Custom Response - HTML, Stream, File, others

By default, **FastAPI** will return the responses using `JSONResponse`.

You can override it by returning a `Response` directly as seen in [Return a Response directly]{.internal-link target=_blank}.

But if you return a `Response` directly, the data won't be automatically converted, and the documentation won't be automatically generated (for example, including the specific "media type", in the HTTP header `Content-Type` as part of the generated OpenAPI).

But you can also declare the `Response` that you want to be used, in the *path operation decorator*.

The contents that you return from your *path operation function* will be put inside of that `Response`.

And if that `Response` has a JSON media type (`application/json`), like is the case with the `JSONResponse` and `UJSONResponse`, the data you return will be automatically converted (and filtered) with any Pydantic `response_model` that you declared in the *path operation decorator*.

!!! note If you use a response class with no media type, FastAPI will expect your response to have no content, so it will not document the response format in its generated OpenAPI docs.

Use `ORJSONResponse`

For example, if you are squeezing performance, you can install and use [`orjson`] and set the response to be `ORJSONResponse`.

Import the `Response` class (sub-class) you want to use and declare it in the *path operation decorator*.

```
{!../../../docs_src/custom_response/tutorial001b.py!}
```

!!! info The parameter `response_class` will also be used to define the "media type" of the response.

```
In this case, the HTTP header `Content-Type` will be set to `application/json`.

And it will be documented as such in OpenAPI.
```

!!! tip The `ORJSONResponse` is currently only available in FastAPI, not in Starlette.

HTML Response

To return a response with HTML directly from **FastAPI**, use `HTMLResponse`.

- Import `HTMLResponse`.
- Pass `HTMLResponse` as the parameter `response_class` of your *path operation decorator*.

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

!!! info The parameter `response_class` will also be used to define the "media type" of the response.

```
In this case, the HTTP header `Content-Type` will be set to `text/html`.

And it will be documented as such in OpenAPI.
```

Return a `Response`

As seen in [Return a Response directly](){.internal-link target=_blank}, you can also override the response directly in your *path operation*, by returning it.

The same example from above, returning an `HTMLResponse`, could look like:

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

!!! warning A `Response` returned directly by your *path operation function* won't be documented in OpenAPI (for example, the `Content-Type` won't be documented) and won't be visible in the automatic interactive docs.

!!! info Of course, the actual `Content-Type` header, status code, etc, will come from the `Response` object your returned.

Document in OpenAPI and override `Response`

If you want to override the response from inside of the function but at the same time document the "media type" in OpenAPI, you can use the `response_class` parameter AND return a `Response` object.

The `response_class` will then be used only to document the OpenAPI *path operation*, but your `Response` will be used as is.

Return an `HTMLResponse` directly

For example, it could be something like:

```
{!../../../docs_src/custom_response/tutorial004.py!}
```

In this example, the function `generate_html_response()` already generates and returns a `Response` instead of returning the HTML in a `str`.

By returning the result of calling `generate_html_response()`, you are already returning a `Response` that will override the default **FastAPI** behavior.

But as you passed the `HTMLResponse` in the `response_class` too, **FastAPI** will know how to document it in OpenAPI and the interactive docs as HTML with `text/html`:

Available responses

Here are some of the available responses.

Have in mind that you can use `Response` to return anything else, or even create a custom sub-class.

!!! note "Technical Details" You could also use `from starlette.responses import HTMLResponse`.

```
**FastAPI** provides the same `starlette.responses` as `fastapi.responses` just as a convenience for you, the developer. But most of the available responses come directly from Starlette.
```

`Response`

The main `Response` class, all the other responses inherit from it.

You can return it directly.

It accepts the following parameters:

- `content` - A `str` or `bytes`.
- `status_code` - An `int` HTTP status code.
- `headers` - A `dict` of strings.
- `media_type` - A `str` giving the media type. E.g. `"text/html"`.

FastAPI (actually Starlette) will automatically include a Content-Length header. It will also include a Content-Type header, based on the media_type and appending a charset for text types.

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

`HTMLResponse`

Takes some text or bytes and returns an HTML response, as you read above.

`PlainTextResponse`

Takes some text or bytes and returns an plain text response.

```
{!../../../docs_src/custom_response/tutorial005.py!}
```

`JSONResponse`

Takes some data and returns an `application/json` encoded response.

This is the default response used in **FastAPI**, as you read above.

`ORJSONResponse`

A fast alternative JSON response using `orjson`, as you read above.

`UJSONResponse`

An alternative JSON response using `ujson`.

!!! warning `ujson` is less careful than Python's built-in implementation in how it handles some edge-cases.

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

!!! tip It's possible that `ORJSONResponse` might be a faster alternative.

`RedirectResponse`

Returns an HTTP redirect. Uses a 307 status code (Temporary Redirect) by default.

You can return a `RedirectResponse` directly:

```
{!../../../docs_src/custom_response/tutorial006.py!}
```

Or you can use it in the `response_class` parameter:

```
{!../../../docs_src/custom_response/tutorial006b.py!}
```

If you do that, then you can return the URL directly from your *path operation* function.

In this case, the `status_code` used will be the default one for the `RedirectResponse`, which is `307`.

You can also use the `status_code` parameter combined with the `response_class` parameter:

```
{!../../../docs_src/custom_response/tutorial006c.py!}
```

`StreamingResponse`

Takes an async generator or a normal generator/iterator and streams the response body.

```
{!../../../docs_src/custom_response/tutorial007.py!}
```

Using `StreamingResponse` with file-like objects

If you have a file-like object (e.g. the object returned by `open()`), you can create a generator function to iterate over that file-like object.

That way, you don't have to read it all first in memory, and you can pass that generator function to the `StreamingResponse`, and return it.

This includes many libraries to interact with cloud storage, video processing, and others.

```
{!../../../docs_src/custom_response/tutorial008.py!}
```

1. This is the generator function. It's a "generator function" because it contains `yield` statements inside.
2. By using a `with` block, we make sure that the file-like object is closed after the generator function is done. So, after it finishes sending the response.
3. This `yield from` tells the function to iterate over that thing named `file_like`. And then, for each part iterated, yield that part as coming from this generator function.

   So, it is a generator function that transfers the "generating" work to something else internally.

   By doing it this way, we can put it in a `with` block, and that way, ensure that it is closed after finishing.

!!! tip Notice that here as we are using standard `open()` that doesn't support `async` and `await`, we declare the path operation with normal `def`.

`FileResponse`

Asynchronously streams a file as the response.

Takes a different set of arguments to instantiate than the other response types:

- `path` - The filepath to the file to stream.
- `headers` - Any custom headers to include, as a dictionary.
- `media_type` - A string giving the media type. If unset, the filename or path will be used to infer a media type.
- `filename` - If set, this will be included in the response `Content-Disposition`.

File responses will include appropriate `Content-Length`, `Last-Modified` and `ETag` headers.

```
{!../../../docs_src/custom_response/tutorial009.py!}
```

You can also use the `response_class` parameter:

```
{!../../../docs_src/custom_response/tutorial009b.py!}
```

In this case, you can return the file path directly from your *path operation* function.

Default response class

When creating a **FastAPI** class instance or an `APIRouter` you can specify which response class to use by default.

The parameter that defines this is `default_response_class`.

In the example below, **FastAPI** will use `ORJSONResponse` by default, in all *path operations*, instead of `JSONResponse`.

```
{!../../../docs_src/custom_response/tutorial010.py!}
```

!!! tip You can still override `response_class` in *path operations* as before.

Additional documentation

You can also declare the media type and many other details in OpenAPI using `responses`: Additional Responses in OpenAPI{.internal-link target=_blank}.