Plataforma e-commerce e infraestructura interna profesional No.Studio

E-commerce platform and internal business infrastructure for No.Studio

Grado en Ingeniería del Software

Facultad de Informática

UNIVERSIDAD
COMPLUTENSE
MADRID

Trabajo de Fin de Grado

Matteo Isnenghi Tamayo

Curso 2022/2023

Director: Manuel Montenegro Montes

Agradecimientos

Le doy las gracias a Manuel Montenegro Montes, el director de este proyecto, por su completa disponibilidad y flexibilidad a la hora de ayudarme en cualquier problema que haya aparecido durante el desarrollo.

Agradezco también a la cliente del proyecto y dueña de la empresa No.Studio, Irene Sánchez Lillo, por confiar en mí como desarrollador de este proyecto, y por su completa disponibilidad a la hora de realizar las reuniones necesarias.

Agradezco también a mi familia por el esfuerzo, apoyo y cariño que me han mostrado siempre. Han sido la mayor fuente de motivación y apoyo, no solo durante el desarrollo de este proyecto, sino durante toda mi carrera universitaria.

Índice

Resumen

Este proyecto desarrolla la transformación digital de una empresa de producción y venta de ropa, llamada No.Studio. La tienda está ubicada en la ciudad de Toledo, y su negocio consiste en la producción y venta de prendas de ropa, de la marca No.Studio, y la reventa de otras prendas de diferentes marcas.

La idea del proyecto nació cuando conocí la necesidad de la dueña de la empresa de digitalizar su empresa. Quería crear un portal digital para la tienda, y tener herramientas para poder gestionar las tareas correspondientes a su negocio. Decidí que la mejor idea para resolver las necesidades de la cliente del proyecto sería desarrollar dos aplicaciones web, separando el objetivo de cada una de ellas.

La primera aplicación consiste en una tienda web, que cumple con las necesidades de gestión que un cliente de la tienda necesita. Esta aplicación permite navegar por los diferentes productos de la tienda, añadiendo los deseados por el cliente en carritos de la compra y realizando pedidos de ellos.

La segunda aplicación consiste en un software SCM (*Supply Chain Management*, en español Gestión de la Cadena de Suministro), es decir, una herramienta que gestiona todas las tareas relacionadas con la gestión de suministros, como la gestión de tallas e inventario de los productos, la gestión de envíos y devoluciones de los pedidos, y las relacionadas con los proveedores de materiales para la manufacturación, y los pedidos realizados a ellos.

Con este proyecto he conseguido realizar dos aplicaciones web que cumplen con las necesidades inmediatas de la cliente del proyecto para comenzar el proceso de transformación digital de la empresa. Las dos presentan una buena base sobre la que asentar incorporaciones de nuevas funcionalidades en un futuro, ya que se han diseñado, tanto a la hora de elegir las herramientas como durante el desarrollo, con esto en mente.

Palabras clave

Aplicaciones web, comercio electrónico, SCM, React, Sails.js, MongoDB.

Abstract

This project consists of the digital transformation of a clothing manufacturing and sales company, called No.Studio. The store is located in the city of Toledo, and its business consists of the manufacture and sales of self-made clothing, with the brand No.Studio, and the resale of clothing pieces from different brands. The brand's style draws heavily from skater and graffiti culture, with teenagers and young adults being the company's target audience.

The idea of the project was born when I knew about the need of the owner of the company to digitize her company. She wanted to create a digital portal to the store, and to have computerized tools to manage the tasks corresponding to her business. I decided that the best idea to solve the client's needs would be to develop two web applications, focusing on separating the objectives of each of them.

The first application would consist of a web store, which would meet the needs of the front-office, that is, towards the customer. This would allow them to go through the different products of the store, gathering the desired ones in carts and ordering them.

The second would consist of a SCM (Supply Chain Management) web application. An SCM is a tool that manages all the tasks related to supply management, such as product sizing and inventory management, order shipping and returns management, and those related to suppliers of materials for manufacturing, and the orders placed with them.

With this project I have managed to create two web applications that meet the immediate needs of the project's client to begin the process of digital transformation of the company. Both of them present a good base on which to build new functionalities in the future, since they have been designed with this in mind, both when choosing the tools and during the development.

Keywords

Web applications, e-commerce, SCM, React, Sails.js, MongoDB.

1. Introducción

Este proyecto consiste en desarrollar dos aplicaciones web para transformar digitalmente el negocio de una tienda de ropa de la empresa/marca No.Studio.

La primera aplicación consiste en una tienda web, a modo de portal digital a la tienda para navegar a través de sus productos e introducirlos en carritos de compra, para sucesivamente realizar el proceso de pago y generar un pedido. Los pedidos se recogen en la segunda aplicación web, un software SCM (*Supply Chain Management*) que permite realizar desde la nube las operaciones de gestión de la cadena de suministro de la empresa. Este programa permite introducir y actualizar los productos a mostrar en la tienda web, permitiendo también especificar y modificar el inventario de cada uno de ellos. El SCM mantiene también un historial de los pedidos realizados por los clientes de la tienda y permite modificar el estado del pedido. Permite también mantener un historial de los proveedores de la tienda, otras tiendas que ofrecen materiales necesarios para producir los productos a vender, y mantener un historial de los pedidos a proveedores, permitiendo también manejar el estado de estos pedidos.

Este capítulo presenta la motivación que me ha llevado a decidir desarrollar este proyecto, incluyendo las razones por las que he decidido realizar un SCM personalizado para la gestión de las tareas de gestión de suministros. Se describen también los antecedentes de aplicaciones similares (tanto para el SCM como para la tienda web), los objetivos planteados al principio del proyecto, y el plan de trabajo seguido.

1.1 Antecedentes y motivación

Existen varias herramientas ya desarrolladas que cumplen la función del SCM pedido por la cliente del proyecto, y, aunque muchas de ellas ofrecen características y funcionalidades interesantes, la mayoría suponían un coste demasiado elevado para el actual negocio, y las que no lo son, no permiten cumplir con los requisitos propuestos por la cliente del proyecto.

El hecho de que los SCMs más potentes sean de pago era un problema para la cliente del proyecto: la empresa es muy joven, y actualmente tiene un alcance e ingresos limitados. Además, gran parte de la funcionalidad ofrecida por estos SCM de elevado coste no era necesaria para el correcto funcionamiento de la web de la tienda, y además por lo general estas herramientas requieren de una formación previa para su correcto uso. Por estas razones decidí, junto con la cliente del proyecto, realizar un SCM que se ajustase a las necesidades de esta última, y con la capacidad de implementar nuevas funcionalidades en el caso de que fueran necesarias.

Como alternativas se plantearon algunos SCM ya desarrollados como Salesforce[1], Zoho[2] , Pipedrive[3] o Odoo[4]. Todas estas aplicaciones son o de pago, u ofrecen un plan gratuito limitado, además de requerir de un período de formación previo para poder utilizarlos.

Respecto a la tienda web, también existen herramientas para generar páginas web de tipo *e-commerce* sin la necesidad de desarrollarla desde cero, pero los inconvenientes son similares: la gran mayoría son de pago, y además presentan un grado de libertad creativa y funcional limitado, ya que se basan en plantillas. Por tanto, se decidió realizar una tienda web desde cero.

Algunas de las alternativas que se plantearon antes de desarrollar la tienda web desde cero fueron Shopify[5], Wix[6], o Squarespace[7]. De modo similar a los SCM, todas son de pago, y ofrecen una libertad creativa y funcional limitada.

En este proyecto se pretende desarrollar tanto un SCM como una tienda web que resuelva los inconvenientes de utilizar aplicaciones de plantillas o SCMs ya desarrollados, ofreciendo una experiencia adaptada a los requisitos planteados por la cliente del proyecto, tanto para el usuario de la tienda web como para el empleado que opere el SCM.

---

[1] www.salesforce.com/es/
[2] https://www.zoho.com/es-xl/
[3] https://www.pipedrive.com/es-es
[4] https://www.odoo.com/es_ES
[5] https://www.shopify.com/es-es
[6] https://es.wix.com/
[7] https://es.squarespace.com/

1.2 Objetivos

El objetivo principal del proyecto es desarrollar dos aplicaciones web: un SCM y una tienda web. La tienda web ofrecerá un acceso público a los productos vendidos por la empresa No.Studio, permitiendo también generar un carrito de compra con los productos seleccionados y posteriormente realizar el pago. El SCM debe permitir realizar todas las operaciones necesarias por parte de la empresa para realizar su negocio. Esto implica poder gestionar los productos que se muestran en la web, la lógica asociada a los pedidos realizados desde la tienda web, la información sobre los proveedores de materiales y sus pedidos asociados, y la información sobre los administradores del sistema y los usuarios de la tienda web.

Para definir el alcance del proyecto he tenido que identificar, durante el desarrollo, una serie de objetivos a alcanzar:

- **Definir los requisitos de la cliente del proyecto, a través de reuniones, e investigar sobre las tecnologías necesarias para poder implementarlos:** el proyecto nace por la necesidad de la cliente de transformar su negocio de manera digital. Es necesario planear y realizar varias reuniones, donde se establecen las distintas entidades y los datos de cada una, además de la funcionalidad requerida por las aplicaciones. Posteriormente, cuando estuviesen los requisitos definidos, es necesario realizar un trabajo de investigación sobre las mejores herramientas para realizar estas dos aplicaciones web.

- **Diseñar un modelo de datos, basado en los requisitos de la cliente, que permita mantener la información necesaria para las dos aplicaciones:** Es necesario diseñar un modelo de datos que permita mantener toda la información necesaria, teniendo en cuenta que las dos aplicaciones se comunican entre sí a través de la base de datos. Esto también requiere de la presencia, mediante reuniones, de la cliente del proyecto, para tener claro qué datos es necesario mantener en el sistema.

- **Desarrollar el SCM:** Es necesario crear una aplicación web que permita gestionar todas las tareas del *back office* de la empresa. Esto implica que la aplicación debe permitir realizar las siguientes funcionalidades:

    - crear, modificar y eliminar los productos de la tienda, y comprobar y modificar los estados de los pedidos de productos realizados desde la tienda web;
    - crear, modificar y eliminar proveedores de materiales para la tienda, y listar y modificar los estados de los pedidos a proveedores;
    - crear, modificar y eliminar los administradores del sistema, que son los usuarios permitidos en el SCM.
    - listar los datos de los usuarios registrados en la tienda web.

  Esta aplicación debe ser solo accesible por los administradores de la aplicación, es decir, los empleados de la empresa asignados a gestionar las tareas de gestión de la cadena de suministro.

- **Desarrollar una tienda web:** Similarmente al objetivo anterior, es necesario crear otra aplicación web que permita realizar las tareas del *front office* de la empresa. Las funcionalidades que deben ser posibles desde esta aplicación son las siguientes:
    - crear un usuario, para poder realizar pedidos de productos y mantener un historial de los pedidos realizados previamente;
    - permitir ver los productos de la tienda, para generar carritos de la compra con ellos, eligiendo la talla y cantidad de cada productos requerida;
    - permitir crear pedidos de los productos del carrito de la compra, realizando el proceso de pago, y guardandolos en la base de datos para recogerlos en el SCM.

  Esta aplicación debe ser accesible al público, siendo una representación digital de la tienda.

1.3 Plan de trabajo

Una vez definidos los objetivos del proyecto, se planteó el plan de trabajo en distintas fases, descritas en cada una de las secciones siguientes.

1.3.1 Definición de los requisitos y del modelo

Durante esta primera fase del proyecto, se realizaron las reuniones necesarias para definir los requisitos propuestos por la cliente del proyecto, y se definió el esquema de datos a manejar entre las dos aplicaciones.

Dado que durante esta fase todavía compaginaba la realización de este proyecto con algunas asignaturas del grado, y la cliente del proyecto trabajaba, estimamos que esta tarea se realizaría en un máximo de tres semanas.

1.3.2 Investigación y formación

Definidos los requisitos y el modelo de datos para implementar las aplicaciones, la siguiente fase consistió en realizar el trabajo de investigación necesario para decidir cuáles serían las tecnologías más adecuadas para cumplir con las necesidades del proyecto. Esto implicó buscar varias alternativas para cada una de las aplicaciones web, recuperando documentación y ejemplos de cada una de ellas. Cuando se eligieron las herramientas, fue necesario disponer de un periodo de formación sobre estas. La selección de las herramientas analizadas durante esta fase se expone en el capítulo 3.

Estimé que esta fase se realizaría en un máximo de seis semanas, considerando que las herramientas iban a ser diferentes para cada una de las aplicaciones.

1.3.3 Desarrollo de las entidades comunes a las dos aplicaciones

Habiendo realizado la fase previa, ya estaba en disposición de comenzar con el desarrollo de las aplicaciones web. Las dos comparten algunas de las entidades del modelo de datos; se desarrollan las dos aplicaciones en paralelo, realizando primero los requisitos del SCM para una entidad específica, y posteriormente los de la tienda web. Esto permitió comprobar que las dos aplicaciones comparten los datos de manera satisfactoria.

Por este motivo, se estimó que esta fase era la más extensa del proyecto, asignándole una duración máxima de cuatro meses. Está detallada en los capítulos 4 y 5, relativos al SCM y a la tienda web respectivamente.

1.3.4 Desarrollo de las entidades específicas de cada aplicación

El siguiente paso consistió en desarrollar las funcionalidades asociadas a las entidades específicas de cada aplicación web. Estas entidades no se compartían entre aplicaciones. Por tanto se decidió realizar primero las de la tienda web, para poder generar carritos de compra y subsecuentemente generar pedidos, y luego las del SCM.

Se estimó que esta fase tendría una duración máxima de 6 semanas, ya que estas entidades eran más simples que las desarrolladas en la fase anterior. Está descrita también en los capítulos 4 y 5, para el SCM y la tienda web respectivamente.

1.3.5 Corrección de errores y pruebas

Es inevitable que durante el desarrollo de cualquier producto software se generen errores y haya que realizar cambios. Esta fase final, que duraría 4 semanas, se reservó para comprobar que las dos aplicaciones funcionaban correctamente antes de la entrega.

2. Introduction

This project consists of the development of two web applications to digitally transform the business of a clothing store of the company/brand No.Studio.

The first consists of a web store, a digital portal to the store to browse through its products and enter them in shopping carts, and then proceed with the payment process and generate an order. The orders are collected in the second web application, a SCM (Supply Chain Management) software that allows the company's supply chain management operations to be carried out from the cloud. This program allows users to generate and update the products to show in the web store, allowing also to specify and modify the inventory of each one of them. The SCM also maintains a history of the orders placed by the store's customers and allows to modify the status of the order, i.e. the stage of the order in the shipment process. It also enables to keep a history of the store's suppliers, other stores that offer materials needed to produce the products to be sold, and maintain a history of orders to suppliers, also permitting the management of the status of these orders.

This chapter presents the motivation that led me to decide to develop this project, which consists of developing two web applications to digitally transform the business of a clothing store, including the reasons why I decided to develop a customized SCM to manage the supply management tasks of the company No.Studio. It also describes the antecedents of similar applications (both for the SCM and the web store), the objectives set at the beginning of the project, and the work plan that was followed.

2.1 Antecedents and motivation

There are several tools already developed that fulfill the supply management functionality required by the SCM requested by the project client, and, although many of them offer interesting features and functionalities, most of them are too expensive for the current business, and those that are not, do not allow us to meet the requirements proposed by the project’s client.

The fact that the most powerful SCMs are not fee-based was a problem for the project client: the company is very young, and currently has limited reach and revenue. In addition, much of the functionality offered by these high-cost SCMs was not necessary for the proper functioning of the store's website.

Furthermore, these tools usually require prior training for their correct use. For these reasons I decided, together with the project's client, to develop a SCM that would meet the needs of the latter, and with the possibility of implementing new functionalities in case they were necessary.

As alternatives, some already developed SCMs such as Salesforce, Zoho, Pipedrive or Odoo were proposed. All these applications are either paid, or offer a limited free plan, in addition to requiring a previous training period to be able to use them.

Regarding the e-commerce web store, there are also tools to generate e-commerce type web pages without the need to develop it from scratch, but the disadvantages are similar: most of them are paid, and also present a limited degree of creative and functional freedom, since they are based on templates. Therefore, it was decided to build an e-commerce web store from scratch.

Some of the alternatives that were considered before developing the e-commerce web store from scratch were Shopify, Wix, or Squarespace. Similar to SCMs, all these alternatives are paid, and offer limited creative and functional freedom.

This project aims to develop both a SCM and a web store that solves the drawbacks of using template applications or already developed SCMs, offering an experience tailored to the requirements set by the client of the project, both for the user of the e-commerce web store and for the employee operating the SCM.

2.2 Objectives

The main objective of the project is to develop two web applications: a SCM and an e-commerce web store. The web store will offer public access to the products sold by the company No.Studio, also allowing users to generate shopping carts with the selected products and then proceed with the payment.

The SCM must be able to perform all the operations necessary for the company to conduct its business. This involves being able to manage information regarding the products displayed on the web, the logic associated with the orders placed from the web store, information about the material suppliers and their associated orders, and information about the system administrators and web store users.

To define the scope of the project I had to identify, during the development, a series of key objectives to be achieved:

- **Define the client's requirements for the project, through meetings, and investigate the technologies needed to implement them:** the project was born out of the client's need to digitally transform their business. It is necessary to plan and conduct several meetings, where the different application entities and the data of each one are established, in addition to the functionality required by the applications. Subsequently, once the requirements are defined, it is necessary to carry out research work on the best tools to make these two web applications.

- **Design a data model, based on the client's requirements, that allows maintaining the necessary information for the two applications:** it is required to design a data model that allows maintaining all the necessary information, taking into account that the two applications communicate with each other using the database. This also requires the presence, through meetings, of the project's client, in order to be aware of what data needs to be maintained in the system.

- **Develop the SCM:** It is necessary to create a web application that allows one to manage all the tasks of the company's back office. This implies that the application must allow to perform the following functionalities:
    - create, modify and delete products from the store, and view and modify the status of product orders placed from the web store;
    - create, modify and delete suppliers of materials for the store, and list and modify the status of orders to suppliers;
    - create, modify and delete system administrators, which are the users allowed access to the SCM;
    - list the data of users registered in the web store.

This application should only be accessible by application administrators, i.e. company employees assigned to manage supply chain management tasks.

- **Develop an e-commerce web store:** Similar to the previous objective, it is necessary to create another web application to perform the company's front office tasks. The functionalities that should be possible from this application are the following:
    - create a user, to be able to place product orders, and keep a history of previously placed orders;
    - allow to view the products in the store, in order to generate a shopping cart with them, choosing the size and quantity of each product required;
    - allow to create orders for the products in the shopping cart, to proceed with the payment process, and to save them in the database to be collected in the SCM.

    This application has to be accessible to the public, as it is a digital representation of the store.

2.3 Work plan

Once the project key objectives were defined, the work plan was divided into different phases, described in each of the following sections.

2.3.1 Definition of requirements and the model

During this first phase of the project, meetings were held to define the requirements proposed by the project client, and the data schema to be handled between the two applications was defined.

Given that during this phase I was still finishing some subjects of the Degree, and the client of the project was currently working, we estimated that this task would be completed in a maximum of three weeks.

2.3.2 Research and training

Once the requirements and the data model to implement the applications were defined, the next phase consisted of carrying out the necessary research work to define the most appropriate technologies to meet the needs of the project. This involved searching for several alternatives for each of the web applications, retrieving documentation and examples of each of them. When the tools were chosen, it was necessary to have a training period on them. The selection of the analyzed tools during this phase is presented in chapter 3.

I estimated that this phase would take a maximum of six weeks, considering that the tools would be different for each of the applications.

2.3.3 Development of common entities for the two applications

Having completed the previous phase, I was ready to start with the development of the web applications. The two share some of the entities of the data model; both applications are developed in parallel, first realizing the requirements of the SCM for a specific entity, and then those of the web store. This made it possible to verify that the two applications share data successfully.

For this reason, this phase was considered to be the most extensive phase of the project, with a maximum duration of four months. It is detailed in chapters 4 and 5, regarding the SCM and the web store, respectively.

2.3.4 Development of specific entities for each application

The next step was to develop the functionalities associated with the specific entities of each web application. These entities were not shared between applications. Therefore, it was decided to first develop those of the web store, in order to generate shopping carts and subsequently generate orders, and then those of the SCM.

It was estimated that this phase would last a maximum of 6 weeks, since these entities were simpler than those developed in the previous phase. It is also described in chapters 4 and 5, for the SCM and the web store respectively.

2.3.5 Bug fixes and testing

Inevitably, during the development of any software product, errors will occur and changes will have to be made. This final phase, which would last 4 weeks, was reserved for testing that the two applications were working properly before delivery.

3. Selección de herramientas y tecnologías

Antes de comenzar con el desarrollo de la aplicación, realicé un estudio de las herramientas y tecnologías más apropiadas para implementar los casos de uso establecidos por la cliente del proyecto. Para ello tuve en cuenta tanto los criterios de cuánto de adecuadas eran estas tecnologías para los objetivos definidos, como la experiencia personal previa con cada una ellas, y, en el caso de no tener experiencia previa, la documentación y cursos existentes.

3.1 Lenguajes y frameworks

En este apartado, muestro tanto los lenguajes y *frameworks* utilizados para el proyecto, como los planteados pero no elegidos finalmente.

3.1.1 HTML y CSS [1]

HTML es un lenguaje utilizado en aplicaciones web para definir la estructura y contenidos de las interfaces de usuario, mientras que CSS es un lenguaje de diseño gráfico que, junto con HTML, permite describir el aspecto y formato de una página web.

He utilizado estos dos lenguajes para definir la interfaz gráfica de las dos aplicaciones, conjuntamente con JavaScript:

3.1.1 JavaScript [2]

Javascript es un lenguaje de programación orientado a objetos, utilizado tanto en el lado del cliente para manejar la lógica de las interfaces de usuario, como en el lado del servidor para recibir las solicitudes de la aplicación, realizar la lógica correspondiente y devolver la respuesta a la interfaz. Está diseñado con una sintaxis similar a Java, adoptando nombres y convenciones de este, y es el único lenguaje de programación que los navegadores web entienden de forma nativa: las demás alternativas se traducen a JavaScript previamente a la ejecución.

He decidido utilizar JavaScript para el lado del cliente y del servidor de las dos aplicaciones web, dada su importancia en el mundo del desarrollo web[8], como por tener experiencia previa con este lenguaje, habiendo usado JavaScript previamente en asignaturas del grado.

3.1.2 Node.js [3]

Node.js es un entorno de ejecución de JavaScript orientado a eventos asíncronos y con un modelo de operaciones de entrada y salida sin bloqueos. Se basa en estas dos características para conseguir ser un entorno eficiente y altamente escalable.

Elegí este entorno de ejecución tanto por la experiencia previa que tenía con la herramienta, como por ser, similarmente a JavaScript, una de las tecnologías web más utilizadas actualmente. Ha sido el entorno de ejecución tanto para el SCM como para la WEB.

3.1.3 NPM

Node Package Manager, conocido como NPM, es un gestor de paquetes preinstalado en Node. Permite instalar o desinstalar paquetes Javascript desde su biblioteca de paquetes. Esto permite añadir o modificar las funcionalidades ofrecidas por Node, reduciendo así el proceso de importación de paquetes, y permitiendo utilizar código generado por terceros. Estos paquetes varían en características, pudiendo ser desde componentes para la interfaz de usuario, a adaptadores para otros paquetes, o hasta conjuntos de componentes en sí, como son los frameworks. Es accesible mediante la línea de comandos.

He utilizado NPM para descargar y mantener todas las dependencias de los frameworks utilizados, descritos posteriormente, y otros paquetes adicionales. De estos últimos, podemos por ejemplo destacar Nodemon, herramienta que permite reiniciar automáticamente la aplicación de node cuando se detectan cambios en el directorio del proyecto, lo cual reduce el tiempo de espera a la hora de ver cambios en las aplicaciones, o GoogleApis, paquete que

[8] "*2022 marks JavaScript's tenth year in a row as the most commonly used programming language.*" https://survey.stackoverflow.co/2022/

permite realizar una conexión autorizada con los servidores de Google para utilizar sus distintas herramientas web.

3.1.4 Express.js [3]

Express es un framework web escrito en Javascript y alojado dentro del entorno de ejecución de Node.js. Es el más popular de Node.js[9], y es la dependencia de un gran número de *frameworks* web de Node populares, como Sails.js. Pertenece a la pila MERN como *framework back-end*, junto con MongoDB, React y Node.js, todas las cuales se utilizan para desarrollar la tienda web de este proyecto. Express es no dogmático, también conocido como transigente. Esto implica una gran libertad a la hora de utilizar Express.js, ya que permite estructurar la aplicación sin un esqueleto subyacente, o insertar distintas funciones que se pueden ejecutar dentro de la cadena de manejo de la petición, conocidas como *middlewares*.

Decidí utilizar Express.js para desarrollar el *back-end* de la página web de la tienda, por su gran difusión en el ámbito profesional y la experiencia previa con la herramienta, ya que la he utilizado extensamente durante el grado.

3.1.5 React [4]

React es una biblioteca JavaScript para crear interfaces de usuario. Diseñada con el objetivo de facilitar el desarrollo de aplicaciones en una sola página (SPA), se comporta como la Vista en el patrón Modelo-Vista-Controlador (MVC). Desarrollado por Facebook y mantenido por la comunidad de software libre, es uno de los frameworks de interfaces de usuario web más utilizados en el mundo, siendo la segunda tecnología web más utilizada tanto por desarrolladores profesionales como por no profesionales, según StackOverflow[10].

React permite diseñar vistas simples para cada estado de la aplicación, ocupándose de actualizar y renderizar de manera eficiente los componentes correctos cuando el estado cambia. Permite crear componentes modulares, cada

[9] "*Express es el framework web más popular de Node, …*" https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

[10] "*Node.js and React.js are the two most common web technologies used by Professional Developers and those learning to code*.", https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe

uno con su propio estado, para acabar formando una interfaz de usuario compleja.

Decidí utilizar esta librería para desarrollar la interfaz de usuario de la aplicación web. He aprendido a utilizarla mediante la documentación oficial, alojada en su página web, y varios cursos digitales, consiguiendo así experiencia con un framework de *front-end* muy utilizado profesionalmente.

3.1.6 Sails.js [5]

Sails.js es un framework de *back-end* para Node.js. Implementado sobre el framework Express, incluye varias capas de abstracción, para facilitar las tareas de desarrollo. Sigue el modelo de MVC. Posee un ORM, métodos para crear una API RESTful y soporte para manejar peticiones en tiempo real gracias a Socket.io. Es especialmente adecuado para realizar aplicaciones que funcionan en tiempo real, como chats, juegos, o aplicaciones colaborativas.

Decidí utilizar este framework para el *back-end* del SCM, ya que, gracias a su ORM integrado, muchas de las funcionalidades requeridas eran sencillas de implementar. Es además un buen punto introductorio a la hora de aprender a utilizar *frameworks* usados profesionalmente.

3.1.7 Bootstrap [6]

Bootstrap es una biblioteca multiplataforma, de código abierto, para diseñar la interfaz de usuario de aplicaciones web. Desarrollada por la empresa Twitter para el diseño de la misma aplicación, y siendo el segundo proyecto más destacado en GitHub, es uno de los *frameworks* más utilizados en el mundo. Esta herramienta permite, a través de una enorme cantidad de plantillas de diseño (con tipografía, formularios, botones, …), realizar interfaces de usuario de manera sencilla y eficiente. Por tanto, este *framework* se ocupa solo del desarrollo *front-end*.

Decidí utilizar este *framework* por varias razones. Se adecua a los requisitos para el desarrollo de la interfaz gráfica del SCM, ya que esta es una aplicación utilizada para gestionar las tareas de inventario y la gestión de pedidos, y no abierta al público. El gran número de plantillas y componentes proporcionados por Bootstrap simplificó la tarea de realizar una interfaz

funcional y adaptable a distintos dispositivos. Además es una herramienta con la que tenía experiencia previa, al haberla utilizado en proyectos de asignaturas del grado.

3.1.8 Axios

Axios es un cliente HTTP basado en promesas para Node.js y el navegador. Presenta una estructura *isomórfica*, es decir, puede ejecutarse en el navegador y Node.js con el mismo código base. Utiliza el módulo nativo *http* de Node.js en el lado del servidor, y la clase *XMLHttpRequests* en el lado del cliente.

He utilizado Axios para gestionar las peticiones HTTP de la página web de la tienda, debido a que la tienda web se comunica con el servidor a través de una API REST.

3.1.9 Redux [7]

Redux es una librería JavaScript para el manejo del estado de las aplicaciones, usada normalmente junto a React, para construir interfaces de usuario. Gracias a la definición de estados para cada tipo de funcionalidad de la aplicación, permite tener un claro entendimiento de cómo afectan al estado las acciones producidas por el usuario.

El estado de toda la aplicación está almacenado en una variable llamada `store`. Este estado sólo es modificable a través de un *state reducer*, que es una función que, a partir del estado actual y una acción del usuario, devuelve un nuevo estado. Este último será el estado de la aplicación tras la ejecución de dicha acción.

He decidido utilizar Redux en este proyecto con el fin de aprender a utilizar una tecnología muy usada en ámbitos empresariales, sobre todo en aquellos en los que se utiliza también React.

3.2 Gestores de Bases de Datos y herramientas de integración

Para mantener la información necesaria para el funcionamiento de las dos aplicaciones web fue necesaria la administración de una base de datos. A continuación se muestran las distintas tecnologías y herramientas que evalué durante el período de investigación:

3.2.1 MySQL y MongoDB [8]

A la hora de desarrollar un modelo de datos, la cuestión principal es si la aplicación usará una base de datos relacional o no relacional. Primero describiré las dos tecnologías valoradas para mantener los datos de las aplicaciones.

MySQL es un gestor de bases de datos relacionales, desarrollado por Oracle Corporation. Permite almacenar los datos en tablas, que representan las distintas entidades del modelo de datos de la aplicación. Estas tablas se enlazan entre sí mediante reglas, conocidas como relaciones, utilizadas para encontrar información relevante a distintas entidades. Se accede a sus funciones mediante sentencias SQL (*Structured Query Language*).

Por su parte, MongoDB es un sistema de base de datos NoSQL, orientado a documentos y de código abierto. Basado en un modelo de documentos, MongoDB almacena datos en documentos flexibles, con formato BSON, similares a JSON. Al estar basado en documentos flexibles, y no en tablas con una definición estricta, los campos pueden variar entre documentos y el esquema de datos puede cambiarse con el tiempo.

A la hora de elegir entre una base de datos relacional (SQL) o no relacional (NoSQL), muchas de las diferencias principales no eran relevantes para el alcance del proyecto:

- Las relacionales presentan una escalabilidad vertical, lo que implica una mejora del equipo donde esté alojado el servidor, mientras las no relacionales presentan una escalabilidad horizontal, donde se añaden servidores en lugar de mejorarlos. Para este proyecto esto no era un problema real, dado que la información a mantener, según la cliente, iba a

ser limitada, al tener la empresa una producción y un nivel de difusión modesto en este momento.

- En términos de comunidad y documentación, los dos gestores de bases de datos dos son muy similares, ya que, aunque las bases de datos relacionales han sido usadas durante más tiempo, las no relacionales son cada día más populares, y en los casos de MySQL y MongoDB, las dos presentan una documentación extensa y precisa.
- Por último, y el motivo por el que he terminado eligiendo las bases de datos no relacionales, es por la flexibilidad a la hora de representar los datos. Las bases de datos relacionales requieren una definición precisa del esquema de datos a mantener, mientras que las no relacionales almacenan datos a través de documentos, y no precisan de un esquema que los defina. Para este proyecto, me han parecido mucho más adecuadas las bases de datos no relacionales pues al ser para una empresa joven, de relativa nueva creación y por ello todavía cambiante, estas permitirán agilizar el proceso de desarrollo, y cambiar los modelos de forma flexible adecuándose mejor a los cambios y nuevas necesidades en los requisitos de la cliente.

También he valorado en esta elección que, habiendo realizado durante el grado muchos proyectos con bases de datos relacionales, y menos con no relacionales era esta una oportunidad de profundizar en esta herramienta. Estas razones me han llevado a decidirme por MongoDB, obteniendo así más experiencia con otra tecnología.

3.2.3 Mongoose [9]

Mongoose es una librería para Node.js que permite escribir consultas para una base de datos de MongoDB, y dispone de características como validaciones, construcción de consultas, *middlewares*, conversión de tipos y algunas otras, que enriquecen la funcionalidad de la base de datos.

Mongoose es un ODM (Object Document Mapper). Como he explicado antes, MongoDB mantiene los datos en documentos y colecciones, en formato BSON. Mongoose convierte este formato en un objeto JavaScript, permitiendo trabajar con los datos de una manera más simple y ordenada. Ofrece también consultas estáticas para realizar todas las operaciones CRUD, desde las más

simples como `find()` o `findOne()`, que buscan los documentos un documento de la colección, respectivamente, hasta algunas algo más complejas como `findOneAndReplace()`, que busca un documento de la colección y lo reemplaza por el proporcionado.

Gracias a estas características, es el que he decidido utilizar para el desarrollo de la Web de la tienda, por considerarlo el más adecuado para los requisitos planteados.

3.2.4 Waterline

Waterline es el ORM/ODM (Object Relational Mapper y Object Document Mapper, respectivamente) de Sails.js, una herramienta independiente del almacén de datos que simplifica drásticamente la interacción con una o más bases de datos. Proporciona una capa de abstracción para la base de datos subyacente, permitiendo consultar y manipular los datos sin necesidad de escribir código de integración específico del sistema de base de datos elegido.

Waterline permite realizar todas las operaciones CRUD con una misma sintaxis, independientemente de la base de datos que estemos utilizando. Permite incluso realizar consultas cruzadas, aunque sea con información alojada en distintas bases de datos. Esto implica poder cambiar de base de datos durante el desarrollo, minimizando la cantidad de cambios en el código. También ofrece la posibilidad de acceder a las funcionalidades de bajo nivel específicas a cada base de datos, con métodos como `.query()` o `.native()`.

En mi caso he utilizado Waterline con el framework Sails.js para el desarrollo del SCM, utilizando el adaptador Sails-Mongo:

3.2.5 Sails-mongo

Sails-mongo es un adaptador de Waterline mantenido por el equipo de Sails.js. Su función es la de proporcionar un acceso robusto y de fácil uso a MongoDB desde Sails.js y Waterline. Como he descrito antes para Waterline, implementa una capa de abstracción para la base de datos subyacente, en mi caso MongoDB, para realizar operaciones CRUD e incluso consultas cruzadas de una manera simple y ordenada.

Durante el desarrollo del proyecto me encontré con la necesidad de decidir qué tipo de base de datos sería el más adecuado según los requisitos, como he mencionado anteriormente. Gracias a Waterline, y a sus adaptadores para cada tipo de base de datos, pude realizar cambios y pruebas teniendo que realizar muy pocos cambios en el código.

3.2.6 Skipper

Skipper es un middleware, instalado por defecto en Sails.js, para convertir los archivos enviados desde la interfaz de usuario en un objeto JavaScript, realizando así la función de un *body parser*. Permite acceder al archivo desde la petición `req`, permitiendo cargar los archivos con una sola llamada al método `upload`:

```
req.file('avatar').upload(function (err, uploadedFiles){
  if (err) return res.send(500, err);
  return res.send(200, uploadedFiles);
});
```

Mediante `req.file` accedemos al componente del formulario (con nombre `avatar`) y lo cargamos en la base de datos elegida mediante el método `upload()`. Actualmente, skipper permite cargar los archivos en el sistema de ficheros del servidor, pero tiene una gran cantidad de adaptadores para poder cargarlos a bases de datos. En mi caso, utilizando MongoDB, probé a utilizar Skipper-gridfs:

3.2.6 Skipper-gridfs

Skipper-gridfs es un adaptador para Skipper que permite subir archivos directamente a MongoDB, utilizando GridFS. GridFS es una especificación de almacenamiento para documentos BSON que exceden el límite de 16MB establecido por MongoDB para cada archivo. GridFS se ocupa de dividir los ficheros en partes, o *chunks*, y almacena cada parte en un documento distinto. A la hora de recuperar los archivos, GridFS también se ocupa de recoger las partes y ordenarlas.

En este proyecto era necesario poder mantener las imágenes en la nube, ya que las dos aplicaciones web comparten esos archivos. GridFS se ocuparía de fragmentar los archivos a la hora de cargarlos desde el SCM, y guardarlos en la base de datos. Además, se ocuparía de descargar las partes de cada archivo tanto para el SCM como la Web, para poder representar las imágenes.

El inconveniente de este adaptador es que no está pensado para trabajar con muchos archivos pequeños, sino más bien para hacerlo con pocos archivos de tamaño mayor que 16Mb. Por tanto, para este proyecto, a la hora de cargar las imágenes en la interfaz, el tiempo de carga era muy elevado, influyendo negativamente en la experiencia del usuario. Decidí entonces a probar a utilizar un sistema de ficheros en la nube, como Google Drive.

3.2.7 Google Drive

Google Drive es un servicio de alojamiento y sincronización de archivos desarrollado por Google. Permite a sus usuarios almacenar archivos en la nube, sincronizar archivos entre dispositivos y compartir archivos. Ofrece 15GB de espacio gratuito para almacenar archivos, ampliables mediante diferentes planes de pago.

Google Drive ofrece la posibilidad de manejar los archivos desde Node.js mediante una librería, `googleapis/drive`, ofreciendo métodos para crear, cargar o descargar archivos. Además, a la hora de cargarlos, devuelve un objeto con la información asociada al archivo subido. Esto permite simplificar el almacenamiento de imágenes en la base de datos, ya que solo ha de almacenarse la URL donde está alojada cada imágen, haciendo así Google Drive también el trabajo de un servicio de *hosting*.

Google Drive no está pensado para ser usado como servicio de almacenamiento de ficheros públicos, o de una web; más bien para el uso personal de cada usuario. Aun así, para aplicaciones con alcance limitado, como es en el caso de este proyecto, realiza el trabajo de una manera satisfactoria, y, considerando que tiene hasta 15GB de almacenamiento gratuito, utilizable hasta que las aplicaciones tuvieran un alcance más elevado. Además, Google ofrece sistemas para migrar desde Google Drive hasta servicios de almacenamiento de archivos públicos, como Google Cloud, desarrollados específicamente para este

uso. En el caso de que las aplicaciones web tuvieran un alcance mayor, realizar este cambio de tecnología no sería costoso; pero como Google Cloud no ofrece una tarifa gratuita, para el desarrollo del proyecto he decidido utilizar Google Drive.

3.3 Tecnologías adicionales

Para el correcto desarrollo del proyecto, han sido necesarias herramientas de control de versiones, administración de proyectos y de documentación.

3.3.1 Visual Studio Code [11]

Visual Studio Code es un editor de código fuente desarrollado por Microsoft. Basado en Electron, un *framework* que se utiliza para implementar por ejemplo Node.js, incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, y mucho más.

He utilizado este editor por su gran uso en ámbitos empresariales y por tener experiencia previa, al haberlo usado en proyectos durante el grado.

3.3.2 Git [20] [10]

Git es un sistema de control de versiones distribuido, o DVCS, que permite coordinar el trabajo en equipos de varios desarrolladores, manteniendo el historial de versiones del código. Tiene una arquitectura distribuida: el historial de versiones no se almacena en un único espacio, sino que la copia del código de cada integrante del repositorio es también un repositorio que mantiene el historial completo de todos los cambios. Más allá de ser distribuido, Git basa su diseño en el rendimiento, la seguridad y la flexibilidad.

En el caso de mi proyecto, Git no fue usado como herramienta para mejorar la coordinación de trabajo, al haber realizado el proyecto en solitario. Aun así, la he utilizado como herramienta de control de versiones, para poder desarrollar las aplicaciones desde distintas máquinas.

3.3.3 GitHub

GitHub es una plataforma que aloja en la nube el sistema de control de versiones Git. Permite acceder a las funcionalidades de Git desde un portal web, pudiendo crear repositorios y mantener un control de versiones desde su plataforma web. Esto hace que sea más fácil usar Git como herramienta de control de versiones y colaboración.

Git es un sistema distribuido, en el que cada usuario tiene una copia del repositorio entero, incluyendo todas las versiones. Cuando un integrante del proyecto realiza una modificación al código fuente y lo sincroniza con el repositorio alojado en GitHub, se genera una nueva versión del proyecto. La sincronización se realiza mediante las operaciones *COMMIT* y *PUSH*, donde la primera añade una versión nueva al repositorio local, mientras que la segunda envía las versiones al repositorio remoto, compartido por los demás integrantes del equipo.

En el caso de mi proyecto, utilicé GitHub como portal a Git, ya que la capa de abstracción que ofrece permite realizar las operaciones de control de versiones de una manera más simple y directa. Cada una de las aplicaciones web realizadas tiene un repositorio diferente, con documentación asociada a cada una de ellas.

3.3.4 Notion

Notion es un software gratuito de gestión de proyectos y documentación. Está diseñado para ayudar a los miembros de una empresa u organización a coordinar plazos, establecer objetivos y documentar tareas en aras de la eficiencia y productividad.

Para la realización de este proyecto he utilizado Notion tanto para gestionar el proyecto, utilizando un tablero Kanban para dividir y gestionar las distintas tareas, así como para recopilar y mantener documentación respecto a las herramientas utilizadas, de las reuniones con la cliente del proyecto o de las reuniones con el tutor del trabajo. Notion me ha permitido mantener toda esta información en un mismo entorno, reduciendo así el número de herramientas necesarias para gestionar este ámbito del trabajo.

3.3.5 Google Drive

Como ya he explicado anteriormente, Google Drive es un servicio de alojamiento y sincronización de archivos desarrollado por Google. No solo lo he utilizado como servicio de alojamiento de las imágenes; ha servido también como herramienta de comunicación y de transferencia de archivos con el tutor del trabajo, en el repositorio asociado a mi cuenta de la Universidad Complutense de Madrid.

3.3.6 Google Docs

Google Docs es un procesador de texto en línea integrado con Google Drive. Permite crear y editar documentos de texto desde el repositorio de Google Drive. He utilizado esta herramienta para desarrollar esta memoria, y poder compartirla con el tutor del trabajo para su corrección previa.

4. Descripción del SCM

La funcionalidad del SCM consiste en gestionar todas las tareas de gestión de suministros asociadas al modelo de negocio de la empresa No.Studio. Esto implica poder añadir productos, modificarlos, tanto para actualizar datos como para gestionar el stock de cada uno de ellos, gestionar los pedidos realizados por los usuarios desde la web, manteniendo un historial de ellos y pudiendo ordenarlos por estado, y gestionar también la lista de proveedores y los pedidos a proveedores, manteniendo también un historial de ellos. Debe también mantener un historial de los usuarios de la tienda web.

4.1 Modelo de dominio

El SCM maneja una serie de entidades, necesarias para el correcto funcionamiento de la aplicación. Algunas de ellas son compartidas por las dos aplicaciones web, mientras que otras son específicas de una de ellas. Todos las instancias de las entidades tienen asociado un identificador único generado por la base de datos, mediante el campo `_id`, y la información sobre la fecha de creación y de actualización, mediante los campos `createdAt` y `updatedAt`. El borrado de la aplicación es lógico: esto implica que no se borran físicamente los datos de la base de datos, sino que se añade a las entidades un campo de tipo booleano `active`, que determina si el elemento ha sido eliminado o no (excepto en las entidades de pedidos, véase 4.1.2 y 4.1.5).

4.1.1 Administradores (`admin`)

Los administradores son los empleados de la tienda, y los únicos usuarios capaces de acceder al SCM. Se les representa por un nombre, `name`; un correo, `emailAddress`, que necesariamente tiene que ser único; y una contraseña, `password`, siendo estos últimos dos campos los utilizados para iniciar sesión en la aplicación. También contienen un campo para determinar si el administrador en cuestión es el de la cliente del proyecto, `root`.

Un administrador sólo puede ser creado por otro, ya que la funcionalidad de crear administradores sólo está habilitada a la hora de iniciar sesión. El

administrador maestro, determinado por el campo `root`, en este caso el de la cliente del proyecto, se genera en fase de desarrollo, con los datos proporcionados por esta, y no se permite que se borre. Sí se permite tanto crear nuevos administradores, para el caso en el que se amplíe la plantilla de empleados, modificarlos, y eliminarlos (borrado lógico), en caso de despido o traspaso a otras áreas de la empresa.

Esta entidad se contempla solo desde el SCM.

4.1.2 Pedidos (`orders`)

La entidad Pedidos se ocupa de gestionar los pedidos realizados por los usuarios de la web de la tienda. Los pedidos son representados por varios campos. Cada pedido contiene una lista de productos, llamada `orderItems`, que recoge los productos asociados al pedido, cada uno de ellos asociado con la cantidad de unidades a comprar, y la talla seleccionada. Contiene también el identificador del usuario que ha realizado el pedido con un campo `idUser`, y la dirección de envío, `shippingAdress`, que puede o no ser la misma especificada por el usuario a la hora de registrarse. Cada pedido recoge también el precio total del pedido, calculando también el IVA correspondiente (en el caso de artículos de ropa, el 21%), mediante los campos `itemsPrice`, `taxPrice` y `totalPrice`, el método de pago seleccionado, `paymentMethod`, y el estado en el que se encuentra actualmente, mediante el campo `status`.

El estado de un pedido ha sido definido como los distintos pasos por los que puede pasar un pedido, siguiendo los requisitos definidos por la cliente. Los posibles estados son los siguientes:

- Pendiente de envío (`toSend`): el pedido ha sido pagado, es decir, acaba de ser generado;
- Enviado (`sent`): el pedido ha sido enviado desde el almacén hacia la dirección de envío proporcionada por el cliente de la tienda;
- Recibido (`received`): el pedido ha sido enviado satisfactoriamente y recibido por el cliente de la tienda;
- Devuelto (`returned`): el pedido ha sido devuelto por el cliente de la tienda, siguiendo la política de devoluciones establecida por la empresa;

- Cancelado (`cancelled`): el pedido ha sido cancelado por el cliente de la tienda. Este último estado contempla el caso en el que el cliente de la tienda decida, en última instancia, cancelar el pedido (con posterioridad a ser pagado).

Las únicas operaciones que se pueden realizar desde el SCM con relación a los pedidos es la de listarlos, y cambiar su estado. Los pedidos se generan desde la tienda web, y no se contempla la posibilidad de eliminarlos, ya que la cliente del proyecto requiere mantener un historial de todos los pedidos.

4.1.3 Productos (`products`)

La entidad Productos se ocupa de gestionar los datos y operaciones asociadas a los productos de la tienda. Está compuesta por varios campos. Contiene un nombre, `name`, una descripción, `description`, el precio por unidad, `price`, la categoría del producto, `category`, un campo `id_images`, la colección de los identificadores de las imágenes asociadas al producto, y un campo `productSize`, que representa una colección de las tallas y el *stock* asociado a cada una de ellas.

La categoría de un producto no es más que una cadena de texto, que sirve para buscar productos en la web de la tienda por su categoría.

Desde el SCM se procesa toda la información y se implementa lógica asociada a esta entidad. Se permite crear productos, listarlos todos o listarlos bajo filtros y/o cadenas de búsqueda. Se permite también modificarlos, tanto para cambiar alguno de los datos informativos, como para cambiar el stock asociado a cada talla. Por último se permite también eliminarlos, como hemos dicho anteriormente, de manera lógica (es decir, se cambia el campo `active` a falso).

4.1.4 Proveedores (`suppliers`)

La entidad Proveedores gestiona toda la información referente a los proveedores de artículos y materiales. La empresa vende tanto productos realizados a mano por la propia empresa, como productos ya terminados que la

tienda revende, sea *merchandising* u otros. Esta entidad gestiona toda la información sobre los Proveedores de estos artículos.

La entidad Proveedores está compuesta por varios campos. Contiene el nombre del proveedor, con el campo `name`; el correo electrónico asociado al proveedor, con el campo `email`; la dirección de la tienda o empresa del proveedor, `address`; una breve descripción del proveedor mediante el campo `description`; y un campo enumerado que contiene la categoría del proveedor, `category`.

Esta entidad se contempla solo desde el SCM.

4.1.5 Pedidos a proveedores (`supplyorder`)

La entidad Pedidos a proveedores permite mantener un historial de los pedidos realizados a los proveedores almacenados en el sistema, pudiendo en el historial actualizar el estado del pedido. Sus datos son el id del proveedor, `idSupplier`; los materiales o artículos en el pedido, llamados `items`; y el estado en el que se encuentra el pedido, con el campo `status`.

El estado de un producto se define como las distintas fases por las que puede pasar un pedido, siguiendo los requisitos definidos por la cliente del proyecto. Los posibles estados son los siguientes:

- Pendiente de petición (`toOrder`): el pedido todavía no se ha comunicado al proveedor; este estado existe ya que la cliente del proyecto comunicó como requisito la necesidad de mantener esta información, en el caso de que tenga que realizar un pedido y quiera recordarlo.
- Pedido (`ordered`): el pedido ha sido comunicado al proveedor, y está en proceso de recibirse.
- Recibido (`received`): el pedido ha sido recibido y almacenado en el inventario.
- Cancelado (`cancelled`): el pedido ha sido cancelado antes de su llegada al inventario.

Esta entidad se maneja íntegramente desde el SCM, el cual permite generar nuevos pedidos, listarlos todos, filtrar por su estado, buscar por sus atributos, y modificar el estado de cada pedido. No se contempla la posibilidad de eliminar pedidos a proveedores, ya que la cliente del proyecto requiere como requisito mantener un historial de todos ellos.

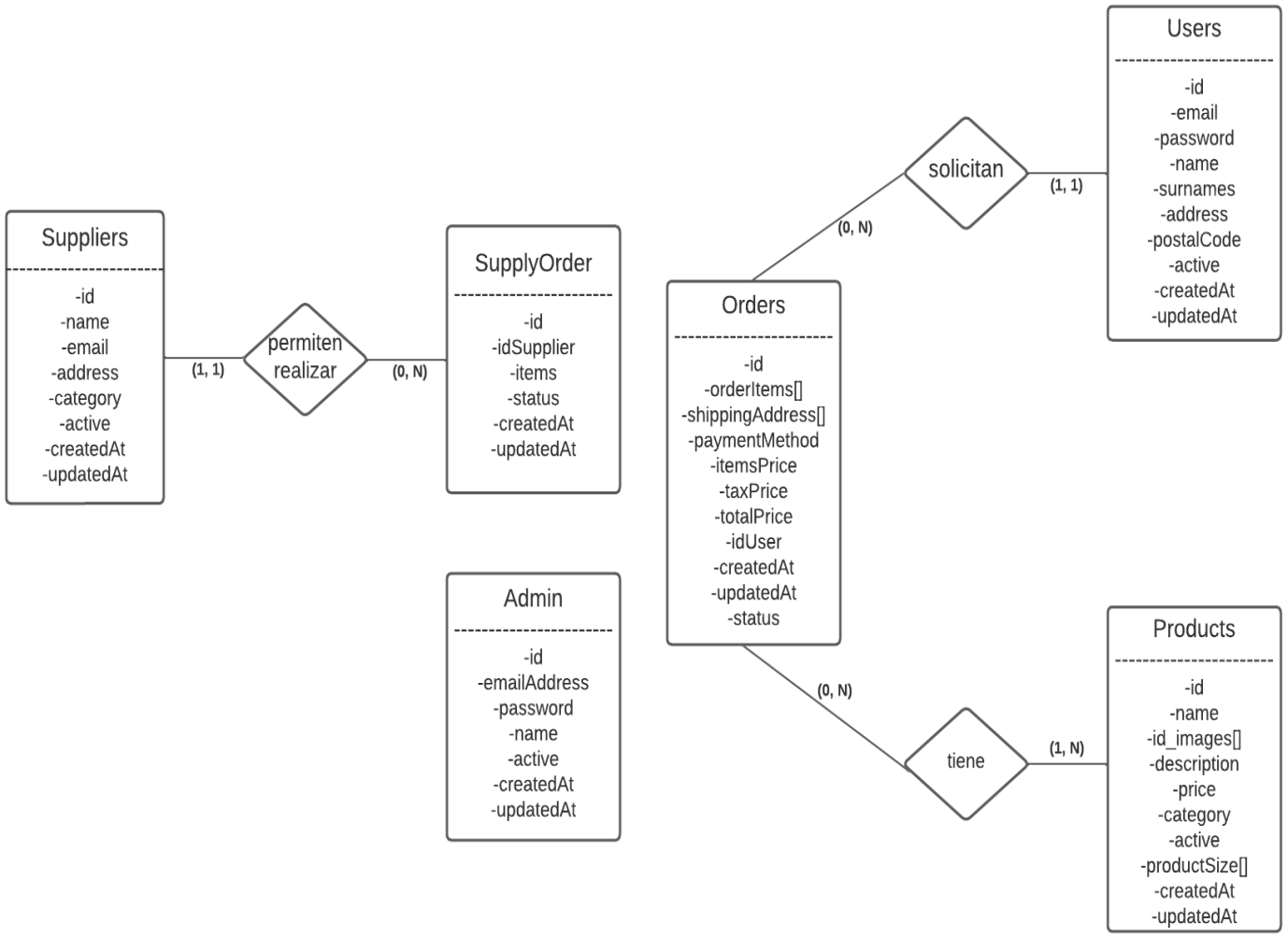
4.1.6 Usuarios (`users`)

Los Usuarios son los clientes de la tienda web, que se registran desde la misma. Sus datos se componen del nombre y apellidos del usuario, `name` y `surnames` respectivamente; su correo electrónico y contraseña, `email` y `password` respectivamente; y la dirección del usuario y el código postal, mediante los campos `address` y `postalCode`.

En el SCM, se permite solamente ver los datos de los usuarios, para poder realizar los envíos de pedidos asociados. Las operaciones de crear y eliminar usuarios se gestionan desde la tienda web.

4.2 Modelo Entidad-Relación del SCM

A continuación muestro el diagrama de Entidad Relación del SCM:



Como se ve en el diagrama, las relaciones entre entidades son las siguientes:

- Proveedores (`suppliers`) tiene una relación uno a muchos con Pedidos a Proveedores (`supplyOrder`).
- Usuarios (`users`) tiene una relación uno a muchos con Pedidos (`orders`)
- Productos (`products`) y Pedidos (`orders`) tienen una relación muchos a muchos entre sí.

4.3 Esquema de datos MongoDB para el SCM

En este capítulo represento el esquema de datos de cada una de las colecciones utilizadas por el SCM. En el caso de que una colección contenga internamente otra colección, se mostrará en otra tabla.

4.3.1 Administradores (`admin`)

Nombre	Tipo	Referencia	Nullable
`_id`	Id		No
`name`	String		No
`emailAddress`	String		No
`password`	String		No
`active`	Boolean		No
`root`	Boolean		No
`createdAt`	Date		No
`updatedAt`	Date		No

Estos datos representan a un administrador del SCM. Se registran utilizando el correo electrónico (`emailAddress`) y la contraseña (`password`). Un administrador con el campo `active` igual a `false` no puede iniciar sesión en la aplicación, y debe pedir a un administrador activo que lo reactive. El campo `root` determina si el administrador es el administrador maestro, y no permite desactivar al administrador si es igual a `true`.

4.3.2 Pedidos (`orders`)

Nombre	Tipo	Referencia	Nullable
`_id`	Id		No
`orderItems`	Document[]		No

`shippingAddress`	Document		No
`paymentMethod`	String		No
`itemsPrice`	Number		No
`taxPrice`	Number		No
`totalPrice`	Number		No
`idUser`	Id	Usuarios (`users`)	No
`status`	String		No
`createdAt`	Date		No
`updatedAt`	Date		No

En este esquema, tenemos anidadas dos colecciones:

- **`orderItems:`**

Nombre	**Tipo**	**Referencia**	**Nullable**
`_id`	Id		No
`name`	String		No
`price`	Number		No
`idProduct`	Id	Productos (`products`)	No
`qty`	Number		No
`size`	String		No

Esta colección representa a un producto específico, con los datos asociados a él;
`_id` es el identificador único del objeto que recoge la información, y `qty` es la cantidad del producto en el pedido. Los demás campos ya han sido descritos

anteriormente. Cabe destacar que realmente es una colección anidada dentro de otra: es decir, hay una colección de productos por cada uno de los pedidos.

- **`shippingAddress`:**

Nombre	Tipo	Referencia	Nullable
`fullName`	String		No
`address`	String		No
`city`	String		No
`postalCode`	String		No
`country`	String		No

Esta colección representa la dirección de envío del pedido. No tienen por qué ser los mismos datos que los del usuario que ha generado el pedido; a la hora de realizarlo, se permite introducir una dirección de envío diferente a la registrada por el usuario. De esta manera, se permite también mandar pedidos a personas no registradas como usuarios, por ejemplo, si se quiere mandar un producto como regalo.

4.3.3 Productos (`products`)

Nombre	Tipo	Referencia	Nullable
`_id`	Id		No
`name`	String		No
`description`	String		No
`price`	Number		No
`category`	String		No
`id_images`	Document		No
`productSize`	Document[]		No

`active`	Boolean		No
`createdAt`	Date		No
`updatedAt`	Date		No

En este esquema, tenemos anidadas dos colecciones:

- `id_images` es una colección de cadenas de texto con el identificador de la imagen en Google Drive.
- `productSize` es una colección que representa las parejas de talla (`size`) y las existencias (`stock`) asociada a cada talla. Su estructura es la siguiente:

Nombre	**Tipo**	**Referencia**	**Nullable**
`size`	String		No
`stock`	Number		No

4.3.4 Proveedores (`suppliers`)

Nombre	**Tipo**	**Referencia**	**Nullable**
`_id`	Id		No
`name`	String		No
`email`	String		No
`address`	String		No
`category`	String		No
`active`	Boolean		No
`createdAt`	Date		No
`updatedAt`	Date		No

Estos datos representan a un Proveedor en el sistema. Un proveedor puede ser eliminado, asignando el campo `active` a `false`. Esto implica que no

serán seleccionables a la hora de generar un pedido a proveedores. Los demás datos han sido previamente explicados en el capítulo 4.1.4.

4.3.5 Pedidos a proveedores (`supplyorder`)

Nombre	**Tipo**	**Referencia**	**Nullable**
`_id`	Id		No
`idSupplier`	Id	Proveedores (`suppliers`)	No
`items`	String		No
`status`	String		No
`createdAt`	Date		No
`updatedAt`	Date		No

Estos son los datos almacenados en la aplicación cuando se genera un nuevo pedido a proveedores. Se mantiene una referencia al Proveedor, mediante `idSupplier`, y una colección, actualmente una cadena de texto, de los materiales y artículos introducidos en el pedido. El campo `status` representa al estado del pedido, modificado manualmente por el administrador desde el SCM, y sus valores son los descritos en el capítulo 4.1.5.

4.3.6 Usuarios (`users`)

Nombre	**Tipo**	**Referencia**	**Nullable**
`_id`	Id		No
`name`	String		No
`surnames`	String		No
`email`	String		No
`password`	String		No

`address`	String		No
`postalCode`	String		No
`active`	Boolean		No
`createdAt`	Date		No
`updatedAt`	Date		No

Los datos del usuario de la tienda web son almacenados cuando el cliente de la tienda se registra en la aplicación. Un cliente puede eliminar su usuario desde la tienda web, cambiando el campo `active` a `false`, y no pudiendo acceder a las funcionalidades reservadas a usuarios registrados hasta que vuelva a activar su usuario. Esto permite, en un futuro, eliminar al usuario de posible contacto desde la tienda.

4.4 Capa de acceso a datos mediante Waterline/ Sails-mongo

Como he descrito anteriormente en el capítulo 3 (Selección de herramientas y tecnologías), el acceso a la base de datos desde el SCM está proporcionado por el ODM Waterline y su adaptador Sails-mongo.

Waterline viene por defecto en cualquier proyecto Sails.js. El adaptador a utilizar con Waterline se asigna en el archivo de configuración `config.datastores`. Aquí se especifica tanto el adaptador a utilizar, como la URL de conexión a la base de datos, de la siguiente manera:

```
default: {
      adapter: 'sails-mongo',
      url:'mongodb+srv://NoStudio:[contraseña]@cluster0.sorbo.mongodb.net/[
      nombreBBDD]?retryWrites=true&w=majority',
         },
```

De esta manera el adaptador sails-mongo se conecta con MongoDB.

Como he explicado anteriormente en la descripción del modelo de dominio (capítulo 4.1), existen campos definidos en todos los esquemas de

documentos simultáneamente, específicamente el identificador de cada documento, y las fechas de creación y de última actualización de todos los documentos, `_id`, `createdAt` y `updatedAt` respectivamente. Estos campos se definen en otro fichero de configuración, `config.models`, de la siguiente manera:

```
attributes: {
    createdAt: { type: 'number', autoCreatedAt: true, },
    updatedAt: { type: 'number', autoUpdatedAt: true, },
    id: { type: 'string', columnName: '_id' },
  };
```

Estos campos se generarán siempre que se cree un nuevo documento, y `updatedAt` se actualizará automáticamente para mantener la fecha de la última actualización del documento.

En el archivo de configuración `config.models` se encuentra también la clave de desencriptación de los campos que se hayan generado con la opción `encrypt: true`. Esta opción se asigna a la hora de definir el campo en el esquema, y encripta la cadena de caracteres introducida. En este archivo se pueden añadir muchas más opciones; para este proyecto solo se han utilizado las nombradas anteriormente.

Para el correcto funcionamiento de Waterline es necesario generar *schemas* (esquemas) de datos, para luego poder realizar operaciones sobre ellos. Estos esquemas no son requeridos por MongoDB, al ser una base de datos no relacional; pero si lo son para utilizar las operaciones de acceso a la base de datos proporcionadas por Waterline. He aquí el esquema de datos de la entidad Pedidos:

```
module.exports = {

  attributes: {
    orderItems: { type: 'json', required: true },
    shippingAdress: { type: 'json', required: true },
    paymentMethod: { type: 'string', required: true },
    itemsPrice: { type: 'number', required: true },
    taxPrice: { type: 'number', required: true },
    totalPrice: { type: 'number', required: true },
```

```
    idUser: { type: 'string', required: true },
    status: {type: 'string', required: true}
  },
};
```

Habiendo definido el esquema, realizar las consultas a la base de datos resulta muy sencillo. He aquí un ejemplo de creación de un producto:

```
    var createProduct = await Products.create({
      name: inputs.name,
      id_images: uploadInfo,
      description: inputs.description,
      price: inputs.price,
      category: inputs.category,
      productSize: productSize,
      active: true,

    }).fetch();
```

Con el método `.fetch()` recuperamos la información generada a la hora de crear el documento; así recuperamos el identificador del documento generado por la base de datos.

He aquí un ejemplo de actualización de un producto:

```
var update = await Products.updateOne({
        id: inputs.id,
      }).set({
        name: inputs.name,
        image: inputs.image,
        description: inputs.description,
        price: inputs.price,
        category: inputs.category,
      });
```

Mediante el método `.updateOne(id)` seleccionamos el documento a actualizar, e introducimos los campos a introducir con el método `.set(data)`.

4.5 Funcionalidad implementada

Sails.js está basado en la arquitectura MVC (Modelo-Vista-Controlador). El flujo de esta arquitectura es el siguiente:

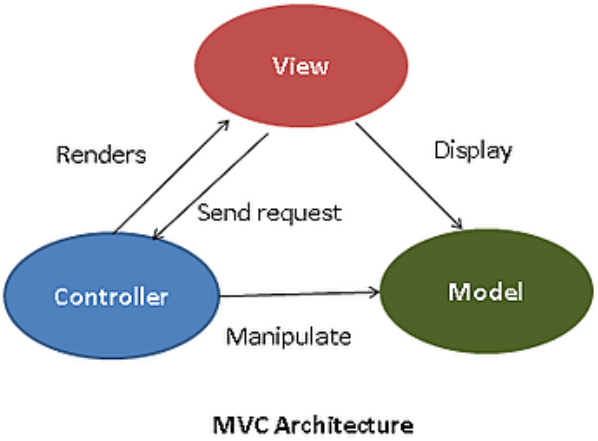


Figura 4.5.1 - Arquitectura MVC

En Sails.js, la lógica de los controladores se recoge en ficheros JavaScript llamados `actions`. Una acción se genera desde el terminal mediante la siguiente instrucción:

```
sails generate action nombreAccion
```

Una acción es un fragmento de la lógica asociada a una entidad. Cada acción realiza el trabajo de una funcionalidad específica. Esto es posible gracias al diseño de estas acciones. En vez de trabajar con la sintaxis *request-response*, Sails.js se comunica mediante `inputs` y `exits`, respectivamente; la lógica se realiza en una función asíncrona `fn`. He aquí un ejemplo de la acción `create-product` (he omitido todos los datos de entrada, `inputs`, menos el de `name`, para reducir el tamaño del código):

```
module.exports = {


  friendlyName: 'Create product',
   description: 'This action creates a new Product entity from the data
given by the administrator. It also authenticates the program to the
Google Drive API, through the credentials.json file, to be able to
upload images to the filesystem, and uploads them.',

  inputs: {
    name: {
      type: 'string',
      required: false,
      description: 'Name of the product',
      maxLength: 100,
    },
  },


  exits: {
    success: {
      responseType: 'view',
      viewTemplatePath: 'pages/products/create-product'
    },
    errorCreate: {
      responseType: 'view',
      viewTemplatePath: 'pages/products/create-products'
    },
  },

  fn: async function (inputs, exits) {

    //lógica de la acción

    if (createProduct) { //true si no ha habido errores.
      return exits.success({ msg: "Producto creado." });
    } else {
      return exits.errorCreate({ msg: "Error al crear el producto" });
    }
}
```

El atributo `inputs` define todos los datos de entrada de la acción. Realiza también el trabajo de validación de datos, devolviendo un error si no recibe los datos requeridos (con `required: true`), y comprobando que el tipo

de dato que se recibe concuerda con la definición del campo en este atributo. No es una copia del esquema definido en los modelos: solo define y valida los campos que llegan desde la llamada a la acción.

El atributo `exits` define las distintas salidas posibles a la hora de ejecutar la lógica, y se definen en el mismo campo. Esto permite mostrar vistas distintas dependiendo del resultado de la lógica. El campo `msg` se manda a la vista para comunicar al usuario si la operación se ha ejecutado correctamente o no.

La función `fn` contiene toda la lógica de la acción; procesa la información recibida en los `inputs`, trabaja con ella, y devuelve una vista mediante los `exits`. La función `fn` es siempre asíncrona, lo que permite realizar operaciones en paralelo, de modo que el servidor pueda seguir atendiendo otras peticiones mientras se procesan estas. He aquí el código de la función de `create-product` (he omitido el código de las funciones externas `uploadFileToTemp` y `uploadFileToDrive` para reducir el tamaño del código):

```
fn: async function (inputs, exits) {
    const fs = require('fs');
    const path = require('path');

    const imageInfo = await sails.helpers.uploadFileToTemp(this.req); //carga
                                        //imágenes en sistemas de ficheros local

    if (imageInfo.media.length == 0) return exits.errorImage({ msg: "Error al
crear la imagen del producto" });

    let uploadInfo = [];
    for (let i = 0; i < imageInfo.media.length; i++) {
      const data = {
        media: imageInfo.media[i],
        requestBody: imageInfo.requestBody[i]
      }
      uploadInfo.push(await sails.helpers.uploadFileToDrive(data)); //carga
    }                                                    //imágenes a Google Drive

    fs.readdir('.tmp/uploads/', (err, files) => {//eliminamos los archivos en
                                                 //sistema de ficheros local
      if (err) throw err;
```

```
    for (const file of files) {
      fs.unlink(path.join('.tmp/uploads/', file), (err) => {
        if (err) throw err;
      });
    }
  });

  if (uploadInfo.length == 0) return exits.errorImage({ msg: "Error al
  crear la imagen del producto" });

  let productSize = [];
  if (typeof inputs.size == 'string') { //si solo se asocia una talla, se
                        //guarda como una colección con solo un elemento.
    productSize.push({
      size: inputs.size,
      stock: inputs.stock
    })
  } else {
    for (let i = 0; i < inputs.size.length; i++) {//en cambio, si se
              //introducen más tallas, se introduce el array completo.
      productSize.push({
        size: inputs.size[i],
        stock: inputs.stock[i]
      })
    }
  }

  var createProduct = await Products.create({
    name: inputs.name,
    id_images: uploadInfo,
    description: inputs.description,
    price: inputs.price,
    category: inputs.category,
    productSize: productSize,
    active: true,

  }).fetch();

  if (createProduct) {
    return exits.success({ msg: "Producto creado." });
  } else {
    return exits.errorCreate({ msg: "Error al crear el producto" });
  }
},
```

Esta acción es una de las más extensas en términos de lógica de la aplicación. Recibe los datos del nuevo producto desde la vista, y opera con ellos. La lógica de subida de imágenes está definida en las funciones `uploadFileToTemp()`, que se ocupa de recibir los archivos desde la vista, los carga en el sistema de ficheros del servidor, y se ocupa de generar los objetos con los archivos y los metadatos necesarios para cargar las imagenes a Google Drive, y `uploadFileToDrive(data)`, que recibe los objetos mencionados anteriormente y los carga a Google Drive.

No toda la lógica se encuentra dentro de las acciones; Sails.js permite crear también *helpers*, archivos JavaScript con una sintaxis muy parecida a las acciones, a los que se les puede llamar desde cualquier acción. En el caso visto anteriormente, de crear productos, los métodos `uploadFileToTemp` y `uploadFileToDrive` son *helpers*. Esto permite diseñar las acciones de manera que realicen una tarea y solo una, mientras que todo el código que pueda ser llamado por varias acciones se mantenga en ficheros separados y no tenga que repetirse. En este proyecto, he utilizado los *helpers* para la lógica de recuperar documentos para mostrar, es decir, recuperar un número específico de documentos de la base de datos, y mandarlos a la acción que los requiera.

5. Descripción de la web de la tienda

La funcionalidad de la web de la tienda consiste en gestionar las tareas de *front-office* descritas por el modelo de negocio de la empresa No.Studio. Esto implica permitir a un usuario registrarse en la aplicación, modificar sus datos, ver los productos que ofrece la tienda, añadir a su carrito de compras productos con talla y cantidades específicas, realizar la compra, y ver un historial de los pedidos que ha realizado anteriormente.

5.1 Modelo de dominio

La página web de la tienda maneja una serie de entidades, necesarias para el correcto funcionamiento de la aplicación. Como hemos comentado en el capítulo anterior, algunas entidades son compartidas por las dos aplicaciones web, mientras que otras son específicas de cada una. En este capítulo describiré las entidades que se manejan desde la web de la tienda, omitiendo la información sobre las entidades ya comentadas anteriormente.

También en este caso, todos los documentos de las entidades tienen asociado el identificador único generado por la base de datos, `_id`, la información sobre la fecha de creación y de actualización, `createdAt` y `updatedAt` respectivamente, y un campo de tipo booleano `active`, que determina si el elemento ha sido eliminado o no (excepto en las entidades de pedidos, véase 4.1.2 y 4.1.5, y la nueva entidad Carrito de la compra (`cart`)).

Las entidades que comparten las dos aplicaciones web son la de Pedidos (4.1.2), Productos (4.1.3), y Usuarios (4.1.6), mientras que la única entidad que solo se maneja desde la tienda web es la de Carrito de la compra.

5.1.1 Pedidos (`orders`)

La entidad Pedidos, como ya he explicado en el capítulo anterior, se ocupa de gestionar los pedidos realizados por los usuarios de la web de la tienda. Es en esta aplicación donde se generan los pedidos, en particular cuando el usuario decide realizar el pago del carrito de la compra. Los datos que se manejan aquí son los mismos que para el SCM, siendo la única diferencia que

en esta aplicación solo se permite crearlos, y no modificarlos de ninguna manera.

5.1.2 Productos (`products`)

La entidad Productos representa los productos que vende la tienda. Los datos son los mismos que los descritos en el capítulo anterior. En este caso, los productos no se pueden crear, modificar o eliminar. Solo se listan, y se añaden a carritos de compra y, sucesivamente, a un pedido. Desde la tienda web solo se modifica el número de existencias: a la hora de pagar un producto, se reduce en el número de existencias elegido por el usuario, para la talla correspondiente. En el caso de que un pedido se cancele, o se devuelva, es el SCM el indicado para gestionar esa lógica.

5.1.3 Usuarios (`users`)

La entidad Usuarios representa a los usuarios registrados desde la web de la tienda. Los datos que maneja esta entidad son los mismos que los descritos en el capítulo anterior. En esta aplicación es donde se registran, lo que es necesario para añadir productos a su carrito y posteriormente realizar el pago para enviar el pedido, pero no lo es para ver los productos. También se permite a un usuario modificar sus datos, y eliminar su cuenta de usuario.

5.1.4 Carrito de la compra (`cart`)

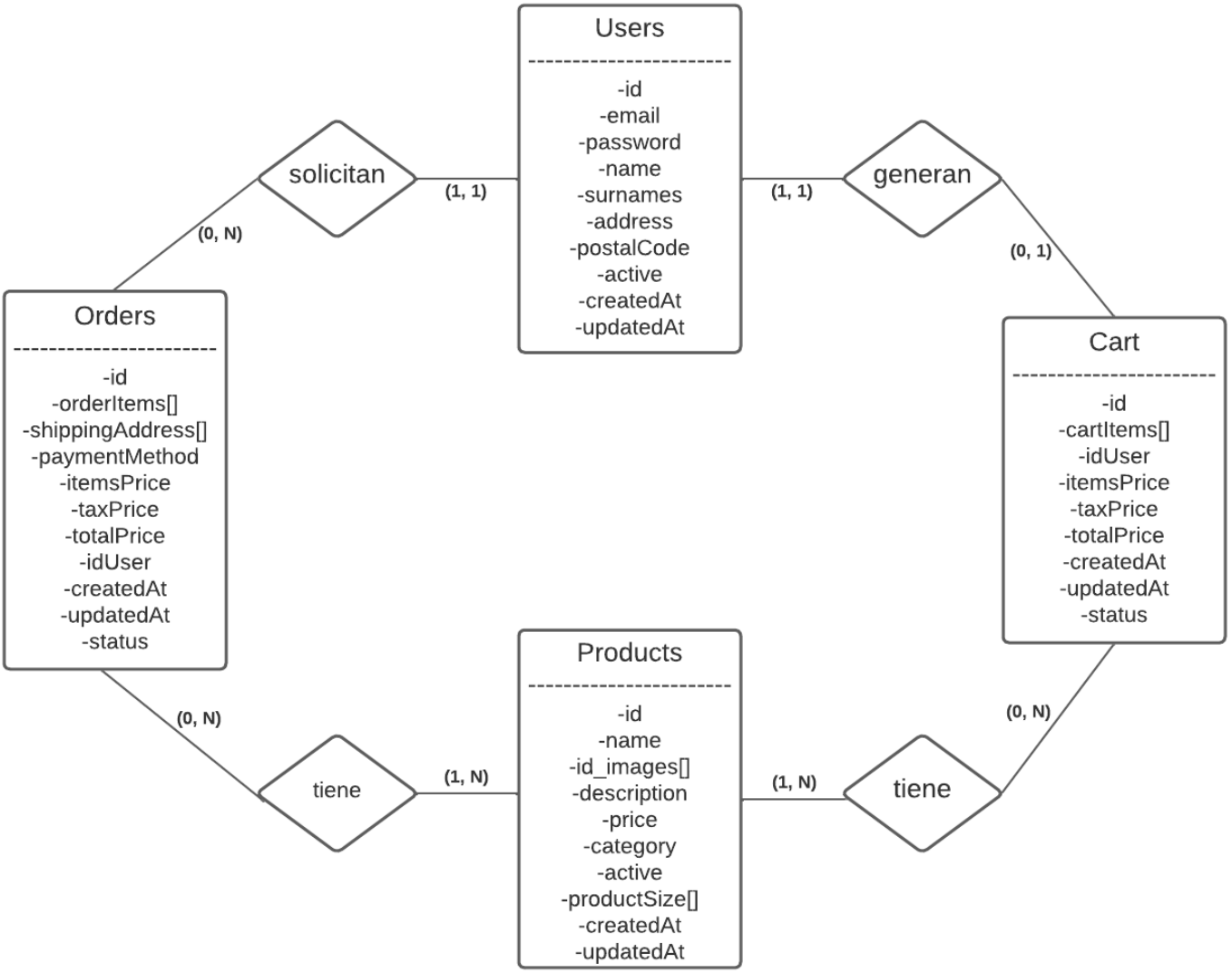
La entidad Carrito de la compra se ocupa de gestionar los datos y la funcionalidad asociada al carrito personal de cada uno de los usuarios. Este carrito es una lista de los productos que el usuario ha decidido comprar en un futuro. Sus campos están compuestos por, además de los definidos para todas las entidades, una lista de los productos en el carrito, `cartItems`; campos relacionados con el precio del producto, el IVA añadido y el precio total, `itemsPrice`, `taxPrice` y `totalPrice` respectivamente; y un identificador del usuario que ha generado el carrito, `idUser`.

Las operaciones que se contemplan para esta entidad son las de crearlo, cuando un usuario añade el primer producto al carrito; modificarlo, sea

añadiendo o quitando productos, o modificando la cantidad de productos de la talla asociada; y de eliminarlo, cuando todos los productos que existían en el carrito han sido eliminados.

5.2 Modelo Entidad-Relación de la web de la tienda.

A continuación muestro el diagrama de Entidad Relación del SCM:



Como se ve en el diagrama, las relaciones entre entidades son las siguientes:

- Carrito de la compra (`cart`) tiene una relación uno a muchos con Productos (`products`).

- Carrito de la compra (`cart`) tiene una relación uno a uno con Usuarios (`users`), ya que solo puede existir un usuario por carrito y viceversa.

Se puede ver además que las Colecciones Proveedores (`suppliers`), Pedidos a proveedores (`supplyOrder`) y Administradores (`admin`) no se contemplan en esta aplicación, siendo exclusivas del SCM.

5.3 Esquema de datos MongoDB para el SCM

En este capítulo represento el esquema de datos de la única colección que no he descrito anteriormente, la de Carrito de la compra (`cart`).

5.3.1 Carrito de la compra (`cart`)

Nombre	**Tipo**	**Referencia**	**Nullable**
`_id`	Id		No
`cartItems`	Document[]		No
`itemsPrice`	Number		No
`taxPrice`	Number		No
`totalPrice`	Number		No
`idUser`	Id	Usuarios(`users`)	No
`createdAt`	Date		No
`updatedAt`	Date		No

En este esquema, tenemos anidada una colección:

- **`cartItems:`**

Nombre	Tipo	Referencia	Null able
`_id`	Id		No
`name`	String		No
`id_images`	[]		No
`price`	Number		No
`product`	Id	Productos (`products`)	No
`qty`	Number		No
`size`	String		No

Esta entidad tiene la misma estructura que he descrito previamente para la colección anidada de `orderItems`, en la entidad Pedidos (4.3.2).

5.4 Capa de acceso a datos mediante Mongoose

Como he descrito anteriormente en el capítulo de Selección de herramientas y tecnologías (3), el acceso a la base de datos desde la tienda web está proporcionado por el ODM Mongoose.

La instrucción para conectarse con la base de datos mediante Mongoose está en el fichero `server.js`, que es el primero en ejecutarse a la hora de lanzar la aplicación. El código es el siguiente:

```
import mongoose from 'mongoose';

mongoose.connect(process.env.MONGODB_URL                                    ||
'mongodb+srv://NoStudio:[contraseña]@cluster0.sorbo.mongodb.net/[nombreBB
DD]?retryWrites=true&w=majority', {
  useNewUrlParser: true,
```

```
  useUnifiedTopology: true,
});
```

El primer argumento de la función `connect` es la URL de conexión a Mongo DB, con el usuario y contraseña y el nombre de la base de datos. A esta función pasamos también otros parámetros:

- `useNewUrlParser`: el driver de MongoDB desarrollado por Node.js ha desarrollado un parser de la cadena de conexión, al estar el driver desarrollado por Mongoose obsoleto; al ser un cambio sustancial que podría romper la compatibilidad con clientes ya existentes, se decidió deshabilitarlo por defecto, y que se tuviese que activar de manera explícita mediante este parámetro. Declarando esta opción como `true` asignamos el parser de Node.js.
- `useUnifiedTopology`: parecida a la opción anterior, sirve para poder utilizar el servicio de monitorización refactorizado por Node.js, ya que el de Mongoose está siendo actualizado, y se considera actualmente obsoleto.

El siguiente paso para poder utilizar la funcionalidad de Mongoose es el de generar los esquemas (`schema`) de datos. La estructura es similar a la utilizada para definir los esquemas en Sails.js. He aquí el ejemplo del esquema Carrito de la compra (`cart`):

```
import mongoose from 'mongoose';

const cartSchema = new mongoose.Schema(
    {
        cartItems: [
            {
                name: { type: String, required: true },
                qty: { type: Number, required: true },
                size: { type: String, required: true },
                image: { type: Array, required: true },
                price: { type: Number, required: true },
                idProduct: { type: mongoose.Schema.Types.ObjectId,
                   ref:  'Product', required: true, }, },
                   ],
        itemsPrice: { type: Number, required: true },
```

```
        taxPrice: { type: Number, required: true },
        totalPrice: { type: Number, required: true },
        active: { type: Boolean, required: true, default: true },
        idUser: { type: mongoose.Schema.Types.ObjectId, ref: 'User',
           required: true },
    },
    {
        timestamps: true,
    }
);
const Cart = mongoose.model('Cart', cartSchema);
export default Cart;
```

El primer paso es definir un `schema` con la estructura y tipos de los documentos que queremos guardar en esta colección; a continuación generamos un modelo a través de ese esquema, y finalmente lo exportamos, para poder utilizarlo de manera global en el proyecto, mediante la sentencia `import` en cualquier fichero que lo importe.

Habiendo generado el esquema y el modelo asociado a él, ya podemos acceder a la funcionalidad de Mongoose. He aquí un ejemplo de su uso, específicamente de la función crear o actualizar carrito modelo de Carrito de la compra (`cart`):

```
cartRouter.post(
    '/:id',
    isAuth,
    expressAsyncHandler(async (req, res) => {

        const cartItem = {  //generamos el objeto que contiene la información del
                            //producto del carrito.
            name: req.body.product.name,
            price: req.body.product.price,
            qty: req.body.qty,
            size: req.body.size,
            product: req.body.product._id
        }

        Cart.findOne({ idUser: req.user._id }, async (error, cart) => {
            if (error) {
                res.status(404).send({ message: 'Error de la base de datos.' });
```

```
            } else if (cart) { //hay un carrito para el usuario

                //si el producto ya está en el carrito con la misma talla,
                //se actualiza la cantidad de productos en el carrito.
                let alreadyInCart = false;
                for (let item = 0; item < cart.cartItems.length; item++) {
                    if (cart.cartItems[item].product == req.body.product._id &&
                    cart.cartItems[item].size == req.body.size) {
                        cart.cartItems[item].qty = Number(req.body.qty);
                        alreadyInCart = true;
                    }
                }

                //si el producto no está en el carrito, o está con otra talla
                //se añade el producto al carrito
                if (!alreadyInCart) {
                    cart.cartItems.push(cartItem);
                }

                const updatedCart = await cart.save();
                res.status(201).send({ message: 'Carrito actualizado', cart:
                updatedCart });

            } else { //no hay un carrito para el usuario
                const cart = new Cart({ //carrito no existe, se crea uno nuevo.
                    cartItems: cartItem,
                    itemsPrice: req.body.product.price,
                    taxPrice: req.body.product.price * 0.21,
                    totalPrice: req.body.product.price * req.body.qty,
                    idUser: req.user._id,
                });

                const createdCart = await cart.save();
                res
                    .status(201)
                    .send({ message: 'Generado carrito nuevo', cart: createdCart
});
            }
        });
    }
    )
);
```

Esta función se ocupa de crear o añadir un producto al carrito de la compra. Comprueba si existe un carrito para el usuario en cuestión o no. Si existe, quiere decir que ya había al menos un producto en el carrito del usuario.

En ese caso, se comprueba que el producto a añadir exista o no en el carrito con la talla seleccionada. En el caso de que ya exista, se actualiza la cantidad de productos de esa talla a la nueva cantidad proporcionada. En caso contrario, se añade un producto nuevo al carrito. En el caso de que no exista un carrito en la base de datos para el usuario, se genera un nuevo carrito con el producto seleccionado. La información de los productos y de la cantidad de productos para el stock asociado se encuentra en `cartItems`.

5.5 Funcionalidad implementada

Para la página web de la tienda, como ya he mencionado anteriormente, decidí utilizar una arquitectura específica a Redux, que tiene el mismo nombre. A diferencia de MVC, que es la arquitectura usada para el SCM, Redux no es una arquitectura bidireccional, si no unidireccional. El flujo de esta arquitectura es el siguiente:

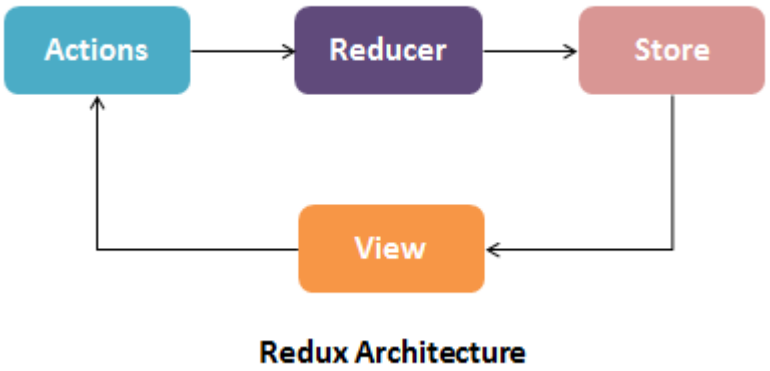


Figura 5.5.1 - Arquitectura Redux

Como ya he comentado anteriormente en el capítulo 3.1.9, Redux permite mantener el estado completo de la aplicación en un único componente, llamado `Store`. La única manera de mutar el estado es mediante *reducers*, funciones que reciben el estado anterior del árbol y una acción a realizar, y devuelven el nuevo estado de la aplicación. Cuando el `Store` ha sido actualizado, se devuelve a la vista el estado y los datos a mostrar.

Como ejemplo, muestro la ejecución y el flujo de datos durante la operación de añadir un producto al carrito. Desde la vista de un producto se permite añadirlo al carrito, de esta manera:

- primero, seleccionamos desde la interfaz la opción de añadir al carrito:

```
<button onClick={addToCartHandler} className="primary block">Add to Cart
</button>
     ...
const addToCartHandler = () =>{
    dispatch(addToCart(productId, qty,age));
  };
```

- `dispatch()` se ocupa de llamar a la función addToCart con los campos correspondientes. La llamada a `addToCart` devuelve otra función, que es la encargada de procesar el evento de pulsación del botón:

```
export const addToCart = (productId, qty, size) => async(dispatch, getState)
=> {
    const {data} = await Axios.get(`/api/products/${productId}`);
    dispatch({
        type: CART_ADD_ITEM,
        payload:{
            name: data.name,
            id_images: data.id_images,
            price: data.price,
            product: data._id,
            qty,
            size,
        },
    });
}
```

- Esta función se ocupa de realizar una llamada HTTP mediante Axios, para recuperar los datos desde el servidor, en este caso los datos del producto, llamando al *router* asociado. Devueltos los datos, lanza otro `dispatch()`, con los datos del producto a añadir al carrito, y los manda al `Reducer` de Carrito de la compra. El `Reducer` realiza lo siguiente:

```
export const cartReducer = (state={cartItems:[]}, action)=>{
    switch(action.type){
```

```
case CART_ADD_ITEM:
    const item = action.payload;
    const existItem = state.cartItems.find(x=> x.product ===
    item.product);
    if(existItem){
        return {
            ...state,
            cartItems: state.cartItems.map(x =>
                x.product === existItem.product? item: x
            ),
        }
    }else{
        return {...state, cartItems: [...state.cartItems, item] };
    } //otros action.type }
```

- El `Reducer` se ocupa de devolver el siguiente estado de la aplicación, es decir, modifica los campos en el atributo `state`. A continuación devuelve el nuevo estado de la aplicación a la vista.

6. Conclusiones y trabajo futuro

En este capítulo revisaré los objetivos definidos antes del comienzo del proyecto, comprobando si han sido realizados o no, y los objetivos futuros, si el proyecto se continuase desarrollando.

6.1 Objetivos

Se han cumplido gran parte de los objetivos planteados:

- **Definir los requisitos de la cliente del proyecto:**

Este objetivo se realizó satisfactoriamente y dentro del tiempo establecido en el plan de trabajo. Las reuniones eran siempre eficientes al tener la cliente del proyecto las ideas claras respecto a los requisitos a implementar, y al estar siempre disponible para responder a dudas o dar ideas sobre el desarrollo.

- **Diseñar un modelo de datos, basado en los requisitos de la cliente, que permita mantener los datos necesarios para las dos aplicaciones:**

Se consiguió diseñar un modelo de datos, eligiendo la tecnología más adecuada para cada aplicación web, y teniendo también en este caso reuniones con la cliente del proyecto para definir toda la información a manejar. Algunos de los datos fueron modificados durante el desarrollo de las aplicaciones, pero el modelo inicial es muy similar al final.

- **Desarrollar el SCM:**

Todos los requisitos requeridos por la cliente del proyecto fueron satisfactoriamente implementados en el SCM. La gestión de pedidos fue la tarea más prioritaria para la cliente, y fue la primera que se desarrolló, para que fuese dando indicaciones sobre la interfaz de usuario y el flujo de estados de los pedidos. Iterando a través de las entidades del proyecto, y siguiendo la misma metodología de reuniones constantes con la cliente, se fueron desarrollando las demás funcionalidades de la aplicación.

- **Desarrollar la tienda web:**

Casi todos los requisitos planteados para la tienda web durante la primera fase del proyecto han sido implementados. Se permite visualizar todos los productos de la tienda, crear y modificar usuarios, generar carritos de compra y, sucesivamente, pedidos. La pasarela de pago, en cambio, no ha dado tiempo a ser implementada, ya que durante el proceso de desarrollo no tuve tiempo para conseguir que funcionase satisfactoriamente.

Existe una intención por parte de la cliente de utilizar las dos aplicaciones web del proyecto de manera profesional. Está ya planteado continuar desarrollando las dos aplicaciones para convertir la tienda web en una página más competitiva, mejorando la interfaz de usuario y aumentando la funcionalidad, e implementar funcionalidades adicionales al SCM.

6.2 Dificultades durante el desarrollo del proyecto

Durante el desarrollo del proyecto, me he encontrado con una serie de dificultades que resolver. En este capítulo recogeré las más importantes y describiré si se han solucionado, y en qué ha consistido la solución.

Una de estas dificultades ha sido la gran cantidad de tecnologías utilizadas para desarrollar las dos aplicaciones web, la gran parte de ellas totalmente desconocidas para mí previamente a realizar este trabajo. Durante el tiempo de formación conseguí asentar unas bases para, durante el desarrollo, continuar a aprender sobre estas, pero aun así gran parte del tiempo de desarrollo ha sido utilizado para familiarizarme con las nuevas herramientas y tecnologías. Actualmente, con el desarrollo del proyecto terminado, he conseguido ampliar mis conocimientos sobre el desarrollo web, y, en el caso de que el proyecto continúe, sería capaz de continuar el desarrollo de una manera más rápida y efectiva.

Una dificultad más específica, y derivada de la anterior, fué la de implementar el sistema de guardado, edición y eliminación de las imágenes de los productos. Como he explicado anteriormente en los capítulos 3.2.6 y 3.2.7, fue necesario probar distintas tecnologías a la hora de intentar cumplir con este

requisito. En un primer momento intenté convertir las imágenes en una cadena de texto en *base64*, y guardar esta cadena en la base de datos mediante `skipper-gridfs`, para luego recogerla y recomponerla del lado del cliente. Esto funcionó, pero recoger la cadena y reconvertir la imágen es un proceso lento, y no era posible realizar esta carga de manera asíncrona, ralentizando la carga de la página y afectando a la experiencia de usuario de las dos aplicaciones. Por tanto, tuve que buscar una alternativa y, comentándolo con la cliente del proyecto, decidimos almacenar las imágenes en Google Drive. Esto se decidió así para que la cliente tuviese también acceso a un repositorio conocido por ella previamente, y pudiese ver las imágenes almacenadas en el repositorio de manera sencilla. Esto implicó investigar sobre la *API* de Google, generar una clave de acceso a esta e implementar la funcionalidad. La diferencia en la experiencia de usuario de las dos aplicaciones es considerable, aunque se podría mejorar implementando algún sistema para reducir la resolución de las imágenes durante la carga, manteniendo en Google Drive la resolución máxima necesaria para el uso de la imagen en las aplicaciones.

Sin duda, el mayor contratiempo encontrado durante el desarrollo de este proyecto ha sido perder a un integrante del mismo. Comencé el proyecto durante el primer cuatrimestre de 2021 con un compañero. Por motivos personales, tuvo que separarse del trabajo durante la fase de desarrollo de las dos aplicaciones. Esto implicó tener que realizar un trabajo de formación adicional acerca de las herramientas que mi compañero había decidido aprender a utilizar para realizar su parte del trabajo, y ralentizó drásticamente el proceso de desarrollo. Aun así, creo que he conseguido cumplir con los requisitos propuestos por la cliente del proyecto, a excepción de la pasarela de pago, que está planeado implementarla cuanto antes.

6.3 Trabajo futuro

Durante la continuación del proyecto, se plantean implementar las siguientes funcionalidades a las aplicaciones:

6.3.1 SCM

- **Análisis de datos de pedidos y productos/materiales**:

Se plantea generar algoritmos y vistas para representar información interesante para el modelo de negocio de la empresa, como recuperar los productos más vendidos o los más rentables, añadiendo la información necesaria para calcular el beneficio de comprar un material a un cierto proveedor y convertirlo en un producto a vender en la tienda.

- **Envío de correos electrónicos desde el SCM:**

Sería útil permitir enviar correos electrónicos directamente desde el SCM. Esto permitiría comunicarse directamente con los clientes de la tienda respecto a un pedido, con la entidad que entregue los pedidos a estos, o que entregue los materiales a la tienda, o con los proveedores respecto a algún pedido realizado.

6.3.2 Tienda web

- **Implementar la pasarela de pago:**

Terminado el proyecto, queda claro que implementar la pasarela de pago es fundamental para poder lanzar una primera versión funcional de la tienda web.

- **Desarrollar una vista que permita mostrar contenido variado:**

La cliente comentó en las fases finales del desarrollo del proyecto que sería interesante tener una vista donde poder presentar contenido como videos o fotografías representativas de la marca de la empresa.

- **Crear una lista de deseos:**

Sería interesante, en el futuro, que cada usuario de la tienda web dispusiera de una lista de deseos, donde pudiese mantener los productos que posiblemente decida comprar en un futuro.

7. Conclusions and future work

In this chapter I will review the objectives defined at the start of the project, checking whether they have been achieved or not, the difficulties encountered during the project's development, and the future objectives, should the project be further developed.

7.1 Objectives

Most of the objectives have been achieved:

- **Define the client's requirements for the project:**

This objective was achieved satisfactorily and within the time established in the work plan. The meetings were always efficient as the project client had clear ideas about the requirements to be implemented, and was always available to answer questions or give ideas about the development.

- **Design a data model, based on the client's requirements, that allows maintaining the necessary information for the two applications:**

A data model was successfully designed, choosing the most appropriate technology for each web application, and also having meetings with the project's client to define all the information to be handled. Some of the data were modified during the development of the applications, but the initial model is very similar to the final one.

- **Develop the SCM:**

All the client's requirements for the project were successfully implemented in the SCM. Order management was the highest priority task for the client, and it was the first one to be developed, so that she was giving indications about the user interface and the order status flow. Iterating through the project entities, and following the same methodology of constant meetings with the client, the other functionalities of the application were developed.

- **Develop the e-commerce web store:**

Almost all of the requirements set for the web store during the first phase of the project have been implemented. It is possible to view all the products in the store, create and modify users, generate shopping carts and, subsequently, orders. The payment system, on the other hand, has not had time to be implemented, since during the development process I did not have time to make it work satisfactorily.

The client intends to use the two web applications of the project in a professional manner. It is already planned to continue developing the two applications to make the web store more competitive, improving the user interface and increasing the functionality, and to implement additional functionalities to the SCM.

7.2 Challenges during project development

During the development of the project, I have encountered a series of challenges to solve. In this chapter I will list the most important ones and describe whether they have been solved, and what the solution consisted of.

One of these challenges has been the large number of technologies used to develop the two web applications, most of them totally unknown to me before developing this project. During the training time I managed to lay some foundations to continue learning about them during the development, but still much of the development time has been used to familiarize myself with new tools and technologies. Currently, with the development of the project completed, I have managed to expand my knowledge of web development, and, in the event that the project continues, I would be able to continue the development in a faster and more effective way.

A more specific challenge, and derived from the previous one, was to implement the system for storing, editing and deleting product images. As I have explained previously in chapters 3.2.6 and 3.2.7, it was necessary to try to use several technologies when trying to fulfill this requirement. At first I tried converting the images into a base64 text string, and saving this string in the database using `skipper-gridfs`, and then retrieving and recomposing it on

the client side. This did work, but collecting the string and reconverting the image is a lengthy process, and it was not possible to perform this loading asynchronously, slowing down the page loading times and compromising the user experience of the two applications. Therefore, I had to look for an alternative and, discussing it with the client of the project, we decided to store the images in Google Drive. This was decided so that the client would also have access to a repository previously known to her, and would be able to view the images stored in the repository easily. This involved researching the Google API, generating a key to access it and implementing the functionality. The difference in the user experience of the two applications is considerable, although it could be improved by implementing some system to reduce the resolution of the images during loading, keeping in Google Drive the maximum resolution necessary for the use of the image in the applications.

But without a doubt the biggest setback encountered during the development of this project has been losing a member of the project itself. I started the project during the first quarter of 2021 with a partner. For personal reasons, he had to separate from work during the development phase of the two applications. This meant having to do additional training work on the tools my partner had decided to learn to use to do his part of the work, and drastically slowed down the development process. Even so, I think I have managed to meet the requirements proposed by the client of the project, with the exception of the payment gateway, which is planned to be implemented as soon as possible.

7.3 Future work

During the continuation of the project, we plan to implement the following functionalities to the applications:

7.3.1 SCM

- **Order and product/material data analysis:**

It is proposed to generate algorithms and views to represent interesting information for the company's business model, such as recovering the best-selling or most profitable products, adding the necessary information to

calculate the benefit of buying a material from a certain supplier and converting it into a product to sell in the store.

- **Sending emails from the SCM:**

It would be useful to be able to send emails directly from the SCM. This would allow communicating directly with the store's customers regarding an order, with the entity delivering orders to them or delivering materials to the store, or with suppliers regarding an order placed.

7.3.2 E-commerce Web Store

- **Implement the payment system:**

Now that the project is finished, it is clear that implementing the payment gateway is essential to be able to launch a first operational version of the e-commerce web store.

- **Development of a view that allows displaying varied content:**

The project's client mentioned in the final stages of development that it would be interesting to have a view where they could present content such as videos or photographs representative of the company's brand.

- **Create a wish list:**

In the future, it would be interesting if each user of the web store could have a wish list, where he/she could keep the products that he/she might decide to buy in the future.

Bibliografía

[1] Jon Duckett, *HTML & CSS: Design and Build Websites*, Wiley, ISBN 9781118008188, 2011

[2] David Flanagan, *JavaScript: The Definitive Guide : Master the World's Most-used Programming Language*, O'Reilly Media, Incorporated, ISBN 9781491952023, 2020

[3] Ethan Brown, *Web Development with Node and Express*, O'Reilly Media, Incorporated, ISBN 9781491988640, 2018

[4] Robin Wieruch, *The Road to Learn React: Your Journey to Master Plain Yet Pragmatic React. Js*, CreateSpace Independent Publishing Platform, ISBN 9781986338820, 2018

[5] Irl Nathan and Michael McNeil, *Sails.js in Action*, Manning, ISBN 9781638353478, 2017

[6] Jennifer Kyrnin, *Bootstrap in 24 Hours*, Sams, ISBN 9780672337048, 2015

[7] Marc Garreau, Will Faurot, *Redux in Action*, Manning, ISBN 9781617294976, 2018

[8] Kristina Chodorow, Eoin Brazil, Shannon Bradshaw, *MongoDB: The Definitive Guide : Powerful and Scalable Data Storage*, O'Reilly Media, Incorporated, ISBN 1491954469, 2019

[9] Simon Holmes, *Mongoose for Application Development*, Packt Publishing, ISBN 9781782168195, 2013

[10] Scott Chacon , Ben Straub, *Pro Git*, Apress, ISBN 1484200772, 2014