Engineer

# JSON

</> tekstformatet, de fleste data rejser i

## Hvad det er

JSON er en tekstnotation for strukturerede data. Den skrives med krøllede og firkantede parenteser, dobbelte anførselstegn og kommaer, så et program i stort set et hvilket som helst sprog kan læse præcis det tilbage, som et andet program har skrevet. Forkortelsen står for JavaScript Object Notation, fordi syntaksen er lånt fra JavaScript, men et JSON-dokument tilhører intet bestemt sprog og kræver ikke JavaScript for at blive læst. Filerne ender som regel på `.json`, og over HTTP bliver indholdet mærket med medietypen `application/json`.

Hele formatet kan stå på én side. Ordforrådet består af fire simple typer (strenge, tal, værdierne `true` og `false` samt `null`) og to sammensatte: et objekt, som er en uordnet samling af navne og værdier, og et array, som er en ordnet liste. Alt kan indlejres i alt andet. Datoer, binære data, kommentarer og henvisninger mellem dele af et dokument klares med konventioner uden for formatet.

Douglas Crockford registrerede første gang formatet hos IETF i RFC 4627 i juli 2006. Den gældende definition er RFC 8259 fra december 2017. Den er enig med Ecmas standard ECMA-404 om grammatikken og tilføjer de regler, der tæller, når to systemer udveksler data: tekst, der forlader et lukket økosystem, skal være UTF-8, og afsenderen må ikke sætte et byte order mark foran.

XML er ældre og mere udtryksfuldt med attributter, navnerum og en lang tradition for skemasprog. YAML blev i version 1.2 gjort til et strengt supersæt af JSON og lægger de kommentarer og den lettere tegnsætning oveni, som folk vil have i filer, de redigerer i hånden. Binære formater som Protocol Buffers og CBOR bytter læsbarhed for størrelse og hastighed.

Brug begrebet i en beslutning

Brug »Hvad er JSON?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Den del, brugerne aldrig ser: hvor data ligger, hvilke regler der gælder, og hvordan andre systemer må tale med den.

**Se vores ydelser** Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har designet og drevet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tre databasemotorer

**Har designet og drevet 6 ETL/ELT-pipelines** Kontakt os →

## Hvorfor det betyder noget

Den, der vælger JSON, vælger i virkeligheden en kontrakt mellem to systemer: formen på det, der sendes over nettet, og nogle gange formen på det, der ender i databasen.

Over nettet er JSON standardvalget, fordi alt allerede taler det. En browser læser det uden et bibliotek, alle udbredte sprog har en moden parser, og en udvikler, der læser en log, kan se, hvad en forespørgsel indeholdt. Omkostningerne ligger et andet sted. Hver post gentager sine feltnavne, tal rejser som decimaltekst, der skal parses tilbage, og formatet siger ikke selv noget om, hvilke felter der er obligatoriske, eller hvilken type de har. Et binært, skemabaseret format som Protocol Buffers er mindre og hurtigere at parse, og en compiler tjekker kontrakten, men et menneske skal bruge et værktøj for at læse en enkelt besked. Til et offentligt API eller et, som browseren taler med, er JSON det fornuftige valg. Mellem interne services, hvor volumen eller latens vejer tungest, kan regnestykket falde ud til den anden side.

Et skema leverer typisk den kontrakt, der mangler. JSON Schema beskriver, hvilke egenskaber et objekt skal have, hvilke typer de har, og om ekstra egenskaber er tilladt, og OpenAPI bygger videre på det for at beskrive et helt HTTP-API. Skemaet er et separat dokument og passer kun, så længe noget tvinger det til at følge koden.

I databasen er spørgsmålet, om en struktur hører hjemme i en JSON-kolonne eller i rigtige kolonner. PostgreSQL har to typer. `json` gemmer en nøjagtig kopi af inputteksten og parser den igen, hver gang en funktion rører den, mens `jsonb` gemmer en opdelt binær form, der er lidt langsommere at skrive, langt hurtigere at søge i og kan indekseres. Dokumentationen anbefaler `jsonb` til de fleste applikationer. En JSON-kolonne passer til data, hvis form varierer fra række

til række eller tilhører klienten, for eksempel en brugers gemte indstillinger. Et felt, som databasen skal joine på, håndhæve en begrænsning på eller summere, fortjener sin egen kolonne.

## Sådan ser det ud i praksis

På én platform bygger PostgreSQL-funktionerne selv hele svaret som JSON, og Go-API'et sender den JSON uændret videre fra databasen til browseren. Handleren modtager svaret som `json.RawMessage` og giver det videre uden at afkode eller genkode det, og hjælpefunktionen `QueryJSON` gjorde det til standarden. De DTO'er, mappere og svarstrukturer, et lagdelt API plejer at samle, forsvandt. Ændrer en funktion sit output, når den nye form helt frem til klienten uden en eneste rettelse i handlerkoden.

Gemt JSON giver et andet problem, fordi den lever længere end den kode, der skrev den. En platform gemmer notifikationsindstillinger og lignende præferencer som brugerens egne værdier lagt oven på et sæt standardværdier, oprindeligt flettet i ét niveau. En nøgle, der senere kom til i standardværdierne, manglede i alle rækker gemt før den, så klientkoden læste et felt som undefined og gik ned, og det ramte netop de brugere, der havde været med længst. Løsningen blev en deep merge, der bruger standardværdierne som katalog: hver nøgle i kataloget kommer ud, også de indlejrede, uanset hvad den gemte klump indeholder. En migrering førte det ind i de eksisterende rækker med det samme, og frontenden kunne droppe sine defensive tjek.

Data ankommer også ofte som JSON. Til historiske elektricitetsdata leveret som JSONL, én JSON-hændelse pr. linje, gik opgaven ud på at gøre rå hændelser til timeforbrug pr. zone, og der blev bygget tre varianter, i Python, JQ og SQL. Python var hurtigst med 34 ms for én dags data og 180 ms for 30 dage.

## Hvor det går galt

Grammatikken er så lille, at parsere er enige om, hvad gyldig tekst er. Problemerne opstår i alt det, grammatikken lader stå åbent.

**Et stort id kommer tilbage med et andet ciffer.** Grammatikken sætter intet loft over tal, men de fleste parsere, også JavaScripts egen, læser dem ind som 64-bit flydende kommatal, der kun rummer heltal præcist op til $2^{53} – 1$. RFC 8259 udpeger intervallet fra $–(2^{53} – 1)$ til $2^{53} – 1$ som det, implementeringer er enige om. Over den grænse ændrer id'er sig uden varsel:

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

Send 64-bit-id'er som strenge, og send beløb som strenge eller som et heltal af den mindste møntenhed, aldrig som kommatal.

**To services læser det samme objekt forskelligt.** RFC 8259 siger, at navnene i et objekt bør være unikke, og lader resten stå åbent: møder en parser et gentaget navn, beholder mange den sidste værdi, andre melder fejl, og nogle returnerer alle parrene. Når en gateway tjekker den første `role`, og backenden handler på den anden, har tjekket beskyttet ingenting. Afvis dublerede navne der, hvor data kommer ind i systemet. I-JSON, defineret i RFC 7493, er en profil af JSON, der forbyder dem, og en kontrakt kan henvise til profilen ved navn.

**En håndredigeret fil holder op med at parse.** Kommentarer og afsluttende kommaer ligger uden for grammatikken, så den første, der skriver en kommentar i en konfigurationsfil eller efterlader et komma efter sidste element, ødelægger den. Visse værktøjer accepterer en løsere dialekt, og så er den samme fil gyldig for én læser og ugyldig for den næste. Brug JSON til data, som maskiner skriver, og giv filer, folk vedligeholder i hånden, et format lavet til redigering, for eksempel YAML eller TOML.

**Teksten kommer forvansket frem, eller én parser afviser, hvad en anden accepterer.** RFC 8259 kræver UTF-8 mellem systemer og forbyder et byte order mark i starten. En producent, der skriver Latin-1, eller en editor, der sætter sådan et mærke foran, giver tekst, hvis behandling afhænger af parseren. Grammatikken tillader også en escape-sekvens som `"\uDEAD"`, den ene halvdel af et surrogatpar uden sin partner, som ingen gyldig Unicode-streng kan rumme, og bibliotekerne håndterer den hver på sin måde. Kod som UTF-8 ved kilden, og afvis ugyldige sekvenser ved grænsen.

**En lille forespørgsel binder parseren.** Nogle få kilobyte indlejrede klammer kan sende en rekursiv parser tusindvis af niveauer ned, og en overstor body bliver parset helt, før noget skema ser den. RFC 8259 tillader en implementering at begrænse både tekstens størrelse og indlejringsdybden, og et API, der tager imod JSON fra fremmede, bør bruge begge dele: et loft over størrelsen, før der læses, et loft over dybden under parsningen og først derefter et tjek af formen.

# Læs mere

- RFC 8259: The JSON Data Interchange Format — den gældende IETF-definition med noterne om tal, dublerede navne og tegnkodning.
- ECMA-404: The JSON Data Interchange Syntax — Ecmas standard for samme grammatik, som RFC 8259 henviser til som normativ.
- RFC 7493: The I-JSON Message Format — en strammere profil, der kræver UTF-8 og forbyder dublerede navne.
- JSON Schema-specifikationen — ordforrådet til at erklære, hvilke egenskaber et dokument skal have.
- PostgreSQL: JSON Types — `json` over for `jsonb`, og hvad hver af dem bevarer af den oprindelige tekst.
- Go: json.RawMessage — typen, der lader et Go-program udskyde afkodningen eller sende JSON uændret videre.

Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

**Se, hvad det koster** Kontakt os →

## Gør begreberne praktiske

Spørg, hvordan »Hvad er JSON?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

**Se vores ydelser**

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

Tag denne side med dig

Visitkort

**Lavet med kærlighed af D.**

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

**Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.**

**Udskrevet fra https://engineer.company/da/glossary/json/**