Engineer

OpenAPI

en maskinläsbar beskrivning av ett HTTP-API

Vad det är

OpenAPI är ett standardformat för att skriva ner allt som en anropare behöver veta om ett HTTP-API — adresser, operationer, indata, utdata, fel och säkerhetskrav — i ett enda JSON- eller YAML-dokument som människor och program läser på samma sätt. Själva standarden heter OpenAPI Specification, ofta förkortad OAS, och ett dokument skrivet enligt den kallas en OpenAPI-beskrivning. Formatet började som Swagger Specification, och det äldre namnet lever kvar i verktyg som byggts kring det, till exempel Swagger UI.

Specifikationen beskriver sig själv som en gränssnittsbeskrivning för HTTP-API:er som är oberoende av programspråk. Målet den anger är att en människa eller ett program ska kunna förstå vad en tjänst gör utan dess källkod, ytterligare dokumentation eller en inspelning av dess nätverkstrafik. Rotdokumentet rekommenderas heta `openapi.json` eller `openapi.yaml`, och större beskrivningar delas ofta upp i flera dokument som fogas samman med referenser.

Swagger 2.0 släpptes i september 2014 och skänktes till OpenAPI Initiative den 31 december 2015. Version 3.0.0 kom i juli 2017 under det nya namnet, följd av 3.1.0 i februari 2021 och 3.2.0 i september 2025.

En beskrivning har en handfull delar på toppnivå. `info` namnger API:et och dess version; `servers` listar var det körs; `paths` innehåller varje adress och, under den, en operation per HTTP-metod, med parametrar, anropets body och möjliga svar. `components` samlar det som återanvänds: datascheman, parametrar, svar och de säkerhetsscheman som anger hur en anropare autentiserar sig. Sedan 3.1 kan en sektion `webhooks` dessutom beskriva anrop som API:et skickar till sina konsumenter. Ett 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
```

Datastrukturerna skrivs i ett schemaspråk. Från och med 3.1 är Schema Object en övermängd av JSON Schema Draft 2020-12; 3.0 använde en utökad delmängd av ett äldre utkast med egna tillägg. JSON Schema beskriver ett enskilt dataobjekt, medan OpenAPI beskriver ett helt HTTP-API och lånar JSON Schema för delarna. Grannarna beskriver andra slags gränssnitt: AsyncAPI täcker meddelandedrivna system, gRPC-tjänster deklareras med Protocol Buffers i `.proto` -definitioner, och en GraphQL-tjänst publicerar sitt eget schema.

Använd begreppet i ett beslut

Använd ”Vad är OpenAPI?” 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

En beskrivning i ett standardformat är ett enda dokument som många verktyg kan läsa, och vart och ett sparar arbete som ett team annars hade fått göra för hand. Referensdokumentationen renderas från den. Klientbibliotek i flera språk genereras från den, så att en frontend anropar en typad funktion i stället för att sätta ihop en URL. Mockservrar svarar utifrån den, så att en frontend kan byggas

innan backend finns. API-gateways importerar rutter från den, testsviter validerar trafik mot den, och AI-agenter som anropar API:er läser den för att se vad som erbjuds.

Den räckvidden är också risken. Vart och ett av verktygen litar på dokumentet. En beskrivning som är fel är värre än ingen alls, eftersom den genererade klienten kompilerar, dokumentationen ser auktoritativ ut och misstaget visar sig vid körning i någon annans produkt.

Det viktigaste beslutet gäller därför var beskrivningen kommer ifrån. Vid design-first skriver människor beskrivningen, granskar den som kontraktet och bygger sedan servern så att den stämmer. API:ets form diskuteras innan det finns någon kod, vilket passar publika API:er och team som arbetar parallellt. Priset är att beskrivningen och servern är två saker som kan säga emot varandra. Vid code-first genereras beskrivningen från serverns egna typer och annoteringar. Den kan inte beskriva en rutt som servern saknar, men den är bara så exakt som typerna, och designdiskussionen sker i kodgranskningen i stället för i ett dokument. Hur man än gör förblir beskrivningen sann bara om ett bygge kontrollerar den.

Steget från 3.0 till 3.1 ändrade hur schemadelen skrivs, eftersom 3.1 tog över JSON Schema 2020-12. Generatorer, validerare och gateways stöder var och en sitt eget spann av versioner, så den version ett team skriver begränsas av det äldsta verktyget i kedjan.

Så ser det ut i praktiken

En OpenAPI-beskrivning kan vara dokumentation som följer koden eller en länk i en kedja som genererar kod, och arbetet här har gett den båda uppgifterna.

På en plattform började API:et på Fiber med validering skriven för hand, slutpunkt för slutpunkt, och utan något samlat dokument som beskrev dess form. HTTP-lagret flyttades till Huma v2, ovanpå Fiber, så varje slutpunkt deklareras med structar för indata och utdata och OpenAPI-beskrivningen genereras från de typerna; dokumentationen följer koden i stället för att åldras i en wiki. De två slutpunkterna för WebSocket ligger kvar på ren Fiber, utanför den modell med anrop och svar som Huma bygger på. Sidan förklarar varför skalan spelar roll: med 760 operationer fördelade på 649 sökvägar är beskrivningen i praktiken det enda sätt som någon kan läsa API:et på, så vetskapen om att den är komplett väger tyngre än vid femtio. En kontroll i CI håller pariteten mellan rutter och beskrivning på 100 %.

På en annan plattform är beskrivningen mittlänken i en kedja som går från databasen och utåt. Schemat och dess funktioner formar Go-typerna, Huma genererar beskrivningen, och det som frontenden använder genereras från den: en typad TypeScript-klient på 44 076 rader, 61 mock-hanterare för frontendens tester och de gränser som varje formulär upprätthåller. Den motverkar att frontend och backend glider isär, som oftast visar sig som en 422 i webbläsaren efter att avvikelsen redan har släppts. En kontraktskontroll i CI fäller bygget vid avvikelse, med en sond riktad mot valfria fält i anropets body, där ”saknas” och ”null” blandas ihop, och mot enum-värden i query-parametrar. Sidan sätter namn på priset: ett genereringssteg mitt i allt, och en kedja som bara är så pålitlig som sin sämst skyddade länk.

Företagets sida om att vara redo för AI-agenter tillämpar samma resonemang på agenter. En applikation som agenter kan använda erbjuder ett beskrivet API: ett OpenAPI-dokument, en MCP-server där agenter ska anropa det direkt, och OAuth-metadata så att en agent kan agera för en användare utan att få lösenordet. Sidan klassificerar varje rekommendation i stället för att lyda den, och eftersom företagets egen webbplats saknar API är en API-katalog och OAuth-metadata två av de fem som valdes bort där.

Var det går fel

Beskrivningen lovar något som servern inte gör. En handskött beskrivning halkar efter: en ny query-parameter, ett omdöpt fält eller en felkod som bara servern känner till. Genererade klienter kompilerar då mot en fiktion. Generera beskrivningen från koden, eller håll den separat och validera verkliga anrop och svar mot den i testsviten, och jämför de rutter som servern registrerar med de sökvägar som beskrivningen listar.

Varje omgenerering av klienten döper om hälften av metoderna. Utan `operationId` hittar generatorer på metodnamn utifrån sökvägen och HTTP-metoden, så att en flyttad rutt döper om den funktion som varje anropare använder. Specifikationen kräver att varje `operationId` är unikt och skiljer på stora och små bokstäver. Sätt ett på varje operation, välj det lika omsorgsfullt som namnet på en publik funktion, och låt en linter fälla bygget när ett saknas.

Fält som får vara null börjar avvisa null efter en uppgradering. OpenAPI 3.0 markerade dem med `nullable: true`. I 3.1 är det nyckelordet borta och typen blir en lista, och en validerare som stöter på ett nyckelord den inte känner igen brukar ignorera det. Ett schema för 3.0 som bara märks om till 3.1 avvisar därför

null överallt där författaren menade att tillåta det. Konvertera med uppgraderingsguiden från OpenAPI Initiative, som också tar upp att `exclusiveMinimum` blir ett tal:

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

Exemplen klarar inte sitt eget schema. Specifikationen säger att ett exempel ska stämma med sitt schema och lämnar kontrollen åt verktygen. En läsare kopierar exemplet ur dokumentationen och får en 422; en mockserver serverar det, och frontenden byggs mot data som det verkliga API:et kommer att avvisa. Validera varje exempel i CI, lika strikt som skarp trafik.

Den publika beskrivningen kartlägger admin-API:et. En beskrivning som genereras från varje rutt och serveras på en publik adress visar interna operationer och deras datastrukturer för vem som helst som frågar. Publicera en beskrivning per målgrupp, genererad med de interna operationerna bortfiltrerade. Specifikationen tillåter att objekt lämnas tomma eller tas bort för just den här sortens åtkomstkontroll.

Läs mer

- OpenAPI Specification — den senaste versionen av standarden, med varje objekt och fält definierat och en historik över varje utgåva.
- JSON Schema Draft 2020-12 — schemaspråket som OpenAPI 3.1 och senare använder för varje body i anrop och svar.
- Upgrading from OpenAPI 3.0 to 3.1 — listan från OpenAPI Initiative över ändringar i schemadelen, `nullable` bland dem.
- Huma: OpenAPI Generation — hur Go-ramverket som används i projekten ovan härleder en beskrivning ur typerna i hanterarna.

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 OpenAPI?” 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/openapi/