Introducción a los *frameworks*

PROGRAMACIÓN DE APLICACIONES UTILIZANDO *FRAMEWORKS*. UNIDAD 0

2º DESARROLLO DE APLICACIONES WEB. ARCIPRESTE DE HITA. 2025/2026

AARÓN MONTALVO

0. Introducción a los *frameworks*

0.1. Desarrollo web

0.2. *Frameworks* web

0.3. APIs web

0.4. GraphQL

0.1. Desarrollo web

Proceso de desarrollo de un sitio web para Internet o una intranet.

Abarca desde el desarrollo de una simple página estática de texto plano hasta aplicaciones web complejas

- Comercio electrónico y servicios de redes sociales.
- Ingeniería, diseño, contenido, comunicación, *scripting*, seguridad.
- Se pueden utilizar CMS para que los cambios de contenido sean más sencillos.

Generalmente el desarrollo web son los aspectos no relacionados con el diseño de sitios web: lenguaje de marcas y codificación.

0.1. Desarrollo web

El desarrollo web ha evolucionado desde aplicaciones estáticas a dinámicas, y estas últimas, desde aplicaciones de múltiples páginas a SPA *(Single-Page Application).*

Existe la necesidad de separar la información almacenada y manejada por el servidor de la mostrada en el interfaz gráfico: el servidor debe devolver los datos de manera independiente a cómo se van a mostrar

- **Patrón MVC: Modelo-Vista-Controlador**

0.1. Desarrollo web

- *Frontend*
- *Backend*
- Bases de datos
  - SQL
  - CRUD
- Servicio HTTP
  - cliente-servidor
  - petición/respuesta

0.1. Desarrollo web. *Frontend vs backend*

Frontend

- Parte visible de la aplicación web. El camarero del restaurante.
- Interactúa directamente con el usuario.
- Tecnologías principales
  - HTML: estructura del contenido.
  - CSS: estilo y diseño.
  - JavaScript: interactividad.
- Ejemplos: botones, formularios, menús, animaciones, muestra de datos.

0.1. Desarrollo web. *Frontend vs backend*

Backend

- Parte lógica y oculta de la aplicación.
- Procesa peticiones y gestiona datos.
- Tecnologías principales
  - Lenguajes: PHP, Python, Java, JavaScript (Node.js).
  - Bases de datos: MySQL/MariaDB, PostgreSQL, MongoDB.
  - Servidores: Apache, Nginx.
- Funciones principales: Procesar formularios, consultar la base de datos, gestionar usuarios y sesiones.

0.1. Desarrollo web. Línea histórica

Años 90

- Inicio de la web
  - HTML (1993)
  - Aplicaciones con CGI (1997)
- Primeros lenguajes de generación de aplicaciones web dinámicas
  - PHP/FI (1995). PHP 3 (1998)
  - ASP (1996)
  - JSP / Servlets (1999)

0.1. Desarrollo web. Línea histórica

Años 2000s

- Consolidación de los lenguajes de *backend*
  - PHP 4/5 (2000-2008/2004-2018)
  - ASP.NET (2002)
- Aplicaciones dinámicas: AJAX (2006, XMLHttpRequest 1999)
- Gestores de contenido
  - WordPress (2003)
  - Joomla! (2005)

0.1. Desarrollo web. Línea histórica

Años 2010s:

- Auge del JavaScript en cliente
  - jQuery (1.0 en 2006, 2.0 en 2013)
  - JSON (2013)
- *Frameworks* MVC
  - Laravel (2011, PHP)
  - Django (v1.0 en 2008, Python)
  - Ruby on Rails (2004, Rails 3.0 en 2013, Ruby)

0.1. Desarrollo web. Línea histórica

Años 2020s:

- Popularización de *Frameworks frontend* (JavaScript)
  - React (2013), Angular (2016), Vue.js (2014)
- Popularización de *Frameworks backend*
  - MVC: Django, Laravel
  - API: Spring Boot (2014, Java), Express.js (Node.js, JavaScript)

0.2. *Frameworks* web

Problemas del desarrollo sin frameworks

- Código desordenado y repetitivo.
- Dificultad para mantener proyectos medianos o grandes.
- Riesgos de seguridad
  - Inyecciones SQL
  - Validaciones mal hechas
- Dificultad de escalar y añadir nuevas funcionalidades.

0.2. *Frameworks* web

Un *framework* es un conjunto de herramientas, librerías y buenas prácticas para desarrollar aplicaciones de forma rápida y ordenada.

- Se basa en el principio de “no reinventar la rueda”.

Características:

- Organización clara de carpetas y ficheros.
- Uso de patrones de diseño (MVC).
- Funcionalidades integradas: rutas, validación de formularios, autenticación, seguridad.
- Extensibilidad (módulos/paquetes/*plugins*).

0.2. *Frameworks* web. Tipos

Frontend:

- React
- Angular
- Vue.js

Backend

- Django
- Laravel
- Spring Boot
- Express

Fullstack / Isomorphic JavaScript

- Next.js (React)
- Meteor (Node.js)

0.2. *Frameworks* web. Ventajas e incoventientes

- ✅ Desarrollo más rápido y organizado.
- ✅ Buenas prácticas y seguridad por defecto.
- ✅ Documentación y comunidades grandes.
- ❌ Curva de aprendizaje inicial.
- ❌ Dependencia del *framework* y sus actualizaciones.

0.2. *Frameworks* web. PHP puro vs. Laravel

PHP	vs.	Laravel
Modular	**Estructura**	MVC
Varios niveles	**Flexibilidad**	Reglas estrictas
Sin mecanismo	**Caché**	Múltiples configuraciones
Sin dependencias externas	**Dependencias**	Con dependencias externas
Necesidad de reglas durante desarrollo	**Seguridad**	Autenticación y autorización por defecto
No tiene por defecto	**Comunicación datos**	*Token* de seguridad
Definición manual	**Manejo excepciones**	Protocolos de manejo

Actividad 0.1

Recordad un proyecto del curso anterior

- ¿Qué problemas hubo al organizar y mantener código?.
- ¿Qué ventajas tendría habría tenido trabajar con *frameworks*?

¿Por qué creéis que los *frameworks* son el estándar en las empresas?

Actividad 0.2

https://www.reddit.com/r/PHP/comments/1iamhhj/someone_still_using_raw_php_over_frameworks_like

¿Qué opináis sobre lo que dice ese usuario de Reddit sobre rendimiento y productividad?

¿Qué ventajas observas en Laravel?

¿En qué casos tendría sentido usar solo PHP puro?

0.3. APIs web

API (Application Programming Interface): Interfaz para comunicar aplicaciones.

En web, interfaz de programación (o protocolo) para la interacción entre el cliente (*frontend*) y el servidor (*backend*).

- La API define cómo se intercambian datos cliente y servidor.

https://jsonplaceholder.typicode.com/todos/1

https://pokeapi.co/api/v2/pokemon/umbreon

https://catfact.ninja/fact

https://dog.ceo/api/breeds/image/random

0.3. APIs web. Ejemplo

Más herramientas -> Herramientas para desarrolladores -> Consola

Tecla F12

```
fetch("https://api.agify.io?name=aaron")
   .then(r => r.json())
   .then(data => console.log(data));
```

0.3. APIs web. Tipos

Los servicios SOAP (Simple Object Access Protocol) fueron los primeros en aparecer, pero su creación y mantenimiento es muy tediosa

- Utilizan ficheros WSDL (basados en XML) para la descripción de los servicios de forma estática o dinámica.

Las APIs REST (REpresentational State Transfer), utiliza las peticiones HTTP para dar una semántica a las transacciones

- GET/POST/PATCH/PUT/DELETE
- El mayor inconveniente que necesidad presentar un conjunto de puntos diferenciados de entrada (*endpoints*).
  - Tiene diferentes direcciones URL y formas de las mismas

0.3. APIs web. Tipos

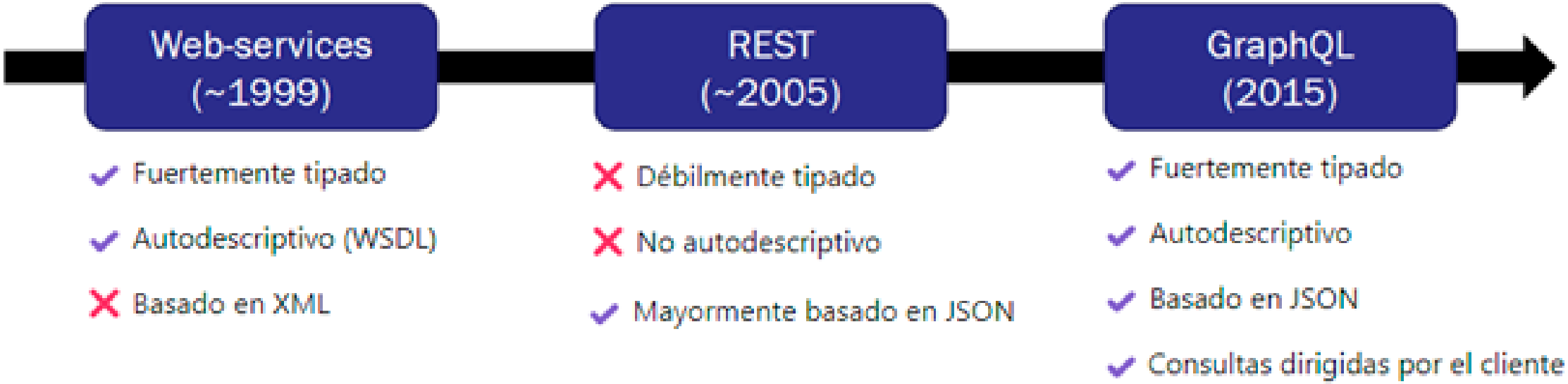
SOAP

- más antiguo
- usa XML
- hoy casi en desuso en aplicaciones web comunes (excepto administración)

REST

- más ligero
- normalmente usa JSON
- se basa en peticiones HTTP
- es el estándar de facto.

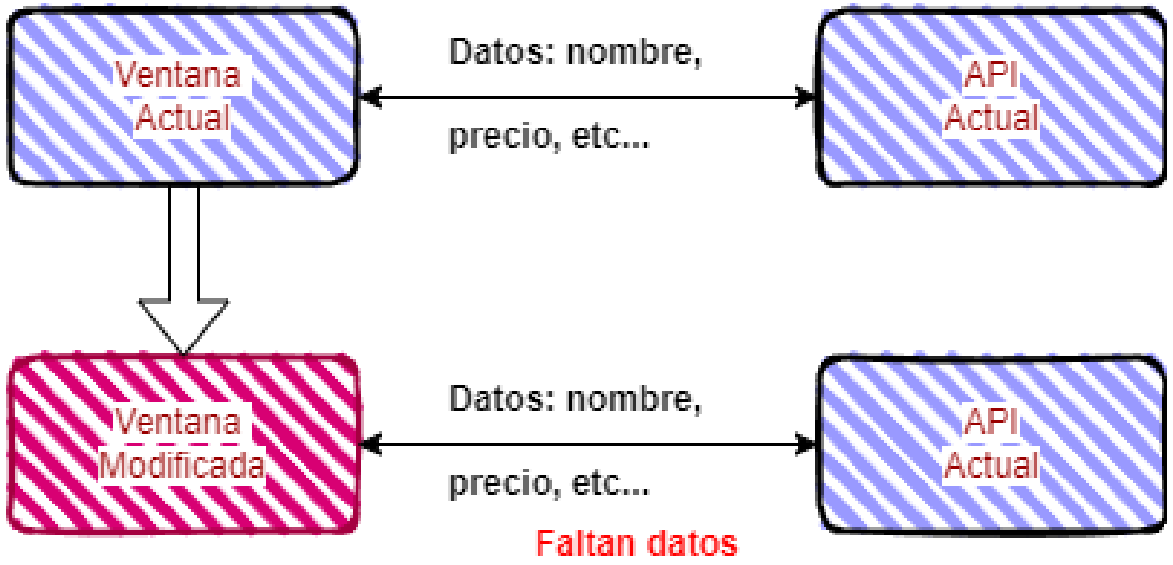
0.3. APIs web.
Tipos



0.3. APIs web. Problema REST

Partimos de una empresa que ya tiene establecida una aplicación web con una API REST para dar servicio a los clientes (azul).

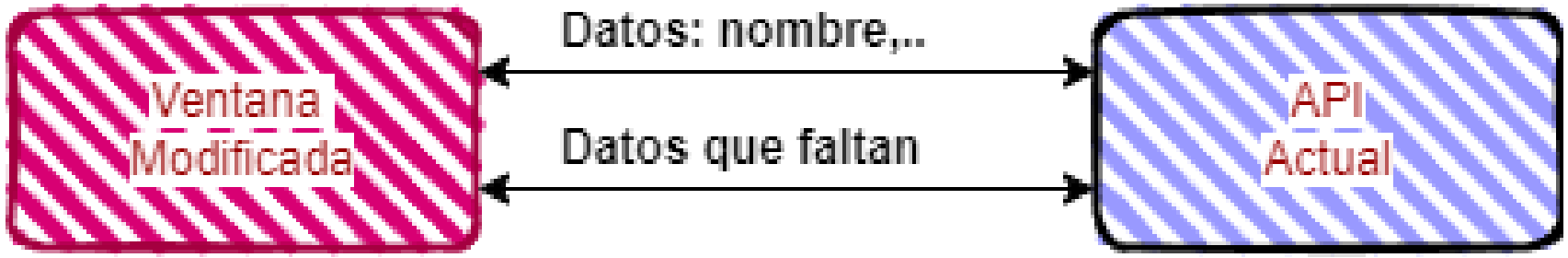
Se desea ahora que aparezca en una ventana información de la compañía que actualmente no existe (rojo).



0.3. APIs web. Problema REST

Solución 1: Hacer 2 peticiones

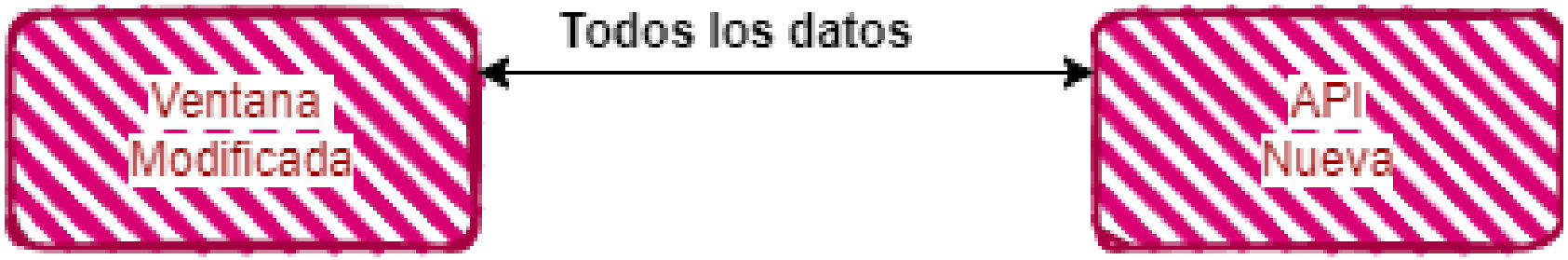
- Aumenta el número de peticiones y por tanto la latencia de la aplicación
- Puede no ser factible



0.3. APIs web. Problema REST

Solución 2: Hacer una API nueva

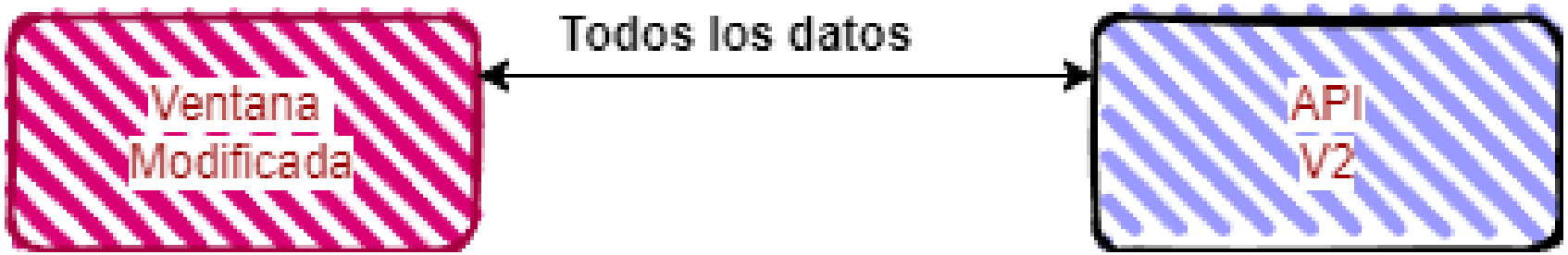
- API inconsistente con la anterior versión
  - Genera incertidumbre a los clientes
- Se mandan más datos de los realmente necesarios



0.3. APIs web. Problema REST

Solución 3: Añadir un *flag* de petición

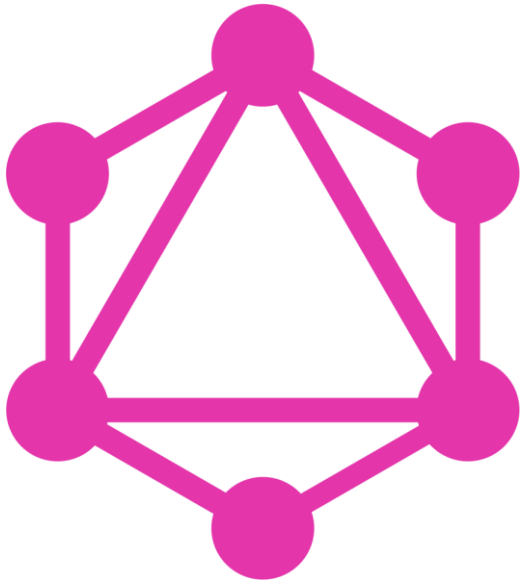
- API inconsistente con la anterior versión
  - Genera incertidumbre a los clientes
- Se mandan más datos de los realmente necesarios



0.3. APIs web. GraphQL

GraphQL aparece para dar solución a varios problemas:

- presenta un único punto de entrada
- permite mecanismos de descubrimiento para los clientes
- es autodocumentada
- utiliza un formato JSON



0.4. GraphQL

GraphQL es un lenguaje de consulta para su API y un entorno de ejecución ***del lado del servidor*** para ejecutar consultas

- utiliza un sistema de tipos estrictamente definido para los datos, de código abierto, implementado en una variedad de lenguajes de programación.
- no está vinculado a ninguna base de datos o motor de almacenamiento específico.

0.4. GraphQL

GraphQL es una nueva manera de comunicarse entre *frontend* y *backend.*

Permite desarrollos flexibles y mantenibles a lo largo del tiempo

Contiene una sintaxis fácil y clara

- Recopila la misma información que los correspondientes esquemas UML.

Facilita la migración entre versiones del API sin modificación de puntos de acceso.

Se adecúa muy bien a las metodologías actuales ágiles, pero también a las fases de análisis y diseño tradicionales.

0.4. GraphQL

GraphQL no es apto para todos los proyectos.

Para desarrollos pequeños, con pocos datos o con datos muy estáticos a lo largo del tiempo, una API REST es más adecuada

- Por su rápida implantación y su sencillez de uso.

0.4. GraphQL. Empresas

1. Meta

- Desarrolló GraphQL internamente para resolver problemas de sobrecarga de datos y para optimizar las aplicaciones móviles. Posteriormente, lo liberó como proyecto de código abierto en 2015.

2. GitHub

- Adoptó GraphQL para su API v4, permitiendo a los desarrolladores especificar exactamente los datos que necesitan, lo que mejora la eficiencia y flexibilidad en las integraciones.

3. Shopify.

- Ha marcado su API REST Admin como obsoleta y ha promovido el uso de su API GraphQL Admin para nuevas aplicaciones.

0.4. GraphQL. Empresas

4. Netflix

- Migró sus aplicaciones móviles a GraphQL sin tiempo de inactividad, utilizando microservicios GraphQL para su *backend*.

5. Airbnb

- Adoptó GraphQL para mejorar la velocidad de desarrollo y la eficiencia en la gestión de datos en sus aplicaciones.

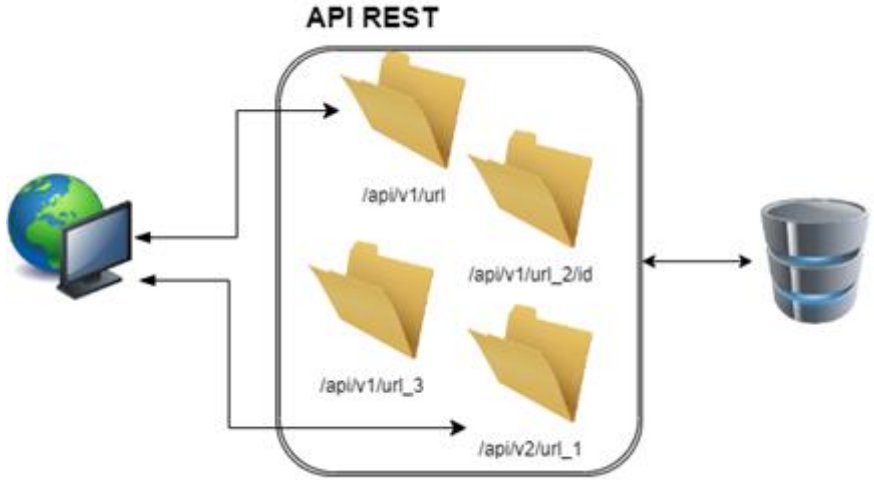
6. Twitter/X

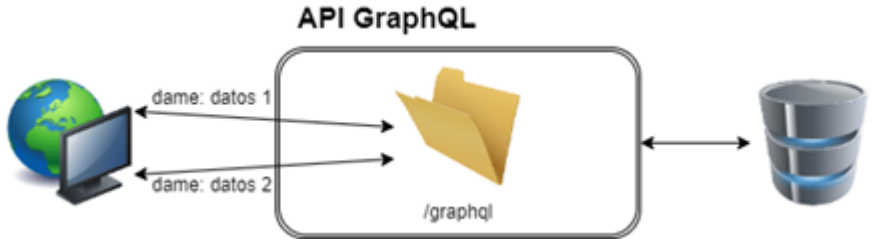
- Reconstruyó su API pública utilizando GraphQL para mejorar la flexibilidad y eficiencia en las consultas de datos.

0.4. GraphQL. Vs. REST

GraphQL y REST son dos enfoques distintos para diseñar una API de intercambio de datos a través de Internet.

- REST es una arquitectura que facilita la comunicación de aplicaciones.
  - GraphQL funciona con varios *endpoints* mediante HTTP.
- GraphQL un lenguaje de consulta de API con su especificación y herramientas.
  - GraphQL funciona en un único punto de conexión mediante HTTP.





0.4. GraphQL. Vs. REST

Elementos de una solicitud REST:

- Verbos HTTP: determinan la acción.
  - GET/POST/PATCH/PUT/DELETE
  - GET sirve para obtener datos de solo lectura de un recurso
  - POST sirve para agregar una nueva entrada de recurso
  - PUT sirve para actualizar un recurso
- URL: identifica cada recurso sobre el que se realizará la acción del verbo HTTP.
- Parámetros y valores: para analizar lo que se desea consultar, crear o modificar en un recurso existente en el servidor.

0.4. GraphQL. Vs. REST

Elementos de una solicitud GraphQL:

- Consulta: para obtener datos de solo lectura.
- Mutación: para modificar datos.
- Suscripción: para recibir actualizaciones de datos de *streaming* o basadas en eventos.
- Un formato de datos: describe cómo desea el cliente que el servidor devuelva los datos
  - Incluidos los objetos y campos que coinciden con el esquema del servidor.
  - También se puede introducir datos nuevos.
  - Internamente, GraphQL envía cada solicitud de cliente como una solicitud HTTP POST.

0.4. GraphQL. Vs. REST

	REST	GraphQL
¿Qué es?	Conjunto de reglas que define el intercambio de datos estructurados entre un cliente y un servidor.	Lenguaje de consulta, estilo de arquitectura y conjunto de herramientas para crear y manipular las API.
Casos de uso	Orígenes de datos simples donde los recursos están bien definidos.	Orígenes de datos grandes, complejos e interrelacionados.
Acceso a los datos	Varios puntos de conexión en forma de URL para definir los recursos.	Único punto de conexión de URL.
Datos devueltos	Estructura fija definida por el servidor.	Estructura flexible definida por el cliente.
Estructura y definición de datos	Tipado débil: el cliente debe decidir cómo interpretar los datos formateados cuando se devuelvan.	Tipado fuerte: el cliente recibe los datos en formatos predeterminados y mutuamente comprendidos.
Comprobación de errores	El cliente debe comprobar si los datos devueltos son válidos.	Las solicitudes no válidas suelen ser rechazadas por la estructura del esquema, autogenerando mensajes de error.

0.4. GraphQL. Vs. REST

	REST	GraphQL
Método de uso	GET/POST según *endpoint*	Normalmente POST a un único *endpoint*
URL	Cada recurso tiene su URL (/pokemon/ditto)	Un único endpoint (/graphql)
Datos enviados	Parámetros en URL o body	Query en JSON: {"query": "..."}
Selección de datos	Predefinidos por el *endpoint*	El cliente decide qué campos quiere

0.4. GraphQL. Vs. REST vs. SQL

	REST	GraphQL	SQL
Petición	Se usa un *endpoint* para devolver todos los datos de Ditto.	Se usa un único *endpoint* (/graphql/v1beta2). El cliente decide qué campos quiere.	La API GraphQL de PokeAPI está montada en Hasura, que traduce la *query* a SQL y consulta PostgreSQL.
Datos de petición	https://pokeapi.co/api/v2/pokemon/ditto	{ pokemon(limit: 1, where: {name: {_eq: "ditto"}}) { id name height weight pokemontypes { type { name } } } }	SELECT id, name, height, weight FROM pokemon WHERE name = 'ditto' LIMIT 1;
Respuesta	JSON gigante con id, nombre, tipos, stats, sprites, movimientos, etc	JSON solo con lo pedido: id, nombre, altura, peso y tipos (normal)	Fila de la tabla pokemon con Ditto.

Actividad 0.3 (1/3)

1. Realiza las siguientes consultas sobre la pokeapi
   a) En REST

https://pokeapi.co/api/v2/pokemon/umbreon

   b) En GraphQL
      - En la aplicación, disponible en la siguiente URL, ejecuta la consulta mostrada en la página siguiente:

https://graphql.pokeapi.co/v1beta2/console

Actividad 0.3 (2/3)

```
query {
  pokemon (limit: 1, where: {name: {_eq: "umbreon"}}) {
    name
    height
    weight
    pokemontypes {
      type {
        name
      }
    }
  }
}
```

Actividad 0.3 (3/3)

2. ¿Qué crees que devuelve la siguiente consulta GraphQL? ¿Cómo se obtendría la misma información con *endpoints* de la API REST?

```
query tallest {
  pokemon: pokemon(order_by: {height: desc}, limit: 3, where:
{is_default: {_eq: true}}) {
    name
    height
  }
}
```