Engineer

JSON

Formato de texto habitual para intercambiar datos

Qué es

JSON es una notación de texto plano para datos anidados, escrita con llaves, corchetes, comillas dobles y comas para que un programa de casi cualquier lenguaje pueda leer exactamente lo que escribió otro. Las siglas significan JavaScript Object Notation, porque la sintaxis procede de cómo JavaScript escribe valores literales; sin embargo, un documento JSON no pertenece a ningún lenguaje ni necesita JavaScript para leerse. Los archivos suelen terminar en `.json` y, en HTTP, el contenido se identifica con el tipo de medio `application/json`.

Todo el formato cabe en una página. Su vocabulario consta de cuatro tipos primitivos —cadenas, números, los literales `true` y `false`, y `null` — y dos estructurados: un objeto, que es un conjunto sin orden de pares de nombre y valor, y un array, que es una lista ordenada. Cualquier elemento puede anidarse dentro de otro. Las fechas, los datos binarios, los comentarios y las referencias entre partes de un documento quedan en manos de convenciones externas al formato.

Douglas Crockford lo registró por primera vez en el IETF mediante RFC 4627, en julio de 2006. La definición actual es RFC 8259, de diciembre de 2017, que coincide con el estándar ECMA-404 de Ecma en la gramática y añade las reglas importantes para el intercambio entre dos sistemas. El texto que sale de un ecosistema cerrado debe ser UTF-8 y el emisor no debe anteponer una marca de orden de bytes.

XML es más antiguo y expresivo, con atributos, espacios de nombres y una larga tradición de lenguajes de esquemas. YAML se revisó en la versión 1.2 para ser un superconjunto estricto de JSON, añadiendo los comentarios y la puntuación más

ligera que se buscan en archivos editados a mano. Las codificaciones binarias, como Protocol Buffers y CBOR, sacrifican legibilidad para reducir el tamaño y aumentar la velocidad.

Utilice el concepto en una decisión

Aplique «¿Qué es JSON?» 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

Elegir JSON es, en realidad, elegir un contrato entre dos sistemas: la estructura de lo que se transmite y, a veces, la de lo que se almacena.

En la transmisión es la opción predeterminada porque todo el ecosistema ya lo admite. Un navegador lo analiza sin bibliotecas, todos los lenguajes habituales tienen un analizador maduro y un ingeniero que lea un registro puede ver qué contenía una petición. Los costes están en otra parte. Cada registro repite los nombres de sus campos, los números viajan como texto decimal que debe volver a analizarse y el formato no establece qué campos son obligatorios ni qué tipo contiene cada uno. Una codificación binaria que parte de un esquema, como Protocol Buffers, ocupa menos y se analiza más deprisa, y un compilador comprueba su contrato; pero una persona necesita una herramienta para leer un solo mensaje. Para una API pública o dirigida al navegador, JSON es una elección razonable. Entre servicios internos donde dominan el volumen o la latencia, la decisión puede inclinarse al otro lado.

Un esquema suele aportar el contrato que falta. JSON Schema declara qué propiedades debe tener un objeto, de qué tipo son y si se permiten propiedades adicionales; OpenAPI se basa en él para describir API HTTP completas. El esquema es un documento aparte y solo sigue siendo exacto mientras algo lo obligue a coincidir con el código.

En el almacenamiento, la pregunta es si una estructura pertenece a una columna JSON o a columnas convencionales. PostgreSQL ofrece dos tipos. `json` conserva una copia exacta del texto de entrada y vuelve a analizarlo cada vez que

una función lo utiliza, mientras que `jsonb` almacena una forma binaria descompuesta que resulta algo más lenta de escribir, mucho más rápida de consultar y que se puede indexar; la documentación recomienda `jsonb` para la mayoría de las aplicaciones. Una columna JSON encaja con datos cuya estructura varía entre filas o pertenece al cliente, como las preferencias guardadas de un usuario. Un campo que la base de datos deba usar para unir tablas, someter a restricciones o sumar merece una columna propia.

Cómo se aplica en la práctica

En una plataforma, las funciones PostgreSQL ensamblan por sí mismas cada respuesta completa como JSON, y la API Go reenvía ese JSON desde la base de datos al navegador sin modificarlo. El manejador lo recibe como `json.RawMessage` y lo transmite sin decodificarlo ni volverlo a codificar; una pequeña función auxiliar `QueryJSON` convirtió ese comportamiento en el predeterminado. Desaparecieron los DTO, los mapeadores y las estructuras de respuesta de una API por capas. Cuando cambia la salida de una función, la nueva estructura llega al cliente sin modificar el código del manejador.

El JSON almacenado plantea otro problema, porque sobrevive al código que lo escribió. Una plataforma conserva los ajustes de notificación y otras preferencias como los valores guardados del usuario superpuestos a un conjunto de valores predeterminados, originalmente combinados con un solo nivel de profundidad. Una clave añadida después a esos valores faltaba en todas las filas guardadas antes de que existiera, por lo que el código cliente leía un campo no definido y fallaba; les ocurría a los usuarios con más antigüedad. La solución fue una combinación profunda que trata los valores predeterminados como un catálogo: todas las claves del catálogo quedan presentes, incluidas las anidadas, independientemente de lo que contengan los datos guardados. Una migración la aplicó de inmediato a las filas existentes y el frontend eliminó sus comprobaciones defensivas.

JSON también es, sencillamente, la forma en que llegan los datos. Para datos históricos de electricidad entregados como JSONL, con un evento JSON por línea, había que convertir los eventos brutos en consumo horario por zona, y se construyeron tres variantes, en Python, JQ y SQL. Python fue la más rápida: 34 ms para los datos de un día y 180 ms para treinta días.

Dónde surgen los problemas

La gramática es lo bastante pequeña para que los analizadores coincidan en qué texto es válido. Los problemas proceden de todo lo que la gramática deja abierto.

Un identificador grande vuelve con una cifra distinta. La gramática no fija un límite a los números, pero la mayoría de los analizadores, incluido el de JavaScript, los leen como números de coma flotante de 64 bits, que solo representan enteros con exactitud hasta $2^{53} – 1$. RFC 8259 identifica el intervalo de $–(2^{53} – 1)$ a $2^{53} – 1$ como aquel en que coinciden las implementaciones. Más allá, los identificadores cambian sin avisar:

```
JSON.parse('{"id": 9007199254740993}').id;
// 9007199254740992
```

Envíe los identificadores de 64 bits como cadenas y el dinero como una cadena o como una cantidad entera de unidades mínimas, nunca como coma flotante.

Dos servicios leen el mismo objeto de forma distinta. RFC 8259 indica que los nombres de un objeto deberían ser únicos y deja abierto lo demás: ante un nombre repetido, muchos analizadores conservan el último valor, otros dan error y algunos devuelven todos los pares. Si una pasarela comprueba el primer `role` y el backend actúa sobre el segundo, la comprobación no protegía nada. Rechace los nombres duplicados a la entrada del sistema. I-JSON, definido en RFC 7493, es un perfil de JSON que los prohíbe, y un contrato puede citarlo por su nombre.

Un archivo editado a mano deja de poder analizarse. Los comentarios y las comas finales quedan fuera de la gramática, por lo que la primera persona que anota un archivo de configuración o deja una coma después del último elemento lo rompe. Algunas herramientas admiten un dialecto más permisivo, lo que hace válido el mismo archivo para un lector e inválido para el siguiente. Reserve JSON para los datos escritos por máquinas y dé a los archivos mantenidos a mano un formato pensado para editarse, como YAML o TOML.

El texto llega ilegible o un analizador rechaza lo que otro acepta. RFC 8259 exige UTF-8 entre sistemas y prohíbe una marca de orden de bytes al principio. Un productor que escribe Latin-1 o un editor que antepone una marca genera un texto cuya interpretación depende del analizador. La gramática también permite

un escape como `"\uDEAD"`, la mitad de un par sustituto sin su pareja, que ninguna cadena Unicode válida puede contener y que las bibliotecas tratan de formas distintas. Codifique en UTF-8 en el origen y rechace las secuencias inválidas en la frontera.

Una petición pequeña ocupa todo el analizador. Unos pocos kilobytes de corchetes anidados pueden llevar a un analizador recursivo a miles de niveles de profundidad, y un cuerpo excesivamente grande se analiza entero antes de que ningún esquema lo vea. RFC 8259 permite limitar tanto el tamaño del texto como la profundidad del anidamiento; una API que acepta JSON de desconocidos debería limitar ambos: acote el tamaño de la petición antes de leerla, la profundidad al analizarla y solo después compruebe su estructura.

Lecturas adicionales

- RFC 8259: formato de intercambio de datos JSON — la definición actual del IETF, con sus notas sobre números, nombres duplicados y codificación.
- ECMA-404: sintaxis de intercambio de datos JSON — el estándar de Ecma para la misma gramática, que RFC 8259 cita como normativo.
- RFC 7493: formato de mensajes I-JSON — un perfil más estricto que exige UTF-8 y prohíbe nombres duplicados.
- Especificación de JSON Schema — el vocabulario para declarar qué propiedades debe tener un documento.
- PostgreSQL: tipos JSON — `json` frente a `jsonb` y qué conserva cada uno del texto original.
- Go: json.RawMessage — el tipo que permite a un programa Go aplazar la decodificación o transmitir JSON sin modificarlo.

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 JSON?» 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/json/