Used Technologies

6 Technologies used

In this part, I will 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. [18]

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. [19]

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. [20]

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. [21]

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. [22]

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 will 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 will 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 4 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).

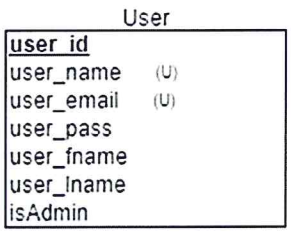


Figure 4: User table

Regarding the application data, figure 5 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.

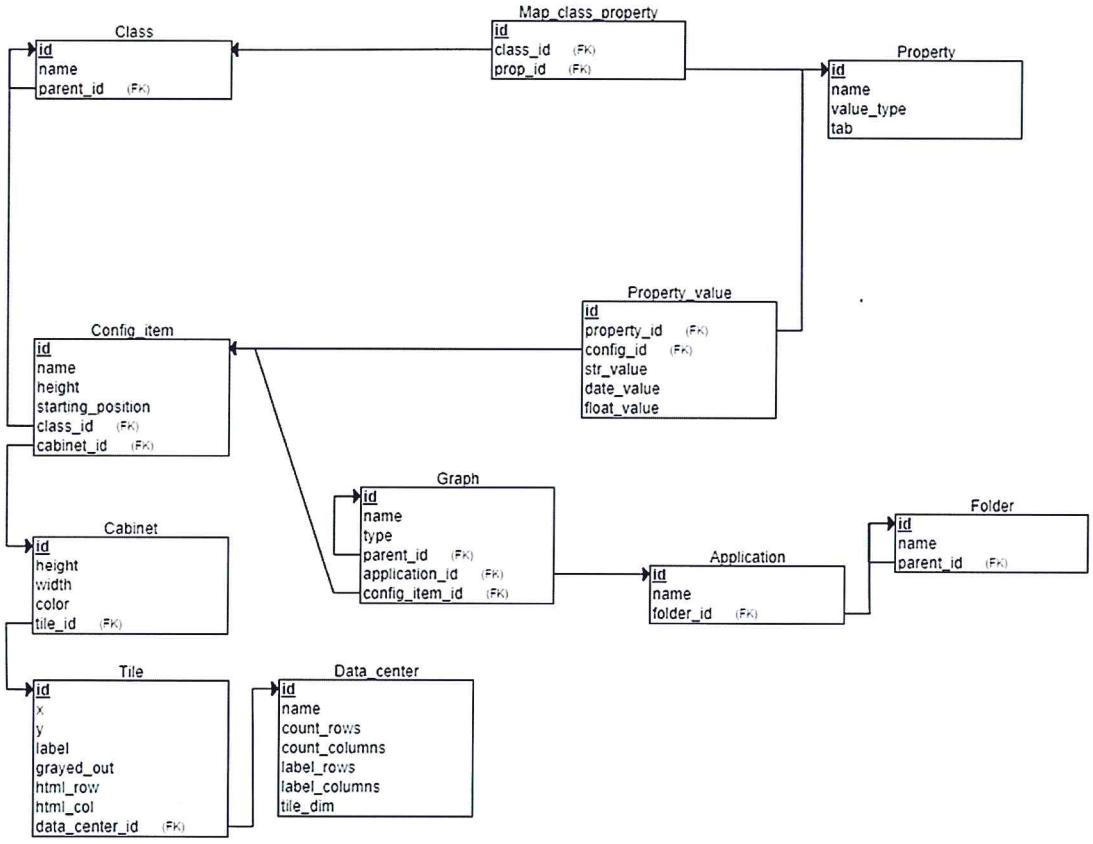


Figure 5: Data model

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 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 we have seen the different components of the data model, we 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 6.

Figure 6: Login screen

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.

The following image shows how the application would look like on an iPhone6 Plus.

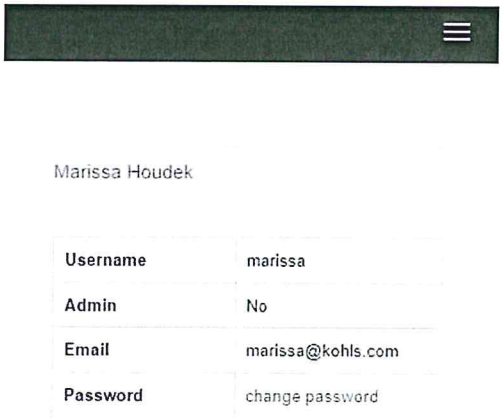


Figure 10: Screenshot from the interface as it would appear on an iPhone6 Plus

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 (shown on the screenshot below).

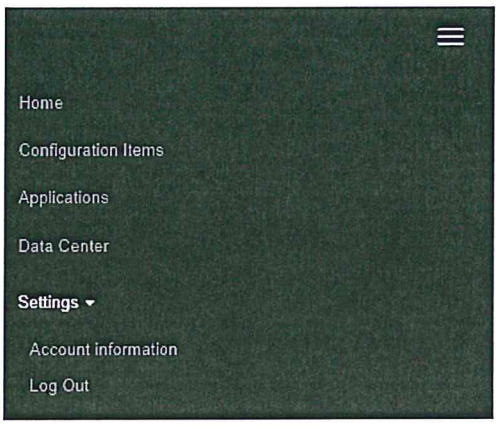


Figure 11: Toggled menu as it would appear on an iPhone6 Plus

Finally, the settings menu keeps its dropdown property, and reveals the additional items when clicked upon.

To check how the application would look like on a specific mobile device, I used Google Chrome Mobile Emulation. [25] 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 we have seen the different characteristics of the user interface, let's focus more specifically on the different functionalities of the application.

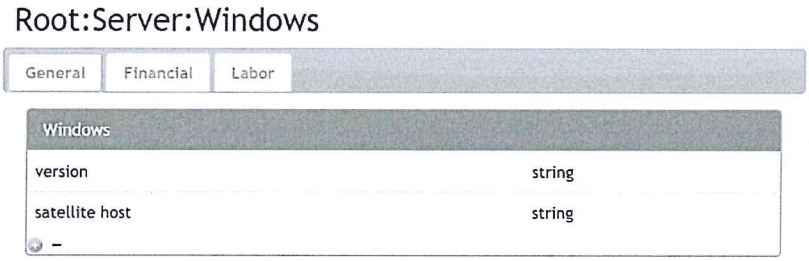


Figure 13: View of the right panel when a class is selected

The tree is rendered through a jQuery plugin, jsTree. [20] 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. The code snippet shows how these steps are implemented.

Listing 1. Definition of jsTree in the code

Numéroter les listing et référencer dans le texte au même titre que les figures

```
[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

7.5 Application features

In this part, I ~~will~~ 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.

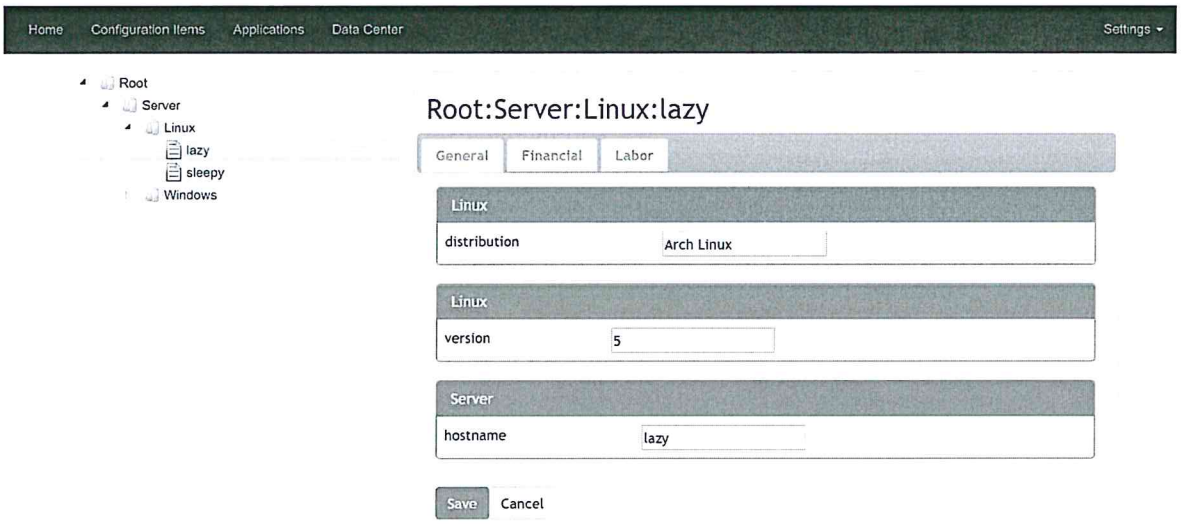


Figure 12: Configuration Items management page screenshot

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 12, 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 13. The only exception is when the user clicks on the root node. In that case, the right panel is left blank.

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 part presenting the implementation of the solution, we were able to see the different features that the cloud-based CMDB provides to the user.

We now can move forward to studying the technological progress and economic strategy for the CMDB application.

8 Technological progress and economic strategy

In this part, I will 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 [29], 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 our technology brings is making a tool available to the market no matter which device you use 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, as we will see in the following part.

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 will cover 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 will focus on the functionalities added in the upcoming months.

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

The next feature will be 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 will be 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 will be directed to the application graph.

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

Another feature that would be nice to develop is a three dimensional view of a cabinet. The user would be able, using the mouse, to move the cabinet and see it with the same texture as in real life. The servers in this cabinet would 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 would be rendered in 3D, taking into account the height of the cabinet. Using such graphic designs would be more appealing to the user

and a great marketing tool for the product. It would also enhance the importance of representing the reality on the computer as close as possible.

Something really important regarding the security will also be added. When the website will be made available online, an HTTPS will be 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 will not be an issue; however if it is not possible, it will be 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.

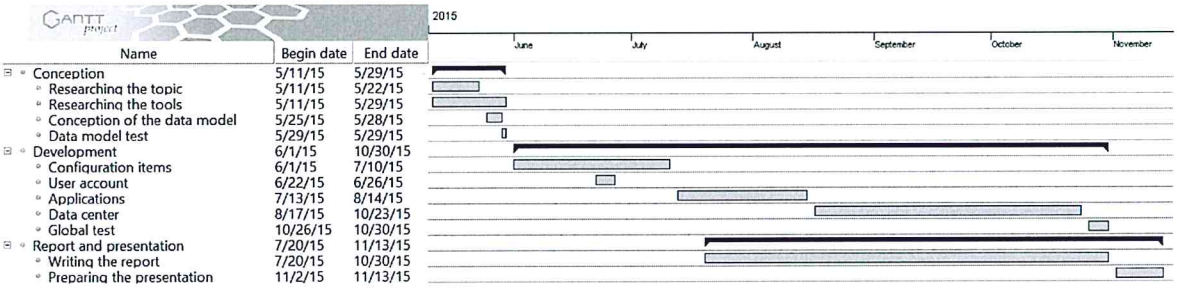
I will now talk about the project management aspect of the internship.

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. [30]



Name	Begin date	End date
Conception	5/11/15	5/29/15
Researching the topic	5/11/15	5/22/15
Researching the tools	5/11/15	5/29/15
Conception of the data model	5/25/15	5/28/15
Data model test	5/29/15	5/29/15
Development	6/1/15	10/30/15
Configuration items	6/1/15	7/10/15
User account	6/22/15	6/26/15
Applications	7/13/15	8/14/15
Data center	8/17/15	10/23/15
Global test	10/26/15	10/30/15
Report and presentation	7/20/15	11/13/15
Writing the report	7/20/15	10/30/15
Preparing the presentation	11/2/15	11/13/15

Figure 23: 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, that I will present now.

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.

I will now talk about the meetings with my industrial supervisor which were defining in how the conception and development were conducted.

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. [31] 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 of 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 before, 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.

~~This final part concludes the report and brings us to the conclusion.~~

Bonne conclusion !
Une amélioration possible : regrouper quelques paragraphes pour éviter les ~~para~~ trop courts (2 phrases) lorsque l'idée développée est la même.

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.