**Spécification OpenAPI**

1. **Introduction :**

Ces dernières années, les systèmes informatiques modernes sont souvent une interaction de services, vu que le réseau de communication le plus fréquemment utilisé est Internet. La plupart de ces services utilisent le Web pour leurs interactions via un réseau de communication. Maintenant que les clients dépendent d’avantage des produits et services numériques, les entrepreneurs ont réalisé à quel point il est important d'investir dans des applications logicielles d'entreprise. Un exemple de la sorte est l'interface de programmation d'application API.

Ces clients, dans leur rôle de demandeurs, intègrent les services existants de différents fournisseurs dans de nouveaux systèmes informatiques et ces fournisseurs utilisent des cadres comme OpenAPI pour créer des spécifications de service syntaxique à partir desquelles les demandeurs génèrent du code pour intégrer les services.

Cependant, la spécification OpenAPI s'est imposée comme la norme commune pour définir les API REST et de permettre aux développeurs de créer des interfaces d'API REST de manière à ce que les utilisateurs puissent les comprendre avec un minimum d’efforts.

Tous les exemples illustratifs dans ce document sont en YAML et la version de spécification utilisée est 3.

1. **Définition et Description :**

Une API est une interface de programmation, c'est-à-dire un ensemble de classes, de fonctions et de méthodes qui servent de façade à un logiciel permettant à d'autres logiciels d’accéder aux services de ce logiciel. Une spécification est une description des services et fonctionnalités fournis par un logiciel. Ainsi la spécification OpenAPI (OAS) anciennement connue sous le nom de spécification Swagger, est une spécification qui décrit une d'interface standard pour les API web qui, permet aux humains et aux ordinateurs de découvrir, de consulter et de comprendre les capacités du service (web) sans connaissance de l'implémentation du serveur ou sans avoir accès au code source.

Elle offre huit objets parents qui ne sont pas tous obligatoires pour bien décrire une API et eux aussi peuvent contenir plusieurs autres objets. Ces huit objets sont les suivants :

paths

object

servers

object

info

object

openapi

string

externalsDocs

object

tags

object

security

object

components

object

1. ***openapi*** :

C’est une chaine correspondant à la version de la spécification de openAPI qui est remplacé par « swagger » pour les versions de spécification inférieur à 3.

1. ***info*** :

Il contient les informations de base à propos de l’API parmi lesquelles 2 sont obligatoires :

* **title** : il s’agit du titre de l’API du type chaine de caractère
* **version** : ce champ indique la version de l’API
* **Autres** : les facultatifs comme : description, contact et licence etc.

1. ***servers*** :

Il spécifie le lien et une description du serveur utilisé et on a la possibilité de définir plusieurs serveurs.

1. ***paths*** :

C’est un objet JSON dans lequel les opérations telles que GET, PUT, POST et DELETE seront définis. C’est une des parties dans laquelle nous pouvons ajouter des extensions c’est-à-dire créer notre propre operateur.

**exemple** :

paths:

'/chemin:

x-monMethode:

description: ma propre methode

1. ***components*** :

C’est cette partie qui définit les objets réutilisables. Parfois, plusieurs opérations d’une API peuvent avoir les mêmes paramètres communs ou retourner les mêmes structures de réponse. Pour éviter la duplication du code on peut le définir dans une place commune (components) et le référencé en utilisant la propriété **$*ref*** (mais « *securitySchemes* » est une exception car il est référencé par son nom) qui indique le chemin où le code est défini. Il serve de conteneur et ces composantes n’ont pas d’effet direct sur l’API a moins qu’ils soient référencés quelques parts à l’extérieur de l’objet « components ». Il est considéré comme une enveloppe dans laquelle plusieurs composantes sont définis, parmi ces composantes les plus importantes sont

* ***schemas*** : il décrit la structure d’une requête, une réponse, un paramètre ou un type. Ce type peut être composé ou primitif, un type d’entrée ou fourni comme sortie. Il précise aussi les contraintes d’un type, son modèle, son format, énumérer les valeurs prise en compte par ce type, définir une valeur par défaut ou autres seulement en définissant les propriétés correspondantes.

**Exemple**:

components:

schemas:

personne:

type: object

properties:

nom:

type: string

prenom:

type: string

adresse:

type: object

properties:

rue:

type: integer

codePostal:

type: integer

format: int64

default: 1500

ville:

type: string

example: Tizi-Ouzou

pays:

type: string

example: Algérie

status:

type: string

enum:

- Celibateur

- Mariée

* ***parameters*** : il décrit les paramètres d’une opération comme ceux nécessité dans une requête et peut contenir un ***schema*** (= schemas).

**Exemple :**

components:

parameters:

parametre:

name: ID

in: query

description: "identifiant"

required: true

schema:

type: "integer"

format: "int32"

default: 1

* ***securitySchemes*** : il décrit les différentes mode de sécurité applicable aux différentes parties fonctionnelles d’une API (voir l’exemple d’utilisation).
* ***requestBody***: il décrit les manières de représentation d’une réponse tel que en JSON, XML, TEXT/PLAIN.

**Exemple :**

requestBody:

content:

application/json:

schema:

$ref: "#/components/schemas/etudiant"

application/xml:

schema:

$ref: "#/components/schemas/etudiant"

1. ***security*** :

L’objet security de la spécification de openAPI 3 décrit les manières dont une partie d’une API ou une API peut être sécurisée, il spécifie le protocole de sécurité ou d'autorisation utilisé lors de la soumission des demandes. Cette spécification supporte quatre approches de définition de sécurité qui sont : "apiKey", "http", "oauth2 (le plus utilisé)", "openIdConnect". Plusieurs types de sécurité peuvent être définit et réutilisé dans la même et différentes parties de la spécification avec cet objet.

Après avoir défini les schémas de sécurité dans la section securitySchemes(dans components), la sécurité peuvent être appliquer à l'ensemble de l'API ou à des opérations individuelles en ajoutant respectivement la section de sécurité(security) au niveau racine ou au niveau opération. Dans le premier cas, les schémas de sécurité spécifiés sont appliqués globalement à toutes les opérations de l'API et dans l’autre ils ne concernent que l’opération en question (voir l’exemple d’utilisation).

1. ***tags*** :

C’est un tableau dont les entrés doivent être le même que la valeur du propriété tags de l’objet d’un opérateur. Il donne un résumé des différentes parties de l’objet **paths**.

1. ***externalsDocs*** :

Il permet d’inclure des liens vers un document externe. Cet objet peut être aussi définit dans les opérateurs dans l’objet **paths**.

Exemple:

paths:

/etudiants:

get:

tags:

- **Récupération**

responses:

200:

description: une liste d’étudiant

tags:

- name: **Récupération**

- name: Modification

Une API est décrite grâce à la définition des propriétés de ces différents objets. L’inclusion de ces objets en suivant la syntaxe (en JSON ou en YAML) permet une description détaillée ce qui renseigne l’utilisateur : de la version, du format, des types de donnée prise en charge par l’API, le type de sécurité utilisé, etc. car c’est un formalisme de description : de structure des documents APIs, des manières pour adresser des requêtes correctes aux serveurs http, de quelle forme est la réponse et sur quelle contrainte les services sont fournis.

1. **Les langages et outils utilisés** :
2. **Les langages utilisés** :
3. **JSON (JavaScript Objet Notation)** :

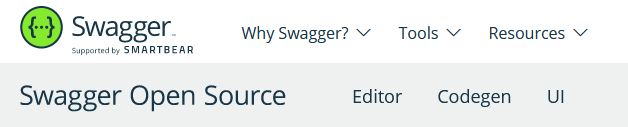
JavaScript Objet Notation est un format de données basé sur les types de données de la programmation en langage JavaScript [4]. C’est un langage léger et est devenu le format principal d’échange d’information. Il permet de représenter les données avec une structure arborescence dont l’affichage respect la règle clé-valeur. C'est devenu le format le plus populaire pour envoyer des requêtes API et des réponses via le protocole HTTP car il est facile à comprendre pour développeurs et les machines. C’est une représentation d’objet sachant que ces objets peuvent contenir d’autres objets. JSON prend en charge six (6) type de données qui sont : objets (*objects*), tableaux (*arrays*), chaînes (*strings*), nombres (*numbers*), valeurs booléennes (*boolean values*) et null.

1. **YAML (Yet Another Markup Language)** :

YAML est un langage de sérialisation de données conçu pour être fonctionnel et lisible pour les humains [5]. Ce n’est pas un langage de balisage, il est utilisé dans les applications où les données sont stockées ou transmises. Il est compréhensible par les programmeurs, respect le principe d’indentation et utilise les tirets comme une nouvelle entrée dans un tableau. Ce langage spécifie les mêmes types d’informations que JSON. YAML a été spécialement créé pour bien fonctionner dans les cas d'utilisation courants tels que : fichiers de configuration, fichiers journaux, messagerie interprocessus, partage de données en plusieurs langues afin de faciliter la compréhension et la visualisation pour simplifier la programmation.

1. **Les outils utilisés** :

L’environnement Swagger est un environnement composé de différente partie :



* **Swagger editor :** permet de décrire des documents APIs web, de modifier des fichier APIs et de concevoir de nouvelle API. C’est un éditeur open source en ligne et c’est le moyen le plus facile pour commencer avec la spécification OpenAPI. Les langages supportés sont soit JSON soit YAML.
* **Swagger codegen** : ou le générateur de code, c’est ensemble de fichier à l’intérieur de Swagger permettant de générer le code client/serveur de différente plateforme de développement. Il génère le code pour la construction plus rapide de l’API définie avec la spécification OpenAPI.
* **Swagger User Interface** : L'interface utilisateur Swagger permet à n'importe qui, que ce une équipe de développement ou les consommateurs finaux, de visualiser et d'interagir avec les ressources de l'API sans avoir mis en place la logique de mise en œuvre [6]. Cette interface est générée automatiquement à partir de la spécification. Elle permet la documentation visuelle facilitant la mise en œuvre du back-end et la consommation côté client.
* **Swagger Inspector** : c’est un outil qui permet de tester facilement les API sur les Cloud. Il facilite la création de la documentation des APIs.

1. **La syntaxe de cette spécification** :

La spécification OpenAPI est en générale versionnée de manière sémantique impliquant les fonctionnalités incompatibles apportée par rapport aux versions précédentes et l’ajout des nouvelles extensions. C’est une spécification qui commence de la version 3.0.0. Cette spécification doit suivre une syntaxe bien précise comme suit :

openapi: 3.0.0

info:

title: My API Specification

version: 1.0.0

servers:

- url: http

- url: https

paths:

/etudiants:

get:

tags:

- Récupération

responses:

200:

description: une liste d’étudiant

components:

securitySchemes:

api\_key:

name: apiKey1

type: apiKey

in: query

security:

- api\_key: []

tags:

- name: Récupération

- name: Modification

externalDocs:

url: https://swagger.io/specification/

description: Apprendre la spécification

Figure 1 : Exemple d’une description d’une API

Cet exemple définit le code minimum pour la description d’une API en précisant la version de la spécification et de l’API, le nom de la spécification et l’absence des opérateurs.

1. **Ses Caractéristiques :**

Une spécification de OpenAPI peut être reconnu par :

* Son utilisation d’un ensemble d’objet JSON (peut être traduit en YAML) avec leur schéma spécifique pour définir la dénomination, l’ordre, et le contenu de chaque part d’une API.
* Sa description des APIs (REST API) dans le format standard (open source) et l’utilisation du modèle Client/serveur.
* La validité de sa description quel qu’en soit le langage de programmation utilisé par les clients.
* Elle supporte l’ajout des extensions.

1. **Son principe de fonctionnement :**

La spécification de OpenAPI est conçue sur la base d’apporter de l’aide aux utilisateurs des services APIs pendant les développements logiciels. Elle met le comportement d’une API en évidence vis-à-vis des utilisateurs avec usage des objets, la mise en place d’une opération, le type de serveurs (http, https) utilisés.

Elle propose un bref résumé sur la structure des différentes opérations en précisant les paramètres (id) dans l’objet « *parameter* » d’un operateur et si nécessaire définir une sécurité d’authentification avant leur utilisation.

La communication est basée uniquement sur le protocole http qui constitue l’envoie de requêtes et récupération de réponses. C’est une spécification basée sur le modèle Client/serveur. Elle indique des informations sur les codes d’état http pour préciser le succès (200), la redirection (301,302), accès refusés (403), l’erreur client et erreur serveur (500,503). Si ces codes sont définis, l’état de la requête sera interprété par la description du code correspondant.

**Exemple d’utilisation** :

openapi: 3.0.0

info:

title: My API Specification

version: 1.0.0

components:

securitySchemes:

api\_key:

name: apiKey1

type: apiKey

in: query

parameters:

parametre:

name: ID

in: query

description: "identifiant d’un item"

required: true

schema:

type: "integer"

format: "int32"

schemas:

etudiant:

type: object

properties:

matricule:

type: string

example: MH54PP55232

nom:

type: string

prenom:

type: string

dateNaissance:

type: string

example: 01/01/2021

paths:

/etudiants:

get:

tags:

- Operations

parameters:

- $ref: "#/components/parameters/parametre"

responses:

200:

description: une d’étudiant

default:

description: Autre reponse

#Cas 2 : sécurisée GET

security:

- api\_key: []

#Cas 1 : sécurisée l'API

security:

- api\_key: []

Figure2 : Exemple de l’opérateur **GET**

Ce code décrit l’opérateur **GET**, sa réponse, la structure du succès (200) d’une requête et default pour les autres, les manières de description de sécurité cas 1 (sécurisée l’API) cas 2 (sécurisée une opération) et comment référencé un type un type défini dans components (paramètre ou autre).

1. **Son utilité** :

La spécification de OpenAPI prépare le terrain pour les consommateurs des services API en mettant à leurs dispositions les outils nécessaires pour l’utilisation de l’API. Elle précise comment formuler une requête prise en charge par l’API, de quoi s’attendre comme réponse d’une requête, quelle sont les différents champs, quelle sont les contraintes de sécurité (méthodes d'authentification) et comment avoir l’accès, ce qui permet aux clients de bénéficier une plus grande clarté dans le processus de conception. En cas de besoin, les informations de contact pertinentes, la licence et les conditions d'utilisation son présents.

1. **Conclusion** :

La spécification de OpenAPI est essentiellement le manuel de référence pour garantir aux développeurs une excellente expérience avec l'API, elle aide les utilisateurs et les développeurs d'API à définir clairement les attentes lors de la consommation et de la construction d'une API. Elle fournit une compréhension globale du comportement d'une API.

Avec ces explications de comment l'API fonctionne et les résultats à attendre lors de l'utilisation de l'API, elle réduise le temps nécessaire pour intégrer les nouveaux consommateurs d'API.

**Références**

[4] Pezoa, Felipe, et al. "Foundations of JSON schema." *Proceedings of the 25th International Conference on World Wide Web*. 2016 p263.

[5] Ben-Kiki, Oren, Clark Evans, and Brian Ingerson. "Yaml ain't markup language (yaml™) version 1.1." *Working Draft 2008-05* 11 (2009) p1.

[6] https://swagger.io/tools/swagger-ui/

1. OpenAPI : <https://swagger.io/specification/>
2. <https://idratherbewriting.com/learnapidoc/>
3. Interface: <https://editor.swagger.io/>