Introduction

There are several ways we've seen how we can return responses from our application. We could use the JSONResponse to specify the status code and the content, we could return a pydantic model, or we could also just return a python object. In addition, we also saw that we could set the status_code in our path decorator to define the default status code.

However, in some cases we'd maybe just want to return a response code if we reach a certain part of our code without any further content.

A response object we can use for this is called Response, which you can import from fastapi.responses.

Exercise 1

Implement a dummy GET endpoint which returns a 204 response.

Exercise 2

A better way than just referencing the actual number of the status code response is to use a defined variable that holds that value instead. This can either be an individual variable, or an attribute of a class (often an enumeration class).

FastAPI also provides each status code in a variable form, which you can import by doing from fastapi import status. Import the variable corresponding to the 204 response (hint: it's called HTTP_204_NO_CONTENT), and use its value instead of just the raw numerical value in the dummy method from exercise 1.

Solution

Exercise 1

```python
from fastapi.responses import Response


@user_router.patch("/dummy")
async def test_response_code() -> Response:
    return Response(status_code=204)
```

Exercise 2

```python
from fastapi import status
from fastapi.responses import Response


@user_router.patch("/dummy")
async def test_response_code() -> Response:
    return Response(status_code=status.HTTP_204_NO_CONTENT)
```

Although we can define our http responses just numerically, it’s good practice to use variables that contain these values. For one, it helps with readability, but it also helps with avoiding typos and being able to more-easily re-use them across the application. Although FastAPI has already implemented these for us, so we don’t need to implement them ourselves, it’s good to keep this in mind in case you recognize in the future you’ve built something where you’re often re-using the same number, string, or the like. In those cases you should then think about putting it in a variable instead that specifies what the value actually means.

In fact, even in some cases where you’re just comparing to a hard limit, e.g.

```python
if page_row <= 5:
   continue
```

It can be good to replace it with a variable for readability sakes, e.g.

```python
HEADER_ROW_SIZE = 5
if page_row <= HEADER_ROW_SIZE:
   continue
```