

Minería de Datos y Redes Sociales

Proyecto Final



Equipo 4:

Ximena Denise Macías Mateos - 0245555

Yael Emiliano Salinas Lozano - 0219039

Sara Rocio Miranda Mateos - 0244643

Edgar Daniel Chacón Amaro - 0213679

Javier André Soto Hernández - 0237568

Introducción

El proyecto consiste en una mini-red social basada en el modelo de contenido visual (similar a Pinterest), donde los usuarios pueden crear y compartir imágenes (Pines), organizarlas en colecciones temáticas (Tableros o Boards), e interactuar a través de seguir a otros usuarios, dar likes y comentar.

Objetivo

El principal objetivo de usuario de esta plataforma es facilitar el descubrimiento y la organización de contenido visual. Esto se logra permitiendo a los usuarios curar y organizar el contenido (Pines) de la red en colecciones temáticas personales (Tableros), lo cual se combina con algoritmos de recomendación basados en grafos para sugerir activamente nuevo contenido y cuentas de usuario a seguir, fomentando así la expansión de la red social y la interacción a través de las funciones de LIKES, FOLLOWS y WROTE (Comment).

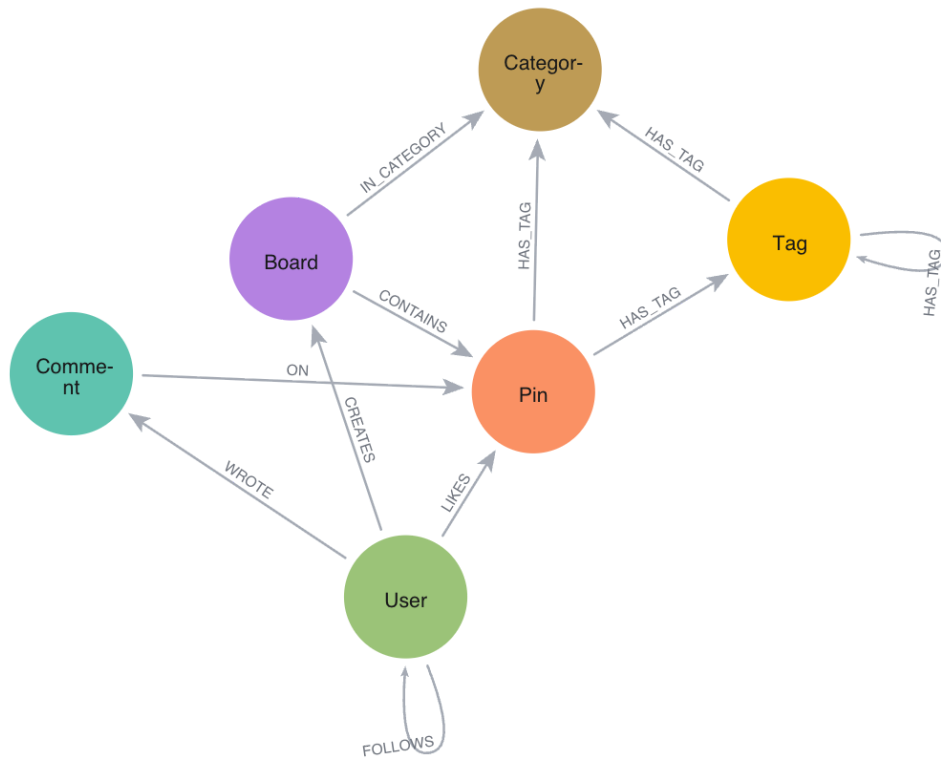
Modelo de grafo

Nodos y propiedades

Etiqueta (Node)	Propiedades Clave	Propósito
User	id_user, name, username, profile_picture	Identificador Único (Primary Lookup).
Pin	id_pin, title, description, url_image, created_at	El contenido visual central de la plataforma.
Board	id_board, title, description, created_at	Colecciones temáticas para organizar Pines.
Comment	id_comment, body, created_at	Interacciones de texto sobre los Pines.
Tag	name	Categorización y descubrimiento de contenido.

Relaciones y Propiedades

Relación (Type)	Desde → Hacia	Propiedades	Propósito
CREATES	User → Pin/Board	Ninguna	Indica la autoría y propiedad del contenido
LIKES	User → Pin	date: datetime	Conexión social para el feed y sugerencias. Permite el Filtro Temporal para calcular tendencias.
FOLLOWS	User → User	since: datetime	Registro de engagement (popularidad y tendencia).
CONTAINS	Board → Pin	Ninguna	Organiza el contenido en colecciones.
WROTE	User → Comment	Ninguna	Conecta el Pin con sus palabras clave descriptivas.
ON	Comment → Pin	Ninguna	Autoría del comentario
HAS_TAG	Pin → Tag	Ninguna	Indica a qué Pin se refiere el comentario.



Justificación de Consultas

El proyecto implementa un total de 13 consultas clave distintas a través de los endpoints de la API. Estas consultas abarcan los requisitos funcionales de la rúbrica (Búsqueda, Recomendación, Tendencias, Centralidad, Comunidades, Subgrafo Personal) y utilizan patrones avanzados de recorrido y agregación de grafos.

Las siguientes consultas demuestran el uso de algoritmos analíticos (GDS Concepts implementados con Cypher):

Requisito Obligatorio	Consulta Implementada (API Endpoint)	Patrón Clave y Justificación

Búsqueda por Texto/Filtros	GET /api/search	Multi-Type Search. Búsqueda de coincidencia de subcadenas en Pines, Usuarios y Tableros, permitiendo el descubrimiento primario.
Amigos/Seguidores de 2º Nivel o "Usuarios Similares"	GET /api/users/:id/similar	Similitud Jaccard. Calcula qué tan similares son dos usuarios basándose en el conjunto de Pines que ambos han dado [LIKES], ideal para sugerencias sociales.
Contenido Recomendado (Similitud)	GET /api/pins/:id/similar	Similitud por Tags. Encuentra Pines que comparten el mismo conjunto de (Tag) mediante recorrido de dos saltos, siendo el núcleo de la recomendación de contenido visual.

Tendencias (Top-N por Likes)	GET /api/trending/pins	Filtro Temporal. Filtra la relación [LIKES] por la propiedad date (últimos 7 días) para identificar Pines que tienen mayor actividad reciente.
Tendencias (Top-N por Uso de Etiquetas)	GET /api/trending/tags	Filtro Temporal y Agregación. Combina el filtro de 7 días con la agregación de [HAS_TAG] para identificar las etiquetas que impulsan la actividad reciente.
Centralidad (Degree)	GET /api/analytics/centrality/users	Centralidad de Grado. Mide la conectividad total de un usuario (followers + following), identificando a los nodos más activos en la red.

Detección de Comunidades	GET /api/analytics/communities/tags	Comunidades por Intereses. Agrupa a los usuarios que demuestran un interés común al dar [LIKES] a Pines con los mismos (:Tag).
Subgrafo Personal (Perfil)	GET /api/user/:id	Recorrido de Propiedad. Obtiene la información básica del usuario, esencial para la vista de perfil.

Consultas Secundarias y de Soporte

Funcionalidad / Propósito	Consulta Implementada (API Endpoint)	Patrón Clave y Justificación
Listado General (Feed)	GET /api/pins	Listado paginado de Pines. Actúa como la vista de <i>Home</i> o <i>Explorar</i> .
Pines Populares (Histórico)	GET /api/pins/trending (All-time)	Agregación de todos los [LIKES] históricos. Complementa las tendencias de 7 días.

Detalle de Contenido	GET /api/pin/:id	Aggregación con Contexto de Usuario. Devuelve el Pin, el conteo de <i>likes</i> y el estado del usuario logueado (ej., <i>likedByMe</i>), optimizando la carga de la vista de detalle.
Perfiles/Tableros	GET /api/boards	Colección con Previsualización. Obtiene los Tableros creados por el usuario y los Pines que contienen (usado para la UX de previsualización).
Recomendación de Contenido (Afinidad)	Pin Suggestions	Sugerencias de contenido basadas en los creadores y los tableros, reforzando la afinidad por <i>creators</i> y curadores.

Algoritmos Graph Data Science (GDS)

Para extraer insights profundos se implementaron consultas Cypher y así demostrar la lógica analítica de grafos directamente sobre el modelo.

Concepto GDS	Implementación en Cypher Nativo	Propósito / API Endpoint

Similitud (Jaccard)	Cálculo de la similitud entre dos Usuarios basado en la intersección de sus [LIKES] compartidos (Patrón: MATCH (me)-[LIKES]->(Pin)<-[LIKES]-(other)).	Recomendación: Sugerir usuarios similares para seguir.
Centralidad (Degree)	Cálculo de la Centralidad de Grado para (Usuario) sumando el total de followers y following (MATCH ()-[r]-(User)).	Análisis: Identificar usuarios más conectados/activos.
Detección de Comunidades	Agrupación de (Usuario) basada en los (Tag) compartidos de los Pines a los que dan [LIKES].	Análisis: Identificar nichos o intereses comunes en la red.
Tendencias	Filtrado temporal de la relación [LIKES] usando la propiedad date y la función datetime() con duration('P7D').	Descubrimiento: Identificar los Pines y Tags más populares de la última semana.

Decisiones de diseño (trade-offs)

Stack Tecnológico

Componente	Tecnología Elegida	Justificación
Base de Datos	Neo4j	Eficiencia al modelar y consultar relaciones sociales (FOLLOWS, LIKES) y

		contenido (HAS_TAG, CONTAINS). Es inherentemente superior a las bases de datos relacionales para las consultas de 2º nivel (Amigos de amigos) y GDS.
Backend (API)	Express.js (Node.js)	Trade-off: Rápido desarrollo y buen rendimiento I/O. Se eligió sobre Python/FastAPI por la familiaridad del equipo con JavaScript.
Frontend (UI)	React	Desarrollo de componentes modular y escalable para las tres vistas requeridas (/ , /pin/:id, /profile).

Modelado de Datos

- Decisión: Modelar (:Comment) como un nodo independiente.
 - Ventaja: Permite adjuntar propiedades ricas al comentario (como un flag de moderación o un score de sentimiento futuro) y conectar al autor con [:WROTE], facilitando la búsqueda de comentarios por usuario.
 - Trade-off: Mayor complejidad en consultas de detalle de Pin (se requiere MATCH adicional).
- Decisión: Usar datetime en relaciones de engagement.
 - Relaciones: [:LIKES] {date} y [:FOLLOWS] {since}.
 - Ventaja: Permite implementar consultas temporales, como la búsqueda de tendencias ("Top-N likes de los últimos 7 días"), cumpliendo con el requisito de Tendencias.

Riesgos y Mitigaciones

Métricas

Métricas de Desempeño

```
🚀 INICIANDO TEST DE RENDIMIENTO DE NEO4J BACKEND
=====
Datos obtenidos para test: User[USER-1], Pin[UB-USER-140-CAT-1-P2]
Probando GET Users (Lista simple) (20 veces)... OK
Probando GET Boards (Lista simple) (20 veces)... OK
Probando GET Feed Pins (Heavy Join) (20 veces)... OK
Probando GET Pin Detail + Sugerencias (20 veces)... OK
Probando Analytics: Usuarios Similares (Jaccard) (20 veces)... OK
Probando Analytics: Centralidad (20 veces)... OK
Probando Search: Búsqueda Texto (20 veces)... OK
Probando POST Like (Escritura) (20 veces)... OK
```

RESULTADOS FINALES:						
(index)	Endpoint	Min (ms)	Max (ms)	Avg (ms)	P95 (ms)	Errores
0	'GET Users (Lista simple)'	'70.24'	'151.92'	'89.60'	'151.92'	0
1	'GET Boards (Lista simple)'	'72.62'	'154.36'	'86.72'	'154.36'	0
2	'GET Feed Pins (Heavy Join)'	'125.93'	'207.38'	'141.10'	'207.38'	0
3	'GET Pin Detail + Sugerencias'	'300.63'	'1267.48'	'408.72'	'1267.48'	0
4	'Analytics: Usuarios Similares (Jaccard)'	'78.45'	'280.93'	'96.07'	'280.93'	0
5	'Analytics: Centralidad'	'69.93'	'135.03'	'80.01'	'135.03'	0
6	'Search: Búsqueda Texto'	'207.15'	'489.69'	'266.27'	'489.69'	0
7	'POST Like (Escritura)'	'197.50'	'349.00'	'229.48'	'349.00'	0

Limitaciones

- **Arquitectura Monolítica y Deuda Técnica:** El backend reside actualmente en un único archivo (`app.js`). Si bien esto agiliza el desarrollo inicial y el despliegue de pruebas, genera un alto acoplamiento. Esto dificulta la mantenibilidad a largo plazo, la colaboración en equipo y la realización de pruebas unitarias aisladas, ya que la lógica de negocio, el enrutamiento y el acceso a datos no están desacoplados.
- **Cómputo Intensivo en Tiempo Real:** Algoritmos complejos como la Similitud de Jaccard (para recomendación de usuarios) y la Centralidad de Grado se ejecutan sincrónicamente en cada petición HTTP. En un entorno de producción con miles de nodos y relaciones, esto provocaría una latencia inaceptable y bloquearía el *Event Loop* de Node.js, degradando la experiencia de usuario (UX). El costo computacional de recorrer el grafo crece exponencialmente con el volumen de datos.
- **Ausencia de Capa de Seguridad Robusta:** El sistema actual carece de un middleware de autenticación estándar (como JWT o OAuth2). La identificación se realiza pasando el `userId` directamente en el cuerpo o la

query de las peticiones, lo cual es funcional para demostraciones pero vulnerable a ataques de suplantación de identidad (IDOR - Insecure Direct Object References) en un entorno productivo.

- **Escalabilidad de Lectura:** Aunque se utiliza `LIMIT` en las consultas Cypher, no existe un sistema de paginación real (basado en cursores) para el *feed* infinito. Esto significa que el rendimiento de la base de datos disminuirá a medida que el historial de "Pines" crezca, ya que el motor debe escanear y ordenar grandes volúmenes de datos antes de recortar los resultados.

Trabajo futuro

- **Refactorización a Arquitectura MVC o Microservicios:** Dividir la aplicación en capas lógicas: **Controladores** (lógica de entrada), **Servicios** (lógica de negocio) y **Repositorios** (consultas a Neo4j). Esto permitirá una mejor organización del código y facilitará la implementación de pruebas automatizadas (Testing con Jest/Supertest).
- **Optimización de Algoritmos de Grafos (Batch Processing):** Mover el cálculo de métricas pesadas (Jaccard, Centralidad, Tendencias) a procesos en segundo plano (*background jobs*) que se ejecuten periódicamente (por ejemplo, cada noche o cada hora). Los resultados se pre-calcularían y almacenarían como propiedades en los nodos (`u.similarityScore`), permitiendo lecturas instantáneas $O(1)$ en lugar de cálculos en tiempo real.
- **Implementación de Caching (Redis):** Integrar una capa de caché en memoria (Redis) para almacenar datos de acceso frecuente, como el *feed* de tendencias o los perfiles de usuarios populares. Esto reduciría drásticamente la carga directa sobre Neo4j y mejoraría los tiempos de respuesta.
- **Seguridad y Autenticación:** Integrar un proveedor de identidad como Auth0 o Firebase Auth para gestionar tokens JWT seguros. Esto eliminaría la dependencia de enviar IDs de usuario en texto plano y permitiría implementar roles y permisos (ej. administradores vs. usuarios estándar).
- **Infraestructura y Dockerización:** Crear contenedores Docker para el backend y la base de datos, asegurando que el entorno de desarrollo sea idéntico al de producción, facilitando el despliegue continuo (CI/CD) en la nube.