Es como flask, prestamente lo puedes estructurar como quieras

# Requeriments

fastapi

uvicorn[standard]

sqlalchemy

python-decouple

cryptography

python-jose[cryptography]

passlib[bcrypt]

bcrypt==3.2.0

pydantic

python-multipart

httpx

pytest

# main.py

Pero es común que al menos exista un archivo main.py

from fastapi import FastAPI

app=FastAPI()

En este archivo también se puede crear los endpoins, y en teoría podrías crear toda la app

# Run

El comando de arranque depende de la ubicación del main en el proyecto

/main.py

* uvicorn main:app –reload

/src/main.py

* uvicorn src.main:app –reload

## puerto

* uvicorn main:app --reload --port 5000

# Endpoints

## Get

@app.get("/")

def home():

    return "Hola Mundo"

## PlainTextResponse

La forma vieja

from fastapi.responses import HTMLResponse,JSONResponse,PlainTextResponse

@app.get("/",tags="Home")

def home():

    return PlainTextResponse("Hola Mundo")

## json

@app.get("/home",tags="Home")

def home():

    return {'data':"Hola Mundo"}

### JSONResponse

La manera que creo que era la antigua era con esto

from fastapi.responses import JSONResponse

@app.get("/home",tags="Home")

def home():

    return JSONResponse({'data':"Hola Mundo"})

## HTMLResponse

from fastapi.responses import HTMLResponse

@app.get("/hola",tags="Home")

def homeConHtml():

    return HTMLResponse("<h1>Hola</h1>")

## parametro

Hay que tiparlos siempre

@app.get("/parametro/{id}",tags="Home")

def home(id:int):

    return "El parametro - "+str(id)

### Path

Para agregarle avalidacion extra a un parámetro, además de su tipo, como por ejemplo que un numero sea mayor que algo

from fastapi import FastAPI,Body,Path

@app.get("/parametro/{id}",tags="Home")

def home(id:int=Path(gt=0)):

    return "El parametro - "+str(id)

esto da error 422

## query param

los query param son los parametros de la función que no están definidos dentro de la ruta, si no se les da un valor por defecto son obligatorios (Hay que tiparlos)

@app.get("/query",tags="Home")

def home(qpcategory:str,qpopcional:str="zxc"):

    return "El query param - "+qpcategory

<http://localhost:8000/query?qpcategory=asd>

### Query

Para validar los query params (además de su tipo)

from fastapi import FastAPI,Query

@app.get("/query",tags="Home")

def home(qpcategory:str=Query(max\_length=5)):

    return "El query param - "+qpcategory

## post

A los parametros que se le pongan =Body(), son de tipo post (no me gusta esta sintaxis)

from fastapi import FastAPI,Body

@app.post("/create",tags="Data")

def create\_data(

    name:str=Body(),

    age:int=Body()

    ):

    data.append({

        name:name,

        age:age

    })

    return data

## put

@app.put("/data/{id}",tags="Data")

def edit\_data(

    id:int,

    name:str=Body(),

    age:int=Body()

    ):

    for d in data:

        if d["id"]==id:

            d["name"]=name

            d["age"]=age

    return data

## delete

@app.delete("/data/{id}",tags="Data")

def delete\_data(

    id:int,

    ):

    for d in data:

        if d["id"]==id:

            data.remove(d)

    return data

## RedirectResponse

Fijate que es necesario agregar el status\_code=303 para que funcione si es una redirección local

from fastapi.responses import HTMLResponse,JSONResponse,PlainTextResponse,RedirectResponse

@app.get("/data",tags="Data")

def get\_data()->List[Data]:

    return JSONResponse(data)

@app.post("/data",tags="Data")

def create\_data(

    data\_entry:Data

    ):

    new\_data=data\_entry.model\_dump()

    if "id" in new\_data and new\_data["id"] is None:

        new\_data["id"]=len(data)

    data.append(new\_data)

    return RedirectResponse("/data",status\_code=303)

## FileResponse

from fastapi.responses import FileResponse

@app.get("/file")

def get\_file():

    url=""

    return FileResponse(url)

## status\_code default

el status que se espera por defecto se puede cambiar agregándoselo al @app.tipometodo(…status\_code=201)

@app.post("/data",tags="Data",status\_code=201)

def create\_data(

    data\_entry:Data

    ):

    new\_data=data\_entry.model\_dump()

    if "id" in new\_data and new\_data["id"] is None:

        new\_data["id"]=len(data)

    data.append(new\_data)

    return RedirectResponse("/data",status\_code=303)

## HTTPException

Para lanzar error personalizados (4XX), no se van a ver igual que el de validación

from fastapi import HTTPException

@data\_router.get("/error",tags=["Data"])

def get\_data()->List[Data]:

    raise HTTPException(status\_code=400,detail="porque si")

# swagger

<http://localhost:8000/docs>

trae documentacion por defecto, por online

se pueden modificar algunos datos con

app.title="el titulo en swagger"

app.version="2.0.0"

## tags

Para agrupar rutas en swagger

@app.get("/",tags="Home")

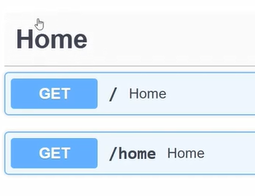
def home():

    return "Hola Mundo"

@app.get("/home",tags="Home")

def home():

    return "Hola Mundo"



# BaseModel

Para definir estructura de datos en clases

from pydantic import BaseModel

class Data(BaseModel):

    name:str

    age:int

Que lugeon pueden se pasada como parámetros, para definir la estructura del body, solo se puede aceptar un parámetro de BaseModel para que sea tomado como todo el body, como resultado desde el punto de vista del endpoint no importa el nombre de la variable, asi que lo que esperaría en este caso seria algo como esto

{

    "name":"newname",

    "age":2

}

Luego para extraer los datos en forma de dict

V= data\_entry.model\_dump()

@app.post("/data",tags="Data")

def create\_data(

    data\_entry:Data

    ):

    data.append({

        "id":len(data),

        \*\*data\_entry.model\_dump()

    })

    return data

Los errores por validación retornan 422

## Optional

Para tipos optionales

id:Optional[int]=None

Ejemplo

class Data(BaseModel):

    id:Optional[int]=None

    name:str

    age:int

Aunque en la peticion no se envie el tipo opcional, este va a estar presente con un valor de None

{

    "name":"newname",

    "age":2

}

@app.post("/data",tags="Data")

def create\_data(

    data\_entry:Data

    ):

    new\_data=data\_entry.model\_dump()

    if "id" in new\_data and new\_data["id"] is None:

        new\_data["id"]=len(data)

    data.append(new\_data)

    return data

## Field

Automáticamente se validan los tipos de datos, pero a veces hay que ser mas específico

from pydantic import BaseModel,Field

### string

class Data(BaseModel):

    id:Optional[int]=None

    name:str=Field(min\_length=5,max\_length=15)

    age:int

### numeros

puedes elegir entre “gt”,”ge”,”lt”,”le”

Ejemplo con el año actual (numero porque es solo el año)

class Fecha(BaseModel):

    year:int=Field(le=datetime.date.today().year)

    mount:int=Field(ge=1,le=12)

    day:int=Field(ge=1,le=31)

### default

class Fecha(BaseModel):

    year:int=Field(le=datetime.date.today().year,ge=1900,default=2025)

    mount:int=Field(ge=1,le=12)

    day:int=Field(ge=1,le=31)

## json\_schema\_extra

Podemos personalizar el ejemplo que sale para nuestro modelo en swagger

class Fecha(BaseModel):

    year:int=Field(le=datetime.date.today().year,ge=1900,default=2025)

    mount:int=Field(ge=1,le=12)

    day:int=Field(ge=1,le=31)

    model\_config={

        'json\_schema\_extra':{

            'example':{

                'year':2024,

                'mount':11,

                'day':25,

            }

        }

    }

## ~~@validator~~

Esta forma esta deprecada

class Data(BaseModel):

    id:Optional[int]=None

    name:str=Field(min\_length=5,max\_length=15)

    age:int=Field(ge=18)

    @validator('name')

    def validate\_title(cls,value):

        if value=="aaaaa":

            raise ValueError("este titulo no esta permitido")

        return value

# APIRouter

Normalmente se tienen las rutas en un paquete a parte

/src/routes/nombreentidad\_routes.py

Y Desde aquí no se debe de importar a app, asi que se utiliza en su lugar a APIRouter

from fastapi import APIRouter

data\_router=APIRouter()

@data\_router.get("/data",tags=["Data"])

def get\_data()->List[Data]:

    return JSONResponse(data)

y luego en el main.py

from .routes.data\_routes import data\_router

app.include\_router(router=data\_router)

## prefix

app.include\_router(prefix="/data",router=data\_router)

o tambien

data\_router=APIRouter(

    prefix="/data"

)

## responses

Puedes planear un tipo de respuesta común a incluir en la doc

data\_router=APIRouter(

    responses={404:{"message":"Not found"}}

)

## Tags

data\_router=APIRouter(

    responses={404:{"message":"Not found"}},

    tags=["Data"]

)

# BaseHTTPMiddleware

Para manejar errores no controlados (no validaciones)

import typing

from starlette.middleware.base import BaseHTTPMiddleware,DispatchFunction

from starlette.types import ASGIApp

from fastapi import status

from fastapi.responses import JSONResponse

class HTTPErrorHandler(BaseHTTPMiddleware):

    def \_\_init\_\_(self, app, dispatch = None):

        super().\_\_init\_\_(app, dispatch)

    async def dispatch(self, request, call\_next):

        try:

            return await call\_next(request)

        except Exception as e:

            content=str(e)

            status\_code=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR

            return JSONResponse(

                content=content,

                status\_code=status\_code

            )

Luego

app.add\_middleware(HTTPErrorHandler)

# @app.middleware

Otra manera de agregar un middleware

@app.middleware('http')

async def http\_error\_handler(request:Request,call\_next)->Response|JSONResponse:

    try:

        return await call\_next(request)

    except Exception as e:

        content=str(e)+"--2"

        status\_code=500

        return JSONResponse(

            content=content,

            status\_code=status\_code

        )