OpenAPI

OpenAPI Specification also known as Swagger Specification is an API description format for REST APIs.

An OpenAPI file allows you to describe your entire API, including:

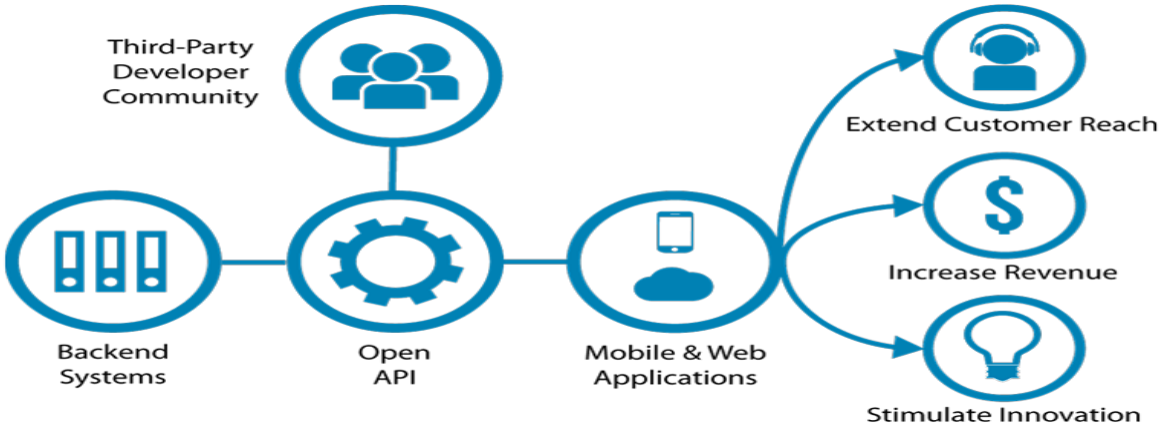
- Available endpoints and operations on each endpoint (GET, POST)
- Operation parameters Input and output for each operation
- Authentication methods
- Contact information, license, terms of use.

These API specifications can be written in YAML or JSON formats. These formats are interchangeable and include the same elements.

There are three primary areas in every OpenAPI document:

- Endpoints (i.e. paths appended to the server URL) and the HTTP methods they support. For each method, any parameters that may or must be included in the request and the response formats for the possible HTTP response codes are specified.
- Reusable components that can be used across multiple endpoints in the API, such as common request parameters and response formats.
- Meta information, including the title, version, and description of the API, authentication method, and location of the API servers.

Swagger is a tool that is used to implement Open API specification.It is a set of open-source tools built around the OpenAPI Specification that can help you design, build, document and consume REST APIs.



The major Swagger tools include:

- **Swagger Editor** – browser-based editor where you can write OpenAPI definitions.
- **Swagger UI** – renders OpenAPI definitions as interactive documentation.
- **Swagger Codegen** – generates server stubs and client libraries from an OpenAPI definition.
- **Swagger Editor Next (beta**) – browser-based editor where you can write and review OpenAPI and AsyncAPI definitions.
- **Swagger Core** – Java-related libraries for creating, consuming, and working with OpenAPI definitions.
- **Swagger Parser** – standalone library for parsing OpenAPI definitions
- **Swagger APIDom** – provides a single, unifying structure for describing APIs across various description languages and serialization formats.

Basic Structure

You can write OpenAPI definitions in YAML or JSON. Below is Sample OpenAPI 3.0 definition written in YAML looks like:

```
openapi: 3.0.0

info:

  title: Sample API

  description: Optional multiline or single-line description in
[CommonMark](http://commonmark.org/help/) or HTML.

  version: 0.1.9

servers:

  - url: http://api.example.com/v1

    description: Optional server description, e.g. Main (production) server

  - url: http://staging-api.example.com

    description: Optional server description, e.g. Internal staging server for testing
```

```
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML.
      responses:
        '200':    # status code
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

Note: All keywords are case sensitive.

Base URL:

```
https://api.example.com/v1/users?role=admin&status=active
\_____________________/\___/ \_____________________/
         server URL       endpoint    query parameters
                            path
```

Server Templating:

Any part of the server URL – scheme, host name or its parts, port, subpath – can be parameterized using variables. Variables are indicated by {curly braces} in the server url, like so:

```
servers:
  - url: https://{customerId}.saas-app.com:{port}/v2
    variables:
      customerId:
        default: demo
        description: Customer ID assigned by the service provider
      port:
        enum:
          - '443'
  - '8443'
        default: '443'
```

Query String in Paths

Query string parameters **must not** be included in paths. They should be defined as query parameters instead.

```yaml
paths:
  /users:
    get:
      parameters:
        - in: query
          name: role
          schema:
            type: string
            enum: [user, poweruser, admin]
          required: true
```

Path Template:

```yaml
paths:
  /users/{id}:
    get:
      tags:
        - Users
      summary: Gets a user by ID.
      description: >
        A detailed description of the operation.
        Use markdown for rich text representation,
        such as **bold**, *italic*, and [links](https://swagger.io).
      operationId: getUserById
      parameters:
        - name: id
          in: path
          description: User ID
          required: true
          schema:
            type: integer
```

Parameter Types:

OpenAPI 3.0 distinguishes between the following parameter types based on the parameter location. The location is determined by the parameter's in key, for example, in: query or in: path.

- path parameters, such as /users/{id}
- query parameters, such as /users?role=admin
- header parameters, such as X-MyHeader: Value
- cookie parameters, which are passed in the Cookie header, such as Cookie: debug=0; csrftoken=BUSe35dohU3O1MZvDCU

For example, the /users/{id} endpoint would be described as:

```yaml
paths:
  /users/{id}:
    get:
      parameters:
        - in: path
          name: id   # Note the name is the same as in the path
          required: true
          schema:
            type: integer
            minimum: 1
          description: The user ID
```

```
1. GET /notes?offset=100&limit=50
```

```yaml
      parameters:
        - in: query
          name: offset
          schema:
            type: integer
          description: The number of items to skip before starting to collect the result set
        - in: query
          name: limit
          schema:
            type: integer
          description: The numbers of items to return
```

Reserved Characters in Query Parameters:

RFC 3986 defines a set of reserved characters :/?#[]@!$&'()*+,;= that are used as URI component delimiters. When these characters need to be used literally in a query parameter value, they are usually percent-encoded. For example, / is encoded as %2F (or %2f), so that the parameter value quotes/h2g2.txt would be sent as

```
1. GET /file?path=quotes%2Fh2g2.txt
```

If you want a query parameter that is not percent-encoded, add allowReserved: true to the parameter definition:

```
      parameters:
        - in: query
          name: path
          required: true
          schema:
            type: string
          allowReserved: true    # <-----
```

In this case, the parameter value would be sent like so:

```
GET /file?path=quotes/h2g2.txt
```

Header Parameters:

An API call may require that custom headers be sent with an HTTP request. OpenAPI lets you define custom request headers as in: header parameters. For example, suppose, a call to GET /ping requires the X-Request-ID header:

```
1. GET /ping HTTP/1.1
2. Host: example.com
3. X-Request-ID: 77e1c83b-7bb0-437b-bc50-a7a58e5660ac
```

```
paths:
  /ping:
    get:
      summary: Checks if the server is alive
      parameters:
        - in: header
          name: X-Request-ID
          schema:
            type: string
            format: uuid
          required: true
```

Cookie Parameters:

Operations can also pass parameters in the Cookie header, as Cookie: name=value. Multiple cookie parameters are sent in the same header, separated by a semicolon and space.

```
1. GET /api/users
```

```
2. Host: example.com
3. Cookie: debug=0; csrftoken=BUSe35dohU3O1MZvDCUOJ
```

Use in: cookie to define cookie parameters:

```yaml
      parameters:
        - in: cookie
          name: debug
          schema:
            type: integer
            enum: [0, 1]
            default: 0
        - in: cookie
          name: csrftoken
          schema:
            type: string
```

Describing Responses:

An API specification needs to specify the responses for all API operations. Each operation must have at least one response defined, usually a successful response. A response is defined by its HTTP status code and the data returned in the response body and/or headers. Here is a minimal example:

```yaml
paths:
  /ping:
    get:
      responses:
        '200':
          description: OK
          content:
            text/plain:
              schema:
                type: string
                example: pong
```

```yaml
paths:
  /users:
    get:
      summary: Get all users
      responses:
        '200':
          description: A list of users
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ArrayOfUsers'
            application/xml:
              schema:
                $ref: '#/components/schemas/ArrayOfUsers'
            text/plain:
              schema:
                type: string
```

```yaml
      responses:
        '200':
          description: OK
        '400':
          description: Bad request. User ID must be an integer and larger than 0.
        '401':
          description: Authorization information is missing or invalid.
        '404':
          description: A user with the specified ID was not found.
        '5XX':
          description: Unexpected error.
```

Response Headers:

```yaml
paths:
  /ping:
    get:
      summary: Checks if the server is alive.
      responses:
        '200':
          description: OK
          headers:
            X-RateLimit-Limit:
              schema:
                type: integer
              description: Request limit per hour.
            X-RateLimit-Remaining:
              schema:
                type: integer
              description: The number of requests left for the time window.
            X-RateLimit-Reset:
              schema:
                type: string
                format: date-time
              description: The UTC date/time at which the current rate limit window resets.
```

Default Response:

Sometimes, an operation can return multiple errors with different HTTP status codes, but all of them have the same response structure:

```yaml
      responses:
        '200':
          description: Success
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/User'

        # These two error responses have the same schema
        '400':
          description: Bad request
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
        '404':
          description: Not found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
```

OpenAPI Info Object:

The object provides metadata about the API.

Field Name	Type	Description
title	string	**REQUIRED**. The title of the API.
description	string	A short description of the API. .
termsOfService	string	A URL to the Terms of Service for the API. MUST be in the format of a URL.
contact	Contact Object	The contact information for the exposed API.
license	License Object	The license information for the exposed API.
version	string	**REQUIRED**. The version of the OpenAPI document (which is distinct from the OpenAPI Specification version or the API implementation version).

Example:

```
{
 "title": "Sample Pet Store App",
 "description": "This is a sample server for a pet store.",
 "termsOfService": "http://example.com/terms/",
 "contact": {
```

```
    "name": "API Support",
    "url": "http://www.example.com/support",
    "email": "support@example.com"
  },
  "license": {
    "name": "Apache 2.0",
    "url": "https://www.apache.org/licenses/LICENSE-2.0.html"
  },
  "version": "1.0.1"
}
```

License Object:

License information for the exposed API.

Fixed Fields

Field Name	Type	Description
name	string	**REQUIRED**. The license name used for the API.
url	string	A URL to the license used for the API. MUST be in the format of a URL.

OpenAPI Servers Object:

An object representing a Server.

Fixed Fields

Field Name	Type	Description
url	string	**REQUIRED**. A URL to the target host. This URL supports Server Variables and MAY be relative, to indicate that the host location is relative to the location where the OpenAPI document is being served. Variable substitutions will be made when a variable is named in {brackets}.
description	string	An optional string describing the host designated by the URL. CommonMark syntax MAY be used for rich text representation.
variables	Map[string, Server Variable Object]	A map between a variable name and its value. The value is used for substitution in the server's URL template.

- OpenAPI Schema and Components

- JSON Schema
- OpenAPI Data Types:

The data type of a schema is defined by the type keyword, for example, type: string. OpenAPI defines the following basic types:

string (this includes dates and files)

number

integer

boolean

array

object

- OpenAPI Objects
- OpenAPI Enums:

  You can use the enum keyword to specify possible values of a request parameter or a model property. For example, the sort parameter in GET /items?sort=[asc|desc] can be described as:

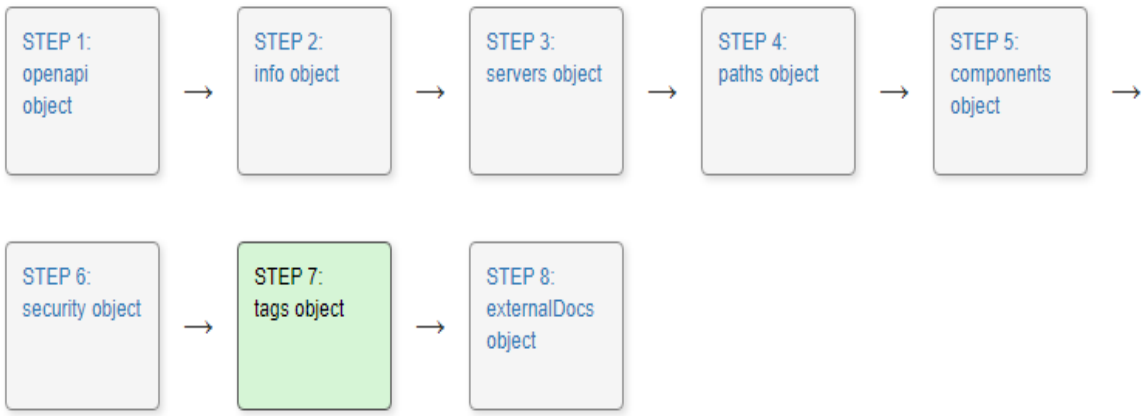
```
paths:
 /items:
   get:
     parameters:
       - in: query
         name: sort
         description: Sort order
```

```
schema:
  type: string
  enum: [asc, desc]
```

Nullable enums:

A nullable enum can be defined as follows:

```
type: string
nullable: true  # <---
enum:
  - asc
  - desc
  - null
```



Document Structure

An OpenAPI document *MAY* be made up of a single document or be divided into multiple, connected parts at the discretion of the author. In the latter case, Reference Objects and Schema Object $ref keywords are used.

It is *RECOMMENDED* that the root OpenAPI document be named: openapi.json or openapi.yaml.