Dependencies in path operation decorators

In some cases you don't really need the return value of a dependency inside your *path operation function*.

Or the dependency doesn't return a value.

But you still need it to be executed/solved.

For those cases, instead of declaring a *path operation function* parameter with `Depends`, you can add a `list` of `dependencies` to the *path operation decorator*.

Add `dependencies` to the *path operation decorator*

The *path operation decorator* receives an optional argument `dependencies`.

It should be a `list` of `Depends()`:

Python 3.9+

```
from typing import Annotated

from fastapi import Depends, FastAPI, Header, HTTPException

app = FastAPI()


async def verify_token(x_token: Annotated[str, Header()]):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: Annotated[str, Header()]):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from fastapi import Depends, FastAPI, Header, HTTPException
from typing_extensions import Annotated

app = FastAPI()


async def verify_token(x_token: Annotated[str, Header()]):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: Annotated[str, Header()]):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI, Header, HTTPException

app = FastAPI()


async def verify_token(x_token: str = Header()):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: str = Header()):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key
```

```
@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

These dependencies will be executed/solved the same way as normal dependencies. But their value (if they return any) won't be passed to your *path operation function*.

Tip

Some editors check for unused function parameters, and show them as errors.

Using these `dependencies` in the *path operation decorator* you can make sure they are executed while avoiding editor/tooling errors.

It might also help avoid confusion for new developers that see an unused parameter in your code and could think it's unnecessary.

Info

In this example we use invented custom headers `X-Key` and `X-Token`.

But in real cases, when implementing security, you would get more benefits from using the integrated Security utilities (the next chapter).

Dependencies errors and return values

You can use the same dependency *functions* you use normally.

Dependency requirements

They can declare request requirements (like headers) or other sub-dependencies:

Python 3.9+

```
from typing import Annotated

from fastapi import Depends, FastAPI, Header, HTTPException

app = FastAPI()


async def verify_token(x_token: Annotated[str, Header()]):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: Annotated[str, Header()]):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from fastapi import Depends, FastAPI, Header, HTTPException
from typing_extensions import Annotated

app = FastAPI()


async def verify_token(x_token: Annotated[str, Header()]):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: Annotated[str, Header()]):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from fastapi import Depends, FastAPI, Header, HTTPException

app = FastAPI()


async def verify_token(x_token: str = Header()):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: str = Header()):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

Raise exceptions

These dependencies can `raise` exceptions, the same as normal dependencies:

Python 3.9+

```python
from typing import Annotated

from fastapi import Depends, FastAPI, Header, HTTPException

app = FastAPI()


async def verify_token(x_token: Annotated[str, Header()]):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: Annotated[str, Header()]):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

▸ 🤓 Other versions and variants

Python 3.8+

```python
from fastapi import Depends, FastAPI, Header, HTTPException
from typing_extensions import Annotated

app = FastAPI()


async def verify_token(x_token: Annotated[str, Header()]):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: Annotated[str, Header()]):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from fastapi import Depends, FastAPI, Header, HTTPException

app = FastAPI()


async def verify_token(x_token: str = Header()):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")
```

```
async def verify_key(x_key: str = Header()):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

Return values

And they can return values or not, the values won't be used.

So, you can reuse a normal dependency (that returns a value) you already use somewhere else, and even though the value won't be used, the dependency will be executed:

Python 3.9+

```
from typing import Annotated

from fastapi import Depends, FastAPI, Header, HTTPException

app = FastAPI()


async def verify_token(x_token: Annotated[str, Header()]):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: Annotated[str, Header()]):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from fastapi import Depends, FastAPI, Header, HTTPException
from typing_extensions import Annotated

app = FastAPI()


async def verify_token(x_token: Annotated[str, Header()]):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: Annotated[str, Header()]):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key


@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI, Header, HTTPException

app = FastAPI()


async def verify_token(x_token: str = Header()):
    if x_token != "fake-super-secret-token":
        raise HTTPException(status_code=400, detail="X-Token header invalid")


async def verify_key(x_key: str = Header()):
    if x_key != "fake-super-secret-key":
        raise HTTPException(status_code=400, detail="X-Key header invalid")
    return x_key
```

```
@app.get("/items/", dependencies=[Depends(verify_token), Depends(verify_key)])
async def read_items():
    return [{"item": "Foo"}, {"item": "Bar"}]
```

Dependencies for a group of *path operations*

Later, when reading about how to structure bigger applications (Bigger Applications - Multiple Files), possibly with multiple files, you will learn how to declare a single `dependencies` parameter for a group of *path operations*.

Global Dependencies

Next we will see how to add dependencies to the whole `FastAPI` application, so that they apply to each *path operation*.