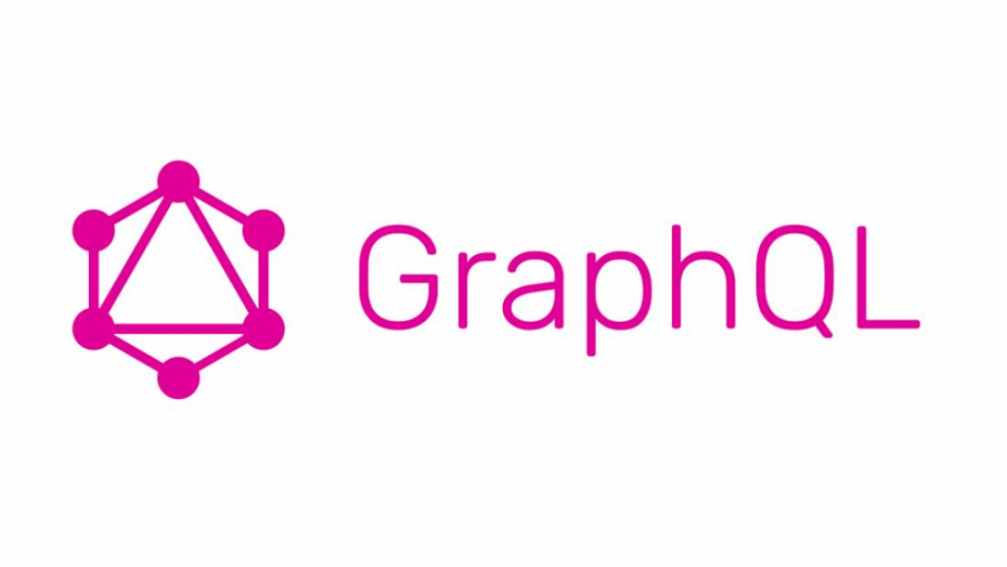
¿Por qué utilizar GraphQL?



Nombre: Rafael Ortiz Barrera

Cargo: Ingeniero en Desarrollo Senior

1.- Situación actual

Hoy en día las empresas usan REST (REpresentational State Transfer) para realizar interacciones entre sistemas independientes, nos permite separar el cliente del servidor pudiendo tener más de un cliente como por ejemplo una aplicación móvil y una aplicación de escritorio, además, al ser Stateless nos permite tener más de un servidor respondiendo las peticiones realizada por los clientes.

REST es una arquitectura que está basada en el protocolo HTTP, define como se debe comunicar el cliente con el servidor en base a una serie de directrices y buenas practicas. Una de sus características es la flexibilidad en el intercambio de datos ya que permite formatos como JSON, XML, texto plano, entre otros.

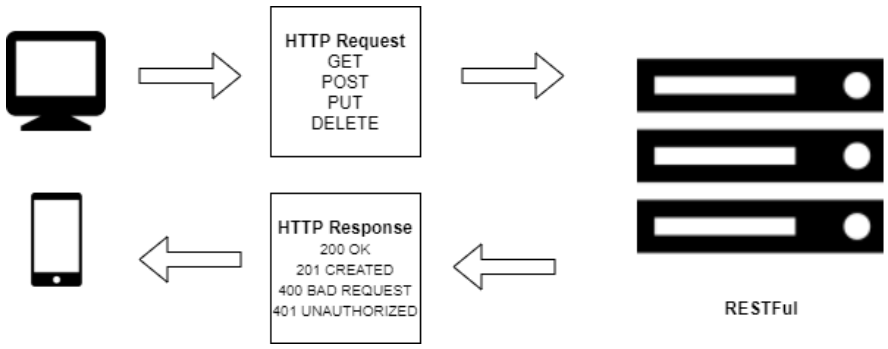


Ilustración 1 Arquitectura REST

Antes de que REST se hiciera popular en las arquitecturas actuales se usaba el protocolo SOAP (Simple Object Access Protocol) que se basa en las especificaciones WS-*, donde los sistemas realizan el intercambio de datos mediante XML.

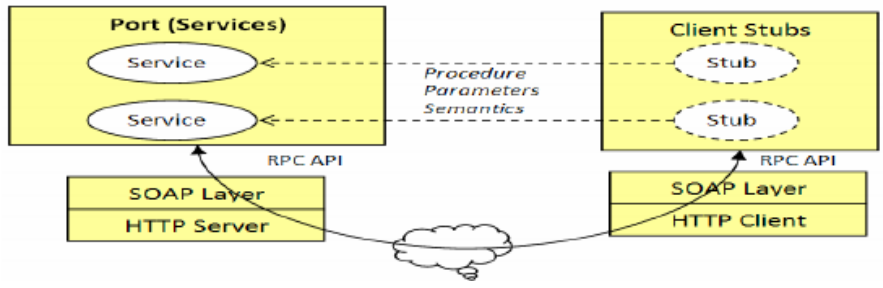


Ilustración 2 Arquitectura Web Service SOAP

REST se empezó a utilizar con mayor frecuencia ya que nos ofrece una comunicación más eficiente y flexible; en cambio, SOAP tiene un alto acoplamiento con el cliente ya que necesita la existencia de un contrato que es conocido como el WSDL (Web Services Description Language), Si el Web Service tiene algún cambio en las entradas y salidas, el cliente tiene que volver a ser regenerado.

REST no llego para reemplazar a SOAP, como en todas las arquitecturas hay que ver la mejor opción a la problemática que se nos presenta, hoy estamos en una era donde los dispositivos móviles se tomaron el mundo dando motivos suficientes para pensar en realizar una arquitectura escalable, por otro lado, hay que tener presente otros factores como la importancia de procesar, analizar y renderizar datos eficazmente permitiéndonos un uso eficiente del ancho de banda.

2.- Objetivo

El objetivo de esta investigación es presentar una forma diferente de comunicación entre cliente-servidor al igual que SOAP y REST. Si bien REST, vino a solucionar problemas que se presentan en el protocolo SOAP, no quiere decir que esta todo resuelto, cada día nacen nuevas necesidades y de soluciones nacen nuevos problemas y así sucesivamente.

El 2012 nació una nueva forma de comunicación entre cliente servidor llamada GraphQL, que es un lenguaje de consulta de datos, fue creado por Facebook los cuales decidieron liberar esta herramienta al mundo el año 2015. Proporciona una alternativa a las arquitecturas basadas en REST con el propósito de aumentar la productividad del desarrollo y minimizar las cantidades de datos transferidos. GraphQL está siendo utilizado por grandes organizaciones como Twitter, GitHub, Intuit, PayPal, the New York Times, entre otras.

Se realizará una prueba de concepto utilizando el lenguaje Java, con el Framework Spring Boot. Esta prueba de concepto está enfocada del lado del servidor. Escogí este Framework porque es uno de los más famosos y utilizados en la actualidad, además tengo conocimientos solidos sobre estos.

3.- ¿Por qué usar GraphQL?

En esta sección se irán revelando las características que hace que GraphQL sea más productivo a la hora de realizar un desarrollo, que personalmente encuentro que este punto es muy importante hoy en día en las empresas, donde muchas veces se cuenta con poco tiempo para desarrollar peticiones exigentes por parte de los usuarios finales.

Un servidor GraphQL expone un único Endpoint para responder las solicitudes del cliente, esta característica facilita demasiado el desarrollar. ¿Cuánto tiempo destinamos en pensar

en la firma de un endpoint? ¿Este nuevo endpoint es un método GET, POST, PUT, DELETE?, ¿Estamos validando las cabeceras como corresponde? ¿Qué código HTTP tengo que retornar para este endpoint? ¿Realmente estamos diseñando RESTFul? Bueno todas estas preguntas con GraphQL quedan obsoletas ya que solo hay un Endpoint el que por default es "/graphql" y se accede únicamente por el método POST, el nombre del endpoint no es del todo rígido, ya que se puede modificar según como lo necesites. En cuanto a los códigos HTTP de respuesta GraphQL siempre retorna 200, sabremos si el servicio nos responde correctamente únicamente por la query de consulta que estamos enviando.

Las consultas del cliente se hacen a través de un lenguaje declarativo y estricto, el beneficio de este lenguaje de consulta es que yo puedo obtener múltiples recursos en una sola consulta a diferencia de una API REST donde tengo que realizar peticiones en cascada para llegar al último recurso que necesito. A continuación, se muestra un ejemplo gráfico (imágenes obtenidas de documentación oficial https://www.howtographql.com/).

Peticiones RESTFul

Ilustración 3 Comunicación cliente-servidor usando REST

En la imagen se ven 3 endpoint, uno para obtener el usuario por su id, otro para obtener los posts de ese usuario por id usuario y el ultimo que obtiene los seguidores de un usuario por su id.

Petición GraphQL

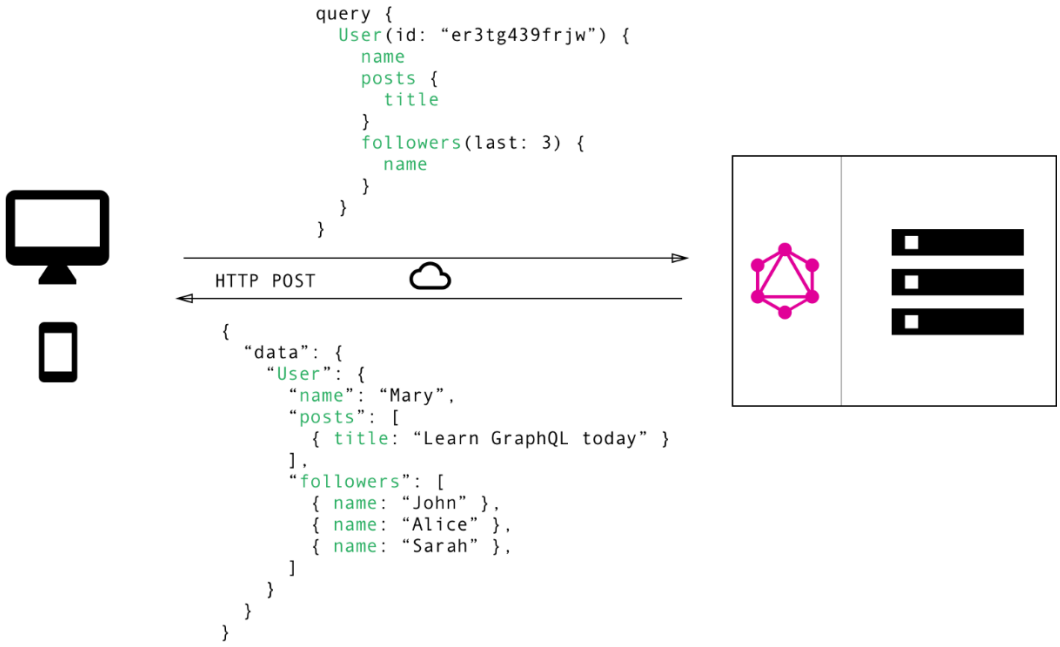


Ilustración 4 Comunicacion cliente-servidor usando GraphQL

Analizando la consulta GraphQL podemos ver que tiene una estructura muy intuitiva. La palabra query, una palabra reservada en el lenguaje GraphQL, nos indica que queremos obtener datos, que en REST sería igual a un método GET. A continuación de la palabra query se indica que información quiero obtener, que es muy parecida a un objeto en los lenguajes de programación, se pasan como parámetros el id de usuario y los filtros de búsqueda como si fuera una función. Finalmente, la respuesta es un JSON con la misma estructura que se solicitó en la query. En mi opinión la forma de solicitar la información favorece la comunicación entre los desarrolladores front-end y back-end.

Los términos más comunes utilizados en GraphQL son los siguientes:

- **DSL**: Schema Definition Language, es el contrato entre cliente y servidor, define como el cliente puede acceder a los datos. Es un archivo. graphql que se encuentra en el lado del servidor, donde se definen las Query y Mutations, que básicamente son funciones que se componen de un nombre, parámetros de entrada y parámetros de salida. Además, se define un esquema como la palabra lo dice, que es muy parecido a definir objetos en Golang.
- **Query**: Como mencione anteriormente es muy similar hacer un GET en REST.
- **Mutations**: Es equivalente a realizar un POST/PUT en REST.

Una pregunta que podría surgir, ¿Cómo puedo saber si el servicio respondió OK en GraphQL? En la imagen anterior la respuesta JSON está envuelta en la llave data, el cliente simplemente debería validar si el valor de data es distinto de nulo. En caso que sea nulo se

retorna un JSON con una key de error que contiene un mensaje o descripción del error, como se muestra en las siguientes imágenes:

QUERY

```
query datosDeUsuarioPorRut($rut: ID!){
    obtenerUsuarioPorRut(rut:$rut){
        rut
        dv
        nombre
        apellido
        email
    }
}
```

Ilustración 5 Query GraphQL

Body Cookies Headers (8) Test Results Status: 200 OK Time: 2.17 s Size: 499 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "data": null,
    "errors": [
        {
            "message": "Cannot return null for non-nullable type: 'Usuario' within parent 'Query' (/obtenerUsuarioPorRut)",
            "path": [
                "obtenerUsuarioPorRut"
            ],
            "locations": null,
            "errorType": "DataFetchingException",
            "extensions": null
        }
    ]
}
```


Ilustración 6 Respuesta de Error de un servidor GraphQL

Otras de las ventajas de GraphQL es que nosotros podemos decidir qué datos queremos que nos devuelva el servicio, a diferencia de REST que si en la firma del método GET retorna 5 parámetros de salida por ej: rut, dv, nombre, apellido y email, este siempre retornara la misma salida, la estructura de datos es fija y es difícil diseñar APIS que se ajusten a la medida de los clientes, sobrecargando muchas veces de información innecesaria. GraphQL nos da la libertad de poder escoger que datos quieres consumir por ejemplo de los 5 parámetros que retorne solo nombre y apellido. Esto nos da una gran ventaja sobre la optimización de renderizar datos, consumo de datos móviles, reutilización de query.

Con la monitorización de APIs podemos saber cómo se están utilizando los datos por parte del cliente si tenemos una query que retorna 100 datos y con la monitorización muestra que de esos 100 datos solo se usan 50, podemos deprecar los otros 50 datos para dar una mantención a nuestras APIs.

Otra razón para usar esta arquitectura, es Open Source, contiene una gran cantidad de documentación, ya se han creado comunidades como GraphQL community en Facebook que cuenta con 7.7 mil miembros, comunidad en stack overflow, se puede chatear con GraphQL developer en IRC a través de freenode, entre otros. Por otra parte, existe GraphQL Fundation para fomentar las contribuciones, documentación, framework, soporte.

En resumen, una tabla comparativa entre REST y GraphQL.

GraphQL	REST
1 endpoint "/graphlq"	Hay que definir recursos: {collection}/{store}/{document}
HTTP method: POST	HTTP method: GET, POST, PUT, DELETE
HTTP Status: 200	HTTP Status: 200, 201, 400, 401, 404, etc
JSON Payload	JSON, XML, texto plano, text/html, etc
Es agnóstico a la capa de transporte.	Protocolo HTTP
El cliente decide qué información obtener.	Respuestas definidas
Con una llamada puedo hacer múltiples operaciones	Una petición, una operación

4.- Seguridad

GraphQL le da libertad al cliente para que pueda consultar lo que quiera y como quiera. Esta libertad puede traer problemas ya que el cliente puede realizar consultas complejas o se pueden realizar consultas con malas intenciones que pueden deshabilitar el servidor GraphQL, es por esta razón que hay que tomar las siguientes precauciones

Timeout: Una de las estrategias más simples de usar es configurar un timeout para defenderse de las consultas grandes. Esta configuración se realiza a nivel de servidor, es la primera barrera de seguridad ya que aun así las consultas llegan al servidor.

Profundidad máxima de consulta: Las consultas pueden caer en ciclos, esto depende de cómo se construye el Schema. Si no se controla la profundidad del esquema se podría crear una consulta demasiado grande lo suficiente para que el servidor GraphQL colapse.

```
query IAmEvil {
  author(id: "abc") {
    posts {
      author {
        posts {
          author {
            posts {
              author {
                # that could go on as deep as the client wants!
              }
            }
          }
        }
      }
    }
  }
}
```

Ilustración 7 query con profunidad

Es importante analizar el AST (Abstract Syntax Tree), con este análisis se puede rechazar consultas que sobrepasen un límite de profundidad por ejemplo se define una profundidad máxima de 3, si llega una consulta con una profundidad 4 será rechazada y esta consulta no se alcanzara a ejecutar en el servidor GraphQL.

```
query {                # Depth: 0
  me {                 # Depth: 1
    friends {          # Depth: 2
      friends {        # Depth: 3
        friends {      # Depth: 4
          friends {    # Depth: 5
            name       # Depth: 6
          }
        }
      }
    }
  }
}
```

Ilustración 8 query con limitación de profundidad

Complejidad de una consulta: Controlar la profundidad de consulta no es suficiente para saber si la consulta es grande o costosa, ya que hay ciertos campos que son más complejos de calcular que otros. Por ejemplo, tenemos la siguiente consulta:

```
query {
  author(id: "abc") { # complexity: 1
    posts {           # complexity: 1
      title           # complexity: 1
    }
  }
}
```

Ilustración 9 definición de complejidad

La consulta que se muestra en la parte superior tiene una complejidad 3, en el cual se define que la obtención de cada uno de esos parámetros tiene un valor de 1. Si nuestro límite de complejidad de consulta fuera 2 esta consulta no se ejecuta en el servidor GraphQL.

En GraphQL se puede establecer una complejidad diferente para cada campo y además puede variar dependiendo de los argumentos, en la siguiente imagen el parámetro post ahora tiene una complejidad de 5 debido a que está recibiendo un argumento “first: 5”.

```
query {
  author(id: "abc") {     # complexity: 1
    posts(first: 5) {     # complexity: 5
      title               # complexity: 1
    }
  }
}
```

Ilustración 10 query pequeña con complejidad mayor

Regulación de tiempo de consultas: Con un conocimiento solido del sistema se puede obtener una estimación de cuánto cuesta en tiempo ejecutar una consulta y como resultado podemos obtener el tiempo máximo de un cliente realizando consultas. Por otro lado, se puede usar algoritmos de limitación para decidir si debemos agregar o quitar tiempo a un cliente.

Bueno y para finalizar, ¿Que dicen los reportes de tendencia de arquitectura de Software?

Ilustración 11 Grafico de tendencias de arquitectura y diseño año 2020 por InfoQ

En un artículo de InfoQ exponiendo las arquitecturas de software realizado el 28 de abril del 2020, posicionan a GraphQL en la sección de Early Majority, donde podemos ver claramente que paso la curva donde nos recomienda empezar su adopción y hacer pruebas en nuestras arquitecturas.

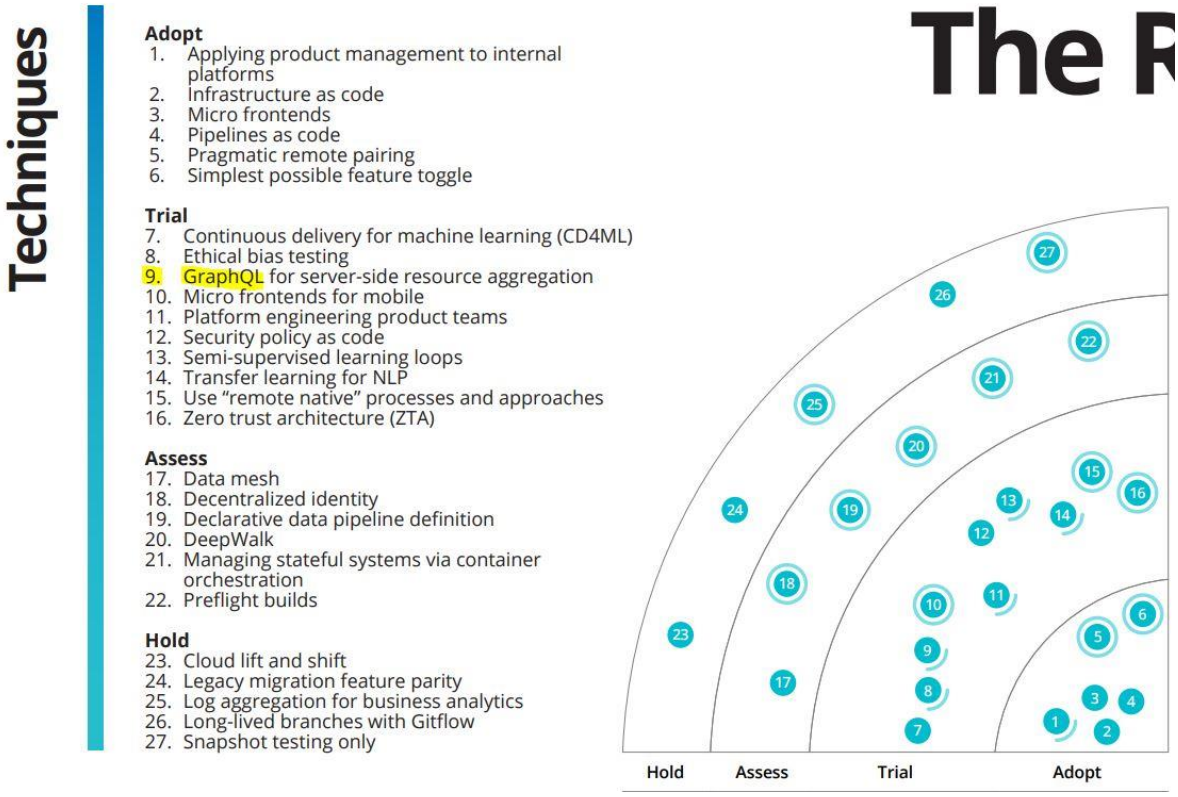


Ilustración 12 Radar de tendencias por Thoughtworks

En el reporte Thoughtworks, obtenido el 09-06-2020. Posicionan a GraphQL en la parte Trail del radar. Que concuerda con el artículo de InfoQ

5.- Arquitecturas

1. GraphQL Server conectado a una base de datos

La arquitectura más común para proyectos que empiezan desde cero. Un servidor web que implementa la especificación GraphQL con una base de datos. El cliente manda la query al servidor, el servidor lee la consulta y obtiene los datos requeridos desde la base de datos. Para finalizar el servidor construye el objeto según el descrito en el DSL y lo devuelve al cliente en formato JSON. Ideal para realizar un mantenedor o aplicación sencilla.

GraphQL es agnóstico a la capa de transporte, lo que significa que puede usar cualquier protocolo de red disponible como TCP, WebSockets, entre otros.

Ilustración 13 Arquitectura Servidor GraphQL con Base de Datos

2. Capa GraphQL que integra múltiples sistemas ya existentes

GraphQL nos permite integrar varios sistemas a través de una capa única que será visible al cliente. Esta arquitectura es conveniente cuando las empresas tienen varios sistemas con distintas comunicaciones cliente-servidor como REST o SOAP, y se requiere unir los datos de diferentes APIS para entregar una única respuesta al cliente.

Otra utilidad de esta arquitectura es cuando un sistema se vuelve poco mantenible por su complejidad detrás de este sistema por lo general son sistemas Legacy. Unos de los problemas que arrastran estos sistemas que es muy difícil innovar sobre ellos. GraphQL nos permite ocultar su complejidad detrás de una API GraphQL de esta forma podemos crear nuevos clientes con tecnologías actuales que se comuniquen con este nuevo Servidor GraphQL.

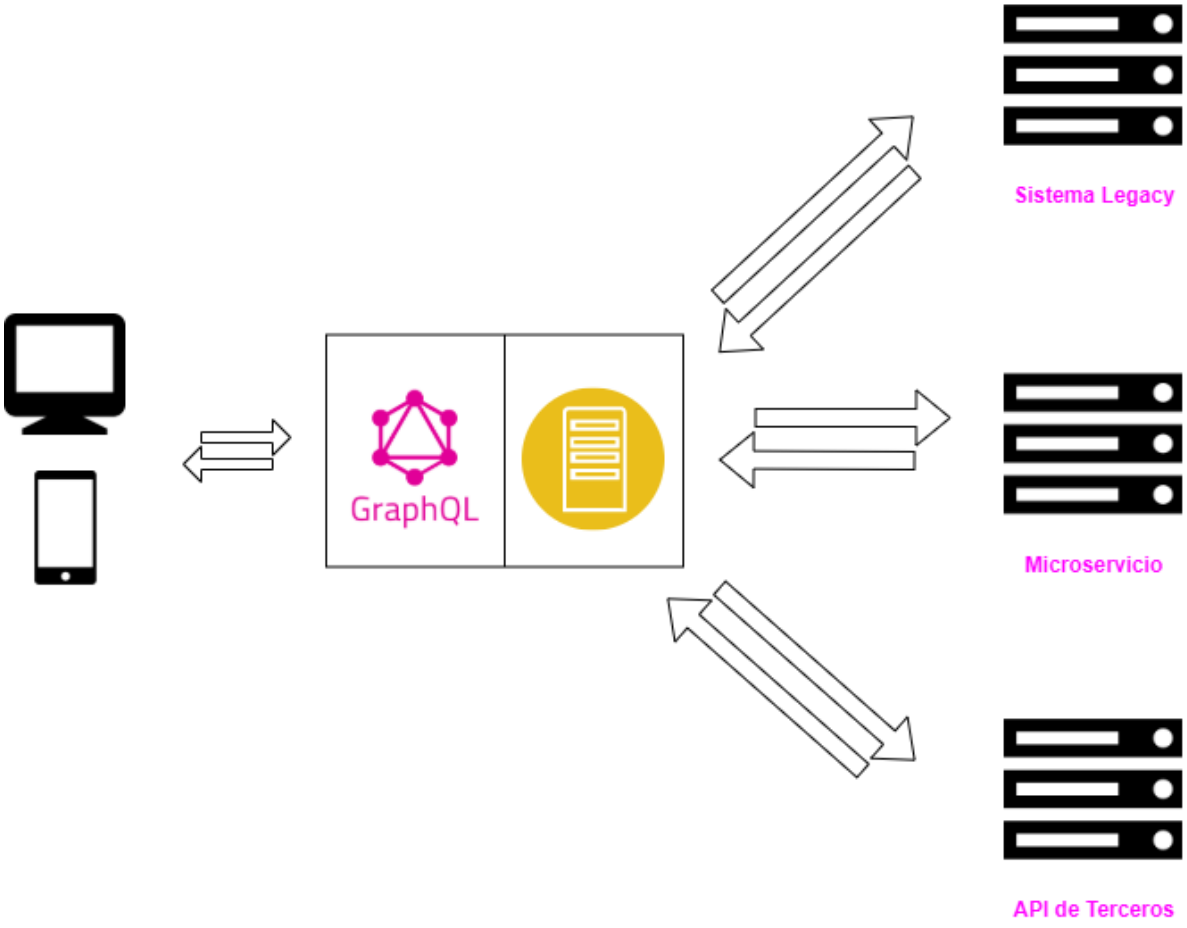


Ilustración 14 Arquitectura de Servidor GraphQL conectado a múltiples sistemas

3. Arquitectura Hibrida

Esta arquitectura es una combinación de las dos anteriores. Donde GraphQL Server tiene su propia base de datos y además se comunica con sistemas externos.

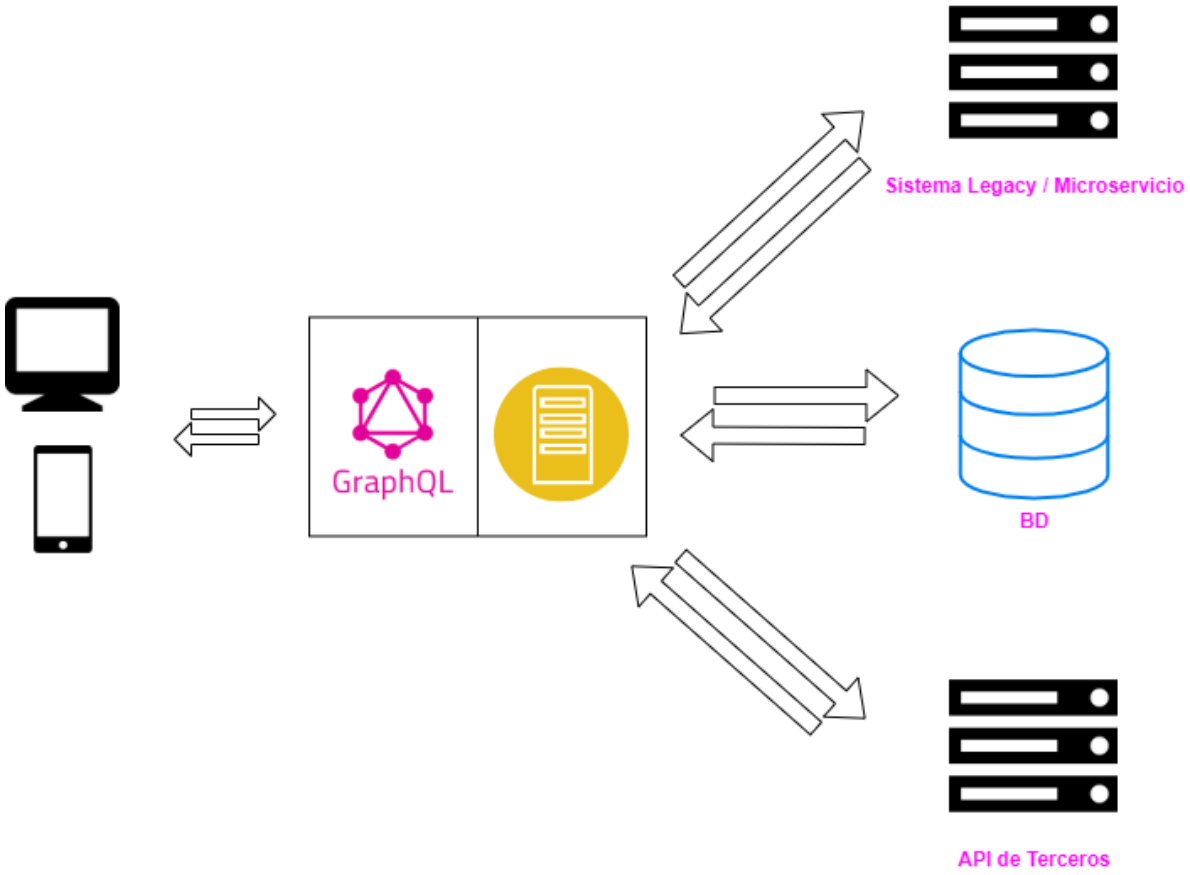


Ilustración 15 Arquitectura Hibrida

6.- Frameworks e integraciones

1. Framework Client Side

GraphQL contiene librerías para integrarse con los siguientes lenguajes: JavaScript, Java/Android, Go, Python, Flutter, entre otros. A continuación, se nombrarán algunos de los tantos framework que existen para integrarse con GraphQL del lado del cliente.

- Relay GraphQL: Es framework de Javascript para crear aplicaciones React para iniciar comunicaciones basadas en las directrices de GraphQL. Se encuentra en la versión 9.1.0, es utilizada por Facebook, Oculus, Gatsby, entre otros.

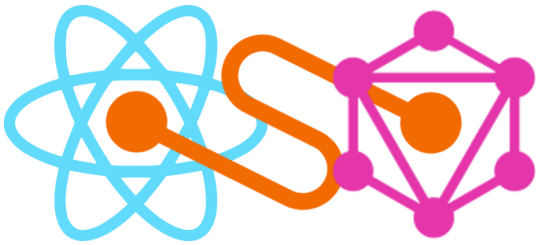


Ilustración 16 Relay GraphQL

- Apollo: Es un framwork de Javascript que funciona tanto del lado del cliente como del servidor. Por el lao del cliente tiene soporte para Angular, Vue, Metor, Ember, Native IOS con Swift, Native Android con Java y además se integra con componentes web con Apollo Elements.

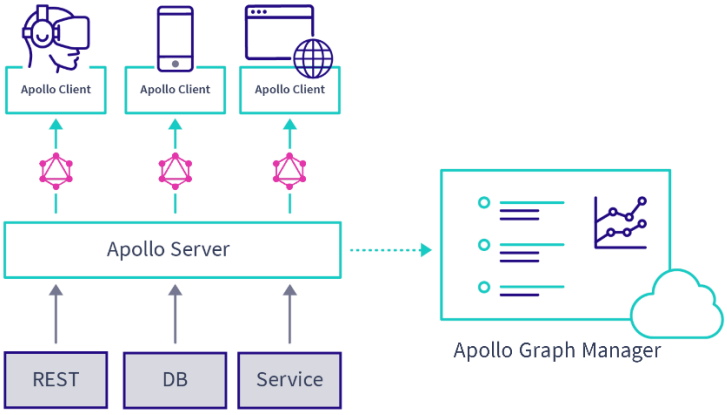


Ilustración 17 Apollo

2. Framework Server Side

GraphQL del lado del servidor también cuenta con una gran cantidad de librerías abarcando muchos lenguajes como C#/.NET, Java, Go, JavaScript, Ruby, Rust, Clojure, Groovy entre otros. A continuación, se mencionan algunos de los frameworks con los que se puede realizar un servidor GraphQL:

- Spring Boot: no podía no existir esta integración, con uno de los más famosos Framework de java. Esta integración funciona bastante bien, la prueba de concepto de esta investigación se hizo con esta librería. Muy fácil de comenzar a realizar un API con GraphQL y Spring boot.

Ilustración 18 Spring Boot GraphQL

- Graphene: es un Framework para el Lenguaje Python, que facilita el desarrollo de API en este famoso lenguaje, que se ha hecho muy popular en los últimos años.

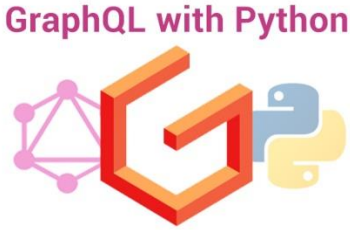


Ilustración 19 Graphene

3. Herramientas
   - GraphiQL IDE: Es un proyecto oficial de GraphQL Fundation. El código de este desarrollo usa la licencia MIT. Este IDE esta hecho en JavaScript, nos permite realizar querys, mutations, scheme, validación de la estructura, entre otros.
   - GraphQL Playground: Es un IDE creado por Prisma y basado en GraphiQL, donde podemos realizar pruebas a nuestras APIs.

7.- Conclusión

Lo que puedo concluir de esta investigación es que GraphQL va a tener mucha presencia en las arquitecturas actuales porque nos permite enfocarnos en la lógica de negocio en vez de realizar definiciones que tienen poco valor para el cliente o usuario final. Perder tiempo en definir cómo se va a llamar un endpoint o que código debería retornar un endpoint, personalmente creo que son poco relevantes.

Me gusto la facilidad con la que pude levantar una API GraphQL, es muy intuitivo ya que su esquema o contrato no es diferente de lo que hacemos los programadores habitualmente, definir objetos con sus parámetros y tipos, crear funciones, trabajar con JSON. Respecto a la curva de aprendizaje de este lenguaje de consultas cliente-servidor, la definirá como una curva empinada donde en poco tiempo se aprende mucho debido a lo que mencione anteriormente.

Cuenta con mucha documentación en internet. La documentación oficial es muy completa donde contiene videos, textos explicativos, diagramas, entre otros. Para realizar este documento me base en la información oficial no necesite buscar información en otros lugares.

Para finaliza quiero hacer mención de sus puntos más fuertes según mi punto de vista que son: tener un único endpoint con un único método http para realizar las consultas, poder seleccionar los datos que quiero obtener desde el servidor mejorando la reutilización de APIs y por otro lado nos permite optimizar los datos móviles.