JSON

JSON (JavaScript Object Notation) es un formato de datos utilizado para representar información estructurada en forma de objetos y arrays en texto plano. Fue diseñado como una alternativa liviana al formato XML y es ampliamente utilizado en aplicaciones web y móviles para intercambiar datos.

En JSON, los datos se representan como pares de clave-valor y se organizan en objetos y arrays. Por ejemplo, un objeto JSON que representa una persona podría verse así:

json Copy code

```json
{
  "nombre": "Juan",
  "apellido": "Pérez",
  "edad": 30,
  "dirección": {
    "calle": "Calle Falsa 123",
    "ciudad": "Buenos Aires"
  },
  "telefonos": [
    {
      "tipo": "casa",
      "numero": "555-1234"
    },
    {
      "tipo": "trabajo",
      "numero": "555-5678"
    }
  ]
}
```

Los objetos y arrays pueden anidarse para representar estructuras más complejas.

JSON se utiliza comúnmente en aplicaciones web para intercambiar datos entre el cliente y el servidor en formato AJAX, así como en servicios web y API RESTful. También es comúnmente utilizado en bases de datos NoSQL y sistemas de mensajería.

En resumen, JSON es un formato de datos popular y versátil que se utiliza para representar información estructurada en aplicaciones web y móviles, bases de datos y sistemas de mensajería.
JSON es un lenguaje usado para el intercambio de datos entre sistemas, está basado en la notación de los literales de objeto de Javascript.
JSON son las siglas de JavaScript Object Notation, o sea, Notación de Objeto Javascript. Básicamente usa la misma notación o forma con la que se escriben los objetos Javascript en el código, los literales de objeto, con algunas restricciones y añadidos extra.

La utilidad de JSON es la de intercambiar datos, por eso se conoce como lenguaje de intercambio de información o lenguaje de transporte. Sirve para la comunicación entre servicios web (web services) y los clientes que los consumen, enviando y recibiendo la información en formato JSON.

Características de JSON
JSON es un lenguaje de modelador de datos
Consiste en pares "clave - valor"
Los valores pueden cadenas, números o boleanos, así como otros objetos JSON, con cualquier nivel de anidación
Es un formato flexible, ligero y fácilmente transferible a través de las redes

Ventajas de JSON
La lectura del código resulta de fácil lectura y la información es suficienteme expresiva para poder ser leída por personas, además de máquinas.
El tamaño de los archivos que se transfieren es ligero.
El código está basado en el lenguaje Javascript, lo que es ideal para las aplicaciones web.
Todos los lenguajes disponen de funciones para interpretar cadenas JSON y convertir datos en cadenas JSON válidas.
Se escribe en archivos de texto plano con codificación UTF8, que es compatible con todos los sistemas.
Estas ventajas son especialmente interesantes en el entorno web, ya que Javascript es el lenguaje de programación para frontend y el código es directamente entendible por cualquier sistema que soporte Javascript. Esta facilidad de procesamiento del JSON y su ligereza para la transmisión lo hacen ideal como formato de intercambio de información y por eso es ampliamente usado, mucho más que otras alternativas como podría ser el lenguaje XML. Por estos motivos, la mayoría de los servicios web que se construyen para ser consumidos por clientes web se implementan bajo **JSON.**

Manuales

Trabajar con JSON desde PHP

Manual para explicar el uso de JSON, la notación de objetos Javascript, dentro de aplicaciones web realizadas con PHP y...

Colecciones

Herramientas para trabajar con API y JSON

Herramientas trabajar con APIs y archivo...

Para qué se usa el JSON

JSON es un lenguaje para representar datos, con la sintaxis propia de los literales de objetos Javascript. Sus usos son varios, pero principalmente serían los siguientes:

Compartir y transferir información entre distintos sistemas de software.

Almacenar información en sistemas de persistencia.

Compartir información entre sistemas

El hecho de que JSON se representa en archivos de texto plano hace muy fácil su transferencia entre sistemas informáticos, usando protocolos ampliamente extendidos como HTTP.

Por tanto, JSON es un lenguaje habitualmente usado para transferir datos entre aplicaciones. En Internet el uso más importante es el de generar datos en el backend y enviarlos a las aplicaciones frontend. El flujo sería el siguiente:

Las aplicaciones frontend requieren datos para representar al usuario.

Mediante Javascript realizan solicitudes HTTP a un endpoint (URL) en el backend. Estas solicitudes se realizan de manera asíncrona mediante **Ajax.**

El backend recibe la solicitud sobre el dato que se requiere desde el frontend, o la operación que se desea realizar (ediciones, inserciones, etc.)

El backend realiza la operación, enviando el resultado al frontend en formato JSON.

El frontend recibe el JSON y generalmente construye mediante Javascript el HTML necesario para representar esa información convenientemente al usuario.

Este flujo es el que se realiza en los servicios web o APIs, que encontramos habitualmente en las aplicaciones web modernas.

Dentro del esquema MVC, desde el punto de vista del backend el JSON sería la vista. Mientras que en la aplicación frontend el JSON sería el modelo. Este JSON también se conoce con el nombre de DTO (Data Transfer Object) en estas arquitecturas de aplicaciones.

Almacenar información en bases de datos

Además de las bases de datos relacionales, que almacenan la información en tablas, existen actualmente otras bases de datos llamadas NoSQL o simplemente no relacionales, que almacenan los datos en formato JSON o alguna variante o extensión de este formato.

En estas bases de datos generalmente se tienen colecciones de elementos, como si fueran arrays, y en cada colección se almacenan objetos, que tienen una representación directa en JSON. Ejemplos de estas bases de datos serían MongoDB, CouchDB, Base de datos en tiempo real de Firebase, etc.

Actualmente también bases de datos relacionales como MySQL, Oracle o PostgreSQL soportan el almacenamiento de JSON. Es decir, tienen tipos de datos JSON que son capaces de entender y procesar como objetos.

EDITAR

Ejemplo de código JSON

Aquí podemos ver un ejemplo de un código en formato JSON.

```
{
  "nombre":"Jonh Doe",
  "profesion":"Programador",
  "edad":25,
  "lenguajes":["PHP","Javascript","Dart"],
  "disponibilidadParaViajar":true,
  "rangoProfesional": {
      "aniosDeExperiencia": 12,
      "nivel": "Senior"
  }
}
```

Los archivos JSON pueden tener cualquier nivel de anidación que sea necesario, ya que podemos colocar unos objetos dentro de otros, creando un árbol de datos de cualquier profundidad.

Reglas de sintaxis en JSON

Los archivos en formato JSON tienen las mismas reglas de sintaxis del lenguaje Javascript. Pero además necesitan cumplir algunas normas adicionales, que no son necesarias en los literales de objeto Javascript:

Estructuras disponibles

En JSON podemos estructurar los datos de dos maneras distintas, como arrays y objetos. Todos los archivos de JSON representan una de estas dos estructuras:

On objeto, es decir, una lista de pares clave / valor.

Una colección de elementos, lo que se conoce como array o arreglo.

Esto quiere decir que no puedes poner un valor primitivo (números, cadenas, boleanos) tal cual como un JSON válido, sino que tendrá que estar dentro de una estructura permitida, como un objeto o un array.

Llaves para representar objetos

Los archivos JSON que representan objetos comienzan siempre con una llave de inicio { y acaban con la llave de cierre }.

```
{
   "id": 44,
   "game": "Colonos de Catan",
   "author": "Klaus Teuber",
   "aves": "10+"
}
```

Corchetes para representar los arrays

Los archivos JSON que representan una lista de valores, es decir, un arreglo, comienzan por [y terminan por].

```
[ 1, 2, 8 ]
```

El anterior array es un JSON válido, pero es más común el uso de valores complejos en las casillas del array. De hecho, dentro de una casilla podríamos tener cualquier otro valor JSON válido, incluso otros objetos complejos.

```
[
   {
      "id": 56431
      "name": "El Padrino",
      "year": 1972
   },
   {
      "id": 7553
      "name": "El Padrino 2",
      "year": 1974
   },
   {
      "id": 19563
      "name": "El Padrino III",
      "year": 1990
   },
]
```

Valores representables con JSON

Todos los valores que podemos encontrar en JSON representan una de estas posibilidades:

Un valor primitivo, que pueden ser de tipos diversos de Javascript, numérico, cadena de caracteres, y boleanos (true / false).

Arrays, es decir, una secuencia de valores separados por comas. En los arrays utilizamos los corchetes para delimitar el principio y fin de los valores que contienen.

Objetos, es decir, otros elementos JSON con pares clave / valor.

Además, también es válido el valor null.

Uso de comillas dobles

Se deben usar siempre comillas dobles a la hora de encerrar cadenas y nombres de los atributos del objeto.

Este punto es importante, porque la sintaxis de Javascript para los literales de objetos permite colocar los nombres de las propiedades con y sin comillas, mientras que en JSON estamos obligados a colocar los nombres de las propiedades siempre con comillas dobles.

```
{
        "id": 1,
        "name": "Patatas nuevas",
        "description": "Patatas cosechadas este año",
        "available": true
}
```

En el código anterior, los nombres de las propiedades "id", "name", etc. se deben colocar con comillas dobles. Todos los nombres de los atributos del objeto deben tener comillas necesariamente. Además, las cadenas también se deben colocar con comillas dobles.

Comas para separar los pares clave / valor

Cada elemento del objeto JSON se separa del siguiente con una coma (,). Pero no debe haber una coma después del último elemento en ningún caso (Javascript sí que la permitiría).

UTF-8 obligatorio

Todos los archivos con código JSON deben estar codificados con UTF-8 de manera obligatoria. El archivo transferido en un JSON debe ser necesariamente codificado

como UTF8 para que no haya problemas de visualización de los caracteres contenidos en él.

Limitaciones de JSON

A pesar que JSON es un formato estupendo para su función, también tiene algunas limitaciones que convendría destacar.

No hay un esquema

Aunque en la mayoría de las ocasiones no necesitamos un esquema (schema), el lenguaje JSON no podría proporcionar un mecanismo para conseguir crear un esquema con el que forzar la corrección de los objetos, o validarlos. En muchos casos no tener esquema no es un problema y puede ser hasta una ventaja en los casos en los que se requiere flexibilidad de los modelos de los objetos, pero esa carencia se podría considerar una limitación.

No hay tipos

En JSON no podemos indicar los tipos de los datos de los elementos. Al tratarse de Javascript el lenguaje origen del JSON esto es absolutamente natural, pero no sería posible mejorarlo. Además, existe una única clase de valores numéricos.

No es posible insertar comentarios

De nuevo, no es que los necesitemos, pero si fuera necesario incluir información más allá de los datos puros, como comentarios, el lenguaje no lo permite.

Por qué JSON ha sido adoptado para la mayoría de los servicios web en lugar de XML

JSON es un formato de intercambio de información. Pero no es el único. De hecho, XML ha sido mucho más tradicional entre las alternativas de formatos para comunicación entre sistemas distribuidos. Sin embargo, JSON ha sido quien ha resultado como ganador en esta lucha entre formatos.

Actualmente JSON es la primera alternativa para compartir información, debido principalmente a que procesar en JSON es prácticamente inmediato en el lado del cliente. Es verdad que en el lado del servidor también resulta un formato ligero y soportado por todos los lenguajes, pero la clave está en el lado del frontend.

JSON es Javascript y parsearlo para que pueda ser introducido en la memoria del navegador, como un objeto nativo Javascript es prácticamente inmediato, no requiere coste alguno para el intérprete de Javascript. Esta situación es inmensamente más favorable que la que nos ofrece XML, que también se podría procesar por Javascript, pero que requeriría algoritmos mucho más costosos, en los que impicaría un recorrido a un árbol de etiquetas, la creación de las propiedades de los objetos en memoria, una a una, etc. Eso, sin contar con los posibles errores de escritura del XML, como una hipotética etiqueta mal cerrada, que obligarían al intérprete de Javascript a gastar mucho más tiempo.

Es cierto que los ordenadores actuales tienen suficiente capacidad de procesamiento para realizar todas estas tareas sobre un XML, pero son innecesarias ya que el formato JSON no necesita ningún trabajo adicional para poder interpretarse. Además, tenemos que pensar que las API REST o servicios web en general pueden ser consumidos por todo tipo de máquinas, no necesariamente un ordenador potente. Puede que se trate de un móvil de baja gama o un reloj, por ejemplo. En estos casos además con XML tendríamos el problema del consumo de batería para realizar todo ese procesamiento, lo que puede acortar el tiempo de funcionamiento de los sistemas y empeorar la experiencia de usuario. Si tenemos en cuenta además que en un acceso a una aplicación frontend moderna se pueden realizar decenas o cientos de consultas para obtener respuestas de un servicio web, esa necesidad de procesamiento extra en un XML se multiplicaría varias veces.

Por todo, JSON se ha impuesto sobre XML, aunque el propio XML tenga también otra serie de ventajas como lenguaje que también pueden ser apreciables en diversas situaciones. En la práctica el motivo fundamental de la elección es sobre todo la maximización del rendimiento de los clientes de las API.

Funciones para JSON en Javascript
El trabajo en Javascript con JSON incluye un par de métodos de utilidad, que dependen del objeto global "JSON". Mediante éstos es posible convertir un objeto Javascript a una cadena JSON y viceversa.

JSON.Parse(cadena)
Esta función permite convertir una cadena en formato JSON en una variable Javascript que contiene la representación del valor de esa cadena.

```
var objeto = JSON.parse('{"nombre": "juman", "edad": 33, "casado": false}');
```

JSON.stringify(objeto)
Esta función Javascript convierte un objeto, un array u otro valor en su representación como cadena en formato JSON.

```
var coche = {
  modelo: "Ford Focus",
  anioFabricacion: "2020",
  motorizacion: 'Gasolina'
}
var cadena = JSON.stringify(coche);
```

Las funciones de JSON para Javascript están disponibles en todos los navegadores actualmente. En Internet Explorer fueron introducidas en IE8, por lo que se pueden usar con total tranquilidad en cualquier navegador existente.

Accediendo a un servicio web que devuelve JSON desde Javascript mediante fetch
Uno de los mecanismos más frecuentes de las aplicaciones modernas es el acceso a datos de servicios web, generalmente implementados mediante un API REST que devuelve archivos en formato JSON.
Esto se puede hacer mediante diversos mecanismos del lado del cliente. El más estándar sería Ajax y el uso de fetch, que es la nueva función de Javascript para el acceso a datos del servidor, que devuelve una promesa.
Fetch es una función a la que le indicamos la URL a la que queremos acceder.

```
fetch('https://url/del/servicio/web)
```

Esta función nos devuelve una promesa, que debemos que esperar que se resuelva. Una vez resuelta obtenemos una respuesta, que nos indica los datos de la solicitud HTTP que se acaba de realizar.

```
.then(response => {
    // Código a ejecutar con la respuesta
})
```

Ahora bien, si esa respuesta era en formato JSON podemos acceder al objeto Javascript que representa ese JSON por medio del método response.json(). La dificultad es que este método devuelve a su vez otra promesa, que tenemos que encadenar.
El código completo de encadenar dos promesas sería el siguiente:

```
fetch('https://url/del/servicio/web')
  .then(response => response.json())
  .then(json => {
  // aquí tenemos el parámetro json que contien el dato que nos ha devuelto el servicio web.
  // podemos hacer cualquier cosa con él...
```

```
  console.log(json)
  });
```
Se puede encontrar más información de este mecanismo de acceso a datos del servidor en el artículo de fetch.

Funciones JSON para PHP

Por supuesto, PHP soporta JSON con funciones nativas, aunque la incorporación de estas funciones solamente se produjo en PHP 5.2. Existen dos funciones para el trabajo con JSON, una para codificar un elemento en formato JSON y otra para descodificar un JSON y crear una estructura de datos PHP.
json_encode($valor)
Esta función recibe un valor y devuelve un JSON que resulta de la codificación de ese valor en el formato de notación de objeto Javascript. El valor puede ser de cualquier tipo de dato, como podría ser una variable numérica, cadena, aunque generalmente se usa más habitualmente con arrays asociativos u objetos.

```
$cadena_en_formato_JSON = json_encode([
    'propiedad' => 'valor',
    'edad' => 33
]);
```

json_decode($cadena)

Esta función recibe una cadena de texto que debe tener un formato correcto en JSON. Devuelve una variable PHP que contiene la representación nativa del JSON en PHP. Por ejemplo, típicamente devolverá un objeto en el que encontramos las propiedades del JSON y los valores.

```
$producto = json_decode('{"producto":"ordenador portatil","precio":599}');
```

Trabajar con JSON en PHP es bastante sencillo con estas dos funciones. Se puede encontrar varios ejemplos prácticos en el Manual de JSON para PHP.
https://desarrolloweb.com/manuales/php-json.html