Engineer

JSON

</> Format texte dans lequel circulent la plupart des données

Ce que c’est

JSON est une notation en texte brut pour des données imbriquées, écrite avec des accolades, des crochets, des guillemets doubles et des virgules, afin qu’un programme dans presque n’importe quel langage puisse relire exactement ce qu’un autre a écrit. Les lettres signifient JavaScript Object Notation, car la syntaxe vient de l’écriture des valeurs littérales en JavaScript ; pourtant, un document JSON n’appartient à aucun langage et ne nécessite pas JavaScript pour être lu. Les fichiers se terminent généralement par `.json` et, sur HTTP, le contenu porte le type de média `application/json` .

Tout le format tient sur une page. Quatre types primitifs — chaînes, nombres, les littéraux `true` et `false` , et `null` — et deux types structurés constituent son vocabulaire : un objet, ensemble non ordonné de paires nom-valeur, et un tableau, liste ordonnée. Les éléments peuvent s’imbriquer les uns dans les autres. Les dates, les données binaires, les commentaires et les références entre parties d’un document relèvent tous de conventions extérieures au format.

Douglas Crockford l’a enregistré pour la première fois auprès de l’IETF dans la RFC 4627, en juillet 2006. La définition actuelle est la RFC 8259 de décembre 2017, qui s’accorde avec la norme ECMA-404 d’Ecma sur la grammaire et ajoute les règles importantes pour les échanges entre deux systèmes. Le texte qui sort d’un écosystème fermé doit être en UTF-8, et l’émetteur ne doit pas le faire précéder d’une marque d’ordre des octets.

XML est plus ancien et plus expressif, avec ses attributs, ses espaces de noms et sa longue tradition de langages de schémas. YAML a été révisé en version 1.2 pour constituer un surensemble strict de JSON, en ajoutant les commentaires et

la ponctuation allégée souhaités dans les fichiers modifiés à la main. Les encodages binaires comme Protocol Buffers et CBOR sacrifient la lisibilité pour réduire la taille et gagner en vitesse.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que JSON ? » 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

Choisir JSON revient à choisir un contrat entre deux systèmes : la forme de ce qui circule sur le réseau et parfois celle de ce qui arrive dans le stockage.

Sur le réseau, c’est le choix courant parce que tout le monde le comprend déjà. Un navigateur l’analyse sans bibliothèque, tout langage courant possède un analyseur éprouvé, et un ingénieur qui lit un journal peut voir ce que contenait une requête. Les coûts se trouvent ailleurs. Chaque enregistrement répète ses noms de champs, les nombres circulent sous forme de texte décimal qu’il faut réanalyser, et le format lui-même ne précise ni les champs obligatoires ni leur type. Un encodage binaire fondé sur un schéma, comme Protocol Buffers, est plus compact et plus rapide à analyser, et son contrat est vérifié par un compilateur ; en revanche, une personne a besoin d’un outil pour lire un seul message. Pour une API publique ou destinée au navigateur, JSON est un choix raisonnable. Entre services internes où le volume ou la latence domine, le compromis peut s’inverser.

Un schéma apporte généralement le contrat manquant. JSON Schema déclare les propriétés qu’un objet doit posséder, leurs types et l’autorisation éventuelle de propriétés supplémentaires ; OpenAPI s’appuie dessus pour décrire des API HTTP entières. Le schéma est un document séparé, qui ne reste exact que si quelque chose l’oblige à correspondre au code.

Dans le stockage, la question est de savoir si une structure relève d’une colonne JSON ou de véritables colonnes. PostgreSQL propose deux types. `json` conserve une copie exacte du texte d’entrée et le réanalyse chaque fois qu’une

fonction le manipule, tandis que `jsonb` stocke une forme binaire décomposée, légèrement plus lente à écrire, bien plus rapide à interroger et indexable ; sa documentation recommande `jsonb` pour la plupart des applications. Une colonne JSON convient aux données dont la forme varie d'une ligne à l'autre ou appartient au client, comme les préférences enregistrées d'un utilisateur. Un champ que la base doit utiliser pour une jointure, soumettre à des contraintes ou additionner mérite sa propre colonne.

À quoi cela ressemble en pratique

Sur une plateforme, les fonctions PostgreSQL assemblent elles-mêmes chaque réponse complète en JSON, et l'API Go transmet ce JSON de la base de données au navigateur sans le modifier. Le gestionnaire le reçoit sous forme de `json.RawMessage` et le retransmet sans le décoder ni le réencoder ; une petite fonction auxiliaire `QueryJSON` en a fait le comportement standard. Les DTO, les fonctions de correspondance et les structures de réponse d'une API en couches ont disparu. Lorsque la sortie d'une fonction change, la nouvelle forme parvient au client sans retoucher le code du gestionnaire.

Le JSON stocké pose un autre problème : il survit au code qui l'a écrit. Une plateforme conserve les réglages de notification et d'autres préférences sous forme de valeurs enregistrées par l'utilisateur, superposées à un ensemble de valeurs de référence, initialement fusionnées sur un seul niveau de profondeur. Une clé ajoutée ensuite aux valeurs de référence manquait dans toutes les lignes enregistrées avant son existence : le code client lisait donc un champ non défini et plantait, précisément pour les utilisateurs inscrits depuis le plus longtemps. La correction a été une fusion profonde qui traite les valeurs par défaut comme un catalogue : chaque clé du catalogue est présente en sortie, y compris les clés imbriquées, quel que soit le contenu enregistré. Une migration l'a appliquée d'un coup aux lignes existantes, et le frontend a supprimé ses vérifications défensives.

JSON est aussi, tout simplement, la forme sous laquelle les données arrivent. Pour des données historiques d'électricité livrées en JSONL, avec un événement JSON par ligne, il fallait transformer les événements bruts en consommation horaire par zone, et trois variantes ont été construites, en Python, JQ et SQL. Python a été le plus rapide : 34 ms pour une journée de données et 180 ms pour trente jours.

Où les problèmes surviennent

La grammaire est assez petite pour que les analyseurs s'accordent sur ce qui constitue un texte valide. Les difficultés viennent de tout ce qu'elle laisse ouvert.

Un grand identifiant revient avec un chiffre différent. La grammaire ne plafonne pas les nombres, mais la plupart des analyseurs, dont celui de JavaScript, les lisent en virgule flottante sur 64 bits, qui ne représente exactement les entiers que jusqu'à $2^{53} - 1$. La RFC 8259 donne l'intervalle de $-(2^{53} - 1)$ à $2^{53} - 1$ comme celui sur lequel les implémentations s'accordent. Au-delà, les identifiants changent sans avertissement :

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

Envoyez les identifiants de 64 bits sous forme de chaînes, et l'argent sous forme de chaîne ou d'un nombre entier d'unités minimales, jamais en virgule flottante.

Deux services lisent le même objet différemment. La RFC 8259 indique que les noms d'un objet devraient être uniques et laisse le reste ouvert : face à un nom répété, de nombreux analyseurs gardent la dernière valeur, d'autres produisent une erreur et certains renvoient toutes les paires. Quand une passerelle vérifie le premier `role` et que le backend agit sur le second, la vérification n'a rien protégé. Rejetez les noms en double à l'entrée du système. I-JSON, défini dans la RFC 7493, est un profil de JSON qui les interdit, et un contrat peut le citer nommément.

Un fichier modifié à la main ne peut plus être analysé. Les commentaires et les virgules finales sont hors grammaire : la première personne qui annote un fichier de configuration ou laisse une virgule après le dernier élément le casse. Certains outils acceptent un dialecte assoupli, ce qui rend le même fichier valide pour un lecteur et incorrect pour le suivant. Gardez JSON pour les données écrites par les machines, et donnez aux fichiers maintenus à la main un format conçu pour l'édition, comme YAML ou TOML.

Le texte arrive illisible, ou un analyseur refuse ce qu'un autre accepte. La RFC 8259 exige UTF-8 entre systèmes et interdit une marque d'ordre des octets au début. Un producteur qui écrit en Latin-1, ou un éditeur qui ajoute une marque initiale, produit un texte dont le sort dépend de l'analyseur. La grammaire

autorise aussi un échappement comme `"\uDEAD"`, moitié d'une paire de substituts privée de son partenaire, qu'aucune chaîne Unicode valide ne peut contenir et que les bibliothèques traitent différemment. Encodez en UTF-8 à la source et rejetez les séquences invalides à la frontière.

Une petite requête monopolise l'analyseur. Quelques kilooctets de crochets imbriqués peuvent entraîner un analyseur récursif à des milliers de niveaux de profondeur, et un corps trop volumineux est analysé en entier avant qu'un schéma le voie. La RFC 8259 permet à une implémentation de limiter la taille du texte comme la profondeur d'imbrication ; une API acceptant le JSON d'inconnus devrait appliquer les deux limites : plafonnez la taille de la requête avant de la lire, la profondeur pendant l'analyse, puis seulement vérifiez la forme.

Pour aller plus loin

- RFC 8259 : format d'échange de données JSON — la définition actuelle de l'IETF, avec ses remarques sur les nombres, les noms en double et l'encodage.
- ECMA-404 : syntaxe d'échange de données JSON — la norme d'Ecma pour la même grammaire, citée comme normative par la RFC 8259.
- RFC 7493 : format de message I-JSON — un profil plus strict qui exige UTF-8 et interdit les noms en double.
- Spécification de JSON Schema — le vocabulaire pour déclarer les propriétés qu'un document doit posséder.
- PostgreSQL : types JSON — `json` et `jsonb`, et ce que chacun conserve du texte original.
- Go : json.RawMessage — le type qui permet à un programme Go de différer le décodage ou de transmettre du JSON sans modification.

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 ? » 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/