Writing async microservices in Python

Iacopo Spalletti – CTO @ Nephila

{codemotion}
Online Tech Conference
- Italian edition -

9-10-11 Novembre, 2021

nephila

Hello, I am Iacopo

Founder and CTO @NephilaIT

Djangonaut and open source developer

Iacopo Spalletti - @yakkys

nephila

intro

async programming

Iacopo Spalletti - @yakkys

nephila

async programming

blocking IO

cooperative multitasking

Iacopo Spalletti - @yakkys

nephila

async programming

concurrency

Q: When do we want?

A: More concurrency!

A: Now!

Q: What do we want?

nephila®

concurrency

more performance?

nephila

concurrency

more performance?

better resource usage!

nephila

concurrency

example

nephila

concurrency

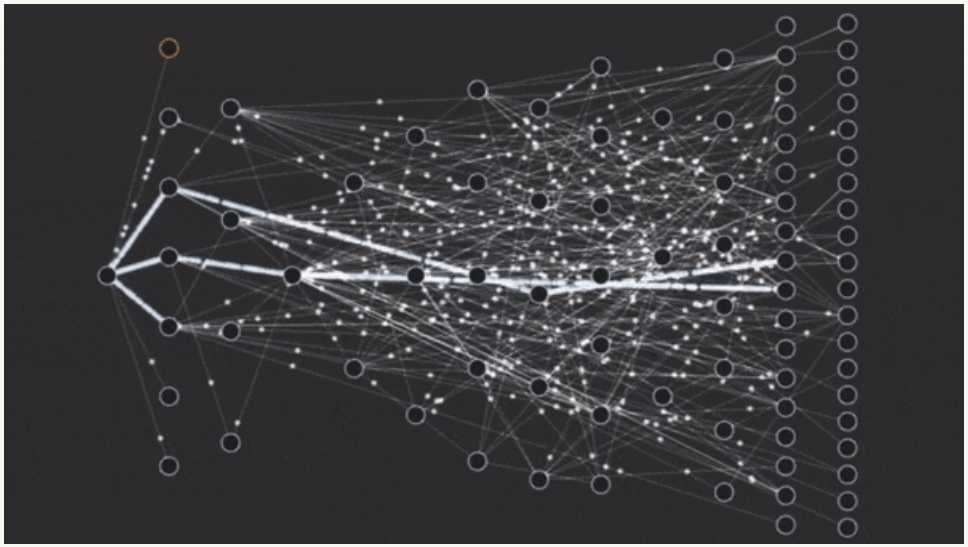
example

```
async def serve_client():
    take_order()
    await prepare_dish()
    serve_plate()
```

nephila

concurrency

microservice architecture



nephila

async programming

event-driven programming

no request / response

pub/sub

message queues

...

nephila

async in python

twisted

tornado

asyncio

trio

...

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    prepare_dish()
    serve_plate()

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(serve_client())
```

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    prepare_dish()
    serve_plate()

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(serve_client())
```

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    await prepare_dish()
    serve_plate()

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(serve_client())
```

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    await prepare_dish()
    serve_plate()

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(serve_client())
```

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    await prepare_dish()
    serve_plate()

if __name__ == "__main__":
    asyncio.run(serve_client())
```

nephila

FastAPI: A web async framework

- type hints based
- opinionated
- native OpenAPI / JSON Schema
- some batteries included

nephila

FastAPI: A web async framework

built on top of proven components

Starlette

Pydantic

Iacopo Spalletti - @yakkys

nephila

why not django?

nephila

why not django?

pros

async views in 3.1+

good ol' boring Django code

nephila

why not django?

cons

higher footprint

unused code

no Django REST framework async (yet)

less new things to learn

nephila

fastapi

the basics

```
app = FastAPI()

@app.get("/items/{item_id}", response_model=Item)
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

the basics

```
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

concepts

ASGI ?

nephila

concepts

ASGI ?

Standard python protocol for async applications

nephila

concepts

ASGI ?

Standard python protocol for async applications

HTTP → ASGI Server → Async application

nephila

concepts

ASGI ?

Standard python protocol for async applications

HTTP → ASGI Server → Async application

Everything is an app

- entrypoints
- routers
- middlewares

nephila

fastapi

routing / path operations

```
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

routing / path operations

```
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

business logic

```
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

business logic

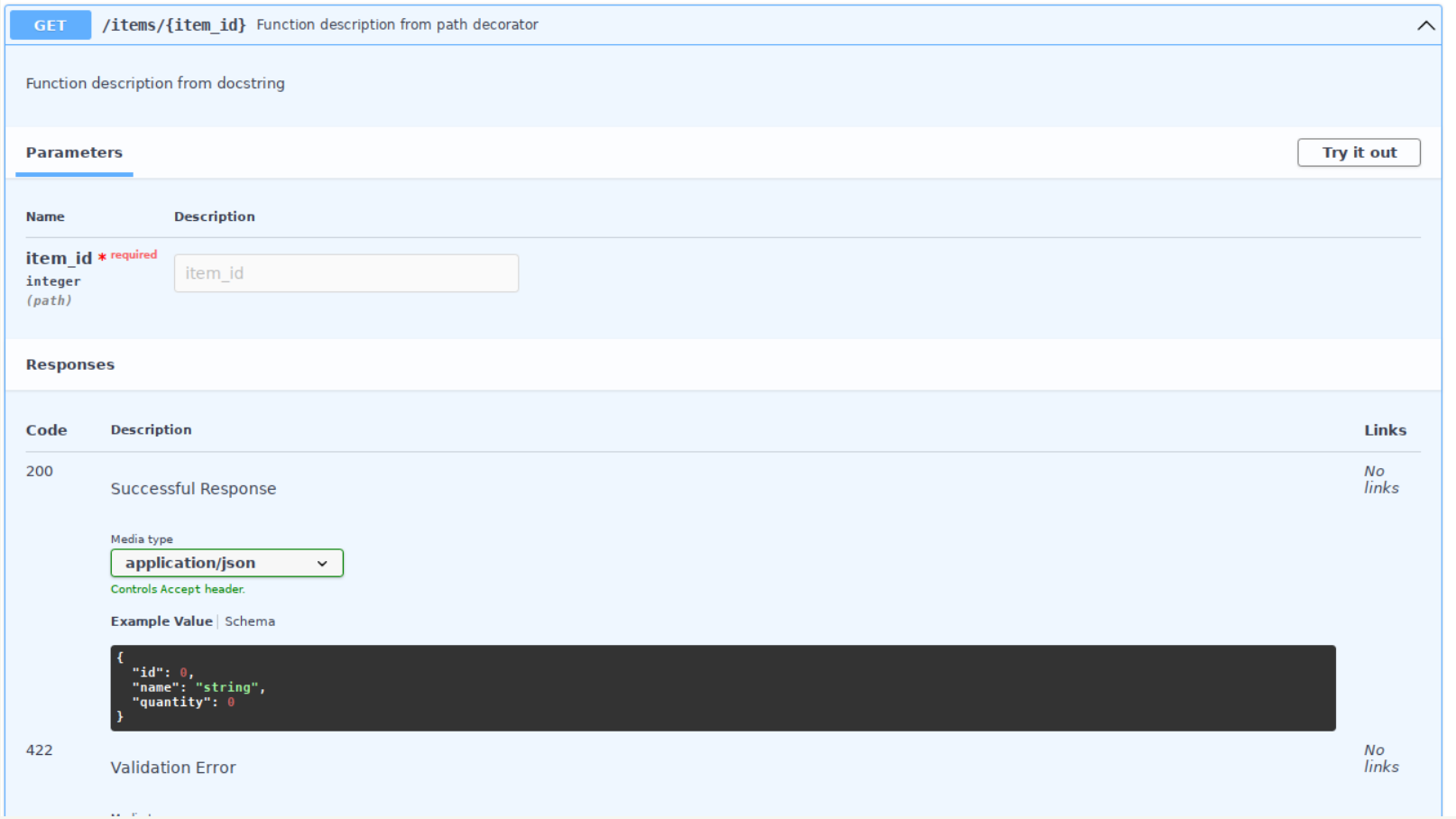
```
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

documentation



nephila

sample application

https://github.com/yakky/microservice-talk/

nephila

sample application

ASGI server

```
uvicorn book_search.main:app
```

nephila

sample application

ASGI server

```
book_search/__main__.py

def main():
    from book_search.app_settings import get_settings
    settings = get_settings()

    uvicorn.run(
        "book_search.main:app",
        host=settings.API_IP,
        port=settings.API_PORT,
        ...
    )

main()
```

```
python -mbook_search
```

nephila

sample application

Entrypoint

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

sample application

Settings

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

sample application

Settings

```
class Settings(BaseSettings):
    PROJECT_NAME: str = "Book Search"
    API_V1_STR: str = "/api/v1"
    ES_HOST: str = "127.0.0.1:9200"
    BACKEND_CORS_ORIGINS: str = ".*"
    API_IP = "0.0.0.0"
    API_PORT = 5000
```

```
ES_HOST=es:9200 PROJECT_NAME=MyAPI python -mbook_search
```

nephila

sample application

Settings

```
class Settings(BaseSettings):
    PROJECT_NAME: str = "Book Search"
    API_V1_STR: str = "/api/v1"
    ES_HOST: str = "127.0.0.1:9200"
    BACKEND_CORS_ORIGINS: str = ".*"

    class Config:
        env_file = ".env"
```

nephila

sample application

Entrypoint

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

sample application

Middleware

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

sample application

Middleware

```python
@app.middleware("http")
async def add_process_time_header(request: Request, call_next):
    start_time = time.time()
    response = await call_next(request)
    process_time = time.time() - start_time
    response.headers["X-Process-Time"] = str(process_time)
    return response
```

nephila

sample application

Routing

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

Search function

Routing

```python
search_router = APIRouter()

@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Routing

```
from .. import app

@app.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    ...
```

nephila

Search function

Dependency injection

```python
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Parameters

```python
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Dependency injection

```python
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Business logic

```python
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Response

```
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Response

```
...

class Book(BaseModel):
    book_id: int
    title: str
    isbn13: str
    authors_list: List[Author] = Field(..., alias='authors')
    tags: List[Tag]
    original_publication_year: int


class BookList(BaseModel):
    results: List[Book]
    count: int
```

nephila

Search function

Response

```
...

class Book(BaseModel):
    book_id: int
    title: str
    isbn13: str
    authors_list: List[Author] = Field(..., alias='authors')
    tags: List[Tag]
    original_publication_year: int


class BookList(BaseModel):
    results: List[Book]
    count: int
```

nephila

Pydantic

Caveat

nephila

Pydantic

Caveat

Data validation **not** serialization

Manipulating data is a bit tricky

Return data ready for serialization

nephila

sample application

Testing

```
@pytest.mark.asyncio
async def test_search_basic(load_books):
    async with httpx.AsyncClient(app=app, base_url="http://test") as client:
        url = app.url_path_for("search")
        params = urlencode({"q": "Susan Collins"})
        response = await client.get(f"{url}?{params}")
        assert response.status_code == 200
```

nephila

sample application

Testing

```
from fastapi.testclient import TestClient
from book_search.main import app

def test_search_basic_sync(load_books):
    client = TestClient(app)
    url = app.url_path_for("search")
    params = urlencode({"q": "Susan Collins"})
    response = client.get(f"{url}?{params}")
    assert response.status_code == 200
```

nephila

sample application

Async test

```python
@pytest.mark.asyncio
async def test_search_basic(load_books):
    async with httpx.AsyncClient(app=app, base_url="http://test") as client:
        url = app.url_path_for("search")
        params = urlencode({"q": "Susan Collins"})
        response = await client.get(f"{url}?{params}")
        assert response.status_code == 200
```

nephila

sample application

Async test

```python
@pytest.mark.asyncio
async def test_search_basic(load_books):
    async with httpx.AsyncClient(app=app, base_url="http://test") as client:
        url = app.url_path_for("search")
        params = urlencode({"q": "Susan Collins"})
        response = await client.get(f"{url}?{params}")
        assert response.status_code == 200
```

nephila

Outro

async programming can be tricky

great benefits for some workloads

frameworks like FastAPI makes life easier

nephila

Grazie

Iacopo Spalletti - @yakkys

nephila

Want more?

Building real time applications with Django and Channels

https://bit.ly/3klvBfB

Iacopo Spalletti - @yakkys

nephila

Questions?

nephila

Iacopo Spalletti

Founder and CTO @NephilaIT

Djangonaut and open source developer

@yakks

@yakky@mastodon.social

https://github.com/yakky

https://speakerdeck.com/yakky

Writing async microservices in Python

Iacopo Spalletti – CTO @ Nephila

{codemotion}
Online Tech Conference
- Italian edition -

9-10-11 Novembre, 2021

nephila

Hello, I am Iacopo

Founder and CTO @NephilaIT

Djangonaut and open source developer

Iacopo Spalletti - @yakkys

nephila

intro

async programming

Iacopo Spalletti - @yakkys

nephila

async programming

blocking IO

cooperative multitasking

Iacopo Spalletti - @yakkys

nephila

async programming

concurrency

Q: When do we want?

A: More concurrency!

A: Now!

Q: What do we want?

nephila®

concurrency

more performance?

nephila®

concurrency

more performance?

better resource usage!

nephila

concurrency

example

nephila

concurrency

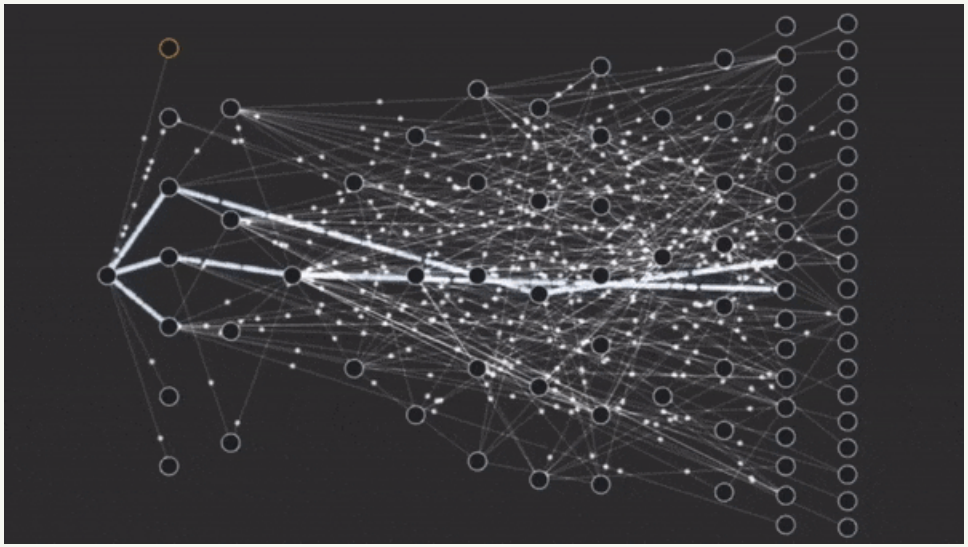
example

```
async def serve_client():
    take_order()
    await prepare_dish()
    serve_plate()
```

nephila

concurrency

microservice architecture



nephila

async programming

event-driven programming

no request / response

pub/sub

message queues

...

nephila

async in python

twisted

tornado

asyncio

trio

...

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    prepare_dish()
    serve_plate()

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(serve_client())
```

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    prepare_dish()
    serve_plate()

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(serve_client())
```

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    await prepare_dish()
    serve_plate()

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(serve_client())
```

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    await prepare_dish()
    serve_plate()

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(serve_client())
```

nephila

asyncio

```
async def prepare_dish():
    ...

async def serve_client():
    take_order()
    await prepare_dish()
    serve_plate()

if __name__ == "__main__":
    asyncio.run(serve_client())
```

nephila

FastAPI: A web async framework

- type hints based
- opinionated
- native OpenAPI / JSON Schema
- some batteries included

nephila

FastAPI: A web async framework

built on top of proven components

Starlette

Pydantic

Iacopo Spalletti - @yakkys

nephila

why not django?

nephila

why not django?

pros

async views in 3.1+

good ol' boring Django code

nephila

why not django?

cons

higher footprint

unused code

no Django REST framework async (yet)

less new things to learn

nephila

fastapi

the basics

```
app = FastAPI()

@app.get("/items/{item_id}", response_model=Item)
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

the basics

```
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

concepts

ASGI ?

nephila

concepts

ASGI ?

Standard python protocol for async applications

nephila

concepts

ASGI ?

Standard python protocol for async applications

HTTP → ASGI Server → Async application

nephila

concepts

ASGI ?

Standard python protocol for async applications

HTTP → ASGI Server → Async application

Everything is an app

- entrypoints
- routers
- middlewares

nephila

fastapi

routing / path operations

```
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

routing / path operations

```python
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

business logic

```
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

business logic

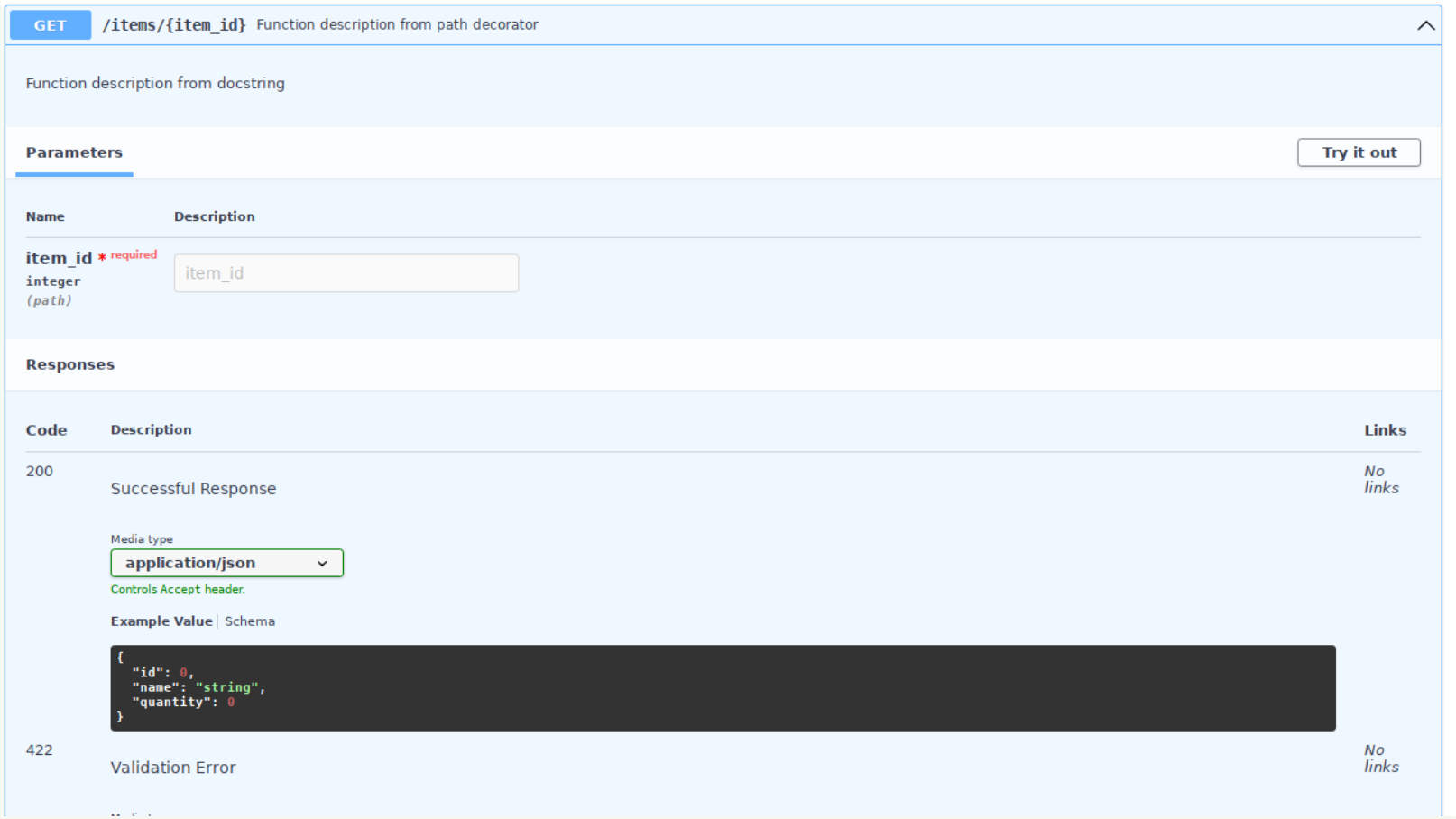
```
app = FastAPI()

@app.get("/items/{item_id}")
async def get_item(item_id: int, q: Optional[str]=None):
    item = await get_item_obj(item_id)
    return {
        "name": item.name,
        "id": item.id,
        "quantity": item.quantity
}
```

nephila

fastapi

documentation



nephila

sample application

https://github.com/yakky/microservice-talk/

nephila

sample application

ASGI server

```
uvicorn book_search.main:app
```

nephila

sample application

ASGI server

book_search/__main__.py

```
def main():
    from book_search.app_settings import get_settings
    settings = get_settings()

    uvicorn.run(
        "book_search.main:app",
        host=settings.API_IP,
        port=settings.API_PORT,
        ...
    )

main()
```

```
python -mbook_search
```

nephila

sample application

Entrypoint

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

sample application

Settings

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

sample application

Settings

```
class Settings(BaseSettings):
    PROJECT_NAME: str = "Book Search"
    API_V1_STR: str = "/api/v1"
    ES_HOST: str = "127.0.0.1:9200"
    BACKEND_CORS_ORIGINS: str = ".*"
    API_IP = "0.0.0.0"
    API_PORT = 5000
```

```
ES_HOST=es:9200 PROJECT_NAME=MyAPI python -mbook_search
```

nephila

sample application

Settings

```
class Settings(BaseSettings):
    PROJECT_NAME: str = "Book Search"
    API_V1_STR: str = "/api/v1"
    ES_HOST: str = "127.0.0.1:9200"
    BACKEND_CORS_ORIGINS: str = ".*"

    class Config:
        env_file = ".env"
```

nephila

sample application

Entrypoint

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

sample application

Middleware

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

sample application

Middleware

```python
@app.middleware("http")
async def add_process_time_header(request: Request, call_next):
    start_time = time.time()
    response = await call_next(request)
    process_time = time.time() - start_time
    response.headers["X-Process-Time"] = str(process_time)
    return response
```

nephila

sample application

Routing

```
settings = Settings()
app = FastAPI(title=settings.PROJECT_NAME)

app.add_middleware(
    CORSMiddleware,
    allow_origin_regex=settings.BACKEND_CORS_ORIGINS,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

app.include_router(search_router, prefix=settings.API_V1_STR)
```

nephila

Search function

Routing

```python
search_router = APIRouter()

@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Routing

```
from .. import app

@app.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    ...
```

nephila

Search function

Dependency injection

```python
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Parameters

```python
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Dependency injection

```python
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Business logic

```python
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Response

```
search_router = APIRouter()


@search_router.get("/search/", response_model=models.BookListResponse, name="search")
async def search(
    request: Request,
    settings: Settings = Depends(get_settings),
    q: str = Query(None, description="Free text query string."),
    tags: List[str] = Query([], description="List of tags slug."),
    year: int = Query(None, description="Filter by year."),
    size: int = Query(20, description="Results per page."),
):
    """
    Search ES data according to the provided filters.
    """
    results, total = await search_es(settings, q, year, tags, size)
    return {"results": results, "count": total}
```

nephila

Search function

Response

```
...

class Book(BaseModel):
    book_id: int
    title: str
    isbn13: str
    authors_list: List[Author] = Field(..., alias='authors')
    tags: List[Tag]
    original_publication_year: int


class BookList(BaseModel):
    results: List[Book]
    count: int
```

nephila

Search function

Response

```
...

class Book(BaseModel):
    book_id: int
    title: str
    isbn13: str
    authors_list: List[Author] = Field(..., alias='authors')
    tags: List[Tag]
    original_publication_year: int


class BookList(BaseModel):
    results: List[Book]
    count: int
```

nephila

Pydantic

Caveat

nephila

Pydantic

Caveat

Data validation **not** serialization

Manipulating data is a bit tricky

Return data ready for serialization

nephila

sample application

Testing

```
@pytest.mark.asyncio
async def test_search_basic(load_books):
    async with httpx.AsyncClient(app=app, base_url="http://test") as client:
        url = app.url_path_for("search")
        params = urlencode({"q": "Susan Collins"})
        response = await client.get(f"{url}?{params}")
        assert response.status_code == 200
```

nephila

sample application

Testing

```
from fastapi.testclient import TestClient
from book_search.main import app

def test_search_basic_sync(load_books):
    client = TestClient(app)
    url = app.url_path_for("search")
    params = urlencode({"q": "Susan Collins"})
    response = client.get(f"{url}?{params}")
    assert response.status_code == 200
```

nephila

sample application

Async test

```python
@pytest.mark.asyncio
async def test_search_basic(load_books):
    async with httpx.AsyncClient(app=app, base_url="http://test") as client:
        url = app.url_path_for("search")
        params = urlencode({"q": "Susan Collins"})
        response = await client.get(f"{url}?{params}")
        assert response.status_code == 200
```

nephila

sample application

Async test

```python
@pytest.mark.asyncio
async def test_search_basic(load_books):
    async with httpx.AsyncClient(app=app, base_url="http://test") as client:
        url = app.url_path_for("search")
        params = urlencode({"q": "Susan Collins"})
        response = await client.get(f"{url}?{params}")
        assert response.status_code == 200
```

nephila

Outro

async programming can be tricky

great benefits for some workloads

frameworks like FastAPI makes life easier

nephila

Grazie

Iacopo Spalletti - @yakkys

nephila

Want more?

Building real time applications with Django and Channels

https://bit.ly/3klvBfB

Iacopo Spalletti - @yakkys

nephila

Questions?

nephila

Iacopo Spalletti

Founder and CTO @NephilaIT

Djangonaut and open source developer

@yakks

@yakky@mastodon.social

https://github.com/yakky

https://speakerdeck.com/yakky