Parâmetros da rota da URL

Você pode declarar os "parâmetros" ou "variáveis" com a mesma sintaxe utilizada pelo formato de strings do Python:

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

O valor do parâmetro que foi passado à `item_id` será passado para a sua função como o argumento `item_id`.

Então, se você rodar este exemplo e for até http://127.0.0.1:8000/items/foo, você verá a seguinte resposta:

```
{"item_id":"foo"}
```

Parâmetros da rota com tipos

Você pode declarar o tipo de um parâmetro na função usando as anotações padrões do Python:

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

Nesse caso, `item_id` está sendo declarado como um `int`.

!!! Check Verifique Isso vai dar à você suporte do seu editor dentro das funções, com verificações de erros, autocompletar, etc.

Conversão de dados

Se você rodar esse exemplo e abrir o seu navegador em http://127.0.0.1:8000/items/3, você verá a seguinte resposta:

```
{"item_id":3}
```

!!! Verifique Observe que o valor recebido pela função (e também retornado por ela) é `3`, como um Python `int`, não como uma string `"3"`.

```
Então, com essa declaração de tipo, o **FastAPI** dá pra você um <abbr
title="convertendo a string que veio do request HTTP em um dado Python">"parsing"
</abbr> automático no request .
```

Validação de dados

Mas se você abrir o seu navegador em http://127.0.0.1:8000/items/foo, você verá um belo erro HTTP:

```
{
    "detail": [
        {
            "loc": [
                "path",
                "item_id"
            ],
            "msg": "value is not a valid integer",
```

```
            "type": "type_error.integer"
        }
    ]
}
```

devido ao parâmetro da rota `item_id` ter um valor `"foo"`, que não é um `int`.

O mesmo erro apareceria se você tivesse fornecido um `float` ao invés de um `int`, como em:
http://127.0.0.1:8000/items/4.2

!!! Verifique Então, com a mesma declaração de tipo do Python, o **FastAPI** dá pra você validação de dados.

```
Observe que o erro também mostra claramente o ponto exato onde a validação não passou.

Isso é incrivelmente útil enquanto se desenvolve e debuga o código que interage com a sua API.
```

Documentação

Quando você abrir o seu navegador em http://127.0.0.1:8000/docs, você verá de forma automática e interativa a documtação da API como:

!!! check Novamente, apenas com a mesma declaração de tipo do Python, o **FastAPI** te dá de forma automática e interativa a documentação (integrada com o Swagger UI).

```
Veja que o parâmetro de rota está declarado como sendo um inteiro (int).
```

Beneficios baseados em padrões, documentação alternativa

Devido ao schema gerado ser o padrão do OpenAPI, existem muitas ferramentas compatíveis.

Por esse motivo, o próprio **FastAPI** fornece uma API alternativa para documentação (utilizando ReDoc), que você pode acessar em http://127.0.0.1:8000/redoc:

Da mesma forma, existem muitas ferramentas compatíveis. Incluindo ferramentas de geração de código para muitas linguagens.

Pydantic

Toda a validação de dados é feita por baixo dos panos pelo Pydantic, então você tem todos os benefícios disso. E assim você sabe que está em boas mãos.

Você pode usar as mesmas declarações de tipo com `str`, `float`, `bool` e muitos outros tipos complexos de dados.

Vamos explorar muitos destes tipos nos próximos capítulos do tutorial.

A ordem importa

Quando você cria operações de rota, você pode se deparar com situações onde você pode ter uma rota fixa.

Algo como `/users/me` por exemplo, digamos que essa rota seja utilizada para pegar dados sobre o usuário atual.

E então você pode ter também uma rota `/users/{user_id}` para pegar dados sobre um usuário específico associado a um ID de usuário.

Porque as operações de rota são avaliadas em ordem, você precisa ter certeza que a rota para `/users/me` está sendo declarado antes da rota `/users/{user_id}`:

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

Caso contrário, a rota para `/users/{user_id}` coincidiria também para `/users/me`, "pensando" que estaria recebendo o parâmetro `user_id` com o valor de `"me"`.

Valores predefinidos

Se você tem uma operação de rota que recebe um parâmetro da rota, mas que você queira que esses valores possíveis do parâmetro da rota sejam predefinidos, você pode usar `Enum` padrão do Python.

Criando uma classe `Enum`

Importe `Enum` e crie uma sub-classe que herde de `str` e de `Enum`.

Por herdar de `str` a documentação da API vai ser capaz de saber que os valores devem ser do tipo `string` e assim ser capaz de mostrar eles corretamente.

Assim, crie atributos de classe com valores fixos, que serão os valores válidos disponíveis.

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

!!! informação Enumerations (ou enums) estão disponíveis no Python desde a versão 3.4.

!!! dica Se você está se perguntando, "AlexNet", "ResNet", e "LeNet" são apenas nomes de modelos de Machine Learning (aprendizado de máquina).

Declare um *parâmetro de rota*

Logo, crie um *parâmetro de rota* com anotações de tipo usando a classe enum que você criou (`ModelName`):

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

Revise a documentação

Visto que os valores disponíveis para o parâmetro da rota estão predefinidos, a documentação interativa pode mostrar esses valores de uma forma bem legal:

Trabalhando com os *enumeration* do Python

O valor do *parâmetro da rota* será um *membro de enumeration*.

Compare *membros de enumeration*

Você pode comparar eles com o *membro de enumeration* no enum `ModelName` que você criou:

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

Obtenha o *valor de enumerate*

Você pode ter o valor exato de enumerate (um `str` nesse caso) usando `model_name.value`, ou em geral, `your_enum_member.value`:

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

!!! conselho Você também poderia acessar o valor `"lenet"` com `ModelName.lenet.value`

Retorne *membros de enumeration*

Você pode retornar *membros de enum* da sua *rota de operação*, em um corpo JSON aninhado (por exemplo um `dict`).

Eles serão convertidos para o seus valores correspondentes (strings nesse caso) antes de serem retornados ao cliente:

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

No seu cliente você vai obter uma resposta JSON como:

```
{
  "model_name": "alexnet",
  "message": "Deep Learning FTW!"
}
```

Parâmetros de rota que contém caminhos

Digamos que você tenha uma *operação de rota* com uma rota `/files/{file_path}`.

Mas você precisa que o próprio `file_path` contenha uma *rota*, como `home/johndoe/myfile.txt`.

Então, a URL para este arquivo deveria ser algo como: `/files/home/johndoe/myfile.txt`.

Suporte do OpenAPI

O OpenAPI não suporta uma maneira de declarar um *parâmetro de rota* que contenha uma *rota* dentro, dado que isso poderia levar a cenários que são difíceis de testar e definir.

No entanto, você pode fazer isso no **FastAPI**, usando uma das ferramentas internas do Starlette.

A documentação continuaria funcionando, ainda que não adicionaria nenhuma informação dizendo que o parâmetro deveria conter uma rota.

Conversor de rota

Usando uma opção direta do Starlette você pode declarar um *parâmetro de rota* contendo uma *rota* usando uma URL como:

```
/files/{file_path:path}
```

Nesse caso, o nome do parâmetro é `file_path`, e a última parte, `:path`, diz que o parâmetro deveria coincidir com qualquer *rota*.

Então, você poderia usar ele com:

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

!!! dica Você poderia precisar que o parâmetro contivesse `/home/johndoe/myfile.txt`, com uma barra no inicio (`/`).

```
Neste caso, a URL deveria ser: `/files//home/johndoe/myfile.txt`, com barra dupla (`//`) entre `files` e `home`.
```

Recapitulando

Com o **FastAPI**, usando as declarações de tipo do Python, você obtém:

- Suporte no editor: verificação de erros, e opção de autocompletar, etc.
- Parsing de dados
- "Parsing" de dados
- Validação de dados
- Anotação da API e documentação automática

Você apenas tem que declará-los uma vez.

Essa é provavelmente a vantagem mais visível do **FastAPI** se comparado com frameworks alternativos (além do desempenho puro).