Actividad 14: Conectando a la API GraphQL a la base de datos

Objetivo: Esta actividad proporciona una comprensión profunda de cómo integrar una API GraphQL con una base de datos MongoDB utilizando Mongoose, promoviendo buenas prácticas de desarrollo como la modularidad, la separación de responsabilidades y la optimización de conexiones.

Los ejercicios teóricos refuerzan el entendimiento conceptual, mientras que los prácticos aseguran la aplicación efectiva de los conocimientos adquiridos.

Introducción

Para esta actividad debes utilizar el código dado en la lectura de mongodb y mongoose dado.

Rehagamos la API GraphQL de nuestra aplicación meteorológica para que lea los datos de respuesta desde la base de datos en lugar de desde un archivo JSON estático. El código te resultará familiar, ya que utilizaremos los mismos patrones que en el ejemplo de la API REST.

Primero, verifica que has agregado la implementación de memoria de MongoDB y Mongoose a tu proyecto. A continuación, asegúrate de que has creado los archivos en las carpetas middleware y mongoose.

Ahora, para conectar la API GraphQL a la base de datos, necesitamos hacer dos cosas: implementar la conexión a la base de datos y refactorizar los resolvers de GraphQL para que utilicen sus conjuntos de datos.

Conectando a la base de datos

Para consultar la base de datos a través de la API GraphQL, necesitamos tener una conexión a la base de datos. Todas las llamadas a la API tienen el mismo punto final, /graphql. Este hecho ahora resultará increíblemente conveniente para nosotros; debido a que todas las solicitudes tienen el mismo punto de entrada, solo necesitamos manejar la conexión a la base de datos una vez.

Por lo tanto, abre el archivo api/graphql.ts y modifícalo para que coincida con el código siguiente

```
import {ApolloServer} from "@apollo/server";
import {startServerAndCreateNextHandler} from "@as-integrations/next";
import {resolvers} from "../../graphql/resolvers";
import {typeDefs} from "../../graphql/schema";
import {NextApiHandler, NextApiRequest, NextApiResponse} from "next";
import dbConnect from "../../middleware/db-connect";
//@ts-ignore
const server = new ApolloServer({
    resolvers,
    typeDefs
});
```

```
const handler = startServerAndCreateNextHandler(server);

const allowCors = (fn: NextApiHandler) =>
  async (req: NextApiRequest, res: NextApiResponse) => {
        res.setHeader("Allow", "POST");
        res.setHeader("Access-Control-Allow-Origin", "*");
        res.setHeader("Access-Control-Allow-Methods", "POST");
        res.setHeader("Access-Control-Allow-Headers", "*");
        res.setHeader("Access-Control-Allow-Credentials", "true");

        if (req.method === "OPTIONS") {
            res.status(200).end();
        }
return await fn(req, res);
    };

const connectDB = (fn: NextApiHandler) =>
    async (req: NextApiRequest, res: NextApiResponse) => {
        await dbConnect();
        return await fn(req, res);
    };

export default connectDB(allowCors(handler));
```

Hicimos tres cambios en el archivo. Primero, importamos la función dbConnect desde el middleware; luego creamos un nuevo wrapper similar a la función allowCors y lo usamos para asegurarnos de que cada llamada a la API se conecte a la base de datos.

Pudimos hacerlo de manera segura porque implementamos dbConnect para forzar solo una conexión a la base de datos al mismo tiempo. Finalmente, envolvimos el handler con el nuevo wrapper y lo exportamos como la opción predeterminada.

Agregar servicios a los resolvers de GraphQL

Ahora es el momento de agregar los servicios a los resolvers. Ya aprendiste que los resolvers de consulta implementan la lectura de datos, mientras que los resolvers de mutación implementan la creación, actualización y eliminación de datos.
Allí también definimos dos resolvers: uno para devolver un objeto meteorológico para un código postal dado y otro para actualizar los datos meteorológicos de una ubicación. Ahora agregaremos los servicios findByZip y updateByZip, que creamos en la lectura mongodb y mongoose a los resolvers. En lugar de las implementaciones ingenuas con el objeto de datos estáticos, modificaremos los resolvers para consultar y actualizar los documentos de MongoDB a través de los servicios.

El listado siguiente muestra el código modificado para el archivo graphql/resolvers.ts en el que refactorizamos estos dos resolvers.

```
import {WeatherInterface} from "../mongoose/weather/interface";
import {findByZip, updateByZip} from "../mongoose/weather/services";

export const resolvers = {
    Query: {
        weather: async (_: any, param: WeatherInterface) => {
            let data = await findByZip(param.zip);
            return [data];
        },
    },
    Mutation: {
        weather: async (_: any, param: {data: WeatherInterface}) => {
            await updateByZip(param.data.zip, param.data);
            let data = await findByZip(param.data.zip);
            return [data];
        },},
};
```

Reemplazamos la funcionalidad ingenua de array.filter con los servicios adecuados. Para consultar los datos, utilizamos el servicio findByZip y le pasamos la variable zip del payload de la solicitud y luego devolvemos los datos de resultado envueltos en un array. Para la mutación, utilizamos el servicio updateByZip.
Según la definición del tipo, la mutación weather devuelve el conjunto de datos actualizado. Para hacerlo, consultamos nuevamente el documento modificado con el servicio findByZip y devolvemos el resultado como un elemento de array.

Visita el sandbox de GraphQL en http://localhost:3000/api/graphql y juega con los puntos finales de la API para leer y actualizar documentos desde la base de datos MongoDB.

Ejercicios

Explicación de componentes:

Pregunta: ¿Cuál es el papel de typeDefs y resolvers en una API GraphQL?

Ventajas de usar servicios:

Pregunta: ¿Cuáles son las ventajas de separar la lógica de acceso a datos en servicios independientes en lugar de manejarla directamente en los resolvers?

Funcionamiento de Middleware:

Pregunta: ¿Cómo ayuda el middleware dbConnect a manejar las conexiones a la base de datos en una aplicación Next.js?

Configuración de la conexión a MongoDB:

Tarea: Crea un archivo db-connect.ts en la carpeta middleware que establezca una conexión a MongoDB utilizando Mongoose. Asegúrate de manejar correctamente los errores y evitar múltiples conexiones.

Refactorización de Resolvers:

Tarea: Modifica los resolvers en graphql/resolvers.ts para que utilicen los servicios findByZip y updateByZip en lugar de acceder directamente a datos estáticos.

Implementación de una nueva mutación:

Tarea: Añade una nueva mutación deleteWeather que permita eliminar un registro meteorológico por código postal. Implementa tanto el resolver como el servicio correspondiente.

Pruebas en GraphQL Sandbox:

Tarea: Utiliza GraphQL Sandbox en http://localhost:3000/api/graphql para realizar las siguientes acciones:

- Consultar datos meteorológicos por código postal.
- Actualizar los datos meteorológicos de una ubicación específica.
- Eliminar un registro meteorológico.

Gestión de CORS:

Tarea: Modifica las configuraciones de CORS en el archivo api/graphql.ts para restringir el acceso a orígenes específicos en lugar de permitir cualquier origen (*).

Optimización de conexiones:

Tarea: Investiga y aplica técnicas para optimizar las conexiones a la base de datos en entornos de producción, como el uso de variables de entorno para gestionar las cadenas de conexión.