GraphQL con Neo4J

Luis Miguel Urbez Villar

Programación Concurrente
Ingeniería Informática

Resumen

*E*ste trabajo ha sido usado para aprender las bases de la nueva forma de aplicar clientes API Rest en la programación moderna usando GraphQL, un modelo que refresca las ideas de la programacion del lado del cliente y servidor. Para hacer aun más completa la realización de este proyecto h optado por usa una base de datos con un modelo de grafos, en este caso, Neo4J. Base famosa y con una amplia documentación muy util para usar grandes cantidades de datos usando la concurrencia que ofrecen los grafos. Se puede encontrar completo en el siguiente repositorio de Github.

1 Enunciado

2 Sistema de Administración de Biblioteca en Línea con GraphQL y Programación Concurrente

Este proyecto tiene como objetivo diseñar y desarrollar un sistema de administración de biblioteca en línea utilizando GraphQL y técnicas de programación concurrente.Este sistema permitirá a los usuarios buscar libros, autores y categorías de libros. Además, los usuarios podrán hacer reservas y pedir prestados libros electrónicamente. Los administradores podrán agregar, editar y eliminar libros, autores y categorías.

2.1 Funcionalidades Implementadas

2.1.1 1. Buscar Libros

- Los usuarios pueden buscar libros por título, autor o categoría utilizando consultas GraphQL.
- El sistema responde a estas consultas obteniendo datos de la base de datos mediante GraphQL.

2.1.2 2. Gestionar Reservas y Préstamos

- Los usuarios pueden hacer reservas y pedir prestados libros electrónicamente.
- Se utiliza programación concurrente para manejar eficientemente múltiples solicitudes.

2.1.3 3. Gestionar Libros, Autores y Categorías

- Los administradores pueden agregar, editar y eliminar libros, autores y categorías.
- Mutaciones de GraphQL se utilizan para realizar estas operaciones en la base de datos.

2.1.4 4. Interfaz de Usuario

Se proporciona una interfaz de usuario fácil de usar para que tanto usuarios como administradores interactúen con el sistema.

2.1.5 5. Pruebas

- Se implementan pruebas que cubren consultas y mutaciones de GraphQL.
- Las pruebas abarcan la lógica de programación concurrente y la funcionalidad de la interfaz de usuario.

3 Primeros pasos

Para la creación de este proyecto he necesitado nutrirme de mucha información externa sobre la base de datos de Neo4J, ya que esta para mi era completamente nueva. Pese a tener una amplia documentación, esta es a menudo confusa y es necesario nutrirse de otras bases como son los grafos o el lenguaje por consultas como çypher".

Para comenzar el proyecto aprendi las bases del lenguaje cypher y me adentre en la creacion de mi propia base de datos para almacenar mi biblioteca. En el Readme.md llamado Çreación de Neo4j.[es]ta explicado de forma más extensa y concisa como funciona cada consulta y parte de la base de datos.

Para comenzar, los CSV. EStos, al igual que mis anteriores trabajos de esta asignatura han sido creados a través de la pagina web de Mockaroo. Esta ofrece infinitas posibilidades para la creacion de documentos para bases de datos, y para mi biblioteca no iba a ser diferente. Primero creé los objetos que iba a usar como nodos en mi base de datos por grafos. Estos objetos tienen en su interior los datos

necesarios con los cuales son creados, al igual que las relaciones que tienen con distintos nodos.

Los objetos que iban a requerir mi base de datos serian los siguientes:

- Usuario.csv: https://www.mockaroo.com/e5e409b0
- Bibliotecario.csv: https://www.mockaroo.com/b32ebdd0
- Administrador.csv: https://www.mockaroo.com/eca16b10
- Libro.csv: https://www.mockaroo.com/7e4b1300
- Cliente.csv: https://www.mockaroo.com/c0cebe90

Tras conseguir realziar con exito todos mi requerimientos para mi base de datos, me adentre en la creacion de la misma. Pese a tener una interfaz grafica amigable, la mejor forma de trabajar con Neo4j es usando las consultas con su propio lenguaje, cypher. Lenguaje de consultas a priori engorroso que con el paso de varias consultas se acaba haciendo amigable y más entendible.

La base de datos puede ser accedida desde la siguiente web:

- Web gráfica de la base de datos: https://workspace-preview.neo4j.io/workspace/explore
- Las credenciales para entrar a la base de datos se encuentran en el archivo ubicado en la raiz del proyecto llamado ”Neo4j-65bc96ba-Created-2023-12-24.txt”.

Como ya he dicho, todas las consultas para la realización de la base de datos estan en el Çreacion de Neo4j.md”, el cual es un documento con todo lujo de detalles sobre como se crean nuevos nodos, relaciones, mutaciones y demás peculiaridades que tienen este tipo de bases.

Para terminar con el inicio de este proyecto, terminé la vision general de mi Neo4j y me adentraría a la realización de la parte logica que controlaría y enseñaría al usurio esta base de datos.

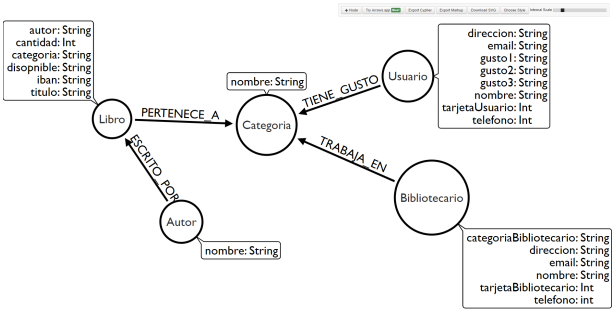

Figura 1: *Modelo contextual de las relaciones de la base*

4 Inicialización de la lógica

Para comenzar este proyecto sabia que iba a requerir de 3 partes fundamentales para el funcionamiento del mismo. Un backend que trabajase las peticiones a la base de datos, un frontend el cual maquetara todas esas peticiones y sus respuestas, y un middleware el cual ayudara con demás conexiones del trabajo.

4.1 G.R.A.N.D. Stack

En cuanto empecé a indagar un poco encontre el proyecto de William Lyon, trabajador interno de Neo4J. William es el creador de uno de los mejores inicializadores que, a día de hoy, existen para realizar proyectos full stack con Neo4j, GraphQL y React. El nombre, el cual tiene un significado, nos esta dando las bases que usara este inicializador. La explicación más detallada del funcionamiento de este inicializador se encuentra en los documentos oficiales y la gran cantidad de videos del propio William Lyon explicando excelentemente como funciona internamente el proyecto. (Bolg con la documentación)

Para los primeros pasos en este ejercicio me guié mucho por los videos y la documentacion del propio William, el cual tiene un repositorio el cual explica y enseña las entrañas del inicializador del proyecto aqui.

Cabe destacar, que como ya nos muestra el repositorio de github, este proyecto ha sido archivado desde el 2022, por lo cual la tecnologia que usa esta la mayoria de ella obsoleta, pero como esqueleto es util para la organización del trabajo.

Esto ultimo es el mayor problema que tiene este inicializador. El avance constante y continuo de las tecnologias ha conseguido que se quede en su mayor parte obsoleto, sin embargo, volver a ponerlo al orden del dia no es del todo dificil si sabes donde hay que cambiar las cosas. Esto ultimo, pese a tedioso, se acabo convirtiendo en algo sencillo y repetitivo. En un principio, el proyecto no funciona, ya que su deploy con Node y React usa versiones que los servidores principales como Vercel o Render no mantienen ya. Por lo que el primer paso a realizar es crear una completa reinstalación de la parte del frontend (web-react), asi de esta manera tambien aproveché para poder usar el lenguaje que estaba mas cómodo para realizar el frontend, TypeScript con React, ya que de primeras viene inicializado con Javascript nativo.

Dejando atrás el problema de las implementaciones de React, lo único que queda por solucionar para que el proyecto sea usable son los 3 package.json. Estos pertenecen cada uno a una de las partes del proyecto, frontend, backend y el correspondiente a la raíz. Las dependencias de estos la mayoria estan marcadas como ”deprecadas”, ya que son versiones no soportadas por node y las tecnologias de graphql. Ponerlos al dia es ua tarea sencilla y que tras realizarla, tienes tu entorno funcional y usable. Por parte del backend (api), apenas hay que cambiar configuraciones y dependencias, ya que este al estar escrito en Javascript, y que las tecnologias relacionadas con Apollo y Graphql no han avanzado tanto, hace que usarlos en esas versiones sea viable.

5 Parte Backend

La parte backend en mi proyecto se constituye de la carpeta .ªpi". Esta crpeta en su interior configura el entorno de graphql, neo4j y apollo server. Las configuraciones de todos estos, al igual que los resolvers"de neo4j (funciones en JS usadas para que las querys funcionen más eficientemente), se encuentran en el archivo principal, index.js. Otra parte realemte importante es el esquema de graphql. Este esquema me ha dado muschos problemas ya que las nuevas tecnologias de Neo4j usadas para las relaciones, como etiquetas "@relationshipz otras usadas para la logica de la base de datos, no estaban soportadas en mi entorno. Finalmente tras la implementacion de nuevas dependencias y de reinstalar otras hicieron que la implementación de mi esquema se hiciese posible.

Por otra parte, una incorporacion muy importante para el proyecto es la de Apollo server. ESte permite no solo la conexion en local entre el backend y el frontend, sino que depues de añadir esta parte a un servidor puede usarse de forma remota como si de una api de verdad se tratase. Esta incorporación a mi proyecto ha sido vital, y no solo por lo último mencionado, sino porque tambien te ofrece un "playground"para que puedas crear y probar consultas en la pagina localhost:4001/graphql. Esta ultima es una especie de paginas de entorno de pruebas, la cual se alimenta de todo el backend del proyecto para ofrecerte una api usable a través de consultas gql (GraphQL consultas), para probar las conexiones y consultas a la base de datos.

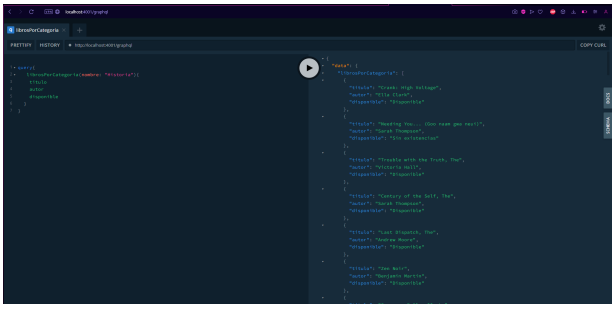

Figura 2: *Consulta sobre las categorias de los libros en Apollo Server Playground*

6 Parte Frontend

Para el frontend, como ya se ha explicado antes, he tenido que reinstalar todo el framework de React, ya que la instalación que ofrece el inicializador de GRAND stack esta obsoleto por las dependencias que usa. De esta forma tambien me permite istalar el framework con mis propias necesidades, como TypeScript con react y tailwind CSS, este último siendo muy util para poder ahorrarme todas las clases de estilos .css y poder aplicar estas en los mismos componentes de TypeScript.

El frontend a priori es sencillo, la web, pese a tener muchas consultas a la api, no es lenta, ya que la concurrencia aplicada del propio neo4j y la que he aplicado yo mismo en el frontend hacen que sea una experiencia de usuario muy correcta teniendo en cuenta que la base de datos maneja más de 4000 libros, con sus datos, categorias, autores, etc.

En un principio lo más complicado de la parte frontend fue hacer funcionar Tailwind CSS, ya que la instalación de este no estaba adaptada a los demas niveles de instalacion que soportaba el proyecto, sobretodo por el package.json de la raiz del mismo. Una de las partes más a destacar del desarrollo del frontend es la posibilidad de almacenar todas las Querys en un mismo archivo. Este se encuentra en la caroeta ”querysz me permite acceder a realizar cambios y comprobar informacion de forma sencilla y directa.

Por lo demás, el grosor del frontend se encuentra en la carpeta components. Esta almacena todos los componentes de la web, dandome asi un orden y limpieza más que necesaria para la edición de los mismos y creación de nuevos componentes. El componente que sin duda está mas completo es Lista.tsx, ya que este no solo maneja la query más pesada de todas (GET-LIBROS), la cual se usa para mostrar todos los libros de la base de datos. Sino que también es la encargada de los botones ”toggle.entre la lista de libros y la lista de autores. EStos últimos tienen la funcion de desplegarse para enseñar al usuario que libros han escrito.

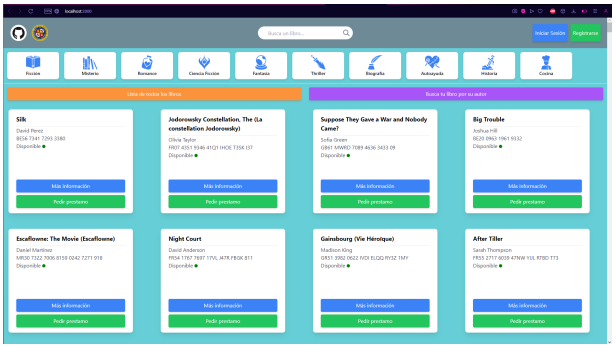

Figura 3: *Pagina principal de la web, mostrando ya los primeros libros de la base de datos.*

Conclusiones y advertencias de uso

Este proyecto me ha llevado bastante tiempo y muchos commits, todos ellos se pueden ver a lo largo del repositorio publico de Github, al igual que las diferentes ramas dentro de este que he seguido para que todo finalmente quede bien montado. Pese a que tiene errores y faltan implementaciones, como un sistema mejorado de roles al iniciar sesion, ya que este aún no esta implementado del todo. El objetivo principal que me plnateé al comienzo del proyecto lo he conseguido, hacer funcionar una base de datos Neo4j creada por mi mismo en un entorno frontend. Pese a que no es posible usar la web al completo en el servidor de vercel, ya que este no admite el backend de graphql, en localhost es completamente usable. Para iniciar el proyecto debes posicionarte en la raiz del mismo en la terminal de comandos, una vez estas en la raiz es aconsejable realizar una build del mismo, usando npm run build. Tras realziar la build del proyecto puedes ejecutarlo usando npm run start, tras esto se te abrira automaticamente la web del mismo, si quieres usar la pagina de Apollo con su playground a la base de datos puedes usar el entorno localhost:4001/graphql.

En conclusión, creo que la base de datos neo4j es excepcionalmente útil, no solo por su estilo de construccion de grafos, sino por la rapidez y concurrencia que ofrece esta con sus consultas cypher. El uso de graphql me ha resultado más amigable y rapido que al que venía más que acostumbrado de modelo vista controlador. Creo que tiene muchas más utilidad este modelo de Graphql, por su rapidez y concurrencia que ayuda tanto al desarrollador como al cliente que esta usando la web.

Referencias

[1] Grand stack starter. [Online]. Available: https://github.com/grand-stack/grand-stack-starter

[2] Introduction to the grandstack. [Online]. Available: https://graphqleditor.com/blog/grandstack/

[3] Apollo client - react. [Online]. Available: https://www.apollographql.com/docs/react/get-started

[4] Introduction to cypher. [Online]. Available: https://neo4j.com/docs/cypher-manual/current/introduction/

[5] Getting started with neo4j. [Online]. Available: https://neo4j.com/developer/get-started/

[6] Getting started with neo4j. [Online]. Available: https://neo4j.com/developer/get-started/

[7] Grand stack starter. [Online]. Available: https://github.com/grand-stack/grand-stack-starter

[8] Proyecto en grand stack. [Online]. Available: https://youtube.com/playlist?list=PL9Hl4pk2FsvUjfSsxLolVToO5t1hwEIKK&si=v-KIEBjPMcpXm4sB

[9] Explicación de grand stack. [Online]. Available: https://www.youtube.com/watch?v=Kz5HMIVgWK0&t=409s