eTEACHER

D3.8: Dashboarding Demonstrator

WP 3, T 3.5

Date of document

November, 2019 (M26)

Authors: Michael Hornke, Nicolas Mayer (ASC)

eTEACHER

EE-07-2016-2017

Innovation Action

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 768738.

Disclaimer

The information reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains.

Technical References

Project Acronym	**eTEACHER**
Project Title	end-user Tools to Empower and raise Awareness of Behavioural Change towards EneRgy efficiency
Project Coordinator	Noemi Jimenez Cemosa noemi.jimenez@cemosa.es
Project Duration	1 October 2017 – 31 September 2020

Deliverable No.	D3.8
Dissemination Level	CO[1]
Work Package	WP 3 – Empower tools II – User Friendly Solutions
Task	T 3.5 – Dashboarding
Lead beneficiary	7 (ASC)
Due date of deliverable	30 September 2019
Actual submission date	15 November 2019

[1] CO = Confidential, only for members of the consortium (including the Commission Services)

Versions

Version	Person	Partner	Date
V1.0	Michael Hornke Nicolas Mayer	ASC	13 November 2019

Table of Content

List of Tables

List of Figures

0 Introduction

The eTEACHER concept consists of encouraging and enabling energy behaviour change of building users by means of continuous interventions displayed through a set of empower tools to drive informed decisions in order to save energy and optimise indoor environment quality. These empower tools are a set of ICT solutions that ensures friendly connection in between end-users and building systems, implement continuous behavioural change interventions and provide tailored advice.

The tools can be classified into:

- The **BACS add-ons** (What-if-Analysis, data processing and universal BACS/monitoring system interface)
- The **user-friendly solutions**

This deliverable is about the first prototype of Task 3.5 – Dashboarding which is part of the user-friendly solutions.

The Dashboarding component is a tool to collect facts and to extract insights from the facts to visualize them in a graphical manner, like charts. An example of this, is the consumption of an apartment or the different gamification visualizations. With its web based interface it is compatible to nearly all browsers and can be used on multiple platforms, which is quite important, since the component should run on different devices, like iOS, Android, Windows and macOs. It is integrated into the Advisor App, but also can run in standalone mode. The standalone mode is interesting in scenarios, where it should be used to show information in more public areas, like for example entrance halls or offices for multiple persons. It comes in an easy deployable docker container, so the administrative effort to setup the system is minimized. The UI is assisted by the API, which does all the business logic that is necessary to run such a data analysis platform. Users are able to consume statistics, driven from the Insights, presented in chart widgets, which can be configured per user. Altogether, this component is meant as a tool to make more information available from the collected data of the project and visualize it. Through the API all statistics can be shared between other components and can be fed from other components. This makes the Dashboarding component flexible and offers the capability to meet all needs rising in this project.

This deliverable describes the dashboarding component from different perspectives. These perspectives are developers, who want to integrate it (like technical project partners), administrators and end users.

Described in this document is the architecture, the setup of the system, the API description and brief outlook, how the component can be improved and which additional features could be interesting after the project ends.

0.1 Deliverable Purpose, Scope and Context

The purpose of this deliverable is to accompany the prototype implementation of T3.5. As such, its main purpose is to briefly clarify the scope of the prototype and to show the download, installation instructions and the use of the API of the software. The document is limited in length as the main focus of the task is the software itself rather than its accompanying document.

0.2 Document Status and Target Audience

This deliverable is qualified as confidential in the Description of Action (DoA), for this reason the information gathered and their distribution is mainly for the consortium members.

0.3 Document Structure

This deliverable is broken down into the following sections:

- Section 0 (Introduction) provides an introduction for this deliverable including a general overview of the project and outlines of the purpose, scope, context, status and target audience of this deliverable.
- Section 1 (Scope and Relationship) clarifies the context and scope of the software prototype deliverable and its relationship with other architectural modules.
- Section 2 (State of the Prototype) gives a brief overview about implemented features and the current state of the provided prototype
- Section 3 (Requirements and Preparations) outline the prerequisites, installation instructions and methods to use the software component
- Section 4 (Limitation, Research and further developments) shows the limitations of the current prototype
- Section 5 (Summary and Conclusion) gives a brief summarization about the component and this document.

1 Scope and Relationship

The Dashboarding component is used to visualize gamification data as well as consumption data to the end user. It is part of the Advisor App (T3.2) of the user-friendly solutions and so usable on all common platforms like macOS, Windows, Android and iOS. Further, it can be run in standalone mode as well. It connects to the Advisor Core module from T3.1 and the UBCI container from WP2. The authorization and authentication are handled by the Advisor Core (T3.1) as well, as the profile data and aggregation of the gamification data. Data regarding the consumption is consumed from the BACS component and gets prepared for the visualization in the dashboarding component.

Therefore, the dashboarding component provides functionalities to support and ease the access and interaction with the integrated components, which can be seen in Figure 1: eTEACHER Software Component Overview.

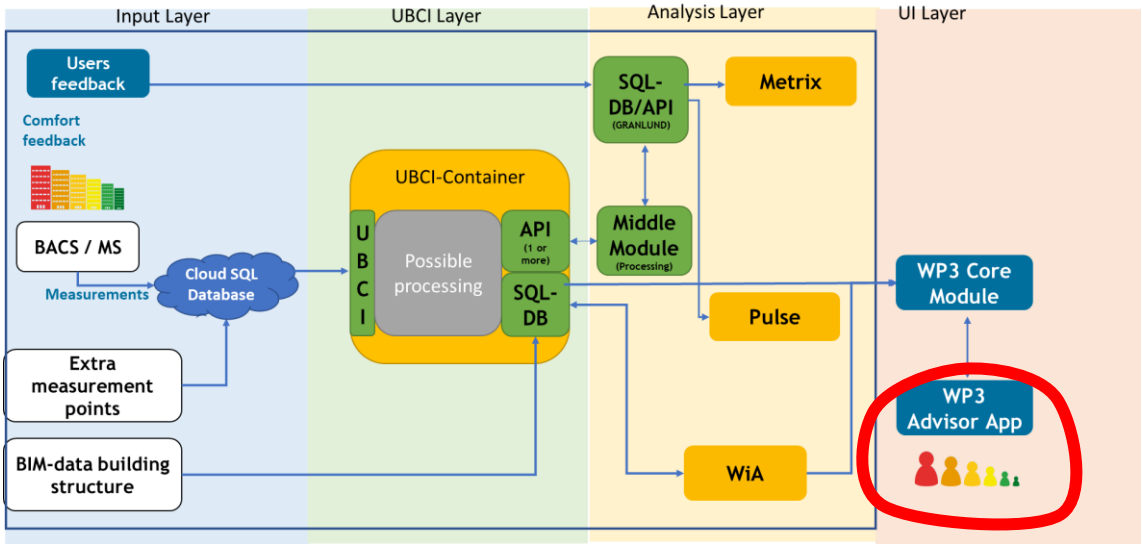

Figure 1: eTEACHER Software Component Overview

While Task 3.2 works as a visual central entry point for eTEACHER users, the Dashboarding is a central part of it. Thereby, it does reduce and convert the information and make certain things more accessible to the end user, by visualizing it. Since the data is quite different in nature, the server logic of the Dashboarding is projecting those different data formats to one, which is more general and can be used for visualizations. Because of that, for every data/different visualization, extra development is needed. But the component is made so flexible, that as less as possible have to be added and that all general stuff is generalized to a degree, to allow rapid development of those extra visualizations. Because of its generalization, the dashboarding component can be used in completely different contexts, which is important to keep component exploitable for other research projects or as a commercial component after the project.

Figure 2 shows, where the software component is located in WP3 infrastructure, provided by T3.1. In Figure 3 the internal structure of the component is described and its interaction layers to other components of WP3.

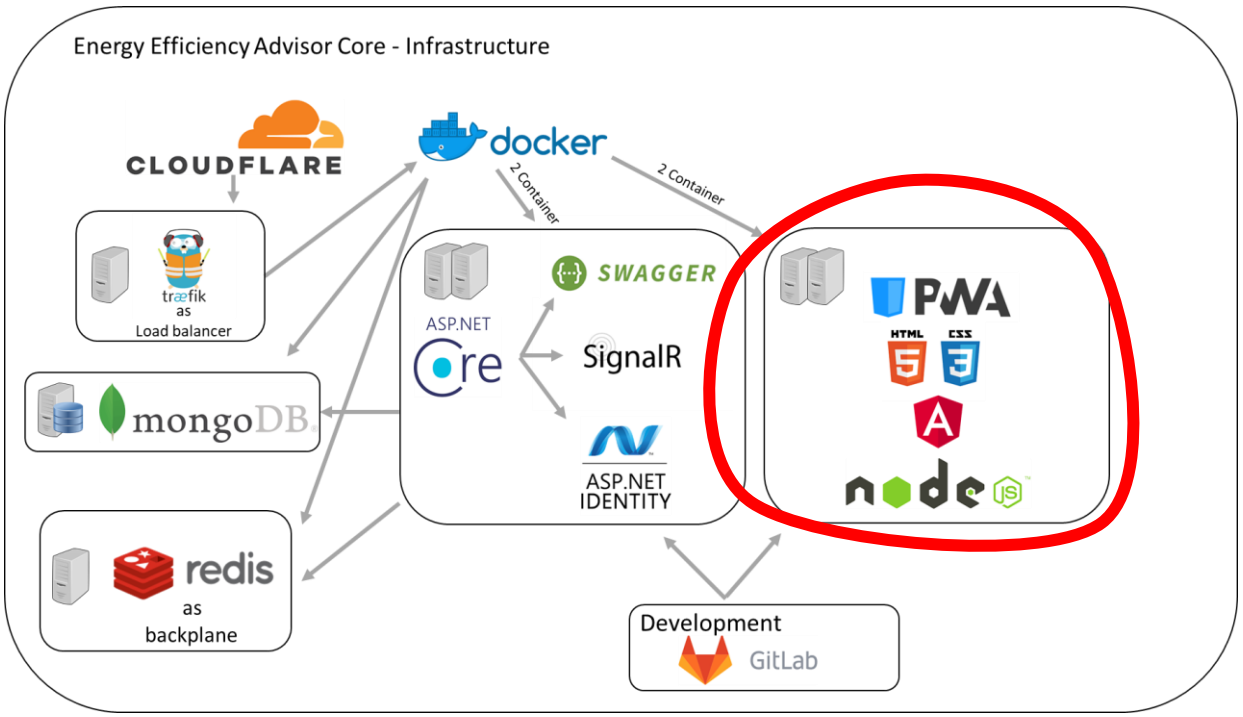

Figure 2: Frameworks and Server Infrastructure

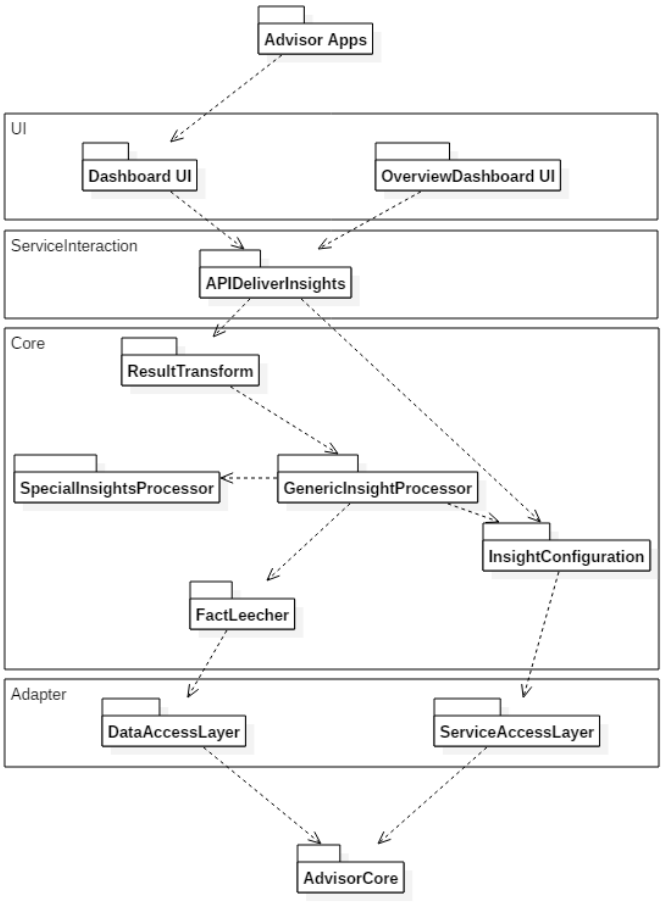

Figure 3 - UML Package Diagram

2 State of the Prototype

The current prototype can be accessed in the eTEACHER web app[2] and the Android/iOS App.

The following table lists the key features of the Dashboarding component and their status accordingly. Everything is on schedule and all features got provided. Further improvement is possible, to extend the functionalities at the end of the project.

Feature	Status
REST API	**provided**
OpenAPI Interface	**provided**
Deployable Docker Container	**provided**
Integration of MongoDB for Data Storage	**provided**
Fact Injector (Information Input)	**provided**
Data Integrity Checker (Information Check)	**provided**
Flexible Insight Generator (Verify Information)	**provided**
Different Chart Types (Consume Insights)	**provided**
Account Based Settings (Configuration per User)	**provided**
Generalized Chart Component	**provided**
Generalized Dashboard UI	**provided**
Integration of T3.1 Authorization	**provided**
Integration into the Advisor T3.2	**provided**
Integration of BACS (WP2)	**provided**
Integration of Gamification Datasource (T3.3)	**provided**
Ranking Dashboard	**provided**
Energy Distribution Dashboard	**provided**

[2] https://eteacher-app.ascora.eu/

Carbon Footprint Dashboard	**provided**
Energy Consumption Dashboard	**provided**

Table 1: Feature Matrix

3 Requirements and Preparations

This section provides information about what administrators and software developers need to prepare in order to use the functionalities of the prototype.

The server-side part will be executed by administrators, whereas the Client API and REST interfaces will be used by developers.

3.1 Server Side (System Administrators)

The Dashboarding Service Repository manages the storage and provisioning of Proxy Service Wrapper objects. It consists of four Docker images:

1. Dashboard UI – The frontend for the data analysis (WebUI)
2. Dashboard API – The Core of the system, including the API
3. Dashboard Runner – The Runner executes the next pending job from the cronjob pipeline. Multiple Runners can speed up the system, as different jobs are executed simultaneously. Runners executes all jobs in the Dashboarding component that are stored in a pipeline to generate statistics, check data integrity, run fact leechers to give a few examples.

3.1.1 Requirements

Some preparations to install the Dashboarding subcomponent must be set to run a stable instance of it. It is recommended to use the Linux distribution Ubuntu 18.04.2 LTS as the operating system, due to dependencies of the used Docker, at least version 18.09.5. Any other operating system can be used too, but it cannot be guaranteed that it will work successfully with the Docker version, which is used. According to this, Docker must be installed. Furthermore, it must be ensured to have access to the Linux root user, as resources are used that are in need of root privileges.

The following commands are required to set up the Docker Engine on Ubuntu 18.04.1 LTS:

```
1. sudo apt update
2. sudo apt install apt-transport-https ca-certificates curl software-properties-common
3. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
4. sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
bionic stable"
5. sudo apt update
6. sudo apt install docker-ce
7. sudo systemctl status docker
8. Output Check:

docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2018-07-05 15:08:39 UTC; 2min 55s ago
     Docs: https://docs.docker.com
 Main PID: 10096 (dockerd)
    Tasks: 16
   CGroup: /system.slice/docker.service
           ├─10096 /usr/bin/dockerd -H fd://
           └─10113 docker-containerd --config /var/run/docker/containerd/containerd.toml
```

The following commands are required to set up the MongoDB Instance utilized by the system. This is only meant for developing purposes, as the component utilizes the MongoDB of the Advisor Core (T3.1).

```
1. docker pull mongo
2. docker run -d -p 27017-27019:27017-27019 --name mongodb mongo

MongoDB is running now.
```

3.1.2 Installation

The following commands show, how to setup the Dashboard UI, the Dashboard API and the Dashboard Runners. The installation can be done with the installed docker engine (3.1.1) and will set up the Dashboard component.

Installing the Dashboard UI:

```
1. docker pull registry.ascora.eu:8443/discovery/ascora-discovery-ui:latest
2. docker run -d -p 8000:80 --name discovery-ui discovery-ui

Discovery UI is running on Port 8000 now.
```

The Dashboard UI is the frontend component for configuring and consuming statistics.

Installing the Dashboard API:

```
1. docker pull registry.ascora.eu:8443/hornke/ascora-discovery-api:latest
2. docker run -v /host/directory/config.php:/var/html/www/config.php -d -p 80:80 --
   name discovery-api discovery-api

Discovery API is running on Port 80 now.
```

The Dashboard API provides all necessary data for serving the UI. Additionally, there is a CLI that provides system functions for administrators.

Installing the Dashboard Runner:

```
1. docker pull registry.ascora.eu:8443/webteam/discovery-runner:latest
2. docker run -d --name discovery-runner discovery-runner

Discovery Runner is running now.
```

The Runner contacts the API and executes the next pending cronjob. A good strategy is to run multiple runners to execute pending cronjobs simultaneous. This will influence the refresh rate of the charts in a positive way. If the statistics are delayed and get more and more delayed, further instances are necessary and can be started pretty easy like this.

These docker containers are needed to run the Dashboard properly.

After setting up all docker containers, the installation can be checked by listing the docker services. The listing can be show by executing the following command:

```
docker ps
```

3.2 REST-API (Developers)

The main API is included in the Dashboard API. It is responsible for, fact injection, providing chart data, and all other functions needed by the Dashboard UI. The documentation of the API is a living document provided in the OpenAPI standard, which you can access at https://discovery-api.ascora.de/docs/index.html. It explains how to use the Dashboard API with the provided RESTful interfaces and can be used to directly access the API. Further, a file is provided, with which clients for different programming languages can be created.

If no client is generated, it can be used as a document, with which developers with knowledge of the general use of RESTful interfaces are able to connect and use the component. For each interface, a description is presented as a table, which includes all necessary request and response parameters and their expected value types and values, if applicable.

3.2.1 API for a Specific Programming Language

The following steps are guidance for generating the clients under Ubuntu 18.04. For other versions of OS or application server, please, consult the documentation for the distributions you are using.

First step would be to install Java.

```
$ sudo apt-get update
$ sudo apt-get install default-jdk
```

For setting up a client to access the Dashboarding components, OpenAPI Generator can be used, which can be found online[3].

The software generates clients, servers, and documentation from OpenAPI 2.0/3.x documents. It supports 50+ client generators (for different programming languages and frameworks, ranging from Haskell to AngularJS), which can be easily used to generate code to interact with any server which exposes an OpenAPI document. Those clients can even be customized, to fit the developers special needs. Many of these generators use Inversion of Control, to support updating the client side, without many influences on the code base.

Currently, version 3.3.3 is used, which can be found on maven[4].

Newer versions should work as well but are not tested at the current state.

Further, a json file describing the software component is needed, which can be found on the following website and should be downloaded in the same directory.

[3] https://openapi-generator.tech/

[4] http://central.maven.org/maven2/org/openapitools/openapi-generator-cli/3.3.3/openapi-generator-cli-3.3.3.jar

https://discovery-api.ascora.de/docs/swagger.json

After downloading, the generator can be executed with the following command and will generate in this example the typescript-angular client, which is used in T3.2

```
$ java -jar ./openapi-generator-cli-3.3.3.jar  generate -i ./swagger.json
-g typescript-angular -o ./OpenAPIEteacher/ --additional-properties
ngVersion=7.1.2
```

3.2.2 Content Format

This section is about a communication example, that show the structure of the common Dashboard objects for the communication with other components via the REST interface.

The responses of the RESTful interface are standardized. They always have two parts. The **meta** part and the **data** part. The meta part contains some useful information about the request and the answer (like example error codes and error messages). The data part contains the data of the response, related to the request.

The following listings describe the interface along the **fact injection call**. This is the way to send information to the system. The injection limit is capped at 10000 objects per call at the moment, but that could be increased easily if needed.

3.2.3 Request Example for Fact Injection

```
{
  "source": "e-teacher",
  "injectKey": "xyz",
  "value": [
      {
      "occurDate": "2019-10-15T12:03:22.000-0100",
      "originId": "I-AM-A-UNIQUE-STRING",
      "accountId": "5b9b745e6600bd66c011d941",
      "value": {
            "test1": "bla1",
            "test2": "foo1",
            "email": "bar2"
      }
    }
  ]
}
```

Listing 1: Fact Injection Request to Provide Data into the Dashboard Component

Key	Description
source	the project name
injectKey	security: only with this key can be injected
value	array of facts
value[i].occurDate	the date timestamp of the fact
value[i].originId	unique id of this fact
value[i].accountId	user id for this fact, if you have personalized data from users, or any other data where different users play a role (like gamification statistics)
value[i].value	the content of the fact. This can be any JSON object.

3.2.4 Positive Response

In case of a fact injection request, the data part of the response contains information that summarizes the processing of the request. In this case one fact was injected, no fact was rejected, and no errors occurred.


```
{
    "meta": {
        "project": "e-techer",
        "version": "v1",
        "status": true,
        "statusCode": 200,
        "statusMessage": "facts saved successfully",
        "responseTimeMs": 392.30012893677,
        "count": 0,
        "page": 0,
        "perPage": 0
    },
    "data": {
        "totalFacts": 1,
        "importedFacts": 1,
        "rejectedFacts": 0,
        "importErrors": []
    }
}
```


Listing 2: Positive Response to a Fact Injection Request

3.2.5 Negative Response

In case of a problem during the fact injection, the API will respond with an error. In this case, the injectKey was not given correctly, so the API responded with an error. This is a security feature to prevent unauthorized people or services from injecting data to slow down our systems etc.


```
{
      "meta": {
      "project": "e-teacher",
      "version": "v1",
      "status": false,
      "statusCode": 1010,
      "statusMessage": "fact type injection not permitted, wrong injection key",
      "responseTimeMs": 6.2999725341797,
      "count": 0,
      "page": 0,
      "perPage": 0
    },
    "data": []
}
```


Listing 3: Possible Negative Response to a Fact Injection Request

The meta part of the response contains a false as status, a statusCode other than 200 and a status message which is not empty. These attributes indicate that an error occurred. Additionally, the data part is false, as there is no data responded in an error case.

Key	Description
meta.project	This project string contains the project part of the request URL
meta.version	The version string contains the API version of the request URL (v1, v2, …)
meta.status	The status Boolean contains true in case of success, false on error
meta.statusCode	The statusCode contains 200 on success, or a predefined code on error
meta.statusMessage	The error message contains the reason in a displayable form on error
meta.responseTimeMs	The time that was consumed by the API logic to produce the result
meta.count	In case of reading data, the count contains the total results
meta.page	In case of reading data, the count contains the current selected page (>=0)
meta.perPage	In case of reading data, the count contains the results per page (>0)
data	The data object contains the payload of the response. Normally, this will contain statistic data to be displayable on the dashboard. In error case this will be false. A detailed description of the data flag can be found in the API description for every different route.

Table 2: Message Description

4 The Discovery UI (Users and Administrators)

The Dashboard UI enables users to consume statistics generated from facts that are gained in the project. With an account based approach each user configure the statistics in a way that fits best for his needs. This configuration is done from Administrators, so that the final end users of eTEACHER can just use the app.

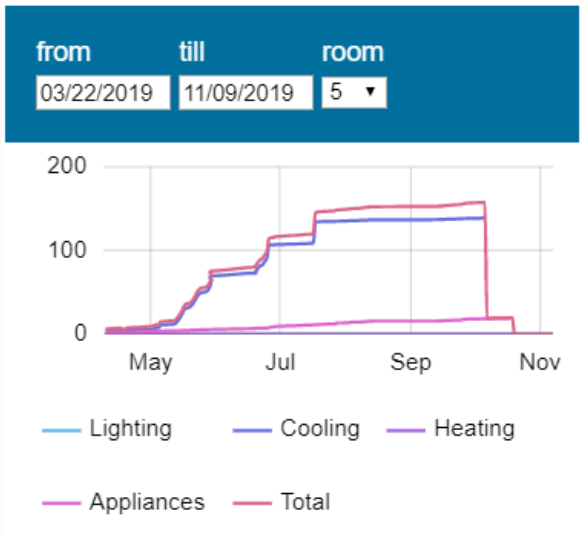

Figure 5: Energy Consumption

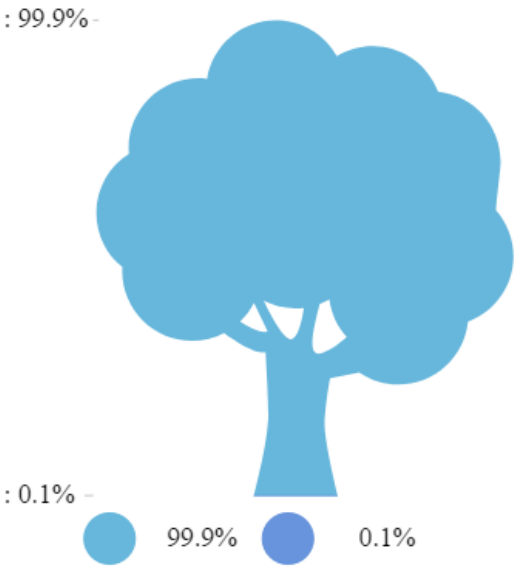

Figure 4: Carbon Footprint

Figure 5 and Figure 4 show examples of charts. Dashboard users may change the room/apartment for the data or setup the date range. The configurable values can be defined by admins in the background. Downloading the source number, deleting the widget, adding/removing/editing graphs from a chart widget, reloading data are further actions admins can perform. Last but not least, there is the widget full screen mode to mention, that enables the Dashboard to run on an integral dashboard visible e.g. for an entrance hall of a pilot site.

Widgets can contain charts, and widgets can be part of a page. Administrators can organize their own pages, and pages can be accessed through the navigation. Administrators can create pages that are visible for all users, as a default.

Figure 6 shows an example for multiple graphs in a chart. By overlaying different insights, it is possible to get deeper into the numbers.

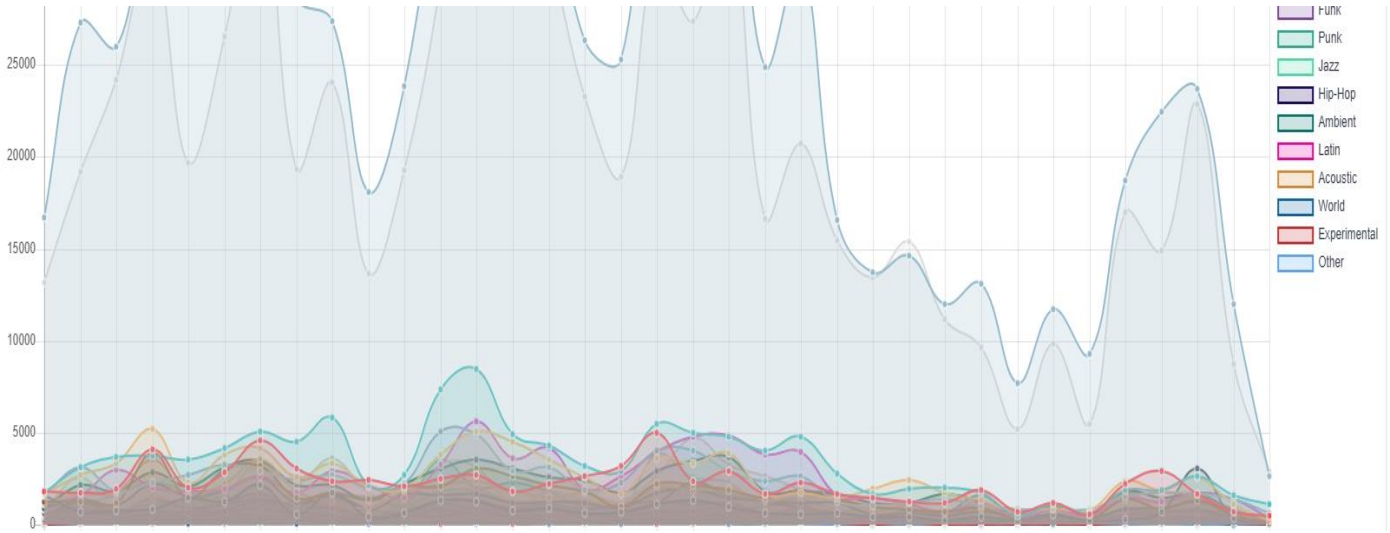

Figure 6: Other Example Statistic

Figure 7 shows the most interesting tool for administrators. The tool to configure insights. Next to general information about this insight, the source facts can be defined. The analysis script is defined here as well. It is doing the aggregation of the source facts and send the results to the insight writer. Filtering attributes can be defined, and the processing intervals and granularities. This tool steers the background processes which are necessary to generate the insights being consumed by the user.

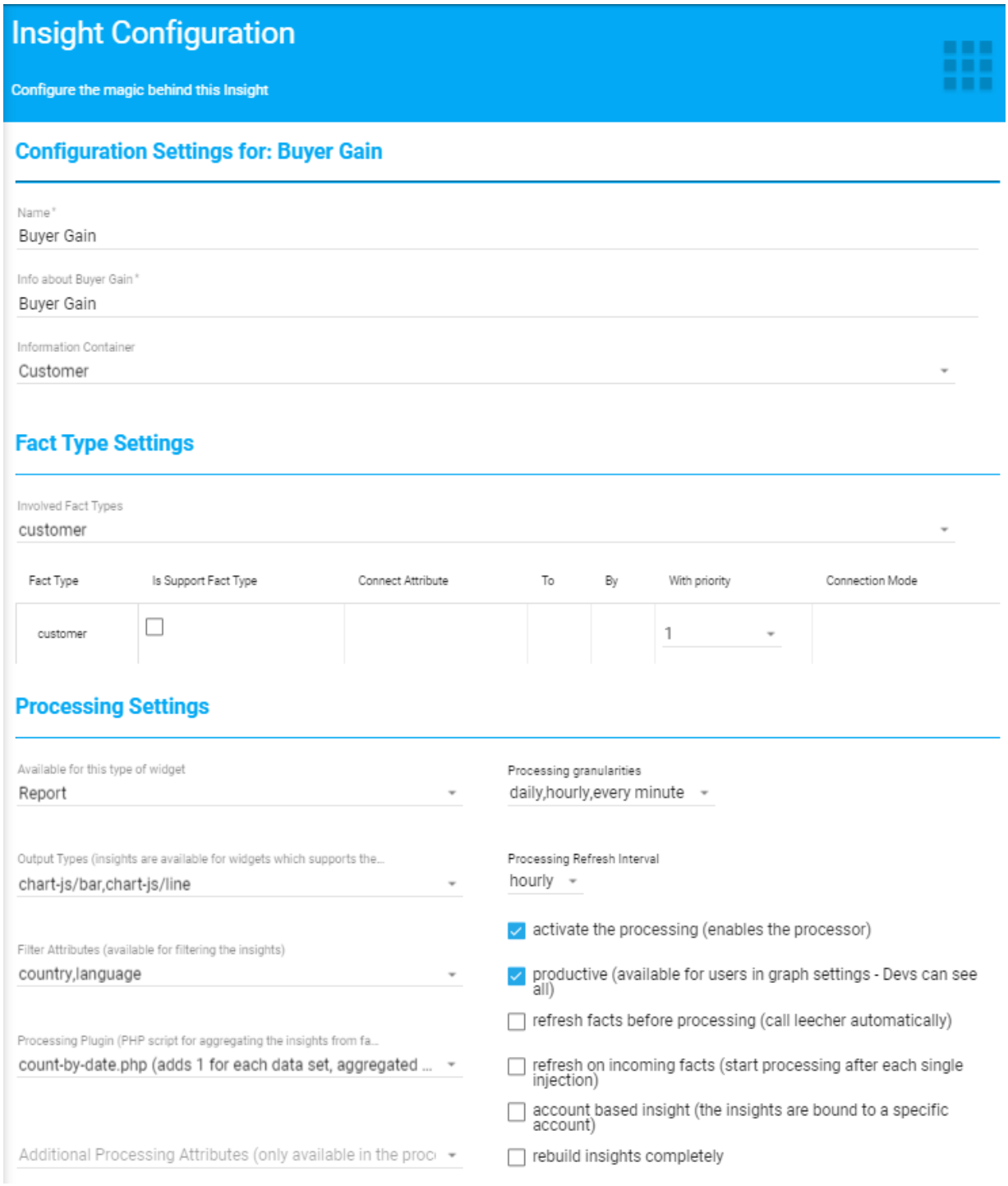

Figure 7: Administrator Tool

5 Test Plan

Testing of the Dashboard is handled with PHPUnit. This allows the testing of the REST API. It will be included into the T3.1 monitoring system, to have a continuous testing and notification system. This will enable detecting failures very quick and will shorten outages significantly.

The test cases reside within the CLI. To run the test cases the following command, which is also reachable by a server to server call over the REST API, must be triggered to enable monitoring via HTTP protocol. A specific monitoring token will be configured to fulfil this purpose.

```
Usage:

Run the unit tests, and write the results to the log database. By default all tests are executed. It is possible to run a specific, mostly for development purposes
discovery test [name-of-specific-test]
```

Listing 4: Execution of Unit Tests

The passage of all test cases indicates that the system is functioning as expected. An example of the output is shown in the following listing (the user-login did not pass the test successfully):

```
{
  "meta": {
    "project": "eTEACHER",
    "version": "v1",
    "status": true,
    "statusCode": 200,
    "statusMessage": "ok",
    "responseTimeMs": 2882.0763,
    "count": 0,
    "page": 0,
    "perPage": 0,
  },
  "data": {
    "fact-injection": {
      "status": true
    },
    "user-login": {
      "status": false
    }
    …
   }
```

Listing 5: Example of Unit Test Result

6 Limitations, Research and Further Developments

The Dashboard has to be fed with real data to unleash its full potential. Charts and all necessary configurations like the fact injection can be prepared with dummy data. But the most interesting point of the provided Dashboard is to gain deeper insights from analysing real data, like the data provided from the different pilot sites.

Possible features for future versions are a predictive analysis to extrapolate future stats from todays' data. This will be an interesting insight to eTEACHER, where it can be used to give the facility managers or residents the opportunity to extrapolate the consumption. So facility managers can use different settings and save energy in the whole building. But it is interesting as a feature for a standalone system as well. Further, a notification system, reacting on KPIs that can be defined in the insight configuration. With this it would be possible, to send messages to a facility manager, if there is a higher demand for energy and some settings have to be adjusted.

Gathering data from open data sources and gaining insights from them is also a point to mention belonging to the research character of this component.

7 Summary and Conclusion

The Dashboard component is responsible for making gamification statistics and consumption data more accessable to the final end user of the app. This is done by collecting all necessary data from other components of eTEACHER, converting them in a generalized format and providing data visualisations, which can be added to an UI. The UI is done with web-based technologies, so that it can be used on all commonly used operating systems.

All needed sub-components got provided, the integration with other components of WP3 (T3.1, T3.2, T3.4) and WP2 (BACS) are done and the different visualisation got provided as well. Further refinements consist mainly of bugfixing and will be provided in WP4. The prototype is currently running, full functional and can be accessed via the eTEACHER app.

In summary, this deliverable describes, where the component is located and all required steps to install, deploy and execute the Dashboard component. It shows the installation and configuration from the perspective of end users, administrators and developers. The description for end users is shorter, but will be explained in more detail in D4.4.