STScI | SPACE TELESCOPE SCIENCE INSTITUTE

EXPANDING THE FRONTIERS OF SPACE ASTRONOMY

OpenAPI & Code Generation

Joshua Fraustro

June - 2025

What is OpenAPI?

- Standard for describing HTTP RESTful API's
- Machine and human-readable
- Described in JSON or YAML
- Defines paths, operations, parameters, payloads and responses.
- Clear, concise, technical
- Large tooling ecosystem

Adoption in the IVOA

- Tested and pushed for by the recent Protocol Transition Tiger Team (P3T)
- TAP-next, SCS, DALI are getting OpenAPI descriptions
- Looking at how to include in the Registry

```yaml
openapi: 3.0.4
info:
  title: Swagger Petstore - OpenAPI 3.0
  version: 1.0.12
tags:
  - name: pet
    description: Everything about your Pets
  - name: store
    description: Access to Petstore orders
  - name: user
    description: Operations about user
paths:
  /pet:
    put:
      tags:
        - pet
      summary: Update an existing pet.
      description: Update an existing pet by Id.
      operationId: updatePet
      requestBody:
        description: Update an existent pet in the store
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Pet'
          application/xml:
            schema:
              $ref: '#/components/schemas/Pet'
          application/x-www-form-urlencoded:
            schema:
              $ref: '#/components/schemas/Pet'
        required: true
      responses:
        '200':
          description: Successful operation
```

Code Generation & Tooling

Because OpenAPI specifications are machine-readable, it is possible to generate client and server code from the specification itself.

- Usually done through a templating engine
- Available tooling varies by language
- Attempted in the past by P3T (to varying levels of success)
    - Issues with polymorphism (oneOf, allOf, anyOf etc.)

Lots of open-source tools around this idea, with varying levels of support, completeness, as well as commercial tools.

Code Generation

OpenAPI Specification

```yaml
paths:
  /pet:
  /pet/findByStatus:
    get:
      tags:
        - pet
      summary: Finds Pets by status.
      description: Multiple status values
        separated strings.
      operationId: findPetsByStatus
      parameters:
        - name: status
          in: query
          description: Status values that
            filter
          required: false
          explode: true
          schema:
            type: string
            default: available
            enum:
              - available
              - pending
              - sold
```

Jinja2 Template

```jinja
from enum import StrEnum
from typing import Optional
from fastapi import Response, APIRouter, Query

{{ router.name }} = APIRouter()

class {{ enum.name }}(StrEnum):
    """{{ enum.description}}"""
    {% for value in enum.values %}
    {{ value.name.upper }} = "{{ value.value }}"
    {% endfor %}

@{{router.name }}.{{ path.method }}("{{ path.path }}"
def {{ path.operationId }}(
    {% for param in path.parameters %}
    {{ param.name }}: Optional[{{ param.type }}] = Qu
        default={{ param.default }},
        description="{{ param.description }}"
    )
    {% if not loop.last %},
    {% endif %}
    {% endfor %}
):
    """{{ path.summary }}
    {{ path.description }}"""

    ...
    return Response(
        content="{{ response.content }}",
        status_code={{ response.status_code }},
    )
```

Python Code (FastAPI)

```python
from enum import StrEnum
from typing import Optional
from fastapi import Response, APIRouter

pet_router = APIRouter()

class PetStatus(StrEnum):
    """Enum for Pet Status"""

    AVAILABLE = "available"
    PENDING = "pending"
    SOLD = "sold"

@pet_router.get("/pet/findByStatus")
def find_pets_by_status_route(
    status: Optional[list[PetStatus]] = None,
):
    """Finds Pets by status
    Multiple status values can be provided with comma separated strings.
    """

    pets = find_pets_by_status(status)

    return Response(
        content=pets,
        status_code=200,
        media_type="application/json"
    )
```

This Talk

- What does the current tooling around Python server-generation look like?

Why Python?

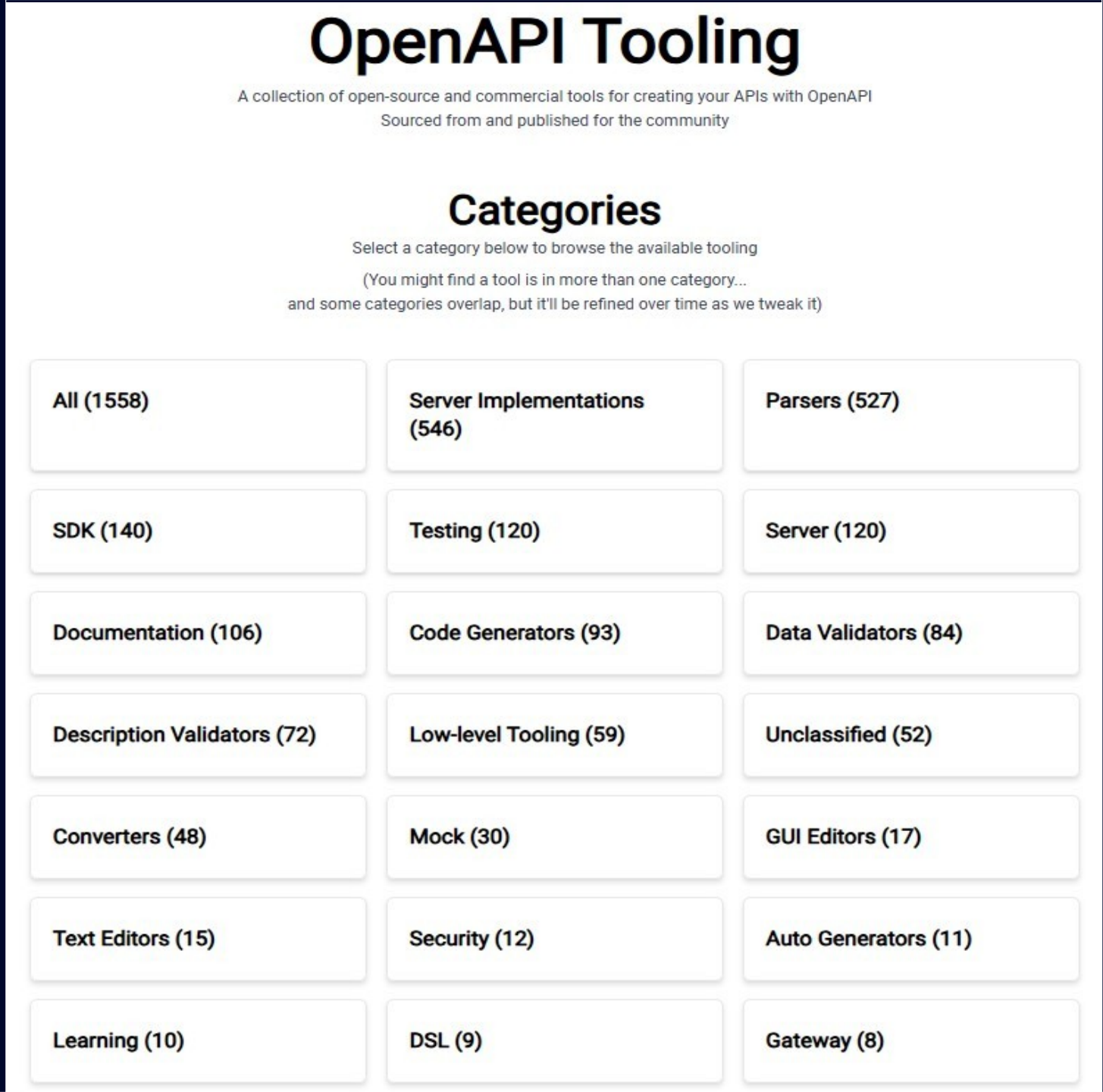
- Popularity.
- Difficulty in the past.
- I like it.

Why Servers? (and not clients?)

- Lots of excellent clients in IVOA
- Largest burden for new implementors:
  - Compliant, easy-to-use servers
  - In their framework/language of choice

Why not use *my* XYZ server?

- Institutional policies around software / security
- It doesn't use our language / framework.
- Reliance on outside support / developers.



Criteria for Success

CRITERIA	NOTES
Running the Tool	How easy is it to run the tool itself? Can I use docker? Does it require Java? NodeJs? Python package?
Valid Python	Are there syntax errors? Obviously broken code? Why is my IDE immediately complaining?
Packaging Best Practices	Does it create a README, requirements.txt, setup.py, a Dockerfile, unit tests?
Python Best Practices	Type annotations, linting, formatting, separation of concerns.
Does it Run?	It's a low bar, I know.
Schema Coverage	Does it cover the relevant endpoints, parameters, requests and responses?
Does it pass the vibe check?	How recently was it updated? Are there 80+ issues open? Does it have passable documentation?

Tools Evaluated

TOOL	LINK
openapi-generators/python-aiohttp	https://openapi-generator.tech/docs/generators/python-aiohttp/
openapi-generators/python-blueplanet	https://openapi-generator.tech/docs/generators/python-blueplanet/
openapi-generators/python-fastapi	https://openapi-generator.tech/docs/generators/python-fastapi/
openapi-generators/python-flask	https://openapi-generator.tech/docs/generators/python-flask/
fastapi-code-generator	https://github.com/koxudaxi/fastapi-code-generator
swagger-codegen/python-flask	https://github.com/swagger-api/swagger-codegen/
oasdiff (not a server generator)	https://github.com/oasdiff/oasdiff

Bonus!

openapi-generators/python-aiohttp

✔ Easy to use the generator tool

✔ Generated requirements.txt, setup.py, README, tests

✔ Logic functions separate from routing

```
catalog = .from_dict(catalog)
format = .from_dict(format)
delimiter = .from_dict(delimiter)
```

❌ Immediately broken.

✔ Custom base model for schemas / parameters.

❌ Used that model for enums?? Instead of, you know:

```
from enum import Enum
```

❌ Inconsistent type-hinting, no enforcement based off OpenAPI docs (3.0 < v < 3.0.0)

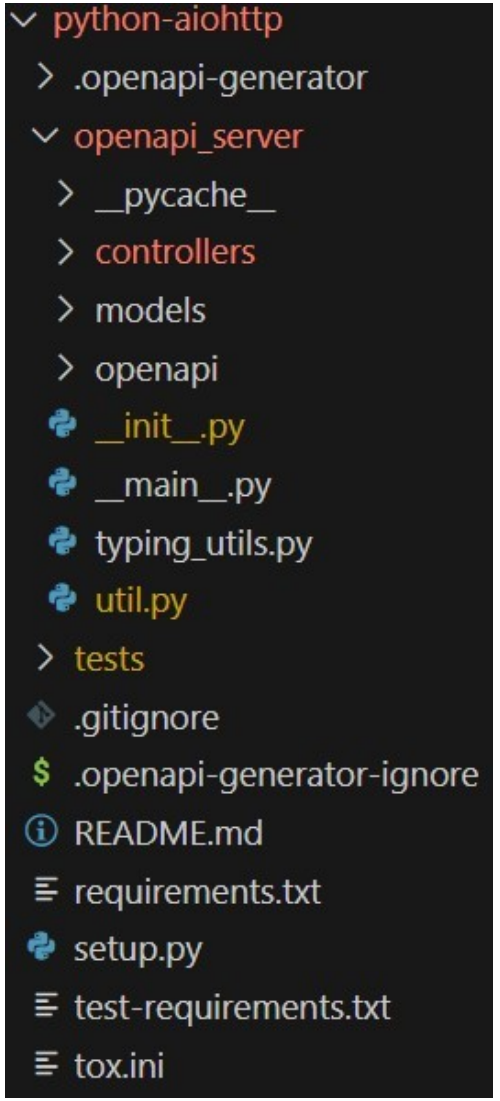
❌ Failed immediately on launch. No OpenAPI 3.1 support (undocumented)

❌ You can just change the version number to “3.0”, *it’s just regex.*

✔ Swagger docs!

❌ Broken imports in test file + relative imports.

✔ Did run, eventually. ❌ Wouldn’t use it personally.



openapi-generators/python-blueplanet

For Ciena's BluePlanet platform, but can be run as a microservice.

✔ Nice verbose terminal output, optional file post-processing.

✔ OpenAPI 3.1.0 beta support. *(It warned me!)*
✔ Generated requirements.txt, setup.py, README, tests **+ Makefile & Dockerfile!**

✔ Nothing immediately broken, no syntax errors.

✔ Swagger docs!
✔ Custom base model for schemas / parameters.

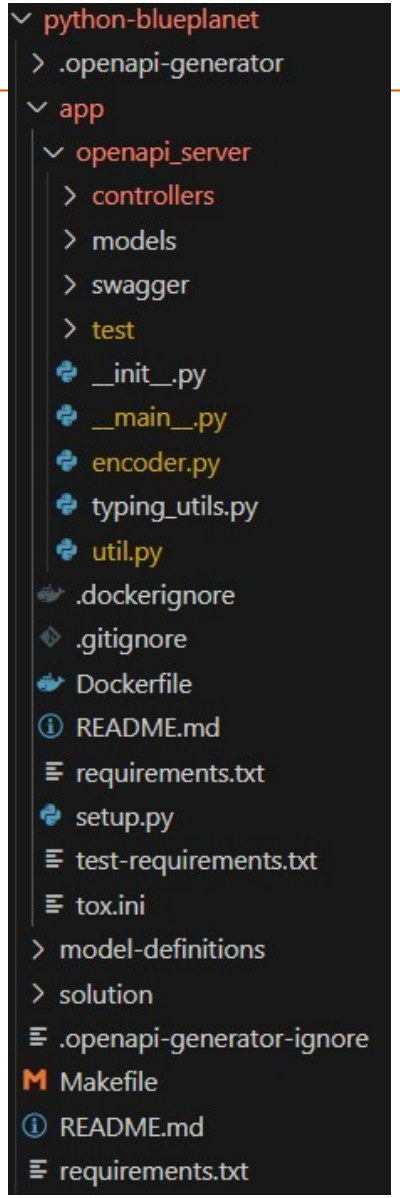
❌ Used that model for enums….

```
from enum import Enum
```

… wait a second …

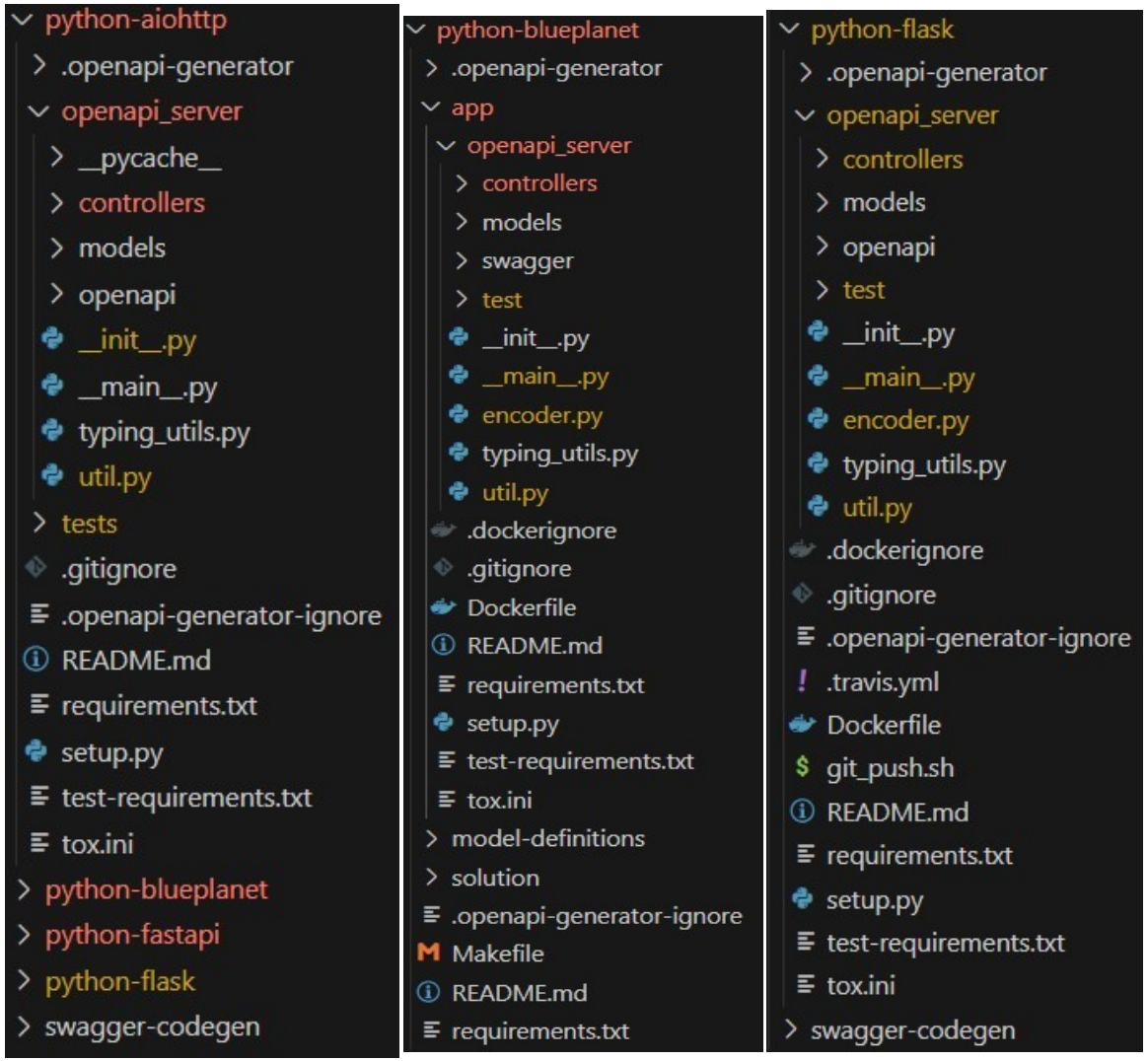
❌ Same broken imports in test file + relative imports.

Both packages appear to use Connexion package under-the-hood.



openapi-generators/python-flask

Can you guess what's about to happen...?



- All use the same underlying generator engine.
- All have the same fundamental limitations.
- The only real changes are the framework they use.

openapi-generators/python-fastapi

✔ Does not use the Connexion engine!

✔ Uses Pydantic for typehinting, validation and enforcement!

✔ Model / routes / implementation separation.

✔ All the package files + goodies **+ docker-compose file!**

✔ No errors, ran out of the box.

❌ Documentation is lacking to say the least.

❌ True of all of the tools under the openapi-generators toolset.

Given how well-documented FastAPI + Pydantic are, it's not a huge impediment to using the package.

```python
@router.get(
    "/vo-conesearch/api/v0.1/{catalog}",
    responses={
        200: {"description": "Successful Response"},
        422: {"model": HTTPValidationError, "description": "Validation Error"},
    },
    tags=["default"],
    summary="Conesearchhandler.Get",
    response_model_by_alias=True,
)
async def conesearch_handler_get(
    catalog: CatalogName = Path(..., description=""),
    ra: Annotated[
        Union[
            Annotated[float, Field(le=360.0, strict=True, ge=0.0)],        ],
        Field(
            description="right-ascension in the ICRS coordinate system for the posit
        ),
    ] = Query(
        description="right-ascension in the ICRS coordinate system for the positon c
        alias="ra",
    ),
    dec: Annotated[
        Union[
            Annotated[float, Field(le=90.0, strict=True, ge=-90.0)],
        ],
        Field(
            description="declination in the ICRS coordinate system for the positon c
```

fastapi-code-generator

Describes itself as experimental, solo developer.

❌ Lots of open Github issues.
❓Seems like the developer doesn't close issues, even if they're finished?

✔ Pydantic data-models from OpenAPI (and GraphQL)

✔ Ability to create custom code templates.

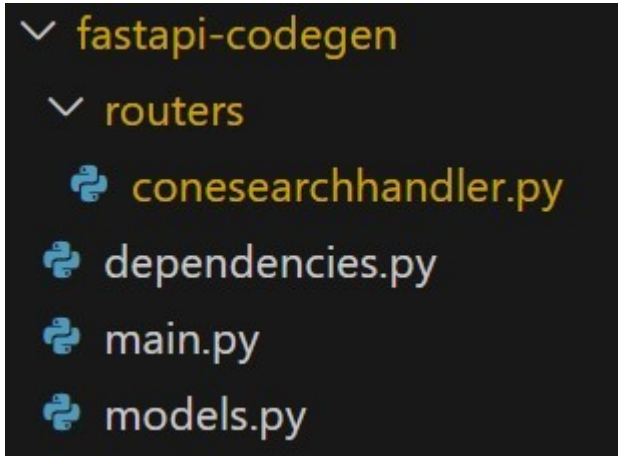
❌ Doesn't use typical FastAPI patterns for annotations.

✔ Properly constrains parameters.

❌ No packaging, produces very minimalistic code.

✔ Ran perfectly when dropped into a barebones package.

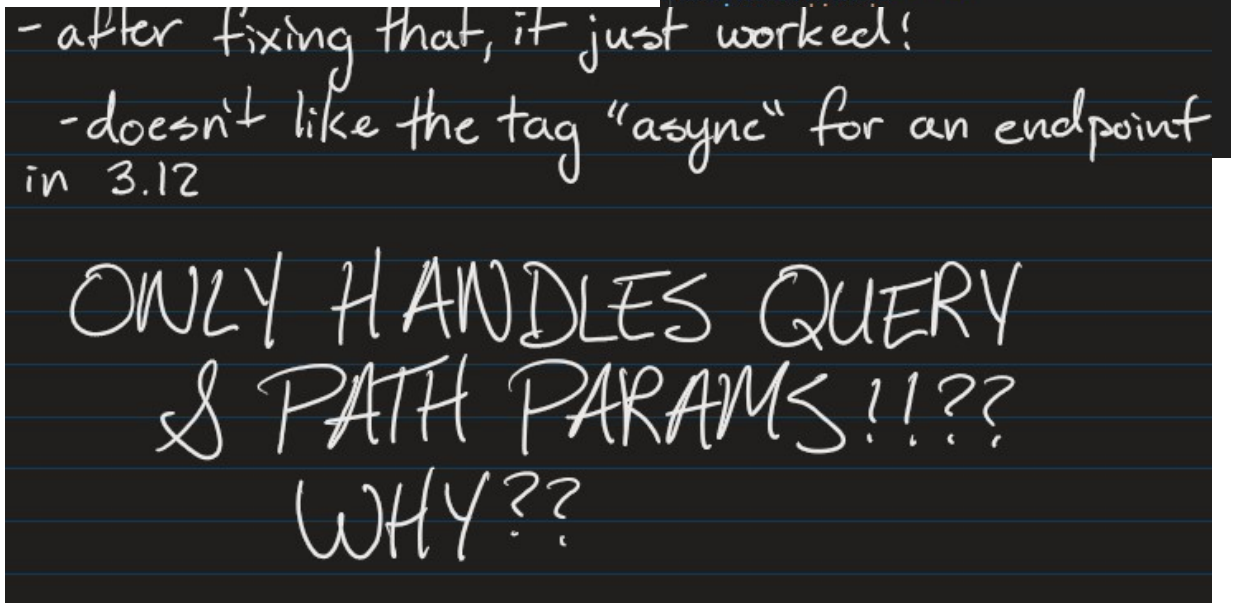
An actual snippet from my presentation notes ->




```python
class CSVDelimiter(Enum):
    space = 'space'
    tab = 'tab'
    comma = 'comma'
    semicolon = 'semicolon'
    pipe = 'pipe'
    iraf = 'iraf'


class CatalogName(Enum):
    caom = 'caom'
    gaiadr3 = 'gaiadr3'
    hsc_summary_aper2 = 'hsc_summary_aper2'
    tic = 'tic'
    ps1dr2 = 'ps1dr2'


class EncodingEnum(Enum):
```



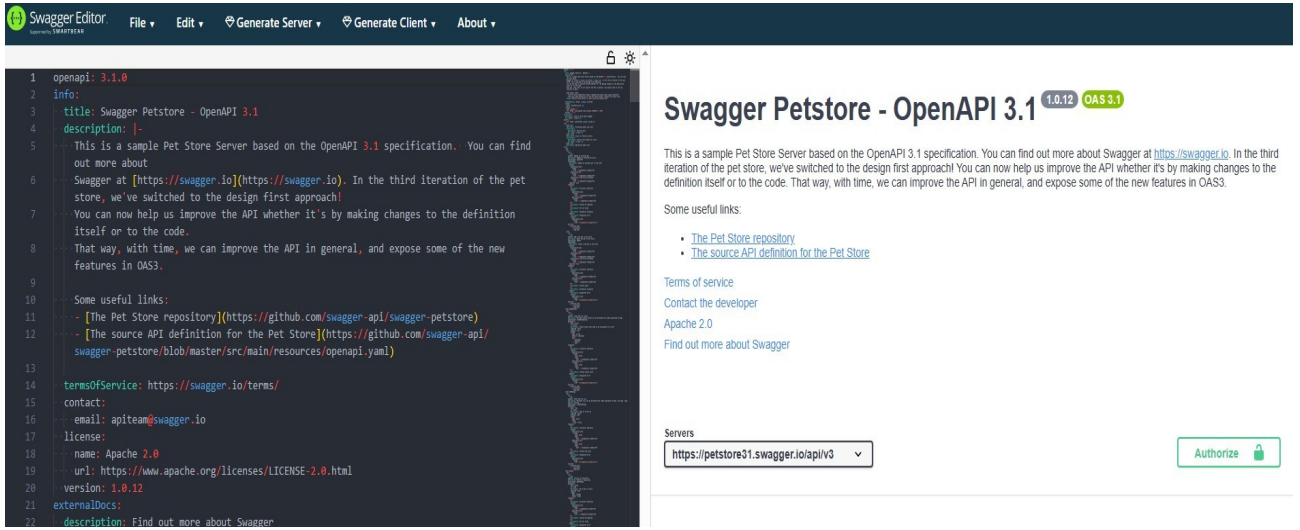
swagger-codegen/python-flask

By Swagger – Smartbear

✔ Java-based CLI, with Docker option.

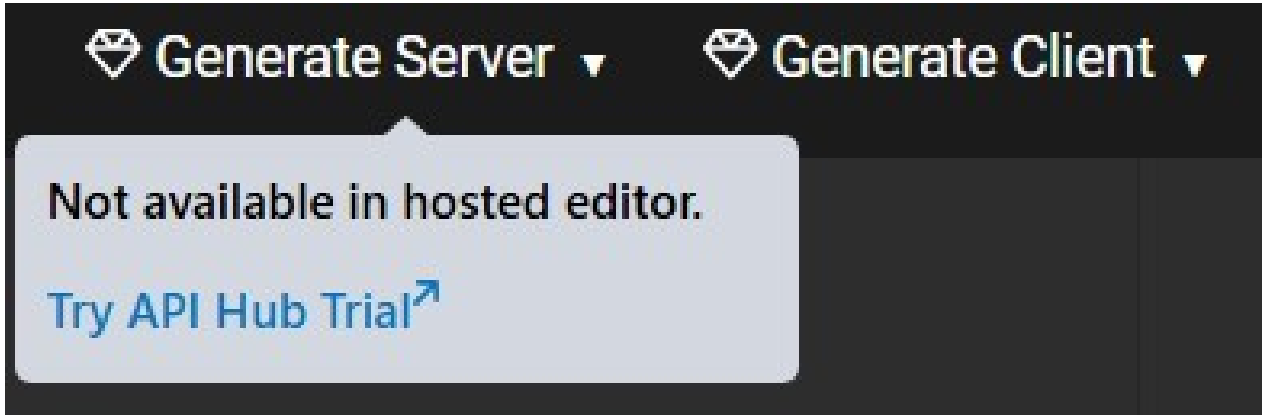
✔ Large organization, lots of support.

✕ It’s just Connexion again.



Swagger has their online SwaggerEditor tool that allowed real-time visualization of the OpenAPI doc and endpoints.

The online tool had options to generate a variety of hosts and clients straight from your browser but...



They put it behind a paywall. Boo.

Takeaways

- Unfortunately, not exceptional news.
- Somewhat disappointed there aren't more options.
- Old issues persist with polymorphism, discriminated models, etc.
- Better options for other languages.

Best Option?

- Currently, probably openapi-generators/python-fastapi
- Developers experienced with FastAPI + Pydantic will be fine with it.

Something to remember:

- Code generation isn't the goal of using OpenAPI
- Just a cherry on top.

TOOL	VERDICT
openapi-generators/python-aiohttp	✗
openapi-generators/python-blueplanet	✗
openapi-generators/python-fastapi	Okay...
openapi-generators/python-flask	✗
fastapi-code-generator	Only very simple APIs
swagger-codegen/python-flask	✗

BONUS – OASDIFF

https://github.com/oasdiff/oasdiff

A diff generator for OpenAPI that detects breaking changes!

- Human readable reports!
- Github actions!

Issues with identifying breaking changes in standards would happen automatically!

```
(openapi-generator-env) PS C:\repos\ivoa-code-generation-tests> .\oasdiff.exe breaking .\sp
1 changes: 1 error, 0 warning, 0 info
error   [request-parameter-became-required] at .\specs\vo-tap-openapi-v2.json
        in API GET /vo-tap/api/v0.1/{catalog}/sync
               the 'query' request parameter 'responseformat' became required
```

Bonus: PetStore IVOA Spec

Can you diff… the LaTeX?

```
1 2.2.3.1. Getting the Pet List
2
3 The list of pets available in the Pet Store may be
  retrieved by sending a GET request to the endpoint /pets.
  In this case, the query parameters that may be included
  in the request are LIMIT, which restricts the number of
  pet items returned, and STATUS, which allows the client
  to specify a filter based on the current availability of
  the pets. The status parameter accepts the values
  "AVAILABLE", "PENDING", or "SOLD". The response to this
  request will contain a JSON array of pet objects, each
  representing a distinct pet record in the system.
4
5 Upon successful retrieval, the response code will be 200
  "OK", and the response body will contain a JSON
  representation of the pets matching the query parameters,
  if any. The server may also respond with a 400 "Bad
  Request" status code if the request parameters are
  invalid (for example, if limit is non-numeric or
  negative).
```

```
1 2.2.3.1. Getting the Pet List
2
3 A client may access the list of pets in the Pet Store by
  initiating a GET request to /pets. Clients can include
  optional query parameters in the request to influence the
  returned data. These parameters include QUANTITY, which
  specifies the desired count of pet items in the response,
  and STATE, which allows clients to specify a filter based
  on the current state of the pets, accepting values such
  as "ACTIVE", "RESERVED", or "SOLD". The server's response
  will include a JSON-formatted array of objects, each
  representing a pet with various attributes.
4
5 On successful execution of the request, the server will
  return a status code of 200 "OK", with a JSON body
  representing the list of pets that meet the specified
  criteria. The server may return a 400 "Bad Request" if it
  detects invalid parameter values, such as a non-numeric
  quantity or unrecognized state value.
```

UWS through OpenAPI, Joshua Fraustro, 2024

API Changelog 1.0.0 vs. 2.0.0

GET /pets UPDATED

- deleted the 'query' request parameter 'limit'
- deleted the 'query' request parameter 'status'