FastAPI

Production-grade JSON APIs made EZ

Patrik Hagara @ Brněnské Pyvo, 2023-10-26

About me

Patrik Hagara

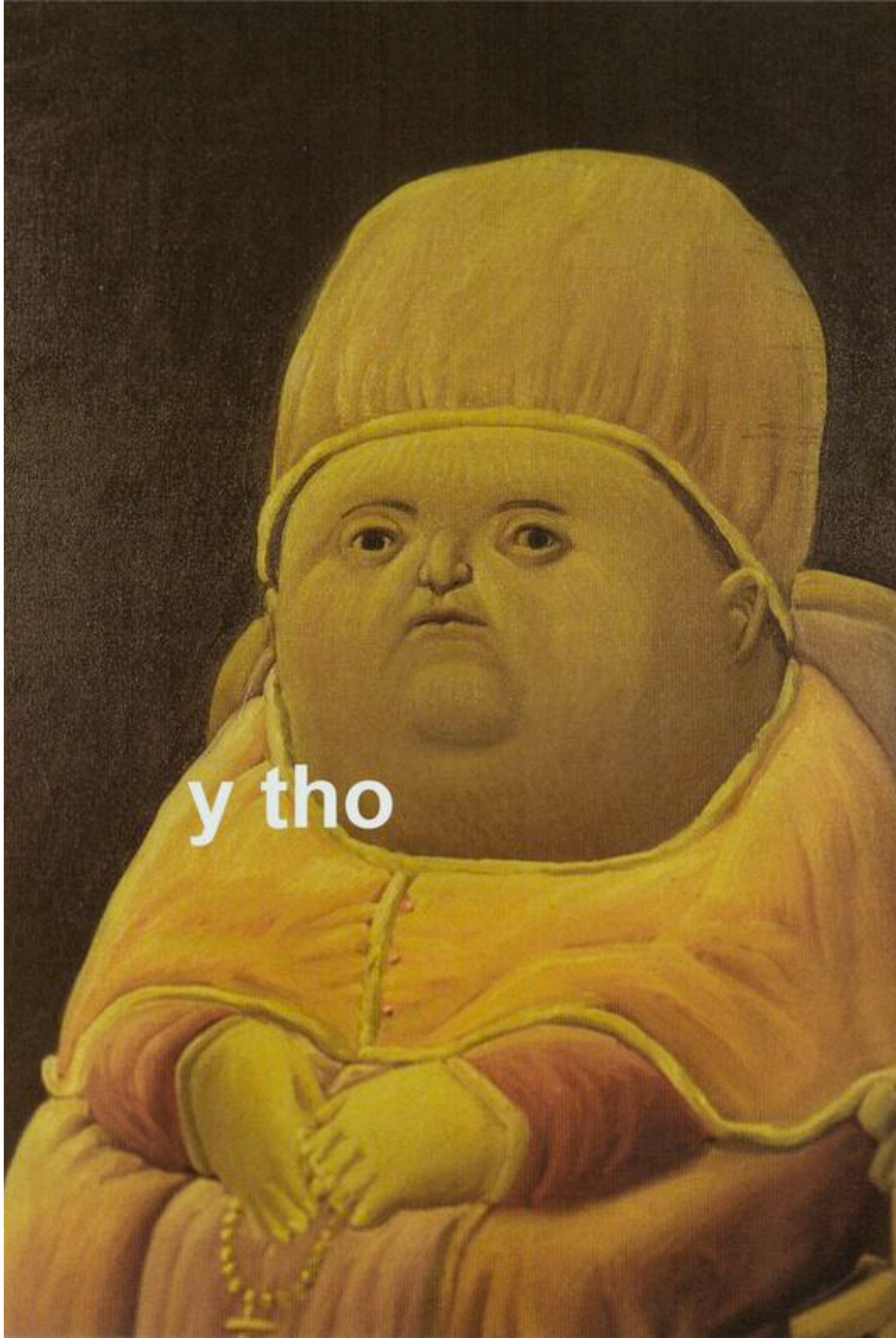
- Reformed “just code it in BASH with a bunch of regexes” programmer
- Likes asyncio, mypy, and writing Python code that looks more like Java
- Cybersecurity and privacy aficionado

Why FastAPI

instead of your other favorite framework?

- Fast
  - well, it depends [1]
- Concise
- Async
- Typed
- Pydantic
  - data validation
- Swagger UI / ReDoc
  - pretty API docs
- OpenAPI
  - machine-readable API docs
  - generate client libraries [2]

[1] https://github.com/tiangolo/fastapi/discussions/7320
[2] https://openapi-generator.tech/



Getting started with FastAPI

Install FastAPI

```
$ pip install "fastapi[all]"
```

- The above installs a bunch of optional dependencies to get you up & running *Fast*, eg.:
  - uvicorn
    - Python ASGI web server to actually run your app
    - ASGI stands for "Asynchronous Server Gateway Interface"
  - ujson / orjson
    - fast JSON en-/decoder libraries written in C / Rust
  - & extra validators for Pydantic, unit testing deps, etc.

Hello, World!

Writing hello.py

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def root() -> dict[str, str]:
    return {"message": "Hello, World!"}
```

API object

HTTP method

path

✨decorator✨

Hello, World!

Serving the app

module

watch for code changes

```
$ uvicorn hello:app --reload
```

ASGI server

object

```
INFO:     Uvicorn running on
          http://127.0.0.1:8000 👈
          (Press CTRL+C to quit)
INFO:     Started reloader process [28720]
INFO:     Started server process [28722]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Hello, World!

Trying it out

```
$ curl http://127.0.0.1:8000
{"message":"Hello, World!"}
```

Dependency injection

Sharing is caring... and avoids code duplication

- FastAPI has a dependency injection system
  - Allows us to share stuff between API routes
  - Reduces code duplication
  - Supports both "normal" and async callables, even classes!
  - Dependencies can have proper type annotations using typing.Annotated (Python 3.9+), keeping mypy happy
  - Can be easily overridden when writing unit tests

Dependency injection

Sharing is caring... and avoids code duplication

```python
from typing import Annotated
from fastapi import Depends, FastAPI

async def word_of_the_day() -> str:
    return "foo"

WordOfTheDay = Annotated[str, Depends(word_of_the_day)]
app = FastAPI()

@app.get("/wotd")
async def get_wotd(word: WordOfTheDay) -> str:
    return word

@app.get("/wotd/reversed")
async def get_wotd_reversed(word: WordOfTheDay) -> str
    return word[::-1]
```

callable

dependency

injected by type thanks to

Doing something useful (with FastAPI)

github repo (code & slides)

https://github.com/phagara/pyvo-fastapi-demo

Snek catalog

Part 0: What do we want to do?

- A catalog for our virtual pet Sneks, in which we can:
  - Create a new Snek
  - Get details about a single Snek
  - List all Sneks
  - Make a Snek grow!
  - Delete Snek 😭
  - Whatever else you can think of!

Images by pch.vector on Freepik

Snek catalog

Part 1: Data modeling

- Sneks have attributes, we're going to track these:
    - Species name
    - Current length
    - Is the snake venomous?
- Time to write a Pydantic model!

Snek catalog

Part 1: Data modeling

```python
from pydantic import BaseModel

class Snek(BaseModel):
    species: str = Field(
        description="Name of the Snek species in English.",
        min_length=3,
        max_length=255,
        examples=["Indian Python", "King Cobra"],
    )
    length: float = Field(
        description="Current length of the Snek in meters.",
        gt=0,  # sneks usually don't have a negative length
        le=20,  # longest snek ever recorded was 10m long
        examples=[3, 3.6],
    )
    venomous: bool = Field(
        description="Whether the Snek species is venomous.",
        examples=[False, True],
    )
```

Snek catalog

Part 1: Data modeling

- But wait... This calls for a database, right? 😐
  - Add Snek ID (primary key) to the attribute list
- Draw a nice DB diagram:

Snek	
id 🔑	integer
species	varchar(255)
length	float
venomous	boolean

Snek catalog

Part 1: Data modeling

- Look for an ORM library with FastAPI integration
  - ORM stands for Object-Relation Mapping
    - Used for representing DB tables & rows as objects in our program
- SQLModel
  - Made by the same guy as FastAPI itself!
  - Combines the power of Pydantic's data validation
    - Which means we won't have to throw away our Snek model, yay!
  - ...with SQLAlchemy (a widely-used ORM library)

Snek catalog

Part 1: Data modeling

```python
from sqlmodel import Field, SQLModel

class Snek(SQLModel, table=True):
    id: int = Field(primary_key=True)
    species: str = Field(
        description="Name of the Snek species in english.",
        min_length=3, max_length=256,
        schema_extra={
            "examples": ["Indian Python", "King Cobra"],
        },
    )
    length: float = Field(
        description="Current length of the Snek in meters.",
        gt=0, le=20,
        schema_extra={
            "examples": [3, 3.6],
        },
    )
    venomous: bool = Field(
        description="Whether the Snek species is venomous.",
        schema_extra={
            "examples": [False, True],
        },
    )
```

notice the examples had to move slightly :(looked like this before:

```python
Field(examples=[...])
```

Snek catalog

Part 1: Data modeling

- Don't want to manually assign IDs when adding new Sneks to the DB!
  - Make it optional, the DB knows how to auto-increment

```python
class Snek(SQLModel, table=True):
    id: int | None = Field(
        default=None,
        primary_key=True,
    )
```

Snek catalog

Part 1: Data modeling

- But now we have weird type annotation for the ID field
  - Need to handle ID being None in all our code
  - While the DB will never return NULL for a primary key...
  - Solution: separate input and output models!

Snek catalog

Part 1: Data modeling

```python
class SnekBase(SQLModel):
    species: str = Field(...)
    length: float = Field(...)
    venomous: bool = Field(...)

class Snek(SnekBase, table=True):
    id: int | None = Field(default=None, primary_key=True)

class SnekCreate(SnekBase):
    pass

class SnekRead(SnekBase):
    id: int = Field(description="Internal ID of the Snek.")
```

not user visible fields

fields visible to users,
put documentation here!

Snek catalog

Part 1: Data modeling

- Now we can convert Snek objects read from the DB into SnekRead objects which we can return to users!
    - And have the ID field's description documented in the OpenAPI data
- When users want to send new Snek data for us to add into the DB, SnekCreate model can be used
    - Does not contain the ID field at all, the DB will fill it in on its own

Snek catalog

Part 1: Data modeling

- Converting from Snek (optional ID) to SnekRead (always has ID):

```
>>> from pyvo_fastapi_demo.models import Snek, SnekRead
>>> db_snek = Snek(
...     id=1,
...     species="Random Python",
...     length=1,
...     venomous=False,
... )
>>> SnekRead.from_orm(db_snek)
SnekRead(species='Random Python', length=1.0,
venomous=False, id=1)
```

Snek catalog

Part 1: Data modeling

- Wait... Why was the "id" field syntax highlighted in green?
  - It's the built-in id() function!
- Linters will go nuts!

```
class Snek(SnekBase, table=True):
    id: int | None = Field(...)
```

- How do we fix this?
  - Rename the field to something else, eg. "id_"
  - But we want to keep the "id" name in DB and when returning data to the API users!
    - Add an alias

Snek catalog

Part 1: Data modeling

```python
class Snek(SnekBase, table=True):
    id_: int | None = Field(
        default=None,
        primary_key=True,
        alias="id",
    )

class SnekRead(SnekBase):
    id_: int = Field(
        description="Internal ID of the Snek.",
        alias="id",
    )
```

Snek catalog

Part 1: Data modeling

```
>>> from pyvo_fastapi_demo.models import Snek, SnekRead
>>> db_snek = Snek(id_=1, species="Random Python", length=1, venomous=False)
>>> SnekRead.from_orm(db_snek)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File ".../python3.11/site-packages/sqlmodel/main.py", line 554, in from_orm
    raise validation_error
pydantic.error_wrappers.ValidationError: 1 validation error for SnekRead
id
  field required (type=value_error.missing)
```

Snek catalog

Part 1: Data modeling

- Fix: allow populating fields by their alias

this feature comes from Pydantic

```
class SnekBase(SQLModel):
    class Config:
        allow_population_by_field_name = True

>>> from pyvo_fastapi_demo.models import Snek, SnekRead
>>> db_snek = Snek(id_=1, species="Random Python",
length=1, venomous=False)
>>> SnekRead.from_orm(db_snek)
SnekRead(species='Random Python', length=1.0,
venomous=False, id_=1)
```

Snek catalog

Part 1: Data modeling

- We also need a model for our other responses, like when making a Snek grow longer.

```
class OperationResult(BaseModel):
    result: str
```

- No need to specify any constraints for the "result" field, as it's an output-only model (does not accept user input)

Snek catalog

Part 2: Connect to database

- We'll most likely need to use the DB from multiple API endpoints
- Use dependency injection for the DB session!

Snek catalog

Part 2: Connect to database

```python
from typing import Annotated, Iterator
from fastapi import Depends
from sqlmodel import Session, create_engine

DB_ENGINE = create_engine(
    "sqlite:///snek-catalog.sqlite3",
    echo=True,
    connect_args={"check_same_thread": False},
)

def db_session() -> Iterator[Session]:
    with Session(DB_ENGINE) as session:
        yield session

DBSession = Annotated[Session, Depends(db_session)]
```

log executed SQL statements, useful for debugging

workaround specific to SQLite, related to thread-safety checks

we're creating a new DB session for each request, so it's safe*

* https://ricardoanderegg.com/posts/python-sqlite-thread-safety/

Snek catalog

Part 2: Connect to database

```python
from fastapi import FastAPI
from sqlmodel import SQLModel

app = FastAPI(
    on_startup=[
        lambda: SQLModel.metadata.create_all(DB_ENGINE),
    ],
)
```

beware!

- Make sure all your models were imported, as this relies on subclass hooks!

Snek catalog

Part 3: Add authentication

- Don't want random strangers on the Internet to delete all our Sneks or add new ones!
  - Many types of auth (non-exhaustive list)
    - HTTP "Authorization" header
      - Basic (pesky username/password pop-up in browser)
      - Bearer tokens (eg. JWT) -- hard to get right!
    - API keys
      - URL query string (hope you don't accidentally screen share)
      - HTTP header (eg. "X-API-Key")
      - Cookie (also HTTP header, just more clunky; consent pop-ups for API clients?!)
  - Delegating auth to a 3rd party (eg. the usual "Log in via Google/Facebook/Apple/...")
    - SAML (old and bad)
    - OAuth 2.0 (fairly standard nowadays)
    - OpenID Connect Discovery (best, but not as widely deployed)

this one is easy!

Snek catalog

Part 3: Add authentication

```python
from fastapi import HTTPException, Security, status
from fastapi.security import APIKeyHeader

API_KEY_HEADER = APIKeyHeader(name="X-API-Key")
SECRET_API_KEY = "sikrit"

def check_api_key(
    key: Annotated[str, Security(API_KEY_HEADER)],
) -> None:
    if key != SECRET_API_KEY:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid API key!",
        )
```

Snek catalog

Part 3: Add authentication

```python
from fastapi import FastAPI

app = FastAPI()

@app.get(
    "/secure",
    dependencies=[Depends(check_api_key)],
)
def get_secure() -> str:
    return "This endpoint sure is secure!"
```

Snek catalog

Part 3: Add authentication

```
$ curl -D- http://127.0.0.1:8000/secure
HTTP/1.1 403 Forbidden
date: Mon, 23 Oct 2023 13:52:25 GMT
server: uvicorn
content-length: 30
content-type: application/json

{"detail":"Not authenticated"}
```

Snek catalog

Part 3: Add authentication

```
$ curl -D- \
    -H 'X-API-Key: sikrit' \
    http://127.0.0.1:8000/secure
HTTP/1.1 200 OK
date: Mon, 23 Oct 2023 13:54:17 GMT
server: uvicorn
content-length: 31
content-type: application/json

"This endpoint sure is secure!"
```

Snek catalog

Part 4: Create API endpoints

- Now we have all the tidbits ready:
  - Data models
  - DB connection dependency
  - API key authentication dependency
- It's time to finally implement some API endpoints! 🚀

Snek catalog

Part 4: Create API endpoints

endpoints get grouped by tags in the generated OpenAPI spec

```python
from fastapi import HTTPException

@app.get("/snek/{snek_id}", tags=["public"])
async def get_snek(db: DBSession, snek_id: int) -> SnekRead:
    snek = db.get(Snek, snek_id)
    if snek is None:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Snek not found!",
        )
    return SnekRead.from_orm(snek)
```

normally, return values get automagically converted to the declared return type, but mypy doesn't know that

Snek catalog

Part 4: Create API endpoints

```python
from fastapi.responses import RedirectResponse
from fastapi import Depends, Request, status

@app.post("/snek", dependencies=[Depends(check_api_key)], tags=["admin"])
async def add_snek(
    request: Request,
    db: DBSession,
    snek: SnekCreate,
) -> RedirectResponse:
    db_snek = Snek.from_orm(snek)
    db.add(db_snek)
    db.commit()
    db.refresh(db_snek)
    return RedirectResponse(
        status_code=status.HTTP_201_CREATED,
        url=request.url_for("get_snek", snek_id=db_snek.id_),
    )
```

Snek catalog

Part 4: Create API endpoints

```python
@app.delete(
    "/snek/{snek_id}",
    dependencies=[Depends(check_api_key)],
    tags=["admin"],
)
async def remove_snek(db: DBSession, snek_id: int) -> SnekRead:
    snek = db.get(Snek, snek_id)
    if snek is None:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Snek not found!",
        )
    db.delete(snek)
    db.commit()
    return SnekRead.from_orm(snek)
```

Snek catalog

Part 4: Create API endpoints

```python
@app.get("/", tags=["public"])
async def list_sneks(db: DBSession) -> list[SnekRead]:
    query = select(Snek).order_by(Snek.id_)
    return [
        SnekRead.from_orm(x)
        for x in db.exec(query).all()
    ]
```

Snek catalog

Part 4: Create API endpoints

```python
import asyncio
import secrets

@app.post("/snek/{snek_id}/sleep", tags=["public"])
async def sleep_snek(db: DBSession, snek_id: int) -> OperationResult:
    snek = db.get(Snek, snek_id)
    if snek is None:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Snek not found!",
        )
    await asyncio.sleep(3)
    growth_mm = secrets.randbelow(20) + 1
    snek.length += growth_mm / 1_000
    db.add(snek)
    db.commit()
    return OperationResult(
        result=f"{snek.species} had a nice nap and grew by {growth_mm} mm.",
    )
```

Snek catalog

Part 4: Create API endpoints

- We've now raised a 404 exception multiple times with the same arguments... Time to deduplicate our code!

```
class SnekNotFound(HTTPException):
    def __init__(self) -> None:
        super().__init__(
            status_code=status.HTTP_404_NOT_FOUND,
            detail="Snek not found!",
        )
```

- Rewrite our exception raising code bits like this:

```
if snek is None:
    raise SnekNotFound()
```

Snek catalog

Part 4: Create API endpoints

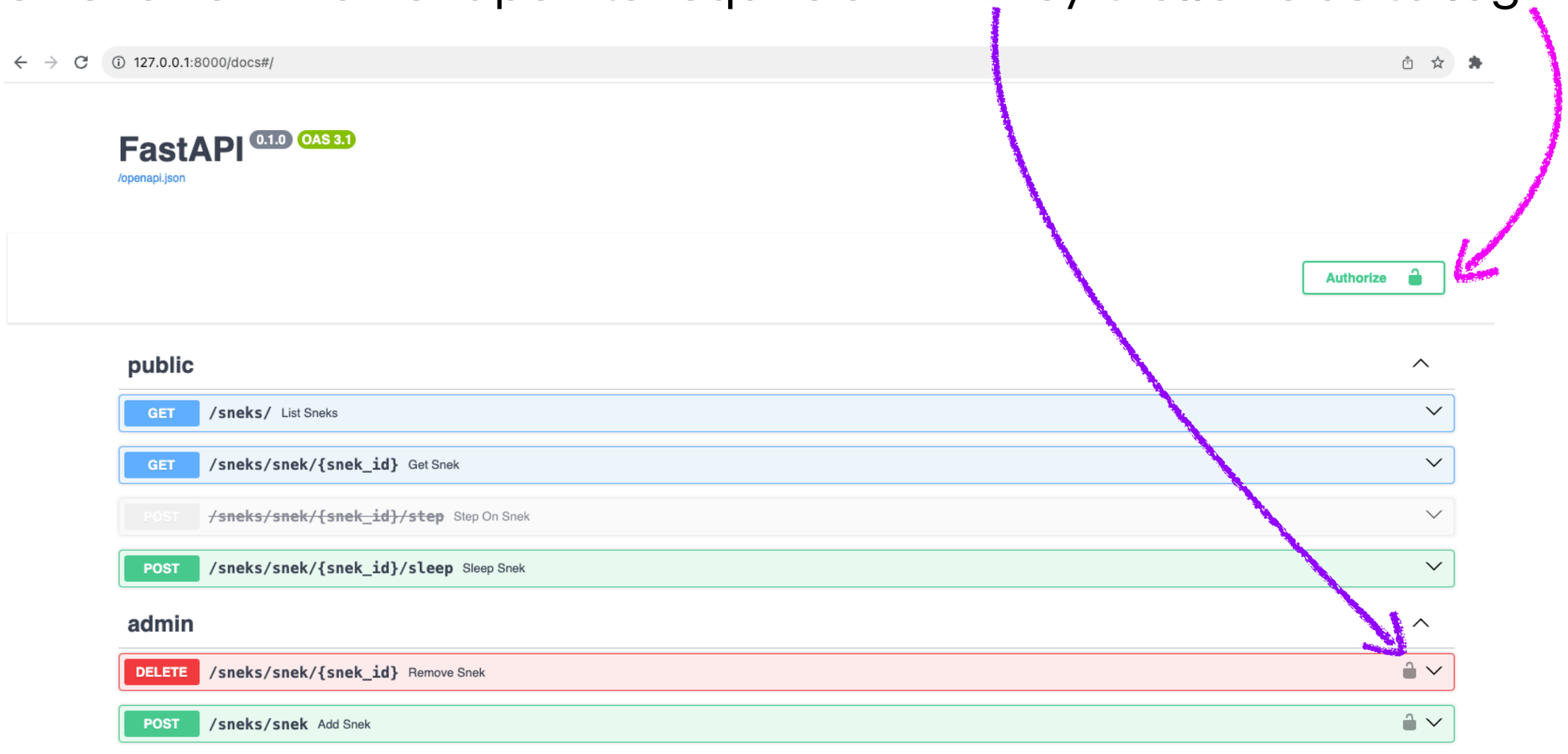
- API endpoints can be easily marked as deprecated.

```python
@app.post(
    "/snek/{snek_id}/step",
    description="Don't try this at home!",
    deprecated=True,
    tags=["public"],
)
async def step_on_snek(db: DBSession, snek_id: int) -> OperationResult:
    snek = db.get(Snek, snek_id)
    if snek is None:
        raise SnekNotFound()
    return OperationResult(
        result="You died." if snek.venomous else "No step on Snek!",
    )
```

Snek catalog

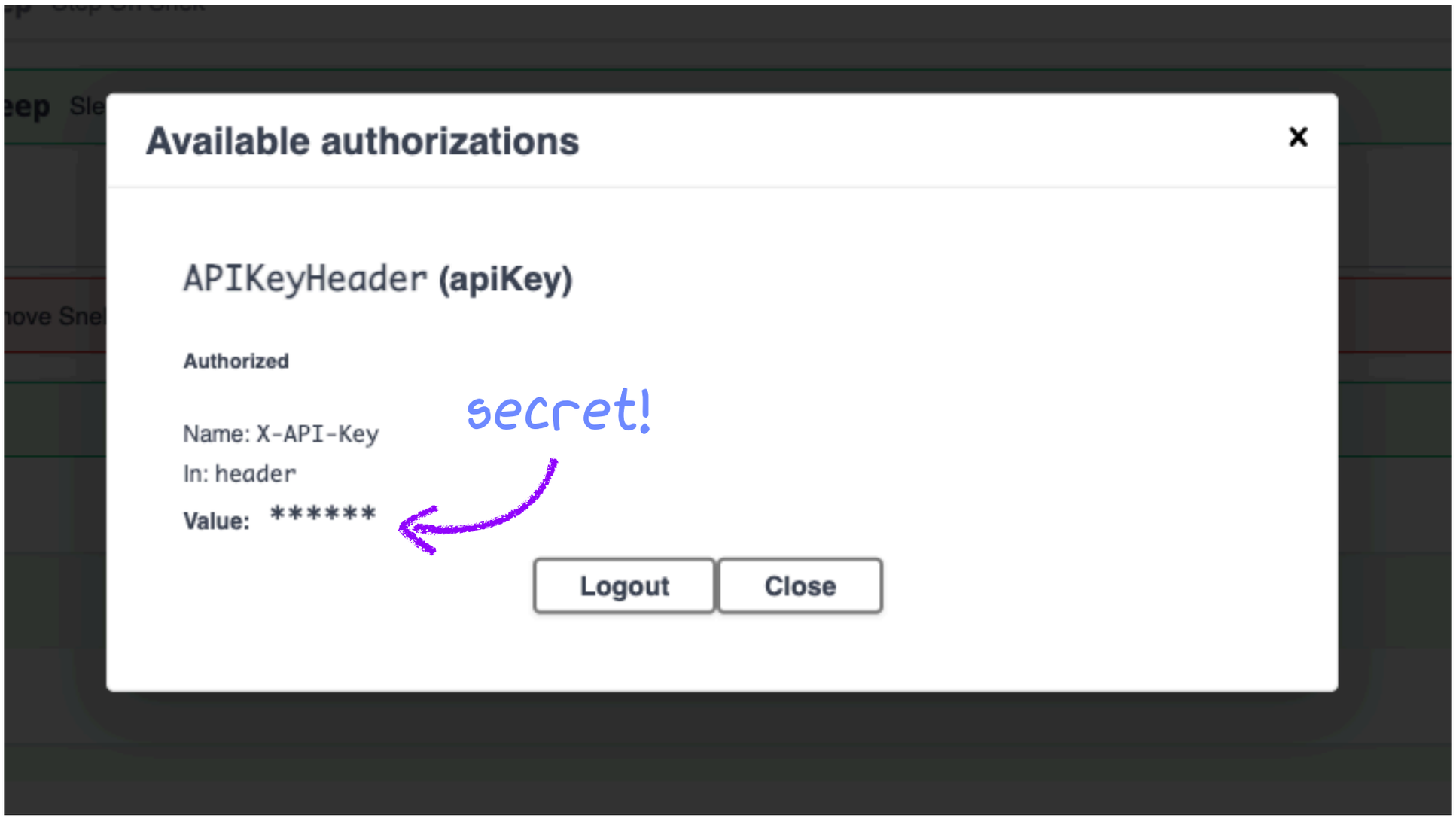
Part 5: Look at the auto-generated docs

- Pretty colors and everything, powered by Swagger UI!
- Even shows which endpoints require an API key & allows us to log in!



Snek catalog

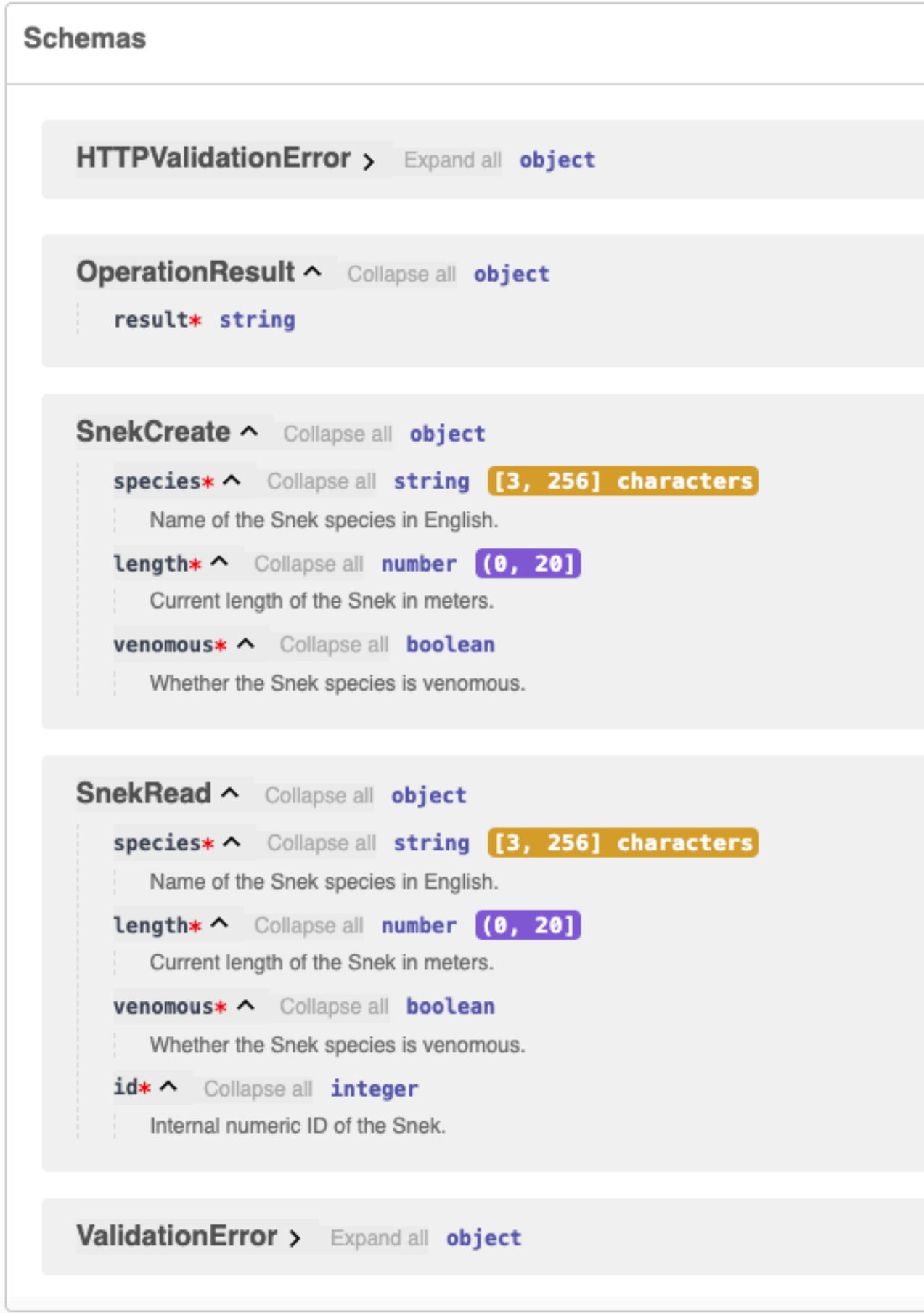
Part 5: Look at the auto-generated docs



Snek catalog

Part 5: Look at the auto-generated docs

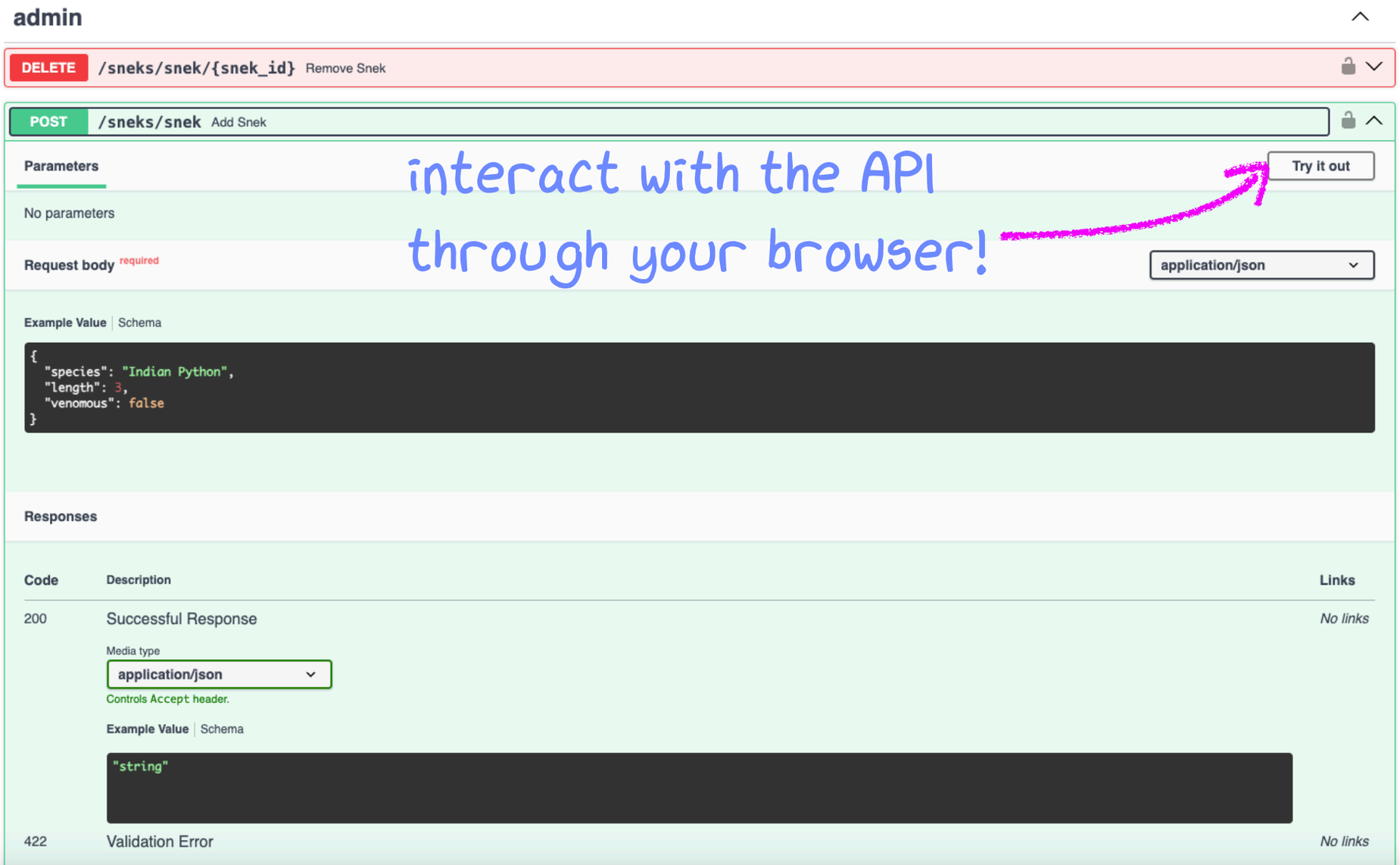
- Full schema of our models is included at the bottom! 🎉



Snek catalog

Part 5: Look at the auto-generated docs

- Expanding an endpoint shows input examples and the schema.

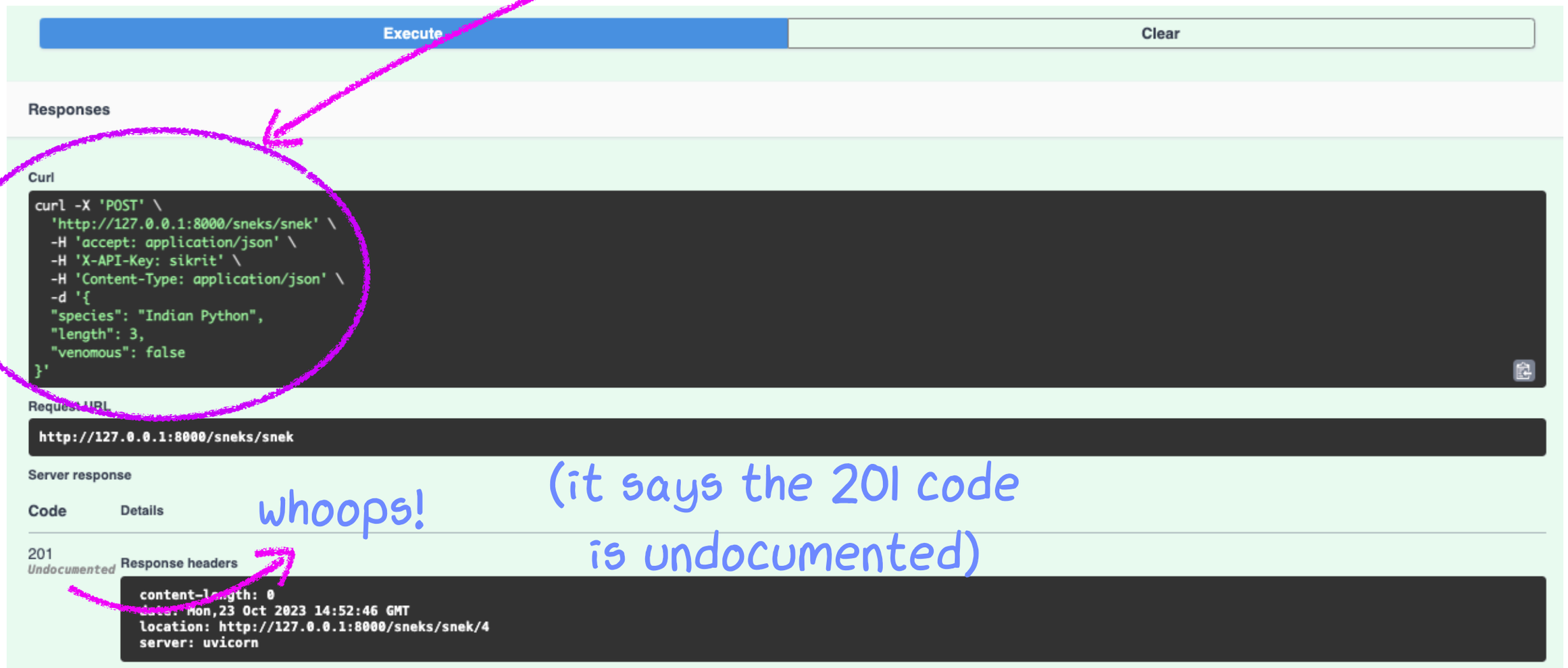


interact with the API through your browser!

Snek catalog

Part 5: Look at the auto-generated docs

- Shows the full curl command to reproduce the request.



Snek catalog

Part 5: Look at the auto-generated docs

```python
@app.post(
    "/snek",
    status_code=status.HTTP_201_CREATED,
    responses={
        status.HTTP_201_CREATED: {
            "headers": {
                "location": {
                    "description": "URL of the new Snek",
                },
            },
        },
    },
    dependencies=[Depends(check_api_key)],
    tags=["admin"],
)
async def add_snek(
    request: Request,
    db: DBSession,
    snek: SnekCreate,
) -> RedirectResponse:
    ...
```

easy fix!

btw, same should be done for 401/403 auth error codes & the good ol' 404

Snek catalog

Part 6: Add paginated search

we'll put link to the next page of results into HTTP headers, document it here

```python
from fastapi import Response

@app.get(
    "/search",
    responses={
        status.HTTP_200_OK: {
            "headers": { "link": {"description": "URL to the next page of results, if available."}},
        },
    },
    tags=["public"],
)
async def search_sneks(
    request: Request,
    response: Response,
    db: DBSession,
    name: str | None = Query(default=None, description="Search by part of the species name."),
    min_length: float | None = Query(default=0, description="Sneks at least this long (in meters)."),
    max_length: float | None = Query(default=None, description="Sneks at most this long (in meters)."),
    venomous: bool | None = Query(default=None, description="Return only (non-)venomous Sneks."),
    offset: int = Query(default=0, ge=0, description="Internal Snek ID to start the search from."),
    limit: int = Query(default=100, le=100, description="Return at most this many matching sneks."),
) -> list[SnekRead]:
    ...
```

filters

pagination

Snek catalog

Part 6: Add paginated search

```python
async def search_sneks(
    request: Request, response: Response, db: DBSession, name: str | None = ...,
    min_length: float | None = ..., max_length: float | None = ..., venomous: bool | None = ...,
    offset: int = ..., limit: int = ...,
) -> list[SnekRead]:
    query = (
        select(Snek)
        .where(
            col(Snek.id_) >= offset,
            name is None or col(Snek.species).contains(name),
            min_length is None or col(Snek.length) >= min_length,
            max_length is None or col(Snek.length) <= max_length,
            venomous is None or col(Snek.venomous) == venomous,
        )
        .order_by(Snek.id_)
        .limit(limit)
    )
    sneks = [SnekRead.from_orm(x) for x in db.exec(query).all()]
    if sneks:
        links: dict[str, URL] = {}
        links["next"] = request.url.include_query_params(offset=sneks[-1].id_ + 1)
        response.headers["link"] = ", ".join(
            [f'<{v}>; rel="{k}"' for k, v in links.items()]
        )
    return sneks
```

Snek catalog

Part 7: API routers

- Isolate logical, then attach them at different URL prefixes of the main app object

```python
from fastapi import APIRouter, FastAPI

SnekRouter = APIRouter()

@SnekRouter.get("/sound")
def get_snek_sound() -> str:
    return "Ssssss."

RodentRouter = APIRouter()

@RodentRouter.get("/sound")
def get_rodent_sound() -> str:
    return "Squeak!"

app = FastAPI()
app.include_router(SnekRouter, prefix="/sneks")
app.include_router(RodentRouter, prefix="/rodents")
```

GET /sneks/sound => Ssssss.

GET /rodents/sound => Squeak!

Snek catalog

Part 8: Testing

- Don't touch the production DB!
  - Need some kind of mocking
  - Let's use pytest
    - It has this cool thing called "fixtures"
      - Same concept as FastAPI dependencies
        - Dependency injection!
- Use an in-memory SQLite DB for testing
- Clean slate for every single unit test

Snek catalog

Part 8: Testing

```python
import pytest
from sqlmodel import Session, SQLModel, create_engine
from sqlmodel.pool import StaticPool

@pytest.fixture(name="db_session")
def db_session_fixture():
    engine = create_engine(
        "sqlite://",
        connect_args={"check_same_thread": False},
        poolclass=StaticPool,
    )
    SQLModel.metadata.create_all(engine)
    with Session(engine) as session:
        yield session
```

in-memory DB

ensure both our unit tests and the app we're testing use the same connection, as new connection == new DB (due to ●)

Snek catalog

Part 8: Testing

```python
import pytest
from fastapi.testclient import TestClient
from sqlmodel import Session

from pyvo_fastapi_demo.dependencies import db_session as orig_db_session
from pyvo_fastapi_demo.main import app


@pytest.fixture(name="client")
def client_fixture(db_session: Session):
    def get_db_session_override():
        yield db_session

    app.dependency_overrides[orig_db_session] = get_db_session_override
    yield TestClient(app)
    del app.dependency_overrides[orig_db_session]
```

Snek catalog

Part 8: Testing

```python
def test_create_snek(client: TestClient):
    response = client.post(
        "/sneks/snek",
        json={
            "species": "Indian Python",
            "length": 3,
            "venomous": False,
        },
        headers={
            "X-API-Key": SECRET_API_KEY,
        },
    )
    assert response.status_code == 201
    assert "location" in response.headers
    assert response.headers["location"].endswith("/snek/1")
```

Thank you?

Questions!

github repo
(code & slides)

https://github.com/phagara/pyvo-fastapi-demo