Comparative Study of Data Sending Methods for XML and JSON Models

Anca-Raluca Breje[1], Robert Győrödi[2], Cornelia Győrödi[3], Doina Zmaranda[4], George Pecherle[5]
Department of Computer Science and Information Technology
University of Oradea Oradea, Romania

Abstract—Data exchange between different devices and applications has become a necessity nowadays. Data is no longer stored locally on the device, but in the cloud. In order to communicate with the cloud and exchange data, web services are being used. To keep the communication consistent across different devices and platforms, the data needs to be formatted using a standard data format, such as JSON or XML. This paper compares both standards and provides an in depth analysis of their performance. In order to perform the analysis a web API was built in the PHP framework Laravel, which was then tested with the help of the API development environment called Postman for different number of transferred items.

Keywords–XML; JSON; data model; data transfer; application programming interface

I. Introduction

Mobile applications have taken a large scale lately and they often require to be connected to a database hosted on a server. Common approaches imply that access to these data to be accomplished via web APIs [1], and consequently, these web APIs must have a very good response time to ensure that there are no delays in displaying the required data to the end-user.

In order to have the data that reaches the applications usable, it needs to be formatted according to a standard that can be parsed, read and used by both the API and the application that will access it. Two of the most popular formatting standards for applications that use web APIs are JSON and XML. Both JSON and XML formats have strengths and drawbacks that qualify them for specific purposes and each of them can be used according to the need of the system [2]. It is well known nowadays that JSON power stays into its simple structure that makes it suitable for simple data transmission [3]. On the other hand, one of the main advantages of XML is represented by its flexibility, given by the possibility of storing (theoretically) all possible data types, unlike JSON where storing is limited to common data types [4]. This flexibility comes with a cost, XML format being much more difficult to parse and to convert to objects, due to its strict structure definition (tree-like), than the much more simplistic JSON format [5].

Another well-known advantage of XML format is represented by the large availability of technologies that could be used for validating XML documents, such as XML namespaces or XMLSchema [6]. Even if for JSON format, in the last years, similar technologies emerged, such as JSON Schema, their range of functions are still not comparable with XML technologies ones [7].

Starting from these two well-known technologies, a performance comparison between JSON and XML data formats, from both runtime and memory usage point of view, is presented in this paper. The paper starts by reviewing the related work, as presented in Section II. A specific testing architecture was specifically developed for running the tests: the algorithm and data structure that are involved are described in Section III. Section IV presents the API developed for accessing the database while Section V illustrates issues related to validation. In Section VI several tests were run, and the obtained results were analysed from several perspectives. Furthermore, Section VII resumes the conclusions of the study.

II. Related Work

Several comparisons based on different scenarios were done between the two formats in [8]. Also, a comprehensive analysis of XML and JSON for web technology is described in [2]. In [9], the process data exchange between a mobile application and remote servers using JSON format is described. A performance comparison between the two interchange formats for simple structures is described in [10]. As outlined in [11], switching between a format to another is possible by using converters, meanwhile preserving data content. Common conclusion that results is that generally, JSON with its simple format behaves faster and uses fewer resources than XML. However, with is much more complex structure and validation techniques, XML remains actual for applications that are manipulating various types of data.

In this idea, this paper presents a benchmark performance comparison between JSON and XML data formats, from both runtime and memory usage point of view, when different types of data were involved. Thus, the performance tests carried on in this paper explore the execution time and memory footprint when using XML and JSON for data sending methods for various applications. Besides other comparative studies existing in the literature, we tried to consider into comparison, for both formats, two additional issues: data validation and data compression. The main goal is to assess the performance impact on the two methods when including data validation and also when compression on the server is enabled.

III. Test Performance Algorithm and Data

No matter what format is chosen, conversion of data needs using some specific server-side language, such as PHP. Consequently, to be able to analyse the two data sending formats, a web API using the PHP framework Laravel was built [12]. The developed API supports both formats for the four basic operation that are related to data storage, namely data request, sending and inserting data in the database, sending and updating data in the database, deleting data.

Each of the four operations was tested through the developed web API, which for this paper was called using an API development environment called Postman.

These operations were then analysed based on two criteria: data receiving speed and the size of the data received.

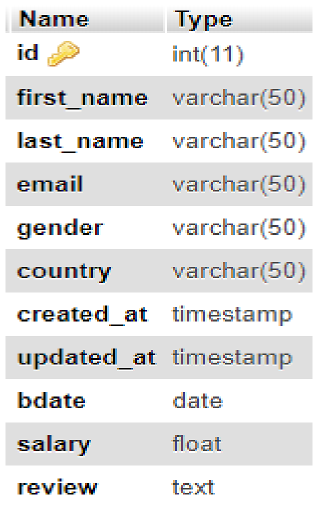
To test the data parsing performance of the two formats, JSON and XML, we used the following generic algorithm

```
var XML / var JSON;
var TIME BEFORE = get current timestamp
validate XML/JSON;
decode XML/JSON;
var TIME AFTER = get current timestamp
display TIME AFTTER - TIME BEFORE
```

This performance test uses the PHP language to execute the parsing of an array with three elements that is encoded in XML and JSON format. The functions used for the test were *json_decode* for the JSON data and *simplexml_load_string* for the XML data.

Fig. 1 illustrates the structure of the database table that is used in the application, containing sample data of some people, and it has the following columns: id, first name, last name, email, gender, country. In order to illustrate the impact of using different types of data for both XML and JSON formats, columns of several other types were included into the table: timestamp, date, float and text.

The table data is handled using SQL queries written in PHP using the Eloquent ORM, which is part of the Laravel framework [12].



Name	Type
id	int(11)
first_name	varchar(50)
last_name	varchar(50)
email	varchar(50)
gender	varchar(50)
country	varchar(50)
created_at	timestamp
updated_at	timestamp
bdate	date
salary	float
review	text

Fig. 1. Users Table with Different Column Types.

IV. API URLs and HTTP Request Methods

The API built for the testing of the two models, is divided in two big parts, the part that expects data and send back data formatted as JSON, and the other part that expects data and sends it back formatted as XML, the URL bases for the two parts are the following [1]:

- http://localhost/api/json
- http://localhost/api/xml

When calling one of the API URLs, the method with which the API is called must be also specified, the available methods are the following [13]

- GET – for requesting data
- POST – for adding new data
- PUT – for updating existing data
- DELETE – for deleting data

Fig. 2 explains the interaction between the request of the API and the data stored in the database.

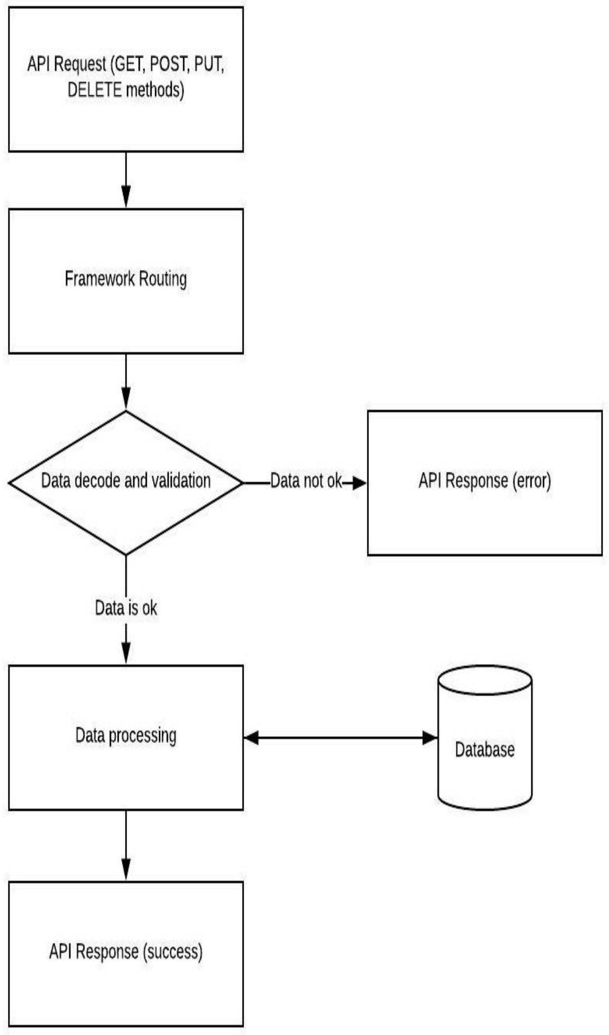


Fig. 2. API Request Diagram.

V. Data Validation

To make sure the data received by the API is correct and it contains all the needed elements, validation schemas were used.

For the JSON format data, we used JSON Schema which is a vocabulary to annotate and validate JSON documents [14]. A library that implements JSON Schema that is compatible with Laravel is *JSON Schema for PHP* [15]. An example of how the name field is defined in the JSON schema, to be validated in the input, is the following

```
[
   "type"=>"array",
       "properties"=>(object)[
          "first_name"=>(object)[
             "type"=>"string"
          ],
          "last_name"=>(object)[
             "type"=>"string"
          ],
          …
        ]
     ]
```

For the XML format data, we used a XSD schema which defines the elements of the correct XML document. A library that implements XSD Schema and is compatible with Laravel is *PHP Xml validator* [16]. An example of how the name field is defined in the XML schema, to be validated in the input, is the following

```
<xs:element     name="record"     minOccurs="0"
maxOccurs="10000">
  <xs:complexType>
    <xs:sequence>
      <xs:element    ref="id"     minOccurs="0"
maxOccurs="1"/>
      <xs:element                ref="first_name"
minOccurs="0" maxOccurs="1"/>
      <xs:element ref="last_name" minOccurs="0"
maxOccurs="1"/>
…
    </xs:sequence>
  </xs:complexType>
</xs:element>
…
<xs:element                     name="first_name"
type="xs:string"/>
<xs:element name="last_name" type="xs:string"/>
```

VI. Comparison between Transmission of Data for XXL and JSON Models

A. Data Request (GET)

For the GET method, the data is obtained by calling one of the following URLs (the example is using the *cURL* function), where the limit parameter represents how many elements we want to get from the database with the API.

To request data in JSON format

```
curl -X GET
http://localhost/api/json/people?limit=1000
```

To request data in XML format:

```
curl -X GET
http://localhost/api/xml/people?limit=1000
```

In order to obtain the data, the called API web function will execute a database query, which returns a number of rows less than or equal to the value in the limit parameter.

The data returned by the SQL query is then converted using *simple_load_string()* for XML and *json_decode()* PHP functions to the required format and returned as a response to the API call.

Formatting data in the XML format can consume more memory than in JSON format. Moreover, adding attributes to the XML tags contributes to the final data size of the XML result text, to lowering the performance of the applications that use it.

In contrast to the XML format, JSON is more simplistic and easier to use in applications, with lower impact on application performance, especially because JSON is a format based on JavaScript objects, which most programming languages (PHP, JS, C #, etc.), can use without the need of including any external libraries [17].

As can be seen in Fig. 3 and Table 1, the XML format for the same data and for the same number of items takes up more storage space, but not more than 20% more than the JSON format. This also affects the data response time, the impact on this being noticeably higher, being more than three times higher (in the case of 10,000 items).

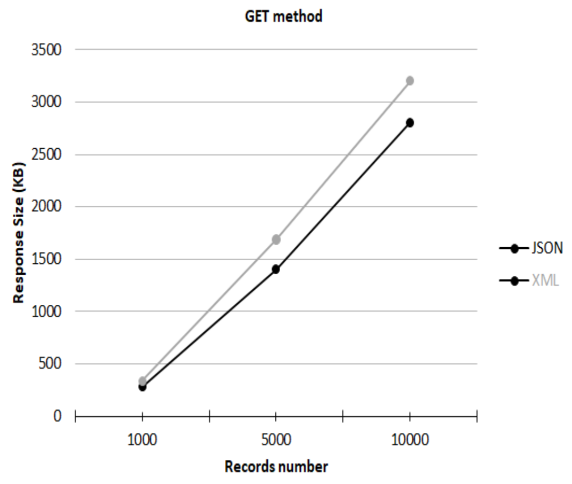


Fig. 3. XML and JSON Graph for Response Size in KB – GET Method without GZIP Compression.

TABLE I. XML and JSON Response Size in KB – GET Method – without GZIP Compression

Records no.	**JSON**	**XML**	**% XML over JSON**
1000	281 KB	337 KB	119
5000	1402 KB	1689 KB	120
10000	2805 KB	3205KB	114

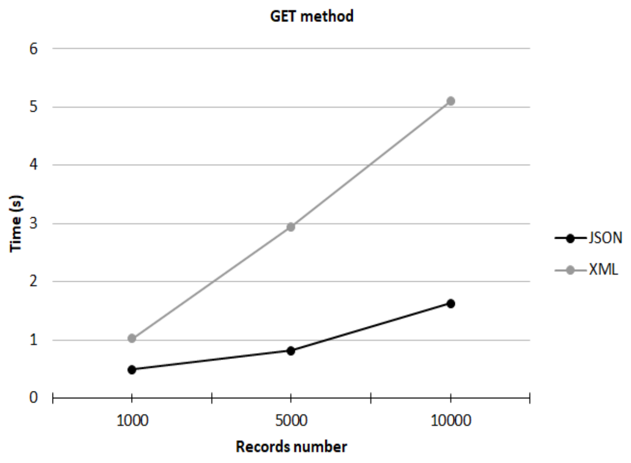


Fig. 4. XML and JSON Graph for Response Time in seconds GET Method without GZIP Compression.

TABLE II. XML AND JSON RESPONSE TIME IN SECONDS GET METHOD WITHOUT GZIP COMPRESSION

Records no.	**JSON**	**XML**	**% XML over JSON**
1000	0.49 s	1.03 s	210
5000	0.82 s	2.95 s	359
10000	1.63 s	5.1 s	312

As presented in Fig. 4 and Table 2, for JSON format, the call response time for 1000 items is 0.49 s, the response size is 281 KB, and for XML format, for the same number of items, the response time is 1.03 s, the response size is 337 KB.

For the GET method, we also tried requesting the data from a server that has the gzip compression activated to see if any if there are benefits in having a server-side compression.

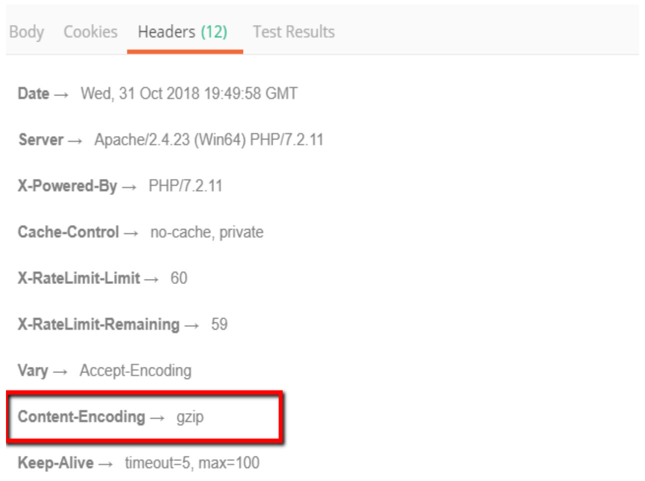
Before using the gzip compression, the deflate module needs to be enabled (on an Apache server) and also the data that should be compressed needs to be specified on the *.htaccess* file of the project or on the virtual host configuration.

In order to have the compression for the JSON and XML format we are using, the following lines were added in the *.htaccess* file of the project:

```
<IfModule mod_deflate.c>
    AddOutputFilterByType                DEFLATE
            application/json application/xml
</IfModule>
```

To see if the gzip compression was used on the data from the response, the header of the response can be checked (Fig. 5), if the Content-Encoding is set to gzip, it means the gzip compression was correctly enabled for the data type sent from the server in the response.

As it is shown in Table 3 and Table 4, the size of the response changed after the gzip was applied, and the response time is better if gzip is enabled, namely for JSON the response size is approximately 5 times smaller and the response time is 25% to 35% faster.



```
Date → Wed, 31 Oct 2018 19:49:58 GMT
Server → Apache/2.4.23 (Win64) PHP/7.2.11
X-Powered-By → PHP/7.2.11
Cache-Control → no-cache, private
X-RateLimit-Limit → 60
X-RateLimit-Remaining → 59
Vary → Accept-Encoding
Content-Encoding → gzip
Keep-Alive → timeout=5, max=100
```

Fig. 5. Response Headers after the GZIP Compression is Enabled.

TABLE III. XML AND JSON RESPONSE SIZE IN KB – GET METHOD – WITH GZIP COMPRESSION

Records no.	**JSON**	**XML**	**% XML over JSON**
1000	52.9 KB	73.4 KB	139
5000	263.1 KB	356.8 KB	135
10000	529.3 KB	731.3KB	138

TABLE IV. XML AND JSON RESPONSE TIME IN SECONDS–GET METHOD WITH GZIP ENABLED

Records no.	**JSON**	**XML**	**% XML over JSON**
1000	0.32 s	0.6 s	187
5000	0.61 s	2.21 s	362
10000	1.22 s	4.14 s	339

For XML, the results are similar, the response size is around 4.6 times smaller, the response time is 20% to 40% better. Based on the results, if the server allows the gzip compression, it would be recommended to use it for both JSON and XML data request.

B. Send and Insert Data (POST)

For the POST method, sending and inserting data is done by calling the following URL (*cURL* function), while putting in the body of the call the data that is being sent and is going to be inserted.

To send and insert data in JSON format:

```
curl -X POST http://localhost/api/json/people
```

To send and insert data in XML format:

```
curl -X POST http://localhost/api/xml/people
```

Data sent to the web API is processed using PHP functions and inserted into the database through SQL INSERT queries, and as a result, a success message is returned if everything went well. Because the web API must know that it will receive data in JSON or XML format, we will specify this fact in the header of the call by setting the content type property as it follows:

- Content-type: application/json-for JSON
- Content-type: application/xml-for XML

The data that is sent to the web API, formatted as JSON needs to have the following shape:

```
[   {
            "first_name":"Lucille",
            "last_name":"Baddoe",
           "email":"lbaddoe0@earthlink.net",
            "gender":"Female",
            "country":"China",
            "bdate":"1989-05-12",
            "salary": 7433.55,
            "review":"Lorem ipsum...",
    },
    ... ]
```

As for the XML, data the need to be formatted like the example below:

```
<?xml version="1.0”?>
<root>
  <Collection>
    <row_0>
            <first_name>Elise</first_name>
            <last_name>McGurn</last_name>
            <email>emcgurn0@a8.net</email>
            <gender>Female</gender>
            <country>Philippines</country>
            <bdate>1989-05-12</bdate>
            <salary>7433.55</salary>
            <review>Lorem ipsum...</review>
    </row_0>
            …
    </Collection>
</root>
```

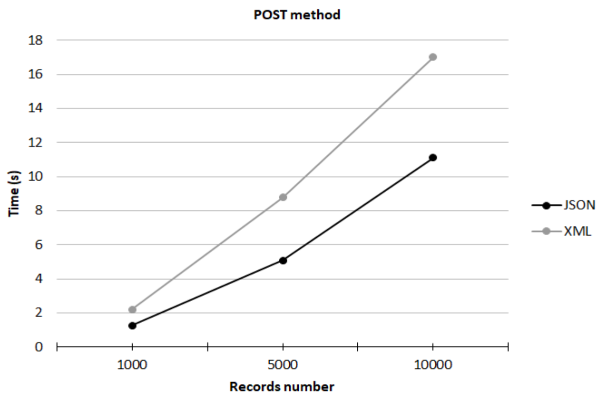


Fig. 6. XML and JSON Graph for Response Time in Seconds – POST Method.

TABLE V. XML AND JSON RESPONSE TIME IN SECONDS–POST METHOD

Records no.	JSON	XML	% XML over JSON
1000	1.25 s	2.21 s	176
5000	5.09 s	8.78 s	172
10000	11.11 s	17.02 s	153

Data sent in the XML format is bigger as size that the data send as JSON this will also affect the call response time, as the data needs more time to get to the server. This can be seen in Fig. 6 and Table 5 where for the same number of elements, the time is bigger for the call that send data as XML to the server.

The response time was bigger each try for the data sent as XML, for 10,000 records sent to be inserted the time for JSON was 11.11 seconds and for XML it was 17.02 seconds, this means that the call using JSON formatted data is 1.53 times faster for 10,000 records. Even if using XML format implies more than 50% more response time, it should be noticed that this percent does not increase much if number of records increases, remaining more or less at similar levels or even decreasing. Consequently, the (negative) impact of using XML format is decreasing as the number of implied records increase.

C. *Send and Update Data (PUT)*

For the PUT method, sending and inserting data is done by calling the following URL (*cURL* function) and putting into the body of the call the data that is being sent and is going to be inserted.

To send and update data in JSON format

```
curl -X PUT http://localhost/api/json/people
```

To send and update data in XML format

```
curl -X PUT http://localhost/api/xml/people
```

The data that is sent to the server is updated if is found in the database by the ID that is specified for each record, as a response message we get a success one if all the records were updated successfully or an error one if the operation could not be executed.

The data that is sent to the web API, formatted as JSON needs to have the following shape:

```
[
    {       "id":1,
            "first_name":"Lucille",
            "last_name":"Baddoe",
           "email":"lbaddoe0@earthlink.net",
            "gender":"Female",
            "country":"China",
            "bdate":"1989-05-12",
            "salary": 7433.55,
            "review":"Lorem ipsum...",
    },
    …
]
```

As for the XML, data the need to be formatted like the example below:

```
<?xml version="1.0"?>
<root>
  <Collection>
    <row_0>
            <id>1</id>
            <first_name>Elise</first_name>
            <last_name>McGurn</last_name>
            <email>emcgurn0@a8.net</email>
            <gender>Female</gender>
            <country>Philippines</country>
            <bdate>1989-05-12</bdate>
            <salary>7433.55</salary>
            <review>Lorem ipsum...</review>
    </row_0>
            …
    </Collection>
</root>
```

As can be seen from Fig. 7 and Table 6, for the PUT method, the response time for sending data and updating it, for both the XML format and the JSON format, has a similar value for a small number of elements, the difference between them appears only for a larger number of items, where the JSON format has a better time.

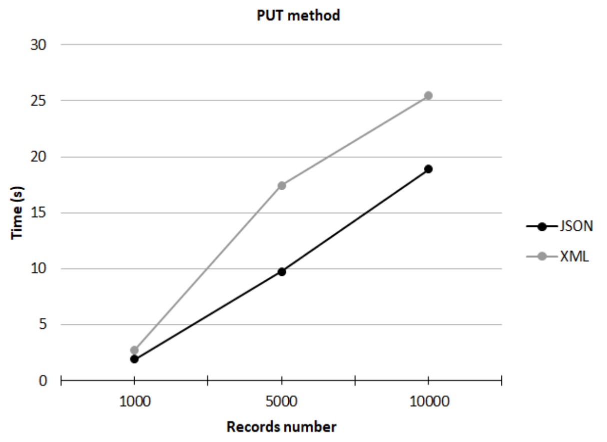


Fig. 7. XML and JSON Graph for Response Time in Seconds–PUT Method.

TABLE VI. XML AND JSON RESPONSE TIME IN SECOND –PUT METHOD

Records no.	**JSON**	**XML**	**% XML over JSON**
1000	1.9 s	2.73 s	143
5000	9.73 s	17.43 s	179
10000	18.9 s	25.44 s	134

D. Data Deletion (DELETE)

The deletion of the elements in the database is done by calling one of the following API URLs (*cURL* function), where the limit parameter represents how many elements will be deleted from the database through the API.

To delete data in JSON format

```
curl -X DELETE
http://localhost/api/json/people?limit=1000
```

To delete data in XML format

```
curl -X DELETE
http://localhost/api/xml/people?limit=1000
```

To delete the records, the called API function will execute a database query that deletes a number of rows less than or equal to the value sent in the limit parameter.

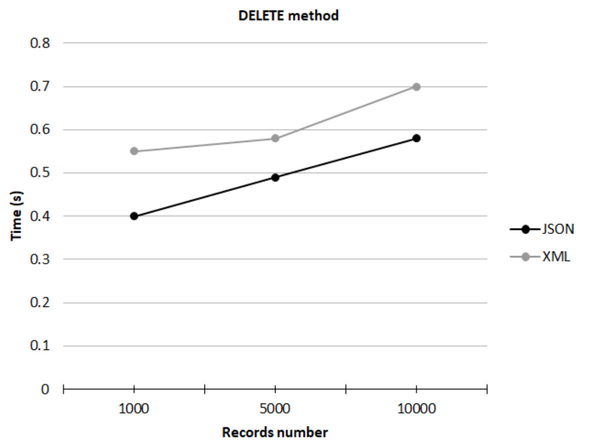


Fig. 8. XML and JSON Graph for Response Time in Seconds–Delete Method.

TABLE VII. XML AND JSON RESPONSE TIME IN SECOND–DELETE METHOD

Records no.	**JSON**	**XML**	**% XML over JSON**
1000	0.4 s	0.55 s	137
5000	0.49 s	0.58 s	118
10000	0.58 s	0.7 s	120

As can be seen in Fig. 8 and Table 7, for the DELETE method, the response time, both the XML format and the JSON format, has a similar value for all tested values for example when deleting 1000 or 5000 elements. For the deletion of 10000 elements, the response time is 0.58 s for JSON and 0.7 s for XML. However, the impact of using XML implies only around 20% more response time than for JSON.

Time response values are similar because the functions used to encode and to decode data in JSON or XML format are less used for this method, here the execution time of the SQL DELETE query is having a greater importance in the response time of the API.

VII. CONCLUSIONS

In conclusion, the case study from this paper presents a comparison of the data transfer methods for XML and JSON models.

The comparison of the two formats was achieved by building a web API in the PHP framework called Laravel. This API supports both formats for four operations that are related to data transfer: data request (GET method), send and insert data (POST method), send and update data (PUT method), deleting data (DELETE method).

Each of the four operations was tested through the API, which was called for this paper using an API development environment called Postman.

These operations were then analysed by two criteria, the response speed, in seconds, and the size of the data received, in KB. The data is also validated to prevent wrong data to be sent to the server. The tests are done with and without gzip server compression, because not all servers have this option enabled. The obtained results for each test were analysed and discussed in detail; as an overall evaluation, we noticed that for all operations, except the deletion one, the JSON format is more effective both in terms of data size and time of web API response time, which was in some cases 30 to 40% faster for JSON than the XML format, being generally lower as number of records implied is increasing. However, for some applications that require sending heterogenous complex structures, for which XML format offer better support, the above percentage impact over performance must be assumed. Consequently, when speaking about the necessity of XML utilization, an issue that could be investigated in the future implies the use of a converter for switching from XML to JSON. The possibility of using such converters and their impact on performance issues represent a future development that will be further investigated.

References

[1] B. Matthias, “Restful Api Design”, CreateSpace Independent Publishing Platform, ISBN-10: 1514735164, ISBN-13: 978-1514735169, 2016

[2] Z.U. Haq, G.F. Khan and T. Hussain, “A Comprehensive analysis of XML and JSON web technologies”, New Developments in Circuits, Systems, Signal Processing, Communications and Computers, pp. 102-109, 2013

[3] H. S. Padda1 and G. K. Gupta, “Analysing Impact of Delimiters on the Size of JSON Data Interchange Format”, International Research Journal of Engineering and Technology, Vol. 2, No. 8, -ISSN: 2395-0056, www.irjet.net, 2015

[4] A. Simec and M. Maglicic, “Comparison of JSONamd XML Data Formats”, Central European Conference on Information and Intelligent Systems; Varazdin, Croatia, pp. 272-275, 2014

[5] D. Peng, L.Cao, and W Xu, “Using JSON for Data Exchanging in Web Service Applications”, Journal of Computational Information Systems 7: 16, ISSN 5883-5890, 2011

[6] P. Bourhis, J. L. Reutter, F. Suárez and D.Vrgoč , “JSON: Data model, Query languages and Schema specification”, Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, pp. 123-135, 2017

[7] M. Šabanović, M. Saračević and E. Azizović, “Comparative analysis of AMF, JSON and XML technologies for data transfer between the server and the client”, Periodicals Of Engineering and Natural Sciences, Vol. 4, No.2, 2016

[8] S. Zunke and V. D’Souza, “JSON vs XML: A Comparative Performance Anaysis of Data Exchange Formats”, IJCSN International Journal of Computer Science and Network, Volume 3, Issue 4, ISSN 2277-5420, www.IJCSN.org, 2014

[9] K. Ishwarjit, K. Sharanpreet and K. Gurinder, “Accessing Remote Database in IOS Application using JSON Parsing with Objective-C”, International Journal of Advanced Technology in Engineering and Science, Vol. no. 5, No. 1, www.ijates.com, 2017

[10] N. Nurseitov, M. Paulson, R. Reynolds and C. Izurieta, “Comparison of JSON and XML Data Interchange Formats”: A Case Study, ISCA 22nd International Conference on Computers and Their Applications in Industry and Engineering, pp. 157-162, 2009

[11] B. Šandrih, D. Tošić and V. Filipović, “Towards Efficient and Unified XML/JSON Conversion - A New Conversion”, IPSI BgD Transactions on Internet Research (TIR) vol. 13, no. 1, ISSN 1820-4503, 2017

[12] W. Natham, “Learning Laravel 5 - Building Practical Applications”, 5th edition, 2017

[13] G. David, T. Brian, S. Marjorie, A. Anshu and R. Sailu, “HTTP: The Definitive Guide”, O'Reilly Media, ISBN-13: 978-1565925090, ISBN-10: 9781565925090, 2002

[14] JSON Schema - http://json-schema.org/

[15] https://github.com/justinrainbow/json-schema

[16] https://packagist.org/packages/seromenho/xml-validator

[17] M. Tom, “JSON at Work”, O'Reilly Media, ISBN-13: 978-1449358327, ISBN-10: 1449358322, 2017