Engineer

REST

🏛 estilo arquitectónico de la web, basado en recursos

Qué es

REST, abreviatura de Representational State Transfer, es el estilo arquitectónico que Roy Fielding expuso en el capítulo 5 de su tesis en 2000. No es un protocolo ni un estándar. Es un conjunto de restricciones sobre cómo pueden interactuar clientes, servidores y los intermediarios que hay entre ellos. Fielding explica qué aporta cada restricción y, en la mayoría de los casos, qué cuesta. RESTful es el adjetivo: en sentido estricto, describe un sistema que respeta todas las restricciones. En el uso cotidiano, ambos términos se amplían mucho más, hasta casi cualquier API HTTP que intercambie JSON, respete o no las restricciones siguientes.

La idea central es el recurso. En palabras de Fielding, «cualquier información que se pueda nombrar puede ser un recurso»: un documento, una imagen, un conjunto de otros recursos, una persona o incluso el tiempo que hace en una ciudad. Lo que cruza la red nunca es el recurso en sí, sino una representación: una secuencia de bytes y metadatos que describen esos bytes, como un documento HTML o una imagen JPEG. Un cliente actúa sobre un recurso enviando o recibiendo una representación de su estado actual o deseado.

Fielding construye el estilo a partir de un conjunto vacío de restricciones y las añade de una en una. Cliente-servidor separa la interfaz de usuario del almacenamiento de datos, de modo que ambos puedan evolucionar de forma independiente. La ausencia de estado significa que cada petición lleva todo lo necesario para entenderla y no depende de ningún contexto almacenado en el servidor; por tanto, el estado de sesión reside por completo en el cliente. La caché exige que cada respuesta se marque, explícita o implícitamente, como almacenable en caché o no, para que el cliente pueda reutilizar una respuesta almacenable en caché en otra petición posterior y equivalente. La interfaz uniforme, descrita más abajo, da a todos los componentes la misma interfaz general. Un sistema por capas significa que cada componente solo ve la capa con

la que se comunica directamente. Así caben proxies, pasarelas y cortafuegos intermedios sin cambiar las interfaces de ambos extremos. El código bajo demanda permite a un servidor ampliar un cliente enviándole scripts para ejecutar. Fielding lo convierte en la única restricción opcional porque reduce la visibilidad.

Fielding llama a la interfaz uniforme la característica central que distingue REST de otros estilos basados en red, y la divide en cuatro restricciones: los recursos se identifican, en la web mediante URL; se manipulan a través de representaciones; los mensajes se describen a sí mismos, de modo que su significado resulta visible para cualquier intermediario; y el hipermedia es el motor del estado de la aplicación. Esto último significa que el cliente elige su siguiente transición a partir de las representaciones recibidas, igual que una persona sigue enlaces en un navegador.

Utilice el concepto en una decisión

Aplique «¿Qué es REST?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La parte que el usuario no ve: dónde se guardan los datos, qué reglas se aplican y cómo pueden comunicarse otros sistemas con ella.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Diseño y operación de 6 flujos ETL/ELT — 6 flujos ETL en tres motores de bases de datos

Diseño y operación de 6 flujos ETL/ELT Póngase en contacto →

Por qué importa

Cada restricción aporta algo concreto. La ausencia de estado permite a un sistema de monitorización entender una petición sin buscar fuera de ella, facilita la recuperación ante fallos parciales y permite al servidor liberar recursos en cuanto responde. Una caché puede eliminar por completo algunas interacciones. Las capas permiten a los intermediarios repartir la carga entre redes y máquinas, y a una caché en el perímetro de una organización atender a todas las personas de su interior. Como la interfaz uniforme desacopla las implementaciones de los servicios que ofrecen, un cliente y un servidor pueden cambiar sin que el otro tenga que hacerlo al mismo tiempo.

Fielding es igual de claro sobre los costes. La ausencia de estado repite en cada petición datos que un contexto compartido en el servidor habría ahorrado. Una caché puede entregar datos obsoletos. Cada capa añade sobrecarga y latencia. La interfaz uniforme reduce la eficiencia porque la información viaja con un formato

estandarizado, no uno ajustado a las necesidades de una aplicación concreta. Fielding la diseñó para transferencias de datos hipermedia de grano grueso, el caso habitual en la web, y dice claramente que no es óptima para otros tipos de interacción. Ese es el terreno de las alternativas. Un servidor GraphQL recibe todas las peticiones en un único punto de entrada, identifica sus entidades por algo distinto de las URL y deja que el cliente elija exactamente qué campos recibe; gRPC, por su parte, llama a métodos declarados en una definición de servicio. Cada uno renuncia a parte de la interfaz uniforme para ajustarse a otro tipo de tráfico.

En una API HTTP, buena parte de la interfaz uniforme ya viene resuelta. HTTP define métodos con significados acordados, códigos de estado y campos de cabecera para la caché y las peticiones condicionales. Como ese significado es compartido, un proxy o una caché intermedios pueden actuar sobre un mensaje sin saber nada de esa API concreta. Una API que utiliza esos elementos según su definición obtiene esa maquinaria sin trabajo adicional. Si entrega a sus clientes plantillas de URL para rellenar en lugar de enlaces que seguir, abandona la restricción de hipermedia. Sus clientes quedan ligados a la estructura de URL del servidor: cuando cambia, dejan de funcionar. Ese es el acoplamiento que la interfaz uniforme pretende eliminar.

Cómo se aplica en la práctica

El intercambio siguiente crea un pedido y recibe como respuesta la dirección del recurso nuevo en `Location` y una etiqueta de entidad en `ETag`. La propia RFC 9110 utiliza como ejemplo un `ETag` en una respuesta 201: la etiqueta puede servir de validador en peticiones condicionales posteriores para impedir que dos clientes sobrescriban en silencio los cambios del otro.

```
POST /orders HTTP/1.1
Host: api.example.com
Content-Type: application/json

{"item": "notebook", "quantity": 2}

HTTP/1.1 201 Created
Location: /orders/7421
ETag: "v1"
Content-Type: application/json

{"id": 7421, "item": "notebook", "quantity": 2, "status": "open"}
```

Cada método tiene un significado en el que cualquier intermediario puede apoyarse. La RFC 9110 define GET, HEAD, OPTIONS y TRACE como seguros: el cliente no solicita ni espera ningún cambio de estado en el servidor. PUT y DELETE, junto con los métodos seguros, son idempotentes, de modo que varias peticiones idénticas tienen el mismo efecto previsto que una sola. POST no tiene ninguna de las dos propiedades, y la RFC 5789 dice lo mismo de PATCH. Esas dos propiedades permiten a un rastreador seguir enlaces sin causar daño, a una caché adelantar descargas y a un cliente repetir un PUT cuya conexión se haya cortado antes de recibir la respuesta.

Una API REST se puede manejar con un simple cliente HTTP genérico, y este sitio muestra tanto lo que ahorra como lo que cuesta. Cuando se aprovisionó un segundo servidor a partir de código, los cortafuegos de ambos proveedores cloud se configuraron directamente mediante sus API REST con una tarea HTTP genérica, en vez de una colección del proveedor. Eso elimina una dependencia y la divergencia de sus versiones, a cambio de escribir a mano la estructura de cada petición. La misma página señala un segundo coste: la tarea HTTP genérica no declara compatibilidad con el modo de comprobación, por lo que una ejecución en modo de simulación del play que crea la máquina omite todas sus tareas.

Dónde surgen los problemas

Un GET cambia algo. Una dirección como `/orders/7421/cancel` que cancela un pedido al consultarla rompe la promesa de que GET es seguro. La RFC 9110 explica por qué existe esa promesa: para que los rastreadores y las cachés que adelantan descargas puedan trabajar sin miedo a causar daños. El primer rastreador o caché con descarga anticipada que encuentre el enlace cancela el pedido. Ponga la acción en el método, por ejemplo mediante un POST al pedido o un PATCH de su estado.

Todas las respuestas son 200 OK. Algunas API devuelven 200 para todo y ponen el resultado real, haya error o no, en el cuerpo. Los mensajes dejan de describirse por sí mismos: las cachés, los proxies, la monitorización y las bibliotecas cliente leen el código de estado, y todos ven éxito. La RFC 9457 define un cuerpo estándar para errores, `application/problem+json`, con miembros como `type`, `title`, `status` y `detail`. Su miembro `status` es solo orientativo: el servidor debe enviar el mismo código de estado en la propia respuesta HTTP para que el software HTTP genérico, que desconoce ese formato, se comporte correctamente.

Un POST repetido crea dos pedidos. Si la conexión se corta después de que el servidor haya procesado un POST pero antes de que el cliente haya leído la respuesta, este no puede saber si el pedido existe. La RFC 9110 dice que un cliente no debería repetir automáticamente una petición no idempotente, salvo que sepa que en realidad sí lo es o pueda determinar que el primer intento no llegó a aplicarse, y que un proxy no debe hacerlo en ningún caso. También señala que algunos clientes toman la vía más arriesgada y repiten un POST de todas formas cuando la conexión se cierra antes de recibir parte alguna de la respuesta, especialmente en una conexión persistente inactiva. Una solución es permitir al cliente elegir el identificador y enviar el pedido mediante PUT a esa dirección, ya que PUT es idempotente: la primera petición crea el pedido con 201 y una repetición solo lo sustituye por el mismo estado.

Dos clientes sobrescriben los cambios del otro. Ambos leen el pedido, cada uno cambia un campo distinto y ambos envían de nuevo el contenido completo mediante PUT: la segunda escritura deshace en silencio la primera. La RFC 9110 lo llama problema de actualización perdida. La solución es devolver en `If-Match` la etiqueta de entidad que cada cliente recibió al leer el pedido. El servidor solo aplica el cambio si el recurso sigue en el estado que ese cliente vio por última vez. Si no es así, no debe aplicarlo y puede indicarlo con 412 Precondition Failed. Después, el cliente lee otra vez el pedido, vuelve a aplicar su cambio y lo intenta de nuevo.

```
PUT /orders/7421 HTTP/1.1
Host: api.example.com
If-Match: "v1"
Content-Type: application/json

{"item": "notebook", "quantity": 3, "status": "open"}

HTTP/1.1 412 Precondition Failed
```

El servidor recuerda la conversación. Un servidor que conserva estado por cliente entre peticiones, como la página actual de una lista o un formulario a medio completar guardado en memoria, ha abandonado la restricción de ausencia de estado. Una petición ya no se entiende por sí sola, el balanceador debe enviar cada cliente a la misma máquina y el estado se pierde al reiniciar. Se pierden precisamente las razones de Fielding para no conservarlo: visibilidad, recuperación más fácil ante fallos parciales y un servidor que puede liberar

recursos al responder. Incluya el estado en la petición, por ejemplo como un token de página en la URL, o convierta lo que está a medio completar en un recurso propio, como un borrador de pedido, que cualquier servidor pueda leer.

Lecturas adicionales

- Fielding, capítulo 5: Representational State Transfer, REST — La definición: cada restricción, las propiedades que induce y lo que sacrifica a cambio.
- RFC 9110: semántica HTTP — Métodos, seguridad e idempotencia, códigos de estado y peticiones condicionales con `ETag` e `If-Match`.
- RFC 5789: método PATCH para HTTP — Modificación parcial de un recurso y por qué PATCH no es seguro ni idempotente.
- RFC 9457: detalles de problemas para API HTTP — Un cuerpo JSON estándar para respuestas de error que conserva el significado del código de estado.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es REST?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/rest/