How a Server Works

Everything FastAPI does between a request and your function

E190BO · AI-Amplified Prototyping

Seven big ideas

1. **A server is a loop that never ends.**
2. **Routing picks your function:** a request's verb and path decide which code runs.
3. **Part of the URL can be an argument:** `/ideas/2` calls your function with `2`.
4. **FastAPI checks what the client sends,** before your function runs.
5. **The status code says what happened:** some codes you choose, some FastAPI does.
6. **The server forgets;** anything that has to last goes in a database.
7. **A test is a request and a comparison.**

1 · A server is a loop that never ends

A whole web server, without a framework

servers-1-hello/server.py

```
import socket

server = socket.socket()
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)  # quick restarts
server.bind(("127.0.0.1", 8000))        # 1. claim a port
server.listen()
print("listening on port 8000")

while True:                             # 2. loop forever
    conn, addr = server.accept()        #    wait until someone connects
    request = conn.recv(4096).decode()
    print(request.split("\r\n")[0])     #    e.g. GET /ideas HTTP/1.1
    body = '{"message": "hello"}'
    conn.sendall((                      # 3. answer: status, headers,
        "HTTP/1.1 200 OK\r\n"           #    a blank line, then the body
        "Content-Type: application/json\r\n"
        f"Content-Length: {len(body)}\r\n"
        "\r\n" + body).encode())
    conn.close()
```

▶ LIVE Run it, and talk to it

1. Terminal 1: `python server.py`. It prints `listening on port 8000` and **doesn't give the prompt back**.
2. Open a **second terminal** with the `+`.
3. Terminal 2: `curl -i localhost:8000/ideas`. Terminal 1 now shows `GET /ideas HTTP/1.1`.
4. Terminal 2: `curl localhost:8000/ideas/3`. The same hello.
5. Terminal 1: **Ctrl-C** stops it.

PAUSE THE VIDEO

You stop the server with Ctrl-C.

Then you run `curl localhost:8000/ideas` again.

What happens?

2 · Routing picks your function

Routing by hand: (verb, path) → code

servers-2a-routing/server.py

```
import socket

def respond(conn, status, body):
    conn.sendall((f"HTTP/1.1 {status}\r\n"
                  "Content-Type: application/json\r\n"
                  f"Content-Length: {len(body)}\r\n"
                  "\r\n" + body).encode())

server = socket.socket()
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind(("127.0.0.1", 8000))
server.listen()

while True:
    conn, addr = server.accept()
    request_line = conn.recv(4096).decode().split("\r\n")[0]
    method, path, version = request_line.split(" ")   # "GET", "/ideas", ...

    if method == "GET" and path == "/ideas":
        respond(conn, "200 OK", '[{"id": 1, "title": "Automatic plant waterer"}]')
    elif method == "POST" and path == "/ideas":
        respond(conn, "201 Created", '{"id": 2, "title": "(pretend we saved it)"}')
    else:
        respond(conn, "404 Not Found", '{"detail": "Not Found"}')
    conn.close()
```

▶ LIVE Ask it three things

1. Terminal 1: `python server.py`.
2. Terminal 2, one at a time:

Request	Answer
`curl -i localhost:8000/ideas`	`200 OK`, the list
`curl -i -X POST localhost:8000/ideas`	`201 Created`
`curl -i localhost:8000/nope`	`404 Not Found`

3. Terminal 1: **Ctrl-C**.

The same idea, with FastAPI

servers-2b-fastapi/main.py

```
from fastapi import FastAPI

app = FastAPI()
ideas = [{"id": 1, "title": "Automatic plant waterer"}]

@app.get("/ideas")                    # the verb, and the path
def list_ideas():                     # the code that answers it
    return ideas                      # a list or dict becomes JSON for you
```

```
uvicorn main:app --reload
```

- **`uvicorn`** is the loop from chapter 1: it holds the port and hands each request over.
- **`main:app`** means *the variable `app`, in the file `main.py`*.
- **`--reload`** restarts the server whenever you save.

▶ LIVE `/docs` comes free

1. Terminal 1: `uvicorn main:app --reload`.
2. Browser: `localhost:8000/docs`.
3. Every route is listed, with its verb.
4. Open `GET /ideas` → **Try it out** → **Execute**.

3 · Part of the URL can be an argument

The `2` in `/ideas/2`

servers-3-path-parameter/main.py

```python
from fastapi import FastAPI

app = FastAPI()
ideas = [{"id": 1, "title": "Automatic plant waterer"}]

@app.get("/ideas")
def list_ideas():
    return ideas

@app.get("/ideas/{idea_id}")          # {idea_id} becomes the argument
def get_idea(idea_id: int):           # int: FastAPI turns "1" into 1 first
    return ideas[idea_id - 1]         # first take: fine for ids that exist
```

- `{idea_id}` in the path becomes the `idea_id` argument.
- `: int` makes FastAPI convert the text **before** calling you. If it can't, your function **never runs**.

▶ LIVE Try two ids

1. Terminal 1: `uvicorn main:app --reload`.
2. `/docs` → `GET /ideas/{idea_id}` → **Try it out**.
3. `1` → **200**, the idea.
4. `abc` → **422**:


```
{"detail":[{"type":"int_parsing","loc":["path","idea_id"],
  "msg":"Input should be a valid integer, unable to parse string as an
integer","input":"abc"}]}
```

PAUSE THE VIDEO

Your route is `/ideas/{idea_id}` with `idea_id: int`.

Which of these reach your function?

`/ideas/2` · `/ideas/abc` · `/ideas/99999`

4 · FastAPI checks what the client sends

A class says what the client may send

servers-4-request-body/main.py

```python
from fastapi import FastAPI
from sqlmodel import SQLModel

app = FastAPI()
ideas = [{"id": 1, "title": "Automatic plant waterer"}]

@app.get("/ideas")
def list_ideas():
    return ideas

@app.get("/ideas/{idea_id}")
def get_idea(idea_id: int):
    return ideas[idea_id - 1]      # first take: fine for ids that exist

class IdeaCreate(SQLModel):        # what a client may send: no id
    title: str
    body: str = ""

@app.post("/ideas", status_code=201)
def create_idea(payload: IdeaCreate):
    idea = {"id": len(ideas) + 1, **payload.model_dump()}
    ideas.append(idea)
    return idea
```

- The JSON body is checked against the class **before** your function runs. `title` is required; `body` has a default.

What the client sends isn't what you store

	The client sends	You store
`title`, `body`	✓	✓
`id`	no	✓ the server picks it

- So the request gets **its own class**, without an `id`.
- In Lab 5 your table class has an `id`, and it's also **not validated**: used as a request body, a bad `POST` crashes at the database (500) instead of a 422.

▶ LIVE Send it something wrong

1. Terminal 1: `uvicorn main:app --reload`.
2. `/docs` → `POST /ideas` → **Try it out** → `{"title": "Sticker vending machine"}` → **201**, `"id": 2`.
3. Send it again with `{}`. **422**:

```
{"detail":[{"type":"missing","loc":["body","title"],"msg":"Field required","input":{}}]}
```

5 · The status code says what happened

Your API answers with a number first

Code	Means	Who sends it
200 OK	here's what you asked for	your function, by default
201 Created	I made something new	your function: `status_code=201`
404 Not Found	no such thing	your function: `raise HTTPException(404, …)`
422 Unprocessable	your request's data is wrong	**FastAPI**, before your function runs
500 Server Error	my code crashed	nobody meant to: it's a bug

▶ LIVE Ask for an idea that isn’t there

1. Still running from chapter 4 (`servers-4-request-body`).
2. `/docs` → `GET /ideas/{idea_id}` → `99` → **500**, body `Internal Server Error`.
3. The response says nothing useful. **Terminal 1** says why: `IndexError: list index out of range`.

The fix: decide what “not there” means

servers-5-status-codes/main.py

```python
from fastapi import FastAPI, HTTPException
from sqlmodel import SQLModel

app = FastAPI()
ideas = [{"id": 1, "title": "Automatic plant waterer"}]

@app.get("/ideas")
def list_ideas():
    return ideas

@app.get("/ideas/{idea_id}")
def get_idea(idea_id: int):
    for idea in ideas:
        if idea["id"] == idea_id:
            return idea
    raise HTTPException(404, "no such idea")

class IdeaCreate(SQLModel):        # what a client may send: no id
    title: str
    body: str = ""

@app.post("/ideas", status_code=201)
def create_idea(payload: IdeaCreate):
    idea = {"id": len(ideas) + 1, **payload.model_dump()}
    ideas.append(idea)
    return idea
```

PAUSE THE VIDEO

Which status code?

1. `POST /ideas` with `{"title": "Bike lock"}`, saved
2. `GET /ideas/7`, and there is no idea 7
3. `POST /ideas` with `{"title": 42}`
4. `GET /ideas`, and your code has a typo that raises `NameError`

6 · The server forgets; a database remembers

▶ LIVE A server’s memory isn’t a database

1. Still running from chapter 5 (`servers-5-status-codes`).
2. `POST` two ideas. `GET /ideas` shows them, after the plant waterer.
3. Save `main.py` (no change needed). Terminal 1: `WatchFiles detected changes in 'main.py'. Reloading...`
4. `GET /ideas`: only the plant waterer.

Every request stands alone

- Your function gets **only** the request: the path, the body.
- Anything it needs to remember, it reads from the **database**; anything it changes, it writes back.
- In Lab 5, `ideas` is a table, and your functions read and write it through SQLModel: the next video.

7 · A test is a request and a comparison

A test script, with no test library

servers-7-check/check.py

```
import httpx

URL = "http://localhost:8000"

def check(what, ok):
    print("PASS" if ok else "FAIL", what)

r = httpx.get(URL + "/ideas")
check("GET /ideas answers 200", r.status_code == 200)
check("GET /ideas is a JSON list", isinstance(r.json(), list))

r = httpx.post(URL + "/ideas", json={"title": "Test idea"})
check("POST /ideas answers 201", r.status_code == 201)
check("POST /ideas returns the new id", "id" in r.json())

r = httpx.post(URL + "/ideas", json={})
check("POST with no title answers 422", r.status_code == 422)

r = httpx.get(URL + "/ideas/99999")
check("a missing idea answers 404", r.status_code == 404)
```

▶ LIVE Run it, break it, run it again

1. Terminal 1: the chapter 5 server (`servers-5-status-codes`). Terminal 2: `python check.py`: six PASS.
2. Terminal 1: Ctrl-C, and start the chapter 4 server instead (`servers-4-request-body`): the first take of `get_idea`.
3. Terminal 2: `python check.py`. The last line: `FAIL a missing idea answers 404`.
4. Back to the chapter 5 server, run it: six PASS.

What’s yours, and Lab 5

Yours: the function. Look the idea up, decide 404 or not, shape the answer.

FastAPI’s and uvicorn’s: the port, the loop, routing, turning `"2"` into `2`, checking the body, writing JSON.

Lab 5 endpoint	Chapters
`GET /ideas`, each with a `vote_count`	2 routing, 6 the database
`POST /ideas`	4 a request class, 5 a 201
`GET /ideas/{id}`, with its votes	3 a path parameter, 5 a 404
`POST /ideas/{id}/vote`	3, 4 and 5, all at once

Lab 5 comes with `CONTRACT.md`, the promises, and `check.py`: chapter 7’s script, with more checks.