NOSQL

JSON

Présentation

Rémy Lesiak

Architecte

15 ans d’expérience

Worldline

remy.lesiak@worldline.com

Au programme

JSON

MONGODB

NOSQL

ELASTICSEARCH

3 Cours de 4 heures
1 Cours de 1 heure

4 TP de 2 heures

Au programme

JSON

MONGODB

NOSQL

ELASTICSEARCH

Les formats de données structurées

Les formats de données structurées

CSV, XML, JSON

- ✓ **CSV** :
  - Comma Separated Value
  - Première utilisation 1972
  - RFC (Request For Comments) publiée en 2005

- ✓ **XML** :
  - eXtensible Markup Language
  - 1999
  - W3C

- ✓ **JSON** :
  - JavaScript Object Notation
  - Première utilisation : 2005 (Yahoo, Google)
  - Première spécification : 2013

Les formats de données structurées

CSV

- Format d'échange et de stockage de données
- Utilisé et rendu populaire par Microsoft Excel
- Lisibilité et exploitabilité humaine facile
- Structure de fichier très simple :
  - Header pour la dénomination des colonnes
  - Données organisées par ligne
  - Données séparés par une virgule
  - Retour à la ligne pour une nouvelle entrée

- Se rapproche des bases de données SQL
- Très utilisé pour des échanges de fichiers entre système d'information
- Utilisé pour des exports de données vers des utilisateurs

Les formats de données structurées

csv

Prenom	Nom	Email	Age	Ville
Robert	Lepingre	bobby@exemple.com	41	Paris
Jeanne	Ducoux	jeanne@exemple.com	32	Marseille
Pierre	Lenfant	pierre@exemple.com	23	Renne

Version Tableau

Version texte

Prenom, Nom, Email, Age, Ville
Robert, Lepingre, bobby@exemple.com, 41, Paris
Jeanne, Ducoux, jeanne@exemple.com, 32, Marseille
Pierre, Lenfant, pierre@exemple.com, 23, Rennes

Les formats de données structurées

XML

- Format d'échange de données introduit principalement avec les Webservices (SOAP)
- Types et format de données :
  - Chaine de caractère
  - Date
  - Nombre
  - …
- Structuration complexe des données :
  - Liste
  - Sous objet
  - Enumaration
  - …
- Correspond à une représentation objet de la donnée
- Schéma de validation des données : XSD
- Permet de valider les formats de données, les structures de données, …
- Données séparées par des balises ouvrantes/fermantes : *<data> azerty </data>*
- Format très verbeux

Les formats de données structurées

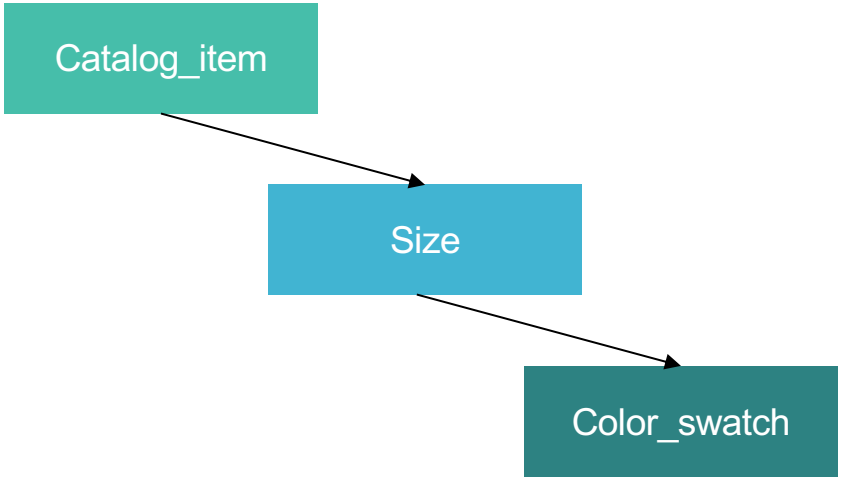
XML

Prenom	Nom	Email	Âge	Ville
Robert	Lepingre	bobby@exemple.com	41	Paris
Jeanne	Ducoux	jeanne@exemple.com	32	Marseille
Pierre	Lenfant	pierre@exemple.com	23	Renne

```
<personnes>
    <personne>
        <prenom>Robert</prenom>
        <nom>Lepingre</nom>
        <email>bobby@exemple.com</email>
        <age>41</age>
        <ville>Paris</ville>
    </personne>
    <personne>
        <prenom>Jeanne</prenom>
        <nom>Ducoux</nom>
        <email>jeanne@exemple.com</email>
        <age>32</age>
        <ville>Marseille</ville>
    </personne>
    <personne>
        <prenom>Pierre</prenom>
        <nom>Lenfant</nom>
        <email>pierre@exemple.com</email>
        <age>23</age>
        <ville>Rennes</ville>
    </personne>
</personnes>
```

Les formats de données structurées

XML




```
<?xml version="1.0"?>
<catalog>
   <product description="Cardigan Sweater" product_image="cardigan.jpg">
      <catalog_item gender="Men's">
         <item_number>QWZ5671</item_number>
         <price>39.95</price>
         <size description="Medium">
            <color_swatch image="red_cardigan.jpg">Red</color_swatch>
            <color_swatch image="burgundy_cardigan.jpg">Burgundy</color_swatch>
         </size>
         <size description="Large">
            <color_swatch image="red_cardigan.jpg">Red</color_swatch>
            <color_swatch image="burgundy_cardigan.jpg">Burgundy</color_swatch>
         </size>
      </catalog_item>
      <catalog_item gender="Women's">
         <item_number>RRX9856</item_number>
         <price>42.50</price>
         <size description="Small">
            <color_swatch image="red_cardigan.jpg">Red</color_swatch>
            <color_swatch image="navy_cardigan.jpg">Navy</color_swatch>
            <color_swatch image="burgundy_cardigan.jpg">Burgundy</color_swatch>
         </size>
         <size description="Medium">
            <color_swatch image="red_cardigan.jpg">Red</color_swatch>
            <color_swatch image="navy_cardigan.jpg">Navy</color_swatch>
            <color_swatch image="burgundy_cardigan.jpg">Burgundy</color_swatch>
            <color_swatch image="black_cardigan.jpg">Black</color_swatch>
         </size>
         <size description="Large">
            <color_swatch image="navy_cardigan.jpg">Navy</color_swatch>
            <color_swatch image="black_cardigan.jpg">Black</color_swatch>
         </size>
         <size description="Extra Large">
            <color_swatch image="burgundy_cardigan.jpg">Burgundy</color_swatch>
            <color_swatch image="black_cardigan.jpg">Black</color_swatch>
         </size>
      </catalog_item>
   </product>
</catalog>
```

JSON

Les formats de données structurées

- Volonté d'alléger et de simplifier les formats d'échange
- Il est le cousin du XML.
- Format d'échange de données introduit principalement avec les Webservices (REST)
- Structuration complexe des données : gestion de liste, de sous-objets
- Correspond à une représentation objet de la donnée
- Schéma de validation des données : Json Schema
- Possibilité de préciser les types et formats de données : chaîne de caractère, date, nombre, …
- Organisé sous forme de clé/valeur principalement : {**"propriété" : "valeur"**}

Les formats de données structurées

JSON

Prenom	Nom	Email	Âge	Ville
Robert	Lepingre	bobby@exemple.com	41	Paris
Jeanne	Ducoux	jeanne@exemple.com	32	Marseille
Pierre	Lenfant	pierre@exemple.com	23	Renne


```
{
"personnes": [
    {
      "prenom": "Robert",
      "nom": "Lepingre",
      "email": "bobby@exemple.com",
      "age": "41",
      "ville": "Paris"
    },
    {
      "prenom": "Jeanne",
      "nom": "Ducoux",
      "email": "jeanne@exemple.com",
      "age": "32",
      "ville": "Marseille"
    },
    {
      "prenom": "Pierre",
      "nom": "Lenfant",
      "email": "pierre@exemple.com",
      "age": "23",
      "ville": "Rennes"
    }
  ]
}
```

JSON

JSON

La syntaxe

- Accolade ouvrante : { = Début d'un objet
- Accolade fermante : } = Fin d'un objet

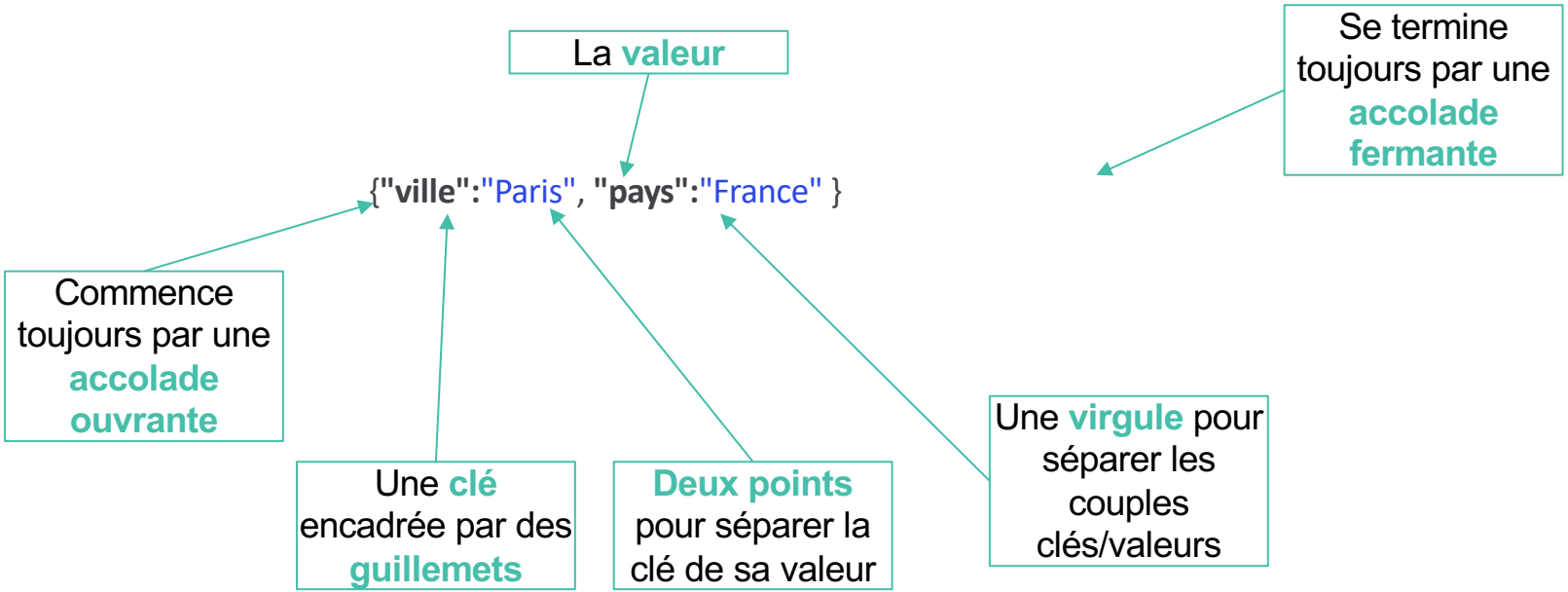
- Guillemet : " = Pour tous les textes clé et valeur
- Virgule : , = Séparation des champs

- Crochet ouvrant : [= Début d'une liste
- Crocher fermant :] = Fin d'une liste

```
{
  "personnes": [
    {
      "prenom": "Robert",
      "nom": "Lepingre",
      "email": "bobby@exemple.com",
      "age": 41,
      "ville": "Paris"
    },
    {
      "prenom": "Jeanne",
      "nom": "Ducoux",
      "email": "jeanne@exemple.com",
      "age": 32,
      "ville": "Marseille"
    },
    {
      "prenom": "Pierre",
      "nom": "Lenfant",
      "email": "pierre@exemple.com",
      "age": 23,
      "ville": "Rennes"
    }
  ]
}
```

JSON

La syntaxe



JSON

Les types de données

Type de données	Exemple	Commentaires sur la valeur
Number	{"age":30}, {"temperature":-5}, {"prix":9,99}	Pas de guillemet
Integer	{"age":30}	Pas de guillemet
String	{"name":"John"}	**Des guillemets**
Boolean	{"sale":true}	Pas de guillemet
null	{"myCount": null}	Pas de guillemet

JSON

Les types de données Structurées

Type de données	Exemple

Object	{
	"employee":{"name":"John", "age":30, "city":"New York"}
	}

Array	{
	“names":["John", "Anna", "Peter"]
	}

TD

JSON

TD1 : XML et JSON

- TD1.1 : Ecrire en XML la fiche descriptive d'une voiture avec pour caractéristique :
  - Marque,
  - série,
  - année,
  - stock disponible ou non
  - pour chaque voiture
    - les couleurs disponibles
    - les motorisations disponibles avec comme caractéristique : l'énergie, la consommation et le nombre de chevaux fiscaux.
- TD1.2 : Ecrire la même fiche en JSON.
- TD 1.3 : Ajouter un attribut XML à la caractéristique consommation pour préciser l'unité : l/100 km. Faire de même en JSON.

JSON

TD1.1 : Correction

```
<voiture>
  <marque>SEAT</marque>
  <serie>LEON</serie>
  <annee>2021</annee>
  <stock>true</stock>
  <couleurs>
    <couleur>blanc</couleur>
    <couleur>noir</couleur>
  </couleurs>
  <motorisation>
    <moteur>
      <energie>essence</energie>
      <conso>7</conso>
      <cv>5</cv>
    </moteur>
    <moteur>
      <energie>diesel</energie>
      <conso unit= "L/100 km" >7</conso>
      <cv>5</cv>
    </moteur>
  </motorisation>
</voiture>
```

JSON

TD1.2 : Correction

```xml
<voiture>
  <marque>SEAT</marque>
  <serie>LEON</serie>
  <annee>2021</annee>
  <stock>true</stock>
  <couleurs>
    <couleur>blanc</couleur>
    <couleur>noir</couleur>
  </couleurs>
  <motorisation>
    <moteur>
      <energie>essence</energie>
      <conso>7</conso>
      <cv>5</cv>
    </moteur>
    <moteur>
      <energie>diesel</energie>
      <conso unit= "L/100 km" >7</conso>
      <cv>5</cv>
    </moteur>
  </motorisation>
</voiture>
```

```json
{
  "voiture": {
    "marque": "SEAT",
    "serie": "LEON",
    "annee": "2021",
    "stock": "true",
    "couleurs": {
      "couleur": [ "blanc", "noir" ]
    },
    "motorisation": {
      "moteur": [
        { "energie": "essence", "conso": 7, "cv": "5" },
        { "energie": "diesel", "conso": 7, "cv": "5" }
      ]
    }
  }
}
```

JSON

TD1.3 : Correction

```xml
<voiture>
  <marque>SEAT</marque>
  <serie>LEON</serie>
  <annee>2021</annee>
  <stock>true</stock>
  <couleurs>
    <couleur>blanc</couleur>
    <couleur>noir</couleur>
  </couleurs>
  <motorisation>
    <moteur>
      <energie>essence</energie>
      <conso unit= "L/100 km" >7</conso>
      <cv>5</cv>
    </moteur>
    <moteur>
      <energie>diesel</energie>
      <conso unit= "L/100 km" >7</conso>
      <cv>5</cv>
    </moteur>
  </motorisation>
</voiture>
```

```json
{
  "voiture": {
    "marque": "SEAT",
    "serie": "LEON",
    "annee": "2021",
    "stock": "true",
    "couleurs": {
      "couleur": [ "blanc", "noir" ]
    },
    "motorisation": {
      "moteur": [
        { "energie": "essence", "conso": { unit: "litres/100", "valeur": 7 }, "cv": "5" },
        { "energie": "diesel", "conso": { unit: "litres/100", "valeur": 7 }, "cv": "5" }
      ]
    }
  }
}
```

JSON Schéma

JSON Schéma

Définition

- Spécification JSON Schéma : https://json-schema.org/
- Permet de faire une description et une validation d'un document JSON

```
{ "productId": 1, "productName": "A green door", "price": 12.50, "tags": [ "home", "green" ] }
```

Questions que l'on peut se poser :

- Que signifie « productId » ?
- Quels sont les types de données autorisées pour chaque clé ?
- Est-ce que « productName » est obligatoire ?
- Est-ce que « price » peut être zéro (0) ?
- Quelles sont les valeurs de « tags » valident ?
- …

JSON Schéma

Exemple

```
{
"$schema": "https://json-schema.org/draft/2020-12/schema",
"$id": "https://example.com/product.schema.json",

"title": "Product",
"description": "A product in the catalog",

"type": "object"
}
```

- Un JSON Schéma respect la syntaxe JSON
- Il y a 3 éléments distincts

- Des mots clé du schéma :
  - $schema = indique la version de la spécification utilisée
  - $id = URI de ce schéma

- Des annotations du schéma : title, description
  - Uniquement des informations descriptives

- Des éléments de validation du schéma : type
  - Permet de poser des contraintes sur le JSON

JSON Schéma

Exemple

```
{
"$schema": "https://json-schema.org/draft/2020-12/schema",
"$id": "https://example.com/product.schema.json",
"title": "Product",
"description": "A product from Acme's catalog",
"type": "object",

"properties":{
   "productId": {
      "description": "The unique identifier for a product",
      "type": "integer"
      }
   },
   "required": [ "productId" ]
}
```

- **Properties** : indique la liste des clés du document Json.
- **productId** : fait référence à une clé nommée « productId » dans le JSON. Contiens ensuite les éléments de validation et de contrainte.
- **Required** : Indique les champs obligatoires pour l'ensemble du document JSON.

JSON Schéma

Les contraintes pour toutes les types de clé

Contraintes	Description	Exemple
type	Précise le type JSON de la clé	{“type“:"number“}
enum	Liste de valeurs possible	{ "enum": ["red", "amber", "green"] }
const	Valeur unique d’un champ	"const": “France"

JSON Schéma

Les types de données - Rappel

Type de données	Exemple
Nombre	{"age":30}, {“temperature":-5}, {“prix":9,99}
Entier	{"age":30}
Chaîne de caractère	{"name":"John"}
Boolean	{"sale":true}

Objet	{
	"employee":{"name":"John", "age":30, "city":"New York"}
	}

Tableau	{
	"employees":["John", "Anna", "Peter"]
	}

null	{"myCount": null}

Le type Boolean

JSON Schéma

Les contraintes pour les clés de type « boolean »

- Uniquement deux valeurs possibles : true ou false

{ "type": "boolean" }

true

false

Correspond à une valeur de type « String »

“true”

Correspond à une valeur de type « Number »

0

Les types Number et Integer

JSON Schéma

Les contraintes pour les clés de type « number » et « integer »

Contraintes	Description	Exemple
maximum	$x \leq$ maximum	“maximum": 100
exclusifMaximum	$x <$ exclusiveMaximum	"exclusiveMaximum": 100
minimum	$x \geq$ minimum	"minimum": 100
exclusiveMinimum	$x >$ exclusiveMinimum	"exclusiveMinimum": 100
multipleOf	Le résultat de la division de la valeur du champs par la contrainte doit être un entier.	"multipleOf" : 10

JSON Schéma

Les contraintes pour les clés de type « number » et « integer »

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

42

-1

Entier avec un valeur de zéro après la virgule :

1.0

Valeur flottante

3.15

Valeur sous forme de « String »

“42”

JSON Schéma

Les contraintes pour les clés de type « number » et « integer »

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

42

-1

Entier avec un valeur de zéro après la virgule :

1.0

Valeur flottante

3.15

Valeur sous forme de « String »

“42”

JSON Schéma

Les contraintes pour les clés de type « number » et « integer » - multipleOf

```
{
    "type": "number",
    "multipleOf" : 10
}
```

0

10

20

N’est pas un multiple de 10

23

JSON Schéma

Les contraintes pour les clés de type « number » et « integer » - les ranges

```
{
  "type": "number",
  "minimum": 0,
  "exclusiveMaximum": 100
}
```

Inférieur à 0

-1

0

10

99

exclusiveMaximum à pour valeur 100

100

Plus grand que 100

101

Le type String

JSON Schéma

Les contraintes pour les clés de type « string »

Contraintes	Description	Exemple
minLength maxLength	Taille de la chaîne de caractère	{ "type": "string", "minLength": 2, "maxLength": 3 }
pattern	*Expression régulière*	"pattern": "^(\\([0-9]{3}\\))?[0-9]{3}-[0-9]{4}$"
format	*N'est pas une contrainte mais une information. Permet de préciser comment interpréter le champ notamment pour les dates*	

JSON Schéma

La contrainte format

- Précise le format pour une chaîne de caractère pour des cas particulier
- Exemple: Les dates
- Il n'existe pas de type date dans le format JSON. Elles sont considérées comme des chaînes de caractères
- Le mot-clé « format » : il s'agit uniquement d'une description, il n'y a aucune contrainte associée. Permet à l'auteur de préciser comment interpréter le champ. Certaine implémentation de validateur JSON peuvent l'utiliser.

Format existant	Valeur	definition
date-time	2018-11-13T20:20:39+00:00	{"format":"date-time"}
email	jean.dupont@exemple.com	{"format":"email"}
ipv4	192.160.10.1	{"format":"ipv4"}
uuid	3e4666bf-d5e5-4aa7-b8ce-cefe41c7568a	{"format":"uuid"}

JSON Schéma

Les contraintes pour les propriétés de type « string »

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

“ceci est une chaîne de caractère”

“”

“20”

Valeur numérique

20

JSON Schéma

Les contraintes pour les propriétés de type « string » - Length

```
{
  "type": "string",
  "minLength": 2,
  "maxLength": 3
}
```

Taille inférieur à 2

“A”

“AB”

“ABC”

Taille supérieur à 3

“ABCD”

JSON Schéma

Les contraintes pour les propriétés de type « string » - Regular Expressions

"555-1212"

"(888)555-1212"

```
{
  "type": "string",
  "pattern": "^(\\([0-9]{3}\\))?[0-9]{3}-[0-9]{4}$"
}
```

Par de caractère et d'espace autorisé

"(888)555-1212 ext. 532"

"(800)FLOWERS"

Le type Array

JSON Schéma

Les contraintes pour les propriétés des type « array »

Contraintes	Description	Exemple
items	Permet de définir un type pour l'ensemble des éléments de la liste	"items": { "type": "number" }
minItems maxItems	La taille de la liste	"minItems": 2, "maxItems": 3
uniqueItems	Les éléments de la liste sont unique	"uniqueItems": **true**
Contains	Contient au moins un élément de ce type	"type": "number"
minContains maxContains	Le nombre d'occurrence du type d'un contains	"minContains": 2, "maxContains": 3

JSON Schéma

Les contraintes pour les propriétés des type « array »

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

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

```
[3, "different", { "types" : "of values" }]
```

```
{"Not": "an array"}
```

JSON Schéma

Les contraintes pour les propriétés des type « array » - items

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

[1, 2, 3, 4, 5]

[3, "different", { "types" : "of values" }]

Tableau vide

[]

JSON Schéma

Les contraintes pour les propriétés des type « array » - Min et Max items

```
{
  "type": "array",
  "minItems": 2,
  "maxItems": 3
}
```

[]

[1]

[1, 2]

[1, 2, 3]

[1, 2, 3, 4]

JSON Schéma

Les contraintes pour les propriétés des type « array » - UniqueItems

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

[]

[1, 2, 3, 4, 5]

Deux fois la valeur 3

[1, 2, 3, 3, 4, 5]

JSON Schéma

Les contraintes pour les propriétés des type « array » - Contains

```
{
  "type": "array",
  "contains": {
    "type": "number"
  }
}
```

[1, 2, 3, 4, 5]

Au moins un « number »

["life", "universe", "everything", 42]

Aucun « number »

["life", "universe", "everything", "forty-two"]

JSON Schéma

Les contraintes pour les propriétés des type « array » - Min et Max contains

```
{
  "type": "array",
  "contains": {
    "type": "number"
  },
  "minContains": 2,
  "maxContains": 3
}
```

Un seul nombre, inférieur à minContains

["apple", "orange", 2]

["apple", "orange", 2, 4]

["apple", "orange", 2, 4, 8]

Trop de nombre, supérieur à maxContains

["apple", "orange", 2, 4, 8, 16]

JSON

Les contraintes pour les propriétés des type « array »

- Contraintes pour champs de type « array » - Les validation par groupe (tuple validation)

Contraintes	Description	Exemple
prefixItems	Permet de valider le type des items de la liste.	{ "type": "array", "prefixItems": [{ "type": "number" }, { "type": "string" }, { "enum": ["Street", "Avenue", "Boulevard"] }, { "enum": ["NW", "NE", "SW", "SE"] }] }
items	Permet de préciser si la liste peut contenir des éléments supplémentaires non précisés par prefixItems	"items": **false**

JSON Schéma

Les contraintes pour les propriétés des type « array » - Tuples validation

```
{
  "type": "array",
  "prefixItems": [
    { "type": "number" },
    { "type": "string" },
    { "enum": ["Street", "Avenue", "Boulevard"] },
    { "enum": ["NW", "NE", "SW", "SE"] }
  ]
}
```

[1600, "Pennsylvania", "Avenue", "NW"]

Drive ne fait pas parti de l'énum

[24, "Sussex", "Drive"]

Le premier élément n'est pas un nombre

["Palais de l'Élysée"]

Tous les items ne sont pas obligatoire

[10, "Downing", "Street"]

Possibilité d'ajouter des valeurs

[1600, "Pennsylvania", "Avenue", "NW", "Washington"]

JSON Schéma

Les contraintes pour les propriétés des type « array » - Additionnal Items

```
{
  "type": "array",
  "prefixItems": [
    { "type": "number" },
    { "type": "string" },
    { "enum": ["Street", "Avenue", "Boulevard"] },
    { "enum": ["NW", "NE", "SW", "SE"] }
  ],
  "items": false
}
```

[1600, "Pennsylvania", "Avenue", "NW"]

Tous les items ne sont pas obligatoire

[10, "Downing", "Street"]

Possibilité d'ajouté des valeurs

[1600, "Pennsylvania", "Avenue", "NW", "Washington"]

Le type Objet

JSON Schéma

Les contraintes pour les propriétés des type « objet »

Contraintes	Description	Exemple

properties	Une liste de propriétés autorisées avec un schéma de validation	"properties": {
		"number": { "type": "number" },
		"street_name": { "type": "string" },
		"street_type": { "enum": ["Street", "Avenue", "Boulevard"] }
		}

patternProperties	Applique un schéma de validation suivant les noms des propriétés qui correspondent à un pattern	"patternProperties": {
		"^S_": { "type": "string" },
		"^I_": { "type": "integer" }
		}

additionalProperties	Possibilité d'ajouter ou non des propriétés non définies	"additionalProperties": **false** "additionalProperties": { "type": "string" }
required	Liste de propriétés obligatoires	"required": ["name", "email"]

propertyNames	Les noms des propriétés doivent respecter un pattern	"propertyNames": {
		"pattern": "^[A-Za-z_][A-Za-z0-9_]*$"
		}

minproperties Maxproperties	Le nombre de propriétés autorisées pour un objet	"minProperties": 2, "maxProperties": 3

JSON Schéma

Les contraintes pour les champs de type « objets »

PROPERTIES

Une liste des propriétés autorisés avec un schéma de validation

```
{
  "type": "object",
  "properties": {
    "number": { "type": "number" },
    "street_name": { "type": "string" },
    "street_type": { "enum": ["Street", "Avenue", "Boulevard"] }
  }
}
```

```
{ "number“: 1600, "street_name”: "Pennsylvania", "street_type”: "Avenue" }
```

```
{ "number“: "1600", "street_name": "Pennsylvania", "street_type": "Avenue" }
```

JSON Schéma

Les contraintes pour les champs de type « objets »

Pattern Properties

Applique un schéma de validation suivant les noms des propriétés qui correspondent à un pattern

```
{
  "type": "object",
  "patternProperties": {
    "^S_": { "type": "string" },
    "^I_": { "type": "integer" }
  }
}
```

```
{ "S_25": "This is a string" }
```

```
{ "I_0": 42 }
```

```
{ "S_0": 42 }
```

```
{ "I_42": "This is a string" }
```

JSON Schéma

Les contraintes pour les champs de type « objets »

Additionnal Properties

Possibilité d'ajouter ou non des propriétés non définies. Par default tous est autorisé.

```
{
  "type": "object",
  "properties": {
    "number": { "type": "number" },
    "street_name": { "type": "string" },
    "street_type": { "enum": ["Street", "Avenue", "Boulevard"] }
  },
  "additionalProperties": { "type": "string" }
}
```

```
{ "number": 1600, "street_name": "Pennsylvania", "street_type": "Avenue", "direction": "NW" }
```

```
{ "number": 1600, "street_name": "Pennsylvania", "street_type": "Avenue", "office_number": 201 }
```

JSON Schéma

Les contraintes pour les champs de type « objets »

Required Properties

Liste de propriétés obligatoires

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "email": { "type": "string" },
    "address": { "type": "string" },
    "telephone": { "type": "string" }
  },
  "required": ["name", "email"]
}
```

```
{
  "name": "William Shakespeare",
  "email": "bill@stratford-upon-avon.co.uk",
  "address": "Henley Street, Stratford-upon-Avon, Warwickshire, England",
  "authorship": "in question"
}
```

```
{
  "name": "William Shakespeare",
  "address": "Henley Street, Stratford-upon-Avon, Warwickshire, England",
}
```

JSON Schéma

JSON-Schéma – Les contraintes pour les champs de type « objets »

Property names

Les noms des propriétés doivent respecter un pattern

```
{
  "type": "object",
  "propertyNames": {
    "pattern": "^[A-Za-z_][A-Za-z0-9_]*$"
  }
}
```

```
{  "_a_proper_token_001": "value"}
```

```
{  "001 invalid": "value"}
```

JSON Schéma

JSON-Schéma – Les contraintes pour les champs de type « objets »

Size (minProperties, maxProperties)

Le nombre de propriétés autorisé pour un objet

```
{
  "type": "object",
  "minProperties": 2,
  "maxProperties": 3
}
```

{}

{ "a": 0 }

{ "a": 0, "b": 1 }

{ "a": 0, "b": 1, "c": 2 }

{ "a": 0, "b": 1, "c": 2, "d": 3 }

JSON Schéma

JSON-Schéma – Les contraintes pour les champs de type « objets »

Dependent Required

Permet de rendre des propriétés obligatoire suivant d'autres propriété

```
{
  "type": "object",
  "properties": {
    "name": { "type": "string" },
    "email": { "type": "string" },
    "address": { "type": "string" },
    "telephone": { "type": "string" }
  },
 " dependentRequired": {
      email: ["name"]
}
```

```
{
  "name": "William Shakespeare",
  "email": "bill@stratford-upon-avon.co.uk",
  "address": "Henley Street, Stratford-upon-Avon, Warwickshire, England",
  "authorship": "in question"
}
```

```
{
  "name": "William Shakespeare",
  "address": "Henley Street, Stratford-upon-Avon, Warwickshire, England",
}
```

TD

JSON Schéma

TD2 : JSON

- TD2.1 : Ecrire un json valide suivant le schéma suivant :
- TD2.2 : Comment améliorer le schéma ?
- TD2.2 : Ecrire un schéma pour valider ce json :

```
{
   "adresse": {
      "numéro": 32,
      "complément": "bis",
      "voie": "boulevard",
      "rue": "du chemin vert",
      "code_postal": "59000",
      "ville": "LILLE"
   }
}
```

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "nom": {"type": "string"},
    "prenom": {"type": "string"},
    "date_de_naissance": {"type": "string"},
    "adresse": {"type": "string"},
    "mensurations": {
      "type": "object",
      "properties": {
        "poids": {"type": "number"},
        "taille": {"type": "integer"}
      },
    },
    "sexe": {
      "type": "string",
    },
    "hobbies": {
      "type": "array",
      "items": [
        {"type": "string"}
      ],
     "minItems": 2
    },
    "permis": {"type":"boolean"}
  }
}
```

JSON Schéma

TD2.1 : correction


```
{
  "nom": "ABCDEFGHIJKLMNOPQRS",
  "prenom": "ABCDEFG",
  "date_de_naissance": "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
  "adresse": "ABCDEFGHI",
  "mensurations": {
    "poids": -23.0,
    "taille": 544
  },
  "sexe": "F",
  "hobbies": [
    "ABCDEFGHIJ",
    null
  ],
  "permis": true
}
```

JSON Schéma

TD2.2 : correction

Améliorations possible :

- Date de naissance : ajouter un format
- Poids : valeur positive
- Sexe : Ajouter une Enum
- Définir des champs required

JSON Schéma

TD 2.3 : Correction

```
{
   "adresse": {
      "numéro": 32,
      "complément": "bis",
      "voie": "boulevard",
      "rue": "du chemin vert",
      "code_postal": "59000",
      "ville": "LILLE"
   }
}
```

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "type": "object",
 "properties": {
  "adresse": {
   "type": "object",
   "properties": {
    "numéro": {"type": "integer", "minimum":1},
    "complément": {
     "type": "string",
     "enum": ["bis", "ter"]
    },
    "voie": {
     "type": "string",
     "enum": ["Rue", "Boulevard", "Chemin", "Avenue" ]
    },
    "rue": {"type": "string"},
    "code_postal": {
     "type": "string",
     "pattern": "[0-9]{5}"
    },
    "ville": {"type": "string"}
   },
   "required": ["numéro","voie","rue", "code_postal", "ville"]
  }
 },
 "required": [
  "adresse"
 ]
}
```

JSON et Java

JSON

Modèle objet en Java

- Librairie Java la plus populaire : Jackson (https://github.com/FasterXML/Jackson)
- Les principales fonctionnalités :
  - Parser un fichier Json
  - Charger un fichier Json dans un modèle objet : Désérialisation
  - Transformer un modèle objet en Json : Sérialisation
  - Génération de Json Schema
  - Validation d'un Json par un schéma

JSON et Java

Modèle objet en Java - exemple

```
public class Car {
  private String color;
  private String type;
  // standard getters setters
}
```

```
String json = "{ \"color\" : \"Black\", \"type\" : \"BMW\" }";
Car car = objectMapper.readValue(json, Car.class);
```

```
String carAsString = objectMapper.writeValueAsString(car);
```

```
{"color":“Black","type":“BMW"}
```

JSON et Java

Et dans tous les langages

Bibliographie

Bibliographie

Bibliographie

- http://json-schema.org/understanding-json-schema/
- https://www.baeldung.com/jackson-object-mapper-tutorial
- https://github.com/FasterXML/Jackson
- https://www.json.org/json-fr.html
- https://www.liquid-technologies.com/online-schema-to-json-converter