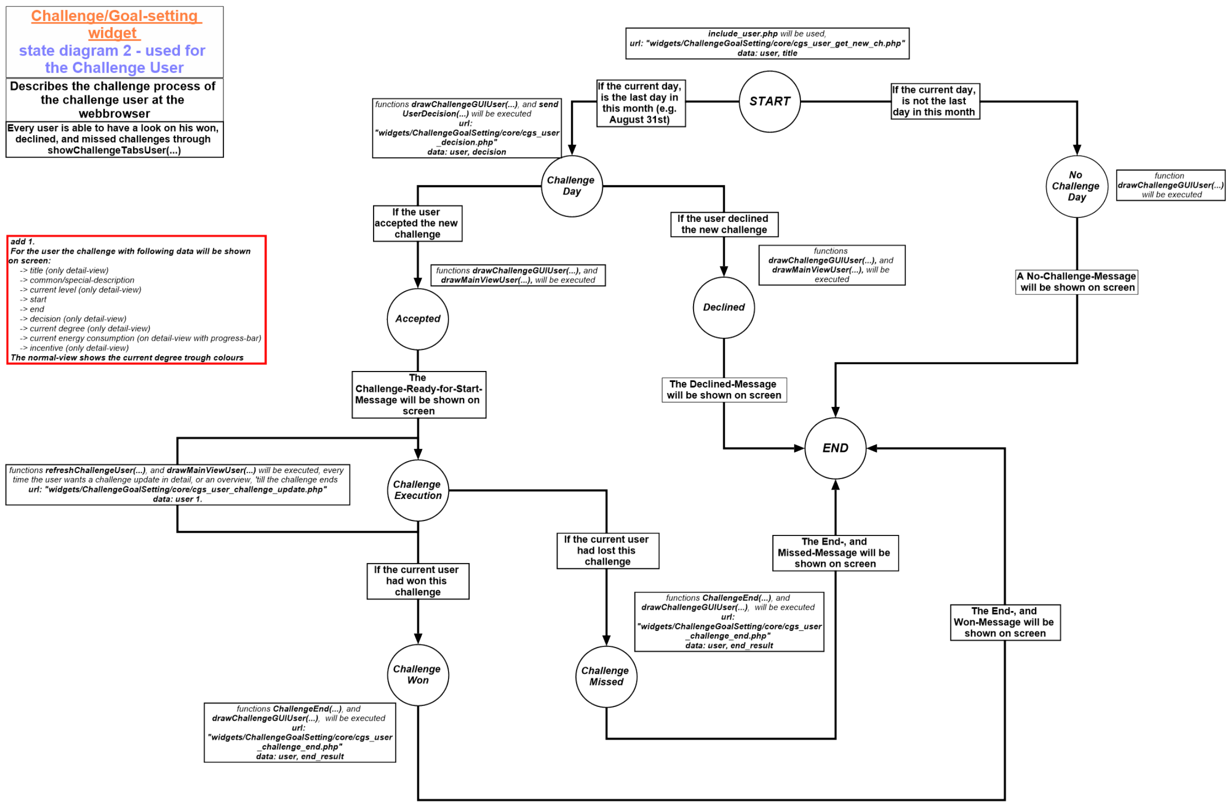

Challenge/Goal-setting widget
state diagram 2 - used for the Challenge User
Describes the challenge process of the challenge user at the webbrowser
Every user is able to have a look on his won, declined, and missed challenges through showChallengeTabsUser(...)
add 1.
For the user the challenge with following data will be shown on screen:
-> title (only detail-view)
-> common/special-description
-> current level (only detail-view)
-> start
-> end
-> decision (only detail-view)
-> current degree (only detail-view)
-> current energy consumption (on detail-view with progress-bar)
-> incentive (only detail-view)
The normal-view shows the current degree trough colours
include_user.php will be used,
url: "widgets/ChallengeGoalSetting/core/cgs_user_get_new_ch.php"
data: user, title
START
If the current day, is the last day in this month (e.g. August 31st)
If the current day, is not the last day in this month
functions drawChallengeGUIUser(...), and send UserDecision(...) will be executed
url: "widgets/ChallengeGoalSetting/core/cgs_user_decision.php"
data: user, decision
Challenge Day
No Challenge Day
function drawChallengeGUIUser(...) will be executed
A No-Challenge-Message will be shown on screen
If the user accepted the new challenge
functions drawChallengeGUIUser(...), and drawMainViewUser(...), will be executed
Accepted
The Challenge-Ready-for-Start-Message will be shown on screen
If the user declined the new challenge
functions drawChallengeGUIUser(...), and drawMainViewUser(...), will be executed
Declined
The Declined-Message will be shown on screen
END
functions refreshChallengeUser(...), and drawMainViewUser(...) will be executed, every time the user wants a challenge update in detail, or an overview, 'till the challenge ends
url: "widgets/ChallengeGoalSetting/core/cgs_user_challenge_update.php"
data: user 1.
Challenge Execution
If the current user had won this challenge
If the current user had lost this challenge
Challenge Won
Challenge Missed
functions ChallengeEnd(...), and drawChallengeGUIUser(...), will be executed
url: "widgets/ChallengeGoalSetting/core/cgs_user_challenge_end.php"
data: user, end_result
functions ChallengeEnd(...), and drawChallengeGUIUser(...), will be executed
url: "widgets/ChallengeGoalSetting/core/cgs_user_challenge_end.php"
data: user, end_result
The End-, and Missed-Message will be shown on screen
The End-, and Won-Message will be shown on screen

Figure 31: program process challenge-/goal-setting widget: javascript overview for the user

6. Conclusion

The main task of the *Test-Data-Input* widget is that recorded test data will be inserted into the *Mjölnir* data base, the most of the other paper chapters are describing this. Through this *widget*, the whole system functionality can be tested. The chapter *Evaluation*, describes the *widget* result in more detail.

The main goal of the *Challenge-/Goal-setting* widget is to motivate modern society humans, to save energy constructively. The whole widget implementation was not possible, because of the time required. So, the section *Possible Future Widget: Challenge-/Goal-setting Widget* gives a short summary, how this *widget* could be implemented.

By testing the *Test-Data-Input* widget, the following points have been noticed, which should be supported by *Mjölnir* in the future:

- If a new widget is developed, it should be possible for it to be installed by an admin user. At the moment, a widget can only be uploaded via the *application file system.*
- A lot of time will be saved, if a *bootstrap* documentation exists for the *widget-design.*
- It's also important to check the used *Javascript-library*, if it is up to date (e.g. *Consumption Gauge* widget)
- If particular *widgets* are opened, it can be, that other *widgets* won't be shown correctly on the screen. That depends probably on the *cell-IDs* (every *cell-ID* is related to a currently opened *widget*), which have been allocated to the *html-content*, when the *widget* is opend every time.

A. Source Code

A.1. Function: writingConsumptionevents(...)

```
// Function for writing the .csv data files into the table
   consumptionevents of the data base
if (!function_exists("writingConsumptionevents")) {
    function writingConsumptionevents($connection, $user,
        $building, $data_directory, $data_filename,
        $current_line, $current_line_expected)
    {
        // the system inserts the energy sum of 240 data
           file lines, for all current needed devices,
           every time, this function is called
        $needed_data_lines = get_DataLines($current_line,
           $current_line_expected, $data_directory,
           $data_filename); // all current needed data
           lines are stored into an array
        $i = 0; // array index of needed_data_lines
        $current_devices = getNeededDevices($building,
           $data_filename, "ID");
        $cons_event_stream = "INSERT INTO consumptionevents
            (owner, device, start, duration, consumption)
           VALUES ";
        $current_con_event_array = get_ZeroLine(count(
           $current_devices));
        $current_cir_event_array = array();

        // get beginning timestamp of consumptionevents
        require("../tmp_log_files/cif_".$user."_".$building
           .".php");
        $current_date = $current_inputted_files["file ".
           count($current_inputted_files)]["date"]; //
           current date, of the current inserting file
        $date_format = DateTime::createFromFormat('d.m.y H:
           i:s', $current_date.' 00:00:00'); // creates an
           instance of the DateTime-object, with a special
           format
        $current_date_timestamp = $date_format->
           getTimestamp() + $current_line;
```

```
while ($i < count($needed_data_lines)) // computes
   also the circuit power of the last four minutes
{
    $line_array = explode(",", $needed_data_lines[
       $i]); // splits the current energy data line
        into an array, at every comma
    $circuit_energy_help = 0; // circuit energy for
        every power event (means every second)

    // controls, if it's a data line, or only text
    if ($line_array[0] != "timestamp")
    {
        for ($k = 0; $k < count($current_devices);
           $k++)
        {
            if ($line_array[$k + 1] != "NULL")
            {
                $conv_current_dev_energy = (float)
                   $line_array[$k + 1]; // device
                   energy in Watt
                $current_con_event_array[$k] +=
                   $conv_current_dev_energy; //
                   sums the consumptionevent energy
                    for every device
                $circuit_energy_help +=
                   $conv_current_dev_energy; //
                   sums the circuit power for every
                    second (= sum over all devices)
            }
        }

        $circuit_energy_help = round(
           $circuit_energy_help, 12); // current
           circuit power in W (rounded up to twelve
            decimal place)
    }

    $current_cir_event_array[$i][0] =
       $current_date_timestamp;
    $current_cir_event_array[$i][1] =
        $circuit_energy_help;

    $current_date_timestamp += 1;
    $i++;
}

// creates the consumption event stream
for ($k = 0; $k < count($current_devices); $k++)
{
    $current_dev_energy = round(((
       $current_con_event_array[$k]) / 3600000),
       14); // energy in kWh (1/3600 ... hour
       coefficient)
    $cons_event_stream .= "('$user', '".
       $current_devices[$k].substr($user, 0, 4)."',
        '$current_date_timestamp', '$i', '
       $current_dev_energy')";

    if($k == (count($current_devices) - 1))
        $cons_event_stream .= ";";
    else
        $cons_event_stream .= ", ";
}

$connection->addEventTDI($cons_event_stream); //
   inserts the current expected data lines into the
    data base (only one insert command)

// refreshes the last entry file, of the current
   building, through the last inserted data line
require("../tmp_log_files/le_".$user."_".$building
   .".php");
$current_insert_key = $last_entry["insert key"];
$last_insert_time = $last_entry["last insert time
   "];
$input_stream_last_entry = "<?php \$last_entry =
   array(\"log-ID\" => \"" .$data_filename."\", \"
   user\" => \"".$user."\", \"current line\" =>
   \"".($i + $current_line)."\", \"count of rows\"
   => \"".$last_entry["count of rows"]."\", \"
   insert key\" => \"".$current_insert_key."\", \"
   last insert time\" => \"".$last_insert_time."\")
   ; ?>";
writeFile("../tmp_log_files/","le_".$user."_".
   $building.".php", $input_stream_last_entry);

// returns the energy consumption line, of the last
    4 minutes (= 240 lines)
$return_data_array = last_insertedLine(
   $current_con_event_array);
$next_data_array_index = count($return_data_array);
$return_data_array[$next_data_array_index] = $i +
   $current_line; // current inserted line number
$next_data_array_index++;
$return_data_array[$next_data_array_index] =
   $current_date_timestamp; // timestamp of the
   last inserted consumptionevent

// the whole energy consumption of the last
   consumption event, which will be used for the
   other events inside the data base (e.g.
   production, ...)
$next_data_array_index++;
$return_data_array[$next_data_array_index] = 0;
for ($j = 0; $j < count($current_con_event_array);
   $j++)
     $return_data_array[$next_data_array_index] +=
        $current_con_event_array[$j];

$return_data_array[$next_data_array_index] = round
   (($return_data_array[$next_data_array_index] /
   3600000), 14); // energy consumption in kWh

// energy sum over all devices, for every data line
    (is needed for the circuit power / circuit
   average event)
$next_data_array_index++;
$return_data_array[$next_data_array_index] =
   get_ZeroMatrix(count($current_cir_event_array));
for ($k = 0; $k < count($current_cir_event_array);
            $k++) {
              $return_data_array[$next_data_array_index][$k
                 ][0] += $current_cir_event_array[$k][0]; //
                 timestamp
              $return_data_array[$next_data_array_index][$k
                 ][1] += $current_cir_event_array[$k][1]; //
                 circuit power event
          }

          return $return_data_array;
      }
}
```

A.2. SQL-Statement: cons_event_stream

```
<?php
...
$cons_event_stream = "INSERT INTO consumptionevents (owner,
    device, start, duration, consumption) VALUES ";
...
// creates the consumption event stream
          for ($k = 0; $k < count($current_devices); $k++)
          {
              $current_dev_energy = round(((
                 $current_con_event_array[$k]) / 3600000),
                 14); // energy in kWh (1/3600 ... hour
                 coefficient)
              $cons_event_stream .= "('$user', '".
                 $current_devices[$k].substr($user, 0, 4)."',
                  '$current_date_timestamp', '$i', '
                 $current_dev_energy')";

              if($k == (count($current_devices) - 1))
                  $cons_event_stream .= ";";
              else
                  $cons_event_stream .= ", ";
          }

          $connection->addEventTDI($cons_event_stream); //
             inserts the current expected data lines into the
              data base (only one insert command)
```

```
...
?>
```

A.3. Function: addingEvents(...)

```
// This method can be extended anytime
// adds additional data events into the Mj[U+FFFD]ir data base
   system, so that the most widgets are able to run with
   the test data
if (!function_exists("addingEvents"))
{
    function addingEvents($con, $building, $user,
       $table_name, $energy_data_prod = 0, $timestamp = 0,
       $energy_data_circuit = array())
    {
        switch ($table_name)
        {
            case "circuit_avg":
                $cir_id = $con->getCircuitID($user,
                   $building);
                $current_hour = date("G",
                   $energy_data_circuit[count(
                   $energy_data_circuit) - 1][0]);
                $current_minute = date("i",
                   $energy_data_circuit[count(
                   $energy_data_circuit) - 1][0]);
                $min_max = array();
                $min_max = $con->getCircuitAvg($cir_id,
                   $current_hour);
                $min = $min_max["min"];
                $max = $min_max["max"];

                // sets a new minimum start value, and the
                   number of recorded days, if the current
                   time is 00:00
                if(($current_hour == "0") && (
                   $current_minute == "03")) {
                    $min = $energy_data_circuit[0][1];
                    $current_n = $con->getCirAvgRecDays(
                       $cir_id);
                    $new_cir_n = $current_n + 1;
                    $con->updateCirAvgRecDays($cir_id,
                       $new_cir_n);
                }

                // sets the new min/max value
                for ($i = 0; $i < count(
                   $energy_data_circuit); $i++) {
                    if($max < $energy_data_circuit[$i][1])
                       $max = $energy_data_circuit[$i][1];
                    if($min > $energy_data_circuit[$i][1])
                       $min = $energy_data_circuit[$i][1];
                }

                $con->updateCirAvgMinMax($cir_id,
                   $current_hour, $min, $max); // circuit
                   average update
                break;
            case "circuit_power":
                $cir_power_event_stream = "INSERT INTO
                   circuit_power (id, timestamp , power)
                   VALUES ";
                $cir_id = $con->getCircuitID($user,
                   $building);

                // creates the circuit power event stream
                for ($k = 0; $k < count(
                   $energy_data_circuit); $k++) {
                    $cir_power_event_stream .= "('$cir_id',
                       ".$energy_data_circuit[$k][0].", ".
                       $energy_data_circuit[$k][1].")";

                    if($k == (count($energy_data_circuit) -
                       1))
                       $cir_power_event_stream .= ";";
                    else
                       $cir_power_event_stream .= ", ";
                }

                $con->addCirPowerEventTDI(
                   $cir_power_event_stream);
                break;
            case "production":
                $current_date = date("d.m.y", $timestamp);
                    // creates the date of the current used
                    timestamp
                $date_format = DateTime::createFromFormat('
                    d.m.y H:i:s', $current_date.' 00:00:00')
                    ; // creates an instance of the DateTime
                    -object, with a special format
                $current_date_timestamp = $date_format->
                    getTimestamp(); // timestamp of the
                    current date at 00:00

                $existing_agg_cons_array = array();
                $existing_agg_cons_array = $con->
                    getExistingAggCons($user,
                    $current_date_timestamp);

                if ($existing_agg_cons_array[0] == "EXISTS
                    ") {
                     $existing_agg_cons_array[1] +=
                         $energy_data_prod;
                     $con->updateAggData($user,
                         $current_date_timestamp,
                         $existing_agg_cons_array[1]);
                     /*fwrite($file2,
                         $existing_agg_cons_array[1]."\n");
                     fwrite($file2, $current_date."\n");*/
                }
                else
                     $con->addAggData($user,
                         $current_date_timestamp,
                         $energy_data_prod, 0);
                break;
        }
    }
}
```

A.4. Temporary File: cif_user_building x.php

```
<?php $current_inputted_files = array("file 1" => array("
    file name" => "dataset_2013-12-07.csv", "date" =>
    "01.04.21", "input state" => "AUTOMATICALLY STOPPED"));
    ?>
```

A.5. Temporary File: le_user_building x.php

```
<?php $last_entry = array("log-ID" => "dataset_2013-12-07.
    csv", "user" => "Administrator", "current line" =>
    "3600", "count of rows" => "3600", "insert key" => "", "
    last insert time" => ""); ?>
```

A.6. Test Devices: building x.php

```
<?php
$building_devs = array(
    "device 0" => array("ID" => "000D6F0002906FA7", "name"
        => "coffee_machine", "type" => 37),
    "device 1" => array("ID" => "000D6F0002907BA2", "name"
        => "washing_machine", "type" => 105),
    "device 2" => array("ID" => "000D6F0002907BC8", "name"
        => "radio", "type" => 22),
    "device 3" => array("ID" => "000D6F0002907BDF", "name"
        => "water_kettle", "type" => 33),
    "device 4" => array("ID" => "000D6F0002907BF5", "name"
        => "fridge_and_freezer", "type" => 13),
    "device 5" => array("ID" => "000D6F0002907C89", "name"
        => "dishwasher", "type" => 107),
    "device 6" => array("ID" => "000D6F0002908150", "name"
        => "kitchen_lamp", "type" => 8),
    "device 7" => array("ID" => "000D6F0002908162", "name"
        => "TV", "type" => 14),
    "device 8" => array("ID" => "000D6F00029C506A", "name"
        => "vacuum_cleaner", "type" => 91)
);
?>
```

A.7. Process Hours Determining

```
 if (last_insert_process == "AUTOMATICALLY STOPPED") {
                              ...

                              $.ajax
```

```
        ({
            type: "POST",
            url: "widgets/TestDataInput/core/
               tdi_start_df_lines.php",
            data: "user=" + selectedUser + "&
               building=" + selectedBuilding +
               "&ci_file=" + log_id_array[1] +
               "&last_insert_process=" +
               last_insert_process,
            success: function (data) {
                object = JSON.parse(data);
                data_array = object.data;
                current_state = data_array[2];
                   // next click-state ("state:
                    choose rows")
                whole_count_of_rows = parseInt(
                   data_array[1]);
                line_counter = parseInt(
                   data_array[0]) + 3600;
                hour_counter = Math.round(
                   line_counter / 3600);

                ...
            }
        });
    } else {
        ...

        $.ajax
        ({
            type: "POST",
            url: "widgets/TestDataInput/core/
               tdi_start_df_lines.php",
            data: "building=" +
               selectedBuilding + "&ci_file=" +
                selectedDataFile + "&
               last_insert_process=" +
               last_insert_process,
            success: function (data) {
                object = JSON.parse(data);
```

```
                    data_array = object.data;
                    current_state = data_array[1];
                       // next click-state ("state:
                        choose rows")
                    whole_count_of_rows = parseInt(
                       data_array[0]);
                    line_counter = 3600;
                    hour_counter = 1;

                    // html-content, which has to
                       be added to this widget
                    $("#test_data_input" +
                       widget_id).append('<div
                       class="tdi-data" id="
                       tdi_data_rows_div' +
                       widget_id + '"><p id="
                       tdi_data_rows_1' + widget_id
                        + '">' + selectedDataFile +
                        '</p></div>');
                    selectRowsContent("#
                       tdi_select_rows",
                       line_counter, hour_counter,
                       data_array[0]);

                    ...
                }
            });
        }
```

A.8. Method: timerIP()

```
// simulates a countdown
    function timerIP() {
        if (timer_index > 0) {
            $(element_id).text(timer_index + " seconds!");
            timer_index = timer_index - 1;
        } else {
            if (element_id == ("#tdi_data_ip_prep" +
               widget_id)) {
                clearInterval(current_timer_id_counter); //
                    deletes the interval-event, of the
```

```
                  entered interval-id
                drawInsertProcess(help_array_devices,
                  current_file_ip);
            } else if (element_id == ("#
               tdi_input_process_infotext_stop_counter" +
               widget_id)) {
                clearInterval(current_timer_id_counter);
                ipDelete(process_var, st_as_fs_text);
            } else {
                clearInterval(current_timer_id_counter);
                ipDelete(process_var, st_as_fs_text);
            }
        }
    }
```

A.9. Method: insertProcess()

```
// simulates the test data insert process through a
   simulation block
    function insertProcess() {
        var selectedBuilding = $("#tdi_select_building" +
           widget_id).find(':selected').text();
        var selectedUser = $("#tdi_select_users" +
           widget_id).find(':selected').text();
        window_width = $(window).innerWidth();
        current_screen_res = getScreenResolution();

        if ((process_var == "NEXT LINES") && (
           current_row_expected < current_count_of_rows) &&
            (current_row == current_row_expected)) {
            current_row_expected = current_row_expected +
               240;

            $.ajax
            ({
                type: "POST",
                url: "widgets/TestDataInput/core/
                  tdi_input_process.php",
                data: "authkey=" + dashboard.key + "&user="
                    + selectedUser + "&building=" +
                  selectedBuilding + "&current_row=" +
        current_row + "&current_row_expected=" +
          current_row_expected ,
    success: function (data) {
        var object , data_array ,
           help_variable_data , current_date;
        object = JSON. parse(data);
        data_array = object.data;

        // swaps the last inserted lines table
           content
        // swaps the column contents "Line"
        help_variable_data = $("#
           tdi_table_line1").text();
        $("#tdi_table_line0").text(
           help_variable_data);
        current_date = new Date(data_array[2] *
            1000); // creates a new date−object
            of the unix−timestamp; the
           timestamp will be converted into
           milliseconds
        $("#tdi_table_line1").text(current_date
           .getHours() + ":" + current_date.
           getMinutes());

        // swaps the column contents of the
           devices
        for (i = 0; i < data_array[0].length; i
           ++) {
             help_variable_data = $("#tdi_table_
                " + help_array_devices[i] + "
                _next" + widget_id).text();
             $("#tdi_table_" +
                help_array_devices[i] + "_prev"
                + widget_id).text(
                help_variable_data);
             $("#tdi_table_" +
                help_array_devices[i] + "_next"
                + widget_id).text(data_array[0][
                i]);
             console.log("#tdi_table_" +
                    help_array_devices[i] + "_next"
                    + widget_id + ": " + $("#
                    tdi_table_" + help_array_devices
                    [i] + "_next" + widget_id).text
                    ()); // probleme beim Einf[U+FFFD]gen
                }

                current_length_pb = (data_array[1] /
                    current_count_of_rows) * 100; //
                    computes the current length of the
                    inner process bar
                current_pc_pb = Math.round(
                    current_length_pb);
                $("#tdi_input_process_inner_pb" +
                    widget_id).css({width:
                    current_length_pb + '%'}); //
                    refreshes the inner process bar
                    length
                $("#tdi_input_process_inner_pb_text" +
                    widget_id).text(current_pc_pb + "
                    %"); // refreshes the inner process
                    bar text in per cent

                if (current_row_expected ==
                    current_count_of_rows) {
                     $("#tdi_input_process_stop" +
                        widget_id).attr("disabled", true
                        );
                     $("#tdi_input_process_stop" +
                        widget_id).css({background: "#9
                        d9d9d"});
                }

                current_row = parseInt(data_array[1]);
            }
        });
    } else if ((process_var == "STOP") && (
        current_row_expected < current_count_of_rows)) {
         clearInterval(current_timer_id_ip);
         var used_insert_key = $("#tdi_insert_key" +
   widget_id).text();

$.ajax
({
    type: "POST",
    url: "widgets/TestDataInput/core/
       tdi_ip_stop.php",
    data: "user=" + selectedUser + "&building="
        + selectedBuilding + "&insert_key=" +
       used_insert_key,
    success: function (data) {
        var object, data_array;
        object = JSON.parse(data);
        data_array = object.data;

        if(data_array[0] != "OUT OF TIME") {
            $("#tdi_input_process_infotext_stop
               " + widget_id).text(data_array
               [0]);
            $("#tdi_input_process_infotext_stop
               " + widget_id).show();
            $("#
               tdi_input_process_infotext_stop_counter
               " + widget_id).text(data_array
               [1]);
            $("#
               tdi_input_process_infotext_stop_counter
               " + widget_id).show();

            timer_index = 5;
            element_id = "#
               tdi_input_process_infotext_stop_counter
               " + widget_id;
            st_as_fs_text = data_array[2];
            current_timer_id_counter =
               setInterval(timerIP, 1000);
        } else ipDelete("OUT OF TIME",
           data_array[1]);

    }
```

```
        });

        $("#tdi_input_process_continue" + widget_id).
            attr("disabled", true);
        $("#tdi_input_process_continue" + widget_id).
            css({background: "#9d9d9d"});
        $("#tdi_input_process_break" + widget_id).attr
            ("disabled", true);
        $("#tdi_input_process_break" + widget_id).css({
            background: "#9d9d9d"});
        $("#tdi_input_process_stop" + widget_id).attr("
            disabled", true);
        $("#tdi_input_process_stop" + widget_id).css({
            background: "#9d9d9d"});
    } else if (current_row == current_count_of_rows) {
        clearInterval(current_timer_id_ip);

        if (current_count_of_rows !=
            whole_count_of_rows)
              process_var = "AUTOMATICALLY STOPPED";
        else if (current_count_of_rows ==
            whole_count_of_rows)
              process_var = "FINISHED SUCCESSFUL";

        $.ajax
        ({
            type: "POST",
            url: "widgets/TestDataInput/core/
                tdi_ip_as_fs.php",
            data: "user=" + selectedUser + "&building="
                 + selectedBuilding + "&process_var=" +
                process_var,
            success: function (data) {
                var object, data_array;
                object = JSON.parse(data);
                data_array = object.data;

                $("#tdi_input_process_infotext_fs_as" +
                    widget_id).text(data_array[0]);
                $("#tdi_input_process_infotext_fs_as" +
                     widget_id).show();
                 $("#
                    tdi_input_process_infotext_fs_as_counter
                    " + widget_id).text(data_array[1]);
                 $("#
                    tdi_input_process_infotext_fs_as_counter
                    " + widget_id).show();

                 timer_index = 5;
                 element_id = "#
                    tdi_input_process_infotext_fs_as_counter
                    " + widget_id;
                 st_as_fs_text = data_array[2];
                 current_timer_id_counter = setInterval(
                    timerIP, 1000);
              }
          });

          $("#tdi_input_process_break" + widget_id).attr
             ("disabled", true);
          $("#tdi_input_process_break" + widget_id).css({
             background: "#9d9d9d"});
          $("#tdi_input_process_continue" + widget_id).
             attr("disabled", true);
          $("#tdi_input_process_continue" + widget_id).
             css({background: "#9d9d9d"});
          $("#tdi_input_process_stop" + widget_id).attr("
             disabled", true);
          $("#tdi_input_process_stop" + widget_id).css({
             background: "#9d9d9d"});
      }
   }
```

References

[1] plugwise Plug: Installation manual. Technical report, Plugwise B.V, 2018. URL `https://www.plugwise.com/wp-content/uploads/2021/02/Plugwise-User-Manual-Plug-ML.pdf`.

[2] A. Monacchi. Installation guide v0.1 – July 23rd 2015. Technical report, Alpen-Adria-Universität, 2015. URL `https://sourceforge.net/projects/mjoelnir/files/instructions.pdf/download`.

[3] W. Elmenreich and D. Egarter. Design guidelines for smart appliances. In *Proceedings of the 10th International Workshop on Intelligent Solutions in Embedded Systems (WISES'12)*, Klagenfurt, Austria, July 2012.

[4] F. Versolatto, A. M. Tonello, A. Monacchi, W. Elmenreich, D. Egarter, T. Khatib, A. Kercek, M. Biondi. Validation and analysis of results. Technical report, MONERGY: ICT solutions for energy saving in Smart Homes, 2015. URL `https://www.monergy-project.eu/deliverables/deliverable_4.pdf`.

[5] Kristina Lizenberger. Die Design Challenge – Wie funktioniert Design Thinking? blog article, nativeDigital. URL `https://nativdigital.com/design-challenge/`. accessed: April, 2021.

[6] Manuel Herold, Andrea Monacchi, Dominik Egarter. Mjölnir: The open source energy advisor. Technical report, Alpen-Adria-Universität, 2016. URL `https://master.dl.sourceforge.net/project/mjoelnir/documentation_0-3-1.pdf?viasf=1`.

[7] Mohit Jain, Vikas Chandan, Marilena Minou, George Thanos, Tri K Wijaya, Achim Lindt, Arne Gylling. Methodologies for effective demand response messaging. Technical report, 2007-2013. URL `http://nes.aueb.gr/publications/SGC_submitted.pdf`.

[8] A. Monacchi, D. Egarter, W. Elmenreich, S. D'Alessandro, and A. M. Tonello. GREEND: An energy consumption dataset of households in italy and austria. In *Proc. IEEE International Conference on Smart Grid Communications (SmartGridComm'14)*, Venice, Italy, 2014.

[9] A. Monacchi, F. Versolatto, M. Herold, D. Egarter, A. Tonello, and W. Elmenreich. An open solution to provide personalized feedback for building energy management. *Journal of Ambient Intelligence and Smart Environments*, 9(2): 147–162, 2017.

[10] Sandra Holze. 7 Erfolgsregeln für eine Challenge, die dir garantiert Kunden bringt. blog article. URL `https://sandraholze.com/7-erfolgsregeln-challenge/?cn-reloaded=1`. accessed: April, 2021.