**Objetivo del Proyecto**

El objetivo es crear una aplicación web full-stack que permita a los usuarios explorar la red de conocimiento de Wikipedia como un grafo interactivo. En lugar de simplemente leer un artículo, los usuarios podrán visualizar cómo un concepto se conecta con otros, descubriendo relaciones y expandiendo su exploración de forma dinámica.

***El Concepto Central: El Grafo de Conocimiento***

Consideramos Wikipedia como un grafo dirigido donde cada artículo es un **Nodo** y cada hipervínculo interno es una **Arista** que conecta dos nodos. Tu misión es construir una herramienta que permita a un usuario iniciar en un nodo (un artículo) y explorar visualmente sus conexiones.

***Stack Tecnológico Sugerido***

* **Backend:** FastAPI (Python con type hints)
* **Frontend:** React (con TypeScript)
* **Base de Datos:** MongoDB (Neo4J Online)

***Requisitos del Proyecto***

**1. Lógica y API del Backend (FastAPI)**

El backend es el cerebro de la operación y debe modelar la lógica del grafo.

* **Endpoint de Búsqueda de Artículos:**

GET /api/search?term={query}   
 

Recibe un término de búsqueda, consulta la API de Wikipedia y devuelve una lista de artículos sugeridos para que el usuario elija su "nodo de inicio".

* **Endpoint de Exploración del Grafo:**

GET /api/explore/{article\_title}?depth={level}   
 

Este es el endpoint principal. Recibe el título de un artículo (nodo raíz) y un nivel de **profundidad** (ej. depth=1).

* + **Lógica:**
    - Obtiene el contenido del artículo principal (article\_title).
    - Extrae todos los enlaces a otros artículos de Wikipedia que contiene. Estos son sus vecinos de profundidad=1.
    - *Opcional pero recomendado:* Para cada vecino, obtén también su resumen.
  + **Respuesta:** Debe devolver una estructura de datos de grafo en JSON que el frontend pueda visualizar. Formato sugerido:

{   
  "nodes": [   
    { "id": "Albert Einstein", "label": "Albert Einstein", "summary": "..." },   
    { "id": "Theory of relativity", "label": "Theory of relativity", "summary": "..." }   
  ],   
  "edges": [   
    { "from": "Albert Einstein", "to": "Theory of relativity" }   
  ]   
}   
 

* **Endpoints CRUD para "Exploraciones Guardadas":** Los usuarios pueden querer guardar el estado de un grafo que han explorado.
  + POST /api/explorations: Guarda una "instantánea" de un grafo (la colección de nodos y aristas) en la base de datos.
  + GET /api/explorations: Lista las exploraciones guardadas.
  + DELETE /api/explorations/{exploration\_id}: Elimina una exploración guardada.

**2. Interfaz de Usuario (Frontend)**

El frontend debe dar vida a la exploración del grafo.

* **Página Principal:** Debe contener una barra de búsqueda para encontrar el artículo inicial.
* **Vista de Exploración:**
  + Al seleccionar un artículo, esta vista debe renderizar el grafo devuelto por el backend.
  + La visualización debe ser **interactiva**. Hacer clic en un nodo podría, por ejemplo, mostrar su resumen en un panel lateral.
  + **Expansión dinámica:** Al hacer doble clic (o mediante un botón en el nodo), la aplicación debe llamar al backend para explorar ese nuevo nodo (ej. GET /api/explore/Theory\_of\_relativity?depth=1) y añadir los nuevos nodos y aristas al grafo existente en el frontend, expandiendo la visualización sin recargar la página.
* **Sección "Mis Exploraciones":** Un lugar para ver y cargar las exploraciones de grafos previamente guardadas.
* **Diseño:** Debe ser responsivo y manejar adecuadamente los estados de carga (mientras se explora) y los errores de la API.

***Características Adicionales (Opcional)***

Si tienes tiempo y quieres demostrar tu seniority, considera implementar una o más de las siguientes:

* **Análisis de Centralidad del Grafo:** En el backend, calcula y muestra una métrica simple para cada nodo, como el **Grado de Centralidad** (número de enlaces entrantes/salientes dentro del grafo explorado). Muestra esta métrica en el frontend (ej. haciendo el tamaño del nodo proporcional a su centralidad).
* **Búsqueda del Camino Más Corto:** Implementa un endpoint GET /api/path?from={article1}&to={article2} que encuentre y devuelva el camino más corto de enlaces entre dos artículos de Wikipedia. Visualiza este camino en el frontend. (Pista: investiga sobre el algoritmo de Búsqueda en Amplitud - BFS).
* **Uso de una Base de Datos de Grafos:** En lugar de MongoDB, implementa la persistencia usando una base de datos nativa de grafos como Neo4j o ArangoDB.
* **Exploración en Tiempo Real con WebSockets:** A medida que el backend descubre nuevos nodos mientras explora, envíalos al frontend en tiempo real usando WebSockets para una experiencia de usuario más fluida.

***Criterios de Evaluación Clave***

Nos enfocaremos en:

* **Calidad y Diseño del Código:** Claridad, organización, eficiencia y buenas prácticas (SOLID, DRY).
* **Abstracción y Modelado de Datos:** Tu enfoque para modelar el grafo, tanto en la lógica del backend como en el esquema de la base de datos, es **fundamental**.
* **Comprensión Algorítmica:** Cómo manejas la exploración del grafo y la prevención de bucles o llamadas redundantes.
* **Diseño de API REST:** Claridad, consistencia y pragmatismo de los endpoints.
* **Gestión de Estado en Frontend:** Cómo manejas un estado complejo y dinámico (el grafo) en React.
* **Funcionalidad y Usabilidad:** La aplicación funciona como se espera y la experiencia de usuario para explorar el grafo es intuitiva.
* **Documentación (README.md):**
  + Instrucciones claras de configuración y ejecución.
  + **Descripción de tus decisiones de arquitectura y diseño (muy importante).** ¿Por qué elegiste ese esquema de base de datos? ¿Qué desafíos encontraste al modelar el grafo?
  + Definición clara de los endpoints de la API.

***Entrega***

Sube tu solución a un repositorio Github y envíanos el enlace. Asegúrate de que el repositorio incluya todo el código fuente y el README.md. ¡Mucha suerte!