Form Models

You can use **Pydantic models** to declare **form fields** in FastAPI.

Info

To use forms, first install `python-multipart`.

Make sure you create a virtual environment, activate it, and then install it, for example:

```
$ pip install python-multipart
```

Note

This is supported since FastAPI version `0.113.0`. 🤓

Pydantic Models for Forms

You just need to declare a **Pydantic model** with the fields you want to receive as **form fields**, and then declare the parameter as `Form`:

Python 3.9+

```
from typing import Annotated

from fastapi import FastAPI, Form
from pydantic import BaseModel

app = FastAPI()


class FormData(BaseModel):
    username: str
    password: str


@app.post("/login/")
async def login(data: Annotated[FormData, Form()]):
    return data
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from fastapi import FastAPI, Form
from pydantic import BaseModel
from typing_extensions import Annotated

app = FastAPI()


class FormData(BaseModel):
    username: str
    password: str


@app.post("/login/")
async def login(data: Annotated[FormData, Form()]):
    return data
```

Python 3.8+ - non-Annotated

Tip

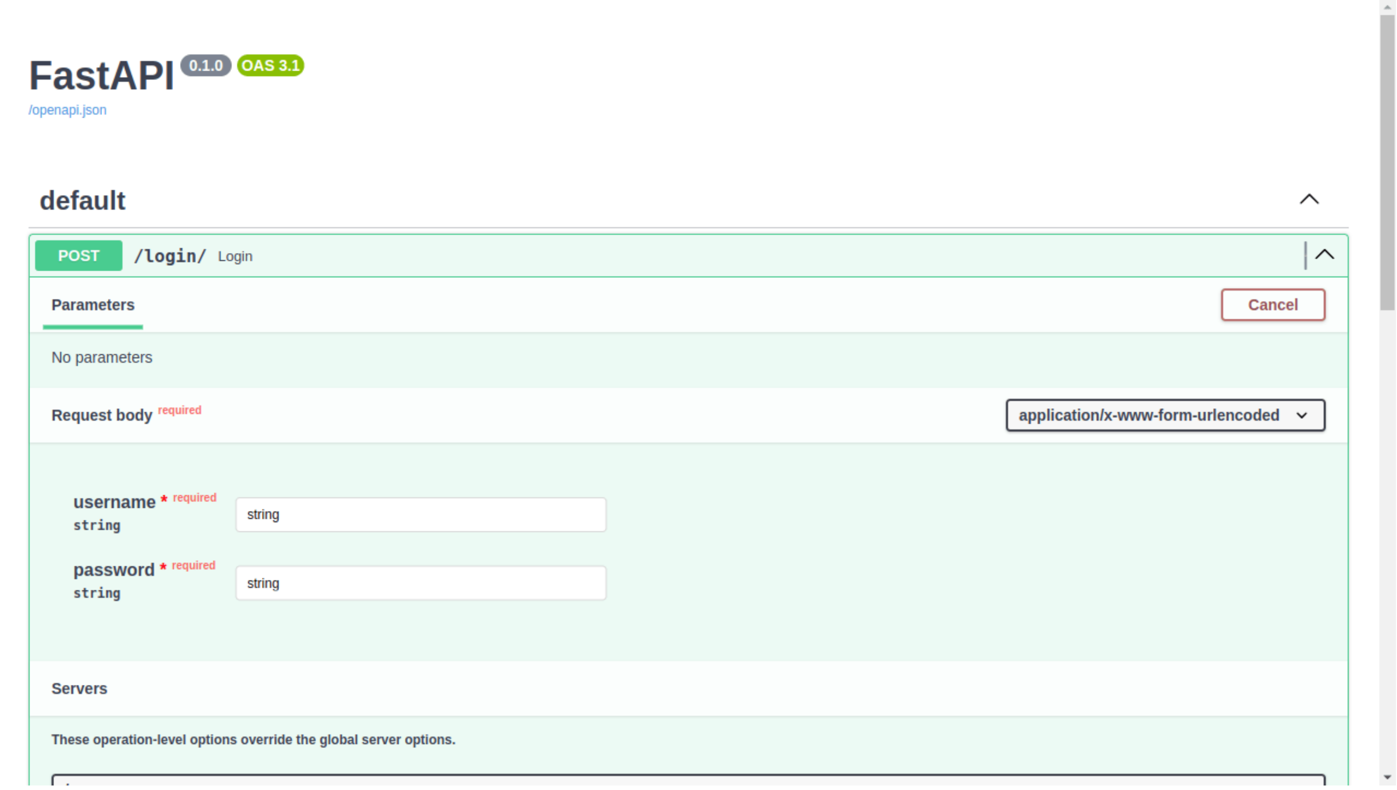
Prefer to use the `Annotated` version if possible.

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

app = FastAPI()


class FormData(BaseModel):
    username: str
    password: str


@app.post("/login/")
async def login(data: FormData = Form()):
    return data
```

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

Check the Docs

You can verify it in the docs UI at `/docs` :



These operation-level options override the global server options.

Forbid Extra Form Fields

In some special use cases (probably not very common), you might want to **restrict** the form fields to only those declared in the Pydantic model. And **forbid** any **extra** fields.

Note

This is supported since FastAPI version `0.114.0` . 🤓

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

Python 3.9+

```
from typing import Annotated

from fastapi import FastAPI, Form
from pydantic import BaseModel

app = FastAPI()


class FormData(BaseModel):
    username: str
    password: str
    model_config = {"extra": "forbid"}


@app.post("/login/")
async def login(data: Annotated[FormData, Form()]):
    return data
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from fastapi import FastAPI, Form
from pydantic import BaseModel
from typing_extensions import Annotated

app = FastAPI()


class FormData(BaseModel):
    username: str
    password: str
    model_config = {"extra": "forbid"}
```

```
@app.post("/login/")
async def login(data: Annotated[FormData, Form()]):
    return data
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

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

app = FastAPI()


class FormData(BaseModel):
    username: str
    password: str
    model_config = {"extra": "forbid"}


@app.post("/login/")
async def login(data: FormData = Form()):
    return data
```

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

For example, if the client tries to send the form fields:

- `username` : `Rick`
- `password` : `Portal Gun`
- `extra` : `Mr. Poopybutthole`

They will receive an error response telling them that the field `extra` is not allowed:

```
{
    "detail": [
        {
            "type": "extra_forbidden",
            "loc": ["body", "extra"],
            "msg": "Extra inputs are not permitted",
            "input": "Mr. Poopybutthole"
        }
    ]
}
```

Summary

You can use Pydantic models to declare form fields in FastAPI. 😎