Définir des endpoints dans FastAPI

Un **endpoint** est une route HTTP associée à une fonction Python, qui traite une requête et renvoie une réponse.

FastAPI lie les fonctions aux routes via des décorateurs comme `@app.get()`, `@app.post()`, etc., correspondant aux méthodes HTTP.

Sommaire

Paramètres dans FastAPI

Les paramètres que vous pouvez récupérer dans une route sont de plusieurs types :

- **Paramètres d'URL** (Path Parameters) : intégrés dans le chemin de l'URL, ils sont obligatoires.
- **Paramètres de requête** (Query Parameters) : passés après le `?` dans l'URL, ils sont optionnels.
- **Paramètres du corps de la requête** (Request Body) : utilisés pour envoyer des données complexes, souvent en JSON, dans les requêtes `POST`, `PUT`, `PATCH`.

> Ces différents types de paramètres permettent de structurer les données envoyées par le client et de les traiter efficacement dans votre application FastAPI.

1. Paramètres d'URL (Path Parameters)

Ce sont des variables intégrées dans le chemin de l'URL. Ils sont obligatoires.

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/users/{user_id}")
def read_user(user_id: int):
    return {"user_id": user_id}
```

- Exemple d'URL : `/users/5`
- `user_id` est automatiquement converti en `int`
- Si l'URL ne contient pas `user_id`, la route ne correspond pas

2. Paramètres de requête (Query Parameters)

Passés après le `?` dans l'URL, ce sont des paramètres optionnels.

```python
@app.get("/users/")
def read_users(active: bool = True, page: int = 1):
    return {"active": active, "page": page}
```

- Exemple d'URL : `/users/?active=false&page=3`
- FastAPI convertit automatiquement les types (`bool`, `int`, etc.)
- Si non spécifiés, les valeurs par défaut sont utilisées (`True` et `1` ici)

3. Paramètres du corps de la requête (Request Body)

Utilisés surtout avec les méthodes `POST`, `PUT`, `PATCH`, ils contiennent souvent des données complexes en JSON.

Pour les manipuler, on utilise des **modèles Pydantic** pour valider et typer ces données.

```
from pydantic import BaseModel

class User(BaseModel):
    name: str
    age: int
    email: str

@app.post("/users/")
def create_user(user: User):
    return {"message": f"User {user.name} created", "user": user}
```

- Le client envoie un JSON dans le corps de la requête :

```
{
  "name": "Alice",
  "age": 30,
  "email": "alice@example.com"
}
```

- FastAPI valide automatiquement que toutes les données sont présentes et du bon type
- La fonction reçoit un objet `User` déjà typé et validé

4. Combiner paramètres

```
@app.put("/users/{user_id}")
def update_user(user_id: int, user: User, notify: bool = False):
    return {
        "user_id": user_id,
        "updated_data": user,
        "notify": notify
    }
```

- `user_id` est extrait de l'URL
- `user` du corps JSON
- `notify` un paramètre de requête optionnel

Exemple d'appel : `PUT /users/3?notify=true` avec le corps JSON `{ "name": "Bob", "age": 35, "email": "bob@example.com" }`