Academia

Leer datos de JSON con Selectores

Aprende a utilizar los selectores de BillMySales para personalizar los datos que se usan en la creación de tus DTE.

Anonymous · 17/09/2026 · #espanol, #basico

Tabla de contenidos

Leer datos de JSON con Selectores

Aprende a utilizar los selectores de BillMySales para personalizar los datos que se usan en la creación de tus DTE.

Anonymous · 17/09/2026 · #espanol, #basico

Leer datos de JSON con Selectores

Última actualización el 17/09/2026

Introducción

Anonymous · 17/09/2026

Introducción

Última actualización el 17/09/2026

¿Qué es un JSON?

¿Qué es un JSON?

Anonymous · 17/09/2026

¿Qué es un JSON?

JSON, que significa JavaScript Object Notation, es un formato de archivo de texto utilizado para almacenar y transportar datos. Su popularidad se debe a su simplicidad, legibilidad y capacidad para representar estructuras de datos complejas de manera eficiente. Es ampliamente usado en la comunicación entre servidores y aplicaciones web, así como en numerosos otros contextos, incluyendo la facturación electrónica.

Características del Formato JSON

- **Texto Plano:** JSON es un formato de texto, lo que lo hace legible por humanos y fácilmente intercambiable entre sistemas y plataformas.
- **Estandarizado:** Sigue una sintaxis específica y reglas para representar datos, lo que asegura su consistencia y fiabilidad en diferentes aplicaciones y sistemas.

Estructuras de Datos en JSON

JSON soporta varias estructuras de datos, incluyendo:

Objetos

Son colecciones de pares clave-valor. Las claves son siempre strings, mientras que los valores pueden ser cualquier otro tipo de datos JSON. Un objeto se representa con llaves “{}”.

```
{
  "transaccion": {
    "id": "TX123",
    "monto": 1000,
    "moneda": "USD"
  }
}
```

Arreglos

Son listas ordenadas de valores y se representan con corchetes “[]”. Los valores dentro de un arreglo

(o array) pueden ser de cualquier tipo de datos JSON, incluyendo otros arreglos u objetos.

```
{
  "productos": [
    {"nombre": "Laptop", "precio": 800},
    {"nombre": "Teclado", "precio": 100}
  ]
}
```

Valores Primitivos

Estos incluyen tipos como cadenas literales de texto (strings), números, booleanos (true o false) y null.

```
{
  "nombre": "Empresa X",
  "activo": true,
  "balance": null
}
```

Importante

BillMySales está diseñado para trabajar con todas estas estructuras. Sin embargo, para estructuras muy anidadas o complejas, es posible que se requiera un mayor detalle en la especificación de las configuraciones para acceder correctamente a los datos deseados. Te recomendamos usar la herramienta de pruebas que se encuentra en tu cuenta, específicamente dentro de una orden que tengas en BillMySales.

Ejemplo en Facturación Electrónica

En un contexto de facturación electrónica genérica (no asociada a ningún país en concreto), un JSON podría utilizarse para representar una factura con todos sus detalles. Un ejemplo sería:

```
{
  "factura": {
    "numero": "F123456",
    "fecha": "2023-01-01",
    "cliente": {
      "nombre": "Cliente ABC",
      "identificacion": "ID12345"
    },
    "items": [
      {
        "producto": "Producto 1",
        "cantidad": 2,
```

```
        "precio_unitario": 150
      },
      {
        "producto": "Producto 2",
        "cantidad": 1,
        "precio_unitario": 200
      }
    ],
    "total": 500
  }
}
```

Este JSON representa una factura con información sobre el cliente, los productos vendidos y el total de la transacción. La estructura clara y jerárquica de JSON facilita la organización y acceso a la información, lo cual es esencial en procesos como la facturación electrónica.

En resumen, JSON es un formato poderoso y versátil para el intercambio de datos, y su capacidad para representar estructuras de datos complejas lo hace ideal para una variedad de aplicaciones, incluyendo la facturación electrónica.

Última actualización el 17/09/2026

¿Cómo leer un JSON?

¿Cómo leer un JSON?

Anonymous · 17/09/2026

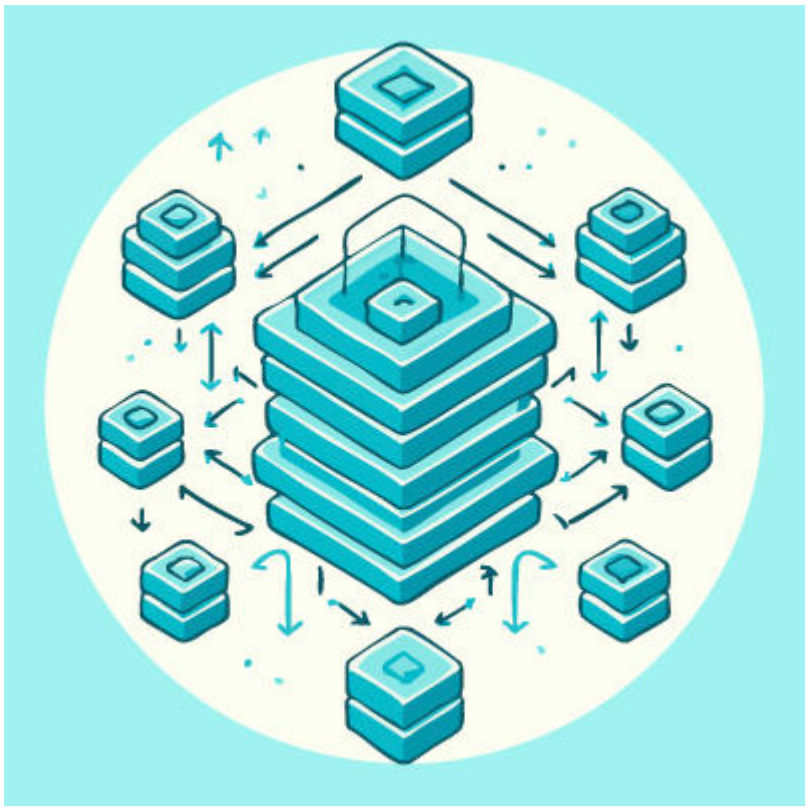
¿Cómo leer un JSON?

Es común que un usuario desee personalizar desde dónde obtener ciertos datos en sus órdenes procesadas por BillMySales. Por ejemplo, un usuario puede tener el RUT (VAT number) de un cliente en cierto campo, pero otro usuario podría tenerlo en otro campo (por configuraciones o personalizaciones de su checkout).

En BillMySales creemos en la personalización y configuración extrema de tu pasarela. Que puedas decidir exactamente qué dato quieres extraer de la orden de tu tienda y que puedas decidir dónde colocarla en tu facturador. Para lograr esto, cada usuario puede definir diferentes campos que son configurables en su cuenta, donde estas configuraciones permiten extraer datos de un JSON (el de la tienda) y pasarlos a otro JSON (el del facturador).

Selector

Para resolver esta situación, y que puedas configurar tus pasarelas para extraer los datos de los JSON, BillMySales utiliza una herramienta que llamamos selector (o selectores). Este selector es simplemente una forma de escribir “cómo” obtener cierto dato desde un JSON. Permitiendo que los campos sean configurables (personalizables) por cada usuario según sus necesidades.



Específicamente, un selector es una cadena de texto que describe cómo localizar un valor específico dentro de una estructura de datos JSON. BillMySales ofrece una variedad de tipos de selectores, incluyendo selectores simples, selectores anidados, selectores con operadores lógicos y selectores con condiciones de IF ternarios.

En BillMySales, un selector es una herramienta poderosa que permite a los usuarios especificar cómo y de dónde obtener datos específicos de un JSON para su uso en procesos de facturación electrónica. Un selector puede ser simple, extrayendo datos de un nivel específico del JSON, o avanzado, permitiendo la extracción de datos anidados, la selección basada en condiciones, y más.

Esta guía proporciona instrucciones detalladas sobre cómo utilizar los selectores en el módulo BillMySales para leer datos de estructuras JSON. Los selectores permiten acceder a datos específicos dentro de un JSON de manera flexible y potente.

Importante

Actualmente no es posible leer y asignar cualquier campo en los documentos. Puedes leer cualquiera, pero la asignación requiere programación con las aplicaciones actuales de BillMySales. Si requieres una mejora respecto a esto, y que se pueda personalizar un campo que hoy no es personalizable, contáctanos y revisaremos la factibilidad de agregar un selector para dicho campo.

¿Qué se puede hacer con el selector?

El selector es un patrón de elementos y otros términos que permite seleccionar un valor dentro de un

diccionario (dentro del JSON). Su principal funcionalidad es permitir capturar uno o más valores que puede tener un índice dentro del JSON.

En general:

- Permite seleccionar nodos anidados dentro de un JSON.
- Permite seleccionar elementos de un arreglo dentro de un JSON.
- Permite seleccionar elementos de un arreglo de diccionarios donde el elemento a obtener depende del valor de otro índice en el mismo diccionario.
- Permite obtener un subconjunto de los datos (objeto). Esto es útil al usar pruebas de selectores que se deben buscar solo dentro de un índice en particular del JSON.
- Se permite concatenar selectores. Tanto con otros selectores como con cadenas de texto.
- Se permite elegir el primer elemento con valor dentro de un listado de selectores. Esto además permite añadir un valor por defecto al campo si no existe.

Es posible combinar los selectores de diferentes maneras para conseguir seleccionar los datos del JSON.

Última actualización el 17/09/2026

Probar un selector

Probar un selector

Anonymous · 17/09/2026

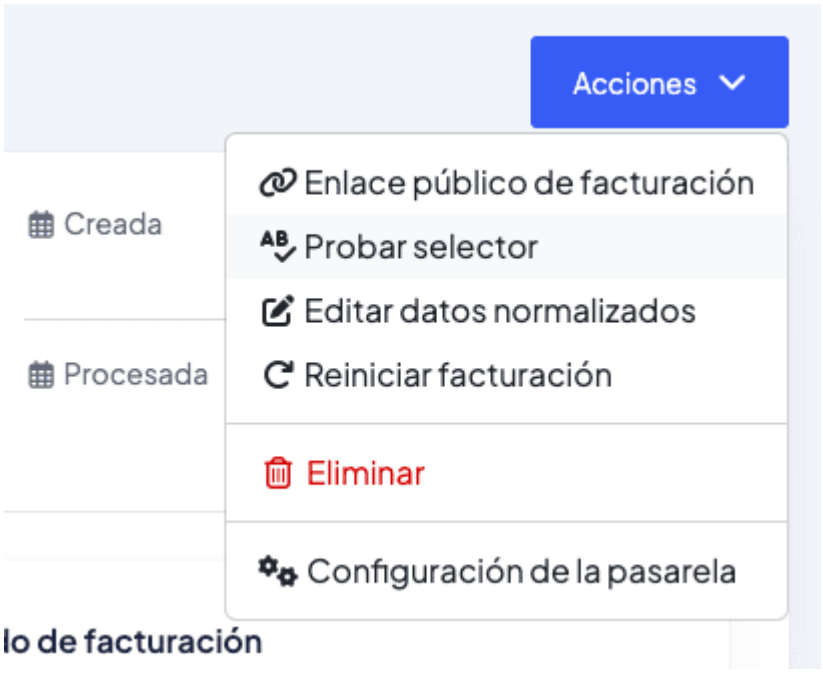
Probar un selector

Escribir un selector puede parecer complicado, y a veces es fácil cometer errores. Por lo que te recomendamos que pruebes tu selector con alguna orden de tu cuenta de BillMySales. O sea, pruébalo con datos reales. Solo cuando lo tengas probado lo usas en la configuración de tu pasarela de facturación. Así no tendrás problemas en tu configuración.

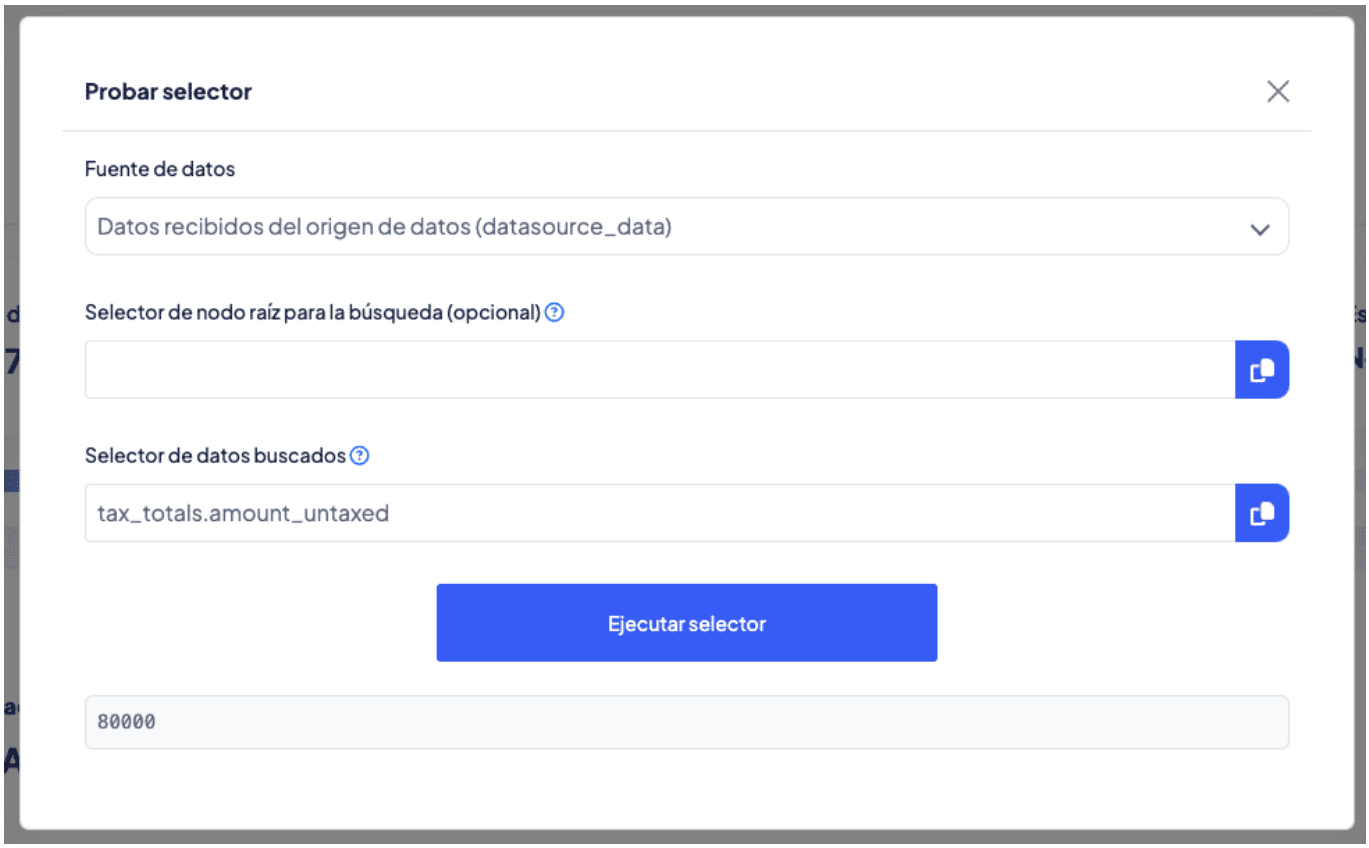
Importante

Es crucial probar los selectores con datos reales de tus órdenes en la interfaz de BillMySales. Así podrás ingresar tu selector y ver el resultado en tiempo real con los datos de una orden específica.

Para probar solo debes dirigirte a una orden y en la esquina superior derecha buscar en el menú la opción “Probar selector”:



Al hacer clic tendrás una ventana con las opciones para ingresar el selector y probar si trae los datos que necesitas. Luego podrás usar ese selector en la configuración de tu pasarela para personalizar tu cuenta.



Nota

No te preocupes de la opción “Selector de nodo raíz para la búsqueda”, es opcional y normalmente no la usarás, ya que es para casos muy específicos cuando se hacen desarrollos o integraciones. Así que ese campo déjalo en blanco.

Los campos del formulario son:

- **Fuente de datos:** son los posibles JSON con datos que podemos usar. Las opciones acá son:
  - Datos recibidos del origen de datos (datasource_data).
  - Datos normalizados a partir de los datos recibidos (datasource_invoice).
  - Datos preparados que se van a mandar al facturador (biller_data).
  - Datos de respuesta del facturador (biller_invoice).
- **Selector de nodo raíz para la búsqueda:** permite elegir un subconjunto de los datos de la fuente de datos (normalmente no usado).
- **Selector de datos buscados:** este es el selector real que se desea usar luego en una configuración.

En el resultado podremos ver, dependiendo del selector usado, lo siguiente:

- Los datos **escalares** obtenidos: texto (string), un número o un valor booleano.
- Un valor **null** cuando el selector no encuentra datos.
- Un **JSON** con el objeto obtenido, cuando no es un texto (string), un número o un valor booleano.

Última actualización el 17/09/2026

Selectores básicos

Anonymous · 17/09/2026

Selectores básicos

Última actualización el 17/09/2026

Selector simple

Selector Simple

Anonymous · 17/09/2026

Selector Simple

Un selector simple accede a un valor directamente por su clave.

- **Formato:** clave
- **Ejemplo:** Para obtener el valor de la clave nombre en { “nombre”: “Juan” }, se usa el selector “nombre”.

Ejemplos

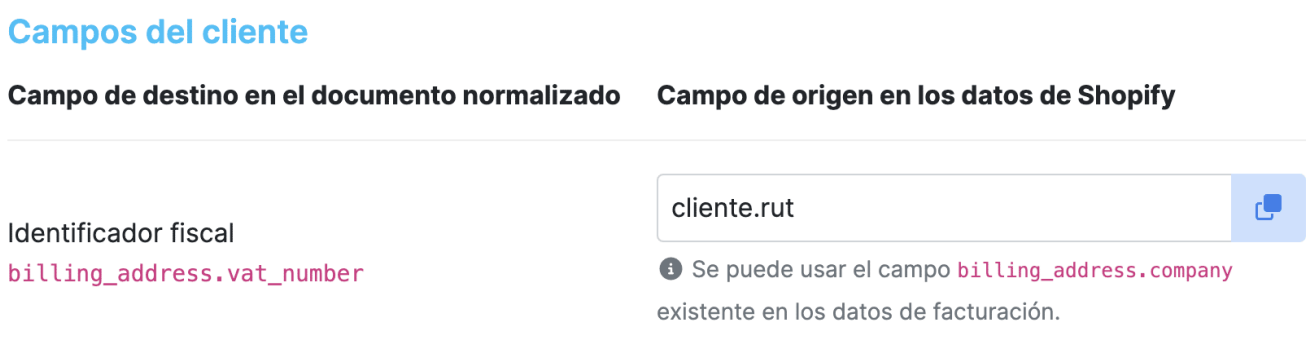
Acceso Directo a una Clave:

- Dado el JSON: {“nombre”: “Ana”}
- Selector: nombre
- Resultado: “Ana”

Acceso Directo a una Clave:

- Dado el JSON: {“cliente_rut”: “1-9”}
- Selector: cliente_rut
- Resultado: “1-9”

Este último selector permite extraer los datos del RUT (VAT number en Chile) y asignarlo en un campo en la configuración de BillMySales. En este caso sería en el campo “Identificador fiscal (VAT number)”. En la configuración de una pasarela de facturación, sección origen de datos, se vería así:



Selector anidado

Selector Anidado Los selectores anidados acceden a valores dentro de estructuras JSON anidadas.

Anonymous · 17/09/2026

Selector Anidado

Los selectores anidados acceden a valores dentro de estructuras JSON anidadas.

- **Formato:** clave1.clave2...claveN
- **Ejemplo:** Para acceder al valor en { “grupo”: { “subgrupo”: { “clave”: “valor” }}}, se usa el selector “grupo.subgrupo.clave”.

Ejemplos

Acceso a una Clave Anidada:

- **Dado el JSON:** {“usuario”: {“nombre”: “Carlos”, “edad”: 30}}
- **Selector:** usuario.nombre
- **Resultado:** “Carlos”

Selector Anidado:

- **Dado el JSON:** {“cliente”: {“rut”: “1-9”}}
- **Selector:** cliente.rut
- **Resultado:** “1-9”

Última actualización el 17/09/2026

Selector con OR

Selectores con OR

Anonymous · 17/09/2026

Selectores con OR

Permiten especificar múltiples opciones, retornando el valor de la primera opción no vacía.

- **Formato:** selector1||selector2
- **Ejemplo:** “claveInexistente||claveExistente” devuelve el valor de claveExistente si claveInexistente no existe o su valor es vacío.

Ejemplos

Selector con Alternativa:

- **Dado el JSON:** {“nombre”: “Laura”, “apodo”: “Lau”}
- **Selector:** nombreInexistente||apodo
- **Resultado:** “Lau”

Cadena Literal como Alternativa:

- **Dado el JSON:** {“nombre”: “Pedro”}
- **Selector:** apodo||“Desconocido”
- **Resultado:** “Desconocido”

Selector con Valor por Defecto:

- **Selector:** (selector_inexistente)||“Valor por defecto”
- **Resultado:** “Valor por defecto”

Nota

Aunque los operadores como OR ofrecen gran flexibilidad, su uso excesivo puede hacer que los selectores sean difíciles de entender y mantener.

Última actualización el 17/09/2026

Selectores con IF ternario

Selectores con IF ternario

Anonymous · 17/09/2026

Selectores con IF ternario

Permiten seleccionar un valor basado en una condición.

- **Formato:** ((selector_condición) operador “valor_condición” ? (selector_si_verdadero) : (selector_si_falso))
- **Ejemplo:** ((edad) >= “18” ? “adulto” : “menor”) devuelve “adulto” si la edad es mayor o igual a 18, de lo contrario devuelve “menor”.

Operadores disponibles en IF ternario

Los selectores con IF ternarios soportan varios operadores:

- Igualdad: =, ==
- Desigualdad: !=, <>
- Mayor que: >
- Menor que: <
- Mayor o igual que: >=
- Menor o igual que: <=
- Contiene (para listas y cadenas): contains
- Longitud (para listas y cadenas): length

Ejemplos

Condición Simple:

- Dado el JSON: {“edad”: 20}
- Selector: ((edad) < “18” ? “Menor” : “Adulto”)
- Resultado: “Adulto”

Condición con Contiene en Lista:

- Dado el JSON: {“frutas”: [“manzana”, “banana”, “naranja”]}
- Selector: ((frutas) contains “banana” ? “Encontrada” : “No Encontrada”)
- Resultado: “Encontrada”

Selector con Condiciones IF:

- Dado el JSON: {“valor”: 100}
- Selector: ((valor) > “50” ? “Alto” : “Bajo”)
- Resultado: “Alto”

Nota

Aunque las condiciones IF ofrecen gran flexibilidad, su uso excesivo puede hacer que los selectores sean difíciles de entender y mantener.

Última actualización el 17/09/2026

Casos especiales

Anonymous · 17/09/2026

Casos especiales

Última actualización el 17/09/2026

Cadenas de texto

Cadenas de texto

Anonymous · 17/09/2026

Cadenas de texto

Es posible definir cadenas literales. De esta forma se puede usar un texto (string) para ser concatenado (unido) a otros selectores.

- **Formato:** “cadena literal”

Se mostrará la cadena tal cual es ingresada.

Ejemplos

Concatenación:

- Dado el JSON: {“nombre”: “Empresa”, “id”: “123”}
- Selector: “ID de “(nombre)”: “(id)
- Resultado: “ID de Empresa: 123”

Última actualización el 17/09/2026

Diccionarios

Diccionarios

Anonymous · 17/09/2026

Diccionarios

Los selectores pueden acceder a elementos específicos en diccionarios.

Ejemplo en Diccionario Anidado: diccionario.clave.subclave accede a subclave dentro de un diccionario anidado en clave.

Última actualización el 17/09/2026

Listas

Listas

Anonymous · 17/09/2026

Listas

Los selectores pueden acceder a elementos específicos en listas o arreglos

Ejemplo en Lista: lista[2] accede al tercer elemento de una lista.

Ejemplos

Selector de Arreglos:

- **Dado el JSON:** {“array”: [1, 2, 3]}
- **Selector:** array[1]
- **Resultado:** 2

Selector de Arreglo Anidado:

- **Dado el JSON:** {“nested_array”: {“array”: [1, 2, 3]}}
- **Selector:** nested_array.array[2]
- **Resultado:** 3

Selector de Arreglo de Diccionarios:

- **Dado el JSON:** {“mixed”: [{“key”: 10, “value”: “hola”}, {“key”: 20, “value”: “mundo”}]}
- **Selector:** mixed[key=20:value]
- **Resultado:** “mundo”

Nota

Este último caso es interesante porque permite obtener, de un objeto que está en una lista de objetos, el valor de un índice a partir del valor de otro índice del mismo objeto.

Última actualización el 17/09/2026

Formato de salida

Formato de salida

Anonymous · 17/09/2026

Formato de salida

Los selectores adecuarán el formato de la salida según el tipo de datos de los elementos que participen en la selección del selector.

Respetar tipo de datos original

Si existe un solo selector como resultado, ya sea porque:

- Se pidió solo un selector
- O, el selector está formado por varios selectores unidos mediante OR y se obtiene el primer valor no vacío.

El tipo de dato del resultado del selector será el tipo de dato del elemento seleccionado del JSON.

Tipo de datos como cadena de texto (string)

Si el selector está concatenando resultados de varios selectores o cadenas literales, y el resultado no es vacío, el tipo de datos del resultado será siempre una cadena de texto o string.

Tipo de datos null

Si el elemento seleccionado mediante el selector tiene los valores en el JSON:

- null.
- “” (cadena de texto vacía).

El resultado del selector será null.

Selector no encontrado en el JSON

Si se escribe un selector de manera errónea, o que en el JSON que estamos usando para extraer el dato no existe lo buscado, el resultado será **null**, un caso no existente.

Por ejemplo, si tenemos el siguiente JSON:

```
{
    "array": [1, 2, 3]
}
```

Y usamos el siguiente selector:

```
array2[2]
```

Obtendremos **null** como resultado, porque se buscó el índice **array2** y este índice no existe en el JSON.

Última actualización el 17/09/2026

JSON Path

Anonymous · 17/09/2026

JSON Path

Última actualización el 17/09/2026

Ventajas y desventajas

Ventajas y desventajas

Anonymous · 17/09/2026

Ventajas y desventajas

Es verdad, es natural pensar “¿para qué creamos selectores si existe JSONPath?”. Tanto la implementación personalizada con selectores como el uso de JSONPath tienen sus ventajas y desventajas.

Principalmente las ventajas de usar los selectores son:

1. Tenemos control total de la funcionalidad y la podemos adaptar según sea necesario para nuestra plataforma BillMySales.
2. Es muy simple escribir un selector para casos específicos.
3. Nos encanta aprender ~~y reinventar la rueda.~~

Algunas ventajas de usar JSONPath son:

1. Es un estándar de la industria, por lo que muchos usuarios técnicos ya lo conocerán.
2. Tiene más características y funcionalidades para manejar los datos de un JSON.

Nuestros selectores personalizados han sido probados minuciosamente y están funcionando bien para las necesidades específicas de BillMySales. Sin embargo, como nos gusta siempre estar mejorando, hemos añadido compatibilidad con JSONPath. De esta forma, puedes escribir un selector tanto con nuestro formato como con el estándar de la industria.

Última actualización el 17/09/2026

Selector JSONPath

Selector JSONPath

Anonymous · 17/09/2026

Selector JSONPath

Consideraremos el siguiente JSON como base para todos los ejemplos de esta sección. Este JSON representa datos típicos en un contexto de facturación electrónica (no asociada a ningún país específico):

```
{
 "cliente": {
   "nombre": "Juan Pérez",
   "es_vip": false
 },
 "total": 150000,
 "items": [
   {"nombre": "Producto A", "precio": 50000},
   {"nombre": "Producto B", "precio": 100000}
 ],
 "factura": {
   "detalles": [
     {"producto": "Producto A", "cantidad": 1},
     {"producto": "Producto B", "cantidad": 2}
   ]
 },
 "productos": [
   {"id": "A100", "nombre": "Producto A", "precio": 25000},
   {"id": "B200", "nombre": "Producto B", "precio": 50000}
 ],
 "ventas": [
   {"producto": "Producto 1", "cantidad": 3},
   {"producto": "Producto 3", "cantidad": 6},
   {"producto": "Producto 5", "cantidad": 10}
 ],
 "historial_compras": [
   {"codigo": "C101", "detalle": {"fecha": "2023-01-15"}},
   {"codigo": "C102", "detalle": {"fecha": "2023-03-15"}},
   {"codigo": "C103", "detalle": {"fecha": "2023-05-20"}}
 ],
 "numeros_de_serie": ["NS100", "NS101", "NS102"]
}
```

Ejemplos

1. Acceso Directo a una Clave en Primer Nivel

- **Selector JSONPath:** $.total
- **Descripción:** Extrae el valor total de la factura.
- **Resultado:** 150000

2. Acceso a un Elemento Anidado

- **Selector JSONPath:** $.cliente.nombre
- **Descripción:** Obtiene el nombre del cliente de la factura.
- **Resultado:** “Juan Pérez”

3. Acceso a un Elemento de un Arreglo por Índice

- **Selector JSONPath:** $.items[0].precio
- **Descripción:** Recupera el precio del primer artículo en la factura.
- **Resultado:** 50000

4. Acceso a un Elemento Anidado Dentro de un Arreglo

- **Selector JSONPath:** $.factura.detalles[1].cantidad
- **Descripción:** Accede a la cantidad del segundo artículo en el detalle de la factura.
- **Resultado:** 2

5. Filtrar Elementos de un Arreglo (por valor existente)

- **Selector JSONPath:** $.productos[?(@.id == ‘A100’)].precio
- **Descripción:** Busca y obtiene el precio de un producto específico, identificado por su ID.
- **Resultado:** 25000

6. Filtrar Elementos de un Arreglo (resultado vacío)

- **Selector JSONPath:** $.productos[?(@.id == ‘A999’)]
- **Descripción:** Intenta encontrar un producto con un ID que no existe en la lista, esperando un resultado vacío.
- **Resultado:** None

7. Acceso a Todos los Elementos de un Arreglo

- **Selector JSONPath:** $.numeros_de_serie[*]
- **Descripción:** Obtiene todos los números de serie de los productos.
- **Resultado:** [“NS100”, “NS101”, “NS102”]

8. Obtener Elementos Basados en una Condición Compleja

- **Selector JSONPath:** $.ventas[?(@.cantidad > 5)].producto
- **Descripción:** Selecciona los nombres de los productos que se vendieron en cantidades mayores a cinco.
- **Resultado:** [“Producto 3”, “Producto 5”]

9. Obtener un Elemento Anidado en un Diccionario dentro de un Arreglo

- **Selector JSONPath:** $.historial_compras[?(@.codigo == ‘C102’)].detalle.fecha
- **Descripción:** Encuentra la fecha de una compra específica en el historial de compras, usando su código.
- **Resultado:** “2023-03-15”

10. Acceso Directo a un Elemento Booleano

- **Selector JSONPath:** $.cliente.es_vip
- **Descripción:** Verifica si el cliente es marcado como VIP.
- **Resultado:** false

Estos ejemplos demuestran la versatilidad y potencia de JSONPath para acceder y manipular datos en un JSON, especialmente útil en escenarios de facturación electrónica, permitiendo desde el acceso a datos simples hasta la extracción basada en condiciones complejas.

Última actualización el 17/09/2026

Combinaciones con JSONPath

Combinaciones con JSONPath

Anonymous · 17/09/2026

Combinaciones con JSONPath

Consideraremos el siguiente JSON como base para todos los ejemplos de esta sección. Este JSON representa datos típicos en un contexto de facturación electrónica (no asociada a ningún país específico):

```
{
  "factura_id": "FAC-00123",
  "cliente": {
    "nombre": "Empresa XYZ",
    "rut": "76.123.456-7"
  },
  "fecha_emision": "2023-10-05",
  "detalles": [
    {"producto": "Laptop Pro", "cantidad": 2, "precio_unitario": 1200000},
    {"producto": "Monitor 24\"", "cantidad": 3, "precio_unitario": 240000}
  ],
  "estado_pago": "pendiente",
  "montos": {
    "neto": 3120000,
    "iva": 592800,
    "total": 3712800
  },
  "historial_pagos": [],
  "es_exportacion": false
}
```

Ejemplos

1. Acceso Directo y Concatenación de Texto

- **Selector:** “factura_id: “($.factura_id)
- **Descripción:** Concatena el texto “factura_id: “ con el ID de la factura.
- **Resultado:** “factura_id: FAC-00123”

2. Selección con OR y JSONPath

- **Selector:** ($.estado_pago_inexistente)||($.estado_pago)

- **Descripción:** Utiliza el operador OR para seleccionar el estado de pago de la factura, proporcionando un valor por defecto si el campo original no existe.
- **Resultado:** “pendiente”

3. Valor por Defecto con OR y JSONPath

- **Selector:** ($.historial_pagos_inexistente)||“Sin pagos”
- **Descripción:** Ofrece un valor por defecto en caso de que el historial de pagos no esté presente o esté vacío.
- **Resultado:** “Sin pagos”

4. Condición con IF Ternario y JSONPath

- **Selector:** ((($.estado_pago_inexistente)||($.estado_pago)) == “pendiente” ? (“Pendiente”) : (“Pagado”))
- **Descripción:** Comprueba si el estado del pago es “pendiente” y muestra “Pendiente” o “Pagado” en consecuencia.
- **Resultado:** “Pendiente”

5. Concatenación con Valor de un Arreglo

- **Selector:** “detalles[0]: “($.detalles[0].producto)
- **Descripción:** Concatena la descripción “detalles[0]: “ con el nombre del primer producto en los detalles de la factura.
- **Resultado:** “detalles[0]: Laptop Pro”

6. Filtrado de Arreglo y Concatenación

- **Selector:** “Productos con precio > 500000: “ + str($.detalles[?(@.precio_unitario > 500000)].producto)
- **Descripción:** Concatena una descripción con la lista de productos cuyo precio unitario supera los 500000.
- **Resultado:** “Productos con precio > 500000: [‘Laptop Pro’]”

7. Acceso a Elemento Anidado y Concatenación

- **Selector:** “Total factura: “($.montos.total)
- **Descripción:** Concatena “Total factura: “ con el monto total de la factura.
- **Resultado:** “Total factura: 3712800”

8. Filtrado de Arreglo de Diccionarios y Concatenación

- **Selector:** “Cantidad de Laptops: “($.detalles[?(@.producto == “Laptop Pro”)].cantidad)
- **Descripción:** Concatena “Cantidad de Laptops: “ con la cantidad de laptops compradas según los detalles de la factura.
- **Resultado:** “Cantidad de Laptops: 2”

9. Concatenación de un Valor Extraído del Arreglo con Texto

- **Selector:** “Primer producto: “($.detalles[0].producto)**
- **Descripción:** Concatena “Primer producto: “ con el nombre del primer producto en la factura.
- **Resultado:** “Primer producto: Laptop Pro”

10. Acceder a un Valor Anidado y Concatenar con un Valor de un Arreglo Usando JSONPath

- **Selector:** “Cliente y neto: “($.cliente.nombre)” y “($.montos.neto)
- **Descripción:** Concatena el nombre del cliente y el monto neto de la factura, separados por “ y “.
- **Resultado:** “Cliente y neto: Empresa XYZ y 3120000”

Estos ejemplos demuestran cómo los selectores pueden ser utilizados de manera flexible para extraer y combinar datos de un JSON, especialmente en contextos de facturación electrónica.

Última actualización el 17/09/2026

¿Por qué usar selectores?

¿Por qué usar selectores?

Anonymous · 17/09/2026

¿Por qué usar selectores?

En el mundo digital de hoy, la eficiencia y precisión en el manejo de datos son cruciales, especialmente cuando se trata de procesos tan importantes como la facturación electrónica. Aquí es donde entra en juego nuestra innovadora solución que combina la potencia de JSONPath con la flexibilidad de nuestros selectores personalizados.

¿Por qué esta combinación es una revolución en la manipulación de datos JSON? JSONPath por sí solo es una herramienta potente para extraer datos de estructuras JSON, permitiendo acceder a información compleja de manera sencilla y directa. Sin embargo, cuando se enfrenta a necesidades específicas de personalización y manejo de datos en contextos únicos, como la facturación electrónica, su alcance puede resultar limitado.

Aquí es donde nuestros selectores personalizados entran en juego, ofreciendo un nivel de personalización y flexibilidad inigualable. Al combinarlos con JSONPath, se desbloquean posibilidades ilimitadas para manipular, transformar y presentar datos de formas que antes eran imposibles o requerían un esfuerzo considerable de codificación.

Beneficios clave de esta combinación poderosa

1. Personalización Avanzada: Nuestros selectores personalizados permiten ajustes finos en la extracción y presentación de datos, lo que es crucial en procesos de facturación donde cada detalle cuenta.

2. Mayor Control y Precisión: Mientras JSONPath maneja la extracción de datos de manera eficiente, nuestros selectores añaden una capa adicional de control, permitiendo ajustes precisos y condiciones específicas.

3. Facilidad de Uso y Aprendizaje: Aunque JSONPath es simple, la integración con nuestros selectores hace que sea aún más accesible para usuarios no técnicos, permitiendo realizar tareas complejas sin necesidad de conocimientos profundos de programación.

4. Soluciones Creativas para Requisitos Únicos: La combinación de JSONPath con selectores personalizados abre un mundo de soluciones creativas para requisitos específicos que van más allá de la extracción de datos estándar.

5. Optimización de Procesos de Negocio: Esta herramienta no solo mejora la precisión en la facturación electrónica, sino que optimiza los procesos de negocio, ahorrando tiempo y recursos valiosos.

En resumen, al combinar JSONPath con nuestros selectores personalizados, no solo aprovechas las fortalezas de ambos sino que también abres la puerta a un mundo de posibilidades en la gestión de datos. Esta combinación no es simplemente una mejora; es una transformación en la forma en que interactúas y utilizas los datos en tu día a día, especialmente en procesos críticos como la facturación electrónica. ¡Descubre el poder de esta integración y lleva la gestión de tus datos a un nuevo nivel de eficiencia y efectividad!

Última actualización el 17/09/2026

Casos de uso

Anonymous · 17/09/2026

Casos de uso

Última actualización el 17/09/2026

Casos genéricos

Casos genéricos

Anonymous · 17/09/2026

Casos genéricos

Los JSON, y por lo tanto índices y valores que se verán en los siguientes casos, son solo variables de ejemplo para mostrar el funcionamiento de los diferentes selectores y sus combinaciones. No son casos reales de órdenes de cierto comercio electrónico. Es importante que cada usuario, usando esta documentación, adapte el selector a los índices de su JSON.

Valor en índice de primer nivel

Si tenemos el siguiente JSON:

```
{
    "k_simple": "v_simple"
}
```

Para obtener el valor del índice k_simple usamos el siguiente selector:

```
k_simple
```

Valor en índice de segundo nivel (anidado)

Se utiliza un “.” (punto) para ingresar a un nivel interior, o sea para entrar al objeto anidado.

Si tenemos el siguiente JSON:

```
{
    "k_nested1": {
        "k_nested2": "v_nested"
    }
}
```

Para obtener el valor del índice **k_nested2** usamos el siguiente selector:

```
k_nested1.k_nested2
```

Valor en índice de tercer nivel (anidado)

Este caso es idéntico al caso anterior. Se deja solo para ilustrar que se puede seguir agregando "." para ingresar a tantos niveles como sea necesario según el JSON que tengamos.

Si tenemos el siguiente JSON:

```
{
    "k_nested1p": {
        "k_nested2p": {
            "k_nested3p": "v_nested"
        }
    }
}
```

Para obtener el valor del índice **k_nested3p** usamos el siguiente selector:

```
k_nested1p.k_nested2p.k_nested3p
```

Valor en índice de un arreglo

Si tenemos el siguiente JSON:

```
{
    "array": [1, 2, 3]
}
```

Para obtener el valor del índice **1 del arreglo** (segundo elemento, ya que los arreglos comienzan desde el índice 0) usamos el siguiente selector:

```
array[1]
```

Valor en índice de un arreglo anidado

Este caso es una combinación de 2 casos vistos previamente:

- Valor en índice de segundo nivel (anidado).
- Valor en índice de un arreglo.

Si tenemos el siguiente JSON:

```
{
    "nested_array": {
        "array": [1, 2, 3]
    }
}
```

Para obtener el valor del índice 1 del arreglo usamos el siguiente selector:

```
nested_array.array[1]
```

Valor en índice asociado a otro índice de un arreglo de diccionarios

Este es un caso bastante interesante, aquí tenemos un arreglo con diccionarios y necesitamos elegir un valor desde uno de los diccionarios pero no sabes cuál es su posición dentro del arreglo. Debido a lo anterior, la alternativa es buscar en los diccionarios por un “filtro” (valor conocido) y elegir otro índice dentro del mismo diccionario (diccionario filtrado) como el valor real que necesitamos obtener.

Se usarán 2 índices para este selector:

- Primer índice, lo llamaremos key, tiene el valor que estamos buscando para filtrar el diccionario que elegiremos del arreglo.
- Segundo índice, lo llamaremos value, tiene el valor real que estamos buscando obtener.

Si tenemos el siguiente JSON:

```
{
    "mixed": [
        {
            "key": 10,
            "value": "hola"
        },
        {
            "key": 20,
            "value": "mundo"
        },
        {
            "key": 30,
            "value": "chao"
        },
    ]
}
```

Para obtener el valor del índice **value** asociado al índice **key** que tiene como valor **20** usamos el siguiente selector:

```
mixed[key=20:value]
```

El formato de este selector es el siguiente:

```
array[key=filtro:value]
```

Donde:

- **array:** es el selector del arreglo que deseamos usar.
- **key:** es el índice que usaremos para hacer el filtro.
- **filtro:** es el valor que buscamos que tenga el índice key.
- **value:** es el valor asociado al índice key que queremos recuperar para usar.

Estos 4 campos irán cambiando según el JSON (al igual que todos los índices y valores de estos casos de ejemplo).

Valor en índice asociado a otro índice de un arreglo de diccionarios que está anidado

Este caso es una combinación de 2 casos vistos previamente:

- Valor en índice de segundo nivel (anidado).
- Valor en índice asociado a otro índice de un arreglo de diccionarios.

Si tenemos el siguiente JSON:

```
{
    "nested_mixed": {
        "mixed": [
            {
                "key": 10,
                "value": "hola"
            },
        ]
    }
}
```

Para obtener el valor del índice **value** asociado al índice **key** que tiene como valor **10** usamos el siguiente selector:

```
nested_mixed.mixed[key=10:value]
```

Valor en índice anidado dentro de un índice asociado a otro índice de un arreglo de diccionarios

Este caso es una combinación de 2 casos vistos previamente:

- Valor en índice asociado a otro índice de un arreglo de diccionarios.
- Valor en índice de segundo nivel (anidado).

Importante

El orden de la lista anterior si importa. Se usan los mismos casos que el caso previo, pero en otro orden. Acá primero se filtra y luego se extrae el valor desde el índice anidado.

Si tenemos el siguiente JSON:

```
{
    "mixed_with_childs": [
        {
            "key": 20,
            "child": {
                "value": "hijo"
            }
        }
    ]
}
```

Para obtener el valor del índíce **value** que está anidado dentro del índice child asociado al índice **key** que tiene como valor **20** usamos el siguiente selector:

```
mixed_with_childs[key=20:child].value
```

Nota

Si en vez de seleccionar **value** usáramos este selector:

```
mixed_with_childs[key=20:child]
```

Obtendríamos lo siguiente:

```
{
    "value": "hijo"
}
```

Como se podría suponer, esto significa que obtendremos un objeto, en este caso un diccionario. Los

selectores permiten obtener objetos. Sin embargo, en las configuraciones que se realizan en BillMySales esto nunca se debe hacer. En las configuraciones de BillMySales siempre debemos llegar al valor escalar (no objeto), ya sea: un texto (string), un número o un valor booleano.

Última actualización el 17/09/2026

Añadir texto al resultado

Añadir texto al resultado

Anonymous · 17/09/2026

Añadir texto al resultado

Usando las cadenas literales y los selectores con paréntesis, tenemos 2 casos interesantes que nos permiten mejorar el formato de la salida de nuestro selector añadiendo texto y múltiples resultados concatenados.

Añadir texto al resultado del selector

Es posible añadir un texto fijo al resultado del selector. Esto se logra colocando el texto entre comillas dobles.

Por ejemplo, el siguiente selector siempre entregará el texto Hola Mundo:

```
"Hola Mundo"
```

Ya que al estar todo entre comillas dobles se interpreta como el texto tal cual está escrito y no como un selector que debe buscar datos en el JSON.

Para usar esto con un selector que extraiga datos, sólo se debe escribir el selector entre paréntesis acompañado del texto entre comillas dobles. Esto permite hacer cosas como la siguiente:

```
"Valor anidado es: "(k_nested1.k_nested2)
```

O la siguiente:

```
"$"(total)" CLP"
```

Concatenar selectores

Para concatenar selectores sólo se deben escribir entre paréntesis cada uno de ellos.

Ejemplos:

```
(k_simple)(k_nested1.k_nested2)
```

Sin embargo, es más interesante cuando lo unimos con lo anterior agregando un texto:

`"Simple es "(k_simple)"` y anidado es `"(k_nested1.k_nested2)`

Última actualización el 17/09/2026

Elegir valor a seleccionar

Elegir valor a seleccionar

Anonymous · 17/09/2026

Elegir valor a seleccionar

Gracias al operador OR de los selectores podemos elegir qué valor usar de una lista de selectores, tomando el primer elemento con valor (que no sea null). También podemos definir valores por defecto cuando un selector no entrega resultado (es null o vacío).

Seleccionar primer selector con un valor de una lista de selectores

Para poder separar diferentes selectores y elegir el primero que exista y tenga un valor se utiliza el OR con los símbolos “||” (doble barra vertical o doble pipe).

Si tenemos el siguiente JSON:

```
{
  "selector_k1": null,
  "selector_k3": "valor3",
  "selector_k4": "valor4",
}
```

Y usamos el siguiente selector:

`selector_k1||selector_k2||selector_k3||selector_k4`

Obtendremos el valor:

`valor3`

Porque es el primer valor encontrado que existe y no es null.

Valor por defecto de un selector

Este uso es una combinación de 2 usos vistos previamente:

- Añadir texto al resultado del selector.
- Seleccionar primer selector con un valor de una lista de selectores.

Si combinamos esos 2 usos podemos escribir el siguiente selector.

```
(selector)||"Valor por defecto"
```

Donde se buscará el valor asociado al **selector**, y si no existe, se pasará al siguiente elemento de la lista, que en este caso es un selector formado sólo por el texto **Valor por defecto**. Con esto, si el selector no tiene un valor, siempre, por el valor por defecto, se tendrá uno.

Última actualización el 17/09/2026

¡Ponte a prueba!

Anonymous · 17/09/2026

¡Ponte a prueba!

Última actualización el 17/09/2026

Autoevaluación

Anonymous · 17/09/2026

Autoevaluación

Autoevaluación sobre Selectores en BillMySales

Preguntas para evaluar la comprensión sobre el uso de selectores y lectura de datos JSON en BillMySales.

1. ¿Qué es un selector en el contexto de BillMySales?

- ☐ Un script para transformar JSON en XML
- ☐ Una cadena que describe cómo localizar un valor dentro de un JSON
- ☐ Un programa para crear archivos JSON
- ☐ Una base de datos para almacenar JSON

2. Los selectores en BillMySales sólo pueden acceder a valores simples, no a valores anidados dentro del JSON.

- ☐ True
- ☐ False

3. Si tienes este JSON: { "array": [1, 2, 3] }, ¿qué selector usarías para obtener el número 2?

- ☐ array[2]
- ☐ array(1)
- ☐ array[1]
- ☐ array.value[1]

4. Para obtener un valor dentro de un arreglo de objetos en JSON, ¿qué debe tener el selector?

- ☐ Un filtro que identifique la clave que buscas
- ☐ Sólo el índice del arreglo sin más detalles
- ☐ El nombre completo del archivo JSON
- ☐ Un código HTML que coincida con el valor

5. Si un selector devuelve un objeto JSON en vez de un valor escalar, ¿qué se debe hacer?

- ☐ Usar directamente el objeto en la configuración

- ☐ Modificar el selector para obtener un valor escalar, como un texto o número
- ☐ Ignorar el valor y dejar el campo vacío
- ☐ Pedir al sistema que convierta el objeto en texto automáticamente

6. Si un selector está escrito de manera incorrecta o no existe en el JSON utilizado para la extracción, el resultado será null.

- ☐ True
- ☐ False

7. ¿Qué tipo de valores deben devolver los selectores en las configuraciones de BillMySales?

- ☐ Objetos o diccionarios completos
- ☐ Solo valores escalares como texto, números o booleanos
- ☐ Listas vacías
- ☐ Archivos JSON completos

8. ¿Qué debe hacer un usuario si quiere que se agregue un selector para un campo que actualmente no es personalizable?

- ☐ Esperar a que el sistema lo agregue automáticamente
- ☐ Reescribir todo el JSON manualmente
- ☐ Contactar a BillMySales para evaluar la posibilidad de agregar el selector
- ☐ No es posible agregar nuevos selectores

Respuestas

1. ¿Qué es un selector en el contexto de BillMySales?

✓ Una cadena que describe cómo localizar un valor dentro de un JSON

Un selector es una cadena de texto que describe cómo localizar un valor específico dentro de una estructura JSON.

---

2. Los selectores en BillMySales sólo pueden acceder a valores simples, no a valores anidados dentro del JSON.

✓ False

Los selectores permiten seleccionar nodos anidados dentro de un JSON, no sólo valores simples.

---

3. Si tienes este JSON: { "array": [1, 2, 3] }, ¿qué selector usarías para obtener el número 2?

✓ array[1]

Los arreglos en JSON comienzan en el índice 0, por lo que el segundo elemento se accede con array[1].

---

4. Para obtener un valor dentro de un arreglo de objetos en JSON, ¿qué debe tener el selector?

✓ Un filtro que identifique la clave que buscas

El selector debe incluir un filtro para identificar el objeto correcto dentro del arreglo.

---

5. Si un selector devuelve un objeto JSON en vez de un valor escalar, ¿qué se debe hacer?

✓ Modificar el selector para obtener un valor escalar, como un texto o número

Se debe siempre obtener un valor escalar para usar en la configuración, no un objeto JSON.

---

6. Si un selector está escrito de manera incorrecta o no existe en el JSON utilizado para la extracción, el resultado será null.

✓ True

Cuando un selector no existe o está mal escrito, no se obtiene ningún dato y el sistema devuelve null para representar que no hay coincidencia.

---

7. ¿Qué tipo de valores deben devolver los selectores en las configuraciones de BillMySales?

✓ Solo valores escalares como texto, números o booleanos

En BillMySales, los selectores deben devolver siempre valores escalares, no objetos.

---

8. ¿Qué debe hacer un usuario si quiere que se agregue un selector para un campo que

actualmente no es personalizable?

✓ Contactar a BillMySales para evaluar la posibilidad de agregar el selector

El usuario debe contactar a BillMySales para evaluar la adición de nuevos selectores.

Última actualización el 17/09/2026