Engineer

JSON Schema

Langage décrivant la structure valide de données JSON

Ce que c’est

JSON Schema est un langage qui décrit la forme attendue de données JSON et qui s’écrit lui-même en JSON. Une description dans ce langage, appelée schéma, pose les règles : cette valeur est un objet, il doit posséder ces propriétés, celle-ci est un entier d’au moins un, celle-là une chaîne d’au plus 200 caractères. Un programme appelé validateur reçoit un schéma et une instance, les données à vérifier, puis indique si l’instance est conforme et, sinon, où et pourquoi. Le projet écrit son nom avec un S majuscule, JSON Schema, tandis qu’un document écrit dans ce langage est simplement un schéma, en minuscule.

La spécification est maintenue par le projet JSON Schema sur json-schema.org. La version actuelle est 2020-12 et la précédente 2019-09 ; les plus anciennes étaient des brouillons numérotés, comme draft-04 et draft-07, et les anciens outils et articles les désignent encore par ces noms. Le texte comprend deux parties. Le document central définit le traitement d’un schéma, les références entre schémas et le type de média `application/schema+json` ; un second document, sur la validation, définit les mots-clés qui vérifient les valeurs. Les deux documents actuels de 2020-12 sont datés du 16 juin 2022.

Chaque mot-clé ajoute une contrainte. `type` nomme l’un des six types de base, string (chaîne), number (nombre), boolean (booléen), array (tableau), object (objet) ou null (nul), ou bien `integer`, qui n’accepte que les nombres dont la partie fractionnaire est nulle. Les nombres prennent `minimum`, `maximum` et `multipleOf` ; les chaînes, `minLength`, `maxLength` et `pattern`, une expression régulière ; les tableaux, `minItems`, `maxItems` et `uniqueItems` ; les objets, `properties`, `required` et `additionalProperties`. Des mots-clés comme `allOf`, `anyOf`, `oneOf` et `not` combinent les schémas, et `$ref` pointe vers un schéma défini ailleurs, souvent sous `$defs` dans le même document, pour réutiliser

en plusieurs endroits une forme écrite une seule fois. Tout ce qu'un schéma ne mentionne pas est autorisé. Le schéma vide, `{}`, accepte toutes les instances possibles.

Un schéma peut aussi indiquer la version du langage dans laquelle il est écrit. Le mot-clé `$schema` contient l'adresse d'un métaschéma, un schéma qui décrit les schémas valides pour cette version ; la page de spécification précise que le métaschéma se valide lui-même. Pour 2020-12, cette adresse est `https://json-schema.org/draft/2020-12/schema`.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que JSON Schema ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. La partie invisible pour les utilisateurs : où résident les données, quelles règles s'appliquent et comment les autres systèmes peuvent y accéder.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 6 pipelines ETL/ELT conçus et exploités — 6 pipelines ETL sur trois moteurs de bases de données

6 pipelines ETL/ELT conçus et exploités Contactez-nous →

Pourquoi c'est important

JSON ne dit rien du sens. Tout document bien formé peut être analysé, que le programme qui le lit puisse ou non en faire quelque chose. Sans description écrite, chaque programme qui lit un document garde sa propre idée de ce qui est valide, éparpillée dans les conditions de son code ; ces idées divergent jusqu'à ce qu'une requête échoue en production. Un schéma rassemble les règles dans une forme lisible par les personnes comme par les programmes. Il peut être stocké, versionné, revu et publié comme n'importe quel fichier.

La spécification énonce un objectif large : JSON Schema est destiné à la validation et à la documentation des données JSON, à la navigation par hyperliens et au contrôle des interactions ; le vocabulaire de validation vise aussi à fournir des indications aux interfaces qui affichent les données. En pratique, une même description peut rejeter les entrées incorrectes à la frontière d'un service, documenter une API, piloter un formulaire et vérifier que le JSON renvoyé par un modèle de langage possède les champs attendus à l'étape suivante.

Il est aussi devenu le socle commun d'autres spécifications. OpenAPI 3.1 définit son Schema Object comme un surensemble de JSON Schema 2020-12 : les corps de requête et de réponse d'une description d'API sont donc du JSON Schema avec quelques ajouts. MCP, le protocole par lequel les agents d'IA découvrent et appellent des outils, décrit les paramètres de chaque outil par un `inputSchema` qui est un JSON Schema. MCP peut aussi décrire le résultat de l'outil par un `outputSchema` auquel les clients devraient confronter les résultats structurés pour les valider. Une même description peut donc servir à un validateur, à un document d'API et à un agent.

À quoi cela ressemble en pratique

Le schéma ci-dessous décrit l'entrée d'un outil qui réserve un créneau de livraison. Il nomme la version dans laquelle il est écrit, énumère quatre propriétés avec leurs règles, indique les trois qui doivent être présentes et refuse toute propriété non répertoriée :

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

Une instance qui enfreint les règles :

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

L'instance échoue sur deux points. `pallets` est inférieur au minimum de 1, et `pallet`, une faute de frappe pour `pallets`, est une propriété absente de la liste du schéma. L'absence de `note` n'est pas une erreur, car `note` ne figure pas dans `required` ; si elle avait été envoyée avec la valeur `null`, elle serait aussi acceptée, puisque son type autorise null. La date est le cas instructif. Le format

`date` exige l'année en premier et `27/10/2026` place le jour en premier, mais un validateur avec ses réglages initiaux l'accepte : la spécification fait de `format` une annotation qui n'est vérifiée que si cette vérification est activée. Activez-la dans un validateur qui vérifie les dates, et la même instance échouera sur trois points.

C'est d'un outil ainsi décrit que traite la page de ce site sur la transformation de logiciels existants en outils pour agents d'IA. Elle présente chaque opération dont un agent devrait disposer, comme réserver un créneau ou libérer une commande, sous la forme d'un outil aux entrées typées, aux erreurs auxquelles un agent peut réagir utilement et aux appels idempotents, afin qu'une nouvelle tentative ne réserve jamais deux fois. La description est un document OpenAPI généré depuis le code, et un serveur MCP propose les mêmes opérations. La page souligne qu'un agent fait confiance à la description d'un outil même lorsqu'elle est périmée, et qu'une description écrite à la main peut diverger de l'implémentation. Les descriptions sont donc générées depuis le code vers l'extérieur, en passant par la base de données, l'API, la description et le client, avec à chaque étape une vérification qui fait échouer le processus de construction lorsque deux éléments divergent.

Où les problèmes surviennent

Une propriété répertoriée qui n'est jamais obligatoire. `properties` ne vérifie que les propriétés présentes : un objet qui les omet toutes est donc accepté. La présence relève d'un autre mot-clé, et omettre `required` signifie que rien n'est obligatoire. Écrivez toujours `required` à côté de `properties` et conservez un cas de test par propriété obligatoire, où cette propriété a été retirée.

Des fautes de frappe acceptées comme propriétés supplémentaires. Sans `additionalProperties`, toute propriété non répertoriée par le schéma est autorisée : `pallet` passe et la véritable propriété `pallets` manque silencieusement. Régler ce mot-clé sur `false` ferme l'objet, mais ce mot-clé ne voit que les `properties` et `patternProperties` de son propre objet de schéma. Un schéma qui en prolonge un autre par `allOf` ou `$ref` puis se ferme ainsi rejette précisément les propriétés héritées. La version 2020-12 propose `unevaluatedProperties` pour ce cas, car ce mot-clé compte aussi les propriétés vérifiées par `allOf`, `$ref` et les autres mots-clés qui appliquent des schémas au même emplacement de l'instance. Utilisez `additionalProperties: false` pour un schéma autonome et `unevaluatedProperties: false` pour un schéma composé.

Un `format` qui ne vérifie rien. Sans activation explicite de sa vérification, `format` est une annotation et non une assertion : la spécification exige qu'un validateur capable de vérifier les formats garde cette vérification désactivée sauf demande explicite. Même lorsque cette vérification est activée, un validateur peut laisser passer n'importe quel format sans le vérifier. OpenAPI 3.1 suit la même règle et ajoute que la qualité de la vérification varie selon l'implémentation. Activez la vérification des formats dans votre validateur, prouvez-la avec un cas de test qui doit échouer et, lorsqu'un format compte mais ne peut être vérifié, ajoutez un `pattern` ou une vérification dans le code.

Des motifs qui correspondent à une partie d'une valeur. Un `pattern` n'est pas ancré : `[0-9]{4}` accepte donc `order 4471b`, car quatre chiffres s'y trouvent quelque part ; l'exemple de la spécification est que `es` correspond à `expression`. Commencez le motif par `^` et terminez-le par `$` lorsque la valeur entière doit correspondre. Les motifs suivent le dialecte ECMA-262, et la spécification recommande un petit ensemble de constructions pour qu'ils se comportent partout de la même manière : tenez-vous-en aux classes, quantificateurs et ancrages simples.

Confondre absence, null et vide. Une propriété absente, une propriété ayant la valeur `null` et une chaîne vide sont trois instances différentes. `"type": "string"` rejette `null`, et une propriété absente de `required` peut être omise. Un client qui envoie `null` pour signifier l'absence de valeur face à un schéma prévu pour l'absence de propriété échoue à chaque requête de ce type. Décidez pour chaque propriété du sens de ces trois cas et écrivez-le : `["string", "null"]` lorsque null a un sens, l'absence de la propriété dans `required` lorsqu'elle est facultative, et `minLength: 1` lorsqu'une chaîne vide n'est pas une valeur.

Pas de `$schema`, ou deux versions mélangées. Lorsque la racine d'un schéma n'a pas de `$schema`, la spécification laisse le comportement à l'implémentation et chaque validateur applique sa version prédéfinie. Les versions diffèrent sur des points importants. En 2019-09, un tableau fourni à `items` décrivait un tuple ; en 2020-12, ce rôle appartient à `prefixItems`, et `items` a repris celui d'`additionalItems`. Placez `$schema` dans chaque racine, suivez les notes de version lors d'une migration et validez chaque schéma avec son métaschéma en intégration continue.

Pour aller plus loin

- Spécification de JSON Schema — l'index du projet pour la version actuelle, ses métaschémas et les notes de version de chaque ancien brouillon.
- JSON Schema : un type de média pour décrire des documents JSON — Core 2020-12 : traitement des schémas, `$schema`, `$ref` et `$defs`, et mots-clés qui combinent les schémas.
- Validation de JSON Schema : vocabulaire de validation structurelle de JSON — mots-clés de chaque type de valeur, `required` et règles de `format`.
- Notes de version de JSON Schema 2020-12 — changements depuis 2019-09, avec `prefixItems` et `$dynamicRef` mis en regard de ce qu'ils ont remplacé.
- Spécification OpenAPI v3.1.1 — Schema Object comme surensemble de JSON Schema 2020-12 et `format` comme annotation sans activation explicite de sa vérification.
- Model Context Protocol : outils — déclaration des paramètres et de la sortie structurée d'un outil en JSON Schema.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que JSON Schema ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/json-schema/