

PROYECTO GRAPHQL-SERVER

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

Vamos a crear un servidores graphql.

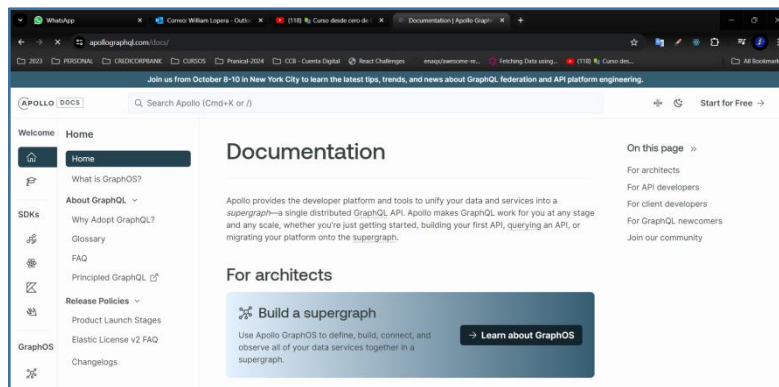
- Crear carpeta ..\GraphQL\graphql-server
- Crear proyecto ..\GraphQL\graphql-server>npm init -y

```
MINGW64:/c:/A_CURSOS/2024/GraphQL/graphql-server

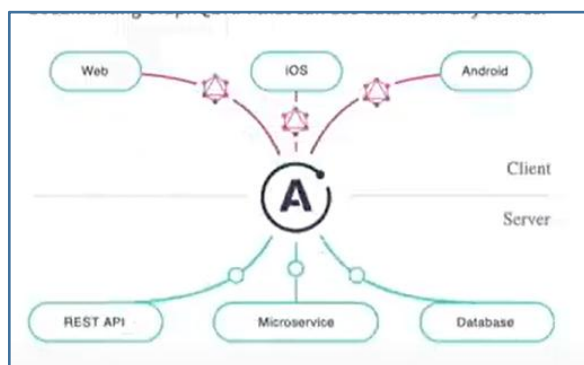
loper@wlopera MINGW64 /c:/A_CURSOS/2024/GraphQL/graphql-server
$ npm init -y
Wrote to C:\A_CURSOS\2024\GraphQL\graphql-server\package.json:

{
  "name": "graphql-server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

- Uso de framework o plataforma Apollo



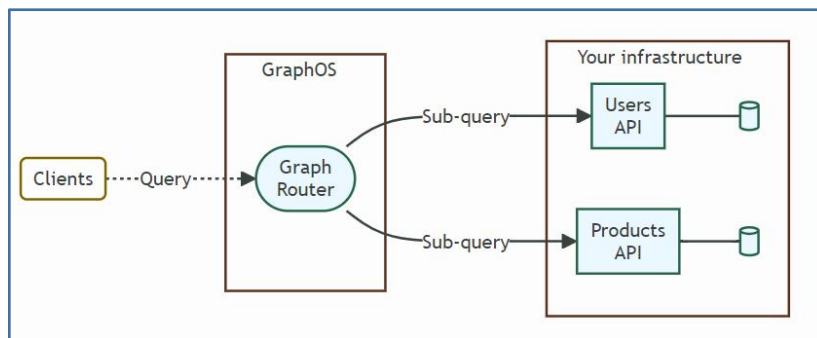
La plataforma Apollo



Unifique su infraestructura con GraphQL

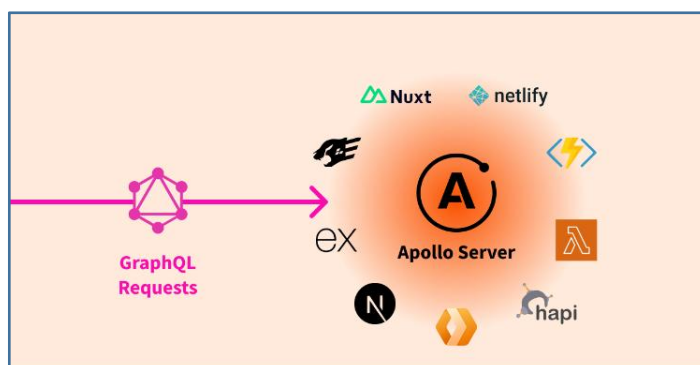
Apollo GraphOS es la plataforma para construir, administrar y escalar un supergrafo : una red unificada de los microservicios de su organización y sus fuentes de datos , todos compuestos en una única API distribuida.

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 abierto Servidor 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

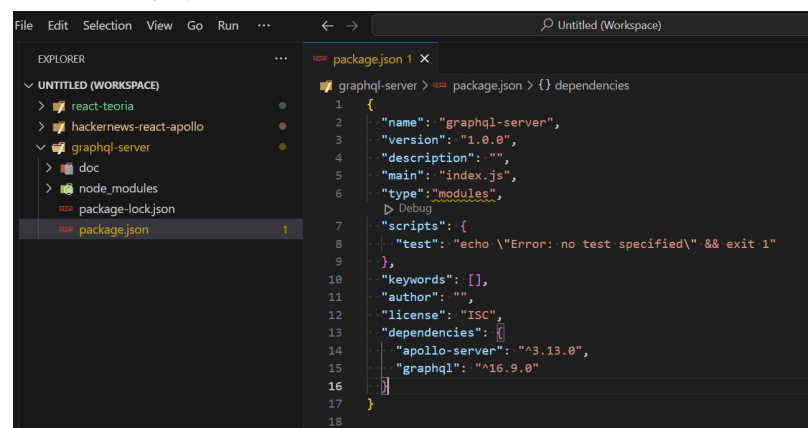
Librerías requeridas

- Servidor de Apollo: `_> npm install apollo-server`
- Grafos (GraphQL): `_> npm install graphql`

```
laper@wlopera MINGW64 /c/A_CURSOS/2024/GraphQL/graphql-server
$ npm i apollo-server
npm WARN deprecated apollo-server-errors@3.3.1: The 'apollo-server-errors' package is part of Apollo Server v2 and v3, which are no longer supported. This package's functionality is now found in the 'apollo/server' package. See https://www.apollographql.com/docs/apollo-server/previous-versions/ for more details.
npm WARN deprecated apollo-datasource@3.3.2: The 'apollo-datasource' package is part of Apollo Server v2 and v3, which are no longer supported. This package's functionality is now found in the 'apollo/server' package. See https://www.apollographql.com/docs/apollo-server/previous-versions/ for more details.
npm WARN deprecated apollo-server-types@3.8.0: The 'apollo-server-types' package is part of Apollo Server v2 and v3, which are no longer supported. This package's functionality is now found in the 'apollo/server' package. See https://www.apollographql.com/docs/apollo-server/previous-versions/ for more details.
npm WARN deprecated apollo-server-env@4.2.1: The 'apollo-server-env' package is part of Apollo Server v2 and v3, which are no longer supported. This package's functionality is now found in the 'apollo/utils.fetcher' package. See https://www.apollographql.com/docs/apollo-server/previous-versions/ for more details.
npm WARN deprecated apollo-reporting-protobuf@3.4.0: The 'apollo-reporting-protobuf' package is part of Apollo Server v2 and v3, which are no longer supported. This package's functionality is now found in the 'apollo/usage-reporting-protobuf' package. See https://www.apollographql.com/docs/apollo-server/previous-versions/ for more details.
npm WARN deprecated apollo-server-plugin-base@3.7.2: The 'apollo-server-plugin-base' package is part of Apollo Server v2 and v3, which are no longer supported. This package's functionality is now found in the 'apollo/server' package. See https://www.apollographql.com/docs/apollo-server/previous-versions/ for more details.
added 151 packages, and audited 152 packages in 13s
14 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

laper@wlopera MINGW64 /c/A_CURSOS/2024/GraphQL/graphql-server
$ npm i graphql
up to date, audited 152 packages in 1s
14 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities
```

● 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 resolvers 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 srevidores Apollo
const server = new ApolloServer({
  typeDefs: typeDefinitions,
  resolvers,
});

server.listen().then(({ url }) => {
  console.log(`Servidor Listo en ${url}`);
});

```

- Levantar el servidor

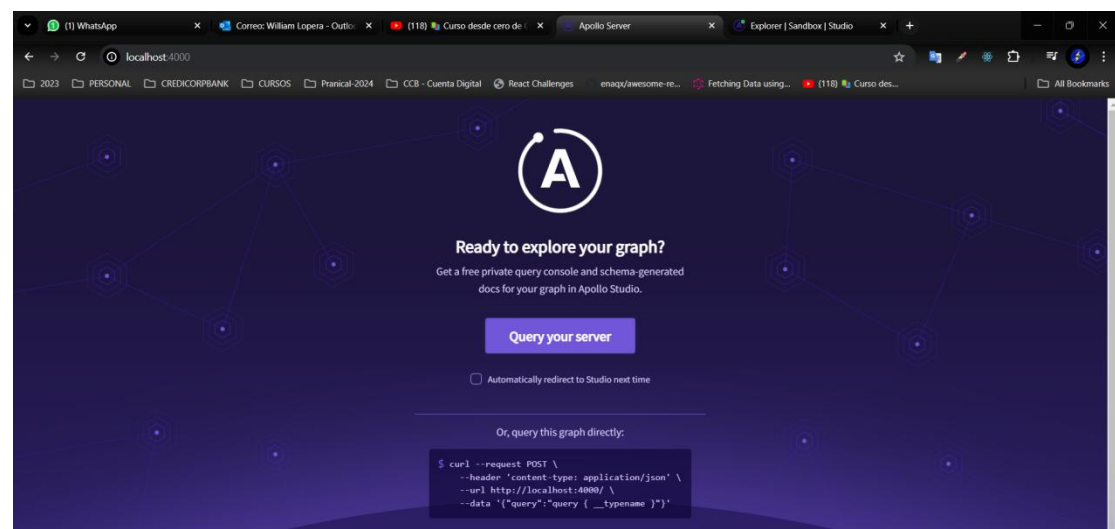
```

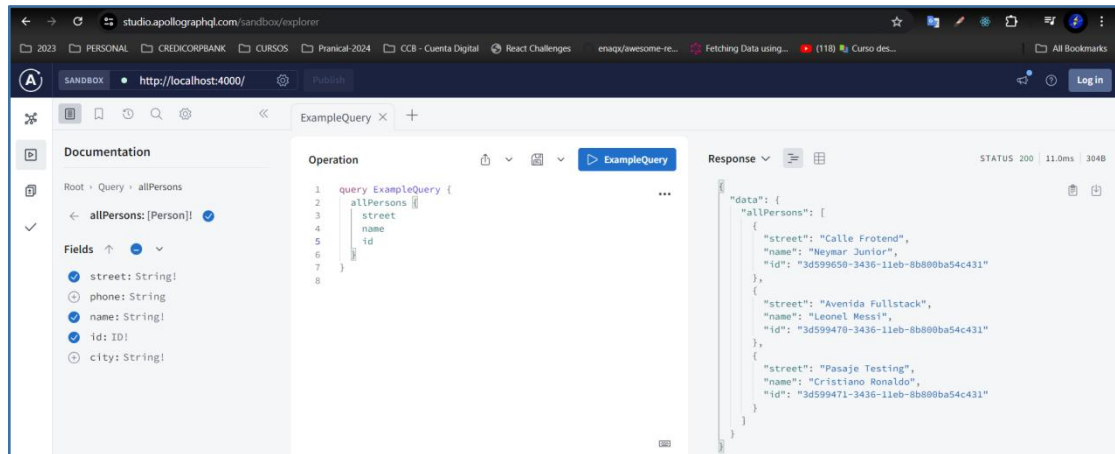
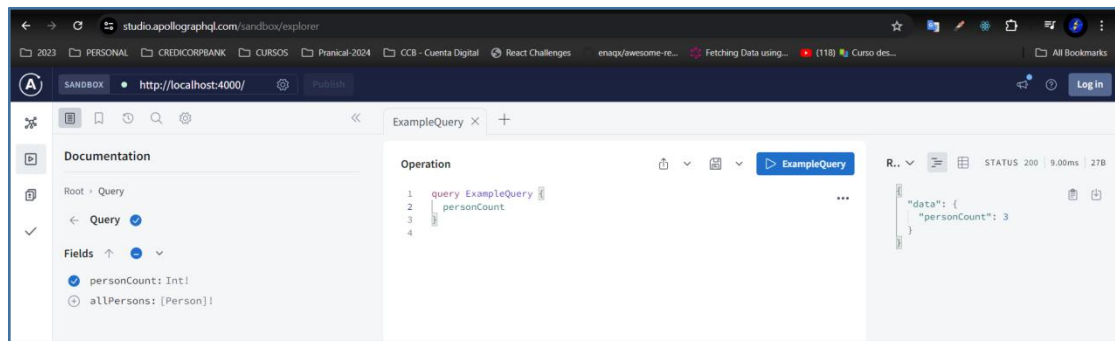
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

PS C:\_CURSOS\2024\GraphQL\graphql-server> node index.js
Servidor Listo en http://localhost:4000/

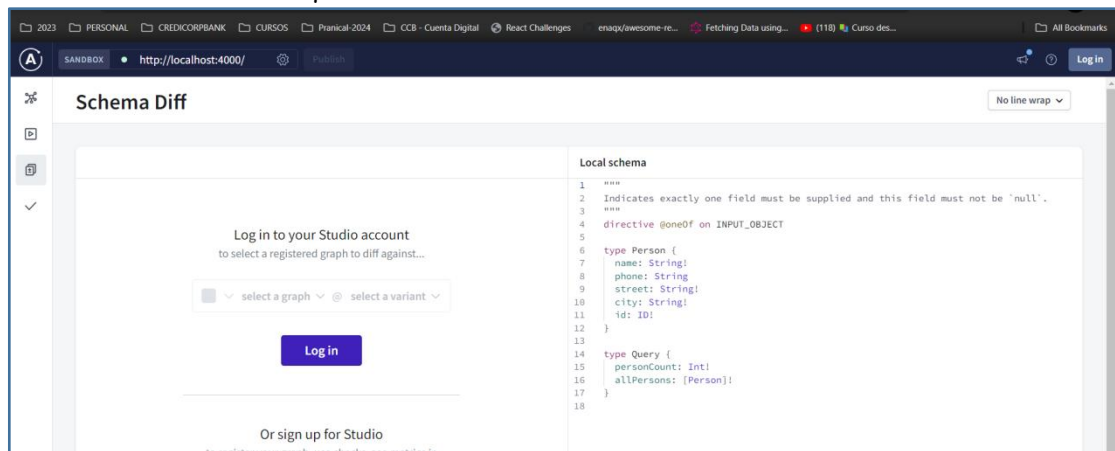
```

- Probar en la web

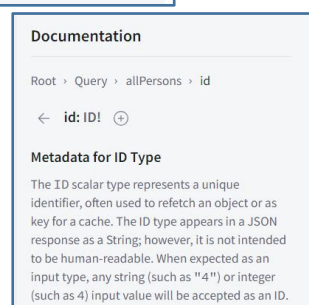
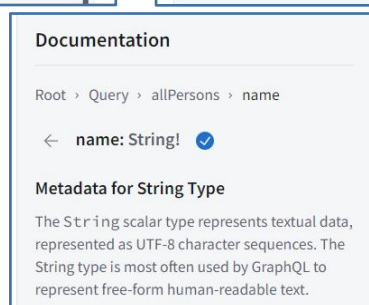
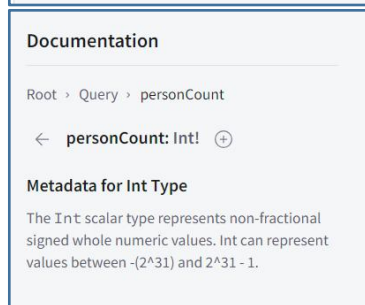
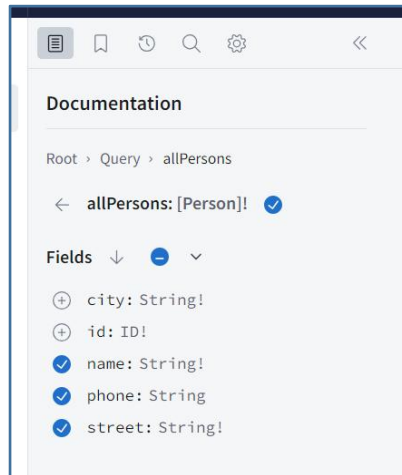
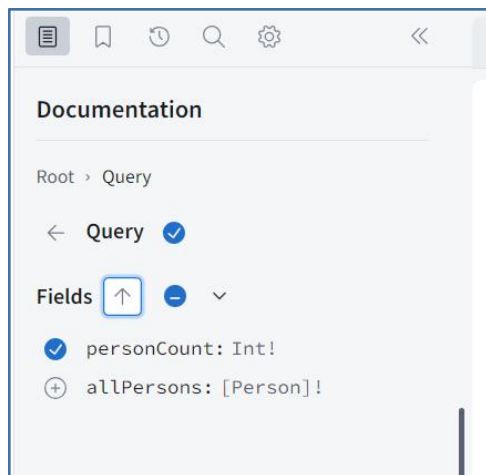




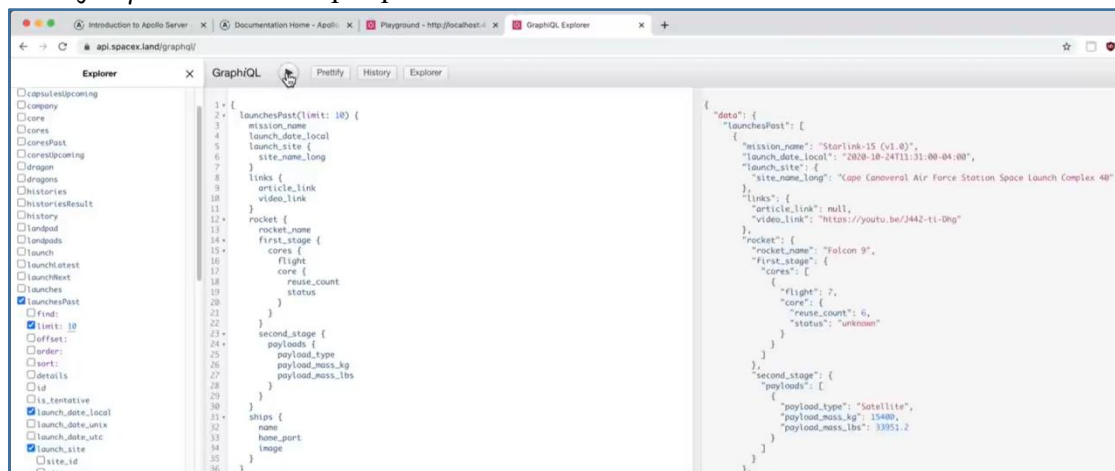
● Definición de los esquemas:



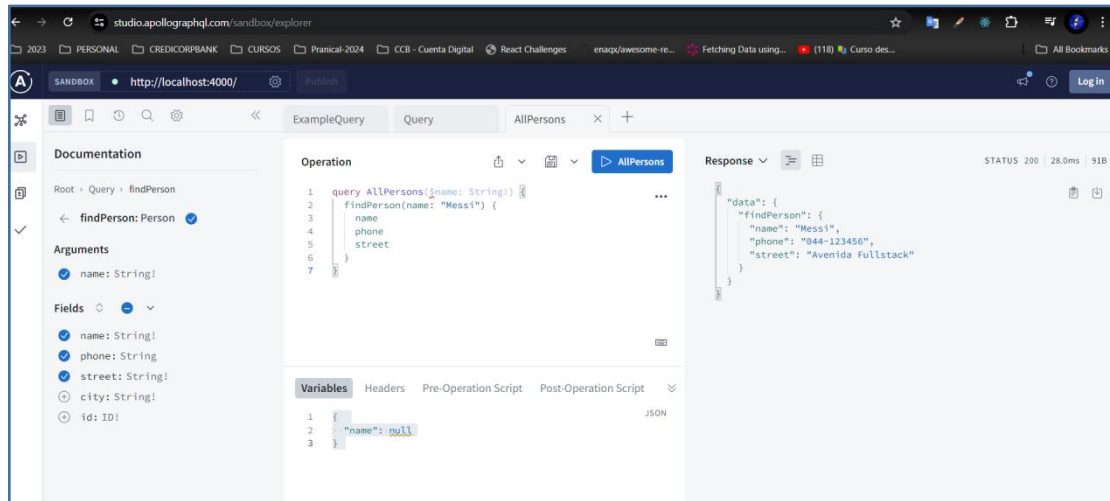
● Documentación



● Ejemplos de otra Graphiql



- Realizar una consulta en nuestros queries



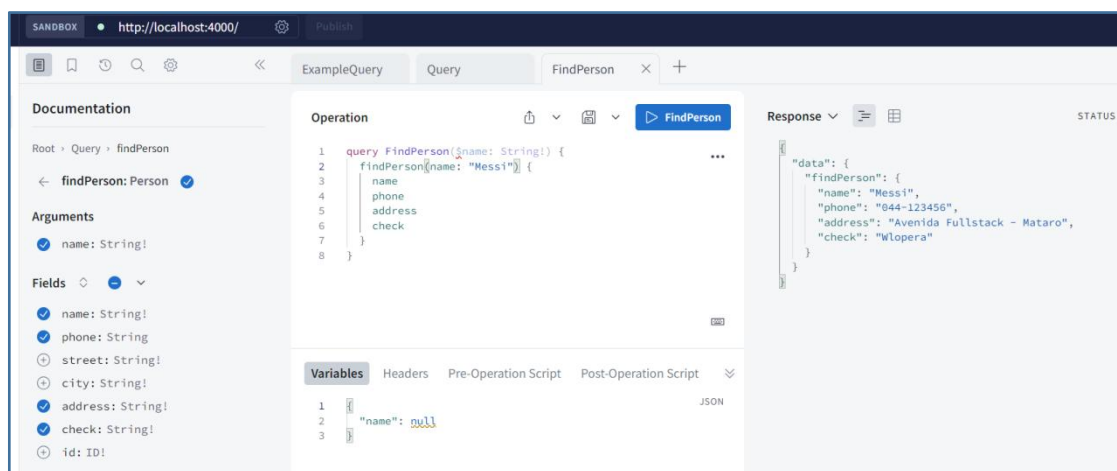
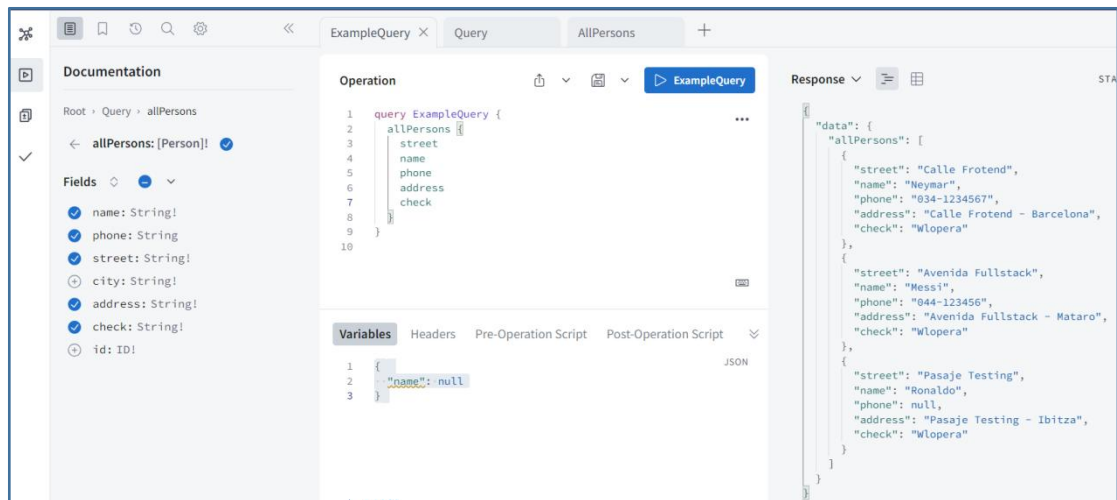
- Agregar y modificar información del grafo

```
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 graphql

```
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!
  }

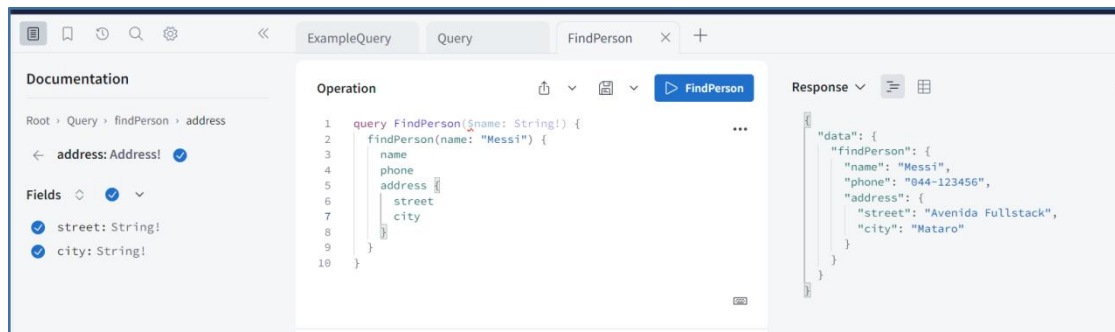
  type Query {
    personCount: Int!
    allPersons: [Person]!
    findPerson(name: String!): Person
  }
`;

// 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 srevidores Apollo
const server = new ApolloServer({
  typeDefs: typeDefinitions,
  resolvers,
});

server.listen().then(({ url }) => {
  console.log(`Servidor Listo en ${url}`);
});

```



MUTACIONES

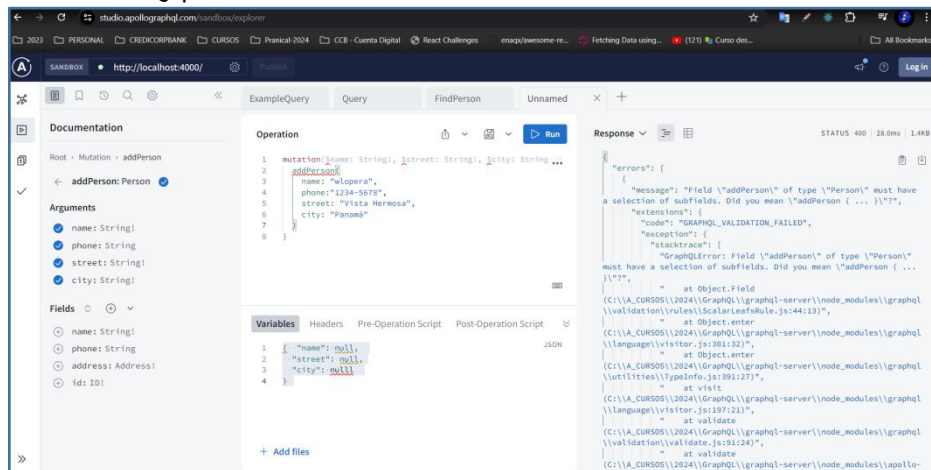
- *Importar librería uuid para manejo de identificador ID*
_> npm i uuid
- *Agregar el código para manejo de mutaciones*

```
import { ApolloServer, gql } from "apollo-server";
import { v1 as uuid } from "uuid";

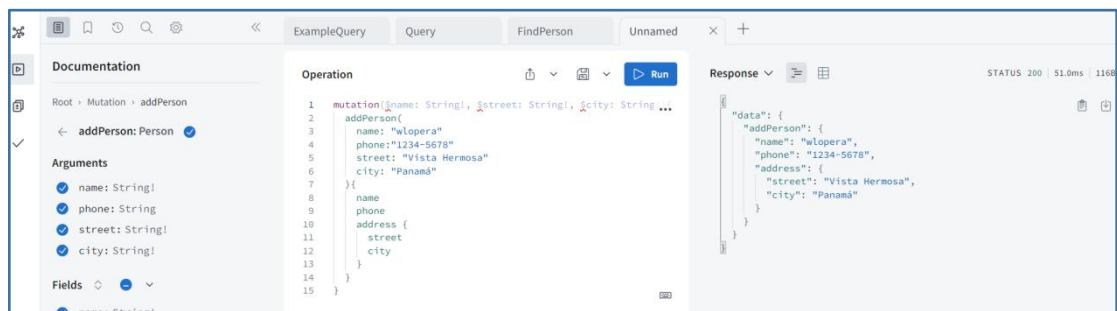
...
const typeDefinitions = gql`
...
  type Mutation {
    addPerson(
      name: String!
      phone: String
      street: String!
      city: String!
    ): Person
  }
`;

// Resolvedores: Como se obtiene y sacan los datos
const resolvers = {
  ...
  Mutation: {
    addPerson: (root, args) => {
      const person = { ...args, id: uuid() };
      persons.push(person);
      return person;
    },
  },
  //Person: {
  //  address: (root) => `${root.street} - ${root.city}`,
  //  check: () => "Chequeo de prueba",
  //},
  Person: {
    address: (root) => ({
      street: root.street,
      city: root.city,
    }),
  },
};
```

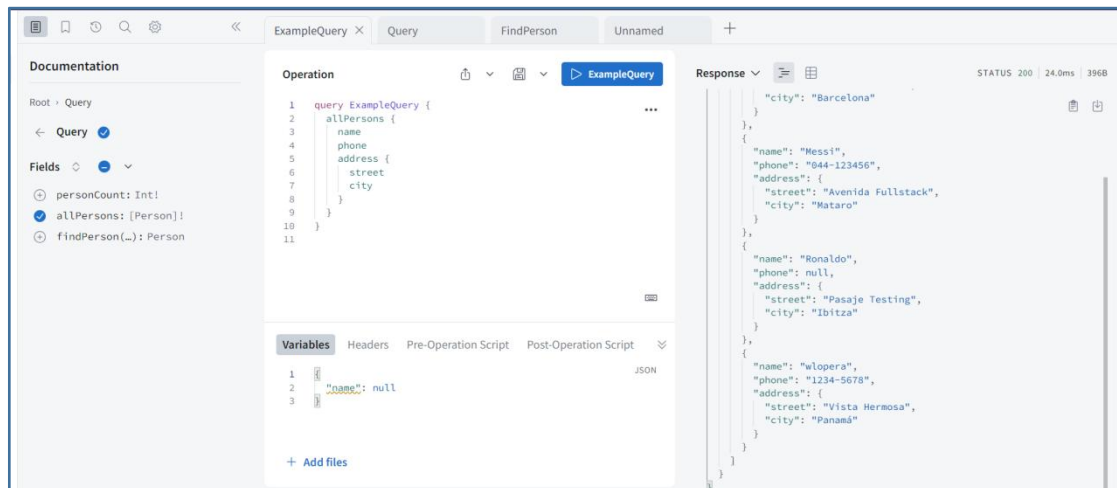
- *Correr y probar la mutación*



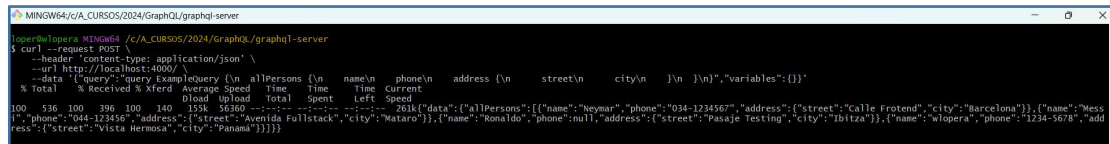
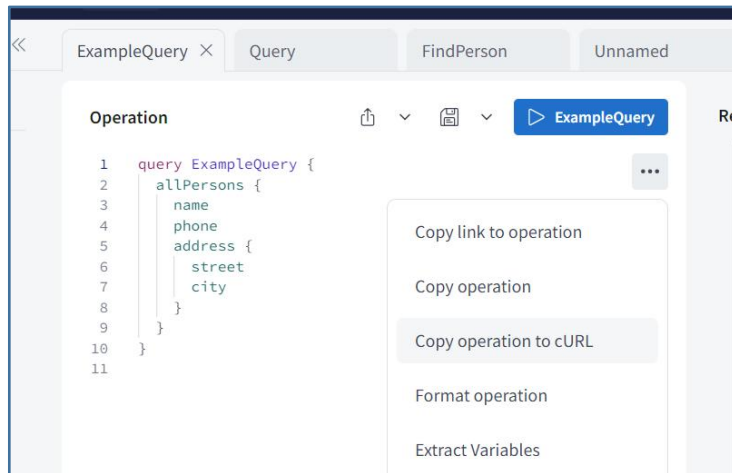
- *Obtener los datos de la persona y armar el objeto*



- *Consultar data*

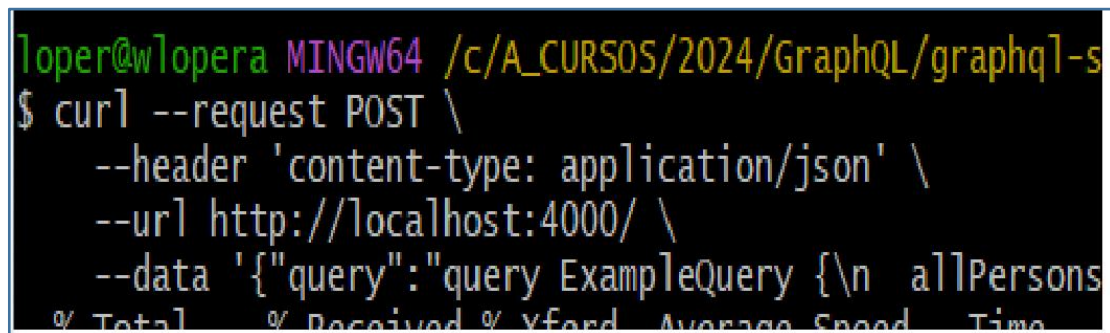


- Copiar el cURL y probar

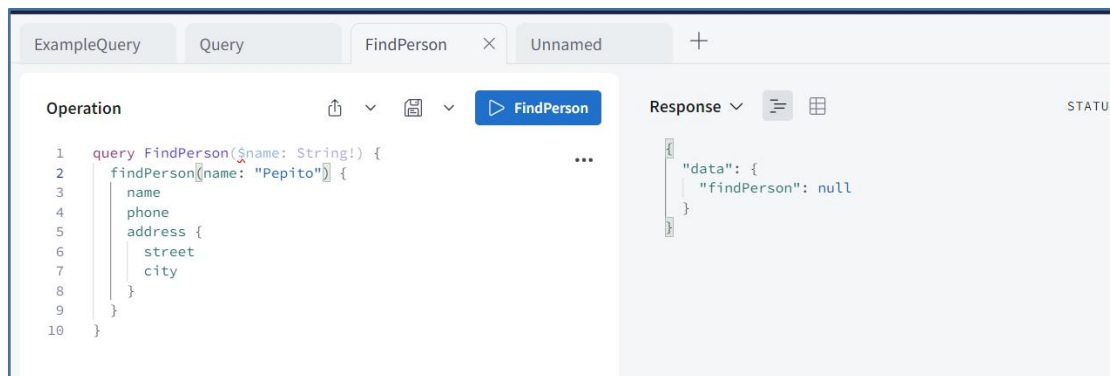


NOTA:

- 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

The screenshot shows the GraphQL Playground interface. On the left, the 'Operation' tab contains a mutation query: `mutation { addPerson(name: "wlopera", phone: "1234-5678", street: "Vista Hermosa") { name phone address { street city } } }`. The 'Variables' tab below it shows `{ "name": null, "street": null, "city": null }`. On the right, the 'Response' tab shows a JSON error response: `{ "errors": [{ "message": "Field 'addPerson' argument 'city' of type 'String!' is required, but it was not provided.", "extensions": { "code": "GRAPHQL_VALIDATION_FAILED", "exception": { "stacktrace": ["GraphQLError: Field 'addPerson' argument 'city' of type 'String!' is required, but it was not provided.", "at Object.leave (C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql\\validation\\rules\\ProvidedRequiredArgumentsRule.js:60:15)", "at Object.leave (C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql\\language\\visitor.js:324:32)", "at Object.leave (C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql\\utilities\\TypeInfo.js:411:21)", "at visit (C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql\\language\\visitor.js:197:21)", "at validate (C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql\\validation\\validate.js:91:24)", "at validate (C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\apollo-server-core\\dist\\requestPipeline.js:188:39)"] } } }] }`. The status bar at the top right indicates 'STATUS: 400 | 9.00ms | 1.4KB'.

This screenshot shows a validation error message in the GraphQL Playground: 'Field "addPerson" argument "city" of type "String!" is required, but it was not provided.' The message is displayed in a light blue box. Below the message, the 'Operation' tab shows the same mutation query as the previous screenshot. The 'Variables' tab shows `{ "name": null, "street": null, "city": null }`. The 'Response' tab shows the error response: `{ "errors": [{ "message": "Field 'addPerson' argument 'city' of type 'String!' is required, but it was not provided." }] }`.

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

The screenshot shows the Apollo GraphQL documentation page for error codes. The page title is 'Built-in error codes'. The table below lists the error codes and their descriptions.

CODE	DESCRIPTION
GRAPHQL_PARSE_FAILED	The GraphQL operation string contains a syntax error.
GRAPHQL_VALIDATION_FAILED	The GraphQL operation is not valid against the server's schema.
BAD_USER_INPUT	The GraphQL operation includes an invalid value for a field argument.
PERSISTED_QUERY_NOT_FOUND	A client sent the hash of a query string to execute via automatic persisted queries, but the query was not in the APQ cache.

On the right side of the page, there is a section titled 'On this page' with links to 'Built-in error codes', 'Custom errors', 'Throwing errors', 'Including custom error details', 'Omitting or including stacktrace', 'Masking and logging errors', 'For client responses', 'For Apollo Studio reporting', and 'Setting HTTP status code and headers'.

- 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

The image shows two screenshots of a GraphQL IDE interface, likely Apollo Studio, demonstrating the validation logic implemented in the previous code block.

Top Screenshot (Successful Mutation):

- Operation:** A GraphQL mutation is shown with variables: `name: "wlopera"`, `phone: "1234-5678"`, `street: "Vista Hermosa"`, and `city: "Panamá"`.
- Variables:** The variables are listed as `"name": null`, `"street": null`, and `"city": null`.
- Response:** The response shows a successful mutation with a status of 200. The JSON response includes the new person's data: `{ "name": "wlopera", "phone": "1234-5678", "address": { "street": "Vista Hermosa", "city": "Panamá" } }`.

Bottom Screenshot (Failed Mutation):

- Operation:** The same GraphQL mutation is shown, but with the variable `name` set to a value that already exists in the database (e.g., "wlopera").
- Response:** The response shows an error with a status of 200. The JSON response includes an error message: `"message": "Nombre debe ser único"`, indicating that the name is not unique.

- 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

The screenshot shows the GraphQL Playground interface. On the left, the 'Operation' tab is active, displaying a query named 'ExampleQuery' that requests the 'allPersons' field, including 'name', 'phone', and 'address' (with 'street' and 'city' sub-fields). Below the query editor, the 'Variables' tab shows a JSON object with 'name' set to null. On the right, the 'Response' tab displays the JSON output of the query. The response is a JSON object with a 'data' field containing an 'allPersons' array. This array lists three people: Neymar (phone: '034-1234567', address: 'Calle Frotend', city: 'Barcelona'), Messi (phone: '044-123456', address: 'Avenida Fullstack', city: 'Mataro'), and Ronaldo (phone: null, address: 'Pasaje Testing', city: 'Ibitza'). The status bar at the top right indicates 'STATUS 200'.

```
Operation
```

```
1 query ExampleQuery {
2   allPersons {
3     name
4     phone
5     address {
6       street
7       city
8     }
9   }
10 }
11
```

```
Variables
```

```
1 {
2   "name": null
3 }
```

```
Response
```

```
{
  "data": {
    "allPersons": [
      {
        "name": "Neymar",
        "phone": "034-1234567",
        "address": {
          "street": "Calle Frotend",
          "city": "Barcelona"
        }
      },
      {
        "name": "Messi",
        "phone": "044-123456",
        "address": {
          "street": "Avenida Fullstack",
          "city": "Mataro"
        }
      },
      {
        "name": "Ronaldo",
        "phone": null,
        "address": {
          "street": "Pasaje Testing",
          "city": "Ibitza"
        }
      }
    ]
  }
}
```

STATUS 200

■ Con pasar parámetro Enum “YES”

The screenshot shows a GraphQL IDE interface with tabs for 'ExampleQuery', 'Query', 'FindPerson', and 'Unnamed'. The 'Operation' tab is active, displaying a query:

```
1 query ExampleQuery {  
2   allPersons (phone: YES) {  
3     name  
4     phone  
5     address {  
6       street  
7       city  
8     }  
9   }  
10 }  
11
```

The 'Response' tab shows the JSON output:

```
{  
  "data": {  
    "allPersons": [  
      {  
        "name": "Neymar",  
        "phone": "034-1234567",  
        "address": {  
          "street": "Calle Frotend",  
          "city": "Barcelona"  
        }  
      },  
      {  
        "name": "Messi",  
        "phone": "044-123456",  
        "address": {  
          "street": "Avenida Fullstack",  
          "city": "Mataro"  
        }  
      }  
    ]  
  }  
}
```

Below the query, the 'Variables' tab shows:

```
1 {  
2   "name": null  
3 }
```

■ Con pasar parámetro Enum “NO”

The screenshot shows the same GraphQL IDE interface, but with the query changed to:

```
1 query ExampleQuery {  
2   allPersons (phone: NO) {  
3     name  
4     phone  
5     address {  
6       street  
7       city  
8     }  
9   }  
10 }  
11
```

The 'Response' tab shows the JSON output:

```
{  
  "data": {  
    "allPersons": [  
      {  
        "name": "Ronaldo",  
        "phone": null,  
        "address": {  
          "street": "Pasaje Testing",  
          "city": "Ibitza"  
        }  
      }  
    ]  
  }  
}
```

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

The screenshot shows the 'Operation' tab with the query:

```
1 query ExampleQuery {  
2   allPersons (phone: ) {  
3     name  
4     phone  
5     address {  
6       street  
7       city  
8     }  
9   }  
10 }  
11
```

A dropdown menu is open next to the 'phone' parameter, showing options: 'abc NO', 'abc YES', and 'YesNo'. The 'abc YES' option is selected.

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;
  },
},
...

```

The screenshot shows a GraphQL IDE interface with a tab labeled "EditNumber". The "Operation" pane on the left contains the following GraphQL mutation:

```
1 mutation EditNumber($name: String!, $phone: String!) { ...
2   editNumber(name: "Messi", phone: "1111-1111") {
3     name
4     phone
5   }
6 }
```

The "Response" pane on the right shows the JSON output of the mutation:

```
{
  "data": {
    "editNumber": {
      "name": "Messi",
      "phone": "1111-1111"
    }
  }
}
```

ExampleQuery × Query FindPerson Unnamed EditNumber +

Operation

```
1 query ExampleQuery {
2   allPersons (phone: YES) {
3     name
4     phone
5     address {
6       street
7       city
8     }
9   }
10 }
11
```

Response

```
{
  "data": {
    "allPersons": [
      {
        "name": "Neymar",
        "phone": "034-1234567",
        "address": {
          "street": "Calle Frotend",
          "city": "Barcelona"
        }
      },
      {
        "name": "Messi",
        "phone": "1111-1111",
        "address": {
          "street": "Avenida Fullstack",
          "city": "Mataro"
        }
      }
    ]
  }
}
```

Variables Headers Pre-Operation Script Post-Operation Script

1 { JSON

2 "name": null

Consultas compuestas (Queries anidados)

ExampleQuery Query FindPerson × Unnamed EditNumber +

Operation

```
1 query FindPerson($name: String!) {
2   findPerson(name: "Messi") {
3     name
4     phone
5     address {
6       street
7       city
8     }
9   }
10   allPersons {
11     name
12     phone
13   }
14 }
15
```

Response

```
{
  "data": {
    "findPerson": {
      "name": "Messi",
      "phone": "1111-1111",
      "address": {
        "street": "Avenida Fullstack",
        "city": "Mataro"
      }
    },
    "allPersons": [
      {
        "name": "Neymar",
        "phone": "034-1234567"
      },
      {
        "name": "Messi",
        "phone": "1111-1111"
      },
      {
        "name": "Ronaldo",
        "phone": null
      }
    ]
  }
}
```

Variables Headers Pre-Operation Script Post-Operation Script

1 { JSON

2 "name": null

3 }

- No puedo llamar a una función con argumentos diferentes

The screenshot displays the GraphQL Playground interface. On the left, the 'Query' tab is active, showing a query that finds a person by name and phone, and lists all persons. The query is as follows:

```
query FindPerson($name: String!, $phone: YesNo) {  
  findPerson(name: "Messi") {  
    name  
    phone  
    address {  
      street  
      city  
    }  
  }  
  allPersons {  
    name  
    phone  
  }  
  allPersons(phone: YES) {  
    name  
    phone  
  }  
}
```

Below the query editor, the 'Variables' tab is active, showing the following variables:

```
{  
  "name": null,  
  "phone": null  
}
```

On the right, the 'Response' tab is active, showing the JSON response. The status is 400, and the response contains an error message:

```
{  
  "errors": [  
    {  
      "message": "Fields \"allPersons\" conflict because they have  
differing arguments. Use different aliases on the fields to fetch  
both if this was intentional.",  
      "extensions": {  
        "code": "GRAPHQL_VALIDATION_FAILED",  
        "exception": {  
          "stacktrace": [  
            "GraphQLError: Fields \"allPersons\" conflict because  
they have differing arguments. Use different aliases on the fields  
to fetch both if this was intentional.",  
            "    at Object.SelectionSet  
(C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql  
\\validation\\rules\\OverlappingFieldsCanBeMergedRule.js:67:11)",  
            "    at Object.enter  
(C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql  
\\language\\visitor.js:301:32)",  
            "    at Object.enter  
(C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql  
\\utilities\\TypeInfo.js:391:27)",  
            "    at visit  
(C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql  
\\language\\visitor.js:197:21)",  
            "    at validate  
(C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql  
\\validation\\validate.js:91:24)",  
            "    at validate  
(C:\\A_CURSOS\\2024\\GraphQL\\graphql-server\\node_modules\\graphql
```

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

The image shows a VS Code editor with a REST client configuration. The top bar has tabs for "Operation" and "Response".

Operation Tab:

```
1 query FindPerson($name: String!, $phone: YesNo) {  
2   findPerson(name: "Messi") {  
3     name  
4     phone  
5     address {  
6       street  
7       city  
8     }  
9   }  
10  allPersonsData: allPersons {  
11    name  
12    phone  
13  }  
14  
15  personsWithPhone: allPersons(phone: YES) {  
16    name  
17    phone  
18  }  
19  
20 }
```

Variables Tab:

```
1 {  
2   "name": null,  
3   "phone": null  
4 }
```

Response Tab:

```
{  
  "data": {  
    "findPerson": {  
      "name": "Messi",  
      "phone": "1111-1111",  
      "address": {  
        "street": "Avenida Fullstack",  
        "city": "Mataro"  
      }  
    },  
    "allPersonsData": [  
      {  
        "name": "Neymar",  
        "phone": "034-1234567"  
      },  
      {  
        "name": "Messi",  
        "phone": "1111-1111"  
      },  
      {  
        "name": "Ronaldo",  
        "phone": null  
      }  
    ],  
    "personsWithPhone": [  
      {  
        "name": "Neymar",  
        "phone": "034-1234567"  
      },  
      {  
        "name": "Messi",  
        "phone": "1111-1111"  
      },  
      {  
        "name": "Ronaldo",  
        "phone": null  
      }  
    ]  
  }  
}
```

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 es un proyecto simple que ayuda a configurar una API REST con operaciones CRUD muy rápidamente.

_> npm i json-server

```
PS C:\A_CURSOS\2024\GraphQL\graphql-server> npm i json-server
[#####.....] / idealTree:graphql-server: timing idealTree:#root Completed in 4297ms
```

- Crear archivo db.json con la data a procesar

```
{
  "persons": [
    {
      "name": "Neymar",
      "phone": "034-1234567",
      "street": "Calle Frotend",
      "city": "Barcelona",
      "id": "3d599650-3436-11eb-8b80ba54c431"
    },
    {
      "name": "Messi",
      "phone": "044-123456",
      "street": "Avenida Fullstack",
      "city": "Mataro",
      "id": "3d599470-3436-11eb-8b80ba54c431"
    },
    {
      "name": "Ronaldo",
      "street": "Pasaje Testing",
      "city": "Ibitza",
      "id": "3d599471-3436-11eb-8b80ba54c431"
    }
  ]
}
```

- 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

```
PS C:\A_CURSOS\2024\GraphQL\graphql-server> npm i axios

added 7 packages, and audited 218 packages in 2s

34 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
...
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
      );
    },
    ...
  },
};
```

The screenshot shows the GraphQL Playground interface with a query named 'FindPerson' and its corresponding JSON response.

Operation:

```
1 query FindPerson($name: String!, $phone: YesNo) {
2   findPerson(name: "Messi") {
3     name
4     phone
5     address {
6       street
7       city
8     }
9   }
10  allPersonsData: allPersons {
11    name
12    phone
13  }
14
15  personsWithPhone: allPersons(phone: YES) {
16    name
17    phone
18  }
19
20 }
```

Response:

```
{
  "data": {
    "findPerson": {
      "name": "Messi",
      "phone": "044-123456",
      "address": {
        "street": "Avenida Fullstack",
        "city": "Mataro"
      }
    },
    "allPersonsData": [
      {
        "name": "Suarez",
        "phone": "034-1234567"
      },
      {
        "name": "Rondón",
        "phone": "044-123456"
      },
      {
        "name": "Dibu",
        "phone": null
      }
    ],
    "personsWithPhone": [
      {
        "name": "Suarez",
        "phone": "034-1234567"
      }
    ]
  }
}
```

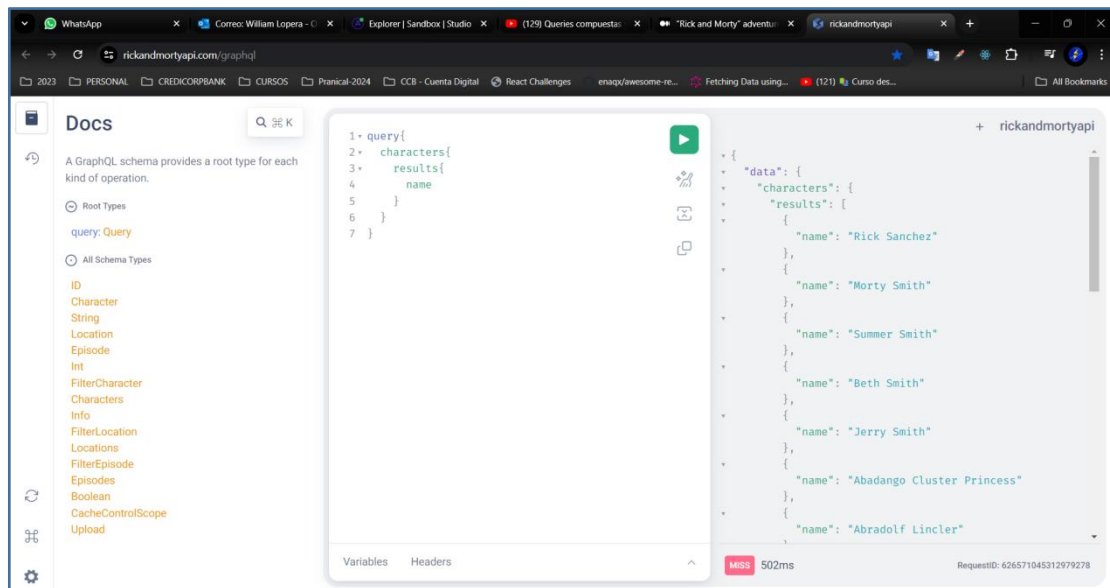
Variables:

```
1 {
2   "name": null
}
```

Buttons: Variables, Headers, Pre-Operation Script, Post-Operation Script, Add files

Ejemplo Rick and morty Api GraphQL

<https://rickandmortyapi.com/graphql>



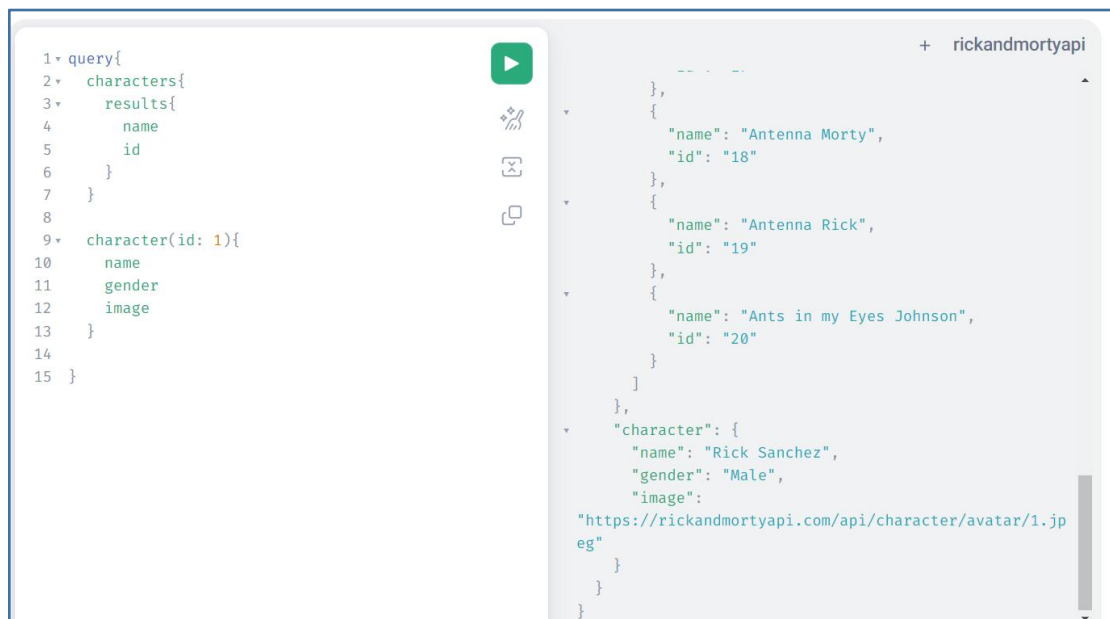
The screenshot shows the Rick and Morty API GraphQL interface. On the left, there's a sidebar with 'Docs' and a search bar. The main area displays a query and its result.

```
1 query{
2   characters{
3     results{
4       name
5     }
6   }
7 }
```

The result is a JSON object:

```
{
  "data": {
    "characters": {
      "results": [
        {
          "name": "Rick Sanchez"
        },
        {
          "name": "Morty Smith"
        },
        {
          "name": "Summer Smith"
        },
        {
          "name": "Beth Smith"
        },
        {
          "name": "Jerry Smith"
        },
        {
          "name": "Abadango Cluster Princess"
        },
        {
          "name": "Abradolf Lincler"
        }
      ]
    }
  }
}
```

At the bottom, it shows 'Variables', 'Headers', and a status bar with '502ms' and 'RequestID: 626571045312979278'.



The screenshot shows the Rick and Morty API GraphQL interface. On the left, there's a sidebar with 'Docs' and a search bar. The main area displays a query and its result.

```
1 query{
2   characters{
3     results{
4       name
5       id
6     }
7   }
8
9   character(id: 1){
10    name
11    gender
12    image
13  }
14
15 }
```

The result is a JSON object:

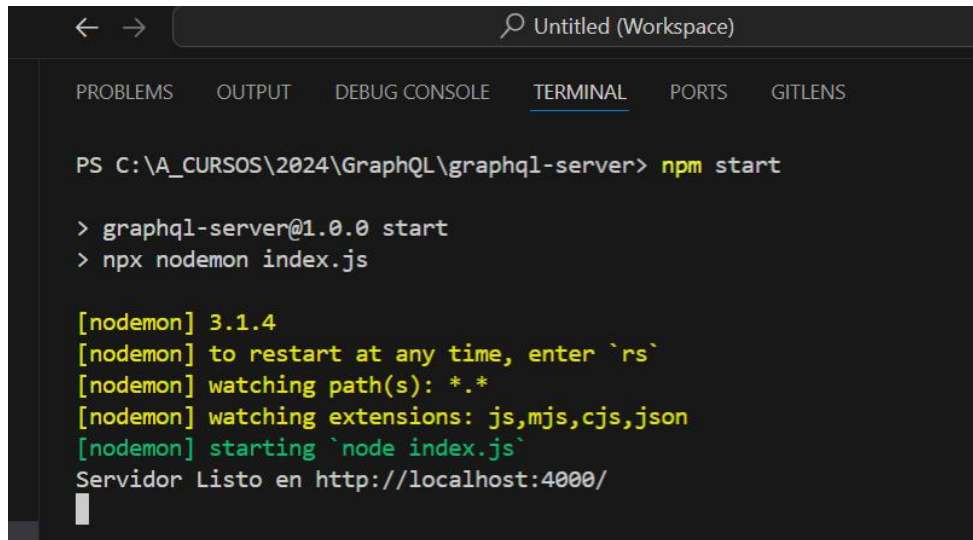
```
{
  "data": {
    "characters": {
      "results": [
        {
          "name": "Antenna Morty",
          "id": "18"
        },
        {
          "name": "Antenna Rick",
          "id": "19"
        },
        {
          "name": "Ants in my Eyes Johnson",
          "id": "20"
        }
      ]
    },
    "character": {
      "name": "Rick Sanchez",
      "gender": "Male",
      "image": "https://rickandmortyapi.com/api/character/avatar/1.jpeg"
    }
  }
}
```

At the bottom, it shows 'Variables', 'Headers', and a status bar with '502ms' and 'RequestID: 626571045312979278'.

- 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:

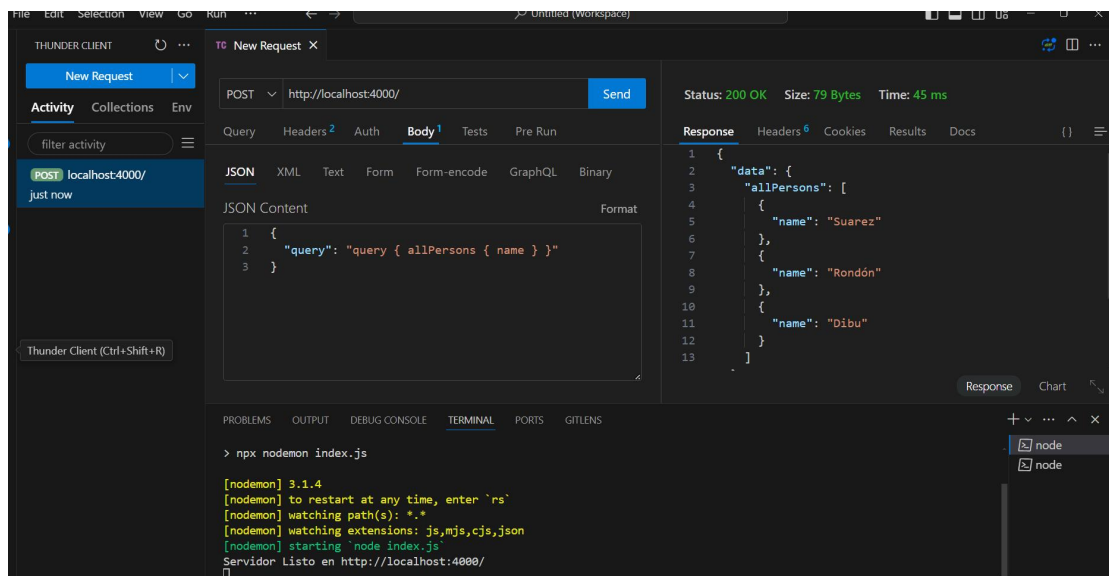


```
PS C:\A_CURSOS\2024\GraphQL\graphql-server> npm start

> graphql-server@1.0.0 start
> npx nodemon index.js

[nodemon] 3.1.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Servidor Listo en http://localhost:4000/
```

- Probar desde Thunder Client de VSCODE



The screenshot shows the Thunder Client interface with a POST request to `http://localhost:4000/`. The request body contains a GraphQL query: `query { allPersons { name } }`. The response status is `200 OK` with a size of `79 Bytes` and a time of `45 ms`. The response body is a JSON object: `{ "data": { "allPersons": [{ "name": "Suarez", "name": "Rondón", "name": "Dibu" }] }`. The terminal at the bottom shows the server running with nodemon.