Engineer

JSON Schema

Sprache zur Beschreibung der Struktur von JSON-Daten

Was es ist

JSON Schema ist eine Sprache, die beschreibt, wie JSON-Daten aussehen müssen, und selbst in JSON geschrieben wird. Eine solche Beschreibung, ein Schema, legt die Regeln fest: Dieser Wert ist ein Objekt, es muss diese Eigenschaften besitzen, dieser Wert ist eine ganze Zahl von mindestens eins, jener eine Zeichenkette mit höchstens 200 Zeichen. Ein als Validator bezeichnetes Programm erhält ein Schema und eine Instanz, also die zu prüfenden Daten. Es gibt an, ob die Instanz den Regeln entspricht, und falls nicht, wo und warum. Das Projekt schreibt seinen Namen mit großem S, JSON Schema. Ein einzelnes in der Sprache geschriebenes Dokument heißt dagegen auf Englisch schlicht schema mit kleinem s.

Die Spezifikation wird vom Projekt JSON Schema auf json-schema.org gepflegt. Die aktuelle Version ist 2020-12, die vorherige 2019-09. Frühere Versionen waren nummerierte Entwürfe wie draft-04 und draft-07, auf die ältere Werkzeuge und Artikel weiterhin unter diesen Namen verweisen. Der Text besteht aus zwei Teilen. Das Kerndokument definiert die Verarbeitung eines Schemas, die Verweise zwischen Schemas und den Medientyp `application/schema+json`. Ein zweites Dokument zur Validierung definiert die Schlüsselwörter für Wertprüfungen. Beide aktuellen Dokumente für 2020-12 sind auf den 16. Juni 2022 datiert.

Jedes Schlüsselwort ergänzt eine Einschränkung. `type` nennt einen von sechs Grundtypen, string, number, boolean, array, object oder null, oder `integer`, das nur Zahlen mit einem Nachkommateil von null akzeptiert. Für Zahlen gibt es `minimum`, `maximum` und `multipleOf`; für Zeichenketten `minLength`, `maxLength` und `pattern`, einen regulären Ausdruck; für Arrays `minItems`, `maxItems` und `uniqueItems`; für Objekte `properties`, `required` und `additionalProperties`. Schlüsselwörter wie `allOf`, `anyOf`, `oneOf` und `not` kombinieren Schemas. `$ref` verweist auf ein andernorts definiertes Schema, oft unter `$defs` im

selben Dokument, damit eine einmal beschriebene Struktur vielerorts verwendet werden kann. Was ein Schema nicht erwähnt, ist erlaubt. Das leere Schema `{}` akzeptiert jede beliebige Instanz.

Ein Schema kann auch angeben, in welcher Sprachversion es geschrieben ist. Das Schlüsselwort `$schema` enthält die Adresse eines Metaschemas, das gültige Schemas dieser Version beschreibt. Die Spezifikationsseite hält fest, dass das Metaschema sich selbst validiert. Für 2020-12 lautet diese Adresse `https://json-schema.org/draft/2020-12/schema` .

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist JSON Schema?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Der Teil, den Nutzer nie sehen: wo die Daten liegen, welche Regeln gelten und wie andere Systeme darauf zugreifen dürfen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

JSON sagt nichts über Bedeutung aus. Jedes wohlgeformte Dokument lässt sich parsen, unabhängig davon, ob das lesende Programm damit etwas anfangen kann. Ohne schriftliche Beschreibung hat jedes lesende Programm eine eigene Vorstellung von gültigen Daten, verteilt auf Bedingungen in seinem Code. Diese Vorstellungen driften auseinander, bis eine Anfrage im Produktionsbetrieb scheitert. Ein Schema bündelt die Regeln in einer für Menschen und Programme lesbaren Form. Es lässt sich wie jede andere Datei speichern, versionieren, prüfen und veröffentlichen.

Die Spezifikation fasst ihren Zweck weit: JSON Schema dient der Validierung und Dokumentation von JSON-Daten, der Navigation über Hyperlinks und der Interaktionssteuerung. Das Validierungsvokabular soll außerdem Oberflächen, die Daten anzeigen, Hinweise über diese Daten geben. Praktisch kann eine Beschreibung damit fehlerhafte Eingaben am Eingang eines Dienstes abweisen, eine API dokumentieren, ein Formular steuern und prüfen, ob die JSON-Ausgabe eines Sprachmodells die vom nächsten Schritt erwarteten Felder enthält.

JSON Schema ist auch zur gemeinsamen Grundlage anderer Spezifikationen geworden. OpenAPI 3.1 definiert sein Schema Object als Obermenge von JSON Schema 2020-12. Die Anfrage- und Antwortbodys einer API-Beschreibung sind daher JSON Schema mit einigen Ergänzungen. MCP, das Protokoll, über das KI-Agenten Werkzeuge finden und aufrufen, beschreibt die Parameter jedes Werkzeugs mit einem `inputSchema` in JSON Schema. Es kann dessen Ergebnis außerdem mit einem `outputSchema` beschreiben, gegen das Clients strukturierte Ergebnisse prüfen sollten. Eine gemeinsame Beschreibung kann damit einem Validator, einem API-Dokument und einem Agenten dienen.

Wie es in der Praxis aussieht

Das folgende Schema beschreibt die Eingabe eines Werkzeugs zum Buchen eines Lieferzeitfensters. Es nennt seine Sprachversion, führt vier Eigenschaften mit ihren Regeln auf, legt die drei erforderlichen fest und weist jede nicht aufgeführte Eigenschaft zurück:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$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
}
```

Eine Instanz, die die Regeln verletzt:

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

Die Instanz scheitert aus zwei Gründen. `pallets` liegt unter dem Mindestwert 1. `pallet`, ein Tippfehler für `pallets`, ist eine nicht aufgeführte Eigenschaft. Das fehlende `note` ist kein Fehler, denn `note` steht nicht in `required`. Auch als `null` gesendet wäre es gültig, weil sein Typ null erlaubt. Lehrreich ist das Datum: Das Format `date` verlangt zuerst das Jahr, `27/10/2026` stellt dagegen

den Tag voran. Ein Validator mit seinen Voreinstellungen akzeptiert es trotzdem, denn laut Spezifikation ist `format` eine Annotation, deren Prüfung eigens eingeschaltet werden muss. Schalten Sie diese Prüfung in einem Validator ein, der Datumswerte prüft, und dieselbe Instanz scheitert aus drei Gründen.

Solche Werkzeuge sind das Thema der Seite dieser Website, die beschreibt, wie man bestehende Software zu Werkzeugen für KI-Agenten macht. Sie beschreibt, wie jede für einen Agenten vorgesehene Operation, etwa die Buchung eines Zeitfensters oder die Freigabe eines Auftrags, als Werkzeug angeboten wird: mit typisierten Eingaben, Fehlern, auf die ein Agent reagieren kann, und idempotenten Aufrufen, damit ein erneuter Versuch nie doppelt bucht. Die Beschreibung ist ein aus dem Code erzeugtes OpenAPI-Dokument, und ein MCP-Server bietet dieselben Operationen an. Die Seite argumentiert, dass ein Agent der Beschreibung eines Werkzeugs auch dann vertraut, wenn sie veraltet ist, und eine von Hand geschriebene Beschreibung von der Implementierung abweichen kann. Deshalb werden die Beschreibungen dort vom Code ausgehend durch Datenbank, API, Beschreibung und Client hindurch erzeugt. Bei jedem Schritt lässt eine Prüfung den Build scheitern, wenn zwei davon voneinander abweichen.

Wo es schiefgeht

Eine aufgeführte Eigenschaft ist nie erforderlich. `properties` prüft nur vorhandene Eigenschaften. Ein Objekt, das alle weglässt, besteht daher die Prüfung. Für das Vorhandensein gibt es ein eigenes Schlüsselwort. Ohne `required` ist überhaupt nichts erforderlich. Schreiben Sie jedes Mal `required` neben `properties` und behalten Sie für jede erforderliche Eigenschaft einen Testfall bei, in dem genau diese Eigenschaft fehlt.

Tippfehler werden als zusätzliche Eigenschaften akzeptiert. Ohne `additionalProperties` ist jede nicht aufgeführte Eigenschaft erlaubt. So kommt `pallet` durch und das eigentliche `pallets` fehlt unbemerkt. Der Wert `false` schließt das Objekt, doch das Schlüsselwort sieht nur `properties` und `patternProperties` im eigenen Schemaobjekt. Baut ein Schema über `allOf` oder `$ref` auf einem anderen auf und schließt sich dann so, weist es gerade seine geerbten Eigenschaften zurück. Für diesen Fall hat Version 2020-12 `unevaluatedProperties`, das auch Eigenschaften mitzählt, die über `allOf`, `$ref` und andere Schlüsselwörter für die Anwendung von Schemas an derselben Stelle geprüft wurden. Verwenden Sie `additionalProperties: false` bei einem eigenständigen Schema und `unevaluatedProperties: false` bei einem zusammengesetzten.

`format` **prüft nichts.** Standardmäßig ist `format` eine Annotation und keine Prüfbedingung: Laut Spezifikation muss ein Validator, der Formate prüfen kann, diese Prüfung ausgeschaltet lassen, sofern sie nicht angefordert wird. Selbst bei eingeschalteter Prüfung darf ein Validator jedes Format ungeprüft passieren lassen. OpenAPI 3.1 folgt derselben Regel und ergänzt, dass der Umfang der Formatprüfung zwischen Implementierungen schwankt. Schalten Sie die Formatprüfung Ihres Validators ein und belegen Sie sie mit einem Testfall, der scheitern muss. Ist ein wichtiges Format nicht prüfbar, ergänzen Sie ein `pattern` oder eine Prüfung im Code.

Muster akzeptieren schon einen Teil des Werts. Ein `pattern` ist nicht verankert. Deshalb akzeptiert `[0-9]{4}` den Wert `order 4471b`, weil irgendwo darin vier Ziffern vorkommen. Das eigene Beispiel der Spezifikation lautet, dass `es` auf `expression` passt. Beginnen Sie das Muster mit `^` und beenden Sie es mit `$`, wenn der gesamte Wert passen muss. Muster folgen dem ECMA-262-Dialekt. Für überall gleiches Verhalten empfiehlt die Spezifikation eine kleine Auswahl von Konstrukten. Bleiben Sie daher bei einfachen Zeichenklassen, Quantifizierern und Ankern.

Fehlend, null und leer gelten als dasselbe. Eine fehlende Eigenschaft, eine auf `null` gesetzte Eigenschaft und eine leere Zeichenkette sind drei unterschiedliche Instanzen. `"type": "string"` weist `null` zurück, und eine nicht in `required` aufgeführte Eigenschaft darf fehlen. Sendet ein Client `null` für „kein Wert“, während das Schema eine fehlende Eigenschaft vorsieht, scheitert jede solche Anfrage. Entscheiden Sie je Eigenschaft, was die drei Fälle bedeuten, und halten Sie es fest: `["string", "null"]`, wenn null eine Bedeutung hat, Weglassen aus `required`, wenn die Eigenschaft optional ist, und `minLength: 1`, wenn eine leere Zeichenkette kein Wert ist.

Kein `$schema` oder zwei vermischte Versionen. Enthält das Wurzelobjekt eines Schemas kein `$schema`, überlässt die Spezifikation das Verhalten der Implementierung. Jeder Validator wendet dann seine voreingestellte Version an. Die Versionen unterscheiden sich an entscheidenden Stellen. In 2019-09 beschrieb ein an `items` übergebenes Array ein Tupel. In 2020-12 übernimmt `prefixItems` diese Aufgabe, während `items` die Rolle von `additionalItems` übernommen hat. Setzen Sie `$schema` in jede Schemawurzel, beachten Sie bei Versionswechseln die Versionshinweise und validieren Sie in CI jedes Schema gegen sein Metaschema.

Weiterführende Lektüre

- JSON-Schema-Spezifikation — Der Projektindex mit aktueller Version, ihren Metaschemas und den Versionshinweisen zu allen früheren Entwürfen.
- JSON Schema: Ein Medientyp zur Beschreibung von JSON-Dokumenten — Core 2020-12: Verarbeitung von Schemas, `$schema`, `$ref` und `$defs` sowie die Schlüsselwörter zum Kombinieren von Schemas.
- JSON-Schema-Validierung: Ein Vokabular zur strukturellen Validierung von JSON — Schlüsselwörter je Werttyp, `required` und die Regeln für `format`.
- JSON Schema 2020-12: Versionshinweise — Änderungen seit 2019-09, darunter `prefixItems` und `$dynamicRef` im Vergleich zu ihren Vorgängern.
- OpenAPI-Spezifikation v3.1.1 — Das Schema Object als Obermenge von JSON Schema 2020-12 und `format` als standardmäßige Annotation.
- Model Context Protocol: Werkzeuge — Wie ein Werkzeug seine Parameter und seine strukturierte Ausgabe in JSON Schema deklariert.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist JSON Schema?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/json-schema/