

Query Language

Hacer consultas y esperar una respuesta predecible

SQL `SELECT * FROM cursos;`

Alternativa a REST

Ventajas orientadas a **optimización**

Traslada responsabilidad del **servidor** al **cliente**

Agnóstico de plataforma

Implementado en ~20 lenguajes de servidor

Describe tus datos

Pide lo que necesitas

```
type Curso {
  titulo: String
  profesor: Profesor
  alumnos: [Alumno]
}
```

```
{
  curso(name: "GraphQL") {
   titulo
  }
}
```

Obtiene resultados predecibles



```
"curso": {
    "titulo": "Curso de GraphQL"
    }
}
```



Sus origenes

Creado por Facebook en 2009

Impulsado por el equipo mobile

Motivaciones

Mejorar lo que ofrece REST

Evitar múltiples pedidos al servidor

GraphQL Open Source

Lanzado en **2015**

- Especificación $\mathscr S$
- graphql-js 🔗

Gran adopción de la comunidad

API v4 de Github en GraphQL

REST vs.





REST

VS.



Es sólo una convención

El **servidor** expone recursos

Suele enviar información de más

Múltiples requests por vista o custom endpoints

Documentación ajena al desarrollo

Es un lenguaje tipado y validable

El **cliente** define qué recibe

Se envía sólo lo necesario

1 (un) sólo request por vista

Documentado por definición

Schema

Define qué puede pedir el cliente

Los Types

Un conjunto de Types forman un Schema

Scalars

Int Números enteros

Float Números decimales

String Cadena de texto

Boolean Verdadero o Falso

ID Identificador único

Objects



Enum

```
enum Genero {
  MASCULINO
  FEMENINO
}
```

Interface

```
interface Perfil {
  nombre: String!
  email: String!
}
```

Union

union Busqueda = Amigo | Lugar | Evento | Pagina

Modificadores de tipo

String! no null

[String] Lista

[String]! Lista no null

null **no** es válido, [null, 'texto', null] **es** válido

[String!]! Lista no-nulleable de textos

[null, null, null] **no es** válido

Root Type: Query

endpoints en GraphQL

```
type Query {
  cursos: [Curso]
  profesores: [Profesor]
  curso(id: String!): Curso
  profesor(id: String!, limite: Int): Profesor
}
```

Root Type: Mutation

```
type Mutation {
  profesorAdd(
    nombre: String!,
    genero: Genero
  ): Profesor
}
```

```
type Mutation {
  profesorAdd(
    nombre: String!,
    genero: Genero
  ): Profesor
}
```

Input Types

```
input NuevoProfesor {
  nombre: String!
  genero: Genero
}
```

Clientes GraphQL

¿Por qué necesito uno?

- Queries más fáciles
- Caching
- Normalización de datos
- Optimistic UI
- Paginación





Relay

Creado por Facebook

Cliente más maduro

Optimizado para alta performance

Sólo para React

Estructura opinionada

Demasiado complejo para la mayoría de los casos





Creado por Meteor

API sencilla

Funciona con cualquier framework JS

Estructura flexible

Adopción incremental



iOS y Android

GraphQL Real Time: Subscriptions

Implementación en Schema

```
type Subscription {
  nuevoComentario(cursoId: Int!): Comentario
}
```

Implementación en servidor

Soporte real-time: Websockets

Arquitectura Pub/Sub

- El cliente se suscribe
- El servidor publica cambios

Ejemplo



★ Codenames ★



Conclusiones

¿Qué es GraphQL? Ventajas sobre REST

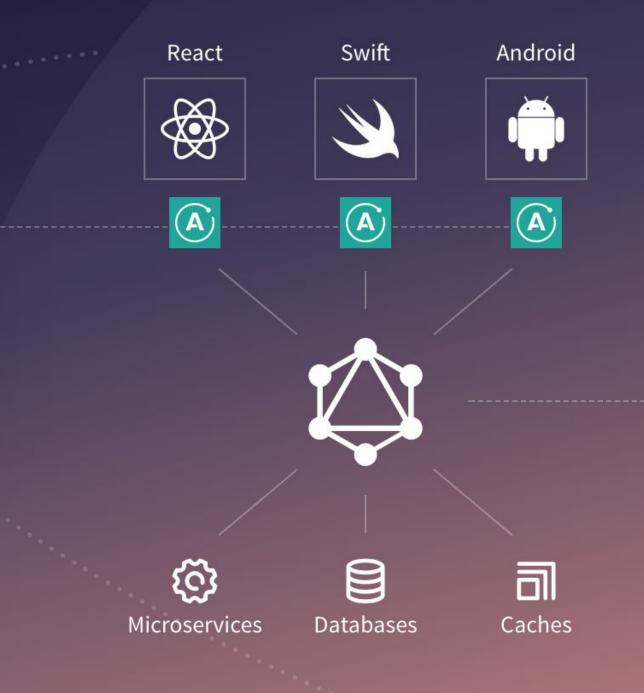
Cómo crear un Esquema GraphQL

Creamos un servidor desde cero

Clientes de GraphQL

¿Cuándo usar GraphQL?





¿Cuándo NO usar GraphQL?



API REST existente con cliente muy simple



¡Gracias por ver el curso!





medium.com/@p4bloch



twitter.com/@p4bloch