Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Software para la planificación del almacenamiento de mercancías

Software for planning the storage of goods

Saúl Pérez García

La Laguna, 13 de julio de 2023

D. **Christopher Expósito Izquierdo**, con N.I.F. 78.851.649-J profesor Ayudante Doctor adscrito al Departamento Ingeniería Informática y de Sistemas de la Universidad de La Laguna como tutor

D. **Israel López Plata**, con N.I.F. 42.193.801-W profesor Ayudante Doctor adscrito al Departamento Ingeniería Informática y de Sistemas de la Universidad de La Laguna como cotutor

C E R T I F I C A N

Que la presente memoria titulada:

”*Software para la planificación del almacenamiento de mercancías*”

ha sido realizada bajo su dirección por D. **Saúl Pérez García**, con N.I.F. 43.835.385-V.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de julio de 2023

Agradecimientos

Hace unos años daba el primer paso en una dirección que me permitiera tener total libertad, donde el límite fuera yo mismo y donde tuviera la capacidad de crear todo tipo de soluciones nuevas, originales y propias, con aplicaciones al mundo.

Ha sido un proceso largo y lleno de escollos que he podido superar a base de persistencia y perseverancia.

Quiero agradecer en primer lugar a mis tutores, Christopher e Israel, por su tutela, su paciencia y guía en todos estos meses de desarrollo.

A mi familia que siempre ha estado ahí para alentarme y arroparme en todos esos momentos donde ya no podía más y cuidarme para llegar a tallecer como lo he hecho. Sin ellos no sería la persona que soy hoy en día.

Finalmente dar gracias a todos mis amigos y personas que han estado apoyándome a lo largo de camino. En especial a Jaime y Anabel por ser los mejores compañeros de batallas que podría haber pedido. Siempre han sido ese rincón donde poder desahogarme y encontrar el apoyo necesario para recargar energías y levantarme nuevamente con más fuerza.

Licencia

Resumen

El contexto del TFG es el almacenamiento de las mercancías y aquellas acciones que se pueden realizar sobre el patio de contenedores tales como la localización de la misma, la capacidad para añadirla o retirarla, así como el manejo de los recursos para gestionarla entre los que se encuentran el personal, las grúas, los camiones y los barcos que traen y retiran los contenedores.

El objetivo que se persigue en este proyecto consiste en el desarrollo de un software para la gestión de mercancía en el contexto portuario. El mismo se va a estructurar con arquitectura hexagonal en el frontend que utilizará Vue.js y una librería de componentes como Vuetify y en backend que va a llevar Spring como framework para Java que permite crear APIs y proporciona la capacidad necesaria para escalar la aplicación. Para el despliegue de la aplicación se va a emplear Docker Compose.

Palabras clave: Frontend, backend, arquitectura hexagonal, framework, API

Abstract

The context of the TFG will be the storage of the goods and those actions that can be carried out on the container yard such as its location, the ability to add or remove it, as well as the management of resources to manage it, among which are they find the personnel, the cranes, the trucks and the ships that bring and remove the containers.

The objective pursued in this project will consist of the development of a software for the management of merchandise in the port context. It will be structured with a hexagonal architecture in the frontend that will use Vue.js and a component library such as Vuetify and in the backend that Spring will be used as a framework for Java that allows creating APIs and provides the necessary capacity for scale the app. Docker Compose will be used to deploy the application.

Keywords: Frontend, backend, hexagonal architecture, framework, API

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

El reto por optimizar los procesos logísticos y el comercio internacional, unido a un entorno económico cada vez más competitivo y dinámico ha motivado el desarrollo de iniciativas y herramientas que supongan una mejora sustancial de los procedimientos logístico-portuarios.

En este contexto, los **softwares de planificación de mercancías** desempeñan un papel fundamental al permitir la optimización de los procesos logísticos y el comercio internacional en los puertos. Estas herramientas abarcan una amplia gama de áreas, como la gestión del transporte, la previsión de la demanda, la infraestructura, el personal y la seguridad de la mercancía. Su implementación conlleva mejoras sustanciales en la eficiencia y la efectividad, lo que resulta fundamental para hacer frente a los desafíos actuales en el ámbito del comercio internacional.

La cadena logística abarca ámbitos tan dispares como la gestión del transporte, previsión de la demanda, infraestructuras, personal o seguridad de la mercancía. Como principales eslabones de la misma, se pueden identificar tres procesos fundamentales que son clave para la gestión eficiente y automatizada de los procesos portuarios:

- **Aprovisionamiento:** Este proceso se refiere a la planificación y gestión de la llegada de la mercancía al puerto. Incluye la programación de los barcos y la gestión de los procesos de descarga, así como la recepción, clasificación y almacenamiento de las mercancías en el almacén.
- **Manipulación y almacenamiento:** Una vez que la carga llega al puerto, el siguiente proceso clave es su manipulación y almacenamiento. Esto implica la gestión de la carga y descarga de los contenedores, su clasificación y depósito en el almacén, así como la gestión de la documentación necesaria para el seguimiento y la trazabilidad de las mercancías.
- **Distribución:** Una vez que las mercancías han sido manipuladas y almacenadas, el último proceso clave es la distribución de la misma. Esto implica la planificación y gestión de la entrega de las mercancías a los destinatarios finales, incluyendo la programación de los transportistas y la gestión de la documentación necesaria para el seguimiento y la trazabilidad de las mercancías.

Por tanto, el papel que desempeñan los softwares de planificación de mercancías en puertos son fundamentales para optimizar los procesos logísticos y el comercio internacional, y los tres procesos fundamentales de la cadena logística - aprovisionamiento,

manipulación y almacenamiento, y distribución - son clave para una gestión eficiente y automatizada de los procesos portuarios.

1.2. Justificación

Cuando se habla de terminales marítimas portuarias, se hace referencia al espacio físico para la manipulación de mercancías dentro del puerto. Su función principal es proporcionar a través de un proceso logístico los recursos y la organización necesarios para que las maniobras de carga y descarga de contenedores entre los medios de transporte terrestre y marítimo se realicen en los mejores términos de rapidez y eficiencia posible.

La gran mayoría de las mercancías que se consumen en Canarias entran a través de los puertos, es por esto que juegan un papel de gran relevancia en torno a la actividad económica y social del archipiélago. Es un sector muy importante debido a la alta dependencia del exterior para poder lograr el abastecimiento de las islas, por ello resulta crucial la capacidad para mantener una alta conectividad que permita la competencia y que, por tanto, facilite que los precios del transporte sean competitivos.

Según el informe **Container Port Performance Index 2022**[3] elaborado por el Banco Mundial y S&P Global Market Intelligence, reconocidas entidades especializadas en comercio y cadena de suministro, el puerto de Santa Cruz de Tenerife se destaca como uno de los puertos más eficientes en España y Europa, situándose en tercer lugar a nivel nacional por detrás de los puertos de Algeciras y Barcelona. También ocupa el puesto 14º en el ranking de eficiencia portuaria para Europa y el norte de África.

El análisis se basa en el tiempo promedio que los buques permanecen en el puerto desde su llegada hasta su partida, teniendo en cuenta las actividades de carga y descarga. Cuanto menos tiempo se requiera para estas tareas, mayor eficiencia se atribuye a la infraestructura portuaria.

Este informe resalta la importancia de un puerto eficiente tanto para la economía de su área de influencia como para el correcto funcionamiento de la cadena logística, evitando costos adicionales e interrupciones en el suministro internacional.

Las terminales marítimas portuarias cuentan con cuatro zonas principales. En primer lugar, la **zona de operación**, correspondiente al lugar junto a los muelles, en ella se realizan las actividades de carga y descarga de contenedores a los buques. La **zona de almacenamiento** cuya función es albergar la totalidad de contenedores del puerto. Una zona destinada al cambio de modo de transporte donde se realizan las **actividades de carga y descarga** de contenedores a los medios de transporte terrestres. Y finalmente una **zona de servicio** encontrándose las oficinas de control, talleres de reparación, aparcamientos o aduanas.

Una vez la carga llega al muelle se inicia el flujo de mercancías y cuyos movimientos dependen de la finalidad de la misma. Entre ellos puede ser pasar a otro buque mediante un transbordo, quedarse en el muelle y ser trasladada en el interior del puerto para ser depositada en un almacén o ser trasladada por un transporte terrestre para su distribución.

Sin embargo, y a pesar de contar con numerosos beneficios en términos de política aduanera y comercial debido a la condición insular de las islas, juega en contra la rigidez en los servicios y los horarios establecidos para los controles de entrada y salida de mercancías, generando importantes cuellos de botella que tienen como resultado

Port Name	Region	Overall Ranking
Tanger-Mediterranean	ENA	4
Port Said	ENA	10
Algeciras	ENA	16
Barcelona	ENA	34
Marsaxlokk	ENA	40
Yarimca	ENA	41
Piraeus	ENA	51
Haifa	ENA	56
Ambarli	ENA	57
Bremerhaven	ENA	60
Zeebrugge	ENA	62
Antwerp	ENA	66
Savona-Vado	ENA	68
Santa Cruz De Tenerife	ENA	71

Figura 1.1: CPPI por región: Europa y Norte de África

retrasos y desajustes en la logística portuaria[13].

Con objetivo de seguir aumentando la influencia del puerto es indispensable el desarrollo de un software de estas características que busca una reducción en los tiempos de espera característicos del manejo de grandes volúmenes de mercancías tanto para la carga como la descarga de los buques así como proporcionar información en tiempo real sobre el estado disponibilidad de los recursos. Otro factor determinante reside en la mejora del proceso de toma de decisiones, la información en tiempo real puede ayudar a los gerentes y operadores del puerto a tomar decisiones más acertadas. De la misma forma se obtendría una mayor eficiencia en la gestión del espacio e inventariado y reducir las posibilidades de errores.

1.3. Objetivo del proyecto

El contexto del TFG será el almacenamiento de las mercancías y aquellas acciones que se pueden realizar sobre el patio de contenedores tales como la localización de la misma, la capacidad para añadirla o retirarla, así como el manejo de los recursos para gestionarla entre los que se encuentran el personal, las grúas, los camiones y los barcos que traen y retiran los contenedores.

El objetivo que se persigue con la ejecución de este proyecto consistirá en el desarrollo de un **software para la gestión de mercancía en el contexto portuario**. El mismo se va a estructurar con **arquitectura hexagonal** en el **frontend** implementado con **Vue 3** y una librería de componentes como **Vuetify**. Por otra parte, el **backend** usará **Spring** como framework para **Java** que permite crear APIs y proporciona la ca-

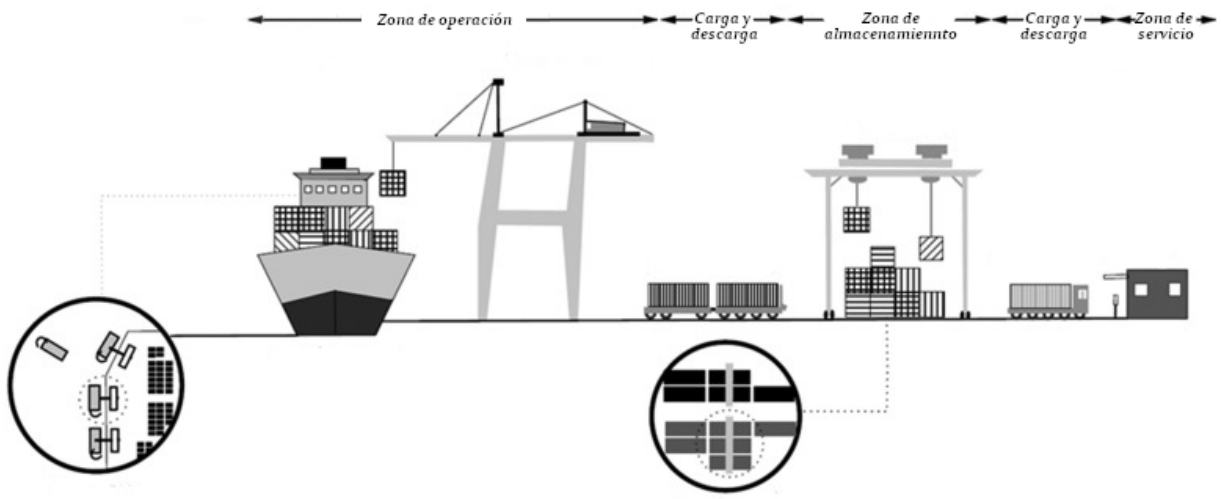


Figura 1.2: Zonas del puerto

pacidad necesaria para escalar la aplicación. Finalemente, para el despliegue de la aplicación se va a emplear **Docker Compose**.

El desarrollo de este proyecto podría estar definido en la ejecución de los siguientes pasos:

- **Diseño, implementación y validación de simulador de flujos de mercancías**
- **Diseño y creación del back-end:**
  - Gestión de mercancías
  - Gestión de maquinaria
  - Gestión de personal
- **Diseño y creación del front-end:**
  - Generación de flujos de mercancías
  - Estado de los contenedores a lo largo del tiempo
  - Visualización gráfica del almacenamiento
- **Dashboard con indicadores clave de rendimiento**

Este proyecto abre una línea de trabajo con el fin de poder maximizar la disponibilidad de los contenedores y la capacidad de almacenamiento y rotación de los mismos a la vez que se pretende minimizar entre otros el espacio empleado con el fin de aumentar la rentabilidad, las necesidades de inversión y costes de administración de inventarios y las manipulaciones, para que los recorridos y movimientos de personal o equipos como grúas o camiones sean los mínimos posibles.

1.4. Estructura del documento

- **Capítulo 1 - Introducción:** En este capítulo se expone el contexto del proyecto, su justificación y los objetivos que se persiguen en su desarrollo.

- **Capítulo 2 - Estado del arte:** Se realiza una vista histórica del ámbito en el que se desenvuelve este trabajo y se señalizan algunas de las soluciones actuales en el mercado.
- **Capítulo 3 - Solución tecnológica:** Apartado dedicado a la exposición de las tecnologías empleadas para el desarrollo del prototipo, así cómo la justificación de la elección de las mismas.
- **Capítulo 4 - Despliegue del proyecto:** Proporciona información sobre la arquitectura de la aplicación y describe los pasos para el despliegue local y el despliegue con Docker.
- **Capítulo 5 - Resultados:** En este capítulo se simula un caso real de planificación de almacenamiento en un puerto y describe la interfaz de usuario de la aplicación.
- **Capítulo 6 - Presupuesto:** Presupuesto para la realización del trabajo.
- **Capítulo 7 - Conclusiones:** Conclusiones del proyecto y líneas futuras para su desarrollo.
- **Capítulo 8 - Conclusions:** Conclusions of the project and future lines for its development.

Capítulo 2

Estado del arte

2.1. Antecedentes

Los softwares de planificación de mercancías en puertos han evolucionado significativamente en las últimas décadas, impulsados por la creciente necesidad de una gestión más eficiente y automatizada de los procesos portuarios.

En la década de 1980, los sistemas informáticos para la gestión de puertos se centraban principalmente en la gestión de la facturación y las operaciones de carga y descarga de barcos, y se basaban en sistemas mainframe centralizados. Sin embargo, a medida que la demanda de capacidad portuaria aumentaba, se hizo evidente que se necesitaban soluciones más avanzadas y eficientes para la planificación y gestión de la carga y descarga de barcos.

En la década de 1990, se produjo una gran evolución en los softwares de planificación de puertos, con la introducción de sistemas de gestión de terminales portuarias (TOS). Estos sistemas permitieron una mayor automatización de los procesos de gestión de carga y descarga de barcos, lo que a su vez mejoró la eficiencia y la velocidad de los procesos portuarios.

En los últimos años, los softwares de planificación de mercancías en puertos han seguido evolucionando, gracias a los avances en tecnologías como la inteligencia artificial, el aprendizaje automático y el Internet de las cosas (IoT). Estas tecnologías están permitiendo la creación de soluciones más avanzadas e integrales para la planificación y gestión de la cadena de suministro[7], que permiten una mayor eficiencia y eficacia en los procesos portuarios.

2.2. Situación en el ámbito internacional

Durante las últimas décadas, el transporte por mar ha experimentado un notable crecimiento debido al aumento del comercio y la globalización.

Frecuentemente conocido como el pilar central del comercio internacional, el transporte marítimo sigue siendo la forma principal de transporte imperante para los bienes que se comercializan y es fundamental para las redes de suministro a nivel global. De acuerdo con los datos proporcionados por la **Organización Mundial del Comercio**[1], más del $80\,\%$ del volumen total de las mercancías intercambiadas a nivel mundial son transportadas a través de vías marítimas.

[1] https://www.wto.org/spanish/tratop_s/serv_s/transport_s/transport_maritime_s.htm

Los softwares de planificación de mercancías en puertos son herramientas esenciales para la gestión eficiente de la carga y descarga de barcos, así como para la optimización de los procesos de almacenamiento y transporte dentro del puerto. En la actualidad, existen diversas soluciones de software en el mercado que se enfocan en la planificación y gestión de mercancías en puertos.

Hoy en día, existe una amplia gama de tecnologías de la información y comunicación disponibles para optimizar los procesos logísticos de la cadena de suministro[12]. En cuanto a la **logística de entrada**, se destacan el intercambio electrónico de documentos **(EDI)**, la gestión de inventario por parte del proveedor **(VMI)**, los programas de reabastecimiento continuo **(CRP)**, la adquisición electrónica **(e-procurement)** y la búsqueda electrónica de proveedores **(e-sourcing)**. En cuanto a la **logística interna**, se consideran sistemas de planificación de recursos empresariales **(ERP)**, planificación de requerimientos de materiales I **(MRP I)**, **MRP II** y sistemas de gestión de almacenes **(WMS)**. Por último, para la **logística de salida** se emplean sistemas de gestión de transporte **(TMS)**, **EDI** o códigos electrónicos de productos **(EPC)**, entre otros.

En lo referido al patio de almacenamiento, las grúas son el equipo crucial para realizar el traslado de los contenedores. La mayor problemática a resolver sería en el momento de situarlas en la zona de operación dependiendo de la cantidad de grúas disponibles que pueden ser desplegadas en cada sección del patio de mercancías y cómo desplazar el grúas de patio entre los distintos sectores así como el manejo del personal portuario disponible.

En relación a lo anterior se suelen utilizar diferentes soluciones tales como, el **sistema de manejo de almacén (WMS)** mencionado anteriormente, que ofrece visibilidad de todo el inventario de una empresa y gestiona las operaciones de logística de la cadena de suministro, el uso de **código de barras y sistemas de radiofrecuencia, sistemas de señalización sin papeles**, que se basan en redes luminosas y sistemas de voz y se dividen en **Pick to Light**, un conjunto de luces que indican al operario las ubicaciones y las cantidades a recoger de los productos y **Pick to Voice**, cuando el operario lleva un equipo de comunicación que permite recibir y enviar mensajes acerca de las operaciones de recogida. Además del empleo de **Yard Management System (YMS)** que sincroniza las operaciones de carga y descarga en los muelles y monitoriza en todo momento el tránsito de patios, muelles y zonas de estacionamiento.

Algunos de los softwares más populares son:

- **Navis:** Es una solución de software de planificación de terminales portuarias que se enfoca en la gestión de la carga y descarga de barcos, así como en la gestión de la logística y el transporte dentro del puerto.
- **Jade Logistics:** Ofrece una solución de software integral para la planificación de terminales portuarias, que incluye la gestión de la carga y descarga de barcos, el seguimiento de las mercancías, la gestión de almacenes y la optimización de la cadena de suministro.
- **Port Optimizer:** Es una solución de software desarrollada por la Autoridad Portuaria de Los Ángeles y la Autoridad Portuaria de Long Beach que se enfoca en la planificación y optimización de la cadena de suministro en los puertos.
- **Terminal Operating System (TOS):** Es una solución de software que se enfoca en la gestión de terminales portuarias, incluyendo la planificación y programación

de la carga y descarga de barcos, la gestión de almacenes, la gestión de contenedores y la optimización de la cadena de suministro.

- **Softship:** Es una solución de software integral para la gestión de terminales portuarias y la planificación de la carga y descarga de barcos, así como para la gestión de almacenes y la optimización de la cadena de suministro.

Capítulo 3

Solución tecnológica

3.1. Estructura del proyecto

El proyecto sigue un modelo de desarrollo en el frontend basado en **arquitectura hexagonal**, un patrón de diseño centrado en separar la lógica empresarial de la interfaz de usuario, utilizando interfaces y puertos para lograr esta división. De esta forma, se garantiza que el núcleo de la aplicación sea fácilmente testeable y reutilizable.

El propósito es aislar la lógica empresarial de cualquier dependencia externa, como la interfaz de usuario, bases de datos o llamadas a la red, con el fin de mejorar el mantenimiento y la escalabilidad de la aplicación. Con ello se garantiza que los cambios en una parte de la aplicación no afecten a otras, lo que hará que el desarrollo y el mantenimiento a largo plazo sean más fáciles y efectivos.

También se va a emplear **Docker Compose**, una herramienta fundamental en el despliegue de aplicaciones. Permite definir, configurar y ejecutar aplicaciones multicontenedor de una manera sencilla y coherente, así como automatizar el proceso de creación y configuración de los mismos y gracias a su escalabilidad permite añadir o quitar contenedores de forma dinámica para adaptarse a las necesidades de la aplicación. También facilita la creación de entornos de prueba, desarrollo y producción de manera más rápida y eficiente. Otra ventaja importante es que Docker Compose permite aislar la aplicación y sus dependencias en contenedores separados, lo que aumenta la seguridad y estabilidad de la misma.

La combinación de **Spring Boot, Vue3 y MongoDB** permite a los desarrolladores crear aplicaciones web escalables y rápidas con una arquitectura altamente modular. Esta combinación también permite crear **aplicaciones CRUD** (crear, leer, actualizar y eliminar) de manera eficiente y sencilla. En general, el uso de estas tecnologías permite a los desarrolladores crear aplicaciones web modernas y escalables de manera rápida y eficiente.

3.2. Frontend

A la hora de seleccionar un framework de desarrollo para el frontend, se ha optado por **Vue3**, una herramienta capaz de crear interfaces de usuario interactivas y dinámicas en el lado del cliente. Es altamente modular, lo que significa que los desarrolladores pueden integrarlo fácilmente en sus aplicaciones y utilizar sus características para crear interfaces de usuario atractivas y responsivas.

Combinado con **Vuetify**, una biblioteca de componentes de interfaz de usuario para

Vue3 que permite a los desarrolladores crear aplicaciones web atractivas y responsivas con facilidad. Está diseñado con el concepto de "Material Design[1]" en mente, lo que significa que utiliza un conjunto de pautas de diseño para crear una interfaz de usuario coherente y atractiva. Esta librería también consta de una amplia variedad de componentes de interfaz de usuario, desde botones y formularios hasta tablas y gráficos que se pueden personalizar fácilmente para adaptarse a las necesidades del proyecto.

Según las estadísticas de npm trends[2], se puede apreciar el número de descargas de cada paquete de NPM durante el último año. Vue ha tenido un claro despunte hasta comienzos de 2023, figura 3.1.

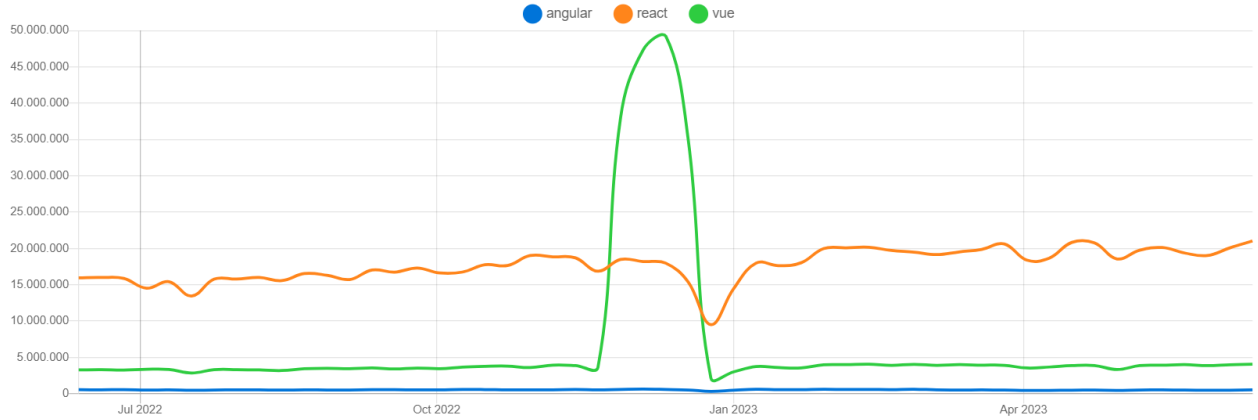


Figura 3.1: Comparativas frameworks

Por su parte Vuetify matiene una tendencia ligeramente superior sobre sus competidores, figura 3.2.

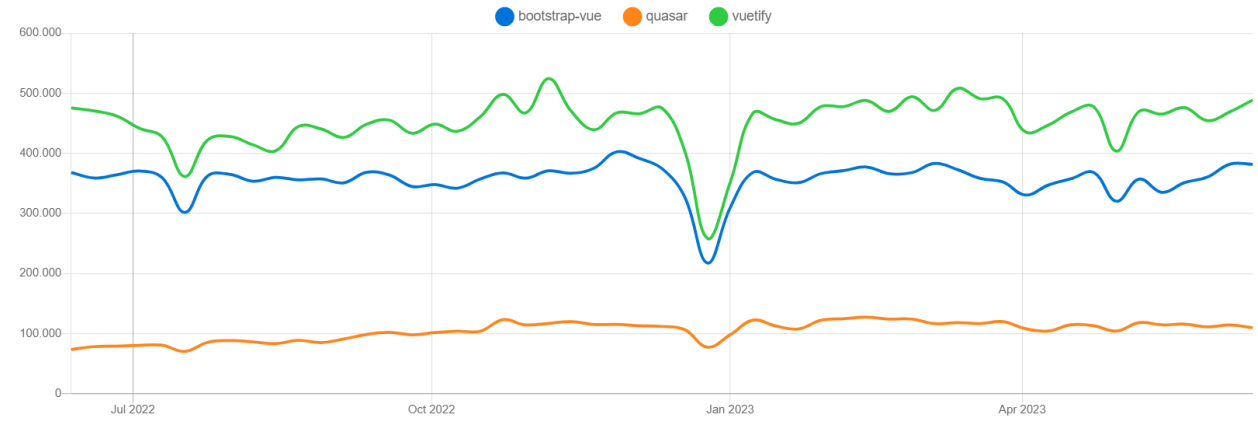


Figura 3.2: Comparativa bibliotecas

Por otra parte, la estructura de directorios es fundamental para organizar y separar de manera clara las diferentes partes de la aplicación. En la siguiente tabla 3.1 se presentan los directorios principales del frontend y su función correspondiente.

Estos abarcan desde componentes reutilizables y vistas principales hasta la integración de API y la gestión de rutas. Además, se destacan los directorios para la lógica de negocio, como las entidades y los objetos valor.

[1] `https://m2.material.io/design/introduction`
[2] `https://npmtrends.com`

Directorios Frontend	
Tipo	Función
/src/adapter	/vuejs/components: Este directorio contiene los componentes reutilizables de Vue3.
	/vuejs/views: En este directorio se ubican los componentes que representan las vistas principales de la aplicación.
	/rest/axios: En este directorio se encuentran los archivos relacionados con la integración de Axios, cuya función será realizar solicitudes HTTP en aplicaciones JavaScript.
	/rest/dto: El directorio /dto se utiliza para almacenar los Data Transfer Objects (DTO) relacionados con la comunicación con la API.
/src/application	En este directorio se alojan las interfaces en TypeScript para cada uno de los recursos de la API REST, definen cómo se recuperan o almacenan los datos desde donde se encuentren.
/src/assets	Se almacenan los recursos estáticos del proyecto como imágenes.
/src/domain	/entity: Se utiliza para definir las entidades de dominio, estas son una representación de un objeto o concepto que tiene una identidad única y propiedades asociadas.
	/valueObject: Se emplea para definir objetos valor, estos representan un concepto dentro del dominio de la aplicación, pero no tienen una identidad propia. Ejemplos de objetos valor serían el peso, ancho, alto o proveedor de un contenedor.
/src/router	En este directorio se sitúa el enrutador para gestionar la navegación entre diferentes vistas o componentes de la aplicación. Se definen las rutas y se configura cómo la aplicación manejará las solicitudes de navegación, cada una se mapea a un componente o vista específica, lo que permite cargar y mostrar dinámicamente diferentes contenidos.
/src/utils	Es comúnmente utilizado en aplicaciones para almacenar funciones o utilidades de propósito general que se utilizan en diferentes partes del código.

Tabla 3.1: Directorios Frontend

3.2.1. Entidades del dominio

En este apartado se expone la descripción y el análisis de las entidades de dominio. Estas son clases que representan los elementos o conceptos fundamentales del dominio. Encapsulan las características esenciales expresadas a través de propiedades, y también definen las relaciones existentes entre ellos permitiendo una representación estructurada y coherente de los elementos esenciales de la aplicación. Como cita Eric Evans[5], un objeto definido principalmente por su identidad se llama entidad.

Puerto

La entidad `Port` encapsula la información relevante de un puerto, como su identificador, nombre o ubicación geográfica dividida en coordenadas de latitud y longitud. Presenta las siguientes propiedades:

- **id**: Una identificación única para el puerto.
- **name**: El nombre del puerto, representado por el objeto valor `PortName`.
- **latitude**: La latitud del puerto, representada por el objeto valor `PortLatitude`.
- **longitude**: La longitud del puerto, representada por el objeto valor `PortLongitude`.

Bloque

La entidad `Block` engloba la información relevante de un bloque del puerto, como su ubicación, capacidad o listado de bahías que lo componen. Presenta las siguientes características:

- **port**: Representa el puerto al que pertenece el bloque. Es de tipo `Port`.
- **id**: Representa la identificación única del bloque. Es de tipo `string`.
- **blockRow**: Representa la fila del bloque. Es de tipo `BlockRow` (opcional).
- **blockColumn**: Representa la columna del bloque. Es de tipo `BlockColumn` (opcional).
- **blockHeight**: Representa la altura del bloque. Es de tipo `BlockHeight` (opcional).
- **stackCapacity**: Representa la capacidad de pila del bloque. Es de tipo `BlockStackCapacity`.
- **bays:** Representa la lista de bahías asociadas al bloque. Es una `List<BlockBay>`.

Contenedor

La entidad `Container` representa un contenedor dentro del dominio de la aplicación, contiene información y características relacionadas con todos los datos relacionados con un contenedor específico y es utilizada para realizar operaciones y cálculos relacionados con la gestión de contenedores. Sus propiedades incluyen:

- **id**: Una identificación única para el contenedor.
- **code**: El código del contenedor, representado por el objeto valor `ContainerCode`.
- **type**: El tipo de contenedor, representado por el objeto valor `ContainerType`.
- **provider**: El proveedor del contenedor, representado por el objeto valor `ContainerProvider`.
- **row**: La fila en la que se encuentra el contenedor, representada por el objeto valor `ContainerRow`.
- **column**: La columna en la que se encuentra el contenedor, representada por el objeto valor `ContainerColumn`.

- **stackIndex**: El índice de apilamiento del contenedor, representado por el objeto valor `ContainerStackIndex`.
- **weight**: El peso del contenedor, representado por el objeto valor `ContainerWeight`.
- **length**: La longitud del contenedor, representada por el objeto valor `ContainerLength`.
- **width**: El ancho del contenedor, representado por el objeto valor `ContainerWidth`.
- **height**: La altura del contenedor, representada por el objeto valor `ContainerHeight`.
- **block**: El bloque al que pertenece el contenedor, representado por la clase `Block`.
- **status**: El estado del contenedor, representado por el objeto valor `ContainerStatus`.

Grúas

La entidad `Crane` se utiliza para modelar y gestionar las grúas en el sistema. Proporciona información importante sobre la ubicación, identificación, marca y personal asociado a cada grúa. Los atributos de esta entidad son:

- **port**: El puerto al que pertenece la grúa, representado por la clase `Port`.
- **block**: El bloque en el que se encuentra la grúa, representado por la clase `Block`. Este atributo es opcional, lo que significa que la grúa puede o no estar asociada a un bloque específico.
- **id**: Una identificación única para la grúa.
- **installationDate**: La fecha de instalación de la grúa. Es un atributo opcional y se representa como una cadena de texto.
- **brand**: La marca de la grúa, representada por el objeto valor `CraneBrand`. Este atributo también es opcional.
- **staff**: El personal asignado a la grúa, representado por la clase `Staff`. Al igual que los atributos anteriores, este atributo puede ser opcional.

Personal

La entidad `Staff` representa al personal del puerto y contiene la información sobre su identificación, nombre, fecha de contratación y su asociación con un puerto y una grúa específica. Presenta las siguientes características:

- **port**: El puerto al que está asociado el personal, representado por la clase `Port`.
- **crane**: La grúa a la que está asignado el personal, representada por la clase `Crane`. Este atributo es opcional, lo que significa que el personal puede o no estar asignado a una grúa específica.
- **dni**: El número de identificación personal del miembro del personal, representado por el objeto valor `StaffDNI`.

- **name**: El nombre del miembro del personal, representado por el objeto valor `StaffName`.
- **surname**: El apellido del miembro del personal. Este atributo es opcional y se representa mediante el objeto valor `StaffSurname`.
- **hireDate**: La fecha de contratación del miembro del personal. Este atributo también es opcional y se representa mediante el objeto valor `StaffHireDate`.

3.2.2. Servicios del dominio

Los servicios del dominio son componentes clave en el desarrollo de la lógica de negocio de una aplicación. Estas clases se encargan de ejecutar acciones específicas y se adhieren a los principios **SOLID**[4] para crear un modelo robusto, escalable y fácilmente testeable mediante el uso de pruebas unitarias.

Los servicios del dominio encapsulan la lógica compleja que no pertenece directamente a entidades u objetos de valor específicos, por ello su diseño busca que cada método se enfoque en una tarea específica y permita una modificación o ampliación sencilla sin afectar otras partes del sistema.

Servicio de los Bloques

La clase `BlockService` se encarga de encapsular la lógica de negocio relacionada con los bloques así como la interacción con el almacenamiento de datos. Se comunica con el repositorio `BlockRestRepository` para realizar operaciones CRUD en los bloques.

- `findAll()`: Retorna una promesa que se resuelve con una matriz de objetos `Block`, obtenidos mediante el método `findAll` del repositorio `BlockRestRepository`.
- `findById(id: string)`: Retorna una promesa que se resuelve con un objeto `Block` correspondiente al `id` especificado.
- `findByPosition(row: number, column: number)`: Retorna una promesa con la posición del bloque dentro del puerto partiendo de las filas y columnas solicitadas.
- `createBlock(block: Block)`: Retorna una promesa que se resuelve con el bloque creado, obtenido mediante el método `createBlock` del repositorio `BlockRestRepository`, pasando el objeto `Block` como argumento.
- `updateBlock(id: string, block: Block)`: Retorna una promesa que se resuelve con el bloque actualizado, obtenido mediante el método `updateBlock` del repositorio `BlockRestRepository`, pasando el objeto `Block` actualizado y el `id` como argumentos.
- `deleteBlock(id: string)`: Elimina un bloque por su `id` mediante el método `deleteBlock` del repositorio `BlockRestRepository`.

Servicio de los Contenedores

El servicio `ContainerService` es el encargado de gestionar las operaciones relacionadas con los contenedores y utiliza el repositorio `ContainerRestRepository` para ello. Tiene los siguientes métodos:

- `findAll()`: Retorna una promesa que se resuelve con una matriz de objetos `Container`, obtenidos mediante el método `findAll` del repositorio `ContainerRestRepository`.
- `findAllPaginated(size: number, page: number)`: Retorna una promesa que se resuelve con un objeto `ContainerPage` que contiene la información paginada de los contenedores, pasando el tamaño de la página y el número de página como argumentos.
- `findById(id: string)`: Retorna una promesa que se resuelve con un objeto `Container` correspondiente al `id` especificado, obtenido mediante el método `findById` del repositorio `ContainerRestRepository`.
- `findByPosition(row: number, column: number)`: Retorna una promesa que se resuelve con un objeto `Container` con la situación del contenedor dentro del bloque, obtenido mediante el método `findByPosition` del repositorio ContainerRestRepository.
- `createContainer(container: Container)`: Retorna una promesa que se resuelve con el contenedor creado, obtenido mediante el método `createContainer` del repositorio `ContainerRestRepository`, pasando el objeto `Container` como argumento.
- `updateContainer(id: string, container: Container)`: Retorna una promesa que se resuelve con el contenedor actualizado, obtenido mediante el método `updateContainer` del repositorio `ContainerRestRepository`, pasando el objeto `Container` actualizado y el `id` como argumentos.
- `deleteContainer(id: string)`: Elimina un contenedor por su `id` mediante el método `deleteContainer` del repositorio `ContainerRestRepository`.

Servicio de las Grúas

El servicio `CraneService` se comunica con el repositorio de grúas `CraneRestRepository`. Sus métodos permiten realizar operaciones de consulta, creación, actualización y eliminación relacionadas con las grúas.

- `findAll()`: Retorna una promesa que resuelve en un arreglo de objetos `Crane`, representando todas las grúas existentes en la aplicación.
- `findAllPaginated(size: number, page: number)`: Retorna una promesa que resuelve en un objeto `CranePage`, el cual contiene una página de grúas en una lista paginada. El tamaño de la página y el número de página se especifican como parámetros.
- `findById(id: string)`: Retorna una promesa que resuelve en un objeto `Crane` correspondiente a la grúa con el ID especificado.

- `createCrane(crane: Crane)`: Retorna una promesa que resuelve en un objeto `Crane`, representando la grúa creada en la aplicación. Se debe proporcionar un objeto `Crane` como parámetro.
- `updateCrane(id: string, crane: Crane)`: Retorna una promesa que resuelve en un objeto `Crane`, representando la grúa actualizada con el ID y los datos de la grúa especificados.
- `deleteCrane(id: string)`: Se utiliza para eliminar la grúa con el ID dado.

Servicios del Personal

El servicio `StaffService` es responsable de manejar la lógica correspondiente con el personal en la aplicación, tiene acceso al repositorio `StaffRestRepository` para realizar operaciones de consulta, creación, actualización y eliminación.

- `findAll()`: Retorna una promesa que resuelve en un arreglo de objetos `Staff`, representando a todo el personal existente en la aplicación.
- `findAllPaginated(size: number, page: number)`: Retorna una promesa que resuelve en un objeto `StaffPage`, el cual contiene una página de personal en una lista paginada. Los parámetros `size` y `page` determinan el tamaño de la página y el número de página, respectivamente.
- `findById(id: string)`: Retorna una promesa que resuelve en un objeto `Staff` correspondiente al personal con el ID especificado.
- `createStaff(staff: Staff)`: Retorna una promesa que resuelve en un objeto `Staff`, representando al personal creado en la aplicación. Se debe proporcionar un objeto `Staff` como parámetro.
- `updateStaff(id: string, staff: Staff)`: Retorna una promesa que resuelve en un objeto `Staff`, representando al personal actualizado con el ID y los datos de personal especificados.
- `deleteStaff(id: string)`: Se emplea para eliminar al personal mediante el ID especificado.

3.2.3. Interfaces de los repositorios

Es responsabilidad del dominio definir las interfaces que los repositorios deben implementar. De esta manera, se mantiene agnóstico con respecto al sistema de persistencia utilizado, pero establece los requisitos que se deben seguir que el dominio funcione correctamente, garantizando la independencia y que sea la capa de persistencia la que dependa del dominio. Las siguientes interfaces determinan cómo se recuperan o almacenan los datos:

PortRepository

Se emplea para establecer la lógica que deben seguir las implementaciones de repositorio relacionadas con la entidad `Port`.

- `findAll()`: Devuelve una promesa que resuelve en un array de objetos `Port`. Se utiliza para obtener todos los puertos existentes en el repositorio.
- `findById(id: string)`: Recibe un `id` como parámetro y devuelve una promesa que resuelve en un objeto `Port` correspondiente al `id` proporcionado. Si no se encuentra ningún puerto con ese `id`, devuelve undefined.
- `findByName(name: PortName)`: Recibe un objeto `PortName` como parámetro y devuelve una promesa que resuelve en un objeto `Port` correspondiente al nombre proporcionado. Si no se encuentra ningún puerto con ese nombre, devuelve undefined.
- `createPort(port: Port)`: Recibe un objeto `Port` como parámetro y devuelve una promesa que resuelve en el objeto `Port` creado en el repositorio.
- `updatePort(id: string, updatedPort: Port)`: Recibe un `id` y un objeto `Port` como parámetros. Actualiza el objeto `Port` correspondiente al `id` proporcionado y lo devuelve.
- `deletePort(id: string)`: Recibe un `id` como parámetro y borra el objeto `Port` correspondiente al `id` proporcionado.

BlockRepository

Se emplea para establecer la lógica que deben seguir las implementaciones de repositorio relacionadas con la entidad `Block`.

- `findAll()`: Devuelve una promesa que se resuelve en un array de objetos `Block`. Se utiliza para obtener todos los bloques del repositorio.
- `findById(id: string)`: Recibe un parámetro `id` de tipo `string` y devuelve una promesa que se resuelve en un objeto `Block` correspondiente al `id` proporcionado. Se utiliza para buscar un bloque por su identificador único.
- `findByPosition(row: number, column: number)`: Se utiliza para buscar un bloque por su posición. Recibe dos parámetros `row` y `column`, ambos de tipo `number`, y devuelve una promesa que se resuelve en un objeto `Block` correspondiente a la posición de fila y columna proporcionadas.
- `createBlock(block: Block)`: Se emplea para crear un nuevo bloque. Recibe un parámetro `block` de tipo `Block`, devuelve el objeto `Block` creado en el repositorio.
- `updateBlock(id: string, block: Block)`: Recibe dos parámetros, un `id` de tipo `string` y un `block` de tipo `Block`. Devuelve el objeto `Block` actualizado.
- `deleteBlock(id: string)`: Recibe un parámetro `id` de tipo `string` y borra el bloque correspondiente al `id` proporcionado.

ContainerRepository

La interfaz `ContainerRepository` define los métodos y atributos que deben implementarse en un repositorio de contenedores.

- `findAllPaginated(size: number, page: number)`: Recibe dos parámetros, `size` y `page`, que representan el tamaño de página y el número de página, respectivamente. Devuelve una promesa que se resuelve en un objeto `ContainerPage` que contiene una lista paginada de contenedores.
- `findById(id: string)`: Recibe un parámetro `id` de tipo `string`. Este método se utiliza para buscar un contenedor por su identificador único, devolviendo como resultado un objeto `Container` correspondiente al `id` proporcionado.
- `findByPosition(row: number, column: number)`: Recibe dos parámetros, `row` y `column`, que representan la posición de fila y columna del contenedor. Devuelve una promesa que se resuelve en un objeto `Container` correspondiente a la posición proporcionada.
- `createContainer(container: Container)`: Recibe un parámetro `container` de tipo `Container` y crea un nuevo objeto `Container` al resolverlo.
- `updateContainer(id: string, container: Container)`: Recibe dos parámetros, un `id` de tipo `string` y un `container` de tipo `Container`, utilizando el contenedor existente en el repositorio.
- `deleteContainer(id: string)`: Recibe un parámetro `id` de tipo `string` y borra el contenedor correspondiente al `id` proporcionado.

CraneRepository

Esta interfaz define los métodos disponibles para acceder y manipular objetos de tipo `Crane` en un repositorio.

- `findAll()`: Este método se utiliza para obtener todas los grúas almacenados en el repositorio.
- `findAllPaginated(size: number, page: number)`: Recibe dos parámetros, `size` y `page`, que representan el tamaño de página y el número de página, respectivamente. Devuelve una promesa que se resuelve en un objeto `CranePage`, que contiene una lista paginada de grúas.
- `findById(id: string)`: Recibe un parámetro `id` de tipo `string` y devuelve una promesa que se resuelve en un objeto `Crane` correspondiente al `id` proporcionado, buscar una grúa por su identificador.
- `createCrane(crane: Crane)`: Recibe un parámetro `crane` de tipo `Crane` y devuelve una promesa que devuelve el objeto `Crane` creado en el repositorio.
- `updateCrane(id: string, crane: Crane)`: Recibe dos parámetros, un `id` de tipo `string` y `crane` de tipo `Crane`, y devuelve una promesa con el objeto `Crane` actualizado.
- `deleteCrane(id: string)`: Recibe un parámetro `id` de tipo `string` y borra la grúa correspondiente al `id` proporcionado.

StaffRepository

Con esta interfaz se establece la lógica que deben seguir las implementaciones de repositorio relacionadas con la entidad `Staff`.

- `findAll()`: Devuelve una promesa que se resuelve en un array de objetos `Staff`. Este método se utiliza para obtener todos los miembros del personal.
- `findAllPaginated(size: number, page: number)`: Recibe dos parámetros, `size` y `page`, que representan el tamaño de página y el número de página, respectivamente. Devuelve una lista paginada con los miembros del personal.
- `findById(id: string)`: Recibe un parámetro `id` de tipo `string` y devuelve una promesa que se resuelve en un objeto `Staff` correspondiente al `id` proporcionado.
- `createStaff(staff: Staff)`: Recibe un parámetro `staff` de tipo `Staff`. Este método se utiliza para crear un nuevo miembro del personal.
- `updateStaff(id: string, staff: Staff)`: Recibe dos parámetros, un `id` de tipo `string` y un `staff` de tipo `Staff`. Devuelve el objeto `Staff` actualizado.
- `deleteStaff(id: string)`: Recibe un parámetro `id` de tipo `string` y borra el miembro del personal correspondiente al `id` dado.

3.2.4. Componentes

Los componentes de frontend se componen de código **HTML**, **CSS** y **Typescript**[8], junto con Vuetify y se utilizan para construir la interfaz de usuario de la aplicación web. Estos pueden ser desde botones o menús hasta elementos más complejos, como indicadores o cubos para representar las pilas de contenedores del puerto. Algunos componentes son interactivos, ejecutando el método correspondiente que utiliza el enrutador de Vue para redirigir al usuario a la ruta especificada.

UserInteractionComponents

El siguiente componente compone el header de la aplicación, figura 3.3. Dispone de varios botones destinados a facilitar al usuario su navegación por las distintas secciones de la aplicación. La vista por bloques y los listados de contenedores, personal y grúas de un puerto específico. Además, hay un botón “Home“ que redirige al usuario a la página de principal.

Figura 3.3: Componente UserInteractionHeader

Otros componentes del tipo `UserInteraction` tienen la función de mostrar una barra de navegación en la parte superior de la aplicación que contiene el título de la sección en la que se encuentra el usuario y uno o varios botones que permiten navegar hacia atrás en la jerarquía de la app.

Figura 3.4: Componente UserInteraction

CircularStatisticsComponents

Este tipo de componentes representa un círculo compuesto por barras verticales, muestra una estadística con una animación de incremento gradual. Este valor numérico indicar el progreso y el porcentaje ocupado de cada entidad del puerto.

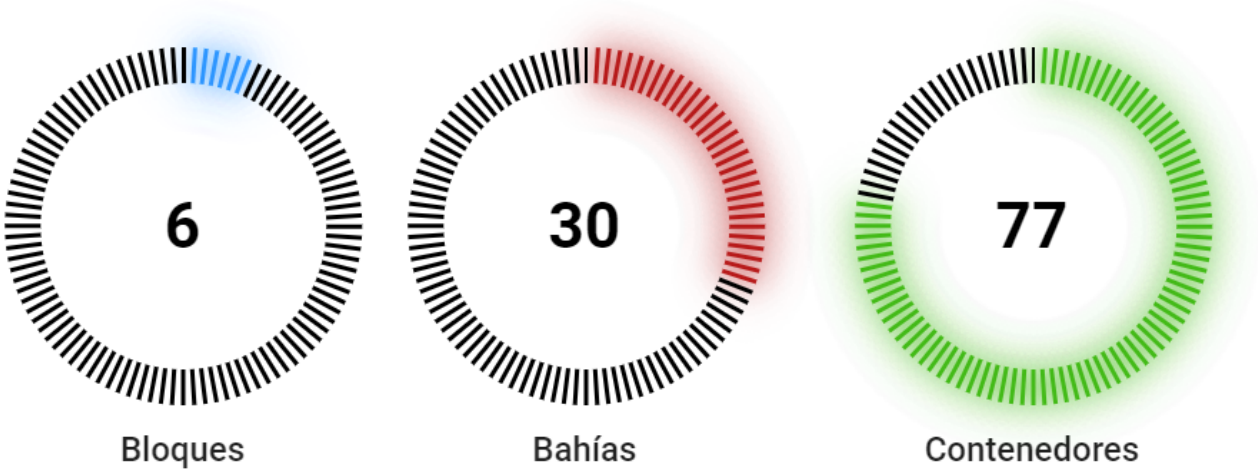


Figura 3.5: Componente CircularStatistics

BlockComponent

Representa un bloque del puerto de forma visual en la interfaz de usuario. Cada bloque tiene un número de bloque y muestra una serie de elementos en forma de cuadrados que representan las pilas de contenedores del mismo. Al hacer clic, se redirige al usuario a una vista detallada del bloque y sus contenedores asociados, figura 3.6.

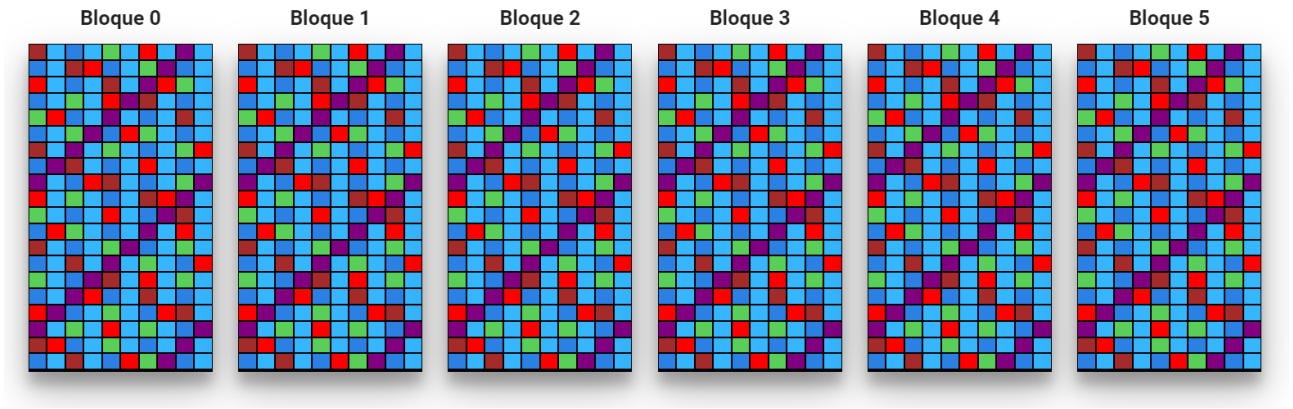


Figura 3.6: Componente Block

CubeBayComponent

Se emplea para representar visualmente las pilas de contenedores en un puerto. La apariencia de la pila de contenedores varía según la cantidad de contenedores que tenga, y se refleja mediante el color de las caras del cubo. Ofrece al usuario información visual en un contexto relacionado con la gestión de contenedores y puertos 3.7.

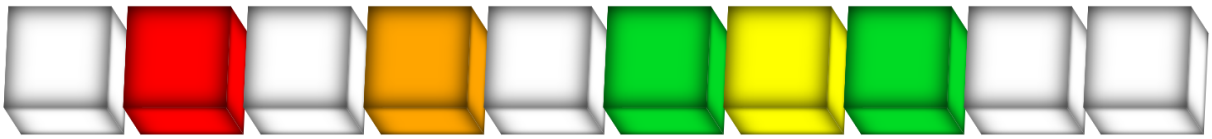
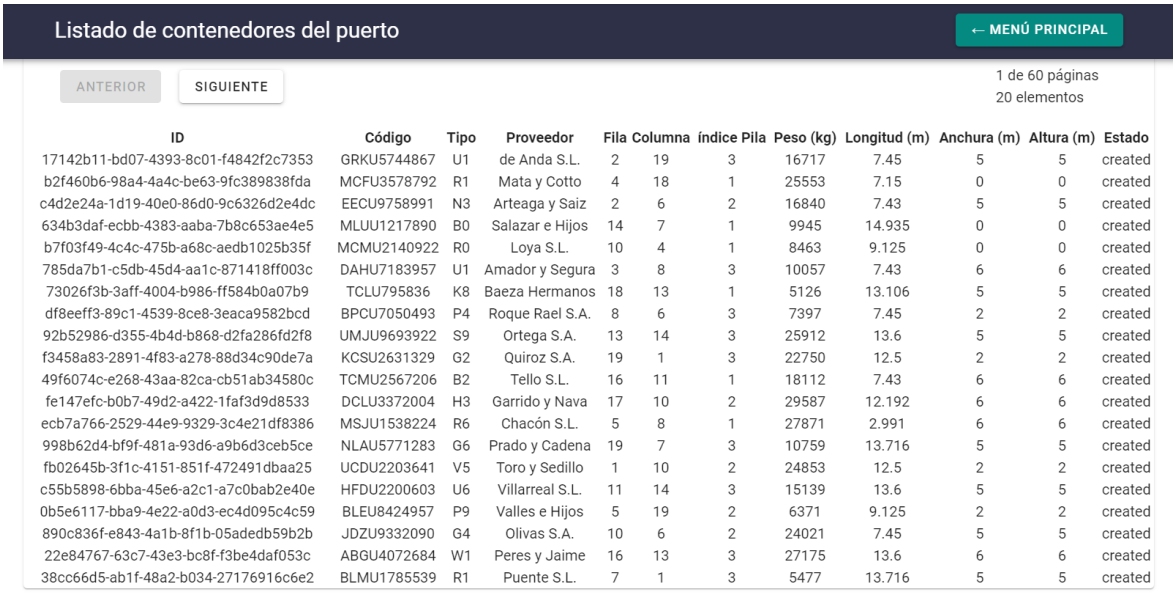


Figura 3.7: Componente pilas bahía

ListComponent

Los componentes del tipo `ListComponent` proporcionan información paginada de los registros tanto de contendores, como de grúas o personal de un puerto. Permiten navegar entre las páginas de la tabla y ajustar el número de registros mostrados en las mismas. Es útil para visualizar y administrar la información relacionada con el puerto 3.8.



ID	Código	Tipo	Proveedor	Fila	Columna	índice Pila	Peso (kg)	Longitud (m)	Anchura (m)	Altura (m)	Estado
17142b11-bd07-4393-8c01-f4842f2c7353	GRKU5744867	U1	de Anda S.L.	2	19	3	16717	7.45	5	5	created
b2f460b6-98a4-4a4c-be63-9fc389838fda	MCFU3578792	R1	Mata y Cotto	4	18	1	25553	7.15	0	0	created
c4d2e24a-1d19-40e0-86d0-9c6326d2e4dc	EECU9758991	N3	Arteaga y Saiz	2	6	2	16840	7.43	5	5	created
634b3daf-ecbb-4383-aaba-7b8c653ae4e5	MLUU1217890	B0	Salazar e Hijos	14	7	1	9945	14.935	0	0	created
b7f03f49-4c4c-475b-a68c-aedb1025b35f	MCMU2140922	R0	Loya S.L.	10	4	1	8463	9.125	0	0	created
785da7b1-c5db-45d4-aa1c-871418ff003c	DAHU7183957	U1	Amador y Segura	3	8	3	10057	7.43	6	6	created
73026f3b-3aff-4004-b986-ff584b0a07b9	TCLU795836	K8	Baeza Hermanos	18	13	1	5126	13.106	5	5	created
df8eeff3-89c1-4539-8ce8-3eaca9582bcd	BPCU7050493	P4	Roque Rael S.A.	8	6	3	7397	7.45	2	2	created
92b52986-d355-4b4d-b868-d2fa286fd2f8	UMJU9693922	S9	Ortega S.A.	13	14	3	25912	13.6	5	5	created
f3458a83-2891-4f83-a278-88d34c90de7a	KCSU2631329	G2	Quiroz S.A.	19	1	3	22750	12.5	2	2	created
49f6074c-e268-43aa-82ca-cb51ab34580c	TCMU2567206	B2	Tello S.L.	16	11	1	18112	7.43	6	6	created
fe147efc-b0b7-49d2-a422-1faf3d9d8533	DCLU3372004	H3	Garrido y Nava	17	10	2	29587	12.192	6	6	created
ecb7a766-2529-44e9-9329-3c4e21df8386	MSJU1538224	R6	Chacón S.L.	5	8	1	27871	2.991	6	6	created
998b62d4-bf9f-481a-93d6-a9b6d3ceb5ce	NLAU5771283	G6	Prado y Cadena	19	7	3	10759	13.716	5	5	created
fb02645b-3f1c-4151-851f-472491dbaa25	UCDU2203641	V5	Toro y Sedillo	1	10	2	24853	12.5	2	2	created
c55b5898-6bba-45e6-a2c1-a7c0bab2e40e	HFDU2200603	U6	Villarreal S.L.	11	14	3	15139	13.6	5	5	created
0b5e6117-bba9-4e22-a0d3-ec4d095c4c59	BLEU8424957	P9	Valles e Hijos	5	19	2	6371	9.125	2	2	created
890c836f-e843-4a1b-8f1b-05adedb59b2b	JDZU9332090	G4	Olivas S.A.	10	6	2	24021	7.45	5	5	created
22e84767-63c7-43e3-bc8f-f3be4daf053c	ABGU4072684	W1	Peres y Jaime	16	13	3	27175	13.6	6	6	created
38cc66d5-ab1f-48a2-b034-27176916c6e2	BLMU1785539	R1	Puente S.L.	7	1	3	5477	13.716	5	5	created

Figura 3.8: Componente listado Contenedores

LegendComponent

Este componente cumple la función de leyenda proporcionando información visual sobre los colores utilizados en la representación de las bahías donde cada uno representa un nivel de densidad de contenedores dentro de las pilas. Esto ayuda al usuario a interpretar rápidamente la cantidad de contenedores asociados a cada pila según su color.

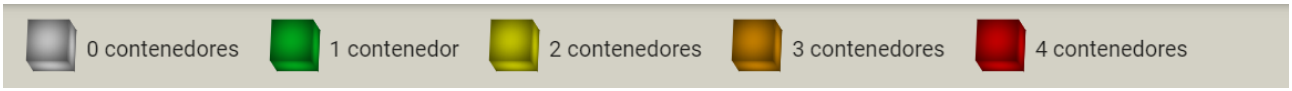


Figura 3.9: Componente leyenda de la bahía

3.3. Backend

El desarrollo backend es un componente crucial en un sistema, en este proyecto estará gestionando las mercancías, maquinarias, así como el personal del puerto. Para llevar a cabo esta tarea, se utilizará Spring Boot[6] como framework para el lenguaje de programación Java.

Java sigue siendo ampliamente utilizado en la actualidad en diversos ámbitos de la industria del desarrollo de software como:

- **Desarrollo de aplicaciones empresariales.**
- **Desarrollo de aplicaciones móviles.**
- **Desarrollo de aplicaciones web.**
- **Big Data y procesamiento de datos.**
- **Juegos y gráficos.**

Esto se debe a su portabilidad, seguridad y el gran ecosistema de herramientas y frameworks disponibles. Además, según los resultados de la encuesta basados en los votos de desarrolladores profesionales en el Stack Overflow Developer Survey 2023 [3], Java es uno de los lenguajes punteros, figura 3.10.

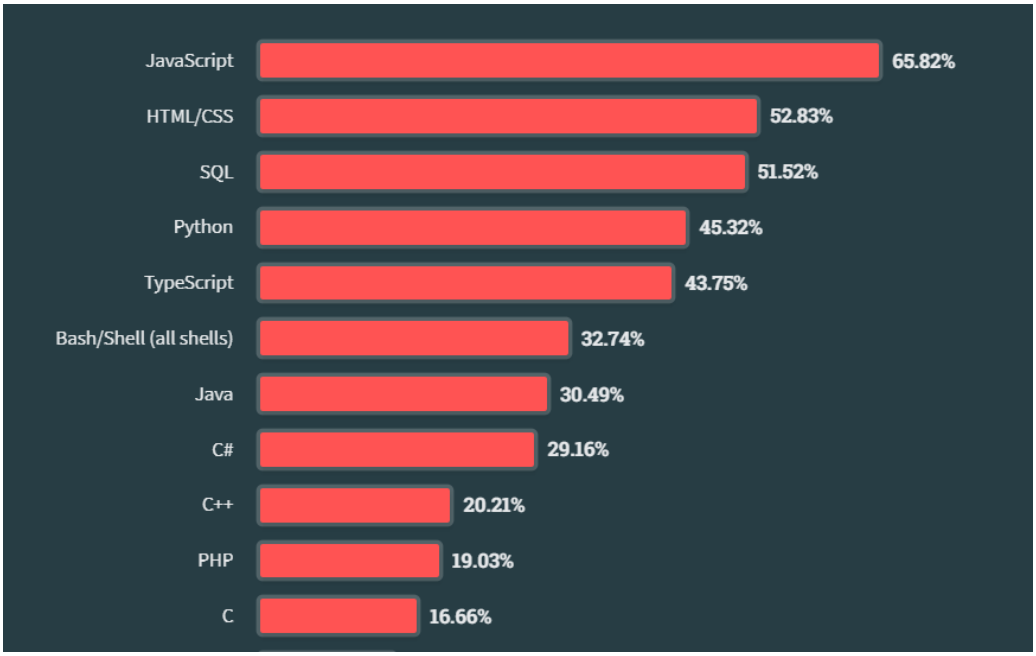


Figura 3.10: Stack Overflow Developer Survey 2023

[3] https://survey.stackoverflow.co/2023/

Por su parte, Spring Boot simplifica y acelera el proceso de creación de aplicaciones robustas y escalables. Proporciona un conjunto de herramientas y bibliotecas que facilitan la implementación de características comunes en el desarrollo de backend, como la gestión de bases de datos, la seguridad, el manejo de solicitudes HTTP y la integración con otros servicios[4].

El framework Spring hace uso de una herramienta de gestión de dependencias conocida como **Maven**[5]. Esta permite la construcción de proyectos a través de su modelo de objeto de proyecto (POM), un archivo XML que contiene información sobre el proyecto y los detalles de configuración que Maven necesita para el desarrollo del mismo. Esta herramienta se ha convertido en un estándar permitiendo declarar dependencias en bibliotecas de código abierto alojadas en un repositorio.

También existen otras herramientas como **Gradle**[6] para la automatización de la compilación. Este utiliza un lenguaje de dominio específico (DSL) basado en Groovy[7] o Kotlin[8] para definir las tareas y configuraciones del proyecto.

3.3.1. Diagrama de clases

Como se ha comentado con anterioridad, las entidades de dominio son independientes de cualquier tecnología o infraestructura específica, y se centran en la representación precisa y coherente de los elementos del dominio.

- La entidad `Port` representa la información relevante de un puerto, como su nombre, ubicación geográfica y otros detalles específicos del puerto.
- La entidad `Block` encapsula la información relevante de un bloque en el puerto, como su ubicación, altura de las pilas y otros detalles relacionados con el bloque.
- La entidad `Container` representa un contenedor en el dominio de la aplicación. Contiene información relacionada con el contenedor, como su código, tipo, proveedor, ubicación en el bloque y estado.
- La entidad `Crane` se utiliza para modelar y gestionar las grúas en el sistema. Proporciona información sobre su ubicación, identificación, marca y otros detalles específicos de la grúa.
- La entidad `Staff` representa al personal del puerto y contiene información sobre su identificación, nombre, fecha de contratación y su asociación con un puerto y una grúa específica.

El diagrama 3.11 muestra la interconexión existente entre las distintas clases a través de sus asociaciones.

[8]Herramientas y bibliotecas en Spring Boot.
[8]`https://maven.apache.org`
[8]`https://gradle.org`
[8]`https://groovy-lang.org`
[8]`https://kotlinlang.org`

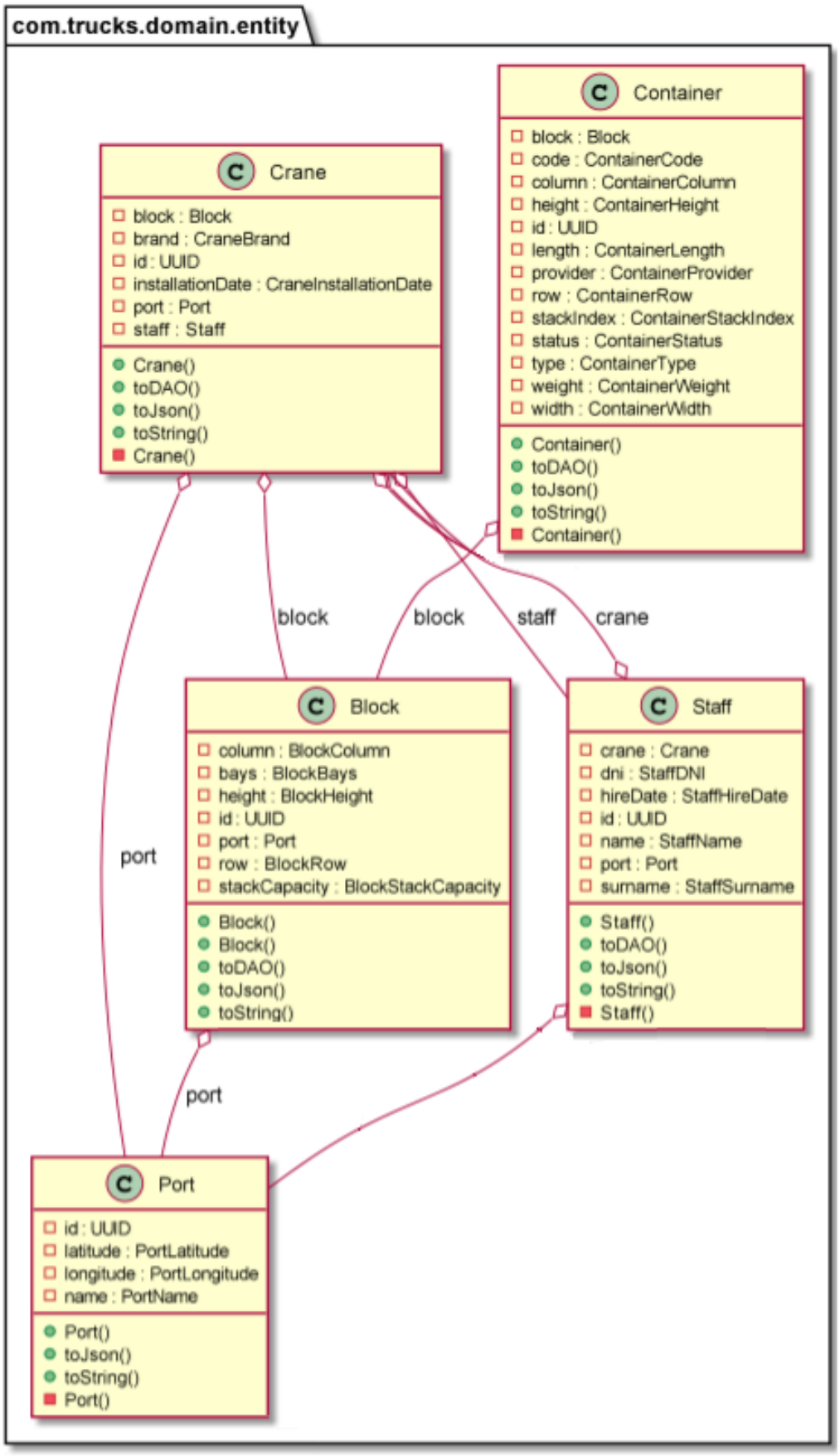


Figura 3.11: Diagrama de clases

3.3.2. DAO

Las entidades de dominio representan los conceptos y la lógica del negocio de una aplicación, mientras que los **DAO**[9] proporcionan una capa de abstracción para acceder y manipular los datos relacionados con esas entidades de dominio.

Por lo general, una clase DAO es responsable de dos conceptos: encapsular los detalles de la capa de persistencia y proporcionar una interfaz CRUD para una entidad. Aportan una capa de abstracción sobre la lógica de acceso a los datos y oculta los detalles de implementación específicos de la base de datos, es fundamental para crear consultas y realizar solicitudes.

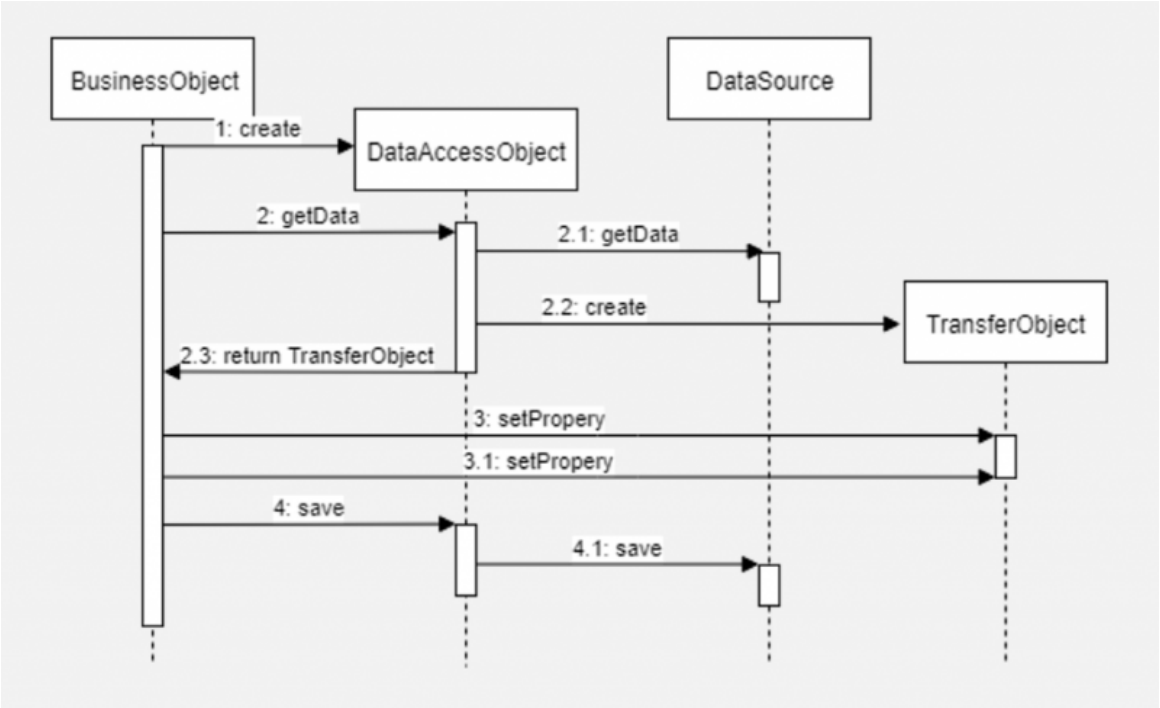


Figura 3.12: DAO Pattern

En la figura anterior 3.12 `BusinessObject`, representa una entidad de la lógica de negocio e interactúa con el `DataAccessObject` (DAO), para obtener o modificar datos, posteriormente este se encarga de acceder al `DataSource`, en nuestro caso particular MongoDB, y obtiene los datos solicitados. Finalmente, el DAO crea un objeto capaz de transportar la información entre las distintas capas de la aplicación llamado `TransferObject` (DTO)[10].

BlockDAO

La clase `BlockDAO` es similar a la entidad `Block`, pero con la diferencia de que no tiene un atributo de tipo `Port`, en su lugar tiene un atributo `portID` de tipo **UUID**[9] que representa el identificador del puerto al que pertenece el bloque. Esto permite almacenar en la base de datos solo el identificador del puerto y no todo el objeto `Port`. Para poder recuperar el objeto `Port` completo, se puede utilizar el método `getPortFromDAO` del repositorio `MongoDbBlockRepository`.

El método `getPortFromDAO` utiliza el `mongoTemplate` para realizar una consulta en la base de datos MongoDB. Se crea una consulta con un criterio que coincide con el

[9] `https://www.baeldung.com/java-uuid`

identificador del puerto (`portID`) del `BlockDAO`. Luego, se utiliza el método `findOne` del `mongoTemplate` para obtener el objeto `Port` correspondiente al identificador del puerto. El resultado se devuelve como un objeto de tipo `Port`. Esto permite recuperar el objeto `Port` completo a partir del identificador almacenado en el `BlockDAO`.

Cuenta con otro método fundamental para la actualización de las bahías llamado `updateBaysInfo`, su propósito es actualizar la información de las bahías en un bloque. Para ello se crea una query, que se utiliza para construir una consulta y buscar aquellos contenedores contenedores que tengan el mismo blockID que el bloque actual ubicados en una fila y columna específica.

```
public Block updateBaysInfo (Block block) {
    logger.info("Updating bays info for block {}", block.getId());
    for (int i = 0; i < 20; i++) {
        for (int j = 0; j < 10; j++) {
            Query query = new Query();
            query.addCriteria(Criteria.where("blockID").is(block.getId()));
            query.addCriteria(Criteria.where("row").is(i+1).and("column").is(j+1));
            long currentQuantity = this.mongoTemplate.count(query, ContainerDAO.class,
                MongoDbContainerRepository.COLLECTION_NAME);
            block.getBay(i).getValue().get(j).setCurrentStack(currentQuantity);
        }
    }
    return block;
}
```

ContainerDAO

Un caso parecido al anterior ocurre con la clase `ContainerDAO`. Esta es similar a la entidad `Container`, solo que almacena el UUID del bloque en el que se encuentra dicho contenedor. Para recuperar el objeto `Block` completo a partir del `ContainerDAO`, se utiliza el método `getBlockFromDAO` del repositorio `MongoDbContainerRepository` con un funcionamiento similar al descrito anteriormente.

```
public Block getBlockFromDAO (ContainerDAO container) {
    Query query = new Query();
    query.addCriteria(Criteria.where("_id").is(container.getBlockID()));
    BlockDAO block = this.mongoTemplate.findOne(query, BlockDAO.class,
        MongoDbBlockRepository.COLLECTION_NAME);
    Port port = this.mongoTemplate.findOne(new Query().addCriteria(Criteria.where("_id").
        is(block.getPortID())), Port.class, MongoDbPortRepository.COLLECTION_NAME);
    Block blockEntity = block.toBlock();
    blockEntity.setPort(port);
    return blockEntity;
}
```

CraneDAO

La clase `CraneDAO` es similar a las anteriores en términos de estructura, pero en lugar de tener objetos completos de `Port`, `Block` y `Staff`, almacena sus identificadores como objetos `UUID`. Esto permite reducir la cantidad de información almacenada en la base de datos y, posteriormente, recuperarlos utilizando los métodos correspondientes

de `MongoDbCraneRepository`.

Para recuperar los objetos completos a partir de un objeto `CraneDAO`, se utilizan los siguientes métodos del repositorio `MongoDbCraneRepository`:

- `getPortFromDAO`: Recupera el objeto `Port` completo a partir del identificador de puerto almacenado en el objeto `CraneDAO`.
- `getBlockFromDAO`: Recupera el objeto `Block` completo a partir del identificador de bloque almacenado en el objeto `CraneDAO`.
- `getStaffFromDAO`: Recupera el objeto `Staff` completo a partir del identificador de personal almacenado en el objeto `CraneDAO`.

StaffDAO

Finalmente, la clase `StaffDAO` sigue el mismo patrón que los casos anteriormente expuestos. Guarda identificadores como objetos `UUID` en lugar de tener objetos completos de `Port` y `Crane`.

Para recuperar los objetos completos a partir de un objeto `StaffDAO`, se utilizan los siguientes métodos:

- `getPortFromDAO` empleado para obtener el objeto `Port` completo.
- `getCraneFromDAO` que proporciona el objeto `Crane` completo.

```
public Crane getCraneFromDAO(StaffDAO staffDAO) {
    if (staffDAO.getCrane().isEmpty()) {
        return null;
    }
    Query query = new Query();
    query.addCriteria(Criteria.where("_id").is(staffDAO.getCrane().get()));
    return this.mongoTemplate.findOne(query, Crane.class, MongoDbCraneRepository.
        COLLECTION_NAME);
}
```

El aspecto destacado de este método es que para la entidad Crane, Staff es de tipo Optional, por tanto se debe hacer una comprobación para verificar si la referencia a la grúa en el objeto staffDAO está vacía.

3.3.3. Peticiones y Endpoints

En la puesta en marcha del backend, las **operaciones CRUD**[10] son fundamentales para interactuar con una base de datos y gestionar los datos de manera eficiente. CRUD es un acrónimo que representa las cuatro operaciones básicas: Crear (Create), Leer (Read), Actualizar (Update) y Borrar (Delete).

Cuando hablamos del desarrollo de aplicaciones web, los **endpoints** son puntos de acceso o URLs específicas a través de los cuales los clientes o usuarios pueden interactuar con la aplicación. Indican las diferentes operaciones que se pueden realizar, como obtener información, enviar datos, actualizar registros o eliminar contenido según se muestra en la tabla 3.2.

[10] `https://www.mongodb.com/docs/manual/crud`

Operaciones CRUD	
Operación	Función
CREATE	Las operaciones de creación o inserción añaden nuevos documentos a una colección. Si la colección no existe actualmente, las operaciones de inserción crearán la colección.
READ	Las operaciones de lectura recuperan documentos de una colección; es decir, consultar una colección de documentos.
UPDATE	Las operaciones de actualización modifican los documentos existentes en una colección.
DELETE	Las operaciones de eliminación eliminan documentos de una colección.

Tabla 3.2: Operaciones CRUD

Métodos HTTP	
Operación	Función
GET	Se utiliza para solicitar y obtener datos de un servidor. Al realizar una solicitud GET a un endpoint específico, el servidor responderá proporcionando los datos solicitados.
POST	Se utiliza para enviar datos al servidor para su procesamiento. Al realizar una solicitud POST a un endpoint, se envía un conjunto de datos en el cuerpo de la solicitud, y el servidor realiza la acción correspondiente, como crear un nuevo recurso o procesar una transacción.
PUT	Se utiliza para actualizar o reemplazar un recurso existente en el servidor.
DELETE	Se utiliza para eliminar un recurso existente en el servidor.

Tabla 3.3: Métodos HTTP

Cada endpoint corresponde con una ruta específica en el servidor, al acceder a una a través de una solicitud HTTP, se desencadena una acción en la aplicación. Los diferentes métodos HTTP indican una diferente acción a realizar en el servidor como se puede apreciar en la siguiente referencia 3.3

Para el desarrollo de este proyecto, se han definido endpoints que permiten realizar operaciones CRUD en la base de datos. Cada uno corresponde con una ruta única en el servidor, y al realizar una solicitud HTTP específica a través de esa ruta, se desencadena una acción en la aplicación.

- Endpoints para la entidad puerto, figura 3.4.
- Endpoints para la entidad bloque, figura 3.5.
- Endpoints para la entidad de los contenedores, figura 3.6.
- Endpoints para la entidad de las grúas, figura 3.7.
- Endpoints para la entidad del personal, figura 3.8.

Puerto	
Tipo	Función
GET	`/ports` - Devuelve la información de todos los puertos
	`/ports/{id}` - Devuelve la información del puerto solicitado según el identificador
POST	`/ports` - Se establece un nuevo puerto con los datos introducidos.
UPDATE	`/ports/{id}` - Actualiza campos del puerto solicitado por su identificador.
DELETE	`/ports/{id}` - Elimina la información del puerto solicitado por su identificador.

Tabla 3.4: Solicitudes HTTP del puerto

Bloques	
Tipo	Función
GET	`/blocks` - Devuelve la información de todos los bloques
	`/blocks/{id}` - Devuelve la información del bloque solicitado según el identificador
POST	`/blocks` - Se establece un nuevo bloque con los datos introducidos.
UPDATE	`/blocks/{id}` - Actualiza campos del bloque solicitado por su identificador.
DELETE	`/blocks/{id}` - Elimina la información del bloque solicitado por su identificador.

Tabla 3.5: Solicitudes HTTP de los bloques

Contenedores	
Tipo	Función
GET	`/containers` - Devuelve la información de todos los contenedores
	`/containers/{id}` - Devuelve la información del contenedor solicitado según el identificador
	`/containers?size=20` - Devuelve la información de los contenedores en formato lista de 20 en 20.
	`/containers?row=number&column=number` - Devuelve la información de los contenedores situados en una posición concreta.
POST	`/containers` - Crea un nuevo contenedor con los datos dados.
UPDATE	`/containers/{id}` - Actualiza campos del contenedor solicitado por su identificador.
DELETE	`/containers/{id}` - Elimina la información del contenedor solicitado por su identificador.

Tabla 3.6: Solicitudes HTTP de los contenedores

Grúas	
Tipo	Función
GET	`/cranes` - Devuelve la información de todas las grúas
	`/cranes/{id}` - Devuelve la información de la grúa solicitada según el identificador
POST	`/cranes` - Crea una nueva grúa a partir de los datos proporcionados.
PUT	`/cranes/{id}` - Actualiza los campos de la grúa solicitada por su identificador.
DELETE	`/cranes/{id}` - Elimina la información de la grúa solicitada por su identificador.

Tabla 3.7: Solicitudes HTTP de las grúas

Personal	
Tipo	Función
GET	`/staff` - Devuelve la información de todos los trabajadores
	`/staff/{id}` - Devuelve la información un individuo de entre todo el personal mediante su identificador
POST	`/staff` - Añade un nuevo trabajador a partir de los datos proporcionados.
PUT	`/staff/{id}` - Actualiza los campos del un miembro del personal por su identificador.
DELETE	`/staff/{id}` - Elimina la información de un trabajador por su identificador.

Tabla 3.8: Solicitudes HTTP del personal

3.4. Generador de contenedores

Este proyecto cuenta con un generador de contenedores en tiempo real, estableciéndose una conexión **WebSocket**[11] con un servidor, esta tecnología de comunicación bidireccional en tiempo real permite una interacción continua y fluida entre el cliente y el servidor a través de una conexión persistente.

El generador de contenedores se encarga de mandar de forma periódica y secuencial contenedores con información relevante. Estos son recogidos por el backend, donde se procesarán a través de los endpoints correspondientes. Posteriormente, los datos resultantes serán enviados al frontend para su visualización y manipulación según el esquema de la figura 3.13.

3.4.1. WebSocketService

La clase `WebSocketService` es responsable de establecer una conexión WebSocket al iniciar la aplicación. Utiliza un cliente WebSocket para conectarse a un servidor específico. La conexión se realiza a través del método `doHandshake()` del cliente WebSocket.

El método `WebSocketConnect()` se ejecuta automáticamente después de que se haya creado una instancia de la clase `WebSocketService`, gracias a la anotación `@PostConstruct`.

[11] `https://www.oscarblancarteblog.com/2017/02/20/introduccion-a-lo-websocket`

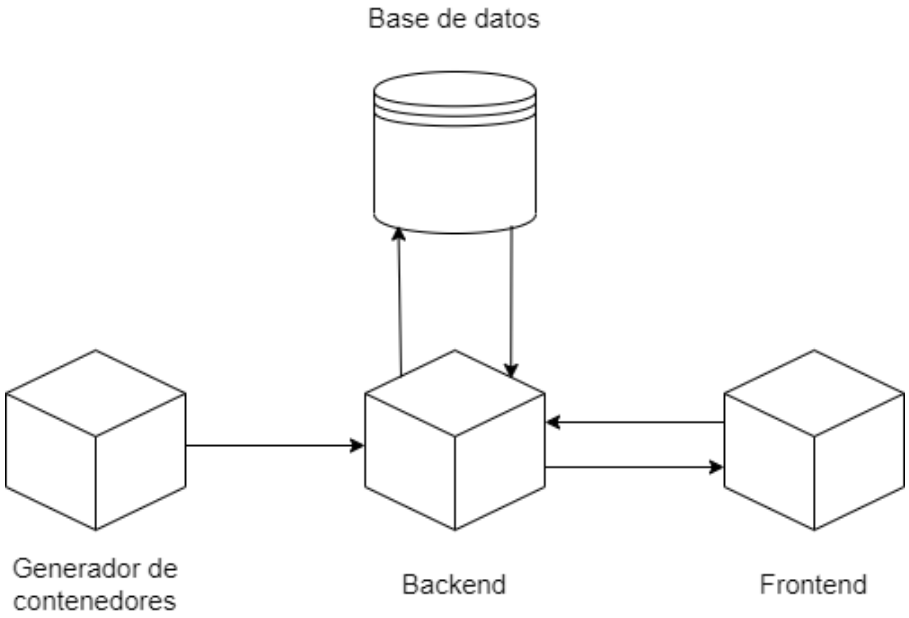


Figura 3.13: Interacción componentes del proyecto

Dentro de este método, se intenta establecer la conexión WebSocket utilizando el cliente WebSocket y el manejador de eventos personalizado `MyWebSocketHandler`.

El `MyWebSocketHandler` es responsable de manejar los eventos que ocurren durante la comunicación WebSocket, como la recepción de mensajes. En este caso, los mensajes recibidos contienen información sobre contenedores. Se encarga de procesar esos mensajes y realizar operaciones correspondientes, como crear o actualizar objetos de contenedor en la base de datos utilizando los servicios `ContainerService` y `PortService`.

3.4.2. WebSocketConfiguration

La clase `WebSocketConfiguration` es una clase de configuración en una aplicación Spring que se encarga de configurar el WebSocket. Esta clase implementa la interfaz `WebSocketConfigurer`, lo que permite definir la configuración del WebSocket en la aplicación.

Dentro de la clase, se definen algunos métodos y se inyectan dependencias utilizando la anotación `@Autowired`. Los dos servicios inyectados son `ContainerService` y `PortService`. El método `registerWebSocketHandlers` se utiliza para registrar el manejador de WebSocket en el `WebSocketHandlerRegistry`.

3.4.3. MyWebSocketHandler

Su función es actuar como un controlador para manejar las conexiones y mensajes en un servidor WebSocket. Procesa los mensajes y realiza las acciones correspondientes, como crear o actualizar objetos `Container`. Además, `MyWebSocketHandler` también se encarga de manejar eventos como la conexión exitosa, los errores de transporte y el cierre de la conexión WebSocket. Por tanto, su finalidad es facilitar la comunicación entre el servidor y el cliente permitiendo la interacción y actualización de datos en tiempo

real.

El método `handleMessage` se invoca cuando se recibe un mensaje como una cadena de texto y se realiza el siguiente proceso:

- Crea un objeto `Container`, en caso contrario, se sale del método.
- Se busca en la base de datos si existe un contenedor previo con el mismo identificador. El resultado se almacena en un objeto `Optional<Container>`.
- Si no existe un contenedor previo, se guarda el nuevo contenedor. Si ya existe un contenedor previo, se actualiza el primer contenedor encontrado.

3.5. Base de Datos

Las bases de datos son una pieza fundamental en cualquier proyecto de desarrollo de software. Su función principal es almacenar, recuperar y manipular datos de manera eficiente y estructurada. Proporcionan un entorno seguro para atesorar la información relacionada con la aplicación, como perfiles de usuarios, registros de transacciones y colecciones. Además, permiten realizar consultas y búsquedas rápidas para acceder a los datos necesarios.

Las bases de datos garantizan la integridad y consistencia de los mismos mediante la aplicación de restricciones y reglas definidas. También son escalables, lo que facilita la capacidad para adaptarse a los cambios que puedan surgir a medida que el proyecto crece, tanto en términos de volumen de datos como de demanda de tráfico. Además, ofrecen mecanismos de optimización para mejorar el rendimiento y la eficiencia de las operaciones.

Para este proyecto se ha seleccionado **MongoDB**[11][12], un sistema de base de datos **NoSQL**[13] orientado a documentos. A diferencia de las bases de datos relacionales, que almacenan datos en tablas, MongoDB almacena estructuras de datos en documentos utilizando el formato **BSON**[14], que es similar a JSON, y utiliza un esquema dinámico. A diferencia de este último, BSON destaca por su capacidad de manejar tipos de datos más avanzados. Por ejemplo, mientras que JSON no distingue entre números enteros y números con precisión decimal, BSON sí lo hace, por ello, se puede almacenar y representar con precisión diferentes tipos numéricos, proporcionando una mayor flexibilidad y exactitud en la manipulación de la información.

Otra diferencia importante es la escalabilidad. Las bases de datos SQL suelen tener dificultades para escalar horizontalmente, lo que significa manejar grandes volúmenes de datos y altas cargas de trabajo distribuyendo la carga entre múltiples servidores. Por otro lado, las bases de datos NoSQL están diseñadas para ser altamente escalables y permiten un crecimiento flexible sin afectar al rendimiento.

Otro aspecto importante es la flexibilidad, las bases de datos SQL requieren un esquema fijo y predefinido, donde se definen las tablas, columnas y tipos de datos. Esto significa que realizar cambios en el esquema existente puede ser complicado. Por el contrario, las bases de datos NoSQL son más flexibles en cuanto al esquema y permiten cambios más sencillos en la estructura de los datos sin requerir modificaciones extensas.

[12] `https://www.mongodb.com`
[13] `https://www.mongodb.com/es/nosql-explained`
[14] `https://www.mongodb.com/json-and-bson`

Aunque existen otras opciones de bases de datos NoSQL, como Apache Cassandra[15], CouchDB[16] o Redis[17], así como bases de datos SQL como PostgreSQL[18], MySQL[19] u Oracle[20] entre otras.

Este proyecto cuenta con una serie de colecciones en MongoDB, están constituidas por un conjunto de documentos almacenados en la base de datos. Cada uno de estos documentos conforman una entidad independiente y representa una unidad de datos. Se representan como pares clave-valor, donde las claves son los campos o atributos del documento y los valores son los datos almacenados. A continuación se aprecian las colecciones existentes:

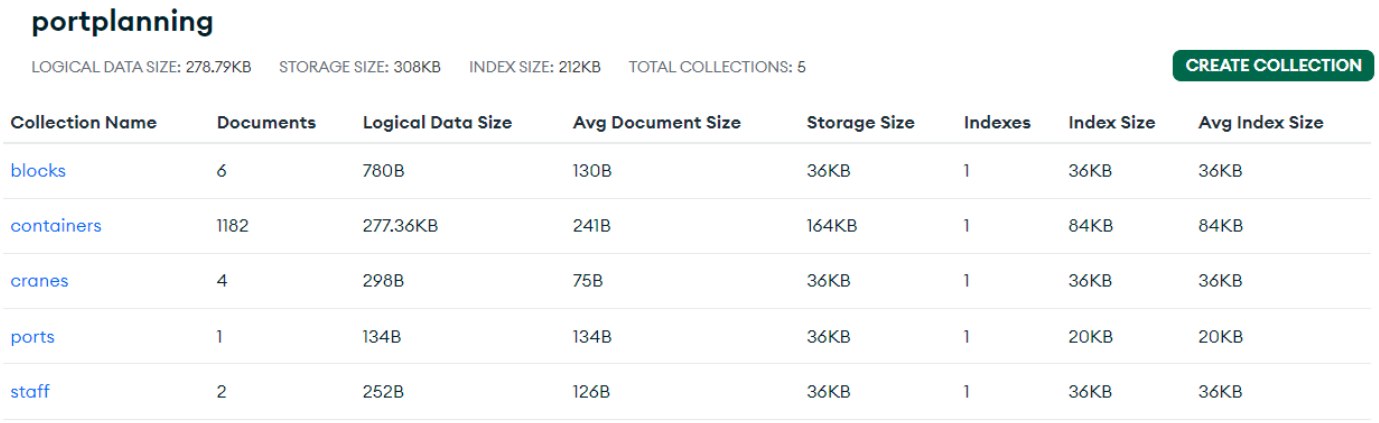


Figura 3.14: Colecciones del proyecto en MongoDB

3.6. Docker

Docker[21] es una herramienta de código abierto que simplifica el despliegue de aplicaciones al utilizar contenedores virtuales. Su enfoque se basa en el uso de imágenes, lo que permite compartir una aplicación o conjunto de servicios con todas sus dependencias en diferentes entornos. Los contenedores son entornos livianos y portables que encapsulan todas las dependencias y configuraciones necesarias para que una aplicación se ejecute de manera consistente en diferentes entornos, ya sea en una máquina local o en la nube 3.15.

También se ha utilizado **Docker Hub**[22], una plataforma en la nube que actúa como un repositorio centralizado de imágenes de contenedores. Permite a los usuarios cargar y almacenar imágenes de contenedores, las cuales contienen aplicaciones o servicios completos junto con sus dependencias. Las imágenes pueden ser etiquetadas y organizadas para facilitar su búsqueda y referencia posterior. Además, Docker Hub ofrece funciones de control de versiones que permiten rastrear las actualizaciones y revisiones realizadas en las imágenes.

[15] https://cassandra.apache.org/_/index.html
[16] https://couchdb.apache.org
[17] https://redis.io
[18] https://www.postgresql.org
[19] https://www.mysql.com
[20] https://www.oracle.com/es/database
[21] https://www.docker.com
[22] https://hub.docker.com/search?q=ullsoftware&source=community

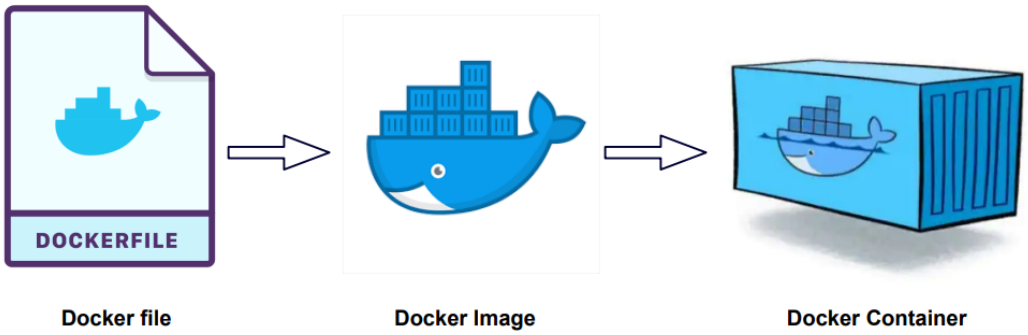


Figura 3.15: Estructura de Docker

Para definir y ejecutar los conjuntos de contenedores que conforman la aplicación, se utilizará **Docker Compose**[23]. Proporciona una forma sencilla de definir y orquestar la infraestructura necesaria para ejecutar una aplicación completa compuesta por varios servicios interconectados.

La principal ventaja de utilizar Docker Compose es que permite describir la configuración de los servicios de la aplicación en un archivo YAML. En este archivo se especifican los contenedores, las imágenes Docker, las variables de entorno, los volúmenes y las redes necesarias para el funcionamiento de la aplicación.

El papel del fichero **docker-compose.yml** es describir y configurar la infraestructura y los servicios necesarios para ejecutar varios contenedores Docker. En este caso, se definen los servicios **producer-containers**, **front-end**, **back-end-prod**, **back-end-dev**, **mongodb** y **watchtower**. A continuación se detallan las características de cada uno:

```
services:
  producer-containers:
    image: kaizten/producer-containers:latest
    container_name: producer-containers
    ports:
      - "8081:8081"
    environment:
      - PERIOD_MAXIMUMNUMBEROFCONTAINERSPERDAY=50
    profiles: ["database", "back-end", "full"]
```

- **producer-containers:** Nombre del servicio.
- **image:** Imagen de Docker que se utiliza para crear el contenedor.
- **container_name:** Define un nombre personalizado para el contenedor.
- **ports:** Define la configuración de los puertos del contenedor.
- **environment:** Permite configurar variables de entorno dentro del contenedor. Aquí se establece la variable de entorno `PERIOD_MAXIMUMNUMBEROFCONTAINERSPERDAY` con un valor de `50`.

[23] `https://docs.docker.com/compose/compose-file/03-compose-file`

- **profiles:** Es una opción específica de Spring Boot que permite activar perfiles específicos para el servicio.

```
services:
  # Front-end
  front-end:
    image: ullsoftware/yard-manager_front-end:latest
    container_name: front-end
    environment:
      - VITE_API_URL+8080
    ports:
      - "5173:80"
    profiles: ["full"]
```

- **front-end:** Nombre del servicio.
- **image**: Imagen de Docker que se utiliza para crear el contenedor.
- **container_name**: Otorga un nombre personalizado para el contenedor.
- **environment**: Permite configurar variables de entorno dentro del contenedor.
- **ports**: Define la configuración de los puertos del contenedor.
- **profiles**: Opción específica de Spring Boot que permite activar perfiles específicos para el servicio.

```
services:
  # Back-end
  back-end-prod:
    image: ullsoftware/yard-manager_back-end:latest
    container_name: back-end
    ports:
      - "8080:8080"
    environment:
      - SPRING_DATA_MONGODB_URI=mongodb+srv://alu0101129785:YJPaMIRWW6gYFu96@cluster0.
          jrkjmzk.mongodb.net/portplanning?retryWrites=true&w=majority
      - spring_profiles_active=production
    profiles: ["full"]

  # Back-end
  back-end-dev:
    image: ullsoftware/yard-manager_back-end:latest
    container_name: back-end-dev
    ports:
      - "8080:8080"
    environment:
      - SPRING_DATA_MONGODB_URI=mongodb://mongo:27017/portplanning
      - spring_profiles_active=development
    profiles: ["back-end"]
    depends_on:
      - mongodb
```

- **back-end-prod y back-end-dev:** Nombre del servicio.

- **image**: Imagen de Docker que se utiliza para crear el contenedor.
- **container_name**: Otorga un nombre personalizado para el contenedor.
- **ports**: Define la configuración de los puertos del contenedor.
- **environment**: Permite configurar variables de entorno dentro del contenedor.
- **profiles**: Opción específica de Spring Boot que permite activar perfiles específicos para el servicio.
- **depends_on**: Indica que el servicio back-end-dev depende del servicio mongodb.

```
services:
  mongodb:
    image: mongo:latest
    container_name: mongo
    command: mongod --port 27017
    ports:
      - "27017:27017"
    volumes:
      - "/data/db"
    profiles: ["database", "back-end", "full"]
```

- **mongodb:** Nombre del servicio.
- **image**: Imagen de Docker que se utiliza para crear el contenedor.
- **container_name**: Otorga un nombre personalizado para el contenedor.
- **command**: Define el comando que se ejecuta dentro del contenedor, por tanto, inicia el proceso mongod en el puerto 27017.
- **ports**: Define la configuración de los puertos del contenedor.
- **volumes**: Establece un volumen para persistir los datos de MongoDB.
- **profiles**: Opción específica de Spring Boot que permite activar perfiles específicos para el servicio.

3.7. GitHub

GitHub[24] es una plataforma de desarrollo de software basada en la nube que facilita la colaboración entre equipos de desarrollo. Permite a los desarrolladores alojar y revisar el código fuente de sus proyectos, realizar un seguimiento de los cambios realizados en el código, gestionar problemas y colaborar en la creación de nuevas características.

Una de las propiedades clave de GitHub es su sistema de control de versiones distribuido, que permite trabajar en paralelo en diferentes ramas del código y fusionar sus cambios de manera eficiente.

[24] `https://github.com`

Además del control de versiones, GitHub también ofrece otras funcionalidades importantes y que se ha usado durante el desarrollo de todo el proyecto. **GitHub Actions** ha sido utilizado para la definición y automatización de las tareas del proyecto, así como para controlar su desarrollo mediante commits referenciados a cada tarea, ofrece ventajas como la monitorización del estado de las mismas y la notificación sobre los avances realizados. Esta herramienta proporcionó un entorno de ejecución y flujo de trabajo que facilitó la comunicación y colaboración entre el tutor y el alumno.

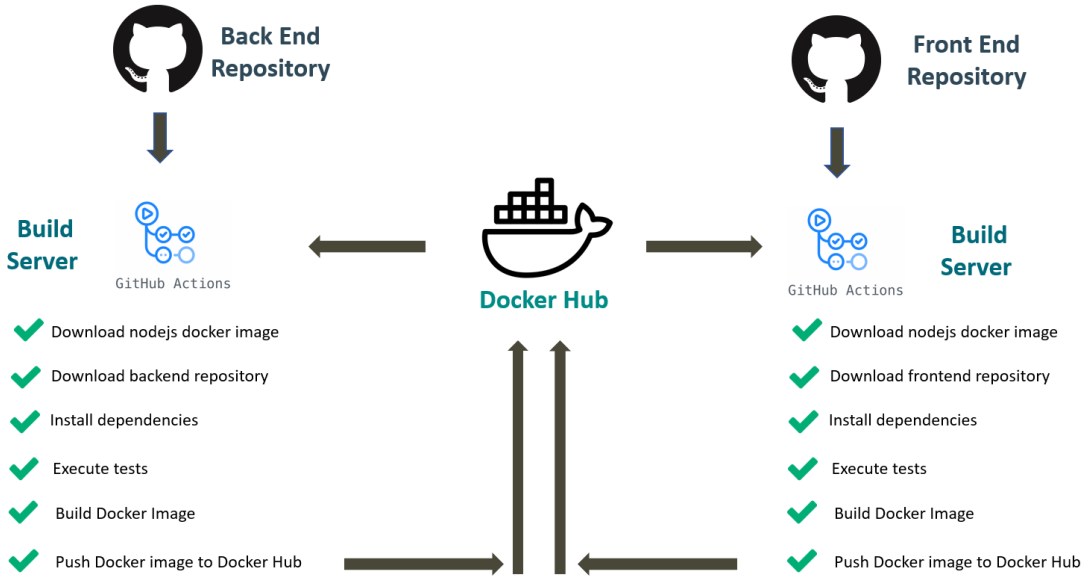


Figura 3.16: Integración GitHub Actions con Docker CI

GitHub Actions y Docker CI son herramientas que pueden trabajar en conjunto para facilitar el desarrollo de software. Una vez que la acción ha completado con éxito, se inicia el proceso de construcción de la imagen de Docker. Esto implica la compilación y empaquetado de la aplicación junto con sus dependencias en una imagen Docker lista para su implementación.

La imagen de Docker generada puede ser etiquetada con una versión específica, como un número de versión o un hash git, para facilitar la referencia y el control de versiones de la misma.

3.8. Pruebas

En el ámbito del desarrollo de software, es de vital importancia asegurar la calidad y el correcto funcionamiento de la aplicación resultante. Una de las prácticas fundamentales en este proceso es la realización de pruebas exhaustivas en los diferentes componentes a desarrollar del proyecto.

Los **tests unitarios** y **de integración** desempeñan un papel fundamental en el desarrollo de cualquier proyecto, ya que permiten comprobar y garantizar el correcto funcionamiento de la aplicación desarrollada. A continuación, se describen algunas de las razones por las cuales estos tests son importantes:

- **Detectar errores tempranamente.** Ayudan a identificar posibles errores o fallos en el código antes de que se propaguen y se conviertan en problemas mayores.

Al ejecutar pruebas exhaustivas, se pueden encontrar y solucionar rápidamente fallas, evitando que afecten otras partes del sistema.

- **Mejorar la calidad del software.** Verifican que cada componente de la aplicación cumpla con los requisitos y funcionalidades esperadas. Al asegurarse de que todos los aspectos del software funcionen correctamente, se garantiza una mayor calidad del producto final.
- **Facilitar la refactorización.** Actúan como una red de seguridad al realizar cambios en el código. Al tener una serie de pruebas sólida, se puede refactorizar el código sabiendo que si se introducen errores, los tests lo detectarán.
- **Agilizar el proceso de desarrollo.** Al automatizar los tests, se reduce la necesidad de realizar pruebas manuales repetitivas. Esto permite ahorrar tiempo y recursos, al tiempo que proporciona una mayor fiabilidad en los resultados.

Capítulo 4

Despliegue del proyecto

La aplicación está compuesta por varios servicios que se ejecutan en diferentes contenedores de Docker. Cada servicio tiene una funcionalidad específica y se arrancan en un puerto determinado.

- El servicio **producer-containers** es responsable de generar contenedores y está configurado para ejecutarse en el puerto 8081.
- El servicio **front-end** contiene la interfaz de usuario y se ejecuta en el puerto 5173. Utiliza la imagen ullsoftware/yard-manager_front-end:latest y tiene el perfil “full“.
- Los servicios de **back-end** se ejecutan con la imagen ullsoftware/yard-manager_back-end:latest y se arrancan en el puerto 8080. El servicio de back-end-dev utiliza una base de datos MongoDB local en el puerto 27017.

4.1. Despliegue local

El despliegue en local ha sido muy utilizado durante todo el desarrollo puesto que permite comprobar los cambios de forma automática. Para ello se aplica el comando 4.1. Este comando se ejecuta en el directorio /back-end y permite iniciar el servicio de la API-REST.

```
/port-storage-planning/back-end
    > mvn spring-boot:run
```

Listing 4.1: Ejecución del comando para desplegar el backend de la API-REST

Posteriormente se inicia el servicio producer-containers a través de la imagen kaizten/producer-containers:latest disponible en el puerto 8081 como se aprecia en el siguiente comando:

```
/port-storage-planning/
    > docker run -p 8081:8081 kaizten/producer-containers:latest
```

Listing 4.2: Ejecución del comando para desplegar el generador de contenedores

Finalmente, se pone en marcha la interfaz de usuario ejecutando el comando 4.3. Este comando se ejecuta en el directorio /front-end y permite iniciar la interfaz de usuario.

```
/port-storage-planning/front-end
    > npm run dev
```

Listing 4.3: Ejecución del comando para desplegar el frontend de la API-REST

Una vez que todos los servicios están desplegados, se puede acceder a la dirección http://localhost:5173 en un navegador web y visualizar el contenido correctamente.

4.2. Despliegue con Docker

Este método despliega la aplicación haciendo uso de los contenedores de Docker, utilizando para ejecutar los servicios definidos el archivo docker-compose.yml. Con el siguiente comando 4.4 se inicia la aplicación en un entorno más controlado con una experiencia más cercana al nivel de un usuario normal.

```
/port-storage-planning
    > docker-compose --profile full up
```

Listing 4.4: Ejecución con comando Docker

Siguiendo estos pasos, y habiendo arrancado Docker, se debe ver un resultado similar al de la imagen 4.1.

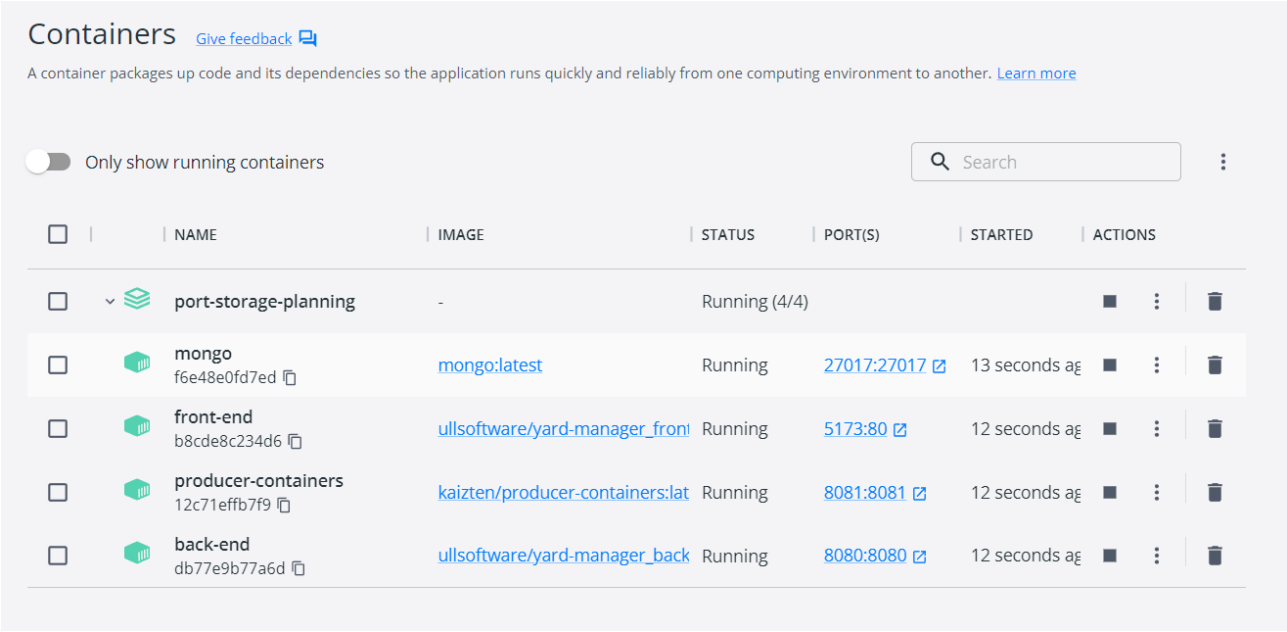


Figura 4.1: Despliegue con Docker

Capítulo 5

Resultados

En este capítulo se representa un caso real en el que se simula la planificación del almacenamiento de mercancías en un puerto. El objetivo es utilizar el software desarrollado para gestionar eficientemente las operaciones de almacenamiento en el contexto portuario.

La correcta disposición del almacenamiento de las mercancías es fundamental para optimizar las operaciones logísticas en el puerto. Es necesario asegurarse de que cada contenedor se ubique en la posición adecuada. Para este caso concreto el puerto va a tener las siguientes características:

- **Bloques**. El puerto estará conformado por 6 bloques.
- **Pilas**. Cada bloque constará de 200 pilas de contenedores distribuidas en 10 filas y 20 columnas.
- **Altura**. Cada fila constará con una capacidad máxima para albergar 4 contenedores.

Por tanto, la capacidad máxima de almacenamiento en este puerto se regirá por la siguiente operación:

```
Total de contenedores = Bloques × Pilas por bloque × Altura de cada pila
```

```
Total de contenedores = 6 × 200 × 4 = 4800
```

Al acceder a la página de inicio de sesión, los usuarios son recibidos con una tarjeta central donde se encuentran dos campos de texto que se deben cumplimentar para realizar el inicio de sesión.

Justo debajo de estos campos, se encuentra un botón que permite acceder al home de la aplicación, realizando de esta forma el inicio de sesión 5.1.

Una vez que los usuarios acceden con éxito, son dirigidos automáticamente a la página de inicio5.2. En la parte superior de la página, se puede encontrar una barra de navegación que permite acceder a otras secciones de la aplicación como el desglose por bloques, o los listados de contenedores, personal y grúas que conforman el puerto.

Justo debajo de la barra de navegación, se aprecia la información disponible del puerto. Se muestra el nombre del mismo en un título destacado e indicadores circulares que proporcionan información visual sobre los diferentes aspectos del puerto, como las estadísticas de contenedores, bloques y bahías.

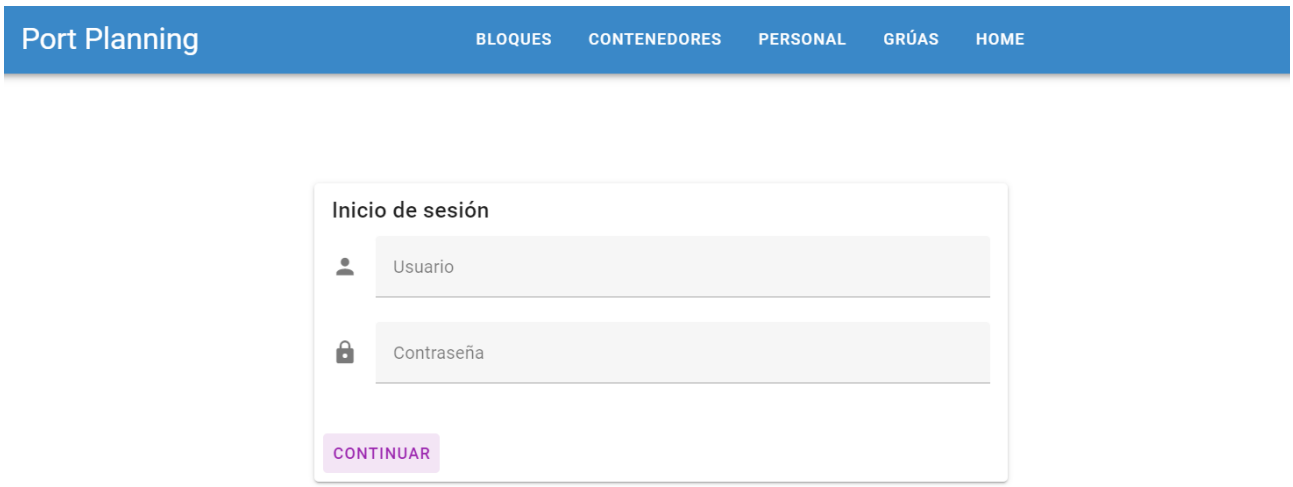


Figura 5.1: Pantalla de Login

Figura 5.2: Menú principal

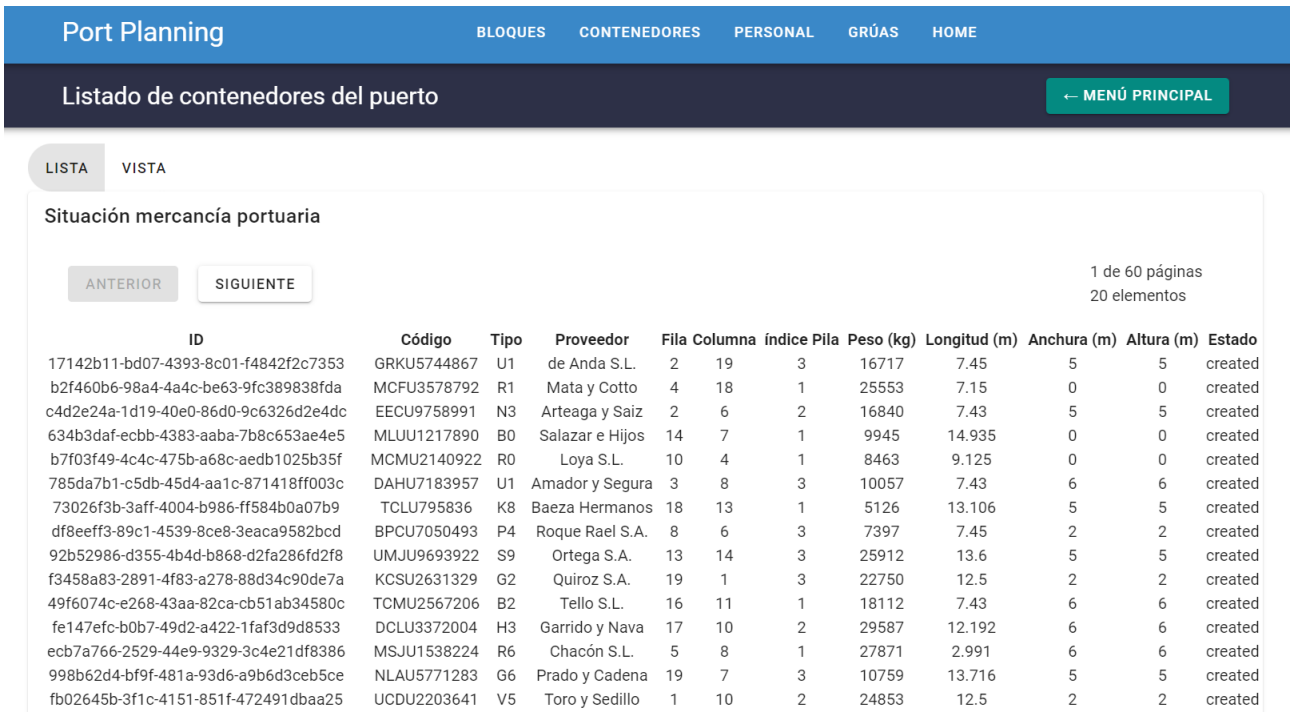
Al presionar sobre la sección de **Contenedores** aparece una lista en formato de tabla con datos detallados sobre la mercancía en el puerto. En la parte superior volvemos a encontrar el header que permite un rápido desplazamiento a otra sección. Debajo se encuentra un componente de tipo `UserInteraction` que proporciona el título de la sección así como un botón para retroceder al menú principal 5.3.

El contenido más destacado de esta vista es la tabla que muestra información sobre los contenedores del puerto.‘El usuario puede navegar por las diferentes páginas de la tabla utilizando los botones “Anterior“ y “Siguiente“. También aporta datos sobre la página actual, el número total de páginas y el número de elementos por página.

Cada fila de la tabla representa un contenedor y muestra varias características del mismo, como el identificador, código, tipo, proveedor, su fila y columna, el índice de pila, el peso, la longitud, la anchura, la altura y el estado. Los valores **tipo**, **anchura** y **altura** siguen el estándar **ISO 6346** [1] que permite una rápida identificación de las características de un contenedor en particular.

En la sección de **Personal** se accede a la información de los trabajadores del puerto. La vista de listado de personal muestra una tabla con información detallada compuesta

[1] `https://www.bic-code.org/size-type-code`

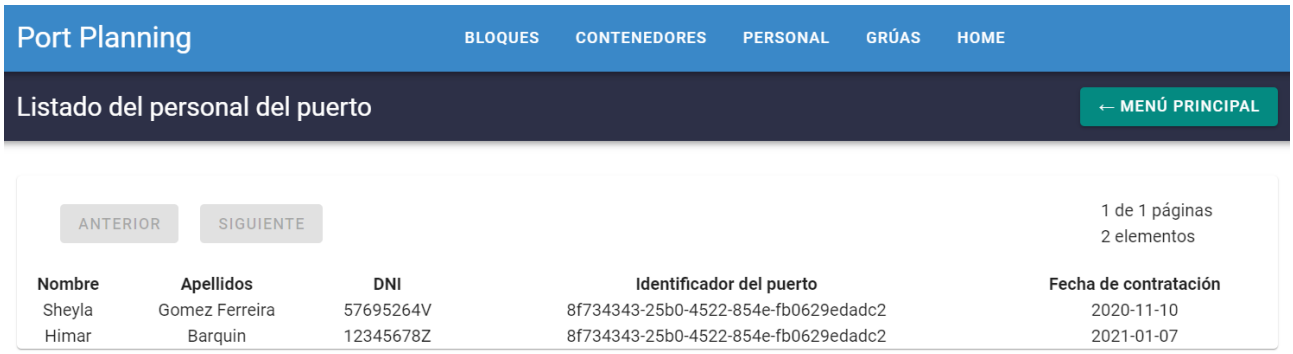


ID	Código	Tipo	Proveedor	Fila	Columna	índice Pila	Peso (kg)	Longitud (m)	Anchura (m)	Altura (m)	Estado
17142b11-bd07-4393-8c01-f4842f2c7353	GRKU5744867	U1	de Anda S.L.	2	19	3	16717	7.45	5	5	created
b2f460b6-98a4-4a4c-be63-9fc389838fda	MCFU3578792	R1	Mata y Cotto	4	18	1	25553	7.15	0	0	created
c4d2e24a-1d19-40e0-86d0-9c6326d2e4dc	EECU9758991	N3	Arteaga y Saiz	2	6	2	16840	7.43	5	5	created
634b3daf-ecbb-4383-aaba-7b8c653ae4e5	MLUU1217890	B0	Salazar e Hijos	14	7	1	9945	14.935	0	0	created
b7f03f49-4c4c-475b-a68c-aedb1025b35f	MCMU2140922	R0	Loya S.L.	10	4	1	8463	9.125	0	0	created
785da7b1-c5db-45d4-aa1c-871418ff003c	DAHU7183957	U1	Amador y Segura	3	8	3	10057	7.43	6	6	created
73026f3b-3aff-4004-b986-ff584b0a07b9	TCLU795836	K8	Baeza Hermanos	18	13	1	5126	13.106	5	5	created
df8eeff3-89c1-4539-8ce8-3eaca9582bcd	BPCU7050493	P4	Roque Rael S.A.	8	6	3	7397	7.45	2	2	created
92b52986-d355-4b4d-b868-d2fa286fd2f8	UMJU9693922	S9	Ortega S.A.	13	14	3	25912	13.6	5	5	created
f3458a83-2891-4f83-a278-88d34c90de7a	KCSU2631329	G2	Quiroz S.A.	19	1	3	22750	12.5	2	2	created
49f6074c-e268-43aa-82ca-cb51ab34580c	TCMU2567206	B2	Tello S.L.	16	11	1	18112	7.43	6	6	created
fe147efc-b0b7-49d2-a422-1faf3d9d8533	DCLU3372004	H3	Garrido y Nava	17	10	2	29587	12.192	6	6	created
ecb7a766-2529-44e9-9329-3c4e21df8386	MSJU1538224	R6	Chacón S.L.	5	8	1	27871	2.991	6	6	created
998b62d4-bf9f-481a-93d6-a9b6d3ceb5ce	NLAU5771283	G6	Prado y Cadena	19	7	3	10759	13.716	5	5	created
fb02645b-3f1c-4151-851f-472491dbaa25	UCDU2203641	V5	Toro y Sedillo	1	10	2	24853	12.5	2	2	created

Figura 5.3: Listado de contenedores

por dos componentes principales: UserInteractionStaff y StaffListTable.

El componente UserInteractionStaff se encuentra en la parte superior donde se visualiza un botón que facilita al usuario su vuelta al menú principal 5.4.



Nombre	Apellidos	DNI	Identificador del puerto	Fecha de contratación
Sheyla	Gomez Ferreira	57695264V	8f734343-25b0-4522-854e-fb0629edadc2	2020-11-10
Himar	Barquin	12345678Z	8f734343-25b0-4522-854e-fb0629edadc2	2021-01-07

Figura 5.4: Listado del personal

El componente StaffListTable muestra una tabla con la lista de personal del puerto. En la tabla, se presentan campos como el nombre, apellidos, DNI, identificador del puerto, grúas designadas y fecha de contratación. Además, se incluye una paginación que permite navegar entre las diferentes páginas de resultados a través de los botones “Anterior“ y “Siguiente“.

Por su parte, la vista de la sección **Grúas** es similar a las anteriores, muestra una lista de las grúas pertenecientes al puerto y conformada por dos componentes principales: `UserInteractionCrane` y `CraneListTable`.

El componente `UserInteractionCrane` ubicado bajo la barra de navegación en la parte superior de la página brinda información como el título de la sección e interacción mediante un botón de vuelta al menú principal 5.5.

El componente `CraneListTable` presenta un listado con las grúas disponibles. Cada fila de la tabla representa una grúa y muestra información relevante, como el puerto

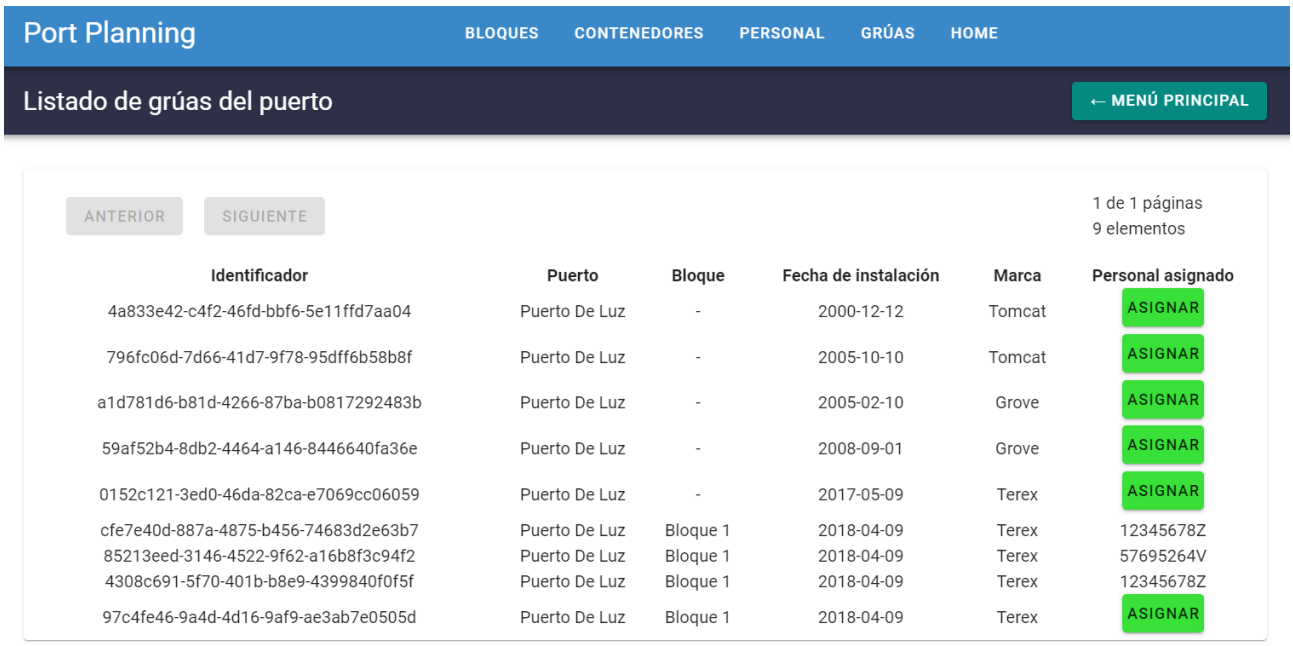


Figura 5.5: Listado de las grúas

al que pertenece, el bloque al que está asignada, su identificador único, la fecha de instalación, la marca y un botón para asignar un trabajador a esa grúa. Además, la información se presenta paginada como en las secciones anteriores.

Finalmente, en el sector que corresponde al desglose por **Bloques** 5.6 se proporciona una vista de los bloques del puerto. En la parte superior de la vista se encuentra el componente `UserInteractionPort` que permite la interacción del usuario con el puerto con un título que indica el nombre de la sección y un botón que permite al usuario regresar a la página principal.

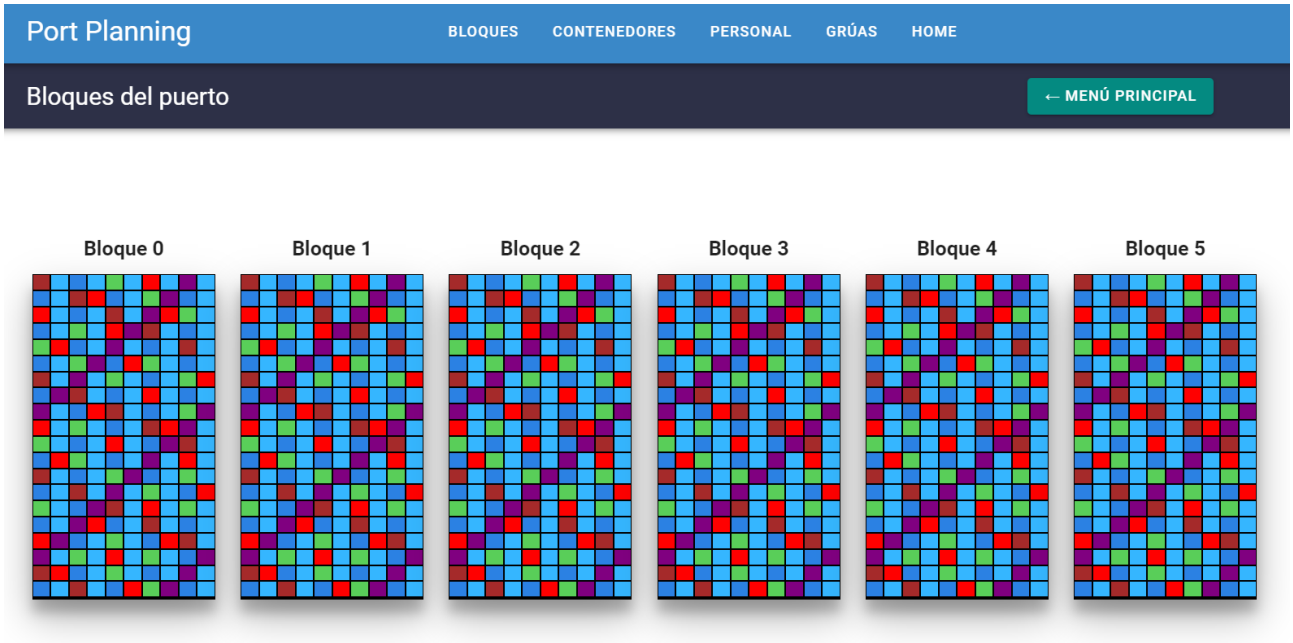


Figura 5.6: Desglose por bloques

En el cuerpo se muestra una serie de bloques representados por el componente

`BlockComponent`. Cada bloque se renderiza como un elemento en forma de rectángulo dentro de la cuadrícula. Los bloques se obtienen de forma dinámica mediante una llamada a la API utilizando el ID del puerto.

Este componente representa visualmente los bloques del puerto mediante un rectángulo compuesto por cuadrados que simbolizan a cada una de las pilas del mismo. Al seleccionar un bloque, el usuario accede a la vista de las bahías de este. Características de `BlockComponent`:

- **Representación de pilas.** Cada bloque se compone de 200 cuadrados dentro del rectángulo, donde cada cuadrado representa una pila en el bloque.
- **Selección de bloque.** Al hacer clic en un bloque específico, se activa el método `submitContainerBayView`, que redirige al usuario a la vista de las bahías correspondientes a ese bloque en particular.

La siguiente sección corresponde a las **Bahías del bloque** 5.7. `BlockBayView` conforma la vista principal que muestra la distribución de las bahías en un bloque específico del puerto. Proporciona una representación visual de las pilas de la bahía en forma de cubos, donde cada uno de ellos simboliza una pila y su color varía en función del número de contenedores asociados a la misma.

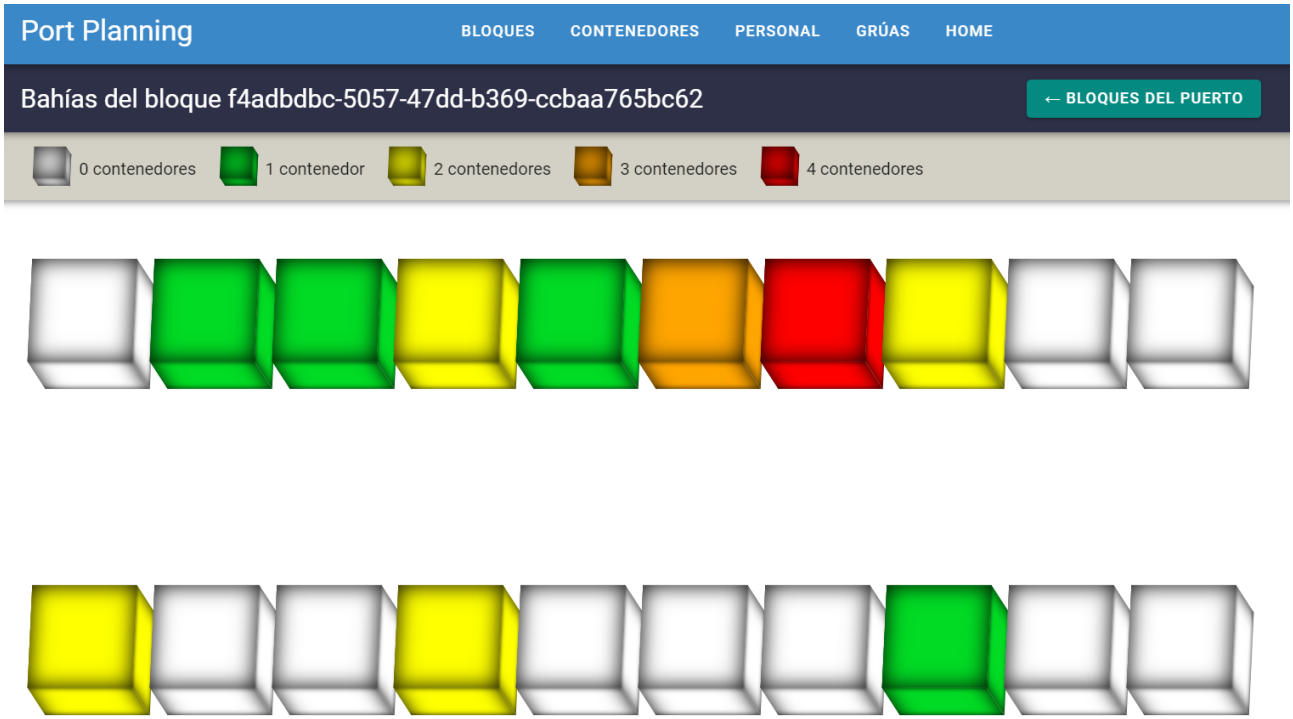


Figura 5.7: Bahías del bloque

En la parte superior de la vista se encuentra el componente `UserInteractionBlock` que permite la interacción del usuario con el bloque. Incluye el título que indica el número de bloque y un botón que permite al usuario regresar a la lista de bloques del puerto.

Inmediatamente después, está el componente `LegendBay` el cual muestra una leyenda que proporciona una referencia visual para comprender los colores utilizados en la representación de las bahías.

- Blanco: No hay contenedores.
- Verde: 1 contenedor.
- Amarillo: 2 contenedores.
- Naranja: 3 contenedores.
- Rojo: 4 contenedores (llena).

En el cuerpo de la vista se muestra una cuadrícula de bahías representadas por el componente `CubeBayComponent`. El número de bahías se determina dinámicamente a partir de la capacidad del bloque obtenida mediante una llamada a la API utilizando el identificador del bloque. En esta vista se muestra una representación visual de las pilas del puerto, su diseño está pensado para mostrarlas de forma organizada en una cuadrícula de 10x20.

Al hacer clic en una de las pilas, se activa el evento `@click` redirigiendo al usuario a una vista frontal que muestra todas las pilas de esa bahía en particular. Esta vista permite al usuario obtener una visión más detallada de las pilas de una bahía específica y los contenedores situados en ellas.

Esta vista cuenta con una barra circular de progreso que se muestra mientras se carga la información del bloque y las bahías.

La siguiente sección lleva a la vista frontal de una **Bahía** 5.8. Como en los casos anteriores, en la parte superior de la vista se encuentra un componente de tipo `UserInteraction`, este permite al usuario regresar a la vista anterior donde se alojan las bahías del puerto 5.7 y también al menú principal 5.2. En el título se indica el número de la bahía seleccionada.

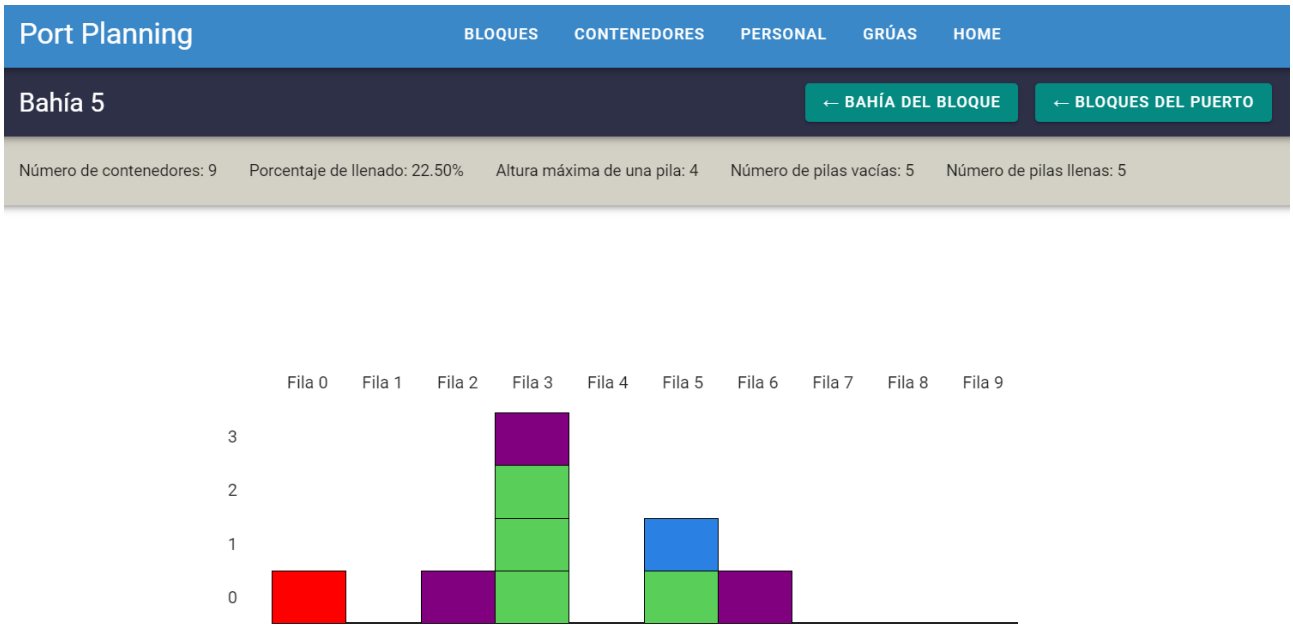


Figura 5.8: Vista frontal de la bahía

Inmediatamente después está un componente llamado **BayStatisticsComponent** que aporta información referida al estado actual de la bahía que se está visualizando:

- **Número de contenedores.** Indica la cantidad de contenedores que hay alojados en esa bahía actualmente.
- **Porcentaje de llenado.** Muestra una estadística sobre el porcentaje de ocupación de esa bahía respecto a la cantidad de contenedores que puede albergar. Se considera que el 100 % del espacio equivale a 40 contenedores por bahía.
- **Altura máxima de una pila.** Informa de que la altura máxima de las pilas de este puerto es de 4.
- **Número de pilas vacías.** Cantidad de pilas en las que no se ha ubicado ningún contenedor.
- **Número de pilas ocupadas.** Refleja el número de pilas donde al menos hay un contenedor depositado.

En el cuerpo de la vista se muestran los contenedores apilados en cada pila. Después de ser asignados a un bloque de forma aleatoria entre los disponibles, se le asigna a cada contenedor de forma aleatoria una fila y una columna cuya pila aún no esté completa, mostrándose como resultado una imagen como la figura 5.8. Se podrán seguir asignando contenedores a esta bahía mientras haya alguna pila cuya capacidad no haya sido alcanzada.

Para facilitar la visualización al usuario se ofrece una enumeración tanto de las filas como de las columnas que conforman cada bahía.

Finalmente, al pinchar sobre uno de los contenedores situados en la bahía, se proporcionará otra vista con la **información específica de ese contenedor** en concreto 5.9.

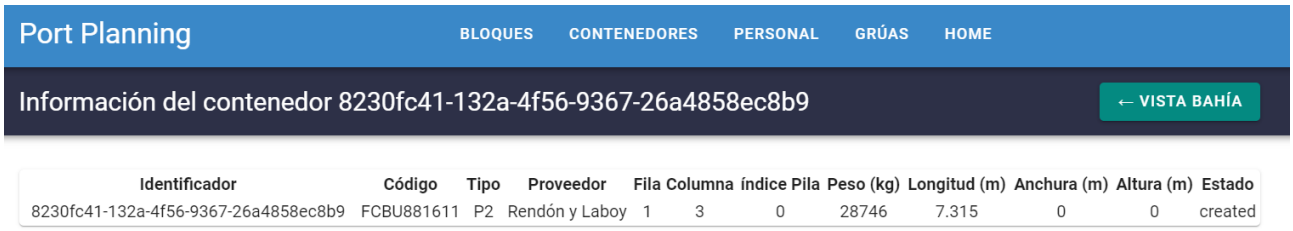


Figura 5.9: Información de un contenedor de la bahía

En un formato de tabla, se muestra la información específica sobre el contenedor seleccionado. Esta vista consta de dos componentes: `UserInteractionContainerDetail` y `ContainerDetailTable`.

El componente UserInteractionContainerDetail indica el identificador del contenedor. También cuenta con un botón que permite al usuario regresar a la vista de la bahía.

El componente ContainerDetailTable se encarga de visualizar los detalles del contenedor en un formato de tabla. Estos incluyen información como el identificador del contenedor, el código, el tipo, el proveedor, la fila, la columna, el índice de la pila, el peso, la longitud, la anchura, la altura y el estado del contenedor. Estos detalles se obtienen mediante una llamada a la API utilizando el identificador del contenedor.

Capítulo 6

Presupuesto

Este capítulo ofrece una visión detallada de los honorarios asociados al desarrollo del proyecto de este Trabajo de Fin de Grado, dividiéndolos en conceptos como costos de infraestructuras tecnológicas de desarrollo, especificando las horas empleadas así como el precio por hora. Esta estimación tiene como referencia el sueldo aproximado de un ingeniero Fullstack[1] en España. La tabla 6.1 está desglosada en hitos, horas empleadas para la resolución de cada uno de ellos y el coste asociado a cada tarea.

Presupuesto			
Hito	Tiempo empleado	Precio por hora	Precio total
Diseño de la aplicación	30 horas	17€	510€
Desarrollo Backend	170 horas	17€	2890€
Desarrollo Frontend	130 horas	17€	2210€
Generador contenedores	20 horas	17€	340€
Dashboards e indicadores	10 horas	17€	170€
			6120€

Tabla 6.1: Presupuesto

[1] `https://es.talent.com/salary?job=full+stack`

Capítulo 7

Conclusiones

En la actualidad, el transporte marítimo ha experimentado un impresionante auge y se ha convertido en la columna vertebral del comercio internacional. Los barcos cargueros atraviesan océanos y mares, transportando ingentes cantidades de mercancías de un continente a otro. Este crecimiento exponencial se ha visto impulsado por diversos factores. En primer lugar, la globalización del comercio ha generado una mayor demanda de transporte de mercancías a larga distancia. Además, la capacidad y escalabilidad del transporte marítimo han desempeñado un papel fundamental, ya que los buques portacontenedores modernos pueden transportar grandes volúmenes de carga.

Asimismo, el desarrollo de infraestructura portuaria ha mejorado la capacidad y conectividad de los puertos, facilitando un flujo eficiente de mercancías. La implementación de estándares internacionales, como el ISO 6346[1][2] aplicado en este Trabajo de Fin de Grado, ha desempeñado un papel crucial en la identificación estandarizada de los contenedores, solucionando casuísticas relacionadas con confusiones, retrasos y dificultades en la gestión logística. Por último, los avances tecnológicos en la industria, como la gestión de contenedores y la digitalización de los procesos logísticos, han optimizado las operaciones y mejorado la eficiencia en el transporte marítimo.

Otro elemento clave en el éxito y la eficiencia del transporte marítimo es la implementación de un software especializado para la gestión de mercancías capaz de manejar grandes volúmenes de carga y realizar complejas operaciones logísticas, siendo este el objetivo final que se ha perseguido durante la realización del proyecto.

Como resultado, la aplicación web resultante tras la finalización del proyecto es capaz de gestionar la información a través de una API REST desde el backend y cuenta con una interfaz de usuario fácil de manejar. Posee un simulador de flujo de mercancías, capacidad para hacer un seguimiento en tiempo real del estado de las pilas y los contenedores que se acumulan en ellas y su representación gráfica. También posibilita la obtención de la información sobre los diferentes actores del puerto tales como los contenedores, la maquinaria disponible y el personal en formato de listados paginados.

7.1. Líneas futuras

A pesar de que el proyecto actual ha logrado desarrollar una aplicación web capaz de gestionar la información de manera eficiente y proporcionar una interfaz de usuario intuitiva, existen oportunidades para continuar mejorando y ampliando su funcionalidad en el futuro. A continuación, se presentan algunas líneas futuras que podrían considerarse:

- **Capacidad para la gestión de varios puertos.** Actualmente, la aplicación se centra en la gestión de un único puerto. Sin embargo, sería beneficioso expandir su capacidad para gestionar múltiples puertos. Esto permitiría a los usuarios acceder y administrar la información de diferentes puertos en una única plataforma, lo que facilitaría la coordinación y la toma de decisiones a nivel global.
- **Seguimiento del estado de los contenedores a lo largo del tiempo.** Sería interesante incorporar una funcionalidad que permita realizar un seguimiento del estado de los contenedores a lo largo del tiempo. Esto implicaría registrar y mostrar el historial de movimientos y eventos relacionados con cada contenedor, lo que proporcionaría una mayor visibilidad y trazabilidad de las mercancías a lo largo de su vida por el puerto.
- **Visualizaciones personalizadas por roles de usuario.** Para mejorar la experiencia y adaptar la información a las necesidades específicas de cada usuario, sería útil desarrollar visualizaciones personalizadas por roles. Esto permitiría que diferentes usuarios, como operadores portuarios, agentes de carga o autoridades aduaneras, accedan a vistas específicas y obtengan información relevante para sus responsabilidades y tareas específicas.

Capítulo 8

Conclusions

Currently, maritime transportation has experienced an impressive growth and has become the backbone of international trade. Cargo vessels cross oceans and seas, transporting vast amounts of goods from one continent to another. This exponential boom has been driven by various factors. Firstly, the globalization of trade has generated a higher demand for long-distance transportation of goods. Additionally, the capacity and scalability of maritime transportation have played a crucial role, as modern container ships can carry large volumes of cargo.

Furthermore, the development of port infrastructure has improved the capacity and connectivity of ports, facilitating an efficient flow of goods. The implementation of international standards, such as ISO 6346[1][2] applied in this Bachelor's Thesis. It has played a crucial role in the standardized identification of containers, solving issues related to confusion, delays, and logistical difficulties. Lastly, technological advancements in the industry, such as container management and the digitalization of logistical processes, have optimized operations and improved efficiency in maritime transportation.

Another key element in the success and efficiency of maritime transportation is the implementation of specialized cargo management software capable of handling large volumes of cargo and performing complex logistical operations. This has been the ultimate goal pursued throughout the completion of this project.

As a result, the web application developed as part of this project is capable of managing information through a REST API from the backend and features a user-friendly interface. It includes a cargo flow simulator, real-time tracking of the status of stacks and the containers accumulating within them, and graphical representations. It also provides information about various port actors, such as containers, available machinery, and personnel, in paginated list formats.

8.1. Future Directions

Although the current project has successfully developed a web application capable of efficiently managing information and providing an intuitive user interface, there are opportunities to further improve and expand its functionality in the future. The following are some potential future directions that could be considered:

- **Capability to manage multiple ports:** Currently, the application focuses on managing a single port. However, expanding its capability to handle multiple ports would be beneficial. This would allow users to access and manage information

from different ports on a single platform, facilitating coordination and decision-making at a global level.

- **Tracking container status over time:** It would be interesting to incorporate functionality that enables tracking the status of containers over time. This would involve recording and displaying the history of movements and events related to each container, providing greater visibility and traceability of goods throughout their journey within the port.
- **Customized visualizations based on user roles:** To enhance the user experience and tailor information to specific user needs, developing customized visualizations based on user roles would be beneficial. This would allow different users, such as port operators, cargo agents, or customs authorities, to access specific views and obtain relevant information for their respective responsibilities and tasks.

Bibliografía

[1] AESTIR. CBP Automated Export System Trade Interface Requirements. https://www.cbp.gov/sites/default/files/assets/documents/2020-Feb/ACE%20Appendix%20P%20-%20Container-Equipment%20Type%20Codes_0.pdf, 2013.

[2] German Insurance Association. Container Handbook. https://www.containerhandbuch.de/chb_e/stra/index.html?/chb_e/stra/stra_03_04_00.html, 2000.

[3] World Bank. The container port performance index 2022: A comparable assessment of performance based on vessel time in port. Technical report, May 2023. Citation: " World Bank . 2023 . The Container Port Performance Index 2022: A Comparable Assessment of Performance Based on Vessel Time in Port . CPPI . © World Bank . http://hdl.handle.net/10986/39824 License: CC BY 3.0 IGO . ".

[4] Carlos Blé Jurado. *Código sostenible : cómo programar código fácil de mantener*. Savvily, 2022.

[5] Eric Evans. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.

[6] Felipe Gutierrez and Felipe Gutierrez. Spring boot, simplifying everything. *Introducing Spring Framework: A Primer*, pages 263–276, 2014.

[7] Rodrigo Gómez-Montoya, Abdul Mazo, Nancy Ceballos-Atehourtu, and David Palacio Jiménez. Gestión de la cadena de suministros y productividad en la literatura científica. *I+D Revista de Investigaciones*, 14:34–45, 06 2019.

[8] Jörg Krause. Developing web components with typescript.

[9] Xinyu Luan. Implementation and analysis of softwaredevelopment in spring boot. *Doctoral dissertation*, 2021.

[10] Alexandar Pantaleev and Atanas Rountev. *Identifying Data Transfer Objects in EJB Applications*. 2007.

[11] Mithun Satheesh, Bruno Joseph D'mello, and Jason Krol. *Web development with MongoDB and NodeJs*. Packt Publishing Ltd, 2015.

[12] Victoria Eugenia Pino Terán and Hugo Hernan Castro Gonzalez. Desarrollo de nuevas tecnologías en gestión portuaria internacional.

[13] Lourdes Trujillo Castellano and Andrea Rodríguez Ramos. Identificación de los cuellos de botella que se generan en los puertos en lo referido a la mercancía que entra y sale de canarias. *Cátedra Marítimo-Portuaria*, 2022.

[14] Balaji Varanasi. *Introducing Maven: A Build Tool for Today's Java Developers*. Apress, 2019.