**UNIVERSIDAD TÉCNICA NACIONAL**

**INGENIERÍA EN TECNOLOGÍAS DE INFORMACIÓN**

**SEDE DEL PACÍFICO**

**GUÍA DE ACTIVIDAD PRÁCTICA Y/O LABORATORIO**

**Curso:**

ITI

-

912

–

Administració

n

de

Bases

de Datos

Avanzada

Puntos

100

**Profesor:**

Jorge Ruiz

(

york

)

Valor

%

5

%



**Fecha Entrega:** 18/Noviembre/2024 **Tiempo:** 1 Semana

|  |  |
| --- | --- |
| • | **Objetivos de la actividad.** |

o Evaluar el uso de tecnologías para implementar GraphQL contra varios motores de bases de datos, como por ejemplo Microsoft SQL-Server, MySQL, Oracle o MongoDB.

|  |  |
| --- | --- |
| • | **Instrucciones de la práctica.** |

**Laboratorio GraphQL:**

**Observaciones:**

* Lea cuidadosamente cada uno de los problemas planteados y en caso de duda, puede utilizar el chat dispuesto para tal fin, que se encuentra al inicio del curso en la plataforma Moodle del Campus Virtual. U otra que indique el profesor de turno.

* Debe entregar un documento PDF tanto con los problemas planteados, así como con sus correspondientes respuestas. Debe agregar pantallazos de los procesos realizados, los scripts que se están ejecutando, planes de ejecución, estadísticas.

* Considere además la implementación de alguna base de datos sencilla para elaboración de su demo.

**Problema:**

En días pasados su profesor realizó una demostración contra datos a memoria y posteriormente los mismos datos, pero contra el motor de MongoDB, sobre el uso de alguna estrategia de implementación de GraphQL.

Las herramientas utilizadas en dicho momento fueron:

* Apollo Server
* Graphql
* Node.JS
* JavaScript como lenguaje de programación.

**Que se solicita:**

1. Investigar que otras alternativas existen en mercado para el desarrollo de GraphQL.

“La [compatibilidad con el lenguaje](https://graphql.org/code/) de GraphQL es amplia, lo que le permite elegir el lenguaje que prefiera para compilar aplicaciones graphQL. GraphQL también es compatible con muchas [bibliotecas cliente](https://graphql.org/code/#graphql-clients) y [bibliotecas de servidores](https://graphql.org/code/#graphql-servers).” (Microsoft, 2024, párr. 4)

Con NodeJS además de Apolo Server se puede integrar con YOGA para realizar las consultas y mutaciones desde el navegador.

Referencia

Microsoft. (2024) *¿Qué es GraphQL para Azure?* [WEB]

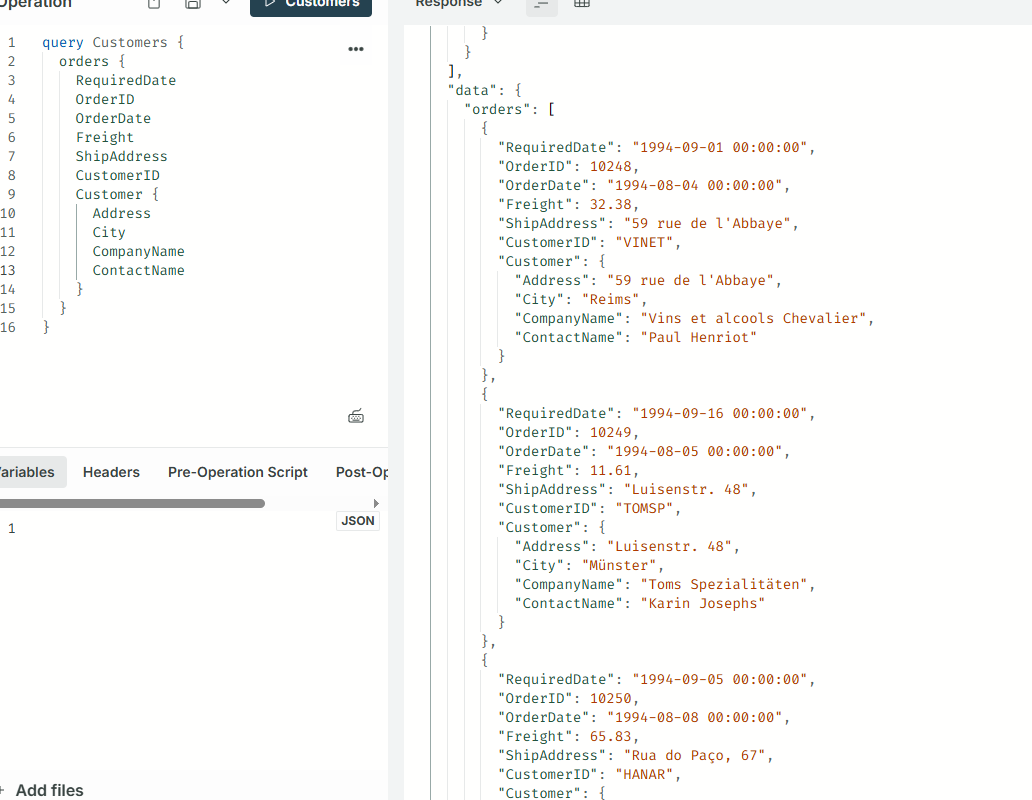
<https://learn.microsoft.com/es-es/azure/developer/javascript/graphql-developer-guide>

1. Sobre que motores de datos pueden aplicarse y si solo son exclusivos a un motor en particular.
2. Desarrollar una demostración para la implementación de la herramienta consumiendo datos, de forma que incluya, los procesos básicos conocidos:
   1. Consultar los datos totales de una tabla.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* 1. Consultar seleccionando campos específicos de una tabla.
  2. Consultar aplicando filtros sobre una tabla.
  3. Consultar aplicando agregaciones o joins entre tablas.



* 1. Insertar, modificar y eliminar datos.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

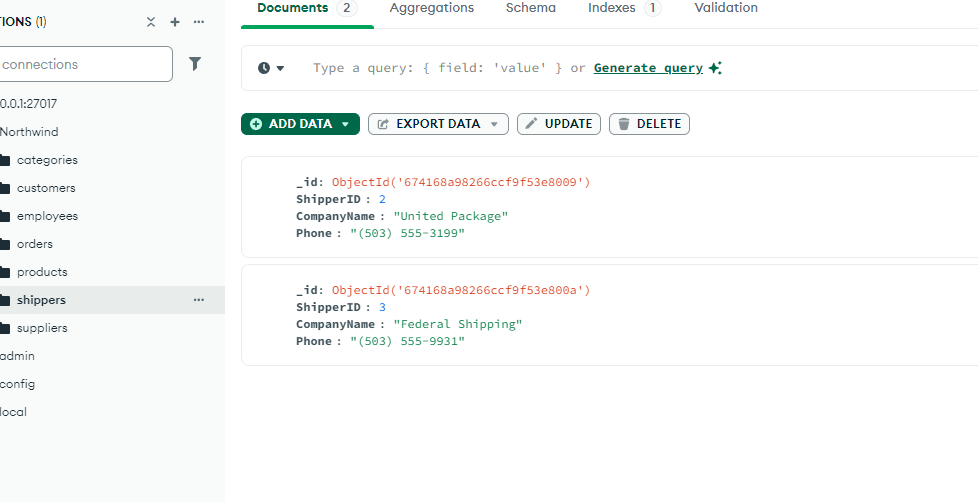
Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente



1. Debe documentar el proceso de instalación de herramientas y de la ejecución de cada proceso de su demo.

Step 01 - Update db

- Open <root\_project>/data/\_db.js and writes the next data from Examen\_db.txt file, after collections categories, products and suppliers:

let customers = [

    {

        CustomerID : "ALFKI",

        CompanyName : "Alfreds Futterkiste",

        ContactName : "Maria Anders",

        ContactTitle :  "Sales Representative",

        Country : "Germany",

        City : "Berlin",

        Address : "Obere Str. 57"

    },

    {

        CustomerID : "ANATR",

        CompanyName : "Ana Trujillo Emparedados y helados",

        ContactName : "Ana Trujillo",

        ContactTitle :  "Owner",

        Country : "Mexico",

        City : "México D.F.",

        Address : "Avda. de la Constitución 2222"

    },

    {

        CustomerID : "ANTON",

        CompanyName : "Antonio Moreno Taquería",

        ContactName : "Antonio Moreno",

        ContactTitle :  "Owner",

        Country : "Mexico",

        City : "México D.F.",

        Address : "Mataderos  2312"

    },

.

.

}

]

let employees = [

    {

        EmployeeID : 1,

        LastName : "Davolio",

        FirstName : "Nancy",

        Title : "Sales Representative",

        TitleOfCourtesy : "Ms.",

        BirthDate : "1948-12-08 00:00:00",

        HireDate : "1992-05-01 00:00:00",

        Country : "USA",

        City : "Seattle",

        Address : "507 - 20th Ave. E. Apt. 2A"

    },

    {

        EmployeeID : 2,

        LastName : "Fuller",

        FirstName : "Andrew",

        Title : "Vice President, Sales",

        TitleOfCourtesy : "Dr.",

        BirthDate : "1952-02-19 00:00:00",

        HireDate : "1992-08-14 00:00:00",

        Country : "USA",

        City : "Tacoma",

        Address : "908 W. Capital Way"

    },

    {

        EmployeeID : 3,

        LastName : "Leverling",

        FirstName : "Janet",

        Title : "Sales Representative",

        TitleOfCourtesy : "Ms.",

        BirthDate : "1963-08-30 00:00:00",

        HireDate : "1992-04-01 00:00:00",

        Country : "USA",

        City : "Kirkland",

        Address : "722 Moss Bay Blvd."

    },

.

.

}

]

let orderDetails =

[

    {

        OrderID : 10248,

        ProductID :11,

        UnitPrice : 14.0000,

        Quantity : 12,

        Discount : 0.00

    },

    {

        OrderID : 10248,

        ProductID :42,

        UnitPrice : 9.8000,

        Quantity : 10,

        Discount : 0.00

    },

    {

        OrderID : 10248,

        ProductID :72,

        UnitPrice : 34.8000,

        Quantity : 5,

        Discount : 0.00

    },

    {

        OrderID : 10249,

        ProductID :14,

        UnitPrice : 18.6000,

        Quantity : 9,

        Discount : 0.00

    },

.

.

.

}

]

let orders =

[

    {

        OrderID : 10248,

        CustomerID : "VINET",

        EmployeeID : 5,

        OrderDate : "1994-08-04 00:00:00",

        RequiredDate :"1994-09-01 00:00:00",

        ShippedDate : "1994-08-16 00:00:00",

        ShipVia : 3,

        Freight : 32.3800,

        ShipName : "Vins et alcools Chevalier",

        ShipCountry : "France",

        ShipCity : "Reims",

        ShipAddress : "59 rue de l'Abbaye",

        ShipPostalCode : "51100"

    },

    {

        OrderID : 10249,

        CustomerID : "TOMSP",

        EmployeeID : 6,

        OrderDate : "1994-08-05 00:00:00",

        RequiredDate :"1994-09-16 00:00:00",

        ShippedDate : "1994-08-10 00:00:00",

        ShipVia : 1,

        Freight : 11.6100,

        ShipName : "Toms Spezialitäten",

        ShipCountry : "Germany",

        ShipCity : "Münster",

        ShipAddress : "Luisenstr. 48",

        ShipPostalCode : "44087"

    },

.

.

.

}

]

shippers =

[

    {

        ShipperID : 1,

        CompanyName : "Speedy Express",

        Phone : "(503) 555-9831"

    },

    {

        ShipperID : 2,

        CompanyName : "United Package",

        Phone : "(503) 555-3199"

    },

    {

        ShipperID : 3,

        CompanyName : "Federal Shipping",

        Phone : "(503) 555-9931"

    }

]

Step 02 - Update schemas

- Open <root\_project>/data/schema.js

export const typeDefs = `#graphql

    type Categories {

        CategoryID: Int!,

        CategoryName: String!,

        Description: String!,

        Products: [Products]!

    }

    type Suppliers{

        SupplierID: Int!,

        CompanyName: String!,

        ContactName: String!,

        ContactTitle: String!,

        Country: String!,

        City: String!,

        Address: String!,

        Products: [Products]!

    }

    type Products {

        ProductID: Int!,

        ProductName: String!,

        SupplierID: ID!,

        Supplier: Suppliers!,

        CategoryID: ID!,

        Category: Categories!,

        QuantityPerUnit: String!,

        UnitPrice: Float!,

        UnitsInStock: Int!,

        ReorderLevel: Int!

    }

    type Customers {

        CustomerID: String!,

        CompanyName: String!,

        ContactName: String!,

        ContactTitle: String!,

        Country: String!,

        City: String!,

        Address: String!,

        Orders: [Orders]!

    }

    type Employees {

        EmployeeID: Int!,

        LastName: String!,

        FirstName: String!,

        Title: String!,

        TitleOfCourtesy: String!,

        BirthDate: String!,

        HireDate: String!,

        Country: String!,

        City: String!,

        Address: String!,

        Orders: [Orders]!

    }

    type Orders {

        OrderID: Int!,

        CustomerID: ID!,

        Customer: Customers!,

        EmployeeID: ID!,

        Employee: Employees!,

        OrderDate: String!,

        RequiredDate: String!,

        ShippedDate: String!,

        ShipVia: Int!,

        Freight: Float!,

        ShipName: String!,

        ShipCountry: String!,

        ShipCity: String!,

        ShipAddress: String!,

        ShipPostalCode: String!

    }

    type Shippers {

        ShipperID: Int!,

        CompanyName: String!,

        Phone: String!,

    }

    type Query {

        categories: [Categories]!,

        category(CategoryID: Int!): Categories,

        products: [Products]!,

        product(ProductID: Int!): Products,

        suppliers: [Suppliers]!,

        supplier(SupplierID: Int!): Suppliers,

        customers: [Customers]!,

        customer(CustomerID: String!): Customers,

        employees: [Employees]!,

        employee(EmployeeID: Int!): Employees,

        orders: [Orders]!,

        order(OrderID: Int!): Orders,

        shippers: [Shippers]!,

        shipper(ShipperID: Int!): Shippers

    }

    .

.

.

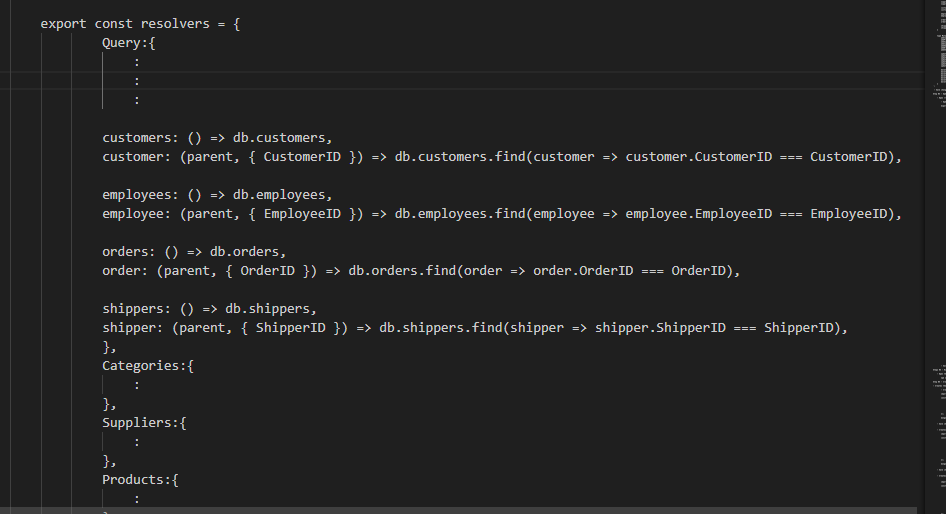
    }

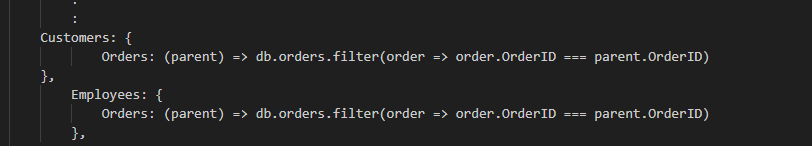
`;

Step 03 - Update resolvers.js

- Open <root\_project>/resolvers.js

- Update const resolvers, go to the end:





Mutation: {

            .

.

.

        addCustomer: (parent, args) => {

            const newCustomer = { ...args };

            db.customers.push(newCustomer);

            return newCustomer;

        },

        addEmployee: (parent, args) => {

            const newEmployee = { ...args };

            db.employees.push(newEmployee);

            return newEmployee;

        },

        addOrder: (parent, args) => {

            const newOrder = { ...args };

            db.orders.push(newOrder);

            return newOrder;

        },

        addShipper: (parent, args) => {

            const newShipper = { ...args };

            db.shippers.push(newShipper);

            return newShipper;

        },

updateCustomer: (parent, args) => {

            const { CustomerID, ...rest } = args;

            const customer = db.customers.find(customer => customer.CustomerID === CustomerID);

            Object.assign(customer, rest);

            return customer;

        },

        updateEmployee: (parent, args) => {

            const { EmployeeID, ...rest } = args;

            const employee = db.employees.find(employee => employee.EmployeeID === EmployeeID);

            Object.assign(employee, rest);

            return employee;

        },

        updateOrder: (parent, args) => {

            const { OrderID, ...rest } = args;

            const order = db.orders.find(order => order.OrderID === OrderID);

            Object.assign(order, rest);

            return order;

        },

        updateShipper: (parent, args) => {

            const { ShipperID, ...rest } = args;

            const shipper = db.shippers.find(shipper => shipper.ShipperID === ShipperID);

            Object.assign(shipper, rest);

            return shipper;

        },

 deleteCustomer: (parent, { CategoryID }) => {

            const customer = db.customers.find(customer => customer.CustomerID === CustomerID);

            db.categories = db.customers.filter(customer => customer.CustomerID !== CustomerID);

            return customer;

        },

        deleteEmployee: (parent, { ProductID }) => {

            const employee = db.employees.find(employee => employee.EmployeeID === EmployeeID);

            db.employees = db.employees.filter(employee => employee.EmployeeID !== EmployeeID);

            return employee;

        },

        deleteOrder: (parent, { OrderID }) => {

            const order = db.orders.find(order => order.OrderID === OrderID);

            db.orders = db.orders.filter(order => order.OrderID !== OrderID);

            return order;

        },

        deleteShipper: (parent, { ShipperID }) => {

            const shipper = db.shippers.find(shipper => shipper.ShipperID === ShipperID);

            db.shippers = db.shippers.filter(shipper => shipper.ShipperID !== ShipperID);

            return shipper;

        }

}

}

Stepo 04 - Run application

- Open the terminal window in your favorite IDE and execute the next command:

npm start

Step 05 - Create the MongoDB models

- Creates the new models for each data object into <root\_project>/models folder

- Creates mdl\_Customers.js and writes:

import mongoose from "mongoose";

const customerSchema = new mongoose.Schema({

    CustomerID: { type: String, required: true },

    CompanyName: { type: String, required: true },

    ContactName: { type: String, required: true },

    ContactTitle: { type: String, required: true },

    Country: { type: String, required: true },

    City: { type: String, required: true },

    Address: { type: String, required: true }

});

mongoose.model('Customers', customerSchema);

- Save changes

- Creates mdl\_Employees.js and writes:

import mongoose from "mongoose";

const employeeSchema = new mongoose.Schema({

    EmployeeID: { type: Number, required: true },

    LastName: { type: String, required: true },

    FirstName: { type: String, required: true },

    Title: { type: String, required: true },

    TitleOfCourtesy: { type: String, required: true },

    BirthDate: { type: String, required: true },

    HireDate: { type: String, required: true },

    Country: { type: String, required: true },

    City: { type: String, required: true },

    Address: { type: String, required: true }

});

mongoose.model('Employees', employeeSchema);

- Save changes

- Creates mdl\_Orders.js and writes:

import mongoose from "mongoose";

const orderSchema = new mongoose.Schema({

    OrderID: { type: Number, required: true },

    CustomerID: { type: String, required: true },

    EmployeeID: { type: Number, required: true },

    OrderDate: { type: String, required: true },

    RequiredDate: { type: String, required: true },

    ShippedDate: { type: String, required: true },

    ShipVia: { type: Number, required: true },

    Freight: { type: Number, required: true },

    ShipName: { type: String, required: true },

    ShipCountry: { type: String, required: true },

    ShipCity: { type: String, required: true },

    ShipAddress: { type: String, required: true },

    ShipPostalCode: { type: String, required: true }

});

mongoose.model('Orders', orderSchema);

- Save changes

- Creates mdl\_OrderDetails.js and writes:

import mongoose from "mongoose";

const orderDetailSchema = new mongoose.Schema({

    OrderID : { type: Number, required: true },

    ProductID : { type: Number, required: true },

    UnitPrice: { type: Number, required: true },

    Quantity : { type: Number, required: true },

    Discount: { type: Number, required: true }

});

mongoose.model('OrderDetails', orderDetailSchema);

- Save changes

- Creates mdl\_Shippers.js and writes:

import mongoose from "mongoose";

const shipperSchema = new mongoose.Schema({

    ShipperID : { type: Number, required: true },

    CompanyName: { type: String, required: true },

    Phone: { type: String, required: true }

});

mongoose.model('Shippers', shipperSchema);

Step 06 - Update the new resolvers using MongoDB

- Create new file: "<root\_project>/data/resolversMongo.js" and writes

import mongoose from "mongoose";

const Customers = mongoose.model('Customers');

const Employees = mongoose.model('Employees');

const Orders = mongoose.model('Orders');

const Shippers = mongoose.model('Shippers');

export const resolvers = {

    Query: {

        .

.

.

        customers: async () => await Customers.find(),

        customer: async (parent, { CustomerID }) => await Customers.findOne({CustomerID: CustomerID}),

        employees: async () => await Employees.find(),

        employee:  async (parent, { EmployeeID }) => await Employees.findOne({EmployeeID: EmployeeID}),

        orders: async () => await Orders.find(),

        order: async (parent, { OrderID }) => await Orders.findOne({OrderID: OrderID}),

        shippers: async () => await Shippers.find(),

        shipper: async (parent, { ShipperID }) => await Shippers.findOne({ShipperID: ShipperID})

    },

    .

.

.

    Customers: {

        Orders: async (parent) => await Orders.find({ OrderID: parent.OrderID })

    },

    Employees: {

        Orders: async (parent) => await Orders.find({ OrderID: parent.OrderID })

    },

    Orders: {

        Customer: async (parent) => await Customers.findOne({CustomerID: parent.CustomerID}),

        Employee: async (parent) => await Employees.findOne({EmployeeID: parent.EmployeeID})

    },

    Mutation: {

.

.

.

        addCustomer: async (parent, args) => {

            const newCustomer = new Customers(args);

                return newCustomer.save();

        },

        addEmployee: async (parent, args) => {

            const newEmployee = new Employees(args);

                return newEmployee.save();

        },

        addOrder: async (parent, args) => {

            const newOrder = new Orders(args);

                return newOrder.save();

        },

        addShipper: async (parent, args) => {

            const newShipper = new Shippers(args);

                return newShipper.save();

        },

.

.

.

        updateCustomer: async (parent, args) => {

            const { CustomerID, ...rest } = args;

                return Customers.findOneAndUpdate({CustomerID: CustomerID}, rest, { new: true });

        },

        updateEmployee: async (parent, args) => {

            const { EmployeeID, ...rest } = args;

                return Employees.findOneAndUpdate({EmployeeID: EmployeeID}, rest, { new: true });

        },

        updateOrder: async (parent, args) => {

            const { OrderID, ...rest } = args;

                return Orders.findOneAndUpdate({OrderID: OrderID}, rest, { new: true });

        },

        updateShipper: async (parent, args) => {

            const { ShipperID, ...rest } = args;

                return Shippers.findOneAndUpdate({ShipperID: ShipperID}, rest, { new: true });

        },

.

.

.

        deleteCustomer: async (parent, { CustomerID }) => {

            return Customers.findOneAndDelete({CustomerID: CustomerID});

        },

        deleteEmployee: async (parent, { EmployeeID }) => {

            return Employees.findOneAndDelete({EmployeeID: EmployeeID});

        },

        deleteOrder: async (parent, { OrderID }) => {

            return Orders.findOneAndDelete({OrderID: OrderID});

        },

        deleteShipper: async (parent, { ShipperID }) => {

            return Shippers.findOneAndDelete({ShipperID: ShipperID});

        }

    }

}

Step 07 - Update <root\_project>/index.js

- Remember:

- Change the data connection with mongoDB

- Run the MongoDB instance

- Open <root\_project>/index.js and delete all code and writes:

// Call ApolloServer and startStandaloneServer

import { ApolloServer } from '@apollo/server';

import { startStandaloneServer } from '@apollo/server/standalone';

//  Connection to MongoDB

import mongoose from "mongoose";

mongoose.connect('mongodb://localhost:27017/', {dbName: 'Northwind'})

    .then(r  => console.log('Connected to MongoDB'));

// Load models

import './models/mdl\_Customers.js';

import './models/mdl\_Employees.js';

import './models/mdl\_Orders.js';

import './models/mdl\_Shippers.js';

// Load data resources

import { typeDefs } from './data/schema\_db.js';

// Mongo DB

 import { resolvers } from './data/resolversMongo.js';

// Apolo Server

//import { resolvers } from './data/resolvers.js';

// Publish data resources

const server = new ApolloServer({

    typeDefs,

    resolvers

},);

// Run Apollo Server

const {url} = await startStandaloneServer(server,{

    listen: { port: 4005 }

});

1. Realizar demostración al resto de compañeros

|  |  |
| --- | --- |
| • | **Evaluación.** |

|  |  |  |
| --- | --- | --- |
| **Ítem** | **Concepto** | **Puntos** |
| 1 | Documento en formato pdf, sobre la instalación de herramientas y otros requerimientos, así como corrida de la demostración, incluye información general de las herramientas utilizadas. | 15 |
| 2 | Guía con los pasos para desarrollar el demos e instalación de requerimientos | 15 |
| 3 | Demostración del uso de la herramienta utilizada para usar GraphQL ante el grupo de clase | 10 |
|  | **Total de Puntos** | **40** |

CRUD de NorthWind En Graphql y Mongo DB. El archivo Demo.txt contiene los pasos de desarrollo

<https://github.com/Ach1c0d1a/Northwind_Graphql.git>