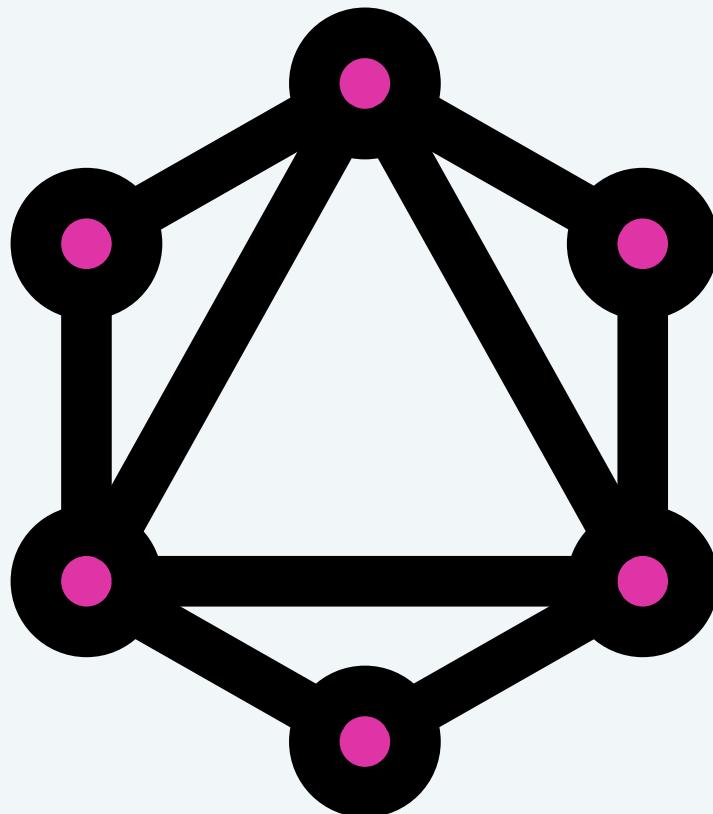




GraphQL API

GraphQL desde Cero: La guía más completa y todo lo que necesitas para dominar APIs Modernas



PABLO DEL ÁLAMO



Introducción

GraphQL ha llegado para revolucionar la forma en que hacemos APIs.

No solo es más flexible y potente, sino que también te permite ser mucho más específico en las peticiones.

¡Vamos a desglosar qué es y por qué deberías considerarlo en tu próximo proyecto!



PABLO DEL ÁLAMO



¿Qué es GraphQL?

GraphQL es un lenguaje de consulta para tu API que te permite obtener exactamente lo que necesitas.

Supone un cambio de paradigma respecto a REST, donde defines las relaciones de datos y consultas en un solo lugar.

Vamos, perfecto para quien no quiere andar mandando mil peticiones.

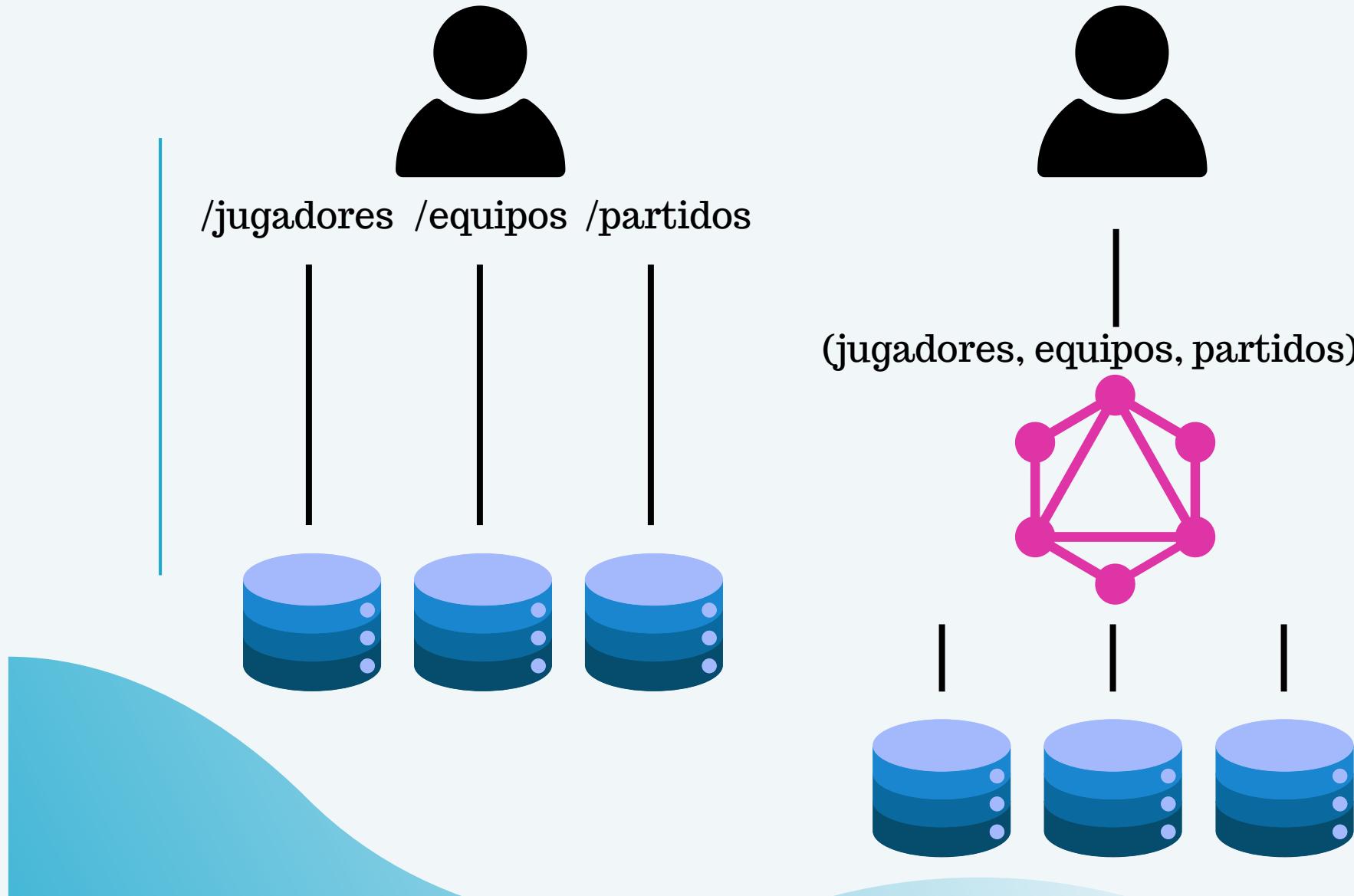


PABLO DEL ÁLAMO



Rest API

GraphQL API





Ventajas de GraphQL

Aquí van las claves:

Flexibilidad, porque decides qué datos recibir.

Depuración más sencilla, pues ves qué datos pides y recibes en cada paso.

Adiós versionado: Olvídate de mantener distintas versiones de tu API, ¡los días de 'v1', 'v2', 'v3' se acabaron!



PABLO DEL ÁLAMO



Un ejemplo rápido de una consulta

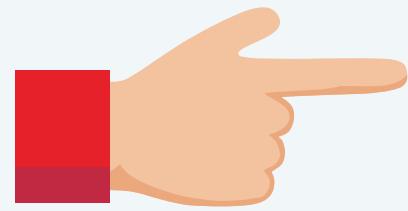
Imagina que necesitas datos de los jugadores, equipos y partidos que se dieron el sábado pasado.

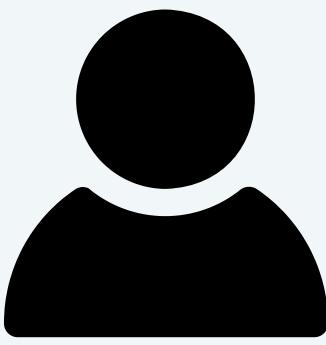
Con REST, múltiples endpoints serían necesarios.

En GraphQL, una sola consulta podría lucir así, proporcionando control total y eficiencia.

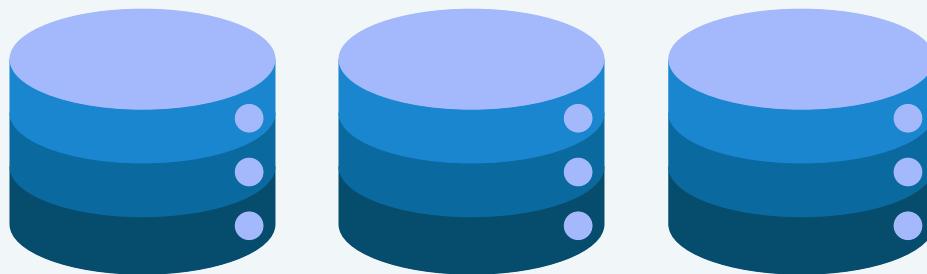
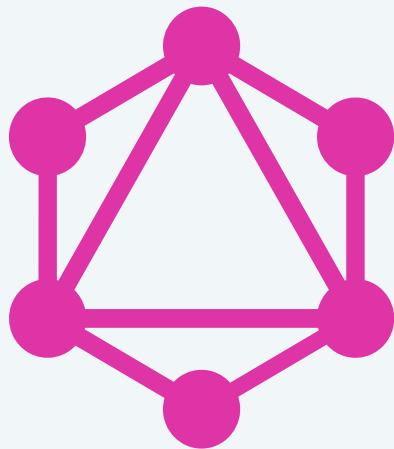


PABLO DEL ÁLAMO





(jugadores, equipos, partidos)





Respuestas a medida

GraphQL responde solo con lo que has pedido.

Ni más, ni menos. Piensa en una pizza: solo pides los ingredientes que te gustan, sin sorpresas.

Ideal para conservar recursos y ahorrar procesamiento.



PABLO DEL ÁLAMO



Componentes principales

GraphQL se sostiene sobre 3 pilares: Schema, Queries y Mutations.

El schema define la estructura y las relaciones;

Queries obtienen los datos, y Mutations los modifican.

Todo interconectado para un flujo de trabajo fluido.



PABLO DEL ÁLAMO



El Schema

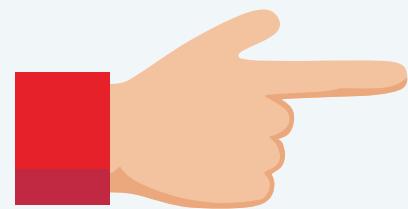
El Schema es la definición de los tipos de datos y las relaciones entre ellos en una API de GraphQL.

Es la estructura o "contrato" que define qué datos están disponibles en la API, cómo se relacionan entre sí y qué operaciones pueden realizarse sobre ellos.

En GraphQL, un esquema se define mediante tipos, que incluyen tipos de objeto (como Player, Match, y Team en el caso de una API deportiva), y las operaciones Query y Mutation que representan las acciones posibles sobre esos datos.



PABLO DEL ÁLAMO



```
1 type Player {  
2   id: ID!  
3   name: String!  
4   position: String!  
5   stats: Stats!  
6 }  
7  
8 type Stats {  
9   goals: Int!  
10  assists: Int!  
11  yellowCards: Int!  
12 }  
13  
14 type Team {  
15   id: ID!  
16   name: String!  
17   players: [Player]!  
18 }  
19  
20 type Match {  
21   id: ID!  
22   date: String!  
23   teams: [Team]!  
24   score: Score!  
25 }  
26  
27 type Score {  
28   team1: Int!  
29   team2: Int!  
30 }  
31  
32 type Query {  
33   matches(date: String!): [Match]!  
34   player(id: ID!): Player  
35   team(id: ID!): Team  
36 }  
37  
38 type Mutation {  
39   addPlayerToTeam(playerId: ID!, teamId: ID!): Team  
40   updatePlayerStats(playerId: ID!, goals: Int!, assists: Int!, yellowCards: Int!): Player  
41 }  
42
```



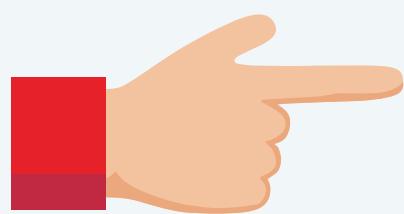
Ejemplo de código de una petición

Vamos a suponer que tienes un esquema con los tipos Player, Match y Team, y que cada partido (Match) tiene datos sobre los equipos que jugaron, los jugadores destacados y otros detalles como el resultado final.

Este es un ejemplo de una consulta en GraphQL para obtener los datos deseados. La consulta solicita información sobre los partidos que se jugaron el sábado pasado, incluyendo detalles sobre los equipos y los jugadores destacados.



PABLO DEL ÁLAMO



```
1 query GetMatchesFromLastSaturday {  
2   matches(date: "2024-10-26") { # Aquí estamos especificando el sábado pasado  
3     id  
4     date  
5     teams {  
6       id  
7       name  
8       players {  
9         id  
10        name  
11        position  
12        stats {  
13          goals  
14          assists  
15          yellowCards  
16        }  
17      }  
18    }  
19    score {  
20      team1  
21      team2  
22    }  
23    highlights {  
24      player {  
25        id  
26        name  
27      }  
28      action  
29      minute  
30    }  
31  }  
32}  
33
```

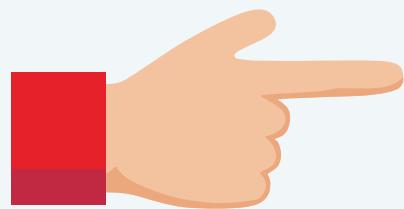


Ejemplo de código de una respuesta

- matches: Una lista de partidos jugados en la fecha especificada.
- teams: Para cada partido, incluye detalles sobre los equipos participantes.
- players: En cada equipo, proporciona información sobre los jugadores principales, incluyendo estadísticas relevantes.
- score: Muestra el puntaje final del partido.
- highlights: Las jugadas más importantes de cada partido, mostrando el minuto, la acción realizada y el jugador responsable.



PABLO DEL ÁLAMO



```
        "matches": [
            {
                "id": "match123",
                "date": "2024-10-26",
                "teams": [
                    {
                        "id": "teamA",
                        "name": "FC Barcelona",
                        "players": [
                            {
                                "id": "player1",
                                "name": "Lionel Messi",
                                "position": "Forward",
                                "stats": {
                                    "goals": 2,
                                    "assists": 1,
                                    "yellowCards": 0
                                }
                            },
                            {
                                "id": "player2",
                                "name": "Marc-André ter Stegen",
                                "position": "Goalkeeper",
                                "stats": {
                                    "goals": 0,
                                    "assists": 0,
                                    "yellowCards": 1
                                }
                            }
                        ]
                    },
                    [
                        {
                            "id": "teamB",
                            "name": "Real Madrid",
                            "players": [
                                {
                                    "id": "player3",
                                    "name": "Karim Benzema",
                                    "position": "Forward",
                                    "stats": {
                                        "goals": 1,
                                        "assists": 0,
                                        "yellowCards": 0
                                    }
                                },
                                {
                                    "id": "player4",
                                    "name": "Thibaut Courtois",
                                    "position": "Goalkeeper",
                                    "stats": {
                                        "goals": 0,
                                        "assists": 0,
                                        "yellowCards": 0
                                    }
                                }
                            ]
                        ]
                    ],
                    "score": {
                        "team1": 3,
                        "team2": 1
                    },
                    "highlights": [
                        {
                            "player": {
                                "id": "player1",
                                "name": "Lionel Messi"
                            },
                            "action": "Goal",
                            "minute": 23
                        },
                        {
                            "player": {
                                "id": "player3",
                                "name": "Karim Benzema"
                            },
                            "action": "Goal",
                            "minute": 67
                        }
                    ]
                ]
            }
        ]
```



Autenticación en GraphQL

La autenticación en GraphQL no se diferencia mucho de otras APIs, pero requiere un poco de estrategia.

Generalmente, se utiliza JWTs (JSON Web Tokens) o middleware para manejar la autenticación. Vamos a ver cómo.



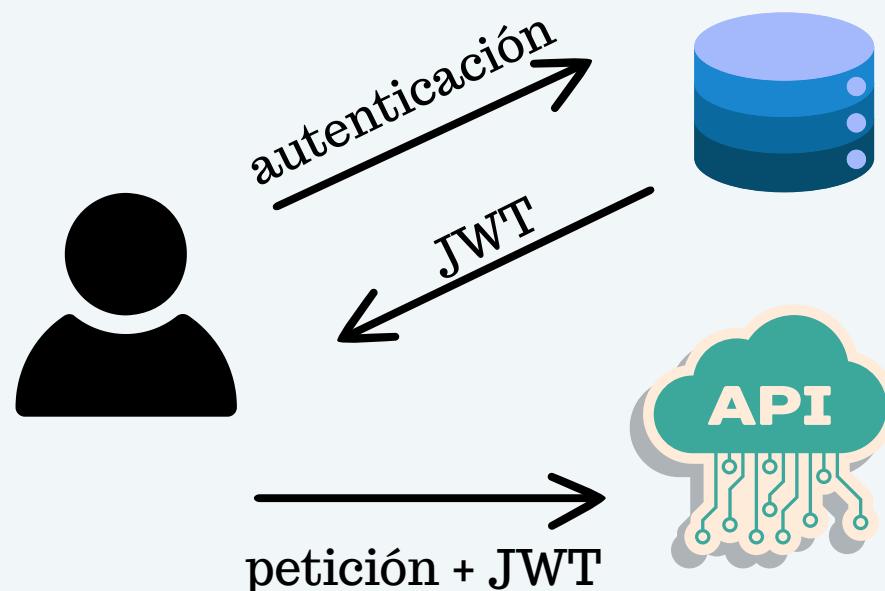
PABLO DEL ÁLAMO



Autenticación con JWT

JWT son fichas compactas y seguras usadas para verificar identidades de usuario.

En GraphQL, estas fichas se incluyen en las cabeceras de las peticiones. ¡Vamos a ver un ejemplo!

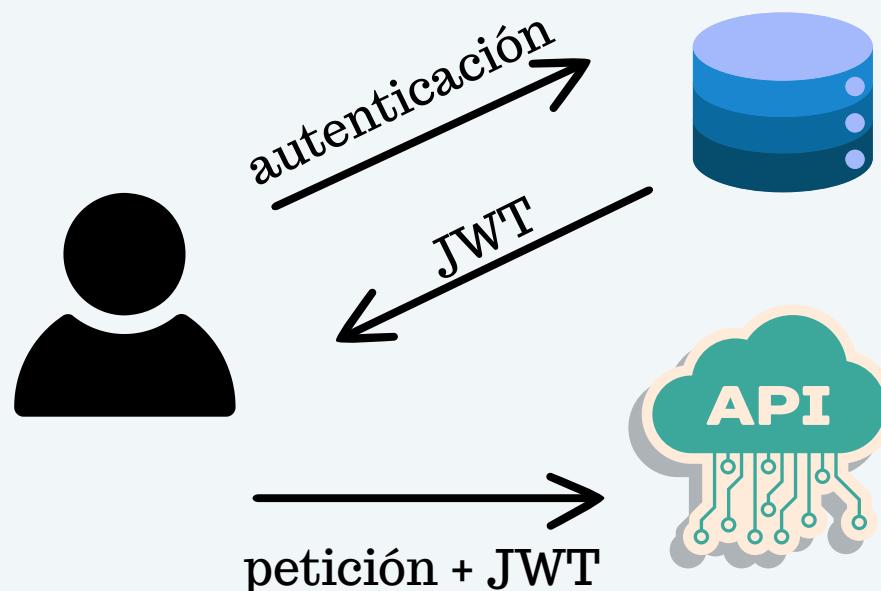


PABLO DEL ÁLAMO



Flujo básico de JWT

- 1** El usuario inicia sesión y recibe un token JWT.
- 2** El cliente envía peticiones con el token en las cabeceras.
- 3** El servidor verifica el token y procesa la solicitud si es válido.



PABLO DEL ÁLAMO



Ventajas de usar JWT

Stateless: No necesitas almacenar sesión en el servidor.

Portable: Funciona con cualquier sistema y lenguaje.

Seguro: La carga útil está firmada, evitando manipulaciones.



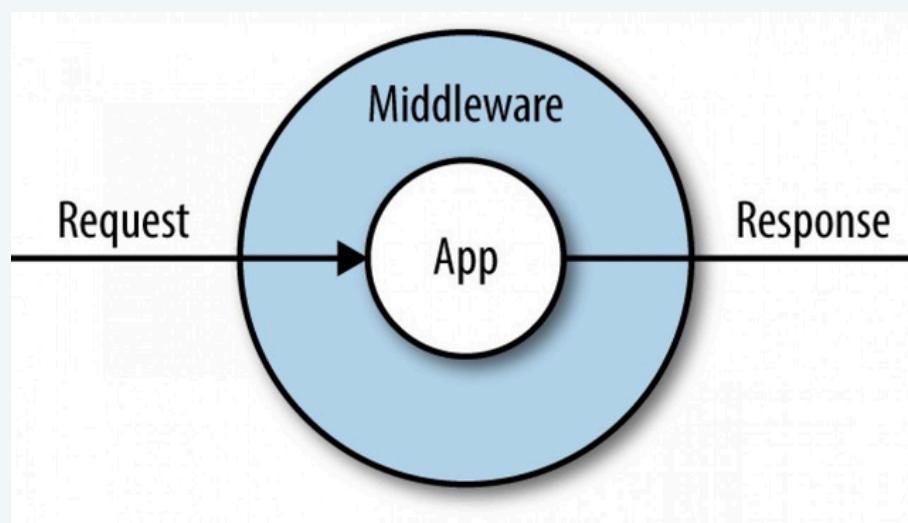
PABLO DEL ÁLAMO



Middleware de autenticación

Un middleware puede interceptar peticiones para comprobar autenticación.

¡Perfecto para proteger ciertas rutas o tipos de datos en tu API!



PABLO DEL ÁLAMO



Seguridad en GraphQL: Principios básicos

GraphQL permite realizar peticiones complejas, pero ojo: esto puede ser un arma de doble filo si no se gestiona bien.

Veamos algunos consejos para mantener seguro tu servidor.



PABLO DEL ÁLAMO



Limitar la profundidad de consultas

GraphQL permite consultas anidadas fácilmente, lo que podría dar lugar a consultas demasiado profundas, impactando el rendimiento.

Usa herramientas como 'graphql-depth-limit' para poner límites.



PABLO DEL ÁLAMO



Rate limiting

Rate limiting en GraphQL es una técnica que controla la cantidad de solicitudes o consultas que un usuario o cliente puede hacer en un período de tiempo específico.

Esto es fundamental para proteger la API de abusos, optimizar el uso de recursos del servidor y asegurar que el servicio sea eficiente para todos los usuarios.

En una API REST tradicional, el rate limiting se suele basar en el número total de solicitudes (por ejemplo, máximo 1000 solicitudes por hora por usuario).

Sin embargo, en GraphQL, las consultas pueden variar significativamente en complejidad; algunas consultas pueden ser muy simples y otras muy complejas, lo que hace que el rate limiting sea más complicado.



Registro y monitoreo

Registra y monitoriza las consultas para detectar posibles abusos o problemas.

Herramientas como Apollo Server pueden ayudarte a identificar patrones inusuales.



PABLO DEL ÁLAMO



Validación de esquemas

Asegúrate de que tu esquema y resolvers están correctamente validados.

Esto previene posibles ataques debido a errores o configuraciones incorrectas.



PABLO DEL ÁLAMO



Ejemplo completo implementación GraphQL API

Vamos a implementar una API GraphQL completa, que gestione datos de partidos, jugadores y equipos, usando Node.js con Express y Apollo Server.

Estos son algunos de los marcos más usados, y te llevaré a través de cada paso necesario para configurar y ejecutar esta API.



PABLO DEL ÁLAMO



Preparar el Entorno

Asegúrate de tener Node.js instalado y luego crea un proyecto nuevo:

```
mkdir graphql-sports-api  
cd graphql-sports-api  
npm init -y
```



PABLO DEL ÁLAMO



Instalar las Dependencias Necesarias

Instala Apollo Server, Express y GraphQL:

```
npm install apollo-server express graphql
```

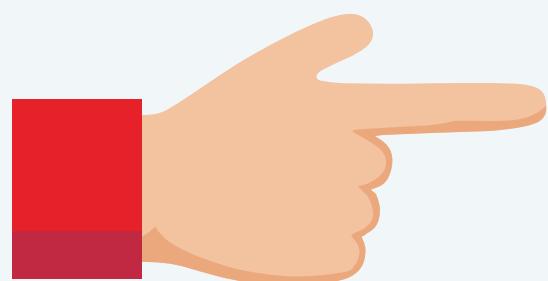


PABLO DEL ÁLAMO



Definir el Esquema (Schema)

En GraphQL, el esquema define los tipos de datos y las relaciones entre ellos. Creamos un archivo schema.js en el directorio raíz.



PABLO DEL ÁLAMO

```
1 const { gql } = require('apollo-server-express');
2
3 const typeDefs = gql`  

4   type Query {  

5     players: [Player]  

6     teams: [Team]  

7     matches: [Match]  

8     team(id: ID!): Team  

9     player(id: ID!): Player
10   }
11
12   type Mutation {  

13     addPlayer(name: String!, position: String!, teamId: ID!): Player
14   }
15
16   type Player {  

17     id: ID!
18     name: String!
19     position: String!
20     team: Team
21   }
22
23   type Team {  

24     id: ID!
25     name: String!
26     players: [Player]
27     matches: [Match]
28   }
29
30   type Match {  

31     id: ID!
32     date: String!
33     homeTeam: Team!
34     awayTeam: Team!
35     homeScore: Int!
36     awayScore: Int!
37   }
38 `;
39
40 module.exports = typeDefs;
```



Crear Datos Simulados (Mock Data)

Creamos un archivo data.js para simular una base de datos en memoria:

```
1 const players = [
2   { id: '1', name: 'Player 1', position: 'Forward', teamId: '1' },
3   { id: '2', name: 'Player 2', position: 'Midfielder', teamId: '1' },
4   { id: '3', name: 'Player 3', position: 'Defender', teamId: '2' },
5 ];
6
7 const teams = [
8   { id: '1', name: 'Team A' },
9   { id: '2', name: 'Team B' },
10];
11
12 const matches = [
13   { id: '1', date: '2023-10-01', homeTeamId: '1', awayTeamId: '2', homeScore: 3, awayScore: 1 },
14   { id: '2', date: '2023-10-10', homeTeamId: '2', awayTeamId: '1', homeScore: 2, awayScore: 2 },
15 ];
16
17 module.exports = { players, teams, matches };
18 |
```



PABLO DEL ÁLAMO



Resolver las Consultas (Resolvers)

Los resolvers indican cómo obtener los datos solicitados en las queries y mutations. Crearemos un archivo resolvers.js

Este archivo define:

- Resolvers para las Queries: como obtener los players, teams, matches, o un equipo o jugador específico.
- Resolvers para la Mutación: addPlayer permite agregar un jugador a un equipo específico.
- Resolvers para campos específicos: para relacionar datos (e.g., Team.players), y definir cómo se obtiene el equipo de cada jugador o los equipos de cada partido.



PABLO DEL ÁLAMO

```
1 const { players, teams, matches } = require('./data');
2
3 const resolvers = {
4   Query: {
5     players: () => players,
6     teams: () => teams,
7     matches: () => matches,
8     team: (_, { id }) => teams.find((team) => team.id === id),
9     player: (_, { id }) => players.find((player) => player.id === id),
10   },
11   Mutation: {
12     addPlayer: (_, { name, position, teamId }) => {
13       const newPlayer = {
14         id: `${players.length + 1}`,
15         name,
16         position,
17         teamId,
18       };
19       players.push(newPlayer);
20       return newPlayer;
21     },
22   },
23   Team: {
24     players: (team) => players.filter((player) => player.teamId === team.id),
25     matches: (team) =>
26       matches.filter(
27         (match) => match.homeTeamId === team.id || match.awayTeamId === team.id
28       ),
29   },
30   Match: {
31     homeTeam: (match) => teams.find((team) => team.id === match.homeTeamId),
32     awayTeam: (match) => teams.find((team) => team.id === match.awayTeamId),
33   },
34   Player: {
35     team: (player) => teams.find((team) => team.id === player.teamId),
36   },
37 };
38
39 module.exports = resolvers;
40
```



Configurar el Servidor Express con Apollo Server

Ahora configuramos el servidor en index.js:

```
const express = require('express');
const { ApolloServer } = require('apollo-server-express');
const typeDefs = require('./schema');
const resolvers = require('./resolvers');

const startServer = async () => {
  const app = express();
  const server = new ApolloServer({ typeDefs, resolvers });

  await server.start();
  server.applyMiddleware({ app });

  app.listen({ port: 4000 }, () =>
    console.log(`Server running at http://localhost:4000${server.graphqlPath}`)
  );
};

startServer();
```



PABLO DEL ÁLAMO



Ejecutar el Servidor

En la terminal, ejecuta el siguiente comando para iniciar el servidor: `node index.js`

La API GraphQL estará disponible en <http://localhost:4000/graphql>.

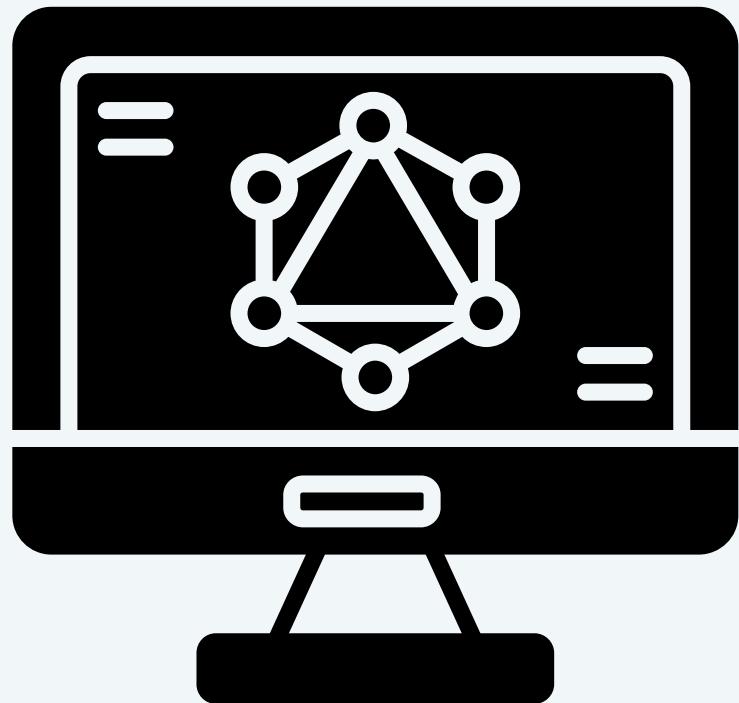


PABLO DEL ÁLAMO



Ejemplos de Uso

Ahora puedes interactuar con la API usando consultas GraphQL.



PABLO DEL ÁLAMO



Ejemplo de Query para Obtener Todos los Jugadores

```
query {  
  players {  
    id  
    name  
    position  
    team {  
      name  
    }  
  }  
}
```



PABLO DEL ÁLAMO



Ejemplo de Query para Obtener un Equipo y sus Partidos

```
query {
  team(id: "1") {
    name
    players {
      name
    }
    matches {
      date
      homeTeam {
        name
      }
      awayTeam {
        name
      }
      homeScore
      awayScore
    }
  }
}
```



PABLO DEL ÁLAMO



Ejemplo de Mutation para Aregar un Nuevo Jugador

```
mutation {
  addPlayer(name: "Player 4", position: "Goalkeeper", teamId: "1") {
    id
    name
    team {
      name
    }
  }
}
```



PABLO DEL ÁLAMO



Ejemplo de Respuesta a la Query de Equipo y Partidos

```
{
  "data": {
    "team": {
      "name": "Team A",
      "players": [
        {
          "name": "Player 1"
        },
        {
          "name": "Player 2"
        }
      ],
      "matches": [
        {
          "date": "2023-10-01",
          "homeTeam": { "name": "Team A" },
          "awayTeam": { "name": "Team B" },
          "homeScore": 3,
          "awayScore": 1
        }
      ]
    }
  }
}
```



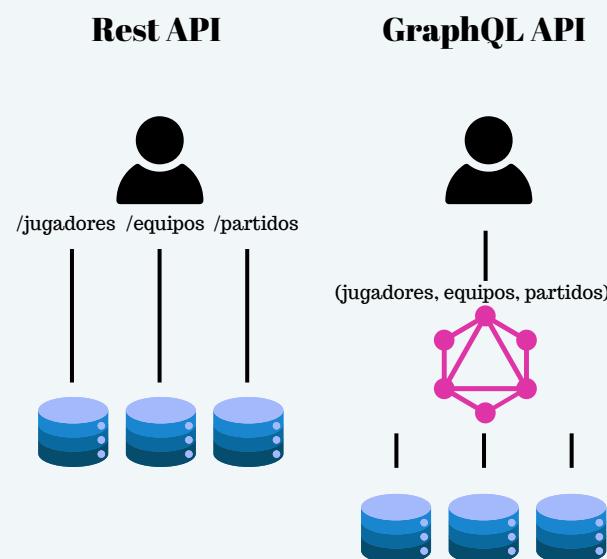
PABLO DEL ÁLAMO



Comparativa General: GraphQL vs REST

GraphQL y REST son maneras de crear APIs, pero sus enfoques son bastante diferentes.

Vamos a ver cómo se comparan y cuándo podrías preferir uno sobre el otro.





- **Flexibilidad en Peticiones:** GraphQL te permite pedir exactamente los datos que necesitas de manera precisa, en REST las respuestas suelen ser fijas, lo que puede llevar a obtener más información de la necesaria.
- **Instalación y Complejidad Inicial:** GraphQL puede tener una curva de aprendizaje más pronunciada dado su esquema y configuración inicial. REST es más sencillo de empezar si ya estás familiarizado con HTTP y sus métodos.



PABLO DEL ÁLAMO



- **Eficiencia de Redes:** GraphQL, al pedir solo lo necesario, reduce la sobrecarga en el ancho de banda. Perfecto para aplicaciones móviles o de red restringida. En REST las múltiples llamadas para obtener recursos relacionados pueden saturar la red.
- **Necesidad de Versionado:** GraphQL naturalmente maneja cambios en el schema, eliminando la necesidad de versionar. En REST los cambios en la API suelen requerir versionado, lo que puede resultar en más mantenimiento.



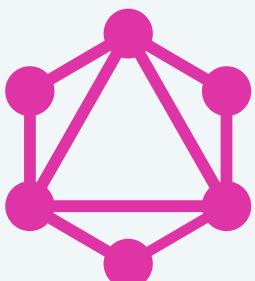
PABLO DEL ÁLAMO



Uso Ideal de GraphQL

Opta por GraphQL cuando:

- 1** Necesitas flexibilidad en las consultas de datos.
.
- 2** Quieres reducir sobrecargas de red.
- 3** Estás diseñando una aplicación con múltiples relaciones de datos complejas.



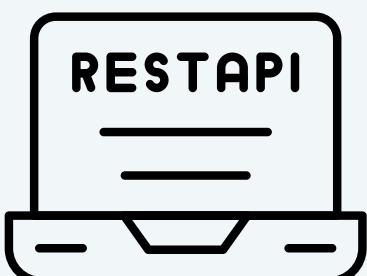
PABLO DEL ÁLAMO



Uso Ideal de REST

Elige REST cuando:

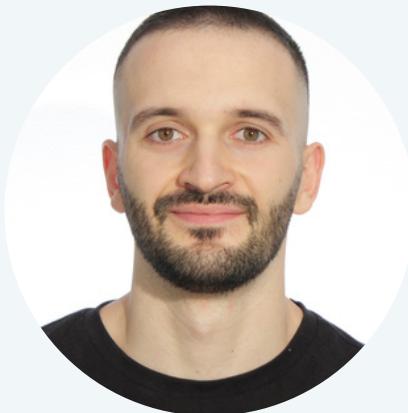
- 1** Requieres una implementación sencilla y rápida.
- 2** La API sigue un patrón estandarizado de recursos CRUD.
- 3** Prefieres un sistema bien conocido para integrar rápidamente.



PABLO DEL ÁLAMO



¿Te ha resultado útil?



- Comparte esta guía con tu equipo o amigos desarrolladores.
- Guárdala para tenerla siempre a mano.
- ¡Dale un like o comenta si tienes preguntas!

