# Documentando APIS de GraphQL

**Liliana Iriarte Sanabria**
*Senior Technical Writer*

# Temario

1. ¿Qué es GraphQL?
2. Características de GraphQL
3. ¿REST o GraphQL?
4. Estructura básica de GraphQL
5. El rol de un Technical Writer en la documentación de Graphql
6. Exploradores de documentación de GraphQL (IDEs)
    - GraphQL Playground como herramienta base de documentación
    - Ejemplos de documentación usando GraphQL Playground

# 1. Que es GraphQL?

Es un lenguaje de consultas y operaciones para APIs diseñado para que sea flexible y declarativo.

## Lenguajes soportados

| | | | | |
|---|---|---|---|---|
| Javascript | PHP | Java/Kotlin | C#/ .NET | Python |
| Rust | Ruby | Elixir | Swift | Scala |
| C/C++ | Haskel | Groovy | Julia | Perl |

**Server / Client / Tools**

# 2. Características de GraphQL

# Caracteristicas

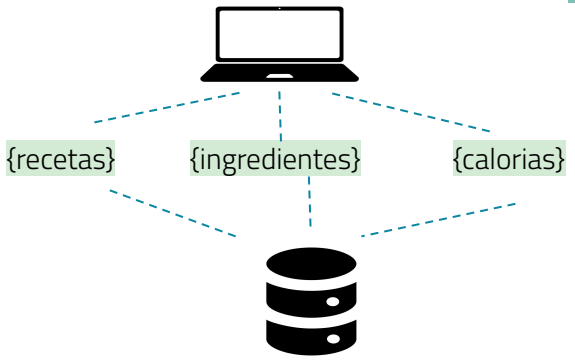
- Es usado como alternativa a las APIs de Rest
- Puede implementarse en más de 20 lenguajes de programación.
- No está vinculado a una base de datos específica
- Permite pedir recursos anidados en la misma operación, evitando la necesidad de peticiones en cascada al estilo REST
- Puede devolver consultas mediante el uso de una query
- Puede modificar data usando una mutación

# 3. ¿REST o GraphQL?

## GraphQL


- Utiliza una sola petición
- **Se auto documenta desde el desarrollo**
- No usa verbos HTTP para determinar el tipo de solicitud.
- Comunicación Sync/Asyn en múltiples protocolos (HTTP, AMQP, MQTT)
- Facilita la supervisión del uso de campos
- Payload en JSON

## REST

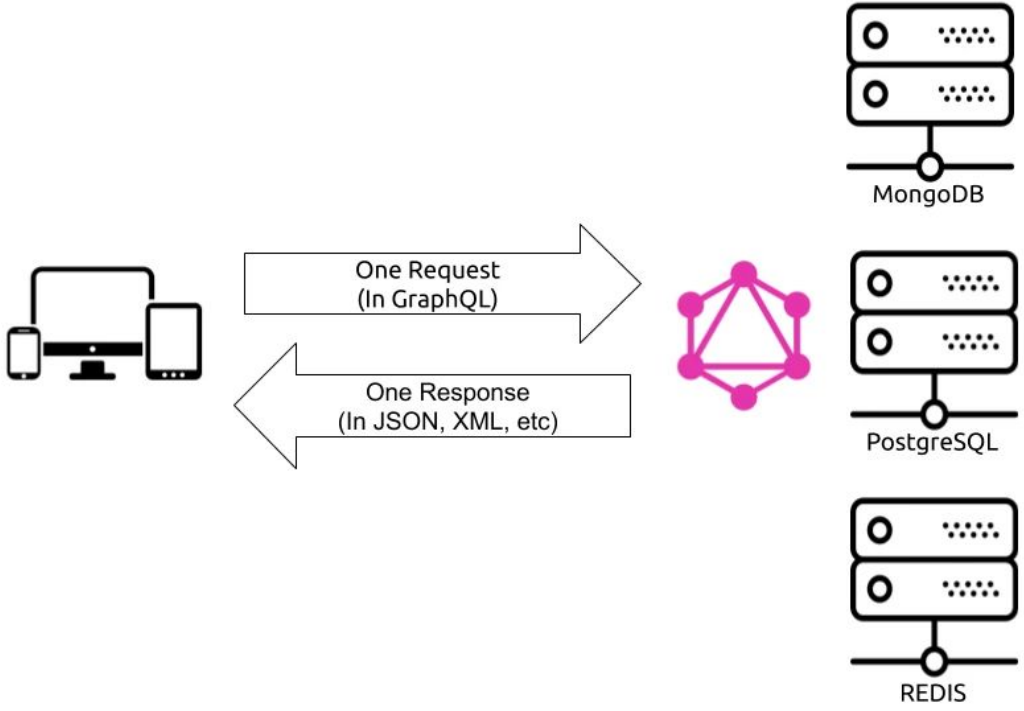



- Utiliza varias peticiones
- **Se documenta después del desarrollo**
- Usa verbos HTTP para distinguir acciones de lectura (GET) y escribir acciones (POST,PUT,DELETE
- Comunicación Sync a través de HTTP
- No hay forma de determinar si el cliente necesita un campo
- Payload en JSON

# ¿Cual es mejor?

# 4. Estructura basica de GraphQL?

# Comunicación en GraphQL



# Conceptos Básicos

## Schema

Sección donde se definen los tipos de operaciones, tipo de objeto que se puede solicitar y los campos que lo contienen.

```
schema {
  query: Query
  mutation: Mutation
}
```

```
type User {
  email: String!
  id: Int!
  name: String!
  posts: [Post!]!
}
```

# Conceptos Básicos

## Query

Operación que se encarga de hacer consultas al servidor, utilizando solo los campos que necesita. Es el equivalente al GET de REST

# Conceptos Básicos

## Ejemplo

```
query {
  users{
    id
    name
    phone
    email
  }
}
```

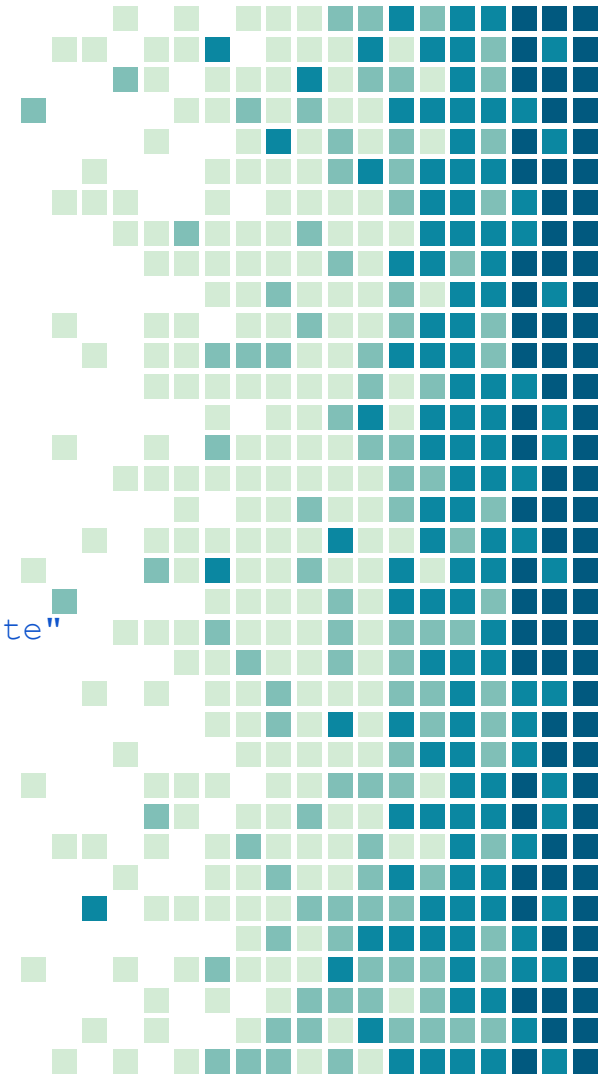
```
{
  "data": {
    "users": [
      {
        "id": 1,
        "name": "Liliana Iriarte ",
        "phone": "12345",
        "email": "liliana@gmail.com"
      },
      {
        "id": 2,
        "name": "Carlos Fernandez",
        "phone": "12345678",
        "email": "carlos@gmail.com"

      }
    ]
  }
}
```

# Conceptos Básicos

## Ejemplo

```
query {
  user(id:1){
    city
    username
   }
}
```

```
{
  "data": {
    "user": [
      {
        "city": "La Paz",
        "username": "lilianairiarte"
      }
    ]
  }
}
```



# Conceptos Básicos

## Ejemplo

```
{
  user{
    id
    name
    phone
    company {
      id
      name
    }
  }
}
```

```
{
  "data": {
    "user": [
      {
        "id": 1,
        "name": "Liliana Iriarte",
        "phone": "12345",
        "company": [
          {
           "id": "1",
           "name": "mojix",
           }
          ],
       }
      ]
    }
}
```

# Conceptos Básicos

## Mutation

Operación que se encarga de modificar los datos que se envían. Es el equivalente a POST, PUT y DELETE en REST

# Conceptos Básicos

## Ejemplo

```
mutation {
  signupUser(
      name: "liliana",
      email: "liliana@gmail.com"
  ) {
    id
  }
}
```

```
{
  "data": {
    "signupUser": {
      "id": 4
    }
  }
}
```

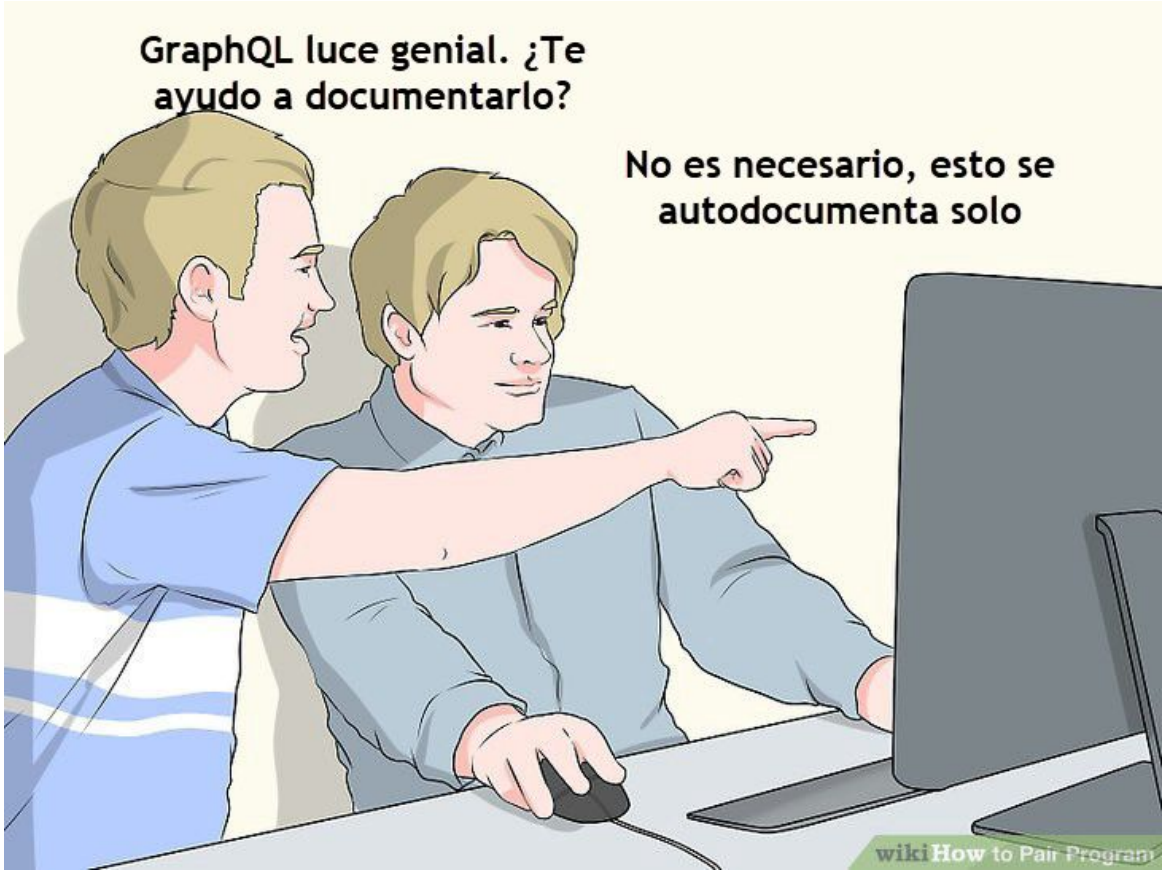
# Conceptos Básicos

## Ejemplo

```
mutation {
  addUser(username: "liriarte",
city: "Bolivia"){
    success
    message
    id
  }


}
```

```
{
  "data": {
    "adduser": {
      "success": true,
      "message": "User created
successfully",
      "id": 1

    }

  }
}
```

# 5. El rol de un Technical Writer en la documentación de Graphql

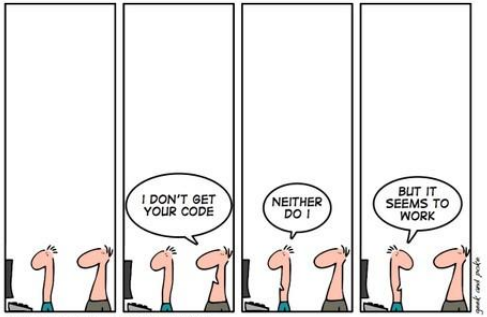

GraphQL luce genial. ¿Te ayudo a documentarlo?
No es necesario, esto se autodocumenta solo
wikiHow to Pair Program

La documentación ayuda a entender cómo funcionan queries y mutaciones cuando no tienes conocimiento de lo que es GraphQL.

- Getting started
- Tutoriales
- Documentación conceptual
- Como funciona cada endpoint
  - Tipos
  - Campos
  - Atributos
  - Consultas
  - Mutaciones
  - etc

GraphQL puede ser auto documentado, si solo si:

- Se trabaja en usar  nombres descriptivos
- Si se añaden descripciones correctas
- Provee de suficiente documentación de soporte, especialmente para personas que están aprendiendo GraphQL




El rol que cumple un technical writer es importante por que se asegura que lo que se documenta este funcionando

Si el API no tiene documentacion, las personas no podrán usarlo en su totalidad

# 6. Exploradores de documentación de GraphQL (IDEs)

# GraphiQL

- Finalización automática de consultas
- Resaltado de sintaxis
- Depuración a medida que escribe
- Explorador de documentación
- JSON Viewer
- El trabajo es fácilmente compartible

# GraphQL Playground

- Explorador de esquemas de columnas múltiples
- Encabezados personalizados
- Esquemas de color
- Pestañas
- Historial de consultas
- Fácilmente compartible para colaboraciones (Graphql Bin)

# GRACIAS!

@liliiriarte

liliana.iriarte.sanabria@gmail.com

# REFERENCIAS


- https://graphql.org/
- https://perspectives.mobilelive.ca/
- https://idratherbewriting.com/learnapidoc/docapis_graphql_apis.html
- https://medium.com/@scottydocs/graphql-if-theres-no-documentation-people-aren-t-going-to-be-able-to-use-it-5c156641bb20
- https://codesandbox.io/
- https://graphql.org/graphql-js/
- https://www.npmjs.com/package/graphql