Engineer

OpenAPI

Description d'une API HTTP lisible par les machines

Ce que c'est

OpenAPI est un format standard qui consigne tout ce que l'appelant doit savoir sur une API HTTP : ses adresses, opérations, entrées, sorties, erreurs et mécanismes de sécurité, dans un fichier JSON ou YAML lisible aussi bien par les personnes que par les programmes. La norme elle-même s'appelle OpenAPI Specification, souvent abrégée en OAS, et un fichier qui la suit est une description OpenAPI. Le format a d'abord porté le nom de Swagger Specification, qui subsiste dans les outils construits autour de lui, comme Swagger UI.

La spécification indique qu'elle définit une « description d'interface standard pour les API HTTP, indépendante du langage de programmation ». Son objectif déclaré est qu'une personne ou un programme puisse découvrir ce que fait un service sans son code source, une documentation supplémentaire ou une capture de son trafic réseau. Il est recommandé de nommer le document d'entrée `openapi.json` ou `openapi.yaml` . Les descriptions plus volumineuses sont souvent réparties entre plusieurs fichiers reliés par des références.

Swagger 2.0 est sorti en septembre 2014 et a été donné à l'OpenAPI Initiative le 31 décembre 2015. La version 3.0.0 a suivi sous le nouveau nom en juillet 2017, puis la 3.1.0 en février 2021 et la 3.2.0 en septembre 2025.

Une description comporte quelques grandes parties. `info` nomme l'API et indique sa version ; `servers` énumère les endroits où elle s'exécute ; `paths` contient chaque adresse et, sous celle-ci, une opération par méthode HTTP, avec ses paramètres, son corps de requête et les réponses possibles. `components` rassemble les éléments réutilisés : schémas de données, paramètres, réponses et

mécanismes de sécurité qui indiquent comment l'appelant s'authentifie. Depuis la version 3.1, une section `webhooks` peut également décrire les requêtes que l'API envoie à ses consommateurs. Un fragment minimal :

```
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
```

La structure des données s'écrit dans un langage de schéma. Depuis la version 3.1, l'objet de schéma (Schema Object) est un surensemble de JSON Schema Draft 2020-12 ; la version 3.0 utilisait un sous-ensemble étendu d'une ancienne version de travail, avec ses propres ajouts. JSON Schema décrit un élément de données, tandis qu'OpenAPI décrit une API HTTP entière et emprunte JSON Schema pour ses éléments. Les formats voisins décrivent d'autres types d'interfaces : AsyncAPI couvre les systèmes à messages, les services gRPC se déclarent dans des fichiers `.proto` de Protocol Buffers, et un serveur GraphQL publie son propre schéma.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'OpenAPI ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. La partie invisible pour les utilisateurs : où résident les données, quelles règles s'appliquent et comment les autres systèmes peuvent y accéder.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 6 pipelines ETL/ELT conçus et exploités — 6 pipelines ETL sur trois moteurs de bases de données

6 pipelines ETL/ELT conçus et exploités Contactez-nous →

Pourquoi c’est important

Une description dans un format standard est un fichier que de nombreux outils peuvent lire, chacun épargnant à l’équipe un travail qu’elle ferait autrement à la main. La documentation de référence est produite à partir de ce fichier. Des bibliothèques clientes sont générées dans plusieurs langages, si bien qu’un frontend appelle une fonction typée plutôt que d’assembler une URL. Des serveurs simulés répondent à partir de cette description, ce qui permet de construire le frontend avant que le backend existe. Les passerelles en importent les routes, les suites de tests valident le trafic par rapport à elle, et les agents d’IA qui appellent des API la lisent pour découvrir les possibilités offertes.

Cette portée est aussi une source de risque. Chacun de ces outils fait confiance au fichier. Une description erronée est pire qu’une absence de description : le client généré compile, la documentation paraît faire autorité, et l’erreur se révèle à l’exécution dans le produit de quelqu’un d’autre.

La décision principale porte donc sur l’origine de la description. Dans une démarche où la conception vient d’abord, l’équipe rédige la description, la revoit comme un contrat, puis construit le serveur pour qu’il la respecte. La forme de l’API se discute avant que le moindre code existe, ce qui convient aux API publiques et aux équipes qui travaillent en parallèle. Le prix est que la description et le serveur sont deux choses qui peuvent se contredire. Dans une démarche où le code vient d’abord, la description est générée à partir des types et des annotations du serveur. Elle ne peut pas décrire une route absente du serveur, mais sa précision dépend de celle de ces types, et la discussion de conception se tient en revue de code plutôt que dans un document. Dans les deux cas, la description ne reste exacte que si une étape de construction en vérifie l’exactitude.

Le passage de la version 3.0 à la version 3.1 a changé l’écriture des schémas, car la version 3.1 a adopté JSON Schema 2020-12. Générateurs, validateurs et passerelles prennent chacun en charge leur propre plage de versions : la version qu’une équipe peut utiliser est donc limitée par l’outil le plus ancien de la chaîne.

À quoi cela ressemble en pratique

Une API reposait initialement sur Fiber, avec une validation écrite à la main, point de terminaison par point de terminaison, sans document unique pour en décrire la forme. La couche HTTP est passée à Huma v2, par-dessus Fiber : chaque point

de terminaison se déclare avec des structures d'entrée et de sortie, et la description OpenAPI est générée à partir de ces types. La documentation suit le code au lieu de vieillir dans un wiki. Les deux points de terminaison WebSocket restent sur Fiber directement, en dehors du modèle requête-réponse de Huma. L'échelle donne son importance à cette description : avec 760 opérations sur 649 chemins, elle constitue en pratique le seul moyen de lire l'API. Savoir qu'elle est complète compte donc davantage que lorsqu'il n'y avait que cinquante opérations. Une vérification en CI maintient la parité des routes à 100 %.

Dans un autre projet, la description est le maillon central d'une chaîne qui part de la base de données. Le schéma de la base et ses fonctions déterminent les types Go ; Huma produit la description, à partir de laquelle est généré ce qu'utilise le frontend : un client TypeScript typé de 44 076 lignes, 61 gestionnaires simulés pour les tests du frontend et les limites que chaque formulaire fait respecter. Cette chaîne répond aux divergences entre frontend et backend, qui se manifestent généralement par un code 422 dans le navigateur une fois l'incohérence livrée. Une vérification du contrat en CI fait échouer la construction en cas de divergence, avec un test ciblant les champs facultatifs du corps, où « absent » et « null » se confondent, ainsi que les énumérations des paramètres de requête. Le prix est une étape de génération au milieu de tout, et une chaîne dont la fiabilité se limite à celle de son maillon le moins contrôlé.

La page de l'entreprise consacrée à la préparation aux agents d'IA applique le même raisonnement aux agents. Une application utilisable par des agents propose une API décrite : un document OpenAPI, un serveur MCP lorsque les agents devraient l'appeler directement, et des métadonnées OAuth pour qu'un agent puisse agir pour un utilisateur sans recevoir son mot de passe. La page classe chaque recommandation au lieu de s'y conformer systématiquement. Comme le site de l'entreprise lui-même n'a pas d'API, le catalogue d'API et les métadonnées OAuth font partie des cinq recommandations qu'elle y a écartées.

Où les problèmes surviennent

La description promet quelque chose que le serveur ne fait pas. Une description maintenue manuellement prend du retard : un nouveau paramètre de requête, un champ renommé ou un code d'erreur connu du seul serveur. Les clients générés compilent alors sur la base d'un contrat fictif. Générez la description à partir du code, ou gardez-la séparée en validant les requêtes et réponses réelles par rapport à elle dans la suite de tests. Comparez également les routes enregistrées par le serveur aux chemins énumérés dans la description.

Chaque régénération du client renomme la moitié des méthodes. Sans `operationId`, les générateurs inventent les noms des méthodes à partir du chemin et de la méthode HTTP. Déplacer une route renomme donc la fonction qu'utilise chaque appelant. La spécification exige que chaque `operationId` soit unique et le traite comme sensible à la casse. Définissez-en un pour chaque opération, choisissez-le avec autant de soin qu'un nom de fonction publique, et configurez un linter pour faire échouer la construction lorsqu'il en manque un.

Des champs qui acceptaient null le refusent après une mise à niveau. OpenAPI 3.0 les marquait avec `nullable: true`. En version 3.1, ce mot-clé disparaît et le type devient une liste. Un validateur ignore généralement les mots-clés qu'il ne connaît pas. Un schéma 3.0 simplement réétiqueté 3.1 refuse donc null là où son auteur souhaitait l'autoriser. Convertissez les schémas avec le guide de mise à niveau de l'OpenAPI Initiative, qui explique aussi comment `exclusiveMinimum` devient un nombre :

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

Les exemples ne respectent pas leur propre schéma. La spécification indique qu'un exemple devrait respecter son schéma et laisse les outils le vérifier. Un lecteur copie l'exemple depuis la documentation et reçoit un code 422 ; un serveur simulé le fournit, et le frontend se construit autour de données que la vraie API refusera. Validez chaque exemple en CI aussi rigoureusement que le trafic réel.

La description publique expose la structure de l'API d'administration. Une description générée à partir de toutes les routes et servie à une adresse publique énumère les opérations internes et leurs schémas pour quiconque la demande. Publiez une description par public visé, générée en excluant les opérations internes. La spécification autorise à laisser des objets vides ou à les retirer précisément pour ce type de contrôle d'accès.

Pour aller plus loin

- Spécification OpenAPI — la dernière version de la norme, avec chaque objet et chaque champ définis et un historique de chaque version.
- JSON Schema Draft 2020-12 — le langage de schéma qu'OpenAPI 3.1 et les versions suivantes utilisent pour chaque corps de requête et de réponse.
- Passer d'OpenAPI 3.0 à 3.1 — la liste des changements de schéma établie par l'OpenAPI Initiative, dont `nullable` .
- Huma : génération OpenAPI — comment le framework Go utilisé dans les projets ci-dessus tire une description des types de ses gestionnaires.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'OpenAPI ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/openapi/