



Universidad Católica  
**San Pablo**

# Ciencia de la Computación

Desarrollo Basado en Plataformas

Docente RENZO HERNAN MEDINA ZEBALLOS

GraphQL

Entregado el 15/11/2024

PACSI PONCE, ANGEL FERNANDO

Semestre III

2024-2

"El alumno declara haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo"

# GRAPHQL

## 1. ¿Qué es?

GraphQL es un lenguaje de consulta para APIs desarrollado por Facebook en 2012 y lanzado al público en 2015. Su propósito principal es permitir que los clientes obtengan exactamente los datos que necesitan de una API en lugar de recibir datos adicionales innecesarios, como sucede frecuentemente en REST.

## 2. Conceptos:

### a) Schemas:

Define la estructura de datos que se puede consultar, incluyendo sus tipos, relaciones y métodos disponibles. Se crea un "esquema" que especifica todas las queries (consultas) y mutaciones (operaciones de escritura) que se pueden ejecutar. Facilitando el desarrollo, ya que tanto el frontend como el backend trabajan con una estructura de datos claramente definida.

```
4  const typeDefs = gql`
5    type Query {
6      saludo(nombre: String): String
7    }
8  `;
9
```

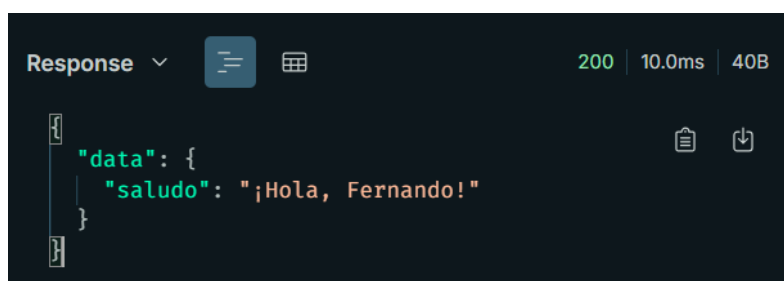
### b) Queries:

Son las consultas que hace el cliente para obtener datos. Cada query permite especificar los campos exactos que se necesitan, lo que evita que el servidor envíe información no requerida.



The screenshot shows a GraphQL client interface with a dark theme. At the top, there's a header with the word "Operation" and several icons (upload, dropdown, save, dropdown, and a "Run" button). Below the header, a query is entered in a text area:

```
1  query {
2    saludo(nombre: "Fernando")
3  }
4
```



The screenshot shows the response section of the GraphQL client interface. It has a header with the word "Response", a dropdown menu, and icons for JSON, JSON Schema, and a table view. To the right, it shows the status "200", the time "10.0ms", and the size "40B". Below the header, the response is displayed in a text area:

```
{
  "data": {
    "saludo": "¡Hola, Fernando!"
  }
}
```

### c) Resolver:

Funciones que "resuelven" las queries y mutaciones. Cuando un cliente solicita datos a través de una query, el resolver es el responsable de buscar esos datos en la base de datos u otra fuente de datos y devolverlos al cliente. Cada campo de una query tiene su resolver, que es ejecutado cuando el campo se consulta.

```
11  const resolvers = {
12    Query: {
13      saludo: (_, { nombre }) => {
14        return `¡Hola, ${nombre || 'desconocido'}!`;
15      },
16    },
17  };
18
```

## 3. GraphQL vs. REST APIs (Ejemplo al final del documento)

### Flexibilidad:

- **GraphQL:** El cliente decide qué datos necesita. Las consultas son más específicas y el servidor solo devuelve los campos solicitados.
- **REST:** El servidor decide qué datos se envían. A menudo, uno puede recibir más datos de los que realmente se necesita (over-fetching) o menos de lo que uno espera (under-fetching).

### Rendimiento:

- **GraphQL:** Permite hacer una sola solicitud que regresa exactamente los datos necesarios. Evita múltiples peticiones para obtener información de diferentes fuentes.
- **REST:** A menudo es necesario hacer múltiples solicitudes a distintos endpoints para obtener toda la información.

**Versionado:**

- **GraphQL:** No requiere crear nuevas versiones de los endpoints. Se puede modificar el esquema sin crear versiones adicionales.
- **REST:** Suele necesitar nuevas versiones de los endpoints cuando los requisitos cambian, lo que puede complicar la evolución de la API.

**4. Ventajas y desventajas****a. Ventajas:**

- i. Precisión en las consultas: el cliente recibe solo los datos necesarios.
- ii. Menos peticiones: una sola petición en GraphQL puede sustituir varias de REST.
- iii. Mejora el rendimiento en aplicaciones móviles y de baja conectividad.

**b. Desventajas:**

- i. Complejidad en el aprendizaje y la implementación.
- ii. Problemas de caché y optimización que REST resuelve mejor.
- iii. Puede generar sobrecarga en el servidor si las consultas son muy extensas.

## GraphQL.js

```
JS graphql.js > ...
1  const { ApolloServer, gql } = require('apollo-server');
2
3  const typeDefs = gql`
4    type Query {
5      saludo(nombre: String): String
6    }
7  `;
8
9  const resolvers = {
10    Query: {
11      saludo: (_, { nombre }) => {
12        return `¡Hola, ${nombre || 'desconocido'}!`;
13      },
14    },
15  };
16
17  const server = new ApolloServer({
18    typeDefs,
19    resolvers,
20  });
21
22  server.listen().then(({ url }) => {
23    console.log(`Servidor listo en ${url}`);
24  });
25
```

## RestAPI.js

```
JS restapi.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 4001;
4
5  app.get('/saludo', (req, res) => {
6    const nombre = req.query.nombre || 'desconocido';
7    res.json({ mensaje: `¡Hola, ${nombre}!` });
8  });
9
10 app.listen(port, () => {
11   console.log(`Servidor REST listo en http://localhost:${port}`);
12 });
13
```

localhost:4001/saludo?nombre=Fernando

Impresión con formato estilístico ☒

```
{
  "mensaje": "¡Hola, Fernando!"
}
```

Response ▾



200 14.0ms 40B

```
{
  "data": {
    "saludo": "¡Hola, Fernando!"
  }
}
```