



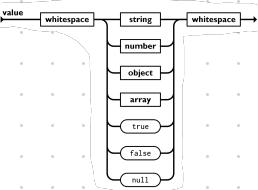
T2: JSON

JSON

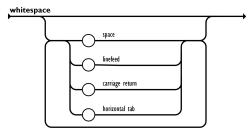
- Formato ligero para el ex^{ce} de datos
- Basado en el estándar ECMA - 262 de 1999.
- Soporta: obj arrays (no permite anidamiento) valores

Sintaxis

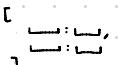
Valor
true/false/null
obj/num/string



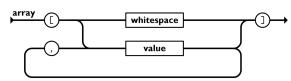
Espacios en blanco:



Tiene q haber un único elem. raíz q tiene q ser un valor:



Array Colección ordenada de valores rodeado por []



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

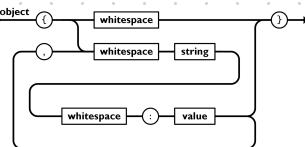
Objeto:

- set desordenado de pares clave/valor
- rodeado de { }
- la clave: "String", no puede estar repetida dentro del mismo obj, lower camel case recomendable
- toda clave/valor + separadora ,
- puede estar vacío + ;

```

{
  "nombreCompleto": "Juan Pérez Rodríguez",
  "edad": 27,
}
  
```

Cuidado con la trailing comma



Anidar elementos

```

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

No se garantiza el orden del contenido de los obj

Los elem. de un array q no están ordenados

JSON CON JS

- convertir un Obj JSON a texto

```
let text = JSON.stringify(obj);
```

- convertir un texto m^{ás} JSON

```
let obj = JSON.parse(text);
```

- Retornar un obj JSON

```
for(let key in jsonobj) {
  console.log("key:" + key + ", value:" + jsonobj[key]);
}
```

JSON SCHEMA

no es la teoría

vocabulario para anotar y validar doc. JSON

Estructura básica:

```

{
  "$schema": "https://json-schema.org/draft/2020-12/schema", // version JSON SCHEMA
  "$id": "https://example.com/product.schema.json", // URL base del esquema
  "title": "Product", // título
  "description": "A product in the catalog" // descripción
}
  
```

Asignaciones anidadas sin implicaciones en la validación

"type": "object" → valor elem. q no es un objeto

"properties": (...) → q no son keywords

"required": (...) → Array de requireds q son obligatorios

Para cada keyword q type

Ejemplo:

```

{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/product.schema.json",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {
    "productId": {
      "type": "integer"
    }
  },
  "required": ["productId"]
}
  
```

Type

- "string"
- "number": tanto n^{umeros} enteros como con decimales
- "integer": pueden ser decimales que terminen en 0: 1.0
- "object"
- "array"
- "boolean": true o false, sin comillas
- "null": null

q puede ser un array q contiene varios tipos

```
"type": ["number", "string"]
```

Ventajas

- Gran adoraci^{on} de esta especificaci^{on}
- Usado en parte de OpenAPI (swagger)
- F堤mete variables (variables, constantes, expresiones)
- Lógica boolean y unibit sin etiquetas
- schemas anidados y dependencias
- Restricciones en el rango de N^º y tamaño
- Validaci^{on} binaria con patrones
- Distribuci^{on} de funcionalidades de registro estrictas entre N schemas
- Se puede usar para la validaci^{on} de obj. JavaScript y exclusivos (array objects)

Inconvenientes

- Define las restricciones sobre los datos, en lugar de la forma de los datos
- No soporta extender para uniones obligatorias
- Complejo + Propenso a errores + Niveles jerárquicos
 - Aj+ tiene el modo establecido por defecto. Una implementaci^{on} Poco NO es MultiPlataforma
- Algunas partes de la especificaci^{on} son difíciles de implementar, lo q crea el riesgo de divergencias en la implementaci^{on}:
- Modelo de resolución
- Propiedades no evaluadas / ejem. no evaluadas
- Referencias recursivas dinámicas
- Internet Draft Status (en lugar de RFC)

JSON Type Definition JTD

Describir la forma de los datos | schema ligero | standar más reciente q JSON Schema

```

{
  "properties": {
    "name": { "type": "string" },
    "isAdmin": { "type": "boolean" }
  },
  "optionalProperties": {
    "middleName": { "type": "string" }
  }
}
  
```

define forma de los datos de JSON definiendo explicitamente los campos del schema

Inconvenientes

- Llamado, comprendido por JSON Schema, q no
- No hay meta-schema en la definición

Ventajas

- Definido para q los errores sean únicos
- soporte efectivo para variables + expresiones
- Simple de implementar, asegurando consistencia
- Integrados en sistemas de tipos de lenguajes
- API basado como API Swagger

T2: JSON

Node.js

Hay varias bibliotecas para validar JSON
ajv ponschema
ajv

Ajv

- JSON Validator
- Soporta la ult. versión de JSON schema

Demo

```
const Ajv = require('ajv');
const ajv = new Ajv(); // options can be passed, e.g. {allErrors: true}
const schema = {
  type: "object",
  properties: {
    foo: {type: "integer"},
    bar: {type: "string"}
  },
  required: ["foo"],
  additionalProperties: false
};

const validate = ajv.compile(schema);
const data = { foo: 1, bar: "abc" };
const valid = validate(data);
if (!valid) console.log(validate.errors);
```

scheme
validate

npm install ajv

```
//Archivo schemas/index.js
const Ajv2020 = require('ajv/dist/2020');
const ajv = new Ajv2020();

const schema_person = require("./person.schema.json")
const schema_coordinate = require("./coordinate.schema.json")

ajv.addSchema(schema_person, "person")
ajv.addSchema(schema_coordinate, "coordinate")

module.exports = ajv;
```

//Archivo schemas/coordinate.schema.json

```
{
  "$id": "https://example.com/geographical/location/schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Longitude and Latitude Values",
  "description": "A geographical coordinate.",
  "required": ["latitude", "longitude"],
  "type": "object",
  "properties": {
    "latitude": {
      "type": "number",
      "minimum": -90,
      "maximum": 90
    },
    "longitude": {
      "type": "number",
      "minimum": -180,
      "maximum": 180
    }
  }
}
```

```
//Archivo index.js
const ajv = require("./schemas");
const person_json_ok = { "firstname": "John", "lastname": "Doe", "age": 21 };
const person_json_err = { "firstname": "John", "lastname": 2, "age": -18 };
const validatePerson = ajv.get("person");
validatePerson(person_json_err);
validatePerson(person_json_ok);

function validatePerson(json) {
  const validate = ajv.getSchema("person");
  if (validate(json)) {
    console.log("JSON OK");
  } else {
    console.log("JSON NOT OK");
    console.log(validate.errors);
  }
}
```

//Archivo schemas/person.schema.json

```
{
  "$id": "person.schema.json",
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "title": "Person Schema",
  "type": "object",
  "properties": {
    "firstname": {
      "type": "string",
      "description": "The person's first name."
    },
    "lastname": {
      "type": "string",
      "description": "The person's last name."
    },
    "age": {
      "description": "Age in years which must be equal to or greater than zero.",
      "type": "integer",
      "minimum": 0
    }
  }
}
```

index.js → Validarlos con Ajv
schema
↳ person.schema.json ↳ 2 JSON schema
↳ coordinate.schema.json
↳ index.js → Gargar y compilar ↑