FastAPI

A quick reference for FastAPI, building typed web APIs where your function signatures define the routes, validation, and docs.

Routes: declare endpoints

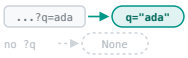
One decorator per HTTP method maps a path to a function.

PURPOSE	COMMAND	RESULT
Create the app.	`app = FastAPI(title="My API", version="1.0")`	
Read endpoint (GET).	`@app.get("/items")` `def read_items(): ...`	
Create endpoint (POST).	`@app.post("/items")` `def make_item(): ...`	
Update / partial / delete.	`@app.put("/items/{id}")` `@app.patch(...) @app.delete(...)`	
Group routes in a router.	`router = APIRouter(prefix="/v1", tags=["items"])` `app.include_router(router)`	
Return value becomes the Response.	`def read_items():` `  return {"items": [...]}`	

`@app.get / .post / .put / .patch / .delete bind a method and path to the function below; APIRouter groups them under a prefix`

Params: typed path and query

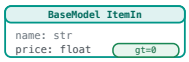
Type hints become validation; declare path and query with Annotated.

PURPOSE	COMMAND	RESULT
Typed path parameter.	`@app.get("/items/{item_id}")` `def get_item(item_id:int): ...`	
Constrain a path param.	`item_id: Annotated[int,Path(ge=1)]`	
Optional query param.	`q: Annotated[str | None,Query()] = None`	
Constrain a query param.	`limit: Annotated[int,Query(ge=1, le=100)] = 10`	
Auto type coercion.	`flag:bool, page:int= 1`	
Bad type is rejected.	`GET /items/abc # item_id:int` `422 Unprocessable Entity before your code runs`	

`Path params come from the URL, plain args become query params; Annotated[T, Path/Query(...)] adds constraints and a bad value returns 422 automatically`

Models: Pydantic Request & Response

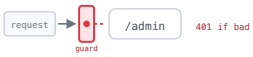
A BaseModel validates the JSON body in and shapes the JSON out.

PURPOSE	COMMAND	RESULT
Define a request model.	`class ItemIn(BaseModel):` `name: str; price: float = Field(gt=0)`	
Accept it as the body.	`def make(item: ItemIn): ...`	
Reject an invalid body.	`{"price": -1}` vs `Field(gt=0)`	
Shape the response.	`@app.post("/items", response_model=ItemOut)`	
Extra body field controls.	`tags: list[str] = []` `q: Annotated[str, Body()]`	
Validated docs for free.	`# the model drives the OpenAPI schema`	

A BaseModel arg parses and validates the JSON body (422 on failure); response_model shapes what leaves and feeds /docs

Depends: dependency injection

Declare what a route needs; FastAPI builds and injects it.

PURPOSE	COMMAND	RESULT
Define a dependency.	`def common(q: str|None=None, page=1):`	
Inject it into a route.	`p: Annotated[dict, Depends(common)]`	
Reuse across many routes.	`3 routes share Depends(common)`	
Setup and teardown with yield.	`def get_db(): db=connect()` `try: yield db; finally: db.close()`	
Route-level guard (no value).	`@app.get("/admin", dependencies=[Depends(verify)])`	
Layer sub-dependencies.	`def b(a=Depends(dep_a)): ...`	

Annotated[T, Depends(provider)] injects a value; dependencies=[Depends(guard)] runs a check with nothing passed in; yield gives setup/teardown

Errors

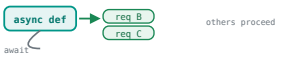
Set the success code; raise HTTPException for the rest.

PURPOSE	COMMAND	RESULT
Set the success status.	`@app.post("/items", status_code=status.HTTP_201_CREATED)`	
Named status constants.	`status.HTTP_404_NOT_FOUND, status.HTTP_204_NO_CONTENT`	
Raise a client error.	`raise HTTPException(status_code=404, detail="not found")`	
Add headers to the error.	`raise HTTPException(401, detail="…", headers={"WWW-Authenticate": "Bearer"})`	
Validation errors are automatic.	`# bad body or param FastAPI builds it`	
Custom exception handler.	`@app.exception_handler(MyError) def handle(...)`	

status_code= sets the happy path; raise HTTPException for failures; bad input becomes a 422 automatically

Async & Background Work

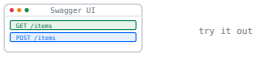
async def for awaitable I/O; plain def runs in a threadpool.

PURPOSE	COMMAND	RESULT
Async handler awaits I/O.	`async def read(): await fetch()`	
Plain def offloaded to a thread.	`def read(): return blocking_call()`	
Await an async dependency.	`async def get_db(): async with pool.acquire() as c: yield c`	
Run work after responding.	`bg: BackgroundTasks; bg.add_task(send_email)`	
App startup and shutdown.	`async def lifespan(app): setup(); yield; teardown() FastAPI(lifespan=lifespan)`	
Avoid the deprecated decorator.	`@app.on_event("startup")`	

async def awaits non-blocking I/O; plain def runs in a threadpool; BackgroundTasks defers work; FastAPI(lifespan=...) replaces @app.on_event

Docs: automatic OpenAPI

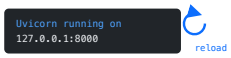
Your types generate an OpenAPI schema and live docs, free.

PURPOSE	COMMAND	RESULT
Interactive docs (Swagger).	`openhttp://127.0.0.1:8000/docs`	
Alternative docs (ReDoc).	`openhttp://127.0.0.1:8000/redoc`	
The raw schema (JSON).	`openhttp://127.0.0.1:8000/openapi.json`	
Describe routes for docs.	`@app.get("/items", tags=["items"], summary="List items")`	
Document the model fields.	`name: str = Field(examples=["Ada"], …)`	
Customize or disable docs.	`FastAPI(docs_url="/api-docs", redoc_url=None)`	

Typed routes and Pydantic models generate OpenAPI 3.1 at /openapi.json, with Swagger UI at /docs and ReDoc at /redoc for free

Run: serve with uvicorn

fastapi dev to develop, fastapi run (uvicorn) to serve.

PURPOSE	COMMAND	RESULT
Develop with auto-reload.	`fastapi dev main.py`	
Run for production.	`fastapi run main.py`	
Invoke uvicorn explicitly.	`uvicorn main:app --reload --port 8000`	
Run from inside Python.	`if __name__ == "__main__":` `uvicorn.run("main:app", reload=True)`	
Scale with worker processes.	`uvicorn main:app --workers 4`	
Find the docs once it is up.	`server up -> visit /docs`	

fastapi dev (reload) and fastapi run wrap uvicorn; uvicorn main:app or uvicorn.run("main:app") work too, --workers N to scale