Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
DE TELECOMUNICACIÓN

DOBLE INGENIERÍA DE TELECOMUNICACIÓN E
INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

PROYECTO FIN DE CARRERA

Creation of an HTML5 application for Software Development Dashboards visualization

(Creación de aplicación HTML5 para la visualización de Software Development Dashboards)

Autor: Juan Carlos Cámara Checa

Tutor: Jesús M. González-Barahona

Curso académico 2013/2014

Abstract

This document contains the description and development process of the Software Development Dashboard HTML5 application. The project has been carried out with the objective of adding more functionality and dynamism to dashboards. A Software Development Dashboard is a set of graphs and data which show information taken from software development repositories.

One of the problem of current Software Development Dashboards is that most of them are static and the data which is visualized cannot be specified dynamically. The main goal of this project is creating a web application that would improve this situation for users and fix this problem.

Another main objective of the final application is to be a multi-platform application. Between the big amount of possibilities, the project is decided to be created using HTML5, JavaScript and CSS3 technologies. HTML5 set of technologies allows application to be executed in most modern browsers, and it is also possible to create native mobile applications for different mobile operative systems from HTML5 code through frameworks such as Cordova or Titanium.

The application is able to show, using different graph libraries, some different widgets related with Software Development. In the end, the user will be the only one who will specify which info will be shown. To improve user experience, the web application has been developed based on a draggable and resizable widget grid. Due to this, user will not be able only to manage the info he wants to show, but also size, shape and position of all of the widgets and graphs.

The whole information that the Dashboards will show, and also all kind of configuration attributes, is received from HTML5 tags and JSON files. Server side functionality has been done with PHP5 programming language, implemented as a REST JSON server to provide all the JSON files just announced. To add a more social and multiuser point of view, application has been also designed to include dashboard sharing functionality.

Resumen

Este documento contiene la descripción y proceso de desarrollo de la aplicación HTML5 de Software Development Dashboard (en español paneles de seguimiento sobre desarrollo de software). El proyecto se ha llevado a cabo con el objetivo de añadir más funcionalidad y dinamismo a los paneles de seguimiento actuales. Un panel de seguimiento sobre desarrollo de software es un conjunto de gráficas y datos que muestran información tomada desde distintos proyectos de control de versiones.

El problema con el que se encuentran los actuales paneles de seguimiento es que la mayoría de ellos son estáticos y los datos visualizados en ellos, así como su contenido o su distribución, no puede ser especificado dinámicamente ni a elección del usuario. La meta de este proyecto es crear una aplicación que mejore este comportamiento para los usuarios, así como este problema.

Además de las anteriores, para el desarrollo del software existía la especificación de que la aplicación fuese multiplataforma. Entre la gran variedad de posibilidades de desarrollo, se decidió que el proyecto fuese creado usando tecnología HTML5, JavaScript y CSS3. Este conjunto de lenguajes de programación permite una aplicación ser ejecutada en la mayoría de los navegadores, y es también posible crear aplicaciones móviles nativas para distintos sistemas operativos a partir de código HTML5.

El software es capaz de mostrar, usando diferentes librerías gráficas, diferentes widgets con datos relacionados con el Desarrollo de Software. En última instancia, el usuario será el único que indique la información que se mostrará a través de ellos. Para mejorar la experiencia de usuario, la aplicación ha sido desarrollada sobre una estructura de widgets desplazables y estructurales, lo que permite al usuario gestionar el tamaño y disposición de las gráficas y widgets, junto a su contenido.

Todo el conjunto de información mostrada, ya sean datos o configuraciones del Dashboard, se recibe y procesa a partir de etiquetas HTML5 y archivos JSON. Existe además una funcionalidad de servidor implementada como servidor REST JSON. Además de proporcionar los JSON previamente indicados, el servidor se utiliza para añadir funcionalidad social y de multiusuario.

Index

1. Introduction

This chapter summarizes preliminary information that the reader should get for a complete understanding of the results and temporal evolution of the project.

This document provides a complete explanation about Software Development Dashboards and their functionality, and why are they used by companies. Furthermore, this document will explain the code, frameworks and libraries that have been used for developing the application.

Project has been created following a simplified version of the Scrum software development methodology. Temporal evolution and sprints will be explained in detail to get the user knowing the same steps and tasks that were taken to develop final application.

This application has been created as a free software project, and can be completely forked to be modified from its gitHub repository[1]. A functional copy of the project can be found in its Heroku server[2].

For future references, the application, as well as the project, will be referenced with Version Control Dashboard name.

1.1. Project motivation

Version control has become very popular between software developers society. Programs like Git or SVN, among others, allow developers to create more reliable, effective and cleaner code. Furthermore of the version control functionality that these programs provide to developers and users, it is possible to achieve different data about some repository behaviour and its temporal evolution. Most of this data is statistical data.

Whole this information is processed by other utilities, called Software Development Dashboards, that show this data through different graphs and widgets to provide users more information about repository evolution and how the related project is going on.

[1] GitHub application repository https://github.com/JuanCarlosCamara/Dashboard-pfc
[2] Application at Heroku server http://versioncontroldashboard.Herokuapp.com.

However, information provided by current Software Development Dashboards is not really useful to user. User cannot normally choose which information to be visualized or the way it is shown, neither any option related with widgets or graphs.

Creating some dynamic behaviour to current Software Development Dashboard projects would improve their user experience, as well as providing a higher knowledge about repository and data meaning.

1.2. Project goals

This project tries to create a Software Development Dashboard application which fixes this problem previously announced, providing dynamism and high user experience level to user. All used technologies and how the application works will be explained in next steps.

The objective of this project is creating a multiplatform application. The application is decided to be developed as a HTML5 web, able to be run with different browsers and devices.

The reason why HTML5 is chosen as programming language for the application, instead of any other language, is because HTML5 web applications feature common code for different platforms. Furthermore, it is possible to create native mobile applications for different mobile devices using some tools, as Cordova. This document will be back later to these tools.

Project specification includes different functionality, that is listed next as a scrum user history:

- As user, I want to be able to create an empty dashboard.
- As user, I want to be able to insert widgets into a dashboard from a list with all of them, to choose which one to add.
- As user, I want to be able to create different tabs to organize the content of a dashboard
- As user, I want to be able to remove a previous inserted widget from a dashboard.
- As user, I want to be able to change the widget size and dimensions, in order to show its content in a higher size.

- As user, I want to be able to modify my dashboard structure, as well as moving all widgets positions.
- As user, I want to be able to share a created dashboard to another user, to avoid him to see, modify and share its content.

The project development methodology to be followed is Scrum agile methodology, and each of the previous points corresponds to a different sprint. Agile methodology will be explained later, in another step of this document.

Besides previous information, the project has to be created as a free software project. The reason for this is that free software developer community can make a project grow up faster, as a result of the amount of people who belong to it. Whole community can modify and improve source code, fix bugs and integrate the project into society in a faster way. At the end of the last sprint there will be a functional copy of the source code, able as free software in GitHub repository.

1.3. Report content sections

To understand completely the development of the project and all its features, along this document Version Control Software and Software Development Dashboards will be completely described. All features, history and different examples will be named, as well as their different branches.

Project code will also be described. As it was said before, project has been developed using HTML5 technology. There is a chapter of this document specially prepared for readers that have not work with this kind of technology before. This chapters covers some fundamentals of HTML, as well as CSS and JavaScript. If a reader reads this chapter, later code fragments should be completely understood.

Before dealing with these concepts, there will be a chapter specially written to describe basic fields related with the project environment. This chapter, as well as HTML chapter, is especially dedicated to non technical readers, to provide them a bigger knowledge about project field.

Furthermore, there will be a chapter describing the whole development process. As it was written before, development methodology which has been chosen for this project has been Scrum agile methodology. This chapter will explain each sprint, including all found difficulties during development, as well as third part tools that has been used and the way they are integrated with written native code.

In the chapter about the development process most of the content is related with HTML5, so that it is very important for a non skilled user to read previous chapter specially dedicated to HTML technology in order to get some knowledge and skill with this language syntax.

After development process chapter, there will be two more chapters, including results and conclusions. First one include final prototype functionality description, with user experience examples and images. Between others, the creation of a dashboard, add and modify widgets and project sharing will be exampled. On the other hand, conclusions chapter will include a little descriptions about the project development, his good and bad points and his current status and how it can be improved by free software society.

2. State of the art

This section explains main blocks which the project has worked around, from Version Control systems to HTML, as well as several ideas about technical concepts used along the document.

These concepts will help to add fundamental knowledge about the project thread and its technologies, besides of understanding the project domain, as well as project code and its terms.

2.1. Description of related technology fields

Before starting explaining big blocks, some fundamental concepts, programming details and other informatics notions related with application environment are explained. However, these concepts will not be explained in details, but just a simple description of the concept to make the reader has a simple idea of main term.

2.1.1. Communication architecture between client and server

There are many protocols created to communicate processes over the Internet. HTTP protocol has been mentioned previously, and it is the normal way that browsers and other protocols use to exchange information. This protocol is used in many cases with the REST architecture technique. Systems and software which follow this standard are commonly known as RESTful systems.

Main concept in REST architecture is existence of resources which can be accessed using a global unique identifier. Through this identifier, called URI, resources can be managed and modified, using a standard HTTP interface. With this interface, resources status can be changed and propagated through network.

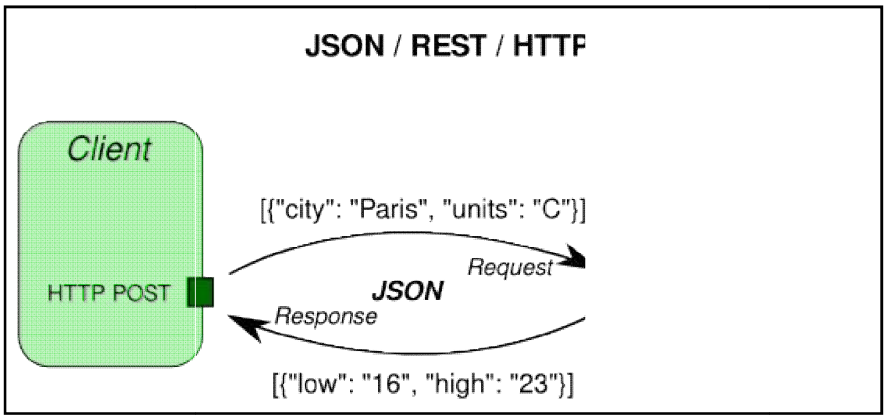

JSON / REST / HTTP
Client
HTTP POST
[{"city": "Paris", "units": "C"}]
Request
JSON
Response
[{"low": "16", "high": "23"}]

Figure 1: REST JSON communication example

There are only four allowed functions for resources, that are GET, POST, PUT and DELETE. The reason why there are only four functions is that REST architectures are thought to alter a network full of "things" and "objects". These previous methods functionality is, respectively, obtain, update, create or delete object instances.

Moreover, many applications run some code in an intermediate remote machine, which only returns execution result. For this, RPC (Remote Procedure Call) protocols were created, and they provide a wide variety of functions to do that. SOAP or XML-RPC are some examples of this kind of communication protocol.

Main difference between RESTful and RPC systems is that, while RESTful architectures only have four possible operations to do, RPC systems are featured by an extensive IDL (Interface Description Language) with the whole set of operations that server can do.

2.1.2. Data change between client and server

To carry out communications previously named, it is also necessary to specify the way the information is formatted. Any of the previous protocols, RPC or REST, may send in their messages some different information to the server in order to carry out some specific action.

Because there must be synchronization between client and server to make a clear communication, some pre-defined formats or data structures exist. To establish a communication over HTTP, main of these data structures are XML, HTTP and JSON.

XML and HTML are markup languages that define how documents or objects must be structured. Each mark specify a little section of the document, ready to be read by humans or machines at the same time. Any of these marks has been named labels or tags.

Even though both XML and HTML are markup languages, divided into tags and labels, their purposes are different, as well as the rules they follow.

On one hand, HTML is normally used to define web documents, ready to be requested by browsers. Browsers interpret HTML tags and show the page content.

On the other hand, XML is used as data structure to exchange information about an object. As it will be seen, is quite similar to JSON data structures, but with a more similar syntax to HTML. There is another difference between HTML and XML, which is that XML files must follow more strict syntax rules[3]. XML specification is focused on creating only well-formed documents, while HTML is focused in communicating the HTML content.

JSON (JavaScript Object Notation), as its own name says, is created to store information following the same structure which data is stored in JavaScript objects. JSON does not use tags or markups, that provides a simplicity that improves data transfer process and make the data extraction process easier. This is the reason why JSON has almost become a substitute of XML.

2.1.3. Web programming languages

Any application, which carries out any action, as exchanging data, modifying objects or doing calls to servers, is written with its own source code. In the same way that people from different countries communicate each other in a different way, there exist different ways to establish a communication between developers and machines.

Programming languages are the set of rules, syntax and words which make possible a developer to communicate with a machine, and make her doing some specific instructions, which are grouped to create a program. There are plenty of programming languages. C, Java, JavaScript or PHP are some examples. As it was said before, in this project development, HTML technologies have been used, as well as a little REST JSON server done with PHP.

[3] XML syntax rules http://www.w3schools.com/xml/xml_syntax.asp

Programming languages used in this set of technologies have been JavaScript and PHP. HTML and CSS are being excluded as programming languages because they are not really programming languages, but more structure definition and design languages.

Both, PHP and JavaScript, are programming languages designed to create dynamic web pages. However, while JavaScript is executed directly in the browser, PHP is running as server program. In this project, dynamic functionality has been done only with JavaScript and PHP has been created as server to implement share functionality.

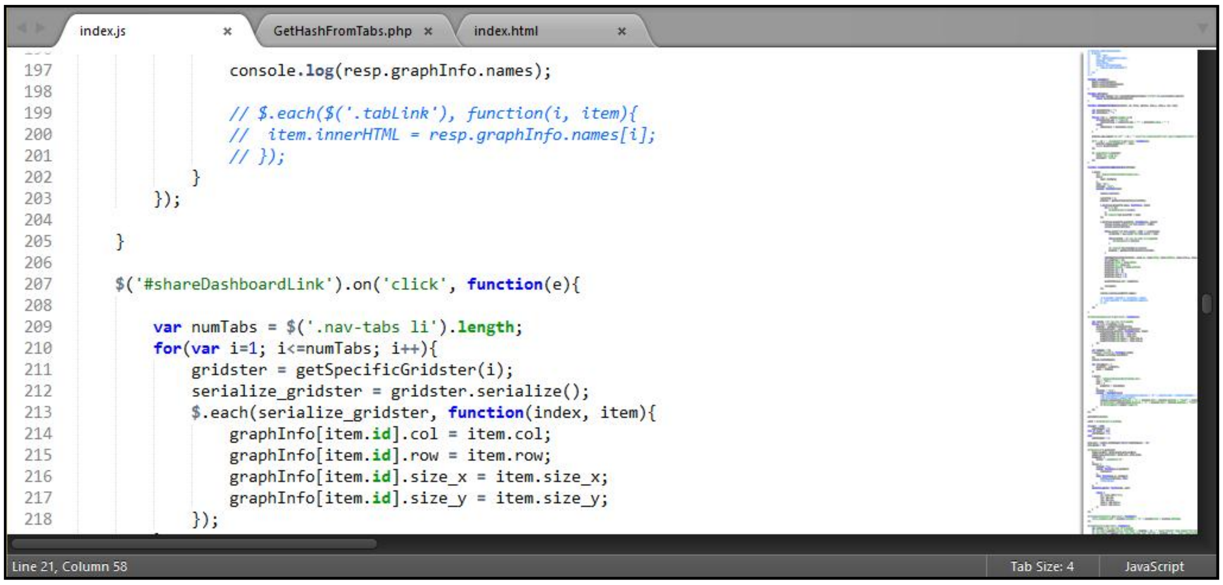
2.1.4. Integrated Development Environment

An Integrated Development Environment (IDE) is a software application, which provides development facilities. An IDE normally includes a source code text editor, build automation and some different compiler/debugger tools, depending of the programming language is destined for.

There exist a big amount of different IDEs, with more or less functionality. The simplest ones just contain the source code text editor, like Vim or Notepad++. Others are more complex like Eclipse, Microsoft Visual Studio or Netbeans. All of them include debugger, compiler and option to include plugins for adding other programming languages, as well as all their related tools.

These more complex IDEs can be destined to Visual programming, a kind of programming by which a developer can create software applications just moving blocks, and designed the software in a more visual way.

To carry out the project, there were other options, specially created to HTML and web development, like Aptana Studio, Dreamweaver or Sublime. The chosen one was the last one. Sublime[1] is a sophisticated text editor for code, markup and prose. Besides of text editor, it includes different tools that allow *Multiple Selections*, *Distraction Free Mode, Command Palette and Split Editing,* between others.



```
                console.log(resp.graphInfo.names);

                // $.each($('.tabLink'), function(i, item){
                //  item.innerHTML = resp.graphInfo.names[i];
                // });
            }
        });

    }

    $('#shareDashboardLink').on('click', function(e){

        var numTabs = $('.nav-tabs li').length;
        for(var i=1; i<=numTabs; i++){
            gridster = getSpecificGridster(i);
            serialize_gridster = gridster.serialize();
            $.each(serialize_gridster, function(index, item){
                graphInfo[item.id].col = item.col;
                graphInfo[item.id].row = item.row;
                graphInfo[item.id].size_x = item.size_x;
                graphInfo[item.id].size_y = item.size_y;
            });
```

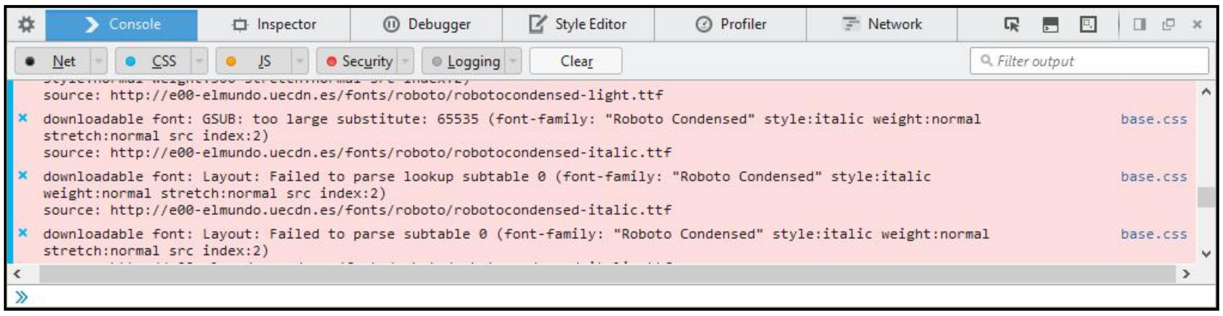
Figure 2: Sublime text IDE.

2.1.5. Browsers

A web browser, or just browser, is a software application used to locate, retrieve and display content on the World Wide Web. This content, also called resources, is identified by a unique identifier, its Uniform Resource Identifier (URI). Resources can be from any kind, like Web pages, Videos, JSON files, images...

As it was said before, web application execute in client side, in the browsers. Currently there are different browsers, from Internet Explorer to Opera, Mozilla Firefox or Google Chrome, between others. Through HTML technology, it is possible to run any web application in most of web browsers, with some minimal changes between each other. Web application that is developed should be correctly designed to be adjusted to each web browser.

When developing web applications, browsers are especially important because they usually have some developer tools, as debugger, JavaScript console and network message grid, between others. All these tools provide developer a lot of facilities when web application is being developed, in order to see the program flow and issues.



```
source: http://e00-elmundo.uecdn.es/fonts/roboto/robotocondensed-light.ttf
× downloadable font: GSUB: too large substitute: 65535 (font-family: "Roboto Condensed" style:italic weight:normal      base.css
stretch:normal src index:2)
source: http://e00-elmundo.uecdn.es/fonts/roboto/robotocondensed-italic.ttf
× downloadable font: Layout: Failed to parse lookup subtable 0 (font-family: "Roboto Condensed" style:italic            base.css
weight:normal stretch:normal src index:2)
source: http://e00-elmundo.uecdn.es/fonts/roboto/robotocondensed-italic.ttf
× downloadable font: Layout: Failed to parse subtable 0 (font-family: "Roboto Condensed" style:italic weight:normal     base.css
stretch:normal src index:2)
```

Figure 3: Mozilla Firefox developer tools.

2.1.6. Agile development methodology

A Software development methodology is a framework which is used to structure, plan and control the process of developing an information system or software. A wide variety of such strategies have evolved over the years, each with its own recognized strengths and weaknesses.

Every software development methodology acts as a basis for applying specific frameworks to develop and maintain software. All several software development techniques can be classified into two big groups: Software development life cycle and agile methodologies.

In some methodologies, whole software development is included into a life cycle. These methodologies divide the life cycle into different steps, for example, planning, creating, testing and deploying. As a function of the development methodology, it is possible that each of these steps take place once or more, with iterative calls.

There is a problem with previous methodologies. During a project execution, some unpredictable situations may happen. With this situations, some parts, or maybe the whole project can change at any moment. Life cycle methodologies were not ready to face this unpredictability, and most of the projects could probably go down.

Agile methodologies are specially prepared to face this unpredictability and get a higher success probability in the project development. Even though there are some different methodologies, and each of them has its own rules, all they have the common feature that they base their solution through collaboration between self-organization, cross-functional teams.

Scrum[2] is the most popular agile methodology. Scrum is featured by three roles in the team: Product owner, development team and scrum master. Product owner is the client of the final products. If the client cannot play this role, then someone else acts like him. Scrum master is the team manager and the development team is the set of people who will develop the project. Scrum defines a concept called sprint, which is a period of time between two prototype deliveries.

In Scrum, there are a big number of meetings per sprint between product owner and development team. Each sprint, development team shows product owner a new prototype of the project, with additional functionality. In this way, the client can see quickly how the project is evolving and can communicate them if there is something to be changed. If this happens, development team can quickly adapt the new change to the previous schedule.

2.1.7. Version control

When carrying out a project, it is normal that it is not only developed by one only person, but many of them. In this situation, development team may manipulate plenty of files and information. With so many people and so many files, it is important to have a system that provides different facilities to development process. Version control programs provide these facilities.

Version control, also called *Revision control* or *Source control* is the management of changes to documents, computer programs, large websites and other collections of information. Every change made to the source code is tracked, along with who made the change, why they made it or the description change. Between others, version control software provide avoiding file change conflicts between users, merging of file changes, file recovery and file previous version recovery.

Version control systems are essentially for any form of distributed, collaborative development. Whether it is the history of a wiki page or large software development project, the ability of tracking each realized changes, and reversing them when necessary can make all the difference between a well managed and controlled process and an uncontrolled system.

There exist different version control systems. Through the history, version control systems have evolve and become a bit different between each other. Terminology can vary between them, but some terms in common usage include: branch, change, checkout, commit, conflict, head, merge, repository, tag, trunk or working copy.

Version control systems, besides of making developing easier to users, provide statistical data from repositories, like the number of commits, added or removed lines, authors, ... per period of time, between others. Whole this data is the data which is processed in Software Development Dashboards. More information will come later.

Through history, version control systems have evolve different between each other. Following git description[3], there are three main types of version control systems, which are Local, Centralized and Distributed version control systems.

Local version control is commonly created for one person work, not interaction or collaboration work. What people basically do is copy working files into different directories, but this is a big mistake. It can be really normal to overwrite accidentally files or directories.

Main architecture of a Centralized version control system is a single server where all versioned and working files are included. Every new client and developer have to check out these files from the central place and, after it, they can interact and develop as parallel processes, but with a common code. This central server can be a problem if it falls or suffers any attack.

In Distributed system, clients download full repository from an original specified server. After it, they work over their local repository. Distributed version control systems are able to connect directly between different clients, not only with a central server. In this way, if there is any problem with the original server, the file historic tree is not lost, because it is distributed among the users.

2.2. Software Development Dashboards

2.2.1. What is a Software Development Dashboard? Content and metrics.

As it was said before, Version Control systems provide different statistical data taken from the different changes along the tree history of a project. All this data can be processed in order to manage and improve the project development.

This is what a Software Development Dashboard do. Software Development Dashboards are a set of graphs and data information about a project or set of projects, generally used as a project management tool. Thanks to this visualized information, different decisions about the project evolution can be taken, or the project development can change in order to improve the result, delivery times, or people on it. Figure 4 shows one personalized dashboard carried out with the final version of the current project.

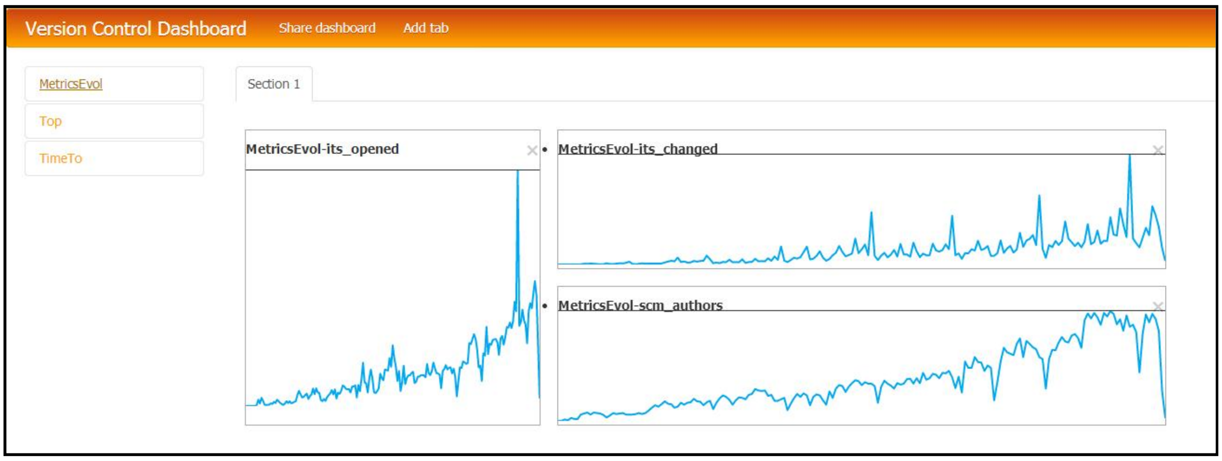


Figure 4: Software Development Dashboard created with Version Control Dashboard.

Depending of the Version Control system used for each project, it is possible that they provide different statistical data. Along the project, the JSON files that were used included different information, classified by source code, tickets, mailing, ... summarizing, statistical data.

Inside these classifications, there are plenty of different data, like sent mails, mail senders, source code committers, source code authors, source code branches, closed tickets, ticket changers, ticket openers, etc.

What normal Software Development Dashboards do is choosing some of these previous parameters and visualized them, statically. Version controls dashboard stores different JSON files with all the statistical data previously named. With this, any user can create a personal and completely personalized dashboard completely dynamic for his own use.

2.2.2. Software development dashboards related projects and companies

There exist some projects dedicated to create dashboards of other companies and their projects. Some of them are Gluster community[4], RDO project[5], Wikimedia[6] or OpenStack dashboard project[7]. Figure 5 shows and example of the info shown at index page of Open Stack dashboard project.

These companies take statistical data from other companies' FLOSS repositories, like IBM's, Bit Switch Networks', Cisco Systems', HP's, Huawei's, Microsoft's, between others, and visualize them. A FLOSS repository is the repository of a "free/libre/open-source software". Even though these companies can create their own way to visualize the data, there already exist some libraries and packages which analyze and visualize resulting data from a given HTML structure and data JSON files.

One of these libraries is vizGrimoire library, developed by Bitergia group. Bitergia[4] define himself as a company "specialized in getting, providing and analyzing Software Development metrics and information about FLOSS projects and communities for managing and improving them."

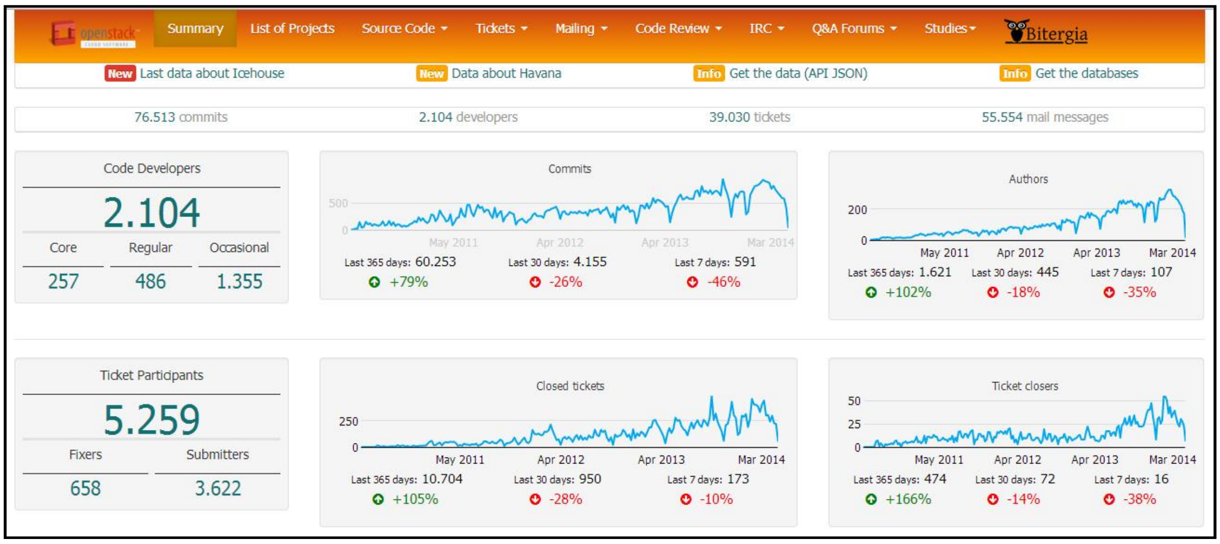


Figure 5: Dashboard example from OpenStack project.

[4] RedHat GlusterFS Report http://projects.bitergia.com/redhat-glusterfs-dashboard/ browser/
[5] RDO Report http://projects.bitergia.com/redhat-rdo-dashboard/browser/
[6] Wikimedia Community Metrics http://korma.wmflabs.org/browser/
[7] OpenStack Report http://activity.openstack.org/dash/browser/

In summary, Bitergia provides software development analytics services to FLOSS projects in order to improve the development process. One of these services is vizGrimoire project[5].

VizGrimoire is a open source toolset and framework created to analyze and visualize data about software development, focused on data produced by the MetricsGrimoire tools[6], a set of tools created also by Bitergia and related with data retrieving from Version Control repositories and storing in some specific JSON format.

VizGrimoire tools are provided for different programming languages. VizGrimoire JavaScript version[7] has been used in last steps of Version Control Dashboard project development.

However, Bitergia is not the only company that provides tools for version control dashboards. There exist other companies that work with different tools, as can be Oloh dashboards[8].

Oloh is a free, public directory of Free and Open Source Software. It also stores information about the developers who create and maintain these softwares. In the end, Oloh shows statistical widgets and graphs about the statistical information, like contributors per month or lines of code.

Another important example of dashboard can be seen in gitHub repository. GitHub provides a graph functionality for showing and visualizing the project information. Mainly, these graphs are classified as contributors, commits, code frequency or punch card graphs[9].

As user can see, it does not matter which dashboard he chose. All of them are completely static, and the option of dynamism, creation or edition are not available. Even though these dashboards provide very useful information, it could be improved if any customization for the user would be available.

[8] Oloh dashboards http://www.ohloh.net/
[9] GitHub dashboard https://help.github.com/articles/using-graphs

2.3. Web applications and working technologies

2.3.1. Web applications

A web application, or web app, is any software which runs in a browser or has been created using browser technology, like HTML, CSS and JavaScript. These applications can be created as client side or server side web applications. There are a few differences between these two types.

On one hand, client side web applications are directly created as HTML applications. HTML code, as well as JavaScript and CSS code, is hosted in a server, ready to be called from browsers through HTTP protocol. All application dynamism is directly executed with JavaScript in the browser.

On the other hand, server side web applications are not directly created using HTML technologies, but other programming languages, such as PHP or Python. In this kind of programming languages, HTML code is created in server side, and is returned to browser from special servers. The HTML page returned from server can be different for a same PHP resource. It could be said that, in server side web apps, dynamism is located in server side.

Any user can create his own web site using his own code. He can create his HTML architecture, as well as defining a CSS style for it and a script code for element interaction. However, there already exist plenty of libraries and frameworks which provide some functionalities to users and make development way easier.

It is not possible to talk about all different frameworks and JavaScript plugins, because there are plenty of them. Besides, that ones used along the Version Control Dashboard project are going to be explained, talking about how they work, what are they used for, and how they have been used along the project.

2.3.1.1. Bootstrap

Bootstrap[8] is an HTML5 and CSS3 framework designed to help with quick starting of web apps and pages. Bootstrap is open source and is hosted, developed and maintained in GitHub[9]. Even though current version is 3.1.1, for project development, it has been used is

Bootstrap 2.3.2[10]. Last version has not been used in order to keep compatibility with vizGrimoireJS package and Bootstrap version that is used in it.

Bootstrap provides different elements and JavaScript tools. Between others, buttons, transitions, modals, tabs, collapse, navs, pagination or progress bars elements are included into Bootstrap package. Besides, Bootstrap does not only include elements but also responsive grids and layouts, for web site design. All these tools and features can be found completely described in Bootstrap official web site.

As it will described later, Bootstrap has been very important in project development. Version Control Dashboard has a span2 - span10 responsive layout, where span2 is a collapse menu including the graph Version Control types and span10 is the area for visualizing the data, that includes navs, tabs, tooltip and modals. These last elements add human friendly experience to user.

2.3.1.2. JQuery

JQuery[11] is a fast, small and feature-rich JavaScript library that makes things like HTML document traversal and manipulation, event handling, animation and Ajax much simpler for web developer. It includes an easy-to-use API[12] that works across most of common current browsers. JQuery JavaScript library is written as open source code and it is hosted in a GitHub repository[13].

Besides of JQuery library, there exist plenty of plugins to add functionality to JQuery and web pages. Some examples are JQuery Validate, that adds form validation implementation, or JQueryUI, which adds some graphical elements, such as droppable or draggable elements, sliders, etc, between others. A list of main JQuery plugins can be found at plugin tab of JQuery web page[14].

2.3.1.3. Gridster

For Version Control Dashboard project development, JQuery has been used, as well as Gridster. Gridster[15] is an open source JQuery plugin created for building drag-and-drop multi-column grids, with different size cells.

What makes Gridster really interesting is the amount of actions and functionalities that can be applied over widgets. Gridster allows user the possibility of including or removing new widgets to the grid at any time. With Gridster, it is also possible to save grid status in an specified moment, to recover it later and modify.

Gridster includes other features such as custom drag handles, callbacks, or the option of expanding and resizing widgets.

With this, it has been seen that Gridster has a lot of features which make it a very good option for satisfying the needs of this project. Gridster is the option perfect for including all customization and dynamism that is required for the development of the project.

A developer can work with Gridster simply using *Gridster.min.js* or forking the gitHub repository.

2.3.1.4. Flotr2

Even though graphs and widgets will be visualized using vizGrimoireJS library, along the project will appeared a learning chapter about how to draw graphs in the web application. This learning step was realized with Flotr2[16], which is a library using for drawing HTML5 charts and graphs.

Flotr2 is written as open source code, hosted in GitHub[17]. Its features are mobile support, framework independency, custom charts and complete browsers compatibility with Firefox, Chrome, IE6+, Android and IOS.

The reason why Flotr2 has been chosen for learning in the project is because vizGrimoireJS has a visualization layer which uses this same library for drawing. In this way, working with Flotr2 before using vizGrimoireJS can help user save debugging times looking for solving error for hours.

Flotr2 is JavaScript library used for drawing charts. It is a very complete library which allows developers creating any type of graph and char, as can be line, range, column, pie or bubble charts, and include them all different options he desires. With Flotr2 allows the possibility of including charts listeners and events, like range zoom on x selector, pointer on hover or chart creating on mouse click.

Even though in this project chart visualization will be abstract thanks to vizGrimoireJS, it can be clearly seen that Flotr2 provides a very complete chart abstraction, and work with it is as simple as downloading minified file from its web page or forking the gitHub project from its repository.

2.3.2. Native mobile applications

What if any web application wants to be installed as a native mobile application instead of one stored in any web hosting? In that case, that web application should be developed and created as a native mobile application, written in the native programming language of each operative system.

This is a really hard task from any point of view, since it will be necessary a developer for each operative system. Even though most common mobile operative systems are android and IOS, there exist other platforms with a minor number of users, like Windows Phone 7, Windows Phone 8, Blackberry, Firefox OS, ... This means that a different code must be developed for each platform, that involves one managing and maintenance for each platform too.

Apart from that, there is another option for multiplatform development. There exist frameworks that allow developers to create several native mobile applications from a common HTML5 code. Some of them are Apache Cordova, Sencha Touch or Appcelerator Titanium. They are quite similar between each other.

The most commonly used is Apache Cordova[10] which allows creating native applications for many platforms, from Android and IOS to Bada, WebOS, Ubuntu or Tizen, going through Windows Phone or Firefox OS. Apache Cordova adds to the work environment the possibility of interacting with low lever operative systems thanks to plugins, created by other developers.

These plugins are commonly written as source code and shared through the network, which makes developers easier to create a native mobile application with no knowledge of native programming language.

These plugins are written to supply functionalities as Bluetooth communication, barcode scanner, geolocation, SDcard storage, or others.

[10] Apache Cordova website http://cordova.apache.org/

3. Development process

This chapter will explain the development process. As it was said before, a Scrum agile methodology has been user as development methodology. This chapter will describe the whole process, looking carefully every sprint, the technologies used and every difficulty or problem found.

3.1. First sprint. Creation of a dashboard with two graphs using Flotr2

First sprint goal is getting basic knowledge of how Flotr2 library works. The reason why this is done is because main widget type are graphs. Dashboard application code will be written from these widgets, and using them. Bigger the knowledge of Flotr2, bigger the power of the widgets.

The idea for this sprint is creating a web page able to read data from a JSON file and visualize it in two different graphs, with different options. The first one will draw data with a simple line graph, while the second one should show it with an area graph.

In the end of the sprint, the information included in the JSON file will have useful data, taken form a real repository. For this sprint, next tasks have been defined:

3.1.1. First task. Getting data from a JSON file

Graph data is stored in a JSON file, so the first task that must be done is read it. JQuery library provides a specific function for this purpose. This function is getJSON and it is used following *JQuery.getJSON(url[, data][, success(data, textStatus, jqXHR)])*, where *url* is the resource URL, *data* is a plain object or string that is sent to the server with the request and *success* is a callback function that is executed if the request success.

3.1.2. Second task. Visualizing random data array using in a single graph

First thing that must be done with all new technologies is start with some basic example, commonly named "Hello World" program. Talking about Flotr2, this "Hello World" program is just a simple graph, using some specific data array and some simple draw options.

To draw something using Flotr2, it is necessary to use *draw* function whose syntax is *Flotr.draw(container, data, options)*. Parameters have next meaning: *container* refers to the element where the info will be visualized and must have positive height and width, while *data* is the information to be visualized and *options* are the set of visualization options.

3.1.3. Third task. Visualizing random data from a JSON file in a single graph

This task combines first two task, and the objective of it is visualizing a single graph, but using this time data provided by a JSON file. In this case, the information must be read from the JSON file using getJSON JQuery method, store it in an array and then visualize it using *Flotr.draw* method, as was previously described.

3.1.4. Fourth task. Visualizing random data from a JSON file in two graphs

This task will take the code from the previous step and will modify it to include another container, for another graph. This new container must have a different id from the previous one, in order to differentiate where the information will be drawn. For drawing all charts, *draw* method must be called one time for each container, using its own id and their own options.

3.1.5. Fifth task. Visualizing repository data JSON file in two graphs

This last task modifies code in order to adapt it to new JSON file structure. Final dashboard for this sprint will show commits info about two companies, HP and IBM, so the program reads these keys from JSON file and store them in different arrays, each of them used in different *Flotr.draw* calls.

Conclusions and results of the sprint

Final design of the sprint prototype looks like Figure 6. After completing this sprint, Flotr2 has proved that is a useful library for drawing charts, really complete, easy to use and with a lot options for different chart types.

In the final sprint prototypes, Flotr2 will not be used directly, because graphs will be drawn using vizGrimoireJS library, which uses Flotr2 in lower levels. Despite this, this sprint is really important because it provides knowledge about how Flotr2 works, and its visibility feature. Along future sprints, it will be necessary to draw graphs in different not visible tabs, which will report errors. Experience acquired along this sprint will help facing this troubles.

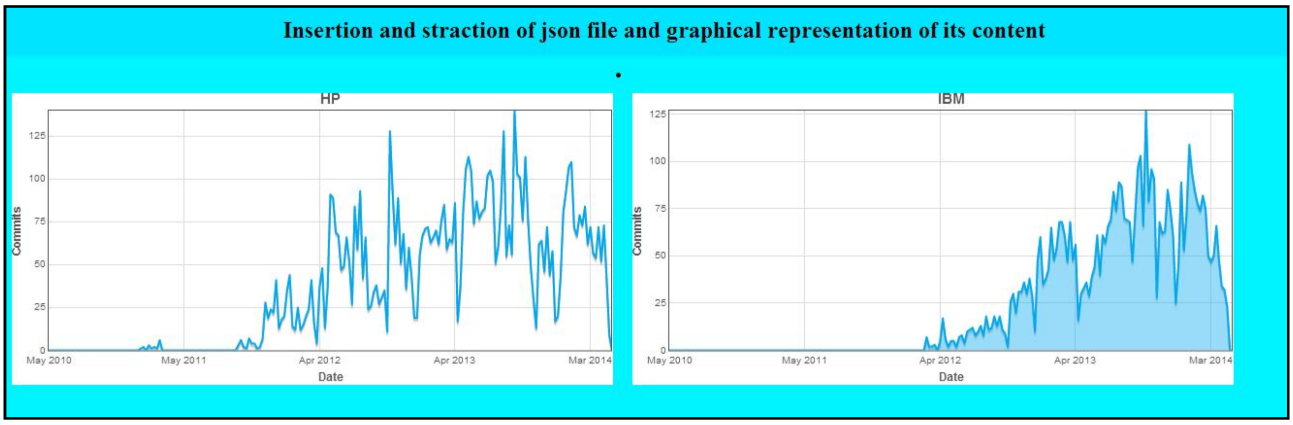

Insertion and straction of json file and graphical representation of its content
HP
IBM
Commits
Date
May 2010
May 2011
Apr 2012
Apr 2013
Mar 2014
0
25
50
75
100
125

Figure 6: Dashboard with two graphs created using Flotr2.

3.2. Second sprint. Creation of a dashboard with draggable graphs

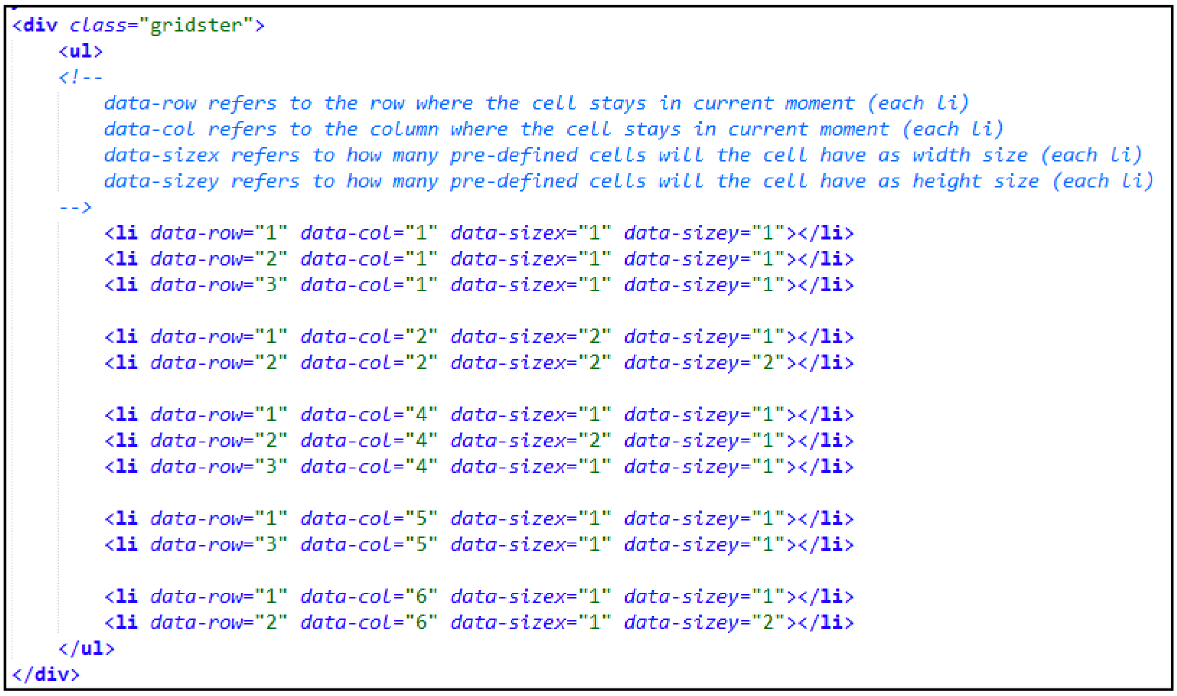
At the end of this sprint it was decided to have a dashboard prototype which would include both Flotr2 and Gridster, with a more complex style. The dashboard of the previous sprint was quite simple, and just had some simple CSS style file. For this sprint, style is taken from JQuery Mobile libraries. The reason why JQuery Mobile is used now as style is because the idea, at this point of the project, is to add new functionalities, like sharing and tabs, following JQuery Mobile schema and elements.

It was necessary to get some knowledge about Gridster and how it works, JQuery Mobile schema and how these two new components could be integrated together. These are the tasks that compose the second sprint:

3.2.1. First task. Creation of a draggable grid with Gridster

This is the main example to start working with Gridster, its "Hello world" program. In this case, the grid and its cells will be created directly in HTML file, but it can be directly injected from JavaScript if the widget must be dynamically added.

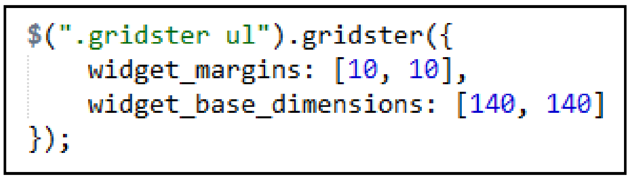
To create a basic grid, two main thing must be known. Firstly HTML code, which can be seen in Figure 7, describes the grid structure. Each *li* element contains the necessary attributes for defining his position in the grid and his size. All these attributes are mandatory.



```
<div class="gridster">
    <ul>
    <!--
        data-row refers to the row where the cell stays in current moment (each li)
        data-col refers to the column where the cell stays in current moment (each li)
        data-sizex refers to how many pre-defined cells will the cell have as width size (each li)
        data-sizey refers to how many pre-defined cells will the cell have as height size (each li)
    -->
        <li data-row="1" data-col="1" data-sizex="1" data-sizey="1"></li>
        <li data-row="2" data-col="1" data-sizex="1" data-sizey="1"></li>
        <li data-row="3" data-col="1" data-sizex="1" data-sizey="1"></li>

        <li data-row="1" data-col="2" data-sizex="2" data-sizey="1"></li>
        <li data-row="2" data-col="2" data-sizex="2" data-sizey="2"></li>

        <li data-row="1" data-col="4" data-sizex="1" data-sizey="1"></li>
        <li data-row="2" data-col="4" data-sizex="2" data-sizey="1"></li>
        <li data-row="3" data-col="4" data-sizex="1" data-sizey="1"></li>

        <li data-row="1" data-col="5" data-sizex="1" data-sizey="1"></li>
        <li data-row="3" data-col="5" data-sizex="1" data-sizey="1"></li>

        <li data-row="1" data-col="6" data-sizex="1" data-sizey="1"></li>
        <li data-row="2" data-col="6" data-sizex="1" data-sizey="2"></li>
    </ul>
</div>
```

Figure 7: HTML Gridster structure.

On the other hand, Figure 8 shows JavaScript code for initializing Gridster. In this example, Gridster constructor does only include widgets dimensions, but it can contain other features as draggable handlers or specify return parameters when an event is triggered over some widget. This will appear again in some future sprints. Figure 9 shows Grister div created by using the code specified in both Figure 7 and 8, including some style not specified here. It can be seen that each cell has same features that defined in previous Figure.



```
$(".gridster ul").gridster({
    widget_margins: [10, 10],
    widget_base_dimensions: [140, 140]
});
```

Figure 8: JavaScript Gridster launcher.

3.2.2. Second task. Creation of a basic page following JQuery Mobile schema

JQuery Mobile defines a schema to create HTML web pages. In them, there are three main roles, that are *header*, *content* and *footer*, all of them inside a bigger div which role is the *page*. The idea is creating a web page in this task, to get knowledge with JQuery Mobile and all their elements. JQuery elements will be used in later tasks, such as modals.

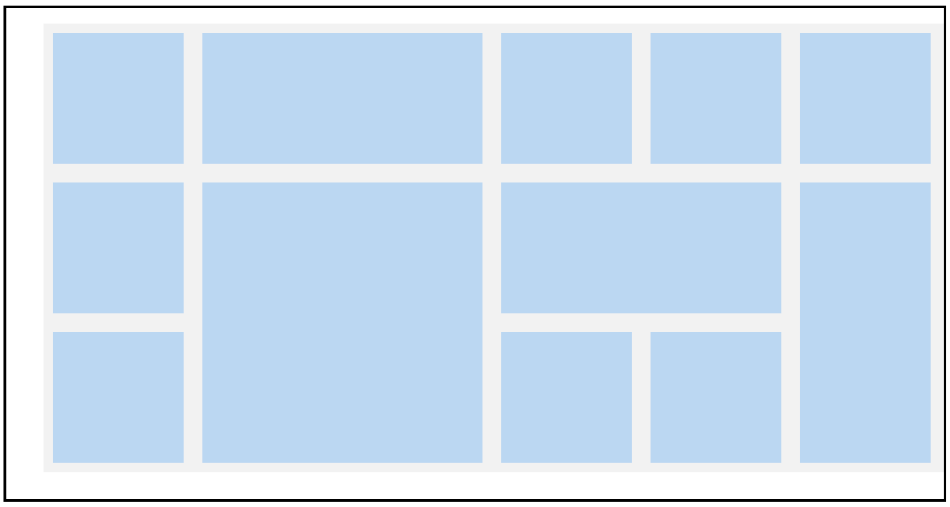
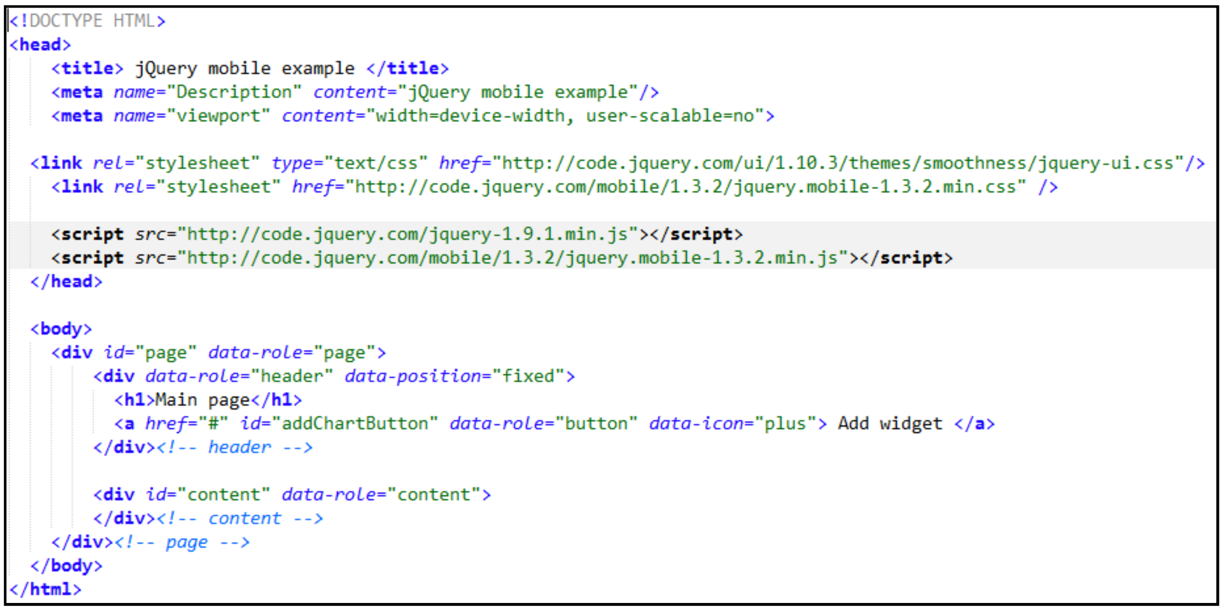


Figure 9: Grister cells distribution.

JQuery page created in this task is done to be combined with a Gridster in next task. JQuery page will have a header with one button to include a new widget, an empty content and footer divs. Figure 10 shows JQuery Mobile HTML code for the basic schema of next task.



```
<!DOCTYPE HTML>
<head>
    <title> jQuery mobile example </title>
    <meta name="Description" content="jQuery mobile example"/>
    <meta name="viewport" content="width=device-width, user-scalable=no">

  <link rel="stylesheet" type="text/css" href="http://code.jquery.com/ui/1.10.3/themes/smoothness/jquery-ui.css"/>
    <link rel="stylesheet" href="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.css" />

    <script src="http://code.jquery.com/jquery-1.9.1.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.3.2/jquery.mobile-1.3.2.min.js"></script>
  </head>

  <body>
    <div id="page" data-role="page">
        <div data-role="header" data-position="fixed">
          <h1>Main page</h1>
          <a href="#" id="addChartButton" data-role="button" data-icon="plus"> Add widget </a>
        </div><!-- header -->

        <div id="content" data-role="content">
        </div><!-- content -->
    </div><!-- page -->
  </body>
</html>
```

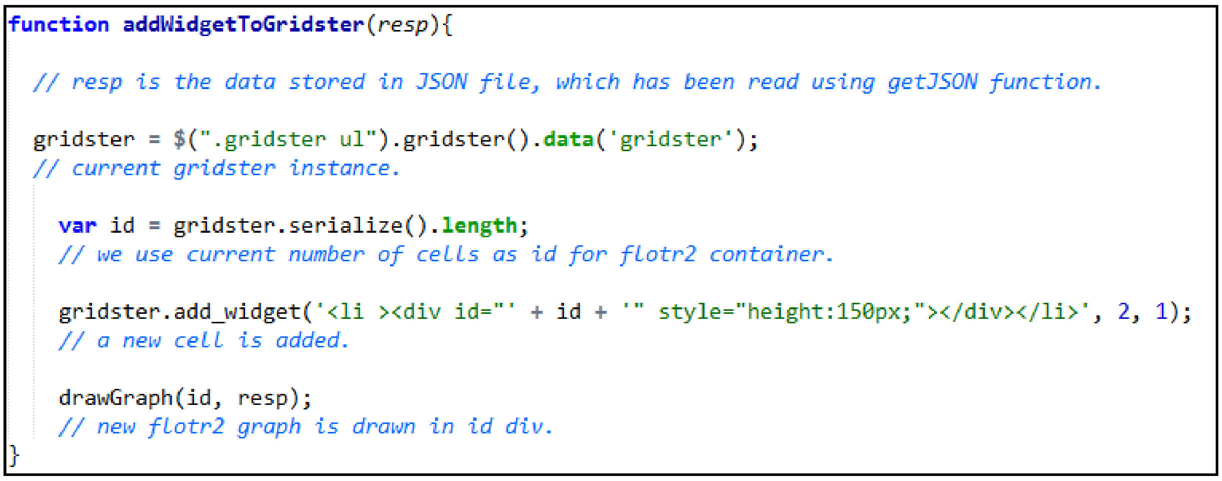
Figure 10: JQuery Mobile schema.

3.2.3. Third task. Creation of a JQuery Mobile style dashboard

This task will complete the sprint combining all technologies shown until the moment. The ending prototype dashboard will have a JQuery Mobile style with a Gridster div inside content role.

In the beginning, this dashboard will have only one button. After clicking it, a new widget is added to current Gridster and visualize a chart inside. This chart will be drawn using Flotr2 and its data will proceed from a single JSON file. In this example, the visualized data will always be the same, so all widgets will contain the same graph. This will change in future sprints.

This task will go on from the previous task code. First thing that must be done is including all necessary libraries and after it, Gridster div must be added to *content* role of JQuery Mobile schema. However, in this case the Gridster is composed by an empty div, because the grid will be created dynamically when a new widgets is created. Code that creates grid and adds widgets can be seen in next Figure 11.



```
function addWidgetToGridster(resp){

  // resp is the data stored in JSON file, which has been read using getJSON function.

  gridster = $(".gridster ul").gridster().data('gridster');
  // current gridster instance.

    var id = gridster.serialize().length;
    // we use current number of cells as id for flotr2 container.

    gridster.add_widget('<li ><div id="' + id + '" style="height:150px;"></div></li>', 2, 1);
    // a new cell is added.

    drawGraph(id, resp);
    // new flotr2 graph is drawn in id div.
}
```

Figure 11: Code for inserting a new widget into the Gridster.

This function is executed when an user clicks new chart button. Here the main line is the *Gridster.add_widget* function. This function inserts HTML as a new cell, and formats it with dimensions specified in second and third parameters. These parameters express 2 and 1 width and height size units, respectively. A unit has size specified in Gridster JavaScript contructor code. Main behaviour of the final prototype of this sprint can be seen in figures 12 and 13.

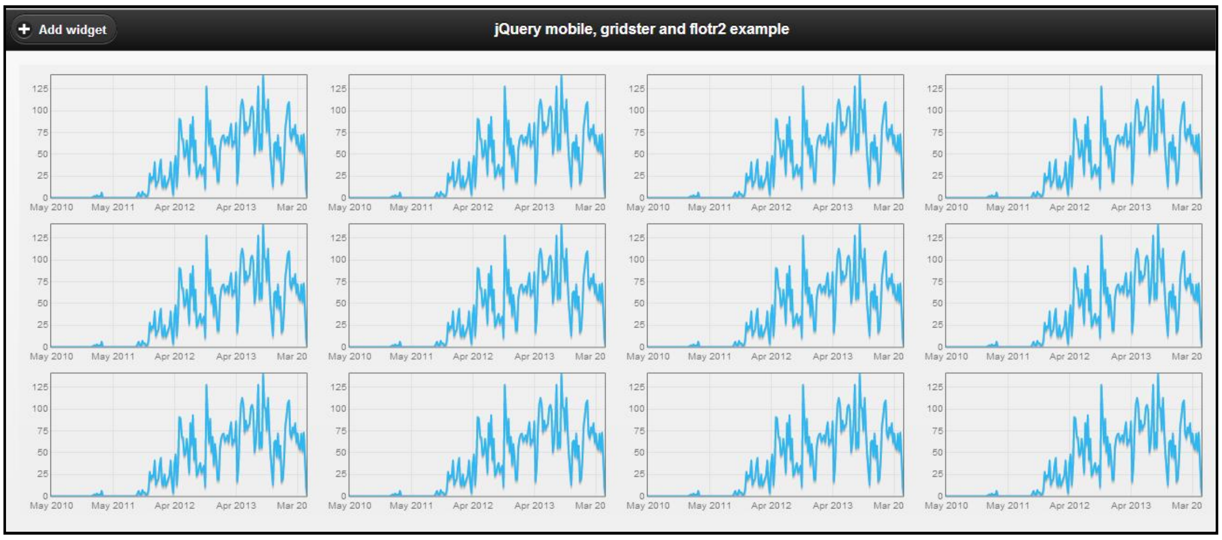


Figure 12: Dashboard view when "Add widget" button has been pressed 12 times.

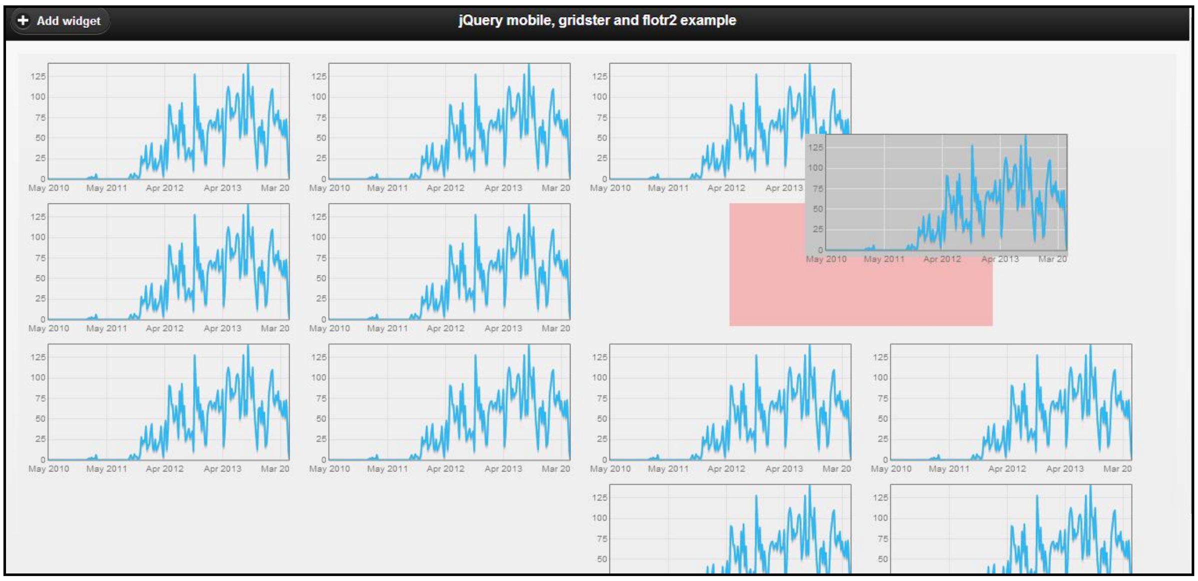


Figure 13: Dashboard view when a widget is being dragged.

<u>Conclusions and results of the sprint</u>

Previous is the final dashboard prototype of this sprint. It mainly includes the draggable Gridster, that adds dynamism to dashboard and is easy to use. There were no main problems along this sprint, because it was mainly a first contact with Gridster.

3.3. Third sprint. Tabs and multi JSON data source dashboard integration

This sprint means to make the dashboard a bit more real. For it, final dashboard prototype of this sprint has to include tabs for becoming the interface a bit more friendly for users. Furthermore, data will proceed from more than one JSON file, and there will be a JQueryMobile panel in order to allow user to chose the source and data which wants to add to the dashboard.

To finish correctly this sprint, it is necessary to understand how panels and tabs work. After it, they can be integrated in the prototype of the previous sprint.

3.3.1. First task. Using JQuery UI for insert tabs in a web application

There exist some different libraries that provide tab integration. JQuery UI has been chosen to be combined with JQuery Mobile for this sprint. This task just creates a static application with a fix number of tab regions all of them empty.

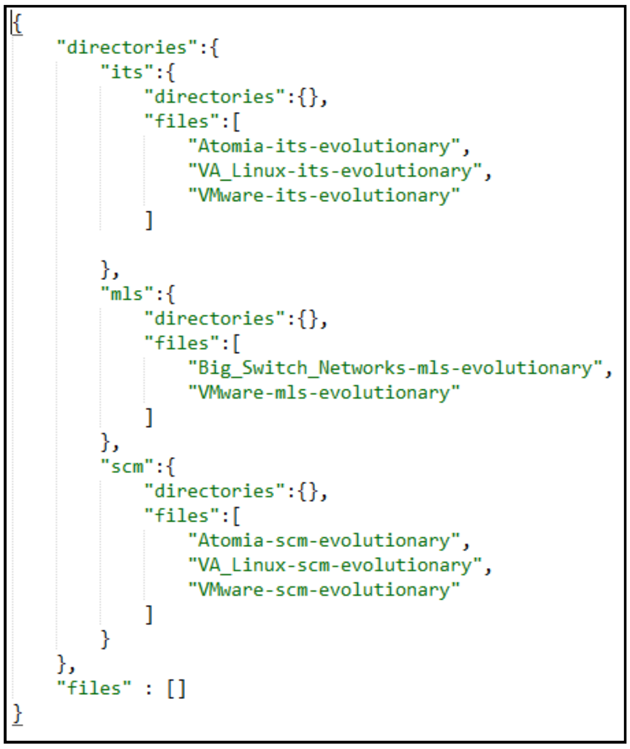
JQuery UI tabs[11] work as follows. There exist two main elements in the HTML code inside the tab div: tabs and regions. Regions are normally hidden, and only one is shown at the same time. When some tab is clicked, current visible region is hidden, and then the region which is pressed by the clicked tab is shown. All this functionality is automatically done just using JavaScript constructor code for tabs *$('#tabs').tab()*

3.3.2. Second task. Designing JSON file structure and data read

All the data is now stored in different JSON files. These JSON files are distributed along a directory structure which must be said to application in order to read correctly all data and visualize it. To make it compatible with all browsers, this directory structure will be stored in a specified file. JSON structure file looks like figure 14.

[11] JQuery UI tab page http://JQueryui.com/tabs/

As reader can see, this JSON file has a structure which can be easily travelled with some recursive algorithm.



```
{
    "directories":{
        "its":{
            "directories":{},
            "files":[
                "Atomia-its-evolutionary",
                "VA_Linux-its-evolutionary",
                "VMware-its-evolutionary"
            ]

        },
        "mls":{
            "directories":{},
            "files":[
                "Big_Switch_Networks-mls-evolutionary",
                "VMware-mls-evolutionary"
            ]
        },
        "scm":{
            "directories":{},
            "files":[
                "Atomia-scm-evolutionary",
                "VA_Linux-scm-evolutionary",
                "VMware-scm-evolutionary"
            ]
        }
    },
    "files" : []
}
```

In this way, user can navigate through directories until he chose a file. When a file is chosen, then the different keys that are content inside can be shown, so the user can chose that parameter want to show.

Figure 14: JSON file structure file.

3.3.3. Third task. Using JQuery Mobile for showing side panel

There exist different tools that can provide navigation functionality, like panels, collapse, trees or others. In this sprint of the project, panel were chosen as element for navigation, specifically JQuery Mobile panels[12].

In this task, the main objective is the creation of a simple JQuery Mobile web with a button that, when is clicked, makes a panel appear. JQuery Mobile panels are really easy to use. When some panel is wanted to be shown, developer just has to call *$('#panel').panel('open')* method. Else, if the developer wants to hide the panel, he has to call *$('#panel').panel('close')* method. There is one important thing to say about panels and JQuery Mobile basic schema. The panel div must be placed either before or after *header*, *content* and *footer* elements, but not in between them. Else, the panel will not work correctly.

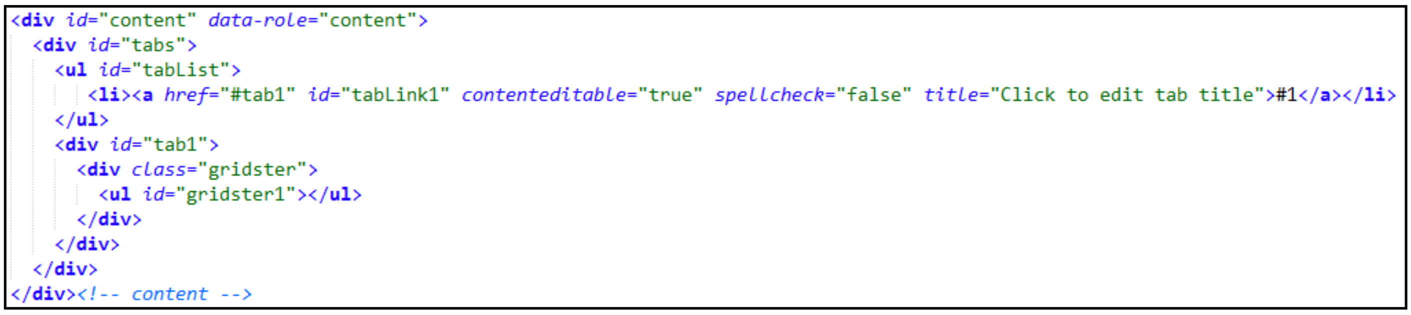
3.3.4. Fourth task. Integrating tabs, panels and JSON file structure

This task works on integrating all previous technology to get a more complex dashboard, with tabs and a side panel to navigate between the file structure. In the final prototype of this

[12] JQuery Mobile panels page http://demos.JQuerymobile.com/1.3.2/widgets/panels/

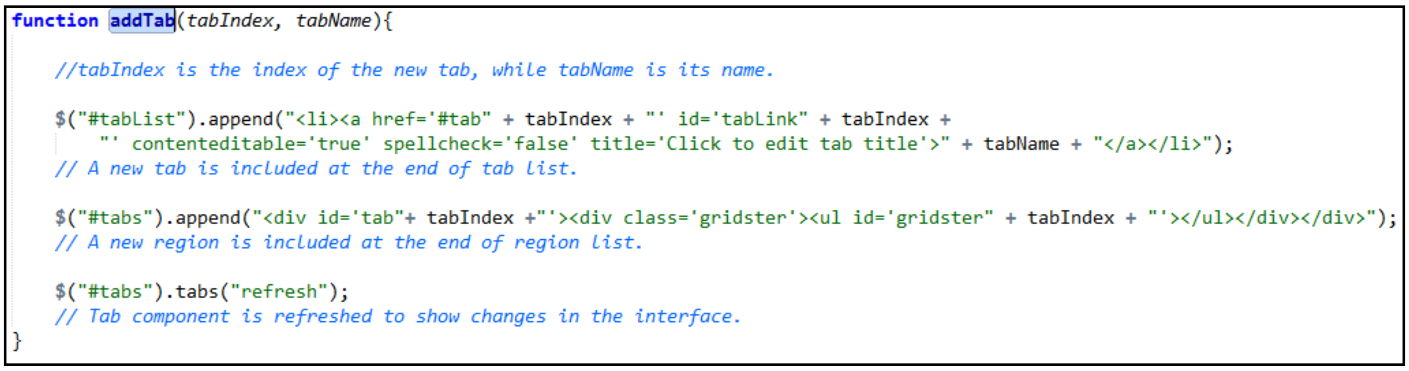
sprint, user will be able to chose between files and the data which content, and visualize them in the dashboard as graph widgets.

First thing that must be done is including the tab structure inside the *content* role. Each tab region has a different Gridster, which must be properly initialized. *Content* role can be seen in figure 15, and the code which modifies it, adding new tabs and Gridsters is shown in figure 16. The JavaScript code appends new elements to tab list and region list.



```html
<div id="content" data-role="content">
  <div id="tabs">
    <ul id="tabList">
       <li><a href="#tab1" id="tabLink1" contenteditable="true" spellcheck="false" title="Click to edit tab title">#1</a></li>
    </ul>
    <div id="tab1">
      <div class="gridster">
        <ul id="gridster1"></ul>
      </div>
    </div>
  </div>
</div><!-- content -->
```

Figure 15:*Content* role including tab schema.



```javascript
function addTab(tabIndex, tabName){

    //tabIndex is the index of the new tab, while tabName is its name.

    $("#tabList").append("<li><a href='#tab" + tabIndex + "' id='tabLink" + tabIndex +
        "' contenteditable='true' spellcheck='false' title='Click to edit tab title'>" + tabName + "</a></li>");
    // A new tab is included at the end of tab list.

    $("#tabs").append("<div id='tab"+ tabIndex +"'><div class='gridster'><ul id='gridster" + tabIndex + "'></ul></div></div>");
    // A new region is included at the end of region list.

    $("#tabs").tabs("refresh");
    // Tab component is refreshed to show changes in the interface.
}
```

Figure 16: JavaScript code executed to add new tab.

Figures 17 and 18 shows tab working. Figure 17 shows dashboard when "Add tab" button has been clicked thrice, and Figure 18 shows dashboard when tab #3 has been selected.

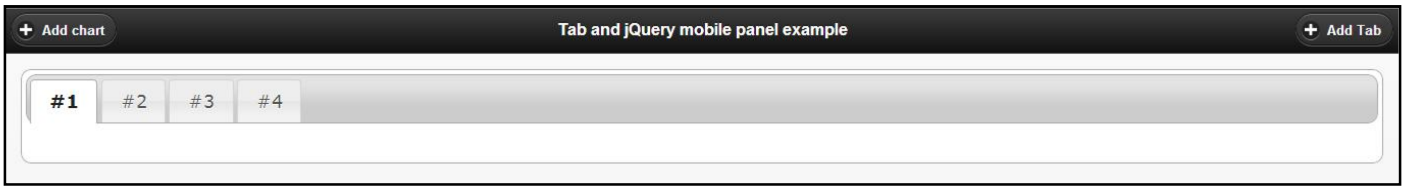


Figure 17: Dashboard view when "Add tab" button has been pressed thrice.

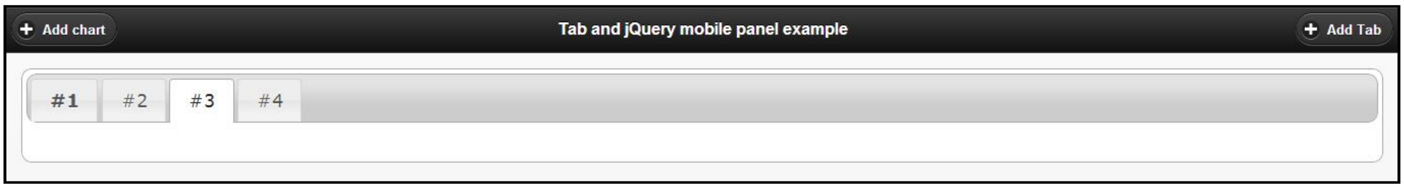


Figure 18: Dashboard view when tab #3 is current selected.

Once the tab element has been added, it is time for adding navigation panel. Navigation panel is shown when "Add chart" button is clicked. In this part, recursive algorithm is used, mixed with JQuery Mobile panels. Recursive algorithm is divided in two sub-algorithms. The first one is used for navigating though directories, while the second one is used for reading JSON keys and adding the chart to dashboard, into the current selected tab. Figures 19 shows the navigation process and chart addition to first tab. Figure 20 shows how the new graph is inserted into the selected tab of the dashboard.

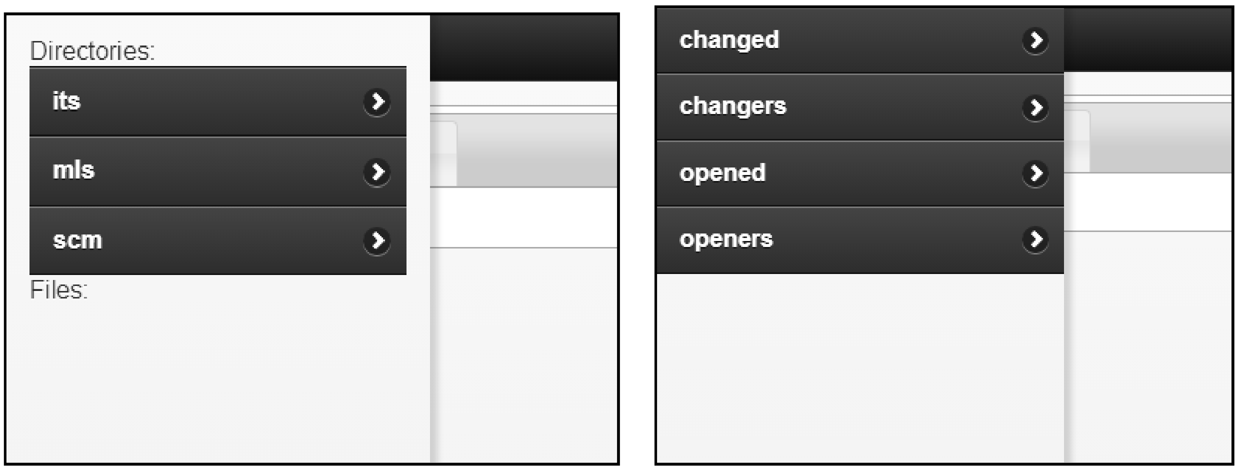


Figure 19: Example of panel navigation at directory and file level, respectively.

<u>Conclusions and results of the sprint</u>

With this final prototype, it can be seen that the dashboard is becoming really close of what the project goal: a web application able to allow user creating his own dashboard, with different tabs and content in each of them.

There is one bad point about current prototype, and it is the style. Current prototype does not looks like a professional web application, specially because of the side navigation panel.

This is not a big problem because, as it was explained previously, interface style will change and JQuery Mobile will not be used anymore from a future sprint.

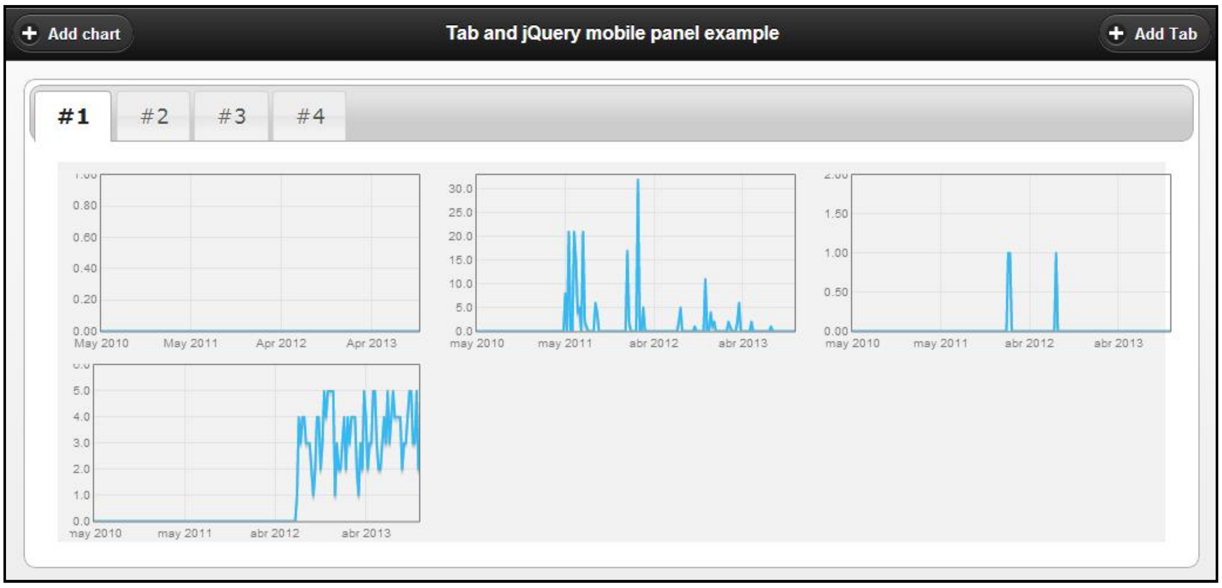


Figure 20: Some different data parameters have been inserted in first tab.

Bootstrap will be the new style framework that will be used. The reason why JQuery Mobile is being used along these first sprints is because it provides some tools easy to use that allow developer creating a simple web application without worrying a lot about the interface. In that way, he can start working in other parts of the application that can need more attention. In this sprint, main example would be the algorithm created for travelling along the directories and files, as well as the JSON files visualization.

Once completed this sprint, it is time to start working in new Bootstrap style and including vizGrimoireJS library. However, before this, what about some social functionality? What about sharing?

3.4. Fourth sprint. Creation of a dashboard that can be shared

This sprint is destined to include sharing functionality into the prototype of the previous sprint. This functionality can be implemented using different several ways, but in this project hash methodology has been used.

Hash methodology works as follows: dashboard status is stored as a JSON when user presses "share" button. In that moment, application generates a new URL which user can share. This new URL contains a hash code which refers to the previously stored JSON. When a shared URL is called from a browsers, application reads hash code and paints the dashboard in the status of the respective dashboard.

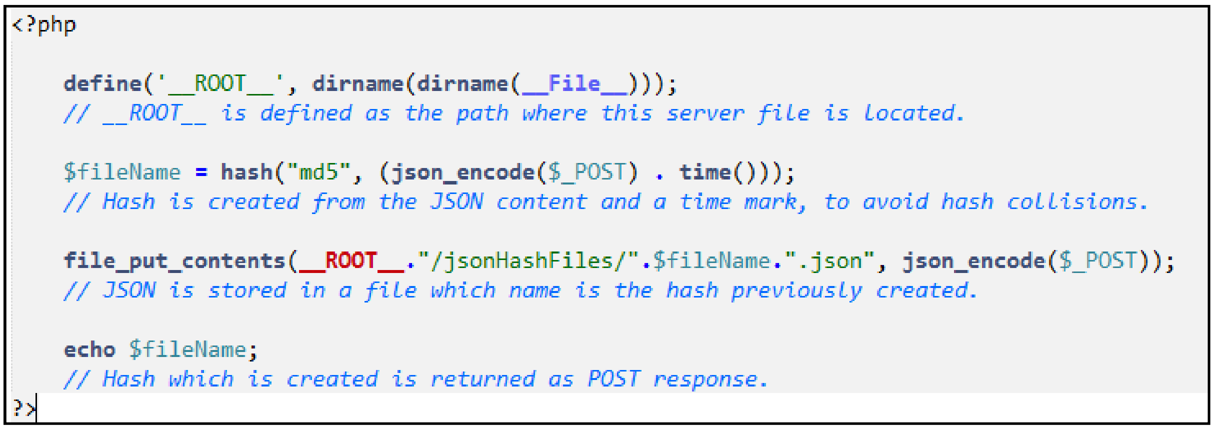
From all this, it is obvious that a server part is necessary to implement completely the functionality. Along this sprint both, server and client sides actions of the application will be explained carefully, and they will communicate each other using REST JSON protocol. Main two tasks refers to the client code for sharing and the server implementation.

3.4.1. First task. Creating a server using Xampp

Xampp[13] is a PHP development environment which includes technologies enough to create a server in a few minutes. This is one of the reasons because Xampp has been chosen as server. There exist other environments, like Django or node.js, but PHP syntax was more familiar to developer and any other environment would have taken more time that using Xampp and PHP.

Actually, what is needed for this task of the sprint, is a PHP server, so Xampp would have not been necessary. Installing an Apache server and PHP would have fixed server needs, but Xampp abstract developer of multi package installation, including all of them together.

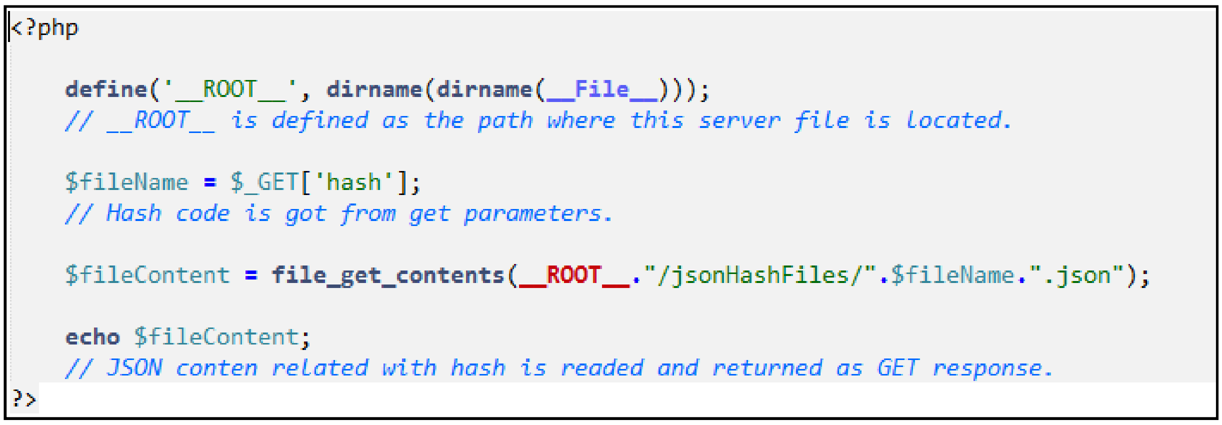
PHP server is quite simple. It is composed by two methods, one which stores (POST) a JSON structures and returns a hash code and other which, given a hash code (GET), returns his respective JSON structure.



```php
<?php

    define('__ROOT__', dirname(dirname(__File__)));
    // __ROOT__ is defined as the path where this server file is located.

    $fileName = hash("md5", (json_encode($_POST) . time()));
    // Hash is created from the JSON content and a time mark, to avoid hash collisions.

    file_put_contents(__ROOT__."/jsonHashFiles/".$fileName.".json", json_encode($_POST));
    // JSON is stored in a file which name is the hash previously created.

    echo $fileName;
    // Hash which is created is returned as POST response.
?>
```

Figure 21: PHP server code to get hash code from JSON dashboard structure.

[13] Xampp apache friends page https://www.apachefriends.org/index.html



```
<?php

    define('__ROOT__', dirname(dirname(__File__)));
    // __ROOT__ is defined as the path where this server file is located.

    $fileName = $_GET['hash'];
    // Hash code is got from get parameters.

    $fileContent = file_get_contents(__ROOT__."/jsonHashFiles/".$fileName.".json");

    echo $fileContent;
    // JSON conten related with hash is readed and returned as GET response.
?>
```

Figure 22: PHP server code to get JSON dashboard structure from hash code.

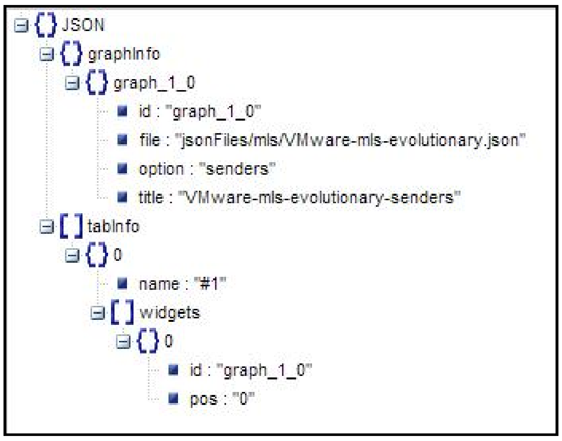
The JSON is stored in a file and read from it. Figures 21 and 22 shows PHP code which carry out POST and GET calls, respectively.

3.4.2. Second task. Implementing share functionality in client side

A little part of sharing action is implemented in the client side of the application. To add share functionality in the client, first thing which must be done is deciding JSON structure which will be stored in the server.

From Figure 20, user can deduce that the dashboard can be expressed as a list of graphs, distributed along a list of tabs. Each graph has different information, as title (this is not shown in Figure 20, but it will be added in future prototypes), or data which is shown. Both list of tabs and list of widgets can be related each other like is shown in Figure 23.

In this Figure, it can be seen that JSON includes both tab list and widgets list. Each graph has a unique id and it´s referred from the tab where is located.



```
{} JSON
  {} graphInfo
    {} graph_1_0
        id : "graph_1_0"
        file : "jsonFiles/mls/VMware-mls-evolutionary.json"
        option : "senders"
        title : "VMware-mls-evolutionary-senders"
  [] tabInfo
    {} 0
        name : "#1"
      [] widgets
        {} 0
            id : "graph_1_0"
            pos : "0"
```

Figure 23: JSON example which will be stored as dashboard structure.

Therefore, client functionality must store graphs globally in the application when they are added to dashboard. Tab list structure is created when "share" button is clicked, storing in each tab a reference to all widgets which contains.

With this, the problem of associating widgets with tabs is solved but ¿what about the data? If there were 30 tabs and each tab had 30 widgets, how would it be stored? It´s obvious that storing the raw data is not a good idea, because it could create a size problem in the disk space of the server.

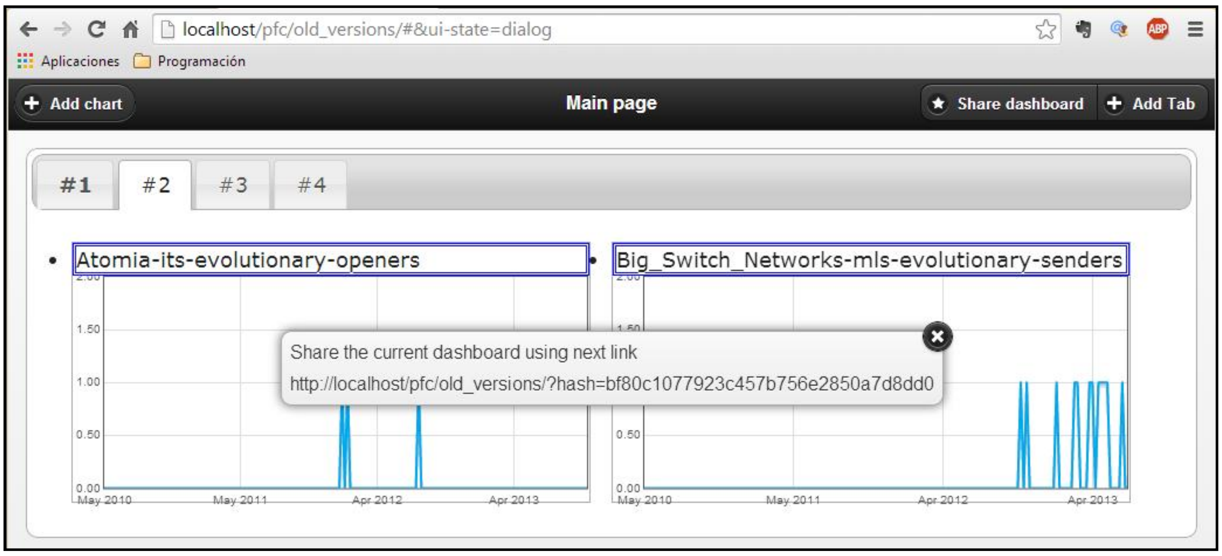
The best solution for this problem is associating each widget with the source of the data that is showing. This means, storing in each element of the graph structure the file and option that were chosen by the user to be visualized.

Once the user presses "share" button, JSON structure is created and POST request is done to PHP server that returns hash code created from the sent JSON structure. After receiving hash, a pop up with a URL is shown to the user.

When a URL is shared and web application is called with a hash in the URL, application executes a GET call requesting for this hash and obtaining a JSON data. The client parses this JSON and creates automatically the tabs and widgets specified in the JSON data.

<u>Conclusions and results of the sprint</u>

Next Figures 24 and 25 show dashboard message when "share" button has been pressed and shared link is visualized, respectively.


localhost/pfc/old_versions/#&ui-state=dialog
Aplicaciones
Programación
Add chart
Main page
Share dashboard
Add Tab
#1
#2
#3
#4
Atomia-its-evolutionary-openers
Big_Switch_Networks-mls-evolutionary-senders
May 2010
May 2011
Apr 2012
Apr 2013

Share the current dashboard using next link
http://localhost/pfc/old_versions/?hash=bf80c1077923c457b756e2850a7d8dd0

Figure 24: Share dashboard message.

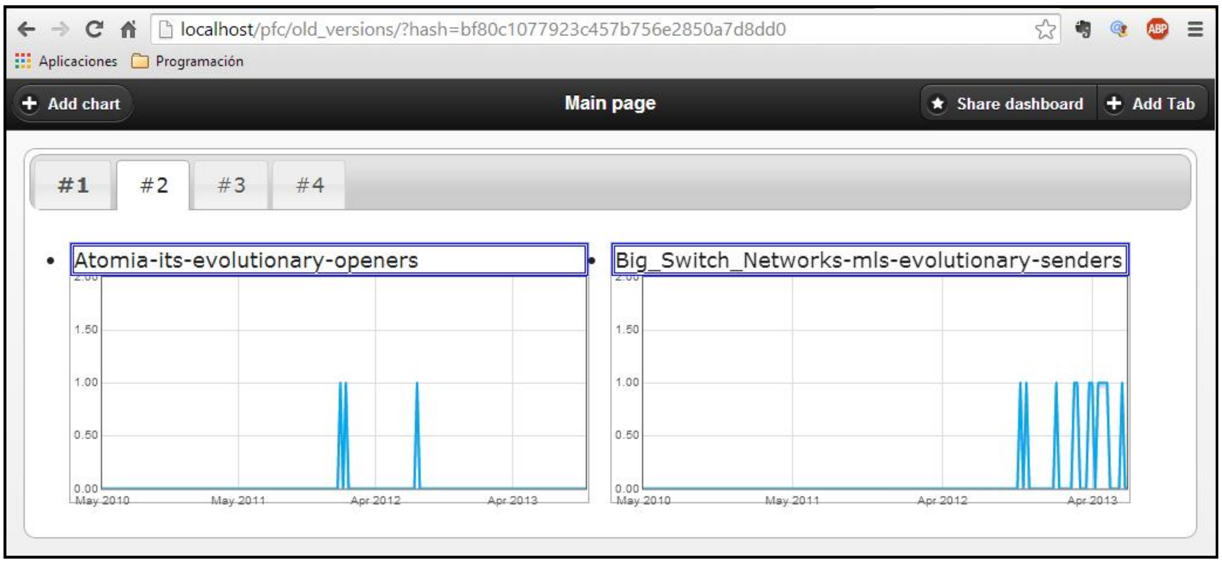

localhost/pfc/old_versions/?hash=bf80c1077923c457b756e2850a7d8dd0
Aplicaciones
Programación
Add chart
Main page
Share dashboard
Add Tab
#1
#2
#3
#4
Atomia-its-evolutionary-openers
Big_Switch_Networks-mls-evolutionary-senders
1.50
1.00
0.50
0.00
May 2010
May 2011
Apr 2012
Apr 2013

Figure 25: Share dashboard message.

As can be seen, the result obtained after using share link is the same that the original dashboard from Figure 24. The algorithm and structure designed for sharing work quite well, but it can show some problems. User can see that the figure 25 shows a hash code into the URL, which determines that it is the shared link.

As it was said before, the widgets structure is stored as the file and options that represents each graph instead of the raw data. To rebuild full dashboard is necessary to call again all JSON files and visualize all data. If there were a lot of graphs or the connection is not fast enough, user would face speed problems and waits. Anyway, this solution is better than the solution of sending all graphs data in the POST call.

Other problem which appeared during the development was related with Flotr2 visibility feature. It was really important to draw a graph in his related tab only if this is the current selected. Else, a JavaScript exception was received. It was really difficult to detect this error.

Furthermore of sharing functionality, in this sprint, some details have been applied, like graph style and graph title. Each graph title is created from the file and parameter related with its graph.

3.5. Fifth sprint. VizGrimoireJS integration and Bootstrap migration

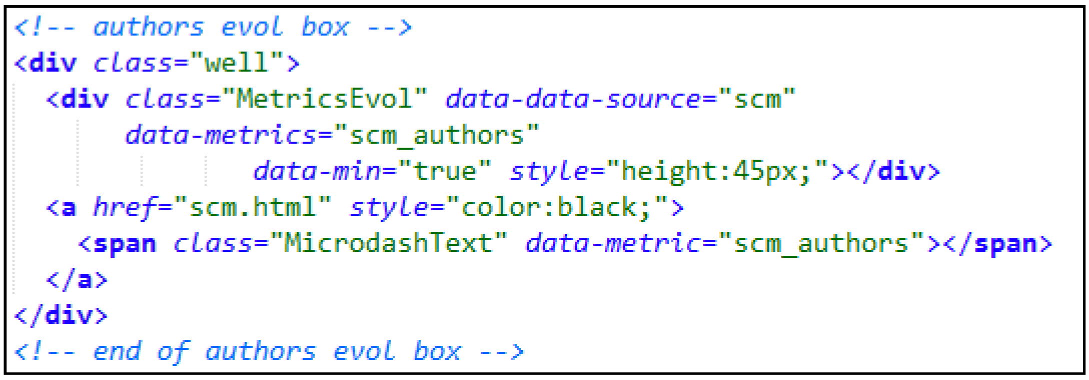
In this moment of the project, sharing is completely implemented and it is time to start migrating JQuery Mobile environment to Bootstrap framework. In this sprint, Flotr2 graphs will be automatically created through vizGrimoireJS library.

This sprint is not destined to add any other action into the web application, but to change style and tools which have been used. There will be some changes on code or tools previously announced, but it will try to use mostly possible. This is a long sprint with a lot of little tasks which must be combined to get the almost final prototype, the final application.

3.5.1. First task. Starting using vizGrimoireJS

It is time for including vizGrimoireJS into the project. VizGroimoire is going to replace all widgets creation to do it automatically from a simple structure formed by tags and attributes. Previous tasks about widget visualization were very important to face vizGrimoire integration, because vizGrimoire uses Flotr2 in deeper levels and some problems can take place. These problems are quite similar to problems faced when using Flotr2 library, during the first sprints.

VizGrimoireJS works as follows. It reads HTML file looking for a special structure similar to shown in next figure 26, which defines data, and attributes about the widgets. VizGrimoire is able to visualize different kind of widgets, from graphs to footers, headers or information widgets, all of them created from their div attributes.



```
<!-- authors evol box -->
<div class="well">
  <div class="MetricsEvol" data-data-source="scm"
       data-metrics="scm_authors"
               data-min="true" style="height:45px;"></div>
  <a href="scm.html" style="color:black;">
    <span class="MicrodashText" data-metric="scm_authors"></span>
  </a>
</div>
<!-- end of authors evol box -->
```

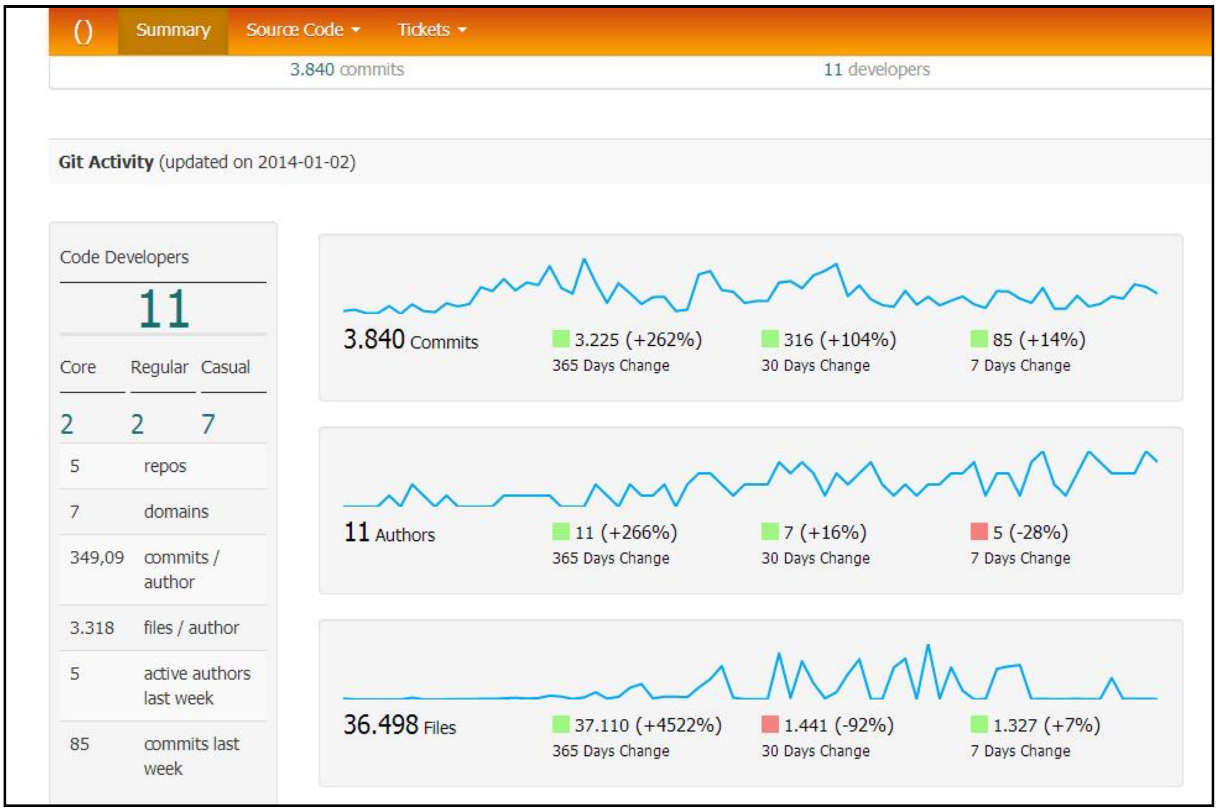
Figure 26: HTML structure of a vizGrimoire widget.

From previous figure, it can be seen that there exist a big div, that is a widget, defined by *well* class. Inside this *well*, there is another tab which is a *MetricsEvol* class. This class is what really matters from previous figure. This class is defined by vizGrimoireJS library, as well as the *data*-attributes contained in the divs.

This class belong to a set of classes which specify some predefined widgets. For example, *MetricsEvol* specify a time evolution chart, while *FilterItemsSummary* just show the result of a filter. VizGrimoire provides plenty more of widgets, that can be seen in doc folder inside github project[7].

When the vizGrimoireJS library is included in the HTML page, it is responsible for reading file looking for these classes and visualizing its related widget.

The objective of this task is creating a simple dashboard, completely static, with predefined graphs using vizGrimoire library. A dashboard example was created modifying other developers´ vizGrimoire examples, which can be seen in next figure 27. Even though some other elements can be seen in this figure, user can see also the widgets, which have ben created from similar divs to figure 26.



3.840 commits 11 developers

Git Activity (updated on 2014-01-02)

Code Developers

11

Core	Regular	Casual
2	2	7

5	repos
7	domains
349,09	commits / author
3.318	files / author
5	active authors last week
85	commits last week

3.840 Commits	🟩 3.225 (+262%) 365 Days Change	🟩 316 (+104%) 30 Days Change	🟩 85 (+14%) 7 Days Change
11 Authors	🟩 11 (+266%) 365 Days Change	🟩 7 (+16%) 30 Days Change	🟥 5 (-28%) 7 Days Change
36.498 Files	🟩 37.110 (+4522%) 365 Days Change	🟥 1.441 (-92%) 30 Days Change	🟩 1.327 (+7%) 7 Days Change

Figure 27:VizGrimoire static dashboard example.

3.5.2. Second task. Bootstrap migration

Bootstrap, like JQuery Mobile, has some predefined tags and attributes ready for a fast web development. The main purpose of this task is creating the basic skeleton of the Bootstrap prototype. Bootstrap framework has been also chosen because vizGrimoire normally works with Bootstrap, and has some little dependences of it. Furthermore, most of dashboard pages that use vizGrimoireJS have similar Bootstrap style, so having a similar style to other environments which have been previously seen by the user provide him a more friendly experience.

There exist differences between the elements of JQuery Mobile and Boostrap, so some of them must be adapted to new framework. Anyway, at this point of the project, the idea is trying to keep most possible functional work. Main things to change in this migration are the top *header* from JQuery Mobile and content distribution.

Top JQuery Mobile *header* is changed to Bootstrap *navbar*, which is including "share" and "add tab" actions. Mainly, it has same functionality that JQuery Mobile, but it has some responsive features provided by Bootstrap, which make it completely adaptive to different device resolutions.

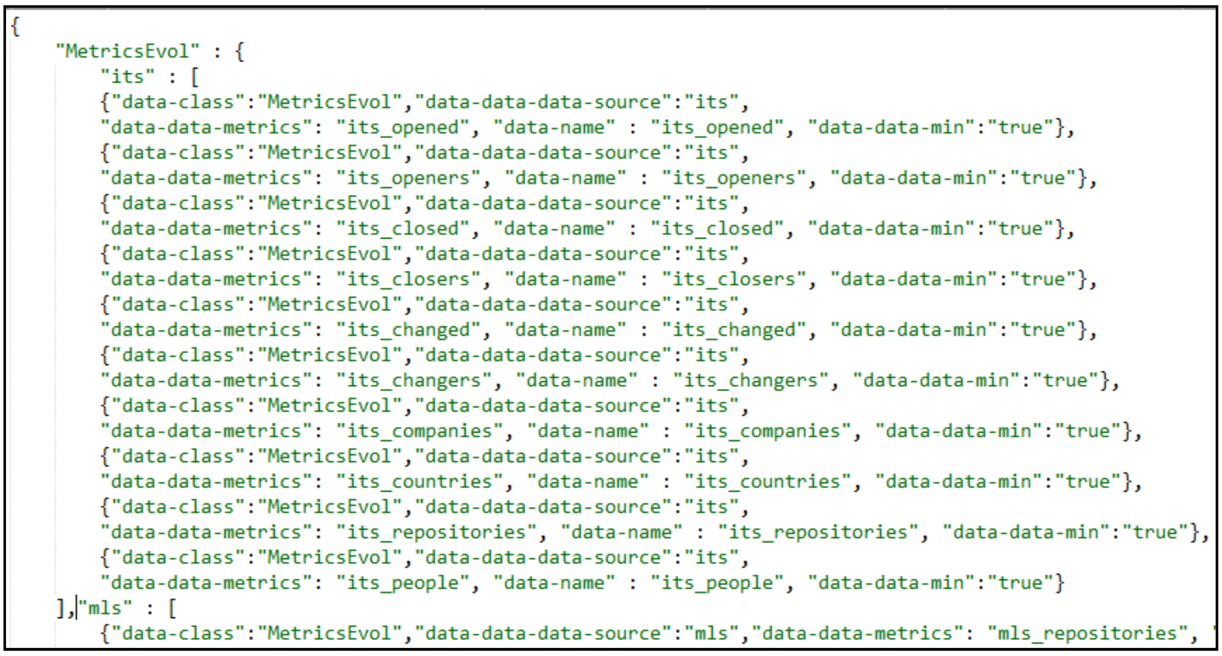
Body is distributed using a *span2 - span10* layout. Bootstrap divides imaginarily a width device in 12 columns, so the developer can distribute its HTML content along these 12 columns as he prefers. Bootstrap prototype is ready to have a 2 columns space destined for navigation panel and a 10 columns space destined for widgets, tabs, etc.

After this, prototype skeleton is almost done. One more thing remains: Navigation panel and JSON files structure.

3.5.3. Third task. Creating navigation panel

Navigation panel is changing a bit from JQuery Mobile version. Watching carefully figure 26, it can be seen that each widget has different attributes which specify its options. In this task, JSON file structure file is being kept, but with vizGrimoire integration, some things have changed.

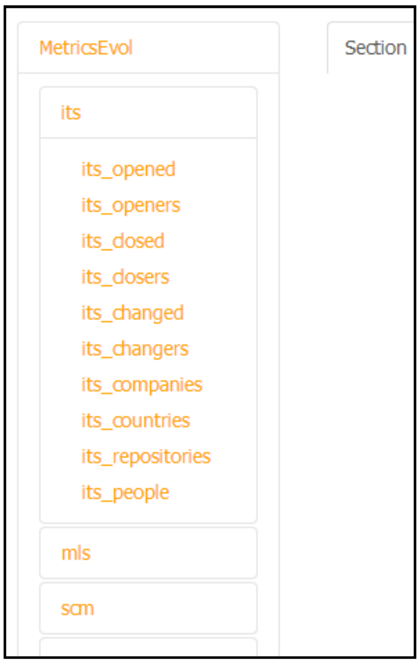
VizGrimoire is ready for reading JSON files from *data/json* folder in ROOT directory as default path. VizGrimoire reads all these JSON files and process them to visualize their data when it is specified in HTML skeleton. With VizGrimoireJS integration, JSON structure file is no longer used for defining directory structure, but for defining widgets options. More details can be seen in figure 28.



```
{
    "MetricsEvol" : {
        "its" : [
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_opened", "data-name" : "its_opened", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_openers", "data-name" : "its_openers", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_closed", "data-name" : "its_closed", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_closers", "data-name" : "its_closers", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_changed", "data-name" : "its_changed", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_changers", "data-name" : "its_changers", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_companies", "data-name" : "its_companies", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_countries", "data-name" : "its_countries", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_repositories", "data-name" : "its_repositories", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_people", "data-name" : "its_people", "data-data-min":"true"}
    ],"mls" : [
        {"data-class":"MetricsEvol","data-data-data-source":"mls","data-data-metrics": "mls_repositories",
```

Figure 28: MetricsEvol attributes definition.

Previous figure shows different attributes for *its* charts, inside *MetricsEvol* group. This file is also used for defining the navigation panel, which is going to be replaced from JQuery Mobile to collapse Bootstrap elements. Collapse is a type of element similar to a tree which bends and unbends to show and hide deeper levels. Figure 28 shows *MetricsEvol* sub-menu and its deeper levels. After completing JSON metrics file and including collapse code into dashboard prototype of the previous sprint, it can be seen in figure 29.



When some parameter like *its_opened* is clicked, a new element in inserted into Gridster panel using its related attributes from metrics file. If it is clicked, it generates a HTML code similar to figure 32. Figure 36 shows dashboard result when this button has been pressed.

Figure 29: Collapse navigation panel visualization.

Conclusions and results of the sprint

This sprint has proved how efficient is working with Bootstrap instead of JQuery Mobile. Besides of all amount of elements and facilities which provides, a native responsive behaviour is completely implemented, like can be seen comparing figures 30 and 31.

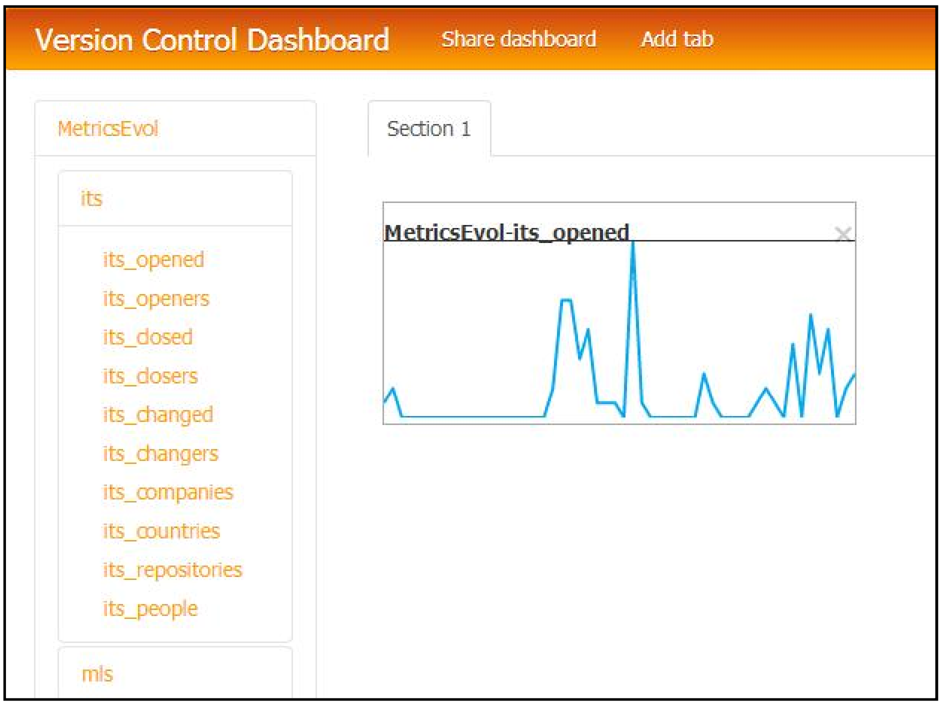


Figure 30: Dashboard view of one chart in big resolution device.

Between them, it can be seen that top header buttons are replaced for a three lines button which collapses and show them. This last figure is seeing in small screen devices, like smartphones or tablets.

VizGrimoire has been easy to use, and useful for project, because provides developer abstraction code creation for different types of widgets using its attributes system.

In this sprint, working previously with Flotr2 as a unique sprint has been useful because some problems have appeared when vizGrimoireJS and Bootstrap hav been integrated together. These problems have been related again with widgets visibility. When a new widget is inserted into the dashboard, it is necessary to be painted. VizGrimoire provides a method, *resizedw,* to paint all present widgets in a webpage. In this case, this refers to visible and non visible widgets (those ones inserted in tabs that are not currently selected). Knowing about Flotr2 errors has helped to solve this integration problems.

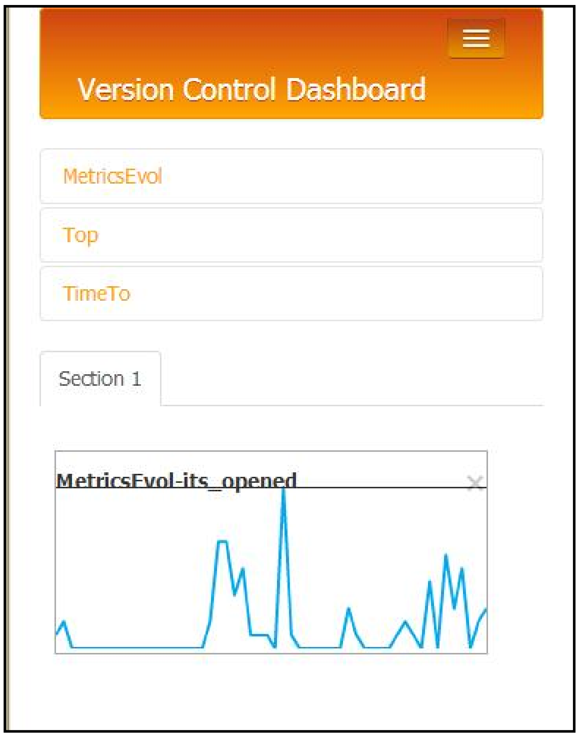


Figure 31: Dashboard view of one chart in small resolution device.

3.6. Sixth sprint. Increasing user experience into dashboard

At this point, the project is almost completed. An user is able to create a dynamic dashboard with different tabs and types of widgets. Once he has created a dashboard, he is able to share it easily, so another user or friend can watch, modify or share again it.

This sprint is devoted for adding last details and functionalities into dashboard to improve it. These details are not really necessary, but they add some user experience into the project. The last thing to do in this sprint, and also in the project, is uploading HTML project to Heroku[14].

3.6.1. First task. Improving user experience in tabs

This task pretends to add user experience in tabs allowing users to change tab names. This task is devoted to implement behaviour of tab name change and also the way which this

[14] Heroku hosting https://www.Heroku.com/

is informed to user, because this is a functionality that is going to be hidden except it is correctly said to user.

There exist a lot of ways to make user know that this functionality is available. Here, it has been done using tooltips. Tooltips are Bootstrap elements which appeared near a specified area. When user hovers mouse over a tab button, these tooltips make visible. Tooltip messages indicates that tab name can be switched double-clicking it.

Tab name functionality is implemented using a modal panel which tells user the previous name of the tab and also the possibility of changing it for a new one.

3.6.2. Second task. Improving user experience in widgets

This task is also destined to improve user experience, but this time is not tab user experience, but widgets user experience. Three features have been added to widgets in this task.

First feature added has affects drag functionality. Until now, all widgets could be dragged clicking at any part of the widget, neither title or graph. From now, Gridster constructor JavaScript code has changed in order to allow dragging only when title is clicked. To make user know this feature, when he hovers mouse over widget title, mouse icon changes and a tooltip is shown reporting this drag feature.

Second feature which is added in this task is removing functionality. Until now, a user was able to create and add a widget to the dashboard, but never remove it. Even though it is not a real problem or a bug, dashboard would not be completely customizable if there would not be a functionality for removing widgets. With this, a close button is added in right corner of any widget to include this functionality.

Last feature added in this task is allowing user to resize widgets. Similarly to Gridster drag region feature, Gridster constructor has changed in order to allow resizing of widget widths and heights. Gridster report this feature automatically when a user hovers any part of a widget, showing a symbol in the right bottom corner of the widget, easily visible by the user.

3.6.3. Third task. Uploading project to Heroku

With this previous details, project is already finished. Now, it is time to deploy it to some release public place. Heroku is a web service created for hosting different web applications in different programming languages, like PHP, Ruby or Python. Heroku provides some different pricing hosting, but for this project, free hosting is enough.

To upload a project to Heroku, it is necessary to have a Heroku account, install Heroku Toolbet in current operative system and follow some instructions[15]. After completing these instructions, this task is resulting to a public URL where the complete project is hosted.

[15] Getting Started with PHP Heroku guide https://devcenter.Heroku.com/articles/getting-started-with-php

4. Description of final application

The result of this project is a web application with a frontend part done using HTML5 technologies and a backend part written in PHP. A functional version of the application can be found hosted in Heroku server[16]. Frontend and backend communicates each other using REST JSON protocols. This application has been created in order to fix and improve current Software Development Dashboards.

A Software Development Dashboard is a set of graphs and data which show information taken from software development repositories. As it was said in other chapters of this document, current dashboards face the problem that their content is normally static. This means that an user cannot chose which information to be displayed or not.

Version Control Dashboards displays graphs and widgets with information taken from different repositories. All this widgets are creating through vizGrimoireJS library, which is a JavaScript library created Bitergia company, a group of people specialized in analyzing free software projects, take different stat data from them and visualized them.

VizGrimoireJS takes JSON files in a specific format containing the data acquired from repositories and visualized it as graphs, tables, or statistical numbers. The information which is visualized normally is related with software development, and can contain parameters like code management, issue tracking systems or mailing list, among others.

The JSON files which vizGrimoireJS uses are taken with another Bitergia tool. In this case, this tool is MetricsGrimoire and is responsible for obtaining data from repositories related to software development, like all types which have just been named.

The application has been created as a free software application. This has been done because a free software application grows up faster, because it has all community of free software developers behind in order to improve it or find any error. The application has been developed using GitHub repository[17] and all development commits can be found on it.

The application allows user to carry out a lot of different customizable operations over the dashboard he desires to create. From the beginning, he can create an empty dashboard

[16] Heroku application URL http://versioncontroldashboard.Herokuapp.com/
[17] GitHub application repository https://github.com/JuanCarlosCamara/Dashboard-pfc

where adding different kinds of widgets, which different content in them. Trying to add more customization dashboard, user can chose the position and size of every widget, so he can move or resize them as he prefers. If he wants to remove any widget, application allows this functionality too.

Following this customization principle, the application provides user the option for classifying dashboard content in different tabs, so the user can chose how to order its content.

At last, application has implemented a share functionality. With this functionality, a user can get a reference URL for the current status of his dashboard. Once he has obtained this link, he can recover when he wants or give it to any other person to allow him to modify it and create a new one for him, without worrying about modifying the original one. An example of a view of a dashboard created with the application can be seen in next figure 32.

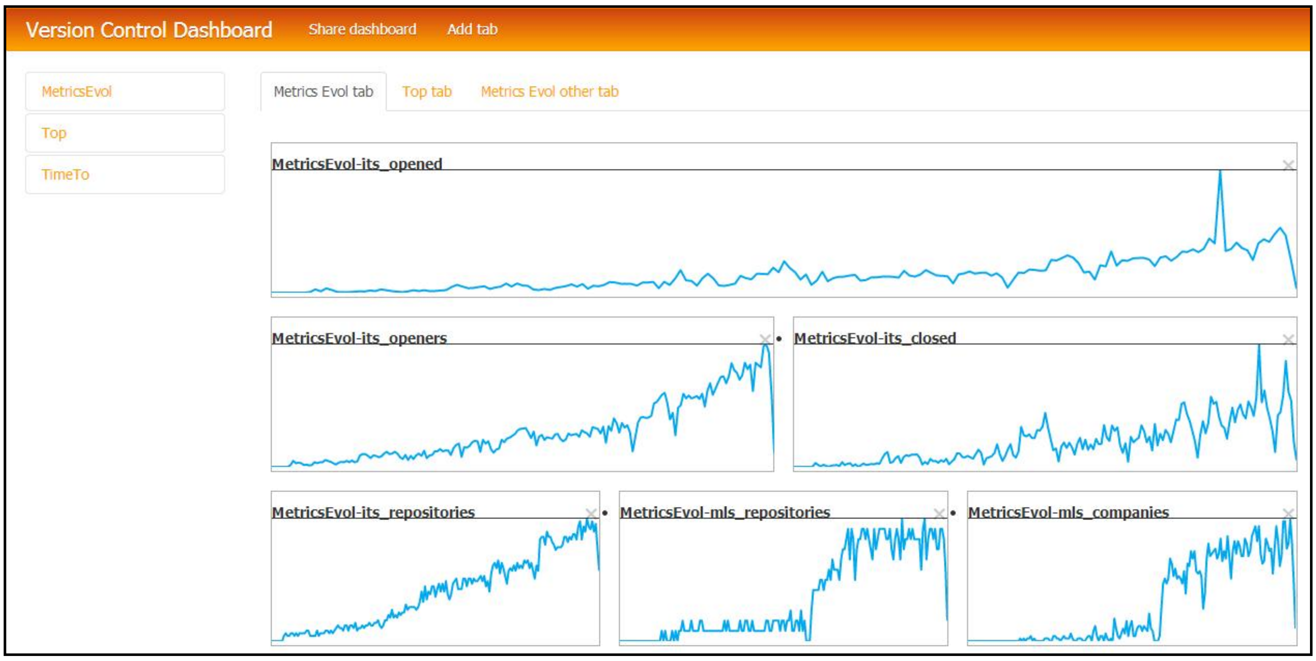


Figure 32: Application view of a completed build dashboard.

In next apart of the document, the resulting file application structure will be studied carefully. After it, all previous functionalities are going to be explained in next step, with a lot of details about how they work, descriptions and images of them. Last apart of this chapter will be a little description about installation process.

4.1. Application structure

This little chapter has been done in order to explain the structure of the resulting web application, as well as given a simple explanation about the code which makes it works.

Two main parts can be differentiate in the application. On one hand, the main functionalities are found in the client part, which is purely done with JavaScript and it includes the functionalities realized by all the different libraries which have been used, like dragging, tabs, Bootstrap components, etc.

On the other hand, application includes a server part developed with PHP. These PHP server has been implemented in order to carry out share functionality. There is a communication REST JSON between client and server. This PHP server provides two methods, one for storing the dashboard content (POST) and other for recovering it(GET). The share functionality will be explained later, in next chapter. Next figure 33 visualizes the JSON communication between client and server.

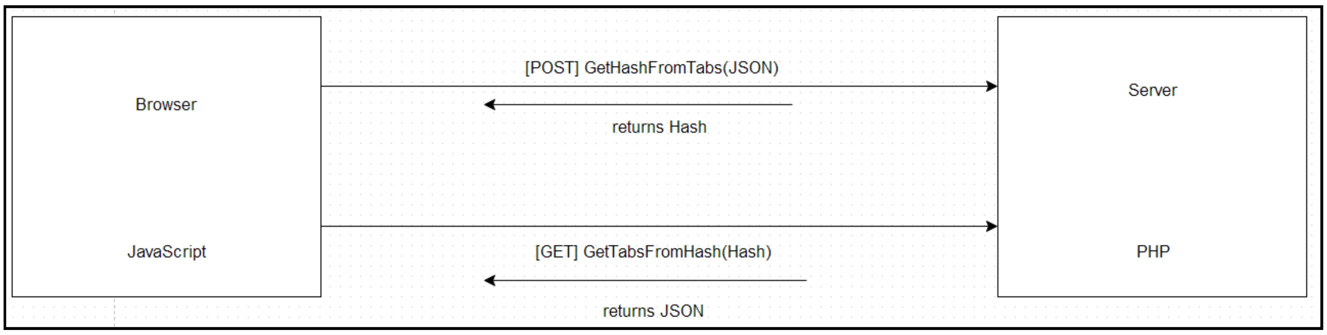


Figure 33: Basic schema communication between client and server parts.

Client application part includes the whole JavaScript code, the HTML architecture and the CSS style files. Basically, the application structure is divided into different directories that classify project code files as a function of its type.

Mainly, the application is specified with HTML file, *index.html*, and the JavaScript code. The JavaScript code is divided in different parts. On one hand, the main code developed along the project can be found in path *js/index.js* file, and it is where the whole project functionality is implemented. This means that this file includes creating navigation panel, communication between client and server, reconstruction of a dashboard when sharing, tab insertion, etc

Main logic is included in previous file. However, other JavaScript files are included inside the set of directories. These files are the libraries that has been used along the project, as Gridster or vizGrimoireJS.

VizGrimoireJS has been really important during the project and it has a full directory for itself, where all its functionality is included, as well as the Bootstrap framework libraries inside it. The reason why whole vizGrimoireJS directory is included in the result of the project is because it includes some documentation and examples that can be useful for later situations.

Last important directory is data directory. This file includes the JSON files that refers to the widgets and data to show. It is important these files to have a specific structure. Else, vizGrimoireJS library will not be able to process correctly the information and visualize it.

Normally all these files are obtained through MetricsGrimoire library, that analyze repositories and creates all these necessary files, ready to be processed by vizGrimoire. However, this is not mandatory, and user can obtained them by different ways.

4.2. Functionality

This apart of the document is destined to summarize all functionalities of the application and show reader how they work. To start from this point, it is necessary the application to be correctly installed, following the steps announced in next step.

4.2.1. Creating an empty dashboard

The application automatically created an empty dashboard when it is called as first time. This is one of the initials specifications of the project.

Empty dashboard presents three main differentiate regions. These regions are side navigation panel, top bar and main dashboard region. In the top bar, two main actions exist, *share dashboard* and *add tab* buttons. White region is main region of the application, and it contains all different widgets, as well as division tabs to organize the dashboard. Side panel content refers to the kind of widgets which can be added, their content and features.

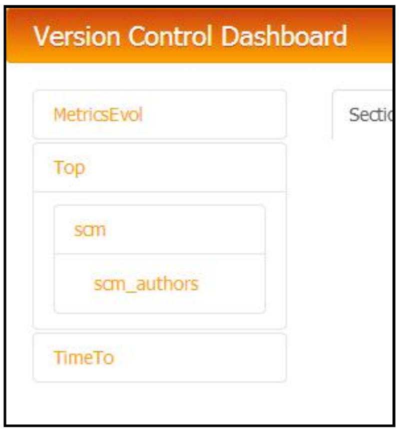
4.2.2. Adding widgets to dashboard

Widgets are added from navigation panel. Before showing navigation panel working, some little feature about it must be known.

Navigation panel is formed from the content of *metrics.json* file, inside *data/json* directory. This file will be explained later in installation process but, as summary, this JSON file contents the structure of the navigation panel, specifying types of widgets their content and all their features.

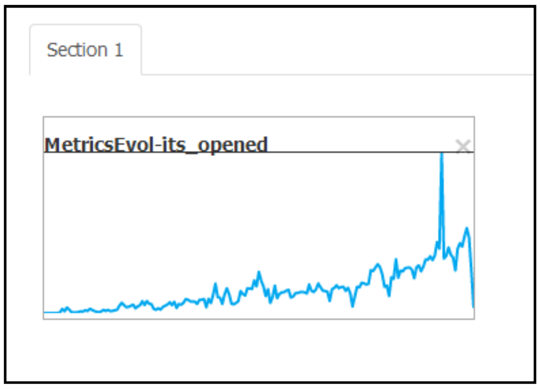
Once the application has read *metrics.json* file, the navigation panel is formed as a collapse menu, whose each directory can be expended and collapsed, showing or hiding its content. This behaviour can be clearly seen in next figure 34.

When one link of the navigation panel is clicked, a new *div* tag is included into current tab, including widgets features which were specified into *metrics.json* file.



After clicking and including tab into dashboard, vizGrimoireJS code is called. vizGrimoireJS reads current HTML file and looks for some specified class and tags. Once file reading has been completed, vizGrimoireJS repaints all tag content with respective widget.

Figure 34: Top expansion of the side navigation panel.



Once there is some widget added to the dashboard, basic functionalities can be applied on it. These functionalities are changing position with other widgets of the same tab, resize or removing from dashboard.

Figure 35: Widget included into section1.

4.2.3. Widget functionality: Changing widgets position.

Application has been developed using a library called Gridster.js[15]. The template of each new tab and it content is based on this library. Gridster provides a grid of draggable elements, where width and height of each new element is based on grid cell features. This cell features refers to minimal unity of the grid, and they are specified in Gridster initial code.

Normally, Gridster elements can be moved clicking and dragging from any part of the widgets, but in this application this has been modified in order to allow draggin only clicking in widget title, as can be seen in figure 36, where it is notified with a tooltip.

These functionality only allows moves inside same tab. Inter-tab moves are forbidden in this version of the application. One example of moving can be seen in figure 37, where the widget of the figure 35 has been moved to mid position.

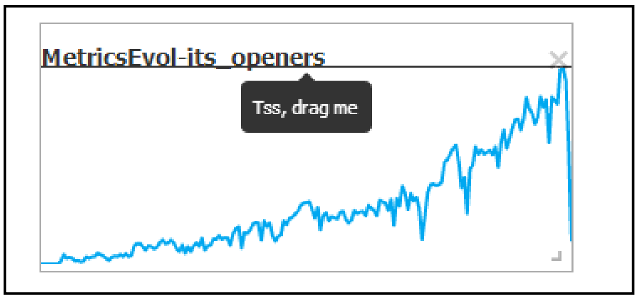


Figure 36: Moving, resizing and removing wigdet notifications.

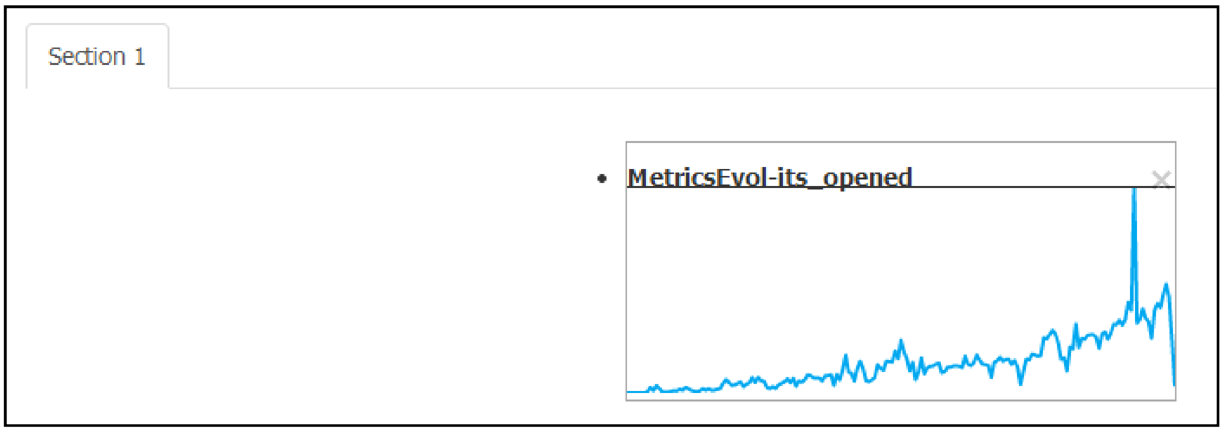


Figure 37: Result of moving operation on a widget.

4.2.4. Widget functionality: Resizing widgets

Gridster implements some more functionalities than just moving widgets from one to another position. If it is specified in its initial JavaScript code, widgets size can be changed by user if it is desired.

When this flag is activated in initial Gridster code, a new mark appears inside widget to notify that this feature is available. This notification is found at right bottom corner when user hovers mouse over element, as can be seen in figure 36. Gridster only allows resizing from the right and bottom parts of the element, as well as common corner, but not resize from left or top of widget.

As it was said before, Gridster specifies a size for the elements which contains depending on a basic unit size. With this, the final widget size after resizing will be a multiple number of this basic unit size. Next figure 38 shows figure 35 after resizing widget to double width and double height.

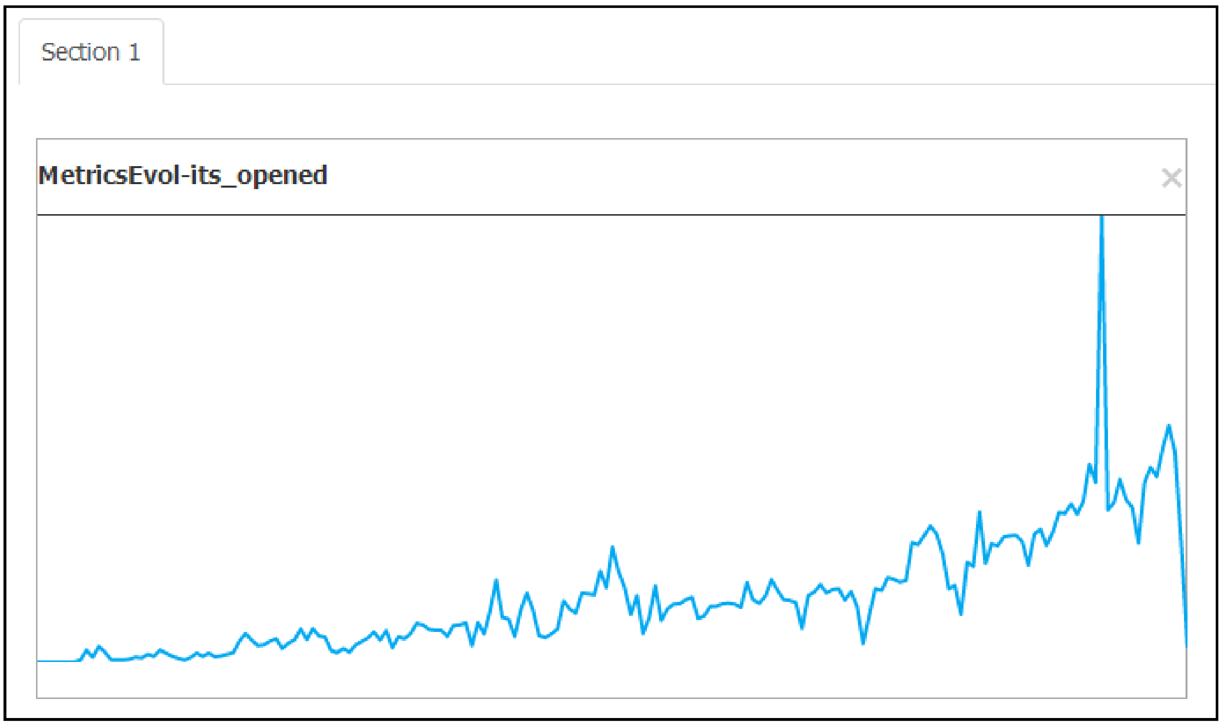


Figure 38: Result of resizing operation on a widget.

4.2.5. Widget functionality: Removing widgets

Last functionality implemented using Gridster library has been removing widgets. If users wants to remove a widget from dashboard, it can be done clicking on cross symbol of the top right corner of the widget. This cross symbol is seen in figure 36.

4.2.6. Classifying dashboard: Including tabs

As it has been explained before, when *add tab* button is pressed, a new tab is added to dashboard region, as can be seen in figure 39, where *add tab* button has been clicked 3 times.

When this button is clicked, it includes a new tab to the tab list, and also it creates a new region which is referred from this new created tab. Each tab refers to a different region which is hidden or shown as a function of which tab is active. This means that each tab refers to a different content.

When a widget is included, application looks which one is the current tab and inserts new widget in it, in the respective new position.

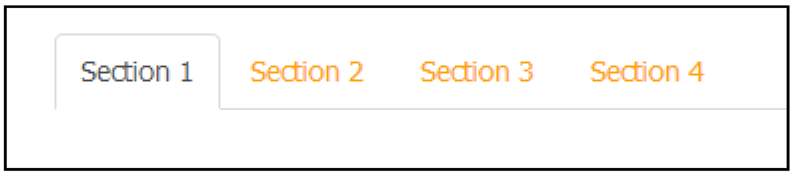


Figure 39: Add tab button has been clicked thrice.

Relative to tabs, another functionality apart of changing exist, which is allowing user to change tab names.

User may change tab name to add description of each tab, of a symbolic name about content. A tooltip message notifies this when user hovers tab with mouse, and it says that tab name can be change if it is double click, as can be seen in figure 40.

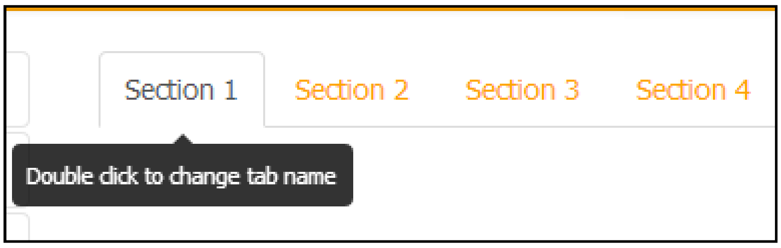


Figure 40: Tooltip which notifies tab name change action.

Once a tab is double clicked, a modal asking for new name is shown, and this will replace previous tab name for the new one. This modal can be seen in figure 41, while tab name change is visible in figure 42.

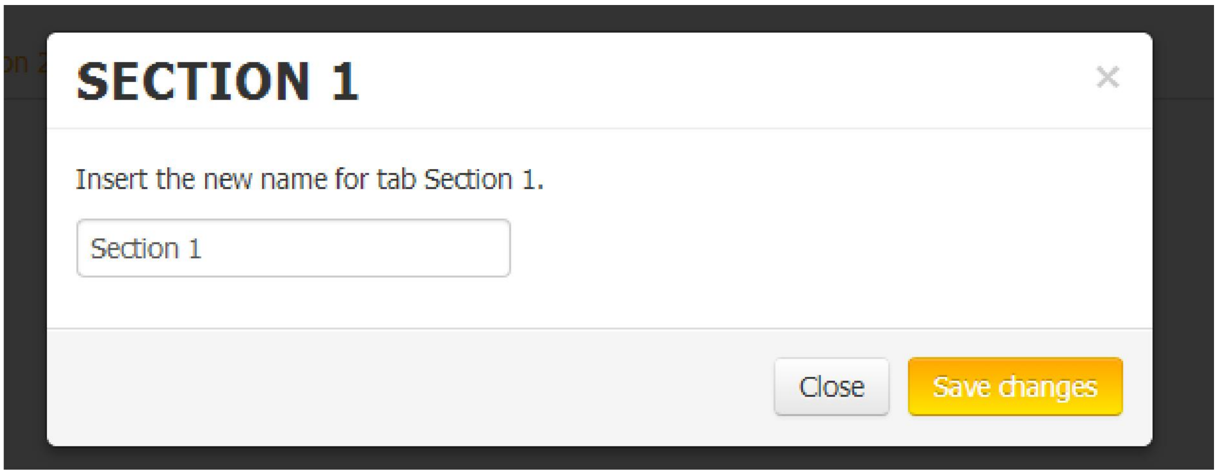


Figure 41: Modal box which appear when tab name is clicked twice.

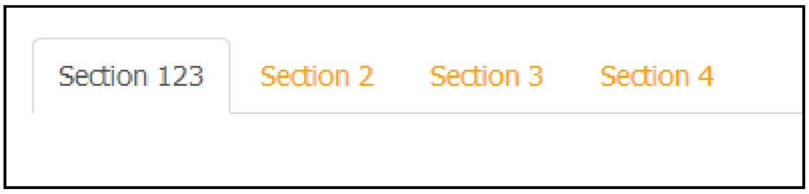
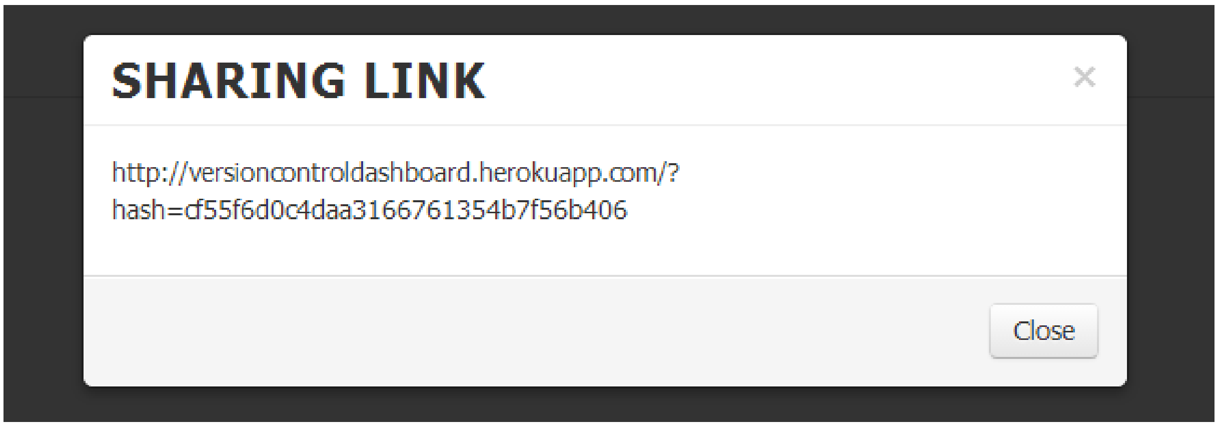


Figure 42: Tab names after changing.

4.2.7. Sharing a dashboard

At top bar, another button can be found. The button *share dashboard* allows user to get a URL containing a hash code which refers to current dashboard status. Once user dashboard is completely finalized and he wants to store a permanent copy of it or share it with any other person, he can use this functionality. Figure 43 shows URL returned for a dashboard which has been desired to be shared.


SHARING LINK

http://versioncontroldashboard.herokuapp.com/?hash=cf55f6d0c4daa3166761354b7f56b406

Figure 43: Box containing sharing link.

Share functionality works as follows. When the button is pressed, the current status of the dashboard, including number of tabs, their names, their widgets and positions are stored in a *object* variable in JavaScript. This object is sent through a POST call to PHP server, where it is stored permanently as a file. PHP server converts this object to String and calculates its respective hash code, which is returned to client application. This returned hash code is shown in sharing link box shown in figure 43. This hash is also the name of the respective file where the JSON is stored.

When share URL is called, inverse process is carried out. In his situation, application reads URL and looks for any hash code. If there is some hash code inserted in the URL, application asks for the respective JSON object to the PHP server. It carries out a GET call to the server, using the hash code obtained from the URL.

When PHP server receives a GET request asking for a JSON content, it looks the file which is referred by the requested hash code and reads its content. PHP server returns client the information obtained from the file as JSON type. After receiving the JSON, client is responsible for repainting the whole dashboard, with all containing information. After recovering a shared dashboard, the number of tags, their names and the widgets which each tab contains will be exactly the same than in the original dashboard.

For example, if the dashboard of the figure 38 is shared, the resulting dashboard is shown in figure 44, which is exactly equal to the original one.

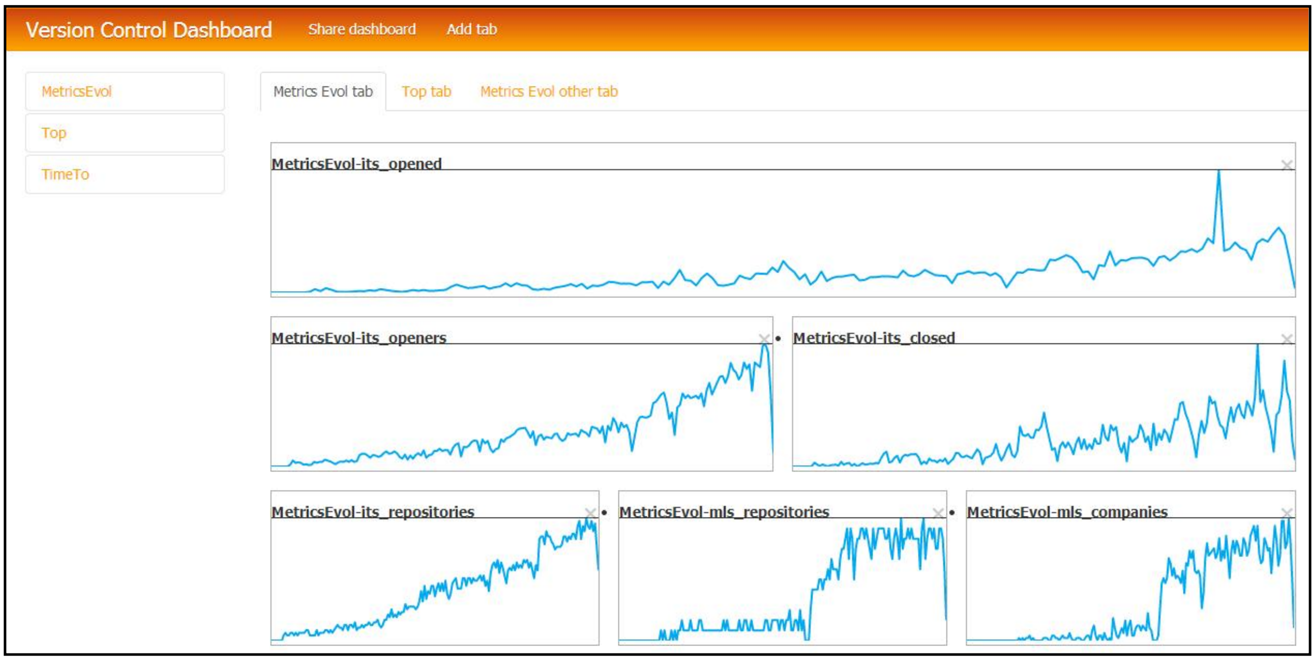


Figure 44: Resulting dashboard after sharing.

4.3. Web application installation process

Since result application is a web application, first thing which reader has to do is get a web server able to resolve http requests. To allow share functionality, it is also important that server host has PHP service installed.

There exist different ways to get application installed in a machine. On one hand, it is possible to use some of the free web hosting services provided through Internet, like Heroku or ByeThost. One bad thing with this kind of services is that they provide free hosting in the beginning, but if the disk space or bandwidth is increased, prices are also increased.

On the other hand, user can install his own web server in one machine of his own. In this document, Xampp has been recommended, because it includes both services, web and PHP, between others. However, this project has been developed as a free software project, and any person can checkout it from Github repository and modify and adapt it to work with other kind of service and programming language, such as Python or nodeJS.

Xampp can be easily installed from Apachefriends page, where user has to chose his platform binary installation package. Once this file is downloaded, installer steps must be followed and some options should be chosen.

Once Xampp package is installed, it is time to allocate HTML project, which last version can be forked from gitHub repository, in *htdocs* directory inside Xampp root directory. This *htdocs* directory is the directory where Apache Xampp service resolves HTTPs URL calls. For example, if user desires to have Version Control Dashboard project reachable through URL http://localhost/versioncontroldashboard/index.html, whole HTML project has to be allocated in *{Xampp_path}/htdocs/versioncontroldashboard* directory.

Last version of the project has a directory structure which can be seen in figure 45. This whole structure must be allocated inside *docs/versioncontroldashboard* folder. Each directory contains different parts of the project.

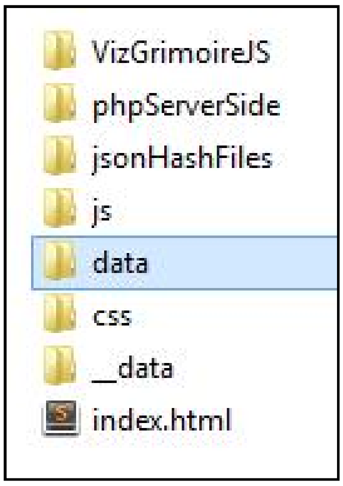
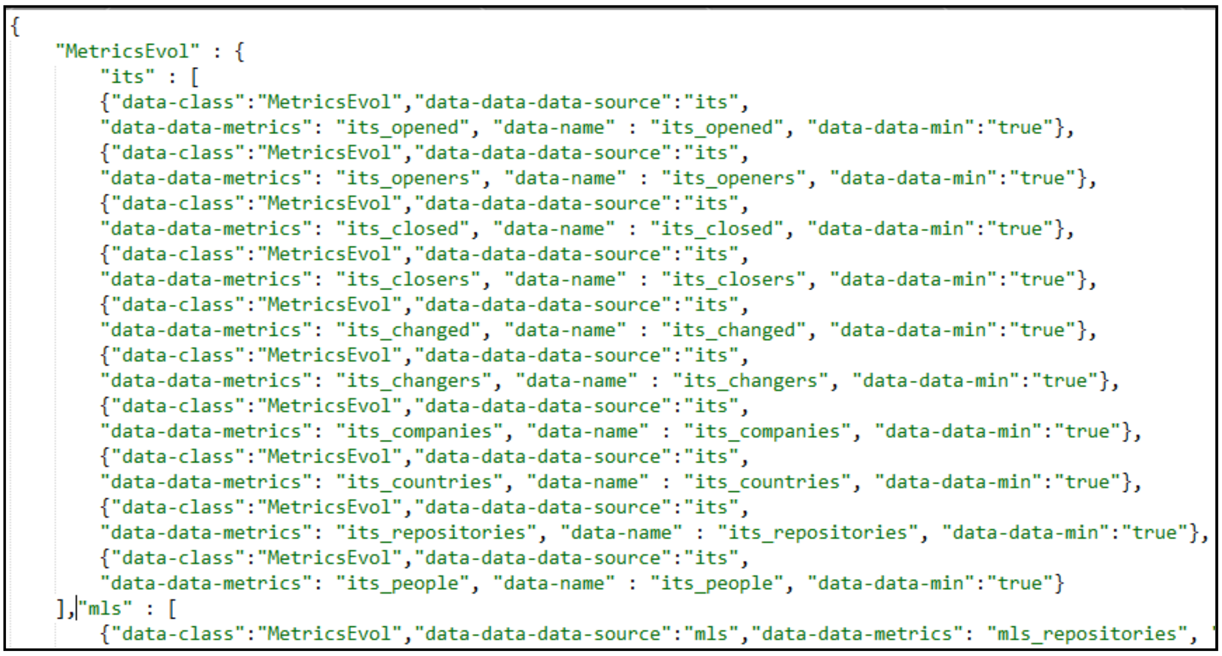

VizGrimoireJS
phpServerSide
jsonHashFiles
js
data
css
__data
index.html

Figure 45: Version Control Dashboards project directory structure.

For example, VizGrimoireJS includes all vizGrimoireJS forked project, while phpServerSide includes REST json methods for share functionality. When a dashboard, is shared, the JSON file which is created is stored in jsonHashFiles directory. At last, js, css and index.html are HTML directories, which contains main javascript, css and HTML code of the project.

There are two more directories of special attention. VizGrimoireJS library process JSON files for visualizing from *data/json* directory. This *data* and *__data* include some example JSON files. In theses directories, *metrics.json* files is also included, necessary for showing navigation side panel. If a user does not fill correctly this file, navigation panel will not work correctly.

Metrics.json file contains a JavaScript object which refers the structure of the side navigation panel. It contains the types of widgets, as well as their attributes and the information which is related to. An example of this file is shown in next figure 51, where a type of widget "MetricsEvol" has been defined. Inside this type of widget, it can show different information, such as *its* or *mls* information. Following *its* example, figure 46 indicates that inside *its* widgets, the features and the data can be chosen too.



```
{
    "MetricsEvol" : {
        "its" : [
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_opened", "data-name" : "its_opened", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_openers", "data-name" : "its_openers", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_closed", "data-name" : "its_closed", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_closers", "data-name" : "its_closers", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_changed", "data-name" : "its_changed", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_changers", "data-name" : "its_changers", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_companies", "data-name" : "its_companies", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_countries", "data-name" : "its_countries", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_repositories", "data-name" : "its_repositories", "data-data-min":"true"},
        {"data-class":"MetricsEvol","data-data-data-source":"its",
        "data-data-metrics": "its_people", "data-name" : "its_people", "data-data-min":"true"}
    ],"mls" : [
        {"data-class":"MetricsEvol","data-data-data-source":"mls","data-data-metrics": "mls_repositories",
```

Figure 46: Fragment of *metrics.json* file.

5. Conclusions

At the beginning of this document, it was said that a web application was wanted to be developed. This application would use JavaScript technology as frontend part and PHP as backend server, and it would be created following a Scrum methodology, divided by sprints were different prototypes would be created. In each sprint, some new knowledge would be acquired and prototype would be improved.

This application should be able to create dashboards and show different information related with projects, like the amount of people involved in the project development, the amount of modified lines, or other statistical data.

Summarizing, the main objective of this project was creating an application which would provide information about the evolution of projects, as well as reporting amount of issues or problems which would appear along its duration.

At the end of the project, all this previous has been reached. The result has been a web application which can be executed from any browser. This application allows user the possibility of creating dashboards with different content in order to show the information he desires. This application contains all functionalities which were specified at the beginning of the development process: creating an empty dashboard, including and removing resizable and draggable widgets and sharing its content.

Along the project, some different conclusions have been obtained. For example, the project has been reached easily and without many problems thanks to the development methodology which was implemented. Scrum tries to carry out a project dividing its process in little parts. At the end of each part, a prototype of the application is obtained and shown to the Product Owner, which is the role of the person responsible of saying if the current prototype follows expectations or there is something to be changed on it.

Scrum has proved to be a very good development methodology. Scrum tries to avoid errors in development and creating things which the client does not really want. This take place at the end of each sprint, when prototype is shown to product owner. Scrum methodology avoids wasting time in development and a lot of things to be changed at the end of the product, in the final version.

Different hours were dedicated for each sprint depending of its difficulty and the different task which included. Reader can get more details checking development process chapter. Next there is a little time details about each sprint:

- Creation of a dashboard with two graphs using Flotr2: The idea for this sprint was creating a web page able to read data from a JSON file and visualize it in two different graphs, with different options. For this sprint development, it was necessary around 10 hours.
- Creation of a dashboard with draggable graphs using Gridster and JQuery Mobile: At the end of this sprint it was desired to have a dashboard prototype which would include both Flotr2 and Gridster. It was necessary around 10 hours to complete this sprint.
- Creation of a draggable dashboard with tabs and multi JSON data source: dashboard prototype of this sprint should include tabs for becoming the interface a bit more friendly for the user. In this sprint, first contact with real data appeared. Nearly 15 hours were necessary to complete this sprint.
- Creation of a dashboard with share functionality: This sprint was destined to include share functionality into the prototype of the previous sprint. This sprint needed approximately 20 hours to be completed.
- VizGrimoire.js integration and Bootstrap migration: Migration took a bit more of time that all other sprints, because of changes and looking carefully that everything worked as before. This sprint was finished in approximately 30 hours.
- Introducing last details into dashboard to improve user experience: This was a very light sprint, with simple tasks. It was completed in around 10 hours work.

Some different tools, libraries and frameworks have been used along the project development. Mainly, they have been Gridster and vizGrimoireJS JavaScript libraries and Bootstrap framework.

Talking about Bootstrap, it has proved to be a very easy to use HTML framework. It is really simple to use and create a HTML system in a few minutes, even though for people which is not a specialist in this field. For example, there exist a lot of backend programmers who are not really good working with HTML or CSS. Bootstrap provides some pre defined elements and tags, among others, that give developer abstraction about the hidden HTML or CSS code. With this, it is possible to add pages nav bars, grids, footers, modals, ...

Gridster has also given a very good impression. Gridster allows abstraction for creating a grid of draggable elements. In this project, these elements are the different graphs or widgets inserted by the user into the dashboard, while the grid itself refers to the proper tabs content inside it. Gridster has proved to be very powerful, allowing not only dragging, but also a lot of actions over the widgets, all of them to provide dynamism and customization to the user.

VizGrimoireJS is the library responsible for drawing and inserting the widgets in the tabs. VizGrimoireJS abstracts user of the visualization process, as well as the specification of the different features of the graphs, as well of the HTML code. All related information about what vizGrimoireJS needs for visualizing is specified in the *metrics.json* file, which mainly contains the types of widgets, the drawing features and the data which visualized each option.

VizGrimoireJS is rather simple to be used, but some little problems working with it have appeared. This library contains a hidden implementation of Flotr2 library. This library needs container div of the graph to be visible, with a positive height and width. This is not really a problem, and it has been easy to carry out.

Dashboard application is divided in tabs to classify and order its content. This provides user some simple classification functionality. However, some troubles appeared between vizGrimoireJS and this multi tab system. When a new widget is inserted in the dashboard, vizGrimoireJS has to look for all tags referred to a widget and repaint its respective widget. When vizGrimoireJS tried to paint a widget that did not belong to the current tab, application crashed showing a message telling that objective *div* is not visible.

Previous situation took place also when a dashboard was tried to be recovered from sharing. When recovering, the same function that named in previous paragraph was called, and same error was obtained because, again, the objective *div* was not visible. It was a bit difficult to find this error and face them.

Resulting application can be used in a lot of different scenarios. Main of them is based on its share functionality. A project leader can create a dashboard with information related about development evolution, and give it to the development team in order to motivate them or show them the statistical evolution. Anyway, this application has been developed for carrying out different project managing task and purposes.

This project has been based on a simple idea and final application is only a basic implementation of it. There are still a lot of different things which can be done about dashboards related with projects and version control. Future about it is not written yet and some different applications can be created from this one.

For example, using this HTML5 technology, is possible to create a native mobile application for different platforms. A combination of Bootstrap and Cordova allows any person to create a responsive application easily for different devices and their operative systems using HTML5. Other example of a dashboard application can have a register users structure to keep their dashboard configurations.

Summarizing, application created provides manage knowledge and skills in a customizable way, but still can be improved thanks to free software technology. It is expected this application to be improved quickly because of free software community.

6. References

[1] Sublime Text official Website http://www.sublimetext.com/

[2] Scrum Methodology and Project management http://www.mountaingoatsoftware.com/agile/scrum

[3] Git Official Website http://git-scm.com/book/en/Getting-Started-About-Version-Control

[4] Software development analytics for Open Source projects - Bitergia http://bitergia.com/

[5] The vizGrimoire project http://vizgrimoire.bitergia.org

[6] MetricsGrimoire project http://metricsgrimoire.github.io

[7] VizGrimoireJS github repo. https://github.com/VizGrimoire/VizGrimoireJS

[8] Bootstrap v3.1.1 Website http://getBootstrap.com/

[9] Bootstrap GitHub repository https://github.com/twbs/Bootstrap

[10] Bootstrap v2.3.2 Website http://getBootstrap.com/2.3.2/

[11] JQuery http://JQuery.com/

[12] JQuery API documentation http://api.JQuery.com/

[13] JQuery GitHub repository https://github.com/JQuery/JQuery

[14] JQuery plugin web page http://plugins.JQuery.com/

[15] Gridster JQuery plugin http://Gridster.net/

[16] Flotr2 official Website http://humblesoftware.com/flotr2

[17] Flotr2 GitHub repository https://github.com/HumbleSoftware/Flotr2