Engineer

OpenAPI

en maskinlesbar beskrivelse av et HTTP-API

Hva det er

OpenAPI er et standardformat for å skrive ned alt den som kaller et HTTP-API, trenger å vite – adresser, operasjoner, inndata, utdata, feil og sikkerhet – i én JSON- eller YAML-fil som både mennesker og programmer kan lese. Selve standarden heter OpenAPI Specification, ofte forkortet OAS, og en fil skrevet etter den kalles en OpenAPI-beskrivelse. Formatet startet som Swagger Specification, og det eldre navnet lever videre i verktøy bygd rundt det, f.eks. Swagger UI.

Spesifikasjonen beskriver seg selv som en grensesnittbeskrivelse for HTTP-API-er som er uavhengig av programmeringsspråk. Målet den oppgir, er at en person eller et program skal kunne forstå hva en tjeneste gjør, uten kildekoden, uten ekstra dokumentasjon og uten å fange opp nettverkstrafikken. Inngangsdokumentet anbefales navngitt `openapi.json` eller `openapi.yaml`, og større beskrivelser deles ofte opp i flere filer som henger sammen gjennom referanser.

Swagger 2.0 kom i september 2014 og ble gitt til OpenAPI Initiative 31. desember 2015. Versjon 3.0.0 fulgte i juli 2017 under det nye navnet, 3.1.0 i februar 2021 og 3.2.0 i september 2025.

En beskrivelse har en håndfull deler på øverste nivå. `info` navngir API-et og versjonen; `servers` lister hvor det kjører; `paths` inneholder hver adresse og, under den, én operasjon per HTTP-metode, med parametere, forespørselskropp og mulige svar. `components` samler det som gjenbrukes: dataskjemaer, parametere, svar og sikkerhetsordningene (security schemes) som sier hvordan den som kaller, autentiserer seg. Fra og med 3.1 kan en egen `webhooks` -del også beskrive forespørsler API-et sender til sine brukere. Et minimalt utdrag:

```
openapi: 3.1.0
info:
    title: Orders
    version: 1.4.0
paths:
    /orders/{id}:
        get:
            operationId: getOrder
            parameters:
                - name: id
                  in: path
                  required: true
                  schema:
                      type: string
            responses:
                "200":
                    description: One order
```

Dataformene skrives i et skjemaspråk. Fra 3.1 er Schema Object en overmengde av JSON Schema Draft 2020-12; 3.0 brukte en utvidet delmengde av et eldre utkast, med egne tillegg. JSON Schema beskriver ett enkelt dataelement, mens OpenAPI beskriver et helt HTTP-API og låner JSON Schema til bitene. Naboene beskriver andre typer grensesnitt: AsyncAPI dekker meldingsdrevne systemer, gRPC-tjenester deklareres i `.proto`-filer for Protocol Buffers, og en GraphQL-server publiserer sitt eget skjema.

Bruk begrepet i en beslutning

Bruk «Hva er OpenAPI?» 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

En beskrivelse i et standardformat er én fil som mange verktøy kan lese, og hvert av dem sparer et team for arbeid det ellers ville gjort for hånd. Referansedokumentasjonen gjengis fra den. Klientbiblioteker i flere språk genereres fra den, så en frontend kaller en typet funksjon i stedet for å sette sammen en URL. Mock-servere svarer ut fra den, så frontenden kan bygges før

backenden finnes. API-gatewayer importerer ruter fra den, testsuiter validerer trafikk mot den, og AI-agenter som kaller API-er, leser den for å finne ut hva som tilbys.

Den rekkevidden er også risikoen. Hvert eneste av disse verktøyene stoler på filen. En beskrivelse som er feil, er verre enn ingen, fordi den genererte klienten kompilerer, dokumentasjonen ser autoritativ ut, og feilen dukker opp under kjøring i noen andres produkt.

Den viktigste beslutningen er derfor hvor beskrivelsen kommer fra. Ved design-first skriver folk beskrivelsen, gjennomgår den som kontrakten og bygger deretter serveren slik at den stemmer. Formen på API-et diskuteres før det finnes kode, og det passer for offentlige API-er og for team som jobber parallelt. Prisen er at beskrivelsen og serveren er to ting som kan være uenige. Ved code-first genereres beskrivelsen fra serverens egne typer og annotasjoner. Den kan ikke beskrive en rute serveren ikke har, men den er aldri mer presis enn de typene, og designdiskusjonen skjer i kodegjennomgangen i stedet for i et dokument. Uansett holder beskrivelsen seg sann bare hvis et bygg kontrollerer den.

Overgangen fra 3.0 til 3.1 endret hvordan dataskjemaer skrives, fordi 3.1 tok i bruk JSON Schema 2020-12. Generatorer, validatorer og gatewayer støtter hver sitt utvalg av versjoner, så versjonen et team skriver i, begrenses av det eldste verktøyet i kjeden.

Slik ser det ut i praksis

På én plattform startet API-et på Fiber, med validering skrevet for hånd, endepunkt for endepunkt, og uten noe samlet dokument som beskrev formen. Så ble HTTP-laget flyttet over på Huma v2, oppå Fiber, slik at hvert endepunkt deklareres med struct-typer for inndata og utdata, og OpenAPI-beskrivelsen genereres fra de typene. Dokumentasjonen følger koden i stedet for å forfalle i en wiki. De to WebSocket-endepunktene ble igjen på ren Fiber, utenfor forespørsel/svar-modellen til Huma. Siden forklarer hvorfor størrelsen betyr noe: med 760 operasjoner fordelt på 649 stier er beskrivelsen i praksis den eneste måten noen kan lese API-et på, så det å vite at den er fullstendig, betyr mer enn det gjorde ved femti. En kontroll i CI holder samsvaret mellom rutene og beskrivelsen på 100 %.

På en annen plattform er beskrivelsen det midterste leddet i en kjede som går fra databasen og utover. Databaseskjemaet og funksjonene i det former Go-typene, Huma lager beskrivelsen, og det frontenden bruker, genereres fra den: en typet TypeScript-klient på 44 076 linjer, 61 mock-handlere for frontendens tester og grensene hvert webskjema håndhever. Den løser avvik mellom frontend og backend, som ellers gjerne viser seg som en 422 i nettleseren etter at avviket er levert. En kontraktskontroll i CI får bygget til å feile ved avvik, med en probe rettet mot valgfrie felt i forespørselskroppen, der «mangler» og «null» blandes sammen, og mot enumer i spørringsparametere. Siden sier rett ut hva det koster: et genereringstrinn midt i alt sammen, og en kjede som bare er så pålitelig som sitt dårligst bevoktede ledd.

Selskapets side om å være klar for AI-agenter bruker det samme resonnementet på agenter. En app som agenter kan bruke, tilbyr et beskrevet API: et OpenAPI-dokument, en MCP-server der agenter bør kalle det direkte, og OAuth-metadata, så en agent kan handle på vegne av en bruker uten å få passordet. Siden klassifiserer hver anbefaling i stedet for å følge den blindt, og fordi selskapets eget nettsted ikke har noe API, er en API-katalog og OAuth-metadata to av de fem den takket nei til der.

Hvor det går galt

Beskrivelsen lover noe serveren ikke gjør. En beskrivelse som vedlikeholdes for hånd, sakker akterut: en ny spørringsparameter, et felt med nytt navn eller en feilkode som bare serveren kjenner til. Genererte klienter kompilerer da mot en fiksjon. Generer beskrivelsen fra koden, eller hold den adskilt og valider ekte forespørsler og svar mot den i testsuiten, og sammenlign rutene serveren registrerer, med stiene beskrivelsen lister opp.

Hver ny generering av klienten gir halvparten av metodene nye navn. Uten en `operationId` finner generatorene på metodenavn ut fra stien og HTTP-metoden, så når en rute flyttes, får funksjonen alle kallerne bruker, nytt navn. Spesifikasjonen krever at hver `operationId` er unik, og skiller mellom store og små bokstaver. Sett en på hver operasjon, velg den like nøye som navnet på en offentlig funksjon, og la en linter stoppe bygget når den mangler.

Felt som kan være null, begynner å avvise null etter en oppgradering. OpenAPI 3.0 merket dem med `nullable: true`. I 3.1 er det nøkkelordet borte, og typen blir en liste, og en validator som møter et nøkkelord den ikke kjenner, ignorerer det som regel. Et 3.0-skjema som bare merkes om til 3.1, avviser derfor

null overalt der forfatteren ville tillate det. Konverter dataskjemaene med oppgraderingsveiledningen fra OpenAPI Initiative, som også dekker at `exclusiveMinimum` blir et tall:

```
# OpenAPI 3.0
type: string
nullable: true
---
# OpenAPI 3.1
type: [string, "null"]
```

Eksemplene består ikke sitt eget skjema. Spesifikasjonen sier at et eksempel bør stemme med skjemaet sitt, og overlater kontrollen til verktøyene. En leser kopierer eksempelet fra dokumentasjonen og får en 422; en mock-server serverer det, og frontenden bygges mot data det ekte API-et kommer til å avvise. Valider hvert eksempel i CI, like strengt som ekte trafikk.

Den offentlige beskrivelsen kartlegger admin-API-et. Én beskrivelse generert fra alle ruter og servert på en offentlig adresse viser interne operasjoner og dataskjemaene deres til alle som spør. Publiser én beskrivelse per målgruppe, generert med de interne operasjonene filtrert bort. Spesifikasjonen tillater at objekter står tomme eller fjernes nettopp for denne typen tilgangskontroll.

Les mer

- OpenAPI Specification — siste versjon av standarden, med hvert objekt og felt definert og historikken for hver utgave.
- JSON Schema Draft 2020-12 — skjemaspråket OpenAPI 3.1 og senere bruker for hver forespørselskropp og hvert svar.
- Upgrading from OpenAPI 3.0 to 3.1 — OpenAPI Initiatives egen liste over endringer i dataskjemaene, `nullable` blant dem.
- Huma: OpenAPI Generation — hvordan Go-rammeverket fra prosjektene over avleder en beskrivelse fra typene til handlerne.

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 OpenAPI?» 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/openapi/