**What is Open API?**

* It is public API
* Made publicly available for software developers
* Two major architectures
  + REST API
  + SOAP API
* Open API might be universally accessible

Purpose of Open API

* Reduce dependencies between develop certain application components
* Reduce the time of development
* Easy to integrate in any development tool

**Open API Specification**

* Standard, language-agnostic interface to discover and understand of the service
* It is having
  + Version
  + Format
  + Document Structure
  + Data types
  + Rich Text Formatting
  + Relative References in URLs
  + Schema
  + Open API Object
  + Info Object
  + Contact Object
  + License Object
  + Server Object
  + Path Object
  + Operation Object
  + Parameter Object
  + Request Body Object
  + Responses Object
* Reference Link
  + <https://swagger.io/specification/>

Version:

* Open API is versioned using Semantic Versioning
* Versioning format is MAJOR.MINOR.PATCH
  + MAJOR version when you make incompatible API changes
  + MINOR version when you add functionality in a backwards compatible manner
  + PATCH version when you make backwards compatible bug fixes
* Example: openapi: 3.0.1

Format:

* The OpenAPI format may be represented either in JSON or YAML format
* Example if a field has an array value then it will be represented by
  + {“field”:[1,2,3]}
* All field names in the specification are case sensitive

Document Structure:

* Open API document may be made up of single document or be divided into multiple
* Multiple document can be connected by using the keyword “$ref”

Data Types:

* Data Type in the Open API specification are based on the types supported by JSON schema
* Null is not supported as type
* Primitives have an optional modifier property as “format”
* If email is specified as string type but the format is email

Rich Text Formatting

* Throughout the specification description fields are noted as supporting Common Mark markdown formatting
  + <https://spec.commonmark.org/0.27/#what-is-markdown->
* Rich text must support at a minimum markdown syntax

Relative References in URLs

* All URL properties may be relative references as defined by RFC3986
  + <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>
* It used $ref are processed as per JSON Reference, Using the URL of the current document as the base URI

OpenAPI Object

* The openAPI object having the following property
  + Openapi – String
  + Info – Info Object
  + Servers – Servers Object
  + Paths – Paths Object
  + Components – Components Object
  + Security – Security Requirement Object
  + Tags – Tags Object
  + externalDocs – External Documentation Object

Info Object

* The object provides metadata about the API
* It’s having the following property
  + Title – String
  + Description – String
  + termsOfService – String
  + Contact – Contact Object
  + License – License Object
  + Version – string

Contact Object

* Contact Information for the exposed API
* This object having the following property
  + Name – String
  + URL – String
  + Email – String

License Object

* License information for the exposed API
* This object having the following property
  + Name – String
  + URL - String

{

"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"

}

Server Object:

* This object represents the server details
* Adding multiple server details in the single object
* Server object having the following fields
  + Name – String
  + URL – String

{

"servers": [

{

"url": "https://development.gigantic-server.com/v1",

"description": "Development server"

},

{

"url": "https://staging.gigantic-server.com/v1",

"description": "Staging server"

},

{

"url": "https://api.gigantic-server.com/v1",

"description": "Production server"

}

]

}

Path Object:

* It holds the relative paths to the individual endpoints and their operations
* The path is appended to the URL from the server object in order to construct the full URL
* The path object having the following fields
  + $ref – String
  + Summary – String
  + Description – String
  + Get / Put / post / delete / options / head / patch / trace – Operation Object

Operation Object:

* It describes a single API operations on a path
* It having the following fields
  + Tag – string
  + Summary – string
  + Description – string
  + operationId – string
  + parameters – Parameter Object
  + requestBody – Request body object
  + responses – Responses Object
  + callbacks – Callback object

Parameter Object:

* It describe the single operation parameter
* It having the following fields
  + Name – string
  + In – string
  + Description – string
  + Required – Boolean
  + Deprecated – Boolean
  + allowEmptyValue – Boolean
  + style – string
  + schema – schema object

Request Body Object:

* It describe the single request body
* It having the following fields
  + Description – string
  + Content – Media type object
  + Required – Boolean

Responses Object:

* It is having the expected responses of an operation
* This is maps a HTTP response code to the expected response
* This is having the following fields
  + Description – string
  + Headers – Header object
  + Content – Media type object
  + Link – Reference object

In this spectral linter documentation we are using YAML and JSON Open API for testing spectral linter

Before we are move into the open API example we look in to what is YAML and JSON

YAML

* It stands for “yet another markup language”
* It is a data serialization language
* It is often used in configuration files
* Reference Link
  + https://yaml.org/

YAML Syntax

* YAML uses Python style indication to indicate nesting
* Tab character not allowed so whitespace used instead
* Usual format symbols not used such as braces, square brackets, closing tags, quotation marks
* Extension - .yaml, .yml
* The new list are starting from ---
* Multi line strings are written by using the character > or |

YAML Example

---

-Project: spectral

title: analysis

description: |

Wrapped text

Will be folded

JSON

* Java Script Object Notation
* It is a lightweight format for storing and transporting data
* Reference Link
  + https://en.wikipedia.org/wiki/JSON

JSON Syntax

* Data is in name value pairs
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays
* *{*

*“key”:”value”*

*}*

JSON Example

*{*

*“project”:”spectral”,*

*“description”:”Analysis spectral”*

*}*

**Example of Open API**

In YAML File

openapi: 3.0.0

servers:

# Added by API Auto Mocking Plugin

- description: SwaggerHub API Auto Mocking

url: https://virtserver.swaggerhub.com/self51142/API1/1.0.0

info:

description: This is a simple API

version: "1.0.0"

title: Simple Inventory API

contact:

email: you@your-company.com

license:

name: Apache 2.0

url: 'http://www.apache.org/licenses/LICENSE-2.0.html'

tags:

- name: admins

description: Secured Admin-only calls

- name: developers

description: Operations available to regular developers

paths:

/inventory:

get:

tags:

- developers

summary: searches inventory

operationId: searchInventory

description: |

By passing in the appropriate options, you can search for

available inventory in the system

parameters:

- in: query

name: searchString

description: pass an optional search string for looking up inventory

required: false

schema:

type: string

- in: query

name: skip

description: number of records to skip for pagination

schema:

type: integer

format: int32

minimum: 0

- in: query

name: limit

description: maximum number of records to return

schema:

type: integer

format: int32

minimum: 0

maximum: 50

responses:

'200':

description: search results matching criteria

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/InventoryItem'

'400':

description: bad input parameter

post:

tags:

- admins

summary: adds an inventory item

operationId: addInventory

description: Adds an item to the system

responses:

'201':

description: item created

'400':

description: 'invalid input, object invalid'

'409':

description: an existing item already exists

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/InventoryItem'

description: Inventory item to add

components:

schemas:

InventoryItem:

type: object

required:

- id

- name

- manufacturer

- releaseDate

properties:

id:

type: string

format: uuid

example: d290f1ee-6c54-4b01-90e6-d701748f0851

name:

type: string

example: Widget Adapter

releaseDate:

type: string

format: date-time

example: '2016-08-29T09:12:33.001Z'

manufacturer:

$ref: '#/components/schemas/Manufacturer'

Manufacturer:

required:

- name

properties:

name:

type: string

example: ACME Corporation

homePage:

type: string

format: url

example: 'https://www.acme-corp.com'

phone:

type: string

example: 408-867-5309

type: object

In JSON Format

{

"openapi": "3.0.0",

"servers": [

{

"description": "SwaggerHub API Auto Mocking",

"url": "https://virtserver.swaggerhub.com/self51142/API1/1.0.0"

}

],

"info": {

"description": "This is a simple API",

"version": "1.0.0",

"title": "Simple Inventory API",

"contact": {

"email": "you@your-company.com",

"name": "chennai"

},

"license": {

"name": "Apache 2.0",

"url": "http://www.apache.org/licenses/LICENSE-2.0.html"

}

},

"tags": [

{

"name": "admins",

"description": "Secured Admin-only calls"

},

{

"name": "developers",

"description": "Operations available to regular developers"

}

],

"paths": {

"/inventory": {

"get": {

"tags": [

"developers"

],

"summary": "searches inventory",

"operationId": "searchInventory",

"description": "By passing in the appropriate options, you can search for\navailable inventory in the system\n",

"parameters": [

{

"in": "query",

"name": "searchString",

"description": "pass an optional search string for looking up inventory",

"required": false,

"schema": {

"type": "string"

}

},

{

"in": "query",

"name": "skip",

"description": "number of records to skip for pagination",

"schema": {

"type": "integer",

"format": "int32",

"minimum": 0

}

},

{

"in": "query",

"name": "limit",

"description": "maximum number of records to return",

"schema": {

"type": "integer",

"format": "int32",

"minimum": 0,

"maximum": 50

}

}

],

"responses": {

"200": {

"description": "search results matching criteria",

"content": {

"application/json": {

"schema": {

"type": "array",

"items": {

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

}

}

}

}

},

"400": {

"description": "bad input parameter"

}

}

},

"post": {

"tags": [

"admins"

],

"summary": "adds an inventory item",

"operationId": "addInventory",

"description": "Adds an item to the system",

"responses": {

"201": {

"description": "item created"

},

"400": {

"description": "invalid input, object invalid"

},

"409": {

"description": "an existing item already exists"

}

},

"requestBody": {

"content": {

"application/json": {

"schema": {

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

}

}

},

"description": "Inventory item to add"

}

}

}

},

"components": {

"schemas": {

"InventoryItem": {

"type": "object",

"required": [

"id",

"name",

"manufacturer",

"releaseDate"

],

"properties": {

"id": {

"type": "string",

"format": "uuid",

"example": "d290f1ee-6c54-4b01-90e6-d701748f0851"

},

"name": {

"type": "string",

"example": "Widget Adapter"

},

"releaseDate": {

"type": "string",

"format": "date-time",

"example": "2016-08-29T09:12:33.001Z"

},

"manufacturer": {

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

}

}

},

"Manufacturer": {

"required": [

"name"

],

"properties": {

"name": {

"type": "string",

"example": "ACME Corporation"

},

"homePage": {

"type": "string",

"format": "url",

"example": "https://www.acme-corp.com"

},

"phone": {

"type": "string",

"example": "408-867-5309"

}

},

"type": "object"

}

}

}

}

Example Explained

* Openapi
  + It is a keywords that indicates the version of the open api document
* Servers
  + It is an array
  + It specify one or more base URLs for your API
  + Reference Link
    - <https://swagger.io/docs/specification/api-host-and-base-path/>
* Info
  + It is having the general information about the version number, license notes, contact data, title
  + Title and version properties are required other options are optional
  + Reference Link
    - <https://swagger.io/docs/specification/api-general-info/>
* Tags
  + It is used to group the displayed operation
  + Tag name must be unique in the sense you cannot have two tags with the same name
  + Reference Link
    - https://swagger.io/docs/specification/grouping-operations-with-tags/
* Paths
  + It is end points that your API exposes and operations are the HTTP methods used to manipulate these paths such as GET, POST, DELETE
  + Path is relative to the server-url, the full URL is constructed as <server-url>/path
  + Reference Link
    - <https://swagger.io/docs/specification/paths-and-operations/>
* Summary
  + It is a short one or two sentence explanation of what the purpose of this path
* operationId
  + Each operation may specify a unique operationId
  + Some code generator use this value to name the corresponding methods in code
  + Reference Link
    - <https://swagger.io/docs/specification/2-0/paths-and-operations/>
* Description
  + It is the details explanation of the paths operation
  + It may contains two or more lines of words
* Parameters
  + It defines the parameters used in the path
  + It is an array should have all the parameters details used in the path
  + It is describe in 4 types
    - Path Parameter
    - Query Parameter
    - Header Parameter
    - Cookie Parameter
      * Path Parameters
        + Variable parts of the URL path
        + /users/{id}

Here {id} is the parameter

* + - * + It is mentioned in parameters tag as

-in: path

Name: id

Required: true

Schema:

type: integer

* + - * Query Parameter
        + It appears at the end of the request URL after the question mark(?) with different name=value and separated by ampersands(&)
        + /users/?limit = 10

Here limit = 10 as query parameter

* + - * + It is mentioned in parameter tag as

-in: query

Name: limit

Required: false

Schema:

Type: integer

* + - * Header Parameter
        + Suppose the HTTP API require custom header then OpenAPI support the custom header to be added in the parameter
        + GET / ping HTTP 1.1

Host: example.com

X-Request-ID: 77e1c83b-77e1c83b-77e1c83b-77e1c83b

* + - * + It is mentioned in parameter tag as

-in: header

Name: X-request-ID

Required: true

Schema

Type: string

Format: UUID

* + Reference Link
    - <https://swagger.io/docs/specification/describing-parameters/>
* Request Body
  + It is used in CREATE and UPDATE operations
  + The datatype is mentioned by using “type” keyword
  + In default the required keyword is true and its having the list of data fields to be used in the request body
  + Each field is describe in “properties” keyword
  + It is more flexible and reusable for example one schema is used by multiple request body
* Responses
  + API specification needs to specify “response” for all API operations
  + Every operation must have at least one response code such as success
  + A response is defined by its HTTP status code and each response code indicates the API behaviors
  + Response body is describe by using the keyword “schema”
  + The “content” keywords having which type of data we are getting the successful response
  + https://swagger.io/docs/specification/describing-responses/
* Schema
  + “schema” keyword is used to describe the response body
  + “object” or “array” typically used in JSON and XML API
  + Inside schema object the data type of response is mentioned as “type” keyword
  + The data field is mentioned inside the “properties” keyword it also contains the datatype of field and its description
  + If you want to define the schema in globally then use the keyword “ref” and its having the location of the schema which is present
  + The global schema is present in “components.schemas”, and it is useful for multiple media type use the same schema
  + Reference Link
    - <https://swagger.io/docs/specification/describing-responses/>

**Spectral Linter**

* It is open source JSON / YAML linter
* Create style guides for your OpenAPI / AsyncAPI
* Reference Link
  + https://stoplight.io/open-source/spectral

Purpose of Spectral

* Spectral Linter is used to making sure that APIs are not just technically correct
* Spectral linter allows you to create style guides for your structured data

Spectral License

* Spectral is 100% free and open source, under Apache License 2.0
* Reference Link
  + https://github.com/stoplightio/spectral/blob/develop/LICENSE

**How to install spectral?**

In Windows

There are two ways to install spectral

* Using NPM
* Using Executable binaries

Using NPM

Local Installation

* Run the below command in cmd to install the Spectral in local directory

npm install @stoplight/spectral

Global Installation

* Run the below command in cmd to install the Spectral in globally, it can be access from anywhere in the system

npm install –g @stoplight/spectral

Reference Link - https://www.npmjs.com/package/@stoplight/spectral/v/3.1.0

Executable binaries

Standalone package for all major platform

* Windows
* Linux
* Mac

<https://github.com/stoplightio/spectral/releases>

The package download from the above URL and proceed with the standard procedure for running CLI tools

It is not automatically updating we need to download the latest version for updating

How to check spectral installed properly or not?

By checking spectral is installed properly in your system for running the following command in cmd

spectral –version

And you have get the spectral version which is one installed in your system

6.3.0

How to lint spectral?

By linting spectral using the following syntax

Spectral lint –ruleset {ruleset\_name} {openapi\_name}

Example

spectral lint --ruleset ruleset.yaml openapi.yaml

Or

spectral lint --ruleset ruleset.yaml openapi.json

Suppose the ruleset and openapi is presented in different repository then the command will be like

spectral lint --ruleset ruleset/ruleset.yaml openapi/openapi.yaml

From the above command we assume ruleset present in the ruleset directory and openapi present in the openapi directory

**Spectral Output**

* Spectral output classified as four types
  + Errors
  + Warning
  + Infos
  + Hints

Example of the output

1:1 warning oas3-api-servers OpenAPI "servers" must be present and non-empty array.

2:10 error oas3-schema Property "servers1" is not expected to be here.

2 problems (1 error, 1 warning, 0 infos, 0 hints)

By the above example the error is mentioned as warning and error, in the last line indicates the total number of error and warning

**Spectral Rulesets**

* It is collection of rules written in JSON / YAML
* Spectral having two rulesets – OpenAPI V2 / V3

Example of Rulesets

rules:

'openapi-tags-alphabetical': {

      description: 'OpenAPI object must have alphabetical "tags".',

      given: '$',

      then: {

        field: 'tags',

        function: alphabetical,

        functionOptions: {

          keyedBy: 'name',

        },

      },

    },

* rules
  + keyword
* openapi-tags-alphabetical
  + Name of the rules
* description
  + Describe the usage of the ruleset or something
* Given
  + part of the documents to apply rules
  + If you are using the functions in ruleset then the function part is present in the given value
  + Ex: $.paths.[\*].[\*].responses
  + $.paths – refer the root
  + [\*] – refer the directory
* then
  + It explains which functions to apply to the given part
  + It may be a core or custom function
* Field
  + It is optional
  + It is applying the function to a specific property in an object
  + It is not given the function should be applied or entire target of the given object
* Function
  + Name of the function
  + May be a core or custom function
* FunctionOptions
  + Input parameter of the function

**Core Functions:**

Spectral having few inbuilt function that can be used to save the effort of writing functions for common tasks

**Alphabetical**

This function is used for arrange the alphabetical content for simple arrays or object by passing a key

Example

By using this function check the response status code is in ascending order

By creating new OpenAPI name as “openapi-alphabetical.yaml” should contains response order in descending order

openapi: 3.0.2

info:

  title: Alphabetical Order

  version: 0.0.0

paths:

  /foo:

    get:

      operationId: get-foo

      responses:

        '404':

          description: ''

        '200':

          description: ''

          content:

            application/json:

              schema:

                $ref: '#/components/schemas/foo'

components:

  schemas:

    foo:

      title: More incorrect casing

      type: object

      properties:

        id:

          type: integer

        bar:

          type: string

And then create the ruleset name as “ruleset-alphabetical.yaml” for the above API to check the response is order

rules:

  response-order:

    message: Responses should be in alphabetical order

    recommended: true

    given: $.paths.\*.\*.responses

    then:

      function: alphabetical

Now run the linter

spectral lint --ruleset rulesets/ruleset-alphabetical-order.yaml examples/openapi-alphabetical-order.yaml

Output

10:15 warning response-order Responses should be in alphabetical order paths./foo.get.responses[404]

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

Now change the response code as alphabetical and run the linter the output looks like

No results with a severity of 'error' or higher found!

**Enumeration**

* This function check the field value exist in this set of possible values
* The function name called as “enumeration”
* Function options having values keywords contains an array of possible values, the value should be number or string

function: enumeration

functionOptions:

values:

- users

- articles

- categories

**Pattern**

* This function having regular expression to check the value match the regular expression
* It having two function option
  + Match
  + notMatch
* Match
  + If provided the value must match this regular expression

function: pattern,

        functionOptions: {

          match: "^[A-Za-z0-9-.\_~:/?#\\[\\]@!\\$&'()\*+,;=]\*$",

        },

* notMatch
  + If provided the value must not match the regular expression

function: pattern,

        functionOptions: {

          notMatch: 'eval\\(',

        },

Example

In this example we have check the description having <script> tag, If it is having the waring throws description must not have <script tag

Ruleset

Rules:

'no-script-tags-in-markdown': {

      description: 'Markdown descriptions must not have "<script>" tags.',

      recommended: true,

      type: 'style',

      given: '$..[description,title]',

      then: {

        function: pattern,

        functionOptions: {

          notMatch: '<script',

        },

      }

Open API

tags:

  - name: admins

    description: Secured Admin-only calls <script>

Output

19:18 warning no-script-tags-in-markdown Markdown descriptions must not have "<script>" tags. tags[1].description

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

**Length**

* It is used for count the length of a string or an array, the number of properties in an object or a numeric value
* It is define by minimum and maximum values
  + Min – The minimum length to match
  + Max – The maximum length to match

function: length

functionOptions:

max: 3

min: 1

Example

In this example we have check how this function is works, by verifying the function add two tags in open API and set the max is 1 in ruleset

Ruleset

rules: {

  'operation-singular-tag': {

      description: 'Operation must not have more than a single tag.',

      recommended: true,

      type: 'style',

      given: "$",

      then: {

        field: 'tags',

        function: length,

        functionOptions: {

          max: 1

        },

      },

    }

}

Open API

tags:

  - name: admins

    description: Secured Admin-only calls

  - name: developers

    description: Operations available to regular developers

Output:

15:6 warning operation-singular-tag Operation must not have more than a single tag. tags

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

**Custom Rulesets:**

How to create custom ruleset?

The custom ruleset is looks like

*rules:*

*my-rule-name:*

*description: “check custom ruleset”.*

*given: $.tags[\*]*

*severity: error*

*then:*

*field: description*

*function: truthy*

* Rules
  + keyword
* my-rule-name
  + Name of the rules
  + ex: custom-rules
* description
  + Describe the usage of the ruleset or something
  + ex: For replace the string from one word to another
* given
  + part of the documents to apply rules
  + If you are using the functions in ruleset then the function part is present in the given value
  + Ex: $.paths.[\*].[\*].responses
    - $.paths – refer the root
    - [\*] – refer the directory
    - Responses – function name
* then
  + It explains which functions to apply to the given part
  + It may be a core or custom function
* Field
  + It is optional
  + It is applying the function to a specific property in an object
  + It is not given the function should be applied or entire target of the given object
* Function
  + Name of the function
  + May be a core or custom function

**Custom Functions**

* Spectral allows to write and use custom function
* Custom function is directly call from ruleset
* The specific function should mentioned in the top of the rules tag

Example

In this example we have check the given openapi has 3.X.X version else it will throw the error it is not a valid openapi document

We have change the openapi version specification as 2.0.1

openapi: 2.0.1

🡪 Create one new JavaScript file name as “responses.js” and it is looks like

module.exports = (responseObjects) => {

  const okOrDefault = Object.keys(responseObjects)

    .filter((key) => key.match(/2[0-9]{2}|default/));

  if (okOrDefault.length === 0) {

    return [{ message: 'Both 2xx operations and default are missing' }];

  }

};

🡪 Then write one custom ruleset for this function

extends: spectral:oas

functions: [responses]

rules:

  default-response-fallback:

    message: "Response object does not have 2xx operation or default set"

    given: "$.paths.[\*].[\*].responses"

    severity: error

    then:

      function: responses

🡪 Now linting the ruleset by using the following command and see the output

spectral lint --ruleset rulesets/ruleset-example2.yaml examples/openapi.json

The output will like

1:1 warning unrecognized-format the provided document does not match any of the registered formats [OpenAPI 2.0 (Swagger), OpenAPI 3.x, OpenAPI 3.0.x, OpenAPI 3.1.x]

Where the Ruleset present in spectral

Following steps to be followed to find the ruleset

* Step1: Download the spectral libraries from the GitHub URL
  + <https://github.com/stoplightio/spectral>
* After downloading the source code the ruleset present in the following directories
  + Packages/rulesets/src/oas/index.js

Where the Core function present in spectral

Following steps to be followed to find the ruleset

* Step1: Download the spectral libraries from the GitHub URL
  + <https://github.com/stoplightio/spectral>
* After downloading the source code the ruleset present in the following directories
  + Packages/functions/src/

**Test Case**

**Case1: Versioning Test Using Custom Function**

Steps to reproduce

1. We use the above example to check the spectral check the versioning of OpenAPI
2. The ruleset be like

extends: spectral:oas

functions: [responses]

rules:

  default-response-fallback:

    message: "Response object does not have 2xx operation or default set"

    given: "$.paths.[\*].[\*].responses"

    severity: error

    then:

      function: responses

1. The response function looks like

module.exports = (responseObjects) => {

  const okOrDefault = Object.keys(responseObjects)

    .filter((key) => key.match(/2[0-9]{2}|default/));

  if (okOrDefault.length === 0) {

    return [{ message: 'Both 2xx operations and default are missing' }];

  }

};

Result:

Input1:

openapi: 2.0

Output1:

1:1 warning unrecognized-format the provided document does not match any of the registered formats [OpenAPI 2.0 (Swagger), OpenAPI 3.x, OpenAPI 3.0.x, OpenAPI 3.1.x]

Input2:

openapi: 3.0

Output2:

No results with a severity of 'error' or higher found!

**Case2: Checking Tags**

In this test case we are going to test the OpenAPI tags is followed style guides and also we will check what the output is and how the spectral warn if it is not proper tags

Steps to reproduce

1. We use the above example to remove the global tags value
2. From the above example change the name of the tag value
3. Given the correct tag
4. Change the local tag value

Results

Input1: remove the tag named as “admins”

tags:

- name: developers

    description: Operations available to regular developers

Output1:

64:11 warning operation-tag-defined Operation tags must be defined in global tags. paths./inventory.post.tags[0]

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

Input2: Change the Tag name value

tags:

  - name: admins1

    description: Secured Admin-only calls

  - name: developers

    description: Operations available to regular developers

Output2:

65:11 warning operation-tag-defined Operation tags must be defined in global tags. paths./inventory.post.tags[0]

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

Input3: Given the correct tag

tags:

  - name: admins

    description: Secured Admin-only calls

  - name: developers

    description: Operations available to regular developers

Output3:

No results with a severity of 'error' or higher found!

Input4: Change the local tag value

In this example we have change the local tag value, this value is not present in the global tag

paths:

  /inventory:

    get:

      tags:

        - developers1

Output4:

23:11 warning operation-tag-defined Operation tags must be defined in global tags. paths./inventory.get.tags[0]

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

**Case3: Checking Info Object**

In this test case we are going to test the info object and it is fields and each fields having proper values

Steps to reproduce

1. Remove the Info Object
2. Remove the description
3. Change the description value, gives invalid data and check
4. Remove the version
5. Change the version value and check the result
6. Remove the title
7. Check with invalid title data
8. Remove the contact object
9. Check with invalid email id data
10. Remove the license

Results

Input1: Remove the info object in openapi and check with linter

Output1:

1:1 warning info-contact Info object must have "contact" object.

1:1 warning info-description Info "description" must be present and non-empty string.

1:1 error oas3-schema Object must have required property "info".

✖ 3 problems (1 error, 2 warnings, 0 infos, 0 hints)

Input2: Remove the description

info:

  version: "1.0.0"

  title: Simple Inventory API

  contact:

    email: you@your-company.com

  license:

    name: Apache 2.0

    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'

Output2:

6:6 warning info-description Info "description" must be present and non-empty string. info

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

Input3: Check with invalid description data

info:

  description: 1234587

  version: "1.0.0"

  title: Simple Inventory API

  contact:

    email: you@your-company.com

  license:

    name: Apache 2.0

    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'

Output3:

7:16 error oas3-schema “description" property type must be string. info.description

✖ 1 problem (1 error, 0 warnings, 0 infos, 0 hints)

Input4: Remove the version

info:

  description: This is a simple API

  title: Simple Inventory API

  contact:

    email: you@your-company.com

  license:

    name: Apache 2.0

    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'

Output4:

6:6 error oas3-schema “info" property must have required property "version". Info

Input5: Check with invalid version value

  version: back

Output5:

No results with a severity of 'error' or higher found!

Input6: Remove the title

info:

  description: This is a simple API

  version: "1.0.0"

  contact:

    email: you@your-company.com

  license:

    name: Apache 2.0

    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'

Output6:

6:6 error oas3-schema “info" property must have required property "title". Info

Input7: Check with Invalid title data

title: 1245

Output7:

9:10 error oas3-schema “title" property type must be string. info.title

Input8: Remove the contact object

info:

  description: This is a simple API

  version: "1.0.0"

  title: Simple Inventory API

  license:

    name: Apache 2.0

    url: 'http://www.apache.org/licenses/LICENSE-2.0.html'

Output8:

6:6 warning info-contact Info object must have "contact" object. info

Input9: Check with Invalid email data

contact:

    email: invaliddata.com

Output9

11:12 error oas3-schema "email" property must match format "email". info.contact.email

Input10: Remove the license

info:

  description: This is a simple API

  version: "1.0.0"

  title: Simple Inventory API

  contact:

    email: you@your-company.com

Output10:

No results with a severity of 'error' or higher found!

**Case4: Checking Server Object**

In this test case we are going to test the server object and its fields and each fields having proper values

Steps to reproduce

1. Remove the Server Object
2. Remove the description
3. Check with invalid description value
4. Remove the URL
5. Check with invalid URL value

Results

Input1: Remove the server Object from the above example

Output1:

1:1 warning oas3-api-servers OpenAPI "servers" must be present and non-empty array.

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

Input2: Remove the description

servers:

    url: https://virtserver.swaggerhub.com/self51142/API1/1.0.0

Output2:

2:9 warning oas3-api-servers OpenAPI "servers" must be present and non-empty array. servers

2:9 error oas3-schema "servers" property type must be array. servers

✖ 2 problems (1 error, 1 warning, 0 infos, 0 hints)

Input3: Check with invalid description value

servers:

  - description: 145212

    url: https://virtserver.swaggerhub.com/self51142/API1/1.0.0

Output3:

3:18 error oas3-schema "description" property type must be string. servers[0].description

Input4: Remove the URL

servers:

  - description: "SwaggerHub API Auto Mocking"

Output4:

3:5 error oas3-schema "0" property must have required property "url". servers[0]

✖ 1 problem (1 error, 0 warnings, 0 infos, 0 hints)

Input5: Check with invalid URL value

servers:

  - description: "SwaggerHub API Auto Mocking"

    url: 12457

Output5:

4:10 error oas3-schema "url" property type must be string. servers[0].url

✖ 1 problem (1 error, 0 warnings, 0 infos, 0 hints)

**Case5: Open API Version Test**

Steps to reproduce

1. Change Open API version as 1.0
2. Removing openapi keyword

Result:

Input1:

openapi: 1.0

Output1:

1:1 warning unrecognized-format The provided document does not match any of the registered formats [OpenAPI 2.0 (Swagger), OpenAPI 3.x, OpenAPI 3.0.x, OpenAPI 3.1.x]

Input2: Removing the openapi keyword

Output2:

1:1 warning unrecognized-format The provided document does not match any of the registered formats [OpenAPI 2.0 (Swagger), OpenAPI 3.x, OpenAPI 3.0.x, OpenAPI 3.1.x]

**Case6: Operation Id Unique Test**

Steps to reproduce

1. Give same operation id in different operation

Result:

Input1:

operationId: searchInventory

This operation id is same for both GET and POST method

Output1:

66:20 error operation-operationId-unique Every operation must have unique "operationId". paths./inventory.post.operationId

**Case7: Operation Parameter Unique Test**

Steps to reproduce

1. Give same property in multiple times in single operation

Result:

Input1:

 - in: query

          name: searchString

          description: pass an optional search string for looking up inventory

          required: false

          schema:

            type: string

this seatchString value present in multiple times

Output1:

29:18 error oas3-schema "parameters" property must not have duplicate items (items ## 0 and 1 are identical). paths./inventory.get.parameters

**Case8: Path Parameter Empty Check**

Steps to reproduce

1. Remove all the properties in the parameter

Result:

Input1:

 Check with empty parameter

Output1:

29:18 error oas3-schema "parameters" property type must be array

**Case9: Contact property Check**

Contact object must have "name", "url" and "email"

Steps to reproduce

1. Add new keyword name as “address”

Result:

Input1:

 address: ”XXXXXXXX”

Output1:

11:14 error oas3-schema Property "address" is not expected to be here. info.contact.adderss

**Case10: Path declaration check**

Steps to reproduce

1. Path value having with /
2. Path value having empty parameter {}
3. Path having query

Result:

Input1:

Paths: /inventory/

Or

Paths: /inventory//id

Output1:

21:16 warning path-keys-no-trailing-slash Path must not end with slash. paths./inventory//

Input2:

Paths:

/inventory/{}

Output2:

21:17 warning path-declarations-must-exist Path parameter declarations must not be empty, ex."/given/{}" is invalid. paths./inventory/{}

Input3:

paths:

  /inventory/?id=12:

Output3:

warning path-not-include-query Path must not include query string

**Case11: No $ref check**

Steps to reproduce

1. $ref siblings not found
2. Unused Components
3. Duplicate Components

Result:

Input1:

New $ref value used, but the value is not present in components

$ref: '#/components/schemas/InventoryItem1'

Output1:

260:25 error invalid-ref '#/components/schemas/InventoryItem1' does not exist paths./inventory.get.responses[200].content.application/json.schema.items.$ref

Input2: Unused Components

Create new component but it does not ref to any operation

Output2:

105:20 warning oas3-unused-component Potentially unused component has been detected. components.schemas.InventoryItem1

Input3: Duplicate Component

The component keyword contains duplicate schema object

Output3:

error parser Duplicate key:InventoryItem components.schemas.InventoryItem

**Case12: Schema check**

Steps to reproduce

1. Remove the schema from responses keyword
2. Change schema property type as string

Result:

Input1:

Remove the schema from response

Output1:

56:30 error oas3-schema "application~1json" property must not be valid. paths./inventory.get.responses[200].content.application/json

Input2: Change schema property type as string

schema: "scheme"

Output2:

56:30 error oas3-schema "application~1json" property must not be valid. paths./inventory.get.responses[200].content.application/json

**Case13: Duplicate enum values**

Steps to reproduce

1. Add duplicate enum values in open API

Result:

Input1:

schema:

            type: string

            enum: [dummy,dummy, not\_so\_dummy]

Output1:

37:18 warning duplicated-entry-in-enum "enum" property must not have duplicate items (items ## 0 and 1 are identical) paths./inventory.get.parameters[0].schema.enum

**Case14: Description present eval( values**

Steps to reproduce

1. Add eval( string in description keyword

Result:

Input1:

tags:

  - name: admins

    description: Secured Admin-only calls eval()

Output1:

17:18 warning no-eval-in-markdown Markdown descriptions must not have "eval(". tags[0].description

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

**Case15: Description present <script> values**

Steps to reproduce

1. Add <script> string in description keyword

Result:

Input1:

tags:

  - name: admins

    description: Secured Admin-only calls <script>

Output1:

17:18 warning no-script-tags-in-markdown Markdown descriptions must not have "<script>" tags. tags[0].description

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

**Case16: Length function check**

Steps to reproduce

1. Add two object in tags keyword and set max 1 in rule set

Result:

Input1:

Ruleset:

tags:

  - name: admins

    description: Secured Admin-only calls

  - name: developers

    description: Operations available to regular developers

Function:

function: length,

        functionOptions: {

          max: 1

        },

Output1:

15:6 warning operation-singular-tag Operation must not have more than a single tag. tags

✖ 1 problem (0 errors, 1 warning, 0 infos, 0 hints)

|  |  |  |
| --- | --- | --- |
| Use Case | Output: Error / Warning | Output |
| Open API Version | Warning | unrecognized-format The provided document does not match any of the registered formats [OpenAPI 2.0 (Swagger), OpenAPI 3.x, OpenAPI 3.0.x, OpenAPI 3.1.x] |
| Remove openapi | Warning | unrecognized-format The provided document does not match any of the registered formats [OpenAPI 2.0 (Swagger), OpenAPI 3.x, OpenAPI 3.0.x, OpenAPI 3.1.x] |
| Operation ID Unique | Error | operation-operationId-unique Every operation must have unique "operationId". paths./inventory.post.operationId |
| Operation Parameter Unique | Error | oas3-schema "parameters" property must not have duplicate items (items ## 0 and 1 are identical). |
| Remove the tag | Warning | operation-tag-defined Operation tags must be defined in global tags |
| Remove the tag value | Warning | operation-tag-defined Operation tags must be defined in global tags. |
| Duplicate the tag value | Error | oas3-schema "tags" property must not have duplicate items (items ## 0 and 1 are identical) |
| Change the tag name | Warning | operation-tag-defined Operation tags must be defined in global tags |
| Change the operation tag value | Warning | operation-tag-defined Operation tags must be defined in global tags |
| Remove Info-Object | Error / Warning | oas3-schema Object must have required property "info". / info-contact Info object must have "contact" object / info-description Info "description" must be present and non-empty string |
| Remove the Info description | Warning | info-description Info "description" must be present and non-empty string |
| Remove the Info Version | Error | oas3-schema "info" property must have required property "version". Info |
| Remove the title | Error | oas3-schema "info" property must have required property "title". Info |
| Remove the License | Success | No results with a severity of 'error' or higher found! |
| Remove the server object | Warning | oas3-api-servers OpenAPI "servers" must be present and non-empty array |
| Remove the server object description | Success | No results with a severity of 'error' or higher found! |
| Empty parameter check | Error | oas3-schema "parameters" property type must be array |
| Add different keyword in contact object | Error | oas3-schema Property "address" is not expected to be here. info.contact.adderss |
| Path having with / | Warning | path-keys-no-trailing-slash Path must not end with slash |
| Path having empty parameter | Warning | path-declarations-must-exist Path parameter declarations must not be empty, ex."/given/{}" is invalid. |
| Path having query parameter | Warning | path-not-include-query Path must not include query string. |
| $ref siblings not found | Error | invalid-ref '#/components/schemas/InventoryItem1' does not exist |
| Unused Components | Warning | oas3-unused-component Potentially unused component has been detected |
| Duplicate Components | Error | parser Duplicate key: InventoryItem components.schemas.InventoryItem |
| Remove schema from responses | Error | oas3-schema "application~1json" property must not be valid |
| Change schema as string | Error | oas3-schema "application~1json" property must not be valid |
| Duplicate entry in enum array | Warning | duplicated-entry-in-enum "enum" property must not have duplicate items (items ## 0 and 1 are identical) |
| Eval( present in description | Warning | no-eval-in-markdown Markdown descriptions must not have "eval(" |
| <script tag present in description | Warning | no-script-tags-in-markdown Markdown descriptions must not have "<script>" tags |
| Length function violation | Warning | operation-singular-tag Operation must not have more than a single tag |

**Limitation of Spectral Linter**

**Case1: Not detecting invalid parameter**

* If the parameter in keyword having invalid parameter type it is not detecting by spectral

Input:

parameters:

        - in: invalidparameter

          name: searchString

          description: pass an optional search string for looking up inventory

          required: true

          schema:

            type: string

In the above input “invalidparameter” value is used in keyword but the result is success, it is not detected by spectral

Output

No results with a severity of 'error' or higher found!

**Case2: Not detecting wrong URL**

* If the URL ends with /.com it is not detected it is not a valid URL

Input:

servers:

  - description: SwaggerHub API Auto Mocking

    url: <https://virtserver.swaggerhub.com/self51142/API1/1.0.0/.com>

Output

No results with a severity of 'error' or higher found!

**Case3: Not detecting invalid response code**

* If the operation response code is invalid it is not detecting by spectral

Input:

responses:

        '-100':

          description: search results matching criteria

          content:

            application/json:

              schema:

                type: array

                items:

                  $ref: '#/components/schemas/InventoryItem'

Output

No results with a severity of 'error' or higher found!

**Alternatives of Spectral Linter**

1. Speccy
2. Zally
3. Openapi-lint

1. Speccy

* It ensures a specification is valid against OpenAPI v3
* Speccy is initiated with usable CLI commands: lint, resolve, and serve
* Resolve command will combine multiple files into a single spec
* It is used MIT License
* Speccy is unmaintained
* Reference Link
  + https://www.npmjs.com/package/speccy

2. Zally

* A minimalistic, simple-to-use OpenAPI 2 and 3 linter
* [zally](https://github.com/zalando/zally) is an open source tool for linting APIs against OpenAPI v2 and v3
* Implement your own rule in Kotlin
* Reference Link
  + <http://opensource.zalando.com/zally/>

3. Openapi-lint

* OpenAPI v3 linter with features for Visual Studio users
* Openapi-lint provides a method to convert between OpenAPI v2 and v3, and offers an ability to validate and lint OpenAPI 3.0.x documents
* Reference Link
  + https://marketplace.visualstudio.com/items?itemName=mermade.openapi-lint

**Spectral in JavaScript**

Add spectral in the header fields

*const { Spectral, Document } = require("@stoplight/spectral-core");*

By using the above line the spectral is added in your project and you can able to use it is ruleset and function and also able to create the custom ruleset and function

How to declare the spectral object?

*const spectral = new Spectral();*

If you want to use the spectral you must declare the spectral object then only you have access the ruleset via the spectral object

How to add ruleset in javascript?

After creating the spectral object then use the setRuleSet() function to add the ruleset in the spectral

*spectral.setRuleset({*

*// a ruleset has to be provided*

*rules: {*

*"no-empty-description": {*

*given: "$..description",*

*message: "Description must not be empty",*

*then: {*

*function: truthy,*

*},*

*},*

*},*

*});*

How to Run Spectral in javascript?

By running the spectral in javascript using the function run()

spectral.run()