RÉFÉRENCE RAPIDE FASTAPI

Opérations de chemin, validation, dépendances, auth, tests

Configuration

Application minimale

```
from fastapi import FastAPI
app = FastAPI()

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

Lancer l'application

```
pip install "fastapi[standard]"
fastapi dev main.py    # dev with auto-reload
fastapi run main.py    # production
```

Fonctionnalités clés

`Async native`	async/await avec ASGI (Uvicorn)
`Auto docs`	Swagger UI sur `/docs`, ReDoc sur `/redoc`
`Type validation`	Modèles Pydantic pour requête/réponse
`OpenAPI`	Schéma OpenAPI généré automatiquement
`Dependency injection`	Système DI intégré

Opérations de chemin

Méthodes HTTP

```
@app.get("/items")
@app.post("/items")
@app.put("/items/{item_id}")
@app.patch("/items/{item_id}")
@app.delete("/items/{item_id}")
```

Paramètres de chemin

```
@app.get("/users/{user_id}")
async def get_user(user_id: int):
    return {"user_id": user_id}

# Enum constraint
from enum import Enum
class Color(str, Enum):
    red = "red"
    blue = "blue"
```

Codes de statut et tags

```
from fastapi import status

@app.post("/items", status_code=status.HTTP_201_CREATED,
          tags=["items"])
async def create_item(item: Item):
    return item
```

Corps de requête

Modèles Pydantic

```
from pydantic import BaseModel, Field

class Item(BaseModel):
    name: str
    price: float = Field(gt=0, description="Must be positive")
    tags: list[str] = []
```

Modèles imbriqués

```
class Address(BaseModel):
    street: str
    city: str
    zip_code: str

class User(BaseModel):
    name: str
    address: Address
```

Utilisation dans un endpoint

```
@app.post("/items")
async def create_item(item: Item):
    return {"name": item.name, "price": item.price}
```

Fonctionnalités de validation

`Field(gt=0)`	Supérieur à 0
`Field(min_length=1)`	Longueur minimale de chaîne
`Field(max_length=100)`	Longueur maximale de chaîne
`Field(pattern='^[a-z]+$')`	Correspondance par pattern regex
`Field(default=None)`	Optionnel avec valeur par défaut
`EmailStr`	Validation d'email (pydantic[email])

Paramètres de requête

Paramètres de requête basiques

```
@app.get("/items")
async def list_items(skip: int = 0, limit: int = 10):
    return items[skip : skip + limit]
# GET /items?skip=0&limit=20
```

Validation de requête

```
from fastapi import Query

@app.get("/search")
async def search(
    q: str = Query(min_length=3, max_length=50),
    page: int = Query(default=1, ge=1),
):
    return {"q": q, "page": page}
```

Optionnel et requis

```
async def read_items(
    q: str | None = None,   # optional
    name: str = ...,        # required (Ellipsis)
    tags: list[str] = Query(default=[]),
):
    return {"q": q, "name": name}
```

En-têtes et cookies

```
from fastapi import Header, Cookie

async def read(
    user_agent: str | None = Header(default=None),
    session_id: str | None = Cookie(default=None),
):
    return {"ua": user_agent}
```

Modèles de réponse

Modèle de réponse

```
class ItemOut(BaseModel):
    name: str
    price: float

@app.get("/items/{id}", response_model=ItemOut)
async def get_item(id: int):
    return items[id]  # filters out extra fields
```

Types de réponse multiples

```
from fastapi.responses import JSONResponse, HTMLResponse

@app.get("/html", response_class=HTMLResponse)
async def get_html():
    return "<h1>Hello</h1>"
```

Options du modèle de réponse

`response_model`	Modèle Pydantic pour filtrer la sortie
`response_model_exclude_unset`	Omettre les champs non définis explicitement
`response_model_include`	Liste blanche de champs spécifiques
`response_model_exclude`	Liste noire de champs spécifiques

Réponses d'erreur

```
from fastapi import HTTPException

@app.get("/items/{id}")
async def get_item(id: int):
    if id not in items:
        raise HTTPException(status_code=404, detail="Not found")
    return items[id]
```

Dépendances

Dépendance de fonction

```
from fastapi import Depends

async def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Utilisation dans un endpoint

```
@app.get("/users")
async def list_users(db: Session = Depends(get_db)):
    return db.query(User).all()
```

Dépendances basées sur des classes

```
class Pagination:
    def __init__(self, skip: int = 0, limit: int = 10):
        self.skip = skip
        self.limit = limit

@app.get("/items")
async def list_items(pg: Pagination = Depends()):
    return items[pg.skip : pg.skip + pg.limit]
```

Portées de dépendance

`Depends(func)`	Dépendance par endpoint
`app = FastAPI(dependencies=[...])`	Dépendance globale pour toutes les routes
`APIRouter(dependencies=[...])`	Dépendance au niveau du router
`yield`	Setup/teardown (sessions DB, verrous)

Authentification

OAuth2 Password Bearer

```
from fastapi.security import OAuth2PasswordBearer

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

@app.get("/users/me")
async def read_me(token: str = Depends(oauth2_scheme)):
    user = decode_token(token)
    return user
```

Flux de token JWT

```
from jose import jwt
SECRET = "your-secret-key"

def create_token(data: dict):
    return jwt.encode(data, SECRET, algorithm="HS256")

def decode_token(token: str):
    return jwt.decode(token, SECRET, algorithms=["HS256"])
```

Endpoint de token

```
from fastapi.security import OAuth2PasswordRequestForm

@app.post("/token")
async def login(form: OAuth2PasswordRequestForm = Depends()):
    user = authenticate(form.username, form.password)
    if not user:
        raise HTTPException(status_code=401)
    return {"access_token": create_token({"sub": user.id})}
```

Schémas de sécurité

`OAuth2PasswordBearer`	Token Bearer via formulaire de connexion
`HTTPBasic`	Auth basique nom d'utilisateur/mot de passe
`APIKeyHeader`	Clé API dans l'en-tête
`APIKeyCookie`	Clé API dans le cookie

Tâches en arrière-plan

Tâche en arrière-plan simple

```
from fastapi import BackgroundTasks

def send_email(to: str, body: str):
    # slow operation runs after response
    email_client.send(to, body)

@app.post("/notify")
async def notify(bg: BackgroundTasks):
    bg.add_task(send_email, "user@example.com", "Hello!")
    return {"status": "queued"}
```

Dépendance avec arrière-plan

```
async def log_request(bg: BackgroundTasks):
    bg.add_task(write_log, "request received")

@app.get("/items", dependencies=[Depends(log_request)])
async def list_items():
    return items
```

Arrière-plan vs workers

`BackgroundTasks`	Tâches légères après la réponse (emails, logs)
`Celery / ARQ`	Tâches lourdes nécessitant des workers séparés
`asyncio.create_task`	Coroutines async fire-and-forget

Middleware

Middleware personnalisé

```
import time
from starlette.middleware.base import BaseHTTPMiddleware

class TimingMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request, call_next):
        start = time.time()
        response = await call_next(request)
        duration = time.time() - start
        response.headers["X-Process-Time"] = str(duration)
        return response
```

Ajouter un middleware

```
app.add_middleware(TimingMiddleware)
```

CORS

```
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://example.com"],
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Middleware intégré

`CORSMiddleware`	Partage de ressources cross-origin
`TrustedHostMiddleware`	Restreindre les noms d'hôtes autorisés
`GZipMiddleware`	Compression Gzip des réponses
`HTTPSRedirectMiddleware`	Rediriger HTTP vers HTTPS

Tests

Client de test

```
from fastapi.testclient import TestClient

client = TestClient(app)

def test_read_root():
    resp = client.get("/")
    assert resp.status_code == 200
    assert resp.json() == {"message": "Hello, World!"}
```

Test POST

```
def test_create_item():
    resp = client.post("/items", json={
        "name": "Widget",
        "price": 9.99,
    })
    assert resp.status_code == 201
    assert resp.json()["name"] == "Widget"
```

Surcharger les dépendances

```
async def mock_db():
    return FakeDB()

app.dependency_overrides[get_db] = mock_db

def test_with_mock_db():
    resp = client.get("/users")
    assert resp.status_code == 200
```

Tests asynchrones

```
import pytest
from httpx import AsyncClient, ASGITransport

@pytest.mark.anyio
async def test_async():
    transport = ASGITransport(app=app)
    async with AsyncClient(transport=transport) as ac:
        resp = await ac.get("/")
    assert resp.status_code == 200
```