Engineer

OpenAPI

en maskinlæsbar beskrivelse af et HTTP-API

Hvad det er

OpenAPI er et standardformat til at skrive alt det ned, en kalder skal vide om et HTTP-API — adresser, operationer, input, output, fejl og sikkerhed — i én JSON- eller YAML-fil, som mennesker og programmer læser på lige fod. Selve standarden hedder OpenAPI Specification, ofte forkortet OAS, og en fil skrevet efter den er en OpenAPI-beskrivelse. Formatet begyndte som Swagger Specification, og det ældre navn lever videre i værktøjer bygget omkring det, som Swagger UI.

Specifikationen kalder sig selv en programmeringssprogsuafhængig grænsefladebeskrivelse for HTTP-API'er. Målet, den sætter sig, er, at et menneske eller et program kan forstå, hvad en tjeneste gør, uden kildekoden, ekstra dokumentation eller en optagelse af dens netværkstrafik. Hoveddokumentet anbefales navngivet `openapi.json` eller `openapi.yaml`, og større beskrivelser deles ofte over flere filer, der er bundet sammen med referencer.

Historien går gennem én overdragelse. Swagger 2.0 udkom i september 2014 og blev doneret til OpenAPI Initiative den 31. december 2015. Version 3.0.0 fulgte i juli 2017 under det nye navn, 3.1.0 i februar 2021 og 3.2.0 i september 2025.

En beskrivelse har en håndfuld dele på øverste niveau. `info` navngiver API'et og dets version; `servers` angiver, hvor det kører; `paths` rummer hver adresse og under den én operation pr. HTTP-metode med dens parametre, request body og mulige svar. `components` samler det, der genbruges: dataskemaer, parametre, svar og de sikkerhedsskemaer, der fortæller, hvordan en kalder autentificerer sig. Fra 3.1 kan en `webhooks` -sektion også beskrive de forespørgsler, API'et selv sender til sine brugere. Et minimalt udsnit:

```
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
```

Dataformerne skrives i et skemasprog. Fra 3.1 er Schema Object et supersæt af JSON Schema Draft 2020-12; 3.0 brugte en udvidet delmængde af et ældre udkast med egne tilføjelser. JSON Schema beskriver ét stykke data, mens OpenAPI beskriver et helt HTTP-API og låner JSON Schema til stykkerne. Naboerne beskriver andre slags grænseflader: AsyncAPI dækker beskedbaserede systemer, gRPC-tjenester erklæres i Protocol Buffers-filer med endelsen `.proto`, og en GraphQL-server udgiver sit eget skema.

Brug begrebet i en beslutning

Brug »Hvad er OpenAPI?« 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

En beskrivelse i et standardformat er én fil, som mange værktøjer kan læse, og hvert af dem sparer et stykke arbejde, et team ellers ville gøre i hånden. Referencedokumentationen renderes ud fra den. Klientbiblioteker i flere sprog genereres ud fra den, så en frontend kalder en typet funktion i stedet for at stykke en URL sammen. Mockservere svarer ud fra den, så en frontend kan

bygges, før backenden findes. Gateways importerer ruter fra den, testsuiter validerer trafik mod den, og AI-agenter, der kalder API'er, læser den for at se, hvad der bliver tilbudt.

Den rækkevidde er også risikoen. Hvert eneste af de værktøjer stoler på filen. En forkert beskrivelse er værre end ingen, for den genererede klient kompilerer, dokumentationen ser autoritativ ud, og fejlen viser sig først i drift, i en andens produkt.

Hovedbeslutningen er derfor, hvor beskrivelsen kommer fra. I design-first-arbejde skriver folk beskrivelsen, reviewer den som kontrakten og bygger derefter serveren, så den passer. API'ets form bliver diskuteret, før der findes kode, hvilket passer til offentlige API'er og til teams, der arbejder parallelt. Prisen er, at beskrivelsen og serveren er to ting, der kan blive uenige. I code-first-arbejde genereres beskrivelsen ud fra serverens egne typer og annotationer. Den kan ikke beskrive en rute, serveren mangler, men den er kun så præcis som de typer, og designdiskussionen foregår i code review i stedet for i et dokument. Uanset vejen forbliver beskrivelsen kun sand, hvis et build tjekker den.

Versioner betyder mere, end de ser ud til. Skiftet fra 3.0 til 3.1 ændrede, hvordan skemaobjekter skrives, fordi 3.1 tog JSON Schema 2020-12 til sig. Generatorer, validatorer og gateways understøtter hver deres udsnit af versioner, så den version, et team skriver i, er begrænset af det ældste værktøj i kæden.

Sådan ser det ud i praksis

Et API på Fiber begyndte med validering skrevet i hånden, endpoint for endpoint, og uden et samlet dokument over dets form. HTTP-laget flyttede til Huma v2, oven på Fiber, så hvert endpoint erklæres med input- og output-structs, og OpenAPI-beskrivelsen genereres ud fra de typer; dokumentationen holder trit med koden i stedet for at forældes i en wiki. De to WebSocket-endpoints bliver på rå Fiber, uden for Humas model med forespørgsel og svar. Siden siger ligeud, hvorfor skalaen tæller: ved 760 operationer fordelt på 649 paths er beskrivelsen i praksis det eneste sted, nogen kan læse API'et, så vished om, at den er komplet, vejer tungere, end den gjorde ved halvtreds. Et tjek i CI holder pariteten mellem ruter og beskrivelse på 100 %.

Beskrivelsen kan også være det midterste led i en kæde, der løber fra databasen og udad. Skemaet og dets funktioner former Go-typerne, Huma udsender beskrivelsen, og det, frontenden bruger, genereres ud fra den: en typet

TypeScript-klient på 44.076 linjer, 61 mock-handlere til frontendens tests og de grænser, hver formular håndhæver. Det modvirker, at frontend og backend glider fra hinanden, som typisk viser sig som et 422 i browseren, efter at uoverensstemmelsen er sendt ud. Et kontrakttjek i CI får buildet til at fejle ved afvigelse og kigger særligt efter i valgfrie felter i request body, hvor »mangler« og »null« bliver forvekslet, og i enums for query-parametre. Siden nævner også prisen: et genereringstrin midt i det hele og en kæde, der kun er så pålidelig som sit svagest bevogtede led.

Virksomhedens side om parathed til AI-agenter bruger samme tankegang over for agenter. En applikation, som agenter kan bruge, tilbyder et beskrevet API: et OpenAPI-dokument, en MCP-server, hvor agenter skal kalde applikationen direkte, og OAuth-metadata, så en agent kan handle på vegne af en bruger, uden at nogen giver sin adgangskode fra sig. Siden klassificerer hver anbefaling til agenter i stedet for blot at følge den, og fordi virksomhedens eget site ikke har noget API, er et API-katalog og OAuth-metadata to af de fem, der blev valgt fra dér.

Hvor det går galt

Beskrivelsen lover noget, serveren ikke gør. En beskrivelse, der vedligeholdes i hånden, sakker bagud: en ny query-parameter, et omdøbt felt eller en fejlkode, som kun serveren kender til. Genererede klienter kompilerer så mod en fiktion. Generér beskrivelsen ud fra koden, eller hold den adskilt og validér rigtige forespørgsler og svar mod den i testsuiten, og sammenlign de ruter, serveren registrerer, med de paths, beskrivelsen opregner.

Hver regenerering af klienten omdøber halvdelen af metoderne. Uden et `operationId` finder generatorerne selv på metodenavne ud fra stien og HTTP-metoden, så når en rute flytter, skifter den funktion, som alle kaldere bruger, navn. Specifikationen kræver, at hvert `operationId` er unikt, og skelner mellem store og små bogstaver i det. Sæt et på hver operation, vælg det lige så omhyggeligt som navnet på en offentlig funktion, og lad en linter få buildet til at fejle, når et mangler.

Felter, der må være null, begynder at afvise null efter en opgradering. OpenAPI 3.0 markerede dem med `nullable: true`. I 3.1 er det nøgleord væk, og typen bliver en liste, og en validator, der møder et nøgleord, den ikke kender, springer det som regel over. Et 3.0-skema, der blot får påskriften 3.1, afviser

derfor null alle de steder, forfatteren ville tillade det. Konvertér skemaerne efter OpenAPI Initiatives opgraderingsguide, som også dækker, at `exclusiveMinimum` bliver et tal:

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

Eksemplerne består ikke deres eget skema. Specifikationen siger, at et eksempel bør matche sit skema, og overlader kontrollen til værktøjerne. En læser kopierer eksemplet fra dokumentationen og får et 422; en mockserver serverer det, og frontenden bliver bygget mod data, som det rigtige API vil afvise. Validér hvert eksempel i CI, lige så strengt som rigtig trafik.

Den offentlige beskrivelse kortlægger admin-API'et. Én beskrivelse genereret ud fra alle ruter og serveret på en offentlig adresse viser interne operationer og deres skemaobjekter til enhver, der spørger. Udgiv én beskrivelse pr. publikum, genereret med de interne operationer filtreret fra. Specifikationen tillader, at objekter står tomme eller fjernes netop til den slags adgangskontrol.

Læs mere

- OpenAPI Specification — den seneste version af standarden med hvert objekt og felt defineret og en historik over hver udgivelse.
- JSON Schema Draft 2020-12 — det skemasprog, OpenAPI 3.1 og senere bruger til hver request- og response-body.
- Upgrading from OpenAPI 3.0 to 3.1 — OpenAPI Initiatives egen liste over skemaændringer, med `nullable` iblandt.
- Huma: OpenAPI Generation — hvordan Go-frameworket fra projekterne ovenfor udleder en beskrivelse af handlernes typer.

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 OpenAPI?« 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/openapi/