

PyWeb Symposium 2024

웹 프레임워크 (Web Framework)

FastAPI와 함께 하는 문서 우선 개발 (Documentation First Development, DFD)

발표자: 이태현

발표 자료



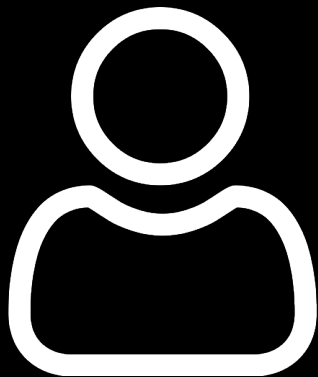
- 당근 인터널 프로덕트팀 백엔드 엔지니어

- PyCon Korea 2023 발표

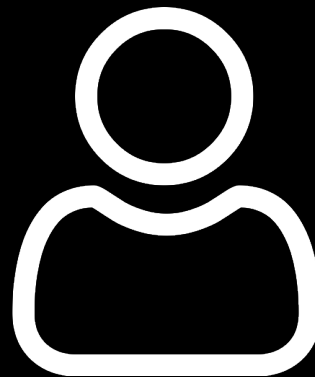
: 파이썬을 처음 사용하는 동료와 효율적으로 일하는 방법

<https://url.kr/rzkas6>

소통

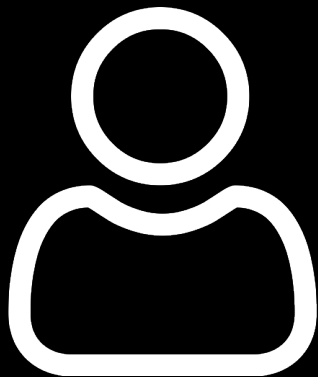


클라이언트 개발자



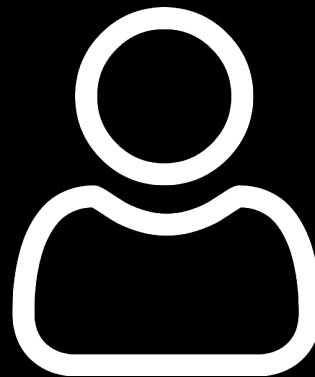
서버 개발자

소통



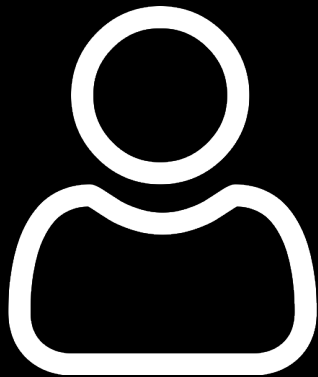
클라이언트 개발자

어떻게 요청해야 하나요?



서버 개발자

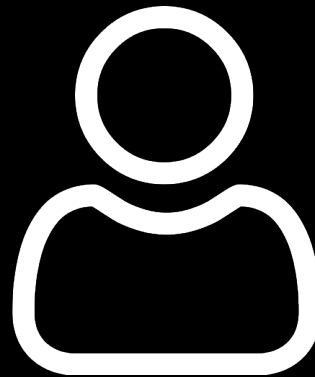
소통



클라이언트 개발자

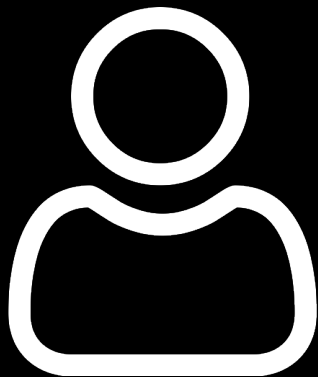


이렇게 보내주세요!



서버 개발자

소통

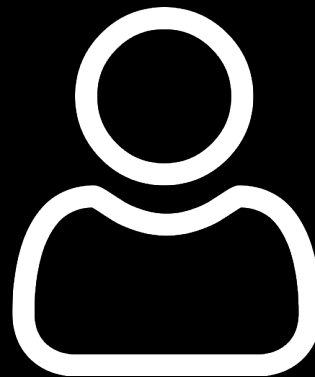


클라이언트 개발자

어떻게 요청해야 하나요?

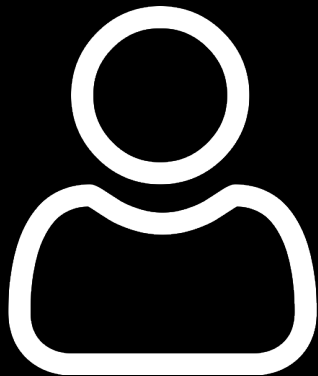


이렇게 보내주세요!



서버 개발자

소통



클라이언트 개발자

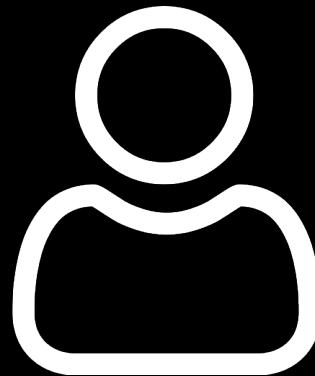
어떻게 요청해야 하나요?



조금 더 효율적으로 알려줄
수는 없을까?

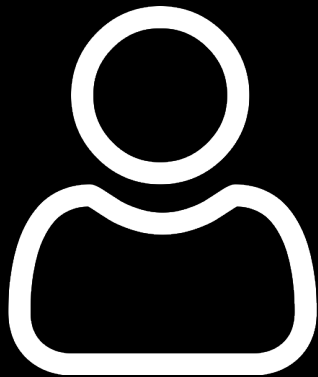


이렇게 보내주세요!



서버 개발자

소통



클라이언트 개발자

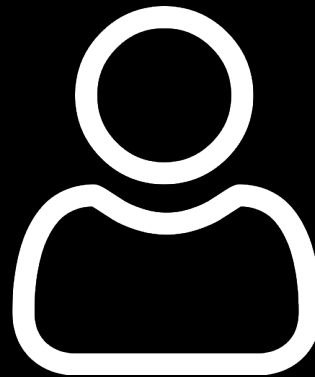
어떻게 요청해야 하나요?



기술로 해결할 수는 없을까?



이렇게 보내주세요!



서버 개발자

발표에서 다루는 내용

1. 문서의 중요성

■ 발표에서 다루는 내용

1. 문서의 중요성

2. 기존 문서 생성 방식과 단점

■ 발표에서 다루는 내용

1. 문서의 중요성

2. 기존 문서 생성 방식과 단점

3. FastAPI 웹 프레임워크를 통한 코드 기반 문서

■ 발표에서 다루는 내용

1. 문서의 중요성
2. 기존 문서 생성 방식과 단점
3. FastAPI 웹 프레임워크를 통한 코드 기반 문서
4. 효율적이고 안전한 문서 관리

■ 발표에서 다루지 않는 내용

1. Open API 명세

■ 발표에서 다루지 않는 내용

1. Open API 명세

2. FastAPI 등 소개되는 각 패키지의 내부 작동 방식

■ 발표에서 다루지 않는 내용

1. Open API 명세

2. FastAPI 등 소개되는 각 패키지의 내부 작동 방식

3. Python의 타입

■ 발표에서 다루지 않는 내용

1. Open API 명세
2. FastAPI 등 소개되는 각 패키지의 내부 작동 방식
3. Python의 타입
4. 만능 해결책 (No Silver Bullet)

■ 발표에서 다루지 않는 내용

1. Open API 명세
2. FastAPI 등 소개되는 각 패키지의 내부 작동 방식
3. Python의 타입
4. 만능 해결책 (No Silver Bullet)

■ 문서의 중요성

문서는 왜 중요할까?

■ 문서의 중요성: 협업의 관점

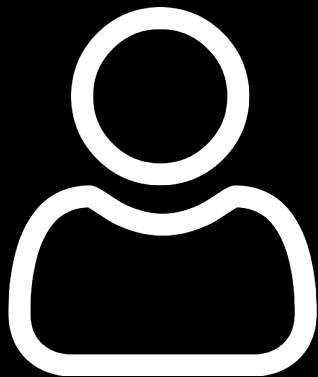
협업의 관점으로 본 중요성

■ 문서의 중요성: 협업의 관점

거시적 협업

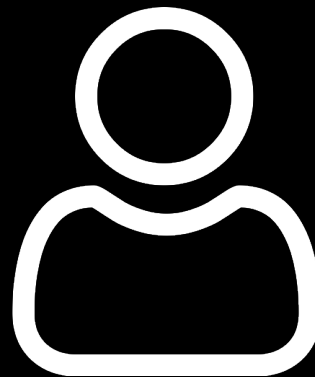
미시적 협업

문서의 중요성: 거시적 협업



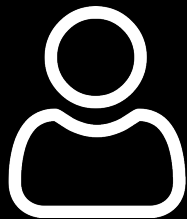
클라이언트 개발자

소통

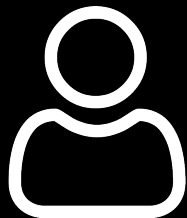


서버 개발자

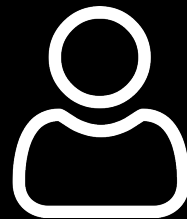
문서의 중요성: 거시적 협업



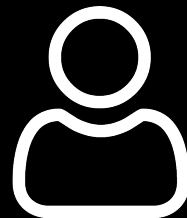
⋮



클라이언트 개발자

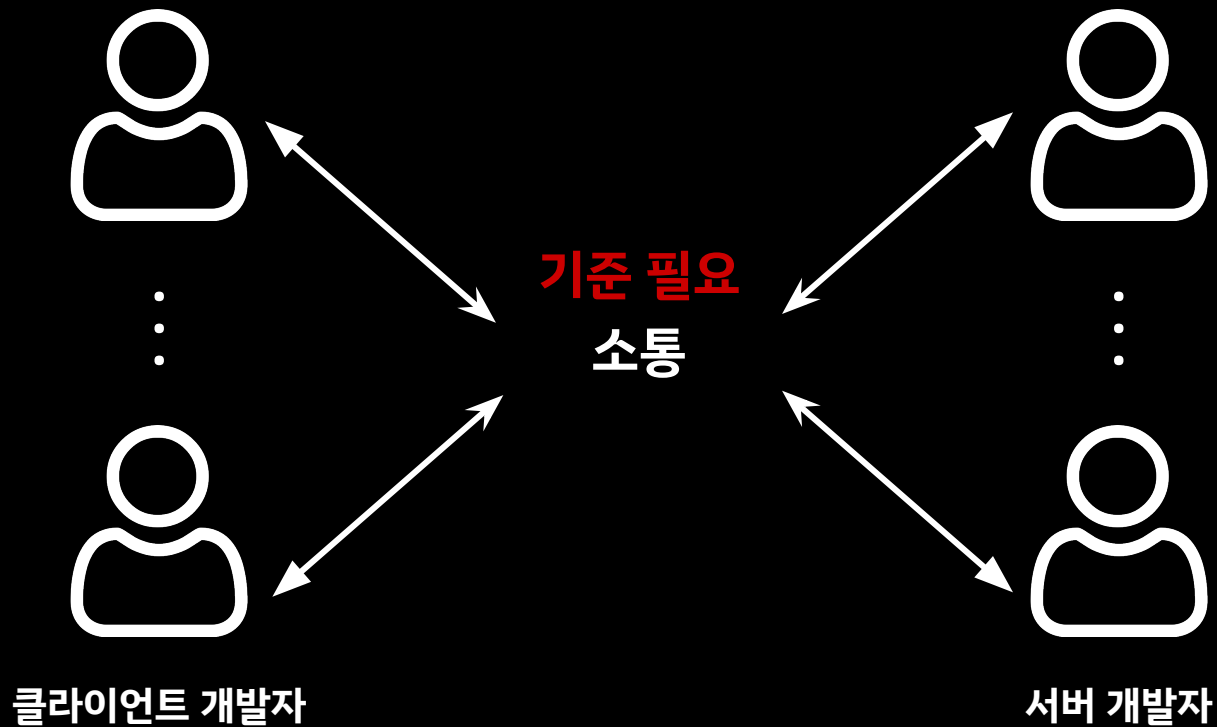


⋮

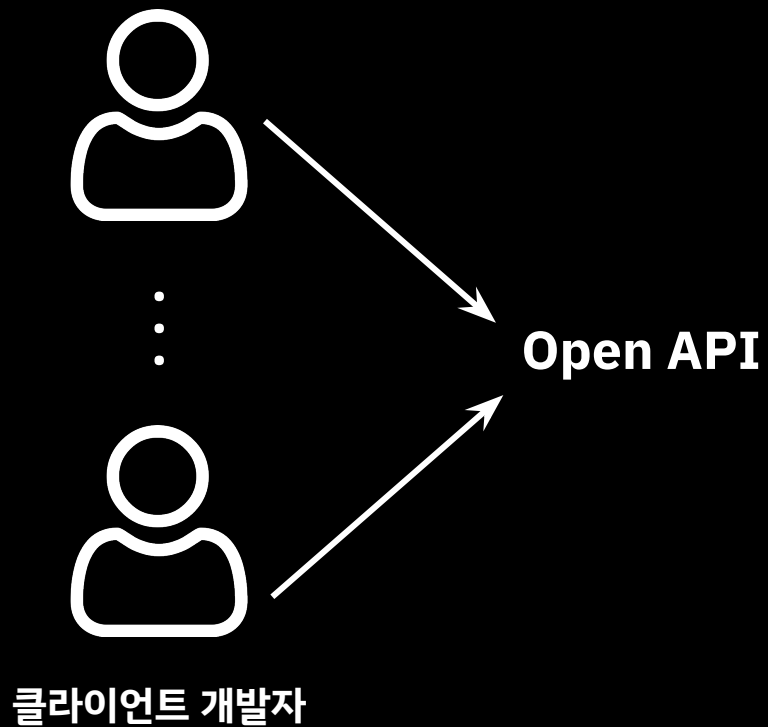


서버 개발자

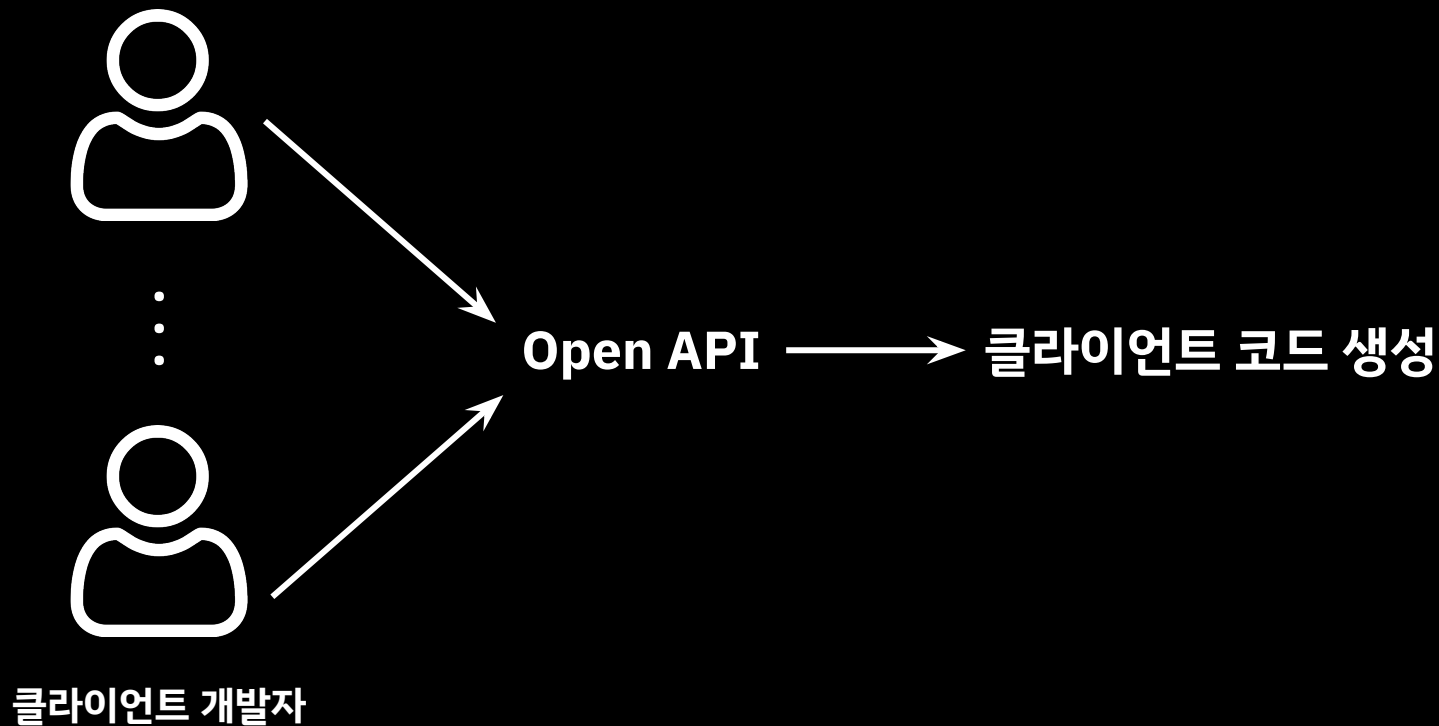
문서의 중요성: 거시적 협업



문서의 중요성: 거시적 협업



문서의 중요성: 거시적 협업



문서의 중요성: 거시적 협업

components:

schemas:

Book:

properties:

name:

type: string

title: Name

description: Name of book.

type: object

required:

- name

문서의 중요성: 거시적 협업

components:

schemas:

Book:

properties:

name:

type: string

title: Name

description: Name of book.

type: object

required:

- name



```
/**
```

```
*
```

```
* @export
```

```
* @interface Book
```

```
*/
```

```
export interface Book {
```

```
/**
```

```
* Name of book.
```

```
* @type {string}
```

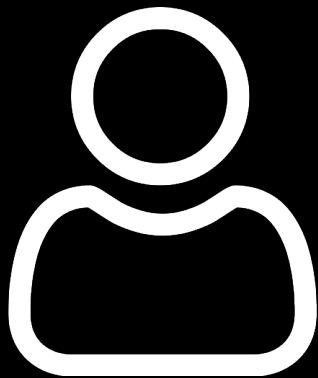
```
* @memberof Book
```

```
*/
```

```
name: string;
```

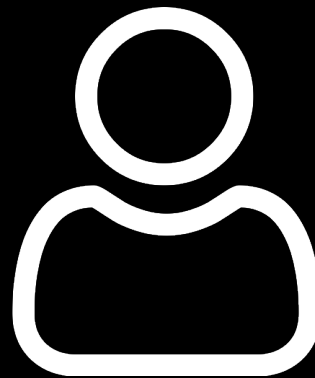
```
}
```

문서의 중요성: 미시적 협업



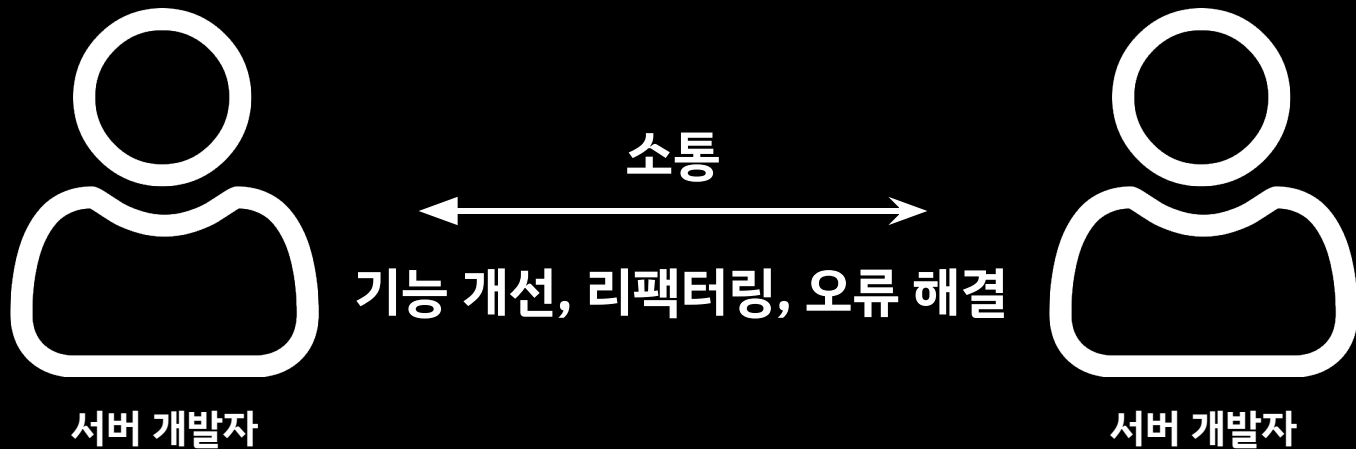
서버 개발자

소통

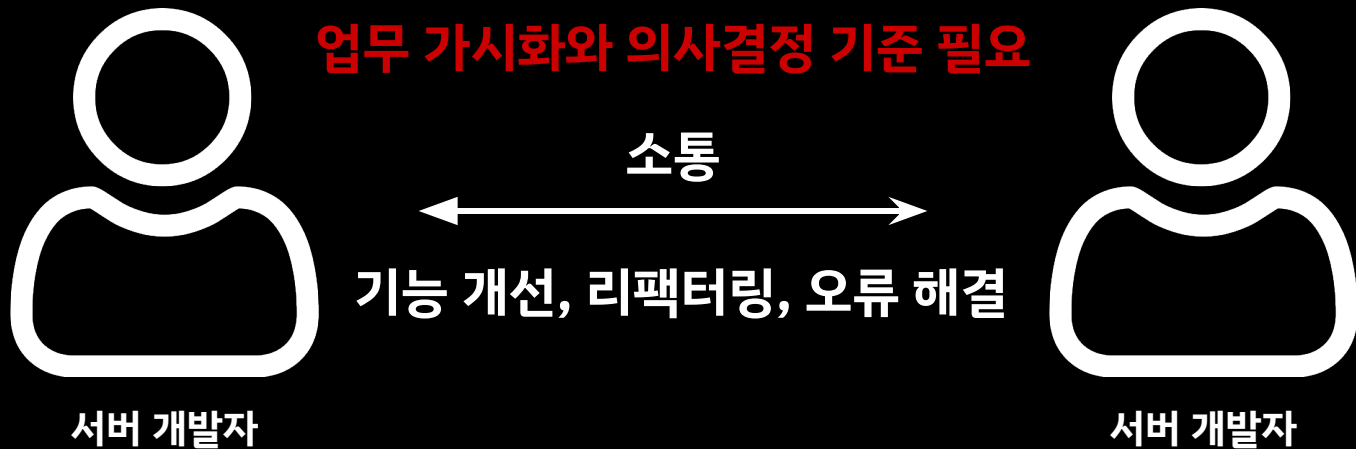


서버 개발자

문서의 중요성: 미시적 협업



문서의 중요성: 미시적 협업



■ 문서의 중요성

잘 작성된 문서

■ 문서의 중요성

잘 작성된 문서 → 성장하는 소프트웨어

■ 전통적인 문서화 방식

이전에는 어떻게 하고 있었을까?

■ 전통적인 문서화 방식: 첫 번째, 주석 활용

```
@router.route(rule="/books", methods=["GET"])
```

```
def get_books():
```

```
    """
```

```
    ---
```

```
    definitions:
```

```
        Book:
```

```
            type: object
```

```
            properties:
```

```
                name:
```

```
                    type: str
```

```
    """
```

```
    return
```

GET

/books Get Books

get_books

Description

1. Get all books.

To-do

1. Search by name with query parameter.

Parameters

Try it out

No parameters

Responses

Response content type

application/json



Code

Description

200

Successful Response

Example Value | Model

```
[
  {
    "name": "Demian"
  }
]
```

Models



```
Book {
  name str
}
```

GET

/books

Get Books

get_books

Description

1. Get all books.

To-do

1. Search by name with query parameter.

Parameters

Try it out

No parameters

Responses

Response content type

application/json



Code

Description

200

Successful Response

Example Value | Model

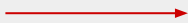
```
[
  {
    "name": "Demian"
  }
]
```

Models



```
Book {
  name
}
```

str

**definitions** 부분에 정의되었던 모델

■ 전통적인 문서화 방식: 첫 번째, 주석 활용

코드와 주석 동기화 문제 존재

■ 전통적인 문서화 방식: 첫 번째, 주석 활용

```
@router.route(rule="/books", methods=["GET"])
```

```
def get_books():
```

```
    """
```

```
    ---
```

```
    definitions:
```

```
        Book:
```

```
            type: object
```

```
            properties:
```

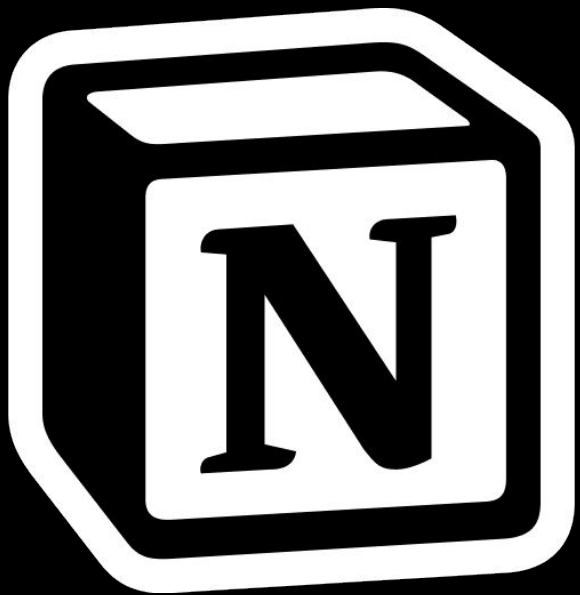
```
                name:
```

```
                    type: str
```

```
    """
```

```
    return
```

■ 전통적인 문서화 방식: 두 번째, 문서 작성 도구 활용



Request

Body Parameter Example

- firstName , lastName , age , email , birth

```
{
  "firstName": "Taehyun",
  "lastName": "Lee",
  "age": 19,
  "email": "0417taehyun@gmail.com",
  "birth": "1997-04-17"
}
```

Response

Success Example

- Status Code 201

```
{
  "data": {
    "firstName": "Taehyun",
    "lastName": "Lee",
    "age": 19,
    "email": "0417taehyun@gmail.com",
    "birth": "1997-04-17",
    "id": "A0001",
    "name": "Taehyun Lee"
  }
}
```


■ 전통적인 문서화 방식: 두 번째, 문서 작성 도구 활용

코드와 문서 동기화 문제 존재

Request

Body Parameter Example

- firstName , lastName , age , email , birth

```
{
  "firstName": "Taehyun",
  "lastName": "Lee",
  "age": 19,
  "email": "0417taehyun@gmail.com",
  "birth": "1997-04-17"
}
```

Response

Success Example

- Status Code 201

```
{
  "data": {
    "firstName": "Taehyun",
    "lastName": "Lee",
    "age": 19,
    "email": "0417taehyun@gmail.com",
    "birth": "1997-04-17",
    "id": "A0001",
    "name": "Taehyun Lee"
  }
}
```

■ 전통적인 문서화 방식: 세 번째, API 클라이언트 플랫폼 활용





GET



http://localhost:8000/books

Try ↗

Params

Headers

Body

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body

Headers (4)

Status Code

200 OK

Pretty

Raw

Preview

JSON



```
1  [
2    {
3      "name": "Demian"
4    },
5    {
6      "name": "Siddhartha"
7    }
8  ]
```



GET



http://localhost:8000/books

Try ↗

Params

Headers

Body

Query Params

	Key	Value	Description	⋮ Bulk Edit
	Key	Value	Description	

Body

Headers (4)

Status Code

200 OK

Pretty

Raw

Preview

JSON



```
1  [
2    {
3      "name": "Demian"
4    },
5    {
6      "name": "Siddhartha"
7    }
8  ]
```

A Simple API

A simple API to describe how to use Postman as a documentation generator.

Books

GET Get Books

```
http://localhost:8000/books
```

Description

1. Get all books.

To-do

1. Serach by name with query parameter.

Example Request

Example ▾

curl



```
curl --location 'http://localhost:8000/books'
```

Example Response

Body

Headers (4)

200 OK

json



```
[
  {
    "name": "Demian"
  },
  {
    "name": "Siddhartha"
  }
]
```

■ 전통적인 문서화 방식: 세 번째, API 클라이언트 플랫폼 활용

코드와 플랫폼 동기화 문제 존재

A Simple API

A simple API to describe how to use Postman as a documentation generator.

Books

GET Get Books

```
http://localhost:8000/books
```

Description

1. Get all books.

To-do

1. Serach by name with query parameter.

Example Request

Example ▾

curl

```
curl --location 'http://localhost:8000/books'
```

Example Response

Body

Headers (4)

200 OK

json

```
[
  {
    "name": "Demian"
  },
  {
    "name": "Siddhartha"
  }
]
```


■ 전통적인 문서화 방식: 네 번째, 메타프로그래밍 활용

반대로 문서를 통해 코드를 만들 수 없을까?

■ 전통적인 문서화 방식: 네 번째, 메타프로그래밍 활용

반대로 문서를 통해 코드를 만들 수 없을까?

■ 전통적인 문서화 방식: 네 번째, 메타프로그래밍 활용

components:

schemas:

Book:

properties:

name:

type: string

title: Name

description: Name of book.

type: object

required:

- name

■ 전통적인 문서화 방식: 네 번째, 메타프로그래밍 활용

```
components:
  schemas:
    Book:
      properties:
        name:
          type: string
          title: Name
          description: Name of book.
      type: object
      required:
        - name
```



```
// Book defines model for Book.
type Book struct {
    // Name Name of book.
    Name string `json:"name"`
}

// ServerInterface represents all server handlers.
type ServerInterface interface {
    // Get Books
    // (GET /books)
    GetBooksBooksGet(c *gin.Context)
}
```

■ 전통적인 문서화 방식: 네 번째, 메타프로그래밍 활용

한정적인 자유도 존재

■ 전통적인 문서화 방식: 정리

완벽하게 코드와 일치할 수 없는 문제

■ 전통적인 문서화 방식: 정리

완벽하게 코드와 일치할 수 없는 문제

FastAPI는 어떻게 문서를 만들까?

FastAPI와 Pydantic



Starlette 

Pydantic의 객체

```
from typing import Annotated
```

```
from pydantic import BaseModel, Field
```

```
class Book(BaseModel):  
    name: Annotated[  
        str,  
        Field(default=..., description="Name of book.", examples=["Demian", "Siddhartha"])  
    ]  
]
```

Pydantic의 객체

```
from typing import Annotated
```

```
from pydantic import BaseModel, Field
```

```
class Book(BaseModel):  
    name: Annotated[  
        str,  
        Field(default=..., description="Name of book.", examples=["Demian", "Siddhartha"]  
    )  
]
```

—————▶ **DTO(Data Transfer Object) 역할**

Pydantic의 객체

```
from typing import Annotated
```

```
from pydantic import BaseModel, Field
```

```
class Book(BaseModel):
```

```
    name: Annotated[
```

```
        str,
```

```
        Field(default=..., description="Name of book.", examples=["Demian", "Siddhartha"]
```

```
    )  모델이 가져야 할 필드를 타입과 Field 객체 통해 정의
```

```
]
```

FastAPI의 객체

```
@router.get(path="/books", response_model=Sequence[Book], status_code=status.HTTP_200_OK )
```

```
async def get_books():
```

```
    """
```

```
    Description \n
```

```
        1. Get all books.
```

```
    To-do \n
```

```
        1. Search by name with query parameter.
```

```
    """
```

```
    return [
```

```
        Book(name="Demian"),
```

```
        Book(name="Siddhartha")
```

```
    ]
```

FastAPI의 객체

```
@router.get(path="/books", response_model=Sequence[Book], status_code=status.HTTP_200_OK)
```

```
async def get_books():
```

 **응답 모델에 Pydantic 통해 생성한 객체 할당**

```
"""
```

```
Description \n
```

```
1. Get all books.
```

```
To-do \n
```

```
1. Search by name with query parameter.
```

```
"""
```

```
return [
```

```
    Book(name="Demian"),
```

```
    Book(name="Siddhartha")
```

```
]
```

FastAPI의 객체

```
@router.get(path="/books", response_model=Sequence[Book], status_code=status.HTTP_200_OK)
```

```
async def get_books():
```

```
    """
```

```
    Description \n
```

```
    1. Get all books.
```



주석은 곧 API 문서의 설명

```
    To-do \n
```

```
    1. Search by name with query parameter.
```

```
    """
```

```
    return [
```

```
        Book(name="Demian"),
```

```
        Book(name="Siddhartha")
```

```
    ]
```

GET

/books Get Books



Description

1. Get all books.

To-do

1. Serach by name with query parameter.

Parameters

[Try it out](#)

No parameters

Responses

Code

Description

Links

200

Successful Response

No links

Media type

application/json

Controls [Accept header](#).Example Value | [Schema](#)

```
[
  {
    "name": "Demian"
  }
]
```

Schemas



Book >

Expand all

object

GET

/booksGet Books

Description

1. Get all books.

To-do

1. Serach by name with query parameter.

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

[
 {
 "name": "Demian"
 }
]

Schemas

Book > Expand all object

—————→ Pydantic의 BaseModel 클래스 상속 받았던 객체

GET

/books

Get Books

⌵

Description

1. Get all books.

To-do

1. Serach by name with query parameter.

Parameters

Try it out

No parameters

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

—————→ Pydantic BaseModel 객체 상속 및 Field 객체 정의를 통해 생성된 모델과 예시

```
[
  {
    "name": "Demian"
  }
]
```

Schemas

⌵

Book >

Expand all

object

GET

/books Get Books



Description

1. Get all books.

→ 주석으로 작성했던 API에 대한 설명

To-do

1. Serach by name with query parameter.

Parameters

Try it out

No parameters

Responses

Code

Description

Links

200

Successful Response

No links

Media type

application/json



Controls Accept header.

Example Value | Schema

```
[
  {
    "name": "Demian"
  }
]
```

Schemas



Book >

Expand all

object

■ 좋은 문서의 조건

어떻게 하면 더 좋은 문서를 쓸 수 있을까?

■ 좋은 문서의 조건: 명시성

명시성

■ 좋은 문서의 조건: 명시성

명시성 = 직관성

■ 좋은 문서의 조건: 통일성

통일성

■ 좋은 문서의 조건: 통일성

통일성 = 예측 가능성

■ 좋은 문서의 조건: 안전성

안전성

■ 좋은 문서의 조건: 안전성

안전성 = 문서 보안

■ 좋은 문서를 작성하기 위한 방법: 명시적 객체

```
from typing import Annotated
```

```
from pydantic import BaseModel, Field
```

```
class Book(BaseModel):
```

```
    name: Annotated[
```

```
        str,
```

```
        Field(default=..., description="Name of book.", examples=["Demian", "Siddhartha"]
```

```
    )  명시적인 설명과 예시 작성
```

```
]
```

■ 좋은 문서를 작성하기 위한 방법: 명시적 객체

```
class BookEntity(BaseModel):  
    country: Annotated[  
        Country,  
        Field(default=..., description="A country of the book.")  
    ]  
    language: Annotated[  
        Language,  
        Field(default=..., description="A language of the book.")  
    ]
```

■ 좋은 문서를 작성하기 위한 방법: 명시적 객체

```
class BookEntity(BaseModel):
```

```
    country: Annotated[
```

```
        Country,  Literal 객체 사용
```

```
        Field(default=..., description="A country of the book.")
```

```
    ]
```

```
    language: Annotated[
```

```
        Language,  StrEnum 객체 사용
```

```
        Field(default=..., description="A language of the book.")
```

```
    ]
```

■ 좋은 문서를 작성하기 위한 방법: 명시적 객체

```
from enum import StrEnum  
from typing import Literal
```

```
Country = Literal["USA", "KOREA"]
```

```
class Language(StrEnum):  
    KOREAN: str = "korean"  
    ENGLISH: str = "english"
```

■ 좋은 문서를 작성하기 위한 방법: 명시적 객체

```
@router.get(
    path="",
    response_model=BookEntity,
    status_code=status.HTTP_200_OK,
)

async def get_books(
    country: Annotated[Country, Query(default=..., description="A country of book.")],
    language: Annotated[Language | None, Query(description="A language of book.")] = None
):
```

■ 좋은 문서를 작성하기 위한 방법: 명시적 객체

```
@router.get(
    path="",
    response_model=BookEntity,
    status_code=status.HTTP_200_OK,
)
async def get_books(
    country: Annotated[Country, Query(default=..., description="A country of book.")],
    language: Annotated[Language | None, Query(description="A language of book.")] = None
):
```

—————▶ 라우터 계층에서 재사용되는 객체

GET

/books Get Books

Description

1. Get all books with specific country and language.

To-do

1. Search by name with query parameter.

Parameters

Try it out

Name

Description

country * required

A country of book.

string
(query)

Available values : USA, KOREA

USA



language

A language of book.

(query)

language

Literal 객체 통해 제한된 값

GET

/books Get Books



Description

1. Get all books with specific country and language.

To-do

1. Search by name with query parameter.

Parameters

Try it out

Name

Description

country * required

A country of book.

string

Available values : USA, KOREA

(query)

USA

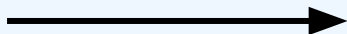


language

A language of book.

(query)

language



Optional 필드의 문제

■ 좋은 문서를 작성하기 위한 방법: 명시적 필드

```
class AuthorBase(BaseModel):  
    age: Annotated[  
        int,  
        Field(default=..., ge=19, description="The age of the author", examples=[19])  
    ]  
    email: Annotated[  
        EmailStr,  
        Field(default=..., description="An email of the author.", examples=["0417taehyun@gmail.com"])  
    ]  
    birth: Annotated[  
        date,  
        Field(default=..., description="A date of birth of the author.", examples=["1997-04-17"])  
    ]
```

■ 좋은 문서를 작성하기 위한 방법: 명시적 필드

```
class AuthorBase(BaseModel):
```

```
    age: Annotated[
```

```
        int,  19 이상의 정수
```

```
        Field(default=..., ge=19, description="The age of the author", examples=[19])
```

```
    ]
```

```
    email: Annotated[
```

```
        EmailStr,  이메일
```

```
        Field(default=..., description="An email of the author.", examples=["0417taehyun@gmail.com"]))
```

```
    ]
```

```
    birth: Annotated[
```

```
        date,  날짜
```

```
        Field(default=..., description="A date of birth of the author.", examples=["1997-04-17"]))
```

```
    ]
```

AuthorDTO ^ Collapse all **object**

firstName* > Expand all **string**

lastName* > Expand all **string**

age* ^ Collapse all **integer** **≥ 19**

The age of the author

email* ^ Collapse all **string** **email**

An email of the author.

birth* ^ Collapse all **string** **date**

A date of birth of the author.

AuthorDTO ^ Collapse all **object**

firstName* > Expand all **string**

lastName* > Expand all **string**

age* ^ Collapse all **integer** **≥ 19**

The age of the author

email* ^ Collapse all **string** **email**

An email of the author.

birth* ^ Collapse all **string** **date**

A date of birth of the author.

AuthorDTO ^ Collapse all **object**

firstName* > Expand all **string**

lastName* > Expand all **string**

age* ^ Collapse all **integer** **≥ 19**

The age of the author

email* ^ Collapse all **string** **email**

An email of the author.

birth* ^ Collapse all **string** **date**

A date of birth of the author.

AuthorDTO ^ Collapse all **object**

firstName* > Expand all **string**

lastName* > Expand all **string**

age* ^ Collapse all **integer** **≥ 19**

The age of the author

email* ^ Collapse all **string** **email**

An email of the author.


birth* ^ Collapse all **string** **date**

A date of birth of the author.

■ 좋은 문서를 작성하기 위한 방법: 명시적 필드

```
class AuthorBase(BaseModel):  
    first_name: Annotated[  
        str, Field(default=..., description="The first name of the author.", examples=["Taehyun"])]  
    ]  
    last_name: Annotated[  
        str, Field(default=..., description="The last name of the author.", examples=["Lee"])]  
    ]  
  
    @computed_field(description="A full name of the author", examples=["Taehyun Lee"])  
    @property  
    def name(self) -> str:  
        return " ".join([self.first_name, self.last_name])
```

■ 좋은 문서를 작성하기 위한 방법: 명시적 필드

```
class AuthorBase(BaseModel):  
    first_name: Annotated[  
        str, Field(default=..., description="The first name of the author.", examples=["Taehyun"])]  
    ]  
    last_name: Annotated[  
        str, Field(default=..., description="The last name of the author.", examples=["Lee"])]  
    ]  
  
    @computed_field(description="A full name of the author", examples=["Taehyun Lee"])  
    @property  
    def name(self) -> str:  다른 필드를 바탕으로 생성되는 필드  
        return " ".join([self.first_name, self.last_name])
```

AuthorEntity ^ Collapse all **object**

firstName* > Expand all **string**

lastName* > Expand all **string**

age* > Expand all **integer** **≥ 19**

email* > Expand all **string** **email**

birth* > Expand all **string** **date**

id* > Expand all **string**

name* > Expand all read-only **string**

AuthorEntity ^ Collapse all **object**

firstName* > Expand all **string**

lastName* > Expand all **string**

age* > Expand all **integer** **≥ 19**

email* > Expand all **string** **email**

birth* > Expand all **string** **date**

id* > Expand all **string**

name* > Expand all read-only **string**

■ 좋은 문서를 작성하기 위한 방법: 명시적 분리

API 버전 관리

■ 좋은 문서를 작성하기 위한 방법: 명시적 분리

```
from fastapi import FastAPI
```

```
from src.constant import APIPath  
from src.router.v1 import book
```

```
app = FastAPI(  
    title="The first version of API",  
    version="1.0.0",  
)
```

```
from fastapi import FastAPI
```

```
from src.constant import APIPath  
from src.router.v2 import author, book
```

```
app = FastAPI(  
    title="The second version of API",  
    version="2.0.0",  
)
```

■ 좋은 문서를 작성하기 위한 방법: 명시적 분리

```
from fastapi import FastAPI
```

```
from src.constant import APIVersion
```

```
from src.router import v1, v2
```

```
app = FastAPI()
```

```
app.mount(path=APIVersion.V1.value, app=v1.app, name="The first version of API")
```

```
app.mount(path=APIVersion.V2.value, app=v2.app, name="The second version of API")
```

The first version of API

1.0.0

OAS 3.1

[/v1/openapi.json](#)

Describe the fundamental concept of DFD(Documentation-First Development).

[Contact the developer](#)

Servers

/v1



The second version of API

2.0.0

OAS 3.1

[/v2/openapi.json](#)

Describe how to write a good code-based documentation.

[Contact the developer](#)

Servers

/v2



■ 좋은 문서를 작성하기 위한 방법: 모델 재사용

```
{  
  "detail": [  
    {  
      "type": "enum",  
      "loc": ["query", "language"],  
      "msg": "Input should be 'korean' or 'english'",  
      "input": "swedish",  
      "ctx": { "expected": "'korean' or 'english'" }  
    }  
  ]  
}
```

■ 좋은 문서를 작성하기 위한 방법: 모델 재사용

```
{  
  "detail": [  
    {  
      "type": "enum",  
      "loc": ["query", "language"],  
      "msg": "Input should be 'korean' or 'english'",  
      "input": "swedish",  
      "ctx": { "expected": "'korean' or 'english'" }  
    }  
  ]  
}
```

■ 좋은 문서를 작성하기 위한 방법: 모델 재사용

```
class UnauthorizedErrorResponse(ErrorResponse):
```

```
    class Config:
```

```
        json_schema_extra = {
```

```
            "examples": [
```

```
                {
```

```
                    "detail": {
```

```
                        "loc": ["authors", "token"],
```

```
                        "msg": "user is unauthorized.",
```

```
                        "type": ErrorType.UNAUTHORIZED.value,
```

```
                    }
```

```
                }
```

```
            ]
```

```
        }
```

■ 좋은 문서를 작성하기 위한 방법: 모델 재사용

```
class UnauthorizedErrorResponse(ErrorResponse):
```

```
    class Config:
```

```
        json_schema_extra = {
```

```
            "examples": [  동일한 구조의 오류 응답에 대해 예시만 변경
```

```
                {
```

```
                    "detail": {
```

```
                        "loc": ["authors", "token"],
```

```
                        "msg": "user is unauthorized.",
```

```
                        "type": ErrorType.UNAUTHORIZED.value,
```

```
                    }
```

```
                }
```

```
            ]
```

```
        }
```

■ 좋은 문서를 작성하기 위한 방법: 모델 재사용

```
@router.post(
  path="",
  responses={
    status.HTTP_401_UNAUTHORIZED: {"model": UnauthorizedErrorResponse},
    status.HTTP_403_FORBIDDEN: {"model": ForbiddenErrorResponse},
    status.HTTP_404_NOT_FOUND: {"model": NotFoundErrorResponse},
    status.HTTP_500_INTERNAL_SERVER_ERROR: {"model": InternalServerErrorResponse}
  }
)
```

—————→ 예시만 다른 모델을 인자로 전달

401

Unauthorized

Media type

application/json ▾

[Example Value](#) | [Schema](#)

```
{
  "detail": {
    "loc": [
      "authors",
      "token"
    ],
    "msg": "user is unauthorized.",
    "type": "unathorized"
  }
}
```

403

Forbidden

Media type

application/json ▾

[Example Value](#) | [Schema](#)

```
{
  "detail": {
    "loc": [
      "authors",
      "token"
    ],
    "msg": "user is forbidden.",
    "type": "forbidden"
  }
}
```

404

Not Found

Media type

application/json ▾

[Example Value](#) | [Schema](#)

```
{
  "detail": {
    "loc": [
      "books",
      "A002"
    ],
    "msg": "entity is not found.",
    "type": "not_found"
  }
}
```

401 Unauthorized

Media type

application/json

Example Value | Schema

```
{
  "detail": {
    "loc": [
      "authors",
      "token"
    ],
    "msg": "user is unauthorized.",
    "type": "unauthorized"
  }
}
```

403 Forbidden

Media type

application/json

Example Value | Schema

```
{
  "detail": {
    "loc": [
      "authors",
      "token"
    ],
    "msg": "user is forbidden.",
    "type": "forbidden"
  }
}
```

→ 예시만 다른 모델

404 Not Found

Media type

application/json

Example Value | Schema

```
{
  "detail": {
    "loc": [
      "books",
      "A002"
    ],
    "msg": "entity is not found.",
    "type": "not_found"
  }
}
```

■ 좋은 문서를 작성하기 위한 방법: 모델 재사용

```
from typing import TypeVar, Generic, Annotated
```

```
Data = TypeVar("Data", bound=BaseModel)
```

```
class DataResponse(BaseModel, Generic[Data]):  
    data: Annotated[Data, Field(default=..., description="A data object.")]
```

```
class GetDataResponse(DataResponse, Generic[Data]):  
    pass
```


■ 좋은 문서를 작성하기 위한 방법: 모델 재사용

```
from typing import TypeVar, Generic, Annotated
```

```
Data = TypeVar("Data", bound=BaseModel) —————> 각 모델을 제네릭으로 전달
```

```
class DataResponse(BaseModel, Generic[Data]):  
    data: Annotated[Data, Field(default=..., description="A data object.")]
```

```
class GetDataResponse(DataResponse, Generic[Data]):  
    pass
```

■ 좋은 문서를 작성하기 위한 방법: 모델 재사용

```
@router.get(  
    path="/{id}",  
    response_model=GetDataResponse[v2.AuthorEntity],  
    status_code=status.HTTP_200_OK,  
)
```

—————▶ 제네릭으로 전달된 모델

```
@router.get(  
    path="",  
    response_model=GetMultipleDataResponse[v2.BookEntity],  
    status_code=status.HTTP_200_OK,  
)
```

—————▶ 제네릭으로 전달된 모델

Code

Description

200

Successful Response

Media type

application/json



Controls **Accept** header.

Example Value | Schema

```
{
  "data": {
    "firstName": "Taehyun",
    "lastName": "Lee",
    "age": 19,
    "email": "0417taehyun@gmail.com",
    "birth": "1997-04-17",
    "id": "A0001",
    "name": "Taehyun Lee"
  }
}
```

Code

Description

200

Successful Response

Media type

application/json



Controls **Accept** header.

Example Value | Schema

```
{
  "data": [
    {
      "name": "Demian",
      "country": "USA",
      "language": "korean"
    }
  ],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 100
  }
}
```



통일성 덕분에 인해 예측 가능해진 응답 구조

■ 명시성과 통일성의 장점

앞서 사례로 살펴본 명시성과 통일성의 장점은?

■ 명시성과 통일성의 장점

요구사항 변경, 설계 수정 등이 있을 때
소스 코드가 곧 문서가 되어 사용

■ 명시성과 통일성의 장점

책 관련 데이터에 국가와 언어가 추가 됐어요.

■ 명시성과 통일성의 장점

책 관련 데이터에 국가와 언어가 추가 됐어요.

- 국가와 언어는 현재 값이 정해져 있나?

■ 명시성과 통일성의 장점

책 관련 데이터에 국가와 언어가 추가 됐어요.

- 국가와 언어는 현재 값이 정해져 있나? → Enum 또는 Literal 객체 필요성

■ 명시성과 통일성의 장점

책 관련 데이터에 국가와 언어가 추가 됐어요.

- 국가와 언어는 현재 값이 정해져 있나?
- 조회할 때 국가와 언어로 필터링이 되어야 하나?

■ 명시성과 통일성의 장점

책 관련 데이터에 국가와 언어가 추가 됐어요.

- 국가와 언어는 현재 값이 정해져 있나?
- 조회할 때 국가와 언어로 필터링이 되어야 하나?
- 추가 해야 할 클라이언트 오류 응답은 없나?

→ 재사용을 통해 예측 가능한 응답의 필요성

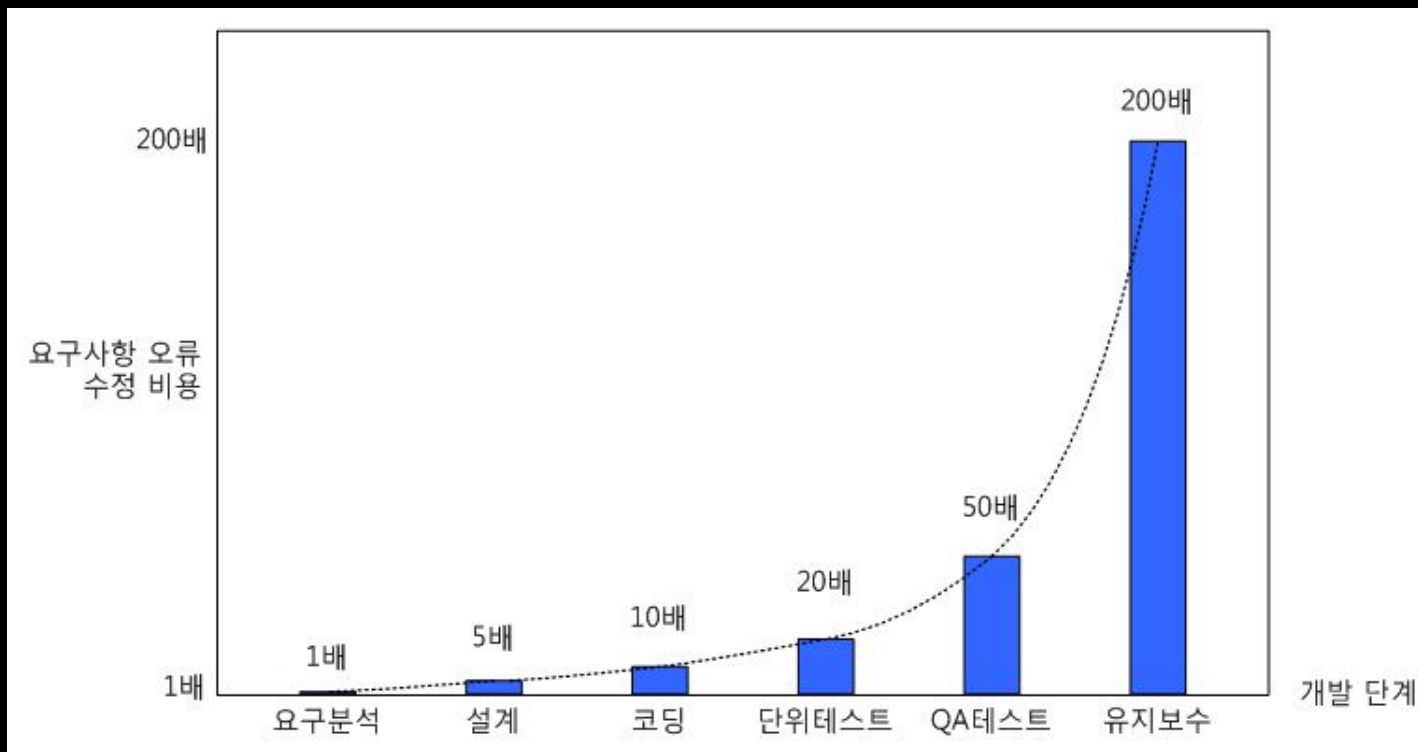
■ 명시성과 통일성의 장점

도메인에 대한 이해를 바탕으로 요청과 응답 모델링

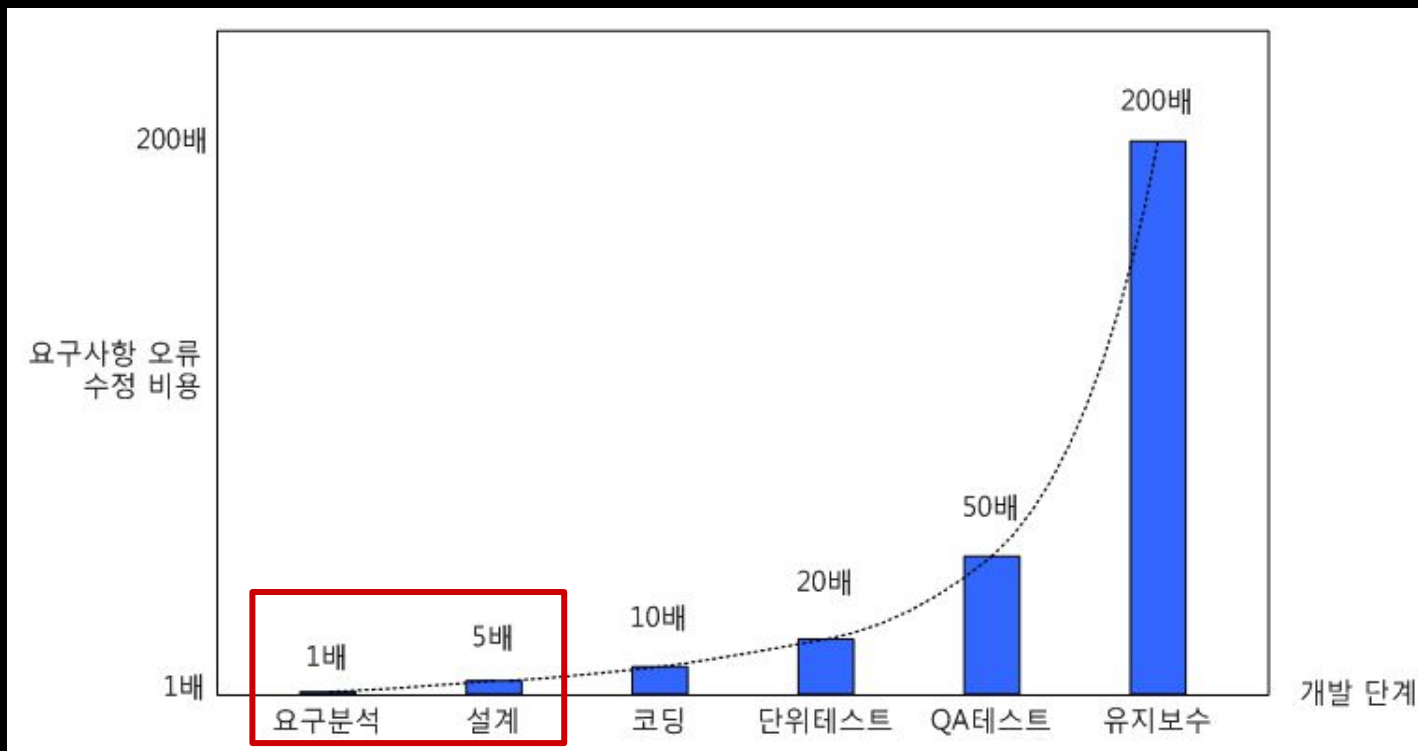
■ 명시성과 통일성의 장점

문서를 쓰듯 Pydantic 및 FastAPI 모델링

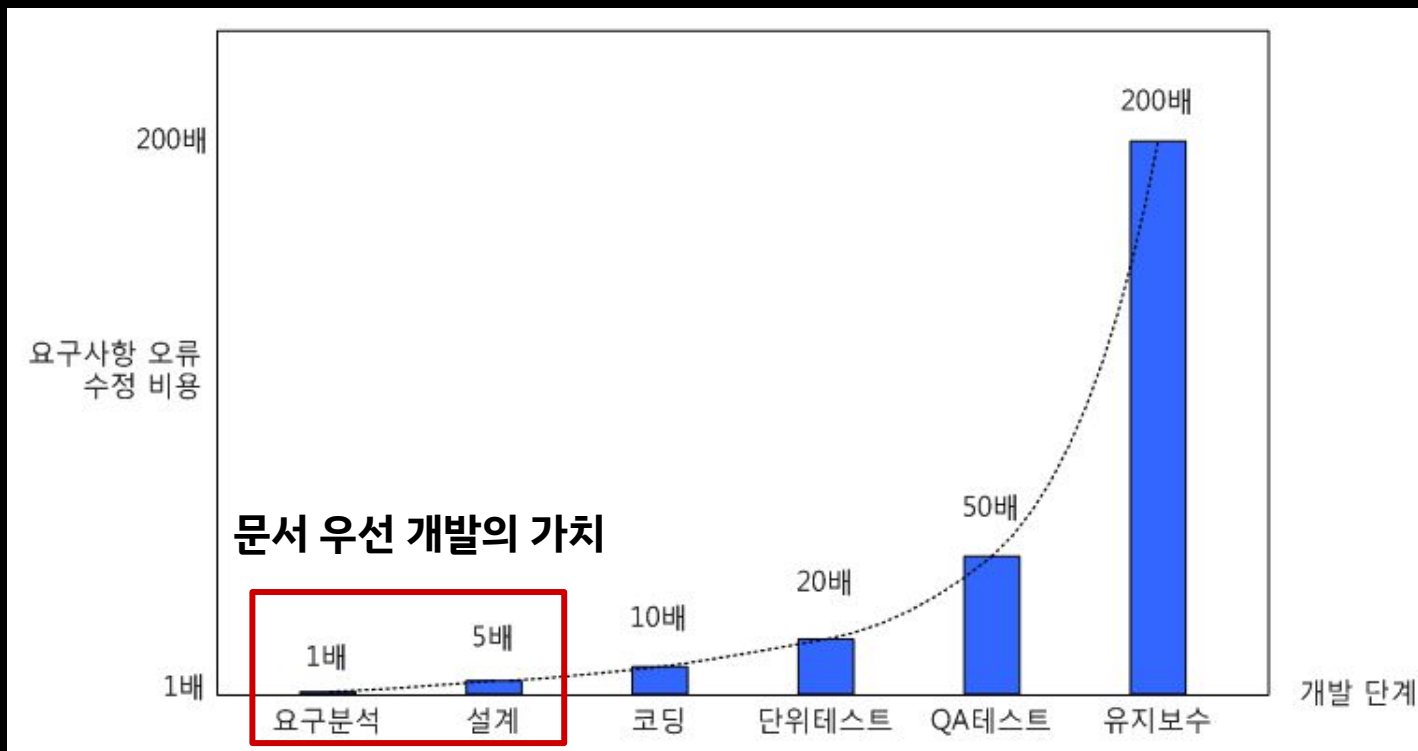
명시성과 통일성의 장점



명시성과 통일성의 장점



명시성과 통일성의 장점



■ FastAPI 애플리케이션의 보안 취약점

자동으로 생성되는 문서

■ FastAPI 애플리케이션의 보안 취약점

자동으로 생성되는 문서

`/openapi.json`

`/docs`

`/redoc`

■ API 문서 노출의 문제점

소프트웨어의 취약점 탐지 가능

■ 문서의 보안을 위한 문서 미노출

```
app = FastAPI(openapi_url=None)
```

```
app = FastAPI(docs_url=None)
```

```
app = FastAPI(redoc_url=None)
```

■ 문서의 보안을 위한 문서 미노출

`app = FastAPI(openapi_url=None)` → 문서 완전 미노출

`app = FastAPI(docs_url=None)` → `/openapi.json` 및 `/redoc` 경로 노출

`app = FastAPI(redoc_url=None)` → `/openapi.json` 및 `/docs` 경로 노출

■ 문서의 보안을 위한 엔드포인트 미노출

```
app.include_router(  
    router=event.router,  
    prefix=APIPath.EVENTS.value,  
    include_in_schema=False  
)
```

■ 문서의 보안을 위한 엔드포인트 미노출

```
app.include_router(  
    router=event.router,  
    prefix=APIPath.EVENTS.value,  
    include_in_schema=False  
)
```

—————▶ Open API 명세에 미포함

Slack Webhook 등의 문서 노출이 필요 없는 API에 유용

■ 문서의 보안을 위한 서브 애플리케이션 패턴

```
app.mount(path=APIBoundary.PRIVATE.value, app=private.app, name="A private API")
```

```
app.mount(path=APIBoundary.PUBLIC.value, app=public.app, name="A public API")
```


■ 문서의 보안을 위한 서브 애플리케이션 패턴

```
app.mount(path=APIBoundary.PRIVATE.value, app=private.app, name="A private API")
```

```
app.mount(path=APIBoundary.PUBLIC.value, app=public.app, name="A public API")
```

경로 별로 방화벽 설정 등을 다르게 하여 보안 강화 가능

문서의 보안을 위한 미들웨어

```
_IP_ALLOW_LIST: Final[list[str]] = ["127.0.0.1"]
```

```
async def validate_client_ip_address(request: Request) -> bool:
```

```
    ip_address: str = request.client.host
```

```
    if ip_address not in _IP_ALLOW_LIST:
```

```
        raise HTTPException(
```

```
            status_code=status.HTTP_403_FORBIDDEN,
```

```
            detail="Forbidden request."
```

```
        )
```

```
    return True
```

문서의 보안을 위한 미들웨어

```
_IP_ALLOW_LIST: Final[list[str]] = ["127.0.0.1"]
```

```
async def validate_client_ip_address(request: Request) -> bool:
```

```
    ip_address: str = request.client.host
```

```
    if ip_address not in _IP_ALLOW_LIST:  API 문서 접근 때 클라이언트 IP 주소 검증
```

```
        raise HTTPException(  
            status_code=status.HTTP_403_FORBIDDEN,  
            detail="Forbidden request."  
        )
```

```
    return True
```

■ 문서의 보안을 위한 미들웨어

```
@app.get(path="/openapi.json", include_in_schema=None)
async def get_open_api_json(_: Annotated[bool, Depends(dependency=validate_client_ip_address)]):
    return get_openapi(title=app.title, version=app.version, routes=app.routes)
```

```
@app.get(path="/docs", include_in_schema=None)
async def get_swagger_ui(_: Annotated[bool, Depends(dependency=validate_client_ip_address)]):
    return get_swagger_ui_html(openapi_url="/private/openapi.json", title=app.title)
```

■ 문서의 보안을 위한 미들웨어

```
@app.get(path="/openapi.json", include_in_schema=None)
async def get_open_api_json(_: Annotated[bool, Depends(dependency=validate_client_ip_address)]):
    return get_openapi(title=app.title, version=app.version, routes=app.routes)
```

—————> get_openapi 함수 활용하여 /openapi.json 경로 사용자 정의

```
@app.get(path="/docs", include_in_schema=None)
async def get_swagger_ui(_: Annotated[bool, Depends(dependency=validate_client_ip_address)]):
    return get_swagger_ui_html(openapi_url="/private/openapi.json", title=app.title)
```

—————> get_swagger_ui_html 함수 활용하여 /docs 경로 사용자 정의

■ 결론: 더 나아가야 하는 점

코드 기반 문서화가 가능한 영역

■ 결론: 오늘의 이야기 정리

1. 지속적인 소프트웨어의 성장을 위한 문서의 중요성

■ 결론: 오늘의 이야기 정리

1. 지속적인 소프트웨어의 성장을 위한 문서의 중요성
2. FastAPI 및 Pydantic을 활용한 문서 우선 개발의 효용성

■ 결론: 오늘의 이야기 정리

1. 지속적인 소프트웨어의 성장을 위한 문서의 중요성
2. FastAPI 및 Pydantic을 활용한 문서 우선 개발의 효용성
3. Pydantic 객체를 활용한 명시적인 필드와 모델 작성

■ 결론: 오늘의 이야기 정리

1. 지속적인 소프트웨어의 성장을 위한 문서의 중요성
2. FastAPI 및 Pydantic을 활용한 문서 우선 개발의 효용성
3. Pydantic 객체를 활용한 명시적인 필드와 모델 작성
4. Literal 및 Generic 타입, Enum 객체를 활용한 효과적 문서 작성법

■ 결론: 오늘의 이야기 정리

1. 지속적인 소프트웨어의 성장을 위한 문서의 중요성
2. FastAPI 및 Pydantic을 활용한 문서 우선 개발의 효용성
3. Pydantic 객체를 활용한 명시적인 필드와 모델 작성
4. Literal 및 Generic 타입, Enum 객체를 활용한 효과적 문서 작성법
5. 서브 애플리케이션 패턴을 활용한 문서 분리와 보안 강화의 가능성

■ 결론: 오늘의 이야기 정리

1. 지속적인 소프트웨어의 성장을 위한 문서의 중요성
2. FastAPI 및 Pydantic을 활용한 문서 우선 개발의 효용성
3. Pydantic 객체를 활용한 명시적인 필드와 모델 작성
4. Literal 및 Generic 타입, Enum 객체를 활용한 효과적 문서 작성법
5. 서브 애플리케이션 패턴을 활용한 문서 분리와 보안 강화의 가능성
6. Depends 및 문서 조회 요청 방식 재정의 through 보안 강화 방법