Universidad San Carlos de Guatemala

Introducción a la programación y computación 2

28 de diciembre del 2025

Auxiliar Luis Antonio

Ingeniero Marlon

Proyecto 2

Alison Sofia De Paz Gómez

202400822

Luis Pedro Gonzales Xiloj

202406139

Introducción

Este documento describe el desarrollo e implementación de LogiTrack, un sistema para optimizar las operaciones de distribución de una empresa ficticia que opera múltiples centros en diferentes ubicaciones geográficas.

LogiTrack aborda problemas reales que enfrentan las empresas de logística: la asignación eficiente de mensajeros a paquetes, la gestión de capacidades en centros de distribución, el seguimiento en tiempo real del estado de los envíos, y la priorización inteligente de solicitudes de servicio. A través de una arquitectura cliente-servidor moderna, el sistema proporciona una interfaz intuitiva que permite a los operadores tomar decisiones informadas basadas en datos actualizados y confiables.

Sistema LogiTrack

Estructuras de Datos Fundamentales

El diseño de LogiTrack se fundamenta en conceptos clásicos de estructuras de datos y algoritmos. La implementación hace uso extensivo de estructuras lineales y no lineales para representar las diferentes entidades del dominio logístico.

Las listas enlazadas se utilizan implícitamente a través de las colecciones de Java para mantener relaciones entre entidades. Por ejemplo, cada centro de distribución mantiene una lista de paquetes almacenados y mensajeros asignados, permitiendo acceso rápido y modificaciones dinámicas conforme cambia el estado del sistema.

El sistema implementa una cola de prioridad para la gestión de solicitudes de envío. Esta estructura es fundamental para el correcto funcionamiento del negocio, ya que permite atender primero aquellas solicitudes más urgentes o importantes. La implementación utiliza la interfaz priorityQueue de Java, que internamente emplea un heap que ve las operaciones eficientes de inserción y extracción del elemento de mayor prioridad.

Los grafos dirigidos ponderados representan la red de centros de distribución y las rutas que los conectan. Cada nodo del grafo corresponde a un centro, mientras que las aristas representan rutas disponibles con sus respectivas distancias. Aunque el sistema actual no implementa algoritmos de búsqueda de caminos como Dijkstra, la estructura está preparada para futuras expansiones que requieran optimización de rutas.

Arquitectura Cliente-Servidor

LogiTrack adopta una arquitectura de tres capas que separa claramente las responsabilidades del sistema. En el backend, construido con Spring Boot, encontramos la capa de, la capa de servicios que es lo que contiene toda la lógica del proyecto, y la capa de repositorios que nos apoya con la memoria del mismo. Esta separación permite que el frontend, desarrollado con React, se comunique con el backend únicamente a través de peticiones HTTP estandarizadas, promoviendo el desacoplamiento y facilitando el mantenimiento. El frontend consume la API REST y presenta la información al usuario de manera clara y organizada.

Patrones de Diseño Aplicados

El sistema implementa varios patrones de diseño reconocidos en la industria. El patrón Repository abstrae el acceso a datos, proporcionando una interfaz operaciones CRUD independientemente del mecanismo de persistencia. Esto facilita la migración futura a una base de datos real si el volumen de operaciones lo requiere.

El patrón MVC se refleja en la organización del código, donde los modelos representan las entidades de negocio, los controladores manejan las peticiones HTTP y coordinan las operaciones, y las presentan la información al usuario.

Modelado del Dominio

El dominio del problema se compone de cinco entidades principales que interactúan entre sí para representar el ecosistema logístico completo.

Centro de Distribución:

Representa una instalación física donde se almacenan paquetes y desde donde operan los mensajeros. Cada centro tiene una capacidad máxima que define cuántos paquetes puede almacenar simultáneamente. El sistema valida constantemente que no se exceda esta capacidad, rechazando operaciones que violarían esta restricción.

Paquete:

Constituye la unidad básica de trabajo del sistema. Un paquete tiene información del cliente, peso, centro actual y centro destino. Transita por tres estados durante su ciclo de vida: PENDIENTE que es cuando alguien acaba de entrar, EN_TRANSITO que es cuando ya fue puesto con un mensajero, y ENTREGADO que es cuando ya se llegó con su destino. El sistema implementa validaciones estrictas que impiden transiciones de estado inválidas.

Mensajero:

Representa a los empleados encargados de transportar paquetes. Cada mensajero tiene una capacidad máxima de paquetes que puede llevar simultáneamente y está asignado a un centro específico. El estado del mensajero alterna entre DISPONIBLE y EN_TRANSITO, lo que permite al sistema identificar rápidamente qué recursos están disponibles para nuevas asignaciones.

Ruta:

Define las conexiones entre centros de distribución. Una ruta es unidireccional y tiene asociada una distancia en kilómetros. Esta información es crucial para futuras optimizaciones de planificación de entregas, aunque actualmente el sistema no calcula automáticamente las rutas óptimas.

Solicitud:

Encapsula una petición de envío para un paquete específico. Cada solicitud tiene un nivel de prioridad que determina su orden de procesamiento. El sistema mantiene las solicitudes en una cola de prioridad y las procesa en orden descendente de urgencia.

Flujo de Operaciones Principales

El flujo típico de trabajo comienza con la importación de datos desde un archivo XML. Este archivo contiene la configuración inicial del sistema: centros, rutas, mensajeros disponibles, paquetes pendientes y solicitudes de envío. El procesamiento del XML utiliza las APIs estándar de Java para parseo de documentos, validando la estructura y los datos antes de persistirlos en el sistema.

Una vez cargados los datos, el operador puede procesar solicitudes. El algoritmo de procesamiento extrae la solicitud de mayor prioridad de la cola, verifica que el paquete asociado esté en estado PENDIENTE, busca un mensajero disponible en el mismo centro del paquete, y si existe capacidad disponible, realiza la asignación. Este proceso actualiza automáticamente los estados del paquete y del mensajero.

El sistema permite modificaciones dinámicas del estado de los envíos. Cuando un mensajero completa una entrega, el operador actualiza el estado del paquete a ENTREGADO, lo que dispara una serie de acciones: el paquete se desasocia del mensajero, el mensajero recupera capacidad disponible, y si ya no tiene más paquetes asignados, su estado vuelve a DISPONIBLE.

La exportación de resultados genera un archivo XML con el estado actual completo del sistema: todos los centros con sus paquetes y mensajeros actuales, todas las rutas definidas, y el estado final de cada paquete y solicitud. Este archivo sirve como reporte del trabajo realizado y puede ser procesado por otros sistemas.

Decisiones Técnicas Relevantes

La elección de Spring Boot como framework backend se justifica por su capacidad de proporcionar una infraestructura robusta con mínima configuración. Las anotaciones declarativas

simplifican el desarrollo de controladores REST, mientras que la inyección de dependencias facilita el testing y el mantenimiento.

React fue seleccionado para el frontend por su modelo de componentes reutilizables y su eficiente sistema de actualización del DOM. Cada sección de la interfaz es un componente independiente que gestiona su propio estado y se comunica con el backend según sea necesario.

La persistencia en memoria mediante Maps de Java fue una decisión consciente para simplificar el desarrollo inicial. Aunque no es apropiada para un sistema de producción, permite desarrollo rápido y testing sin las complejidades de configurar y mantener una base de datos. La arquitectura del sistema, sin embargo, facilita la migración futura a persistencia real gracias al patrón Repository.

Capa de Modelo

Las clases del modelo encapsulan tanto los datos como el comportamiento básico de cada entidad. La clase centro, por ejemplo, no solo almacena su capacidad y ubicación, sino que incluye métodos para agregar y remover paquetes, verificando siempre que se respete la capacidad máxima. Esto ejemplifica el principio de encapsulación, donde la lógica relacionada con una entidad está contenida dentro de la misma.

La clasevmensajero implementa validaciones inteligentes como puedeCargarPaquete, que verifica tanto que el mensajero esté disponible como que tenga capacidad restante. Este diseño evita código duplicado en los controladores y centraliza las reglas de negocio donde corresponde.

Capa de Repositorio

Los repositorios proporcionan una abstracción sobre el almacenamiento de datos. Aunque actualmente usan Maps internos, la interfaz pública es idéntica a la que tendría un repositorio que trabaje con JPA o JDBC.

Capa de Servicio

La capa de servicios orquesta operaciones complejas que involucran múltiples entidades.

Solicitudservice contiene la lógica para procesar solicitudes, que requiere interactuar con paquetes, mensajeros y solicitudes simultáneamente.

xmlService el parseo y generación de documentos XML. Utiliza las APIs DOM de Java para navegar la estructura del documento de entrada y construir los objetos del sistema. La lógica incluye manejo robusto de errores, asegurando que datos malformados no corrompan el estado del sistema.

Capa de Controladores

Los controladores exponen la funcionalidad del sistema como endpoints HTTP RESTful. Cada controlador se enfoca en un tipo de entidad, siguiendo el principio de responsabilidad única.

Frontend React

El frontend está organizado en componentes funcionales que utilizan hooks de React para gestionar estado y efectos secundarios. Dashboard es el componente principal que muestra una vista general del sistema: cantidad de centros, paquetes en cada estado, mensajeros disponibles, etc.

El componente ImportarXML permite al usuario cargar el archivo de configuración inicial. Muestra feedback inmediato sobre el éxito o fracaso de la operación, y presenta un desglose detallado de cuántos elementos de cada tipo fueron creados.

Conclusiones

El desarrollo de LogiTrack ha demostrado cómo conceptos fundamentales de estructuras de datos y algoritmos se aplican a problemas reales de negocio. La implementación de una cola de prioridad para gestionar solicitudes, el uso de colecciones para mantener relaciones entre entidades, y el diseño de una arquitectura en capas que separa responsabilidades, son ejemplos concretos de aplicación práctica de la teoría.

El proceso de desarrollo reforzó la importancia del diseño antes de la implementación. Definir claramente las entidades, sus relaciones y responsabilidades antes de escribir código resultó en una implementación más limpia y mantenible. La experiencia también destacó la relevancia del manejo apropiado de errores y validaciones: un sistema que falla de manera predecible y comprensible es infinitamente más útil que uno que colapsa silenciosamente ante situaciones complicadas

Diagramas

Diagrama de actividades

g

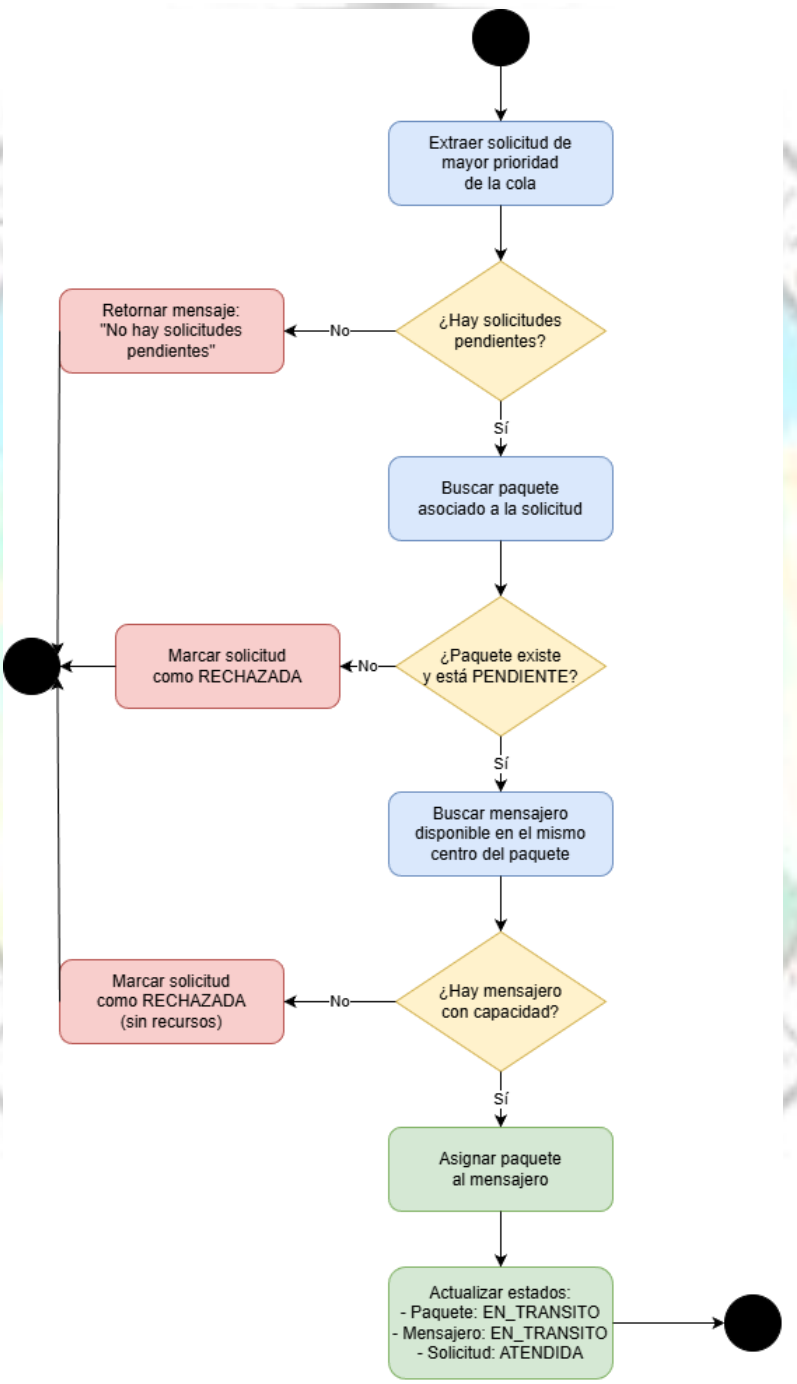

Diagrama de clases