open api 3.0 spec

Summary of the open api 3.0 specification.

Recall

- What is the OpenAPI spec?
- What is the basic structure?

Overview

OpenAPI Specification (formerly known as Swagger Specification) is an open-source format for describing and documenting APIs. It is a de-facto standard for designing and describing RESTful APIs, and is used by millions of developers and organizations for developing their APIs, be it internal or client facing.

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Sample API
  description: A sample API to illustrate OpenAPI concepts
paths:
  /list:
    get:
      description: Returns a list of stuff
      responses:
        '200':
          description: Successful response
```

OpenAPI 3.0 Tutorial

An integrated API design and documentation platform, SwaggerHub is built for teams to drive consistency and discipline across the API development workflow.

https://support.smartbear.com/swaggerhub/docs/tutorials/openapi-3-tutorial.html

Creating an API

An API defined using the OpenAPI Specification can be divided into 3 main sections –

- Meta information
- Path items (endpoints):
  - Parameters
  - Request bodies
  - Responses
- Reusable components:
  - Schemas (data models)
  - Parameters
  - Responses
  - Other components

In this example, a basic API for a record label wil be used. Let's assume that the record label has a database of artists with the following information:

- Artist name
- Artist genre
- Number of albums published under the label
- Artist username

The API will let consumers obtain the list of artists stored in the database and add a new artist to the database.

Meta Information

A simple API definition that contains just meta information, such as the API title, version, server URL and other descriptive information.

Each API definition starts with the version of the OpenAPI Specification that this definition uses. In our example, it is `openapi: 3.0.0`. The `info` object contains the API `title` and `version`, which are required, and an optional `description`.

The `servers` array specifies one or more server URLs for API calls. The API endpoint paths are appended to the server URL. Some APIs have a single server, others may have multiple servers, such as production and sandbox. In our example, the server URL is `https://example.io/v1`.

We also secure the API using Basic authentication, so that only authorized users can consume the API. For more advanced security, see Authentication.

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple Artist API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

# Basic authentication
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []
```

```
paths: {}
```

Path Items

The path items are the endpoints of your API under which you can specify HTTP verbs for manipulating the resources in the desired manner. These endpoints are relative to the server URL, which in our example is `https://example.io/v1`.

In this example the `/artists` endpoint and the GET method for this endpoint will be defined. So, a client will use `GET https://example.io/v1/artists` to get a list of artists.

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []

#  ----- Added lines  ----------------------------------------
paths:
  /artists:
    get:
     description: Returns a list of artists
#  ---- /Added lines  ----------------------------------------
```

Responses

The GET method, under the `artists` endpoint, lets the consumer of the API obtain the details of a list of artists from the https://example.io/v1 database.

Every response would need at least one HTTP status code to describe the kind of responses a consumer is likely to expect. The description gives details on what the responses of the API would be. In our sample code, we have specified 200, which is a

successful client request, while 400 is an unsuccessful request. You can find more information about HTTP status codes here. A successful response will return the artist name, genre, username and albums recorded. An unsuccessful request is described under the 400 HTTP code, with a corresponding error message detailing why the response is invalid.

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []

paths:
  /artists:
    get:
      description: Returns a list of artists
      #  ----- Added lines  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - username
                  properties:
                    artist_name:
                      type: string
                    artist_genre:
                      type: string
                    albums_recorded:
                      type: integer
                    username:
                      type: string

        '400':
```

```
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
      #  ---- /Added lines  ---------------------------------------
```

Parameters

RESTful parameters specify the variable part of the resource a user works with.

Query Parameters

These are the most common type of parameters. They appear at the end of a URL following a question mark. Query parameters are optional and non unique, so they can be specified multiple times in the URL. It is a non-hierarchical component of the URL.

In the example, it would make sense to let the client limit the information required instead of returning the entire list of artists from the database, which would lead to unnecessary load on the server. The client could describe the page number (offset) and the amount of information on the page (limit), for example:

```
GET https://example.io/v1/artists?limit=20&offset=3
```

These variables are defined under the `parameters` object in the OpenAPI definition. Thus, the specification would now look as follows –

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

components:
  securitySchemes:
    BasicAuth:
      type: http
```

```
      scheme: basic
security:
  - BasicAuth: []

paths:
  /artists:
    get:
      description: Returns a list of artists
      #  ----- Added lines  ----------------------------------------
      parameters:
        - name: limit
          in: query
          description: Limits the number of items on a page
          schema:
            type: integer
        - name: offset
          in: query
          description: Specifies the page number of the artists to be displayed
          schema:
            type: integer
      #  ---- /Added lines  ----------------------------------------

      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - username
                  properties:
                    artist_name:
                      type: string
                    artist_genre:
                      type: string
                    albums_recorded:
                      type: integer
                    username:
                      type: string

        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
```

Request Body

POST, PUT and PATCH requests typically contain the request body. The request body is defined by using the `requestBody` object. For this API, let's add the ability for a user to post an artist to our database. This would be under the `/artists` resource.

The API would now look as follows:

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []

paths:
  /artists:
    get:
      description: Returns a list of artists
      parameters:
        - name: limit
          in: query
          description: Limits the number of items on a page
          schema:
            type: integer
        - name: offset
          in: query
          description: Specifies the page number of the artists to be displayed
          schema:
            type: integer
      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - username
```

```
                properties:
                  artist_name:
                    type: string
                  artist_genre:
                    type: string
                  albums_recorded:
                    type: integer
                  username:
                    type: string
      '400':
        description: Invalid request
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string

  #  ----- Added lines  ----------------------------------------
  post:
    description: Lets a user post a new artist
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            required:
              - username
            properties:
              artist_name:
                type: string
              artist_genre:
                type: string
              albums_recorded:
                type: integer
              username:
                type: string

    responses:
      '200':
        description: Successfully created a new artist

      '400':
        description: Invalid request
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
  #  ---- /Added lines  ----------------------------------------
```

Path Parameters

The path parameters can be used to isolate a specific component of the data that the client is working with, for example, `https://example.io/v1/artists/{username}`. Path parameters are part of the hierarchical component of the URL, and are thus stacked sequentially.

Let's create a new endpoint which returns a specific artist's information based on the username provided. The path parameter here would be the `username` of the artist whose info we need. Path parameters (`username` in this case) have to be mandatorily described in the `parameters` object under the method.

Here, we have specified the `username` as a path parameter. The thing to note is that path parameters have to have a true property set to the required parameter, for the spec to be valid.

If you followed through till here, then congratulation! You have just designed a simple API for a record label!

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []

paths:
  /artists:
    get:
      description: Returns a list of artists
      parameters:
        - name: limit
          in: query
          description: Limits the number of items on a page
          schema:
```

```
            type: integer
        - name: offset
          in: query
          description: Specifies the page number of the artists to be displayed
          schema:
            type: integer
      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - username
                  properties:
                    artist_name:
                      type: string
                    artist_genre:
                      type: string
                    albums_recorded:
                      type: integer
                    username:
                      type: string
        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string

    post:
      description: Lets a user post a new artist
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required:
                - username
              properties:
                artist_name:
                  type: string
                artist_genre:
                  type: string
                albums_recorded:
                  type: integer
                username:
```

```
                type: string
    responses:
      '200':
        description: Successfully created a new artist
      '400':
        description: Invalid request
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string

#  ----- Added lines  ----------------------------------------
/artists/{username}:
  get:
    description: Obtain information about an artist from his or her unique username
    parameters:
      - name: username
        in: path
        required: true
        schema:
          type: string

    responses:
      '200':
        description: Successfully returned an artist
        content:
          application/json:
            schema:
              type: object
              properties:
                artist_name:
                  type: string
                artist_genre:
                  type: string
                albums_recorded:
                  type: integer

      '400':
        description: Invalid request
        content:
          application/json:
            schema:
              type: object
              properties:
                message:
                  type: string
#  ---- /Added lines  ----------------------------------------
```

Reusable Components

What we have just described are just 2 endpoints and 3 actions. This was about 130 lines of specification, and the spec will only get longer as the API gets bigger. One of the things you may notice in the spec we have so far is that we have the same Artist schema (artist name, genre, username and albums published) that gets repeated in various 200 and 400 responses. Bigger APIs would involve rewriting and reusing a lot of the same specs, so it would be a tedious task writing a more complex API.

The OpenAPI Specification has a solution – reusable components that can be used across multiple endpoints in the same API. These components are defined in the global `components` section and then are referenced in individual endpoints. The Specification defines various types of reusable components:

- Schemas (data models)
- Parameters
- Request bodies
- Responses
- Response headers
- Examples
- Links
- Callbacks

Schemas

The `schemas` subsection of the global `components` section can contain various data models consumed and returned by the API. Here is how we can use `components` to store the schema for an HTTP 200 OK response. Our API definition already had the `components` section containing `securitySchemes`, now we have moved `components` to the bottom and added the `schemas` subsection.

The spec is not only shorter, but anytime a new endpoint with the same schema is needed, the designer does not need to spend time writing the piece. See here for more information about `components`.

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

security:
  - BasicAuth: []

paths:
  /artists:
    get:
      description: Returns a list of artists
      parameters:
        - name: limit
          in: query
          description: Limits the number of items on a page
          schema:
            type: integer
        - name: offset
          in: query
          description: Specifies the page number of the artists to be displayed
          schema:
            type: integer
      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  ----------------------------------------
                  $ref: '#/components/schemas/Artist'
                  #  ---- /Added line  ----------------------------------------
        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string

    post:
      description: Lets a user post a new artist
      requestBody:
        required: true
```

```
        content:
          application/json:
            schema:
              #  ----- Added line  ----------------------------------------
              $ref: '#/components/schemas/Artist'
              #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully created a new artist
        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string

  /artists/{username}:
    get:
      description: Obtain information about an artist from his or her unique username
      parameters:
        - name: username
          in: path
          required: true
          schema:
            type: string

      responses:
        '200':
          description: Successfully returned an artist
          content:
            application/json:
              schema:
                type: object
                properties:
                  artist_name:
                    type: string
                  artist_genre:
                    type: string
                  albums_recorded:
                    type: integer

        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
```

```
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic

  #  ----- Added lines  ----------------------------------------
  schemas:
    Artist:
      type: object
      required:
        - username
      properties:
        artist_name:
          type: string
        artist_genre:
            type: string
        albums_recorded:
            type: integer
        username:
            type: string
  #  ---- /Added lines  ----------------------------------------
```

Parameters and Responses

The `components` section also has subsections for storing reusable parameters and responses. In the spec below, we define the reusable query parameters `offset` and `limit` and then reference them in the `/artists` endpoint. We also define a reusable `400Error` response, which we then reference from all the endpoints.

To jump to a definition, simply click the `$ref` link.

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

security:
  - BasicAuth: []

paths:
  /artists:
    get:
      description: Returns a list of artists
      parameters:
```

```
        #  ----- Added line  ----------------------------------------
        - $ref: '#/components/parameters/PageLimit'
        - $ref: '#/components/parameters/PageOffset'
        #  ---- /Added line  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  #  ----- Added line  --------------------------------
                  $ref: '#/components/schemas/Artist'
                  #  ---- /Added line  --------------------------------
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400Error'
          #  ---- /Added line  ----------------------------------------

    post:
      description: Lets a user post a new artist
      requestBody:
        required: true
        content:
          application/json:
            schema:
              #  ----- Added line  ------------------------------------
              $ref: '#/components/schemas/Artist'
              #  ---- /Added line  ------------------------------------
      responses:
        '200':
          description: Successfully created a new artist
        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400Error'
          #  ---- /Added line  ----------------------------------------

  /artists/{username}:
    get:
      description: Obtain information about an artist from his or her unique username
      parameters:
        - name: username
          in: path
          required: true
          schema:
            type: string

      responses:
        '200':
          description: Successfully returned an artist
          content:
            application/json:
              schema:
```

```
                type: object
                properties:
                  artist_name:
                    type: string
                  artist_genre:
                    type: string
                  albums_recorded:
                    type: integer

        '400':
          #  ----- Added line  ----------------------------------------
          $ref: '#/components/responses/400Error'
          #  ---- /Added line  ----------------------------------------

components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic

  schemas:
    Artist:
      type: object
      required:
        - username
      properties:
        artist_name:
          type: string
        artist_genre:
            type: string
        albums_recorded:
            type: integer
        username:
            type: string

  #  ----- Added lines  ----------------------------------------
  parameters:
    PageLimit:
      name: limit
      in: query
      description: Limits the number of items on a page
      schema:
        type: integer

    PageOffset:
      name: offset
      in: query
      description: Specifies the page number of the artists to be displayed
      schema:
        type: integer

  responses:
    400Error:
      description: Invalid request
```

```
      content:
        application/json:
          schema:
            type: object
            properties:
              message:
                type: string
  #  ---- /Added lines  ----------------------------------------
```