Engineer

JSON

</> tekstformatet de fleste data reiser i

Hva det er

JSON er en ren tekstnotasjon for strukturerte data, skrevet med krøllparenteser, hakeparenteser, doble anførselstegn og komma, slik at et program i nesten hvilket som helst språk kan lese tilbake nøyaktig det et annet program skrev. Bokstavene står for JavaScript Object Notation, fordi syntaksen ble hentet fra måten JavaScript skriver litterale verdier på, men et JSON-dokument tilhører ikke noe språk og trenger ingen JavaScript for å bli lest. Filer slutter som regel på `.json`, og over HTTP merkes innholdet med medietypen `application/json`.

Hele formatet får plass på én side. Fire primitive typer (strenger, tall, litteralene `true` og `false`, og `null`) og to strukturerte utgjør vokabularet: et objekt, som er en uordnet mengde par av navn og verdi, og en tabell, array på engelsk, som er en ordnet liste. Alt kan nøstes inne i alt annet. Datoer, binære data, kommentarer og referanser mellom deler av et dokument er overlatt til konvensjoner utenfor formatet.

Douglas Crockford registrerte det første gang hos IETF i RFC 4627, i juli 2006. Den gjeldende definisjonen er RFC 8259 fra desember 2017, som er enig med Ecmas standard ECMA-404 om grammatikken og legger til reglene som betyr noe når to systemer utveksler JSON. Tekst som forlater et lukket økosystem, må være UTF-8, og en avsender skal ikke sette et byte order mark foran den.

XML er eldre og mer uttrykksfullt, med attributter, navnerom og en lang tradisjon av skjemaspråk. YAML ble revidert i versjon 1.2 for å bli en streng overmengde av JSON, med kommentarene og den lettere tegnsettingen folk vil ha i filer de redigerer for hånd. Binære kodinger som Protocol Buffers og CBOR bytter bort lesbarhet mot størrelse og fart.

Bruk begrepet i en beslutning

Bruk «Hva er JSON?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Delen brukerne aldri ser: hvor dataene ligger, hvilke regler som gjelder, og hvordan andre systemer får lov til å snakke med den.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Designet og driftet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tvers av tre databasemotorer

Designet og driftet 6 ETL/ELT-pipelines Ta kontakt →

Hvorfor det er viktig

Å velge JSON er egentlig å velge en kontrakt mellom to systemer: formen på det som går over ledningen, og noen ganger formen på det som havner i lagringen.

På ledningen er det standardvalget fordi alt allerede snakker det. En nettleser parser det uten et bibliotek, hvert utbredt språk har en moden parser, og en utvikler som leser en logg, kan se hva en forespørsel hadde med seg. Kostnadene ligger andre steder. Hver post gjentar feltnavnene sine, tall reiser som desimaltekst som må parses tilbake, og formatet selv sier ingenting om hvilke felt som er påkrevd, eller hvilken type hvert felt har. En binær koding med skjemaet først, som Protocol Buffers, er mindre og raskere å parse, og kontrakten dens kontrolleres av en kompilator, men et menneske trenger et verktøy for å lese én enkelt melding. For et offentlig API eller et som nettleseren snakker med, er JSON det fornuftige valget. Mellom interne tjenester der volum eller latens veier tyngst, kan avveiningen gå den andre veien.

Et skjema leverer som regel den manglende kontrakten. JSON Schema deklarerer hvilke egenskaper et objekt må ha, hvilke typer de har, og om ekstra egenskaper er tillatt, og OpenAPI bygger på det for å beskrive hele HTTP-API-er. Et skjema er et eget dokument, og det forblir riktig bare så lenge noe tvinger det til å stemme med koden.

I lagringen er spørsmålet om en struktur hører hjemme i en JSON-kolonne eller i ekte kolonner. PostgreSQL tilbyr to typer. `json` beholder en eksakt kopi av inndatateksten og parser den på nytt hver gang en funksjon rører den, mens `jsonb` lagrer en nedbrutt binær form som er litt tregere å skrive, mye raskere å spørre mot og kan indekseres; dokumentasjonen anbefaler `jsonb` for de fleste applikasjoner. En JSON-kolonne passer for data der formen varierer fra rad til rad eller tilhører klienten, som en brukers lagrede preferanser. Et felt databasen må koble på, begrense eller summere, fortjener en egen kolonne.

Slik ser det ut i praksis

På én plattform setter PostgreSQL-funksjonene selv sammen hvert komplette svar som JSON, og Go-API-et sender den JSON-en urørt videre fra databasen til nettleseren. Handleren tar den imot som en `json.RawMessage` og sender den videre uten å dekode eller kode den på nytt, og en liten hjelper, `QueryJSON`, gjorde det til standardveien. DTO-ene, mappingkoden og respons-structene i et lagdelt API falt bort. Når det en funksjon returnerer, endrer seg, når den nye formen klienten uten at noen redigerer handlerkoden.

Lagret JSON gir et annet problem, fordi den lever lenger enn koden som skrev den. En plattform lagrer varslingsinnstillinger og lignende preferanser som en brukers lagrede verdier lagt over et sett med standardverdier, opprinnelig slått sammen på ett nivå. En nøkkel som ble lagt til i standardverdiene senere, manglet i hver rad som var lagret før den fantes, så klientkoden leste et udefinert felt og krasjet, og den krasjet for de brukerne som hadde vært med lengst. Løsningen var en dyp sammenslåing som bruker standardverdiene som katalog: hver nøkkel i katalogen finnes i resultatet, nøstede nøkler inkludert, uansett hva den lagrede bloben inneholder. En databasemigrering brukte den på eksisterende rader med en gang, og frontenden kvittet seg med de defensive kontrollene sine.

JSON er også rett og slett slik data kommer inn. For historiske strømdata levert som JSONL, én JSON-hendelse per linje, var oppgaven å gjøre rå hendelser om til forbruk per time og sone, og det ble bygd tre varianter, i Python, JQ og SQL. Python var raskest, med 34 ms for én dags data og 180 ms for tretti dager.

Hvor det går galt

Grammatikken er liten nok til at parserne er enige om hvordan gyldig tekst ser ut. Problemene kommer fra alt grammatikken lar stå åpent.

En stor ID kommer tilbake med ett siffer annerledes. Grammatikken setter ikke noe tak på tall, men de fleste parsere, også den i JavaScript, leser dem inn som 64-biters flyttall, som bare holder heltall eksakt opp til $2^{53} – 1$. RFC 8259 navngir området fra $–(2^{53} – 1)$ til $2^{53} – 1$ som det implementasjonene er enige om. Utenfor det endrer identifikatorer seg uten noen advarsel:

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

```
// 9007199254740992
```

Send 64-biters identifikatorer som strenger, og send pengebeløp som en streng eller som et heltall av den minste enheten, aldri som et flyttall.

To tjenester leser det samme objektet forskjellig. RFC 8259 sier at navnene i et objekt bør være unike, og lar resten stå åpent: får de et gjentatt navn, beholder mange parsere den siste verdien, andre gir en feil, og noen returnerer hvert eneste par. Når en gateway sjekker den første `role` og backenden handler på den andre, beskyttet sjekken ingenting. Avvis dupliserte navn der data kommer inn i systemet. I-JSON, definert i RFC 7493, er en profil av JSON som forbyr dem, og en kontrakt kan vise til den ved navn.

En fil som er redigert for hånd, slutter å kunne parses. Kommentarer og avsluttende komma ligger utenfor grammatikken, så den første som kommenterer en konfigurasjonsfil, eller lar et komma stå etter det siste elementet, ødelegger den. Noen verktøy godtar en løsere dialekt, og da blir den samme filen gyldig for én leser og ødelagt for den neste. Bruk JSON til data som maskiner skriver, og gi filer som vedlikeholdes for hånd, et format laget for redigering, som YAML eller TOML.

Teksten kommer fram forvrengt, eller én parser avviser det en annen godtar. RFC 8259 krever UTF-8 mellom systemer og forbyr et byte order mark i starten. En produsent som skriver Latin-1, eller en editor som setter inn et slikt merke foran, gir tekst der utfallet avhenger av parseren. Grammatikken tillater også en escape som `\uDEAD`, halvparten av et surrogatpar der partneren mangler, noe ingen gyldig Unicode-streng kan inneholde, og bibliotekene behandler det på ulike måter. Kod som UTF-8 ved kilden og avvis ugyldige sekvenser ved grensen.

Én liten forespørsel binder opp parseren. Noen få kilobyte med nøstede hakeparenteser kan drive en rekursiv parser tusenvis av nivåer ned, og en altfor stor body blir parset i sin helhet før noe skjema får se den. RFC 8259 lar en implementasjon begrense både tekststørrelsen og nøstedybden, og et API som tar imot JSON fra fremmede, bør bruke begge: sett et tak på størrelsen på forespørselen før den leses, sett et tak på dybden under parsingen, og sjekk formen først etter det.

Les mer

- RFC 8259: The JSON Data Interchange Format — den gjeldende definisjonen fra IETF, med merknadene om tall, dupliserte navn og koding.
- ECMA-404: The JSON Data Interchange Syntax — Ecmas standard for den samme grammatikken, som RFC 8259 viser til som normativ.
- RFC 7493: The I-JSON Message Format — en strengere profil som krever UTF-8 og forbyr dupliserte navn.
- JSON Schema-spesifikasjonen — vokabularet for å deklarere hvilke egenskaper et dokument må ha.
- PostgreSQL: JSON Types — `json` mot `jsonb`, og hva hver av dem beholder av den opprinnelige teksten.
- Go: json.RawMessage — typen som lar et Go-program utsette dekodingen eller sende JSON urørt videre.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er JSON?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/json/