FastAPI Cheat Sheet

Async · OpenAPI · Pydantic · Dependency Injection

SETUP & RUN

```
pip install fastapi uvicorn[standard]
from fastapi import FastAPI
app = FastAPI(title='My API',
version='1.0.0')
uvicorn main:app --reload
uvicorn main:app --host 0.0.0.0 --port
8000
```

Docs: /docs (Swagger) and /redoc (ReDoc)

PATH OPERATIONS

```
@app.get('/items',
response_model=List[Item])
async def list_items():
return await db.items.find_all()
@app.post('/items', status_code=201)
async def create(item: Item):
return await db.items.insert(item)
```

HTTP verbs: get post put patch delete head options

PATH & QUERY PARAMS

```
@app.get('/users/{user_id}')
async def get_user(user_id: int):
return {'id': user_id}
@app.get('/search')
async def search(q: str, limit: int =
10, skip: int = 0):
...
```

Type annotations auto-validate and document

REQUEST BODY (PYDANTIC)

```
from pydantic import BaseModel, EmailStr
class User(BaseModel):
name: str
email: EmailStr
age: int = Field(ge=0, le=120)
@app.post('/users')
async def create(user: User): ...
```

Automatic validation + OpenAPI schema generation

DEPENDENCY INJECTION

```
from fastapi import Depends
def get_db() -> Generator:
db = SessionLocal()
try: yield db
finally: db.close()
async def
get_current_user(token=Depends(oauth2_scheme)):
...
def endpoint(db=Depends(get_db),
user=Depends(get_current_user)):
```

RESPONSE MODELS

```
class UserOut(BaseModel):
id: int
name: str
# no password field
@app.get('/users/{id}',
response_model=UserOut)
async def get_user(id: int): ...
response_model_exclude_unset=True  #
skip defaults
response_model_exclude={'password'}
```

AUTH (OAUTH2 + JWT)

```
from fastapi.security import
OAuth2PasswordBearer
oauth2_scheme =
OAuth2PasswordBearer(tokenUrl='token')
@app.post('/token')
async def login(form:
OAuth2PasswordRequestForm=Depends()):
user = authenticate(form.username,
form.password)
token = create_jwt(user.id)
return {'access_token': token,
'token_type': 'bearer'}
```

ASYNC ENDPOINTS

```
@app.get('/slow')
async def slow_endpoint():
result = await slow_io_operation()
return result
import asyncio
await asyncio.gather(task1(), task2())
```

Use sync def for CPU-bound (runs in threadpool)
Use async def for I/O-bound operations

BACKGROUND TASKS

```
from fastapi import BackgroundTasks
@app.post('/notify')
def notify(bg: BackgroundTasks, email:
str):
bg.add_task(send_email, email,
'Welcome!')
return {'message': 'queued'}
```

For heavy work: use Celery + Redis instead

TESTING

```
from fastapi.testclient import
TestClient
client = TestClient(app)
def test_create():
resp = client.post('/users',
json={'name':'Alice','email':'a@b.com'})
assert resp.status_code == 201
assert resp.json()['name'] == 'Alice'
```

Override deps:
app.dependency_overrides[get_db] = mock_db

COMMON GOTCHAS

async def needs await forgetting await blocks event loop
Pydantic V2 breaking changes vs V1 (field vs Field)
Don't use sync I/O inside async endpoints
global state is shared across requests use Depends
Response model strips extra fields can miss data