Query Parameters and String Validations

FastAPI allows you to declare additional information and validation for your parameters.

Let's take this application as example:

Python 3.10+

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from typing import Union

from fastapi import FastAPI

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[str, None] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

The query parameter `q` is of type `Union[str, None]` (or `str | None` in Python 3.10), that means that it's of type `str` but could also be `None`, and indeed, the default value is `None`, so FastAPI will know it's not required.

Note

FastAPI will know that the value of `q` is not required because of the default value `= None`.

The `Union` in `Union[str, None]` will allow your editor to give you better support and detect errors.

Additional validation

We are going to enforce that even though `q` is optional, whenever it is provided, **its length doesn't exceed 50 characters**.

Import `Query` and `Annotated`

To achieve that, first import:

- `Query` from `fastapi`
- `Annotated` from `typing` (or from `typing_extensions` in Python below 3.9)

Python 3.10+
In Python 3.9 or above, `Annotated` is part of the standard library, so you can import it from `typing`.

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str | None, Query(max_length=50)] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+
In versions of Python below Python 3.9 you import `Annotated` from `typing_extensions`.

It will already be installed with FastAPI.

```
from typing import Union
```

```python
from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[Union[str, None], Query(max_length=50)] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Info

FastAPI added support for `Annotated` (and started recommending it) in version 0.95.0.

If you have an older version, you would get errors when trying to use `Annotated`.

Make sure you Upgrade the FastAPI version to at least 0.95.1 before using `Annotated`.

Use `Annotated` in the type for the `q` parameter

Remember I told you before that `Annotated` can be used to add metadata to your parameters in the Python Types Intro?

Now it's the time to use it with FastAPI. 🚀

We had this type annotation:

Python 3.10+

```python
q: str | None = None
```

Python 3.8+

```python
q: Union[str, None] = None
```

What we will do is wrap that with `Annotated`, so it becomes:

Python 3.10+

```python
q: Annotated[str | None] = None
```

Python 3.8+

```python
q: Annotated[Union[str, None]] = None
```

Both of those versions mean the same thing, `q` is a parameter that can be a `str` or `None`, and by default, it is `None`.

Now let's jump to the fun stuff. 🎉

Add `Query` to `Annotated` in the `q` parameter

Now that we have this `Annotated` where we can put more information (in this case some additional validation), add `Query` inside of `Annotated`, and set the parameter `max_length` to `50`:

Python 3.10+

```python
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str | None, Query(max_length=50)] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.8+

```python
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[Union[str, None], Query(max_length=50)] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=None, max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[str, None] = Query(default=None, max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Notice that the default value is still `None`, so the parameter is still optional.

But now, having `Query(max_length=50)` inside of `Annotated`, we are telling FastAPI that we want it to have **additional validation** for this value, we want it to have maximum 50 characters. 😎

Tip

Here we are using `Query()` because this is a **query parameter**. Later we will see others like `Path()`, `Body()`, `Header()`, and `Cookie()`, that also accept the same arguments as `Query()`.

FastAPI will now:

- **Validate** the data making sure that the max length is 50 characters
- Show a **clear error** for the client when the data is not valid
- **Document** the parameter in the OpenAPI schema *path operation* (so it will show up in the **automatic docs UI**)

Alternative (old): `Query` as the default value

Previous versions of FastAPI (before 0.95.0) required you to use `Query` as the default value of your parameter, instead of putting it in `Annotated`, there's a high chance that you will see code using it around, so I'll explain it to you.

Tip

For new code and whenever possible, use `Annotated` as explained above. There are multiple advantages (explained below) and no disadvantages. 🍰

This is how you would use `Query()` as the default value of your function parameter, setting the parameter `max_length` to 50:

Python 3.10+ - non-Annotated

```
from fastapi import FastAPI, Query
```

```
app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=None, max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str | None, Query(max_length=50)] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[Union[str, None], Query(max_length=50)] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[str, None] = Query(default=None, max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

As in this case (without using `Annotated`) we have to replace the default value `None` in the function with `Query()`, we now need to set the default value with the parameter `Query(default=None)`, it serves the same purpose of defining that default value (at least for FastAPI).

So:

```
q: Union[str, None] = Query(default=None)
```

...makes the parameter optional, with a default value of `None`, the same as:

```
q: Union[str, None] = None
```

And in Python 3.10 and above:

```
q: str | None = Query(default=None)
```

...makes the parameter optional, with a default value of `None`, the same as:

```
q: str | None = None
```

But the `Query` versions declare it explicitly as being a query parameter.

Info

Keep in mind that the most important part to make a parameter optional is the part:

```
= None
```

or the:

```
= Query(default=None)
```

as it will use that `None` as the default value, and that way make the parameter **not required**.

The `Union[str, None]` part allows your editor to provide better support, but it is not what tells FastAPI that this parameter is not required.

Then, we can pass more parameters to `Query`. In this case, the `max_length` parameter that applies to strings:

```
q: Union[str, None] = Query(default=None, max_length=50)
```

This will validate the data, show a clear error when the data is not valid, and document the parameter in the OpenAPI schema *path operation*.

`Query` as the default value or in `Annotated`

Keep in mind that when using `Query` inside of `Annotated` you cannot use the `default` parameter for `Query`.

Instead use the actual default value of the function parameter. Otherwise, it would be inconsistent.

For example, this is not allowed:

```
q: Annotated[str, Query(default="rick")] = "morty"
```

...because it's not clear if the default value should be `"rick"` or `"morty"`.

So, you would use (preferably):

```
q: Annotated[str, Query()] = "rick"
```

...or in older code bases you will find:

```
q: str = Query(default="rick")
```

Advantages of `Annotated`

Using `Annotated` is recommended instead of the default value in function parameters, it is **better** for multiple reasons. 🤓

The **default** value of the **function parameter** is the **actual default** value, that's more intuitive with Python in general. 😌

You could **call** that same function in **other places** without FastAPI, and it would **work as expected**. If there's a **required** parameter (without a default value), your **editor** will let you know with an error, **Python** will also complain if you run it without passing the required parameter.

When you don't use `Annotated` and instead use the **(old) default value style**, if you call that function without FastAPI in **other places**, you have to **remember** to pass the arguments to the function for it to work correctly, otherwise the values will be different from what you expect (e.g. `QueryInfo` or something similar instead of `str`). And your editor won't complain, and Python won't complain running that function, only when the operations inside error out.

Because `Annotated` can have more than one metadata annotation, you could now even use the same function with other tools, like Typer. 🚀

Add more validations

You can also add a parameter `min_length`:

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[str | None, Query(min_length=3, max_length=50)] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[Union[str, None], Query(min_length=3, max_length=50)] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[Union[str, None], Query(min_length=3, max_length=50)] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=None, min_length=3, max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(default=None, min_length=3, max_length=50),
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Add regular expressions

You can define a regular expression `pattern` that the parameter should match:

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        str | None, Query(min_length=3, max_length=50, pattern="^fixedquery$")
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

► 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        Union[str, None], Query(min_length=3, max_length=50, pattern="^fixedquery$")
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        Union[str, None], Query(min_length=3, max_length=50, pattern="^fixedquery$")
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: str | None = Query(
        default=None, min_length=3, max_length=50, pattern="^fixedquery$"
    ),
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(
        default=None, min_length=3, max_length=50, pattern="^fixedquery$"
    ),
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

This specific regular expression pattern checks that the received parameter value:

- `^` : starts with the following characters, doesn't have characters before.
- `fixedquery` : has the exact value `fixedquery` .
- `$` : ends there, doesn't have any more characters after `fixedquery` .

If you feel lost with all these **"regular expression"** ideas, don't worry. They are a hard topic for many people. You can still do a lot of stuff without needing regular expressions yet.

But whenever you need them and go and learn them, know that you can already use them directly in **FastAPI**.

Pydantic v1 `regex` instead of `pattern`

Before Pydantic version 2 and before FastAPI 0.100.0, the parameter was called `regex` instead of `pattern` , but it's now deprecated.

You could still see some code using it:

Pydantic v1

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        str | None, Query(min_length=3, max_length=50, regex="^fixedquery$")
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

But know that this is deprecated and it should be updated to use the new parameter `pattern` . 🤓

Default values

You can, of course, use default values other than `None` .

Let's say that you want to declare the `q` query parameter to have a `min_length` of `3` , and to have a default value of `"fixedquery"` :

Python 3.9+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str, Query(min_length=3)] = "fixedquery"):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str, Query(min_length=3)] = "fixedquery"):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str = Query(default="fixedquery", min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Note

Having a default value of any type, including `None` , makes the parameter optional (not required).

Required parameters

When we don't need to declare more validations or metadata, we can make the `q` query parameter required just by not declaring a default value, like:

```
q: str
```

instead of:

```
q: Union[str, None] = None
```

But we are now declaring it with `Query` , for example like:

Annotated

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

non-Annotated

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

So, when you need to declare a value as required while using `Query` , you can simply not declare a default value:

Python 3.9+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str, Query(min_length=3)]):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str, Query(min_length=3)]):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str = Query(min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Required with Ellipsis (`...`)

There's an alternative way to explicitly declare that a value is required. You can set the default to the literal value `...` :

Python 3.9+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str, Query(min_length=3)] = ...):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str, Query(min_length=3)] = ...):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query
```

```
app = FastAPI()


@app.get("/items/")
async def read_items(q: str = Query(default=..., min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Info

If you hadn't seen that `...` before: it is a special single value, it is part of Python and is called "Ellipsis".

It is used by Pydantic and FastAPI to explicitly declare that a value is required.

This will let **FastAPI** know that this parameter is required.

Required, can be `None`

You can declare that a parameter can accept `None`, but that it's still required. This would force clients to send a value, even if the value is `None`.

To do that, you can declare that `None` is a valid type but still use `...` as the default:

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str | None, Query(min_length=3)] = ...):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[Union[str, None], Query(min_length=3)] = ...):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[Union[str, None], Query(min_length=3)] = ...):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query
```

```
app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=..., min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[str, None] = Query(default=..., min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Tip

Pydantic, which is what powers all the data validation and serialization in FastAPI, has a special behavior when you use `Optional` or `Union[Something, None]` without a default value, you can read more about it in the Pydantic docs about Required fields.

Tip

Remember that in most of the cases, when something is required, you can simply omit the default, so you normally don't have to use `...`.

Query parameter list / multiple values

When you define a query parameter explicitly with `Query` you can also declare it to receive a list of values, or said in another way, to receive multiple values.

For example, to declare a query parameter `q` that can appear multiple times in the URL, you can write:

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[list[str] | None, Query()] = None):
    query_items = {"q": q}
    return query_items
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[Union[list[str], None], Query()] = None):
    query_items = {"q": q}
    return query_items
```

Python 3.8+

```
from typing import List, Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated
```

```
app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[Union[List[str], None], Query()] = None):
    query_items = {"q": q}
    return query_items
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: list[str] | None = Query(default=None)):
    query_items = {"q": q}
    return query_items
```

Python 3.9+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[list[str], None] = Query(default=None)):
    query_items = {"q": q}
    return query_items
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import List, Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[List[str], None] = Query(default=None)):
    query_items = {"q": q}
    return query_items
```

Then, with a URL like:

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

you would receive the multiple `q` *query parameters'* values (`foo` and `bar`) in a Python `list` inside your *path operation function*, in the *function parameter* `q` .

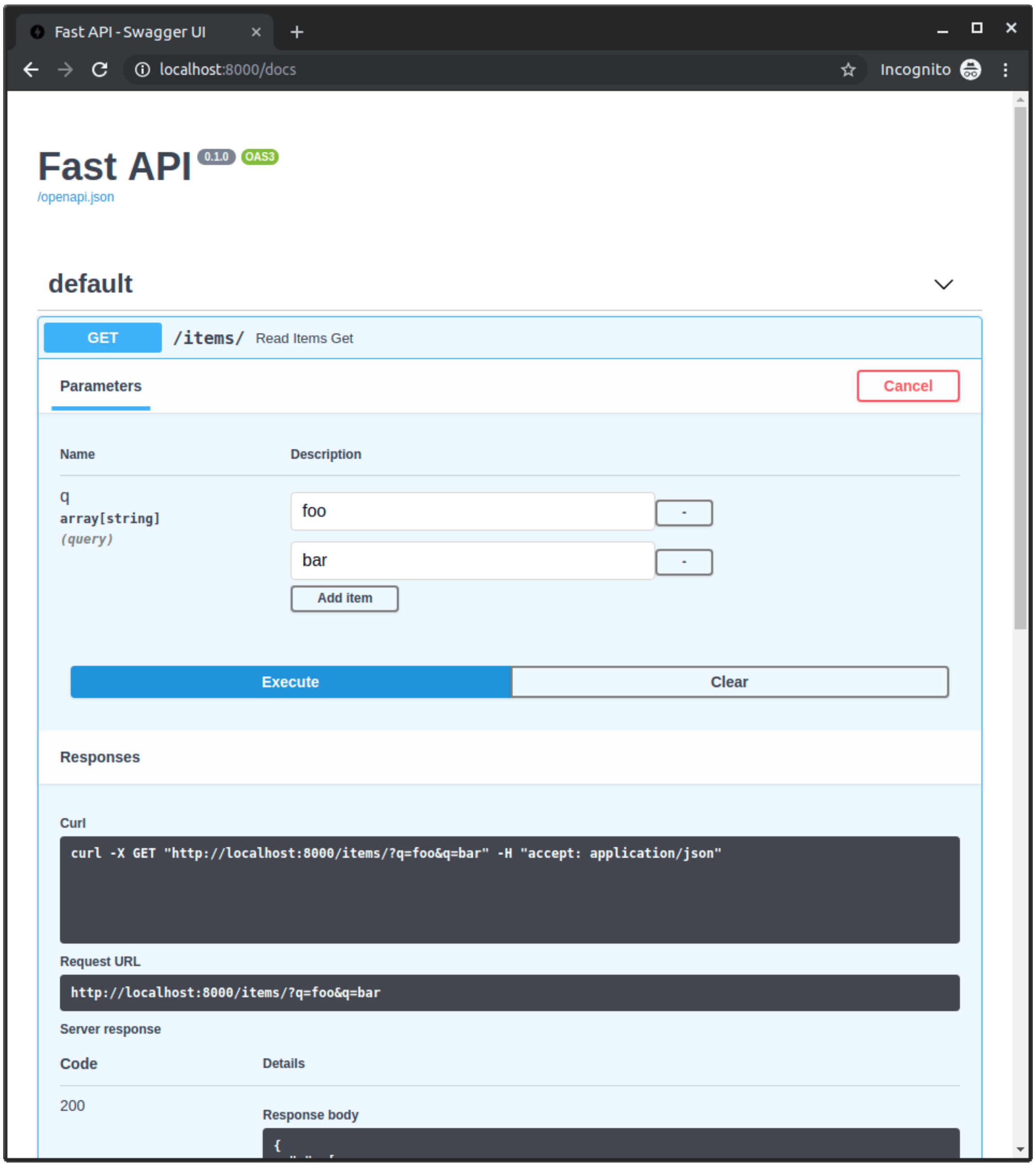
So, the response to that URL would be:

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

Tip

To declare a query parameter with a type of `list` , like in the example above, you need to explicitly use `Query` , otherwise it would be interpreted as a request body.

The interactive API docs will update accordingly, to allow multiple values:



```
curl -X GET "http://localhost:8000/items/?q=foo&q=bar" -H "accept: application/json"
```

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

Query parameter list / multiple values with defaults

And you can also define a default `list` of values if none are provided:

Python 3.9+

```python
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[list[str], Query()] = ["foo", "bar"]):
    query_items = {"q": q}
    return query_items
```

▸ 🤓 Other versions and variants

Python 3.8+

```python
from typing import List

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()
```

```
@app.get("/items/")
async def read_items(q: Annotated[List[str], Query()] = ["foo", "bar"]):
    query_items = {"q": q}
    return query_items
```

Python 3.9+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: list[str] = Query(default=["foo", "bar"])):
    query_items = {"q": q}
    return query_items
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import List

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: List[str] = Query(default=["foo", "bar"])):
    query_items = {"q": q}
    return query_items
```

If you go to:

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

the default of `q` will be: `["foo", "bar"]` and your response will be:

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

Using just `list`

You can also use `list` directly instead of `List[str]` (or `list[str]` in Python 3.9+):

Python 3.9+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[list, Query()] = []):
    query_items = {"q": q}
    return query_items
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()
```

```
@app.get("/items/")
async def read_items(q: Annotated[list, Query()] = []):
    query_items = {"q": q}
    return query_items
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: list = Query(default=[])):
    query_items = {"q": q}
    return query_items
```

Note

Keep in mind that in this case, FastAPI won't check the contents of the list.

For example, `List[int]` would check (and document) that the contents of the list are integers. But `list` alone wouldn't.

Declare more metadata

You can add more information about the parameter.

That information will be included in the generated OpenAPI and used by the documentation user interfaces and external tools.

Note

Keep in mind that different tools might have different levels of OpenAPI support.

Some of them might not show all the extra information declared yet, although in most of the cases, the missing feature is already planned for development.

You can add a `title` :

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[str | None, Query(title="Query string", min_length=3)] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[Union[str, None], Query(title="Query string", min_length=3)] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+

```python
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[Union[str, None], Query(title="Query string", min_length=3)] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: str | None = Query(default=None, title="Query string", min_length=3),
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(default=None, title="Query string", min_length=3),
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

And a `description`:

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        str | None,
        Query(
            title="Query string",
            description="Query string for the items to search in the database that have a good match",
            min_length=3,
        ),
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        Union[str, None],
        Query(
            title="Query string",
            description="Query string for the items to search in the database that have a good match",
            min_length=3,
        ),
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        Union[str, None],
        Query(
            title="Query string",
            description="Query string for the items to search in the database that have a good match",
            min_length=3,
        ),
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()
```

```
@app.get("/items/")
async def read_items(
    q: str | None = Query(
        default=None,
        title="Query string",
        description="Query string for the items to search in the database that have a good match",
        min_length=3,
    ),
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(
        default=None,
        title="Query string",
        description="Query string for the items to search in the database that have a good match",
        min_length=3,
    ),
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Alias parameters

Imagine that you want the parameter to be `item-query`.

Like in:

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

But `item-query` is not a valid Python variable name.

The closest would be `item_query`.

But you still need it to be exactly `item-query`...

Then you can declare an `alias`, and that alias is what will be used to find the parameter value:

Python 3.10+

```
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[str | None, Query(alias="item-query")] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import FastAPI, Query

app = FastAPI()
```

```python
@app.get("/items/")
async def read_items(q: Annotated[Union[str, None], Query(alias="item-query")] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+

```python
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(q: Annotated[Union[str, None], Query(alias="item-query")] = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=None, alias="item-query")):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[str, None] = Query(default=None, alias="item-query")):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Deprecating parameters

Now let's say you don't like this parameter anymore.

You have to leave it there a while because there are clients using it, but you want the docs to clearly show it as deprecated.

Then pass the parameter `deprecated=True` to `Query` :

Python 3.10+

```python
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        str | None,
        Query(
            alias="item-query",
            title="Query string",
            description="Query string for the items to search in the database that have a good match",
            min_length=3,
            max_length=50,
            pattern="^fixedquery$",
            deprecated=True,
        ),
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

► 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        Union[str, None],
        Query(
            alias="item-query",
            title="Query string",
            description="Query string for the items to search in the database that have a good match",
            min_length=3,
            max_length=50,
            pattern="^fixedquery$",
            deprecated=True,
        ),
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+

```
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Annotated[
        Union[str, None],
        Query(
            alias="item-query",
            title="Query string",
            description="Query string for the items to search in the database that have a good match",
            min_length=3,
            max_length=50,
            pattern="^fixedquery$",
            deprecated=True,
        ),
    ] = None,
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query

app = FastAPI()
```

```
@app.get("/items/")
async def read_items(
    q: str | None = Query(
        default=None,
        alias="item-query",
        title="Query string",
        description="Query string for the items to search in the database that have a good match",
        min_length=3,
        max_length=50,
        pattern="^fixedquery$",
        deprecated=True,
    ),
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

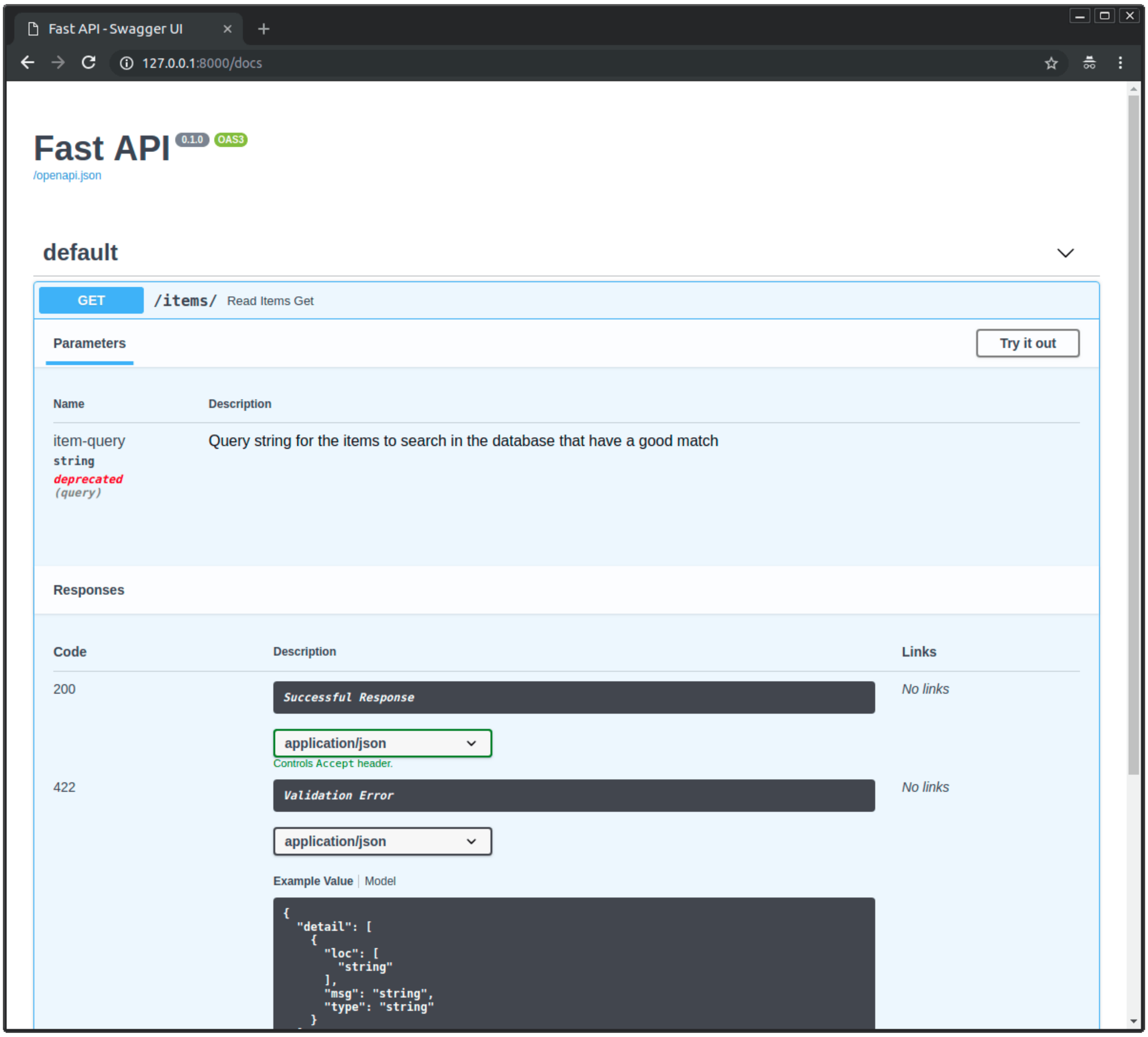
```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(
        default=None,
        alias="item-query",
        title="Query string",
        description="Query string for the items to search in the database that have a good match",
        min_length=3,
        max_length=50,
        pattern="^fixedquery$",
        deprecated=True,
    ),
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

The docs will show it like this:



```
{
  "detail": [
    {
      "loc": [
        "string"
      ],
      "msg": "string",
      "type": "string"
    }
```

Exclude parameters from OpenAPI

To exclude a query parameter from the generated OpenAPI schema (and thus, from the automatic documentation systems), set the parameter `include_in_schema` of `Query` to `False` :

Python 3.10+

```python
from typing import Annotated

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    hidden_query: Annotated[str | None, Query(include_in_schema=False)] = None,
):
    if hidden_query:
        return {"hidden_query": hidden_query}
    else:
        return {"hidden_query": "Not found"}
```

▸ 🤓 Other versions and variants

Python 3.9+

```python
from typing import Annotated, Union

from fastapi import FastAPI, Query

app = FastAPI()
```

```python
@app.get("/items/")
async def read_items(
    hidden_query: Annotated[Union[str, None], Query(include_in_schema=False)] = None,
):
    if hidden_query:
        return {"hidden_query": hidden_query}
    else:
        return {"hidden_query": "Not found"}
```

Python 3.8+

```python
from typing import Union

from fastapi import FastAPI, Query
from typing_extensions import Annotated

app = FastAPI()


@app.get("/items/")
async def read_items(
    hidden_query: Annotated[Union[str, None], Query(include_in_schema=False)] = None,
):
    if hidden_query:
        return {"hidden_query": hidden_query}
    else:
        return {"hidden_query": "Not found"}
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    hidden_query: str | None = Query(default=None, include_in_schema=False),
):
    if hidden_query:
        return {"hidden_query": hidden_query}
    else:
        return {"hidden_query": "Not found"}
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    hidden_query: Union[str, None] = Query(default=None, include_in_schema=False),
):
    if hidden_query:
        return {"hidden_query": hidden_query}
    else:
        return {"hidden_query": "Not found"}
```

Recap

You can declare additional validations and metadata for your parameters.

Generic validations and metadata:

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

- `deprecated`

Validations specific for strings:

- `min_length`
- `max_length`
- `pattern`

In these examples you saw how to declare validations for `str` values.

See the next chapters to learn how to declare validations for other types, like numbers.