NoSQL

JSON

JavaScript Object Notation

Enrique Barra Arias
Álvaro Alonso González

Introducción – El problema

- ¿Cómo representamos datos?

- Queremos:
  - Enviar
  - Recibir
  - Almacenar
  - Procesar
  - Cargar
  - Describir
  - Estructurar

- Se han desarrollado muchas soluciones ad-hoc donde se separan un conjunto de valores separados por comas, puntos y comas u otros separadores pero de serialización y deserialización complicadas

- Hay que evitar tener que construir parsers (parsear es analizar, diseccionar) cada vez que queremos intercambiar mensajes con el servidor

Introducción – JSON

JSON (**J**ava**S**cript **O**bject **N**otation)

- **Formato ligero** de representación, almacenamiento e intercambio de datos independiente de cualquier lenguaje de programación
- Tiene forma de texto plano, de simple de lectura, escritura y generación
- Se basa en la construcción de una lista ordenada de valores (array literal) o bien una colección de pares nombre/valor (objeto literal)

- Ejemplo objeto literal pares variable/valor:

```
{
    "country": "New Zealand",
    "population": 3993817,
    "animals": true
 }
```

Ejemplo Array literal:

```
["Enrique", 35, true]
```

Introducción – JSON

- Con JSON se pueden declarar dos tipos de estructuras de datos:
  - Objetos
  - Arrays

- Un objeto es un conjunto **no ordenado** de pares de clave/**valor**

```
{
    "country": "New Zealand",
    "population": 3993817,
    "animals": true
}
```

- Un array es un conjunto **ordenado** de **valores**

```
["Enrique", 35, true]
```

Nota – ordenación de datos

- En un objeto no importa el orden
- { a: 5, b: 6, c: 89 } es igual que {b: 6, c: 89, a:5 }
- {nombre: “Enrique”, edad: 40} es igual que {edad: 40, nombre: “Enrique”}

- En un array el orden es imprescindible
- [5,6,89] es distinto de [89,5,6]
- [“coche”, “casa”] es distinto de [“casa”, “coche”]
- [{x: 7, y: 8}, {x: 19, y: 21}] es distinto de [{x: 19, y: 21}, {x: 7, y: 8}]

Introducción – json

- JSON:
  - Independiente de un lenguaje específico (interoperable)
  - Basado en texto
  - De Formato ligero
  - Fácil de parsear
  - NO define funciones
  - NO tiene espacios de nombres (Namespaces)
  - NO es extensible
  - No soporta comentarios (muy criticado)

- Su tipo MIME es -> **application/json**
- Existe **BSON** -> Binay JSON

Clave

- Las claves son cadenas de caracteres (strings) y por lo tanto tienen que ir entre comillas
- Esta es la única diferencia entre un JSON y un objeto JavaScript

JSON

```
{
    "country": "New Zealand",
    "population": 3993817,
    "animals": true
}
```

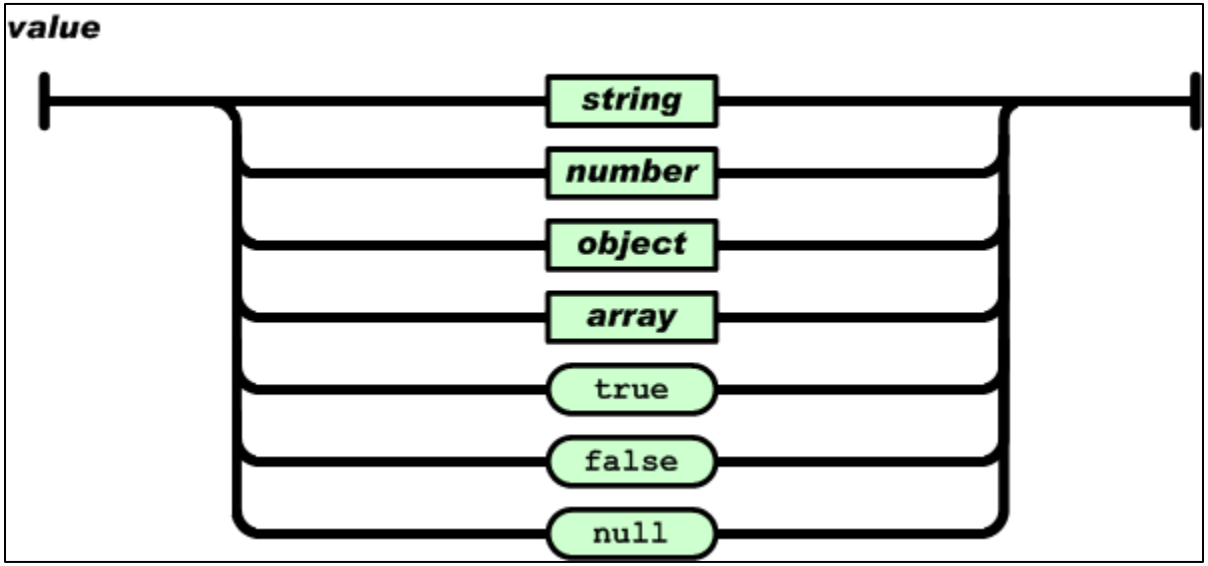
JS OBJECT

```
{
    country: "New Zealand",
    population: 3993817,
    animals: true
}
```

- A veces a las claves se les llama "nombres" y se dice que JSON son un conjunto de nombre/valor
- En MongoDB se puede poner con o sin comillas

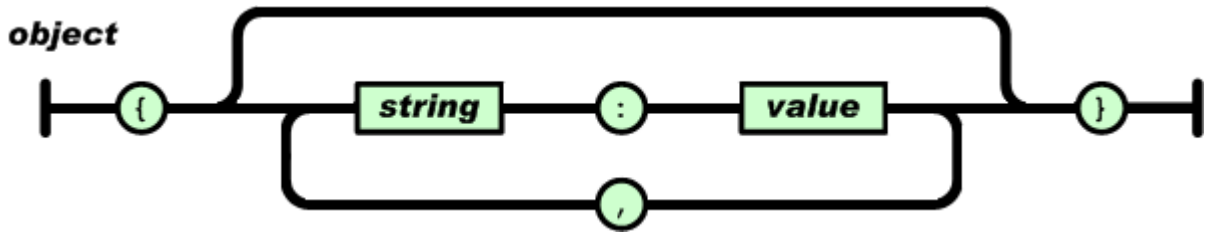
Valor

- El valor debe ser un tipo de datos válido en JSON
- Estos son:
  - **string**.- Colección de cero o más caracteres unicode
  - **number**.- Valor numérico **sin comillas**
  - **object**.- Conjunto desordenado de pares nombre/valor
  - **array**.- Colección ordenada de valores
  - **boolean**.- valor true o false
  - **null**.- nulo



Forma de object

- Es un conjunto de propiedades , cada una con su valor
- Notación
  - Empieza con una llave de apertura {
  - Terminan con una llave de cierre }
  - Sus propiedades
    - Se separan con comas
    - El nombre y el valor están separados por dos puntos :

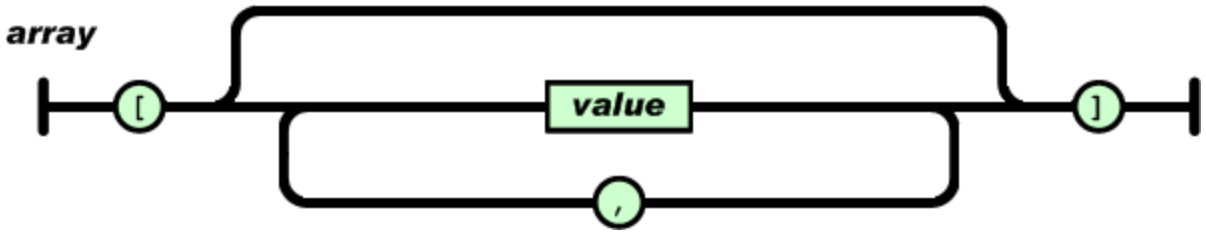


Forma de object

```
{
    “country”: “New Zealand”,
    “population”: 3993817,
   “area”: 268021,
    “monarchy”: true,
    “animals”: [“sheep”, “kiwi”]
 }
```

Forma de array

- Colección ordenada de valores
- Notación
  - Empieza con un corchete izquierdo [
  - Termina con un corchete derecho]
  - Los valores se separan con una coma ,

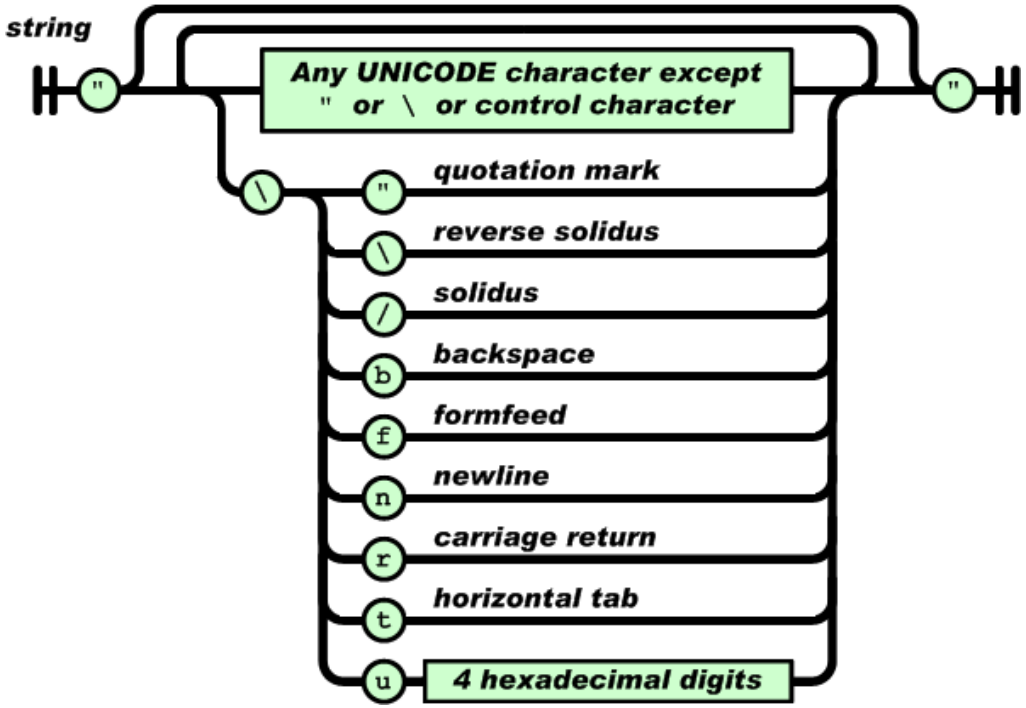


Forma de array

```
[
  {
    "country": "New Zealand",
    "population": 3993817,
    "animals": ["sheep", "kiwi"]
  },
  {
    "country": "Singapore",
    "population": 4353893,
    "animals": ["merlion", "tiger"]
  }
]
```

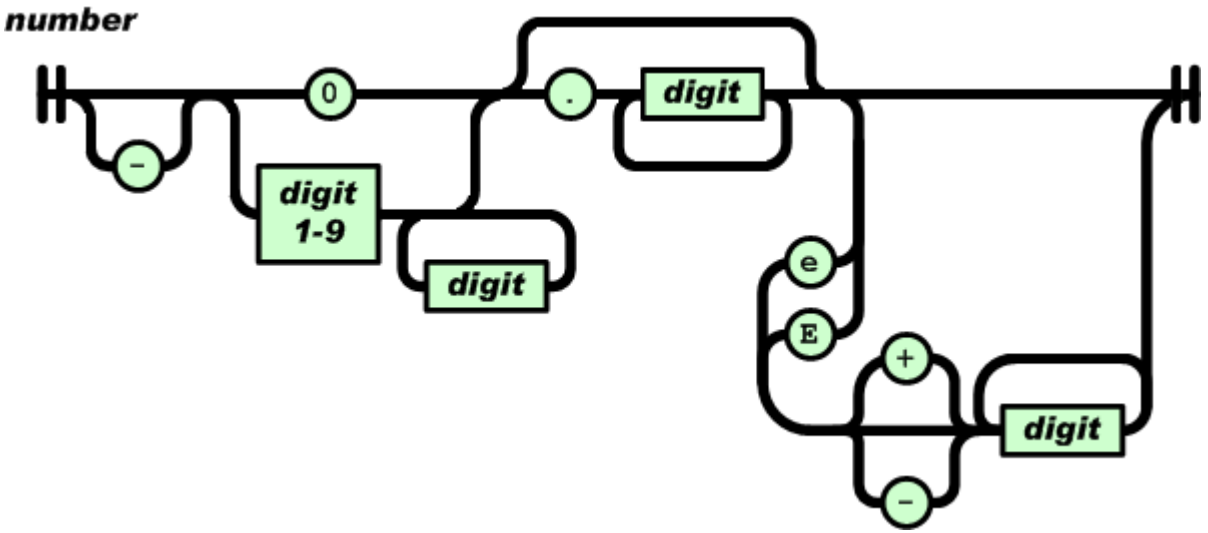
Forma de string

- Colección de cero a más caracteres Unicode encerrados entre comillas dobles
- Los caracteres de escape utilizan la barra invertida
- Es parecida a una cadena de caracteres en C o Java.



Forma de number

- Similar a los numeros de C o Java
- No usa formato octal o hexadecimal
- No puede ser **NaN** o **Infinity**, en su lugar se usa **null**
- Puede representar
  - Integer
  - Real
  - Scientific



Codificación de caracteres

- Por defecto es UTF-8
- UTF-16 y UTF-32 también están permitidos

json vs xml

XML

```xml
<?xml version="1.0"?>
<alumno>
    <numero_ID>1232654</numero_ID>
    <nombre>Adan Perez Gomez</nombre>
    <telefono>99999999</telefono>
    <asignaturas>
        <nombre>BBDD</nombre>
        <nombre>BDNR</nombre>
        <nombre>PROG</nombre>
    </asignaturas>
</alumno>
```

JSON

```json
{
"numero_ID": "1232654",
"nombre": "Adan Perez Gomez",
"telefono": "99999999",
"asignaturas": ["BBDD","BDNR","PROG" ]
}
```

json vs xml (Arrays)

XML

```xml
<?xml version="1.0"?>
<listado>
    <persona>
        <nombre>Juan</nombre>
        <apellidos>Palomo</apellidos>
        <fecha>10/10/1980</fecha>
    </persona>
    <persona>
        <nombre>Pepe</nombre>
        <apellidos>Rodriguez</apellidos>
        <fecha>10/11/1965</fecha>
    </persona>
</listado>
```

JSON

```json
{
"listado": {
    "persona": [
        {
            "nombre": "Juan",
            "apellidos": "Palomo",
            "fecha": "10/10/1980"
        },
        {
            "nombre": "Pepe",
            "apellidos": "Rodriguez",
            "fecha": "10/11/1965"
        }
    ]
}
}
```

json vs xml (similitudes)

- Ambos son legibles por los humanos (son textuales)
- Tienen una sintaxis muy simple
- Son jerárquicos
- Son independientes del lenguaje de programación

json vs xml (diferencias)

- JSON es un formato XML es un metalenguaje (sirve incluso para definir otros lenguajes de etiquetas como HTML)
- Sintaxis diferente

- JSON
  - Es más compacto (ocupa menos que el XML)
  - Puede ser parseado usando el método **eval()** de JavaScript
  - Puede incluir Arrays
  - Los nombres de las propiedades no pueden ser palabras reservadas de JavaScript

- XML
  - Los nombres son más extensos
  - No tiene tipos de datos (usaremos esquemas adicionales para definir el tipo de datos)
  - Mucho más potente y versátil
    - Datos más complejos, atributos en las etiquetas

YAML

- Acrónimo inicialmente de: **Y**et **A**nother **M**arkup **L**anguage
- Cambiado a: **Y**AML **A**in't **M**arkup **L**anguage

- Es un superset de JSON, con más capacidades
  - Listas, casting, etc
  - No maneja caracteres Unicode de escape
  - JSON puede ser parseado por los parsers de YAML

- Hay que tenerlo en cuenta cuando JSON no sea suficiente para nuestras necesidades

YAML ejemplo

```
# PostgreSQL setup (default)
development:
  adapter: postgresql
  encoding: unicode
  host: localhost
  username: vish_dev
  password: yourpassword
  database: vish_development
  pool: 5

test:
  adapter: postgresql
  encoding: unicode
  host: localhost
  username: vish_dev
  password: yourpassword
  database: vish_test
  pool: 5
```

CSV

- CSV: Comma-Separated alues
- Formato muy simple que usa una coma para separar valores
- A veces otro separador (; : . tab space …) ya que no está completamente estandarizado (aunque hay una RFC)
- Cada fila es un registro
- Tiene normalmente una cabecera con el nombre de los campos

CSV ejemplo


```
firstname,lastname,password,email
Josep Lluis,Serrano Perez,mypass1,ejemplo@gmail.com
Enrique,Barra Arias,mypass2, ejemplo2@gmail.com
```

NoSQL

JSON - USO

Enrique Barra Arias
Álvaro Alonso González

Json - Utilización

- **Serialización**: Transformación de objetos a cadenas de texto
- **Deserialización**: Transformación de cadenas de texto a objetos

- Directamente con JavaScript
- Mediante Librerías (en cualquier lenguaje)
- Frameworks de cliente
- NoSQL
- package.json

JSON – JavaScript

- Define los siguientes métodos
  - JSON.parse
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse
  - JSON.stringify
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/stringify

ejemplos JSON.parse()

```
JSON.parse('{}');              // {}
JSON.parse('true');            // true
JSON.parse('"foo"');           // "foo"
JSON.parse('[1, 5, "false"]'); // [1, 5, "false"]
JSON.parse('null');            // null
```

```
var obj = JSON.parse('{ "employees" : [{ "firstName":"John"
, "lastName":"Doe" },{ "firstName":"Anna" ,
"lastName":"Smith" },{ "firstName":"Peter" ,
"lastName":"Jones" } ]}');
```

ejemplos JSON.stringify()

```
JSON.stringify({});                    // '{}'
JSON.stringify(true);                  // 'true'
JSON.stringify('foo');                 // '"foo"'
JSON.stringify([1, 'false', false]); // '[1,"false",false]'
JSON.stringify({ x: 5 });              // '{"x":5}’

JSON.stringify(new Date(2006, 0, 2, 15, 4, 5))
// '"2006-01-02T15:04:05.000Z"'

JSON.stringify({ x: 5, y: 6 });
// '{"x":5,"y":6}' o '{"y":6,"x":5}'
```

Un par de strings para hacer pruebas con las dev-tools de Chrome:

```
'{ "employees" :  [ { "firstName":"John" , "lastName":"Doe" }, { "firstName":"Anna" , "lastName":"Smith" }, { "firstName":"Peter" , "lastName":"Jones" } ] }’
```

'{ "country": " New Zealand ", "population": 3993817, "area": 268021, "monarchy": true, "animals": [" sheep ", " kiwi "] }'

FRAMEWORKS o LIBRERÍAS WEB

- Actualmente existen frameworks y librerías que utilizan de forma nativa JSON para presentar y tratar la información proveniente del servidor.

  - **YUI (Yahoo User Interface)**
  - Dojo
  - **jQuery**
  - **Extjs**
  - Otros toolkits Ajax.

- Puede recuperar datos en formato JSON

- API
  - jQuery.parseJSON(json)
    - http://api.jquery.com/jquery.parsejson/
  - jQuery.getJSON(url, [data], [callback(data, textStatus, xhr)])
    - http://api.jquery.com/jquery.getjson/

NoSQL

- Este término se refiere a bases de datos “no relacionales” que no dan garantías ACID, su característica que más llama la atención es que no existen esquemas de tablas predefinidos.

- Algunas de las bases de datos que exponen sus datos mediante JSON/BSON son:
  - CouchDB
  - **MongoDB**
  - Elasticsearch
  - ArangoDB
  - RethinkDB
  - Couchbase
  - RavenDB
  - Riak

package.json

- Fichero de configuración de un proyecto nodejs. Usado por el "node package manager" o npm
- Es formato JSON, e indica todo lo que hace falta saber para un proyecto, nombre, versión, descripción, dependencias, repositorios, bugs, etc
- Info completa: https://docs.npmjs.com/files/package.json
- Ejemplo:

```
{ "name": "ethopia-waza",
  "description": "a delightfully fruity coffee varietal",
  "version": "1.2.3",
  "devDependencies": {
      "coffee-script": "~1.6.3"
  },
  "scripts": {
      "prepublish": "coffee -o lib/ -c src/waza.coffee"
  },
  "main": "lib/waza.js"
}
```

AutoEvaluación

Tests JSON

- ¿Cuáles de las siguientes expresiones son documentos JSON válidos en MongoDB?
  - Recuerde, MongoDB no requiere comillas alrededor de las claves, ya que siempre deben ser cadenas.

A. { a : 1, b : 2, c : 3 }
B. { a,1; b, 4, c, 6}
C. { a : 1; b : 1; c : 4 }
D. (A, 1; b : 2; c, 4 }

Tests JSON

- ¿Cuáles de las siguientes expresiones son documentos JSON válidos en MongoDB?
  - Recuerde, MongoDB no requiere comillas alrededor de las claves, ya que siempre deben ser cadenas.

A. { a : 1, b : 2, c : 3 }
B. { a,1; b, 4, c, 6}
C. { a : 1; b : 1; c : 4 }
D. (A, 1; b : 2; c, 4 }

Tests JSON

- ¿Cuáles de las siguientes expresiones son documentos JSON válidos en MongoDB?
  - Recuerde, MongoDB no requiere comillas alrededor de las claves, ya que siempre deben ser cadenas.

1. { "days" : { "SUN" , "MON" , "TUE" , "WED" , "THU" , "FRI" , "SAT" } }
2. { "days" = ["SUN" , "MON" , "TUE" , "WED" , "THU" , "FRI" , "SAT"] }
3. { "days" = { "SUN" , "MON" , "TUE" , "WED" , "THU" , "FRI" , "SAT" } }
4. { "days" : ["SUN" , "MON" , "TUE" , "WED" , "THU" , "FRI" , "SAT"] }

Tests JSON

- ¿Cuáles de las siguientes expresiones son documentos JSON válidos en MongoDB?
  - Recuerde, MongoDB no requiere comillas alrededor de las claves, ya que siempre deben ser cadenas.

1. { "days" : { "SUN" , "MON" , "TUE" , "WED" , "THU" , "FRI" , "SAT" } }
2. { "days" = ["SUN" , "MON" , "TUE" , "WED" , "THU" , "FRI" , "SAT"] }
3. { "days" = { "SUN" , "MON" , "TUE" , "WED" , "THU" , "FRI" , "SAT" } }
4. { "days" : ["SUN" , "MON" , "TUE" , "WED" , "THU" , "FRI" , "SAT"] }

Tests JSON

- ¿Cuáles de las siguientes expresiones son documentos JSON válidos en MongoDB?
  - Recuerde, MongoDB no requiere comillas alrededor de las claves, ya que siempre deben ser cadenas.

A. { a : 1, b : 2, c : 3 }
B. { a : 1, b : 2, c : [1, 2, 3, 4, 5] }
C. { a : 1, b : { }, c : [{ a : 1, b : 2 }, 5, 6] }
D. { }

Tests JSON

- ¿Cuáles de las siguientes expresiones son documentos JSON válidos en MongoDB?
  - Recuerde, MongoDB no requiere comillas alrededor de las claves, ya que siempre deben ser cadenas.

A. { a : 1, b : 2, c : 3 }
B. { a : 1, b : 2, c : [1, 2, 3, 4, 5] }
C. { a : 1, b : { }, c : [{ a : 1, b : 2 }, 5, 6] }
D. { }

- Más preguntas tipo test sobre JSON:
- https://www.includehelp.com/mcq/json-multiple-choice-questions-mcqs.aspx

JSON Ejercicios

- Crea un documento JSON que represente el menú e información de un restaurante

- Incluir los siguientes campos:
- Información del restaurante
  - Nombre, si tiene opciones vegetarianas o gluten free, si tiene delivery, cuanto suele tardar, tarifa de envío, dirección, web, …
  - Horas de apertura cada dia
- Lista de categorías de comidas (tipo array de objetos, cada objeto debe incluir: nombre de la categoría y una lista de platos de la categoría)
- Lista de bebidas (tipo array de objetos, cada objeto debe incluir: nombre de la bebida, descripción y precio)

JSON Ejercicios

- Ver solución posible en:
- https://github.com/BBDD-ETSIT/JSON_SCHEMA_AND_VALIDATORS/blob/main/json_examples/restaurant.json

JSON Ejercicios

- Escribir un documento JSON con un CV de una persona

```
{
    "personal_information": {
      "full_name": "John Doe",
      "email": "johndoe@email.com",
      "phone_number": "+1 123 456 7890",
      "location": "New York, NY, USA",
      "linkedin_profile": "https://linkedin.com/in/johndoe",
      "github_profile": "https://github.com/johndoe"
    },
    "professional_summary": "A highly motivated software engineer with 5 years of experience in developing web a
    "education": [
      {
        "degree": "Bachelor of Science in Computer Science",
        "institution": "University of ABC",
        "start_date": "2015-09-01",
        "end_date": "2019-06-01",
        "gpa": "3.8/4.0"
      }
    ],
    "work_experience": [
      {
        "position": "Software Engineer",
        "company": "XYZ Inc.",
        "start_date": "2019-06-01",
        "end_date": "2022-12-01",
        "responsibilities": [
          "Designed and developed highly scalable web applications using Java, Python, and JavaScript.",
          "Collaborated with cross-functional teams to ensure successful project delivery.",
          "Implemented agile methodologies to improve team productivity and efficiency."
        ]
      }
    ],
    "skills": [
      "Java",
      "Python",
      "JavaScript",
      "Agile Development",
      "Web Development"
    ]
  }
```


- https://github.com/BBDD-ETSIT/JSON_SCHEMA_AND_VALIDATORS/blob/main/json_examples/cv.json

JSON Ejercicios

- Crea un documento JSON que represente el registro médico de un paciente.

- Incluir los siguientes campos:
  - Nombre completo del paciente (tipo string)
  - Edad del paciente (tipo número)
  - Género del paciente (tipo string)
  - Número de identificación del paciente (tipo string)
  - Historial médico del paciente (tipo array de objetos, cada objeto debe incluir: fecha de la visita, motivo de la visita y notas del médico)
  - Lista de medicamentos actuales (tipo array de strings)
  - Alergias (tipo array de strings)

```json
{
    "full_name": "John Smith",
    "age": 40,
    "gender": "Male",
    "patient_id": "A123456789",
    "medical_history": [
      {
        "date": "2022-01-01",
        "reason": "Annual Checkup",
        "notes": "Patient is in good health. No concerns noted."
      },
      {
        "date": "2022-06-15",
        "reason": "Sinus Infection",
        "notes": "Prescribed antibiotics for sinus infection. Follow up in 1 week."
      }
    ],
    "current_medications": [
      "Ibuprofen",
      "Lisinopril",
      "Albuterol"
    ],
    "allergies": [
      "Penicillin",
      "Bee Stings"
    ]
  }
```

- https://github.com/BBDD-ETSIT/JSON_SCHEMA_AND_VALIDATORS/blob/main/json_examples/patient.json

Conclusiones

Conclusiones

- Formato de intercambio de datos, potente, flexible y **sobre todo ligero** para intercambiar datos (por ejemplo, vía HTTP) o almacenarlos.
- Independiente de cualquier lenguaje de programación.
- Es soportado por los principales lenguajes del lado servidor
  - Java, .Net, PHP, … (pueden serializar y deserializar objetos en formato JSON)
- Ideal para construir aplicaciones con frameworks JavaScript
  - Ej.: jQuery
- Existen diferentes bases de datos **NoSQL** que guardan sus datos en formato JSON plano o binario (BSON)

NoSQL

JSON Schema

Enrique Barra Arias
Álvaro Alonso González

JSON Schema

- Al igual que ocurre en XML, es posible describir el formato de datos JSON a través de los JSON Schema
- "JSON Schema es un vocabulario que permite **anotar** y **validar** documentos JSON."

- Ventajas:
  - describe los formatos de datos que se utilicen
  - crea documentación fácil de leer y entender tanto por humanos como por máquinas
  - aporta validación estructural completa, útil para crear pruebas automatizadas y validar datos enviados por el cliente

- La especificación del esquema se puede encontrar en:
  - https://json-schema.org/

JSON Schema: Ejemplo

- Dado este JSON:

```
{
  "productId": 1,
  "productName": "A green door",
  "price": 12.50,
  "tags": [ "home", "green" ]
}
```

- Pueden surgir muchas preguntas:
  - ¿Qué es productId?
  - ¿Es obligatorio el nombre del producto?
  - ¿El precio puede ser cero (0)?
  - ¿Son todas las etiquetas/tags strings?

Recursos para JSON Schema

- Seguimos el tutorial:
  - https://json-schema.org/learn/getting-started-step-by-step
- Libro referencia:
  - https://json-schema.org/understanding-json-schema/
- Mas recursos:
  - https://json-schema.org/learn/
- Especificación oficial:
  - https://json-schema.org/specification.html

Glosario

- JSON
- JSON Schema
- JSON hyper-schema
  - amplía JSON Schema y ofrece un vocabulario para anotar documentos JSON con controles hipermedia Especialmente la keyword “link”
- JSON Pointer
  - una sintaxis string para identificar un valor en una ubicación específica dentro de un documento JSON
  - https://opis.io/json-schema/2.x/pointers.html
- Draft
  - Una versión de la especificación del JSON Schema
- **Keyword**
  - Cualquier propiedad que aparece dentro de un JSON Schema
- Tool o herramienta o implementación
  - es cualquier aplicación de software o biblioteca para trabajar con esquemas o evaluarlos de alguna manera
  - https://json-schema.org/implementations
- Vocabulario
  - Colección de keywords que sirven para un uso concreto
  - https://github.com/json-schema-org/json-schema-vocabularies
- Ver todas en:
  - https://json-schema.org/learn/glossary

keywords

- Especiales
  - $id
  - $schema
  - $comment
- Annotation keywords
- Validation keywords

- Muchas más:
  - https://json-schema.org/understanding-json-schema/keywords

Un primer Schema

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

- La “keyword” **$schema** indica que este esquema está escrito de acuerdo con una versión (draft) específica del estándar y se utiliza por diversas razones, principalmente el control de versiones.
- La “keyword” **$id** define un URI para el esquema, y el URI base con el que se resuelven otras referencias URI dentro del esquema.
- Las “Anotations keywords” **title** y **description** son sólo descriptivas. No añaden restricciones a los datos que se validan. La intención del esquema se establece con estas dos palabras clave.
- La “Validation keyword” **type** define la primera restricción en nuestros datos JSON y en este caso tiene que ser un objeto JSON.

Validando ProductId

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

- “Validation keywords”: properties y required

Types - tipos

- http://json-schema.org/understanding-json-schema/reference/type.html
- Solo hay los siguientes tipos básicos:
  - string
  - number
  - integer
  - object
  - array
  - boolean
  - null
- Especialmente interesante string y sus formatos
  - http://json-schema.org/understanding-json-schema/reference/string.html#string
  - Así validaré una fecha:

```
"fecha": {
   "description": "Fecha del catálogo",
   "type": "string",
   "format": "date"
 },
```

Otras Validation Keywords

- Ver todas en:
  - https://json-schema.org/draft/2020-12/json-schema-validation.html#rfc.section.6.1.1
  - https://json-schema.org/understanding-json-schema/keywords
- exclusiveMinimum
- minItems
- uniqueItems
- …

Esquema final

```json
{
  "productId": 1,
  "productName": "A green door",
  "price": 12.50,
  "tags": [ "home", "green" ]
}
```

```json
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/product.schema.json",
  "title": "Product",
  "description": "A product from Acme's catalog",
  "type": "object",
  "properties": {
    "productId": {
      "description": "The unique identifier for a product",
      "type": "integer"
    },
    "productName": {
      "description": "Name of the product",
      "type": "string"
    },
    "price": {
      "description": "The price of the product",
      "type": "number",
      "exclusiveMinimum": 0
    },
    "tags": {
      "description": "Tags for the product",
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    }
  },
  "required": [ "productId", "productName", "price" ]
}
```

NoSQL

JSON Schema

Validando un JSON

Enrique Barra Arias
Álvaro Alonso González

JSON SCHEMA VALIDATOR ONLINE

- https://www.jsonschemavalidator.net/

Select schema: Custom

```
    },
    "tags": {
      "description": "Tags for the product",
      "type": "array",
      "items": {
        "type": "string"
      },
      "minItems": 1,
      "uniqueItems": true
    },
    "dimensions": {
      "type": "object",
      "properties": {
        "length": {
          "type": "number"
        },
        "width": {
          "type": "number"
        },
        "height": {
          "type": "number"
        }
      },
      "required": [ "length", "width", "height" ]
    }
  },
  "required": [ "productId", "productName", "price" ]
}
```

Input JSON:

```
{
  "productId": 1,
  "productName": "An ice sculpture",
  "price": 12.50,
  "tags": [ "cold", "ice" ],
  "dimensions": {
    "length": 7.0,
    "width": 12.0,
    "height": 9.5
  },
  "warehouseLocation": {
    "latitude": -78.75,
    "longitude": 20.4
  }
}
```

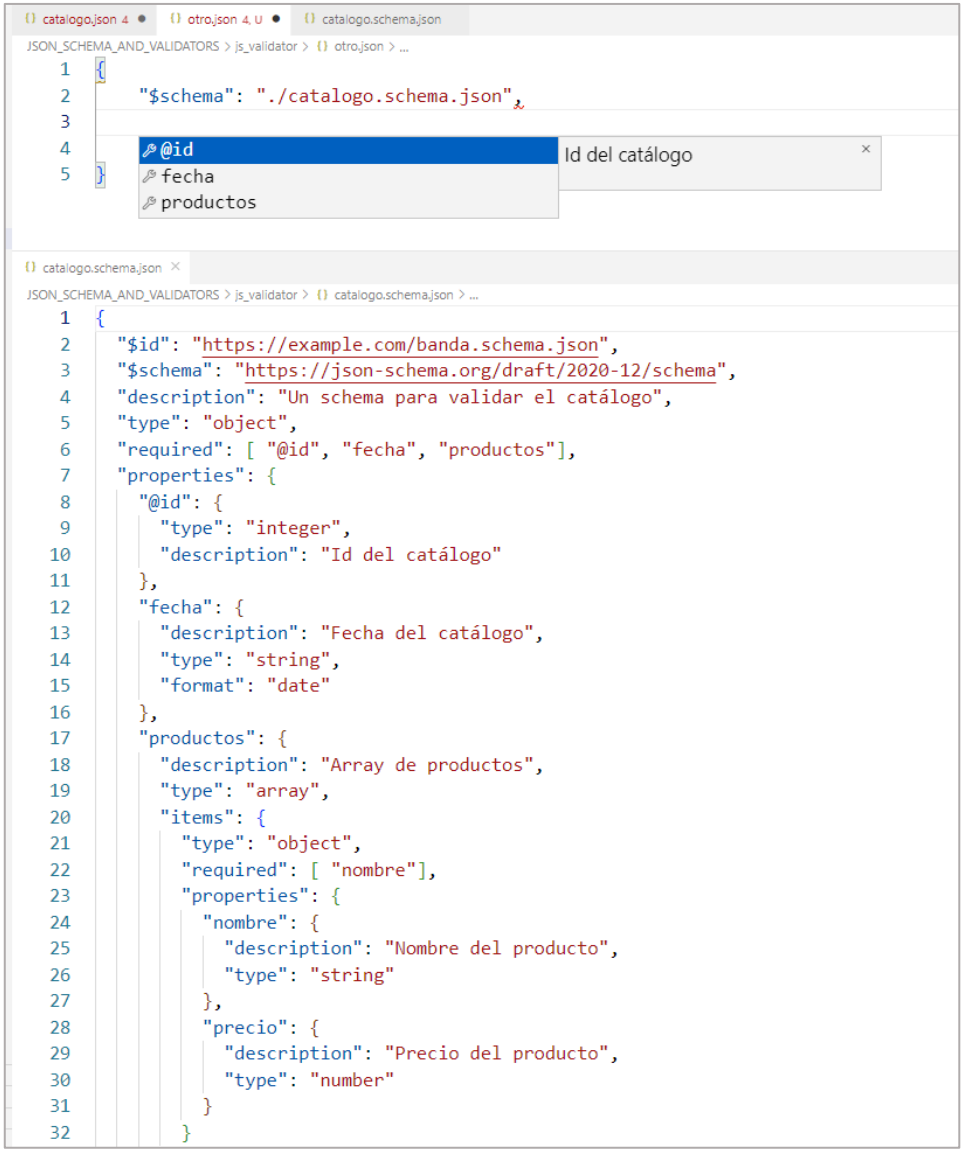
✔ No errors found. JSON validates against the schema

VSCODE

- Lo hace solo:
- https://code.visualstudio.com/docs/languages/json

- La asociación de un archivo JSON a un esquema se puede realizar en el propio archivo JSON utilizando el atributo $schema (método recomendado) o en la configuración de usuario o espacio de trabajo (Preferencias > configuración > archivo) en la propiedad json.schemas.

- No preocuparse si subraya el schema porque no lo incluye vscode. Usad siempre la última version del schema



- Hay muchos:
  - https://json-schema.org/tools?query=&sortBy=name&sortOrder=ascending&groupBy=toolingTypes&licenses=&languages=JavaScript&drafts=&toolingTypes=
- Usaremos este:
  - https://github.com/ajv-validator/ajv
  - Y para soportar los formatos del schema hace falta añadir:
    - https://github.com/ajv-validator/ajv-formats
- Programa completo funcionando:
  - https://github.com/BBDD-ETSIT/JSON_SCHEMA_AND_VALIDATORS/tree/main/js_validator

PYTHON

- Hay muchos:
  - https://json-schema.org/tools?query=&sortBy=name&sortOrder=ascending&groupBy=toolingTypes&licenses=&languages=Python&drafts=&toolingTypes=
- Usaremos este:
  - https://github.com/Julian/jsonschema
- Programa completo funcionando (BDFI, SIBD, BBDD):
  - https://github.com/BBDD-ETSIT/JSON_SCHEMA_AND_VALIDATORS/tree/main/python_validator
- Programa completo funcionando (BDNR):
  - https://github.com/ebarra/BDNR_ejemplos/tree/main/json_json_schema/JSONSCHEMA_python_validator

Cada vez más importante y usado

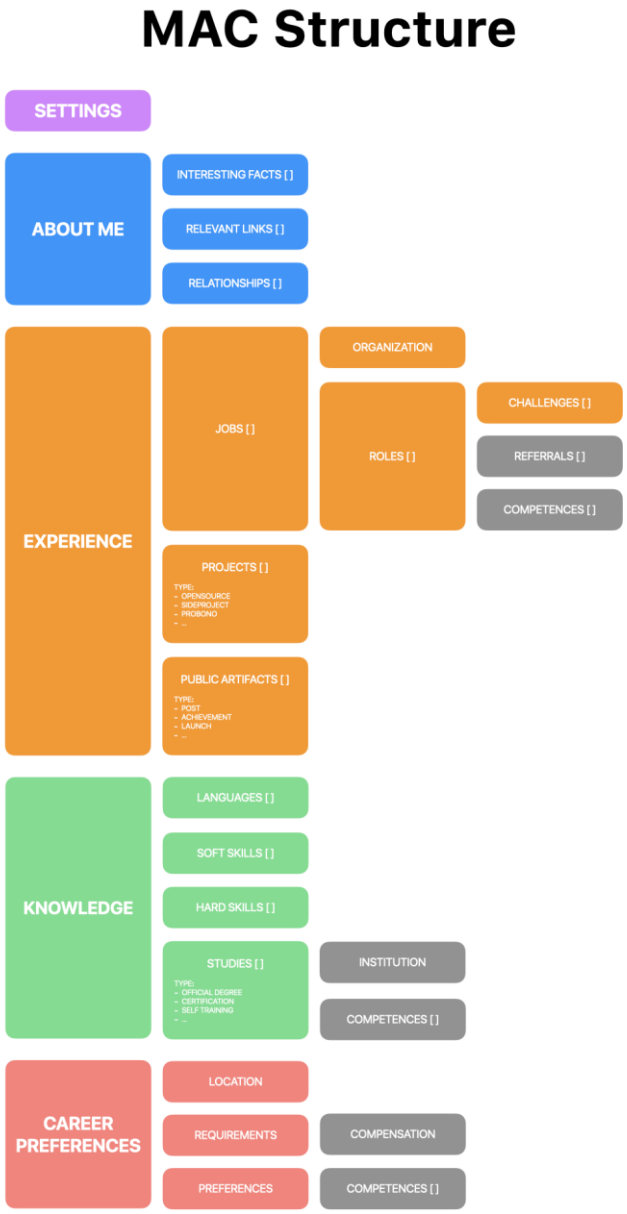
- Para formatear el output de ChatGPT:
  - https://openai.com/index/introducing-structured-outputs-in-the-api/
- Ejemplos en producción
  - GitHub, PostMan, Manfred, …:
  - https://json-schema.org/overview/case-studies
- En MongoDB
  - https://www.mongodb.com/docs/manual/reference/operator/query/jsonSchema/
  - https://www.mongodb.com/resources/languages/json-schema-examples

JSON SCHEMA real

- Un Proyecto completo y real, incluye schema completo, validador, generador de ejemplos, etc:
  - https://github.com/getmanfred/mac
  - https://www.youtube.com/watch?v=KB2DkeQo8d8

- https://github.com/getmanfred/mac/blob/master/schema/schema.json

- https://raw.githubusercontent.com/getmanfred/mac/master/assets/readme/MAC_Structure.png



Referencias y citas

- Esta presentación está basada en los enlaces que han ido apareciendo en las diferentes diapositivas y en los siguientes:
- https://www.slideshare.net/Emmerson_Miranda/json-short-manual-5732305