

DataGen

JSON/XML Dataset Generator



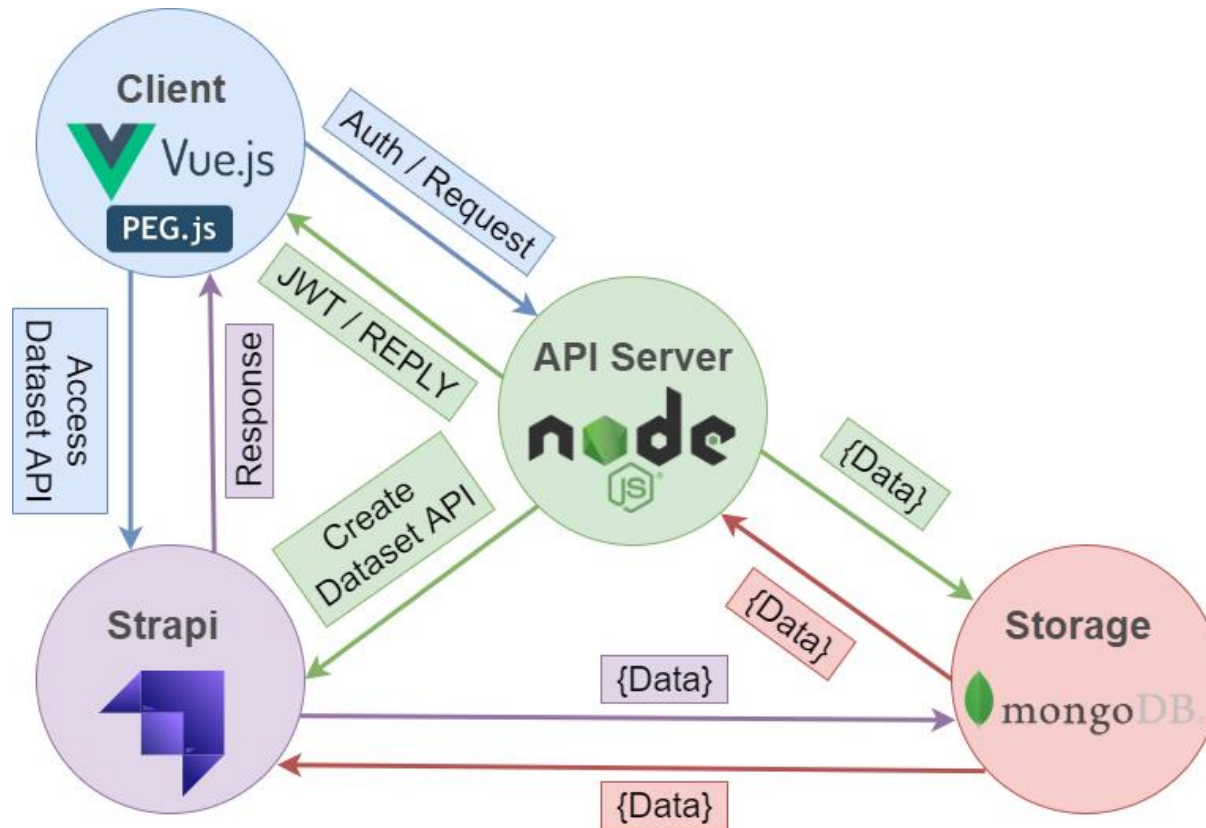
Concept

- Generation of datasets (JSON/XML)
- Generation of REST APIs on the datasets + CRUD
- Download the APIs and datasets
- Application Integration

Motivation

- Synthetic Data (User Privacy)
- Existing tools not complex enough
- Limited formats
- No fuzzy generation
- Multilingual support
- Etc...

Architecture



Strapi vs JSON-Server



- ✓ Generates a REST API
- ✓ Arbitrary Number of APIs
- ✓ Scalable
- ✗ Does not digest datasets
- ✗ Complex “Components”



- ✓ Generates a REST API
- ✓ Simple configuration
- ✓ Digests datasets
- ✗ Only allows 1 API
- ✗ Not scalable

Grammar

PEG.js

```
1 <!LANGUAGE en>
2 {
3   profile: [
4     'repeat(3)': {
5       name: '{{fullName()}}',
6       age: '{{integer(15,60)}}',
7       or() {
8         BI: '{{integerOfSize(8)}}-{{integer(0,9)}}',
9         CC: '{{integerOfSize(8)}}-{{integer(0,9)}}-{{letter("uppercase")}}{{letter("uppercase")}}{{integer(0,9)}}'
10      },
11     having(70) {
12       bio: '{{lorem(1,"sentences")}}'
13     },
14     nr_books: '{{integer(1,10)}}',
15     books: [ 'repeat(this.nr_books)': {
16       title: '{{lorem(1,"words")}}',
17       support: '{{random("Físico","Digital")}}',
18       rating: '{{integer(1,5)}}'
19     } ],
20     favorite_book(gen) {
21       var titles = this.books.map(x => x.title)
22       return gen.random(...titles)
23     }
24   ]
25 }
26 }
```

```
1 {
2   "profile": [
3     {
4       "name": "Lester Saianda",
5       "age": 34,
6       "CC": "23268383-7-FH4",
7       "bio": "Non id eiusmod consectetur duis et consectetur.",
8       "nr_books": 2,
9       "books": [
10        {
11          "title": "ad",
12          "support": "Físico",
13          "rating": 4
14        },
15        {
16          "title": "cillum",
17          "support": "Digital",
18          "rating": 1
19        }
20      ],
21       "favorite_book": "ad"
22     },
23     {
24       "name": "Daldip Rot",
25       "age": 50,
26       "BI": "96657177-2",
27       "nr_books": 8,
28       "books": [
29        {
30          "title": "velit",
31          "support": "Digital",
```

JSON-like

```
1 <!LANGUAGE en>
2 {
3   profile: [
4     'repeat(3)': {
5       name: '{{fullName()}}',
6       age: '{{integer(15,60)}}',
7       or() {
8         BI: '{{integerOfSize(8)}}-{{integer(0,9)}}',
9         CC: '{{integerOfSize(8)}}-{{integer(0,9)}}-{{letter("uppercase")}}{{letter("uppercase")}}{{integer(0,9)}}'
10      },
11     having(70) {
12       bio: '{{lorem(1,"sentences")}}'
13     },
14     nr_books: '{{integer(1,10)}}',
15     books: [ 'repeat(this.nr_books)': {
16       title: '{{lorem(1,"words")}}',
17       support: '{{random("Físico","Digital")}}',
18       rating: '{{integer(1,5)}}'
19     } ],
20     favorite_book(gen) {
21       var titles = this.books.map(x => x.title)
22       return gen.random(...titles)
23     }
24   ]
25 }
26 }
```

Interpolation Functions

```
1 <!LANGUAGE en>
2 {
3   examples: [
4     'repeat(2)': {
5       objectId: '{{objectId()}}',
6       guid: '{{guid()}}',
7       index: '{{index()}}',
8       letter: '{{letter()}}',
9       position: '{{position()}}',
10      date: '{{date("05-01-2001", "10-12-2020", "DD/MM/AAAA")}}',
11      random: '{{random("Blue", false, 31, [1,2,3])}}'
12    }
13  ]
14 }
```

```
1 {
2   "examples": [
3     {
4       "objectId": "60dcae00780814f12eca083c",
5       "guid": "571390f0-771f-4a6e-9ff6-a3a02af4d758",
6       "index": 0,
7       "letter": "l",
8       "position": "(-56.85308, -84.15255)",
9       "date": "06/12/2008",
10      "random": "Blue"
11    },
12    {
13      "objectId": "60dcae0035668e02d5290843",
14      "guid": "ffe962c6-2cbc-44f6-b496-94191a4bc148",
15      "index": 1,
16      "letter": "y",
17      "position": "(46.54091, 16.24033)",
18      "date": "22/10/2015",
19      "random": 31
20    }
21  ]
22 }
```


Interpolation Functions

```
1 <!LANGUAGE en>
2 {
3   examples: [
4     'repeat(2)': {
5       pt_district: '{{pt_district()}}',
6       name: '{{fullName()}}',
7       actor: '{{actor()}}',
8       animal: '{{animal()}}',
9       brand: '{{brand()}}',
10      day: '{{weekday()}}',
11      job_title: '{{job()}}',
12      musician: '{{musician()}}'
13    }
14  ]
15 }
```

```
1 {
2   "examples": [
3     {
4       "pt_district": "Braga",
5       "name": "Tyler-Autumn Rollon",
6       "actor": "Tom Cruise",
7       "brand": "Netflix",
8       "day": "Wednesday",
9       "job_title": "Proof coins inspector",
10      "musician": "R.E.L."
11    },
12    {
13      "pt_district": "Viana do Castelo",
14      "name": "Tyianna Escayola",
15      "actor": "Toni Collette",
16      "brand": "YouTube",
17      "day": "Saturday",
18      "job_title": "Fitness services manager",
19      "musician": "Kid Rock"
20    }
21  ]
22 }
```

Javascript Functions

```
1 <!LANGUAGE en>
2 {
3   examples: [
4     'repeat(2)': {
5       name: "Oliver",
6       email(gen) {
7         var i = gen.integer(1,30);
8         return `${this.name}.${gen.surname()}${i}@gmail.com`.toLowerCase(),
9       },
10      probability: gen => { return Math.random() * 100; }
11    }
12  ]
13 }
```

```
1 {
2   "examples": [
3     {
4       "name": "Oliver",
5       "email": "oliver.barlehner20@gmail.com",
6       "probability": 29.6027670967689
7     },
8     {
9       "name": "Oliver",
10      "email": "oliver.volking21@gmail.com",
11      "probability": 55.616499949869635
12    }
13  ]
14 }
```

Fuzzy generation

Probabilities

```
1 <!LANGUAGE en>
2 {
3   examples: [ 'repeat(3)': {
4     missing(50) {
5       prop1: 1,
6       prop2: 2
7     },
8     having(80) { prop3: 3 }
9   } ]
10 }
```

```
1 {
2   "examples": [
3     {},
4     {
5       "prop1": 1,
6       "prop2": 2,
7       "prop3": 3
8     },
9     {
10      "prop3": 3
11    }
12  ]
13 }
```

Logical conditions

```
1 <!LANGUAGE en>
2 {
3   examples: [ 'repeat(3)': {
4     type: '{{random("A","B","C')}}',
5     if (this.type == "A") { A: "type is A" }
6     else if (this.type == "B") { B: "type is B" }
7     else { C: "type is C" }
8   } ]
9 }
```

```
1 {
2   "examples": [
3     {
4       "type": "A",
5       "A": "type is A"
6     },
7     {
8       "type": "A",
9       "A": "type is A"
10    },
11    {
12      "type": "C",
13      "C": "type is C"
14    }
15  ]
16 }
```

Fuzzy generation

Or (mutual exclusivity)

```
1 <!LANGUAGE en>
2 {
3   examples: [ 'repeat(3)': {
4     or() {
5       prop1: 1,
6       prop2: 2,
7       prop3: 3
8     }
9   } ]
10 }
```

```
1 {
2   "examples": [
3     {
4       "prop3": 3
5     },
6     {
7       "prop1": 1
8     },
9     {
10      "prop1": 1
11    }
12  ]
13 }
```

At least

```
1 <!LANGUAGE en>
2 {
3   examples: [ 'repeat(3)': {
4     at_least(2) {
5       prop1: 1,
6       prop2: 2,
7       prop3: 3
8     }
9   } ]
10 }
```

```
1 {
2   "examples": [
3     {
4       "prop3": 3,
5       "prop1": 1
6     },
7     {
8       "prop2": 2,
9       "prop1": 1
10    },
11    {
12      "prop2": 2,
13      "prop1": 1,
14      "prop3": 3
15    }
16  ]
17 }
```

Functional capabilities

```
1 <!LANGUAGE en>
2 {
3   example: [ 'repeat(1)': {
4     map: range(5).map(value => { return value+1 }),
5
6     filter: [0,1,2].filter(function(value, index) {return [0,1,2][index]>0}),
7
8     reduce: range(5).reduce((accum, value, index, array) => {
9       return accum + array[index]
10    }),
11
12    combined: range(5).map((value) => { return value+3 })
13      .filter(x => { return x >= 5 })
14      .map(x => { return x*2 })
15      .reduce((a,c) => {return a+c})
16   } ]
17 }
```

```
1 {
2   "examples": [
3     {
4       "map": [
5         1,
6         2,
7         3,
8         4,
9         5
10      ],
11      "filter": [
12        1,
13        2
14      ],
15      "reduce": 10,
16      "combined": 36
17    }
18  ]
19 }
```

Real-life study case

```
1 <!LANGUAGE pt>
2 {
3   autoEliminação: {
4     fonteLegitimação: {
5       tipo: '{{random("PGD/LC", "TS/LC", "PGD", "RADA", "RADA/CLAV")}}',
6       diploma(gen) {
7         var portaria = `Portaria ${gen.integer(300,500)}/${gen.integer(2000,2021)}`
8         var despacho = `Despacho DGLAB ${gen.integer(100,500)}/${gen.integer(2000,2021)}`
9         return gen.random("LC", portaria, despacho)
10      }
11    },
12    fundos(gen) {
13      if (["PGD/LC","TS/LC","PGD"].includes(this.fonteLegitimação.tipo))
14        return [gen.pt_entity()]
15      else {
16        var arr = []
17        for (var i = 0; i < gen.integer(1,5); i++) arr.push(gen.pt_entity())
18        return arr
19      }
20    },
```

Real-life study case

```
21     classes: [ 'repeat(2,5)': {
22         if (["PGD/LC","TS/LC"].includes(this.fonteLegitimacao.tipo)) {
23             código: gen => {
24                 var nivel1 = gen.random(...gen.range(100,950,50))
25                 var nivel2 = gen.random(10,20,30,40,50)
26                 var nivel3 = gen.integer(1,999,3)
27                 var nivel4 = gen.random("01","02")
28
29                 var classe = nivel1 + '.' + nivel2 + '.' + nivel3
30                 if (Math.random() > 0.5) classe += '.' + nivel4
31                 return classe
32             }
33         }
34         else {
35             at_least(1) {
36                 código(gen) {
37                     var nivel1 = gen.random(...gen.range(100,950,50))
38                     var nivel2 = gen.random(10,20,30,40,50)
39                     var nivel3 = gen.integer(001,999)
40                     var nivel4 = gen.random("01","02")
41
42                     var classe = nivel1 + '.' + nivel2 + '.' + nivel3
43                     if (Math.random() > 0.5) classe += '.' + nivel4
44                     return classe
45                 },
46                 referência: '{{random(1,2,3,55,56)}}'
47             }
48         },
```

Real-life study case

```
49     if (["PGD/LC","TS/LC"].includes(this.fonteLegitimação.tipo)) {
50         naturezaIntervenção: '{{random("DONO", "PARTICIPANTE", "DONO/PARTICIPANTE')}}',
51         if (["PARTICIPANTE","DONO/PARTICIPANTE"].includes(this.naturezaIntervenção)) {
52             donos: [ 'repeat(1,5)': '{{pt_entity_abbr()}}' ]
53         }
54     },
55     anoInício: '{{integer(1921,2021)}}',
56     anoFim(gen) {
57         var ano = gen.integer(1921,2021)
58         while (ano < this.anoInício) ano = gen.integer(1921,2021)
59         return ano
60     },
61     dimensãoSuporte: {
62         at_least(1) {
63             papel: '{{integer(1,2000)}}',
64             digital: '{{integer(1,2000)}}',
65             outro: {
66                 valor: '{{integer(1,2000)}}',
67                 unidade: '{{lorem(1,"words')}}'
68             }
69         }
70     },
```


Real-life study case

```
71     númeroAgregações: '{{integer(1,50)}}',
72     agregações: [ 'repeat(this.númeroAgregações)': {
73         código: '{{pt_entity_abbr()}} - {{integer(1,200)}}',
74         título: '{{lorem(3,"words')}}',
75         ano: '{{integer(1921,2021)}}',
76         if (["PGD/LC","TS/LC"].includes(this.fonteLegitimação.tipo)) {
77             naturezaIntervenção: '{{random("PARTICIPANTE","DONO')}}'
78         }
79     }
80 }
81 }
82 }
```

Interface - Home

The screenshot displays the DataGen web interface. At the top, there are navigation links: Gerar, Modelos, Documentação, Sobre, and Hugo. Below the navigation, there are buttons for Gerar, Guardar Modelo, JSON, XML, Nome: dataset, Download, Gerar API, and Download API. The main area is split into two panels. The left panel shows the JSON Schema definition in Portuguese, and the right panel shows the generated JSON data.

```
1 <!LANGUAGE pt>
2 {
3   perfil: [
4     'repeat(3)': {
5       nome: '{{fullName()}}',
6       idade: '{{integer(15,60)}}',
7       or() {
8         BI: '{{integerOfSize(8)}-{{integer(0,9)}}',
9         CC: '{{integerOfSize(8)}-{{integer(0,9)}}-{{letter("uppercase")}}{{letter("uppercase")}}{{integer(0,9)}}'
10      },
11     having(70) {
12       descrição: '{{lorem(1,"sentences")}}'
13     },
14     nr_livros: '{{integer(1,10)}}',
15     livros: [ 'repeat(this.nr_livros)': {
16       titulo: '{{lorem(1,"words")}}',
17       suporte: '{{random("Físico","Digital')}}',
18       rating: '{{integer(1,5)}}'
19     } ],
20     livro_favorito(gen) {
21       var titulos = this.livros.map(x => x.titulo)
22       return gen.random(...titulos)
23     }
24   ]
25 }
26 }
```

```
1 {
2   "perfil": [
3     {
4       "nome": "Kawinuar Caveiro",
5       "idade": 41,
6       "BI": "82251624-9",
7       "descrição": "Eiusmod dolore qui Lorem irure consequat consectetur.",
8       "nr_livros": 2,
9       "livros": [
10        {
11          "titulo": "aliqua",
12          "suporte": "Físico",
13          "rating": 3
14        },
15        {
16          "titulo": "dolore",
17          "suporte": "Físico",
18          "rating": 4
19        }
20      ],
21     "livro_favorito": "dolore"
22   },
23   {
24     "nome": "Girisha Baldo",
25     "idade": 57,
26     "CC": "18356153-9-VV7",
27     "nr_livros": 2,
28     "livros": [
29      {
30        "titulo": "incididunt",
31        "suporte": "Físico",
32        "rating": 1
33      },
34      {
35        "titulo": "ipsum",
36        "suporte": "Digital",
37        "rating": 4
38      }
39    ],
40     "livro_favorito": "ipsum"
41   },
42   {
43     "nome": "Ianis Andrei Alendouro",
44     "idade": 33,
```

Interface – A User's Models

The screenshot displays the DataGen web interface. At the top, the navigation bar includes 'DataGen', 'Gerar', 'Modelos', 'Documentação', 'Sobre', and 'Hugo'. The main heading is 'Modelos'. Below this, there are search filters: 'Procurar por título...', 'Procurar por autor...', and 'Procurar entre...'. A dropdown menu is open for 'Livros (Hoje às 01:12)', showing a 'Usar Modelo' button and a code editor. The code editor contains the following JSON schema:

```
1 <!--LANGUAGE pt-->
2 {
3   perfil: [
4     'repeat(3)': {
5       nome: '{{fullName()}}',
6       idade: '{{integer(15,60)}}',
7       or() {
8         BI: '{{integerOfSize(8)}}-{{integer(0,9)}}',
9         CC: '{{integerOfSize(8)}}-{{integer(0,9)}}-{{letter("uppercase")}}{{letter("uppercase")}}{{integer(0,9)}}'
10      },
11     having(70) {
12       descrição: '{{lorem(1,"sentences")}}'
13     },
14     nr_livros: '{{integer(1,10)}}',
15     livros: [ 'repeat(this.nr_livros)': {
16       titulo: '{{lorem(1,"words")}}',
17       suporte: '{{random("Físico","Digital")}}',
18       rating: '{{integer(1,5)}}'
19     } ],
20     livro_favorito(gen) {
21       var titulos = this.livros.map(x => x.titulo)
22       return gen.random(...titulos)
23     }
24   ]
25 }
26 }
```

Below the code editor, there is a dropdown for 'Cidades (Último Domingo às 19:20)' and a pagination control showing '1'.

Interface - Documentation

The screenshot shows the documentation page for DataGen. At the top, there is a navigation bar with the title 'DataGen' and several menu items: 'Gerar', 'Modelos', 'Documentação', 'Sobre', and 'Hugo'. The main content area is titled 'Documentação' and contains an 'Introdução' section. The introduction explains that the application uses a PEG.js compiler to process user input and generate datasets. It mentions that the application uses a domain-specific language (DSL) similar to JSON. Below the text, there is a code block showing a JSON-like structure for a person's name and age. The 'Definição de Pares Chave-Valor' section explains that a key-value pair consists of two elements separated by a colon. It lists the allowed data types: Number, String, Array, Boolean, null, DSL Object, and 'Moustache' function. The 'Interpolações (Funções "Moustache")' section describes how to use interpolation functions to generate random values. It includes a code block with examples of interpolation functions like `{{objectid()}}`, `{{integer(59,100)}}`, and `{{random(23, "old", [1,2,3], true)}}`.

DataGen

Gerar Modelos Documentação Sobre Hugo

Documentação

Introdução

A aplicação usa um compilador baseado numa gramática PEG.js para processar o input do utilizador e gerar o dataset pretendido. A gramática mencionada define uma linguagem específica de domínio (DSL), com sintaxe semelhante a JSON, disponibilizando muitas ferramentas que permitem a geração de datasets complexos e diversificados. Estas ferramentas incluem capacidades relacionais e lógicas, fornecendo meios para os datasets satisfazerem vários tipos de limitações - o que facilita a utilização de frameworks declarativos com esta especificação -, bem como capacidades funcionais, permitindo uma gestão e processamento facilitados de certas propriedades dos datasets.

A primeira e mais fundamental das ferramentas implementadas é a sintaxe semelhante a JSON - o utilizador pode especificar propriedades chave-valor, onde o valor pode tomar qualquer tipo básico ou estrutura de dados JSON, desde inteiros a objetos e arrays. O utilizador pode também aninhar estes valores para criar uma estrutura com qualquer profundidade que pretenda.

```
1 nome: {
2   first: ["Hugo", "Cardoso"],
3   last: "Miguel"
4 },
5 age: 21
```

Definição de Pares Chave-Valor

Um par chave-valor é composto por dois elementos separadas por dois pontos (:)

A chave não pode conter espaços brancos (exceto entre o último caractere da *String* e o separador) nem qualquer outro caractere que não pertença ao alfabeto ou que não seja um *underscore*.

Por exemplo, `lorem_ipsum` é uma chave válida enquanto `lorem ipsum`, `lorem:ipsum` ou `lorem-ipsum` não são.

A única exceção é a diretiva `repeat`, que está entre plicas e que recebe como argumento um inteiro. Esta é responsável por gerar um *array* de objetos cujo comprimento é o dado por argumento.

Os valores podem ser um dos seguintes:

- Número
- *String*
- *Array*
- Booleano
- null
- Objeto DSL
- Função "Moustache"

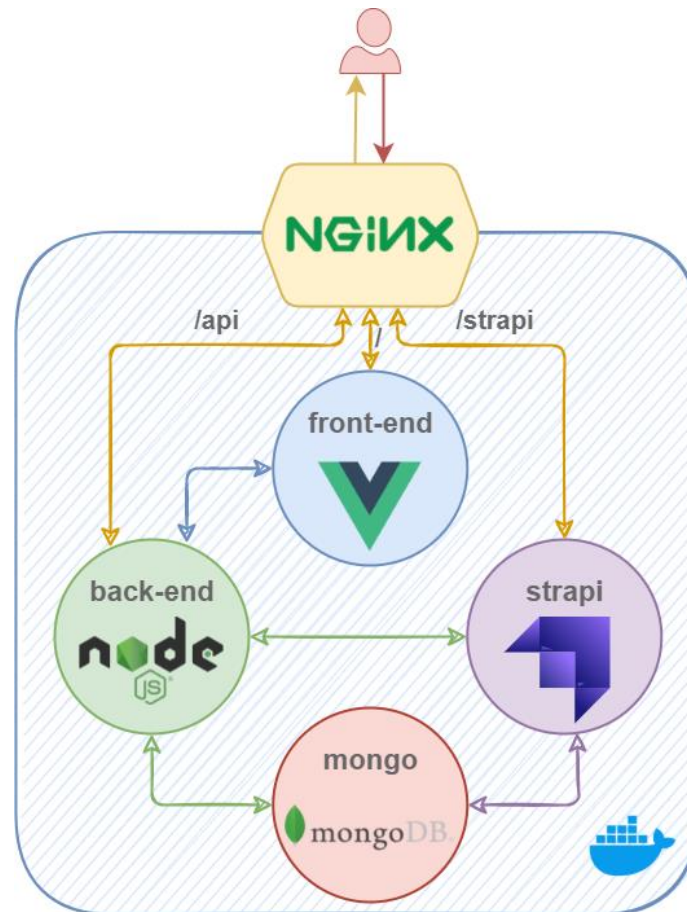
Interpolações (Funções "Moustache")

Para definir o valor de uma propriedade, o utilizador pode também usar interpolação. Para aceder a uma função de interpolação, esta necessita de estar envolta em chavetas duplas. Há dois tipos de funções de interpolação:

- Funções que geram valores espontâneos em tempo de execução, de acordo com as instruções do utilizador - por exemplo, existe uma função de geração de números inteiros aleatórios, onde o utilizador precisa de indicar, no mínimo, a gama de valores que pretende para o resultado.

```
1 id: '{{objectid()}}',
2 int: '{{integer(59,100)}}',
3 random: '{{random(23, "old", [1,2,3], true)}}
```

Deployment



THE END

Questions?

DataGen

JSON/XML Dataset Generator



fliper6/DataGen
Abjiri/DataGen

Jcc20/DataGen
wurzy/DataGen

