Visualization of data flows in the Internet of Things

Bachelor Thesis

Rouiller Bastien

29/07/2016

Supervisor : Graf Marcel

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD

www.heig-vd.ch

Acknowledgements

I wish to express my sincere thanks to my thesis supervisor during this project, Mr. Graf Marcel, who provided me precious assistance and guidance through the process of realizing this project.

I would like also to thank M. Julien Rebetez who explained to me the Android app used for previous events and that I used for further tests.

Abstract

Large amounts of data can be collected through sensors from the real world, thus constituting the Internet of Things (IoT). The IoT infrastructure is composed by sensors, low energy networks, gateways, servers that collect the information and databases. The last two of those elements are usually located in the cloud. The role of the infrastructure is to collect, store and process sensor data streams. The main challenge consists of dealing with the huge quantity of data, extracting insight from it.

To monitor a big event such as the Paleo festival an application is needed that allow following participants in real time with a heatmap, or explore past measurements with various visualization tools. In this work we develop a tool to explore IoT data and we describe the process of designing and implementing a web graphical console. This console allows visualizing data from participants' phones through different approaches, among them rendering information such as location, speed and bearing on a map. It aims to make the work of the IoT engineer easier by allowing him to stay on top of the incoming data streams, quickly get an overview of what the data are like and keep an eye on the database.

Visualization methods were explored and implemented with aggregations, bubble maps, charts and others. Performance has been an important concern and improved through aggregation, indexing and distributing the load between server and client. The user is able to interactively explore the data by using cross filters (smart controllers), such as sliders on top of histograms that represent the distribution of the data.

Requirements specification

In the Internet of Things (IoT) applications collect large volumes of data originating from many physical sensors deployed in the real world or logical sensors inside software components. The sensors continuously measure, often several times per minute, physical phenomena (temperature, pressure, ...) or detect events (presence of a car, door is open, user clicked on a button, ...). IoT applications need an infrastructure to collect, store and process sensor data streams: message queues, datastores and compute nodes. This infrastructure is often located in the cloud, where resources are dynamically allocated and freed depending on the load (pay-as-you-go billing model).

For an operations engineer it can be a challenge to remain on top of the incoming data. The volumes of the different data streams can be highly variable. This makes it difficult to predict the load on the storage and compute resources. Problems in the data delivery (network interruptions, misconfigured gateways) are difficult to detect because of the large volume of data.

The goal of this project is to develop a graphical console for IoT operations engineers that provides a number of visualizations of the data streams. The visualization should help in understanding the data at two levels:

- Characteristics of the data flow: Its volume, the distribution of sources contributing to the data flow, its historical evolution, etc.
- The content of the data: There are many different types of sensors, but some data attributes are shared by different sensors, for example the geographic location of the sensor.

The visualizations should also work in two modes regarding time:

- Real-time visualization of data being measured right now
- Visualization of historical data

Objectives

- The console provides a number of different visualizations of the data flow and the data content that can be explored interactively by the operations engineer.
- Definition of use cases for capacity planning and troubleshooting.
- Design of visualizations and their interaction model
- Implementation and integration with an existing data collection component based on MongoDB
- Test using pre-recorded real-world sensor data and by generating simulated data
- Implementation as a web application in Python
- Visualization implemented by using a client-side (JavaScript) library to enable fluid interactivity

Contents

Acknowledgements ..... **iii**

Abstract ..... **iv**

Requirements specification ..... **v**

Objectives ..... **v**

Chapter 1 Introduction **2**

1.1 Internet of Things 2

1.2 iNuit (Internet of Things for Urban Innovation) 2

1.3 Challenge 2

1.4 Focus of the project 2

Chapter 2 Analysis **3**

2.1 Existing solution 3

2.1.1 Architecture 3

2.2 Future Architecture 4

2.3 GeoMapping 5

2.3.1 TopoJSON with D3 5

2.3.2 Leaflet with OpenStreetMaps 6

Chapter 3 Tools and programming language **8**

3.1 MongoDB 8

3.2 Python 8

3.3 Django 9

3.4 Javascript 9

3.4.1 D3.js 9

3.4.2 dc.js 10

3.4.3 Leaflet.js 10

3.4.4 Leaflet-heat.js 10

Chapter 4 Visualization techniques 11
4.1 Choosing a visualization method 11
4.2 Aggregation 12
4.2.1 Problem 12
4.2.2 Using aggregation 12
4.2.3 Bubble map 13
4.2.4 Directional arrow 15
4.2.5 Combination 15
4.3 More detailed visualization methods 16
4.3.1 Paths 16
4.3.2 Individual path 17
4.3.3 Heat map visualization 17
4.4 Visualization parameters 18
4.5 Controls: Cross filter 19
4.5.1 Geographical area 19
4.5.2 Time slider 20
4.5.3 Sensor type 20
4.5.4 Speed slider 21
4.6 Final result 21
Chapter 5 Monitoring the database 22
5.1 Disk space availability 22
5.2 Prediction 22
Chapter 6 Implementation 23
6.1 Server side 23
6.1.1 Web server 23
6.1.2 REST API 23
6.2 Client side 24
6.2.1 Rendering 26
6.2.2 Diagram 28
Chapter 7 Performance optimization 30
7.1 Aggregation 30
7.1.1 Purpose of aggregation 30

7.1.2 Example 30
7.1.3 Aggregation on the server side 30
7.1.4 Example of trade-off 31
7.1.5 Implementation 31
7.2 Indexing 32
7.2.1 2dsphere Indexing performance 32
7.2.2 How does index improves performance? 33
7.2.3 Pre-Computing in the DB 34
Chapter 8 Difficulties 36
8.1 Data format 36
8.2 Projection problem 36
8.3 Improving user experience 37
8.3.1 Lost in cyberspace 37
8.3.2 User-friendly values 37
8.3.3 Legends 37
8.4 Measurements errors 38
Chapter 9 Tests 40
9.1 Testing with pre-recorded real-world sensor data 40
9.2 Generating measurements 41
9.3 Benchmark between browsers 41
9.4 Browsers compatibility 42
9.4.1 Responsiveness 42
Chapter 10 Project progress 43
10.1 Week 1 – 22.02-28.02 43
10.2 Week 2 – 29.02-06.03 43
10.3 Week 3 – 07.03-13.03 43
10.4 Week 4 – 14.03-20.03 43
10.5 Week 5 – 21.03-27.03 43
10.6 Week 6 – 04.04-10.04 43
10.7 Week 7 – 11.04-17.04 43
10.8 Week 8 – 18.04-24.04 43
10.9 Week 9 – 25.04-01.05 44

10.10 Week 10 – 02.05-08.05 44
10.11 Week 11 – 09.05-15.05 44
10.12 Week 12 – 16.05-22.05 44
10.13 Week 13 – 23.05-29.05 44
10.14 Week 14 – 30.05-05.06 44
10.15 Week 15 – 06.06-12.06 44
10.16 Week 16 – 13.06-17.06 44
10.17 17.06 44
10.18 Week 17 – 20.06-26.06 44
10.19 Week 18 – 27.06 – 03.07 45
10.20 Week 19 – 04.07 – 10.07 45
10.21 Week 20 – 11.07 – 17.07 45
10.22 Week 21 – 18.07-24.07 45
10.23 Week 22 – 25.07-29.07 45
10.24 29.07 45
Chapter 11 Conclusion 46
11.1 Achieved Results 46
11.2 Future development 46
List of Figures 47
References 51
Annexes 53
Chapter 1 Annex 1: Installation manual 54
1.1 Introduction 54
1.2 Requirements 54
1.3 Steps 54
1.3.1 Retrieving the code 54
1.3.2 Running the code 54
1.3.3 Accessing the console 55
Chapter 2 Annex 2: Operation manual 56
2.1 Introduction 56
2.2 Requirements 56
2.3 Accessing the console 56

2.4 Using the map 57
2.5 Choosing the visualization method 57
2.5.1 Circles 58
2.5.2 Arrows 59
2.5.3 Both 59
2.5.4 Resolution 59
2.5.5 Paths 60
2.6 Displaying the heatmap 61
2.7 Controls 62
2.7.1 Time controls 62
2.7.2 Speed controls 63
2.7.3 Sensor type controls 64
2.8 Monitoring the database 64

Chapter 1 Introduction

1.1 Internet of Things

The internet of things or IoT is the extension of Internet to the field of physical devices. Devices are embedded with thermometers, sensors or microchips and network connectivity. Measurements are directly sent to the cloud or transit through a gateway. A large amount of data is usually collected and it contributes to the big data. Special tools need to be developed to process this kind of data. Traditional tools are no longer sufficient. [1]

As described by Intel, there are various applications of IoT to improve people's lives. In the agriculture field it can improve the irrigation system by using multiple sensors measuring humidity level. It helps using the exact amount of water and avoids the waste of water or drought. In healthcare, by collecting thousands of real-time patient's data, a health profile can be established and a better treatment decision can be make via analytics. [2]

1.2 iNuit (Internet of Things for Urban Innovation)

iNuit consists of a group of projects related to the research about the Internet of Things. Those projects are conducted by several universities of applied sciences. In a first phase the Inuit project focuses on big events such as music festivals (Paleo) by monitoring the crowd. The aim is to improve security and mobility in this kind of event and in the future apply this to the scale of a city. [3]

1.3 Challenge

The biggest challenge in the Internet of Things is not the collection of data itself but rather the exploration of what is collected. We have a huge number of measurements since data are periodically sent by thousands of sensors. Sensors are very basic equipment and thus sent simple information. It's the data analyst role to explore those data and extract insights out of it.

Designing a user-friendly interface in order to provide a good user experience and reducing response time when interacting with the database are part of this challenge.

1.4 Focus of the project

The project focuses on data collected by smartphones. Later in this document I will describe how to allow the operations engineer to explore those data such as zooming in a geographical and time space to narrow down to a subset of data. We want to provide a tool to an operations engineer to facilitate the exploration of data generated during big events and detect behaviors like crowd movements.

The operations engineer must also be able to monitor the database state and disk occupation in order to anticipate future growth of data.

Chapter 2 Analysis

First of all, it is necessary to know what was built until now in order to choose a correct approach. The choice of a geomapping solution is also crucial for the following of this project and must be made before implementing visualization methods which are discussed later.

2.1 Existing solution

2.1.1 Architecture

A solution for collecting and storing data has already been developed. The smartphones automatically send measurements using an App to a REST-API. I described a simplified version of the process in this diagram below:

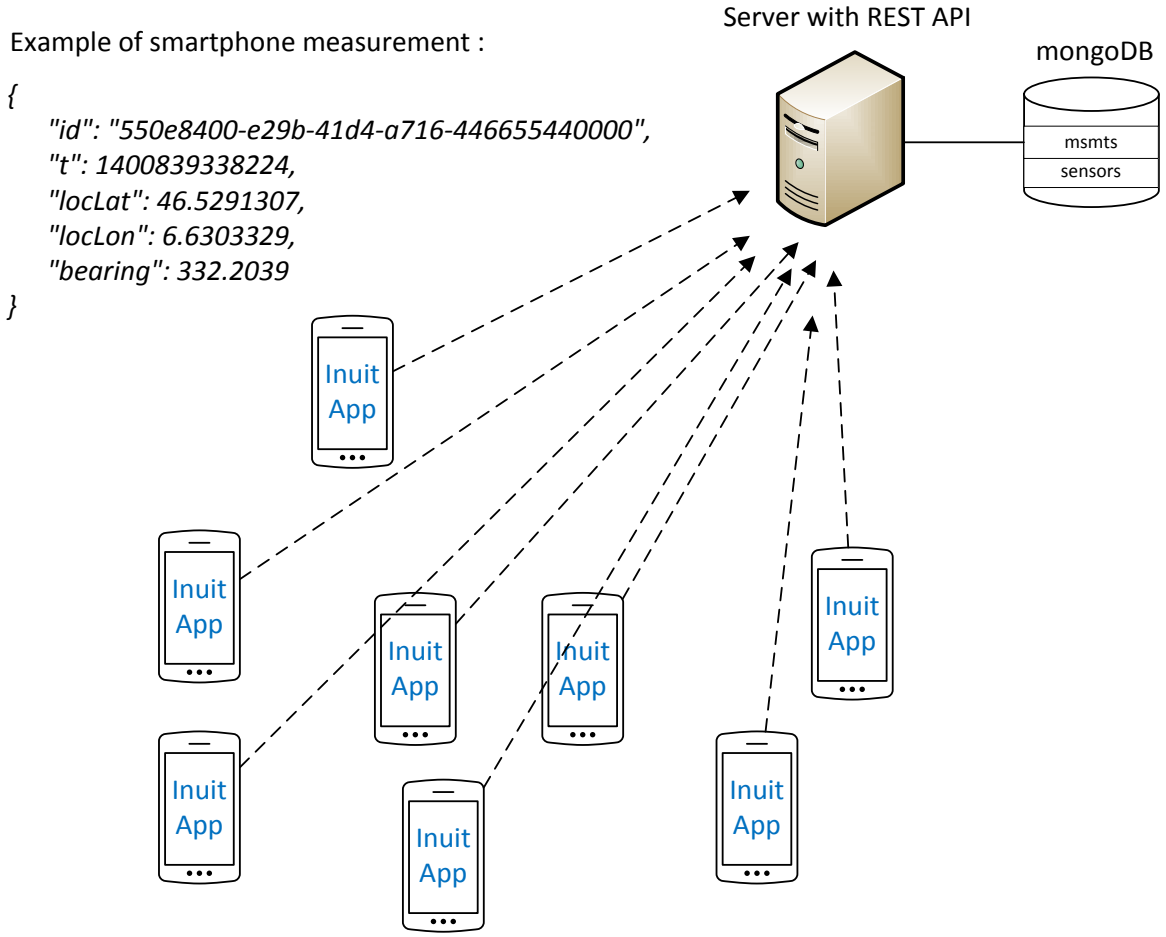


Figure 2:1 Collection of data

In the example measurement, the smartphone sends an identifier representing the device, a timestamp, a geographical position and a bearing indication by http to the server's REST API. The server receives the JSON and is able to store it in the same format because mongoDB uses a format similar to JSON. The mongoDB database consists of two collections, one for storing the sensors and the other for storing measurements.

For further information about the REST API that was previously built, consult the specs here: http://inuit.iict.ch/1/doc/spec.html

It was later decided with my thesis supervisor, that building our own REST API for the creation of measurements is better in order to have full control on it. The specs are discussed later in this document.

2.2 Future Architecture

The future architecture is an extension of the current architecture but with a different REST API and a web server allowing the operations engineer to access the new developed web console.

Typically the operation engineer is accessing the web console from his browser. Firstly the web browser will send a HTTP request to the webserver. Then, when the browser will receive the JavaScript code, along the HTML, it will be able to asynchronously execute and to send requests to the REST API.

For example, it will reach the endpoint "/countsBySquare" in order to have a aggregations of measurements by area. The server will send a response in the form of JSON.

Finally the JavaScript will represent the data on the map. Next time the user will interact with the controls, JavaScript will again send requests to the REST API without reloading the entire page and will update what is shown on the map. The Javascript is only on the client side.

In the future implementation, the web server and the DB server are on the same machine. In production we could imagine that another machine hosts the DB server. Only a few settings are required to make this change in the code.

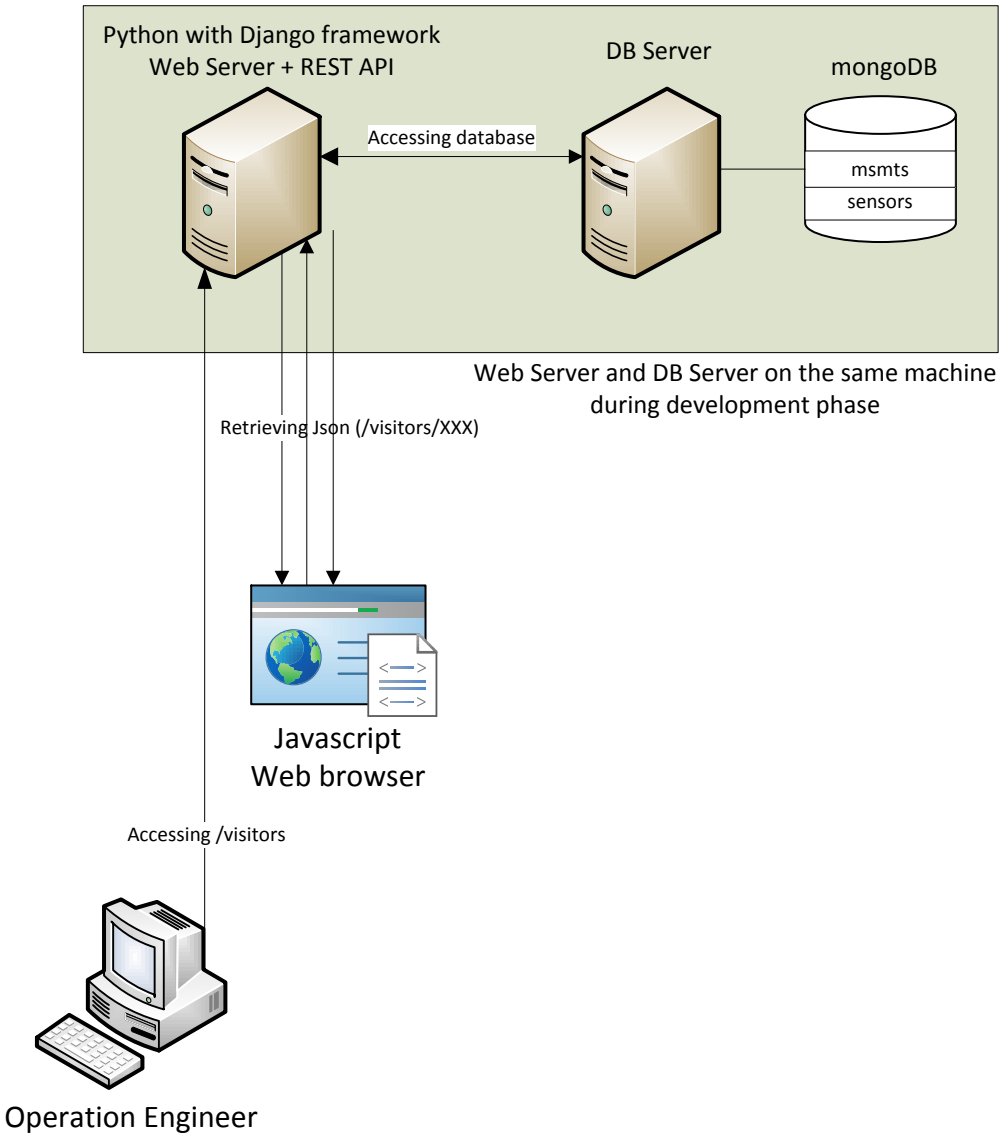


Figure 2:2 Physical diagram

2.3 GeoMapping

We need to represent GPS coordinates collected by the sensors on a map. Choosing a geomapping solution is crucial for the next phases of the project. There are two interesting solutions to do it on a web browser. The first solution is to use TopoJSON [4], an extension of GeoJSON [5], that encodes topology with D3. Another solution uses the Leaflet library.

2.3.1 TopoJSON with D3

TopoJSON is a format for encoding a variety of geographic data structures. We can use it along D3 to represent a map. The first thing to do is to find boundaries in a correct format. The data can be obtained from official sources such as governments. In our case we will use the data from the Swiss Federal Office of Topography (swissBOUNDARIES3D) through the Swiss Maps projects by Jeremy Stucki (https://github.com/interactivethings/swiss-maps) that allows generating the correct format.

Figure 2:3 Projection of swiss municipalities from a GeoJSON file

The advantage of this solution is that we have full control of the geographical data because it is stored on the server. The client only interacts with the server and no third party service such as Google Maps is required.

The problem is that we are limited by the resolution of the TopoJSON file. It is not possible to “zoom” on the map as we want. Furthermore we are not able to see more details than the municipalities. For example we cannot see the streets. We then decided to choose the second solution.

2.3.2 Leaflet with OpenStreetMaps

As described previously in the report Leaflet uses preexisting maps and lets the user zoom and move the map as he wants. It also offers some tools such as markers and circles to show data on the map. However we can also use D3 to project objects on the map and have customized objects instead of Leaflet objects so we can show interactive data. We will discuss the kind of data we can show in a later chapter.

Figure 2:4 Example of Leaflet map with a marker

Chapter 3 Tools and programming language

In this chapter I will describe the tools I used in order to achieve my project. On the server side, MongoDB and Python are mentioned in the requirements specification as well as Javascript for the client-side. The Javascript libraries were chosen after discussion with my thesis supervisor or after experimentations.

3.1 MongoDB

MongoDB is a NoSQL and document-oriented database. The documents allow a dynamic schema; there is no restriction about document size and format. MongoDB also allows sharding. Sharding is the process of storing data records across multiple machines in order to spread the load [6]. That makes it a perfect database to store Internet of Things data.

MongoDB uses BSON in order to store its documents. BSON is an extended implementation of JSON and allows efficient coding and decoding with different languages. It is a big advantage because it facilitates the use of a REST API. After retrieving data in the DB you can almost directly forward it through HTTP in a JSON format.

```
{
  name: "sue",                    ←— field: value
  age: 26,                        ←— field: value
  status: "A",                    ←— field: value
  groups: [ "news", "sports" ]    ←— field: value
}
```

Figure 3:1 Example of MongoDB document [6]

3.2 Python

Python is a high-level language allowing to code in a few lines while in C++ or Java it would take much more. It is dynamically typed, strongly typed, object oriented and emphasizes code readability. [7]

I used it hand in hand with pyMongo in order to query the mongoDB database.

```
cursor = db.restaurants.find({"grades.score": {"$gt": 30}})
```

Figure 3:2 Example of MongoDB query made with pyMongo [8]

```
for document in cursor:
    print(document)
```

Figure 3:3 Iterating over results in a document [8]

3.3 Django

Django is a Python web framework that encourages rapid development. It follows the Model-View-Controller (MVC) architectural pattern. I am using it to host the web console and provide at the same time a REST API.

```
# /visitors/countsBySquare
url(r'^countsBySquare/$', views.countsBySquare, name='countsBySquare'),
```

Figure 3:4 Sample of Django routing configuration

When accessing the URL "/visitors/countsBySquare", the request will be redirected to the view "countsBySquare". Then after making a request to the DB in this view we can respond by sending a JSON document via HTTP.

3.4 Javascript

Given that our web application will need some interaction with the user, Javascript is a must-have. It is used alongside HTML and CSS in the browser on the client side. By dynamically refreshing the selected area on a page without refreshing the whole page it enhances the user's experience.

3.4.1 D3.js

D3 allows to use multiple selectors in a same ways that jQuery does. What makes D3 better for my project is that it facilitates the manipulation of data. For example, you can easily generate an object for each value in an array and dynamically set values to it as seen in this extract of code.

```
var dataset = [ 25, 7, 5, 26, 11 ];

d3.select("body").selectAll("div")
    .data(dataset)
    .enter()
    .append("div")
    .attr("class", "bar")
    .style("height", function(d) {
        var barHeight = d * 5;
        return barHeight + "px";
    });
```

Figure 3:5 Example of D3 code to make a simple bar chart [9]

In order to achieve this:

Figure 3:6 Rudimental barchart constructed with D3 [9]

You can extend it to make much more complex visualization tools.

3.4.2 dc.js

This library is used to “leverage D3 engine to render charts in css friendly svg format” [10]. It allows using higher level commands than with D3 engine in order to render charts such as histograms. The charts are interactive and thus can be used to select a specific range of values with a brush for example.

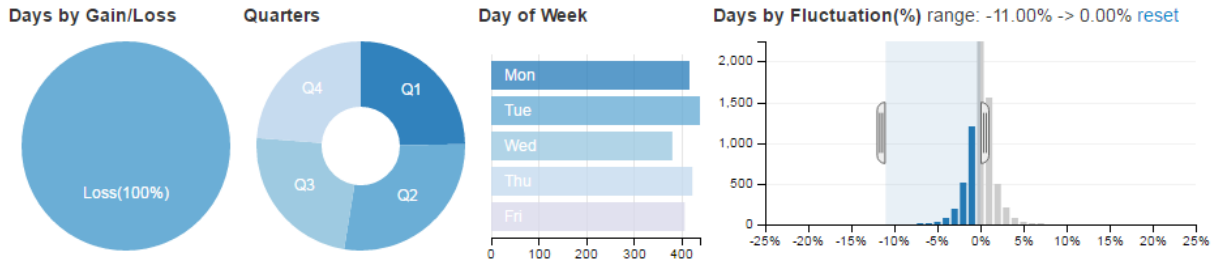


Figure 3:7 Example of dc chart [10]

3.4.3 Leaflet.js

Leaflet is a Javascript library used to create web mapping application. It allows the programmer to easily display tiled web maps and put an interactive layer on top of that. The tiles can typically be fetched from OpenStreetMaps.

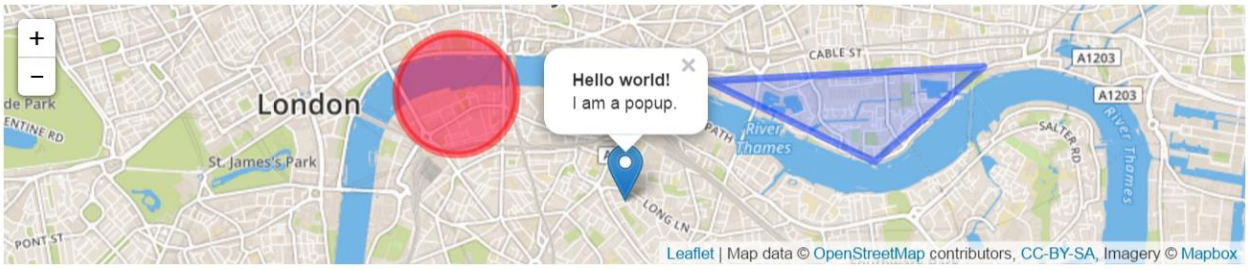


Figure 3:8 Leaflet map with marker, circle and polygon [11]

3.4.4 Leaflet-heat.js

Leaflet-heat is a plugin for leaflet. A heat map can be rendered as a layer on top of a leaflet map.

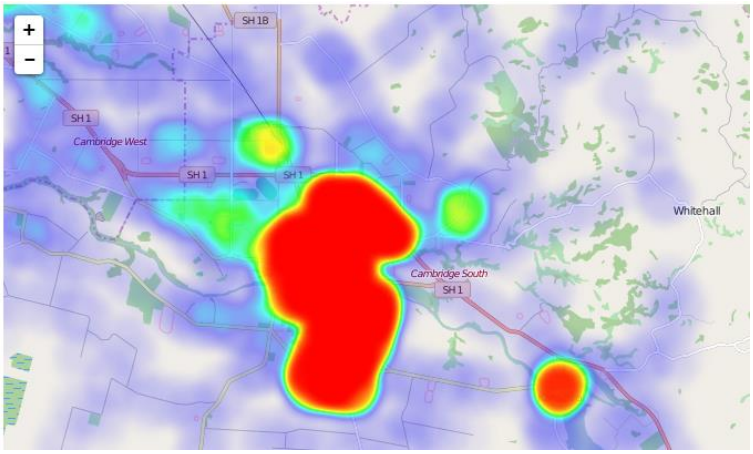


Figure 3:9 Heat map with Leaflet-heat.js

Chapter 4 Visualization techniques

Visualization techniques are the core of my project. I will describe pieces by pieces the methods chosen to display the data and how to extract what the operations engineer need to see.

Scott Murray mentions in his book: “Data visualization is an interdisciplinary field” [9]. It is true that it does not only require web developer skills. Choosing an adequate way to extract insight from data requires research and depends on many factors I will later describe.

4.1 Choosing a visualization method

There are two main factors in choosing a visualization method: the kind of data and what we want to show. When a smartphone is running the SmartCrowd app, it sends a measurement every 5 seconds. Here is how a measurement is stored in the database:

```
{
      "_id" : ObjectId("54476e1cbb20a622ae21c66c"),
      "_updated" : ISODate("2014-10-22T08:43:08Z"),
      "bearing" : 0.6000000238418579,
      "name":"Android[product=vodafone_ch-de,brand=htc,
manufacturer=HTC,model=HTC One_M8]",
      "locAcc" : 12,
      "speed" : 1.5,
      "t" : ISODate("2014-10-22T08:43:08Z"),
      "locAlt" : 516,
      "_created" : ISODate("2014-10-22T08:43:08Z"),
      "type" : "smartphone",
      "id" : "a2b58698-f412-4ca3-aad7-88e788cfef35",
      "nodeLoc" :
            {
                  "type" : "Point",
                  "coordinates" : [ 6.65975984, 46.77927315 ]
            }
}
```

Figure 4:1 Measurement sample in the mongoDB database

Those are the most important fields for this project:

_id	Universally unique identifier [13]
bearing	Indicates the direction of motion, show the direction the event participant is heading.
name	Gives information about the sensor
speed	Speed in m/s
t	Timestmap, indicates when the measure was made

id	Identifies the sensor that took this measure
nodeLoc	Store the GPS location in GeoJSON format [5]

Figure 4:2 Measurement fields in the database

We want to show how people behave. Basically, how many people are in one place are a given moment or moving to a place.

4.2 Aggregation

4.2.1 Problem

What efficient method should we use to display information on a map? The display of GPS locations can be made using markers provided by Leaflet.

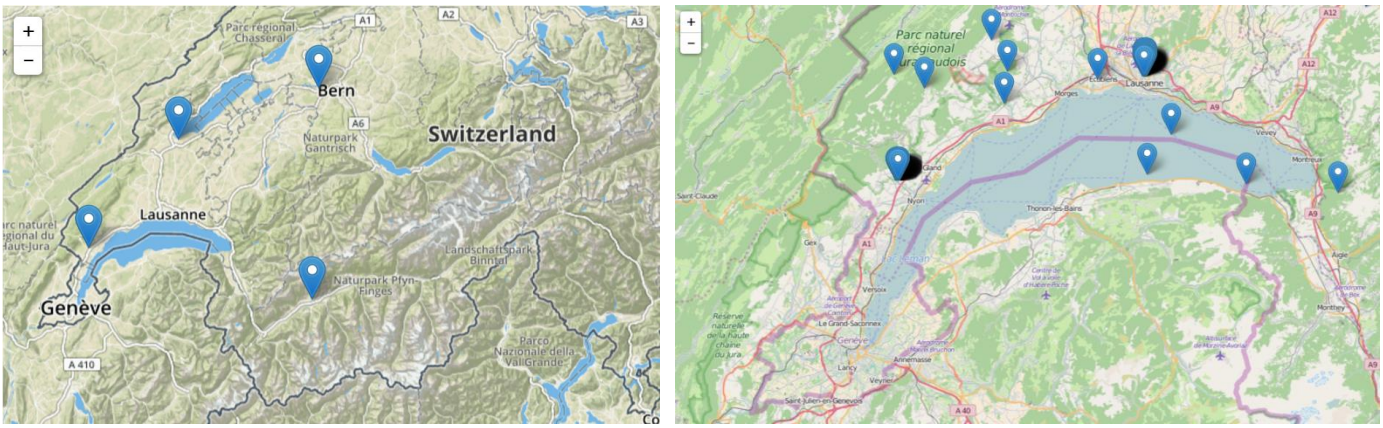


Figure 4:3 Using leaflet markers on a map

However it is not a suitable method in our case. First of all, we have millions of measurements and some areas have hundreds of thousands of measurements in a single place; for example, in the Paleo festival. It is almost impossible to determine if a place has twice the data than another when you have thousands of measurements. On the second image above, you may think there were more measurements in Lausanne than in Nyon. However there are 5 times more measurements in Nyon. This is because the markers overlap.

Furthermore the measurements have more than GPS coordinates; they also contain information about the time dimension and the speed and this methods do not allow to explore that.

We need a way to efficiently group the information. Aggregation meets this need.

4.2.2 Using aggregation

We want to show how many measurements were made in an area, such as a city, a festival or a street. The exact location of each measurement is not important; we just need a general idea.

In order to have something easier to interpret for the human eye, regrouping measurements can be a good approach. By creating an aggregation on the map, the user is able to see in a blink of eye where most of measurements are located. I explored two kinds of aggregations: square aggregation and dynamic aggregation. The first one consists of manually regrouping measurements by square area on the map so we

can get a kind of grid. The second one use more complex algorithms which dynamically determine cluster to regroup locations. It has been implemented in a plugin for Leaflet. [13]

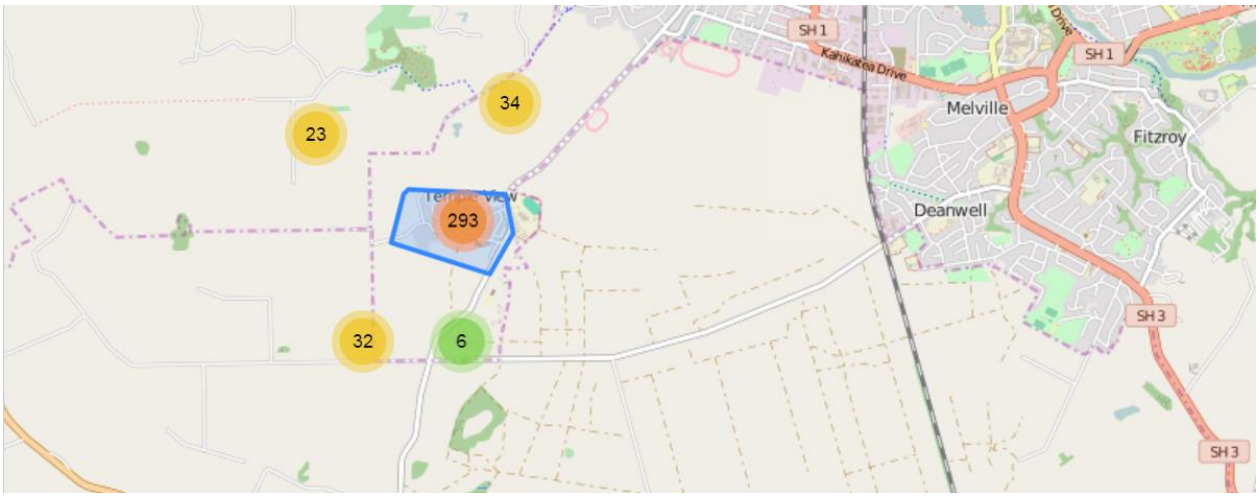


Figure 4:4 Aggregation using clustering [13]

The second aggregation, which uses dynamic clustering, can be used for the visualizations that only represent geographic location. We chose the manual square aggregation, because we want to develop our own algorithm and display more information such as speed. I am now going to describe the two aggregation visualizations methods I used.

4.2.3 Bubble map

The method consists of aggregating the quantity of measurements by area and showing the average speed. The operations engineer can thus differentiate the most frequented areas from the transit areas.

The aggregation is represented by circles, each of them located in the middle of each cell of a grid. The circle size will depend on the quantity of measurements in his area and the color is related to the average speed. I drew a grid to show the aggregation principle.

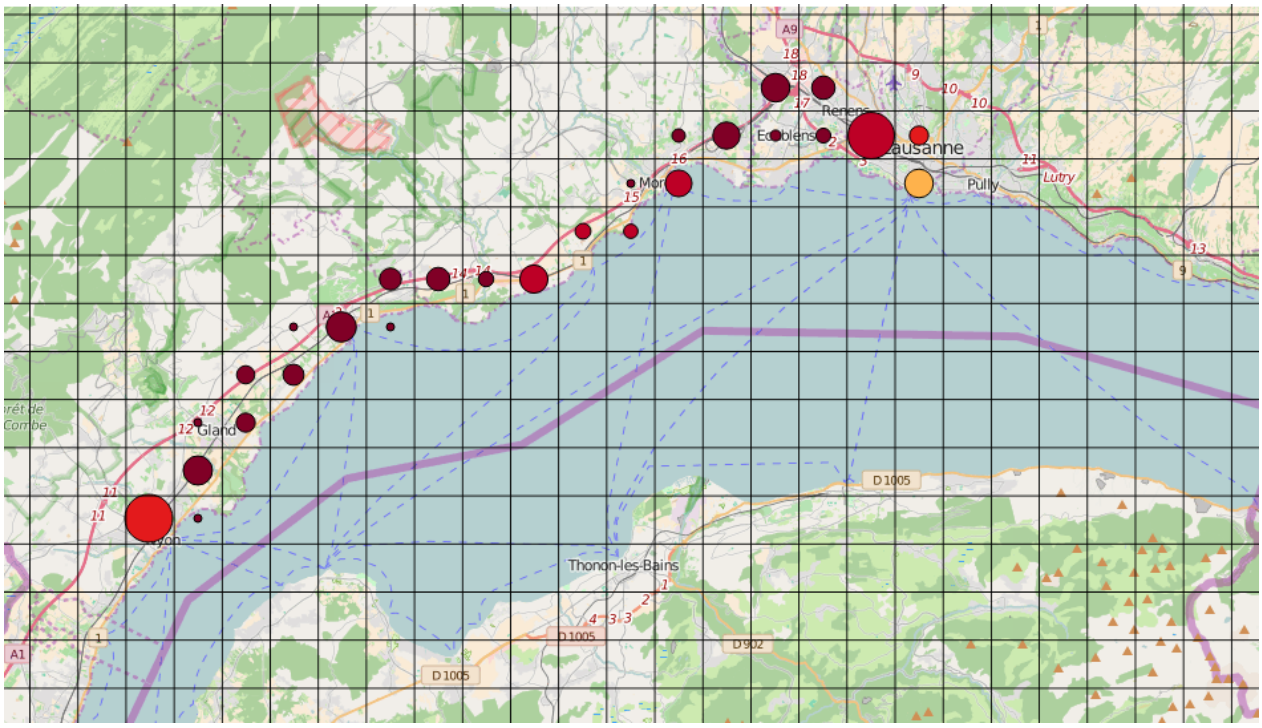


Figure 4:5 Representation of geographical aggregation with circles

It was inspired by this example below from Mike Bostock, the creator of the library D3.js. In this case, he uses a JSON file for drawing the states and the circles are centered on counties. However it is not possible to zoom on the map since it is static. In our case we are able to move the map and zoom. Each time the operations engineer make a different requests using the cross filters (described later) the bubbles will refresh.

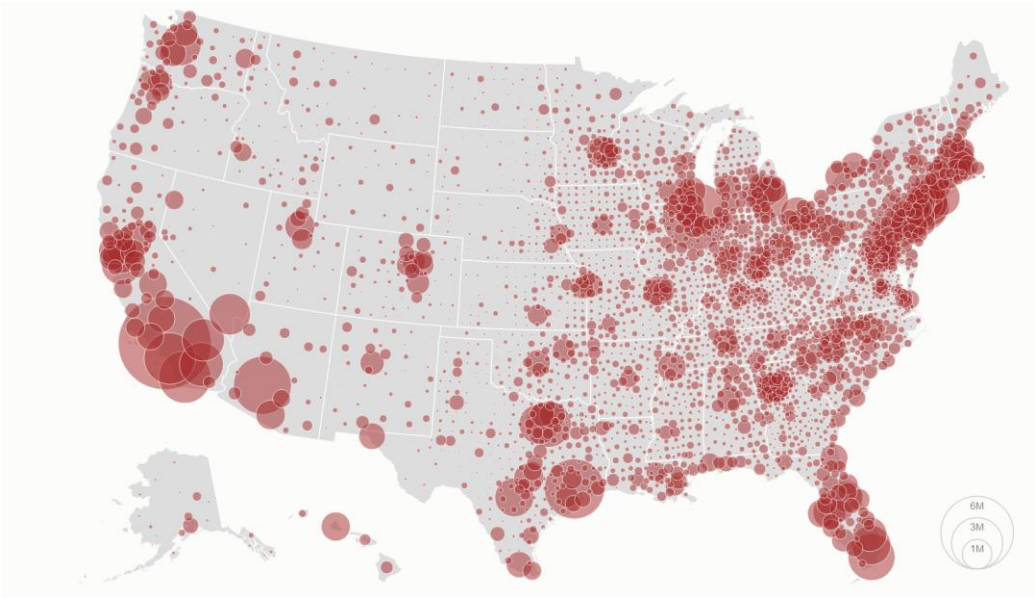


Figure 4:6 Example from Mike Bostock, creator of D3 and inspiration for the bubble map [14]

Aggregation has two advantages. The first advantage is that it is easier for the user to quickly compare the values of two areas. The second advantage is that the quantity of data transferred between the server and

the user is reduced if the aggregation is done on the server side, this process is described later in this document.

4.2.4 Directional arrow

In certain cases, we want to focus on people's movements and directions; we need a straightforward visualization method simulating this.

Most measurements have bearing and speed parameters when they are stored. We can use them to show the average flow of people. In order to represent this I use arrows. Each arrow has a length and a direction; the length representing the speed and the direction representing the bearing. The same aggregation is made than with circle, each measurement in a square area is grouped.

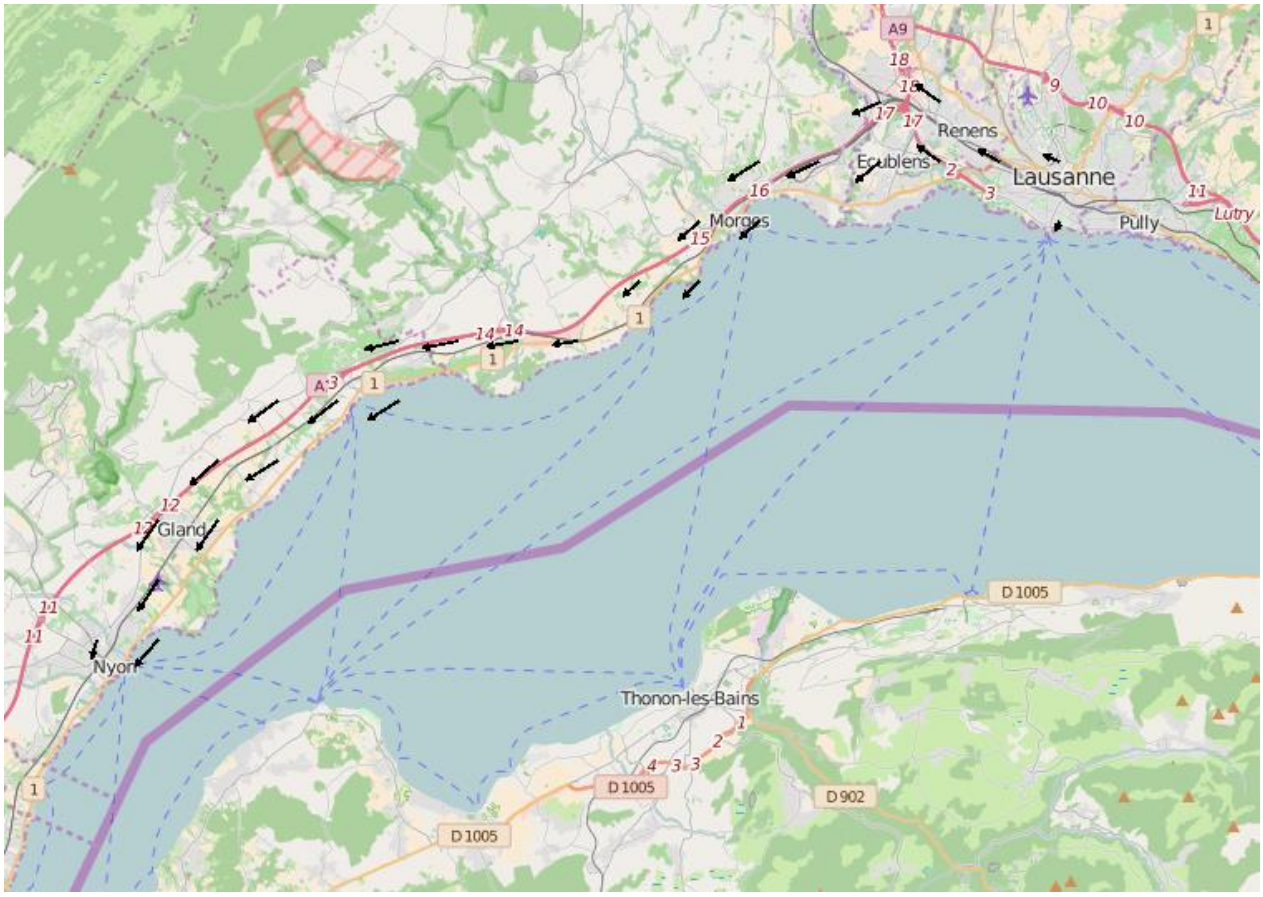


Figure 4:7 Aggregation of with Arrows on the map to show speed and direction

4.2.5 Combination

By combining these two visualizations methods, the operations engineer is already able to isolate some behaviors.

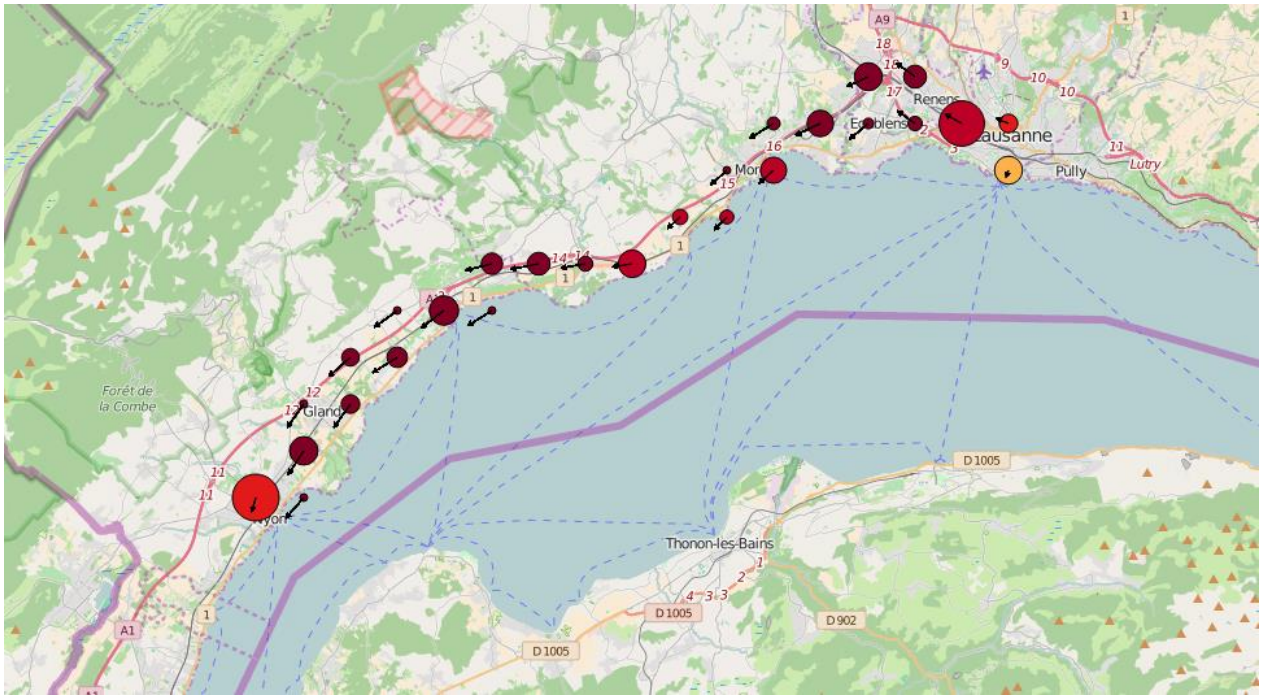


Figure 4:8 Combination of visualization methods

In a blink of an eye the IoT engineer can notice that there is a flow of people that goes from Lausanne to Nyon. Later we will introduce the legends to show the scale of those data.

4.3 More detailed visualization methods

The operations engineer might need a more detailed view on the measurements. For example, he wants to display the movements of each festival participant in a specific area. Or else, he wants to get the density of the crowd in different zones of a festival.

4.3.1 Paths

With this visualization method, the operations engineer can precisely track users' movements. In order to do this, each of the GPS coordinates captured by event participants' phone is connected in a chronological way on the map. The triangles show the direction in which the participants are heading.

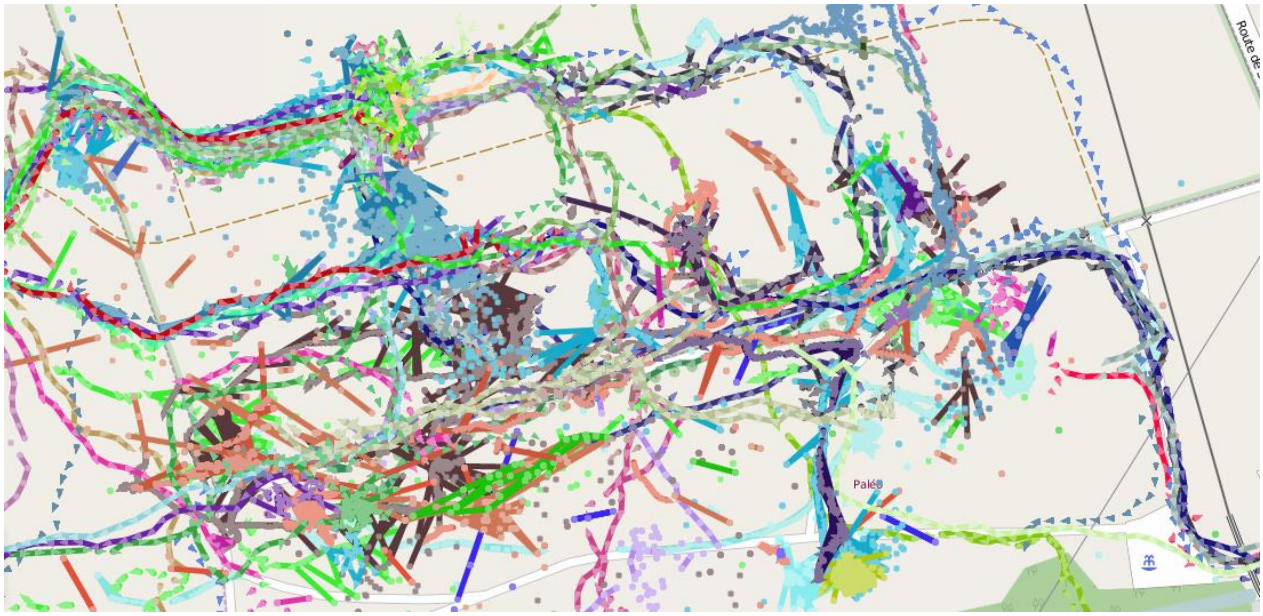


Figure 4:9 Vizualisation of paths taken by event participants during two days in the Paléo Festival

4.3.2 Individual path

It is possible to click on path to select a specific user. Then you can follow all of his movements. Below we can see an example of this functionality. We see the user arriving by train at Nyon station and then probably taking a bus to the festival since the path is following the road and taking the roundabout in the correct way.

Figure 4:10 Path of a user arriving by train at the Paleo Festival

4.3.3 Heat map visualization

We need a method to show the festival participants density during a time interval such as the last five minutes. It is also usefull to have a regular refresh of the visualization to see a quasi-real time evolution of the crowd movement.

A good method to quickly spot people concentration is a heat map. In order to avoid confusion it was discussed with my supervisor to move this to a separate map. The heat map uses a color scheme to show the concentration of measurements by surface area.

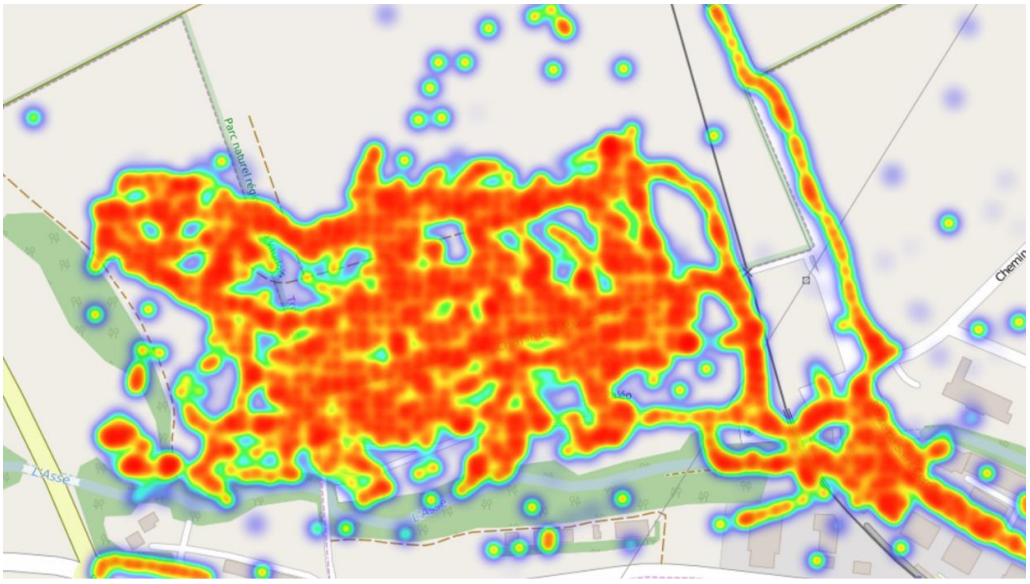


Figure 4:11 Heat map of the Paleo festival

4.4 Visualization parameters

In order to choose which representation of data we want, there are controls. Those controls are directly located on the map to facilitate user interaction.

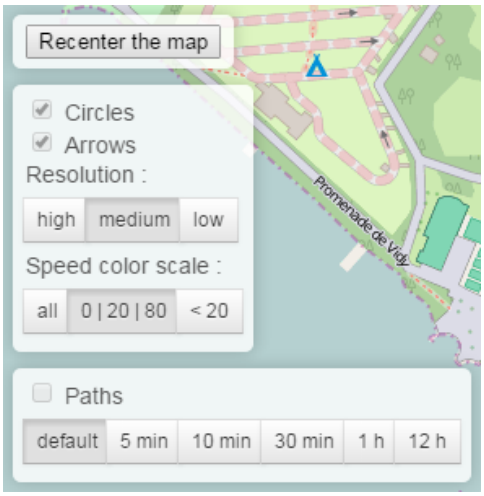


Figure 4:12 Visualization options and tunings

For the paths visualization, we can also select to show only recent data in order to limit the quantity of data retrieved from the server. For the Circles and arrow visualizations we can choose the resolution, higher resolution equals smaller circles, and speed color scale, determining the circles filling color.

The speed color scale allows isolating specific behavior. The default scale show any speed based on a logarithmic scale for the colors but others scales might be useful when detecting specific behaviors. Two

others scales have been studied. The first one allows classifying measurements into categories of means of transport. The second is used to distinguish pedestrians' behavior.

The "0 | 20 | 80" scale categorize measurements into three categories : 0 – 20 km/h for people on foot, 20-80 km/h for people on transport and 80+ km/h for people on faster means of transport like trains or highways. The <20 km scale can be used to focus on the movements of pedestrians.

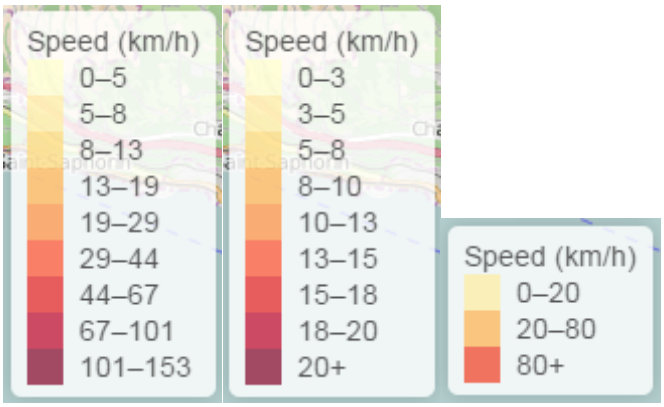


Figure 4:13 Speed color scales

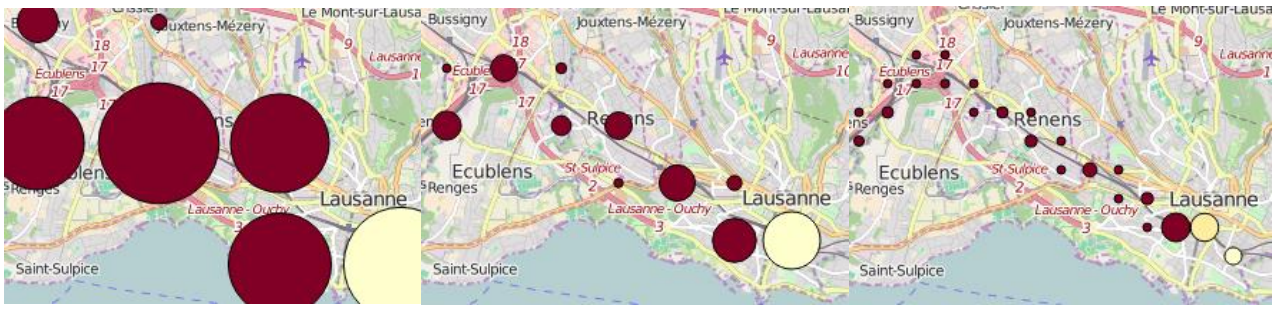


Figure 4:14 Circles resolution

4.5 Controls: Cross filter

As an operations engineer I wants to see how many measurements were made by hours during the week of the Paleo festival.

Cross filters are perfect for this kind of application. They are made of several controls. Each control displays data along a certain dimension and allows selecting a subset of the data. By combining the different controls (time, type, location, speed) the operations engineer is able to focus on a specific case.

4.5.1 Geographical area

The primary control is the map. It acts as a location control. Every time the user zoom in/out or drag the map to another location, the corresponding data will be selected. For example, he can move the map to the Paleo festival area in order to shows related data.

4.5.2 Time slider

The first control is a slider that allows selecting a range of date, allowing the operations engineer to select a specific week such as during the Paleo.

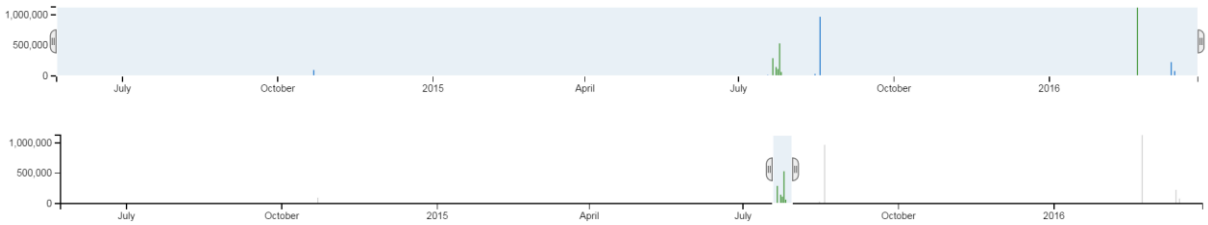


Figure 4:15 Date slider control

The seconds allows to user to choose a range of hour inside the selected range of days:

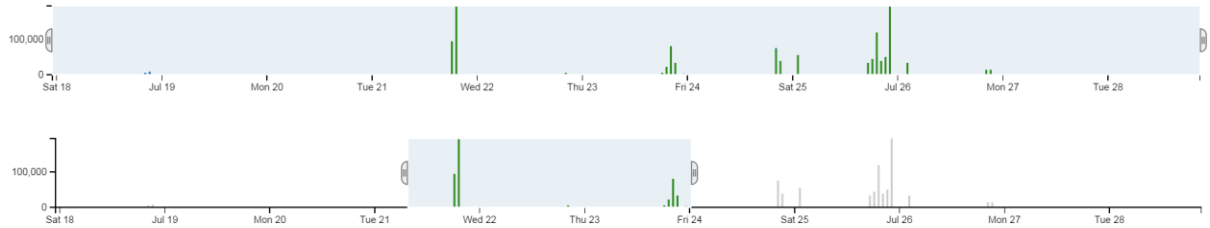


Figure 4:16 Hour slider control

Both sliders have a role in the visualization progress because they show how much measurements have been made each day and each hour, and the color shows if it passes all the filters (green) or not (blue). Basically the green part is shown on the map while the blue part is ignored.

4.5.3 Sensor type

Then you can choose which kind of smartphone to show:

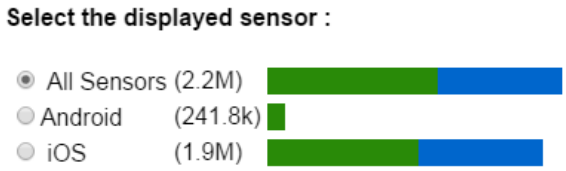


Figure 4:17 Sensor controls

We are keeping the same meaning of the colors: green for selected data and blue for the rest. We can see that selecting Android changed the color in the histogram shown on the slider:

Figure 4:18 Repercussion of crossfiltering on other control

It’s an example of how cross filters work. Same goes in the other way.

4.5.4 Speed slider

With the time slider you can quickly see the speed distribution and focus on specific means of travel. A good example is focusing on people moving faster than 100 km/h and thus moving by train or by car.

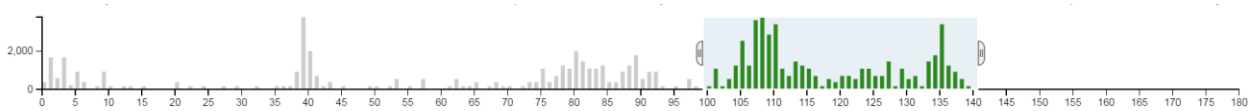


Figure 4:19 Speed slider control

4.6 Final result

Each time the users change a filter, it changes the requests sent by the Javascript on the client side to the REST API then resulting in a new result set for the visualization and also for the values shown in the controls.

On the screenshot below you can see the final results of the visualization techniques and controls. We can quickly see that there is a group of people in the train station and a flow of people on the rails. Obviously people are traveling by train.

The arrows length shows the speed and one can see that it logically increases while the train is leaving the station and is gaining speed. The number of measurements per grid cell is also decreasing because of the speed there is less time to capture measurements.

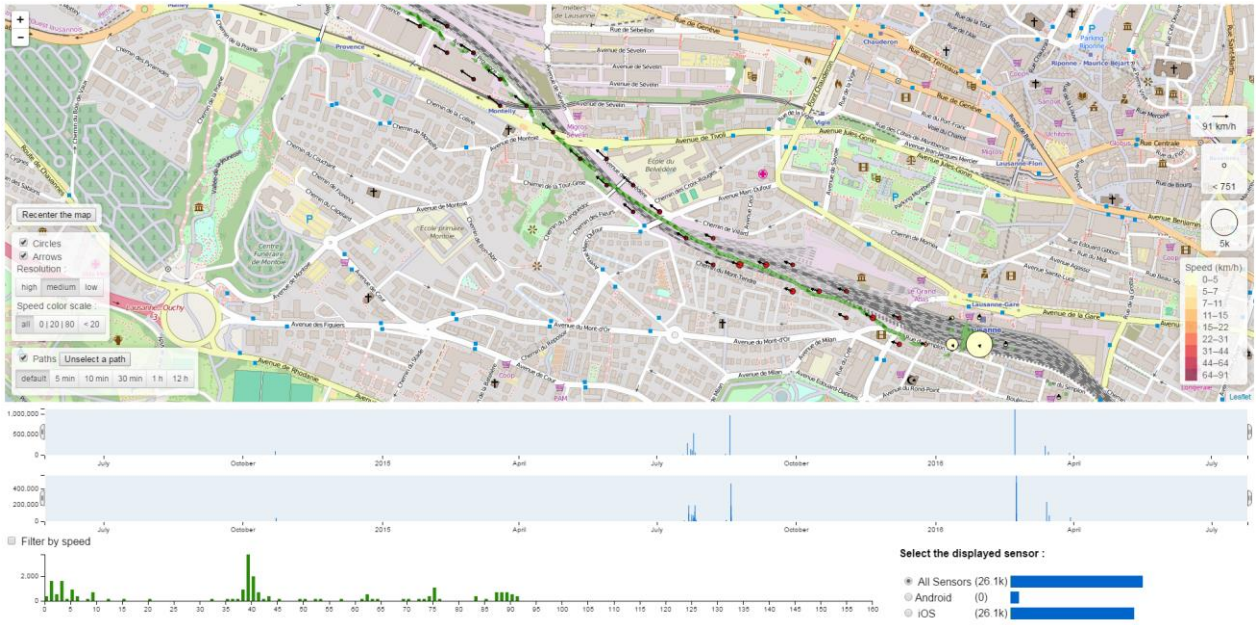


Figure 4:20 Overview of the console and controls

Chapter 5 Monitoring the database

Monitoring the state of the database is one of the tasks of the IoT engineer. He needs to know how much space on the disk represents all measurements or only those selected by the cross filters, for example the measurements made during the first day of the Paleo Festival. The operations engineer also wants to predict when the DB will be full so that he can take actions before it becomes out of service.

5.1 Disk space availability

The operations engineer needs to see the space taken by the DB on the disk. It is also interesting to know how much space the measures taken during an event, like the Paleo festival, occupy on the disk. After discussion with my thesis supervisor we decided to implement a stacked chart representing this.

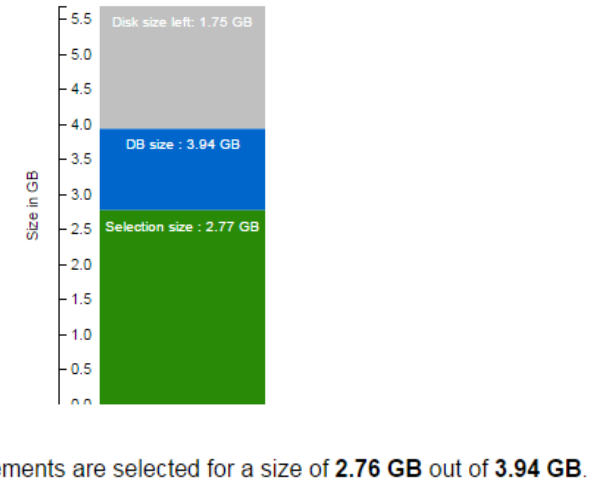


Figure 5:1 Chart of available space

5.2 Prediction

We let the user choose between different intervals of time in order to predict when the database will be full. In practice, during an event you can predict when the DB will be full supposing that the rate of measurements is the same than the one measured during a certain interval. It also informs him about the quantity of measurements and allows him to anticipate when an upgrade of the machine becomes necessary or the introduction of sharding, a distribution of the database across multiple machines. [15]

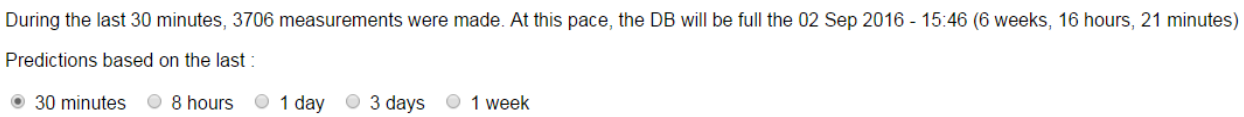


Figure 5:2 Prediction based on past measurements

Chapter 6 Implementation

This chapter describes the implementation logic on both server and client side.

6.1 Server side

6.1.1 Web server

The server web server hosts the web page that contains the HTML and the Javascript. The Javascript will only be executed on the client side once the client's browser received it. The Javascript will then interact with the REST API described below.

6.1.2 REST API

A REST API has already been implemented previously but it does not provide specific information that we need. So I decided to implement my own REST API, specifically made to give the information I need. Here are the most important endpoints:

/countsBySquare	Return the number of measurements by area in order to make a grid-based aggregation.
/directions	Similar as countsBySquare but returned average speed and bearing
/sensorCounts	Count sensor by types
/paths	Return the list of coordinates captured by each device in an area
/measurementsByDay	Return the number of measurements by day
/measurementsByHour	Return the number of measurements by hour
/mapBounds	Return the extreme coordinates in a selected area
/statistics	Return information about the database including the db size
/countSince	Returns the total counts of measurements and since the last X minutes
/getAllCoordinates	Return all the coordinates in an area
/measurementBySpeed	Return the number of measurement by each km/h
/getSensorFromCoordinates	Return the sensor from a measurement made at given coordinates
/pathFromSensor	Return the path of an individual sensor

Figure 6:1 Sample of REST API endpoints

/msmts + POST	Create a new measurement
/msmts_b + POST	Create a batch of measurements
/sensors + POST	Create a new sensor
/sensor + GET	Get a sensor by id

Figure 6:2 REST API endpoints used for measurements creation

In addition to this, parameters are specified to retrieve data selected by the filters: map location, time interval, type of sensor and speed interval.

6.2 Client side

This is the main structure of the mechanisms on the client side. Once the browser receives the Javascript from the web server, code execution can begin. The map and the different visualization controls are initialized and also the slider in order to show the data distribution in terms of speed, time and sensor type.

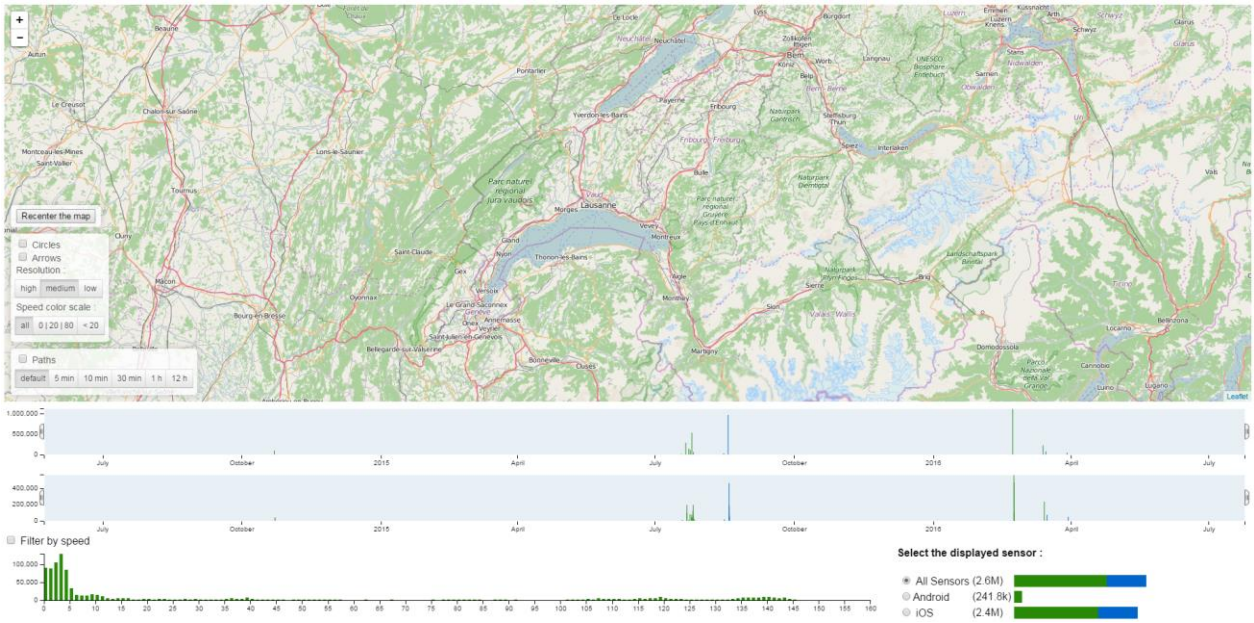


Figure 6:3 Initial state

There are three main actions:

- The user moves or zooms (in/out) on the map
- The user changes one of the filters (time, speed, sensor type)
- The user adds or removes a visualization method (circles, arrows, paths)

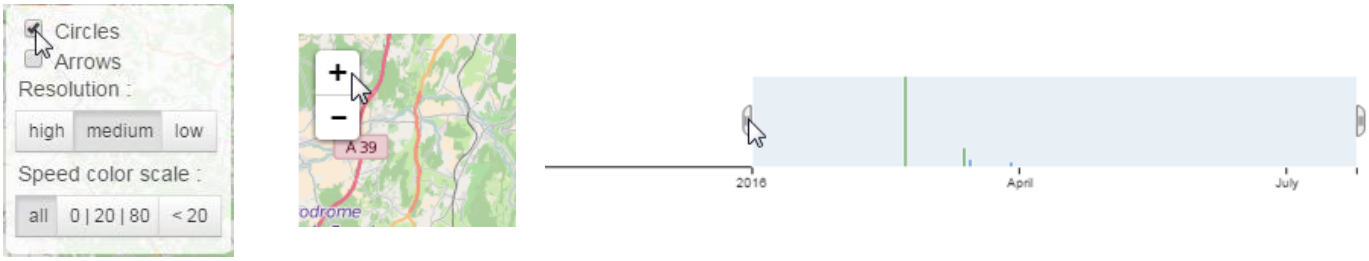


Figure 6:4 Sample of user's potential actions

Each of them triggers one or more functions used to refresh information shown on the screen. The function "refresh(action)" plays the role of dispatcher by handling what should be refreshed.

```
switch(action)
{
    case "mapMove":
        refreshAll();
        break;
    case "lastMin":
        refreshPaths();
        break;
    case "sensor":
        refreshAll("exceptSensor");
        break;
    case "date":
        refreshAll("exceptDate");
        break;
    case "speed":
        refreshAll("exceptSpeed");
        break;
    case "circlesControl":
        refreshCircles();
        break;
    case "arrowsControl":
        refreshArrows();
        break;
    case "GPSPathsControl":
        refreshPaths();
        break;
}
```

Figure 6:5 Snippet from the function "refresh(action)" where it shows the dispatcher

In order to update the different elements of the console, the REST API is called to retrieve the information asked by the user. Basically, each time the user performs an action, the filters change and this way the request is sent with new parameters to the REST API by the Javascript, which is forwarded to the database. After the database responds by the result set, it is forwarded to the Javascript that will update the html in the operations engineer's browser. This process is repeated at each engineer interaction.

Below is an example of what is refreshed when the operations engineer is zooming on the map, selecting the data from the current year and activating the visualization with circles. Those commands were previously shown on a previous figure (Figure 6:4 Sample of user's potential actions). Each of those actions has an impact on the controls' histograms that each represents a dimension.

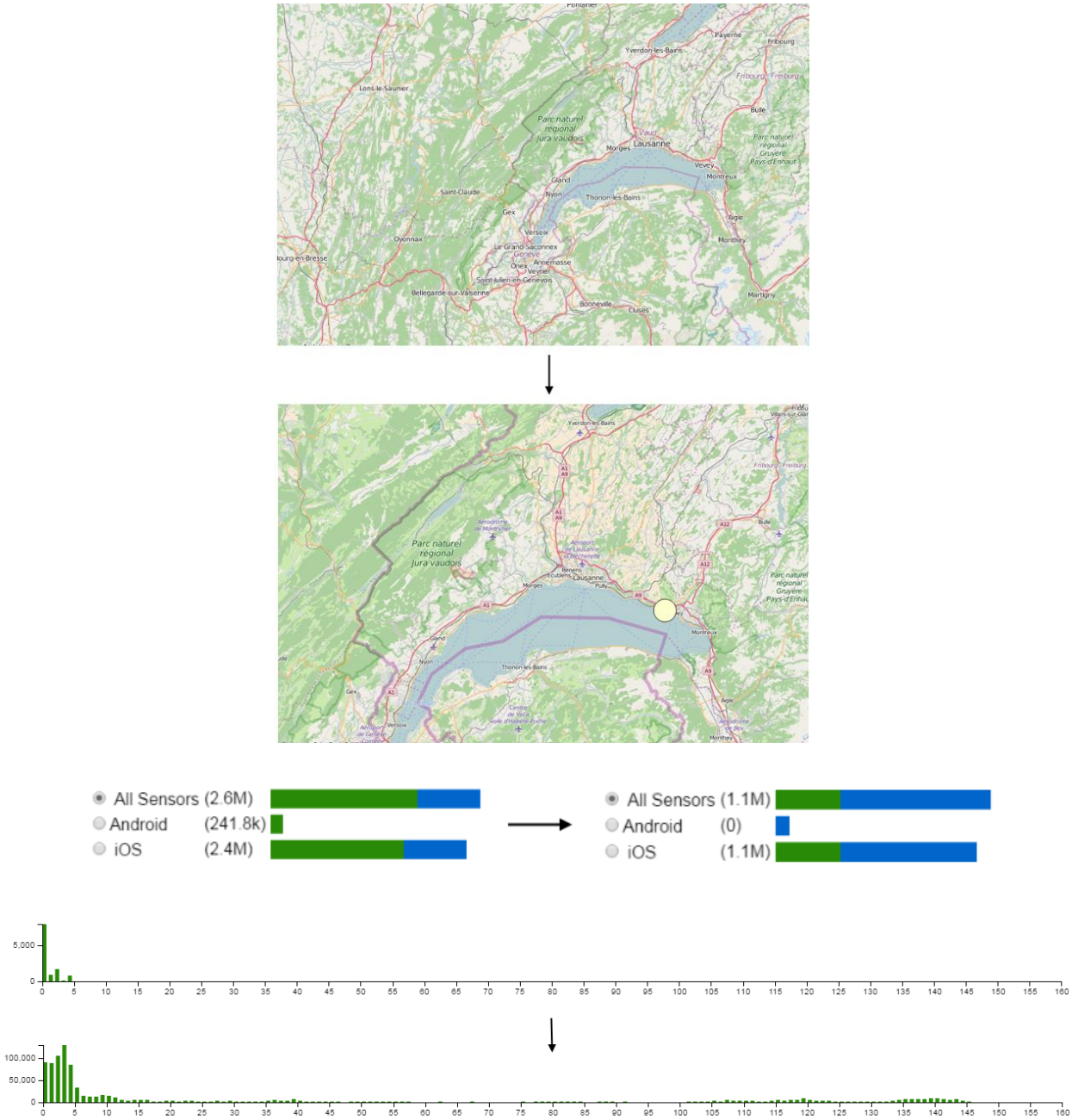


Figure 6:6 Update of controls and visualization

6.2.1 Rendering

SVG is draw on top of the leaflet map to show information. D3 library assists in this process. Below is the code to draw the circles representing the quantity of measurements in each area of the map (cf. 4.2.3 Bubble map). A recurrent way of doing with D3 is binding each element of an array containing measurements into different SVG graphical objects. [16]

```
// Create a new g element that contains the svg circles
var svg = d3.select("#mapid").select("svg");
var g = svg.append("g").attr("id", "circles");

g.selectAll("circle")
    .data(dataArray)
    .enter()
    // Bind each value of the array to a different circle
    .append("circle")
    .style("stroke", "black")
    // Set the circle's color depending on the average speed in the area
    .style("fill", function (d)
    {
            return getColorCategory(d.avgSpeed, maxSpeed)
    })
    // Set the size depending on the number of measurements
    .attr("r", function (d)
    {
        var size = d.count / maxCount * squareSizeInPixel / 2;

        // Set a minimal size
        if (size < 3)
            size = 3;

        return size;
    })
    // Move to the exact location
    .attr("transform", function (d)
        {
            return "translate(" +
                myMap.latLngToLayerPoint(d.LatLng).x + "," +
                myMap.latLngToLayerPoint(d.LatLng).y + ")";
        })
    // Add text for the mouseover
    .append("title")
    .text(function (d)
    {                                                        // m/s to km/h
        return "Count: " + d.count + " measurements\nAverage speed: " + (Number(d.avgSpeed) * 3.6).toFixed(2) + " km/h\n";
    });
```

Figure 6:7 Drawing svg with D3 on the leaflet map

The array contains an element for each cell in the grid shown the bubble map visualization. This array is obtainable from a REST API endpoint (/countsBySquare).

Latlng is the location of the circle, latGroup and lonGroup represent the grid indexation. Finally the data itself: average speed and a count of measurements.

```
▼ Array[41]
  ▼ 0: Object
    ▼ LatLng: o.LatLng
        lat: 46.458046722010835
        lng: 6.330324525987904
      ▶ __proto__: Object
    ▼ _id: Object
        latGroup: 46.4533210535415
        lonGroup: 6.323421227704384
      ▶ __proto__: Object
      avgSpeed: 39.00459433890678
      count: 6623
    ▶ __proto__: Object
  ▶ 1: Object
  ▶ 2: Object
```

Figure 6:8 Sample of array for used for svg rendering

6.2.2 Diagram

On this diagram you can find a simplification of the overall process.

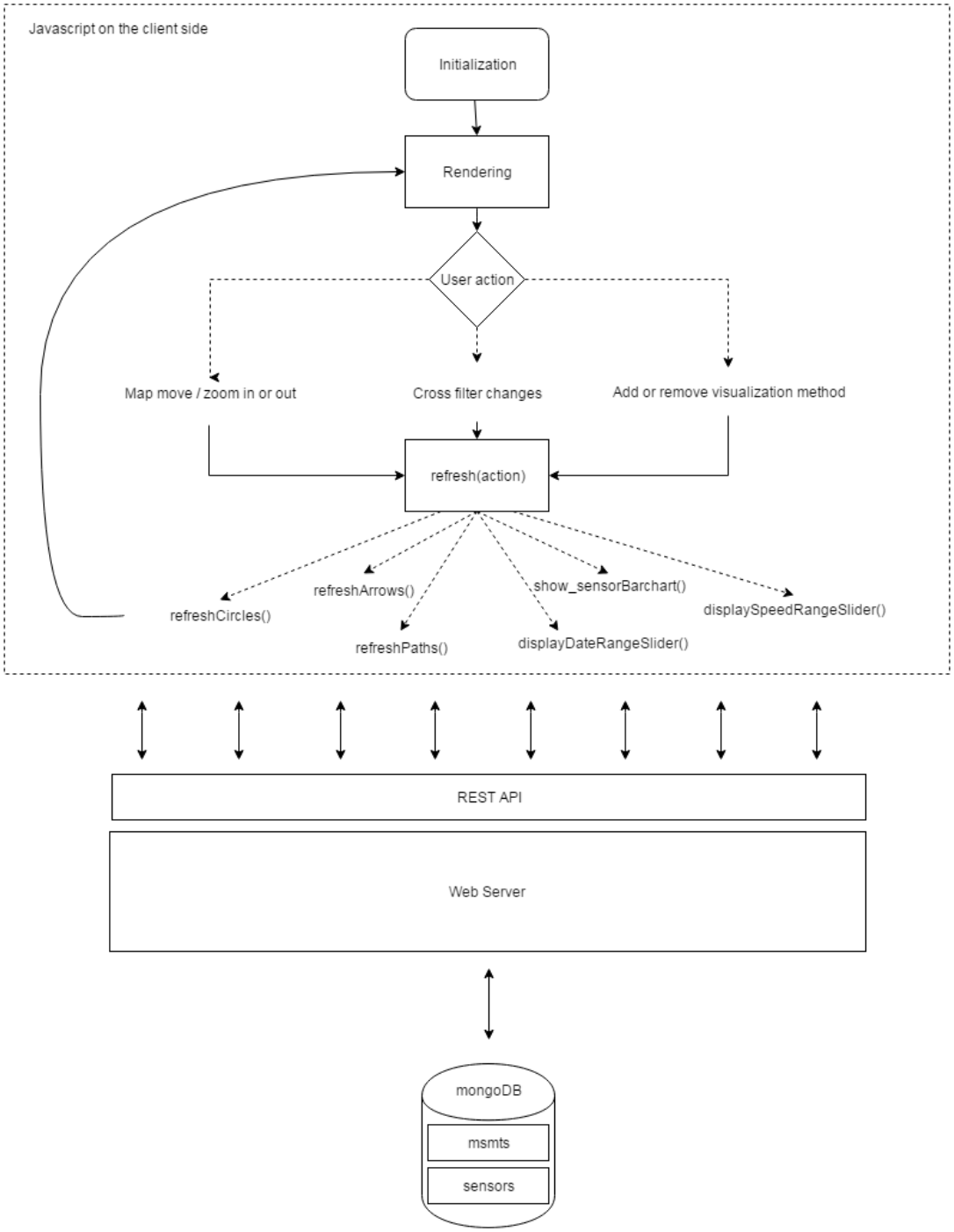


Figure 6:9 Overall application's process

The first step is the initialization of the interface elements, for example loading the map and the controls. Then when the user makes an action like changing one of the filters, the Javascript will send multiple requests, in an asynchronous manner, with the new parameters to the REST API, located on the web server. A request is sent for each control and each visualization method because they must be updated to match the new filters.

The web server will then interact with the database to get a specific subset or aggregation requested. Finally, each time the Javascript receives a reply from the REST API he can refreshed a part of the interface. This whole process repeats at each of the user's interactions with the cross filters.

Chapter 7 Performance optimization

Considering the fact that the database actually contains more than 3 million documents, performance optimization will have a crucial role in the implementation. We cannot just transfer all the data to the client.

7.1 Aggregation

Aggregation consists of regrouping elements and doing some kind of summary. In my case: measurements by hours, days, speed, sensor type, geographical area. The way aggregations are implemented can greatly influence overall performance.

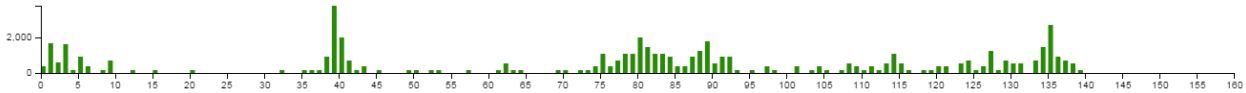


Figure 7:1 Speed aggregation

7.1.1 Purpose of aggregation

Aggregation not only helps the operations engineer to have a quick grasp on what's going on, like knowing how much measurements were made at a specific date without manually parsing and counting. It can used to save the amount of data transferred when it is done on the server side. However the server might be overwhelmed by the processing load required by such an aggregation.

The 3 million documents represent about 4 gigabytes of data. Transferring such a big load through HTTP to the client does not make sense in term when considering network load and memory and processing load for the browser. MongoDB allows to aggregate data by processing it and returning a computed result. [17]

7.1.2 Example

For example, we need to know how many measurements were made for each day of the year. Instead of requesting all measurements and processing then Javascript to group the measurements by day, mongoDB can group by day and then transfer the result of the grouping. This example might sound obvious because we do not need other information to be transferred. The data items that we need to visualize in this project are location, speed or sensor type. We could be tempted to transfer all this data in a single request and process it locally. This solution could have been viable in the case we have a much smaller amount of data like 10'000 measurements.

7.1.3 Aggregation on the server side

The obvious solution to this problem is to do all of the aggregations on the server side. However a disadvantage is that the server needs much more resources. Each time the client changes the filter for the visualization, the Javascript sends a request to the server with the new parameters. The server then needs

to formulate a request to the database to get a new aggregation according to the new parameters. It takes a lot of resources to do frequent aggregations.

7.1.4 Example of trade-off

Most aggregations in this project made on the server but compromises need to be done in order to have a balance between the server load and the client load. An example of this compromise is for the date and hour histogram slider. Initially I had to make separate requests to retrieve measurements by hours and by days. I decided to only do the request for grouping the hours and aggregate by days on the client side. This allows reducing the database load.

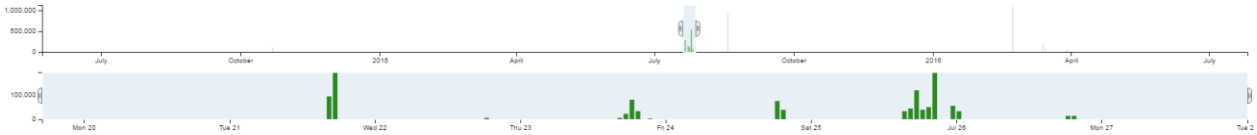


Figure 7:2 Day and hour time slider histograms use the same dataset

7.1.5 Implementation

A mongoDB aggregation consists of a pipeline made of one or several stages.

The first stage of the pipeline I am using below consists of filtering the measurements in the collections to match the specified conditions. The filters are those selected by the user (time, type, location).

The second stage groups the measurements by hours and counts them. In order to this I am rounding down the time into the nearest hour. I use modulo and subtraction. I subtract the difference between the hour and the difference between the hour and the nearest hour.

<u>Example :</u> 16:30:23 – (16:30:23 % 01:00:00) = 16:30:23 – 00:30:23 = 16:00.00

```
cursor = db.msmts.aggregate(
    [
     {
         "$match":
         {
             "t": {"$gt": tMin, "$lt": tMax},
             "nodeLoc.coordinates.1": {"$gt":minLat, "$lt": maxLat},
             "nodeLoc.coordinates.0": {"$gt": minLon, "$lt": maxLon},
             "name": regxSensorType,
             "speed": {"$gte": speedMin, "$lte": speedMax},
             "name": regxSensorType
         }
     },
     {
         "$group":
         {
             "_id": {"hourGroup":
                         {"$subtract":
                             ["$t",
                             {"$mod":
                                 [{"$subtract": ["$t", epoch]},
                                 60*60*1000]}
                             ]
                         }
                     },
             "count": {"$sum": 1},
         }
     }
    ]
)
```

Figure 7:3 mongoDB aggregate query

7.2 Indexing

Initially no indexes were used in the database apart from the default index on the id field. In order to improve performance, indexes can be created on one or more fields. With indexes the database does not need to scan the whole collection and this limits the number of document to inspect. [18]

I am creating a compound index to support queries that match on multiple fields. [19]

```
db.msmts.createIndex({t: 1, "nodeLoc.coordinates.1": 1, "nodeLoc.coordinates.0": 1, name: 1, speed: 1})
```

Figure 7:4 Example of compound index on the msmts collection

7.2.1 2dsphere Indexing performance

Previously the coordinates were stored in two fields as standard properties. It is not possible in this way to benefit from indexes that improve computation of geometries on an earth-like sphere. In order to use the 2dsphere index we need to format GPS coordinates into the standardized format [20]. This format allows the use of operators such as $geowithin that allows finding coordinates inside a specified area. It can be used to extract measurements located in a delimited geographical zone. [21]

```
"locLat" : 46.4899117, "locLon" : 6.7673238
↓
"nodeLoc" : { "type" : "Point", "coordinates" : [ 6.7673238, 46.4899117 ] }
```

Figure 7:5 GPS coordinates normalization

When I was trying to find what causes the queries to take such a long time to load, I found out that when a query uses $geowithin it takes much longer time. I make this breakthrough by removing each part of a query and by comparing the execution time; removing the geographical filter eases the database load.

I tried then to use a more rudimentary method by manually comparing if the latitude of a measurement is between a minimum and a maximum latitude (aka the top and the bottom of the current map) and same goes for the longitude.

Surprisingly I was able to reduce the query time by a factor of three. It took an average of 5.36 s instead of 15.03s. It might sound counter-intuitive. I suppose $geowithin is not optimal when the polygon is a rectangle. It might serve when you need to find coordinates inside a more complex geographical area.

I choose to keep to the format for storing GPS coordinates because it might be useful in the future to use function compatible with 2dsphere index.

7.2.2 How does index improves performance?

Do indexes actually improve performance? I ran several tests to verify this hypothesis and try to figure it out when indexes should be used.

I compare the time required to execute the query in the chapter 7.1.5 that returns the quantity of measurements for each hour with and without the index mentioned above. I used the output of the mongoDB server to obtain the execution time.

I ran three times each test and got an average of 5,36s with index and 15,77s without index. The query is about three times faster using indexes. I was able to verify what is written in the mongoDB documentation: “The $match and $sort pipeline operators can take advantage of an index when they occur at the beginning of the pipeline.” [22]

I also could infer from the citation below that the $group stage in the pipeline does not confer an advantage. I assume that a query using a mechanism other than $group should be faster to execute. On the figure above you can see on the left the query with a grouping stage and on the right multiple queries that don’t use group but give the same results. The queries on the right should be faster to execute according to my hypothesis.

```
db.msmts.aggregate([
{
    $match:
    {
        "nodeLoc.coordinates.0": {$lt: 9.34661865234375, $gt: 4.163818359375},
        t: {$lt: new Date(1469203410573), $gt: new Date(1388534400000)},
        speed:{$lte: 44.44444444444444, $gte: -1.0},
        "nodeLoc.coordinates.1": {$lt: 47.03456658359043, $gt: 46.01413132237831}
    }
},
{
    $group:
    {
        _id: {},
        minLat: {$min: {$arrayElemAt: ["$nodeLoc.coordinates", 1]}},
        maxLat: {$max: {$arrayElemAt: ["$nodeLoc.coordinates", 1]}},
        maxLon: {$max: {$arrayElemAt: ["$nodeLoc.coordinates", 0]}},
        minLon: {$min: {$arrayElemAt: ["$nodeLoc.coordinates", 0]}}
    }
}])
```

```
# find minimum latitude
db.msmts.find(
{
    "nodeLoc.coordinates.1":{ $lt: 47.03456658359043, $gt: 46.01413132237831 },
    "nodeLoc.coordinates.0": { $lt: 9.34661865234375, $gt: 4.163818359375 },
    "t": { $lt: new Date(1469203410573), $gt: new Date(1388534400000) },
    "speed": { $lte: 44.44444444444444, $gte: -1.0 }
}).sort({"nodeLoc.coordinates.0": 1}).limit(1)

# find maximum latitude
db.msmts.find(
{
    "nodeLoc.coordinates.1":{ $lt: 47.03456658359043, $gt: 46.01413132237831 },
    "nodeLoc.coordinates.0": { $lt: 9.34661865234375, $gt: 4.163818359375 },
    "t": { $lt: new Date(1469203410573), $gt: new Date(1388534400000) },
    "speed": { $lte: 44.44444444444444, $gte: -1.0 }
}).sort({"nodeLoc.coordinates.0": -1}).limit(1)

# find minimum longitude
db.msmts.find(
{
    "nodeLoc.coordinates.1":{ $lt: 47.03456658359043, $gt: 46.01413132237831 },
    "nodeLoc.coordinates.0": { $lt: 9.34661865234375, $gt: 4.163818359375 },
    "t": { $lt: new Date(1469203410573), $gt: new Date(1388534400000) },
    "speed": { $lte: 44.44444444444444, $gte: -1.0 }
}).sort({"nodeLoc.coordinates.1": 1}).limit(1)

# find maximum latitude
db.msmts.find(
{
    "nodeLoc.coordinates.1":{ $lt: 47.03456658359043, $gt: 46.01413132237831 },
    "nodeLoc.coordinates.0": { $lt: 9.34661865234375, $gt: 4.163818359375 },
    "t": { $lt: new Date(1469203410573), $gt: new Date(1388534400000) },
    "speed": { $lte: 44.44444444444444, $gte: -1.0 }
}).sort({"nodeLoc.coordinates.1": -1}).limit(1)
```

Figure 7:6 Finding min and max value in mongoDB : aggregation with $group versus multiple find request

I compare the execution time between two methods: finding min and max value for latitude and longitude in the $group phase in an aggregation pipeline and finding it in four separate find queries.

It turns out that the single aggregation query takes almost ten times longer (18,68s) than the four find requests all together (2.26s). The find queries benefit from the compound index previously created while the group stage in the aggregation pipeline is suboptimal because it cannot use the index. This confirms my hypothesis.

7.2.3 Pre-Computing in the DB

My supervisor make the assumption that adding a field in each document storing a precomputed value later used for aggregation could help performance. He suggested me this for the hour aggregation. In this case, pre-computation basically consists of the creation of a new field storing the timestamp rounded down to the nearest hour. In this way, the database just needs to group on this new field. I tried this solution but the query time approximatively remains the same.

However the pre-computation can have benefits in some cases. Typically, to find out if a string contains a string, a regular expression (regex) can be used. I was using a regex to determine if the smartphone is Android or iOS because a field contains various information about the device as shown on the figure below, including the operating system. Regex use a lot of processing time. I decided to iterate over all measurements' records and add a new field containing the mobile OS type. I also need to generate this new field each time a measurement is collected in the future. From now on, the database does not need to use a regex at each time to find out if a measurement was made from a specific device. It just needs to look if the new pre-computed field equals the specific type.

```
"name" : "Android[product=m0xx,brand=samsung,manufacturer=samsung,model=GT-I9300]"
```

↓

```
db.msmts.find({ "name" : {$exists: true}}).forEach( function(x) {

    if((x.name).indexOf("Android") > -1)
        sensorType = "Android";
    else if((x.name).indexOf("iOS") > -1)
        sensorType = "iOS";
    else
        sensorType = "other";

    x.sensorType = sensorType;

    db.msmts.save(x); });
```

↓

```
"sensorType" : "Android"
```

Figure 7:7 Pre-aggregation with sensor type

Chapter 8 Difficulties

In this chapter I will describe some more specific problems.

8.1 Data format

In the database some documents contained invalid values because they were part of a test dataset. Those documents prevent indexing and the correct computation of average values from measurements. In some case when no valid information could be extracted from the sensor, I completely removed it from the database while sometimes I just ignored it when I was able to isolate those values.

8.2 Projection problem

In order to do an aggregation by cells from a grid (cf. Figure 4:5 Representation of geographical aggregation with circles), we need to know which measurements are contained in each cell of the map. To determine this, we already know the latitude and longitude of each coordinates. However what we consider a square is defined by a dimension in pixels. The problem is that a degree of latitude in pixel is not equivalent to a degree of longitude in pixel. The reason is that longitude and latitude are spherical coordinates and a map in pixels is a projection of a sphere.

The picture bellow shows the grid formed by the longitude and latitude. Only at the crossing of the prime meridian and the equator the cells have approximatively the dimension of a square. That is the reason why we need to find a relationship between this two geographical coordinates to simulate a square on the map displayed in pixel.

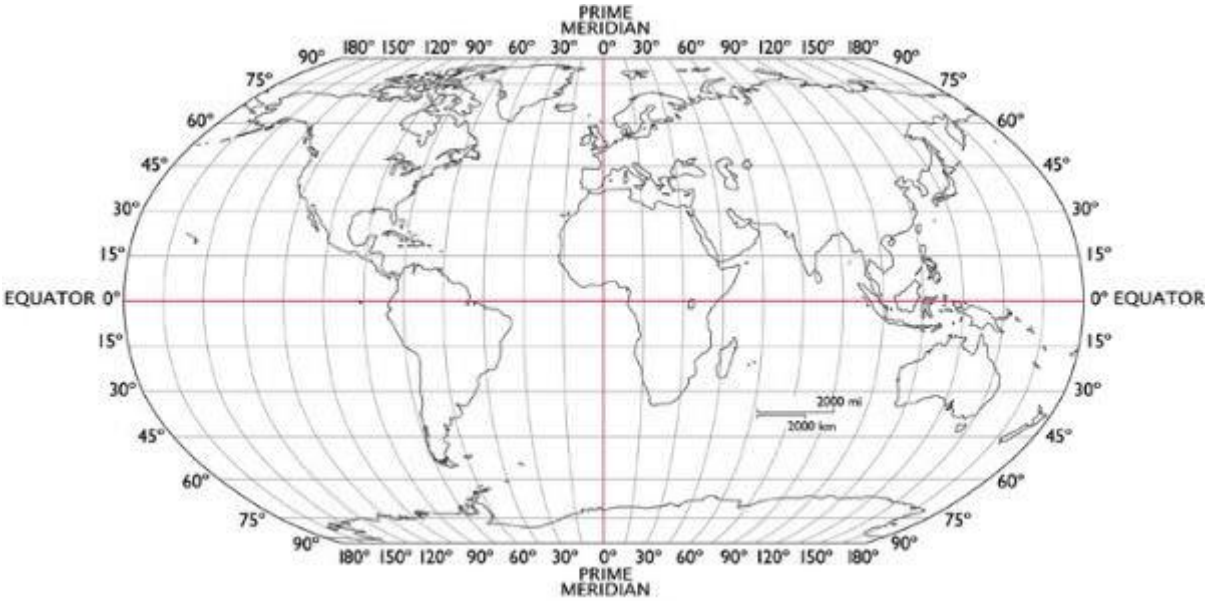


Figure 8:1 Projection of the earth showing the longitude and latitude system

The code computes the relationship between longitude and latitude. A number of vertical cells is fixed, from this it deducts the “size” in degree of latitude and finally it obtains the “size” in degree of longitude based on the computed relationship. This approximation is working when the map scale is relatively small, about the size of a country. Otherwise the relationship between longitude and latitude varies too much within the bounds of the map.

8.3 Improving user experience

8.3.1 Lost in cyberspace

When designing user interface I did not realize how confusing it can be for a new user to use it. Some elements particularly caused problem, such as controls that disappear and reappear after data refreshing is done.

The user needs landmarks when using software. We must avoid the elements that appear and disappear or move. Otherwise the user might feel lost and confused.

8.3.2 User-friendly values

My thesis supervisor draws my attention on the human readability of large values shown on the screen. For example, the number of measurements is displayed this way: 2653219. The human brain is not able to process this 7 digits numbers and will not remember it. It is more natural to him to read 2’600’000 or even better 3 M.

In order to do this I am using a Javascript library, humanize.js [23]. With a single function call, it allows formatting a number in a more human-friendly way.

2.6M out of **3.8M** measurements are selected for a size of **2.76 GB** out of **3.94 GB**.

Figure 8:2 Human-readable output with humanize.js

8.3.3 Legends

Legends are essential for the operations engineer to have an idea of the scale of values. It consists of information about the speed color scale and the size for the circles and arrows. Those legends will dynamically change as the same time as the elements on the map are changing, for example, to show the new maximum speed and quantity of measurements. A range of speed for each color is also mentioned for the speed color scale selected.

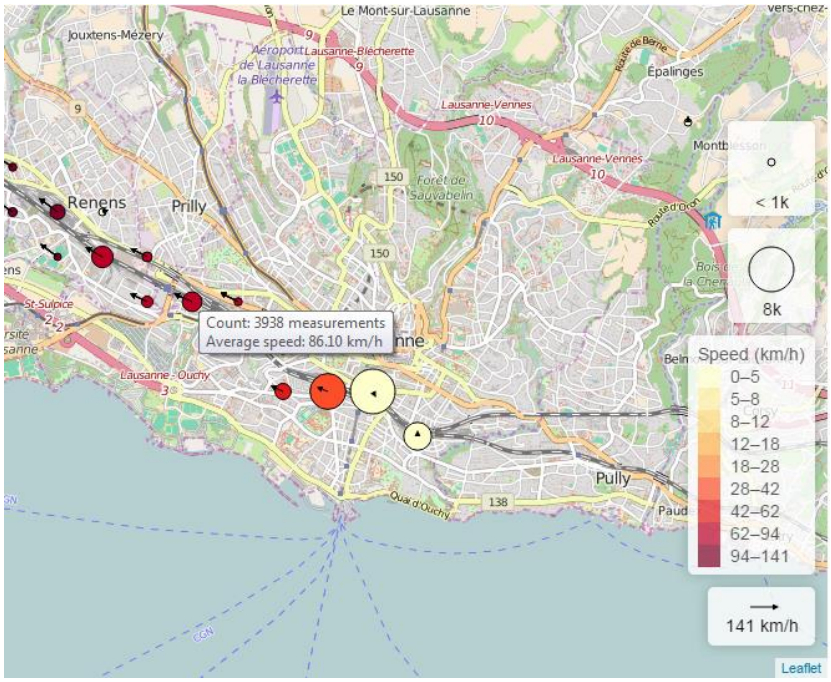
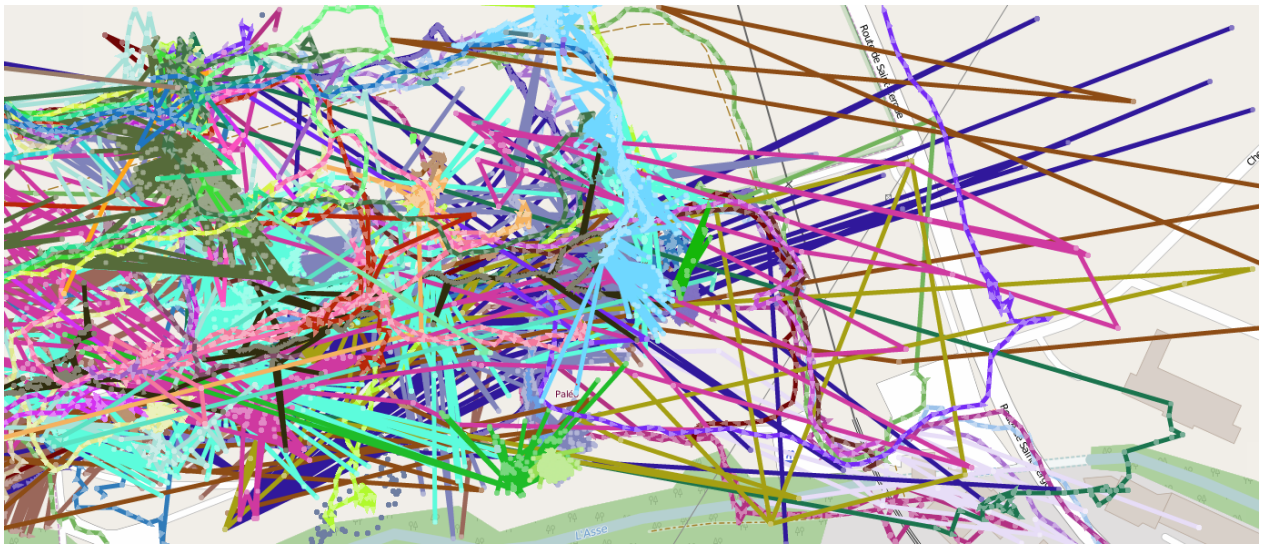


Figure 8:3 Legends and mouseover text on circles

8.4 Measurements errors

When making measurements, the equipment might measure incorrect values because of network issues or GPS dysfunction. This results in incoherent information when visualizing data. For example when showing the paths taken by the users, the GPS measurements sometimes jump about 50 meters away due to lack of signal. I tried to eliminate or at least reduce those errors by detecting a gap in terms of distance and time between two measures and removing the path linking them.

Below is a comparison before and after eliminating incorrect values. We can see that it is much easier for the operator to see a pattern.



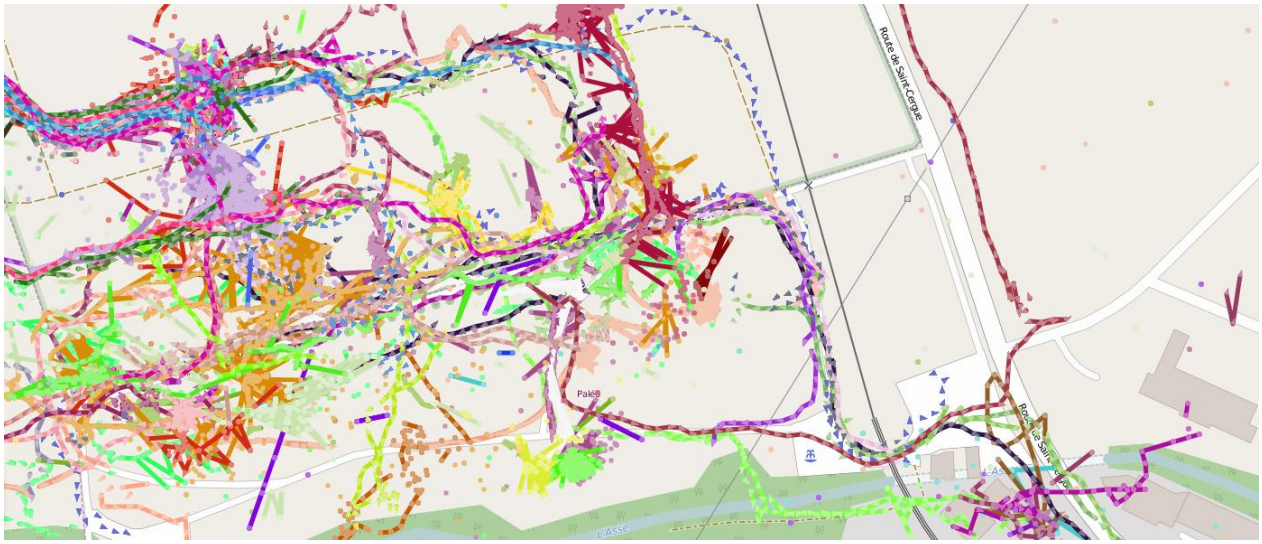


Figure 8:4 Before and after fixing path issues

However there are sometimes “false positives”, for example when the user is moving really fast there is a huge gap of distance between two measurements and it can be falsely interpreted as incorrect measure.

Chapter 9 Tests

The database already contained about 3 million records when I started to work on this project. This allows to directly testing the application with real data values. The compatibility of the application will also be described in this chapter.

9.1 Testing with pre-recorded real-world sensor data

The database was initially created for the iNuit project [3]. It was used to collect data from the Paleo Festival in 2015. A mobile app developed by Mr. Rebetez allowed collecting data from event participants who volunteer to use this app. Other measurements were also made outside of the festival in the western part of Switzerland for other tests.

The pre-existing database is a good source for testing. I was able to monitor what happened during the Paleo festival. We can see where the scene is located because there are much more measurements generated and the average speed is low. There is also no particular movement because people move in all directions and thus cancel each other which results in no speed or particular direction.

Figure 9:1 Exploring data captured during the Paleo Festival event

9.2 Generating measurements

I used the app developed by Mr. Rebetez to send measurements from my phone. I changed the source code in order to redirect the measurements made from my phone directly to the new REST API (6.1.2) allowing populating the database from the webserver.

Below you can see my movement. I did a small walk in the forest. You can notice the direction I was heading and the difference of speed with the arrows when I started running, on the bottom of the map.

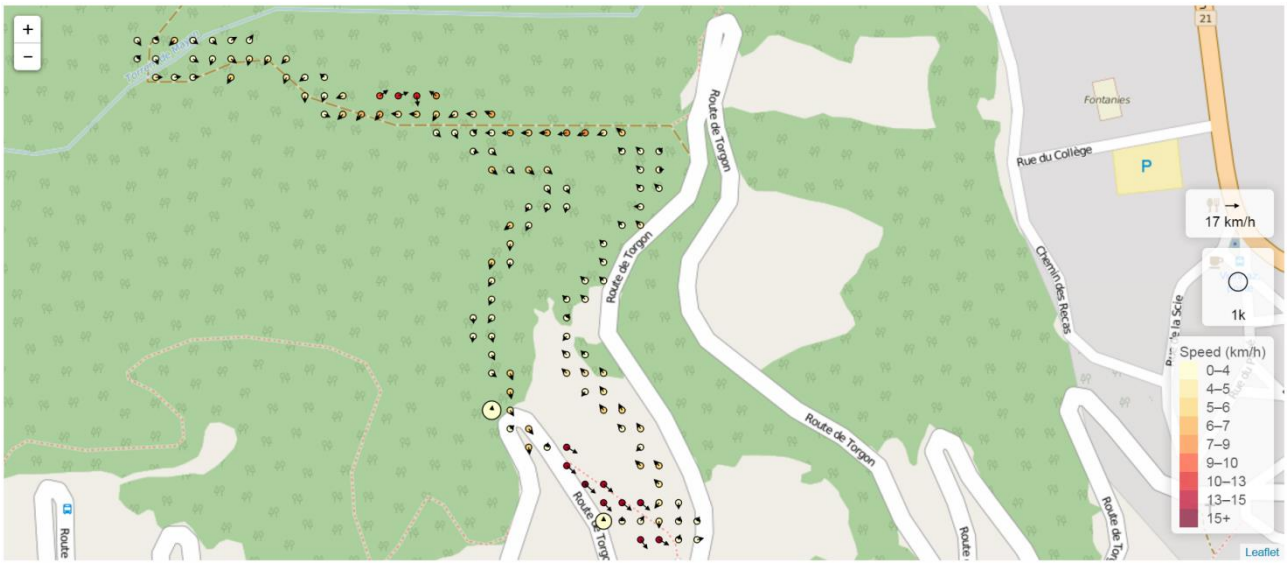


Figure 9:2 Measurements generated with my phone during a walk

9.3 Benchmark between browsers

I load the first page of the console once and I compare the time between each browser. The controllers are loading data such as the measurements by hour and by speed and interacting with the REST API which take some time because the mongoDB is heavily solicited.

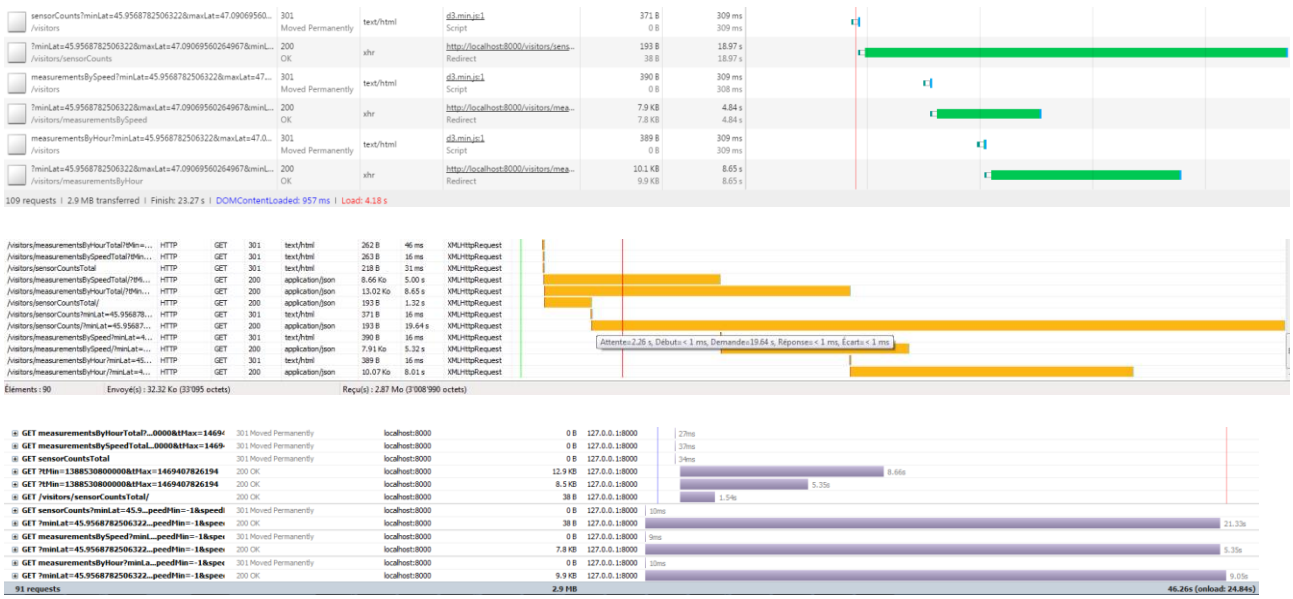


Figure 9:3 Respectively : Chrome, Internet Explorer and Firefox performance.

Chrome took 23.27 s, Internet Explorer took 21.90s, Firefox took 24.84 sec.

9.4 Browsers compatibility

Everything that I have described in this document was made to work on Google Chrome. I have also tested on Firefox and did not notice major differences except cosmetic ones. However, on Internet Explorer 11 there are some bugs such as the sensor type bar that does not correctly show information. It likely comes from a CSS property not supported by IE 11. I did not spend time fixing compatibility between browsers and focused on Chrome during the project.

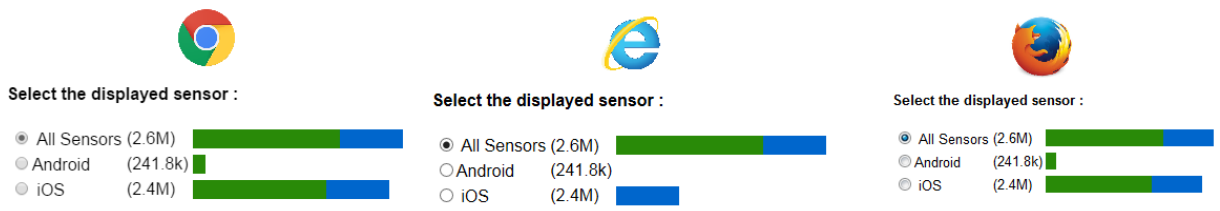


Figure 9:4 Incompatibility with IE 11

9.4.1 Responsiveness

The console has not been designed for smaller screen and resolution size. I tested it on my smartphone, an Android phone with a resolution of 1080x1920. The phone is able to display the entire interface but the size of controls and visualization are not adjusted for a comfortable use. I doubt that a phone with a smaller resolution can handle it.

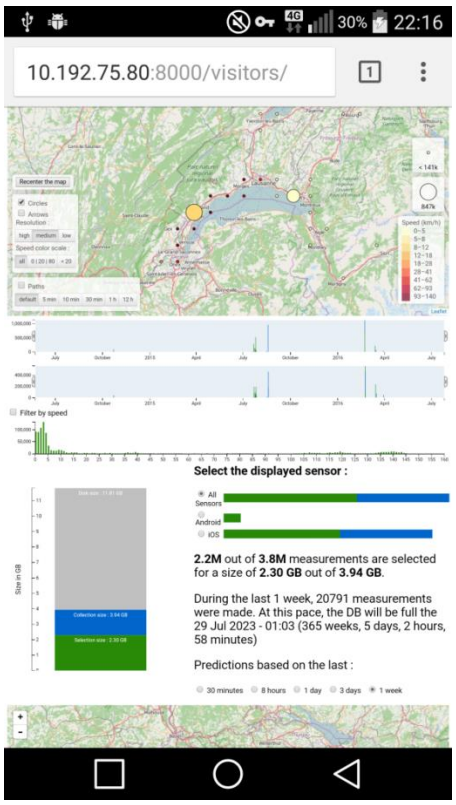


Figure 9:5 Console on a smartphone (LG G3)

Chapter 10 Project progress

During the first sixteen weeks, two days by week are dedicated to work on the project. After that I will work six complete weeks. In total it represents about 62 days. I will first describe week by week my progress on the project during the semester and then twice a week during the summer. Redacting the report is not included in this description because it was a constant task.

10.1 Week 1 – 22.02-28.02

During the first week I learned the basics of Python and start to study about Django, the web framework for Python.

10.2 Week 2 – 29.02-06.03

I continued learning Django. I also start exploring the javascript library D3 by doing some experimentation.

10.3 Week 3 – 07.03-13.03

I made a small project which combines Django, D3 and some data from MongoDB

10.4 Week 4 – 14.03-20.03

After advice from M. Graf I read the book “Interactive Data Visualization for the Web”, O’Reilly edition. It helped me understand D3 in a more detailed way

10.5 Week 5 – 21.03-27.03

I explored the concept of cross filters and applied it on some data with D3 library in javascript. I also explored GeoMapping with Geojson in D3.

10.6 Week 6 – 04.04-10.04

I compared two solutions of GeoMapping: TopoJSON file projected with D3 and Leaflet, allowing interactive map similar to Google Maps.

10.7 Week 7 – 11.04-17.04

I started doing a draft of REST API on the server side in order to retrieve data from MongoDB. I also experiment combining TopoJSON and Leaflet.

10.8 Week 8 – 18.04-24.04

I was able to cluster data by square area in the map and draw circle in each of these area representing the amount of data.

10.9 Week 9 – 25.04-01.05

I worked on drawing a path that each of devices took in an area. I mistakenly compute the average speed of each area while this is already stored in the database.

10.10 Week 10 – 02.05-08.05

I worked on displaying arrows that represents the speed and the direction a user was heading when moving with his device. It is possible to choose an interval of days with a simple JQuery slider.

10.11 Week 11 – 09.05-15.05

I improved the slider system by using dc.js a library working in pair with d3 which can show at the same data in the slider in a histogram.

10.12 Week 12 – 16.05-22.05

I developed a functionality that allows the user to “reset” the maps by focusing it on the data. The interface now showed information when there is no data to show.

10.13 Week 13 – 23.05-29.05

The histogram and the sensor barchart now showed unselected data. I encountered some difficulty when migrating to the new DB that contains new format for the GPS Data.

10.14 Week 14 – 30.05-05.06

I fixed the bugs relative to the DB migration and restructured my code in order to have a better control on it.

10.15 Week 15 – 06.06-12.06

I focused on the interim report because the deadline was approaching.

10.16 Week 16 – 13.06-17.06

I finalized the report.

10.17 17.06

Interim report deadline

10.18 Week 17 – 20.06-26.06

I deployed the webserver on a distant machine. A graphic shows how much data the database uses and shows what amount of data is selected by the filters. I added REST interface to add measurements in the database. I optimized the time sliders on both server and client side.

10.19 Week 18 – 27.06 – 03.07

I optimized the process of determining if a measurement is inside a geographical area in the mongoDB queries. I fixed a bug with the registration of the sensor when sending measures to the server. I use a logarithmic scale for the color of the circles that represents the speed in an area.

10.20 Week 19 – 04.07 – 10.07

It is now possible to choose between three sizes for the aggregation of the measures and there are three different scales for the circles' colors. I am predicting when the database will be full according to the actual pace. I improved the path functionality by removing inconsistent gap between two measurements. I added legends for the visualizations.

10.21 Week 20 – 11.07 – 17.07

I improved the interface by moving some controls directly on the map area. There is now a slider allowing choosing a range of speed and showing the distribution of it. I improved the "path" functionality by showing the direction the user is heading. I added another map to showing a heat map and refreshing automatically.

10.22 Week 21 – 18.07-24.07

I ran benchmark and tests between different browsers and documented it. I fixed some bugs with the interface. I added tuning for the heat map. I almost finished redacting my report.

10.23 Week 22 – 25.07-29.07

I added made some changes to my reports and add guidance to my interfaces, after discussions with my supervisors. I reviewed my rapport and print it.

10.24 29.07

Final report deadline

Chapter 11 Conclusion

11.1 Achieved Results

I am satisfied with what has been accomplished. Several functionalities have been implemented. The operator is now able to explore data and infer some patterns and interesting behaviors from the measurements allowing drawing conclusions. The IoT operator has now a wide variety of visualization tools. He is also now able to monitor the database utilization.

11.2 Future development

There are no limits in terms of visualization methods. This is a research field in itself. More complexed behaviors patterns might be discovered by using further sophisticated visualization methods.

This console could be used to analyze by what means people go to a festival and determine specific needs in order to improve transport systems and avoid traffic jams. Giving real time information to the festival participants about the traffic could also help in that process. Optimal itinerary could be suggested to them as well as available car park zones.

Furthermore, security could be improved by preventing crowd movements through forecasting based on the latest measurements and sending warnings to the users. In case of incident, the most appropriate emergency exits could be notified to allow safe and efficient evacuation.

On top of this, sharding methods must also be studied because the database could have an exponential growth and would need more than one physical machine in order to support a growing load.

Finally, privacy issues have not been discussed because it was not part of my requirements specification. These issues should be taken into account in further studies. Indeed, in the current state it is possible to follow a participant to his house. This could be avoided by disabling data collection outside of specific geographic areas.

List of Figures

Figure 2:1 Collection of data 3
Figure 2:2 Physical diagram 5
Figure 2:3 Projection of swiss municipalities from a GeoJSON file 6
Figure 2:4 Example of Leaflet map with a marker 7
Figure 3:1 Example of MongoDB document [6] 8
Figure 3:2 Example of MongoDB query made with pyMongo [8] 8
Figure 3:3 Iterating over results in a document [8] 8
Figure 3:4 Sample of Django routing configuration 9
Figure 3:5 Example of D3 code to make a simple bar chart [9] 9
Figure 3:6 Rudimental barchart constructed with D3 [9] 9
Figure 3:7 Example of dc chart [10] 10
Figure 3:8 Leaflet map with marker, circle and polygon [11] 10
Figure 3:9 Heat map with Leaflet-heat.js 10
Figure 4:1 Measurement sample in the mongoDB database 11
Figure 4:2 Measurement fields in the database 12
Figure 4:3 Using leaflet markers on a map 12
Figure 4:4 Aggregation using clustering [13] 13
Figure 4:5 Representation of geographical aggregation with circles 14
Figure 4:6 Example from Mike Bostock, creator of D3 and inspiration for the bubble map [14] 14
Figure 4:7 Aggregation of with Arrows on the map to show speed and direction. 15
Figure 4:8 Combination of visualization methods 16
Figure 4:9 Vizualisation of paths taken by event participants during two days in the Paléo Festival 17
Figure 4:10 Path of a user arriving by train at the Paleo Festival 17
Figure 4:11 Heat map of the Paleo festival 18

Figure 4:12 Visualization options and tunings 18
Figure 4:13 Speed color scales 19
Figure 4:14 Circles resolution 19
Figure 4:15 Date slider control 20
Figure 4:16 Hour slider control 20
Figure 4:17 Sensor controls 20
Figure 4:18 Repercussion of crossfiltering on other control 20
Figure 4:19 Speed slider control 21
Figure 4:20 Overview of the console and controls 21
Figure 5:1 Chart of available space 22
Figure 5:2 Prediction based on past measurements 22
Figure 6:1 Sample of REST API endpoints 24
Figure 6:2 REST API endpoints used for measurements creation 24
Figure 6:3 Initial state 24
Figure 6:4 Sample of user's potential actions 25
Figure 6:5 Snippet from the function "refresh(action)" where it shows the dispatcher 25
Figure 6:6 Update of controls and visualization 26
Figure 6:7 Drawing svg with D3 on the leaflet map 27
Figure 6:8 Sample of array for used for svg rendering 27
Figure 6:9 Overall application's process 28
Figure 7:1 Speed aggregation 30
Figure 7:2 Day and hour time slider histograms use the same dataset 31
Figure 7:3 mongoDB aggregate query 32
Figure 7:4 Example of compound index on the msmts collection 32
Figure 7:5 GPS coordinates normalization 33
Figure 7:6 Finding min and max value in mongoDB : aggregation with $group versus multiple find request 34
Figure 7:7 Pre-aggregation with sensor type 35
Figure 8:1 Projection of the earth showing the longitude and latitude system 36
Figure 8:2 Human-readable output with humanize.js 37
Figure 8:3 Legends and mouseover text on circles 38
Figure 8:4 Before and after fixing path issues 39
Figure 9:1 Exploring data captured during the Paleo Festival event 40

Figure 9:2 Measurements generated with my phone during a walk..... 41

Figure 9:3 Respectively : Chrome, Internet Explorer and Firefox performance. 41

Figure 9:4 Incompatibility with IE 11 42

Figure 9:5 Console on a smartphone (LG G3) 42

References

[1] Wikipedia, «Internet of things,» *https://en.wikipedia.org/wiki/Internet_of_things,* 17.06.2016.

[2] Intel, «IoT Insights with Intel's Brian Krzanich Video,» *http://www.intel.com/content/www/us/en/internet-of-things/videos/iot-insights-2015-short-keynote-video.html,* 17.06.2016.

[3] HES-SO, «Programme iNUIT,» *https://www.youtube.com/watch?v=pm878kHFAwM,* 10.03.2016.

[4] M. Bostock, «TopoJSON,» *https://github.com/mbostock/topojson,* 22.07.2016.

[5] GeoJSON, «GeoJSON,» *http://geojson.org/,* 27.07.2016.

[6] mongoDB, «Sharding,» *https://docs.mongodb.com/manual/sharding/,* 21.07.2016.

[7] mongoDB, «A mongoDB document,» *https://docs.mongodb.com/v3.0/_images/crud-annotated-document.png,* 17.06.2016.

[8] Wikipedia, «Python (programming language),» *https://en.wikipedia.org/wiki/Python_(programming_language),* 17.06.2016.

[9] mongoDB, «Find or Query Data with PyMongo,» *https://docs.mongodb.com/getting-started/python/query/,* 17.06.2016.

[10] S. Murray, Interactive Data Visualization, O'Reilly, 2013.

[11] dc.js, «Dimensional Charting Javascript Library,» *https://dc-js.github.io/dc.js/,* 21.07.2016.

[12] Leaflet, «Leaflet Quick Start Guide,» *http://leafletjs.com/examples/quick-start.html,* 17.06.2016.

[13] wikipedia, «Universally unique identifier,» *https://en.wikipedia.org/wiki/Universally_unique_identifier,* 27.07.1026.

[14] Leaflet, «Marker Clustering plugin for Leaflet,» *https://github.com/Leaflet/Leaflet.markercluster,* 17.06.2016.

[15] M. Bostock, «Let’s Make a Bubble Map,» *https://bost.ocks.org/mike/bubble-map/,* 06.05.2014.

[16] S. Murray, «Binding data,» *http://alignedleft.com/tutorials/d3/binding-data,* 15 9 15.09.2016.

[17] mongoDB, «Aggregation,» *https://docs.mongodb.com/manual/aggregation/,* 23.07.3016.

[18] mongoDB, «Indexes,» *https://docs.mongodb.com/manual/indexes/,* 23.07.2016.

[19] mongoDB, «Compound Indexes,» *https://docs.mongodb.com/manual/core/index-compound/,* 23.07.2016.

[20] mongoDB, «2dsphere Indexes,» *https://docs.mongodb.com/manual/core/2dsphere/,* 24.07.2016.

[21] mongoDB, «$geowithin,» *https://docs.mongodb.com/manual/reference/operator/query/geoWithin/,* 24.07.2016.

[22] mongoDB, «Aggregation Pipeline,» *https://docs.mongodb.com/manual/core/aggregation-pipeline/#pipeline-operators-and-indexes,* 23.07.2016.

[23] Hubspot, «Humanize Plus,» *https://github.com/HubSpot/humanize,* 24.07.2016.

Annexes

Chapter 1 Annex 1: Installation manual

1.1 Introduction

This chapter explains how to deploy the webserver and his REST API. This procedure was tested on Ubuntu 14.04.4 LTS.

1.2 Requirements

A Linux distribution is recommended to run the code, however Windows 7 is also compatible.

Below are the tools required to run the source code. Django and the other python packages must be installed on python 3, not Python 2.

- Python 3.4.3 or higher (https://www.python.org/downloads/)
- pip (recommended python package manager (https://pip.pypa.io/en/stable/installing)
- Django 1.9.2 or higher(https://docs.djangoproject.com/en/1.9/intro/install/)
- Python packages (https://docs.python.org/3/installing/)
    - geopy
    - django-cors-headers
    - pymongo
- MongoDB with a database that contains two collections (msmts and sensors)
- Google Chrome

1.3 Steps

1.3.1 Retrieving the code

The code can be fetched via a git repo: https://github.com/beedle-/IoTDashboard

1.3.2 Running the code

Launch this command to launch the webserver (and the REST API at the same time):

```
python3 manage.py runserver <current ip>:8000 &
```

Replace <current ip> with the ip used by the webserver, not the loopback address.

If there are error about missing python packages, install them following the instructions on this website: https://docs.python.org/3/installing/

1.3.3 Accessing the console

The console is now accessible at this address:

```
http://<server_address>:8000/visitors/
```

In the next chapter I will describe how to use the console.

Chapter 2 Annex 2: Operation manual

2.1 Introduction

This manual is made to guide the IoT operations engineers using the console.

2.2 Requirements

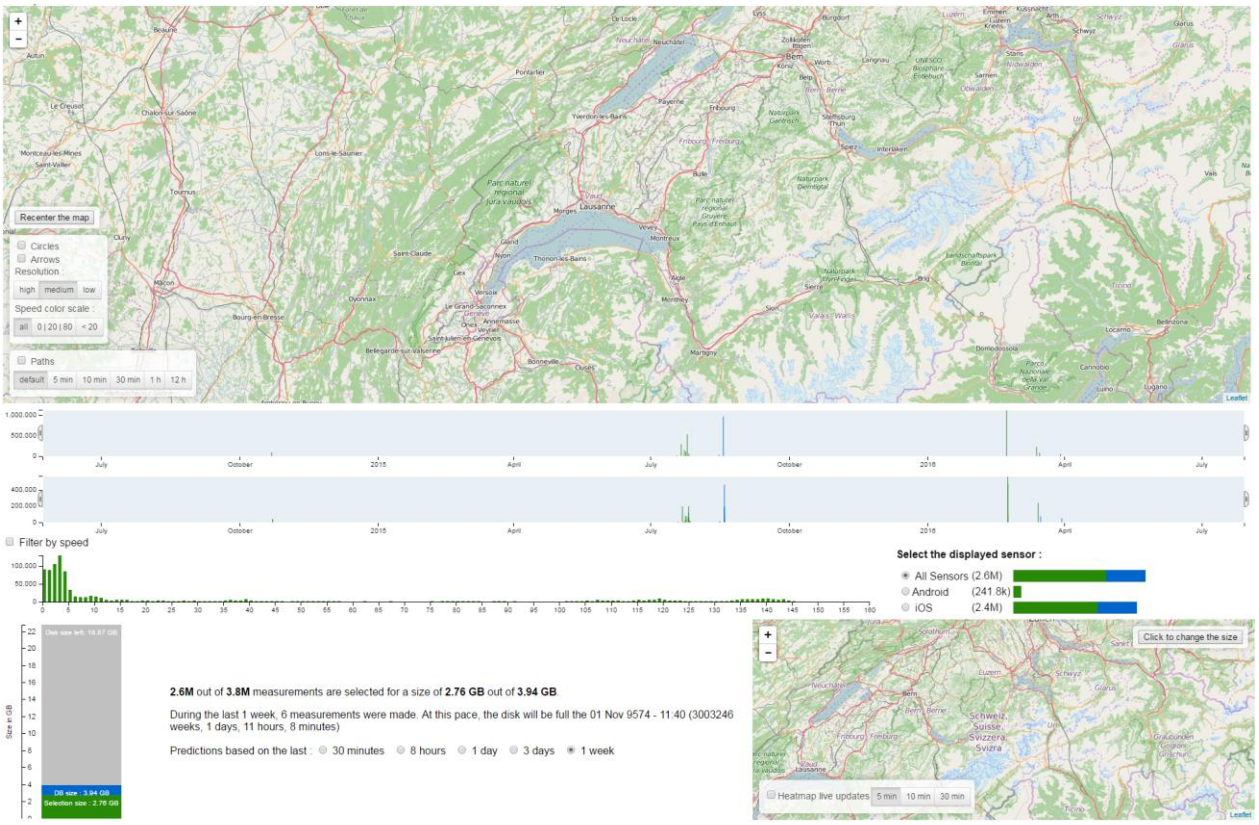
In order to use the console, only a web browser is required. Google Chrome is recommended. A large screen is also recommended for a better user experience.

2.3 Accessing the console

The console is accessible at this address:

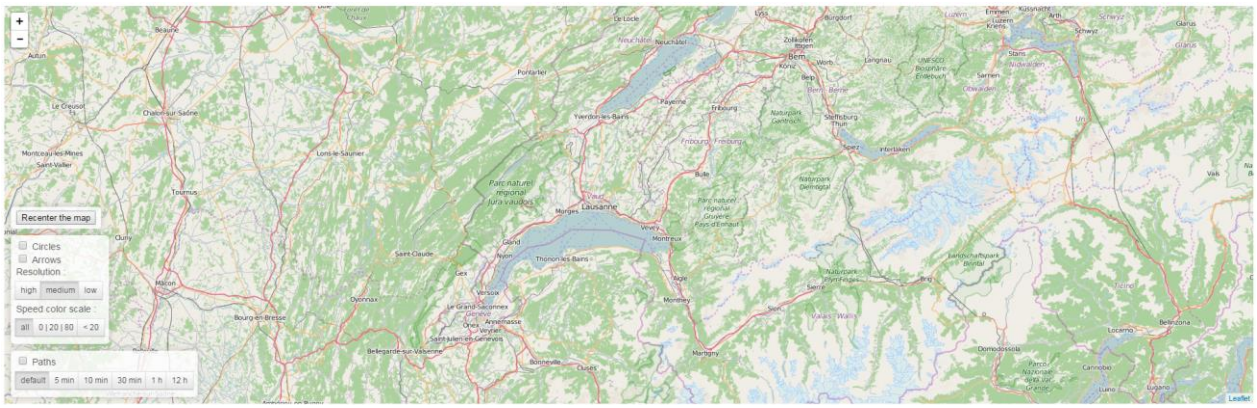
```
http://<server_address>:8000/visitors/
```

When accessing this addres, a map will automatically zoom on the measurements made in Switzerland. In this case below, measurements were only made in western Switzerland. Several controls are displayed under the map: time control, speed control and sensor type control. There is also information about the database state. Finally another smaller map is on the bottom right and can be used to display a heat map. I will describe each of those parts in the next chapters.



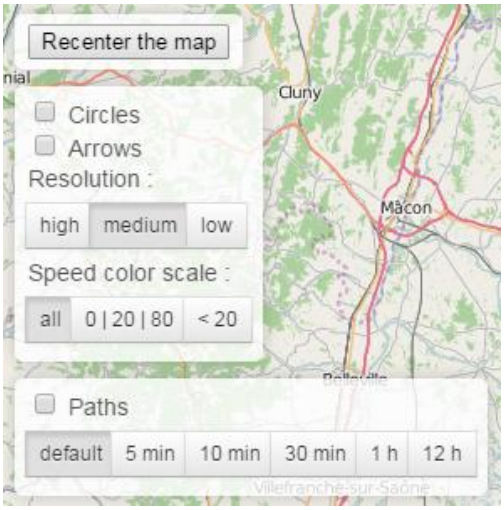
2.4 Using the map

The main control/display is the map. It is a classic map like google maps that allows panning and zooming in and out as you want.



2.5 Choosing the visualization method

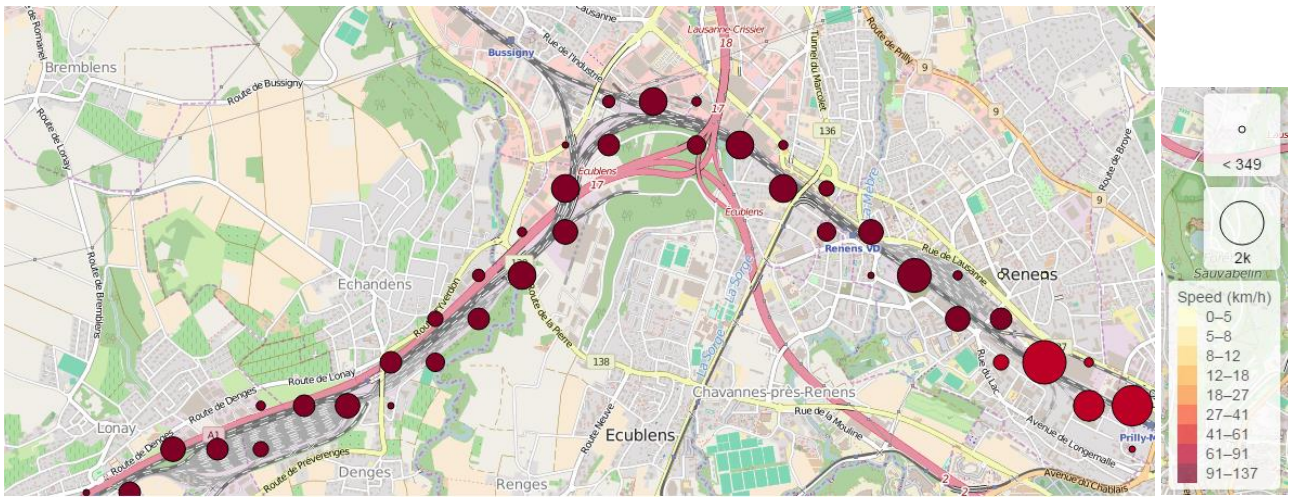
It is possible to choose between multiple visualization methods using these controls:



When checking any of those checkboxes (Circles, Arrows and Path) visualization will be displayed.

2.5.1 Circles

This visualization method aggregates the measurements by area. The radius of each circle depends on the number of measurements. A legend on the right of the map indicates the maximum number of measurements in one area. The circles colors show the average speed; the color scale is described on the right of the map.



This is the default color scale. However it is possible to choose between two others color scales : 0 | 20 | 80 and < 20. The firt one consists of three categories : one from 0 t o20 km/h, that consists of pedestrians. Then from 20 to 80 km/h, for the people moving by car or transport. Finally, speeds higher than 80km/h for the train or the highway. The last scale is made of 9 categories from 0 to 20km/h to only focus on pedestrians.

2.5.2 Arrows

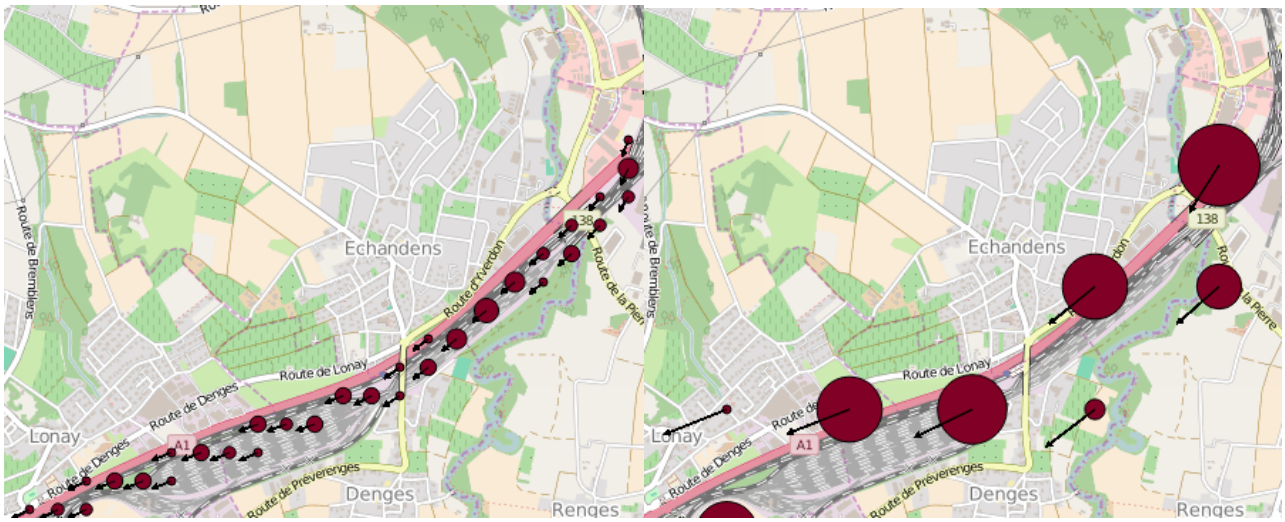
The arrows represent the average speed and direction for the same area than the circles. The legends show the highest speed.

2.5.3 Both

It is of course possible to combine those two methods :

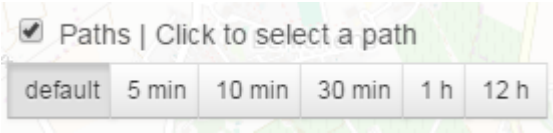
2.5.4 Resolution

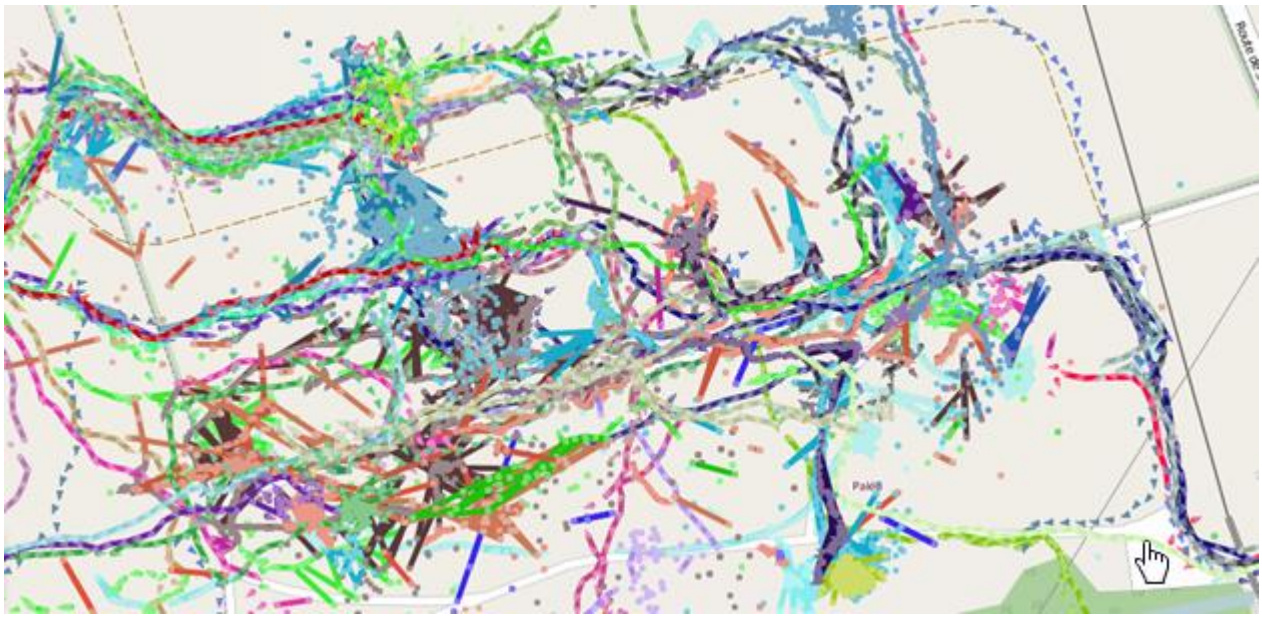
A higher or lower resolution can be choose to display less or more circles and/or arrows. Below, on the left is a higher resolution, and lower on the right.



2.5.5 Paths

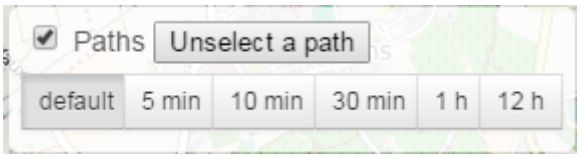
Selecting the “Paths” option will display every movement of every festival participant in the area. Each color is a different participant.





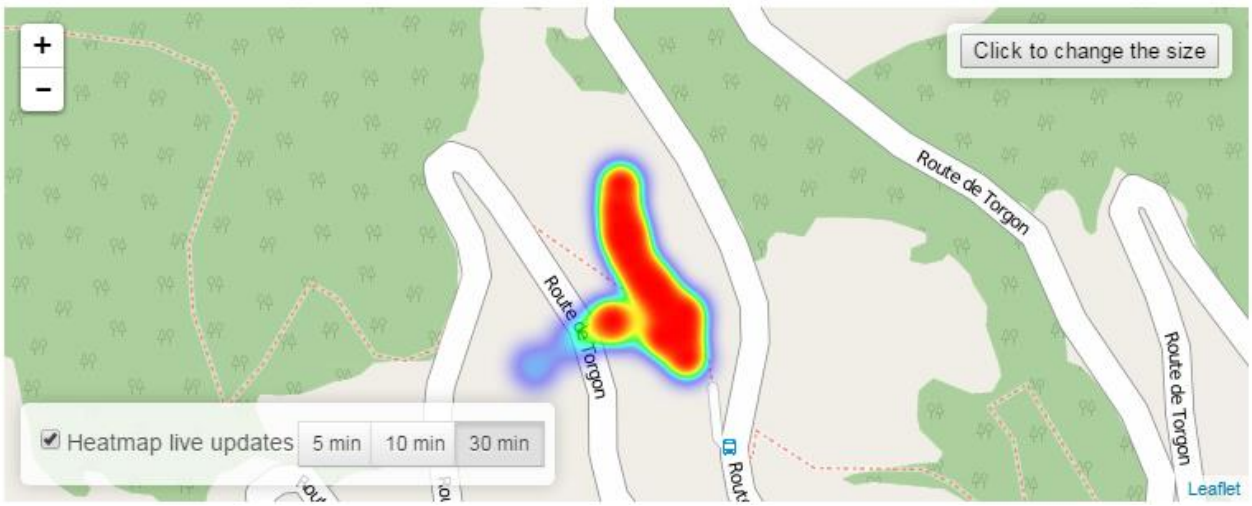
You can then click on of the path to follow a participant further. When moving the map, only its path will be displayed.

You can then click on “Unselect a path” to show again all paths.



2.6 Displaying the heatmap

On the bottom right of the page, you can see a smaller map that can display a heat map. The heat map shows the intensity of measurements by area. When the color is close to red it means that there is a higher density of measurements.



You can choose to display a heat map for the last 5, 10 or 30 min. It is also possible to make the map bigger by clicking on “Click to change the size”. The heat map is not affected by the crossfilter. It is independent.

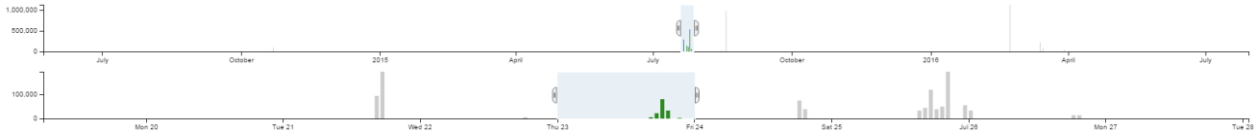
Clicking a again on “Click to change the size” will reduce the map size.

2.7 Controls

In order to narrow down to specific measurements visualization there are several controls available. They act as cross filters, meaning that interacting with one of them will affect the others. These interactions will be described below.

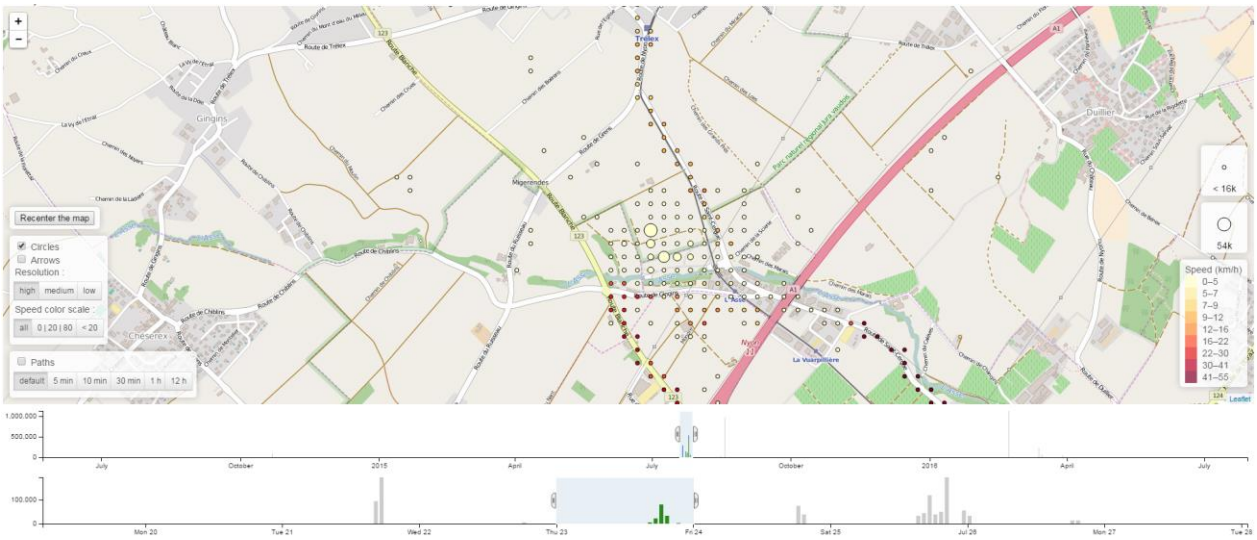
2.7.1 Time controls

Two sliders allow choosing a time interval, the first one is used to select a range of days and the second allow a fined-grained selection, by hours for example. The sliders also display a distribution of the measurements. When the bar is green it means that the measurements were made in the actual selection (geographical area for example).



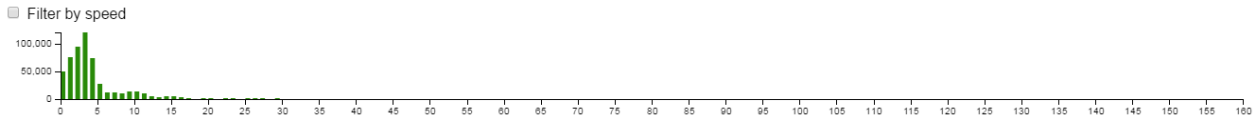
If you the map is on Lausanne you will not see any measurements during this time interval because only in the Paleo festival there was measurements. The colors on the histograms show this.

Now you can see the measurements made on Thursday the 23rd July 2015 during the Paleo festival.



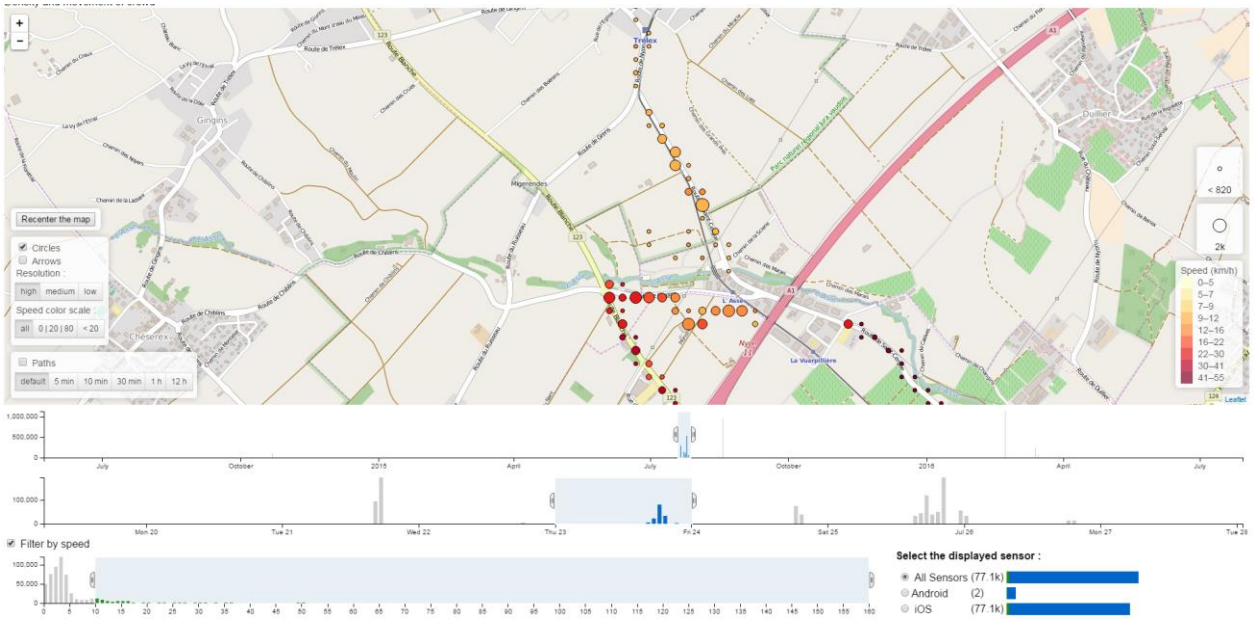
2.7.2 Speed controls

The speed controls allow displaying measurements that have a particular speed. It also displays an histogram like the time sliders. You can notice that during the Paleo Festival most of the measurements were made with a speed between 0 and 10. The reason is that most measurements were sent from people walking with their phone.



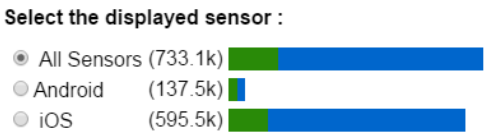
You can display measurements made by pedestrians by selecting the range from 0 to 10 km/h. You will notice that the color speed scale will automatically adapt to the new speed selection.

If you now select only speed higher than 10 km/h you will notice that the visualization changes and also the histogram shown for the time selection. Their color change to blue because only a very little portion of it is selected



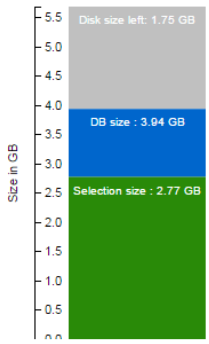
2.7.3 Sensor type controls

Finally you can select to display only Android or iOS measurements. Horizontal bars use also the same color to show the selected part (measurements made with this kind of devices during the Paleo Festival on Thursday the 23rd). The numbers in brackets show the number measurements by devices for the current visualization.



2.8 Monitoring the database

The console also provides tools to monitors the database. Firstly a chart shows how much space the database uses on the disk and how much space is left. You can also notice that the same colors than before are used to show the selected measurements and the rest. Then information about the number of selected measurements and how much space it takes out of the total. Finally information about when the database will be full based on the selected interval of times. In this case, 1251 were made during the last eight hours so if the same amount of measurements are made each eight hours the database will be full the 22 October 2017.

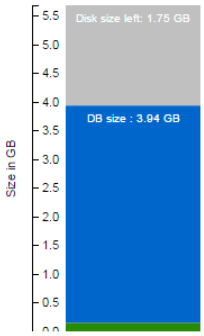


2.7M out of **3.8M** measurements are selected for a size of **2.77 GB** out of **3.94 GB**.

During the last 8 hours, 1251 measurements were made. At this pace, the disk will be full the 22 Oct 2017 - 10:39 (64 weeks, 2 days, 23 hours, 47 minutes)

Predictions based on the last : 30 minutes 8 hours 1 day 3 days 1 week

When visualizing the data from the Thursday during the Paleo Festival we can see how much space this represents for the database.



141.3k out of **3.8M** measurements are selected for a size of **149.63 MB** out of **3.94 GB.**

During the last 8 hours, 1251 measurements were made. At this pace, the disk will be full the 22 Oct 2017 - 15:29 (64 weeks, 3 days, 4 hours, 31 minutes)

Predictions based on the last : 30 minutes 8 hours 1 day 3 days 1 week