

Master's Thesis

Development of a Cloud-Based Configuration Management Database

David Rubino

Year 2014-2015

Final year internship, conducted at Prime Resources
to obtain the engineering degree of TELECOM Nancy

Internship supervisor: Ritch Houdek
University supervisor: Moufida Maimour

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Rubino, David

Élève-ingénieur(e) régulièrement inscrit(e) en 3^e année à TELECOM Nancy

N° de carte d'étudiant(e) : 31010226

Année universitaire : 2014 - 2015

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

Development of a Cloud-Based Configuration Management Database

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Milwaukee, le 08/ 31/ 2015

Signature:



Master's Thesis

Development of a Cloud-Based Configuration Management Database

David Rubino

Year 2014-2015

Final year internship, conducted at Prime Resources
to obtain the engineering degree of TELECOM Nancy

David Rubino
W170 N11525 Armada Dr. Germanton, WI 53022
262-302-9049
david.m.rubino@gmail.com

TELECOM Nancy
193 avenue Paul Muller,
CS 90172, VILLERS-LES-NANCY
+33 (0)3 83 68 26 00
contact@telecomnancy.eu

Prime Resources LLC
N58 W24793 Quail Run Ln. Sussex, WI 53089
262-465-6750

Internship supervisor: Ritch Houdek
University supervisor: Moufida Maimour



Acknowledgments

This project would not have been possible without the support of many people.

First and foremost, I wish to express my deepest gratitude to Ritch Houdek, my industrial supervisor, who supported me during the entire project. With his numerous suggestions and ideas, he was able to help me pursue the development of the application. Ritch was always there when I needed help, and I am deeply grateful to him.

Then, I would like to thank John Houdek, director of Prime Resources, for giving me the opportunity to do my internship in his company. I also thank him for his help with all the paperwork involved, especially for getting the J-1 visa in order to come to the United States.

I would like to show my gratitude to Kasandra Wysocki and Colin Brady, concierges at the Hudson Business Center, for the help they provided me as well as their daily good mood that made each day of work a great experience.

I also would like to thank all the other Hudson members I was able to meet, for their advice, good moods, and uplifting spirits that helped me understand the spirit of American entrepreneurship.

I thank my university supervisor, Moufida Maimour, for following my progress during my internship and correcting this report.

Last but not least, I thank all members of the jury for reviewing my work and giving me the opportunity to present the results of this great professional experience that my internship was.

Summary

Acknowledgments	5
Summary.....	7
1 Introduction	11
2 Company Presentation.....	13
3 Project Description	15
4 State of the Art.....	19
4.1 Information Technology Infrastructure Library (ITIL).....	19
4.2 Configuration Management Database (CMDB).....	20
5 Problem Analysis and the Solution: a Cloud-Based CMDB	23
5.1 From a User Point of View.....	23
5.2 From a Developer Point of View	25
5.3 Why a Minimally Viable Product	25
5.4 Answering the Flaws of Enterprise CMDBs.....	27
5.5 The Importance of Cloud-Based Applications	28
6 Used Technologies	33
6.1 Programming Languages.....	33
6.2 XAMPP	33
6.3 phpMyAdmin	34
6.4 jsTree	34
6.5 Vis.js.....	34
6.6 Aptana Studio	35
7 Implementation of the Solution	37
7.1 Data Model	37
7.2 User Administration	41

7.3	Interaction with the Database	46
7.3.1	Connection to the Database	46
7.3.2	User Session	47
7.3.3	Retrieving Data with Ajax.....	48
7.3.4	Updating Data with Ajax.....	50
7.4	Graphic User Interface Components	51
7.4.1	Desktop View	52
7.4.2	Mobile Device View.....	53
7.5	Application Features.....	55
7.5.1	Configuration Items.....	55
7.5.2	Applications.....	62
7.5.3	Data Centers	66
8	Technological Progress and Economic Strategy	77
8.1	Technological Progress	77
8.2	Economic Strategy	78
9	Future of the Application.....	79
10	Work Methods and Project Management	81
10.1	Gantt Diagram	81
10.2	Deliverables.....	82
10.3	Meetings with the Industrial Supervisor	82
10.4	Source Code Management.....	83
11	Conclusion.....	85
12	References	87
	List of Figures.....	91
	Glossary.....	93
	Addenda.....	95

Résumé	104
Abstract.....	104

1 Introduction

This project took place for my last year internship at TELECOM Nancy to get my Master's Degree in Computer Science. It was directed by a small start-up company from Wisconsin, Prime Resources.

The main goal of the internship was to apply in a specific project all the skills and knowledge I learned during my training as a graduate student. Though challenging, it helped me develop new skills as a software developer, and also helped me to improve my analysis and conception skills.

The task that was given to me was to develop a Configuration Management Database (more commonly referred to as CMDB in this report) application aimed at small and medium-sized companies. The objective was mainly to write something simple, user-friendly, and efficient in terms of rendering, based on the notion of Minimally Viable Product (MVP), meaning that the development is focused on the main functionalities of the application.

The goal was to create a CMDB that users would want to use. Without competing with major CMDBs available on the market such as OneCMDB, our application was aimed for people who would like to quickly access the different dependencies of their system, and who would feel the need to use it without spending too much time on it. Thus, the software was designed as a cloud application for an easy access, whether on a phone, tablet, or PC. The advantage of not having to download and install something on the computer or phone was essential for us, as it improved the attractiveness of the product and the time that a user would spend to configure the application.

A CMDB consists mainly of listing all the different resources that a company has (servers, databases...) in order for a specific user to know which of these resources are used by a specific department or network. It is really useful when an administrator would like to upgrade one of his servers, but first would want to make sure that the different dependencies linked to the server would not create a version problem for a specific application linked to that server. Thus, it enables not only administrators, but every single user to know exactly how the company resources are implemented, and to have an idea of which parts would be impacted by a specific change.

The problem about CMDBs though, is that it can soon become too heavy to keep up-to-date. For example, big retail companies like Kohl's or Macy's require heavy CMDB software in order to support their vast network spread across the country. It soon becomes a big charge to make sure the CMDB is up-to-date and running with the latest versions of the different components of the company. Administrators do not always have time to update the software, and users can get discouraged to use it since it does not really reflect the state of the company at the present time.

CMDBs can then become neglected, which is a financial loss since thousands of dollars are usually invested in getting a license. Obviously, small and medium-sized companies would have no interest in investing such a big amount of money in products that are complicated to use and that would require too much time to keep up-to-date.

Although CMDBs on the market are apparently complex, the idea of knowing exactly what is going on in the company with all the different resources is an important, if not necessary, measure every company should strive to have. We believe that such companies are looking for a simple and easy way to manage their network, without having to spend too much time learning how to do it and actually doing it.

That is where the subject of my internship comes up. My objective was to write a CMDB specifically aimed at such companies. It should be easy to use by any employee, no matter what their position could be, and easily maintainable as well.

In order to present the subject in more detail, I first introduce the company in which I did my internship, Prime Resources. Then, I describe the project in more detail and take a look at the different solutions available on the market. Next, I analyze the problem and describe the solution that was implemented before exposing the different technologies used to develop the product. Then, I study in more detail the implementation of the solution before talking about the economic strategy behind it. Finally, after describing the future potential of the application, I expose the different work and management methods used during the project.

2 Company Presentation

In this part, I present the company in which I had the opportunity to do my internship. Created in 2014, Prime Resources LLC is a start-up company focusing on providing top-notch services to other companies. Its main goal is to recruit people having the skills necessary to answer one of the clients' most challenging problems. The different domains of expertise of the company do not only include information technology but also manufacturing, executive search, and engineering. In the latter category, different subdomains relative to mechanical engineering are provided by the company, like machine design, product design, solid modeling, and finite element analysis, or FEA [1].

Prime Resources was originally located in Hartland, a small town in the Greater Milwaukee Area. It recently moved its headquarters to the nearby city of Sussex. It is located on Main Street, right in the heart of the city. Figure 1 below shows the location of Prime Resources with two maps. On the left, it is a view of the company's headquarters in Sussex. The map on the right shows where the company is located regarding Milwaukee and the other surrounding towns. Both maps were taken using Google Maps. [2]

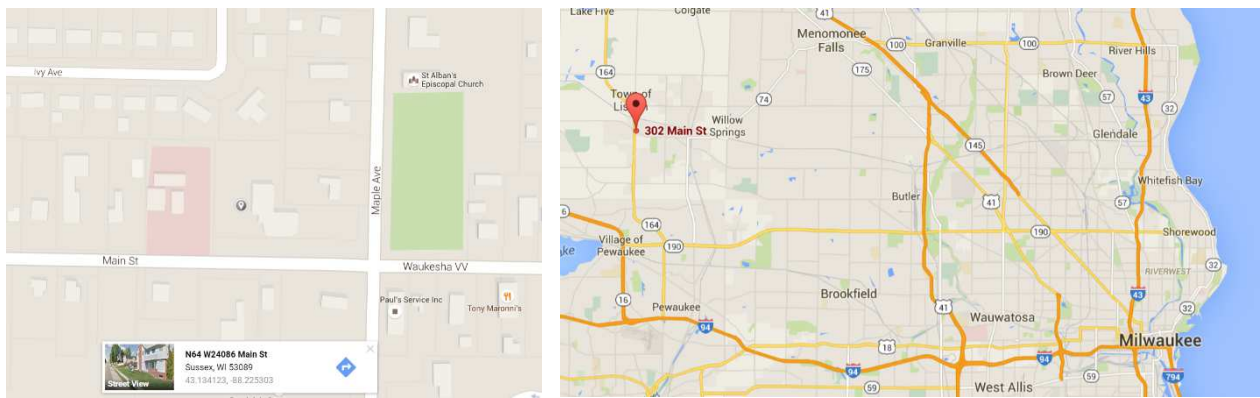


Figure 1: Maps showing the location of Prime Resources

Although the company was created very recently, its story is interesting to tell as it illustrates the spirit of American entrepreneurship that I was able to experience a lot during my internship. In 2014, John and Karen Houdek, residents of Sussex, WI, created their own company, Prime Resources. Owned by Mrs. Houdek and led by her husband, it was their desire, after spending 25 years in the industry leadership, to run their own business. The company has grown since then and answers the challenges of many other businesses who come to Prime Resources in order to find the best professionals available to solve their problems.

The company is registered as an LLC (Limited Liability Company). According to the IRS (Internal Revenue Service), an LLC company is “a business structure allowed by state statute. Each state may use different regulations, and you should check with your state if you are interested in starting a Limited Liability Company” [3].

This legal structure is the most convenient to adopt for an entrepreneur as the registration process at state level is fairly quick and does not include a lot of constraints. An owner of an LLC is called a member, and there is no limit on the number of members that can be included in an LLC. Actually, a single person can also be the sole member of his LLC, and different entities (such as other companies) can be members as well.

The type of activities a company can do as an LLC usually depends on state law, but most of the time bank and insurance companies cannot claim this status. On the other hand, this business structure is very popular for IT services start-ups.

Regarding the different work methods, Prime Resources will usually place their employees at a client's location. Most of the time, people will not work from the headquarters but from each client's site. In my case, it was a little different. Since I was working on a project that was not for a specific client but for a later release on the market as a software solution, I had the opportunity to work from different locations throughout my internship.

Although I worked from the headquarters, I also had the opportunity to work from a business lounge located in downtown Milwaukee: the Hudson Business Lounge. The Hudson is a meeting, work, and event center located in the heart of Milwaukee's Third Ward, one of the oldest parts of the city. The purpose of this location is to provide a work environment for start-ups, as well as a meet-up area with other like-minded professionals. Offering over 11,000 square feet of office space, it is a futurist concept that gives people the opportunity to grow their network, either by meeting new clients or potential employers/employees [4].

I was able to meet new people working on different IT projects. Some were just developers like me who worked on a specific project for their company, and others were managers and recruiters who hold conferences and events in order to broaden people's minds in the business world.

Working from this location influenced me deeply. It helped me better understand the world of start-up companies and the spirit of entrepreneurship that is so specific to the United States. It was also Prime Resources' goal for me to meet other entrepreneurs, as it gave me an opportunity to discuss my project and get ideas for the development.

3 Project Description

The goal of my internship at Prime Resources was to develop a simple version of a Configuration Management Database, commonly referred to as CMDB. A CMDB is a software that allows businesses to keep track of their IT infrastructure to know exactly what resources are available to the company, whether it be hardware like servers, software like applications, or even human resources like the number of database administrators.

In today's market, companies can use a CMDB in order to keep their different resources up-to-date. While not all businesses can invest time and money in such tools, most big companies like Kohl's, General Motors, or Walmart will usually invest in a CMDB so they can have a better view of all resources available in each department of the company. Although this practice is well-defined in companies' strategic development, the use of CMDBs can rapidly become overwhelming and much too time-consuming, thus making employees less inclined to update them, or even to use them at all.

Experience proved to my manager, currently a vice-president at Kohl's Corporate, that people never use the CMDBs because of their complexity and their difficulty of access. A first problem is the time required to learn how to use a CMDB. Although different systems are available on the market, they all require some training to know how to research or update a specific item in the database. Most of the time, employees only need to know about one specific function, but the complexity of the system makes it impossible to speed up the learning process or to specialize the training in a specific area of the software.

A second problem that comes to mind (and actually results from the first one) is the inaccuracies caused by the data in the CMDB. When the structure of the company changes (for example when a new device is added to the network or when a specific department is reworked), it becomes necessary to take into account these changes into the CMDB. They need to be registered in the software by either an administrator or a qualified employee who owns the rights to modify the data according to his own department.

Those changes being frequent, it can be fairly hard to keep track of them, which ultimately leads to some inaccuracies in the CMDB. Employees will then not feel they can rely on a database that could potentially be out-of-date. Thus CMDBs become seldom updated and used.

Although bigger companies can find time and money to invest in getting a CMDB and training some employees to keep it updated, small- and medium-sized companies do not have this luxury. They usually have to use simpler methods to access the different resources of their company. The principles of CMDBs could however be applied to any company, the size not being a decisive factor for a CMDB to be efficient. It is clear, though, that with the current products available on the market, those companies would not be likely to invest time and money in purchasing a license for one of these applications.

This is where the project takes its place. What if there was a simple CMDB on the market that would meet all the requirements regarding what a small- or medium-sized company could expect from a CMDB? Such an application should, of course, be simple to use and very intuitive, while enjoying the perks of having a top-notch interface and using some of the most recent technologies available on the market. This could be easily implemented if the conception and development always kept in mind the following aspect: **Minimally Viable Product (MVP)** (*see glossary*).

My task was then to develop a software that could answer most flaws described earlier. The application would be divided into three parts:

- Configuration items listing
- Applications listing
- Data center mapping

The first part would show the list of all configuration items from the company. A configuration item, as we will see later in the state of the art, is either a virtual element in the company like a server, a database, or a virtual storage service. The application should give the user the possibility to view all configuration items in the company, as well as their properties. An administrator should also be able to add new items or properties to existing configuration items. Moreover, the interface should display these items in a hierarchical tree.

The second listing should present the different applications used in the company. Each application can reference one or multiple configuration items. Thus, in this listing, there will be two parts: first, a file explorer tree that shows the hierarchy of the applications grouped into folders; second, a graph that will list all configuration items used by a specific application. As with the previous part, there will be two views: user and administrator. The user should be able to access all applications and visualize the tree. The administrator should be able to organize the different applications into folders and be able to rename, add, or delete applications. He also can add configuration items to a specific application and reorganize them into different folders.

The third part is the most ambitious in the whole project. It should present the company's different data-centers. As with the two previous points, a hierarchical tree will list all the data centers. After clicking on a specific node, the user will access a map of the data center that shows the position of all the elements included in the room. If the user is an administrator, they can modify the map by editing the room's characteristics (such as the number of rows and columns and the position of the different servers). They can also add or delete cabinets. Now, when a user clicks on a cabinet, they will be redirected to a closer view that will show them the different configuration items (servers) present on that cabinet. By clicking on it, the user will be redirected to the properties of that configuration item, as shown in the first part. It will also allow the user to see the path to all the applications that are using this configuration item.

The application will also feature other functionalities like a login and sign up page allowing users to access the application and register. A user will be able to register either as a user or administrator. This distinction will determine if the user logged in is allowed to make changes to the application.

Another feature included will be an information page showing the different settings for the connected user. The settings included will be the username, the email address, the administrator status, as well as the possibility to change the current password.

Now that I have described the problem the application is trying to solve, as well as the different requirements that it should have, I focus on the work that was done before regarding CMDDBs as well as the different software available on the market.

4 State of the Art

Configuration Management Databases, commonly referred to as CMDBs, are part of the Information Technology Infrastructure Library framework, also called ITIL. Before describing what the purpose of a CMDB is, it is important to understand what ITIL is and what its specifications are in order to better understand the big picture.

4.1 Information Technology Infrastructure Library (ITIL)

The Information Technology Infrastructure Library framework, more commonly referred to as ITIL framework, consists of different tools and practices to “help individuals and organizations use IT to realize business change, transformation, and growth” [5]. More specifically, it focuses on providing different requirements in order to help IT services to focus on business needs.

ITIL consists of a succession of procedures and tasks to be completed by the IT department of each company in order to master the model described in a set of books. The first requirements of this kind appeared in the 1980s in the United Kingdom, before being released into a major version in 2000 as ITIL v2. The current version in use as today is ITIL v3, with an update which came in July 2011.

The advantage of following this model is to help companies achieve their goals in a more efficient and less costly way, as well as to grow and develop the size and budget of the company over the years. To achieve this goal, five key points are incorporated into the ITIL specifications:

- Strategy management for IT services
- Service portfolio management
- Financial management for IT services
- Demand management
- Business relationship management

Although understanding the specific role of each process is not necessary in the continuation of this report, it is nonetheless interesting to note where CMDBs are positioned in this larger scale. The IT services strategy management includes two core points: service support and service delivery. CMDBs are part of the first one. This first service focuses on the user; it makes sure that they, whether a final consumer or a developer in the company, can have a positive experience and

will be able to ask information, request update, or notify of an issue. The CMDB should be able to process any request and, if it fails, notify a third party which will analyze the problem and likely propose a solution.

4.2 Configuration Management Database (CMDB)

Part of the ITIL process chain, a Configuration Management Database, is, as the name suggests, a database containing the references of the different assets of a company (servers, databases, networks, applications...). Each single item in this database is referred to as a configuration item. A CMDB's task is to list all these assets along with their properties, thus allowing any user in the company to know which assets are available in a specific department.

Most of the time, there are two different views in a CMDB: administrator and user. The first one must be able to manage the different items, either globally or in a specific department. That includes being able to add different configuration items in the database when the company is purchasing new hardware, but also being able to update properties on specific objects. An administrator will also grant permissions of access to certain types of users according to the department in which they work. A global administrator can also be used to manage the different administrators at each department's level.

A user, on the contrary, is only interested in viewing the contents of the database, and especially in knowing specific properties relative to his department. Most CMDBs will then grant a view permission only to certain types of items related to the user's department.

Having these two main roles in mind, it is important to note the different interests motivated by using a CMDB. On one hand, we have a user only interested in knowing certain properties on a specific configuration item. The CMDB must provide a very fast and intuitive interface so the search can be done quickly. On the other hand, there is an administrator that needs to frequently update the database and make sure each update will not have a negative impact on another configuration item. A CMDB implementation needs then to make sure each of these specifications is respected.

One might wonder what should be included in a CMDB. The answer to this question is obviously both large and subjective. Most of the time, the configuration items will be virtual objects such as a database instance, a server host, a network, a virtual storage instance, etc. Basically, it can be pretty much everything included in the company's IT infrastructure. The different human resources can be added to this as well.

Visually, the structure of a CMDB uses trees and graphs to show the different interactions between the components. For example, if an administrator wants to change a setting on a certain server, the CMDB gives them the possibility to see the different configuration items that would be impacted

by this change. Visually, this solution is most of the time rendered under a tree or graph to be read in an easier way by the user.

The ITIL specifications describe the four major tasks that a CMDB must fulfill:

- Identification of configuration items to be included in the CMDB
- Control of data to ensure that it can only be changed by authorized individuals
- Status maintenance, which involves ensuring that the current status of any configuration item is consistently recorded and being kept up to date
- Verification, through audits and reviews of the data to ensure that it is accurate [6].

Today, the market presents different implementations for CMDBs, both in private and public domain. Although CMDBs were initially being developed as commercial products requiring licensing, some open-source CMDBs started to appear on the market by the mid-2000s; the main reason explaining this phenomena was the desire for some smaller companies to customize the CMDB to their needs, a difficult and even sometimes impossible task to achieve without the help of a consultant from the developing company on commercial products. Although not being as complete as commercial applications, open-source CMDBs give certain advantages to users. Below is a description of two of the most popular open-source CMDBs [7].

The first one, Itop, is developed by Combodo, a French company. It offers a full customizable CMDB, along with a service desk, allowing users to create and access IT products in their company. The software also includes an error and incident management system allowing the administrator to keep track of the different errors in the system.

Another popular open-source CMDB is OneCMDB. Aimed at small- and medium-sized businesses, it was developed in 2006 by Lokomo Systems AB, a company based in Sweden. The version currently available is a stand-alone edition coming with source code, and released under a GNU General Public License. The code is accessible on Source Forge and requires Java installed on the system in order to function [8].

Although open-source CMDBs are free to use and more easily accessible for small companies, they still require to be installed and configured on the company's IT infrastructure. This process can take time and requires a knowledgeable administrator on the application. This is an obstacle for the company if it lacks time and qualified personnel. An alternative solution could be provided by an all cloud-based solution, which would solve the problem regarding the time spent to configure the application.

For companies who can afford it, there are also other alternatives to open-source CMDBs. In early 2015, Capterra, a website dedicated to helping companies find the best business software available

on the market, published an article entitled “Top ITSM Software Products”. For information, ITSM, or IT Service Management, refers to all the procedures in a company to manage the IT department. ITIL is part of these procedures [9]. The article pointed to Freshservice as the most popular solution used by companies. Developed by Freshdesk, a company based in San Francisco, it features a whole helpdesk system along with the basics of CMDBs described earlier. This solution is used by a number of big companies, such as Sony or Honda. As with the other commercial software, the annual licensing can get very expensive depending on the companies’ needs.

There are many other solutions on the market. Some are being developed for an internal use only, meaning that the company that is developing it will be the one to use it. That solution can be a nice alternative to buying a license to classic CMDB provider, as it will be created to fit the company’s needs. However, it is first important to show how the CMDB can be a real advantage to the company, as it will take time and money to develop it.

In brief, there are a lot of different CMDBs on the market to choose from, but most of them have the same problems: they are designed for big companies who can afford a high annual licensing cost, and they require a special training to use it as it takes time to ensure that everything is kept up to date. Thus, the complexity of the system can discourage users to take advantage of it, sometimes leading to cases where the CMDB is not kept up-to-date, or is not even used at all.

All of these different problems warrant a reconsideration of how to use and/or develop CMDBs. The product that I developed during my internship comes as an alternative solution in order to prove that a CMDB can be both simple and useful to use. I now describe the solution that I created.

5 Problem Analysis and the Solution: a Cloud-Based CMDB

In this part, I describe the different steps of the development of the cloud-based CMDB. During the first days of my internship, my manager explained to me the main flaws of CMDBs. As a vice president at Kohl's, he understood the importance of having CMDBs in the company, both from a corporate and user point of view. But the reality of the market makes it harder to find a tool that is affordable, maintainable, and easy to use, both for administrators and for users. To him, the most important point of a CMDB is summed up with three key words: Minimally Viable Product. This notion of MVP is very important to understand for the rest of this report, and was the main thing to remember as I developed the application.

Basically, a minimally viable product is, as the name suggests, a product that is viable so it can do what it is supposed to in an efficient way with as few flaws as possible, but at the same time a product that is conceived in pure simplicity, or simple enough that a user should not need a manual or an explanatory note to know what to do with it. These notions should apply both to the final product, i.e. what the user will see, and the code, so that other developers who come after can understand precisely what was done. I now describe what this implies for both point of views.

5.1 From a User Point of View

For a user, as was previously written, the solution needs to be user-friendly. This goes from the main functionalities of the product to the simplest and tiniest details such as where the option bar is located or how a user signs up in the application. While developing an application that can fulfill these requirements, there are five main points to keep in mind [10]. According to Walker Fenton, CEO of Sepia Labs, the company behind the professional social network Glassboard, these are:

- **Understanding the context of the users that will use the application**

This point may seem obvious, but too often in the corporate world engineers and developers conceive applications and keep adding features to them while losing the view of what the application was originally intended for. It is really important to ask the following questions: what is the user expecting from the application? In which circumstances will he use the application? What main functionalities is the user expecting? With this in mind, it becomes easier to target the different options to include in the application and the ones to leave along.

- **Having a clean presentation**

The size of the screen on which the application will run does not really matter, what matters is how to visually organize the different graphic elements of the application so that the screen will not be overloaded with components and functionalities. It is important to take the best advantage of the space available, while also leaving some empty room in order for the eyes of the user to ‘rest’ at some point.

- **Be intuitive**

This is probably one of the most important aspect to keep in mind. The user must not spend a lot of time looking for an obvious functionality. Everything needs to be smooth, easily accessible and understandable. If we take the example of Amazon, it does not take long for a user to know where to search for a product, how to add it to the cart, and how to checkout. Actually, this process is almost immediate. So should it be for any well-conceived application. While using the application, the question “What should I do?” should not be asked.

- **Have a great design**

Whether a clean presentation means using the space wisely and not overloading the screen with various components, a great design implies that the graphic used will not be of poor quality or too simplistic. In today’s digital age, users have higher expectations regarding graphics. While a need for high definition graphics will depend on the purpose of the application (videogames developers should obviously make sure this point is respected), it is still good to make sure that the images used are not blurry or of low-quality. It is actually better to not provide any graphics at all than to include low-quality images.

- **Responsiveness**

With the availability to run the same application on different devices (PC, iPhone, or tablet), the developer needs to make sure that the application will fit on the screen of each of these different devices while keeping a clean presentation. Depending on the application, responsiveness can sometimes be a challenge for the developer. That is why it is important to know how the application will be run and on which devices it will be used.

Ideally, these five points always need to be followed in order to provide the user with the best experience. However, the reality can sometimes be a little different from the theory. Sometimes, some of these rules will not be followed for a specific reason. But in most cases, one must ensure these properties will always be implemented in any application made available on the market. From a developer point of view, there are also some aspects to consider in order to produce a minimally viable product of good quality, as described in the next part.

5.2 From a Developer Point of View

On the developer side, the minimally viable product implies that all the code that is being developed is essential to the core functionalities of the application. In other words, code that is used to prettify the application in terms of functionalities or design should not be written. As we previously saw, the application must of course be well-designed and intuitive, but this can be done while keeping code at a minimum quantity.

In terms of coding, it can be achieved by keeping simple functions that should be reused as often as possible. For each functionality, it is important to consider if something that has been previously written in the application can be reused and slightly altered to fit the new use case. This will allow to reuse as much code as possible, keeping a rather small number of lines in each file.

The other advantage of reusing some of the same material is to allow the development to go much faster. This is a key point in a minimally viable product. Since the final product is more of a prototype than a final version, the development hours should be shortened so the product can be released sooner and thus get the first reviews in order to add new features to the product for a final version.

Testing is also affected: tests should be kept to a minimal number, sometimes the test consisting of checking that the application is just working well. Unit testing is not encouraged as it requires time. The only concern a developer needs to have is that the different functionalities are working and producing the desirable effect. If bugs are later discovered, they can just be fixed in the future releases of the product.

Whenever a developer needs to add a feature or modify the code, there is one question that they should consider: will the modifications affect the product in a way that the user will want it more? If the answer to that question is no, then the changes should not be made. Indeed, it is first necessary to prove that the basic concept carried by the minimally viable product can be sold on the market before improving it.

5.3 Why a Minimally Viable Product

One might wonder why I chose to develop a minimally viable product instead of a classic application which incorporates test cases and advanced functionalities, especially when developing a MVP is the exact opposite of what most companies expect. Usually, a company will keep asking more features to the product, thus leading to a final application that can sometimes double the size of the one initially intended.

The reason is that this cloud-based CMDB is not designed for a specific company; actually, the product was not the request of a client, or even a group of clients. It came up as an idea and, like every idea in the business world, needed to be verified.

Selling an MVP is more about selling a concept or a prototype than a real product. The product, though conceived for users, will not be sold to users, but to investors, IT managers, and other businessmen who would welcome a solution to help them be more productive in their company. Steve Blank, a manager who specialized in start-up management methods, puts it in this way: *“You’re selling the vision and delivering the minimum feature set to visionaries not everyone”* [11].

That is a key concept to keep in mind for the rest of this paper. The cloud-based CMDB that I developed is not only an application, but a new vision of using CMDBs that has not been tried before. The goal is then to write a solution with all the functionalities needed to prove to potential investors the efficiency and potential of the solution. In the case where the product is not convincing enough, then time will not have been wasted in developing something unsuccessful.

As stated earlier, developing a MVP is not common for most companies, but it can be an advantage for start-ups, especially while looking for new investors. Those investors, who could also be called “earlyvangelists” for early adopter and internal evangelist, are people with five major concerns that can be schemed by the pyramid represented on Figure 2 [11].



Figure 2: Earlyvangelist pyramid

The pyramid addresses the concerns of these potential investors:

- They have a problem.
- They understand they have a problem.
- They are actively searching for a solution and have a timetable for finding it.

- The problem is painful enough that they have cobbled together an interim solution.
- They have, or can quickly acquire dollars to purchase the product to solve their problem.

These people should be the target of the product, because they have the means to provide financial support to help the company pursue its development. They do not necessarily care if the product is ready to be sold, but care more about what the different available functions are, because they can see the potential the product can have on the market.

Let's take a well-known example in the business world: Facebook. At its beginning, it was just a simple social network that allowed people to access photos and basic information on people registered on the network. It was just a minimally viable product: it allowed people to be on the same network and share data. Rapidly, though, investors started being interested in the product and using it. At first it was mostly universities and college campuses, then the network expanded into the corporate world, until finally being able to reach everybody aged 13 years old and older with a valid email address. During all this evolution, developers kept adding features to the social network because it worked well. Investors were convinced, so they sponsored the product, thus allowing Facebook to grow and become the billion-user network we know today [12].

This example proves the importance of a MVP. Before investing time and money in a product, it is first necessary to check if the concept can work and find sponsors to allow its growth into a larger application.

5.4 Answering the Flaws of Enterprise CMDBs

As I have previously discussed, CMDBs available on the market are hard to keep up-to-date and, as a result, employees do not rely on them. The only way to fix that is to create a simple application that people will want to use so it can stay updated most of the time. There are, however, two important issues.

First, it is evident that a minimally viable product will never replace an enterprise CMDB published by Oracle. Even if features would progressively be added to the product, creating an enterprise CMDB will require time and money. Plus, if the solution was as ambitious as most software found on the market, the application will lose simplicity and we would end up with another “big” CMDB which would have the same flaws that were supposed to be fixed.

Second, according to the size of the company, having a CMDB is sometimes not an option. Most big companies will require one, but this implies having an enterprise solution as described in the previous paragraph. Regarding smaller companies, a CMDB could be really useful but such

companies obviously will not need an enterprise application like bigger companies do, so they usually end up with no CMDB at all.

With the application, the main focus was to answer these two flaws with two strategies targeting both environments. First, we decided to primarily target small- and medium-sized companies as potential customers for our CMDB. Such companies are looking for something simple, which perfectly described the kind of application we want to give them. They just want to focus on the core functionalities of a CMDB, so this describes perfectly the minimally viable product.

The other target could be units and departments of bigger companies, and even employees. While we do not expect huge businesses to replace their complex CMDB by a simple prototype, the solution can be advertised as a local alternative in a smaller scale and be used in different departments. Instead of representing the structure of the whole company, the CMDB could simply show the architecture of a specific department and its subcomponents. Thus, keeping the CMDB up-to-date will not require a lot of time, and employees will be able to know exactly what the different interactions between configuration items are.

This strategy of targeting two types of users with different goals could actually work if it helps people become more efficient and productive. But for this to be done, it is important to develop the solution using the latest technologies to help ensure a smooth experience for the user.

5.5 The Importance of Cloud-Based Applications

After targeting the audience for the application, it was important to determine which platform would support the application. Originally, my manager proposed a Java application working with Google App Engine. The advantage of this solution would be the opportunity to run the solution on any operating system. This solution would have followed OneCMDB, an open-source CMDB that was developed in Java.

When I started to select all the tools I would need to start the Java development, I encountered an issue with the Java version. Google App Engine did not support the latest Java version (Java 8) and it was necessary to develop the application using Java 7. Although it was not a problem to download and install that version, it made me think of another problem I encountered when I tested OneCMDB, the open-source application written in Java. This solution offered a visualization of the configuration items that I could not try on my computer, because the Java plug-in was made unavailable in the latest version of Google Chrome for security issues [13] [14]. In order for it to work, it was necessary to use a previous version of Chrome.

This little problem made me realize how crucial compatibility problems are in applications. It would be very unfortunate if users could not use the product just because the technologies used encounter compatibility problems or are outdated. Using another browser could have been a

solution, but if we take a look at the statistics given by the Digital Analytics Program (DAP) from the Federal Government, we realize that Chrome is the most used browser, all devices included, with 34.7% of all visitors [15]. This means that it was crucial to make the solution work on Chrome if we wanted to keep using the same technologies.

Intra-browsers compatibility can be one of the toughest parts of web development. Indeed, it is important to ensure that the application being developed will be working with the majority of browsers on the market. Figure 3 shows the market shares among browsers in the United States. It is interesting to note that Chrome is the most used one today, with a difference of 14 points with its main competitor, Internet Explorer, as of September 4, 2015 [16].

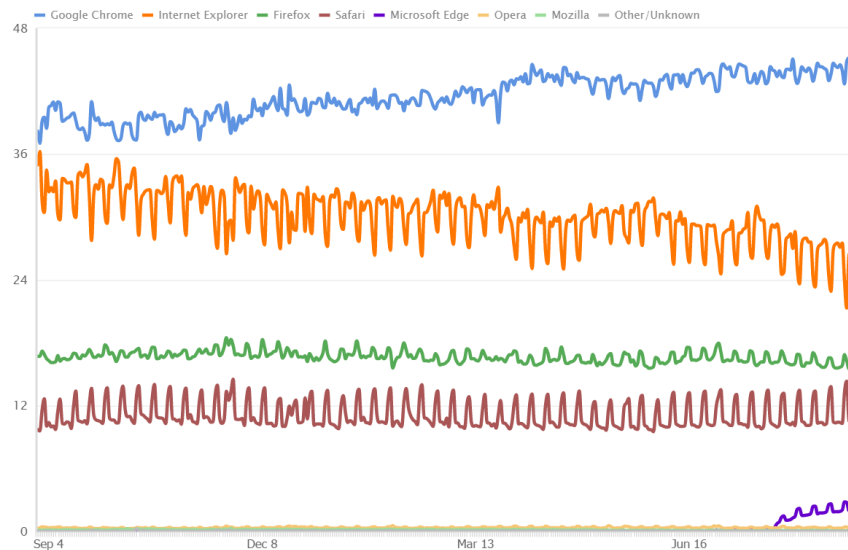


Figure 3: Web browsers market shares in the United States

This schema shows the importance of taking into account the cross-browser compatibility during the development. Although Chrome is the most used browser, IE, Firefox, and Safari are nonetheless widely used and it is necessary to make sure the application that will be developed will be compatible on these browsers as well.

In our original plan, although most part of the application would be for desktop, managing the database had to be done through Google App Engine, so using a browser was required anyways. The other consideration I had were the restrictions offered by a desktop application. The original application, as imagined by my manager and I, would be run only on a desktop computer. That could dramatically limit how the application is used.

According to the Pew Research Center, nearly two-thirds of Americans own a smartphone [17]. This does not only mean that there is an important market available regarding smartphone applications, but it also implies that Americans will most likely use their smartphone more frequently than they would use their laptop, due to having permanent access to their phone. The report also shows also that people use their smartphones to do much more than just accessing

Facebook or reading emails; for example they also use it for online banking or accessing different services related to their businesses. Thus, an application that would be available to both desktop computers and smartphones could be a real advantage for the popularity of the CMDB and for the user's convenience.

However, writing a desktop application is not the same than writing a smartphone application. Also, the compatibility problems between operating systems would still be there. A solution that appeared to me was to create a cloud-based application. It was a perfect compromise, both for the developer and the user. First, there is only one main application to develop, without worrying about which operating system or type of device will run the application. Second, it will be much faster to use. Basically, a user (here the concept of user can also refer to a company) will just need to register and will be ready to use the application, as there will be no need to download an installer and configure the application locally.

If we get into detail, we can also see the advantages of using cloud computing compared to a more standard way of developing applications. Figure 4 presents ten of these advantages [18].



Figure 4: The 10 benefits of cloud computing

- **Flexibility:** the demand in bandwidth can be satisfied really easily, so there is no worries to have about a potential system overflow
- **Disaster recovery:** since everything is stored on the cloud on a database, there is no need to invest time and money into complex recovery modes
- **Automatic software updates:** the user will never have to update any software since all updates will be done by the cloud provider

- **Cap-ex free:** there is no need for capital expenditure, since services are usually pay-as-you-go. This way, it allows users to know exactly how much it will cost per month. It is also more accessible to start-ups.
- **Increased collaboration:** no matter where the different employees are, they can synchronize their work at any time, allowing updates to be released faster. It also contributes to multi-tasking on a same project.
- **Work from anywhere:** all that is needed is an Internet access and then users can log in to their accounts and start working, no matter where they are in the world.
- **Document control:** all documents are located in a central spot. Thus, if employees are not necessarily working in the same state or time zone, there is no worries to have about sending the documents by email. They will be automatically updated.
- **Security:** if a laptop gets lost, there is no worries about the data on the computer since they are stored on the cloud. And since most cloud accounts will require a password, a theft will not be able to access the cloud account of the user, and thus will fail in retrieving any useful data from the company.
- **Competitiveness:** cloud-computing makes companies more competitive and more dynamic, providing smaller companies the same advantage of access to the cloud than bigger companies.
- **Environmentally friendly:** since cloud computing only uses the space needed for the applications to run, the carbon footprint is decreased. According to Salesforce, there is at least 30% less of energy consumptions than using on-site servers.

These different advantages show the importance that cloud-computing plays in today's IT world. It requires companies to rethink how they conceive and use their products in order to stay competitive on the market while saving money and eventually increasing their income.

After seeing all the advantages of cloud computing, and after approval from my supervisor, I decided to head in that direction and create a cloud-based CMDB. Historically, this would be the first of its kind, since enterprise solutions were too complex to transfer to the cloud. The advantage of having only an MVP to develop gave me more freedom for choosing the platforms and the technologies involved. Let's now see which technologies were used for developing the product.

6 Used Technologies

In this part, I present the different tools that I used to implement the solution.

6.1 Programming Languages

The project did not have specifications regarding a particular language to use. My manager also gave me the opportunity to choose the language I was the most comfortable developing in. While doing web development, there are a lot of possibilities to choose from. Obviously, web pages would be written in HTML5 and styled with CSS3, using the framework Bootstrap for a better design. Regarding the server side, I would use PHP to communicate between the application and the server.

Regarding the different graphic elements and their interactions with the user, the code would be written in JavaScript and jQuery. This was chosen because of the tools that I found to represent the data from the server. Each of these tools will be explained in subsequent paragraphs. Since they were written in JavaScript, it would be easier to write the code in the same language. I also used jQuery to dynamically interact with some HTML elements, and also to asynchronously load different parts of the webpages, thus providing a more user-friendly interface. Finally, I used a MySQL database with the SQL query language.

6.2 XAMPP

XAMPP is an Apache distribution of a full PHP development environment. It includes an Apache server, a MySQL database, FileZilla, Mercury, and Tomcat. For the project, I only used the Apache server to run the webserver, and the MySQL database. I chose XAMPP because it is a pretty useful tool; indeed, there is no need to configure a server and a database separately since XAMPP provides both. The configuration time is non-significant: all that is required is to download the installer and follow the instructions provided. Also, it is open-source, which dismisses any cost that a commercial license could have [19].

6.3 phpMyAdmin

Written in PHP as the name suggests, phpMyAdmin is a free MySQL database that provides a web interface to manage the different tables. One of the most popular MySQL databases, it is included in the XAMPP development environment, thus being simpler to handle. It includes a lot of features, like administering multiple servers, global search in a specific database or in different subsets, data import and export, and much more [20].

Although I was free regarding the choice of my development tools, my manager and I were aware that, according to the data model of the application, a relational database needed to be used. Indeed, there are many interactions between the different tables, as we will see in the next part, so only a relational database could provide the best storing options.

The choice of a MySQL database was made because of its license type. Being open-source, it is easier to use for developing an application prototype because it is costless for the company. Moreover, installing a standard Oracle SQL database requires both time and money, and is something much too ambitious regarding the purpose of the application being developed.

6.4 jsTree

jsTree is an open-source jQuery plugin providing interactive trees to include in web pages. The trees provided have hierarchical views and can be customized with personal icons. It also supports JSON data sources and AJAX loading. The project is still active on GitHub and involves a community of 24 contributors. Currently, the stable version is 3.2.1 [21].

I needed a tree to display the different configuration items into folders, and that plugin was exactly what I was looking for. It offers a great number of functionalities and allows the developer to incorporate his own functions regarding the different interactions available with the tree. It also has built-in functions such as renaming, creating, deleting, or selecting nodes, which can be easily called to handle events with the user. The API documentation is also plentiful.

6.5 Vis.js

Vis.js is a visualization library that provides different interfaces to interact with dynamic data. The library includes different components allowing to visualize data on different representations. It includes interactive 2D and 3D graphs, different network views used to show the interdependence in a graph, a timeline allowing the creation of events at specific dates, and a data set, a data structure

to format the data imported into the different components. The Vis library is a JavaScript component compatible with all modern browsers and allows the user to dynamically interact with the data. It is an open-source project available on GitHub and is still being developed [22].

Using the library is fairly easy; the developer includes it in his work repository, references it in the HTML page he wants to use it on, and gives a specific id to the HTML component that they wish to integrate the visualization. Vis.js being developed in JavaScript, the different functions to interact with the library need to be written in JavaScript as well.

6.6 Aptana Studio

Aptana Studio is an open-source web development IDE. It provides support for many different languages, including HTML, CSS, PHP, JavaScript, jQuery, Python, Ruby, and Rails. Considered the world's most powerful open-source web development IDE, it offers an Eclipse-like interface as well as different tools allowing the developer to be more productive. The software is being developed by the company Appcelerator. The current version is Aptana Studio 3 [23].

Before choosing an IDE, I wanted to make sure that it would offer support for all the languages I needed, but I also wanted something that would not be too cumbersome to use. For example, I could have used Visual Studio to develop the application, because that IDE also supports web development. However, it would not have been necessary to use such a complex tool if there was an alternative.

The other reason for choosing Aptana is the open-source license. It is easier and cheaper to work with open-source licenses than to purchase a Microsoft license. Since my project is for a company and not for myself or a school, I could not have used my Visual Studio student license to write the application, since by law a professional license would have been required.

All these different tools were crucial in developing the application. I now enter into the core part of this report, namely how the solution to the initial problem was implemented.

7 Implementation of the Solution

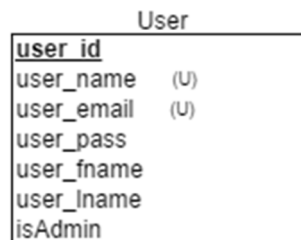
In this part, I present the implementation process of the cloud-based CMDB. Beforehand, it is necessary to mention that the first month of my internship was spent in research and brainstorming with my supervisor in order for me to better understand the problem, master the aspects that a CMDB should include, and have the vision on how to develop the product with choosing the right technologies, as I described earlier in this report.

This step was essential to start the development of the application in the best possible conditions. Once it was completed, my first task was to focus on writing the data model for the application.

7.1 Data Model

The first step in the development process was to write a data model for the database. This would be the building block of the application. It was very important to keep this data model simple in order to follow the specification of a minimally viable product. The following two figures show the different interactions between the tables of the model.

Figure 5 represents the user table. The first step of the application is to register users. A new user will be prompted to register when he accesses the application for the first time. There are two different views: user and administrator, represented in the table with the column *isAdmin* (0 for user, 1 for administrator). Although a generated id is used for the primary key, the unique identifiers used during the authentication process are the email address and the user name. These two fields are unique according to the user, and interchangeable during the login process (this means that one can either use his/her email address or username to log in to the application). Regarding the password, it is encrypted (cf. 7.2. on user administration).



The diagram shows a rectangular box representing a database table. Above the box, the word "User" is centered. Inside the box, the following fields are listed from top to bottom: "user_id" (underlined), "user_name (U)", "user_email (U)", "user_pass", "user_fname", "user_lname", and "isAdmin".

User	
<u>user_id</u>	
user_name	(U)
user_email	(U)
user_pass	
user_fname	
user_lname	
isAdmin	

Figure 5: User table

Regarding the application data, Figure 6 shows the interaction between the different tables used to manage the configuration items and their properties. There are three main parts of this diagram:

the first one focuses on managing configuration items, as can be found in the first part of the application; the second group manages the data centers; and finally, the last group of tables is used to represent the application graphs. Each of these groups will be described in the following paragraphs.

First, let's talk about the tables used to manage the configuration items. Each configuration item is contained within a class. For a reminder, a configuration item is viewed as a specific item within the company, for example a Linux server with a certain hostname. In order to organize them, similar configuration items are placed into a class if they share similar properties. For example, all the servers will be included in the class Server. A class can have a parent class as well; if we take the class Server, it can have two children classes: Linux Server and Windows Server. The identifier for each parent's class is saved in the child class table; thus, knowing a specific configuration item, it is easy to get the whole set of classes in which the item is included in. It is important to note that a configuration item cannot have children. To understand that, we can view a class as a folder and a configuration item as a file.

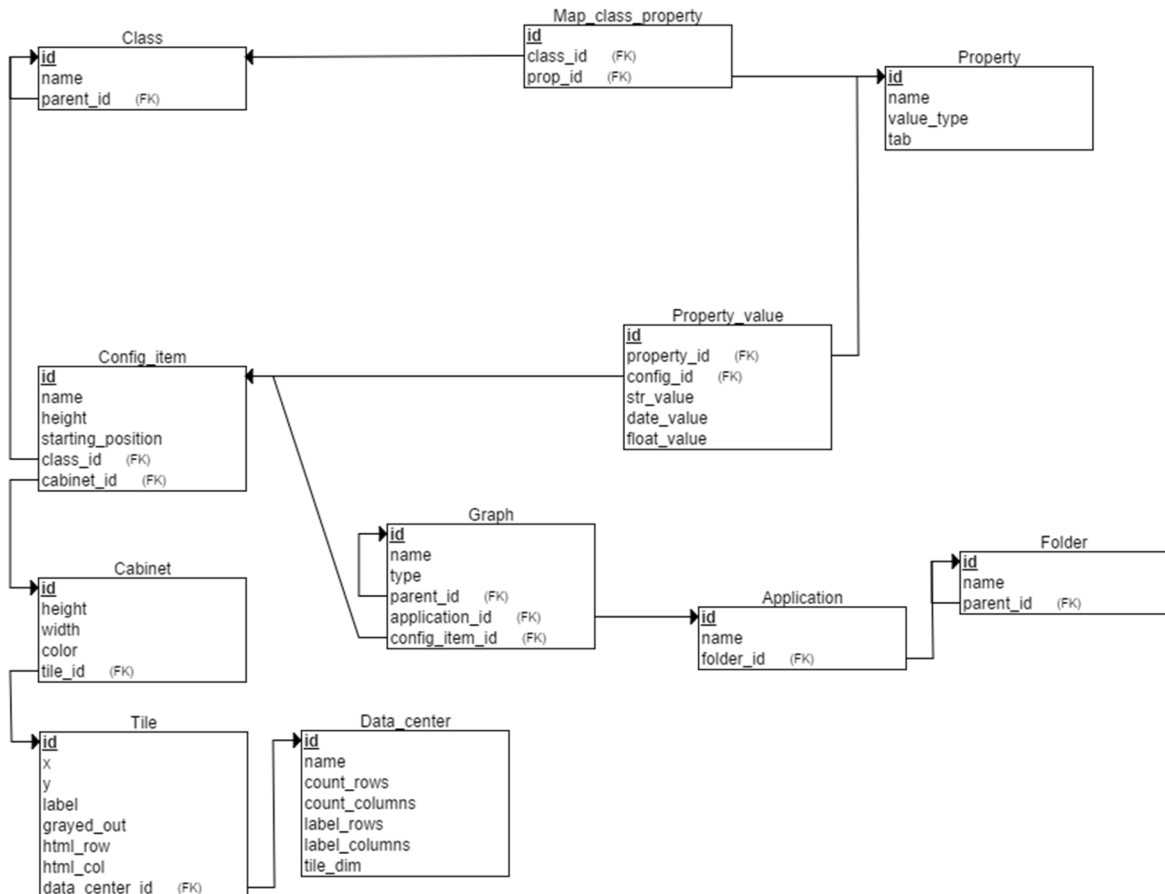


Figure 6: Data model

Classes contain properties as well. Each configuration item in a class will inherit its parents' properties. Since the same property can be used by multiple classes that are not linked together, it was necessary to place the columns in a different table, the *Property* table, mapped to a specific class through the *Map_class_property* table in order to know all the properties referenced to a specific class.

A property is defined by a unique id and a name known to the user (like the other tables' id, the user will never access the id unless he goes directly to the database). Each property also has a value type that can be a float, a string, or a date. The last field determines which type the property is assigned to. Indeed, properties can be of three different types: general (for example a hostname), financial (like the maintenance cost), or labor (like the number of employees using the configuration item).

Since the same property can be used with multiple classes, there cannot be a value field in the property table, otherwise all properties would have the same value no matter what class it would be assigned to. This calls for another table, *Property_value*, which links the value to its property and to the configuration item that will receive the value. To store it, the table includes three different columns according to the property type. When a new value is created, it is inserted in its corresponding type field and the two other columns are set to NULL. The choice of having three columns instead of a single one was done regarding the data format. Each field has a special input format specific to its value type that was necessary to follow when retrieving the value to load it onto the web page. All these tables are used to display the configuration items properties on the first panel in the application.

The second part of the data model focuses on the data centers. The specifications of the project included the possibility to manage views of the different data centers used in the company. That is what the tables *Data_center*, *Tile*, and *Cabinet* are for.

In real life, a room where a data center is located is divided into tiles. Each tile is identified by a specific row and column that allows someone to know precisely where a specific server is located. The table *Data_center* will record the properties of the room, *Tile* will, as the name suggests, record the properties of the tile, and the table associated to a cabinet will represent the physical object positioned in the data center. It can be a server or something different; for example, ventilation systems can be represented in the room as well.

The first table specifies the properties of the data center with a number of rows and columns, and a name. The labels are used to know which letter or number will start the columns or rows. Most of the time, it will be a letter and a number like A1. Sometimes though, it can be different. Some rooms can use either two letters and a number (like AA1) or numbers not starting at 1 (like A30). It is then essential to give the user the opportunity to choose this start label to match the reality. The last column specifies the dimension of the tiles. Since all tiles are squared, only one field is needed for the dimensions.

The *Tile* table has a certain number of parameters, all essential to get all the important information to the application. The coordinates (x, y) will give the exact position of the tile, taking into account the tile dimensions. It means that, if a tile is 2x2, the coordinates of the first tile on the upper left will be x=2 and y=2. These columns ought not to be confused with *html_row* and *html_col*, which are the coordinates used by the application to locate the cells. In the previous example, the upper left cell parameters would be *html_row*=1 and *html_col*=1. The label will use the room notation system as defined in the *Data_center* table. If *label_rows*=1 and *label_columns*=A, then for that same tile we would have *label*=A1. Lastly, the *grayed_out* column specifies if the tile is accessible, meaning if a cabinet can be positioned on the cell. The values possible are either 0 or 1, the first case being if the tile can be used.

The *Cabinet* table represents a component positioned on a tile. Most of the time it will be a server, but it can also be an air conditioning cabinet or something else as well. The property *height* will indicate the height of the cabinet in number of units. This means that it will indicate the maximum number of racks possible to insert in the cabinet instead of a standard dimension in feet and inches. Typically, there are 48 units in a cabinet. The choice of doing so was justified by simplicity; instead of calculating what the dimensions of the cabinet are, the number of units will be simpler to process to position a server on the racks (or, according to the data model, a configuration item on the cabinet). The *width* property is only for information, as it will not be rendered on the grid. Regarding *color*, each cabinet will have its tile styled with the color specified. When the user will have a closer view of the cabinet, its borders will be colored as well. Finally, *tile_id* references the tile on which the cabinet is located.

If we go back to the *config_item* table, there are some parameters that have not been explained before. They are useful only in the data center part of the application. The first important parameter, *cabinet_id*, is the reference to a cabinet. The system must know which configuration items are saved on a specific server. Thus, a user, knowing the reference to a specific cabinet, will also be able to get the references to all the configuration items saved on that server.

The other parameters specific to the data center part are *height* and *starting_position*. The first one saves the number of units that are necessary to contain the item on the racks. The second one specifies at which first rack the configuration item is located. Thus, knowing the number of units of the cabinet, it is easy to display the item accordingly on the racks. This ensures a closer representation of reality regarding the rendering of a data center.

Finally, the last part of the data model represents the different applications used within the company. An application can either be a software purchased by the company (like Oracle Application Express), or a service website with an annual fee (like Microsoft Azure). Each application incorporates a network graph which shows the interactions between an application and the configuration items using it. Each configuration item can be used by one application or more, and the user can organize them into folders in the graph.

As with the other tables, each application is identified by a unique identifier automatically generated upon insertion in the database. It is not known to the user. Applications are grouped into folders according to their type (for example all IT applications will be grouped inside the same folder). The graph table is used to represent the network. The architecture of this graph is interesting because of the way it is built. A graph is none other than a node with a reference to its parent and a type, which can be an application, a folder, or a configuration item. It is important to note that the folder in the graph is totally different from the *Folder* table of the data model.

The root node is the application, so its *parent_id* is set to NULL. Then, the user can add a node, which will create a new Graph object with a reference to the previous node in the *parent_id* column. The graph is then built on this same pattern. The nodes added after the root node can either be folders or configuration items. In the latter, a reference to the item is included in the column *config_item_id*. Thus, when it is deleted in the configuration items panel, the graph node will automatically be deleted.

Adding nodes this way makes it very easy to maintain a simple data model that is efficient with less complexity. It is important to note that only the root node can be an application, and that a configuration item cannot have children nodes. Now that I have described the different components of the data model, I can go deeper into the different functions of the cloud-based CMDB while looking at the code of the core functions.

7.2 User Administration

One of the first functions to implement were the ones related to the user administration. It was important to have the two different views at first in order to know exactly what view would be available to the user and the administrator. Obviously, security was a key factor in order to make sure maleficent users could not steal passwords or other information. When accessing the application for the first time, a user will be directed to the login screen showed on Figure 7.

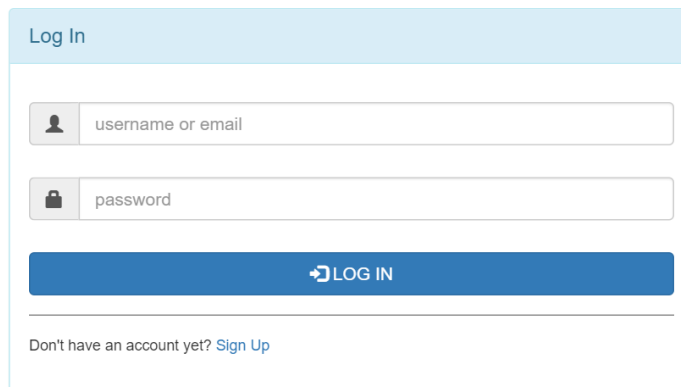
The image shows a login screen with a light blue header containing the text "Log In". Below the header are two input fields: the first is labeled "username or email" with a person icon, and the second is labeled "password" with a lock icon. Below these fields is a blue button with a right-pointing arrow and the text "LOG IN". At the bottom of the form, there is a link that says "Don't have an account yet? Sign Up".

Figure 7: Login screen

If the user is not registered, he can click on sign up and will be redirected to the following page.

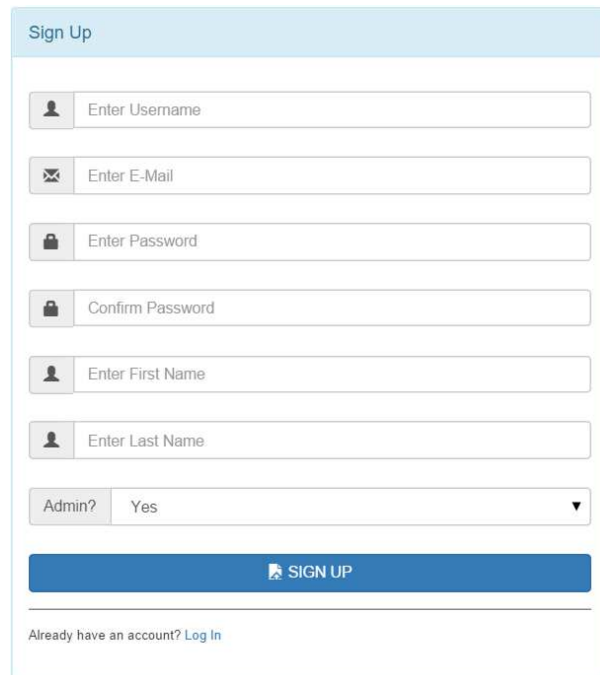
A sign-up form with a light blue header containing the text "Sign Up". Below the header are seven input fields, each with a small icon on the left and placeholder text: "Enter Username" (person icon), "Enter E-Mail" (envelope icon), "Enter Password" (lock icon), "Confirm Password" (lock icon), "Enter First Name" (person icon), "Enter Last Name" (person icon), and "Admin?" (dropdown arrow icon) with "Yes" selected. Below these fields is a large blue button with a white mouse cursor icon and the text "SIGN UP". At the bottom, there is a link that says "Already have an account? Log In".

Figure 8: Sign up screen

This asks for the user's basic information as most websites would do. There is a password check field in order to make sure that the user did not make any spelling mistakes in his password when creating it. The type of user is determined with the admin field.

Each field, as it is with the login page as well, is checked to make sure that what was input in the field corresponds to the type of data it should contain. This allows the server to not get useless queries that could overload it.

Once the information entered matches the data type and the different specifications (for example, an email address matches a specific format), the data is sent to the server and processed according to the function presented on Listing 1.

Listing 1: *Register* function

```
public function register($fname, $lname, $uname, $umail, $upass, $isAdmin) {
    try {
        $new_password = password_hash($upass, PASSWORD_DEFAULT);
        $stmt = $this->db->prepare("INSERT INTO
user(user_name,user_email,user_pass, user_fname, user_lname, isAdmin)
VALUES(:uname, :umail, :upass,
:fname, :lname, :isAdmin)");
        $stmt->bindParam(":uname", $uname);
        $stmt->bindParam(":umail", $umail);
        $stmt->bindParam(":upass", $new_password);
        $stmt->bindParam(":fname", $fname);
        $stmt->bindParam(":lname", $lname);
        $stmt->bindParam(":isAdmin", $isAdmin);
        $stmt->execute();
        return $stmt;
    } catch(PDOException $e) {
        echo $e->getMessage();
    }
}
```

Note the use of the PHP function *bindParam*. It is used to create **prepared statements** (see *glossary*). Doing this has two advantages:

- It can execute the same query with different sets of parameters with a higher efficiency
- It helps preventing **SQL injections** (see *glossary and addendum I for more information on SQL injections and how to avoid them*)

Similar techniques were used during the whole project to make sure any inputs from the user was securely processed before being sent to the server, thus making the application free from any SQL injections.

An important notion to consider in the user administration is how passwords are managed. For security reasons, they cannot be plainly recorded in the database. The idea is to apply a hashing function and record the hash value in the database. In Listing 1, this is done with the function *password_hash* that takes two parameters: the password entered by the user and the hashing algorithm. PHP provides different algorithms; the one used here is the *bcrypt* algorithm (see *addendum II for more information regarding password hash and the bcrypt algorithm*).

Once the user has successfully registered, he can then log in to the application using the screen as previously shown on Figure 7. The PHP login function included in Listing 2 will query the database to know if a user with either the email address or the username entered by the user exists. Then, the built-in PHP function *password_verify* makes sure that the given password matches the hash recorded in the database. All information needed to verify the password are included in the function, which makes things easier for the developer.

Listing 2: Login function

```
public function login($uname, $umail, $upass) {
    try {
        $stmt = $this->db->prepare("SELECT * FROM user WHERE
user_name=:uname OR user_email=:umail LIMIT 1");
        $stmt->execute(array(':uname' => $uname, ':umail' => $umail));
        $userRow = $stmt->fetch(PDO::FETCH_ASSOC);
        if ($stmt->rowCount() > 0) {
            if (password_verify($upass, $userRow['user_pass'])) {
                $_SESSION['user_session'] = $userRow['user_id'];
                return true;
            } else {
                return false;
            }
        }
    }
    catch(PDOException $e) {
        echo $e->getMessage();
    }
}
```

Once a user is logged in to the application, a session is created for that specific user, which will allow the application to know which users are currently logged in to the application. Thus, the system can manage the different authorizations, because not all users will have the same access privilege to certain web pages. This is important to make sure that a page reserved to an administrator cannot be accessed by a simple user, and vice versa. In order to do that, each page will include at its beginning a few lines to:

- Check that the user viewing the page is logged in
- Make sure the user logged in has the right to view the page

Listing 3 shows how the application first checks that the user is logged in, then retrieves his information from the database. Once these information are retrieved, the permission of the user is known, allowing the system to redirect him to another page if the visibility of the current page is not intended for type *user*. This code is included at the beginning of the administrator configuration item page. Each page has a similar heading as well.

Listing 3: Code to retrieve the user's information

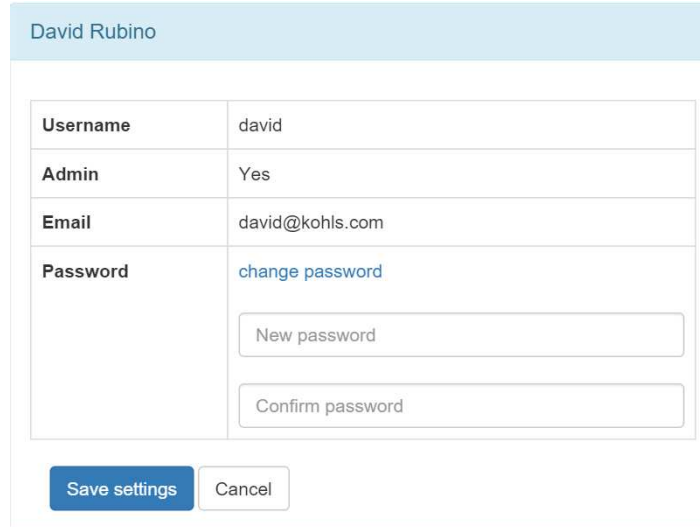
```
if (!$user -> is_loggedin()) {
    $user -> redirect('login.php');
}

$user_id = $_SESSION['user_session'];
$stmt = $DB_con -> prepare("SELECT * FROM user WHERE user_id=:user_id");
$stmt -> execute(array(":user_id" => $user_id));
$userRow = $stmt -> fetch(PDO::FETCH_ASSOC);

$permission = $userRow['isAdmin'];

if ($permission == 0) {
    $user -> redirect('ci.php');
}
```

The last feature necessary for a smooth user management is the ability to change passwords. It is the only parameter that can be directly changed by the user. The choice of not allowing the user to change anything else was done regarding the notion of MVP. I thought it was not necessary since the email address or the username will rarely be changed. The user cannot change the administrator rights as well, but this makes sense because, if he is a simple user, he will need the authorization from the database administrator in order to be promoted and access more restricted information. While on the interface, a menu will allow the user to access his personal information where he can modify his password, as shown on Figure 9.



David Rubino	
Username	david
Admin	Yes
Email	david@kohls.com
Password	change password
	<input type="text" value="New password"/>
	<input type="text" value="Confirm password"/>
<input type="button" value="Save settings"/> <input type="button" value="Cancel"/>	

Figure 9: Account information page

The data sent by the user to the server is processed with a PHP function similar to the *login* function seen earlier. As seen previously, the new password is hashed before being recorded in the database. This concludes the part regarding users' administration. Now, let's study the interaction between the application and the database.

7.3 Interaction with the Database

In this part, I present how the application is interacting with the MySQL database, which is phpMyAdmin in this case.

7.3.1 Connection to the Database

In PHP, there are two ways to communicate with a database: PDO or MySQLi. One of the main differences between these two is that PDO supports twelve different database drivers, whether MySQLi only support MySQL driver [24]. Although the application will be running with a MySQL database, what if a future client uses an Oracle database? The communication protocol would then need to be rewritten with PDO to match the client's database. Another advantage of PDO is the possibility to write prepared statements. As we have seen in the previous part, prepared statements are more efficient and essential for security purposes. For these two reasons I adopted PDO as the communication protocol in the application.

To start communicating with the database in PDO, the first thing to do is to open a session (as shown on Listing 4). This is done when the user first accesses the application (accessing the application does not mean logging in, but simply going to the login or signup webpage).

Listing 4: Function to connect to the database

```
<?php
session_start();

$DB_host =
$DB_user =
$DB_pass =
$DB_name =

try {
    $DB_con = new PDO("mysql:host={$DB_host};dbname={$DB_name}", $DB_user,
$DB_pass);
    $DB_con -> setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
    $DB_con -> setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch(PDOException $e) {
    echo $e -> getMessage();
}

include_once 'user.php';
$user = new USER($DB_con);
?>
```


First, a session is created with the function *session_start*. The database parameters are then given to the system (they are kept confidential here for security purposes) and used to create a PDO object that will represent the connection. Then, the user file containing the different functions previously seen (login, signup...) is included, allowing the creation of a user based on the current connection. This simple script allows a user to connect to the database, either by creating an account or logging in.

7.3.2 User Session

Once a user is logged in, the application needs to remember who they are when they access the different pages. On each web page of the application, the system needs to make sure that:

- The user has been identified
- The user has the permission to access the current web page

For this, it is necessary to create a session per user to grant them access to the different parts of the website. This is done by inserting the code included on Listing 5 on each page.

Listing 5: Code to create a session for the user

```
<?php
include_once 'db_connect.php';

if (!$user -> is_loggedin()) {
    $user -> redirect('login.php');
}

$user_id = $_SESSION['user_session'];
$stmt = $DB_con -> prepare("SELECT * FROM user WHERE user_id=:user_id");
$stmt -> execute(array(":user_id" => $user_id));
$userRow = $stmt -> fetch(PDO::FETCH_ASSOC);

$permission = $userRow['isAdmin'];

if ($permission == 0) {
    $user -> redirect('ci.php');
}

?>
```

If the user is not logged in, they are redirected to the login page from which they can either enter their credentials or create a new account. Once they are logged in, a session is created, allowing them to stay connected on the website as long as they are active. After a certain period of inactivity, they are automatically logged out. That security measure helps to prevent unauthorized account access in case the user, for some reason, leaves their computer while accessing the website.

The application also checks the permission. Each part of the website comes with two different pages: one for the user, and one for the administrator. Thus, according to the permission, the user will be redirected to the correct page.

The choice of having two views for each page was done for simplicity reasons. Administrator pages call for numerous functions to process the different events and interactions with the database. Adding different events to manage the scenarios whether the user is administrator or not would have complicated the code, so I decided to separate the two processing views. They will be discussed in the continuation of this report.

7.3.3 Retrieving Data with Ajax

In order to load data from the database into the application, Ajax is a common tool used. It is a client-side script that communicates from the database to the application and vice-versa. There are four main advantages to using this technique:

- **Asynchronous calls:** the client browser can make asynchronous calls to the server, which allows the user to keep interacting on the webpage without waiting while all the data is being loaded
- **Callbacks:** data can be sent or retrieved from the server without having to send all form data or reload the whole page. It is very useful in order to reload only a specific part of the webpage.
- **User-friendly:** because of the callbacks, the web page will be more responsive and faster to load.
- **Speed:** Ajax allows major improvements over speed in an application thanks to the callbacks. There is no need to wait for the page to reload each time there is an update or a request to the database [25].

For these reasons, Ajax was used whenever there was an update to the database or a request to retrieve data from it. For all the operations available to the user (updating/retrieving data, creating new items...), a PHP script would include the SQL query. Listing 6 shows how the configuration items from the database are loaded.

Listing 6: Function to load the configuration items

```
<?php
include 'db_connect.php';
header('Content-Type: application/json');

function loadConfigItems($conn) {
    try {
        $sql = "
        select name from config_item;
        ";
        $stmt = $conn -> prepare($sql);
        $stmt -> execute();
        $row = $stmt -> fetchAll(PDO::FETCH_ASSOC);
        echo json_encode($row);
    } catch (PDOException $e) {
        echo $e -> getMessage();
    }
}
loadConfigItems($DB_con);
?>
```

After including the file necessary to open the connection, the header specifies the format in which the data will be returned. JSON was used as it is the most convenient format to process. The SQL query is then written, taking into account the security specifications discussed earlier. The result is then encoded in JSON and sent to the client. The previous file returns the results included on Listing 7 from the server.

Listing 7: JSON returned from the server

```
[{"name": "bashful"}, {"name": "doc"}, {"name": "root"}]
```

This JSON can then be processed on the client side. Listing 8 shows how Ajax is used to call the PHP script and handle the JSON sent back from the server.

Listing 8: Script to send an Ajax request to the server

```
$.ajax({
    type : "POST",
    url : "app_db_loadConfigItems.php",
    success : function(data) {
        var items = new Array();
        for (var i = 0; i < data.length; i++) {
            items.push('<input class="btn btn-large btn-info i-graph"
type="button" onclick="addConfigItem(value)" value="' + data[i].name + '">');
        }
        $(".span-cfg").html(items);
    }
});
```

The Ajax query takes different fields into account. First, the type specifies the http method used to send and retrieve information to the server. There are usually two main methods, POST and GET. POST is used here because it is more secure than GET; indeed, in the last one, data is sent as part of the URL, which makes it very unsafe to use if some private information is sent in the query.

The next field, the URL parameter, defines the PHP file which will be executed with the Ajax request. If the file is successfully executed without throwing any exceptions, then the success field is executed. The last field, *success*, is actually a callback function which tells the application what to do with the data retrieved from the server. In the example, it creates an input control for each configuration item in the system. This example shows how Ajax is used to retrieve data. The next paragraphs describes how that same technology can be used to update data.

7.3.4 Updating Data with Ajax

When sending data to the database, the application will use the same techniques as retrieving data, with some differences in how the data is handled. First, the SQL query will be included in a PHP file as previously seen, but no data will be returned from the server this time. Let's take the example of renaming a configuration item, as seen on Listing 9.

Listing 9: Function to rename a configuration item

```
<?php
include 'db_connect.php';

function renameConfigItem($conn) {
    $name = $_POST['name'];
    $id = $_POST['id'];
    try {
        $sql = 'update config_item
        set name = :name
        where id = :id';
        $stmt = $conn -> prepare($sql);
        $stmt -> execute(array(':name' => $name, ':id' => $id));
        $row = $stmt -> fetchAll(PDO::FETCH_ASSOC);
        echo "Update successful!";

    } catch(PDOException $e) {
        echo $e -> getMessage();
    }
}

renameConfigItem($DB_con);
?>
```

The data posted by the user are processed in a secure way to avoid a SQL injection before executing the query. Then, the Ajax callback function shown on Listing 10 is executed.

Listing 10: Ajax request triggered after a configuration item is renamed

```
$.ajax({
  type : "POST",
  url : "db_renameConfigItem.php",
  data : "name=" + name + "&id=" + data.node.id,
  success : function() {
    $("#tree").jstree("refresh");
  }
});
```

Another field that was not included in the previous example is the data field. It includes the different parameters that are sent to the server and used in the query (retrieved by the `$_POST` function in the PHP script). If the script is executed successfully, the tree showing the configuration items is refreshed without reloading the whole page. This shows the importance of using Ajax to process data while communicating with the server in a web application. I will now focus on the different aspects of the application's design.

7.4 Graphic User Interface Components

The Graphic User Interface (or GUI as it will be now referred to) was developed using the standard web technologies HTML5 and CSS3. Although the product was conceived as minimally viable as previously discussed, it was nonetheless very important to include a nice design so it can be appealing to potential clients or investors. Nowadays, there are a lot of tools available to make websites appealing, and these tools were included in the solution. As seen in the programming languages part, the CSS framework Bootstrap was used. This allows a clean and neat interface, with a nice layout.

A major advantage of Bootstrap is the ability to create components that are responsive. That was an essential part in the GUI development to make sure the website would have a nice rendering on a mobile device. First, I present the standard desktop view, which was the view that was used during most of the development. Then, I talk about mobile devices compatibility.

7.4.1 Desktop View

The following screenshot shows the typical view of the application from a desktop browser. In the example, it is displayed using the latest version of Google Chrome (Version 45.0.2454.99 m) as of September 2015 on a Windows 10 laptop.

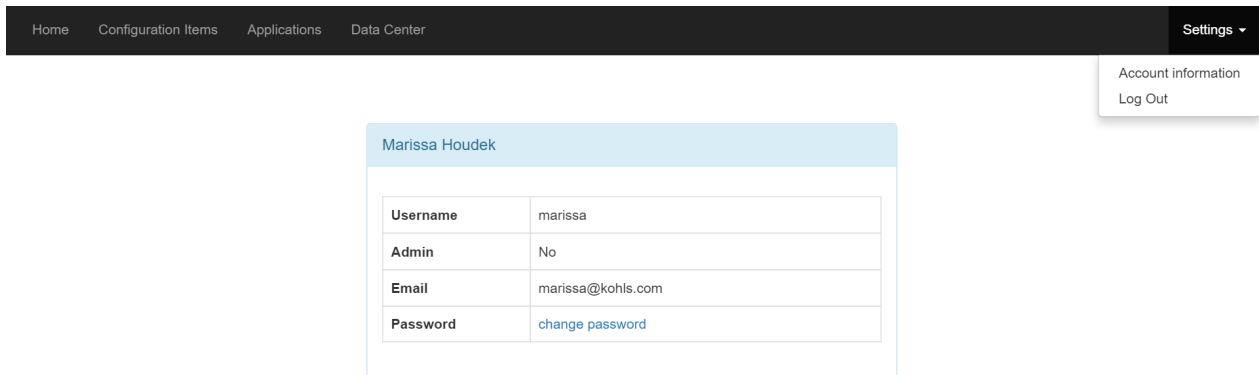


Figure 10: Screenshot from the interface

The first component that is essential is the navigation bar on the top. The four different panels (*Home*, *Configuration Items*, *Applications*, and *Data Center*) allow the user to navigate between the different parts of the application. A view of each of these components will be given later as we will go into the core functionalities of each of these options.

On the right part of this top navigation bar, there is one other component: a drop-down menu to show different settings. *Account information* (the view displayed on the screenshot) allows the user to access their basic information and modify their password (as previously discussed in 7.2.). *Log Out* allows the user to sign out of the application and redirects them to the login page.

The main part of the application with the white background is where all the information will be displayed when clicking on one of the panels. It will usually be separated into two parts: one for visualizing a hierarchy tree between the components, another to directly interact with the options of the selected component. These views will be presented later along with each panel's functions.

In a design perspective, all components present on the page use Bootstrap, but the settings dropdown menu also uses jQuery. JQuery was one of the core tools of the application, both for design and data processing perspectives. CSS does not dynamically interact with the components but just renders them, whereas jQuery allows the user to trigger certain events upon actions on the

webpage, like dropping down a menu when a click on *Settings* is detected. Now that I have described how the application is rendered on a desktop browser, I can present the case for mobile devices.

7.4.2 Mobile Device View

Regarding mobile devices, the views can be slightly different depending on the model used. Developing for these kinds of devices can sometimes become very complicated depending of the device because of the different parameters to consider in the design of the application. Although the whole development was done on a desktop computer, having a mobile-friendly, responsive application is one of the advantages to have a cloud-based application and, as previously seen in the specifications, can be more appealing to potential users and investors.

The counterpart of that thinking was the notion of minimally viable product. It was important to ensure that the application could be used on mobile devices. However, considering each device's characteristics with adapting the layout according to each of them was not an option, as it would make the development process slightly longer.

Using Bootstrap was a way to compromise. The framework has tools allowing the application to adapt its views to the different platforms it is run on. Adopting the framework is also a good way to have a clean, neat code, while using previously defined CSS classes widely used instead of developing my own. To illustrate this, let's take the account information example as shown on Figure 10. The table which contains the different data has a specific styling, which gives it its form with the round-cornered layout. Listing 11 shows the first lines required to build the table.

Listing 11: Code building the account information table

```
<div class="table-responsive">
  <form class="form-horizontal" role="form" method="post">
    <table class="table table-bordered table-hover" style="table-layout: fixed">
      <tbody>
        <tr>
          <td style="width: 150px;"><strong>Username</strong></td>
          <td><?php echo $decoded_json[0]['user_name']; ?></td>
        </tr>
      </tbody>
    </table>
  </div>
```

The different classes associated to the components are part of the Bootstrap framework and allow the table to be styled, along with its content. The class *table-responsive* will literally make the table

responsive, meaning it will size it according to the screen format. Of course, the position according to the other components is kept, but in this way it will be rendered as the central element on the page without requiring to zoom in to see its content. Figure 11 shows how the application would look like on an iPhone6 Plus.

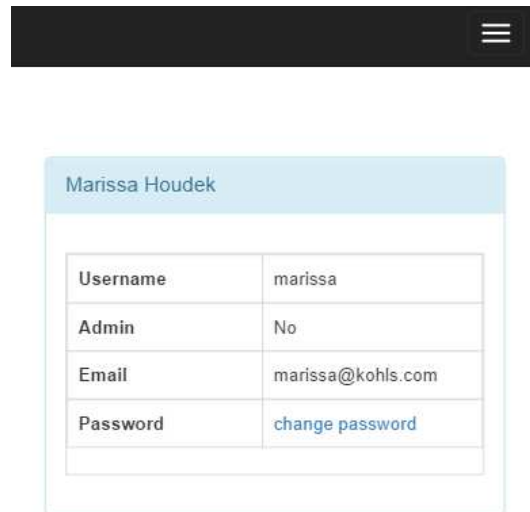


Figure 11: Screenshot from the interface as it would appear on an iPhone6 Plus

There are a couple of things to notice. First, the table keeps its size relatively to the other components. It still is the central part of the view and there is no need to zoom to read what is written. Second, the navigation bar is replaced by a menu icon. When a menu would not fit on the size of the screen, Bootstrap automatically replaces it with an icon which, when clicked, toggles the menu, as shown on Figure 12. Finally, the settings menu keeps its dropdown property, and reveals the additional items when clicked upon.

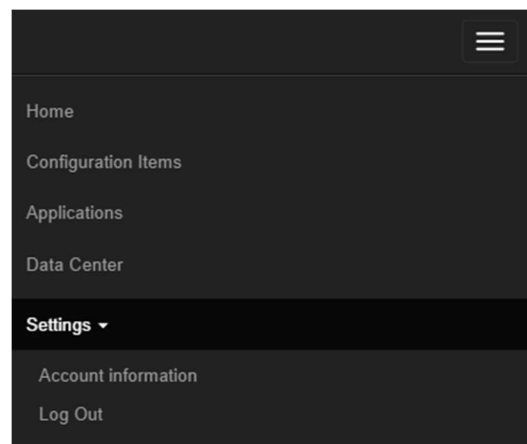


Figure 12: Toggled menu as it would appear on an iPhone6 Plus

To check how the application would look like on a specific mobile device, I used Google Chrome Mobile Emulation [26]. This useful tool is included in Chrome's Developer Tools. Although it will

never be the same as running the application on a real device, it gives a great overview of what it would look like and allowed me to know which parts of the website needed more effort to be made responsive. Not all devices would have a good rendering though. As said earlier, web development for mobile devices is more complex than just adding a few CSS classes to the elements on the page. However, for a minimally viable product, it was not necessary to invest more development time on this aspect. Now that I have talked about the different characteristics of the user interface, I can focus more specifically on the different functionalities of the application.

7.5 Application Features

In this part, I describe in more detail the different parts of the application and their implementation. As seen in the requirements, there are three main parts in the website: configuration item which lists the different items in the company along with their properties, application which displays a graph showing what configuration items are used by a specific application, and data center which allows the user to visualize the map of the different data centers of the company along with their servers. Let's focus on the configuration items page first.

7.5.1 Configuration Items

This page is the core of the application. Managing configuration items is the prime goal of a CMDB, so it is important that the interface provides key features allowing this management to be both efficient and easy to use. The following screenshot presents the interface.

The screenshot displays the Configuration Items management interface. On the left, a navigation tree shows the hierarchy: Root > Server > Linux > lazy. The main panel shows the configuration for 'Root:Server:Linux:lazy'. It features three tabs: 'General' (active), 'Financial', and 'Labor'. The 'General' tab contains three sections: 'Linux' with a 'distribution' field set to 'Arch Linux', another 'Linux' section with a 'version' field set to '5', and a 'Server' section with a 'hostname' field set to 'lazy'. At the bottom of the form are 'Save' and 'Cancel' buttons.

Figure 13: Configuration Items management page screenshot

There are two main parts. On the left, a tree presents the hierarchy between the different configuration items in the company. On the right, a panel displays the properties of the selected node. If the user clicks on a configuration item, they see the view presented on Figure 13, showing the properties inherited from the parent classes as well as the values associated with them for this specific configuration item. When they click on a class, they will have a view displaying the properties and their types (float, string, or date) as shown on Figure 14. The only exception is when the user clicks on the root node. In that case, the right panel is left blank.

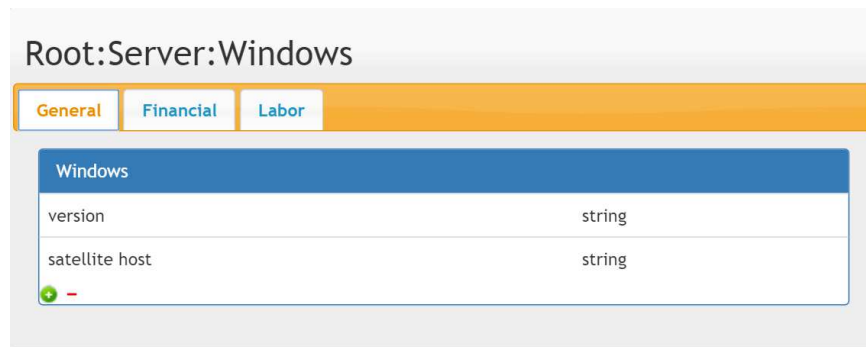


Figure 14: View of the right panel when a class is selected

The tree is rendered through a jQuery plugin, jsTree [21]. In this representation, folders represent classes, and files are used for configuration items. As we have seen in the data model, each item is associated with a class, and a class has properties that every configuration item associated with it will inherit. It is essential to note that the previous examples are displayed using administrator rights, allowing the modification, creation, and deletion of new items, classes, properties, and values.

To create the tree, first the JavaScript file defining jsTree needs to be referenced in the HTML document and an element in which the tree will be displayed is selected using the CSS selectors. Once this is done, a second JavaScript file, *ci_admin_tree.js*, is included in the HTML file to process the interactions between the tree and the user on one part, and the queries to the database on the other part. Listing 12 shows how these steps are implemented.

Listing 12: Definition of jsTree in the code

```
[HTML]
<div class="col-md-4" id="tree"></div> [...]
<script src="dist/jstree.min.js"></script>
<script src="ci_admin_tree.js"></script>

[JavaScript]
$("#tree").jstree({
  "core" : { ... },
  "data" : { ... },
  "types" : { ... },
  "contextmenu" : { ... },
  "plugins" : [ ... ]
});
```

We can notice five properties that are defined to generate the tree. The *core* property is used to check callbacks before an operation is completed on the tree. It is used to confirm the deletion of a node. *Data* is what defines the values inserted in the tree. It takes a JSON file as a parameter and builds the tree according to the data in the file. The JSON file must specify three parameters: the node name, the node type, and its parent. By default, there is one root node, and all the other initial nodes like Database or Server in the example have it as a parent. However, the root node is only used to format the data in the tree and has no properties associated to it. In the example, it is a PHP file returning JSON from the server that is passed to the *data* parameter.

The next parameter, *type*, specifies the different types possible in the tree. Here there are only two, folder and file, but there can be more. It is possible to customize each type by associating an icon with them. *Contextmenu* defines the menu when a right-click is detected on one of the tree items. It references another function that builds the contextual menu. For further information, the whole code building the tree and context menu is included in addendum III. Finally, *plugins* lists all the different plugins used to manage the tree. Indeed, each functionality, like defining types or using a context menu, requires the import of a specific plugin. Like that, only the plugins used are loaded, which accelerates the rendering of the tree and improves the performances.

An important feature of jsTree is the ability to know the id of the selected node. This is very useful in order to dynamically load each node's data on the right panel. In *ci_admin_tree.js*, a global variable *global_id* is defined, along with two functions, *getCurrentId* and *setCurrentId*. The first one is used to retrieve the node id when the application sends it to the database, the second sets the variable to the current id node value when the user selects a new node in the tree.

The right part of the interface displays the properties and values corresponding to the selected node. jsTree implements events allowing the system to know which node type is selected. Thus, a different function is called according to the type. Then, the property is displayed in its corresponding category defined in the database (general, financial, or labor). The tabs are created with jQuery UI, a plugin built on top of jQuery. [27] Just like the system knows which tree node is selected, it is also possible to know which tab the user is currently on, thanks to the function *getCurrentTab* presented on Listing 13.

Listing 13: Function to get the selected tab

```
function getCurrentTab() {  
    var active = $('#tabs').tabs('option', 'active');  
    var tab = $("#tabs ul>li a").eq(active).attr("href");  
    return tab.substring(1);  
}
```

Contrary to the node value though, there is no need for a global variable. When a user is on a specific tab, jQuery knows which one is active. The function retrieves the current active tab and returns its name to be used with the current id to get the specific values/properties of a tab. When the selected node is a folder (meaning the object represented is a class), the function *getProperties*

seen on Listing 14 is executed and retrieves the selected class properties from the server using Ajax.

Listing 14: Function to retrieve the properties of the selected configuration item

```
function getProperties(tab, id, table, classTitle) {
    $.ajax({
        type : "POST",
        url : "db_loadClassProperties.php",
        data : "tab=" + tab + "&class_id=" + id,
        success : function(data) {
            var htmlResult = new Array();
            var title = new Array();
            for (var i = 0; i < data.length; i++) {
                for (var j = 0; j < data[i].content.length; j++) {
                    htmlResult.push('<tr><td>' + data[i].content[j].name
+ '</td><td>' + data[i].content[j].value_type + '</td></tr>');
                }
                for (var k = 0; k < data[i].title.length; k++) {
                    title.push(data[i].title[k].name);
                }
            }
            $(table).html(htmlResult);
            $(classTitle).html(title);
        }
    });
}
```

The function sends the id of the selected class along with the tab id that the user is on. The query returns a JSON string containing the name of the property, its type, and the class name. They are then pushed into an HTML string that is displayed into the HTML tags specified when the function is called. The function *getValues* that retrieves the values for the selected item's properties works on the same basis. It calls a PHP script that sends a query to the database and returns the result in a JSON string. That string is then parsed and rendered into HTML.

The advantage of these functions is the ability to load only the data needed by the user. Only the data from the current id-tab couple is loaded, allowing the results to be processed and displayed faster. When there are just a few items, it does not really matter, but if the application processes hundreds of configuration items it can have a big impact on the rendering time.


For an administrator, the system also allows an interaction with the data. Regarding the properties, two controls are included to allow the user to add or delete a property (see Figure 15). When the user clicks on the  control, a line is appended to the table prompting the user to enter a new property name and select its type.

Figure 15: Adding a property

Notice how the *save* and *cancel* buttons appear. Once the user submits the data, the function *createProperty* is called with the parameters entered by the user. An Ajax request is then sent to add the new property. Adding a property in the database is more complex than a single query: the property needs to be mapped with the class, and the property's values need to be instantiated as well. Indeed, when a property is created, all configuration items automatically inherit it, so new rows need to be added to the *Property_value* table. This is done in the file *db_createProperty.php* presented in addendum IV.

The other control **-** is used to remove the selected property. When the user clicks on a row in the table, it is highlighted in light grey as seen on Figure 15. Then, the user can delete the property by clicking on the control, which calls the function *removeProperty* presented on Listing 15.

Listing 15: Function to remove the selected property

```
function removeProperty(name, id) {
    if ($("#tr").hasClass("highlight")) {
        if (confirm("Are you sure you want to permanently delete this
property?")) {
            $.ajax({
                type : "POST",
                url : "db_deleteProperty.php",
                data : "name=" + name + "&id=" + id,
                success : function(data) {
                    $('#highlight').remove();
                }
            });
        } else {
            return;
        }
    } else {
        alert("Please select a property to remove!");
    }
}
```

The function starts by checking that the row is selected (represented by a highlight class in CSS). Then, an alert prompts the user to confirm his choice before calling an Ajax request to perform the deletion of the item in the database. The SQL will not only delete the property, but also all the

values associated with it along with the mapping with the class. Upon success, the row is removed from the table right away.

By default, when a property is added, its value is set to NULL. The user can then edit the value when they click on a configuration item in the tree. The view available to them was previously shown on Figure 13. Sometimes, it can happen that some configuration items do not need a value for a property which is needed by other configuration items of the same class. In that case, the user has the opportunity to let the value to NULL, which will simply indicate that the property is not set for this specific item. Like that, the system is very flexible and the programming part is kept simple as well. It can also happen that a specific tab will have no properties. In that case, a simple message will show it to the user, as seen on Figure 16.

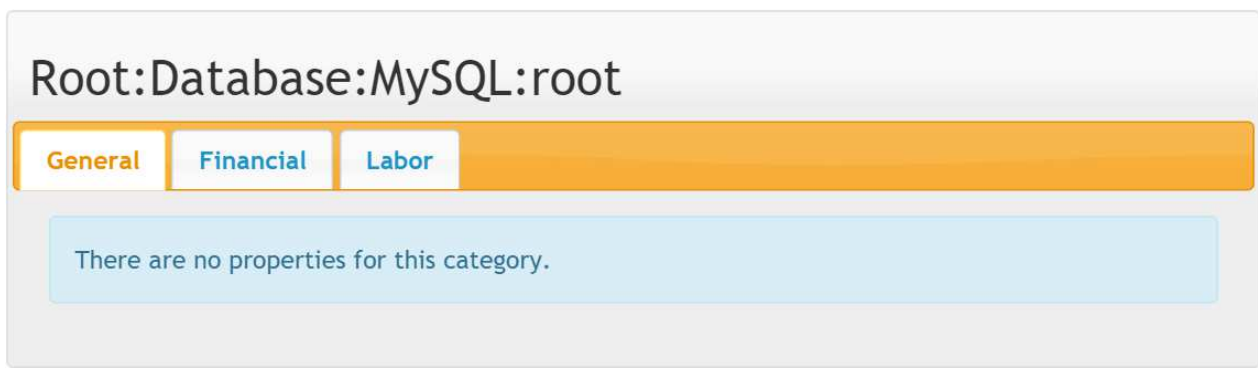


Figure 16: Configuration item view with no properties

To update a property, the user clicks on the save button which sends the values to the database using Ajax. The query then processes the data and, according to the property type, updates the corresponding field in the *Property_value* table, using the function *updateDB* shown on Listing 16. A message is then displayed on the interface to let the user know if the update was successful.

Listing 16: Function to update the properties in the database

```
function updateDB($conn) {
    $conf_id = $_POST['id'];
    $name = $_POST['name'];
    $value = $_POST['value'];
    try {
        $sql = 'update property_value, property
                set    property_value.str_value = if(property.value_type = "string",
:value1, null),
                property_value.date_value = if(property.value_type = "date",
:value2, null),
                property_value.float_value = if(property.value_type = "float",
:value3, null)
                where property_value.property_id = property.id
                and property_value.config_id = :conf_id
                and property.name = :name';
        $stmt = $conn -> prepare($sql);
        $stmt -> execute(array(':value1' => $value, ':value2' => $value,
':value3' => $value, ':name' => $name, ':conf_id' => $conf_id));
        $row = $stmt -> fetchAll(PDO::FETCH_ASSOC);
        echo "Update successful!";
    } catch(PDOException $e) {
        echo $e -> getMessage();
    }
}
```

The last point on the configuration items interface is the context menu for the tree. We have seen earlier that it was defined during the initialization process of the tree. According to the node type, the options available will change. The following images show the context menu according to the three different node types possible.

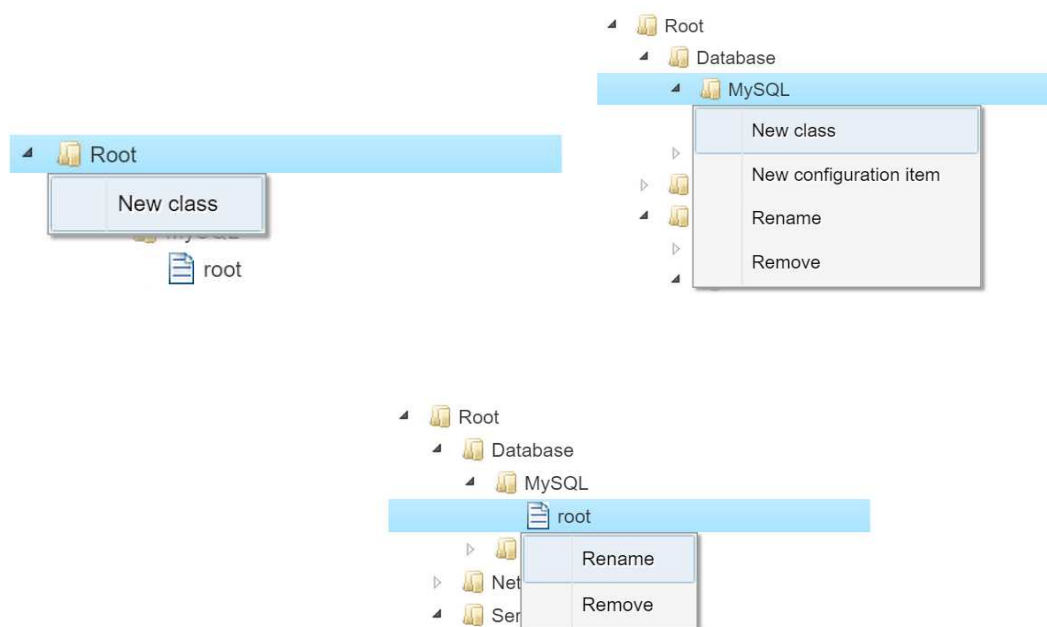


Figure 17: Context menu according to the node type

When the node is the tree root, the only possible action is to add a class. A configuration item cannot have the root node as a parent because it would not make sense (the root node does not have properties). Classes, on the contrary, can have both configuration items and other classes as their children. Nested classes are possible, just as shown in the example. The typical example is a class needed to represent the server with subclasses to represent the different operating systems available, since properties can be OS-specific.

It is possible to rename a class. However, removing a class requires it to have no children. This was done with a security aspect in mind, as it would be easy to mistakenly delete dozens of elements (the class but also all the elements it contains). Finally, on a configuration item, the only action possible is to either rename or remove the item. Now that I have presented in more detail what the configuration item page does, I can move onto the application page.

7.5.2 Applications

The purpose of the application page is to list in a tree similar to the one used for the configuration items all the applications used within the company. For each application, a graph should be displayed showing the different configuration items used by the application. An administrator will be able to interact with the elements on the page and modify them, while a user will only have a read-only privilege.

Graphically, the structure of the page is similar to the one used for the configuration items. On the left, a hierarchical tree displays the applications into categories. On the right, a panel shows the graph of the selected application (see Figure 18).

The tree is really similar to the one used on the configuration item page. The files represent the applications, and the folders are used to organize these applications according to the department in which they are used (human resources, IT...). A folder can be included in another folder and be the parent of an application, except for the root folder which can only be the parent of a folder. Applications can only be deleted or renamed. Finally, a folder can be deleted only if it is empty (same reason than in the previous tree).

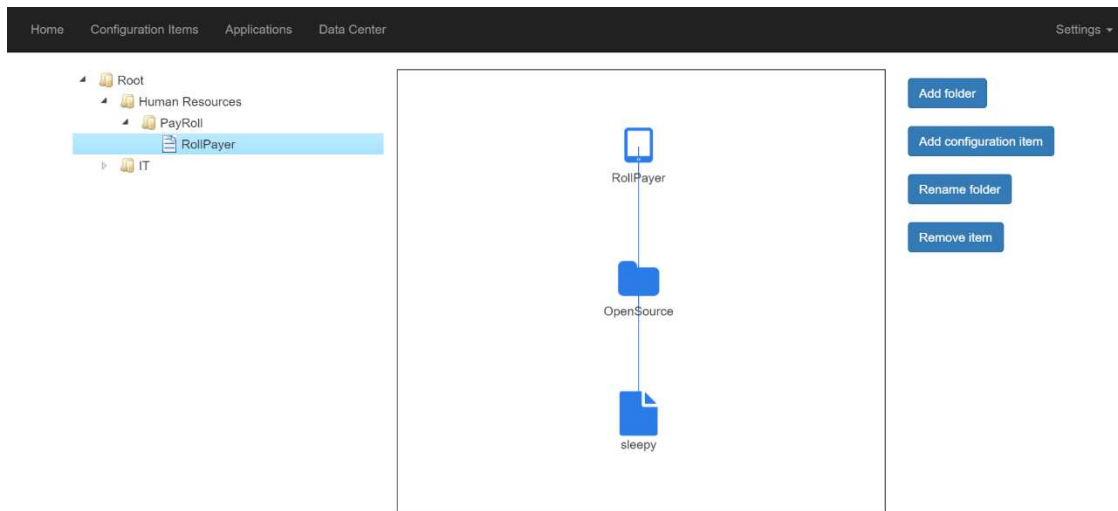


Figure 18: Applications management page screenshot

The screenshot shows that a graph is displayed when an application is selected. When it is a folder, the right part is left blank. Indeed, folders are just used to organize the applications, and do not have properties like the classes did for the configuration items. As mentioned in 6.5, the graph is displayed using Vis.js, a JavaScript library. In the HTML file, an element is identified to host the graph. The HTML file also references the Vis.js source code, as well as a JavaScript file which includes all the functions necessary to interact with the tree. In this last file, the graph is created and inserted into the HTML element. Listing 17 presents the different steps to set up the graph.

Listing 17: Code to set up the graph

```
[HTML]
<div class="col-md-6" id="mynetwork" style="display:none"></div>

[JavaScript]
function setContainer() {
    container = document.getElementById('mynetwork');
}

function setupNetwork() {
    network = new vis.Network(container, g, options);
    network.fit();
    network.on("selectNode", function(params) {
        node_id = params.nodes[0];
        size = params.edges.length;
        $('.cat1').hide();
        $('.span-cfg').hide();
    });
    network.on("deselectNode", function(params) {
        resetNodeId();
        $('.cat1').hide();
        $('.span-cfg').hide();
    });
}
```

The function *setContainer* uses a CSS tag to select the HTML element in which the graph will be rendered. That function is then called in *setupNetwork*, which creates the graph. The variable *g* represents the graph as an object. It is defined in the function *setGraph*, which will be presented later. *Options* are the different parameters associated with the graph's nodes, comprising the icon's color and size. Finally, the function defines the events for the selection and deselection of a node. In the first case, the node id is retrieved and passed into a global variable. In the second case, the id is reset to its default value.

The data used to create the graph is retrieved from the server. In the tree, when the user selects an application, an Ajax query is sent to the server via a PHP file to retrieve the graph from the database. The result is returned in a JSON string. Listing 18 gives an example of a JSON file returned from the server; the data included in it will allow the application to build the graph.

Listing 18: Example of a JSON file returned from the server

```
{ "nodes":
  [
    { "id":1, "group":"app", "label":"ADP", "level":1},
    { "id":2, "group":"folder", "label":"AppServers", "level":2},
    { "id":7, "group":"config_item", "label":"bashful", "level":3},
    { "id":8, "group":"config_item", "label":"doc", "level":3}],
  "edges":
  [
    { "id":1, "from":1, "to":2},
    { "id":2, "from":2, "to":7},
    { "id":3, "from":2, "to":8}] }
```

To define a network using Vis.js, two categories need to be defined: the nodes and the edges. The nodes represent the folders and the configuration items. They are associated with some properties: *id* which identifies the node, *group* which determines the type of node (application for the root node, folder or configuration item for the other nodes), *label* is the name of the node displayed under the icon, and *level* gives the node depth in the network. The second category, *edges*, represent the connection between the nodes. The id is used to identify a specific edge, *from* and *to* are used to connect two nodes together. There is no direction in a link between two nodes, which means the *to* and *from* values can be switched without altering the rendition of the network. When the user selects an application in the tree, the function *setGraph* shown on Listing 19 is called and builds the network using the JSON data as a parameter. It is then displayed in the HTML container.

Listing 19: Function to load the graph from a data source

```
function setGraph(data) {
  g = {
    nodes : data.nodes,
    edges : data.edges
  };
}
```

Regarding the different icons used, the network root represents the application itself. Each application uses configuration items; these are represented as leaves in the network with file icons. They cannot be parents of another node. These configuration items can be organized in folders (not to be confused with the folders from the tree). Their nesting level is unlimited. All these different icons are defined in the function *setOptions*. Listing 20 shows how an icon is defined.

Listing 20: Function to set the options for the graph nodes

```
function setOptions() {
  options = {
    groups : {
      app : {
        shape : 'icon',
        icon : {
          face : 'FontAwesome',
          code : '\uf10a',
          size : 50,
          color : '#2B7CE9' } },
      folder : { ... },
      config_item : { ... } },
    layout : {
      hierarchical : {
        direction : 'UD' } } } };
}
```

The icons used belong to a specific CSS font called Font Awesome [28]. To use them, the library must be included in the html references. Since it is quite simple to implement, it was the solution retained for displaying the icons. There is the possibility to customize the size and the color. A last thing to note in the options is the *layout* property. For the network, it is displayed hierarchically in an up-down position. Other positions are also available, such as down-up and horizontal alignment as well.

A number of controls allowing the user to interact with the network are available. The user can add a folder or a configuration item. In the case of the last one, a list with the different configuration items included in the database appears and lets the user picks one that they want to add. The same item can be reused many times, since a configuration item can be used in multiple applications. For example, a Windows server can use a human resources platform as well as a payroll application.

The next option allows the user to rename a node. Only folders can be renamed. Indeed, configuration items need to be renamed using the regular configuration items page, and the application nodes need to be renamed in the tree and not on the network. This choice was made for a practical reason; managing those elements should not be a task performed by the network, but by the appropriate interfaces of the software.

The last control, when clicked, removes the selected component from the network. The application node cannot be removed since it is the root node in the network. Concerning the configuration

items, they are deleted from the network but not from the database. Indeed, the graph only deletes the reference to the item, and not the item itself. This terminates the part about the application management features. I now present the different functions associated with the data centers.

7.5.3 Data Centers

In this part, I talk about the part of the software dedicated to visualize the company's data centers. The purpose of the webpage is to display the data center rooms and to show the exact position of the cabinets in the room, as well as a closer look at each cabinet to see the different servers it contains. As with the other pages, there are two views: administrator and regular user.

Before describing the interface, it is necessary to have a basic understanding on how a data center works. It is a room divided into tiles (or units) designated by its horizontal and vertical position. Each row and column has a label giving the position of a specific element, just like a grid. Cabinets, which are metal-structure furniture, are positioned in the room and are used to store servers on them. There can be more than one server per cabinet. A cabinet can also be used to store air conditioners to cool the room temperature down. Not all tiles in the room are used, since some space is needed for technicians to come and inspect each server during maintenance periods. The goal of this web page is then to display a data center as previously described. Figure 19 presents the general view of the data center webpage.

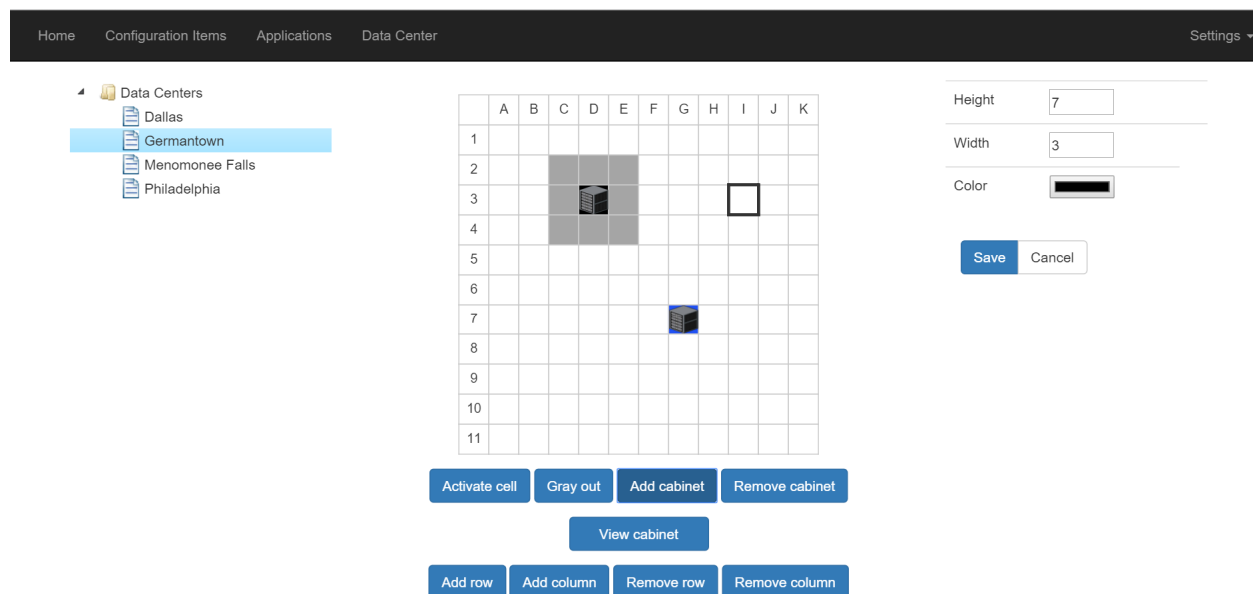


Figure 19: Data center page screenshot

As with the previous page, the tree on the left helps the user to create, edit, and delete data centers, and the right view shows the details of the selected data center. When the user selects the root node,

he has the opportunity to create a new data center. The form on Figure 20 then appears instead of the grid shown on Figure 19.

Name	<input type="text" value="Seattle"/>
Number of rows	<input type="text" value="10"/>
Number of columns	<input type="text" value="10"/>
Rows label	<input type="text" value="1"/>
Columns label	<input type="text" value="A"/>
Tile dimension	<input type="text" value="2"/>

Figure 20: Form for creating a data center

These parameters help to define the properties of the data center that will be used to build the grid. They all correspond to the different settings explained in the data model regarding the *Data_center* table. Once the user validates the form, the grid is created. Listing 21 shows the function that is called after the form is validated.

Listing 21: Function to create the data center grid

```
function clickableGrid(rows, cols, labelRows, labelCols, callback) {

    var i = "";
    var grid = document.createElement('table');
    grid.className = 'grid';
    grid.id = 'table';

    for (var r = 0; r < rows + 1; ++r) {

        var tr = grid.appendChild(document.createElement('tr'));

        for (var c = 0; c < 1; ++c) {
            var xCell = tr.appendChild(document.createElement('td'));
            if (r != 0) {
                if ($.isNumeric(labelRows)) {
                    xCell.innerHTML = labelRows - 1;
                } else {
                    xCell.innerHTML = previousChar(labelRows);
                }
            }
        }
        for (var c = 1; c < cols + 1; ++c) {

            if (r == 0) {
                var yCell = tr.appendChild(document.createElement('td'));
                yCell.innerHTML = labelCols;
            } else {
                var cell = tr.appendChild(document.createElement('td'));
                cell.addEventListener('click', (function(el, r, c, i) {
                    return function() {
                        callback(el, r, c, i);
                    };
                })(cell, r, c, i), false);

                if ($.isNumeric(labelCols)) {
                    labelCols++;
                } else {
                    labelCols = nextChar(labelCols);
                }
            }
            if ($.isNumeric(labelRows)) {
                labelRows++;
            } else {
                labelRows = nextChar(labelRows);
            }
        }
    }
    return grid;
}
```

The grid is no more than a HTML table with rows and columns. The function loops through the number of rows and columns specified in the parameters, and then builds each cell while adding a listener on the mouse click, allowing the user to select the cell by clicking on it. A callback function is also added to the cell to retrieve the cell position and its values. When a cell is selected, it is styled with CSS with a thicker border as shown on Figure 19. An additional row and column are also added to show the grid labels.

There are a certain number of controls below the grid allowing the user to interact with it. First, the user can *gray out* or *activate* a cell. This is used to show a cell that is not suitable to host cabinets in the room. A grayed out cell is represented in gray, as seen on Figure 19. The function called does two things: it sends to the server a Boolean to update the *grayed_out* parameter in the table (true if the cell is grayed out). Then, it changes the background of the HTML cell.

Next to these controls are the ones allowing the creation and deletion of a cabinet. When *add cabinet* is clicked, the form shown on the right on Figure 19 appears. It allows the user to enter the different options of the cabinet. The color feature uses the HTML5 input color field, which returns a result in RGB. Once the form is submitted, the cabinet is created in the database and the function *addCabinet* presented on Listing 22 is called. It adds an image to the cell and changes the background color according to the cabinet color. The image was taken from the Internet [29]. Note that this function is also called when the grid is loaded on the page after retrieving the cabinets previously saved in the database.

Listing 22: Function to add a cabinet on the grid

```
function addCabinet(x, y, color) {  
    var table = document.getElementById("table");  
    var row = table.rows[x];  
    var cell = row.cells[y];  
    $(cell).html("<img src='img/cabinet-icon.png'>");  
    $(cell).css("background-color", color);  
}
```

The next buttons, *add row* and *add column*, will, when clicked, add a row (or column) after the last row (or column) of the table. Similarly, the controls *remove row* (or *remove column*) will remove the last row (or column) from the table. Doing so also erases any cabinets that would be located on tiles in the last row (or column).

The last button *view cabinet* gives a closer look to the cabinet and allows the user to view it with the different racks, just like in reality. From there, the user can see the servers installed on the cabinet, and can also add or remove servers. Figure 21 shows the representation of one of these cabinets.

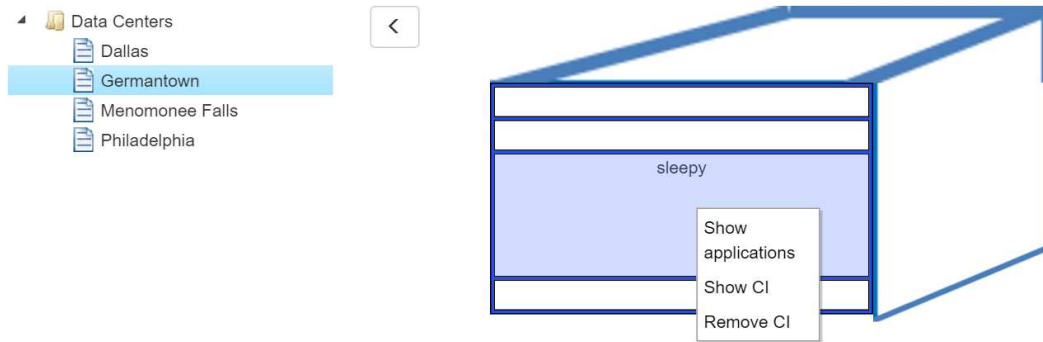


Figure 21: Data center page with the cabinet view

The tree is still present on the left if the user wants to access another data center, but the grid is replaced by the cabinet view. To go back to the grid view, one must click on the button with an arrow. Regarding the cabinets, they are built with a succession of HTML elements *div* (white boxes) representing the racks, grouped into a larger *div* element representing the frame of the cabinet (blue box). The latter element is combined with two other images representing the cabinet (as seen on Listing 23), in order to give the schema a one-point perspective representation.

Listing 23: Code creating the structure of the cabinet

```
<div class="col-md-10">
  <div class="row">
    <div class="col-md-12">
      
    </div>
  </div>
  <div class="row">
    <div class="col-md-10" id="racks-container">
      <div id="racks"></div>
    </div>
    <div class="col-md-2" id="img-container">
      
    </div>
  </div>
</div>
```

The Bootstrap framework is used for the layout. Two different rows are created, the first one containing the image for the top of the cabinet, the second one for the racks and the right part of the cabinet. Although not a necessity, it brings a more realistic touch to the cabinet in a simple way. The images were designed with PowerPoint. The function *buildRack*, shown on Listing 24, is called when a cabinet is selected and builds the racks according to the height of the cabinet, set by the user during the creation of the cabinet.

Listing 24: Function to build the cabinet's racks

```
function buildRacks(id) {
    $.ajax({
        type : "POST",
        url : "dc_db_getHeight.php",
        data : "id=" + id,
        success : function(data) {
            var htmlResult = new Array();
            for (var j = 0; j < data.length; j++) {
                setHeight(data[j].height);
                for (var i = 0; i < data[j].height; i++) {
                    var el = $('<div class="clickable-div second-row"
id="rack' + i + '"></div>');
                    htmlResult.push(el);
                    addContextMenu(el);
                }
                getServers(data[j].id);
                $("#racks").html(htmlResult);
            }
        }
    });
}
```

The function calls a PHP file to retrieve the height of the cabinet. Then, according to the height, a number of racks are created, each being given an id so they can be identified when the user will click on them. The result is pushed into a *div* container representing the cabinet frame. The class *second-row* assigned to each element is used to style them when the user points the mouse on one of them, as seen on Figure 21. The opacity is slightly decreased and the mouse cursor changes.

A context menu is also added when the right click is triggered (or when the user selects for a few seconds the element on a touch screen). This context menu will allow the user to interact with the servers. Four options are available: *show CI*, *remove CI*, *show application*, and *add CI*. The first three ones are displayed when there is a server in the selected rack (like the previous screenshot).

When *show CI* is clicked, the website redirects to the configuration item page to see the properties of the configuration item selected. The second option, *remove CI*, will remove the server from the cabinet. It will not remove the server from the database though, but only the reference to the cabinet.

The next option, *show application*, will display the different applications which are using the selected configuration item. Since many applications can use one configuration item, it is necessary to show them all; thus, a popup window appears with the different applications path, as seen on Figure 22.



Figure 22: Applications popup

The user can click on the file name, which will redirect him to the application window. If there is no application linked to the selected configuration item, then a message notifying the user is displayed instead of the list of items.

The popup is built using jQuery UI, the same jQuery plugin that is used to build the tab panels on the configuration items page [27]. Once the user clicks on *show application*, an Ajax request queries the server to retrieve the full path of the applications after sending the name of the selected configuration item. Since names are unique in the configuration item table, there is no need to use an id-based query here. Listing 25 shows how this is done.

Listing 25: Code creating the application popup

```
'show_app' : function(t) {
    setPosition(t.id);
    var ci_name = document.getElementById(getPosition()).innerHTML;
    $.ajax({
        type : "POST",
        url : "dc_db_getPopupContent.php",
        data : "name=" + ci_name,
        success : function(data) {
            var chain = "";
            if (data.length != 0) {
                for (var i = 0; i < data.length; i++) {
                    for (var j = data[i].parents.length - 1; j >= 0; j--) {
                        chain += '' + data[i].parents[j].name;
                    }
                    chain += '<a
href="app_admin.php">' + data[i].application + '</a>';
                    chain += '<br>';
                }
            } else {
                chain = "There is no application using this configuration item.";
            }
            $("#dialog").html(chain);
            $("#dialog").dialog("open");
        }
    });
},
```

To work, the plugin requires a reserved *div* in the HTML file. The content of the popup is no more than a string that is set to be the content of that *div* and that is opened once all the data are loaded into the dialog box. The last option in the context menu, *add CI*, is available only when the field selected is blank (means there is no server associated with it). Clicking on it does not add a server automatically, but instead shows a form on the right letting the user choose which server to add and which height it will have in the racks. Figure 23 shows this form.

Figure 23: Form to add a server to a cabinet

All configuration items inheriting from the *Server* class are retrieved and shown in the select box. Since a server can only be present on one cabinet, the list does not show the items that are already positioned on another cabinet. The height field allows the user to select the height of the server in the cabinet. It is not measured in feet but in number of racks. Thus, only integer can be used. This system is simpler than using real dimensions. Moreover, the number of racks is also used as a measurement unit in real data centers too.

Before the form is submitted, the system needs to check that the server height can fit in the cabinet, and also that there are no elements in between the space the server would take. Listing 26 shows extracts from the form code doing that.

Listing 26: Code checking if a new server can be added

```
var pos = parseInt(getPosition().substring(4));
var height = parseInt($("#item-height").val()); //height of the server
var bool = true;
var cabinet_height = getHeight(); // height of the cabinet
var adjusted_pos = pos + height;

if (adjusted_pos <= cabinet_height) {
    for (var i = pos; i < adjusted_pos; i++) {
        if ($("#rack" + i).html() != "") {
            bool = false;
        }
    }
    if (bool) {
        [send data to the server]
    }
}
```

Let's explain the variables; *pos* represents the position at which the server will be inserted; *height* is the height of the server retrieved from the form; *bool* is a Boolean that will be used to check if the server can be inserted at the specified position with the specified height; *cabinet_height* is the height of the cabinet; and finally *adjusted_pos* is the position the server will take once inserted in the cabinet.

First, the function needs to check that there is enough room in the cabinet to insert the server at the given position (if the adjusted position is inferior or equal to the cabinet height). If this condition is met, the system needs to check, for all the elements where the server would be positioned, that there is no other server in the middle (if the html element has some text inside, then there is a server). If there is but one, the Boolean will be set to false and the rest of the function will not be executed.

Graphically, when a server is added, the next HTML *div* components will be merged with the selected element on which the server was inserted. Thus, the real dimensions are kept and faithfully show what can be seen on a real cabinet. Note that the elements merged are positioned down to the selected element. No upper elements will be modified. The function *addServer*, presented on Listing 27, shows how this is done.

Listing 27: Function to add a server to the selected rack

```
function addServer(position, name, height) {  
    var limit = position + height;  
    var pxHeight = height * 25;  
    $("#rack" + position).html(name);  
    for (var i = 1; i < height; i++) {  
        $("#rack" + position).next('.second-row').remove();  
        $("#rack" + position).css("height", pxHeight + "px");  
    }  
}
```

Similarly, when a server is removed, new HTML *div* elements need to be added to keep the height of the cabinet in harmony with its specifications. This is done by resizing the element that contained the server to a one-element size, and then by adding other *div* between the resized element and the next one. This is performed by the code shown on Listing 28; it is executed when the user confirms the deletion of a server from the cabinet.

Listing 28: Code executed when a server is deleted from the rack

```
if (confirm("Are you sure you want to remove this server from the cabinet?")) {
    var pos = parseInt(getPosition().substring(4));
    var incr = pos + 1;
    $.ajax({
        type : "POST",
        url : "dc_db_deleteServer.php",
        data : "position=" + pos + "&id=" + getTileProperties().id,
        success : function(data) {
            $("#" + getPosition()).html("");
            $("#" + getPosition()).css("height", "25px");
            for (var i = 0; i < data.length; i++) {
                for (var j = 0; j < data[i].height - 1; j++) {
                    $("#" + rack" + pos).after('<div class="clickable-div
second-row" id="rack' + incr + '"></div>');
                    pos++;
                    incr++;
                }
            }
        }
    });
}
```

First, the position of the element is retrieved. Then, another variable, *incr*, is created. That variable represents the *div* element located right after the current position. Two variables are necessary because the algorithm will be using both. On the server side, an Ajax request deletes the reference to the server in the configuration item table. If successful, it returns the height of the server that was deleted.

Graphically, the text of the *div* containing the server is erased and the HTML component is resized to a standard rack unit size of 25 pixels. Then, based on the height of the deleted element, new racks are added to fill in the blank left by the vacant server. The variable *pos* defines the id of the current rack, and *incr* defines the id of the next rack. The new elements are inserted after the existing element, which justifies the use of two variables.

An alternative to doing that would be to only remove the reference in the database and reload the rack with an Ajax request. However, working with the graphic HTML elements allows less requests to be sent to the server, which will avoid an overload if many users modify a cabinet at the same time.

This concludes the part explaining the data centers. Throughout this whole section presenting the implementation of the solution, I described the different features that the cloud-based CMDB provides to the user. I can now move forward to studying the technological progress and economic strategy for the CMDB application.

8 Technological Progress and Economic Strategy

In this part, I discuss the different innovations that the cloud-based CMDB brings to the software world, as well as the economic strategy for this product.

8.1 Technological Progress

In terms of technology, the product is an innovation as it brings two worlds together: desktop enterprise software and web applications. As of today, no commercial solutions can manage to bring the CMDBs in the cloud world. As we have seen in parts 4.2 and 5.4, current CMDBs were not conceived for the cloud, and bringing an enterprise application would simply not work because of its heaviness. However, finding an adaptive solution that could bring the two worlds together in a new way could technologically work. This is why the application I developed is innovative, as it brings this new technology concept that simple CMDBs could work in the cloud in an easy way.

Regarding the tools employed, it is also a great step in the open-source world, since the application uses a lot of third-part resources, thus decreasing costs of implementation and licenses. As of today, there is no solution on the market coming close to this concept of application. Some come close [30], but these solutions would rather seek a replacement of the CMDB concept with another technology, which is not the goal here.

The other innovation that this technology brings is making a tool available to the market no matter which device is used to access it. The industry sometimes produces different versions of an application in order to adapt it better to a specific platform or device. The online CMDB that was developed has the advantage to be responsive and adaptive to most platforms. Of course, as it is a prototype, it will not work on all devices, only on the most popular, but the concept of having only one application that can be accessible by thousands of devices at the same time, no matter what the platform is, is a technological progress that need to be mentioned. These new technological steps are important, and will allow the company to adopt an economic strategy to make the product available to customers in the future. The next part is focusing on that aspect.

8.2 Economic Strategy

The strategy developed in an economic way is intrinsically linked to the way the product was being developed. I mentioned before that one of the most important features was to present a minimally viable product, which is a product created to attract investors and convince them that the application will be a good product for the market. Thus, the final product will not be released to customers. It will be shown to investors to explain to them the concept of the application and convince them that an online cloud-based CMDB is a solution that can and will work if adopted in companies. It is then the idea, more than the product itself, which will be sold to these investors.

To fulfil this objective, it was necessary to have a first draft of the solution, which is the application I developed. Using only open-source tools, the production costs were then decreased since no licenses had to be bought in order to use third-party tools. Thus, the costs of the application was only the workforce.

The potential investors can be members of a company's board, IT managers, or marketing specialists, who would be willing to try the product internally for a simulation. There would be no cost to try the application. If the investors are won over by the idea, they will be ready to invest in the product. Prime Resources will then be able to create a new team dedicated to the development of the application, and the final result will then be accessible to all customers and released globally to all companies.

Another parallel solution would be to release the prototype version on platforms such as Source Forge or GitHub, allowing developers worldwide to access, use, and even develop on the initial solution. This open-source approach could then reach more people, and advertisements would be used as a revenue source for this potential release.

For Prime Resources, building software and selling them is not their primary mission. The CMDB could be an opportunity for the company to reach out to a different market of the same industry, thus adding a new activity to its portfolio and unlocking potential new revenue. With its network of clients already established, Prime Resources could secure another stable source of income. Moreover, its geographic position in the Milwaukee area makes it easier to find bigger companies that have enough resources to invest in such a product.

To sum up, the strategy of Prime Resources regarding the application is to advertise it to IT managers and potential investors to give them the vision that an online CMDB adopted widely in a company will work. Resulting from the investment, the product will then be rebranded in a new and more solid form to be sold to a wider market. Meanwhile, the simpler version can be made available online as an open-source project, thus reaching to a wider audience of potential clients or investors. We will now study the future of the application regarding the features that could be added to the software.

9 Future of the Application

This part covers the different features that will be added to the application. Some will be developed in the near future, while others will be added once investors are secured for sponsoring the product. First, we focus on the functionalities added in the upcoming months.

The first of these features is to develop a panel dedicated to the human resources. This panel, like the previous ones, includes a tree using jsTree, the same jQuery plugin previously used. Its goal is to display a hierarchy of each department with the positions and the employees. This panel helps access all the information regarding an employee (hiring date, salary...) and link them to positions which are organized according to departments. Each position also has properties of its own (salary range, budget...).

The next feature is about a search made by the user in the application. The goal is, like a search browser, to display all the elements matching a set of keywords. The results are displayed on a webpage with a link giving the user access to the properties of the element on its own page. For example, if the result is an application, the user is directed to the application graph.

The last immediate feature is adding the possibility to snapshot the current state of the application to a file saved on the user's computer and load a previously saved file. This gives the opportunity to review the different changes made to the system since the last snapshot was taken. It gives the advantage to compare the evolutions for the different components.

Next to these previous improvements to the global application, there are a couple of other features that will be added once the project will have investors. At this point in the development process, the notion of minimally viable product will no longer be necessary, since the goal will be to expand the features for a business.

The first step is to broaden the view regarding the users. The features of administrator is expanded according to the position of an employee in the company. For example, a Linux manager is able to modify only the items belonging to his category, such as the different Linux configuration items, servers, and applications. They are still able to view all the other categories, like databases or networks, but do not have the possibility to modify them. This ensures that only knowledgeable users can modify the items they know about. There is not a simple difference between users and administrators, each user being an administrator of a part of the application at some point.

Another feature that can be nice to develop is a three dimensional view of a cabinet. The user, helped with the mouse, moves the cabinet and sees it with the same texture as in real life. The servers in this cabinet can be rendered as real servers as well, and not just as boxes like in the current version. On the same idea, the data center map is rendered in 3D, taking into account the height of the cabinet. Using such graphic designs is more appealing to the user and a great marketing tool for the product. It also enhances the importance of representing the reality on the computer as close as possible.

Something really important regarding the security also needs to be added. When the website is made available online, an HTTPS protocol is necessary to ensure that the data provided in the forms is securely sent to the server. Indeed, although SQL injections are not possible in the current website, malicious users could still retrieve data sent to the server. This must not happen as it would compromise the users' data. Right now, the application only runs locally, so it is not an issue, but it needs to be implemented before it is made available online.

The last concern to address before releasing the application regards the license content. Since some open-source tools were used, it is necessary to check their license terms and make sure it is possible to use them in the commercial release. Most of the time, it is not an issue; however, if the external tool cannot be used, it is necessary to either buy a license from an equivalent product, or develop a plugin specific for the application to replace these open-source tools. This could greatly impact the code already written regarding the client side. There will surely be other features to add, but these are the main ones that are very likely to be implemented in the future.

10 Work Methods and Project Management

In this final part, I will talk about the different work methods applied during the project, along with how the project was managed.

10.1 Gantt Diagram

One of the first things my manager and I did during the first weeks of the internship was to think about a schedule with the different steps necessary to implement the functions of the application. Although my manager did not require me to write a Gantt diagram, I decided to create one showing the real progress of the development, shown below. To create it, I used a free software tool called GanttProject [31].

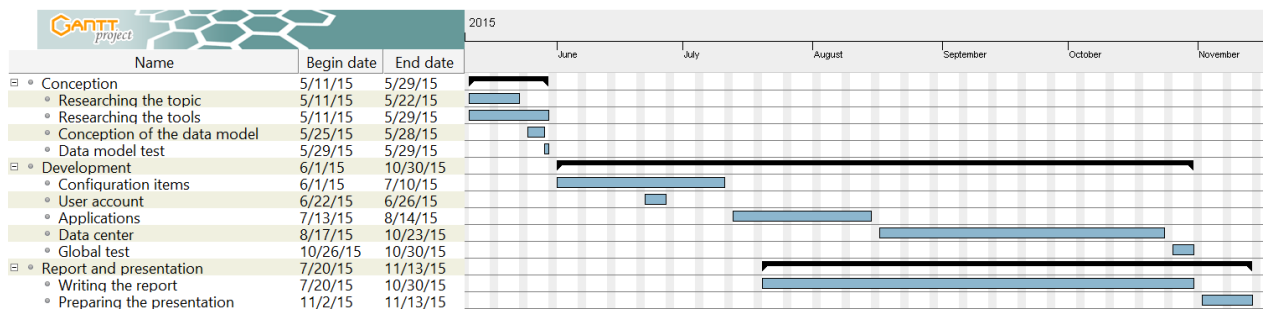


Figure 24: Gantt diagram

The project is divided into three main parts. The first part relates to the conception of the application, and took approximately one month. These steps were extremely important; indeed, they allowed me to research about CMDBs, write the first drafts of a solution, conceive the data model, and select the technologies to use for the development. For the data model, a test was conducted with a local MySQL database to make sure it was coherent and worked smoothly.

The next step in the diagram is the core development of the application. It is divided according to each panel of the application, starting with the configuration items. The administrator view was conceived first, then it was necessary to create the user account features in order to develop the user view. We can indeed see on the graph that the user account task, which takes approximately one week, was developed along with the configuration items features at some point. Each of the features took approximately a month to be developed, except for *user account* and *data center*, the latter being the most complex in the application. Although tests were run along the development of each feature, a global test was necessary to make sure the whole application worked smoothly. The test also included a simulation with a potential administrator and user.

The last part of the Gantt diagram shows the steps to write the thesis and prepare the presentation. It spans across a long period; however, contrary to the application development, this task was done part-time. It also followed closely the progress of the application development. This Gantt diagram shows the two deliverables resulting from the project, presented in the next section.

10.2 Deliverables

There were two main deliverables for this project: the product and the thesis report. The first one is delivered to the company in form of a compressed zip file containing all the source code, libraries, images, and other files used to develop the application. Once unzipped, the files can be placed on a local server and immediately run by opening any of the webpages. Although there was no official deadline for this delivery, it was supposed to be done by the end of October. This delivery will be the minimally viable product to be shown to investors.

The second deliverable includes the thesis, detailing all the aspects of the project. Due on November 7, it will be given to the university supervisor, after a review by my industrial supervisor. The thesis will also be used by Prime Resources to explain the solution in detail and give the incentive for developing the product. Eventually, some of its content could be reused in another technical document, explaining the code in further details, which could be studied by IT managers from potential clients.

10.3 Meetings with the Industrial Supervisor

The meetings were an important part of the development process throughout the internship. They not only helped me to get more insight regarding the application, but also guided me in the right direction and deepened my understanding of the CMDB concepts.

When the internship started, my supervisor Ritch Houdek and I met often, at least twice a week to discuss the conception of the project. My first meeting occurred the day after I arrived in the United States, at which time Ritch explained the principal goals of the projects and his vision of a simple CMDB and how it could be achieved. He gave me some suggestions and tracks to follow, but ultimately he wanted me to study the issues and come up with my own solutions; then we could talk about it and see if it would work in the project. It was an enriching opportunity as it widened my creativity and analysis skills, which an engineer must develop in any company.

In the example of the data model, we often exchanged about the different features it would include, so it was a very creative aspect allowing me to inject some of my own conception solutions in the

application. After the first month, the meeting pace decreased a little bit and we usually met once a week, although we could meet at almost any time if there was an important subject to talk about.

This close follow-up was greatly beneficial, as it helped me to be guided in the right direction for the entire internship. It also gave me the opportunity to talk about my solutions and explain them to my supervisor.

10.4 Source Code Management

Regarding the code management, my supervisor let me chose the tool that would be the best for me. I used Bitbucket, a hosting service powered by Atlassian [32]. It uses the technologies of Git and Mercurial for revision control. Thus, it helps keep track of the different modifications in the program and allows viewing the different changes since the start of the project. The code is hosted online in a private repository, which has the following advantages:

- Immediate access to the latest project version stored online
- Possibility to undo the local changes and load one of the previous versions stored online
- If two or more people work on the project at the same time, they can both work locally and then upload their changes online.
- Finally, it is easy to upload. Once Git or Mercurial is installed on the computer, the commands to upload the changes can be called from the command line.

Having previously worked with Bitbucket, I thought it would be the best suitable tool to ensure the project was versioned and available no matter what would happen to my computer. There are other versioning tools on the market, but Bitbucket has the advantage to allow private repositories without any subscriptions. It is for all these reasons that I chose Bitbucket to store my code.

11 Conclusion

Throughout this internship, I was able to develop a cloud-based Configuration Management Database aiming at small- and medium-sized businesses. Coded as a minimally viable product using standard web technologies (HTML5, CSS3, Bootstrap, JavaScript, PHP, and jQuery) and a MySQL database, the product was developed with investors in mind, thus focusing the whole development process on the most important features that a cloud-based CMDB should have.

These features will allow the users to manage the configuration items of the company with a set of useful and simple tools. The first of these features will help manage the items by organizing them into classes and assigning properties and values for each of them. A second feature will show the applications used within the company with a graph showing the interactions between the applications and the configuration items.

The last feature allows the user to manage a view of the company's datacenters. Using a 2D map, it is possible to add cabinets. The user is also able, after selecting a cabinet, to access to a closer view allowing to see the different servers installed there. This comes in handy especially when someone wants to know which configuration items are located on a specific cabinet. These servers can also be selected and, through a contextual menu, the user can access to the properties of the corresponding configuration item as well as a list of all the applications within the company that use this specific server.

The software is also topped with a two-view user interface: standard user and administrator. While the first one has a read-only view on the application, the latter can directly interact with the data and update the database as well. This allows a more secure way regarding who has the rights to modify the data. Thus, this software perfectly matches the client's requirements, which were to develop a CMDB prototype allowing to manage the different configuration items in a company. On this note, I can say that the objectives were accomplished.

However, there could still be room for improvement. For example, a new feature that could be developed would take into account the user's position within the company. When they want to update some data, they can only do so for the department in which they work, and would have a user view on the other items. Another possible feature would be to render a 3D map of the datacenters room, allowing a representation of the cabinets closer to reality. The height of the cabinet could then be represented, as well as the shape and the different textures, thus making the application more realistic.

Finally, regarding my personal progress, this internship, done according to my training as a software engineer at TELECOM Nancy, helped me understand the whole process of writing an application in the business world, from the conception to the development. It allowed me to deepen my understanding on mainstream web technologies that are widely used on the market, like jQuery or PHP.

This internship not only strengthened my skills as a developer, but also opened my mind regarding the business world and how a product can be released on the market. It also helped me discover the world of start-up companies which I never encountered before. Throughout my internship, I was able to understand the economic strategy that such companies implement, notably with the notion of minimally viable product. Moreover, it gave me an overview of the spirit of entrepreneurship thanks to the people I worked with.

The different aspects of this internship showed me more closely how an economic strategy can impact the conception and the development of a whole product. This, as well as the fact that I was able to contribute to the different stages of the product's development, was a valuable experience that prepared me for my career as an engineer.

12 References

- [1] Prime Resources LLC, "What We Do," May 2015. [Online]. Available: <http://www.primeresources-llc.com/index.php>. [Accessed 1 September 2015].
- [2] Google, "Google Maps," 2015. [Online]. Available: <https://www.google.com/maps/place/302+Main+St,+Sussex,+WI+53089/@43.087526,-88.2219205,11.17z/data=!4m2!3m1!1s0x8805ab63d7631e51:0xdda98752871ed302?hl=en>. [Accessed 13 November 2015].
- [3] Internal Revenue Service, "Limited Liability Company (LLC)," 5 August 2015. [Online]. Available: <http://www.irs.gov/Businesses/Small-Businesses-&Self-Employed/Limited-Liability-Company-LLC>. [Accessed 1 September 2015].
- [4] Hudson Business Lounge, "The Hudson," 2014. [Online]. Available: <http://www.hudson-business-lounge.com/>. [Accessed 1 September 2015].
- [5] AXELOS Global Best Practice, "ITIL - IT Service Management," [Online]. Available: https://www.axelos.com/best-practice-solutions/itil.aspx?utm_source=itil-officialsite&utm_medium=redirect&utm_campaign=redirects. [Accessed 1 September 2015].
- [6] M. Rouse, "Configuration management database (CMDB) definition," December 2006. [Online]. Available: <http://searchdatacenter.techtarget.com/definition/configuration-management-database>. [Accessed 1 September 2015].
- [7] Riccardo, "6 Open Source CMDB," Linuxaria, 19 February 2011. [Online]. Available: <http://linuxaria.com/article/6-cmdb-open-source>. [Accessed 1 September 2015].
- [8] OneCMDB, "Home Page," 2 April 2012. [Online]. Available: http://onecmdb.org/wiki/index.php?title=Main_Page. [Accessed 1 September 2015].
- [9] Capterra, "Top ITSM Software Products," [Online]. Available: <http://www.capterra.com/itsm-software/>. [Accessed 1 September 2015].
- [10] W. Fenton, "Top 5 Rules for Creating User Friendly Mobile Apps," 31 May 2012. [Online]. Available: <http://www.cmswire.com/cms/customer-experience/top-5-rules-for-creating-user-friendly-mobile-apps-015841.php>. [Accessed 1 September 2015].

- [11] S. Blank, "Perfection By Subtraction – The Minimum Feature Set," 4 March 2010. [Online]. Available: <http://steveblank.com/2010/03/04/perfection-by-subtraction-the-minimum-feature-set/>. [Accessed 1 September 2015].
- [12] S. Phillips, "A brief history of Facebook," The Guardian, US Edition, 27 July 2007. [Online]. Available: <http://www.theguardian.com/technology/2007/jul/25/media.newmedia>. [Accessed 1 September 2015].
- [13] Oracle, "Java and Google Chrome Browser," [Online]. Available: <https://java.com/en/download/faq/chrome.xml>. [Accessed 2 September 2015].
- [14] Google, "NPAPI deprecation: developer guide," January 2015. [Online]. Available: <https://www.chromium.org/developers/npapi-deprecation>. [Accessed 2 September 2015].
- [15] S. J. Vaughan-Nichols, "Most popular US web browsers, according to the federal government," ZDNet, 26 March 2015. [Online]. Available: <http://www.zdnet.com/article/the-most-u-s-popular-web-browsers/>. [Accessed 2 September 2015].
- [16] Clicky Web Analytics, "Web browsers (US marketshare)," 4 September 2015. [Online]. Available: <https://clicky.com/marketshare/us/web-browsers/>. [Accessed 4 September 2015].
- [17] A. Smith, "U.S. Smartphone Use in 2015," Pew Research Center, 1 April 2015. [Online]. Available: <http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/>. [Accessed 2 September 2015].
- [18] Salesforce, "Why Move to the Cloud? 10 Benefits of Cloud Computing," 2015. [Online]. Available: <http://www.salesforce.com/uk/socialsuccess/cloud-computing/why-move-to-cloud-10-benefits-cloud-computing.jsp>. [Accessed 2 September 2015].
- [19] Apache Friends, "XAMPP," 2015. [Online]. Available: <https://www.apachefriends.org/index.html>. [Accessed 5 September 2015].
- [20] phpMyAdmin contributors, "Bringing MySQL to the web," 2015. [Online]. Available: <https://www.phpmyadmin.net/>. [Accessed 5 September 2015].
- [21] jsTree, "What is jsTree?," [Online]. Available: <https://www.jstree.com/>. [Accessed 5 September 2015].
- [22] Almende B.V., "vis.js," 2015. [Online]. Available: <http://visjs.org/>. [Accessed 5 September 2015].

- [23] Appcelerator, Inc., "Aptana Studio 3," 2014. [Online]. Available: <http://www.apтана.com/index.html>. [Accessed 5 September 2015].
- [24] D. Marjanovic, "PDO vs. MySQLi: Which Should You Use?," tuts+, 21 February 2012. [Online]. Available: <http://code.tutsplus.com/tutorials/pdo-vs-mysqli-which-should-you-use--net-24059>. [Accessed 9 September 2015].
- [25] W. Fote, "What is Ajax and Where is it Used in Technology?," Segue Technologies, 13 March 2013. [Online]. Available: <http://www.seguetech.com/blog/2013/03/12/what-is-ajax-and-where-is-it-used-in-technology>. [Accessed 10 September 2015].
- [26] Google, "Device Mode & Mobile Emulation," 2015. [Online]. Available: <https://developer.chrome.com/devtools/docs/device-mode>. [Accessed 12 September 2015].
- [27] The jQuery Foundation, "About jQuery UI," 2015. [Online]. Available: <http://jqueryui.com/about/>. [Accessed 16 September 2015].
- [28] D. Gandy, "Font Awesome, the iconic font and CSS toolkit," 2015, [Online]. Available: <https://fortawesome.github.io/Font-Awesome/>. [Accessed 22 September 2015].
- [29] Ocal, "Server Rack Cabinet Clip Art," 12 October 2010. [Online]. Available: <http://www.clker.com/clipart-server-rack-cabinet.html>. [Accessed 26 September 2015].
- [30] B. Harzog, "Is the CMDB irrelevant in a virtual and cloud based world?," 2 June 2010. [Online]. Available: <https://www.virtualizationpractice.com/is-the-cmdb-irrelevant-in-a-virtual-and-cloud-based-world-5726/>. [Accessed 26 September 2015].
- [31] D. Barashev, "GanttProject," 2015. [Online]. Available: <http://www.ganttproject.biz/>. [Accessed 2 October 2015].
- [32] Atlassian, "Bitbucket," 2015. [Online]. Available: <https://bitbucket.org/>. [Accessed 2 October 2015].
- [33] w3schools.com, "SQL Injection," Refsnes Data, 2015. [Online]. Available: http://www.w3schools.com/sql/sql_injection.asp. [Accessed 8 September 2015].
- [34] Wikipedia, "Hash function," 7 August 2015. [Online]. Available: https://en.wikipedia.org/wiki/Hash_function. [Accessed 8 September 2015].
- [35] S. K. Bansal, "Securing Passwords with Bcrypt Hashing Functions," The Hacker News, 10 April 2014. [Online]. Available: <http://thehackernews.com/2014/04/securing-passwords-with-bcrypt-hashing.html>. [Accessed 8 September 2015].

List of Figures

Figure 1: Maps showing the location of Prime Resources	13
Figure 2: Earlyvangelist pyramid	26
Figure 3: Web browsers market shares in the United States	29
Figure 4: The 10 benefits of cloud computing	30
Figure 5: User table	37
Figure 6: Data model	38
Figure 7: Login screen.....	41
Figure 8: Sign up screen	42
Figure 9: Account information page.....	45
Figure 10: Screenshot from the interface	52
Figure 11: Screenshot from the interface as it would appear on an iPhone6 Plus	54
Figure 12: Toggled menu as it would appear on an iPhone6 Plus	54
Figure 13: Configuration Items management page screenshot	55
Figure 14: View of the right panel when a class is selected.....	56
Figure 15: Adding a property	59
Figure 16: Configuration item view with no properties	60
Figure 17: Context menu according to the node type.....	61
Figure 18: Applications management page screenshot	63
Figure 19: Data center page screenshot.....	66
Figure 20: Form for creating a data center	67
Figure 21: Data center page with the cabinet view	70
Figure 22: Applications popup	72
Figure 23: Form to add a server to a cabinet	73
Figure 24: Gantt diagram.....	81

Figure 25: Principles of a hash function.....	97
---	----

Glossary

Minimally Viable Product (MVP): strategic way of developing a product with only the minimum features required for the product to work. A MVP is usually released to investors and potential customers in order to see how the market would react. It is more about selling the idea behind the product rather than selling the product itself.

Prepared statement: database feature used to execute the same query at multiple times with different parameters and a higher efficiency. Parameters in the query are not specified but replaced by variables. The database then compiles the query and optimizes it. Then, when the application specifies or *binds* the variables to the database, the query is executed.

SQL injection: technique used by hackers to attack data-driven applications. It consists of inserting SQL statement inside a field sent to the server in order to interact with the database (for example to get the user's email address or login information).

Addenda

I. SQL injections and how to prevent them

A SQL injection is a technique used by malicious users to destroy the database or get data from it. If not secured, they can be highly dangerous as they compromise the security of the application. A SQL injection can be created using a text field, for example on a search field or a login page. The W3 school website provides a good example of a typical SQL injection [33].

Let's consider a login page where the user is requested to enter his user name and password.

User Name:

Password:

To process the data entered by the user without any security features, a typical request would look like this:

```
uName = getRequestString("UserName");  
uPass = getRequestString("UserPass");
```

```
sql = "SELECT * FROM Users WHERE Name = '" + uName + "' AND Pass = '" +  
uPass + "'"
```

If the user is honest, it would work fine. But someone with bad intentions could get the names and passwords by inserting a SQL query inside one of the fields, like `" or ""="`.

The SQL statement executed by the server then becomes:

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

This statement will always be true, and then the server will return the list of all users with their names and passwords. This is the reason why it is necessary to secure the inputs sent by the user to the server. To do this, some steps need to be added into the PHP file. In Listing 29 taken directly from the CMDB application, the user is asked to rename a field.

Listing 29: Renaming a node name

```
$sql_graph = 'update graph
set name = :name
where application_id = :id
and type = "app"';
$stmt_graph = $conn -> prepare($sql_graph);
$stmt_graph -> execute(array(':name' => $name, ':id' => $id));
$row_graph = $stmt_graph -> fetchAll(PDO::FETCH_ASSOC);
```

We can notice that the user's inputs are not directly inserted into the query. First, the SQL statement is prepared, it means the system parses it once and then it can be executed many times with different parameters without the need to parse the query another time. Then, it is executed with the parameters entered by the user. If these are invalid or potentially dangerous as previously seen, then the query will not be executed and there will be no risk of data corruption.

This technique was used for the whole development of the application to maintain a secure application, safe to use, both for users and administrators.

II. Hash functions

When securing a password to store in a database, it is common practice to use a hash function. The purpose of a hash function is to encode the passwords in order to prevent a maleficent user who would get in possession of the hash to retrieve the original password. The following schema shows how a hash function works [34].

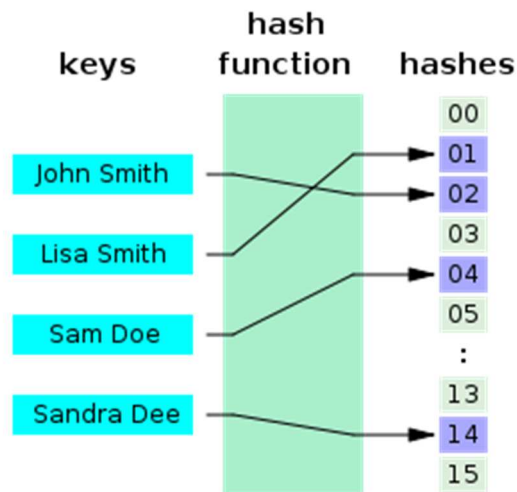


Figure 25: Principles of a hash function

The result of this function – the hash – is then stored in the database instead of the real password. There are currently different algorithms that produce a hash. Some are more secured than others. PHP offers different hash functions. The one used in the application is based on the Bcrypt algorithm, a “method of hashing passwords [that] is solid enough for most web applications that stores users’ passwords and other sensitive data” [35].

III. Code to create the tree using jsTree

```
function customMenu($node) {
    var tree = $("#tree").jstree(true);
    var items = {
        "createfolder" : {
            "separator_before" : false,
            "separator_after" : false,
            "label" : "New class",
            "action" : function(obj) {
                $node = tree.create_node($node, {
                    type : "folder"
                });
            }
        },
        "createitem" : {
            "separator_before" : false,
            "separator_after" : false,
            "label" : "New configuration item",
            "action" : function(obj) {
                $node = tree.create_node($node, {
                    type : "file"
                });
            }
        },
        "rename" : {
            "separator_before" : false,
            "separator_after" : false,
            "label" : "Rename",
            "action" : function(obj) {
                tree.edit($node);
            }
        },
        "remove" : {
            "separator_before" : false,
            "separator_after" : false,
            "label" : "Remove",
            "action" : function(obj) {
                tree.delete_node($node);
            }
        }
    };

    if ($node.type === 'default') {
        delete items.createitem;
        delete items.remove;
        delete items.rename; }

    if ($node.type === 'file') {
        delete items.createfolder;
        delete items.createitem;
    }

    return items;
}
```

```

$(document).ready(function() {
    $("#tree").jstree({
        "core" : {
            "check_callback" : function(operation, node) {
                if (operation === 'delete_node') {
                    if (node.children.length == 0) {
                        return confirm("Are you sure you want to delete this node?");
                    } else {
                        alert("Folder not empty! Please delete all of its children first!");
                        return false;
                    }
                }
            },
        },

        "data" : {
            "type" : "POST",
            "url" : function(node) {
                return node.id === '#' ? 'db_treeLoaderRoot.php' : 'db_treeLoader.php';
            },
            "data" : function(node) {
                return {
                    'id' : node.id
                };
            }
        },

        "types" : {
            "file" : {
                "icon" : "img/file-icon.png",
                "valid_children" : []
            },
            "folder" : { }
        },

        "contextmenu" : {
            "items" : customMenu
        },

        "plugins" : ["contextmenu", "json_data", "massload", "search", "sort", "state",
            "themes", "types", "ui", "unique", "wholerow"]
    });
});

```

IV. Code to create a property in the database

```
<?php

include 'db_connect.php';

$array_ci = array();

function getConfigItems($conn, $class_id) {
    global $array_ci;

    try {
        $sql_ci = 'select id from config_item
        where class_id = :class_id';
        $stmt_ci = $conn -> prepare($sql_ci);
        $stmt_ci -> execute(array(':class_id' => $class_id));
        $row_ci = $stmt_ci -> fetchAll(PDO::FETCH_ASSOC);

        if (!empty($row_ci)) {
            foreach ($row_ci as $i) {
                array_push($array_ci, $i['id']);
            }
        }

        $sql_parent = 'select id from class
        where parent_id = :class_id';
        $stmt_parent = $conn -> prepare($sql_parent);
        $stmt_parent -> execute(array(':class_id' => $class_id));
        $row_parent = $stmt_parent -> fetchAll(PDO::FETCH_ASSOC);

        if (empty($row_parent)) {
            return $array_ci;
        } else {
            foreach ($row_parent as $i) {
                return getConfigItems($conn, $i['id']);
            }
        }
    } catch(PDOException $e) {
        echo $e -> getMessage();
    }
}

function createProperty($conn) {
    try {
        $name = $_POST['property-name'];
        $value_type = $_POST['property-type'];
        $class_id = $_POST['class_id'];
        $tab = $_POST['tab'];

        $sql1 = 'insert into property
        (name, value_type, tab)
        values
        (:name, :value_type, :tab)';
```

```

$stmt1 = $conn -> prepare($sql1);
$stmt1 -> execute(array(':name' => $name, ':value_type' => $value_type,
':tab' => $tab));
$row1 = $stmt1 -> fetchAll(PDO::FETCH_ASSOC);

echo "Property created!";
echo "<br>";

$sql2 = 'insert into map_class_property
(class_id, prop_id)
values
(:class_id, (select id from property
order by id desc
limit 1));';
$stmt2 = $conn -> prepare($sql2);
$stmt2 -> execute(array(':class_id' => $class_id));
$row2 = $stmt2 -> fetchAll(PDO::FETCH_ASSOC);

echo "Mapping complete!";
echo "<br>";

$iterate = getConfigItems($conn, $class_id);

foreach ($iterate as $row) {
    $sql3 = 'insert into property_value
(property_id, config_id, str_value, date_value, float_value)
values
((select id from property
order by id desc
limit 1), :config_item, NULL, NULL, NULL);';
    $stmt3 = $conn -> prepare($sql3);
    $stmt3 -> execute(array(':config_item' => $row));
    $row3 = $stmt3 -> fetchAll(PDO::FETCH_ASSOC);
}

echo "Values inserted!";

} catch(PDOException $e) {
    echo $e -> getMessage();
}
}

createProperty($DB_con);
?>

```


Résumé

L'objectif du projet était de développer un prototype de CMDB basé dans le cloud pour les petites et moyennes entreprises. Utilisant HTML, CSS, PHP, JavaScript, et jQuery, le site web permet de gérer une base de données MySQL pour sauvegarder les différents items de configuration de l'infrastructure de la compagnie.

Deux vues différentes, utilisateur et administrateur, sont disponibles, la dernière permettant d'interagir avec les données au travers de l'application. Le site web a trois fonctions principales: permettre la création et l'édition des items de configuration, afficher un arbre montrant l'interaction entre les différentes applications utilisées par le système et les items de configuration, et enfin afficher les data centers et chaque cabinet avec les serveurs installés dessus.

Avec un design responsif, la plupart des appareils mobiles sont compatibles pour afficher l'application dans une vue agréable à l'utilisateur, accédant ainsi au plein potentiel d'une application basée dans le cloud.

Mots-clés: CMDB, item de configuration, data center, ITIL

Abstract

The objective of the project was to develop a cloud-based Configuration Management Database designed as a prototype for small- and medium-sized businesses. Using HTML, CSS, PHP, JavaScript, and jQuery, the website allows to manage a MySQL database recording the different configuration items of the company's infrastructure.

Two different views, user and administrator, are available, the latter being able to interact with the data through the application. The website has three main functions: allowing the creation and edition of configuration items, displaying a tree showing the interaction between the different applications used in the system and the configuration items, and viewing the data centers and each cabinet with the servers installed on them.

With a responsive design, most mobile device are compatible to view the application in a user-friendly display, thus reaching to the whole potential of a cloud-based application.

Keywords: CMDB, configuration item, data center, ITIL