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

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. - Store open source projects data in the elasticsearch database engine.
2. - Use Kibana integrated tools to retrieve the data we need from elasticsearch.
3. - Build this new 3D visualizations: pie chart,
4. - Integrate threedc.js 3D scenes in Kibana visualization
5. - Integrate threedc.js visualization in Kibana dashboard
6. - Add custom events to threedc charts in order to filter data on click
7. - 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.
- . . .

Chapter 2

State of art and context

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

Descripción de las tecnologías que utilizas en tu trabajo. Con dos o tres párrafos por cada tecnología, vale.

Puedes citar libros, como el de Bonabeau et al. sobre procesos estigmérgicos [1].

También existe la posibilidad de poner notas al pie de página, por ejemplo, una para indicarte que visite la página de LibreSoft[1].

2.1 Sección 1

[1]http://www.libresoft.es

Chapter 3

Used technologies

3.1 HTML5

3.2 Javascript

3.3 Kibana

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.

3.4 ElasticSearch

3.5 ThreeDC.js

3.6 Three.js

3.7 AngularJS

Chapter 4

Development

4.1 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

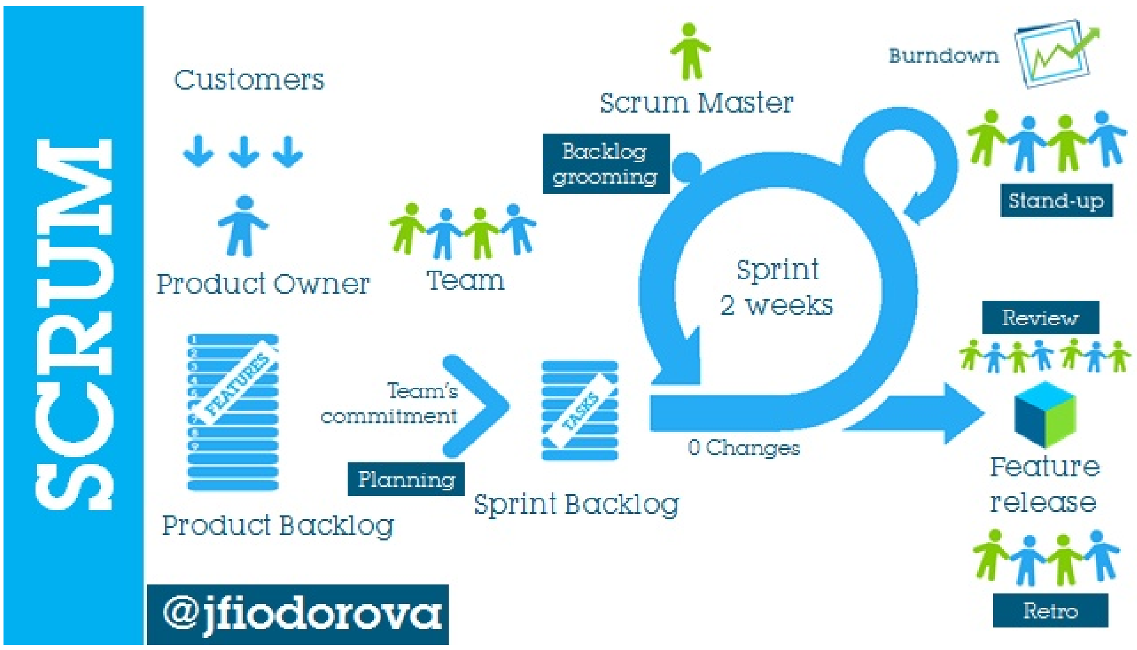


Figure 4.1: Scrum workflow

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**: Separate server showing up elasticSearch data and a threeDC chart
3. **Sprint 2**: Hello World plugin
4. **Sprint 3**: Simple elasticSearch query and visualization in Kibana custom plugin
5. **Sprint 4**: ThreeDC and aux modules integration in Kibana plugin (1. three cube in Kibana. 2. threedc in kibana)
6. **Sprint 5**: 3D Pie Chart
7. **Sprint 6**: Data filtering
8. **Sprint 7**: 3D Bars Chart
9. **Sprint 8**: 3D Bubbles Chart

   TODO: mencionar mejoras en la biblioteca threeDC según voy desarrollando estos sprints. Por ejemplo el rompecabezas que tuve con varias vis en el dashboard

4.2 Sprint 0

4.2.1 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.

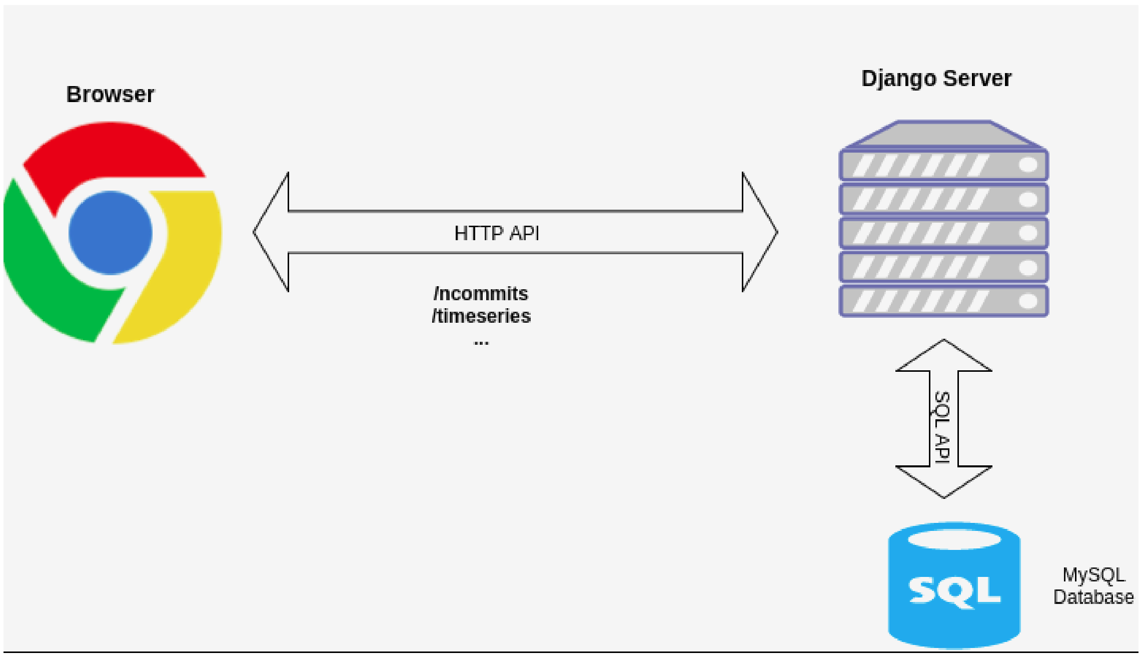


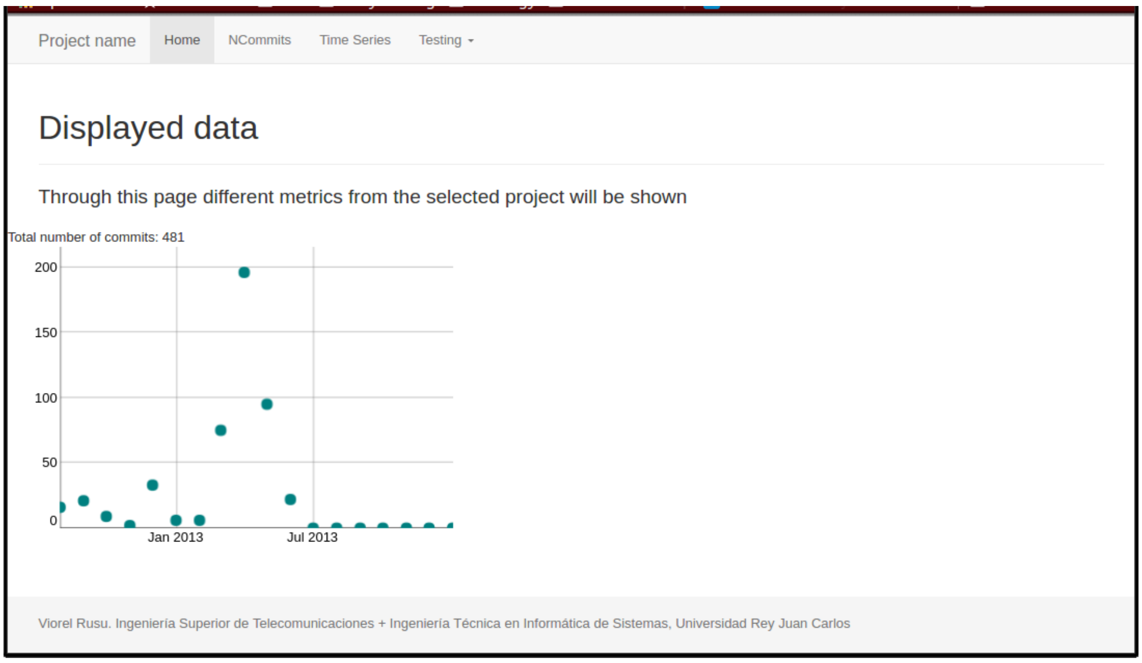
Figure 4.2: Proposed architecture

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
- 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:



Displayed data

Through this page different metrics from the selected project will be shown

Viorel Rusu. Ingeniería Superior de Telecomunicaciones + Ingeniería Técnica en Informática de Sistemas, Universidad Rey Juan Carlos

Figure 4.3: Simple browser visualization of commits timeseries

- 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.

4.2.2 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:

```
curl -XGET 'localhost:9200/_cat/indices?v'
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:

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

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]:

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

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

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 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:

```
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:

```
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.

4.3 Sprint 1

Separate server showing up elasticSearch data and threeDC graphics

This is the only Sprint 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 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

[3]https://github.com/elastic/kibana/blob/master/CONTRIBUTING.md

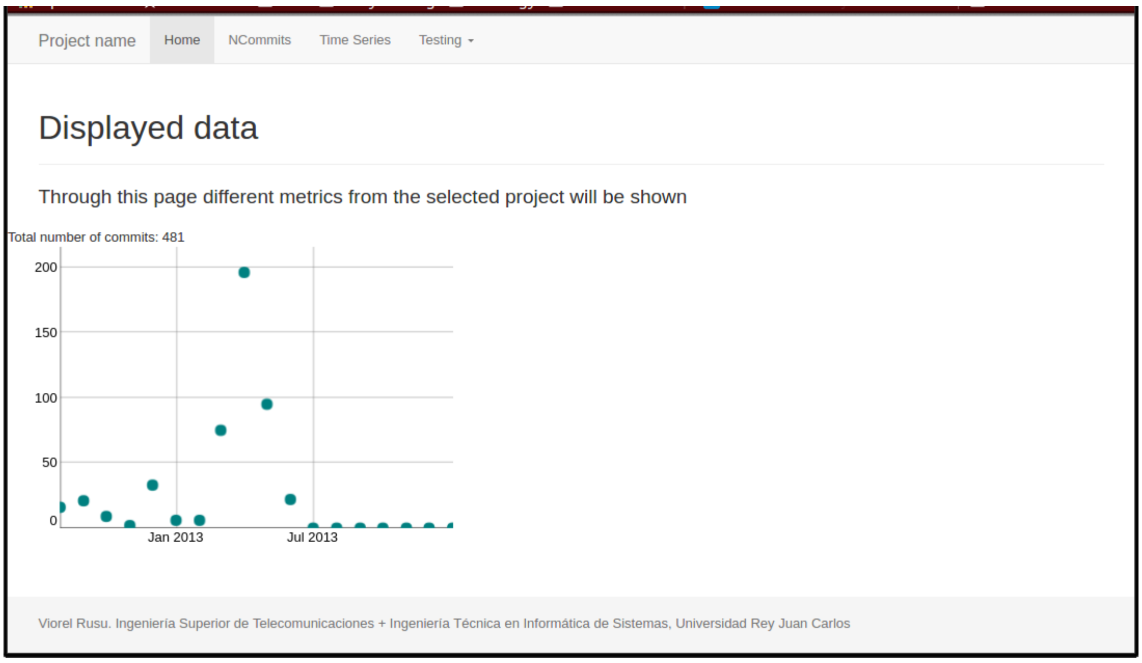

Project name
Home
NCommits
Time Series
Testing
Displayed data
Through this page different metrics from the selected project will be shown
Total number of commits: 481
200
150
100
50
0
Jan 2013
Jul 2013
Viorel Rusu. Ingeniería Superior de Telecomunicaciones + Ingeniería Técnica en Informática de Sistemas, Universidad Rey Juan Carlos

Figure 4.4: Simple browser visualization of commits timeseries

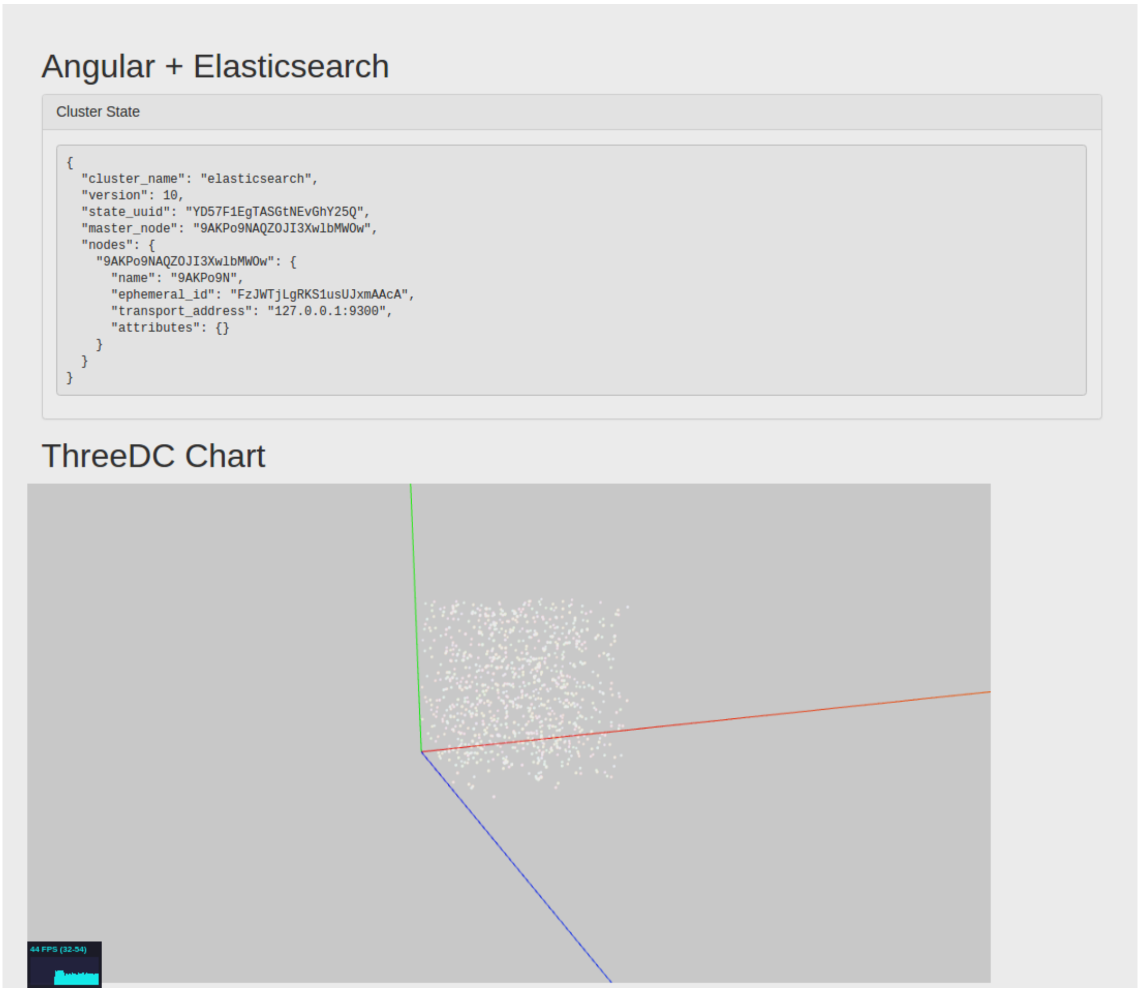
will view a single page with two divs, one containing some sort of elasticSearch data and the other plotting some sort of threeDC charts. This will set the basis for a much more complex server like Kibana, with much more complex queries to elasticSearch and more issues with the threeeDC library.

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:

```
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 6.5 of this paper. You can see the final result of this Sprint in figure 4.5.

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



Angular + Elasticsearch

Cluster State

```
{
  "cluster_name": "elasticsearch",
  "version": 10,
  "state_uuid": "YD57F1EgTASGtNEvGhY25Q",
  "master_node": "9AKPo9NAQZOJI3XwlbMWOw",
  "nodes": {
    "9AKPo9NAQZOJI3XwlbMWOw": {
      "name": "9AKPo9N",
      "ephemeral_id": "FzJWTjLgRKS1usUJxmAAcA",
      "transport_address": "127.0.0.1:9300",
      "attributes": {}
    }
  }
}
```

ThreeDC Chart

Figure 4.5: Html page showing the result of an ES query and a threeDC chart

4.3.1 ElasticSearch interaction

We need a way to interact with 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.

```
ExampleApp.service('client', function (esFactory) {
  return esFactory({
    host: 'localhost:9200',
    apiVersion: '2.3',
    log: 'trace'
  });
});
```

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.

```
    client.cluster.state({
      metric: [
        'cluster_name',
        'nodes',
        'master_node',
        'version'
      ]
      })
    .then(function (resp) {
      $scope.clusterState = resp;
      }
```

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

Then, the result is attached to the scope so it is easy to print this result directly from the html through angular. The result can be seen in the first section of the previous figure 4.5. The complete code in detail can also be found in Appendix A.2.

4.3.2 threeDC graph

Let's build 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:

```
    // set up camera
  camera = new THREE.PerspectiveCamera( VIEW_ANGLE, ASPECT, NEAR, FAR);
  // add the camera to the scene
  scene.add(camera);
  // the camera defaults to position (0,0,0)
  //    so pull it back (z = 400) and up (y = 100) and set the angle
   towards the scene origin
  camera.position.set(0,150,400);
  camera.lookAt(scene.position);

  renderer = new THREE.WebGLRenderer( {antialias:true} );
  renderer.setSize(SCREEN_WIDTH, SCREEN_HEIGHT);
  renderer.setClearColor( 0xd8d8d8 );

  var light = new THREE.PointLight(0xffffff,0.8);
  light.position.set(0,200,250);
  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.

```
THREEDC.initializer(camera,scene,renderer,container);

```

```
  var cloud= THREEDC.pointsCloudChart([0,0,0]);
  cloud.getPoints(getRandomPoints(1000));

  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.

4.4 Sprint 2: Hello World plugin

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

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.

```
{
  "name": "sprint2_plugin",
  "version": "5.0.0"
}
```

- 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:

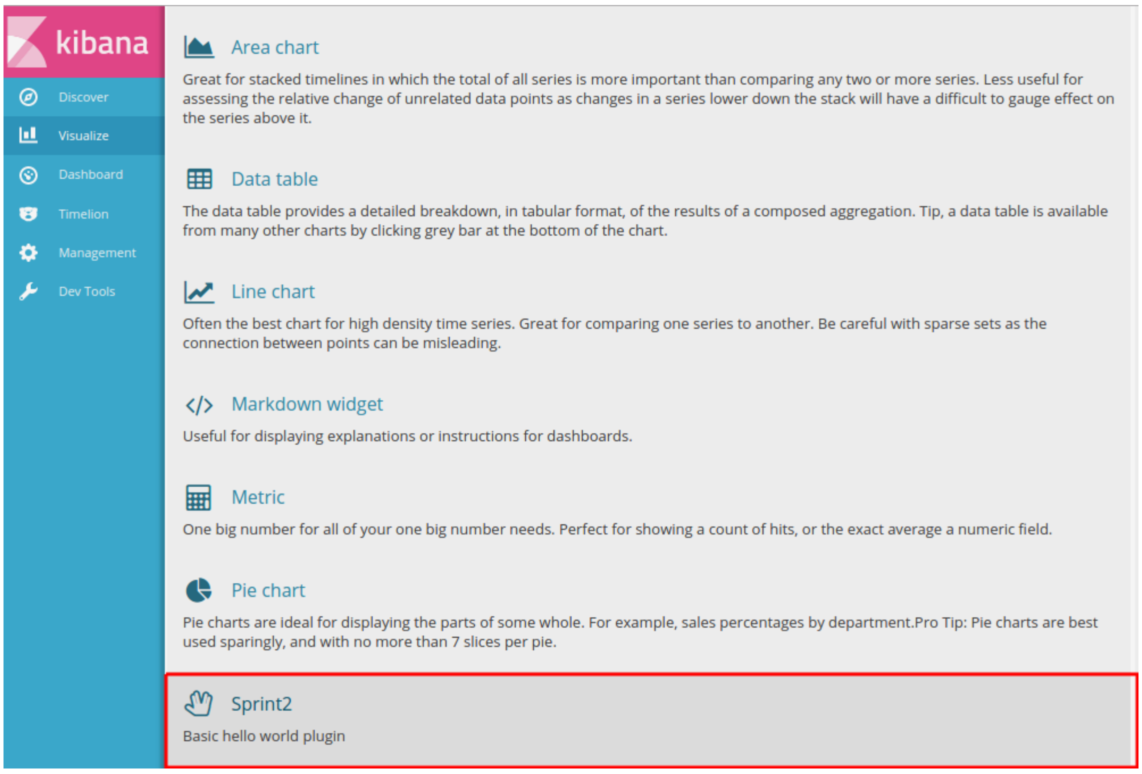
```
export default function (kibana) {

  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 4.6 our first visualization plugin can be seen in the list, together with default Kibana visualization plugins. Figure 4.7 shows the final plugin result. With this basic plugin, we have the skeleton to build everything we want.



Area chart

Great for stacked timelines in which the total of all series is more important than comparing any two or more series. Less useful for assessing the relative change of unrelated data points as changes in a series lower down the stack will have a difficult to gauge effect on the series above it.

Data table

The data table provides a detailed breakdown, in tabular format, of the results of a composed aggregation. Tip, a data table is available from many other charts by clicking grey bar at the bottom of the chart.

Line chart

Often the best chart for high density time series. Great for comparing one series to another. Be careful with sparse sets as the connection between points can be misleading.

Markdown widget

Useful for displaying explanations or instructions for dashboards.

Metric

One big number for all of your one big number needs. Perfect for showing a count of hits, or the exact average a numeric field.

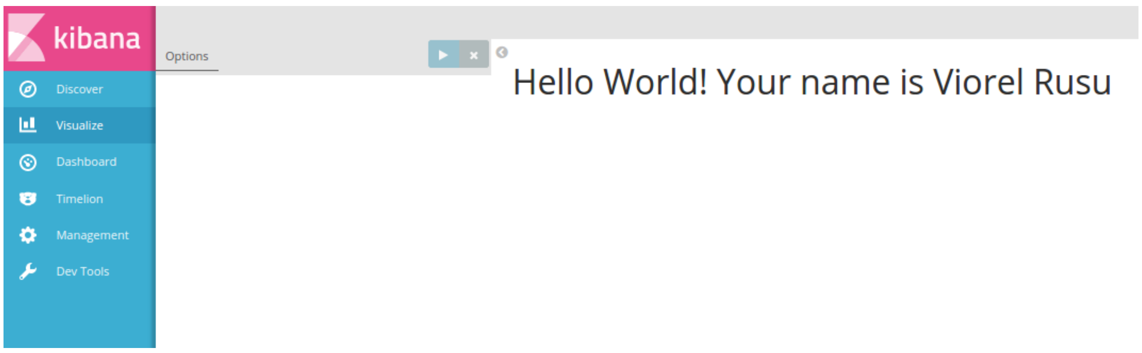
Pie chart

Pie charts are ideal for displaying the parts of some whole. For example, sales percentages by department.Pro Tip: Pie charts are best used sparingly, and with no more than 7 slices per pie.

Sprint2

Basic hello world plugin

Figure 4.6: Hello World plugin in visualizations plugin list



Hello World! Your name is Viorel Rusu

Figure 4.7: Hello World plugin result

4.5 Sprint 3: Simple elasticSearch query and visualization in Kibana custom application

In this sprint 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);
  }
```

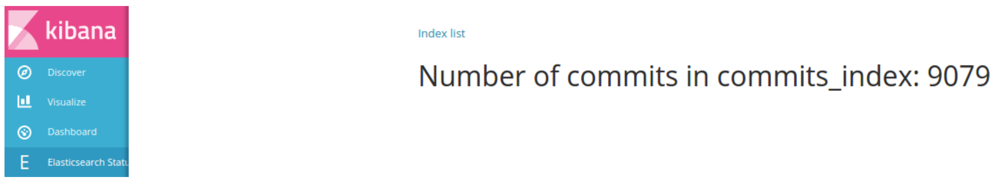
The init method adds a new server API to Kibana. In the api file, we specify the api calls we want to define. In our case, we define the /ncommits GET server API call in the following way:

```
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);
      });
    }
  });
}
```

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:

```
uiRoutes.enable();
.when('/ncommits', {
  template: ncommitsTemplate,
  controller:'elasticsearchNCommitsController',
```



Number of commits in commits_index: 9079

Figure 4.8: Custom app plugin result

```
  controllerAs: 'ctrl'
});

uiModules
.get('app/elasticsearch_status_vio', [])
.controller('elasticsearchNCommitsController', function ($http) {
  $http.get('../api/elasticsearch_status/ncommits').then((response) => {
    this.ncommits = response.data;
  });
});
```

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:

```
<div class="container">
  <div class="row">
    <div class="col-12-sm">
      <a href="#/">Index list</a>
      <h1>Number of commits in commits_index: {{ ctrl.ncommits }}</h1>
    </div>
  </div>
</div>
```

The final result can be seen in Fig 4.8.

4.6 Sprint 4: ThreeDC and aux modules integration in Kibana plugin (1. three cube in Kibana. 2. threedc in kibana)

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 elastic-Search 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.

4.6.1 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.

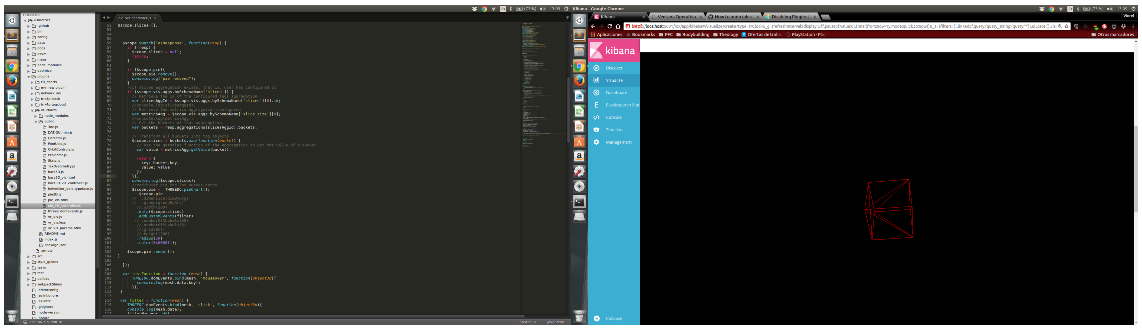
2. From the main plugin file, the new library can easily be imported:

```
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:

```
module.controller('3DCubeController', function($scope, $element){



    var camera, scene, renderer;
    var geometry, material, mesh;

    init();
    animate();



    function init() {

        camera = new THREE.PerspectiveCamera(75, window.innerWidth / window
    .innerHeight, 1, 10000);
        camera.position.z = 1000;

        scene = new THREE.Scene();

        geometry = new THREE.BoxGeometry(200, 200, 200);
        material = new THREE.MeshBasicMaterial({
            color: 0xff0000,
            wireframe: true
        });

        mesh = new THREE.Mesh(geometry, material);

        renderer = new THREE.WebGLRenderer();
        renderer.setSize(window.innerWidth, window.innerHeight);


        var idchart = $element.children().find(".3Dcubechart");

      container = idchart[0];
        container.appendChild(renderer.domElement);
    }

    function animate() {

        requestAnimationFrame(animate);

        mesh.rotation.x += 0.01;
        mesh.rotation.y += 0.02;

        renderer.render(scene, camera);

    }

  });
```



```
$scope.slices=[];



  $scope.$watch('esResponse', function(resp) {
    if (!resp) {
      $scope.slices = null;
      return;
    }

    if ($scope.pie){
      $scope.pie.remove();
      console.log("pie removed");
    }
    //if slices aggregation exists, that is, user has configured it
    if ($scope.vis.aggs.bySchemaName['slices']) {
      // Retrieve the id of the configured tags aggregation
      var slicesAggId = $scope.vis.aggs.bySchemaName['slices'][0].id;
      //console.log(slicesAggId);
      // Retrieve the metrics aggregation configured
      var metricsAgg = $scope.vis.aggs.bySchemaName['slice_size'][0];
      //console.log(metricsAgg);
      // Get the buckets of that aggregation
      var buckets = resp.aggregations[slicesAggId].buckets;

      // Transform all buckets into tag objects
      $scope.slices = buckets.map(function(bucket) {
        // Use the getValue function of the aggregation to get the value of a bucket
        var value = metricsAgg.getValue(bucket);

        return {
          key: bucket.key,
          value: value
        };
      });
      console.log($scope.slices);
      //redibujar pie con los nuevos datos
      $scope.pie =  THREEDC.pieChart();
         $scope.pie
      //  .dimension(dimByOrg)
      //  .group(groupByOrg)
        //.width(200)
        .data($scope.slices)
        .addCustomEvents(filter)
       // .numberOfXLabels(50)
        //.numberOfYLabels(5)
        //.gridsOn()
        //.height(200)
        .radius(50)
        .color(0x0000ff);

    $scope.pie.render();
}

  });

  var testFunction = function (mesh) {
      THREEDC.domEvents.bind(mesh, 'mouseover', function(object3d){
        console.log(mesh.data.key);
      });
  }

 var filter = function(mesh) {
    THREEDC.domEvents.bind(mesh, 'click', function(object3d){
    console.log(mesh.data);
    filterManager.add(
```

Figure 4.9: Three.js scene correctly inserted in Kibana

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.

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

The final result can be seen in Figure 4.9

4.6.2 ThreeDC.js scene integration

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 fonts or

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

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.

```
 require("plugins/3D_kibana_charts_vis/FontUtils");
 require("plugins/3D_kibana_charts_vis/TextGeometry");
 require("plugins/3D_kibana_charts_vis/Projector");
 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.

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

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

Once imported all these modules, threeDC scenes can be created. Basic scenes have been created and correctly tested, but won't be included here for briefty and because there there will be plenty of them in the following sprints.

4.7 Sprint 5: 3D Pie Chart

At this point in our progress, the plugin has the following structure:

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/writing-kibana-4-plugins-simple-visualizations/

4.8 Sprint 6: Data filtering

4.9 Sprint 7: 3D Bars Chart

4.10 Sprint 8: 3D Bubbles Chart

Chapter 5

Resultados

Chapter 6

Conclusiones

6.1 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.

6.2 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

6.3 Lecciones aprendidas

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

1. a

2. b

6.4 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.

6.5 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

```
<!DOCTYPE html>
<html>
  <head>
    <title>Threeboard</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, user-scalable=no,
   minimum-scale=1.0, maximum-scale=1.0">
    <link rel="stylesheet" type="text/css" href="node_modules/bootstrap/
   dist/css/bootstrap.css">
  </head>
  <body ng-app="ExampleApp">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery
   .min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r78/three.
   min.js"></script>
    <script src="js/FontUtils.js"></script>
    <script src="js/TextGeometry.js"></script>
    <script src="js/Projector.js"></script>
    <script src='js/threex.domevents.js'></script>
    <script src="js/Detector.js"></script>
    <script src="js/Stats.js"></script>
    <script src="js/OrbitControls.js"></script>
    <script src="js/THREEx.WindowResize.js"></script>
    <script src="js/THREEx.FullScreen.js"></script>
    <script type="text/javascript" src='js/DAT.GUI.min.js'></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/crossfilter/1.3.12/
   crossfilter.min.js"></script>
    <script src="js/3dc.js"></script>
    <div id="ThreeJS" style="position: absolute; left:0px; top:0px">
    <script src="fonts/gentilis_bold.typeface.js"></script>
    <script src="fonts/gentilis_regular.typeface.js"></script>
    <script src="fonts/optimer_bold.typeface.js"></script>
    <script src="fonts/optimer_regular.typeface.js"></script>
    <script src="fonts/helvetiker_bold.typeface.js"></script>
    <script src="fonts/helvetiker_regular.typeface.js"></script>
    <script src="fonts/droid_sans_regular.typeface.js"></script>
    <script src="fonts/droid_sans_bold.typeface.js"></script>
    <script src="fonts/droid_serif_regular.typeface.js"></script>
    <script src="fonts/droid_serif_bold.typeface.js"></script>

  <!-- include npm modules in proper order -->
  <script src="node_modules/angular/angular.min.js"></script>
  <script src="node_modules/elasticsearch-browser/elasticsearch.angular.min
   .js"></script>
    <script src="script.js"></script>

  <!-- attach the ExampleController to our main content -->
  <div ng-controller="ExampleController" class="container">
    <h1>Angular + Elasticsearch</h1>

    <!-- if there is an error, display its message -->
    <div ng-if="error" class="alert alert-danger" role="alert">{{error.
   message}}</div>

    <!-- if clusterState is available, display it as formatted json -->
    <div ng-if="clusterState" class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title">Cluster State</h3>
      </div>
      <div class="panel-body">
        <pre>{{clusterState | json}}</pre>
      </div>
    </div>
  </div>

  </body>
</html>
  </body>
</html>
```

A.2 script.js from Sprint1

```

// standard global variables
var container, scene, camera, renderer, stats;

//JSON data saved here
var json_data;


 elasticstuff();

// initialization
  //getJSON call, draw meshes with data
   $.getJSON("jsons/scm-commits.json", function(data) {
      json_data=data;
      init();
      // animation loop / game loop
      animate();
   });

///////////////
// FUNCTIONS //
```

```
///////////////

function elasticstuff() {
  //elasticsearch and angular
    // App module
    //
    // The app module will contain all of the components the app needs (
   directives,
    // controllers, services, etc.). Since it will be using the components
   within
    // the elasticsearch module, define it a dependency.
    var ExampleApp = angular.module('ExampleApp', ['elasticsearch']);

    // Service
    //
    // esFactory() creates a configured client instance. Turn that instance
    // into a service so that it can be required by other parts of the
   application
    ExampleApp.service('client', function (esFactory) {
      return esFactory({
        host: 'localhost:9200',
        apiVersion: '2.3',
        log: 'trace'
      });
    });

    // Controller
    //
    // It requires the "client" service, and fetches information about the
   server,
    // it adds either an error or info about the server to $scope.
    //
    // It also requires the esFactory to that it can check for a specific
   type of
    // error which might come back from the client
    ExampleApp.controller('ExampleController', function ($scope, client,
   esFactory) {

      client.cluster.state({
        metric: [
          'cluster_name',
          'nodes',
          'master_node',
          'version'
        ]
        })
      .then(function (resp) {
        $scope.clusterState = resp;
        $scope.error = null;
      })
      .catch(function (err) {
        $scope.clusterState = null;
        $scope.error = err;

        // if the err is a NoConnections error, then the client was not
    able to
        // connect to elasticsearch. In that case, create a more detailed
    error
        // message
        if (err instanceof esFactory.errors.NoConnections) {
          $scope.error = new Error('Unable to connect to elasticsearch. ' +
            'Make sure that it is running and listening at http://localhost
    :9200');
        }
      });

    });
}

function init () {

   ///////////
   // SCENE //
   ///////////
```

```
  scene = new THREE.Scene();

  ////////////
  // CAMERA //
  ////////////
  // set the view size in pixels (custom or according to window size)
  var SCREEN_WIDTH = window.innerWidth/2;
  var SCREEN_HEIGHT = window.innerHeight/2;
  // camera attributes
  var VIEW_ANGLE = 45;
  var ASPECT = SCREEN_WIDTH / SCREEN_HEIGHT;
  var NEAR = 0.1;
  var FAR = 20000;
     // set up camera
  camera = new THREE.PerspectiveCamera( VIEW_ANGLE, ASPECT, NEAR, FAR);
  // add the camera to the scene
  scene.add(camera);
  // the camera defaults to position (0,0,0)
  //    so pull it back (z = 400) and up (y = 100) and set the angle
  towards the scene origin
  camera.position.set(0,150,400);
  camera.lookAt(scene.position);

  //////////////
  // RENDERER //
  //////////////
  renderer = new THREE.WebGLRenderer( {antialias:true} );
  renderer.setSize(SCREEN_WIDTH, SCREEN_HEIGHT);
  renderer.setClearColor( 0xd8d8d8 );

  // attach div element to variable to contain the renderer
  container = document.getElementById( 'ThreeJS' );
  // attach renderer to the container div
  container.appendChild( renderer.domElement );

   ////////////
 // EVENTS //
```

```
////////////



// automatically resize renderer
THREEx.WindowResize(renderer, camera);
  // toggle full-screen on given key press
THREEx.FullScreen.bindKey({ charCode : 'm'.charCodeAt(0) });

 ///////////
 // LIGHT //
 ///////////
 var light = new THREE.PointLight(0xffffff,0.8);
 light.position.set(0,200,250);
 scene.add(light);
 var ambientLight = new THREE.AmbientLight(0x111111);
 // scene.add(ambientLight);

 // create a set of coordinate axes to help orient user
 //    specify length in pixels in each direction
 var axes = new THREE.AxisHelper(1000);
 scene.add(axes);

//STATS
stats = new Stats();
stats.domElement.style.position = 'absolute';
stats.domElement.style.bottom = '0px';
stats.domElement.style.zIndex = 100;
container.appendChild( stats.domElement );

 //////////////
 // CUSTOM //
 //////////////

 // most objects displayed are a "mesh":
 //  a collection of points ("geometry") and
 //  a set of surface parameters ("material")
```

```

  var parsed_data=[];

  // Crossfilter and dc.js format
  json_data.values.forEach(function (value) {
    var record = {}
    json_data.names.forEach(function (name, index) {
        if (name == "date") {
          var date = new Date(value[index]*1000);
          record[name] = date;
          record.month = new Date(date.getFullYear(), date.getMonth(), 1);
          record.hour = date.getUTCHours();
        } else {
          record[name] = value[index];
        }
    });
    parsed_data.push(record);
  });

  //example data for cloud

  function getRandomPoints(numberOfPoints){
    var points=[];
    for (var i = 0; i < numberOfPoints; i++) {

      points[i]={x:Math.random()*100,y:Math.random()*100,z:Math.random()
   *100};
     // console.log(points[i]);
    };
    return points;
  }

 //CUSTOM DASHBOARD//

  THREEDC.initializer(camera,scene,renderer,container);

  var cloud= THREEDC.pointsCloudChart([0,0,0]);
  cloud.getPoints(getRandomPoints(1000));

 // var bars= THREEDC.barsChart([0,0,0]);
  //bars.group(groupByRepo);

  THREEDC.renderAll();


}

function animate()
{
   requestAnimationFrame( animate );
   render();
   update();
}

function render()
{
   renderer.render( scene, camera );
}

function update()
{
  THREEDC.controls.update();
  stats.update();
```

[frame=single]

A.3 sprint2.js from Sprint2

```

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

    $scope.name="Viorel Rusu"

  });


export default function Sprint2Provider(Private) {
    var TemplateVisType = Private(require('ui/template_vis_type/template_vis_type'));
    return new TemplateVisType({
      name: 'Sprint2', // the internal id of the visualization
      title: 'Sprint2', // the name shown in the visualize list
      icon: 'fa-hand-spock-o', // the class of the font awesome icon for this
      description: 'Basic hello world plugin', // description shown to the user
      requiresSearch: false, // Cannot be linked to a search
      template: require('plugins/sprint2_plugin/sprint2.html') // Load the template of the visualization
    });
  }

  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/

Bibliography

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Articial Systems*. Oxford University Press, Inc., 1999.