**CURSO REACTJS AVANZADO - FULL STACK REACT, GRAPHQL, APOLLO, MONGODB**

**Proyecto completo con las siguientes tecnologías:**

(Video 1 - Seccion 1)

ReactJs

GraphQL

Apollo Cliente y Apollo Server

MongoDB

**Que es GraphQL?**

(Video 2 - Seccion 2)

GraphQL = Graph Query Language

Creador: Facebook - Se creo como necesidad para consultar datos por los inconvenientes que tenían con las otras tecnologías al obtener las entradas o estados de los amigos de un perfil. Es muy útil para manejar grandes volúmenes de datos.

Desarrollado en el 2012 y se convirtio Open Sources en el 2015.

Independiente de cualquier lenguaje, Base de datos o FrameWork.

Puedes usarlo asi ya tengas un proyecto hecho.

**GraphQL y SQL**

(Video 3 - Seccion 2)

Similar a SQL en el servicio que presta mas no en su sintaxis ni en funcionalidad

Se usa para hacer la conexión entre UI y Base de Datos. Se pueden realizar consultas y leer resultados.

Es seguro por que hay separacion entre el cliente y la base de datos.

Necesario tener instalado GraphQL en el servidor.

Mucha compatibilidad con versiones anteriores.

GraphiQL**:** Herramienta visual de GraphQL ejecutada en el navegador web para realizar las consultas

**GraphQL vs REST API´s:**

(Video 4 - Seccion 2)

- La gran diferencia esta en los End Points que en REST son fijos y te mando siempre los mismos y el total de los datos esto genera mas trafico de datos y una app mas pesada, tendrias que hacer muchos end points para traer solo los que necesites. En GraphQL con las QUERY puedes traer solo los datos que necesites.

Las consulta es variable gracias al Schema

EL CRUD de GraphQL = Query - Mutation

**QUERY - MUTATION (Terminologia)**

(Video 5 - Seccion 2)

Query = Leer

Mutation = Crear, Modificar y Eliminar

Son universales: Funcionan con cualquier Lenguaje, base de datos o Framework

CRUD: El Query de GraphQL es la R del CRUD. Osea Leer

CRUD: El mutation de GraphQL es la C, U y D del CRUD. Osea Crear, Actualizar y Eliminar.

QUERY - Ejemplos:

*query {*

*nombredelQuery{*

*Id*

*Nombre*

*Precio*

*stock*

*}*

*}*

QUERY - Ejemplos:

*mutation eliminarProducto($id : ID)*

*{*

*eliminarProducto(id: $id)*

*}*

**RESOLVER - SCHEMA (Terminologia)**

(Video 6 - Seccion 2)

Resolver:Los Resolver se utilizan para gestionar las bases de datos con un backend

- Se puede usar en cualquier lenguaje pero es especifico del lenguaje que se use y las interaciones a la base de datos. (EJ: Si se usa en JavaScript hay que escribirlo en JavaScript)

Resolver1:JavaScript y SQL

Resolver2:JavaScript y MongoDB

Ejemplo resolver hecho en MongoDB

getCliente : (root, {id}) => {

Return new Promise((resolver, object) => {

Clientes.findById(

})

}

Schema:

-Es lo que describe tus tipos de objeto, Queries y datos de tu aplicacion.

-El Schema en GraphQL utiliza un typing en el que le defines si un campo sera de tipo String, Int, Boolean u otro tipo de dato.

- El Schema y el resolver estan muy relacionados, El Shema define la forma de los datos mientras que el resolver la comunicacion con el lenguaje del servidor y la base de datos.

Ejemplo:

type Cliente {

Id: ID

nombre: String

apellido: String

empresa: String

emails: [Email] // Se usa [] para guardar multiples datos (Varios emails)

edad: Int

}

type Email {

email: String

}

La estructura del Schema debe de ser similar o iguales a la de la base de datos.

Como trabajan en conjunto el Schema y el Resolver

- En el Schema (Creo una consulta que se llama getCliente)

- Este Schema es esclusivo de GraphQL

type Query {

getCliente(id: ID) : Cliente

}

- En el Resolver(Uso el mismo nombre de la consulta y le digo como traer los datos)

- Este Resolver es esclusivo del lenguaje de programacion en el server y la base de datos que se use.

getCliente : (root, {id}) => {

return new Promise((resolve, object) => {

Clientes.findById (id, (error, cliente)) => {

if (error) rejects (error)

else resolve (cliente)

});

});

},

Previa-mente se ha tenido que crear el type cliente

type Cliente {

id: ID

nombre: String

apellido: String

empresa: String

emails: [Email]

edad: Int

}

**QUERIES Y QUERIES CON ARGUMENTOS (Consultas con GraphQL)**

(Video 7 - Seccion 3)

**-** En el curso se usaran dos GraphiQL de internet. La de Github y la de GraphQL Hub

Github GraphiQL url

<https://developer.github.com/v4/explorer/>

GraphQL Hub url

<https://www.graphqlhub.com/>

**Ej Consulta 1 en GraphiQL de Github:**

viewer: muestra los datos del usuario logeado en github

query {

viewer {

name

company

createdAt

}

}

**Queries con Argumentos y Donde Encontrar los argumentos**

(Video 8 - Seccion 3)

Ej Consulta en Graph Hub con argumentos:

Ej1:

query {

twitter {

Search (q: “GraphQL”) {

text

retweet\_count

}

}

}

Ej2:

query {

repositoryOwner (login: "andresganc"){

id

url

resourcePath

}

}

Ej3:

{

hn{

topStories(limit: 2){

id

time

text

url

title

}

}

}

**Argumentos obligatorios y como identificarlos**

(Video 9 - Seccion 3)

Se identifican con una ! y se debe de buscar en la documentacion de la API

Ej:

FIELDS

repository(owner: String!, name: String!)

{

repository(owner:"google" name:"fonts"){

id

description

}

}

**Que es el Schema de GraphQL**

(Video 10 - Seccion 3)

- El Schema de GraphQL son como las tablas en SQL.

**-** Tambien muestra los objetos y los tipos de datos

- Y lo que corresponderia a las columnas

Todo es el Schema

{

hn{ // Es como la base de datos en SQL

topStories(limit: 2){ // Es como la tabla en SQL

Id // Son como las columnas en SQL

time

text

url

title

}

}

}

**Aliases en los Queries**

(Video 11 - Seccion 3)

Sin Aliases (Genera un error de duplicidad de argumentos search)

{

twitter{

search(q:"ReactJS")

{

id

created\_at

text

user {

id

}

}

search(q:"GraphQL")

{

id

created\_at

text

user {

id

}

}

}

}

Con Aliases (No genera error)

{

twitter{

AliasesReacJS: search(q:"ReactJS")

{

id

created\_at

text

user {

id

}

}

AliasesGraphQL: search(q:"GraphQL")

{

id

created\_at

text

user {

id

}

}

}

}

**Consulta con Fragments**

(Video 12 - Seccion 3)

- Los fragments se usan cuando tenemos muchos schemas con consultas parecidas y necesitamos agregar consultas a todas. para cambiar schemas repetitivos de varias partes sin necesidad de repetir y escribir en cada schema.}

- Las modificaciones se hacen en una sola y se ven reflejadas en todas las que queramos

Sin Fragments

{

twitter{

AliasesReacJS: search(q:"ReactJS")

{

id

created\_at

text

user {

id

}

}

AliasesGraphQL: search(q:"GraphQL")

{

id

created\_at

text

user {

id

}

AliasesJavaScript: search(q:"JavaScript")

{

id

created\_at

text

user {

id

}

}

}

}

Con Fragments

{

twitter{

AliasesReacJS: search(q:"ReactJS")

{

...camposTweet

}

AliasesGraphQL: search(q:"GraphQL")

{

...camposTweet

}

AliasesJavaScript: search(q:"JavaScript")

{

...camposTweet

# Puedo usar un campo que no se comparta en cualquiera

}

}

}

fragment camposTweet on Tweet {

id

created\_at

text

user {

id

}

}

**Campos anidados**

(Video 13 - Seccion 3)

{

twitter{

AliasesReacJS: search(q:"ReactJS")

{

...camposTweet

}

AliasesGraphQL: search(q:"GraphQL")

{

...camposTweet

}

AliasesJavaScript: search(q:"JavaScript")

{

...camposTweet

# Puedo usar un campo que no se comparta en cualquiera

}

}

}

fragment camposTweet on Tweet {

id

created\_at

text

user {

id

name

screen\_name

}

}

**Conexiones en GraphQL**

(Video 14 - Seccion 3)

Es cuando un objeto tiene varios nodos. (De uno a muchos)

Se llama por medio de los objetos conection

En este caso edges es el objeto conection

{

viewer{

id

name

repositories(last: 5){

edges{

node{

id

name

}

}

}

}

}

**Multiples campos anidados**

(Video 15 - Seccion 3)

{

twitter{

ReactTweets: search(q: "ReactJS"){

id

text

retweet\_count

user{

name

screen\_name

description

tweets\_count

url

tweets(limit: 5){

id

retweet\_count

}

}

}

}

}

**Paginando resultados en GraphQL (Offsets y Limits)**

(Video 16 - Seccion 3)

- Con el offset voy paginando

{

hn{

topStories(limit: 10, offset:1){

id

text

title

url

}

}

}

**Querys con Nombres**

(Video 17 - Seccion 3)

query Cosulta1\_NC{

twitter{

search(q: "JavaScript"){

id

text

}

}

}

**Pasar variables a las consultas**

(Video 18 - Seccion 3)

query Cosulta1\_NC($tweet: String!){

twitter{

search(q: $tweet){

id

text

}

}

}

QUERY VARIABLES

{

"tweet": "Java"

}

**Pasar multiples variables a las consultas**

(Video 19 - Seccion 3)

query Cosulta1\_NC($tweet: String!, $total: Int){

twitter{

search(q: $tweet, count: $total){

id

text

}

}

}

QUERY VARIABLES

{

"tweet": "Java",

"total": 5

}

**Mutations: Que son y como utilizarlos**

(Video 20 - Seccion 3)

- Para obtener los datos de una API con GraphQL hemos utilizado Query

- Si deseas Editar o Eliminar un registro, Debertas utilizar un mutation

- Los datos se pasan como Payload

- GraphQL hace los cambios

La API es la que define que cambios se puedenrealizar y que parametros tomar, Igual que una base de datos o aplicacion web.

mutation NuevoComentario($input: SetValueForKeyInput!){

keyValue\_setValue(input: $input){

clientMutationId

item{

id

value

}

}

}

query LeerComentario{

keyValue{

getValue(id:"123456"){

id

value

}

}

}

QUERY VARIABLES

{

"input":{

"id": "123456",

"value": "Hola soy NC",

"clientMutationId": "654321"

}

}

**CREANDO UN SERVIDOR - SECCION 4**

**Editor de Codigo**

(Video 21 - Seccion 4

Visual studio Code

Extension

- GraphQL for VSCode

- ES7 React/Redux/GraphQL/React-Native snippets

- GraphQL

- GraphQL Language Support

- Prettier Now

**Instalando el servidor express nodemon y dependencias babel**

(Video 22 - Seccion 4

- Creamos la carpeta del proyecto (No se puede llamar graphql porque crea conflicto con las dependencias que se van a instalar)

- Entramos en consola a la carpeta y Escribimos npm init para generar el package.json

$ npm init

Dependencias a instalar

Primero se escribe en consola npm install --save y el nombre de las dependencias separadas por un espacio.

(save) Ya casi no se usa, Ya en las nuevas versiones de npm se guarda automaticamente

$ npm install --save dependenciaservidor1 dependenciaservidor2 dependenciaservidor3

$ npm install --save-dev dependenciadesarrollo1 dependenciadesarrollo2 dependenciadesarrollo3

**Dependencias del servidor a instalar**

$ npm install --save

**nodemon** : Sirve para que el proyecto recargue automaticamente el servidor al realizar un cambio.

**express** : Servidor web de NodeJS

**Dependencias de desarrollo a instalar**

$ npm install --save-dev

**babel-cli** : Babel es una herramienta que nos permite transformar nuestro código JS de última generación (o con funcionalidades extras) a JS que cualquier navegador o versión de Node.js entienda. (podemos hacer que entienda código de JSX y lo convierta a código JS normal.)

**babel-preset-env** :

**babel-preset-stage-0** :

**GraphiQL y servidor para GraphiQL**

(Video 23 - Seccion 4

Dependencias a instalar

Npm install --save

- express-graphql : Servidor express para GraphQL

- graphql : GraphQL

**PRIMEROS PASOS CON EL SERVIDOR - SECCION 5**

**Agregando los primeros Types y Queries**

(Video 24 - Seccion 5