

Query Parameter Models

If you have a group of query parameters 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 . 😎

Query Parameters with a Pydantic Model

Declare the query parameters that you need in a Pydantic model, and then declare the parameter as `Query` :

Python 3.10+

```
from typing import Annotated, Literal

from fastapi import FastAPI, Query
from pydantic import BaseModel, Field

app = FastAPI()

class FilterParams(BaseModel):
    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: list[str] = []

@app.get("/items/")
async def read_items(filter_query: Annotated[FilterParams, Query()]):
    return filter_query
```

► 😎 Other versions and variants

Python 3.9+

```
from fastapi import FastAPI, Query
from pydantic import BaseModel, Field
from typing_extensions import Annotated, Literal

app = FastAPI()

class FilterParams(BaseModel):
    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: list[str] = []

@app.get("/items/")
async def read_items(filter_query: Annotated[FilterParams, Query()]):
    return filter_query
```

Python 3.8+

```
from typing import List

from fastapi import FastAPI, Query
from pydantic import BaseModel, Field
from typing_extensions import Annotated, Literal

app = FastAPI()

class FilterParams(BaseModel):
    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: List[str] = []

@app.get("/items/")
async def read_items(filter_query: Annotated[FilterParams, Query()]):
    return filter_query
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Literal

from fastapi import FastAPI, Query
from pydantic import BaseModel, Field

app = FastAPI()

class FilterParams(BaseModel):
    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: list[str] = []

@app.get("/items/")
async def read_items(filter_query: FilterParams = Query()):
    return filter_query
```

Python 3.9+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query
from pydantic import BaseModel, Field
from typing_extensions import Literal

app = FastAPI()

class FilterParams(BaseModel):
    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: list[str] = []

@app.get("/items/")
async def read_items(filter_query: FilterParams = Query()):
    return filter_query
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import List

from fastapi import FastAPI, Query
from pydantic import BaseModel, Field
from typing_extensions import Literal

app = FastAPI()

class FilterParams(BaseModel):
    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: List[str] = []

@app.get("/items/")
async def read_items(filter_query: FilterParams = Query()):
    return filter_query
```

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

Check the Docs

You can see the query parameters in the docs UI at `/docs` :

FastAPI 0.1.0 OAS 3.1

[/openapi.json](#)

default

GET [/items/](#) Read Items

Cancel

Parameters

Name	Description
limit integer (query)	100 maximum: 100
offset integer (query)	0 minimum: 0
order_by string (query)	created_at
tags array[string] (query)	Add string item

Servers

Forbid Extra Query Parameters

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

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

Python 3.10+

```
from typing import Annotated, Literal

from fastapi import FastAPI, Query
from pydantic import BaseModel, Field

app = FastAPI()

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

    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: list[str] = []

@app.get("/items/")
async def read_items(filter_query: Annotated[FilterParams, Query()]):
    return filter_query
```

► 📸 Other versions and variants

Python 3.9+

```
from fastapi import FastAPI, Query
from pydantic import BaseModel, Field
from typing_extensions import Annotated, Literal

app = FastAPI()

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

    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: list[str] = []

@app.get("/items/")
async def read_items(filter_query: Annotated[FilterParams, Query()]):
    return filter_query
```

Python 3.8+

```
from typing import List

from fastapi import FastAPI, Query
from pydantic import BaseModel, Field
from typing_extensions import Annotated, Literal

app = FastAPI()

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

    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: List[str] = []

@app.get("/items/")
async def read_items(filter_query: Annotated[FilterParams, Query()]):
    return filter_query
```

Python 3.10+ - non-Annotated**Tip**

Prefer to use the `Annotated` version if possible.

```
from typing import Literal

from fastapi import FastAPI, Query
from pydantic import BaseModel, Field

app = FastAPI()

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

    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: list[str] = []

@app.get("/items/")
async def read_items(filter_query: FilterParams = Query()):
    return filter_query
```

Python 3.9+ - non-Annotated**Tip**

Prefer to use the `Annotated` version if possible.

```
from fastapi import FastAPI, Query
from pydantic import BaseModel, Field
from typing_extensions import Literal

app = FastAPI()

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

    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: list[str] = []

@app.get("/items/")
async def read_items(filter_query: FilterParams = Query()):
    return filter_query
```

Python 3.8+ - non-Annotated**Tip**

Prefer to use the `Annotated` version if possible.

```
from typing import List

from fastapi import FastAPI, Query
from pydantic import BaseModel, Field
from typing_extensions import Literal

app = FastAPI()

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

    limit: int = Field(100, gt=0, le=100)
    offset: int = Field(0, ge=0)
    order_by: Literal["created_at", "updated_at"] = "created_at"
    tags: List[str] = []

@app.get("/items/")
async def read_items(filter_query: FilterParams = Query()):
    return filter_query
```

If a client tries to send some extra data in the query parameters, they will receive an error response.

For example, if the client tries to send a `tool` query parameter with a value of `plumbus`, like:

```
https://example.com/items/?limit=10&tool=plumbus
```

They will receive an error response telling them that the query parameter `tool` is not allowed:

```
{
  "detail": [
    {
      "type": "extra_forbidden",
      "loc": ["query", "tool"],
      "msg": "Extra inputs are not permitted",
      "input": "plumbus"
    }
  ]
}
```

Summary

You can use Pydantic models to declare query parameters in FastAPI. 😎

Tip

Spoiler alert: you can also use Pydantic models to declare cookies and headers, but you will read about that later in the tutorial. 🤫

© 2018 Sebastián Ramírez
Licensed under the MIT License.
<https://fastapi.tiangolo.com/tutorial/query-param-models/>

Exported from DevDocs — <https://devdocs.io>