

Nurturing REST API References

Joaquim Baptista
September 2023

How can we make Open API specifications useful?

- Identify the roles of Open API in the REST API ecosystem.
- Identify how Open API descriptions address different developer decisions.
- Visualize relationships within the API.
- Recognize incomplete specifications.

Or, the subtle art
of answering questions
before they are asked.

PART 1:

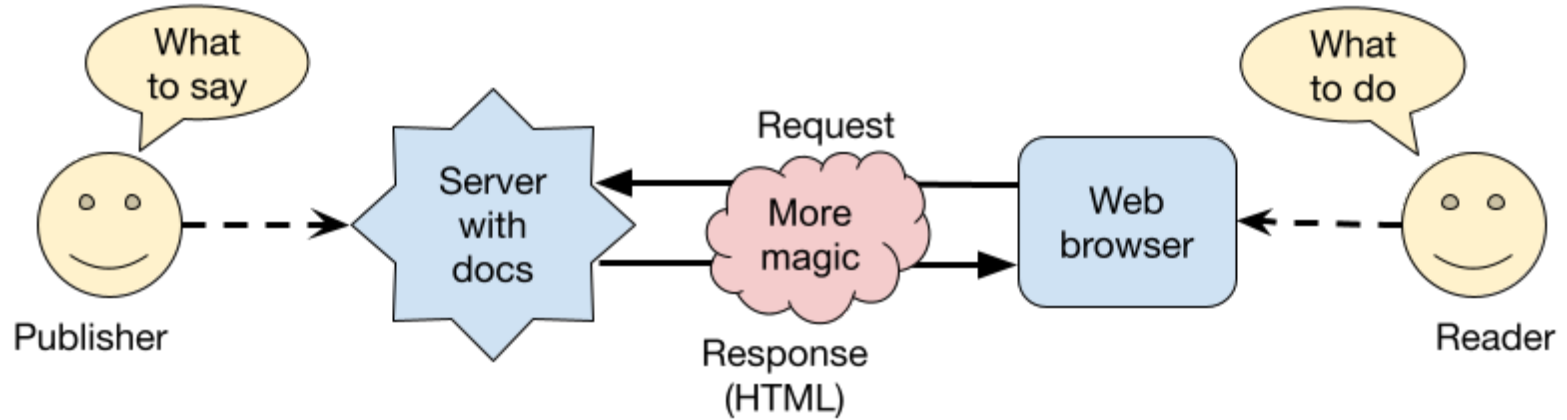
WHAT IS REST AND OPEN API

PhD by Roy Fielding (2000): Architectural Styles and the Design of Network-based Software Architectures.

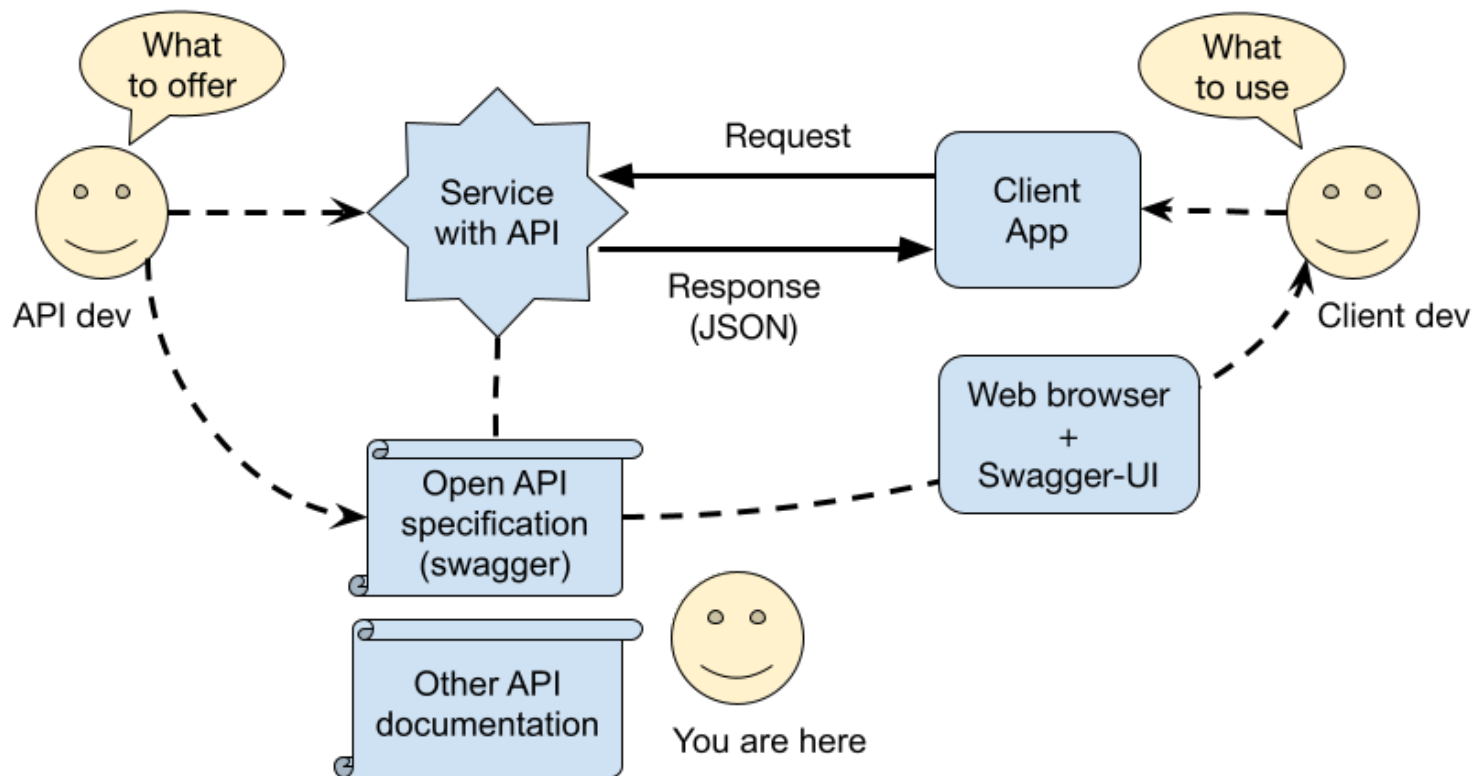
Distributed hypermedia system, enables further sophistication.

- Representational State Transfer (REST).
- Resource (has a name aka URL).
- Representation of resource (data with media-type and more metadata).
- Stateless operations (each request contains all of the information necessary for a connector to understand the request, independent of any requests that may have preceded it).
- Proxies and cache servers (for large sites).

Large-scale Web browsing?



REST APIs are like the Web for machines



API Economy

The API economy refers to the controlled exchange of digital data and services through APIs.

- API producers expose services to extend reach, market share, and monetization opportunities.
- API consumers use best-of-breed technology without the investment.

Well-known examples:

- Amazon Web Services, 2002 — Infrastructure as a service, aka Cloud computing.
- Google Maps, 2006 — Maps, routes, and places.
- Twilio, 2007 — Messaging and contact centers.
- Stripe, 2011 — Payment processing.

Open API tool ecosystem

Explore and interact

Given a specification, use tools (such as “Swagger UI” or Postman) to explore and interact with the API.

Generate code

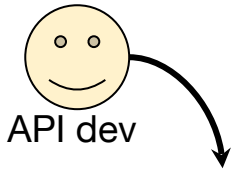
Given a specification, use “codegen” tools to automatically generate client and server code in multiple programming languages.

Generate a specification

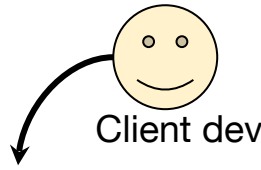
Given code with comments, automatically generate a specification (introspection). Code and comments must follow conventions.

Validate implementations

Given a specification, validate HTTP requests and responses, or server implementations (introspection).



Open API and Swagger-UI



```
Swagger Editor
Supported by SMARTBEAR

116 - write:pets
117 - read:pets
118 - /pet/findByStatus:
119 - get:
120 - tags:
121 - pet
122 summary: Finds Pets by status
123 description: Multiple status values can be provided with comma separated strings
124 operationId: findPetsByStatus
125 parameters:
126 - name: status
127 in: query
128 description: Status values that need to be considered for filter
129 required: false
130 explode: true
131 schema:
132 type: string
133 default: available
134 enum:
135 - available
136 - pending
137 - sold
138 responses:
139 '200':
140 description: successful operation
141 content:
142 application/json:
143 schema:
144 type: array
145 items:
146 $ref: '#/components/schemas/Pet'
147 application/xml:
148 schema:
149 type: array
150 items:
151 $ref: '#/components/schemas/Pet'
152 '400':
153 description: Invalid status value
154 security:
155 - petstore_auth:
156 - write:pets
```



pet Everything about your Pets [Find out more](#)

PUT /pet Update an existing pet

POST /pet Add a new pet to the store

GET /pet/findByStatus Finds Pets by status

Multiple status values can be provided with comma separated strings

Parameters

Try it out

Name	Description
status	Status values that need to be considered for filter
string	Available values : available, pending, sold
(query)	Default value : available

Responses

Code	Description	Links
------	-------------	-------

Flow of decisions of the client developer

Using Swagger-UI to explore an API while coding a client application:

1. Choose the correct tag.
2. Select the correct operation to call.
3. Choose, collect or create the parameters for calling the operation.
4. Understand the operation response.

Many questions, many doubts...
to be clarified in Markdown descriptions.

Step 1 (tag): Where is the functionality that I need?

Answer in the tag descriptions:

- Distinguish between what can be done in each tag.
- Overview of can be done in the tag.

The image shows the Swagger Editor interface. On the left, a dark-themed code editor displays a YAML definition for an API. The definition includes servers, tags, externalDocs, and paths. The right panel shows a light-themed visual representation of the API, with a dropdown menu for the selected server and a list of tags with their descriptions and external documentation links.

Swagger Editor (Supported by SMARTBEAR)

File ▾ Edit ▾ Insert ▾ Generate Server ▾ Generate Client ▾ About ▾ [Try our new Editor ↗](#)

```
25 servers:
26   - url: https://petstore3.swagger.io/api/v3
27 tags:
28   - name: pet
29     description: |
30       Everything
31       about your Pets
32   externalDocs:
33     description: Find out more
34     url: http://swagger.io
35   - name: store
36     description: Access to Petstore orders
37   externalDocs:
38     description: Find out more about our store
39     url: http://swagger.io
40   - name: user
41     description: Operations about user
42 paths:
43   /pet:
44     put:
```

Servers

▾ [Authorize](#) 🔒

pet Everything about your Pets [Find out more](#) ▾

store Access to Petstore orders [Find out more about our store](#) ▾

user Operations about user ▾

Step 2 (operations): Which of these operations should I use?

Answer in the operation summaries:

- Distinguish between operations in each tag.
- Short sentence to fit space in Swagger-UI.

If needed, continue in the operation descriptions:

- Further detail of overall purpose if needed.
- Issues that may surprise the developer.

The image shows a side-by-side comparison of Swagger Editor and Swagger-UI. On the left, the Swagger Editor interface displays a JSON definition for a pet store API. The definition includes a `paths` object with endpoints `/pet` and `/pet/findByStatus`. The `/pet` endpoint has two operations: `put` (update an existing pet) and `post` (add a new pet). The `/pet/findByStatus` endpoint has a `get` operation (find pets by status). The `put` operation includes a `requestBody`, `responses`, and `security` field. The `post` operation includes a `requestBody`, `responses`, and `security` field. The `get` operation includes a `responses` field. The `put` operation includes a `requestBody`, `responses`, and `security` field. The `post` operation includes a `requestBody`, `responses`, and `security` field. The `get` operation includes a `responses` field.

On the right, the Swagger-UI interface displays the same API definition in a user-friendly format. The title is "pet Everything about your Pets" with a "Find out more" link. The operations are listed in a table with columns for the HTTP method, the endpoint, the description, and a dropdown menu. The operations are:

Method	Endpoint	Description	Dropdown
PUT	/pet	Update an existing pet	Dropdown with lock icon
POST	/pet	Add a new pet to the store	Dropdown with lock icon
GET	/pet/findByStatus	Finds Pets by status	Dropdown with lock icon
GET	/pet/findByTags	Finds Pets by tags	Dropdown with lock icon
GET	/pet/{petId}	Find pet by ID	Dropdown with lock icon
POST	/pet/{petId}	Updates a pet in the store with form data	Dropdown with lock icon

Step 3 (parameters): What parameters do I need?

Questions:

- Is a parameter value needed? Default value? Default behavior?
- Where do I get a valid value? Or, how do I create a valid value?
- For enumerations, how do I choose? What are the consequences of each choice?

The image shows the Swagger Editor interface. On the left, the OpenAPI definition for the `/pet/findByStatus` endpoint is displayed in a code editor. The definition includes a `get` method with a `status` query parameter of type `string` and an enumeration of possible values: `available`, `pending`, and `sold`. The parameter has a default value of `available`.

```
118 /pet/findByStatus:
119   get:
120     tags:
121       - pet
122     summary: Finds Pets by status
123     description: Multiple status values can be provided with comma
124       separated strings
125     operationId: findPetsByStatus
126     parameters:
127       - name: status
128         in: query
129         description: Status values that need to be considered for filter
130         required: false
131         explode: true
132         schema:
133           type: string
134           default: available
135           enum:
136             - available
137             - pending
138             - sold
```

On the right, the visual representation of the API is shown. The endpoint `/pet/findByStatus` is selected, and the `Parameters` tab is active. The `status` parameter is listed with its description and the available values. A dropdown menu is shown with the value `available` selected.

Multiple status values can be provided with comma separated strings

Parameters

Name	Description
status	Status values that need to be considered for filter Available values : available, pending, sold Default value : available

string (query) available

Step 4 (schemas): What does the response mean?

Answer in the the object descriptions:

- What does the object represent?

Answer in the property descriptions:

- How to understand the value in a response.
- How to use the value in a response.
- If a property is absent or null, what does it mean?

The screenshot displays the Swagger Editor interface, which is used for defining and visualizing API schemas. The interface is split into two main panels.

Left Panel (Schema Definition): This panel shows the JSON Schema for a `Pet` object. The schema is defined as follows:

```
Pet:
  required:
    - name
    - photoUrls
  type: object
  properties:
    id:
      type: integer
      format: int64
      example: 10
    name:
      type: string
      example: doggie
    category:
      $ref: '#/components/schemas/Category'
    photoUrls:
      type: array
      items:
        type: string
    tags:
      type: array
      items:
        type: string
    status:
      type: string
      description: pet status in the store
      enum:
        - available
        - pending
        - sold
```

Right Panel (Response Visualization): This panel shows the result of a successful operation (200 status code). It displays the media type as `application/json` and provides an example value for the `Pet` object:

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "http://example.com/pet-photo-1.jpg"
  ],
  "tags": [
    "pet"
  ],
  "status": "available"
}
```

The right panel also includes a dropdown menu for selecting the media type and a button to view the schema.

Also: Examples in descriptions

Open API 3 and JSON Schema have confusing variations on example and examples.

- Put examples in descriptions.
- Explain the meaning of the example values.

currency:

description: |

ISO 4217 currency code of the amount.

For example,

'USD' for United States Dollar.

Also: Describe enums in descriptions

Values of enumerations do not have their own descriptions. Consider a Markdown list.

plan:

type: string

enum:

- silver

- priority

- monthly

description: |

Promotional plan contracted with store.

* `silver` - Frequent pet care,
including washing.

* `priority` - Priority service.

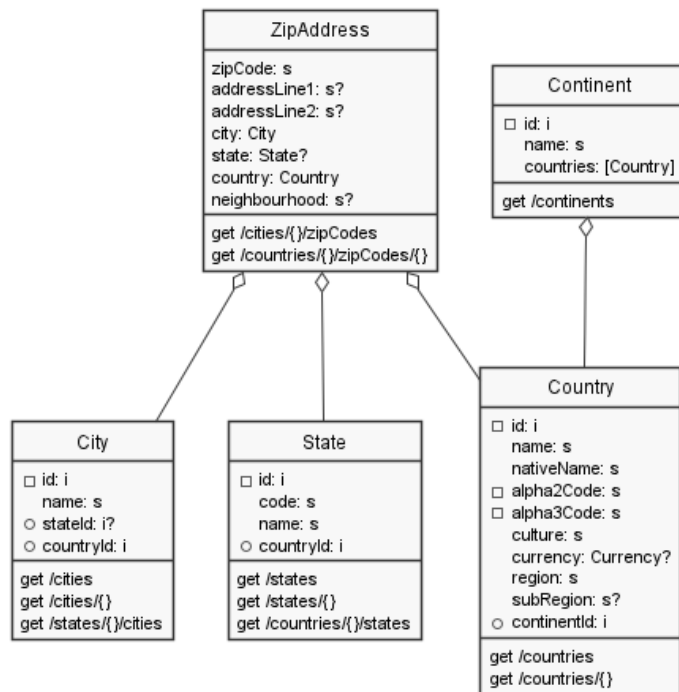
* `monthly` - Schedule day each month
for pet care.

Tags as UMLish diagrams



```
@startuml
object Pet {
  id: n
  name: s
  petType: s
  color: s
  gender: s
  breed: s
  ---
  post /pets
  get /pets
  get /pets/{}
  put /pets/{}
  delete /pets/{}
}
@enduml
```

Example: Geography at FARFETCH

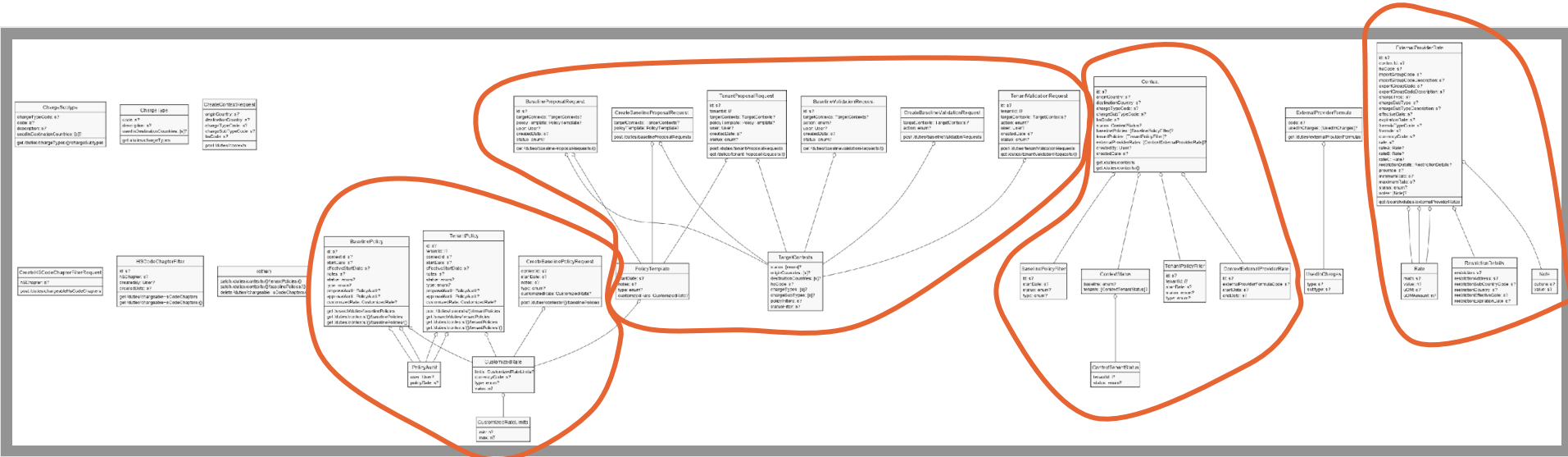


Arrows denote a sub-object.

```
@startuml
. . .
ZipAddress o-- City
ZipAddress o-- State
ZipAddress o-- Country
Continent o-- Country
@enduml
```

How to chunk tags

1. Create PlantUML diagram for API or obvious parts of it.
2. Refine diagram to make structure more obvious.
3. Partition diagram in parts, minimizing the number of lines cut.



Breaking changes

*If we don't document clearly, we are not accountable for the changes.
— Developer myth*

Managing API change:

- Add new, improve old features.
- Minimal impact on existing clients.

A breaking change is any change that can break a client application, usually:

- Deleting part of an API.
- Modify part of existing API.

Structural issues:

- Properties and some parameters are optional by default.
- Optional properties and parameters need default value or behavior.
- Enumerations are absent.
- Objects are not defined.
- Objects have no name.

Making a specification tighter
is a breaking change.

But are we changing the API
or the description of the API?

USE REFERENCES TO CAPTURE DETAIL

Walk in the shoes of client developers
to write useful descriptions.

If the API is large enough, everyone needs a map.

You may find yourself to be the guardian of the standard.

BE A PART OF THE ECOSYSTEM!

