Engineer

JSON Schema

et språk for å beskrive hvordan JSON-data skal se ut

Hva det er

JSON Schema er et språk for å beskrive hvordan et stykke JSON skal se ut, og det er selv skrevet i JSON. En beskrivelse i språket, et skjema, setter opp reglene: denne verdien er et objekt, den skal ha disse egenskapene, denne er et heltall på minst én, den andre en tekststreng på høyst 200 tegn. Et program som kalles en validator, tar et skjema og en instans, altså dataene som skal sjekkes, og svarer på om instansen følger skjemaet, og hvis ikke, hvor og hvorfor. Prosjektet skriver navnet med stor S, JSON Schema, mens et enkelt dokument skrevet i språket på engelsk bare heter et schema, med liten s, og på norsk et skjema.

Spesifikasjonen vedlikeholdes av JSON Schema-prosjektet på json-schema.org. Den gjeldende versjonen er 2020-12 og den forrige 2019-09; tidligere versjoner var nummererte drafts, som draft-04 og draft-07, og eldre verktøy og artikler omtaler dem fortsatt med de navnene. Teksten består av to deler. Kjernedokumentet definerer hvordan et skjema behandles, hvordan ett skjema viser til et annet, og medietypen `application/schema+json`, og et annet dokument, om validering, definerer nøkkelordene som sjekker verdier. De gjeldende dokumentene for 2020-12 er begge datert 16. juni 2022.

Hvert nøkkelord legger til én betingelse. `type` angir en av seks primitive typer, null, boolean, object, array, number eller string, eller `integer`, som matcher ethvert tall der desimaldelen er null. For tall finnes `minimum`, `maximum` og `multipleOf`; for tekststrenger `minLength`, `maxLength` og `pattern`, et regulært uttrykk; for arrayer `minItems`, `maxItems` og `uniqueItems`; for objekter `properties`, `required` og `additionalProperties`. Nøkkelord som `allOf`, `anyOf`, `oneOf` og `not` kombinerer skjemaer, og `$ref` peker på et skjema som er definert et annet sted, ofte under `$defs` i samme dokument, så en form som er skrevet én gang, kan brukes mange steder. Alt et skjema ikke nevner, er tillatt. Det tomme skjemaet, `{}`, godtar enhver instans.

Et skjema kan også angi hvilken versjon av språket det er skrevet i. Nøkkelordet `$schema` inneholder adressen til et metaskjema, et skjema som beskriver gyldige skjemaer i den versjonen, og spesifikasjonssiden påpeker at metaskjemaet validerer seg selv. For 2020-12 er adressen `https://json-schema.org/draft/2020-12/schema` .

Bruk begrepet i en beslutning

Bruk «Hva er JSON Schema?» 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

JSON sier ingenting om betydning. Ethvert velformet dokument kan parses, uansett om programmet som leser det, kan bruke det til noe. Uten en nedskrevet beskrivelse har hvert program som leser et dokument, sin egen oppfatning av hva som er gyldig, spredt utover betingelser i koden, og oppfatningene glir fra hverandre til en forespørsel feiler i produksjon. Et skjema samler reglene på ett sted i en form som både mennesker og programmer kan lese, og det kan lagres, versjoneres, gjennomgås og publiseres som enhver annen fil.

Spesifikasjonen beskriver formålet sitt bredt: JSON Schema er ment for validering, dokumentasjon, navigering via hyperlenker og styring av samspillet med JSON-data, og valideringsvokabularet skal også gi brukergrensesnitt som arbeider med dataene, hint om dem. I praksis betyr det at én beskrivelse kan avvise feil inndata i kanten av en tjeneste, dokumentere et API, styre feltene på en side der folk fyller inn data, og sjekke at JSON-en en språkmodell returnerer, har feltene neste steg forventer.

Det har også blitt det felles grunnlaget som andre spesifikasjoner bygger på. OpenAPI 3.1 definerer sitt Schema Object som et supersett av JSON Schema 2020-12, så innholdet i forespørsler og svar i en API-beskrivelse er beskrevet i JSON Schema med noen få tillegg. MCP, protokollen AI-agenter finner og kaller verktøy gjennom, beskriver hvert verktøys parametere med et `inputSchema` , som

er et JSON Schema, og kan beskrive resultatet med et `outputSchema`, som klienter bør validere strukturerte resultater mot. En felles beskrivelse kan altså danne grunnlag for en validator, et API-dokument og en agent.

Slik ser det ut i praksis

Skjemaet nedenfor beskriver inndataene til et verktøy som bestiller et leveringsvindu. Det angir versjonen det er skrevet i, lister opp fire egenskaper med reglene deres, sier hvilke tre som må være med, og avviser enhver egenskap det ikke lister opp:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/tools/book-slot.json",
  "title": "Bestill et leveringsvindu",
  "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 reglene:

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

Instansen feiler på to punkter. `pallets` ligger under minimumet på 1, og `pallet`, en skrivefeil for `pallets`, er en egenskap skjemaet ikke lister opp. At `note` mangler, er ingen feil, for `note` står ikke i `required`; hadde den blitt sendt som `null`, ville det også gått gjennom, siden typen tillater null. Datoen er den lærerike. Formatet `date` krever året først, og `27/10/2026` er ikke skrevet slik, men en validator som kjører med standardinnstillingene sine, godtar den, fordi spesifikasjonen gjør `format` til en annotasjon som bare blir sjekket når den sjekken er slått på. Slå på sjekken i en validator som kan sjekke datoer, og den samme instansen feiler på tre punkter.

Et verktøy beskrevet på den måten er nettopp temaet for siden om programvaren din som verktøy for AI-agenter her på nettstedet. Den beskriver hvordan hver operasjon en agent skal ha tilgang til, tilbys som et verktøy med navn etter arbeidet det gjør, som å bestille en time eller frigi en ordre, med typede inndata, feilmeldinger en agent kan handle ut fra, og idempotente kall, så et nytt forsøk aldri fører til dobbeltbooking. Beskrivelsen er et OpenAPI-dokument generert fra koden, og en MCP-server tilbyr de samme operasjonene. Sidens poeng er at en agent stoler på en verktøybeskrivelse også når den er utdatert, og en håndskrevet beskrivelse kan avvike fra implementasjonen. Derfor genereres beskrivelsene fra koden og utover, gjennom databasen, API-et, beskrivelsen og klienten, med en sjekk i hvert steg som stopper bygget når to av dem ikke stemmer overens.

Hvor det går galt

En egenskap som er listet opp, men aldri påkrevd. `properties` sjekker bare de egenskapene som er til stede, så et objekt som utelater alle sammen, går gjennom. At en egenskap må være med, styres av et eget nøkkelord, og utelates `required`, er ingenting påkrevd. Skriv `required` ved siden av `properties` hver gang, og ha en testsak for hver påkrevde egenskap der nettopp den egenskapen er fjernet.

Skrivefeil som slipper gjennom som ekstra egenskaper. Uten `additionalProperties` er enhver egenskap skjemaet ikke lister opp, tillatt, så `pallet` går gjennom, og den riktige `pallets` mangler uten at noen merker det. Settes nøkkelordet til `false`, lukkes objektet, men nøkkelordet ser bare `properties` og `patternProperties` i sitt eget skjemaobjekt. Et skjema som bygger på et annet gjennom `allOf` eller `$ref` og deretter lukker seg selv på den måten, avviser nettopp de egenskapene det har arvet. Versjon 2020-12 har `unevaluatedProperties` for det tilfellet, siden det også regner med egenskapene som er sjekket gjennom `allOf`, `$ref` og de andre nøkkelordene som anvender skjemaer på stedet. Bruk `additionalProperties: false` på et skjema som står alene, og `unevaluatedProperties: false` på et som er satt sammen.

`format` **som ikke sjekker noe.** Som standard er `format` en annotasjon som bare beskriver, ikke en valideringsregel som kan avvise data: spesifikasjonen krever at en validator som kan sjekke formater, holder den sjekken slått av med mindre noen ber om den, og selv når den er slått på, kan en validator la et hvilket som helst format passere usjekket. OpenAPI 3.1 følger den samme regelen og legger til at hvor godt formater blir sjekket, varierer mellom implementasjoner. Slå på

formatsjekk i validatoren du bruker, bevis det med en testsak som skal feile, og der et format betyr noe, men ikke kan sjekkes, legg til et `pattern` eller en sjekk i koden.

Mønstre som matcher en del av en verdi. Et `pattern` er ikke forankret, så `[0-9]{4}` godtar `order 4471b`, fordi fire sifre forekommer et sted i verdien; spesifikasjonens eget eksempel er at `es` matcher `expression`. Begynn mønsteret med `^` og avslutt det med `$` når hele verdien skal matche. Mønstre følger ECMA-262-dialekten, og spesifikasjonen anbefaler et lite sett konstruksjoner for mønstre som skal oppføre seg likt overalt, så hold deg til enkle tegnklasser, kvantorer og ankere.

Manglende, null og tom behandlet som det samme. En egenskap som ikke er med, en egenskap satt til `null` og en tom tekststreng er tre forskjellige instanser. `"type": "string"` avviser `null`, og en egenskap som ikke står i `required`, kan mangle. En klient som sender `null` for å si «ingen verdi» til et skjema som er skrevet med tanke på at feltet mangler, feiler på hver eneste slik forespørsel. Bestem for hver egenskap hva hver av de tre betyr, og skriv det ned: `["string", "null"]` når null har en betydning, utelatelse fra `required` når egenskapen er valgfri, og `minLength: 1` når en tom tekststreng ikke er en verdi.

Ingen `$schema`, eller to versjoner blandet. Når roten av et skjema ikke har noe `$schema`, overlater spesifikasjonen oppførselen til implementasjonen, og hver validator bruker den versjonen den har som standard. Versjonene skiller seg fra hverandre der det betyr noe. I 2019-09 beskrev en array gitt til `items` en tuppel; i 2020-12 er den oppgaven flyttet til `prefixItems`, og `items` har overtatt rollen til `additionalItems`. Sett `$schema` i hver rot, les utgivelsesnotatene når du bytter versjon, og valider hvert skjema mot metaskjemaet sitt i CI.

Les mer

- JSON Schema Specification — prosjektets oversikt over den gjeldende versjonen, metaskjemaene og utgivelsesnotatene for hvert tidligere draft.
- JSON Schema: A Media Type for Describing JSON Documents — Core 2020-12: hvordan skjemaer behandles, `$schema`, `$ref` og `$defs`, og nøkkelordene som kombinerer skjemaer.
- JSON Schema Validation: A Vocabulary for Structural Validation of JSON — nøkkelordene for hver type verdi, `required` og reglene for `format`.
- JSON Schema 2020-12 Release Notes — hva som er endret siden 2019-09, med `prefixItems` og `$dynamicRef` side om side med det de erstattet.

- OpenAPI Specification v3.1.1 — Schema Object som et supersett av JSON Schema 2020-12, og `format` som annotasjon som standard.
- Model Context Protocol: Tools — hvordan et verktøy oppgir parameterne og det strukturerte resultatet sitt i JSON Schema.

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 Schema?» 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-schema/