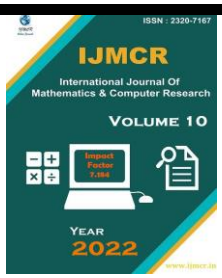
International Journal of Mathematics and Computer Research

ISSN: 2320-7167

Volume 10 Issue 04 April 2022, Page no. – 2655-2661

Index Copernicus ICV: 57.55, Impact Factor: 7.184

DOI: 10.47191/ijmcr/v10i4.04



About One Approach to Deploying an API Gateway Development Environment

Denis Zolotariov

Ph.D., Kharkov, Ukraine, https://orcid.org/0000-0003-4907-7810

ARTICLE INFO

Published online:
26 April 2022

Corresponding Author:
Denis Zolotariov

ABSTRACT

The article outlines the problem of an integral approach to building an API gateway that includes the API gateway itself, the development of documentation for the API, sandbox for API end users, and automated testing. This problem is divided into tasks, each of which is solved. As a result, a general approach has been developed for deploying an environment for developing and executing API gateway, which takes into account: convenient configuration and launch of the gateway server, flexible development of APIs' documentation and its convenient viewing by end users, as well as end-to-end and performance testing developed APIs. A demo example is given and its implementation features are described. The ways of its further development are proposed.

KEYWORDS: API, API gateway, microservices, API documentation, cloud computing

I. INTRODUCTION

In the modern world, there are more and more Internet services that use APIs in their work [1,2]. Akamai publishes a report [3] according to which more than 83% of all web traffic currently comes from API-based services. According to [1,4], these are regular sites, web components (maps, widgets) and web applications and programs for mobile platforms.

According to Slashdata [2], at the end of 2020, 69% of developers use third-party APIs, and 20% use internal or private APIs. In total, according to [1], about 90% of developers use the API in their work. And that is why the API management market will be worth $5.1 billion by 2023 [5].

The wider the practice of using the API, the more often it becomes necessary to standardize its development within the company to unify this process. There are already approaches to API standardization, such as OpenAPI [6] or Open Digital Architecture [7], but they are either limited to an external API interface or intended for a full digital transformation of enterprise software. The task of standardizing the complex development of API-gateways within small and medium APIs remains unexplored.

The article proposes one of the options for building such a process for developing and executing a highly productive API gateway, including the flexible development of API documentation and its convenient viewing by API end users, as well as end-to-end and performance testing of developed APIs.

II. PROBLEM STATEMENT

The above problem can be solved based on the following components.

Since the API-gateway must be highly productive, it is proposed to use nginx [8] as a server or its assembly, for example, OpenResty [9].

Documentation for APIs is proposed to be maintained in accordance with the OpenAPI Specification (OAS) [10] version 3.0 – the one of the world standards for RESTful APIs, which has been developed since 2015 and has gone through many stages of transformations and improvements. This specification creates a RESTful interface by mapping all API resources and their associated operations into a single yaml file. This greatly simplifies API development, testing and use for developers, DevOps and API end users. The final yaml file itself is proposed to be created using a wide range of snippets, developed once for a whole family of APIs.

To view the documentation developed in this way, it is proposed to use web-based software, which should allow to view and use the API in a sandbox mode in a web browser, which will allow a wide range of people to do this without installing additional software.

Also, an important component is the general testing of the developed API. This type of testing includes end-to-end and

performance testing, which should also be taken into account.

III. APPROACH

To separate each component from the rest, but at the same time to combine them all into a single system that is understandable and convenient for the developer, DevOps specialist and system administrator, it is proposed to use a single directory with the following subdirectory structure.

All data regarding one API-gateway is stored in one directory. The basic overall folder structure (ordered alphabetically) is shown in Fig. 1 and corresponds to the proposed approach:

- bash – contains service scripts for first run and reload configuration of API-gateway and others;
- conf – configurations for the nginx server and the logrotate service;
- logs – logs of the web server, the application itself, and other applications;
- openapi – contains snippets for building API documentation according to the OpenAPI Specification;
- tests – end-to-end and performance tests of the API to check the correct operation of the API, as well as determine the regression of API performance;
- www – web applications for viewing and editing documentation according to the OAS specification, as well as ready-made files for such documentation.

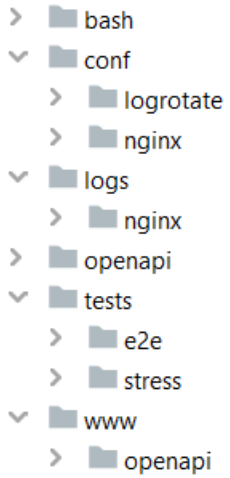


Fig. 1. Common basic folder structure for API development environments

Let's take a closer look at each of the components.

A. Configurations

The configuration directory shown in Fig. 2 contains files with settings for the logrotate service [11] and the nginx server [8].

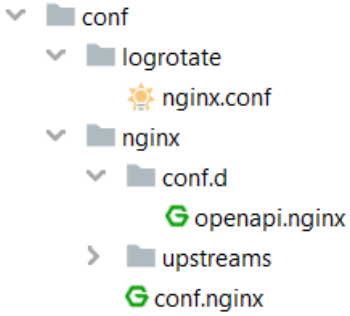


Fig. 2. The structure of the configuration directory of the API gateway

In the base case, the logrotate system service is used to manage the logs of a web server and API applications, as the most simple and common way. The *conf/logrotate/nginx.conf* file contains the nginx log rotation settings for the gateway API being developed.

The *conf/nginx/conf.d/openapi.nginx* file describes the settings for the API documentation viewer web application according to the OAS specification.

The *conf/nginx/upstreams* directory contains settings for nginx of the same name, which are placed here for ease of editing.

If a separate copy of nginx is launched for the gateway API being developed, then the full nginx configuration is written to the *conf/nginx/conf.nginx* file. Otherwise, a soft-link is created from *conf/nginx/conf.nginx* to the */etc/nginx/api-enabled/* directory, all configurations from which should be automatically included in the main configuration file */etc/nginx/nginx.conf* when loading a single copy of nginx.

Also, it's worth noting that the OpenResty build can be used instead of the default nginx build because it includes a lot more features based on Lua programming language.

B. Service scripts

The bash directory shown in Fig 3 contains scripts that are responsible for:

- *make_yaml.sh* – assembly of yaml documentation for viewing;
- *reloady.sh* – updating the configuration on the running nginx server;
- *run.sh* – the initial launch of the new API with the addition of configuration to nginx and logrotate.

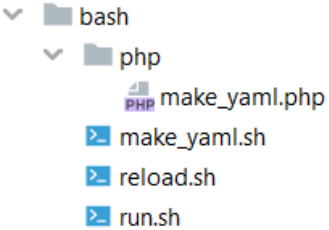


Fig. 3. Service scripts

The standard yaml file does not have the ability to include links to external yaml sources. Therefore, to solve this problem, a PHP yaml builder has been developed, located in

the *bash/php/make_yaml.php* file, and wrapped in a bash script *bash/make_yaml.sh* for ease of calling via the CLI. The finished documentation is collected by the builder in the *www/openapi/yaml* directory.

The yaml builder is developed using standard PHP functions: yaml_parse, yaml_parse_file, yaml_emit_file. And it additionally includes the possibility of both simple substitutions like "{{name}}", which replaces it with some static text, and recursive substitutions of the contents of other files with parameters, for example:

```
!include
"relative/path/to/file|param1=value1|param2=value2",
```

where the parameters are simple substitutions of the first type, for example "{{param1}}" will be replaced by the string "value1".

Also, the following mechanism is used to add fields of a yaml file to the contents of another yaml file. The "x-placeholder" and the name of the yaml file to replace it are specified in the main yaml file, and the "include_add" section is added to it as follows:

```
Components:
    examples:
      x-error-examples: !include "example/error"
  include_add:
    'components.examples.x-error-examples':
'components.examples'
```

In this case, the contents of the yaml file *example/error.yaml* will be added to the contents of the current yaml file to "components.examples" yaml-path. The introduction of the "x-error-examples" intermediate directive is only necessary for convenient identification of the place of adding by the developer of documentation, which was the result of practical tests of the yaml builder.

As a starting point for building, the yaml builder uses the *.exports.yaml* file located at the root of the directory, the path to which is passed to it as a command line parameter when called. This file contains 3 sections: "dest", "replace" and "files". The "dest" section describes the path to the directory where the finished yaml files will be saved. The "replace" section contains an array of static replacements of the form "name: replace string". The next section, "files", is the main one, as it contains a list of files in the root of the directory that need to be processed by the yaml builder.

This approach allows you to build a single yaml file recursively from multiple sources using dynamic parameters. This allows you to use many once-designed general purpose yaml files as snippets.

C. API documentation

The API documentation is generated according to the OpenApi Specification, in this work the current version of OAS 3.0 is used, and is located in directories according to the following hierarchy (see Fig. 4).

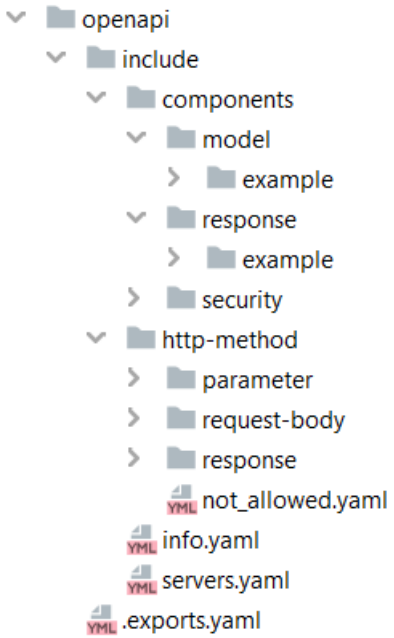


Fig. 4. Directory structure for developing documentation

The main API documentation files are formed in the *openapi/* root directory, each yaml file from which includes snippets from the file *openapi/.exports.yaml* for simple static substitutions and *openapi/include/* subdirectory. When properly formed, the main yaml files are essentially collections of links to snippet-files.

Let's take a closer look at the subdirectories in *openapi/include/*. This directory contains a general purposes snippets like an API general description or a sandbox servers.

The subdirectory *./components* contains snippets for: server responses (*./response*) for one object, array of objects and errors with examples for them (*./response/example*); object (model) fields description (*./model*) with examples; and security modes (*./security*) like cookie, query string or bearer.

The subdirectory *./http-method* contains snippets for HTTP-methods for each type of request to API, for example, GET, POST or DELETE. For a client request, these are the query string parameters (*./parameter*) and the request body (*./request-body*). For a server response, these are response codes with corresponding response body (*./response*).

The following lines in the file *openapi/mymodel.yaml* can serve as an example of including these snippets:

```
info:
  title: MyApi/{{File_name}} API
security: !include "security-all"
tags:
- name: "Get MyModel"
  description: >
    Get MyModel [docs]({{url_ui}}?yaml=mymodel)
paths:
  /:
    delete: !include "http-method |method=delete|code=405"
components:
  schemas:
```

```
    Basic: !include "components/model/basic"
  examples:
    x-error-examples: !include "response/example/error"
include_add:
  'components.examples.x-error-examples': 'compo-
nents.examples'
```

The contents of the *openapi/.exports.yaml* file can be as follows:

```
dest: www/openapi/yaml
replace:
  url_ui: https://example.com/openapi/ui/
files:
- mymodel
```

D. View documentation

Viewing the final documentation is intended primarily for end users – application developers who use the proposed API for their needs. To view it, the Swagger UI web application [12] is used, which loads ready-made yaml files located in the *www/openapi/yaml* directory (see Fig. 5).

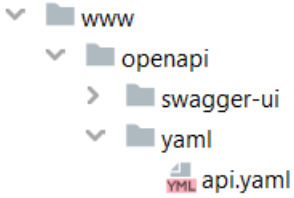


Fig. 5. The directory structure of the finished API documentation

Swagger UI was chosen because it is a simple, single-page web application written in a pure HTML and JS that runs entirely client-side in a browser.

It is also worth noting that for the Swagger UI to work correctly, the *www/openapi/swagger-ui/index.html* file must have several minor changes in start of JS code:

```
const params = new URLSearchParams(
    window.location.search);
const yaml=params.get('yaml');
const ui = SwaggerUIBundle({
    url: window.location.protocol + '//' +
        window.location.host + '/openapi/yaml/' + yaml,
```

These changes will allow short links to files like *https://example.com/openapi/ui/?yaml=filename*.

In addition, to use short URLs for yaml files like https://*example.com/openapi/yaml/filename.yaml*, the following alias must be specified in the nignx configuration file *conf/nginx/conf.d/openapi.nginx*:

```
location ~ ^/openapi/yaml/(?<yaml>[^/]+)$ {
  alias / path/to/example.com/www/openapi/yaml/;
  try_files $yaml.yaml =404;
}
```

IV. APPROACH VALIDATION

Based on the approach proposed in the article, a demo example of the API for Warehouse and it's Orders and Products was developed, below in Fig. 6 shows the configuration for an API based on the Orders and Products microservices:

- warehouse → orders (*/v1/warehouse/orders*),
- warehouse → products (*/v1/warehouse/products*) and
- order → product (*/v1/warehouse/orders/10/product*).

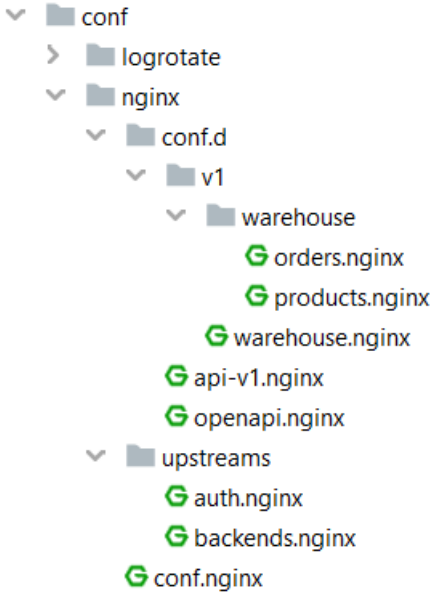


Fig. 6. Structure of nginx server settings for Warehouse API gateway

Access to each of them is determined by the firewall, as well as the static secret token from *upstreams/auth.nginx*. Multiple upstreams for each of them is described in *upstreams/backends.nginx*.

In all templates and snippets of the documentation presented in Fig. 7 and Fig. 8, the principle of inheritance is applied: there is a "basic" model, from which all subsequent ones are inherited: "order" and "product". The file *openapi/api.yaml* correctly describes errors for requests to URLs */* and */v1/*.

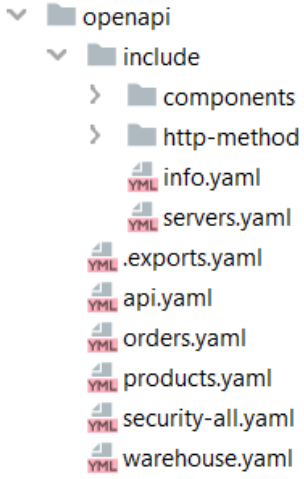


Fig. 7. Main yaml documentation files

The final view of the finished documentation in the Swagger UI viewer looks like in Fig. 9.

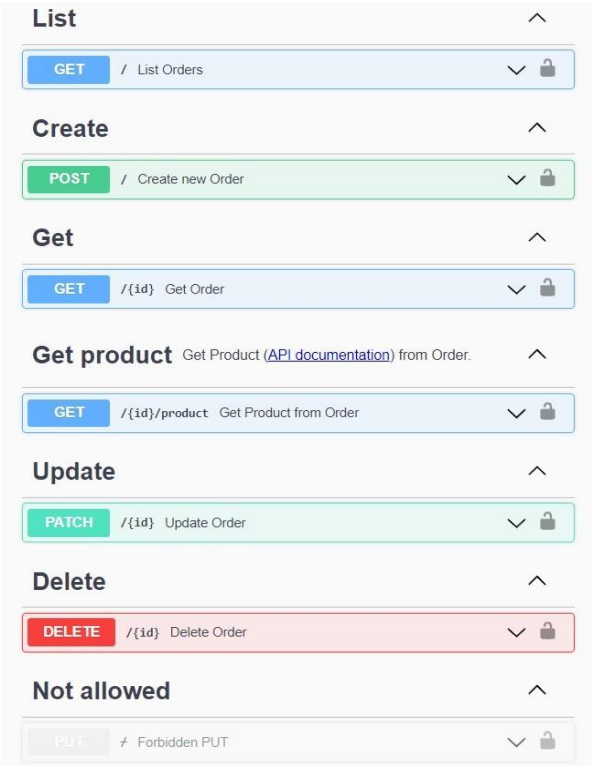


Fig. 9. Swagger UI for developed documentation for Warehouse/Orders API

End-to-end tests are built on the top of PHP Unit Framework and check the correctness of the gateway API and application API responses to all possible correct and incorrect REST requests. Without going into implementation details, as this is not the subject of this article, the general structure of the tests is shown in Fig. 10.

File *openapi/.exports.yaml* contains the following yaml code:

```
dest: www/openapi/yaml
replace:
  url_ui: https://example.com/openapi/ui/
  uri_server: https://example.com
files:
- api
- warehouse
- products
- orders
```

As mentioned earlier, the files in the *openapi/* root directory contain, for the most part, only links to the snippet files in the *openapi/include* subdirectory. For example, the most informative part of the *openapi/orders.yaml* file looks like this:

```
info:
  title: Warehouse/{{File_name}} API
x-info-replacer: !include include/info
servers:  !include "include/servers|path_prefix=warehouse/{{file_name}}"
security: !include "security-all"
tags:
- name: "Get product"
  description: >
    Get Product ([API documentation]({{url_ui}}?yaml=products)) from Order.
paths:
  /:
    get:
      responses:
        '200':    !include "include/http-method/response/200|schema=ResponseOrderArray|example_name=Array of
Orders|example=ResponseOrderArray"
        '4XX':   !include "include/http-method/response/errors-list-400,401,403,429"
        '5XX':   !include "include/http-method/response/5XX"
    post:
```

```
    requestBody: !include "include/http-method/request-body/simple|entity=Order|entity-example=Order-Create"
    responses:
      '201':    !include "include/http-method/response/201|schema=ResponseOrder|example_name=Order|example=
ResponseOrder"
      '4XX':   !include "include/http-method/response/errors-create-400,401,403,413,415,422,429"
  delete: !include "include/http-method/not_allowed|METHOD=DELETE|httpcode=405"
components:
 parameters:
  id: !include "include/http-method/parameter/path/id32|name=id|entity=Order"
 schemas:
  Basic: !include "include/components/model/basic"
  Order: !include "include/components/model/order|basic=Basic"
  ResponseBasic: !include "include/components/response/basic"
  ResponseOrder: !include "include/components/response/entity|basic=ResponseBasic|Entity=Order"
  ResponseOrderArray: !include "include/components/response/entity_array|basic=ResponseBasic|Entity=Order"
  ResponseError: !include "include/components/response/error|basic=ResponseBasic"
 examples:
  Order:
    value: !include "include/components/model/example/order"
  Order-Create:
    value: !include "include/components/model/example/order-create"
  ResponseOrder: !include "include/components/response/example/entity|Entity=Order|example=order|code=200"
  ResponseOrderArray: !include "include/components/response/example/entity_array|Entity=Order|example=
order_array|code=200"
  x-error-examples: !include "include/components/response/example/error"
include_add:
 'components.examples.x-error-examples': 'components.examples'
```

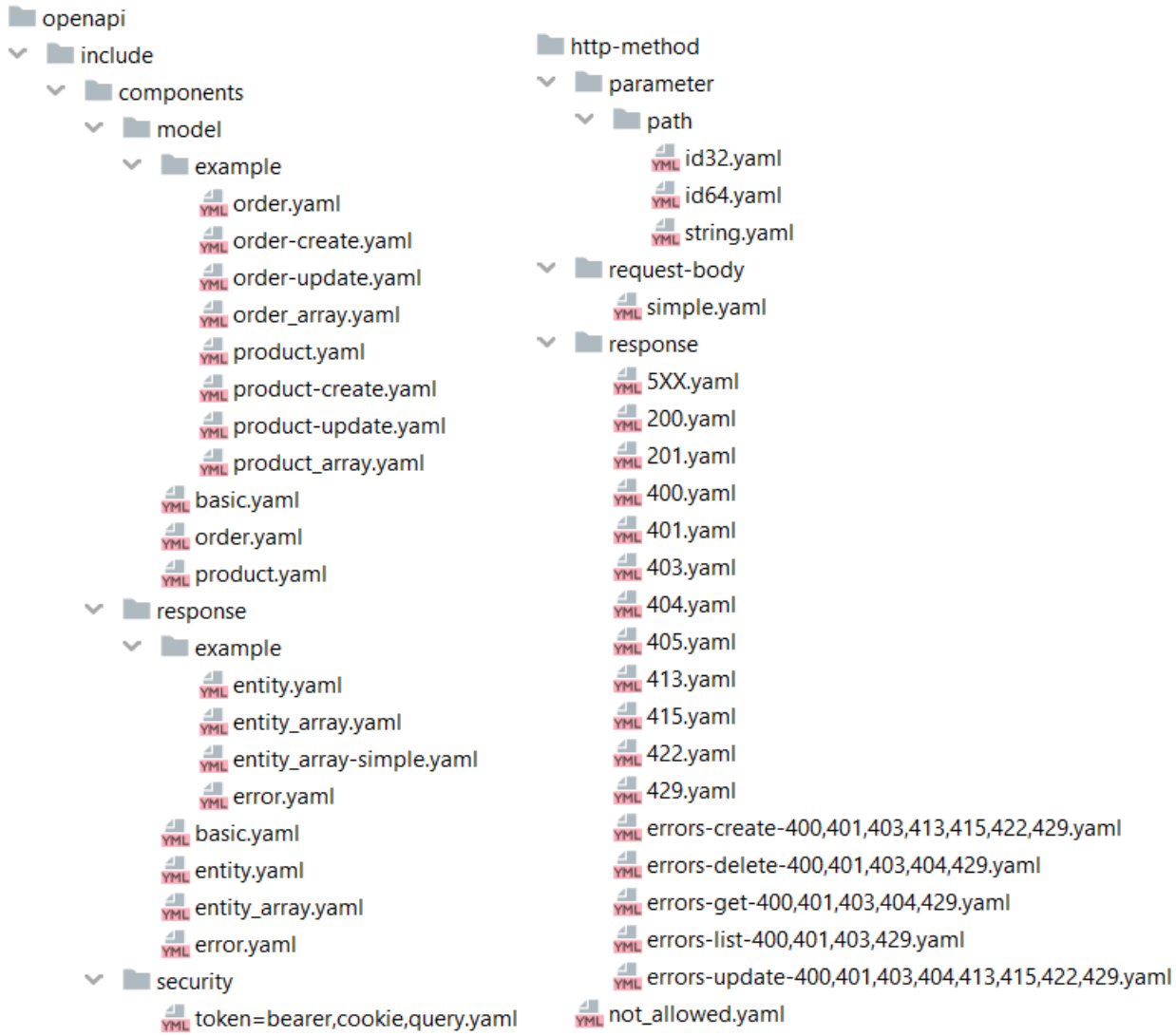


Fig. 8. The structure of the documentation snippets

V. IMPORTANT NOTES

To ease the deployment of the described environment for the development of API gateway, it is recommended to create a bash script containing commands for:

- creating an initial directory structure,
- creating a framework for generating API documentation according to OAS 3.0,
- downloading the current version of Swagger UI using the wget utility and changing its *index.html* file in accordance with the recommendations above,
- creating a framework for end-to-end and performance tests (for example, using Composer and PHP Unit Framework).

All files with content, such as yaml builder, and other static data can be stored in this bash script as base64 code from their archive, for example, in tar.xz. Also, the resulting single bash script can be posted in the Git repository for version history and constant updates.

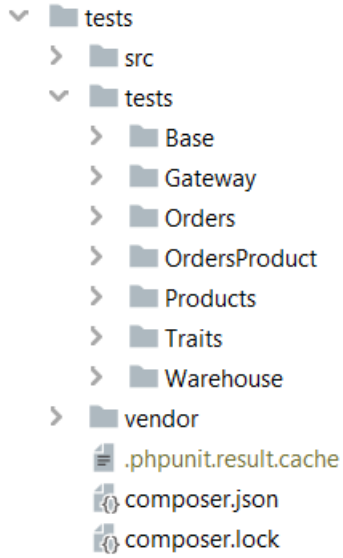


Fig. 10. Structure of end-to-end tests based on PHP Unit Framework

VI. CONCLUSION

The article developed an approach to deploying an environment for developing and executing an API gateway, taking into account: convenient configuration and launch of the gateway server, flexible development of API documentation and convenient viewing by API end users, as well as end-to-end and performance testing of developed APIs.

In the course of solving the problem of flexible and convenient development of API documentation according to OpenAPI Specification 3.0, a PHP yaml file builder was developed for calling via CLI. It is shown for what purposes it serves and why it is necessary.

A demo example is given and its implementation features are described.

A mechanism for quick deployment of all necessary files and directories is proposed according to the approach based on a single bash file.

CONFLICTS OF INTEREST

The author declares that there is no conflict of interest regarding the publication of this article.

REFERENCES

1. "APIs Have Taken Over Software Development", Nordic APIs, (October 27, 2020), https://nordicapis.com/apis-have-taken-over-software-development/.
2. "20 Impressive API Economy Statistics", Nordic APIs, (February 9, 2022), https://nordicapis.com/20-impressive-api-economy-statistics/
3. "The State of the Internet Reports", Akamai, (accessed April 19, 2022), https://www.akamai.com/our-thinking/the-state-of-the-internet
4. "The status of Open API adoption", TM Forum Inform, (accessed April 19, 2022), https://inform.tmforum.org/insights/2021/05/the-status-of-open-api-adoption/
5. "API Management Market by Solution & Services - 2023", MarketsandMarkets, (accessed April 19, 2022), https://www.marketsandmarkets.com/Market-Reports/api-management-market-178266736.html
6. "Open API Introduction", TM Forum, (accessed April 19, 2022), https://www.tmforum.org/oda/about-open-apis/
7. "Open Digital Architecture", TM Forum, (accessed April 19, 2022), https://www.tmforum.org/oda/
8. "nginx news", nginx, (accessed April 19, 2022), https://nginx.org/
9. "OpenResty - Official Site", OpenResty, (accessed April 19, 2022), https://openresty.org/en/
10. "API Resources", Swagger, (accessed April 19, 2022), https://swagger.io/resources/open-api/
11. "logrotate(8) - Linux man page", die.net, (accessed April 19, 2022), https://linux.die.net/man/8/logrotate
12. "REST API Documentation Tool", Swagger UI, (accessed April 19, 2022), https://swagger.io/tools/swagger-ui/