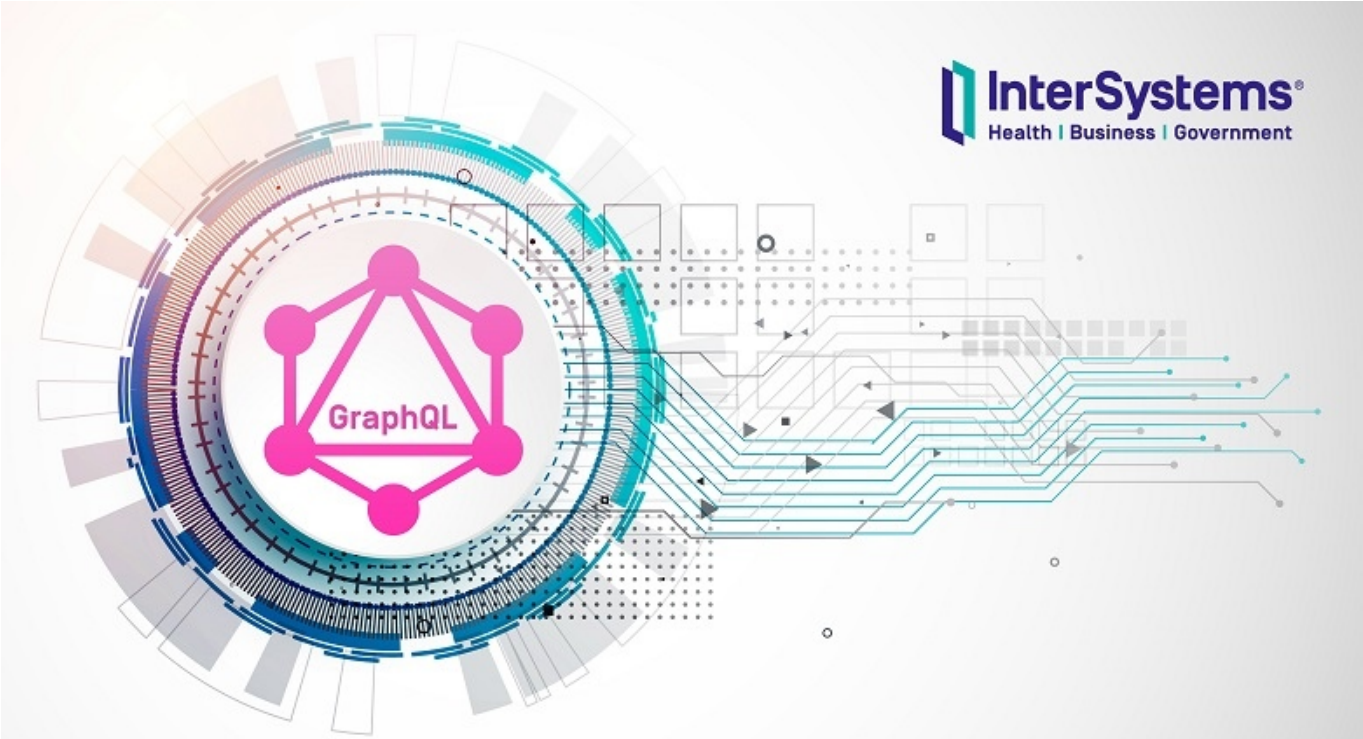
Artículo
Joel Espinoza · 1 jul, 2020 Lectura de 7 min

GraphQL para las plataformas de datos de InterSystems



GraphQL es un lenguaje estándar para declarar estructuras y métodos de acceso a datos, que sirve como una capa de middleware entre el cliente y el servidor. Si nunca has oído hablar de GraphQL, aquí puedes encontrar algunos recursos útiles: aquí, aquí y aquí.

En este artículo, explicaré cómo puedes usar GraphQL en tus proyectos basados en las tecnologías de InterSystems.

Actualmente, las plataformas de InterSystems son compatibles con varios métodos para la creación de aplicaciones cliente/servidor:

- REST
- WebSocket
- SOAP

¿Cuál es la ventaja de usar GraphQL? ¿Qué beneficios aporta comparado con REST, por ejemplo?

GraphQL cuenta con varios tipos de solicitudes:

- **consultas (query)** - son solicitudes del servidor para la obtención de datos, funcionan de forma similar a las peticiones GET que se recomiendan para obtener datos mediante REST.
- **mutaciones** - este tipo de solicitudes es responsable de los cambios en los datos del lado del servidor, funcionan de forma similar a las solicitudes POST (PUT, DELETE) que se utilizan en REST.
  Ambas (consultas y mutaciones) pueden devolver datos – esto es útil si quieres solicitar datos actualizados al servidor inmediatamente después de realizar una mutación.
- **suscripciones** - son similares a las consultas. La única diferencia es que las consultas se ejecutan

mediante una página renderizada en el lado del cliente, mientras las suscripciones se activan por la mutaciones.

Principales características y ventajas de GraphQL

El cliente decide qué datos deben ser devueltos como respuesta

Una de las principales características de GraphQL es que la estructura y el volumen de los datos que son devueltos como respuesta se definen por la aplicación del cliente. La aplicación del cliente especifica qué datos quiere recibir, usando una estructura de tipo gráfico, muy parecida al formato JSON. La estructura de respuesta corresponde a la de la consulta.

Así es como se ve una sencilla consulta en GraphQL:

```
{
  Sample_Company {
    Name
  }
}
```

Una respuesta en el formato JSON:

```
{
  "data": {
    "Sample_Company": [
      {
        "Name": "CompuSoft Associates"
      },
      {
        "Name": "SynerTel Associates"
      },
      {
        "Name": "RoboGlomerate Media Inc."
      },
      {
        "Name": "QuantaTron Partners"
      }
    ]
  }
}
```

Endpoint único

Cuando utilizamos GraphQL para trabajar con datos, siempre nos conectamos a un único **endpoint** (servidor GQL) y obtenemos diferentes datos al modificar la estructura, los campos y los parámetros de nuestras consultas. REST, en cambio, utiliza varios endpoints para distintos objetivos (aún cuando un endpoint soporta multiples verbos, e.g. /usuario puede soportar POST,GET,DEL y PUT).

Vamos a comparar REST con GraphQL mediante un sencillo ejemplo:

Vamos a suponer que tenemos que subir el contenido de un usuario. Si estamos usando REST, necesitamos enviar tres consultas al servidor:

1. Obtener los datos del usuario por medio de su id

2. Utilizar su id para subir sus publicaciones
3. Utilizar su id para obtener una lista de sus seguidores/suscriptores

Este es el mapa REST correspondiente a esas consultas:

```
<Route Url="/user/:id" Method="GET" Call="GetUserByID"/>
<Route Url="/user/:id/posts" Method="GET" Call="GetUserPostsByID"/>
<Route Url="/user/:id/followers" Method="GET" Call="GetUserFollowersByID"/>
```

Para obtener un nuevo conjunto de datos, necesitaremos actualizar este mapa REST con un nuevo endpoint.

GraphQL lo hace con una sola consulta. Para ello, solo hay que especificar lo siguiente en el cuerpo de la solicitud:

```
{
operationName: null,    //la consulta puede tener un nombre  ( query TestName(...){..
.} )
query: "query {
       User(id: "ertg439frjw") {
        name
        posts {
           title
        }
        followers(last: 3) {
           name
        }
    }
}",
variables: null      // inicializacion  de las variables usadas en la query
}
```

Este es el mapa REST correspondiente a esta consulta:

```
<Route Url="/graphql" Method="POST" Call="GraphQL"/>
```

Ten en cuenta que este es el único endpoint en el servidor.

Instalación de GraphQL y GraphiQL

Antes de empezar a utilizar GraphQL, es necesario completar algunos pasos:

1. Descargar la última versión de GitHub e importarla al namespace requerido
2. Ir al portal de administración del sistema y crear una nueva aplicación web basada en los productos de las plataformas de datos de InterSystems (Caché, Ensemble o IRIS):
    - Nombre - **/**
    - Namespace - **por ejemplo, SAMPLES**
    - Clase del controlador - **GraphQL.REST.Main**
3. GraphiQL — es un intérprete para probar las consultas de GraphQL. Descargar la última versión o build tu propia versión desde los recursos disponibles.
4. Crear una nueva aplicación web:
    - Nombre - **/graphiql**

- Namespace - **por ejemplo, SAMPLES**
- Ruta física hacia los archivos CSP - **C:\InterSystems\GraphiQL**

Veamos cuál es el resultado

Ve a este enlace en tu navegador http://localhost:57772/graphiql/index.html (servidor local — server, 57772 — puerto)

Espero que esté claro el uso de los namespaces **Query** y **Response**. Un **Esquema (Schema)** es un documento que se genera para todas las clases almacenadas en un namespace.

El esquema contiene:

- Las clases
- Las propiedades, los argumentos y sus tipos
- Las descripciones de todo lo visto anteriormente, generadas a partir de los comentarios

Vamos a ver con más detalle un esquema para la clase **SampleCompany** :

GraphiQL también es compatible con el completado automático del código, que puede activarse presionando al mismo tiempo las teclas **Ctrl + Space**:

Consultas

Las consultas pueden ser tanto sencillas como complejas para varios conjuntos de datos. A continuación, se muestra un ejemplo de una consulta para los datos de diferentes clases, **SamplePerson** y **SampleCompany** :

Filtros

Por el momento, solo se admite la igualdad estricta:

Paginación

La paginación se sustenta mediante 4 funciones que pueden combinarse para lograr el resultado deseado:

- **after: n** – para todos los registros con un id mayor que n
- **before: n** – para todos los registros con un id menor que n
- **first: n** – para los primeros n registros
- **last: n** – para los últimos n registros

Áreas visibles

En la mayoría de las situaciones, la lógica empresarial de una aplicación indica que únicamente ciertos clientes tendrán acceso a determinados namespace de las clases (autorizaciones basadas en las funciones). De acuerdo con esto, es posible que se necesite limitar la visibilidad de la clase para un cliente:

- Todas las clases que se encuentren en el namespace (**GraphQL.Scope.All**)
- Las clases que se hereden de una superclase (**GraphQL.Scope.Superclass**)
- Las clases que pertenecen a un paquete específico (**GraphQL.Scope.Package**)

Para cambiar el método con el que se restringe la visibilidad, en el entorno, cambia al namespace que necesites y después abre la clase **GraphQL.Settings**. Tiene un parámetro **SCOPECLASS** con el valor predeterminado de

GraphQL.Scope.All — esta es la clase que contiene la descripción para la visibilidad de dicha clase en el namespace de la interfaz:

Para cambiar las restricciones en la visibilidad de la clase, es necesario definir alguno de los valores que se proporcionaron anteriormente: **GraphQL.Scope.Package** o **GraphQL.Scope.Superclass**.

Si seleccionaste **GraphQL.Scope.Package**, también tendrás que ir a esa clase y cambiar el valor del parámetro **Package** con el nombre del parámetro que se requiera – por ejemplo, **Sample**. Esto hará que todas las clases que se almacenaron desde este paquete estén completamente disponibles:

Si seleccionaste **GraphQL.Scope.Superclass**, simplemente serán las clases que se heredaron desde esta clase, una vez más en las clases requeridas:

Actualmente es compatible con

Consultas:

- Básicas
- Objetos incrustados
    - Únicamente los que tengan una relación de muchos a uno
- Lista de tipos simples
- Lista de objetos

Actualmente en desarrollo

Consultas:

- Objetos incrustados
    - Compatibles con todo tipo de relaciones
- Filtros
    - Compatibles con desigualdades

Planes

- Mutaciones
- Alias
- Directivas
- Fragmentos

→ Enlace al repositorio del proyecto
→ Enlace al servidor de demostración

#API #IRIS Analytics Architect #InterSystems IRIS

URL de fuente:https://es.community.intersystems.com/post/graphql-para-las-plataformas-de-datos-de-intersystems