Received: 3 January 2021 | Revised: 20 February 2021 | Accepted: 29 February 2021
DOI: 10.37917/ijeee.17.1.4 Vol. 17| Issue 1| June 2021

Open Access

Iraqi Journal for Electrical and Electronic Engineering
Original Article

# Encoding JSON by using Base64

**Mohammed Thakir Shaamood**
College of Arts, Al-Iraqia University, Iraq

**Correspondence**
***Mohammed Thakir Shaamood**
College of Arts, Al-Iraqia University, Iraq
Email: m4alani1@gmail.com

**Abstract**
***Transmitting binary data across a network should generally avoid transmitting raw binary data over the medium for several reasons, one would be that the medium may be a textual one and may not accept or correctly handle raw bitstream, another would be that some protocols may misinterpret the meaning of the bits and causes a problem or even loss of the data. To make the data more readable and would avoid misinterpretation by different systems and environments, this paper introduces encoding two of the most broadly used data interchange formats, XML and JSON, into the Base64 which is an encoding scheme that converts binary data to an ASCII string format by using a radix-64 representation. This process, will, make the data more readable and would avoid misinterpretation by different systems and environments.***
***The results reflect that encoding data in Base64 before the transmission will present many advantages including readability and integrity, it will also enable us to transmit binary data over textual mediums, 7 Bit protocols such as SMTP, and different network hardware without risking misinterpretation.***

**KEYWORDS: JSON, Base64, Encoding, Binary, XML.**

## I. Introduction

The technical foundation of the Web is constantly evolving [1]. Exchanging multiple information formats mainly depends on web services so as to exchange more thoughts like a search engine, information by the web services, mobile information, enterprise application, XML (Extensible Markup Language) and JSON (JavaScript Object Notation). Data interchange formats are considered an influencing factor in data serialization, in terms of the performance and rate of data transfer. JSON structure is similar to the type model of the essential java scripting programming language which gives it many features like, flexibility, independent text format and it is easy to use [2].
Technology is evolving rapidly. From AI to geo-targeting and automation, along with other advancements in information technology has set the stage for more technological evolution. Robotics are becoming more sophisticated, and even our daily appliances can now be connected to the internet. All of These advancements have brought many inventions and creations, created new and easier ways of communications and overall, made life much easier, these new and emerging technologies have made us highly dependent on information technology.
Along with these new advancements came new problems, and the obstacle we are discussing in this paper is one of efficient transmission of information and data, which is the backbone of every information system, the goal of every system is to move the highest possible volume of accurate information through the multiple layers of the system in the least possible time while consuming as little processing overhead as possible, while maintaining the security and integrity of the data. This can be achieved, in part, by using an efficient format by which the data can be interchanged.

## II. Numerical Systems

Numerical data is generated by different computing sources [3]. Representing and numbering of digits in the computer mainly done in computer ideology by a number system and that's digit called "inner " in the computer system. The computer used the binary system to represent all kinds of data and information. That means representing every (value/number) that the user saves or fetching from/ feeding into the computer memory. Computer architecture supports the following number of systems [3].

- Binary number system (Base 2)
- Octal number system (Base 8)
- Decimal number system (Base 10)
- Hexadecimal number system (Base 16)

Binary System

The Binary System is represented by a set of 1s and 0s that represents ON and OFF state respectively, in other words, it represents the presence of an electrical signal and lack thereof. Binary is a base 2 system, which means that the base of the system is only 1 and 0, the first digit is worth 1 decimal value, and the second is worth 2, the third is worth 4 and so on, multiplying by a factor of 2 as you go. The main motive behind using binary in computer science is Hardware limitations and physics, during the early days of computers; it was very hard to control and measure electrical signal so precise to represent individual decimal values, so it was much more logical and feasible to distinguish them within those two states, ON and OFF. For easier use, every 8 bits in binary system is grouped into one unit called a Byte, which can represent decimal values from 1 to 256, you can imagine, that in large amounts of data, Binary bits will be a lot and harder to manage and understand, which is one of the disadvantages of binary system. It is for that reason, the need arises again for a system that allows grouping of binary numbers which makes it easier to read, write and understand in a more human-friendly way, as humans are used to grouping together numbers and things for easier understanding. Also, to help in writing in less digits and lowering the possibility of error occurring [4].

## III. Data Interchange

Data interchange between front end and back end is a big factor in building web applications; the format by which the data is exchanged is an important determining factor in both efficiency and functionality of a web application, Common data exchange formats between Web Front End and Web Back End in the following subjects [5].

### A. *Extensible Markup Language (XML)*

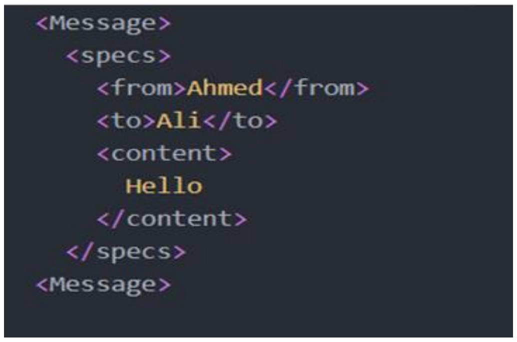
XML stands for Extensible Markup Language [6]. XML is uses encoding format understandable by both humans and machine for data intended to be published on the web [7]. The main goal of designing XML was to simplify and standardize data exchange.

XML is explicitly developed to provide World Wide Web information, much like HTML, the basic language used for the development of web pages since the start of the web. Since we already have HTML, which is changing to fulfill more specifications.

There are two major XML applications: the first is to display low-level details, example configuration files. The second is a way to apply metadata to records, for example by playing italics or bold in a text, you may want to highlight a specific word.

XML is intended for the first time to replace the conventional way of viewing data, including the name and value pairs chart.

The second XML application is close to the working of HTML. The document text is in an overall jar, the <body> part, with single phrases surrounded by tags I or <b>. In both cases, several strategies were developed over the years [8]. Since increased Internet usage and widespread use of distributed applications especially with components built and controlled by separate parties, the problem with these various approaches has become more evident than ever. This is an intercommunication challenge.

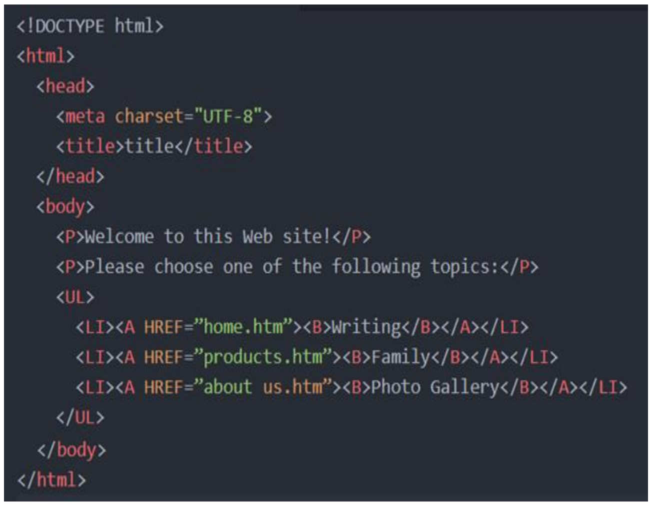


```
<Message>
  <specs>
    <from>Ahmed</from>
    <to>Ali</to>
    <content>
      Hello
    </content>
  </specs>
<Message>
```

Fig. 1: XML Body

### B. *Why XML*

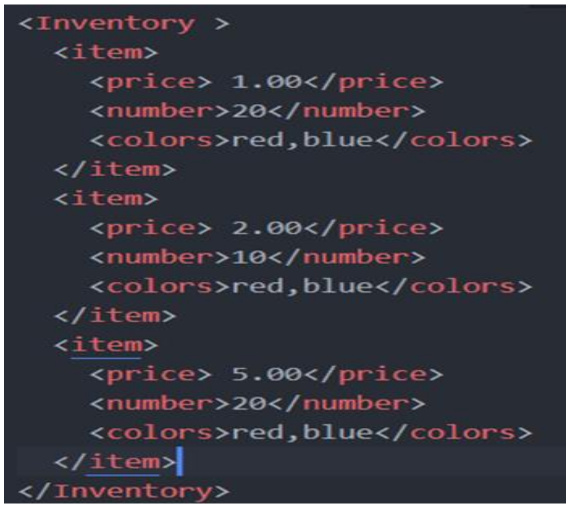
Extensible Markup Language (XML) has become a data structure that widely used in web services [8]. HTML uses a fixed number of elements to describe a default Web page component. Headers, charts, tables, photographs and hyperlinks [9] are examples of these components. For eg, HTML functions well to create a homepage, as in the example below. (see figure 2):



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>title</title>
  </head>
  <body>
    <P>Welcome to this Web site!</P>
    <P>Please choose one of the following topics:</P>
    <UL>
      <LI><A HREF=”home.htm”><B>Writing</B></A></LI>
      <LI><A HREF=”products.htm”><B>Family</B></A></LI>
      <LI><A HREF=”about us.htm”><B>Photo Gallery</B></A></LI>
    </UL>
  </body>
</html>
```

Fig. 2: Example of HTML Page

Each element starts with a start-tag: a block of text preceded by a bracket of the left angle (<), followed by a right-angle bracket (>) which contains the element name and probably other details. Most elements finish with an end-tag, which is like the respective start-tag, except that it contains just a slash (/) character with an element name. The output of the feature is the text between the beginning and end tag, if any. Note that several of the elements in the example above have nested elements (that is, elements within other elements) [9]. When you create an XML document, you are not restrained by a set of predefined data tags, instead, you create your own tags and associate them with names, for that reason, you can use xml to describe anything starting from a database or and inventory to objects. In the example below (figure 3), we are using xml to describe a store inventory



```
<Inventory >
  <item>
    <price> 1.00</price>
    <number>20</number>
    <colors>red,blue</colors>
  </item>
  <item>
    <price> 2.00</price>
    <number>10</number>
    <colors>red,blue</colors>
  </item>
  <item>
    <price> 5.00</price>
    <number>20</number>
    <colors>red,blue</colors>
  </item>
</Inventory>
```

Fig. 3: Xml to Describe a Store Inventory

The name of the element in an XML document (in this case INVENTORY, Object, PRICE) does not form part of an XML description. It is important to understand the element names. You should instead render names for the details.

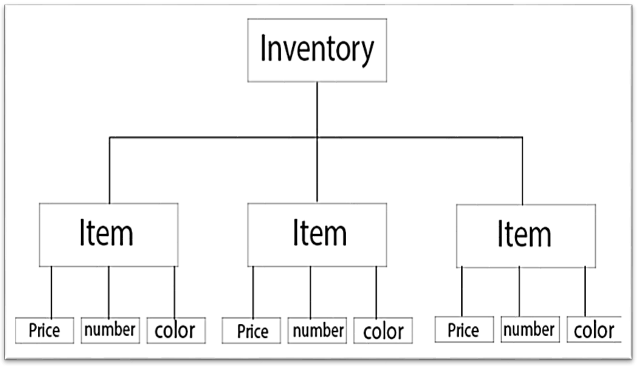


Fig. 4: XML Inventory Example

As you can see in the example above, an XML database is organized in a tree hierarchy, which includes elements that are entirely nestled in other elements and has a single top-level element (the document element or root node, as in this example). You can conveniently use XML in order to describe a hierarchically ordered database, such as products with costs, numbers and different levels of segment, as described above the structure of the example XML document is this.

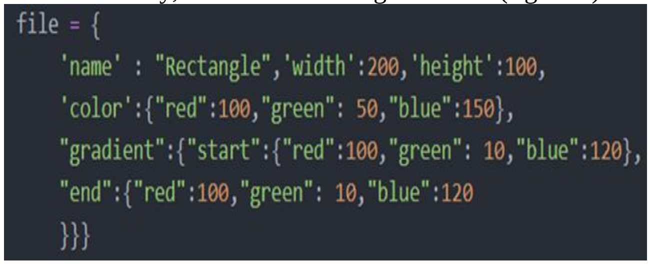
HTML is the primary language used to tell browsers how to display information on the Web. XML is used in conjunction with HTML and expands the websites' capacity to include almost any sort of information considerably, it's also used to sort, filter, arrange, manipulate and Present highly structured information.

### *C. Generating and writing XML*

An XML document is modeled as a tree that contains labeled nodes and a designated root [10]. In the below example, we will use a python code to generate an XML file that describes multiple objects with different attributes.

1) We will start with a rectangular object, a circular object, time object, a human object and a file. the first object (the rectangle) has the following attributes: (Type, Name, Width, Height, Gradient, and Color), the color has three colors (Red, Green, and Blue). The Gradient has sub features (Start of gradient with three colors Red, Green, and Blue). End of gradient is the second Gradient sub feature also with three colors (Red, Green, and Blue).

2) First defining those attributes using a standard nested dictionary, as shown in the figure below (figure 5):

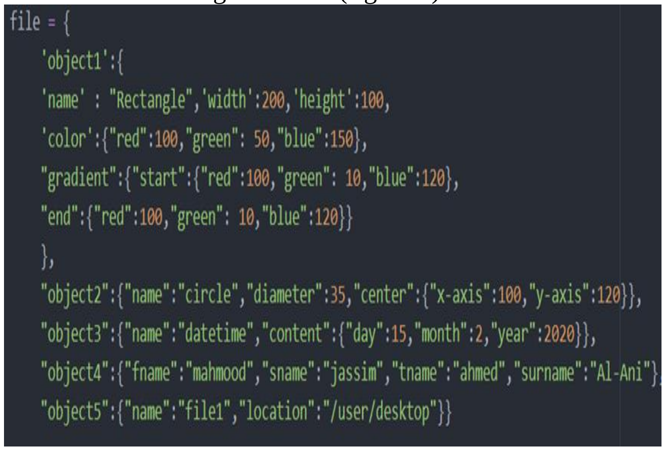


```
file = {
    'name' : "Rectangle",'width':200,'height':100,
    'color':{"red":100,"green": 50,"blue":150},
    "gradient":{"start":{"red":100,"green": 10,"blue":120},
    "end":{"red":100,"green": 10,"blue":120
    }}}
```

Fig. 5: Defining the Attributes

Using nested dictionaries is very helpful when generating an xml file, as it helps us define parent/child relations and multiple sub attributes within a single general attribute.

3) The same method applies for the other objects; we used name/value pairs to define each attribute/sub attribute, as shown in figure below (figure 6):

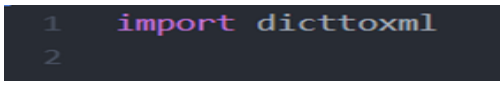


```
file = {
    'object1':{
    'name' : "Rectangle",'width':200,'height':100,
    'color':{"red":100,"green": 50,"blue":150},
    "gradient":{"start":{"red":100,"green": 10,"blue":120},
    "end":{"red":100,"green": 10,"blue":120}}
    },
    "object2":{"name":"circle","diameter":35,"center":{"x-axis":100,"y-axis":120}},
    "object3":{"name":"datetime","content":{"day":15,"month":2,"year":2020}},
    "object4":{"fname":"mahmood","sname":"jassim","tname":"ahmed","surname":"Al-Ani"}
    "object5":{"name":"file1","location":"/user/desktop"}}
```

Fig. 6: Using name/value

4) After defining each object, and assigning its attributes, we will need to convert these name/value pairs, for that we will use a module called "dicttoxml".

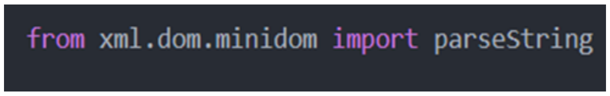
5) Then, import the module from python's library to our work place, using the "import "command, doing so will enable us to use classes and methods within the module.



```
import dicttoxml
```

Fig. 7: Importing the Module.

6) Then import one of the subclasses of the module, called "parseString", this class mainly enables the program to parse and manipulate string values within the dictionary.

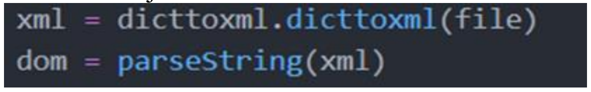


```
from xml.dom.minidom import parseString
```

Fig. 8: Using parseString

7) Next, we create an instance of the class "dicttoxml" within the dicttoxml module file with the dictionary we created "file" as class attribute, we also instantiated an

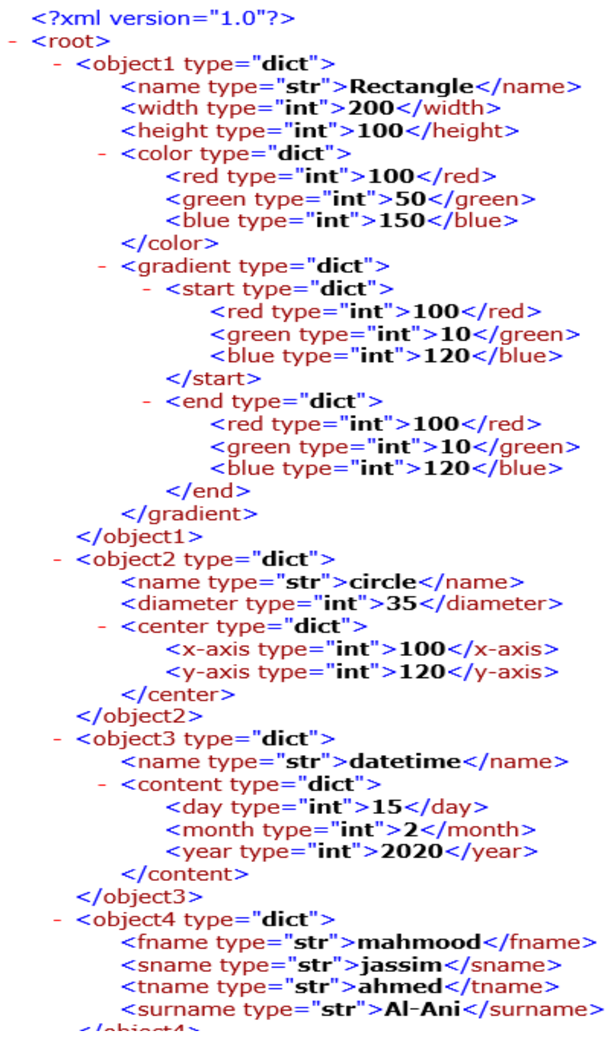
object of class parseString which we imported earlier with the object/instance “xml” as its attribute.



```
xml = dicttoxml.dicttoxml(file)
dom = parseString(xml)
```

Fig. 9: Using class dicttoxml

Finally, to show the results, we print out the method “toprettyxml” of the object “dom” to convert and display the results, the resulting xml file will be as shown below in:



```
  <?xml version="1.0"?>
- <root>
  - <object1 type="dict">
      <name type="str">Rectangle</name>
      <width type="int">200</width>
      <height type="int">100</height>
    - <color type="dict">
        <red type="int">100</red>
        <green type="int">50</green>
        <blue type="int">150</blue>
      </color>
    - <gradient type="dict">
      - <start type="dict">
          <red type="int">100</red>
          <green type="int">10</green>
          <blue type="int">120</blue>
        </start>
      - <end type="dict">
          <red type="int">100</red>
          <green type="int">10</green>
          <blue type="int">120</blue>
        </end>
      </gradient>
    </object1>
  - <object2 type="dict">
      <name type="str">circle</name>
      <diameter type="int">35</diameter>
    - <center type="dict">
        <x-axis type="int">100</x-axis>
        <y-axis type="int">120</y-axis>
      </center>
    </object2>
  - <object3 type="dict">
      <name type="str">datetime</name>
    - <content type="dict">
        <day type="int">15</day>
        <month type="int">2</month>
        <year type="int">2020</year>
      </content>
    </object3>
  - <object4 type="dict">
      <fname type="str">mahmood</fname>
      <sname type="str">jassim</sname>
      <tname type="str">ahmed</tname>
      <surname type="str">Al-Ani</surname>
```

Fig. 10: the produced XML file has a size of about 2.06 MB.

## IV. XML Results

As we have seen in the example above, XML is quite easy to generate, another benefit of XML is that namespaces and comments can be helped by putting metadata in tags as identifiers, manipulating data and measuring data.

XML also has the benefit of viewing it in a readable and organized fashion by browsers. This formatting is well provided by XML's tree structure which enables browsers to naturally crumble individual tree components i.e., strings that contain organized markups, to express the mixed material. The programmer only wants the marked-up text to be placed into a child tag of the parent in which it belongs in order to use XML. As JSON includes only data, there is no such easy way to indicate markup, similar to the metadata situation. Again, metadata will be expected to be stored as data that could be viewed as format abuse [10]. The XML is very stable because it uses honesty and authorization assurance, particularly when debugging. A node marked as a tree has been defined in the XML data model [11]. In architecture, on the other hand, XML requires the use of a rendering app, which means it must be parsed by a sluggish and bulky text parser. Because of the verbose and costs of parsing big XML files, many of these DOM manipulation libraries can contribute to the application using vast quantities of storage. It's also pretty verbose, indicating a bigger file size. The XML syntax often parallels other web-based bookmarking languages, which may contribute to confusion. JSON (JavaScript Object Notation) is a common lightweight replacement for XML

## V. Base64

Base64 [12] is a encoding scheme that converts binary data to an ASCII string format by using a radix-64 representation. Initially, the algorithm was named as “printable encoding” and was later changed to “Base64” in 1992. Base64 is one of the algorithms used to encode and decode data in ASCII format [13]. Base64 is designed for carrying stored data across channels in binary formats that only accept text content reliably [14]. Commons of all binary to text encoding systems are based on Base64. The general idea is to pick 64 characters widely used to most encoding and readable to humans and to use in many applications, including email, [15] by stored complex XML format database64 files. This will avoid data that were historically not 8-bit, being changed or lost in transit by information systems, such as email. "Base64" was developed as the earliest case of use for dialing communication between devices running the same operating system and may thus allow further decisions as to which characters are safe to use. For eg, unencoding uses upper case letters, numbers, and various dots, but not fewer [16].

## VI. JSON

JavaScript Object Notation (JSON) is an open-standard data interchange format [17] that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types. It uses human-readable text to store and transmit data objects [18]. JSON easy to process by machine and it is easy to manipulate by a user [19]. Is a data format that is based on the JavaScript programming language data types and it gained large popularity among web developers. JSON is a powerful format for exchanging and publishing data in many application fields, JSON combines well-known data structures and XML [15]. Also, JSON became the main format for the web to exchange data, the executing functions of the software ordered by remote machines that must establish a precise protocol for answering and receiving requests which called Application Programming Interface (API). It is a language that can be easily understood by the machines and the developers. Sending API responses and

requests over the (HTTP) protocol because JSON becomes the most familiar format [20]. Applications can communicate through a network usually using APIs4, using JSON data format. The key characteristics of JSON are versatility, accessibility and high expressivity [21]. JSON is non-proprietary, compact and technology-agnostic. The development (serializing), and consumption (deserialization), of JSON information are provided in all modern languages and platforms. JSON is easy to create and consists of elements that serve people such as objects, dictionaries and pairs of names / values. JSON datasets are typically collected from distant, unregulated sources with pieces of unfinished data or schema-free data [6]. Douglas Crockford was originally developed in 2001 and first standardized by RFC 4627 under IETF in 2006 [22].
In JSON's development several factors contributed, including:

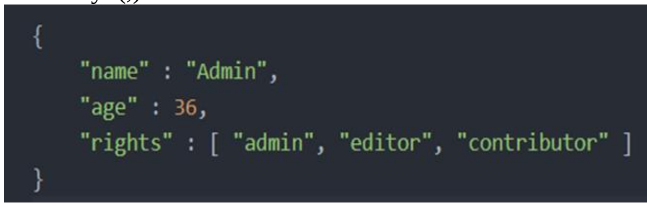
▪ Cessation of JSON-based explosive development of Restful APIs.

▪ Keep the essential constructs of JSON in mind.

▪ JavaScript's growing popularity enhances the popularity of JSON.

## A. *Generating and writing JSON*

JSON is constructed on two structures: name/value pair set. The object, record, struct, dictionary, hash table, wrench list, or related sequence is generally referred to as this in different languages. A list of values that has been ordered. There are universal data structures in most languages, also known as an array, vector, list or sequence. Almost all contemporary programming languages accept them in one way or another. It is also prudent to draw on these constructs in a database format interchangeable with programming languages. The unordered name/value pairs of a JSON entity. It starts from the right brace with "{"left strap and ends with"}." The colon and the name/value pair are divided by comma each name will obey. (,).



```
{
    "name" : "Admin",
    "age" : 36,
    "rights" : [ "admin", "editor", "contributor" ]
}
```

Fig. 11: JSON Object

## B. *JSON Syntax*

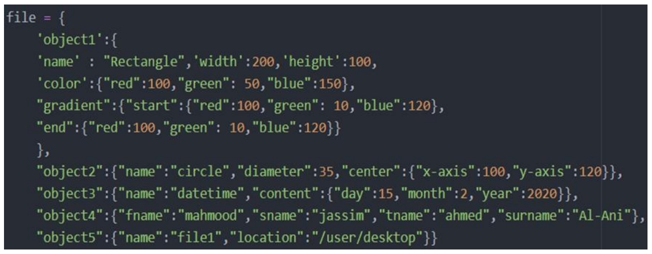
JSON syntax describes a sequence of Unicode code points. JSON also depends on Unicode in the hex numbers used in the \u escapement notation [23].
JSON JSON's critical features:

- Object and Array: on the two universal data structures, JSON is designed
- An object that holds several called value pairs, is labelled with braces ({}).

A column (:) is used to distinguish names and values for each pair of names.

Value is divided by commas while some pairs of name-value (,) (Jiang and Ahmed, 2017).
We will use python to write, generate and test some features of JSON file.

*1) First, we will need to prepare the objects that we want to convert into a JSON file following the rules mentioned above, for the sake of consistency, we will use the same objects in the XML example (figure (12)*

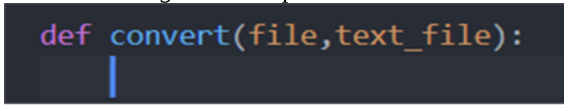


```
file = {
    'object1':{
    'name' : "Rectangle",'width':200,'height':100,
    'color':{"red":100,"green": 50,"blue":150},
    "gradient":{"start":{"red":100,"green": 10,"blue":120},
    "end":{"red":100,"green": 10,"blue":120}}
    },
    "object2":{"name":"circle","diameter":35,"center":{"x-axis":100,"y-axis":120}},
    "object3":{"name":"datetime","content":{"day":15,"month":2,"year":2020}},
    "object4":{"fname":"mahmood","sname":"jassim","tname":"ahmed","surname":"Al-Ani"},
    "object5":{"name":"file1","location":"/user/desktop"}}
```

Fig. 12: Using Objects

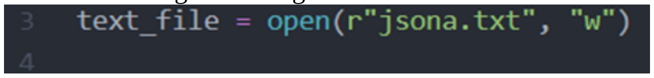
Objects here are defined as an array, in name/value pairs. Then, as we did in the earlier XML example, we will need to import the necessary modules into our python file, In this case, we will need to import the "json" module, which is one of python's standard built-in library modules that is used to parse, read and write JSON syntax.

*2) Next, we will define a function called "convert", this function will take two variables as input, first is the "file" variable which contains the array we want to convert, the second is called "text file", which is a file object that we will be writing JSON output to.*



```
def convert(file,text_file):
```

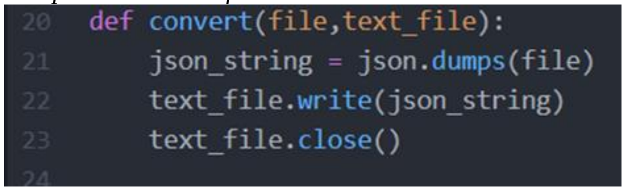
Fig. 13: Using Convert Function



```
text_file = open(r"jsona.txt", "w")
```

Fig. 14: File Object

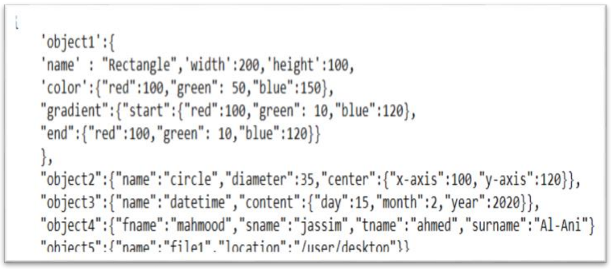
*3) Next, we will call a function called "dump" within the class "json" and place it's outputsin a variable called "json_string" using our "file" as an attribute. This function will parse our "file" array and convert it into JSON format. We will so call the "write" function within the previously define "text_file" object in order to write output into the text file.*



```
def convert(file,text_file):
    json_string = json.dumps(file)
    text_file.write(json_string)
    text_file.close()
```

Fig. 15: Dump Function

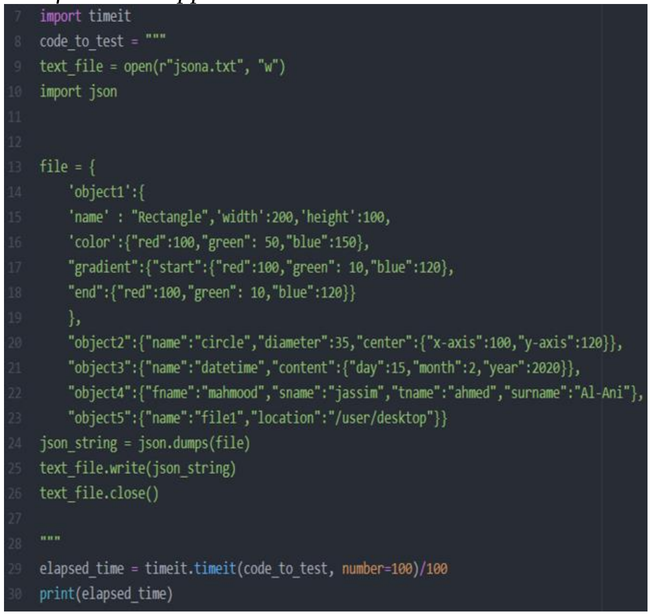
*4) Up to this stage, this code will convert the string values into JSON format and write it to a text file. The produced text file has a size of about 500kb, almost a quarter of the equivalent XML file.*



```
{
    'object1':{
    'name' : "Rectangle",'width':200,'height':100,
    'color':{"red":100,"green": 50,"blue":150},
    "gradient":{"start":{"red":100,"green": 10,"blue":120},
    "end":{"red":100,"green": 10,"blue":120}}
    },
    "object2":{"name":"circle","diameter":35,"center":{"x-axis":100,"y-axis":120}},
    "object3":{"name":"datetime","content":{"day":15,"month":2,"year":2020}},
    "object4":{"fname":"mahmood","sname":"jassim","tname":"ahmed","surname":"Al-Ani"}
    "object5":{"name":"file1","location":"/user/desktop"}}
```

Fig. 16: Convert the String Values into JSON Format and Write it to a Text File

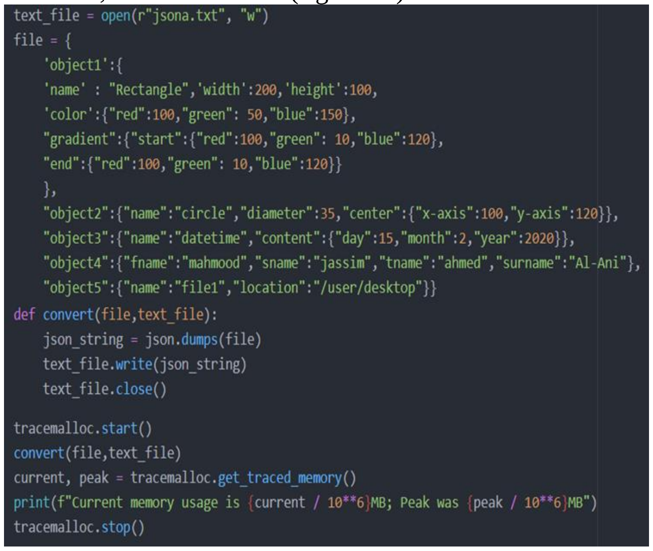
*5) Next, we will add some extra lines of code to measure the execution speed of the code and memory usage, to get some data that will help us in comparison between XML and JSON. The first module we will use "timeit", it's a standard library module used to find the execution time of a code snippet.*



```
import timeit
code_to_test = """
text_file = open(r"jsona.txt", "w")
import json


file = {
    'object1':{
    'name' : "Rectangle",'width':200,'height':100,
    'color':{"red":100,"green": 50,"blue":150},
    "gradient":{"start":{"red":100,"green": 10,"blue":120},
    "end":{"red":100,"green": 10,"blue":120}}
    },
    "object2":{"name":"circle","diameter":35,"center":{"x-axis":100,"y-axis":120}},
    "object3":{"name":"datetime","content":{"day":15,"month":2,"year":2020}},
    "object4":{"fname":"mahmood","sname":"jassim","tname":"ahmed","surname":"Al-Ani"},
    "object5":{"name":"file1","location":"/user/desktop"}}
json_string = json.dumps(file)
text_file.write(json_string)
text_file.close()

"""
elapsed_time = timeit.timeit(code_to_test, number=100)/100
print(elapsed_time)
```

Fig. 17: Measure the Execution Speed

The other module we used is called "tracemalloc", this module will profile the memory allocated for execution of this code, as shown below (figure 18).



```
text_file = open(r"jsona.txt", "w")
file = {
    'object1':{
    'name' : "Rectangle",'width':200,'height':100,
    'color':{"red":100,"green": 50,"blue":150},
    "gradient":{"start":{"red":100,"green": 10,"blue":120},
    "end":{"red":100,"green": 10,"blue":120}}
    },
    "object2":{"name":"circle","diameter":35,"center":{"x-axis":100,"y-axis":120}},
    "object3":{"name":"datetime","content":{"day":15,"month":2,"year":2020}},
    "object4":{"fname":"mahmood","sname":"jassim","tname":"ahmed","surname":"Al-Ani"},
    "object5":{"name":"file1","location":"/user/desktop"}}
def convert(file,text_file):
    json_string = json.dumps(file)
    text_file.write(json_string)
    text_file.close()

tracemalloc.start()
convert(file,text_file)
current, peak = tracemalloc.get_traced_memory()
print(f"Current memory usage is {current / 10**6}MB; Peak was {peak / 10**6}MB")
tracemalloc.stop()
```

Fig. 18: Tracemalloc Module

## VII. JSON Results

The final result shows that this code took about 3/10000 of a second to execute with peak memory usage of 0.00038 MB, those are much lower values when compared to XML, which took about 5/1000 of a second to execute with about 0.0042 MB peak memory usage. This is mainly why JSON is much more efficient over XML and why it is becoming an industry standard.

Because of it simpler and smaller syntax, JSON is faster to parse and process as we have seen in the previous examples. It also has a wide range of support and compatible browsers and operational systems, which means less workload for a developer working with it; it is also less verbose, meaning that it uses much more words than XML which results in much smaller file size. And has data structures that are very similar to primitive data structures used by most programming language.

Another strong point of JSON is faster server-side parsing, which means that the user will get a faster response from the back-end processes of the application, It is also more easy to create and use and is human friendly. On the other hand, Unlike XML, JSON does not provide display capabilities as it is not a markup language and cannot include Meta data; it is also less secure than XML. Also, JSON supports only text and number data type, which is one of the down sides of using JSON. One of those issues is the transmission of binary data over a textual medium. When data is transmitted over a medium, the data will be viewed in the same format as expected, we cannot be sure. Any systems can misunderstand or even drop the most important piece of data. In comparison, the disparity between system end codes means that the character 10 and 13 of the ASCII are often altered as well.

Most computers often store 8-bit, but not all, files. Such transmitting devices and media can accommodate just 7 bits or less simultaneously. Such a medium can view the stream in more than 7 bits so you won't get what you want from the other side if you send 8 bits.

## VIII. Binary Data and JSON

Binary data is not allowed in the JSON format. It is important to escape binary data to be interpreted as a string variable (i.e., zero or more Unicode chars with backslash escapes in double quotes). UTF-8 is available, but it does not encrypt the UTF-8 binary data as spatially efficient as 150 percent of their initial size.

Incorrect UTF-8 encoding also includes several random binary byte strings. The advantage of this constraint on UTF-8 is that multi byte characters start and end whatever byte are robust and feasible.

As a result, if the byte encoding inside [0...127] only takes one byte in the UTF-8 encoding, it will take 2 bytes to encrypt the byte value within the range [128...255]! Worse than that. Worse than that. Check characters cannot be shown in a string in JSON. The binary data will thus have to be accurately encoded for any transformation.

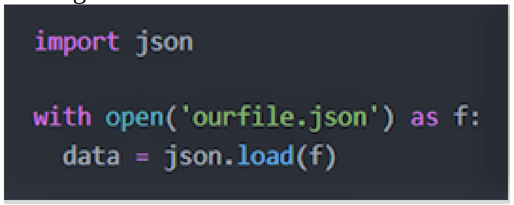
In UTF-8, it is the most space efficient to encode a 128 ASCII value. You will save 7 bits in 8 bits. If the binary data is then broken down in 7-bit chunks, so that the data encoded expands to 114% of the original byte in a UTF-8 encoded

string. More than Base64. Better. Sadly, we can't use this simple trick as certain ASCII chars are not allowed by JSON. The ASCII (([0...31] and 127) control characters and \ are excluding, which leaves 128-35 = 93 characters alone.
In principle, we might then describe the Based93 encodes, which would increase the encoded size to eight/log2(93) = eight*log10(2)/log10(93) = 122%. But the Base93 encoding would not be as simple as the Base64 one. Base64 cuts down in 6bit-chunks the input byte set, which is simple by bit [24].

## IX. Encoding Binary elements within JSON with Base64

To further understand the advantages of using base64 encoding scheme, we will go through the process of converting a JSON dictionary into, first a binary bit stream and then into a base64 representation, to achieve this:

• The first step is to start with a JSON file that will contain an inventory of items with different attributes. First, we import the "json" module into our environment and parse our json file using it.

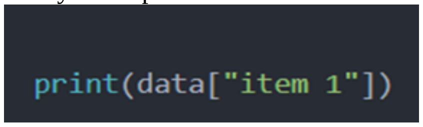


```
import json

with open('ourfile.json') as f:
  data = json.load(f)
```

Fig. 19:"json.load" Method to Parse "ourfile.json"

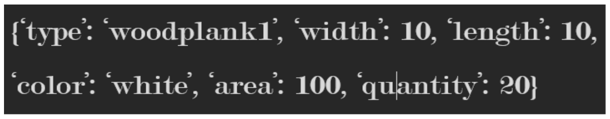
As we can see above, we have used the "json.load" method to parse "ourfile.json" and convert it into a python dictionary.

• Using print command to make sure that the elements are accessible in key/value pair format.



```
print(data["item 1"])
```

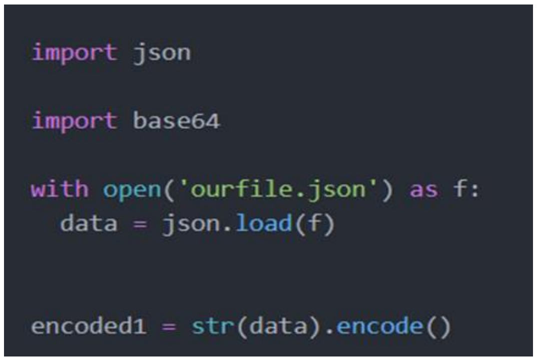
Fig. 20: using print command



```
{'type': 'woodplank1', 'width': 10, 'length': 10,
'color': 'white', 'area': 100, 'quantity': 20}
```

Fig. 21: key/value pair format

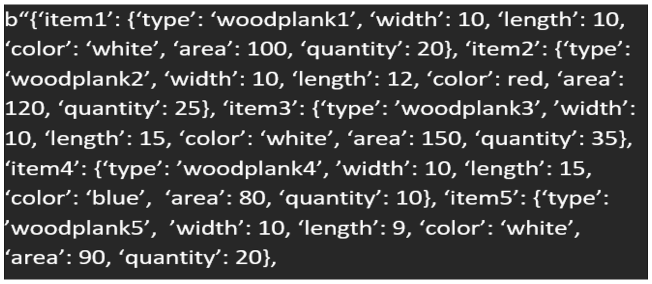
• Next, the following step is: encode the elements within our dictionary using basee64 format, it is worth mentioning that all string values must first be converted first into a string, and then into bytes objects before it can be encoded using base64, in this case, we will use "UTF-8" format.



```
import json

import base64

with open('ourfile.json') as f:
  data = json.load(f)


encoded1 = str(data).encode()
```

Fig. 22: Using encode Function

• As we can see in the figure above, we used ".encode()" function to encode the string representation of our value in a "UTF-8" format.

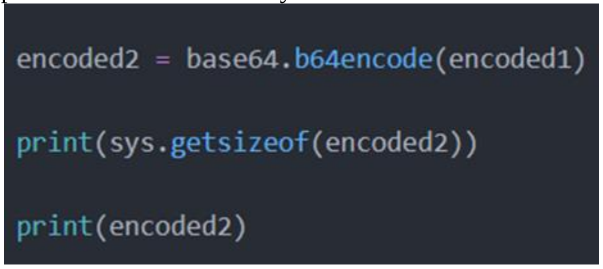


```
b"{'item1': {'type': 'woodplank1', 'width': 10, 'length': 10,
'color': 'white', 'area': 100, 'quantity': 20}, 'item2': {'type':
'woodplank2', 'width': 10, 'length': 12, 'color': red, 'area':
120, 'quantity': 25}, 'item3': {'type': 'woodplank3', 'width':
10, 'length': 15, 'color': 'white', 'area': 150, 'quantity': 35},
'item4': {'type': 'woodplank4', 'width': 10, 'length': 15,
'color': 'blue',  'area': 80, 'quantity': 10}, 'item5': {'type':
'woodplank5',  'width': 10, 'length': 9, 'color': 'white',
'area': 90, 'quantity': 20},
```

Fig. 23: Values

• We can also get the size of the encoded string file using "sys" library, in this case, the resulted string file is 550 bytes in size.

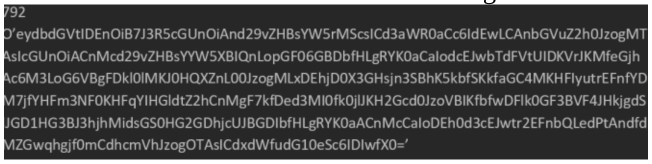
• Next, using the "base64" module to encode our string representation of a dictionary.



```
encoded2 = base64.b64encode(encoded1)

print(sys.getsizeof(encoded2))

print(encoded2)
```

Fig. 24: Encode String Representation

• We used the "b64encode" method inside the "base64" class to encode our string, we also used the "sys.getsizeof" standard library method to calculate the size of the string after encoding, and it resulted in a base64 hash of 729 bytes, which is roughly 30 percent more than the string file, which is the standard increase in size when encoding in base64.



```
792
O'eydbdGVtIDEnOiB7J3R5cGUnOiAnd29vZHBsYW5rMScsICd3aWR0aCc6IdEwLCAnbGVuZ2h0JzogMT
AsIcGUnOiACnMcd29vZHBsYYW5XBIQnLopGF06GBDbfHLgRYK0aCalodcEJwbTdFVtUIDKVrJKMfeGjh
Ac6M3LoG6VBgFDkl0lMKJ0HQXZnL00JzogMLxDEhjD0X3GHsjn3SBhK5kbfSKkfaGC4MKHFlyutrEFnfYD
M7jfYHFm3NF0KHFqYIHGldtZ2hCnMgF7kfDed3MI0fk0jlJKH2Gcd0JzoVBIKfbfwDFlk0GF3BVF4JHkjgdS
JGD1HG3BJ3hjhMidsGS0HG2GDhjcUJBGDIbfHLgRYK0aACnMcCaloDEh0d3cEJwtr2EFnbQLedPtAndfd
MZGwqhgjf0mCdhcmVhJzogOTAsICdxdWfudG10eSc6IDIwfX0='
```

Fig. 25Encode String

While it is true that base64 has ~33% expansion rate, it is not necessarily true that processing overhead is significantly more than this: it really depends on JSON library/toolkit you are using. Encoding and decoding are simple straight-forward operations, and they can even be optimized with character encoding (as JSON only supports UTF-8/16/32) -- base64 characters are always single-byte for JSON String entries. For example, on Java platform there are libraries that

can do the job rather efficiently, so that overhead is mostly due to expanded size.
Base64 is simple, commonly used standard, so it is unlikely to find something better specifically to use with JSON (base-85 is used by postscript etc. but benefits are at best marginal) compression before encoding (and after decoding) may make lots of sense, depending on data you use.

## X. Conclusion

Our findings reflect that encoding data in Base64 prior to transmission will present many advantages including readability and integrity, it will also enable us to transmit binary data over textual mediums, 7 Bit protocols such as SMTP and different network hardware without risking misinterpretation. Although data bloating may occur, since In case of encoding, the Base64 algorithm replaces each 3 bytes with a total of 4 bytes and adds padding characters if necessary, thereby resulting in a number of four characters always. The result will always be 33 percent bigger than the original data (more accurately, 4⁄3). The formula for calculating the result string length without padding is n*4/3 where n is the original data length. and some extra processing would be required to encode or decode, my findings shows that it does not overshadows the aforementioned advantages. Also, Although XML has been used extensively by the industry for a long time due to its security and ability to represent new types of data tags that other markup languages cannot, it's also capable of presenting data in an organized, hierarchal and readable manner. our findings also shows that JSON has many advantages over XML, and are the reason why its gradually becoming the standard for interchanging data between multiple stacks, those advantages include less verbosity, which translates to smaller file size, which is a critical component of an efficient system, it also has a structure that is identical to data types used by developer and, Unlike XML, doesn't require an application to parse and manipulate data, which greatly reduces its processing overhead. For those reasons, JSON was used to illustrate the process of encoding JSON strings with Base64.

## Conflict of Interest

The authors have no conflict of relevant interest to this article.

## References

[1] L. Irshad, L. Yan, and Z. Ma, "Schema-Based JSON Data Stores in Relational Databases," *J. Database Manag.*, vol. 30, no. 3, pp. 38–70, 2019.

[2] M. S. Marev, E. Compatangelo, and W. Vasconcelos, "Towards a context-dependent numerical data quality evaluation framework Technical Report," *arXiv*, pp. 1–12, 2018.

[3] A. Olusola Olajide, *Number System*. Adekunle Ajasin University, 2017.

[4] S.R.Chaudhari, "Number Systems," in *Principles of Digital Electronics*, 5th ed., Pune: licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License., 2019.

[5] A. Kelbert, "EMTF XML: New data interchange format and conversion tools for electromagnetic transfer functions," *Geophysics*, vol. 85, no. 1, pp. F1–F17, 2020, doi: 10.1190/geo2018-0679.1.

[6] M. K. Yusof and M. Man, "Efficiency of JSON for data retrieval in big data," *Indones. J. Electr. Eng. Comput. Sci.*, vol. 7, no. 1, pp. 250–262, 2017, doi: 10.11591/ijeecs.v7.i1.pp250-262.

[7] Suha Mohammed Hadi, "A New Approach for Designing Multi Information Management System Using XML Technology," *Al-khwarizmi Eng. J.* , vol. 1, no. 1, pp. 46–51, 2005.

[8] C. Späth, C. Mainka, V. Mladenov, and J. Schwenk, "Sok: XML parser vulnerabilities," *10th USENIX Work. Offensive Technol. WOOT 2016*, 2016.

[9] B. Li and J. Hu, "Analysis of the HTML to XML Conversion Method," no. Isci, pp. 64–69, 2015, doi: 10.2991/isci-15.2015.10.

[10] M. Naseriparsa, C. Liu, M. S. Islam, and R. Zhou, "XPloreRank: exploring XML data via you may also like queries," *World Wide Web*, vol. 22, no. 4, pp. 1727–1750, 2019, doi: 10.1007/s11280-018-0630-x.

[11] H. Bohring and O. Auer, "Mapping XML to OWL ontologies," *Lect. Notes Informatics (LNI), Proc. - Ser. Gesellschaft fur Inform.*, vol. P-72, pp. 147–156, 2005.

[12] Abdullah A. Abdullah, "Enhancing Cost and Security of Arabic SMS Messages over Mobile Phone Network," *AL-Rafidain J. Comput. Sci. Math.*, vol. 6, no. 3, pp. 111–127, 2009.

[13] H. Nurdiyanto, R. Rahim, and N. Wulan, "Symmetric Stream Cipher using Triple Transposition Key Method and Base64 Algorithm for Security Improvement," *J. Phys. Conf. Ser.*, vol. 930, no. 1, 2017, doi: 10.1088/1742-6596/930/1/012005.

[14] L. Cantara, "METS: The Metadata Encoding and Transmission Standard," *Cat. Classif. Q.*, vol. 40, no. 3–4, pp. 237–253, Sep. 2005, doi: 10.1300/J104v40n03_11.

[15] A. M. Logunleko, K. B. Logunleko, and O. O. Lawal, "An End-to-End Secured Email System using Base64 Algorithm," *Int. J. Comput. Appl.*, vol. 175, no. 28, pp. 1–6, 2020, doi: 10.5120/ijca2020920669.

[16] G. J. Sussman and J. Sussman, "Structure and interpretation of computer programs, (second edition)," *Comput. Math. with Appl.*, vol. 33, no. 4, p. 133, 1997, doi: 10.1016/s0898-1221(97)90051-1.

[17] D. Petković, "JSON Integration in Relational Database Systems," *Int. J. Comput. Appl.*, vol. 168, no. 5, pp. 14–19, 2017, doi: 10.5120/ijca2017914389.

[18] M. B. S. Narayana, H. Khalifa, and W. van der Aalst, "JXES: JSON support for the XES event log standard," *arXiv*, no. i, 2020.

[19] Jiří Papoušek, "Traffic Flow Processing in JSON Format," *Spring*, 2020.

[20] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of JSON Schema," in

*Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 263–273, doi: 10.1145/2872427.2883029.

[21] Y. Li, N. R. Katsipoulakis, B. Chandramouli, J. Goldstein, and D. Kossmann, "Mison: A Fast JSON Parser for Data Analytics," *Proc. VLDB Endow.*, vol. 10, no. 10, pp. 1118–1129, Jun. 2017, doi: 10.14778/3115404.3115416.

[22] P. Bourhis, J. L. Reutter, F. Suárez, and D. Vrgoč, "JSON: Data Model, Query Languages and Schema Specification," in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2017, pp. 123–135, doi: 10.1145/3034786.3056120.

[23] ECMA International, "The JSON Data Interchange Syntax," *Stand. ECMA-404*, vol. 2nd Editio, no. December 2017, p. 8, 2017, [Online]. Available: http://www.ecma-international.org/publications/standards/Ecma-404.htm.

[24] M. T. Goodrich and R. Tamassia, *Data Structures and Algorithms in Java*. 2020.