



Universidad Nacional Experimental De Guayana

Vicerrectorado Académico

Coordinación General de Pregrado

P.F.G: Ingeniería en informática.

Unidad Curricular: Bases de Datos II.

# **Sistema de Optimización de Rutas de Entrega con Neo4j y Cypher**

Profesor:

Clinia Cordero

Autores:

Jose Ortiz 30.810.658

Jeannybeth Herrera 30.664.502

Joselyn Velásquez 31.058.772

Mauricio Arismendi 30.820.542

Karliani Scholtz 28.682.933

Ciudad Guayana, julio del 2025

## Planteamiento del Problema

La empresa de entregas a domicilio enfrenta dificultades para optimizar las rutas de distribución dentro de una ciudad altamente congestionada. Las entregas, al depender de zonas y calles interconectadas, pueden retrasarse por factores como tráfico, bloqueos temporales o zonas de alta demanda. Por lo tanto, se requiere una solución que permita modelar estas relaciones de forma flexible y eficiente, para luego consultar rutas óptimas, realizar simulaciones de incidentes (como el cierre de calles) y analizar la conectividad global de la red.

El uso de bases de datos relacionales tradicionales no ofrece la agilidad suficiente para modelar redes altamente conectadas y dinámicas. En este sentido, se optó por el uso de una base de datos de grafos (Neo4j), con consultas en Cypher, que permite representar las relaciones de una forma más intuitiva, poderosa y eficiente.

## Objetivo General

Modelar una red de distribución para una empresa de delivery utilizando Neo4j, con el fin de consultar rutas óptimas, gestionar incidentes de red en tiempo real, analizar puntos críticos y mejorar la eficiencia de las entregas.

## Objetivos Específicos

- Diseñar un modelo de grafo que represente zonas y calles de la ciudad.
- Implementar nodos y relaciones en Cypher con propiedades relevantes como tiempo de tránsito y tráfico.
- Consultar rutas más rápidas usando algoritmos de camino mínimo.
- Detectar zonas con alto tráfico o posibles puntos de congestión.
- Simular el cierre o apertura de calles.
- Analizar la conectividad y el impacto de incidentes en la red de distribución.
- Visualizar la red a través de HTML, CSS y JavaScript.

## Marco Teórico

### ➤ Base de Datos de Grafos – Neo4j

Neo4j es una base de datos NoSQL orientada a grafos. Permite representar datos como nodos (entidades) y relaciones (conexiones) con propiedades. Esta estructura es ideal para escenarios donde la conectividad entre los elementos es más importante que los datos individuales, como en redes sociales, sistemas de recomendación o en este caso, redes de entrega urbana.

### ➤ Cypher – Lenguaje de Consulta

Cypher es el lenguaje de consulta de Neo4j. Utiliza una sintaxis declarativa muy similar al SQL, pero adaptado a grafos, permitiendo consultar nodos, relaciones, caminos, aplicar filtros, actualizaciones, entre otros.

### ➤ Algoritmo de Dijkstra

Es un algoritmo que permite calcular el camino más corto entre un nodo origen y uno destino, considerando pesos en las aristas. Es fundamental en sistemas de rutas, GPS y delivery.

## Diseño del Modelo de Datos

El diseño del modelo de datos para este proyecto se fundamenta en una base de datos de grafos, utilizando Neo4j. Esta elección arquitectónica es óptima para la representación y análisis de estructuras relacionales complejas, lo cual es esencial para el sistema de cálculo de rutas implementado. Toda la interacción con este modelo, desde la creación hasta la consulta y manipulación, se realiza a través de Cypher, el lenguaje de consulta declarativo de Neo4j, que permite expresar patrones de grafos de manera intuitiva y eficiente.

Centralmente, el modelo define nodos (entidades) de tipo Lugar. Cada instancia de Lugar posee atributos fundamentales como nombre (una clave de identificación unívoca) y tipo

(una categorización que define su rol dentro de la topología de la red, como "CentroDistribucion", "Zona" o "Interseccion"). Los nodos clasificados como "CentroDistribucion" actúan como puntos de partida en la red, mientras que los nodos "Zona" representan los puntos finales o destinos. Los nodos de tipo "Interseccion" son cruciales, ya que modelan puntos intermedios en la red, facilitando el enrutamiento a través de caminos múltiples. La definición y poblamiento de estos nodos se logra directamente con sentencias Cypher.

La conectividad lógica y física entre estos nodos Lugar se establece mediante relaciones (aristas) etiquetadas como CONECTA\_A. Cada relación CONECTA\_A denota una conexión directa entre dos nodos y, críticamente, incorpora una propiedad tiempo. Esta propiedad numérica (flotante o entera, en minutos) cuantifica la duración estimada del tránsito entre los nodos conectados. Aunque la direccionalidad de estas relaciones puede adaptarse a variaciones asimétricas de tiempo (ej., calles de un solo sentido), en el contexto de este proyecto, se asume la bidireccionalidad con tiempos de viaje consistentes para simplificar la implementación. El esquema fundamental del grafo se expresa mediante la sintaxis de patrones de Cypher como (Lugar {nombre, tipo}) - [CONECTA\_A {tiempo}]-> (Lugar {nombre, tipo}). Este modelo de grafo es instrumental para la aplicación eficiente de algoritmos de optimización de rutas, como Dijkstra, cuyas ejecuciones y resultados son orquestados mediante llamadas a procedimientos en Cypher.

## **Implementación y Desarrollo**

La implementación de este sistema se estructura en dos componentes interdependientes: el backend (servidor) y el frontend (interfaz de usuario), colaborando a través de solicitudes de red para ofrecer la funcionalidad central de gestión y cálculo de rutas. El backend está construido sobre Node.js, empleando el framework Express.js para la gestión de solicitudes HTTP y el Neo4j JavaScript Driver para la interacción programática con la base de datos de grafos Neo4j. Es en esta capa donde el lenguaje de consulta Cypher es intensivamente utilizado para todas las operaciones de base de datos. Para las operaciones de cálculo de ruta, el backend integra y ejecuta algoritmos

avanzados, como el algoritmo de Dijkstra, mediante la Biblioteca de Algoritmos de Grafos (GDS) de Neo4j. Estas ejecuciones se invocan y sus parámetros se gestionan directamente a través de consultas Cypher optimizadas, lo que permite un procesamiento eficiente de grafos directamente en el motor de la base de datos. Las funcionalidades expuestas por el backend incluyen un endpoint GET `/api/lugares` que retorna una representación JSON de todos los nodos Lugar del grafo, obtenidos mediante consultas Cypher. Adicionalmente, un endpoint POST `/api/ruta-mas-rapida` recibe parámetros de origen y destino, construye y ejecuta la consulta Cypher apropiada para el algoritmo de Dijkstra, y devuelve la secuencia de nodos de la ruta y el costo temporal total. La robustez del backend se complementa con un sofisticado manejo de errores, que gestiona adecuadamente fallos de conexión a la base de datos, rutas inexistentes o parámetros de entrada inválidos, comunicando mensajes de error descriptivos al frontend.

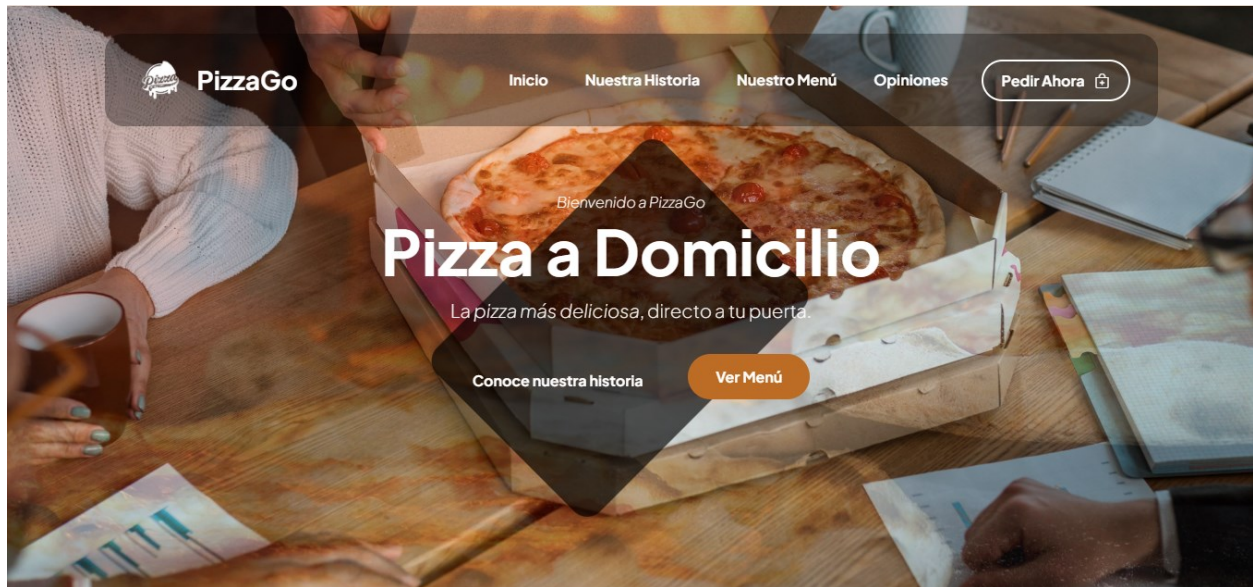
Por su parte, el frontend constituye la interfaz de interacción al usuario final. Se desarrolla utilizando HTML5 para definir la estructura semántica de la página y los elementos del formulario, CSS3 para el estilizado visual y la adaptabilidad a diferentes dispositivos, y Bootstrap 5 para acelerar el desarrollo mediante componentes reutilizables y un sistema de cuadrícula responsivo. La interactividad y la lógica del lado del cliente son gestionadas completamente por JavaScript (Vanilla JS). Las funcionalidades clave del frontend abarcan un formulario estructurado para la entrada de datos, incluyendo selectores de origen y destino que se populan dinámicamente: al cargar la página, una petición asíncrona (fetch) al backend (`/api/lugares`) recupera los datos de los nodos (originalmente consultados con Cypher), filtrando y presentando solo los CentroDistribucion para el origen y las Zona para el destino. La interacción principal se centra en un botón de acción que, al ser activado, desencadena una serie de validaciones tanto a nivel de formulario HTML5 como a nivel de lógica JavaScript (asegurando la selección de origen y destino válidos y distintos). Si todas las validaciones son exitosas, se inicia una petición asíncrona al backend (`/api/ruta-mas-rapida`) con los parámetros de la ruta. el frontend implementa una funcionalidad para resetear el formulario después de una operación exitosa, permitiendo nuevas interacciones sin requerir una recarga completa de la página, lo que optimiza la experiencia del usuario.

## **Conclusión**

El uso de Neo4j para modelar una red de entregas resultó altamente eficiente para consultas de rutas óptimas, manejo de incidentes y análisis de conectividad. La flexibilidad de los grafos supera a las bases relacionales en este tipo de escenarios. Además, su integración con la web (HTML, CSS, JS) permite una experiencia visual amigable para los usuarios finales o administradores del sistema de logística. Se recomienda esta arquitectura para cualquier empresa de delivery que desee optimizar su red urbana de entregas.

## Anexos

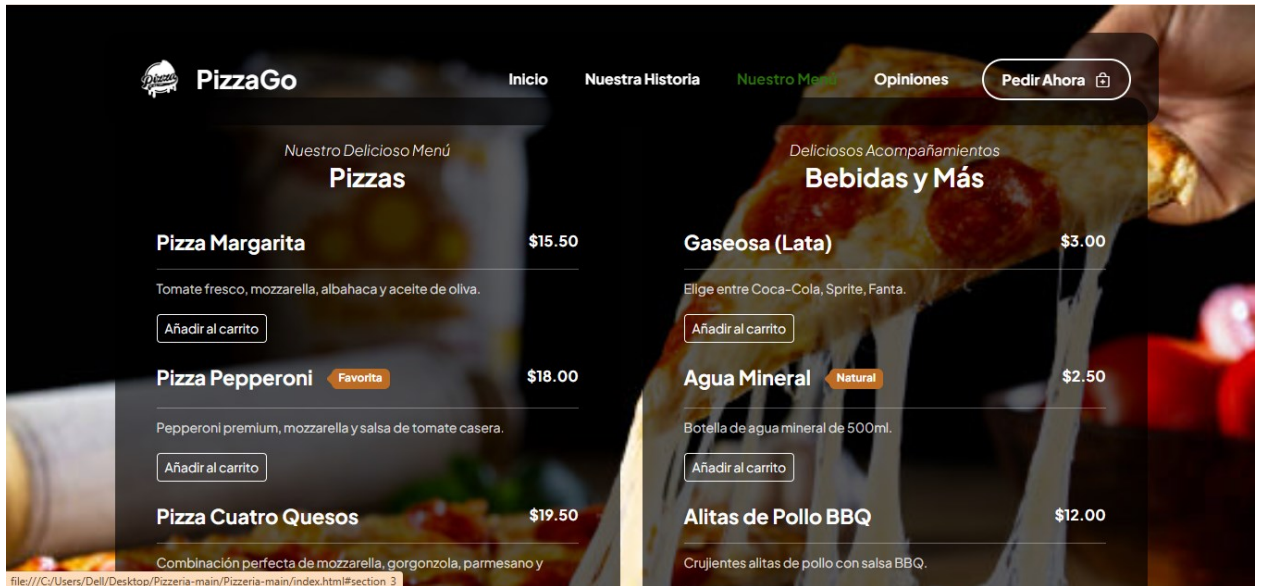
### Ventana de inicio:



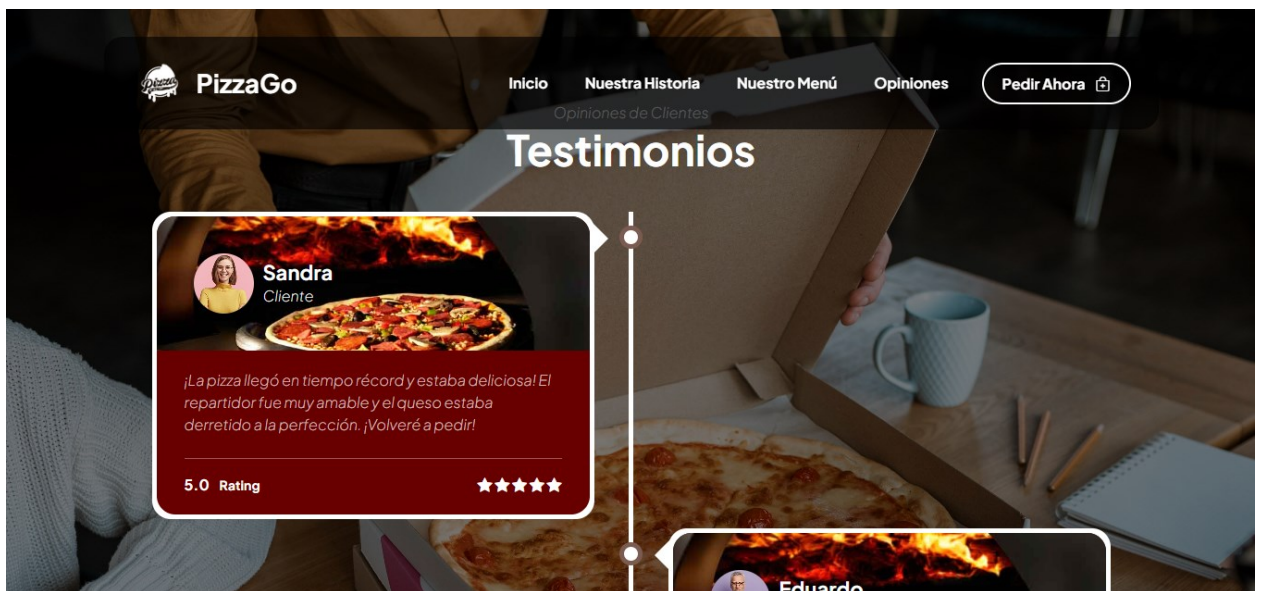
### Ventana nuestra historia:



## Ventana nuestro menu:



## Opciones:





## Pedir ahora:


**PizzaGo**  
Realiza tu Pedido a Domicilio

[Inicio](#)
[Nuestra Historia](#)
[Nuestro Menú](#)
[Opiniones](#)
[Pedir Ahora](#)

## Pedir Ahora y Recibir en Casa

### Información del Cliente y Pedido

Nombre Completo\*

Teléfono\*

Dirección de Entrega\*

¿Qué deseas ordenar?

### Detalles de Envío y Ruta

Sucursal de Origen :

Zona de Entrega :

[Realizar Pedido](#)

### Información de tu Entrega:

El tiempo estimado para su entrega es de: minutos.

## Base de datos en Neo4j Desktop

Neo4j Desktop 2.0.1  
File Edit View Window Help Developer

neo4j Desktop

[Feedback](#)
[Fully-managed cloud database Neo4j AuraDB](#)

**Data services**

- Local instances
- Remote connections
- Import
- Tools**
  - Query**
  - Explore
- About**
  - Settings

Instance: Proyecto Database: neo4j User: neo4j

**Database information**

Nodes (24)

- CentroDistribucion Zona

Relationships (18)

- CONECTA


Property keys

- capacidad data id name nodes nombre
- relationships style tiempo\_minutos
- tipo\_zona trafico\_actual visualisation

neo4j\$

neo4j\$ MATCH p=()-[:CONECTA]->() RETURN p LIMIT 25;

Graph Table RAW



**Results overview**

Nodes (24)

- (24) CentroDistribucion (2) Zona (10)

Relationships (18)

- (18) CONECTA (18)

Automatic updates of node and relationship counts have been disabled for performance reasons, likely

Started streaming 18 records after 2 ms and completed after 9 ms.