Journal of Theoretical and Applied Information Technology
30th June 2019. Vol.97. No 12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

STUDYING OPEN BANKING PLATFORMS WITH OPEN SOURCE CODE

ANDREY KOLYCHEV, KONSTANTIN ZAYTSEV

National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Kashirskoe Avenue 31, Moscow, 115409, Russia

ABSTRACT

Intensive growth of public web interfaces started early in 2010; and if initially API was a procedure of interaction of various software tools, then at present web interfaces are genuine digital products on the basis of which companies, especially major companies, can derive profits while providing their internal services to third parties via open API. Banks are not an exception. They also can derive profits by providing access to their internal services for third-party developers. The advantage of banking enterprises is that they possess unique data and services, which can hardly be competed. As a consequence, there appeared the software market for the development of open source API and provision of access to them with monetization capabilities. API management platform is comprised generally of three components: developer site, API development tools, and API gateway. API gateway is the most important component since it is responsible for interface operation; hence, this work is aimed at the determination of the most efficient API gateways. Three software variants have been considered: Gravitee API Platform, APIMan, and WSO_2 API Manager, which meet two preset criteria: Java product implementation, open source code of the product. The study has been performed in comparison environment with three coordinates: intensity of performed functions for API development, labor intensity of API implementation, the performance of API gateway. During the experiments, Gravitee.io API Platform was the best software with regard to each coordinate.
Keywords: *API Management, API Management System, API Platform, API Manager, API Gateway, Open API, Software Functionality, Performance*

.

1. INTRODUCTION

At early stages of software development, it was necessary to solve the problem of interaction among applications in order to provide possibility of data exchange overriding physical and logical boundaries. Integration of various software products is peculiar for numerous business scenarios. The number of integrating interactions continuously increases, this is stipulated by sophisticated ecosystems and business processes which are supported by complex interactions with several endpoints in user software, internal software of various companies and various public services. One of the variants of software interaction, especially in the case of various logics and architecture, is API.

According to data by ProgrammableWeb service, the number of open web interfaces from the early 2010 increased by about 20 times [1] (Fig. 1).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

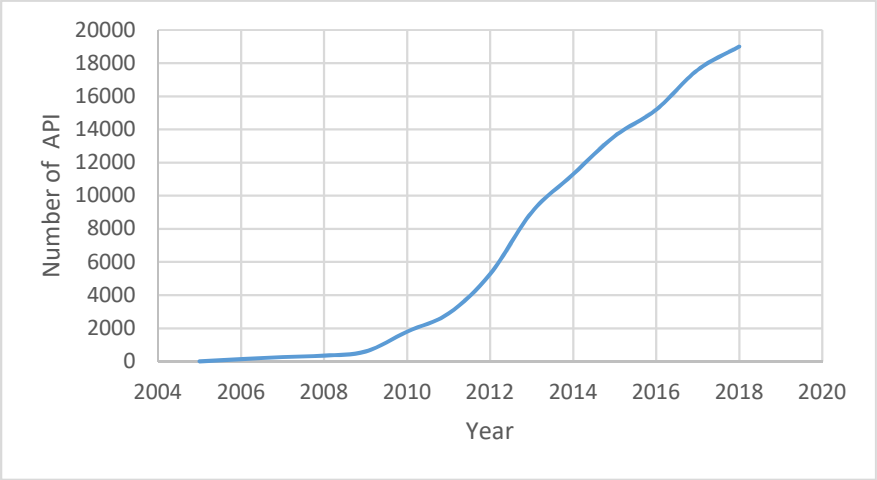

Figure 1: Quantitative variations of open interfaces

It should be mentioned that in addition to hi-tech industries, open interfaces are also applied in such fields as telecommunications, mass media, travelling, tourism, and real estate. Major financial market players analyze open banking platforms in order to compete with IT giants, which already provide their financial services such as PayPal, Billtrust, Amazon. In addition, if initially API was a method of interaction of various softwares, then at present web interfaces are genuine digital products, on the basis of which companies, especially major companies, can derive profits [2]. Banking enterprises are not exceptions. In European banking sector, development of open API is a requirement stipulated in PSD2 directive. Therefore, banks are stimulated to develop open API by two forces: market and law. Hence, each modern bank aiming at competitive business should develop open API. The concept of open interfaces is not new, therefore, numerous software solutions are available in the market for development of open interface infrastructure [3], which leads to selection of efficient system of API management. This work is devoted mainly to API gateways with open source code.

This problem is considered by several researchers. They apply various approaches to comparison of software products. Some works are based on customer opinions [4-6], such criteria are highlighted as functional capabilities of various components of API platform, estimations of support services, usability, software cost. Other studies combine estimations by users and experts [3, 7], various criteria are also highlighted. Nearly all researches [3, 6, 7] include such criterion as presence of software platform in the market (amount of clients and geographical distribution of software). All studies consider mainly paid solutions; this work analyzes platform with open source code and compares the main component of API platform: API gateway.

2. MATERIALS AND METHODS

2.1. Selection of software products for comparison

API management systems are comprised conventionally of three components: API gateway, API manager, developer site.

API gateway is a network gateway (or web server, if it is not required to combine network segments, for instance, Internet and intrabank network) where source code of developed API is physically located. Requests to API are addressed exactly to API gateway, where authentication and authorization are carried out, validity of the request in accordance with tariff plan is verified, the request is handled according to policies described in API, transformations are carried out, then the request is directed to bank internal systems, where the handling is performed according to these systems, and API gateway receives response from bank internal systems, this response can also be handled and transformed, then the response is returned to the application which requested API. API gateway is the most important component of API management system since it provides availability of API.

API gateways are subdivided into test and production ones (it can be one and the same physical gateway), test gateway contains the same API but without request to bank internal system, the requests are responded by stubs simulating operation of bank internal systems, this is required

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

in order to facilitate API user to adjust application using test data prior to paying for actual data.

API manager provide possibility to determine API using any notation, for instance, OpenAPI or RAML, as well as policies applied during request or response to API and arbitrary handlers. As a rule, API gateway can be configured using the same tool.

Tools of API publication are required for development of tariff plans and API binding, this tool also controls access to API.

Developer site is an Internet portal where third party developer can evaluate API and relevant specifications, to register application, to subscribe for API according to certain tariff plan, in addition it would be possible to make test request directly from the portal page [8].

Software products were selected for analysis on the basis of the following criteria important for subsequent use:

- the product should be implemented in Java;
- the product should have open source code.

Initial selection of software products in this field was based on analysis of publications about API management systems. Numerous API management systems were detected during the study, such as: CA API Management, Apigee, IBM API Connect, Mulesoft Anypoint API Manager, Microsoft Azure API Management, Akana API Management, 3scale API Management, OpenLegacy, Apiary. Then, using the aforementioned criteria, the software products were selected which satisfied these criteria. These are three variants with open source code for API management: Gravitee.io API Platform [9]; APIMan [10], and WSO_2 API Manager [11].

2.2. Selection of coordinates of comparison environment

The most important properties of each software product are intensity of performed functions and performance, that is, the ratio of performed work to the time of its execution. Since the given software products are used also for API development, then it is required to define the list of possibilities provided for such development. In addition, it is required to understand how readily and rapid such interfaces can be developed.

Here and below the software functions are interface policies. Policy is a unit work executed during request to API. When during execution an API call is carried out, a chain of policies is created and applied to incoming request (or outgoing response) prior to transfer of this request to implementation by internal API.

Considering this, the following coordinates of comparison environment were selected:

1) intensity of functions of API management systems for development of interface;
2) performance of software product;
3) labor intensity of API development.

The software products were compared with regard to the following properties:

- possibility of request transformation (modification of its body, access to request parameters);
- possibility of transformation of request body format;
- possibility to execute additional network call within API;
- possibility to develop proper arbitrary handlers;
- modification of HTTP method (API call is made using one method and system call behind API using another method);
- possibility of error handling.

These criteria were selected on the following basis. Possibility to transform request body or its format often occurs when bank internal systems intend to work using request formats differing from those proposed by API. Possibility to make additional call within API is necessary for implementation of complex scenarios of interface operation where one interface includes interaction with several bank internal systems. Possibility to develop proper arbitrary handlers is important because despite numerous possibilities of initial function library there comes a point of time when it is required to determine proper policy with unique behavior. Replacement of HTTP method is necessary when bank internal system by any reasons should receive requests using a method differing from that proposed by API. Possibility to handle errors is important because interfaces contain software code where exception cases are inevitable and should be handled in a particular manner.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Cumulative estimation of each product was calculated as follows:

$$S = 100 \cdot \sum_{i=1}^{n} V_i \cdot Z_i \quad (1)$$

where S was the estimation of tool; n was the number of comparison criteria; Z_i was the value criterion execution; V_i was the criterion weight (from 0 to 1).

The following scale was proposed to estimate criterion values: 0 - if a criterion was not fulfilled or fulfilled by the third party software; 1 - if a criterion was fulfilled completely or with minor restrictions; 0.5 - of a criterion was fulfilled with significant restrictions.

2.3. Comparison of performances

Software products were verified according to the following scenario. API was developed which during incoming request performed several outgoing HTTP calls, thus emulating complex scenario of API operation where API not only redirected external call into internal system but also performed additional request to internal system as well as enhanced data or performed any other verification or calculations. In addition, long operation of this internal call was simulated: five second delay was programmed. Simple service written in Java was developed as internal system, which responded with five second delay. Prior to performance testing, this service was tested with respect of its operability under selected load. In order to perform comparison, the considered systems were deployed, similar API in terms of functionality were developed, then requests were sent in several threads and the response time was measured. All systems were deployed using Docker virtualization program on the basis of official images. APIMan software was considered in gateway implementation using Vert.x platform since it was used for implementation of Gravitee management system.

The performance of Gravitee API gateway was tested using “Groovy” policy where HTTP call was implemented by Groovy HTTP client embedded into programming language which was not absolutely correct but nevertheless did not result in performance loss by this gateway.

In order to test performance of of APIMan gateway written in Java, the policy was developed and added to the gateway where Java HTTP client was used, since the HTTP client embedded into APIMan supported only asynchronous operation, i.e. upon HTTP call, execution of subsequent policies was not blocked, which was not supported by the test scenario.

In WSO_2 management system, the API handlers are implemented by other WSO_2 software: ESB (service bus) with specific xml notation. The code used in WSO_2 management system is exemplified below:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Declaring handler sequence -->
<sequence xmlns="http://ws.apache.org/ns/synapse" name="performance_test" trace="disable">
  <!-- Module of http call, duplicated for 5 times -->
  <call blocking="true">
    <!-- Determining http method and url -->
    <endpoint>
      <http method="get" uri-template="http://192.168.99.100:8888/stubFORAPIMan/ServletWithTimeout" />
    </endpoint>
  </call>
  <call blocking="true">
    <endpoint>
      <http method="get" uri-template="http://192.168.99.100:8888/stubFORAPIMan/ServletWithTimeout" />
    </endpoint>
  </call>
  <call blocking="true">
    <endpoint>
      <http method="get" uri-template="http://192.168.99.100:8888/stubFORAPIMan/ServletWithTimeout" />
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
    </endpoint>
  </call>
  <call blocking="true">
    <endpoint>
      <http                                    method="get"                                    uri-
template="http://192.168.99.100:8888/stubFORAPIMan/ServletWithTimeout" />
    </endpoint>
  </call>
  <call blocking="true">
    <endpoint>
      <http                                    method="get"                                    uri-
template="http://192.168.99.100:8888/stubFORAPIMan/ServletWithTimeout" />
    </endpoint>
  </call>
  <!-- Handler of response from previous module -->
  <payloadFactory media-type="json">
    <!-- Presetting pattern for message, in this case it is JSON -->
    <format>
              {
       "Data":{
         "PaymentSubmissionId":"$1",
         "PaymentId":"$1",
         "Status":"$1",
         "CreationDateTime":"$1"
       }
              }
    </format>
<!-- Variables are determined for input into the above pattern;
the $body variable contains only response body from service requested in
the block <call>...</call>, since the response format of xml service, then
the required poll is requested by XPath query language
  -->
    <args>
      <arg evaluator="xml" expression="$body//some" />
    </args>
  </payloadFactory>
  <!-- Forming header Content-Type -->
  <property name="messageType" value="application/json" scope="axis2" type="STRING" />
</sequence>
```

As can be observed, overall code is an xml configuration, which is not very convenient.

2.4. Comparison Of Labor Intensity Of API Development

In order to compare with respect to this coordinate, it was decided to implement test interface using each software product which would contain the following blocks: request transformation with possibility to modify request body and headers; error handling with possibility to generate message with preset error text in the case of error within interface, and in the case of error in bank internal system, to catch error with possibility to correct error message; additional HTTP call within API with possibility of its handling.

Implementation of each block was estimated using the following scale:

- 0, if implementation was impossible;
- 0.5, if implementation was labor intensive or had restrictions;
- 1, implementation was completely possible.

Cumulative estimation was calculated by Eq. (1), where S was the cumulative estimation of API management system; n was the number of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

blocks; Z_i was the estimation of labor intensity of block implementation; V_i was the weight of criterion (from 0 to 1).

The logics and order of handler execution in test API in more details was as follows: new header was added, some initial header was removed, request parameters were transferred to message body, HTTP call was performed by GET method and its result was added to current message body, possible code errors were handled, then the request was performed to assumed bank internal system, and then the error handler was executed comprised of replacement of message body if internal system returned HTTP codes 400 and 500.

test interface of gravitee system

Policy management in Gravitee system assumes addition of policies, which are applied upon interface call in the order of their addition, to API definition.

Addition and removal of headers can be carried out using the embedded policy "Transforms Headers", where the phase of policy application should be mentioned: request or response, header names should be mentioned which should be removed, name and value of headers to be added should be mentioned. In addition, access to headers can be obtained in "Groovy" policy, where Groovy programming language can be used to write arbitrary script, access to headers and message body is provided by means of context variables "request" and "response". Request parameters can be transferred to message body using "Groovy" policy and the following script:

```
//request parameters are obtained from context
def params = request.parameters();
//generating new body
def newBody = '<person>' + request.content +
  '<age>' + params.getFirst('age') + '</age>' +
    '<name>' + params.getFirst('name') + '</name></person>';
//returning result
return newBody;
```

Network call can be made by means of special policy "Callout HTTP", its response can be placed into context variable with subsequent access to it. (At the stage of performance test this policy was not developed, and HTTP requests were performed by "Groovy" policy.)

No special handlers or policies were stipulated in Gravitee for handling of errors occurring upon API operation, thus, in the case of error, the interface would return response with HTTP code 500. Error handling in policies implemented by Groovy can be performed by "try-catch" structure wrapping overall code with it, such as:

```
try {
//some code
}
 catch(Exception ex){
      //setting error state for policy
            result.state = State.FAILURE;
      //setting HTTP code
            result.code = 500
      //setting error text
            result.error = 'Interval Server Error'
      //returning empty string
            return ''
   }
 }
```

A peculiar feature is that the "Groovy" policy has four possible applications with respect to interface: request and response phase, each of them has two more variants: with and without access to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

request data; if the variant with access is selected, then the script should return any string result by means of key word “return”, hence, in the above example it is required to return at least empty string.

```
//Gravitee library for operation with interface state
import io.gravitee.policy.groovy.PolicyResult.State

String errorMessage = '';
int  statusCode = 0;

if (response.status  == 500) {
  statusCode = 500
  errorMessage = 'Interval Server Error'
}

if (response.status == 400) {
  statusCode = 400
   errorMessage = "Bad Request";
}
if (statusCode != 0) {
  //array with errors and their description
  def handlers = [ 400 : "Bad Request", 500 : "Internal server error"]
  result.state = State.FAILURE;
  result.code = statusCode
  result.error = '{"httpCode":' + '\"' + statusCode + '\"' +
  ', "httpMessage":'  + '\"' + handlers[statusCode] + '\"' +
  ', "moreInformation":' + '\"' + errorMessage + '\"' + '}'
  result.contentType = 'application/json'
}
```

test interface in apiman system

APIMan software, similar to Gravitee, has embedded policy for addition or removal of headers. In order to implement other modules of test interface, the policy was developed written in Java and added to API gateway. The policy is a Java applet, which should contain the class implementing *IPolicy* interface which contains two *apply* methods: request data are transferred to one of them, and response data – to another, the methods are executed at the stages of request and response, respectively. The class object method ApiRequest getQueryParams() was applied for access to request parameters which returned key value structure.

```
String name = request.getQueryParams().get("name");
String age = request.getQueryParams().get("age");
```

In order to add request parameters to request body, it is necessary that the policy could implement IDataPolicy interface; this is required for operation with request or response body.

Error handling can be implemented similar to the test interface for Gravitee system, that is, to use try-catch structure; execution of policies can be interrupted with returning error message to client by doFailure method to which the object should be transferred capable to describe all attributes of response message: HTTP code, message body, headers, for instance:

```
doFailure(new PolicyFailure(PolicyFailureType.Other,400,"BAD REQUEST"));
```

test interface in wso2 system

Handler of errors of internal system was implemented by “Groovy” policy as follows:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Handlers in WSO_2 management system can be applied in incoming flow, prior to message sending, after receiving response; in addition, the flow is stipulated to which control is transferred in the case of errors during execution of API code.

HTTP call module code in WSO_2 is similar to that described in Section 2.3, the module of response transformation is also implemented by the handler ***<payloadFactory>...</payloadFactory>.*** The modules of header transformation and transfer of request parameters to message body are as follows:

```
<!-- Adding header customHeader1 with the value -->
<header name="customHeader1" scope="transport" value="value"/>
<!-- Removing header customHeader2 -->
<header name="customHeader2" scope="transport" action="remove"/>
<!-- Request parameters are as follows: ?name=Jonn&age=40.
  Generating variable name, its value is the parameter name -->
<property expression="$url:name" name="req.var.name"/>
<!-- Generating variable age, its value is the parameter name -->
<property expression="$url:age" name="req.var.age"/>
 <!-- Using this handler we modify the message body and
    add request parameters to the body       -->
          <payloadFactory media-type="xml">
                    <format>
           <person>
                                  $1
                                  <name>$2</name>
                                  <age>$3</age>
                      </person>
                    </format>
            <args>
        <arg expression="//contacts" evaluator="xml"/>
        <arg evaluator="xml" expression="get-roperty('req.var.name')"/>
        <arg evaluator="xml" expression="get-  property('req.var.age')"/>
            </args>
          </payloadFactory>
```

Errors during API execution can be handled by special Fault Flow. Two embedded handlers are provided: json_fault and debug_ json_fault, the latter one logs more detailed information about error and will be useful at the stage of interface development. The errors in response can be handled by < filter>…</filter>, which is the if-else operator. For instance, in this way:

```
<filter source="get-property('axis2', 'HTTP_SC')" regex="400|500">
          <then>
            <payloadFactory>
                                <format>
              <!— Required format of error message -->
                            </format>
            </payloadFactory>
          </then>
</filter>
```

2.5. Generalization of results

After each comparison, ranks were assigned to the software products. The best product obtained rank 1, then followed rank 2, and etc.; if several tools obtained one and the same rank, then the rank was calculated using averaging equation (2)

$$r = \frac{\sum_{i=0}^{n-1}(r' + i)}{n} \quad (2)$$

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

where r was the total rank; n was the number of tools which obtained one and the same rank; r' was the rank which corresponded to all tools.

The comparison results were generalized by summation of ranks assigned to the tools in all comparisons, and then by ranking of the obtained summed ranks.

3. RESULTS

3.1. Comparison Of Software Products In Terms Of Intensity Of Performed Functions

The comparison results of software products with respect to this coordinate are summarized in Table 1. This information was obtained while studying functionality of the considered software products after their installation with consideration for their official specifications [12-14]. It is assumed that the functions summarized in the table already exist in software product; it is not said about their possible implementation and addition to the software. It is obvious that each software product provides possibility to develop proper handler and to implement it.

Table 1: Comparison of software products in terms of intensity of performed functions

Function		**Weight**	***APIMan***	***Gravitee.io API Platform***	***WSO₂ APIManager***
Request transformation	Transformation of headers	1/18	1	1	0
	Transformation of message body	1/18	0	1	0
	Transformation of request parameters	1/18	0	1	0
Transformation of request formats	XML into JSON	1/12	1	1	1
	JSON into XML	1/12	1	0	1
Possibility of additional request inside API		1/6	0	0.5	0.5
Development of proper arbitrary handlers		1/6	0.5	1	0.5
Error handling		1/6	0	0	1
Replacement of HTTP method		1/6	0	1	0
Sum of estimations, %			31	67	50

The sum of errors is calculated by Eq. (1).

Therefore, rank 1 can be assigned to Gravitee management system, rank 2 – to WSO_2 API Manager, and rank 3 – to APIMan. The results are illustrated in Figs. 2 and 3.

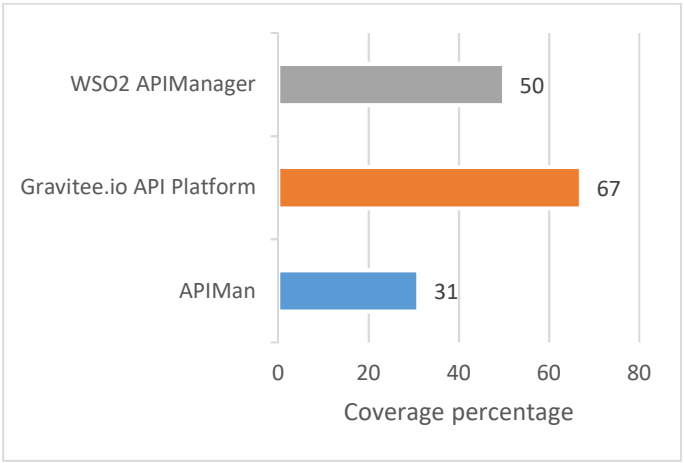

Figure 2: Intensity of functions performed by software products.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

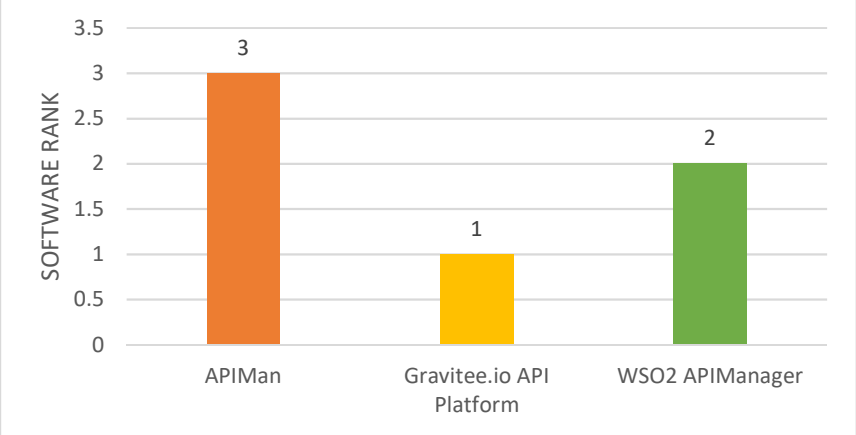

Figure 3: Ranks of software products according to comparison of intensity of performed functions (the less – the better)

3.2. Comparison Of Tools In Terms Of Performance

The main concept of comparison in terms of performance was determination of possibility to process operation scenario by the system where internal calls were executed within API. Initial testing was performed with default adjustment of API network gateway. The test results of Gravitee software are summarized in Table 2.

Table 2: Test results of Gravitee API gateways

Test scenario	***Average time of response to request, ms***	***Median, ms***	***Percentille 90, ms***	***Min, ms***	***Max, ms***	***Errors, %***
5 flows of 50 requests, 5 internal requests	25,140	25,040	25,050	25,027	37,517	0
10 flows of 50 requests, 5 internal requests	31,126	26,152	48,390	25,031	49,852	0

The obtained test results were expectable, since in average the requests were executed in slight excess of 25 seconds, and within API, five internal calls were executed, each in five seconds.

The test results of APIMan software are summarized in Tables 3 and 4.

Table 3: Tests results of APIMan API gateways

Test scenario	***Average time of response to request, ms***	***Median, ms***	***Percentille 90, ms***	***Min, ms***	***Max, ms***	***Errors, %***
5 flows of 50 requests, 5 internal requests	125,309	125,317	125,386	124,723	125,727	19

It can be seen in the table that significant portion of requests was terminated unsuccessfully (API gateway released connection), and successful requests were executed for longer time than expected (it was assumed that a request should be executed in slight excess of 25 sec because within API five internal calls were executed, each in five seconds). Then, in API gateway configuration file, the number of handlers was increased (by default, it was in "auto" state; and judging by log, only one handler was activated). The test results after increase of handler number are summarized in Table 4.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 4: Tests results of APIMan API gateways after increase of handlers

Test scenario	*Average time of response to request, ms*	*Median, ms*	*Percentille 90, ms*	*Min, ms*	*Max, ms*	*Errors, %*
5 flows of 50 requests, 5 internal requests	60,591	50,259	74,909	25,035	99,556	0

It can be seen in Table 4 that there are no error requests, however, the time of request execution exceeds the expected one due to unknown reasons. Variations in the number of handlers did not result in any qualitative changes. No other configuration tools were identified, thus, the analysis of this problem was terminated. In addition, it should be mentioned that this software product supports handlers with HTTP calls using components described by developers, however, such call can be only asynchronous, thus, JAVA HTTP client was used because synchronous call was required. At the same time, in Gravitee, HTTP call was executed by Groovy script embedded in Groovy HTTP client, which did not lead to problems with performance.

The test results of WSO_2 APIManager software are summarized in Table 5.

Table 5: Test results of WSO2 APIManager API gateways

Test scenario	*Average time of response to request, ms*	*Median, ms*	*Percentille 90, ms*	*Min, ms*	*Max, ms*	*Errors, %*
5 flows of 50 requests, 5 internal requests	25,098	25,089	25,164	25,035	25,305	0
10 flows of 50 requests, 5 internal requests	25,095	25,078	25,151	25,027	25,577	0

The obtained results are similar to those of Gravitee software testing: no unexplained delays, the results are expectable.

Based on the obtained results, it possible to conclude that Gravitee and WSO_2 APIManager software products are the best in this comparison, rank 1.5 could be assigned to them, and rank 3 could be assigned to APIMan management system. The results are illustrated in Fig. 4.

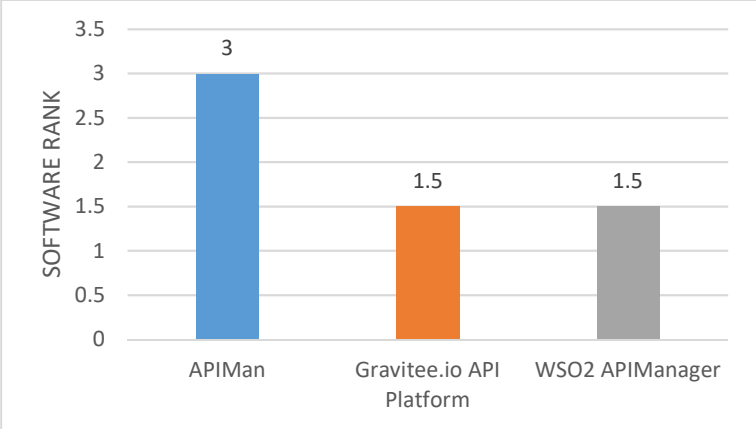

Figure 4: Ranks of software products according to comparison of performance (the less – the better)

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.3. Comparison in terms of labor intensity of API implementation

The respective comparison results are summarized in Table 6.

Table 6: Comparison of systems in terms of API implementation

Block		Weight	*Gravitee.io API Platform*	*APIMan*	*WSO_2 APIManager*
Block of request transformation	Transformation of headers	1/9	1	1	1
	Handling of request parameters	1/9	1	1	1
	Transformation of body	1/9	1	1	1
Block of error handling	Handling of API errors	1/6	0.5	0.5	1
	Handling of customer (4**) and server (5**) errors	1/6	1	1	1
Block of HTTP request	Direct HTTP request	1/6	1	0.5	0.5
	Handling of response	1/6	1	1	1
Sum of estimations, %			92	83	92

Based on the obtained results, it is possible to conclude that the best software products in this comparison are Gravitee and WSO_2 APIManager, thus, according to Eq. (2), rank 1.5 is assigned to them, and rank 3 is assigned to APIMan. The results are illustrated in Fig. 5.

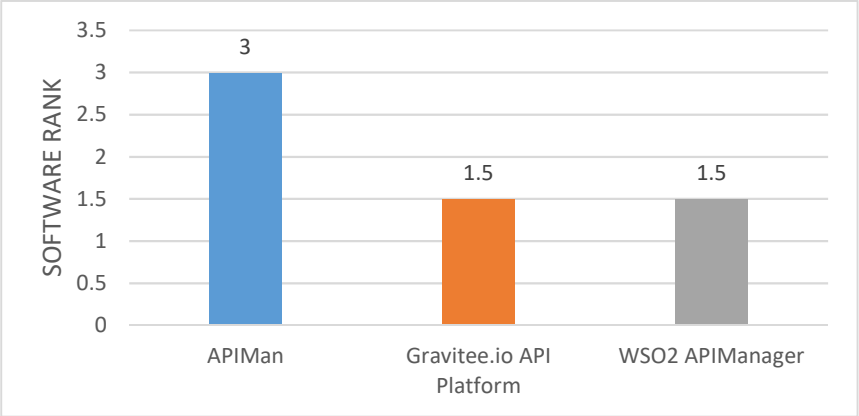

Figure 5: Ranks of software products according to comparison of labor consumption pf API implementation (the less – the better)

Gravite management system received only 0.5 due to complicated handling of API errors. It can be implemented only in "Groovy" policy, and it cannot be performed in other policies upon errors during their execution.

APIMan management system also lost one half due to implementation of API error handling similar to that described for Gravitee. Another one half was deducted for implementation of HTTP request, it was required to use Java client, and embedded code supported only asynchronous operation.

WSO_2 management system lost one half for implementation of HTTP request, because if a request was made at the stage of response in interface, then it was impossible to access to message body received after the request. Information about this event was unavailable in specifications.

From subjective point of view, Gravitee management system is characterized by lower labor intensity of implementation of the considered interfaces, all difficulties are related mainly with poor specifications. APIMan requires for self-development of policies with subsequent setting in API gateway, which is time consuming and also

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

requires for developer competences in Java development. WSO_2 software for implementation of policies uses specific and unobvious xml notation, which requires for knowledge of WSO_2 ESB.

3.4. Generalization of comparison results

The comparison results of API management systems in terms of all coordinates are summarized in Table 7.

Table 7: Comparison of API management systems

Comparison coordinate	*Gravitee.io API Platform*	*APIMan*	*WSO_2 API Management*
Intensity of performed functions	1	3	2
Possibility of interface implementation	1.5	3	1.5
Performance	1,5	3	1,5
Cumulative rank	4	9	5
Final rank	1	3	2

The results are also illustrated in Fig. 6.

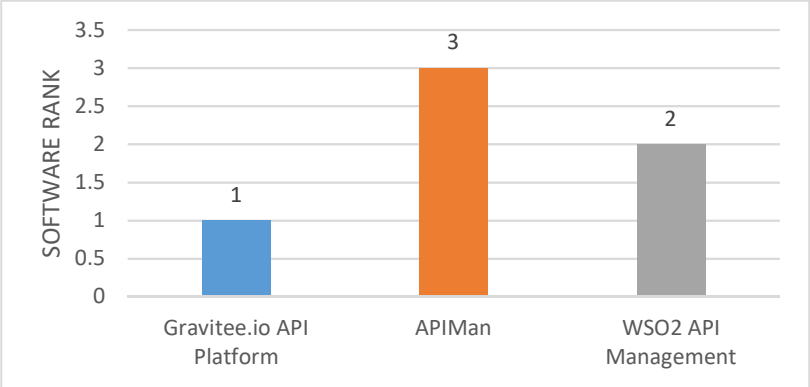

Figure 6: Final ranks of software products according to comparison (the less – the better)

Therefore, the Gravitee software is the most efficient product in the environment of preset criteria.

4. DISCUSSION

In this work, we analyzed API management systems with open source code implemented in Java. Some studies [3-7] consider mainly paid solutions, which are not suitable for everybody. Part of studies is based on user reviews [4-6], the following criteria are highlighted in these reviews: functional possibilities of various components of API platform, estimation of supporting services, usability, software cost, etc. Other studies combine estimations by users and experts [3, 7] also highlighting various criteria. Nearly all studies [3, 6, 7] include such criteria as presence of software platform in the market (amount of clients and geographical distribution of software). In total, the mentioned studies are of general character, which makes it possible to form comprehensive idea of each software product, though, not very detailed in order to understand whether it is efficient for application in certain field or upon solution of a given problem. This work attempted to perform more detailed analysis of platform solutions, however only for API gateway.

In addition, it should be mentioned that in all mentioned publications, the considered API management systems are oriented at conventional approach to development of interfaces. However, recently new procedure of API representation has been introduced: GraphQL, which modifies estimations of previously analyzed platforms, since it is both the data manipulation logic with open source code for API, and the environment of requests to stored data [15]. Contrary to conventional interfaces with data fixed in predefined format, while using GraphQL it is possible to obtain only required data and not all data as in SQL for databases. Using this technology, a client is able not to request data from several API but to operate with data flowchart without consideration for certain flowchart fragments with regard to certain API.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Another important issue upon development of open API is computer security. Since API is a certain access point to company software system, then this entry should be secured [16]. Not only access to API should be secure, that is, authentication and authorization systems, but the whole mechanism of API functioning, that is, API gateway [17].

This work considered web interfaces operating according to HTTP protocol, however, a new protocol appeared recently, WebSocket, which is, contrary to HTTP, is asynchronous and symmetrical, which facilitates communication in real time, decreasing latency of network interaction and traffic amount [18]. Taking into account these advantages, it is obvious that the WebSocket protocol will be used in open interfaces, hence, while selecting API management system, it would be required to consider for support of this protocol. Though, some software products already support this protocol, for instance, considered here WSO_2 API Management [19] or Tyk API Gateway [20].

5. CONCLUSION

The most efficient API gateways were studied in this work. Three software products were considered: Gravitee, APIMan, and WSO2 API Management, which met two preset criteria: Java product implementation, open source code of the product.

The API gateways were compared using three-dimensional environment with the following coordinates: intensity of performed functions for API development, labor intensity of API implementation, performance of API gateway.

The intensity of API management functions performed by the systems was compared with regard to preset criteria on the basis of analysis of specifications of software tools and subsequent verification of the mentioned functions during operation with software. The comparison revealed that Gravitee was the best software product.

The labor intensity of API implementation was compared using each product for development of test interface comprised of three blocks: block of request transformation, block of error handling, block of HTTP request. In terms of this comparison, the best software products were Gravitee and WSO_2 API Management.

The performance of the software products was compared using the developed test interface, which, upon access to it, generated several HTTP requests, the respective response was obtained with five second delay, thus simulating complex scenario of API operation. Then the interface was requested several times. In terms of this comparison, the best software products were Gravitee and WSO_2 API Management.

Therefore, in terms of all coordinates the best software product was Gravitee.

ACKNOWLEDGMENTS

This work was supported by the Competitiveness Program of National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), contract with the Ministry of Education and Science of the Russian Federation No. 02.A03.21.0005, 27.08.2013.

REFERENCES:

[1] ProgrammableWeb. Research Shows Interest in Providing APIs Still High; 2018. Available from: https://www.programmableweb.com/news/research-shows-interest-providing-apis-still-high/research/2018/02/23.

[2] Collins G, Sisk D. API economy. Delloite Insights; 2015. Available from: https://www2.deloitte.com/insights/us/en/focus/tech-trends/2015/tech-trends-2015-what-is-api-economy.html.

[3] Heffner R. The Forrester Wave™: API Management Solutions, Q4 2018. Leveraging; 2018. Available from: https://b.content.wso2.com/sites/all/forrester-q4-2018/The-Forrester-Wave-API-Management-Solutions-Q4-2018.pdf.

[4] IT Central Station. Best API Management Tools: Comparison of API Gateway Solutions; 2019. Available from: https://www.itcentralstation.com/categories/api-management.

[5] Capterra. API Management Software. Available from: https://www.capterra.com/api-management-software/.

[6] G2 Crowd. Best API Management Software; 2019. Available from: https://www.g2crowd.com/categories/api-management.

[7] Predictive Analytics Today. Top 9 API Management Platforms; 2018. Available

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

from: https://www.predictiveanalyticstoday.com/top-api-management-platforms/.
[8] Wikipedia. API management; 2018. Available from: https://en.wikipedia.org/wiki/API_management.
[9] Gravitee.io. 2018. Available from: https://gravitee.io/.
[10] APIMan. Open Source API Management; 2017. Available from: http://www.apiman.io/latest/index.html.
[11] WSO2. WSO2 API Management; 2018. Available from: https://wso2.com/api-management/.
[12] APIMan GitBooks. APIMAN USER GUIDE; 2018. Available from: https://apiman.gitbooks.io/apiman-user-guide/.
[13] Gravitee.io. API Management; 2018. Available from: https://docs.gravitee.io/apim_publisherguide_manage_apis.html.
[14] WSO2. WSO2 API Manager Documentation; 2018. Available from: https://docs.wso2.com/display/AM260/.
[15] Wikipedia. GraphQL; 2018. Available from: https://en.wikipedia.org/wiki/GraphQL.
[16] Macy J. API security: Whose job is it anyway?. Network Security 2018; 9: 6-9.
[17] Macy J. How to build a secure API gateway. Network Security 2018; 6: 12-14.
[18] IETF Tools. The WebSocket Protocol; 2011. Available from: https://tools.ietf.org/html/rfc6455.
[19] WSO2. Create a WebSocket API; 2019. Available from: https://docs.wso2.com/display/AM210/Create+a+WebSocket+API.
[20] Tyk Open Source API Gateway. Websockets; 2018. Available from: https://www.tyk.io/docs/other-protocols/websockets/.