Engineer

JSON Schema

Lenguaje para describir la estructura válida de datos JSON

Qué es

JSON Schema es un lenguaje para describir qué forma debe tener un dato JSON, escrito a su vez en JSON. Una descripción en ese lenguaje, llamada esquema, establece las reglas: este valor es un objeto, debe tener estas propiedades, una de ellas es un número entero de al menos uno y otra una cadena de como máximo 200 caracteres. Un programa llamado validador recibe un esquema y una instancia, los datos que se comprueban, e indica si la instancia cumple las reglas y, si no, dónde y por qué. El proyecto escribe su nombre con S mayúscula, JSON Schema, mientras que un documento escrito en el lenguaje es simplemente un esquema, en minúscula.

La especificación la mantiene el proyecto JSON Schema en json-schema.org. La versión actual es 2020-12 y la anterior 2019-09; las versiones más antiguas eran borradores numerados, como draft-04 y draft-07, y las herramientas y los artículos antiguos siguen usando esos nombres. El texto tiene dos partes. El documento principal define cómo se procesa un esquema, cómo se refiere uno a otro y el tipo de medio `application/schema+json`; un segundo documento, sobre validación, define las palabras clave que comprueban los valores. Ambos documentos actuales de 2020-12 están fechados el 16 de junio de 2022.

Cada palabra clave añade una restricción. `type` indica uno de seis tipos básicos: string (cadena), number (número), boolean (booleano), array (lista de elementos), object (objeto) o null (nulo), o bien `integer`, que solo acepta números cuya parte fraccionaria sea cero. Los números admiten `minimum`, `maximum` y `multipleOf`; las cadenas, `minLength`, `maxLength` y `pattern`, una expresión regular; los arrays, `minItems`, `maxItems` y `uniqueItems`; y los objetos, `properties`, `required` y `additionalProperties`. Palabras clave como `allOf`, `anyOf`, `oneOf` y `not` combinan esquemas, y `$ref` apunta a un esquema definido en otro lugar, a

menudo bajo `$defs` en el mismo documento, para reutilizar en muchos sitios una estructura escrita una sola vez. Todo lo que un esquema no menciona está permitido. El esquema vacío, `{}`, acepta cualquier instancia.

Un esquema también puede indicar en qué versión del lenguaje está escrito. La palabra clave `$schema` contiene la dirección de un metaesquema, un esquema que describe los esquemas válidos de esa versión, y la página de la especificación señala que el metaesquema se valida a sí mismo. Para 2020-12, esa dirección es `https://json-schema.org/draft/2020-12/schema`.

Utilice el concepto en una decisión

Aplique «¿Qué es JSON Schema?» 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

JSON no dice nada sobre el significado. Todo documento bien formado se puede analizar, independientemente de que el programa que lo lea pueda hacer algo con él. Sin una descripción escrita, cada programa que lee un documento conserva su propia idea de qué es válido, repartida entre condiciones de su código, y esas ideas se separan hasta que una petición falla en producción. Un esquema reúne las reglas en un lugar y en una forma legible tanto para personas como para programas, y puede almacenarse, versionarse, revisarse y publicarse como cualquier archivo.

La especificación plantea un propósito amplio: JSON Schema sirve para validar y documentar datos JSON, para navegar mediante hipervínculos y para controlar la interacción; el vocabulario de validación también pretende dar indicaciones a las interfaces que muestran los datos. En la práctica, una descripción puede rechazar entradas incorrectas en el límite de un servicio, documentar una API, guiar el funcionamiento de un formulario y comprobar que el JSON devuelto por un modelo de lenguaje contiene los campos que espera el siguiente paso.

También se ha convertido en la base común de otras especificaciones. OpenAPI 3.1 define su Schema Object como un superconjunto de JSON Schema 2020-12, de modo que los cuerpos de petición y respuesta de una descripción de API son JSON Schema con unos pocos añadidos. MCP, el protocolo con el que los agentes de IA descubren e invocan herramientas, describe los parámetros de cada herramienta mediante un `inputSchema` que es un JSON Schema, y puede describir el resultado mediante un `outputSchema` con el que los clientes deberían validar los resultados estructurados. Así, una misma descripción puede servir a un validador, a un documento de API y a un agente.

Cómo se aplica en la práctica

El esquema siguiente describe la entrada de una herramienta que reserva una franja de entrega. Indica la versión en la que está escrito, enumera cuatro propiedades con sus reglas, señala cuáles son las tres obligatorias y rechaza cualquier propiedad no enumerada:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/tools/book-slot.json",
  "title": "Book a delivery slot",
  "type": "object",
  "properties": {
    "order": { "type": "string", "pattern": "^[0-9]{4,8}$" },
    "date": { "type": "string", "format": "date" },
    "pallets": { "type": "integer", "minimum": 1, "maximum": 30 },
    "note": { "type": ["string", "null"], "maxLength": 200 }
  },
  "required": ["order", "date", "pallets"],
  "additionalProperties": false
}
```

Una instancia que incumple las reglas:

```
{ "order": "4471", "date": "27/10/2026", "pallets": 0, "pallet": 2 }
```

La instancia falla por dos motivos. `pallets` está por debajo del mínimo de 1, y `pallet`, una errata de `pallets`, es una propiedad que el esquema no enumera. La ausencia de `note` no es un fallo, porque `note` no está en `required`; si se hubiera enviado como `null`, también sería válido, ya que su tipo admite null. La fecha es el caso instructivo. El formato `date` exige el año primero y `27/10/2026`

pone el día primero, pero un validador con su configuración predeterminada lo acepta, porque la especificación hace que `format` sea una anotación que solo se comprueba cuando se activa esa comprobación. Actívela en un validador que compruebe fechas y la misma instancia fallará por tres motivos.

Una herramienta descrita así es el tema de la página de este sitio sobre convertir software existente en herramientas para agentes de IA. Presenta cada operación que debería tener un agente, como reservar una franja o liberar un pedido, como una herramienta con entradas tipadas, errores ante los que el agente puede actuar y llamadas idempotentes para que un reintento nunca reserve dos veces. La descripción es un documento OpenAPI generado a partir del código, y un servidor MCP ofrece las mismas operaciones. La página sostiene que un agente confía en la descripción de una herramienta aunque esté desactualizada, y que una descripción escrita a mano puede alejarse de la implementación. Por ello, las descripciones se generan desde el código hacia fuera, pasando por la base de datos, la API, la descripción y el cliente, con una comprobación en cada paso que hace fallar la compilación cuando dos discrepan.

Dónde surgen los problemas

Una propiedad enumerada que nunca es obligatoria. `properties` solo comprueba las propiedades presentes, por lo que un objeto que las omite todas pasa. La presencia se establece con otra palabra clave, y omitir `required` significa que no hay nada obligatorio. Escriba siempre `required` junto a `properties` y conserve un caso de prueba por cada propiedad obligatoria en el que se haya eliminado esa propiedad.

Erratas que pasan como propiedades adicionales. Sin `additionalProperties`, se permite cualquier propiedad que el esquema no enumere, por lo que `pallet` pasa y la verdadera `pallets` falta sin que se advierta. Establecerla en `false` cierra el objeto, pero la palabra clave solo ve las `properties` y `patternProperties` de su propio objeto de esquema. Un esquema que se basa en otro mediante `allOf` o `$ref` y después se cierra así rechaza precisamente las propiedades heredadas. La versión 2020-12 ofrece `unevaluatedProperties` para ese caso, porque también cuenta las propiedades comprobadas mediante `allOf`, `$ref` y las demás palabras clave que aplican esquemas en el mismo lugar. Utilice `additionalProperties: false` en un esquema independiente y `unevaluatedProperties: false` en uno compuesto.

Un `format` que no comprueba nada. De forma predeterminada, `format` es una anotación, no una aserción: la especificación exige que un validador capaz de comprobar formatos mantenga esa comprobación desactivada salvo que se solicite. Incluso activada, un validador puede dejar pasar cualquier formato sin comprobarlo. OpenAPI 3.1 sigue la misma regla y añade que el grado de comprobación varía entre implementaciones. Active la comprobación de formatos en su validador, demuéstrela con un caso de prueba que deba fallar y, cuando un formato importe pero no se pueda comprobar, añada un `pattern` o una comprobación en el código.

Patrones que coinciden con una parte del valor. Un `pattern` no está anclado, por lo que `[0-9]{4}` acepta `order 4471b` porque en alguna parte aparecen cuatro cifras; el propio ejemplo de la especificación es que `es` coincide con `expression`. Inicie el patrón con `^` y termínelo con `$` cuando deba coincidir el valor completo. Los patrones siguen el dialecto ECMA-262 y la especificación recomienda un conjunto reducido de construcciones para que se comporten igual en todas partes; limítese, por tanto, a clases, cuantificadores y anclas sencillos.

Tratar ausencia, null y vacío como lo mismo. Una propiedad ausente, una establecida en `null` y una cadena vacía son tres instancias distintas. `"type": "string"` rechaza `null`, y una propiedad no incluida en `required` puede estar ausente. Un cliente que envía `null` para indicar que no hay valor, frente a un esquema que espera ausencia, falla en todas esas peticiones. Decida para cada propiedad qué significa cada uno de los tres casos y déjelo escrito: `["string", "null"]` cuando null tiene significado, exclusión de la propiedad de `required` cuando es opcional y `minLength: 1` cuando una cadena vacía no es un valor.

Falta `$schema` o se mezclan dos versiones. Cuando la raíz de un esquema no tiene `$schema`, la especificación deja el comportamiento a la implementación y cada validador aplica su versión predeterminada. Las versiones difieren en puntos importantes. En 2019-09, un array pasado a `items` describía una tupla; en 2020-12 esa función corresponde a `prefixItems`, e `items` asumió el papel de `additionalItems`. Incluya `$schema` en cada raíz, siga las notas de versión al migrar y valide todos los esquemas frente a sus metaesquemas en la integración continua.

Lecturas adicionales

- Especificación de JSON Schema — el índice del proyecto con la versión actual, sus metaesquemas y las notas de versión de todos los borradores anteriores.
- JSON Schema: un tipo de medio para describir documentos JSON — Core 2020-12: cómo se procesan los esquemas, `$schema`, `$ref` y `$defs`, y las palabras clave que combinan esquemas.
- Validación de JSON Schema: vocabulario para la validación estructural de JSON — las palabras clave para cada tipo de valor, `required` y las reglas de `format`.
- Notas de versión de JSON Schema 2020-12 — los cambios desde 2019-09, con `prefixItems` y `$dynamicRef` junto a lo que sustituyeron.
- Especificación de OpenAPI v3.1.1 — Schema Object como superconjunto de JSON Schema 2020-12 y `format` como anotación predeterminada.
- Model Context Protocol: herramientas — cómo declara una herramienta sus parámetros y su salida estructurada en JSON Schema.

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 Schema?» 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-schema/