FASTAPI QUICK REFERENCE

Path operations, validation, dependencies, auth, testing

Setup

Minimal App

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

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

Run the App

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

Key Features

`Async native`	async/await with ASGI (Uvicorn)
`Auto docs`	Swagger UI at `/docs`, ReDoc at `/redoc`
`Type validation`	Pydantic models for request/response
`OpenAPI`	Auto-generated OpenAPI schema
`Dependency injection`	Built-in DI system

Path Operations

HTTP Methods

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

Path Parameters

```
@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"
```

Status Codes & Tags

```
from fastapi import status

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

Request Body

Pydantic Models

```
from pydantic import BaseModel, Field

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

Nested Models

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

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

Use in Endpoint

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

Validation Features

`Field(gt=0)`	Greater than 0
`Field(min_length=1)`	Minimum string length
`Field(max_length=100)`	Maximum string length
`Field(pattern='^[a-z]+$')`	Regex pattern match
`Field(default=None)`	Optional with default
`EmailStr`	Email validation (pydantic[email])

Query Parameters

Basic Query Params

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

Query Validation

```
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}
```

Optional & Required

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

Headers & 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}
```

Response Models

Response Model

```
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
```

Multiple Response Types

```
from fastapi.responses import JSONResponse, HTMLResponse

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

Response Model Options

`response_model`	Pydantic model for output filtering
`response_model_exclude_unset`	Omit fields not explicitly set
`response_model_include`	Whitelist specific fields
`response_model_exclude`	Blacklist specific fields

Error Responses

```
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]
```

Dependencies

Function Dependency

```
from fastapi import Depends

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

Use in Endpoint

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

Class-Based Dependencies

```
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]
```

Dependency Scopes

`Depends(func)`	Per-endpoint dependency
`app = FastAPI(dependencies=[...])`	Global dependency for all routes
`APIRouter(dependencies=[...])`	Router-level dependency
`yield`	Setup/teardown (DB sessions, locks)

Authentication

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
```

JWT Token Flow

```
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"])
```

Token Endpoint

```
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})}
```

Security Schemes

`OAuth2PasswordBearer`	Bearer token via form login
`HTTPBasic`	Basic username/password auth
`APIKeyHeader`	API key in header
`APIKeyCookie`	API key in cookie

Background Tasks

Simple Background Task

```
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"}
```

Dependency with Background

```
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
```

Background vs Workers

`BackgroundTasks`	Light tasks after response (emails, logs)
`Celery / ARQ`	Heavy tasks needing separate workers
`asyncio.create_task`	Fire-and-forget async coroutines

Middleware

Custom Middleware

```
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
```

Add 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=["*"],
)
```

Built-in Middleware

`CORSMiddleware`	Cross-origin resource sharing
`TrustedHostMiddleware`	Restrict allowed hostnames
`GZipMiddleware`	Gzip response compression
`HTTPSRedirectMiddleware`	Redirect HTTP to HTTPS

Testing

Test Client

```
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"
```

Override Dependencies

```
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
```

Async Testing

```
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
```