**Spectral Linter**

* It is open source JSON / YAML linter
* Create style guides for your OpenAPI / AsyncAPI

Purpose of Spectral

* It can checking API against API guidelines
* We have also create custom rules against API and check their API support the rules

Spectral License

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

**What is Open API?**

* It is public API
* Made publicly available to 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
  + Schema
  + Open API Object
  + Info Object
  + Contact Object
  + License Object
* Reference Link
  + <https://swagger.io/specification/>

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://www.redhat.com/en/topics/automation/what-is-yaml#:~:text=YAML%20is%20a%20data%20serialization,is%20for%20data%2C%20not%20documents.

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": "3.1.0",

    "info": {

      "title": "test-2",

      "version": "1.0"

    },

    "servers": [

      {

        "url": "http://localhost:8080"

      }

    ],

    "paths": {

      "/users/{userId}": {

        "parameters": [

          {

            "schema": {

              "type": "integer"

            },

            "name": "user\_id",

            "in": "path",

            "required": true,

            "description": "Id of an existing user."

          }

        ],

        "get": {

          "summary": "Get User Info by User ID",

          "tags": [],

          "responses": {

            "200": {

              "description": "User Found",

              "content": {

                "application/json": {

                  "schema": {

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

                  },

                  "examples": {

                    "Get User Alice Smith": {

                      "value": {

                        "id": 142,

                        "firstName": "Alice",

                        "lastName": "Smith",

                        "email": "alice.smith@gmail.com",

                        "dateOfBirth": "1997-10-31",

                        "emailVerified": true,

                        "signUpDate": "2019-08-24"

                      }

                    }

                  }

                }

              }

            },

            "404": {

              "description": "User Not Found"

            }

          },

          "operationId": "get-users-userId",

          "description": "Retrieve the information of the user with the matching user ID."

        }

      }

    },

    "components": {

      "schemas": {

        "User": {

          "title": "User",

          "type": "object",

          "description": "",

          "examples": [

            {

              "id": 142,

              "firstName": "Alice",

              "lastName": "Smith",

              "email": "alice.smith@gmail.com",

              "dateOfBirth": "1997-10-31",

              "emailVerified": true,

              "signUpDate": "2019-08-24"

            }

          ],

          "properties": {

            "id": {

              "type": "integer",

              "description": "Unique identifier for the given user."

            },

            "firstName": {

              "type": "string"

            },

            "lastName": {

              "type": "string"

            },

            "email": {

              "type": "string",

              "format": "email"

            },

            "dateOfBirth": {

              "type": "string",

              "format": "date",

              "example": "1997-10-31"

            },

            "emailVerified": {

              "type": "boolean",

              "description": "Set to true if the user's email has been verified."

            },

            "createDate": {

              "type": "string",

              "format": "date",

              "description": "The date that the user was created."

            }

          },

          "required": [

            "id",

            "firstName",

            "lastName",

            "email",

            "emailVerified"

          ]

        }

      }

    }

  }

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
  + <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
  + <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
  + 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
  + <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
  + <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

* + <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
  + <https://swagger.io/docs/specification/describing-responses/>

--what is spectral and purpose

**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 Rulesets**

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

Example of Rulesets

*rules:*

*oas3-api-servers:*

*description: "OpenAPI `servers` must be present and non-empty."*

*formats: ["oas3"]*

*given: "$.path.schema"*

*severity: error*

*then:*

*field: servers*

*function: schema*

*functionOptions:*

*schema:*

*items:*

*type: object*

*minItems: 1*

*type: array*

**Core Functions:**

**Alphabetical**

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

Example

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!

**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

**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 js 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]*

**Test Case**

**Case1: Versioning Test**

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)

|  |  |  |
| --- | --- | --- |
| Use Case | Output: Error / Warning | Output |
| Remove the version | Warning | error oas3-schema "info" property must have required property "version". Info |
| Remove the info object | Error |  |

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: abcd

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 it is 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)

**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()