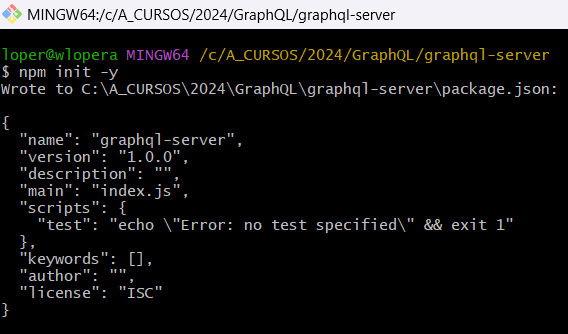
**PROYECTO GRAPHQL-SERVER**

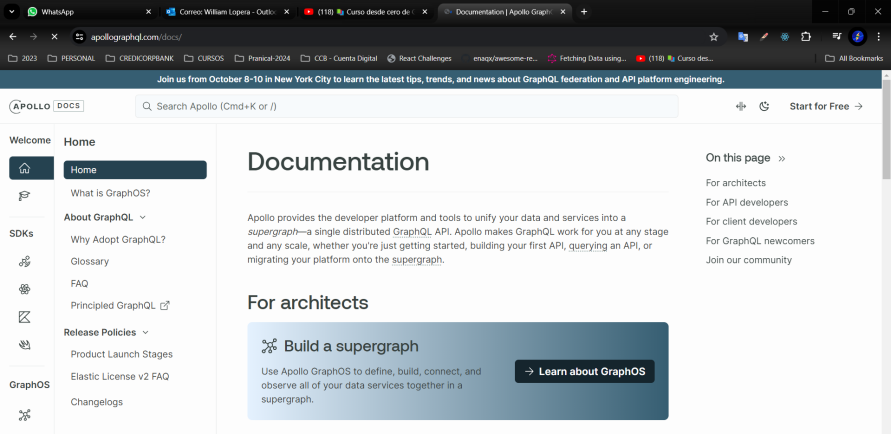
El servidor Graphql es agnóstico al lenguaje o plataforma que lo requiere

Vamos a crear un servidores graphql.

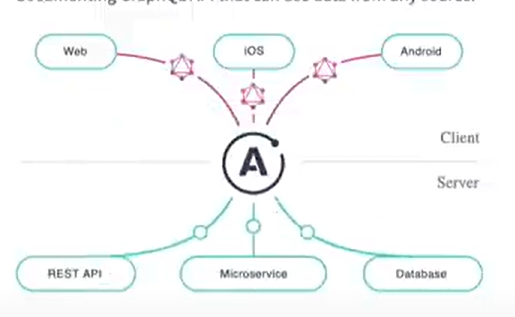
* Crear carpeta ..\GraphQL\graphql-server
* Crear proyecto ..\GraphQL\graphql-server>npm init -y



* Uso de frameword o plataforma Apollo



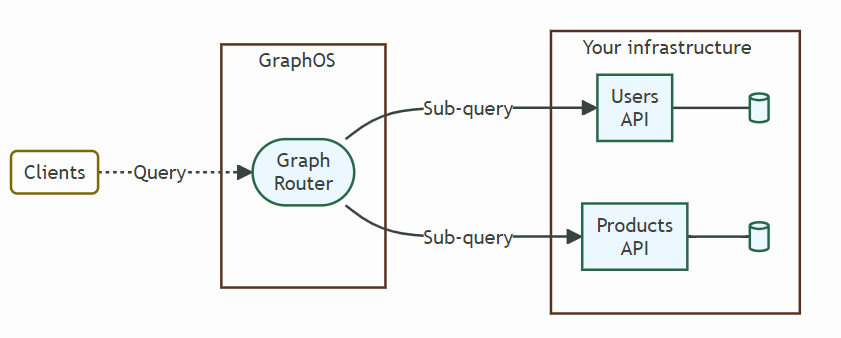
**La plataforma Apollo**



Unifique su infraestructura con GraphQL

Apollo GraphOS es la plataforma para construir, administrar y escalar un supergrafo : una red unificada de los microservicios de su organización y sus fuentes de datos , todos compuestos en una única API distribuida.

Con una consulta al enrutador de su supergrafo , un cliente de aplicación puede obtener datos de cualquier combinación de sus servicios:



Los clientes consultan su supergrafo con un lenguaje poderoso llamado GraphQL , que les permite recuperar exactamente los datos que necesitan, sin sobrecapturas. La arquitectura de su supergrafo no está expuesta: los clientes envían consultas a un único punto final (su enrutador), sin importar qué datos necesiten.

**Apollo Server** es un servidor de código abiertoServidor GraphQL compatible con especificaciones y compatible con cualquier cliente GraphQL , incluido Apollo Client . Es la mejor manera de crear una API GraphQL autodocumentada y lista para producción que pueda usar datos de cualquier fuente.



Puedes utilizar Apollo Server como:

El servidor GraphQL para un subgrafo en un supergrafo federado

Un complemento para cualquier aplicación Node.js nueva o existente; esto incluye aplicaciones que se ejecutan en Express (incluidas las aplicaciones de pila MERN ), AWS LambdaFunciones de Azure, Nubeflare, Fastificar, y más

<https://www.apollographql.com/docs/apollo-server/>

**Apollo Server ofrece:**

Configuración sencilla , para que los desarrolladores de sus clientes puedan comenzar a obtener datos rápidamente

Adopción incremental , que le permite agregar funciones a medida que sean necesarias

Compatibilidad universal con cualquier fuente de datos , cualquier herramienta de compilación y cualquier cliente GraphQL

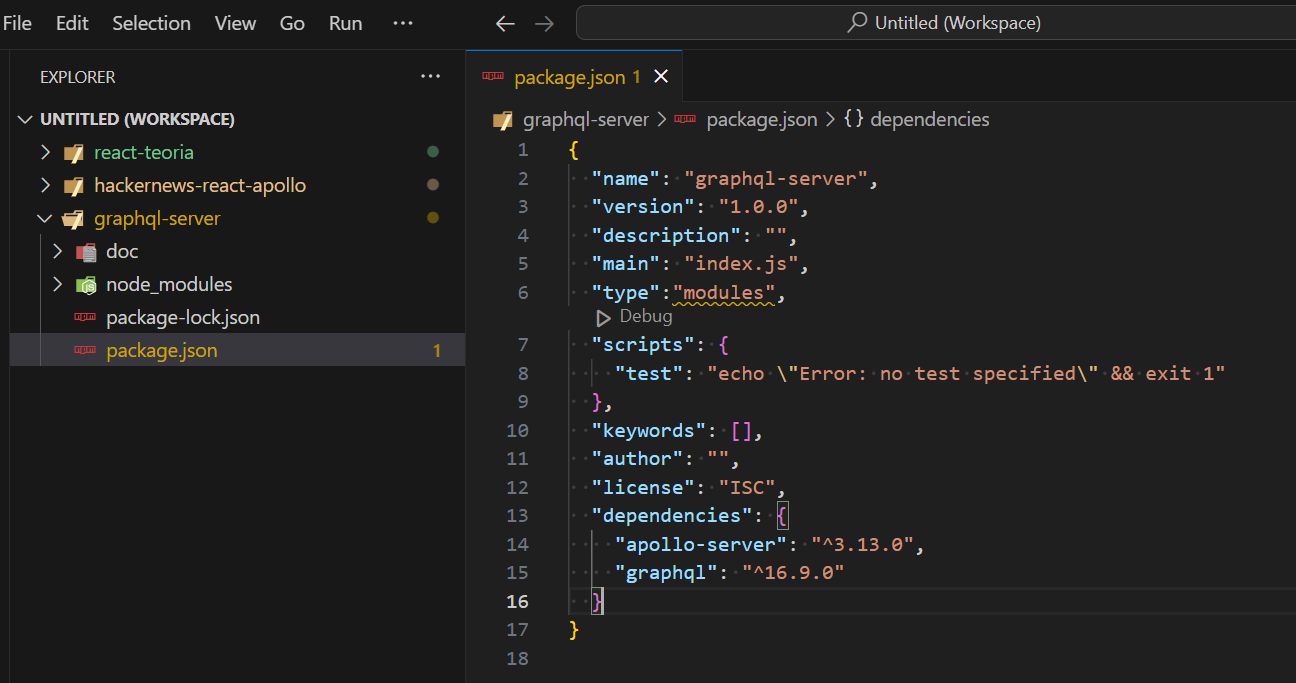
Preparación para producción , lo que le permite ejecutar su gráfico con confianza en producción

**Librerías requeridas**

* + Servidor de Apollo: \_> npm install apollo-server
  + Grafos (Graphql): \_> npm install graphql



* En VSCODE



package.json

{

  "name": "graphql-server",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

**"type":"module",**

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "dependencies": {

    "apollo-server": "^3.13.0",

    "graphql": "^16.9.0"

  }

}

Nota agregar type: module

* Crear un archivos index.js. Servidor node
  + Agregar un JSON de datos (DUMMY) para prueba (persons)
  + Agregar los tipos de definiciones
  + Agregar los resolvedores para las consultas de los datos
  + Agregar el servidor Apollo

import { ApolloServer, gql } from "apollo-server";

// Data DUMMY

const persons = [

  {

    name: "Neymar Junior",

    phone: "034-1234567",

    street: "Calle Frotend",

    city: "Barcelona",

    id: "3d599650-3436-11eb-8b800ba54c431",

  },

  {

    name: "Leonel Messi",

    phone: "044-123456",

    street: "Avenida Fullstack",

    city: "Mataro",

    id: "3d599470-3436-11eb-8b800ba54c431",

  },

  {

    name: "Cristiano Ronaldo",

    street: "Pasaje Testing",

    city: "Ibitza",

    id: "3d599471-3436-11eb-8b800ba54c431",

  },

];

// Tipos de datos y consultas

// ! => Obligatorio

// Query metodos de consulta expuestos

const typeDefinitions = gql`

  type Person {

    name: String!

    phone: String

    street: String!

    city: String!

    id: ID!

  }

  type Query {

    personCount: Int!

    allPersons: [Person]!

  }

`;

// Resolvedores:  Como se obtiene y sacan los datos

const resolvers = {

  Query: {

    personCount: () => persons.length,

    allPersons: () => persons,

  },

};

// Generar srevidors Apollo

const server = new ApolloServer({

  typeDefs: typeDefinitions,

  resolvers,

});

server.listen().then(({ url }) => {

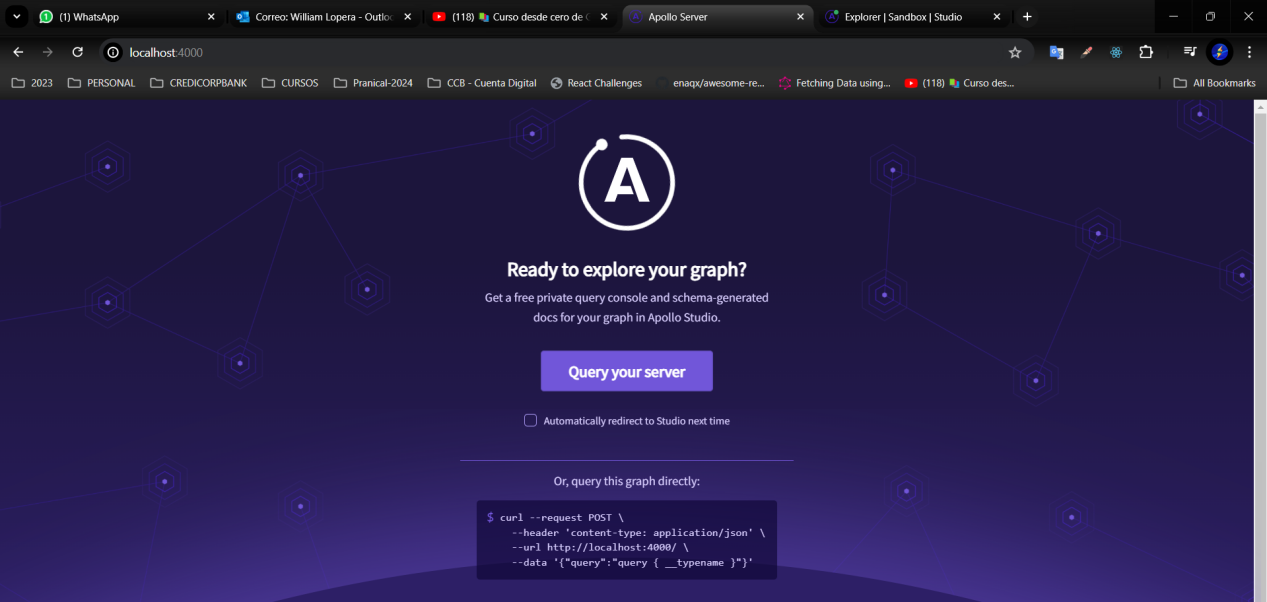
  console.log(`Servidor Listo en ${url}`);

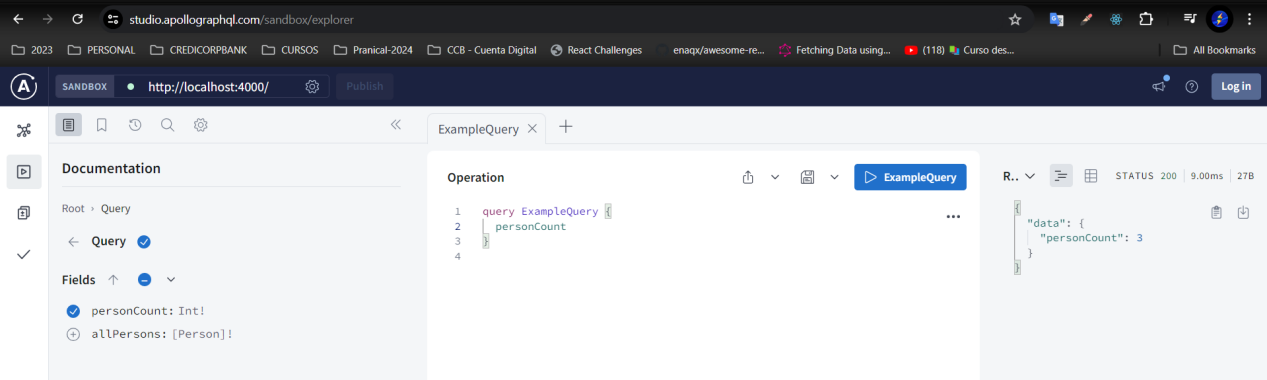
});

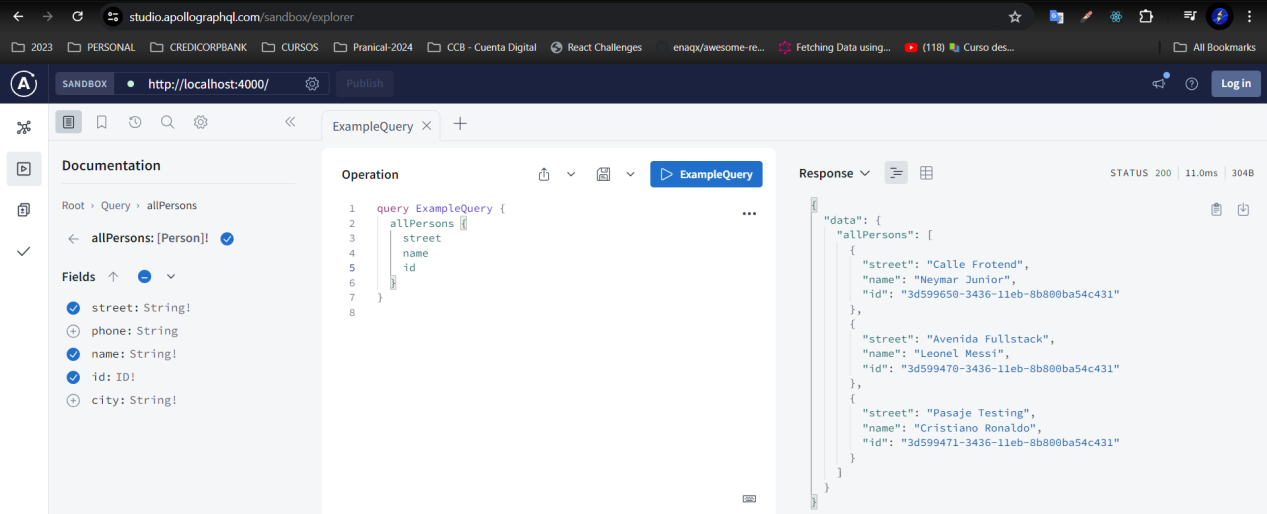
* Levantar el servidor



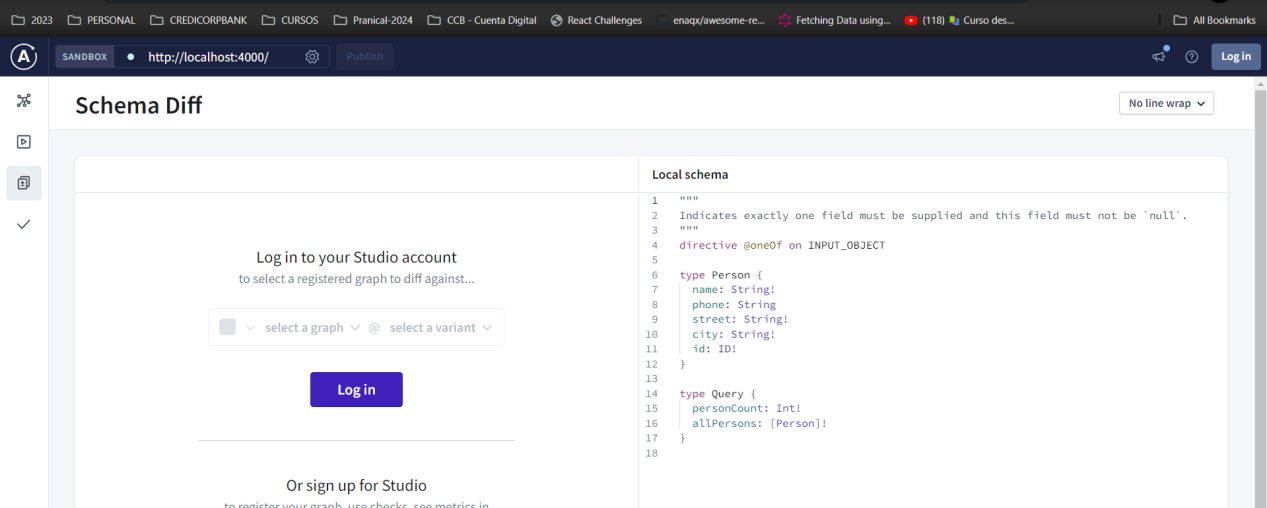
* Probar en la web



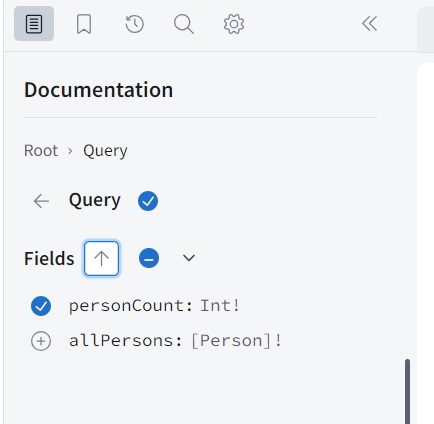
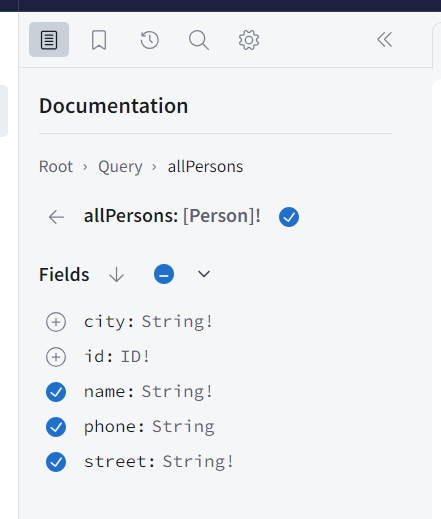


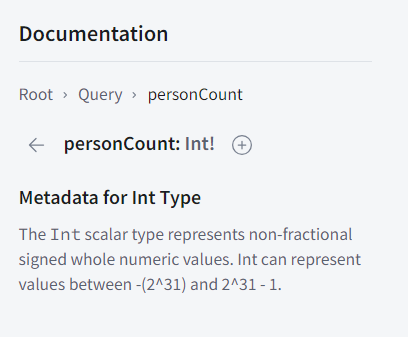
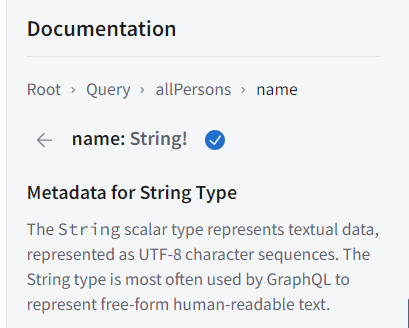
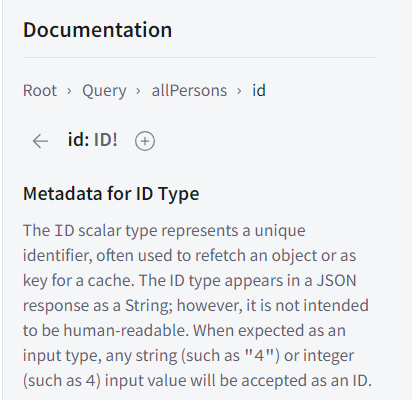


* Definición de los esquemas:

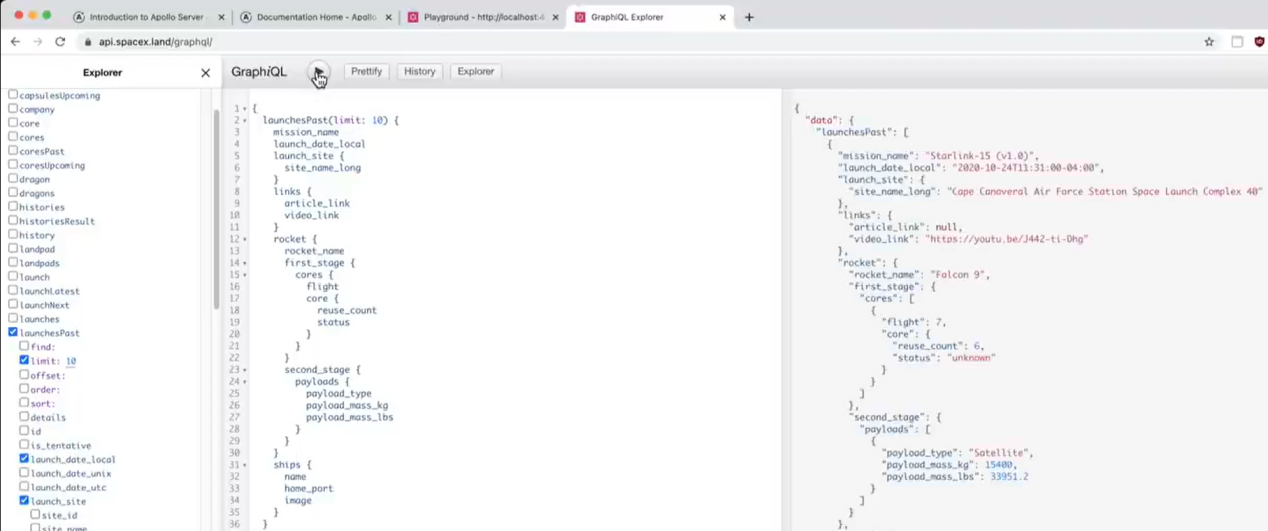


* Documentación

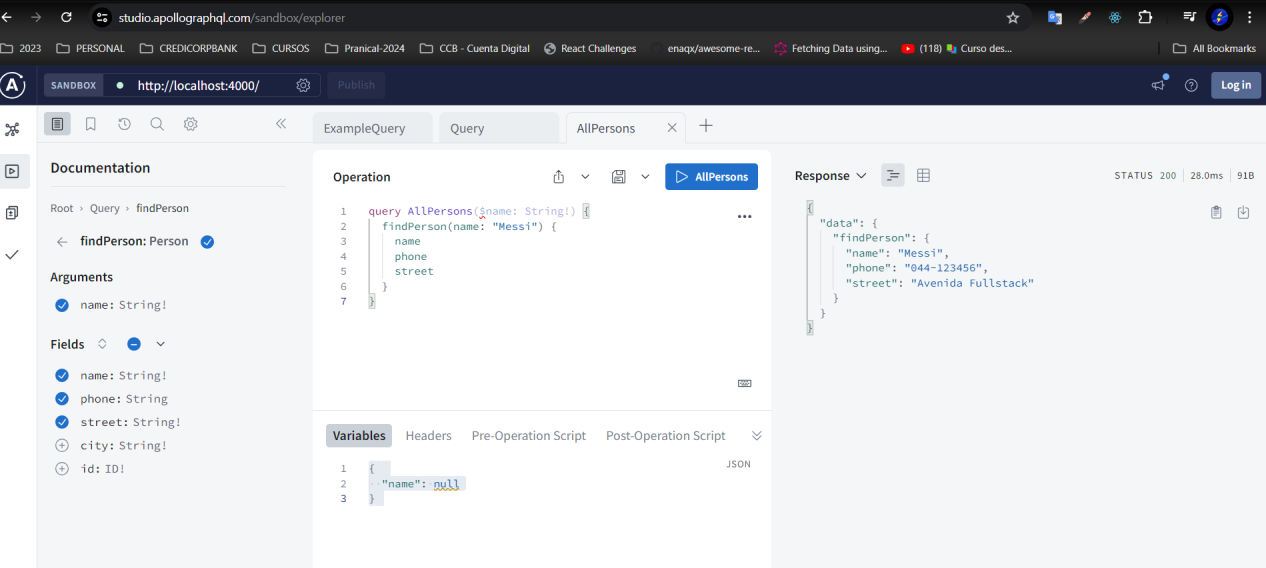
 

* Ejemplos de otra Graphiql



* Realizar una consulta en nuestros querys



* Agregar y modificar información del grafo

…

const typeDefinitions = gql`

  type Person {

    name: String!

    phone: String

    street: String!

    city: String!

    address: String!

    check: String!

    id: ID!

  }

…

const resolvers = {

  Query: {

    personCount: () => persons.length,

    allPersons: () => persons,

    findPerson: (root, args) => {

      const { name } = args;

      return persons.find((person) => person.name === name);

    },

  },

  Person: {

    address: (root) => `${root.street} - ${root.city}`,

    check: () => "Wlopera",

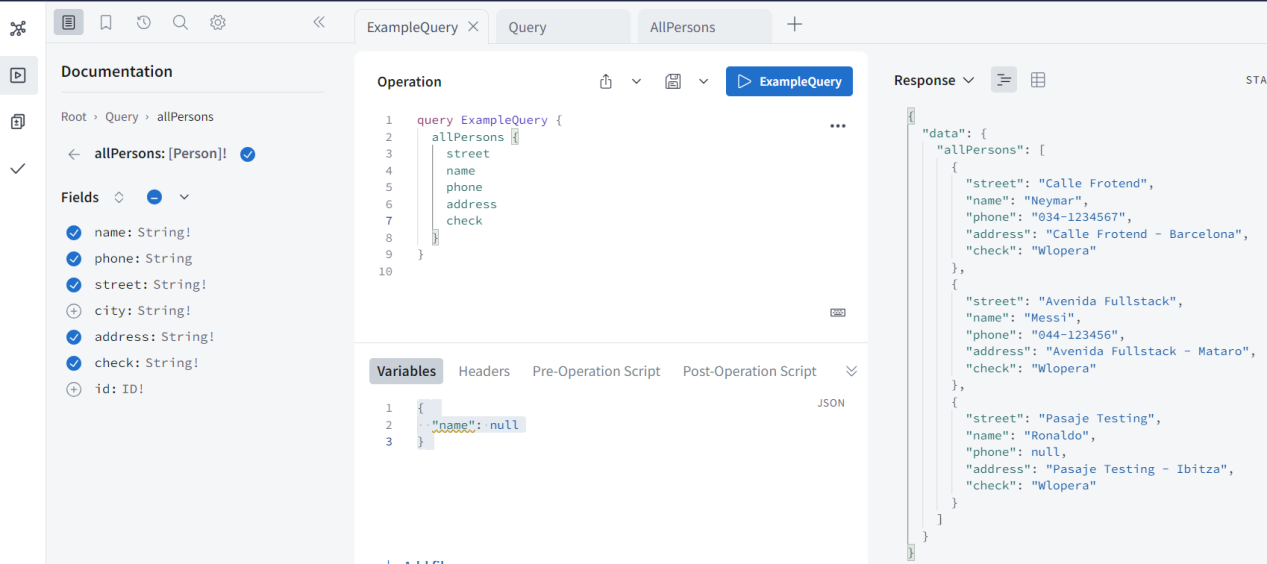
  },

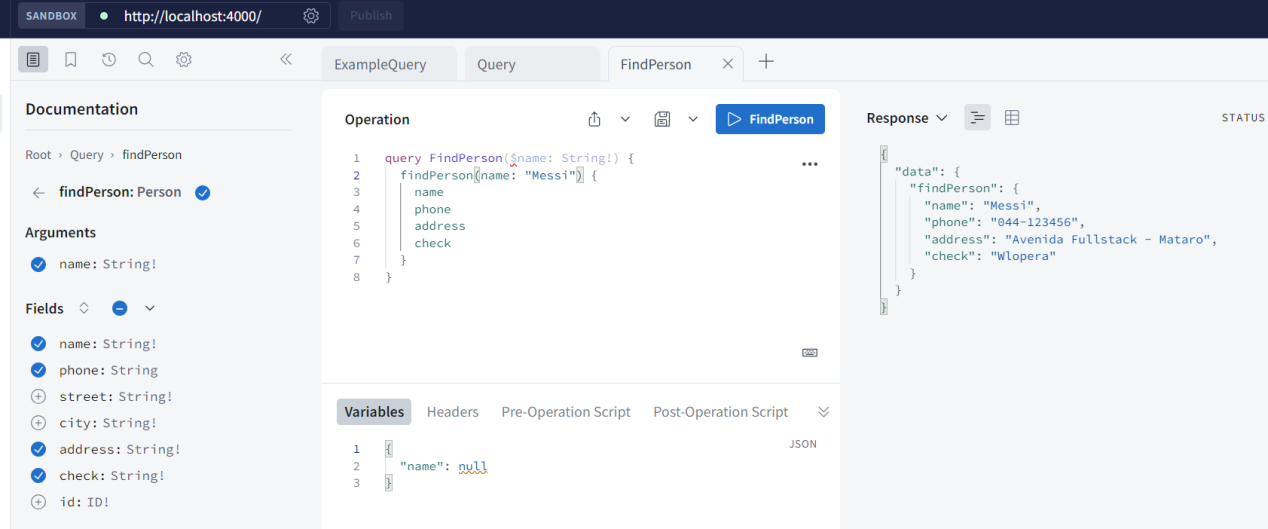
};

…

El root retorna la información de la consulta y de allí extraigo los valores a modificar o procesar. De esa forma graphql es como realiza la extracción de valores que se solicitan en su propio resolver.

Esto permite realizar cálculos o procesamiento de la data del lado del servidor





* Crear objetos dentro de un objeto en grapghql

import { ApolloServer, gql } from "apollo-server";

// Data DUMMY

const persons = [

  {

    name: "Neymar",

    phone: "034-1234567",

    street: "Calle Frotend",

    city: "Barcelona",

    id: "3d599650-3436-11eb-8b800ba54c431",

  },

  {

    name: "Messi",

    phone: "044-123456",

    street: "Avenida Fullstack",

    city: "Mataro",

    id: "3d599470-3436-11eb-8b800ba54c431",

  },

  {

    name: "Ronaldo",

    street: "Pasaje Testing",

    city: "Ibitza",

    id: "3d599471-3436-11eb-8b800ba54c431",

  },

];

// Tipos de datos y consultas

// ! => Obligatorio

// Query metodos de consulta expuestos

const typeDefinitions = gql`

  type Address {

    street: String!

    city: String!

  }

  type Person {

    name: String!

    phone: String

    address: Address!

    #check: String!

    id: ID!

  }

  type Query {

    personCount: Int!

    allPersons: [Person]!

    findPerson(name: String!): Person

  }

`;

// Resolvedores:  Como se obtiene y sacan los datos

const resolvers = {

  Query: {

    personCount: () => persons.length,

    allPersons: () => persons,

    findPerson: (root, args) => {

      const { name } = args;

      return persons.find((person) => person.name === name);

    },

  },

  //Person: {

  //  address: (root) => `${root.street} - ${root.city}`,

  //  check: () => "Chequeo de prueba",

  //},

  Person: {

    address: (root) => ({

      street: root.street,

      city: root.city,

    }),

  },

};

// Generar srevidors Apollo

const server = new ApolloServer({

  typeDefs: typeDefinitions,

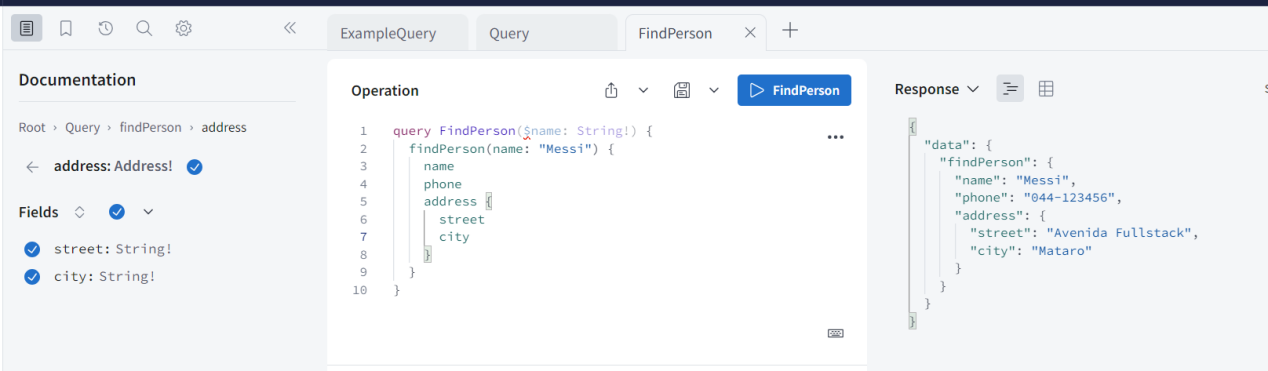
  resolvers,

});

server.listen().then(({ url }) => {

  console.log(`Servidor Listo en ${url}`);

});



**MUTACIONES**

* Importar librería uuid para manejo de identificador ID

\_> npm i uuid

* Agregar el código para manejo de mutaciones

import { ApolloServer, gql } from "apollo-server";

import { v1 as uuid } from "uuid";

...

const typeDefinitions = gql`

...

type Mutation {

    addPerson(

      name: String!

      phone: String

      street: String!

      city: String!

    ): Person

  }

`;

// Resolvedores:  Como se obtiene y sacan los datos

const resolvers = {

...

  Mutation: {

    addPerson: (root, args) => {

      const person = { ...args, id: uuid() };

      persons.push(person);

      return persons;

    },

  },

  //Person: {

  //  address: (root) => `${root.street} - ${root.city}`,

  //  check: () => "Chequeo de prueba",

  //},

  Person: {

    address: (root) => ({

      street: root.street,

      city: root.city,

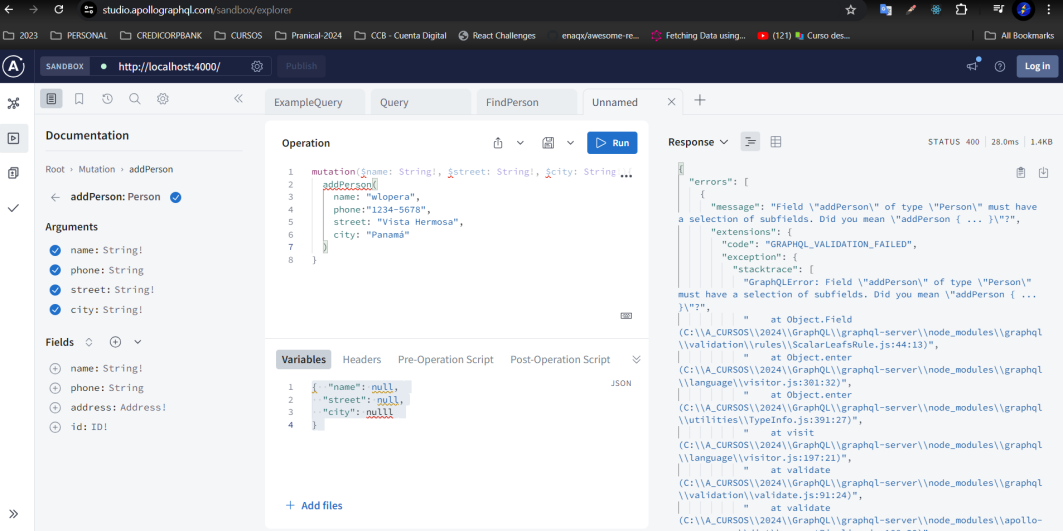
    }),

  },

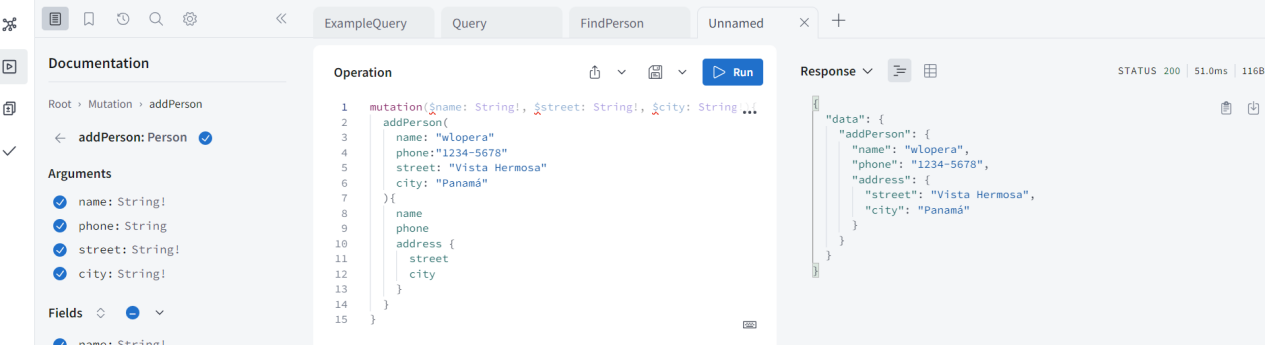
};

...

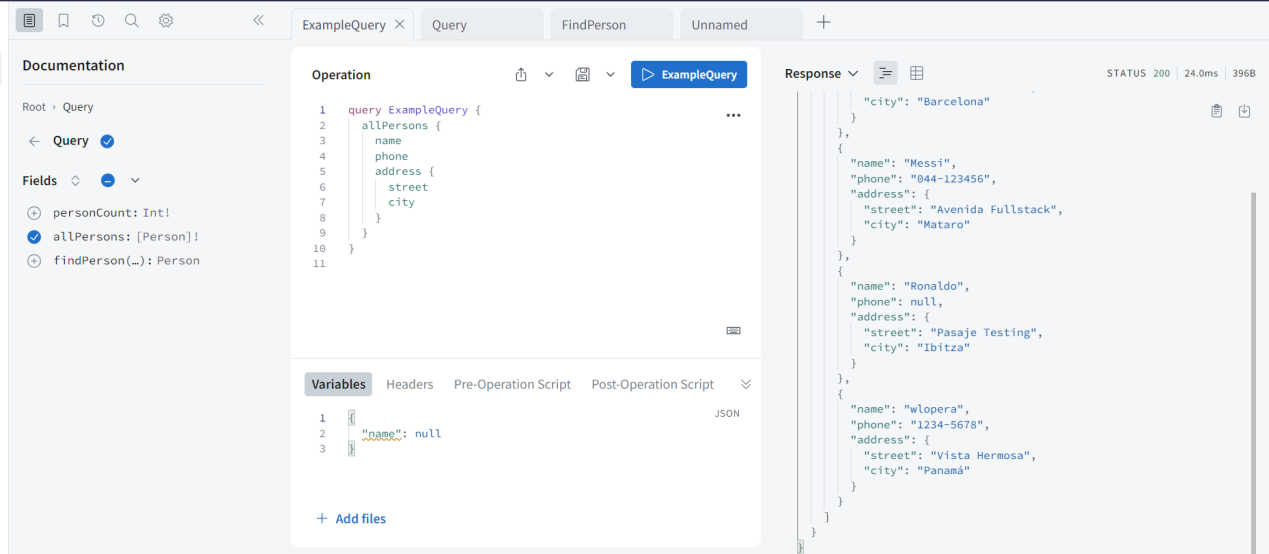
* Correr y probar la mutación



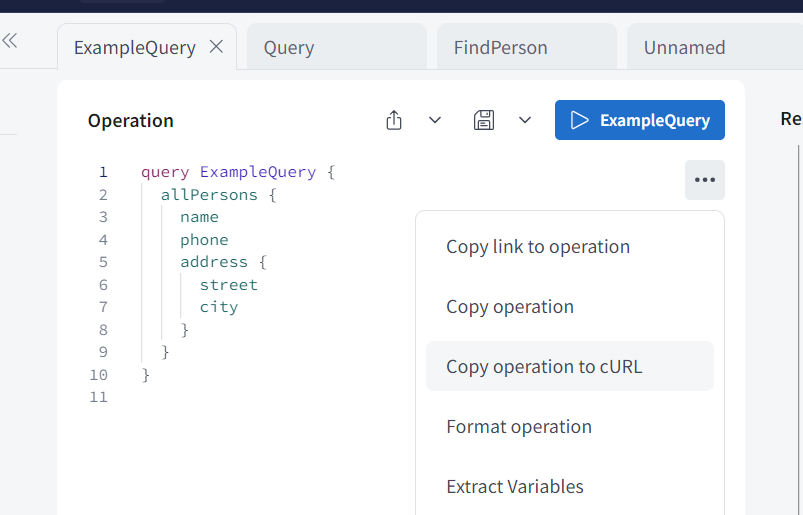
* Obtener los datos de la persona y armar el objeto

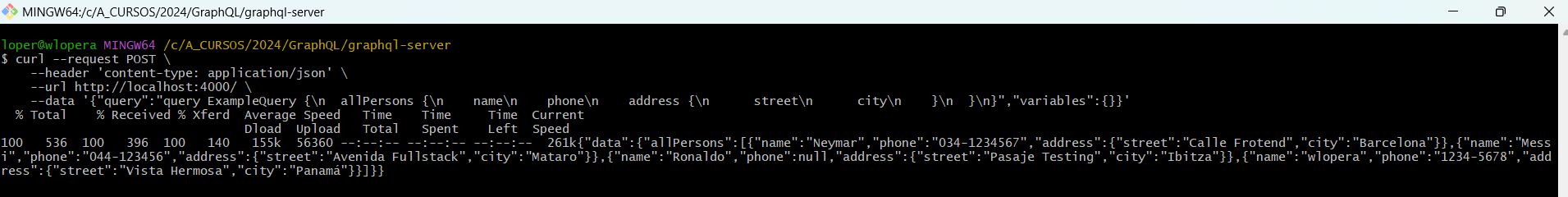


* Consultar data



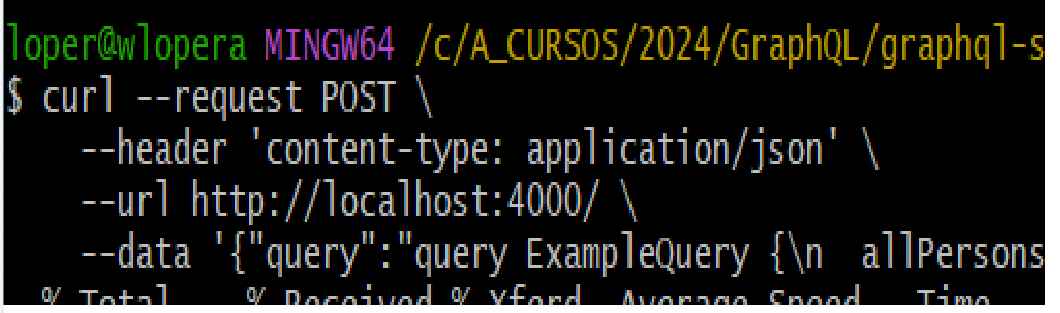
* Copiar el cURL y probar



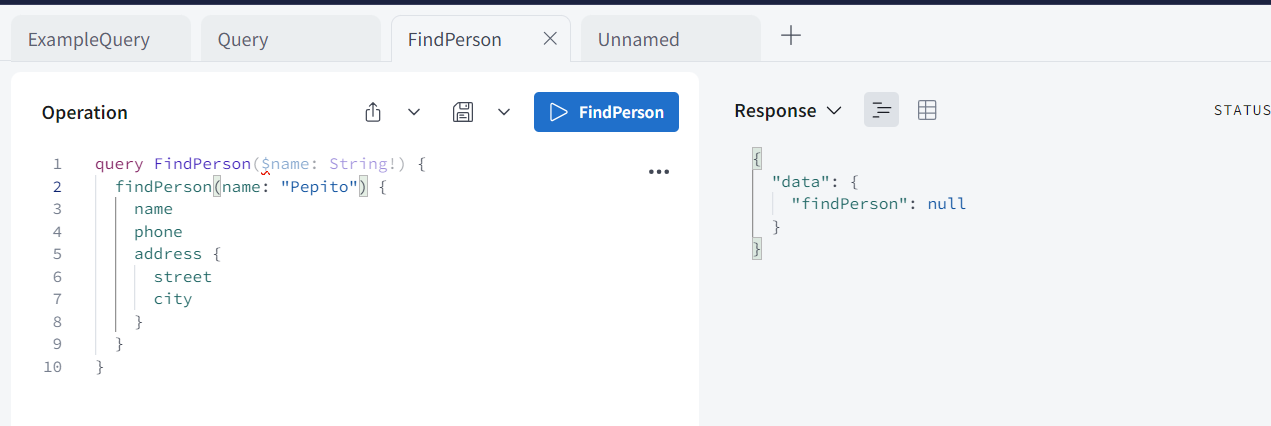


NOTA:

* + **Solo de utiliza una sola URL o punto de entrada a la API y pasarle la query**
  + **Siempre es un post a una misma url**



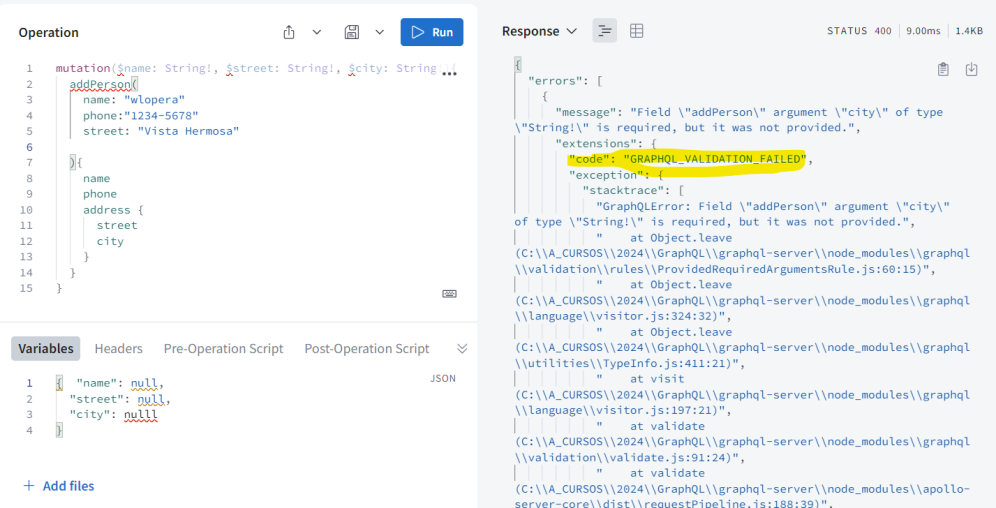
* Si la consulta no existe te retorna null

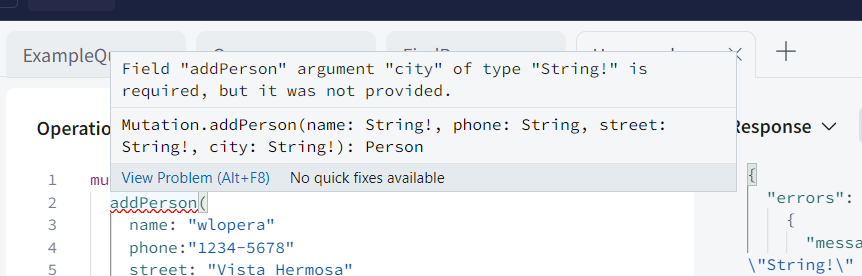


Nota: Se puede manejar y controlar del lado del cliente

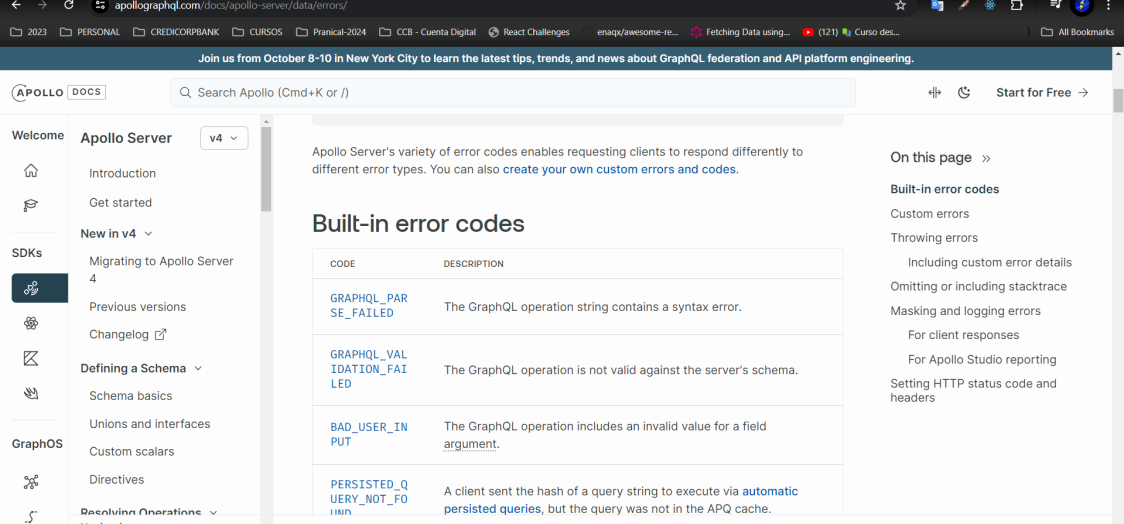
**Manejo de Errores y Validaciones**

* Si no paso la ciudad que es un campo obligatorio





<https://www.apollographql.com/docs/apollo-server/data/errors/>



* Evitar que se pueda agregar una persona dos veces validando por su nombre
  + Agregamos validación a la mutación para validar el campo name para la persona

...

Mutation: {

    addPerson: (root, args) => {

      if (persons.find((person) => person.name === args.name)) {

        throw new UserInputError("Nombre debe ser único", {

          invalidArgs: args.name,

        });

      }

      const person = { ...args, id: uuid() };

      persons.push(person);

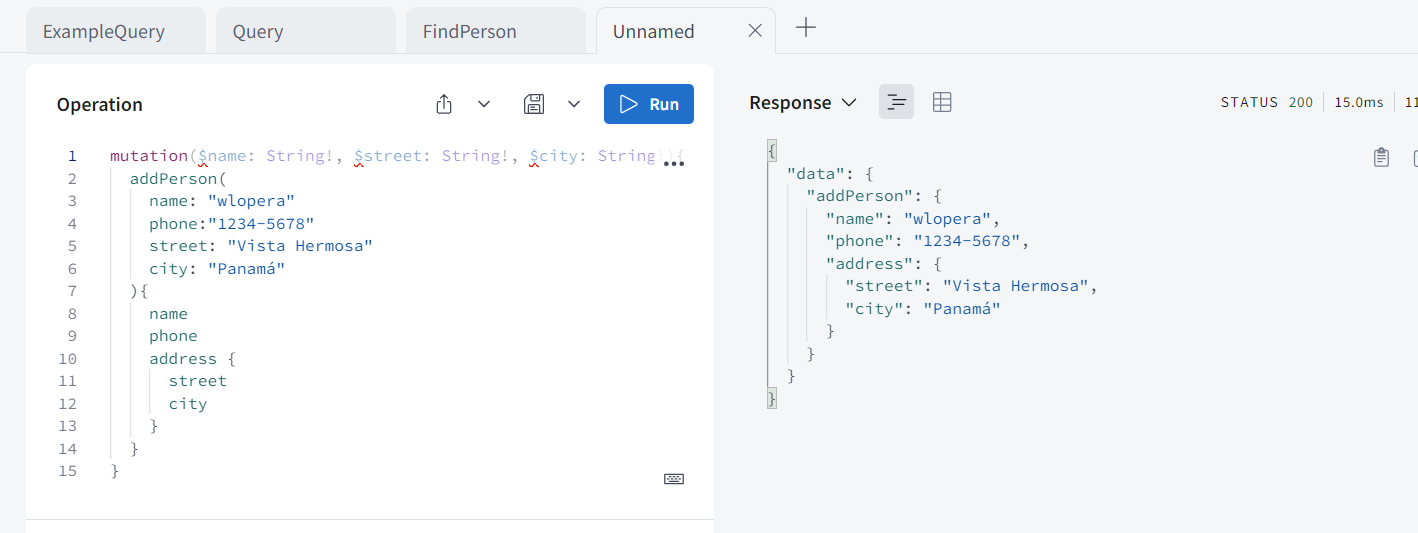
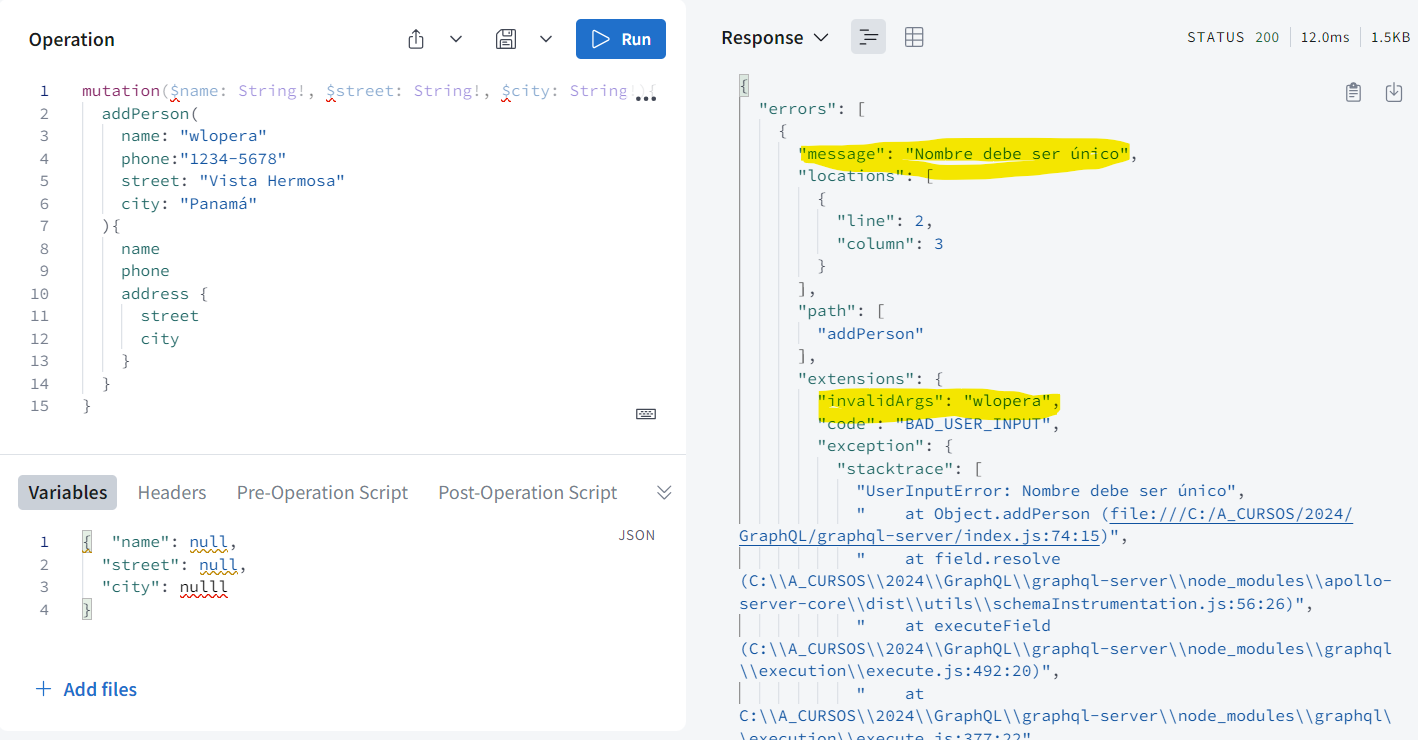
      return person;

    },

  },

...

* + Genero una vez



* + Y la segunda vez valida y retorna el error

**Uso de Enums**

* Crear una consulta de personas que tengan teléfono

...

const typeDefinitions = gql`

  enum YesNo {

    YES

    NO

  }

...

const resolvers = {

  Query: {

...

allPersons: (root, args) => {

      if (!args.phone) {

        return persons;

      }

      return persons.filter((person) =>

        args.phone === "YES" ? person.phone : !person.phone

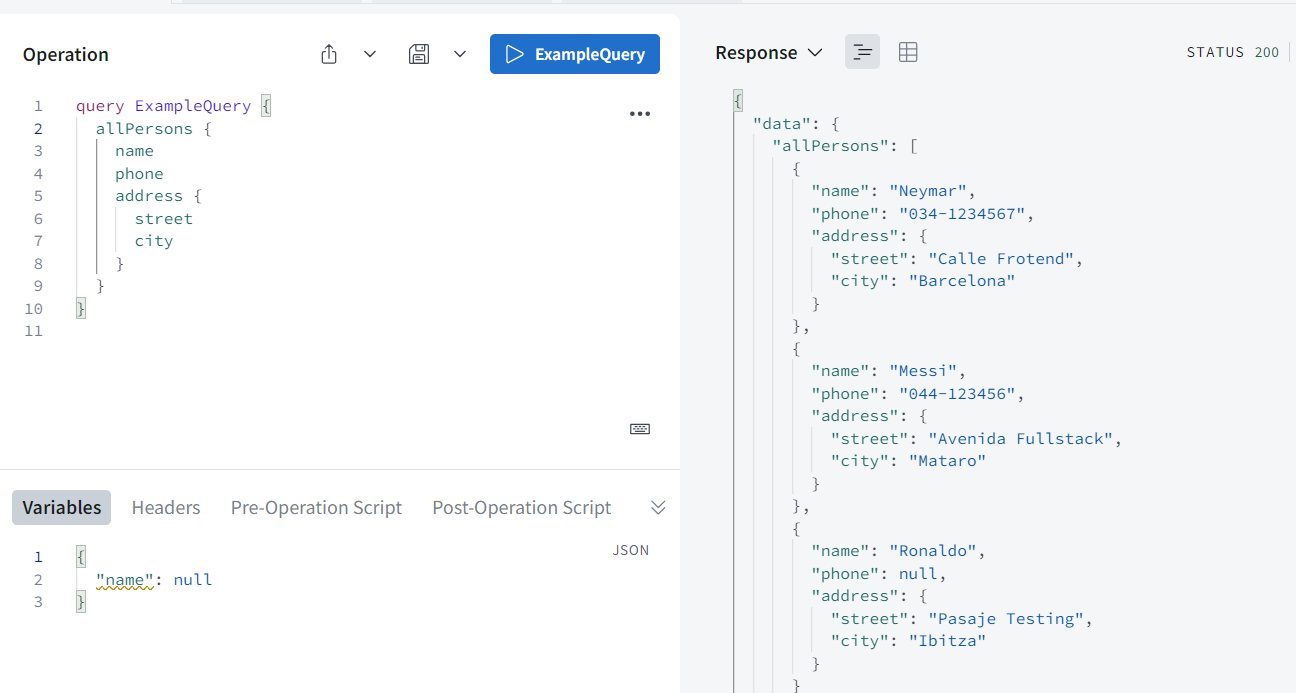
      );

    },

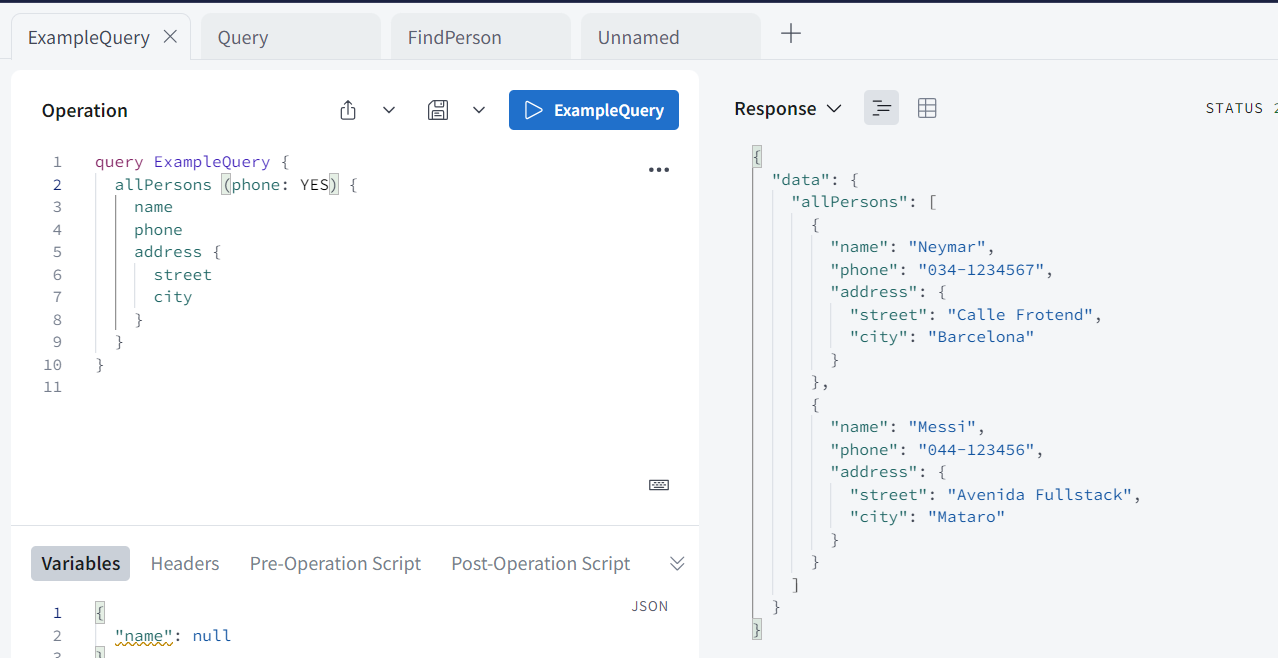
    ...

  },

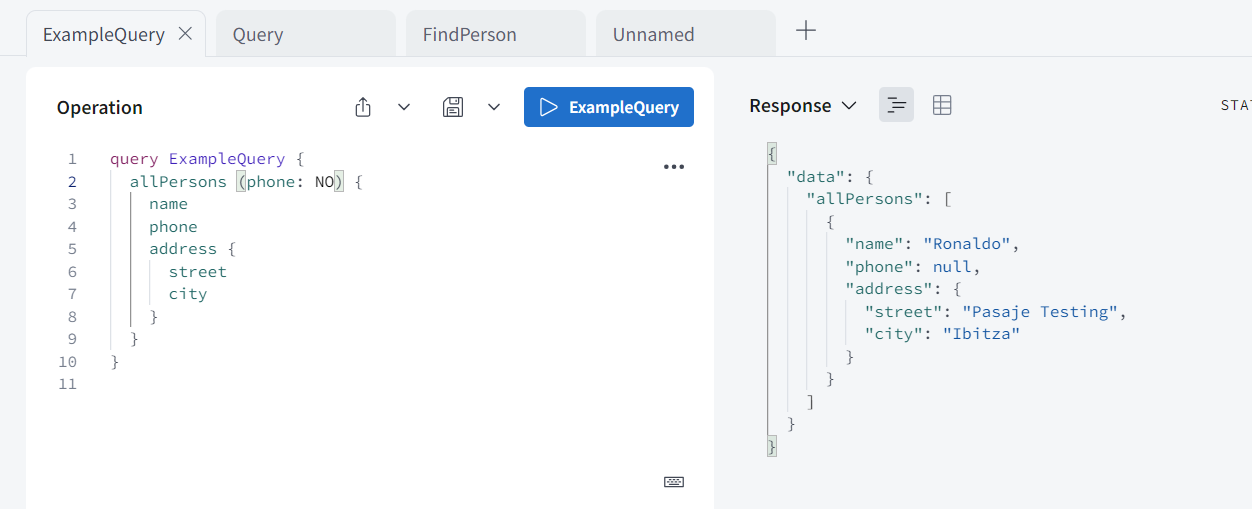
* + Sin pasar parámetro Enum



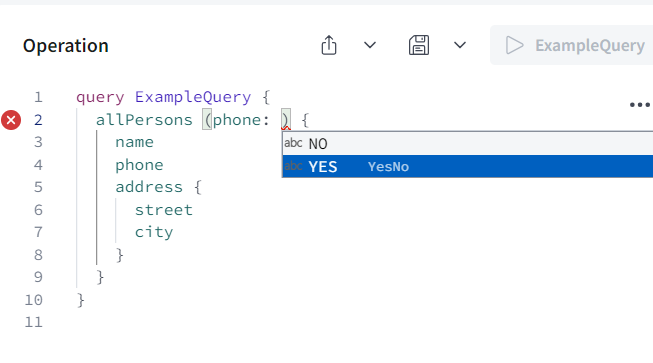
* + Con pasar parámetro Enum “YES”



* + Con pasar parámetro Enum “NO”



* Nota: La web me permite controlar la entrada de datos



**Modificar un número telefónico**

...

const typeDefinitions = gql`

  ...

  type Mutation {

    addPerson(

      name: String!

      phone: String

      street: String!

      city: String!

    ): Person

    editNumber(name: String!, phone: String!): Person

  }

`;

...

 Mutation: {

    ...

    editNumber: (root, args) => {

      const personIndex = persons.findIndex(

        (person) => person.name === args.name

      );

      if (personIndex === -1) {

        return null;

      }

      const personUpdate = persons[personIndex];

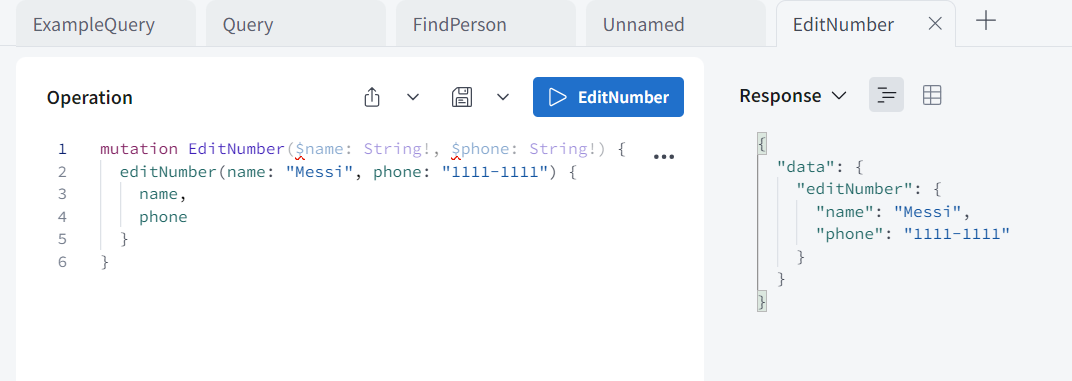
      persons[personIndex] = { ...personUpdate, phone: args.phone };

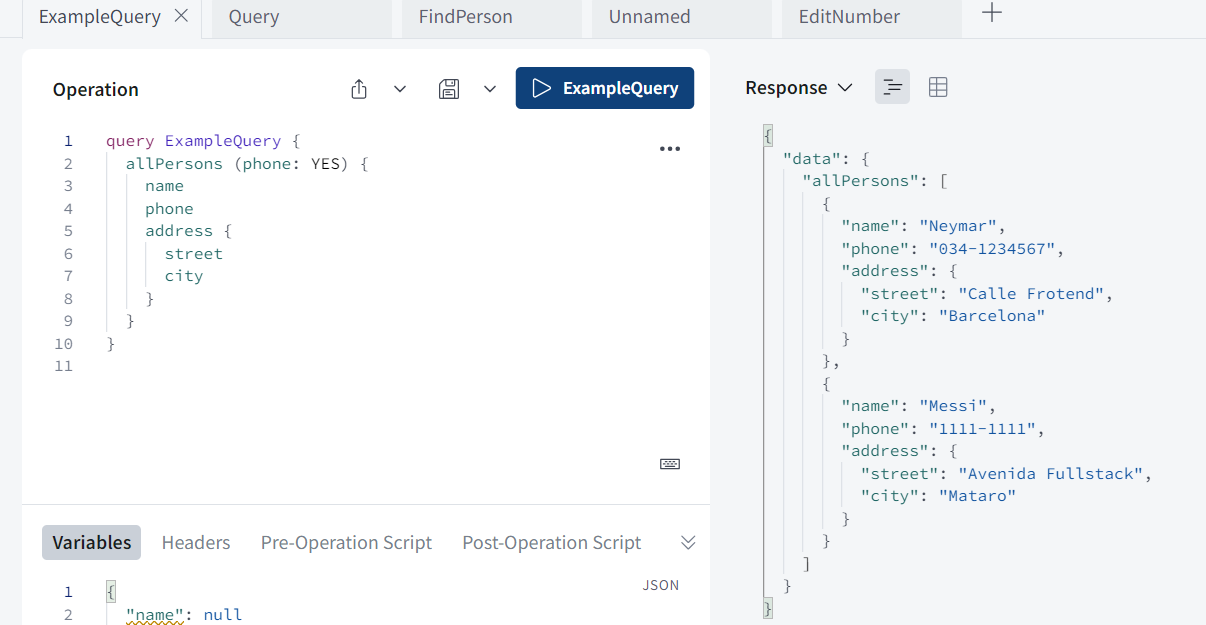
      return personUpdate;

    },

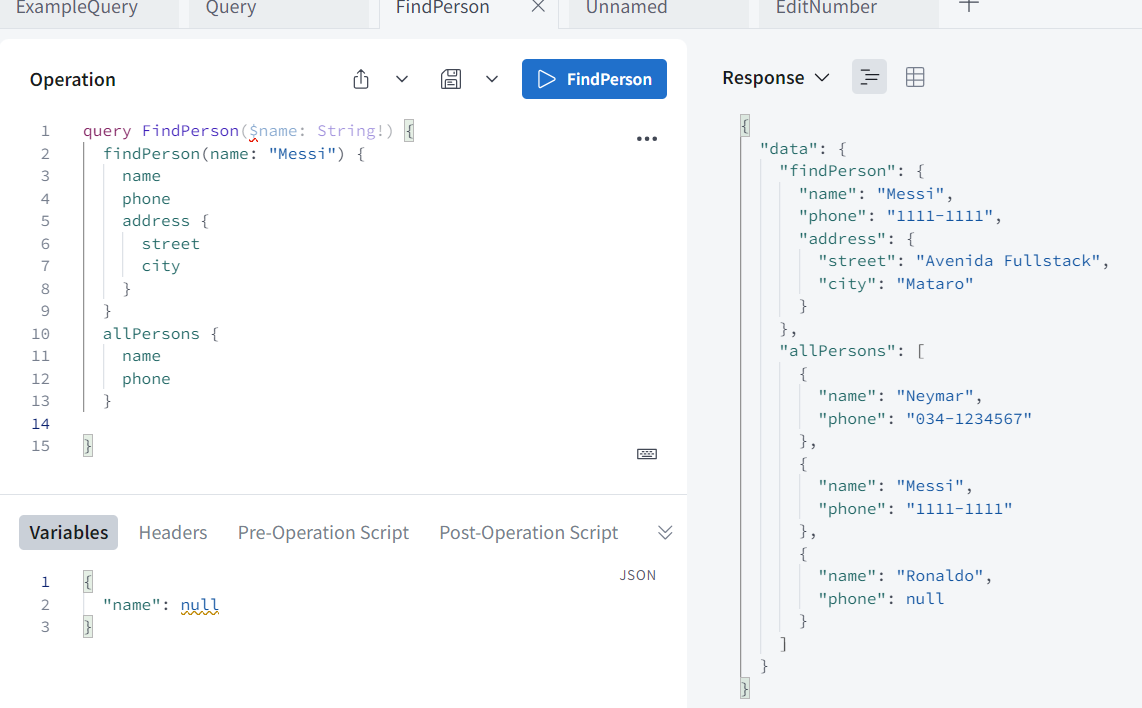
  },

...

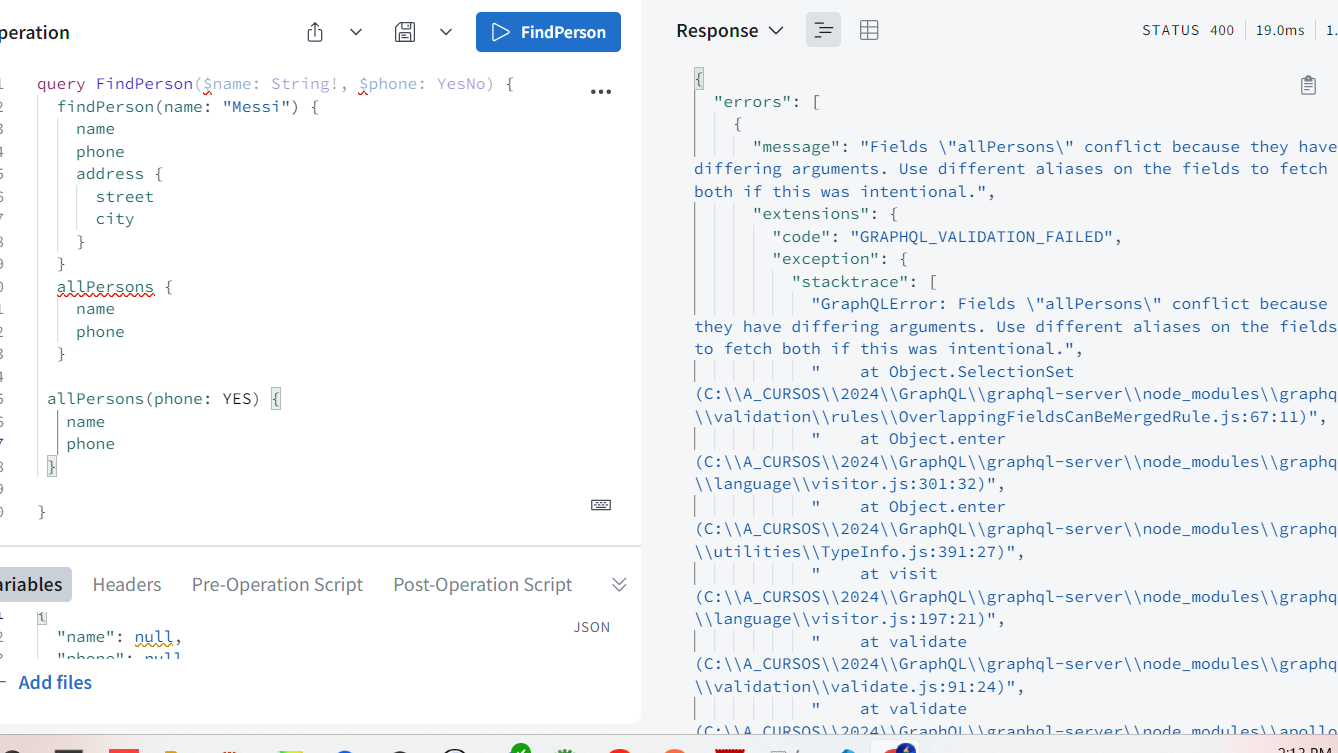




**Consultas compuestas (Queries anidados)**



* No puedo llamar a una función con argumentos diferentes



* Le paso un nombre de campo (la key de retorno) a cada llamada y si lo acepta



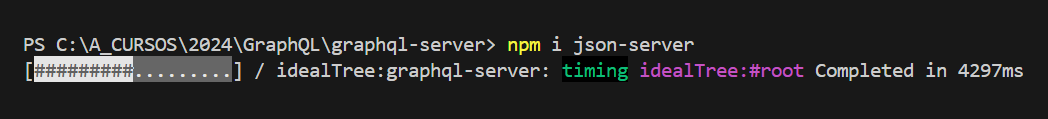
**Crear un API utilizando un json-server**

Las API representan un ****conjunto de procedimientos que se utilizan para hacer que dos o más aplicaciones se comuniquen entre sí.****

Por supuesto, puede crear un servidor back-end completo utilizando la pila que prefiera. Sin embargo, esto requiere mucho tiempo de desarrollo.

**[JSON Server](http://github.com/typicode/json-server)** es un proyecto simple que ayuda a configurar una API REST con operaciones CRUD muy rápidamente.

\_> npm i json-server



* Crear archivo db.json con la data a procesar

{

    "persons": [

        {

            "name": "Neymar",

            "phone": "034-1234567",

            "street": "Calle Frotend",

            "city": "Barcelona",

            "id": "3d599650-3436-11eb-8b800ba54c431"

        },

        {

            "name": "Messi",

            "phone": "044-123456",

            "street": "Avenida Fullstack",

            "city": "Mataro",

            "id": "3d599470-3436-11eb-8b800ba54c431"

        },

        {

            "name": "Ronaldo",

            "street": "Pasaje Testing",

            "city": "Ibitza",

            "id": "3d599471-3436-11eb-8b800ba54c431"

        }

    ]

}

* Actualizar package.json con script para poder levantar el json-server de db.json

...

 "scripts": {

    "json-server":"json-server --watch db.json",

    "test": "echo \"Error: no test specified\" && exit 1"

  },

...

* Levantar y probar el json-server

PS C:\A\_CURSOS\2024\GraphQL\graphql-server> npm run json-server

> graphql-server@1.0.0 json-server

> json-server --watch db.json

--watch/-w can be omitted, JSON Server 1+ watches for file changes by default

JSON Server started on PORT :3000

Press CTRL-C to stop

Watching db.json...

Index:

http://localhost:3000/

Static files:

Serving ./public directory if it exists

Endpoints:

<http://localhost:3000/persons>

{

    "persons": [

        {

            "name": "Suarez",

            "phone": "034-1234567",

            "street": "Calle Frotend",

            "city": "Barcelona",

            "id": "3d599650-3436-11eb-8b800ba54c431"

        },

        {

            "name": "Rondón",

            "phone": "044-123456",

            "street": "Avenida Fullstack",

            "city": "Mataro",

            "id": "3d599470-3436-11eb-8b800ba54c431"

        },

        {

            "name": "Dibu",

            "street": "Pasaje Testing",

            "city": "Ibitza",

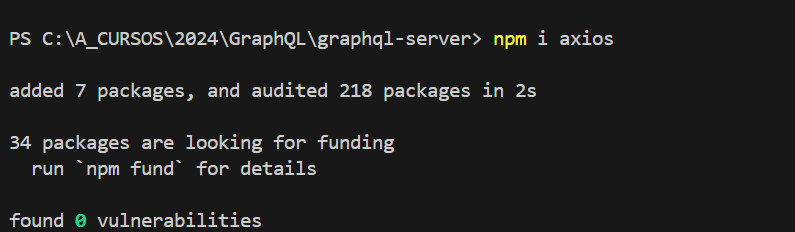
            "id": "3d599471-3436-11eb-8b800ba54c431"

        }

    ]

}

* Instalar axios para consumir el API Rest



...

import axios from "axios";

...

const resolvers = {

  Query: {

...

    allPersons: async (root, args) => {

      const { data: personsFromRestApi } = await axios.get(

        "http://localhost:3000/persons"

      );

      if (!args.phone) {

        return personsFromRestApi;

      }

      return personsFromRestApi.filter((person) =>

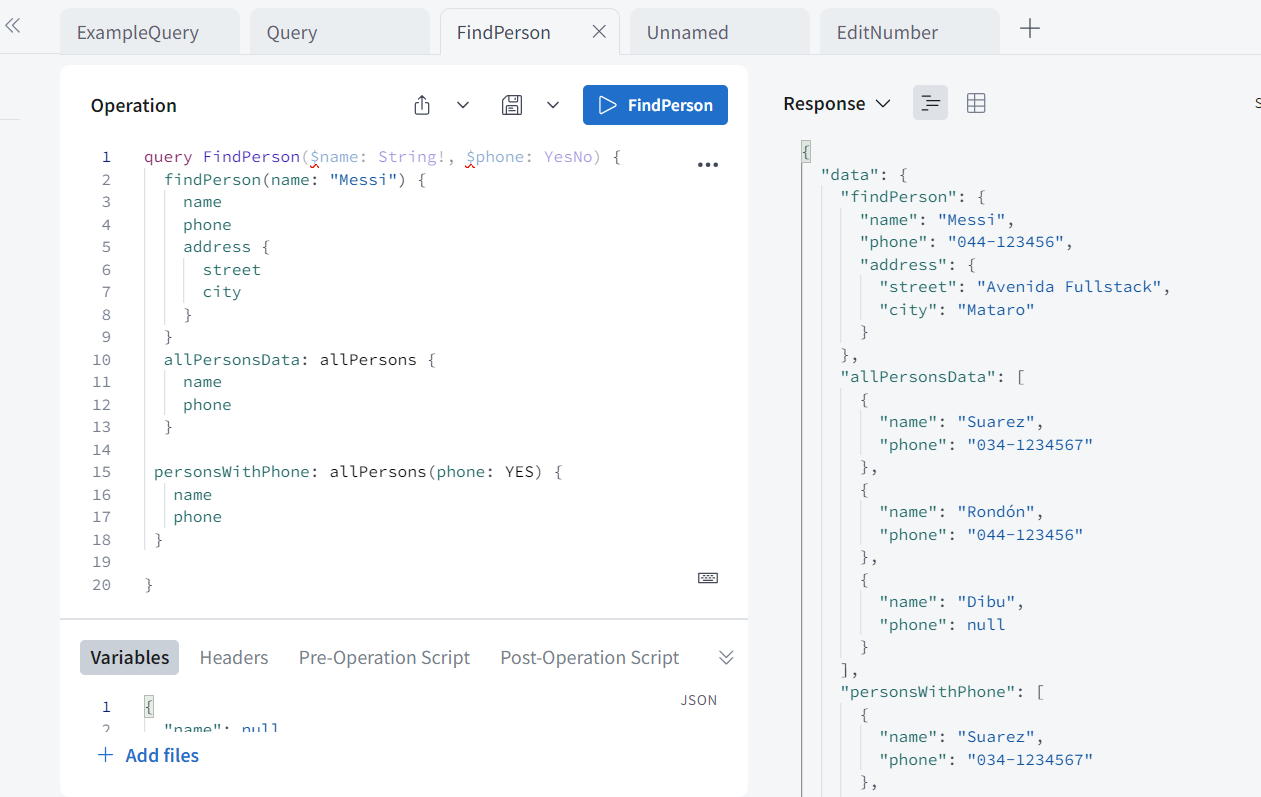
        args.phone === "YES" ? person.phone : !person.phone

      );

    },

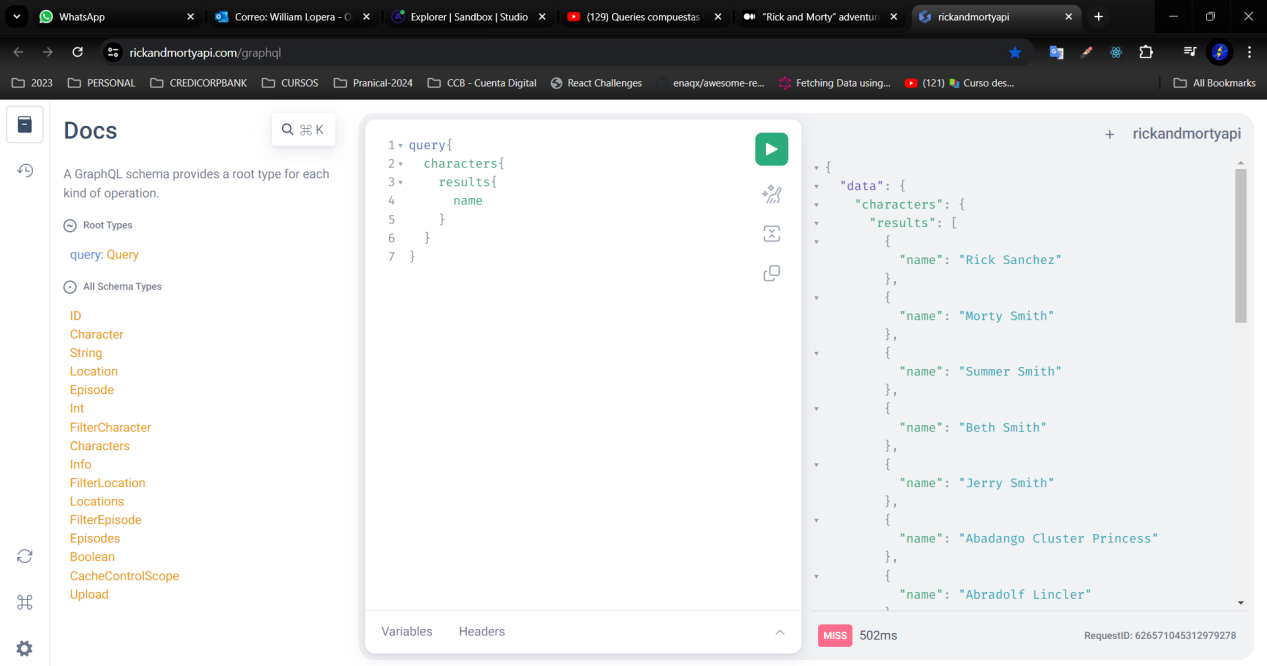
    ...

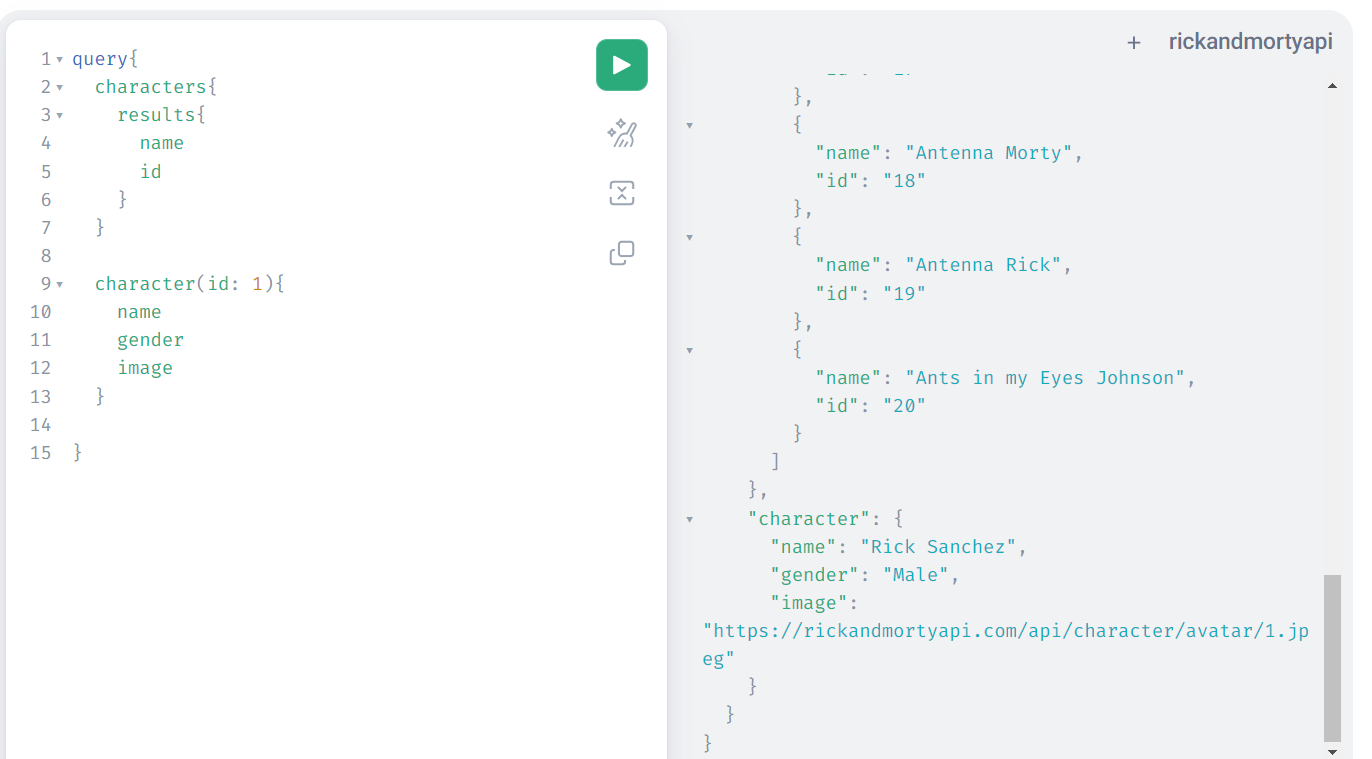
  },



**Ejemplo Rick and morty Api Graphql**

<https://rickandmortyapi.com/graphql>





* Agregar un script en package.json para levantar nodemon (Servidor de NodeJS)

 "scripts": {

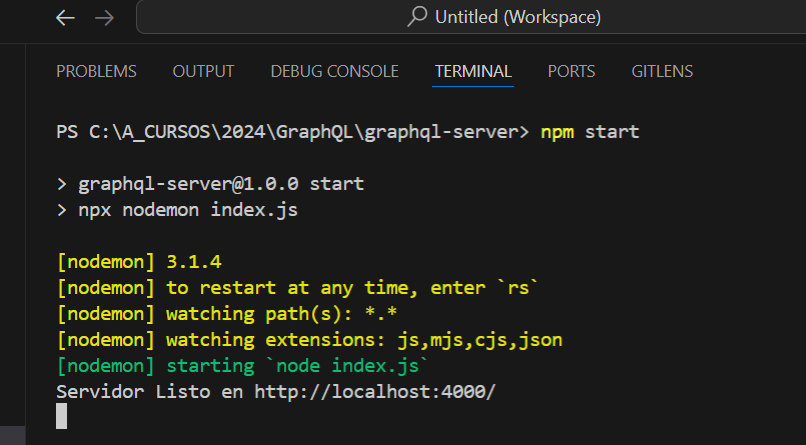
    "start": "npx nodemon index.js",

    "json-server": "json-server --watch db.json",

    "test": "echo \"Error: no test specified\" && exit 1"

  },

* En una terminal de VSCODE:



* Probar desde Thunder Client de VSCODE

