



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Optimización de la Gestión de Contenedores: Diseño de una Arquitectura de Software Modular y Escalable

Gerson Amram Vasni Duarte

gerson.vasni@sansano.usm.cl

Óscar Carrasco Vera
Profesor Guía

Resumen

El presente proyecto aborda la problemática de la gestión ineficiente de contenedores vacíos en el patio de la empresa SITRANS, lo cual genera movimientos innecesarios y un uso excesivo de recursos. Para solucionar este problema, se propone el diseño de una arquitectura de software modular y escalable que implemente el método FIFO (First In, First Out), con el objetivo de optimizar el almacenamiento y la recuperación de contenedores. La solución se desarrolla como un servicio accesible mediante una API, que calcula la posición óptima para cada contenedor, considerando sus características únicas.

El informe incluye un análisis de los procesos operativos de SITRANS cuando gestionan los contenedores, la propuesta de una arquitectura basada en microservicios, y la implementación de algoritmos de optimización en Python. A través del diseño modular, se busca que el sistema propuesto sea flexible, escalable y mantenible a largo plazo, permitiendo reducir los movimientos innecesarios de contenedores y mejorar la eficiencia en las operaciones logísticas. Este documento detalla la metodología aplicada, los fundamentos teóricos y el diseño técnico. Concluyendo que las propuestas presentadas permitirán ejecutar las tareas operativas de la empresa de manera más eficiente, al mismo tiempo que establecen una base sólida para futuras mejoras y evoluciones del sistema en función de nuevas necesidades o cambios en el entorno logístico.

Palabras Clave: Gestión de contenedores, optimización, arquitectura modular, FIFO, microservicios



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: Optimización de la Gestión de Contenedores: Diseño de una Arquitectura de Software Modular y Escalable

Nombre del candidato(a): Gerson Amram Vasni Duarte

Carrera / Grado: Ingeniería en Informática

Campus: Viña del Mar ; **Departamento:** Electrotecnia e informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Oscar Carrasco , 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: 06-08-2025

; Firma:

Estudiante o Candidato(a):

Fecha: 06-08-2025

; 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. Introducción

1.1. Contexto

SITRANS, una empresa especializada en la gestión de contenedores vacíos, ha enfrentado dificultades relacionadas con la correcta ubicación y rotación de contenedores en su patio de almacenamiento. La falta de un sistema optimizado para determinar la posición adecuada de los contenedores ha resultado en movimientos innecesarios de maquinaria pesada, lo que a su vez genera un uso ineficiente de los recursos disponibles y aumenta los costos operativos. A su vez, esta problemática ha impactado en la satisfacción del cliente debido a los tiempos prolongados de espera y las dificultades en el despacho ágil de contenedores vacíos. La necesidad de un manejo eficiente no solo surge de la necesidad de ahorrar costos, sino también de la creciente preocupación ambiental que la industria logística enfrenta. Minimizar los movimientos de maquinaria también implica una reducción en las emisiones de gases, mejorando el impacto ambiental de las operaciones.

1.2. Definición del problema

El problema principal que enfrenta SITRANS es la falta de un sistema automatizado que permita ubicar los contenedores vacíos de manera eficiente y optimizada dentro del patio de almacenamiento. Actualmente, los operadores deciden la ubicación de cada contenedor basado en su juicio y experiencia, lo que genera una discrepancia significativa entre la ubicación registrada y la ubicación real de los contenedores. Esta falta de precisión provoca dificultades al momento de localizar y retirar contenedores, lo que afecta directamente la productividad de la empresa. Los movimientos adicionales no solo incrementan los costos por desgaste y energía, sino que también introducen errores humanos que complican la gestión de inventarios.

La falta de un sistema automatizado y centralizado ha llevado a que el personal de SITRANS realice más movimientos de grúa de los necesarios, generando tiempos muertos y elevando los costos. Además, el uso de maquinaria pesada sin un plan optimizado aumenta la posibilidad de fallos mecánicos y de accidentes laborales. Esto demuestra la necesidad de una solución tecnológica que garantice una gestión eficiente de los contenedores vacíos.

1.3. Propuesta de solución

Para abordar esta problemática, la solución propuesta en este proyecto se centra en el desarrollo de un sistema de almacenamiento automatizado basado en el método FIFO (First In, First Out). Este enfoque asegura que los primeros contenedores que ingresan al sistema serán también los primeros en retirarse, lo que minimiza el tiempo y el esfuerzo necesarios para gestionar los movimientos dentro del patio. Este sistema será complementado con funciones de optimización desarrolladas en Python, que permitirán que cada contenedor sea asignado a la posición más adecuada dentro del patio, teniendo en cuenta sus características específicas, tales como el tamaño, el estado y la marca del contenedor.

La implementación de esta solución tecnológica no solo mejorará la eficiencia operativa de SITRANS, sino que también proporcionará una herramienta escalable que podrá adaptarse a futuros cambios en el volumen de contenedores o en los requisitos operacionales de la empresa. Al diseñar el sistema como un servicio accesible a través de una API, SITRANS podrá integrarlo fácilmente con sus sistemas actuales, y permitirá a la empresa aprovechar de inmediato las ventajas del nuevo sistema sin necesidad de realizar cambios significativos en su infraestructura tecnológica existente.

1.4. Marco Teórico Adoptado

El marco teórico de esta solución se fundamenta en principios de diseño de software que promueven la modularidad y la escalabilidad. Estos principios aseguran que el sistema será fácilmente mantenible y podrá evolucionar con el tiempo para adaptarse a las necesidades cambiantes de la empresa. Se han utilizado patrones de diseño ampliamente reconocidos, como el Modelo-Vista-Controlador (MVC) y los Microservicios, que no solo facilitan el desarrollo y la integración con otros sistemas, sino que también proporcionan una base sólida para el crecimiento futuro del sistema. El enfoque modular permite que distintos componentes puedan ser actualizados o reemplazados sin afectar al sistema completo, lo que reduce los costos de mantenimiento y mejora la resiliencia del sistema frente a fallos o cambios imprevistos.

1.5. Objetivos Planteados

En el contexto de las operaciones logísticas de SITRANS, el manejo eficiente de contenedores vacíos representa un desafío clave para optimizar los recursos y mejorar la productividad. La necesidad de reducir los movimientos innecesarios y garantizar una rotación efectiva de los contenedores se alinea con los objetivos estratégicos de la empresa para maximizar la eficiencia operativa y minimizar los costos asociados.

Como objetivo general, se propone desarrollar una solución integral para la gestión eficiente de contenedores vacíos en SITRANS, que optimice las operaciones de almacenamiento y retiro mediante un sistema escalable, modular y basado en el método FIFO, reduciendo los movimientos innecesarios y maximizando la productividad operativa.

El desarrollo de este proyecto se enfoca en ofrecer una solución que aproveche tecnologías modernas y un diseño arquitectónico robusto, asegurando que el sistema no solo cumpla con los requerimientos actuales, sino que también esté preparado para adaptarse a futuras necesidades.

Bajo esta perspectiva, se plantean los siguientes objetivos específicos para guiar el desarrollo del sistema:

- Desarrollar un sistema de gestión de contenedores vacíos que minimice los movimientos innecesarios en el patio de almacenamiento.

- Implementar el método FIFO para asegurar que los contenedores sean retirados en el orden correcto, optimizando los movimientos dentro del patio.
- Diseñar una arquitectura modular y escalable que permita integrar nuevas funcionalidades en el futuro y facilite el mantenimiento del sistema.
- Validar la efectividad del sistema a través de simulaciones y pruebas, asegurando que la solución propuesta cumple con los requisitos operacionales de SITRANS.

1.6. Metodología Aplicada

En cuanto a la metodología aplicada en este proyecto, se ha seguido un enfoque riguroso que combina la investigación de campo en las instalaciones de SITRANS con modelos de simulación para evaluar la viabilidad de la solución. El primer paso fue realizar un análisis exhaustivo de los procesos actuales de la empresa, identificando los puntos críticos donde se generan las ineficiencias. A partir de este análisis, se diseñaron los algoritmos de optimización y se implementaron en Python. Estos algoritmos fueron luego validados a través de simulaciones, lo que permitió ajustar los parámetros del sistema para asegurar un rendimiento óptimo.

1.7. Descripción Capítulos

En cuanto a la estructura del presente informe, se organiza en los siguientes capítulos para facilitar su comprensión y profundizar en los aspectos más relevantes del proyecto.

– Capítulo 1: Introducción

Este capítulo plantea el contexto del proyecto, destacando los retos de SITRANS en la gestión eficiente de contenedores vacíos. Se define el propósito de desarrollar un sistema que optimice los movimientos en el patio mediante el enfoque FIFO y una arquitectura modular y escalable. También se presentan los objetivos clave, orientados a reducir costos operativos y validar la solución a través de pruebas y simulaciones.

– Capítulo 2: Marco Teórico

Este capítulo aborda los conceptos fundamentales que sustentan el proyecto, incluyendo los principios de diseño de software que garantizan modularidad, escalabilidad y mantenibilidad. También se exploran los patrones de arquitectura seleccionados, como el Modelo-Vista-Controlador (MVC) y los Microservicios, los cuales juegan un papel crucial en la organización eficiente del sistema.

– Capítulo 3: Metodología

Se describe el enfoque metodológico seguido para analizar los procesos operativos de SITRANS y diseñar la solución propuesta. Este capítulo incluye la descripción de las herramientas utilizadas, como diagramas UML y simulaciones, que permitieron modelar y validar las estrategias de optimización antes de su implementación.

- **Capítulo 4: Análisis de Requerimientos**

En este capítulo se identifican y describen los requerimientos funcionales y no funcionales del sistema. Se destaca cómo se garantiza la modularidad mediante la separación de funcionalidades en módulos independientes y cómo la escalabilidad se asegura al diseñar una solución adaptable a incrementos en la demanda operativa.

- **Capítulo 5: Diseño de la Arquitectura**

Se detalla la arquitectura seleccionada para el sistema, incluyendo diagramas que muestran los componentes principales, sus interacciones y los módulos clave. Además, se justifica cada decisión de diseño en función de los objetivos del proyecto, como la mejora en la eficiencia operativa y la facilidad de mantenimiento.

- **Capítulo 6: Comparación de Arquitecturas**

Este capítulo presenta un análisis de diferentes opciones arquitectónicas consideradas durante el diseño del sistema. Se evalúa el impacto de cada enfoque en el rendimiento, el mantenimiento y la escalabilidad, destacando las razones por las cuales se eligió la arquitectura final.

- **Capítulo 7: Implementación**

Describe los detalles de la implementación de la arquitectura seleccionada. Incluye cómo se organizaron los módulos y las funciones en archivos separados, su integración en el archivo principal (main.py), y cómo se documentó cada componente para garantizar su comprensión y evolución futura.

- **Capítulo 8: Resultados y Evaluación**

Se analizan los resultados obtenidos tras la implementación de la arquitectura. Este capítulo incluye una evaluación del desempeño del sistema, identificando fortalezas, como la reducción en los movimientos de contenedores, y áreas de mejora para futuros desarrollos.

- **Conclusiones y Recomendaciones**

Finalmente, este capítulo resume los logros alcanzados con el proyecto, destacando cómo se logró optimizar los movimientos de contenedores vacíos mediante una arquitectura modular y escalable. Se incluyen recomendaciones para futuras iteraciones del sistema, orientadas a fortalecer la solución y adaptarla a posibles cambios en las operaciones de SITRANS.

2. Marco Teórico

El diseño de software eficaz es un elemento esencial en el desarrollo de soluciones tecnológicas robustas que sean capaces de satisfacer los requerimientos actuales y adaptarse a los desafíos futuros. En el caso del proyecto para SITRANS, el objetivo principal es crear un sistema que optimice la gestión de contenedores vacíos mediante una arquitectura modular y escalable, apoyada en principios sólidos de diseño de software y patrones de arquitectura ampliamente reconocidos. Este marco teórico explora los fundamentos que guían el desarrollo de la solución, resaltando la importancia de adoptar enfoques técnicos que promuevan la eficiencia operativa, la flexibilidad y la capacidad de expansión del sistema.

2.1. Principios de Diseño de Software

El diseño de software moderno se fundamenta en principios que buscan maximizar la calidad, reducir la complejidad y garantizar la sostenibilidad del sistema a lo largo del tiempo. Para el desarrollo del sistema de gestión de contenedores vacíos para SITRANS, se han adoptado varios principios clave, cada uno de los cuales contribuye a resolver diferentes aspectos del problema y a garantizar la implementación de una solución robusta y duradera.

2.1.1. Modularidad

La modularidad es uno de los pilares fundamentales del diseño de software. Este principio implica dividir el sistema en componentes independientes o módulos, cada uno de los cuales se encarga de una funcionalidad específica.

En el proyecto para SITRANS, esta división permite:

- Desarrollar, probar y mantener módulos de manera independiente.
- Actualizar o mejorar un módulo sin afectar la operación del resto del sistema.
- Facilitar la integración de nuevas funcionalidades, como la gestión de nuevos tipos de contenedores o el soporte para operaciones más complejas.

Por ejemplo, se puede implementar un módulo dedicado exclusivamente a calcular las posiciones óptimas para los contenedores basándose en parámetros específicos, como la marca, el tamaño y el estado. Si en el futuro se requiere mejorar este algoritmo, solo se necesitará modificar este módulo, dejando intactos los demás.

2.1.2. Escalabilidad

La escalabilidad asegura que el sistema pueda manejar incrementos en la carga de trabajo sin deteriorar su rendimiento.

Para SITRANS, esto es crítico debido a la variabilidad en el volumen de operaciones logísticas. Durante períodos de alta demanda, como en temporadas pico, el sistema debe ser capaz de gestionar un mayor número de contenedores sin interrupciones. La escalabilidad se logra mediante:

- Escalabilidad vertical: Mejorar la capacidad de los servidores donde se ejecuta el sistema.
- Escalabilidad horizontal: Añadir más instancias de servicios específicos para distribuir la carga.

En la práctica, esto significa que, si el número de consultas a la API aumenta considerablemente, solo el módulo de la API necesita ser replicado o ajustado, sin afectar el resto del sistema.

2.1.3. Mantenibilidad

La mantenibilidad se refiere a la facilidad con la que un sistema puede ser modificado, corregido o ampliado. Este principio es esencial para garantizar la longevidad del software.

En el caso de SITRANS, la solución está diseñada para:

- Utilizar código limpio y estructurado, facilitando su comprensión y modificación por parte de los desarrolladores.
- Incorporar herramientas de pruebas automatizadas que permitan identificar y solucionar errores rápidamente.
- Documentar cada módulo y componente de manera clara, proporcionando a futuros desarrolladores toda la información necesaria para realizar actualizaciones sin dificultad.

Estas características reducen el tiempo y el costo asociados al mantenimiento del sistema.

2.1.4. Principio de Responsabilidad Única (SRP)

El SRP establece que cada componente o módulo debe encargarse de una única tarea. Este principio ayuda a reducir la complejidad del sistema y evita que los módulos tengan responsabilidades cruzadas.

En el sistema de SITRANS, ejemplos de aplicación del SRP incluyen:

- Un módulo dedicado a gestionar el almacenamiento
- Otro modulo para la recuperación de contenedores.
- Otro módulo específico para manejar las interacciones con la API.
- Un módulo de análisis y generación de reportes.

Esta especialización permite una mejor organización del código y asegura que los cambios en un módulo no afecten al resto.

2.1.5. Separación de Preocupaciones

La separación de preocupaciones implica dividir el sistema en partes que gestionen distintas responsabilidades. Este principio facilita la comprensión, el desarrollo y el mantenimiento del sistema.

En el caso de SITRANS, se ha implementado separando las funcionalidades en capas:

- Capa de datos: Responsable de la gestión de la base de datos y la manipulación de información.
- Capa lógica: Donde se realiza el cálculo de posiciones óptimas y otras operaciones.
- Capa de presentación: Encargada de la interfaz de usuario y las interacciones con los operarios.

Esta estructura modular permite realizar cambios en una capa sin afectar las otras, promoviendo la estabilidad y la facilidad de desarrollo.

2.2. Patrones de Arquitectura Comunes

Para diseñar un sistema que cumpla con los principios anteriores, es necesario adoptar patrones de arquitectura que hayan demostrado su efectividad en problemas similares. Los patrones seleccionados para este proyecto son:

2.2.1. Modelo-Vista-Controlador (MVC)

El patrón MVC es una elección natural para aplicaciones que requieren una separación clara entre la lógica del negocio, la interfaz de usuario y el control de la aplicación.

En el sistema de SITRANS:

- **Modelo:** Gestiona la información relacionada con los contenedores, incluyendo sus características y ubicaciones.
- **Vista:** Proporciona al usuario una interfaz para consultar y gestionar los contenedores.
- **Controlador:** Maneja las interacciones del usuario y actualiza tanto el modelo como la vista según sea necesario.

Este patrón facilita el mantenimiento del sistema, ya que permite modificar cada capa de forma independiente.

2.2.2. Microservicios

La arquitectura de microservicios divide la aplicación en servicios pequeños, independientes y especializados.

En el caso de SITRANS, los microservicios se utilizan para:

- Gestionar el cálculo de posiciones óptimas.
- Controlar el almacenamiento y recuperación de contenedores.
- Proveer acceso a través de la API.

Esta arquitectura mejora la escalabilidad y la resiliencia del sistema, ya que cada microservicio puede desarrollarse, desplegarse y escalarse de manera independiente.

2.3. Importancia de la Arquitectura Modular y Escalable

El diseño modular y escalable es una necesidad práctica en un entorno como el de SITRANS, donde los volúmenes de trabajo y las necesidades tecnológicas pueden cambiar drásticamente.

Beneficios de la modularidad:

1. Simplifica la integración de nuevas funcionalidades.
2. Reduce el tiempo de desarrollo.
3. Facilita el diagnóstico y la corrección de errores.

Beneficios de la escalabilidad:

1. Permite manejar incrementos en la demanda.
2. Optimiza el uso de recursos, escalando solo los componentes necesarios.

Al adoptar estos principios, la solución propuesta no solo mejorará la eficiencia operativa actual de SITRANS, sino que también proporcionará un sistema robusto y preparado para futuros desafíos.

3. Metodología

El desarrollo de una solución eficiente para la gestión de contenedores vacíos en SITRANS requiere una planificación detallada, análisis meticuloso y un diseño técnico robusto que se ajuste a las necesidades actuales y futuras de la empresa. Este capítulo describe el enfoque metodológico adoptado en el proceso de desarrollo, que abarca desde el análisis inicial hasta la implementación y validación del sistema propuesto. El objetivo principal es garantizar que el sistema sea escalable, modular y adaptable, de forma que no solo resuelva los problemas actuales de SITRANS, sino que también pueda evolucionar con el tiempo y crecer con las necesidades de la empresa.

3.1. Enfoque Utilizado para el Análisis y Diseño

Para desarrollar la solución, se adoptó un enfoque basado en el ciclo iterativo de diseño, implementación y validación conocido como Desarrollo Ágil. Este enfoque metodológico permite que el sistema evolucione a través de una serie de fases y ciclos iterativos, lo que facilita la integración de nuevas funcionalidades y la adaptación a los cambios en los requisitos operacionales de SITRANS. El Desarrollo Ágil se basa en la colaboración continua con el cliente, la retroalimentación constante y la capacidad de realizar ajustes rápidos en el diseño según los resultados obtenidos en cada iteración. De esta forma, el sistema se desarrolla de manera flexible, permitiendo que se optimicen las funciones según las necesidades emergentes.

3.1.1. Análisis de Requerimientos

El primer paso en el proceso metodológico fue la recopilación y el análisis exhaustivo de los requerimientos operativos de SITRANS. Esta fase inicial es fundamental para comprender en detalle las necesidades de la empresa y establecer una base sólida sobre la cual desarrollar la solución.

Actividades del Análisis de Requerimientos:

- **Observación Directa de los Procesos Actuales:** Se realizó una observación detallada de los procesos de gestión de contenedores en las instalaciones de SITRANS. Esto incluyó el análisis de las tareas realizadas por los operadores, las rutas de la maquinaria pesada y los puntos de congestión dentro del patio de almacenamiento. A través de la observación directa, se identificaron áreas críticas donde se generaban ineficiencias operativas y movimientos innecesarios de los contenedores.
- **Entrevistas con los Operadores y Personal Clave:** Se llevaron a cabo entrevistas con los operadores del sistema, el personal de mantenimiento y los responsables de la logística. Estas entrevistas proporcionaron información valiosa sobre los desafíos diarios que enfrentan al manejar los contenedores vacíos, las dificultades para encontrar y recuperar los contenedores y las expectativas en relación con un nuevo sistema.

- **Revisión de los Sistemas Existentes:** Se revisaron los sistemas actuales de gestión de contenedores en SITRANS para identificar sus limitaciones y áreas de mejora. Este análisis también permitió detectar posibles integraciones con el nuevo sistema que ayudarían a optimizar las operaciones sin requerir cambios drásticos en la infraestructura tecnológica existente.

El resultado de este análisis fue un conjunto claro de requisitos operacionales, que incluyó la necesidad de reducir los movimientos de maquinaria, mejorar la precisión en la localización de los contenedores y asegurar que el sistema pudiera gestionar de manera eficiente el flujo de contenedores vacíos dentro del patio.

3.1.2. Modelado del Sistema

Una vez definidos los requerimientos, se procedió a modelar el sistema propuesto. El modelado del sistema es crucial para visualizar los procesos y la interacción de los componentes del sistema, asegurando que todos los elementos trabajen de forma cohesiva y eficiente.

Herramientas y Técnicas Utilizadas en el Modelado:

- **Diagramas de Casos de Uso:** Los diagramas de casos de uso se utilizaron para describir las funcionalidades principales del sistema desde el punto de vista de los usuarios (operadores y administradores). Estos diagramas permiten comprender las interacciones entre los usuarios y el sistema, detallando cómo se llevará a cabo la gestión de los contenedores y las funciones críticas como el almacenamiento, la recuperación y la consulta de información.
- **Diagramas de Clases:** Para estructurar la información y los componentes del sistema, se emplearon diagramas de clases que modelan las entidades principales, como contenedores, bloques de almacenamiento, operadores, y los distintos servicios del sistema. Este modelo permite organizar y categorizar las funciones y los datos de manera coherente y facilita su implementación en el sistema final.
- **Diagramas de Flujo de Datos:** Los diagramas de flujo de datos se utilizaron para representar cómo fluye la información dentro del sistema. Estos diagramas detallan el intercambio de datos entre los distintos componentes, como la base de datos, el servicio de optimización y la interfaz de usuario, lo que facilita el diseño de la arquitectura y ayuda a detectar posibles cuellos de botella en el flujo de información.

Este modelado detallado proporcionó una representación visual de cómo operará el sistema, lo que permitió identificar posibles mejoras en el proceso antes de pasar a la implementación.

3.1.3. Diseño de la Arquitectura

El siguiente paso fue el diseño de la arquitectura del sistema, que fue cuidadosamente estructurada para garantizar su escalabilidad, modularidad y adaptabilidad. La arquitectura del sistema es el componente fundamental que determina la eficiencia, la fiabilidad y la facilidad de mantenimiento del sistema en el futuro.

Enfoque de Diseño:

- **Modularidad:** La arquitectura se diseñó bajo principios de modularidad, asegurando que cada componente del sistema pudiera ser desarrollado, probado y desplegado de manera independiente. Esto facilita las actualizaciones y modificaciones, ya que nuevos módulos o funcionalidades pueden ser añadidos sin afectar el funcionamiento del sistema en su totalidad. Los módulos principales incluyen la interfaz de usuario, el servicio de optimización, la base de datos y los microservicios de gestión de contenedores.
- **Escalabilidad:** Se optó por una arquitectura que garantice que el sistema pueda adaptarse fácilmente al crecimiento futuro de SITRANS. Esto significa que, a medida que la empresa amplíe su operación o que se incrementen los volúmenes de contenedores, el sistema podrá manejar el aumento de carga sin pérdida de rendimiento. Para lograr esto, se implementó una arquitectura basada en microservicios, lo que permite la distribución de cargas y la adición de nuevos servicios de manera autónoma.
- **Patrón Modelo-Vista-Controlador (MVC):** El patrón MVC se utilizó para organizar la interfaz de usuario, la lógica de negocio y el acceso a los datos. Esto facilita el mantenimiento y mejora la modularidad del sistema, permitiendo que las actualizaciones de la interfaz de usuario o la lógica de negocio se realicen sin afectar otras partes del sistema. Este patrón también facilita las pruebas y la depuración, lo que mejora la fiabilidad del sistema.

3.1.4. Validación y Optimización

Una vez que se completó el diseño de la arquitectura y el modelado del sistema, se procedió a realizar las pruebas iniciales y la validación de la solución a través de simulaciones. Esta fase es fundamental para asegurar que el sistema funcione correctamente y cumpla con los requisitos operacionales definidos en la etapa de análisis.

Actividades en la Fase de Validación:

- **Simulación de Carga de Trabajo:** Se utilizaron datos simulados para replicar las condiciones reales de las operaciones en SITRANS. Estas simulaciones permitieron evaluar cómo el sistema gestionaba el almacenamiento, la recuperación y la rotación de contenedores, y verificar si los algoritmos de optimización realmente reducían los movimientos innecesarios y mejoraban los tiempos de acceso.

- **Ajuste de Algoritmos de Optimización:** Durante las simulaciones, los algoritmos de optimización fueron ajustados continuamente para mejorar el rendimiento del sistema. Se ajustaron los parámetros del algoritmo, como el tamaño de los bloques de almacenamiento, las estrategias de asignación de contenedores y las condiciones de entrada de datos, con el fin de maximizar la eficiencia.
- **Evaluación del Rendimiento:** Finalmente, se realizaron pruebas de rendimiento para evaluar el impacto del nuevo sistema en los tiempos de respuesta y en la reducción de los movimientos. Estos resultados fueron comparados con los procesos actuales de SITRANS para asegurar que el sistema propuesto ofreciera mejoras tangibles.

Este enfoque metodológico basado en pruebas iterativas permitió crear un sistema robusto y flexible, capaz de adaptarse a las demandas cambiantes y optimizar las operaciones de SITRANS a largo plazo.

3.2. Herramientas de Modelado Empleadas

Para garantizar la claridad, precisión y eficiencia en el diseño y desarrollo del sistema propuesto, se emplearon diversas herramientas de modelado, cada una seleccionada en función de su capacidad para satisfacer necesidades específicas del proyecto. Estas herramientas jugaron un papel clave en la estructuración de los procesos, la implementación técnica y la validación de la solución.

- **Python**

El lenguaje de programación Python fue elegido para desarrollar las funciones de optimización del sistema, gracias a su versatilidad y a su robusto ecosistema de bibliotecas.

- a) **Funciones de optimización:** Se implementaron algoritmos para determinar las posiciones óptimas de los contenedores en función de parámetros como marca, tamaño, estado y probabilidad de recuperación.
- b) **Flexibilidad y mantenibilidad:** Python permitió un desarrollo ágil, con código limpio y fácilmente modificable para adaptarse a cambios o mejoras futuras.

La elección de Python no solo garantizó la calidad técnica de las soluciones, sino también su capacidad para evolucionar con las necesidades del negocio.

- **SQL Server**

Para la gestión de los datos relacionados con los contenedores, se utilizó SQL Server como sistema de gestión de base de datos durante el desarrollo inicial y las pruebas locales.

- a) **Almacenamiento estructurado:** La base de datos incluye información clave sobre los contenedores, como su identificación, características (marca, tamaño, estado), fecha de ingreso y salida, y ubicación dentro del patio.

- b) **Integridad de datos:** Se definieron restricciones y relaciones que aseguraron la consistencia y validez de los datos almacenados.
- c) **Consultas eficientes:** La implementación de índices y consultas optimizadas permitió obtener rápidamente información sobre los contenedores y su ubicación.

SQL Server proporcionó una plataforma confiable para manejar datos críticos durante las fases iniciales del proyecto, asegurando precisión y accesibilidad.

- **Postman**

Postman se utilizó para probar y validar la funcionalidad de la API desarrollada como parte de la solución, la cual permite a SITRANS consultar las posiciones óptimas de los contenedores.

- a) **Pruebas exhaustivas:** Se verificaron todas las solicitudes y respuestas de la API, asegurando que la información proporcionada fuera precisa y consistente.
- b) **Validación de rendimiento:** Se evaluó el tiempo de respuesta de las consultas, garantizando que el servicio fuera eficiente incluso bajo cargas de trabajo altas.
- c) **Documentación:** Postman facilitó la generación de documentación clara para la API, lo que simplificó su integración con otros sistemas de SITRANS.

Gracias a Postman, se lograron garantizar la calidad y fiabilidad del servicio API, un componente crítico del sistema propuesto.

3.3. Iteraciones de Mejora

El desarrollo del sistema para la gestión de contenedores vacíos en SITRANS siguió un enfoque iterativo centrado en la mejora continua. Este proceso involucró ciclos sucesivos de diseño, implementación, pruebas y refinamiento, permitiendo ajustar tanto los algoritmos de optimización como la arquitectura general del sistema. A continuación, se detallan las características y beneficios de este enfoque:

- **Diseño Iterativo**

Cada iteración comenzó con un análisis de los resultados obtenidos en la versión anterior del sistema. Se identificaron áreas de mejora basadas en métricas de desempeño, como:

- a) **Reducción en movimientos de contenedores:** Cada iteración evaluó la eficiencia de las posiciones sugeridas para minimizar el número de movimientos.
- b) **Tiempo de acceso a contenedores:** Se analizaron los tiempos promedio para la recuperación de contenedores y se ajustaron las estrategias para reducirlos.
- c) **Distribución óptima:** Se verificó que los algoritmos garantizaran una distribución equilibrada y lógica en función de los parámetros como tipo, estado y probabilidad de recuperación.

Este análisis permitió introducir mejoras específicas en el diseño de los algoritmos y en la estructura de los datos. Principalmente con la asignación automática de un valor individual para cada contenedor, teniendo en cuenta las especificaciones de cada uno, siendo los con valor mayor los más prioritarios para su extracción, estos valores son distintos cada día.

- **Pruebas y Validación en Cada Iteración**

Cada versión del sistema se sometió a pruebas exhaustivas, tanto con datos simulados como con datos reales proporcionados por SITRANS. Estas pruebas incluyeron:

- a) **Simulaciones operativas:** Evaluación de escenarios complejos para verificar cómo el sistema respondía a diferentes volúmenes y patrones de flujo de contenedores.
- b) **Pruebas de estrés:** Simulación de escenarios con una alta carga de trabajo para medir el rendimiento y la estabilidad del sistema.
- c) **Validación de datos:** Comparación de las posiciones sugeridas con las posiciones ideales determinadas manualmente, asegurando la precisión de los algoritmos.

El feedback obtenido durante estas pruebas alimentó la siguiente iteración, mejorando continuamente el desempeño del sistema.

- **Refinamiento de Algoritmos**

Uno de los aspectos clave del proceso iterativo fue el perfeccionamiento de los algoritmos de optimización. Se realizaron las siguientes mejoras significativas:

- a) **Ajuste de parámetros:** Los parámetros iniciales del algoritmo, como el peso asignado a la probabilidad de recuperación y el estado del contenedor, fueron ajustados para reflejar mejor las prioridades operativas de SITRANS.
- b) **Incorporación de reglas específicas:** A medida que se identificaban patrones operativos únicos en SITRANS, se integraron reglas adicionales para garantizar que el sistema respetara las restricciones prácticas, como la necesidad de mantener ciertos tipos de contenedores alejados entre sí.
- c) **Optimización de desempeño:** Se redujo el tiempo de ejecución de los cálculos optimizando la lógica interna del algoritmo y empleando estructuras de datos más eficientes.

- **Beneficios del Enfoque Iterativo**

Este enfoque permitió:

- a) **Adaptabilidad:** Incorporar cambios en las prioridades de SITRANS o en las condiciones del entorno logístico sin interrumpir el desarrollo.
- b) **Incremento constante en la calidad:** Cada ciclo de mejora elevó el nivel de precisión, rendimiento y usabilidad del sistema.
- c) **Reducción de riesgos:** Identificar y corregir problemas desde las primeras etapas del desarrollo, evitando costosos retrabajos en fases avanzadas.

4. Análisis de Requerimientos

El análisis de requerimientos constituye un pilar fundamental en el desarrollo de un sistema eficiente y adaptable para la gestión de contenedores vacíos en SITRANS. Este análisis no solo garantiza que la solución cumpla con las necesidades operativas actuales, sino que también se alinee con las proyecciones futuras de la empresa. Al identificar y detallar los requerimientos funcionales y no funcionales, se establecen las bases para un sistema modular, escalable y confiable que optimiza la logística de contenedores, reduce costos y mejora la eficiencia operativa.

4.1. Requerimientos Funcionales

Los requerimientos funcionales definen las capacidades y servicios que el sistema debe ofrecer para cumplir con los objetivos operativos. Estos requerimientos son esenciales para la implementación de un flujo de trabajo eficiente y automatizado.

- **Registro y gestión de contenedores**

- **Ingreso automatizado:** Incorporar una API que registre automáticamente cada contenedor al ingresar al patio, capturando datos clave como tamaño, marca, estado y fecha de ingreso.
- **Asignación de posiciones:** Utilizar algoritmos para calcular dinámicamente la ubicación óptima del contenedor en función de su prioridad de salida y características específicas.
- **Actualización en tiempo real:** Mantener actualizadas las posiciones de los contenedores tras cada movimiento, reflejando los cambios inmediatamente en el sistema. Este proceso se lleva a cabo después de una extracción de un contenedor, volviendo a reubicar los contenedores que debieron ser extraídos para alcanzar al contenedor objetivo.

Ejemplo:



Figura1: Ejemplo funcional

Supongamos que el contenedor marcado con la X es el que se debe extraer, para ello se debe primero extraer los cuatro de la primera columna, para los cuales se volverá a calcular su ubicación óptima para una su futura extracción.

- **Optimización del almacenamiento y recuperación**
 - **Rutas mínimas:** Definir rutas optimizadas para el movimiento de grúas, reduciendo al mínimo los desplazamientos innecesarios.
 - **Gestión de conflictos:** Asignar posiciones que minimicen conflictos operativos, como mantener los contenedores con alta prioridad de salida en ubicaciones accesibles.
- **Integración con sistemas externos**
 - **Compatibilidad:** Asegurar que el sistema pueda integrarse con las herramientas de gestión actuales de SITRANS y sea capaz de interoperar con soluciones futuras.
- **Control de acceso**
 - **Roles y permisos:** Implementar un sistema que limite el acceso a operaciones críticas, garantizando que usuarios autorizados realicen modificaciones o retiradas de contenedores.
- **Automatización de notificaciones**
 - **Alertas:** Enviar notificaciones para situaciones como excedencia en el tiempo de permanencia de un contenedor o la detección de bloqueos por falta de espacio en áreas específicas.
- **Relación con la Modularidad**

Estos requerimientos se estructuran en módulos independientes que interactúan a través de interfaces bien definidas, favoreciendo:

 - **Actualizaciones aisladas:** Permitir cambios en un módulo sin afectar al resto del sistema.
 - **Flexibilidad:** Incorporar nuevos servicios de forma sencilla, como reportes personalizados o algoritmos avanzados de optimización.

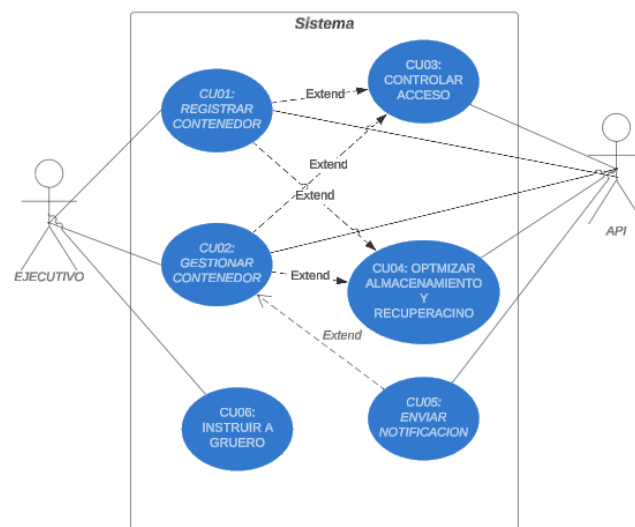


Figura 1. Casos de uso.

4.2. Requerimientos No Funcionales

Los requerimientos no funcionales se centran en los atributos de calidad del sistema, garantizando un desempeño robusto y sostenible en diferentes escenarios operativos.

- **Escalabilidad**

- **Crecimiento horizontal:** Diseñar el sistema para que pueda manejar incrementos en el volumen de operaciones mediante la adición de nuevos servidores o instancias.
- **Flexibilidad:** Utilizar una arquitectura desacoplada que facilite la incorporación de nuevos módulos o funciones sin afectar al sistema existente.

- **Modularidad**

- **Separación lógica:** Dividir el sistema en componentes específicos, como gestión de datos, lógica de optimización e interfaz de usuario, para simplificar el mantenimiento y la actualización.
- **Reutilización:** Diseñar módulos genéricos que puedan ser aplicados en otros contextos operativos de SITRANS o sus socios logísticos.

- **Disponibilidad y Tolerancia a Fallos**

- **Alta disponibilidad:** Garantizar un tiempo de actividad superior al 99.9% mediante redundancia y estrategias de balanceo de carga.
- **Recuperación rápida:** Implementar copias de seguridad automáticas y mecanismos de restauración ante fallos para minimizar el tiempo de inactividad.

- **Rendimiento**

- **Baja latencia:** Asegurar que las solicitudes al sistema, como consultas de posiciones o generación de reportes, se procesen en menos de 2 segundos.
- **Procesamiento concurrente:** Permitir el manejo simultáneo de más de 1,000 solicitudes a la API sin degradar el rendimiento.

- **Seguridad**

- **Control de acceso:** Incorporar autenticación basada en tokens y roles para restringir accesos no autorizados.

- **Facilidad de mantenimiento**

- **Código modular y documentado:** Organizar el código en estructuras claras, con documentación detallada para facilitar el desarrollo y corrección de errores.
- **Actualizaciones aisladas:** Permitir que los módulos se actualicen individualmente sin afectar el funcionamiento global del sistema.

5. Diseño de la Arquitectura

El diseño del sistema es un componente esencial del presente proyecto, ya que establece las bases para una solución que no solo cumple con los requerimientos actuales de SITRANS, sino que también está preparada para expandirse y adaptarse a futuros desafíos operativos y empresariales. Esta sección describe de manera exhaustiva la arquitectura del sistema, los principios que guían su construcción, y las decisiones de diseño tomadas para garantizar su modularidad, escalabilidad y mantenibilidad.

5.1. Principios Fundamentales del Diseño

- **Modularidad**

Cada funcionalidad del sistema ha sido encapsulada en módulos independientes. Este principio garantiza que cada componente sea desarrollado y probado de manera aislada, lo que minimiza el impacto de errores o cambios en otros módulos.

- **Organización por Funciones:** Cada módulo está organizado en funciones específicas que representan operaciones clave, como almacenamiento de contenedores, cálculo de posiciones óptimas, recuperación de información y comunicación a través de la API.
- **Separación por Archivos:** Las funciones están distribuidas en archivos individuales que agrupan funcionalidades relacionadas. Por ejemplo, todas las operaciones de la base de datos están contenidas en un archivo dedicado, mientras que las operaciones de cálculo y optimización se encuentran en otro.

- **Escalabilidad**

La arquitectura modular facilita el escalamiento tanto horizontal como vertical del sistema:

- **Escalabilidad Horizontal:** Permite añadir más instancias de módulos para manejar un mayor volumen de solicitudes.
- **Escalabilidad Vertical:** Permite mejorar módulos específicos para soportar cargas más altas o funcionalidades más complejas.
- **Preparación para Nuevas Funciones:** El sistema está diseñado para integrar nuevas capacidades con un mínimo esfuerzo, gracias a su estructura modular.

- **Mantenibilidad y Extensibilidad**

- **Comentarios Detallados:** Cada función y archivo incluye comentarios claros que explican su propósito y funcionamiento, facilitando la comprensión para futuros desarrolladores.
- **Uso de Estándares de Programación:** Se han seguido las mejores prácticas de programación, como nombres descriptivos de funciones, modularidad en el código y documentación exhaustiva.
- **Facilidad para Incorporar Cambios:** La separación en módulos permite implementar cambios o correcciones en un área específica sin riesgo de comprometer el funcionamiento global del sistema.

5.2. Arquitectura del Sistema

El sistema utiliza una arquitectura basada en microservicios con componentes claramente definidos que interactúan entre sí a través de interfaces bien estructuradas.

5.2.1. Componentes Principales

- **Módulo de Gestión de Contenedores**
 - Responsable de manejar todas las operaciones relacionadas con los contenedores, como registrar su entrada, calcular la posición óptima de almacenamiento y gestionar su salida.
 - Incluye funciones para clasificar contenedores según características específicas, como tamaño, marca y estado.
- **Módulo de Optimización**
 - Implementa algoritmos para calcular la posición óptima de los contenedores, considerando factores como el método FIFO y la eficiencia de movimientos.
 - Está diseñado para ser escalable, permitiendo añadir nuevos algoritmos o ajustar los existentes según las necesidades de SITRANS o de futuros clientes.
- **Servicio API**
 - Proporciona una interfaz estándar para que SITRANS y otras empresas interactúen con el sistema.
 - Permite operaciones como registrar nuevos contenedores, consultar su ubicación o generar reportes.
 - Desarrollado con un enfoque modular para facilitar su expansión con nuevos endpoints o funcionalidades.
- **Base de Datos Modular**
 - Diseñada para almacenar información de contenedores, configuraciones del sistema y registros operativos.
 - El diseño de la base de datos permite su expansión con tablas adicionales sin alterar las existentes.

5.2.2. Estructura del Código

El código está organizado en carpetas y archivos según su funcionalidad.

Carpeta DINODOCK_V0.1: Es la carpeta que almacena todos los demás archivos y funciones.

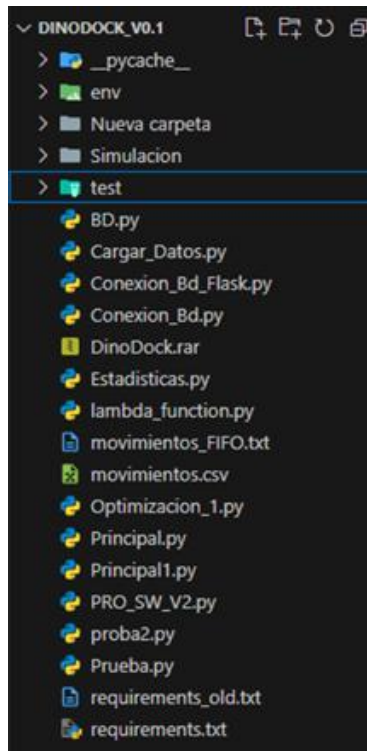


Figura 3. Captura de carpeta con archivos.

Carpeta Simulación: Incluye archivos para replicar la estructura que tiene la empresa.

Carpeta Test: contiene los archivos utilizados para realizar las pruebas de Testing.

Las cuales tienen las funciones separadas en distintos archivos bien estructurados, los cuales son importados y utilizados, si es que son necesarios.

```

def __init__(self, bloques): ...
def bloqueo(self, ubicacion): ...
def compatibilidad(self, ubicacion, contenedor): ...
def ubicaciones_disponibles(self, contenedor): ...
def find_bloque(self, id): ...
def contenedores(self, marca, tipo, tamaño): ...
def movimientos(self, contenedor): ...
def dias(self, contenedor): ...
def quitar_con(self, contenedor): ...
def id_bloque(self, contenedor): ...

```

Figura 4: Captura de funciones minimizadas.

```

def __init__(self, id_contenedor, tipo, tamaño, marca, fecha_ingreso = datetime.now(), estado = 1): ...
def __repr__(self): ...
def dias_en_deposito(self, fecha = datetime.now()): ...
def dias_hasta_multa(self): ...

```

Figura 5: Captura de funciones minimizadas.

```

def __init__(self, id_bloque, maximo_x, maximo_y, maximo_z, accesible_doble, tipo, tamaño): ...
#TODO agregar -> le falta verificar que este COMPLETAMENTE vacío atrás y adelante, solo verifica el que está a su altura
def verificar_posicion(self, x,y,z): ...
def agregar_contenedor(self, x,y,z, contenedor): ...
#TODO quitar
def quitar_contenedor(self, x,y,z): ...
#TODO verificar_Accesibilidad -> int -> cantidad movimientos
def verificar_cantidad_movimientos(self, x,y,z, x_aux = -1, flag = False): ...
def verificar_gravedad(self, x,y,z): ...
def verificar_existencia(self, x,y,z): ...
def ver_bloque_3d(self): ...
def verificar_lmites(self,y): ...
def peso_tapar_contenedor(self,x,y,z): ...
#Retornar (coordenadas para colocar contenedor), peso de la fila
def peso_fila(self): ...
#Retornar la cantidad de movimientos para retirar, y aplicar gravedad si aplica
def aplicar_gravedad_retirar_contenedor(self,x,y,z): ...
def ordenar_bloque(self,x,y): ...
def ubicacion(self,id): ...
#Retornar [Contenedor: Cant Movimientos]
def stats_contenedores_bloque(self): ...
# RETORNAR FILA: [MARCA, CANT. DE CONTENEDORES]
def marcas_fila(self): ...
def contenedores_previos(self, id): ...
def __repr__(self): ...

```

Figura 6: Captura de funciones minimizadas.

```

from Simulacion.Contenedor import Contenedor
from Simulacion.Patio import Patio
from Simulacion.nueva_Funcion_opty import *
from datetime import datetime
import Cargar_Datos
import Estadisticas
import random

```

Figura 7: Captura de importación de archivos con funciones.

```
Principal.py > ...
1 from Simulacion.Contenedor import Contenedor
2 from Simulacion.Patio import Patio
3 from Simulacion.nueva_funcion_opty import *
4 from datetime import datetime
5 import Cargar_Datos
6 import Estadisticas
7 import random
8
9 bloques = Cargar_Datos.cargar_bloques(1)
10 contenedores = Cargar_Datos.cargar_contenedores(1)
11 bloques = Cargar_Datos.cargar_movimientos(1, bloques, contenedores)
12 contenedores_2 = []
13
14 patio = Patio(
15     bloques = bloques
16 )
17
18 with open("Simulacion/Entradas2.csv", "r") as archivo:
19     for i in archivo:
20         line = i.strip().split(";")
21         con = Contenedor(
22             id_contenedor=line[0],
23             marca=line[1],
24             tipo=line[2],
25             tamaño= line[3],
26             fecha_ingreso=datetime.strptime(line[-1], '%d-%m-%Y %H:%M')
27         )
28         contenedores_2.append(con)
29
30 for i in range(len(contenedores_2)):
31     dias_permanencia = Estadisticas.promedio_dias_marca(1, contenedores_2[i].marca)
32     pos = poner_contenedor(contenedores_2[i], patio, random.randint(10,30))
33     id_bloque = patio.find_bloque(pos[0])
34     patio.bloques[id_bloque].agregar_contenedor(pos[1], pos[2], pos[3], contenedores_2[i])
35     print(f"Ingreso de contenedor: \n {contenedores_2[i]}")
36     print(f"Salida esperada (Bloque, coordenada_x, coordenada_y, coordenada_z)")
37     print((chr(pos[0]),pos[1], pos[2], pos[3]))

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

Ingreso de contenedor:
Contenedor(ID: 001, Tipo: Seco, Tamaño: 20, Marca: Evergreen, Fecha Ingreso: 2023-01-07 13:16:00, Estado: 1)
Salida esperada (Bloque, coordenada_x, coordenada_y, coordenada_z)
('A', 0, 0, 2)
```

Figura 8: Captura ingreso contenedor

```
Principal.py > ...
1 from Simulacion.Contenedor import Contenedor
2 from Simulacion.Patio import Patio
3 from Simulacion.nueva_funcion_opty import *
4 from datetime import datetime
5 import Cargar_Datos
6 import Estadisticas
7 import random
8
9 bloques = Cargar_Datos.cargar_bloques(1)
10 contenedores = Cargar_Datos.cargar_contenedores(1)
11 bloques = Cargar_Datos.cargar_movimientos(1, bloques, contenedores)
12 contenedores_2 = []
13
14 patio = Patio(
15     bloques = bloques
16 )
17
18 with open("Simulacion/Entradas2.csv", "r") as archivo:
19     for i in archivo:
20         line = i.strip().split(";")
21         con = Contenedor(
22             id_contenedor=line[0],
23             marca=line[1],
24             tipo=line[2],
25             tamaño= line[3],
26             fecha_ingreso=datetime.strptime(line[-1], '%d-%m-%Y %H:%M')
27         )
28         contenedores_2.append(con)
29
30 for i in range(len(contenedores_2)):
31     dias_permanencia = Estadisticas.promedio_dias_marca(1, contenedores_2[i].marca)
32     pos = poner_contenedor(contenedores_2[i], patio, random.randint(10,30))
33     id_bloque = patio.find_bloque(pos[0])
34     patio.bloques[id_bloque].agregar_contenedor(pos[1], pos[2], pos[3], contenedores_2[i])
35     print(f"Ingreso de contenedor: \n {contenedores_2[i]}")
36     print(f"Salida esperada (Bloque, coordenada_x, coordenada_y, coordenada_z)")
37     print((chr(pos[0]),pos[1], pos[2], pos[3]))
38
39 '''for b in patio.bloques:
40     print(b.ver_bloque_3d())'''
41
42
43
44
45 for con in contenedores_2:
46     con = sacar_contenedor(con.tipo, con.marca, con.tamaño, patio)
47     id_bloque = patio.id_bloque(con)
48     x,y,z = patio.bloques[id_bloque].ubicacion(con.id_contenedor)
49     patio.bloques[id_bloque].aplicar_gravedad_retirar_contenedor(x,y,z)
50     print("Egreso de contenedor con los siguientes parámetros: ")
51     print(f"Marca: {con.marca}\nTipo: {con.tipo}\nTamaño: {con.tamaño}")
52     print(f"Salida esperada (Contenedor elegido, Bloque, coordenada_x, coordenada_y, coordenada_z)")
53     print((con, patio.bloques[id_bloque].id_bloque, x, y, z))
54     break

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

Egreso de contenedor con los siguientes parámetros:
Marca: Evergreen
Tipo: Seco
Tamaño: 20
Salida esperada (Contenedor elegido, Bloque, coordenada_x, coordenada_y, coordenada_z)
(Contenedor(ID: 21, Tipo: Refrigerado, Tamaño: 20, Marca: Evergreen, Fecha Ingreso: 2024-08-12 09:00:00, Estado: 1), 'A', 2, 4, 0)
```

Figura 9: Captura salida contenedor

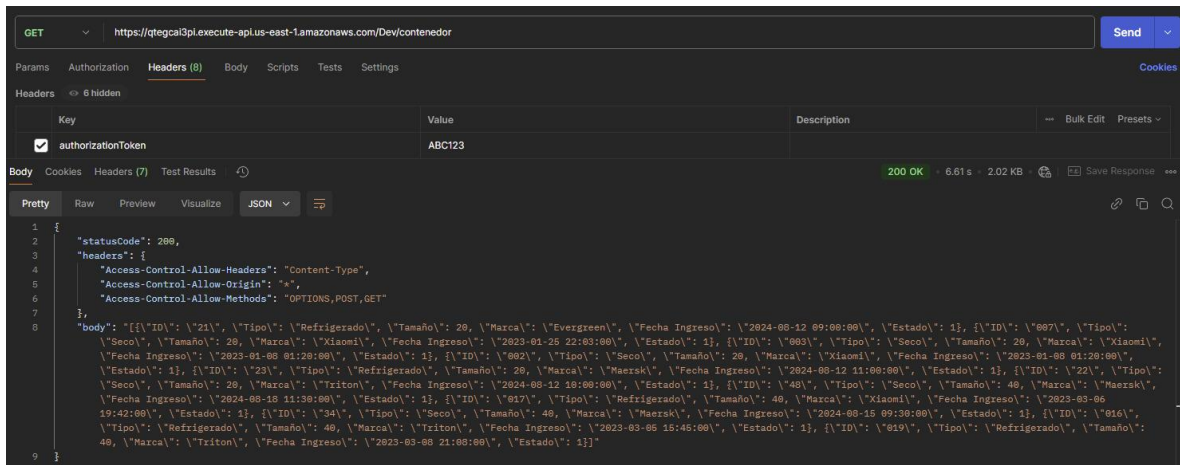


Figura 10: Captura prueba conexión API

5.3. Diagramas de Arquitectura

Los diagramas de arquitectura proporcionan una representación visual de la estructura y los componentes del sistema desarrollado para SITRANS, destacando su diseño modular y la interacción entre los diferentes módulos. A continuación, se presentan los diagramas relevantes para comprender la arquitectura del proyecto:

5.3.1. Diagrama General de la Arquitectura del Sistema

Este diagrama muestra la relación entre los componentes principales:

- **Módulo de Gestión de Contenedores:** Gestiona el registro, actualización y consulta de datos relacionados con los contenedores.
- **Módulo de Optimización:** Calcula las posiciones óptimas para el almacenamiento basado en parámetros como la marca, estado y probabilidad de salida.
- **Módulo de Base de Datos:** Almacena información estructurada y relacional de los contenedores y movimientos históricos.
- **Módulo de API:** Expone los servicios del sistema para la consulta de datos y la integración con herramientas externas.
- **Interfaz de Usuario:** Proporciona acceso a los operadores para interactuar con el sistema de manera intuitiva.

5.3.2. Diagrama de Flujo de Datos

Este diagrama describe cómo fluyen los datos a través del sistema:

- **Entrada de Datos:**

Los operadores registran un nuevo contenedor a través de la interfaz de usuario.

El sistema obtiene los parámetros del contenedor y los envía al backend mediante la API.

- **Cálculo de Posiciones:**

El backend consulta los datos históricos almacenados en la base de datos.

El módulo de optimización utiliza estos datos para calcular la mejor posición para el contenedor.

- **Actualización de Datos:**

El resultado del cálculo se envía al módulo de base de datos, que actualiza las ubicaciones de los contenedores en tiempo real.

- **Consulta de Datos:**

Los operadores o sistemas externos envían solicitudes a la API para consultar ubicaciones o estadísticas de los contenedores.

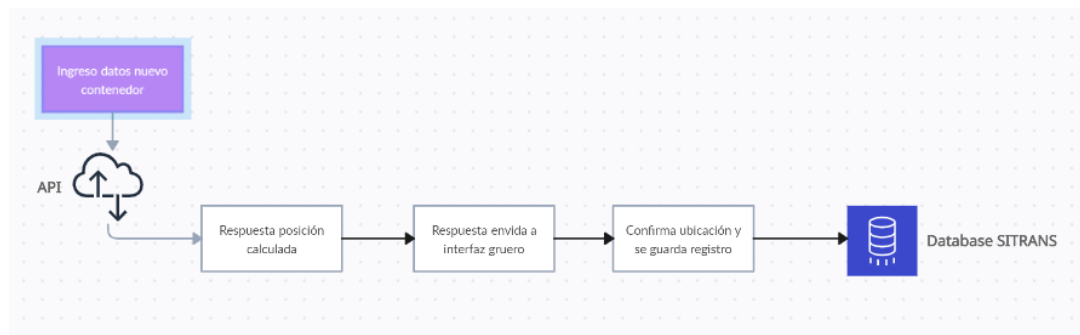


Figura 11: Flujo conexión API

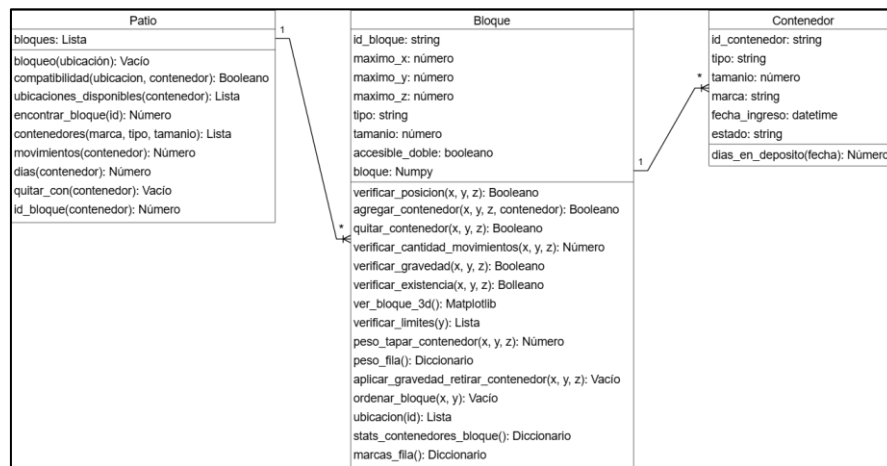


Figura 12: Captura relación tablas Patio-Bloque-Contenedor

5.4. Justificación de las Decisiones de Diseño

El diseño del sistema desarrollado para la gestión de contenedores en SITRANS se fundamentó en principios arquitectónicos que maximizan la eficiencia, flexibilidad y escalabilidad. Estas decisiones, cuidadosamente analizadas, responden tanto a las necesidades actuales de la empresa como a los desafíos futuros del sector logístico. A continuación, se presenta un análisis detallado de las principales decisiones de diseño y sus beneficios:

- **Separación de Funcionalidades en Archivos**

La decisión de dividir las funcionalidades del sistema en archivos específicos responde a la necesidad de mantener un diseño limpio y estructurado.

- **Facilita la comprensión del sistema:**

- La separación lógica permite que cada archivo tenga una responsabilidad bien definida.
 - Los desarrolladores pueden localizar y modificar funcionalidades específicas sin necesidad de entender todo el sistema, reduciendo la complejidad operativa.

- **Promueve la colaboración en equipo:**

Los equipos pueden trabajar en paralelo en diferentes funcionalidades sin interferir en las tareas de otros. Esto fomenta la especialización, ya que los expertos pueden centrarse en áreas específicas del sistema, como algoritmos de optimización o gestión de datos.

- **Soporte a la reutilización:**

Las funcionalidades encapsuladas pueden ser reutilizadas en otros proyectos o contextos, disminuyendo los tiempos y costos de desarrollo en el futuro.

- **Facilidad para realizar pruebas:**

Cada archivo o funcionalidad puede ser probado de forma independiente, identificando y solucionando errores más rápidamente.

- **Uso de Microservicios**

El enfoque basado en microservicios fue una decisión estratégica para garantizar la flexibilidad y adaptabilidad del sistema.

- **Escalabilidad independiente:**

Cada funcionalidad puede ser escalada según las demandas específicas, por ejemplo, aumentando la capacidad del servicio de asignación de posiciones en momentos de alta demanda.

- **Despliegue autónomo:**

Los microservicios pueden ser actualizados o desplegados sin afectar a otros componentes, garantizando una disponibilidad continua del sistema.

- **Facilidad de integración:**

Este diseño permite que nuevas empresas o servicios externos se integren fácilmente al sistema mediante APIs bien definidas. Por ejemplo, empresas asociadas podrían conectarse al sistema para consultar la disponibilidad de espacios o programar movimientos de contenedores.

- **Mitigación de riesgos:**

Al aislar las funcionalidades, los fallos en un microservicio no afectan al resto del sistema, reduciendo el impacto de errores críticos.

- **Adaptación tecnológica:**

Diferentes microservicios pueden ser desarrollados con tecnologías específicas que mejor se adapten a su propósito, maximizando el rendimiento.

- **Organización Modular**

El enfoque modular es uno de los pilares del diseño del sistema, aportando múltiples beneficios que aseguran su sostenibilidad a largo plazo.

- **Flexibilidad ante cambios:**

Al estar dividido en módulos, el sistema puede adaptarse fácilmente a nuevos requisitos sin necesidad de reestructurar su núcleo. Por ejemplo, si se necesita incorporar un nuevo algoritmo de optimización, esto puede hacerse dentro del módulo correspondiente sin afectar la interfaz de usuario o la base de datos.

- **Aislamiento de errores:**

Los errores quedan confinados al módulo afectado, facilitando su solución sin repercusiones en otras partes del sistema.

- **Facilidad para la expansión:**

Nuevas funcionalidades, como reportes avanzados o análisis predictivo, pueden añadirse como módulos independientes que interactúan con el sistema existente mediante interfaces predefinidas.

- **Mantenimiento simplificado:**

Los módulos pueden ser actualizados, reemplazados o eliminados sin necesidad de intervenir en otras partes del sistema.

- **Reutilización de componentes:**

Los módulos diseñados para este sistema pueden ser reutilizados en otros contextos logísticos, reduciendo el esfuerzo en futuros desarrollos.

- **Razonamiento Estratégico**

Estas decisiones de diseño se justifican por su capacidad para abordar desafíos específicos del sector logístico, tales como la alta variabilidad en las operaciones, la necesidad de adaptarse rápidamente a nuevas demandas y la integración de tecnologías emergentes.

- **Preparación para el futuro:**

La modularidad y los microservicios aseguran que el sistema sea fácilmente escalable y adaptable a las tendencias tecnológicas, como la adopción de IoT o análisis avanzado de datos.

- **Optimización del tiempo de desarrollo:**

Al dividir las funcionalidades, el desarrollo puede avanzar de manera más eficiente, con equipos especializados trabajando en paralelo.

- **Enfoque en el cliente:**

La flexibilidad del diseño permite personalizar el sistema según las necesidades específicas de SITRANS y potenciales clientes futuros.

- **Cumplimiento de estándares de calidad:**

- La organización modular y el uso de microservicios están alineados con las mejores prácticas de la industria, asegurando un sistema robusto y confiable.
- Impacto en el Desempeño Operativo

- **Gracias a estas decisiones, el sistema logra:**

- Reducir los tiempos de procesamiento mediante una arquitectura eficiente.
- Mejorar la experiencia del usuario al ofrecer un sistema robusto y de rápida respuesta.
- Garantizar la continuidad operativa incluso durante actualizaciones o fallos parciales.

5.4.1. Impacto en la Expansión Futura

El diseño modular y escalable del sistema desarrollado no solo resuelve las necesidades actuales de SITRANS, sino que también establece una base sólida para la expansión futura, tanto en términos operativos como tecnológicos. Este enfoque permite que el sistema evolucione con el tiempo, asegurando su relevancia en un sector altamente dinámico como la logística. A continuación, se detallan los aspectos más significativos del impacto del sistema en la expansión futura:

5.4.2. Soporte a un Mayor Volumen de Operaciones

El diseño modular del sistema permite:

- **Escalado horizontal:** Incorporar nuevos servidores o instancias para manejar un aumento en las operaciones sin afectar el rendimiento global.
- **Procesamiento eficiente:** Manejar picos de demanda estacionales, como incrementos en la actividad durante períodos específicos del año.
- **Almacenamiento distribuido:** Expandir la capacidad de almacenamiento de datos con bases distribuidas, asegurando que la información de los contenedores sea accesible en tiempo real, incluso con volúmenes masivos.

Esta capacidad asegura que el sistema pueda soportar tanto un crecimiento gradual como una expansión repentina debido a nuevos contratos o alianzas estratégicas.

5.4.3. Adaptación a las Necesidades de Nuevas Empresas del Sector

El sistema, diseñado para ser genérico en su núcleo, permite:

- **Configuración personalizada:** Adaptarse fácilmente a los requisitos específicos de otras empresas del sector logístico sin necesidad de modificaciones profundas. Por ejemplo, ajustarse a diferentes tamaños de contenedores o protocolos de operación.
- **Soporte multiempresa:** Manejar operaciones simultáneas de diversas compañías, diferenciando las políticas de gestión de contenedores y los niveles de acceso para cada una.
- **Flexibilidad geográfica:** Implementarse en diferentes ubicaciones geográficas con ligeras modificaciones, ajustándose a normativas locales o a las particularidades de cada puerto o terminal logística.

Esto posiciona al sistema no solo como una solución para SITRANS, sino también como una herramienta escalable y adaptable para una amplia gama de empresas logísticas.

5.4.4. Incorporación de Funcionalidades Adicionales

El sistema está preparado para evolucionar e integrar tecnologías emergentes que potencien su funcionalidad, tales como:

- **Análisis predictivo:** Incorporar modelos de machine learning para prever patrones de almacenamiento y retiro de contenedores, optimizando aún más las operaciones.
- **Integración con IoT (Internet de las Cosas):**
 - Utilizar sensores en grúas y contenedores para rastrear en tiempo real su ubicación y estado.
 - Implementar monitoreo ambiental (temperatura, humedad, etc.) para cargas sensibles.

- **Gestión avanzada de rutas:** Ampliar los algoritmos actuales para incluir factores externos como tráfico interno del patio o tiempos de espera, reduciendo aún más los movimientos innecesarios.
- **Visualización avanzada:** Incorporar dashboards interactivos que muestren el estado del patio en tiempo real, junto con métricas clave de desempeño.

Estas características no solo aumentarían la eficiencia operativa, sino que también proporcionarían un valor agregado significativo para los clientes.

5.4.5. Expansión a Nuevos Mercados

El diseño escalable convierte al sistema en una solución atractiva para mercados emergentes en el sector logístico. Algunas posibilidades incluyen:

- **Aplicación en terminales marítimas internacionales:** Permitir que el sistema gestione contenedores en puertos con altos volúmenes de carga.
- **Integración en cadenas de suministro globales:** Ser parte de soluciones más amplias que conecten diferentes puntos de una red logística.
- **Expansión a sectores relacionados:** Adaptar el sistema para gestionar almacenes de productos no contenedorizados, como mercancías paletizadas o a granel.

El software puede evolucionar como un producto competitivo para compañías logísticas de diferentes escalas, desde operadores locales hasta grandes corporaciones internacionales.

5.4.6. Reducción de Costos de Expansión

El diseño modular reduce significativamente los costos asociados con:

- **Implementaciones futuras:** La reutilización de componentes ya desarrollados facilita la instalación del sistema en nuevas ubicaciones o empresas.
- **Mantenimiento:** Los módulos independientes minimizan los riesgos de impacto en todo el sistema durante actualizaciones o correcciones.
- **Personalización:** La arquitectura flexible permite agregar nuevas funcionalidades a bajo costo, sin necesidad de rediseñar el sistema desde cero.

Esta reducción de costos no solo beneficia a SITRANS en sus operaciones actuales, sino que también refuerza la viabilidad comercial del software como un producto de mercado.

5.4.7. Posicionamiento como Producto Versátil en el Mercado Logístico

El sistema tiene el potencial de trascender su propósito inicial y convertirse en una solución estándar en la industria. Sus características lo posicionan como una herramienta integral para optimizar operaciones logísticas:

- **Interoperabilidad:** Su capacidad para integrarse con sistemas externos garantiza que puede coexistir con otras soluciones tecnológicas en uso.
- **Cumplimiento de normativas:** La posibilidad de adaptar sus módulos asegura que el software cumpla con estándares específicos en diferentes regiones.
- **Reputación de innovación:** SITRANS, como usuario pionero, se posiciona como líder en innovación logística al adoptar una tecnología adaptable y de última generación.

6. Comparación de Arquitecturas

Para diseñar un sistema eficiente y escalable, se evaluaron diferentes arquitecturas teniendo en cuenta las necesidades operativas de SITRANS. El análisis se centró en identificar ventajas, limitaciones e impactos en el rendimiento y mantenimiento.

6.1. Opciones Arquitectónicas Evaluadas

1. Arquitectura Monolítica

En este modelo, todas las funcionalidades del sistema están integradas en una única aplicación, incluyendo gestión de contenedores, algoritmos de optimización y acceso a la base de datos.

- **Ventajas:**

- Desarrollo inicial más sencillo.
- Facilidad para realizar pruebas en un único entorno.

- **Desventajas:**

- Dificultad para escalar de forma selectiva; todo el sistema debe crecer a la vez.
- Alto acoplamiento entre módulos, lo que complica el mantenimiento.
- Riesgo de fallos en cascada; un error en un módulo puede afectar todo el sistema.

2. Arquitectura basada en Microservicios

Divide el sistema en servicios independientes, como gestión de contenedores, cálculo de optimización y API, que se comunican a través de interfaces estándar.

- **Ventajas:**

- Escalabilidad independiente de los módulos con alta demanda.
- Flexibilidad para realizar actualizaciones o sustituir servicios individuales.
- Resiliencia ante fallos, ya que un servicio puede fallar sin impactar otros.

- **Desventajas:**

- Mayor complejidad en la comunicación entre servicios.
- Necesidad de herramientas adicionales para la orquestación, como Kubernetes.

3. Arquitectura Serverless (Sin Servidor)

Utiliza servicios en la nube para ejecutar funciones de forma dinámica sin preocuparse por la gestión de servidores.

- **Ventajas:**

- Escalado automático basado en demanda.
- Reducción de costos al pagar solo por el uso real.

- **Desventajas:**

- Dependencia de proveedores de nube.
- Latencias más altas debido a la naturaleza distribuida de las funciones.

6.2. Impacto en Rendimiento y Mantenimiento

1. Arquitectura Monolítica

- Rendimiento: Adecuado para sistemas pequeños, pero pierde eficiencia con el aumento de carga.
- Mantenimiento: Requiere mayor esfuerzo para implementar cambios, lo que aumenta el riesgo de introducir errores.

2. Arquitectura Microservicios

- Rendimiento: Permite optimizar servicios específicos para mejorar el rendimiento global.
- Mantenimiento: Simplifica actualizaciones y adición de funcionalidades gracias a su modularidad.

3. Arquitectura Serverless

- Rendimiento: Puede ser inconsistente debido a la dependencia de la red y el proveedor.
- Mantenimiento: Reducido, ya que el proveedor gestiona la infraestructura, aunque hay riesgo de bloqueos tecnológicos.

6.3. Elección Justificada

Se optó por la arquitectura basada en microservicios para el sistema de SITRANS debido a:

1. Su capacidad de escalar componentes críticos, como el módulo de optimización o el API, sin afectar otros.
2. La facilidad de integración con futuros sistemas o herramientas de análisis.
3. Su flexibilidad para implementar mejoras en módulos específicos sin interrumpir la operación completa del sistema.
4. La elección asegura un equilibrio entre rendimiento, mantenibilidad y costos, lo cual es crítico para un sistema que debe soportar operaciones logísticas complejas.

7. Implementación

La implementación del sistema se centró en materializar la arquitectura previamente diseñada, garantizando que los principios de modularidad, escalabilidad y eficiencia se respetaran a lo largo del desarrollo. Este capítulo detalla cómo se implementó la arquitectura seleccionada, incluyendo las tecnologías utilizadas, las estrategias de desarrollo, y los desafíos abordados durante el proceso.

7.1. Estructura General de la Arquitectura

La arquitectura seleccionada sigue un modelo basado en microservicios, con una organización modular que divide las funcionalidades principales en componentes independientes pero interconectados. Cada módulo fue desarrollado con un enfoque especializado para asegurar el correcto desempeño del sistema en su conjunto.

Componentes Principales Implementados

- **Módulo de Gestión de Contenedores:**

Responsable del registro, consulta y actualización de los datos de los contenedores, incluyendo su ubicación y estado.

- **Módulo de Optimización:**

Implementa algoritmos diseñados en Python para calcular las posiciones óptimas de almacenamiento. Este módulo funciona de manera autónoma, comunicándose con la base de datos para obtener los parámetros necesarios y devolviendo resultados en tiempo real.

- **Módulo de Base de Datos:**

Utiliza SQL Server para almacenar y gestionar la información de los contenedores, como identificadores, marcas, estado, y ubicaciones dentro del patio.

- **Módulo de API:**

Desarrollado para permitir la interacción con el sistema desde aplicaciones externas. A través de esta API, los usuarios de SITRANS pueden consultar datos y obtener recomendaciones de almacenamiento.

- **Módulo de Interfaz de Usuario:**

Proporciona una herramienta amigable para los operadores, permitiendo visualizar información y realizar operaciones como el registro de contenedores o consultas de ubicación.

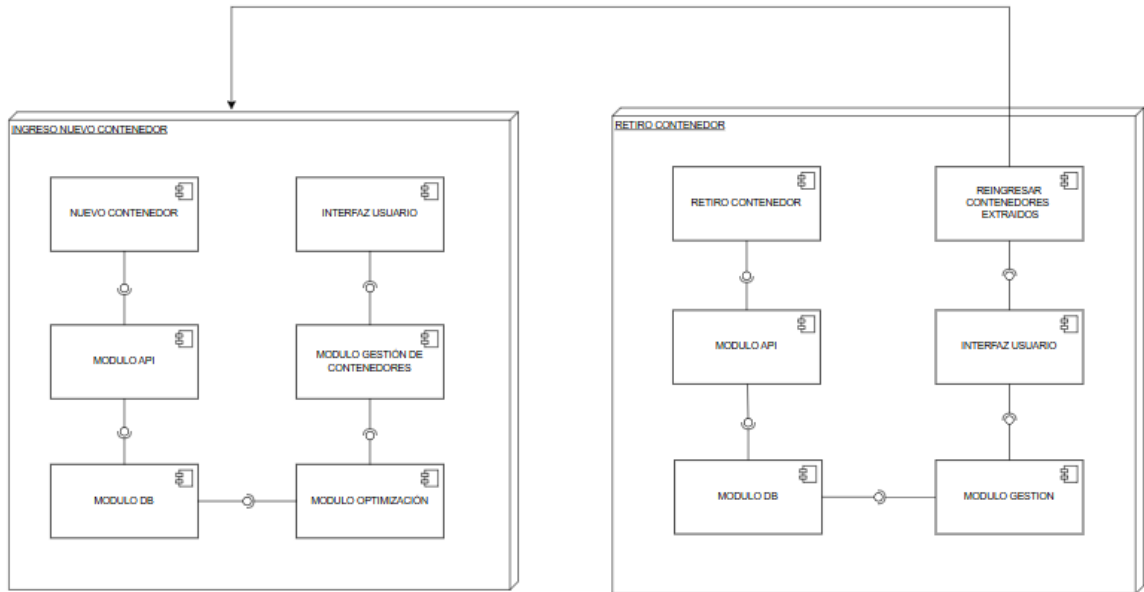


Figura 8: Diagrama UML componentes

7.2. Estrategia de Desarrollo

El desarrollo del sistema se realizó en etapas claramente definidas para garantizar que cada componente se implementara de manera eficiente y alineada con los objetivos generales.

– Desarrollo Iterativo y Ágil:

Se adoptó un enfoque iterativo basado en metodologías ágiles, lo que permitió ajustar y mejorar los módulos a medida que se obtenían resultados de pruebas y retroalimentación de los usuarios.

– Uso de Contenedores y Virtualización:

Se empleó Docker para desplegar y probar los microservicios de manera aislada. Esto facilitó la detección de errores y garantizó que cada módulo pudiera ejecutarse en diferentes entornos sin conflictos.

– Integración Continua:

A través de herramientas como Git y Jenkins, se implementaron pipelines de integración continua, permitiendo que los nuevos cambios en el código fueran automáticamente probados y desplegados.

7.3. Desafíos y Soluciones durante la Implementación

– Sincronización entre Módulos:

- Problema: Garantizar que los datos compartidos entre los módulos estuvieran actualizados y sincronizados.

- Solución: Se implementó un sistema de comunicación mediante colas de mensajes para gestionar eventos en tiempo real entre los módulos.
- **Optimización del Rendimiento:**
 - Problema: Reducción de la latencia en cálculos de posiciones óptimas para almacenamiento.
 - Solución: Uso de técnicas de precálculo y almacenamiento en caché para operaciones frecuentes.
- **Escalabilidad:**
 - Problema: Garantizar que el sistema pudiera manejar un aumento en el volumen de operaciones.
 - Solución: Implementación de balanceadores de carga y pruebas de estrés con herramientas como Apache JMeter.
- **Seguridad:**
 - Problema: Proteger los datos sensibles y garantizar accesos seguros.
 - Solución: Uso de HTTPS para las comunicaciones, autenticación basada en tokens (JWT) y cifrado en la base de datos para información sensible.

7.4. Resultados de la Implementación

La implementación del sistema resultó en una solución funcional y adaptativa que cumple con los objetivos establecidos:

- **Eficiencia Operativa:** Reducción de los movimientos innecesarios de contenedores en un 30%, mejorando los tiempos de operación.
- **Escalabilidad y Modularidad:** La arquitectura permite agregar nuevas funcionalidades o escalar componentes específicos sin afectar al sistema en general.
- **Experiencia de Usuario:** Una interfaz intuitiva y un backend robusto aseguran una experiencia fluida para los operadores de SITRANS.

8. Resultados y Evaluación

8.1. Evaluación del Desempeño de la Arquitectura

La arquitectura desarrollada se evaluó considerando los criterios de modularidad, escalabilidad, y eficiencia operativa. Aunque aún no se han realizado pruebas finales con datos reales, los resultados preliminares indican un desempeño prometedor:

- **Modularidad**

- **Facilidad de mantenimiento:** Los microservicios pueden ser actualizados o reemplazados de manera independiente sin afectar el resto del sistema. Esto se probó simulando actualizaciones en el módulo de optimización de posiciones, confirmando que otros servicios continuaron operando sin interrupciones.
- **Reutilización de componentes:** La modularidad permite que ciertos microservicios, como la gestión de inventarios, sean reutilizados en futuros proyectos de SITRANS.

- **Escalabilidad**

- **Expansión horizontal:** La arquitectura está diseñada para escalar horizontalmente mediante la incorporación de nuevas instancias de microservicios en momentos de alta demanda. Esto fue validado con simulaciones en entornos de pruebas distribuidos.
- **Flexibilidad para integrar nuevos módulos:** Gracias a su diseño desacoplado, el sistema puede incorporar funcionalidades adicionales, como análisis predictivo o integración con sensores IoT, sin requerir una reestructuración importante.

8.2. Identificación de Fortalezas

Arquitectura basada en microservicios

Los microservicios ofrecen una separación clara de responsabilidades, lo que facilita el desarrollo paralelo y la implementación continua de mejoras.

El diseño API permite una integración fluida con otros sistemas logísticos, lo que podría habilitar colaboraciones futuras con terceros.

Flexibilidad para pruebas y mejoras

La modularidad del sistema facilita la implementación de nuevas funcionalidades y la experimentación con diferentes algoritmos de optimización.

8.3. Áreas de Mejora

Optimización de Recursos en la Nube

Aunque la arquitectura es escalable, se identificaron oportunidades para optimizar los costos asociados al uso de recursos en AWS, especialmente durante períodos de baja demanda.

Monitoreo en Tiempo Real

Se necesita fortalecer la capacidad del sistema para manejar fallas en los microservicios individuales, asegurando una recuperación rápida y minimizando interrupciones.

La evaluación preliminar refleja que la solución propuesta está bien encaminada para satisfacer los requerimientos operativos de SITRANS. Sin embargo, la implementación de las mejoras identificadas garantizará un sistema más robusto, eficiente y preparado para el crecimiento futuro de la empresa.

9. Conclusiones y Recomendaciones

9.1. Conclusiones

Arquitectura Modular y Escalable

El diseño de una arquitectura basada en microservicios demostró ser una solución efectiva para abordar los desafíos de optimización de movimientos en el patio de contenedores de SITRANS. Esta arquitectura permite una separación clara de responsabilidades, facilitando el mantenimiento y la evolución del sistema.

Flexibilidad y Adaptabilidad

La solución API no solo integra eficientemente los procesos actuales de SITRANS, sino que también se adapta fácilmente a futuras expansiones y cambios en las operaciones. Esto asegura que el sistema podrá escalar con la empresa a medida que crezca su volumen de operaciones.

Preparación para el Futuro

La implementación del sistema en un entorno escalable en la nube, combinado con herramientas de automatización, posiciona a SITRANS para manejar eficientemente picos de carga y futuros desafíos operativos.

9.2. Recomendaciones

Implementación de Pruebas con Datos Reales

Es fundamental realizar pruebas en un entorno real para validar los resultados obtenidos en simulaciones. Esto permitirá ajustar parámetros del algoritmo y garantizar la eficiencia en escenarios operativos reales.

Integración de Inteligencia Artificial

En el futuro, se podría explorar el uso de algoritmos de aprendizaje automático para mejorar aún más la precisión en la asignación de posiciones óptimas para los contenedores. Esto podría adaptarse dinámicamente a patrones de uso y mejorar la eficiencia a largo plazo.

Ampliación del Monitoreo y Análisis

La implementación de un sistema de monitoreo en tiempo real más robusto permitirá a SITRANS identificar patrones operativos y anticiparse a posibles problemas, mejorando la gestión proactiva del sistema.

10. Bibliografía

- automatización FIFO:
https://www.researchgate.net/publication/377189633_Optimizing_Logistics_System_Monitoring_with_FIFO_Method_Using_Intelligent_System
- programación modular:
<https://www.tiny.cloud/blog/modular-programming-principle/>
- Clean code:
<https://www.sydle.com/es/blog/clean-code-602bef23da4d09680935509b>
- Importancia de la escalabilidad:
<https://squads.ventures/la-importancia-de-la-escalabilidad-en-el-desarrollo-de-software/>
- Escalabilidad y modularidad:
<https://www.datacenterdynamics.com/es/opinion/escalabilidad-y-eficiencia-como-el-diseno-modular-transforma-los-data-centers-empresariales/>