BUILDING REST API ENDPOINT WITH FASTAPI

Chris Choy
PYCON HK 2020
November

Code Available at Github

https://github.com/christlc/pyconhk2020fastapi

Today's Content

REST API

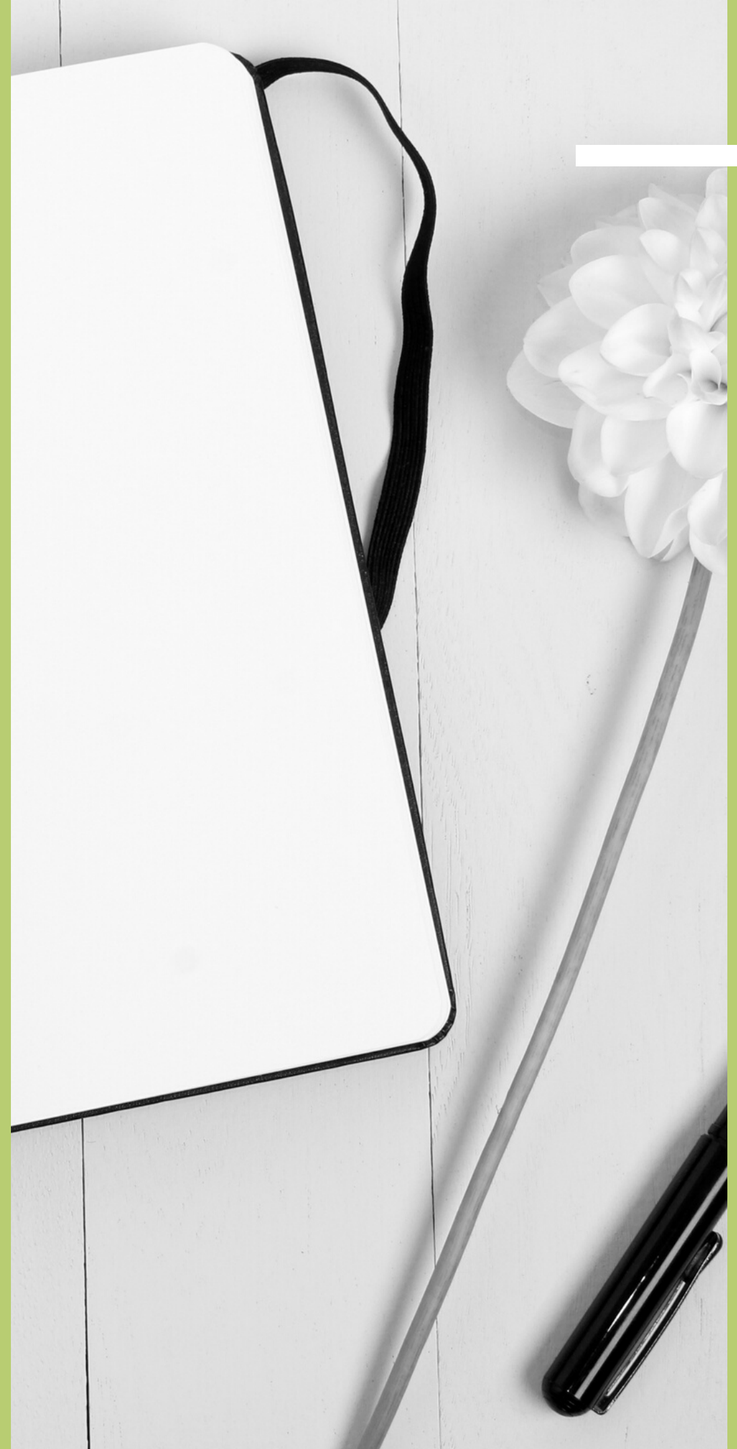
Quick introduction of REST API

DEMO

Getting Parameter: path, query, body
Data Validation

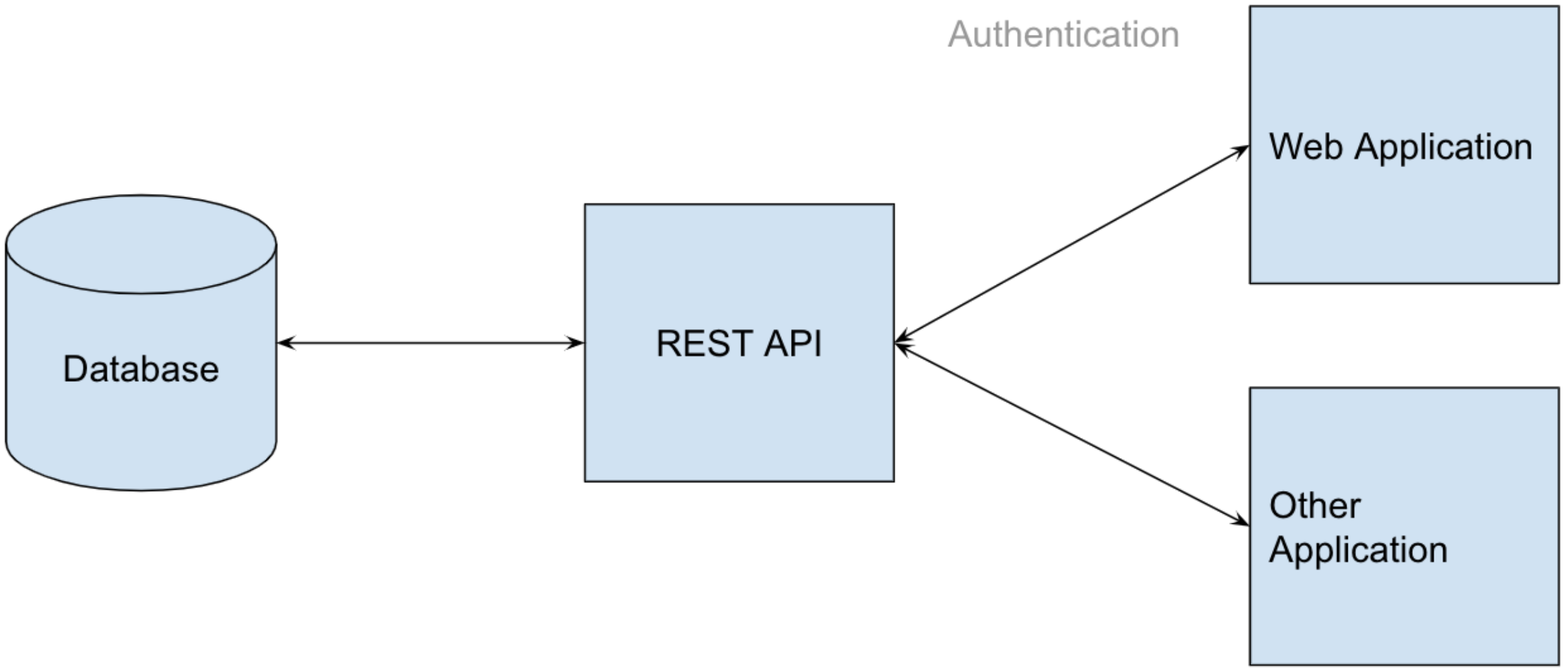
COMPARISON AND TIPS

Comparison with Flask



What is a REST API?

Typical Architecture



FastAPI with (Some) Batteries Included

PARAMETER VALIDATION

Using type hint to automatically validate input format and provide useful error messages

API DOCUMENTATION

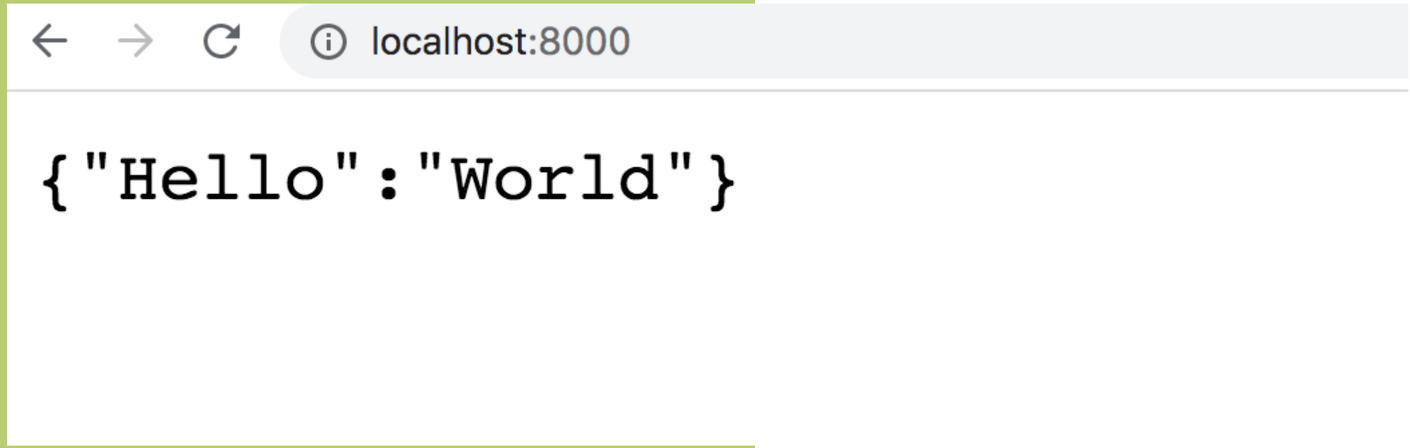
In accessible Swagger doc format with authentication support in-built

MULTIPLE FILES ROUTERS

Much easier than in vanilla Flask

Hello World

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
def read_root():
    return {"Hello": "World"}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run("main:app", host="0.0.0.0", port=8000, log_level="info", workers=1)
```



First API

GET at http://127.0.0.1:8000

Lets do a simple endpoint

GET /message/

Return all message in dictionary

POST /MESSAGE/{ID}

Write message to in memory dictionary at key ID

GET /MESSAGE/{ID}

Return message of key ID

See it in action

API DOC

127.0.0.1:8000/docs

Implementation

```python
local_dict = {}


@app.get("/messages/")
def list_items():
    return local_dict


@app.get("/messages/{item_id}")
def read_item(item_id: int):
    return {'item_id': item_id, 'message': local_dict.get(item_id)}


@app.post("/messages/{item_id}")
def post_item(item_id: int, message: str):
    local_dict[item_id] = message
```

Implemented in a few lines!

local_dict is for simple demo purpose

Type validation

localhost:8000/messages/hello

{"detail":[{"loc":["path","item_id"],"msg":"value is not a valid integer","type":"type_error.integer"}]}

Useful Error Message when type is incorrect

http://localhost:8000/messages/hello

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

# Create a test client
client = TestClient(app)


def test_root():
    result = client.get('/')
    print(result.json())
    assert result.status_code==200
    result_dict = result.json()
    assert result_dict['Hello'] == 'World'
```

Testing

Testing is simple too

Requests based testing

Body Parameter

```python
from typing import Optional
from pydantic import BaseModel


class Message(BaseModel):
    message: str
    meta: Optional[dict] = None

```

```python
from schema import Message

@app.post("/messages/body/{item_id}")
def post_item_json(item_id: int, message: Message):
    local_dict[item_id] = message
```

Define a schema

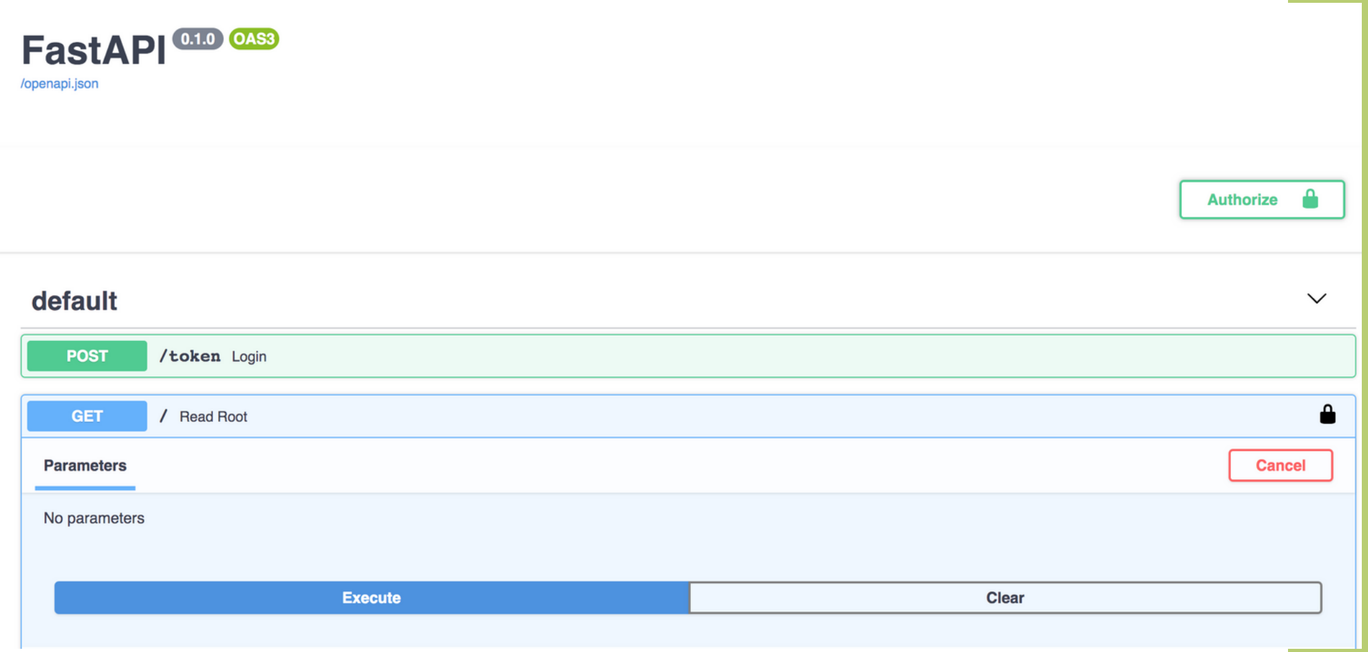
Auto mapping of body to object

```python
from fastapi import FastAPI
from fastapi import Depends, HTTPException
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm


app = FastAPI()

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")


@app.post("/token")
async def login(form_data: OAuth2PasswordRequestForm = Depends()):
    if form_data.username != 'root' or form_data.password != 'password': # should replace with hashed password
        raise HTTPException(status_code=400, detail="Incorrect username or password")
    return {"access_token": 'mytoken', "token_type": "bearer"}


@app.get("/")
def read_root(token: str = Depends(oauth2_scheme)):
    assert token == 'mytoken'
    return {"Hello": "World"}
```

Authentication

A few lines to add basic authentication

Can easily integrate with any Oauth2 authentication flow



Authentication

Authentication in test interface

Usable authenticated test interface

Comparison

Key Benefits Over Vanilla Flask

01

PARAMETER VALIDATION

It's possible to use marshmallow in Flask, but it's very time consuming.

02

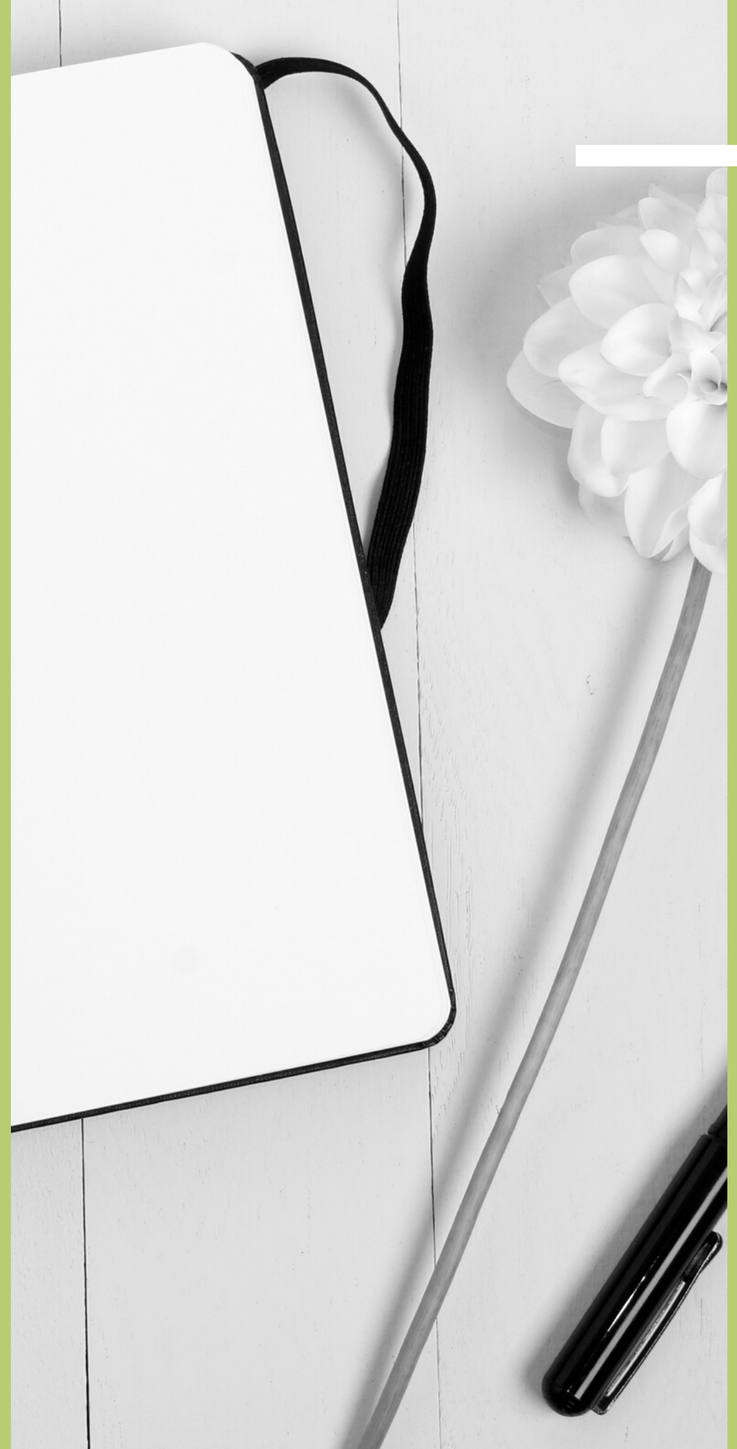
AUTO DOCUMENTATIONS GENERATION

It's also possible to generate docs for Flask API, but it needs extensive annotations.

03

MULTI-FILE ROUTER

No need to use another library.



Tips and Tricks

JSON Encoder

Python default encoder generate non-standard JSON

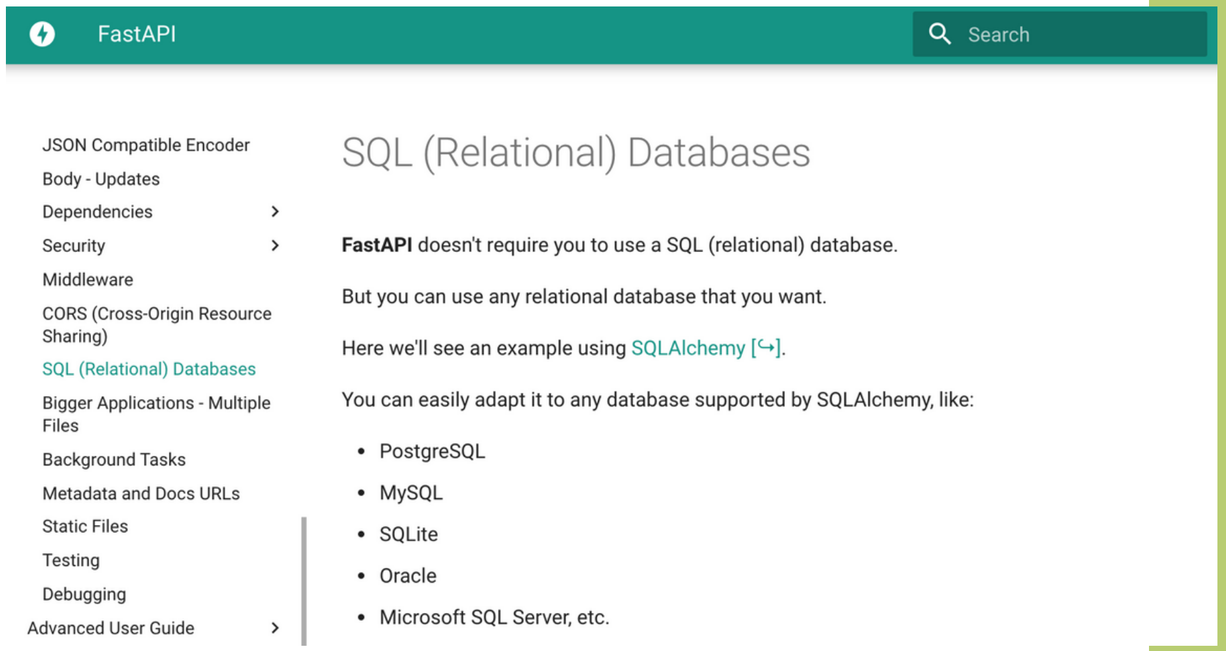
Swapping implementation would solve most of these issues.

Async Support

```python
from fastapi import FastAPI
import time
import asyncio

app = FastAPI()


@app.get("/benchmark1/")
async def benchmark():
    await asyncio.sleep(1)
    return {"message": "ok"}


@app.get("/benchmark2/")
def benchmark2():
    time.sleep(1)
    return {"message": "ok"}
```

Not as much benefits

Some benefits under high concurrency and long processing time. Not much differences under normal use case.



Excellent Documentations and Worked Examples

With good production worthy advice

https://fastapi.tiangolo.com/

Questions?

Code Available at Github

https://github.com/christlc/pyconhk2020fastapi