Universidad de Buenos Aires

Facultad de Ciencias Exactas y Naturales

Departamento de Computación

JSON Schema

Author:
Gerardo Rossel

2020

Contents

1 Introducción

En este apunte daremos una breve descripción de como especificar esquemas de documentos JSON (JavaScript Object Notation). Cómo para el diseño de bases de datos basadas en documentos, en la materia usaremos el formato JSON (independientemente que luego sean almacenados como BSON) entonces *JSON Schema* será la herramienta a usar para especificarlos.

JSON Schema es una herramienta muy útil para la validación de documentos JSON pero también puede ser utilizada para especificar la estructura de los documentos que se diseñan.

1.1 Estructura de JSON

JSON está basado en las siguientes estructuras:

object Una lista de pares clave/valor

```
{ "clave1": "valor1",
  "clave2": "valor2" }
```

Array : Un vector de valores

```
[ "first", "second", "third" ]
```

number : un número incluidos enteros, decimales, etc.

```
45
```

string : cadena de caracteres

```
"Es una cadena de caracteres"
```

boolean : valor lógico: verdadero o falso.

```
true, false
```

null : el valor nulo.

```
null
```

1.2 Ejemplo JSON

Veamos un documento JSON que describe un cliente:

```
{
 "nombre": "Juan",
 "apellido": "Doe",
 "fecha_nacimiento": "01-01-1990",
 "direccion": {
        "calle": "Av. San Martin 2234 1P",
        "ciudad": "CABA",
        "pais": "Argentina"
    }
}
```

El documento es un *object* que cuenta con propiedades. El nombre, el apellido y la fecha de nacimiento son propiedades del documento de tipo cadena de caracteres (string), la dirección a su vez es un documento embebido (otro *object*) que a su vez cuenta con tres propiedades: calle, ciudad y país. Podría haberse elegido otra estructura diferente para el cliente, donde por ejemplo nombre y apellido sea en realidad un *object* con dos propiedades o la dirección sea un simple *string* con todo el texto de la misma.
Claramente si queremos especificar la estructura anterior debemos describir que el documento debe cumplir con determinadas restricciones que establezcan como se representa la información y con que tipo de valores.

1.3 JSON Schema: definición

Un *JSON Schema* es un documento JSON que describe la estructura de otro documento. Para el ejemplo del cliente el *JSON Schema* sería:

```
{
"type": "object",
"properties": {
 "nombre": {"type":"string" },
 "apellido": {"type":"string" },
 "fecha_nacimiento":{"type":"string","format":"date-time"},
```

```
 "direccion":{
     "type":"object",
         "properties":{
         "calle":{"type":"string" },
         "ciudad":{"type":"string" },
         "pais":{"type":"string" }
         }
        }
  }
}
```

Como se ve un *JSON Schema* es una descripción declarativa de la estructura de otro documento JSON.

2 JSON Schema

2.1 Palabra Clave ”type”

La palabra clave *type* permite especificar el tipo de los datos de un esquema. Los tipos principales son: *string*, numericos, *object*, *boolean*, *array* y *null*.

Al definir el tipo de una propiedad se utiliza la palabra clave *”type”* la cual puede ser seguida un *string* que representa al tipo o de un arreglo de *strings*. La utilización de un arreglo de nombres de tipos permite que una propiedad puede asumir diferentes tipos.

Por ejemplo:

```
"nombre": {"type":"string" }
```

Indica que la propiedad *nombre* puede tener valores de tipo *string* solamente. Mientras que:

```
 "codigo":{"type":["number", "string"] }
```

nos está indicando que el código puede ser un número o un *string* pero no puede ser, por ejemplo, un *object* o un *array*.

2.2 Metadata: title, description y default

JSON Schema tiene algunas palabras que si bien no se utilizan para especificar el documentos si sirven a nivel descriptivo.

```
{
 "title" : "Esquema vacio",
 "description" : "Un esquema sin nada adentro",
 "default" : "Algun valor por defecto"
}
```

Tanto *title* como *description* deben ser de tipo *string* y su significado es auto descriptivo. En particular *description* es utilizado para comentar o dar significado a una propiedad:

```
"codigo":{
        "description": "El identificador de un producto",
        "type": "string"
         }
```

2.3 Tipos

En esta sección describiremos brevemente los diferentes tipos utilizados en JSON Schema y las palabras claves asociadas a ellos.

2.3.1 El tipo string

El tipo *string* se refiere a cadenas de caracteres undicode. Una propiedad de tipo *string* puede tener como valor cualquier cadena de caracteres. Para poder limitar los valores posibles existen otras palabras claves asociadas a string.

Para establecer restricciones respecto a la longitud de un string se utilizan las palabras claves: *minLength* y *maxLength*.

```
"codigo":{"type": "string", "minLength": 5,"maxLength": 15 }
```

El ejemplo anterior indica que código es un *string* con una longitud mínima de 5 caracteres y una máxima de 15.

También es posible establecer que un *string* debe cumplir con una expresión regular con la sintáxis de las expresiones regulares de *JavaScript*. Se utiliza la palabra clave *pattern* como se ve en el siguiente código:

```
{
"email" : {"type": "string", "pattern":"^\w+@[a-zA-Z_
   ]+?\.[a-zA-Z]{2,3}$" }
}
```

Es importante notar el uso de "∧ " al comienzo y de "$" al final porque sino la expresión regular se evalúa si aparece en cualquier parte de la cadena. La expresión regular del ejemplo permite establecer el formato de un correo electrónico, pero para ello JSON Schema provee un método más simple para establecer la validación semántica de *strings* y es utilizar la palabra clave *format*. Hay un grupo de formatos preestablecidos:

"date-time" Fecha según el RFC 3339

"email" correo electrónico

"hostname" Nombre de host de internet según el RFC 1034

"ipv4" Dirección IP versión 4 o IPv4

"ipv6" Dirección IP versión 6 o IPv

"uri" Identificador universal de recurso (URI) según el RFC3986.

Utilizando *format* el ejemplo del correo electrónico puede escribirse:

```
{
"email" : {"type": "string", "format":"email" }
}
```

2.3.2 Números

Para tipos númericos puede usarse *integer* para valores enteros o *number* para otros.

```
{
"edad" : {"type": "integer"},
"sueldo":{"type": "number"}
}
```

El tipo *integer* no es un tipo propio de JavaScript y por ende de JSON, por lo cual puede provocar confusión. Desde el punto de vista de la especificación puede quedar más o menos claro qué entendemos por *integer* aunque un documento puede tener 1 y 1.0 y ambos ser considerados válidos. El tipo *number* indica cualquier número incluso en notación exponencial.

Como en el caso de los *strings*, los números puede ser restringidos. Para restringir números hay dos formas establecer: múltiplos o rangos.

La palabra clave *multipleOf* es utilizada para indicar que los números permitidos son únicamente los múltiplos del valor especificado.

```
{
"edad":{"type":"number","multipleOf":1.0},
}
```

En el ejemplo se utiliza *multipleOf* para restringir a enteros, similar a usar *integer* como tipo

Los rangos se establecen como mínimo y máximo mediante las palabras claves *minimum*, *maximum*. Si no se especifica lo contrario el valor mínimo debe ser *menor o igual* al valor de la propiedad y el valor máximo *mayor o igual*. Para una desigualdad estricta se utiliza:*exclusiveMinimum* y/o *exclusiveMaximum* ambas son valores lógicos (booleanos).

```
{
"edad" : {"type": "integer"},
"sueldo":{"type": "number", "minimum":7800, "maximum":
   78000, "exclusiveMinimum": true}
}
```

En este caso estamos diciendo que el sueldo cumple con:
$7800 < sueldo \leq 78000$

2.3.3 object

Este tipo se utiliza para declarar documentos y documentos embebidos. Para el ejemplo de la sección 1.3 tenemos el documento principal que representa a un cliente y un documento anidado que representa a la dirección

```
{
 "type": "object",
 "properties":{
  "nombre":{"type":"string" },
  "apellido":{"type":"string" },
  "fecha_nacimiento":{"type":"string","format":"date-time"},
  "direccion":{
    "type":"object",
    "properties":{
       "calle":{"type":"string" },
       "ciudad":{"type":"string" },
       "pais":{"type":"string" }
      }
    }
  }
}
```

El tipo *object* tiene un conjunto de propiedades (*properties*) que son definiciones de pares clave-valor. Cada propiedad a su vez tiene su propio tipo que puede ser otro *object*. Las propiedades de un *object* no son necesariamente exhaustivas, un documento con una propiedad extra también es válido a no ser que se especifique lo contrario. Para evitar propiedades adicionales se usa *additionalProperties* con el valor *false*. *additionalProperties* es un objeto que además puede establecer condiciones sobre las propiedades adicionales. Por ejemplo en dirección podríamos permitir agregar propiedades sólo de tipo *string*.

```
{
"direccion":{
 "type":"object",
 "properties":{
   "calle":{"type":"string" },
```

```
    "ciudad":{"type":"string" },
    "pais":{"type":"string" }
    },
   "additionalProperties": { "type": "string" }
   }
}
```

En caso necesario puede indicarse cuales propiedades son *obligatorias* para que el documentos sea válido. Para ello se utiliza la palabra clave *required* que toma un arreglo de *strings* donde cada *string* es el nombre de la propiedad que se quiere que sea obligatoria.

```
{
 "direccion":{
  "type":"object",
  "properties":{
    "calle":{"type":"string" },
    "ciudad":{"type":"string" },
    "pais":{"type":"string" }
   },
  "additionalProperties": { "type": "string" } ,
  "required":["calle", "ciudad", "pais"]
 }
}
```

Al tipo *object* además se le pueden limitar la cantidad de propiedades que pueda contener usando *minProperties* y *maxProperties* asignándoles la cantidad mínima y máxima de propiedades respectivamente.

En algunas ocasiones una propiedad puede depender de otra. Por ejemplo podríamos indicar que si una persona tiene tipo de documento entonces debe tener número, de tal modo que el documento que contenga tipo y no número sea inválido.

```
{
 "type": "object",
 "properties":{
    "nombre":{"type": "string"},
```

```
    "apellido":{"type": "string"},
    "tipo_documento" : {"type": "string"},
    "numero":{"type": "number"}
   }
   "required":["nombre", "apellido"],
   "dependencies": {"tipo_documento": ["numero"]}

}
```

La palabra clave *dependencies* indica un objeto donde cada par clave-valor es un mapeo entre una propiedad "p" y un arreglo de *strings* con los nombres de las propiedades obligatorias cuando "p" está presente. En el ejemplo, cuando está presente la propiedad *"tipo_documento"* entonces debe estar presente la propiedad *"numero"*.

Además de las dependencias de propiedades puede haber una extensión del esquema definiendo en dependencias el esquema requerido con sus restricciones. El ejemplo anterior se haría de la siguiente manera:

```
{
 "type": "object",
 "properties":{
        "nombre":{"type": "string"},
        "apellido":{"type": "string"},
        "tipo_documento" : {"type": "string"},
        }
  "required":["nombre", "apellido"],
  "dependencies": {
     "tipo_documento":{
         "properties":{
           "numero": { "type": "number" }
          },
         "required": ["numero"]
      }
  }
}
```

Es decir estoy definiendo un esquema con propiedades y restricciones que se requieren cuando está presente alguna propiedad. Notar que en el ejemplo *"numero"* no se encuentra como propiedad en el documento principal.

2.3.4 El tipo array

Si bien un *array* puede contener cualquier lista en muchos casos es necesario especificar restricciones sobre los ítems del mismo. Para ello se utiliza la palabra clave *items*. Por ejemplo si quisiéramos que todos los elementos de un *array* sean de tipo numérico deberíamos escribir algo así

```
"edades":{"type":"array","items":{"type":"integer"}}
```

En ese casos estamos especificando que la propiedad *edades* es una lista de enteros. Atención que un arreglo vacío siempre es válido. Supongamos que se quiere guardar en un *array* la dirección de una persona en lugar de guardarla en un documento como en ejemplos anteriores. En ese tipo de situaciones puede ocurrir que no todos los elementos del array sean del mismo tipo y además puede pasar que no querramos ítems extras mas allá de los que definimos. Eso podemos especificarlo de la siguiente manera:

```
"direccion": {
    "type": "array",
    "items": [
     {"type": "string"},
     {"type": "number"},
     {"type": "string"},
     {"type": "string"}
     ],
     "additionalItems": false
}
```

En el ejemplo el primer elemento del *array* es un string, el segundo un numero y el tercero y cuarto son string también. Además no se admiten más items porque figura *"additionalItems": false* El tamaño máximo y mínimo de un *array* puede establecerse especificando a *minItems* y *maxItems*. Un arreglo que tenga entre 8 y 10 elementos se especifica así:

```
"integrantes": {
        "type": "array",
        "minItems":8,
        "maxItems":10
}
```

La propiedad *"integrantes"* es un arreglo con un mínimo de 8 y un máximo de 10. Pero además podríamos indicar que cada elemento debe ser único utilizando la palabra clave *uniqueItems*

```
"integrantes": {
        "type": "array",
        "minItems":8,
        "maxItems":10,
        "uniqueItems": true
}
```

2.3.5 El tipo boolean y el tipo null

El tipo *boolean* evalúa a verdadero o falso que se escribe: *true* o *false* sin comillas. Ejemplo:

```
"es_administrador": {"type": "boolean"}
```

Cuando en un esquema se establece que una propiedad es de tipo *null* el único valor aceptado es justamente *null*

```
"sin_valor": {"type": "null"}
```

2.4 Enumerados

En el caso de necesitar restringir los valores posibles de una propiedad a un conjunto fijo de valores se puede utilizar la palabra clave *enum* en la declaración del tipo. Por ejemplo:

```
 "tipo_iva": {"type": "string"
 "enum":["R.I","Montributista","Exento","Consumidor
    Final"] }
```

El *enum* recibe un arreglo de los valores posibles de la propiedad. En el ejemplo *"tipo_iva"* es una propiedad de tipo *string* por lo cual lo valores de *enum* deberían ser todos *string*. Si se incluye un valor que no sea *string* será ignorado en la validación. Por ejemplo si tenemos:

```
"tipo_iva": {"type": "string"
"enum":["R.I","Montributista","Excento","Consumidor
   Final", null] }
```

no puede asignarse *null* a *"tipo_iva"* porque el tipo declarado es *string* por más que figura en la lista.

Es posible, sin embargo, declarar una propiedad enumerada sin definir su tipo de tal manera que acepte valores de diferentes tipos.

```
"tipo_iva": {"enum":["R.I","Montributista","Excento","
   Consumidor Final", null] }
```