Response Model

You can declare the model used for the response with the parameter `response_model` in any of the *path operations*:

- `@app.get()`
- `@app.post()`
- `@app.put()`
- `@app.delete()`
- etc.

=== "Python 3.6 and above"

```
```Python hl_lines="17"
{!> ../../../docs_src/response_model/tutorial001.py!}
```
```

=== "Python 3.9 and above"

```
```Python hl_lines="17"
{!> ../../../docs_src/response_model/tutorial001_py39.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="15"
{!> ../../../docs_src/response_model/tutorial001_py310.py!}
```
```

!!! note Notice that `response_model` is a parameter of the "decorator" method (`get` , `post` , etc). Not of your *path operation function*, like all the parameters and body.

It receives the same type you would declare for a Pydantic model attribute, so, it can be a Pydantic model, but it can also be, e.g. a `list` of Pydantic models, like `List[Item]` .

FastAPI will use this `response_model` to:

- Convert the output data to its type declaration.
- Validate the data.
- Add a JSON Schema for the response, in the OpenAPI *path operation*.
- Will be used by the automatic documentation systems.

But most importantly:

- Will limit the output data to that of the model. We'll see how that's important below.

!!! note "Technical Details" The response model is declared in this parameter instead of as a function return type annotation, because the path function may not actually return that response model but rather return a `dict` , database object or some other model, and then use the `response_model` to perform the field limiting and serialization.

Return the same input data

Here we are declaring a `UserIn` model, it will contain a plaintext password:

=== "Python 3.6 and above"

```Python hl_lines="9  11"
{!> ../../../docs_src/response_model/tutorial002.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="7  9"
{!> ../../../docs_src/response_model/tutorial002_py310.py!}
```

And we are using this model to declare our input and the same model to declare our output:

=== "Python 3.6 and above"

```Python hl_lines="17-18"
{!> ../../../docs_src/response_model/tutorial002.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="15-16"
{!> ../../../docs_src/response_model/tutorial002_py310.py!}
```

Now, whenever a browser is creating a user with a password, the API will return the same password in the response.

In this case, it might not be a problem, because the user themself is sending the password.

But if we use the same model for another *path operation*, we could be sending our user's passwords to every client.

!!! danger Never store the plain password of a user or send it in a response.

Add an output model

We can instead create an input model with the plaintext password and an output model without it:

=== "Python 3.6 and above"

```Python hl_lines="9  11  16"
{!> ../../../docs_src/response_model/tutorial003.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="7  9  14"
{!> ../../../docs_src/response_model/tutorial003_py310.py!}
```

Here, even though our *path operation function* is returning the same input user that contains the password:

=== "Python 3.6 and above"

```Python hl_lines="24"
{!> ../../../docs_src/response_model/tutorial003.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="22"
{!> ../../../docs_src/response_model/tutorial003_py310.py!}
```

...we declared the `response_model` to be our model `UserOut`, that doesn't include the password:

=== "Python 3.6 and above"

```Python hl_lines="22"
{!> ../../../docs_src/response_model/tutorial003.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="20"
{!> ../../../docs_src/response_model/tutorial003_py310.py!}
```

So, **FastAPI** will take care of filtering out all the data that is not declared in the output model (using Pydantic).

See it in the docs

When you see the automatic docs, you can check that the input model and output model will both have their own JSON Schema:

And both models will be used for the interactive API documentation:

Response Model encoding parameters

Your response model could have default values, like:

=== "Python 3.6 and above"

```Python hl_lines="11  13-14"
{!> ../../../docs_src/response_model/tutorial004.py!}
```

=== "Python 3.9 and above"

```Python hl_lines="11  13-14"
{!> ../../../docs_src/response_model/tutorial004_py39.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="9  11-12"
{!> ../../../docs_src/response_model/tutorial004_py310.py!}
```

- `description: Optional[str] = None` has a default of `None`.
- `tax: float = 10.5` has a default of `10.5`.
- `tags: List[str] = []` as a default of an empty list: `[]`.

but you might want to omit them from the result if they were not actually stored.

For example, if you have models with many optional attributes in a NoSQL database, but you don't want to send very long JSON responses full of default values.

Use the `response_model_exclude_unset` parameter

You can set the *path operation decorator* parameter `response_model_exclude_unset=True`:

=== "Python 3.6 and above"

```Python hl_lines="24"
{!> ../../../docs_src/response_model/tutorial004.py!}
```

=== "Python 3.9 and above"

```Python hl_lines="24"
{!> ../../../docs_src/response_model/tutorial004_py39.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="22"
{!> ../../../docs_src/response_model/tutorial004_py310.py!}
```

and those default values won't be included in the response, only the values actually set.

So, if you send a request to that *path operation* for the item with ID `foo`, the response (not including default values) will be:

```JSON
{
    "name": "Foo",
    "price": 50.2
}
```

!!! info FastAPI uses Pydantic model's `.dict()` with its `exclude_unset` parameter to achieve this.

!!! info You can also use:

```
* `response_model_exclude_defaults=True`
* `response_model_exclude_none=True`

as described in <a href="https://pydantic-
docs.helpmanual.io/usage/exporting_models/#modeldict" class="external-link"
target="_blank">the Pydantic docs</a> for `exclude_defaults` and `exclude_none`.
```

Data with values for fields with defaults

But if your data has values for the model's fields with default values, like the item with ID `bar`:

```
{
    "name": "Bar",
    "description": "The bartenders",
    "price": 62,
    "tax": 20.2
}
```

they will be included in the response.

Data with the same values as the defaults

If the data has the same values as the default ones, like the item with ID `baz`:

```
{
    "name": "Baz",
    "description": None,
    "price": 50.2,
    "tax": 10.5,
    "tags": []
}
```

FastAPI is smart enough (actually, Pydantic is smart enough) to realize that, even though `description`, `tax`, and `tags` have the same values as the defaults, they were set explicitly (instead of taken from the defaults).

So, they will be included in the JSON response.

!!! tip Notice that the default values can be anything, not only `None`.

```
They can be a list (`[]`), a `float` of `10.5`, etc.
```

`response_model_include` and `response_model_exclude`

You can also use the *path operation decorator* parameters `response_model_include` and `response_model_exclude`.

They take a `set` of `str` with the name of the attributes to include (omitting the rest) or to exclude (including the rest).

This can be used as a quick shortcut if you have only one Pydantic model and want to remove some data from the output.

!!! tip But it is still recommended to use the ideas above, using multiple classes, instead of these parameters.

```
This is because the JSON Schema generated in your app's OpenAPI (and the docs) will
still be the one for the complete model, even if you use `response_model_include` or
`response_model_exclude` to omit some attributes.

This also applies to `response_model_by_alias` that works similarly.
```

=== "Python 3.6 and above"

```
```Python hl_lines="31  37"
{!> ../../../docs_src/response_model/tutorial005.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="29  35"
{!> ../../../docs_src/response_model/tutorial005_py310.py!}
```
```

!!! tip The syntax `{"name", "description"}` creates a `set` with those two values.

```
It is equivalent to `set(["name", "description"])`.
```

Using `list`s instead of `set`s

If you forget to use a `set` and use a `list` or `tuple` instead, FastAPI will still convert it to a `set` and it will work correctly:

=== "Python 3.6 and above"

```
```Python hl_lines="31  37"
{!> ../../../docs_src/response_model/tutorial006.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="29  35"
{!> ../../../docs_src/response_model/tutorial006_py310.py!}
```
```

Recap

Use the *path operation decorator*'s parameter `response_model` to define response models and especially to ensure private data is filtered out.

Use `response_model_exclude_unset` to return only the values explicitly set.