Paramètres de requête

Quand vous déclarez des paramètres dans votre fonction de chemin qui ne font pas partie des paramètres indiqués dans le chemin associé, ces paramètres sont automatiquement considérés comme des paramètres de "requête".

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

La partie appelée requête (ou **query**) dans une URL est l'ensemble des paires clés-valeurs placées après le `?`, séparées par des `&`.

Par exemple, dans l'URL :

```
http://127.0.0.1:8000/items/?skip=0&limit=10
```

...les paramètres de requête sont :

- `skip` : avec une valeur de `0`
- `limit` : avec une valeur de `10`

Faisant partie de l'URL, ces valeurs sont des chaînes de caractères (`str`).

Mais quand on les déclare avec des types Python (dans l'exemple précédent, en tant qu' `int`), elles sont converties dans les types renseignés.

Toutes les fonctionnalités qui s'appliquent aux paramètres de chemin s'appliquent aussi aux paramètres de requête :

- Support de l'éditeur : vérification d'erreurs, auto-complétion, etc.
- "Parsing" de données.
- Validation de données.
- Annotations d'API et documentation automatique.

Valeurs par défaut

Les paramètres de requête ne sont pas une partie fixe d'un chemin, ils peuvent être optionnels et avoir des valeurs par défaut.

Dans l'exemple ci-dessus, ils ont des valeurs par défaut qui sont `skip=0` et `limit=10`.

Donc, accéder à l'URL :

```
http://127.0.0.1:8000/items/
```

serait équivalent à accéder à l'URL :

```
http://127.0.0.1:8000/items/?skip=0&limit=10
```

Mais si vous accédez à, par exemple :

```
http://127.0.0.1:8000/items/?skip=20
```

Les valeurs des paramètres de votre fonction seront :

- `skip=20` : car c'est la valeur déclarée dans l'URL.

- `limit=10` : car `limit` n'a pas été déclaré dans l'URL, et que la valeur par défaut était `10` .

Paramètres optionnels

De la même façon, vous pouvez définir des paramètres de requête comme optionnels, en leur donnant comme valeur par défaut `None` :

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

Ici, le paramètre `q` sera optionnel, et aura `None` comme valeur par défaut.

!!! check "Remarque" On peut voir que **FastAPI** est capable de détecter que le paramètre de chemin `item_id` est un paramètre de chemin et que `q` n'en est pas un, c'est donc un paramètre de requête.

!!! note **FastAPI** saura que `q` est optionnel grâce au `=None` .

```
Le `Optional` dans `Optional[str]` n'est pas utilisé par **FastAPI** (**FastAPI** n'en utilisera que la partie `str`), mais il servira tout de même à votre editeur de texte pour détecter des erreurs dans votre code.
```

Conversion des types des paramètres de requête

Vous pouvez aussi déclarer des paramètres de requête comme booléens (`bool`), **FastAPI** les convertira :

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

Avec ce code, en allant sur :

```
http://127.0.0.1:8000/items/foo?short=1
```

ou

```
http://127.0.0.1:8000/items/foo?short=True
```

ou

```
http://127.0.0.1:8000/items/foo?short=true
```

ou

```
http://127.0.0.1:8000/items/foo?short=on
```

ou

```
http://127.0.0.1:8000/items/foo?short=yes
```

ou n'importe quelle autre variation de casse (tout en majuscules, uniquement la première lettre en majuscule, etc.), votre fonction considérera le paramètre `short` comme ayant une valeur booléenne à `True` . Sinon la valeur sera à `False` .

Multiples paramètres de chemin et de requête

Vous pouvez déclarer plusieurs paramètres de chemin et paramètres de requête dans la même fonction, **FastAPI** saura comment les gérer.

Et vous n'avez pas besoin de les déclarer dans un ordre spécifique.

Ils seront détectés par leurs noms :

```
{!../../../docs_src/query_params/tutorial004.py!}
```

Paramètres de requête requis

Quand vous déclarez une valeur par défaut pour un paramètre qui n'est pas un paramètre de chemin (actuellement, nous n'avons vu que les paramètres de requête), alors ce paramètre n'est pas requis.

Si vous ne voulez pas leur donner de valeur par défaut mais juste les rendre optionnels, utilisez `None` comme valeur par défaut.

Mais si vous voulez rendre un paramètre de requête obligatoire, vous pouvez juste ne pas y affecter de valeur par défaut :

```
{!../../../docs_src/query_params/tutorial005.py!}
```

Ici le paramètre `needy` est un paramètre requis (ou obligatoire) de type `str`.

Si vous ouvrez une URL comme :

```
http://127.0.0.1:8000/items/foo-item
```

...sans ajouter le paramètre requis `needy`, vous aurez une erreur :

```
{
    "detail": [
        {
            "loc": [
                "query",
                "needy"
            ],
            "msg": "field required",
            "type": "value_error.missing"
        }
    ]
}
```

La présence de `needy` étant nécessaire, vous auriez besoin de l'insérer dans l'URL :

```
http://127.0.0.1:8000/items/foo-item?needy=sooooneedy
```

...ce qui fonctionnerait :

```
{
    "item_id": "foo-item",
    "needy": "sooooneedy"
}
```

Et bien sur, vous pouvez définir certains paramètres comme requis, certains avec des valeurs par défaut et certains entièrement optionnels :

```
{!../../../docs_src/query_params/tutorial006.py!}
```

Ici, on a donc 3 paramètres de requête :

- `needy`, requis et de type `str`.
- `skip`, un `int` avec comme valeur par défaut `0`.
- `limit`, un `int` optionnel.

!!! tip "Astuce" Vous pouvez utiliser les `Enum`s de la même façon qu'avec les Paramètres de chemin{.internal-link target=_blank}.