

Validation et codes de retour

À la page précédente, FastAPI refusait déjà un prix en lettres ou un nom manquant, mais avec son propre format d'erreur. Et il acceptait un nom vide ou un prix négatif. Corrigions tout ça.

Règle n°1 d'une API : **ne jamais faire confiance au client**. Tout ce qui arrive (body, paramètres, headers) doit être validé.

Un format d'erreur unique

Toutes les erreurs seront renvoyées au format `application/problem+json` (chapitre 5). On commence par une fonction utilitaire :

```
from fastapi.responses import JSONResponse

def problem(status, title, detail, **extra):
    return JSONResponse(
        {"type": "about:blank", "title": title, "status": status, "detail": detail, **extra},
        status_code=status,
        media_type="application/problem+json",
    )
```

Valider le body

On décrit la forme attendue dans un **schéma** Pydantic : longueurs, valeurs minimales, champs inconnus refusés.

```
from pydantic import BaseModel, ConfigDict, Field

class CocktailIn(BaseModel):
    model_config = ConfigDict(extra="forbid")  # refuse les champs inconnus
    name: str = Field(min_length=1, max_length=50)
    price: float = Field(gt=0)
    alcohol_free: bool = False
```

```

@app.post("/v1/cocktails", status_code=201)
def create_cocktail(cocktail: CocktailIn, response: Response):
    global next_id
    if any(c["name"] == cocktail.name for c in cocktails):
        return problem(409, "Conflict", "Le cocktail " + cocktail.name + " existe déjà.")
    new = {"id": next_id, **cocktail.model_dump()}
    next_id += 1
    cocktails.append(new)
    response.headers["Location"] = "/v1/cocktails/" + str(new["id"])
    return new

```

Quand la fonction s'exécute, `cocktail` a déjà été validé : il ne contient **que** les champs déclarés, avec des valeurs correctes. On peut l'utiliser sans risque.

Les erreurs de FastAPI au bon format

FastAPI valide tout seul, mais renvoie ses erreurs dans son propre format (`{"detail": [...]}`). On le remplace par des **gestionnaires d'exceptions** :

```

from http import HTTPStatus
from fastapi import Request
from fastapi.exceptions import RequestValidationError
from starlette.exceptions import HTTPException as StarletteHTTPException

@app.exception_handler(RequestValidationError)
async def on_validation_error(request: Request, exc: RequestValidationError):
    errors = [
        {"field": ".".join(str(p) for p in e["loc"][1:]), "message": e["msg"]}
        for e in exc.errors()
    ]
    return problem(422, "Données invalides", "La requête contient des erreurs.",
errors=errors)

@app.exception_handler(StarletteHTTPException)
async def on_http_error(request: Request, exc: StarletteHTTPException):
    return problem(exc.status_code, HTTPStatus(exc.status_code).phrase, str(exc.detail))

```

```
@app.exception_handler(Exception)
async def on_crash(request: Request, exc: Exception):
    return problem(500, "Internal Server Error", "Le barman a fait tomber un verre. Réessayez plus tard.")
```

Désormais, `raise HTTPException(status_code=404, detail="Cocktail introuvable")` produit une réponse `problem+json`.

En production, une erreur 500 ne doit **jamais** renvoyer la stack trace au client. On la logge côté serveur, et on renvoie un message neutre.

Quel code renvoyer ?

Situation	Code
Le body n'est pas du JSON	<code>400</code> (FastAPI renvoie <code>422</code> , c'est acceptable)
Pas de clé, ou clé invalide	<code>401</code>
Clé valide, mais action interdite	<code>403</code>
Ressource inexistante (y compris dans le chemin : <code>/cocktails/999/reviews</code>)	<code>404</code>
Conflit avec l'état actuel (doubleton, état qui ne permet pas l'action)	<code>409</code>
JSON valide, mais données incorrectes	<code>422</code>
Création réussie	<code>201</code> + <code>Location</code>
Suppression réussie	<code>204</code>

Dans quel ordre tester ? D'abord l'authentification (`401`), puis l'existence de la ressource (`404`), puis la validité des données (`422`), puis les règles métier (`409`).

FastAPI valide le body **avant** d'exécuter votre fonction : il renverra un `422` avant que vous ayez pu vérifier la clé ou l'existence du cocktail. Pour respecter l'ordre ci-dessus, vérifiez l'authentification avec une **dépendance** (`Depends`), qui s'exécute avant la validation du body.