victoree

Python/Fast API

[2] Fast API Basic - Request 편

victoree | 2021. 7. 20. 15:41

1. Path Parameters

1) 순서 문제

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/users/me")
async def get_my_info():
    return {"user_id": "the current user"}

@app.get("/users/{user_id}")
async def get_user(user_id: int):
    return {"user_id": user_id}
```

- 위 함수를 정의하는 순서가 중요함
- `get_user` 와 `get_my_info` 의 순서가 변경되면,
  user_id의 값을 "me" 라고 생각해 /users/me도 user_id에 연결됨

2) 사전 정의 값

```
from enum import Enum
from fastapi import FastAPI

class BookCategory(str, Enum):
    IT = "it"
    NATURE = "nature"
    HISTORY = "history"

app = FastAPI()

@app.get("/books/category/{category}")
async def get_book(category: BookCategory):
    if category == BookCategory.IT:
          return {"This category is deprecated :: ": category}

    return {"book's category is ": category}
```

3) 메타데이터 작성

```
from fastapi import FastAPI, Path

item_id: int = Path(..., title="The ID of the item to get")
```

- gt, lt 옵션으로 int 또는 float 과 같은 수 validation을 지정할 수 있음 (Query 객체도 마찬가지)
  - gt: greater than
  - ge: greater than or equal
  - lt: less than
  - le: less than or equal

2. Query Parameters

1) query_params default 작동방식

```
from fastapi import FastAPI

app = FastAPI()

fake_books_db = [{"book_name": "Foo"}, {"book_name": "Bar"}, {"book_name": "Baz"}]

@app.get("/books/")
async def get_book(skip: int = 0, limit: int = 10):
    return fake_books_db[skip : skip + limit]
```

- http://127.0.0.1:8000/items/ 를 호출하면, 디폴트로 아래 주소
  - http://127.0.0.1:8000/items/?skip=0&limit=10 가 호출된다.
  - default 값이 지정되어있기 때문이다.

- http://127.0.0.1:8000/items/?skip=20 을 호출하면,
  - skip = 20, limit = 10으로 호출된다.

- 기본값을 None으로 설정하면, 이는 Optional parameter가 됨
  - ex) `q: Optional[str] = None`

2) Additional Validation

```
from fastapi import FastAPI, Query

app = FastAPI()

@app.get("/books/")
async def get_book(q: Optional[str] = Query(None, min_length=3, max_length=50, regex="^f
        results = {"books": [{"book_name": "Foo"}, {"book_name": "Bar"}, {"book_name": '
    if q:
        results.update({"q": q})
        return results
```

- Query 객체를 이용하여 query_params에 추가적인 Validation을 할 수 있음
- min_length, max_length를 설정하는 것 뿐만 아니라, regex 설정도 가능
- None 위치에 default str을 넣을 수도 있다 → 그럼 Optional 이 필요없어지겠지.

- **required** 세팅을 하되, 조건을 주고싶으면, Ellipsis 객체 `...` 으로 해결할 수 있음
  - `q: str = Query(..., min_length=3)`
  - Ellipsis 객체는 **작동없음**을 기술하는 데 사용(pass와 비슷)
  - https://docs.python.org/3/library/constants.html#Ellipsis
  - https://tech.madup.com/python-ellipsis/

- List[str] 타입으로 query를 받을 수 있음
  - `http://localhost:8000/items/?q=foo&q=bar`
  - `{ "q": [ "foo", "bar" ] }`

3) 메타데이터 작성

```
q: Optional[str] = Query(
        None,
        title="Query string",
        description="Query string for the items to search in the database that have a go
        min_length=3,
)
```

- title이나 description으로 메타 데이터 작성

3. Request Body

1) BaseModel 기반 인자 정의

```
from typing import Optional
from fastapi import FastAPI
from pydantic import BaseModel

class Book(BaseModel):
    name: str
    content: Optional[str] = None
    price: int

app = FastAPI()

@app.post("/books/")
async def create_book(book: Book):
    return book
```

- 위와 같이 BaseModel을 상속하여 Book이라는 새로운 모델을 생성
- 이 모델을 함수의 파라미터로 등록
- 아래를 request body로 설정하여 API 호출

```
{
    "name": "Foo",
    "content": "An optional description",
    "price": 15000
}
```

1. request body를 읽음
2. 필요하다면, 타입들을 상응하는 값으로 전환

3. 데이터 벨리데이션 수행
4. book이라는 파라미터에 전달받은 데이터를 넘겨줌
5. JSON Schema로 정의해줌

2) path parameter도 사용하고, query도 쓰고, body도 쓰고 싶을 때!

```
from typing import Optional
from fastapi import FastAPI
from pydantic import BaseModel

class Book(BaseModel):
    name: str
    content: Optional[str] = None
    price: int

app = FastAPI()

@app.put("/books/{book_id}")
async def create_book(
        *,
        book_id: int = Path(..., title="The ID of the book to get", ge=0, le=1000),
        q: Optional[str] = None,
        book: Optional[Book] = None
):
    return book
```

3) Body 에 두 가지 모델 객체를 받을 때

```
from typing import Optional

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Item(BaseModel):
    name: str
    description: Optional[str] = None
    price: float
    tax: Optional[float] = None

class User(BaseModel):
    username: str
    full_name: Optional[str] = None

@app.put("/items/{item_id}")
async def update_item(item_id: int, item: Item, user: User):
    results = {"item_id": item_id, "item": item, "user": user}
    return results
```

- 위와 같이 Item 객체와 User 객체 두 가지를 Parameter로 받으면,

```
{
    "item": {
        "name": "Foo",
        "description": "The pretender",
        "price": 42.0,
        "tax": 3.2
    },
    "user": {
```

```
        "username": "dave",
        "full_name": "Dave Grohl"
    }
}
```

- 이와 같이, parameter 이름을 키로 갖는 body로 보내주어야 한다.

4) Body 에 단일 parameter를 넣고싶다면?

```
from fastapi import Body, FastAPI

@app.put("/items/{item_id}")
async def update_item(
    item_id: int, item: Item, user: User,
        importance: int = Body(...)
):
```

- Body 객체를 이용해 importance라는 parameter를 body에 추가할 수 있다

5) Single Parameter Embed 추가하기

- Single Parameter body는 응답값이 모델의 키값으로만 돌아옴

```
{
    "name": "Foo",
    "content": "An optional description",
    "price": 15000
}
```

- 키값으로 Parameter 명을 추가하고 싶다면?

```
@app.put("/books/{book_id}")
async def update_book(
        book_id: int,
        book: Book = Body(..., embed=True)
):
    results = {"book_id": book_id, "book": book}
    return results
```

- embed라는 옵션을 추가하면

```
{
        "book_id": {book_id},
    "book": {
                "name": "Foo",
            "content": "An optional description",
            "price": 15000
    }
}
```

4. Header나 Cookie

“

Header와 Cookie는 Path, Query의 "자매"클래스입니다. 이 역시 동일한 공통 Param 클래스를 상속
다.
Query, Path, Header 그리고 다른 것들을 fastapi에서 임포트 할 때, 이들은 실제로 특별한 클래스를 반
는 함수임을 기억하세요.

1) Header

- 대부분의 표준 헤더는 "마이너스 기호(-)" 라고도 하고, "하이픈"이라고 하는 문자로 구분된다
- BUT 파이썬에서는 기본적으로 snake_case를 사용함!
- Header는 기본적으로 매개변수 이름을 언더스코어에서 하이픈으로 변환하여 헤더를 추출하고 기록함
  - user_agent → user-agent로 해석
- 만약 이 자동변환을 비활성화하고 싶을땐, `convert_underscores = False`로 설정
  - Header(None, convert_underscores=False)
- 중복 헤더값의 경우, List로 수신하면 됨
  - x_token: Optional[List[str]] = Header(None)

2) Cookie

- Header나 Path, Query 처럼 fastapi 에서 import하여 사용하면 됨..

5. Model

1) Model Field는!

```
from typing import Optional
from pydantic import BaseModel, Field

app = FastAPI()

class Book(BaseModel):
    name: str
    content: Optional[str] = Field(
        None, title="The content of the book", max_length=1000
    )
    price: int = Field(..., gt=0, description="The price must be greater than zero")
```

- 사실 상, Field는 Body나 Path, Query와 동일하게 작동함
  - Query나 Path는 Pydantic's FieldInfo 클래스의 하위 클래스인 Param class의 하위 클래스의 개체를 생성한다.
  - pydantic의 field는 FieldInfo의 인스턴스도 반환한다
  - Body 또한 FieldInfo의 하위 클래스 객체를 직접 반환한다.
  - Query, Path 등은 실제로 특수 클래스를 반환하는 함수일 뿐이다.

2) Nested Models

1. list, dict, tuple를 선언하고 싶다면!
2. 해당 자료형의 타입을 전달하기 위해선 [,]을 이용!

```python
from typing import List # typing 에 있음

my_list: List[str]
```

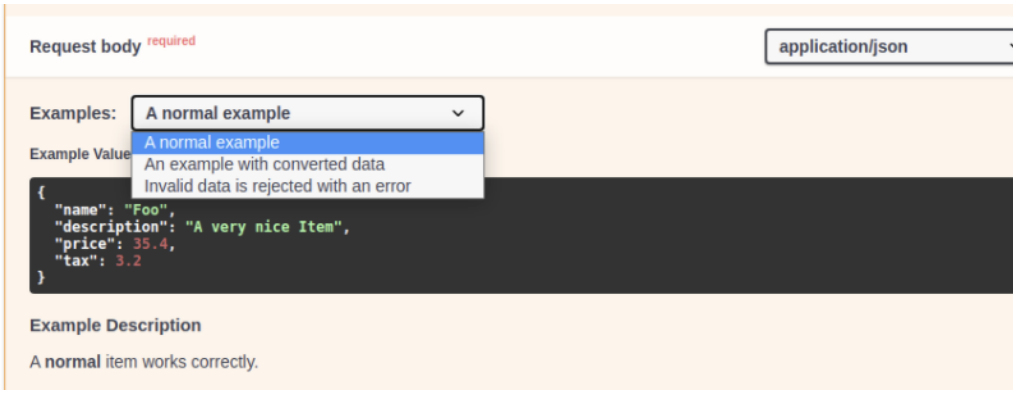
3. 중첩 모델은 다음과 같이 정의할 수 있음

```python
from pydantic import BaseModel, Field

class Image(BaseModel):
    url: str
    name: str

class Item(BaseModel):
    name: str
    description: Optional[str] = None
    price: float = **Field**(..., **example**=35.4)
    tax: Optional[float] = None
    tags: Set[str] = []
    **image: Optional[Image] = None # Optional 빼고 정의하면 required인 상태**
```

- Field 객체를 이용해서, help_text 처럼 example을 추가할 수 있음
- examples

```python
@app.put("/items/{item_id}")
async def update_item(
    *,
    item_id: int,
    item: Item = Body(
        ...,
        examples={
            "normal": {
                "summary": "A normal example",
                "description": "A **normal** item works correctly.",
                "value": {
                    "name": "Foo",
                    "description": "A very nice Item",
                    "price": 35.4,
                    "tax": 3.2,
                },
            },
            "converted": {
                "summary": "An example with converted data",
                "description": "FastAPI can convert price `strings` to actual `numbers`
                "value": {
                    "name": "Bar",
                    "price": "35.4",
                },
            },
            "invalid": {
                "summary": "Invalid data is rejected with an error",
                "value": {
                    "name": "Baz",
                    "price": "thirty five point four",
                },
            },
        },
    ),
):
    results = {"item_id": item_id, "item": item}
    return results
```

- 이것처럼 examples를 작성해놓으면, 아래처럼 example도 옵션처럼 선택해서 테스트할 수 있음



3) Extra Data Types

[

Extra Data Types - FastAPI

Warning The current page still doesn't have a translation for this language. But you can help translating it: Contributing. Up to now, you have been using common data types, like: But you can also use more complex d types. And you will still have the sa

fastapi.tiangolo.com

](https://fastapi.tiangolo.com/ko/tutorial/extra-data-types/)

- `frozenset`:
  - In requests and responses, treated the same as a `set`:
    - 요청에서 목록을 읽고, 중복을 제거하여 `set` 으로 변환
    - 응답에서는 `set` 이 `list` 로 변환됨
    - 생성된 스키마는 설정 값이 고유하도록 (JSON schema의 uniqueItems를 사용) 지정

6. Form

1) Form Field

- OAuth2 스펙으로 username이랑 password를 form fields로 보내라고 함 (password flow)
- Form 은 Body 클래스를 직접 상속 받은 클래스
  - Body, Query, Path, Cookie처럼 벨리데이션, 메타데이터를 설정할 수 있음

- HTML form 양식은 데이터를 특별한 인코딩을 거쳐 보냄
- 데이터는 일반적으로 `application/x-www-form-urlencoded` 을 사용하여 인코딩됨
- 양식에 파일이 포함되면, multipart/form-data로 인코딩됨
- `from fastapi import FastAPI, File, UploadFile app = FastAPI() @app.post("/files/") asy def create_file(file: bytes = File(...)): return {"file_size": len(file)} @app.post("/uploadfile/") async def create_upload_file(file: UploadFile = File(...)): return {"filename": file.filename}`

2) file을 아래와 같이 받을 수 있음

1. **File 객체**
   - file을 bytes로 선언하면, 내용을 bytes로 받음
   - 전체 내용이 메모리에 저장되며, 작은 파일에 잘 작동함

2. **Upload File 객체**
   - spooled file을 사용
   - 최대 크기 제한까지 메모리에 저장된 파일이며, 이 제한을 초과하면 디스크에 저장함
   - 모든 메모리를 소비하지 않아도, 이미지, 비디오, 대용량 바이너리와 같은 대용량 파일에서 잘 작동함
   - 파일과 같은 비동기 인터페이스가 있음
   - 파일류 객체를 기대하는 다른 라이브러리에 직접 전달할 수 있는 실제 SpooledTemporaryFile 객체를 노출

7. Request 객체 직접 받기

```python
from fastapi import FastAPI, Request

app = FastAPI()

@app.get("/items/{item_id}")
def read_root(item_id: str, request: Request):
    client_host = request.client.host
    return {"client_host": client_host, "item_id": item_id}
```


'Python > Fast API' 카테고리의 다른 글

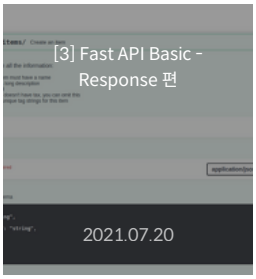
[6] Middleware, Background Tasks, Sub Application in Fast API (0) 202
[5] Dependency Injection이란? in Fast API (0) 202
[4] Pydantic Model (0) 202
[3] Fast API Basic - Response 편 (0) 202
[1] Fast API로 한걸음! (0) 202

Tag BODY, cookie, Fast API, fast api란, File, header, parameter, PATH, query, Request

'Python/Fast API' Related Articles

[5] Dependency Injection이란? in Fast API
2021.07.22

[4] Pydantic Model
2021.07.22



1 Comments

정의김 2022.10.14 13:54 신고

매우 잘읽었습니다! 감사합니다

Name

Password

여러분의 소중한 댓글을 입력해주세요

Secret

Sen