

PNU Mini Bootcamp 백엔드&클라우드 과정

2일차 - FastAPI 프로젝트 구조 설계

송준우

2025년 2월 4일

멋쟁이사자처럼

1. FastAPI 프로젝트 구조 이해

1.1 FastAPI 개발환경 구성

1.2 FastAPI 기초

1.2.1 FastAPI 기초

1.2.2 Path Parameters

1.2.3 Query Parameters

1.2.4 Request Body

1.2.5 파라미터 조합

1.2.6 응답모델

1. FastAPI 프로젝트 구조 이해

1.1 FastAPI 개발환경 구성

프로젝트 폴더에 가상환경 구성하기

가상환경을 만들고 FastAPI를 설치하기 위해 프로젝트 폴더를 생성합니다.
Python 가상환경의 필요성

- 의존성 패키지 분리
- Python 버전 분리
- 협업환경, 배포환경에 쉽게 설치 가능
- 컴퓨터 전역에 불필요한 패키지 설치 방지

Python 3.3 이상에서 제공하는 venv를 사용하여 가상환경을 생성합니다.

1.1 FastAPI 개발환경 구성

프로젝트 폴더에 가상환경 구성하기

```
mkdir api001
cd api001
python3 -m venv .venv
source .venv/bin/activate
(venv) pip3 install "fastapi[standard]"
```

.venv 폴더 구조

```
.venv
├── bin: 가상환경의 실행 파일과 스크립트
├── include: C확장 모듈 빌드를 위한 헤더파일
└── lib: 가상환경에서 사용되는 라이브러리
```

1.2.1 FastAPI 기초

간단한 JSON 응답을 반환하는 API 만들기

프로젝트 폴더 내에 main.py 파일을 생성하고 아래 코드를 작성합니다.

```
from fastapi import FastAPI
app = FastAPI()

@app.get("/")
def root():
    return {"message": "Hello, World!"}
```

FastAPI 서버를 실행합니다.

```
(venv) $ fastapi dev main.py
```

http://127.0.0.1:8000/ 에 접속하여 JSON 응답을 확인합니다.

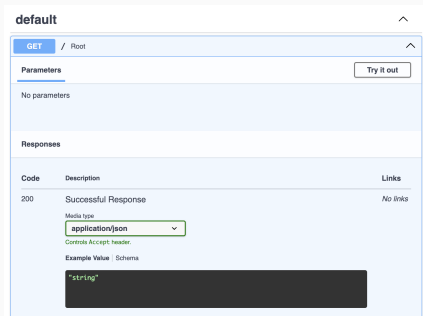
1.2.1 FastAPI 기초

API 문서 확인하기

FastAPI는 Swagger UI를 제공합니다.

Swagger UI를 통해 API 설명서를 확인할 수 있고 API 요청 테스트도 가능합니다.

<http://127.0.0.1:8000/docs> 에 접속하여 API 문서를 확인합니다.



1.2.1 FastAPI 기초

API 테스트

Try it out 버튼 → Execute 버튼을 클릭하면 API 요청이 전달되고 curl 명령어, 응답 헤더, 응답 내용을 확인할 수 있습니다.

Responses


Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/
```

Server response

Code	Details
200	<div>Response body<pre>{ "message": "Hello World!" }</pre><div> Download</div></div> <div>Response headers<pre>content-length: 26 content-type: application/json date: Tue, 21 Jan 2025 14:48:05 GMT server: uvicorn</pre></div>

1.2.1 FastAPI 기초

```
@app.get("/")
def root():
    return {"message": "Hello, World!"}
```

- @app.get()
- @app.post()
- @app.put()
- @app.delete()
- @app.patch()
- @app.options()
- @app.head()
- @app.trace()

1.2.1 FastAPI 기초

Path Parameters

main.py 파일에 아래 코드를 추가한 후

`/http://127.0.0.1:8000/docs`에 접속하여 API를 테스트해봅시다.

```
@app.get("/products/{product_id}")
def get_product(product_id):
    return {
        "products": [
            {"id": product_id, "name": "Product 1"},
        ]
    }
```

1.2.2 Path Parameters

Path Parameters

Path Parameters를 사용하는 API는 반드시 해당 파라미터를 입력해야 합니다. 아래 그림과 같이 파라미터 `products_id`에 아무값이나 입력한 후 테스트를 실행해보세요.

Parameters	
Name	Description
product_id * required (path)	<input type="text" value="product_id"/>

1.2.2 Path Parameters

Path Parameters

아래와 같이 `get_products` 함수의 `product_id` 인자에 타입을 지정하여 Path Parameter의 데이터 타입을 명시할 수 있습니다.

```
@app.get("/products/{product_id}")
def get_product(product_id: int):
    return {
        "products": [
            {"id": product_id, "name": "Product 1"},
        ]
    }
```

1.2.2 Path Parameters

Path Parameters

이번에는 Swagger UI를 사용하지 않고 웹 브라우저에서 API 주소로 아래와 같이 요청을 보내봅시다.

`http://127.0.0.1:8000/products/10`

```
{"products": [{"id": 1, "name": "Product 1"}]}
```

1.2.2 Path Parameters

Path Parameters

웹 브라우저 개발자 도구에서 network 탭을 열고
아래와 같이 Path Parameter에 문자열을 입력해봅시다.

`http://127.0.0.1:8000/products/first`

Name	× Headers	Preview	Response	Initiator	Timing
✖ first	▼ General				
		Request URL:	http://127.0.0.1:8000/products/first		
		Request Method:	GET		
		Status Code:	● 422 Unprocessable Entity		
		Remote Address:	127.0.0.1:8000		
		Referrer Policy:	strict-origin-when-cross-origin		

`https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422`

1.2.2 Path Parameters

Path Parameters

Path Parameter 규칙과 중복되는 형식의 Path를 정의하는 방법

만약 아래와 같이 /products의 하위 주소로 /products/first, /products/second, /products/third와 같이 Path Parameter 위치와 중복되는 형식의 Path를 정의하고 싶다면 어떻게 해야할까요?

```
@app.get("/products/first")
def get_first_product():
    return {"name": "Product 1"}
```

1.2.2 Path Parameters

Path Parameters

FastAPI는 코드 내에서 먼저 정의된 Path를 우선으로 라우팅을 처리합니다. 아래와 같이 `/products/first`를 먼저 정의하고 이후 `/products/product_id`를 정의하면 `/products/first`로 요청이 들어왔을 때 `get_first_product` 함수가 실행됩니다.

```
@app.get("/products/first")
def get_first_product():
    return {"name": "Product 1"}

@app.get("/products/{product_id}")
def get_product(product_id: int):
    return {
        "products": [...]
    }
```


1.2.2 Path Parameters

Path Parameters

Enum을 사용하여 Path Parameter로 전달할 수 있는 값의 종류를 제한할 수 있습니다.

```
from enum import Enum
from fastapi import FastAPI
app = FastAPI()

class CarTypes(str, Enum):
    Truck = "truck"
    Sedan = "sedan"
    SUV = "suv"

@app.get("/cars/{car_type}")
def get_car(car_type: CarTypes):
    return {"car_type": car_type}
```

1.2.3 Query Parameters

Query Parameters

main.py 파일에 아래 코드를 추가한 후 <http://127.0.0.1:8000/docs>에 접속하여 API를 테스트해봅시다.

```
@app.get("/products")
def get_products(q: str | None = None):
    products = {"products": [{"name": "Product 1"}, {"name": "Product 2"}]}
    if q:
        products.update({"q": q})
    return products
```

1.2.3 Query Parameters

Query Parameters

아래 그림과 같이 파라미터 q에 값을 입력하지 않고 테스트를 실행해보세요. 결과를 확인한 후 q에 값을 입력하고 테스트를 실행해보세요.

Parameters	
Name	Description
q string (query)	<input type="text" value="q"/>

1.2.3 Query Parameters

Query Parameters

핸들러 함수의 인자 중 Path Parameter가 아닌 인자가 정의되면 자동으로 Query Parameter로 해석됩니다.

```
@app.get("/products/{product_id}"):
def getProducts(product_id: int, name: str = '', color: str=''):
    return [{"id": product_id, "name": name, "color": color}]
```

- product_id: Path Parameter
- name, color: Query Parameter

```
/products/1?name=Linux&color=Red
```

* Query Parameter의 기본 데이터 타입은 문자열입니다.

1.2.3 Query Parameters

Query Parameters

Query Parameter는 URL의 일부이며 Path Parameter와는 다르게 필수 인자가 아닙니다. 따라서 쿼리 파라미터의 값은 None일 수 있으며 Optional 형태로 타입을 명시하는 것이 좋습니다.

```
@app.get("/products/{product_id}"):
def getProducts(product_id: int, name: str | None = None ):
    if name is None:
        return [{"id": product_id}]
    return [{"id": product_id, "name": name}]
```

```
/products/1
```

1.2.3 Query Parameters

Query Parameters

Query Parameter는 URL의 일부분이기에 데이터 타입이 문자열입니다.
하지만 FastAPI는 핸들러 함수 인자로 정의된 타입에 맞춰 자동으로 변환해줍니다.

```
@app.get("/products") :  
def getProducts(page: int=1, filter: str|None=None, nameonly: bool =  
    False) :  
    print(type(filter))  
    print(type(nameonly))  
    return [{"id": product_id, "name": name}]
```

```
/products?page=1&filter=White&nameonly=false  
/products?page=1&filter=White&nameonly=True  
/products?page=1&filter=White&nameonly=on  
/products?page=1&filter=White&nameonly=yes
```

1.2.3 Query Parameters

Query Parameters

핸들러 함수 인자로 정의한 Query Parameter에 기본값을 지정하지 않으면 필수 입력 인자가 됩니다.

```
@app.get("/products/{product_id}"):
def getProducts(product_id: int, showpicture: bool):
    return {"id": product_id, "showpicture": showpicture}
```

아래와 같이 필수 파라미터인 showpicture 값이 없으면 오류가 발생합니다.

```
/products/1
```

아래와 같이 showpicture에 값을 넣으면 정상적으로 처리됩니다.

```
/products/1?showpicture=true
```

1.2.4 Request Body

GET vs POST

Request Body에 대해 알아보기 전에 HTTP GET 요청과 POST 요청의 차이를 알아보겠습니다.

GET: 단순히 데이터를 가져오는 요청

- 웹 페이지의 HTML, JSON, 이미지, 엑셀파일 등
- 웹 브라우저 히스토리
- 파라미터를 URL로 전달(Query/Path Parameter)
- URL 길이 제한이 있음
- 서버의 상태(DB 데이터 등)를 변경하지 않음
- 검색엔진들은 GET 요청에 사용되는 리소스를 수집

1.2.4 Request Body

GET vs POST

POST: 서버에 데이터를 보내거나 서버의 상태 변경을 요청

- 회원가입, 로그인, 게시물 등록, 수정, 삭제 등
- 파일 업로드, 삭제 등
- 서버 프로그램에서 허용하는 크기만큼 데이터 전송 가능
- 웹 브라우저의 히스토리를 통해 뒤로/앞으로 시 동일작업 불가
- 캐싱 불가

1.2.4 Request Body

Request Body

회원가입, 게시물 등록과 같은 요청을 처리하기 위해서는 클라이언트에게 여러가지 정보를 전달 받아야합니다.

Query/Path Parameter로는 이전에 설명한 이유들로 충분한 파라미터를 전달 받을 수 없습니다.

이런 경우 Request Body를 통해 클라이언트가 서버로 충분한 데이터를 담아서 전달할 수 있습니다.

Request Body를 사용하려면 POST, PUT, DELETE, PATCH 메서드를 사용해야 합니다.

1.2.4 Request Body

Request Body

Request Body의 내용은 일반 텍스트, JSON 텍스트, 파일 등 여러 형식을 지원할 수 있으며 HTTP 요청헤더의 Content-Type을 통해 형식을 지정합니다.

REST API를 통해 가장 많이 사용하는 Request Body, Response Body의 형식은 JSON입니다.

Python의 Dataclass를 사용하여 Request Body를 정의해봅시다.

1.2.4 Request Body

Request Body

Login API 예시를 살펴보겠습니다.

- 아이디: 문자열
- 비밀번호: 문자열

```
{  
    "login_id": "admin",  
    "password": "1234"  
}
```

```
@dataclass  
class RequestLogin:  
    login_id: str  
    password: str
```

1.2.4 Request Body

Request Body

다음은 게시판에 작성된 게시물의 내용을 수정하는 예시입니다.

```
{  
  "id": 1,  
  "title": "New title",  
  "body": "Hello\nWorld"  
}
```

```
@dataclass  
class RequestArticle:  
    id: int  
    title: str  
    body: str
```

1.2.4 Request Body

Request Body

Request Body를 그대로 응답하는 API를 구현해봅시다.

```
from dataclasses import dataclass
from fastapi import FastAPI

@dataclass
class RequestLogin:
    user_id: str
    pwd: str

app = FastAPI()

@app.post("/auth/login")
def login(req: RequestLogin):
    return req
```

1.2.5 파라미터 조합

Request Body + Path Parameters

Request Body와 Path Parameter를 조합하여 사용할 수 있습니다.

```
from dataclasses import dataclass
from fastapi import FastAPI
from typing import Optional

@app.post("/articles/{article_id}/comments")
def add_comments(article_id: int, req: RequestAddComment):
    return {"article_id": article_id, "comment": req.body}
```

1.2.5 파라미터 조합

Request Body + Path Params + Query Params

Request Body와 Path Parameters에 Query Parameters도 조합하여 사용할 수 있습니다.

```
from dataclasses import dataclass
from fastapi import FastAPI
from typing import Optional

@app.post("/articles/{article_id}/comments?showauthoronly=true")
def add_comments(article_id: int, req: RequestAddComment,
                  showauthoronly: bool):
    return {"article_id": article_id, "comment": req, "showauthoronly":
            showauthoronly}
```


1.2.5 파라미터 조합

핸들러 함수의 매개변수 구분방법

핸들러 함수의 매개변수는

- Path Parameter로 선언되면 Path Parameter 값에 대응됨
- dataclass 또는 Pydantic과 같은 모델 클래스의 유형인 경우 Request Body로 해석함
- Path Parameter가 아니고 단일 유형(int, float, str, bool,...)의 변수인 경우 Query Parameter로 해석함

1.2.6 응답 모델

Request Body

dataclass를 이용하여 응답 JSON의 형식도 미리 정의할 수 있습니다.

응답 형식을 미리 정의해놓으면 API 문서화가 쉽고 개발 오류를 줄일 수 있습니다.

```
@dataclass
class RespUser:
    id: int
    name: str
    age: int
    email: str

@app.get("/users/{user_id}")
def get_user_profile(user_id: int) -> RespUser:
    return RespUser(id=user_id, name="Song", age=40, email="")
```