UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Diseño e implementación de un módulo logístico para una aplicación de gestión deportiva

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: López Peña, David

Tutor/a: Oliver Gil, José Salvador

Cotutor/a externo: MORA CARRASCOSA, RAFAEL

CURSO ACADÉMICO: 2021/2022

Resumen

En este proyecto abordaremos el desarrollo de un módulo en una aplicación web y móvil para la logística de artículos, que facilite al usuario, de una manera rápida y sencilla, tanto la gestión como la introducción de contenido.

La aplicación web/móvil está orientada a la gestión deportiva, más concretamente al golf, con la posibilidad de expandir a futuro a más deportes. De momento se está utilizando en España y en algunas universidades de EE. UU. Se ha planteado la necesidad de gestionar los almacenes de dichas universidades para tener más control sobre el stock de los artículos, de ahí la necesidad de crear este módulo.

El proyecto se ha desarrollado en un repositorio *git*, el cual nos facilitará el control de versiones y las actualizaciones para su posterior integración en la aplicación.

Para la realización del módulo se ha usado Angular como *framework* principal, MySQL para la base de datos, para el despliegue se ha usado un servidor VPS en la plataforma OVH y para el *backend* se ha utilizado PHP 7.4.

Palabras clave: gestión, Angular, MySQL, logística, almacén, aplicación web, CMS, stock.

Abstract

In this project we will address the development of a module in a web and mobile application for the logistics of articles, which facilitates the user in a quick and easy way both the management and the introduction of content.

The web/mobile application is oriented to sports management, more specifically golf, with the possibility of expanding to more sports in the future. At the moment it is being used in Spain and in some universities in the USA. The need has arisen to manage the warehouses of these universities to have more control over the stock of items, hence the need to create this module.

The project was developed in a git repository, which will facilitate version control and updates for subsequent integration into the application.

For the realization of the module we used Angular as the main framework, MySQL for the database, for the deployment we used a VPS server on the OVH platform and for the backend we used PHP 7.4.

Key words: management, Angular, MySQL, logistics, warehouse, web application, CMS, stock.

Contenidos

Sumario

1. Introducción

En este proyecto abordaremos el desarrollo de un módulo en una aplicación web y móvil para la logística de artículos, que facilite al usuario de una manera rápida y sencilla tanto la gestión de estos como la introducción de contenido.

1.1 Objetivos del proyecto

El objetivo principal de este proyecto es desarrollar un módulo que sea capaz de gestionar las entradas y salidas de producto, así como también registrar un historial con los movimientos realizados para su consulta en caso de necesidad. Uno de los principales problemas de estas aplicaciones es que pueden ser muy poco intuitivas o que tengas que dar muchos pasos para realizar un simple movimiento, por esto, nosotros tendremos en cuenta las experiencias que nos transmitan los usuarios y realizaremos un módulo usable y sencillo en base a ello.

Además de poder realizar movimientos de producto, otro objetivo fundamental sería la posibilidad de que los clientes pudieran registrar y editar sus propios artículos ya sea mediante la web o desde el móvil.

Otro objetivo importante consiste en que el proyecto sea modular y se adapte correctamente a la aplicación sin alterar su funcionamiento habitual para que su integración sea lo más fácil posible.

Por último, algo para tener en cuenta sería que el módulo va a ser utilizado fuera de España, por lo que este deberá estar disponible tanto en inglés como en español, por ello, haremos uso de las utilidades que nos proporciona el *framework* de Angular y de un estilo tanto homogéneo como responsive durante todo el desarrollo.

1.2 Contexto de la aplicación

La aplicación se define a sí misma como un software diseñado para la gestión, control y seguimiento de centros deportivos o recreativos enfocados al deporte del golf, con un diseño completamente personalizado y adaptado para proporcionarles a los jugadores muchas herramientas para el registro y seguimiento de datos con el fin de mejorar en el ámbito competitivo. Para lograr dicha mejora la aplicación consta de varios módulos, los cuales son:

- **Estadísticas:** Este módulo proporciona tanto a jugadores como a entrenadores la inserción de datos llevados a cabo durante un torneo o durante una práctica con el fin de realizar un seguimiento progresivo de las capacidades de cada jugador.

- **Torneos**: En este módulo se pueden tanto registrar, editar o borrar los torneos en la aplicación asociados a cada universidad con sus temporadas y equipos.
- **Home**: Es la vista principal de la aplicación donde se encuentra el calendario de eventos personales (torneos,viajes,practicas,etc…) y el chat, con el que podemos comunicarnos con miembros de nuestro mismo equipo o con nuestro coach.
- **Jugadores**: Aquí se organizan los jugadores de cada entidad con sus respectivos perfiles y datos relevantes para la entidad.
- **Chat**: Este módulo permite la creación de chats individuales o chats de grupo para que tanto jugadores como entrenadores puedan intercambiar mensajes dentro de la aplicación.
- **Prácticas**: En este módulo se gestionan los entrenamientos que proponen los entrenadores para sus jugadores con el fin de mejorar su rendimiento.
- **Viajes**: Esta parte de la aplicación se encarga de gestionar los viajes que registran los entrenadores, normalmente para ir a jugar un torneo o para viajar a un campo a realizar entrenamientos.
- ***Recruitment***: Este módulo consta de un listado de jugadores que los entrenadores probablemente querrían en un futuro. La finalidad es tener información de que pasos has dado con ciertos jugadores y cuando, algo así como un mini CRM.
- **Gestión**: Todo lo relacionado con la configuración y gestión de: jugadores, instituciones, equipos, usuarios avanzados, etc.
- **Almacén**: El módulo que desarrollaremos en este proyecto, que se encargará de gestionar artículos utilicen los jugadores tanto para entrenar como para competir.

Por otra parte, también tenemos 3 tipos de roles definidos en la aplicación:

- **Administradores**: Tienen acceso total a la aplicación y cuentan con todo tipo de permisos a la hora de crear, borrar o editar contenido.
- **Entrenadores**: Pueden crear y editar contenido que se encuentra dentro de su entidad, mientras que en el borrado tendrán ciertas restricciones
- **Jugadores**: Los jugadores verán solo los módulos de la aplicación a los que se les haya dado acceso y por otra parte podrán tanto interactuar como registrar información en el contenido proporcionado por entrenadores y administradores. Sus permisos de edición y borrado son bastante limitados.

1.3 Motivación

Para este proyecto desarrollaremos un módulo de gestión logística para dar soporte a tanto universidades como federaciones. Teniendo como base una aplicación ya montada y en funcionamiento, nuestro objetivo es tener un sistema modular que se adapte a ella y que sea capaz de mantener el estilo homogéneo e intuitivo de dicha aplicación.

La necesidad de crear la aplicación vino dada por la escasa competencia que había en el sector tanto en España como en EE. UU., lo que generaba una oportunidad de poder asentarse en el mercado y de añadir funcionalidades adicionales para hacer de ésta un atractivo comercial. Entre sus funcionalidades adicionales se encuentra el módulo de gestión logística que desarrollaremos en este proyecto.

La decisión de crear este proyecto es consecuencia de la necesidad que tenían tanto universidades como federaciones de gestionar su material deportivo de una manera fácil y cómoda, y de cómo las otras aplicaciones de gestión deportiva carecían de este aspecto en concreto.

Actualmente las universidades gestionan sus productos mediante hojas ‘excel’ llevando un registro de todos los productos existentes y colocando a mano la cantidad que se extrae o se introduce. Este método da lugar a descuadres de producto, no tener un seguimiento del responsable que dirige los movimientos, artículos prestados a jugadores que no vuelven porque no al cabo del tiempo ya no se sabe a quién se les entregó, etc. Nuestro módulo dará solución a sus necesidades proporcionando:

- Un sistema de seguimiento de artículos capaz de registrar todos los movimientos realizados, con datos tales como, el responsable que realizó el movimiento, la fecha de las gestiones realizadas, el jugador al que se le prestó el producto, etc.
- Una lista de artículos mucho más simplificada e intuitiva.
- Un control de la cantidad de producto cotejada en todo momento con la base de datos a la hora de realizar cada gestión.
- Una forma simple y fácil de realizar gestiones donde cada entrenador solo tendrá acceso a sus equipos y jugadores con la finalidad de que cada responsable se haga cargo de sus equipos para reducir la perdida de material.

Al haber conseguido experiencia previa con el *framework* de Angular, podremos aprovechar y profundizar hasta las funcionalidades más avanzadas para lograr que nuestro proyecto sea capaz de asentarse en la aplicación sobre la que se está desarrollando, y como cabría esperar, que no interfiera en su correcto funcionamiento.

Principalmente, nuestro proyecto será destinado a tratar con material exclusivamente deportivo de universidades y federaciones, lo que quiere decir que no tendremos muchos tipos distintos de productos, por lo podremos permitirnos hacerlos un poco más vistosos y proporcionar a cada universidad una identidad única dentro de nuestro módulo, sin olvidar por supuesto que el sistema que vayamos a desarrollar sea lo más funcional posible.

1.4 Estructura de la memoria

El presente proyecto se divide en los siguientes bloques:

- En este primer capítulo hemos definido los objetivos que nos hemos marcado, el contexto de la aplicación sobre la cual desarrollaremos nuestro proyecto, los detalles sobre cómo las universidades han llevado a cabo sus gestiones actualmente, las necesidades que estas requerían, las ventajas que aportará nuestro proyecto al problema en cuestión y este mismo apartado.
- En el segundo, revisaremos las soluciones existentes que hay en el mercado que se parezcan a la nuestra.
- En el tercero apartado, detallaremos la metodología de trabajo que hemos utilizado para la elaboración de nuestro proyecto.
- En el cuarto punto, explicaremos como hemos diseño la base de datos de nuestro proyecto y como estará organizado internamente.
- En el quinto punto, comentaremos los elementos que intervienen en nuestro módulo, así como los lenguajes, herramientas y el *framework* que hemos empleado para la realización del proyecto y mostraremos algunos ejemplos de código que serán relevantes para cumplir los objetivos planteados en el primer apartado.
- En el sexto apartado, efectuaremos una prueba de concepto, explicando como usaremos nuestro módulo para los diferentes casos de uso que hemos planteado en el apartado de diseño.
- En el último apartado, declararemos una serie de mejoras propuestas que se implementarán a posteriori de la entrega del proyecto, una opinión personal del proyecto y un resumen del trabajo realizado.

2. Estado del arte

Para hacernos una idea más acertada del proyecto que vamos a realizar, analizaremos algunas de las aplicaciones de la competencia para observar que ventajas nos aportarían a nuestro sistema o en caso contrario que puntos negativos no querríamos aplicar a nuestro módulo.

2.1 Ventajas de utilizar un software de gestión de almacén

Actualmente los beneficios de disponer de un sistema de stock, ya sea en una empresa o en un almacén como tal, son tales como:

- **Ahorrar dinero en reducir gastos asociados al espacio**: mercancía que se puede quedar obsoleta o no se vende.
- **Mejorar el servicio al cliente**: un sistema rápido y eficiente equivale a ser mas competitivo y generar un mayor número de ventas a largo plazo.
- **Una correcta gestión**: tanto la gestión de los inventarios de los almacenes como otras tareas administrativas requieren de mucho tiempo y esfuerzo. No obstante, un sistema de gestión rápida y eficaz ayudará a reducir la carga de trabajo lo que se traducirá en reducción de costes.
- **Trazabilidad del producto**: un sistema fiable de seguimiento de productos nos evitará perdidas de dinero por artículos extraviados.

En líneas generales un sistema de gestión de stock se encarga de controlar cada una de las operaciones dedicadas a regular el flujo de mercaderías o productos en una empresa. También se encarga de asegurar que los costes asociados al inventario de productos sean los mínimos posibles sin que ello interfiera con el servicio que se les ofrece a los clientes.

2.2 Soluciones existentes

En este apartado vamos a analizar algunas de las aplicaciones de gestión de *stock* que mas cuota de mercado tiene actualmente para analizar sus pros y sus contras.

2.1.1 Catinfog

Catinfog es un sistema de gestión de almacenes muy sencillo que está orientado al pequeño comercio y nos ofrece un control simple de nuestros productos, permitiéndonos gestionar ventas en la nube para tiendas minoristas y con un control de artículos muy básico para cualquier tipo de tienda que venda al público.

Algunas de sus principales ventajas serían: la interfaz intuitiva que tiene viene con una tienda online integrada, ofrece soporte para devoluciones, funciona en la nube, actualizaciones de software continuas… En cuanto a sus contras podríamos decir que está demasiado encasillado en el sector *retail* y bares por lo que su diseño apenas es personalizable.

Podemos tomar como puntos importantes de esta aplicación lo intuitivo de su interfaz y su sencillez, sin embargo, nuestro diseño estará enfocado a otro tipo de público.

2.1.2 Snipe-IT

Snipe-IT es una aplicación relativamente moderna desarrollada en código libre para su uso en almacenes de todo tipo y tamaño. Tiene una comunidad muy activa y se estima que han tenido aproximadamente a unos 80 programadores aportando código a la aplicación.

Su principal ventaja es que dispone de una api que aporta respuestas rápidas y le permite conectarse con cualquier sistema de planificación de recursos empresariales. Por otra parte, no deja de ser una aplicación de código abierto, lo que implica que podremos encontrarnos con algún que otro bug en el camino y algunas de las opiniones generales de la aplicación sugieren que la interfaz no es muy intuitiva.

Aunque la idea de tener una api en nuestro módulo suene bien, la realidad es que nuestro *backend* ya está desarrollado para el resto de nuestra aplicación, por lo que añadir una api requeriría de un refactor completo, tarea la cual ahora mismo es inviable.

2.1.3 Facturascripts

Facturascripts es un software de código abierto con un código bastante limpio y con unas librerías muy sencillas de utilizar para adaptar el programa a distintos casos de uso.

Su ventaja principal es el buen seguimiento de producto que tiene, ofreciendo todo tipo de información del empleado que ha manipulado el artículo, ya sea a la hora de mover la mercancía entre almacenes o para realizar operaciones de entrada o salida. También permite crear albaranes y facturas, aunque no dispone de gestión comercial ni de procesamiento de pagos.

Sin duda lo más importante de esta aplicación es la ventaja anteriormente mencionada del seguimiento de mercancía y nos inspiraremos en ella a la hora de crear el histórico en nuestro proyecto para solventar el problema de la pérdida de producto en las universidades.

2.1.4 PartKeepr

Partkeepr es un software de código abierto que se dedica única y exclusivamente a la gestión de almacenes sin entrar en el entorno comercial.

Sus principales puntos fuertes son: una base de datos muy optimizada que permite una búsqueda muy rápida de productos, un historial que muestra por todos los procesos por los que ha pasado un producto y una comunidad bastante activa que se preocupa por mantenerlo y arreglar los bugs.

Si bien es cierto que tiene muchos puntos a favor y se asemeja a lo que nosotros estamos buscando, algunos de los contras, como su diseño de programa de escritorio que le impide encajar en nuestra aplicación o su base de datos optimizada que no podríamos aprovechar debido a que nuestros artículos se guardarán en nuestra aplicación, nos impide tenerla en cuenta a la hora de desarrollar nuestro proyecto.

3. Metodología

Debido a que la aplicación que va a englobar nuestro proyecto ha sido desarrollada con el *framework* Angular nos adaptaremos a esta tecnología y la emplearemos a lo largo del desarrollo.

Dicho esto, definiremos tanto una operativa de trabajo óptima que se ajuste a nuestro proyecto como una serie de objetivos de tiempo que deberemos de cumplir. Dado que en esta aplicación no voy a trabajar solo consideré aceptar la metodología de trabajo que ya estaba adoptado el equipo de trabajo de la aplicación.

Dicha metodología consta de entregas parciales y semanales del producto final, definidas por una parte por nosotros mismos y, por otra, por el equipo de trabajo. El consenso y la comunicación con el equipo es esencial a la hora de definir plazos y de saber si estos son coherentes o no, por lo que realizaremos también reuniones semanales para discernir si nuestros objetivos van fuera de plazo y en caso afirmativo los posibles fallos inesperados que nos hayan podido surgir.

Esta metodología de trabajo se denomina Scrumb y lo que pretende es alcanzar el mejor resultado de un proyecto determinado en el que los requisitos son cambiantes y las tareas sueles estar poco definidas. Las prácticas aplicadas con esta metodología nos ayudarán a coordinarnos con el equipo para alcanzar un alto nivel de competitivo, reaccionar frente a competidores directos y identificar y solucionar ineficiencias de forma sistemática.

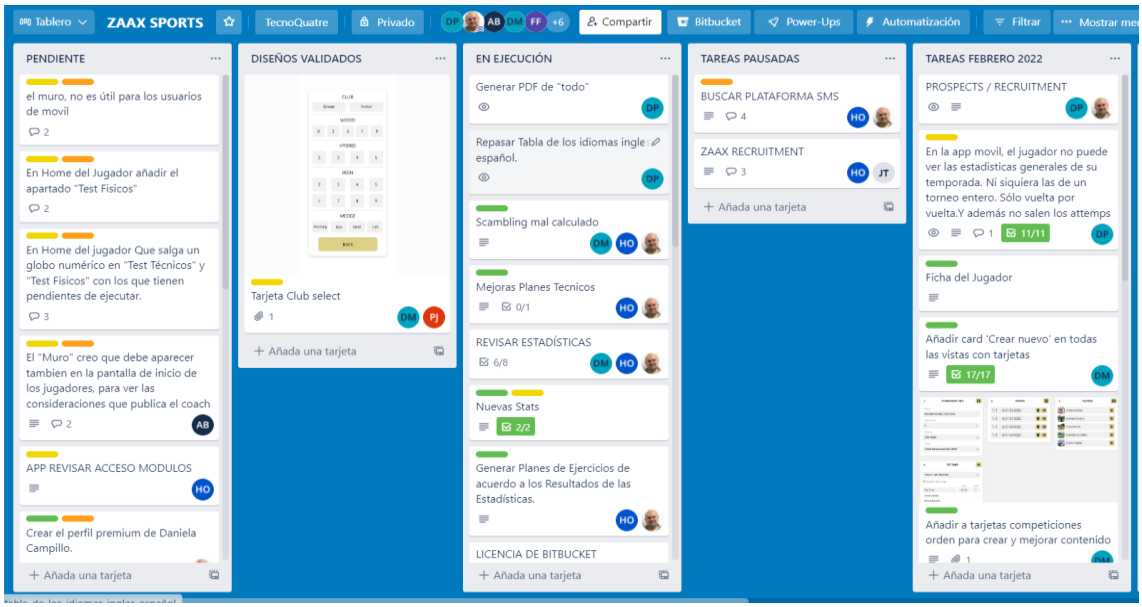

Figura 1 – Tablón de trello en pleno desarrollo del proyecto

Para organizar nuestros objetivos de trabajo en el proyecto nos adheriremos al tablero creado en Trello (ver figura 1) por nuestro equipo, donde registraremos y definiremos todos los procesos que vayamos a seguir en la elaboración de nuestro proyecto, pasando desde el diseño y la maquetación de las distintas pantallas que necesitaremos, hasta las funcionalidades que desarrollaremos tanto en *frontend* como en *backend.*

Por otra parte, ya que estamos trabajando en equipo, necesitaremos cohesionar nuestro trabajo con el de nuestros compañeros habitualmente por lo que el control de versiones es vital a la hora de unir partes de código para reducir problemas espontáneos que puedan surgir a lo largo del desarrollo.

Por lo comentado anteriormente haremos uso de la tecnología Git, un sistema de control de versiones distribuido que nos permitirá guardar nuestro trabajo con la posibilidad de volver sobre nuestros pasos si encontráramos algún obstáculo en el camino y con la seguridad de que nuestro desarrollo no interfiera con las labores de nuestros compañeros. Para lograr dicha seguridad se ha separado el proyecto en varias ramas de desarrollo que son:

-**Master**: es la rama principal donde se encuentra todo el código de la aplicación ya en funcionamiento, código el cual ya ha sido testeado y revisado para evitar fallos que puedan suponer repercusiones fatales sobre todo en el ámbito económico.

-**Desarrollo**: en esta rama se cohesiona el código de todos los integrantes del equipo de trabajo y se revisa para su posterior adhesión a la rama *master*.

-**Traducción**: todo lo referente con la traducción de la aplicación irá incluido en esta rama.

-**Diseño**: aquí se encuentra el código referente a los diseños globales de la aplicación.

-***Features***: estas son las ramas de los módulos de la aplicación, cada *feature* corresponde a una funcionalidad de la aplicación como por ejemplo el módulo de almacén que desarrollaremos en este proyecto.

DP	David López Peña	c219ad6	rsponsive dialog movimientos	2022-03-07
DP	David López Peña	f5c8777	MERGED Merge branch 'master' of https://bitbucket....	2022-03-07
DP	David López Peña	a476a0b	dialog movimientos responsive	2022-03-07
HO	Héctor Orts	fd6e0ff	MERGED Merge branch 'master' of https://bitbucket....	2022-03-07
HO	Héctor Orts	e3a65c0	Arreglado boton activo (error al no cambiar de pagia...	2022-03-07
DP	David López Peña	a2d0915	mejoras almacen	2022-03-03
DP	David López Peña	e872fd4	taduccion almacen	2022-03-02
DP	David López Peña	3c3ce5b	cambiar foto retoques	2022-03-02
DP	David López Peña	ab6b8ce	scanner almacen movimientos + retowues subir ima...	2022-03-01
DP	David López Peña	34ac3ec	entradas/salidas completo	2022-02-28
DP	David López Peña	e87d667	entradas/salidas almost	2022-02-24
DP	David López Peña	c183013	working on dialog	2022-02-22
DP	David López Peña	b1ed21f	MERGED Merge branch 'master' of https://bitbucket....	2022-02-22
DP	David López Peña	c188e1d	retoques galeria + añadidas notas	2022-02-22
HO	Héctor Orts	7ce84d8	barcode	2022-02-21
HO	Héctor Orts	2cf469d	Barcode scanner	2022-02-21

Figura 2 – Registro de commits visualizado en Bitbucket

Por último, Git también nos ofrece un registro con los cambios realizados en cada una de las ramas (ver figura 2) y de su responsable, para en caso de duda o problema de cualquier tipo preguntar al integrante del grupo en cuestión.

4. Diseño de la solución

Antes de empezar con el desarrollo de nuestro código vamos a definir la estructura de la base de datos, así como las dependencias de las que dependerá nuestro proyecto.

Nuestro módulo estará montado sobre una base de datos relacional por lo que tendremos varias tablas, o relaciones, que se compondrán a su vez de campos (columnas) y registros (filas).

Dichas tablas tendrán que ser coherentes entre sí y planificaremos su contenido en función de dos aspectos: la información disponible y la información que necesitamos.

4.1 Estructura de la base de datos

Con el fin de garantizar la persistencia de los datos en nuestro proyecto implementaremos las siguientes tablas en base de datos:

- Stock: en esta tabla registraremos los artículos de nuestro módulo (ver figura 3), ayudándonos de los siguientes campos:
    - Id: identificador único para cada artículo.
    - Entidad: identificador de la entidad a la que pertenece.
    - Creador: identificador del usuario que ha creado el artículo.
    - Nombre: nombre del artículo.
    - CodigoBarras: serie numérica que representa un código de barras o QR asociado a un artículo.
    - Cantidad: número que representa la cantidad actual del artículo.
    - TipoArticulo: el tipo del artículo en cuestión.
    - Talla: la talla del artículo si se trata de ropa o calzado.
    - Color: el color representativo del artículo, si tiene alguno.
    - Equipo: identificador del equipo al que puede pertenecer el artículo.
    - Almacén: identificador del almacén en el que está contenido el artículo.
    - Notas: notas opcionales del artículo registrado.
    - Foto: foto del artículo.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)			No	*Ninguna*			Cambiar Eliminar Más
2	entidad	int(11)			No	*Ninguna*			Cambiar Eliminar Más
3	creador	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
4	nombre	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
5	codigoBarras	varchar(255)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
6	cantidad	int(11)			No	*Ninguna*			Cambiar Eliminar Más
7	tipoArticulo	int(11)			No	*Ninguna*			Cambiar Eliminar Más
8	talla	int(11)			No	*Ninguna*			Cambiar Eliminar Más
9	color	int(11)			No	*Ninguna*			Cambiar Eliminar Más
10	equipo	int(11)			No	0			Cambiar Eliminar Más
11	almacen	int(11)			No	*Ninguna*			Cambiar Eliminar Más
12	notas	text	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
13	foto	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más

Figura 3 – Tabla de stock en base de datos

- Histórico: cuando realicemos cualquier tipo de movimiento de stock en el almacén quedará registrado en esta tabla (ver figura 4), se forma por los siguientes campos:
    - Id: identificador del movimiento realizado
    - Stock: identificador del artículo al que le estamos aplicando la gestión
    - Entidad: identificador de la entidad que está realizando la gestión
    - CreadorStock: identificador del usuario que creó el artículo
    - NombreStock: nombre del artículo que se está gestionando
    - codigoBarras: serie numérica que representa un código de barras o QR asociado a un artículo.
    - Cantidad: número que puede representar la cantidad de artículos que entran o salen del almacén.
    - TipoArticulo: el tipo del artículo en cuestión.
    - Talla: la talla del artículo que se está gestionando
    - Color: el color del artículo que se está gestionando
    - Equipo: identificador del equipo al que pertenece el artículo.
    - Almacen: almacén al que se destina el artículo.
    - NotasStock: notas asociadas al artículo.
    - Foto: la foto del artículo.
    - Notas: notas asociadas a el movimiento para destacar cualquier cosa que al usuario pueda resultarle importante.
    - Tipo: el tipo de movimiento que estamos realizando, ya sea entrada o salida.
    - Creador: un identificador asociado al usuario que está realizando el movimiento.

- EquipoDestino: un identificador del equipo al que va dedicado el artículo en caso de que el movimiento realizado sea una salida.
- UsuarioDestino: un identificador del usuario al que va dedicado el artículo en caso de que el movimiento realizado sea una salida.
- Fecha: fecha en la que se realiza el movimiento.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)			No	*Ninguna*			Cambiar Eliminar Más
2	stock	int(11)			No	*Ninguna*	ID DEL STOCK		Cambiar Eliminar Más
3	entidad	int(11)			No	*Ninguna*			Cambiar Eliminar Más
4	creadorStock	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*	CREADOR DEL STOCK		Cambiar Eliminar Más
5	nombre	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
6	codigoBarras	varchar(255)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
7	cantidad	int(11)			No	*Ninguna*			Cambiar Eliminar Más
8	tipoArticulo	int(11)			No	*Ninguna*			Cambiar Eliminar Más
9	talla	int(11)			No	*Ninguna*			Cambiar Eliminar Más
10	color	int(11)			No	*Ninguna*			Cambiar Eliminar Más
11	equipo	int(11)			No	0			Cambiar Eliminar Más
12	almacen	int(11)			No	*Ninguna*			Cambiar Eliminar Más
13	notasStock	text	utf8mb4_unicode_ci		No	*Ninguna*	NOTAS DEL STOCK		Cambiar Eliminar Más
14	foto	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
15	notas	text	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
16	tipo	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*	ENTRADA, SALIDA, REGULACION		Cambiar Eliminar Más
17	creador	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
18	equipoDestino	int(11)			No	*Ninguna*			Cambiar Eliminar Más
19	usuarioDestino	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
20	fecha	datetime			No	*Ninguna*			Cambiar Eliminar Más

Figura 4 – Tabla de histórico en base de datos

- Talla: información de las tallas disponibles (ver figura 5) formada por:
  - Id: identificador de la talla.
  - Entidad: identificador de la entidad a la que pertenece.
  - Talla: nombre de la talla.
  - tipoArticulo: identificador para el tipo de artículo al que va destinada la talla en cuestión (ropa, zapatillas…).

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)			No	*Ninguna*			Cambiar Eliminar Más
2	entidad	int(11)			No	*Ninguna*			Cambiar Eliminar Más
3	talla	varchar(200)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más
4	tipoArticulo	int(11)			No	*Ninguna*			Cambiar Eliminar Más

Figura 5 – Tabla de talla en base de datos

- Color: información con los colores (ver figura 6) estructurada en base a:

- Id: identificador del color.
- Entidad: identificador de la entidad a la que pertenece.
- Color: nombre del color.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)			No	*Ninguna*			Cambiar Eliminar Más
2	entidad	int(11)			No	*Ninguna*			Cambiar Eliminar Más
3	color	varchar(255)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más

Figura 6 – Tabla de color en base de datos

- Tipo: los tipos de stock que tendremos en nuestro almacén (ver figura 7) formados por:
  - Id: identificador del tipo de artículo.
  - Nombre: nombre del tipo.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
1	id	int(11)			No	*Ninguna*			Cambiar Eliminar Más
2	nombre	varchar(255)	utf8mb4_unicode_ci		No	*Ninguna*			Cambiar Eliminar Más

Figura 7 – Tabla de tipo en base de datos

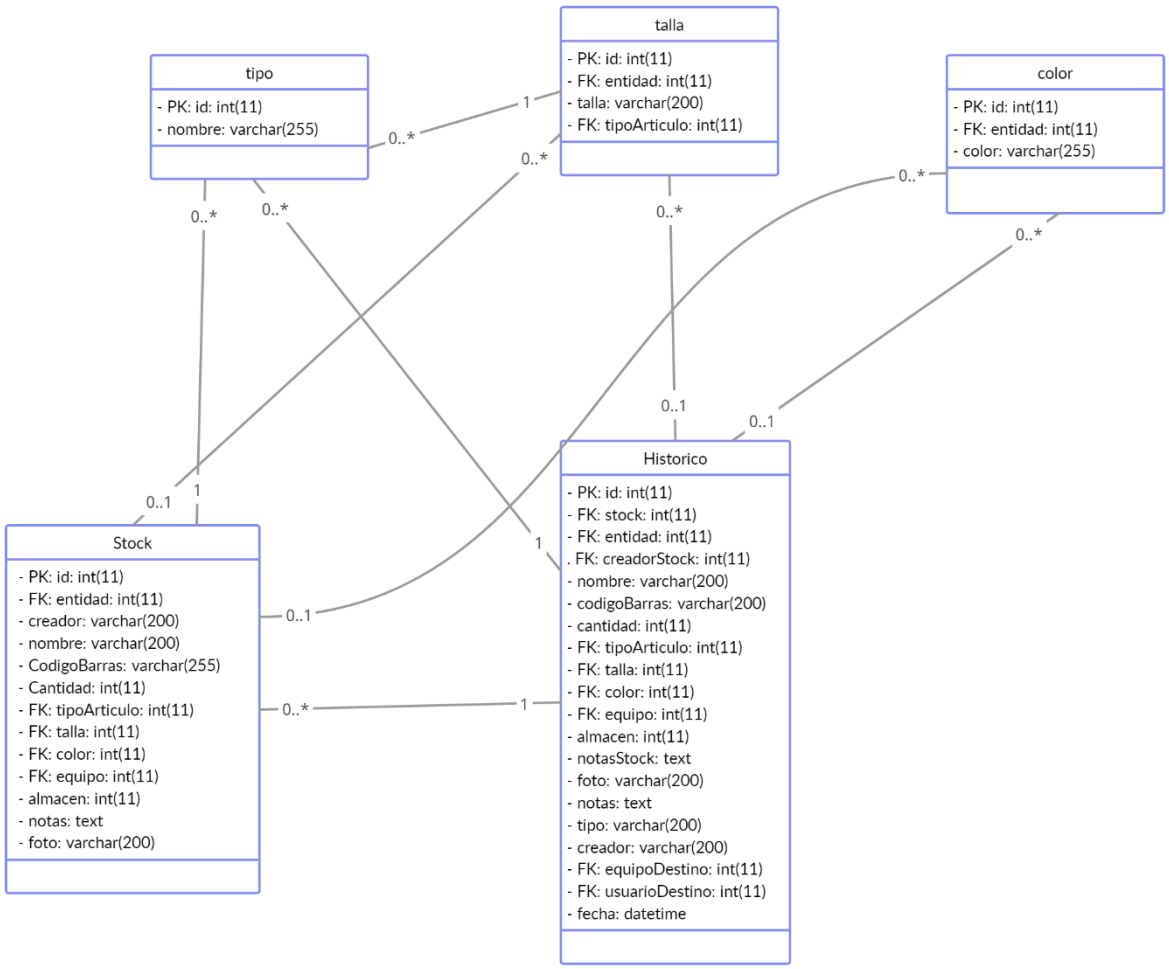

Figura 8 – Diagrama UML

En otro orden de cosas, es importante conocer que en nuestro proyecto habrá dos roles diferentes; administradores y *coaches* o entrenadores, ambos tienen acceso al módulo de la aplicación, pero con diferentes funciones y permisos.

Entrenador:

- Podrá loguearse en la aplicación y tener acceso al módulo almacén.
- Podrá cerrar la sesión.
- Podrá crear artículos nuevos dentro de la entidad a la que pertenezca.
- Podrá editar artículos ya existentes dentro de la entidad a la que pertenezca.
- Podrá realizar movimientos de entrada o salida de artículos que pertenezcan a su entidad.
- Podrá visualizar el histórico con los movimientos que haya hecho el mismo u otros entrenadore, que pertenezcan a la misma entidad.
- Podrá crear o editar tallas personalizadas para los artículos de su entidad.
- Podrá crear o editar colores personalizados para los artículos de su entidad.

- Podrá dejar notas en algún movimiento del histórico que pertenezca a su entidad si lo considera oportuno.

Administrador:

- Podrá loguearse en la aplicación y tener acceso al módulo almacén.
- Podrá cerrar la sesión.
- Podrá crear artículos nuevos dentro de cualquier entidad que esté registrada en la aplicación.
- Podrá editar artículos ya existentes dentro de cualquier entidad que esté registrada en la aplicación.
- Podrá borrar cualquier artículo de cualquier entidad que este registrada en la aplicación.
- Podrá realizar movimientos de entrada o salida de artículos dentro de cualquier entidad que esté registrada en la aplicación.
- Podrá visualizar el histórico con todos los movimientos de todas las entidades que estén registradas en la aplicación.
- Podrá crear, editar o borrar tallas personalizadas de artículos de cualquier entidad.
- Podrá crear, editar o borrar colores personalizados para los artículos de su entidad.
- Podrá dejar notas en algún movimiento del histórico de cualquier entidad si lo considera oportuno.

El administrador será el encargado de dar de alta a la entidad en la aplicación y de dar permiso para usar el módulo de almacén a los entrenadores que previamente hayan solicitado el acceso. Además, supervisarán todo el contenido que se registre en el proyecto y solo ellos tendrán potestad para eliminar cualquier tipo de contenido.

Los entrenadores, por su parte, harán uso del módulo almacén para gestionar lo que consideren conveniente en cada momento, haciendo uso de los casos de uso comentados arriba.

4.2 Arquitectura

Ahora que la estructura de datos y sus relaciones han sido clarificadas pasaremos a explicar su arquitectura.

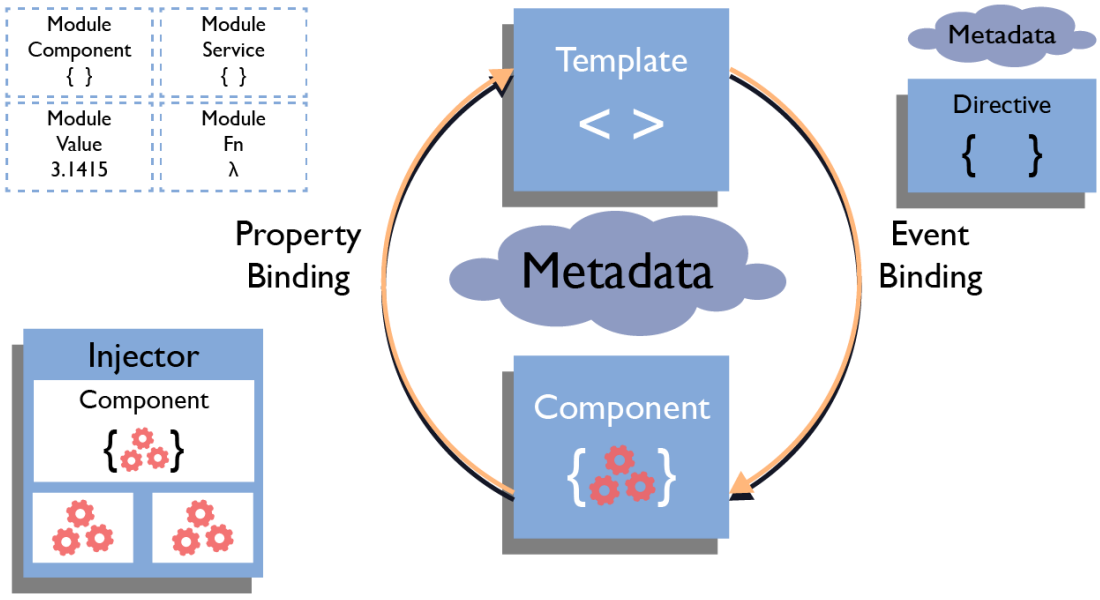

Figura 9 – Esquema de la arquitectura de Angular

Vamos a implementar nuestro proyecto utilizando el *framework* de Angular, que es el mismo con el que esta desarrollada el resto de la aplicación. Ahora para comprender el funcionamiento de nuestro módulo vamos a conocer la arquitectura de dicho *framework* y de cómo se relacionan sus elementos para dar forma a lo que veremos más adelante.

Para crear nuestra aplicación con Angular, generaremos *templates* con HTML y controlaremos estas mismos *templates* con lógica creada en nuestros componentes, que serán exportadas como clases. Así mismo, agregaremos lógica en nuestros servicios para manejar los datos que tendrá nuestro proyecto y finalmente "encapsularemos" nuestros componentes y servicios en módulos o *NgModules*.

Acto seguido, nuestro proyecto se iniciará mediante el *bootstraping* de nuestro *root module*, lo que provocará que Angular muestre el contenido en el navegador con el fin de estar preparados para interactuar con la aplicación.

En nuestro caso nosotros crearemos un archivo *almacen.module* que se encargará de contener: todos los componentes, servicios y otros archivos de código cuyo alcance definamos en nuestro módulo.

También definiremos componentes que son los que contendrán todos nuestros datos y nuestra lógica de aplicación. Para comprender más adelante los ejemplos de código que explicaremos conviene aclarar un poco cómo se relacionan los componentes con las *templates* en Angular.

Una *template* no es más que una plantilla de HTML que se basa en una mezcla de HTML con marcas personalizadas de Angular. La *template* nos permitirá definir la vista de un componente y bajo ciertas directivas conectar los datos que nos está mostrando dicha vista con los datos del archivo que contiene la lógica del componente bajo una relación denominada *data binding*.

El *data binding* lo utilizaremos a lo largo de todo el proyecto y su propósito no es más que reflejar en la *template* cualquier cambio que se produzca en las variables del componente, ya sea debido a una carga de datos o a una modificación de estos. Gracias a las marcas personalizadas de Angular y al *data binding* podremos ahorrarnos colocar código PHP sobre nuestras *templates* y conseguir un código más limpio y cohesionado.

Durante todo el desarrollo del proyecto organizaremos tanto nuestros componentes como nuestros PHP en distintos directorios, por ejemplo, para la parte de *frontend* los ubicaremos en la raíz del proyecto dentro de la carpeta *app* y para el *backend* se ubicarán en una carpeta denominada 'gestión' dentro del servidor (ver figura 10).

Figura 10 – Ubicación del backend visto desde fileZilla

Los componentes y las vistas que utilizaremos para nuestro proyecto se organizarán de acuerdo con el resto de la aplicación para mantener un formato acorde y unificado (ver figura 11). Cada componente estará formado por cuatro archivos (ver figura 12) que son:

- **Archivo.css:** contiene todas las clases de estilos que son únicas al componente y accesibles por su *template*.
- **Archivo.html:** es el que contiene la plantilla HTML, es decir, la *template* de nuestro componente que nos proporcionará la vista que observaremos por navegador.
- **Archivo.spec.ts:** se crean automáticamente y sirven para realizar un *testing* avanzado del funcionamiento y las dependencias de los componentes. Nosotros personalmente no los usaremos en nuestro proyecto, pero está bien dejarlos creados por si se tienen que realizar testeos a futuro.
- **Archivo.ts:** contiene la lógica del componente y es el que proporcionará los datos/variables a nuestra plantilla HTML.

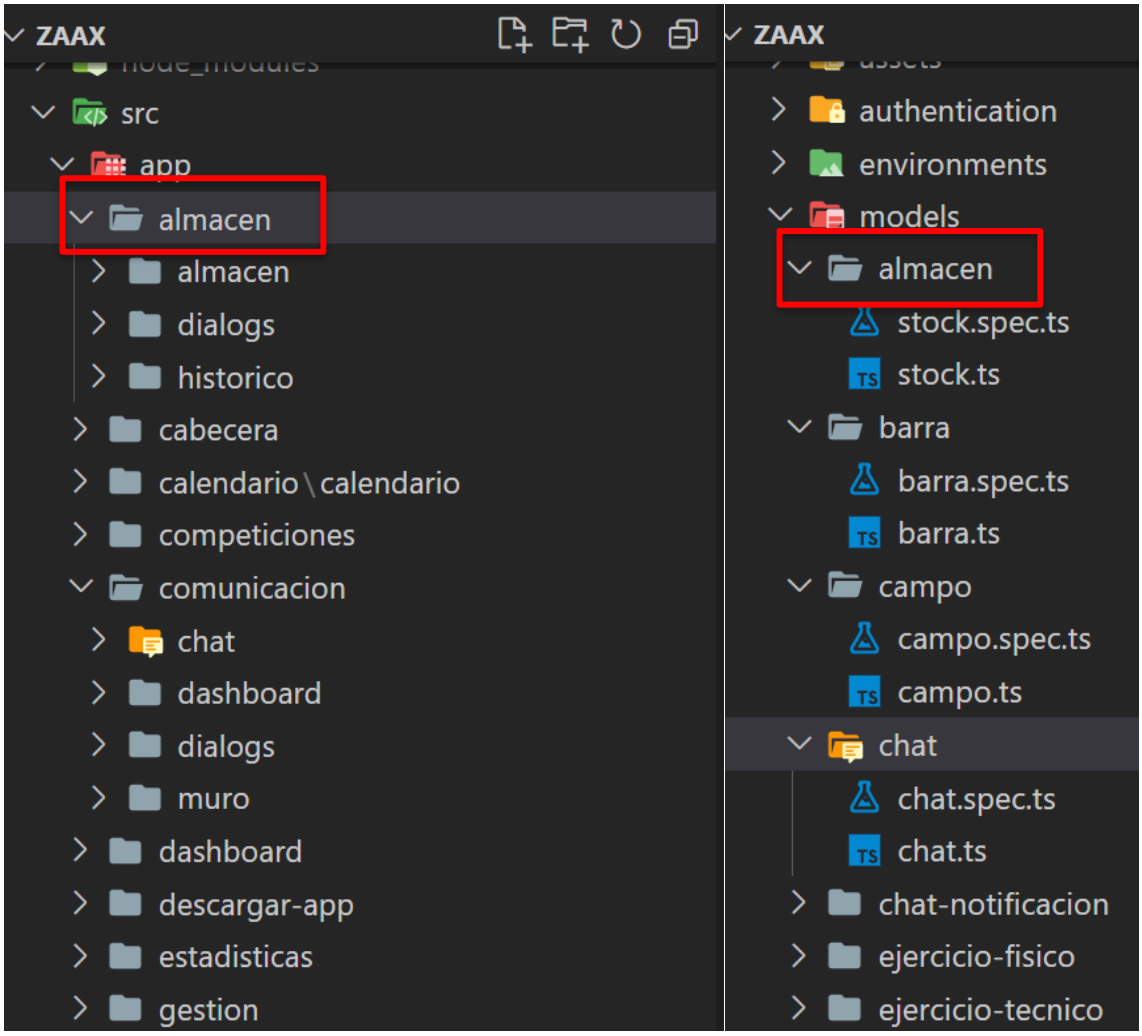

Figura 11 – Ejemplo de organización de la aplicación

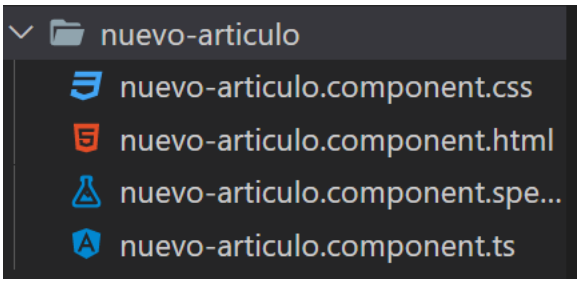

Figura 12 – Ejemplo de organización de un componente

5. Desarrollo de la solución

En este punto detallaremos las herramientas que hemos utilizado en nuestro proyecto y echaremos un vistazo al código que hemos desarrollado.

5.1 Contexto tecnológico

Ahora vamos a detallar las herramientas y/o lenguajes que hemos utilizado para la implantación de nuestro proyecto en la aplicación, después seguiremos con algo de código que nos ayudara a entender las distintas funcionalidades de *backend* y *frontend.*

5.1.1 Angular

Angular es un *framework* de código abierto para aplicaciones web desarrollado en typescript cuyo objetivo es aumentar el uso de las aplicaciones que se basan en el uso del navegador con capacidad de Modelo Vista Controlador (MVC).

Con Angular podemos crear y mantener aplicaciones web de una sola página, lo que nos facilita en varios aspectos el desarrollo, la organización, el tiempo, y el esfuerzo que debemos emplear a la hora de implementar o probar código.

5.1.2 Typescript

Typescript es un lenguaje de código que extiende la sintaxis de Javascript y que se utiliza principalmente para desarrollar aplicaciones que se pueden ejecutar tanto en *frontend* como en *backend*.

Typescript se basa en un lenguaje estático, por lo que al contrario que javascript la verificación de tipos se realizará en tiempo de compilación, además, este lenguaje da soporte a módulos, es fiable y simple de refactorizar. Es por todo esto por lo que será el lenguaje predominante en nuestro proyecto.

5.1.3 HTML5

HTML5 es la quinta revisión del lenguaje HTML que, si bien incorpora mejoras y novedades como nuevos tipos de etiquetas o algunas mejoras de formularios, no lo usaremos en su totalidad, sino que lo mezclaremos con las directivas y las marcas personalizadas de angular.

5.1.4 CSS

CSS (*Cascading Style Sheets*) es el lenguaje que utilizaremos para dar forma a el diseño y presentación de nuestro proyecto.

Con CSS crearemos reglas para decirle a nuestro sitio web como queremos mostrar la información. Para nuestro proyecto crearemos clases únicas, pero también utilizaremos algunas globales a toda la aplicación.

5.1.5 MySQL

Al tener en nuestro proyecto una base de datos relacional utilizaremos MySQL como sistema de gestión y administración. Ahora mismo se trata de uno de los sistemas más usados debido en parte porque su código es abierto, lo que se traduce como una ventaja a la hora de proporcionar soluciones personalizadas para muchos ámbitos empresariales.

5.1.6 FileZilla

FileZilla es un software que nos permitirá conectarnos a nuestro servidor mediante el protocolo FTP, con el fin de poder consultar, adquirir o modificar contenido que se aloje en nuestro servidor.

FileZilla es totalmente gratuito y nos permite tener una conexión constante con todos los directorios de datos que tengamos en el servidor, tiene una transferencia de datos bastante rápida e incluso nos proporciona una navegación sincronizada para poder movernos entre directorios sin perder demasiado tiempo en el proceso.

5.1.7 VisualStudioCode

VisualStudioCode es el editor de código que utilizaremos durante todo el proyecto, pues está muy orientado a desarrollos web y *javascript*, aparte de permitirnos instalar múltiples extensiones que nos facilitarán mucho la organización y programación de código.

5.1.8 Otras herramientas

Además de todo lo anterior aplicaremos *plugins* externos para dar funcionalidad a ciertos aspectos relevantes de nuestro proyecto como el lector de código de barras o el *cropper* para poder subir imágenes de nuestros artículos. También utilizaremos iconos importados de páginas como FontAwesome o FeatherIcons.

5.2 Ejemplos de código

Después de explicar todas las herramientas con las que desarrollaremos nuestro proyecto, introduciremos algunos fragmentos de código que trataran varios de los aspectos relacionados con la sintaxis y el entorno de trabajo.

Figura 13 – Ejemplo del Ngmodule de almacén.module.ts

Podemos apreciar en la figura 13 el código del archivo *almacen.module.ts*, correspondiente a nuestro proyecto. Podemos observar que nuestro módulo consta de tres propiedades que son:

- ***Declarations*:** Contiene todo el set de directivas, componentes y pipes que pertenecen a este módulo en concreto, es decir, aquí estarán declarados única y exclusivamente todos los componentes que hayan sido creados dentro de el fichero de trabajo almacén.
- ***Imports*:** En esta propiedad vendrán declarados todos aquellos módulos que estén situados fuera del fichero de trabajo de almacén y cuyas funcionalidades o componentes queramos utilizar en nuestro proyecto, por ejemplo, el módulo

sharedModule contiene pop-ups de confirmación, pipes para traducir texto y componentes que nos vendrán bien en nuestro desarrollo, mientras que *CommonModule* contiene las directivas y marcas personalizadas que usaremos en nuestros *templates.*

- ***Exports*:** Aquí se declararán todos los componentes que podrán ser utilizados en las templates de otros componentes que formen parte de otro módulo que esté importando el nuestro, como, por ejemplo, nosotros podemos utilizar el pipe de traducir el texto debido a que estamos importando el módulo *SharedModule* y este a su vez tiene el pipe declarado en su propiedad *exports*.

A continuación, explicaremos cómo está estructurado el código que se encarga de gestionar las rutas en nuestro proyecto.

```
{ path: 'estadisticas', pathMatch: FULL, component: DashboardEstadisticasComponent, canActivate: AU
  data: { title: 'Estadisticas', modulos: ['ESTADISTICAS'] } },

{ path: 'ejercicios-tecnicos', pathMatch: FULL, component: EjerciciosTecnicosComponent, canActivate:
  data: { title: 'Ejercicios Técnicos', permissions: ENTRENADOR, modulos: ['TEST TECNICO'] } },

{ path: 'almacen', pathMatch: FULL, component: AlmacenComponent, canActivate: AUTH_AND_PERM,
  data: { title: 'Almacen', permissions: ENTRENADOR, modulos: ['ALMACEN'] } },

{ path: 'historico', pathMatch: FULL, component: HistoricoComponent, canActivate: AUTH_AND_PERM,
  data: { title: 'Historico', permissions: ENTRENADOR, modulos: ['ALMACEN'] } },

{ path: 'planes-tecnicos-calendario', pathMatch: FULL, component: PlanesTecnicosCalendarioComponent,
  data: { title: 'Calendario Ejercicios Técnicos', permissions: ENTRENADOR, modulos: ['TEST TECNICO'

{ path: 'planes-tecnicos-lista', pathMatch: FULL, component: PlanesTecnicosListaComponent, canActiva
  data: { title: 'Lista Ejercicios Técnicos', permissions: ENTRENADOR, modulos: ['TEST TECNICO'] } }

{ path: 'ejercicios-fisicos', pathMatch: FULL, component: EjerciciosFisicosComponent, canActivate: A
  data: { title: 'Ejercicios Físicos', permissions: ENTRENADOR, modulos: ['TEST FISICO'] } },

{ path: 'planes-fisicos-calendario', pathMatch: FULL, component: PlanesFisicosCalendarioComponent, c
  data: { title: 'Calendario Ejercicios Físicos', permissions: ENTRENADOR, modulos: ['TEST FISICO']

{ path: 'planes-fisicos-lista', pathMatch: FULL, component: PlanesFisicosListaComponent, canActivate
  data: { title: 'Lista Ejercicios Físicos', permissions: ENTRENADOR, modulos: ['TEST FISICO'] } },
```

Figura 14 – Fragmento de app.routing

Como podemos observar en la figura 14 se muestran varias líneas de código, cada una con un *path* y una serie de directivas que representan varios de los aspectos que podemos configurar para cada ruta.

El *path* representa el nombre de la ruta que va a estar asociada a la carga de un componente dado, en nuestro caso podemos observar como aparecen las rutas ‘almacen’ y ‘historico’ y están asociadas a los componentes *AlmacenComponent* y *HistóricoComponent* respectivamente. Por otra parte, también tenemos la directiva *pathMatch* que en nuestras rutas está declarada como *FULL* lo que quiere decir que el enrutador siempre buscará una coincidencia con la ruta entera.

La directiva *canActivate* se utiliza para saber si el usuario tiene permiso para cargar dicha ruta, y para comprobarlo le pasamos la constante AUTH_AND_PERM que contiene un array con referencias a dos archivos de código que se encargarán tanto de verificar si el *token* del usuario coincide con la ruta a cargar o de si el usuario contiene el rol correspondiente y por tanto el permiso para ver el contenido de la ruta. Por último, la *data* que le pasamos a la ruta consta del título del componente que le queremos dar, el rol que tiene permisos para cargar esa ruta y el nombre del módulo al que hacemos referencia.

El contenido que cargan las rutas está referenciado en el *index.html* de la aplicación (ver figura 15), que está representado con la marca *<app-root></app-root>*, que hace alusión al componente de *routing* de angular que hemos configurado.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Zaax</title>
  <base href="./">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta http-equiv="Cache-Control" content="no-cache, no-store, must-revalidate">
  <meta http-equiv="Pragma" content="no-cache">
  <meta http-equiv="Expires" content="0">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500&display=swap" rel="stylesheet"
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
</head>
<body class="mat-typography">
  <app-root></app-root>
</body>
</html>
```

Figura 15 – Index.html de la aplicación

Por otra parte, en nuestro proyecto también utilizaremos unas clases denominadas *services*, cuyo propósito es servirnos de enlace entre el *frontend* y *backend*. En estas clases podremos definir funciones que nos servirán para consultar, modificar, crear o borrar contenido de nuestra base de datos, y haciendo uso de la herencia podremos invocarlas en nuestros componentes. A continuación, en la figura 16 tenemos un ejemplo del *service* de almacén, en el que se puede observar que:

- En el constructor heredaremos las clases *httpClient* y *injector* para poder usar sus funciones y atributos.
- Implementaremos una función denominada *sendPost* que será genérica y nos servirá para usarla en todos los métodos que declaremos en el service. Esta función consta de:

  - Una variable *headers* de tipo *headers* cuyo valor vendrá definido por el tipo de contenido que enviaremos (en este caso de tipo *Json*) y por el contenido que aceptaremos.
  - Un *Json* que le pasamos como parámetro a la función que incluirá un código de operación y unos datos que el *php* correspondiente manejará.
  - Un método *post* que se encargará de enviar el *Json* al php correspondiente y nos devolverá el resultado recibido de la base de datos. Además, cuenta con un manejador de errores muy simple que nos llevará al *logout* en caso de que nuestro token de usuario no esté validado.
- Una función ejemplo denominada 'nuevoColor' que hace uso de la *sendPost* antes mencionada, pasándole un código de operación y un dato denominado color que en este caso es un objeto con varias propiedades.

```
export class AlmacenService {

  auth: any;
  constructor(private http: HttpClient, private injector: Injector) {
    this.auth = this.injector.get<AuthenticationService>(AuthenticationService)
  }

  private sendPost(bodyJSON) {

    let headers = new HttpHeaders().set('Content-Type','application/x-www-form-urlencoded');
    headers.append('Accept', 'application/json');
    headers.append('Content-Type', 'application/json');

    bodyJSON['user'] = environment.user;
    let bodyString = JSON.stringify(bodyJSON);       You, 13 months ago • añadidos selects a dialogs co

    return this.http.post( environment.inc + 'almacen.php', bodyString, { headers: headers } )
      .pipe( map( res => { if(res=='ERROR_TOKEN'){this.auth.logout();} return res; } ) );
  }

  /**
   * Nuevo color.
   *
   * @param color Json con (Color String,Entidad int),Inserta un nuevo color en bd
   * @returns
   */
   nuevoColor(color){
    return this.sendPost({ Op: 'nuevoColor',
      color: color
    });
  }
```

Figura 16 – ejemplo de código de service de almacen

Como hemos comentado anteriormente, podemos utilizar un *service* de Angular en uno o varios componentes haciendo uso de la herencia, lo que nos permitirá usar sus métodos en nuestros archivos de typescript para poder acceder al backend de la aplicación con la finalidad de obtener y manejar datos. Si echamos un vistazo a la figura 17 declararemos una instancia de nuestro servicio de almacén en el constructor y después usaremos su método *get_listado_articulos* para obtener la lista de artículos de una entidad con la intención de guardarla en una variable denominada 'articulos' que utilizaremos en nuestra plantilla HTML.

Si observamos la figura 18, dentro de nuestra plantilla HTML podremos referenciar las variables de nuestro *typescript* encapsulando nuestra variable bajo esta nomenclatura '{{}}', además de poder aplicarles directivas personalizadas de Angular tales como:

- **ngFor***: recorrerá una variable que sea de tipo array y repetirá todo el contenido que se encuentre dentro de esta directiva tantas veces como elementos tenga la variable. Esto nos es muy útil para recorrer la variable 'articulos' y asignarle a cada artículo sus estilos y variables y no depender de incrustar PHP en el HTML.
- **ngIf***: solo se mostrará el contenido que se encuentre dentro del contenedor marcado con esta directiva si la condición declarada devuelve *true*. Esta marca nos viene muy bien para por ejemplo esconder los demás artículos cuando estemos escribiendo algo en el buscador o cuando un campo opcional no tenga valor.
- **ngModel**: esta marca nos permitirá enlazar una variable de un *input* o un *select* a cualquier variable de nuestro *typescript*, lo que nos ayudará a la hora de comprobar campos en un formulario o para rellenar los mismos con un valor predeterminado en caso de edición o de tener un valor fijo.
- **ngModelChange**: cada vez que se detecte un cambio en el modelo de datos se lanzará la directiva que se encuentre declarada dentro de esta marca que, o bien puede ser un método que se referencie del typescript o una asignación de variables. Con respecto a esta marca, nos serviremos de ella para programarnos un buscador donde el usuario podrá escribir el nombre del artículo que quiera encontrar y el método asignado se encargue de asignar un booleano a cada artículo con la finalidad de esconder todos aquellos que no coincidan con el criterio de búsqueda.

```
constructor(
  private almacenService:AlmacenService,
  private router:Router,
  public entidadesService: EntidadesService
) {

  this.textoDeBusqueda="";
  this.todosHidden=false;
  this.hayDatos=true;
  this.roles = environment.user.roles;
  this.entidadUser = environment.user.entidad;
  this.entidad=0;
}

ngOnInit(): void {
  this.cargar_entidades();
}

cargar_Datos() {
  let entidadaux = {
    id: this.entidad
  }

  this.almacenService.get_listado_articulos(entidadaux).subscribe(res=>{
    this.articulos=res;
    console.log(this.articulos);      You, 11 months ago • corrección campo
  });
}
```

Figura 17 – fragmento de código de almacen.ts

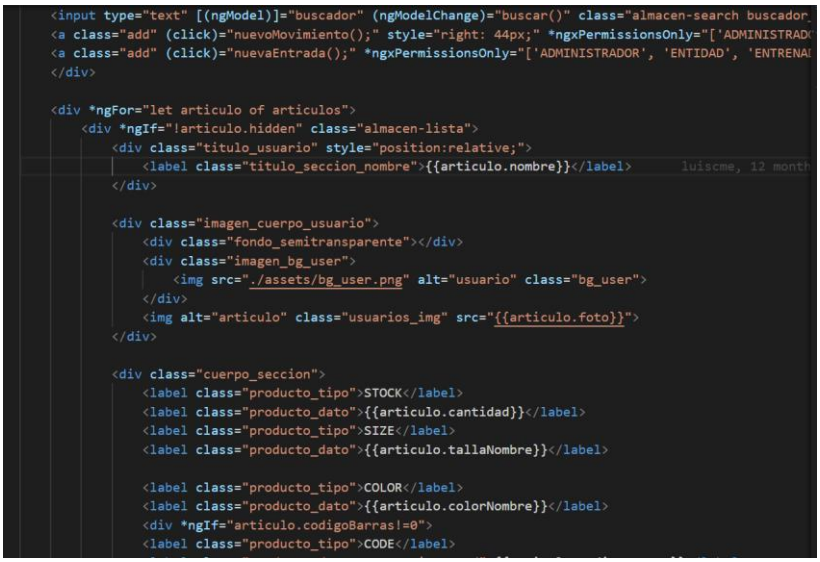

Figura 18 – fragmento de Código de almacen.html

Por otro lado, previamente en esta memoria habíamos hablado de que uno de nuestros objetivos consistía en tener nuestra aplicación traducida tanto al inglés como al español. Para lograrlo nos ayudaremos de una herramienta que nos permitirá transformar la información presentada en pantalla denominada *pipe*. Programaremos nuestro pipe de tal manera que reciba cualquier texto posible desde nuestras plantillas HTML y dependiendo del idioma seleccionado lo coteje con un diccionario donde tendremos todas las palabras de nuestra aplicación. En la figura 19 podemos observar que el diccionario no es más que un objeto *json* alojado en la carpeta 'i18n' (nombre estandarizado que se utiliza para hacer referencia a los archivos de idioma relevantes por el identificador de idioma del sistema o del navegador) que en este caso contiene las coincidencias del inglés al español. Por otra parte, si observamos el código del pipe en la figura 20, para que el *pipe* pueda acceder al diccionario necesitará la ayuda de un *service* de traducción que se encargara de realizar la lógica de la comparación de las palabras proporcionadas con las del diccionario.

```
export const es = {
    /* warehouse.page */
    'Articles': 'Artículos',
    'No added articles': 'No hay artículos añadidos',
    'Stock': 'Stock',
    'Size': 'Talla',
    'Color': 'Color',
    'Code': 'Código',
    'Movement': 'Movimiento',
    'Movement type': 'Tipo de movimiento',
    'OUT': 'SALIDA',
    'IN': 'ENTRADA',
    'Team': 'Equipo',
    'Notes': 'Notas',
    'New article': 'Nuevo artículo',
    'Product name': 'Nombre del producto',
    'Movement type': 'Tipo de movimiento',
    'Clothing': 'Ropa',
    'Shoes': 'Zapatos',
    'Others': 'Otros',
    'Select size': 'Seleccionar tamaño',
    'Select color': 'Seleccionar color',      You, now •
    /* page user-settings */
```

Figura 19 – Fragmento de código de diccionario inglés a español

```
@Pipe({
  name: 'translate',
  pure: false,
})
export class TranslatePipe implements PipeTransform {
  translationService: TranslationService
  constructor() {
    this.translationService = new TranslationService()
  }

  /**llamada a service para traducir*/
  transform(value: any): string {
    // console.log(value, ":", this.translationService.translate(value))
    return this.translationService.translate(value) || value
  }      You, 2 months ago • translate done user done
}
```

Figura 20 – Código de pipe de traducción

Por último, cabe destacar que los colores de cada entidad serán variables globales que utilizaremos en nuestros estilos CSS de cada componente para proporcionarle a cada entidad un color personalizado a su gusto. Los colores de cada entidad se asignan cuando un usuario inicia sesión y se guardan en variables para su uso en las hojas de estilos (ver figura 21).

```
.nueva_tarjeta_icon > fa-icon:hover {
    background-color: var(--secundario);
    color: var(--primario);          dmartin9
}
.nueva_tarjeta_contenido {
    filter: opacity(0.2);
}
```

Figura 21 – ejemplo código CSS de variables globales

6. Prueba de concepto

Como ya hemos comentado anteriormente en la aplicación hay tres tipos de usuarios, los administradores, los entrenadores y los jugadores. En nuestro módulo almacén solamente tendrán acceso los administradores y entrenadores, mientras que los jugadores quedarán relegados a otros módulos de la aplicación.

En este apartado pasaremos a explicar las distintas vistas de nuestro proyecto tanto en la vista web como en la vista móvil, cuáles son sus funcionalidades y cómo se diferencian entre sí dependiendo de si el usuario se trata de un administrador o de un entrenador.

6.1 Vista web

En primer lugar, la página de *login* (ver figura 22) será común a los tres roles, se podrá acceder a la aplicación mediante un nombre de usuario y una contraseña. Cabe destacar que los registros de los usuarios nuevos los realizan los administradores. Al iniciar sesión en la aplicación los usuarios se encontrarán con la página de *home* que a su vez contiene un menú lateral a la izquierda donde se muestran los distintos módulos que contiene la aplicación entre los cuales se encuentra nuestro proyecto (ver figura 16).

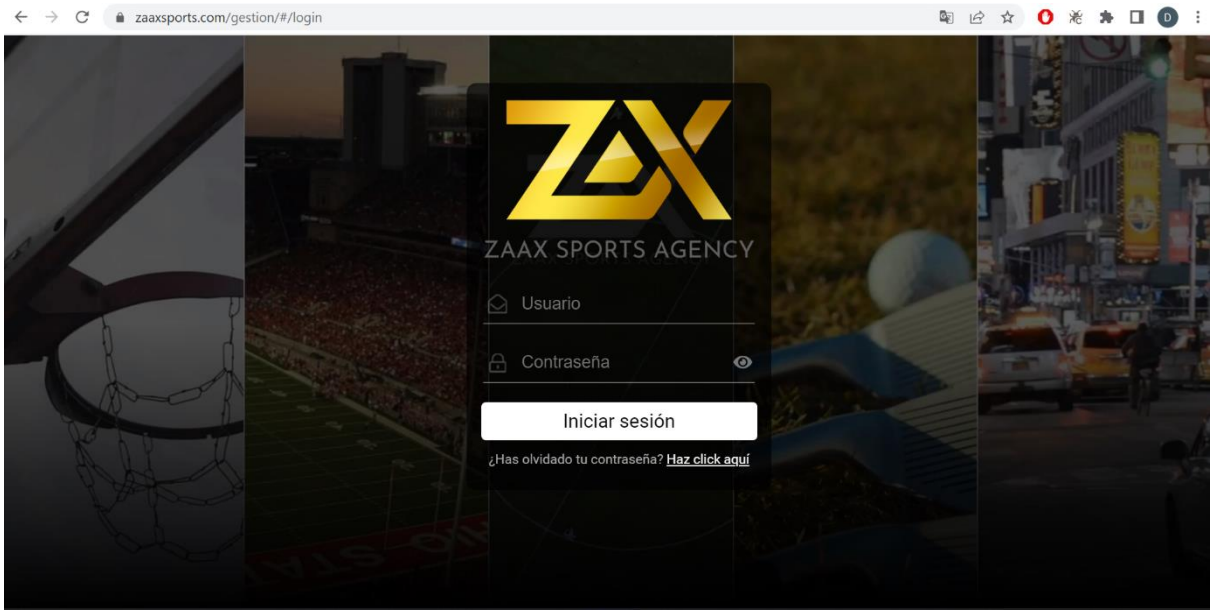

Figura 22 – página de login

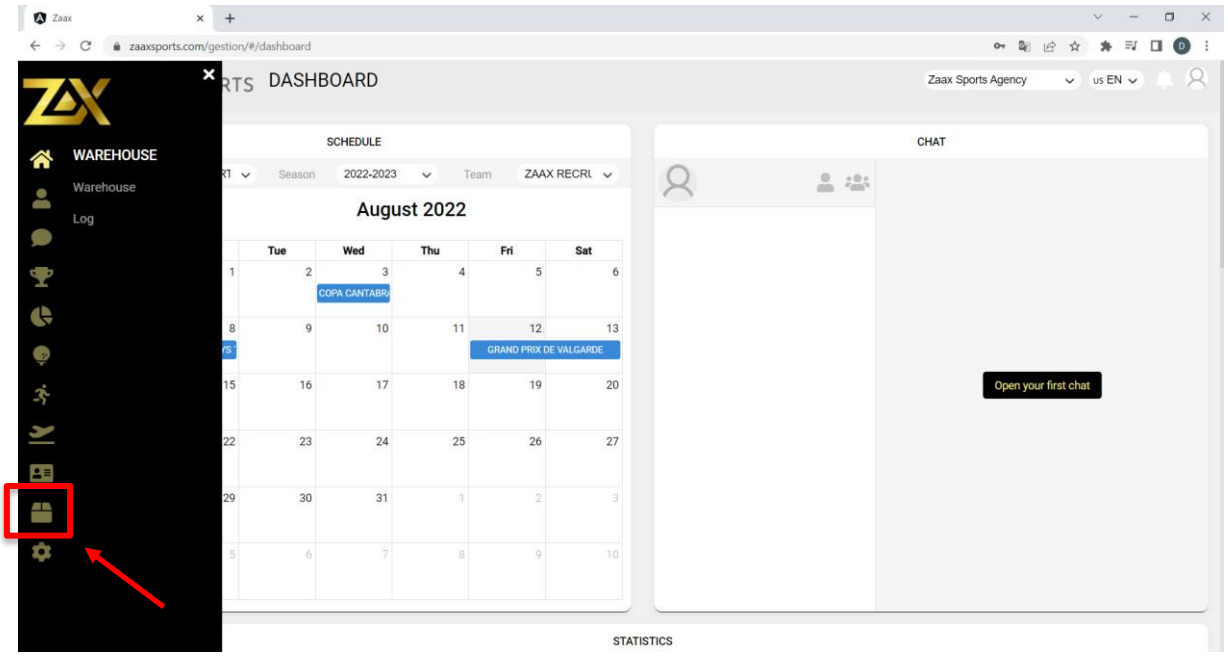

Figura 23 – página de home/principal

En la figura 23 podemos observar cómo se encuentra resaltado el icono del menú lateral que representa nuestro módulo. Cuando situemos nuestro cursor sobre él se nos desplegará un submenú con dos apartados, almacén e histórico, dependiendo del rol que tenga el usuario cambiarán ligeramente las vistas que nos encontraremos.

6.1.1 Administrador

Si clicamos en 'almacén' nos llevará a la lista de artículos de la entidad, en el caso de administradores tendremos que seleccionar una entidad para visualizar su lista de inventario mientras que los entrenadores visualizarán la lista directamente con la entidad a la que estén asociados.

Si observamos la figura 24, para seleccionar las distintas entidades se ha dispuesto un selector en la cabecera de la ventana al lado del menú lateral que dependiendo de su valor nos mostrará la lista de inventario de la entidad seleccionada. Por otra parte, también dispondremos de otro selector en la parte superior derecha que nos permitirá ver como quedan tanto los colores como el logo de cada universidad para testeo de estilos (ver figura 25) como, por ejemplo, en caso de que el contraste de colores de la identidad seleccionada no sea apropiado.

También dispondremos de un botón en cada artículo (icono de la papelera) para borrar el artículo que deseemos, y que al hacer clic en él nos abrirá un dialogo de confirmación (ver figura 26) para que aceptemos o cancelemos la petición en caso de habernos equivocado. Aunque si bien es cierto que tenemos la posibilidad de borrar cualquier artículo la idea es no hacerlo a no ser que sea estrictamente necesario, pues lo que queremos es mantener un histórico

de absolutamente todo lo que haya pasado por el almacén tanto si hay existencias como si no, o como si el artículo en cuestión ha sido descatalogado.

Por otra parte, si clicamos en 'histórico' nos llevará a una vista donde se encuentran registrados todos los movimientos hasta el momento (ver figura 27), representados todos en una tabla con varios campos informativos que son: el tipo de movimiento aplicado, el nombre del artículo, la cantidad, el tipo del artículo, la talla, el color si tiene, el equipo y el jugador al que va dirigido en caso de ser salida de producto, el almacén al que va destinado, la fecha de realización del movimiento y las notas si las tiene. Si volvemos a observar la figura también podemos apreciar los selectores de entidad, puesto que al ser administradores tenemos acceso a todos los históricos de las universidades.

Por último, cabe destacar que el administrador también puede realizar movimientos de entrada o salida de stock, así como la creación de cualquier tipo de artículo, pero este proceso nos lo reservamos para explicarlo en la parte del rol de entrenador.

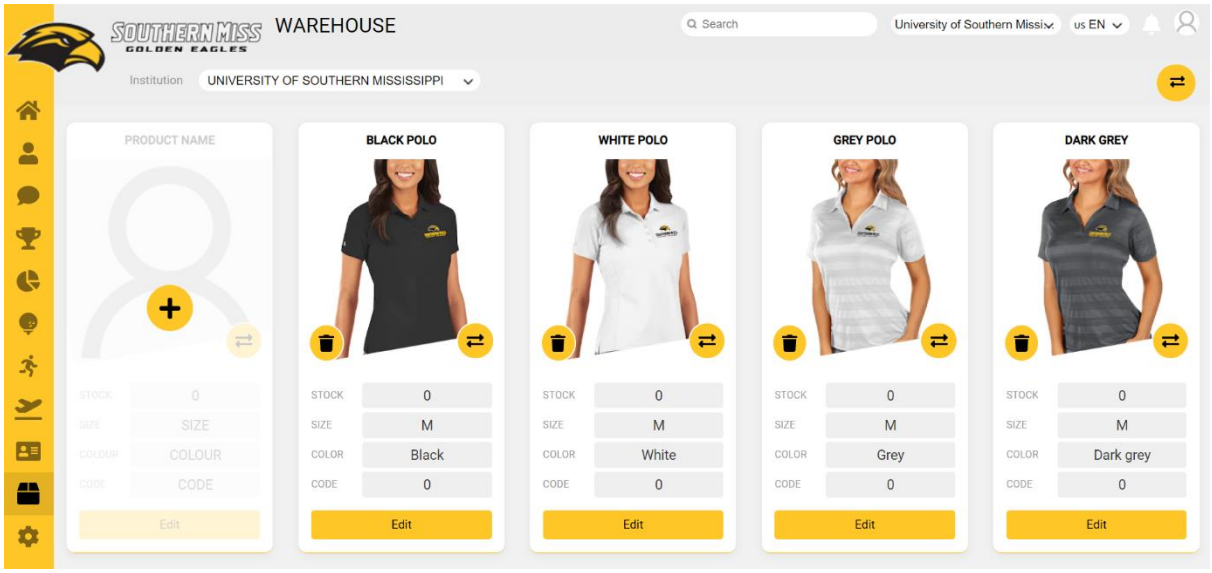

Figura 24 – página de lista de artículos de almacén

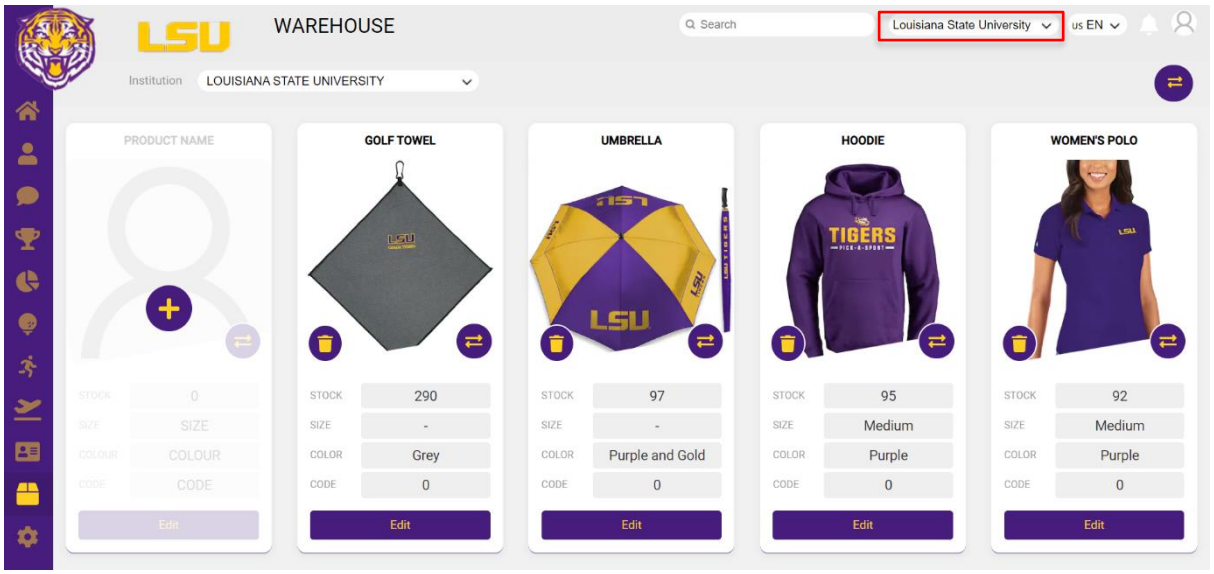

Figura 25 – ejemplo de aplicación de estilos de una entidad

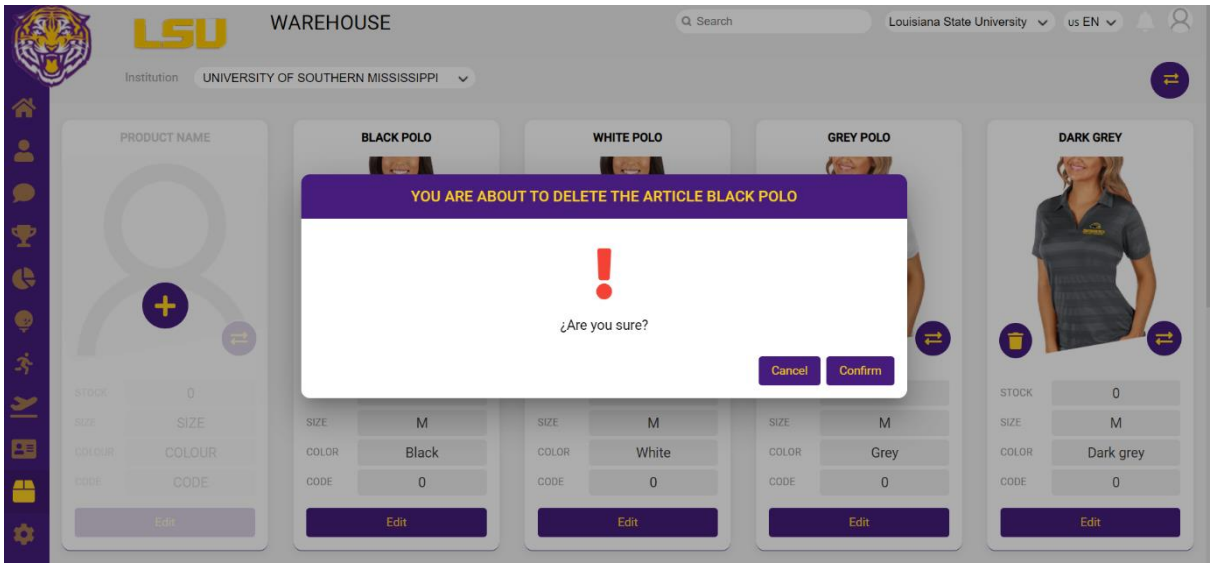

Figura 26 – diálogo de confirmación para el borrado de un artículo

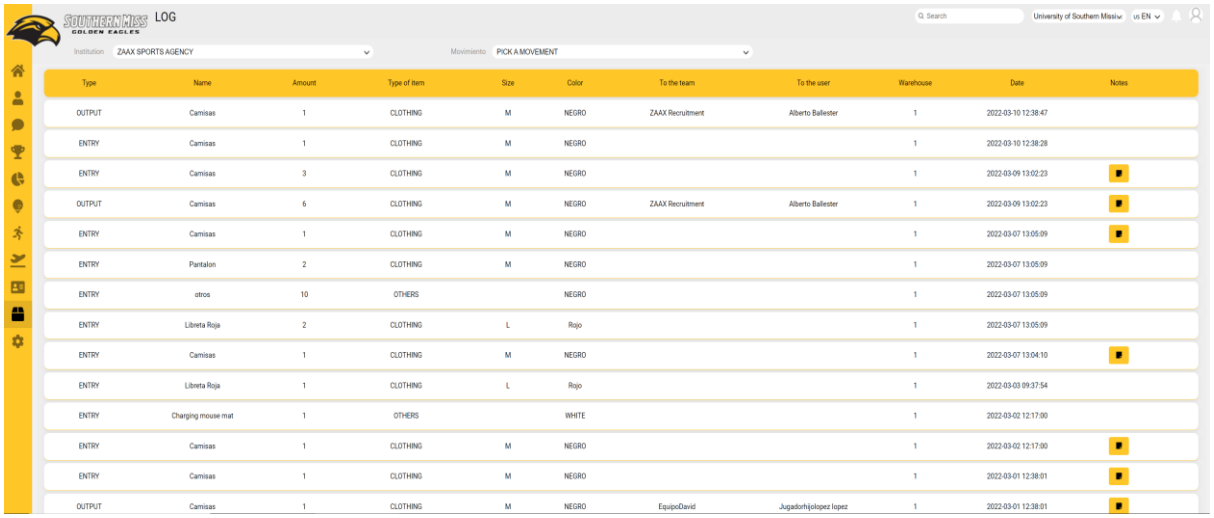

Figura 27 – vista del histórico de movimientos

6.1.2 Entrenador

La vista de almacén del entrenador es muy parecida a la del administrador solo que sin los selectores de entidades y los botones para borrar artículos.

En primer lugar, si observamos la figura 24, un entrenador puede crear un artículo haciendo clic en la primera *card* transparente de la izquierda en la lista de artículos al icono del '+'.Al hacerlo se desplegará un dialogo(ver figura 28) con distintos campos a rellenar que son: el nombre del artículo a crear como campo obligatorio; el tipo del artículo, como campo obligatorio, a elegir entre ropa, calzado u otros; el código de barras o código general que tenga dicho artículo como campo opcional; el color como campo opcional; la foto como campo obligatorio; la talla como campo obligatorio; el almacén al que va destinado como campo opcional y las notas opcionales que se quieran añadir. Cabe destacar que, si seleccionamos el tipo 'otros', el campo del formulario del dialogo de la talla no se mostrará puesto que un palo de golf por ejemplo no tendría talla como tal.

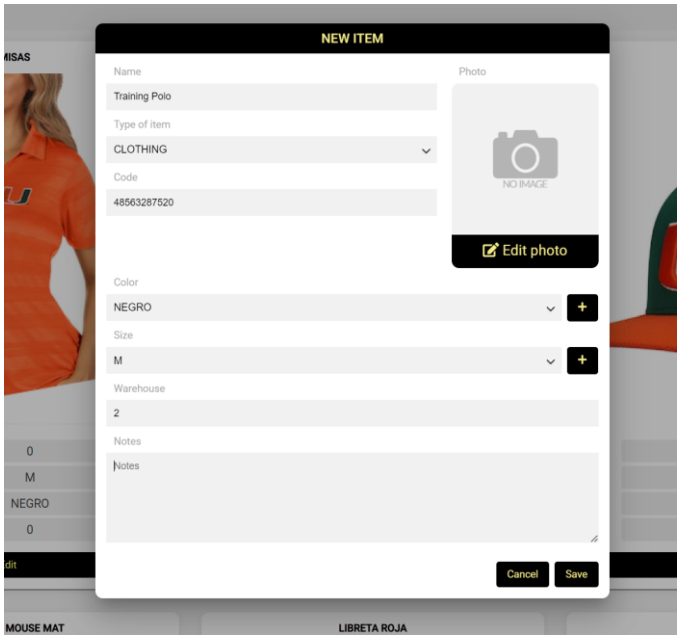

Figura 28 – dialogo de creación de articulo

Para subir una foto de nuestro nuevo artículo clicaremos en el botón de 'editar foto', que nos abrirá otro dialogo para subir una foto que podremos ajustar gracias a un *cropper* integrado en el mismo diálogo (ver figura 29).

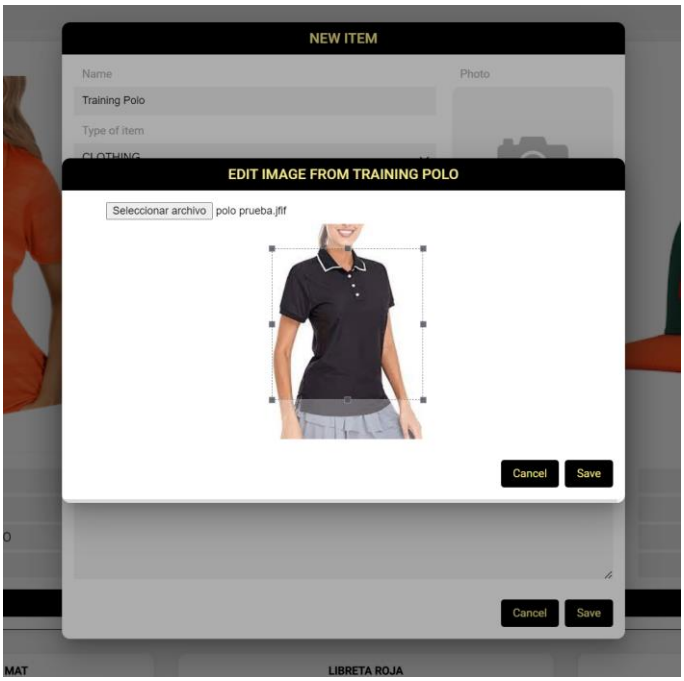

Figura 29 – cropper para subir una imagen

Por otra parte, para seleccionar una talla o un color bastaría con hacer clic en el selector y seleccionar cualquiera de la lista desplegada. En caso de no tener tallas o colores registrados podemos crear nuevos clicando en el icono del '+' que se encuentra a la derecha de estos selectores. Al hacerlo se nos abrirá un dialogo nuevo para que introduzcamos el nombre que queramos para la nueva talla o color (ver figura 30) y en el caso de la talla le especificaremos también si dicha talla va dirigida a el tipo 'ropa' o 'calzado'.

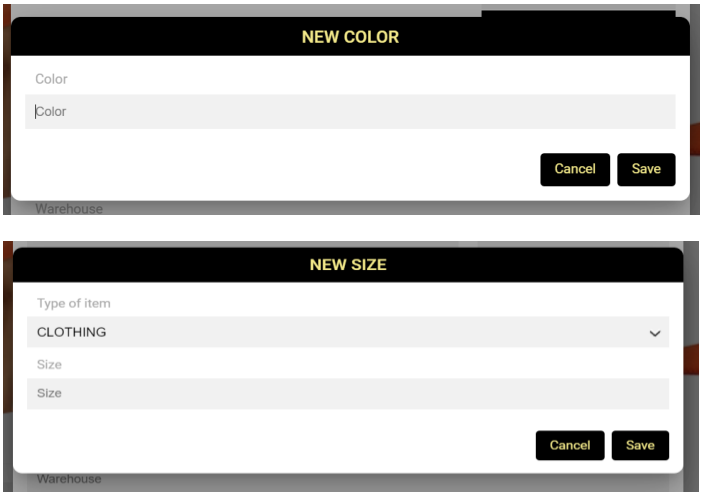

Figura 30 – diálogos de creación de tallas y colores

Cuando hayamos rellenado todos los campos obligatorios de nuestro formulario podremos clicar en 'guardar' o 'save' para registrar nuestro artículo nuevo en base de datos. Una vez hecho esto el artículo nuevo ya está listo para poder realizar movimientos sobre él o para su edición en caso de necesitar cambiar cualquier campo a posteriori.

Para editar un artículo bastará con que hagamos clic sobre el botón ‘editar’ o ‘edit’ del artículo que se encuentra en la parte inferior de su *card*, que nos desplegará el mismo diálogo de crear artículo de la figura 28 solo que con los campos ya rellenados y correspondientes al artículo que estemos editando. La diferencia entre el diálogo de crear y el de editar es que en el de editar solo podremos cambiar la foto y las notas. Esta lógica está programada así para evitar incongruencias con el histórico, ya que si pudiéramos cambiar de nombre o tipo los artículos que quisiéramos perderíamos el registro de los artículos originales y eso no nos interesa.

Por otra parte, para realizar un movimiento de stock clicaremos sobre el icono de las dos flechas que se encuentra en la parte inferior derecha de la foto en la *card* de cualquier artículo, que nos abrirá un dialogo cuya disposición consta de los campos a rellenar a la izquierda y de la información del ítem a la derecha, y a la hora de realizar tanto entradas como salidas sus campos variarán (ver figuras 31 y 32) en función de si elegimos uno u otro. Si hemos elegido ‘entrada’ o ‘entry’ tendremos como campos la cantidad a depositar y las notas opcionales, sin embargo, si hemos elegido la opción de salida los campos serán: la cantidad que vamos a sacar, el equipo al que va destinado el artículo, el jugador al que va destinado el artículo y las notas opcionales. Es necesario seleccionar un equipo antes que un jugador puesto que cada entrenador tendrá distintos equipos asignados y sus jugadores pueden variar.

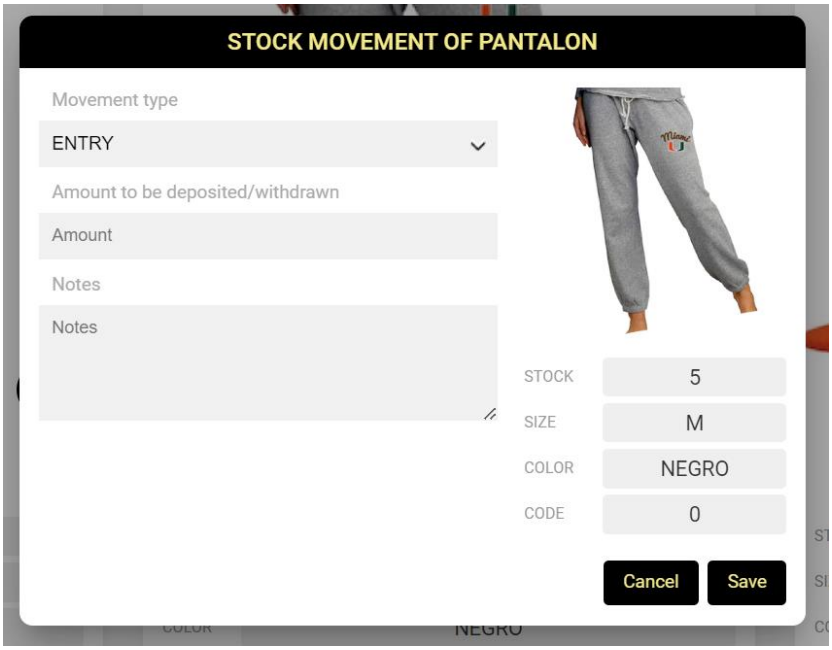

Figura 31 – diálogo de movimiento de artículo con entrada

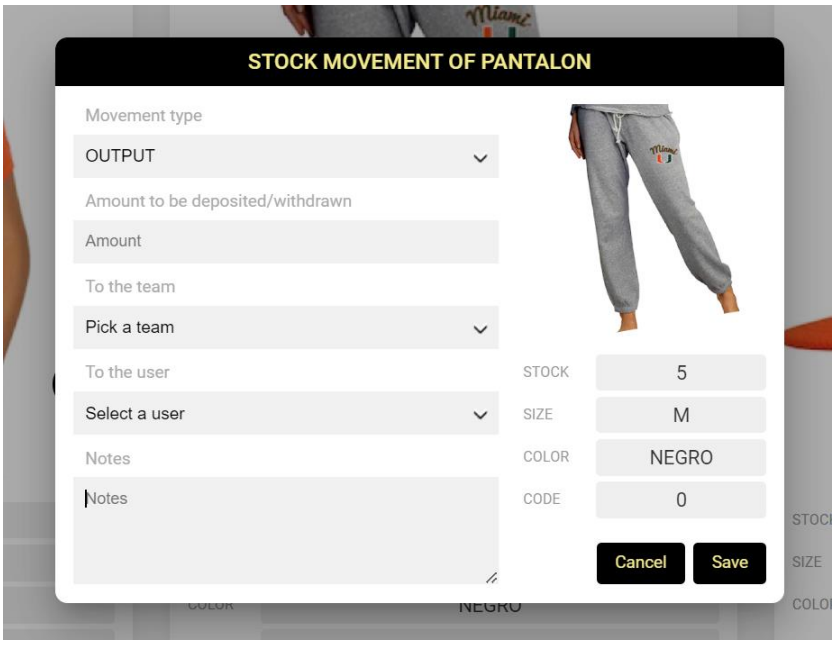

Figura 32 – diálogo de movimiento de artículo con salida

Una vez tengamos todos los campos obligatorios rellenados clicaremos en 'save' o 'guardar' y nuestro movimiento quedará registrado y la cantidad introducida será sumada o restada en función de lo que hayamos escogido. Cabe decir que en esta parte del proyecto se ha programado un sistema de prevención de errores que en el caso de la salida de producto no te permitirá sacar más cantidad de la ya existente en base de datos, es decir que si dos personas se encuentran realizando una salida del mismo producto y el primero saca toda la cantidad posible, cuando el segundo vaya a realizar la salida, se cotejará la cantidad que él ha escrito con la actual que se encuentre en base de datos y le aparecerá en pantalla un mensaje de alerta con un error diciéndole que no es posible extraer dicha cantidad dado que no hay más existencias o que su cantidad ha excedido las existencias del producto.

La vista de histórico del entrenador es prácticamente idéntica a la del administrador comentada anteriormente solo que no tendrá las cabeceras para elegir entidades, sino que solamente podrá ver los movimientos de su propia entidad.

6.2 Vista móvil

En la vista móvil tendremos prácticamente la misma lógica, alguna función añadida y, por supuesto, cambios importantes en el diseño para hacerlo lo más *responsive* posible.

En primer lugar, nos encontraremos con la ventana de *login*, exactamente igual que en la vista web introduciremos nuestro email de usuario y contraseña que nos llevará a la página de *home*/principal. En la vista móvil el menú lateral se encuentra minimizado a la izquierda y para desplegarlo pulsaremos en el icono de los tres palitos o hamburguesa (ver figura 33 izquierda).

En el menú podemos observar al igual que en la vista web todos los módulos de la aplicación y entre ellos nuestro módulo almacén (ver figura 33 derecha) que, al pulsar sobre él nos llevará directamente a la lista de artículos.

No implementaremos en la vista móvil la página del histórico puesto que tiene tantos campos de información que sería incomodo de visualizar y de adaptar en una pantalla móvil. Así pues, el histórico lo relegaremos única y exclusivamente a la parte web.

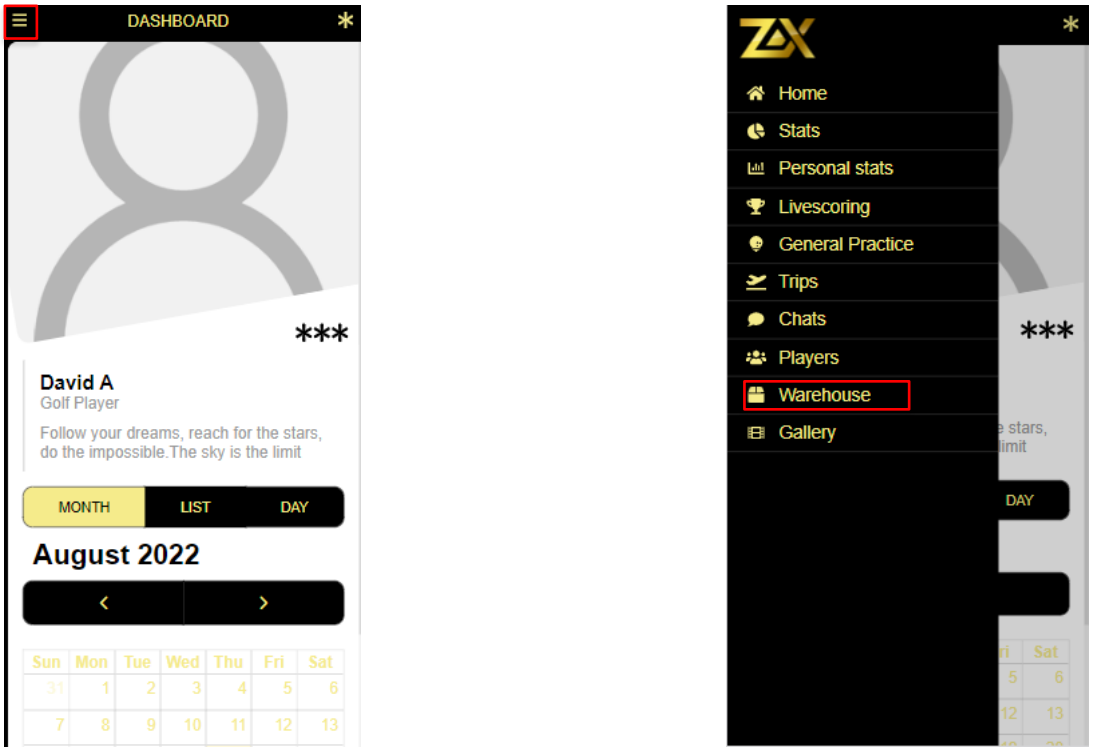

Figura 33 – páginas de home con el menú cerrado y abierto respectivamente

6.2.1 Administrador

Muy parecido a la vista web como administradores dispondremos de un selector de entidades para ver las listas de cada universidad y de un botón de borrado para cada artículo (ver figura 34). El selector de estilos para cada universidad lo relegaremos solo a la vista web, puesto que al ser la que desarrollaremos primero ya testeáramos como quedarán los colores en dicha vista, y al utilizar la misma paleta de colores en nuestra vista móvil nos podemos ahorrar implementarlo aquí.

Los casos de uso de administrador ya han sido explicados previamente en la parte web así que no indagaremos mucho más en ellos puesto que la lógica de estos es idéntica a la mencionada en la vista web. Al igual que en la parte web un administrador también podrá crear o editar artículos de cualquier entidad y realizar cualquier tipo de movimiento, con alguna que otra diferencia en el proceso que explicaremos en la parte del entrenador.

Figura 34 – ejemplo de vista de administrador en la lista de árticulos

6.2.2 Entrenador

El entrenador tendrá los mismos casos de uso que la parte web solo que las vistas cambiarán un poco debido a que aquí haremos uso de una funcionalidad nueva denominada 'escáner', que hará uso de la cámara de nuestro dispositivo para leer y registrar códigos de barras o QRs con la finalidad de poder crear artículos nuevos, que vengan con un código que los identifique, mucho más rápidamente o para poder encontrar un ítem en nuestro inventario de una forma más sencilla.

En primer lugar si observamos la figura 35 izqauierda , para crear un artículo pulsaremos sobre el botón del '+' que se encuentra a la derecha del buscador que nos mandará a una nueva ventana con el formulario para rellenar. Seguiremos la misma lógica que en la versión web solo que tendremos como novedad la funcionalidad antes comentada del escáner para registrar códigos más rápidamente. El escáner lo activaremos pulsando el botón que se encuentra a la derecha del campo 'code' o 'codigo' con el icono de una cámara de fotos, que leerá el código del objeto y automáticamente rellenará el campo con el valor escaneado. Una vez tengamos los

campos rellenados pulsaremos en ‘save’ o ‘guardar’ y el artículo nuevo quedará registrado en base de datos.

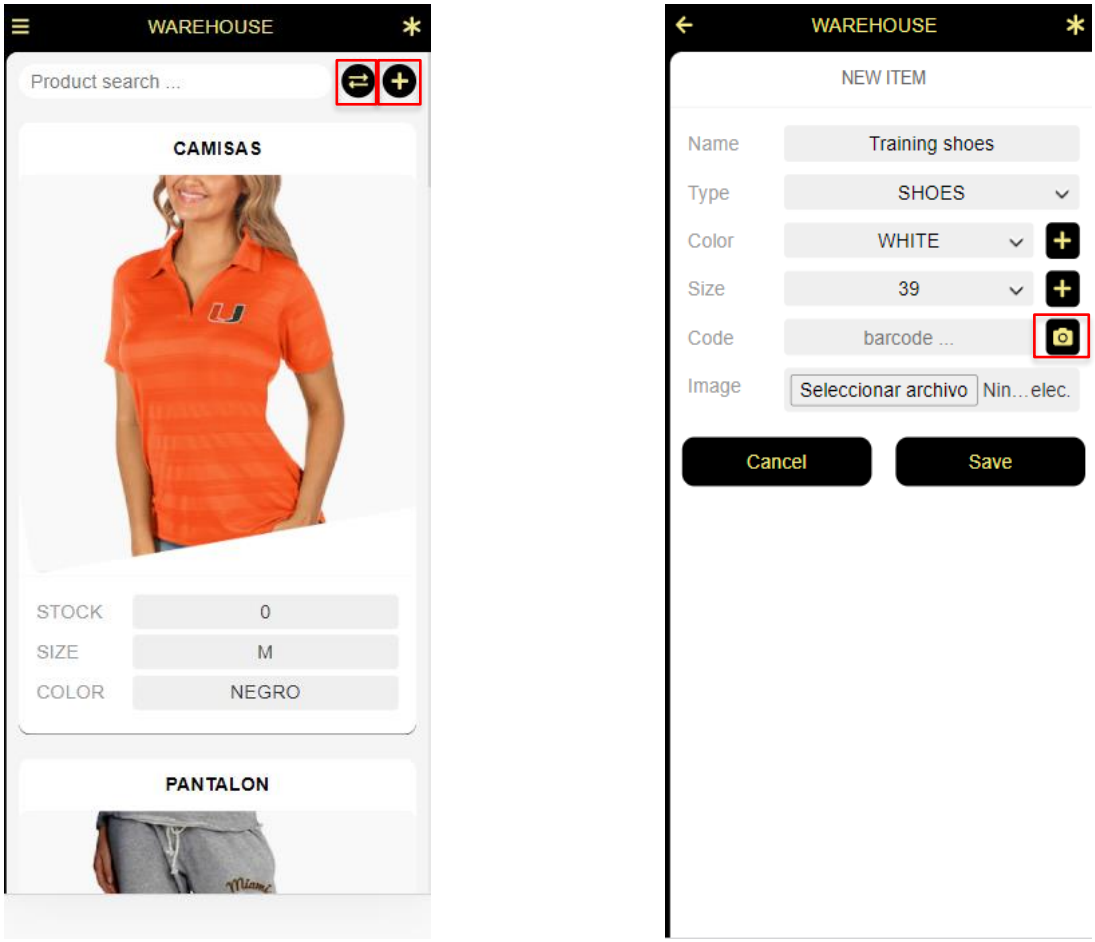

Figura 35 – ejemplos de lista de artículos con vista entrenador y formulario de creación de nuevo artículo respectivamente

Por otra parte, para el caso de realizar un movimiento de stock cambiaremos la lógica un poco para dar uso al escáner que nos permitirá identificar el artículo que vamos a mover y ahorrarnos el buscarlo en la lista. Para gestionar un movimiento primero pulsaremos sobre el icono de las dos flechas (ver figura 35 izquierda) y acto seguido navegaremos a una ventana que tendrá dos botones (ver figura 36) que nos darán las opciones de: buscar manualmente en la lista los artículos que queramos gestionar, o la de utilizar el escáner sobre el objeto directamente para leer su código y que la aplicación nos lo encuentre automáticamente. Una vez encontrado el artículo que queremos gestionar tendremos un formulario (ver figura 36 derecha) con los mismos campos que en la vista web para especificar si es: salida o entrada, la cantidad y si se trata de una salida elegir el equipo y jugador al que va destinado. Cuando hayamos completado el formulario pulsaremos sobre ‘add’ que no nos guardará el movimiento inmediatamente sino que lo mantendremos en una lista en local por si el usuario desea realizar más movimientos con la finalidad de guardarlos todos a la vez (ver figura 37), esto es, para facilitarle al usuario la

gestión rápida de varios artículos al mismo tiempo y que no tenga que entrar en el apartado de movimientos todo el tiempo si quiere gestionar más de un artículo.

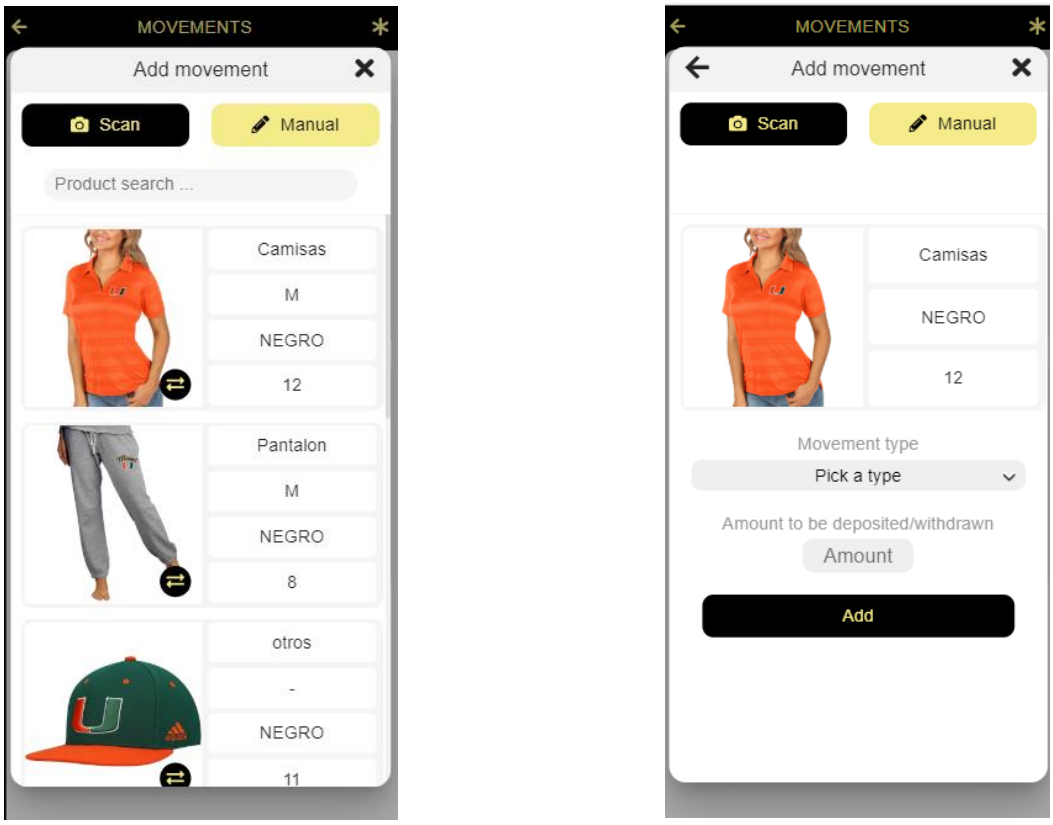

Figura 36 – ejemplos de lista de artículos filtrados manualmente y formulario de creación de nuevo movimiento respectivamente

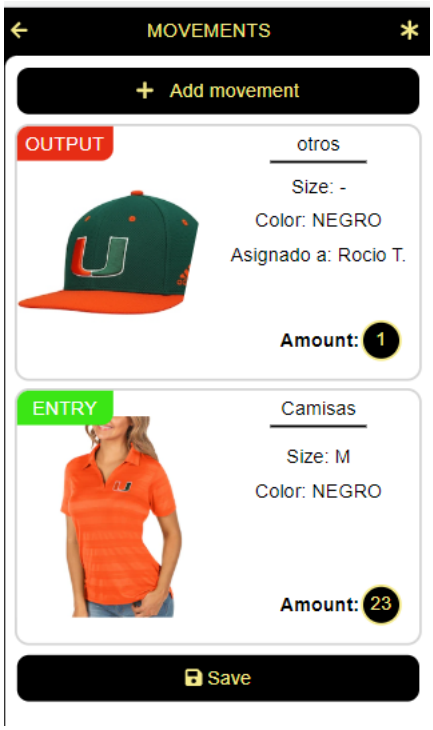

Figura 37 – ejemplo de lista de movimientos

7. Conclusiones y trabajos futuros

En este apartado realizaremos una breve conclusión con algunos de los aspectos finales más relevantes de nuestro proyecto y acto seguido detallaremos algunas de las funcionalidades que se han propuesto después del desarrollo.

Como conclusión para este proyecto y tras haber logrado los objetivos marcados al inicio de esta memoria, tomo en consideración la importancia que es comentar y mantener un orden a la hora de implementar el código, realizar comentarios constantes y compartir mis problemas a lo largo del desarrollo con el resto de los miembros del equipo ha sido clave a la hora de crear funcionalidades o resolver problemas lógicos.

Para resolver la problemática que se me planteó fui autodidacta a la hora de aprender cómo funcionaba el *framework* de angular y adquirí experiencia realizando proyectos menores. Esto me dio las competencias necesarias para poder entender una aplicación a mitad de su desarrollo y para integrar un modelo funcional completamente nuevo dentro de ella.

A la hora de empezar a desarrollar el módulo me planifiqué una serie de tareas, objetivos y posibles problemas que me surgirían, debido a mi falta de experiencia, detalladas sobre papel. Evidentemente no todo es previsible y me surgieron muchos mas problemas de los que pensaba, lo que retrasó un poco la entrega de algunos objetivos, pero en general me ayudó a tener mi trabajo ordenado.

Cuando tuve que elegir la tecnología que iba a utilizar realmente fue un condicionante puesto que tuve que adaptarme a la que tenían mis compañeros. A base de preguntar y comentar me adapté a la técnica que ellos utilizaban y pude seguirles el ritmo. Tuve algunos problemas al principio para comprender como PHP se comunicaba con la base de datos y diseñando algunas ventanas utilizando clases CSS, pero me sirvió como experiencia profesional para ampliar mis conocimientos en esos campos.

7.1 Trabajos futuros

En primer lugar, cuando algunas de las universidades probaron el módulo en cuestión con la vista web, se dieron cuenta de que tardaban mucho tiempo en realizar varios movimientos de artículos a la vez debido a que tenían que ir de uno en uno para realizar la gestión. Por esto se propuso diseñar una nueva lógica en la web que fuera capaz de gestionar varios movimientos al mismo tiempo y que permitiera en una sola ventana realizar varias gestiones de entrada y salida a la vez sobre todos los artículos que se desearan. En respuesta a esto se implementará en un entorno de pruebas lo que podemos observar en la figura 38 que no es más que una ventana de

dialogo donde almacenaremos dos listas, una de entradas y otra de salidas, que se cotejarán con la base de datos y, si todos los movimientos son posibles, se efectuarán con éxito todas las gestiones al mismo tiempo.

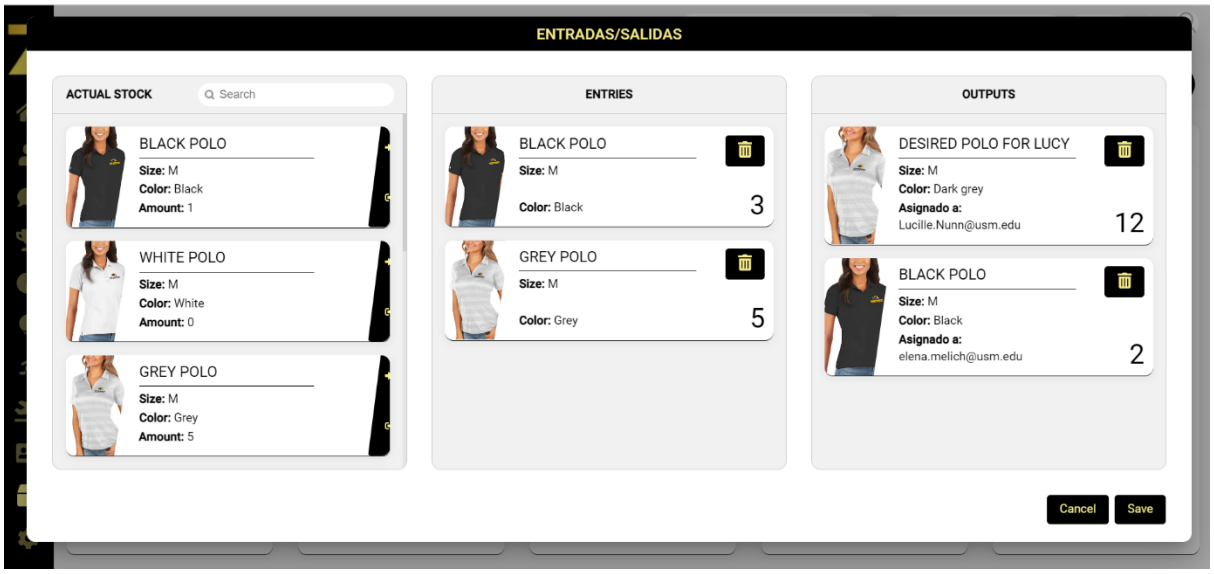

Figura 38 – dialogo de movimientos múltiples

En segundo lugar, la aplicación tuvo unos plazos de tiempo bastante ajustados y esto provocó que el desarrollo fuera un poco accidentado, que el código no estuviera todo lo limpio que se desearía y que los tiempos de carga de algunos elementos o ventanas fuera mas alto de lo esperado. Para paliar esta situación se propuso a los inversores realizar una refactorización completa del código y de paso realizar un rediseño completo de los estilos de la aplicación con la ayuda de un diseñador. En las siguientes imágenes podemos observar que tanto la aplicación como nuestro módulo intentarán ofrecer un estilo mucho más simplificado y menos cargado.

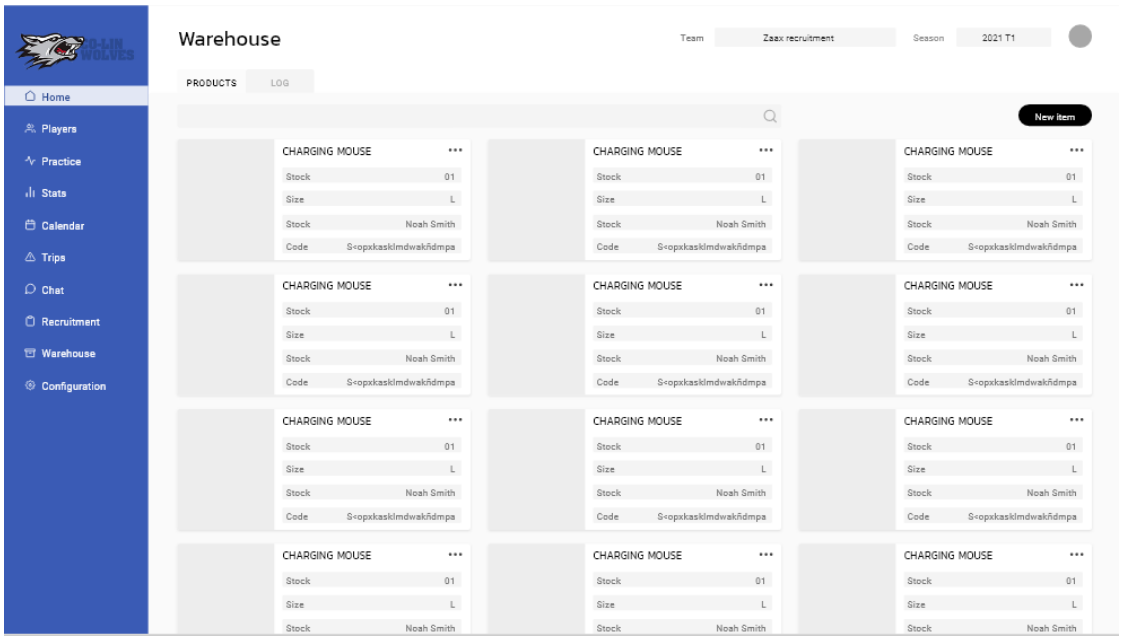

Figura 39 – rediseño de ventana de lista de artículos de almacén

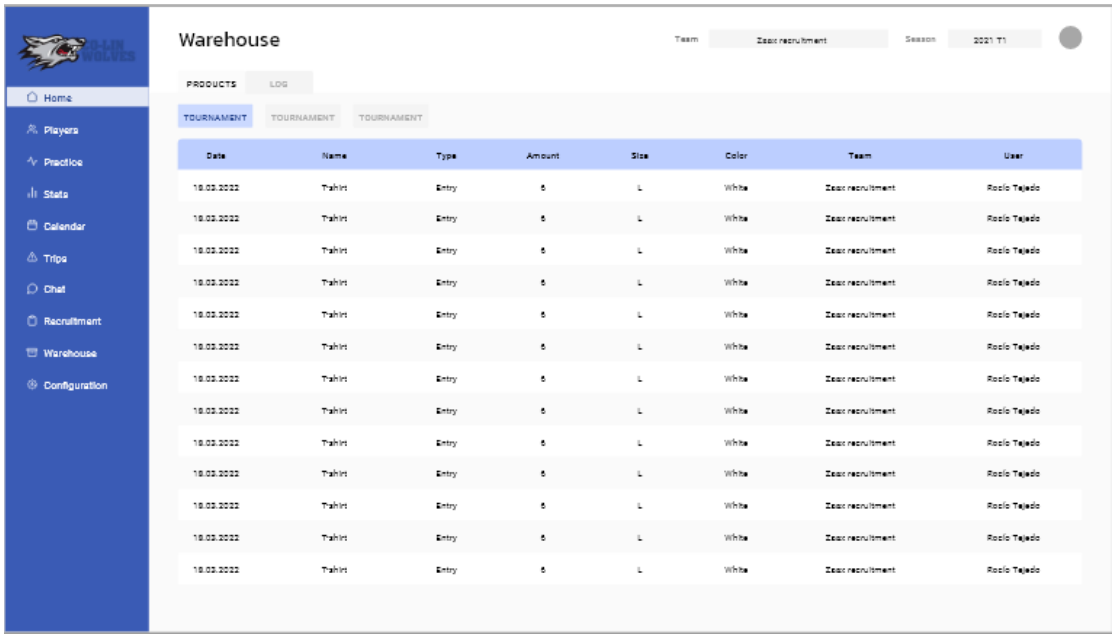

Figura 40 – rediseño de ventana de la vista del histórico

Se puede observar en la figura como a pesar de haber cambiado el diseño seguimos manteniendo las funcionalidades y la identidad única de cada entidad. Rediseñando tanto la aplicación como nuestro proyecto conseguiremos ofrecer una vista mas intuitiva ofreciendo un mejor servicio al cliente y garantizando que toda la información sea mucho mas accesible y de rápido acceso.

En tercer lugar, y gracias a la refactorización de código comentada anteriormente, reduciremos los tiempos de carga de los datos modificando el archivo de *routing*(ver figura 41) para aprovechar el *lazy loading* que nos ofrece Angular. De esta forma la aplicación no cargará todos los módulos a la vez y solamente cargará nuestro modulo almacén cuando el usuario quiera acceder a él, reduciendo así la carga del sistema. También crearemos una API totalmente nueva que disminuirá los tiempos de respuesta de nuestras llamadas a *backend*, reduciendo así los periodos de espera de datos que teníamos previamente.

Figura 41 – fragmento de código de routing refactorizado.

8. Bibliografía

Catinfog. Consultado en https://catinfog.com/programas-almacen/. Fecha de consulta: junio 2022.

Snipe-IT. Consultado en https://snipeitapp.com/. Fecha de consulta: junio 2022.

Facturascripts. Consultado en https://snipeitapp.com/. Fecha de consulta: junio 2022.

PartKeepr. Consultado en https://snipeitapp.com/. Fecha de consulta: junio 2022.

Metodología Scrumb. Consultado en https://www.apd.es/. Fecha de consulta: junio de 2022.

Git. Consultado en https://docs.microsoft.com/. Fecha de consulta: junio de 2022.

Angular. Consultado en https://medium.com/. Fecha de consulta: julio de 2022.

Typescript. Consultado en https://es.wikipedia.org/. Fecha de consulta: julio de 2022.

MySql. Consultado en https://openwebinars.net/. Fecha de consulta: julio de 2022.

FileZilla. Consultado en https://www.hoswedaje.com/. Fecha de consulta: julio 2022.

9. Anexo

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**x**
ODS 2. **Hambre cero.**				**x**
ODS 3. **Salud y bienestar.**				**x**
ODS 4. **Educación de calidad.**			**x**	
ODS 5. **Igualdad de género.**				**x**
ODS 6. **Agua limpia y saneamiento.**				**x**
ODS 7. **Energía asequible y no contaminante.**				**x**
ODS 8. **Trabajo decente y crecimiento económico.**				**x**
ODS 9. **Industria, innovación e infraestructuras.**				**x**
ODS 10. **Reducción de las desigualdades**.				**x**
ODS 11. **Ciudades y comunidades sostenibles**.				**x**
ODS 12. **Producción y consumo responsables.**			**x**	
ODS 13. **Acción por el clima.**			**x**	
ODS 14. **Vida submarina.**				**x**
ODS 15. **Vida de ecosistemas terrestres.**			**x**	
ODS 16. **Paz, justicia e instituciones sólidas.**				**x**
ODS 17. **Alianzas para lograr objetivos.**				**x**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Podríamos decir que, si bien nuestro TFG no está orientado principalmente a cumplir objetivos de desarrollo sostenible, pero si puede contribuir en menor medida a algunos de los mencionados en la tabla superior.

En primer lugar, si nuestro objetivo de reducir las pérdidas de material deportivo en las universidades se cumple, estaríamos hablando de que tendríamos una reducción en los pedidos de transporte de mercancías que podrían influir positivamente en las emisiones de dióxido de carbono.

En segundo lugar, si garantizamos material deportivo para todo el mundo estaríamos ayudando a que todos tuvieran las mismas oportunidades para aprender el deporte y en las mismas condiciones por lo que estaríamos ayudando un poco al objetivo sostenible de educación de calidad.

En tercer lugar, al proporcionar un seguimiento de los productos que salen o se dejan a los jugadores concienciamos un poco sobre la importancia que es mantener los artículos en buen estado para que otros los utilicen después, ayudando así a mantener un consumo responsable.

En cuarto lugar, reducir la perdida de material también podría tener un impacto positivo en los ecosistemas terrestres, pudiendo proporcionarles a los materiales que ya no se utilicen o estén rotos un cicle de reciclaje óptimo.

Por último, hablaremos sobre los ODS que no puede cumplir por varias razones:

- **Fin de la pobreza**: nuestro proyecto no tiene herramientas para suplir la brecha entre los más ricos y los más pobres ni tampoco para garantizarles a estos últimos un medio para salir de ella.
- **Hambre cero**: el ámbito del proyecto no tiene que ver con el sector alimenticio, ni está enfocado a aumentar de alguna manera la inversión ni la cooperación internacional en este espacio.
- **Salud y bienestar**: el proyecto no está enfocado a ello y no proporciona ninguna herramienta para cumplir cualquier meta en este aspecto.
- **Igualdad de género**: nuestro proyecto no aporta ninguna herramienta para poner fin a la discriminación contra la mujer o para promover políticas o leyes para garantizarla
- **Agua limpia y saneamiento**: nuestro proyecto no trata este tema.

- **Energía asequible y no contaminante**: el proyecto no pretende aumentar la eficiencia energética ni promover el uso de renovables.
- **Crecimiento económico**: aunque si es cierto que al reducir la perdida de material estaríamos ayudando a que las universidades ahorraran más, el proyecto no posee funcionalidades que ayuden a la creación de puestos de trabajo ni para mejorar sus derechos ni sus condiciones.
- **Industria innovación e infraestructuras**: el proyecto no tiene herramientas para promover una industrialización inclusiva y sostenible ni para facilitar su desarrollo.
- **Reducción de las desigualdades**: el proyecto no tiene funcionalidades para promover la inclusión de ningún tipo ni para reducir la desigualdad.
- **Ciudades y comunidades sostenibles**: no es el ámbito del proyecto.
- **Vida submarina**: el objetivo no tiene que ver con lo que aborda nuestro proyecto.
- **Paz, justicia e instituciones sólidas**: no es el ámbito del proyecto.
- **Alianzas para lograr objetivos**: en el proyecto no tenemos herramientas para promover o aumentar alianzas en ningún aspecto.