FastAPI Python web APIs

START HERE

`fastapi dev main.py`
Run with auto-reload

`TestClient(app)`
Test without a server

`@app.get("/items/{id}")`
Declare a route

`localhost:8000/docs`
Interactive API docs

`q: str | None = None`
Optional query parameter

`class Item(BaseModel)`
Typed request body

`Depends(`*`<func>`***`)`**
Inject a dependency

`raise HTTPException(404)`
Return an error

Request path, query, body → `validate` → **Pydantic** types to values → `Depends` → **Path function** your code → `response_model` → **Response** JSON + OpenAPI

CONCEPTS

FastAPI reads your type hints: they decide where a parameter comes from and how it is validated.

Validation failures return 422 automatically; no manual checks needed.

Use plain def for blocking code and async def only when everything inside is awaited.

RUN SERVER

`fastapi dev`
Dev server with reload (auto-detects app)

`fastapi dev `*`<path/to/file.py>`***
Run a specific file or package

`fastapi run`
Production mode, no reload, 0.0.0.0

`fastapi dev --port `*`<port>`***
Change the port (default 8000)

`fastapi dev --host 0.0.0.0`
Listen on all interfaces

`fastapi dev --app `*`<name>`***
Pick the app variable in the module

`fastapi dev -e `*`<module:app>`***
Give the import string directly

RUN SERVER (cont.)

`fastapi run --workers `*`<n>`***
Several worker processes
Cannot be combined with --reload.

`fastapi run --root-path /api`
App sits behind a path-prefix proxy

APP AND ROUTES

`app = FastAPI(title="`*`<name>`***`")`**
Create the app

`@app.get("/")`
Handle GET requests

`@app.post("/items")`
Handle POST requests

`@app.put("/items/{item_id}")`
Handle PUT requests

`@app.delete("/items/{item_id}")`
Handle DELETE requests

`@app.get("/x", tags=["x"], summary="...")`
Group and label docs entries

`router = APIRouter(prefix="/items")`
Group routes in a module

`app.include_router(router)`
Mount a router

`app.add_middleware(CORSMiddleware, allow_origins=[...])`
Enable CORS

PARAMETERS

`def f(item_id: int)`
Path parameter from {item_id}

`def f(skip: int = 0, limit: int = 10)`
Query parameters with defaults

`def f(q: str | None = None)`
Optional query parameter

`q: Annotated[str, Query(max_length=50)]`
Validate a query parameter

`n: Annotated[int, Path(ge=1)]`
Validate a path parameter

`tags: Annotated[list[str], Query()]`
Repeated query values (?tags=a&tags=b)

`x: Annotated[str | None, Header()] = None`
Read a request header

`c: Annotated[str | None, Cookie()] = None`
Read a cookie

`name: Annotated[str, Form()]`
Read a form field
Needs the python-multipart package.

`f: UploadFile`
Receive an uploaded file

REQUEST BODY

`class Item(BaseModel):`
Define a body schema

REQUEST BODY (cont.)

`def f(item: Item)`
Parse the JSON body into a model

`price: float = Field(gt=0)`
Constrain a model field

`tags: list[str] = []`
Field with a default value

`item.model_dump()`
Model to a dict

`item.model_dump(exclude_unset=True)`
Only fields the client sent (PATCH)

RESPONSES

`@app.get("/x", response_model=Item)`
Filter and document the output

`@app.post("/x", status_code=201)`
Set the success status code

`from fastapi import status`
Named codes: status.HTTP_201_CREATED

`raise HTTPException(status_code=404, detail="...")`
Send an error response

`return JSONResponse(content=`*`<d>`***`, status_code=`***`<n>`***`)`**
Build the response yourself

`return RedirectResponse("/new")`
Redirect the client

DEPENDENCIES

`def f(db: Annotated[Session, Depends(get_db)])`
Inject a dependency

`CommonParams = Annotated[dict, Depends(common)]`
Reusable dependency alias

`def get_db(): yield db`
Dependency with cleanup after the response

`@app.get("/x", dependencies=[Depends(`*`<f>`***`)])`**
Run a dependency without using its value

`app.dependency_overrides[`*`<d>`***`] = `***`<f>`***
Swap a dependency in tests

`OAuth2PasswordBearer(tokenUrl="token")`
Bearer-token security scheme

`def f(tasks: BackgroundTasks)`
Queue work after the response

`tasks.add_task(`*`<func>`***`, `***`<arg>`***`)`**
Schedule a background job

TESTING AND DOCS

`client = TestClient(app)`
Test client (needs httpx)

`client.get("/items/1")`
Call a route in a test

TESTING AND DOCS (cont.)

`client.post("/items", json={...})`
POST a JSON body

`assert r.status_code == 200`
Check the status

`r.json()`
Decode the JSON response

`/docs`
Swagger UI

`/redoc`
ReDoc docs

`/openapi.json`
OpenAPI schema

RECIPES

Minimal app

```
# main.py
from fastapi import FastAPI
app = FastAPI()

@app.get("/items/{item_id}")
def read(item_id: int, q: str | None = None):
    return {"id": item_id, "q": q}

# run: fastapi dev
```

Create endpoint with body

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

@app.post("/items", status_code=201)
def create(item: Item) -> Item:
    return item
```