# Additional Responses in OpenAPI

!!! warning This is a rather advanced topic.

```
If you are starting with **FastAPI**, you might not need this.
```

You can declare additional responses, with additional status codes, media types, descriptions, etc.

Those additional responses will be included in the OpenAPI schema, so they will also appear in the API docs.

But for those additional responses you have to make sure you return a `Response` like `JSONResponse` directly, with your status code and content.

## Additional Response with `model`

You can pass to your *path operation decorators* a parameter `responses`.

It receives a `dict`, the keys are status codes for each response, like `200`, and the values are other `dict`s with the information for each of them.

Each of those response `dict`s can have a key `model`, containing a Pydantic model, just like `response_model`.

**FastAPI** will take that model, generate its JSON Schema and include it in the correct place in OpenAPI.

For example, to declare another response with a status code `404` and a Pydantic model `Message`, you can write:

`Python hl_lines="18  23" {!../../../docs_src/additional_responses/tutorial001.py!}`

!!! note Have in mind that you have to return the `JSONResponse` directly.

!!! info The `model` key is not part of OpenAPI.

```
**FastAPI** will take the Pydantic model from there, generate the `JSON Schema`, and put it

The correct place is:

* In the key `content`, that has as value another JSON object (`dict`) that contains:
    * A key with the media type, e.g. `application/json`, that contains as value another JS
        * A key `schema`, that has as the value the JSON Schema from the model, here's the c
            * **FastAPI** adds a reference here to the global JSON Schemas in another place
```

The generated responses in the OpenAPI for this *path operation* will be:

```
JSON hl_lines="3-12" {     "responses": {         "404": {             "description":
"Additional Response",             "content": {                 "application/json":
{                     "schema": {                         "$ref":
"#/components/schemas/Message"                     }                 }
}         },         "200": {             "description": "Successful
Response",            "content": {                 "application/json":
{                    "schema": {                            "$ref":
"#/components/schemas/Item"                      }                 }
}         },         "422": {           "description": "Validation
Error",            "content": {                 "application/json":
{                    "schema": {                            "$ref":
"#/components/schemas/HTTPValidationError"                     }
}            }         }     } }
```

The schemas are referenced to another place inside the OpenAPI schema:

```
JSON hl_lines="4-16" {     "components": {         "schemas":
{             "Message": {                "title": "Message",
"required": [                    "message"                ],
"type": "object",                "properties": {                     "message":
{                        "title": "Message",                         "type":
"string"                    }                }            },
"Item": {                "title": "Item",                "required":
[                    "id",                        "value"                ],
"type": "object",                "properties": {                     "id":
{                        "title": "Id",                         "type":
"string"                    },                    "value": {
"title": "Value",                        "type": "string"                     }
}            },            "ValidationError": {                "title":
"ValidationError",                "required": [                    "loc",
"msg",                    "type"                ],                "type":
"object",                "properties": {                    "loc":
{                        "title": "Location",                        "type":
"array",                        "items": {                            "type":
"string"                        }                    },                    "msg":
{                        "title": "Message",                        "type":
"string"                    },                    "type": {
"title": "Error Type",                        "type": "string"
}                }            },            "HTTPValidationError":
{                "title": "HTTPValidationError",                "type":
"object",                "properties": {                    "detail":
{                        "title": "Detail",                        "type":
"array",                        "items": {                            "$ref":
"#/components/schemas/ValidationError"                         }
}                }            }         }     } }
```

## Additional media types for the main response

You can use this same `responses` parameter to add different media types for the same main response.

For example, you can add an additional media type of `image/png`, declaring that your *path operation* can return a JSON object (with media type `application/json`) or a PNG image:

```
Python hl_lines="19-24  28" {!../../../docs_src/additional_responses/tutorial002.py!}
```

!!! note Notice that you have to return the image using a `FileResponse` directly.

!!! info Unless you specify a different media type explicitly in your `responses` parameter, FastAPI will assume the response has the same media type as the main response class (default `application/json`).

```
But if you have specified a custom response class with `None` as its media type, FastAPI wi
```

## Combining information

You can also combine response information from multiple places, including the `response_model`, `status_code`, and `responses` parameters.

You can declare a `response_model`, using the default status code `200` (or a custom one if you need), and then declare additional information for that same response in `responses`, directly in the OpenAPI schema.

**FastAPI** will keep the additional information from `responses`, and combine it with the JSON Schema from your model.

For example, you can declare a response with a status code `404` that uses a Pydantic model and has a custom `description`.

And a response with a status code `200` that uses your `response_model`, but includes a custom `example`:

```
Python hl_lines="20-31" {!../../../docs_src/additional_responses/tutorial003.py!}
```

It will all be combined and included in your OpenAPI, and shown in the API docs:

## Combine predefined responses and custom ones

You might want to have some predefined responses that apply to many *path operations*, but you want to combine them with custom responses needed by each *path operation*.

For those cases, you can use the Python technique of "unpacking" a `dict` with `**dict_to_unpack`:

```
old_dict = {
    "old key": "old value",
    "second old key": "second old value",
}
new_dict = {**old_dict, "new key": "new value"}
```

Here, `new_dict` will contain all the key-value pairs from `old_dict` plus the new key-value pair:

```
{
    "old key": "old value",
    "second old key": "second old value",
    "new key": "new value",
}
```

You can use that technique to re-use some predefined responses in your *path operations* and combine them with additional custom ones.

For example:

`Python hl_lines="13-17  26" {!../../../docs_src/additional_responses/tutorial004.py!}`

## More information about OpenAPI responses

To see what exactly you can include in the responses, you can check these sections in the OpenAPI specification:

- OpenAPI Responses Object, it includes the `Response Object`.
- OpenAPI Response Object, you can include anything from this directly in each response inside your `responses` parameter. Including `description`, `headers`, `content` (inside of this is that you declare different media types and JSON Schemas), and `links`.