**APIs as data exchange (1)**

Si bien es cierto que la información se almacena siempre en los recursos de la compañía, en algún punto será necesario compartir los datos.

Anteriormente, la información se compartía físicamente a través de cintas magnéticas, u otros dispositivos, incluso moviendo la computadora físicamente. Cuando las redes comenzaron a estandarizarse, la información intercambiada digitalmente a través de las líneas telefónicas comenzó a usar protocolos generales como SMTP, FTP, etc.

Estos protocolos no estandarizaban la estructura de los datos, por lo que las empresas recurrían a metodologías propias. Por necesidad, se empezaron a trabajar convenciones como XML, JSON, etc.

La ventaja de seguir este mecanismo de intercambio es que una API puede ser utilizada por cualquier tipo de dispositivo, IoT, celular, etc. La comunicación no se limita a la interacción entre humanos y aplicaciones

Cada mecanismo tiene sus tradeoff, y como arquitectos, desarrolladores, o managers, deben de reconocer los beneficios o desventajas de utilizar esta tecnología.

**GraphQL**

Es una tecnología open-source, inicialmente proyecto de Meta/Facebook, ahora gobernado por la GraphQL Foundation, en las que participan varias empresas, y organizaciones como Apollo, AWS, Butterfly Network, Dgraph Labs, Facebook, Gatsby, GraphZen, Hasura, IBM, Intuit, Neo4j, Novvum, Pipefy, Salsify, Solo.io and Thicit, incluyendo la Linux Foundation.

La fundación se encarga se establecer las políticas y administrar el presupuesto de la comunidad. Para unirse, es necesario aplicar a través de un formulario.

**¿Por qué Graph?**

Más adelante hablaremos de los grafos

Tipificación de datos y ejecución de queries para obtener exactamente lo que se necesita.

Reducción de versionado. Si bien graph provee especificaciones para deprecar, Graph provee herramientas para la evolución constante del esquema. A diferencia de otros mecanismos, cualquier cambio puede ser un breaking change, que compliquen el entendimiento y la mantenibilidad del API. GraphQL sólo retorna lo que es explícitamente solicitado.

**REST vs GraphQL**

Ambos permiten crear, modificar, actualizar, y eliminar datos; son stateless, por lo que no se guarda el historial de las peticiones; utilizan modelo cliente-servidor; diseño basado en recursos, neutralidad de lenguaje y bases de datos.

REST

* Intercambio mediante verbos HTTP (GET para lectura, POST para entrada, PUT para actualizar)
* Mucho código para una consulta
* Sin esquema del servidor (Una API REST funciona sin un esquema, pero su documentación ayuda a su consumo)
* Mantenimiento por versionado (<https://example.com/api/v1/person/12341>)
* Manejo de errores explícito. El sistema no puede identificar errores automáticamente, se necesitan validaciones

GraphQL

* Lenguaje de consulta con especificaciones de cómo un cliente debe solicitar datos a un servidor remoto. (Query para obtención de datos, mutaciones para modificar, y suscripciones para actualizaciones)
* Obtención de lo que se pide
* Con esquema del servidor (definición requiere tipificación de objetos y relaciones explícitas, definición de accesos o modificaciones)
* Compatibilidad entre versiones (campos obsoletos devuelven errores o advertencias)
* Manejo de errores implícitos: al ser un tipado inflexible, y el establecimiento del detalle del esquema, el sistema identifica automáticamente los errores en las solicitudes y provee mensajes de error útiles.

Pensando en grafos

Abstracción útil para el modelado de redes, estructuras de datos, toma de decisiones, etc.

Los círculos se llaman vértices, y las líneas aristas. Cada vértice representa un objeto, y la arista la relación entre estos.

**Pensando en grafos 2**

Al ser representación natural de modelos mentales y descripciones verbales de un proceso. El dominio define el esquema, con sus tipos, nodos, y cómo se conectan o relacionan entre ellos.

El esquema de GraphQL es un lenguaje expresivo. Una buena definición describe el negocio en inglés.

Debe existir consenso de negocio con esta definición. HABLAR DE POR QUÉ EN ESPAÑOL.

Si la definición de los recursos es correcta, utilizar REST y GraphQL no debería ser un problema.

**Pensando en grafos 3**

Si se quiere utilizar graph con un sistema legacy, construir un esquema que describa CÓMO los clientes usan los datos, en vez de hacer un espejo.

Los esquemas de Graph no reflejan qué relación, sino cómo se relacionan. Se puede mejorar la implementación, sin hacer cambios fuertes para los viejos clientes.

Expansión gradual del esquema para tener validaciones y feedback para poder llegar a la mejor solución.

**Adiós a las convenciones, ¡bienvenida especificación de GraphQL!**

Fields: campos para obtener en un query. Lo que se pide es lo que se retornaa.

Arguments: valores para la ejecución de consultas. A diferencia de REST que únicamente se puede pasar un set de argumentos a través de la URL, en Graph es posible hacer consultas anidadas para cada objeto, ya sean valores escalares, enumerables, custom, etc.

Fragments: para la agregación en consultas complejas, donde los valores pueden repetirse por lo menos una vez, se pueden hacer fragmentos

Mutaciones: en REST, cualquier petición puede causar efectos en el servidor, pero por convención se sugiere que GET no haga uso de esta posibilidad. GraphQL técnicamente es similar, pero se necesitan establecer convenciones para que estas causen escrituras en escenarios explícitos a través de la mutación.

Como en los queries, la ejecución de la mutación resulta en la obtención del resultado escrito.