UNIVERSIDAD
REY JUAN CARLOS

INGENIERÍA SUPERIOR DE TELECOMUNICACIONES +

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso Académico 2016/2017

Trabajo Fin de Carrera

3D Charts visualizations in Kibana

Autor : Viorel Rusu Tutor : Dr. Jesús M. González Barahona

Proyecto Fin de Carrera

FIXME: Título

Autor : FIXME

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 20XX

Dedicado a

mi familia / mi abuelo / mi abuela

II

Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Contents

List of Figures

Chapter 1

Introduction

We live in a world of data. Data of all sorts: sensors, logs, statistics, different company data. Specifically, in the world of software projects, we also start to have thousands and thousands of data Terabytes. We should not be surprized, a single software project may have millions of lines of code, written by thousands different programmers, having contributions from houndreds of different companies, with thousands of different versions... ok, you get the idea. The problem is we, humans, lose perspective if we only observe that raw data, and it becomes completely useless. We just can't see the forest for the trees.

1.1 The problem

From the previous paragraph, it seems obvious that what is needed is to get simpler data from that huge amount of data. What is needed is to analyze that data in some way, organize and show it so we, humans, could have an eagle view just by a glance. That way, we could draw conclusions about that information and make better decisions, improving our world.

In the last years, many different projects have arisen with this goal in mind: provide tools that help us analyze huge amounts of data and represent it somehow. We are talking about projects such as business intelligence projects, data mining, real-time analytics and visualizations. Some of them are real Big Data projects, some of them don't have in scope such a big amount of data so we can't call them that way.

Specifically, what we want is to build different visualizations, or dashboards, of our software development data. We are applying our data analysis and visualizations to software projects in

particular, but really we could apply it to any other kind of data. Out there we can find many tools to achieve our goals. There are many different solutions to the mentioned problem of respresenting data. What we want is a solution that lives in the browser. Somehow, we want the final user to interact with his browser and seeing easily different visualizations of his own data, in a new way it hasn't been shown before. We want to make our contribution to this world and make it available for everyone, we want it to be open source.

1.2 The solution

We decided to use Kibana[1]: an open source analytics and visualization platform designed to work with Elasticsearch[2]. Elasticsearch is a highly scalable open-source full-text search and analytics engine. It allows us to store, search, and analyze big volumes of data quickly and in near real time.

Kibana is a very new technology, fully under development by the time this project was realized. It provides single visualizations of different types, like pie or bar charts. Then, it is possible to save and load this visualizations in a dynamic dashboard. This is an interactive dashboard, allowing us to move each separate visualization around, and applying different filters to all data just by interacting with a single visualization for example. This makes it really easy to understand large volumes of data. We can quickly create dynamic dashboards that display changes to Elasticsearch queries in real time. All this features made Kibana a very good choice.

The problem with Kibana is that it has a very limited number of visualizations. Kibana team is aware of this problem and they encourage independent developers to develop their own visualizations and make it public to the community. And this is achieved developing a plugin for Kibana. Official documentation on how to create a custom plugin is poor, but luckily independent developers have succeeded and published the process. With this information, and the information we can obtain of standard plugins by reverse-engineering, reading source code, we are able to create a new plugin.

We saw that basic 2D charts are already available in Kibana, and other developers were already working in adding more variety to it. So a good idea would be to represent the data in

[1] https://www.elastic.co/products/kibana
[2] https://www.elastic.co/products/elasticsearch

a whole new way for Kibana, under the form of 3D charts. There are not many 3D libraries out there written in Javascript that allow us to represent 3D functional data, and we wanted to promote the use of a new library a student at Universidad Rey Juan Carlos developed as his thesis, being in touch with the developer in order to add new functionality and report bugs.

In order to sum it up, we can express our solution with the following words: it is going to take the form of an easy-install plugin fully-integrated in Kibana, containing different 3D visualizations.

1.3 Objectives

1. - Use Kibana integrated tools to retrieve the data we need from elasticsearch.
2. - Build this new 3D visualizations: pie chart, bars chart, bubbles chart
3. - Integrate threedc.js 3D scenes in Kibana visualization
4. - Integrate threedc.js visualization in Kibana dashboard
5. - Add custom events to threedc charts in order to filter data on click
6. - Help to improve threedc.js library by reporting bugs, adding an interface for custom data and for custom events.

1.4 Structure of this paper

En esta sección se debería introducir la esctura de la memoria. Así:

- En el primer capítulo se hace una intro al proyecto.
- En el capítulo **??** se muestran los objetivos del proyecto.
- A continuación se presenta el estado del arte.
- ...

TODO: mencionar soluciones parecidas a la mia, con otras tecnologías o que hay por ahi

Chapter 2

Used technologies and Context

2.1 HTML5

HTML5 is a markup language used for structuring and presenting content on the World Wide Web. It was finalized, and published, on 28 October 2014 by the World Wide Web Consortium (W3C) This is the fifth revision of the HTML standard since the inception of the World Wide Web. The previous version, HTML 4, was standardized in 1997.

Its core aims are to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (web browsers, parsers, etc.). HTML5 is intended to subsume not only HTML 4, but also XHTML 1 and DOM Level 2 HTML.

In particular, HTML5 adds many new syntactic features. These include the new video, audio and canvas elements, as well as the integration of scalable vector graphics (SVG) content (replacing generic object tags) and MathML for mathematical formulas. These features are designed to make it easy to include and handle multimedia and graphical content on the web without having to resort to proprietary plugins and APIs. Other new page structure elements, such as main, section, article, header, footer, aside, nav and figure, are designed to enrich the semantic content of documents. New attributes have been introduced, some elements and attributes have been removed and some elements, such as a, cite and menu¿ have been changed, redefined or standardized. The APIs and Document Object Model (DOM) are no longer afterthoughts, but are fundamental parts of the HTML5 specification.HTML5 also defines in some detail the required processing for invalid documents so that syntax errors will be treated uniformly by all

conforming browsers and other user agents.

2.2 Javascript

JavaScript is a high-level, dynamic, untyped, and interpreted programming language.It has been standardized in the ECMAScript language specification. Alongside HTML and CSS, it is one of the three essential technologies of World Wide Web content production; the majority of websites employ it and it is supported by all modern Web browsers without plug-ins.JavaScript is prototype-based with first-class functions, making it a multi-paradigm language, supporting object-oriented,imperative, and functional programming styles.It has an API for working with text, arrays, dates and regular expressions, but does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.

Despite some naming, syntactic, and standard library similarities, JavaScript and Java are otherwise unrelated and have very different semantics. The syntax of JavaScript is actually derived from C, while the semantics and design are influenced by the Self and Scheme programming languages.

JavaScript is also used in environments that are not Web-based, such as PDF documents, site-specific browsers, and desktop widgets. Newer and faster JavaScript virtual machines (VMs) and platforms built upon them have also increased the popularity of JavaScript for server-side Web applications. On the client side, JavaScript has been traditionally implemented as an interpreted language, but more recent browsers perform just-in-time compilation. It is also used in game development, the creation of desktop and mobile applications, and server-side network programming with runtime environments such as Node.js.

These are Javascript main features:

- Imperative and structured: JavaScript supports much of the structured programming syntax from C (e.g., if statements, while loops, switch statements, do while loops, etc.). One partial exception is scoping: JavaScript originally had only function scoping with var. ECMAScript 2015 adds a let keyword for block scoping, meaning JavaScript now has both function and block scoping. Like C, JavaScript makes a distinction between expressions and statements. One syntactic difference from C is automatic semicolon insertion,

which allows the semicolons that would normally terminate statements to be omitted.

- Dynamic: As with most scripting languages, JavaScript is dynamically typed; a type is associated with each value, rather than just with each expression. For example, a variable that is at one time bound to a number may later be re-bound to a string. JavaScript supports various ways to test the type of an object, including duck typing. JavaScript includes an eval function that can execute statements provided as strings at run-time.

- Prototype-based (Object-oriented): JavaScript is almost entirely object-based. In JavaScript, an object is an associative array, augmented with a prototype (see below); each string key provides the name for an object property, and there are two syntactical ways to specify such a name: dot notation (obj.x = 10) and bracket notation (obj['x'] = 10). A property may be added, rebound, or deleted at run-time. Most properties of an object (and any property that belongs to an object's prototype inheritance chain) can be enumerated using a for...in loop.

  JavaScript has a small number of built-in objects, including Function and Date.

- Functional: A function is first-class; a function is considered to be an object. As such, a function may have properties and methods, such as .call() and .bind().A nested function is a function defined within another function. It is created each time the outer function is invoked. In addition, each nested function forms a lexical closure: The lexical scope of the outer function (including any constant, local variable, or argument value) becomes part of the internal state of each inner function object, even after execution of the outer function concludes.JavaScript also supports anonymous functions.

2.3 ElasticSearch

Elasticsearch is a distributed, scalable, real-time search and analytics engine. It makes it possible to search, analyze, and explore data at a speed and at a scale never before possible. It is used for full-text search, structured search, analytics, and all three in combination.

Elasticsearch is an open-source search engine built on top of Apache Lucene?, a full-text search-engine library. Lucene is arguably the most advanced, high-performance, and fully featured search engine library in existence today?both open source and proprietary. Elasticsearch

is written in Java and uses Lucene internally for all of its indexing and searching, but it aims to make full-text search easy by hiding the complexities of Lucene behind a simple, coherent, RESTful API.

2.3.1 Key concepts

- Document

  A document is a basic unit of information that can be indexed. This document is expressed in JSON. Within an index, as many documents as it's wanted may be stored. Although a document physically resides in an index, a document actually must be assigned to a type inside an index.

- Index

  An index is a collection of documents that have somewhat similar characteristics. An index is identified by a name (that must be all lowercase) and this name is used to refer to the index when performing indexing, search, update, and delete operations against the documents in it.

- Type

  Within an index, one or more types can be defined. A type is a logical category/partition of the index whose semantics is completely up to the user. In general, a type is defined for documents that have a set of common fields.

- Node

  A node is a single server that is part of the cluster, stores the data, and participates in the cluster?s indexing and search capabilities. Just like a cluster, a node is identified by a name which by default is a random Universally Unique IDentifier (UUID) that is assigned to the node at startup. Any node name can be defined if the default is not wanted. This name is important for administration purposes where it is needed to identify which servers in the network correspond to which nodes in the Elasticsearch cluster.

- Cluster

A cluster is a collection of one or more nodes (servers) that together holds the entire data and provides federated indexing and search capabilities across all nodes. A cluster is identified by a unique name which by default is ”elasticsearch”. This name is important because a node can only be part of a cluster if the node is set up to join the cluster by its name.

2.3.2 Data interaction

Elasticsearch provides a rich, flexible, query language called the query DSL, which allows us to build much more complicated, robust queries. The domain-specific language (DSL) is specified using a JSON request body. The queries and all interaction with elasticsearch has the next aspect:

```
1 GET /megacorp/employee/_search
2 {"query" : {"match" : {"last_name" : "Smith"}}}
```

Through these DSL queries any database operation can be performed. Of course, elasticSearch has APIs for the most known languages, so it is possible to make queries from other programming languages.

In this project, an elasticsearch server has been set up and all elasticsearch queries have been made indirectly through Kibana. This means that, by properly using Kibana functions and methods, it is possible to avoid manually having to make queries.

2.4 Kibana

Kibana is an open source analytics and visualization platform for Elasticsearch. It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster. Kibana is used to search, view, and interact with data stored in Elasticsearch indices. Advanced data analysis and visualization can be easily performed in a variety of charts, tables and map. Kibana makes it easy to understand large volumes of data.

Next, we will show basic Kibana functionality and its possibilities, as well as a basic set up.

TODO: Pegar las dos imágenes de Discover, visualize y el dashboard de bitergia. Explicar filtros, datos en la pestaña discover, las visualizaciones y el funcionamiento del dashboard.

2.5 AngularJS

2.6 webGL

WebGL (Web Graphics Library) is a cross-platform, royalty-free web standard for a low-level 3D graphics JavaScript API for rendering interactive 3D computer graphics and 2D graphics within any compatible web browser without the use of plugins. WebGL is integrated completely into all the web standards of the browser allowing GPU accelerated usage of physics and image processing and effects as part of the web page canvas. WebGL elements can be mixed with other HTML elements and composited with other parts of the page or page background.WebGL programs consist of control code written in JavaScript and shader code that is executed on a computer's Graphics Processing Unit (GPU). WebGL is designed and maintained by the non-profit Khronos Group.

WebGL is based on OpenGL ES 2.0 and provides an API for 3D graphics.It uses the HTML5 canvas element and the access is made using Document Object Model interfaces. Automatic memory management is provided as part of the JavaScript language.

The WebGL API may be too tedious to use directly without some utility libraries, which for example set up typical view transformation shaders (e.g. for view frustum). Loading scene graphs and 3D objects in the popular industry formats is also not directly provided for. JavaScript libraries have been built (or sometimes ported to WebGL) to provide the additional functionality.

As WebGL is an advanced technology designed to work directly with the Graphics Processing Unit, it is hard to code compared to other web standards which are more accesibles. This is the reason why many JavaScript libraries have appeared, to solve this problem. The most famous webGL library in terms of users is Three.js. It is light and not so complex compared with the original webGL specification. This is also the library that has been used in this project, and we will explain this library in the next section.

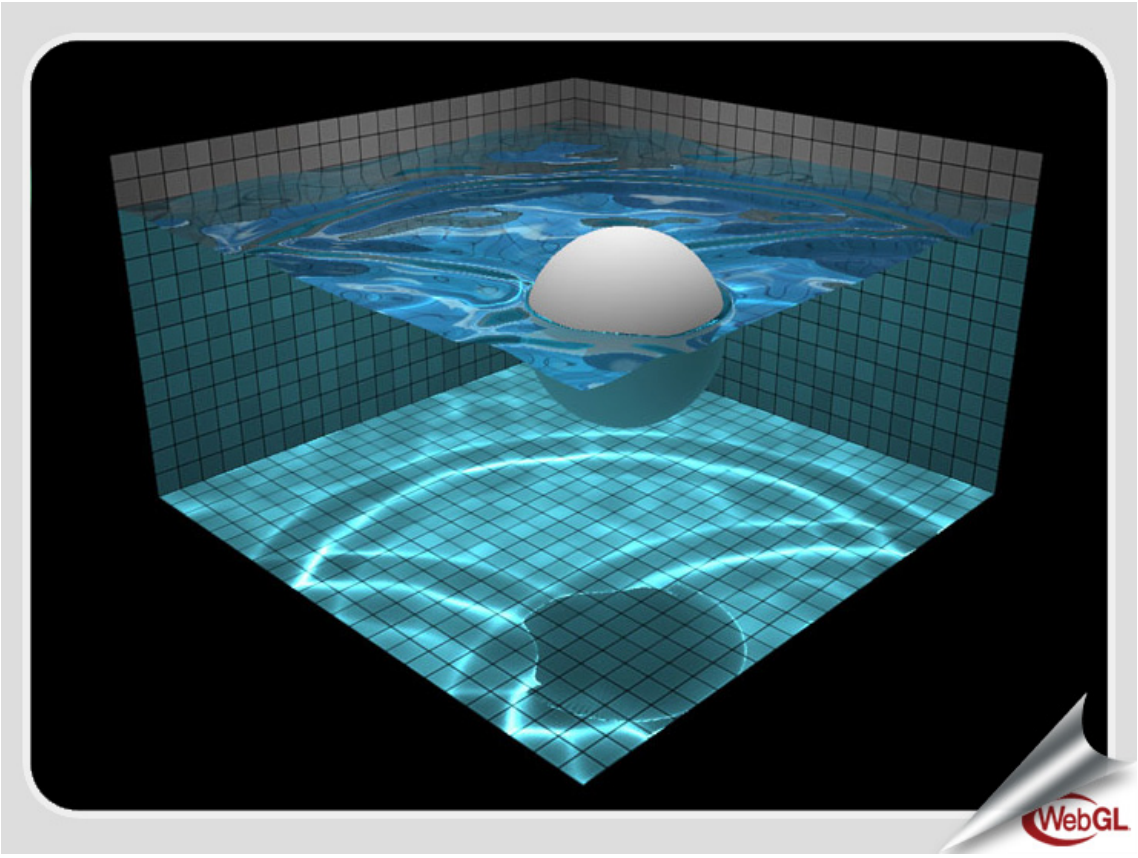

Figure 2.1: A webgl example - water simulation, a Chrome Experiment

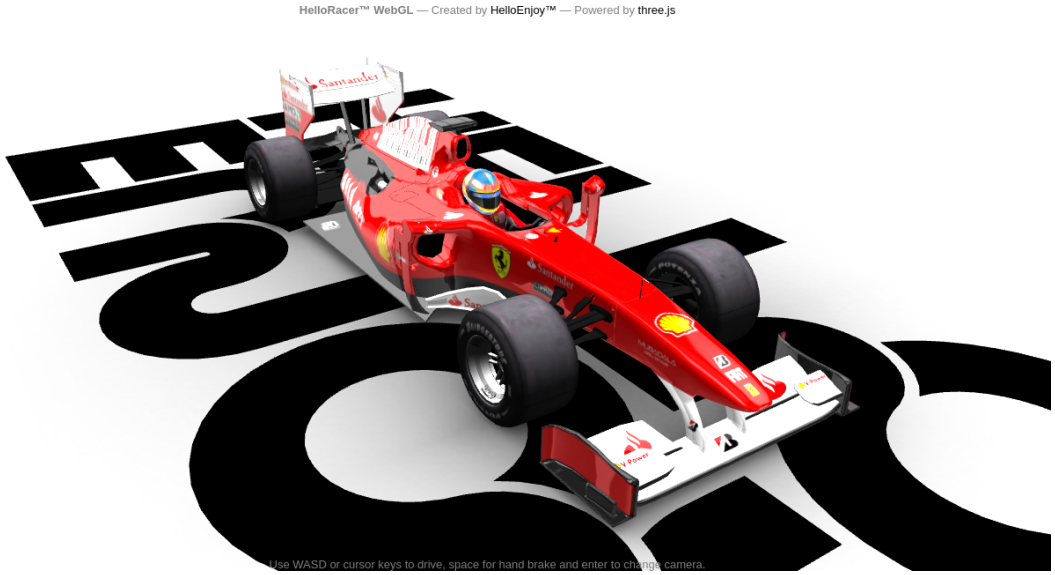

Figure 2.2: A three.js example - Helloracer game

2.7 Three.js

2.8 ThreeDC.js

2.9 Context

Mention ManyEyes, Freeboard, Carabel

Search for dashboards 3D - similar solutions - if there are

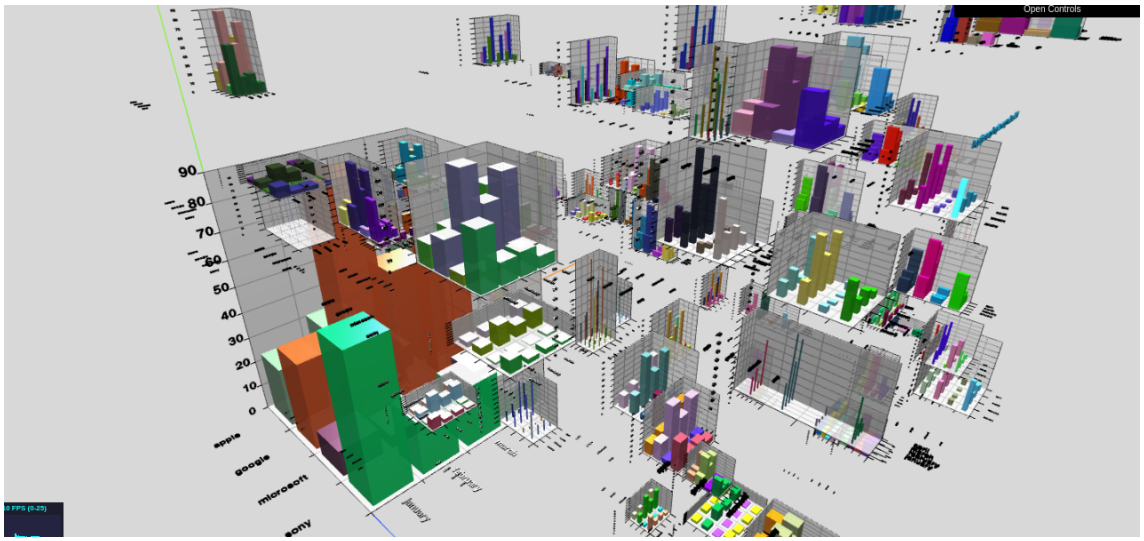

Figure 2.3: 3D charts built with threeDC

Chapter 3

Development

3.1 SCRUM Methodology

Scrum is an iterative and incremental agile software development framework for managing product development. In software projects, time optimization, team coordination, resources managing and task assigning is crucial in order to reach a final working product and specially fast working intermediate versions of the product. It is based on these intermediate versions delivery and offers great agility and flexibility, as small goals are set on the way, after each previous goal has been reached.

There are three basic roles in this methodology:

- **Product Owner**: represents the voice of the customer and those initially interested directly in the project. This role is in charge of defining the product and its functionality. He focuses on the business side of product development. He establishes and negotiates priorities and steers the product in the right direction.

- **Development Team**: responsible for delivering shippable increments at the end of each intermediate version of the product. They do all the technical work: analyse, design, develop, test and document the product.

- **Scrum Master**: ensures that the Scrum Methodology is correctly followed. He coaches the team and product owner on the scrum process and looks for ways to improve its implementability in that particular project. He also looks and resolves impediments and

Figure 3.1: Scrum workflow

distractions of the development team and keeps it focused on the key tasks. This role must not be mistaken with a project manager. A scrum master doesn't have people management responsabilities. A project manager doesn't really have a place in this methodology, the development team and scrum master are self-organizational.

In the Scrum Methodology, work is confined to repeatable work cycles known as sprints or iterations. Scrum is iterative and incremental. This means that product is always build on previous iteration, adding new features each time. During each sprint, the development team creates a potentially shippable product increment.

In each sprint a Sprint Backlog is defined. It collects all the tasks that are needed to be done and who is going to do them, as well as an estimation for the time needed for each task to be completed.

A natural question comes in mind at this point: can this methodology be at any help in a software project developed entirely by a single person? Although Scrum was not designed for this particular simple case, it can be very helpful. There won't be a Product Owner, a Scrum aster and a Team as such, but the methodology remains useful. This is because its main advantages are based on its flexibility and product control, and this is obtained even if the project is entirely developed by a single person! It provides control, adaptability and flexibility

independently of the size of the project and number of people involved.

In this project, following Sprints have been defined and developed:

1. **Sprint 0**: Investigation and technologies exploration
2. **Sprint 1**: First systems using elasticsearch, threeDC charts and Kibana custom plugins
3. **Sprint 2**: 3D Basic Pie Visualization
4. **Sprint 3**: Data treatment
5. **Sprint 4**: Kibana Integration

3.2 Sprint 0

3.2.1 Primitive dashboard

It is worth mentioning that the Kibana-elasticSearch solution was not clear from the beginning of this project. As the main objectives were to build a customizable dashboard living in the web browser, with open source projects data, a completely different solution was started. In this section, an overview of this paralell solution will be presented, even if this path has been abandoned on our way in order to embrace Kibana.

The solution we had in mind had the following architecture:

A MySQL database stores the information about software projects. A Django Server is set up in order to retrieve this information and serve it to the client as the browser demands it. Finally, the browser executes and plots the data. This is the basic workflow in this architecture.

The final result, in a basic version, can be seen in the next browser screenshot:

The technologies used to get to this solution were many. We will make a brief mention of the most important:

- Django Server: where all the application logic lives
- AngularJS as the front-end web application framework to provide a Single Page Application
- Twitter Bootstrap for the front-end design

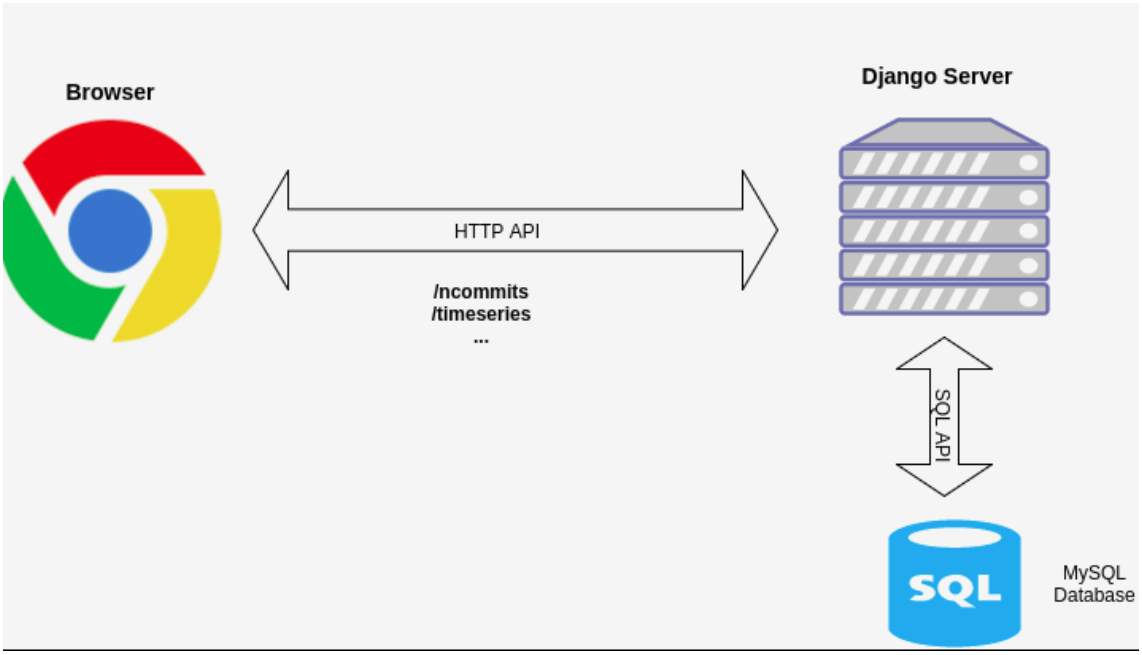

Figure 3.2: Proposed architecture

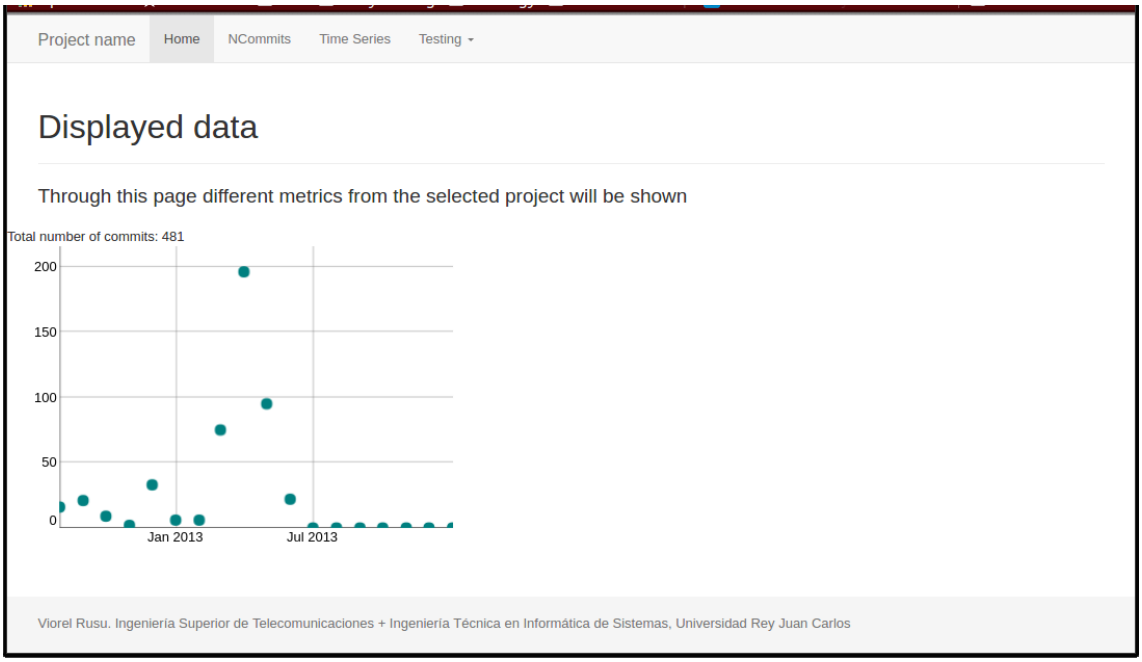

Figure 3.3: Simple browser visualization of commits timeseries

- MySQL as database
- Dygraphs as the Javascript library to plot our data

A dynamic dashboard was to be created using more Javascript libraries. This is a complex and tedious task. But most importantly, it is an unnecesary task. In the last years, great solutions have been given to this problem. Kibana is one of them. By worrying only about a certain visualization, if we do it well, Kibana will integrate it perfectly in its own dashboard, where this visualization lives side by side and interacts with other visualizations developed. This has great advantages:

- We can focus on data analysis and a visualization each time, as we don't have to worry too much about integrating all the visualizations in a dashboard. Kibana does this in a beautiful way.
- We can generalize the use of our product and give it a great platform where to live and be known. With Kibana, our database can be anything, not only a certain type of data with closed schemas.

  And this is enough reasons to abandon the primitive solution and decide to begin with Kibana. In the rest of this paper, we won't mention this primitive solution anymore, as it died on our way to a better solution. It provided a better comprehension of the problem and the different solutions and learning was important too, as many of these technologies and concepts are used in the rest of the project.

3.2.2 Setting up elasticSearch and Kibana

Once it was clear this project was about a Kibana plugin, we started getting familiarized with these technologies.

We will start with elasticSearch. I downloaded the latest client and started storing, exploring and modifying data. The download and installation is easy, it is just needed to download an elasticSearch version from https://www.elastic.co/downloads/elasticsearch. I opted for downloading the tar version. Once uncompressed, elastic search is run by executing bin/elasticsearch contained in the folder. This sets up an elasticSearch server, listening at port 9200. No additional configuration was needed.

ElasticSearch provides a REST API to interact with the database. Every operation with elasticSearch engine will be done through this API. We use 'curl' in order to make HTTP requests to a server from command line. For example, we use the following methods:

```
1 curl -XGET 'localhost:9200/_cat/indices?v'
2 curl -XPUT 'localhost:9200/customer?pretty': to create a new index
```

The first command shows all existing indexes while the second creates a new index in our elasticSearch database.

Once indexes are created, we can put documents inside them, like this:

```
1 curl -XPUT 'localhost:9200/customer/external/1?pretty
2 {
3   "name": "John Doe"
4 }'
```

Every operation or query we need to perform is done through this HTTP API.

These are basic operations. What we want is to work with a great amount of data. We can find test data for elasticSearch out there but we wanted to analyze specifically open source projects data. In my github[1] you cand find a JSON file with data about opnfv project in github, already in a elasticSearch format. We load this file into elasticSearch using taskrabbit tool for managing indexes: 'elastic-dump'[2]:

```
1 elasticdump --input=opnfv_git_es.json --output=http://localhost:9200/
    commits-index
```

This will be the index we will work with in the rest of the project.

Next, we install Kibana. We decided to install the development version directly, following the instructions that elastic team officially gives to developers in "CONTRIBUTING.md"[3]. The latest version was an alpha version of Kibana 5.0.0, and this will be the version we work through

[1] https://github.com/virusu/Docs/blob/master/opnfv_git_es.json.gz
[2] https://github.com/taskrabbit/elasticsearch-dump
[3] https://github.com/elastic/kibana/blob/master/CONTRIBUTING.md

the rest of the project. This is not such a simple process, and can bring some little descriptive errors like in my case when running kibana in development mode with ”npm start” inside kibana directory. This error was the following:

```
1 failed to watch files! Error: watch /home/vio/kibanadev2/src/plugins ENOSPC
      at exports._errnoException (util.js:870:11)
```

Investigating about the error, it was because Kibana run in dev mode requires watching a hundreds of files at the same time, looking for changes and recompiling everything to bring the new changes into the running kibana instance. What I did is to permanently modify the maximum notify watchers variable in sysctl:

```
1 echo fs.inotify.max_user_watches=524288 | sudo tee -a /etc/sysctl.conf &&
     sudo sysctl -p
```

Once Kibana runs correctly, the first time we access it it will ask us to configure an index pattern. We will do this and at this point everything is set up to start focusing on our plugin development.

3.3 Sprint 1

First systems with Kibana custom plugins, elasticSearch and threeDC charts

The main goal in this sprint is to build the first working systems with the main technologies involved in this project: Kibana, elasticSearch and threeDC. A separate and very basic server interacting with elasticsearch will be built in first place. We take advantage of having this server serving a page to also show a basic threeDC chart, killing two birds with one stone. In second place, Kibana will enter into action and it will act as the server, interacting with elasticsearch to show up some basic data by itself.

3.3.1 Separate server showing up ES data and a threeDC chart

This is the only section where we focus on anything separately from Kibana. What we aim here is to have a simple server having two different separate functionalities. On the one side, we

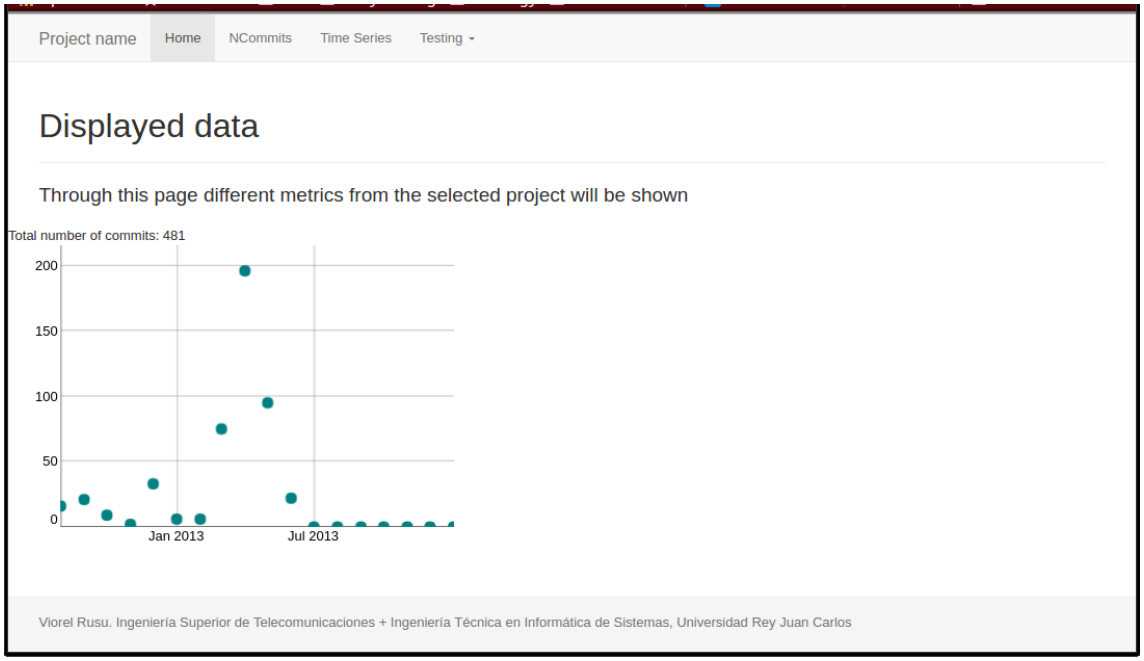

Figure 3.4: Simple browser visualization of commits timeseries

want this server to interact with elasticSearch on its own, without Kibana. We take advantage of this server and take into action the angularJS framework. On the other hand, we want it to show threeDC charts, loading all the libraries needed. Finally, a client through a browser will view a single page with two divs, one containing some sort of elasticSearch data and the other plotting some sort of threeDC charts. This will set the basis for a much more complex server like Kibana, with much more complex queries to elasticSearch and more issues with the threeeDC library.

First of all, we need a server. We decided to use the python built-in HTTP server for its simplicity. Just by running the server in a certain folder, it automatically serves the files in that folder through default port 8000. The command used for this couldn't be simpler:

```
1 python -m SimpleHTTPServer
```

So whatever is put in an index.html file, the server serves it. We start creating and modifying this file, which you can completely see at the appendix 5.5 of this paper. You can see the final result of this Sprint in figure 3.5.

Now we will focus on the two functionalities, on separate.

In the first place, let's focus on elasticSearch interaction. We need a way to interact with

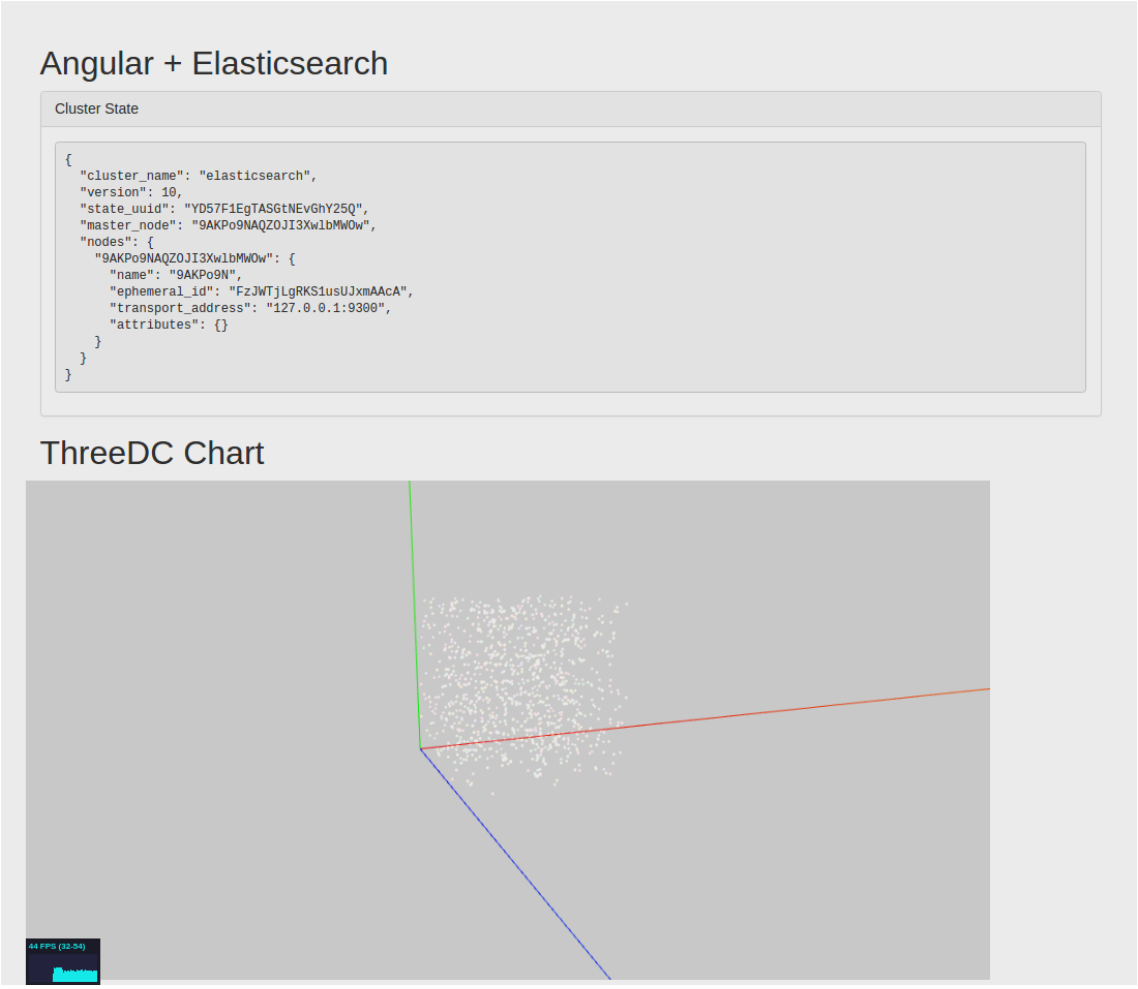

Figure 3.5: Html page showing the result of an ES query and a threeDC chart

elasticSearch. Elastic team provides different APIs for this, one for each of the main programming languages. As it's expected, there is one for Javascript. We will use the Angular Build API. So we do the following in our case.

Inside the angular module app, we create a service named client which we can use directly to make queries from the angular controller.

```
1  ExampleApp.service('client', function (esFactory) {
2    return esFactory({
3      host: 'localhost:9200',
4      apiVersion: '2.3',
5      log: 'trace'
6    });
7  });
```

Now, in the controller, we call a method of this API to retrieve some information from elasticSearch. At this point, it is not important what kind of information we retrieve. For example, we can call the client.state method[4], which is used to get comprehensive details about the state of the whole cluster.

```
1    client.cluster.state({
2      metric: [
3        'cluster_name',
4        'nodes',
5        'master_node',
6        'version'
7      ]
8      })
9    .then(function (resp) {
10     $scope.clusterState = resp;
11     }
```

Then, the result is attached to the scope so it is easy to print this result directly from the

[4] https://www.elastic.co/guide/en/elasticsearch/client/javascript-api/current/api-reference.html#api-cluster-state

html through angular. The result can be seen in the first section of the previous figure 3.5. The complete code in detail can also be found in Appendix A.2.

In the second place, we will focus on building our first real threeDC graph. In the next paragraphs, only main scene scheleton will be described. Details and secondary commentaries will be left for the reader to investigate in Appendix A.2.

First, we create all the three.js objects needed for a scene to correctly be visualized: camera, renderers and light:

```
1     // set up camera
2   camera = new THREE.PerspectiveCamera( VIEW_ANGLE, ASPECT, NEAR, FAR);
3   // add the camera to the scene
4   scene.add(camera);
5   // the camera defaults to position (0,0,0)
6   //    so pull it back (z = 400) and up (y = 100) and set the angle
    towards the scene origin
7   camera.position.set(0,150,400);
8   camera.lookAt(scene.position);
9
10  renderer = new THREE.WebGLRenderer( {antialias:true} );
11  renderer.setSize(SCREEN_WIDTH, SCREEN_HEIGHT);
12  renderer.setClearColor( 0xd8d8d8 );
13
14  var light = new THREE.PointLight(0xffffff,0.8);
15  light.position.set(0,200,250);
16  scene.add(light);
```

We get into threeDC library now. We pass all this 'three' elements to the threeDC library, to initialize a threeDC scene. Next, we create for example a 'pointsCloudChart' object, which paints an entire cloud made of points. In this case, we just use random data to paint that object.

```
1  THREEDC.initializer(camera,scene,renderer,container);
2
3  var cloud= THREEDC.pointsCloudChart([0,0,0]);
4  cloud.getPoints(getRandomPoints(1000));
5
6  THREEDC.renderAll();
```

The animate loop can't be forgotten, it is where all the updates and renders itself to the infinite. This function is called only once from the main code, but it will execute itself over and over again.

```
function animate()
{
    requestAnimationFrame( animate );
    renderer.render( scene, camera );
    THREEDC.controls.update();
}
```

With these basic structure (threeDC, elasticSearch and angularJS) we can start getting at work with Kibana, which undoubtely is the titan of this paper.

3.3.2 Simple elasticSearch query and visualization in Kibana custom plugin

In this section a new app will be created inside Kibana and elasticSearch will be queried from this app. We will make this query the simplest we can, for example querying the total number of documents in our index.

This will be the basic structure of our application:

```
export default function (kibana) {
  return new kibana.Plugin({
    require: ['elasticsearch'],

    uiExports: {

      app: {
        title: 'My new app plugin',
        description: 'my first elasticsearch requests in background',
        main: 'plugins/elasticsearch_status_vio/app'
      }
```

```
    }
});
```

We must load the elasticsearch module here, by specifying it in the require array. This module will be necessary to make elasticsearch queries, as we will do later in this sprint. The rest of the parameters speak on their on: title, description and main file. The way we specify that this is not a visualization type to Kibana is by specifying this inside the uiExports with the 'app' key instead of 'visTypes' as done in the previous sprint for example.

Next, we need to query elasticSearch in some way. We don't want to use the javascript API, this would totally ignore Kibana and is not a clean solution. We want to use it through Kibana, and make Kibana do the queries to elasticsearch, through the elasticsearch module. This is the clean solution. In order to do this, we create a new Kibana Server API in the following way:

```
import api from './server/routes/elasticsearch_routes';

    init(server, options) {
      // Add server routes and initalize the plugin here
      api(server);
    }
```

The init method adds a new server API to Kibana. In the api file, we specify the api calls we want to define. In our case, we define the /ncommits GET server API call in the following way:

```
export default function (server) {
  let call = server.plugins.elasticsearch.callWithRequest;
  server.route({
    path: '/api/elasticsearch_status/ncommits',
    method: 'GET',
    handler(req, reply) {
      call(req, 'count',{index: 'commits-index'}).then(function (response)
    {
        // Return just the names of all indices to the client.
        reply(response.count);
      });
    }
```

```
12    });
13  }
```

The elasticsearch callWithRequest utility must be used in order to access elasticSearch. This method receives as parameters the request from our API (req) and then the name of the function from the elasticSearch Javascript Client to be called. In our case, we use the 'count' method to count the total documents in the commits-index. This method returns a promise that will be resolved with the response from Elasticsearch, and we only have to access the 'count' parameter from the response in order to get the answer returned from elasticSearch to our query.

For routing between pages (in this case it will be only one page), we have to explicitly enable uiRoutes and define our route in our app.js file:

```
1  uiRoutes.enable();
2  .when('/ncommits', {
3    template: ncommitsTemplate,
4    controller:'elasticsearchNCommitsController',
5    controllerAs: 'ctrl'
6  });
7
8  uiModules
9  .get('app/elasticsearch_status_vio', [])
10 .controller('elasticsearchNCommitsController', function ($http) {
11   $http.get('../api/elasticsearch_status/ncommits').then((response) => {
12     this.ncommits = response.data;
13   });
14 });
```

For a detailed explanation on routes please consult the excellent Tim Roes tutorial about writing custom applications(1), in which this sprint was based.

Finally, we define the ncommitsTemplate mentioned as follows:

```
1  <div class="container">
2    <div class="row">
3      <div class="col-12-sm">
```

Figure 3.6: Custom app plugin result

```
4       <a href="#/">Index list</a>
5       <h1>Number of commits in commits_index: {{ ctrl.ncommits }}</h1>
6     </div>
7   </div>
8 </div>
```

The final result can be seen in Fig 3.6.

3.4 Sprint 2: 3D Basic Pie Visualization

3.4.1 Hello World visualization plugin

In this section, a very basic plugin in Kibana will be built from scratch. The final result can be seen in figure XXX

In Kibana, developed plugins go into Kibana/plugins/ directory. The first thing to know about Kibana plugins is that every plugin is actually a npm module. Like every npm module, two basic files need to be created:

- package.json. We specify our plugin name here.

```
1 {
2   "name": "sprint2_plugin",
3   "version": "5.0.0"
4 }
```

- index.js. Here, our module has to instantiate a new instance of Kibana plugin, creating a new plugin of type 'visualization' (there are other type of plugins like completely

separated apps) and registering it in the following way:

```
export default function (kibana) {

  return new kibana.Plugin({
    uiExports: {
      visTypes: [
        'plugins/sprint2_plugin/sprint2'
          ]
      }
    });
};
```

In order to define a basic functionality showing up ”Hello World” inside the visualization plugin, we have to define the following two files also in directory public:

- sprint2.js. In this file, we define our visualization setting parameters like a title, a description or an icon to appear in visualization list. We also define here the controller for our div in Kibana, which we will describe in the next paragraph. Full code can be seen in appendix xxx(sprint2.js)

- sprint2.html. In this file, a basic div is defined. It is defined as a controller, so a basic angularJS structure is used to attach a parameter named ’name’ and show it in Kibana through the template.

In figure 3.7 our first visualization plugin can be seen in the list, together with default Kibana visualization plugins. Figure 3.8 shows the final plugin result. With this basic plugin, we have the skeleton to build everything we want.

Once plugin creation and interaction with elasticsearch from Kibana is clear, the next logical step is to incorporate our visual libraries in a plugin in Kibana. In this Sprint a basic threeDC scene will be created and correctly integrated in a visualization plugin in Kibana. No elasticSearch data will be used yet.

In order to progressively move toward the objective, a basic three.js scene will be integrated before integrating a more complex threeDC.js scene.

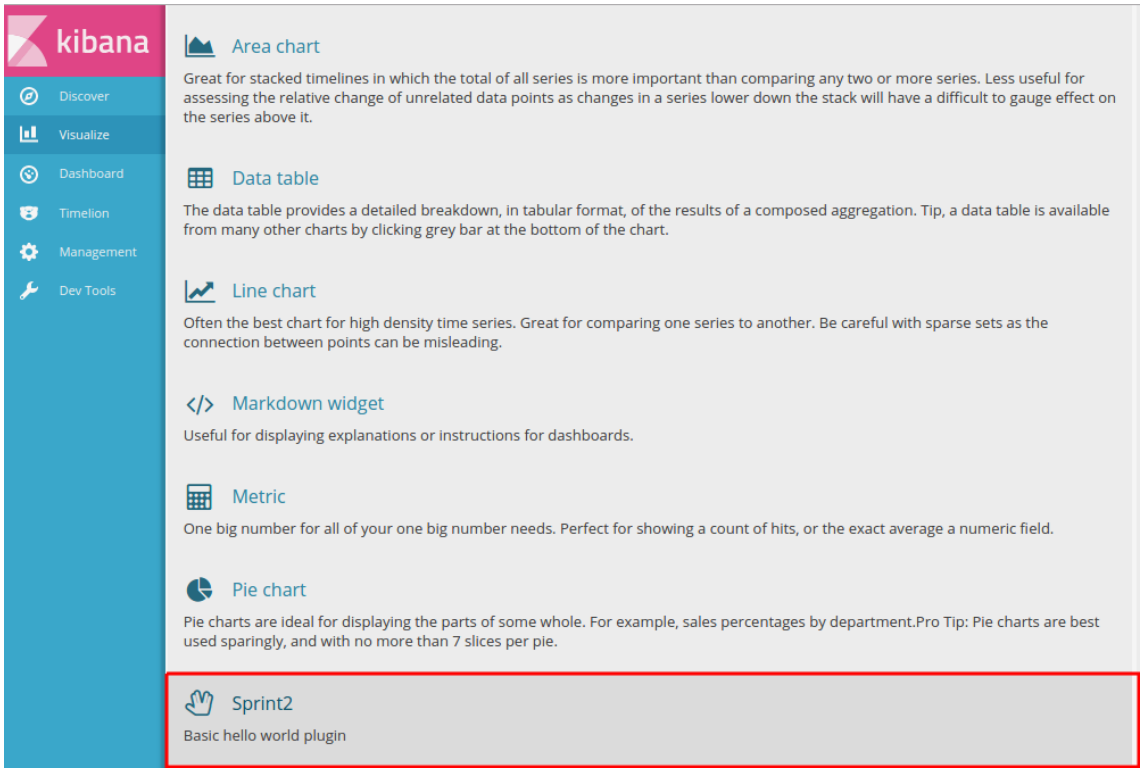

Figure 3.7: Hello World plugin in visualizations plugin list

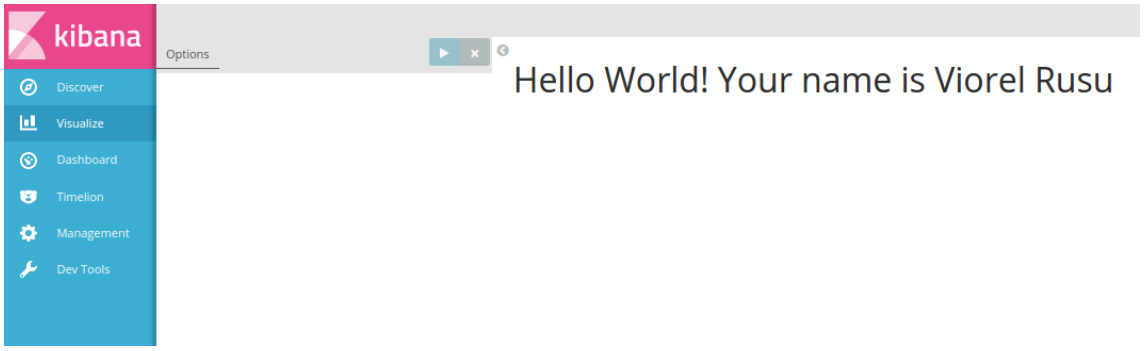

Figure 3.8: Hello World plugin result

3.4.2 Three.js scene integration

In order to insert a basic three.js scene in Kibana, the following steps were followed:

1. Add three.js library through node package manager. To add three.js library is easy because it is already packed online (https://www.npmjs.com/package/three) and ready to install via npm simply with the 'npm install three' command.

2. From the main plugin file, the new library can easily be imported:

```
1 THREE = require("three");
```

   Then, it can be used across this file wherever we consider

3. A controller has been created manipulating a basic three.js rotating cube:

```
1  module.controller('3DCubeController', function($scope, $element){
2
3      init();
4      animate();
5
6      function init() {
7          camera = new THREE.PerspectiveCamera(75, window.innerWidth /
       window.innerHeight, 1, 10000);
8          camera.position.z = 1000;
9          scene = new THREE.Scene();
10         geometry = new THREE.BoxGeometry(200, 200, 200);
11         material = new THREE.MeshBasicMaterial({
12             color: 0xff0000,
13             wireframe: true});
14         mesh = new THREE.Mesh(geometry, material);
15         renderer = new THREE.WebGLRenderer();
16         renderer.setSize(window.innerWidth, window.innerHeight);
17         var idchart = $element.children().find(".3Dcubechart");
18       container = idchart[0];
19         container.appendChild(renderer.domElement);}
20
21     function animate() {
```

```
22          requestAnimationFrame(animate);
23          mesh.rotation.x += 0.01;
24          mesh.rotation.y += 0.02;
25          renderer.render(scene, camera);}});
```

The key in correctly inserting the scene in Kibana is in using the $element service[5], as it can be seen in the controller declaration in the code above. This angularJS service allows us to access jQuery functions from the controller. So jQuery functions children() and find() are used in order to find the correct div to insert in. Once found, all the scene is inserted here by using the appendChild HTML DOM manipulation function and the DOM element containing the scene returned from the renderer itself.

4. And in the main html file a canvas is defined to be controlled by the controller, and to contain a div to be found by jQuery service $element.

```
1 <canvas id="glcanvas" width="640" height="480" ng-controller="3
     DCubeController">
2       <div class="3Dcubecontainer">
3      <div class="3Dcubechart"> </div>
4      </div>
5 </canvas>
```

The final result can be seen in Figure 3.9

3.4.3 ThreeDC.js scene integration

———————— Modulos en Javascript Explicar sobre los módulos. vr vis es un modulo AMD. Necesita estar wrapped en define(function(require) ...) ver https://www.timroes.de/2015/12/02/v kibana-4-plugins-simple-visualizations/ ————————

This section may seem obvious, but it's very tricky. ThreeDC.js library, in order to work properly, needs a set of external libraries itself. These modules are extensions that allow different functionality to the threeDC scene, like adding mouse interactivity or providing threejs

[5] https://docs.angularjs.org/api/ng/function/angular.element

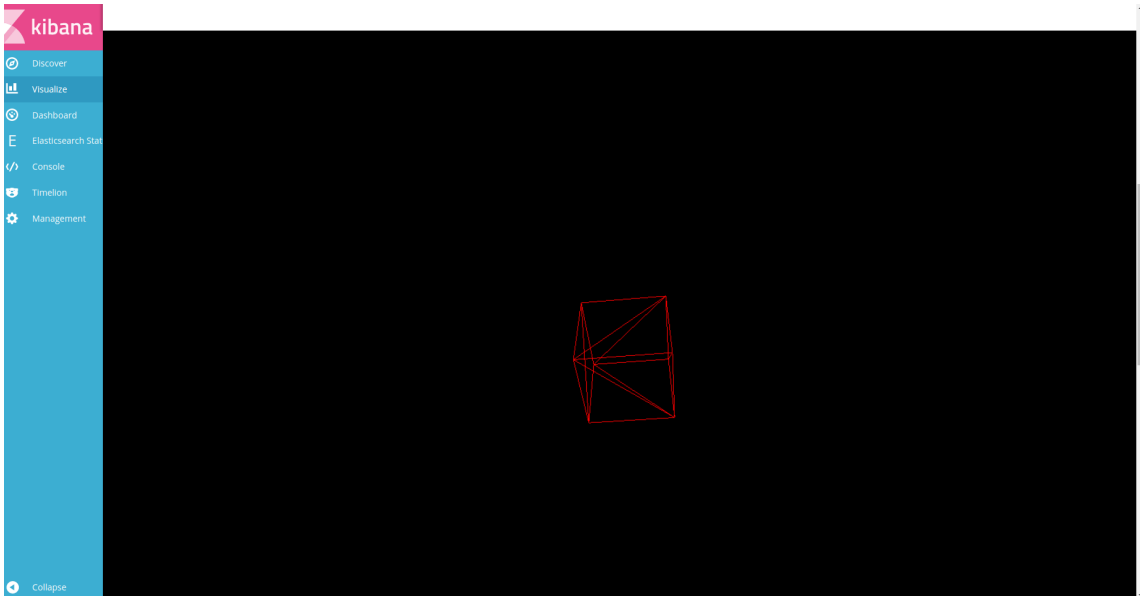

Figure 3.9: Three.js scene correctly inserted in Kibana

fonts or text operations. We won't get into detail at this point, as threeDC already uses all these modules and it's not something new to this project.

Everythig would be easy if we could just include these files in the html. There would be a global variables scope where all of them would be attached and would be working properly. This is what threeDC.js library does. But in Kibana this can not be done, because everything is a module and all the variables have a scope. So, in order to solve this, we have to examine how exactly each one of these modules work and what they do.

The majority of these auxiliary modules simply extend the THREE object we already have from the library, by adding another property to this object. In these cases, by simply doing a require() of that file, the THREE object is correctly extended.

```
1  require("plugins/3D_kibana_charts_vis/FontUtils");
2  require("plugins/3D_kibana_charts_vis/TextGeometry");
3  require("plugins/3D_kibana_charts_vis/Projector");
4  require("plugins/3D_kibana_charts_vis/OrbitControls");
```

Another case is that in the auxiliary modules a new object is created. In these cases, a simple require() is not enough. We have to export the object from inside the file with module.exports, so that it can correctly be imported from our main file. This is the case even with the threeDC

library itself.

```
1 THREEDC = require("plugins/3D_kibana_charts_vis/3dc");
2 THREEx = require("plugins/3D_kibana_charts_vis/threex.domevents");
3 Detector = require("plugins/3D_kibana_charts_vis/Detector");
```

Also, helvetiker bold font has to be imported and loaded as follows. It is necessary for threeDC to work properly and correctly display text in the 3D scene.

```
1 var typeface2 = require('plugins/3D_kibana_charts_vis/helvetiker_bold.
    typeface');
2 THREE.typeface_js.loadFace(typeface2);
```

Once imported all these modules, any threeDC scene can be created and displayed. In our case, we want to build the first 3D Pie chart scene integrated in Kibana. Once the scene is initialized, it is easy to create a pie, following the scheme described in the previous section 3.4.2. We only need to create a pieChart threeDC Object inside our controller, add some sample data to it and display it:

```
1 pie = THREEDC.pieChart();
2 pie.data(sampleData);
```

The visual result is very similar to what can be seen in Figure 4.4.

3.5 Sprint 3: Working with data

So far we have a very basic 3D pie visualization. From this basic scheleton we can built all the advanced functionality we want the pie to have, and also build the other visualizations (bars chart and bubbles chart) using this same structure.

In this Sprint, we focus on data treatment. Sample data will no longer be used, but elasticSearch queries will enter into action instead and get that data. This data is properly transformed in javascript in order to follow the pie chart input format, and then plot or replot the pie.

This data treatment is done in similar ways in the three visualizations developed in this plugin: pie chart, bars chart and bubbles chart, with minor differences. The duplicated functionality won't be repeated here, but a generalization on how to retrieve and treat the data for the pie chart will be given, which will serve for all the visualization charts.

3.5.1 Defining the schemas

After creating our first pie visualization with its visualization provider, a very important step is to define our schemas. Before writing any code, we should already know what kind of bucket and metrics aggregations it is needed for each chart type. Conceptually, the next schemas design has been made for the pie chart: it is needed exactly one bucket aggregation to define the slices and exactly one metric aggregation to define the slice size. With this in mind we define the schemas in the following way in our pie3D.js file:

```
1  schemas: new Schemas([
2          {
3            group: 'metrics',
4            name: 'slice_size',
5            title: 'Slice Size',
6            min: 1,
7            max: 1,
8            aggFilter: ['count', 'avg', 'sum', 'min', 'max', 'cardinality', '
       std_dev']},
9          {
10           group: 'buckets',
11           name: 'slices',
12           title: 'Slices',
13           min: 1,
14           max: 1,
15           aggFilter: '!geohash_grid'}])
```

Each Schemas object takes an array of objects in its constructor. Each object describes one aggregation that will be accepted for the pie visualization. Each aggregation object have the following keys:

- group - either ”metrics” or ”buckets”. It defines which kind of aggregation we want to describe in this object.
- name - the id of this aggregation. This will be used later by our controller to reference this aggregation.
- title - the title shown to the user, when he adds the aggregation. Should describe how that aggregation will be visualized (e.g. in that case the bucket aggregation will create tags, the metrics aggregation will influence the tag size)
- min/max - the number of minimum and maximum aggregations of that type, a user can add. In our case we only allow 1 aggregation of each type, due to the way our visualization works.
- aggFilter - a filter on which aggregations should be allowed. It is an array of either aggregation types (see below), that are allowed in this place (as shown in our metrics aggregation) or an array of aggregation types forbidden (each must be prefixed with a bang). In the later case all other aggregations are allowed. If the array has only one element that can also be specified as a string (as shown in the bucket aggregation).

This schemas defined for the pie chart can be seen in actions in the pie aggregations menu, Figure 4.3.

For the bars and bubbles chart similar schemas have been defined. The bars chart defines exactly two buckets, one for each axis, and one metric, for each bar height. The bubbles chart also define these two buckets for each dimension and a metric for the bubble height, but it defines an extra metric for the bubble’s size. That makes two buckets and two metrics in total.

3.5.2 Data retrieval

Once our schemas are correctly defined, we can access data from the controller by accesing two variables inherited into our angular scope.One is the vis variable, which holds information about your visualization and the settings the user chose. The other variable is named esResponse and holds the Elasticsearch response for your visualization. Kibana will automatically query Elasticsearch with the aggregations set by the user and taking into account currently set queries and filters.

To visualize our data we need to match the response data with the user configuration for our widget. To access the result of the aggregations we can look into $scope.esResponse.aggregations. To find aggregations in that object we need their ids. To find the ids for a specific aggregation we can use several methods of $scope.vis.aggs to find the id.

```
1       // Retrieve the id of the configured tags aggregation
2       var slicesAggId = $scope.vis.aggs.bySchemaName['slices'][0].id;
3       // Retrieve the metrics aggregation configured
4       var metricsAgg = $scope.vis.aggs.bySchemaName['slice_size'][0];
5       // Get the buckets of that aggregation
6       var buckets = resp.aggregations[slicesAggId].buckets;
```

The buckets variable is an array of buckets, and accesing the key field of each bucket we get the bucket name which we can use to display each slice name. In order to get the value of that bucket, we access it in the following way:

```
1 var value = metricsAgg.getValue(bucket);
```

This value will be used to determine each slice's size in the pie chart.

Similar reasoning can be made for the bars and the bubbles chart, with the following particularities:

- In order to get the second buckets aggregation id, we retrieve the second element of the $scope.vis.aggs.bySchemaName['...'], like this:

```
1 var barsyAggId = $scope.vis.aggs.bySchemaName['bars'][1].id;
```

  A similar reasoning goes for retrieving the second metric aggregation. Then, we just have to apply the getValue method on each metrics aggregation.

- In order to get the buckets in the 'y' dimension, we access each bucket in the 'x' dimension and then retrieve its buckets, like this:

```
1 var bucketsy = bucketx[barsyAggId].buckets;
```

3.5.3 Data treatment for threeDC

So far we have managed to make kibana query elasticsearch for all the data we need to plot the charts. But this is not all the work it is need to be done. Some tedious work is still to be performed. This work consists in taking the data from the Kibana buckets and metrics aggregations and adapt it to the threeDC format.

TODO: explicar el formato de entrada de threeDC, explicar el proceso muy por encima

3.6 Sprint 4: Kibana integration

3.6.1 Field filters

3.6.2 Date filters

3.6.3 Dashboard

Chapter 4

Design and results

4.1 Introduction

4.2 Architecture

4.3 User Guide

In this section every functionality of the plugin will be explained. A detailed guide on how to produce each type of visualization will be given, as welll as every possibility every visualization type has.

After correctly installing the plugin, three new visualizations type should be available in the Kibana Visualizations menu, as shown in Figure 4.2

Next, each visualization and its functionalities will be explained separately.

4.3.1 3D Pie Chart

After selecting ”3D Pie Chart” in the menu and selecting the index pattern to work with (Kibana asks for it), we come across with the 3D pie aggregations menu, as seen in Figure 4.3 borded with a red rectangle.

The 3D pie visualization needs exactly one buckets aggregations and one metrics aggregation. By clicking on ’Slices’ we are choosing what each slice of our pie will represent, the buckets in which our data will fall into. We could choose, for example, a Date Histogram, and

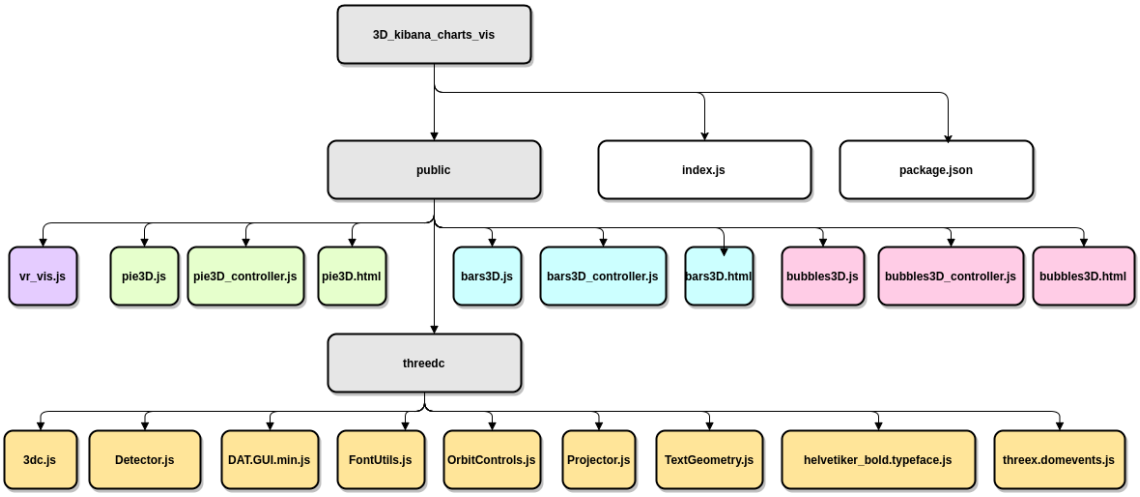

Figure 4.1: The 3 new plugin visualizations in Kibana menu

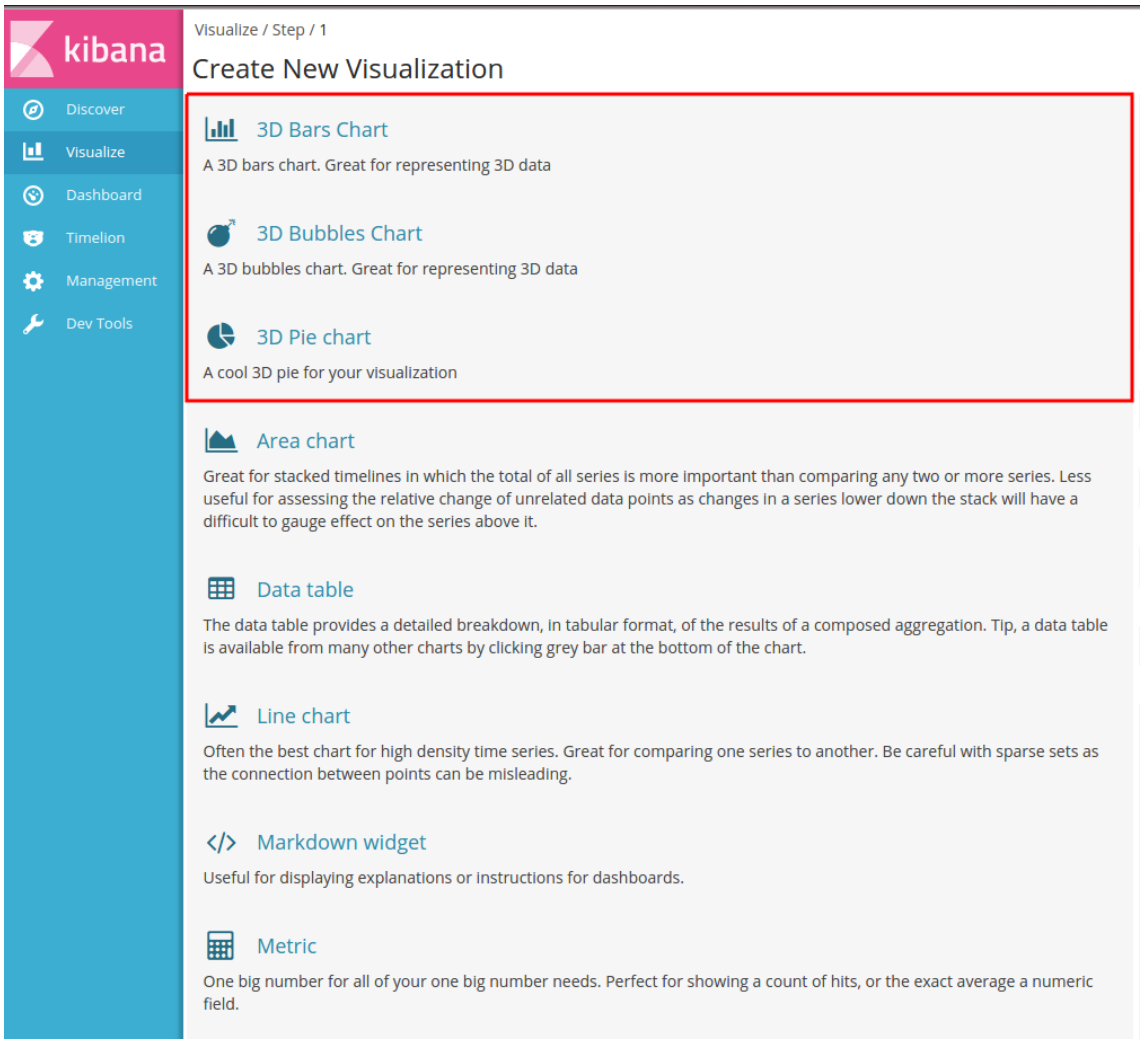

Figure 4.2: The 3 new plugin visualizations in Kibana menu

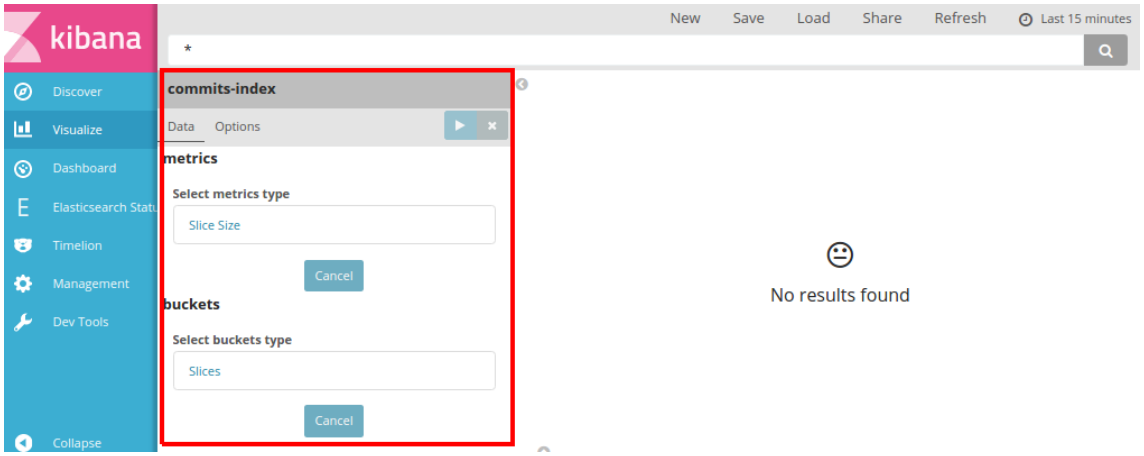

Figure 4.3: 3D Pie visualization aggregations menu

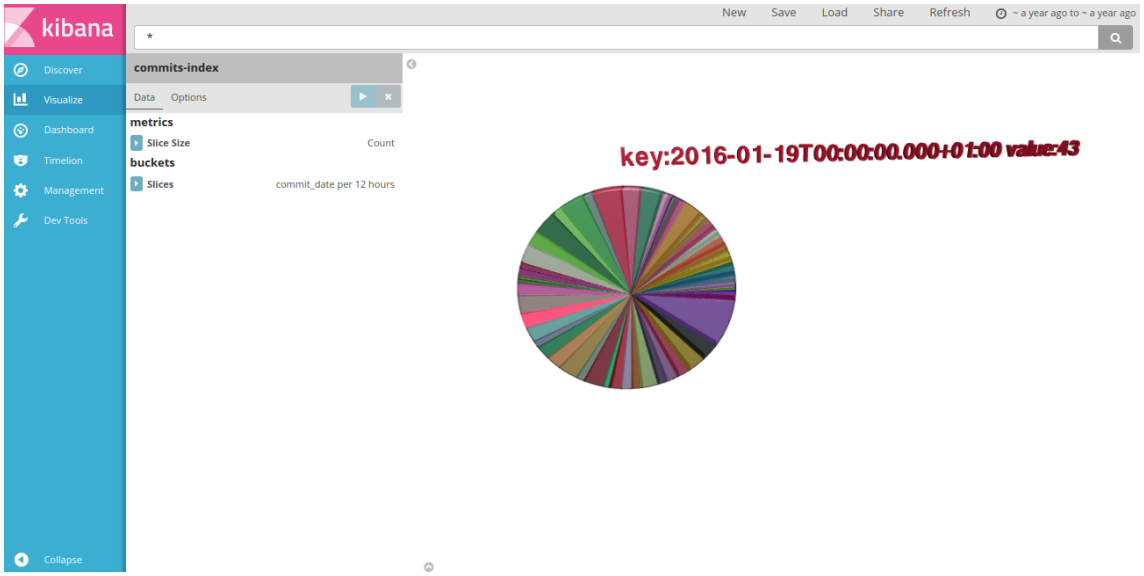

Figure 4.4: The 3 new plugin visualizations in Kibana menu

then each slice will represent a date interval. By clicking on 'Slice Size' we are choosing what the size of each slice means, the metrics of the data. For example, we could choose the 'Count' aggregation, in which case the size of each slice will be determined by the number of documents that fall into each bucket.

The result of such an example can be seen in Figure 4.4. When the mouse moves over a slice, info about that slice is displayed:

- **key**: shows the bucket which that slice represents
- **value**: shows the metric we configured in aggregations menu

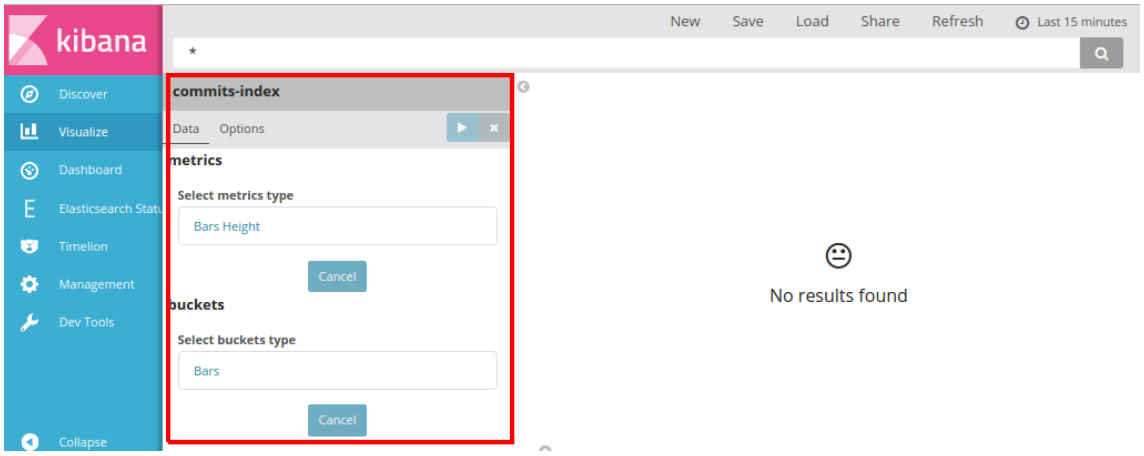

Figure 4.5: 3D Bars visualization aggregations menu

4.3.2 3D Bars Chart

After selecting ”3D Bars Chart” in the menu and selecting the index pattern to work with, we come across with the 3D bars aggregations menu, as seen in Figure 4.5 borded with a red rectangle.

The 3D bars visualization needs exactly two buckets aggregations and one metric aggregation. The first buckets aggregation is for the values in X axis and the second buckets aggregation is for the Y axis. For example, we could choose to group our date in X axis by a Date Histogram, and then in Y axis by some Terms. We define these by clicking on the ’Bars’ button and remembering to click the ’Add sub-buckets’ button to define the Y axis. We define the height of each bar by defining the metric clicking on ’Bars Height’. For example, we could choose the ’Count’ aggregation to represent the number of documents found in that bucket by the height of the 3D Bar.

The result of such an example can be seen in Figure 4.6. When the mouse moves over a bar, info about that bar is displayed:

- **key1**: shows the first bucket id previously defined in aggregations menu
- **key2**: shows the second bucket id previously defined in aggregations menu
- **value**: shows the metric we configured in aggregations menu

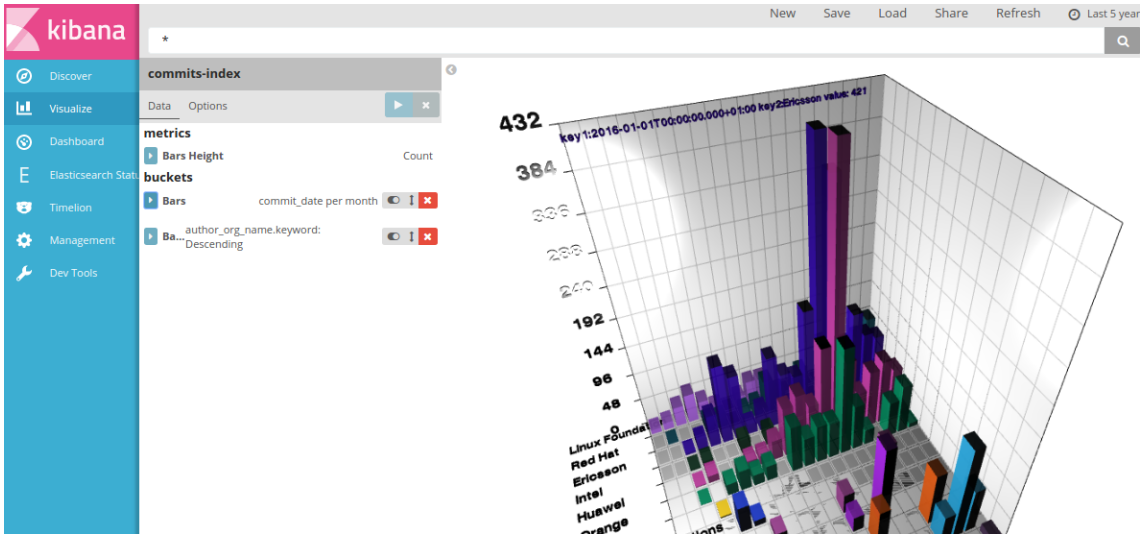

Figure 4.6: The 3 new plugin visualizations in Kibana menu

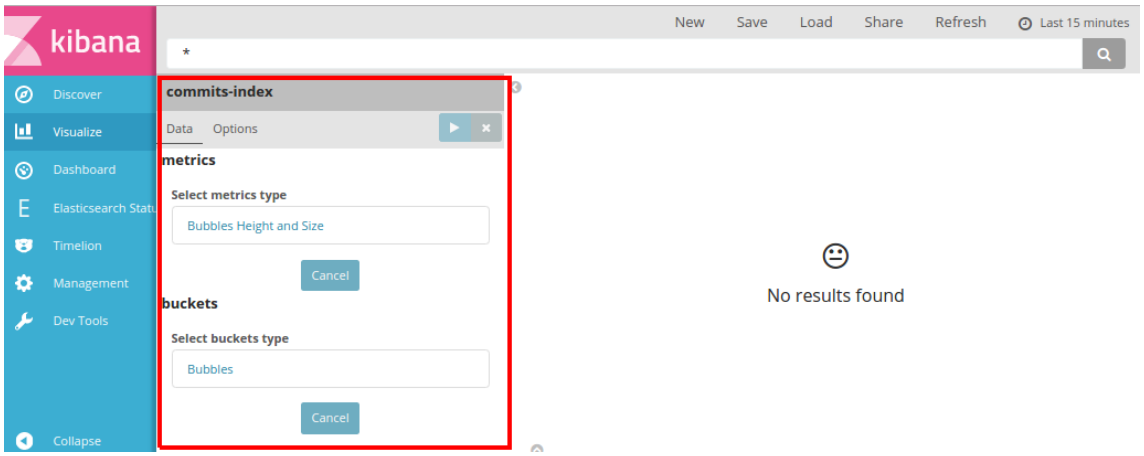

Figure 4.7: 3D Bars visualization aggregations menu

4.3.3 3D Bubbles Chart

After selecting ”3D Bubbles Chart” in the menu and selecting the index pattern to work with, we come across with the 3D bubbles aggregations menu, as seen in Figure 4.7 borded with a red rectangle.

The 3D bubbles visualization needs exactly two buckets aggregations and two metric aggregation. The first buckets aggregation is for the values in X axis and the second buckets aggregation is for the Y axis. For example, we could choose to group our date in X axis by a Date Histogram, and then in Y axis by some Terms. We define these by clicking on the ’Bubbles’ button and remembering to click the ’Add sub-buckets’ button to define the Y axis.

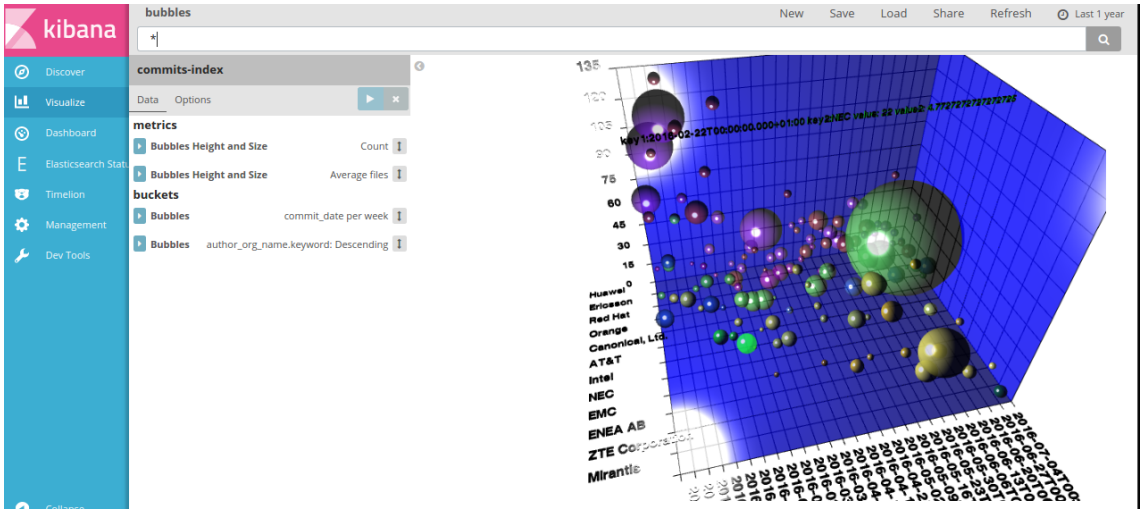

Figure 4.8: The 3 new plugin visualizations in Kibana menu

We define each bubble's height and size clicking the metric 'Bubbles Height and Size'. For example, we could choose the 'Average' metric on some field in first place to establish each bubble's height and then the 'Count' metric to establish each bubble's size.

The result of such an example can be seen in Figure 4.8. When the mouse moves over a bubble, info about that bubble is displayed:

- **key1**: shows the first bucket id previously defined in aggregations menu, represented in X axis
- **key2**: shows the second bucket id previously defined in aggregations menu, represented in Y axis
- **value**: shows the first metric we configured in aggregations menu, represented bubble's height
- **value2**: shows the metric we configured in aggregations menu, represented in bubble's size

4.3.4 Interacting with the 3D charts

In this section we will explain what charts can do once built and how the user can interact with the 3D scene.

In first place, scene can be zoomed in or zoomed out using the mouse wheel. It can also be dragged used the secondary mouse button and rotated using the principal mouse button and moving the mouse.

In second place, in each of the charts, when mousing over a slice, bar or bubble, the object it's visually highlighted. Additionally, when clicking on a particular object, the corresponding filters automatically apply filtering for the buckets defined. In this way, we have only one filter applied when clicking on a slice in the pie chart, and two filters applied when clicking on a bar in the bars chart or a bubble in the bubbles chart. The filter applied automatically know if they should apply a time filter, as if the user manually defined it in the right top corner, or just apply a normal field filter.

Chapter 5

Conclusiones

5.1 Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

5.2 Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TFG/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

1. a
2. b

5.3 Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster.

1. a

2. b

5.4 Trabajos futuros

Ningún software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

5.5 Valoración personal

Finalmente (y de manera opcional), hay gente que se anima a dar su punto de vista sobre el proyecto, lo que ha aprendido, lo que le gustaría haber aprendido, las tecnologías utilizadas y demás.

Appendix A

Appendix

A.1 index.html from Sprint1

Complete index.html file from Sprint 1

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Threeboard</title>
5      <meta charset="utf-8">
6      <meta name="viewport" content="width=device-width, user-scalable=no,
       minimum-scale=1.0, maximum-scale=1.0">
7      <link rel="stylesheet" type="text/css" href="node_modules/bootstrap/
       dist/css/bootstrap.css">
8    </head>
9    <body ng-app="ExampleApp">
10     <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery
       .min.js"></script>
11     <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r78/three.
       min.js"></script>
12     <script src="js/FontUtils.js"></script>
13     <script src="js/TextGeometry.js"></script>
14     <script src="js/Projector.js"></script>
15     <script src='js/threex.domevents.js'></script>
16     <script src="js/Detector.js"></script>
17     <script src="js/Stats.js"></script>
```

```
18      <script src="js/OrbitControls.js"></script>
19      <script src="js/THREEx.WindowResize.js"></script>
20      <script src="js/THREEx.FullScreen.js"></script>
21      <script type="text/javascript" src='js/DAT.GUI.min.js'></script>
22      <script src="https://cdnjs.cloudflare.com/ajax/libs/crossfilter/1.3.12/
        crossfilter.min.js"></script>
23      <script src="js/3dc.js"></script>
24      <div id="ThreeJS" style="position: absolute; left:0px; top:0px">
25      <script src="fonts/gentilis_bold.typeface.js"></script>
26      <script src="fonts/gentilis_regular.typeface.js"></script>
27      <script src="fonts/optimer_bold.typeface.js"></script>
28      <script src="fonts/optimer_regular.typeface.js"></script>
29      <script src="fonts/helvetiker_bold.typeface.js"></script>
30      <script src="fonts/helvetiker_regular.typeface.js"></script>
31      <script src="fonts/droid_sans_regular.typeface.js"></script>
32      <script src="fonts/droid_sans_bold.typeface.js"></script>
33      <script src="fonts/droid_serif_regular.typeface.js"></script>
34      <script src="fonts/droid_serif_bold.typeface.js"></script>
35
36    <!-- include npm modules in proper order -->
37    <script src="node_modules/angular/angular.min.js"></script>
38    <script src="node_modules/elasticsearch-browser/elasticsearch.angular.min
        .js"></script>
39      <script src="script.js"></script>
40
41    <!-- attach the ExampleController to our main content -->
42    <div ng-controller="ExampleController" class="container">
43      <h1>Angular + Elasticsearch</h1>
44
45      <!-- if there is an error, display its message -->
46      <div ng-if="error" class="alert alert-danger" role="alert">{{error.
        message}}</div>
47
48      <!-- if clusterState is available, display it as formatted json -->
49      <div ng-if="clusterState" class="panel panel-default">
50        <div class="panel-heading">
51          <h3 class="panel-title">Cluster State</h3>
```

```
52        </div>
53        <div class="panel-body">
54          <pre>{{clusterState | json}}</pre>
55        </div>
56      </div>
57    </div>
58
59    </body>
60 </html>
61    </body>
62 </html>
```

A.2 script.js from Sprint1

```
1
2  // standard global variables
3  var container, scene, camera, renderer, stats;
4
5  //JSON data saved here
6  var json_data;
7
8
9   elasticstuff();
10
11 // initialization
12   //getJSON call, draw meshes with data
13    $.getJSON("jsons/scm-commits.json", function(data) {
14        json_data=data;
15        init();
16        // animation loop / game loop
17        animate();
18    });
19
20 ///////////////
21 // FUNCTIONS //
```

```
22 ///////////////
23
24 function elasticstuff() {
25   //elasticsearch and angular
26     // App module
27     //
28     // The app module will contain all of the components the app needs (
       directives,
29     // controllers, services, etc.). Since it will be using the components
       within
30     // the elasticsearch module, define it a dependency.
31     var ExampleApp = angular.module('ExampleApp', ['elasticsearch']);
32
33     // Service
34     //
35     // esFactory() creates a configured client instance. Turn that instance
36     // into a service so that it can be required by other parts of the
       application
37     ExampleApp.service('client', function (esFactory) {
38       return esFactory({
39         host: 'localhost:9200',
40         apiVersion: '2.3',
41         log: 'trace'
42       });
43     });
44
45     // Controller
46     //
47     // It requires the "client" service, and fetches information about the
       server,
48     // it adds either an error or info about the server to $scope.
49     //
50     // It also requires the esFactory to that it can check for a specific
       type of
51     // error which might come back from the client
52     ExampleApp.controller('ExampleController', function ($scope, client,
       esFactory) {
```

```
53
54        client.cluster.state({
55          metric: [
56            'cluster_name',
57            'nodes',
58            'master_node',
59            'version'
60          ]
61          })
62        .then(function (resp) {
63          $scope.clusterState = resp;
64          $scope.error = null;
65        })
66        .catch(function (err) {
67          $scope.clusterState = null;
68          $scope.error = err;
69
70          // if the err is a NoConnections error, then the client was not
      able to
71          // connect to elasticsearch. In that case, create a more detailed
      error
72          // message
73          if (err instanceof esFactory.errors.NoConnections) {
74            $scope.error = new Error('Unable to connect to elasticsearch. ' +
75              'Make sure that it is running and listening at http://localhost
      :9200');
76          }
77        });
78
79      });
80  }
81
82  function init () {
83
84    ////////////
85    // SCENE //
86    ////////////
```

```
87    scene = new THREE.Scene();
88
89    ////////////
90    // CAMERA //
91    ////////////
92    // set the view size in pixels (custom or according to window size)
93    var SCREEN_WIDTH = window.innerWidth/2;
94    var SCREEN_HEIGHT = window.innerHeight/2;
95    // camera attributes
96    var VIEW_ANGLE = 45;
97    var ASPECT = SCREEN_WIDTH / SCREEN_HEIGHT;
98    var NEAR = 0.1;
99    var FAR = 20000;
100      // set up camera
101   camera = new THREE.PerspectiveCamera( VIEW_ANGLE, ASPECT, NEAR, FAR);
102   // add the camera to the scene
103   scene.add(camera);
104   // the camera defaults to position (0,0,0)
105   //    so pull it back (z = 400) and up (y = 100) and set the angle
      towards the scene origin
106   camera.position.set(0,150,400);
107   camera.lookAt(scene.position);
108
109   //////////////
110   // RENDERER //
111   //////////////
112   renderer = new THREE.WebGLRenderer( {antialias:true} );
113   renderer.setSize(SCREEN_WIDTH, SCREEN_HEIGHT);
114   renderer.setClearColor( 0xd8d8d8 );
115
116   // attach div element to variable to contain the renderer
117   container = document.getElementById( 'ThreeJS' );
118   // attach renderer to the container div
119   container.appendChild( renderer.domElement );
120
121    ////////////
122  // EVENTS //
```

```
123  ////////////
124
125
126
127  // automatically resize renderer
128  THREEx.WindowResize(renderer, camera);
129    // toggle full-screen on given key press
130  THREEx.FullScreen.bindKey({ charCode : 'm'.charCodeAt(0) });
131
132   ///////////
133   // LIGHT //
134   ///////////
135   var light = new THREE.PointLight(0xffffff,0.8);
136   light.position.set(0,200,250);
137   scene.add(light);
138   var ambientLight = new THREE.AmbientLight(0x111111);
139   // scene.add(ambientLight);
140
141   // create a set of coordinate axes to help orient user
142   //    specify length in pixels in each direction
143   var axes = new THREE.AxisHelper(1000);
144   scene.add(axes);
145
146  //STATS
147  stats = new Stats();
148  stats.domElement.style.position = 'absolute';
149  stats.domElement.style.bottom = '0px';
150  stats.domElement.style.zIndex = 100;
151  container.appendChild( stats.domElement );
152
153   //////////////
154   // CUSTOM //
155   //////////////
156
157   // most objects displayed are a "mesh":
158   //  a collection of points ("geometry") and
159   //  a set of surface parameters ("material")
```

```
160
161   var parsed_data=[];
162
163   // Crossfilter and dc.js format
164   json_data.values.forEach(function (value) {
165     var record = {}
166     json_data.names.forEach(function (name, index) {
167         if (name == "date") {
168           var date = new Date(value[index]*1000);
169           record[name] = date;
170           record.month = new Date(date.getFullYear(), date.getMonth(), 1);
171           record.hour = date.getUTCHours();
172         } else {
173           record[name] = value[index];
174         }
175     });
176     parsed_data.push(record);
177   });
178
179   //example data for cloud
180
181   function getRandomPoints(numberOfPoints){
182     var points=[];
183     for (var i = 0; i < numberOfPoints; i++) {
184
185       points[i]={x:Math.random()*100,y:Math.random()*100,z:Math.random()
      *100};
186       // console.log(points[i]);
187     };
188     return points;
189   }
190
191  //CUSTOM DASHBOARD//
192
193   THREEDC.initializer(camera,scene,renderer,container);
194
195   var cloud= THREEDC.pointsCloudChart([0,0,0]);
```

```
196   cloud.getPoints(getRandomPoints(1000));
197
198  // var bars= THREEDC.barsChart([0,0,0]);
199   //bars.group(groupByRepo);
200
201   THREEDC.renderAll();
202
203 }
204
205 function animate()
206 {
207    requestAnimationFrame( animate );
208    render();
209    update();
210 }
211
212 function render()
213 {
214    renderer.render( scene, camera );
215 }
216
217 function update()
218 {
219   THREEDC.controls.update();
220   stats.update();
```

[frame=single]

A.3 sprint2.js from Sprint2

```
1
2   // Create an Angular module for this plugin
3   var module = require('ui/modules').get('sprint2');
4   // Add a controller to this module
5   module.controller('HelloController', function($scope, $timeout) {
```

```
6
7       $scope.name="Viorel Rusu"
8
9     });
10
11
12  export default function Sprint2Provider(Private) {
13      var TemplateVisType = Private(require('ui/template_vis_type/
        template_vis_type'));
14      return new TemplateVisType({
15        name: 'Sprint2', // the internal id of the visualization
16        title: 'Sprint2', // the name shown in the visualize list
17        icon: 'fa-hand-spock-o', // the class of the font awesome icon for
        this
18        description: 'Basic hello world plugin', // description shown to the
        user
19        requiresSearch: false, // Cannot be linked to a search
20        template: require('plugins/sprint2_plugin/sprint2.html') // Load the
        template of the visualization
21      });
22    }
23
24    require('ui/registry/vis_types').register(Sprint2Provider);
```

[frame=single]

Appendix B

Bibliography

Scrum: http://scrummethodology.com/scrum-sprint/

Javascript: Marijn Haverbeke, *Eloquent JavaScript*. 2014 http://www.w3schools.com/js/

Node and npm: https://www.sitepoint.com/beginners-guide-node-package-manager/

Angular: http://campus.codeschool.com/courses/shaping-up-with-angular-js https://www.toptal.com/angul
js/a-step-by-step-guide-to-your-first-angularjs-app http://www.w3schools.com/angular/

Kibana plugins: https://www.timroes.de/2015/12/02/writing-kibana-4-plugins-basics/ https://www.timroes
kibana-4-plugins-simple-visualizations/ https://www.timroes.de/2015/12/06/writing-kibana-4-plugins-
visualizations-using-data/ https://www.timroes.de/2016/02/17/writing-kibana-4-plugins-field-formatters/
https://www.timroes.de/2016/02/21/writing-kibana-plugins-custom-applications/

webGl

https://en.wikipedia.org/wiki/WebGL https://www.chromeexperiments.com/experiment/webgl-
water-simulation

three

https://github.com/mrdoob/three.js http://helloracer.com/

kibana and elasticsearch: https://www.elastic.co

Bibliography