

Explorando OpenApi: estructura, rutas y seguridad

27/02/2024 · **Fernando de la Torre** · Arquitecto de Software

En este artículo, nos adentraremos en la utilización de OpenApi para crear los diferentes *endpoints* de nuestra aplicación, con sus diferentes objetos de *request* y *response* que necesitamos. ¡Vamos allá!

Contenido

[¿Qué es una API?](#)

[¿Qué es OpenApi?](#)

[Características esenciales de OpenAPI](#)

[Estructura](#)

[Rutas](#)

[Respuesta](#)

[Componentes](#)

[La seguridad en openAPI](#)

[Conclusión](#)

¿Qué es una API?

Las **API** (Interfaz de Programación de Aplicaciones) son **piezas de código** que permiten a diferentes componentes de software **comunicarse** entre sí mediante un conjunto de **definiciones y protocolos**.

¿Qué es OpenApi?

OpenAPI es un **conjunto de reglas y protocolos** que permiten a diferentes aplicaciones y sistemas **comunicarse entre sí** de manera estandarizada. Proporciona una forma de acceso a los servicios y datos de una aplicación a través de una interfaz pública.

OpenAPI se basa en **estándares web como HTTP, JSON y REST**, lo que la hace altamente compatible y fácil de implementar. Permite a los desarrolladores crear aplicaciones que se integren con otras plataformas y servicios, lo que fomenta la interoperabilidad y la colaboración entre diferentes sistemas.

Al utilizar OpenAPI, los desarrolladores pueden acceder a funcionalidades específicas de una aplicación o servicio, como la obtención de datos, la realización de operaciones o la interacción con usuarios. Esto facilita la creación de aplicaciones más completas y personalizadas, ya que se pueden combinar diferentes servicios y aprovechar las capacidades de cada uno.

Características esenciales de OpenAPI

Las características más importantes de OpenAPI son las siguientes:

- Ayuda a establecer un buen [diseño de las APIs](#).
- Documentación completa.
- **Testing más rápido** gracias a la generación de un *sandbox*.
- Generación de un **portal de documentación** que describe la API.

Estructura

En primer lugar, el **objeto OpenAPI**, el cual constituye la raíz del fichero. Este objeto incluye a su vez **6 propiedades** fundamentales:

- openapi
- info
- servers
- paths
- security
- components

```
#Estructura del objeto OpenAPI
openapi: 3.1.0
info:
  version: '1.0.0'
  title: Profile REST API
servers:
  - url: 'localhost:3000'
paths:
  ...
  ...
```

```

components:
  ...
  ...
security:
  ...

```

La propiedad *openapi* permite establecer la versión de OpenAPI con la que se construye el documento.

Rutas

La **propiedad *paths*** es una parte vital del documento. En ella **se definen cada una de las rutas** relativas que conforman la API, las cuales se concatenan a la ruta base definida en la propiedad *server* para construir la ruta absoluta.

Supongamos que queremos crear un documento de especificación para una API que se compone de las siguientes operaciones:

Operación	Ruta	Funcionalidad
post	/user	Crear un nuevo usuario
get	/user/:id	Devolver los datos del usuario
delete	/user/:id	Eliminar un usuario

En este ejemplo, tenemos dos rutas fundamentales a incluir en el fichero de especificación: la ruta **/user** y la ruta **/user/:id**.

Ten en cuenta que cada ruta se especifica a través de un objeto cuyas propiedades incluyen los verbos asociados a las distintas operaciones: *get*, *post*, *update* y *delete*, entre otros. Por lo tanto, dentro de cada una de estas propiedades de operación podrás detallar los encabezados, cuerpos de petición, tipos de respuesta y otros elementos de cada operación que incorpore la ruta.

Así mismo, los **parámetros de una ruta**, ya sean de tipo “path” o “query”, deben ser **especificados en la propiedad *parameters*** de este mismo objeto en forma de colección, siendo posible describir su esquema de datos en esta sección del documento o en la propiedad *components* tal y como veremos más adelante.

Observa cómo quedaría el archivo de especificación para las operaciones descritas en la tabla y cómo debes definir el parámetro “id” de tipo “path”.

```

paths:
  /user:
    post:
      summary: crear nuevo usuario
      operationId: createUser
      responses:
        ...
        ...
  /user/{id}:
    parameters:
      - name: id
        in: path
        description: valor del id del usuario
        required: true
        schema:

```

```

        type: integer
        format: int32
# Primera operación de la ruta '/user/:id'
get:
  summary: recupera detalles de un usuario
  operationId: getUser
  responses:
    ...
    ...
# Segunda operación de la ruta '/user/:id'
delete:
  summary: elimina un usuario
  operationId: deleteUser
  responses:
    ...
    ...

```

Para las operaciones que requieren enviar información en el cuerpo de la petición, es necesario especificar el **esquema de datos**. Siguiendo con esta sencilla API de ejemplo, para la operación «Crear un usuario» resulta necesario enviar al servidor los datos de dicho usuario, como mínimo su «nombre», «email» y «contraseña». Veamos entonces cómo definir el cuerpo de la solicitud para la operación ***post(user/:id)***:

```

...
paths:
  /user:
    post:
      summary: crear nuevo usuario
      operationId: createUser
      requestBody:
        schema:
          # Objeto cuerpo de la petición
          type: object
          required:
            - email
            - name
            - password
          properties:
            email:
              type: string
              format: email
            name:
              type: string
            password:
              type: string
      responses:
        ...
        ...
  /user/{id}:
    ...

```

Resulta importante destacar que los tipos de datos y formatos que se declaran en la especificación de una estructura, esto es una de las actualizaciones de la versión más reciente del estándar OpenAPI, lo cual difiere de versiones anteriores.

Respuesta

Como mínimo, para cada operación **debe incluirse una respuesta exitosa y una respuesta de error**. Además, si se devuelve información en el cuerpo de la respuesta, tenemos que especificar el tipo de contenido de la misma, como por ejemplo *application/json*, *text/plain*, *multipart/form-data*, etc.

```

...
paths:
  /user:
    post:
      summary: crear nuevo usuario
      operationId: createUser
      requestBody:
        ...
      responses:
        # Respuesta exitosa
        '201':
          description: usuario creado correctamente
          content:
            application/json:
              schema:
                type: object
                required:
                  - id
                  - name
                properties:
                  id:
                    type: integer
                    format: int64
                  name:
                    type: string
        # Respuesta de error
        '400':
          description: Error al crear el usuario Bad Request
          content:
            application/json:
              schema:
                type: object
                required:
                  - code
                  - message
                properties:
                  code:
                    type: integer
                    format: int32
                  message:
                    type: string
        # Respuesta negativa
        '500':
          description: Error Interno del Servidor
          content:
            application/json:
              schema:
                type: object
                required:
                  - code
                  - message
                properties:
                  code:
                    type: integer
                    format: int32
                  message:
                    type: string
  /user/{id}:
    ...

```

Componentes

En la propiedad *components*, puedes definir todos los esquemas de datos, parámetros, encabezados, cuerpos de petición, ejemplos, mecanismos de seguridad, entre otros elementos que se utilizan a lo largo del documento de especificación.

Veamos cómo especificar diferentes elementos en la propiedad *components*:

```
components:
  parameters:
    id:
      name: id
      in: path
      description: Id del usuario
      required: true
      schema:
        type: integer
        format: int32
  schemas:
    NewUser:
      type: object
      required:
        - email
        - name
        - password
      properties:
        email:
          type: string
          format: email
        name:
          type: string
        password:
          type: string
    User:
      type: object
      required:
        - id
        - name
      properties:
        id:
          type: integer
          format: int64
        name:
          type: string
    Error:
      type: object
      required:
        - code
        - message
      properties:
        code:
          type: integer
format: int32
message:
  type: string
securitySchemes:
  BearerToken:
    type: http
    scheme: bearer
    bearerFormat: JWT
```

Cada declaración de un esquema en particular **debe ser iniciado con un nombre descriptivo** y único.

Componente: **\$ref**

Con **\$ref** puedes hacer referencia a la **definición de un elemento** que se encuentra alojado dentro o fuera del documento principal de la especificación con solo utilizar las declaraciones “local”, “remoto” o “externo”.

La sintaxis para referenciar un componente según su alcance:

local	\$ref: ‘#/components/definitions/myElement’
-------	--

local	\$ref: '#/components/definitions/myElement'
remoto	\$ref: 'documentOrigin/documentName#/myElement'
externo	\$ref: 'baseURL/documentName#/myElement'

La seguridad en openAPI

Por lo general, la mayoría de los **servicios API requieren de uno o varios mecanismos de seguridad** para al menos una operación. Por tanto, es vital incluir la propiedad *security* del objeto OpenAPI en tu documento y comprender su correcta utilización.

La propiedad *security* permite definir una colección que incluye diferentes mecanismos de seguridad (HTTP authentication, API key, mutual TLS, OAuth2, OpenID Connect Discovery) con alcance global para todas las operaciones.

```

openapi: 3.1.0
...
security:
  - BearerToken: []
paths:
  /user:
    post:
      summary: crear nuevo usuario
      operationId: createUser
      security: []
      requestBody:
        ...
      responses:
        '201':
          ...
        '400':
          ...
        '500':
          ...
        ...
    ...
  ...
  ...

```

Es importante destacar que estos mecanismos deben ser previamente definidos en *components* y que, adicionalmente, cada operación incluye una propiedad *security* para redefinir la colección de mecanismos. De esta forma, si necesitas anular o modificar el alcance de uno de ellos en una operación particular, solo debes redefinir su colección.

Conclusión

OpenAPI es una herramienta poderosa para la integración de sistemas y la creación de aplicaciones más flexibles y escalables. Permite a los desarrolladores aprovechar las funcionalidades de diferentes servicios y plataformas, lo que impulsa la innovación y la creación de soluciones más completas.

Además, podemos consultar todas sus especificaciones en: <https://spec.openapis.org/oas/v3.1.0#openapi-specification>.