**SWAGGER**

Ok, first going into swagger we have known something about the OPEN API.

**OPEN API:--**

It is an API description format for rest API’s

It will describe

1. Available endpoints and operations on each points(GET,POST,PUT,DELETE)
2. Operations parameters input and output for each point
3. Authentication
4. Contact information, license, terms of use and other information

**SWAGGER:--**

It is an open source

It will built the open api specification that can help you design, built, document and consume rest api’s

1. Swagger Editor – we can write
2. Swagger UI --- interactive api doceumation
3. Swagger codegen --- Generates server stubs and client lib
4. Swagger Inspector --- for testing and auto generating openAPI documentation for any environment.

To describe their own structure.

We can write in both the YML and JSON

**SWAGGER FRAME WORK BENIFITS:**

* Swagger uses a common language that everyone can understand , it easily comprehensible for both developers and non developers
* software developers , product and project managers , business analysts and even potential customers can access api design
* Swagger is easily adjustable and it can be successfully used for api testing bug fixing and same can be used accelerating various API dependent processes

**BASIC STRUCTURE:**

Open API: version

Info:

Title: api name

Description: information about api

Version: arbitrary string – specifies the version of api

Servers:

* url : our url

description: the description of the url

* url:

description:

Paths:

/End point:

End point operation that what we are using:

Summary:

Description:

Responses:

“response number”:

Description: return statement

Content:

Schema:

Type: type of the parameter given

Items:

Type:

**API servers and Basic URL:**

https:// 127.0.01.5000/login/etc-----

https:// 127.0.01.5000 ------ server url

Login --------- end point

Etc --- query parameters

Format:

It follows the RFC 3986 url format

Server url must not include query string parameters

Templating:

* Variable must be indicated in curl brace {} in the server
* In this we will not use Schema
* We will assume it as string

Common uses:

* Specifying multiple protocols
* Hosted applications where each customer has their sub domain
* Regional servers in different geographical regions

**Media Types:**

It is a format of a request or response body data

The format is “RFC 6838”

The most common being JSON, XML and images

To use the same data format for several media types, define a custom object in the components section of your spec and then refer to this object in each media type

**Paths and operations:**

* Global servers can also be overridden on the path level or operation level
* Paths may have an optional short ‘summary’ and longer ‘description’

Templating:

We can use {} to mark parts of an url as path parameters

Operations:

* Supports operation parameters passed via path, query string, headers and cookies
* We can also define the request body for operation that transmit data to the server

Overriding global servers:

This global server arrays can be overridden on the path level or operation level

* Different base url for file upload and download operations
* Deprecated but still functional end points.

**Parameters:**

* You can define the parameter data type, format and other details
* Parameters are defined in the parameters section of an operation or path.
* To describe a parameter, you specify its name, location (in), data type (defined by either schema or content) and other attributes, such as description or required.
* Parameters is an array, so, in YAML, each parameter definition must be listed with a dash (-) in front of it.

Parameter types:

1. [Path parameters](https://swagger.io/docs/specification/describing-parameters/#path-parameters) 🡪 such as /users/{id}
2. [Query parameters](https://swagger.io/docs/specification/describing-parameters/#query-parameters) 🡪such as /users?role=admin
3. [Header parameters](https://swagger.io/docs/specification/describing-parameters/#header-parameters) 🡪such as X-MyHeader: Value
4. [Cookie parameters](https://swagger.io/docs/specification/describing-parameters/#cookie-parameters) 🡪which are passed in the Cookie header, such as Cookie: debug=0; csrftoken=BUSe35dohU3O1MZvDCU

**Request Body:**

Request bodies are typically used with “create” and “update” operations (POST, PUT, and PATCH)

If an operation sends a request body, use the “request body” keyword

It should be written next to the summary

* The request Body is more flexible in that it lets you consume different media types, such as JSON, XML, form data, plain text, and others, and use different schemas for different media types
* request Body consists of the content object, an optional [Markdown](https://commonmark.org/help/)-formatted description, and an optional required flag (false by default)
* **Request bodies are optional by default**

**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
* The HTTP status code must be enclosed in quotes (“”)
* To specify the response media types, use the content keyword at the operation level
* Each response status requires a description

Response Body:

The schema keyword is used to describe the response body. A schema can define:

* + an object or an array — typically used with JSON and XML APIs,
  + a primitive data type such as a number or string – used for plain text responses,
  + a file

**Authencation:**

The “security schemes” and “security” keywords are used to describe the Authencation methods used in your api’s

HTTP authentication schemes (they use the Authorization header):

* [Basic](https://swagger.io/docs/specification/authentication/basic-authentication/)

1. [Basic authentication](https://en.wikipedia.org/wiki/Basic_access_authentication) is a simple authentication scheme built into the HTTP protocol.
2. We send HTTP requests with the Authorization header that contains the word Basic word followed by a space and a base64-encoded string username : password

* [Bearer](https://swagger.io/docs/specification/authentication/bearer-authentication/)

1. Bearer authentication (also called token authentication) is an [HTTP authentication scheme](https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication) that involves security tokens called bearer tokens.
2. The name Bearer authentication can be understood as give access to the bearer of this token.
3. The bearer token is a cryptic string, usually generated by the server in response to a login request.
4. We must send this token in the Authorization header when making requests to protected resources

* other HTTP schemes as defined by [RFC 7235](https://tools.ietf.org/html/rfc7235) and [HTTP Authentication Scheme Registry](https://www.iana.org/assignments/http-authschemes/http-authschemes.xhtml)

[API keys](https://swagger.io/docs/specification/authentication/api-keys/) in headers, query string or cookies

* [Cookie authentication](https://swagger.io/docs/specification/authentication/cookie-authentication/)

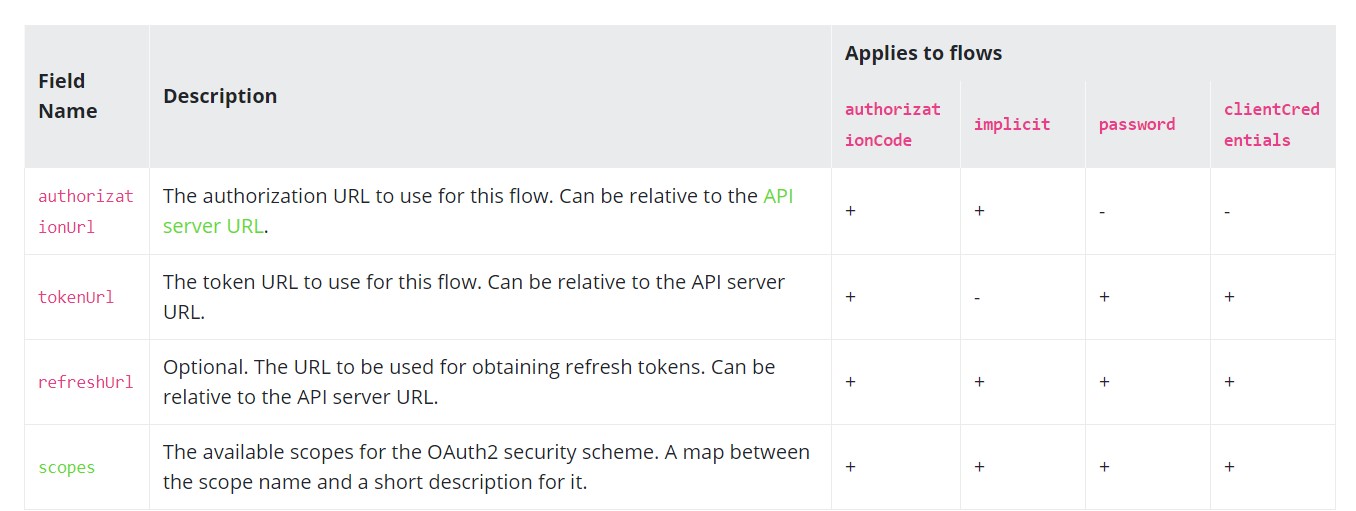
[OAuth 2](https://swagger.io/docs/specification/authentication/oauth2/)

* [OAuth 2.0](https://oauth.net/2/) is an authorization protocol that gives an API client limited access to user data on a web server.
* GitHub, Google, and Facebook APIs notably use it.
* OAuth relies on authentication scenarios called *flows*, which allow the resource owner (user) to share the protected content from the resource server without sharing their credentials

Flows:

The flows (also called grant types) are scenarios an API client performs to get an access token from the authorization server. OAuth 2.0 provides several flows suitable for different types of API clients:

1. Authorization code
2. Implicit
3. Resource owner password credentials (or just password)
4. Client Credentials



[OpenID Connect Discovery](https://swagger.io/docs/specification/authentication/openid-connect-discovery/)

* It is an identity layer built on top of the [OAuth 2.0](https://swagger.io/docs/specification/authentication/oauth2/) protocol and supported by some OAuth 2.0 providers, such as Google and Azure Active Directory.
* It defines a sign-in flow that enables a client application to authenticate a user, and to obtain information about that user, such as the user name, email, and so on.
* User identity information is encoded in a secure JSON Web Token (JWT), called ID token. OpenID Connect defines a discovery mechanism, called [OpenID Connect Discovery](https://openid.net/specs/openid-connect-discovery-1_0.html), where an OpenID server publishes its metadata at a well-known URL, typically

There are 2 steps:

1. Defining security Schemes
2. Applying security

## KEYWORDS

## Supported JSON Schema Keywords:

1. title
2. [pattern](https://swagger.io/docs/specification/data-models/data-types/#pattern)
3. [required](https://swagger.io/docs/specification/data-models/data-types/#required)
4. [enum](https://swagger.io/docs/specification/data-models/enums/)
5. [minimum](https://swagger.io/docs/specification/data-models/data-types/#range)
6. [maximum](https://swagger.io/docs/specification/data-models/data-types/#range)
7. [exclusive Minimum](https://swagger.io/docs/specification/data-models/data-types/#range)
8. [exclusive Maximum](https://swagger.io/docs/specification/data-models/data-types/#range)
9. [multiple Of](https://swagger.io/docs/specification/data-models/data-types/#multipleOf)
10. [minLength](https://swagger.io/docs/specification/data-models/data-types/#string)
11. [maxLength](https://swagger.io/docs/specification/data-models/data-types/#string)
12. [minItems](https://swagger.io/docs/specification/data-models/data-types/#array-length)
13. [maxItems](https://swagger.io/docs/specification/data-models/data-types/#array-length)
14. [uniqueItems](https://swagger.io/docs/specification/data-models/data-types/#uniqueItems)
15. [minProperties](https://swagger.io/docs/specification/data-models/data-types/#property-count)
16. [maxProperties](https://swagger.io/docs/specification/data-models/data-types/#property-count)
17. $schema
18. additionalItems
19. const
20. contains
21. dependencies
22. id,
23. $id
24. patternProperties
25. propertyNames
26. deprecated
27. [discriminator](https://swagger.io/docs/specification/data-models/inheritance-and-polymorphism/)
28. [example](https://swagger.io/docs/specification/adding-examples/)
29. externalDocs
30. [nullable](https://swagger.io/docs/specification/data-models/data-types/#null)
31. [readOnly](https://swagger.io/docs/specification/data-models/data-types/#readonly-writeonly)
32. [writeOnly](https://swagger.io/docs/specification/data-models/data-types/#readonly-writeonly)
33. [xml](https://swagger.io/docs/specification/data-models/representing-xml/)
34. [type](https://swagger.io/docs/specification/data-models/data-types/#type)
35. format
36. description
37. [items](https://swagger.io/docs/specification/data-models/data-types/#array)
38. [properties](https://swagger.io/docs/specification/data-models/data-types/#object)
39. [additionalProperties](https://swagger.io/docs/specification/data-models/data-types/#additionalProperties)
40. [default](https://swagger.io/docs/specification/data-models/data-types/#default)
41. [allOf](https://swagger.io/docs/specification/data-models/oneof-anyof-allof-not/)
42. [oneOf](https://swagger.io/docs/specification/data-models/oneof-anyof-allof-not/)
43. [anyOf](https://swagger.io/docs/specification/data-models/oneof-anyof-allof-not/)
44. [not](https://swagger.io/docs/specification/data-models/oneof-anyof-allof-not/)