



Documentation automatique avec FastAPI

Sommaire

- Documentation automatique avec FastAPI
 - Sommaire
 - 1. Documentation générée grâce aux schémas Pydantic
 - 2. Interfaces graphiques disponibles
 - 3. Exemple d'annotation d'un endpoint
 - 4. Autres options de documentation
 - 5. Résumé

1. Documentation générée grâce aux schémas Pydantic

FastAPI utilise les modèles Pydantic (comme `UserCreate`, `UserRead`, etc.) pour :

- **Générer automatiquement la structure des données attendues en entrée** (exemple : corps JSON)
- **Afficher les formats des données retournées** (réponses)
- **Valider les données envoyées par le client** selon ces schémas

Cela permet d'avoir une documentation précise, toujours à jour et cohérente avec le code.

2. Interfaces graphiques disponibles

FastAPI génère deux interfaces interactives pour tester et documenter l'API :

Interface	URL par défaut	Description
Swagger UI	<code>/docs</code>	Documentation interactive et visuelle très complète, avec possibilité de tester les requêtes
ReDoc	<code>/redoc</code>	Documentation plus détaillée, orientée lecture et compréhension

Représentation visuelle :

The screenshot shows two side-by-side views of the FastAPI documentation. On the left is the Swagger UI interface, which has a light blue header and sidebar. It lists endpoints for 'Utilisateurs' (Users) and 'default'. Under 'Utilisateurs', there are three buttons: 'GET /users/' (List Users), 'POST /users/' (Créer un utilisateur - Create user), and 'GET /users/{user_id}' (Read User). Under 'default', there are three buttons: 'POST /users/' (Créer un utilisateur - Create user), 'GET /users/{user_id}' (Read User), and 'GET /users/{user_id}' (Get User). Below these are sections for 'Schemas' and error handling. The right side shows the ReDoc interface, which has a dark grey header and sidebar. It also lists 'Utilisateurs' and 'default' sections. Under 'Utilisateurs', it shows 'List Users' and 'Responses' (with a green button for '200 Successful Response'). Under 'default', it shows 'Créer un utilisateur' (Create user) with a description and two required fields: 'name' (nom complet de l'utilisateur) and 'email' (adresse email valide). Both interfaces include a 'Download' button for the OpenAPI specification and a 'Download' button for the API documentation itself.

docs

redoc

3. Exemple d'annotation d'un endpoint

```
from fastapi import FastAPI, Path, Query, Body

app = FastAPI()

@app.post(
    "/users/",
    response_model=UserRead,
    summary="Créer un utilisateur"
)
def create_user(
    user:UserCreate=Body(..., example={"name":"Alice", "email":"alice@mail.com"})
):
    """
    Crée un nouvel utilisateur dans la base.

    - **name**: nom complet de l'utilisateur
    - **email**: adresse email valide
    """
    # logique de création ici...
    return user # Exemple simplifié
```

- `response_model=UserRead` : indique le schéma attendu en réponse (sert à générer la doc et la validation)
- `summary` : résumé affiché dans Swagger UI
- `Body(..., example=...)` : exemple concret affiché dans la doc, aide à comprendre le format attendu
- La docstring de la fonction est utilisée comme description détaillée visible dans Swagger UI

POST /users/ Créer un utilisateur

Crée un nouvel utilisateur dans la base.

- **name**: nom complet de l'utilisateur
- **email**: adresse email valide

Parameters

No parameters

Request body required

[Example Value](#) | [Schema](#)

```
{  
    "name": "Alice",  
    "email": "alice@mail.com"  
}
```

4. Autres options de documentation

- **Descriptions sur les paramètres** : on peut ajouter des descriptions dans les paramètres **Path**, **Query**, ou **Body** :

```
@app.get("/users/{user_id}")
def read_user(
    user_id: int = Path(..., description="L'ID unique de l'utilisateur"),
    active: bool | None = Query(None, description="Filtrer selon l'état actif")
):
    ...
```

GET /users/{user_id}/read Read User

Parameters

Name	Description
user_id * required integer (path)	L'ID unique de l'utilisateur user_id
active boolean (boolean null) (query)	Filtrer selon l'état actif -- ▾

- **Tags** : regrouper les endpoints par catégories dans la doc

```
app = FastAPI()

@app.get("/users/", tags=["Utilisateurs"])
def list_users():
    ...
```

Utilisateurs

GET /users/ List Users

- **Réponses personnalisées** : on peut documenter les codes HTTP attendus

```
from fastapi.responses import JSONResponse
from fastapi import status

@app.get(
    "/users/{user_id}",
    responses={
        404: {"description": "Utilisateur non trouvé"},
        200: {"description": "Utilisateur trouvé"}
    }
)
```

```
def get_user(user_id: int):
    ...
```

Responses

Code	Description
200	<p>Utilisateur trouvé</p> <p>Media type</p> <p>application/json ▾</p> <p>Controls Accept header.</p> <p>Example Value Schema</p> <pre>"string"</pre>
404	Utilisateur non trouvé
422	Validation Error

5. Résumé

Grâce aux schémas Pydantic et aux annotations sur les endpoints, FastAPI produit une documentation :

- **automatique, précise et interactive**
- **facile à maintenir** (car directement liée au code)
- **accessible aux développeurs et consommateurs d'API**