UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Desarrollo Back-End para un sistema de gestión logística de contenedores en almacenaje extraportuario Aexsa

Matías Ojeda Saavedra

matias.ojedas@usm.cl

Oscar Carrasco Vera
Profesor Guía

Resumen: Aexsa, empresa dedicada al almacenaje extraportuario, enfrenta dificultades para mantener un control eficiente de la ubicación en tiempo real de sus contenedores en el patio de estos mismos, afectando la toma de decisiones operativas y la trazabilidad de los movimientos que realizan las grúas. Ante este contexto, se desarrolló una solución basada en un sistema BackEnd capaz de gestionar la ubicación de los contenedores y sincronizarla en tiempo real con el FrontEnd. Siendo el objetivo principal diseñar una arquitectura de software eficiente que permita actualizar y consultar la ubicación de los contenedores, además de prepararse para futuras integraciones con tecnologías IOT. La solución se implementó utilizando el conjunto de tecnologías MERN, con una API RESTful para las operaciones CRUD y WebSockets para la sincronización instantánea de datos, empleando una arquitectura monolítica modular. La validación de la solución se realizó mediante pruebas de integración de los mismos componentes del BackEnd con el FrontEnd y la simulación de eventos de entrada y salida de contenedores. Como resultado se logró un sistema funcional capaz de mantener actualizadas las visualizaciones interactivas correspondientes al patio de contenedores en tiempo real, mejorando la trazabilidad de la ubicación de los contenedores y con ello la eficiencia operativa.

Palabras Clave: Backend; Logística; Extraportuario; WebSockets

1. INTRODUCCIÓN

La empresa Aexsa, dedicada al almacenaje extraportuario, enfrenta un reto común en el sector: La gestión eficiente de los movimientos de contenedores que realizan las grúas, impactando directamente en la eficiencia operativa. Un problema crítico es no disponer de la ubicación exacta de cada contenedor en el patio de almacenaje, provocando retrasos en la búsqueda de estos. Actualmente, Aexsa aborda esta problemática asignando manualmente la ubicación de los contenedores, registrando estos datos en hojas de cálculo, sin aplicar medio alguno de verificación sobre la veracidad y exactitud de estos.

La falta de visibilidad de datos en tiempo real dificulta la toma de decisiones informadas, comprometiendo la eficacia y la capacidad de la empresa para optimizar el uso del espacio que disponen en el patio de contenedores.

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Desarrollo Back-End para un sistema de gestión logística de contenedores en almacenaje extraportuario Aexsa

Nombre del candidato(a): Matías Ignacio Ojeda Saavedra

Carrera / Grado: Ingeniería en Informática

Campus: Viña del Mar **;** **Departamento**: Departamento de Electrotecnia e Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Óscar Carrasco Vera, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

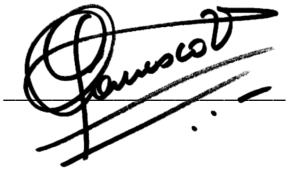
☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

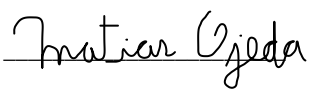
Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 11/8/2025 ; **Firma:** 

Estudiante o Candidato(a):

Fecha: 11/08/2025 ; **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

1.1. Importancia del desarrollo de componentes específicos en el proyecto

El desarrollo de un sistema integrado de gestión logística que permita la visualización en tiempo real de los datos resulta crucial si queremos una solución robusta que facilite la interacción entre operador y sistema. La solución propuesta busca abordar este desafío a través de una aplicación web con visualizaciones interactivas, como lo es un mapa 3D del patio de contenedores de Aexsa, que mostrarán en tiempo real la ubicación exacta de cada contenedor, proveyendo así una mejor representación de datos para una eficiente toma de decisiones y una mejora en la eficiencia operativa al saber dónde están los contenedores que se necesiten mover.

La implementación de una API REST para recursos necesarios que se deban de mostrar en la aplicación web y el uso de WebSockets para una comunicación continua entre servidor y cliente, se vuelven indispensables para garantizar el funcionamiento de la solución propuesta. Este desarrollo no solo mejorará el manejo de los contenedores, si no que ofrecerá una plataforma escalable que pueda adaptarse a futuras mejoras, como la implementación de tecnologías IOT para la automatización de procesos.

1.2. Objetivo general

Desarrollar una arquitectura Back-End escalable para una aplicación web de gestión logística que permita gestionar la información del patio de contenedores de Aexsa.

1.3. Objetivos Específicos

- Desarrollar una API RESTful con Node.js y Express.js que facilite la comunicación entre el Front-End y el Back-End.
- Implementar mecanismos de sincronización automática mediante WebSockets.
- Garantizar la escalabilidad del Back-End, asegurando que pueda soportar la integración de futuras tecnologías IOT

1.4. Justificación

Este proyecto resulta relevante desde el punto de vista técnico porque aborda la necesidad crítica de optimizar los procesos logísticos en el patio de contenedores. El beneficio de contar con un sistema que visualice la ubicación de los contenedores en tiempo real permite tomar decisiones más informadas y reduce el tiempo de búsqueda. El área de desarrollo Back-End para este proyecto aborda varios aspectos tales como la implementación de una arquitectura modular, la sincronización en tiempo real mediante WebSockets, el traslado de la lógica de negocio al Back-End y la creación de APIs REST. La implementación de estas tecnologías permite abordar de manera integral los desafíos logísticos de Aexsa, simplificando el intercambio de datos y construyendo una base sólida para el futuro crecimiento del sistema.

El desarrollo del Back-End proporcionará múltiples beneficios, como mejorar la eficiencia de las operaciones al tener acceso a datos en tiempo real, una arquitectura diseñada para facilitar la escalabilidad del sistema ante un eventual aumento en la cantidad de contenedores. y un mayor nivel de seguridad en la información almacenada y transmitida.

1.5. Metodología

Para la construcción del Back-End se sigue la metodología de desarrollo ágil Scrum, que divide el equipo en roles y el desarrollo en incrementos (Sprint). Esta metodología permitirá al equipo responder de manera efectiva a los cambios, adaptándose a las necesidades del proyecto. La experiencia previa de esta metodología nos proporciona una base sobre la cual desarrollar el proyecto.

A nivel grupal, nuestro equipo trabajará de manera colaborativa, integrando diferentes habilidades en el desarrollo de la solución. La propuesta se basa en implementar un sistema que permita el seguimiento y control eficiente de los contenedores, mejorando la logística y la toma de decisiones.

Mi papel especifico será desarrollar el Back-End utilizando Node.js y Express.js. Debo de implementar una API REST que facilite la comunicación entre Front-End y Back-End e incorporar WebSockets para asegurar que las actualizaciones de estado y ubicación de contenedores se transmitan de inmediato, además de las funcionalidades relacionadas con la lógica de negocio de la empresa.

2. MARCO TEÓRICO

2.1. Fundamentos del desarrollo Back-End

El desarrollo Back-End implica el desarrollo de la lógica e integración interna, tal como un servidor o una base de datos, de un sitio web o aplicación. Trata del conjunto de acciones que pasan en un sistema informático que no son visibles para el usuario.

Algunas funciones que realiza el desarrollador Back-End son:

- Conexión con base de datos.
- Creación de API.
- Creación de scripts (lógica) para el sitio web.
- Mantener la consistencia y seguridad de los datos.

El concepto de Back-End comienza a tomar forma con la introducción del modelo cliente-servidor, el cliente es quien solicita servicios y el servidor es quien provee esos servicios. Pero no fue hasta la década de 1990 con el nacimiento de la web y los navegadores que la distinción entre Front-End y Back-End se volvió más evidente al permitir que los servidores ejecutaran scripts y devolvieran respuestas al navegador.

El desarrollo web moderno nace de la mano con tecnologías como HTML5, debido a que la arquitectura web crece enormemente gracias a las APIs, quienes hacen posible que cualquier Front-End se pueda comunicar con cualquier Back-End creando una frontera más clara entre ambos desarrollos.

2.2. Arquitectura de software.

La arquitectura del Back-End se diseñará con un enfoque monolítico y modular, el cual divide las responsabilidades para facilitar el desarrollo, mantenimiento e integración de las funcionalidades del sistema. Tanto las relacionadas con la lógica del negocio, el acceso a datos y las interfaces de comunicación estarán contenidas dentro de una misma aplicación. Enfoque el cual resulta adecuado para el alcance del proyecto.

El diseño propuesto asegura que cada componente tenga una responsabilidad clara y definida, y que además posibilita escalar el sistema en el futuro.

El cliente se conecta a través del módulo de interfaces de comunicación, para que ya dentro de la arquitectura misma los módulos se comuniquen entre sí, sin mezclar sus responsabilidades y limitándose solo a sus funciones. El módulo de acceso a datos es el responsable de comunicarse con la base datos.

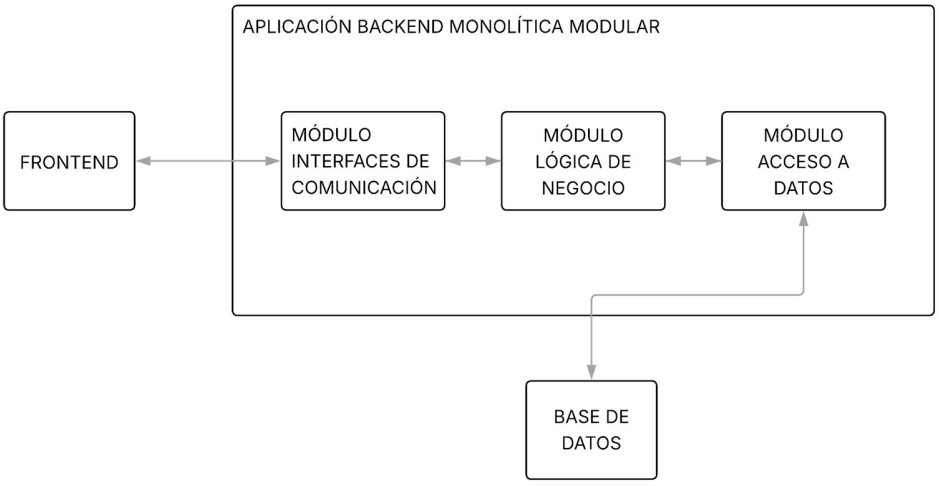


Figura 1. Arquitectura monolítica modular

Fuente: Elaboración propia

2.3. Tecnologías y frameworks relevantes

Tecnologías principales:

- Django: Framework de desarrollo web de alto nivel escrito en Python que sigue el patrón de diseño model-view-template (MVT).

- Laravel: Framework de desarrollo web escrito en PHP conocido por su simplicidad.
- Ruby on Rails: Framework de desarrollo web escrito en Ruby que sigue el patrón de diseño model-view-controller (MVC)
- Node.js: Plataforma de ejecución para JavaScript basado en el motor V8 de Google Chrome.
- Express.js: Uno de los Framework Back-End más populares para Node.js
- APIs RESTful: Arquitectura para APIs que permite la comunicación entre sistemas mediante el uso de recursos identificados por URLs y operaciones realizadas por métodos HTTP.

¿Por qué utilizar Node.js junto a Express.js, y no alguno de los otros frameworks?

- Node.js posee un rico ecosistema en librerías y paquetes, destacando npm, siendo uno de los gestores de paquetes más grandes registrados. Npm provee varias de las librerías utilizadas para este proyecto, un ejemplo es Socket.io, librería basada en WebSockets para el manejo de eventos en aplicaciones web en tiempo real. Por otra parte, si bien Django, Laravel y Ruby on Rails tienen mecanismos de implementación de WebSockets, ninguno de estos frameworks mencionados está focalizado en soluciones en tiempo real.
- Al utilizar Node.js y Express.js estamos siendo coherentes con el Stack tecnológico del proyecto, puesto que para el desarrollo del Front-End se utiliza el framework React, también escrito en JavaScript. Por otra parte, Django está escrito en Python, Laravel está escrito en PHP y Ruby on Rails está escrito en Ruby, lo que complicaría más la integración entre las interfaces.
- El gestor de paquetes npm de Node.js también provee Mongoose, biblioteca de JavaScript que crea una conexión entre MongoDB (el sistema de base de datos utilizado para el proyecto) y Node.js, logrando que de forma nativa ambos componentes puedan comunicarse sin mayor esfuerzo. Por otra parte, si bien existen métodos para conectar Django, Laravel y Ruby on Rails con MongoDB, el rendimiento del sistema con el uso estos frameworks es menor en comparación al uso de Node.js. Uno de los aspectos claves del sistema es que se detecte rápidamente los cambios en la base de datos, por lo que la comunicación e integración entre el Back-End y la base de datos debe de ser eficiente.
- La naturaleza minimalista de Express.js permite manejar componentes del Back-End, como las API y los middlewares, de forma sencilla y concisa debido a su rápido código de implementación. Por otra parte, los otros frameworks requieren más pasos para construir las API.

Tendencias actuales:

- Infraestructura Serverless: Tecnología que permite a los desarrolladores centrarse en escribir código sin la necesidad de manejar además la infraestructura ya que esta es administrada por terceros.
- Docker y Kubernetes: Herramientas que permiten gestionar arquitecturas complejas de forma más sencilla, garantizando escalabilidad y portabilidad.

- Arquitectura basada en eventos (Event-Driven): Paradigma en el que el sistema reaccione a eventos en tiempo real es el diseño central para el desarrollo de sistemas.

Si bien ninguna de estas tendencias actuales está integrada en el proyecto, la arquitectura basada en eventos resulta interesante por su valor de reaccionar a eventos en tiempo real. Lo que permitiría integrar IOT (internet de las cosas), red de dispositivos inteligentes que permiten recopilar datos y enviarlos a través de internet, y la posibilidad de automatizar completamente la lógica de negocio del sistema.

3. METODOLOGÍA

3.1. Organización y planificación del desarrollo

El rol que ocupo en la metodología Scrum es el de Scrum Master, quien se encarga y vela que el equipo cumpla y comprenda el desarrollo según la metodología misma, en donde el progreso de cada integrante es registrado, evaluado y retroalimentado constantemente durante el desarrollo de cada incremento.

La asignación de tareas y la creación del Sprint Backlog antes de cada Sprint resulta esencial para llegar a completar las funcionalidades esperadas en el plazo acordado. Las tareas son creadas a partir de historias de usuario y el Sprint Backlog son las funcionalidades acordadas que necesitamos desarrollar en cada incremento.

El desarrollo de Back-End constará de las siguientes fases:

- Investigación de herramientas a utilizar (1 semana): Se analizarán y seleccionarán las herramientas más adecuadas para el desarrollo del proyecto, buscando coherencia en un mismo stack tecnológico.

- Creación del Sprint Backlog: Funcionalidades que debemos de desarrollar en el primer incremento. Estas son seleccionadas a partir de la creación de historias de usuarios.

- Desarrollo del Incremento (4 semanas): Tiempo en el cual el equipo desarrollador se encargará de crear el código necesario para las funcionalidades seleccionadas en el Sprint Backlog.

- Entrega del Primer Incremento (11/09); Primera entrega y presentación de las funcionalidades desarrolladas frente al Cliente.

- Creación del segundo Sprint Backlog: Funcionalidades que debemos de desarrollar en el segundo incremento y funcionalidades que quedaron pendientes del incremento anterior, si aplica.

- Desarrollo del Incremento (4 semanas)

- Entrega del Segundo Incremento (18/10): Segunda entrega y presentación de las funcionalidades desarrolladas frente al Cliente.

- Creación del tercer y último Sprint Backlog: Funcionalidades restantes para desarrollar el Producto Mínimo Viable.
- Desarrollo del Incremento (4 semanas)
- Entrega del Producto Final (15/11): Entrega y presentación del Producto mínimo viable en Feria de Software USM.

3.2. Enfoque para el desarrollo de componentes

El desarrollo de componentes del sistema se basa en una arquitectura modular, el cual divide la aplicación en partes independientes donde cada componente tiene una responsabilidad específica. Facilitando el mantenimiento del código y permitiendo trabajo colaborativo en diferentes partes del proyecto sin conflictos.

La comunicación entre Front-End y Back-End se realiza a través de una API REST. API que realiza operaciones CRUD a través de los métodos HTTP estándar para manejar las operaciones básicas de la aplicación.

Para el manejo de actualizaciones en tiempo real se utilizarán WebSockets. Protocolo de comunicación bidireccional basado en eventos que permite que el servidor envíe datos al cliente sin que este tenga que solicitarlos constantemente.

3.3. Herramientas y entornos de desarrollo utilizados

Para el desarrollo del Back-End y sus componentes se utilizarán las siguientes herramientas:

- JavaScript: Lenguaje principal para el desarrollo del Front-End y Back-End, lo que provee mayor coherencia con el Stack tecnológico a trabajar.
- Node.js: Entorno de ejecución de JavaScript que permite utilizar este lenguaje en el servidor.
- Express.js: Framework Back-End para Node.js con gran flexibilidad y un ecosistema rico.
- WebSockets: Protocolo para la comunicación en tiempo real basado en eventos.
- Visual Studio Code: Programa de editor de código utilizado para escribir, organizar y depurar el código fuente.
- Postman: Programa utilizado para probar, documentar y validar las APIs REST desarrolladas.
- Git: Sistema de control de versiones que permite la colaboración entre los desarrolladores del equipo.
- GitHub Project: Plataforma de gestión de proyectos utilizada para organizar las tareas y hacer seguimiento del progreso.

4. DISEÑO DE COMPONENTES

4.1. Arquitectura y estructura de los componentes

La arquitectura del Back-End se organiza en módulos funcionales, diseñados para cumplir una tarea en específica, estos módulos son:

- API RESTful: La API RESTful es el módulo responsable de manejar las solicitudes HTTP, mensajes que solicitan información al servidor provenientes del Front-End. La API RESTful se encarga de gestionar las operaciones CRUD (crear, leer, actualizar y eliminar) relacionadas con los contenedores y sus ubicaciones. Las solicitudes se procesan mediante rutas específicas que delegan la lógica de negocio a los controladores.

  La API RESTful requiere que las solicitudes HTTP tengan la siguiente estructura:

  - URL: Ruta que especifica al servidor que recurso requiere el cliente.
  - Método: Los métodos de HTTP siguen la misma lógica de las operaciones CRUD, siendo GET lectura, POST crear, PUT actualizar y DELETE eliminar.
  - Datos: Datos para que los métodos funciones correctamente, como lo puede ser la ubicación de un contenedor en un método POST.
  - Parámetros: Detalles que brindan información detallada al servidor de lo que debe de hacer.

- WebSockets: Protocolo de comunicación que permite la sincronización en tiempo real entre cliente (Front-End) y servidor (Back-End), esto al establecer una conexión bidireccional entre ambos. Cuando se detecta un cambio en la base de datos, como la creación o eliminación de un contenedor, el servidor envía un evento al cliente través de los WebSockets para actualizar las vistas en el Front-End automática e instantáneamente una vez registrado el cambio.

- Controladores: Los controladores son responsables de procesar las solicitudes entrantes del cliente, transformarlas según la lógica del negocio y retornar las respuestas adecuadas. Cada controlador está asociado a un conjunto de rutas específicas de la API. Por ejemplo, el controlador “dataContenedor” contiene métodos para manejar las operaciones sobre los contenedores, como “addContenedor” agregar contenedor, “getContenedor” obtener los contenedores, “updateContenedor” actualizar los atributos del contenedor y “removeContenedor” eliminar un contenedor. Tanto el controlador de “addContenedor” como “removeContenedor” son responsables de disparar el evento que posteriormente los WebSockets envían al cliente para actualizar sus vistas.

- Middleware: Los middlewares son funciones que se encargan de validar datos, gestionar errores y asegurar que las solicitudes cumplan con los requisitos solicitados. Preprocesan la información requerida por los controladores para que estos puedan utilizarla, como lo puede ser validar que el formato JSON de una solicitud HTTP esté estructurada correctamente.

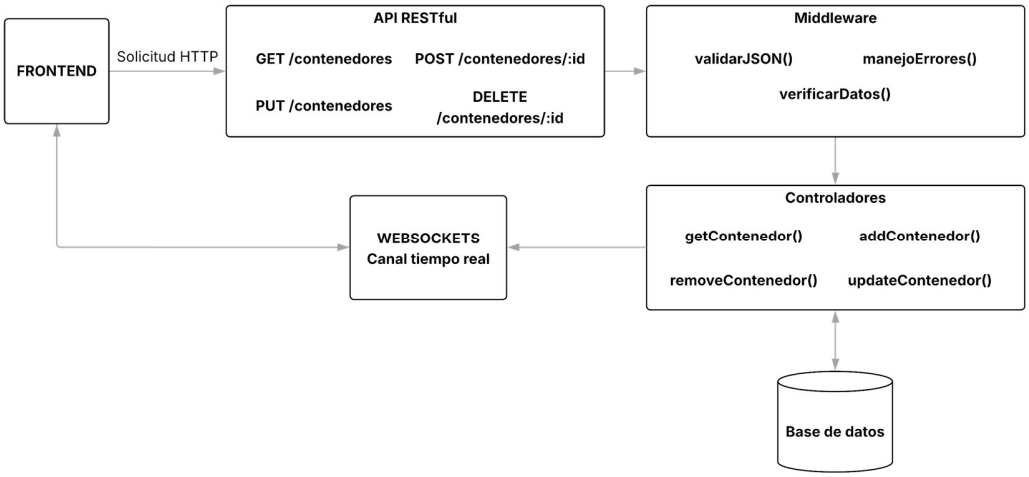


Figura 2. Esquema de componentes

Fuente: Elaboración propia

4.1.1. Flujo de datos

El flujo de datos entre los componentes sigue el siguiente diseño:

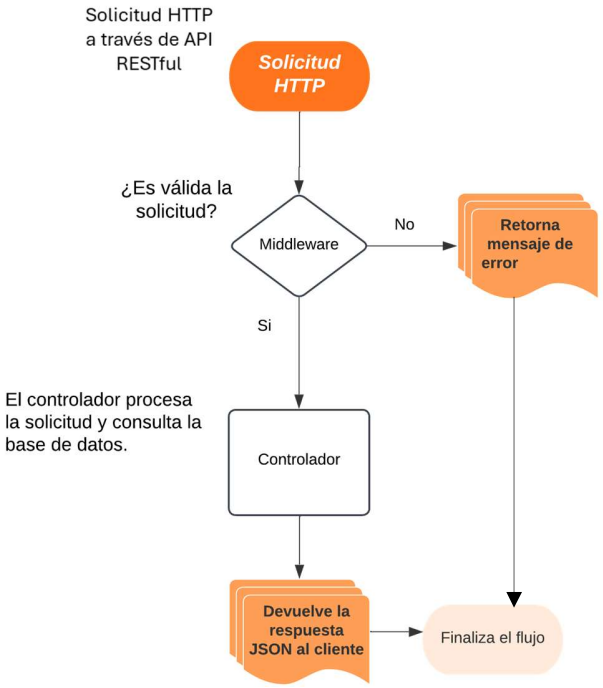


Figura 3. Flujo de datos de una solicitud HTTP

Fuente: Elaboración propia

- El cliente (Front-End) realiza una solicitud HTTP a través del módulo API RESTful
- El middleware valida la solicitud. Si no es válida, retorna un mensaje de error
- Si es válida, el controlador en el servidor (Back-End) procesa la solicitud y realiza las operaciones correspondientes.
- Las operaciones interactúan con la base de datos y devuelve los datos procesados.
- La respuesta se envía al cliente en formato JSON.

El flujo de datos para las actualizaciones en tiempo real es el siguiente:

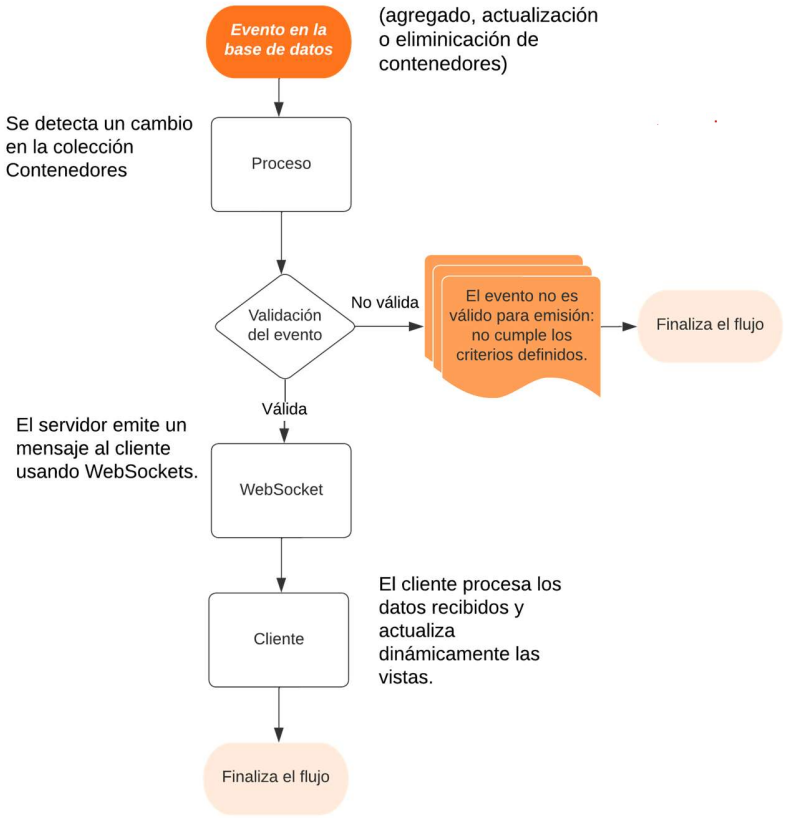


Figura 4. Flujo de datos para evento en tiempo real

Fuente: Elaboración propia

- Se detecta un cambio en la base de datos, como la adición, modificación o eliminación de un contenedor en la colección de contenedores.
- Se valida si es que ese evento debe ser emitido por el WebSocket; para ello se consideran criterios como que el contenedor exista, que el cambio afecte la ubicación o estado del contenedor, y que el nuevo valor difiera del anterior. Si no se cumple, el flujo finaliza.

- Si es válido, el servidor emite un mensaje a través del módulo WebSockets al cliente.
- El cliente actualiza dinámicamente las vistas basándose en los datos recibidos.

4.2. Integración con el sistema general

La arquitectura del Back-End está diseñada para integrarse con el resto de los componentes de la aplicación, utilizando protocolos y formatos estándar para garantizar una comunicación eficiente. En el contexto de este proyecto los componentes del sistema general se ejecutan localmente en cada entorno de desarrollo.

- El Front-End se conecta con el Back-End a través de la API RESTful, esta permite el envío de solicitudes HTTP.

- El Back-End conectado a la base de datos, interactúa con esta a través de operaciones CRUD anteriormente entregadas en la solicitud HTTP del Front-End.

- Front-End y Back-End además están conectados a través de WebSockets, esta comunicación es bidireccional y full-dúplex, lo que significa que el emisor y el receptor pueden enviar y recibir mensajes al mismo tiempo.

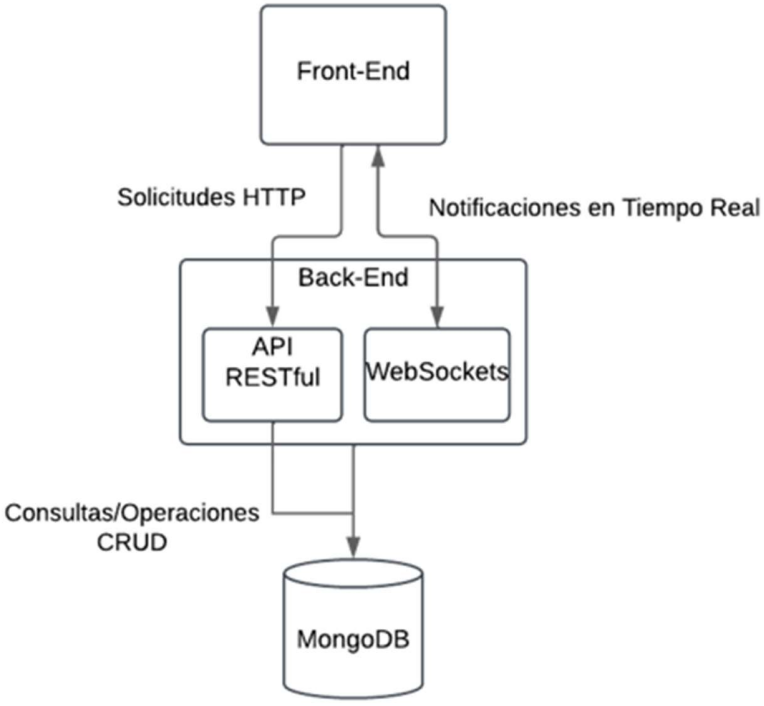


Figura 5. Esquema del sistema general

Fuente: Elaboración propia

4.2.1. Integración con el Front-End

La integración el Front-End se realiza mediante API RESTful y WebSockets.

- API RESTful: La API tiene endpoints, ubicación digital (URL) en donde se reciben las solicitudes HTTP, para operaciones CRUD (Crear, Leer, Actualizar, Eliminar) relacionadas con los contenedores. Todas las solicitudes HTTP siguen el formato JSON, formato de texto JavaScript pensando para el intercambio de datos.

  - Estructura de la API:

Tabla 1. Estructura de la API
Fuente: Elaboración Propia

Operación	Método HTTP	Endpoint	Parámetros	Descripción
Obtener Contenedores	GET	/api/contenedores	Ninguno	Retorna todos los contenedores almacenados en la base de datos.
Agregar Contenedor	POST	/api/contenedores	{id, ubicación, zona, visado}	Agrega un contenedor al sistema.
Actualizar Contenedor	PUT	/api/contenedores/:id	{id del contenedor a actualizar}	Actualizar atributos del contenedor especificado según su identificador
Eliminar Contenedor	DELETE	/api/contenedores/:id	id del contenedor a eliminar	Elimina un contenedor en específico según su identificador.

  - Solicitudes HTTP en formato JSON:

```
//Solicitud de tipo POST esperada
{
  "contenedor": "GDFH-345623-9",
  "ubicacion": "A1-01-01-01",
  "zona": "Comercial",
  "visado": false
}
//Respuesta esperada
{
  "message": "Contenedor agregado exitosamente",
  "contenedor": {
    "_id": "64f1a9...",
    "contenedor": "GDFH-345623-9",
    "ubicacion": "A1-01-01-01",
    "zona": "Comercial",
    "visado": false
  }
}
```

- WebSockets: Las actualizaciones en tiempo real se gestionan mediante WebSockets. Se utilizan para emitir eventos al cliente, como cuando: se agrega un nuevo contenedor, se elimina un contenedor existente o se actualiza la ubicación de un contenedor. El evento emitido por el WebSocket está compuesto por el tipo de acción a realizar y los datos necesarios para que el cliente los utilice.

  - Llamada esperada evento WebSockets

```
{
  "contenedorActualizado": {
    "tipo": "agregar",
    "contenedor": {
      "contenedor": "GDFH-345623-9",
      "ubicacion": "A1-01-01-01",
      "zona": "Comercial",
      "visado": false
    }
  }
}
```

4.2.2. Integración con la base de datos

La integración con la base de datos está diseñada para manejar las operaciones necesarias del sistema mediante la utilización de MongoDB, una base de datos NoSQL orientada a documentos. A diferencia de los sistemas relacionales, MongoDB no utiliza tablas ni esquemas estrictos, sino que almacena los datos en documentos JSON, permitiendo mayor flexibilidad en la estructura de datos.

- Estructura de los esquemas: Se utiliza Mongoose, biblioteca de JavaScript que crea una conexión entre MongoDB y Node.js, para definir esquemas y modelos que estructuran los datos en la base de datos, como la colección de Contenedores, previamente modelada por el desarrollador de base de datos.
  - Esquema de la colección Contenedores:

```
{
  "contenedor": { "type": "String", "required": true },
  "ubicacion": { "type": "String", "required": true },
  "zona": { "type": "String", "required": true },
  "visado": { "type": "Boolean", "default": false }
}
```

- Diccionario de datos:

Tabla 2. Diccionario de datos de la colección Contenedores

Fuente: Elaboración propia

Campo	Tipo	Requerido	Formato	Descripción	Ejemplos
_id	ObjectId	Sí	Generado automáticamente por MongoDB. No se modifica ni se ingresa manualmente.	Identificador único del documento. Se utiliza para referencia interna en la base de datos.	ObjectId("...")
contenedor	String	Sí	Cadena alfanumérica de 11 caracteres (4 letras + 7 números)	Código único que identifica al contenedor estandarizado para contenedores marítimos.	"MSCU1234567"
ubicacion	String	Sí	Coordenada en formato BAROTI "Fila-Columna-Nivel"	Indica la posición exacta del contenedor en el patio, para control y localización visual en el sistema.	"B2-4-1"
zona	String	Sí	Valor restringido a un conjunto predefinido de zonas.	Zona o sector asignado al agrupación o clasificación logística. Los valores posibles son: "Comercial", "Aduana", "Mantención", "Inspección".	"Comercial"
visado	Boolean	No	Valor true o false. Se asume false por defecto	Estado que indica si el contenedor ha sido aprobado para salir del patio (Proceso de despacho).	false

- Modelo de base de datos:

Además de la colección Contenedores, se había previsto la implementación de una colección Movimientos para registrar los cambios de estado y ubicación de cada contenedor. Esta colección cumple una función clave en la trazabilidad de las operaciones logísticas, permitiendo mantener un

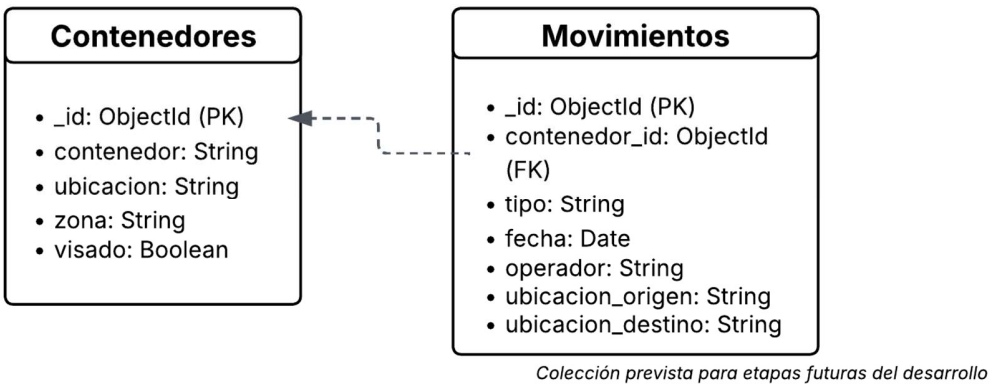


Figura 6. Diagrama del modelo de base de datos MongoDB

Fuente: Elaboración propia

historial estructurado de ingresos, despachos y reubicaciones. Cada documento en Movimientos contiene una referencia al contenedor afectado mediante su identificador, y registra información adicional como el tipo de movimiento, la fecha del evento y las ubicaciones de origen y destino, lo que permite mantener una trazabilidad óptima del historial de cada contenedor.
Las coordenadas de ubicación de un contenedor se especifican mediante un formato denominado BAROTI, el cual representa su posición física en el patio logístico. Este formato combina tres componentes separados por guiones: la fila, la columna y el nivel de apilamiento vertical(altura).
Sin embargo, esta funcionalidad no fue desarrollada dentro del alcance del presente trabajo, aunque se incluye en el diagrama para mostrar la proyección estructural del sistema.

5. IMPLEMENTACIÓN

5.1. Detalles de la codificación y desarrollo

El desarrollo del Back-End se llevó a cabo siguiendo el marco de trabajo Scrum, un enfoque iterativo e incremental, permitiendo implementar funcionalidades de manera progresiva. Por cada Sprint o Incremento se desarrollaban nuevas funcionalidades.

5.1.1. Organización en tareas y backlog

El desarrollo se dividió en 3 Sprint, el cual cada uno de ellos tenían varias tareas específicas que componían el backlog del proyecto.

- Sprint 1:
    - Configuración del Entorno de Desarrollo (A cargo de: Back-End)
    - Creación de la Guía de Estilo (A cargo de: UI-UX)
    - Inicializar el Repositorio GIT (A cargo de: Control de Versiones)
    - Implementar la página de inicio (A cargo de: Front-End, UI-UX)
    - Creación de la primera vista del Mapa (A cargo de: Front-End, UI-UX)
    - Implementación de Mapa 3D (A cargo de: Front-End, Back-End)
    - Avance de la creación del Mapa (A cargo de: Front-End, UI-UX)
- Sprint 2:
    - Agregar vistas restantes (A cargo de: Front-End, UI-UX)
    - Construcción de las subvistas (A cargo de: Front-End, UI-UX)
    - Implementar API (A cargo de: Back-End)
    - Crear estructura de la base de datos (A cargo de: Desarrollador de Base de Datos)
    - Creación de Vista de Login (A cargo de: Front-End, UI-UX)
    - Integración de distintos componentes (A cargo de: Back-End, Front-End)
    - Implementar una base de datos (Desarrollador de Base de Datos)
    - Mejorar del mapa (A cargo de: Front-End)
- Sprint 3:

- Implementar obtener contenedores (A cargo de: Back-End)
- Implementar agregar contenedores (A cargo de: Back-End)
- Implementar eliminar contenedores (A cargo de: Back-End)
- Implementar WebSockets (A cargo de: Back-End)
- Optimizar página (A cargo de: Back-End, Front-End)
- Hospedar la aplicación en un servidor (Desarrollador de Base de Datos, Back-End)

5.1.2. Control de versiones

Se utilizó Git para gestionar el control de versiones del código y GitHub como repositorio del código en cuestión. Cada funcionalidad se desarrolló en ramas separadas según el cargo del desarrollador.

- Rama master: Almacena versiones estables del proyecto.
- Rama frontend: Utilizado para desarrollar funcionalidades correspondientes al rol.
- Rama backend: Utilizado para desarrollar funcionalidades correspondientes al rol.
- Rama database: Utilizado para desarrollar funcionalidades correspondientes al rol.
- Rama optimizacion: Utilizado para refactorizar código.

5.2. Uso de buenas prácticas y patrones de diseño.

En el desarrollo del Back-End se ha utilizado una aproximación del patrón de diseño Modelo-Vista-Controlador (MVC) para organizar el código de forma modular y clara, lo que asegura la separación de responsabilidades:

- Modelo: Los esquemas definidos con Mongo se representan el rol de los modelos al estructurar los datos almacenados en la base den datos MongoDB.

- Vista: Desde el punto de vista del desarrollo Back-End, la vista corresponde a las respuestas JSON generadas por la API RESTful que son enviadas al cliente.

- Controlador: Los controladores actúan como el corazón de la lógica del negocio, procesan las solicitudes HTTP recibidas, validan los datos necesarios y delegan las operaciones a realizar a los demás módulos.

Si bien el uso de patrones de diseños estructura el código para que sea más comprensible, esto no sería posible si es que no aplicamos buenas prácticas, tales como:

- Estructura modular del proyecto: Cada componente se organizó en carpetas separadas para mantener una arquitectura clara.

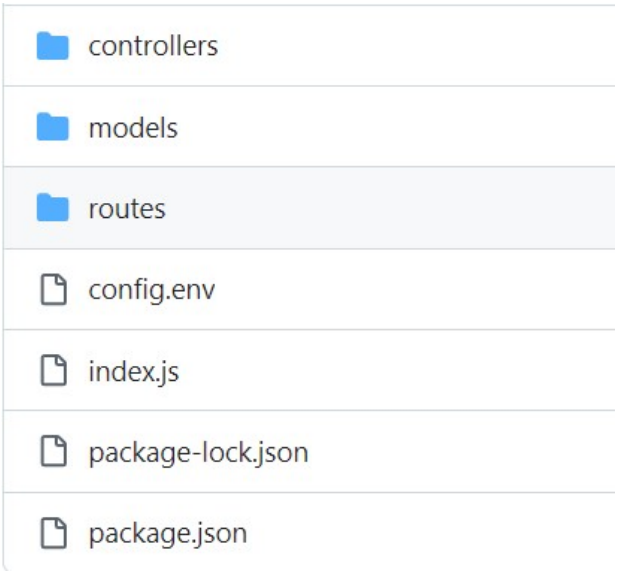


Figura 7. Estructura de directorios

- Gestión de errores: Se implementaron middlewares para manejar errores y retornar mensajes claros al cliente.
- Manejo de configuraciones sensibles: Se utilizo el paquete dotenv, herramienta que permite cargar variables de entorno, para almacenar variables sensibles como las credenciales de la base de datos.
- Uso de principios de desarrollo limpio (Clean Code):
    - Nombres descriptivos para funciones, variables y archivos
    - Funciones pequeñas y enfocadas en realizar una única tarea

6. PRUEBAS Y VALIDACIÓN

6.1. Estrategias de testing aplicadas a los componentes

Para el desarrollo del Back-End se aplicaron distintas estrategias de testing a los componentes de API RESTful, WebSockets y Controladores. Estas pruebas se realizaron manualmente como parte del proceso de desarrollo, asegurando que se cumplieran los requerimientos planteados. Las estrategias aplicadas incluyeron:

- Pruebas Unitarias: El desarrollador verificó directamente la funcionalidad de las operaciones críticas, tales como las operaciones CRUD para la gestión de contenedores, evaluando los resultados en entornos controlados. Estas pruebas permitieron validar funciones aisladas, como la función de parseo de ubicaciones o la validación de datos de entrada.

```
// Controlador para agregar un contenedor
const addContenedor = async (req, res, io) => {
  const { contenedor, ubicacion, zona = '', visado = false } = req.body; // Extrae los datos del body, con valores por defecto

  try {
    const nuevaUbicacion = new Ubicacion({
      Contenedor: contenedor,
      Ubicación: ubicacion,
      Zona: zona,
      Visado: visado
    });

    const savedUbicacion = await nuevaUbicacion.save(); // Guarda en la base de datos
    console.log(savedUbicacion); // Verifica que los datos se hayan guardado correctamente

    io.emit('contenedorActualizado', { tipo: 'agregar', contenedor: savedUbicacion });

    res.status(201).json({ message: 'Contenedor agregado a MongoDB', data: savedUbicacion });
  } catch (error) {
    console.error("Error al guardar en MongoDB:", error); // Log para diagnosticar el error
    res.status(500).json({ error: 'Error al agregar el contenedor en MongoDB' });
  }
};
```

Figura 8. Código de Controlador para Agregar Contenedor

- Pruebas de Integración: Se probaron escenarios de interacción entre los componentes, el middleware y la base de datos, simulando solicitudes HTTP reales para confirmar que la integración entre componentes y el flujo de datos funciona correctamente. Estas pruebas revelaron errores de sincronización y problemas de duplicidad.
- Pruebas Funcionales: Se utilizó la herramienta Postman, herramienta para probar y validar las APIs RESTful, para enviar solicitudes HTTP y verificar que los endpoints respondieran según lo esperado, incluyendo el manejo de errores y la validación de datos enviados.
- Validación de WebSockets: Se realizaron simulaciones manuales de cambios en la base de datos, como la adición o eliminación de contenedores, para observar el comportamiento de las vistas del sistema en tiempo real, verificando que se generen los eventos y estos lleguen correctamente al cliente.

6.2. Resolución de bugs y hallazgos relevantes

Durante el desarrollo del Back-End los errores fueron detectados y corregidos manualmente mediante la observación de los logs generados por el navegador y el servidor. Para la resolución de problemas técnicos se hizo uso de la documentación proporcionada por los desarrolladores de las librerías y de las tecnologías utilizadas. A continuación, se describen los hallazgos más importantes y sus respectivas soluciones:

- Problemas de duplicidad en la base de datos: En pruebas de integración se detectó que podían registrarse múltiples contenedores con el mismo identificador. Se implementaron validaciones en el controlador, que verifica si el contenedor ya existe antes de ser agregado para evitar registros duplicados en MongoDB.
- Errores en el "parseo" de datos: Las pruebas unitarias revelaron que la función encargada de transformar e interpretar la ubicación de los contenedores tenía inconsistencias en el orden de las coordenadas, provocando una errónea renderización de las visualizaciones interactivas. Se modificó la lógica de parseo para asegurar representar los datos correctamente según su ordenamiento.

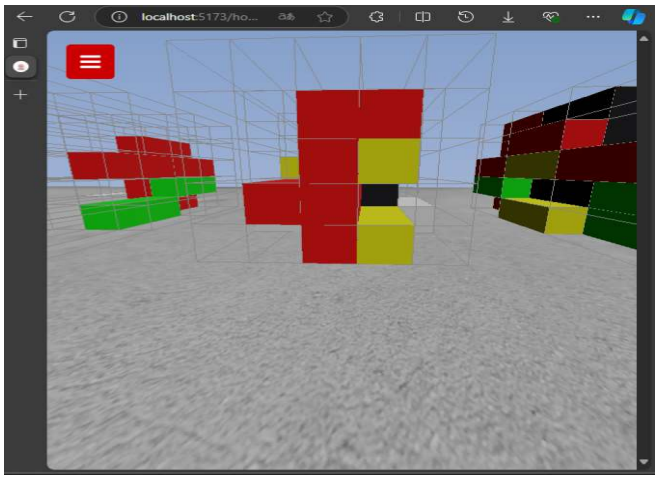


Figura 9. Visualización del patio de contenedores con errores de ubicación

```
export const parseLocation = (ubicacion) => {
    console.log("Ubicación recibida para parseo:", ubicacion);
    if (!ubicacion || typeof ubicacion !== 'string' || !ubicacion.includes('-')) {
        console.warn("Ubicación inválida detectada:", ubicacion);
        return null; // Retorna null si la ubicación no es válida
    }

    const [torreProfundidad, , x, y] = ubicacion.split('-');
    const torre = torreProfundidad.charAt(0);
    const z = parseInt(torreProfundidad.substring(1), 10);

    return {
        torre,
        z,
        x: parseInt(x, 10),
        y: parseInt(y, 10),
```

Figura 10. Función que interpreta la ubicación BAROTI para las visualizaciones

- Sincronización de WebSockets: En la validación de la comunicación en tiempo real se observaban eventos duplicados y actualizaciones innecesarias. Se probaron diferentes configuraciones para optimizar la lógica de emisión de eventos para que solo se envíen cuando ocurre un cambio real en los datos, reduciendo la sobrecarga.

7. Resultados y Discusión

7.1. Evaluación del desempeño de los componentes desarrollados

El sistema logró cumplir con los objetivos planteados satisfactoriamente, logrando que la aplicación web logrará responder correctamente frente a un escenario real.

- API RESTful: La API respondió correctamente como interfaz de comunicación entre Front-End y Back-End, permitiendo que las solicitudes de creación, obtención y eliminación de contenedores fueran manejadas y realizadas correctamente por el Back-End. La solicitud de actualización no se abordó en el alcance del proyecto por falta de tiempo para el desarrollo de esta última funcionalidad, en esencia el método está creado, pero no integrado con el Front-End. La API RESTful siempre juega un papel importante en el desarrollo de software debido a la entrega de información que esta permite, si no se hubiera implementado la API, operaciones como agregar y eliminar contenedores no habrían existido, lo que llevaría a la aplicación web a perder uno de sus propósitos principales, el poder gestionar los contenedores de la empresa.

- WebSockets: La comunicación que estableció WebSockets respondió correctamente y fue de los pilares fundamentales de la aplicación, el garantizar comunicación en tiempo real. Si bien existen métodos alternativos para simular “tiempo real” estos no habrían cubierto el manejo de tantos contenedores en el sistema, además de ser más costoso computacionalmente. El proveer una comunicación continua y bidireccional entre cliente y servidor agrega valor a la aplicación al permitir que los cambios en las vistas se actualicen instantáneamente cuando ocurren los eventos.

- Controladores: El núcleo de la lógica de negocio respondió correctamente y cumplió los objetivos solicitados, realizar operaciones CRUD en la base de datos. Transmitir la solicitud del cliente al servidor era un objetivo, pero poder realizar esa solicitud era otro que eficientemente los controladores lograron cumplir. Sin los controladores, la lógica de negocio de la empresa no habría sido correctamente implementada lo que podría haber llevado a errores en la lógica del funcionamiento de la aplicación.

- Middlewares: Estas funciones lograron establecer una validación exitosa de las solicitudes entrantes a los controladores, facilitando así el flujo de la ejecución de la aplicación para que esta estuviera siempre funcionando sin errores.

7.2. Impacto en el proyecto general

Cada uno de los componentes del Back-End resultaron de suma importancia en las funcionalidades del sistema, ninguno de ellos habría podido ser remplazado por otro componente pese a que en la actualidad con mayor frecuencia aparezcan tecnologías que dejan el desarrollo Back-End en segundo plano. El poseer control del sistema garantiza que todas las funcionalidades puedan ser conocidas por los desarrolladores y no exista caja negra, funcionalidades que no sabemos cómo operan a nivel de código.

La implementación de la sincronización en tiempo real de la aplicación que provee el Back-End es de los aspectos más importantes en la aplicación, esto se evidencia en los comentarios recibidos por la empresa Aexsa. Se recibieron muy buenos comentarios respecto del producto mínimo viable presentado, destacando las visualizaciones interactivas y su capacidad de actualizarse automáticamente cuando ocurre un cambio en el sistema.

8. Conclusiones y Recomendaciones

Desarrollar el Back-End de un sistema resulta una tarea tan importante como cualquier otra del proyecto, pero aun así el desarrollo eficiente de este genera un alto valor en la aplicación completa. Existieron bastantes barreras a lo largo del desarrollo, mayoritariamente barreras técnicas, pero que afortunadamente se logró abordar de la mejor manera posible.

Todas las decisiones tomadas fueron cruciales, desde la selección de las tecnologías hasta los roles de cada miembro del equipo. El haber optado por el Stack tecnológico MERN (MongoDB, Express.js, React, Node.js) fue la mejor opción pese a que la curva inicial de aprendizaje fue alta, ya que ninguno de los miembros del equipo había trabajado previamente con estas tecnologías.

Desde el punto vista de la solución propuesta como tal, una aplicación web que a través de visualizaciones interactivas permitiera gestionar los contenedores en tiempo real garantizó una innovación en la industria de almacenaje extraportuario que también podría llevarse a distintos sectores logísticos, como lo podría ser una bodega de supermercado.

Uno de los principales problemas en el sector extraportuario es realizar una correcta trazabilidad de los contenedores, que si bien nuestra solución logra solucionar esta igual depende del factor humano. Por lo que se recomienda fuertemente considerar implementar IOT para eliminar por completo la dependencia humana.

9. Referencias

[1] GeeksforGeeks, "The Pros and Cons of Node.JS in Web Development", [En línea]. Disponible en: https://www.geeksforgeeks.org/the-pros-and-cons-of-node-js-in-web-development/ [Accedido: 2024].

[2] TatvaSoft, "Advantages of Node.js", [En línea]. Disponible en: https://www.tatvasoft.com/outsourcing/2021/07/advantages-of-node-js.html [Accedido: 2024].

[3] Medium, "Microservices Killer: Modular Monolithic Architecture", [En línea]. Disponible en: https://medium.com/design-microservices-architecture-with-patterns/microservices-killer-modular-monolithic-architecture-ac83814f6862 [Accedido: 2024].

[4] Stack Overflow, "Technology | 2024 Stack Overflow Developer Survey", [En línea]. Disponible en: https://survey.stackoverflow.co/2024/technology [Accedido: 2024].

[5] MDN Web Docs, "Express Tutorial Part 4: Routes and controllers", [En línea]. Disponible en: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes [Accedido: 2024].

[6] AWS, "What is Middleware?", [En línea]. Disponible en: https://aws.amazon.com/what-is/middleware/ [Accedido: 2024].

[7] Aha!, "How To Implement Scrum — 4 Steps for Agile Development Teams", [En línea]. Disponible en: https://www.aha.io/roadmapping/guide/agile/how-to-implement-scrum [Accedido: 2024].

[8] FreeCodeCamp, "Git Best Practices – A Guide to Version Control for Beginners", [En línea]. Disponible en: https://www.freecodecamp.org/news/how-to-use-git-best-practices-for-beginners/ [Accedido: 2024].

[9] FreeCodeCamp, "Git Best Practices – A Guide to Version Control for Beginners", [En línea]. Disponible en: https://www.freecodecamp.org/news/how-to-use-git-best-practices-for-beginners/

[10] QASource, "A Guide To REST API Testing Strategy", [En línea]. Disponible en: https://blog.qasource.com/a-guide-to-rest-api-testing-strategy [Accedido: 2024].

[11] Node.js, "Node.js WebSocket", [En línea]. Disponible en: https://nodejs.org/en/learn/getting-started/websocket [Accedido: 2024].

Agradecimientos. Quiero agradecer a mi madre Carmen, mi padre Marco, mi hermano Joaquín, mi hermana Constanza, mi novia Karina, mis roomies Pedro y Javiera, y toda persona que de alguna forma u otra me vieron crecer en este proceso. Gracias a mis compañeros/amigos de carrera y gracias a mí mismo por no haber dejado la carrera en incontables veces.