Démarrage

Le fichier **FastAPI** le plus simple possible pourrait ressembler à cela :

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

Copiez ce code dans un fichier nommé `main.py`.

Démarrez le serveur :

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
<span style="color: green;">INFO</span>:     Started reloader process [28720]
<span style="color: green;">INFO</span>:     Started server process [28722]
<span style="color: green;">INFO</span>:     Waiting for application startup.
<span style="color: green;">INFO</span>:     Application startup complete.
```

!!! note La commande `uvicorn main:app` fait référence à :

```
* `main` : le fichier `main.py` (le module Python).
* `app` : l'objet créé dans `main.py` via la ligne `app = FastAPI()`.
* `--reload` : l'option disant à uvicorn de redémarrer le serveur à chaque changement
du code. À ne pas utiliser en production !
```

Vous devriez voir dans la console, une ligne semblable à la suivante :

```
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Cette ligne montre l'URL par laquelle l'app est actuellement accessible, sur votre machine locale.

Allez voir le résultat

Ouvrez votre navigateur à l'adresse http://127.0.0.1:8000.

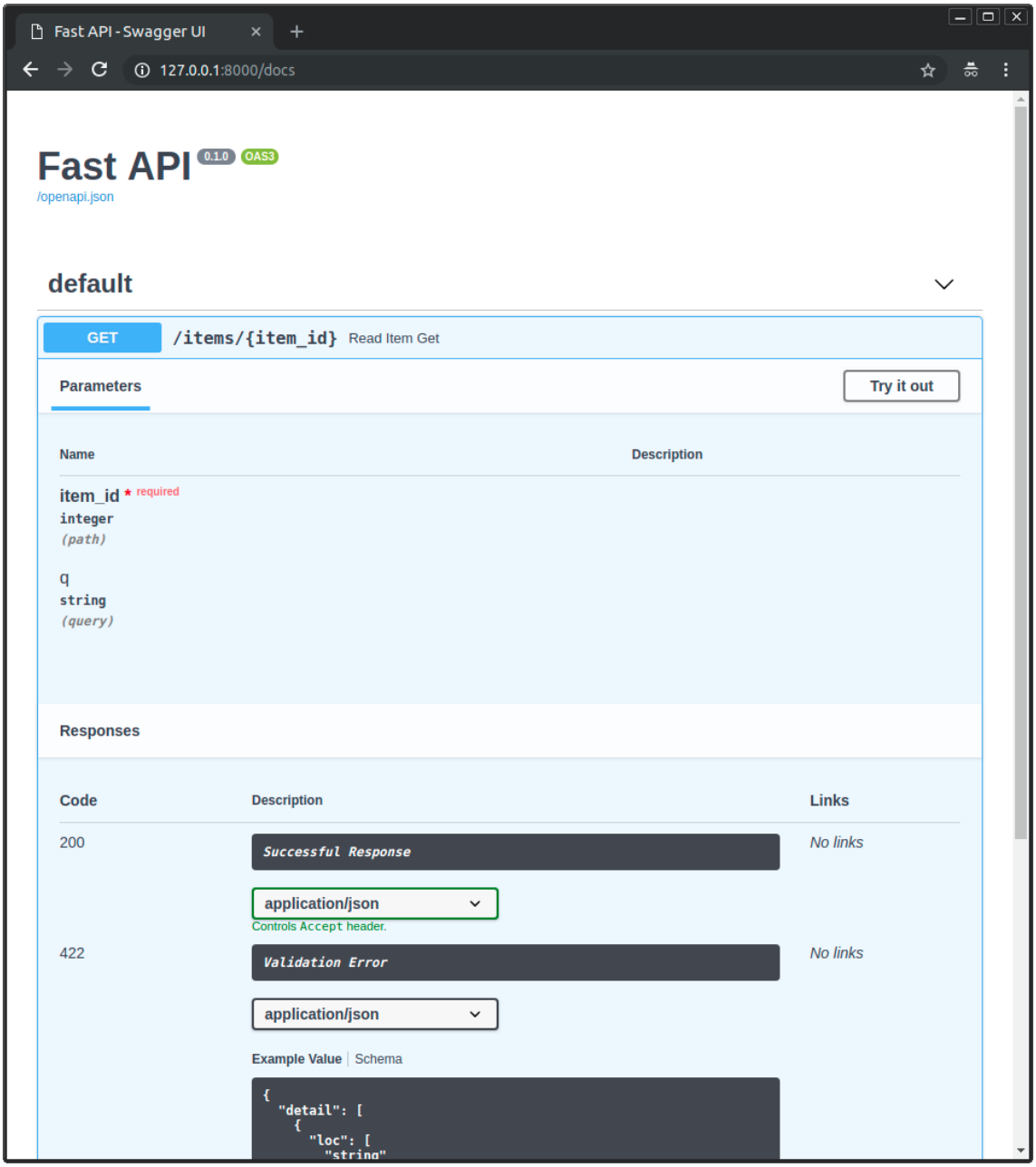
Vous obtiendrez cette réponse JSON :

```
{"message": "Hello World"}
```

Documentation interactive de l'API

Rendez-vous sur http://127.0.0.1:8000/docs.

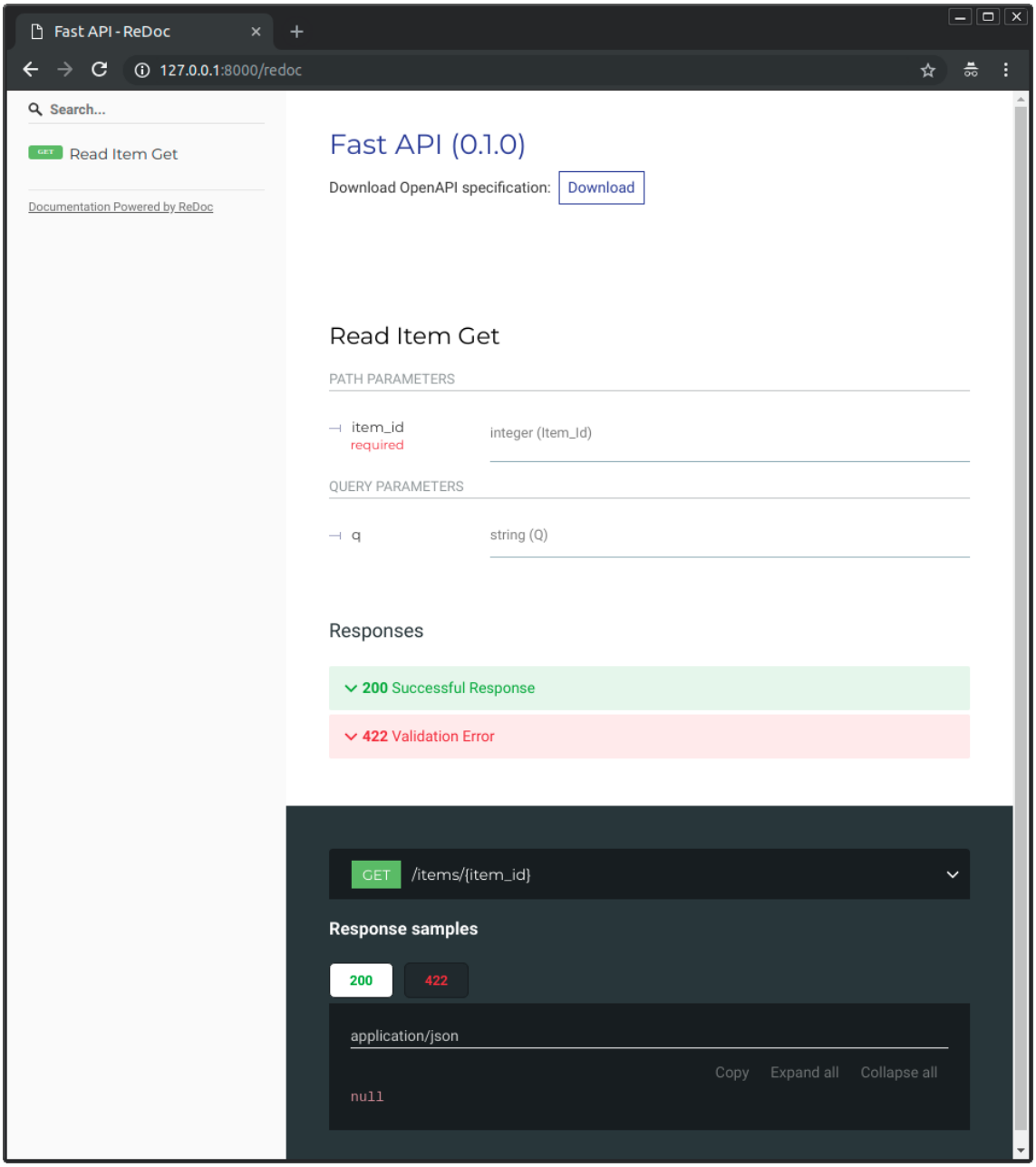
Vous verrez la documentation interactive de l'API générée automatiquement (via Swagger UI) :



Documentation alternative

Ensuite, rendez-vous sur http://127.0.0.1:8000/redoc.

Vous y verrez la documentation alternative (via ReDoc) :



OpenAPI

FastAPI génère un "schéma" contenant toute votre API dans le standard de définition d'API **OpenAPI**.

"Schéma"

Un "schéma" est une définition ou une description de quelque chose. Pas le code qui l'implémente, uniquement une description abstraite.

"Schéma" d'API

Ici, OpenAPI est une spécification qui dicte comment définir le schéma de votre API.

Le schéma inclut les chemins de votre API, les paramètres potentiels de chaque chemin, etc.

"Schéma" de données

Le terme "schéma" peut aussi faire référence à la forme de la donnée, comme un contenu JSON.

Dans ce cas, cela signifierait les attributs JSON, ainsi que les types de ces attributs, etc.

OpenAPI et JSON Schema

OpenAPI définit un schéma d'API pour votre API. Il inclut des définitions (ou "schémas") de la donnée envoyée et reçue par votre API en utilisant **JSON Schema**, le standard des schémas de données JSON.

Allez voir `openapi.json`

Si vous êtes curieux d'à quoi ressemble le schéma brut **OpenAPI**, **FastAPI** génère automatiquement un (schéma) JSON avec les descriptions de toute votre API.

Vous pouvez le voir directement à cette adresse : http://127.0.0.1:8000/openapi.json.

Le schéma devrait ressembler à ceci :

```
{
    "openapi": "3.0.2",
    "info": {
        "title": "FastAPI",
        "version": "0.1.0"
    },
    "paths": {
        "/items/": {
            "get": {
                "responses": {
                    "200": {
                        "description": "Successful Response",
                        "content": {
                            "application/json": {


...
```

À quoi sert OpenAPI

Le schéma **OpenAPI** est ce qui alimente les deux systèmes de documentation interactive.

Et il existe des dizaines d'alternatives, toutes basées sur **OpenAPI**. Vous pourriez facilement ajouter n'importe laquelle de ces alternatives à votre application **FastAPI**.

Vous pourriez aussi l'utiliser pour générer du code automatiquement, pour les clients qui communiquent avec votre API. Comme par exemple, des applications frontend, mobiles ou IOT.

Récapitulatif, étape par étape

Étape 1 : import `FastAPI`

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

`FastAPI` est une classe Python qui fournit toutes les fonctionnalités nécessaires au lancement de votre API.

!!! note "Détails techniques" `FastAPI` est une classe héritant directement de `Starlette`.

```
Vous pouvez donc aussi utiliser toutes les fonctionnalités de <a
href="https://www.starlette.io/" class="external-link" target="_blank">Starlette</a>
depuis `FastAPI`.
```

Étape 2 : créer une "instance" `FastAPI`

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

Ici la variable `app` sera une "instance" de la classe `FastAPI`.

Ce sera le point principal d'interaction pour créer toute votre API.

Cette `app` est la même que celle à laquelle fait référence `uvicorn` dans la commande :

```
$ uvicorn main:app --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Si vous créez votre app avec :

```
{!../../../docs_src/first_steps/tutorial002.py!}
```

Et la mettez dans un fichier `main.py`, alors vous appeleriez `uvicorn` avec :

```
$ uvicorn main:my_awesome_api --reload

<span style="color: green;">INFO</span>:     Uvicorn running on
http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Étape 3: créer une *opération de chemin*

Chemin

Chemin, ou "path" fait référence ici à la dernière partie de l'URL démarrant au premier `/`.

Donc, dans un URL tel que :

```
https://example.com/items/foo
```

...le "path" serait :

```
/items/foo
```

!!! info Un chemin, ou "path" est aussi souvent appelé route ou "endpoint".

Opération

"Opération" fait référence à une des "méthodes" HTTP.

Une de :

- `POST`
- `GET`
- `PUT`
- `DELETE`

...ou une des plus exotiques :

- `OPTIONS`
- `HEAD`
- `PATCH`
- `TRACE`

Dans le protocol HTTP, vous pouvez communiquer avec chaque chemin en utilisant une (ou plus) de ces "méthodes".

En construisant des APIs, vous utilisez généralement ces méthodes HTTP spécifiques pour effectuer une action précise.

Généralement vous utilisez :

- `POST` : pour créer de la donnée.
- `GET` : pour lire de la donnée.
- `PUT` : pour mettre à jour de la donnée.
- `DELETE` : pour supprimer de la donnée.

Donc, dans **OpenAPI**, chaque méthode HTTP est appelée une "opération".

Nous allons donc aussi appeler ces dernières des "**opérations**".

Définir un *décorateur d'opération de chemin*

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

Le `@app.get("/")` dit à **FastAPI** que la fonction en dessous est chargée de gérer les requêtes qui vont sur :

- le chemin `/`
- en utilisant une opération `get`

!!! info " `@décorateur` Info" Cette syntaxe `@something` en Python est appelée un "décorateur".

```
Vous la mettez au dessus d'une fonction. Comme un joli chapeau décoratif (j'imagine que ce terme vient de là 🤷).

Un "décorateur" prend la fonction en dessous et en fait quelque chose.

Dans notre cas, ce décorateur dit à **FastAPI** que la fonction en dessous correspond au **chemin** `/` avec l'**opération** `get`.

C'est le "**décorateur d'opération de chemin**".
```

Vous pouvez aussi utiliser les autres opérations :

- `@app.post()`
- `@app.put()`
- `@app.delete()`

Tout comme celles les plus exotiques :

- `@app.options()`
- `@app.head()`
- `@app.patch()`
- `@app.trace()`

!!! tip "Astuce" Vous êtes libres d'utiliser chaque opération (méthode HTTP) comme vous le désirez.

```
**FastAPI** n'impose pas de sens spécifique à chacune d'elle.

Les informations qui sont présentées ici forment une directive générale, pas des obligations.

Par exemple, quand l'on utilise **GraphQL**, toutes les actions sont effectuées en utilisant uniquement des opérations `POST`.
```

Étape 4 : définir la fonction de chemin.

Voici notre "**fonction de chemin**" (ou fonction d'opération de chemin) :

- **chemin** : `/`.
- **opération** : `get`.
- **fonction** : la fonction sous le "décorateur" (sous `@app.get("/")`).

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

C'est une fonction Python.

Elle sera appelée par **FastAPI** quand une requête sur l'URL `/` sera reçue via une opération `GET`.

Ici, c'est une fonction asynchrone (définie avec `async def`).

Vous pourriez aussi la définir comme une fonction classique plutôt qu'avec `async def` :

```
{!../../../docs_src/first_steps/tutorial003.py!}
```

!!! note Si vous ne connaissez pas la différence, allez voir la section Concurrence : *"Vous êtes pressés ?"*{.internal-link target=_blank}.

Étape 5 : retourner le contenu

```
{!../../../docs_src/first_steps/tutorial001.py!}
```

Vous pouvez retourner un dictionnaire (`dict`), une liste (`list`), des valeurs seules comme des chaines de caractères (`str`) et des entiers (`int`), etc.

Vous pouvez aussi retourner des models **Pydantic** (qui seront détaillés plus tard).

Il y a de nombreux autres objets et modèles qui seront automatiquement convertis en JSON. Essayez d'utiliser vos favoris, il est fort probable qu'ils soient déjà supportés.

Récapitulatif

- Importez `FastAPI` .
- Créez une instance d' `app` .
- Ajoutez une **décorateur d'opération de chemin** (tel que `@app.get("/")`).
- Ajoutez une **fonction de chemin** (telle que `def root(): ...` comme ci-dessus).
- Lancez le serveur de développement (avec `uvicorn main:app --reload`).