ELK stack Big Data visualization using D3 library

In collaboration with Addiliate (Clicktron Media S.L)

Bachelor's Degree in Informatics Engineering
Software Engineering

Author: Nicolás Ernesto Francisquelo Tacca

Director: Jordi Bayo Singla

Tutor: Alberto Abelló Gamazo

Autumn 2019

Facultat d'Informàtica de Barcelona (FIB)

Abstract

This document explains the development of the data visualization tools created with the D3 library for an existing AngularJs web application.

These visuals aim to represent the Big data from an Elastic stack in an understandable way. All the processes involved, from fetching the data to the front-end display in suitable representations and passing through the post-processing, are described in this memory.

Resum

Aquest document explica el desenvolupament de les eines de visualització de dades creades amb la llibreria D3 per a una aplicació web AngularJs existent.

Aquestes visualitzacions tenen com a objectiu representar informació de *Big data* procedent de l'entorn Elastic de manera fàcilment comprensible. Tots els processos involucrats, des de l'obtenció de les dades fins a la visualització *front-end* en representacions adients y passant pel post processament, es troben descrites en aquesta memòria.

Resumen

Este documento explica el desarrollo de las herramientas de visualiación de datos creadas con la libreria D3 para una aplicación web AngularJs existente.

Estas visualizaciones tienen como objetivo representar información de *Big data* procedente de un entorno Elastic de manera facilmente comprensible. Todos los procesos involucrados, desde la obtención de los datos hasta la visualización *front-end* mediante las representaciones adecuadas y pasando por el post-procesado, se encuentran descritos en esta memória.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Nowadays more than half of the world's population has access to the Internet and the average time spent online nearly hits 7 hours per day according to the latest reports from international organization such as ITU [1] and We Are Social [2]. This means huge amounts of data being created, transmitted and consumed every second.

The information contained in all this data is very important, not only for private companies and their specific business purposes but also for other fields like politics and science.

Analyzing the data consists basically in extracting and cleaning the information from the raw data, and then transforming and visualizing it in order to be able to understand it.

In this project we will mainly focus in transforming the data into valuable information and present it mostly in chart-based visualizations. The data we will be dealing with is a representation of internet traffic and we will face the challenge of creating visualizations that need to be readable and understandable for non-technical users.

The project is the final thesis of the Bachelor's Degree in Informatics Engineering, more specifically for the major in Software Engineering of the Facultat d'Informàtica de Barcelona (Universitat Politècnica de Catalunya). It has been developed for Addiliate (Clicktron Media S.L.), an affiliate marketing and media buying company based in Barcelona, under the direction of Jordi Bayo Singla.

1.1 Stakeholders

In this section we will describe the stakeholders of the project.

We accepted the stakeholder definition of the book *Strategic Management: A Stakeholder Approach*[3]: any group or individual who can affect or is affected by the achievement of the organization's objectives.

The different stakeholders of this project are listed as follows.

1.1.1 Developer

As the name suggests, is the person in charge of the project's development process. The role of the developer consist in making most of the decisions regarding the project in order to meet the objectives. To do this, the developer also has to perform the research in the field of study, to gather the requirements and then, implement the solutions following the planning that he or she has previously done.

1.1.2 Director and tutor

They are responsible of following the project's progress providing with help and advice to the developer whenever it's needed. Both roles are very similar, the main difference is that the director belongs to the company while the tutor represents the academic side of the project.

The director of the project is Jordi Bayo Singla, Head of IT department in Addiliate and the tutor is Alberto Abelló Gamazo, researcher and professor at the Universitat Politècnica de Catalunya.

1.1.3 Users

They are the ones that will be using the tools resulting of this project, this means that their feedback and interests should be carefully taken into account. The users of this project can be easily divided in smaller groups to better understand their needs according to their specific interests and use cases. These user types are listed as follows.

- **Advertisers**: An advertising company or individual that has a product and relies on others to advertise it for them. They need to see information about the traffic that they are paying for.

- **Publishers**: An individual or a company in charge of connecting the advert to the final user. They want to see if the traffic they are sending is converting or not, and why. There are a lot of variables they want to visualize to learn about the traffic they are dealing with.

- **Account Managers**: An employee from the company that can be either a publisher account manager or a advertiser account manager.They work directly with their clients in order to help them to configure all the settings properly to create a campaign in the case of advertisers, or to apply for these campaigns if they are publishers. They need to visualize a lot of metrics to provide useful tips and analysis to the clients.

- **Administrators**: Special users from the company that will be using the tools without restrictions. For example the owners or some members in the IT team.

2 State-of-the-art

In this section we will discuss the state-of-the-art and we will explain the three core concepts of the project: the Elastic stack, the AngularJS framework and the D3 library.

2.1 Elastic stack

The Elastic stack, initially also known as ELK stack due to the acronym of the first three components Elasticsearch, Logstash, and Kibana. Later on, however, a component called Beats was included and the company started calling it Elastic stack.

A common simplified architecture of the Elastic stack consist in Beats, a data shipper that reads and forwards the log files to a data processing component called Logstash. This component filters and normalizes the input logs before sending them to Elasticsearch, the storage component.Elasticsearch indexes and stores the data for further analysis and searches. The Kibana component provides visualization options of the Elasticsearch data. [4]

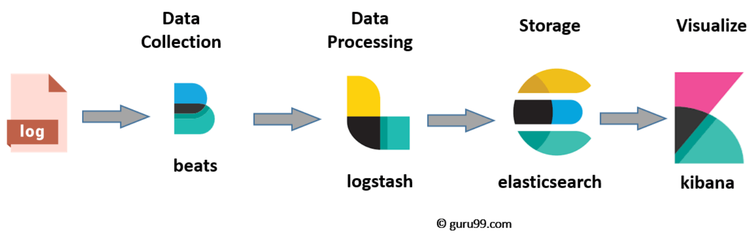

Figure 1: Elastic stack architecture

2.1.1 Elasticsearch

Elasticsearch, now known as Elastic, is an open source, distributed, RESTful, JSON-based search engine. Elasticsearch uses documents to handle the data and it has been designed to deal with huge amounts of data keeping a very low fetching latency to meet the demands of real-time use cases, such as log analysis, full-text search and many others.[5]

Elasticsearch offers high reliability, easy management and simple deployment. It also provides horizontal and vertical scalability.

In order to better understand Elasticsearch we need to deep dive in the most important terminology and core concepts [6]:

- **Document**: is the basic unit of information which can be indexed, expressed in JSON format. A document contains fields comprised of keys and values. The key is a string with the name of the field and the value is the data, which type can be string, number, date, or list. Every single document is associated with a type and a unique id. In **appendix A** there's an example of the documents used in this projects

- **Index**: is a collection of documents representing similar concepts. An index is identified by a name, and it can contain as many documents as desired. Elasticsearch allows you to define as many indexes in one single cluster. Every index can be split into several shards to be able to distribute data.

- **Cluster**: a collection of one or more nodes that together hold the entire data. It's identified by a unique name. Usually different clusters are defined for the development environments.

- **Nodes**: a single server that stores and indexes the data and belongs to a unique cluster. It is created when an Elasticsearch instance begins.

- **Shards**: subdivision of the indices. A shard is a fully functional and independent piece of index that can be stored in any node in the cluster.

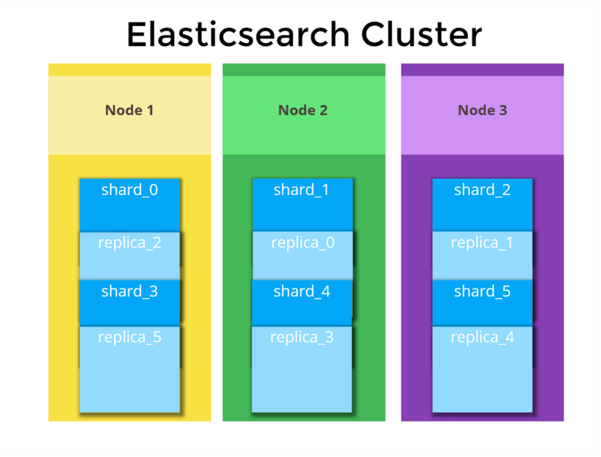

Figure 2: Elasticsearch cluster with 3 Nodes and 5 shards

2.1.2 Logstash

Logstash is a data collection and processing engine that ingests data from a multitude of sources simultaneously. It transforms and normalizes the data and makes it available for further use.

Logstash can unify data from disparate sources and then send it to one or more destinations, typically used for feeding Elasticsearch.[7]

The Logstash pipeline consists of three components:

- **Input**: is responsible of specifying and accessing the data input source.

- **Filters**: filters are a set of conditions checked against the actions or events of the input data.

- **Output**: the output contains parsed input data and some extra fields like timestamp, host or the input source path.

2.1.3 Kibana

Kibana is an open source data visualization and exploration tool used for visualizing the data stored in Elasticsearch. It's a powerful tool to perform advanced data analysis in charts, maps or tables visualizations.

The Kibana component provides a lot of features apart from the data exploration and visualization ones. It also allows the users to share almost everything and to collaborate in the research by easily exporting CSV files and with PNG or PDF reports or even shared dashboards.

Security, management and monitoring are also covered by Kibana, with customization of alerts and all kind of notifications. With the help of machine learning and data modeling, advanced features like forecasting and predictive analysis are also available.[8]

Even though it's the default visualization method for most of the Elastic projects, it doesn't fit our specific needs. Further explanations can be found in section **3.1**

2.1.4 Beats

Beats is an ecosystem of lightweight, purpose-built agents that acquire data to then feed it to the desired destination, usually Elasticsearch or Logstash in case some parsing or filtering needs to be done before shipping it to Elasticsearch. [9]

The potential of Beats resides in the libbeat framework. With Libbeat creating customized agents for ingesting any type of data is very straightforward. Thanks to this flexibility, the number of Beats available and the capabilities of the Beats ecosystem is growing rapidly.

Some examples of Beats data shippers are:

- **Filebeat**: built for logs files, is very useful to forward and centralize events from multiple servers, virtual machines and containers. Filebeat is the data shipper used in our specific Elastic pipeline because the data we store in Elasticsearch comes from the the traffic we handle in our servers in addition of some other business data.
- **Winlogbeat**: it reads data from any channel of the Windows Event Log. Winlogbeat is very useful to track everything that happens in a Windows environment, such as system updates, USB attachments, software usage, etc.
- **Packetbeat**: used for network data, it helps tracking the traffic flowing through the network, it supports many protocols and, since it's a library, new ones can be also added if needed.
- **Metricbeat**: it allows tracking system-level metrics such as CPU usage, memory, file system, disk I/O or even running process statistics in all Linux, Windows and Mac hosts. It also comes with a variety of modules that can be easily enabled in the configuration file to collect metrics from services like Apache, NGINX, MongoDB, MySQL, etc.
- **Heartbeat**: very useful for monitoring the uptime and availability of services by actively probing them.

2.2 AngularJS

2.2.1 Definition

AngularJS is a JavaScript framework which allows users to build well-structured, dynamic web applications. It's known as a MVW (Model View Whatever) framework although it provides functionality to be a typical MVC (Model View Controller) framework too.[10]

AngularJS extends HTML functionality by providing directives to the HTML tags. Directives are attributes or markers that can attach a specified behavior to the container DOM[1] element or even transform it and its children allowing you to create powerful and dynamic front-end applications. AngularJS has a lot of useful built-in directives, but you can also create your own directives to perform any type of DOM manipulation.

This framework is based in the concept of two-way data binding, this allows the easy binding of the JavaScript objects (models) to the HTML (view). This means that any change in the view is propagated to the model, and also the other way around; changes in the model update the view.

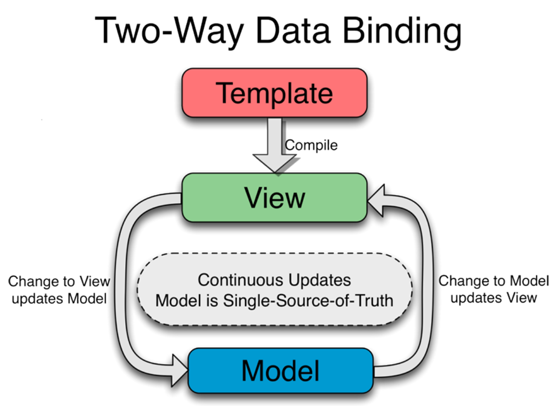
Two-Way Data Binding
Template
Compile
View
Change to View updates Model
Continuous Updates
Model is Single-Source-of-Truth
Change to Model updates View
Model

Figure 3: AngularJS two-way data binding

2.2.2 Components

Components are special kind of directives that were introduced to AngularJS in the 1.5 version. They deserve a deeper explanation since a big part of the project relies on the structure and characteristics of components.

Using components in an AngularJS projects is not just about being a step closer to a possible migration to the Angular[2] framework, but it also improves good practices, isolation, readability and most importantly for us, reusability.

Components consist basically in three parts:

- **Template**: it's the view part of the component, a projection of the models through the enriched HTML. Templates are the way of displaying the information to the user.

[1]DOM: Document Object Model is the standard for accessing, modifying, adding and deleting elements of a document.

[2]Angular: is a complete re-write in TypeScript of its ancestor AngularJS.

- **Controller**: it's a JavaScript object that contains the attributes, properties and functions used to handle the events, fetch data, and all the component's behaviour.

- **Module**: it's the component's definition, a container that links the controller to the template.

2.3 D3.js

D3 (Data-Driven Documents) is a JavaScript library that makes it easier to create data visualizations using the web standards SVG, Canvas and HTML.[11]

The library provides interaction techniques and powerful visualizations relying in dynamic DOM manipulation to take advantage of the full capabilities of browsers to create the desired interface for the data.

D3 provides the necessary tools and building blocks to construct a very own visualization instead of pretending to fulfill all the necessities with ready made charts like most of the charting libraries. This approach is much lower level and it requires more coding and time, yet it gives full control to create bespoke charts.

Some of the library's most important functionality are:

- **Selections**: functions that allow easy DOM elements manipulation. There are only two selection functions, and they both take a single parameter which is selector string (W3C selectors API). These functions are:
  - **d3.select**: selects the first matching element.
  - **d3.selectAll**: selects all the matching elements.

- **Dynamic properties**: styles,attributes and other properties can be defined as functions that depend on the data, instead of simple constants. D3 has many built-in functions and factories that help for example with line or pie-chart generation.

- **Transitions**: with only one function, yet very modular and flexible, D3 can gradually interpolate styles and change attributes over time.

The following line of code is a simple example of the three characteristics mentioned above:

```javascript
d3.select("rect").transition().duration(100).style("opacity", "0");
```

Listing 1: D3 fading out transition example

It fades out all the *rect* elements of the DOM where it's applied. We first select all the elements, and then we apply the opacity dynamically with a transition of 100 miliseconds.

3 Project scope

In this section we will talk about the project scope and the motivations that lead us to do it, including the objectives we want to accomplish and the requirements. We will also evaluate the possible risks and provide solutions.

3.1 Motivation

The company started storing data about the internet traffic received from their publishers into Elasticsearch to be able to analyse it and extract useful business information in real time or near real time.

The data stored in Elasticsearch documents contains information like the type of device used to send the request, the country of origin, the internet browser used, the operative system of the device, the payout for the publisher,the cost for the advertiser, and many others (example of a document in Appendix A).

In affiliate marketing this data is used by the advertisers to decide if they pay or not for the traffic, and it can be also used to detect fraudulent traffic. After 9 months of storing this data, the company decided it was time to make this data available to their clients through their existing website.

In short,the motivation of this project is the need of visualizing business information from the raw data stored in Elasticsearch.

3.2 Solution

The first solution we studied for visualizing the information was using the Kibana component from the Elastic stack. As Kibana was already set up and working with the elastic documents we wanted to analyze and visualize, we thought it could be a good idea to just try to export Kibana's visuals to the web application.

After some research we found out that this solution didn't work for us because even though Kibana has some features and options to share dashboards or embed them using *iframes*, it didn't allow us to have the full control we needed. Other major downsides we found to this solution were the authentication security and the user roles definition which could end in data leaks. Also, relying on a third party technology which is still being developed and suffers frequent changes, was a risk we didn't want to take.

The solution chosen was to create bespoke charts using the D3 library which we briefly introduced in section 2.3. This library allowed us to implement data pagination to control the volume of data returned by Elasticsearch, to be able to post-process the data in order to populate it with extra information when needed and to adapt the visuals to the look and feel of the existing website by applying CSS styles.

We also evaluated other free open-source charting libraries like Chart.js, Dygraphs or Chartist.js but even though they were easier to use and the pre-built charts provided would have considerably reduced the implementation time, they didn't provided us the control we needed over the visualization rendering and the user interaction.

The charts we implemented aim to provide insights to users that wouldn't necessarily posses the expertise to analyze the information presented without help. We focused on providing a user-friendly interface to display the information using clear and intuitive designs which makes it easy to analyze and understand.

These charts show business information obtained from processing the data of the internet traffic tracked by the company. By making this information available to the clients, we intend to help them understand their traffic and find the trends that maximize their performance so they can make business decisions that lead them to increase their income.

Most of the charts we implemented were chosen because they are well known charts that have been fully studied and proved to be very useful when used to display suitable data. Other visuals have been chosen just for being a quick and attractive way to show qualitative information, for example the tag cloud (figure 10).

There were also charts we initially considered to implement but, after some research, we decided not to because of different reasons such as interpretation complexity, high biasing of the data or spatial problems. Examples of discarded charts can be tree map charts, bullet charts or calendar heat-maps.

3.3 Objectives

The company has a web application where the publishers can log in to see new campaigns they can apply to, manage their placements and settings and contact their account managers. On the other side, advertisers can see their own campaigns and also contact their managers.

This web application has been developed in AngularJS, a framework we already introduced in section 2.2. Prior to this project, the only information the users could find about their performance or their traffic was usually outdated and non interactive, in best cases the rudimentary KPI[3] were daily updated using scheduled jobs, in other they were just calculated using heuristics and estimations.

We aimed to provide accurate information with highly interactive and intuitive visualizations by using real data to bring the users the tools they needed in the platform they are already using.

In short, the main objective is developing and integrating data visualization tools in an existing AngularJS web application using the D3 library to represent information extracted from Elasticsearch.

[3]KPI: Key Performance Indicators are a type of performance measurement to evaluate the success of an activity.

3.4 Requirements

- **Usability**: visualizations and user-interaction must be intuitive and self-explanatory as we are not targeting end users with a very technical background.
- **Extensibility**: the implementation of the charts shouldn't strictly depend on the data to display. In the future the company may want to reuse or extend the visualization components with other data and use cases.
- **Maintainability**: the code has to be simple, well documented and maintainable as it will be integrated in an existing web application that the company will continue developing. Good practices and using design patterns to create simple and clean code in both front and back end is a must.
- **Efficiency**: as we mentioned before, we will be feeding the front-end visualizations with big data, this means we need to be very careful with the loading times in order to provide a good user experience.
- **Value**: this is a private's company project ergo it has to add value to the existing platform, otherwise it can't be considered successful.

3.5 Risks and possible solutions

Some problems and inconveniences may happen during the development process, being aware of these possible obstacles and their solutions can help us to react more rapidly.

3.5.1 Failure of the development hardware

When the equipment being used for developing purposes or some peripherals stop working.

The solution is to have backups in the cloud of the latest of the source code and try to get another equipment as soon as possible.

3.5.2 Failure of the development software tools

When the software being used in the development environment stops working, for example local servers, containers, or even the IDE.

The solutions can be restarting the services or shutting down the computer because it may be caused by too many processes running on the CPU, check the software license and renovate them if necessary or try to re-install or update to a newer version.

3.5.3 Wrong temporal planning

When the planning of the tasks are not accurate enough or the programmer gets stuck in a task that was more complicated than expected.

The solution is to make a serious and sensible planning, with some margins to prevent possible delays that may occur during the developing process.

3.5.4 Cloud services failure

When the services we have in the cloud stop working. For example the instances in AWS, the repository of the project, communication programs like Slack,etc.

The solution is to use the services that provide the highest up-time to reduce this risk to the minimum.

4 Project implementation

In this section we will explain all the work done on project implementation.

4.1 Implementation phases

It's important to emphasize that even though visualizations require a lot of front-end related development, we are not just focusing on that area as we also need to retrieve and process the data before being able to display it in the suitable visualizations.

The implementation is divided in three phases.

4.1.1 Data search

This is the first phase and it comprises all the process for obtaining the raw data stored in Elasticsearch.

The Elasticsearch data is exposed through a RESTful API reliying in the usage of DSL queries. This API allows you to interact with data using the standard HTTP verbs in addition of parameters and extra data using the URI components. Elasticsearch provides a full Query DSL (Domain Specific Language) based on JSON to define all kind of queries which ensure complete access to all the Elasticsearch features.

Due to our specific use case, we needed to create these DSL queries dynamically but since there isn't any helper library, we decided to implement a bespoke php builder library, from now on called DSLQueryBuilder.

The DSLQueryBuilder is an interface to create the DSL queries to help us avoiding duplicated code, hardcoded queries and providing the project with better reusability and maintainability.

This library follows an incremental development approach which means that it only contains a mapping of the subset of features that have been needed from the Search API of Elasticsearch. Other features and options are added to the DSLQueryBuilder whenever new DSL queries require it.

For example, to get the monthly traffic and income generated by a user with a SQL query we would do something like this:

```sql
SELECT month,COUNT(clicks),SUM(income) FROM TrafficUsers WHERE id='1' GROUP BY month.
```

Listing 2: SQL query example

The equivalent DSL query in JSON format would be:

```
{
    "query": {
    "bool": {
      "filter": [
        { "term": { "userid": "1" }},
      ]
    }
    }
    "aggs" : {
        "monthly_results" : {
            "date_histogram" : {
                "field" : "date",
                "interval" : "month"
            },
            "aggs": {
                "monthly_income": {
                    "sum": {
                        "field": "income"
                    }
                }
            }
        }
    }
}
```

Listing 3: DSL query example

Using our DSLQueryBuilder library to construct the same DSL query we would write:

```
composeQueryContext(){
    $bool = new Bool();
    $termFilter = new Term("userid",1);
    $bool->addFilter($termFilter);
}

composeAggregations($aggregations){
    $aggregations->setName("monthly_results");
    $dateHistogram = new DateHistogram();
    $dateHistogram->setField("date");
    $dateHistogram->setInterval("month");

    $aggregations->setAggregationType($dateHistogram);

    $trafficAggregation = new Aggregations();
    $sumAggregation = new Sum();
    $sumAggregation->setField("income");

    $trafficAggregation->setName("monthly_income");
    $trafficAggregation->setAggregationType($sumAggregation);

    $aggregations->addAggregation($trafficAggregation);
}
```

Listing 4: DSLQueryBuilder example

As we can see in the example above, there are two methods, *composeQueryContext* to apply filters to the documents we want to find, and *composeAggregations* to create the aggregations or groups on the previously filtered documents.

We implemented the DSLQueryBuilder library in a way that for every new DSL query we only need to inherit from a class called DslQuery and implement these two methods. The DslQuery class contains all the logic needed to parse the objects to create the query in JSON format and to connect to the Elasticsearch instance to send the request.

The different filtering and aggregation options of the API are implemented as php objects, with the same properties as in the official documentation plus their corresponding getters, setters and parsing methods.

4.1.2 Data processing

The second phase consists in processing the data from the Elasticsearch response obtained in the data search phase.

Elasticsearch responses are in JSON format and they have a common core structure and default fields like *took* which is the query execution time in milliseconds , *_shards* with information about the shards involved in the searching, *hits* containing the results or *aggregations* which holds the information of the buckets if any aggregation was defined in the DSL query.

An example of an Elasticsearch response for the DSL query of the previous examples(**Listing 3**) would be:

```
{
   "took": 11,
   "timed_out": false,
   "_shards" : {
    "total" : 3,
    "successful" : 3,
    "skipped" : 0,
    "failed" : 0
  },
   "hits" : {
    "total" : 10,
    "max_score" : null,
    "hits" : [ {
      "_index" : "traffic_user",
      "_id" : "0",
      "sort": [0],
      "_score" : null,
      "_source" : {"user_id":1,
                    "income":10,
                    ...
                   }
              },
       {...}
    ],{
    "aggregations": {
      "monthly_results": {
         "buckets": [
            {
               "key_as_string": "2019/01/01 00:00:00",
               "key": 1546297200,
               "doc_count": 8,
               "monthly_income": {
                  "value": 123.0
               }
            },
            {
               "key_as_string": "2019/02/01 00:00:00",
               "key": 1546383600,
               "doc_count": 6,
               "monthly_income": {
                  "value": 48.0
               }
            },
            {...}
         ]
      }
   }
}
```

Listing 5: Elasticsearch JSON response example

As we can see in this example, the most relevant information, or at least the business information we will use to feed the visualizations is in the *aggregations* property, distributed in the so called buckets. All buckets have a property called *key* containing the value used to aggregate the documents and a *doc_ count* property that is the total documents that fell in the bucket.

In order to filter and parse the results from this format into a more front-end oriented format, we iterate through the buckets of the response, to get the valuable information and compute some extra calculations if needed. Sometimes we also needed to populate the resulting data structure with extra information that wasn't persisted in the elastic documents but in a MySQL database by matching the ids. Although it may seem that this can increase the latency and become a bottleneck making useless the use of Elasticsearch, it's not the case as even though we get data from millions of elastic documents we query few rows in MySQL.

4.1.3 Front-end visualization

The last phase of implementing a visualization is the actual implementation of the visual component that will be fed with the already processed information from the previous phases.

One of the main challenges of these components is the data abstraction required to create reusable components that can represent almost any type of data, with the only limitation of the coherency between the visualization type and the nature of the data. For example, we shouldn't try to visualize data that changes over time in donuts or pie charts because they serve as snapshots of the data not to represent different periods of time.

This last phase is only performed once for each new type of visual component thanks to the data abstraction mentioned above. For example, to represent two distributions of the internet traffic, one by countries and the other by type of devices used, we could use the same visual component, a pie chart, therefore we will only implement it the first time and then reuse it whenever we need it again.

4.2 Table visualizations

The first visualizations we implemented weren't D3 charts but tabular representations of the information. We used these tables as our first approximation to start working on the DSL-QueryBuilder and retrieving information from Elasticsearch.

Despite being our first approach,we knew that tabular visualizations are very useful for deep analysis of the data, so we decided to implement advanced tables using the DataTables plugin for the jQuery[4] library. This plugin helped us implementing the tables and, even though it has many options to provide advanced features, we decided to create our own AngularJS components for handling table pagination,sorting and changing columns visibility.

The decision of implementing these features with our very own components came because we wanted to have most of the HTML directly written in the template instead of binding the HTML generated by rendering functions in the controller. After some research we found out that this could only be achieved by creating our own components instead of using the DataTables built-in options. This way we could clearly separate the logic of the controller from the template's view resulting in a cleaner and more maintainable code.

Our tables consist in four components:

- **Custom table**: this is the actual table where the columns,rows and cells are defined. It also contains functions for requesting the table's data and all the callback methods needed to efficiently react to the changes done in the other three components.
- **Pagination**: this component is responsible for providing the table with a pagination behaviour. It consists in buttons that allows the user to go to the first, last, previous and next page in case the number of rows is bigger than the table's length.
- **Length**: this component allows the user to change the table's length or size (the maximum number of rows to display). When used together with the pagination component this length is the number of rows per page.
- **Column visibility**: this component allows the user to change some columns visibility. We can also define groups of columns for example to hide all the odd columns.

4.3 Chart visualizations

As we mentioned in previous sections, we used the D3 library to implement all the chart visualizations as reusable AngularJS components. The bindings are different for each component and so it's the data structure expected, mostly depending on the way of constructing the chart.

All the charts implemented in this project have tool-tips which can be used to show additional information when the mouse is over a column or section, except from the scatter, where we decided not to add tool-tips because hundreds of dots are usually displayed, which makes tool-tips difficult to read and therefore useless.

We also implemented other features depending on the complexity of the visualization to make them easier to read and use, for example, toggling columns visibility or zooming.

[4]jQuery: a JavaScript helper library for HTML DOM manipulation and event handling

4.3.1 Columns and line chart

This is a dual axis chart that helps spotting the relationship between two variables that are in different scales or belong to different magnitudes.

Columns-line charts can be very useful if there is a strong and meaningful correlation between the two variables so we can extract valuable information by comparing them, otherwise we will end with a confusing and misleading chart that doesn't provide any helpful information.

Some of the most relevant parts of the code are:

```
// We need to define two types of scales for the X axis: a band scale for the columns
    and a point scale for drawing the lines
let xBarsScale = d3.scaleBand().range([0,width]).domain(this.xNames).paddingInner(0.5)
    .paddingOuter(0.25)
let xLinesScale = d3.scalePoint().range([0,width]).domain(this.xNames).padding(0.5);

// Even though both scales are linear, we still need to define two because their
    domain is different
let yBarsScale = d3.scaleLinear().rangeRound([height, 0]).domain([0, d3.max(barsData)
    ]).nice();
let yLinesScale = d3.scaleLinear().rangeRound([height, 0]).domain([0, d3.max(lineData)
    ]).nice();

--------------------------------------------------------------------------------
// Defining the X axis, we can use any of the two scales, in this case we used the
    point scale.
gChart.append("g")
   .attr("class", "x-axis-"+columnLineChartId)
   .attr("transform", "translate(0," + height + ")")
   .call(d3.axisBottom(xLinesScale)); //method that draws the bottom axis

// Left Y axis - Columns
gChart.append("g")
   .attr("class", "blue-axis y-left-axis-"+columnLineChartId)
   .call(d3.axisLeft(yBarsScale).ticks(null, "s")) //method that draws the left axis
   .append("text")
   .attr("dy", "0.32em")
   .attr("fill", "#000")
   .attr("text-anchor", "start")
   .text(yBarsAxisText) //left y-axis labels
   .attr("font-size", 12)
   .attr("text-anchor", "middle")
   .attr("transform", "rotate(-90)")
   .attr("x", 0 - (height / 2))
   .attr("y", 10 - margin.left);

// Right Y axis - Lines
gChart.append("g")
   .attr("class", "orange-axis y-right-axis-"+columnLineChartId)
   .attr("transform", "translate(" + (width) + ",0)")
   .call(d3.axisRight(yLinesScale).ticks(null, "s"))//method that draws the right axis
   .append("text")
   .attr("dy", "0.32em")
```

```
    .attr("fill", "#000")
    .attr("text-anchor", "start")
    .text(yLinesAxisText) //right y-axis labels
    .attr("font-size", 12)
    .attr("text-anchor", "middle")
    .attr("transform", "rotate(-90)")
    .attr("x", 0 - (height / 2))
    .attr("y", margin.right - 10);
```

Listing 6: Columns and line chart code

The **figure 4** is an example of one of the many columns-line charts we created. It displays information about the countries that are generating more revenue to the company in comparison of the number of campaigns they have.

We can easily spot the countries where the campaigns are performing well and which countries are not. With this information we can decide, for example, to increase the efforts in finding more campaigns for the countries with a high performance or trying to increase the performance of the countries that already have a lot of campaigns but a low revenue.

This type of chart is mainly used in the Account Manager's dashboard because it provides a good overview of the whole companies' performance.

Every time they log in they see four of these charts, each of them illustrating the correlation of the campaigns alive with the market verticals, the payment methods, the conversion point or as in the example, with the countries, all of them in a fixed 4 months time frame.

The same four charts are also included in the Data Analysis page but in a more dynamic and interactive way. This page allows users to explore the data by choosing the desired time frame and applying filters.

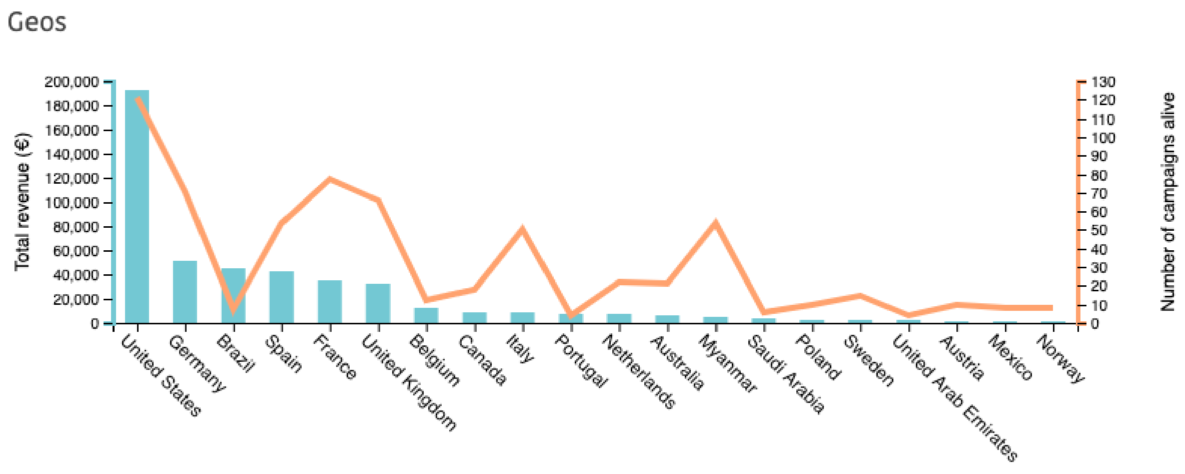

Figure 4: Columns and line chart example

4.3.2 Grouped columns chart

A grouped columns chart, also known as clustered column chart is used to compare different categories of two or more groups.

When populated with relevant information, this chart helps spotting differences between the categories inside a group and also to compare the same category across the different groups.

Visualizing the individual category distribution of a group and evaluate it in a global context using the information of the other groups makes this chart a very powerful graphical tool.

In order to help understanding and analyzing the data displayed on this chart we included an interactive legend, this means that the legend doesn't just tell us what the colors of the categories are, but also allows us to toggle the categories visibility so we can focus in the ones we want.

All the interactive legends implemented have a similar implementation, consisting in dynamic changes of the elements' style when a click event is received in the legend.

```
this.legend = gChart
   .append("g")
   .attr("font-family", "sans-serif")
   .attr("font-size", 10)
   .attr("text-anchor", "end")
   .attr("class", "legend"+this.groupedBarsChartId) // we need classes to be unique
      for each chart
   .selectAll("g")
   .data(this.groupNames.slice()) //the names of the categories of the groups
   .enter()
   .append("g")
   .attr("transform", (d, i) => "translate(60," + i * 20 + ")");

const vm = this; //We need this hack for the onClick function as we want to use the '
    this' DOM variable but we also need a reference to the controller context 'this'

this.legend
   .append("rect")
   .attr("x", this.width - 19)
   .attr("width", 19)
   .attr("height", 19)
   .attr("style", "cursor:pointer") //helps the user to understand it's clickable
   .attr("fill", this.colorScale) //same scale used to paint the columns used to paint
       the rectangles of the legend
   .on("click", function (d){
      let key = ".data-"+vm.groupedBarsChartId+"-"+ d.replace(/ /g, "");
      // determine if current column is visible
      let active = !legendKeys.get(key);
      let newOpacity;
      if(active){
      //maxValues allows us to resize the chart very efficiently
         vm.maxValues = vm.maxValues.filter(value => value!== vm.maxTotalValues[d]);
         newOpacity = 0;
      }
      else {
         newOpacity = 1;
         vm.maxValues.push(vm.maxTotalValues[d]);
```

```
        }
        // hide or show the elements
        d3.selectAll(key).style("opacity", newOpacity);
        // update whether or not the elements are active
        legendKeys.set(key, active);
        // Here 'this' is a variable referencing a DOM element, not the controller
        this.style.opacity = active ? 0.5 : 1;
        vm.updateAndResize(vm.gChart);
    });

this.legend
    .append("text")
    .attr("x", this.width - 24)
    .attr("y", 9.5)
    .attr("dy", "0.32em")
    .text(d => d);
```

Listing 7: Grouped columns chart code

The **figure 5** is an example of a grouped columns chart that shows the revenue and the margin by advertisers. We changed the x-axis labels to hide the real name of the clients.

With this example we can see how easy is to compare the revenue with the profit margin (categories) of an advertiser but also to compare these two categories across the rest of advertisers (groups). As we mentioned before, we can hide the margin or the revenue categories so we can focus just in one category.

This type of chart is part of the Performance Component (**figure 16**) showing the same information as in our example. It's also included in the Conversions page where the categories are the conversions' disapproval types and in the Data Analysis page showing the revenue of the conversion points by market verticals.

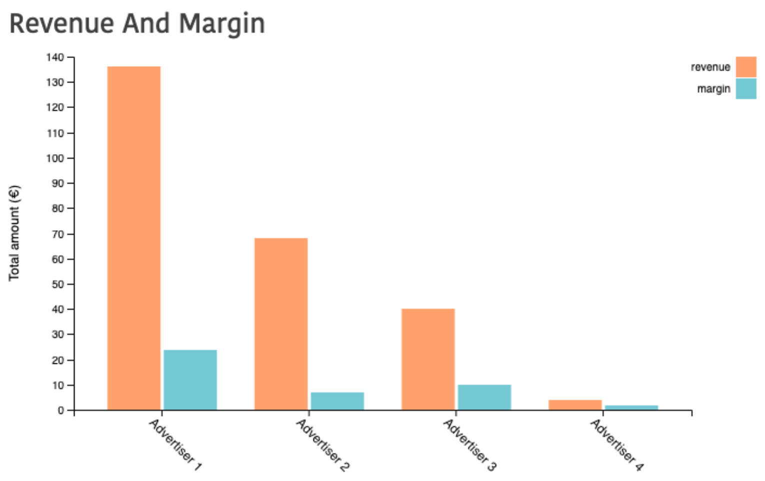

Figure 5: Grouped columns chart example

4.3.3 Stacked columns

The stacked columns chart we implemented is actually a 100% stacked column chart, which illustrates the relative percentage of the data in stacked columns that always add up to 100%.

This chart is used for part-to-whole comparisons across categories and helps spotting proportions differences and changes.

The **figure 6** shows the conversion status percentage of each vertical. When most of the conversions are disapproved it can be an indicator that a vertical is receiving a lot of fraudulent traffic. Verticals with mostly pending, can show us that advertisers owing campaigns with this vertical are taking too long to approve them.

This chart can only be found in the Conversions page.

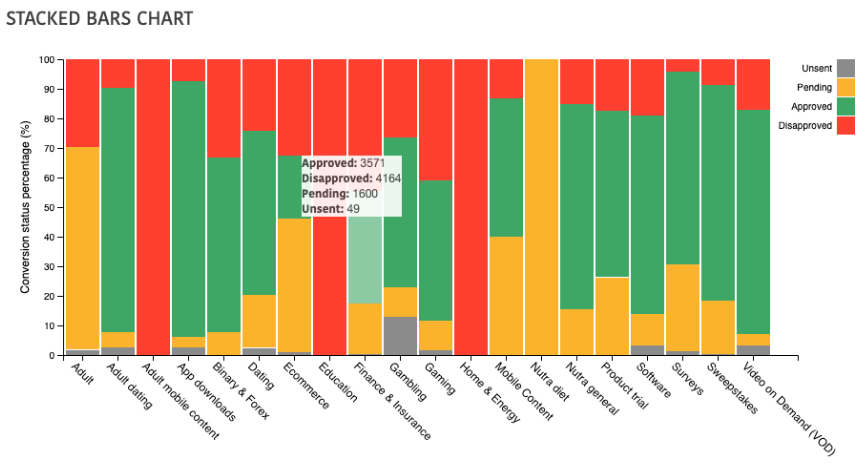

Figure 6: Stacked columns chart example

After finishing this chart, we obtained some feedback saying that, even though it was very useful, they also wanted a chart to easily spot the market verticals that are receiving more traffic as they deserve more attention than the rest, therefore we came up with the one shown in the **figure 7** but we finally discarded this option for being counter-intuitive and we decided to implement a Marimekko chart (section 4.3.4) instead.

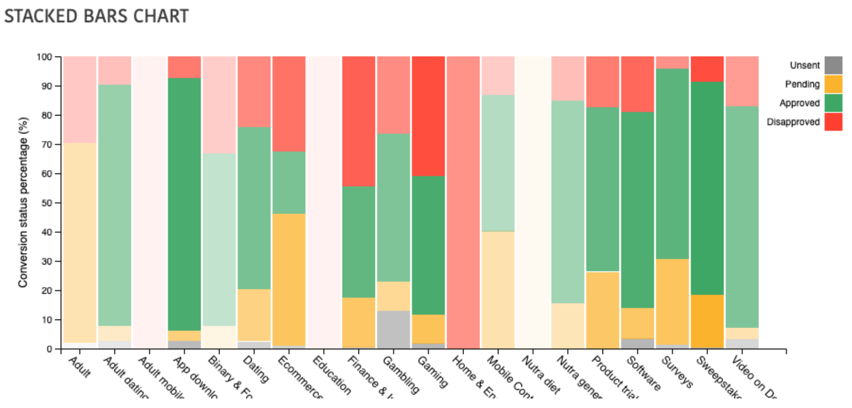

Figure 7: Stacked columns with opacity chart example

4.3.4 Marimekko chart

A Marimekko chart can be defined as a stacked columns chart with variable column's width. This type of chart illustrates two dimensions which allows to focus on large segments(most relevant categories) and identify white spaces(categories missing).

These charts show the proportion composition of each category and its relevance in the data set.

Typical use cases of Marimekko charts illustrate data sets where the value used to calculate the widths of the stacked columns can be scaled with a linear scale. With a wider range, where categories can go from small values to very big ones it becomes difficult to represent the data properly as most of the columns won't even appear.

For our specific use cases, we wanted the column widths to represent the amount of traffic so we decided to use a logarithmic scale. The problem of this implementation is that it's not possible to apply the logarithms directly to the data set because percentages wouldn't make any sense, therefore it wasn't possible to use the D3 stack generator as we did in the stacked columns to calculate the positions of the stacks.

We had to implement our very own solution from scratch as shown in the extract of the code below.

```
let totalLog = 0;

// creating the data structure to construct logarithmic variable width columns
this.data.forEach((element,index) =>{
   let cumulativeStack = 0;
   element.stacks.forEach(stack =>{
       cumulativeStack += stack.value;
       stack.stackStart = cumulativeStack;
       stack.dataIndex = index;
   });
   element.columnStart = totalLog;
   totalLog += Math.log1p(element.total)
});
//the x-axis labels
let label = node.append("g")
   .attr("class", "label")
   .attr("transform", d => { //we calculate the position of the columns
       let columnStart = ((d.columnStart)/totalLog)*100;
       let columnWidth = (Math.log1p(d.total)/totalLog)*100;
       let x = this.xScale(columnStart) + (this.xScale(columnWidth)/2);
       return "translate(" + x + "," + this.height + ")";
   });

node.selectAll("rect")
   .data(d => d.stacks)//for every column we draw the stack rectangles
   .enter()
   .append("rect")
   .attr("class",'cell')
   .attr("fill", d => this.colorsScale(d.key))
   .attr("x", (d) => {
       let percentageAcumulated = ((this.data[d.dataIndex].columnStart)/totalLog)*100;
       return this.xScale(percentageAcumulated);
```

```
    })
    .attr("y", d => {
        let percentageAcumulated = ((d.stackStart/this.data[d.dataIndex].total))*100;
        return this.yScale(percentageAcumulated);
    })
    .attr("height", d => {
        let percentage = (1-(d.value/this.data[d.dataIndex].total))*100;
        return this.yScale(percentage);
    })
    .attr("width", d => {
        let percentage = (Math.log1p(this.data[d.dataIndex].total)/totalLog)*100;
        return this.xScale(percentage)-1;
    })
}
```

Listing 8: Marimekko chart extract

The **Figure 8** is an example of a Marimekko chart that represents the conversions' status of the top 5 publishers. This type of chart is also part of the Performance Component, and it's also included in the Conversions page.

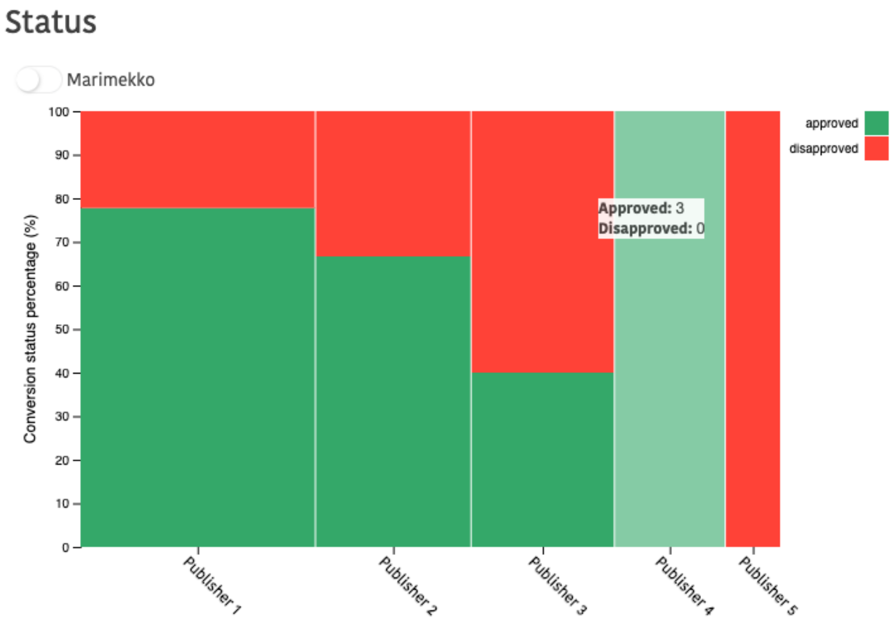

Figure 8: Marimekko chart example

4.3.5 Scatter plot

The scatter plot, also known as correlation plot, is a two-dimensional data visualization that uses dots to illustrate the values of two different variables using Cartesian coordinates.

Scatter plots are very useful tools for conveying the relationship between two variables. It helps spotting the trend when the variables are very correlated or to visually find if there isn't a discernible relationship between them.

In order to deep dive in the data set, we implemented a zooming feature, and also an advanced clickable legend that allows hiding the dots according to their origin and category.

The **Figure 9** is an example of a scatter plot. It shows the amount of time to convert and the conversion time. We also added two more dimensions by using different symbols instead of just dots, to represent the campaign of the conversion (origin) and the color of the symbols to see the conversion status (category).

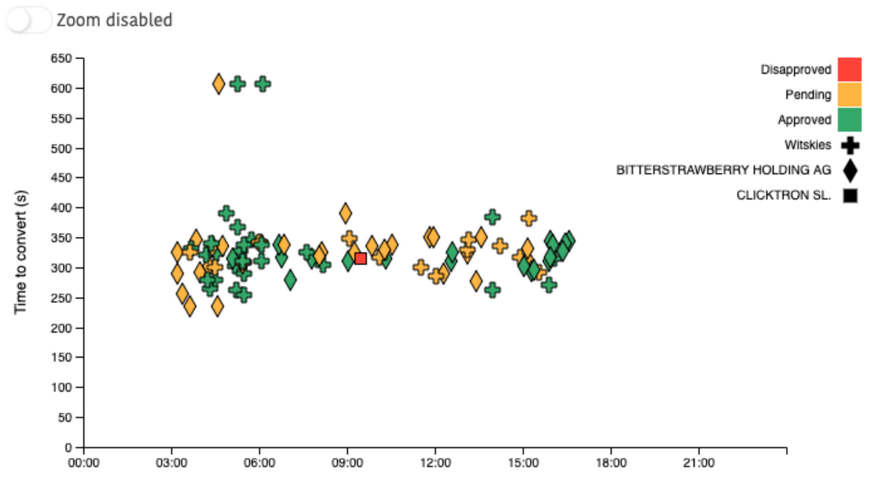

Figure 9: Scatter plot example

The main problem of scatter charts are the data outliers which can affect all the visualization if not treated properly. In order to minimize these outliers we implemented a method to remove the data points having values that differ from the trend in more than a certain percentage given.

4.3.6 Tag cloud

The tag cloud, also known as word cloud, is a visual representation of text data where the importance of the tags is shown with the font size.

It can't be really considered a data analysis tool but it's a simple and attractive way of visualizing the top tags of a list.

For creating this visualization we used the *d3-cloud* which is basically a layout algorithm that calculates the tags positions very efficiently.

The **Figure 10** is an example of a tag cloud that shows the publisher names that have been sending more traffic during the last three days. Tag clouds are found for example in the publisher dashboard, showing the top verticals converting.

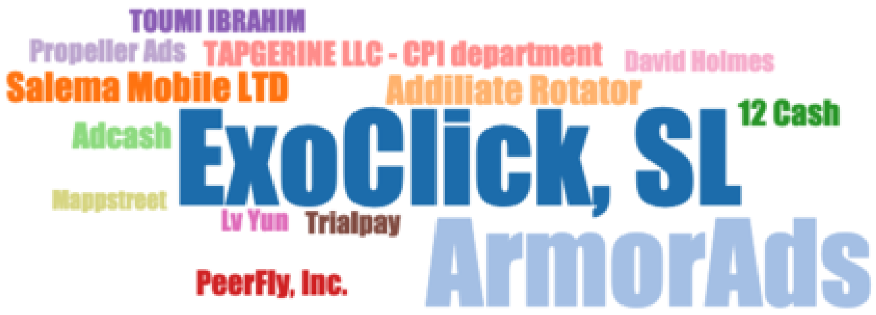

Figure 10: Tag cloud example

4.3.7 World map chart

This chart is a heat map of the world's political map where all the countries are represented. It can only be used to represent geographical-related data sets.

The **Figure 11** is an example of the world's map chart showing the countries the internet traffic comes from. We used a logarithmic scale instead of a linear scale to paint the countries because the range of the traffic of each countries is very wide and otherwise we would have ended with a map where huge traffic countries would have completely hidden the information of the rest of countries with less traffic.

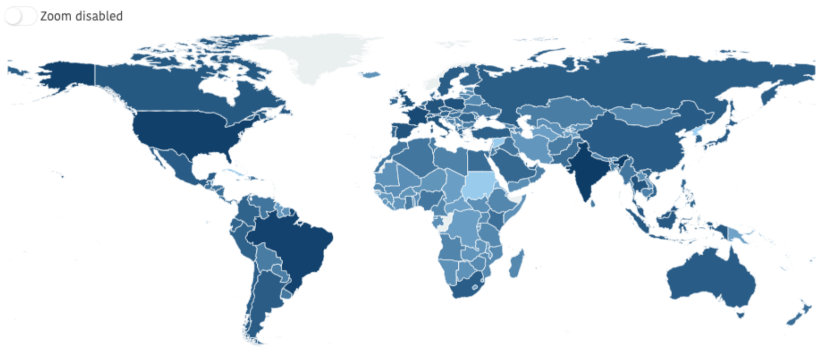

Figure 11: World map chart example

4.3.8 Pie chart

A pie chart is a circular graphic divided in proportional segments to show numerical proportions and percentages between different categories.

In a pie chart you can compare the categories by looking at the ratio of the angles, the ratio of the areas of each segment or slice or the length of the outer circumference.

This type of chart are useful to give the user a quick vision of the proportional distribution of the data set, but it's not as efficient as the 100% stacked column charts because comparing element's size by looking at the areas is less intuitive than it is with the lengths.

Other downsides of pie charts are the spatial efficiency and the low data-ink ratio, which Edward Tufte pointed out to be a problem that clutter and slows data comprehension in his book *The Visual Display of Quantitative Information*[12]

D3 has built-in arc and pie generators. The *d3.arc* method creates a circular or annular sector while *d3.pie* computes the necessary angles to represent the array of data, usually called to prepare the data for the arc generator. Both generators have many options but we need to be careful on what to modify as for example increasing the padding between sectors using the arc generator increases the bias of the resulting representation because applying the same padding affects smaller sectors, but if we add the padding using the pie generator, it computes the corresponding angles to avoid introducing these bias.

The **Figure 12** is an example of a pie chart used in the Conversions page that illustrates the proportion of the countries generating the campaign's conversions.

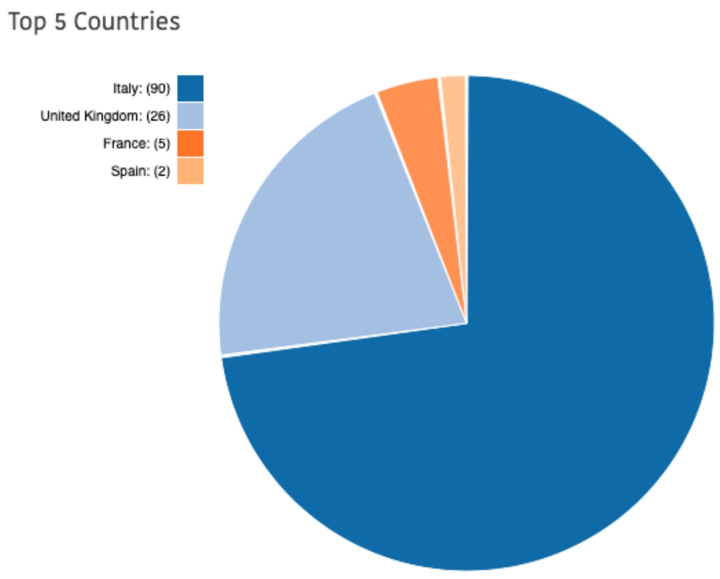

Figure 12: Pie chart example

4.3.9 Donut chart

Donut charts are basically pie charts with a blank central area. Although at first sight the blank space does not seem enough reason to consider that these two graphs are different, there are many features that change and make donut charts a better data visualization option.

Donut charts emphasize focusing on the arcs lengths to compare proportions instead of the sections' area. It also improves spatial efficiency and provides a useful empty area in the center of the chart that can be used to display important information as it draws the eyes of the users.

The **Figure 13** shows two advanced donut charts. We say these are advanced because we used the blank space to add a smaller donut chart inside. Outer donuts represent the traffic distribution while the inner donut represent conversions. The left chart's sections represent the operative systems and the right one the device types.

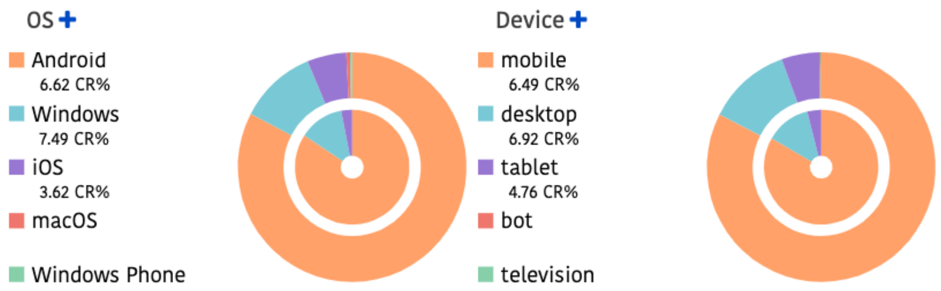

Figure 13: Donut charts example

4.3.10 Sunburst chart

Sunburst charts are circular visualizations that allow representing hierarchical data in a series of rings where the sections of the outer ones are the decomposition of the inner ring's sections. If the data set is not hierarchical, the sunburst chart looks like a donut chart, as it will contain only one ring.

This type of chart is mainly used to illustrate how one ring is divided into its contributing pieces, breaking the data into smaller and more specific subcategories.

During the implementation of the sunburst chart we found out that adding labels to each section wasn't a good approach as it made the chart unreadable, therefore we decided to create a dynamic legend that shows the paths of the sections until the selected one at the top of the chart.

The **Figure 14** is an example of a sunburst chart representing the profit margin generated where the smaller granularity, the innermost ring, shows the device type distribution, the second ring the operative system, and in the outermost we have the operative system version.

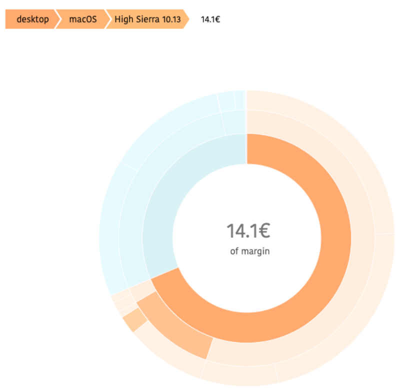

Figure 14: Sunburst chart example

4.3.11 Line chart

Line charts are two-dimensional visualizations used to represent information as series of data points connected by straight line segments.

This type of charts are used to analyze data that changes over a continuous interval or period of time, typically used to display performance measurements and depict trends and behaviours.

When more than one line is drawn in the same chart, it turns to be a perfect tool for comparing information and measures, but too many lines can clutter the chart, increasing it's complexity and making it unreadable.

This type of visual is part of the Performance component and it's also included in the Data analysis page and in the account manager's dashboards.

The **Figure 15** is an example of a line chart with multiple lines extracted from the Data Analysis page.

Even though we are aware that many lines increases chart's readability, we decided to draw them in the same chart instead of creating more due to the limited space. We also knew that this specific chart will only be used by highly experienced users (administrators, finance and head of sales team). However, lines can be hidden by clicking the check-boxes located at the bottom of the chart.

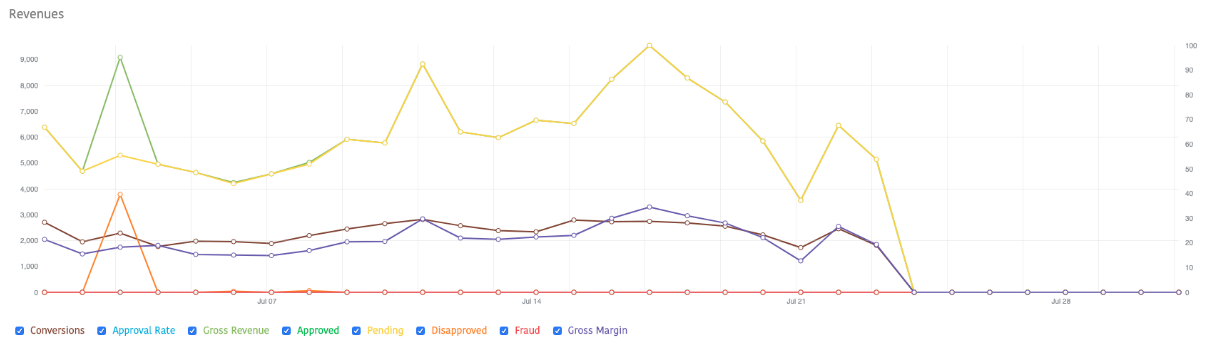

Figure 15: Line chart with multiple lines

4.3.12 Performance component

The performance component consists in four different types of charts that combined become a very useful data visualization tool for measuring performance. This tool is used for analyzing the performance of campaigns, advertisers and publishers.

The **Figure 16** is an example of the performance component of a campaign from an account manager or admin's login. Publisher and advertisers users can't see some of the information shown here, like for example the profit margin.

In this component the Marimekko and the grouped column charts have clickable labels on the x-axis that redirect to the corresponding user. As the space is reduced and names can be very long we decided to display the user id's instead.

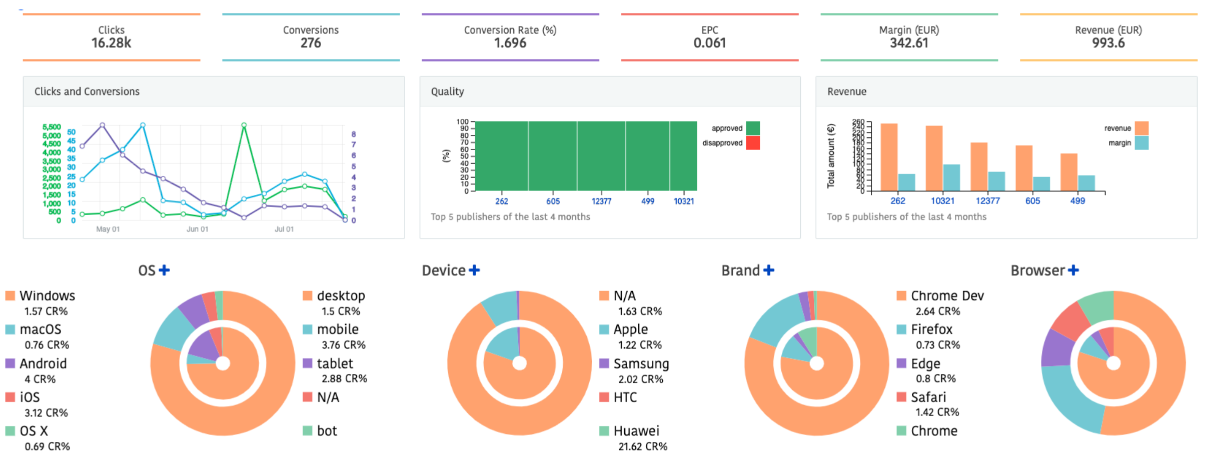

Figure 16: Performance component

5 Project Management

In this section we will explain the working methodologies and the tracking tools used for a proper project management.

5.1 Methodology

The working methodology chosen for this project is based in the agile framework named SCRUM. This methodology consists in an incremental development of small deliverable parts of the project that have to be done in a short interval of time, usually two weeks.

We chose SCRUM because we work side by side with most of our stakeholders and we know that the requirements they gave us the first day will have nothing to do with their expectations at the end of the project. Working in an agile environment improves project's flexibility letting us react to changes before it's too late, that way we can not only avoid big deviations in the temporal planning, but also deliver a final product that fulfills the real requirements.

Even though this is an individual project it has implications in other projects being developed in the company but as the IT team already worked with an agile methodology we didn't have any problem.

5.2 Tracking tools

The main tools used to assure an efficient communication during the whole project were the email or Skype ,to contact the stakeholders that work in the company and the tutor of the project.

For a better communication inside the IT team we used Slack, a cloud-based set of proprietary team collaboration tools.

We also used Git as the version control system, and an online repository named Bitbucket which besides from tracking the commits or creating pull requests, it allowed us to define issues, assign them to the members of the IT team and change their status.

The combination of Slack and Bitbucket helped us a lot with the monitoring of changes in the source code. We defined some triggers in Bitbucket to receive notifications in the Slack chat.

6 Temporal planning

This section describes the tasks that have to be done and the resources needed to execute them. Each task will be given a certain amount of time according to the complexity and based on the personal experience in order to be able to successfully finish the project in the desired time frame. The temporal planning also comprises the tasks dependencies with a Gantt's diagram and the possible deviations that may happen and how they can affect the project.

6.1 Project stages

The whole project is divided in three stages. Each stage consists in a group of tasks that will contribute to the achievement of a specific milestone:

- **Project conception and planning**: is the first stage and it consists in understanding the reason behind the project, usually a problem or a business goal, it's feasibility and expected outcome. This phase also defines the route map of the project.

  We can divide it in two tasks:

  - **Project brief**
  - **Project planning**

- **Project execution**: the project execution phase comprises all the actual work that has to be done and it's formed by 5 tasks:

  - **Study of the tech stack**
  - **Study of the company business**
  - **Set up of development environment**
  - **Implementation of data visualizations**
  - **Testing and evaluation**

- **Project closure**: the closure of the project is the last phase. Even though the project is finished and running, a final documentation wrapping up all the work has to be done. This serves as a retrospective that may help us to find improvement opportunities for future projects.

  - **Final memory**

6.2 Tasks Description

The tasks mentioned before and the resources needed for each one are described as follows.

6.2.1 Project brief

This task will result in the first document of the project.It's an overview of the project, a brief description of its needs and general purpose. It's the first step to understand what the project entails.

In order to do this task we will need to meet the director of the project.

After finishing this project brief, we can successfully explain what are we going to do despite we don't necessary know how yet. Registering the thesis can only be done after this first task.

Resources: We only need a computer with a word processor, in our case we preferred the web-based software named Google Docs because of it's simplicity and the possibility to access the documents from the internet.

6.2.2 Project planning

This task consists in creating a document defining the detailed plan we will follow from beginning to end.

It's important to note that even well-panned projects can fail, but an unplanned project will never succeed. Therefore a good project planning it's crucial for maximizing the success probabilities.

This task is done in the GEP subject, and it's also divided in three smaller parts, each one with representing a deliverable:

- **Project scope and contextualization**
- **Temporal planning**
- **Budget and sustainability**

At the end of GEP all these deliverables are wrapped up and will be used again to create the final documentation in the project closure phase.

Resources: For this task we need a computer with internet connection, Google Docs and the planning software Gantter[5].

[5]Gantter: A cloud-based project management software that allows you to create and edit project schedules and gantt charts

6.2.3 Study of the company business

This task consists in understanding the affiliate marketing business. The knowledge we need to create valuable solutions comprises not only technical concepts and terminologies, but also the market contextualization to detect the gaps and opportunities.

Resources: For this tasks we need a computer with internet access.

6.2.4 Study of the tech stack

This task is about acquiring the level of knowledge necessary for successfully implementing the solutions needed. The project's main stack has been also mentioned before, and it consists in the Elastic stack, the AngularJS framework and the D3 library. However we also need to have a minimum understanding of Docker containers and the Webpack module bundler as they are part of the development environment.

Other technologies and languages needed like Git, HTML or PHP aren't contemplated in this task as we already had the knowledge necessary before starting this project.

Resources: For this task completion we need a computer with access to the internet.

6.2.5 Set up of development environment

This may seem trivial but setting up all the software and tools needed for the developing process also takes time. Based in previous experiences in smaller projects we can figure out that the setting up for this particular project will be long enough to consider it a task and add it to the schedule.

We will install and set up the Docker containers, the PhpStorm IDE with the Xdebug debugger, the Postman ADE, the Slack client, the Sequel Pro database manager and all the dependencies required for these programs.

We need to download the latest version of the source code from Bitbucket and install all the NPM modules required because we will be extending an existing web application. A local copy of the live MySQL database from the Google Cloud instance will be the last thing needed for our development environment.

Resources: For this task we need a computer with internet access that fulfills all the requirements of the software we will use.

6.2.6 Implementation of data visualizations

This is the biggest task of the project and encompasses all the work of the actual implementation of the visualizations.

For each chart visualization created we can differentiate three main phases. We decided not to divide this task in a smaller granularity (one task per phase) due to our agile approach which consisted in the incremental implementation of each visualization to release functional charts on each iteration. This way we received feedback much faster than if we first finished all the first phase, before starting the second one and so on.

The three phases are:

- **Data search**: create the queries DSL to retrieve the data from Elasticsearch. Due to our needs of creating reusable dynamic queries, we implemented a custom PHP library for this purpose.
- **Data processing**: post-processing the raw data coming from Elasticsearch and map it to the desired format for the front-end.
- **Data visualization**: the actual front-end visualization components which display the processed data.

For a deep dive into these phases go to section **4.1 Implementation phases**.

Resources:For this task we need a computer with all the tools from the previous task, and access to the instances running Elasticsearch.

6.2.7 Testing and evaluation

This task consists in testing the visualizations previously implemented. This testing doesn't consist in debugging the code (that should be done during the implementation task), these tests are for measuring the user acceptance (UAT[6]) and evaluating that they fulfill all the requirements.

Resources: For this task we need the development environment previously configured and users that have never used our visualizations before.

6.2.8 Final memory

In this task we are going to prepare the final deliverable of the project. It's in this last task where we will write the final documentation of the project and also prepare the defence of the thesis.

Resources: For this task we need a LaTeX[7] editor for writing a proper memory and a presentation software for creating the thesis presentation. We chose the online editor named *Overleaf* and *LibreOffice* but any others could work as well.

[6]UAT:User Acceptance Test are the ultimate phase of software testing process. During UAT, real end-users test the software to make sure it fulfills the requirements by executing tasks in real-world scenarios

[7]LaTeX: a high-quality typesetting system that enables professional preparation of various types of texts. Texting in LaTeX is most similar to programming.

6.3 Tasks dependency

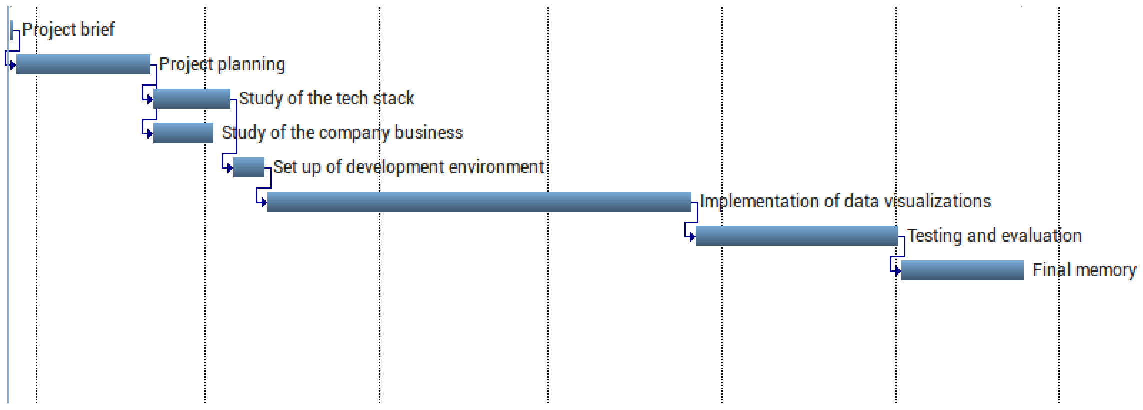

Figure 17: Gantt's diagram of the project

6.4 Temporal estimation per task

The table below shows the amount of hours we estimated for each task to be completed.

Task	**Estimated hours**
Project brief	4
Project planning	70
Study of the tech stack	40
Study of the company business	30
Set up of development environment	16
Implementation of data visualizations	210
Testing and evaluation	105
Final memory	65
Total	**540**

Table 1: Temporal estimation per task

6.5 Resources

We can divided the resources needed in three groups: software, hardware and human resources.

6.5.1 Software resources

- **Operative system**: the operative system used is MacOS version 10.14, also known as Mojave (Liberty).
- **Development software**: we used the *IntelliJ PhpStorm* as our IDE and the PHP debugger named *Xdebug*. Other software required for the development are *Postman* used with the Elasticsearch API, and *Docker* containers.
- **Communication**: the communication tools used were *Slack, Skype* and *Apple Mail*.
- **Planning**: we used Gantter for task planning.
- **Version control**: we used Git as our version control system with a Bitbucket online repository.
- **Other**: the Overleaf online text editor and the Google Docs online word processor.

6.5.2 Hardware resources

- **Laptop**: MacBook Pro 13
- **Extra screen**: BenQ GL2580H 24.5
- **Keyboard**: Magic Keyboard
- **Mouse**: Magic Mouse 2
- **Cloud hardware**: servers and virtual machines from Google Cloud.

6.5.3 Human resources

- **Developer**: The responsible of doing the project.
- **Project director and academic tutor**: They help and guide the developer during the whole project.
- **Support developers**: other members of the IT team that may give some advise and help to the main developer.

6.6 Analysis of project deviations

Despite the planning of the project and the temporal prevision for each tasks are done very carefully, some deviations may occur. It's difficult to prevent all the possible problems and obstacles that a project can face, but we will try to minimize their impact by adding some extra time to the real estimation in the temporal planning in order to have some margin in case of deviations.

The implementation task is more likely to suffer delays than any other because of it's complexity, that's also why we increased the margin in this tasks more than in any other.

If these prevention measures aren't enough and the tasks need more time than the expected we will have to change the scope and reduce it in order to be able to deliver a smaller but still decent project.

On the other hand, if a task is finished before the expected time, we will start with the next one and, if we still have a big margin of time in the final tasks we can consider to change our scope and include new tasks.

6.7 Temporal planning final retrospective

After the project closure, the comparison between the initial temporal planning and the real time spent show us that the overall estimation was quite accurate thanks to the margins we added to prevent possible deviations.

Despite we consider the initial planning a success, some deviations occurred.

The differences of the expected amount of hours necessary for each tasks and the real ones are shown in the table below.

Task	**Estimated hours**	**Real hours**	**Observations**
Project brief	4	4	Successfully completed on time
Project planning	70	70	Successfully completed on time
Study of the tech stack	40	40	Successfully completed on time
Study of the company business	30	30	Successfully completed on time
Set up of development environment	16	20	Complications mostly setting up docker and Xdebug
Implementation of D3 visualizations	210	220	Complications with a requirement change, implementation of Marimekko chart
Testing and evaluation	105	135	User testing done in production environment instead of locally (had to wait for deployment day)
Final memory	65	75	Restructuring documentation and using LaTeX
Total	**540**	**584**	**Delayed a bit more than one week**

Table 2: Temporal deviations per task

7 Economic analysis

In this section we will study the budget estimation for the project and the possible deviations we should be aware of.

7.1 Budget estimation

The budget estimate can be divided into two sections according to the nature of the planned costs.

7.1.1 Direct costs

Direct costs are the expenses that we can relate to a specific object, in our case this project.

Amortization *= (Price - Residual value)/ 7 months*

*Cloud Services: There are three instances in Google Cloud directly related to this project, two of them cost 380 €/month each one and the other one 53 €/month. As our project lasts 7 month the price is calculated as: (380x2+53)x7=5691 €

**Intern Developer: The price is calculated as the total hours estimated for this project multiplied by the price per hour indicated by the university agreement: 540 hours * 8 €/hour = 4320 €

	Concept	**Units or Hours**	**Useful life (years)**	**Price**	**Residual value**	**Amortization per month**
Hardware costs	MacBook Pro 13	1	5	1.500 €	1.000 €	72 €
	BenQ GL2580H 24.5	1	8	130 €	90 €	6 €
	Magic Keyboard	1	4	100 €	60 €	6 €
	Magic Mouse 2	1	4	85 €	45 €	6 €
	Cloud services*	-	-	5.691 €	-	813 €
Software costs	PhpStorm	1	1	0 €	-	-
	Postman 24.5	1	-	0 €	-	-
	Docker	1	-	0 €	-	-
	Slack	1	-	0 €	-	-
	Bitbucket	1	-	0 €	-	-
	Elastic stack	1	-	0 €	-	-
Human costs	Intern Developer**	540	-	4.320 €	-	-
	University taxes	1	-	1.216 €	-	-
Total	-	-	-	**13.042 €**	-	**1.077 €**

Table 3: Direct costs estimation

7.1.2 Indirect costs

The indirect costs go beyond a particular object or product, in our case the indirect costs are the costs of the company that are not strictly related with our project. We are not going to list all the expenses of the company as it's out of the scope and it's private information that the company won't provide us, but we will list some costs that they did give us and which are somehow related to the project.

Concept	**Price/Month**	**Total price**
Cloud Architecture support*	200 €	1.400 €
Total	**200 €**	**1.400 €**

Table 4: Indirect costs estimation

7.1.3 Total costs

Concept	**Price**
Direct costs	13.042 €
Indirect costs	1.400 €
Total	**14442 €**
Total with 5% contingency	**15.164,10 €**

Table 5: Total costs estimation

7.2 Possible budget deviations

The possible errors or deviations that our budget estimation may suffer can only happen in the human costs section of the direct costs because all the other concepts are real values as all the hardware has been purchased before the start of the project and the software licenses are either open-source or free when using a student license.

In order to cover unexpected costs we increased our budget estimation a 5%. Usually the contingency budget is calculated as the 15-20% of the total budget but in our case we have very low possibilities of deviations, therefore we decided that a 5% for contingencies should be enough.

7.3 Budget final retrospective

After finishing the project we can confirm that our initial budget was very accurate. We had a 5% margin for unexpected costs and the final costs ended up being less than a 2,5% higher than our initial estimation without adding the contingency budget. Therefore we covered this deviation using only half of the the contingency budget.

The increase in the final costs can be explained because of the delay suffered on the completion of some tasks, mostly during the testing and evaluation task, which increased the total amount of hours of the developer from 540 to 584 which resulted in 352 € more.

8 Sustainability

In this section we will study the sustainability of the project by addressing to economic, environmental and social dimensions.

The following table shows the final scores of each dimension. We believe the overall results are good because even though this project it's not intended to improve the user's life quality, it has a high economical feasibility yet keeping it's environmental footprint very low.

Economic	**Environmental**	**Social**
8	8	6

Table 6: Sustainability matrix results

8.1 Economic dimension

The estimated costs seem to be really high for this project but we need to emphasize that more than half of these costs, almost a 64%, are indirect costs mostly coming from the office rent which is used by the whole company.

Regarding the direct costs, all the hardware specified it's already owned by the company so despite they are still direct costs, they can't be considered an extra investment for the project, being cloud services the only exception. On the other hand, most of the software used are free-open-source projects excepting the PhpStorm IDE but we managed to get a one-year-free license applying for the JetBrain Student Pack.

Considering all these statements, we can conclude that the real investment of the company in this project is reduced to two main costs, the cloud services mentioned before and the human costs which altogether sum a total of 11.227 €.

The company preferred not to make their finances public therefore we can't estimate the ROI[8], however we should emphasize that the return of this project it's not strictly related with the company's income but with their customer's satisfaction.

[8]ROI: Return of investment, a performance measure used to evaluate the efficiency of an investment.

8.2 Environmental dimension

It's very difficult to estimate the environmental impact of the project as it's not related to any industrial process or has a physical application.

The main impact comes from the local hardware that will be used during the development and also all the hardware provided by the cloud services that will be running it.

All the local hardware can be reused for other projects and purposes. Furthermore,the only peripheral that requires the use of batteries is the *Magic Mouse 2* but the company provided us with rechargeable AA batteries.

Regarding our cloud services provider, we use Google Cloud, the most sustainable provider and winner of an A grade in the latest ClickingClean[13] report. Google Cloud is the world's largest corporate buyer of renewable power, and they already reached the 100% of renewable energy consumption in 2017, which means that the net emissions directly associated with our cloud computing and data storage is zero.

8.3 Social dimension

The social impact of this project can't be directly related with an improvement of the user's life quality. Yet, providing them with tools to analyze their data, we can help them to understand their traffic better, which should lead them to an income increase.

In addition, even though the publication of the source codes of the chart visualization components and the custom DSL-query library are not planned for the immediate future, the company confirmed that they are considering it as a very feasible option.

9 Requirements evaluation

In this section we will discuss the results of evaluating the quality and success on our goal of accomplishing the project's requirements.

The table below is the breakdown of the total score that we obtained by adding the scores of each requirement weighted by the priority we gave them during the project.

Requirement	**Score**	**Weight**	**Weighted score**
Usability	8	0.3	2.4
Extensibility	8	0.1	0.8
Maintainability	7	0.1	0.7
Efficiency	8	0.2	1.6
Value	8	0.3	2.4
Total	-	-	7.9

Table 7: Requirements evaluation scores

The total score sums 7.9 out of 10 which despite not being a perfect score we still think is a big success considering that we didn't have previous experience in almost any of the technologies used to develop this project.

All the individual scores of the requirements are justified as it follows.

9.1 Usability

Usability was one of the most important requirements during all the project development because we knew that otherwise the users wouldn't use the tools and none of the other requirements would matter.

We put a lot of effort in choosing the visuals wisely, not only looking for the ones that suited the data best or that provided more information but also considering that, as no explanation would be provided to the users, they had to be very intuitive.

After receiving the feedback from the UAT we found out that the overall tools were easy to use and understand, except from one specific visualization which most of them agreed it was too complicated and therefore they wouldn't use it. This visual was the scatter plot (figure 9), so we decided to exclude it from the next releases.

9.2 Extensibility

We evaluated the extensibility requirement as the level of data abstraction achieved, in other words, the capability to represent different types of data on the same visual component without changing its implementation.

We gave this section an 8 out of 10 because even though we used the same components to represent many different data without changing anything, in some cases the bindings and the data structure expected from the component were a bit more complex than what we wanted.

9.3 Maintainability

This requirement ended up having the lowest score in comparison to the others. Despite our focus in writing simple code, in some cases the complexity of the methods didn't allow us to, for example in the implementation of the Marimekko chart.

We also evaluated the quality of the documentation written by asking other developers from the IT team to use some of the components without looking at the implementation, only by reading the documentation. Although we received good feedback, we think it would have been better to use documentation tools instead of just writing markdown files.

9.4 Efficiency

The efficiency requirement score has been chosen by looking both the response time of the data requests and the actual Elasticsearch execution of the DSL queries.

We focused on creating data visualizations with acceptable loading times by applying as many filters as possible to the Elasticsearch query, and in some cases we restricted the time frame filters allowed.

The post-processing of the data is also important as sometimes extra data needs to be retrieved from relational databases to complete the information needed.

With all these in mind we concluded that the components are efficient enough to deserve a score of 8 out of 10.

9.5 Value

As we mentioned before, this project belongs to a private company, therefore it makes sense that this requirement is the most important together with usability. It's true that creating valuable tools is strictly dependent of the other requirements, but it's important to note that all the other requirements can be fulfilled and still not be valuable for the company, for example because the data is wrong or meaningless.

We gave this requirement a score of 8 out of 10 because we consider that all the final visualizations released will increase the value of the platform but we still prefer to be skeptical as this will be better evaluated in a longer term.

10 Conclusions

The representation of information is a topic more complex than what I initially thought, it comprises not only interface and design concepts but also the representations characteristics and constraints, for example if they introduce bias to the data set, in which cases and how to avoid or minimize them, which visualization to choose depending on what we want to compare,etc. During the project I learned a lot about this topic also because it's something we've never seen in the university despite the increasing importance of data analysis.

On the other hand, prior to representations, we have all the work of obtaining the data and processing it to convert it in business-valuable information. It's in this part where I learned cutting edge technologies. The use of the Elasticsearch search engine and the Elastic stack in general has given me practical knowledge in the storage and processing of data in a completely different paradigm to the relational databases we have seen in the degree.

Regarding the project, the objectives have been achieved and the requirements met successfully. We have created a php library that allowed us to build requests to Elasticsearch maximizing good practices and facilitating testing processes and debugging work. We implemented methods for post-processing Elasticsearch results and in some cases combining them with information stored in MySQL databases. Finally we created the visual components and decided exactly in which sections of the web application to add them, for what type of users, etc.

We consider the project planning was very accurate, despite we had a small deviation on the time required for the completion of the implementation and testing tasks which also led to an increase on the estimated budget,as we mentioned before in the corresponding section (7.3).

References

[1] ITU. *Measuring the Information Society Report 2018*. Annual report. Geneva, Switzerland: International Telecommunication Union, 2018.

[2] Kepios Pte. Ltd., We Are Social Ltd., and Hootsuite Inc. *Digital 2019: Global Digital Overview*. Annual report. 2019.

[3] R.E. Freeman. *Strategic Management: A Stakeholder Approach*. Pitman Publishing Ltd, 1984.

[4] Elastic company. *ELK Stack: Elasticsearch, Logstash, Kibana*. [Online; accessed April-2019]. URL: `https://www.elastic.co/what-is/elk-stack`.

[5] Elastic company. *Elasticsearch: RESTful, Distributed Search Analytics*. [Online; accessed April-2019]. URL: `https://www.elastic.co/products/elasticsearch`.

[6] Elastic company. *Beats: Data Shippers for Elasticsearch*. [Online; accessed May-2019]. URL: `https://www.elastic.co/guide/en/elasticsearch/reference/6.4/glossary.html`.

[7] Elastic company. *Beats: Data Shippers for Elasticsearch*. [Online; accessed April-2019]. URL: `https://www.elastic.co/products/beats`.

[8] Elastic company. *Kibana*. [Online; accessed April-2019]. URL: `https://www.elastic.co/products/kibana`.

[9] Elastic company. *ELK Stack: Elasticsearch, Logstash, Kibana*. [Online; accessed April-2019]. URL: `https://www.elastic.co/what-is/elk-stack`.

[10] B. Green and S. Seshadri. *AngularJS*. O'Reilly Media, Inc., Apr. 2013.

[11] Mike Bostock. *D3.js - Data-Driven Documents*. [Online; accessed May-2019]. URL: `https://d3js.org/#introduction`.

[12] E.R. Tufte. *The Visual Display of Quantitative Information*. 1983.

[13] Greenpeace. *Clicking Clean*. Annual report. Washington D.C., United States: Greenpeace Inc., 2017.

[14] Alex Birkett. *How to Deal with Outliers in Your Data*. [Online; accessed May-2019]. URL: `https://conversionxl.com/blog/outliers/`.

[15] Peter Cook. *In depth information on D3.js*. [Online; accessed May-2019]. URL: `https://www.d3indepth.com/`.

Appendices

A Elasticsearch document example

```
  {
"_index": "tracking-000005",
"_type": "doc",
"_id": "LGKAhwWRHeaNcBbV6cspIssLrzoykzJ7taeCcGh4",
"_version": 1,
"_score": null,
"_source": {
  "os": "Android",
  "httpv": "1.1",
  "ca": null,
  "sid": [],
  "ct": 1566369909580,
  "apid": "1372481",
  "response": "302",
  "crid": null,
  "pid": "16272",
  "host": {
    "name": "frontend-tracking-autoscaler-66vt"
  },
  "lt": "2019-08-21T06:45:09.000Z",
  "l": "en-US",
  "verb": "GET",
  "det": "mobile",
  "deb": null,
  "chid": "18460",
  "osv": "8.1.0",
  "ci": "BD",
  "co": null,
  "aid": "1941",
  "input": {
    "type": "log"
  },
  "ip": "103.67.157.XXX",
  "pop": false,
  "h": 6,
  "dem": "CPH1803",
  "@timestamp": "2019-08-21T06:45:10.034Z",
  "tpp": {
    "add1": "06dc7ef7aaf5408c8adfd21bfe4c233a"
  },
  "r": null,
  "cn": "Bangladesh",
  "isp": "tobedone",
  "cia": "LGKAhwWRHeaNcBbV6cspIssLrzoykzJ7taeCcGh4",
  "b": null,
  "bv": null,
  "@version": "1",
  "cid": "17033",
  "request": "/go.html?a1=j&ad=224HLEZ3&add2=1046_8551&add1=06
      dc7ef7aaf5408c8adfd21bfe4c233a&add_ref="
},
"fields": {
    "lt": [
      "2019-08-21T06:45:09.000Z"
    ],
    "ct": [
      "2019-08-21T06:45:09.580Z"
    ],
    "@timestamp": [
      "2019-08-21T06:45:10.034Z"
    ]
  },
  "sort": [
    1566369909580
  ]
}
```