Engineer

OpenAPI

a machine-readable description of an HTTP API

What it is

OpenAPI is a standard format for writing down everything a caller needs to know about an HTTP API — its addresses, operations, inputs, outputs, errors and security — in one JSON or YAML file that people and programs read alike. The standard itself is the OpenAPI Specification, often abbreviated OAS, and a file written to it is an OpenAPI description. The format began as the Swagger Specification, and that older name survives in tools built around it, such as Swagger UI.

The specification says it defines a “standard, programming language-agnostic interface description for HTTP APIs”. The aim it states is that a person or a program can learn what a service does without its source code, extra documentation or a capture of its network traffic. The entry document is recommended to be named `openapi.json` or `openapi.yaml`, and larger descriptions are often split across several files joined by references.

Swagger 2.0 was released in September 2014 and donated to the OpenAPI Initiative on 31 December 2015. Version 3.0.0 followed in July 2017 under the new name, 3.1.0 in February 2021 and 3.2.0 in September 2025.

A description has a handful of top-level parts. `info` names the API and its version; `servers` lists where it runs; `paths` holds each address and, under it, one operation per HTTP method, with its parameters, request body and possible responses. `components` collects what is reused: data schemas, parameters, responses and the security schemes that say how a caller authenticates. Since 3.1, a `webhooks` section can also describe requests the API sends to its consumers. A minimal fragment:

```
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
```

The data shapes are written in a schema language. From 3.1 the Schema Object is a superset of JSON Schema Draft 2020-12; 3.0 used an extended subset of an older draft with additions of its own. JSON Schema describes one piece of data, while OpenAPI describes a whole HTTP API and borrows JSON Schema for the pieces. Its neighbours describe other kinds of interface: AsyncAPI covers message-driven systems, gRPC services are declared in Protocol Buffers `.proto` files, and a GraphQL server publishes its own schema.

Use the concept in a decision

Put “What is OpenAPI?” to work in a real decision. We can help relate the concept to the system you are building or buying. The part users never see: where the data lives, what the rules are, and how other systems are allowed to talk to it.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Designed and ran 6 ETL/ELT pipelines — 6 ETL pipelines across three database engines

Designed and ran 6 ETL/ELT pipelines Get in touch →

Why it matters

A description in a standard format is one file that many tools can read, each saving work a team would otherwise do by hand. Reference documentation renders from it. Client libraries in several languages are generated from it, so a frontend calls a typed function rather than assembling a URL. Mock servers

answer from it, so a frontend can be built before the backend exists. Gateways import routes from it, test suites validate traffic against it, and AI agents that call APIs read it to learn what is on offer.

That reach is also the risk. Every one of those tools trusts the file. A description that is wrong is worse than none, because the generated client compiles, the documentation looks authoritative, and the mistake shows up at runtime in someone else's product.

So the main decision is where the description comes from. In design-first work, people write the description, review it as the contract and then build the server to match. The shape of the API is argued over before any code exists, which suits public APIs and teams that work in parallel. The cost is that the description and the server are two things that can disagree. In code-first work, the description is generated from the server's own types and annotations. It cannot describe a route the server lacks, but it is only as precise as those types, and the design discussion happens in code review rather than in a document. Either way, the description stays true only if a build checks it.

Moving from 3.0 to 3.1 changed how schemas are written, because 3.1 adopted JSON Schema 2020-12. Generators, validators and gateways each support their own range of versions, so the version a team writes is limited by the oldest tool in the chain.

What it looks like in practice

Fiber carried one API at first, with validation written out by hand, endpoint by endpoint, and no single document that described its shape. The HTTP layer moved to Huma v2, on top of Fiber, so each endpoint is declared with input and output structs and the OpenAPI description is generated from those types; the documentation follows the code instead of ageing in a wiki. The two WebSocket endpoints stay on raw Fiber, outside Huma's request-and-response model. Scale is what makes that description matter: at 760 operations across 649 paths, it is in practice the only way anyone can read the API, so knowing it is complete matters more than it did at fifty. A CI check holds route parity at 100%.

In another project, the description is the middle link of a chain that runs from the database outward. The database schema and its functions shape the Go types, Huma emits the description, and what the frontend uses is generated from it: a typed TypeScript client of 44,076 lines, 61 mock handlers for the frontend's

tests, and the limits each form enforces. It answers drift between frontend and backend, which usually surfaces as a 422 in the browser after the mismatch has shipped. A contract check in CI fails the build on divergence, with a probe aimed at optional body fields, where “missing” and “null” get confused, and at query-parameter enums. What that costs is a generation step in the middle of everything, and a chain only as trustworthy as its least-guarded link.

The company’s page on readiness for AI agents applies the same reasoning to agents. An application that agents can use offers a described API: an OpenAPI document, an MCP server where agents should call it directly, and OAuth metadata so an agent can act for a user without being given the password. The page classifies each recommendation rather than obeying it, and because the company’s own site has no API, an API catalogue and OAuth metadata are two of the five it declined there.

Where it goes wrong

The description promises something the server does not do. A hand-maintained description falls behind: a new query parameter, a renamed field or an error code that only the server knows about. Generated clients then compile against a fiction. Generate the description from the code, or keep it separate and validate real requests and responses against it in the test suite, and compare the routes the server registers with the paths the description lists.

Each client regeneration renames half the methods. Without an `operationId`, generators invent method names from the path and the HTTP method, so moving a route renames the function every caller uses. The specification requires each `operationId` to be unique and treats it as case-sensitive. Set one on every operation, choose it as carefully as a public function name, and make a linter fail the build when one is missing.

Fields that may be null start rejecting null after an upgrade. OpenAPI 3.0 marked them with `nullable: true`. In 3.1 that keyword is gone and the type becomes a list, and a validator that meets a keyword it does not know usually ignores it. A 3.0 schema relabelled as 3.1 therefore rejects null wherever its author meant to allow it. Convert schemas with the OpenAPI Initiative’s upgrade guide, which also covers `exclusiveMinimum` becoming a number:

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

The examples fail their own schema. The specification says an example should match its schema and leaves checking it to tools. A reader copies the example from the documentation and gets a 422; a mock server serves it, and the frontend is built against data the real API will reject. Validate every example in CI, as strictly as live traffic.

The public description maps the admin API. One description generated from every route, and served at a public address, lists internal operations and their schemas to anyone who asks. Publish one description per audience, generated with the internal operations filtered out. The specification allows objects to be left empty or removed for exactly this kind of access control.

Further reading

- OpenAPI Specification — the latest version of the standard, with every object and field defined and a history of each release.
- JSON Schema Draft 2020-12 — the schema language that OpenAPI 3.1 and later use for every request and response body.
- Upgrading from OpenAPI 3.0 to 3.1 — the OpenAPI Initiative's own list of schema changes, `nullable` among them.
- Huma: OpenAPI Generation — how the Go framework used in the projects above derives a description from handler types.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is OpenAPI?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/openapi/