Engineer

JSON

</> текстовий формат, у якому передають більшість даних

Що це таке

JSON — це текстовий запис даних фігурними й квадратними дужками, подвійними лапками та комами, завдяки якому програма майже будь-якою мовою прочитає саме те, що записала інша програма. Абревіатура розшифровується як JavaScript Object Notation: синтаксис запозичено з того, як JavaScript записує значення, проте документ JSON не прив'язаний до жодної мови. Файли зазвичай мають розширення `.json`, а в HTTP вміст позначають медіатипом `application/json`.

Увесь формат уміщується на одній сторінці. Його словник складається з чотирьох простих типів (рядки, числа, значення `true` і `false` та `null`) і двох складених: об'єкта, тобто невпорядкованого набору пар «ім'я — значення», і масиву, тобто впорядкованого списку. Будь-що можна вкласти в будь-що. Дати, двійкові дані, коментарі й посилання між частинами документа залишено домовленостям поза форматом.

Дуглас Крокфорд уперше описав формат у RFC 4627, опублікованому IETF у липні 2006 року. Чинне визначення — RFC 8259 від грудня 2017 року. Воно узгоджене зі стандартом Ecma ECMA-404 щодо граматики й додає правила, які важать тоді, коли дані передають між двома системами: текст, що залишає замкнену екосистему, має бути в UTF-8, а відправник не має права додавати на початок мітку порядку байтів (BOM).

XML старший і виразніший, з атрибутами, просторами імен і давньою традицією мов схем. YAML у версії 1.2 став строгою надмножиною JSON і додав коментарі та легшу пунктуацію, яких хочуть у файлах, що їх редагують вручну. Двійкові формати на кшталт Protocol Buffers і CBOR жертвують читабельністю заради розміру й швидкості.

Застосуйте поняття до рішення

Застосуйте «Що таке JSON?» 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

Обираючи JSON, команда насправді обирає контракт між двома системами: форму того, що йде мережею, а іноді й форму того, що потрапляє до сховища.

У мережі JSON — вибір за замовчуванням, бо його вже розуміє все. Браузер розбирає його без бібліотек, для кожної поширеної мови є зрілий парсер, а інженер, читаючи лог, бачить, що саме було в запиті. Ціна цього вибору лежить деінде. Кожен запис повторює назви своїх полів, числа передаються десятковим текстом, який треба розібрати назад, а сам формат мовчить про те, які поля обов’язкові та якого вони типу. Двійковий формат зі схемою, як-от Protocol Buffers, компактніший і швидше розбирається, а контракт перевіряє компілятор, але людині потрібен інструмент, щоб прочитати бодай одне повідомлення. Для публічного API або такого, з яким говорить браузер, JSON — розумний вибір. Між внутрішніми сервісами, де вирішують обсяг або затримка, баланс може схилитися в інший бік.

Цю прогалину зазвичай закриває схема. JSON Schema описує, які властивості мусить мати об’єкт, яких вони типів і чи дозволені додаткові властивості, а OpenAPI спирається на неї, щоб описати цілий HTTP API. Схема — окремий документ, і вона точна лише доти, доки щось змушує її збігатися з кодом.

У сховищі питання в тому, чи належить структура до колонки JSON, чи до звичайних колонок. PostgreSQL має два типи. `json` зберігає точну копію вхідного тексту й розбирає її заново щоразу, коли до неї звертається функція, а `jsonb` зберігає розкладену двійкову форму: записувати її трохи повільніше, зате шукати в ній значно швидше, і її можна індексувати. Документація радить `jsonb` для більшості застосунків. Колонка JSON підходить для даних, форма яких змінюється від рядка до рядка або

належить клієнтові, наприклад для збережених налаштувань користувача. Поле, за яким база має з'єднувати таблиці, накладати обмеження чи рахувати суму, заслуговує на власну колонку.

Як це виглядає на практиці

На одній платформі функції PostgreSQL самі збирають повну відповідь у JSON, а Go API передає цей JSON від бази даних до браузера без змін. Обробник отримує його як `json.RawMessage` і пересилає далі, не декодуючи й не кодуючи повторно, а невеликий помічник `QueryJSON` зробив такий підхід стандартним. DTO, мапери й структури відповідей, які зазвичай накопичує багатошаровий API, зникли. Коли функція починає повертати іншу форму, та доходить до клієнта без жодної правки в коді обробника.

Збережений JSON створює іншу проблему, бо живе довше за код, який його записав. Одна платформа зберігає налаштування сповіщень та інші параметри як значення користувача, накладені на набір типових значень, і спершу зливала їх лише на одному рівні. Ключа, який пізніше додали до типових значень, бракувало в кожному рядку, збереженому раніше, тож клієнтський код читав поле як undefined і падав, причому саме в користувачів, які були з платформою найдовше. Рішенням стало глибоке злиття, що використовує типові значення як каталог: кожен ключ із каталогу з'являється в результаті, зокрема вкладені, хоч би що містив збережений блок. Міграція одразу застосувала це до наявних рядків, і фронтенд позбувся своїх захисних перевірок.

Дані також часто надходять просто як JSON. Історичні дані про електроенергію у форматі JSONL, по одній події JSON на рядок, треба було перетворити на погодинне споживання для кожної зони, і для цього створили три варіанти: на Python, JQ і SQL. Найшвидшим виявився Python: 34 мс для даних за один день і 180 мс за 30 днів.

Де виникають проблеми

Граматика настільки мала, що парсери погоджуються щодо того, який текст коректний. Проблеми народжуються в усьому, що граматика залишає відкритим.

Великий ідентифікатор повертається з іншою останньою цифрою. Граматика не обмежує числа, але більшість парсерів, зокрема в JavaScript, читає їх як 64-бітні числа з рухомою комою, які точно зберігають цілі лише до $2^{53} – 1$. RFC 8259 називає діапазон від $–(2^{53} – 1)$ до $2^{53} – 1$ тим, у якому реалізації сходяться. За цією межею ідентифікатори змінюються без попередження:

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

Надсилайте 64-бітні ідентифікатори як рядки, а суми грошей — як рядки або як ціле число найменших одиниць, але ніколи як число з рухомою комою.

Два сервіси читають той самий об’єкт по-різному. RFC 8259 радить робити імена в об’єкті унікальними, а решту залишає відкритим: натрапивши на повторене ім’я, багато парсерів залишають останнє значення, інші повідомляють про помилку, а деякі повертають усі пари. Коли шлюз перевіряє перше поле `role`, а бекенд діє за другим, перевірка нічого не захистила. Відхиляйте дубльовані імена там, де дані входять у систему. I-JSON, визначений у RFC 7493, — це профіль JSON, який їх забороняє.

Вручну відредагований файл перестає розбиратися. Коментарі й кома після останнього елемента лежать поза граматикою, тож перша людина, яка допише коментар до конфігураційного файлу або залишить зайву кому, його зламає. Деякі інструменти приймають м’який діалект, і тоді той самий файл коректний для одного читача й зламаний для наступного. Залиште JSON для даних, які пишуть машини, а файлам, які люди підтримують вручну, дайте формат, створений для редагування, як-от YAML чи TOML.

Текст приходить спотвореним, або один парсер відкидає те, що приймає інший. RFC 8259 вимагає UTF-8 між системами й забороняє мітку порядку байтів на початку. Граматика також допускає екранування на кшталт `"\uDEAD"`, тобто половину сурогатної пари без партнера, яку жоден коректний рядок Unicode вмістити не може, і бібліотеки поводяться з нею кожна по-своєму. Кодуйте в UTF-8 біля джерела й відкидайте некоректні послідовності на межі системи.

Один невеликий запит паралізує парсер. Кілька кілобайтів вкладених дужок можуть завести рекурсивний парсер на тисячі рівнів углиб, а завеликий запит розбирається повністю, перш ніж його побачить будь-яка схема. RFC 8259 дозволяє реалізації обмежувати і розмір тексту, і глибину вкладення, а API, що приймає JSON від сторонніх, має використати обидва: обмежити розмір запиту ще до читання, глибину — під час розбору, і лише після цього перевіряти форму.

Що почитати далі

- RFC 8259: The JSON Data Interchange Format — чинне визначення IETF із примітками про числа, дубльовані імена й кодування.
- ECMA-404: The JSON Data Interchange Syntax — стандарт Ecma для тієї самої граматики, на який RFC 8259 посилається як на нормативний.
- RFC 7493: The I-JSON Message Format — суворіший профіль, що вимагає UTF-8 і забороняє дубльовані імена.
- Специфікація JSON Schema — словник, яким оголошують, які властивості мусить мати документ.
- PostgreSQL: JSON Types — `json` проти `jsonb` і що кожен із них зберігає з початкового тексту.
- Go: json.RawMessage — тип, що дає програмі на Go відкласти декодування або переслати JSON без змін.

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?» 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/