UDC 004.415.2
DOI https://doi.org/10.32838/2663-5941/2021.6/19

Oleshchenko L.M.
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

Movchan K.O.
Ukrainian Scientific and Research Institute of Special Equipment and Forensic Expertise
of the Security Service of Ukraine

SOFTWARE METHODS OF GENERATING DOCUMENTATION FOR NETWORK API

During software development and testing, a lot of time is spent analyzing code, domain logic, writing tests and documentation. Tests and documentation are important artifacts, and if they are high quality, significantly reduce the time spent by developers on the analysis of domain logic and program code. When developing client-server software, the quality of documentation for network APIs that provide access to data is important, because in some cases it is a mechanism by which the interaction between the commands of the client and server part of the software is organized. Creating and maintaining quality documentation using existing methods is time consuming, so the problem of optimizing and finding new methods to create documentation for the API is very important. The practical value of the research results obtained in this work is that the proposed method of creating documentation for network APIs allows to spend less time creating documentation, significantly reduces the likelihood of errors in the documentation caused by the human factor. The article analyzes the standards for creating documentation for software and substantiates the relevance of the problem for the API standard, formed requirements for documentation, analyzed the existing methods of creating documentation, identified their main advantages and disadvantages and compliance with the requirements. The implementation of the method of creating documentation based on tests is presented, its main advantages and disadvantages are revealed. Features of testing of a method and comparison of its efficiency in comparison with other methods are also considered.

Key words: *software methods, documentation, API, tests, OpenAPI documentation.*

Problem statement. The most important requirement for any technical documentation is its relevance. In the case of software development used cascade model this requirement is very easy to meet. But today few people use this model, instead prefer flexible models, which are quite dynamically changing software requirements. In this case, the documentation needs to be updated as dynamically as the software changes. This problem is especially acute for API documentation. In the vast majority of cases, separate commands work on the client and server part – sometimes they are territorially far from each other and do not have the opportunity to interact with each other. That is why documentation is a mechanism by which the interaction between these teams is organized, and its relevance and quality is an important factor that affects the speed of development and quality of the final product.

An individual employee can create and maintain API documentation. An employee will distribute this documentation in the form of files or publish it on the website. This approach has a number of significant shortcomings in terms of the human factor and resource consumption. A person can forget something, write incorrectly, etc. Once the code has changed, the documentation is out of date. Developers are faced with outdated and erroneous API documentation very often, which takes extra time, which ultimately affects the cost of products. Thus, the urgency means the elimination of the "human factor". Documentation should be updated automatically or semi-automatically with changes to the API, and the ability to forget to make changes to the documentation should be kept to a minimum.

The next requirement for documentation is its interactivity. This means that we must provide a specific user interface that not only describes the API in a readable form, but also allows to execute requests to the server side. This is important because usually API users who are the developers of the client side, before writing the code, still make requests to the server part to either check the API's performance or see how the API behaves in situations not described in the documentation. With a graphical interface, we eliminate the need for developers to use third-party software such as Postman for such purposes. Also, if manual testers are involved in the software development process, they will be able to test the

server part separately and it will be easier for them to "find the extreme" (client or server part) if the program does not meet the requirements, because testers using interactive documentation will be able to check the correct operation of the server part independently. Another advantage of interactivity is that they actually eliminate situations where the documentation does not fully describe possible answers or error codes, because in this case, without much effort, API users will be able to check such situations themselves.

Related research. The following software methods are most often used to create and maintain documentation for the REST API [1-10]:

– creating documentation using tools for a specific programming language / framework;

– writing documentation manually;

– creation of documentation with the help of third-party utilities;

– creation of documentation based on tests.

The main goal of the article is to create software methods that allows to optimize and automate the process of creating documentation for the network API.

An overview of existing software methods. Currently, the most commonly used application programming interfaces are GraphQL, JSON RPC, SOAP and REST API.

GraphQL is a standard for declaring the structure and methods of obtaining data or syntax, which describes how user can read data from the server [1].

GraphQL has three main characteristics:

– allows the client to specify exactly what data he needs;

– facilitates aggregation of data from several sources;

– uses a type system to describe data.

This approach, in addition to flexibility, reduces the number of requests and the amount of data at the transport level. The GraphQL API is based on three main building blocks: schemas, queries, and resolvers. GraphQL provides the following types of operations: request (data reading), mutation (data recording) or subscription (continuous data reading) [2]. Any of these operations is simply a string that must be compiled according to the GraphQL query language specification. Once such an operation comes to the server from the client program, it can be interpreted using the entire GraphQL schema and provide the data required by the client application. GraphQL can work with any high-level network protocol (most often using HTTP) and with any data format (usually using JSON).

The advantage of using GraphQL is declarativeness. Also, the advantages include strong and clear typing, no problems with versioning. Another important aspect of GraphQL is its hierarchical nature. GraphQL is built on the relationship between objects, which simplifies the formation of queries, where the RESTful service may need a multiple query system "request / response" or a complex merge operation in SQL [3]. The main disadvantage is considered to be the complexity of implementation on the server side. Typically, this is why GraphQL is used as an additional layer between the client and web services. In this case, web services do not use GraphQL, but provide access to data using the REST API. The task of automating and optimizing the process of creating documentation is irrelevant, because the above-mentioned GraphQL-scheme is documentation for users of network API [4].

JSON-RPC is a remote procedure call protocol that uses JSON as the data format. This protocol is very similar to XML-RPC, its specification defines several types of data and rules for their processing [5]. It is designed as a simple, flexible and understandable

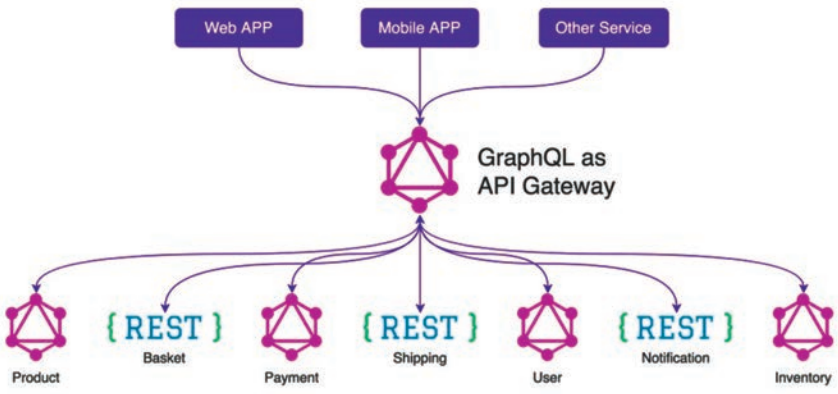

Fig. 1. GraphQL example in microservice architecture [1]

standard. JSON-RPC is based on sending requests to a server that implements a remote protocol. All transmitted data are requests serialized in JSON. A query is a call to a specific method provided by a remote system. It must contain three mandatory components:

– "method" is a line with the name of the method;

– "params" – data transmitted to the method as parameters;

– "id" – a value of any type used for matching the request to the answer.

The server must return a response to each request received. The answer should contain the following properties:

– "result" – data returned by the method. If an error occurred during execution of the method, this property must be set to null;

– "error" – error code in case of error during execution method, otherwise null;

– "id" is the same value as in the query to which it belongs reply.

Notifications have been introduced for situations where no response is required. Notifications differ from the request by the absence of "id". The main advantage of JSON-RPC is its simplicity and intuitiveness. Often, when developing APIs, programmers who know nothing about standards themselves design interfaces with a similar query and response structure. JSON-RPC is well suited for web services with a small amount of functionality and data types. However, the lack of caching and versioning mechanisms, the lack of a clear specification make this standard unsuitable for large-scale web services. The JSON-RPC standard is very simple, so it is a simple task to generate documentation for APIs that use this standard. In particular, all that the documentation should contain is a list of methods, parameters, answers and error codes. Implementations of this standard for different platforms cope well with this task. In particular, it is JSON-RPC. NET for the .NET platform, go / net / rpc for GoLang, php-json-rpc for PHP [6].

SOAP is a protocol for exchanging structured messages in distributed computing systems [7]. For SOAP, there is no difference between calling a procedure and answering a call, it simply defines the message format as an XML document.

The message may contain a call to the procedure, a response to it, a request to perform some other action. The SOAP specification does not use message content analysis, it only specifies its standard for its design. SOAP is based on the XML language and extends one of the application layer protocols – HTTP, FTP, SMTP, etc. As a rule, HTTP is most often used. A SOAP message is an XML document that consists of three main elements: an envelope (SOAP Envelope), a header (SOAP Header), and a body (SOAP Body).

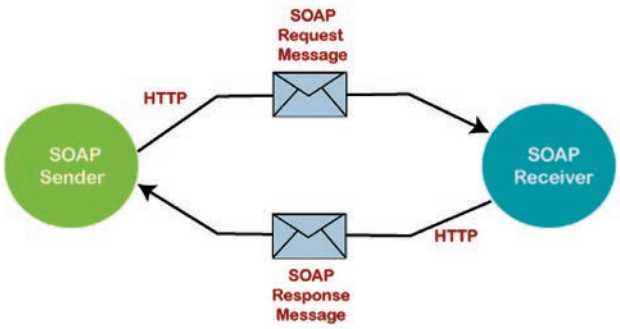

Fig. 2. Messaging between client and server using SOAP [7]

REST is an approach to the architecture of network protocols that provide access to information resources [8; 9]. It was described and popularized by Roy Fielding, one of the creators of the HTTP protocol. Fielding developed REST in parallel with HTTP 1.1 based on the previous version 1.0 [10]. REST has several architectural limitations. One of the limitations is the client-server architecture. This type of architecture requires a division of responsibilities between the components that store and update data (server) and those components that display data on the user interface and respond to actions with this interface (client). This separation allows the components to work independently. The next limitation is that the interaction between the server and the client does not have a state, ie each request contains all the necessary information for its processing, and does not rely on the fact that the server knows something from the previous request.

An additional limitation of the REST style is that systems written in this style must support caching, ie the data transmitted by the server must contain information about whether they can be cached and, if so, for how long. This allows to increase performance while avoiding unnecessary queries, but also reduces the reliability of the system due to the fact that the data in the cache may be outdated. REST API is a set of URIs, HTTP calls to these URIs, and a large number of resource views in JSON or XML format, many of which will contain cross-references. Addressing is based on covering resources with unique identifiers. Restrictions on interface uniformity are partially implemented through combinations URI and HTTP verbs and their use according to standards and conventions. Resources must be nouns, and action on a resource is a verb. The URI should always refer to the resource, not the action.

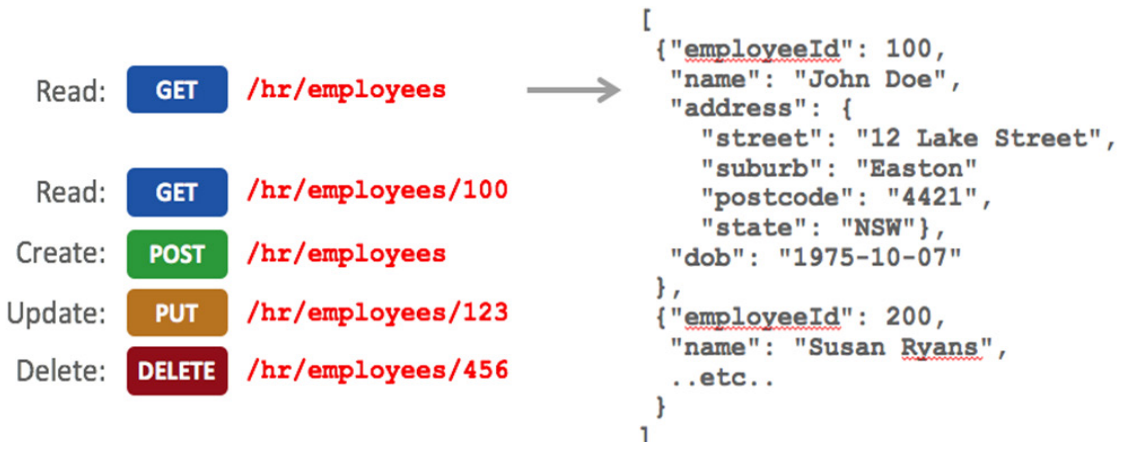

Fig. 3. A typical resource and a set of URIs access [10]

Each service resource must have at least one URI that identifies it. URIs should have a simple hierarchical structure to facilitate understanding of the API, and as a consequence, its usability. The REST API documentation should include a set of all resources, their identifiers and views, as well as a set of URIs and HTTP verbs for accessing resources, authorization information, and possible error and response codes. The REST API specification itself does not provide any automatic mechanisms for creating documentation, as is the case when using GraphQL using a schema, or in SOAP using WSDL. At the same time, the documentation for the REST API is an important artifact for the client part, and the process of its creation and maintenance can be optimized by analyzing the requirements and methods in detail.

The proposed software method. The test-based method is based on generating documentation when running functional tests. The result is a markdown file or a set of HTML and JS files. With the system of continuous integration, it is easy to configure the automatic download of these files to the server, which will allow each member of the team to easily view the documentation in the browser. The main advantage of this method is that it almost completely ensures the relevance of the documentation. After all, with the addition of tests, changes are automatically made to the documentation. The file that results from the tests determines whether the GUI will be interactive.

If it is a swagger file, we can install any graphics client that can work with the OpenAPI standard. Otherwise, it will not be possible to execute API requests using documentation. The SpringRestDocs library, which is an implementation of this method, generates documentation in its own format and does not provide users with interactivity, which is its significant disadvantage. Since the documentation is based on tests, this method requires time to write them. At the same time, API access points are usually covered by functional tests, and under such conditions this method does not require additional time to create and maintain documentation.

This method does not completely minimize the human factor, because we need to make changes to the templates every time, and this can be easily forgotten. Also, the process of setting up SpringRestDocs is quite complex and cumbersome, but this can be explained by the complexity of the Spring ecosystem. The main disadvantage of SpringRestDocs is the lack of interactivity in the documentation. The documentation generated by the library is shown in Fig. 4:

Although API users will be able to see a list of available resources and methods by following the link, they will still be forced to use third-party applications to begin developing the client part. It should also be noted that an additional reason for this is that SpringRestDocs does not know how to generate the code of the client part, only snippet in the specified format. Another disadvantage is the inability to specify the authorization principle for the API. There are currently several ways to authorize in the REST API. These include BasicAuth, BearerAuth, ApiKeyAuth, OpenID and OAuth2. If we use any of them in tests, we will have to duplicate the queries for each of the tests. All these shortcomings are eliminated by using the OpenAPI standard. The OpenAPI specification, originally known as Swagger, is a specification of machine-readable files with interfaces for describing, creating, using, and visualizing REST web services. The principle of using this specification in this method is that instead of using snippets, templates and end files, we must first generate a machine-readable Swagger file, and then convert this file into human-readable documentation. This can result in a

PDF, HTML, etc. file. The final HTML file contains javascript code that allows to log in to the API and send requests to the server directly in the browser. Its appearance for the user in the browser (Fig. 5):

This method does not require manual editing of templates, but it will not be possible to change this template at the same time. The OpenAPI specification also supports all common authentication methods. All we have to do is add a securitySchemes section to the Swagger file. In this case, our method should not be responsible for graphical visualization of documentation. The result of his work is a Swagger file. The programmers will choose the way of visualization, and when the Swagger file will be generated, because it can be done in several ways. In this case, in the system of continuous integration, we have the opportunity to check the percentage of coverage by tests. After successfully running the tests using the same system of continuous integration, the file is sent to a server that stores such files and visualizes them. The role of the server can be played by the project itself. In the case of a microservice architecture, we can create a separate service that will do this.

The principle of this method is to add a new handler that implements a special interface to the process of generating snippets, which later create templates,

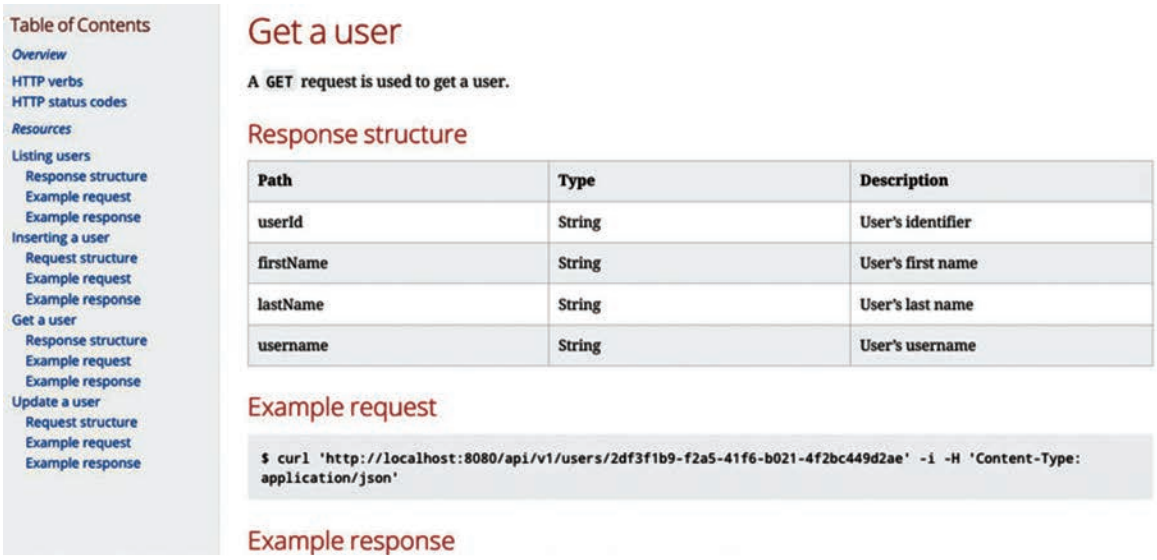

Fig. 4. Documentation generated in SpringRestDocs

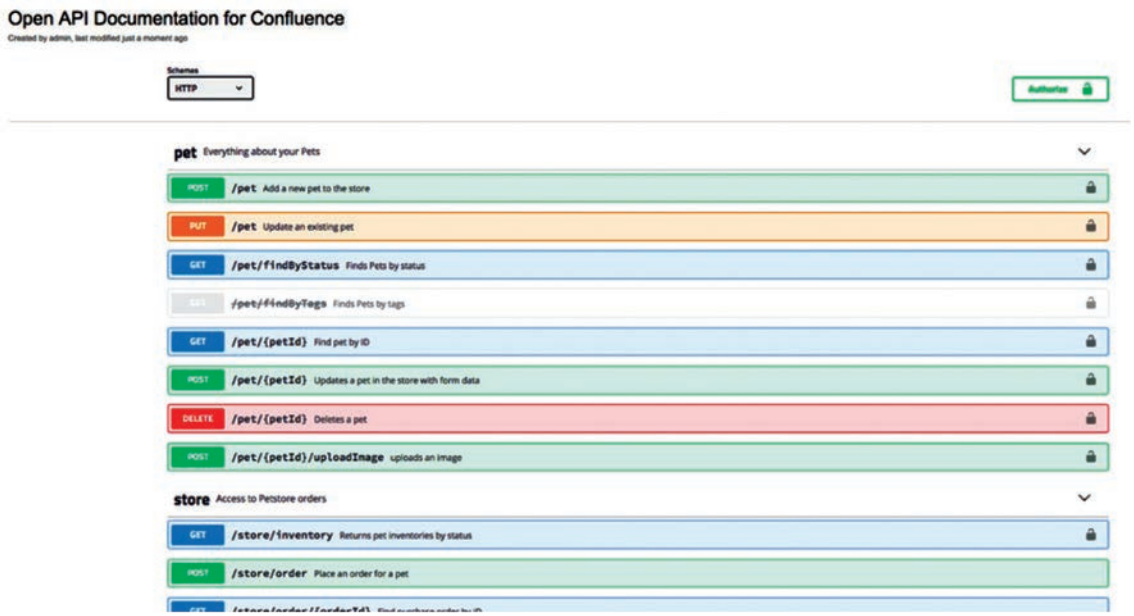

Fig. 5. OpenAPI documentation

and then generate the result as a file in ePub, PDF or HTML, depending on the configuration. The library implements functionalities, such as adding links, adding descriptions for input and output parameters, input headers, and so on. But the current implementation has a number of significant drawbacks. The biggest drawback is that the final documentation is static rather than interactive. There are also no mechanisms to specify the method of authorization, which leads to duplication of code in the tests. All these shortcomings are eliminated by the new principle of operation of the method. In particular, instead of generating snippets, a Swagger file will be generated, which complies with the OpenAPI specification, in which these shortcomings are taken into account and eliminated.

Conclusions and future work. In this research existing software solutions for generating documentation for network API are analyzed. Requirements for the developed software are formed and defined. The REST API documentation generation method is currently implemented in the SpringRestDocs library, which is part of the Spring ecosystem.

The software implementation of the proposed method was covered by automated tests written using the PHPUnit library. The total percentage of code coverage by tests reaches 65%. The efficiency of the method in comparison with other methods and the existing implementation of the method on the basis of tests are analyzed. The proposed method provides documentation interactivity compared to Spring Rest Docs and requires 25% less time than the method of creating documentation tools for a specific framework.

References:

1. Learn GraphQL. URL: https://graphql.org/learn
2. GraphQL Core Concepts Tutorial. URL: https://www.howtographql.com /basics/2-core-concepts/.
3. GraphQL. URL: https://meline.lviv.ua/development/other/graphql/
4. Wieruch, R. The Road to Graphql: Your Journey to Master Pragmatic Graphql in JavaScript with React.Js and Node.Js. New York : Independently published, 2018. 314 p.
5. JSON-RPC 2.0 Specification. URL: https://www.jsonrpc.org/specification.
6. Implementations – JSON-RPC. URL: https://www.jsonrpc.org/archive_json-rpc.org/implementations.html.
7. SOAP Version 1.2 Part 0: Primer (Second Edition). URL: https://www.w3.org/TR/2007/REC-soap12-part0-20070427
8. WSDL Web Services Description Language. URL: https://www.guru99.com/wsdl-web-services-description-language.html/
9. Architectural Styles and the Design of Network-based Software Architectures. URL: https://www.ics.uci.edu/~fielding/pubs/ dissertation/top.htm
10. History of REST APIs. URL: https://www.mobapi.com/history-of-rest-apis.

Олещенко Л.М., Мовчан К.О. ПРОГРАМНІ МЕТОДИ ГЕНЕРУВАННЯ ДОКУМЕНТАЦІЇ ДЛЯ МЕРЕЖЕВИХ ПРИКЛАДНИХ ПРОГРАМНИХ ІНТЕРФЕЙСІВ

Основною проблемою розроблення та тестування програмного забезпечення є витрачання великого об'єму часу на аналіз коду, доменної логіки, написання тестів та створення документації. Якісні тести та документація суттєво зменшують часові витрати розробників на аналіз доменної логіки та програмного коду. При розробленні програмного забезпечення з клієнт-серверною архітектурою важливою є якість документації до мережевих API, що надають доступ до даних, оскільки у деяких випадках це є механізмом, за допомогою якого здійснюється взаємодія між командами клієнтської та серверної частини програмного забезпечення. Створення та підтримка якісної документації за допомогою наявних методів потребує багато часу, тому проблема оптимізації та пошуку нових методів для створення документації до API є актуальною. Практична цінність результатів дослідження, що були отримані в даній роботі, полягає в тому, що запропонований метод створення документації для мережевих API дозволяє витрачати на генерування документації менше часу, зменшує ймовірність появи помилок у документації. У статті проаналізовані існуючі стандарти створення документації для програмного забезпечення та обґрунтована актуальність проблеми генерування документації для стандарту REST API, сформовані вимоги до документації, проаналізовані наявні методи створення документації, виявлено їх основні переваги, недоліки та відповідність сформованим вимогам. Наведено реалізацію методу створення документації на основі тестів, виявлено її основні переваги та недоліки. Також розглянуті особливості тестування методу та проведене порівняння його ефективності у порівнянні з іншими методами.

Ключові слова: *програмні методи, API, тести, OpenAPI документація.*