Advanced Middleware

In the main tutorial you read how to add Custom Middleware{.internal-link target=_blank} to your application.

And then you also read how to handle CORS with the `CORSMiddleware`{.internal-link target=_blank}.

In this section we'll see how to use other middlewares.

Adding ASGI middlewares

As **FastAPI** is based on Starlette and implements the ASGI specification, you can use any ASGI middleware.

A middleware doesn't have to be made for FastAPI or Starlette to work, as long as it follows the ASGI spec.

In general, ASGI middlewares are classes that expect to receive an ASGI app as the first argument.

So, in the documentation for third-party ASGI middlewares they will probably tell you to do something like:

```
from unicorn import UnicornMiddleware

app = SomeASGIApp()

new_app = UnicornMiddleware(app, some_config="rainbow")
```

But FastAPI (actually Starlette) provides a simpler way to do it that makes sure that the internal middlewares to handle server errors and custom exception handlers work properly.

For that, you use `app.add_middleware()` (as in the example for CORS).

```
from fastapi import FastAPI
from unicorn import UnicornMiddleware

app = FastAPI()

app.add_middleware(UnicornMiddleware, some_config="rainbow")
```

`app.add_middleware()` receives a middleware class as the first argument and any additional arguments to be passed to the middleware.

Integrated middlewares

FastAPI includes several middlewares for common use cases, we'll see next how to use them.

!!! note "Technical Details" For the next examples, you could also use `from starlette.middleware.something import SomethingMiddleware`.

```
**FastAPI** provides several middlewares in `fastapi.middleware` just as a convenience
for you, the developer. But most of the available middlewares come directly from
Starlette.
```

`HTTPSRedirectMiddleware`

Enforces that all incoming requests must either be `https` or `wss`.

Any incoming requests to `http` or `ws` will be redirected to the secure scheme instead.

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

`TrustedHostMiddleware`

Enforces that all incoming requests have a correctly set `Host` header, in order to guard against HTTP Host Header attacks.

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

The following arguments are supported:

- `allowed_hosts` - A list of domain names that should be allowed as hostnames. Wildcard domains such as `*.example.com` are supported for matching subdomains to allow any hostname either use `allowed_hosts=["*"]` or omit the middleware.

If an incoming request does not validate correctly then a `400` response will be sent.

`GZipMiddleware`

Handles GZip responses for any request that includes `"gzip"` in the `Accept-Encoding` header.

The middleware will handle both standard and streaming responses.

```
{!../../../docs_src/advanced_middleware/tutorial003.py!}
```

The following arguments are supported:

- `minimum_size` - Do not GZip responses that are smaller than this minimum size in bytes. Defaults to `500`.

Other middlewares

There are many other ASGI middlewares.

For example:

- Sentry
- Uvicorn's `ProxyHeadersMiddleware`
- MessagePack

To see other available middlewares check Starlette's Middleware docs and the ASGI Awesome List.