JSON

JSON

- JavaScript Object Notation
- Formato ligero para el intercambio de datos
- Soporta objetos, arrays y otros valores
- No permiten comentarios
- No se garantiza el orden del contenido de los objetos
- Los elementos de un array si que están ordenados

JSON - Sintaxis

- Objeto
  - Set desordenado de pares clave / valor
    - "clave" : valor
  - Rodeado por llaves { }
  - La clave es un String y tiene que tener comillas dobles
  - La clave debería usar nomenclatura lower camel case
  - Los datos se separan con comas
  - Puede estar vacío { }

Objeto:

```
{
  "nombreCompleto": "Juan Pérez Rodríguez",
  "edad": 27      ← Cuidado con la trailing comma
}
```

Array:

- Colección ordenada de valores
- Rodeado por corchetes: [y]

```
[
  {"movil": 612345678},
  {"fijo": 912345678}
]
```

```
{
  "nombre": "Juan"
```

```
  "nombre" : "Juan",
  "direccion": {
    "calle": "Avenida Ciudad de Barcelona 23",
    "ciudad": "Madrid"
  },
  "telefonos": [
    {"movil": 612345678},
    {"fijo": 912345678}
  ],
  "edad": 27
}
```

- Tiene que haber un único elemento raíz que tiene que ser un valor

```
[
  {"nombre" : "Juan"},
  {"nombre" : "Ana"},
  {"nombre" : "Sofía"},
  {"nombre" : "Andrés"}
]
```

JSON con JS

- Paquete JSON
  - No es específico de Node.js
- Convertir un objeto JSON a texto
  - let text = JSON.stringify(obj);
- Convertir un texto en formato JSON a texto
  - let obj = JSON.parse(text)

JSON Schema

- Vocabulario para anotar y validar documentos JSON

- Estructura básica:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/product.schema.json",
  "title": "Product",
  "description": "A product in the catalog",
  "type": "object"
```

```
  "properties": {…}
  "required": […]
}
```

Estructura básica:

- $schema: Versión de JSON Schema
- $id: URI base del esquema
- title y description: anotaciones descriptivas sin implicaciones en la validación
- type: valor del elemento raíz
- properties: objeto con las keywords
  - para cada keyword
    - description: anotación con la descripción
    - type: valor del elemento
- required: array de propiedades que son obligatorias

JSON Schema - types

- string
- number
  - tanto números enteros como decimales
- integer
  - puden ser decimales que terminen en 0
  - 1.0
- object
- array
- boolean
- null

JSON no distingue entre number y integer JSON Schema si

JSON Schema

- type también puede ser un array si acepta varios tipos
  - "type" : ["number", "string"]
- En vez de type podemos usar enum para listar los valores válidos
  - "enum": ["red", "amber", "green", null, 42]
- Si solo admitimos un valor podemos usar const
  - "const": "Spain"

Rangos de number e integer

- minimum: incluyendo el valor
- exclusiveMinimum (*)
- maximum: incluyendo el valor
- exclusiveMaximum (*)

Multiplos

- multipleOf
  - Restringirlo a un múltiplo de un valor

Para un string

- minLength
- maxLength
- format
  - unos formatos predefinidos
    - date-time
    - email
    - uri
- pattern
  - permite definir una expresión regular que valide el string

array

- minItems
- maxItems
- uniqueItems
  - no se pueden repetir elementos si vale true
- Items
  - para definir las características de todos los ítems
- prefixItems
  - cuando importa el orden
- contains
  - para que al menos un elemento sea del tipo especificado
- mincontains / maxcontains
  - usado junto con contains

Referenciar otro json

```
{ (…)
  "properties": {
    (…)
    "warehouseLocation": {
      "description": "Coordinates of the warehouse where
the product is located.",
      "$ref": "https://example.com/geographical-
location.schema.json"
    }
```

```
  },
  "required": [ "productId", "productName", "price" ]
}
```

Podemos definir subesquemas con la etiqueta $defs y referencias con $ref

```
{ (…)
  "properties": {
    "first_name": { "$ref": "#/$defs/name" },
    "last_name": { "$ref": "#/$defs/name" }
  },
  "required": ["first_name", "last_name"],
  "$defs": {
    "name": { "type": "string" }
  }
}
```

Hay muchas más etiquetas

- deprecated
- readOnly
- writeOnly
- $comment
- default

JSON Schema - Ventajas

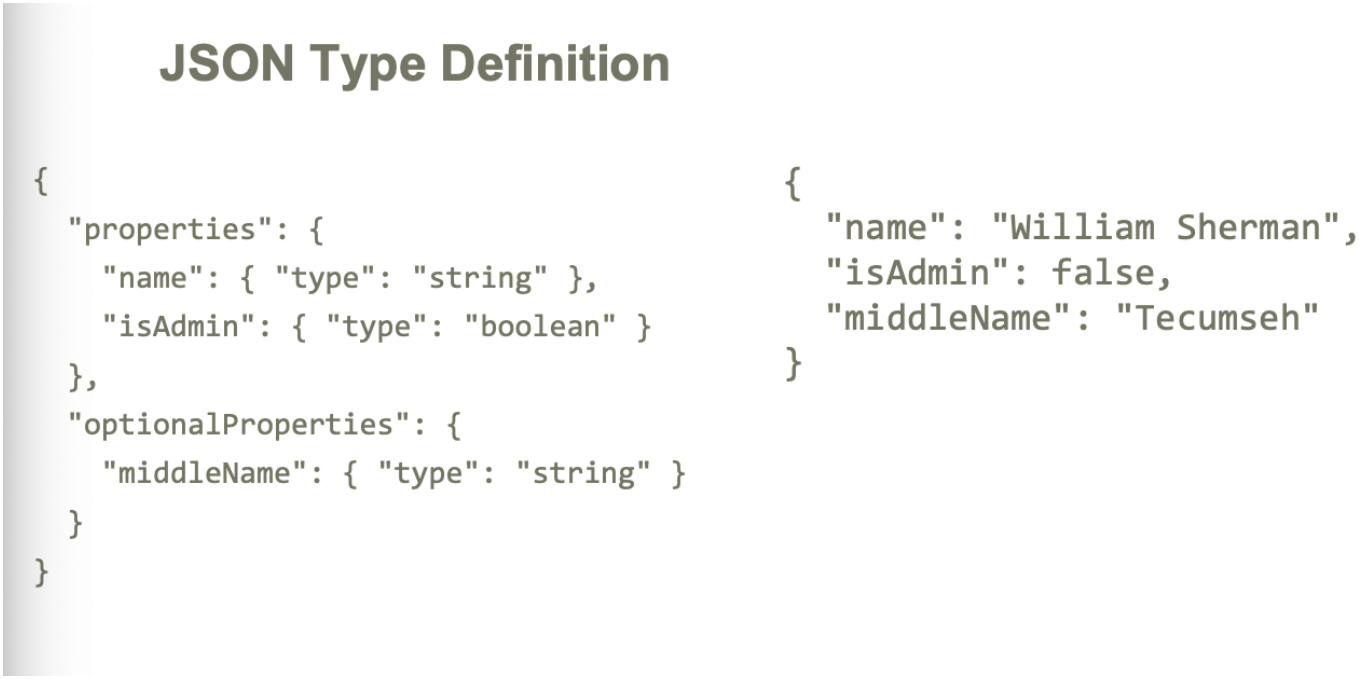
- Wide specification adoption
- Used as part of OpenAPI specification
- Can be effectively used for validation of any JavaScript objects and configuration files

JSON Schema - Desventajas

- Defines the collection of restrictions on the data, rather than the shape of the data
- No standard support for tagged unions
- Complex and error prone for the new users (Ajv has strict mode enabled by default to compensate for it, but it is not cross-platform)
- Internet draft status (rather than RFC)

JSON Type Definition

- JTD
- Describir la forma de los datos
- Schema ligero
- Estándar más reciente que JSON Schema



JSON Type Definition - Ventajas

- Aligned with type systems of many languages - can be used to generate type definitions and efficient parsers and serializers to/from these types
- Very simple, enforcing the best practices for cross-platform JSON API modelling
- Simple to implement, ensuring consistency across implementations
- Defines the shape of JSON data via strictly defined schema forms (rather than the collection of restrictions)
- Effective support for tagged unions

JSON Type Definition - Desventajas

- Limited, compared with JSON Schema - no support for untagged unions*, conditionals, references between different schema files**, etc.

Node.js

- Hay varias librerías para validar JSON
  - ajv
    - Fastest JSON validator
    - Soporta la última versión de JSON Schema (2020-12)
    - Contribuyen y patrocinan empresas como Mozilla y Microsoft
  - jsonschema
  - djv