Engineer

OpenAPI

Eine maschinenlesbare Beschreibung einer HTTP-API

Was es ist

OpenAPI ist ein Standardformat, um alles festzuhalten, was ein Aufrufer über eine HTTP-API wissen muss: ihre Adressen, Operationen, Eingaben, Ausgaben, Fehler und Sicherheitsmechanismen. Das geschieht in einer JSON- oder YAML-Datei, die Menschen und Programme gleichermaßen lesen können. Der Standard selbst heißt OpenAPI Specification, häufig OAS abgekürzt; eine nach ihm verfasste Datei ist eine OpenAPI-Beschreibung. Das Format begann als Swagger Specification. Dieser ältere Name lebt in darauf aufbauenden Werkzeugen wie Swagger UI weiter.

Die Spezifikation definiert eine „standardisierte, von der Programmiersprache unabhängige Schnittstellenbeschreibung für HTTP-APIs“. Ihr erklärtes Ziel ist, dass ein Mensch oder ein Programm herausfinden kann, was ein Dienst tut, ohne dessen Quellcode, zusätzliche Dokumentation oder eine Aufzeichnung seines Netzwerkverkehrs zu benötigen. Für das Einstiegsdokument werden die Namen `openapi.json` oder `openapi.yaml` empfohlen. Größere Beschreibungen sind häufig auf mehrere durch Referenzen verbundene Dateien aufgeteilt.

Swagger 2.0 erschien im September 2014 und wurde am 31. Dezember 2015 der OpenAPI Initiative übergeben. Unter dem neuen Namen folgte Version 3.0.0 im Juli 2017, 3.1.0 im Februar 2021 und 3.2.0 im September 2025.

Eine Beschreibung hat einige wenige Hauptbestandteile. `info` nennt die API und ihre Version; `servers` führt ihre Betriebsadressen auf. `paths` enthält jede Adresse und darunter je eine Operation pro HTTP-Methode mit Parametern, Anfrageinhalt und möglichen Antworten. `components` sammelt wiederverwendete Bestandteile: Datenschemas, Parameter, Antworten und

Sicherheitsschemata, die beschreiben, wie sich ein Aufrufer authentifiziert. Seit 3.1 kann ein Abschnitt `webhooks` auch Anfragen beschreiben, die die API an ihre Nutzer sendet. Ein minimales 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
```

Die Datenstrukturen werden in einer Schemasprache beschrieben. Seit 3.1 ist das Schema Object eine Obermenge von JSON Schema Draft 2020-12; 3.0 verwendete eine erweiterte Teilmenge eines älteren Entwurfs mit eigenen Ergänzungen. JSON Schema beschreibt ein einzelnes Datenelement, OpenAPI dagegen eine vollständige HTTP-API und verwendet JSON Schema für deren Bestandteile. Benachbarte Standards beschreiben andere Schnittstellenarten: AsyncAPI deckt nachrichtengesteuerte Systeme ab, gRPC-Dienste werden in `.proto` -Dateien von Protocol Buffers deklariert, und ein GraphQL-Server veröffentlicht sein eigenes Schema.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist OpenAPI?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Der Teil, den Nutzer nie sehen: wo die Daten liegen, welche Regeln gelten und wie andere Systeme darauf zugreifen dürfen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

Eine Beschreibung in einem Standardformat ist eine Datei, die viele Werkzeuge lesen können. Jedes erspart dem Team Arbeit, die es sonst von Hand erledigen müsste. Daraus entsteht eine Referenzdokumentation. Daraus werden Clientbibliotheken in mehreren Sprachen generiert, sodass ein Frontend eine typisierte Funktion aufruft, statt eine URL zusammenzusetzen. Mock-Server beantworten Anfragen anhand der Beschreibung, sodass das Frontend vor dem Backend entstehen kann. Gateways importieren daraus Routen, Testsuiten prüfen den Datenverkehr dagegen, und KI-Agenten lesen sie, um zu erfahren, welche API-Aufrufe angeboten werden.

Diese Reichweite ist zugleich das Risiko. Jedes dieser Werkzeuge vertraut der Datei. Eine falsche Beschreibung ist schlimmer als gar keine: Der generierte Client lässt sich kompilieren, die Dokumentation wirkt verbindlich, und der Fehler zeigt sich zur Laufzeit im Produkt eines anderen.

Die wichtigste Entscheidung ist deshalb, woher die Beschreibung stammt. Bei Design-first schreiben Menschen die Beschreibung, prüfen sie als Vertrag und bauen danach den passenden Server. Über die Form der API wird vor dem ersten Code diskutiert. Das eignet sich für öffentliche APIs und parallel arbeitende Teams. Der Nachteil: Beschreibung und Server sind zwei getrennte Dinge, die voneinander abweichen können. Bei Code-first wird die Beschreibung aus den Typen und Annotationen des Servers generiert. Sie kann keine Route beschreiben, die der Server nicht hat, ist aber nur so präzise wie diese Typen. Die Entwurfsdiskussion findet dann im Code-Review statt und nicht in einem Dokument. In beiden Fällen bleibt die Beschreibung nur dann korrekt, wenn ein Build sie prüft.

Der Wechsel von 3.0 auf 3.1 änderte die Schreibweise der Schemas, weil 3.1 JSON Schema 2020-12 übernahm. Generatoren, Validatoren und Gateways unterstützen jeweils unterschiedliche Versionsbereiche. Welche Version ein Team schreiben kann, begrenzt deshalb das älteste Werkzeug in der Kette.

Wie es in der Praxis aussieht

Eine API lief zunächst auf Fiber, mit für jeden Endpunkt von Hand geschriebener Validierung und ohne ein gemeinsames Dokument, das ihre Struktur beschrieb. Die HTTP-Schicht wechselte zu Huma v2 auf Fiber. Jeder Endpunkt wird seitdem mit Ein- und Ausgabestrukturen deklariert, aus deren Typen die OpenAPI-

Beschreibung entsteht. Die Dokumentation folgt dem Code, statt in einem Wiki zu veralten. Die beiden WebSocket-Endpunkte bleiben direkt auf Fiber, außerhalb von Humas Anfrage-Antwort-Modell. Erst der Umfang macht die Beschreibung so wichtig: Bei 760 Operationen auf 649 Pfaden ist sie praktisch die einzige Möglichkeit, die API überhaupt zu überblicken. Ihre Vollständigkeit ist deshalb wichtiger als noch bei fünfzig Operationen. Eine CI-Prüfung hält die Routenparität bei 100 %.

In einem anderen Projekt bildet die Beschreibung das mittlere Glied einer Kette, die bei der Datenbank beginnt. Das Datenbankschema und seine Funktionen bestimmen die Go-Typen. Huma erzeugt daraus die Beschreibung, aus der wiederum die vom Frontend verwendeten Bestandteile generiert werden: ein typisierter TypeScript-Client mit 44.076 Zeilen, 61 Mock-Handler für die Frontendtests und die Grenzen, die jedes Formular durchsetzt. Das begegnet Abweichungen zwischen Frontend und Backend, die üblicherweise erst nach der Auslieferung als 422 im Browser auffallen. Eine Vertragsprüfung in der CI lässt den Build bei Abweichungen scheitern. Ihre Prüfungen zielen auf optionale Felder im Anfrageinhalt, bei denen „fehlend“ und „null“ verwechselt werden, sowie auf Aufzählungswerte von Anfrageparametern. Der Preis ist ein Generierungsschritt mitten im gesamten Ablauf und eine Kette, die nur so verlässlich ist wie ihr am wenigsten abgesichertes Glied.

Die Unternehmensseite zur Nutzbarkeit durch KI-Agenten wendet dieselbe Überlegung auf Agenten an. Eine Anwendung, die Agenten verwenden können, bietet eine beschriebene API: ein OpenAPI-Dokument, einen MCP-Server, wenn Agenten sie direkt aufrufen sollten, und OAuth-Metadaten, damit ein Agent im Namen eines Nutzers handeln kann, ohne dessen Passwort zu erhalten. Die Seite ordnet jede Empfehlung ein, statt sie einfach zu befolgen. Da die eigene Website des Unternehmens keine API hat, gehören ein API-Katalog und OAuth-Metadaten zu den fünf dort abgelehnten Empfehlungen.

Wo es schiefgeht

Die Beschreibung verspricht etwas, das der Server nicht tut. Eine von Hand gepflegte Beschreibung gerät in Rückstand: Ein neuer Anfrageparameter, ein umbenanntes Feld oder ein Fehlercode ist nur dem Server bekannt. Generierte Clients werden dann gegen eine Fiktion kompiliert. Generieren Sie die Beschreibung aus dem Code oder pflegen Sie sie getrennt und prüfen Sie in der Testsuite echte Anfragen und Antworten dagegen. Vergleichen Sie außerdem die vom Server registrierten Routen mit den in der Beschreibung aufgeführten Pfaden.

Jede erneute Clientgenerierung benennt die Hälfte der Methoden um. Ohne `operationId` erfinden Generatoren Methodennamen aus Pfad und HTTP-Methode. Eine verschobene Route benennt dadurch die Funktion um, die jeder Aufrufer verwendet. Die Spezifikation verlangt eine eindeutige `operationId` für jede Operation und unterscheidet dabei Groß- und Kleinschreibung. Vergeben Sie für jede Operation eine, wählen Sie sie so sorgfältig wie den Namen einer öffentlichen Funktion und lassen Sie einen Linter den Build bei fehlenden Werten abbrechen.

Felder, die null sein dürfen, lehnen null nach einem Upgrade ab. OpenAPI 3.0 kennzeichnete sie mit `nullable: true`. In 3.1 entfällt dieses Schlüsselwort und der Typ wird als Liste angegeben. Ein Validator ignoriert unbekannte Schlüsselwörter üblicherweise. Ein lediglich als 3.1 umetikettiertes 3.0-Schema lehnt null deshalb überall dort ab, wo es erlaubt sein sollte. Konvertieren Sie Schemas mit der Upgrade-Anleitung der OpenAPI Initiative. Sie behandelt auch, dass `exclusiveMinimum` zu einer Zahl wird:

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

Die Beispiele erfüllen ihr eigenes Schema nicht. Die Spezifikation sagt, dass ein Beispiel seinem Schema entsprechen sollte, überlässt die Prüfung aber den Werkzeugen. Ein Leser kopiert das Beispiel aus der Dokumentation und erhält einen 422. Ein Mock-Server liefert es aus, und das Frontend wird für Daten gebaut, die die echte API ablehnen wird. Validieren Sie jedes Beispiel in der CI genauso streng wie echten Datenverkehr.

Die öffentliche Beschreibung legt die Admin-API offen. Eine aus sämtlichen Routen generierte und öffentlich bereitgestellte Beschreibung zeigt jedem, der sie abruft, die internen Operationen und ihre Schemas. Veröffentlichen Sie für jede Zielgruppe eine eigene Beschreibung, bei deren Generierung interne Operationen herausgefiltert werden. Genau für diese Art der Zugriffskontrolle erlaubt die Spezifikation, Objekte leer zu lassen oder zu entfernen.

Weiterführende Lektüre

- OpenAPI Specification — die aktuelle Fassung des Standards mit Definitionen aller Objekte und Felder sowie der Geschichte jeder Veröffentlichung.
- JSON Schema Draft 2020-12 — die Schemasprache, die OpenAPI ab 3.1 für jeden Anfrage- und Antwortinhalt verwendet.
- Upgrade von OpenAPI 3.0 auf 3.1 — die eigene Liste der Schemaänderungen der OpenAPI Initiative, darunter `nullable`.
- Huma: OpenAPI-Generierung — wie das in den genannten Projekten verwendete Go-Framework aus Handler-Typen eine Beschreibung ableitet.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist OpenAPI?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/openapi/