Microframework pour microservices

Fonctionnalités clés

- Fast to process
- Fast to code (200% more productive)
- Fewer bugs
- Easy to learn
- Production ready
- RESTful

Installation

```
$ pip install "fastapi[all]"
```

Hello World !

```python
from fastapi import FastAPI

app = FastAPI()


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

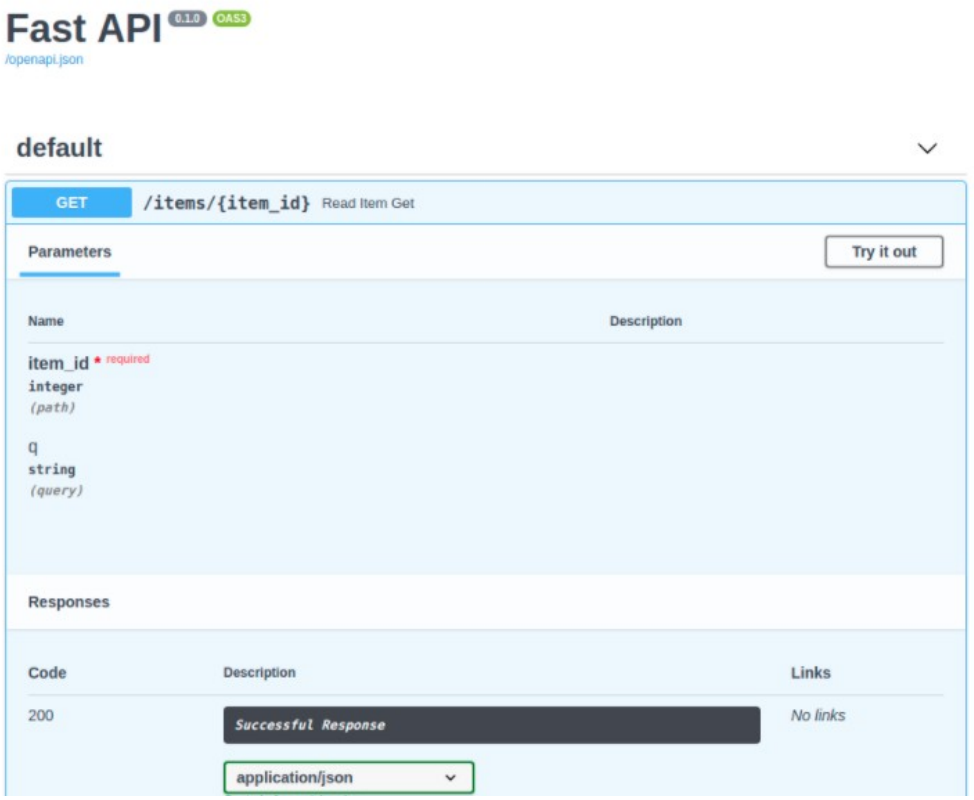
Run live server

```
$ uvicorn main:app --reload

INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [28720]
INFO:     Started server process [28722]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

And the magic goes...

Now go to http://127.0.0.1:8000/docs [↪].

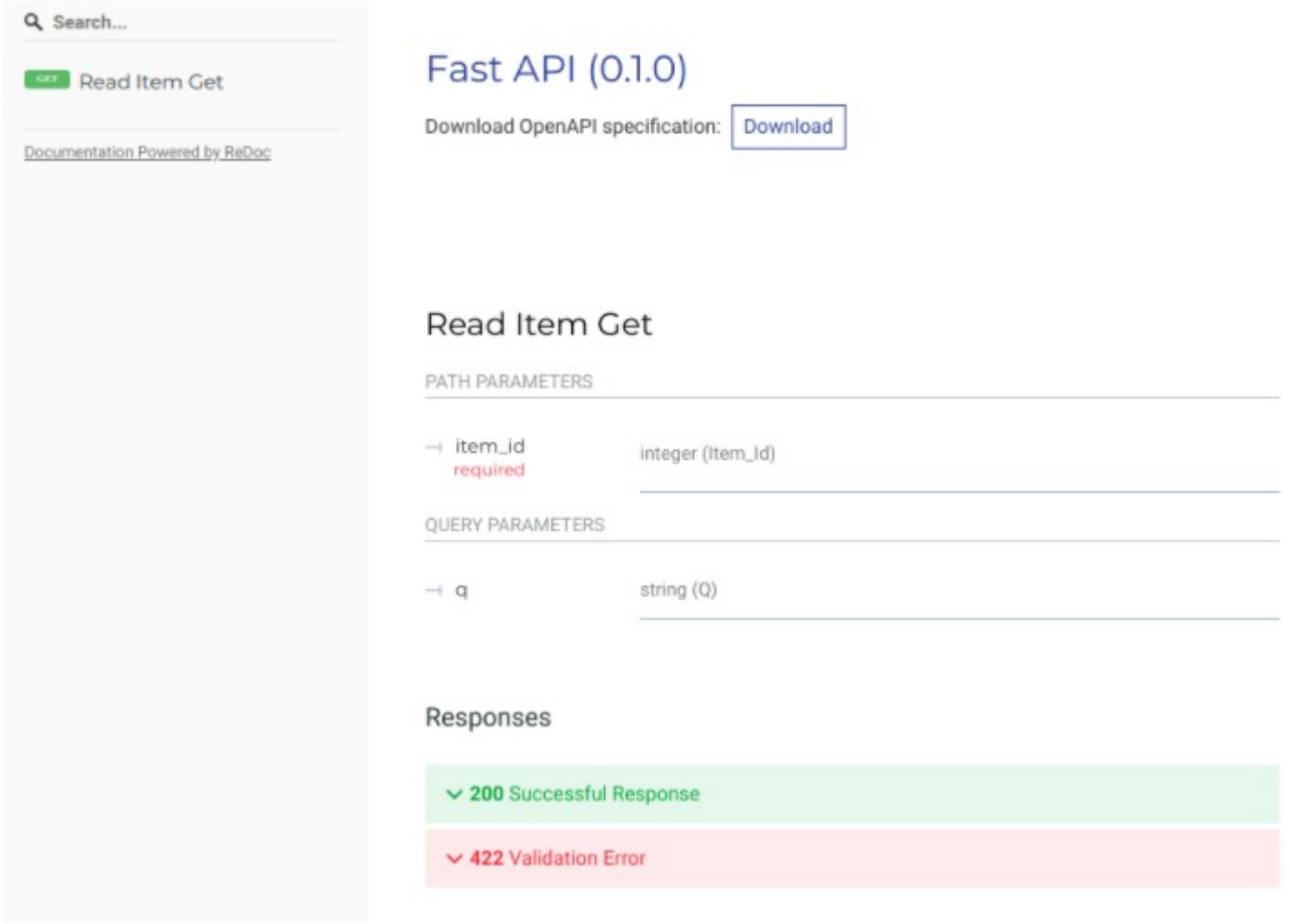
And the magic goes...



And the magic goes...

And now, go to http://127.0.0.1:8000/redoc [↪].

And the magic goes...



What is going on ?

```python
from fastapi import FastAPI

app = FastAPI()


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

Path parameters

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_item(item_id: int):
    return {"item_id": item_id}
```

Path parameters checker

http://127.0.0.1:8000/items/foo [↪],

Path parameters checker


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

Path parameters with Enum

```python
from enum import Enum
from fastapi import FastAPI

class ModelName(str, Enum):
    alexnet = "alexnet"
    resnet = "resnet"
    lenet = "lenet"

app = FastAPI()

@app.get("/models/{model_name}")
async def get_model(model_name: ModelName):
    if model_name == ModelName.alexnet:
        return {"model_name": model_name, "message": "Deep Learning FTW!"}
    if model_name.value == "lenet":
        return {"model_name": model_name, "message": "LeCNN all the images"}
    return {"model_name": model_name, "message": "Have some residuals"}
```

Query parameters

```python
from fastapi import FastAPI

app = FastAPI()

fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]

@app.get("/items/")
async def read_item(skip: int = 0, limit: int = 10):
    return fake_items_db[skip : skip + limit]
```

Query parameters

```
http://127.0.0.1:8000/items/?skip=0&limit=10
```

Combination parameters

```python
from typing import Optional
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")

async def read_item(item_id: str, q: Optional[str] = None):
    if q:
        return {"item_id": item_id, "q": q}
    return {"item_id": item_id}
```

Combination parameters

Try : http://127.0.0.1:8000/items/foo?q=query

Combination parameters

```python
from typing import Optional
from fastapi import FastAPI

app = FastAPI()

@app.get("/users/{user_id}/items/{item_id}")
async def read_user_item(
    user_id: int, item_id: str, q: Optional[str] = None, short: bool = False
):
    item = {"item_id": item_id, "owner_id": user_id}
    if q:
        item.update({"q": q})
    if not short:
        item.update(
            {"description": "This is an amazing item that has a long description"}
        )
    return item
```

Combination parameters

Try : http://127.0.0.1:8000/users/123/items/34?q=optional_query

Body parameters

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

class Item(BaseModel):
    name: str
    description: Optional[str] = None
    price: float
    tax: Optional[float] = None

app = FastAPI()

@app.post("/items/")
async def create_item(item: Item):
    return item
```

Body parameters

POST http://127.0.0.1/items

With :

```
{
    "name": "Foo",
    "description": "An optional description",
    "price": 45.2,
    "tax": 3.5
}
```

```
{
    "name": "Foo",
    "price": 45.2
}
```

Body parameters

Look at the updated documentation, each time you change the code ;)

Put example

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

class Item(BaseModel):
    name: str
    description: Optional[str] = None
    price: float
    tax: Optional[float] = None

app = FastAPI()

@app.put("/items/{item_id}")
async def create_item(item_id: int, item: Item, q: Optional[str] = None):
    result = {"item_id": item_id, **item.dict()}
    if q: result.update({"q": q})
    return result
```

Multiple Body parameters

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

app = FastAPI()

class Item(BaseModel):
    name: str
    description: Optional[str] = None
    price: float
    tax: Optional[float] = None

class User(BaseModel):
    username: str
    full_name: Optional[str] = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item, user: User):
    results = {"item_id": item_id, "item": item, "user": user}
    return results
```

Body parameters

POST http://127.0.0.1/items/42

With :

Body enhancement with Field

```python
from typing import Optional
from fastapi import Body, FastAPI
from pydantic import BaseModel, Field

app = FastAPI()

class Item(BaseModel):
    name: str
    description: Optional[str] = Field(

        None, title="The description of the item", max_length=300

    )
    price: float = Field(..., gt=0, description="The price must be greater than zero")
    tax: Optional[float] = None


@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item = Body(..., embed=True)):
    results = {"item_id": item_id, "item": item}
    return results
```

Body enhancement with Field

422

Error: Unprocessable Entity

Response body


```
{
  "detail": [
    {
      "loc": [
        "body",
        "item",
        "price"
      ],
      "msg": "ensure this value is greater than 0",
      "type": "value_error.number.not_gt",
      "ctx": {
        "limit_value": 0
      }
    }
  ]
}
```


Download

Response headers


```
content-length: 147
content-type: application/json
date: Wed,13 Oct 2021 14:34:44 GMT
server: uvicorn
```


With body « price » = -6

Nested Body

```python
from typing import Optional, Set
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Image(BaseModel):
    url: str
    name: str

class Item(BaseModel):
    name: str
    description: Optional[str] = None
    price: float
    tax: Optional[float] = None
    tags: Set[str] = []
    image: Optional[Image] = None

@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item):
    results = {"item_id": item_id, "item": item}
    return results
```

Nested Body

PUT http://127.0.0.1/items/42

With :

```
{
    "name": "Foo",
    "description": "The pretender",
    "price": 42.0,
    "tax": 3.2,
    "tags": ["rock", "metal", "bar"],
    "image": {
        "url": "http://example.com/baz.jpg",
        "name": "The Foo live"
    }
}
```

Headers

Be careful, **x_token** refers to key **X-Token** in header (case sensitive)

```
X-Token: foo
X-Token: bar
```

```python
from typing import List, Optional
from fastapi import FastAPI, Header

app = FastAPI()

@app.get("/items/")

async def read_items(x_token: Optional[List[str]] = Header(None)):
    return {"X-Token values": x_token}
```

Headers

Should return :

```
{
    "X-Token values": [
        "bar",
        "foo"
    ]
}
```

Response model

```python
app = FastAPI()


class UserIn(BaseModel):
    username: str
    password: str
    email: EmailStr
    full_name: Optional[str] = None


class UserOut(BaseModel):
    # We dont want to display plaintext password in response
    username: str
    email: EmailStr
    full_name: Optional[str] = None


@app.post("/user/", response_model=UserOut)
async def create_user(user: UserIn):
    return user
```

Status response code

```python
from fastapi import FastAPI

app = FastAPI()

@app.post("/items/", status_code=201)
async def create_item(name: str):
    return {"name": name}
```

Status response code

```python
from fastapi import FastAPI, status

app = FastAPI()

@app.post("/items/", status_code=status.HTTP_201_CREATED)
async def create_item(name: str):
    return {"name": name}
```

Binary file upload

```python
from fastapi import FastAPI, File, UploadFile

app = FastAPI()

@app.post("/files/")
async def create_file(file: bytes = File(...)):
    return {"file_size": len(file)}

@app.post("/uploadfile/")
async def create_upload_file(file: UploadFile = File(...)):
    return {"filename": file.filename}
```

HTTP Exceptions

```python
from fastapi import FastAPI, HTTPException

app = FastAPI()

items = {"foo": "The Foo"}

@app.get("/items/{item_id}")
async def read_item(item_id: str):
    if item_id not in items:
        raise HTTPException(status_code=404, detail="Item not found")
    return {"item": items[item_id]}
```

Middlewares

Should interact with requests, between client and server

Middlewares

```python
import time
from fastapi import FastAPI, Request
app = FastAPI()


@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
```

Middlewares CORS

Cross Origin Resource Sharing

An origin is the combination of PROTOCOL, DOMAIN NAME and PORT

- http://localhost
- https://locahost
- https://localhost:8000

Are all different origins...

Middlewares CORS

Assuming you are running a frontend at http://localhost:8000 and the javascript is trying to communicate with backend at http://localhost:4200

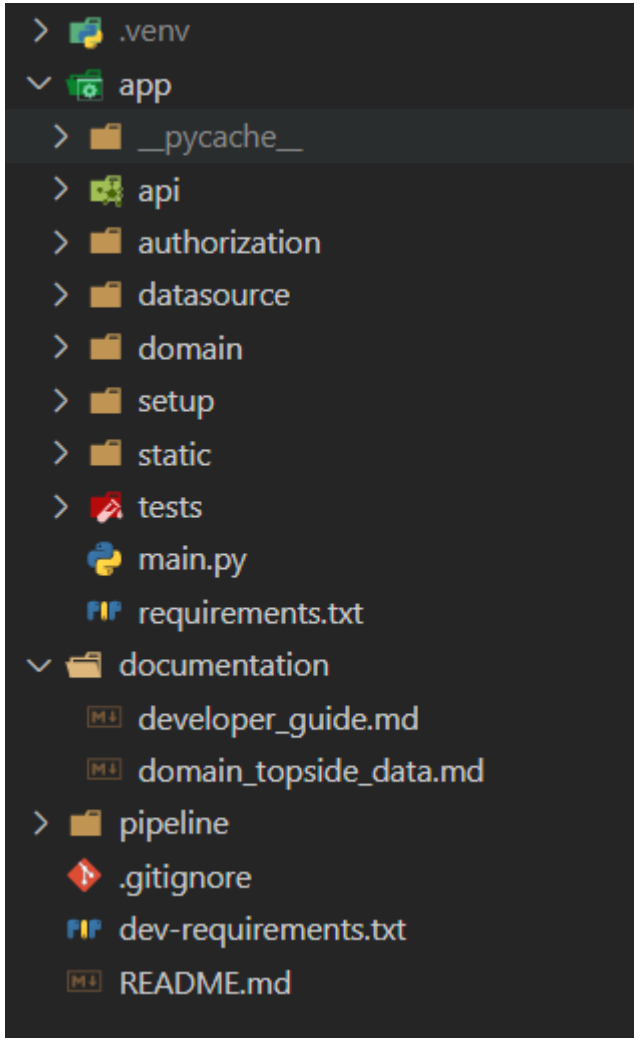
What’s going on ?

- The browser will sent OPTIONS request.
- The backend will validate and authorize the front end to interact with it.
- The browser will sent the request asked by the client to retrieve data from it.

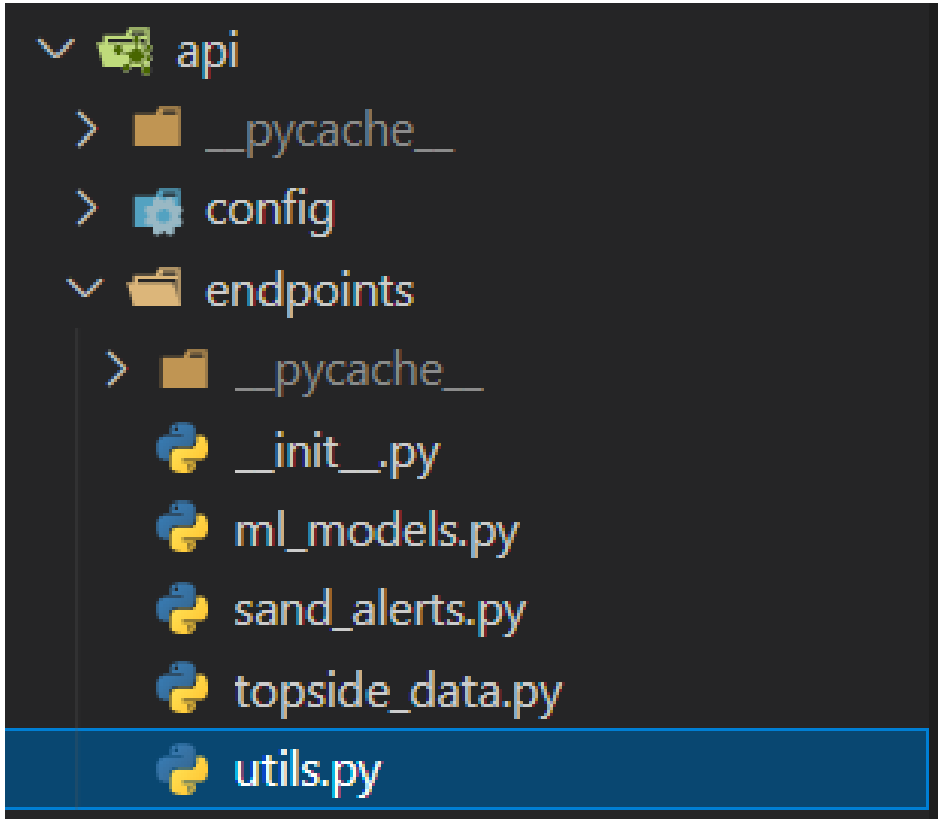
To achieve this, the backend must have a list of « allowed origins », thats done by CORS

```python
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

origins = [
    "http://localhost.tiangolo.com",
    "https://localhost.tiangolo.com",
    "http://localhost",
    "http://localhost:8080",
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)


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

Arborescence FastAPI



Arborescence FastAPI



Arborescence FastAPI

```python
app = FastAPI(
    title=NAME,
    version=VERSION,
    openapi_tags=tags_metadata,
    redoc_url=URL_DOC,
    docs_url=URL_SWAGGER,
    contact=CONTACTS
)

# Utils routes
app.include_router(utils.router)
```

```python
router = APIRouter(tags=[TAG])


@router.get("/")
async def about() -> Dict[str, str]:
    """Give informations about the API.

    Returns:
        Dict[str, str]: With shape :
    `
    {"app_version": <VERSION>,"api_url_doc":<URL_DOC>,"api_url_swagger":<URL_SWAGGER>,"app_contacts":<CONTACTS>}
    `
    """
    return {
        "app_version": VERSION,
        "api_url_doc": URL_DOC,
        "api_url_swagger": URL_SWAGGER,
        "app_contacts": CONTACTS
    }
```

Arborescence FastAPI

Pour le reste, comme une application Python classique

- Séparer son code selon le domaine implémenté (Connecteurs, Modèles de base de données, Process, Utilitaires, Contrôleurs, etc etc)

- Documenter en utilisant la Docstring Google