

빅데이터를 지탱하는 기술

Week 3

Network & FastAPI

23.08.02 / 엄소은

CONTENTS

01. Network

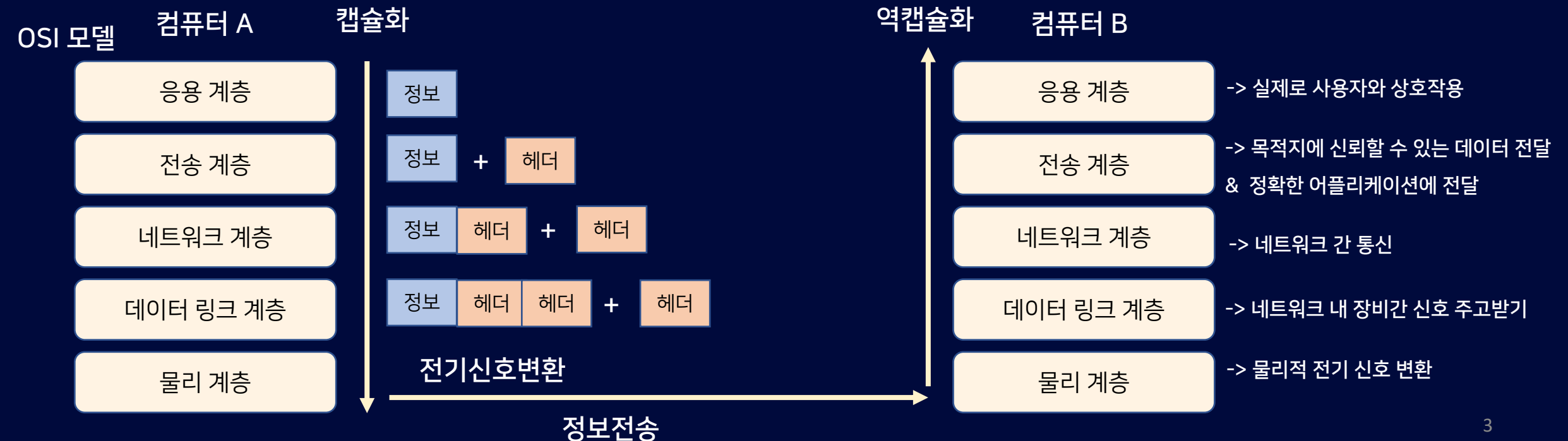
- 네트워크 기초
- Protocol

02. FastAPI

- Get, Post, Put, Delete
- Pydantic
- HTTP Status Code
- 실습

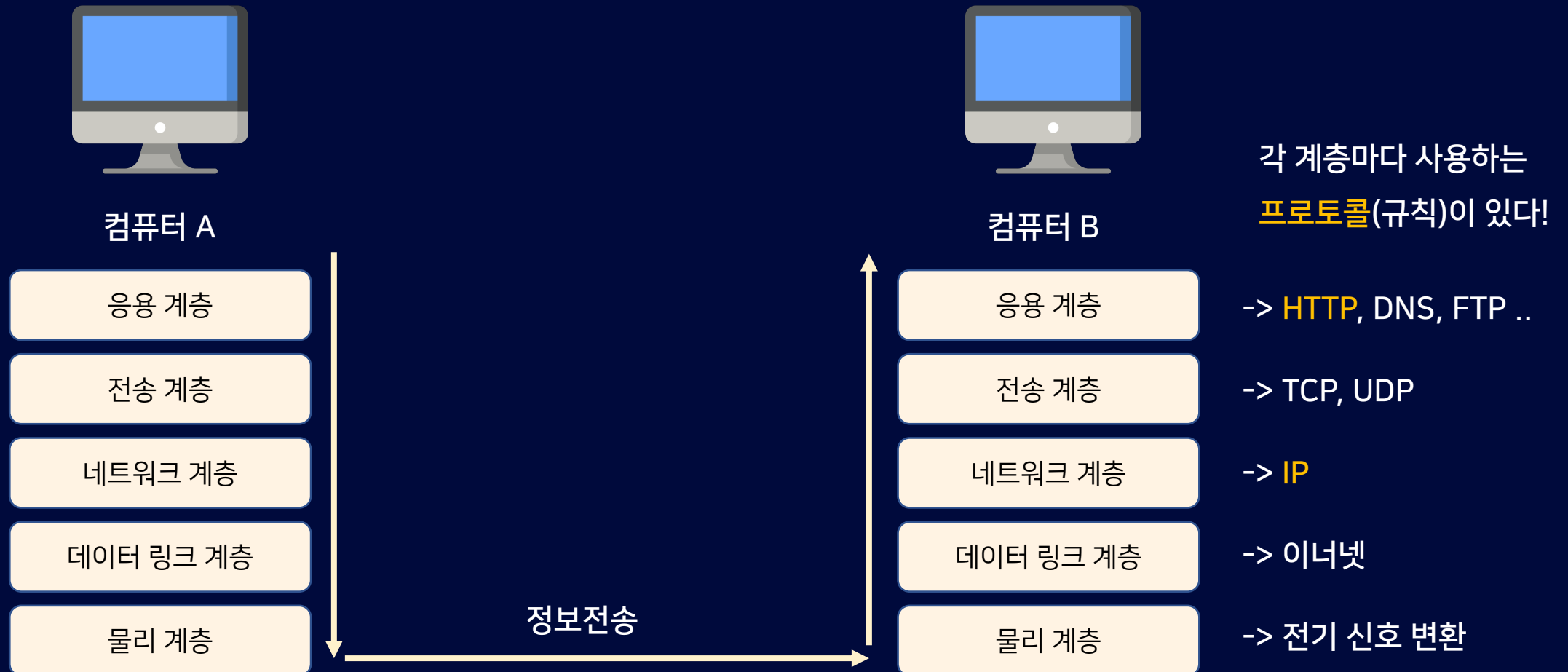
1. Network

네트워크 기초



1. Network

네트워크 기초



1. Network

IP & Port Number

- IP 는 네트워크 계층에서 사용되는 프로토콜
- 네트워크 간 통신을 도와준다 !
- 내 IP 주소 = 내 컴퓨터의 주소

My IP Address is:

IPv4: ? 121.166.94. 

- 포트 주소는 전송계층에서 사용됨
- 포트 주소는 내 컴퓨터 안에서 어떤 어플리케이션을 사용하고 싶은지를 구별해줌

어플리케이션	포트번호
SSH	22
HTTP	80
HTTPS	443
DNS	53

웹 서버 작동방식

- www: World Wide Web
- www 에는 HTML , URL, HTTP 3가지 기술이 작동한다
- 클라이언트는 웹 사이트를 보기 위해 웹 서버의 80번 포트를 이용하여 HTTP 통신을 한다



→
웹 서버 어플리케이션 : HTTP
HTTP Port Number : 80



1. Network

HTTP Protocol



```
waiting for application startup.  
Application startup complete.  
127.0.0.1:65145 - "GET / HTTP/1.1" 404 Not Found  
127.0.0.1:65145 - "GET /docs HTTP/1.1" 200 OK  
127.0.0.1:65145 - "GET /openapi.json HTTP/1.1" 200 OK  
HotFiles detected changes in local file. Reloading
```

HTTP Response status code

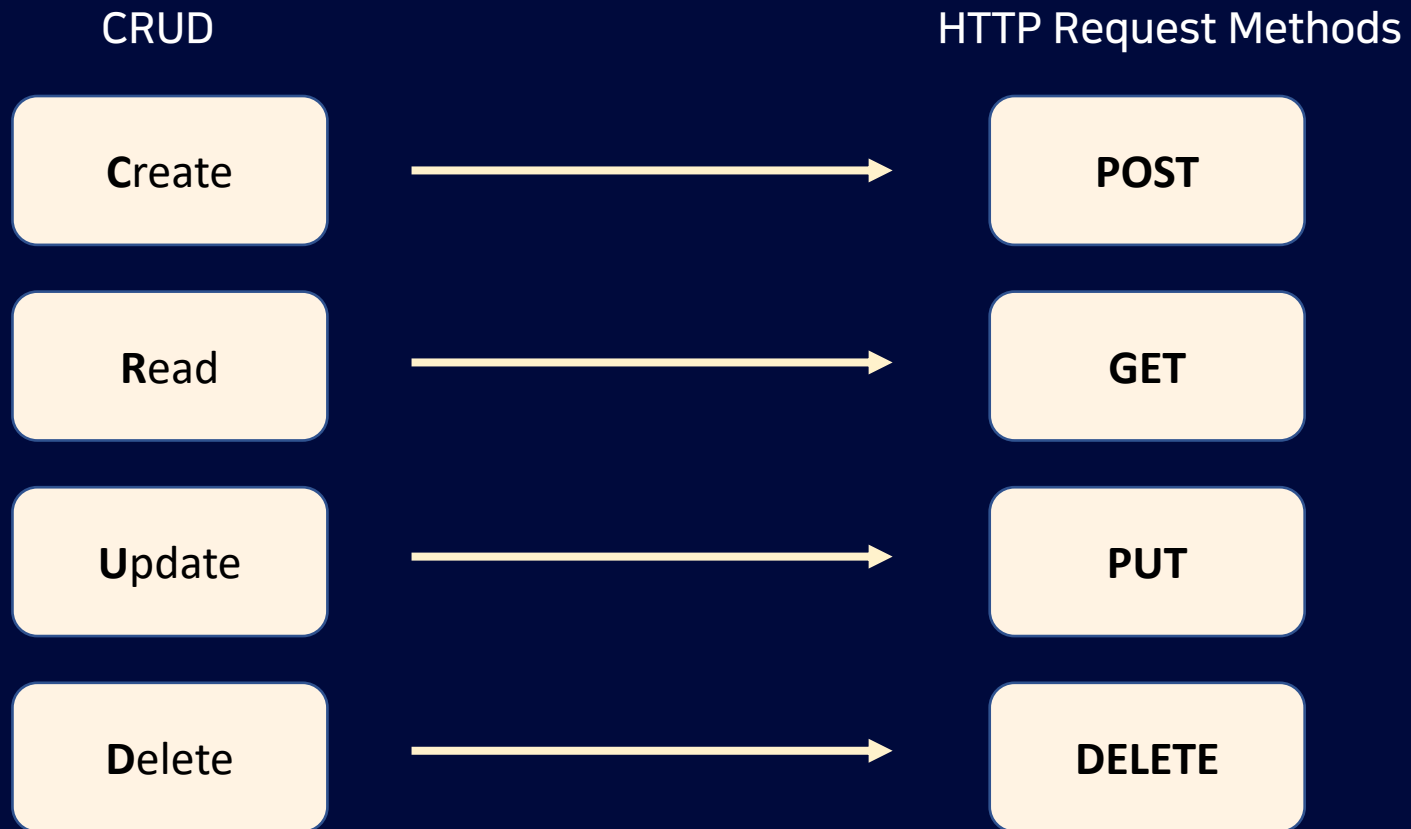
- Informational Response(요청을 받았으며 프로세스를 계속 진행) : 100 ~ 199
- Successful Response (요청을 성공적으로 받았으며 인식했고 수용) : 200 ~ 299
- Redirection Messages(요청 완료를 위해 추가 작업 조치가 필요) : 300 ~ 399
- Client error responses(요청의 문법이 잘못되었거나 요청을 처리할 수 없다) : 400 ~ 499
- Server error responses(서버의 오류로 인해 요청을 수행할 수 없다) : 500 ~ 599

자주 사용되는 상태 코드

- 200 OK : 요청이 성공적 !
- 400 Bad Request : 잘못된 문법으로 서버가 요청을 이해할 수 없다
- 401 Unauthorized : 클라이언트가 미인증 상태여서 요청을 수행할 수 없다
- 404 Not Found: 서버가 요청 받은 리소스를 찾을 수 없다

1. Network

CRUD 와 HTTP Method



2. FastAPI

Python Web Framework

- Django, FastAPI, Flask 등등 웹서비스의 백엔드 프레임워크는 다양하다
- 그 중에서 왜 **FastAPI** 를 사용하냐?
 - ASGI(비동기 처리 (async) 가능 = 병렬 처리 가능)
 - pydantic을 이용한 validation -> API 사용자, 개발자 간 소통을 정확하게
 - OpenAPI 문서 자동 생성 -> LocalHost/docs 로 이동하면 FastAPI 에서 만들어놓은 Swagger UI 문서로 테스트 가능
 - 뛰어난 문서 -> 따라하기 쉽고 상세한 문서가 있다



2. FastAPI

FastAPI 이용해서 API 설계하기 - GET

```
app = FastAPI(title = "API Practice",  
              version = "1.0")
```

```
@app.get('/')  
async def first_api():  
    return ("FastAPI 를 배워봅시다!")  
  
@app.get('/api-endpoint')  
async def first_api_endpoint():  
    return ("내 api-endpoint !")
```

URL : 127.0.0.1:8000/

URL : 127.0.0.1:8000/api-endpoint

"FastAPI 를 배워봅시다!"

"내 api-endpoint !"

```
~/study/FastAPI/sentiment-project ▶ uvicorn books1:app --reload
```

서버 시작하기: uvicorn books1:app --reload

↓
uvicorn: 웹서버 이름
books1: 현재 파이썬 파일 이름
app : FastAPI 이름

2. FastAPI

FastAPI 이용해서 API 설계하기 - GET

- Dynamic Parameter 사용하기

```
@app.get('/book/{dynamic_param}')  
async def test_dynamic_param(dynamic_param):  
    return {'dynamic_param': dynamic_param}
```

Ⓜ <http://127.0.0.1:8000/book/mybook> ✕

```
{"dynamic_param": "mybook"}
```

Ⓜ <http://127.0.0.1:8000/book/yourbook> ✕

```
{"dynamic_param": "yourbook"}
```

2. FastAPI

FastAPI 이용해서 API 설계하기 - GET

```
BOOKS = [  
    {'title': 'Mockingbird', 'author': 'Harper Lee', 'category': 'fiction'},  
    {'title': 'MobyDick', 'author': 'Herman Melville', 'category': 'fiction'},  
    {'title': '1984', 'author': 'Author Three', 'category': 'history'},  
    {'title': 'Great Gatsby', 'author': 'Fitzgerald', 'category': 'fiction'},  
    {'title': 'Frankenstien', 'author': 'Shelly', 'category': 'zombie'},  
    {'title': 'HarryPotter', 'author': 'J.K.Rolling', 'category': 'fiction'}  
]
```

```
@app.get('/books/{book_title}')  
async def read_book(book_title: str):  
    for book in BOOKS:  
        if book.get('title').casefold() == book_title.casefold():  
            return book  
    else:  
        return {'new_book': book_title}
```

Ⓜ http://127.0.0.1:8000/books/Mockingbird ✕

```
{"title": "Mockingbird", "author": "Harper Lee", "category": "fiction"}
```

Ⓜ http://127.0.0.1:8000/books/Lord%20of%20rin ✕

Ⓜ 127.0.0.1:8000/books/Lord of rings

```
{"new_book": "Lord of rings"}
```

2. FastAPI

FastAPI 이용해서 API 설계하기 - GET

- Query Parameters

{name=value} pair 를 가진 파라미터

URL 에서 ? 뒤에 붙여진다

```
@app.get("/books/")
async def read_category_by_query(category:str):
    books_to_return = []
    for book in BOOKS:
        if book.get('category').casefold() == category.casefold():
            books_to_return.append(book)
    return books_to_return
```

 <http://127.0.0.1:8000/books/?category=zombie> 

내가 가진 books 들 중에서 category 가 zombie 인 것들 찾아줘!

```
[{"title": "Frankenstien", "author": "Shelly", "category": "zombie"}]
```

2. FastAPI

FastAPI 이용해서 API 설계하기 - POST

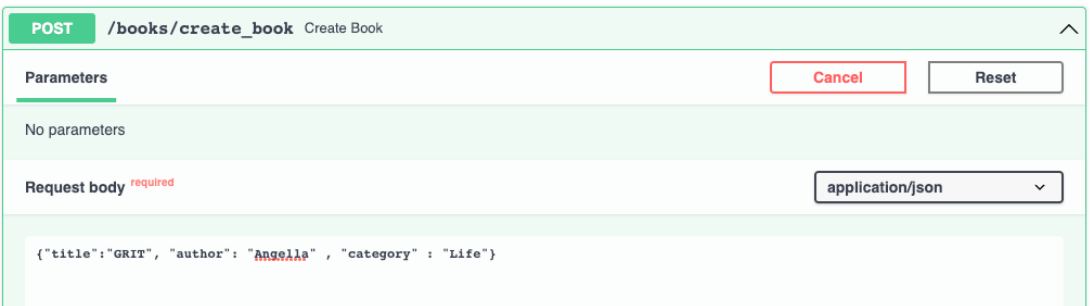
- 새로운 정보를 생성하는 POST Method

```
@app.post("/books/create_book")
async def create_book(new_book = Body()):
    BOOKS.append(new_book)
    return BOOKS
```

POST 는 **Request Body** 가 있다
(정보 전송용)

<-> GET은 Request Body 를 가질 수 없다 !

<https://127.0.0.1:8000/docs>



Code	Details
200	<p>Response body</p> <pre>[{"title": "1984", "author": "Author Three", "category": "history"}, {"title": "Great Gatsby", "author": "Fitzgerald", "category": "fiction"}, {"title": "Frankenstein", "author": "Shelly", "category": "zombie"}, {"title": "Harry Potter", "author": "J.K.Rolling", "category": "fiction"}, {"title": "GRIT", "author": "Angella", "category": "Life"}]</pre>

2. FastAPI

FastAPI 이용해서 API 설계하기 - PUT

- 새로운 정보를 업데이트하는데 사용한다

```
@app.put("/books/update_book")
async def update_book(updated_book = Body()):
    for i in range(len(BOOKS)):
        if BOOKS[i].get('title').casefold() == updated_book.get('title').casefold():
            BOOKS[i] = updated_book
    return BOOKS
```



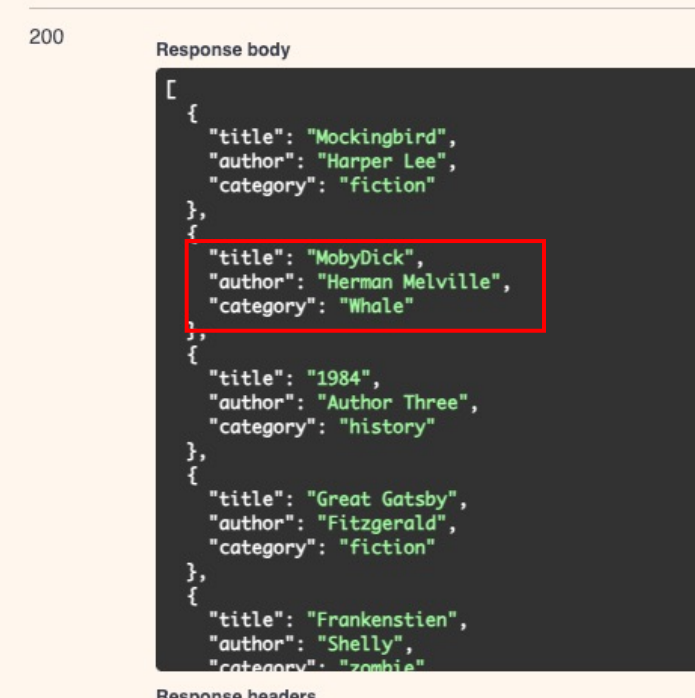
PUT /books/update_book Update Book

Parameters Cancel Reset

No parameters

Request body required application/json

{ "title": "MobyDick", "author": "Herman Melville", "category": "Whale" }



200

Response body

```
[
  {
    "title": "Mockingbird",
    "author": "Harper Lee",
    "category": "fiction"
  },
  {
    "title": "MobyDick",
    "author": "Herman Melville",
    "category": "Whale"
  },
  {
    "title": "1984",
    "author": "Author Three",
    "category": "history"
  },
  {
    "title": "Great Gatsby",
    "author": "Fitzgerald",
    "category": "fiction"
  },
  {
    "title": "Frankenstien",
    "author": "Shelly",
    "category": "zombie"
  }
]
```

Response headers

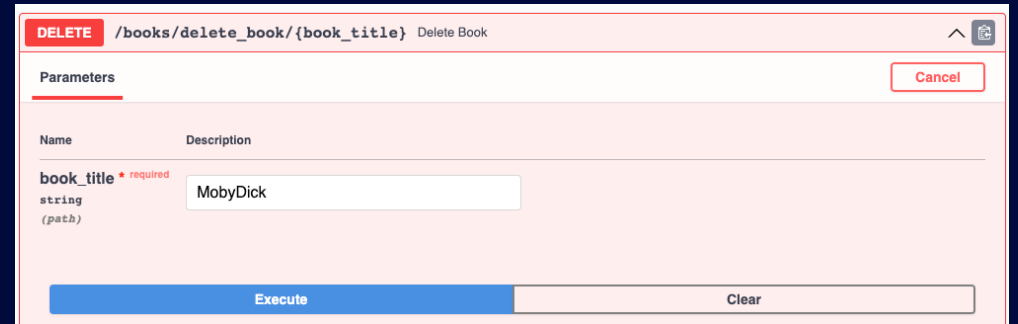
-> Body에서 정보 줄때
" " 사용해야 함 !!
' ' 사용하면 오류 남

2. FastAPI

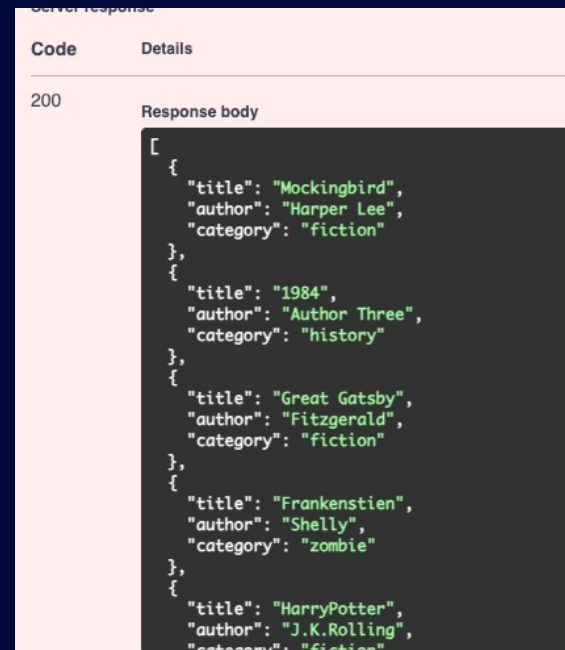
FastAPI 이용해서 API 설계하기 - DELETE

```
@app.delete("/books/delete_book/{book_title}")
async def delete_book(book_title:str):
    for i in range(len(BOOKS)):
        if BOOKS[i].get('title').casefold() == book_title.casefold():
            BOOKS.pop(i)
            break
    return BOOKS
```

book_title 을 parameter 로 받아서
title과 일치하는 책이 있으면 삭제해라 !



The image shows the Swagger UI for a DELETE endpoint. The title bar is red and says "DELETE /books/delete_book/{book_title} Delete Book". Below the title bar, there's a "Parameters" section. It lists a parameter "book_title" with a red asterisk indicating it's required. The parameter type is "string" and the location is "(path)". There's a text input field containing "MobyDick". At the bottom, there are two buttons: "Execute" (blue) and "Clear" (white).



The image shows the "Server response" section of the Swagger UI. It has two tabs: "Code" and "Details". The "Code" tab is selected, showing a status code of "200". The "Details" tab shows the "Response body" as a JSON array of book objects. The JSON is displayed in a dark-themed code editor.

```
[
  {
    "title": "Mockingbird",
    "author": "Harper Lee",
    "category": "fiction"
  },
  {
    "title": "1984",
    "author": "Author Three",
    "category": "history"
  },
  {
    "title": "Great Gatsby",
    "author": "Fitzgerald",
    "category": "fiction"
  },
  {
    "title": "Frankenstein",
    "author": "Shelly",
    "category": "zombie"
  },
  {
    "title": "Harry Potter",
    "author": "J.K.Rolling",
    "category": "fiction"
  }
]
```

2. FastAPI

Pydantics

- POST 에서 정보를 줄 때 validation 을 하고 싶다면? -> Pydantics 이용 !
- 데이터 모델링, 데이터 parsing 을 할 때 에러를 핸들링하는 라이브러리

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

```
class BookRequest(BaseModel):
    id: Optional[int] = Field(title='id is not needed')
    title: str = Field(min_length=3)
    author: str = Field(min_length=1)
    description: str = Field(min_length=1, max_length=100)
    rating: int = Field(gt=0, lt=6)
```

```
@app.post("/create_book")
async def create_book(book_request : BookRequest):
    new_book = Book(**book_request.dict())
    BOOKS.append(new_book)
    return BOOKS
```

-> **: book_request 로 부터 받은 key,value 를
Book() 객체에 넘겨준다
new_book 은 새로운 BOOK 클래스의 인스턴스가 된다

```
print(type(new_book))
```

```
<class 'books2.Book'>
```

2. FastAPI

Pydantics

Request body **required**

```
{
  "id": 7,
  "title": "GRIT",
  "author": "Angella",
  "description": "GRIT is my fav book",
  "rating": 10
}
```

Code

Details

422

Error: Unprocessable Entity

Response body

```
{
  "detail": [
    {
      "loc": [
        "body",
        "rating"
      ],
      "msg": "ensure this value is less than 6",
      "type": "value_error.number.not_lt",
      "ctx": {
        "limit_value": 6
      }
    }
  ]
}
```



Download

2. FastAPI

Pydantics

Schemas	
BookRequest ∨ {	
id	integer <i>title: id is not needed</i>
title*	string <i>title: Title</i> <i>minLength: 3</i>
author*	string <i>title: Author</i> <i>minLength: 1</i>
description*	string <i>title: Description</i> <i>maxLength: 100</i> <i>minLength: 1</i>
rating*	integer <i>title: Rating</i> <i>exclusiveMaximum: 6</i> <i>exclusiveMinimum: 0</i>
}	

내가 만든 BookRequest Schema 를 볼 수 있다

2. FastAPI

Pydantics

```
class BookRequest(BaseModel):
    id: Optional[int] = Field(title='id is not needed')
    title: str = Field(min_length=3)
    author: str = Field(min_length=1)
    description: str = Field(min_length=1, max_length=100)
    rating: int = Field(gt=0, lt=6)

class Config:
    schema_extra = {
        'example': {
            'title': 'A new book',
            'author': 'Put in Author',
            'description': 'New description of book',
            'rating': 'Put in rating from 0~6',
        }
    }
```

Request body **required**

Example Value | Schema

```
{
  "title": "A new book",
  "author": "Put in Author",
  "description": "New description of book",
  "rating": "Put in rating from 0~6"
}
```

2. FastAPI

Pydantics - Path, Query

```
#Path 로 validate 하는 방법 : book_id 가 0 보다 크도록 함
@app.get("/books/{book_id}")
async def read_book(book_id : int = Path(gt=0)):
    for book in BOOKS:
        if book.id == book_id:
            return book
```

-> {book_id} 를 path 로 받으니까 Path 로 validate

```
@app.get("/books/")
async def read_book_by_rating(book_rating:float = Query(gt=0,lt=6)):
    books_to_return = []
    for book in BOOKS:
        if book.rating == book_rating:
            books_to_return.append(book)
    return books_to_return
```

-> book_rating 은 Query (질문 던지는 정보) 니까
Query 로 validate

둘의 차이점 잘 생각해보기 !!

2. FastAPI

HTTP Response status code

2xx Successful Request

- 200 OK : 요청이 성공적 !
- 201 Created : Post 로 새로운 정보 생성 성공
- 204 No Content : PUT 로 업데이트는 했지만 리턴값 이 없을 때

4xx Client Error Statue code

- 400 Bad Request : 잘못된 문법으로 서버가 요청을 이해할 수 없다
- 401 Unauthorized : 클라이언트가 미인증 상태여서 요청을 수행할 수 없다
- 404 Not Found: 서버가 요청 받은 리소스를 찾을 수 없다
- 422 Unprocessable Entity : 내가 보낸 정보에 semantic error 가 있다

5xx Inter Server Error

- 500 Inter Server Error: 서버에서 문제가 발생했다

2. FastAPI

HTTP Exceptions

```
#Path 로 validate 하는 방법 : book_id 가 0 보다 크도록 함
#찾고자 하는 id 가 없을 때 404 에러
@app.get("/books/{book_id}")
async def read_book(book_id : int =Path(gt=0)):
    for book in BOOKS:
        if book.id == book_id:
            return book
    raise HTTPException(status_code=404, detail = "Item not found")
```

- 에러 문구 = 사용자에게 일종의 가이드
- 왜 오류가 났는지를 알려주는 것 !
- 적절한 상황에서 맞는 HTTP Status code 를 쓸 것

Code

Details

404

Undocumented Error: Not Found

Response body

```
{
  "detail": "Item not found"
}
```

Download

Response headers

```
content-length: 27
content-type: application/json
date: Wed, 02 Aug 2023 04:24:12 GMT
server: uvicorn
```

Responses

실습

- 그럼 실제 모델 Deploy 하는 거는 ???
- 알아야 할 기초는 다 배웠다 !
- 실습하면서 해보자

< 해볼 것 >

1. HuggingFace 모델 혹은 자신의 모델 가져와서 로컬에 놓기
2. main.py 파일에 모델 실행시키기
3. api.py 파일에 pydantic, HTTPException error 사용해서 api 설계하기
(input값은 어떤 것이고, output는 어떻게 나와야 하며, 에러는 어떤 상황에 띄울지 고민하며 설계하기)
(모르는 사람이 api 문서를 봐도 어떤 값을 넣어야 겠구나 ! 라는 친절하고 상세한 가이드 만들기
← 현업에서 굉장히 중요 ! 지금부터 습관화 하자)

2. FastAPI

다음주에는 ?

- 데이터를 데이터베이스와 연동하여 저장하는 방법
- AWS EC2 를 이용하여 배포하기
- Authentication & Authorization 추가하기

3. Reference

- 모두의 네트워크, 미즈구치 카츠야
- Udemy, FastAPI - The complete course 2023