Engineer

JSON

</> Textformat, in dem die meisten Daten übertragen werden

Was es ist

JSON ist eine Klartextnotation für verschachtelte Daten. Sie verwendet geschweifte und eckige Klammern, doppelte Anführungszeichen und Kommas, damit ein Programm in fast jeder Sprache genau das wieder einlesen kann, was ein anderes geschrieben hat. Die Buchstaben stehen für JavaScript Object Notation, denn die Syntax wurde von JavaScripts Schreibweise für literale Werte übernommen. Ein JSON-Dokument gehört aber zu keiner Sprache und benötigt zum Lesen kein JavaScript. Dateien enden gewöhnlich auf `.json` ; über HTTP wird der Inhalt mit dem Medientyp `application/json` gekennzeichnet.

Das gesamte Format passt auf eine Seite. Vier primitive Typen, nämlich Zeichenketten, Zahlen, Wahrheitswerte mit den Literalen `true` und `false` sowie `null` , und zwei strukturierte Typen bilden das Vokabular: ein Objekt als ungeordnete Menge von Name-Wert-Paaren und ein Array als geordnete Liste. Alles lässt sich in alles andere verschachteln. Datumswerte, Binärdaten, Kommentare und Verweise zwischen Dokumentteilen bleiben Konventionen außerhalb des Formats überlassen.

Douglas Crockford registrierte es erstmals im Juli 2006 mit RFC 4627 bei der IETF. Die aktuelle Definition ist RFC 8259 vom Dezember 2017. Sie stimmt in der Grammatik mit Ecmas Standard ECMA-404 überein und ergänzt die Regeln für den Austausch zwischen zwei Systemen. Text, der ein geschlossenes Ökosystem verlässt, muss UTF-8 verwenden. Ein Sender darf ihm keine Byte Order Mark voranstellen.

XML ist älter und ausdrucksstärker, mit Attributen, Namensräumen und einer langen Tradition von Schemasprachen. YAML wurde in Version 1.2 zu einer echten Obermenge von JSON überarbeitet. Es ergänzt Kommentare und eine

sparsamere Zeichensetzung, wie sie für von Hand bearbeitete Dateien gewünscht werden. Binärkodierungen wie Protocol Buffers und CBOR tauschen Lesbarkeit gegen geringere Größe und höhere Geschwindigkeit.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist JSON?“ 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

Wer JSON wählt, wählt eigentlich einen Vertrag zwischen zwei Systemen: die Struktur der übertragenen Daten und manchmal auch die Struktur dessen, was gespeichert wird.

Bei der Übertragung ist es der Standardfall, weil bereits alles damit umgehen kann. Ein Browser parst es ohne Bibliothek, jede verbreitete Sprache hat einen ausgereiften Parser, und ein Entwickler kann im Protokoll lesen, was eine Anfrage enthielt. Die Nachteile liegen anderswo. Jeder Datensatz wiederholt seine Feldnamen, Zahlen werden als Dezimaltext übertragen und müssen zurückgeparst werden, und das Format selbst sagt nichts darüber, welche Felder erforderlich sind oder welchen Typ sie enthalten. Eine binäre, vom Schema ausgehende Kodierung wie Protocol Buffers ist kleiner und schneller zu parsen. Ihr Vertrag wird vom Compiler geprüft, aber schon zum Lesen einer einzigen Nachricht benötigt ein Mensch ein Werkzeug. Für eine öffentliche oder an Browser gerichtete API ist JSON eine vernünftige Wahl. Zwischen internen Diensten, bei denen Datenmenge oder Latenz überwiegen, kann die Abwägung anders ausfallen.

Ein Schema liefert gewöhnlich den fehlenden Vertrag. JSON Schema deklariert, welche Eigenschaften ein Objekt haben muss, welche Typen sie enthalten und ob zusätzliche Eigenschaften erlaubt sind. OpenAPI baut darauf auf, um ganze HTTP-APIs zu beschreiben. Ein Schema ist ein eigenes Dokument und bleibt nur korrekt, solange etwas seine Übereinstimmung mit dem Code erzwingt.

Bei der Speicherung stellt sich die Frage, ob eine Struktur in eine JSON-Spalte oder in einzelne reguläre Spalten gehört. PostgreSQL bietet zwei Typen: `json` erhält eine exakte Kopie des Eingabetexts und parst ihn jedes Mal erneut, wenn eine Funktion darauf zugreift. `jsonb` speichert dagegen eine zerlegte Binärform, die sich etwas langsamer schreiben, wesentlich schneller abfragen und indizieren lässt. Die Dokumentation empfiehlt für die meisten Anwendungen `jsonb`. Eine JSON-Spalte eignet sich für Daten, deren Struktur von Zeile zu Zeile variiert oder dem Client gehört, etwa gespeicherte Nutzereinstellungen. Ein Feld, über das die Datenbank Verknüpfungen herstellen, für das sie Beschränkungen durchsetzen oder dessen Werte sie aufsummieren muss, verdient eine eigene Spalte.

Wie es in der Praxis aussieht

Auf einer Plattform setzen die PostgreSQL-Funktionen jede vollständige Antwort selbst als JSON zusammen. Die Go-API reicht dieses JSON unverändert von der Datenbank an den Browser weiter. Der Handler übernimmt es als `json.RawMessage` und reicht es ohne Dekodierung oder erneute Kodierung durch. Ein kleiner Helfer namens `QueryJSON` machte dies zum Standard. Die DTOs, Mapper und Antwortstrukturen einer geschichteten API entfielen. Ändert sich die Ausgabe einer Funktion, erreicht die neue Struktur den Client ohne Änderung am Handlercode.

Gespeichertes JSON bringt ein anderes Problem mit, weil es den Code überlebt, der es geschrieben hat. Eine Plattform speichert Benachrichtigungseinstellungen und ähnliche Präferenzen als gespeicherte Nutzerwerte, die einen Satz von Standardwerten überlagern. Ursprünglich wurden beide nur eine Ebene tief zusammengeführt. Ein später zu den Standardwerten hinzugefügter Schlüssel fehlte in jeder zuvor gespeicherten Zeile. Der Clientcode las daher ein undefiniertes Feld und stürzte ab, ausgerechnet bei den langjährigsten Nutzern. Die Lösung war eine tiefe Zusammenführung, die Standardwerte als Katalog behandelt: Jeder Schlüssel des Katalogs ist im Ergebnis vorhanden, auch verschachtelte Schlüssel, unabhängig vom Inhalt des gespeicherten Blobs. Eine Migration wandte das sofort auf vorhandene Zeilen an, und das Frontend entfernte seine vorsorglichen Prüfungen.

JSON ist auch schlicht das Format, in dem Daten ankommen. Bei historischen Stromdaten in JSONL, einem JSON-Ereignis je Zeile, bestand die Aufgabe darin, aus Rohereignissen den stündlichen Verbrauch je Zone zu ermitteln. Dafür wurden drei Varianten in Python, JQ und SQL entwickelt. Python war am schnellsten: 34 ms für Daten eines Tages und 180 ms für dreißig Tage.

Wo es schiefgeht

Die Grammatik ist klein genug, dass Parser sich darüber einig sind, wie gültiger Text aussieht. Probleme entstehen durch alles, was sie offenlässt.

Eine große ID kommt mit einer anderen Ziffer zurück. Die Grammatik begrenzt Zahlen nicht nach oben. Die meisten Parser, darunter der von JavaScript, lesen sie aber als 64-Bit-Gleitkommazahlen ein. Diese stellen ganze Zahlen nur bis $2^{53} – 1$ exakt dar. RFC 8259 nennt den Bereich von $–(2^{53} – 1)$ bis $2^{53} – 1$ als denjenigen, über den Implementierungen sich einig sind. Darüber ändern sich Kennungen ohne Warnung:

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

Senden Sie 64-Bit-Kennungen als Zeichenketten und Geldbeträge als Zeichenkette oder ganzzahlige Anzahl der kleinsten Einheit, niemals als Gleitkommazahl.

Zwei Dienste lesen dasselbe Objekt unterschiedlich. RFC 8259 sagt, die Namen in einem Objekt sollten eindeutig sein, und lässt den Rest offen. Bei einem wiederholten Namen behalten viele Parser den letzten Wert, andere melden einen Fehler und manche liefern jedes Paar zurück. Prüft ein Gateway die erste `role`, während das Backend auf die zweite reagiert, hat die Prüfung nichts geschützt. Weisen Sie doppelte Namen beim Eintritt der Daten in das System zurück. Das in RFC 7493 definierte I-JSON ist ein JSON-Profil, das sie verbietet. Ein Vertrag kann dieses Profil ausdrücklich benennen.

Eine von Hand bearbeitete Datei lässt sich nicht mehr parsen. Kommentare und abschließende Kommas gehören nicht zur Grammatik. Wer zuerst eine Konfigurationsdatei kommentiert oder nach dem letzten Eintrag ein Komma stehen lässt, macht sie daher ungültig. Manche Werkzeuge akzeptieren einen weniger strengen Dialekt, wodurch dieselbe Datei für einen Leser gültig ist und für den nächsten nicht. Verwenden Sie JSON für maschinell geschriebene Daten und für von Hand gepflegte Dateien ein zum Bearbeiten entworfenes Format wie YAML oder TOML.

Text kommt verstümmelt an, oder ein Parser lehnt ab, was ein anderer akzeptiert. RFC 8259 verlangt zwischen Systemen UTF-8 und verbietet eine Byte Order Mark am Anfang. Ein Erzeuger, der Latin-1 schreibt, oder ein Editor, der eine solche Markierung voranstellt, erzeugt Text, dessen weitere Behandlung vom Parser abhängt. Die Grammatik erlaubt auch eine Escape-Sequenz wie `"\uDEAD"`, die Hälfte eines Surrogatpaars ohne Partner. Keine gültige Unicode-Zeichenkette kann sie enthalten, und Bibliotheken behandeln sie unterschiedlich. Kodieren Sie schon an der Quelle als UTF-8 und weisen Sie ungültige Sequenzen an der Systemgrenze zurück.

Eine kleine Anfrage blockiert den Parser. Wenige Kilobyte verschachtelter Klammern können einen rekursiven Parser Tausende Ebenen tief treiben. Ein übergroßer Nachrichteninhalt wird vollständig geparst, bevor ein Schema ihn überhaupt sieht. RFC 8259 erlaubt Implementierungen, sowohl Textgröße als auch Verschachtelungstiefe zu begrenzen. Eine API, die JSON von Fremden annimmt, sollte beides tun: die Anfragegröße vor dem Lesen begrenzen, die Tiefe beim Parsen beschränken und erst danach die Struktur prüfen.

Weiterführende Lektüre

- RFC 8259: Das JSON-Datenaustauschformat — Die aktuelle IETF-Definition mit ihren Hinweisen zu Zahlen, doppelten Namen und Kodierung.
- ECMA-404: Die JSON-Datenaustauschsyntax — Ecmas Standard für dieselbe Grammatik, den RFC 8259 normativ referenziert.
- RFC 7493: Das I-JSON-Nachrichtenformat — Ein strengeres Profil, das UTF-8 vorschreibt und doppelte Namen verbietet.
- JSON-Schema-Spezifikation — Das Vokabular zum Deklarieren der erforderlichen Eigenschaften eines Dokuments.
- PostgreSQL: JSON-Typen — `json` im Vergleich zu `jsonb` und welche Teile des Originaltexts beide bewahren.
- Go: json.RawMessage — Der Typ, mit dem ein Go-Programm das Dekodieren aufschiebt oder JSON unverändert weitergibt.

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?“ 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/