SUP'COM

المدرسة العليا للمواصلات بتونس
ÉCOLE SUPÉRIEURE DES COMMUNICATIONS DE TUNIS

Eucational cycle for engineers in Telecommunications

Major :

Technology Innovation Management (MIT)

END-OF-STUDIES INTERNSHIP'S REPORT

Topic :

« Planogram & Sales Optimization Visualization Using Shopper Marketing Behavioral Analytics »

Realized by

Nassim Dridi

Supervisors :

Mrs. Asma Ben Letaifa (Sup'Com)
Mr. Bilel Mabrouk (Cognira Tunisia)

Work proposed and carried out in contribution with :

Cognira Tunisia

COGNIRA
COGNITIVE RETAIL ANALYTICS

Année universitaire : 2021/2022

Abstract

In the current digital age, where products have only 5 to 8 seconds to capture a shopper's attention, an optimized planogram can assist in completing a purchase within those seconds. The hosting company sought the need to develop a planogram-based application in order to improve sales.

Key words : Planogram, Web, Dashboard, ReactJS, Scala

Résumé

À l'ère numérique actuelle, où les produits n'ont que 5 à 8 secondes pour capter l'attention d'un acheteur, un planogramme optimisé peut aider à effectuer un achat dans ces secondes. L'entreprise cherchait à développer une application basée sur un planogramme afin d'améliorer les ventes.

Mots clés : Planogramme, Web, Tableau de bord, ReactJS, Scala

الملخص

في العصر الرقمي الحالي ، حيث تمتلك المنتجات من ٥ إلى ٨ ثوانٍ فقط لالتقاط انتباه المتسوق يساعد مخطط محسّن في إتمام عملية شراء ضمن هذه الثواني. سعت الشركة المستضيفة إلى تطوير تطبيق قائم على مخطط المنتجات من أجل تحسين المبيعات

الكلمات لفااتيح : الويب مخطط المنتجات

Acknowledgements

First of all, I would like to address my deepest gratitude to Mr. Bilel Mabrouk, my supervisor at Cognira Tunis and to the whole Frontend team. Thanks to their continuous support and encouragements, I was able to believe in myself. They were leading experts and a role model to follow. Thank you all for transferring the know-how to everybody all around and making from this journey a memorable experience.

I also extend my warmest thanks to Dr.-Ing. Asma Ben Letaifa for her continuous guidance through the completion of this project. First, it was a great pleasure being your student. Your expertise was of great concern to both my academic and personal growth. Thank you, Dr. Asma, for your infinite patience and support.

This internship was a great opportunity to dive into various interesting topics and gain a lot of knowledge that will potentially help me debut my professional career.

Finally, I would also like to thank my family and friends who helped me a lot in finalizing this project within the limited time frame.

Contents

List of Figures

List of Acronyms

GUI	*Graphical User Interface*
API	*Application Programming Interface*
MVC	*Model, View, Controller*
SPA	*Single-Page Application*
SQL	*Structured Query Language*
CRUD	*Create, Read, Update and Delete*
UI	*User Interface*
COP	*Component Oriented Programming*
OOP	*Object Oriented Programming*
UML	*Unified Modeling Language*
FRM	*Functional Relational Mapping*

Foreword

I did my internship at the Tunisian office of Cognira, for a duration of 4-month. I joined the Frontend team, and my project was to create a planogram-based web application.

Cognira is an American company founded in 2015 in Atlanta, Georgia, USA. Currently Cognira employes more than 100 engineers amongst which are data scientists, business consultants and software engineers. Cognira's goal is to help retailers gain the most return on their data, make better decisions, run businesses more efficiently, and deliver improved customer service analytics by bringing a unique and necessary combination of expertise in form of services provided to their clientele

Thanks to the retail knowledge and experience, Cognira applies a wide range of statistical and quantitative analysis in practical and valuable ways. In addition to that, its technical skills ensure navigating the largest and most complex data environments with the most relevant and up to date tools. Retail forecasts can be challenging and time consuming to maintain, even with the most advanced systems. Cognira works with its clients systems to diagnose and address their issues and keep their forecasts in top shape. Cognira is ensuring a successful implementation through mainly three areas: Software expertise: Cognira's consultants have helped a wide variety of retailers achieve measurable benefits from retail planning, supply chain and other analytic software solutions. Retail experience Cognira has worked with retailers around the globe for many years. It's breadth of retail experience

The project of this internship is to build an application which allows the creation of a representative model of an aisle of a store where the user can directly interact with the products in stock and place them on top of it. The customers' purchase behavior will be tracked and that data will be used to improve the representative model.

General Introduction

Software companies & businesses, set to primarily deliver end-to-end software solutions, has proved very helpful for the retail industry. Retailing is a growing business. As a result, there is the need to conduct various types of activities to run your business properly. So, if help is provided from a retail software development company, it would be easier for retailers to increase their business productivity. A modern retail software would prove very helpful to roar a retail business.

The number of products in stores has been increasing dramatically and space has become very important. The continuous introduction of new categories, brands, segments and expansion of retailer brands are putting a lot of pressure on stores and shelf space which is making the shopping experience more complex. Time pressed shoppers are not willing to spend their valuable time navigating through the store aisles and its different shelves. They are looking for simplification in-store which makes them turn to online shopping to save time.The retail industry is undergoing considerable changes under the influence of consumer behavior shift. In essence, people are becoming increasingly impatient. Many shoppers leave the store if an item they are looking for is not easy to obtain and there is no assistant nearby.

With the increasing need to improve the shopping experience and to improve stock and products management the use of a planogram has become a necessity. Any good retailer understands that appropriate visual merchandising is the key to improved sales. A planogram is one of the most effective merchandising tools for presenting customers products. The use of planograms in merchandising creates consistency between store locations, improve visual appeal, and promote product pairing suggestions. They can also help retailers determine how much inventory they will need for each product.

Our mission is to develop a representative model of an aisle of a store where the user can rearrange products from the stock into the aisle, track the customer purchase behavior using analytical charts and heatmaps and visualize the impact of the current arrangement on sales forecasts based on the customer trends.

Our work will be illustrated into four chapters: The first chapter is an introduction to concepts and theoretical foundations of our project. it will serve as an overview to our project. Next the second chapter will identify our application's requirements and illustrate its design. The third chapter will demonstrate the technologies and tools that we used in our project, and its various realization phases. Finally in the forth chapter, we wrap our report with implementation details in the form of scenarios by including illustrations of our work.

Chapter 1

Preliminary study and state of the art

Introduction

A preliminary study is necessary in order to reach our project objectives. It will help us understand the frameworks that surround the new technologies that will be used to build the application.
Many concepts and theoretical foundations were required for this project as it involves many fields of interest such as web technologies, data manipulation and planograms. In the following, we are going to introduce and present the different and essential foundations for our project.

1.1 Planograms

1.1.1 Description

One of the biggest priorities for retailers is to have their own shop clean and organized in an efficient way. The reason behind that is, to bring customers and to get them to come back for more. Therefore, organization is the key to lean retail operations. In this context, planograms are a must-have asset that can assist in setting up the store in the most effective and aesthetically pleasing way.

A planogram's official definition is a schematic plan or drawing in order to display merchandise, so as to maximize sales and minimize wasted space [7]. It can be represented as a model or as a diagram that indicate the layout of the store, as well as the placement of retail products on shelves. Here is an example of a planogram showing the layout of one set of shelves:

Figure 1.1: Example of a planogram

Making a planogram requires a delicate balance of logical organization, such as grouping items in the same category and using consumer behavior and psychology to expose them to new or highly profitable products, as well as increasing sales through cross-selling techniques and triggering impulsive buying behaviors [6]. Stores, for example, will gather all bread-like products in the same aisle before placing peanut butter, jelly, and other condiments in the same location to encourage buyers to fill up on those things at the same time.
Planograms are extremely useful for big retailers and grocery stores that contain many products from a variety of suppliers and have a lot of space to fill. They help retailers know whether products will fit on the shelves or not. They focus on details such as product packaging dimensions, shelveing layouts and dimensions, and product turnover comes into play.

1.1.2 Planogram Benefits

1.1.2.1 Sales maximisation

The usage of a planogram in a retail store will allow the collection of valuable data not only about how products and displays work, but also about the customers purchase behavior. This data can be used later on as actionable insights for in-store sales by mapping each product to an exact shelf or display location [4].

Product placement impacts purchase behavior. Comparing the historical data gathered during product purchases will identify the highest-converting shelves and displays and will help placing compatible items together to increase sales.
From a cross-merchandising approach, planograms also allow for strategic product placement. Milk and bread, for example, are frequently found towards the back part of supermarkets. Why? Those retailers want to make customers walk past other items, which can help increase impulse purchases.
Using a planogram, it becomes easier to map out these routes. If an item is regularly purchased by customers that come back to the store, placing it somewhere that forces them to pass other products that you want to sell must be taken in consideration. Also placing complementary products nearby will create the possibility of a potential upsell.

1.1.2.2 Space optimization

Retail space can be expensive. The cost varies depending on a variety of factors such as location, size, lease term, etc... Regardless of rental expenses, optimizing the usage of your retail space can help retailers run a lean and cost-effective business. Planograms in this situation will help stores stay organized and give a purpose to every area in it.
Planograms support more effective inventory management. With more organization, it becomes easier for staff to stay on top of stock levels. This improved management will lead to an intelligent retail design, meaning customers can easily and excitedly navigate the store.
Finally, planograms will assist in managing third-party relationships. In the case where a retailer is working with wholesalers, vendors or any type of retail partners, planograms will help business owners establish guidelines as to what and how much space they are responsible for. This special kind of setup and prepartion will lessen the workload and establish ownership and accountability.

1.2 Planogram Composition

A proper planogram is extremely detailed and robust. It is mainly formed of three parts: Aisle, sections or shelves, and products. The logical composition of these three parts is what makes a planogram. Setting the dimensions for every component is a very necessary step. The real dimensions of these components are converted into paper or even screen dimensions in order to fit into the diagram schematic representation.

1.2.1 Aisle

A store is composed of a number of aisles. A planogram can be the representation of one of those aisles. Through an aisle, a retail owner can increase sales conversions, capture more customer insights, enhance their experience with a wider choice and faster service, increase cross sell upsell opportunities and enhance sales assistance for store associates.
The aisle is the first element to be designed on the planogram. An aisle can be looked at as a container for the store shelves. The design of an aisle will have a direct impact on the shelves themselves and as a result, on the products. In general, a planogram contains only one aisle and the representation of the aisle dimensions will affect the representation of the rest of the components.

1.2.2 Shelves

Following the process of the planogram creation, the second set of elements to be represented on the planogram are the shelves. We can also refer to them as sections. The shelves can be viewed as the decomposition of the aisle. It creates the container on which the products will be placed. Products can only be placed within the shelf dimensions.
They must be carefully planned and designed since they are the direct representation of the real thing while also they directly impact the planogram's visuals and design. Attracting customers and managing space is the primary role of the shelves.

1.2.3 Products

In a planogram, products are represented as images of the real thing. Size, position container dimensions and image are all taken in consideration when the product is being designed. A lot of variables should be considered when placing a product in a specific position; Where is the best place for this product category? If there is a shortage of this product, what are the best alternatives for this shelf place? How does the placement of this product impact the rest of the sales?
Effective product placement requires careful studying, better stock management and an appealing display.

1.3 Planogram Software

The creation of a planogram is a process that can be done through multiple options. Hiring a planogrammer is an option. He is a specialist who is fully dedicated to creating and managing a store's planograms based on the customer behavior and sales goals. There

is also the possibility of working with third-party experts to execture planograms instead of hiring a dedicated internal role.
However, in the current digital age, planning a planogram has become an easy process that doesn't require a lot of resources or any kind of training. Whether a user is just starting out or have years of experience in retail and planogramming, a planogram software simplifies the process of its creation.
Many planogram software options can be found on the market:

1.3.1 DotActiv

DotActiv allows the creation of product layouts using planogram automation. It offers free planogram software for first time users, however the free plan is limited to only 40 products. It focuses on maximising shelf space performance.

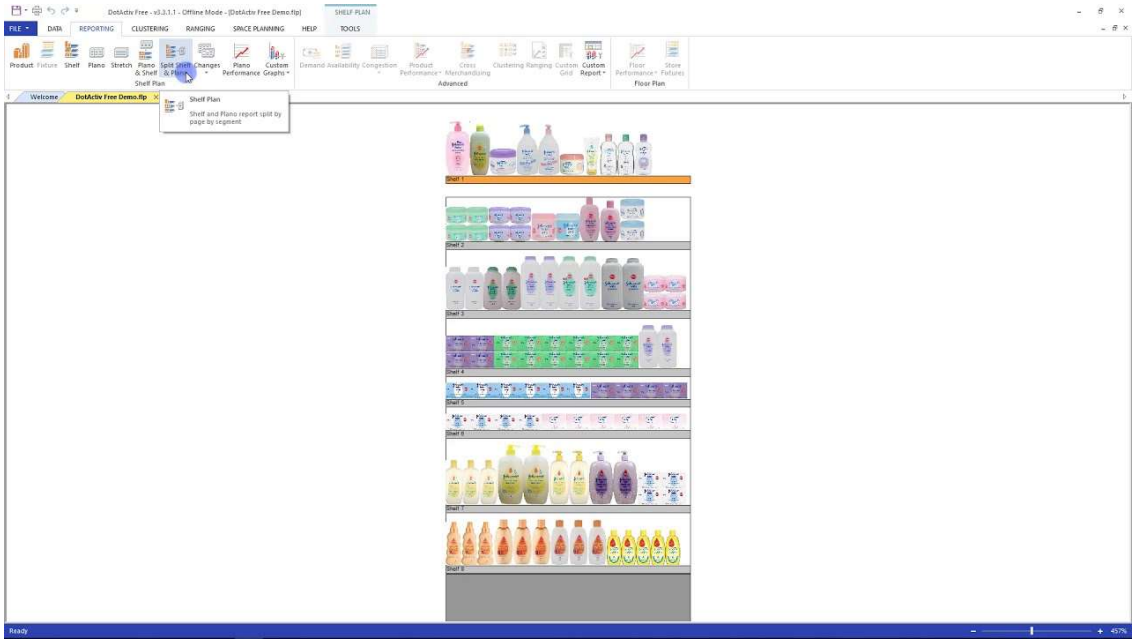

Figure 1.2: DotActiv Planogram software

1.3.2 Shopshape

Shopshape is a software that allows a full MockShop integration, planogram creation and store-related merchandising guidelines. It gives an overview of the store compliance data.

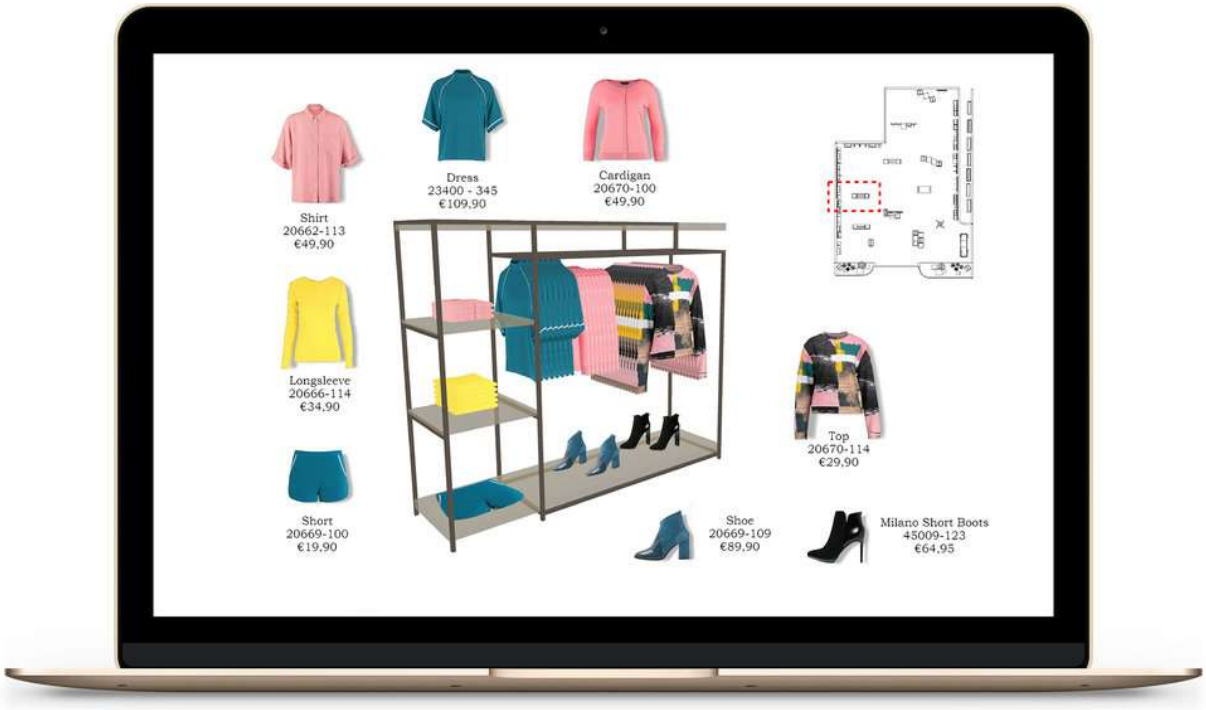

Figure 1.3: Shopshape Planogram software

1.3.3 Smartdraw

Smartdraw allows drawing planograms, it offers multiple templates as a starting point. These templates are easily customizable in order to fit the user's needs. There is also a huge amount of ready-made retail symbols for clothes and other products.

Figure 1.4: Smartdraw Planogram software

1.3.4 Shelf Logic

Shelf Logic offers a software that makes it easier for companies to create robust planograms with ease on a desktop computer using different editions of the software depending on the program the retailer is paying for.

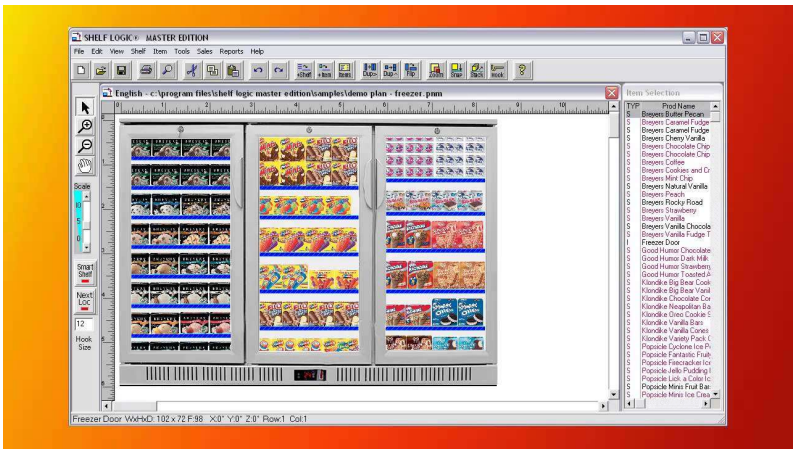

Figure 1.5: Shelf Logic Planogram software

1.4 Constraints and boundaries

Despite the multiple features and all the capabilities presented in the above section, there still exist some limitations and boundaries. Below is a description of what those constraints are and how they could affect the way our application should be designed and implemented

1.4.1 Customization

Being served with a third party software solution corners us with what is being implemented.
Usually, enterprises opt to use more than what they have been offered for their current needs change constantly. In some of the existing solutions, the product customization is sometimes limited to text and templates or to existing models. Products and brands newly introduced to the market will become an issue, especially since the application will need to be consistently updated with new content.

1.4.2 Data Limitations

Most of the features in the existing solutions are limited to the drawing of the planogram and the visualization of products. The data manipulation is limited to the information about the aisles, sections and products. The output of these applications is a planogram drawing without actual tracking of customer purchase behavior. It is the planogrammer's responsibility to manage the products arrangement based on the obtained data.

1.5 Drag & Plan

Each of the mentioned softwares has its shortcomings. In order to respond to the limitations in the existing, our solution, Drag & Plan, will offer numerous features to create and manage planograms in a dynamic and a responsive way. This comparative below sheds light on the different positive and negative points of the existing tools and the added value of our proposed solution.

- Aisles and sections are easily drawn and shaped.
- Products are easily customized and new products can be added.
- The products are placed on the shelves based on a number of logical interactions between all the planogram components such as space, aisles boundaries...

- Sales will directly interact with the planogram by removing the sold product and visualizing the customer's purchase behavior.

Conclusion

In this chapter, we introduced the foundations and the preliminary study of the project including planograms and software solutions. In the following chapter we'll focus on developing an appropriate structure for our application.

Chapter 2

Requirement Analysis and Design of Drag & Plan

Introduction

This chapter is devoted to our project's formal requirement analysis, design and structure presentation. We will analyze the application's functional and non-functional requirements. Additionally we provide an overview of the structural aspect of the project through use case diagrams. Then we will review the application's general design in terms of their structure and functionalities followed by a detailed design using sequence, class and deployment diagrams.

2.1 Requirement Analysis and Specification

This section is devoted to our project's formal presentation and analysis. We will define and analyze our planogram software's functional and non-functional requirements.

2.1.1 Requirements Specification

Taking into account what was said in *Chapter 1 Preliminary study and state of the art* earlier, we must first agree on what we are going to implement before we can introduce our proposed solution and discuss our assumptions and goals.
That applies to both functional and non-functional requirements, in which we specify the behavior and functionality of our system or one of its subsystems, as well as the attributes of our system and how those functional requirements are met and maintained. But first, let's identify Drag & Plan's principal actor.

2.1.1.1 Actor identification

The actors are the direct users of the application, as they have access to the application's core features. Our main actor for this platform is the retailer but we will refer to him as **user**

2.1.1.2 Functional requirements

The following requirements should be implemented in our product for the user to:

- Register and login using a personal account.
- Create, read, update and delete stores.
- Create, read, update and delete planograms.
- Create, read, update and delete products.
- Navigate through the stores.
- Visualize store sales.
- Track users purchase behavior.
- Drag products from the store and drop them into the planogram.
- Stock management.
- Visualize the impact of historical data on the current planogram arrangement.

2.1.1.3 Non-Functional requirements

The application must meet a number of non-functional requirements, including the following:

- **intuitiveness:**
  This is by far, the most important aspect to look for, as the main purpose of building a planogram software application is to simplify the creation of a planogram.
  An application is truly intuitive when the user is the one doing all the intuiting.
- **Ergonomics:**
  The system must respect the uniformity of the design across the whole application and must implement user-friendly interfaces in order to ensure a better user experience to users
  Everything would be provided with a simple, yet a solid user interface.

In the following chapters, we will go through details of the construction of the interfaces in order to offer a pleasing experience for the user.

- **Availability:**
  Under normal circumstances, the platform should be available and not prone to crashing or becoming unavailable.

- **Performance:**
  The optimization of every component of the application will guarantee an acceptable performance for the user. It should take a fraction of time in order to store and update the state of the planogram, products and stores.

- **Efficiency:**
  In terms of memory usage and processing power, the need of as few resources as possible is essential for a web application.

- **Visualization:**
  The application comes equipped with a set of charts of different types. Our application comes with two different techniques:

  - Line-Chart in order to visualize a store's current sales and its revenue.
  - Planogram heatmap in order to display the user's purchase behavior

  Besides displaying what the chart looks like, we need to assure other functionalities and features such as:

  - *Hover effect:* line chart should provide detailed information when we hover over a point. and it should prioritize certain values.
  - *Interaction:* Interaction with the heatmap is an essential part. Our planogram would be able to offer the user the experience to interact with the historical data.

- **Security:**
  The application access is secured by a token-based authentication component.

2.1.2 Requirements Analysis

At this point, we enumerate the various functional and non-functional requirements that the system is supposed to meet.
This section goes over the application's features in detail. We will expand on the analysis of the main possible scenarios and interactions between the user and our application Drag & Plan.

2.1.2.1 Global use case diagram

The main functionalities that our software is offering are: Manage stores, manage planograms and manage products. To perform these functionalities, the user needs at first to be authenticated.
The diagram illustrated in the figure below presents the global use case.

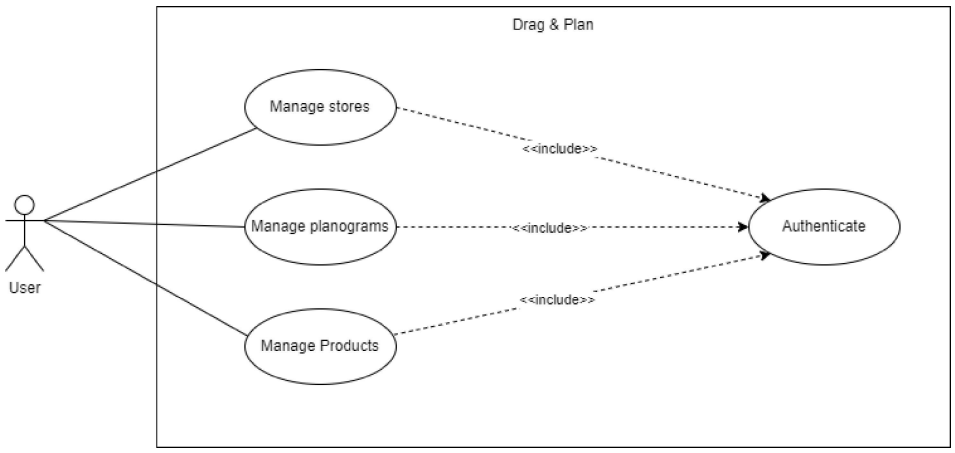

Figure 2.1: Global use case depicting the major functionalities of the system

- **Authentication:** The user should be authenticated and authorized to access the application. In order to do so, he needs to enter a valid username and password.
- **Manage stores:** An authenticated user can add or select a store and visualise sales.
- **Manage planograms:** An authenticated user can create and update aisles. They can also create shelves and place products on top of them.
- **Manage products:** An authenticated user can create new products and new product units. He can also manage the stock and update products.

In next section, we are going to drive deeper into each of these main functionalities apart. We will showcase the main actors, conditions, exceptions and the nominal scenario to achieve certain goals.

2.1.2.2 Manage stores

Below, we have a detailed overview of the *Manage stores* use case.

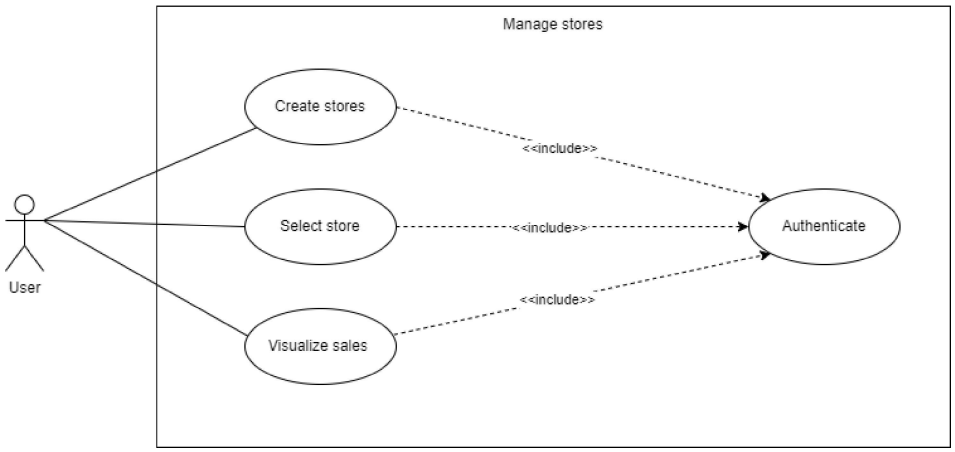

Figure 2.2: Manage stores detailed use case

- **Create stores:** An authenticated user can add a new store to his account.
- **Select store:** An authenticated user has the ability to select a store to manage its products and aisles.
- **Visualize sales:** An authenticated user can visualize the sales of a store in the form of charts.

2.1.2.3 Manage planograms

Below, we have a detailed overview of the *Manage planograms* use case.

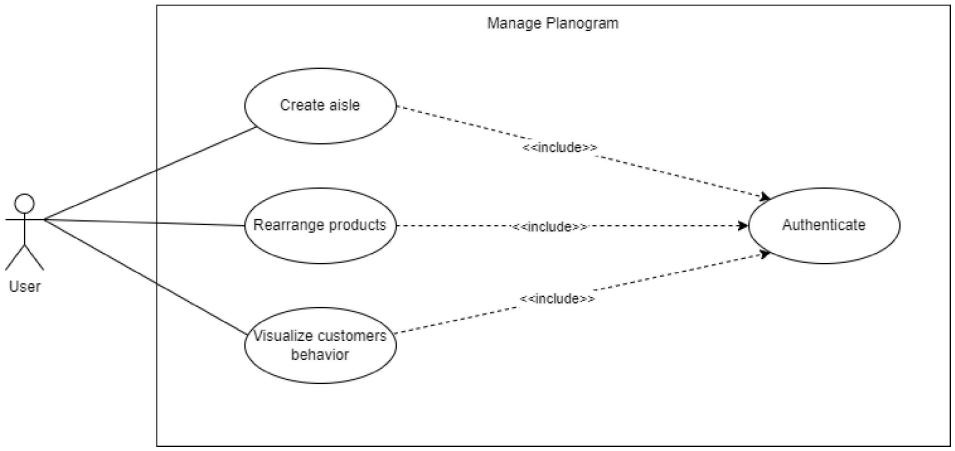

Figure 2.3: Manage planograms detailed use case

- **Create aisle:** An authenticated user can create new aisles in a selected store.

- **Rearrange products:** An authenticated user has the ability to drag products from the stock and rearrange them on the aisle's shelves.

- **Visualize customers behavior:** An authenticated user can visualize his customer's purchase behavior using historical data.

2.1.2.4 Manage products

Below, we have a detailed overview of the *Manage products* use case.

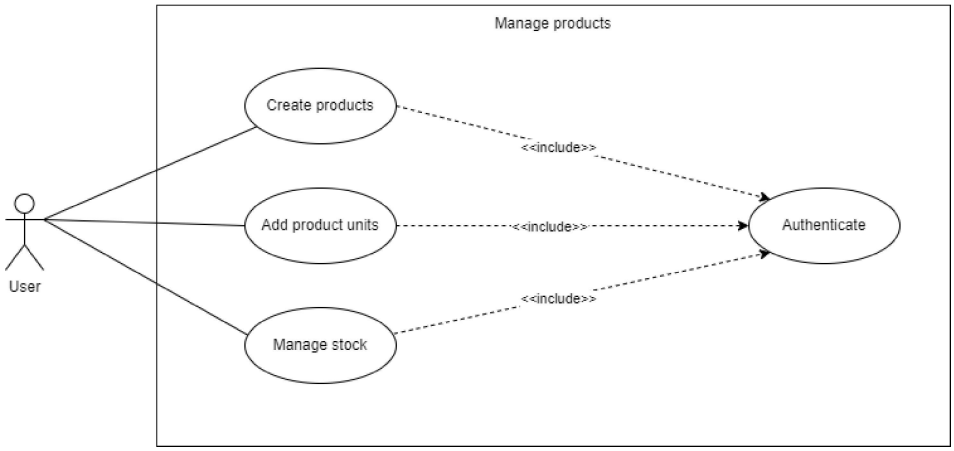

Figure 2.4: Manage products detailed use case

- **Create products:** An authenticated user is allowed to create new product templates.

- **Add product units:** An authenticated user has the ability to add a number of product units to the stock.

- **Manage stock:** An authenticated user is capable of checking the stock, consulting the date of placement of a product unit or its sale date and to sell products.

2.2 Design and Structure

This section will go through the design in terms of their structure and functionalities.

2.2.1 General Design

2.2.1.1 Physical architecture

We will present the technical aspect using the physical architecture. To develop our application, we opted for a three-tier architecture based on multiple levels.
A 3-tier architecture [9] is a software architecture in which the presentation layer and treatment layer are handled by the client or web server, depending on the client type. The database server, on the other hand, manages the data layer. A 3-tier architecture is defined as the separation of these three key pieces into different places in a network to create a distributed system.
Other multi-tier architectures add additional layers to the distributed system, such as an applicative layer or a data layer.
The client is the first tier, the Web server is the second, and the database server is the third. We chose this architecture because of the numerous advantages it brings. In fact, by loading the page once and for all, it speeds up the system. The single page application then retrieves the necessary data and loads it into the page without having to refresh the page, resulting in high performance.
Essentially, the client side corresponds to the personal computer, which we refer to as a heavy client because it is responsible for data processing and presentation. Furthermore, the server side corresponds to an application server that handles requests from a variety of customers. It primarily provides accurate information about the rides and the drivers. The HTTP protocol for middleware ensures a secure connection between the client and the web server.

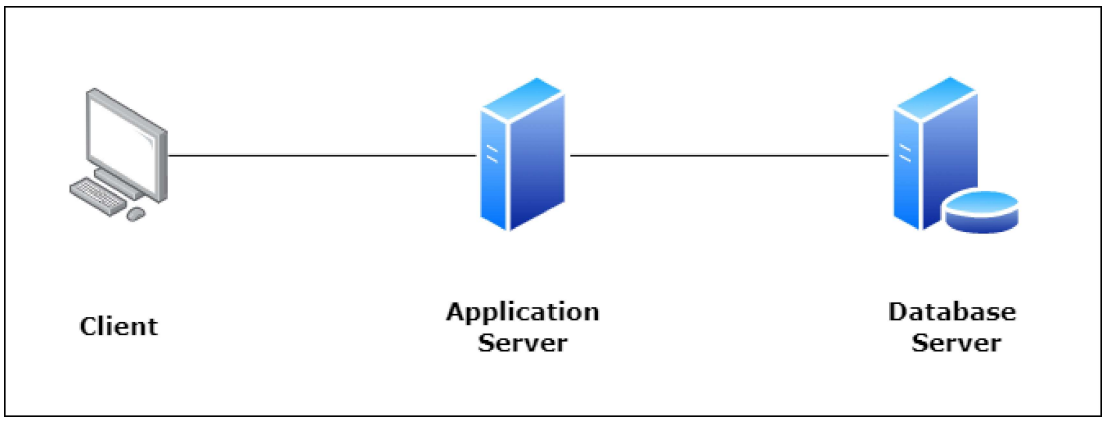

Figure 2.5: Physical Architecture

2.2.1.2 Logical architecture

We are going to present the general logical architecture of our application.We chose a service-oriented architecture with several independent components for our project. We'll begin by introducing a few key concepts before moving on to current web services.

Application architecture
Our application is divided into 3 levels of abstraction [3]:

- **The View** or *Graphical User Interface* (GUI) layer that manages user - machine interactions. It handles layout and display
- **The Model layer** that manages the storage of data, the access to it and the logic behind it.
- **The Controller layer** that manages the controls used in the application's communication with the GUI, as well as the application's processing.

The figure 2.6 below illustrates the application's components and how they communicate:

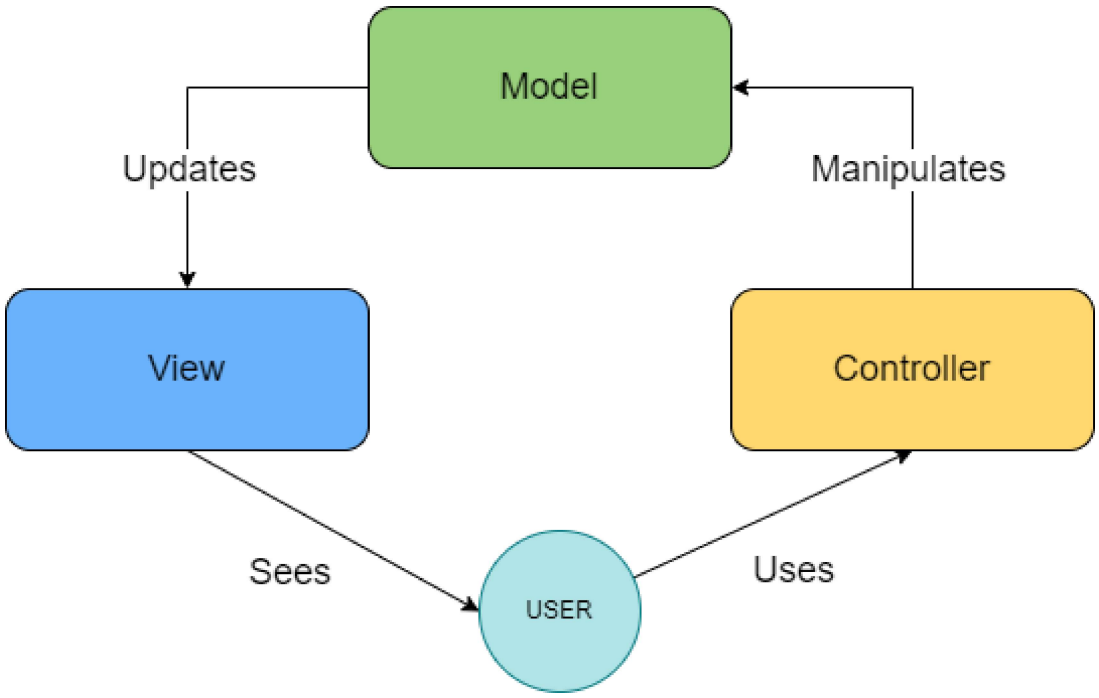

Figure 2.6: Application levels

Component Oriented Architecture

Component Oriented Programming (COP) is the use of reusable core bricks to create a modular approach to a project's design, allowing the software to be more readable and maintained. Because it takes an object-oriented approach in terms of global design, the COP shares several concepts with *Object Oriented Programming* (OOP). Because of the advantages it brings in teamwork, the COP is extremely popular in professional groups that commercialize software. A black box component communicates with the outside through a dedicated interface and allows the management of deployment, persistence, etc.

Definition of a component

A component or service is an independent software module that may be loaded on several platforms and provides a unit of functionality for the same idea. It allows you to export attributes and methods and is configurable. The major interest of these units is that they are configurable basic bricks that may be used to build an application by composing them together. The following figure 2.7 illustrates the schema of a component.

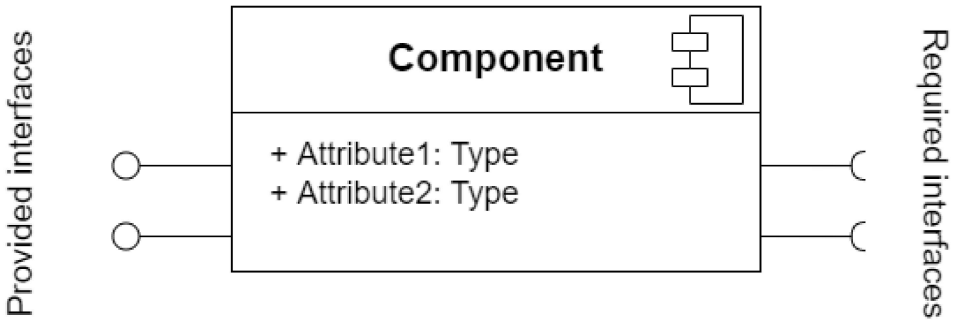

Figure 2.7: Component

Service Oriented Architecture

The distribution of components brings new challenges that must be appropriately managed in order to retain the information system's flexibility and scalability.
On one hand, the system's maintainability and scalability are harmed by the interdependence of distributed components. As can be seen, a distributed design must reduce

interdependence between each of its components in order to maintain its effectiveness, which risks producing cascading dysfunctions that are difficult to detect and pinpoint its source. These parts are heterogeneous, utilizing a variety of products and standards. Maintaining the software's quality of service in the context of distributed architectures, on the other hand, is a challenging, often complex operation, especially when the technical parts of the architecture are heterogeneous and use multiple products and standards. These difficulties necessitate a more flexible architecture in which the components are really independent and autonomous, allowing for faster deployment of new applications. As a result, service-oriented architecture has emerged.

Web services architecture

Web services architecture is a sort of architecture that is built on Internet standards. This is a different approach to classic architectures like "Client / Server" and "n-tiers". This service-oriented architecture allows applications to communicate without having to worry about implant technologies on both sides. Interoperability is thus the most important aspect of such an approach.

Definition of a web service

The web service is a software that interacts with others by universal languages (HTTP, JSON ...) and other various procotols. Web services can be of two kinds: Representational State Transfer (REST) and Simple Object Access Protocol (SOAP).

REST Web Services

REST is a set of architectural principles that describe how data can be sent over a standardized interface (such as HTTP). REST lacks a messaging layer and instead relies on design rules for designing stateless services. The resource can be accessed using the unique URI, and a representation of the resource is delivered to the client. The client is said to transfer state with each new resource representation. When using the HTTP protocol to access RESTful resources, the URL of the resource acts as the resource identifier, and the typical HTTP operations to be performed on that resource are GET, PUT, DELETE, PATCH, POST, and HEAD.

Our Architecture

The architecture of our project is described in the diagram below. As it is obvious, our system is composed of three main parts.

- **Frontend:** containing several componenets having parent-child relations between them.
- **Backend:** consisting of multiple web services that are fully independent. The communication between these services is done with frontend components via RESTful API calls. They are able to query the database when it comes to fetching data and prepare it in the right format for the frontend components.
- **Database:** representing the model layer that organizes access and ensures the storage of structured data in various tables.

Figure 2.8: Application architecture

2.2.2 Detailed Design

2.2.2.1 Sequence Diagrams

Sequence diagrams are a popular dynamic modeling solution. They show how the system's elements interact with one another and with the actors. Messages are exchanged between the components at the heart of a system. Actors use a graphical user interface to interact with the system.
In the subsections below, we'll capture interaction between the objects that make up our app in a sequential order.

Authentication

Authentication is an essential part, throughout the entire application. Only authenticated users can access the application's main features.
Once the user tried to login the authentication service checks the validity of the submitted credentials with a call to the database. If no errors are encountered, the user acquires a token that permits him to use the application. We are talking about the authorization

process.
In case of logging in with wrong credentials, an error message would keep displaying until appropriate credentials are submitted.

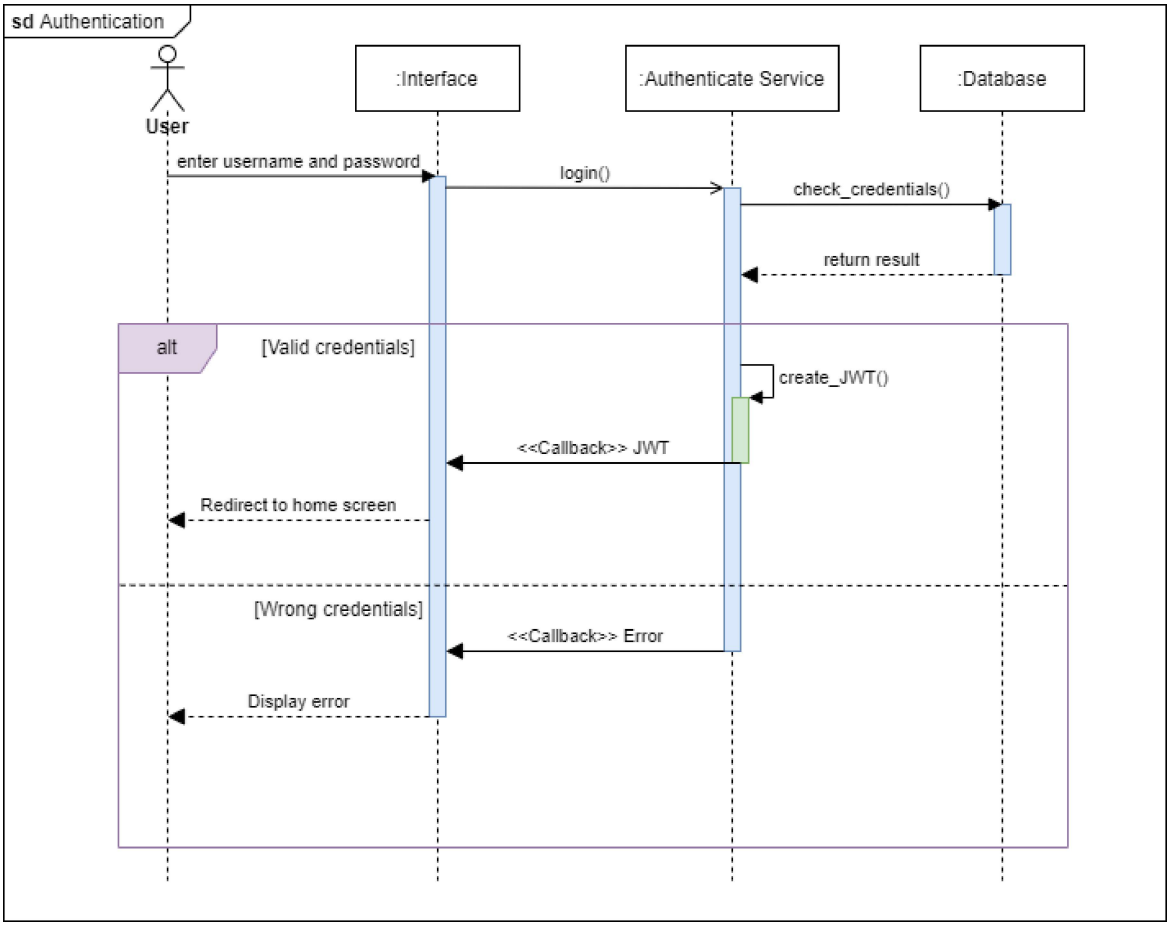

Figure 2.9: Authentication sequence diagram

Manage stores

In the following diagram 2.10, we are presenting a scenario where a user can add stores to the database, select one, or delete them.

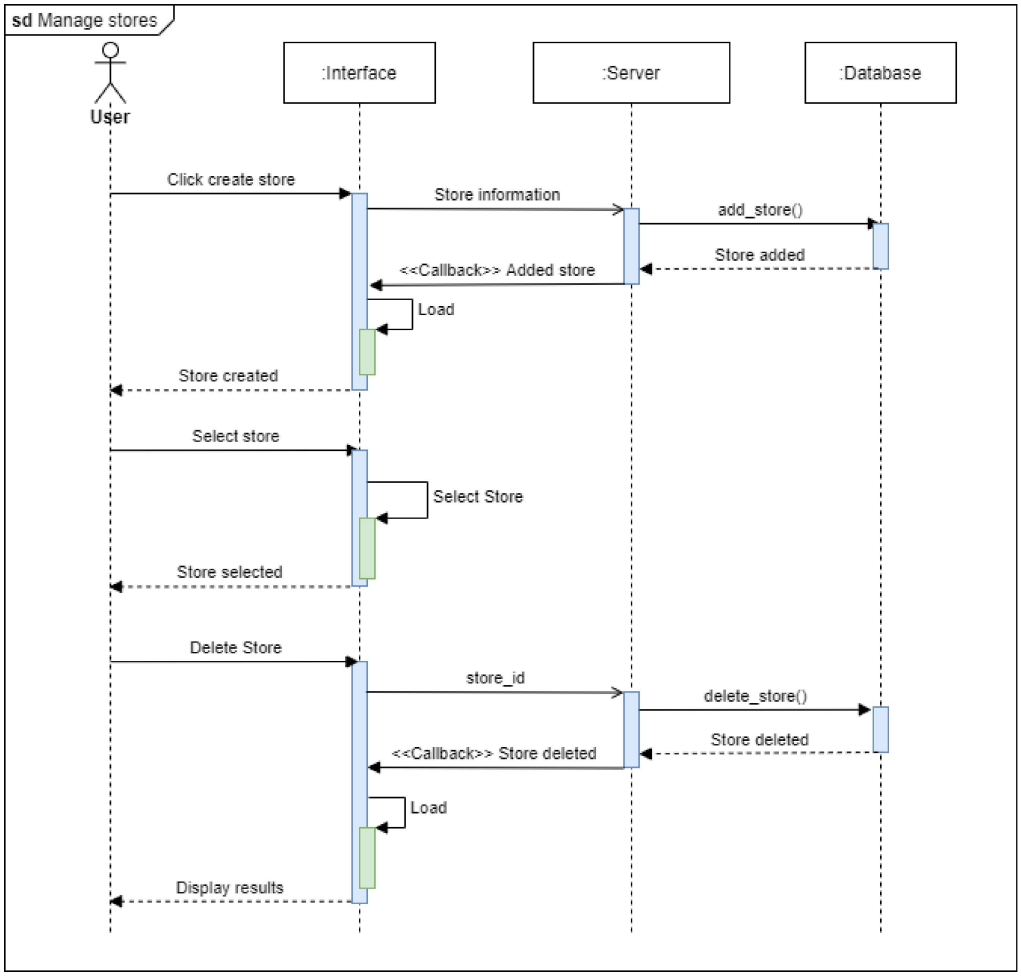

Figure 2.10: Manage stores sequence diagram

The user will manage his stores by directly interacting with the web application. All the data will be added, imported or deleted from the database, Thus the main role of the backend API's is to ensure this interaction. When a user creates a store, the information is transferred to the server side and managed by the API.
If the user chooses a store, the application is updated with that store's information, and the rest of the application's features are made accessible.

Manage products

In the following diagram 2.11, we present a scenario where a user manages stock.

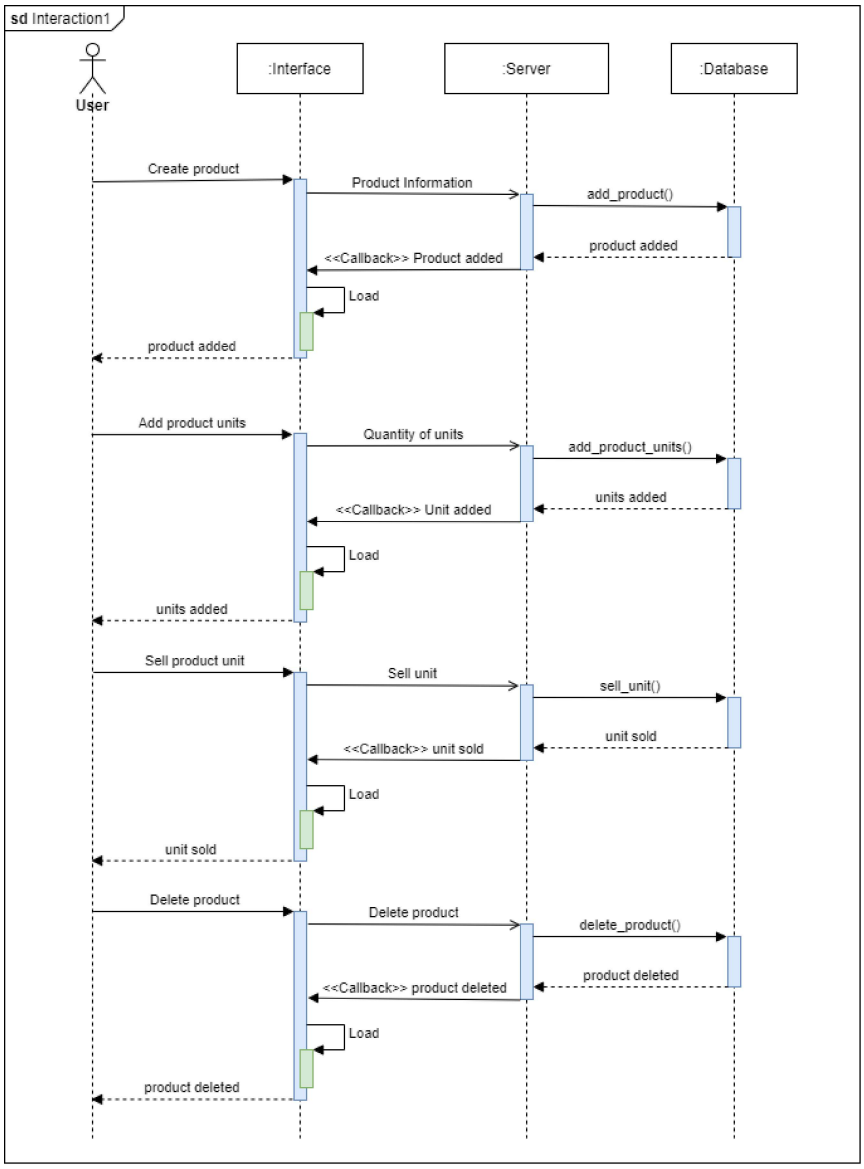

Figure 2.11: Manage products sequence diagram

The user can directly manage the stock by adding products, adding units of these products, selling them and deleting unwanted products.

Manage planogram

the following diagram 2.12 represents the user's interaction with the planogram.

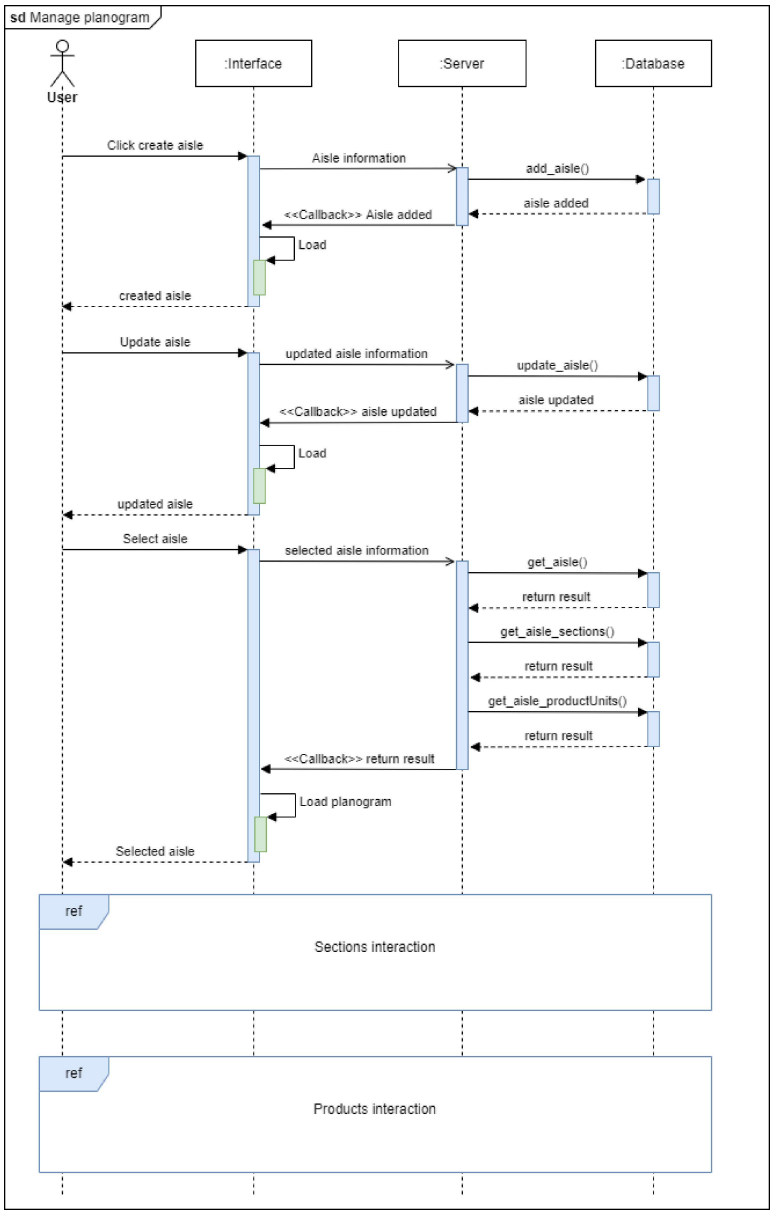

Figure 2.12: Manage planogram sequence diagram

A user can create aisles, update them, select one and interact with sections and products.

Sections interaction

The following diagram 2.13, represents users' interaction with sections

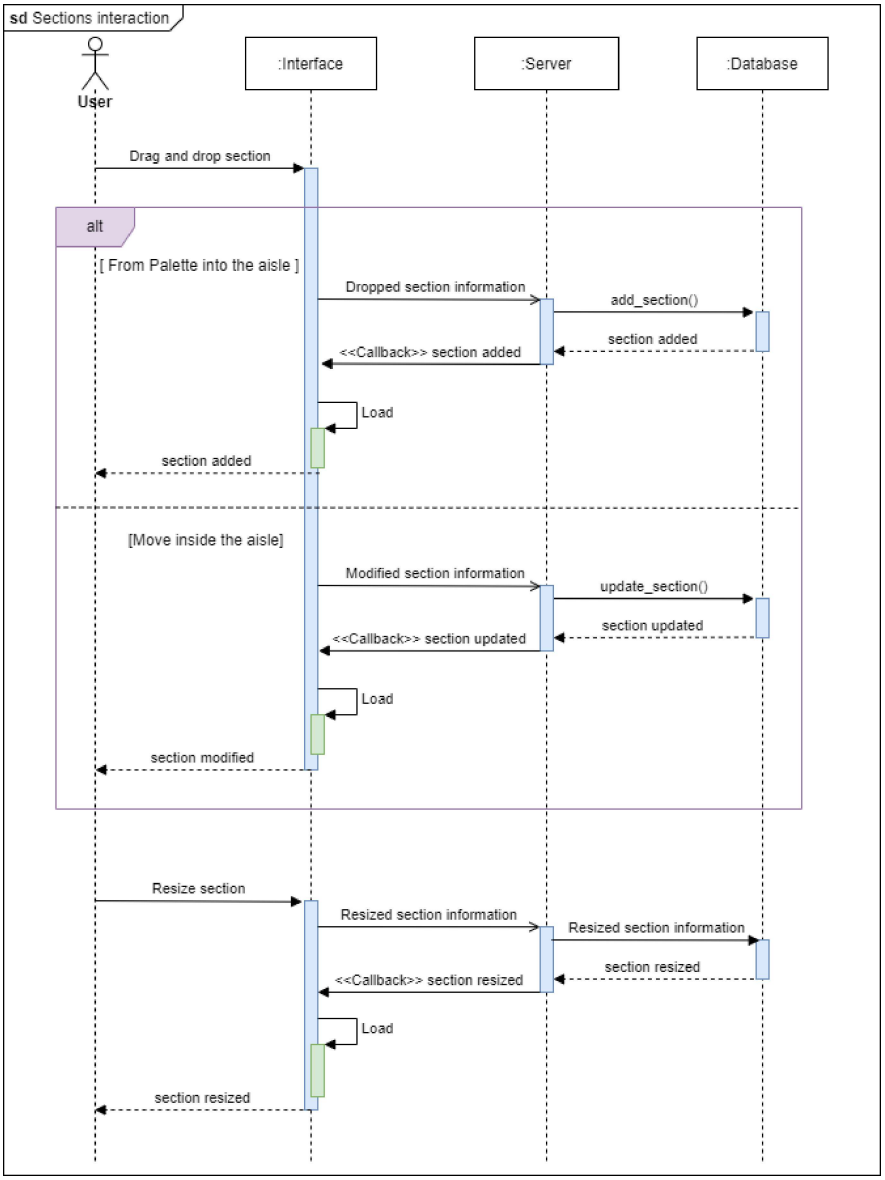

Figure 2.13: Sections interaction sequence diagram

A user can drag sections inside the aisle and update their size, and their position. If the section is dragged from the palette it will result in the creation of a new section in the database.

Products interaction

In the following diagram 2.14, we are presenting a scenario where a user is interacting with the products within the diagram.

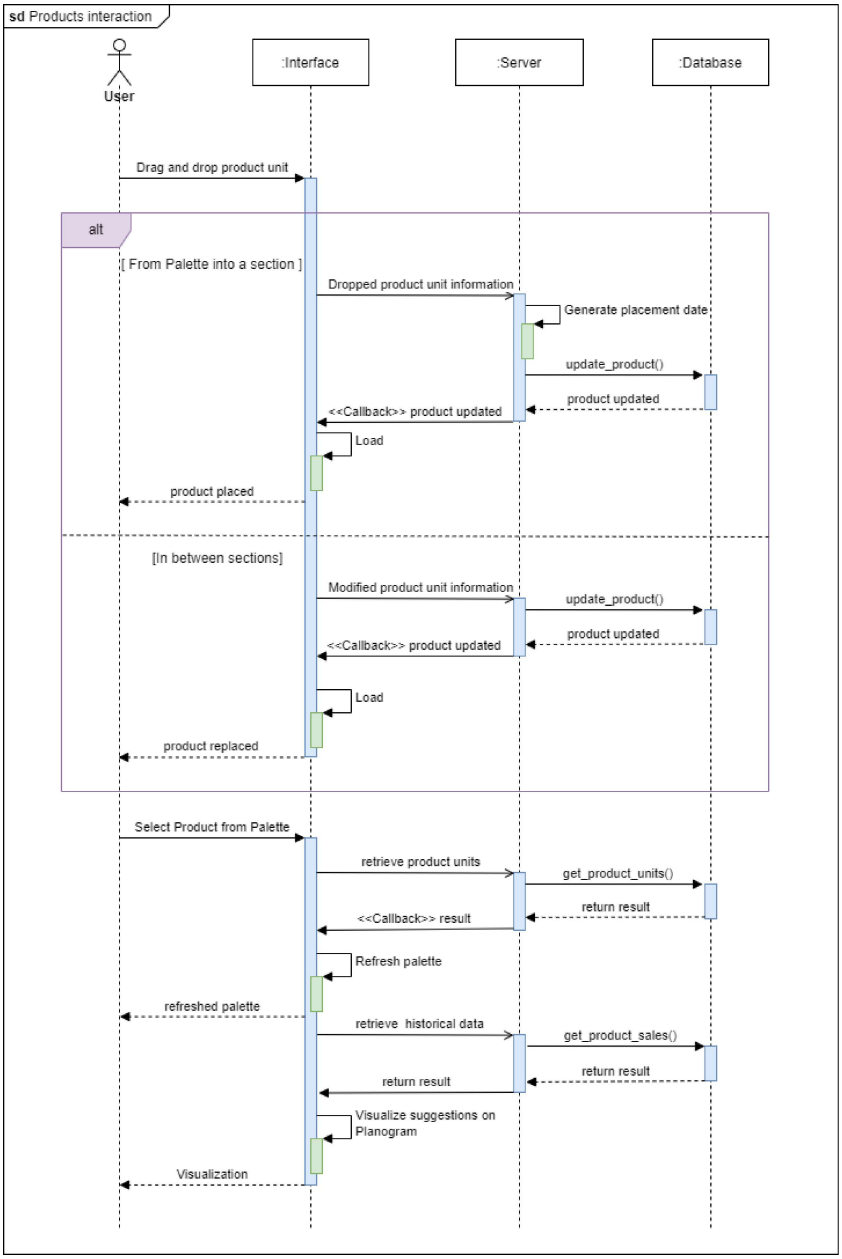

Figure 2.14: Products interaction sequence diagram

The user can select a product which will lead to the visualization of customers purchase behavior using sales data. Select products will be imported into the palette. The user can drag and drop products from the palette into the aisle's sections or in between sections.

2.2.2.2 Class Diagram

Class diagrams are commonly used in the modeling of object-oriented systems in software engineering. These components show a system's structure by displaying its classes, attributes, operations, and relationships between the various objects. The dependencies between the classes in our system are represented in the diagram below.

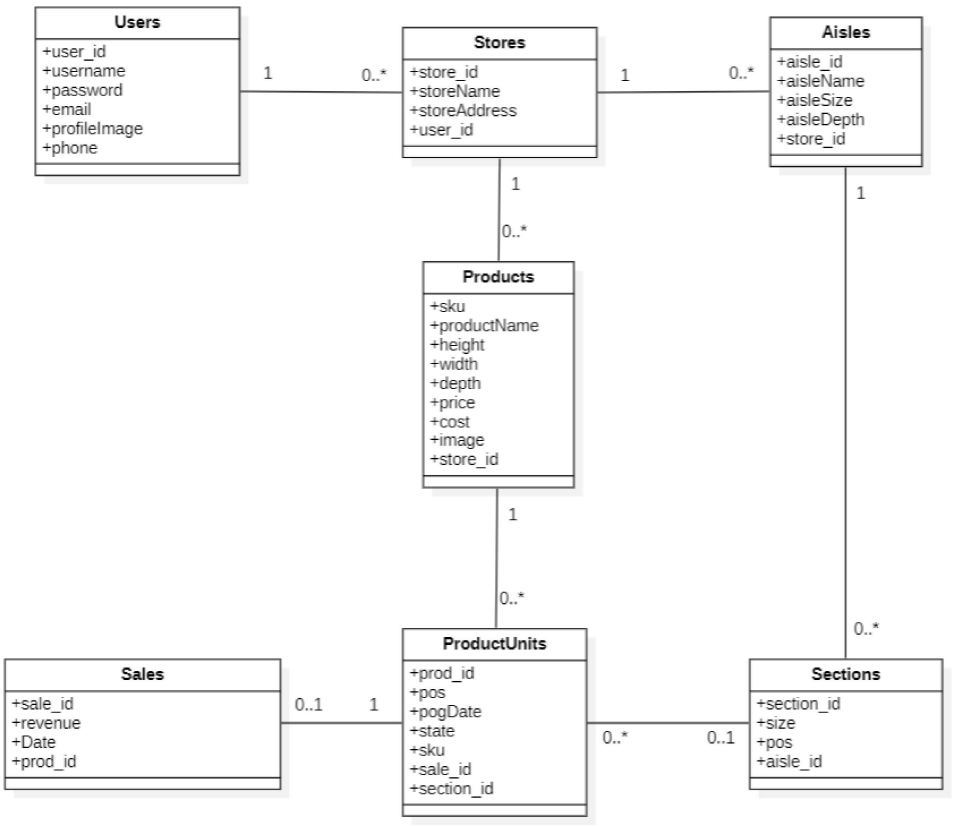

Figure 2.15: Class diagram

2.2.2.3 Deployment diagram

We will give a deeper illustration of our application's components by presenting the deployment diagram. The figure 2.16 Application below represents the various components

involved during the run-time of the platform.

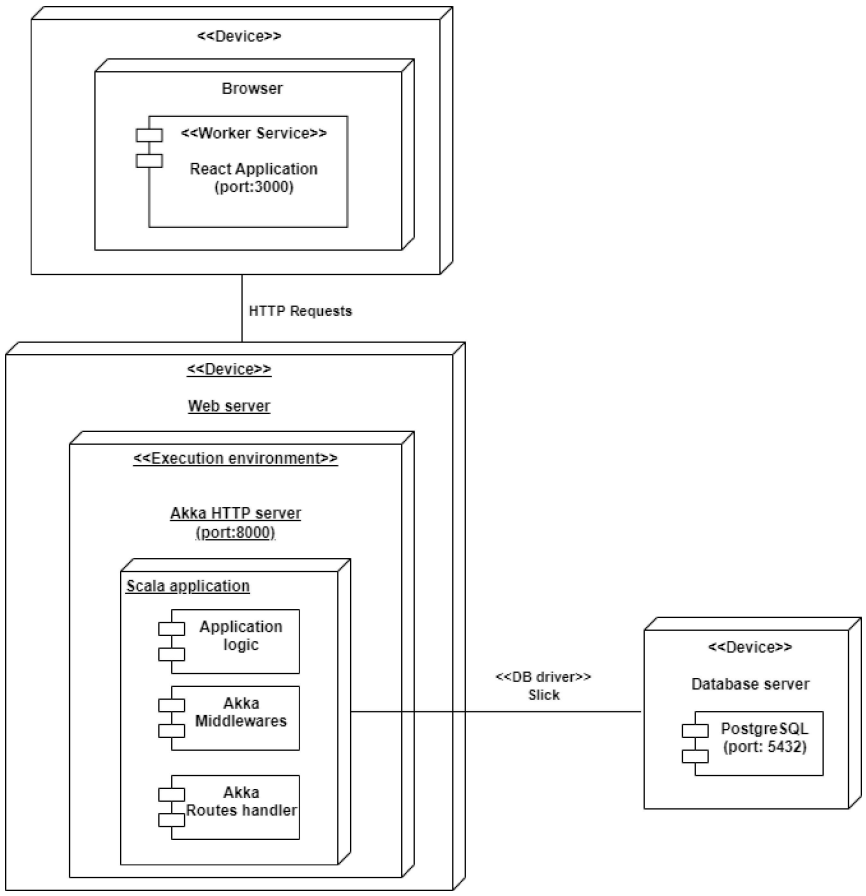

Figure 2.16: Application's deployment diagram

The primary components are as follows:

- **Browser:** The user controls the application via a *Single-Page Application* (SPA). Composed mainly of ReactJS. The communication with the web server is established through the HTTP protocol.
- **Web server:** The back-end node that is responsible for processing the requests from/to the browser. The nested parts are:
  - **Akka:** On top of akka-actor and akka-stream, the Akka HTTP modules implement a full server- and client-side HTTP stack. It is not a web framework, but rather a toolkit for creating and using HTTP-based services.
  - **DB driver:** In this platform we used **Slick**. It is a Scala library for querying and accessing databases. It lets you to work with stored data in a similar way to

Scala collections while also allowing you complete control over when a database access occurs and what data is transferred.

- **Database server:** PostgreSQL is a powerful, open source object-relational database system. It delivers the necessary responsiveness while also assuring greater scalability as data grows.

Conclusion

In this chapter, we went through every facet of our systems' architectures. We presented various diagrams for the broad and thorough designs of our application to help with the transition to the implementation phase. We will go into the environment and realisation phase in the following chapter.

Chapter 3

Environment and Realisation Phase

Introduction

Dividing a project into phases makes it possible to lead it in the best possible direction. In this chapter,we will present the different tools we used, then we will highlight the different stages of realisation of our project.

3.1 Work environment

3.1.1 Hardware environment

We will briefly present the development setup used throught this project. See the following figure 3.1.

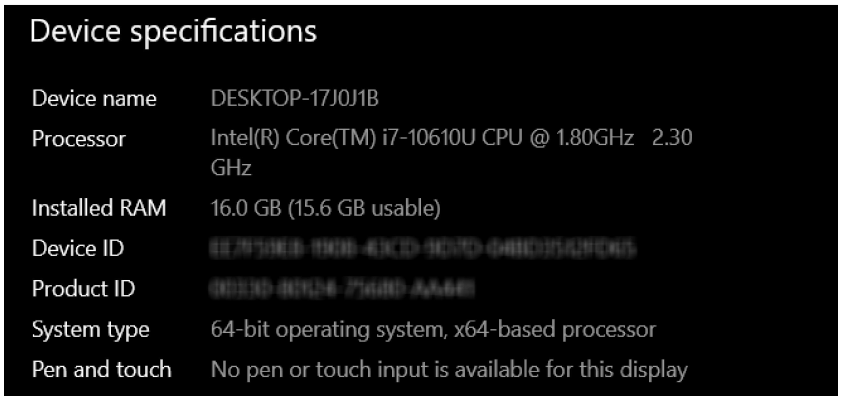

Figure 3.1: Development Hardware

3.1.2 Software environment

In this section we will go over the numerous software tools that were employed during the solutions' design and implementation.

- **Visual Studtio Code:** Visual Studio Code is a source code editor by Microsoft that supports a wide range of languages, including JavaScript. It has an extension library with a variety of useful utilities like version control, terminal, code highlighting, syntax checking, and many others.
- **IntelliJ IDEA:** IntelliJ IDEA is a smart, context-aware IDE for developing applications in Java and other JVM languages like Kotlin, Scala, and Groovy.
- **Postman:** Postman is a very useful tool when it comes to developing and testing RESTful APIs. It offers a simple user interface for making HTTP requests, eliminating the need to write a lot of code only to evaluate an API's performance [5].
- **Slack:** Slack is a workplace communication platform. Not only it provides an instant messaging system with other workplace tools but also it helped us manage the project's progression and stay in touch with the rest of the team especially when working from home.
- **Overleaf:** It's an integrated web-based LaTeX editor. It facilitates collaborative redaction and publication for documents.
- **Draw.io:** It's an online cross-platform tool. Used to build all kinds of diagrams.

3.1.3 Shared Frameworks and libraries

- **Javascript:** JavaScript is a scripting or programming language that allows you to create and control dynamic website content.
- **Scala:** Scala is a strong statically typed general-purpose programming language. It is a high-level programming language that mixes object-oriented and functional programming. Scala is being used by top companies such as Twitter, Netflix, and Linkedin [8].
- **ReactJS:** React is the most popular front-end JavaScript library for web development. Netflix, Airbnb, and Instagram are just a few examples of huge, well-established companies that use it. React is a wonderful alternative to previous frameworks because it adds various new features.When it comes to developing Single-Page Applications SPA, the React framework is ideal since it allows you to rewrite and change content on a web page without having to reload or refresh the page. JSX and Virtual DOMs are the two key features that distinguish React from other libraries

qnd frameworks.
JSX or JavaScript extension combines HTML syntax with JavaScript making it easier for developers to interact with the browser.
Virtual DOM is a virtual copy of the DOM tree created by web browsers that React creates for simplifying the process of keeping track of updates in real-time.

- **React Router:** React Router was used to give the impression that the web application has more than one page. It is a library that keeps the application in sync with the URL. It allows users to navigate between views of different components. Certain components are rendered conditionally based on the specified route in the URL which makes them behave as independent pages.
- **Redux:** Redux is a popular open-source JavaScript library for managing application state. For front-end web development, Redux is typically used with libraries like ReactJS. The difficulty of managing the states increases as the application grows. It's beneficial to work with a library that can assist with state updates and tracking. Redux can be summarized in the architecture in the following figure 3.2

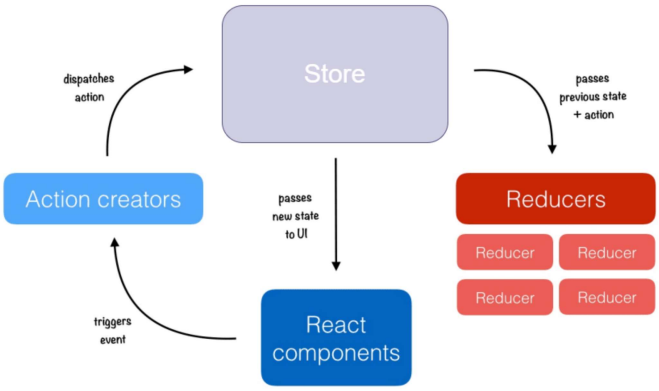

Figure 3.2: Redux Architecture

  - *Store:* The store is the location where the application's current state is loaded. It allows the state to be updated via dispatching actions. Also access to state is allowed by the store. The store is what handles the application's status.
  - *Reducer:* It describes how an action changes one state into another. Reducers define how the state of the application changes as a result of actions passed to the store.
  - *Action:* Actions are information payloads that transfer data from the application to the store. They are the store's only source of information. Actions are plain

JavaScript objects.
Actions only describe what happened, but don't describe how the application's state changes.

- **Dev-Rich-UI:** Dev Rich UI is a library developed and published by Cognira Frontend team. It is a collection of standalone and reusable components built with ReactJS aimed at composing client-specific retail applications and demos. It supports personalized internationalization.

- **Slick:** Slick("Scala Language-Integrated Connection Kit") is Typesafe's Scala *Functional Relational Mapping* (FRM) library, which makes working with relational databases simple. It allows us to work with stored data almost as if we were using Scala collections while also allowing us complete control over when a database connection is made and what data is sent.We get compile-time safety and compositionality when we use Scala instead of raw SQL for queries. Slick's versatile query compiler can generate queries for a variety of back-end databases.

- **Akka:**Akka is a Java and Scala toolkit for creating highly concurrent, distributed, and robust message-driven applications. The Akka HTTP modules implement a full server- and client-side HTTP. Akka HTTP has a rather open design and frequently provides multiple *Application Programming Interface* (API) levels for "doing the same thing." We get to choose the API abstraction level that is best for our application. This means that if we can't get something done using a high-level API, we might be able to get it done with a low-level API, which provides more flexibility but may need us to write more application code.

- **PostgreSQL:** PostgreSQL is a powerful, open source object-relational database system that has a solid reputation for stability, feature robustness, and performance after more than 30 years of active development. It has a solid reputation for its dependable design, data integrity, broad feature set, extensibility, and the open source community's commitment to continually delivering performant and innovative solutions.
  PostgreSQL includes a number of features aimed at assisting developers in the creation of applications, administrators in the protection of data integrity and the creation of fault-tolerant settings, and users in the management of their data, regardless of the size of the dataset. PostgreSQL is very expandable, in addition to being free and open source.

- **GoJS:** GoJS is a JavaScript library that makes creating interactive diagrams in modern web browsers simple. Graphical templates and data-binding of graphical object properties to model data are both supported by GoJS. All the user has to do is save and restore the model, which is made up of simple JavaScript objects that hold whatever properties their application requires. The standard behaviors that

most diagrams require are implemented by a number of preset tools and commands. Customization of appearance and behavior is mostly a matter of setting properties.

3.2 Realization

In this section, we'll go over the implementation methodology we've chosen. Then we go over the steps for putting the various components

3.2.1 Project Management

3.2.1.1 Jira tool

Cognira development teams use JIRA for managing the projects. JIRA is a commercial software product developed by Atlassian. It is used for project management, issue tracking and bug tracking.
Prioritizing, assigning, tracking, reporting, and auditing issues are all possible using JIRA. Indeed, it boosts productivity by reducing time spent tracking down issues and coordinating efforts.
It actually keeps the team on track and allows the project manager to keep track of project progress. Furthermore, it increases quality by ensuring that all tasks are documented in detail and tracked until completion. Furthermore, JIRA is an expandable platform, which means it can be customized to fit a variety of business processes.
It is for that reason, this project was managaed by this tool. The following figure 3.3 is a screenshot that illustrates some Jira tickets assigned to us tracked through the project implementation.

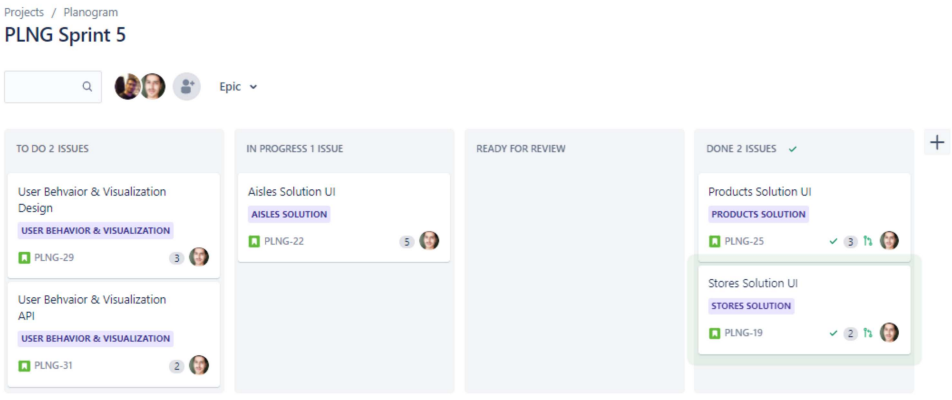

Figure 3.3: Jira tickets

3.2.1.2 Confluence

Confluence is a knowledge and collaboration environment for teams. All of your work can be created, collaborated on, and organized in one spot.
Dynamic pages give your team a space to work on any project or concept by allowing them to create, capture, and collaborate on it. Spaces assist your team in structuring, organizing, and sharing work so that everyone in the team has access to institutional knowledge and the information they need to execute their best work.

3.2.1.3 Bitbucket

The application's file that are vulnerable to manipulation or corruption by viruses or hacking. Therefore, it is a priority to keep one's work in a safe place. For those reasons, we needed a robust and easy to access version control system [2].

Since Cognira uses JIRA and Confluence heavily, it was decided to go for Bitbucket as its version control system especially since these products are owned and were developed by the same company Atlassian which makes their integration easier and without challenges.

3.2.1.4 Communication Platforms

Apart from relying on various and complementary tools to keep track of all parts of the work in progress, communication platforms are critical in keeping the entire team linked and on the same page, regardless of any circumstances that may arise that prevent face-to-face meetings.

Here at Cognira, there is huge dependency on using those platforms to easily propagate information between different teams:

- Slack
- Google Meet
- Google Calendar

3.2.1.5 Design Methodology

To visually represent the system and its main actors, roles, actions and artifacts, *Unified Modeling Language* (UML) is what we opted for as our designing methodology.

UML satisfied a critical requirement in software and system development. It permits us to concentrate on the big picture rather than being sidetracked by a swarms of minor details that should be put off until later. It provides us with a simplified version of a real-world system. Those UML abstractions were developed as conventions to be learned and used easily across different teams and this is what reveals UML's true power

3.2.2 Gantt Chart

The following figure 3.4 illustrates our project's gantt chart which shows all the project tasks displayed against time. Each activity is represented as a bar, with the position and length of the bar reflecting the activity's start, duration, and end dates.

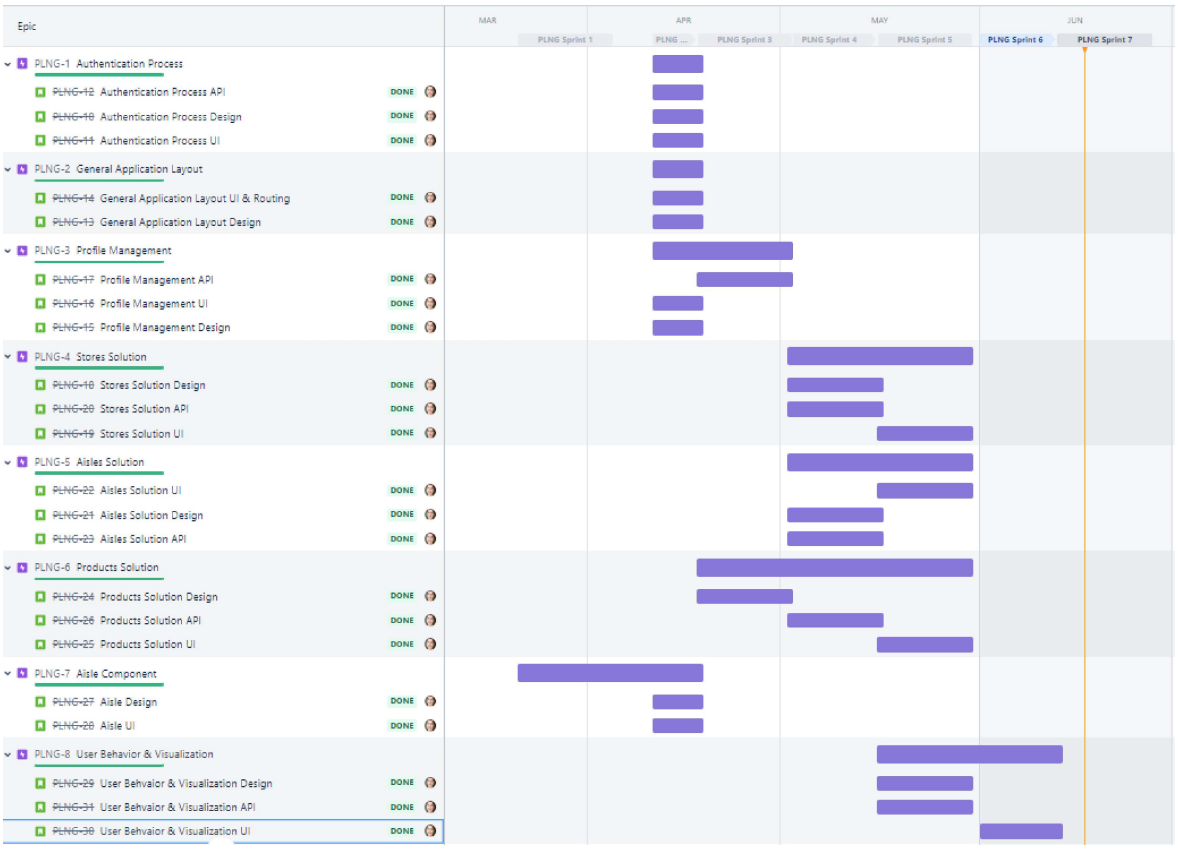

Figure 3.4: Project's Gantt Chart

3.2.3 Realization steps

The project was divided into various milestones that could be completed and assessed in a short period of time. This enabled us to prioritize work on the most important projects while also making it easy for managers to monitor progress. In order to provide clarity to the project, each sprint has three types of tickets.

- **Design:** A the beginning of every sprint we start with designing the API and the *User Interface* (UI). A design allows us to ensure that the program is functional and follows the required process. Later, the design is submitted to Confluence and linked to its ticket.

- **API:** We construct an API that allows us to interact with our back-end services based on the design.

- **UI:** We worked on the UI based on the design. It focuses on the interaction between application users and backend APIs in order to provide the greatest overall experience.

- **Pull Requests:** Once the design is approved by the manager, we start working on the API and the UI. Following the development phase, a pull request is put in place. All file modifications and newly created files are placed for review [1]. Team members will give comments and will expect from us to address them with fixes or clarifications. In the figure 3.5 below we can see an example of an addressed comment.

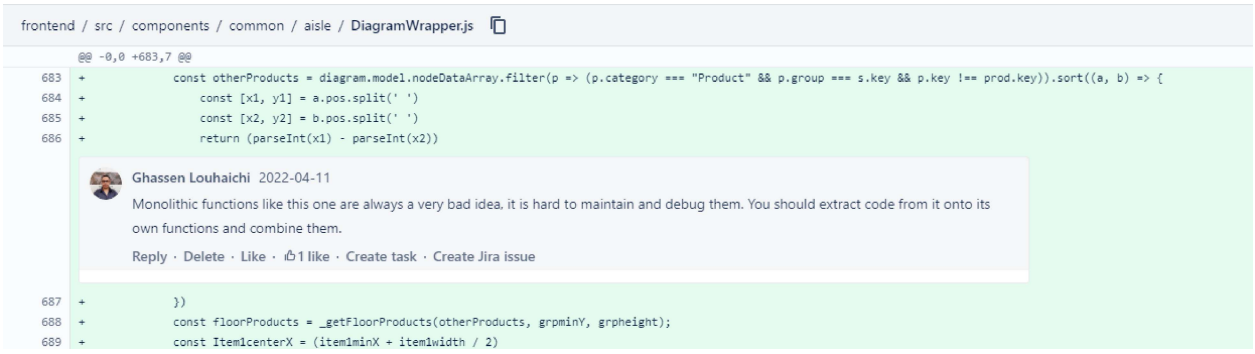

Figure 3.5: Pull Request Comment

Once all the comments are addressed, the reviewers will approve the pull request. following a code merge, we start addressing the tickets of the next sprint.

3.2.3.1 Authentication Process

When a user successfully signs in with their credentials, a JSON Web Token will be returned. Because tokens are credentials, extreme caution must be exercised to avoid security issues. Tokens should not be stored for any longer than necessary. The user agent should provide the JWT whenever they wish to access a protected route or resource, often in the Authorization header using the Bearer schema. The backend API is in charge of generating JWTs and verifying their validity and expiration date. The authentication process is represented in the following figure 3.6.

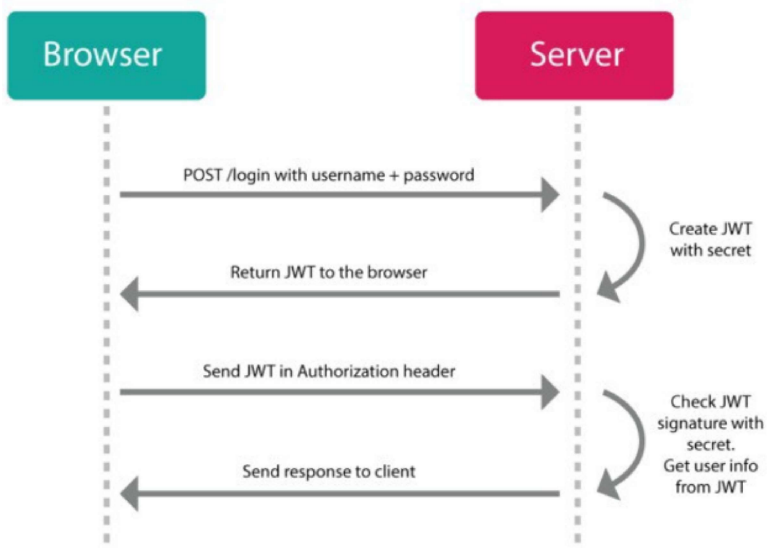

Figure 3.6: JWT Authentication

- **Login:** When a user enters his username and password on the application's login page, this information is transferred to the server. The backend server then compares the password to the hashed password stored in the database. If it does not match, it gives an error indicating that the password was entered incorrectly. However, if it matches, the person is logged in.
  The server generates a JSON Web Token (JWT) with the user's username and ID. This token will be invalid in 7 days. This token is saved in a cookie on the client side in order to keep the user logged in until the token validation expires. On each refresh, the program will pull the current user's information from it, preserving the user's authentication. When the cookie expires, the user must re-login. The username, an ID, and a token are all stored in the login cookie.
  The username, ID, and token will be saved in the redux store once you log in. This will allow them to be used across the application.

- **Register:** When a user registers a new account on the website, they create a unique user account that will be used to validate their identity and allow them to re-enter the account in the future. These details are transferred to the backend server, which checks to see if the username already exists in the database. If not, the password is hashed with a unique key, a new user is formed, and a success response is issued to the user.
- **Logout:** When the user logs out, the cookie is removed from the browser, the current user's information is erased from the redux store, and the user is no longer authenticated to the server. The user is then taken to the Login page.

3.2.3.2 General Application Layout

A website layout is a pattern (or framework) that defines a website's structure. It has the role of structuring the information present on a site both for the website's owner and for users. It provides clear paths for navigation within web pages and puts the most important elements of a website front and center.
The General Application Layout is formed of 3 different sections: Navigation bar, Sidebar and Contents.
We enabled the react routing functionality by configuring the various routes so that we could navigate between pages in our single page application and render specific components that behave as independent pages in the contents frame.
The following figure 3.7 is a showcase of the design.

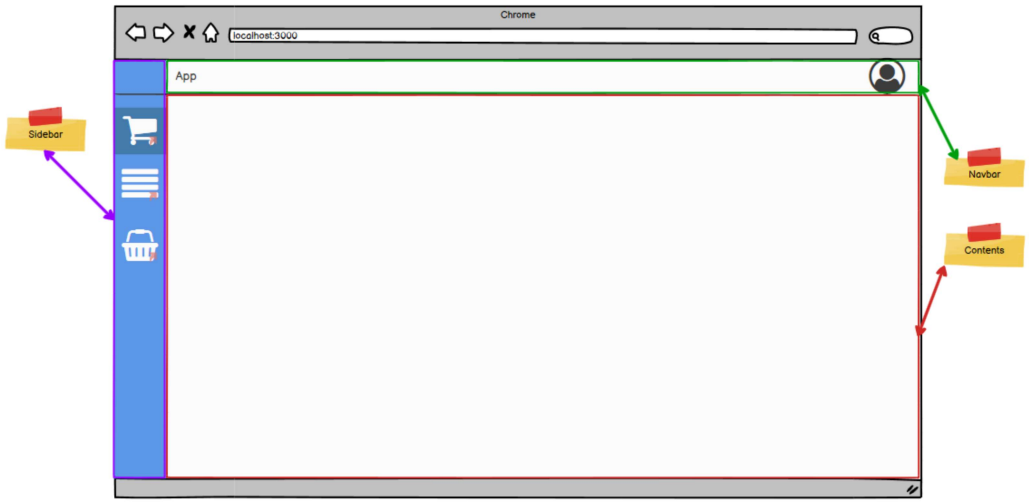

Figure 3.7: General Application Layout

- **Navigation Bar:** containing the application's logo, the currently selected store and

an avatar through which the user can access his profile or logout .

- **Sidebar:** containing the application's different routes(Stores - Aisles - Products) We will go throughout every screen in a moment.
- **Content:** The rest of the frame is where the content will be rendering depending on the React Router.
  - *Stores:* Stores is the page containing the information about the current user's stores. The user can select a store in order to access to the rest of the features.
  - *Aisle:* this page will allow the selection of a specific aisle or the creation of a new one in the currently selected store.
  - *Products:* The products page allows the visualization of the products inside of a store. it also allows the generation of new products and their addition to the store.

3.2.3.3 Stores Solution

The first solution to be developed is the store solution. The built AKKA API endpoint will allow the creation, deletion, and loading of stores. For a GET request, the API will utilize the user's ID from the header token to retrieve the stores owned by him from the database. Once identified, the stores will be returned to the client side in a response, where they will be processed by the React application. When a new shop is created, the backend will return a response with the new store's information. If a store is removed, the response will include a message indicating the success of the deletion action. If there is an error, the application will handle it and return an error message to the client side.
When new data is delivered to the frontend, the application handles the response by saving the data in the response body in the Redux store. The newly updated store will trigger an update in the React application, refreshing the data grid and displaying the data retrieved from the backend.
When we create a new store, a pop-up window will be displayed with the inputs that need to be filled with information about the new store. When you press the add button, a creation request is issued to the API in charge of the store's creation. When the store creation is completed, the pop-up will close and the new store will be added to the data grid. To gain access to the remaining application routes, the user must first select one of his stores.
When a store is selected, the application will trigger an action that will change the store's status and update the selected store. All data associated with that store, such as aisles, products, and sales, will be loaded into the store. A Line-Chart will re-render on top of the data grid with the selected store's data, visualizing all of the store's sales and revenue. When no stores are selected, the buttons leading to the products and aisles routes are

disabled by default. If the selection condition is met, access to the remaining application routes is allowed.

3.2.3.4 Aisles Solution

The Aisle is the main component that will contain the planogram. The diagram will rely on a logic built with Javascript.
The planogram is formed of two parts:

- The Diagram and its different components (Aisle, Sections, Products, Toolkit)
- The Palette containing the Products to be dragged and dropped inside of an aisle.

The following figure 3.8 represents the logic used in order to integrate the GoJS library within the ReactJS application.

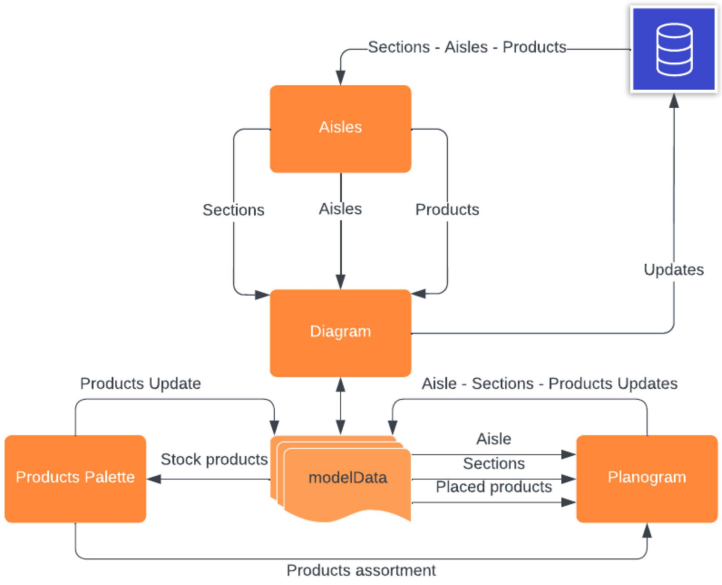

Figure 3.8: GoJS Integration

The Aisles component will import from the database all of the graph components that exist in the planogram as well as the list of products that are currently in stock. This data will be passed to the diagram component, which stores it in a modelData. The diagram is separated from the aisle components to prevent the entire page from being re-rendered instead of just the diagram. The purpose is to improve the application's performance.

Any changes to the planogram or the goods Palette will be sent to the modelData. These changes will be submitted back to the database to maintain it up to date.

Diagram

- **GoJS React-Diagram:** The React Diagram represents the Model, which includes JavaScript Data objects for the Aisle, sections, and products. The model of the Diagram represents the way for identifying relationships between the various components.

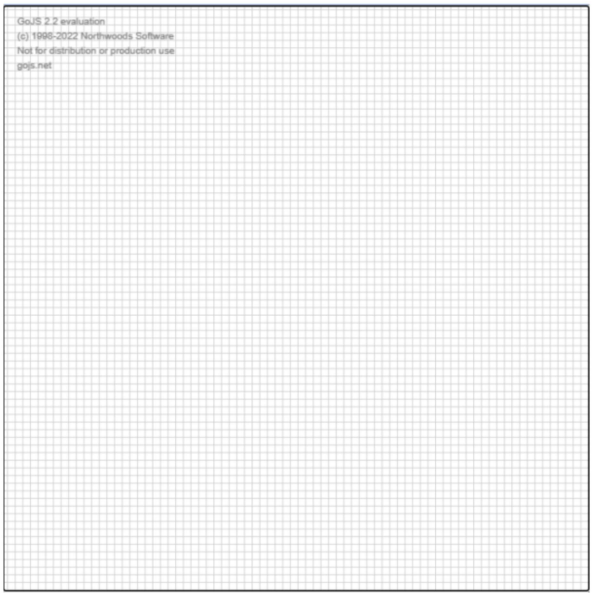

Figure 3.9: GoJS diagram

The Diagram generates a Grid Layout based on the cell size variable, which specifies the minimum size of each graph item in the grid. The cell Size variable can be changed directly from the JavaScript file of the component. Any drag and drop operation that occurs outside of the diagram is canceled, and the graph object returns to its original position. The diagram will include the three graph objects listed below.

- *Aisle:* An aisle is the Layer that will be used to create, position, and manage Sections. The following data will determine the aisle.

Key	Value
key	The Aisle's ID
name	The Aisle's name
size	The Aisle's dimensions
sections	A list of the Aisle's sections

Table 3.1: Aisle GoJS Data Model

The Aisle is drawn directly on the diagram. Once generated, the viewport will concentrate on it and it will be automatically centered within the diagram. Sections can then be dropped inside of it. To manipulate the Aisle, a toolkit is available at the bottom of the diagram. A diagram can only have one Aisle.

– *Section:* The sections are the entities that will contain the products. A section will handle a group of products as if they were a single entity. Sections are resizable, allowing the user to reshape them while placing the products. Sections dropped inside the diagram will be positioned automatically inside the current aisle. If a section is dropped on top of another, the drag-and-drop event is automatically aborted, and the parts are returned to their original location. This will also occur if any of the sections' corners come into touch with the corners of another section. Sections are dependent on the following information.

Key	Value
key	The section's ID
position	The section's position inside of the diagram
size	The section's dimensions inside of the diagram

Table 3.2: Section GoJS Data Model

The section doesn't contain any information about the products it contains. Those information are carried by the products themselves.

– *Products:* This is the entity that represents the store's products. Only products can be placed within the sections. If a product is dropped inside the diagram or the aisle, the drag and drop operation is cancelled and the product is returned to its original position. If a product is dragged and dropped inside another product, it will be positioned on top of it based on the dimensions. Products will be automatically placed at the bottom of the section. If a comparable product exists beneath the dropped product, it will be positioned above it. If one of the product corners interacts with a section, it will be added to and positioned within that section. The following table represents the GoJS data model for the products.

Key	Value
key	The product's ID
name	The product's name
position	the product's position inside of a diagram
group	the key of the section to which the product belongs
image	the product's image.

Table 3.3: Section GoJS Data Model

The products carry the key of the section to which they belong and not the opposite.

- **Palette:** The palette displays a list of selected product units that are still in stock. It also enables for the creation of new aisle sections. The user can drag and drop graph elements from the palette (Sections or Products) into the diagram. When a graph object is correctly inserted, a position and a group identifier are automatically generated.
  It is placed next to the diagram.

3.2.3.5 Products Solution

We created product solutions to handle the stock and the products. To meet the needs of the application, numerous API endpoints have been established. To increase the performance of the data grid component, the creation, deletion, and loading of products and product-units are both independent.
When a store is selected, the client-side will send a request to the backend to load all of the store's existing products in the redux store. Products are a representation of the product units since they carry the information that's shared between all the units.

Products are displayed in a data grid, while product units are displayed in a nested table beneath their respective rows. This table enables the user to manage inventories and visualize products. When a product is expanded or selected, its information is presented in multiple frames above the products table. The nested table will provide information on the current state of a product, such as whether it is placed in an aisle, is still in stock, or has already been sold.

The add product button next to the data grid can be used to create new products. When pressed, a window will appear requiring the user to upload a picture and input details about the new product that will be generated. Units can be added through this window or straight from the data grid.When the create button is pressed, the data is transferred

to the back-end, where it is handled by the API responsible for new product creation, and the product is added to the database. The Redux store will be updated with the new data and will trigger the re-rendering of the updated data grid.

3.2.3.6 User Behavior & Visualization

The final feature to be implemented was the ability to track client purchasing behavior and optimize product assortment depending on it. One of the application's purposes is to track product purchases by consuming the Purchase API whenever a product is passed on sale point terminals, resulting in a specified unit sale. That position will be saved along with the unit's placement date, and new sale date will be generated. In the products grid, a sell button was added to replicate the purchase operation.

When a user selects an aisle and a product, all sales related to that product and that aisle are loaded from the database into the redux store and then update the React application to be visualized in the form of suggestions on the planogram. The intensity of the visualization is determined by the quantity of sold products and their location. The opacity of shared spots will increase, allowing for better product placement on shelves. The implementation of this functionality will be demonstrated in the following chapter.

Conclusion

This chapter covered the work environment and the various tools utilized to meet the project's objectives. Furthermore, we described all of the stages of realization in the second section. In the final chapter, we will examine the product's performance in several scenarios.

Chapter 4

Performance Evaluation and Analysis

Introduction

In this chapter, we will go over the various evaluation outcomes and then discuss the issues that were encountered.

4.1 Key Scenarios

Based on the functional and non-functional requirements laid down at the start of this project, we can see that the Planogram & Sales Optimization Visualization Using Shopper Marketing Behavioral Analytics includes the following important scenarios. We describe them as follows:

- **Authentication and Profile Management** In this scenario we will create a new account, then we will log in with an account and update the profile.
- **Stores Management** In this scenario will we create a new store, and we will demonstrate the impact of selecting a store on the application.
- **Products Management** In this scenario, we will add new products and units to a selected store. We will also visualize the current products.
- **Planogram Management** In this scenario, we will design a planogram, add products to it, exhibit a purchase simulation, and visualize the purchasing behavior of clients.

4.2 Testing Defined Scenarios

4.2.1 Authenticating and Profile Management Scenario

Let's try to create a new user. Empty fields, less than eight characters password and non-matching passwords will generate different types of error. In the figure 4.1 below we have an example.

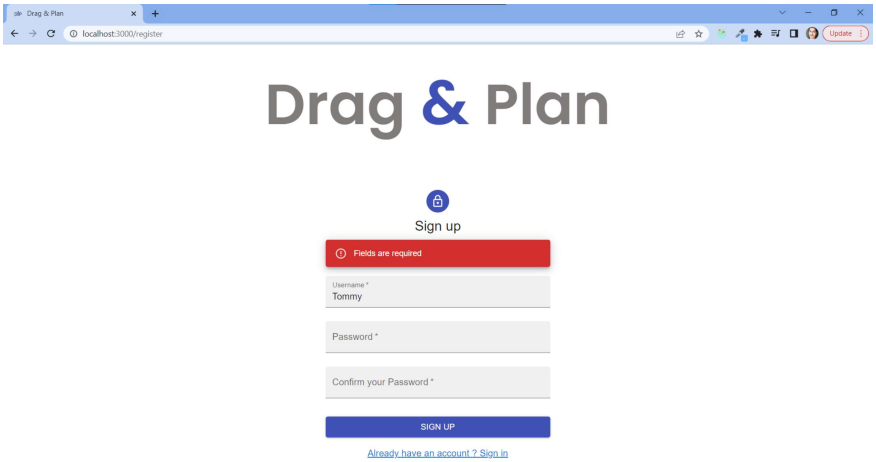

Figure 4.1: Register Test

If the user successfully registers, a pop up will inform him and will redirect him towards the login screen

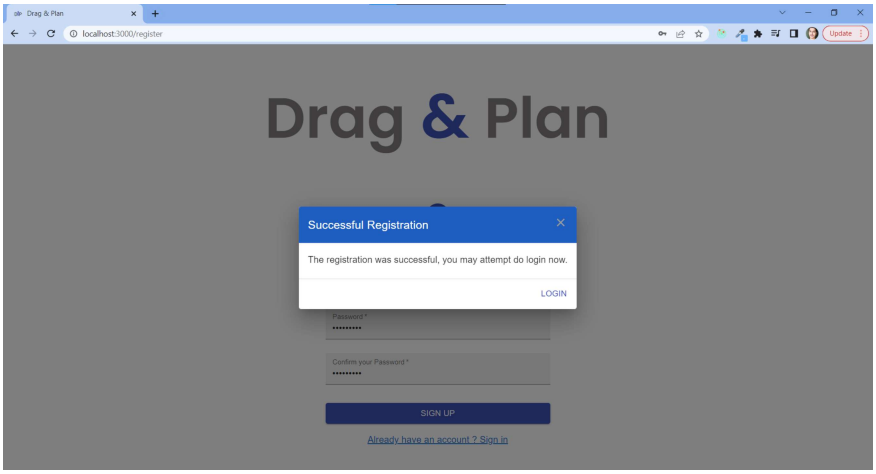

Figure 4.2: Register Success

Same goes for the login part, credentials are checked with the hashed passwords to reinforce the accounts security and an error will pop up in case they are wrong.

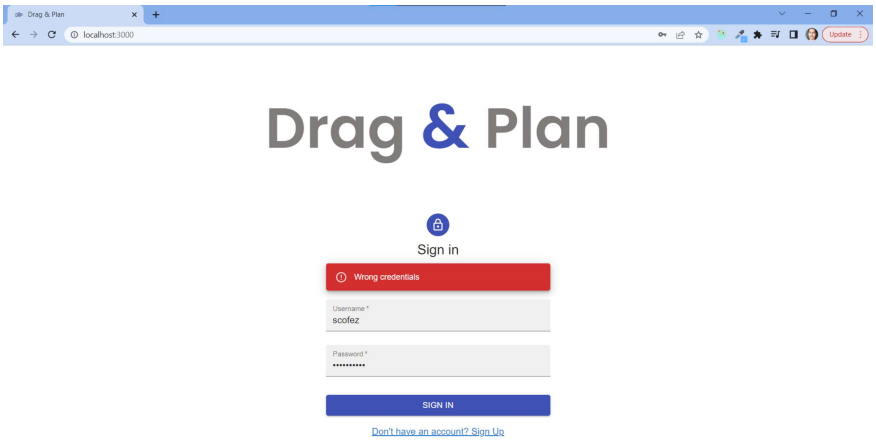

Figure 4.3: Login Test

Once successfully logged in, the user will be redirected to the application's home page. The contents of this page will be discussed in a different scenario. We will press on the avatar and a menu will be displayed contained two buttons: Profile and Log out. We will press the Profile button which will lead us to the Profile page.

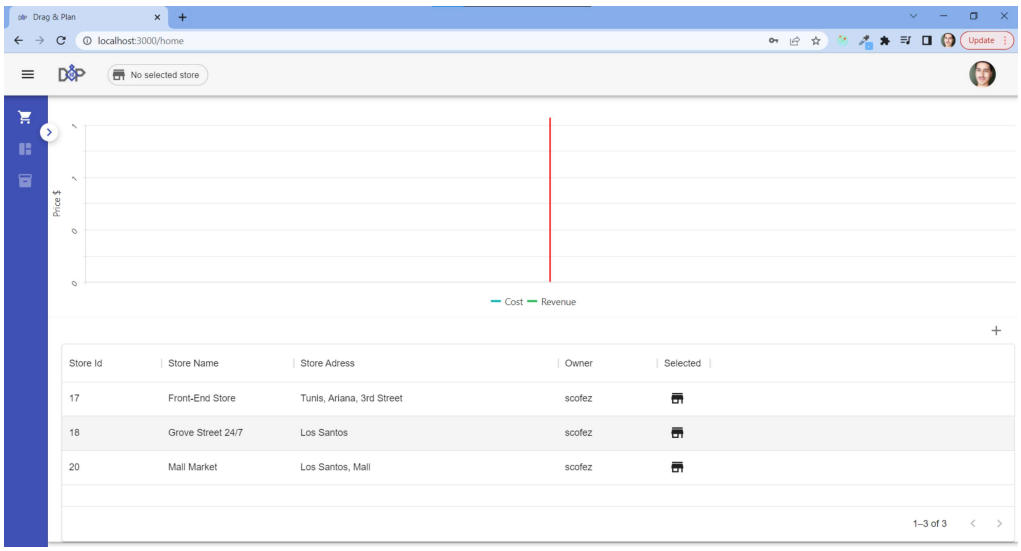

Figure 4.4: Main Page

In the profile page, the user can change his personal information such as profile picture,

phone number, password ... When applying changes, a pop up will inform the user. The figure 4.5 below demonstrates that.

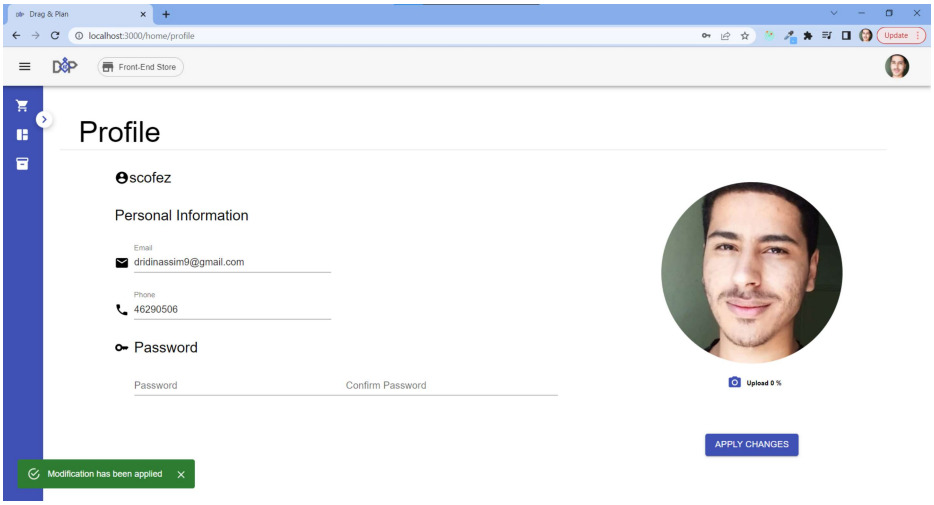

Figure 4.5: Profile Screen

4.2.2 Managing Stores Scenario

If we toggle the sidebar drawer to view all the possible navigation routes we will notice that the access to the rest of the application's features is not allowed yet and that is because there are no stores are selected yet. The figure below illustrates the situation 4.6

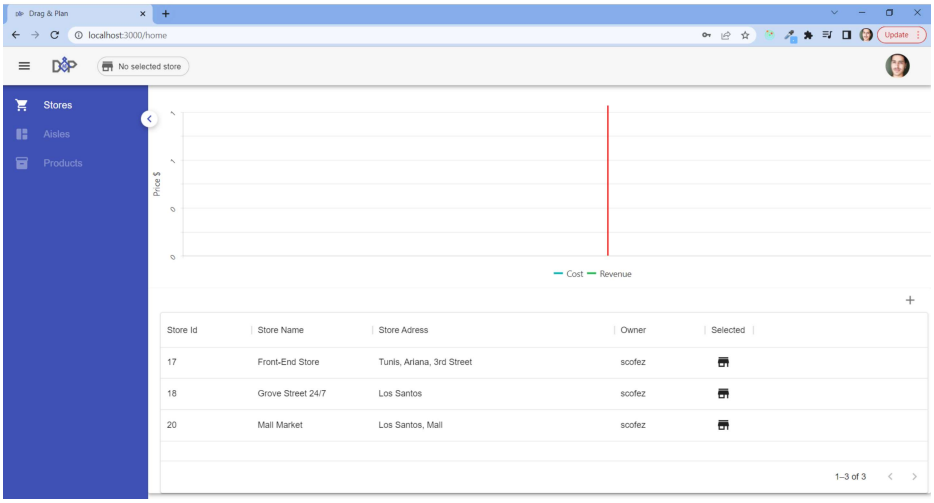

Figure 4.6: Stores Tab

The user has the option of adding new stores to his stores tab. After providing the store name and address, click the add button to finish the creation process.

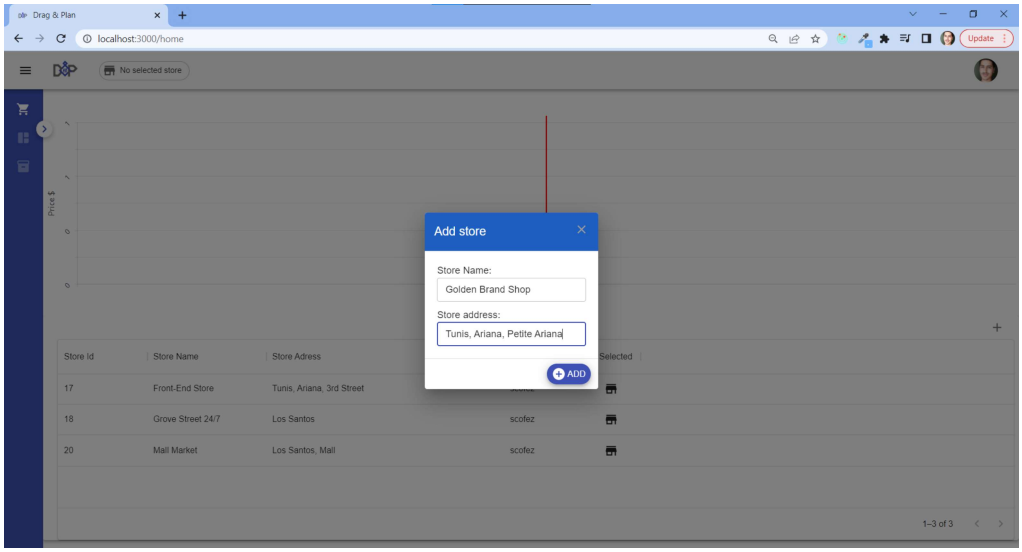

Figure 4.7: Creating Store

The user can choose a store. After selecting a store, the charts will load that store's sales data; access to the rest of the application's features is now provided. The name of the selected store is also updated in the Navigation bar.

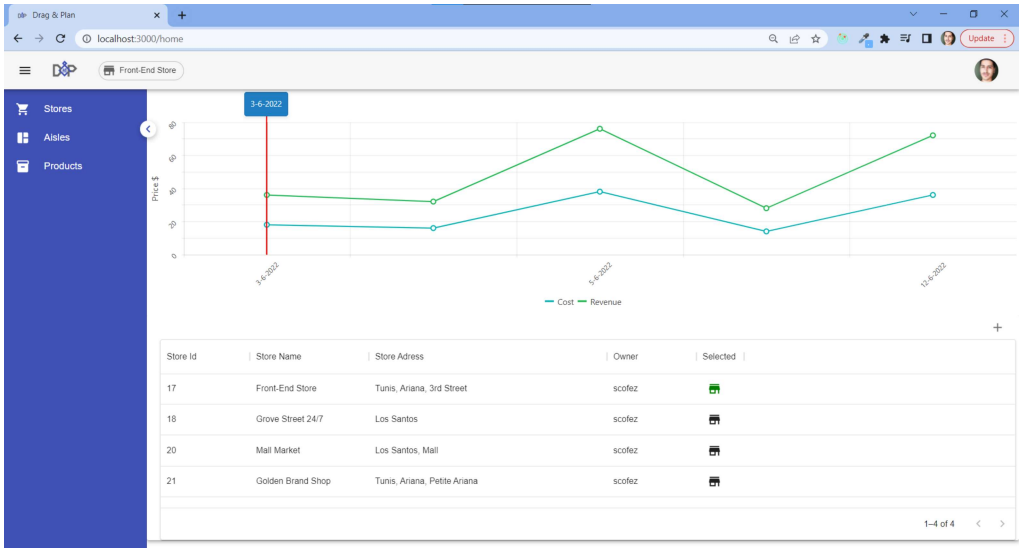

Figure 4.8: Store Metrics Visualisation

4.2.3 Managing Products Scenario

This scenario begins with accessing the products tab after selecting a store. We choose the store Front-end Store, which already has a collection of products. The figure below illustrates the products that are already available in our store. Let's begin by clicking on the first product in the grid that contains all of the products.

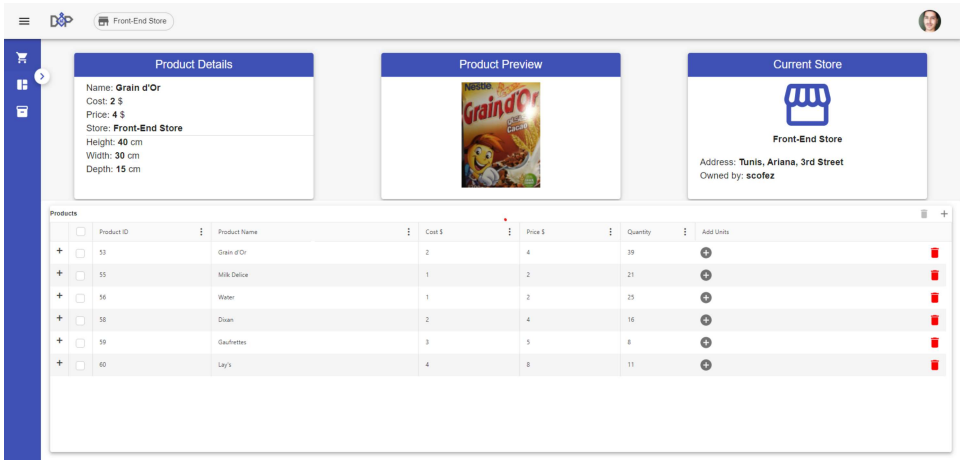

Figure 4.9: Products Tab

The frames at the top of the table are always updated with the most recently selected or enlarged product, displaying information about it.
Let's try expanding one of the products in order to visualize its units.

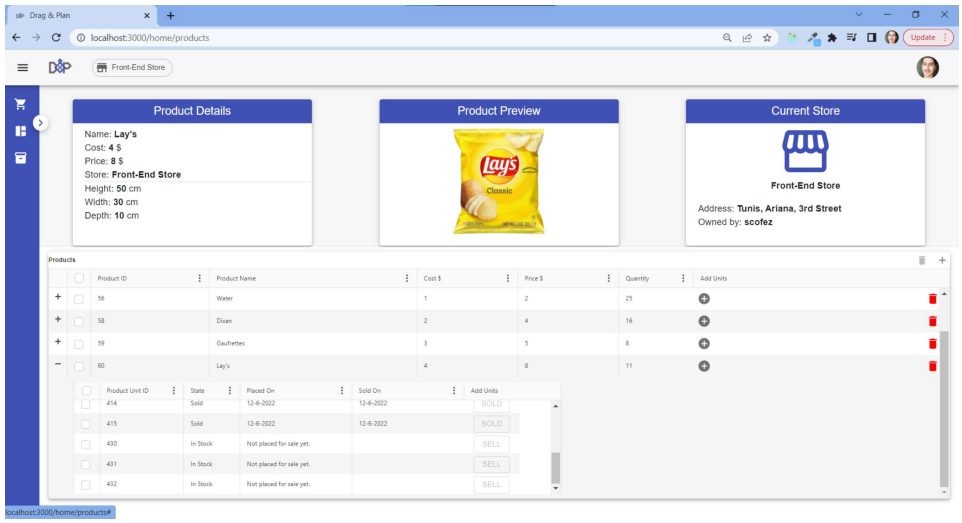

Figure 4.10: Products Tab

The nested grid displays information about the product units and their current status. whether they are currently in stock, advertised for sale, or have already been sold.
The user can introduce new products to his store by pressing the add button to the top right and providing information as shown in the following figure

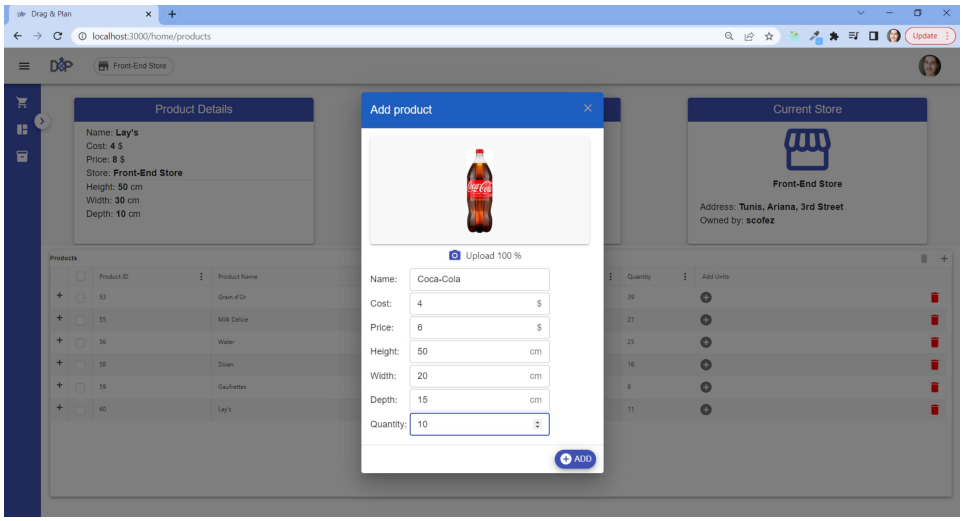

Figure 4.11: Adding Products

After pressing the add button, the new product will be added to the products grid with the quantity of units the user entered.

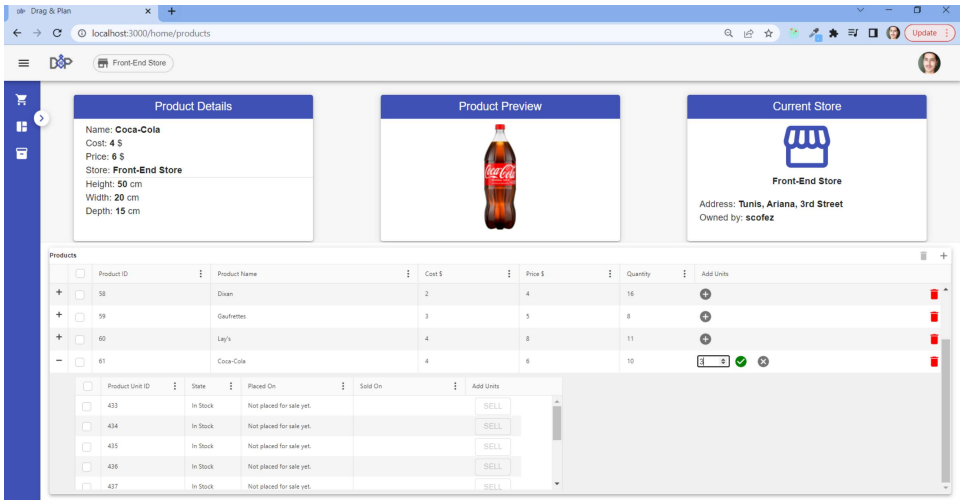

Figure 4.12: Adding Products

All the units added are still in the stock. The user is not restricted to adding units

only when creating new products, they can also add products by pressing the add button in the same row as the product and setting the number of product units to be added. Using the recently introduced products, we will move on to the following scenario where we will interact with the store planograms.

4.2.4 Managing Planogram Scenario

This scenario begins with accessing the planogram tab after selecting a store and making sure that there are stocked product. We choose the store Front-end Store, which already has a collection of products and aisles. After selecting an aisle, the planogram for that aisle will be displayed.

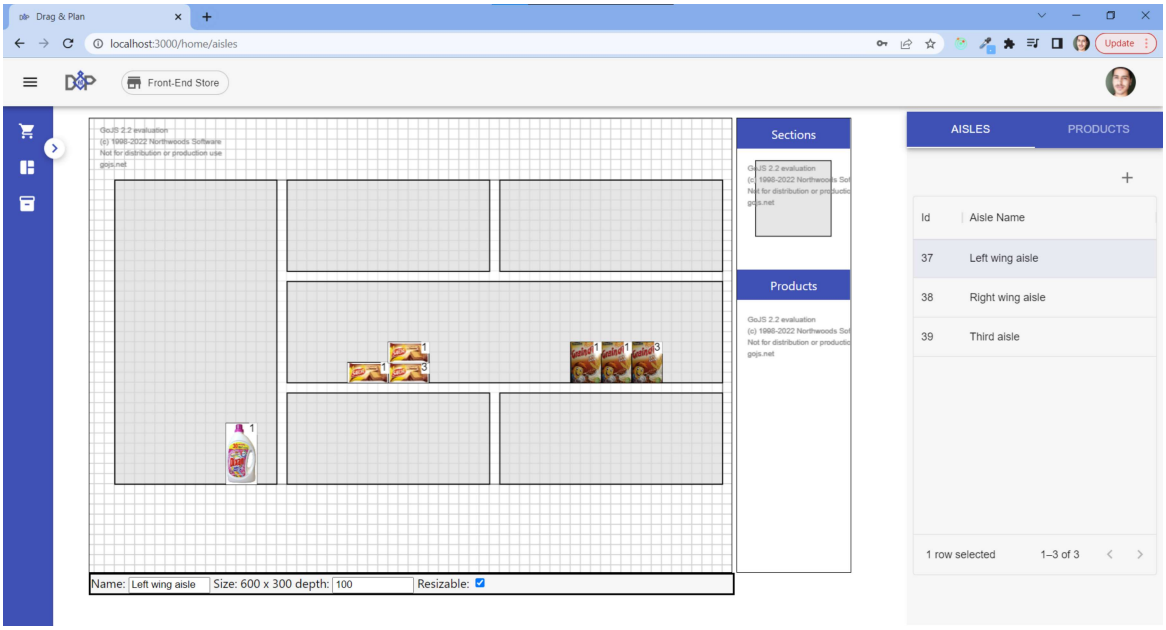

Figure 4.13: Aisles Selection

The user can create new aisles by pressing on the add icon next to the aisles data grid. a pop up will show up asking from the user to provide the necessary information.

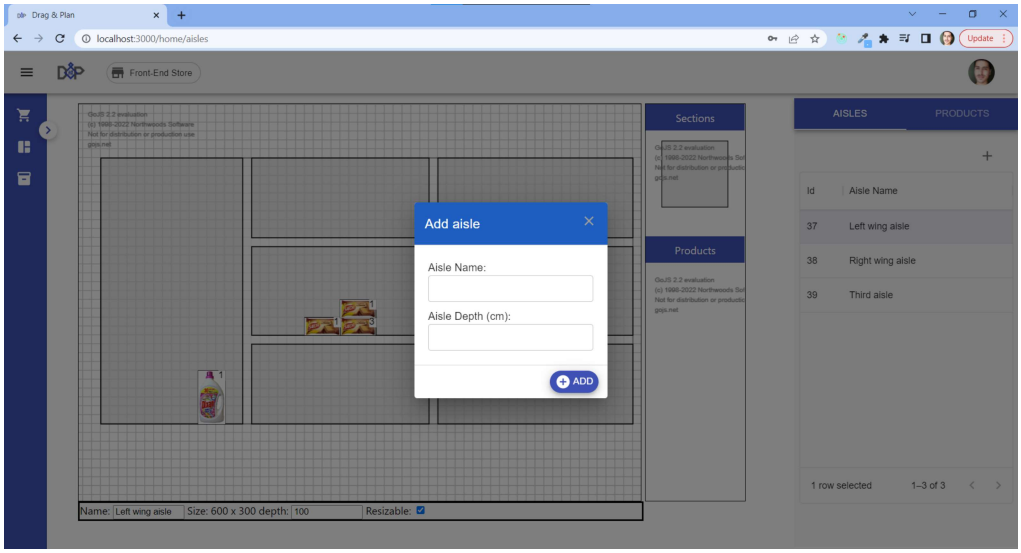

Figure 4.14: Add Aisle

When it comes to the interactions with the planogram itself, the user can resize the aisle and modify its name or depth using the toolkit as shown in the figure below.

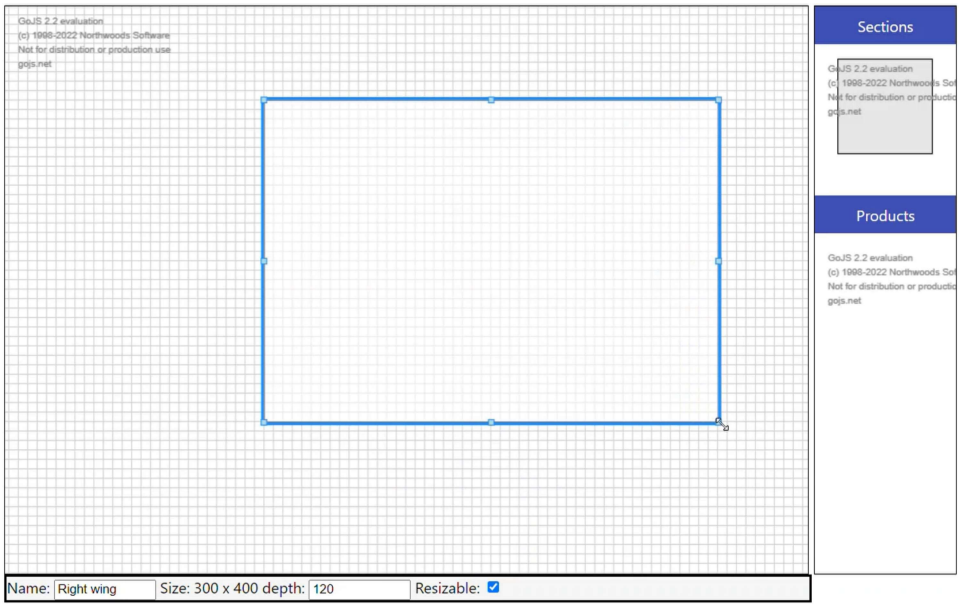

Figure 4.15: Aisle Interaction

The user can drag and drop sections from the palette into the aisle, or from within

the aisle itself. There planogram will always fix the section inside the aisle even when it's dropped outside.

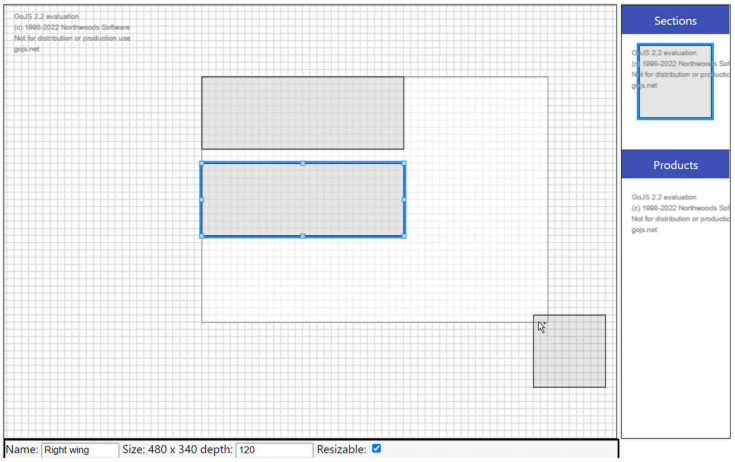

Figure 4.16: Adding Section

Going back to the first aisle, The user can access his products by pressing the products button. the products grid will be displayed instead of the aisles. When the user selects a product, units will be imported into the palette.

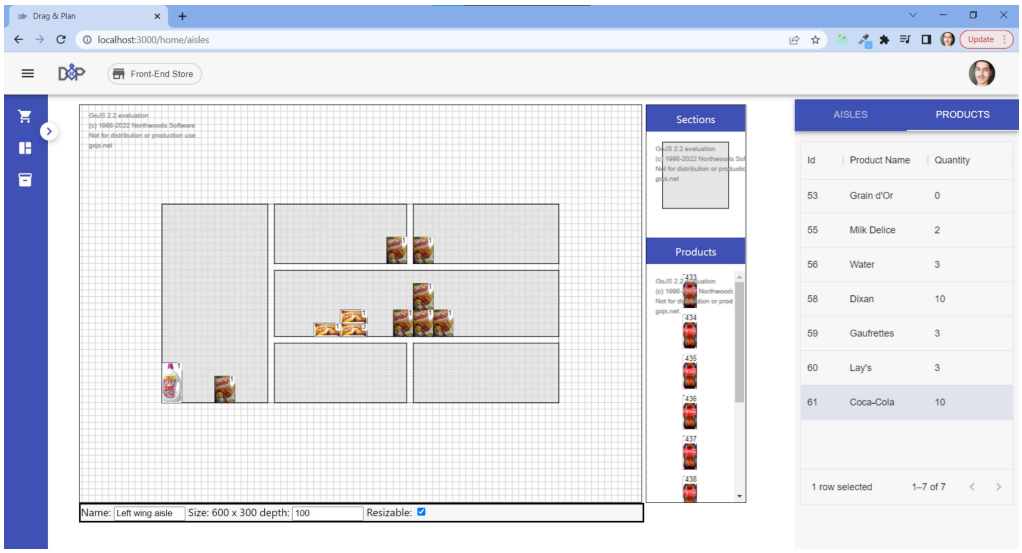

Figure 4.17: Planogram Product Selection

If we navigate to the products tab we can see that the application is notified the planogram updates and that the product units generated a placement date.

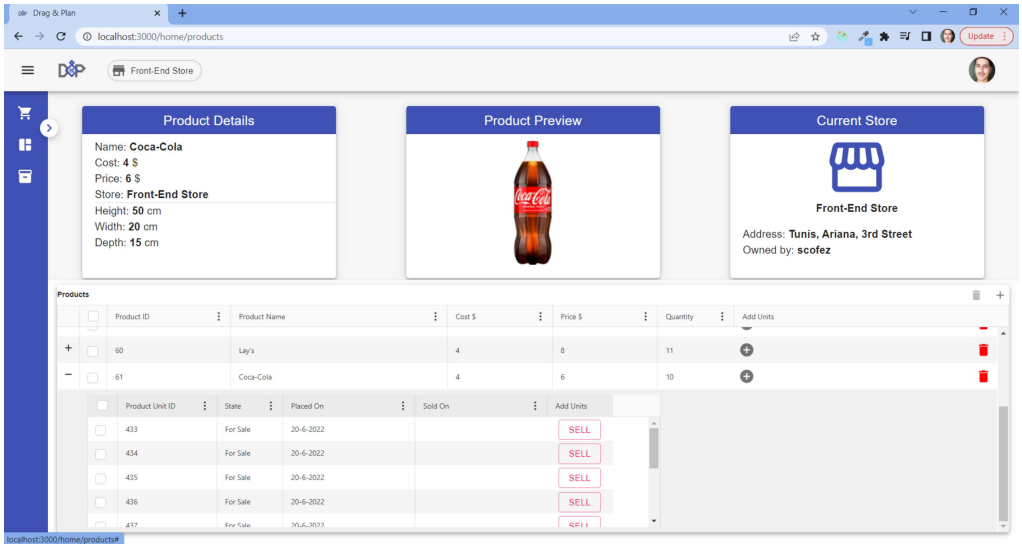

Figure 4.18: Product Updated State

Now we are going to simulate a purchase operation, we are going to sell all the coca-cola goods placed in the aisle and we are going to add 3 product units. a sale date was generated following each purchase operation.

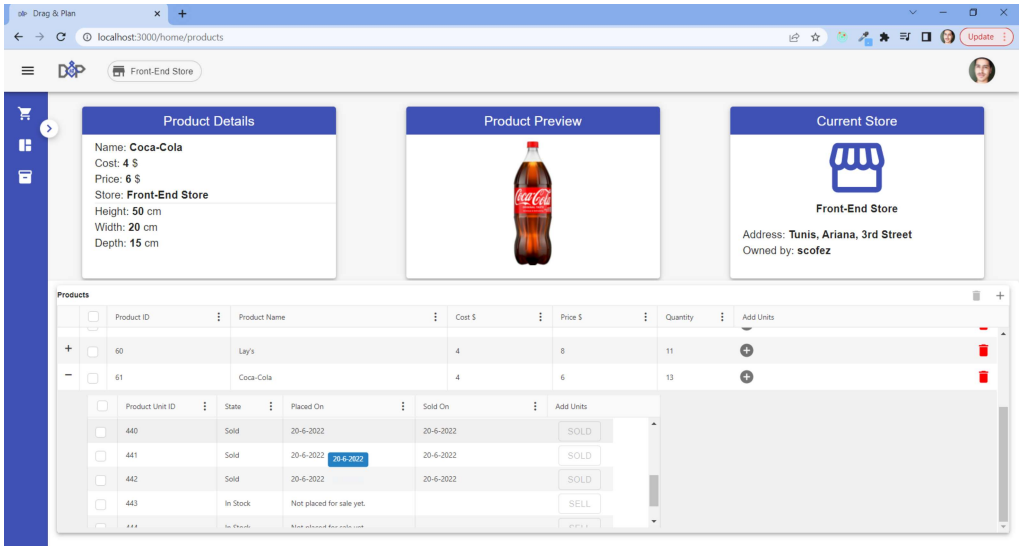

Figure 4.19: Selling Products

If the user navigates back to the planogram in order to place more products of coca-cola, the planogram will visualize suggestion based on users purchase behavior.

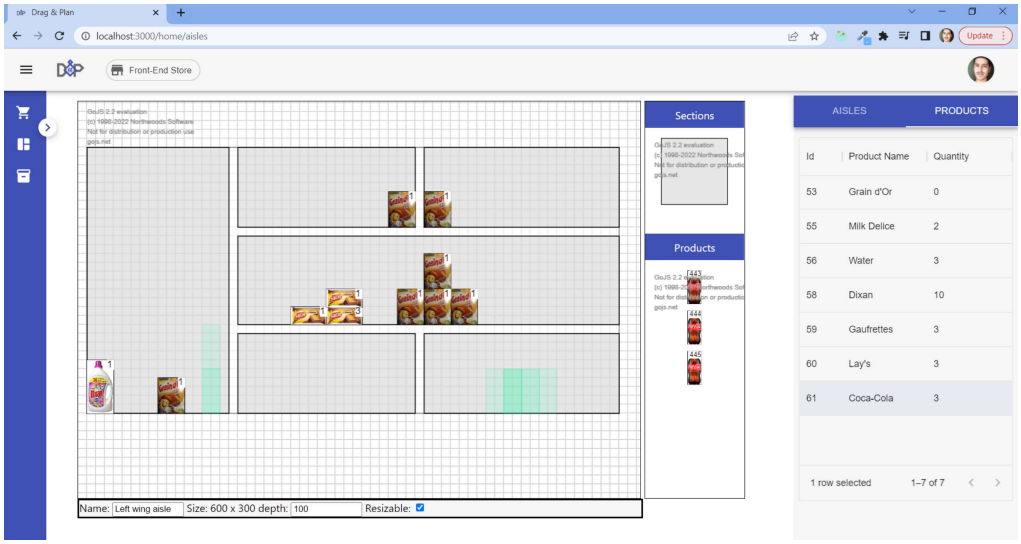

Figure 4.20: User Purchase Behavior Visualisation

Using these suggestions, the user can optimize his planogram by placing products in positions where they will make most sales.

4.3 Difficulties and Challenges

This section discusses the numerous problems arising, as well as the key causes and corrective measures that have been implemented. An adequate description will aid in defining the issue and determining its severity.

4.3.1 Importing Dev-Rich-UI Library

We encountered issues when attempting to include the Dev-Rich-UI library into our application. The application needed the use of a Cognira VPN, which provided us access to the company's private projects. As a result, we requested access to a VPN, even if just temporarily. The DevOps team handled our request by providing personal VPNs and access privileges to all interns, allowing us to access the library and use its reusable components.

4.3.2 Integrating GoJS with ReactJS

Integrating GoJS proved challenging. Re-rendering the diagram with each data change slowed considerably the application's performance.
We separated the diagram from the rest of the page components, and instead of letting ReactJS handle data updates, we implemented a logic that allows the data to stay in sync without the need to re-render the component with each update. It was our responsibility to create an architecture that keeps the planogram and the database synchronized without direct communications between the two entities.

4.4 Perspectives

The selling process is currently a basic API that can be accessed with the click of a button. The application's products data grid is also used for new product units insertion. The application's next step as a future enhancement will be to put it in a real shop, with the addition of new products and units done via bar code scanners. When it comes to sales, whenever a product is sold using a point of sale terminal, the API responsible for product sales should be called, automating the entire process.
The logic of the planogram's graph objects can be further be strengthened and improved. Moreover, AI can be implemented into the application as a feature responsible for automatically rearranging the products to aid in the optimization of planograms.

Conclusion

This chapter we featured some of the most important interfaces of our applications in the form of scenarios. Furthermore, we have discussed the types of challenges encountered as well as the project's future potential. The next part will conclude this report with a general conclusion.

Conclusion and Perspectives

During this graduation project, entitled Planogram Sales Optimization Visualization Using Shopper Marketing Behavioral Analytics in partnership with Cognira, Our goal was to create a representative model of a store aisle that allows the user to rearrange products from the stock into the aisle, track customer purchase behavior using analytical charts and heatmaps, and visualize the impact of the current arrangement on sales forecasts based on customer trends. To do this, we followed the many steps of a software development cycle established by the SCRUM methodology in order to ensure an efficient, scalable, and reliable code. The created application allows its users to manage their stores with minimum effort while visualizing the revenue and the expanses. Users can manage the stock by consulting the store products and the current state of the units in it. Moreover, the user can create planograms by designing the store's aisles in the form of data-driven diagrams. Products can be dragged from the stock directly into an aisle's shelves which will trigger a number of updates to keep tracking that product. Once a product is sold, it generates sales data that will be used to analyze the user's purchase behavior and improve revenue by optimizing products assortment using visual suggestions.

Drag & Plan makes it simple to create aisles and shape them. It has no restrictions on customizing or adding new products. The components of the planogram are created on top of a logic that helps its creation by establishing a number of interactions between the shelves, products, and aisle. These interactions are determined by a variety of factors, including the dimensions and location of the components. The software monitors products in real time. Our solution has been validated following several tests performed at the code level and at the web and mobile application level to ensure it is working properly.

Furthermore, challenges and difficulties experienced throughout the development of

our product should not be overlooked. At the start, goals and expectations were a bit vague and as we progressed through the project, additional directions emerged. At the start of the project, finding a suitable library to develop the planogram and its logic proved difficult, if not impossible. It took a few weeks of trial and error while trying with various libraries and packages. Sprints were delayed as a result, and time became the most valuable asset. In general, integrating several libraries within the application caused performance concerns that needed to be addressed.

When it comes to perspectives, Drag & Plan can be enhanced so that it can be deployed in stores and incorporated into their systems. Product bar code scanning and point-of-sale terminals can be used to increase sales and track client purchasing behavior. In order to be used directly by clients, this requires more development and must give higher levels of sophistication. In terms of optimization, machine learning can be implemented into the application to analyze historical data and improve product assortment suggestions.

Bibliography

[URL1] Atlassian. Atlassian, use pull requests for code review. 2022. https://support.atlassian.com/bitbucket-cloud/docs/use-pull-requests-for-code-review/, Last accessed 06 June 2022.

[URL2] BitBucket. What is git? 2021. https://www.atlassian.com/git/tutorials/what-is-git, Last accessed 02 June 2022.

[URL3] MDN Contributors. Mvc (model-view-controller). 2022. https://developer.mozilla.org/en-US/docs/Glossary/MVC, Last accessed 09 June 2022.

[URL4] Allie Decker. What is a planogram and its role in visual merchandising? 2022. https://www.shopify.com/retail/planogram-visual-merchandising, Last accessed 20 April 2022.

[URL5] Kevin Farmer. What is postman, and why should i use it ? 2021. https://www.digitalcrafts.com/blog/student-blog-what-postman-and-why-use-it, Last accessed 02 June 2022.

[URL6] Matthew hudson. How to create and use a retail planogram? 2019. https://www.vendhq.com/blog/what-is-a-planogram/, Last accessed 21 April 2022.

[URL7] Francesca Nicasio. What is a planogram and how do you read it? 2021. https://www.vendhq.com/blog/what-is-a-planogram/, Last accessed 20 April 2022.

[URL8] Alex Periel. How tech giants use scala. 2022. https://sysgears.com/articles/how-tech-giants-use-scala/, Last accessed 14 June 2022.

[URL9] Juliano Rabelo. Three-tier architecture. 2021. https://www.techopedia.com/definition/24649/three-tier-architecture, Last accessed 24 Mai 2022.