Cookie Parameter Models

If you have a group of **cookies** that are related, you can create a **Pydantic model** to declare them. 🍪

This would allow you to **re-use the model** in **multiple places** and also to declare validations and metadata for all the parameters at once. 😎

Note

This is supported since FastAPI version `0.115.0`. 🤓

Tip

This same technique applies to `Query`, `Cookie`, and `Header`. 😎

Cookies with a Pydantic Model

Declare the **cookie** parameters that you need in a **Pydantic model**, and then declare the parameter as `Cookie`:

Python 3.10+

```
from typing import Annotated

from fastapi import Cookie, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Cookies(BaseModel):
    session_id: str
    fatebook_tracker: str | None = None
    googall_tracker: str | None = None


@app.get("/items/")
async def read_items(cookies: Annotated[Cookies, Cookie()]):
    return cookies
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Cookie, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Cookies(BaseModel):
    session_id: str
    fatebook_tracker: Union[str, None] = None
    googall_tracker: Union[str, None] = None


@app.get("/items/")
async def read_items(cookies: Annotated[Cookies, Cookie()]):
    return cookies
```

Python 3.8+

```
from typing import Union

from fastapi import Cookie, FastAPI
from pydantic import BaseModel
from typing_extensions import Annotated

app = FastAPI()


class Cookies(BaseModel):
    session_id: str
    fatebook_tracker: Union[str, None] = None
    googall_tracker: Union[str, None] = None


@app.get("/items/")
async def read_items(cookies: Annotated[Cookies, Cookie()]):
    return cookies
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Cookie, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Cookies(BaseModel):
    session_id: str
    fatebook_tracker: str | None = None
    googall_tracker: str | None = None


@app.get("/items/")
async def read_items(cookies: Cookies = Cookie()):
    return cookies
```

Python 3.8+ - non-Annotated

Tip

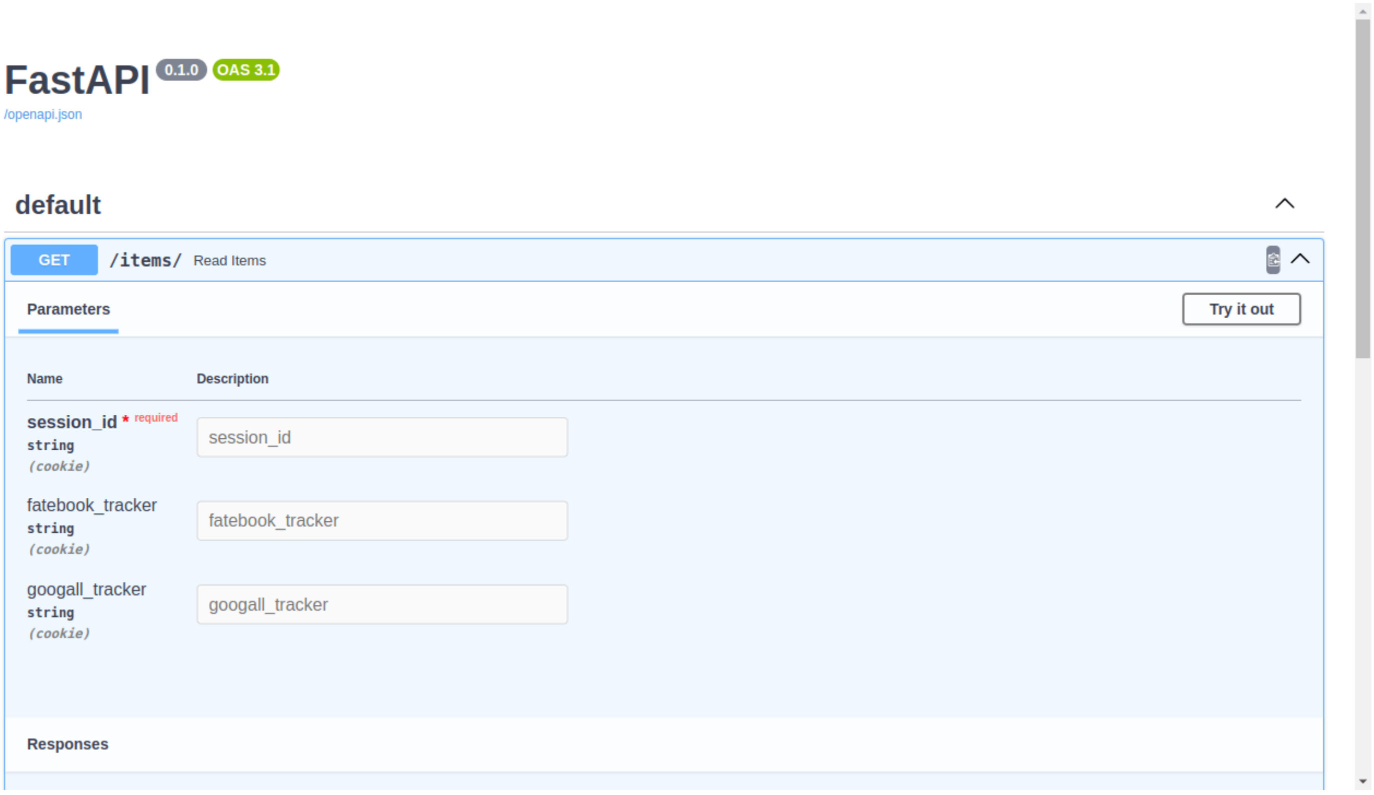
Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Cookie, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Cookies(BaseModel):
    session_id: str
    fatebook_tracker: Union[str, None] = None
    googall_tracker: Union[str, None] = None


@app.get("/items/")
async def read_items(cookies: Cookies = Cookie()):
    return cookies
```

FastAPI will **extract** the data for **each field** from the **cookies** received in the request and give you the Pydantic model you defined.

Check the Docs

You can see the defined cookies in the docs UI at `/docs` :



FastAPI 0.1.0 OAS 3.1
/openapi.json

default

GET /items/ Read Items

Parameters

Name	Description
session_id * required string (cookie)	session_id
fatebook_tracker string (cookie)	fatebook_tracker
googall_tracker string (cookie)	googall_tracker

Responses

Info

Have in mind that, as **browsers handle cookies** in special ways and behind the scenes, they **don't** easily allow **JavaScript** to touch them.

If you go to the **API docs UI** at `/docs` you will be able to see the **documentation** for cookies for your *path operations*.

But even if you **fill the data** and click "Execute", because the docs UI works with **JavaScript**, the cookies won't be sent, and you will see an **error** message as if you didn't write any values.

Forbid Extra Cookies

In some special use cases (probably not very common), you might want to **restrict** the cookies that you want to receive.

Your API now has the power to control its own cookie consent. 🤪🍪

You can use Pydantic's model configuration to `forbid` any `extra` fields:

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Cookie, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Cookies(BaseModel):
    model_config = {"extra": "forbid"}

    session_id: str
    fatebook_tracker: Union[str, None] = None
    googall_tracker: Union[str, None] = None


@app.get("/items/")
async def read_items(cookies: Annotated[Cookies, Cookie()]):
    return cookies
```

▸ 🤓 Other versions and variants

Python 3.10+

```
from typing import Annotated

from fastapi import Cookie, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Cookies(BaseModel):
    model_config = {"extra": "forbid"}

    session_id: str
    fatebook_tracker: str | None = None
    googall_tracker: str | None = None


@app.get("/items/")
async def read_items(cookies: Annotated[Cookies, Cookie()]):
    return cookies
```

Python 3.8+

```
from typing import Union

from fastapi import Cookie, FastAPI
from pydantic import BaseModel
from typing_extensions import Annotated

app = FastAPI()


class Cookies(BaseModel):
    model_config = {"extra": "forbid"}

    session_id: str
    fatebook_tracker: Union[str, None] = None
    googall_tracker: Union[str, None] = None


@app.get("/items/")
async def read_items(cookies: Annotated[Cookies, Cookie()]):
    return cookies
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Cookie, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Cookies(BaseModel):
    model_config = {"extra": "forbid"}

    session_id: str
    fatebook_tracker: str | None = None
    googall_tracker: str | None = None


@app.get("/items/")
async def read_items(cookies: Cookies = Cookie()):
    return cookies
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Cookie, FastAPI
from pydantic import BaseModel

app = FastAPI()


class Cookies(BaseModel):
    model_config = {"extra": "forbid"}

    session_id: str
    fatebook_tracker: Union[str, None] = None
    googall_tracker: Union[str, None] = None


@app.get("/items/")
async def read_items(cookies: Cookies = Cookie()):
    return cookies
```

If a client tries to send some **extra cookies**, they will receive an **error** response.

Poor cookie banners with all their effort to get your consent for the API to reject it. 🍪

For example, if the client tries to send a `santa_tracker` cookie with a value of `good-list-please`, the client will receive an **error** response telling them that the `santa_tracker` cookie is not allowed:

```
{
    "detail": [
        {
            "type": "extra_forbidden",
            "loc": ["cookie", "santa_tracker"],
            "msg": "Extra inputs are not permitted",
            "input": "good-list-please",
        }
    ]
}
```

Summary

You can use **Pydantic models** to declare cookies in **FastAPI**. 😎