Middleware

You can add middleware to **FastAPI** applications.

A "middleware" is a function that works with every **request** before it is processed by any specific *path operation*. And also with every **response** before returning it.

- It takes each **request** that comes to your application.
- It can then do something to that **request** or run any needed code.
- Then it passes the **request** to be processed by the rest of the application (by some *path operation*).
- It then takes the **response** generated by the application (by some *path operation*).
- It can do something to that **response** or run any needed code.
- Then it returns the **response**.

Technical Details

If you have dependencies with `yield`, the exit code will run *after* the middleware.

If there were any background tasks (documented later), they will run *after* all the middleware.

Create a middleware

To create a middleware you use the decorator `@app.middleware("http")` on top of a function.

The middleware function receives:

- The `request`.
- A function `call_next` that will receive the `request` as a parameter.
  - This function will pass the `request` to the corresponding *path operation*.
  - Then it returns the `response` generated by the corresponding *path operation*.
- You can then further modify the `response` before returning it.

Python 3.8+

```
import time

from fastapi import FastAPI, Request

app = FastAPI()


@app.middleware("http")
async def add_process_time_header(request: Request, call_next):
    start_time = time.perf_counter()
    response = await call_next(request)
    process_time = time.perf_counter() - start_time
    response.headers["X-Process-Time"] = str(process_time)
    return response
```

Tip

Keep in mind that custom proprietary headers can be added using the 'X-' prefix.

But if you have custom headers that you want a client in a browser to be able to see, you need to add them to your CORS configurations (CORS (Cross-Origin Resource Sharing)) using the parameter `expose_headers` documented in Starlette's CORS docs.

Technical Details

You could also use `from starlette.requests import Request`.

FastAPI provides it as a convenience for you, the developer. But it comes directly from Starlette.

Before and after the `response`

You can add code to be run with the `request`, before any *path operation* receives it.

And also after the `response` is generated, before returning it.

For example, you could add a custom header `X-Process-Time` containing the time in seconds that it took to process the request and generate a response:

Python 3.8+

```
import time

from fastapi import FastAPI, Request

app = FastAPI()


@app.middleware("http")
```

```
async def add_process_time_header(request: Request, call_next):
    start_time = time.perf_counter()
    response = await call_next(request)
    process_time = time.perf_counter() - start_time
    response.headers["X-Process-Time"] = str(process_time)
    return response
```

Tip

Here we use `time.perf_counter()` instead of `time.time()` because it can be more precise for these use cases. 🤓

Other middlewares

You can later read more about other middlewares in the Advanced User Guide: Advanced Middleware.

You will read about how to handle CORS with a middleware in the next section.