

# BIG BUILDING DATA VISUALIZATION: A VISUALIZATION TOOL FOR THE "SMART LIVING LAB"

Johan Jobin<sup>1</sup>

Supervisor: Prof. Dr. Denis Lalanne<sup>2</sup>

AUGUST 20, 2018

## DEPARTMENT OF INFORMATICS - BACHELOR PROJECT REPORT

Département d'Informatique - Departement für Informatik • Université de Fribourg -  
Universität Freiburg • Boulevard de Pérolles 90 • 1700 Fribourg • Switzerland

phone +41 (26) 300 84 65    fax +41 (26) 300 97 31    Diuf-secr-pe@unifr.ch    <http://diuf.unifr.ch>

---

<sup>1</sup>johan.jobin@unifr.ch, University of Fribourg

<sup>2</sup>denis.lalanne@unifr.ch, University of Fribourg

## Acknowledgements

- I would like to express my sincere thanks to **Prof. Dr. Denis Lalanne** for his precious advices and interesting discussions.
- A special note of thanks goes to **Dr. Julien Nembrini, Pierre Vanhulst and Raphaël Tuor** for having spent their precious time to answer my questions.
- I am also grateful to my family for encouraging me throughout the implementation and the writing of this thesis.

## Abstract

Collecting huge amount of data of building has gained importance in the last years. Indeed, with the growth of the Internet of Things, getting data using sensors has become always simpler and cheaper. All these raw data are usually stored in big databases, ready to be used for all kind of applications.

That's exactly what was done in the "Smart Living Lab", a smart building in Fribourg, in which almost 3000 sensors were installed to measure temperature, carbon dioxide, power, energy, electric potential, etc. All these data are stored in a database which is accessible through a RESTful service that can be queried to get measures of specified sensors for a given interval of time.

Hence, this work consists in the creation of a web dashboard that brings a visual meaning to the data through various visualizations, allowing the user of the system to have a better understanding of the building. Six different visualizations with different objectives are implemented: a line chart that illustrates the variation of the measures, a scatter plot dedicated to binary values like lights, a box plot to identify the quartiles of the measures, a radial plot to highlight the periodicity of one week, a calendar heat map to compare the average value of each day during one year and finally a histogram to see the distribution of the data. All these visualizations are generated on-the-fly using a big table that is a visualization itself and that acts like a menu to choose which sensors to consider in order to display the visualizations. In addition to these visualizations a search engine is also available to explore the sensors and get information about them.

With the created dashboard it is possible to notice that approximately 15% of all the sensors of the building are activated. Furthermore, multiple interesting observations can be made concerning the measure of the sensors: it is possible to see that sometimes anomalies are registered, which sensors of temperature are indoor and which are not, when it rained and when not, the differences between season, etc.

**Keywords:** Data visualization, D3.js, Big building data, RESTful API, Smart Living Lab

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Goals . . . . .	6
1.2	Context . . . . .	6
1.3	Structure of report . . . . .	7
<b>2</b>	<b>State of the art</b>	<b>8</b>
2.1	Generalities about data visualization . . . . .	8
2.2	Similar projects . . . . .	9
<b>3</b>	<b>Bbdata visualization system</b>	<b>10</b>
3.1	Architecture and technologies . . . . .	10
3.1.1	Server Side . . . . .	10
3.1.1.1	KNX standard and cloud infrastructure . . . . .	10
3.1.1.2	Bbdata RESTful interface . . . . .	11
3.1.2	Client Side . . . . .	12
3.1.2.1	Presentation of the website . . . . .	12
3.1.2.2	Technologies . . . . .	13
3.2	Visualizations and different challenges . . . . .	15
3.2.1	Main table . . . . .	15
3.2.2	Line chart . . . . .	19
3.2.3	Scatter plot . . . . .	22
3.2.4	Box plot . . . . .	23
3.2.5	Radial plot . . . . .	23
3.2.6	Calendar heat map . . . . .	26
3.2.7	Histogram . . . . .	26
3.2.8	Other visualizations . . . . .	27
3.2.8.1	Clock . . . . .	27
3.2.8.2	Spiral plot . . . . .	28
3.2.8.3	Thermometer . . . . .	29
3.2.8.4	Light bulb . . . . .	29
3.2.8.5	Pie chart . . . . .	30
<b>4</b>	<b>Exploration of data with Bbdata visualization system</b>	<b>31</b>
<b>5</b>	<b>Conclusion</b>	<b>34</b>
5.1	Synthesis . . . . .	34
5.2	Future works . . . . .	34
	<b>Appendices</b>	<b>35</b>
<b>A</b>	<b>Bbdata RESTful API documentation</b>	<b>35</b>
<b>B</b>	<b>Source code</b>	<b>44</b>

## List of Figures

1	The logo of the Smart Living Lab with the corresponding building . . . . .	6
2	Architecture of the server side of Bbdata . . . . .	10
3	Interaction between the client and the server in the RESTful API offered in Bbdata	11
4	Main page of the Bbdata visualization dashboard developed . . . . .	12
5	Comparison between different layouts for different screen sizes . . . . .	13
6	Main table that has two functions: choosing which sensors to display and comparing the measure of each sensor between them . . . . .	16
7	Data received from Bbdata RESTFul API before reorganization . . . . .	18
8	Data reorganized in suitable way to draw the main table . . . . .	18
9	Line chart with one curve for each sensor . . . . .	19
10	Basic sampling algorithm for the line chart . . . . .	20
11	Comparison between the original graph (in black) and the sample (red) . . . . .	21
12	Scatter plot showing all measures of each sensor during the desired period . . . . .	22
13	Box plot illustrating the different quartiles of the measures . . . . .	23
14	Radial plot to notice the differences between each moment of each day during a week	24
15	Color scale for the radial plot . . . . .	25
16	Calendar heat map to compare the average value of each day during one year . . . .	26
17	Histogram that shows the distribution of the measures . . . . .	26
18	Dynamic clock displaying the current time . . . . .	27
19	Spiral plot to compare the average value of each day during one year . . . . .	28
20	Thermometer dedicated to sensors of temperature . . . . .	29
21	Light bulb dedicated to sensors of lights . . . . .	29
22	Pie chart illustrating the percentage of humidity . . . . .	30
23	Proportion of activated sensors for each type of sensors . . . . .	31

**List of Tables**

1	Exhaustive list of all types of sensors of Bbdata . . . . .	7
2	Various statistics contained in the line chart . . . . .	19
3	Various information given by a box plot . . . . .	23

# 1 Introduction

## 1.1 Goals

Understanding what happens in a building is necessary to create more efficient buildings in the future. Although collecting data of all kinds in a building is a very important step in the comprehension of it, it is equally important to be able to give a visual meaning to these data in order to analyse them in a second time.

Therefore, the idea of this bachelor project consists in providing a web dashboard for the user such that he can have an overview of what happens or happened in the building of the "Smart Living Lab" in Fribourg. Indeed, the "Smart Living Lab" building almost contains 3000 sensors (see table 1), so the challenge is to create an interface that shows all of them, that is clear and that you can use to query the database to get visual information about them (whatever it measures) during a specific period of time.

In addition to this, the project aims to answer questions such as: since the position of each sensor is not clearly defined in the database, is it possible to guess the position of a sensor just by seeing visualizations of its data? Is it possible to notice some recurrent patterns in the data? Are there anomalies in the data? Or, are all sensors activated?

The main technologies that were used for achieving these goals were web technologies, such as HTML, CSS, Bootstrap for the web platform, JavaScript, JQuery and d3.js for the visualization and the interactivity with the user.

## 1.2 Context

This project takes part in the studies that are made in the "Smart Living Lab", a research and development center for the built environment of the future <sup>1</sup> in the "Blue Hall" of the "blueFACTORY" site in Fribourg. Since 2016, sensors of all kinds have been set every year in the building in order to have a maximum of information about it. All these data are stored in a database within a global system called "Bbdata", that stands for "Big Building data" and that contains millions of measures. Hence, it is a real gold mine of information that is available to be exploited in different ways.

As mentioned before, the project exploits data by using them to create visualizations to understand them better, since they are stored in a raw pair "timestamp-value" for each measure (see figure 3).



Figure 1: The logo of the Smart Living Lab with the corresponding building

---

<sup>1</sup><http://www.smartlivinglab.ch/index.php/en/>

### 1.3 Structure of report

The first main section of this report will be devoted to the "Bbdata" visualization system, which is split into two major parts: the first covers technical aspects of the project by describing technologies that were used, while the second part concerns the visualizations themselves and challenges that were taken up. The second main section is about all discoveries that were made during the exploration of the data using the visualization dashboard of the project. This section is finally followed by the last section, in which new ideas are given to create an even more complete dashboard with more functionalities.

Measure of sensor	Unit of measurement	Type of data
Lights	"on/off"	String, Bool
Occupation of rooms, aeration intensity, CO2 rate visualization, lights rate visualization	"none"	Int
Humidity	"%"	Int
Power	"W" or "kW"	Float
Energy	"Wh" or "kWh"	Float
Electric current	"A"	Float
Electric potential	"V"	Float
Temperature	"°C"	Float
Wind	"m/s"	Float
Factor of power	"cos ?"	Float
Illuminance	"lx"	Float
CO2	"ppm"	Float
Frequency	"Hz"	Float
Instantaneous flow of heat totalizer	"m3/h"	Float
Total volume of heat totalizer	"m3"	Float
Pressure	"Pa"	Float
Hours	"h"	Float
Total organic carbon for water	"ppb"	Float
Solar irradiation	"W/m2"	Float

Table 1: Exhaustive list of all types of sensors of Bbdata

## 2 State of the art

### 2.1 Generalities about data visualization

"Data visualization is the presentation of data in a pictorial or graphical format, and a data visualization tool is the software that generates this presentation. Data visualization provides users with intuitive means to interactively explore and analyse data, enabling them to effectively identify interesting patterns, infer correlations and causalities, and supports sense-making activities"<sup>2</sup>.

Although data visualization has not a comprehensive history, humans always tried to give visual significations to what they could count, measure or compute. However, this domain took much more importance with the emergence of computers and more specifically with the "Big Data". Indeed, nowadays data are continually registered and consequently means like data visualizations are necessary to make them understandable by the human perception and cognition.

This can be made by building visualizations using "pre-attentive attributes" (i.e graphical attributes that represent a feature of data and that are understood by humans without significant processing effort). This last point relies on the fact that a human can easily distinguish differences in line length, shape, orientation and color. Hence, using these attributes is necessary to create good visualizations. Furthermore, multiple principles that a good visualisation should have were described in 1983 in the book *The visual Display of Quantitative Information* of Edward Tufte who wrote: "Graphical displays should:

- Show the data
- Induce the viewer to think about the substance rather than about methodology, graphic design, the technology of graphic production or something else
- Avoid distorting what the data has to say
- Present many numbers in a small space
- Make large data sets coherent
- Encourage the eye to compare different pieces of data
- Reveal the data at several levels of detail, from a broad overview to the fine structure
- Serve a reasonably clear purpose: description, exploration, tabulation or decoration be closely integrated with the statistical and verbal descriptions of a data set."<sup>3</sup>

Finally, a system of multiple visualizations with which users can interact, should also handle some aspects such as:

- Real-time interaction: it should be possible to make a request and have a response in a few milliseconds even if there is a huge amount of data.
- On-the-fly processing: the system should create visualizations from raw data and generate visualizations dynamically from it.
- Visual scalability: it should be possible to display large datasets in a visualization.
- User assistance and personalization: users should be able to understand easily how the system works and give to them possibilities to interact with it or to personalize it.

---

<sup>2</sup><https://arxiv.org/pdf/1801.08336.pdf>

<sup>3</sup>[https://en.wikipedia.org/wiki/Data\\_visualization](https://en.wikipedia.org/wiki/Data_visualization)

## 2.2 Similar projects

The following list describes a few projects that are in the same field than this one and explains the differences between them:

- *"Design of smart home sensor visualizations for older adult"* of Thai Lea, Blaine Reederb, Jane Chungc, Hilaire Thompsonc and George Demiris. Their project consists in the creation of two visualizations using the data of sensors of a smart home. That work is different of this one because it does not provide a complete dashboard but it offers two visualizations that are adapted to old people. Furthermore, the visualizations made in this work are based on the position of each sensor (living room, bedroom, bathroom and kitch) which is unknown in the Smart Living Lab.
- *Discovering unexpected information using a building energy visualization tool* of Lange B.a, Rodriguez N.a, Puech W.a and Vasques X. They build a visualization tool in 3D, to see the energy consumption of each room. Therefore, their project is based on the position of sensors and is dedicated to only one type of sensor, unlike this project which displays all types of sensors.
- *What happened in my home ?: An End-User Development Approach for Smart Home Data Visualization* of Nico Castelli, Corinna Ogonowski, Timo Jakobi, Martin Stein, Gunnar Stevens, Volker Wulf. Their work is the one which is the closest of this work. Hence, they implemented a visualization tool that displays lots of graphs to have all information of the smart home at a glance, as this is the case in this project.

## 3 Bbdata visualization system

### 3.1 Architecture and technologies

As sensors have been installed from 2015-2016 until now, this project does not start from scratch. Indeed, all elements of the server side were already existing before the beginning of this work. Consequently, this project is an extension of an existing system in order to bring data visualization to it. Nevertheless, the comprehension of the structure of the server side is required to understand how the client side was implemented and this is the reason why it is described in this chapter.

#### 3.1.1 Server Side

##### 3.1.1.1 KNX standard and cloud infrastructure

The server side of the system is composed as followed (see figure 2): at the bottom of the architecture, there is a big part that concerns all sensors (for example push button, thermometer, etc) and actuators set in the building using the standard KNX<sup>4</sup>. This standard is devoted to domestic building automation and is responsible of collecting data and storing them in the cloud infrastructure.

This is this cloud infrastructure that is accessible through a RESTful API and that allows a programmer to make HTTP requests to communicate with the database. Therefore, thanks to this interface, multiple tools can be easily created on the client side, which is represented by "custom tools" on the figure 2. Hence, this project is a custom tool that offers a visual dashboard to the user.

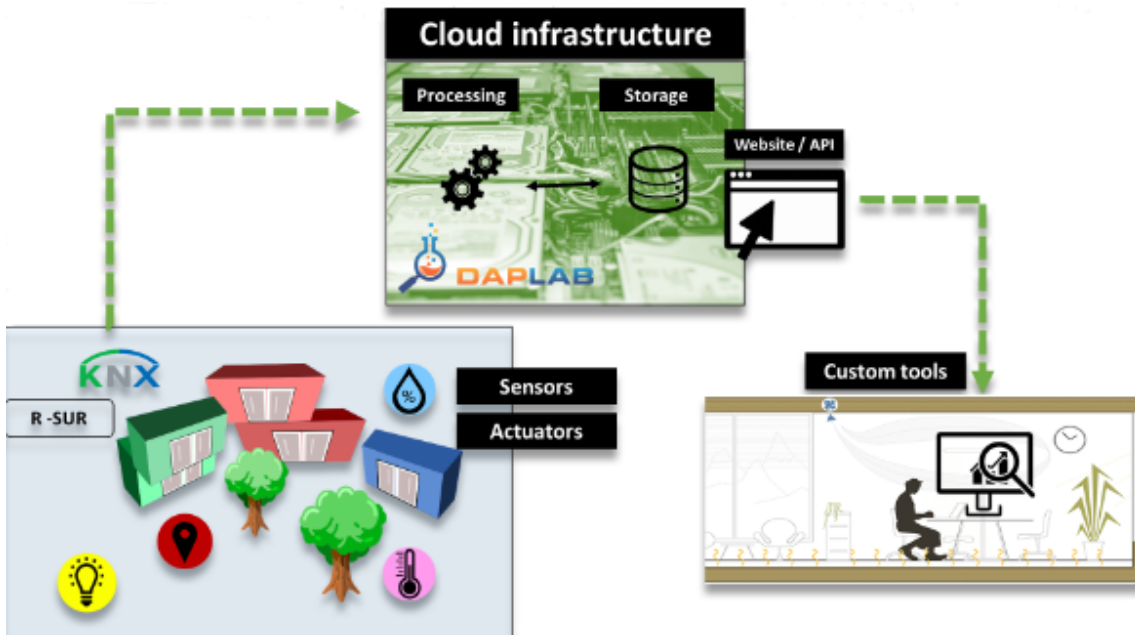


Figure 2: Architecture of the server side of Bbdata

<sup>4</sup>[https://en.wikipedia.org/wiki/KNX\\_\(standard\)](https://en.wikipedia.org/wiki/KNX_(standard))

### 3.1.1.2 Bbdata RESTful interface

The cloud infrastructure previously described is accessible through a RESTful API that allows a programmer to make HTTP requests to communicate with the database.

The database contains all objects (sensors) and users. Objects and users have each an unique id and the latter are registered in groups. In this way it is possible to group users according to their membership of this or that research group.

Regarding the available sensors, the table 1 details which kinds of measures are available in the database.

As it is shown, there are twenty-one different units of measurement used up to now. Furthermore, there are three types of data: boolean values for all sensors that measure binary states, integer values and float values. As a result, it appears necessary to process boolean data differently from integer and float data and this is why visualizations dedicated to those of the first group are not compatible with those of the second group.

As the RESTful interface is the only entry point to communicate with the database, there are many things to know and that explain some choices of implementation.

First of all, it is necessary to ask the administrator of the system to create an account in order to have an access to it. Once this is done, a login can be performed by providing the username and the password to the RESTful interface, which answered by sending back an API key and an user id to the user. This API key and this user id have to be put in the header of each HTTP request to be allowed to execute them on the database. Therefore, they are stored in a cookie to be accessible from all pages at any moment. All requests available are described in the appendix A. They give the user the possibility to interact with the database in order to log in, to get the user profile, to get measures of sensors, etc. Besides, it is possible to query the database to process data before receiving them in order to have mean values instead of independent values which can reduce the amount of data to sent and consequently the time to transmit them too.

Finally, another important information about the transmission of data is that the API relies on the JSON or the CSV format. Both formats are useful because they organize informations in a standardized way. The global running of the server and the client is summarized in the figure 3.

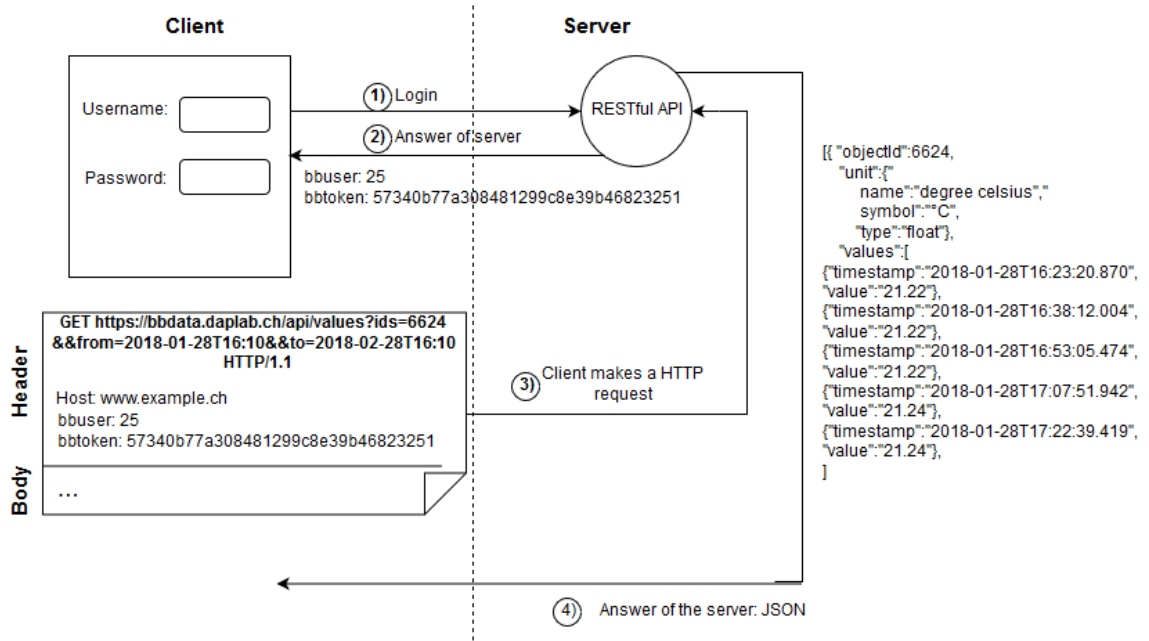


Figure 3: Interaction between the client and the server in the RESTful API offered in Bbdata

### 3.1.2 Client Side

#### 3.1.2.1 Presentation of the website

The website is composed of 4 different pages, each with a particular purpose:

- **The login page**

This page contains only one form in order to proceed to the login. When the user enters his username and his password, the page checks if the fields are not empty and if this is the case both information are sent to the server. Then, it waits until the answer of the server arrives and if the server replies by sending an user id and a token it means that everything runs smoothly. Thus the user is redirected to the dashboard. Conversely, if the server sends back an HTTP status code meaning an error occurred, a message is displayed under the form informing the user something was wrong and inviting him to try again.

- **The dashboard**

This is the core of the project. The page is divided into two parts. One contains the main table (see section 3.2.1) that builds the request according to what was chosen by the user and the other is reserved to display visualizations. There are six different kinds of visualizations, one for boolean values (on/off, true/false) and five for continuous values (float/int). So the challenge was to find a suitable layout to display visualizations in half of the screen. To resolve this, a layout involving tabs was set. In this way, the user can simply switch between all kinds of visualizations without scrolling the page by just clicking on the tab corresponding to the visualization he wants to see.

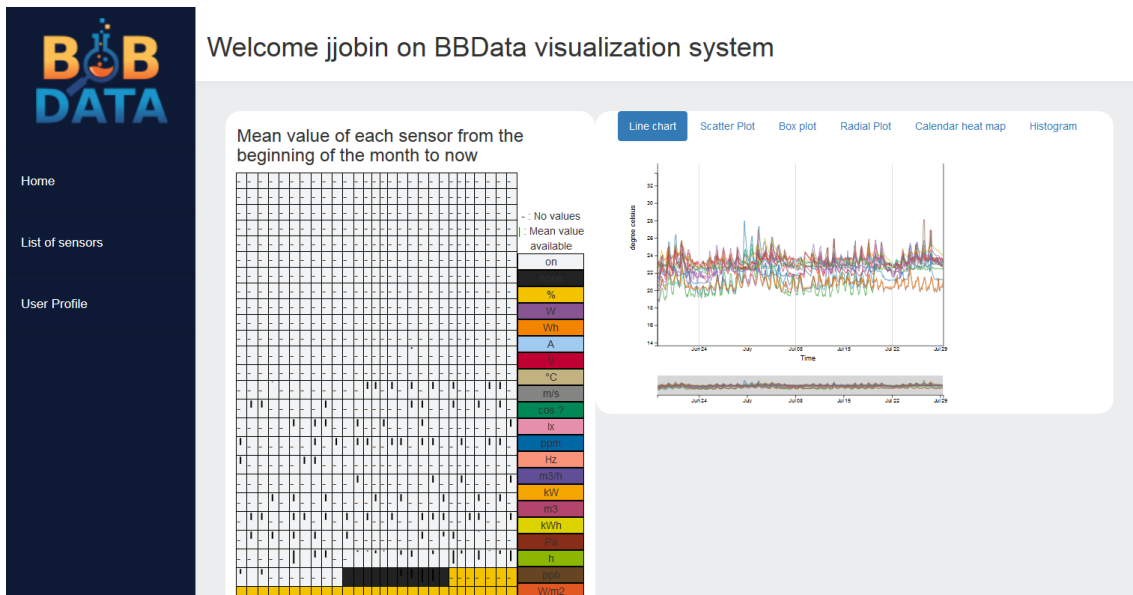


Figure 4: Main page of the Bbdata visualization dashboard developed

- **The list of sensors**

As it is noticeable in the appendix A, it is impossible to query the server to get simultaneously measures of a sensor and information about it. Therefore, it appears necessary to have a page dedicated to displaying all available information about each sensor. As a result, this page contains a big table of all objects of the database. When the page is loading, it makes a query to the server to get all objects and information, and then it generates the table thanks to the DataTable (see: section 3.1.2.2) jQuery plug-in. Furthermore, searching filters were implemented to make accurate search in the database in such a way that the page looks like a search engine of the whole database.

- **The user profile**

As Bbdata contains users in its database, this page just displays personal information of the user who is logged in. Only four information are stored for a user: his unique id, his name, his mail address and the date of the creation of the account. Moreover, as the users are organized in groups in the system, the affiliated groups are also displayed in order to know to which group a user belongs.

About the design, the website is responsive and therefore adapts itself to the screen of the user (see figure 5). There are three types of layout:

- The first one is suitable for screens bigger than a width of 768px and is built with a sidebar menu on the left side. This layout is therefore perfectly adapted for the screen of a personal computer.
- The second layout is dedicated to screens of a width between 576px and 768px and move the left menu of the first layout on top. Consequently, this layout suits for the screen of a tablet for example.
- Finally, the last layout transforms the top menu of the second layout into a drop-down menu. Therefore this layout is dedicated to mobile phones with screen of a width less than 576px.

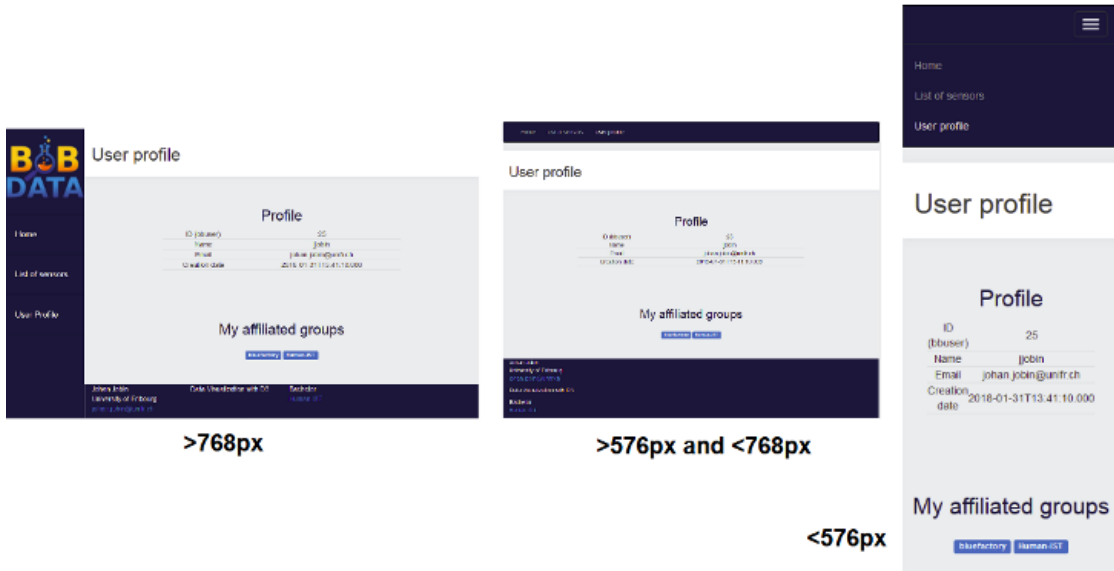


Figure 5: Comparison between different layouts for different screen sizes

### 3.1.2.2 Technologies

As the visualization system should be accessible on the web, web technologies were used to create simultaneously the web site and the system of visualizations. The following list describes shortly all technologies/frameworks/libraries that are parts of this project and explains their concrete use in the context of the project:

- **HTML5**

HTML5 is a markup language for structuring and presenting content on the World Wide Web <sup>5</sup>. It was used to create all pages of the website and all basics elements of the DOM (Document Object Model) like buttons, forms, paragraphs or texts.

<sup>5</sup><https://en.wikipedia.org/wiki/HTML5>

- **CSS3**

Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language like HTML5 <sup>6</sup>. In this work, they were used to describe the presentation, the layout and the fonts of all web pages.

- **Bootstrap**

Bootstrap is a framework intended to facilitate the creation of web applications <sup>7</sup>. It regroups CSS classes and JavaScript libraries in order to provide the programmer with multiple tools to accelerate the development. In this project, Bootstrap was used because of its system of columns: indeed bootstrap splits the screen into 12 columns and lets the programmer define how many columns a layout element should take for each device. As instance, if the screen is divided into two parts of 6 columns on a big screen, it should not behave the same way on a little screen. For that reason, bootstrap will move the right column under the other and accordingly there will only be a vertical scrolling that is perfectly adapted for a little screen like a mobile phone. This creates a responsive layout and this is for this reason that bootstrap was chosen to be part of this project. Furthermore, Bootstrap contains useful elements like navigation tabs, navigation bar, side bars and buttons that are useful components for the layout of this website.

- **JavaScript**

JavaScript is a high level, interpreted programming language that is mainly used in web pages to make them more interactive <sup>8</sup>. In this project, JavaScript was used for multiple purposes:

First, as described in the figure 3, when the server answers with the user id and the token after a login of the latter, JavaScript is used to create a cookie containing these both informations. Hence, the user id and the token, that are necessary to make a query, are available from all pages and act like global variables. Since the token has an expiration time of few hours (after which it is required to ask for a new token to make queries again), it was necessary to implement an algorithm that takes this into account. The solution adopted was to set the expiration date of the cookie equals the one of the token. In this way, it is possible to check if the current time is smaller or bigger than the expiration date of the token: if it is smaller, there is nothing to do because it means that the token is still valid and therefore queries can still be made. On the contrary, if it is bigger, it means that the token is not valid anymore and that a new token is needed. Therefore, when this case occurs, the user is automatically redirected to the login page to proceed to a new login that will generate a new token.

Secondly, JavaScript was used for different tasks like making HTTP requests using Axios library, creating date sliders, dynamically creating the table of the list of sensors and the search functions thanks to the DataTable library.

Finally, the main use of JavaScript was the creation of visualizations: the main table in the main page (see: 3.2.1) was made in pure JavaScript and all others visualizations were made with D3.js.

- **jQuery**

jQuery is a JavaScript library that provides functionalities like DOM manipulation, event handling, AJAX or animations <sup>9</sup>. It was used mainly for the event handling in this project. Indeed, in the main table, all cells can be selected, therefore they all wait for a "click" event to occur. The same behaviour is noticeable in the filters of search functions of the table of all sensors: each time a key is released, the filter is applied to all data of the table which makes a faster search function because it does not need to wait until the end of a word to start searching in the table.

---

<sup>6</sup>[https://en.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets)

<sup>7</sup><https://getbootstrap.com/>

<sup>8</sup><https://en.wikipedia.org/wiki/JavaScript>

<sup>9</sup><https://jquery.com/>

- **Axios**

Axios is a JavaScript library that offers a promise based HTTP client for the browser<sup>10</sup>. As the RESTful API is accessible through HTTP requests, this library gives the programmer an easier way to use *XMLHttpRequest* objects. The latter allows a web page to interact with servers and to retrieve data from URL without having to do a full page refresh, which is particularly suitable in the context of this work. Indeed, a user may want to choose different periods of time or different sensors without refreshing the page. Furthermore, Axios provides a mechanism based on promises which are helpful to organize better asynchronous codes. In fact, when a HTTP request is made, the program is not blocked. This situation occurs when the user of the system makes a query for particular sensors during a certain period of time for example.

- **JSON**

JSON that stands for JavaScript Object Notation is a standard file format that uses human-readable text to transmit data objects consisting of "attribute-value" pairs and array data types<sup>11</sup>. This is the format in which the RESTful API transmits data to the client. Consequently the core of the project consisted in giving a visual meaning to these raw data through visualisations made thanks to D3.js. The figure 3 shows how the measures look like in the JSON format when they are sent to the client, before being processed to create visualizations.

- **D3.js**

D3 that stands for Data-Driven Documents is a JavaScript library for producing dynamic and interactive data visualizations based on SVG, HTML5 and CSS standards in web browser<sup>12</sup>. It gives the user the possibility to draw basic shapes like squares, circles, rectangles or path to create whatever is imaginable. Moreover, the library contains lots of useful functions to bring dynamism, to create different scales, to define color scales etc. All visualizations of this work, except the main table, are built with D3.js.

- **DataTable.js**

DataTables.js is a plug-in for the jQuery JavaScript library that adds advanced features to any HTML table<sup>13</sup>. Among these advanced features are pagination of the table, instant search, columns ordering and responsive behaviour that create a dynamic table from a simple HTML table. The page that contains the list of sensors in an array was made with this library.

## 3.2 Visualizations and different challenges

### 3.2.1 Main table

The main table is illustrated on figure 6. In order to fit the entire page, the table is split to keep only the three first types of sensors (on/off, none and %) but obviously on the website, the table contains all types of sensors. This visualization has two main roles:

- First it acts like the main menu to choose what the user wants to visualize. Indeed, each cell of the table represents a particular sensor of a certain type. The type to which the sensor belongs is given by the background color of the cell. For instance, sensors that measure time have "[h]" unit and the background of cells representing them is green.

The colors chosen for each type of data were not selected randomly. In fact, as there are 21 different types of sensors, the idea was to draw the background of two neighbours with colors that have the maximum contrast between them. To achieve this goal, a study of Kenneth L. Kelly<sup>14</sup> was used. It gives a list of twenty-two different colors of maximum contrast among the ISCC-NBS centroid colors which is a system for naming colors based on a set of 12 basic color terms and a small set of adjectives modifiers. This system was established in

<sup>10</sup><https://github.com/axios/axios>

<sup>11</sup><https://en.wikipedia.org/wiki/JSON>

<sup>12</sup><https://en.wikipedia.org/wiki/D3.js>

<sup>13</sup><https://datatables.net/>

<sup>14</sup>[http://www.iscc-archive.org/pdf/PC54\\_1724\\_001.pdf](http://www.iscc-archive.org/pdf/PC54_1724_001.pdf)

## Mean value of each sensor from the beginning of the month to now

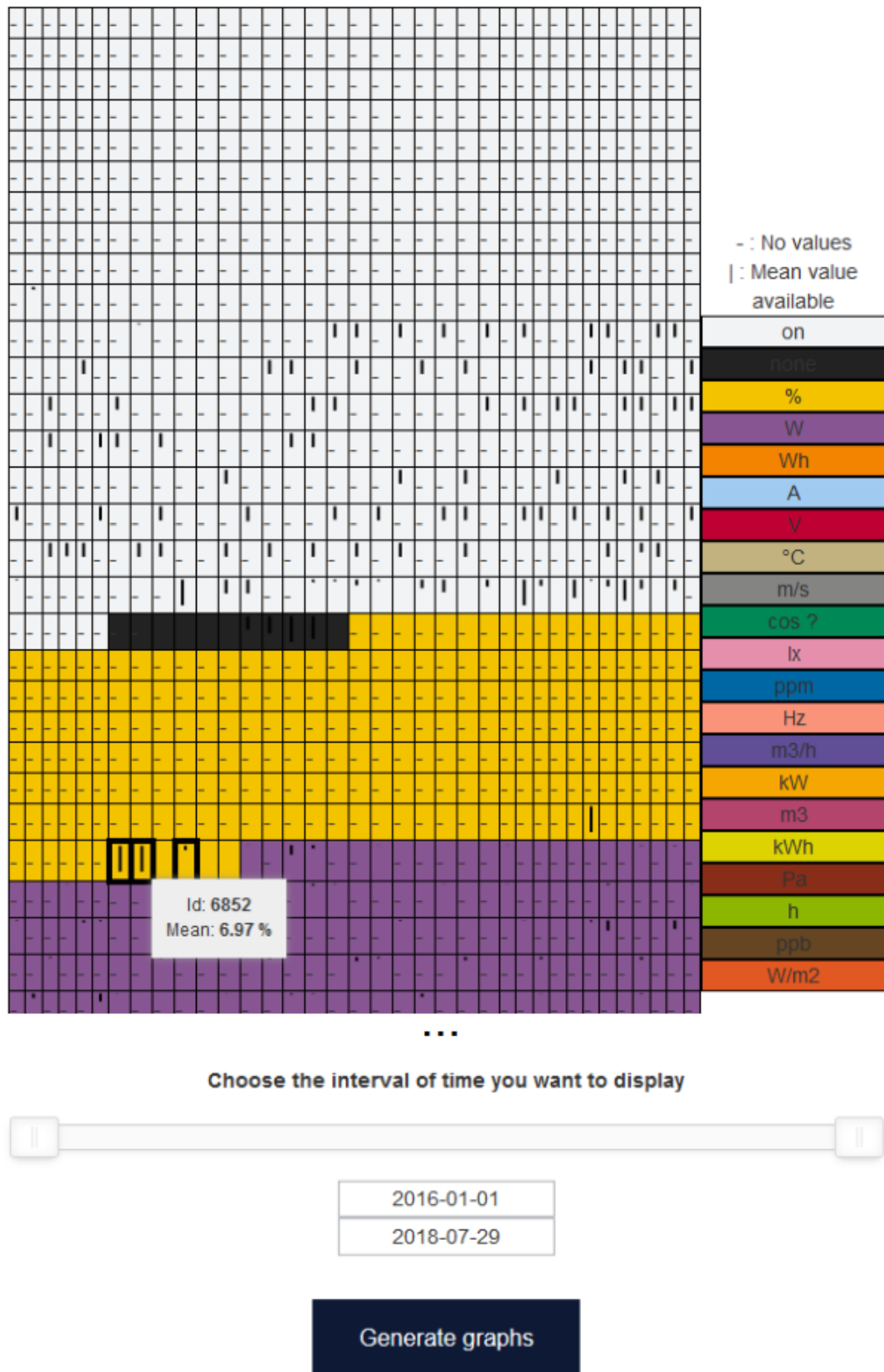


Figure 6: Main table that has two functions: choosing which sensors to display and comparing the measure of each sensor between them

the 1930's by the Inter-Society Color Council made up of delegates from various American trade organizations, and the National Bureau of Standards <sup>15</sup>. Hence, these colors were easily added to the cells of the table and so guaranteed the best visibility to differentiate units of measurement.

Furthermore, when the mouse of the user is over a cell, a text area appears and shows the id of the sensor and the exact mean value corresponding to the sensor represented by the cell. In addition to this, in each cell a line is visible: if the line is horizontal, it means that no values were registered this month for the specified sensor. In contrary, if there is a vertical line, it means that there is at least one registered measure for the specified sensor this month. Moreover, the length of vertical lines is proportional to the mean value of all measures registered this month for each sensor. In other words, for each type of sensor, the minimum and maximum mean values were computed to build a linear scale that defines the length of the line. So, a mean value of one sensor that is near the minimum value of all sensors of a specified type will cause a little line. Conversely, if the mean value of a sensor is near the maximum value of all sensors, the line will be big.

All cells of the table are selectable by clicking on them. This action will add a thicker border to them that indicates to the user that he chose this or that sensor. Besides, instead of being forced to click on each cell, it is possible to click on the cell that contains the unit of measurement to select all activated sensors of the same type (i.e those containing vertical lines). About selecting cells, something important to know is that it is impossible to select cells of different types insofar as displaying two different types of measure on the same graph would not make sense.

Once the selection of desired sensors is made, it only remains to choose the interval of time that will be used to draw the graphs. To do this, the table offers two choices. The first one is to move the date slider and the other one consists in writing the entire date under the date slider. The first method has the advantage to be fast but it is less accurate than the second method.

- Secondly, this table provides a global view of all sensors of the building at a glance and constitute at the same time an useful visualization that gives lots of information about sensors. In fact it is possible to directly see what is measured in the building (thanks to the unit of measurement), which type of sensor is majority (thanks to the background of cells), which sensors are activated (thanks to vertical lines) and it also allows to compare sensors between them (thanks to the length of the vertical line).

Before developing this visualization, the major difficulty was to find a suitable way to display the entire building with its 2761 different sensors. As described above, the answer to this was to create a table with cells corresponding to sensors and lines in them giving an overview of all mean values. Then, at the beginning of the development, the first big challenge was to reorganize the data received from the RESTful API in a more convenient way. As it is seeable in the figure 7, when the request is made, the list of 2761 sensors is transmitted with all measures registered from the beginning of the month until the current day. That is not convenient to draw the table. Therefore, the reorganization of data consisted in classifying sensors in function of their unit of measurement and computing the minimum and maximum for each sensor that has the same unit of measurement (which is useful to establish the linear scale that defines the length of the vertical line). Then, the mean value of all measures for each sensor was computed and at the end, data were organized as shown in figure 8.

After this operation, it was possible to generate the table with pure JavaScript. At the end, it only remained to insert a date slider and this was made with the "noUiSlider.js" library that offers various sliders for developers. Once the table was generated, one major problem concerned the fact that it was not responsive which caused problems in the layout. Hence, to resolve this problem, an event listener on the resizing of the window was set, which causes the redraw of the table (with new dimensions) each time the window was resized.

---

<sup>15</sup>[https://en.wikipedia.org/wiki/ISCC/NBS\\_system](https://en.wikipedia.org/wiki/ISCC/NBS_system)

```

(2761) [...]
  ▶ [0...99]
  ▶ [100...199]
  ▶ [200...299]
  ▶ [300...399]
  ▶ [400...499]
  ▶ [500...599]
  ▶ [600...699]
  ▶ [700...799]
  ▶ [800...899]
  ▼ [900...999]
    ▼ 900: {...}
      objectId: 3025
      unit: Object { name: "power", symbol: "W", type: "float" }
      values: (673) [...]
        ▼ [0...99]
          ▶ 0: Object { timestamp: "2018-07-29T19:00:00.000", last: 12.13700008392334, max: 12.619999885559082, ... }
          ▶ 1: Object { timestamp: "2018-07-29T18:00:00.000", last: 11.456000328063965, max: 12.704999923706055, ... }
          ▶ 2: Object { timestamp: "2018-07-29T17:00:00.000", last: 12.14900016784668, max: 12.53499984741211, ... }
          ▶ 3: Object { timestamp: "2018-07-29T16:00:00.000", last: 12.711000442504883, max: 12.85099983215332, ... }
          ▶ 4: Object { timestamp: "2018-07-29T15:00:00.000", last: 12.53499984741211, max: 12.777000427246094, ... }

```

Raw data from RESTful API

Figure 7: Data received from Bbdata RESTful API before reorganization

```

(21) [...]
  ▶ 0: Object { key: "on", min: 0, max: 100, ... }
  ▼ 1: {...}
    key: "none"
    max: 1.73
    min: 0
    values: (11) [...]
      ▶ 0: Object { objectId: 6685, unit: {...}, values: [] }
      ▶ 1: Object { objectId: 6686, unit: {...}, values: [] }
      ▶ 2: Object { objectId: 6805, unit: {...}, values: 0 }
      ▶ 3: Object { objectId: 6806, unit: {...}, values: 0 }
      ▶ 4: Object { objectId: 6807, unit: {...}, values: 0 }
      ▶ 5: Object { objectId: 6808, unit: {...}, values: 0 }
      ▶ 6: Object { objectId: 6831, unit: {...}, values: 0.84 }
      ▶ 7: Object { objectId: 6832, unit: {...}, values: 1.07 }
      ▶ 8: Object { objectId: 6841, unit: {...}, values: 1.73 }
      ▶ 9: Object { objectId: 6842, unit: {...}, values: 1.42 }
      ▶ 10: Object { objectId: 6845, unit: {...}, values: 0 }
      length: 11
      <prototype>: Array []
    <prototype>: Object { ... }
  ▶ 2: Object { key: "%", min: 0, max: 46.67, ... }
  ▶ 3: Object { key: "W", min: -14.02, max: 1604.78, ... }
  ▶ 4: Object { key: "Wh", min: 0, max: 0, ... }
  ▶ 5: Object { key: "A", min: 0, max: 3.52, ... }
  ▶ 6: Object { key: "V", min: 236, max: 238.87, ... }
  ▶ 7: Object { key: "°C", min: -40, max: 53.47, ... }
  ▶ 8: Object { key: "m/s", min: 0.07, max: 0.66, ... }
  ▶ 9: Object { key: "cos ?", min: 0, max: 0, ... }
  ▶ 10: Object { key: "lx", min: 22.11, max: 16417.42, ... }
  ▶ 11: Object { key: "ppm", min: 459.38, max: 2993.02, ... }
  ▶ 12: Object { key: "Hz", min: 0, max: 0, ... }
  ▶ 13: Object { key: "m3/h", min: 0, max: 0, ... }
  ▶ 14: Object { key: "kW", min: 0, max: 0, ... }
  ▶ 15: Object { key: "m3", min: 0, max: 0, ... }

```

Figure 8: Data reorganized in suitable way to draw the main table

### 3.2.2 Line chart

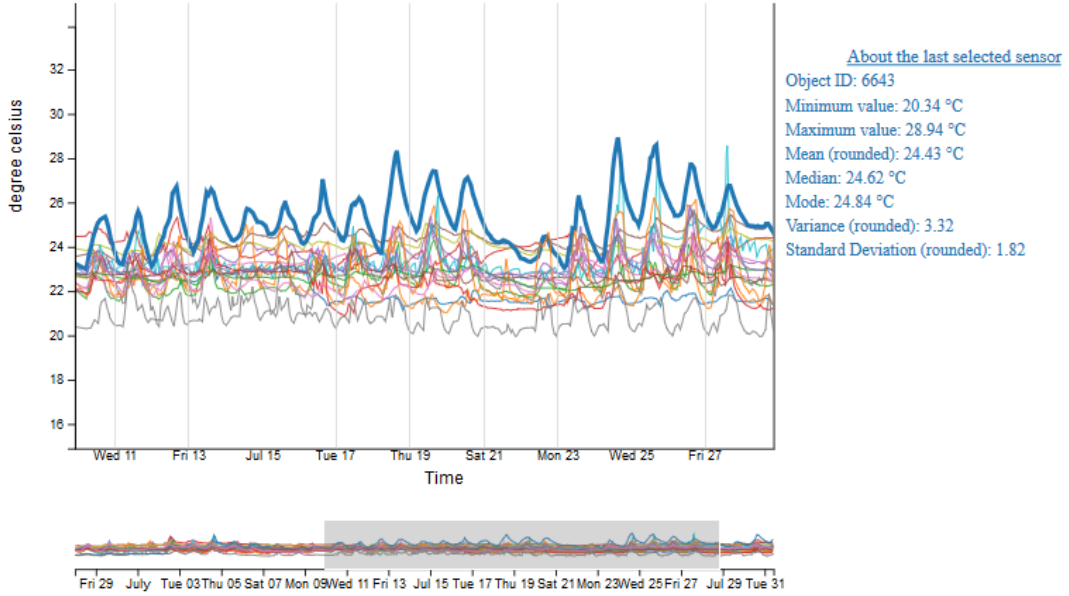


Figure 9: Line chart with one curve for each sensor

The line chart is the first graph of the dashboard and is dedicated to sensors whose units of measurement are registered as integer (int) or as float values (float). It shows the unit of measurement on the Y-axis and the time on the X-axis, that both were selected thanks to the main table. Then based on all measures of each sensor, it draws a path of a unique color for each of them in order to distinguish them. This graph offers multiple functionalities to the user:

- **Statistics**

When the user scrolls his mouse over a line, the line becomes thicker and statistics about the sensor are displayed on the right of the graph. The table 2 explains meanings of each statistical measure.

Criterion	Description
Object ID	<i>This corresponds to the unique id assigned to the sensor in Bbdata</i>
Minimum value	<i>The smallest value among all measures of selected sensors during the chosen interval of time</i>
Maximum value	<i>The biggest value among all measures of selected sensors during the chosen interval of time</i>
Mean	<i>This is the average value of all sensors</i>
Median	<i>This is the value separating the higher half from the lower half of a data sample</i>
Mode	<i>This is the value that appears most often</i>
Variance	<i>This is the expectation of the squared deviation of a random variable from its mean. Informally, it measures how far a set of values are spread out from their average value.</i>
Standard deviation	<i>This is a measure that is used to quantify the amount of variation or dispersion</i>

Table 2: Various statistics contained in the line chart

- **Zoom**

When the user scrolls the mouse wheel, he can see in more detail the area he zoomed on. This is particularly useful when lots of lines overlap. The zoom causes the brush to resize because the visible area always becomes smaller. Moreover, when the user zooms, the scale of the X-axis (time) adapts itself to the new interval of time which allows to have different scales of time according to the zoom.

- **The brush movement/resizing**

The brush is the grey rectangle below the main part of the graph. This rectangle is movable horizontally and defines the visible zone. Besides, the brush can also be resized which causes the zoom described in the previous paragraph.

The implementation of the graph did not cause major problems even if the development of the brush movement and the zoom were not that easy to make. However, the first big difficulty encountered by the graph was at the level of performances. Indeed, as all data of all sensors are processed in the client side (i.e in the web browser), it was neither possible to draw the graph quickly nor to use the brush or the zoom when the amount of data was big. This can be explained because the code adds DOM elements for each measure and if the selected period of time (or the number of sensors) is big enough, the number of measures can be huge. Consequently, the only solution was to reduce the number of measures by sampling data. There are many ways to sample data, in the context of this visualization two different sampling algorithms were implemented and the most efficient one was integrated into the final version of the code:

- **A basic sampling algorithm**

This algorithm relies on the fact that keeping consecutive measures with the same value is storing useless data. In the case of the line chart, if consecutive measures have the same value, it only keeps the first measure and the last one of the series and link them with a line. The main advantage of this algorithm is that it is very simple to be implemented but it still has a big default: it is impossible to define the number of measures to keep. For instance if the data set does not contain consecutive measures that have the same value, applying the algorithm will have no effect. The functioning of the algorithm is illustrated in the figure 10.

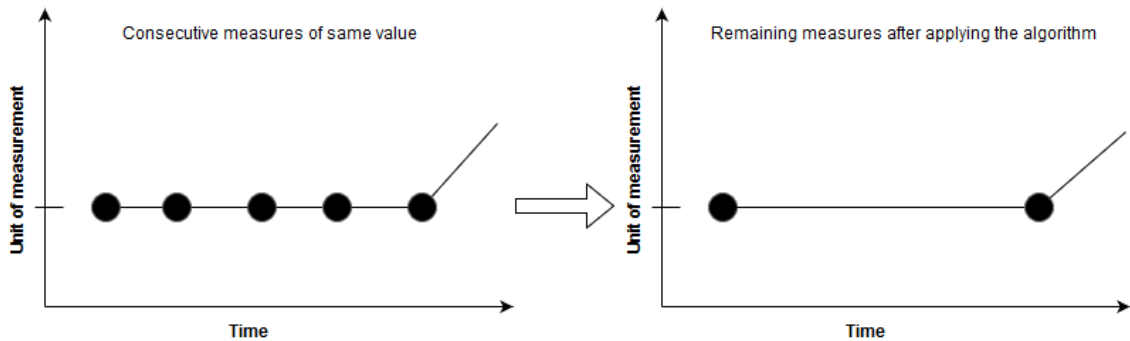


Figure 10: Basic sampling algorithm for the line chart

- **Mode-median algorithm**

This algorithm has a more complex operating mode because it takes as input the maximal number of measures to keep after its execution, which is called the "threshold". Although any number can be defined to be the threshold, it seems obvious that the threshold should not be bigger than the number of pixels of the width of the area containing the graph, otherwise two measures would be displayed on the same pixel. The best number of measures to keep is therefore equals to the width of the area containing the graph.

Once this number is given as input to the algorithm, it can start working as follows: first of all, it distributes all measures between a certain number of buckets of equal sizes (i.e it splits measures into a certain number of sets of equal sizes). This number of buckets corresponds to the defined threshold and the size of each bucket is obtained by dividing the number of

measures by the number of buckets. Then, the goal is to keep only one measure per bucket by following rules that define which measure to keep.

The first rule is to check if the bucket contains a maximum or a minimum, and if so, keeping this measure. Thanks to this rule, it is guaranteed that extreme values (minima and maxima) of measures are kept in the sample of data. The second rule is applied if and only if no measure was chosen after the first rule. It consists in computing the mode (i.e the number of occurrences) of each measure.

After this operation two situations can happen: either there exists a measure with the highest mode, either not. Consequently, if there exists one, the procedure is to keep the measure with the highest mode and if there does not exist any, the median of the bucket is found and chosen to be the value to keep in the sample. These whole conditions are applied to every buckets and at the end, it returns the sampling of data of the desired size given as input.

The figure 11 shows the results of this algorithm according to the amount of measures the user wants to keep. The black line is the original graph of all elements and the red one represents the line after the use of the algorithm on the same measures. As it is noticeable, in both case, the approximation of the algorithm contains the minimum and maximum, which confirms what was said before. In addition, it is also seeable that the approximation of the original graph looks good, even if only 10 elements are kept among the initial 2000 elements. Hence, choosing to keep the number of pixels of the width as the threshold is good enough to approximate accurately the original graph.

The last consequence of this algorithm concerns the execution time necessary to draw the entire graph: as the measures are fewer, the execution time to draw the graph is reduced too. In fact, it was the modification of the DOM that took lots of time, not the algorithm itself.

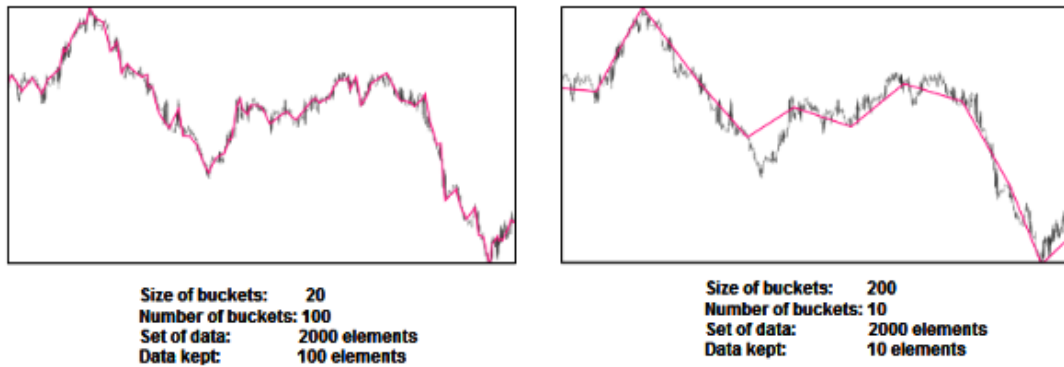


Figure 11: Comparison between the original graph (in black) and the sample (red)

### 3.2.3 Scatter plot

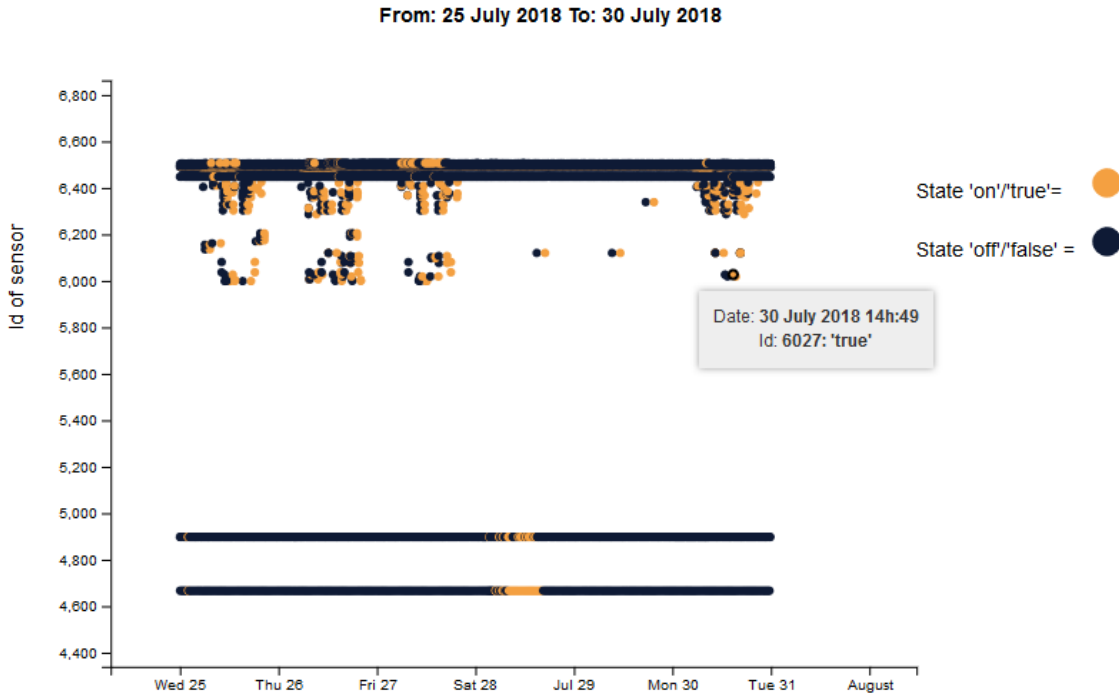


Figure 12: Scatter plot showing all measures of each sensor during the desired period

The scatter plot is the only graph dedicated to binary values like "on/off" or "true/false" for lights, operating status or rainfall etc... The Y-axis defines objects ids and the X-axis is the scale for the time. This graph relies on two circles of two different colors: an orange circle stands for a "on/true" state and a blue circle for an "off/false" state. In this way, each line represents all states for a specific sensor during the chosen interval of time. Each circle can give further information like the exact date of the measure and the object id when the mouse is over it. This graph offers a functionality to make it more interactive:

- **Zoom**

When the user scrolls the mouse wheel, he can see in more detail the area he zoomed on. This is particularly useful when lots of circles overlap. This zoom is not a semantic zoom but a graphical zoom. Indeed, when the user zooms, the scale of the X-axis (time) adapt itself to the new interval of time which allows to have different scales of time according to the zoom, but the same circles as before the zoom are displayed.

Like the line chart, the drawing of the graph took too much time in first versions of the code and sampling was once again implemented. This algorithm of sampling is extremely easy and here is how it works:

First, it takes as input the number of measures desired to keep, called the threshold. This number equals the width of the graph. Then, it computes an interval, called  $n$ , by dividing the total number of measures by the threshold. Finally, it simply loops through all measures and keeps only one measure every  $n$  measures.

### 3.2.4 Box plot



Figure 13: Box plot illustrating the different quartiles of the measures

The box plot is a graph devoted to float and integer values. It has a lot of advantages; it allows quick graphical examination of data sets, it is useful to compare various data sets, it does not take a lot of space and it illustrates different quartiles of data.<sup>16</sup> In the box plot, there is only a scale for Y-axis and elements are positioned according to it. Indeed, as the figure 13 illustrates, the box plot gives multiple information about data in function of this scale. All these informations are summarized in the table 3.

Element	Description
Yellow line	This correspond to the median of measures
Red line	This corresponds to the mean of measures
Q1 and Q3	Q1 stands for the first quartile: the median of the lower half Q3 stands for the third quartile: the median of the upper half
Whiskers	They show extrema: the whisker of the bottom indicates the minimum and the top one the maximum

Table 3: Various information given by a box plot

When the graph is loaded, these information are not directly visible but when the mouse is over an element, it appears in a little rectangle.

This graph, unlike the previous ones, works only for one sensor at a time. Accordingly, if multiple sensors are chosen in the main table, one box plot is created per sensor and they are simply displayed one above the other.

Concerning the development of this graph, no major difficulty was encountered except that on the internet, there was no example of single box plot, so this visualization was made without having a concrete example.

### 3.2.5 Radial plot

The radial plot is a graph that can be used to display sensors registering float or integer values. It highlights the periodicity of measures for each sensor during at most one week. Actually, when the interval of time selected by the user with the main table is bigger than one week, a message appears to tell him that this graph is not suitable for this request. The reason of this limitation comes from the structure of the graph: each circle represents a unique day, so if the interval of time is too big, there is not enough space to draw a circle for each day. The way to get around this limitation of one week, is to use the calendar heat map (see: 3.2.6).

<sup>16</sup>[https://en.wikipedia.org/wiki/Box\\_plot](https://en.wikipedia.org/wiki/Box_plot)

Mean scale: ☐ Median scale: ☒

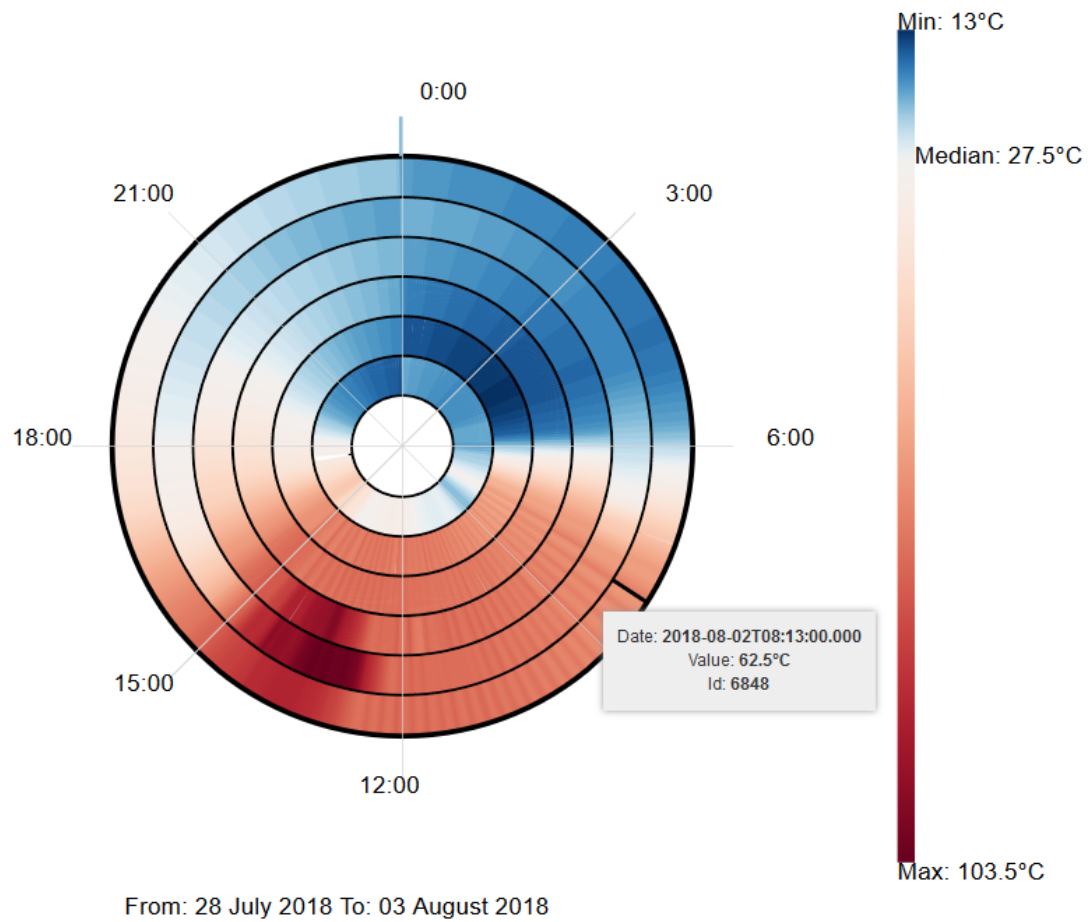


Figure 14: Radial plot to notice the differences between each moment of each day during a week

Moreover about the structure of the graph itself: each day (represented by a circle) is split into twenty-four hours with midnight at the top and midday at the bottom. In this way, it is possible to compare measures of sensors between each day of the week. Furthermore, rectangular bars are perpendicularly drawn all around circles with two determining criteria for each of them:

- **The color**

The color that uses a gradient going from blue to red is an indicator which represents the value of the registered measures. Indeed, when data are loaded, minimum, maximum, mean/-median values are computed to build a linear scale. This scale takes as input the measure and outputs a result that goes from one to zero with one equals the minimum, zero the maximum and one half the mean or the median. Then, after passing a measure in this scale, an interpolation that maps minimum to blue, maximum to red and mean/median to white defines the color of each bar. This procedure is described schematically in the figure 15. In this way, the color white, which corresponds to the color of transition between the blue and the red has a concrete meaning: if the user selects the median scale (thanks to the radio button), the white will be mapped to it and in the other case, if he chooses the mean scale, it will be mapped to the latter. Hence, this gives two different possibilities to display the same set of measures.

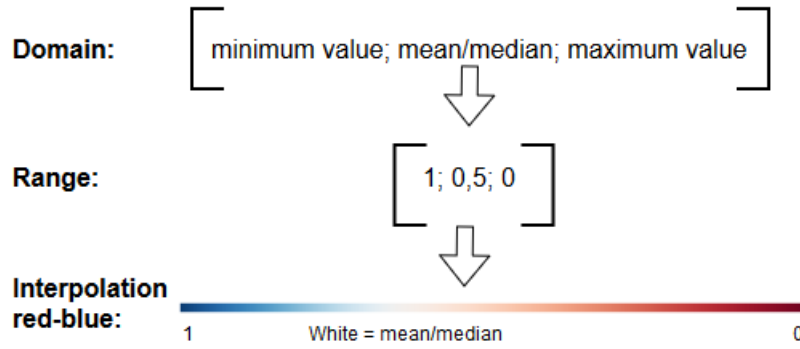


Figure 15: Color scale for the radial plot

- **The height**

In the first versions, only one sensor at a time could be displayed using this graph. The solution to this problem is to define different heights for each sensor. For instance, if the user chooses only one sensor to display, all rectangular bars will have the same and the biggest possible height. Now, if the user chooses  $n$  multiples sensors, the graph will generate  $n$  different heights of rectangular bars. Thus the graph can compare all measures of various sensors for each day of a chosen week.

Moreover, these rectangular bars can give more accurate information if the mouse is over them. Indeed, it gives the exact time of the measure, the rounded value of it and the object id to which the rectangular bar refers to.

Concerning the implementation, the first major difficulty concerned the drawing of circles. Indeed, in the library of d3.js, it is possible to draw svg circles by defining coordinates of the center and the radius but in this way, it is impossible to get coordinates of each point of the line. In other words, it is impossible to query the system to have coordinates of a point which is at a distance of  $n$  pixels from the beginning of the line of the circle. There exists a function "getPointAtLength(int length)" that gives coordinates corresponding to the point at a distance "length" from the beginning of the line but it can only be called on paths, not on circles. Therefore, it was necessary to make a generator of circles using svg paths instead of simply creating an svg circle. The aim of having this function was to be able to draw each rectangular bar at the right place on the line.

Then, an other challenge of this visualization was to offer to the user multiple scales as explained before. Once this was implemented a user could click on one of the two radio buttons and it simply drew again the entire graph with the new chosen scale.

In addition to these two first challenges was the reorganization of data. Indeed, when the RESTful API sends data, they are classified by sensors (see figure 7) but in the context of this graph, it is necessary to classify them by days. To achieve this goal, *d3.nest()* functionality was used to take the raw data and turn them into a nested structure.

Finally, like in the previous visualizations an algorithm of sampling was implemented to draw the graph faster. This algorithm takes the perimeter of a circle in pixel as input and defines it as the maximum threshold of measures to be displayed. Then, it checks if the number of measures given as input is bigger than the threshold. If no, all measures are simply drawn. On the other hand, if no, it computes an interval, called  $n$ , by dividing the total number of measures by the threshold and it simply loops through all measure and keeps only one measures every  $n$  measures.

### 3.2.6 Calendar heat map

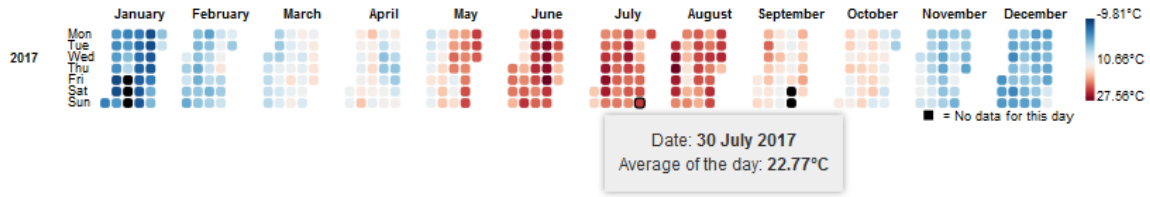


Figure 16: Calendar heat map to compare the average value of each day during one year

The calendar heat map is a visualisation dedicated to float or integer values. It shows the mean value of each day of the year using a little square to represent a day and a color to express the value of the mean. This graph is in a way an extension of the radial plot: indeed the radial plot allows to see the evolution of measures during hours and days for one week but does not display bigger intervals of time, which is the case here.

The calendar heat map is composed of two axes: the X-axis indicates each month of the year and the Y-axis each day of the week. Consequently, these two axes define a grid where each cell is used to represent a day. This grid system provides multiple advantages such as: all vertical columns are real weeks of year, months are clearly visible because they are split each other, the entire layout is like a calendar and the interval of time chosen by the user has no limit of years.

About each day, depending of the mean value measured, the cell is filled with a color from blue to red. The scale used for the color is exactly the same as the one described in figure 15 and the graph gives the possibility to choose the meaning of the white color as in the radial plot (i.e white stands for the mean or the median). What's more, all little squares can give the exact date of the day and the rounded average value of the day when the mouse is over one of them.

Concerning the implementation, the main challenge was to reorganize data in a such a way to have at first level the years, then the months and finally the days. Once it was done, it remained to complete data because very often there were days without measures and instead of not having a measure, it was necessary to create an empty measure (that facilitates the drawing then).

### 3.2.7 Histogram

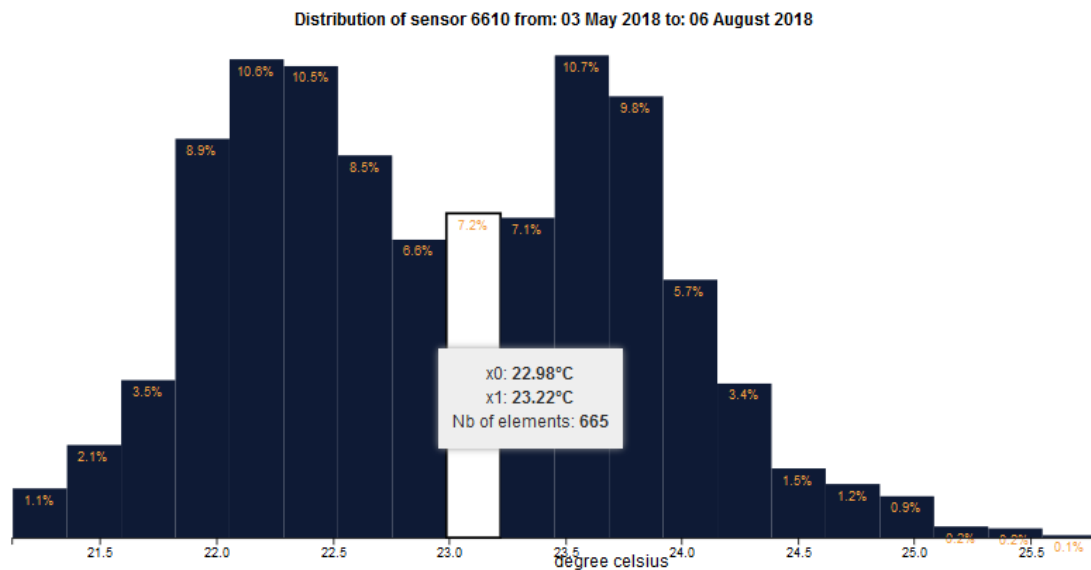


Figure 17: Histogram that shows the distribution of the measures

The histogram is a graph dedicated to sensors that registers float or integer values. It creates various intervals and shows the number of measures that are in each of them. Therefore, this graph relies on two axes: the X-axis represents the values of the measures and the Y-axis defines the percentage of measures in an interval. As it is noticeable on the figure 17, the horizontal scale goes from the minimum value to the maximum value and the vertical scale from zero to the biggest percentage. In this way, the repartition of the distribution is clearly observable. Moreover, if the mouse is on an interval, the lower and upper range are shown, as well as the number of elements inside the interval.

The biggest challenge concerning this graph was to split the data into intervals. In the first versions, this operation was done manually but that was before having discovered that the D3.js library offers functions to achieve this goal. Consequently, it was enough to put all measures into a set, to call the function on it and the entire process was made thanks to the D3.js function. Once the data was organized in intervals, it only remained to add a vertical bar on every interval with a height that is proportional to the percentage of this interval.

Like the box plot, this graph suits only for one sensor at a time, therefore if multiple sensors are chosen, multiple graphs are just displayed one above the other.

### 3.2.8 Other visualizations

All visualizations that are part of this section are not present in the final version and were created either to try some technical aspects of D3.js or either to provide functionalities that were no longer needed in the final version.

#### 3.2.8.1 Clock

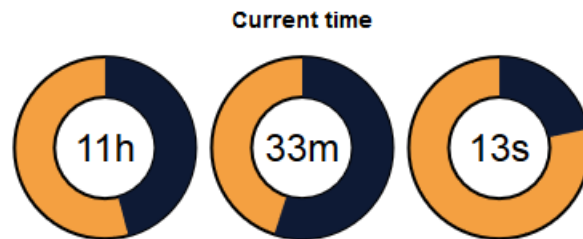


Figure 18: Dynamic clock displaying the current time

This visualization is particular because it does not visualize data of Bbdata but the current time. It is composed of three parts, one for hours, one for minutes and the last for seconds. The blue circle of each arc represents the proportion of time elapsed from the beginning of the current hour/minute/second. It is made to try multiple functionalities offered by D3.js and initially it was planned to be part of the design of the website but as it was useless, it has not been kept in the final version. Here are some features of D3.js that were tried in the graph:

- **The drawing of arcs**

D3.js provides functions to draw arcs by giving the start angle, the end angle, the inner radius and the outer radius as input of the function. In this graph, a first arc of 360° is drawn in orange. Then a second arc is drawn over the previous one, whose color is blue and whose end angle is given by the updater (see below).

- **The updater**

The updater is a function that is called every second to bring dynamism to the graph. It handles the update of the end angle of the blue arcs. This update is made using an "easing" provided by `d3.ease()` and an interpolation (see below).

- **The easing**

As the updater gives to the blue arc the new angle each second, it would be possible to simply draw the blue arc again with the new angle but this would give a less fluent result. Therefore, using "easing" offered by D3.js can generate animations in the graph, which makes it more fluent. The "easing" that is used in this visualization is `d3.easeElastic()` which gives the impression that the end angle bounces multiple times before finding its final position.

### 3.2.8.2 Spiral plot

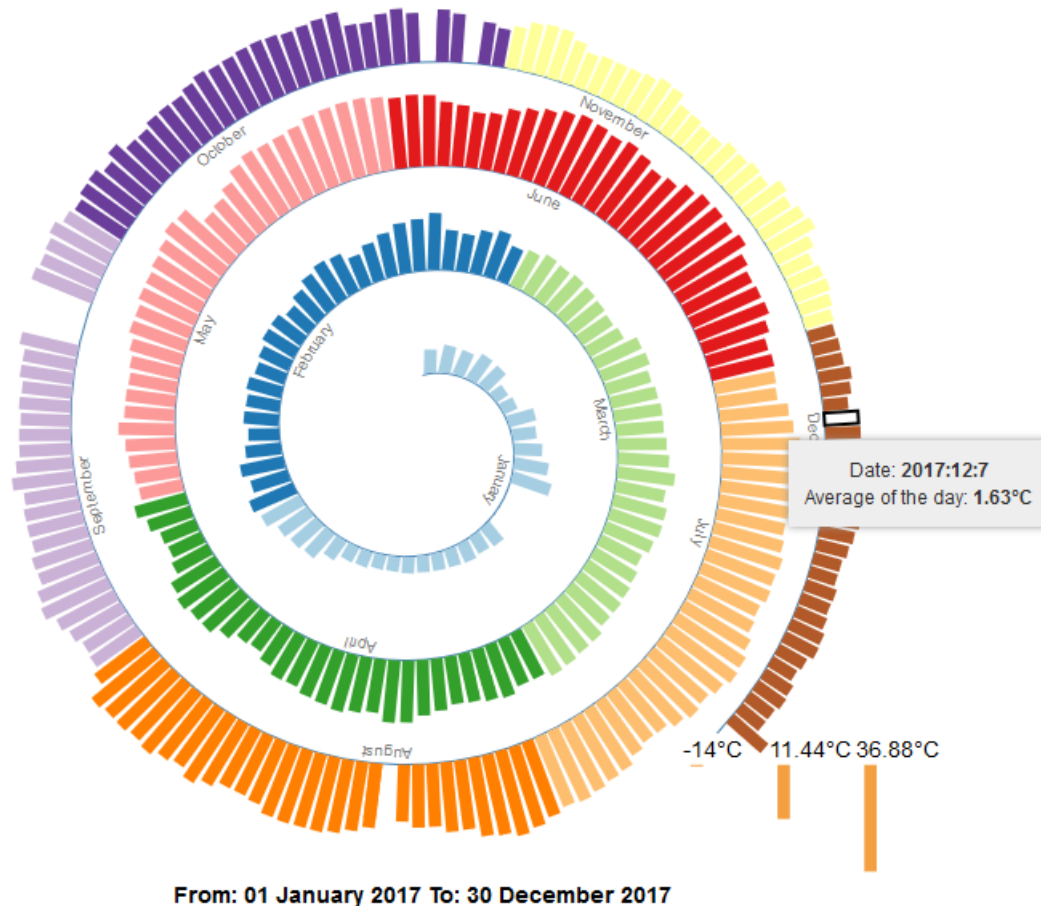


Figure 19: Spiral plot to compare the average value of each day during one year

This graph, which is devoted to be used with float or integer values, is not included in the final version of the project because it shows exactly the same informations as the calendar heat map but with less ease of reading. Indeed, the spiral plot shows the mean value of measures of each day during an entire year. The spiral represents the entire year and each bar corresponds to a day. A bar has two main features:

- **The height**

A linear scale whose domain goes from the minimum value to the maximum value and whose range goes from one pixel to  $((radius / numberOfSpirals) - 20)$  is used to give the height of each bar. In this way, it is possible to have an overview of the measured values of the year. The problem, which caused the graph not to be included in the final version, is that it is very difficult to compare bars that are not neighbours and that does not have the same

orientation. Therefore, although it is a nice designed visualization, it does not fully satisfy the main goal of a visualization.

- **The color**

The color is simply used to split the month in order to differentiate them. Furthermore, under each month is written the name of the month to specify to which color the month is associated with.

Concerning the implementation, like in the calendar heat map, the biggest part was to organize data according to years, months and days. Once this was done, creating the spiral as a path and appending the bars on it was the second biggest difficulty. To achieve these two goals, the same procedure than the one used in the radial plot was implemented.

### 3.2.8.3 Thermometer

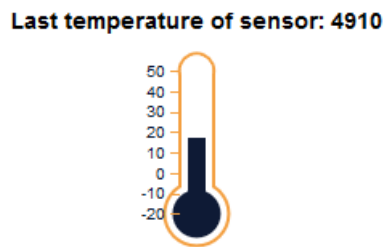


Figure 20: Thermometer dedicated to sensors of temperature

This graph is especially dedicated to sensors that measure temperature. The basic idea consisted in using it in the big table of all sensors with the goal to display the last measure of each sensor of temperature. Unfortunately, it took too much time to get the last measure of all sensors of temperature and to draw a thermometer for each of them. As a result, this graph is not present in the final version but is ready to be used for possible future works.

About the graph itself, it displays the last value measured of a sensor of temperature thanks to a mercury column that is drawn from the thermometer bulb until the measured value. This drawing is made with a linear scale that defines the height of the mercury column. Besides, if the user scrolls his mouse on the mercury column, he can have the exact date of the last measure and its value.

Although the graph looks very basic, the implementation of it was more complex than it looks because of the number of elements that constitute it. As a matter of fact, it is made from one white circle for the rounded top tube, one other in white for the main bulb, one other in blue for the main bulb, one white rectangle for the tube and one blue rectangle for the mercury. All these svg shapes are overlapped to produce the thermometer as shown in figure 20.

### 3.2.8.4 Light bulb



Figure 21: Light bulb dedicated to sensors of lights

This visualization is the smallest one. Like the thermometer, it was also created with the goal to be put into the big table of all sensors but this idea was forgotten for the same reason as the one which causes the thermometer not to be in the final version. This graph is dedicated to binary values like lights for instance. It simply changes its color according to the last state of a sensor that registers binary values: if it is on, the light bulb is orange and if not, it becomes blue.

### 3.2.8.5 Pie chart

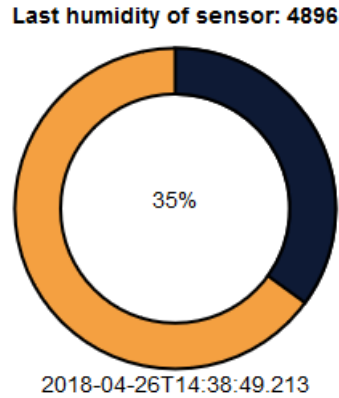


Figure 22: Pie chart illustrating the percentage of humidity

This little graph is the third one initially intended to be integrated to the big table of all sensors but which was finally not. It is implemented to visualize the last percentage of humidity of a sensor. Indeed, the total arc corresponds to hundred percent and the blue one corresponds to the last registered percentage of humidity for one sensor. Under the arc is written the exact date of the measure that is visualized. About the impletentation, the arc was created using the same functions than the ones that were described for the visualization of the clock.

## 4 Exploration of data with Bbdata visualization system

Now that all visualizations were described, this chapter will enumerate in the form of bullet points lots of discoveries that can be made using the Bbdata visualization system developed in this project. In order to demonstrate the utility of each graph, the discoveries are ordered by the graph with which they were made. Furthermore, as there are too many sensors to focus on each of them, the discoveries are general remarks and concern only the following types of sensors: "on/off", "W" and "°C". The other types of sensors are ignored due to their low number of activated sensors.

### Main Table

- As of today (08.08.2018), there are 375 sensors that are activated (i.e that have registered at least one measure from the beginning of the month until now) among the 2761 sensors installed.
- As the caption of the graph shows it, there are 21 types of sensor.
- The different types of sensors do not have the same number of sensors: for example, there are much more sensors of temperature than sensors of speed of wind.
- The figure 23 shows the proportion of activated sensors for each type of sensor.

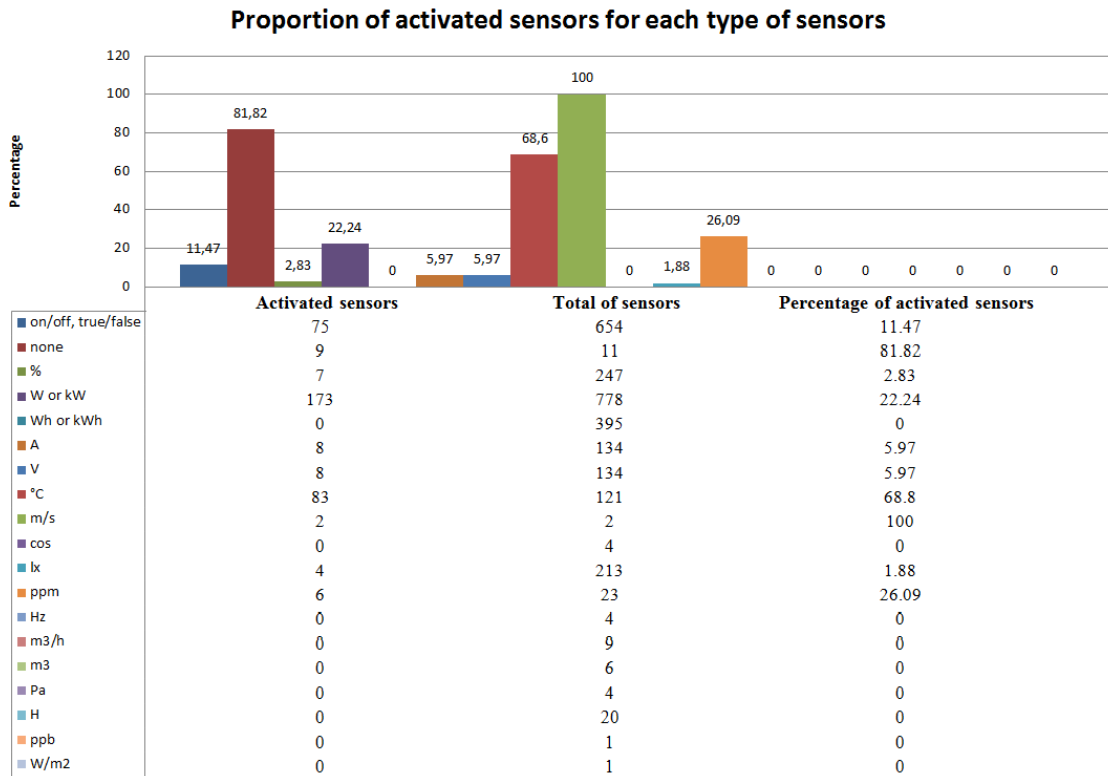


Figure 23: Proportion of activated sensors for each type of sensors

## Line chart

- The major part of sensors started to register measures in July 2017.
- Sensors sometimes register incoherent measures. For example, in the case of measures of carbon dioxide, sometimes there are extremely big values registered or in the case of sensors of temperatures, measures reached -60 °C whereas their mean value is approximately 25 °C. Another example concerns the sensors of watt, where one sensor registers negative value.
- Concerning the sensors of °C, it appears clearly that there exists three categories of sensors:
  - Sensors that are indoor: they are recognizable due to their low variance. Indeed, as the temperature does not vary a lot indoor, the curve is almost constant. Furthermore, it is between twenty and twenty-five, which corresponds to the temperature of a room.
  - Sensors that are outdoor: they are recognizable because of their bell curve. This shape of curve is caused by the fact that during the night the temperature becomes lower and during the day it becomes warmer. Accordingly these sensors have a higher variance than the ones of the first group.
  - Sensors that are near machines: they have a shape that repeats regularly all the time. For instance, the sensors near fans have this characteristic: when the temperature reaches a certain level, a fan is turned on and makes the temperature to drop until another level, at which it is turned off.
- Concerning the sensors of power, it is not possible to sort them as it is done for the ones of temperature. Nevertheless, there is one sensor that has bigger values than all other and that is the one which measures the power of the server of "La Halle Bleue."

## Scatter plot

- Not all sensors register measures at the same pace.
- The first measure registered of this type of sensors is the 3 January 2016. Then, it is clearly noticeable that the whole system started to register at the end of November 2016.
- As the scatter plot is used to visualize binary values, it is possible to see:
  - The occupation of a room thanks to the lights: when the light is "on" it means that someone is in the room. Unfortunately all these sensors are not activated.
  - The rainfall: when it rains the state "on" is registered, so the rainfall of the entire year can be easily displayed and the differences between seasons appear. It rains more in autumn and winter than in spring and summer.
  - The opening of windows: it is possible to know which window was open and at which moment. Unfortunately no sensors of this type are activated. It would have been interesting to compare sensors of temperature with these latter because there might be a relation between them.

## Box plot

- Concerning sensors of temperature:
  - Indoor sensors are represented with flat box plot. What's more, the median and the mean are very similar.
  - Outdoor sensors and sensors near machines are represented with thicker box plots. Furthermore, the distinction between the mean and the median is bigger.
- Concerning sensors of power:
  - Sensors that always have registered a power of 0 W are represented by only one line: therefore it is easy to which sensors measured an use of power.

### **Radial plot**

- This graph allows to see the evolution of the measures during days and weeks.
- Concerning sensors of temperature:
  - Indoor and outdoor sensors are warmer the day and colder the night. Besides, the difference between the day and the night is bigger in the case of outdoor sensors.
- Concerning sensors of power
  - Sensors that measure the power of a light often register measures from 6:00 AM to 18:00, what corresponds to offices hours. The sensor, previously described, that measures the power of the main server has the same behaviour. It can maybe be explained by the fact that there are more requests during the day than the evening or the night.

### **Calendar heat map**

- This graph allows to see the evolution of the measures during months and years.
- Concerning sensors of temperature:
  - Seasons are clearly visible for all sensors of temperature: it is warmer in summer than in winter.
- Concerning sensors of power
  - The sensor of the main server consumes more power in summer than in winter: maybe it is caused by the fact that it needs to be cooled, which causes an extra consumption of power.

### **Histogram**

- This graph gives the shape of the distribution. As the shape of each sensor at each moment is different, it is not possible to give global remarks.

## 5 Conclusion

### 5.1 Synthesis

At the beginning of this project, nothing visual was proposed to the user, only raw data stored in a database and at the end, an entire website dedicated to the visualization is available. Between these two moments multiple steps were passed: learning JavaScript and D3, trying to connect to the RESTful API, getting aware of various JavaScript libraries, looking for ideas to have an overview of all sensors at the same time etc... As a result, it appears that the first goal of this project, which was to implement a functional dashboard to create visualizations, was reached.

Indeed, the created website offers great possibilities to the user to see all sensors, their measures and their evolutions through the time. Furthermore, the developed dashboard goes beyond the visualization by providing a little search engine for the user to get informations about each sensor. As a matter of fact, this project provides a solid foundation in the exploration of data of the Smart Living Lab. Moreover, thanks to the visualizations, lots of discoveries were made and this allowed to make reasonable suppositions concerning the sensors: for instance, even without knowing the position of the sensors of temperature it was possible to guess more or less which of them was outdoors and which was not.

Finally, this projects shows clearly the importance of the visualization in the context of the big data: the initial data had no utility for humans but after having been visualized, they got an understandable meaning for humans cognition and perception.

### 5.2 Future works

The following list contains a few possible extensions

- More visualizations: the dashboard offers 6 visualizations but it would be possible to add more by simply adding a new tab for it.
- Notifications system: this dashboard allows only to see what happens or happened in the building but it would also be interesting to develop a system of notifications that checks if measures of a specified sensor exceed the desired values.
- Possibility to act on the building: for instance, if a notification said that the room is too warm, it would be great to be able to open the window through the dashboard.
- Possibility to manage users: currently only the administrator can add new people to the database, it would be easier to be able to create an account on his own.

# Appendices

## A Bbdata RESTful API documentation

The Bbdata API provides some endpoints that let a programmer manage, view and consult objects and values (see next page). The following code illustrates how to proceed to the login that is described in the figure 3 at the step number one. Furthermore, it shows how to store the user id and the API token in a cookie, which is necessary to make future queries to the database.

```
1
2 var username = document.getElementById("pseudo").value;
3 var password = document.getElementById("mdp").value;
4 var data = JSON.stringify({username: username,password: password});
5
6 axios.post('https://bbdata.daplab.ch/api/login',data,
7 {headers: {'Content-Type':'application/json'}})
8   .then(function (response){
9     var d = new Date();
10    d.setTime(d.getTime()+(1000*60*60));
11    var expires="expires="+d.toUTCString();
12    //cookie containing API key
13    document.cookie ="key="+response.data.secret+";"+expires+"; path=/;";
14    //cookie containing the UserId
15    document.cookie ="userId="+response.data.userId+";"+expires+"; path=/;";
16    document.cookie ="expiration="+d.getTime()+" "+expires+";path=/";
17    document.location="dashboard.html"
18  })
19  .catch(function (error) {
20    var element = document.getElementById("error").firstChild;
21    element.parentNode.removeChild(element);
22    var wrongEntry =document.createElement("h1")
23    .appendChild(document.createTextNode("Wrong password or username"));
24    document.getElementById("error").appendChild(wrongEntry);
25    document.getElementById("formConn").reset();
26  });
```

Once the client owns the API key and the user id, he can start to interact with the RESTful API. The following pages contain an exhaustive list of commands that can be executed, explain them and finally illustrate the answer of the server.

Endpoint	Method	Description and query	JSON received
https://bbdata.daplab.ch/api/info	GET	<b>Get the time of the server, the API version and the build time</b>	{ "server.time": "2017-05-01T12:52:46.711", "build.date": "2017-05-01T12:34", "version": "0.4" }
https://bbdata.daplab.ch/api/units	GET/POST	<b>Get all available units</b> Add units to the system: <i>https://bbdata.daplab.ch/api/units?name="myUnit"&amp;&amp;symbol="mySymbol"&amp;&amp;type="myType"</i> <b>Create a new unit</b> <i>https://bbdata.daplab.ch/api/units?name="myName"&amp;&amp;symbol="mySymbol"&amp;&amp;type="myType"</i>	{ "type": "float", "name": "volts", "symbol": "V" }
https://bbdata.daplab.ch/api/login	POST	<b>Login with username and password.</b> <b>This will create a read/write apikey valid for 2 hours:</b> <i>https://bbdata.daplab.ch/api/login?username="myName"&amp;&amp;password="myPassword"</i>	{ "id": 1, "readOnly": true, "secret": "3f802697606211e6b882504165db36c1", "userId": 1 }
https://bbdata.daplab.ch/api/logout	POST	<b>Logout by revoking the apikey previously created with the /login method</b>	No JSON (HTTP status code 200)
https://bbdata.daplab.ch/api/objectGroups	GET/PUT	<b>Get the list of object groups accessible (in read-only) by the user:</b> <i>https://bbdata.daplab.ch/api/objectGroups?writeObjects="false"&amp;&amp;writable="false"</i> <b>Create a new object group</b> <i>https://bbdata.daplab.ch/api/objectGroups?description="myDescr"&amp;&amp;name="myName"&amp;&amp;unitSymbol="symbol"&amp;&amp;owner="owner"</i>	{ "id": 2, "name": "temp", "owner": { "id": 1, "name": "admin" } No JSON (HTTP status code 200)
https://bbdata.daplab.ch/api/objectGroups/{groupId}	GET/POST/DELETE	<b>Get an object group with all its objects:</b> <i>https://bbdata.daplab.ch/api/objectGroups?id="myNumber"</i>	See next page

https://bbdata.daplab.ch/api/objectGroups/{groupId}/objects		<p><b>Edit/Delete the name and/or the description of the object group.</b>  <a href="https://bbdata.daplab.ch/api/objectGroups?id={myNumber}">https://bbdata.daplab.ch/api/objectGroups?id={myNumber}</a></p>	<pre>{   "id":2,   "name":"temp",   "owner":{     "id":1,     "name":"admin" },   "objects":[     {       "creationdate":"2016-08-09T11:30:50",       "id":3,       "name":"tmp box 1",       "owner":{         "id":1,         "name":"admin" },       "tags":[       ],       "unit":{         "type":"float",         "name":"degree celsius",         "symbol":"°C" } },     ]</pre>	<pre>{   "id":2,   "name":"temp",   "owner":{     "id":1,     "name":"admin" },   "objects":{     "creationdate":"2016-08-09T11:30:50",     "id":3,     "name":"tmp box 1",     "owner":{       "id":1,       "name":"admin" },     "tags":{     },     "unit":{       "type":"float",       "name":"degree celsius",       "symbol":"°C" } },     }</pre>
https://bbdata.daplab.ch/api/objectGroups/{groupId}/objects	GET/PUT/ DELETE	<p><b>Get the list of objects part of the group:</b>  <a href="https://bbdata.daplab.ch/api/objectGroups/myNumber/objects">https://bbdata.daplab.ch/api/objectGroups/myNumber/objects</a></p> <p><b>Add/Delete an object to the group:</b>  <a href="https://bbdata.daplab.ch/api/objects?groupId={myNewObject}">https://bbdata.daplab.ch/api/objects?groupId={myNewObject}</a></p>	Same as above but only objects are sent	No JSON (HTTP status code 200)
https://bbdata.daplab.ch/api/objectGroups/{groupId}/permissions	GET/PUT/ DELETE	<p><b>Get the list of user group which have access to the group</b>  <a href="https://bbdata.daplab.ch/api/objectGroups/myNumber/permissions">https://bbdata.daplab.ch/api/objectGroups/myNumber/permissions</a></p> <p><b>Grant permissions on the group to a user group</b>  <i>Same as previous command</i></p> <p><b>Revoke a permission</b>  <i>Same as previous command</i></p>	<pre>[   {     "id":1,     "name":"Human-IST"   } ]</pre>	<p>No JSON (HTTP status code 200)</p> <p>No JSON (HTTP status code 200)</p>

https://bbdata.daplab.ch/api/objects	GET/PUT	<p><b>Get the list of accessible objects.</b>  <a &amp;&amp;page="numberOfPage" &amp;&amp;perpage="myNumber" &amp;&amp;search="searchedName" &amp;&amp;tags="myTag" href="https://bbdata.daplab.ch/api/objects?writable=TrueOrFalse">https://bbdata.daplab.ch/api/objects?writable=TrueOrFalse"&amp;&amp;tags="myTag"&amp;&amp;search="searchedName"&amp;&amp;page="numberOfPage"&amp;&amp;perPage="myNumber"</a></p> <p><b>Create a new object:</b>  <a &amp;&amp;owner="idOfUser" &amp;&amp;unitsymbol="myUnit" href="https://bbdata.daplab.ch/api/objects?description=" mydescr"&amp;&amp;name="myName">https://bbdata.daplab.ch/api/objects?description="myDescr"&amp;&amp;name="myName"&amp;&amp;unitSymbol="myUnit"&amp;&amp;owner="idOfUser"</a></p> <p><b>Get an object</b>  <a href="https://bbdata.daplab.ch/api/objects/" mynumber"="">https://bbdata.daplab.ch/api/objects/"myNumber"</a></p> <p><b>Edit the name and/or the description of the object.</b>  <i>Same as previous command</i></p> <p><b>Disable this object. All associated tokens will be removed</b>  <a disable"="" href="https://bbdata.daplab.ch/api/objects/" mynumber"="">https://bbdata.daplab.ch/api/objects/"myNumber"/disable</a></p> <p><b>Enable this object</b>  <a enable"="" href="https://bbdata.daplab.ch/api/objects/" mynumber"="">https://bbdata.daplab.ch/api/objects/"myNumber"/enable</a></p> <p><b>Get the list of tokens for the object</b>  <a href="https://bbdata.daplab.ch/api/objects/" mynumber"="" tokens"="">https://bbdata.daplab.ch/api/objects/"myNumber"/tokens</a></p> <p><b>Generate a new secured token</b>  <i>Same as previous command</i></p> <p><b>Edit the token's description</b>  <a href="https://bbdata.daplab.ch/api/objects/" mynumber"="" tokens?description="myDescr">https://bbdata.daplab.ch/api/objects/"myNumber"/tokens?description="myDescr"</a></p> <p><b>Revoke a token</b>  <a href="https://bbdata.daplab.ch/api/objects/" mynumber"="" tokens?tokenid="numberToken">https://bbdata.daplab.ch/api/objects/"myNumber"/tokens?tokenId="numberToken"</a></p>	<pre>{   "creationdate": "2016-08-09T11:30:50",   "description": "",   "id": 1,   "name": "volts box 1",   "owner": {     "id": 1,     "name": "admin"   },   "tags": [     {       "tag": "hello"     }   ],   "unit": {     "type": "float",     "name": "volts",     "symbol": "V"   } }</pre> <p>No JSON (HTTP status code 200)  Same as above</p>
https://bbdata.daplab.ch/api/object/ s/objectId	GET/POST		
https://bbdata.daplab.ch/api/object/ s/objectId/disable	POST		3x No JSON (HTTP status code 200)
https://bbdata.daplab.ch/api/object/ s/objectId/enable	POST		
https://bbdata.daplab.ch/api/object/ s/objectId/tokens	GET/PUT/ POST/DELETE		<pre>[   {     "id": 1,     "objectId": 1,     "token": "ddb3da3c935723ee2ef69..."   } ]</pre> <p>3x No JSON (HTTP status code 200)</p>

https://bbdata.daplab.ch/api/objects/objectId/tags	PUT/DELETE	<p><b>Add tags to the object</b>  <a href="https://bbdata.daplab.ch/api/objects/" mynumber"="" tags?tags="myTags">https://bbdata.daplab.ch/api/objects/"myNumber"/tags?tags="myTags"</a></p> <p><b>Remove tags</b>  <i>Same as previous command</i></p> <p><b>Get all comments attached to this object</b>  <a comments"="" href="https://bbdata.daplab.ch/api/objects/" mynumber"="">https://bbdata.daplab.ch/api/objects/"myNumber"/comments</a></p>	<p>No JSON (HTTP Status code 200)</p> <p>No JSON (HTTP Status code 200)</p> <pre>{   "from": "2016-08-24T08:07:49",   "to": "2016-08-24T09:07:49",   "comment": "Here is the comment",   "id": 7,   "objectId": 3 }</pre>
https://bbdata.daplab.ch/api/object/objectId/comments	GET		<pre>{   "objectId": 1,   "unit": {     "symbol": "V",     "name": "volts",     "type": "float"   },   "values": [     {       "value": 99.16204,       "timestamp": "2016-08-15T12:37:37"     },     {       "value": 26.357155,       "timestamp": "2016-08-15T12:39:37"     }   ] }</pre> <p>Same as above but keep only the last measure</p>
https://bbdata.daplab.ch/api/values/latest	GET		<p><b>Get measures</b>  <a &amp;&amp;to="2018-02-28T16:10" href="https://bbdata.daplab.ch/api/values?ids=" myids"&amp;&amp;from="year-month-dayThour:minute">https://bbdata.daplab.ch/api/values?ids="myIds"&amp;&amp;from="year-month-dayThour:minute"&amp;&amp;to="2018-02-28T16:10"</a></p>
https://bbdata.daplab.ch/api/values/latest	GET		<p><b>Get the latest measure before a given date</b>  <a href="https://bbdata.daplab.ch/api/values/latest?id=" myid"&amp;&amp;before="date">https://bbdata.daplab.ch/api/values/latest?id="myId"&amp;&amp;before="date"</a></p>
https://bbdata.daplab.ch/api/values/quarters	GET		<p><b>Get aggregated measures (15 minutes granularity)</b>  <a href="https://bbdata.daplab.ch/api/values/quarters?ids=" myid"&amp;&amp;withcomments='false&amp;&amp;prettyprint=false&amp;&amp;headers=false&amp;&amp;from="2016-12-31T15:59&amp;&amp;to="2017-12-31T16:59:59"'>https://bbdata.daplab.ch/api/values/quarters?ids="myId"&amp;&amp;withComments=false&amp;&amp;prettyprint=false&amp;&amp;headers=false&amp;&amp;from="2016-12-31T15:59&amp;&amp;to="2017-12-31T16:59:59"</a></p>

https://bbdata.daplab.ch/api/objects/objectId/tags	PUT/DELETE	<p><b>Add tags to the object</b>  <a href="https://bbdata.daplab.ch/api/objects/" mynumber"="" tags?tags="myTags">https://bbdata.daplab.ch/api/objects/"myNumber"/tags?tags="myTags"</a></p> <p><b>Remove tags</b>  <i>Same as previous command</i></p> <p><b>Get all comments attached to this object</b>  <a comments"="" href="https://bbdata.daplab.ch/api/objects/" mynumber"="">https://bbdata.daplab.ch/api/objects/"myNumber"/comments</a></p>	<p>No JSON (HTTP Status code 200)</p> <p>No JSON (HTTP Status code 200)</p> <pre>{   "from": "2016-08-24T08:07:49",   "to": "2016-08-24T09:07:49",   "comment": "Here is the comment",   "id": 7,   "objectId": 3 }</pre>
https://bbdata.daplab.ch/api/object/objectId/comments	GET		<pre>{   "objectId": 1,   "unit": {     "symbol": "V",     "name": "volts",     "type": "float"   },   "values": [     {       "value": 99.16204,       "timestamp": "2016-08-15T12:37:37"     },     {       "value": 26.357155,       "timestamp": "2016-08-15T12:39:37"     }   ] }</pre> <p>Same as above but keep only the last measure</p>
https://bbdata.daplab.ch/api/values/latest	GET		<p><b>Get measures</b>  <a &amp;&amp;to="2018-02-28T16:10" href="https://bbdata.daplab.ch/api/values?ids=" myids"&amp;&amp;from="year-month-dayThour:minute">https://bbdata.daplab.ch/api/values?ids="myIds"&amp;&amp;from="year-month-dayThour:minute"&amp;&amp;to="2018-02-28T16:10"</a></p>
https://bbdata.daplab.ch/api/values/latest	GET		<p><b>Get the latest measure before a given date</b>  <a href="https://bbdata.daplab.ch/api/values/latest?id s=" myid"&amp;&amp;before="date">https://bbdata.daplab.ch/api/values/latest?id s="myId"&amp;&amp;before="date"</a></p>
https://bbdata.daplab.ch/api/values/quarters	GET		<p><b>Get aggregated measures (15 minutes granularity)</b>  <a href="https://bbdata.daplab.ch/api/values/quarters?ids=" myid"&amp;&amp;withcomments='false&amp;&amp;prettyprint=false&amp;&amp;headers=false&amp;&amp;from="2016-12-31T15:59&amp;&amp;to="2017-12-31T16:59:59"'>https://bbdata.daplab.ch/api/values/quarters?ids="myId"&amp;&amp;withComments=false&amp;&amp;prettyprint=false&amp;&amp;headers=false&amp;&amp;from="2016-12-31T15:59&amp;&amp;to="2017-12-31T16:59:59"</a></p>

https://bbdata.daplab.ch/api/values/quarters	GET	<p><b>Get aggregated measures (60 minutes granularity)</b>  <a href="https://bbdata.daplab.ch/api/values/hours?id_s='myId'&amp;withComments=false&amp;prettyprint=false&amp;headers=false&amp;from=2016-12-31T15:59&amp;to=2017-12-31T16:59:59">https://bbdata.daplab.ch/api/values/hours?id_s='myId'&amp;withComments=false&amp;prettyprint=false&amp;headers=false&amp;from=2016-12-31T15:59&amp;to=2017-12-31T16:59:59</a></p>	Same as previous JSON, except that the interval of time is one hour
https://bbdata.daplab.ch/api/userGroups	GET/PUT / DELETE	<p><b>Get the list of all user groups</b>  <a href="https://bbdata.daplab.ch/api/userGroups?admin=false">https://bbdata.daplab.ch/api/userGroups?admin=false</a></p> <p><b>Create a new user group</b>  <a href="https://bbdata.daplab.ch/api/userGroups?name='nameOfGroup'">https://bbdata.daplab.ch/api/userGroups?name='nameOfGroup'</a></p> <p><b>Delete the user group</b>  <a href="https://bbdata.daplab.ch/api/userGroups?id='idGroup'">https://bbdata.daplab.ch/api/userGroups?id='idGroup'</a></p> <p><b>Get a user group, including the list of its users.</b>  <a href="https://bbdata.daplab.ch/api/userGroups/numberOfGroup">https://bbdata.daplab.ch/api/userGroups/numberOfGroup</a></p>	<pre>[   {     "id":1,     "name":"bbfactory-admins"   } ]</pre> <p>No JSON (HTTP status code 200)</p> <p>No JSON (HTTP status code 200)</p> <p>Same as first command</p> <pre>{   "id":1,   "name":"admin",   "users":[     {       "isAdmin":true,       "creationdate":"2016-08-09T11:30:51",       "email":"lucy@lala.com",       "id":1,       "name":"lucy"     }   ] }</pre> <p>Same as users field of previous JSON</p> <p>No JSON (HTTP status code 200)</p> <p>No JSON (HTTP status code 200)</p>
https://bbdata.daplab.ch/api/userGroups/{userGroupId}	GET		
https://bbdata.daplab.ch/api/userGroups/{userGroupId}/users	GET/PUT / DELETE		

<code>https://bbdata.daplab.ch/api/userGroups/userGroupId/users/new?admin=false&amp;name="nameOfUser"&amp;email="email"&amp;password="password"</code>	PUT	<b>create a new user and add it to the group</b> <code>https://bbdata.daplab.ch/api/userGroups/userGroupId/users/new?admin=false&amp;name="nameOfUser"&amp;email="email"&amp;password="password"</code>	No JSON (HTTP status code 200)
<code>https://bbdata.daplab.ch/api/users</code>	GET	<b>Get the list of all users registered</b> <code>https://bbdata.daplab.ch/api/users</code>	<pre>[ {   "creationdate": "2016-08-09T11:30:51",   "email": "lucy@lala.com",   "id": 1,   "name": "lucy" } ]</pre>
<code>https://bbdata.daplab.ch/api/me</code>	GET	<b>Get my profile</b> <code>https://bbdata.daplab.ch/api/me</code>	Same as previous JSON
<code>https://bbdata.daplab.ch/api/me/groups</code>	GET	<b>Get the list of groups I am part of</b> <code>https://bbdata.daplab.ch/api/me/groups?admin=false</code>	<pre>[ {   "id": 1,   "name": "Human-IST" }, {   "id": 1,   "readOnly": false,   "secret": "6ae4c46dl606211eb8825...",   "userId": 1 } ]</pre>
<code>https://bbdata.daplab.ch/api/me/apikeys</code>	GET/PUT/ POST/DELETE	<b>Get the list of apikeys</b> <code>https://bbdata.daplab.ch/api/me/apikeys</code> <b>Generate a new apikey</b> <code>https://bbdata.daplab.ch/api/me/apikeys?write=true&amp;expire="1d-1h-1m-1s"</code> <b>Edit an apikey</b> <code>https://bbdata.daplab.ch/api/me/apikeys?id="id"&amp;readOnly=true&amp;secret="key"&amp;userId="userId"</code> <b>Revoke an apikey</b> <code>https://bbdata.daplab.ch/api/me/apikeys?id="id"</code>	3x No JSON (HTTP status code 200)
<code>https://bbdata.daplab.ch/api/comments/</code>	GET/PUT	<b>Get all comments from objects you have read access to</b> <code>https://bbdata.daplab.ch/api/comments</code> <b>Add a comment attached to an object</b> <code>https://bbdata.daplab.ch/api/comments?id="number"&amp;objectId="number"&amp;from="date"&amp;to="date"&amp;comment="comment"</code>	<pre>[ {   "from": "2016-08-24T08:07:49",   "to": "2016-08-24T09:07:49",   "comment": "Here is a comment",   "id": 7,   "objectId": 3 } ]</pre> No JSON (HTTP status code 200)

https://bbdata.daplab.ch/api/comments/{id}	GET/DELETE/POST	<p><b>Get one comment</b> https://bbdata.daplab.ch/api/comments/"id"</p> <p><b>Delete a comment</b> https://bbdata.daplab.ch/api/comments/"myId"</p> <p><b>Edit a comment</b> https://bbdata.daplab.ch/api/comments/id?objectId="number"&amp;&amp;from="date"&amp;&amp;to="date"&amp;&amp;comment="comment"</p>	<pre>[   {     "from": "2016-08-24T08:07:49",     "to": "2016-08-24T09:07:49",     "comment": "Here is a comment",     "id": 7,     "objectId": 3   } ]</pre> <p>No JSON (HTTP status code 200)</p>
--	-----------------	---	---

## B Source code

The entire source code of the project can be found on my personal GitHub:

<https://github.com/JobinJohan/DataViz>

The folder of the project contains *three folders* and four web pages:

- *css*: this folder is composed of multiple cascading style sheets such as the one for bootstrap, the one for noUiSlider, the one for fonts and the one for the style of the dashboard.
- *js*: this folder contains all JavaScript files to draw the visualizations, connect to the database, create dynamic tables, etc.
- *ressources*: this folder contains only elements of design such as the logo and the loading gif.
- connexion.html
- dashboard.html
- dashboard\_listSensors.html
- dashboard\_profile.html

## References

- [1] Mike Bostock. D3 data-driven documents, 2018. <https://d3js.org/>.
- [2] jQuery foundation. jquery write less, do more., 2018. <https://jquery.com/>.
- [3] avxto. nuselectable, 2015. <https://github.com/avxto/nuSelectable>.
- [4] SpryMedia. Add advanced interaction controls to your html tables the free and easy way, 2018. <https://datatables.net/>.
- [5] Loading.io. Loading.io build your ajax loading icons with svg/css/gif/apng, 2018. <https://loading.io/>.
- [6] Leon Gersen. nouisliders, 2018. <https://refreshless.com/nouislider/>.
- [7] exupero. savesvgaspng, 2018. <https://github.com/exupero/saveSvgAsPng>.
- [8] Dave Gandy. Font awesome, 2018. <http://fontawesome.io/license>.
- [9] iCoSys. Big building data a cloud platform for data storage and processing for smart living lab, 2016. <https://icosys.ch/wp-content/uploads/pdfs/bbdata.pdf>.
- [10] emilyemorehouse. Axios - promise based http client for the browser and node.js, 2018. <https://github.com/axios/axios>.
- [11] SVG. The world's largest web developer site, 2018. [https://www.w3schools.com/graphics/svg\\_intro.asp](https://www.w3schools.com/graphics/svg_intro.asp).
- [12] Bootstrap. Bootstrap, 2018. <https://getbootstrap.com/>.
- [13] Angus Gibbs. Statsjs, 2012. <https://github.com/angusgibbs/stats.js>.
- [14] Bbdata. Bbdata, 2018. <https://bbdata-admin.daplab.ch/auth/>.
- [15] Tompiler. Line chart, 2017. <http://bl.ocks.org/tompiler/24fc6c48058e575576915479c8ae8ab5>.
- [16] Mike Bostock. Scatterplot, 2018. <https://bl.ocks.org/mbostock/3887118>.
- [17] Gerardo Furtado. Box plot, 2018. <https://stackoverflow.com/questions/43281482/d3-js-boxplot-with-already-calculated-data>.
- [18] Arpit Narechania. Condegram spirial plot, 2018. <https://bl.ocks.org/arpitnarechania/027e163073864ef2ac4ceb5c2c0bf616>.
- [19] Joshua Latimore. Circle path generator, 2016. <http://bl.ocks.org/js16906/cb75852db532cee284ed>.
- [20] Dan Joseph. Calendar heat map, 2018. <https://bl.ocks.org/danbjoseph/3f42bb3f0ab6133cfc192e878c9030ed>.
- [21] D3Vienno. D3.js tutorial-17-the histogram layout (1/2), 2013. <https://www.youtube.com/watch?v=cu-I2um024k>.
- [22] D3Vienno. D3.js tutorial-17-the histogram layout (2/2), 2013. <https://www.youtube.com/watch?v=0CZ7-f9wXiM>.
- [23] Mike Bostocks. Arc clock, 2018. <https://bl.ocks.org/mbostock/1098617>.
- [24] Iconfinder. Iconfinder, 2018. [https://www.iconfinder.com/icons/111049/bulb\\_idea\\_light\\_icon#size=128](https://www.iconfinder.com/icons/111049/bulb_idea_light_icon#size=128).
- [25] David Banks. Thermometer with d3.js, 2018. <https://codepen.io/davidbanks/pen/rksLn>.