OpenApi

Full Stack Web Development

Magic Quadrant for Full Life Cycle API Management

Published 28 September 2021 - ID G00735998 - 59 min read

By Shameen Pillai, Kimihiko Iijima, and 4 more

Demand for APIs, fueled by new business models, application development trends and the urgency of modernization, is directing software engineering leaders to the diverse, evolving market for full life cycle API management products. This research assesses 17 vendors to help you make the right choice.

Market Definition/Description

Gartner defines the full life cycle application programming interface (API) management market as the market for software that supports all stages of an API's life cycle, namely planning and design, implementation and testing, deployment and operation, and versioning and retirement.

Central to full life cycle API management offerings' capabilities is support in the following functional areas:

- **Developer portals:** A self-service catalog of APIs for enabling, marketing to, and governing ecosystems of developers who produce and consume APIs.
- **API gateways:** Runtime management, security and usage monitoring for APIs.
- **Policy management and analytics:** Security configuration, API mediation and API usage analytics.
- **API design and development:** A meaningful developer experience and tools for designing and building APIs, and for API-enablement of existing systems.
- **API testing:** From basic mock testing to advanced functional, performance and security testing of APIs.

As noted in the following assessments within this Magic Quadrant, some vendors specialize in subsets of the API life cycle, such as runtime API gateways and API testing, while others focus on the entire life cycle.

Forbes

Jan 18, 2020, 03:37pm EST | 28,869 views

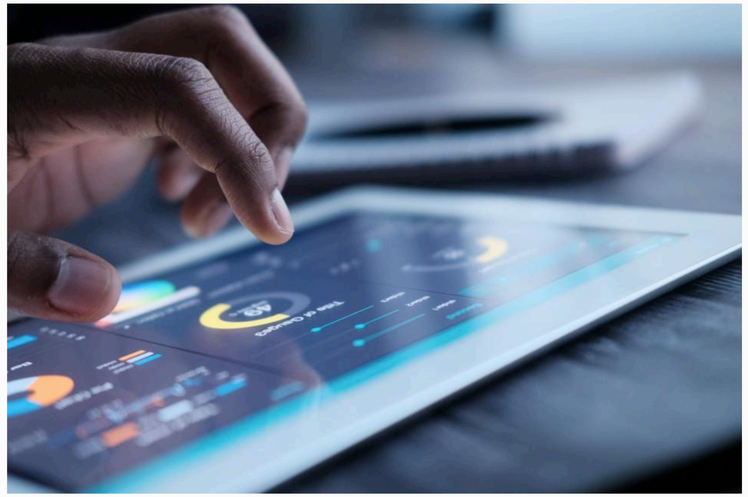
API Economy: Is It The Next Big Thing?

Tom Taulli Former Contributor ⓘ
Entrepreneurs
I write about tech & finance.

Follow

Listen to article 6 minutes

This article is more than 2 years old.



null GETTY

APIs (Application programming interfaces) have been around for decades. "They allow different systems to talk to each other in a seamless, fast fashion," said Gary Hoberman, who is the CEO of Unqork.

Yet it's been during the past decade that this technology has become a major force. For example, in early 2019 Salesforce.com shelled out $6.5 billion for MuleSoft, a management system for APIs. Then there was another notable deal that actually was announced this week: Visa agreed to pay $5.3 billion for Plaid (here's my post about the acquisition in Forbes.com).

How the API Economy is Changing the Business World

Vonage Staff

We are in the midst of a technological and cultural revolution driven by cloud, mobility, SaaS, big data, social and the internet of things (IoT), and companies must re-invent themselves by creating new software-defined business models at a rapid pace or end up being left in the dust. Plus, customer expectations are higher than ever - people simply expect everything to be always-on, personalized, dynamic and highly mobile. Software, device, and system integration along with application programming interfaces (APIs) are the glue that makes it all work.

Enter the API Economy

While APIs that allow various software applications and systems to communicate with and offer services to each other are not new and have been around for at least 20 years, the API economy is surging today. According to ProgrammableWeb, there are more than 17,000 APIs available as of the first quarter of 2017. That's up by a whopping 758% since 2010. The uses are widespread in all industries extending beyond IT teams to all levels of the organization. One of the biggest advantages of leveraging third-party APIs is enabling business leaders that may not be too technical to work with IT to develop new and more scalable strategies while minimizing time to market with new and innovative products and services.

Open API / Swagger

- The OpenAPI Specification (also known as Swagger) defines a standard, language-agnostic interface to RESTful APIs.

- Advantages :

  - All consumers of the service can understand and interact with the service with a minimal amount of implementation logic.

  - Automatic code generation for clients and servers

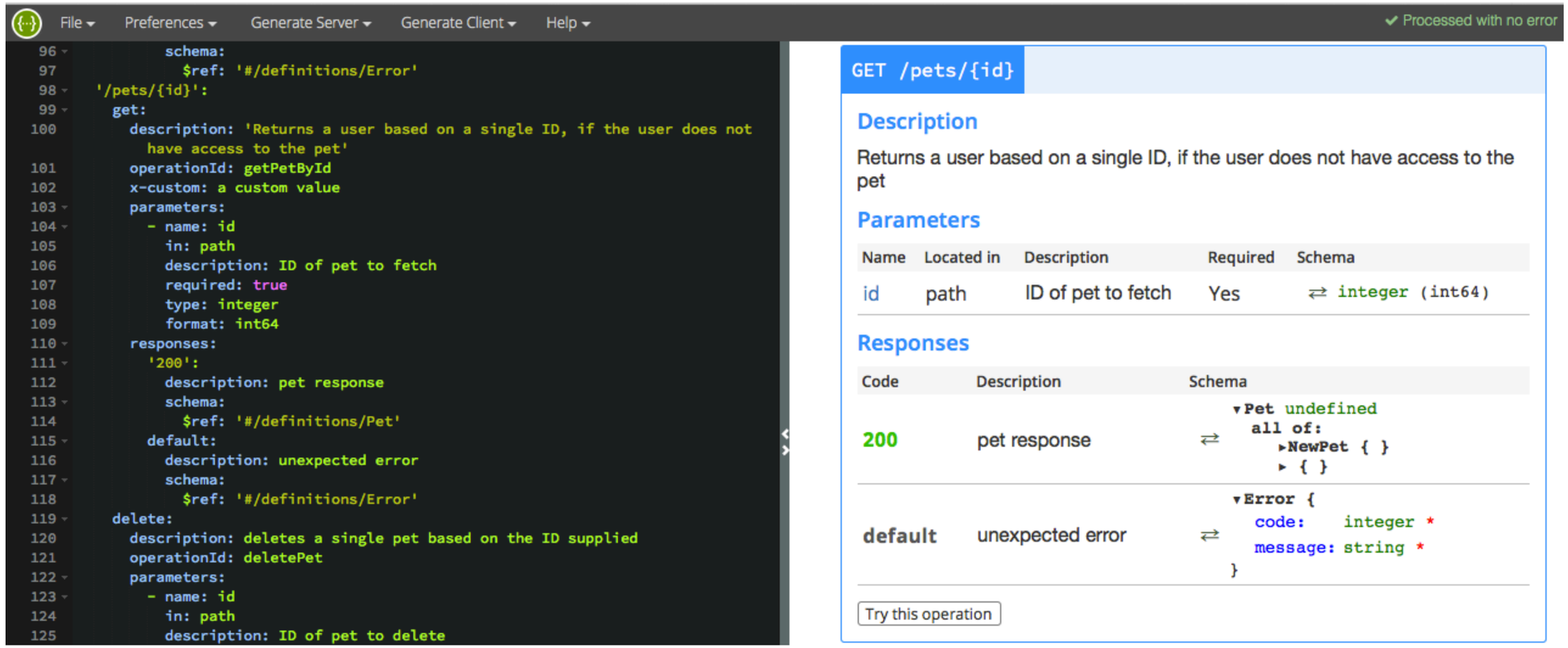
Single Specification Format

- Using an API definition language such as OpenAPI specification helps to describe easily and quickly an API.
- Being a simple text file, the OpenAPI specification file can be shared and managed within GitHub
- Once written, OpenAPI specification file can also be used as:
  - source material for documentation
  - specification for developers
  - partial or complete code generation

```yaml
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons.
      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string
```

API Editors

- An OpenAPI specification is a simple text file which can be edited with any text editor
- Can also be edited by specialised editor : *Swagger Editor*



https://openapi.tools

Tool Types

We've organised everything into categories so you can jump to the section you're interested in.

- **Auto Generators:** Tools that will take your code and turn it into an OpenAPI Specification document
- **Converters:** Various tools to convert to and from OpenAPI and other API description formats.
- **Data Validators:** Check to see if API requests and responses are lining up with the API description.
- **Description Validators:** Check your API description to see if it is valid OpenAPI.
- **Documentation:** Render API Description as HTML (or maybe a PDF) so slightly less technical people can figure out how to work with the API.
- **DSL:** Writing YAML by hand is no fun, and maybe you don't want a GUI, so use a Domain Specific Language to write OpenAPI in your language of choice.
- **GUI Editors:** Visual editors help you design APIs without needing to memorize the entire OpenAPI specification.
- **Learning:** Whether you're trying to get documentation for a third party API based on traffic, or are trying to switch to design-first at an organization with no OpenAPI at all, learning can help you move your API spec forward and keep it up to date.
- **Miscellaneous:** Anything else that does stuff with OpenAPI but hasn't quite got enough to warrant its own category.
- **Mock Servers:** Fake servers that take description document as input, then route incoming HTTP requests to example responses or dynamically generates examples.
- **Parsers:** Loads and read OpenAPI descriptions, so you can work with them programmatically.
- **SDK Generators:** Generate code to give to consumers, to help them avoid interacting at a HTTP level.
- **Security:** By poking around your OpenAPI description, some tools can look out for attack vectors you might not have noticed.
- **Server Implementations:** Easily create and implement resources and routes for your APIs.
- **Testing:** Quickly execute API requests and validate responses on the fly through command line or GUI interfaces.
- **Text Editors:** Text editors give you visual feedback whilst you write OpenAPI, so you can see what docs might look like.

APIs you won't hate

Blog Books Jobs Podcast **Join the Community**

All Articles

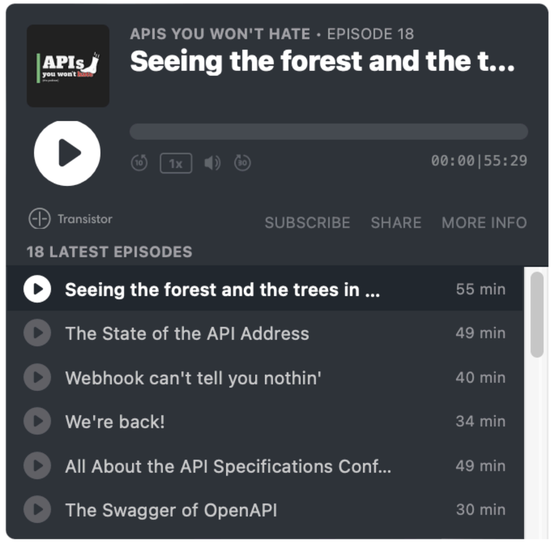
FEATURED ARTICLE

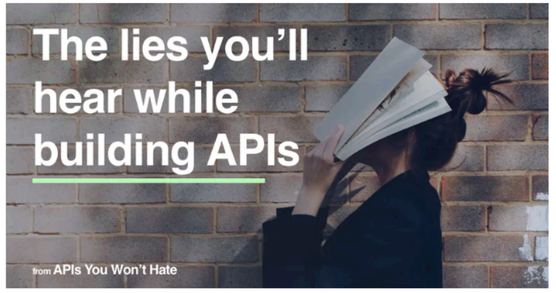
Contract Testing a Laravel API with OpenAPI

written by Phil Sturgeon

Keeping your API and OpenAPI in sync doesn't have to be complicated, the Laravel PHP edition.

Read more →





Eight of the Biggest Lies in APIs

Matthew Reinbold Jan 12, 2022

In the API space, numerous statements are context-dependent, misunderstandings, or outright lies. Here are some of the biggest the APIs You Won't Hate community has heard to date.

Creating OpenAPI from HTTP Traffic

Phil Sturgeon Jan 01, 2022

If you want to get into API design-first, but awkwardly already have a bunch of APIs with no OpenAPI, use a HTTP proxy like Akita to create OpenAPI from your HTTP traffic!

```json
{
    "swagger": "2.0",
    "info": {
        "version": "1.0.0",
        "title": "Simple API",
        "description": "A simple API to learn how to write OpenAPI Specif
    },
    "schemes": [
        "https"
    ],
    "host": "simple.api",
    "basePath": "/openapi101",
    "paths": {
        "/persons": {
            "get": {
                "summary": "Gets some persons",
                "description": "Returns a list containing all persons.",
                "responses": {
                    "200": {
                        "description": "A list of Person",
                        "schema": {
                            "type": "array",
                            "items": {
                                "properties": {
                                    "firstName": {
                                        "type": "string"
                                    },
                                    "lastName": {
                                        "type": "string"
                                    },
                                    "username": {
                                        "type": "string"
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Specification format:

Json

OR

Yaml

```yaml
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons.
      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string
```

- YAML consider more easy to write and read for humans
- Easy to convert between formats

OpenAPI Specification

- Version of Spec: swagger: version of the OpenAPI specification
- Information: The API's version (not to be confused with the specification version) a title and an optional description.
- API URL: scheme (protocol), host and path for endpoints
- API Operations: specified in paths (empty for the moment)

```yaml
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths: {}
```

Spec for API to return list of persons

```yaml
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons.
      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string
```

Adding a path

- In the paths section a path /persons corresponding to the persons resource

```
paths:
  /persons:
```

Adding an http method on path

- On each path add any http verb (like get, post, put or delete) to manipulate the corresponding resource.
- To list some persons, apply the get http method to the /persons resource (or path).
- Short description (summary) and a longer one if necessary (description).

```
get:
  summary: Gets some persons
  description: Returns a list containing all persons.
```

Describing response

- For each operation, describe any response matching an http status code (like 200 OK or 404 Not Found) in the responses section.

- Only handle 200 when responding to get /persons - tell what the response means via its description.

```
responses:
  200:
    description: A list of Person
```

Describing response's content

- The get /persons operation returns a list of persons, describe what this is with the schema section of the response.

- A list of person is an object which type is array.

- Each item in this array is an object containing three properties of type string: firstName, lastName and username.

- Only username will be always provided (i.e. required).

```yaml
schema:
  type: array
  items:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
```

Defining a path parameter

- Access directly a specific person by it’s username
- Add a get /persons/{username} operation to our API.
- {username} is called a path parameter.

```yaml
/persons/{username}:
  get:
    summary: Gets a person
    description: Returns a single person for its username
    parameters:
      - name: username
        in: path
        required: true
        description: The person's username
        type: string
    responses:
      200:
        description: A Person
        schema:
          required:
            - username
          properties:
            firstName:
              type: string
            lastName:
              type: string
            username:
              type: string
      404:
        description: The Person does not exists.
```

Adding a get /persons/{username} operation

- Add a /persons/{username} path in the paths section and define the get operation for this path.

```
/persons/{username}:
  get:
    summary: Gets a person
    description: Returns a single person for its username
```

Describing username path parameter

- As {username} is a path parameter, it must be described.
- Add parameters section to the get operation + adding a required parameter with a name matching the parameter defined in the path (here username) located in path of type string.
- Provide an (optional) description.

```yaml
parameters:
  - name: username
    in: path
    required: true
    description: The person's username
    type: string
```

Adding responses

- 200 OK response returning a person.
- As a username may not match an existing person we also add a 404 response.
- In this case, do not return anything besides the 404 http status code

```
    responses:
      200:
        description: A Person
        schema:
          required:
            - username
          properties:
            firstName:
              type: string
            lastName:
              type: string
            username:
              type: string
```

```
      404:
        description: The Person does not exists.
```

Adding post /persons operation

- Add a post method to the /persons path (after the *get* one) in the *paths* section

```
  post:
    summary: Creates a person
    description: Adds a new person to the persons list.
```

Describing a person body parameter

- Then define a parameter *named* person located in *body* of *type* object.
- The person object's is described via it's *schema*.
- The firstName and lastName attributes are optional and username is *required*

```yaml
    parameters:
      - name: person
        in: body
        description: The person to create.
        schema:
          required:
            - username
          properties:
            firstName:
              type: string
            lastName:
              type: string
            username:
              type: string
```

```yaml
  responses:
    204:
      description: Persons succesfully created.
    400:
      description: Persons couldn't have been created.
```

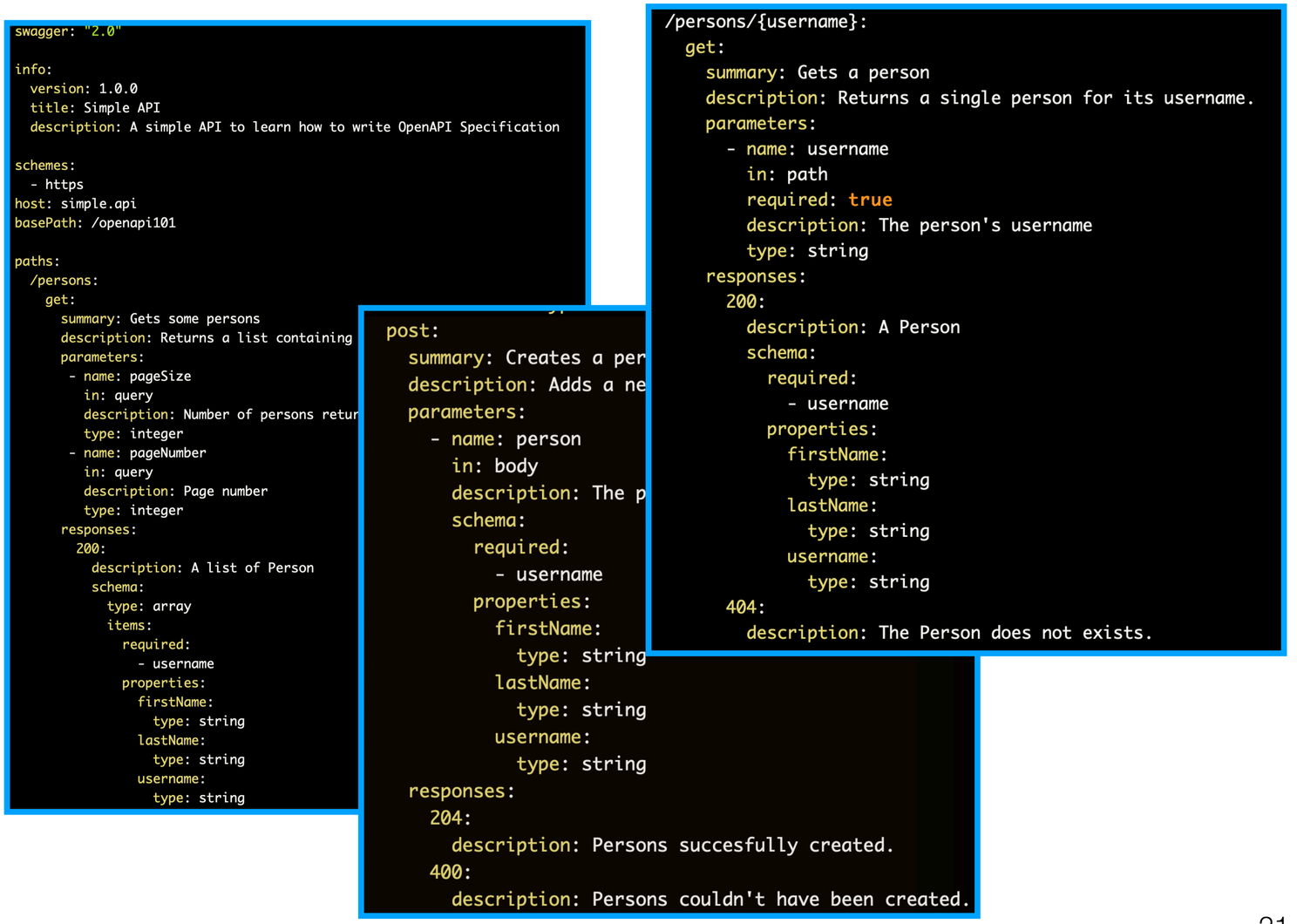
```yaml
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
      responses:
        200:
          description: A list of Person
          schema:
            type: array
            items:
              required:
                - username
              properties:
                firstName:
                  type: string
                lastName:
                  type: string
                username:
                  type: string
```

```yaml
post:
  summary: Creates a person
  description: Adds a new person to the persons list.
  parameters:
    - name: person
      in: body
      description: The person to create.
      schema:
        required:
          - username
        properties:
          firstName:
            type: string
          lastName:
            type: string
          username:
            type: string
  responses:
    204:
      description: Persons succesfully created.
    400:
      description: Persons couldn't have been created.
```

```yaml
/persons/{username}:
  get:
    summary: Gets a person
    description: Returns a single person for its username.
    parameters:
      - name: username
        in: path
        required: true
        description: The person's username
        type: string
    responses:
      200:
        description: A Person
        schema:
          required:
            - username
          properties:
            firstName:
              type: string
            lastName:
              type: string
            username:
              type: string
      404:
        description: The Person does not exists.
```

Simplifying data model description

- Verbose Schema
- The standard has mechanism for simplifying the specification
- Reduce repetition, reuse **definitions**



Adding definitions section

- Add a new *definitions* section at the end of the document

```
definitions:
```

Defining a reusable definition

- Define a Person once
- The information provided in Person are exactly the same as the one provided in the three schemas describing a person in previous version.
- A definition is simply a named schema object.

```yaml
definitions:
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
```

Referencing a definition from another definition

- Use *definition* by referencing it in another one.
- As we have defined a *Person* we'll also define *Persons* which is an array (or a list) of *Persons*.
- The information provided in *Persons* are *almost* the same as the one provided in the *get /persons* response:
- The only difference is that the schema describing the array's items has been replaced by a reference ($ref) to the Persons definition.

```
description: A list of Person
schema:
  type: array
  items:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
```

```
Persons:
  type: array
  items:
    $ref: "#/definitions/Person"
```

Using definitions in responses

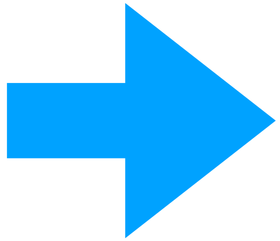
- Once we have defined *Person* and *Persons*, we can use them to replace the inline schemas by references in all operations responses.

```yaml
  responses:
    200:
      description: A list of Person
      schema:
        type: array
        items:
          required:
            - username
          properties:
            firstName:
              type: string
            lastName:
              type: string
```

```yaml
  responses:
    200:
      description: A list of Person
      schema:
        $ref: "#/definitions/Persons"
```

get /persons/{username}

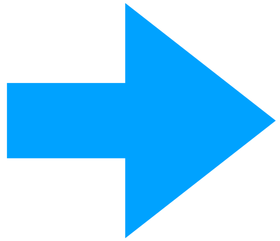
```yaml
responses:
  200:
    description: A Person
    schema:
      required:
        - username
      properties:
        firstName:
          type: string
        lastName:
          type: string
        username:
          type: string
```



```yaml
responses:
  200:
    description: A Person
    schema:
      $ref: "#/definitions/Person"
```

Using definitions in parameters

```yaml
post:
  summary: Creates a person
  description: Adds a new person to the persons list.
  parameters:
    - name: person
      in: body
      description: The person to create.
      schema:
        required:
          - username
        properties:
          firstName:
            type: string
          lastName:
            type: string
          username:
            type: string
```



```yaml
post:
  summary: Creates a person
  description: Adds a new person to the persons list.
  parameters:
    - name: person
      in: body
      description: The person to create.
      schema:
        $ref: "#/definitions/Person"
```

```yaml
swagger: "2.0"

info:
  version: 1.0.0
  title: Simple API
  description: A simple API to learn how to write OpenAPI Specification

schemes:
  - https
host: simple.api
basePath: /openapi101

paths:
  /persons:
    get:
      summary: Gets some persons
      description: Returns a list containing all persons. The list suppor
      parameters:
       - name: pageSize
         in: query
         description: Number of persons returned
         type: integer
       - name: pageNumber
         in: query
         description: Page number
         type: integer
      responses:
        200:
          description: A list of Person
          schema:
            $ref: "#/definitions/Persons"
        500:
          $ref: "#/responses/Standard500ErrorResponse"
    post:
      summary: Creates a person
      description: Adds a new person to the persons list.
      parameters:
        - name: person
          in: body
          description: The person to create.
          schema:
            $ref: "#/definitions/Person"
      responses:
        204:
          description: Persons succesfully created.
        400:
          description: Persons couldn't have been created.
        500:
          $ref: "#/responses/Standard500ErrorResponse"
```

```yaml
  /persons/{username}:
    get:
      summary: Gets a person
      description: Returns a single person for its username.
      parameters:
        - name: username
          in: path
          required: true
          description: The person's username
          type: string
      responses:
        200:
          description: A Person
          schema:
            $ref: "#/definitions/Person"
        404:
          description: The Person does not exists.
        500:
          $ref: "#/responses/Standard500ErrorResponse"
```

```yaml
definitions:
  Person:
    required:
      - username
    properties:
      firstName:
        type: string
      lastName:
        type: string
      username:
        type: string
  Persons:
    type: array
    items:
      $ref: "#/definitions/Person"
  Error:
    properties:
      code:
        type: string
      message:
        type: string
```

APIs
you won't hate

All Articles

OpenAPI v3.1 and JSON Schema

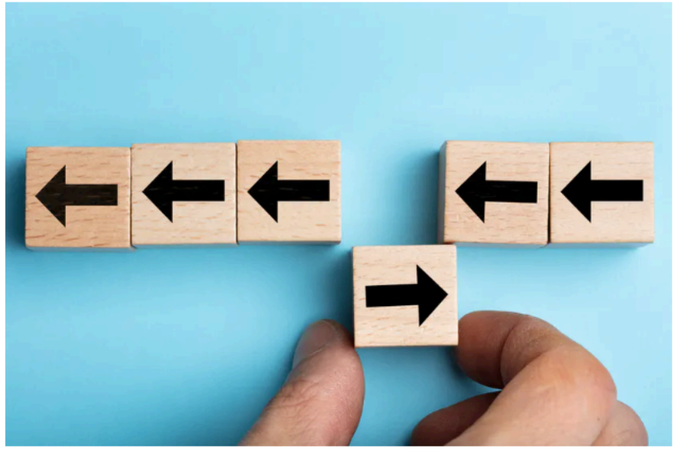
Phil Sturgeon Feb 03, 2020

JSON Schema and OpenAPI will no longer have minor differences in the schema object, but will finally be compatible from OpenAPI v3.1.

API Design-First vs Code First

Phil Sturgeon Oct 14, 2019

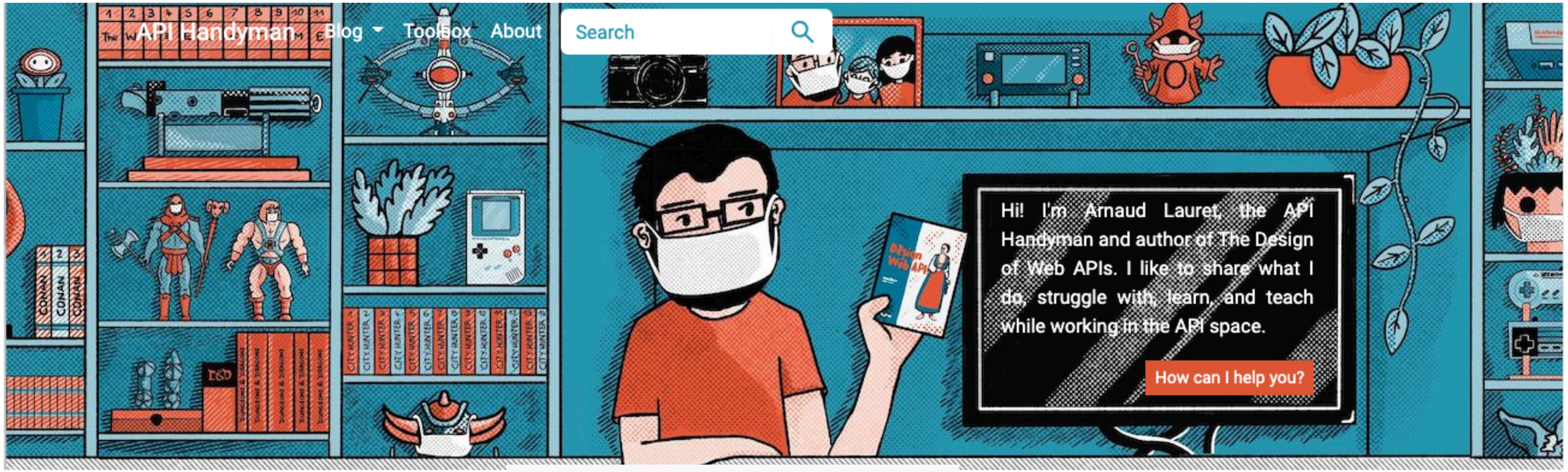
With API descriptions rising in popularity, the main question I hear folks asking about is "API Design-first" or "code-first". This is a bit of a misleading question because these are not two unique things, there are a few variants.



OpenAPI Callbacks and Webhooks

Lorna Mitchell Oct 24, 2019

What's the difference between a callback and a webhook, and which does OpenAPI support? Read on to find out the current status and upcoming changes for asynchronous and two-way APIs.


API Handyman
Blog
Toolbox
About
Search
Hi! I'm Arnaud Lauret, the API Handyman and author of The Design of Web APIs. I like to share what I do, struggle with, learn, and teach while working in the API space.
How can I help you?


The Design of Web APIs

Hacking an Elgato Key Light Series - Part 1

Hacking and reviewing Elgato Key Light API with Postman

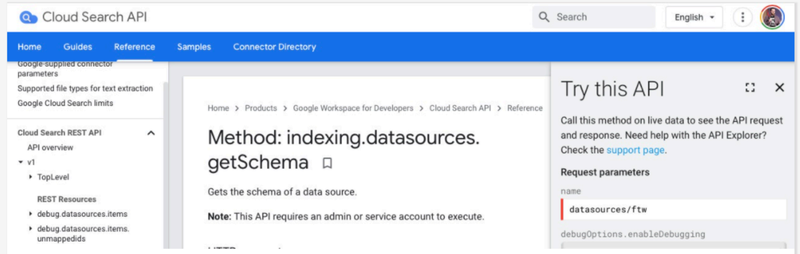
By Arnaud Lauret, February 16, 2022

Want to learn how to hack a desktop app calling an API and learn some API design principles? This post is made for you. When I got my Elgato Key Light, my first questions were: “can I control it without using the official control center using an API?” and “is the API easy to understand and use?”. Thanks to Postman’s proxy feature, I was able to easily hack the API. But I was also able to review it in the making, and there’s some interesting API design learnings to share.

Choosing between raw and processed data when designing an API

By Arnaud Lauret, February 9, 2022

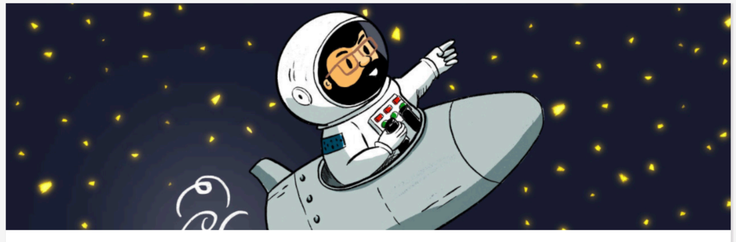
Look how trees are represented in Piet Mondrian’s paintings shown in this post’s banner. On the left they are represented in a figurative way, on the right totally abstracted. This is what we’ll discuss in this post, not figurative vs abstract painting, but raw vs processed data in API design. Choosing between raw or processed data, date of birth vs age for instance, has a direct impact on developer experience but also on provider’s.



A few concerns to be aware of when adding try it feature to API documentation

By Arnaud Lauret, January 19, 2022

“That’s neat! The developer portal/api documentation solution we chose comes with a try it feature. Everything is out of the box, we’ll have absolutely nothing to do to allow people test our APIs.” … If only that was true. Unfortunately, there are a few concerns to be aware of to actually propose a try it feature.



7 years being the API Handyman, the age of reason?

By Arnaud Lauret, February 2, 2022

This time, no “API blah blah blah”, this time, it’s personal. February 2022 marks a major landmark, I can’t believe I launched the API Handyman blog 7 years ago! Being 7 years old is supposed to mark the “age of reason”. Did I reached it? I think so, but it took me more than 7 years.

What to consider when building an API sandbox

By Arnaud Lauret, January 26, 2022

Used through the “try it” feature of an API’s documentation or directly called by a consumer application, an API sandbox is a great way to let developers play with an API and learn how it works without giving them access to the actual production environment. But what is an “API sandbox” actually? Is it just a mock? It can be that … or far more than that. Let’s see what could be the features of an API sandbox.

API design and architecture lessons from a frying pan

By Arnaud Lauret, January 5, 2022

In the kitchen, I’m the dishwasher. And lately, washing our new frying pans has got me thinking about design and API architecture issues. This post is dedicated to the person who complained that my book contained too many analogies. Sorry, but no matter what I do, read, look at, listen to, I’m always trying to see if I can make connections to APIs. And housework is no exception.

We need to talk: OpenAPI 3 is 4 years old, but Swagger 2 is still predominant

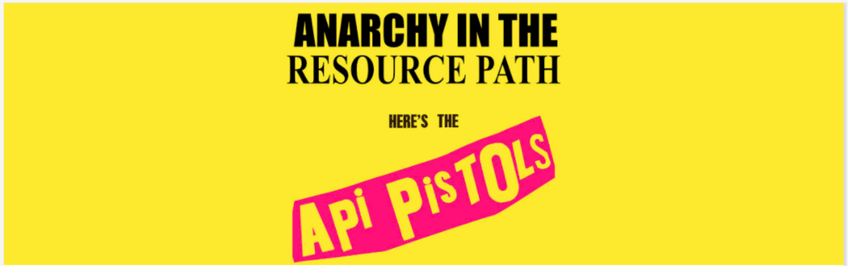
By Arnaud Lauret, November 10, 2021

While quickly doing a first scan of latest Postman State of the API Report , I did my Spock face, raising an eyebrow. Indeed, I read that after JSON Schema, "The next most popular specifications were Swagger 2.0 (54%) and OpenAPI 3.0 (40%)". To be honest and based on my own experience, it's not totally surprising, I'm still hearing/reading "can you check that Swagger" everyday. But why the 4 years old OpenAPI 3 is still struggling to surpass the good old Swagger 2?

How to choose ids and codes to build user-friendly and interoperable APIs

By Arnaud Lauret, December 29, 2021

As an API designer, why should you care about the value of a `productId`, a `countryCode`, or an error `code`? Because wisely choosing the value of such (in a broad sense) "identifiers" greatly participates in the making of a user friendly API; but most importantly an interoperable one.



Anarchy in the resource path

By Arnaud Lauret, October 20, 2021

While doing API design reviews and API design training sessions, I often see resource paths designed in an anarchic way. By anarchic, I mean their various levels seem to have been chosen randomly or some of them seem at awkward places. But why should such paths should be considered wrong? Let's see a few examples of how to not design resource paths to talk about it.

OpenApi

Full Stack Web Development