Combining Django ORM & FastAPI in a Single App

Mia Bajić

About me

- software engineer at Ataccama creating data products

About me

- software engineer at Ataccama creating data products
- based in Prague, Czech Republic

About me

- software engineer at Ataccama creating data products
- based in Prague, Czech Republic
- passionate about community work: co-organizer of Prague Python Pizza, EuroPython, PyCon CZ & Pyvo

My personal experience with Python frameworks

My personal experience with Python frameworks

Flask

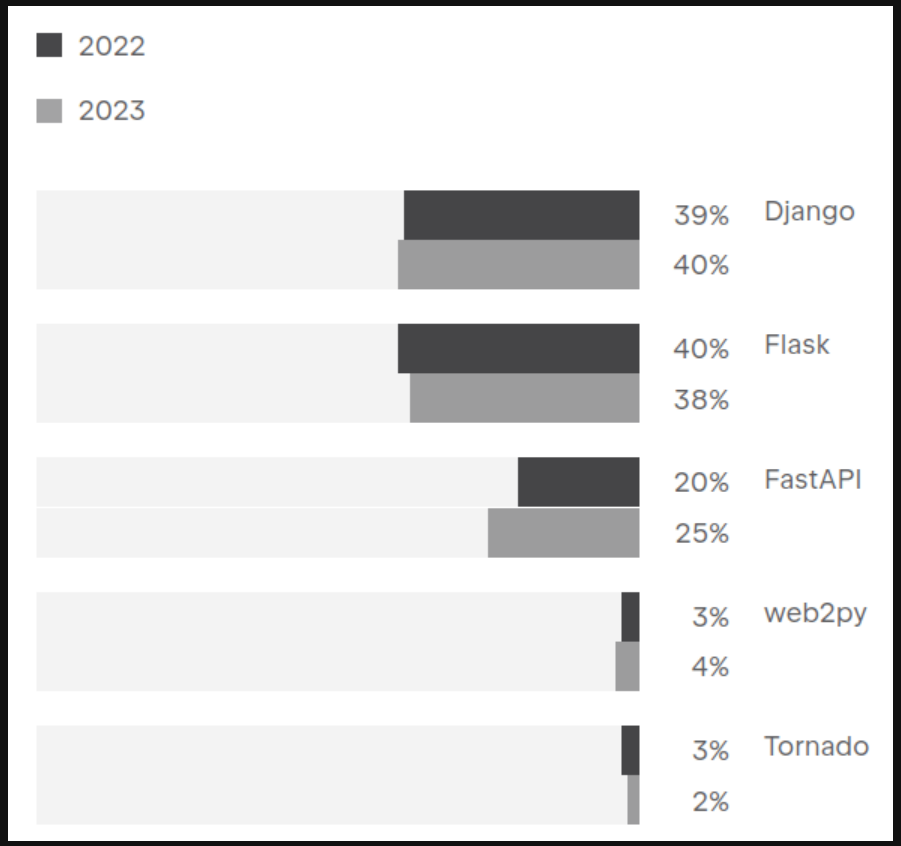
My personal experience with Python frameworks

Flask → Django

My personal experience with Python frameworks

Flask → Django → FastAPI

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 web frameworks/libraries do YOU use in addition to Python?

dispatch Public

bunnybook Public

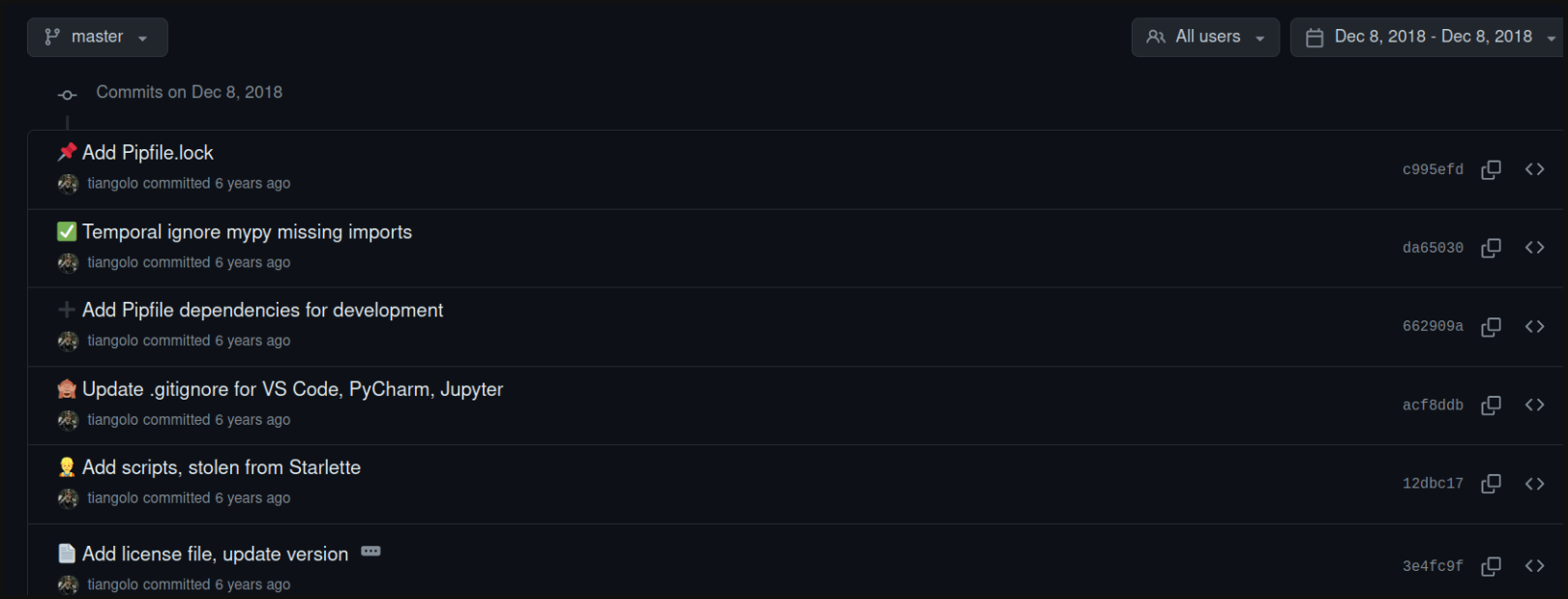
RasaGPT Public

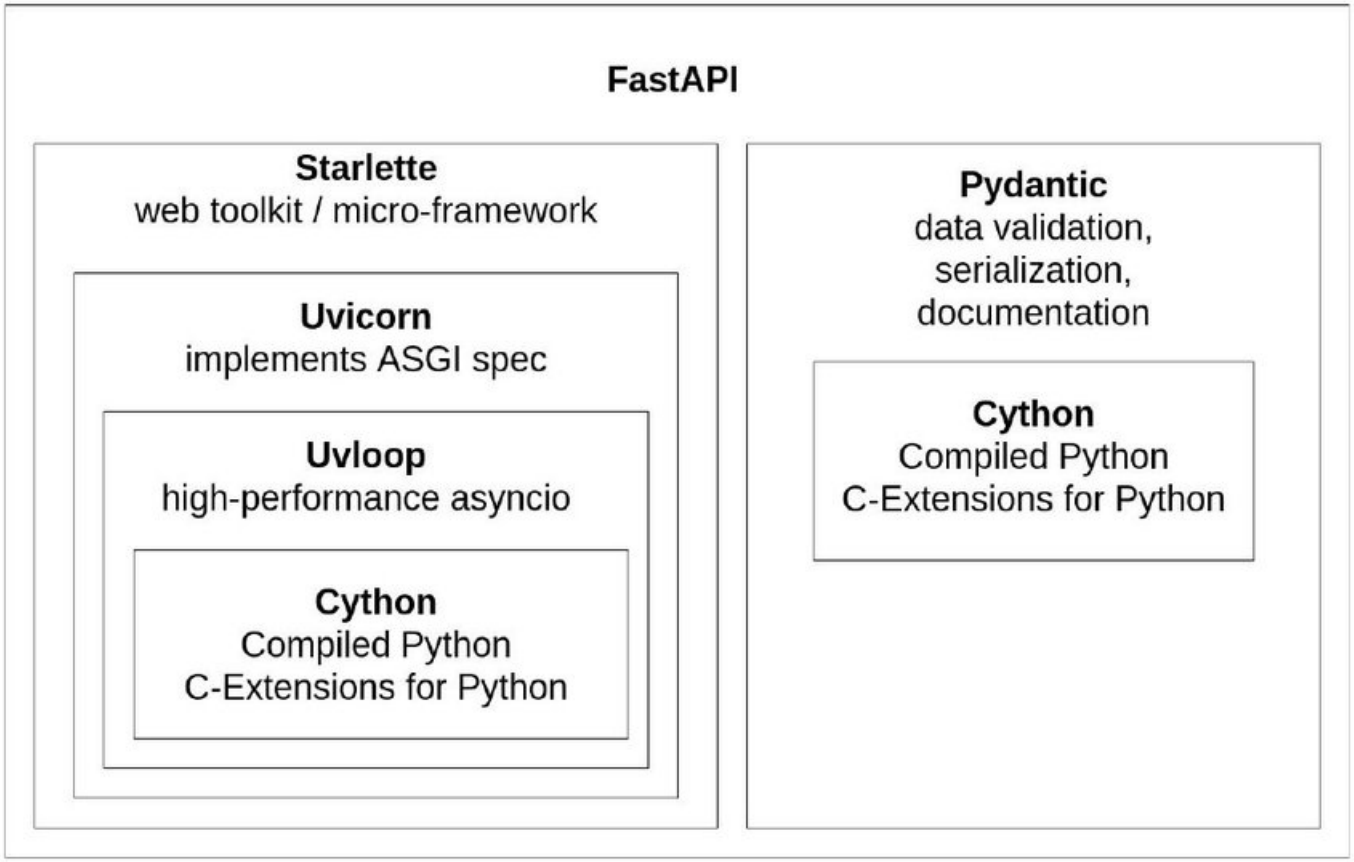
mealie Public

Open-Assistant Public

Why is Django ORM not commonly used with 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

FastAPI's high performance is primarily due to its asynchronous capabilities.

Rnk	Framework	Best performance (higher is better)	Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	fastwsgi [asgi]	69,372 100.0% (11.9)	0	Mcr	Py	asg	fas	Lin	Pg	Lin	Raw	Rea
2	blacksheep	67,629 97.5% (11.6)	0	Mcr	Py	asg	uvi	Lin	Pg	Lin	Raw	Rea
3	granian [rsgi]	59,551 85.8% (10.2)	0	Plt	Py	Non	gra	Lin	Pg	Lin	Raw	Rea
4	apidaora-core	56,304 81.2% (9.6)	0	Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
5	apidaora	51,938 74.9% (8.9)	0	Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
6	microdot-asgi-raw	50,812 73.2% (8.7)	0	Mcr	Py	Non	uvi	Lin	Pg	Lin	Raw	Rea
7	granian [asgi]	47,932 69.1% (8.2)	0	Plt	Py	Non	gra	Lin	Pg	Lin	Raw	Rea
8	fastapi	46,896 67.6% (8.0)	0	Mcr	Py	asy	Gun	Lin	Pg	Lin	Raw	Rea
9	fastapi-uvicorn	44,605 64.3% (7.6)	0	Mcr	Py	asy	uvi	Lin	Pg	Lin	Raw	Rea
10	aiohttp-pg-raw	42,020 60.6% (7.2)	0	Mcr	Py	asy	Gun	Lin	Pg	Lin	Raw	Rea
11	falcon [asgi/tortoise]	30,820 44.4% (5.3)	0	Mcr	Py	asg	uvi	Lin	Pg	Lin	Ful	Rea
12	bottle-raw	30,538 44.0% (5.2)	0	Mcr	Py	Mei	Non	Lin	My	Lin	Raw	Rea
13	falcon [pypy3]	25,903 37.3% (4.4)	0	Mcr	Py	Wsg	Gun	Lin	Pg	Lin	Ful	Rea
14	quart-uvicorn	24,730 35.6% (4.2)	0	Mcr	Py	Non	uvi	Lin	Pg	Lin	Raw	Rea
15	falcon [fastwsgi]	23,303 33.6% (4.0)	0	Mcr	Py	Wsg	fas	Lin	Pg	Lin	Ful	Rea
16	bareasgi	20,202 29.1% (3.5)	0	Mcr	Py	asy	hyp	Lin	Pg	Lin	Raw	Rea
17	falcon	19,208 27.7% (3.3)	0	Mcr	Py	Wsg	Mei	Lin	Pg	Lin	Ful	Rea
18	tornado-py3-uvloop	19,126 27.6% (3.3)	0	Plt	Py	Non	Tor	Lin	Pg	Lin	Raw	Rea
19	falcon [bjoern]	17,549 25.3% (3.0)	0	Mcr	Py	Wsg	bjo	Lin	Pg	Lin	Ful	Rea
20	fastapi-gunicorn-orm	17,038 24.6% (2.9)	0	Mcr	Py	asy	Gun	Lin	Pg	Lin	Ful	Rea
21	microdot-asgi	16,476 23.8% (2.8)	0	Mcr	Py	Non	uvi	Lin	Pg	Lin	Ful	Rea
22	morepath	16,142 23.3% (2.8)	0	Mcr	Py	Mei	Gun	Lin	Pg	Lin	Ful	Rea
23	aiohttp	15,248 22.0% (2.6)	0	Mcr	Py	asy	Gun	Lin	Pg	Lin	Ful	Rea
24	emmett	14,988 21.6% (2.6)	0	Ful	Py	Non	Non	Lin	Pg	Lin	Ful	Rea
25	pyramid	14,730 21.2% (2.5)	0	Ful	Py	Non	Mei	Lin	Pg	Lin	Ful	Rea
26	◆ django-postgresql	14,707 21.2% (2.5)	0	Ful	Py	Wsg	Mei	Lin	Pg	Lin	Ful	Rea
27	tornado-pypy2	14,083 20.3% (2.4)	0	Plt	Py	Non	Tor	Lin	Mo	Lin	Raw	Rea
28	◆ django	14,076 20.3% (2.4)	0	Ful	Py	Wsg	Mei	Lin	My	Lin	Ful	Rea
29	weppy-py3	13,923 20.1% (2.4)	0	Ful	Py	Mei	Non	Lin	Pg	Lin	Ful	Rea
30	microdot-wsgi	12,845 18.5% (2.2)	0	Mcr	Py	Non	Gun	Lin	Pg	Lin	Ful	Rea
31	bottle-pypy2	11,785 17.0% (2.0)	0	Mcr	Py	Tor	Non	Lin	My	Lin	Ful	Rea
32	fastapi-hypercorn	10,831 15.6% (1.9)	0	Mcr	Py	asy	hyp	Lin	Pg	Lin	Raw	Rea
33	tornado-py3	10,771 15.5% (1.8)	0	Plt	Py	Non	Tor	Lin	Mo	Lin	Raw	Rea
34	turbogears	10,233 14.8% (1.7)	0	Mcr	Py	Non	Mei	Lin	My	Lin	Ful	Rea
35	bottle	9,955 14.4% (1.7)	0	Mcr	Py	Mei	Non	Lin	My	Lin	Ful	Rea

Django is a very popular, batteries-included framework, with its first release in 2005.

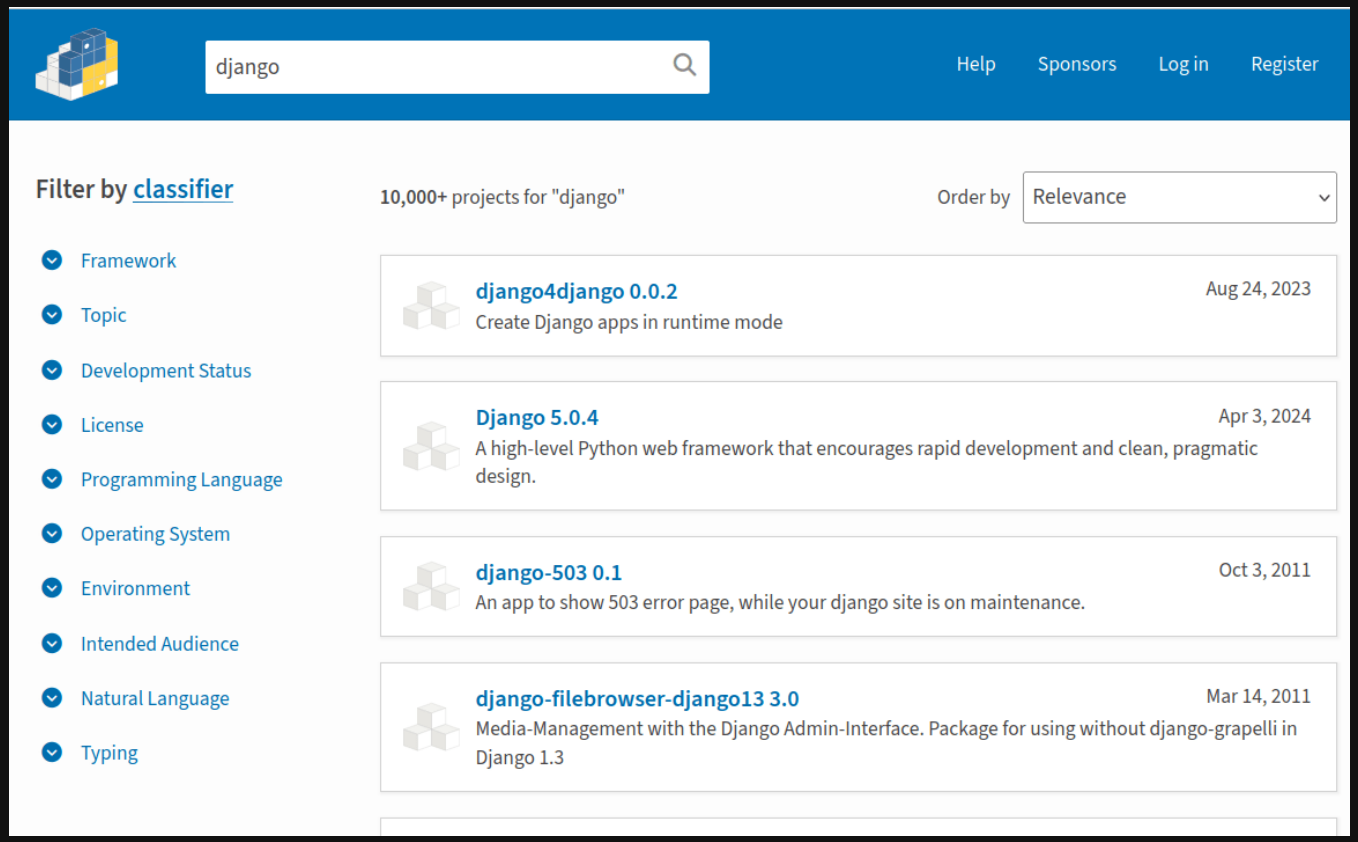
Django is opinionated.

Django ORM is primarily synchronous with some asynchronous features being added from version 4.2.

Why should one consider using Django ORM alongside FastAPI?

Django ORM is easier to learn.

Django has a rich ecosystem.



Django has a rich ecosystem.

Apps

Small components used to build projects. An app is anything that is installed by placing in settings.INSTALLED_APPS.

Show Apps (4,146)

Frameworks

Large efforts that combine many python modules or apps. Examples include Django, django-cms, and Mezzanine. Most CMSes fall into this category, and so do storefronts.

Show Frameworks (189)

Projects

This is for individual projects such as Django Packages, DjangoProject.com, and others.

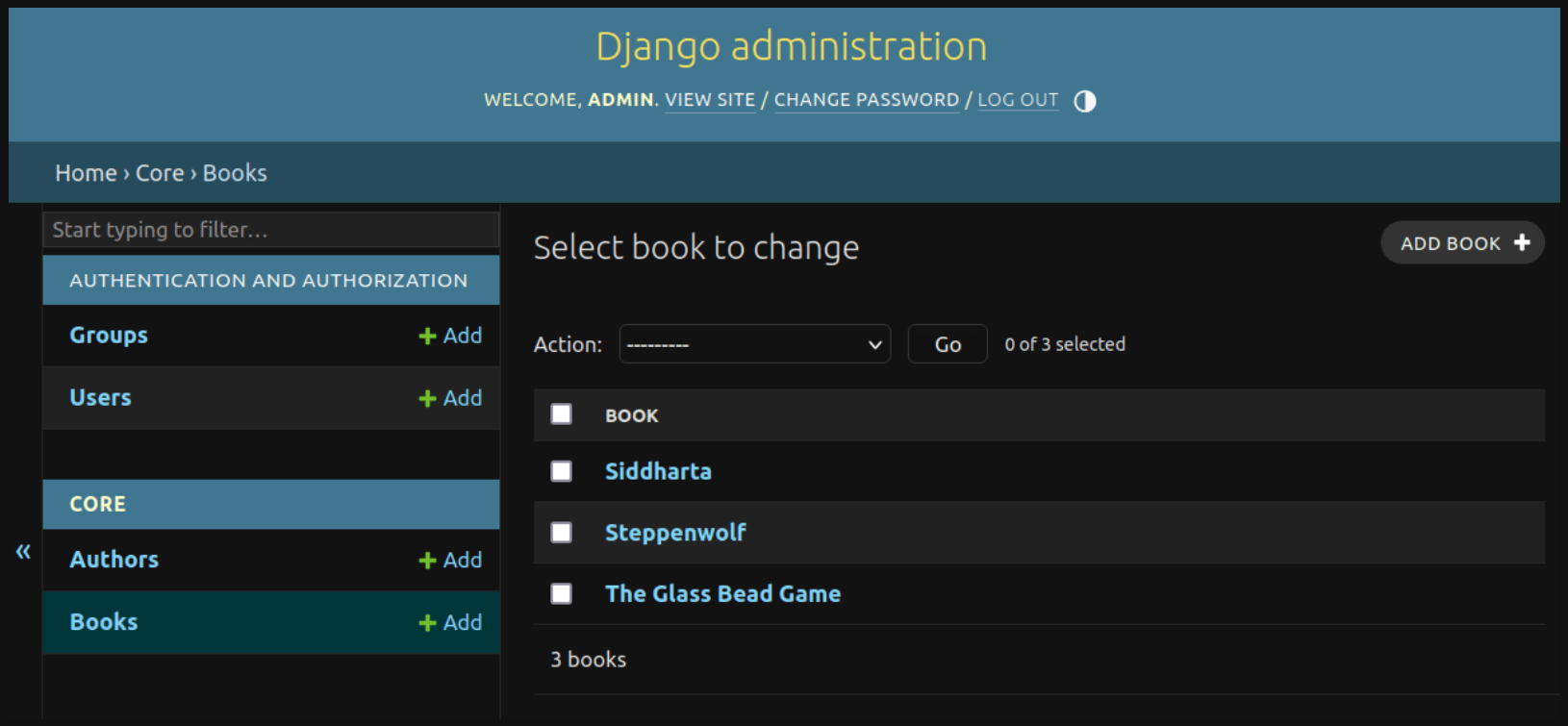
Show Projects (230)

Other

Other are not installed by settings.INSTALLED_APPS, are not frameworks or sites but still help Django in some way.

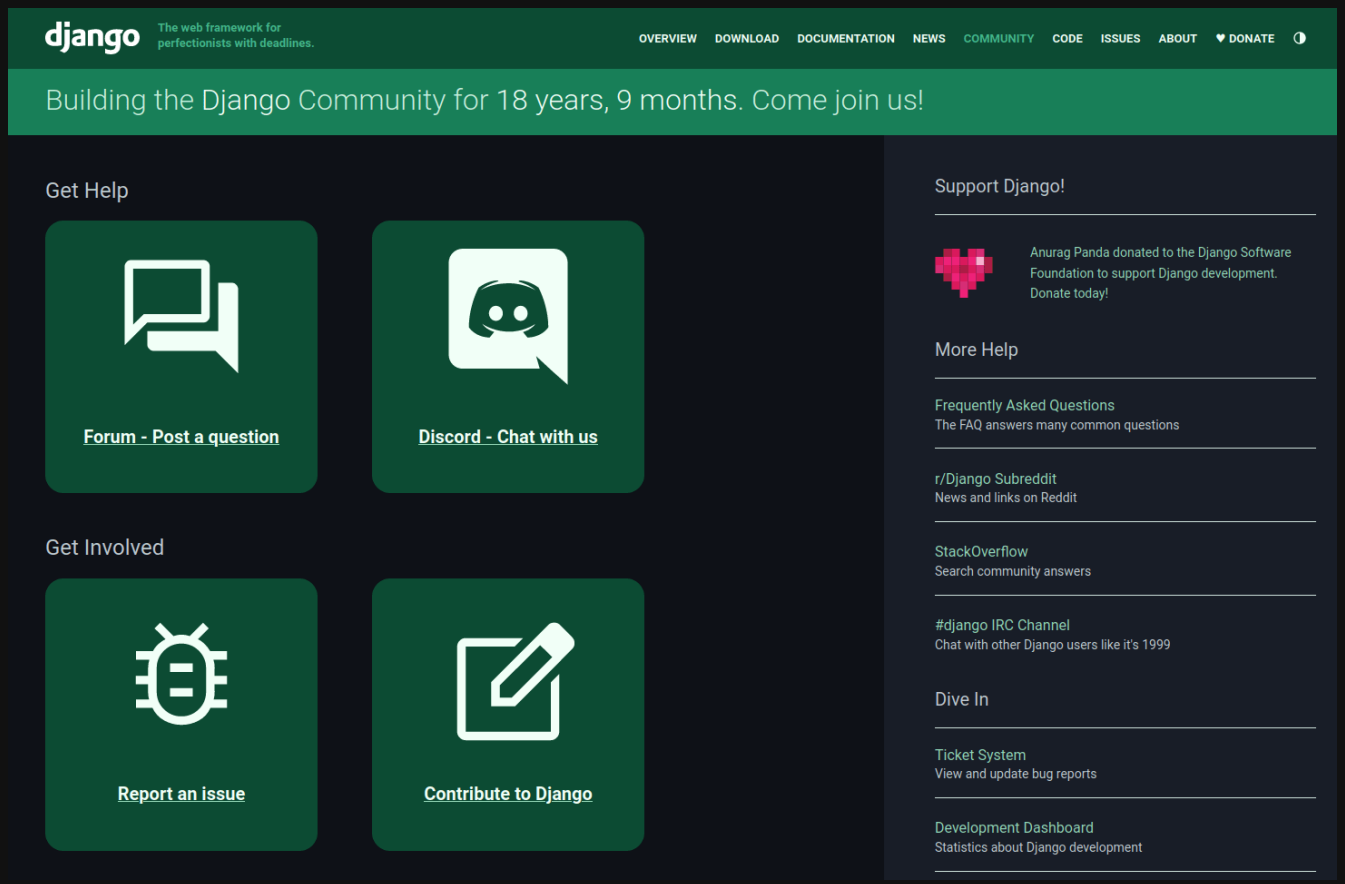
Show Other (743)

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



Django is supported by a big and active community.

Django is supported by a big and active community.



FastAPI is performant.

FastAPI offers documentation of endpoints.

127.0.0.1:8000/docs#/

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

127.0.0.1:8000/redoc 90%

Search...

Books

Authors

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

Copy Expand all Collapse all


```
[
 - {
      "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"}
```

```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"}
```

```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.

Why should you not use Django ORM with FastAPI?

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

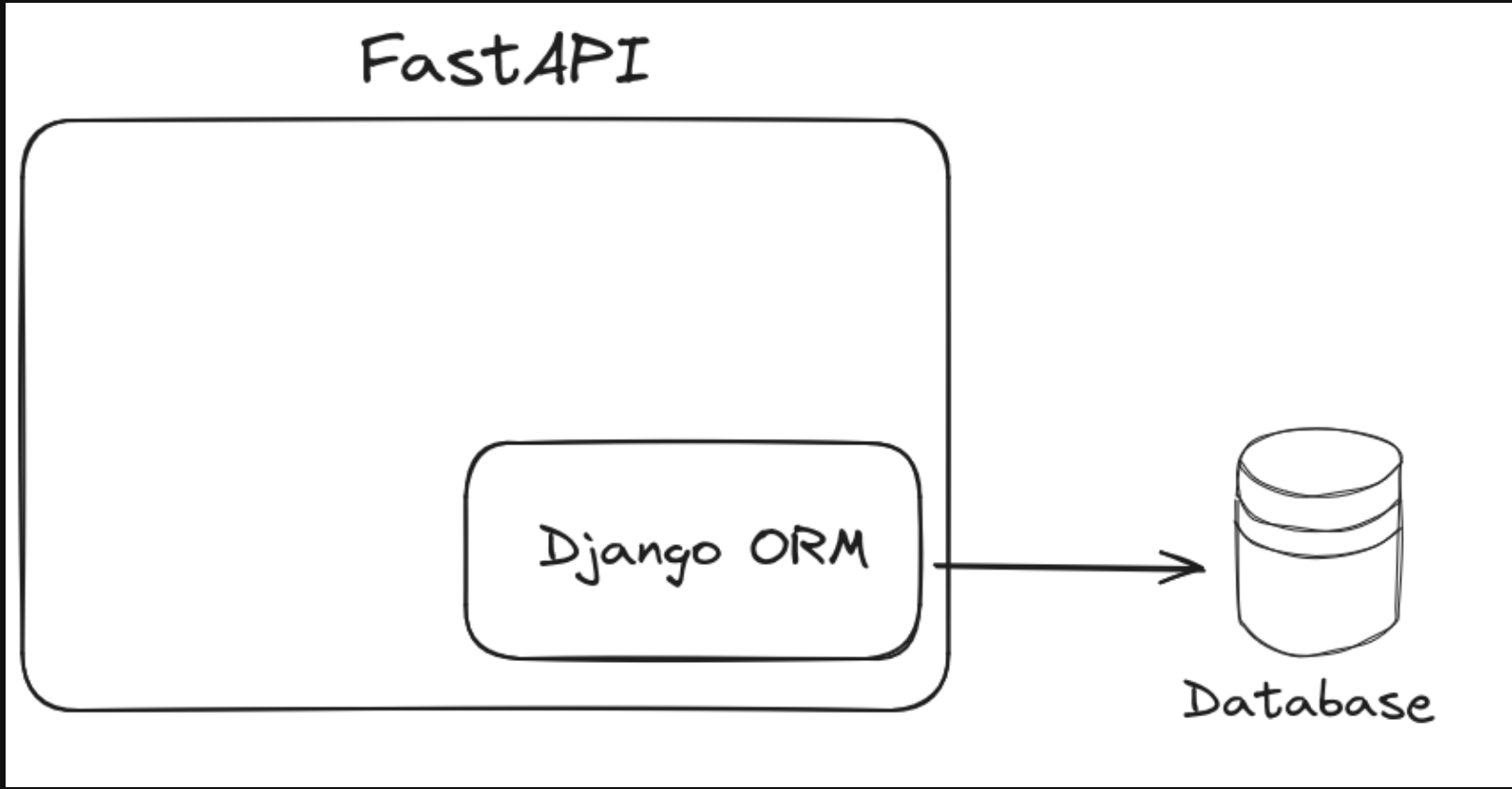
Managing synchronous and asynchronous code adds a layer of complexity.

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.

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,
    )
```

```
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"]
```

```
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"]
```

```
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", "80
```

```
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

Parameters Cancel

No parameters

Execute Clear

Responses

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
  }
]
```

Download

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]
```

```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]

    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?

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

Thank you for your attention!

slides

contact me