FastAPI - Premiers pas avec FastAPI

45 minutes Beginner

DataScientest • com

APIs avec FastAPI

3. Introduction à FastAPI

a. La librairie FastAPI

La librairie FastAPI est une librairie très intéressante pour développer des APIs avec Python. En effet, les APIs sont relativement rapides par rapport à d'autres frameworks Python. De plus, FastAPI permet d'implémenter facilement une documentation ainsi que des contraintes de type sur les données.

b. Première implémentation

Dans cette partie, nous allons voir les principes de base de FastAPI. La première consiste à installer les librairies `fastapi` et `uvicorn`. `uvicorn` est une librairie qui permet de lancer le serveur créé par FastAPI.

> Les librairies Python de création d'API utilisent généralement un serveur autre pour lancer l'API. Par exemple, on peut lancer une API Flask sans uvicorn mais ce n'est en général pas recommandé (cf le message de lancement).

Installez les librairies en utilisant la commande suivante

```
pip3 install fastapi uvicorn

```

Pour créer une API, nous allons devoir instancier la classe `FastAPI` du package `fastapi`.

```
2
3  api = FastAPI()
4
5  @api.get('/')
6  def get_index():
7      return {'data': 'hello world'}
8
9
```

> Une fois que ce fichier est sauvegardé, lancez l'API **dans une autre console** en exécutant la commande suivante

```
1  uvicorn main:api --reload
2
```

Ici, on précise le fichier `main` et le nom de l'API à lancer à l'intérieur de ce fichier: `api`. L'argument `--reload` permet de mettre à jour automatiquement l'API lorsqu'on effectue des changements du fichier source. Dans la console, on doit observer la ligne suivante:

```
1  INFO: Uvicorn running on http://127.0.0.1:
2
```

Cette ligne nous donne l'adresse à laquelle l'API fonctionne.

> **Dans une autre console**, lancez la commande suivante pour faire une requête sur le endpoint `/`

```
1  curl -X GET http://127.0.0.1:8000/
2
```

Le résultat correspond bien à ce qu'on a passé comme valeur à `return`:

```
1  {"data": "hello world"}
2
```

> Exécutez la commande suivante pour afficher les en-têtes de la réponse:

```
1  curl -X GET -i http://127.0.0.1:8000/
2
```

On remarque que le contenu retourné est de type `application/json`: on n'a pas précisé cet argument mais FastAPI par défaut, renvoie des données au format `json`.

> Sans arrêter l'API, modifiez le fichier `main.py` en remplaçant la fonction `get_index` par les lignes suivantes:

```
1  def get_index():
2      return "Hello world"
3
```

Le type du contenu est toujours `application/json`.

Dans le code que nous avons exécuté on peut constater la présence d'un décorateur: `@api.get('/')`. Ce décorateur permet de préciser une route, c'est-à-dire un endpoint ainsi qu'une méthode. Ainsi, la fonction qui est décorée par cette ligne s'exécutera lorsqu'une requête de type `GET` est effectuée au endpoint `/`.

Cette façon de gérer les endpoints ainsi que les méthodes permet de voir facilement quelle fonction est appelée à quel moment et dans quelles conditions.

On peut bien sûr utiliser différentes méthodes et préciser différents endpoints.

Modifiez le fichier `main.py` en y mettant les routes suivantes

```
7
8  @api.get('/other')
9  def get_other():
10     return {
11         'method': 'get',
12         'endpoint': '/other'
13     }
14
15 @api.post('/')
16 def post_index():
17     return {
18         'method': 'post',
19         'endpoint': '/'
20         }
21
22 @api.delete('/')
23 def delete_index():
24     return {
25         'method': 'delete',
26         'endpoint': '/'
27         }
28
29 @api.put('/')
30 def put_index():
31     return {
32         'method': 'put',
33         'endpoint': '/'
34         }
35
36 @api.patch('/')
37 def patch_index():
38     return {
39         'method': 'patch',
40         'endpoint': '/'
41         }
42
```

Exécutez les commandes suivantes pour tester toutes les routes

```
1  # GET at /
2  curl -X GET -i http://127.0.0.1:8000/
```

```
curl -X PUT -i http://127.0.0.1:8000/
# DELETE at /
curl -X DELETE -i http://127.0.0.1:8000
# PATCH at /
curl -X PATCH -i http://127.0.0.1:8000/
# GET at /other
curl -X GET -i http://127.0.0.1:8000/ot
```

En quelques lignes, on a pu créer des routes pour différentes méthodes et différents endpoints. Tentons à présent d'interroger un endpoint qui n'existe pas:

> Exécutez la commande suivante dans une console

```
curl -X GET -i http://127.0.0.1:8000/no_wh

```

On obtient bien une erreur de type `404 Not Found` avec un contenu au format json: `{"detail": "Not found"}`. FastAPI gère donc assez gracieusement ces erreurs de routage.

c. Documentation

Un des enjeux importants des APIs est de fournir une documentation précise qui permet une utilisation simple de l'API. FastAPI présente l'avantage de générer automatiquement cette documentation.

En utilisant un tunnel entre le port `8000` de la machine distante et le port `8000` de la machine locale, on peut ouvrir l'API dans un navigateur web.

> Créez ce tunnel et ouvrez un navigateur web à l'adresse `http://localhost:8000/` (ou en changeant le port si vous avez décidé de faire suivre les données sur un autre port)

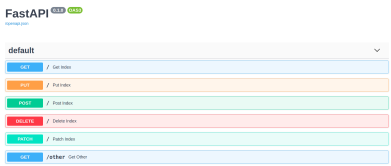
On devrait récupérer le résultat de la requête `GET` au endpoint `/`, c'est-à-dire, le résultat de la fonction `get_index`:

```
{ "method": "get", "endpoint": "/" }

```

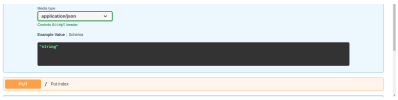
> Ouvrez le endpoint `docs` dans le navigateur: `http://localhost:8000/docs`

Vous devriez arriver sur cette interface:



Il s'agit de l'interface OpenAPI (anciennement Swagger). Cette interface permet de voir facilement les endpoints et les méthodes acceptées.

> Cliquez sur la méthode `GET` du endpoint `/`.



On peut cliquer sur le bouton `Try it out` puis `execute` pour lancer une requête `GET` sur le endpoint `/`. La réponse est retranscrite:

Cette interface nous donne la réponse, les en-têtes de la réponse. De plus, on peut voir la requête `curl` associée à l'essai qu'on vient de faire:

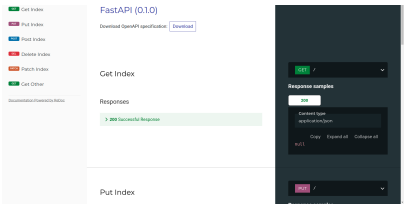
```
curl -X 'GET' \
  'http://127.0.0.1:8000/' \
  -H 'accept: application/json'


```

On a donc un outil très utile pour faire des essais de notre API (à noter que les requêtes sont ici exprimées comme si elles étaient effectuées sur la machine hôte). FastAPI propose une autre version de cette interface au endpoint `/redoc`.

> Ouvrez l'URL `http://localhost:8000/redoc`

Vous devriez arriver sur cette interface:



Cette interface est générée par ReDoc. Enfin, nous pouvons nous rendre au endpoint `/openapi.json`. On retrouvera la déclaration de l'API utilisée par ReDoc et OpenAPI pour générer les documentations:

> Ouvrez l'URL `http://localhost:8000/openapi.json`

On devrait obtenir le json ci-dessous: ici, on l'a formatté pour le rendre plus lisible

```
                "application/json": {
                  "schema": {}
                }
              }
            }
          }
        },
        "put": {
          "summary": "Put Index",
          "operationId": "put_index__put"
          "responses": {
```

```
30              "application/json": {
31                "schema": {}
32              }
33            }
34          }
35        }
36      },
37      "post": {
38        "summary": "Post Index",
39        "operationId": "post_index__pos
40        "responses": {
41          "200": {
42            "description": "Successful
43            "content": {
44              "application/json": {
45                "schema": {}
46              }
47            }
48          }
49        }
50      },
51      "delete": {
52        "summary": "Delete Index",
53        "operationId": "delete_index__c
54        "responses": {
55          "200": {
56            "description": "Successful
57            "content": {
58              "application/json": {
59                "schema": {}
60              }
61            }
62          }
63        }
64      },
65      "patch": {
66        "summary": "Patch Index",
67        "operationId": "patch_index__pa
68        "responses": {
69          "200": {
70            "description": "Successful
71            "content": {
72              "application/json": {
73                "schema": {}
74              }
75            }
76          }
77        }
78      }
79    },
80    "/other": {
81      "get": {
82        "summary": "Get Other",
83        "operationId": "get_other_other
84        "responses": {
85          "200": {
86            "description": "Successful
87            "content": {
88              "application/json": {
```

```
92                 }
93             }
94         }
95     }
96 }
97 }
98
```

Dans la suite, nous verrons comment apporter plus d'informations à nos documentations.

Validated