

UNE API REST

C'est bien

UNE API REST DOCUMENTÉE

C'est bien mieux

***APIs,
like diamonds,
are forever***

Pourquoi documenter une API ?

- Expliquer l'accès aux ressources
- Présenter les paramètres d'appels
- Présenter les structures des ressources

Pourquoi documenter une API ?

**L'augmentation du nombre d'API exposées
dans un système rend obligatoire la
documentation et un accès facile à celle-ci**

LES POSSIBILITÉS ?



Difficilement versionnable

Peu maintenable

Suivi des modifications ardu

AU FINAL NON UTILISABLE

LES POSSIBILITÉS ?

Difficilement maintenable
lourd à produire
Suivi des modifications ardu

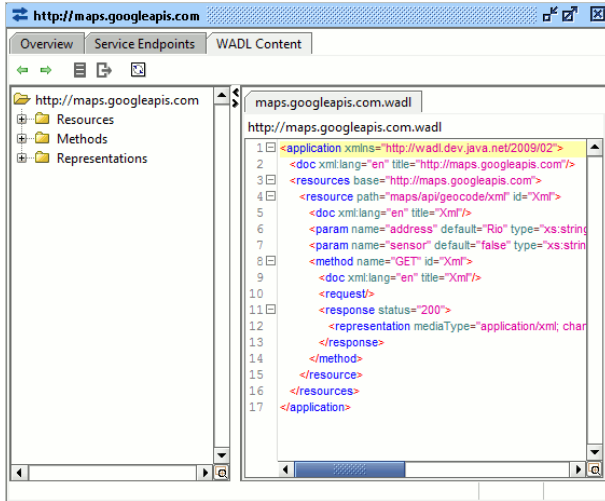


PAS PLUS UTILISABLE

LES POSSIBILITÉS ?

WADL

Tentative de porter WSDL
XML
Brouillon



PERDU DANS LES LIMBES DE L'HISTOIRE

Et c'est bien

SWAGGER 2.0

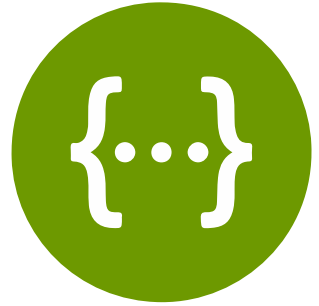
Initié en 2014

Description API REST

format JSON

Top-Bottom ou Bottom-Top

STANDARD DE FAIT



SWAGGER...



2.0 -> 3.0

OPENAPI

C'est quoi ?

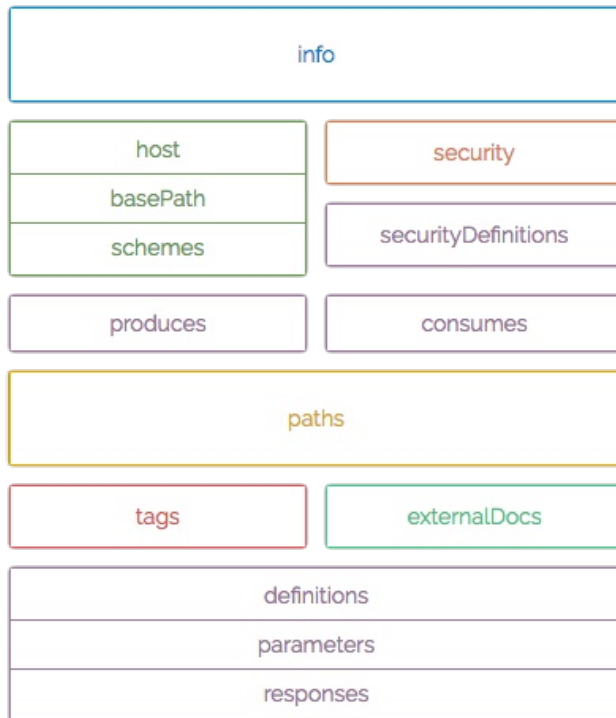
- **OpenAPI Initiative** : Regroupement d'acteurs
- **OpenAPI Spécification**
- **OpenAPI 3** : Version 3 des specs
- **Swagger** : implémentation

OPENAPI

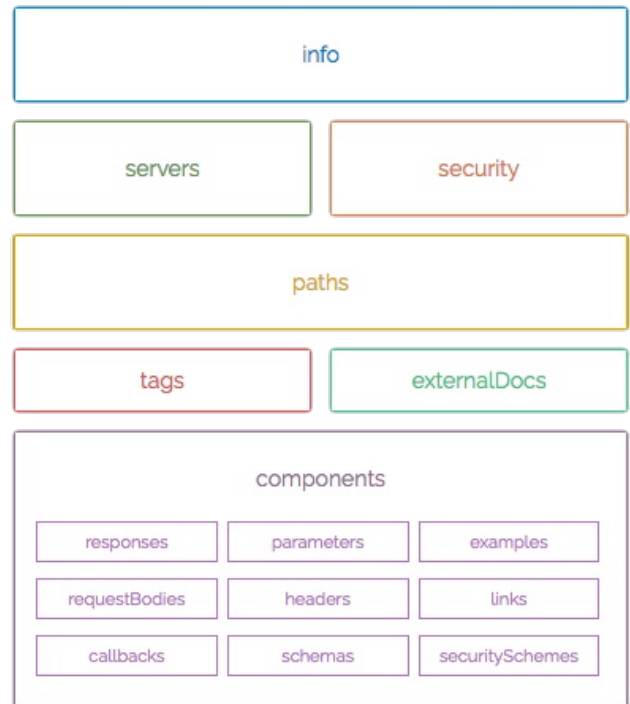
SPÉCIFICATIONS : LES DIFFÉRENCES

SPÉCIFICATIONS : LES DIFFÉRENCES

Swagger 2



OpenAPI 3



OPENAPI

SPÉCIFICATION : FORMAT

- JSON
- YAML

OPENAPI

SPÉCIFICATION: FORMAT JSON

```
...  
"/user/{  
  "get":{  
    "tags":[  
      "user"  
    ],  
    "summary":"get user informations",  
    "description":"",  
    "operationId":"getUser",  
    "produces":[  
      "application/xml",  
      "application/json"  
    ],  
  },  
...  
}
```

Swagger 2 & OpenAPI 3

OPENAPI

SPÉCIFICATION : FORMAT YAML

```
...  
  /user/:  
    get:  
      tags:  
      - "user"  
      summary: "get user informations"  
      description: ""  
      operationId: "getUser"  
      produces:  
      - "application/xml"  
      - "application/json"  
...
```

Mais seulement a partir de OpenAPI 3

SWAGGER

Implementation d'OpenAPI

SWAGGER

Une serie d'outillage

De la génération à la visualisation ...

... et aux tests

SWAGGER

Mais avant de parler outillage

Parlons méthodologie

SWAGGER

GÉNÉRATION DU FICHIER

Deux principes :

- Bottom-top
- Top-bottom

SWAGGER

GÉNÉRATION DU FICHER : *Bottom-Top*

Écriture des spécification de l'API REST d'abord
pour ensuite générer le code.

Specification as code

Pour cela, il existe **Swagger editor**

SWAGGER

GÉNÉRATION DU FICHER : *Bottom-Top*

Swagger Editor

Permet l'écriture des spécifications (au format YAML ou JSON) et de générer le code source correspondant (serveur & client, langages & framework)

<https://editor.swagger.io/>

GÉNÉRATION DU FICHER: *Bottom-Top*

The image shows a Swagger UI interface with a Swagger 2.0 API definition on the left and a rendered API view on the right.

Swagger 2.0 Definition (Left Panel):

```
1 swagger: '2.0'
2 info:
3   description: Global description for basic application demo
4   version: '1'
5   title: swagger-ui-integration demo
6   host: 'localhost:8080'
7   basePath: '/simple/api'
8   tags:
9     - name: person
10      description: Action on person !!
11   schemes:
12     - http
13     - https
14   paths:
15     /person:
16       get: {}
17       post: {}
18     '/person/{personId}':
19       get: {}
20       put: {}
21       delete: {}
22   definitions:
23     Person:
24       type: object
25       properties:
26         id:
27           type: integer
28           format: int32
29           xml:
30             attribute: true
31         firstname:
32           type: string
33         lastname:
34           type: string
35       xml:
36         name: person
```

Rendered API View (Right Panel):

swagger-ui-integration demo

Global description for basic application demo

Version 1

Filter operations by a tag:

person Action on person !!

Paths

- /person**
 - GET /person
 - POST /person
- /person/{personId}**
 - GET /person/{personId}
 - PUT /person/{personId}
 - DELETE /person/{personId}

Models

Person

```
Person {
  id: integer
  firstname: string
  lastname: string
}
```

SWAGGER

GÉNÉRATION DU FICHER : *Top-Bottom*

Code as specification

Différents tools existe (pour différent langages & framework) qui permette d'interpreter une API REST exposer et d'en générer le fichier OpenAPI (ou swagger selon la version).

SWAGGER

GÉNÉRATION DU FICHIER: *Top-Bottom*

```
@RequestMapping(value = ..., produces = {APPLICATION_JSON_VALUE})
@Api(value = "Utilisateurs", description = "Gestion des utilisateurs")
public class UserEndpoint extends ... {
    ...
    @ApiOperation(
        value = "Liste des utilisateurs",
        notes = "Renvoi la liste des utilisateurs", code = 206,
        response = User.class, responseContainer = "List"
    )
    @ApiResponses({
        @ApiResponse(code = 200, message = "La liste contient tous",
        @ApiResponse(code = 206, message = "La liste contient les é
        @ApiResponse(code = 204, message = "La liste ne contient au
    })
    @GetMapping(produces = {APPLICATION_JSON_VALUE})
    public ResponseEntity<List<UserDTO>> getUtilisateurs()
```

Specification as code

Code as specification

Pro

Pro

Facile a initier

Au plus proche du code

Cons

Cons

Désynchronisation

Adhérence au code

SWAGGER

GÉNÉRATION DU FICHIER

Code as specification

Un bonus

SWAGGER

Code as specification

Il existe une méthode pour générer le fichier OpenAPI lors d'une phase de build (ou en mode execution)

Et aussi de transformer le fichier OpenAPI en AsciiDoc pour permettre la génération d'une page de doc HTML et/ou un fichier PDF (usage doc offline)

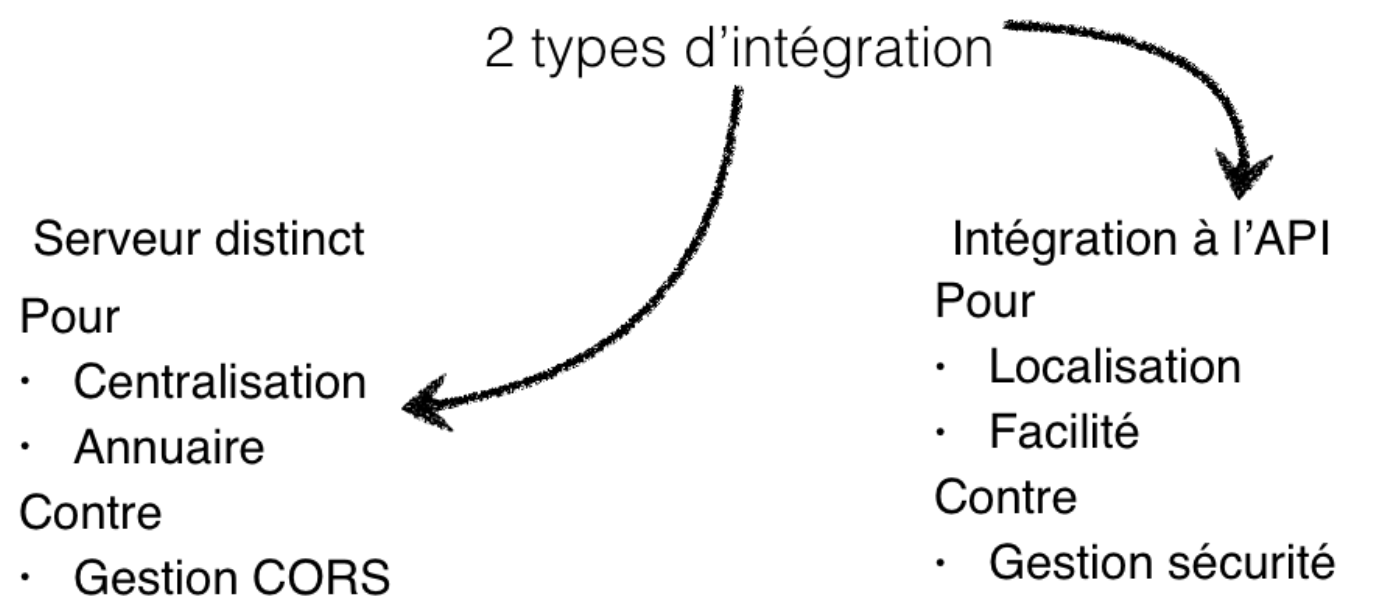
SWAGGER

Mise à disposition du fichier OpenAPI ?

SWAGGER

Mise à disposition du fichier OpenAPI

2 types d'intégration



Serveur distinct

Pour

- Centralisation
- Annuaire

Contre

- Gestion CORS

Intégration à l'API

Pour

- Localisation
- Facilité

Contre

- Gestion sécurité


SWAGGER

Exploration / Test d'une API

Swagger UI

SWAGGER

Swagger UI

 **swagger**

Explore

swagger-ui-integration demo

Global description for basic application demo

person : Action on person !!

Show/Hide | List Operations | Expand Operations

GET	/person	get list of person
POST	/person	Create person
DELETE	/person/{personId}	Delete person by Id
GET	/person/{personId}	get person by Id
PUT	/person/{personId}	Update person by Id

[BASE URL: /simple/api , API VERSION: 1]

SWAGGER

Swagger UI

Description du type

Model	Model Schema
Inline Model [
Person	
]	
Person {	
id (integer, optional),	
firstname (string, optional),	
lastname (string, optional)	
}	

GET

/person/{personId}

get person by Id

Response Class (Status 200)

success

Model

Model Schema

```
{
  "id": 0,
  "firstname": "string",
  "lastname": "string"
}
```

Response Content Type

application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
personId	(required)		path	integer

Response Messages

HTTP Status Code	Reason	Response Model	Headers
404	Person not found		

SWAGGER

Swagger UI

POST **/person** Create person

Parameters

Parameter	Value	Description	Parameter Type	Data Type
firstname	<input type="text"/>		formData	string
lastname	<input type="text"/>		formData	string

Response Messages

HTTP Status Code	Reason	Response Model	Headers
default	successful operation		

Try it out!

Swagger UI

GET /person get list of person

Implementation Notes
if no person already persited, provide a empty list

Response Class (Status 200)
successful operation

Model | **Model Schema**

```
[
  {
    "id": 0,
    "firstname": "string",
    "lastname": "string"
  }
]
```

Response Content Type application/json

[Try it out!](#)

Essayer l'API !!



SWAGGER

Swagger UI

[Try it out!](#) [Hide Response](#)

Curl ← Parce que la console, y'a que ça de vrai

```
curl -X GET --header 'Accept: application/json' 'http://localhost:8080/simple/api/person'
```

Request URL ← Pour essayer dans un navigateur ou REST tools

```
http://localhost:8080/simple/api/person
```

Response Body ← Résultat de la requête

```
[]
```

Response Code ← Code HTTP Résultat de la requête

```
200
```

Response Headers ← Et les headers !!!!!

```
{
```