Hue Learn

Wise Work

DNAi World

JSON Guidelines

Coding Best Practices

by Hue Learn, Wise Work, DNAi World

Contents

1. Introduction

JSON(JavaScript Object Notation) is now a language-independent data format and code for parsing. We use JSON extensively for REST APIs, serializing messages to queues. When working with JSON, it's important to follow certain guidelines to ensure that your data is easy to read, understand, and maintain. Here are some key JSON coding guidelines:

2. Consistent Formatting

1. Indentation: Use consistent indentation (commonly 2 or 4 spaces) to enhance readability. Avoid tabs.
2. Line Breaks: Use line breaks after each key-value pair in objects and after each item in arrays.

3. Values

1. Strings: Always enclosed in double quotes (“).
2. Numbers: Use standard number notation, without quotes.
3. Booleans: Represented as true or false without quotes.
4. Null: Represented as null without quotes.

4. Trailing Commas

- Avoid trailing commas after the last key-value pair in objects or the last element in arrays.

5. Proper Structure

1. JSON data is structured as key-value pairs.

- Data structures include:
  - Objects: Collections of key-value pairs, enclosed in curly braces {}.

Right Coding Style.

```
{"name":"Chris", "age":23, "city":"New York"}
```

- Arrays: Ordered lists of values, enclosed in square brackets [ ]

Right Coding Style.

```
[
    { "name": "Chris", "age": 23, "city": "New York" },
    { "name": "Emily", "age": 19, "city": "Atlanta" },
    { "name": "Joe", "age": 32, "city": "New York" },
    { "name": "Kevin", "age": 19, "city": "Atlanta" },
    { "name": "Michelle", "age": 27, "city": "Los Angeles" },
    { "name": "Robert", "age": 45, "city": "Manhattan" },
    { "name": "Sarah", "age": 31, "city": "New York" }
]
```

- JSON Objects and arrays can contain nested objects and arrays. You can have multiple nesting levels.

Right Coding Style.

```
{
    "name": "Chris",
    "age": 23,
    "address": {
        "city": "New York",
        "country": "America"
    },
    "friends": [
        {
            "name": "Emily",
            "hobbies": [
                "biking",
                "music",
                "gaming"
            ]
        },
        {
            "name": "John",
            "hobbies": [
                "soccer",
                "gaming"
            ]
        }
    ]
}
```

6. Key Naming Conventions

1. Meaningful Names: Use descriptive and meaningful names for keys.
2. Consistent Case: Use a consistent case style for keys, such as camelCase, snake_case, or kebab-case. camelCase is commonly used in JSON.

Right Coding Style.

```
{
  "firstName": "John",
  "lastName": "Doe"
}
```

7. Handling Special Characters

- Escape special characters within strings, such as newlines (\n), tabs (\t), backslashes (\), and quotes (").

8. Whitespace

- Include spaces after colons (:) and commas (,). This improves readability.

Right Coding Style.

```
{
  "name": "Alice",
  "age": 30
}
```

9. Empty Structures

- Use {} for empty objects and [] for empty arrays.

10. Encoding

- Ensure JSON is encoded in UTF-8, especially when dealing with international characters.

11. Comments

- JSON does not support comments. Avoid including comments in JSON data. If needed, use external documentation to explain the data structure.

12. Avoid Duplication

- Avoid duplicate keys within an object, as this is invalid in JSON.

13. Formatter

13.1. Prettier:

A code formatter that supports JSON along with many other languages. It ensures consistent styling, and can be easily integrated into Visual Studio Code or other editors.

13.2. JSON Formatter & Validator (Online):

A simple and easy-to-use online tool for formatting and validating JSON.

14. Streams

- Use streams whenever possible.
- Most JSON parsing libraries can read straight from a stream instead of a string. This is a little more efficient and preferred where possible.

15. Security Considerations

- Be cautious with sensitive information.

Wrong Coding Style.

```
{
    "userId": 12345,
    "name": "Jane Doe",
    "email": "jane.doe@example.com",
    "isActive": true,
    "profile": {
        "age": 28,
        "gender": "female",
        "languages": [
            "English",
            "Spanish",
            "French"
        ]
    },
    "preferences": {
        "notifications": {
            "email": true,
            "sms": false
        },
        "theme": "dark"
    }
}
```

16. Linter

16.1. ESLint (for JSON)

While ESLint is primarily a JavaScript linter, it also has plugins that can lint JSON files. It checks for malformed JSON and style issues.

16.2. jsonlint

A straightforward linter for JSON files that helps validate and format JSON content. It can be used as an npm package or online.

17. Compress your JSON

- Since JSON is just simple text, you can expect to get up to 90% compression. So use gzip wherever possible when communicating with your web services.

18. Parsing in JavaScript

- JavaScript natively supports JSON through the JSON object. To parse JSON using JavaScript, developers use the JSON.parse() method, converting the JSON data into a JavaScript object.

Right Coding Style.

```
{
  var jsonData = '{"name":"John", "age":30, "city":"New York"}';
  var obj = JSON.parse(jsonData);
  alert(obj.name);
}
```

19. Parsing in Python

Developers invoke the json.loads() method to parse JSON data. Here's a Python example:

Right Coding Style.

```
{
  import json
  jsonData = '{"name":"John", "age":30, "city":"New York"}';
  data = JSON.loads(jsonData)
  print(data['name'])

}
```

- Take note that converting JSON data into another data structure (for instance, a Python dictionary or JavaScript object) is called deserialization.
- **Avoid parsing JSON if you don't need to**
- When using something like ASP.NET Web API, don't define your methods expecting specific classes as incoming data and instead read the post body so ASP.NET never parses the JSON.

20. Shorter Field Names

- Using smaller field names can give you a 5-10% parsing performance boost and of course slightly smaller data packets being passed around.

21. JSON Schema Validation

- JSON schema validation ensures code standardization and aids in catching inconsistencies early and avoids future problems. A well-implemented validation process helps maintain the robustness of data interchange.

Take a look at this simple example of JSON schema:

Right Coding Style.

```
{
    "type": "object",
    "properties": {
        "firstName": {
            "type": "string"
        },
        "age": {
            "type": "integer",
            "minimum": 0
        }
    },
    "required": [
        "firstName",
        "age"
    ]
}
```

22. Data Member Attributes

- Check your JSON library settings to see how you can ignore specific fields, omit null values, etc. Most .NET libraries will use Data Contract/Data Member attributes and settings.

Right Coding Style.

```
{
  [DataContract]
  public class Monitor
  {
      [DataMember(Name = "id")]
      public int MonitorID { get; set; }
  }
}
```

23. Conclusion

By following these guidelines, you can ensure that your JSON data is well-organized, easy to understand, and free from common errors.