

JSON

<https://es.stackoverflow.com/questions/194527/generar-un-archivo-json-desde-php>

- JavaScript Object Notation – Notación de Objeto de JavaScript.
- **Concepto.**
 - Formato de texto ligero o sencillo creado para almacenar e intercambiar datos.
 - Es una especie de “lenguaje” común utilizado para la comunicación entre programas.
- **Características.**
 - Los archivos sólo contienen texto y se guardan con la extensión .json.
 - Su sintaxis se basa en la de JavaScript.
 - Para acceder a sus datos debe ser convertido a un objeto nativo de JavaScript, para ello, JavaScript cuenta con el objeto prototípico o clase JSON.
 - Es independiente de cualquier lenguaje de programación.
 - Al estar muy extendido, se han desarrollado API's para distintos lenguajes de programación que permiten analizar, procesar y transformar este tipo de archivos.
 - Debido a su éxito se ha convertido en una alternativa a XML, si bien, es común usar ambos en una misma aplicación.
 - Normativa y estandarización publicada por **IEFT** (Internet Engineering Task Force) y **ECMA** (European Computer Manufacturers Association).
 - Página oficial:
 - <https://www.json.org>
 - Última versión:
 - ECMS 4040 de diciembre del 2017.
- **Ventajas.**
 - Simplicidad en la sintaxis.
 - Rendimiento excelente, porque la información contenida en un archivo JSON se procesa muy rápido.
 - Funcionalidad, por poder usarse en muchas aplicaciones y para múltiples necesidades.
 - Accesibilidad, ya que, al ser texto, es fácil de leer tanto por personas humanas como por programas, lo que facilita el acceso a los datos.

Estructura de un archivo JSON.

- Un archivo JSON es una colección de pares clave (keys) / valor (value).
 - **Claves:**
 - Cadenas de caracteres que equivalen a las propiedades de cualquier objeto.

- Se escriben entre comillas dobles.
- **Valor/es.**
 - Dato/s asociado/s a una propiedad.

Tipos de datos.

- **Tipos de datos válidos en JSON:**
 - Arrays
 - Objetos.
 - Cadenas.
 - Números.
 - Enteros.
 - Decimales.
 - Booleanos.
 - Nulos.
- **Tipos de datos no permitidos en JSON:**
 - Expresiones regulares (RegExp).
 - Funciones (function).
 - Objetos Date.
 - Otros.

Anidamiento.

- Los archivos JSON pueden tener cualquier nivel de anidación, es decir, se pueden incluir objetos y arrays dentro de otros, y crear así una estructura de datos de varios niveles de profundidad.

Creación de un archivo JSON.

- **Vacío**
 - {}
- **Con datos.**
 - {"clave o propiedad 1": valor 1, "clave o propiedad 2": valor 2, ... "clave o propiedad N": valor N}
 - Ejemplo:
 - {"Usuario": "Rosa", "Contraseña": 1234}

Consideraciones sobre la sintaxis:

- El contenido de un objeto JSON se incluye dentro de llaves.
- Cada par clave/valor termina en una coma.
- Las propiedades del objeto (claves), se encierran entre comillas dobles.
- Las cadenas o textos se escriben entre comilla dobles.
- Los números, valores booleanos y el valor *null* se escriben sin comillas.
- Los arrays incluyen sus datos encerrados entre corchetes.

- Los objetos incluyen todo su contenido entre llaves anidado al objeto JSON principal.
- Dentro de un objeto JSON no se pueden incluir comentarios (Variantes de JSON, como JSON5 si los permiten).
- JSONLint, o la extensión Fix JSON para Visual Studio Code, permiten corregir errores en objetos JSON.
- Sintaxis:
 - “clave o propiedad”: “dato o valor”, // Cadenas.
 - “clave o propiedad”: dato o valor, // Números, booleanos o null.
 - “clave o propiedad”: [“dato o valor 1”, “dato o valor 2”, ..., “dato o valor N”], // Arrays.
 - “clave o propiedad”: {“clave o propiedad”: “dato o valor 1”, “clave o propiedad”: “dato o valor 2”, ..., “clave o propiedad”: “dato o valor N”}, // Objeto.
- Ejemplos:
- Array con datos de diversos tipos:
 - {


```

          “teléfono”:”1234567890”

          } // Una cadena, una propiedad.
```
 - {


```

          “nombre_producto”: “Boligrafo”,
          “color”: “Azul”,
          “precio”: 0.60,
          “código”:null

          } // Varios tipos de datos, varias propiedades.
```
 - {


```

          “plataforma”:”Steam”,
          “usuario1”:
            {
              “nombre”: “Juan”,
              “contraseña”:”1234”
            }

          } // Dos propiedades, siendo una un objeto con 2 propiedades.
```
 - {


```

          “plataforma”:”Steam”,
          “usuario1”: [“nombre”: “Juan”, “contraseña”:”1234”]

          } // Dos propiedades, siendo una de ellas un array con 2 propiedades.
```

- {


```

        "plataforma": "Steam",
        "usuario1":
          {
            "nombre": "Juan",
            "contraseña": "1234"
          }
        "usuario2":
          {
            "nombre": "Eva",
            "contraseña": "4321"
          }
      } // Tres propiedades, siendo objetos dos de ellas con sus propiedades.
    
```
- {


```

        "plataforma": "Steam",
        "usuario1":
          {
            "nombre": "Juan",
            "contraseña": "1234",
            "juegos": ["Fallout 4", "Metro Exodus", "Alien
              Isolation"]
          }
      } // Dos propiedades, siendo una de ellas un objeto que incluye 3
      propiedades entre las que hay un array.
    
```

JSON Minificado.

- **Concepto.**
 - La minificación consiste en reducir el tamaño de un archivo para conseguir una mayor velocidad y procesamiento y carga de éste.
 - Para ello se eliminan los espacios, saltos de línea, sangrías, tabulaciones y comentarios.
 - Así se consigue un archivo que, en muchos casos, es de una sola línea.
- **Ventajas.**
 - Ocupan menos espacio a nivel local y en los servidores.
 - Menor consumo de ancho de banda al descargarse más rápido.
 - Lectura rápida por parte de los programas.
 - Se puede usar la técnica con muchos lenguajes CSS, JavaScript, JSON, etc.
- **Inconvenientes.**
 - Son más complejos de leer para un humano.
 - También son más difíciles de depurar ya que no incluyen comentarios y los errores se muestran en una sola línea larga.
- Ejemplo:

- **Contenido normal.**

```
{
  "DNI":"123456789A",
  "nombre":"Rosa",
  "password":"abcd"
}
```

- **Contenido minificado.**

```
{"DNI":"123456789A","nombre":"Rosa","password":"abcd"}
```

JSON Y JAVASCRIPT.

Procesamiento de archivos JSON con JavaScript.

- Para trabajar con un archivo JSON (un string o cadena) en JavaScript, será necesario convertirlo en un objeto, y a la inversa, si se necesita enviar un objeto JavaScript vía red, es necesario convertirlo en una cadena o string y guardarlo en un archivo JSON.
- **Codificación.**
 - Conversión de un objeto JavaScript en un JSON.
 - JSON no soporta funciones, de modo que si se convierte un objeto que incluye funciones, éstas no aparecerán en el JSON.
 - Uso del método stringify().
 - Sintaxis:
 - JSON.stringify(objeto);
 - JSON.stringify(objeto, propiedades);
 - JSON.stringify(objeto, propiedades, número de líneas);
 - Valores:
 - **objeto.**
 - Nombre del objeto a convertir a JSON.
 - **Propiedades.**
 - Nombre de las propiedades con su valor que se quieren incluir en el JSON.
 - Opciones:
 - **[propiedad 1, propiedad2, ..., propiedad N]:**
 - Array de propiedades.
 - Muestra las propiedades y sus valores incluidos entre corchetes.
 - **[].**
 - Array vacío.
 - Se incluyen solo los corchetes y el objeto JSON aparecerá vacío.
 - **null.**

- Muestra todas las propiedades incluidas en el objeto JavaScript.
- **Número de líneas.**
 - Muestra el contenido del JSON en varias líneas, es decir, no minificado.
 - El resultado se visualiza así en la consola del navegador, no en el documento web.
- Ejemplo:
 - **Objeto sin funciones:**
 - `let cliente = {codigo: "23AZ4F", nombre:"Ana"}; //`
Creación del objeto literal *cliente*.
 - `let cadena_cliente = JSON.stringify(cliente); //` Conversión del objeto *cliente* en una cadena JSON cargada en la variable *cadena_cliente*.
 - `document.write(cadena_cliente); //` Se muestra el objeto cliente convertido a una cadena JSON.
 - **Objeto con funciones:**
 - `let usuario = {
 nombre:" Eugenia",
 contraseña: "1234",
 saludo: function () {
 return "Hola!";
 },
}; //` Creación del objeto literal *usuario*.
 - `let persona = JSON.stringify(usuario); //` Conversión del objeto usuario en una cadena JSON cargada en la variable *persona*.
 - `document.write(persona); //` Muestra las propiedades de objeto *usuario* convertido en una cadena JSON, sin incluir la función *saludo*.
 - **Objeto con propiedades a mostrar.**
 - `let cliente = {codigo: "23AZ4F", nombre:"Ana"}; //`
Creación del objeto literal *cliente*.
 - `let cadena_cliente = JSON.stringify(cliente, ["codigo"]); //`
Conversión del objeto *cliente* en una cadena JSON cargada en la variable *cadena_cliente*, que incluye nada más que la propiedad *codigo*.
 - `document.write(cadena_cliente); //` Se muestra únicamente *codigo* del objeto *cliente* convertido a una cadena JSON.
 - **Objeto con propiedades a mostrar en varias líneas.**
 - `let cliente = {codigo: "23AZ4F", nombre:"Ana"}; //`
Creación del objeto literal *cliente*.

- `let cadena_cliente = JSON.stringify(cliente, null, 2); //`
Conversión del objeto *cliente* en una cadena JSON cargada en la variable *cadena_cliente*, que mostrará todas sus propiedades en 2 líneas distintas.
 - `console.log(cadena_cliente); //` Se muestra el contenido del objeto *cliente* convertido en una cadena JSON, en 2 líneas distintas.
- **Decodificación.**
 - Conversión de un JSON en un objeto.
 - Esta acción se suele denominar parsear, y consiste en analizar la cadena que contiene un archivo JSON válido y convertirla en un objeto con la información del JSON estructurada de forma correcta para ser considerada un objeto.
 - Para ello se carga el objeto o cadena JSON en una variable.
 - Uso del método parse().
 - Sintaxis:
 - `JSON.parse(cadena o string);`
 - Ejemplo:
 - `let moviles = '{"marca": "LG", "modelo": "G3", "precio": 300.00}';`
`// Cadena JSON cargada en una variable.`
 - `let movil = JSON.parse(moviles); //` conversión de la cadena JSON cargada en la variable *móviles* en el objeto *móvil*.
 - `document.write ("Marca: " + móvil.marca); //` Acceso a la propiedad *marca* a través del operador punto (.) para objetos, que devuelve el dato "LG".

JSON Y PHP.

Procesamiento de archivos JSON con PHP.

- Al igual que en JavaScript, para trabajar con un archivo JSON en PHP será necesario convertirlo en un objeto (decodificar), o a la inversa, un objeto PHP se puede convertir en una cadena o string (codificar), y guardarlo en un archivo JSON.
- **Codificación.**
 - Consiste en convertir un objeto o un array PHP en un objeto JSON.
 - Para ello se usa el objeto PHP o el array (indexado o asociativo), como argumento de la función `json_encode()`.
 - Sintaxis:
 - `json_encode(objeto o array, máscara, profundidad);`
 - Parámetros:
 - **objeto.**
 - Nombre del objeto a convertir a JSON.
 - Nombre del array a convertir a JSON.
 - **máscara.**

- Constantes que modifican la salida del objeto JSON creado, mostrando caracteres de escape, valores Unicode o hexadecimales, etc.
- Algunas constantes:
 - **JSON_HEX_TAG.**
 - Los caracteres < y > se convierten a \u003C y \u003E respectivamente.
 - **JSON_HEX_APOS.**
 - Las comillas simples (') se convierten a \u0027.
 - **JSON_HEX_QUOT.**
 - Las comillas dobles (") se convierten a \u0022.
 - **JSON_HEX_AMP.**
 - El carácter & se convierte a \u0026.
 - **JSON_UNESCAPED_UNICODE.**
 - Codifica caracteres Unicode.
 - **JSON_FORCE_OBJECT.**
 - Devuelve un objeto en vez de un array cuando se usa un array no asociativo.
 - Útil cuando se espera un objeto y el array está vacío.
- **profundidad.**
 - Nivel de anidamiento hasta el que se quiere llegar en la conversión, suponiendo que existan anidamientos.
 - Se especifica con un valor numérico entero.
- Ejemplos:
 - **Array Asociativo:**
 - `$cliente = array(codigo: "23AZ4F", nombre:"Ana");` // Creación del array asociativo *\$cliente*.
 - `$cadena_cliente = echo json_encode($cliente);` // Conversión del array *\$cliente* en una cadena JSON cargada en la variable *\$cadena_cliente*.
 - `echo $cadena_cliente;` // Se muestra la variable *\$cliente* convertido a una cadena JSON.
 - `echo json_encode($cliente);` // Conversión del array *\$cliente* en una cadena JSON y mostrada con *echo*.
 - **Objeto con funciones:**
 - `class Usuario {`
`nombre="",`


```

        contraseña:"""
        public function getNombre() {
            return $this->nombre;
        }
    };
    $usuario1 = new Usuario(); //Creación del objeto $usuario1.
    $usuario1->nombre = "Eugenia";
    • $persona1= json_encode($usuario1); // Conversión del
      objeto $usuario1 en una cadena JSON cargada en la
      variable $persona1.
    • echo $persona1; // Muestra las propiedades del objeto
      $usuario1 convertido en una cadena JSON, sin incluir la
      función getNombre().

```

- **Decodificación.**

- Consiste en convertir un JSON en un objeto o en un array PHP.
- Para ello se carga el objeto o cadena JSON en una variable y se usa el método `json_decode()`.
- Realiza una validación mientras decodifica.
- Sintaxis:

- `json.decode (string, tipo de conversión, profundidad);`

- Parámetros:

- **string.**

- Cadena JSON a convertir.

- **tipo de conversión.**

- Valor booleano que define que estructura de datos se creará en PHP según los siguientes valores:

- **false.**

- Se creará un objeto PHP.
- Valor por defecto.

- **true.**

- Se creará un array asociativo.

- **profundidad.**

- Nivel de anidamiento hasta el que se quiere llegar en la conversión, suponiendo que existan anidamientos.
- Se especifica con un valor numérico entero.

- Ejemplos:

- Creación de un objeto.

- `$moviles = '{"marca": "LG", "modelo": "G3", "precio": 300.00}';` // Cadena JSON cargada en una variable.
- `$movil = json_decode($moviles);` // Conversión de la cadena JSON cargada en la variable *móviles* en el objeto *móvil*.

- echo "Marca: " . \$movil->marca; // Acceso a la propiedad *marca* a través del operador flecha (->) para objetos, que devuelve el dato "LG".
 - Creación de un array asociativo.
 - \$moviles = '{"marca": "LG", "modelo": "G3", "precio": 300.00}'; // Cadena JSON cargada en una variable.
 - \$movil = json_decode (\$moviles, true); // Conversión de la cadena JSON cargada en la variable *móviles* en el objeto *móvil*.
 - echo "Marca: " . \$movil["marca"]; // Acceso a la propiedad *marca* a través de los corchetes [] para índices o claves de arrays, siendo el resultado el dato "LG".
- **Validación.**
 - Con la función json_validate() se puede comprobar si una cadena se corresponde con un JSON Valido sin necesidad de decodificarlo.
 - Devuelve true o false.
 - No tiene sentido usar la función tras json_decode(), ya que también valida. Su uso mejor cuando no se vaya a decodificar inmediatamente y se quiere conocer si el contenido a decodificar es un JSON válido.
 - A partir de la versión PHP 8.3.
 - Sintaxis:
 - json.validate (string, profundidad, caracteres inválidos);
 - Parámetros:
 - **string.**
 - Cadena JSON a validar.
 - **profundidad.**
 - Nivel de anidamiento hasta el que se quiere llegar en la conversión, suponiendo que existan anidamientos.
 - Se especifica con un valor numérico entero.
 - **Caracteres inválidos.**
 -
 - Ejemplo:
 - \$numeros = '{"uno": 1, "dos": 2}';
 - \$valido = json_validate(\$numeros);
- **Errores.**
 - Cuando una cadena no contiene un JSON válido, o se produce un error de decodificación, se puede obtener información sobre el error producido usando las funciones:
 - json_last_error().
 - Devuelve un número entero que se corresponde con una constante predefinida que identifica el último error producido.
 - Algunos tipos de errores:
 - **JSON_ERROR_NONE.**

- No ha habido ningún error.
 - Valor devuelto 0.
- **JSON_ERROR_DEPTH.**
 - Se ha alcanzado el máximo nivel de profundidad.
 - Valor devuelto 1.
- **JSON_ERROR_STATE_MISMATCH.**
 - JSON inválido o mal formado.
 - Valor devuelto 2.
- **JSON_ERROR_CTRL_CHAR.**
 - Error de control de caracteres, posiblemente incorrectamente codificado.
 - Valor devuelto 3.
- **JSON_ERROR_SYNTAX.**
 - Error de sintaxis.
 - Valor devuelto 4.
- **JSON_ERROR_UTF8.**
 - Caracteres UTF-8 mal formados, posiblemente incorrectamente codificado.
 - Valor devuelto 5.
- Sintaxis:
 - json_last_error().
- Ejemplo:
 - \$movil = json_decode(\$moviles);
 - \$error = json_last_error();
 - echo \$error; // Devolverá un número entero que se corresponde con la constante que identifica el error.
- **json_last_error_msg().**
 - Devuelve un mensaje que identifica el último error producido.
 - Sintaxis:
 - json_last_error_msg().
 - Ejemplo:
 - \$movil = json_decode_msg(\$moviles);
 - \$mensaje_error = json_last_error_msg();
 - echo \$mensaje_error; // Devolverá un texto o mensaje si se produce, o no, un error en la decodificación.