Enunciado del Trabajo: Desarrollo de una API RESTful para Optimización de Redes Logísticas con Grafos

Este documento detalla un proyecto práctico para el desarrollo de una API RESTful centrada en la aplicación de la teoría de grafos para la optimización de redes logísticas. El objetivo es proporcionar una base sólida para un microservicio Spring Boot que demuestre habilidades avanzadas en desarrollo backend, modelado de datos y algoritmos complejos, siendo una pieza clave para un currículum vitae.

Introducción

En el panorama actual del desarrollo de software, las APIs RESTful son el pilar de la comunicación entre sistemas distribuidos, facilitando la integración y la escalabilidad de aplicaciones modernas.[1] Paralelamente, la teoría de grafos emerge como una herramienta matemática indispensable para modelar y resolver problemas complejos en una multitud de dominios, desde redes sociales hasta sistemas de transporte y logística.[2] La capacidad de esquematizar y analizar relaciones complejas entre entidades hace que los grafos sean ideales para optimizar procesos que involucran interconexiones, como la planificación de rutas y la gestión de cadenas de suministro.[4]

Este proyecto desafía la fusión de estas dos áreas, proponiendo el desarrollo de un microservicio backend con Spring Boot que implemente algoritmos de grafos para abordar desafíos reales en la optimización logística. La elección del dominio de la logística y el transporte para la aplicación de la teoría de grafos no es arbitraria; este sector ofrece problemas tangibles donde los conceptos abstractos de grafos encuentran una aplicación directa y cuantificable. Por ejemplo, la aplicación de algoritmos de caminos mínimos en redes de transporte ha demostrado reducciones significativas, del 8% en tiempos de entrega y del 5% en costos totales.[5] Esta capacidad de generar un impacto empresarial medible a través de soluciones técnicas es un diferenciador valioso en un currículum vitae. Al completar este trabajo, se consolidan conocimientos en Spring Boot y bases de datos relacionales como

PostgreSQL, y se demuestra una profunda comprensión de la aplicación práctica de algoritmos de grafos, una habilidad altamente valorada en el mercado laboral.

Objetivos del Proyecto

El desarrollo de esta API RESTful se centrará en los siguientes objetivos clave, diseñados para maximizar el aprendizaje y la demostración de habilidades:

- **Diseñar e implementar una API RESTful robusta:** Se espera desarrollar una API que adhiera a los principios REST, permitiendo la gestión y optimización de rutas dentro de una red de transporte simulada. Esto incluye la implementación de operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para las entidades fundamentales del grafo, asegurando una interacción eficiente y estandarizada con los recursos.[1]
- **Aplicar algoritmos de grafos fundamentales:** Un objetivo central es la integración y exposición, a través de la API, de algoritmos clave de la teoría de grafos. Esto permitirá resolver problemas de caminos óptimos, analizar la conectividad de la red y realizar otras operaciones analíticas complejas.
- **Utilizar Spring Boot como framework backend:** La construcción del microservicio se realizará aprovechando las capacidades de Spring Boot. Este framework facilita la creación de aplicaciones Java robustas, modulares y fáciles de mantener, lo que refleja el uso de tecnologías modernas y estándares de la industria.
- **Persistir datos de grafos en PostgreSQL:** Se requiere el diseño de un esquema de base de datos relacional eficiente para almacenar la estructura del grafo (nodos y aristas) en PostgreSQL. El uso de Neon para la configuración en la nube permitirá familiarizarse con entornos de bases de datos modernas y escalables, una habilidad relevante en arquitecturas distribuidas.
- **Implementar pruebas unitarias y de integración exhaustivas:** Para garantizar la funcionalidad y robustez tanto de la API como de los algoritmos implementados, se deberá desarrollar un conjunto completo de pruebas automatizadas.[1] Esto subraya la importancia de la calidad del código y la fiabilidad del sistema.
- **Producir documentación clara y completa:** La documentación es un componente crítico del proyecto. Se generará una especificación exhaustiva de la API utilizando OpenAPI (anteriormente Swagger) y una interfaz de usuario

(Swagger UI) para facilitar la exploración de los endpoints. Adicionalmente, un archivo README.md detallado con instrucciones de configuración y ejecución será fundamental para la reproducibilidad y comprensión del proyecto por parte de otros desarrolladores.[1]

Requisitos Técnicos

Para la implementación de este proyecto, se establecen los siguientes requisitos técnicos, seleccionados para reflejar las prácticas actuales de la industria y maximizar el valor demostrativo del proyecto:

- **Lenguaje de Programación y Framework:** Se utilizará Java 17+ como lenguaje de programación y Spring Boot 3.x como framework principal. Se recomienda encarecidamente el uso de Spring Data JPA para una interacción eficiente y orientada a objetos con la base de datos, y Spring Web para la creación de los endpoints RESTful. Esta combinación es un estándar en el desarrollo backend empresarial.
- **Base de Datos:** La base de datos elegida será PostgreSQL. La sugerencia de utilizar Neon para la configuración en la nube introduce al desarrollador a entornos de bases de datos modernas y escalables, lo que es cada vez más relevante en la infraestructura de software actual.
- **Representación de Grafos:** El grafo, compuesto por nodos y aristas, se modelará y almacenará utilizando un esquema relacional en PostgreSQL. Este enfoque, a diferencia de utilizar una base de datos de grafos nativa como Neo4j [3], presenta un desafío adicional y una oportunidad para demostrar un dominio avanzado del modelado de datos relacionales. Al no depender de funcionalidades nativas de grafos en la base de datos, la implementación de los algoritmos de grafos deberá realizarse directamente en el código Java, interactuando con las tablas relacionales. Esto pone de manifiesto una comprensión más profunda tanto del modelado de datos como de la lógica algorítmica.
- **Herramientas de Testing:** Para la implementación de pruebas unitarias y de integración, se utilizará JUnit 5 y Mockito. Estas herramientas son ampliamente adoptadas en el ecosistema Java y permiten garantizar la funcionalidad y robustez de la API y los algoritmos.
- **Documentación de API:** La especificación de la API se realizará utilizando OpenAPI (anteriormente Swagger), lo que permitirá generar automáticamente

una interfaz de usuario (Swagger UI) para explorar y probar los endpoints. Además, se requerirá un archivo README.md detallado. Este archivo debe incluir instrucciones claras sobre cómo instalar dependencias, configurar la base de datos y ejecutar la API, así como una descripción de los endpoints y ejemplos de uso.[1] La capacidad de documentar de manera efectiva es fundamental para la colaboración y el mantenimiento de proyectos de software.

Detalles del Proyecto

Este apartado desglosa las funcionalidades y componentes clave que deberán ser desarrollados en el microservicio.

1. Modelo de Datos de Grafos

El primer paso crucial es definir las entidades que representarán la red logística subyacente. Un grafo se compone fundamentalmente de vértices (o nodos) y aristas (o bordes) que los conectan.[7] La elección de PostgreSQL como base de datos relacional implica un diseño cuidadoso del esquema para representar estas estructuras de manera eficiente.

- **Entidades Principales:**
  - **Nodo (Node):** Esta entidad representará una ubicación física o lógica dentro de la red logística. Ejemplos prácticos incluyen ciudades, almacenes, centros de distribución, puntos de entrega o recolección, o incluso intersecciones viales.
    - **Atributos Sugeridos:** Se recomienda incluir un id (preferiblemente UUID o Long para escalabilidad), un nombre (String) para identificación, latitud (Double) y longitud (Double) para la ubicación geoespacial, y un tipo (String) para categorizar el nodo (ej., "Almacén", "Punto de Entrega", "Intersección").
  - **Arista (Edge):** Esta entidad modelará una conexión o ruta entre dos nodos. Las aristas pueden ser dirigidas (unidireccionales, como una calle de sentido único) o no dirigidas (bidireccionales, como una carretera de doble sentido), y son fundamentales para asociar "pesos" que representen el costo de transitar

esa conexión.[5]

- **Atributos Sugeridos:** Un id (UUID o Long), origen_id (clave foránea que referencia el id de un Nodo), destino_id (clave foránea que referencia el id de otro Nodo), distancia_km (Double) para la longitud de la ruta, tiempo_min (Double) para el tiempo de viaje estimado, y costo_eur (Double) para el coste monetario. Un atributo dirigida (Boolean) indicará si la arista es unidireccional. La posibilidad de que los costos sean negativos, aunque inusual para distancias físicas, es relevante si se considera la implementación de algoritmos como Bellman-Ford, que pueden manejar "beneficios" o subsidios en ciertas rutas.[8]

- **Diseño del Esquema de Base de Datos en PostgreSQL:**
  - La creación de tablas en PostgreSQL debe reflejar estas entidades, asegurando la integridad referencial mediante claves foráneas y la eficiencia en las consultas. El diseño de este esquema es una demostración clave de la habilidad para mapear estructuras de datos complejas a un modelo relacional.

La siguiente tabla ilustra un esquema de tablas sugerido para la base de datos PostgreSQL, que permite la representación eficiente de nodos y aristas en una red logística.

Tabla: Esquema de Tablas para Nodos y Aristas en PostgreSQL

Tabla	Campo	Tipo de Dato	Restricciones/Notas
nodos	id	UUID	PRIMARY KEY, DEFAULT gen_random_uuid()
	nombre	VARCHAR(255)	NOT NULL, UNIQUE
	latitud	NUMERIC(10,7)	NOT NULL
	longitud	NUMERIC(10,7)	NOT NULL
	tipo	VARCHAR(50)	NOT NULL, Ej: 'Almacén', 'Punto de Entrega', 'Intersección'
aristas	id	UUID	PRIMARY KEY, DEFAULT gen_random_uuid()

	origen_id	UUID	NOT NULL, FOREIGN KEY REFERENCES nodos(id)
	destino_id	UUID	NOT NULL, FOREIGN KEY REFERENCES nodos(id)
	distancia_km	NUMERIC(10,2)	NOT NULL, CHECK (distancia_km > 0)
	tiempo_min	NUMERIC(10,2)	NOT NULL, CHECK (tiempo_min > 0)
	costo_eur	NUMERIC(10,2)	NOT NULL, CHECK (costo_eur >= 0) (o puede ser negativo si se implementa Bellman-Ford)
	dirigida	BOOLEAN	NOT NULL, DEFAULT TRUE (si es unidireccional)
	descripcion	TEXT	NULLABLE, para detalles adicionales de la ruta

2. Funcionalidades de la API

La API deberá exponer una serie de endpoints RESTful que permitan interactuar con el grafo subyacente y ejecutar los algoritmos de manera controlada.

- **Gestión de Nodos y Aristas (CRUD):**
  - Se implementarán los endpoints RESTful estándar (GET, POST, PUT, DELETE) para las entidades Nodo y Arista. Esto incluye operaciones como la creación de nuevos nodos en la red (POST /api/nodos), la recuperación de detalles de un nodo específico (GET /api/nodos/{id}), la actualización de aristas existentes (PUT /api/aristas/{id}) o la eliminación de nodos y sus aristas asociadas (DELETE /api/nodos/{id}).
  - Es fundamental asegurar que las respuestas HTTP sean claras y

estructuradas, preferentemente en formato JSON, y que se manejen los errores de manera adecuada. Por ejemplo, se debe retornar un código 404 Not Found si un recurso solicitado no existe.[1]

- **Cálculo de Ruta Óptima:**
  - **Endpoint:** GET /api/rutas/optima?origen={idOrigen}&destino={idDestino}&criterio={criterio}
  - **Funcionalidad:** Este endpoint permitirá calcular la ruta más "óptima" entre un nodo de origen y un nodo de destino, basándose en un criterio específico proporcionado por el usuario (ej., distancia_km, tiempo_min, costo_eur). La definición de "óptima" puede variar; por ejemplo, una ruta con el menor número de paradas (no ponderada) o la ruta con la menor distancia/tiempo/costo (ponderada).[10]
  - **Algoritmos a Implementar:**
    - **Algoritmo de Dijkstra:** Este es el algoritmo fundamental para encontrar la ruta más corta en grafos con pesos de aristas no negativos.[11] Es la base para la mayoría de los problemas de rutas logísticas donde los costos (distancia, tiempo) son inherentemente positivos. Su implementación demuestra una comprensión sólida de la optimización de caminos.
    - **Algoritmo de Bellman-Ford (Opcional/Avanzado):** Para un nivel de sofisticación adicional, se puede implementar Bellman-Ford. Este algoritmo es capaz de encontrar la ruta más corta incluso en grafos que contienen pesos de aristas negativos y, crucialmente, puede detectar ciclos negativos.[8] Aunque las distancias físicas no son negativas, los "costos" en una red logística podrían serlo (ej., un tramo subsidiado que "genera" un beneficio). La detección de ciclos negativos es vital para identificar anomalías o rutas que podrían generar un "beneficio infinito" si no se gestionan adecuadamente, lo que demuestra una capacidad avanzada de manejo de escenarios complejos.
- **Exploración de Conectividad y Análisis de Red (Opcional/Avanzado):**
  - La inclusión de estas funcionalidades demuestra una comprensión más profunda de la teoría de grafos y sus aplicaciones analíticas en la resiliencia y eficiencia de las redes logísticas.
  - **Recorrido en Amplitud (BFS):**
    - **Endpoint:** GET /api/grafos/alcanzables/bfs?origen={idOrigen}
    - **Funcionalidad:** Este recorrido explora todos los nodos alcanzables desde un nodo de origen, expandiéndose nivel por nivel. Es particularmente útil para encontrar la ruta más corta en términos de número de "saltos" o "paradas" en grafos no ponderados.[12] Su aplicación en la API permite identificar la conectividad mínima entre puntos.

- **Recorrido en Profundidad (DFS):**
  - **Endpoint:** GET /api/grafos/alcanzables/dfs?origen={idOrigen}
  - **Funcionalidad:** A diferencia del BFS, el DFS explora lo más profundo posible a lo largo de cada rama antes de retroceder.[12] Es útil para verificar la conectividad general del grafo, detectar ciclos o como base para otros algoritmos de grafos más complejos, como el ordenamiento topológico o la identificación de componentes conexas.
- **Ordenamiento Topológico:**
  - **Endpoint:** GET /api/grafos/orden_topologico
  - **Funcionalidad:** Para grafos dirigidos acíclicos (DAGs), este algoritmo devuelve un orden lineal de los vértices tal que si existe una arista de un vértice U a un vértice V, U siempre aparece antes que V en la secuencia.[13] En logística, esto es directamente aplicable para secuenciar tareas con dependencias, como los pasos de ensamblaje de un producto, el orden de carga y descarga de camiones, o la planificación de etapas en una cadena de suministro.
- **Puntos de Articulación y Componentes Fuertemente Conexas (SCCs):**
  - **Endpoint:** GET /api/grafos/puntos_articulacion (para grafos no dirigidos)
  - **Endpoint:** GET /api/grafos/sccs (para grafos dirigidos)
  - **Funcionalidad:** Estas funcionalidades son cruciales para el análisis de la resiliencia de la red. Los **Puntos de Articulación** identifican nodos críticos cuya eliminación (o fallo) desconectaría partes del grafo.[4] En un contexto logístico, estos representan "puntos únicos de fallo" o "cuellos de botella", como un cruce vial vital o un almacén central, cuya interrupción podría paralizar las operaciones. Las **Componentes Fuertemente Conexas (SCCs)**, en grafos dirigidos, identifican subgrafos donde cada vértice es alcanzable desde cualquier otro vértice dentro del mismo subgrafo.[17] Esto es útil para optimizar el flujo de red, identificar clústeres de operaciones interdependientes o zonas de tráfico fluido dentro de una red de transporte.
  - **Algoritmo:** Se recomienda el algoritmo de Tarjan por su eficiencia (O(V+E)) y su capacidad para encontrar tanto puntos de articulación como SCCs en una sola pasada de DFS.[4]
- **Árbol de Expansión Mínima (MST):**
  - **Endpoint:** GET /api/grafos/mst
  - **Funcionalidad:** Este algoritmo calcula el subgrafo que conecta todos los nodos con el mínimo peso total de aristas.[21] Es aplicable en el diseño de redes logísticas para determinar la infraestructura más costo-efectiva para conectar todos los centros de operación o puntos de

recolección/entrega. Un ejemplo práctico es el diseño de una nueva red de fibra óptica para la comunicación entre almacenes, donde se busca minimizar el costo total de cableado manteniendo la conectividad de todos los puntos.[21] Es importante notar que el MST se aplica típicamente a grafos no dirigidos.[21]

- **Algoritmos:** Se puede implementar el algoritmo de Prim o Kruskal.[21]

La siguiente tabla proporciona una visión general de los endpoints RESTful esperados en la API, junto con una breve descripción de su funcionalidad.

Tabla: Endpoints de la API y su Descripción

URL	Método HTTP	Descripción
/api/nodos	POST	Crea un nuevo nodo en la red.
/api/nodos/{id}	GET	Recupera los detalles de un nodo específico.
/api/nodos/{id}	PUT	Actualiza los detalles de un nodo existente.
/api/nodos/{id}	DELETE	Elimina un nodo y todas las aristas asociadas.
/api/aristas	POST	Crea una nueva arista entre dos nodos.
/api/aristas/{id}	GET	Recupera los detalles de una arista específica.
/api/aristas/{id}	PUT	Actualiza los detalles de una arista existente.
/api/aristas/{id}	DELETE	Elimina una arista.
/api/rutas/optima	GET	Calcula la ruta más óptima entre un origen y un destino, según un criterio (distancia, tiempo, costo). Parámetros: origen, destino, criterio.
/api/grafos/alcanzables/bfs (Opcional)	GET	Realiza un recorrido BFS desde un nodo de origen y devuelve los nodos

		alcanzables. Parámetro: origen.
/api/grafos/alcanzables/dfs (Opcional)	GET	Realiza un recorrido DFS desde un nodo de origen y devuelve los nodos alcanzables. Parámetro: origen.
/api/grafos/orden_topologico (Opcional)	GET	Calcula un ordenamiento topológico del grafo (si es un DAG).
/api/grafos/puntos_articulacion (Opcional)	GET	Identifica los puntos de articulación en el grafo.
/api/grafos/sccs (Opcional)	GET	Identifica las componentes fuertemente conexas en el grafo dirigido.
/api/grafos/mst (Opcional)	GET	Calcula el Árbol de Expansión Mínima del grafo.

La siguiente tabla detalla los algoritmos de grafos principales que se deben considerar para la implementación, junto con su propósito en el contexto de la optimización logística.

Tabla: Algoritmos de Grafos a Implementar y su Propósito

Algoritmo	Propósito Principal
Dijkstra	Encontrar la ruta más corta entre dos nodos en grafos con pesos de aristas no negativos (ej., minimización de distancia o tiempo de entrega).[11]
Bellman-Ford (Opcional/Avanzado)	Encontrar la ruta más corta en grafos que pueden contener pesos de aristas negativos y detectar ciclos negativos (ej., análisis de rutas con beneficios o anomalías de costos).[8]
BFS (Recorrido en Amplitud)	Explorar nodos nivel por nivel, útil para encontrar la ruta con el menor número de "saltos" en grafos no ponderados o para

	verificar conectividad.[12]
DFS (Recorrido en Profundidad)	Explorar ramas en profundidad, útil para verificar conectividad, detectar ciclos o como base para otros algoritmos (ej., ordenamiento topológico, SCCs).[12]
Ordenamiento Topológico	Secuenciar tareas con dependencias en un grafo dirigido acíclico (DAG), asegurando que las precedencias se respeten (ej., planificación de procesos de carga o ensamblaje).[13]
Puntos de Articulación (Tarjan)	Identificar nodos críticos cuya eliminación desconectaría partes del grafo, revelando puntos únicos de fallo o cuellos de botella en la red logística.[4]
Componentes Fuertemente Conexas (Tarjan o Kosaraju)	Identificar subgrafos donde cada vértice es alcanzable desde cualquier otro dentro del mismo subgrafo en un grafo dirigido, útil para optimización de flujo o identificación de clústeres operativos.[17]
Árbol de Expansión Mínima (Prim o Kruskal)	Diseñar la infraestructura de red más costo-efectiva para conectar todos los nodos, minimizando el costo total de las conexiones (ej., diseño de una nueva red de comunicación entre almacenes).[21]

3. Controladores y Lógica de Negocio

La arquitectura del microservicio deberá seguir un patrón que se alinee con los principios de diseño de software modular y mantenible. Aunque el documento de referencia menciona el patrón MVC [1], en el contexto de una API RESTful backend con Spring Boot, esto se traduce comúnmente en una separación de responsabilidades clara entre controladores, servicios y repositorios.

- **Controladores (Controllers):** Serán responsables de procesar las solicitudes HTTP entrantes, validar los datos de entrada y delegar la lógica de negocio a la capa de servicios. Deberán asegurar el envío de respuestas HTTP claras y estructuradas, preferentemente en formato JSON, y gestionar adecuadamente

los errores, devolviendo códigos de estado HTTP apropiados (ej., 400 Bad Request para validación fallida, 404 Not Found para recursos inexistentes, 500 Internal Server Error para fallos inesperados).[1]
- **Servicios (Services):** Contendrán la lógica de negocio principal de la aplicación. Aquí residirá la implementación de los algoritmos de grafos y la coordinación de las operaciones con la base de datos. Esta capa abstrae la complejidad de la lógica de negocio de los controladores, facilitando la reutilización y el testing.
- **Repositorios (Repositories):** Serán la interfaz para la interacción con la base de datos PostgreSQL. Utilizando Spring Data JPA, se simplificará la persistencia y recuperación de las entidades Nodo y Arista, traduciendo las operaciones del servicio a consultas de base de datos.

La implementación de esta estructura de capas no solo facilita la organización del código, sino que también promueve la escalabilidad y la facilidad de mantenimiento, aspectos críticos para un microservicio de producción.

4. Pruebas

La calidad del software es primordial, y un proyecto destinado a un currículum vitae debe demostrar un compromiso con las buenas prácticas de desarrollo. Por ello, la implementación de pruebas automatizadas es un requisito fundamental.[1]

- **Pruebas Unitarias:** Se deben escribir pruebas unitarias para cada componente de la lógica de negocio, especialmente para la implementación de los algoritmos de grafos. Esto implica probar funciones individuales o métodos de forma aislada, utilizando Mockito para simular dependencias y asegurar que cada unidad de código funcione como se espera.
- **Pruebas de Integración:** Adicionalmente, se deben implementar pruebas de integración para verificar que los diferentes componentes del sistema (controladores, servicios, repositorios y la base de datos) interactúan correctamente entre sí. Estas pruebas validarán los flujos de trabajo completos de la API, asegurando que los endpoints responden correctamente y que los datos se persisten y recuperan de forma adecuada. Se recomienda cubrir casos de éxito (ej., crear un recurso y recibir un código 201 Created) y casos de error (ej., intentar actualizar un recurso inexistente y recibir un 404 Not Found).[1]
- **Cobertura de Pruebas:** Se buscará una cobertura de pruebas significativa para asegurar que la mayor parte del código esté validada.

5. Documentación

Una API bien documentada es tan importante como una API funcional. La documentación facilita la comprensión, el uso y el mantenimiento del proyecto por parte de otros desarrolladores.[1]

- **Especificación OpenAPI/Swagger:** Se utilizará OpenAPI para definir la especificación de la API. Esto incluye la descripción de cada endpoint (URL, método HTTP), los parámetros requeridos (en la URL, cuerpo o consulta), y ejemplos de solicitudes y respuestas.[1] La generación de Swagger UI permitirá una exploración interactiva de la API.
- **Archivo README.md:** Este archivo será el punto de entrada principal para cualquier persona que desee entender, configurar o ejecutar el proyecto. Deberá incluir:
  - Una descripción concisa del proyecto y sus objetivos.
  - Instrucciones detalladas sobre cómo instalar las dependencias (ej., con Maven o Gradle).
  - Pasos para configurar la base de datos PostgreSQL (incluyendo la conexión a Neon).
  - Instrucciones claras sobre cómo ejecutar la API.
  - Una sección sobre cómo ejecutar las pruebas y dónde encontrar el informe de resultados.
  - Una visión general de los endpoints de la API, posiblemente con ejemplos de uso.

Consideraciones Adicionales

Para elevar la calidad del proyecto y demostrar una comprensión integral del desarrollo de software, se deben tener en cuenta las siguientes consideraciones:

- **Mejores Prácticas de Codificación:** Escribir código modular, separando modelos, controladores y servicios en archivos distintos. Utilizar una convención de nombres consistente (ej., camelCase para variables y funciones) y definir constantes para valores fijos (como códigos de estado HTTP o mensajes de

error).[1] Un código limpio y organizado es un reflejo de profesionalismo.
- **Seguridad Básica:** Implementar validación de entradas de usuario para prevenir errores y posibles ataques de seguridad (ej., asegurar que los campos requeridos estén presentes y que los tipos de datos sean correctos).[1] Aunque este no es un proyecto de seguridad exhaustivo, demostrar conciencia en este aspecto es valioso.
- **Escalabilidad:** Aunque el proyecto es un microservicio único, se recomienda considerar cómo la API podría expandirse en el futuro.[1] Por ejemplo, pensar en cómo se podría integrar la autenticación JWT, cómo se manejarían volúmenes de datos mucho mayores o cómo se añadirían nuevos algoritmos de grafos. Esta mentalidad de "futuro" es clave para arquitectos de software.
- **Depuración (Debugging):** Utilizar herramientas de depuración (como un depurador de IDE o logs detallados) para identificar y resolver problemas durante el desarrollo.[1] La capacidad de depurar eficientemente es una habilidad esencial para cualquier desarrollador.

Entregables

Para la evaluación y presentación del proyecto, se espera la entrega de los siguientes elementos:

1. **Código Fuente:** Todos los archivos de la API, organizados en una estructura de proyecto clara y lógica. Esto incluye las entidades, repositorios, servicios, controladores, configuraciones y clases de algoritmos.
2. **Documentación del Proyecto:** Un archivo README.md completo que siga las directrices mencionadas en la sección de documentación.
3. **Instrucciones de Ejecución:** Un conjunto de instrucciones claras y concisas que permitan a cualquier persona instalar las dependencias, configurar la base de datos y ejecutar la API sin problemas.[1]
4. **Informe de Pruebas:** Evidencia de las pruebas realizadas, incluyendo los resultados de las pruebas unitarias y de integración. Esto puede ser un informe generado por el framework de pruebas o un resumen claro de la cobertura y los resultados.[1]

Criterios de Evaluación

El proyecto será evaluado en función de los siguientes criterios:

- **Funcionalidad:** La API debe realizar correctamente todas las operaciones CRUD y ejecutar los algoritmos de grafos implementados de manera precisa y eficiente.[1]
- **Calidad del Código:** El código debe ser limpio, organizado, modular y seguir las mejores prácticas de programación Java y Spring Boot.[1] Se valorará la legibilidad, la eficiencia y la robustez.
- **Pruebas:** Las pruebas deben cubrir los casos principales (éxito y error) y demostrar la fiabilidad de la API y los algoritmos.[1] La automatización y la cobertura serán aspectos clave.
- **Documentación:** La documentación debe ser clara, completa y suficiente para que otro desarrollador pueda entender y utilizar la API sin dificultades.[1]

Conclusiones y Recomendaciones

El desarrollo de esta API RESTful para la optimización de redes logísticas con grafos representa una oportunidad excepcional para un estudiante de ingeniería informática especializado en backend. Este proyecto no solo permite la aplicación práctica de conocimientos teóricos de estructuras de datos y algoritmos, sino que también aborda problemas del mundo real con un impacto empresarial tangible, como la reducción de costos y tiempos en la logística.[5]

La decisión de implementar algoritmos de grafos sobre una base de datos relacional como PostgreSQL, en lugar de una base de datos de grafos nativa, eleva el nivel de desafío técnico y demuestra una capacidad superior en el modelado de datos y la implementación algorítmica desde cero. La inclusión de algoritmos avanzados como Bellman-Ford (para pesos negativos), ordenamiento topológico (para secuenciación de tareas), puntos de articulación y componentes fuertemente conexas (para análisis de resiliencia y cuellos de botella), y el Árbol de Expansión Mínima (para diseño de red) amplía significativamente el espectro de habilidades demostradas.

Se recomienda abordar este proyecto de manera incremental, comenzando con el modelado de datos y las operaciones CRUD básicas, para luego integrar

progresivamente los algoritmos de grafos. La inversión en pruebas automatizadas y documentación de alta calidad garantizará que el proyecto no solo sea funcional, sino también mantenible y fácil de presentar. Al completar este trabajo, el estudiante poseerá una pieza de portafolio que no solo valida sus habilidades técnicas en Spring Boot y PostgreSQL, sino que también lo posiciona como un profesional capaz de aplicar soluciones informáticas complejas a desafíos empresariales concretos.

Obras citadas

1. Assignment.pdf
2. Aplicación de la Teoría de Grafos para mejorar la planificación de rutas de trabajo de una empresa del sector de la distribución automática = An application of Graph Theory to improve the planning of work routes for a company in the vending sector - ResearchGate, fecha de acceso: julio 12, 2025, https://www.researchgate.net/publication/23649034_Aplicacion_de_la_Teoria_de_Grafos_para_mejorar_la_planificacion_de_rutas_de_trabajo_de_una_empresa_del_sector_de_la_distribucion_automatica_An_application_of_Graph_Theory_to_improve_the_planning_of_wo
3. Trabajo Fin de Grado Grado en Ingeniería de las Tecnologías de Telecomunicación (GITT) Algoritmos para el análisis de redes y grafos - Universidad de Sevilla, fecha de acceso: julio 12, 2025, https://biblus.us.es/bibing/proyectos/abreproy/95270/descargar_fichero/TFG-5270+Hidalgo+Tapia.pdf
4. (PDF) Aplicación de la Teoría de Grafos para mejorar la ..., fecha de acceso: julio 12, 2025, https://www.researchgate.net/publication/28234417_Aplicacion_de_la_Teoria_de_Grafos_para_mejorar_la_planificacion_de_rutas_de_trabajo_de_una_empresa_del_sector_de_la_distribucion_automatica
5. APLICACIÓN DE LA TEORÍA DE GRAFOS EN LA OPTIMIZACIÓN DE REDES DE TRANSPORTE - CIENCIA INTELIGENTE, fecha de acceso: julio 12, 2025, https://cienciainteligente.com/index.php/CIN/article/download/10/10/35
6. APLICACIÓN DE LA TEORÍA DE GRAFOS EN LA OPTIMIZACIÓN DE REDES DE TRANSPORTE | CIENCIA INTELIGENTE, fecha de acceso: julio 12, 2025, https://cienciainteligente.com/index.php/CIN/article/view/10
7. Bfs Y Dfs - FasterCapital, fecha de acceso: julio 12, 2025, https://fastercapital.com/es/tema/bfs-y-dfs.html
8. CAMINO MAS CORTO: ALGORITMO DE BELLMAN-FORD | Algorithms and More, fecha de acceso: julio 12, 2025, https://jariasf.wordpress.com/2013/01/01/camino-mas-corto-algoritmo-de-bellman-ford/
9. (PDF) Comparative analysis of Bellman-Ford and Dijkstra's algorithms for optimal evacuation route planning in multi-floor buildings - ResearchGate, fecha de acceso: julio 12, 2025, https://www.researchgate.net/publication/379538813_Comparative_analysis_of_B

ellman-Ford_and_Dijkstra's_algorithms_for_optimal_evacuation_route_planning_in_multi-floor_buildings
10. Breadth-First Search vs Depth-First Search: Key Differences - Codecademy, fecha de acceso: julio 12, 2025, https://www.codecademy.com/article/bfs-vs-dfs
11. Secuencia competencial | Optimización de rutas, fecha de acceso: julio 12, 2025, https://descargas.intef.es/recursos_educativos/ODES_SGOA/Bachillerato/Matematicas/6E4_SA_BACH_MATG_correos/secuencia_competencial.html
12. Tema: Recorrido de Grafos. Aplicaciones de Grafos., fecha de acceso: julio 12, 2025, https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-9.pdf
13. 3.3 ORDENAMIENTOS TOPOLÓGICOS | compdiscretas, fecha de acceso: julio 12, 2025, https://compdiscretas.wordpress.com/2012/11/18/3-3-ordenamientos-topologicos/
14. Ordenamiento topológico - Wikipedia, la enciclopedia libre, fecha de acceso: julio 12, 2025, https://es.wikipedia.org/wiki/Ordenamiento_topol%C3%B3gico
15. 3.4 COMPONENTES BICONEXAS Y PUNTOS DE ARTICULACIÓN - compdiscretas, fecha de acceso: julio 12, 2025, https://compdiscretas.wordpress.com/2012/11/17/3-4-componentes-biconexas-y-puntos-de-articulacion/
16. Identify Articulation Points in an Undirected Graph (Solved), fecha de acceso: julio 12, 2025, https://www.altcademy.com/blog/identify-articulation-points-in-an-undirected-graph-solved/
17. Componente fuertemente conexo - Wikipedia, la enciclopedia libre, fecha de acceso: julio 12, 2025, https://es.wikipedia.org/wiki/Componente_fuertemente_conexo
18. Algorithm for Finding SCC (Strongly Connected Components) in Graphs - Hypermode, fecha de acceso: julio 12, 2025, https://hypermode.com/blog/algorithm-for-finding-scc
19. Tarjan's Strongly Connected Components Algorithm - Youcademy, fecha de acceso: julio 12, 2025, https://youcademy.org/tarjans-scc-algorithm/
20. Tarjan's strongly connected components algorithm - Wikipedia, fecha de acceso: julio 12, 2025, https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm
21. Unlocking Efficiency with Minimum Spanning Tree - Number Analytics, fecha de acceso: julio 12, 2025, https://www.numberanalytics.com/blog/minimum-spanning-tree-efficiency-guide
22. The shortest route for transportation in supply chain by minimum ..., fecha de acceso: julio 12, 2025, https://www.researchgate.net/publication/282468076_The_shortest_route_for_transportation_in_supply_chain_by_minimum_spanning_tree

23. Optimizing with Strongly Connected Components - Number Analytics, fecha de acceso: julio 12, 2025, https://www.numberanalytics.com/blog/optimizing-with-strongly-connected-components