CSCI235 Database Systems

JSON Schema

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

JSON Schema

Outline

JSON Schema ? What is it ?

JSON Schema is a vocabulary for validation of JSON objects

JSON Schema describes the structures and types of existing JSON objects

JSON Schema allows for automated validation of JSON objects

JSON Schema itself is a JSON object

The latest version of JSON Schema is a version 2020-12

The following JSON object

JSON object

```
{"city":"Sydney", "street":"Victoria", "building number":25, "building name":"James Bond Building"}
```

validates well against the following JSON Schema

JSON schema

```
{ "type":"object",
  "properties": { "city": { "type":"string" },
                  "street": { "type":"string" },
                  "building": { "type" :"number",
                                "minimum": 1}
                }
}
```

JSON Schema

Outline

Basics

The simplest possible JSON schema is the following (an empty object)

The simplest JSON schema

```
{ }
```

The schema given above validates every JSON object

Extensions of the schema given above impose the restrictions on the validated objects

For example, the following JSON schema

A Hello world JSON schema

```
{ "type":"object",
  "properties": { "Hello world message": { "type":"string" } }
}
```

validates well the objects

A Hello world object

```
{ "Hello world message":"Hello world !" }
```

A How are you object

```
{ "Hello world message":"Hello world !", "Greeting":"How are you ?" }
```

An empty object

```
{}
```

Basics

The following extension of the previous schema

An extended Hello world JSON schema

```
{ "type":"object",
  "properties": { "Hello world message": { "type":"string",
                                           "minLength": 1,
                                           "maxLength": 13 }
                }
}
```

fails validation of an object

A Hello world object

```
{ "Hello world message":"Hello world !!!" }
```

because a string `"Hello world !!!"` is too long (15 characters)

It validates well the objects

A Hello world object

```
{ "Hello world message":"Hello world !" }
```

A How are you object

```
{ "Hello world message":"Hello world !", "Greeting":"How are you ?" }
```

JSON Schema

Outline

Types

In JSON schema a `type` keyword is associated with a data type

JSON schema defines the following basic types: `string`, `Numeric types`, `object`, `array`, `boolean`, and `null`

A basic type associated with a keyword `type` can be either one of the types listed above or it can be an array of the types listed above

If `type` keyword is associated with a name of a basic type then a respective value must be of the same basic type, for example

```
{ "type":"string"}
```
Type string

It determines a type of a respective value as a `string`

If `type` keyword is associted with an array of basic types then each element of the array must be unique, for example

```
{ "type": ["string","number"] }
```
Type string or number

It determines a type of a respective value as either `string` or `number`

JSON Schema

Outline

String

A type `string` determines a type of a respective value as a string of characters

For example

String type

```
"city":{ "type":"string" }
```

validates well any string of characters including an empty string associated with a key `"city"`

A `string` type has the following restrictions: `minLength`, `maxLength`, `pattern`, and `format`

For example

String type with length restrictions

```
"city":{"type":"string",
        "minLength": 10,
        "maxLength": 20 }
```

validates well any string longer than 9 characters and shorter than 21 characters associated with a key `"city"`

String

A `pattern` restriction is used to match a string with a given regular expression

For example

String type with regular expression restriction

```
"number":{"type":"string",
          "pattern":"[1-9][0-9]" }
```

validates well any string of characters representing a number in a range from `"10"` to `"99"` with a key `number`

A `format` restriction is used to match a string with a predefined format

For example

String type with format restriction

```
"birthday":{"type":"string",
            "format":"date" }
```

validates well any string of characters representing a date in a format `"YYYY-MM-DD"` for example `"2020-07-21"` with a key `birthday`

JSON Schema

Outline

Numeric types

JSON Schema has two numeric types: `integer` and `number`

A type `integer` is used for validation of integer numbers

For example

Integer type

```
"int":{ "type":"integer" }
```

validates well any integer number like `-5`, `0`, `7`, etc with key `int`

A type `number` is used for validation of any numeric value, either integer or floating point numbers

For example

Number type

```
"num":{ "type":"number" }
```

validates well any integer or floating point number like like `-35.7`, `-7`, `0.0`, `23`, `23.4`, etc with a key `num`

Numeric types

Numeric types `integer` and `number` have the following restrictions: `multipleOf`, `minimum`, `exclusive Minimum`, `maximum`, `exclusiveMaximum`

For example

Integer type with multipleOf restriction

```
"mod5zero":{ "type":"integer",
             "multipleOf": 5 }
```

validates well any integer like `0`, `5`, `10`, `15`, etc, with a key `mod5zero`

For example

Number type with multipleOf restriction

```
"mof2":{ "type":"number",
         "multipleOf": 0.2 }
```

validates well any number like `0.0`, `0.2`, `0.4`, `0.6`, etc, associated with a key `mof2`

Numeric types

The restrictions `minimum`, `maximum`, `exclusiveMinimum`, and `exclusiveMaximum` can be used for expressing ranges

For example

Integer type with a range restriction

```
"intrange":{ "type":"integer",
             "minimum": 0,
             "exclusiveMaximum": 5}
```

validates well integer numbers `0`, `1`, `2`, `3`, and `4` associated with a key `intrange`

JSON Schema

Outline

Boolean

A type `boolean` determines a type of a respective value either as `true` or `false`

For example

Boolean type

```
"bool":{ "type":"boolean" }
```

validates well the values `true` and `false` and no other values associated with a key `bool`

JSON Schema

Outline

Null

A type `null` determines a type of a respective value as `null`

For example

Null type

```
"nothing":{ "type":"null" }
```

validates well a lack of value `null` and no other values associated with key `nothing`

JSON Schema

Outline

Object

A type `object` determines a type of a respective value as JSON object

For example

Object type

```
{ "type":"object" }
```

validates well the objects like

Flat JSON object

```
{ "city":"Sydney", "street":"Victoria", "building":25 }
```

Nested JSON object

```
{ "name":"School of Astronomy",
  "courses":[ {"code":"SOA101",
               "title":"Astronomy for Kids",
               "credits":3},
              {"code":"SOA201",
               "title":"Black Holes",
              "credits":6}
            ]
}
```

Empty JSON object

```
{ }
```

Object

A key `properties` can be used to impose the constraints on some or all of key:value pairs an object consists of

A key `properties` is associated with an object that consists of key:value pairs where each key is a name of a property and each value is a JSON schema used to validate a property

For example JSON schema

JSON schema
```
{ "type":"object",
  "properties": { "city": { "type":"string" },
                  "street": { "type":"string" },
                  "building": { "type":"number", "minimum":1 }
                }
}
```

validates well the objects

JSON object
```
{ "city":"Dapto","street":"Station","building":7}
```

JSON object
```
{ "city":"Dapto","street":"Station"}
```

JSON object
```
{ "city":"Dapto","street":"Station","building":7,"type":"skyscraper"}
```

Empty JSON object
```
{ }
```

Object

A key `additionalProperties` determines whether additional properties not listed in JSON schema are allowed

For example JSON schema

JSON schema
```
{ "type":"object",
  "properties": { "city": { "type":"string" },
                  "street": { "type":"string" },
                  "building": { "type":"number",
                                "minimum":1 }
                },
  "additionalProperties":false
}
```

validates well the objects

JSON object
```
{ "city":"Dapto","street":"Station","building":7}
```

JSON object
```
{ "city":"Dapto","street":"Station"}
```

Empty JSON object
```
{ }
```

It does not validate well an object

JSON object
```
{ "city":"Dapto","street":"Station","building":7,"type":"skyscraper"}
```

Object

A key `requiredProperties` determines the compulsory properties of an object

For example JSON schema

JSON schema
```
{ "type":"object",
  "properties": { "city": { "type":"string" },
                  "street": { "type":"string" },
                  "building": { "type":"number","minimum":1 }
                },
  "requiredProperties": ["city","street"]
}
```

validates well the objects

JSON object
```
{ "city":"Dapto","street":"Station","type":"skyscraper"}
```

JSON object
```
{ "city":"Dapto","street":"Station","building":7}
```

It does not validate well an object

JSON object
```
{ "city":"Dapto","building":7,"type":"skyscraper"}
```

Object

The keys `minProperties` and `maxProperties` determine the minimum and maximum number of properties of an object

For example JSON schema

JSON schema
```
{ "type":"object",
  "properties": { "city": { "type":"string" },
                  "street": { "type":"string" },
                  "building": { "type":"number","minimum":1 }
                },
  "minProperties": 2,
  "maxProperties": 3
}
```

validates well the objects

JSON object
```
{ "city":"Dapto","street":"Station","building":7}
```

JSON object
```
{ "city":"Dapto","street":"Station"}
```

JSON object
```
{ "street":"Station","building":7}
```

It does not validate well an object

JSON object
```
{ "city":"Dapto","street":"Station","building":7,"type":"skyscraper"}
```

Object

A key `dependencies` determine the existence dependencies of one property on another

A value associated with a key `dependencies` is an object

Each entry in the object maps from a name of a property into an array of properties that are required whenever the property is present

For example JSON schema

JSON schema

```
{ "type":"object",
  "properties": { "city": { "type":"string" },
                  "street": { "type":"string" },
                  "building": { "type":"number",
                                "minimum":1 }
                },
  "dependencies": { "street": ["building"]} }
```

validates well the objects

JSON object

```
{ "city":"Dapto","street":"Station","building":7}
```

JSON object

```
{ "street":"Station","building":7}
```

JSON object

```
{ "city":"Dapto","building":7}
```

Object

A key `patternProperties` determines a syntax of key names in an object

A value associated with a key `patternProperties` is an object

For example JSON schema

JSON schema
```
{ "type":"object",
  "patternProperties": { "^(city|CITY)": { "type":"string" },
                         "^(street|STREET)": { "type":"string" },
                         "^(building|BUILDING)": { "type":"number",
                                                   "minimum":1 }
                       },
  "additionalProperties":false }
}
```

validates well the objects

JSON object
```
{ "city":"Dapto","STREET":"Station","building":7}
```

JSON object
```
{ "street":"Station","BUILDING":7}
```

JSON object
```
{ "CITY":"Dapto","BUILDING":7}
```

JSON object
```
{ }
```

JSON Schema

Outline

Array

A type `array` determines a type of a respective value as array

For example

Array type

```
{ "type":"array" }
```

validates well any array, like for example

Array of numbers

```
[1, 2, 3, 4, 5, 6, 7]
```

Array of strings

```
["ab", "cde", "efgh", "ijklm"]
```

Array of arrays

```
[ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
```

Empty array

```
[ ]
```

Array

A key `items` determines a type of values in an array

For example

Array type

```
{ "type":"array",
  "items": { "type":"string" }
}
```

validates well any array of strings like

Array of strings

```
["ab", "cde", "ab", "ijklm"]
```

Array of strings

```
["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
```

Empty array

```
[ ]
```

Array

Tuple validation determines a type of values when an array is a collection of items, each item has a different schema, and position of each item is important

For example

Array type with tuple validation

```
{ "type":"array",
  "items": [ { "type":"string" },
             { "type":"number",
               "minimum":1} ],
  "additionalItems":false
}
```

validates well any array of values like

A tuple

```
["Station St.", 25]
```

Incomplete tuple

```
["Victoria St."]
```

Empty tuple

```
[ ]
```

Array

A key `uniqueItems` requires each item in an array to be unique

For example

Array type

```
{ "type":"array",
  "items": { "type":"string" },
  "uniqueItems":true
}
```

validates well any array of strings like

Array of strings

```
["ab", "cde", "efgh", "ijklm"]
```

Array of strings

```
[ "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" ]
```

Empty array

```
[ ]
```

Array

The keys `minItems` and `maxItems` determine a size of an array

For example

Array type

```
{ "type":"array",
  "items": { "type":"string" },
  "uniqueItems":true,
  "minItems":3,
  "maxItems":5
}
```

validates well any array of strings like

Array of strings

```
["ab", "cde", "efgh", "ijklm"]
```

Array of strings

```
[ "0", "1", "2", "3", "4" ]
```

JSON Schema

Outline

Combined schemas

The keys `anyOf`, `allOf` and `oneOf` can be used to combine JSON Schemas together

For example `anyOf` keyword can be used to validate JSON object against one or more of the given schemas

Validation with anyOf

```
{ "anyOf": [ {"type":"string",
              "maxLength":5},
             {"type":"boolean"} ]
}
```

validates well the objects like strings up to 5 characters long or Boolean values `true` or `false`

Combined schemas

For example `anyOf` keyword can be used to validate JSON object against one or more of the given schemas

Validation with anyOf

```
{ "anyOf": [ {"type":"string",
              "maxLength":5},
             {"type":"string",
              "minLength":3} ]
}
```

validates well any string

Combined schemas

The keys `anyOf`, `allOf` and `oneOf` can be used to combine JSON Schemas together

For example `allOf` keyword can be used to validate JSON object against all of the given schemas

Validation with allOf

```
{ "allOf": [ {"type":"string",
              "maxLength":5},
             {"type":"string",
              "minLength":3} ]
}
```

validates well any string no longer than 5 characters and no shorter than 3 characters

Combined schemas

The keys `anyOf`, `allOf` and `oneOf` can be used to combine JSON Schemas together

For example `oneOf` keyword can be used to validate JSON object against exactly one of the given schemas

Validation with oneOf

```
{ "oneOf": [ {"type":"string",
              "maxLength":5},
             {"type":"string",
              "minLength":3} ]
}
```

validates well either any string longer than 5 characters or shorter than 3 characters

References

JSON Schema

Understanding JSON Schema

MongoDB - JSON Schema validation

MongoDB - $jsonSchema operator