Engineer

JSON Schema

мова для опису структури даних у форматі JSON

Що це таке

JSON Schema — це мова для опису того, яким має бути фрагмент JSON, і пишуть цією мовою теж у JSON. Опис нею, який називають схемою, задає правила: це значення — об’єкт, у нього мають бути такі властивості, ця властивість — ціле число, не менше за одиницю, а та — рядок щонайбільше з 200 символів. Програма, яку називають валідатором, бере схему й екземпляр, тобто дані, які перевіряють, і відповідає, чи екземпляр відповідає схемі, а якщо ні, то де саме і чому. Проєкт пише свою назву з великою S, JSON Schema, а окремий документ, написаний цією мовою, англійською називають просто schema, з малої літери, а українською — схемою.

Специфікацію підтримує проєкт JSON Schema на json-schema.org. Поточна версія — 2020-12, попередня — 2019-09; ранніші версії мали номери чернеток, як-от draft-04 і draft-07, і старіші інструменти та статті досі називають їх так. Текст складається з двох частин. Основний документ, Core, визначає, як обробляють схему, як одна схема посилається на іншу, а також медіатип `application/schema+json`, а другий документ, про валідацію, визначає ключові слова, що перевіряють значення. Чинні документи версії 2020-12 обидва датовано 16 червня 2022 року.

Кожне ключове слово додає одне обмеження. `type` називає один із шести примітивних типів, тобто null, boolean, object, array, number або string, чи `integer`, якому відповідає будь-яке число з нульовою дробовою частиною. Для чисел є `minimum`, `maximum` і `multipleOf`; для рядків — `minLength`, `maxLength` і `pattern`, регулярний вираз; для масивів — `minItems`, `maxItems` і `uniqueItems`; для об’єктів — `properties`, `required` і `additionalProperties`. Ключові слова `allOf`, `anyOf`, `oneOf` і `not` поєднують схеми, а `$ref` указує на схему, визначену деінде, часто в `$defs` того самого документа,

тож структуру, описану один раз, можна використовувати в багатьох місцях. Усе, про що схема не згадує, дозволено. Порожня схема, `{}`, приймає будь-який екземпляр.

Схема може також указати, якою версією мови її написано. Ключове слово `$schema` містить адресу метасхеми, тобто схеми, що описує правильні схеми для цієї версії, і сторінка специфікації зазначає, що метасхема перевіряє сама себе. Для 2020-12 ця адреса — `https://json-schema.org/draft/2020-12/schema`.

Застосуйте поняття до рішення

Застосуйте «Що таке JSON Schema?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Та частина, якої користувач не бачить: де зберігаються дані, які діють правила і як інші системи можуть із нею взаємодіяти.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Спроєктувала й підтримувала 6 пайплайнів ETL/ELT — 6 пайплайнів ETL на трьох рушіях баз даних

Спроєктувала й підтримувала 6 пайплайнів ETL/ELT Зв'язатися з нами →

Чому це важливо

JSON нічого не каже про зміст. Будь-який синтаксично правильний документ розбирається, хоч би що з ним могла зробити програма, яка його читає. Без письмового опису кожна програма, що читає документ, має власне уявлення про те, які дані правильні, розкидане по умовах у її коді, і ці уявлення розходяться, доки якийсь запит не впаде в робочому середовищі. Схема збирає правила в одному місці, у формі, яку можуть прочитати й люди, й програми, і її можна зберігати, версіонувати, рецензувати та публікувати, як будь-який інший файл.

Специфікація формулює своє призначення широко: JSON Schema призначена для валідації, документування, навігації гіперпосиланнями й керування взаємодією з даними JSON, а словник валідації має також давати екранам, що показують ці дані, підказки про них. На практиці це означає, що один опис може відхиляти хибні вхідні дані на межі сервісу, документувати API, будувати форму і перевіряти, що JSON, який повертає мовна модель, має поля, яких очікує наступний крок.

JSON Schema стала й спільною основою, на якій будують інші специфікації. OpenAPI 3.1 визначає свій Schema Object як надмножину JSON Schema 2020-12, тож тіла запитів і відповідей в описі API — це JSON Schema з кількома

доповненнями. MCP, протокол, через який ШІ-агенти знаходять і викликають інструменти, описує параметри кожного інструмента полем `inputSchema`, що є JSON Schema, і може описувати його результат полем `outputSchema`, за яким клієнтам слід перевіряти структуровані результати. Тож спільний опис може служити валідатору, документу API та агентові.

Як це виглядає на практиці

Схема нижче описує вхідні дані інструмента, який бронює слот доставки. Вона називає версію, якою написана, перелічує чотири властивості з їхніми правилами, указує, які три мають бути присутні, і відхиляє будь-яку властивість, якої не перелічено:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/tools/book-slot.json",
  "title": "Забронювати слот доставки",
  "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
}
```

Екземпляр, що порушує правила:

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

Цей екземпляр не проходить із двох причин. `pallets` менше за мінімум 1, а `pallet`, одруківка в `pallets`, — властивість, якої схема не перелічує. Відсутній `note` помилкою не є, бо `note` немає в `required`; якби його надіслали як `null`, це теж пройшло б, бо його тип допускає null. Найповчальніша тут дата. Формат `date` вимагає, щоб рік ішов першим, а `27/10/2026` записано не так, проте валідатор із налаштуваннями за замовчуванням її приймає, бо специфікація робить `format` анотацією,

яку перевіряють, лише коли таку перевірку ввімкнено. Увімкніть її у валідаторі, що вміє перевіряти дати, і той самий екземпляр не пройде вже з трьох причин.

Саме про інструменти, описані так, ідеться на сторінці цього сайту ваші програми як інструменти для ШІ-агентів. Вона описує кожну операцію, потрібну агентові, як-от забронювати слот чи відвантажити замовлення, як інструмент із типізованими вхідними даними, помилками, на які агент може зреагувати, та ідемпотентними викликами, тож повторна спроба не створить другого бронювання. Опис — це документ OpenAPI, згенерований із коду, а MCP-сервер надає ті самі операції. Головна думка сторінки така: агент вірить описові інструмента, навіть коли той уже застарів, а написаний вручну опис може розійтися з реалізацією. Тому описи генеруються з коду, ланцюжком від бази даних через API й опис до клієнта, а перевірка на кожній ланці зупиняє збірку, щойно дві з них розходяться.

Де виникають проблеми

Перелічена властивість, яку ніколи не вимагають. `properties` перевіряє лише ті властивості, що присутні, тож об'єкт, у якому немає жодної з них, проходить. Присутність задає окреме ключове слово, і якщо `required` пропустити, не вимагається нічого. Пишіть `required` поруч із `properties` щоразу й тримайте тестовий кейс для кожної обов'язкової властивості, з якого цю властивість вилучено.

Одруківки, що проходять як зайві властивості. Без `additionalProperties` дозволено будь-яку властивість, якої схема не перелічує, тож `pallet` проходить, а справжньої `pallets` непомітно бракує. Значення `false` закриває об'єкт, але це ключове слово бачить лише `properties` і `patternProperties` у власному об'єкті схеми. Схема, що будується на іншій через `allOf` чи `$ref`, а потім закривається так само, відхиляє саме ті властивості, які успадкувала. У версії 2020-12 для цього випадку є `unevaluatedProperties`, бо воно враховує й властивості, перевірені через `allOf`, `$ref` та інші ключові слова, що застосовують схеми на місці. Ставте `additionalProperties: false` у схемі, що стоїть окремо, і `unevaluatedProperties: false` у складеній.

`format` **, що нічого не перевіряє.** За замовчуванням `format` — це анотація, а не твердження: специфікація вимагає, щоб валідатор, який уміє перевіряти формати, тримав цю перевірку вимкненою, доки її не попросять, і навіть

коли її ввімкнено, валідатор може пропускати будь-який формат без перевірки. OpenAPI 3.1 дотримується того самого правила й додає, що якість перевірки форматів різниться між реалізаціями. Увімкніть перевірку форматів у своєму валідаторі, доведіть це тестовим кейсом, який має впасти, а там, де формат важливий, але перевірити його не можна, додайте `pattern` або перевірку в коді.

Шаблони, що збігаються з частиною значення. `pattern` не прив'язаний до початку й кінця значення, тож `[0-9]{4}` приймає `order 4471b`, бо чотири цифри трапляються десь усередині; власний приклад специфікації — `es`, що збігається з `expression`. Починайте шаблон із `^` і закінчуйте `$`, коли має збігатися все значення. Шаблони пишуть діалектом ECMA-262, і специфікація радить невеликий набір конструкцій для шаблонів, які мають поводитися однаково скрізь, тож тримайтеся простих класів символів, квантифікаторів і якорів.

Відсутнє, null і порожнє вважають одним і тим самим. Відсутня властивість, властивість зі значенням `null` і порожній рядок — це три різні екземпляри. `"type": "string"` відхиляє `null`, а властивість, якої немає в `required`, може бути відсутня. Клієнт, що надсилає `null`, маючи на увазі «немає значення», до схеми, написаної з розрахунку на відсутність, отримує відмову на кожному такому запиті. Вирішіть для кожної властивості, що означає кожен із трьох варіантів, і запишіть це: `["string", "null"]`, коли null щось означає, відсутність у `required`, коли властивість необов'язкова, і `minLength: 1`, коли порожній рядок значенням не є.

Немає `$schema`, або змішано дві версії. Коли в корені схеми немає `$schema`, специфікація залишає поведінку на розсуд реалізації, і кожен валідатор застосовує ту версію, яка в нього за замовчуванням. Версії відрізняються там, де це важливо. У 2019-09 масив, переданий у `items`, описував кортеж; у 2020-12 це робить `prefixItems`, а `items` перебрало роль `additionalItems`. Ставте `$schema` у кожен корінь, читайте примітки до випуску, коли переходите між версіями, і перевіряйте кожну схему за її метасхемою в CI.

Що почитати далі

- JSON Schema Specification — сторінка проєкту з поточною версією, її метасхемами та примітками до випуску для кожної ранішої чернетки.

- JSON Schema: A Media Type for Describing JSON Documents — Core 2020-12: як обробляють схеми, `$schema`, `$ref` і `$defs`, а також ключові слова, що поєднують схеми.
- JSON Schema Validation: A Vocabulary for Structural Validation of JSON — ключові слова для кожного типу значень, `required` і правила для `format`.
- JSON Schema 2020-12 Release Notes — що змінилося з 2019-09: `prefixItems` і `$dynamicRef` поруч із тим, що вони замінили.
- OpenAPI Specification v3.1.1 — Schema Object як надмножина JSON Schema 2020-12 і `format` як анотація за замовчуванням.
- Model Context Protocol: Tools — як інструмент оголошує свої параметри й структурований результат у JSON Schema.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке JSON Schema?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/json-schema/