

Routes, paramètres et body

On construit une mini carte de cocktails, stockée en mémoire. Elle disparaît au redémarrage du serveur : c'est volontaire, on se concentre sur HTTP.

Tout le code de cette page s'ajoute dans **le même fichier** `main.py`, celui de la page précédente. Gardez `fastapi dev main.py` lancé : le serveur redémarre tout seul à chaque enregistrement.

Les imports

```
# main.py, tout en haut : remplace la ligne from fastapi import FastAPI
from fastapi import FastAPI, HTTPException, Response
from pydantic import BaseModel
```

Les données

```
# main.py, juste sous app = FastAPI(...)
cocktails = [
    {"id": 1, "name": "Hello World", "price": 6, "alcohol_free": True},
    {"id": 2, "name": "Segfault Sunrise", "price": 9.5, "alcohol_free": False},
]
next_id = 3
```

Paramètres de query : lister et filtrer

```
GET /v1/cocktails?alcohol_free=true
```

```
# main.py, à la suite
@app.get("/v1/cocktails")
def list_cocktails(alcohol_free: bool | None = None):
    items = cocktails
    if alcohol_free is not None:
        items = [c for c in items if c["alcohol_free"] == alcohol_free]
    return {"items": items}
```

```
curl -i "localhost:8000/v1/cocktails?alcohol_free=true"
curl -i "localhost:8000/v1/cocktails?alcohol_free=peut-etre"
```

Le premier répond `200` avec Hello World seul ; le second répond `422` : « peut-etre » n'est pas un booléen.

Dans l'URL, tout est du texte : `?alcohol_free=true` arrive sous la forme de la chaîne `"true"`. FastAPI la convertit tout seul grâce au type `bool`, et renvoie une erreur si la valeur n'est pas un booléen.

Paramètres de chemin : lire un élément

`GET /v1/cocktails/2`

```
# main.py, à la suite
@app.get("/v1/cocktails/{cocktail_id}")
def get_cocktail(cocktail_id: int):
    for c in cocktails:
        if c["id"] == cocktail_id:
            return c
    raise HTTPException(status_code=404, detail="Cocktail introuvable")
```

```
curl -i localhost:8000/v1/cocktails/2
curl -i localhost:8000/v1/cocktails/42
```

`200` avec Segfault Sunrise, puis `404` : ce cocktail n'existe pas.

Body : créer un élément

`POST /v1/cocktails` doit répondre `201 Created`, avec le header `Location` et la ressource créée.

```
# main.py, à la suite
class CocktailIn(BaseModel):
    name: str
    price: float
    alcohol_free: bool = False
```

```
@app.post("/v1/cocktails", status_code=201)
def create_cocktail(cocktail: CocktailIn, response: Response):
    global next_id
    new = {"id": next_id, **cocktail.model_dump()}
    next_id += 1
    cocktails.append(new)
    response.headers["Location"] = "/v1/cocktails/" + str(new["id"])
    return new
```

```
curl -i localhost:8000/v1/cocktails --json '{"name": "Kernel Panic", "price": 12,
"alcohol_free": false}'
```

201 Created, avec le header `location: /v1/cocktails/3` et le cocktail créé.

Essayez aussi d'envoyer un prix en lettres (`"douze"`), d'oublier le nom, ou d'envoyer un nom vide (`""`) : que se passe-t-il dans chaque cas ?

Ne copiez jamais le body reçu tel quel dans vos données : le client pourrait envoyer un `id`, un `role` ou n'importe quel champ que vous n'avez pas prévu. Ici, `CocktailIn` ne garde que les champs déclarés. Cette faille a un nom, on la reverra au chapitre 8.

Supprimer un élément

`DELETE /v1/cocktails/2` répond `204 No Content`, sans body.

```
# main.py, à la suite
@app.delete("/v1/cocktails/{cocktail_id}", status_code=204)
def delete_cocktail(cocktail_id: int):
    for i, c in enumerate(cocktails):
        if c["id"] == cocktail_id:
            cocktails.pop(i)
            return
    raise HTTPException(status_code=404, detail="Cocktail introuvable")
```

```
curl -i -X DELETE localhost:8000/v1/cocktails/2
curl -i -X DELETE localhost:8000/v1/cocktails/2
```

`204` la première fois ; `404` la seconde, puisque le cocktail n'existe plus.

Le fichier `main.py` complet

```
# main.py

from fastapi import FastAPI, HTTPException, Response
from pydantic import BaseModel

app = FastAPI(title="Mon API")

# Les données, en mémoire
cocktails = [
    {"id": 1, "name": "Hello World", "price": 6, "alcohol_free": True},
    {"id": 2, "name": "Segfault Sunrise", "price": 9.5, "alcohol_free": False},
]
next_id = 3

@app.get("/v1/health")
def health():
    return {"status": "ok"}

@app.get("/v1/cocktails")
def list_cocktails(alcohol_free: bool | None = None):
    items = cocktails
    if alcohol_free is not None:
        items = [c for c in items if c["alcohol_free"] == alcohol_free]
    return {"items": items}

@app.get("/v1/cocktails/{cocktail_id}")
def get_cocktail(cocktail_id: int):
    for c in cocktails:
        if c["id"] == cocktail_id:
            return c
    raise HTTPException(status_code=404, detail="Cocktail introuvable")

class CocktailIn(BaseModel):
    name: str
```

```
price: float
alcohol_free: bool = False
```

```
@app.post("/v1/cocktails", status_code=201)
def create_cocktail(cocktail: CocktailIn, response: Response):
    global next_id
    new = {"id": next_id, **cocktail.model_dump()}
    next_id += 1
    cocktails.append(new)
    response.headers["Location"] = "/v1/cocktails/" + str(new["id"])
    return new


@app.delete("/v1/cocktails/{cocktail_id}", status_code=204)
def delete_cocktail(cocktail_id: int):
    for i, c in enumerate(cocktails):
        if c["id"] == cocktail_id:
            cocktails.pop(i)
            return
    raise HTTPException(status_code=404, detail="Cocktail introuvable")
```

Revision #6

Created 2 October 2026 20:00:53 by Thibaud FRICHET

Updated 14 October 2026 19:32:16 by Thibaud FRICHET