ROBERTO POLLI TEAM PER LA TRASFORMAZIONE DIGITALE

# USING OPENAPI (AND PYTHON) TO STANDARDIZE A COUNTRY'S API ECOSYSTEM

PyCon X
Firenze, May 5th 2019

## Agenda

## Team Mission

Make **public services for citizens and businesses accessible** in an easy manner,

via a mobile first approach,

with **reliable**, scalable and fault tolerant **architectures**,

based on clearly defined **APIs**.

## Who am I

Roberto Polli - love writing in Python, C and Java

RHC{E,VA}, MySQL|MongoDB Certified DBA

API Ecosystem @ TeamDigitale

# From Enterprise to The Web

## REST in the New Italian Interoperability Framework

- Enable the creation of new services for citizens, lowering setup and maintenance/operation costs

- Simplify communication with non-governmental agencies

- Acknowledge that public services are usually about data and resources

- Keep current with the  IT world ;)

- REST without Richardson Maturity Model constraints

## The Quest: an Italian API Ecosystem

- Standardize HTTP APIs for 20k agencies and 60M people
- API-first approach to REST APIs
- Scheme standardization based on national, European and industry standards
- Availability strategy based on a distributed circuit-breaker and throttling patterns
- National API Catalogue

# API

Presentiamo qui una selezione di API della Pubblica Amministrazione su cui Developers Italia è al lavoro, in vista della creazione del catalogo delle API previsto dal Piano Triennale.

AgID Agenzia per l'Italia Digitale

## Repertorio Nazionale dei Dati Territoriali - geodati.gov.it

AgID

Interfaccia REST verso il repertorio nazionale dei dati territoriali.

per saperne di più >

·D TEAM PER LA TRASFORMAZIONE DIGITALE

## Data & Analytics Framework CKAN API

Team Digitale

API per ricercare e visualizzare gli open data del Data & Analytics Framework in api.daf.teamdigitale.it

per saperne di più >

BANCA D'ITALIA EUROSISTEMA

## SIOPE+

Banca d'Italia

Regole tecniche per il colloquio telematico di Amministrazioni pubbliche e Tesorieri con SIOPE+.

per saperne di più >

Regione Lombardia E015 digital ecosystem

## Muoversi in Lombardia

Ministero delle Infrastrutture e Trasporti

## Opendata Trasporti

in arrivo

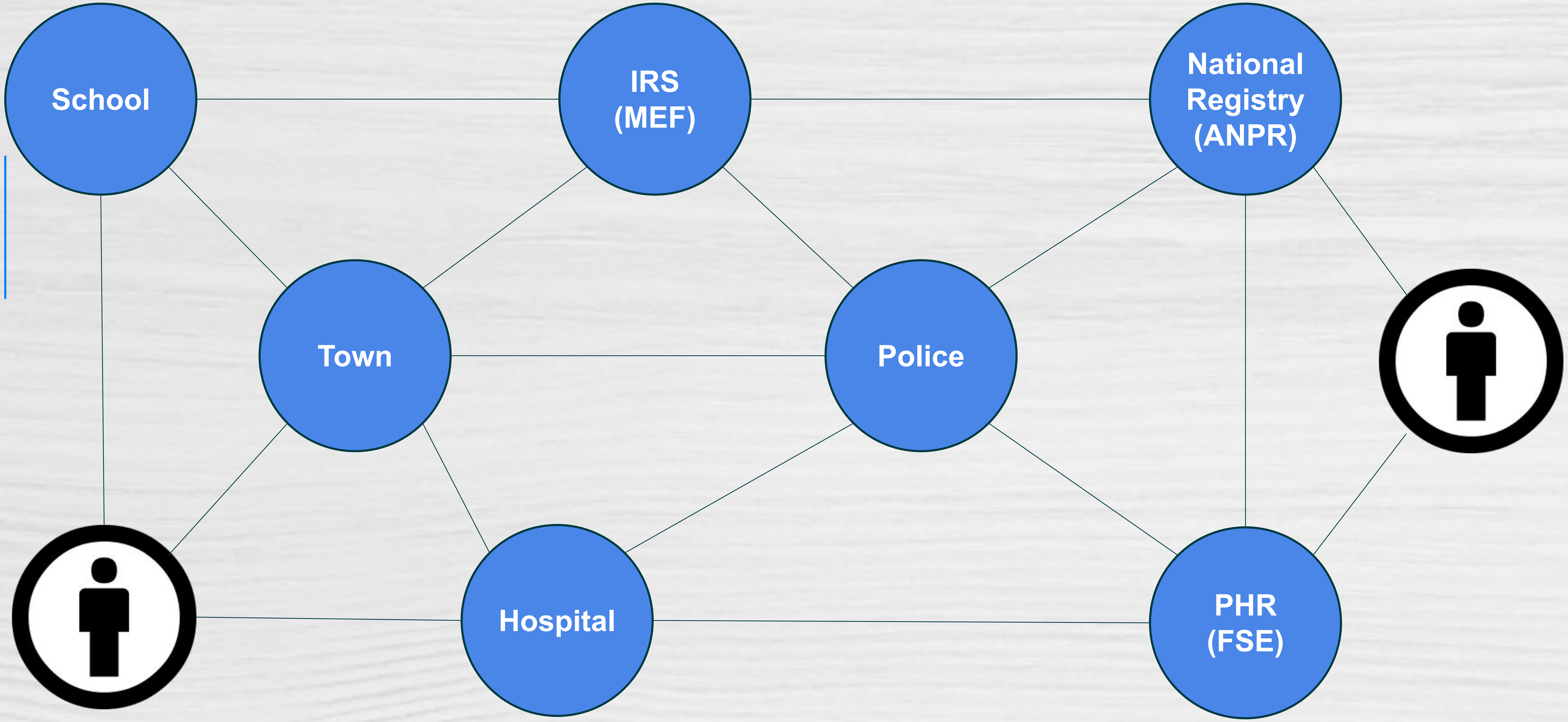
Regione Lombardia E015 digital ecosystem

## Aree Protette Lombardia

## Cooperating in Italy, Europe and abroad!

- Discussing rules with Italian and European agencies
- Ongoing work with some Regions around API metadata / catalogs
- Contributing to the European Commission ongoing work on APIs
- Represent public administration use cases in IETF and W3C forums
- Support the implementation of our guidelines with opensource communities and software vendors

# The Future Ecosystem



## Ongoing work on Standardization and Reliability

## HTTPS

http

binary messages

BANNED

**Always HTTPS**

**Wrap queues (kafka, JMS, AMQP, …) with HTTPS for authentication and authorization**

**Leverage STATUS, METHOD and PATH for auditing and routing**

## Ontology based schemas

BANNED

cod_fiscale
piva fiscalCode CF nato
codice_fisc nome
partIva cfiscale nato_a
cf p_IVA fiscal_code PI
name

**tax_code**
**vat_number**
**given_name**

(from w3id.org/italia)

## Logs, dates: RFC5424 / 3339

```
          ago   6 14:04:50
        ago-06 18:58:50,000
        Aug 02 18:43:47.000
   mer   9 ago 08:45:37 CEST 2018
      Fri May 05 08:45:37 IST
      2018-May-08 10:06:25 AM

        05/12/2018 2018/12/05
        12-05-2018 05/12/2018
        2018-12-05 12-05-2018
```

BANNED

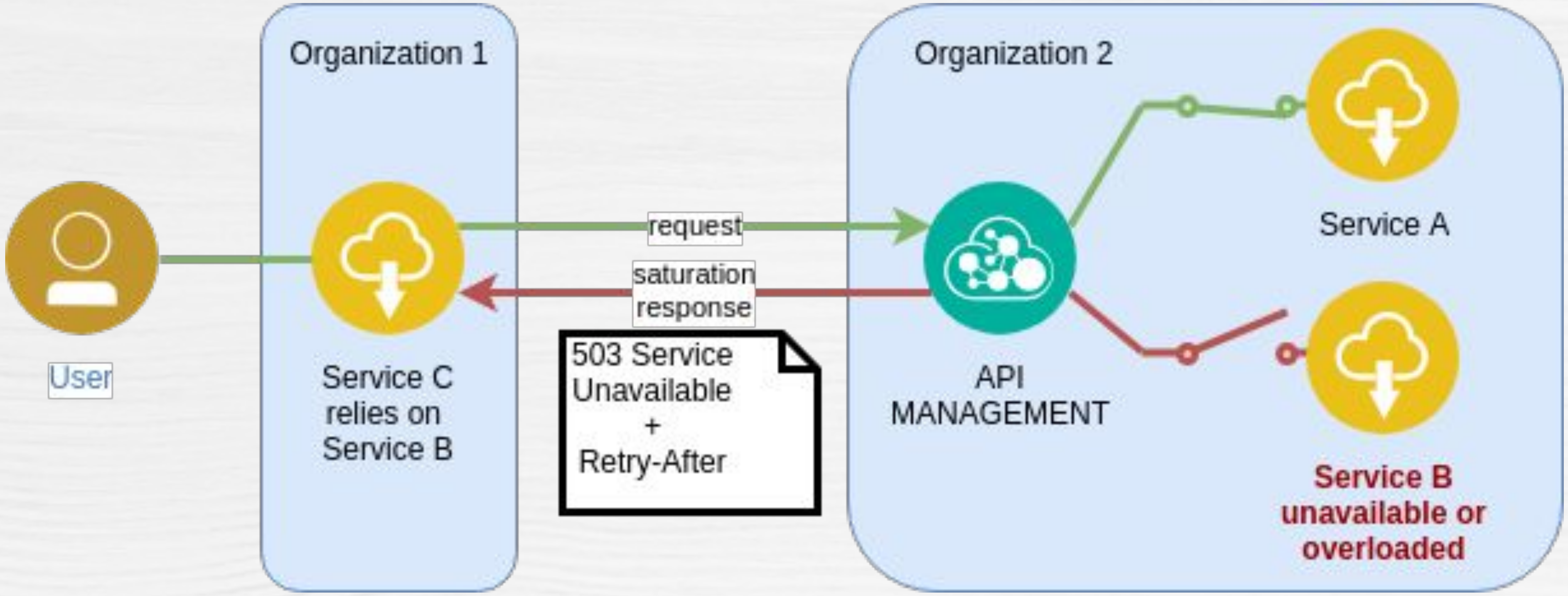
**`2018-05-08T10:06:25Z`**

**`2018-05-08T10:06:25.000Z`**

**`(Unix timestamp is allowed too)`**

# Service Management

Service management techniques (eg. circuit-breaker)



User
Organization 1
Service C relies on Service B
request
saturation response
503 Service Unavailable + Retry-After
Organization 2
API MANAGEMENT
Service A
Service B unavailable or overloaded

## Service Management Headers

x-rate-limit-minute: 100 X-RateLimit-Retry-After: 11529485261

X-RateLimit-UserLimit: 1231513

X-RateLimit-UserRemaining X-Rate-Limit-Limit: name=rate-limit-1,1000

x-custom-retry-after-ms

X-Rate-Limit-Remaining-month

X-Rate-Limit-Reset: Wed, 21 Oct 2015 07:28:00 GMT

x-rate-limit-hour: 1000

### Communicate service limits

X-RateLimit-Limit: #request
X-RateLimit-Remaining: #request
X-RateLimit-Reset: #seconds

### Communicate service status

HTTP 503 (service unavailable)
HTTP 429 (too many requests)
Retry-After: #seconds

Working with opensource API gateways for compliance!

## Errors: RFC7807

{ "message": "Service Unavailable",
"code":￼ 123 }￼ **{ "status" : "error", "message":
"Unable to communicate with database" }** {
"error": { "errors": [ { "reason": "required",
"message": "Login Required", "locationType":
"header", "location": "Authorization" } ], "code":
401, "message": "Login Required" } }" } **{"error": {
"code": "501", "message": "Unsupported
functionality", "target": "query",
"details": "" }**

BANNED

RFC 7807 is an extensible format for error messages

```
{
"type": "https://api.example.it/errors/off-hours",
"title": "Service Unavailable",
"detail": "Service is active in forex hours",
"status": 503,
 "instance": "/account/12345/msgs/abc",
}
```

## Standardized metrics

Set common and simple indicators:

- availability: eg. the service was up for 95% of the time
- success_rate: % of successful requests
- target_response_time: expected latency at 95p

Evaluating APDEX index for its simplicity:

$$Apdex_t = \frac{SatisfiedCount + \frac{ToleratingCount}{2}}{TotalSamples}$$

# OpenAPI v3

## Describing APIs

API-First:
- publish interfaces
- involve stakeholders in API lifecycle

Communicate:
- technical specifications
- metadata
- docs & references

## OpenAPI 3.0 aka OAS3

Initiative under the Linux Foundation, participated by gov & co (gov.uk, Microsoft, Google, Oracle, IBM, ..)

Driver for API adoption

WSDL for REST APIs

Evolution of Swagger 2.0

## OpenAPI aka OAS3

- Lightweight format: YAML
- Generates docs & code via tools (swagger-editor, apicur.io)
- Reusable components via hyperlink (eg. $ref)
- A set of curated objects available on github: interoperability by reuse!

# OpenAPI Examples

## OpenAPI 3 supports metadata and markdown

```
openapi: 3.0.2
info:
  version: "2.1.4"
  title: Tax Code
  description: |
    ## You can markdown!
      Write here the full API docs.
  termsOfService: 'https://tos.example.it'
  contact:
    email: roberto@teamdigitale.governo.it
    name: Team Digitale
    url: 'https://teamdigitale.governo.it'
```

```
# And custom tags for catalog purposes
x-summary:  Get citizen data from tax code.

x-lifecycle:
  published: 2018-01-01
  deprecated: 2020-12-31
  retired: 2021-03-31
  maturity: published

x-healthCheck:
  url: https://example.it/v1/find?id=E472&limit=1
  interval: 300    # seconds
  timeout: 15      # seconds
```

## OpenAPI 3 - can define schemas

```
components:
 schemas:
  Citizen:
   properties:
    given_name:
     type: string
     required: true
     example: Leon Battista
    family_name:
     type: string
     example: Alberti
    tax_code:
     type: string
     pattern: /^[A-Z0-9]{16}/
     example: LBRLBT72D25D969F
```

Define an entry schema and provide examples used by tooling.

```
{
 "given_name": "Leon Battista",
 "family_name": "Alberti",
 "tax_code": "LBRLBT72D25D969F"
}
```

# OpenAPI 3 - and define composing objects

```
components:
 schemas:
  Citizens:
   entries:
    type: array
    items:
     $ref": >-
      "https://definitions.yml#/Citizen"

 responses:
  Citizens:
   description: Return a list of citizens
   content:
    application/json:
     schema:
      $ref: '#/components/schemas/Citizens'
```

Compose local and remote schemas
to enable interoperability by reuse

```
citizens = {
  "entries": [
   {
      "given_name": "Leon Battista",
      "family_name": "Alberti",
      "tax_code": "LBRLBT72D25D969F"
    }, {
      "given_name": "Mario",
      "family_name": "Rossi",
      "tax_code": "MRORSS77T05E472I"
    }
  }
```

## OpenAPI 3 - operations enable code generation

```
components:
 parameters:
  tax_code:
   in: path
   required: true
   schema:
    $ref: 'https://definitions.yml#/TaxCode'
paths:
 /citizens/{tax_code}:
  get:
   summary: Get a citizen by tax_code
   operationId: get_citizen
   parameters:
   - $ref: "#/components/parameters/tax_code"
   responses:
    "200":
     $ref: "#/components/responses/Citizen"
     ...
```

Associate operations to (path, method).

Code generators use operationId
to reference the implementing function.

```
# api.py
def get_citizen(tax_code: str):
    """Returns a Citizen."""
    citizen = db.get(tax_code)
    return Citizen(**citizen)
```

# OpenAPI 3 - yaml anchors are syntactic sugar

```
x-anchors:
  throttling_headers: &throttling_headers
    X-RateLimit-Limit:
      $ref: 'https://cdn.yml/headers#/X-RateLimit-Limit
    X-RateLimit-Remaining:
      $ref:
     'https://cdn.yml/headers#/X-RateLimit-Remaining
    X-RateLimit-Reset:
      $ref: 'https://cdn.yml/headers#/X-RateLimit-Reset

responses:
  Citizens:
    description: Return a list of citizens
    headers:
      >>: *throttling_headers
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Citizens'
```

Add a common set of headers to every operation

```
def get_citizen(tax_code: str):
    """Returns a Citizen."""
    ... use the framework request context  ...
    throttling = get_quota_headers(context.user)
    ...
    citizen = db.get(tax_code)
    return Citizen(citizen), 200, throttling
```

## Our sponsored features in OAS 3.1

Achieved:

- ✔ mutualTLS support PR #1764
- ✔ catalog field (info.summary) PR#1779

Ongoing work on:

- custom securitySchemes PR #1812
- external schemas support (eg. xmlschema) PR #1736

## Connexion

## Connexion: write specs, then code!

An OAS3 framework based on Flask

Ships:
- problem+json predefined responses
- basic and jwt authentication

Great for sketching APIs and test the interoperability rules!

# Connexion 101: minimal example

```python
from connexion import FlaskApp, problem
from connexion.request import headers

def get_status():
    """Connexion processes the yaml, and
        executes `get_status`
    """
    user_agent = headers.get('User-Agent', 'Nemo')

    return problem(status=200, title="Ok",
                   detail=f"Hi {user_agent}")

if __name__ == "__main__":
    app = FlaskApp(
      'hello', port=443,
      specification_dir=".",
      options={"swagger_ui": True}
    )
    app.add_api("simple.yaml", validate_responses=True)
    app.run(ssl_context="adhoc")
```

```yaml
openapi: 3.0.1
info:   ... metadata ...
servers:
 -  url: 'https://localhost/hello/v1'
paths:
  /status:
    get:
      summary: Check API availability
      operationId: api.get_status
      responses:
       '200':
         description:  Hi!
         content:
           application/json:
             schema:
               $ref: '#/components/schemas/Problem'
```

## Connexion: basic_auth & jwt security

```
def my_auth(username, password,required_scopes=None):
    """An dummy authentication function."""

    if username == password:
        return {"sub": username, "scope": ""}

    # Not authenticated
    return None
```

```
>GET /hello/v1/basic-auth
{
 "detail": "No authorization token provided",
 "status": 401,
 "title": "Unauthorized",
 "type": "about:blank"
}

> GET /hello/v1/basic-auth
> Authorization: Basic foo:foo
{  "hello": "world" }
```

```
components:
 securitySchemes:
  myBasicAuth:
   type: http
   scheme: basic
   x-basicInfoFunc: api.my_auth
paths:
 /basic-auth:
  get:
   security:               # Just reference the
   - myBasicAuth: []   # previous securityScheme
   ...
   responses:
    '200':
      ...
```

# Connexion: validating requests

```
def post_hello(body):
  # Basic request/response validation is tolerant
  # so you should check corner cases
  if not isinstance(body, dict):
     return problem(status=400, title="Bad Request",
         detail="Body should be a json object")

  return {"text": body["text"]}
```

```
> POST/hello/v1/echo
> {"foo": "bar"}
{
 "detail": "'text' is a required property",
 "status": 400,
  ...
}
```

```
...
paths:
 /echo:
  post:  # this forwards post requests to
   operationId: app.post_hello
   summary: Requires a json body
   requestBody:
    required: true
    content:
     application/json:
      schema:
       $ref: '#/components/schemas/Text'
   responses:
    '200':
     description:  Hi!
     ...
```

# Connexion: validating responses

```
def post_hello(body):
  ...
  # In OAS3 declared a { "text": "string" } response
  # We instead return an
  return {"UNEXPECTED_ITEM": "1"}
```

```
> POST /hello/v1/echo
>
> {"foo": "bar"}
{
    "detail": "'text' is a required property...Failed validating...",
    "status": 500,
    "title": "Response body does not conform to specification",
    "type": "about:blank"
}
```

```
...
paths:
 /echo:
  post:  # this forwards post requests to
   operationId: app.post_hello
   summary: Validate json responses
   requestBody: ...
   responses:
    '200':
     description:  Hi!
     content:
      application/json:
       schema:
        $ref: '#/components/schemas/Text'
```

## Connexion: customizing validators

Connexion validators are quite tolerant, but you can write your own extending a validator class (eg. for request body, parameters, responses, ...)

```
from connexion.decorator.validation import (
   RequestBodyValidator,
   ResponseBodyValidator,
   ParameterValidator)

def CustomBodyValidator(RequestBodyValidator):
       ...

# Then in your __main__
   app = FlaskApp(..)
   app.add_api("simple.yaml",
        validate_responses=True,
        validator_map={
            'body': CustomBodyValidator,
         }
    )
```

```
...
paths:
 /echo:
   post:  # this forwards post requests to
     operationId: app.post_hello
     summary: Validate json responses
     requestBody: ...
     responses:
       '200':
         description:  Hi!
         content:
           application/json:
             schema:
               $ref: '#/components/schemas/Text'
```

# Connexion: returning problems with RFC7808

Connexion automagically returns application/problem+json in case of errors.

This helps standardizing API error handling.

It's always developer responsibility NOT TO expose stack traces or personal data / reserved informations though!

So test your applications!

```
{
  "type": "https://example.org/out-of-stock",
  "title": "Out of Stock",
  "status": 400,
  "detail": "Item B00027Y5QG is no longer
    available",
  "product": "B00027Y5QG"
}
```

## Connexion: logging at Zulu (UTC)

The interoperability model mandates logging in UTC to save timezone/DST management and ensure monotonical logs.

Use `rfc5424-logging-handler`

```
# pip install rfc5424-logging-handler
from logging.config import dictConfig
from yaml import safe_load as yaml_load

if __name__ == '__main__':
  # Configure the logger.
  with open('logging.yaml') as fh:
    log_config = yaml_load(fh)
    dictConfig(log_config)
```

```
# logging.yaml configuration.
version: 1
formatters:
  # Avoid Flask sending the timestamp twice in the message.
  fmt_syslog:
    format: '%(levelname)s in %(module)s: %(message)s'
handlers:
  # This handler converts log in UTC before sending to syslog
  rfc5424:
    class: rfc5424logging.Rfc5424SysLogHandler
    level: DEBUG
    utc_timestamp: True
    formatter: fmt_syslog

# Use the rfc5424 handler by default.
root:
    level: DEBUG
    handlers: [ rfc5424 ]
```

# References

## New (ongoing) Italian Framework

https://docs.italia.it/italia/piano-triennale-ict/lg-modellointeroperabilita-docs/

https://forum.italia.it/c/piano-triennale/interoperabilita

https://forum.italia.it/t/modi2018-il-modello-di-interoperabilita-2018/3762

Roberto Polli
roberto@teamdigitale.governo.it

@ioggstream
@teamdigitaleIT

@team-per-la-trasformazione-digitale

teamdigitale.governo.it