**What is Open API?**

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

Purpose of Open API

* Reduce dependencies between develop certain application components.
* Reduce the time of development.
* It is easy to integrate into any development tool.

**Open API Specification**

* Standard, language-agnostic interface to discover and understand 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:

* The Open API is versioned using Semantic Versioning.
* The 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.
  + Use the PATCH version for backwards compatibility bug fixes.
* Example: openapi: 3.0.1

Format:

* The OpenAPI format may be represented either in JSON or YAML format.
* For 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:

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

Data Types:

* The Data Types in the Open API specification are based on the types supported by JSON Schema.
* Null is not supported as a type.
* Primitives have an optional modifier property called “format”
* If email is specified as a 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 uses $ref to be processed as per JSON Reference, using the URL of the current document as the base URI.

OpenAPI Object

* The openAPI object has 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.
* This object has 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 has the following property:
  + Name – String
  + URL – String
  + Email – String

License Object

* License information for the exposed API
* This object has 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 contains information about the server.
* Multiple server details in a single object Server object with the fields shown below
  + 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 keeps track of the relative paths to individual endpoints as well as their operations.
* To construct the whole URL, the path is appended to the URL from the server object.
* This object has the following property:
  + $ref – String
  + Summary – String
  + Description – String
  + Get / Put / post / delete / options / head / patch / trace – Operation Object

Operation Object:

* A single API operation on a path is described.
* This object has the following property:
  + Tag – string
  + Summary – string
  + Description – string
  + operationId – string
  + parameters – Parameter Object
  + requestBody – Request body object
  + responses – Responses Object
  + callbacks – Callback object

Parameter Object:

* It specifies a single operation parameter.
* This object has the following property:
  + Name – string
  + In – string
  + Description – string
  + Required – Boolean
  + Deprecated – Boolean
  + allowEmptyValue – Boolean
  + style – string
  + schema – schema object

Request Body Object:

* It describes the single request body.
* This object has the following property:
  + Description – string
  + Content – Media type object
  + Required – Boolean

Responses Object:

* It has the expected responses of an operation.
* This translates an HTTP response code to the expected response.
* This object has the following property:
  + 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 move into the open API example, we look at what YAML and JSON are.

YAML

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

YAML Syntax

* YAML uses Python style indications to indicate nesting.
* Tab characters are not allowed, so whitespace is used instead.
* Usual format symbols are 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 characters > 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 keyword that indicates the version of the open API document.
* Servers
  + It is an array.
  + It specifies one or more base URLs for your API.
  + Reference Link
    - <https://swagger.io/docs/specification/api-host-and-base-path/>
* Info
  + It contains general information such as the version number, licence notes, contact information, and the 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 operations.
  + Tag names must be unique in the sense that you cannot have two tags with the same name.
  + Reference Link
    - https://swagger.io/docs/specification/grouping-operations-with-tags/
* Paths
  + It is the end points that your API exposes and operations are the HTTP methods used to manipulate these paths, such as GET, POST, and DELETE.
  + The 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 brief one- or two-sentence explanation of the path's purpose.
* operationId
  + Each operation may specify a unique operationId
  + Some code generators 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 a detailed explanation of the path operation.
  + It may contain two or more lines of words.
* Parameters
  + It defines the parameters used in the path.
  + It is an array that should have all the parameters 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 a 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 requires a custom header, then OpenAPI supports 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 the “type” keyword.
  + By default, the required keyword is true and it has the list of data fields to be used in the request body.
  + Each field is described in the “properties” keyword.
  + It is more flexible and reusable. For example, one schema is used by multiple request bodies.
* Responses
  + The 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.
  + The response body is described by using the keyword “schema”.
  + The “content” keywords have which type of data we are getting a successful response to.
  + https://swagger.io/docs/specification/describing-responses/
* Schema
  + The “schema” keyword is used to describe the response body.
  + “object” or “array” is typically used in JSON and XML APIs.
  + Inside the schema object, the data type of response is mentioned as the “type” keyword.
  + The data field is mentioned inside the “properties” keyword. It also contains the datatype of the field and its description.
  + If you want to define the schema globally, then use the keyword “ref” and have the location of the schema which is present.
  + The global schema is present in “components.schemas”, and it is useful for multiple media types to use the same schema.
  + Reference Link
    - <https://swagger.io/docs/specification/describing-responses/>

**Spectral Linter**

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

Purpose of Spectral

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

Spectral License

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

**How to install spectral?**

In Windows

There are three ways to install spectral.

* NPM or YARN
* Executable binaries
* Docker

Reference Link

<https://meta.stoplight.io/docs/spectral/ZG9jOjYyMDc0Mw-installation>

Using NPM

* Run the below command in cmd to install the Spectral globally. It can be accessed from anywhere in the system.

npm install –g @stoplight/spectral-cli

Executable binaries

standalone package for all major platforms.

* Windows
* Linux
* Mac

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

* The quickest way to install the appropriate package for your operating system is via this shell script.
  + curl -L https://raw.github.com/stoplightio/spectral/master/scripts/install.sh | sh

It is not automatically updated. We need to download the latest version to update.

Docker

* Spectral is also available as a Docker image, which can be handy for all sorts of things, like if you are contributing code to Spectral, want to integrate it into your CI build.
* If the file you want to lint is on your computer, you will need to mount the directory where the file resides as a volume.
  + docker run --rm -it -v $(pwd):/tmp stoplight/spectral lint --ruleset "/tmp/.spectral.js" "/tmp/file.yaml"

How to check spectral installed properly or not?

By using the following command to check spectral is installed properly in your system.

spectral –-version

And you have to get the spectral version, which is the one installed on your system.

6.3.0

How to lint spectral?

Use this syntax to lint with the predefined ruleset or a ruleset stored in the same directory as your API document:

spectral lint {openapi\_name}

Example

spectral lint myapifile.yaml

by linting spectral using the following syntax:

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

Example

spectral lint --ruleset ruleset\_name.yaml openapi\_name.yaml

Or

spectral lint --ruleset ruleset.yaml openapi.json

Suppose the ruleset and openapi are presented in different repositories, then the command will be like

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

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

**Spectral Error Results**

* Spectral has a few different error severities.
  + Error
  + Warn
  + Info
  + Hint
* They are in order from highest to lowest.

Error

* If the OpenAPI has invalid syntax, then we get this error.

Warn

* If the OpenAPI needs any specifications, then we got this warning.

Info / Hint

* These are the lowest priority errors.
* These errors are not affecting the OpenAPI output.

Example of the output

In this example, we have renamed the “servers” keyword as “servers1”.

Input

servers1:

- description: SwaggerHub API Auto Mocking

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

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)

In the above example, the error is mentioned as a warning and an error. The last line indicates the total number of errors and warnings.

**Spectral Rulesets**

* It is a collection of rules written in JSON or YAML.
* Spectral has two rulesets – OpenAPI V2 and V3.

Syntax of Spectral

rules:

my-rule-name:

description: Tags must have a description.

given: $.tags[\*]

then:

field: description

function: function\_name

functionOptions:

Property of the function

* rules
  + keyword
* my-rule-name
  + Name of the rules
* description
  + Describe the usage of the ruleset or something.
* given
  + To which section of the documents should the rules be applied?
  + If you want to check only the info object, then the given value is $.info.\*.
  + $ - indicates that the rules apply to the entire section
  + $.info.\* – indicates rules applied only to the info object.
* 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 whether the function should be applied or the entire target of the given object.
* function
  + The name of the function
  + may be a core or custom function.
* functionOptions
  + The function’s property
  + Each function has unique function options.

Example of Rulesets

rules:

'openapi-tags-alphabetical': {

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

      given: '$',

      then: {

        field: 'tags',

        function: alphabetical,

        functionOptions: {

          keyedBy: 'name',

        },

      },

    },

**Core Functions:**

Spectral has a few inbuilt functions that can be used to save the effort of writing functions for common tasks.

* Alphabetical
* Pattern
* Length
* Enumeration
* Falsy
* Casing
* Schema
* Truthy
* Defined
* Undefined
* UnreferencedReusableObject
* Xor
* typedEnum

**Alphabetical**

This function is used to arrange the alphabetical content of simple arrays or objects by passing a key.

Example

By using this function, you checkif the tag has ascending order values in the name key.

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

rulesets/ruleset-alphabetical-order.yaml

rules: {

  'alphabetical-order': {

      given: "$",

      then: {

        field: 'tags',

        function: alphabetical,

        functionOptions: {

          keyedBy: name

        },

      },

    }

}

Openapi.yaml

tags:

  - name: developers

    description: Operations available to regular developers

  - name: admins

    description: Secured Admin-only calls

Output

15:6 warning alphabetical-order properties must follow the alphabetical order tags

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

Now change the response code to alphabetical and run the linter. The output looks like

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

**Pattern**

* This function has a regular expression to check if the value matches the regular expression.
* It has two function options.
  + 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 checked the path URL match the given regular expression.

Lint Command:

spectral lint --ruleset rulesets/ruleset-pattern.yaml examples/openapi-pattern.yaml

Ruleset

rules: {

  'pattern-ruleset': {

      given: "$.paths[\*]~",

      then: {

        function: pattern,

        functionOptions: {

          match: '/^\/(v1|v2)\/([a-z])+\/([a-z])+\/([a-z])+$/'

        },

      },

    }

}

Open API

paths:

  /v1/application/program/name/api:

Output

21:35 warning pattern-ruleset Object{} must match the pattern "/^\\/(V1|V2)\\/([a-z])+\\/([a-z])+\\/([a-z])+$/" paths./V1/application/program/name/ce

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

**Length**

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

function: length

functionOptions:

max: integer number

min: integer number

Example

In this example, we have checked how this function works. By verifying the function, we add two tags in the open API and set the max to 1.

Lint Command:

spectral lint --ruleset rulesets/ruleset-length.yaml examples/openapi-length.yaml

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)

**Enumeration**

* Does the field value exist in this set of possible values?

Example

In this example, we have checked the info.title having the values present in the enumeration field.

Lint Command:

spectral lint --ruleset rulesets/ruleset-enumeration.yaml examples/openapi-enumeration.yaml

Ruleset

rules: {

  'emueration-ruleset': {

    type: style,

      given: "$.info.title",

      then: {

        function: enumeration,

        functionOptions: {

          values: ["title1","title2"]

        },

      },

    }

}

OpenAPI

info:

  description: This is a simple API

  title: "OpenAPI"

Output

8:10 warning emueration-ruleset "OpenAPI" must be equal to one of the allowed values: "title1", "title2" info.title

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

**Falsy**

* It should be checked that the value should be “false”, “”,”0”, “null”, “undefined”

Example

In this example, we have checked that the given object value should be null.

Lint Command:

spectral lint --ruleset rulesets/ruleset-falsy.yaml examples/openapi-falsy.yaml

Ruleset

rules: {

  'falsy-ruleset': {

    type: style,

      given: "$.servers.\*",

      then: [{

        field: 'description',

        function: falsy

      }],

    }

}

OpenAPI

servers:

  - description: Swagger API Auto Mocking

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

Output

3:18 warning falsy-ruleset "description" property must be falsy servers[0].description

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

**Casing**

This function should check if the text must match a certain case, like camelCase or snake\_case.

|  |  |
| --- | --- |
| **Name** | **Sample** |
| Flat | verylongname |
| Camel | veryLongName |
| Pascal | VeryLongName |
| Kebab | Very-long-name |
| Cobol | VERY-LONG-NAME |
| Snake | Very\_long\_name |
| Macro | VERY\_LONG\_NAME |

Example:

In this example, we check the title for having a camel case string.

Lint Command:

spectral lint --ruleset rulesets/ruleset-casing.yaml examples/openapi-casing.yaml

Ruleset

rules: {

  'casing-ruleset': {

    type: style,

      given: "$.info.title",

      then: {

        function: casing,

        functionOptions: {

          type: camel

        },

      },

    }

}

OpenAPI

info:

  description: This is a simple API

  version: "1.0.0"

  title: Simple Inventory API

Output

8:10 warning casing-ruleset must be camel case info.title

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

**Schema**

It uses JSON Schema to treat the contents of the $given JSON path as a JSON instance.

**Truthy**

* It should be checked that the value should not be “false”, “”,”0”, “null”, “undefined”

Example

In this example, we have checked that the given object value should not be null.

Lint Command:

spectral lint --ruleset rulesets/ruleset-truthy.yaml examples/openapi-truthy.yaml

Ruleset

rules: {

  'falsy-ruleset': {

    type: style,

      given: "$.servers.\*",

      then: [{

        field: 'description',

        function: truthy

      }],

    }

}

OpenAPI

servers:

  - description: null

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

Output

3:18 warning truthy-ruleset "description" property must be truthy servers[0].description

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

**Defined**

1. The value must be defined, meaning it must be anything but undefined.
2. It is the opposite of what an undefined function does.

Example

In this example, we have checked whether the server object URL property is present or not.

Lint Command:

spectral lint --ruleset rulesets/ruleset-defined.yaml examples/openapi-defined.yaml

Ruleset

rules: {

  'defined-ruleset': {

    type: style,

      given: "$.servers.\*",

      then: {

        field: 'url',

        function: defined

      },

    }

}

OpenAPI

servers:

  - description: SwaggerHub API Auto Mocking

Output

3:5 warning defined-ruleset "[0].url" property must be defined servers[0]

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

**Undefined**

* The value must be undefined.
* When combined with field: foo on an object, the foo property must be undefined.

Example

In this example, we have checked whether the server object url property is present or not.

Lint Command:

spectral lint --ruleset rulesets/ruleset-undefined.yaml examples/openapi-undefined.yaml

Ruleset

rules: {

  'defined-ruleset': {

    type: style,

      given: "$.servers.\*",

      then: {

        field: 'url',

        function: notdefined

      },

    }

}

OpenAPI

servers:

  - description: SwaggerHub API Auto Mocking

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

Output

4:10 warning defined-ruleset "url" property must be undefined servers[0].url

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

**UnreferencedReusableObject**

* This function identifies unreferenced objects within a document.
* Given should point to the member holding the potential reusable objects

**XOR**

* Communicate that one of these properties is required and no more than one is allowed to be defined.

Example

In this example, we have checked if the info object’s title property is present or not.

Lint Command:

spectral lint --ruleset rulesets/ruleset-xor.yaml examples/openapi-xor.yaml

Ruleset

rules: {

  'xor-ruleset': {

      given: "$.info",

      then: {

        function: xor,

        functionOptions: {

          properties: [title,description]

        },

      },

    }

}

OpenAPI

info:

  description: This is a simple API

  version: "1.0.0"

Output

5:6 warning xor-ruleset "title" and "value" must not be both defined or both undefined info

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

**typedEnum**

* When both a type and an enum are defined for a property, the enum values must respect the type.

**Custom Rulesets:**

How to create custom ruleset?

The custom ruleset looks like

rules:

my-rule-name:

description: “check custom ruleset”.

given: $.tags[\*]

then:

field: description

function:

property of the function

* Rules
  + keyword
* my-rule-name
  + Name of the rules
  + ex: custom-rules
* description
  + Describe the usage of the ruleset or something.
* given
  + To which section of the documents should the rules be applied?
  + If you want to check only the info object, then the given value is $.info.\*.
  + $ - indicates that the rules apply to the entire section
  + $.info.\* – indicates rules applied only to the info object.
* 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 whether the function should be applied or the entire target of the given object.
* function
  + The name of the function
  + may be a core or custom function.
* functionOptions
  + The function’s property
  + Each function has unique function options.

**Custom Functions**

* Spectral allows you to write and use custom functions.
* A custom function is directly called from the ruleset.

Example

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

We have changed the openapi version specification to 2.0.1.

openapi: 2.0.1

🡪 Create one new JavaScript file named “responses.js” and it 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 be 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

The following steps are 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 is present in the following directories:
  + Packages/rulesets/src/oas/index.js

Where the Core function present in spectral

The following steps are to be followed to find the core functions.

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

**Test Case**

**Case1: Using a Custom Function for Versioning**

Steps to reproduce

1. We use the above openapi.yaml file spectral to check the version of the OpenAPI.
2. What should the ruleset look 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 if the OpenAPI tags follow 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 tag 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 “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 changed 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: Examining the Info Object**

In this test case, we are going to test the info object and its fields, and each field must have proper values.

Steps to reproduce

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

Results

Input1: Remove the info object from openapi and run the linter test.

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: Change the description value, give invalid data and check.

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: Delete 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: Change the version value and check the result.

  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 for 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 for invalid email ID 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 Objects**

In this test case, we are going to test the server object and its fields, and each field must have proper values.

Steps to reproduce

1. Remove the server object.
2. Remove the description.
3. Check for an invalid description value.
4. Remove the URL.
5. Check for an 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 for an 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 for an 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 the Open API version to 1.0.
2. Removing the 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: Unique Operation Id Test**

Steps to reproduce

1. Give the same operation id in different operations.

Result:

Input1:

operationId: searchInventory

This operation ID is the same for both the GET and POST methods.

Output1:

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

**Case7: Unique Operation Parameter Test**

Steps to reproduce

1. Give the same property multiple times in a single operation.

Result:

Input1:

 - in: query

          name: searchString

          description: pass an optional search string for looking up inventory

          required: false

          schema:

            type: string

This searchString value is present ultiple 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: Check for Path Parameter Emptiness**

Steps to reproduce

1. Remove all of the parameters properties.

Result:

Input1:

 Check with empty parameters

Output1:

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

**Case9: Check the contact property**

The contact object must include "name", "url" and "email".

Steps to reproduce

1. Add a new keyword named “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 that ends in /
2. Path value concludes with an empty parameter {}
3. Path has a 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: There was no $ref check**

Steps to reproduce

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

Result:

Input1:

A new $ref value is 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

Creates a new component but makes no 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 a duplicate schema object.

Output3:

error parser Duplicate key:InventoryItem components.schemas.InventoryItem

**Case12: Schema check**

Steps to reproduce

1. Remove the schema from the response keyword
2. Change the type of schema property to string

Result:

Input1:

Remove the schema from the response keyword

Output1:

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

Input2: Change the type of schema property to 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: Enum values that are duplicates**

Steps to reproduce

1. In the open API, add duplicate enum values.

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: The description contains eval( values**

Steps to reproduce

1. In the description keyword, include the eval( string

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: The description contains <script> values**

Steps to reproduce

1. In the description keyword, include the <script> string

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: Check the length function**

Steps to reproduce

1. Add two objects to the tags keyword and limit the rule set to 1

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)

**Test Caes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl No** | **Use Case** | **Input** | **Output: Error / Warning** | **Output** |
| 1 | Open API Version  Change the Open API version to 1.0. | Openapi:1.0 | 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] |
| 2 | Remove openapi  The Openapi keyword must be present. | Openapi tag is removed | 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] |
| 3 | Operation ID Unique  Operation ID must be unique. | operationId: searchInventory | Error | operation-operationId-unique Every operation must have unique "operationId". paths./inventory.post.operationId |
| 4 | Operation Parameter Unique  Request parameter must be unique. | - in: query  name: searchString | Error | oas3-schema "parameters" property must not have duplicate items (items ## 0 and 1 are identical). |
| 5 | Remove the tag  The tag keyword must be present. | Remove the tag object | Warning | operation-tag-defined Operation tags must be defined in global tags |
| 6 | Remove the tag value  The tag should contain the value. | Tags:  Paths: | Warning | operation-tag-defined Operation tags must be defined in global tags. |
| 7 | Duplicate the tag value  Tag cannot have duplicate values. | tags:  - name: admins  description: Secured Admin-only calls  - name: admins  description: Secured Admin-only calls | Error | oas3-schema "tags" property must not have duplicate items (items ## 0 and 1 are identical) |
| 8 | Change the tag name  The global tag must include an operation tag. | tags:  - name:newtag  description: Secured Admin-only calls | Warning | operation-tag-defined Operation tags must be defined in global tags |
| 9 | Change the operation tag value  Operation tags must be defined in global tags. | paths:/inventory:  get:  tags:  - developers1 | Warning | operation-tag-defined Operation tags must be defined in global tags |
| 10 | Remove Info-Object  Info must be present | Info object is removed from the openapi | 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 |
| 11 | Remove the Info description  A description of the information must be present and must be a non-empty string. | Remove the description tag present inside of info object | Warning | info-description Info "description" must be present and non-empty string |
| 12 | Remove the Info Version  info property must have the required property version. | Remove the version tag present inside of info object | Error | oas3-schema "info" property must have required property "version". Info |
| 13 | Remove the title  info property must have the required property title. | Remove the title tag present inside of info object | Error | oas3-schema "info" property must have required property "title". Info |
| 14 | Remove the License  It is not necessary to have a license. | Remove the license tag present inside of info object | Success | No results with a severity of 'error' or higher found! |
| 15 | Remove the server object  Servers must be present and non-empty array | Remove the server tag from the openapi | Warning | oas3-api-servers OpenAPI "servers" must be present and non-empty array |
| 16 | Remove the server object description  The server object description is not required. | Remove the server object description | Success | No results with a severity of 'error' or higher found! |
| 17 | Empty parameter check  Parameter property type must be array | Remove all the properties in the parameter | Error | oas3-schema "parameters" property type must be array |
| 18 | Add different keyword in contact object  The contact object must include "name", "url" and "email". | address: ”XXXXXXXX” | Error | oas3-schema Property "address" is not expected to be here. info.contact.adderss |
| 19 | Path having with /  Path must not end with slash | Paths:/inventory/ | Warning | path-keys-no-trailing-slash Path must not end with slash |
| 20 | Path having empty parameter  Path parameter declarations must not be empty | Paths:  /inventory/{} | Warning | path-declarations-must-exist Path parameter declarations must not be empty, ex."/given/{}" is invalid. |
| 21 | Path having query parameter  Path must not include query string | paths:    /inventory/?id=12 | Warning | path-not-include-query Path must not include query string. |
| 22 | $ref siblings not found  $ref siblings must be present in components | $ref: '#/components/schemas/InventoryItem1' | Error | invalid-ref '#/components/schemas/InventoryItem1' does not exist |
| 23 | Unused Components  Component siblings must be defined in $ref | Create new component but it does not ref to any operation | Warning | oas3-unused-component Potentially unused component has been detected |
| 24 | Duplicate Components  Components must not have duplicate schema | The component keyword contains duplicate schema object | Error | parser Duplicate key: InventoryItem components.schemas.InventoryItem |
| 25 | Remove schema from responses  Schema property must be valid | Remove the schema from response | Error | oas3-schema "application~1json" property must not be valid |
| 26 | Change schema as string  A non-empty array should be used for the schema property. | schema: "scheme" | Error | oas3-schema "application~1json" property must not be valid |
| 27 | Duplicate entry in enum array  Enum must not have duplicate items | schema:   type: string   enum: [dummy,dummy, not\_so\_dummy] | Warning | duplicated-entry-in-enum "enum" property must not have duplicate items (items ## 0 and 1 are identical) |
| 28 | Eval( present in description  Description must not have eval( | tags:   - name:admins   description: Secured Admin-only calls eval() | Warning | no-eval-in-markdown Markdown descriptions must not have "eval(" |
| 29 | <script tag present in description  Description must not have <script> tag | tags:   - name:admins   description: Secured Admin-only calls <script> | Warning | no-script-tags-in-markdown Markdown descriptions must not have "<script>" tags |
| 30 | Length function violation  API must support the predefined length mentioned in the ruleset | function: length,          functionOptions: {            max: 1          }, | Warning | operation-singular-tag Operation must not have more than a single tag |

**Limitation of Spectral Linter**

**Case1: Not detecting invalid parameters**

* If the parameter in the keyword has an invalid parameter type, it is not detected by spectral.

Input:

parameters:

        - in: invalidparameter

          name: searchString

          description: pass an optional search string for looking up inventory

          required: true

          schema:

            type: string

The keyword "invalidparameter" is used in the above input, but the result is success, as it is not detected by spectral.

Output

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

**Case2: Not detecting the wrong URL**

* If the URL ends with /.com, it is not a valid URL but it is not detected in spectral analysis.

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 codes**

* 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
4. Oas-kit
5. Openapilint
6. Openapi Spec Validator
7. Oval

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.
* The MIT License governs its use.
* Speccy is unmaintained.
* Reference Link
  + <https://www.npmjs.com/package/speccy>
* Pros
  + It made use of the MIT License.
  + It has predefined rules against openapi.
  + It has supported the custom rules also.
* Cons
  + It is unmaintained.
  + It is validated only by openapi V3.

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 standards.
* Implement your own rules in Kotlin.
* Reference Link
  + <http://opensource.zalando.com/zally/>
* Pros
  + It made use of the MIT License.
  + OpenAPI 3 and OpenAPI 2 specifications are supported.
  + Detailed check configuration
  + Using the ignore functionality in your API definition, you can disable rules for a specific API.
  + API-specific code is written in the Java and Kotlin programming languages.
* Cons
  + It checks the API against the rules defined in zalando’s(<https://opensource.zalando.com/restful-api-guidelines/>) but anyone can use it out-of-the-box.

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>
* Pros
  + It is able to convert openapi v2 into openapi v3.
  + It is able to convert JSON into YAML and YAML into JSON files.
  + It provides 11 visual studio commands for operation.
* Cons
  + It is used only in Visual Studio.
  + Custom rulesets and functions cannot be added.

4. Oas-kit

* A collection of packages which comprise an OpenAPI 2.0 to 3.0.X converter, a resolver, a validator, a schema-walker and a linter
* Pros
  + It is able to convert openapi v2 into openapi v3.
  + It is able to convert JSON into YAML and YAML into JSON files.
* Cons
  + Custom rulesets and functions cannot be added.