
# REST and OpenAPI

Alessandro Bocci

name.surname@unipi.it

Advanced Software Engineering (Lab)

08/10/2024

# Today’s Lab

- Introduction to RESTful APIs and OpenAPI.
- Complete Lab 1 service as a full-fledge RESTfull APIs.
- Write the OpenAPI specification of the service.

# Why APIs?

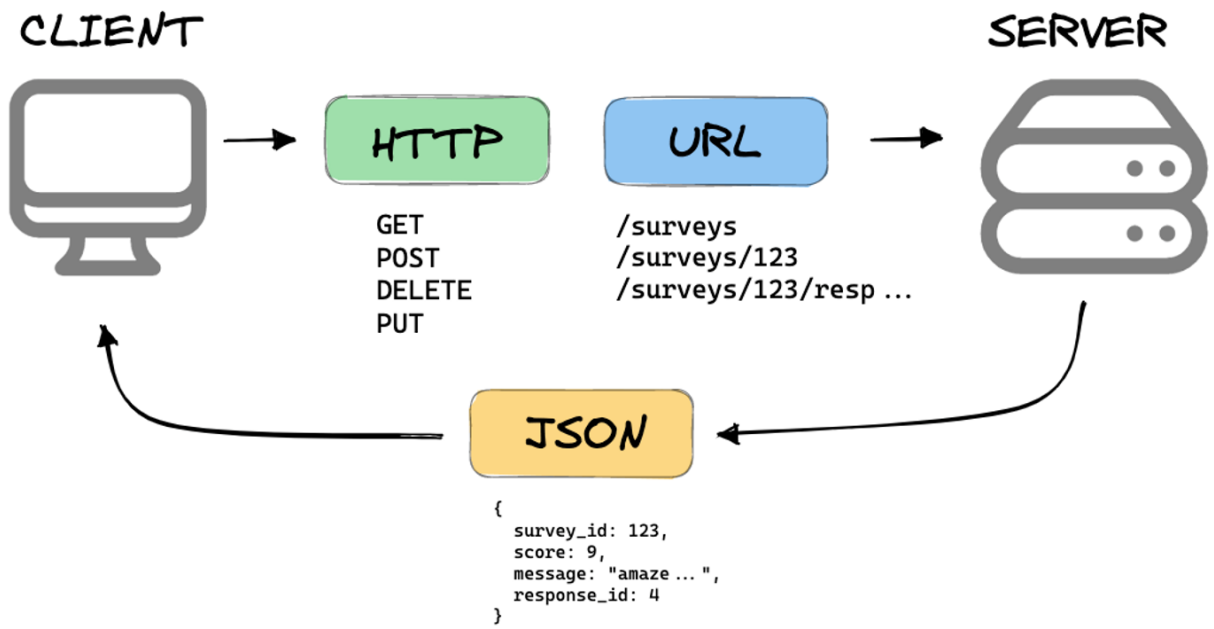
- Connect services, products, and data.
- Enable mobile/web apps, automation, integrations.
- Decouple teams & lifecycles via stable contracts.
- REST: simple, HTTP-native, widely adopted.

**Examples:** payments, maps, social, ML inference, IoT.

# REST in a Nutshell

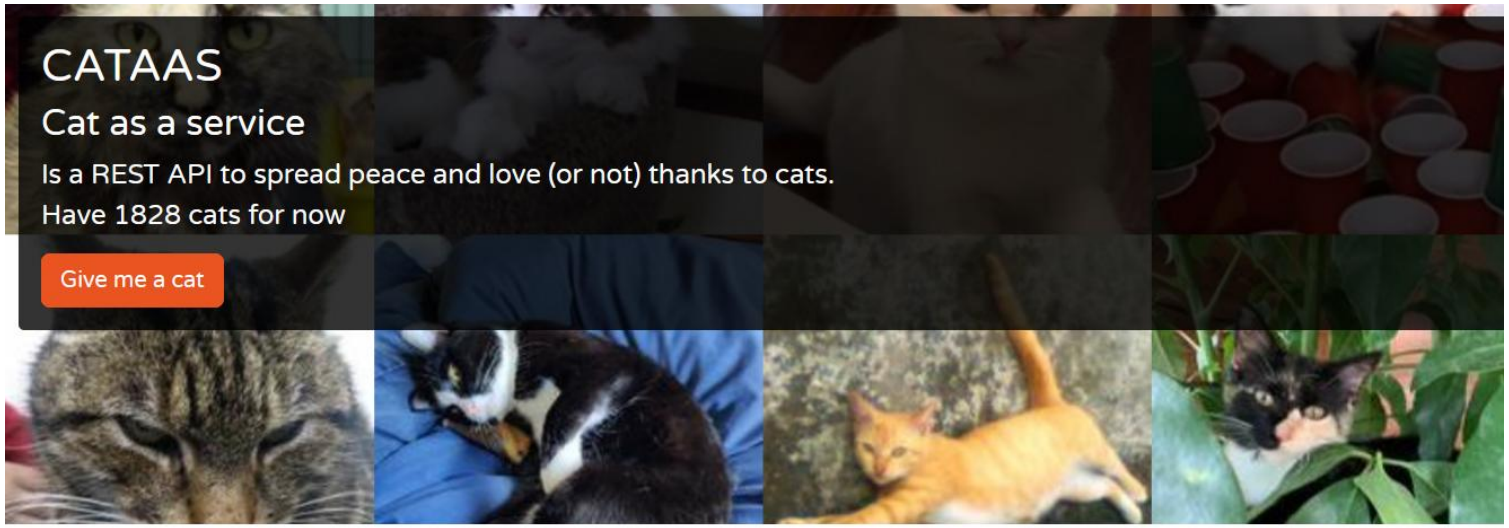
- **RE**presentational **S**tate **T**ransfer.
- **Resource-oriented:** everything is a *resource* with a URI.
- Interact via standard HTTP methods.
- Representations (usually JSON) transfer state between client and server.

**Mental model:** *resources (nouns) + methods (verbs) + representations (JSON).*




**Endpoint**: URL plus the HTTP method

# RESTful API example - CATaaS




Hey ! Do you like Cataas? Do you want to support the project ? Buy me a beer

Hey cat lovers, new major version of cataas :

- Now JSON will returns if request contain `application/json` header, API doc updated here
- Fix tags search and fix tags combined with "," separator (see documentation)

## Basic

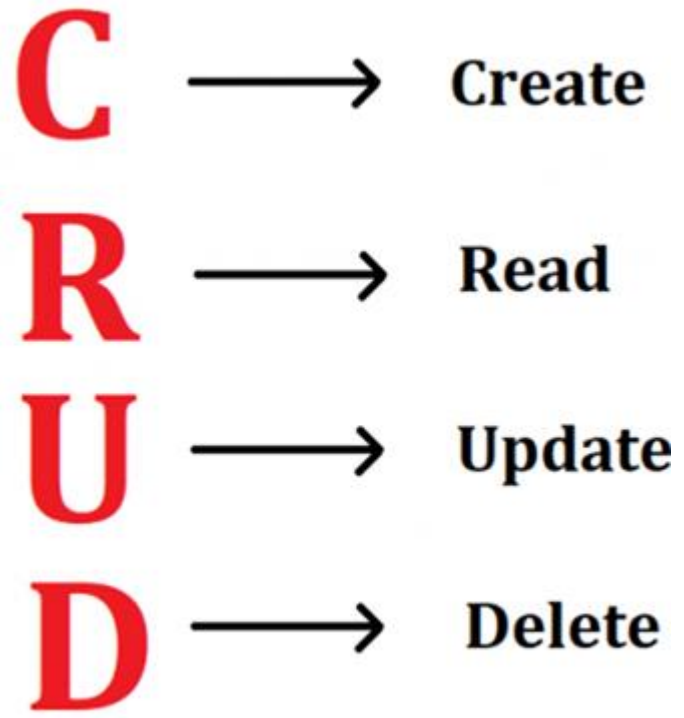
| Url | Description | Example |
|---|---|---|
| **/cat** | Will return a random cat | Random cat |
| **/cat/:tag** | Will return a random cat with a `:tag`, You can combine multiple tags with `:tag` separator | Random orange cute cat |
| **/cat/gif** | Will return a random gif cat \o/ | Random gif cat |
| **/cat/says/:text** | Will return a random cat saying `:text` | Random cat saying hello |
| **/cat/:tag/says/:text** | Will return a random cat with a `:tag` and saying `:text` | Random cute cat saying hello |
| **/cat/says/:text?fontSize=:size&fontColor=:color** | Will return a random cat saying `:text` with text's `:fontSize` and text's `:fontColor` | Custom random cat saying hello |

https://cataas.com/

# HTTP Methods (Semantics)

- **GET**: read (safe, idempotent)
- **POST**: create, non-idempotent
- **PUT**: replace (idempotent)
- **PATCH**: partial update (not guaranteed idempotent, aim to be)
- **DELETE**: delete (idempotent by convention)

Design endpoints to respect method semantics.



# Path design

- Use **nouns, plural, lowercase**, **hyphen-separated** segments. No trailing slash.
  - ✅ `/courses, /courses/{courseId}/assignments`
  - ❌ `/createCourse, /Courses, /courses/`
- **Path = identity.** Put *what it is* in the path; put *how you want it* in query params.
  - ✅ `GET /orders?status=shipped&sort=-created_at&limit=25`
  - ❌ `GET /orders/status/shipped/sort/created_at/desc/25`
- Prefer **opaque IDs** (like UUIDs) to Databases' IDs.
  - ✅ `/articles/8f9a2`
- **Depth limit:** avoid nesting deeper than 2.
  - ✅ `/courses/{courseId}/assignments/{assignmentId}`
  - ➕ Also provide top-level when useful: `/assignments/{assignmentId}`

**Canonicalization:** choose a single style (lowercase, no trailing slash) and redirect others to it (with HTTP 301).

# Status Code usage

- **2xx** success: 200 OK, 201 Created, 204 No Content.
- **3xx** redirects: 304 Not Modified.
- **4xx** client errors: 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 409 Conflict, 422 Unprocessable Entity.
- **5xx** server errors: 500, 502, 503.

**Consistent mapping** helps clients react properly.

For errors prefer RFC 7807 Problem Details

# Authentication & Authorization

- **API keys:** simple, limited
- **Bearer JWTs:** stateless, include claims/exp
- **OAuth 2.0 / OIDC:** delegated auth (users sign in with provider)
- Always use **TLS (HTTPS)**
- **Principle of least privilege** & short-lived tokens.

We will go deep in this part later in the course!

# Recapping General Practices

## Good

Nouns for resources, consistent naming

- ✅ Proper status codes & error format
- ✅ Validation & helpful error messages
- ✅ TLS everywhere; no secrets in URLs
- ✅ Rate limiting & monitoring
- ✅ Observability: request IDs, logs, metrics

## Bad

- ❌ Verbs in paths (/createBook)
- ❌ Overloaded POST for everything
- ❌ 200 OK for errors
- ❌ Leaking internals (stack traces, SQL)
- ❌ Breaking changes without versioning

Once you've developed a RESTful service, how do clients know how to interact with it?

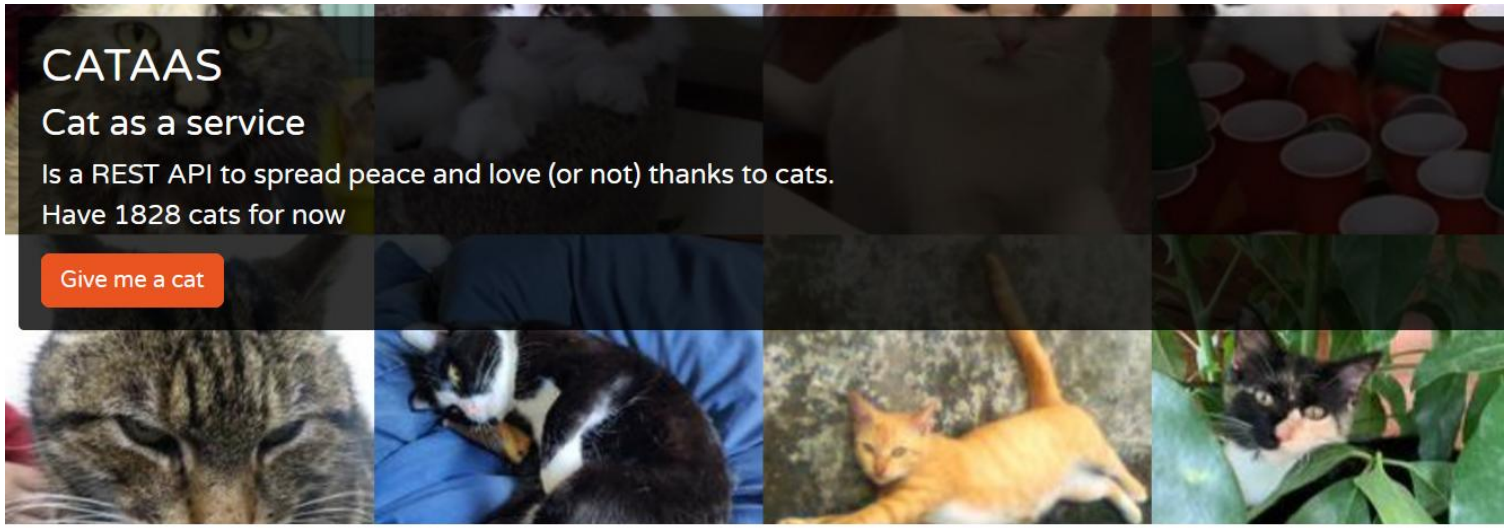
# OpenAPI


Specification for a machine-readable interface oriented to RESTful API.

- Useful tools: Swagger Editor (online), Swagger API (app), Swagger Hub (needs registration)

- It uses either YAML or JSON.

# OpenAPI of CATAAS




Hey ! Do you like Cataas? Do you want to support the project ? Buy me a beer

Hey cat lovers, new major version of

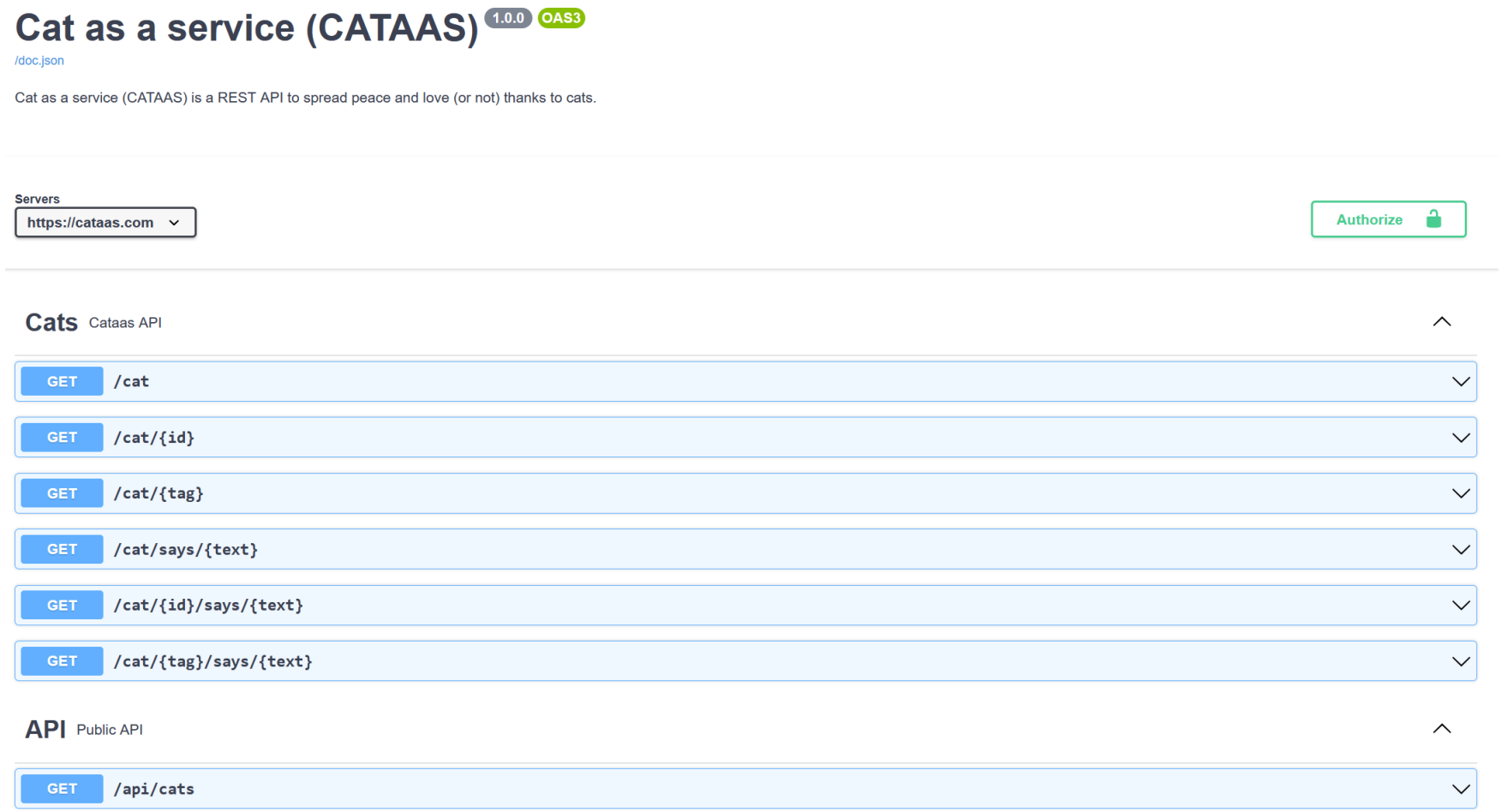
- Now JSON will returns if request contain `application/json` header, API doc updated here
- Fix tags search and fix tags combined with "," separator (see documentation)

## Basic

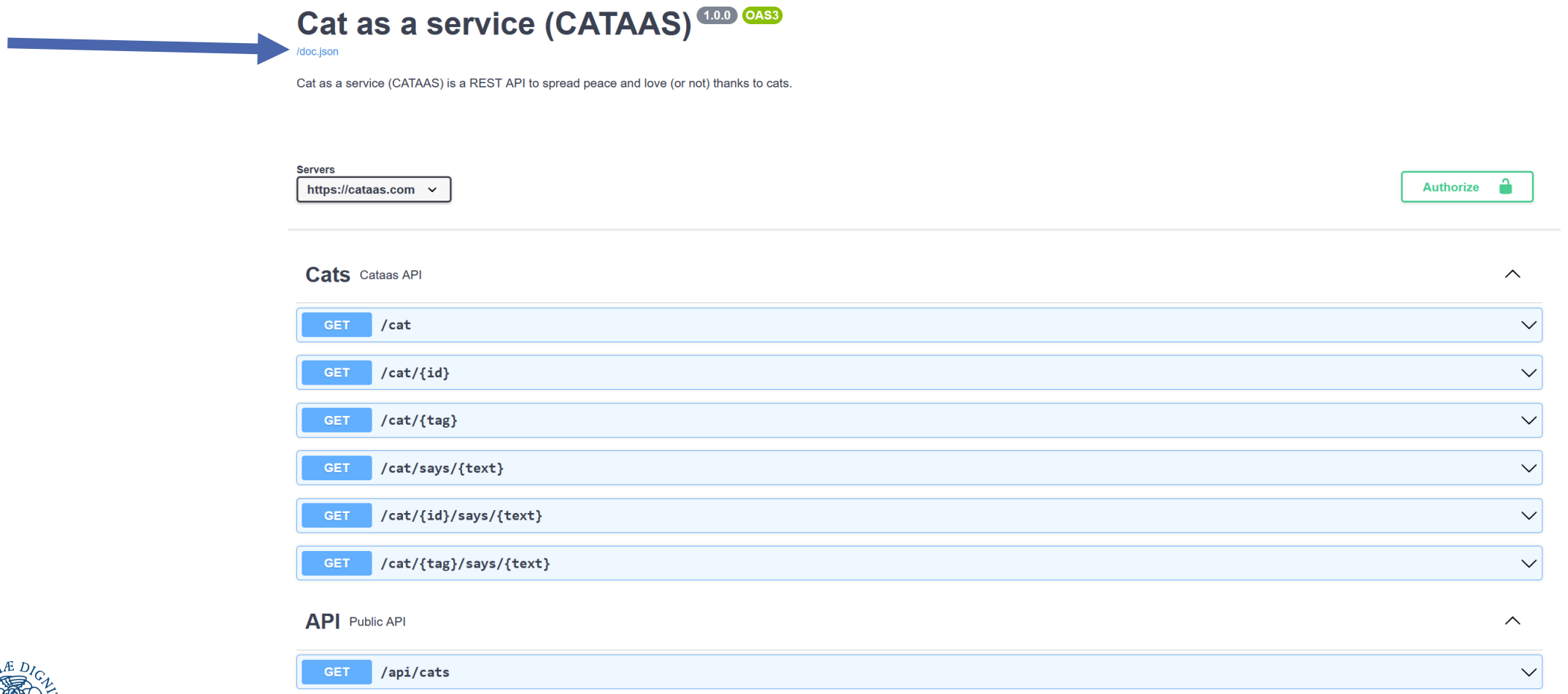
| Url | Description | Example |
|---|---|---|
| **/cat** | Will return a random cat | Random cat |
| **/cat/:tag** | Will return a random cat with a `:tag`, You can combine multiple tags with `:tag` separator | Random orange cute cat |
| **/cat/gif** | Will return a random gif cat \o/ | Random gif cat |
| **/cat/says/:text** | Will return a random cat saying `:text` | Random cat saying hello |
| **/cat/:tag/says/:text** | Will return a random cat with a `:tag` and saying `:text` | Random cute cat saying hello |
| **/cat/says/:text?fontSize=:size&fontColor=:color** | Will return a random cat saying `:text` with text's `:fontSize` and text's `:fontColor` | Custom random cat saying hello |

https://cataas.com/

# CATAAS REST API (Swagger view)



# CATAAS REST API (Swagger view)



# CATAAS REST API (Source file)


```
openapi: "3.0.3"
info:
  version: "1.0.0"
  title: "Cat as a service (CATAAS)"
  description: "Cat as a service (CATAAS) is a REST API to spread peace and love (or not) thanks to cats."
servers:
  0:
    url: "https://cataas.com"
tags:
  0:
    name: "Cats"
    description: "Cataas API"
  1:
    name: "API"
    description: "Public API"
  2:
    name: "Security"
    description: "Security"
  3:
    name: "Admin"
    description: "Admin API"
components:
  securitySchemes:
    jwt:
      type: "http"
      scheme: "bearer"
      bearerFormat: "JWT"
  schemas:
    AdminCat:
      required:
        0: "file"
        1: "tags"
        2: "validated"
      type: "object"
      properties:
        _id:
          type: "string"
        validated:
```

# OpenAPI - Specification


Mandatory parts:

- Version and info
- At least one of:
  - Paths -> endpoints
  - Webhooks -> callbacks
  - Components -> reusable parts of the OpenAPI file

Optional parts:

- Servers
- Security
- Tags
- External Docs

https://swagger.io/specification/

# OpenAPI – Version and info

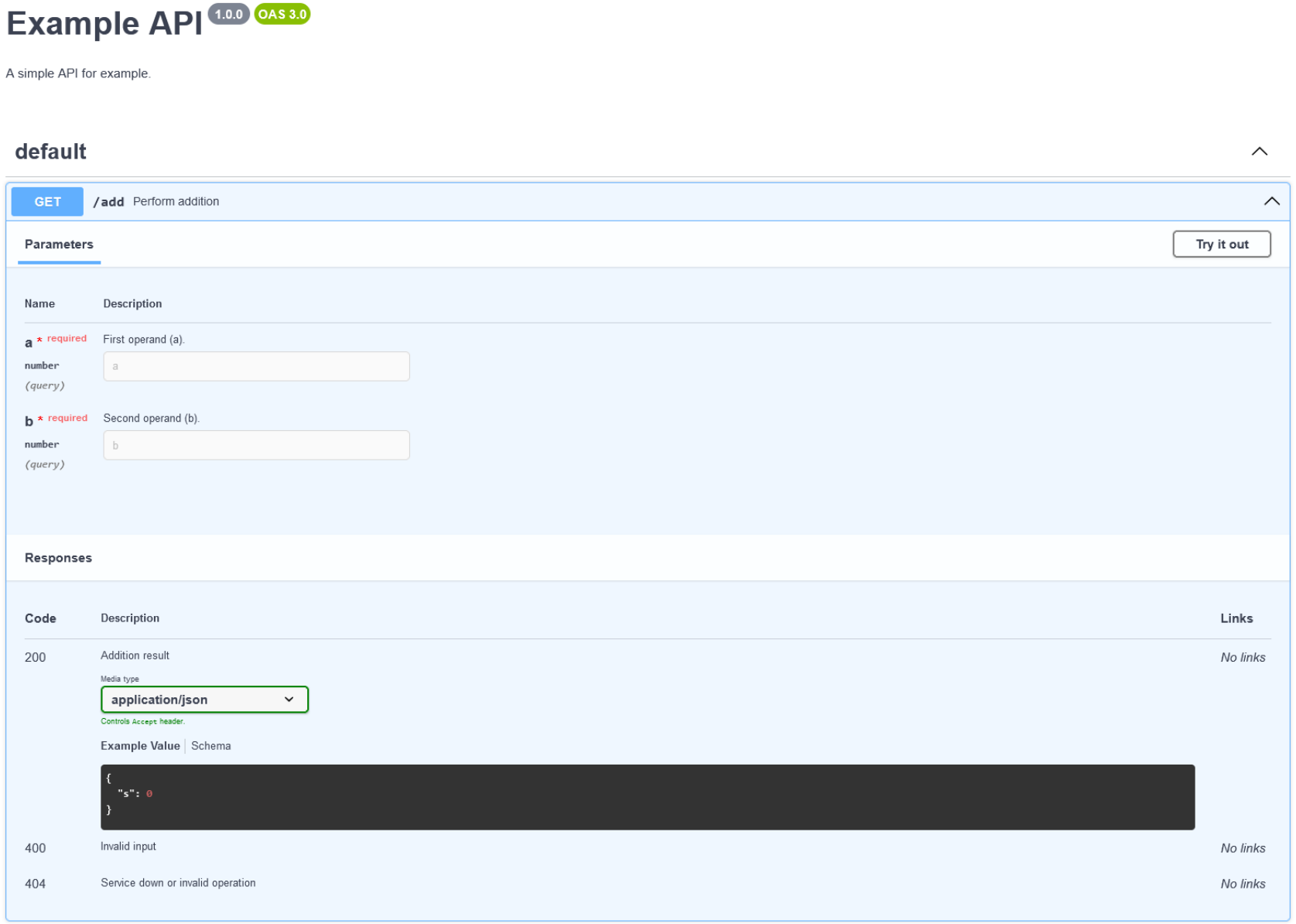
```yaml
openapi: 3.0.0  # The OpenAPI version you're using
info:
  title: Example API  # The name of your API
  description: A simple API for example.  # A brief description
  version: 1.0.0  # Version of the API
```

# OpenAPI – paths

/add?a=1&b=1

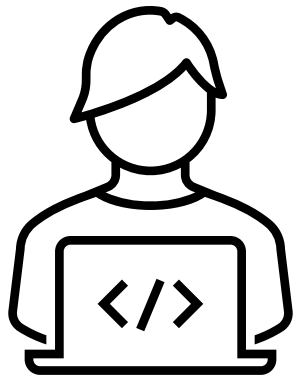
```yaml
paths:
  /add:
    get:
      summary: Perform addition
      parameters:
        - name: a
          in: query
          required: true
          description: First operand (a).
          schema:
            type: number
        - name: b
          in: query
          required: true
          description: Second operand (b).
          schema:
            type: number
      responses:
        '200':
          description: Addition result
          content:
            application/json:
              schema:
                type: object
                properties:
                  s:
                    type: number
        '400':
          description: Invalid input
        '404':
          description: Service down or invalid operation
```

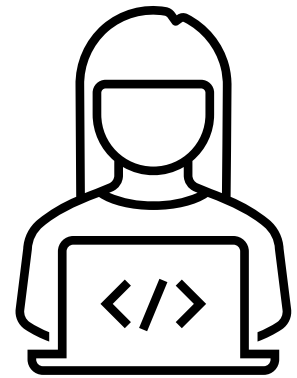
# OpenAPI – Swagger Editor https://editor.swagger.io/



# OpenAPI - Writing

Do I have to do it by hand?





# OpenAPI - Writing

Not necessarily!

There are tools that from your code generate (part of) the OpenAPI spec and vice versa.

For example:

- OAS tools (npm based)
- Flask-RESTX (annotations of Flask code)
- Generative AI

# Now it’s your turn

1. Use the service coded in Lab 1.
2. Make it RESTful following the best practices and the CATAAS example.
3. Write the OpenAPI specification for it.
4. Use Swagger Editor to see its web representation.




# Endpoints recap

- `/add` which given two number return a JSON with their sum.
- `/sub` which given two number return a JSON with their subtraction.
- `/mul` which given two number return a JSON with their multiplication.
- `/div` which given two number return a JSON with their division.
- `/upper` which given a string returns it in a JSON, all in uppercase.
- `/lower` which given a string returns it in a JSON, all in lowercase.
- `/concat` which given two strings returns in a JSON the concatenation of them.
- `/reduce` which takes the operator (one of add, sub, mul, div, concat) and a list string representing a list, apply the operator to all the elements giving the result. For instance, with `add` and `[2,1,3,4]` it returns a JSON containing 10, meaning 2+1+3+4.
- `/crash` which terminates the service execution after responding to the client with info about the host and the port of the service.
- `/last` which returns a string representing the last operation requested with success in a fixed format.

# Lab take away

- ❑ Learn about REST and OpenAPI.
- ❑ Coding a RESTful API.
- ❑ Writing their OpenAPI specification.

# Project take away

- The project consist in writing a backend with RESTful APIs.
- The documentation of the RESTful API of the project must be expressed in OpenAPI format.