Parâmetros de consulta e validações de texto

O **FastAPI** permite que você declare informações adicionais e validações aos seus parâmetros.

Vamos utilizar essa aplicação como exemplo:

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

O parâmetro de consulta `q` é do tipo `Optional[str]`, o que significa que é do tipo `str` mas que também pode ser `None`, e de fato, o valor padrão é `None`, então o FastAPI saberá que não é obrigatório.

!!! note "Observação" O FastAPI saberá que o valor de `q` não é obrigatório por causa do valor padrão `= None`.

```
O `Optional` em `Optional[str]` não é usado pelo FastAPI, mas permitirá que seu editor lhe dê um melhor suporte e detecte erros.
```

Validação adicional

Nós iremos forçar que mesmo o parâmetro `q` seja opcional, sempre que informado, **seu tamanho não exceda 50 caracteres**.

Importe `Query`

Para isso, primeiro importe `Query` de `fastapi`:

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

Use `Query` como o valor padrão

Agora utilize-o como valor padrão do seu parâmetro, definindo o parâmetro `max_length` para 50:

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

Note que substituímos o valor padrão de `None` para `Query(None)`, o primeiro parâmetro de `Query` serve para o mesmo propósito: definir o valor padrão do parâmetro.

Então:

```
q: Optional[str] = Query(None)
```

...Torna o parâmetro opcional, da mesma maneira que:

```
q: Optional[str] = None
```

Mas o declara explicitamente como um parâmetro de consulta.

!!! info "Informação" Tenha em mente que o FastAPI se preocupa com a parte:

```Python
= None
```

Ou com:

```Python
= Query(None)
```

E irá utilizar o `None` para detectar que o parâmetro de consulta não é obrigatório.

O `Optional` é apenas para permitir que seu editor de texto lhe dê um melhor suporte.

Então, podemos passar mais parâmetros para `Query`. Neste caso, o parâmetro `max_length` que se aplica a textos:

```Python
q: str = Query(None, max_length=50)
```

Isso irá validar os dados, mostrar um erro claro quando os dados forem inválidos, e documentar o parâmetro na *operação de rota* do esquema OpenAPI..

Adicionando mais validações

Você também pode incluir um parâmetro `min_length`:

```Python
{!../../../docs_src/query_params_str_validations/tutorial003.py!}
```

Adicionando expressões regulares

Você pode definir uma expressão regular que combine com um padrão esperado pelo parâmetro:

```Python
{!../../../docs_src/query_params_str_validations/tutorial004.py!}
```

Essa expressão regular específica verifica se o valor recebido no parâmetro:

- `^`: Inicia com os seguintes caracteres, ou seja, não contém caracteres anteriores.
- `fixedquery`: contém o valor exato `fixedquery`.
- `$`: termina aqui, não contém nenhum caractere após `fixedquery`.

Se você se sente perdido com todo esse assunto de **"expressão regular"**, não se preocupe. Esse é um assunto complicado para a maioria das pessoas. Você ainda pode fazer muitas coisas sem utilizar expressões regulares.

Mas assim que você precisar e já tiver aprendido sobre, saiba que você poderá usá-las diretamente no **FastAPI**.

Valores padrão

Da mesma maneira que você utiliza `None` como o primeiro argumento para ser utilizado como um valor padrão, você pode usar outros valores.

Vamos dizer que você queira que o parâmetro de consulta `q` tenha um `min_length` de `3`, e um valor padrão de `"fixedquery"`, então declararíamos assim:

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

!!! note "Observação" O parâmetro torna-se opcional quando possui um valor padrão.

Torne-o obrigatório

Quando você não necessita de validações ou de metadados adicionais, podemos fazer com que o parâmetro de consulta `q` seja obrigatório por não declarar um valor padrão, dessa forma:

```
q: str
```

em vez desta:

```
q: Optional[str] = None
```

Mas agora nós o estamos declarando como `Query`, conforme abaixo:

```
q: Optional[str] = Query(None, min_length=3)
```

Então, quando você precisa declarar um parâmetro obrigatório utilizando o `Query`, você pode utilizar `...` como o primeiro argumento:

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

!!! info "Informação" Se você nunca viu os `...` antes: é um valor único especial, faz parte do Python e é chamado "Ellipsis".

Dessa forma o **FastAPI** saberá que o parâmetro é obrigatório.

Lista de parâmetros de consulta / múltiplos valores

Quando você declara explicitamente um parâmetro com `Query` você pode declará-lo para receber uma lista de valores, ou podemos dizer, que irá receber mais de um valor.

Por exemplo, para declarar que o parâmetro `q` pode aparecer diversas vezes na URL, você escreveria:

```
{!../../../docs_src/query_params_str_validations/tutorial011.py!}
```

Então, com uma URL assim:

```
http://localhost:8000/items/?q=foo&q=bar
```

você receberá os múltiplos *parâmetros de consulta* `q` com os valores (`foo` e `bar`) em uma lista (`list`) Python dentro da *função de operação de rota*, no *parâmetro da função* `q`.

Assim, a resposta para essa URL seria:

```
{
  "q": [
    "foo",
    "bar"
  ]
}
```

!!! tip "Dica" Para declarar um parâmetro de consulta com o tipo `list`, como no exemplo acima, você precisa usar explicitamente o `Query`, caso contrário será interpretado como um corpo da requisição.

A documentação interativa da API irá atualizar de acordo, permitindo múltiplos valores:

Lista de parâmetros de consulta / múltiplos valores por padrão

E você também pode definir uma lista (`list`) de valores padrão caso nenhum seja informado:

```
{!../../../docs_src/query_params_str_validations/tutorial012.py!}
```

Se você for até:

```
http://localhost:8000/items/
```

O valor padrão de `q` será: `["foo", "bar"]` e sua resposta será:

```
{
  "q": [
    "foo",
    "bar"
  ]
}
```

Usando `list`

Você também pode utilizar o tipo `list` diretamente em vez de `List[str]` :

```
{!../../../docs_src/query_params_str_validations/tutorial013.py!}
```

!!! note "Observação" Tenha em mente que neste caso, o FastAPI não irá validar os conteúdos da lista.

```
Por exemplo, um `List[int]` iria validar (e documentar) que os contéudos da lista são números inteiros. Mas apenas `list` não.
```

Declarando mais metadados

Você pode adicionar mais informações sobre o parâmetro.

Essa informações serão inclusas no esquema do OpenAPI e utilizado pela documentação interativa e ferramentas externas.

!!! note "Observação" Tenha em mente que cada ferramenta oferece diferentes níveis de suporte ao OpenAPI.

```
Algumas delas não exibem todas as informações extras que declaramos, ainda que na maioria dos casos, esses recursos estão planejados para desenvolvimento.
```

Você pode adicionar um `title`:

```
{!../../../docs_src/query_params_str_validations/tutorial007.py!}
```

E uma `description`:

```
{!../../../docs_src/query_params_str_validations/tutorial008.py!}
```

Apelidos (alias) de parâmetros

Imagine que você queira que um parâmetro tenha o nome `item-query`.

Desta maneira:

```
http://127.0.0.1:8000/items/?item-query=foobaritems
```

Mas o nome `item-query` não é um nome de váriavel válido no Python.

O que mais se aproxima é `item_query`.

Mas ainda você precisa que o nome seja exatamente `item-query`...

Então você pode declarar um `alias`, e esse apelido (alias) que será utilizado para encontrar o valor do parâmetro:

```
{!../../../docs_src/query_params_str_validations/tutorial009.py!}
```

Parâmetros descontinuados

Agora vamos dizer que você não queria mais utilizar um parâmetro.

Você tem que deixá-lo ativo por um tempo, já que existem clientes o utilizando. Mas você quer que a documentação deixe claro que este parâmetro será descontinuado.

Então você passa o parâmetro `deprecated=True` para `Query`:

```
{!../../../docs_src/query_params_str_validations/tutorial010.py!}
```

Na documentação aparecerá assim:

Recapitulando

Você pode adicionar validações e metadados adicionais aos seus parâmetros.

Validações genéricas e metadados:

- `alias`
- `title`
- `description`
- `deprecated`

Validações específicas para textos:

- `min_length`
- `max_length`
- `regex`

Nesses exemplos você viu como declarar validações em valores do tipo `str`.

Leia os próximos capítulos para ver como declarar validação de outros tipos, como números.