UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CRISTIANO MEDEIROS DALBEM

Reengineering of a merchandising web application

Final Report presented in partial fulfillment of the requirements for the degree of Bachelor of Computer Science

Advisor: Prof. Marcelo Soares Pimenta
UFRGS, Porto Alegre, Brasil

Coadvisor: François Bérard
Univ. Grenoble Alpes, Grenoble, France

Porto Alegre
2015

“We are told that talent creates its own opportunities. Yet, it sometimes seems that intense desire creates not only its own opportunities, but its own talents as well.”

– Bruce Lee, Tao of Jeet Kune Do

“Muad'Dib learned rapidly because his first training was in how to learn. And the first lesson of all was the basic trust that he could learn. It's shocking to find how many people do not believe they can learn, and how many more believe learning to be difficult. Muad'Dib knew that every experience carries its lesson.”

― Frank Herbert, Dune

ACKNOWLEDGEMENTS

I would like to thank my university UFRGS and the wonderful professors I had who were highly inspirational people and taught me beyond my expectations. During these several years, the classrooms were my second home, and my classmates were my second family. Thanks so much as well to ENSIMAG, which hosted me as an Exchange student in Grenoble, France, for one year and half. There I lived some of the most enriching experiences of my life.

I would like to thank as well Marcelo Pimenta, my tutor in Brazil, and François Bérard, my tutor in France. Both are truly great examples of inspirational professors who shared with me valuable teachings, as well as contributed to the very shaping of this project with their experience.

I thank my internship supervisor Claude Delaye for being patient with my accent when speaking French and being an open and flexible person to work with. Most of all, he always believed in my work while, at the same time, teaching me how to have a critical view of what I did and ensuring that I reached my greatest potential.

Last but not least, I thank my family who has always been and always will be my safe harbor, the base case of my induction, my tonal center. My accomplishments are theirs as much as they are mine.

RESUMO

Este projeto foi desenvolvido no âmbito de um programa de dupla-diplomação entre Brasil e França, durante um estágio no grupo de P&D da KLEE Group, empresa líder em merchandising na França. No contexto de suas aplicações para gerência de prateleiras e mapas para lojas e supermercados, KLEE queria otimizar a ergonomia de seus módulos e trabalhar em direção a uma versão para plataformas móveis.

O objetivo deste projeto foi então a reengenharia do módulo de merchandising, incluindo o estudo, definição e implementação deste, servindo-se de boas práticas de programação e tecnologias Web modernas como HTML5, CSS3 e JavaScript. Um grande esforço foi feito no processo de refatoração, otimização e documentação das soluções existentes, assim como no desenvolvimento de novas soluções.

Este documento começa apresentando o contexto do trabalho, suas motivações e um resumo do sistema web existente, alem de um resumo das tecnologias e técnicas envolvidas no projeto. Após, discutimos em detalhes o ponto principal do trabalho, i.e. o desenvolvimento de um novo módulo de visualização de planogramas. Antes de concluirmos, apresentamos detalhes técnicos de outras contribuições feitas ao sistema. Finalmente, concluímos discutindo os aprendizados de uma ótica técnica, pessoal e profissional, e apresentamos um resumo das contribuições deste trabalho e possibilidades de trabalhos futuros.

Palavras-chave: Web. JavaScript. Front-End. SVG. HTML. CSS. Planograma. Engenharia de Software. Mobile. Ext JS.

ABSTRACT

This project was developed in the scope of a double-degree program between Brazil and France, during an internship in the R&D team of KLEE Group, the leading company in merchandising in France. In the context of their applications for the management of shelves and plans of stores, KLEE wanted to optimize the ergonomy of their modules and work towards a portable version for mobile platforms.

The objective of this project was the reengineering of the merchandising module, including its study, definition and implementation, taking advantage of good programming practices and modern Web technologies like HTML5, CSS 3 and JavaScript. A big effort was done in refactoring and optimizing their existing solutions, and new ones were also developed in the process.

This document starts by presenting the context of the work, such as its motivations and an overview of the overlying system, as well as an overview of the technologies and techniques involved in this project. Then we take a deep look at the main subject, which is the development of a new planogram viewer module. Before concluding, we present some interesting technical details about additional contributions done to the system. Finally, we conclude by discussing technical, personal and professional lessons learned during this work, an overview of the contributions of this work and possibilities of further work.

Keywords: Web. JavaScript. Front-End. SVG. HTML. CSS. Planogram. Software Engineering. Mobile. Ext JS.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Program Interface
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSS	Cascading Style Sheets
DOM	Document Object Model
ENSIMAG	École nationale supérieure d'informatique et de mathématiques appliquées
HTML	HyperText Markup Language
INP	Institut polytechnique (de Grenoble)
IT	Information Technology
JPEG	Joint Photographic Experts Group
JS	JavaScript
KSW	Klee Store Web
PFE	Projet de Fin d'Études (End of studies project)
PNG	Portable Network Graphics
R&D	Research & Development
RAM	Random-access memory
SVG	Scalable Vector Graphics
UFRGS	Universidade Federal do Rio Grande do Sul
UML	Unified Modeling Language
XML	Extensible Markup Language

SUMMARY

1 INTRODUCTION

This undergraduate final work is an important step both in Brazil and France. In Brazil, specifically, at the Computer Science course at UFRGS, the focus is mainly on academic contributions. However, the objective of the PFE (Projet de Fin d'Etudes, or End of Studies Project) is to be the first important professional experience of a student from the Ensimag school of INP. In both works, the task of the student consists of executing different tasks that makes part of a whole; that is to say, they must be well executed individually and also have a sense of coherence towards a main goal, a big project.

In this project I was faced with a new web version of an old desktop application in the sector of merchandising. It means we were concerned with recreating the original features with more modern approaches and new technologies, while the biggest concern was the User Experience.

Along with other minor tasks I worked on contributing to this web application, my main contribution was developing a new lightweight planogram viewer based on HTML5, CSS3 and SVG. The new viewer should also have a modular and flexible architecture, so new features could be easily plugged in in the future. Moreover, this should be done in a way that maintained compatibility with some modules.

This project was a great way to put in practice many of the good programming practices and techniques I've learned during the Computer Science course, both in UFRGS and ENSIMAG. Even more, it was an opportunity to learn a lot about Web Development, from new programming languages to tools and frameworks. I've worked both in the Front-End and the Back-End of the application, and to do both these roles in a real product was a challenging task.

Finally, the development of this project inside a French IT company was an experience that presented personal challenges beyond the technical ones. These had to be overcome with creativity and maturity, and provided me with great lessons for growing as a professional.

1.1 Structure of the text

This document starts with an introduction to the context of the work, such as its motivations, an overview of the overlying web system and the company. The introductory part ends with the section 2, which presents an overview of technologies and techniques involved in this project, such as web technologies and programming practices. Then, in section 3, we start discussing the main topic in depth, which is the development of the planogram viewer module. We present an informal description of its requirements, the architecture and some technical details of the implementation. Before concluding, we quickly discuss in section 4 some interesting technical details of my other contributions done during this internship to the application. Finally, in section 5 we conclude discussing both technical and personal lessons learned in the process of doing this work, as well as the final contributions to the application and some possibilities of further work.

1.2 Context of the work

In this section I'll introduce the company where this project was developed and in particular the R&D team with which I worked.

1.2.1 The company

KLEE Commerce is a subdivision of the KLEE Group, composed of 47 employees engaged in the design, maintenance and development of software applications aiming sales forces, CRM (Customer Relationship Management) and merchandising.

We could say that the organization of the workforce is made along two axes: one composed of teams of Project Management, R&D, Commercial, Client Services, Hotline and Support. On the other axis, employees are divided by the two main software with which they work: Klee Sales and Klee Store, seldom embracing both.

Both applications work based in a common Oracle database and share a similar User Interface, but have different purposes. Klee Sales "provides a global, centralized view of the Sales Force's commercial activity" [3], including the management of things like sales, stocks, payments, contacts, client visits, etc. A strong point of Klee Sales is its Offline Mode, which is important for employees working *in situ* with their clients where there isn't an available connection to the Klee Commerce Server. Therefore, users are able to use the Mobile version of the application with their notebooks or tablets and later have all changes "automagically" synchronized to the Server.

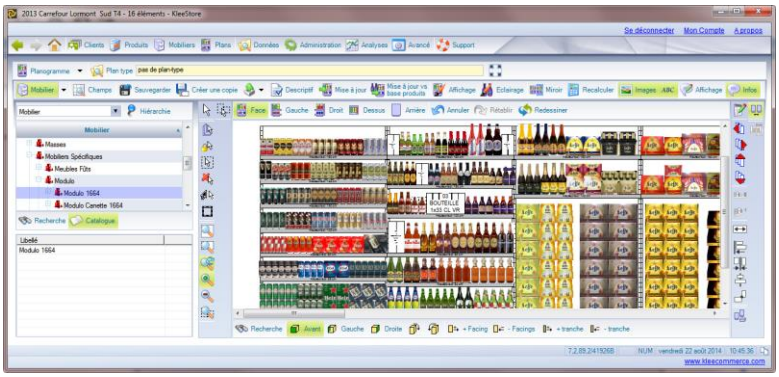

Figure 1 - Edition of a planogram in Klee Store.

In the other side, Klee Store has a focus on *merchandising*, which is "the activity of promoting the sale of goods, especially by their presentation in retail outlets" [8]. The functionalities of the software then involve the visualization and edition of stores' maps and *planograms* (see Figure 1), as well as visualizations of sales analyses. A strong point is the intersection of these two parts, relating sales statistics to the organization of products in stores and their shelves. Since 2013, Klee Store has had a mobile version for iOS (iPad), which was developed using technologies exclusive for this platform.

From 2012, the R&D team has started the development of Klee Store Web (Figure 2), an online version of Klee Store that has been continuously developed, with functionalities from the original Desktop version being slowly recreated with Web technologies. A first release of this application was done this year, and it was shipped with basic functionalities of search, listing and visualization of planograms directly from the browser.

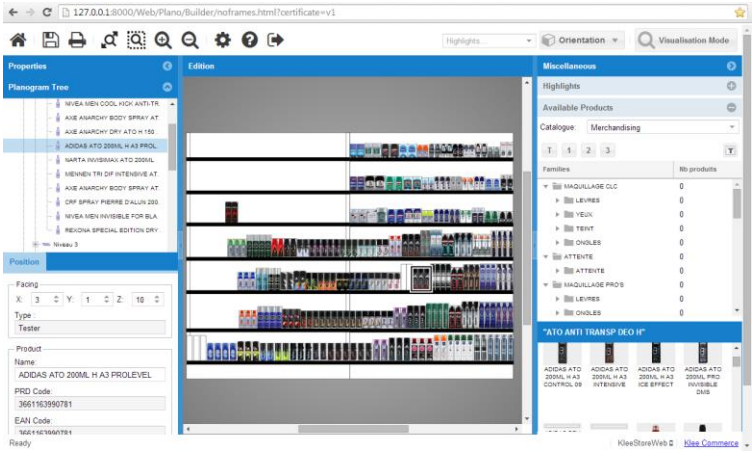

Figure 2 – Edition of a planogram in Klee Store Web.

1.2.2 The R&D working environment

The project was developed inside the R&D group of Klee Commerce. This group is divided into in 3 teams (though some developers might find themselves in more than one), one for each application. I worked with Klee Store Web, the new web version of Klee Store, so I interacted directly with only a very small portion of the team, especially with one colleague and my supervisor Claude Delaye himself, who are both developers on this project, and a few past interns.

We didn't have any special team collaboration tools besides the versioning system, and it was not a major problem given the very small team size and the fact we worked in the very same room. Some further discussion about this is done in the conclusion.

Not many programming tools were needed, as fancy IDEs are not necessary for web development. In the day-to-day work, most of what was used was a text editor, along with a browser and its debugger. Our preferred browser was Chrome and its Chrome Developer Tools, a very powerful yet easy-to-use debugger for web. This will be discussed further in section 2.1.6.

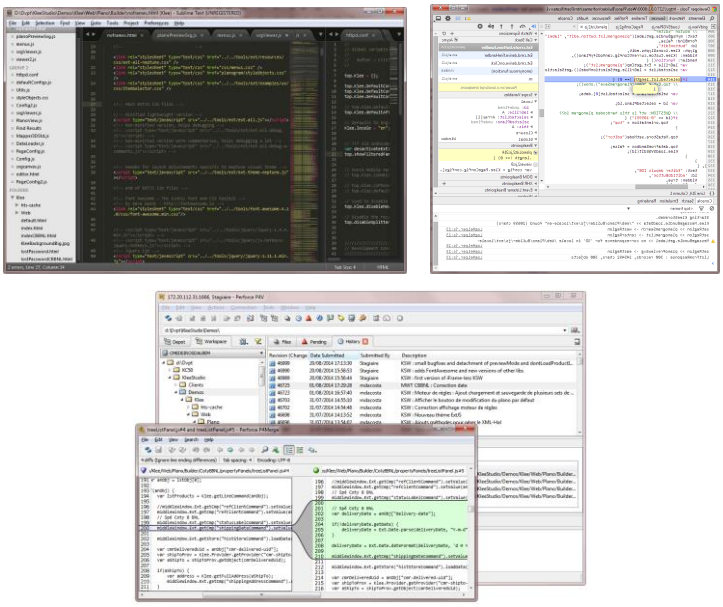

Figure 3 – Screenshots of the 3 main tools needed to develop this project.

2 FOUNDATIONS AND RELATED WORKS

The project was developed entirely in the context of the Web. Even when working in the Mobile version of the application we were dealing with it as it was going to be run on a browser, thanks to the Cordova/PhoneGap framework. In this chapter, some basic concepts of Web Development will be introduced as well as related topics whose knowledge will help the reader to better understand the rest of the document.

2.1.1 Front-end / Back-end

A differentiation that will be very useful for us is between the *front-end* and the *back-end* of a system [18]. Essentially, they correspond respectively to the presentation layer and the data access layers. This kind of separation of concerns into different layers is a recurring concept in Software Engineering and greatly contributes to the understanding of the system, its development and its future maintenance.

While the back-end of a Web application might have some similarities with Desktop applications like operations on databases, messages and protocols over networks, etc., the front-end in the Web is a world apart. Many technologies were created with this purpose, and the patterns that rose from this context today influences others as well, such as some desktop applications using CSS stylesheets to describe its appearance. Since most of the work of this project was done in the front-end, we'll focus on the technologies of this scope.

Although the back/front-end separation has always existed in software development, a bigger concern towards the front-end part of the system is a growing trend in application development for all platforms. It's closely related to the evolution of User Experience [26], which is a more global concept related to everything about the ergonomy of an application and brings together professionals from the most diverse areas, blurring their boundaries. This trend is highly relevant to the evolution of the study of Human-Computer Interaction. This field has been growing in importance the more that computers and mobile devices become accessible to the general public, and a good ergonomy is more and more important in the IT business.

2.1.2 Web front-end basics: HTML, CSS and the DOM

HTML is the heart of the Web. The acronym, which stands for HyperText Markup Language, is the basic language used to write web pages. Its structure resembles that of an XML, with opening and closing tags (such as <*div*> ... </*div*>) in a hierarchical organization.

HTML is not considered a programming language, but rather a mark-up language, such as LaTeX or XML itself. A HTML file is where the interpretation work of a browser starts, parsing the inherent structure in the text file to find out what the page will look like, how it will behave and which other dependencies have to be loaded. A very basic philosophy of separation of concerns states that the HTML file of a page will define its basic layout structure and content, delegating the behavior to the loaded scripts, and the appearance to the CSS stylesheet, both which will be covered soon.

The parsing of the HTML file will generate the DOM (Document Object Model) structure, which organizes document nodes in a tree-like data structure [24]. This is the structure in memory of all the objects described in the HTML file, plus others that might be added with the scripts. Scripts may freely interact with the DOM, by a public API, creating and removing elements or editing their properties and behavior.

2.1.3 JavaScript

JavaScript is, according to Wikipedia's list of "Programming languages used in most popular websites" [11], the most common programming language used in the Web. It is, in most cases, the page responsible for the look and behavior of a web page, but may also be found in the back-end, in web servers, etc.

It's an interpreted dynamic language that allows us to interact with the browser, the DOM, make asynchronous computations, and much more. Even if its birthplace was in the browsers, nowadays it's used in many other places like server side code (Node.js), game development, Mobile and even Desktop applications.

Being a non-compiled language, all errors appear as bugs at runtime (which makes the task of updating a library, for example, a little more frustrating than it would usually be in compiled environments - for example, a change in the API will be noticed only when you try to call a method and it doesn't exist anymore.

2.1.4 The framework Ext JS

Ext JS [19] is a framework written purely on JavaScript. It's one of the oldest frameworks still on the market. Its first version was launched back in 2007, but each new version carried big changes in class structures, APIs and even in the framework's licensing.

Originally, Ext JS was an add-on library extension of YUI, the Yahoo's go to JS frameworks, discontinued in 2014.

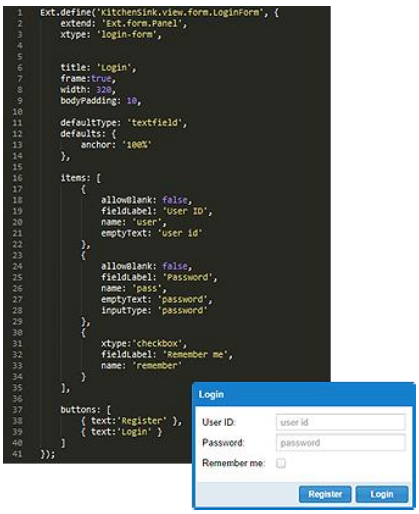

Figure 4 – Example of code for generating a login dialog box using Ext JS.

In this framework everything is described with JavaScript, like in the example of Figure 4**.** This code then is processed when the page is loaded and the correspondent HTML is generated.

The version we worked with for this project was the newly released Ext JS 4.2.0, which went live on March 2013. Despite the fact that the framework had a recent update, its importance to note that the release of 4.0 was back in 2011, and no major improvements were done since then.

It's not in the scope of this paper to do a deep analysis of the framework. We highlight some of the major positive and negative points of its use:

Table 1 – Pros and cons of the ExtJS framework.

Pros	*Cons*
• Handy and powerful off-the-shelf UI components ("widgets") • Good documentation • Vast browsers support • Active community • Integration with Prototype and JQuery	• Up to 4.0, bad to no support of Mobile platforms • Weak official support • Confusing dual license • Auto generated HTML and CSS are hard to debug

2.1.5 SVG

The acronym stands for Scalable Vector Graphics [16]. It consists in a vector image format frequently used in the Web, natively supported by the majority of the modern browsers. The image is described in textual form with a XML structure, and thus can be edited by any text editor, but most of the time is automatically generated by scripts or image edition applications.

Vector images are very different from normal images, most commonly found in PNG or JPEG formats. Vector images are described by forms, or mathematical functions, and thus are independent of size and can be easily distorted, rotated and even stylized.

The most important aspect of SVG, however, is its integration with the Web. A rendered SVG becomes a DOM element, and thus can be fully controlled by the DOM API via JavaScript code, as well as stylized via CSS stylesheets.

Within the format specification there are also interaction functionalities, which were employed in this project.

2.1.6 A web-development coding environment

The environment for coding our Web application was very simple, yet powerful. Notwithstanding when we tested it against other browsers, most of the development was made under Chrome, taking advantage of its famous Dev Tools [25].

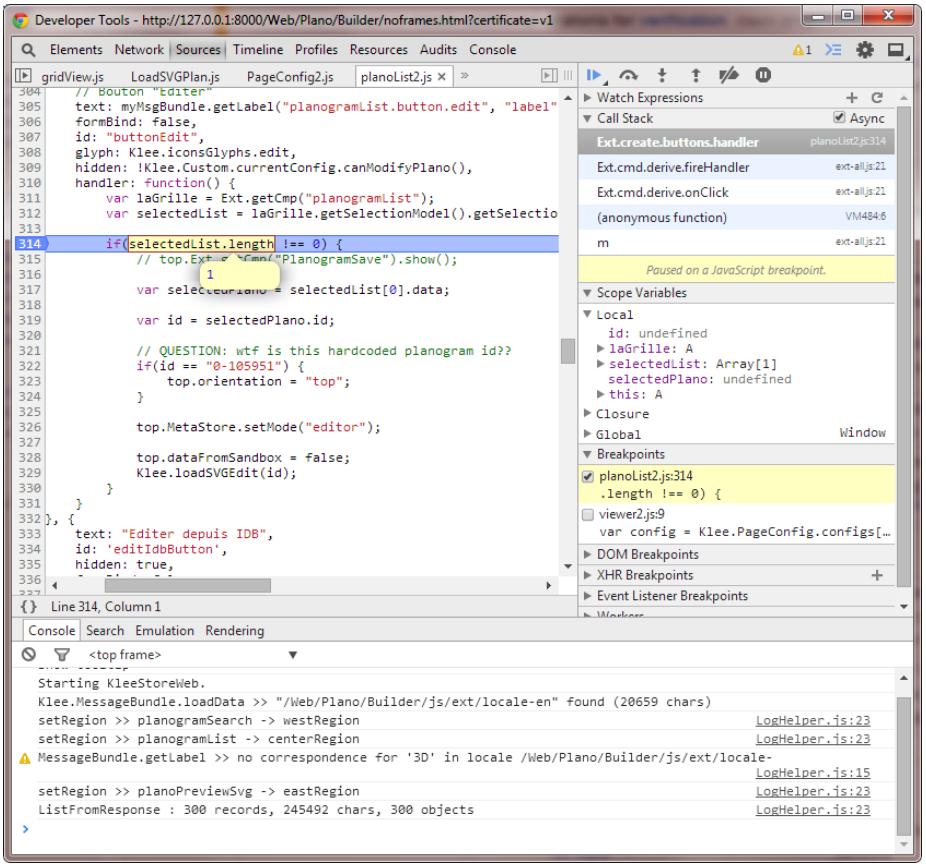

Figure 5 – Screenshot of the Google Dev Tools interface.

In a nutshell, Google Dev Tools is a free, lightweight and powerful tool for debugging everything about a Web application. The several functionalities are divided into sections (which are visible in the screenshot of the Figure 5):

- **Elements**: visualizing and editing, with real-time visual feedback, of the HTML and CSS code of the page;
- **Network**: visualizing of all the network requests sent by your application with information such as size, time taken, displayed in a timeline-like grid.
- **Sources**: debugging of the JavaScript sources with the hability to insert breakpoints, do step-by-step execution, monitor variables values, listen to events, etc.
- **Timeline**: a timeline-like visualization of all the events processed in the page.
- **Profiles**: enables 3 different types of profiling: JavaScript CPU usage, memory distribution and Heap allocations.
- **Resources**: browse through all resources allocated or in use by the page, including databases,
- **Console**: a full interactive JavaScript console.

2.1.7 Code versioning

Code versioning is one of the most useful tools in software development. Its use is strongly recommended as a basic example of good practices in software development.

The idea is that every change to the code repository is registered with a timestamp and generally stored as the difference ("diff") between the versions, thus saving storage space as well as being easy to visualize, modify, apply or reverse modifications.

Code versioning is also a way of documenting the development of a system. Every bundle of changes, called "commits", carries a textual description of it. It's a good practice that this description, written by the developer, will explain whether it's either a bug-fixing commit, a new feature, an optimization of an old function, etc. In the end, the changes in the source files are grouped into logical units that give a great picture of the development timeline. Code versioning is also useful for project managers, since all this logging can be used to measure the individual contributions of the developers.

2.2 Related work

In this section, we'll briefly analyze similar merchandising solutions present in the market.

2.2.1 Online Shelf Planning, by Irian

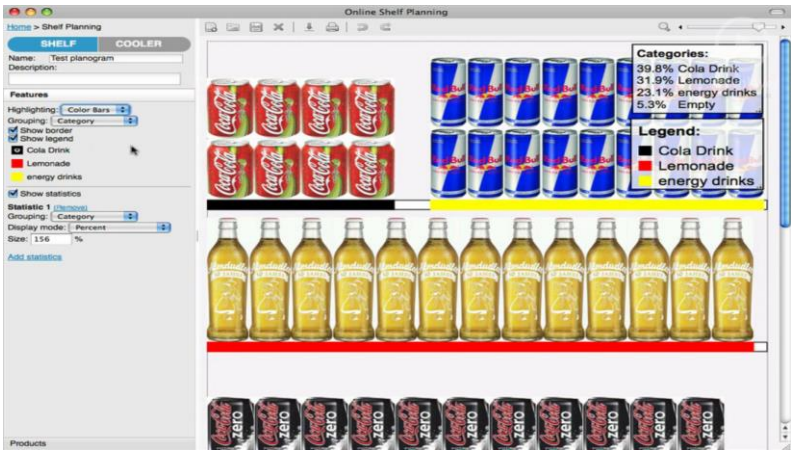

Figure 6 – Screenshot of Online Shelf Planning.

OSP is a small Web application by a Java consulting company.

Website: http://osp.irian.at

Its main features are:

- Fully Web based.
- Simple user interface.
- Full-text search in products database.
- Grouping and highlighting views.

The main disadvantages are:

- Despite the marketing claims, planograms are not visually appealing.
- No 3D view.
- Lack of advanced features.
- Can't customize shelves with things like shelf strips and headers.

2.2.2 Buzz 3D, by Buzz 3D

Figure 7 - Screenshot of a concurrent application, Buzz 3D.

A big, all-purpose 3D virtual environments engine offering several built-in "business solutions" for automotive, real state, education and more, including merchandising.

Website: http://www.buzz3d.com

Its main features are:

- "Near-photorealistic" graphics with fully customizable shelves.
- Ability to import planogram files created with other manufacturers' applications.
- The FootStep™ technology enables virtual user tests in the environment.
- Full store layout and visualization.
- There's no information in the website regarding supporting platforms, nor if there's support for Mobile or Web.

2.2.3 PlanogramBuilder, by zVisuel

Figure 8 - Screenshot of a concurrent application, PlanogramBuilder.

PlanogramBuilder is a fully online solution by a 3D Swiss company.

Website: http://www.planogrambuilder.com

Its main Features are:

- Nice blend of 2D user interface with real-time 3D viewing, including physics.
- Simple interface, however with advanced features.
- Integration with selling statistics and even automatic planogram generation based on profit optimization.
- Complete website with full list of features, video presentation, and even a Trial Demo.
- Full version is rather expensive, but there's a cheaper Light version, as well as reductions for academic and charity uses.

2.2.4 ToolBox Merchandising Service, by ToolBox

Figure 9 - Screenshot of a concurrent application, by ToolBox.

The planogram editor by ToolBox is part of a full suite of applications for retailers and manufacturers for integrating merchandising reports and applications. It seamlessly integrates planogramming, space optimizations, analytics and reporting, all in one solution.

Website: http://www.toolboxsolutions.com/manufacturers-solutions/planogram-solutions/

Its main features are:

- Integration with other information services that will bring the most of merchandising insights into the planogramming process.
- Great graphics and user interfaces.
- Available in all platforms and devices.

2.2.5 My Store, by Dassault Systemes

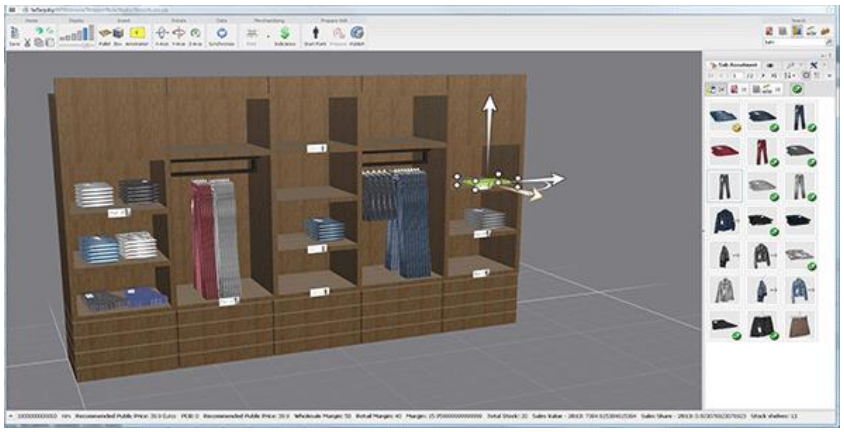

Figure 10 - Screenshot of a concurrent application, by Dassault.

Dassault is the world leader in 3D systems applied to a vast number of solutions to different markets, merchandising being one of them. Due to their size and long experience with 3D and information systems as a whole they are a very strong rival, if not the strongest. However, it's not clear if they attend big retails like supermarkets, since all they public marketing material focus on small shops.

Website: http://www.3ds.com/industries/consumer-goods-retail/my-store/

Its main features are:

- Full 3D representation of the store with high-level graphics.
- Interactive walkthroughs in a virtual store to better communicate the desired shopping experience.
- Advanced analytics features

3 SVG PLANO VIEWER

In this chapter, the main characteristics of SVG Plano Viewer are presented.

3.1 Context and requirements

When the project started, there was only one module for visualizing and editing planograms in Klee Store Web. This module presented many problems:

- It was heavy on memory, consuming from hundreds of MBs up to several GBs of RAM;
- It was slow to load – partly because of the previous problem;
- It was being run on an independent Frame component, which goes against current standards but made proper memory management a harder task;
- The code had bad maintainability, making it hard to improve or add new features.

Most of these problems were due to the module having been built over a third-party tool called *SVG-edit*, an open-source "fast, web-based, JavaScript-driven SVG drawing editor" [17]. We identified several problems with this project decision:

- The original editor was actually a full SVG drawing tool, which means it had many more features than we would normally need for a planogram editor, at the same time lacking many others. Bottom-line is that the intersection of the features already present in *SVG-edit* and those we wanted was small;
- The original editor had a decent architecture, which was well exploited by developing extensions, which could customize how certain elements would be handled by interactions. However, it was not as flexible as we would need it to be in order to turn it into a planogram editor. Thus, interventions in the code were done in a very "unstructured" manner. Moreover, these interventions were not even well marked by the developers, leaving it impracticable to reverse all customizations done over the original code.

These issues not only complicated the development of new features for the editor but also prevented us from being able to update it to its newer versions, which would bring performance improvements and better compatibility with touch devices. These issues could be mitigated if the modifications done by Klee were done in the form of branches of the *SVG-edit* repository.

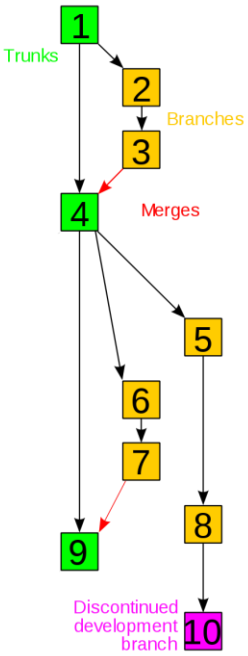

Figure 11 - Illustration of typical operations and structures in versioning systems.

Adapted from: https://en.wikipedia.org/wiki/Revision_control

A branch is a feature supported by many versioning systems, including the one used by *SVG-edit*, SVN. Its purpose is to help developers create multiple independent development paths in a project. It enables people to work over the same base repository but with teams working on their own features, creating *commits* (uploads of code to the server) of their work that won't interfere with the ones of others. If a feature developed in a branch is completed and is ready to be integrated in the final software, this independent branch can be "merged" into the trunk, effectively making both only one. In our case, however, we could use other synchronization techniques like *rebasing*, which brings the modifications committed to the trunk to our branch. With this operation being executed with some frequency, we could have minimized the accumulation of modifications after a long period of time and could take advantage of all latest developments on *SVG-edit*.

With all these issues in mind it was decided that the project needed a new planogram visualizer that would be faster and lighter to run in mobile devices. In addition, we needed something done over a flexible architecture so it would be easier to add new features in the future.

3.2 Architecture

At first, we were aiming to have a lightweight viewer to be used to quickly preview planograms, while edition features were not previewed at the time. The new viewer should also have a modular and flexible architecture, so new features could be easily added to it in the future. Moreover, this should be done in a way that would be compatible with the old *SVG-edit*, since we would continue using it in the Edition mode. This was a hard task, since not only the *SVG-edit* code was complicated, but a lot of Klee Store Web's code for loading planograms, generating and displaying the SVG was highly entangled with *SVG-edit*. Thus, creating a new viewer was not only a question of recreating from scratch the functionalities, visual elements and interactive behavior of the old viewer, but also remodeling how the

DataLoader module of KSW interacted with the viewers' codes, putting an explicit effort in separating functionalities in the proper modules. The bottom line is that a lot of effort had to be done, more than creating new solutions, cleaning up and redesigning what had already been done.

Much attention was given to isolating inside each module the specificities of their own implementations, exposing to the rest of KSW-only common features, minimizing the need of *if* tests, i.e. of specific instructions for each implementation. In this way a common interface between the data loading functions and the viewer classes was defined.

Figure 12 – Class Diagram of the module architecture, first version.

The original architecture, as already mentioned, was not modular and had a single object cluttered with features that sometimes were not even used. The Figure 12 shows a simple UML Class Diagram that shows this original state.

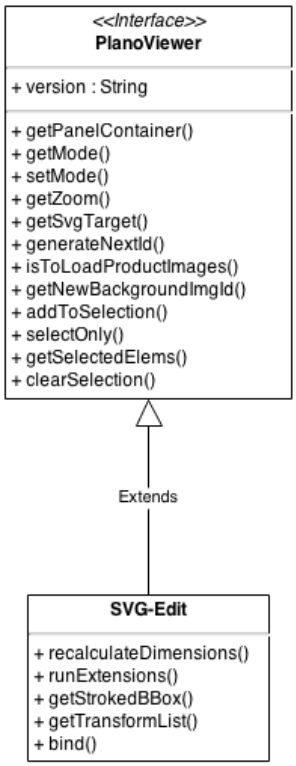

Figure 13 - Class Diagram of the module architecture, second version.

The first step was to define a common interface for planogram viewers and disassociate it from the SVG-Edit editor. Just like all other steps it was not only a conceptual designing

process, but also a mechanical process of refactoring all the application's code to reflect this generalization.

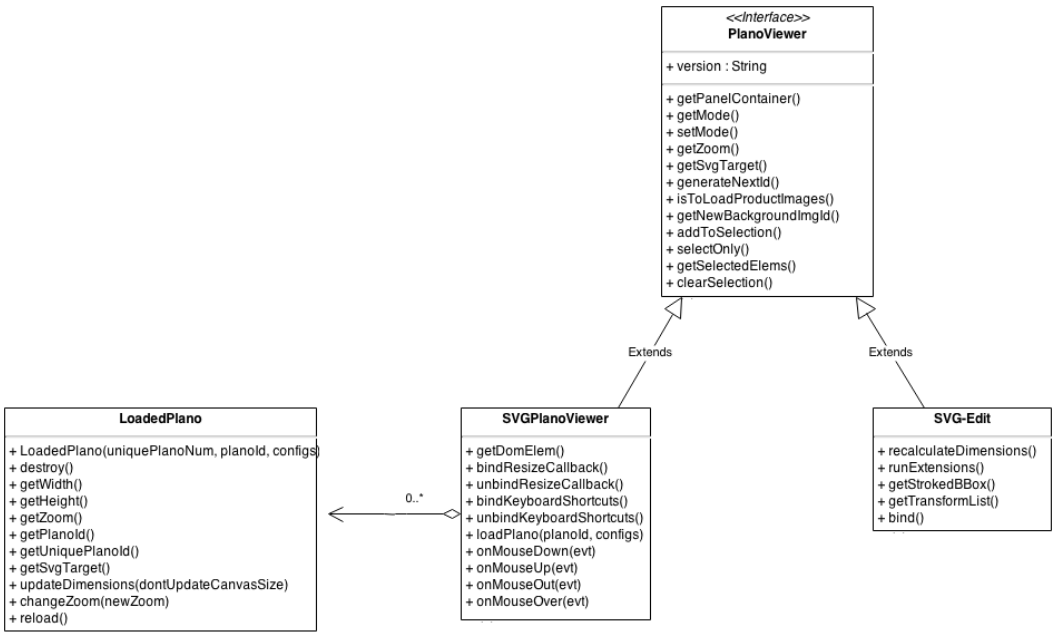

Figure 14 - Class Diagram of the module architecture, third and last version.

The last step was to effectively implement our new viewer, called SVGPlanoViewer, which presented itself as another implementation of the PlanoViewer interface. In the Figure 14, it's possible to see a diagram representation of the final architecture, showing the relationship of the viewers as well as variables and exposed methods.

3.3 Development

In this section some technical details of the implementation will be discussed, including project decisions to idiosyncrasies of the technologies in use.

3.3.1 Classes in JavaScript

JavaScript doesn't have a "class" statement like regular languages that support O.O.P., such as C++, Java, Python, etc. This is due to the fact that JavaScript is a prototype-based language, which is a different kind of "behavior reuse" approach. However for all practical reasons we may consider it as O.O.P. as well, since this technique gives us the same advantages that normal classes would, namely inheritance, encapsulation, polymorphism, etc.

There are two main techniques for creating classes in JavaScript. Both use functions to create objects, the expected *new* directive for instancing an object, and both define attributes as variables inside the function scope. The difference, then, is when defining the class methods.

When using prototypes the class' property *prototype* is "decorated" or "extended" by the new methods. When not using prototypes, the methods are just other functions defined inside

the scope of the class function. Behind the scenes, however, the technical consequences of choosing one to the other are not that simple.

```
var Batman = function () {
    this.publicName = "Batman";
}

Batman.prototype.publicSalut = function() {
    console.log("Hello, I'm " + this.publicName);
}

var hero = new Batman();

// prints "Hello, I'm Batman"
hero.publicSalut();
```

Figure 15 – Example of Class in JS using prototype.

Using prototypes, all objects of a class will share the same reference to the method (because they share the same prototype), thus saving memory when comparing to a prototype-less approach. However, all methods defined in this way are public, and we can't have private methods shared by all of them. Same thing happens with attributes.

Back to SVGPlanoViewer, since it wasn't expected to have many instances of this viewer at the same time, even if it was a big class, we decided in favour of the prototype-less approach. As such, we could take advantage of a more complete model of class, maintaining a stronger encapsulation due to the use of private and public attributes and methods. Figure 16 illustrates with a simple example how we used that with SVGPlanoViewer.

```
// Class Batman
var Batman = function () {
    var privateHello = function() {
        console.log("Actually I'm " + privateName);
    }

    this.publicHello = function() {
        console.log("Hello, I'm " + this.publicName);
    };

    this.veritaserum = function() {
        privateHello();
    }

    // Initialization
    this.publicName = "Batman";
    var privateName = "Clark Kent";
};

var hero = new Batman();

// prints "Hello, I'm Batman"
hero.publicHello();

// throws TypeError (hero.privateHello is not a function)
hero.privateHello();

// prints "Actually I'm Clark Kent"
hero.veritaserum();
```

Figure 16 – Simulating private and public attributes and methods in a Class in JS.

3.3.2 Events bindings

All mouse interactivity with the planogram items is configured inside the very SVG elements, thus delegating to the browser the responsibility of handling the events in the low-level layer [20]. The *setSvgElemAsInteractive()* method (see Figure 17) is called by the back-end modules that create the SVG from the database and configures each item separately. The binding is done with the *setAttribute()* function from the DOM API [30], which binds the mouse events (*onmousedown*, *onmouseup*, …) to the correspondent methods from SVGPlanoViewer.

In the case of the hover events (*onmouseout*, *onmouseover*), which are called when the user passes the mouse over the item, two methods are bound at the same time. One is our method, with our custom behavior, and the other is a default method inherited from the context where SVGPlanoViewer is running. In the present time, this is the method *Klee.tooltip.mouseOver.show()*, which belongs to a tooltip module independent of the viewer. If this concatenation of events weren't performed we'd be overwriting the previous behavior.

```
803
804     // Add the attributes to the DOM element so it will behave as a seletionable
805     //   object.
806     this.setSvgElemAsInteractive = function(svgElement) {
807         if (configs.isInteractive) {
808             // ~ hover
809             svgElement.setAttribute( 'onmouseout', 'top.svgCanvas.onMouseOut(evt);' + defaultOnMouseOutCall);
810             svgElement.setAttribute( 'onmouseover', 'top.svgCanvas.onMouseOver(evt);' + defaultOnMouseOverCall);
811
812             // ~ click
813             if (configs.isSelectable) {
814                 svgElement.setAttribute( 'style', "cursor: pointer");
815
816                 svgElement.setAttribute( 'onmousedown', 'top.svgCanvas.onMouseDown(evt);');
817                 svgElement.setAttribute( 'onmouseup', 'top.svgCanvas.onMouseUp(evt);');
818             }
819         }
820     };
821
```

Figure 17 – The *setSvgElemAsInteractive()* method.

As an example, take the *onMouseUp()* method (see Figure 18). The default behavior will be to simply call the *onObjectClick()* method to handle the simple click action over an object, including all the selection management and the eventual triggering of events on other widgets of the application. However, this bound method could be changed or overwritten to add additional behavior such as the stylization of the selected object, as commented in the original method.

```
1097
1098        this.onMouseUp = function(evt) {
1099            if (evt) {
1100                // Exmple of stylization of the target element
1101                // evt.target.setAttribute('stroke', 'black');
1102
1103                // Default object click handler
1104                this.onObjectClick(evt);
1105            }
1106        };
1107
```

Figure 18 - The *onMouseUP()* method.

As for keyboard interactivity, we import the settings from Keyboard.js (see Figure 19 and Figure 20), a system-wide listing of keyboard shortcuts and default actions. Since this listing is shared by different Viewers, the action is executed upon a common interface between them.

For example, the action of the Figure 19 calls the *getSelectedElems()* function which is part of the interface modeled in Figure 14.

```
703
704     // Binds the handler for keyboard shortcuts
705     this.bindKeyboardShortcuts = function() {
706         if (configs.isSelectable) {
707             var binds = Klee.Keyboard.getBindings();
708             for (var i = 0; i < binds.length; i++) {
709                 var aBind = binds[i];
710                 $(document).bind(aBind.event, aBind.key, aBind.action);
711             }
712         }
713     };
714
```

Figure 19 – The *bindKeyboardShortcuts()* method.

```
107
108     // Selection of the object in the beggining of the shelf
109     var aBind = {};
110     aBind["event"] = "keydown";
111     aBind["key"] = "ctrl+home";
112     aBind["action"] = function(evt) {
113             evt.preventDefault();
114
115             var elem = top.editorWindow.svgCanvas.getSelectedElems()[0];
116             if(elem) {
117                 Klee.Keyboard.selectElement(elem, "home");
118             }
119
120
121             return false;
122         };
123     binds.push(aBind);
124
```

Figure 20 – Excerpt of the Keyboard.js file.

3.3.3 Application features

When initializing the new viewer the constructor will accept several options to customize its behavior and activate/deactivate some of the novel features implemented. These were not present in the original version of the module, and were born in brainstorming with the team and long wanted features that were hard or even impossible to implement before.

isInteractive

Enable or not all interactivity with the rendered SVG.

multiplePlanos

Mode for enabling viewing multiple planograms in the same view. Each new planogram loaded into the viewer will not reset all configurations, but will rather just add the new one. Full interactivity is possible with any of the planograms in the view.

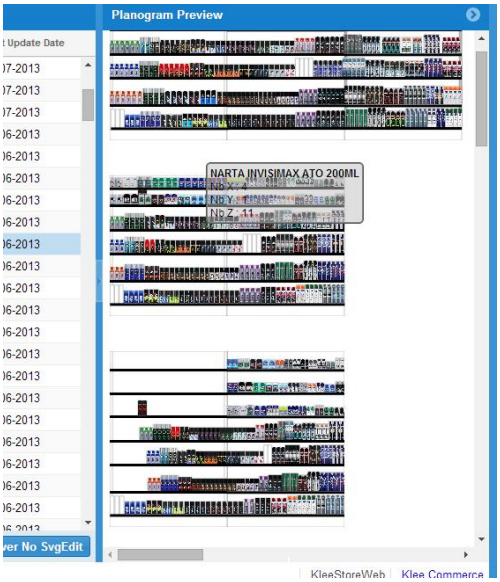

Figure 21 – Multiple planograms view feature.

previewMode

Disables the automatic loading and the two-way interaction with side-panels and other, which is the normal behavior when viewing and editing planograms. It's currently used in the home screen (see Figure 23).

dontLoadProductImages

With this mode, there won't be requests for the individual images of each of the products placed on the shelves. Instead, a big, low-resolution pre-generated image of the entirety of the planogram will be placed under a transparent SVG structure, thus enabling fast loading time but maintaining interactivity. This option was useful when we wanted a small render of the planogram.

preloadedSVG

Another optimization for even faster loading, this will use a pre-generated SVG saved in the database, effectively saving time to have it regenerated each time. This feature was still being tested.

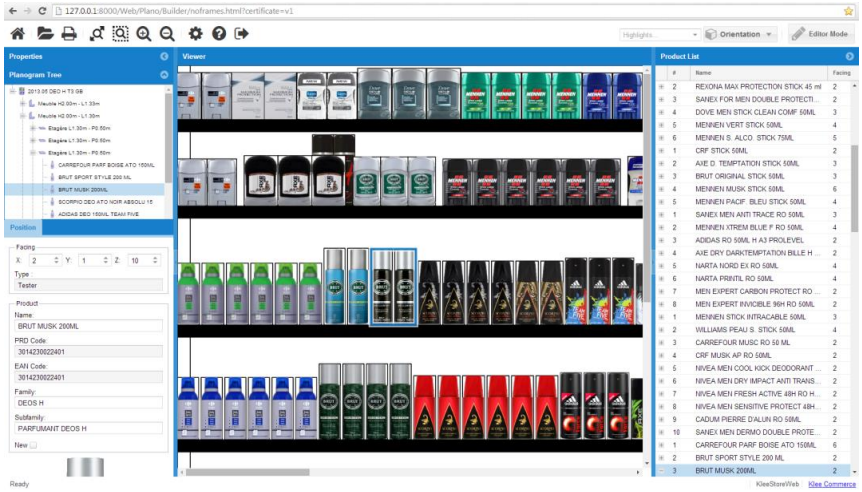

Figure 22 - Result of the SVGPlanoViewer. There are essentially no visual differences from the old viewer.

3.4 Usage Scenarios

Although the complete version of Klee Store is quite a complex software, with several use cases with different flows, Klee Store Web was designed to be simple and easy to use, thus resulting in a small number of screens. In this chapter a basic use case will be described.

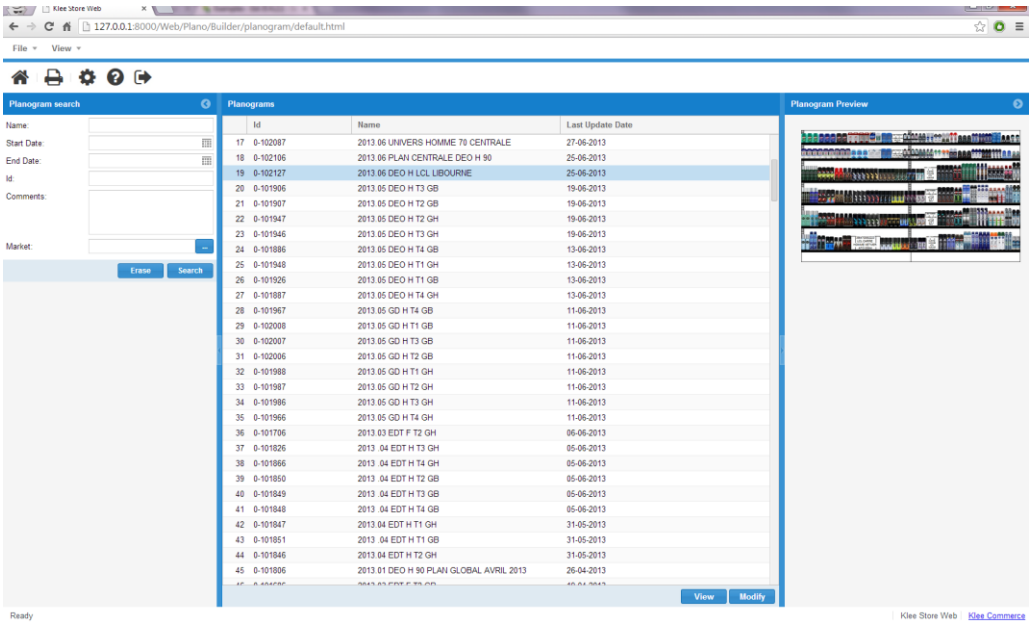

Figure 23 – The home screen of Klee Store Web.

The home screen of the application shows a list of all saved planograms, as well as search fields for filtering these results. Clicking on a planogram in the list will open its preview in the right. In the past, this preview would be a static image file, automatically pregenerated by the server based on the last saved version of the planogram. Nowadays, this preview is an instance of SVGPlanoViewer, thus a fully-interactive SVG representation of the selected planogram with no extra cost in loading compared to the previous approach.

Having a planogram selected, two main actions are clearly available in the two buttons in the bottom: View and Modify.

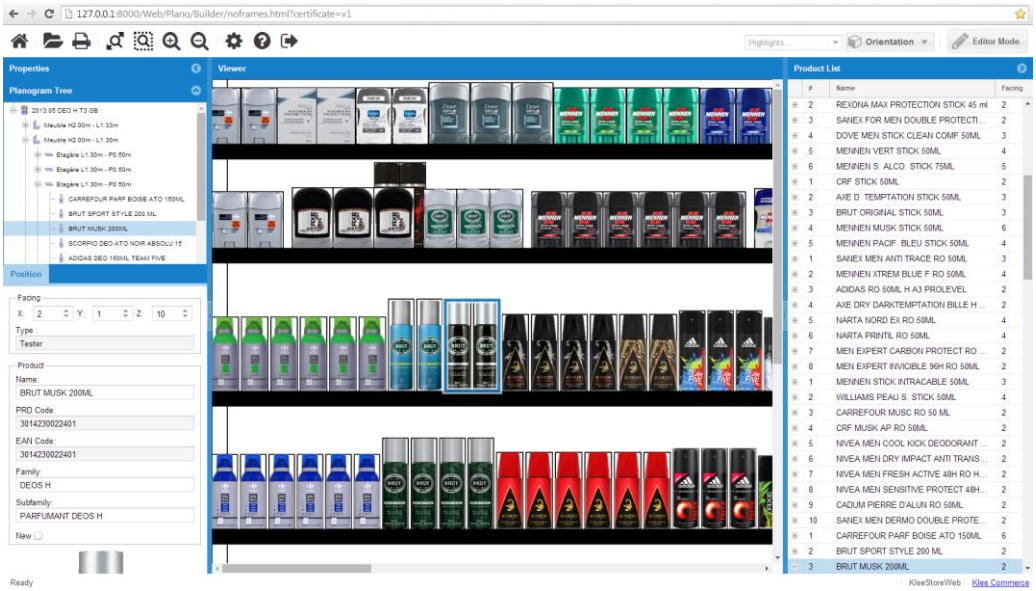

Figure 24 – Planogram viewer screen.

Clicking in the View button the main panels will be updated to reflect the new state. In the center, a bigger and more complete instance of the SVGPlanoViewer will be loaded. In the side panels, ExtJS widgets will be populated with information about the products of the planogram. There's a two-way integration between these widgets and the viewer, which means that interactions done in one will be reflected in all others and vice-versa.

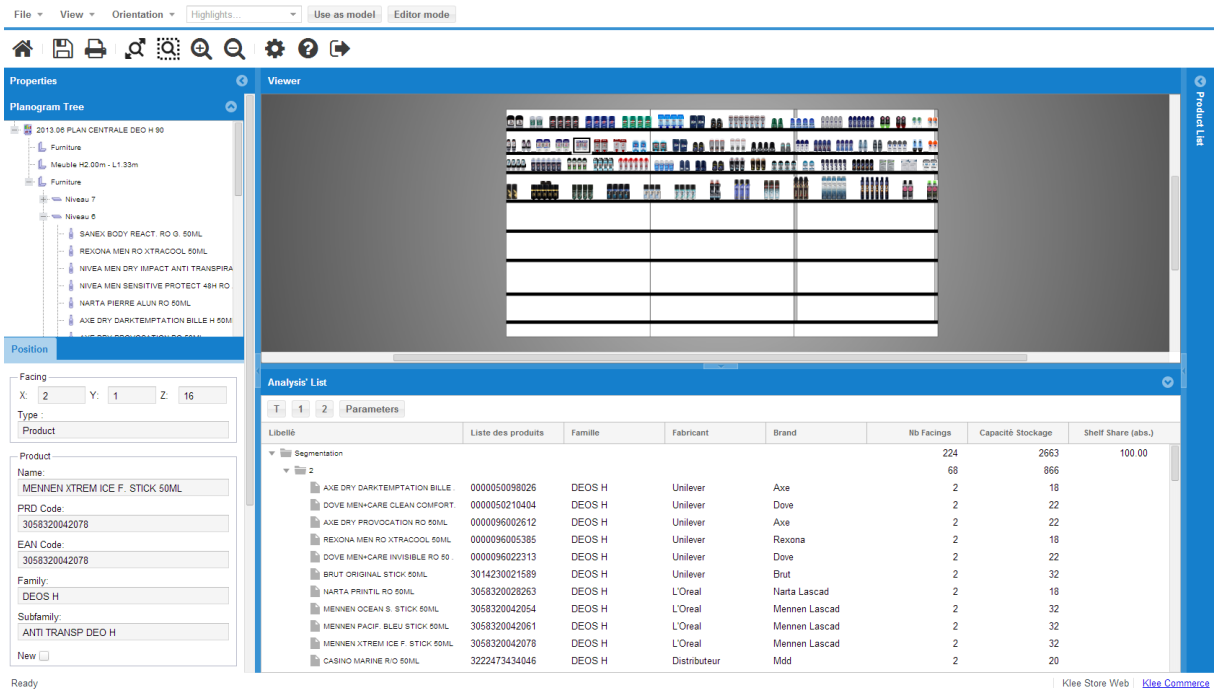

Figure 25 – Another version of the planogram viewer screen, now with a slightly different configuration of panels.

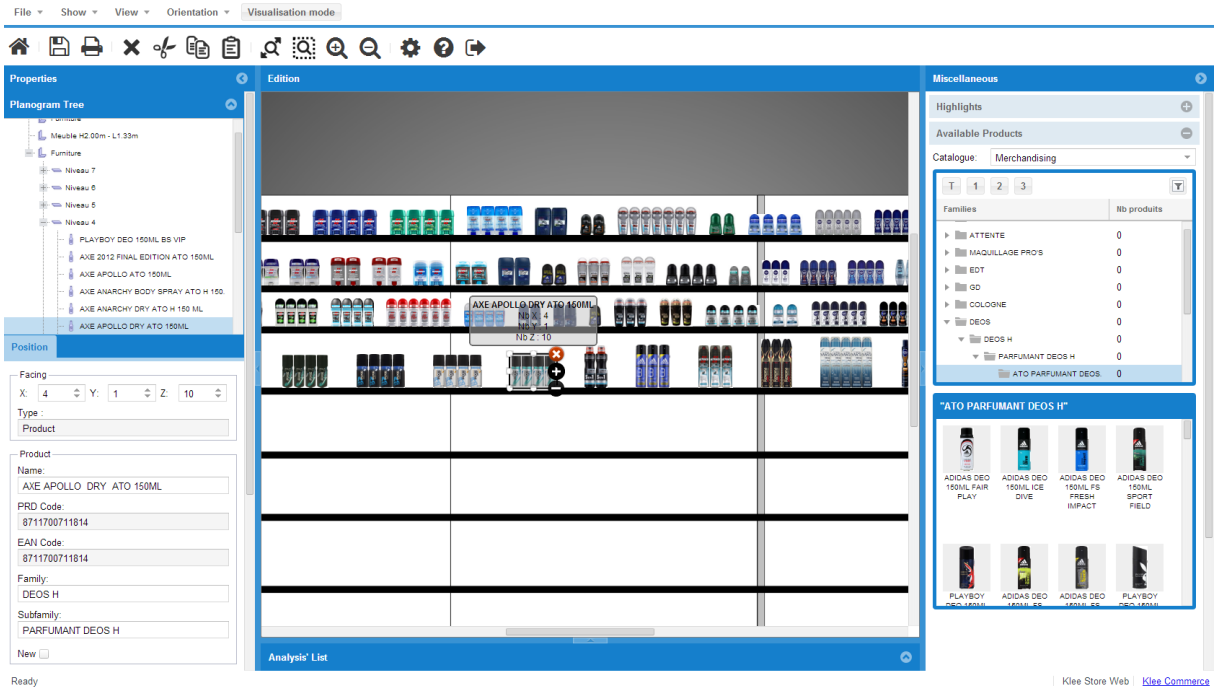

Figure 26 – Planogram edition screen.

Lastly, if chosen to edit a planogram a screen like the one in Figure 26 would be loaded. It doesn't look much different than the visualizing screen, but has some core differences. The left side panel is the same, showing the tree structure of the planogram and the product details panel, but in the right side panel there'll be some advanced highlighting options, as well as the products catalog browser. Moreover, the central panel will have *SVG-edit*, the original editor and viewer of planograms, heavier to load but full of editing features.

4 ADDITIONAL WORKS

During the internship, some other contributions were done to the project of the web application. Despite the fact that the conception and implementation of the planogram viewer module was our main focus, we identified some critical aspects of the system that could be improved with certain ease, demanding just some fresh ideas, basic Software Engineering principles and mildly decent programming skills.

4.1 Towards a portable application

With the unquestionable rise of portable devices, one of the main concerns of the project management was to be able to ship the product as a portable application, ready to be run on the desktop, tablet or mobile phone.

To tackle this problem we started by analyzing the different tools and techniques available for developing a mobile application. Some of them would let us reuse part of the code or the whole code that we already had, while others forced the application to be rewritten in its entirety. Let's take a close look at which one.

4.1.1 Alternative A: Native Android application

Using Android's official SDK (Software Development Kit) we would be able to create the best mobile experience possible within the platform, both for tablets and mobile phones. The use of the SDK enables you to exploit the specificities of the platform at its best, which means a more fluid and overall pleasant experience. This solution however implies the need to develop an application from scratch, since native apps are written in Java, but Klee Store is made in C++ and Klee Store Web in JavaScript.

4.1.2 Alternative B: Optimizing Klee Store Web for mobile browsers

A simple approach for the problem would be just to optimize the ergonomy and the performance of the application so it could be simply run from the browser of the tablet or mobile phone. This was possible thanks to the wonders of web technologies being put in place by Klee Store Web: HTML, CSS and JavaScript are extremely flexible technologies which enable us to run the same application in any device that has a browser to the Internet without the need of special compilations and compatibility layers. In this manner, the original Klee Store, built over C++ and platform-specific libraries, didn't stand a chance!

With this approach, there was still the option to convert the application to use Sencha Touch, yet another JavaScript framework from the same creators of Ext JS (and with a huge intersection of its API ...) devised towards the mobile world. However, this option was discarded since a research done on Sencha's forums by experienced users showed this transition was not at all evident. Even many users would question the market decision of Sencha to launch two different frameworks that seemed so similar but would not have a decent compatibility between each other. It would easily be compared to other libraries like Twitter Bootstrap, which has a perfect cohesion between desktop and mobile interfaces with the same code. It's true, however, that Bootstrap is a lot more UI-oriented, while Ext JS also has a back-end layer. This complaint brought by the community was completely reasonable,

and indeed a few months later Ext JS 5 was launched, featuring a mature support for mobile taken directly from their development in Sencha Touch.

4.1.3 Alternative C: PhoneGap / Cordova

PhoneGap (or, as was known before being bought by Adobe, Cordova) is a framework for developing extremely portable mobile applications. With the same code you can automatically generate apps for the main mobile Operational Systems: iOs, Android, Windows Phone and BlackBerry. This solution would also be very easily done as its applications are nothing more than webpages in HTML, CSS and JavaScript, which was exactly what we had with Klee Store Web.

The final decision was that some quick effort in both alternatives B and C and combining them in one solution could already give us a decent result. Effectively, in less than one month we had a mobile app up and running in a real Android device, counting with the time to learn the framework and adapt the software for it. Even if the user experience that a PhoneGap app offers to the user is not as fluid as a native application, the cost/benefit ratio was much higher in our use case.

4.2 Reenginering the application's pages management

In W3C's article "HTML 5 differences from HTML 4", *frames* are listed as obsolete, with the explanation that they "(...) are not in HTML because using them damages usability and accessibility" [13]. Among the innumerous disadvantages of frames-based architectures, which have contributed to being increasingly considered like that since the early 2000s and consequently removed from the HTML5 standard, we can cite those that mostly affected the project of Klee Store Web:

- Dependencies used by more than one panel had to be loaded by each one of them, including the whole Ext JS framework, and for each time they would be loaded. It means that coming back and forth from a specific screen would each time recharge a big amount of code, impacting both the network and CPU usages;
- The semantic of the "back" action, present in all modern browsers, gets messy when dealing with frames. It happens because each one of them has their own (hidden) URL, which has its own history and so will respond to the back action differently. This makes the pages' navigation not only less intuitive but also harder to be handled;
- Frames were not well supported by Apple's browser Safari.

With that in mind we identified that improving this architecture was crucial for achieving a better website experience not only for the Mobile but even for Desktop, where KSW was already presenting a unsatisfactory performance. For accomplishing this task, substantial knowledge of the system was needed, as it would change some core parts of the front-end code. For this reason, this task was strategically left to be done by the end of the internship.

A little extra challenge of this mission was to maintain the compatibility with the framed navigation, since some modules were found to be very complex to be expurgated of their frames, and we would like to be able to postpone this work for a later time. At the same time

we would already take advantage of the frameless pages. This was the case of the Editor and the 3D View modules.

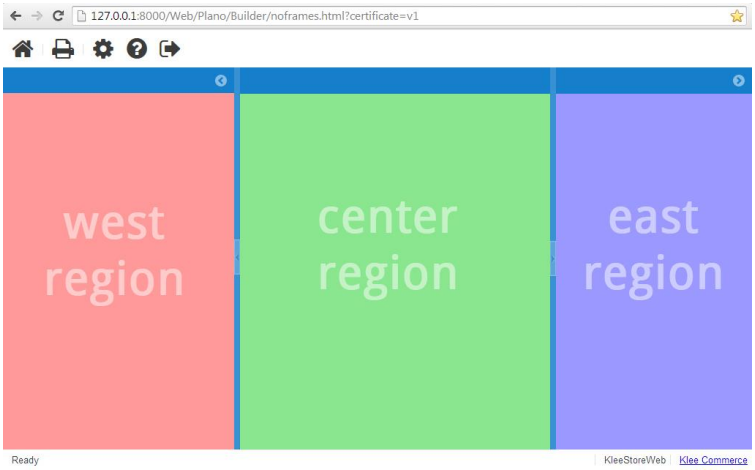

Figure 27 – Illustration of basic layout of the KSW interface.

The changing of pages is controlled by the PageConfig module. When clicking on a button which triggers a screen change it will call the setRegion() function, passing as arguments the region of the screen we want the new page to be loaded (corresponding to one of the 3 main panels) and the identifier of the page. This identifier is used to recover data about this page, like the URL of HTML page, as well as the initialization and destruction functions of the page. These are special functions customized for each screen that encapsulates everything that needs to be done when the screen is loaded and later when the screen is unloaded.

A concern we had when developing this solution was related to the loading of all JS files required for the whole application would be loaded at once. Back with the frames solution, dependencies were solved only when needed, that is, only when we would load the HTML page of a certain panel that its dependencies would be loaded. Now that all those HTML pages became just JS files to be executed, all dependencies would be included already in thehome page. However, a quick analysis of the loading time of all the files using the Firebug debugger of Firefox showed that there was almost no overhead at all, since even when loading all JS files at once they would be quickly loaded in parallel. In fact, the resulting total loading time of the dependencies stayed in the average at the 1 second mark.

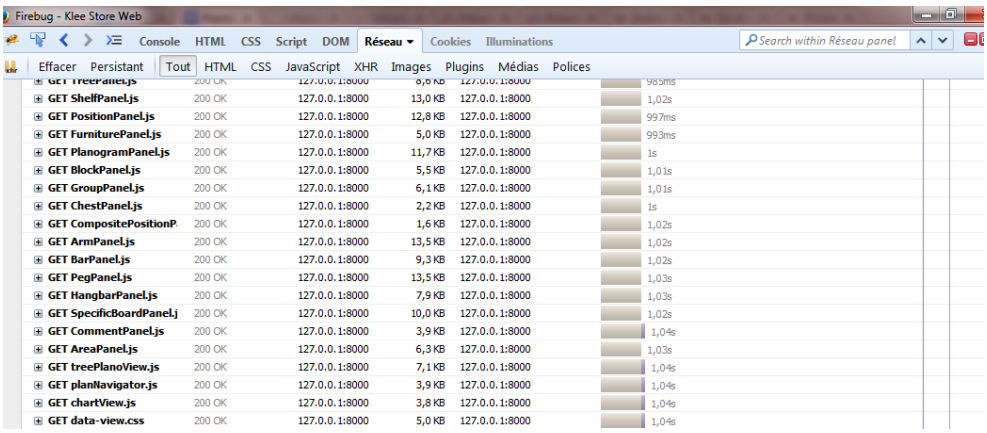

Figure 28 - Timeline in DevTools showing the loading time of the included JavaScript files.

4.3 Offline mode and automatic conversion of C++ code to JavaScript

Both the other main application developed by Klee, i.e. Klee Sales and Klee Store, already had an Offline Mode, which made a lot of sense with the kind of use the clients were making of these tools. While these applications could be used in offices, on desktop computers, there is a big percentage of clients which would use them "on the run", whether visiting their own clients or walking through the corridors of their stores and personally analyzing shelves.

Independently of doing so using notebooks, tablets or mobile phones, it was important that there was a mechanism that would allow changes done offline to be synchronized with the central server running the database. Since Klee Store Web had a planogram edition module, there was also a desire for it to be implemented.

The synchronization of local databases with the central one was performed by KCLib, a library written in C++ that would do complex operations of detection of changes and optimized synchronization of data. Rewriting it was out of the question since it would demand probably a whole internship project on that, and we were more interested in doing interface and usability improvements.

This problem was attacked with what we considered to be the best tool available for this kind of task, which is called Emscripten. It "is a type of compiler termed a source-to-source compiler or transcompiler. As its input, it takes LLVM bytecode, typically created by compiling from C or C++. As output it emits a file in the JavaScript programming language which can run in web browsers." [10]. This is a very popular tool, with almost 7000 "stars" in its GitHub page and which has been successfully employed in impressive projects such as porting the whole Unreal Engine 3 and Ogre3D engines to JavaScript.

Having all dependencies set and Visual Studio properly configured, I set up the compilation to Emscripten mode, and immediately I was being shown a countless number of errors. An effort was done solving all those errors, but eventually we identified the main problem: the main functionalities of our library were dependent on MFC (Microsoft Foundation Class), a proprietary library which not only is exclusive of Windows but also isn't open source, therefore Emscripten cannot convert it. Everybody had thought the project had been freed of MFC at the time it was ported to the iOS, which doesn't support MFC neither. However, we found out that actually a ported version to Mac was developed replacing manually and individually MFC with mocks or reimplementations specific to that platform.

Contacting the person that has worked on the Mac version of KCLib we decided that redoing that job for a platform-generic version of the library was out of the scope of my internship, so this mission as a whole was aborted. Nevertheless, an extensive documentation was left in the Development Journal with the errors that were found during these weeks and the techniques used to solve them, thus facilitating the work of the next person that will attack this problem.

Documentation was also done concerning the different techniques that can be applied to make an Emscripten compiled code be accessible as an API, that is, that public accessible functions and objects of the original code are still exposed to the user in the resulting code. This is not straightforward since the final JavaScript code tends to be very different from the original code, mainly because of the heavy optimization that Emscripten does to minimize the performance gap between C++ and JavaScript (the former is much faster than the latter).

5 CONCLUSION

This conclusion is structured in 3 parts. We'll discuss the lessons learned from this work, first in the technical scope and right after in the personal and professional scope. Finally, we present the final contributions done to the application, as well as some possibilities of further work.

5.1 The technical side

There were a couple of new technologies I've learned during this project and several others that were added to my experience bag. I'll quickly describe my experience with them, giving a critical opinion.

5.1.1 Versioning systems

It was my first time using the versioning system Perforce but, admittedly, it's very close to everything I've already used in the past, namely SVN and Git systems. However, this was one of the best experiences I've already had with versioning systems, since Perforce has a very powerful and intuitive graphical client. Git, the most used versioning technology nowadays is very powerful, but up to these days there isn't yet a good enough graphical client.

5.1.2 Programming environments

My main programming environment was SublimeText, which I consider nowadays the best "almost-IDE" editor available. It really makes me a more efficient programmer with its several clever keyboard shortcuts, which are much easier than a power-user editor like Vim. For even better programming experience, I've included *lint* plugins, which are absurdly useful when programming for Web: they make static analysis of the code, pointing out problems even before testing it in the browser.

5.1.3 Programming languages

Apart from some short occasions when I touched some C++ code and made some SQL queries, most of time I was in direct contact with Web development languages. I already had some experience with HTML and CSS before, which turned out to be handy, but had never used JavaScript before. Although being just another C-like language, with some high-level facilities like dynamic typing, garbage collection and easy support for functional, O.O. and imperative programming, it's still a language that presents some expressive idiosyncrasies of its lambda functions, especially when handling asynchronous programming.

5.1.4 Team collaboration

In the perspective of team collaboration tools I unfortunately didn't have much of an experience during the internship. We had an internal instant messaging system called Cisco Jabber, which was integrated with the email client, with the internal telephone system and with a database of employees. This was very helpful for exchanging files, screenshots and snippets of code between programmers, and also to be able to communicate easily with anyone in the company, even if from another part of KLEE Group.

On the one hand, our R&D team working directly with Klee Store Web was so small that maybe the use of some kinds of team collaboration tools would rather add an overhead to our processes than speeding them up. On the other hand though, something like a simple bug tracker system normally is useful even when working in a project alone, as we tended to continuously discover lots and bugs (and potentially forgetting them).

For that reason I started tracking things up by myself using simple Google Docs documents, which were shared with my colleagues who could at any time view them as well as edit and add comments. On these documents I was tracking bugs I was finding, classifying them by the different sections of the software where I found them. I also was tracking tasks I had to do, sometimes organizing by priority, sometimes by theme. These documentations were of a huge importance to me to have a sense of the quantity of things I was accomplishing, and also so my boss could have at any time an updated view of my advances.

People working with Klee Sales and Klee Store used a bug tracker system called BugZilla, which had a very important role in the dynamic of the work of the programmers in the company. It was one of the most important ways of assigning tasks to people, and sometimes even prioritized over other problems that could be important but didn't have a correspondent BugZilla entry. I've learned, even without directly using this system, that at the same time it was a solid way of tracking advances and things that had to be done, was also a great source of bureaucracy in the software development process. This is not necessarily an inherent drawback of the tool, but maybe just a consequence of how it was integrated in the company's ecosystem.

5.1.5 Testing

Another thing I missed when working in KLEE was testing and continuous integration systems. Things can break at any time. In any case, an automated testing system would give us much more safety and therefore confidence to make modifications, whether adding new features or refactoring and optimizing existing code. This issue was quickly discussed with the team but we didn't find it to be a priority at the moment to implement such kind of thing.

5.1.6 Evaluation

There was no formal evaluation of the new features, neither of the final product as a whole. In addition, since the system wasn't being used yet by real users, we didn't have feedback of this kind. However, the system was constantly tested and evaluated by the R&D team. Although I did miss the use of formal evaluation tools which could prove the positive results in performance, these were rather easy to be tested and confirmed, since there was a big difference between the final and the initial version: a couple of seconds to load a page against, in the original, a couple of dozens of seconds.

5.2 The human side

When choosing where to do my internship one of my main objectives was to stay in France so I could improve my French and dive in to the French culture even more than I had already done when attending classes at ENSIMAG. This objective was very well accomplished. In Klee, differently than other big companies in France, foreigners were very rare, and there weren't many English speakers to add to that. This led me to be intensively practicing the language during the whole day, which has undoubtedly improved after these

half-dozen months. I've also been daily introduced to idiomatic expressions (including many swear words, of course, as it would be expected from people working with computers), the cultural differences between French people from different regions of France, and a lot of new daily vocabulary.

Even after already being in France for half a year the language was a big barrier in the beginning of the internship. In Paris, we have contact with people from all corners of the country, with all sorts of different accents, and so adding to that the background noise of common spaces like the cafeteria rendered the discussions to be intelligible to me. Most importantly, communicating abstract ideas with my boss and colleagues showed to be a challenge as well, but the patience from both parts led us to be able to work together, and in the end I was already able to have a normal and fluid conversation with everyone.

This internship also taught me a lot about responsibilities of working in a real professional environment. From getting to work on time and being polite with everyone, to having a continuous working rhythm during the entire semester. The latter is probably the one that most differs the kind of rhythm we tend to develop in the University when not working in a full-time job. As a student, we tend to study only a couple of weeks before the exams, which are concentrated in two or three clusters during the semester, and projects are easy enough to be developed in a couple of weeks in the end of the deadline. In a full-time internship like this we have a big project to be developed, with many different phases that have to be completed with possibly multiple iterations, and if a regular and consistent rhythm of work is not maintained we won't finish things in time.

Another big difference between what I had already done inside University was the fact that now I was working with a real product. Even if it was still in an initial development stage and still didn't have real users, there was always some kind of pressure to make it the best way possible, whether in terms of the usability to the final user or so it would be easily maintainable for developers in years to come. If the plans of my supervisor goes well, Klee Store Web will keep growing, and may possibly replace completely Klee Store, the flagship of the company.

5.3 Contributions and further work

Thanks to the work developed in this project many future improvements for the software are now possible, or at least facilitated.

- **Smarter architecture equals better code:** new interactions for editing planograms can be easily implemented, since we don't have anymore a code that is mixed up with other functionalities that we don't use neither understand how they work - which was the case with the old *SVG-edit* module;
- **Mobile:** a lighter and faster user interface allows the application to be run in mobile devices. The code was integrated with Phonegap/Cordova, a framework that enables a normal HTML/CSS/JS webpage to be automatically compiled to run in any mobile device, provided that the original application is optimized for that.
- **New frontiers for interaction:** with the MultiPlano mode of SVGPlanoViewer new interfaces can be designed to be more user-friendly, like viewing a list of planograms as thumbnails, or showing as miniatures the different types of highlights possible of an opened planogram;

- **Documentation:** a big part of the code has been refactored and commented during the course of the project, which will make life easier for all developers that work on it in the future. This included a 35 pages development journal containing countless descriptions of bugs I've found and how they were fixed, as well as details about the development of all the features I've worked on.

Several other ideas were devised during the timespan of the project and couldn't be implemented due to the time available, but this is natural. As for limitations of the work done I could cite mainly:

- The planogram viewer module implemented still needs to be much worked and extended to fully meet the needs of the company and match the one present in Klee Store.
- There's a limit with which we can improve how the code is organized without spending too much effort and time with it before it'd be worthier rewriting it from scratch. This is a specially strong argument taking into account the number of tasks with the time I had. Even if a big effort was done in reengineering the system to be more organized and modular, it still wouldn't be possible for instance to reshape the whole system to fully comply with a MVC (Model-View-Controller) Architecture, or similar ones.
- The framework chosen has its own limitations, which were briefly discussed in the section 2.1.4.
- The solution implemented for making the application available on mobile devices is far from an optimal one, as commented in the section 4.1 in the differences between available solutions. If the company decides to heavily invest in the mobile a new solution will have to be found.
- As for documentation, just in-code commentaries and a troubleshooting report as the ones I've written aren't enough for a more mature system. If the development of this application evolves further, a more organized and extensive offline documentation should be developed, serving as a reference for experienced developers as well as supplying guidance for new ones.

6 REFERENCES

[1] Klee Group Official Page - http://www.kleegroup.com
[2] Klee Commerce Official Page - http://www.kleecommerce.com
[3] Klee Sales Official Page - http://www.kleecommerce.com/en/solution/sales-force-automation-software-klee-sales.html
[4] Klee Store Official Page - http://www.kleecommerce.com/en/solution/merchandising-software-klee-store.html
[5] Klee Analysis Official Page - http://www.kleecommerce.com/en/solution/business-intelligence-software-klee-analysis.htm
[6] La Boursidière in Wikipedia - https://fr.wikipedia.org/wiki/La_Boursidi%C3%A8re
[7] Planogram in Wikipedia - https://en.wikipedia.org/wiki/Planogram
[8] Merchandising in Wikipedia - https://en.wikipedia.org/wiki/Merchandising
[9] JavaScript in Wikipedia - https://en.wikipedia.org/wiki/JavaScript
[10] Emscripten in Wikipedia - https://en.wikipedia.org/wiki/Emscripten
[11] "Programming languages used in most popular websites" in Wikipedia - https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites
[12] "merchandising" in Oxford Dictionaries - http://www.oxforddictionaries.com/definition/english/merchandisin
[13] W3C - Differences from HTML4 § Obsolete Elements - http://www.w3.org/TR/html5-diff/#obsolete-elements
[14] "Framing (World Wide Web)" in Wikipedia - https://en.wikipedia.org/wiki/Framing_(World_Wide_Web)
[15] "PhoneGap" in Wikipedia - https://en.wikipedia.org/wiki/PhoneGap
[16] W3C - Scalable Vector Graphcis - http://www.w3.org/Graphics/SVG
[17] SVG-edit official webpage – http://svg-edit.googlecode.com
[18] Font and back ends in Wikipedia - http://en.wikipedia.org/wiki/Front_and_back_ends
[19] Ext JS in Wikipedia- http://en.wikipedia.org/wiki/Ext_JS
[20] W3C – SVG 1.1 – Interactivity - http://www.w3.org/TR/SVG/interact.html
[21] Software versioning in Wikipedia - http://en.wikipedia.org/wiki/Software_versioning
[22] Planogram in Wikipedia - http://en.wikipedia.org/wiki/Planogram
[23] User experience in Wikipedia - http://en.wikipedia.org/wiki/User_experience
[24] Document Object Model in Wikipedia - http://en.wikipedia.org/wiki/Document_Object_Model
[25] Google DevTools Official Page - https://developer.chrome.com/devtools
[26] User Experience Design: The Evolution of a Multi-Disciplinary Approach - http://uxpajournal.org/wp-content/uploads/pdf/JUS_Mayhew_May2008.pdf
[27] Choosing a JavaScript MVC Framework - http://www.funnyant.com/choosing-javascript-mvc-framework
[28] Comparison of JavaScript frameworks in Wikipedia - http://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks
[29] Mozilla – Introduction to Object-Oriented JavaScript - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript
[30] Document Object Model (DOM) Level 3 Core Specification - http://www.w3.org/TR/DOM-Level-3-Core

7 GLOSSARY

Android: a Linux-based open-source operational system designed primarily for tablets and smartphones.

Cordova: a mobile development framework developed by Adobe. "It enables software programmers to build applications for mobile devices using JavaScript, HTML5, and CSS3." [15]

CSS: is a language for defining style sheets for web pages. Stands for Cascading Style Sheets.

Emscripten: a C/C++ to JavaScript compiler.

Ext JS: a JavaScript framework for building interactive web applications.

Frame: “a part of a web page or browser window which displays content independent of its container, with the ability to load content independently.” [14] It is considered to be a bad practice in most cases nowadays, and have been marked as obsolete from HTML 5.

Front-end: the presentation layer of a system, is the interface between the user and the back-end.

HTML: the markup language for creating Web Pages. Stands for HyperText Markup Language.

iFrame: a special type of inline frame. Has some advantages over typical frames, but is also considered to be a bad practice in most cases nowadays.

JavaScript: a dynamic programming language mostly used for the front-end layer of Web Pages.

Klee Store Web: The web version of Klee Store, inside which most of this project was developed.

Klee Store: Klee's software for management of planograms.

Merchandising: "the activity of promoting the sale of goods, especially by their presentation in retail outlets". [8]

Perforce: a proprietary versioning system.

PhoneGap: see Cordova.

Planogram: a visual representations of a store's products.

Sencha Ext JS: see Ext JS.

Sencha Touch: a JavaScript framework for building mobile applications, from the same creators of Ext JS.

SVG: "an XML-based vector image format for two-dimensional graphics with support for interactivity and animation." [16]

SVG-edit: the third-party open-source software over which the old planogram viewer of KSW was built.