Combining Django ORM & FastAPI in a Single App

Mia Bajić

My personal experience with Python frameworks

Flask

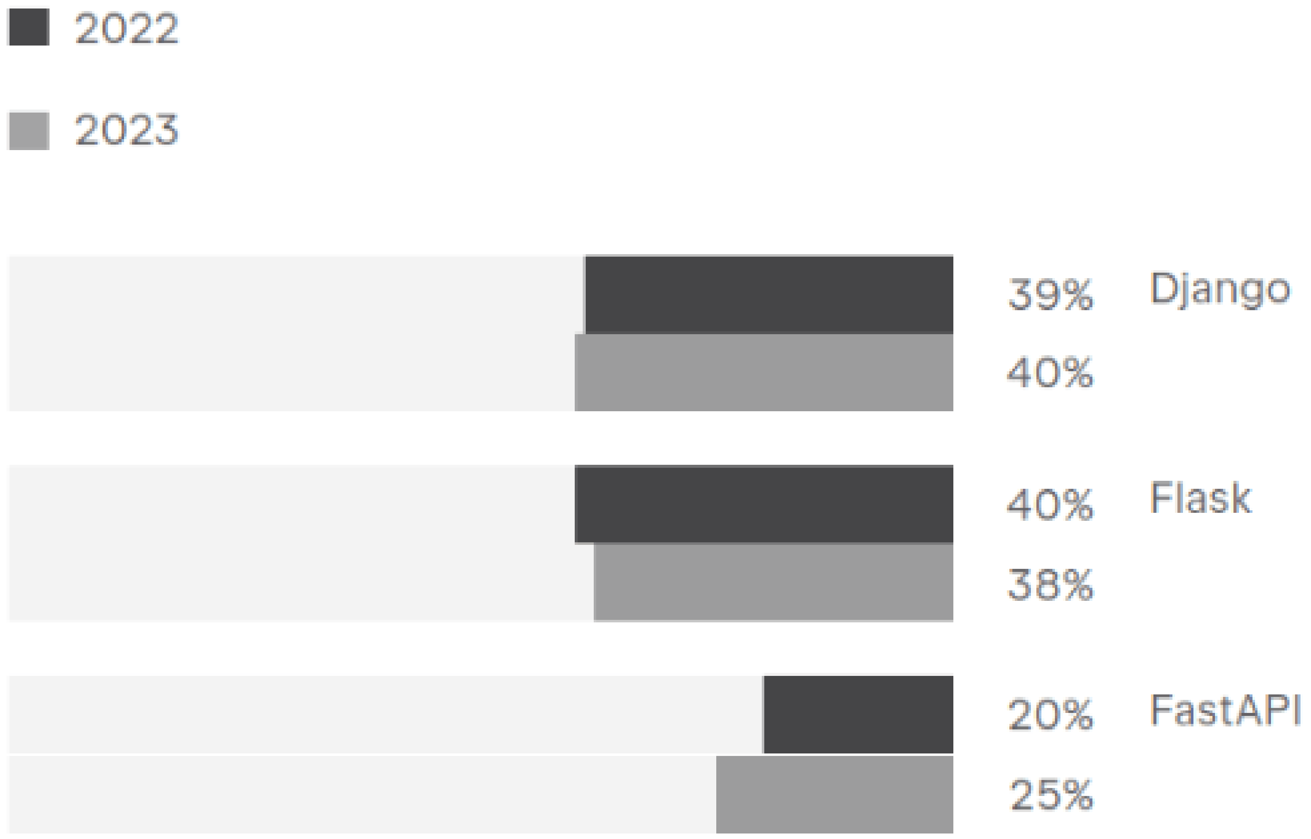
My personal experience with Python frameworks

Flask ➡ Django

My personal experience with Python frameworks

$$\begin{tikzcd}\text{Flask} \arrow[r] & \text{Django} \arrow[r] & \text{FastAPI}\end{tikzcd}$$

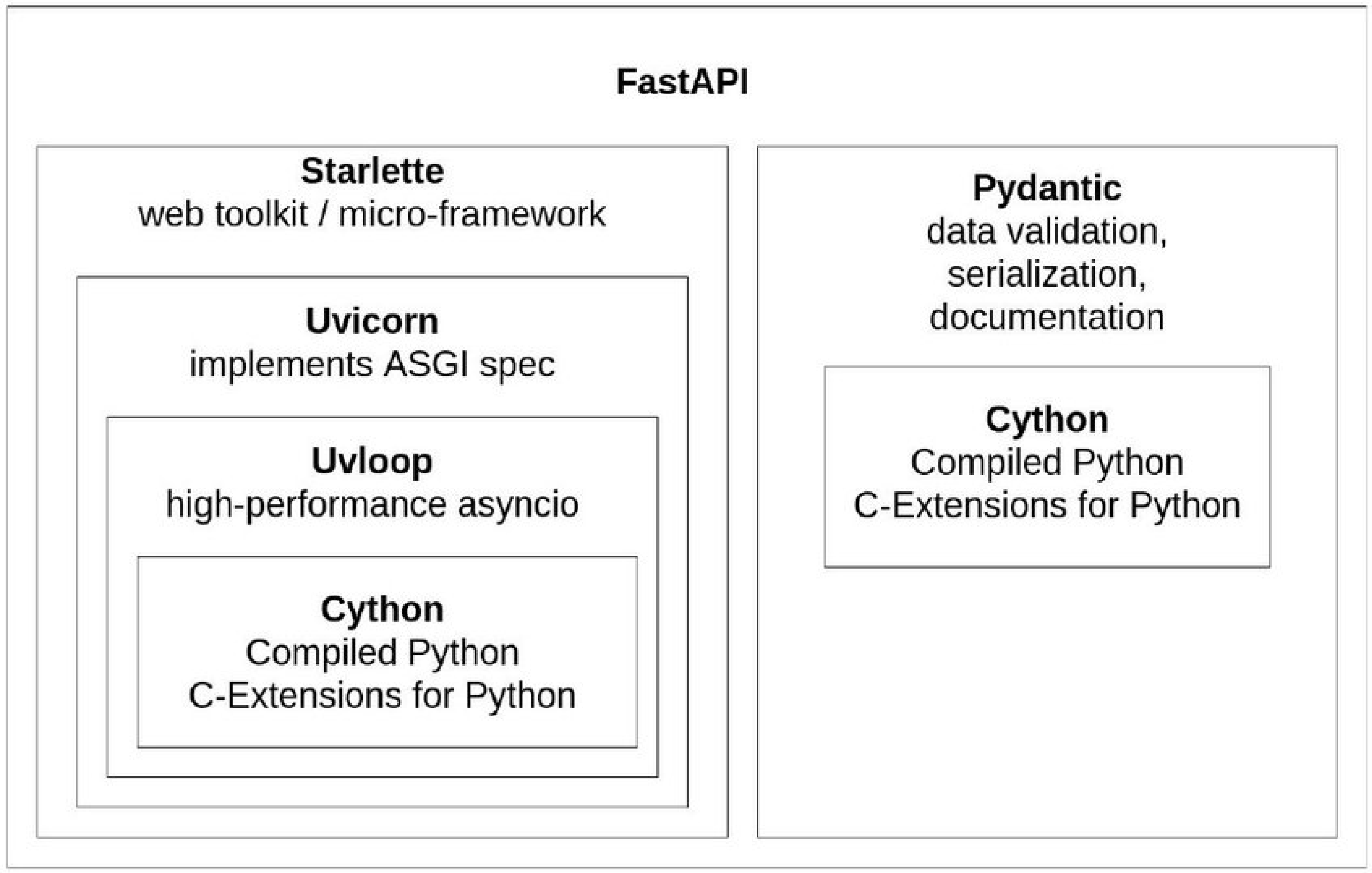
What web frameworks/libraries do you use in addition to Python?



FastAPI has seen increasing usage over the past couple of years, rising from 14% in 2021 to 25% in 2023.

What is FastAPI?

FastAPI is a modern and flexible framework, with the first version released in December 2018.


FastAPI
Starlette
web toolkit / micro-framework
Uvicorn
implements ASGI spec
Uvloop
high-performance asyncio
Cython
Compiled Python
C-Extensions for Python
Pydantic
data validation,
serialization,
documentation
Cython
Compiled Python
C-Extensions for Python

@tiangolo

The Simplest Example

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Validation with Pydantic

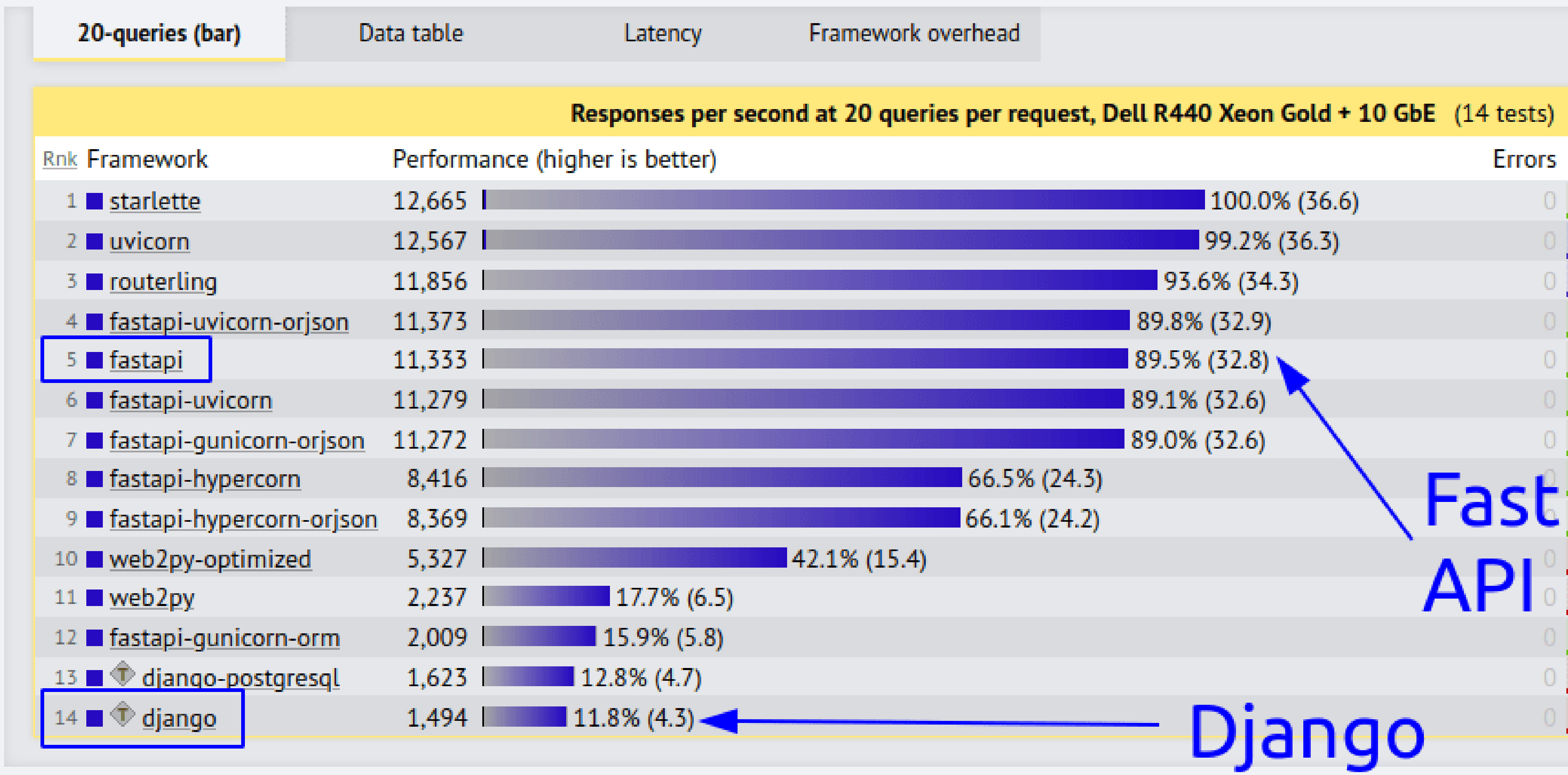
```python
from typing import Union

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class Item(BaseModel):
    name: str
    price: float
    tax: float | None = None
    tags: list[str] = []


@app.post("/items/", response_model=Item)
async def create_item(item: Item) -> Item:
    return item
```

Database interaction

```python
# crud.py
def get_items(db: Session, skip: int = 0, limit: int = 100):
    return db.query(models.Item).offset(skip).limit(limit).all()


# services.py
@app.get("/items/", response_model=list[schemas.Item])
def read_items(skip: int = 0, limit: int = 100, db: Session = Depends(get_db)):
    items = crud.get_items(db, skip=skip, limit=limit)
    return items
```

Why FastAPI?

FastAPI is performant.

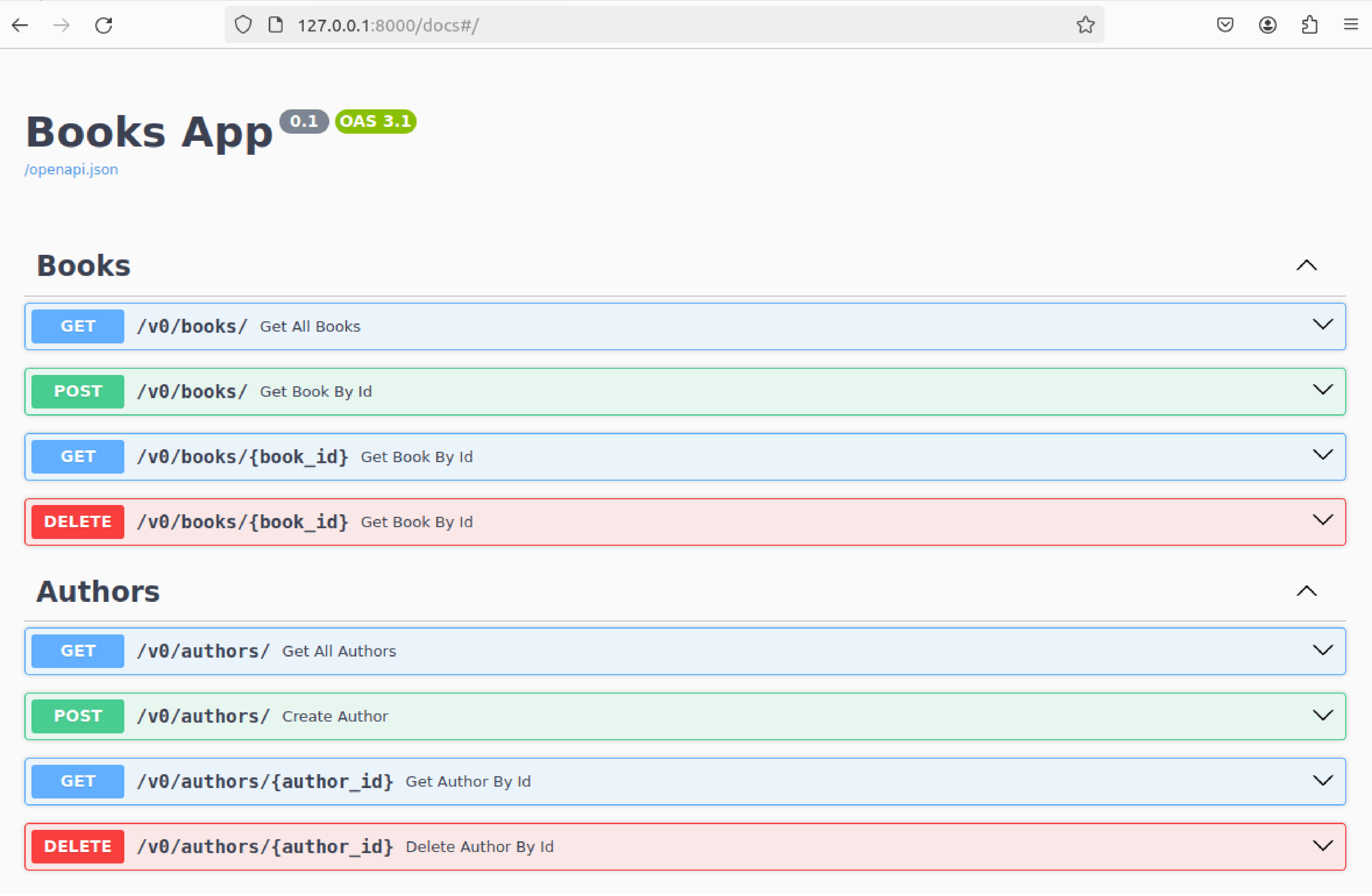


Responses per second at 20 queries per request, Dell R440 Xeon Gold + 10 GbE (14 tests)

Rnk	Framework	Performance (higher is better)		Errors
1	starlette	12,665	100.0% (36.6)	0
2	uvicorn	12,567	99.2% (36.3)	0
3	routerling	11,856	93.6% (34.3)	0
4	fastapi-uvicorn-orjson	11,373	89.8% (32.9)	0
5	fastapi	11,333	89.5% (32.8)	0
6	fastapi-uvicorn	11,279	89.1% (32.6)	0
7	fastapi-gunicorn-orjson	11,272	89.0% (32.6)	0
8	fastapi-hypercorn	8,416	66.5% (24.3)	0
9	fastapi-hypercorn-orjson	8,369	66.1% (24.2)	0
10	web2py-optimized	5,327	42.1% (15.4)	0
11	web2py	2,237	17.7% (6.5)	0
12	fastapi-gunicorn-orm	2,009	15.9% (5.8)	0
13	django-postgresql	1,623	12.8% (4.7)	0
14	django	1,494	11.8% (4.3)	0

Source: https://tinyurl.com/djangocon24-benchmarks

FastAPI offers documentation of endpoints.



Books App 0.1 OAS 3.1

/openapi.json

Books

GET /v0/books/ Get All Books

POST /v0/books/ Get Book By Id

GET /v0/books/{book_id} Get Book By Id

DELETE /v0/books/{book_id} Get Book By Id

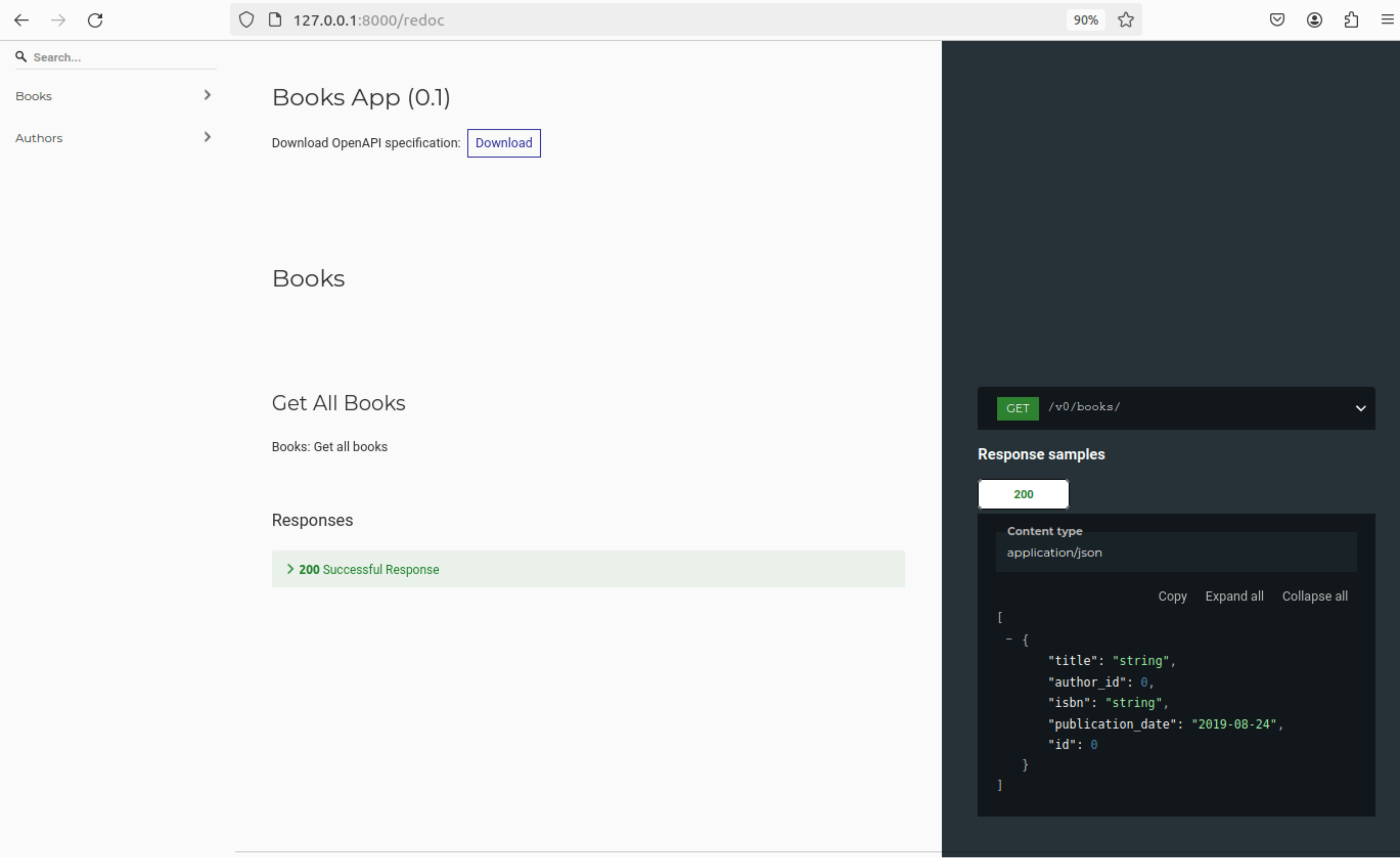
Authors

GET /v0/authors/ Get All Authors

POST /v0/authors/ Create Author

GET /v0/authors/{author_id} Get Author By Id

DELETE /v0/authors/{author_id} Delete Author By Id



Books App (0.1)

Download OpenAPI specification: Download

Books

Get All Books

Books: Get all books

Responses

200 Successful Response

GET /v0/books/

Response samples

200

Content type
application/json

```
[
  - {
        "title": "string",
        "author_id": 0,
        "isbn": "string",
        "publication_date": "2019-08-24",
        "id": 0
    }
]
```

FastAPI utilizes dependency injection to enhance testability.

```python
from fastapi import FastAPI, Depends
import httpx

class MyExternalAPI:
    def __init__(self, url: str):
        self.url = url

    async def fetch(self):
        async with httpx.AsyncClient() as client:
            response = await client.get(self.url)
            return response

def get_external_api():
    return MyExternalAPI(url="https://google.com")
```

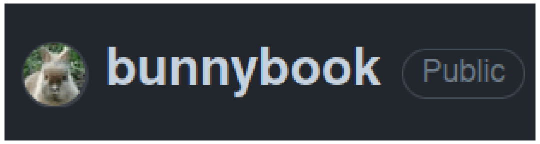
```python
@app.get("/fetch-data")
async def fetch_data(api: MyExternalAPI = Depends(get_external_api)):
    return await api.fetch()
```

```python
from fastapi.testclient import TestClient
from unittest.mock import AsyncMock


def test_fetch_data_from_external_api(client):

    mock_api = MyExternalAPI(url="https://google.com")
    mock_api.fetch = AsyncMock(return_value={"msg": "Mocked response"})


    app.dependency_overrides[get_external_api] = lambda: mock_api

    response = client.get("/fetch-data")

    assert response.status_code == 200
    assert response.json() == {"msg": "Mocked response"}
```

FastAPI is straightforward and simple.

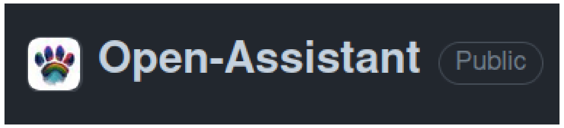
What ORM is commonly used with FastAPI?

dispatch Public


bunnybook Public

RasaGPT Public

mealie Public


Open-Assistant Public

So... why do you use Django ORM with FastAPI?

Django has a powerful and user-friendly built-in admin panel.

Django ORM is simple and easy to learn.

What are the challenges of using Django ORM with FastAPI?

```
django.core.exceptions.SynchronousOnlyOperation:
You cannot call this from an async context - use a
thread or sync_to_async.
```

Managing synchronous and asynchronous code adds a layer of complexity.

Lots of tools from the rich Django ecosystem either cannot be used or have to be tweaked.

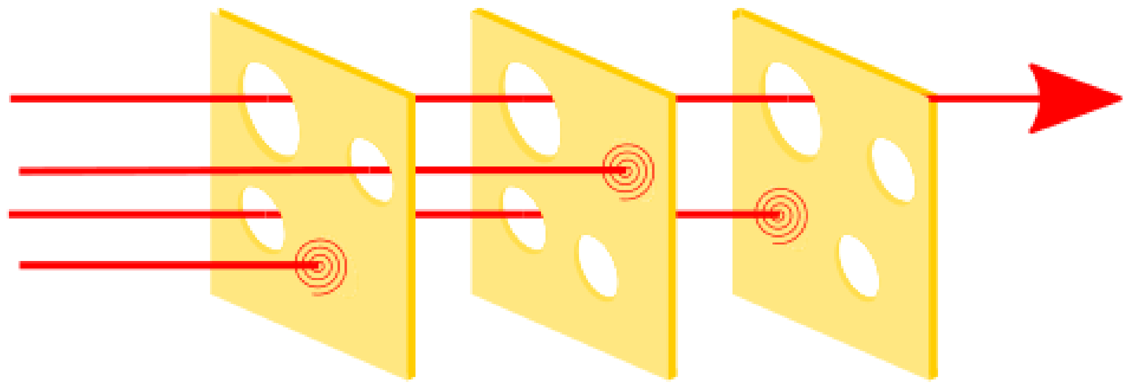
There are not many resources about this particular setup.

Only one person has merge rights in FastAPI.

If I suddenly died being hit by a bus, there are already mechanisms in place for code inheritance, there are others with GitHub rights to take over, etc. but I have requested to still review each PR myself.

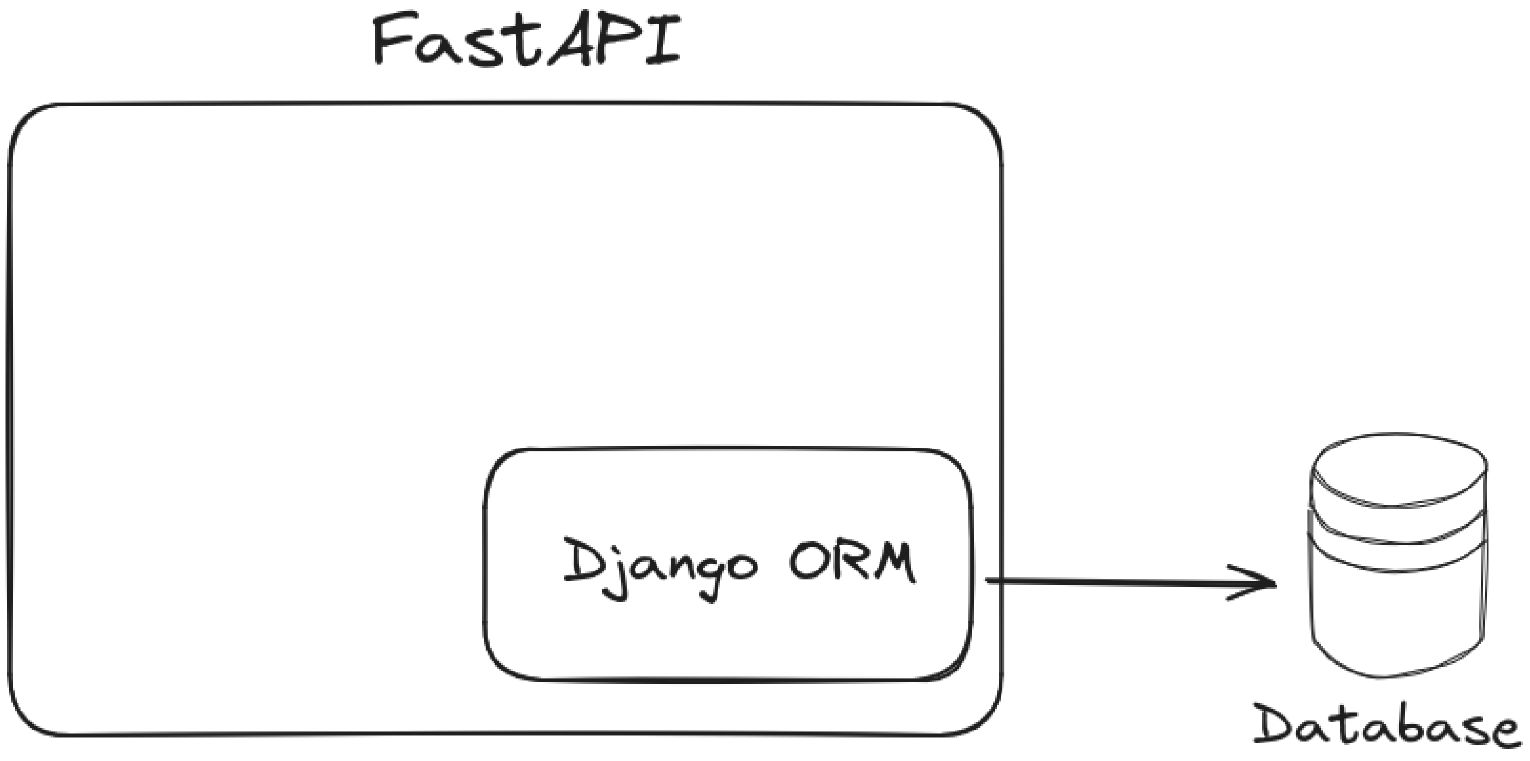
There have been several cases where there's a PR with several approvals (5 or so) where no one really reviewed the code, and it had a bug. Because of this, unfortunately, I still prefer to be quite strict about reviewing each PR that is merged. [redacted]

Swiss cheese model



The inital setup may take more time.

How to set it up?


FastAPI
Django ORM
Database

```python
# settings.py

"""
Django settings for books project.

Generated by 'django-admin startproject' using Django 5.0.4.

For more information on this file, see
https://docs.djangoproject.com/en/5.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.0/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = "django-insecure-8x@alz@%&=fqvp=nd@qnp69fn=9u3k(qk+n64n_(7+g7=

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

```python
# wsgi.py

"""
WSGI config for books project.

It exposes the WSGI callable as a module-level variable named ``application`

For more information on this file, see
https://docs.djangoproject.com/en/5.0/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")

application = get_wsgi_application()
```

```python
# main.py

import os
import importlib
from books.settings import INSTALLED_APPS
from django import setup
from django.apps import apps
from fastapi import FastAPI

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")
setup()

apps.populate(INSTALLED_APPS)
```

```python
# main.py

import os
import importlib
from books.settings import INSTALLED_APPS
from django import setup
from django.apps import apps
from fastapi import FastAPI

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")
setup()

apps.populate(INSTALLED_APPS)
```

```python
# main.py

import os
import importlib
from books.settings import INSTALLED_APPS
from django import setup
from django.apps import apps
from fastapi import FastAPI

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")
setup()

apps.populate(INSTALLED_APPS)
```

```python
# main.py

import os
import importlib
from books.settings import INSTALLED_APPS
from django import setup
from django.apps import apps
from fastapi import FastAPI

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")
setup()

apps.populate(INSTALLED_APPS)

app = FastAPI(title="Books App", version="0.1")

```

```python
# main.py

import os
import importlib
from books.settings import INSTALLED_APPS
from django import setup
from django.apps import apps
from fastapi import FastAPI

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")
setup()

apps.populate(INSTALLED_APPS)

app = FastAPI(title="Books App", version="0.1")

routes = [
    ("books.api.books.routes", "/v0/books", ["Books"]),
    ("books.api.authors.routes", "/v0/authors", ["Authors"]),
]
```

```python
# main.py

import os
import importlib
from books.settings import INSTALLED_APPS
from django import setup
from django.apps import apps
from fastapi import FastAPI

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")
setup()

apps.populate(INSTALLED_APPS)

app = FastAPI(title="Books App", version="0.1")

routes = [
    ("books.api.books.routes", "/v0/books", ["Books"]),
    ("books.api.authors.routes", "/v0/authors", ["Authors"]),
]

for router_module_path, prefix, tags in routes:
    router_module = importlib.import_module(router_module_path)
    app.include_router(
        getattr(router_module, "router"),
        tags=tags,
        prefix=prefix,
    )
```

```dockerfile
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "books.main:app", "--host", "0.0.0.0", "--port",
"8000"]
```

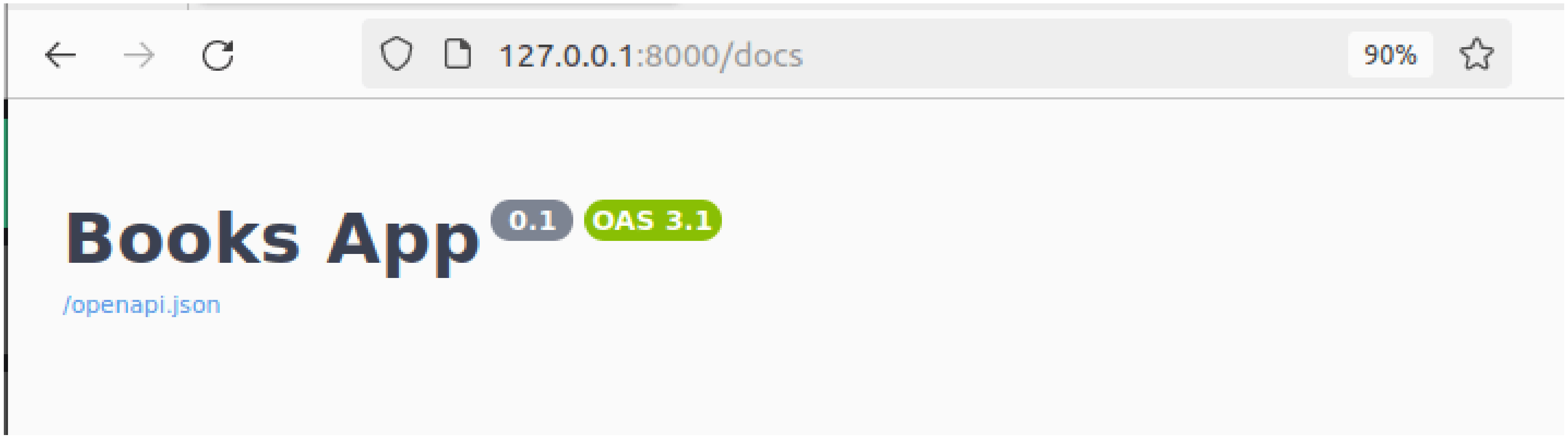
```dockerfile
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "books.main:app", "--host", "0.0.0.0", "--port",
"8000"]
```

```dockerfile
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "books.main:app", "--host", "0.0.0.0", "--port",
"8000"]
```

```dockerfile
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "books.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

```dockerfile
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "books.main:app", "--host", "0.0.0.0", "--port", "8(
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "books.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

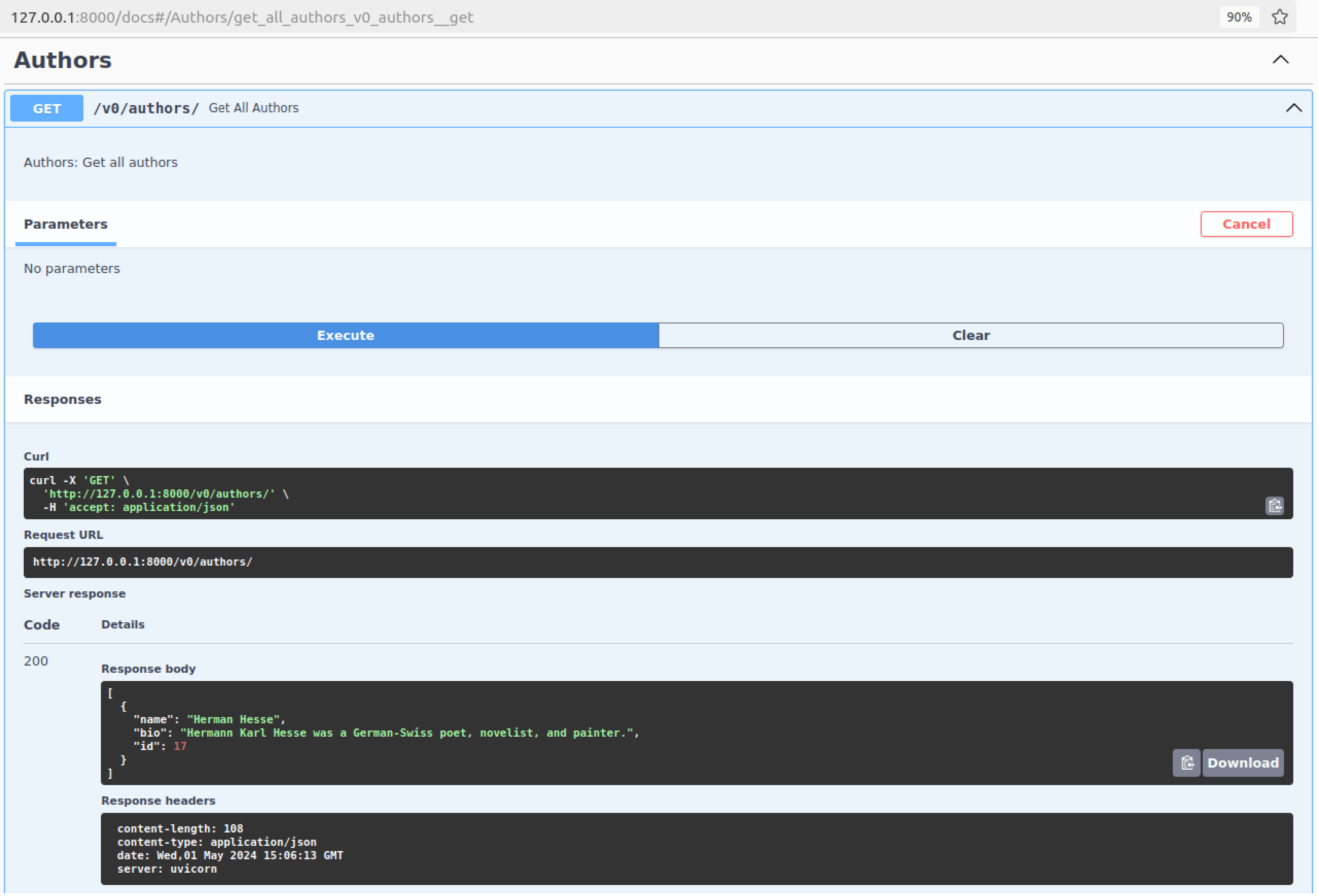
```
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["uvicorn", "books.main:app", "--host", "0.0.0.0", "--port",
"8000"]
```


127.0.0.1:8000/docs
90%
Books App
0.1
OAS 3.1
/openapi.json

```python
@router.get(
    "/{author_id}",
    description="Authors: Get an author by id",
    response_model=ReadAuthorSchema,
)
async def get_author_by_id(author_id: int):
    return await sync_to_async(Author.objects.get)(id=author_id)
```

```python
@router.get(
    "/{author_id}",
    description="Authors: Get an author by id",
    response_model=ReadAuthorSchema,
)
async def get_author_by_id(author_id: int):
    return await sync_to_async(Author.objects.get)(id=author_id)
```

```python
@router.get(
    "/{author_id}",
    description="Authors: Get an author by id",
    response_model=ReadAuthorSchema,
)
async def get_author_by_id(author_id: int):
    return await Author.objects.aget(id=author_id)
```


127.0.0.1:8000/docs#/Authors/get_all_authors_v0_authors__get
90%
Authors
GET
/v0/authors/
Get All Authors

Authors: Get all authors

No parameters

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/v0/authors/' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/v0/authors/
```

Server response

Code Details

200

Response body

```
[
  {
    "name": "Herman Hesse",
    "bio": "Hermann Karl Hesse was a German-Swiss poet, novelist, and painter.",
    "id": 17
  }
]
```

Response headers

```
 content-length: 108
 content-type: application/json
 date: Wed,01 May 2024 15:06:13 GMT
 server: uvicorn
```

```python
# conftest.py

def pytest_sessionstart():
    import os
    from books.settings import INSTALLED_APPS
    from django import setup
    from django.apps import apps
    from django.core import management


    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")
    setup()
    apps.populate(INSTALLED_APPS)

    management.call_command("migrate")
```

```python
# conftest.py

def pytest_sessionstart():
    import os
    from books.settings import INSTALLED_APPS
    from django import setup
    from django.apps import apps
    from django.core import management


    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")
    setup()
    apps.populate(INSTALLED_APPS)

    management.call_command("migrate")
```

```python
# conftest.py

def pytest_sessionstart():
    import os
    from books.settings import INSTALLED_APPS
    from django import setup
    from django.apps import apps
    from django.core import management


    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "books.settings")
    setup()
    apps.populate(INSTALLED_APPS)

    management.call_command("migrate")
```

```python
# conftest.py

@pytest.fixture(scope="function")
def db(django_db_setup):
    import django.apps
    from django.db import connection

    all_models = django.apps.apps.get_models()
    tables = [model._meta.db_table for model in all_models]

    with connection.cursor() as cursor:
        for table in tables:
            cursor.execute(f"TRUNCATE TABLE {table} CASCADE;")
```

```python
# conftest.py

@pytest.fixture(scope="session")
def books_app() -> "FastAPI":
    from books.main import app

    yield app
```

```python
# conftest.py

@pytest.fixture(scope="session")
def books_app() -> "FastAPI":
    from books.main import app

    yield app


@pytest.fixture(scope="session")
def client(books_app) -> TestClient:
    yield TestClient(books_app, base_url="http://testserver:8001")
```

```python
# test_books.py

class FakeBookRepository:
    def __init__(self):
        self.books = []
        self.next_id = 1

    def create(self, title, author_id, isbn, publication_date):
        book = {
            "id": self.next_id,
            "title": title,
            "author_id": author_id,
            "isbn": isbn,
            "publication_date": publication_date,
        }
        self.books.append(book)
        self.next_id += 1
        return book
```

```python
# test_books.py

class FakeBookRepository:
    def __init__(self):
        self.books = []
        self.next_id = 1

    def create(self, title, author_id, isbn, publication_date):
        book = {
            "id": self.next_id,
            "title": title,
            "author_id": author_id,
            "isbn": isbn,
            "publication_date": publication_date,
        }
        self.books.append(book)
        self.next_id += 1
        return book
```

```python
# test_books.py

@pytest.mark.asyncio
async def test_create_book():
    request = CreateBookRequest(
        title="Test Book",
        author_id=1,
        isbn="123-4567890123",
        publication_date=date.today(),
    )
```

```python
# test_books.py

@pytest.mark.asyncio
async def test_create_book():
    request = CreateBookRequest(
        title="Test Book",
        author_id=1,
        isbn="123-4567890123",
        publication_date=date.today(),
    )

    fake_book_repository = FakeBookRepository()
    book_service = BookService(repository=fake_book_repository)
```

```python
# test_books.py

@pytest.mark.asyncio
async def test_create_book():
    request = CreateBookRequest(
        title="Test Book",
        author_id=1,
        isbn="123-4567890123",
        publication_date=date.today(),
    )

    fake_book_repository = FakeBookRepository()
    book_service = BookService(repository=fake_book_repository)

    await book_service.create_book(request)
```

```python
# test_books.py

@pytest.mark.asyncio
async def test_create_book():
    request = CreateBookRequest(
        title="Test Book",
        author_id=1,
        isbn="123-4567890123",
        publication_date=date.today(),
    )

    fake_book_repository = FakeBookRepository()
    book_service = BookService(repository=fake_book_repository)

    await book_service.create_book(request)

    created_book = fake_book_repository.books[-1]
```

```
# test_books.py

@pytest.mark.asyncio
async def test_create_book():
    request = CreateBookRequest(
        title="Test Book",
        author_id=1,
        isbn="123-4567890123",
        publication_date=date.today(),
    )

    fake_book_repository = FakeBookRepository()
    book_service = BookService(repository=fake_book_repository)

    await book_service.create_book(request)

    created_book = fake_book_repository.books[-1]

    assert created_book["title"] == request.title
    assert created_book["author_id"] == request.author_id
    assert created_book["isbn"] == request.isbn
    assert created_book["publication_date"] ==
request.publication_date
```

```python
# test_books.py

@pytest.mark.usefixtures("db")
def test_get_all_books(books_app, client, books):
    response = client.get("v0/books")
    assert response.status_code == 200
```

```python
# test_books.py

@pytest.mark.usefixtures("db")
def test_get_all_books(books_app, client, books):
    response = client.get("v0/books")
    assert response.status_code == 200
```

```python
# test_books.py

@pytest.mark.usefixtures("db")
def test_get_all_books(books_app, client, books):
    response = client.get("v0/books")
    assert response.status_code == 200
```

```python
# test_books.py

@pytest.mark.usefixtures("db")
def test_get_all_books(books_app, client, books):
    response = client.get("v0/books")
    assert response.status_code == 200
```

```python
# test_books.py

@pytest.mark.usefixtures("db")
def test_get_all_books(books_app, client, books):
    response = client.get("v0/books")
    assert response.status_code == 200

        expected = [
        {
            "title": "Siddharta",
            "isbn": "9780553208849",
            "publication_date": "1922-01-01",
        },
        {
            "title": "Steppenwolf",
            "isbn": "9783518031599",
            "publication_date": "1924-01-01",
        },
        {
            "title": "The Glass Bead Game",
            "isbn": "9780030818516",
            "publication_date": "1943-01-01",
        },
    ]
```

```python
# test_books.py

@pytest.mark.usefixtures("db")
def test_get_all_books(books_app, client, books):
    response = client.get("v0/books")
    assert response.status_code == 200

        expected = [
        ...
    ]

        assert (
        DeepDiff(
            response.json(),
            expected,
            exclude_paths=[
                ...
            ],
        )
        == {}
    )
```

Structure

```
books-demo-app/
├── books/
│   ├── books/
│   │   ├── __init__.py
│   │   ├── asgi.py
│   │   ├── settings.py
│   │   ├── urls.py
│   │   ├── wsgi.py
│   ├── manage.py
```

```
books-demo-app/
├── src/
│    ├── authors/
│    │    ├── __init__.py
│    │    ├── dependencies.py
│    │    ├── models.py
│    │    ├── repository.py
│    │    ├── routes.py
│    │    ├── schema.py
│    │    ├── services.py
│    ├── books/
│    │    ├── ...
│    ├── migrations/
│    ├── __init__.py
│    ├── admin.py
│    ├── main.py
│    ├── models.py
│    ├── settings.py
│    ├── urls.py
│    ├── wsgi.py
├── tests/
├── .gitignore
├── Dockerfile
├── Makefile
├── README.md
├── manage.py
└── requirements.txt
```

Is this setup worth it?

Is this setup worth it?

django

SQLA

Mixing two frameworks in a single project sounds crazy but it works

Thank you for your attention!

slides & demo app

talk feedback

contact me