Using the Request Directly

Up to now, you have been declaring the parts of the request that you need with their types.

Taking data from:

- The path as parameters.
- Headers.
- Cookies.
- etc.

And by doing so, **FastAPI** is validating that data, converting it and generating documentation for your API automatically.

But there are situations where you might need to access the `Request` object directly.

Details about the `Request` object

As **FastAPI** is actually **Starlette** underneath, with a layer of several tools on top, you can use Starlette's `Request` object directly when you need to.

It would also mean that if you get data from the `Request` object directly (for example, read the body) it won't be validated, converted or documented (with OpenAPI, for the automatic API user interface) by FastAPI.

Although any other parameter declared normally (for example, the body with a Pydantic model) would still be validated, converted, annotated, etc.

But there are specific cases where it's useful to get the `Request` object.

Use the `Request` object directly

Let's imagine you want to get the client's IP address/host inside of your *path operation function*.

For that you need to access the request directly.

```
Python hl_lines="1  7-8" {!../../../docs_src/using_request_directly/tutorial001.py!}
```

By declaring a *path operation function* parameter with the type being the `Request` **FastAPI** will know to pass the `Request` in that parameter.

!!! tip Note that in this case, we are declaring a path parameter beside the request parameter.

```
So, the path parameter will be extracted, validated, converted to the specified type and ann

The same way, you can declare any other parameter as normally, and additionally, get the `Re
```

`Request` documentation

You can read more details about the `Request` object in the official Starlette documentation site.

!!! note "Technical Details" You could also use `from starlette.requests import Request`.

```
**FastAPI** provides it directly just as a convenience for you, the developer. But it comes
```