Engineer

JSON

</> textformatet som de flesta data färdas i

Vad det är

JSON är en textnotation för strukturerade data, skriven med klammerparenteser, hakparenteser, dubbla citattecken och kommatecken, så att ett program i nästan vilket språk som helst kan läsa tillbaka exakt det som ett annat program skrev. Bokstäverna står för JavaScript Object Notation, eftersom syntaxen togs från det sätt som JavaScript skriver literala värden på, men ett JSON-dokument tillhör inget språk och behöver ingen JavaScript för att läsas. Filer slutar oftast på `.json`, och över HTTP märks innehållet med medietypen `application/json`.

Hela formatet får plats på en sida. Fyra primitiva typer (strängar, tal, literalerna `true` och `false`, och `null`) och två strukturerade utgör ordförrådet: ett objekt, som är en oordnad mängd par av namn och värde, och en array, som är en ordnad lista. Vad som helst kan nästlas inuti vad som helst annat. Datum, binärdata, kommentarer och referenser mellan delar av ett dokument lämnas alla åt konventioner utanför formatet.

Douglas Crockford registrerade det först hos IETF i RFC 4627, i juli 2006. Den gällande definitionen är RFC 8259 från december 2017, som stämmer överens med Ecmas standard ECMA-404 om grammatiken och lägger till de regler som spelar roll när två system utbyter det. Text som lämnar ett slutet ekosystem måste vara UTF-8, och en avsändare får inte lägga en byteordningsmarkering framför den.

XML är äldre och uttrycksfullare, med attribut, namnrymder och en lång tradition av schemaspråk. YAML reviderades i version 1.2 för att bli en strikt övermängd av JSON och lägger till de kommentarer och den lättare interpunktion som folk vill ha i filer som de redigerar för hand. Binära kodningar som Protocol Buffers och CBOR byter läsbarhet mot storlek och hastighet.

Använd begreppet i ett beslut

Använd ”Vad är JSON?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Den del användarna aldrig ser: där data lagras, vilka reglerna är och hur andra system får kommunicera med den.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Utformade och drev 6 ETL/ELT-pipelines — 6 ETL-pipelines över tre databasmotorer

Utformade och drev 6 ETL/ELT-pipelines Hör av er →

Varför det spelar roll

Att välja JSON är egentligen att välja ett kontrakt mellan två system: formen på det som går över tråden, och ibland formen på det som hamnar i lagringen.

På tråden är det förvalet eftersom allt redan talar det. En webbläsare tolkar det utan bibliotek, varje etablerat språk har en mogen parser, och en ingenjör som läser en logg kan se vad en begäran innehöll. Kostnaderna ligger på andra ställen. Varje post upprepar sina fältnamn, tal färdas som decimaltext som måste tolkas tillbaka, och formatet självt säger ingenting om vilka fält som krävs eller vilken typ vart och ett har. En binär kodning där schemat kommer först, som Protocol Buffers, är mindre och snabbare att tolka, och dess kontrakt kontrolleras av en kompilator, men en människa behöver ett verktyg för att läsa ett enda meddelande. För ett publikt API, eller ett som vänder sig mot webbläsare, är JSON det förnuftiga valet. Mellan interna tjänster där volym eller latens dominerar kan avvägningen gå åt andra hållet.

Ett schema står oftast för det kontrakt som saknas. JSON Schema deklarerar vilka egenskaper ett objekt måste ha, vilka typer de har och om extra egenskaper är tillåtna, och OpenAPI bygger på det för att beskriva hela HTTP-API:er. Ett schema är ett separat dokument, och det förblir korrekt bara så länge något tvingar det att stämma med koden.

I lagringen är frågan om en struktur hör hemma i en JSON-kolumn eller i riktiga kolumner. PostgreSQL erbjuder två typer. `json` behåller en exakt kopia av indatatexten och tolkar om den varje gång en funktion rör den, medan `jsonb` lagrar en uppdelad binär form som är något långsammare att skriva, mycket snabbare att fråga mot och kan indexeras; dokumentationen rekommenderar `jsonb` för de flesta applikationer. En JSON-kolumn passar data vars form

varierar från rad till rad eller tillhör klienten, t.ex. en användares sparade inställningar. Ett fält som databasen måste joina på, begränsa eller summera förtjänar en egen kolumn.

Så ser det ut i praktiken

På en plattform sätter PostgreSQL-funktionerna själva ihop varje komplett svar som JSON, och API:et i Go skickar den JSON-koden orörd vidare från databasen till webbläsaren. Handlern tar emot den som en `json.RawMessage` och lämnar den vidare utan att avkoda eller koda om den, och en liten hjälpfunktion, `QueryJSON`, gjorde det till förvalet. DTO:erna, mappningsfunktionerna och svars-structarna i ett lagerindelat API föll bort. När utdata från en funktion ändras når den nya formen klienten utan någon ändring i handlerkoden.

Lagrad JSON för med sig ett annat problem, eftersom den överlever koden som skrev den. En plattform sparar aviseringsinställningar och liknande som en användares sparade värden lagda ovanpå en uppsättning standardvärden, ursprungligen sammanslagna på en enda nivå. En nyckel som lades till i standardvärdena senare saknades i varje rad som sparats innan den fanns, så klientkoden läste ett odefinierat fält och kraschade, och den kraschade för de användare som hade varit med längst. Lösningen var en djup sammanslagning som behandlar standardvärdena som en katalog: varje nyckel i katalogen finns med i resultatet, nästlade nycklar inräknade, oavsett vad den sparade bloben innehåller. En migrering tillämpade den på befintliga rader direkt, och frontenden kunde släppa sina defensiva kontroller.

JSON är också helt enkelt den form som data anländer i. För historiska eldata som levererades som JSONL, en JSON-händelse per rad, var uppgiften att göra om råa händelser till förbrukning per timme och zon, och tre varianter byggdes, i Python, JQ och SQL. Python var snabbast, med 34 ms för en dags data och 180 ms för trettio dagar.

Var det går fel

Grammatiken är så liten att parsrar är överens om hur giltig text ser ut. Problemen kommer från allt som grammatiken lämnar öppet.

Ett stort id kommer tillbaka med en siffra annorlunda. Grammatiken sätter inget tak för tal, men de flesta parsrar, JavaScripts bland dem, läser in dem som 64-bitars flyttal, som rymmer heltal exakt bara upp till $2^{53} - 1$. RFC 8259 anger

intervallet från $-(2^{53} - 1)$ till $2^{53} - 1$ som det som implementationerna är överens om. Bortom det ändras identifierare utan någon varning:

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

Skicka 64-bitars identifierare som strängar, och skicka pengar som en sträng eller som ett heltal av den minsta enheten, aldrig som ett flyttal.

Två tjänster läser samma objekt olika. RFC 8259 säger att namnen i ett objekt bör vara unika och lämnar resten öppet: vid ett upprepat namn behåller många parsrar det sista värdet, andra signalerar ett fel och några lämnar tillbaka varje par. När en gateway kontrollerar den första `role` och backend agerar på den andra skyddade kontrollen ingenting. Avvisa dubblerade namn där data kommer in i systemet. I-JSON, definierat i RFC 7493, är en profil av JSON som förbjuder dem, och ett kontrakt kan hänvisa till den vid namn.

En handredigerad fil går inte längre att tolka. Kommentarer och avslutande kommatecken ligger utanför grammatiken, så den första som kommenterar en konfigurationsfil, eller lämnar ett komma efter sista posten, förstör den. Vissa verktyg accepterar en friare dialekt, vilket gör samma fil giltig för en läsare och trasig för nästa. Använd JSON för data som maskiner skriver, och ge filer som underhålls för hand ett format gjort för redigering, som YAML eller TOML.

Texten kommer fram förvanskad, eller en parser avvisar det som en annan accepterar. RFC 8259 kräver UTF-8 mellan system och förbjuder en byteordningsmarkering i början. En producent som skriver Latin-1, eller en editor som lägger till en markering först, ger text vars öde beror på parsern. Grammatiken tillåter också en escape-sekvens som `"\uDEAD"`, ena halvan av ett surrogatpar där partnern saknas, som ingen giltig Unicode-sträng kan rymma, och bibliotek hanterar den på olika sätt. Koda som UTF-8 vid källan och avvisa ogiltiga sekvenser vid gränsen.

En liten begäran binder upp parsern. Några kilobyte nästlade hakparenteser kan driva en rekursiv parser tusentals nivåer ned, och en överstor kropp tolkas i sin helhet innan något schema ser den. RFC 8259 låter en implementation begränsa både textens storlek och nästlingsdjupet, och ett API som tar emot

JSON från okända avsändare bör använda båda: begränsa storleken på begäran innan den läses, begränsa djupet under tolkningen och kontrollera formen först efter det.

Läs mer

- RFC 8259: The JSON Data Interchange Format — IETF:s gällande definition, med anmärkningarna om tal, dubblerade namn och teckenkodning.
- ECMA-404: The JSON Data Interchange Syntax — Ecmas standard för samma grammatik, som RFC 8259 hänvisar till som normativ.
- RFC 7493: The I-JSON Message Format — en striktare profil som kräver UTF-8 och förbjuder dubblerade namn.
- JSON Schema specification — ordförrådet för att deklarera vilka egenskaper ett dokument måste ha.
- PostgreSQL: JSON Types — `json` mot `jsonb`, och vad var och en behåller av den ursprungliga texten.
- Go: json.RawMessage — typen som låter ett Go-program skjuta upp avkodningen eller skicka JSON vidare orörd.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är JSON?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/json/