JavaScript Object Notation (JSON)

JSON basics

JSON stands for **J**ava**S**cript **O**bject **N**otation, is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is often used for data exchange between a server and a web application or between different systems.

JSON is a text-based data format, which means it is represented as a series of characters that can be easily transmitted over the internet or stored in files. It represents data as **key-value** pairs. Each **key** is a string enclosed in **double quotes**, followed by a colon, and then a value. The **key-value** pairs are separated by commas. **JSON** values can be strings, numbers, booleans, null, arrays, or nested **JSON** objects.

For example:

```
{
  "firstName": "John",
  "lastName": "Doe",
  "age": 30,
  "isStudent": false,
  "hobbies": ["reading", "swimming"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "zipCode": "12345"
  }
}
```

In this example:

"firstName", "lastName", "age", and **"isStudent"** are keys with corresponding values.

"hobbies" is an array containing strings.

"address" is a nested **JSON** object with its own **key-value** pairs.

JSON supports several data types including:

Strings: That enclosed in **double quotes**.
Numbers: **Integers** or **floating-point** numbers.
Booleans: **true** or **false**.
Null: Represents the absence of a value.
Arrays: Ordered lists of values enclosed in **square brackets**.
Objects: Unordered collections of **key-value** pairs enclosed in **curly braces**.

Common Use Cases:

Configuration files: JSON is often used to configure applications.
API Responses: Many web APIs return data in **JSON** format.
Data Storage: JSON is used for storing structured data in **NoSQL** databases.
JavaScript Data Exchange: JSON is a natural format for data transfer between **JavaScript** applications and **web servers**.

Syntax Rules:

Keys must be enclosed in double quotes.
Values can only be one of the supported data types.
Comma separates **key-value** pairs.
Curly braces {} enclose **objects**.
Square brackets [] enclose **arrays**.

Parsing and Generation: Most programming languages provide built-in functions or libraries to parse JSON strings into data structures and generate JSON strings from data structures.

The difference with JavaScript's Object

JavaScript objects are fundamental data structures that allow you to store and organize data. They consist of **key-value** pairs and can represent complex, structured information.

Consider these two examples:

JavaScript's Object

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  isStudent: false,
};
```

JSON

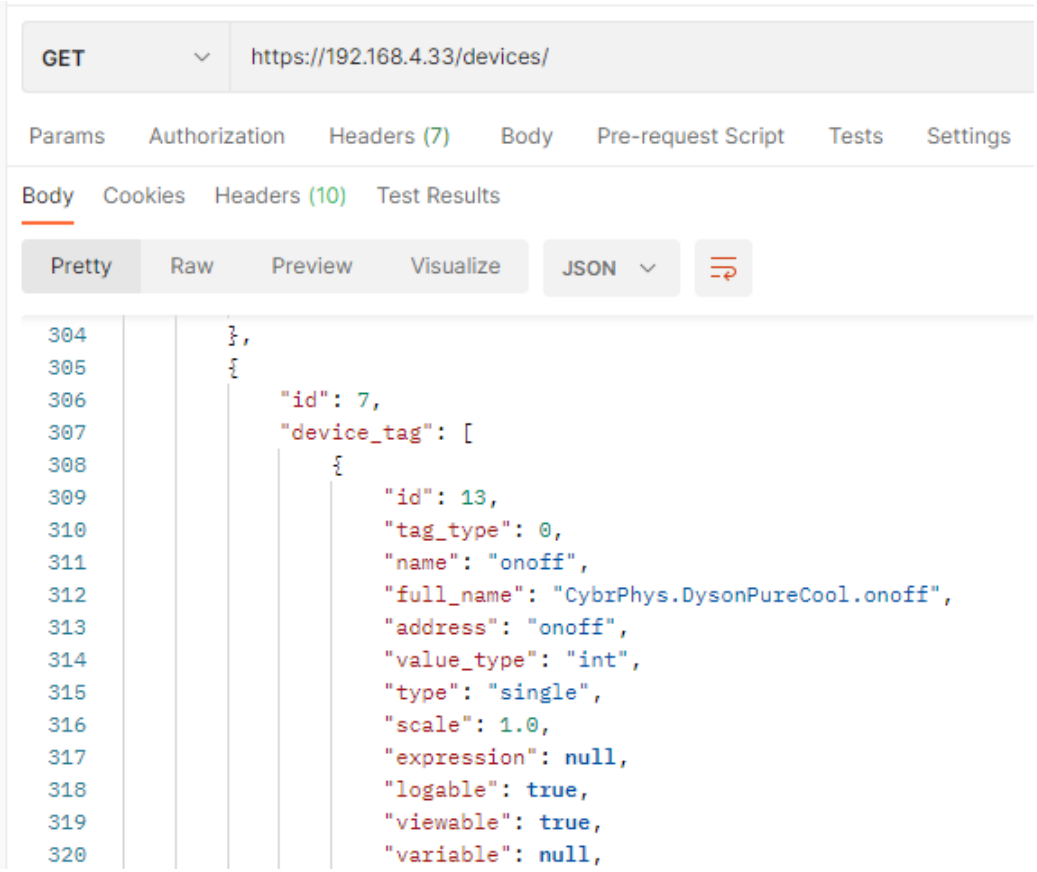
```
{
  "firstName": "John",
  "lastName": "Doe",
  "age": 30,
  "isStudent": false
}
```

From this example, it shows that **Person** object with the same meaning but in different format. It can be seen that while both are **key-value** pairs, there are some different in style and limitation. Such as:

- **JSON** has a stricter syntax. **Keys** must be enclosed in double quotes, and values follow a specific set of data types.
- **JavaScript objects** are used for data manipulation within JavaScript code. **JSON** is primarily used for data interchange between systems or for configuration files.
- **JavaScript objects** can store any data type, including **functions** and **undefined** values. **JSON** supports a limited set of data types for compatibility and data integrity.
- **JavaScript object keys** do not require **double quotes**, while **JSON keys** must have **double quotes**.

Example Application of JSON

API Response Example:



This is one example of **RestAPI** response from "161.246.5.33" API server to the request for "devices". And its return data is in the format of JSON.

Web Server Configuration Example:

In this example:

appName and **version** provide information about the application itself.

server contains server-related configuration, including **host** and **port** settings.

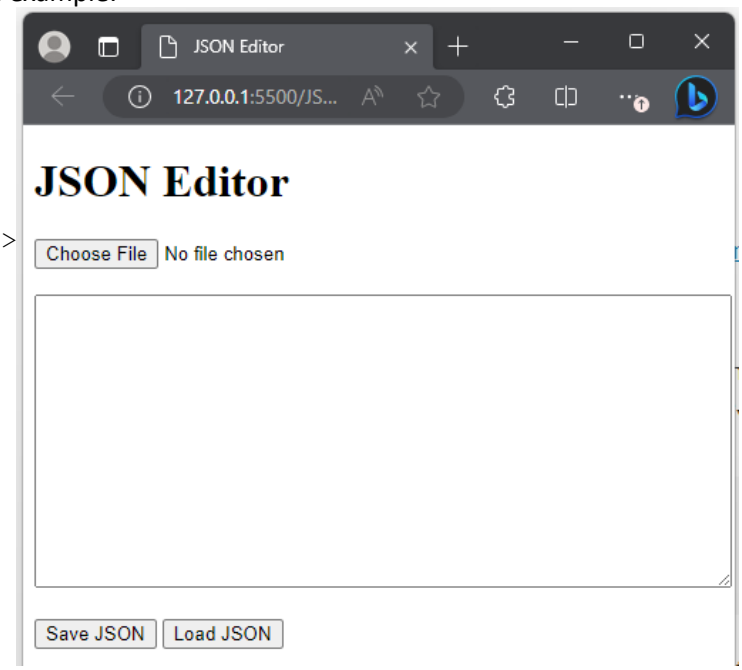
database contains database connection information, such as **host**, **port**, **username**, **password**, and **database name**.

features is an object that specifies various features of the application. In this case, it indicates whether analytics and notifications are enabled.

colors define color schemes for the application's user interface.

You can apply the idea of using this **JSON** configuration file in a **JavaScript** program to set up the program behavior and appearance. For example, you could read this configuration file when the program starts and use the values to configure the server connection, database connection, feature toggles, and UI styles.

Simple **JSON Editor** on HTML web page example:




```
<!DOCTYPE html>
<html>
<head>
    <title>JSON Editor</title>
    <style>
        textarea {
            width: 100%;
            height: 200px;
        }
    </style>
</head>
<body>
    <h1>JSON Editor</h1>
    <!-- Input for uploading JSON file -->
    <input type="file" id="fileInput" accept=".json">
    <br><br>
    <!-- Textarea for displaying and editing JSON data -->
    <textarea id="jsonTextArea"></textarea>
    <br><br>
    <!-- Save and Load buttons -->
    <button onclick="saveJSON()">Save JSON</button>
    <button onclick="loadJSON()">Load JSON</button>
    <script>
        const fileInput = document.getElementById('fileInput');
        const jsonTextArea = document.getElementById('jsonTextArea');
        // Function to handle file selection and load JSON data
```

```
        fileInput.addEventListener('change', function (e) {
            const file = e.target.files[0];
            if (file) {
                const reader = new FileReader();
                reader.onload = function (e) {
                    const content = e.target.result;
                    jsonTextArea.value = content;
                };
                reader.readAsText(file);
            }
        });
        // Function to save JSON data as a file
        function saveJSON() {
            const jsonContent = jsonTextArea.value;
            const blob = new Blob([jsonContent],
                                  { type: 'application/json' });
            const url = URL.createObjectURL(blob);
            const a = document.createElement('a');
            a.href = url;
            a.download = 'data.json';
            a.click();
            URL.revokeObjectURL(url);
        }
        // Function to load JSON data from textarea
        function loadJSON() {
            const jsonContent = jsonTextArea.value;
            try {
                const data = JSON.parse(jsonContent);
                // You can work with the parsed JSON data here
                console.log(data);
            } catch (error) {
                alert('Invalid JSON format');
            }
        }
    </script>
</body>
</html>
```

In this example:

- There is an **<input type="file">** element that allows users to select a JSON file to upload.

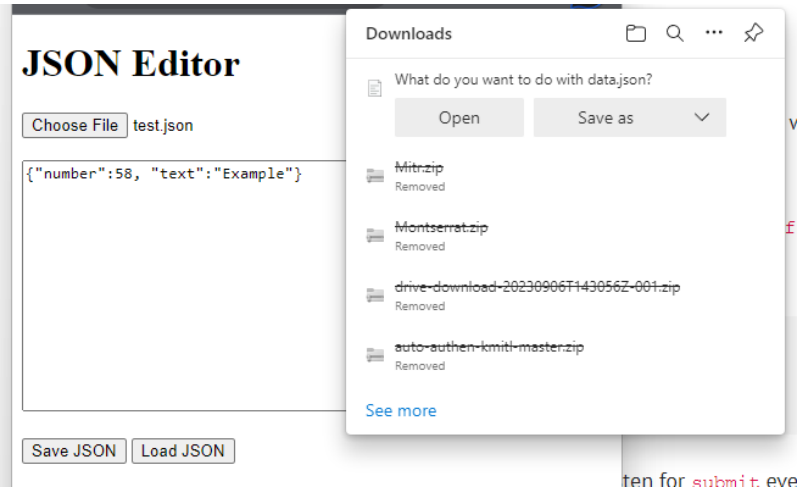
- A **<textarea>** element is used to display and edit the JSON data.

- There are **"Save JSON"** and **"Load JSON"** buttons that trigger JavaScript functions.

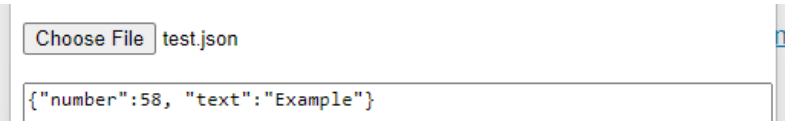
- The JavaScript code handles file selection, loading **JSON** data, and saving **JSON** data as a file.

- When a file is selected, its content is read and displayed in the **textarea**.

- Users can edit the **JSON** data in the **textarea** and click **"Save JSON"** to save it as a file.



- Clicking **"Load JSON"** attempts to parse the **JSON** data in the **textarea**, and if successful, the parsed data is logged to the console.



This example provides a basic use case of JSON for uploading, displaying, and editing JSON files in an HTML page. You can further enhance and customize it according to your specific requirements.
