**Engineer**

# JSON Schema

ett språk för att beskriva hur JSON-data ska se ut

## Vad det är

JSON Schema är ett språk för att beskriva hur ett stycke JSON ska se ut, och det är självt skrivet i JSON. En beskrivning i språket, ett schema, ställer upp reglerna: det här värdet är ett objekt, det ska ha de här egenskaperna, ett annat är ett heltal på minst ett, ett tredje en textsträng på högst 200 tecken. Ett program som kallas validator tar ett schema och en instans, alltså de data som ska kontrolleras, och svarar på om instansen följer schemat och, om inte, var och varför. Projektet skriver namnet med stort S, JSON Schema, medan ett enskilt dokument skrivet i språket bara kallas ett schema, med litet s, på svenska lika väl som på engelska.

Specifikationen underhålls av JSON Schema-projektet på json-schema.org. Den gällande versionen är 2020-12 och den förra 2019-09; tidigare versioner var numrerade utkast, som draft-04 och draft-07, och äldre verktyg och artiklar kallar dem fortfarande vid de namnen. Texten består av två delar. Kärndokumentet definierar hur ett schema behandlas, hur ett schema hänvisar till ett annat och medietypen `application/schema+json`, och ett annat dokument, om validering, definierar de nyckelord som kontrollerar värden. De gällande dokumenten för 2020-12 är båda daterade den 16 juni 2022.

Varje nyckelord lägger till ett villkor. `type` anger en av sex primitiva typer, null, boolean, object, array, number eller string, eller `integer`, som matchar varje tal vars decimaldel är noll. För tal finns `minimum`, `maximum` och `multipleOf`; för textsträngar `minLength`, `maxLength` och `pattern`, ett reguljärt uttryck; för arrayer `minItems`, `maxItems` och `uniqueItems`; för objekt `properties`, `required` och `additionalProperties`. Nyckelord som `allOf`, `anyOf`, `oneOf` och `not` kombinerar scheman, och `$ref` pekar på ett schema som är definierat någon annanstans, ofta under `$defs` i samma dokument, så att en form som skrivs en gång kan användas på många ställen. Allt som ett schema inte nämner är tillåtet. Det tomma schemat, `{}`, godtar varje instans.

Ett schema kan också ange vilken version av språket det är skrivet i. Nyckelordet `$schema` innehåller adressen till ett metaschema, ett schema som beskriver giltiga scheman i den versionen, och specifikationssidan påpekar att metaschemat validerar sig självt. För 2020-12 är adressen `https://json-schema.org/draft/2020-12/schema` .

Använd begreppet i ett beslut

Använd ”Vad är JSON Schema?” 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

JSON säger ingenting om betydelse. Varje välformat dokument går att parsa, oavsett om programmet som läser det kan använda det till något. Utan en nedskriven beskrivning har varje program som läser ett dokument sin egen uppfattning om vad som är giltigt, utspridd i villkor i koden, och uppfattningarna glider isär tills en begäran misslyckas i produktion. Ett schema samlar reglerna på ett ställe i en form som både människor och program kan läsa, och det kan sparas, versioneras, granskas och publiceras som vilken annan fil som helst.

Specifikationen beskriver sitt syfte brett: JSON Schema är tänkt för validering, dokumentation, navigering via hyperlänkar och styrning av interaktionen med JSON-data, och valideringsvokabulären ska också ge användargränssnitt som arbetar med datan ledtrådar om den. I praktiken betyder det att en enda beskrivning kan avvisa felaktiga indata i kanten av en tjänst, dokumentera ett API, styra fälten på en sida där folk fyller i uppgifter och kontrollera att den JSON som en språkmodell returnerar har de fält som nästa steg väntar sig.

Det har också blivit den gemensamma grund som andra specifikationer bygger på. OpenAPI 3.1 definierar sitt Schema Object som en övermängd av JSON Schema 2020-12, så innehållet i begäranden och svar i en API-beskrivning beskrivs i JSON Schema med några få tillägg. MCP, protokollet som AI-agenter hittar och anropar verktyg genom, beskriver varje verktygs parametrar med ett `inputSchema` , som är ett JSON Schema, och kan beskriva resultatet med ett

`outputSchema`, som klienter bör validera strukturerade resultat mot. En gemensam beskrivning kan alltså ligga till grund för en validator, ett API-dokument och en agent.

## Så ser det ut i praktiken

Schemat nedan beskriver indata till ett verktyg som bokar ett leveransfönster. Det anger versionen det är skrivet i, räknar upp fyra egenskaper med deras regler, säger vilka tre som måste finnas med och avvisar varje egenskap som det inte räknar upp:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/tools/book-slot.json",
  "title": "Boka ett leveransfönster",
  "type": "object",
  "properties": {
    "order": { "type": "string", "pattern": "^[0-9]{4,8}$" },
    "date": { "type": "string", "format": "date" },
    "pallets": { "type": "integer", "minimum": 1, "maximum": 30 },
    "note": { "type": ["string", "null"], "maxLength": 200 }
  },
  "required": ["order", "date", "pallets"],
  "additionalProperties": false
}
```

En instans som bryter mot reglerna:

```
{ "order": "4471", "date": "27/10/2026", "pallets": 0, "pallet": 2 }
```

Instansen fallerar på två punkter. `pallets` ligger under minimivärdet 1, och `pallet`, ett stavfel för `pallets`, är en egenskap som schemat inte räknar upp. Att `note` saknas är inget fel, eftersom `note` inte står i `required`; hade den skickats som `null` hade det också gått igenom, eftersom typen tillåter null. Datumet är det lärorika. Formatet `date` kräver året först, och `27/10/2026` är inte skrivet så, men en validator som körs med sina standardinställningar godtar det, eftersom specifikationen gör `format` till en annotering som bara kontrolleras när den kontrollen är påslagen. Slå på kontrollen i en validator som kan kontrollera datum, så fallerar samma instans på tre punkter.

Ett verktyg som beskrivs på det sättet är just ämnet för sidan om din programvara som verktyg för AI-agenter här på webbplatsen. Den beskriver hur varje operation som en agent ska ha erbjuds som ett verktyg namngivet efter arbetet, som att boka en tid eller släppa en order, med typade indata, fel som en agent kan agera på och idempotenta anrop, så att ett nytt försök aldrig bokar två gånger. Beskrivningen är ett OpenAPI-dokument som genereras ur koden, och en MCP-server erbjuder samma operationer. Sidans poäng är att en agent litar på ett verktygs beskrivning även när den har blivit inaktuell, och en handskriven beskrivning kan glida isär från implementationen. Därför genereras beskrivningarna från koden och utåt, genom databasen, API:et, beskrivningen och klienten, med en kontroll i varje steg som stoppar bygget när två av dem är oense.

## Var det går fel

**En egenskap som räknas upp men aldrig krävs.** `properties` kontrollerar bara de egenskaper som finns med, så ett objekt som utelämnar dem allihop går igenom. Att en egenskap måste finnas med styrs av ett eget nyckelord, och utelämnas `required` krävs ingenting. Skriv `required` bredvid `properties` varje gång, och ha ett testfall för varje obligatorisk egenskap där just den egenskapen har tagits bort.

**Stavfel som slinker igenom som extra egenskaper.** Utan `additionalProperties` är varje egenskap som schemat inte räknar upp tillåten, så `pallet` går igenom, och den riktiga `pallets` saknas utan att någon märker det. Sätts nyckelordet till `false` stängs objektet, men nyckelordet ser bara `properties` och `patternProperties` i sitt eget schemaobjekt. Ett schema som bygger på ett annat genom `allOf` eller `$ref` och sedan stänger sig självt på det sättet avvisar just de egenskaper som det har ärvt. Version 2020-12 har `unevaluatedProperties` för det fallet, eftersom det också räknar med de egenskaper som har kontrollerats genom `allOf`, `$ref` och de andra nyckelord som tillämpar scheman på plats. Använd `additionalProperties: false` på ett schema som står för sig självt och `unevaluatedProperties: false` på ett som är sammansatt.

`format` **som inte kontrollerar något.** Som standard är `format` en annotering som bara beskriver, inte en valideringsregel som kan avvisa data: specifikationen kräver att en validator som kan kontrollera format håller den kontrollen avslagen om ingen ber om den, och även när den är påslagen får en validator låta vilket format som helst passera okontrollerat. OpenAPI 3.1 följer samma regel och tillägger att hur väl format kontrolleras varierar mellan implementationer. Slå på

formatkontroll i den validator du använder, bevisa det med ett testfall som ska fallera, och där ett format spelar roll men inte kan kontrolleras, lägg till ett `pattern` eller en kontroll i koden.

**Mönster som matchar en del av ett värde.** Ett `pattern` är inte förankrat, så `[0-9]{4}` godtar `order 4471b`, eftersom fyra siffror förekommer någonstans i värdet; specifikationens eget exempel är att `es` matchar `expression`. Börja mönstret med `^` och avsluta det med `$` när hela värdet ska matcha. Mönster följer ECMA-262-dialekten, och specifikationen rekommenderar en liten uppsättning konstruktioner för mönster som ska bete sig likadant överallt, så håll dig till enkla teckenklasser, kvantifierare och ankare.

**Saknad, null och tom behandlade som samma sak.** En egenskap som inte finns med, en egenskap satt till `null` och en tom textsträng är tre olika instanser. `"type": "string"` avvisar `null`, och en egenskap som inte står i `required` får saknas. En klient som skickar `null` för att säga ”inget värde” till ett schema som är skrivet med tanke på att fältet saknas fallerar på varje sådan begäran. Bestäm för varje egenskap vad var och en av de tre betyder, och skriv ner det: `["string", "null"]` när null har en betydelse, utelämnande ur `required` när egenskapen är valfri och `minLength: 1` när en tom textsträng inte är ett värde.

**Inget `$schema`, eller två versioner blandade.** När roten i ett schema saknar `$schema` överlåter specifikationen beteendet åt implementationen, och varje validator använder den version som den har som standard. Versionerna skiljer sig åt där det spelar roll. I 2019-09 beskrev en array som gavs till `items` en tupel; i 2020-12 har den uppgiften flyttats till `prefixItems`, och `items` har tagit över rollen från `additionalItems`. Sätt `$schema` i varje rot, läs versionsanteckningarna när du byter version och validera varje schema mot sitt metaschema i CI.

## Läs mer

- JSON Schema Specification — projektets översikt över den gällande versionen, metascheman och versionsanteckningarna för varje tidigare utkast.
- JSON Schema: A Media Type for Describing JSON Documents — Core 2020-12: hur scheman behandlas, `$schema`, `$ref` och `$defs`, och de nyckelord som kombinerar scheman.
- JSON Schema Validation: A Vocabulary for Structural Validation of JSON — nyckelorden för varje typ av värde, `required` och reglerna för `format`.
- JSON Schema 2020-12 Release Notes — vad som har ändrats sedan 2019-09, med `prefixItems` och `$dynamicRef` sida vid sida med det de ersatte.

- OpenAPI Specification v3.1.1 — Schema Object som en övermängd av JSON Schema 2020-12, och `format` som annotering som standard.
- Model Context Protocol: Tools — hur ett verktyg anger sina parametrar och sitt strukturerade resultat i JSON Schema.

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 Schema?” 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-schema/**