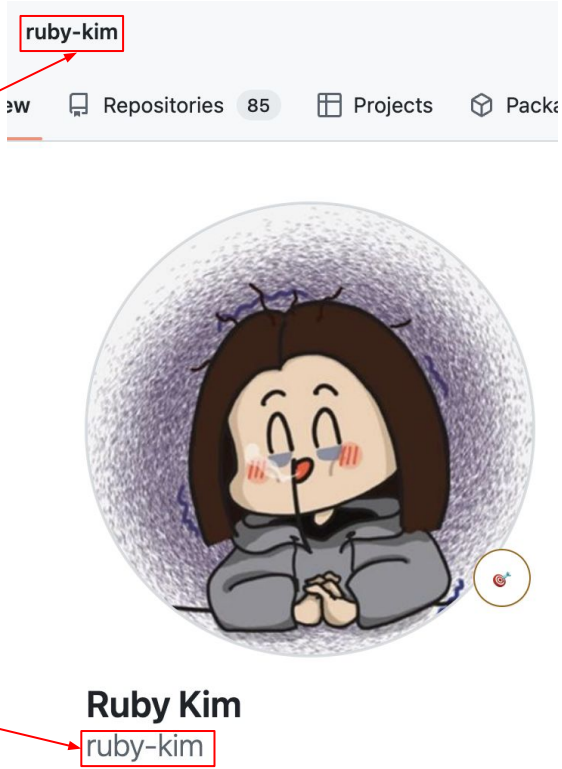
실습 파일 & 교안 받아가세요!!

- https://github.com/mentorships/Invite-guide 접속
- 가이드에 따라 repo초대 API 실행하기
- Response가 200이 나왔을 시, 이메일을 확인 후 Accept 클릭하기
- 초대 완료!

항목	설명
GitHub Username	본인의 GitHub 프로필에 있는 사용자명 (오른쪽 이미지 참고)
Repository 이름	OZ-Coding-BE16-FastAPI
X-API-KEY	oz-be16-260102



커리큘럼

- 1일차: FastAPI 개념 & Pydantic
- 2일차: 기본 CRUD
- 3일차: DB연동 및 인증 & 보안
- 4일차: 실시간 기능 (웹소켓)
- 5일차: FastAPI 프로젝트

FastAPI 특강

1일차: FastAPI 개념 & Pydantic

목 차

- FastAPI 소개
- FastAPI 기본 코드 구조
- Pydantic을 이용한 데이터 검증

FastAPI 소개

FastAPI란?

- Python으로 RESTful API를 쉽고 빠르게 개발할 수 있는 **모던 웹 프레임워크**

- 비동기(Asynchronous) 기반으로 설계되어 높은 성능 제공

- 최신 Python 기능 활용
  - 유지 보수 용이
  - 간결한 코드

FastAPI 핵심 철학

- 빠른 개발: 최소한의 코드로 동작하는 API 작성
- 고성능: Starlette와 Pydantic을 기반으로 높은 처리 속도 제공
- 강력한 타입 검사: Python 타입 힌트를 적극적으로 활용
- 자동화된 문서화: OpenAPI 명세와 Swagger UI를 자동 생성

FastAPI 주요 사용 사례

- 웹 애플리케이션: CRUD API, 대시보드, 비즈니스 로직 처리
- 마이크로서비스: 비동기와 경량 설계로 효율적인 마이크로서비스 작성
- 머신러닝 / AI 배포: FastAPI를 통해 AI모델을 웹 API로 간단히 배포

FastAPI 특징

자동화된 API 문서화

- Swagger UI와 ReDoc: API를 작성하면 OpenAPI 명세가 자동 생성됨
  - `/docs`: Swagger UI
  - `/redoc`: ReDoc 기반 API 문서

- 문서화 작업 없이 클라이언트와 개발자 간의 협업을 빠르게 지원

FastAPI 특징

비동기 처리 (Asynchronous Programming)

- async/await를 기본적으로 지원: 높은 요청 처리량 보장
- 동시 요청 처리에 유리: DB 쿼리, 외부 API 호출 같은 I/O 작업에서 효율적

데이터 검증 및 직렬화

- Pydantic 모델을 통해 데이터 검증과 직렬화를 자동 처리
- 클라이언트 요청(Request) 데이터를 Python 객체로 변환
  + 응답(Response) 데이터를 JSON으로 자동 직렬화

FastAPI 특징

경량 설계

- FastAPI는 필요할 때만 컴포넌트를 추가해 사용할 수 있는 경량 프레임워크
- 주요 구성 요소:
    - Uvicorn: ASGI 서버
    - Starlette: 웹 프레임워크
    - Pydantic: 데이터 검증

FastAPI 특징

타입 힌트 기반 개발

- Python의 타입 힌트를 활용하여 개발자가 타입 오류를 줄이고 유지보수를 쉽게 할 수 있도록 지원

```python
def add_numbers(a: int, b: int) -> int:
    return a + b
```

FastAPI 설치 및 환경 구성

- FastAPI 설치

  `pip install fastapi uvicorn`

- FastAPI 실행

  `uvicorn main:app --reload`

- 개발 환경 설정
  - IDE: VSCode 또는 PyCharm 추천
  - API 테스트 도구: Postman, Swagger UI 또는 cURL

FastAPI 기본 코드 구조

[실습 1] `hello` API 만들기

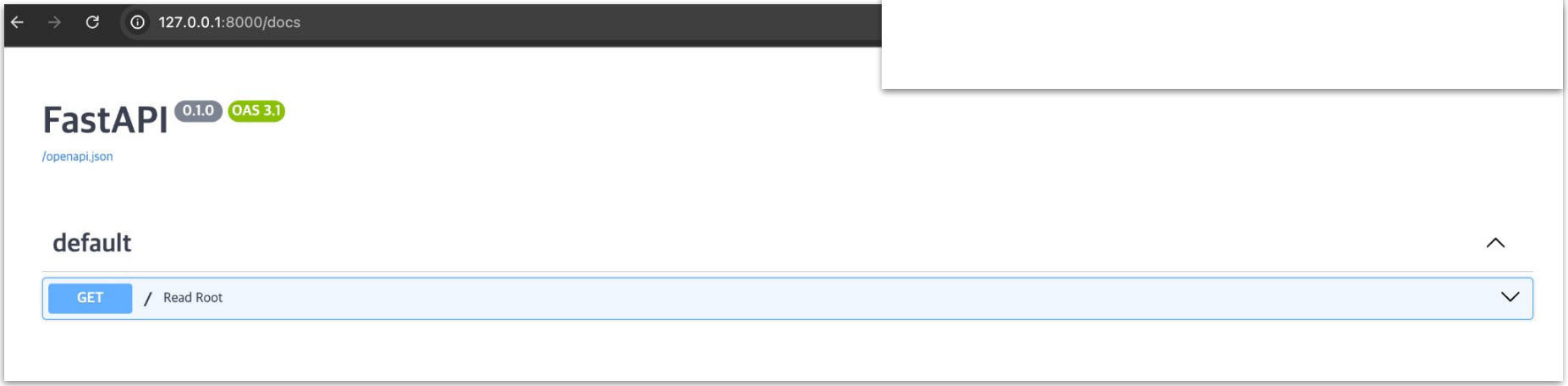
```python
from fastapi import FastAPI

# FastAPI 애플리케이션 인스턴스 생성
app = FastAPI()

# 루트 엔드포인트 정의 (GET 요청)
@app.get("/")
def hello():
    # JSON 응답 반환
    return {"message": "Hello, FastAPI!"}
```

FastAPI 설치 및 환경 구성

[실습 1] `hello` API 만들기



FastAPI 기본 코드 구조와 주요 엔드포인트

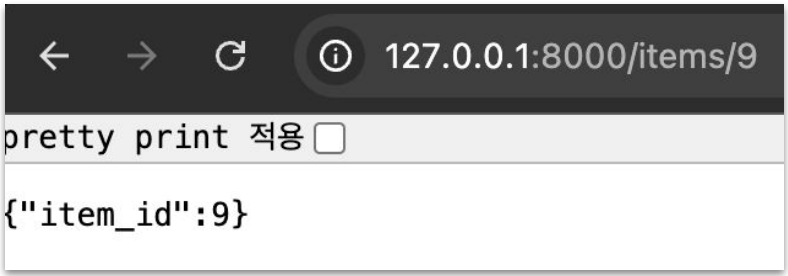
FastAPI 경로 매개변수 (Path Parameters)

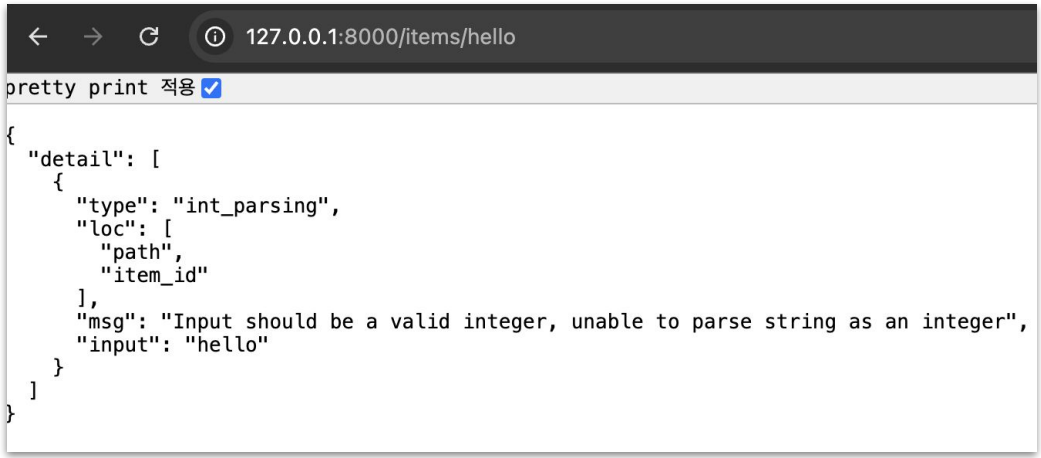
- 경로 매개변수란?
  - URL 경로의 일부로 동적인 값을 전달
  - 예: `/items/9`에서 9는 동적인 값

```python
@app.get("/items/{item_id}")
def read_item(item_id: int):
    return {"item_id": item_id}
```

FastAPI 경로 매개변수 (Path Parameters)

- 경로 매개변수란?
  - URL 경로의 일부로 동적인 값을 전달
  - 예: `/items/9`에서 9는 동적인 값





FastAPI 경로 매개변수 (Path Parameters)

[실습 2] `/users` API 만들기

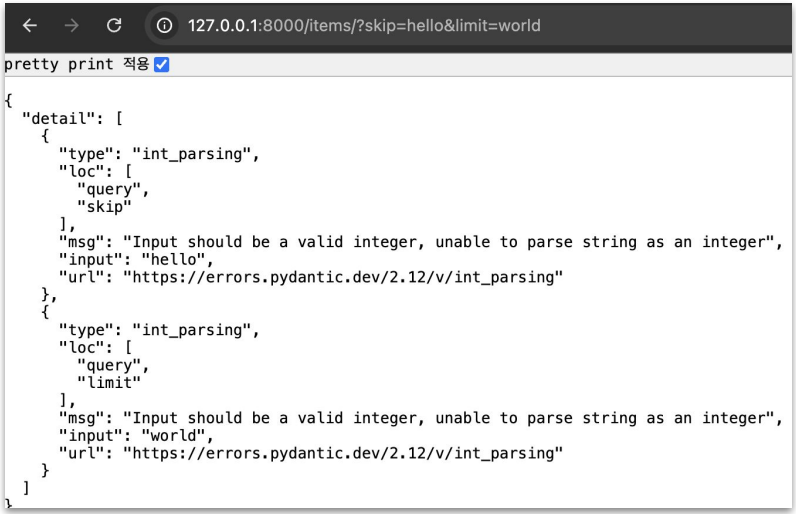
FastAPI 쿼리 매개변수 (Query Parameters)

- 쿼리 매개변수란?
  - URL에서 `?key=value` 형식으로 전달되는 값
  - 경로 매개변수와 달리 선택적으로 사용할 수 있음

```python
@app.get("/items/")
def read_items(skip: int = 0, limit: int = 10):
    return {"skip": skip, "limit": limit}
```

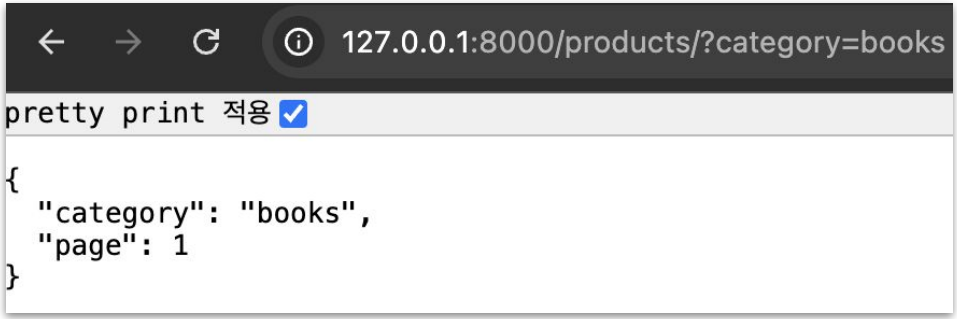
FastAPI 쿼리 매개변수 (Query Parameters)

- 쿼리 매개변수란?
  - URL에서 `?key=value` 형식으로 전달되는 값
  - 경로 매개변수와 달리 선택적으로 사용할 수 있음



FastAPI 쿼리 매개변수 (Query Parameters)

[실습 3] `/products` API 만들기



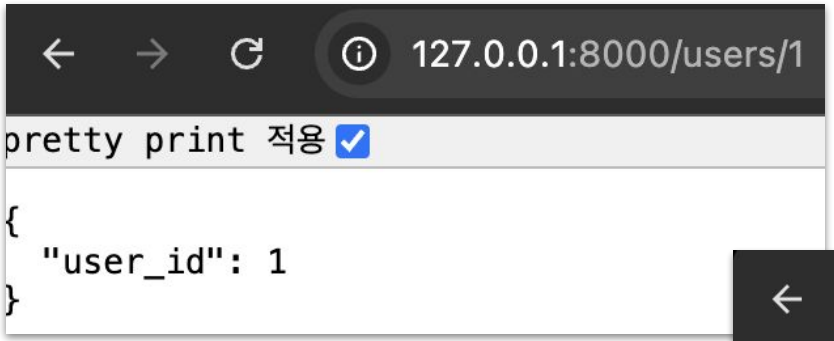
FastAPI 경로 매개변수 + 쿼리 매개변수 조합

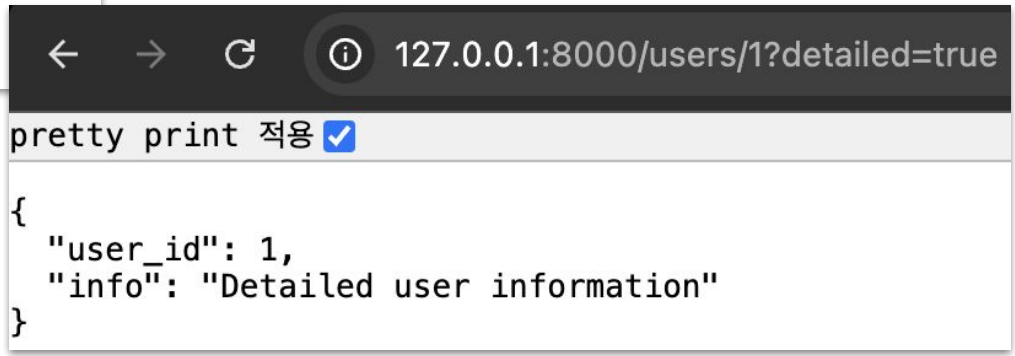
- `user_id`는 경로 매개변수, `detailed`는 쿼리 매개변수

```python
@app.get("/users/{user_id}")
def get_user(user_id: int, detailed: bool = False):
    if detailed:
        return {"user_id": user_id, "info": "Detailed user information"}
    return {"user_id": user_id}
```

FastAPI 경로 매개변수 + 쿼리 매개변수 조합

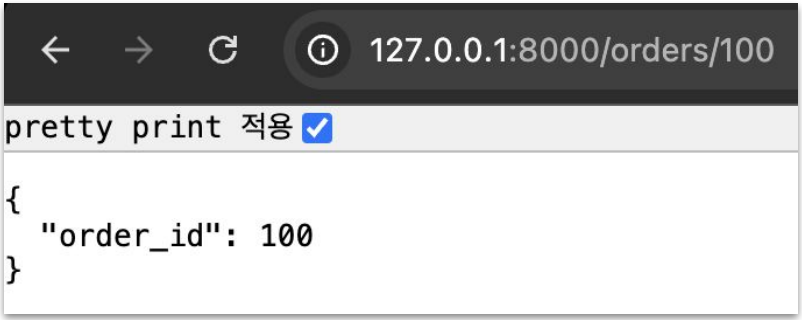
- `user_id`는 경로 매개변수, `detailed`는 쿼리 매개변수

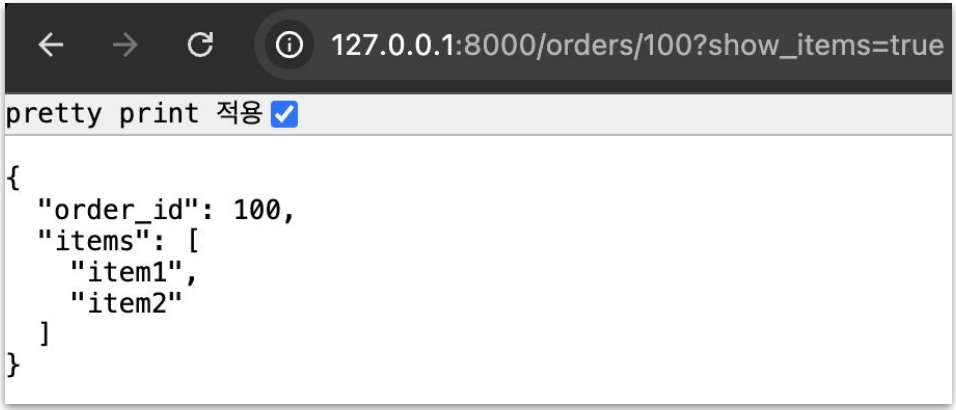




FastAPI 경로 매개변수 + 쿼리 매개변수 조합

[실습 4] `/orders` API 만들기





FastAPI 비동기 처리 (Asynchronous)

비동기의 개념

- 동기(Synchronous): 작업이 순차적으로 실행되며, 하나의 작업이 완료되어야 다음 작업이 시작됨

- **비동기(Asynchronous)**: 작업이 비차단적으로 실행되며, 다른 작업이 진행되는 동안 특정 작업은 대기 상태로 전환됨
  - Python의 `async/await`를 사용하여 비동기 처리를 구현

FastAPI 비동기 처리 (Asynchronous)

FastAPI와 비동기

- FastAPI는 기본적으로 비동기를 지원하며, 비동기 작업에서 성능을 극대화할 수 있음

- 주로 **I/O 작업**(예: 데이터베이스 쿼리, API 호출 등)에서 유리

FastAPI 비동기 처리 (Asynchronous)

[실습 5] `/async-items` API 만들기

```python
import asyncio

@app.get("/async-items/")
async def get_async_items():
    await asyncio.sleep(2)  # 2초 지연
    return {"message": "This is an async response"}
```

Pydantic을 이용한 데이터 검증

Pydantic 기본 개념 및 모델 정의

Pydantic이란?

- FastAPI에서 데이터 검증과 직렬화를 담당하는 핵심 라이브러리
- Python 타입 힌트를 기반으로 데이터를 자동 검증 및 변환

Pydantic 기본 개념 및 모델 정의

기본 데이터 모델 정의: 성공 시 (200)

코드 정의

```python
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    is_offer: bool = False
```

```python
@app.post("/items/")
def create_item(item: Item):
    return {"item": item}
```

요청 JSON

```json
{
  "name": "Book",
  "price": 12.99
}
```

응답 JSON

```json
{
    "item": {
        "name": "Book",
        "price": 12.99,
        "is_offer": false
    }
}
```

Pydantic 기본 개념 및 모델 정의

기본 데이터 모델 정의: 요청에 잘못된 데이터가 포함되었을 때 - 실패

코드 정의

```python
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    is_offer: bool = False
```

```python
@app.post("/items/")
def create_item(item: Item):
    return {"item": item}
```

요청 JSON

```json
{
  "name": "Book",
  "price": "Invalid price"
}
```

응답 JSON

```json
{
    "detail": [
        {
            "type": "float_parsing",
            "loc": [
                "body",
                "price"
            ],
            "msg": "Input should be a valid number, unable to parse string as a
                number",
            "input": "Invalid price"
        }
    ]
}
```

Pydantic 기본 개념 및 모델 정의

[실습 6] Product 모델 정의하기: 아래의 조건을 만족하도록 정의해주세요.

`/products/`로 POST 요청을 받아 새로운 상품을 생성하는 API 작성

- 필수 필드: name(문자열), price(숫자, 소수 가능)
- 선택적 필드: description(문자열, 기본값: “No description”)
- 가격(price)은 0보다 큰 값만 허용 -> 힌트: Fields 사용하기

Pydantic 기본 개념 및 모델 정의

[실습 6] Product 모델 정의하기: 아래의 조건을 만족하도록 정의해주세요.

요청 JSON

```json
{
  "name": "OZ coding",
  "price": 9.999
}
```

응답 JSON

Pydantic 기본 개념 및 모델 정의

[실습 6] Product 모델 정의하기: 아래의 조건을 만족하도록 정의해주세요.

요청 JSON

```json
{
  "name": "OZ coding",
  "price": 9.999,
  "description": "hi"
}
```

응답 JSON

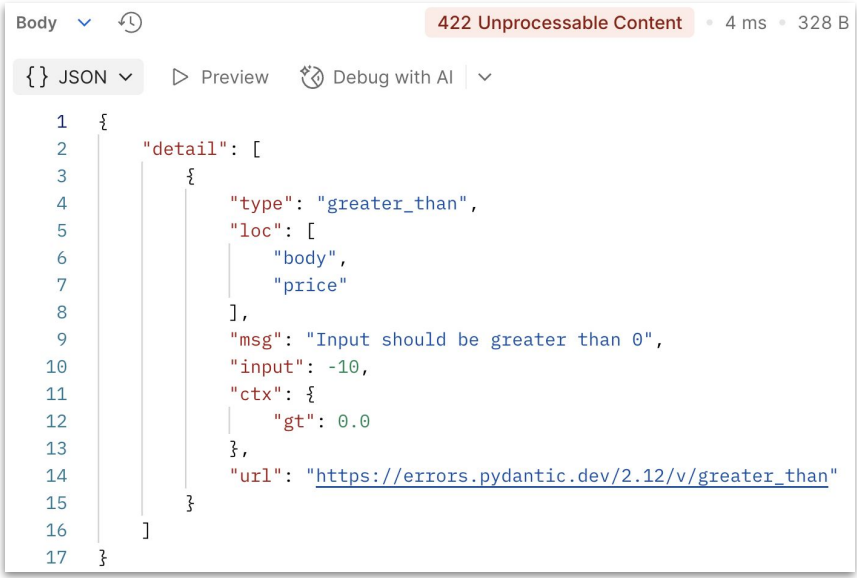
Pydantic 기본 개념 및 모델 정의

[실습 6] Product 모델 정의하기: 아래의 조건을 만족하도록 정의해주세요.

요청 JSON

```json
{
  "name": "OZ coding",
  "price": -10
}
```

응답 JSON



Pydantic 기본 개념 및 모델 정의

중첩 데이터 모델 및 유효성 검사 ex) 사용자 정보 + 집주소 관리

코드 정의

```python
class Address(BaseModel):
    city: str
    zip_code: str

class User(BaseModel):
    name: str
    age: int
    address: Address
```

```python
@app.post("/users/")
def create_user(user: User):
    return {"user": user}
```

요청 JSON

```json
{
  "name": "OZ",
  "age": 20,
  "address": {
    "city": "Seoul",
    "zip_code": "10001"
  }
}
```

응답 JSON

```json
{
    "user": {
        "name": "OZ",
        "age": 20,
        "address": {
            "city": "Seoul",
            "zip_code": "10001"
        }
    }
}
```

Pydantic 기본 개념 및 모델 정의

중첩 데이터 모델 및 유효성 검사: 요청에 잘못된 데이터가 포함되었을 때

코드 정의

```python
class Address(BaseModel):
    city: str
    zip_code: str

class User(BaseModel):
    name: str
    age: int
    address: Address
```

```python
@app.post("/users/")
def create_user(user: User):
    return {"user": user}
```

요청 JSON

```json
{
  "name": "OZ",
  "age": 20,
  "address": {
    "city": "Seoul",
    "zip_code": 10001
  }
}
```

응답 JSON

```json
{
    "detail": [
        {
            "type": "string_type",
            "loc": [
                "body",
                "address",
                "zip_code"
            ],
            "msg": "Input should be a valid string",
            "input": 10001,
            "url": "https://errors.pydantic.dev/2.12/v/string_type"
        }
    ]
}
```

Pydantic 기본 개념 및 모델 정의

[실습 7] Order와 Item 모델 정의하기: 아래의 조건을 만족하도록 모델들을 정의해주세요.

- Item 모델
  - 필드: name (문자열), quantity (정수, 1 이상)
- Order 모델
  - 필드: id (정수), items (여러 개의 Item 객체), -> 힌트: List 사용하기
    total_price (소수, 0 이상)

`/orders/`로 POST 요청을 받아 주문 데이터를 생성하는 API 작성

- 최소 2개의 상품을 포함하는 요청 JSON 작성

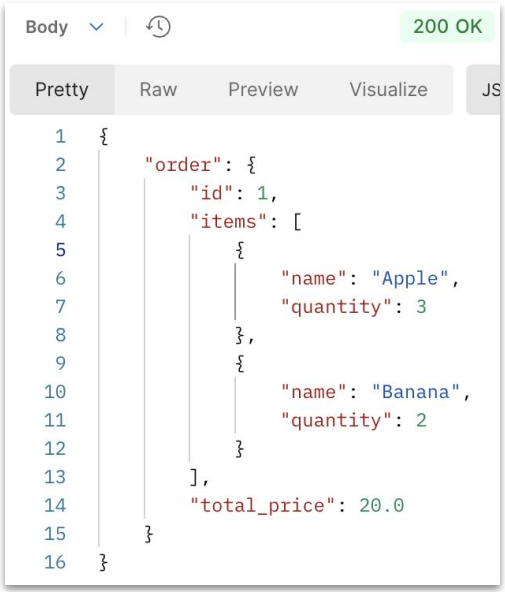
Pydantic 기본 개념 및 모델 정의

[실습 7] Order와 Item 모델 정의하기: 아래의 조건을 만족하도록 모델들을 정의해주세요.

요청 JSON

```
{
    "id": 1,
    "items": [
        {"name": "Apple", "quantity": 3},
        {"name": "Banana", "quantity": 2}
    ],
    "total_price": 20.0
}
```

응답 JSON



Pydantic 의 validator 개념 및 사용법

validator란?

- Pydantic에서 데이터 모델의 특정 필드에 대해 **추가적인 검증 로직**을 구현할 때 사용하는 데코레이터

- 단순한 데이터 타입 검증 이상으로, 조건을 커스터마이징하거나 값 변환을 수행할 수 있음

Pydantic 의 validator 개념 및 사용법

validator의 기본 사용법

- 구조
  - `@field_validator("<필드 이름>")`, `@classmethod` 데코레이터를 사용하여 특정 필드의 값을 검증
  - 검증 함수는 항상 클래스 메서드 형태로 정의되며, 첫 번째 인자로 필드값을 받음
  - 검증 실패 시 예외를 발생시켜야 함: ValueError, TypeError 등

Pydantic 의 validator 개념 및 사용법

validator의 기본 사용법

- 구조 예시

```python
from pydantic import BaseModel, field_validator

class ExampleModel(BaseModel):
    number: int

    @field_validator("number")
    @classmethod
    def validate_number(cls, value):
        if value < 1:
            raise ValueError("Number must be at least 1")
        if value > 100:
            raise ValueError("Number must not exceed 100")
        return value
```

Pydantic 의 validator 개념 및 사용법

validator의 기본 사용법

- 동작
  - 모델 초기화 시, 해당 필드의 값이 검증 로직을 통과해야 함
  - 통과하지 못하면 예외 메시지가 반환됨

```
example = ExampleModel(number=50)
print(example)
# Output: number=50
```

```
example = ExampleModel(number=150)
# ValueError: Number must not exceed 100
```

Pydantic 기본 개념 및 모델 정의

[실습 8] Reservation 모델 정의하기: 아래의 조건을 만족하도록 모델들을 정의해주세요.

- 필수 필드
  - name: 예약자 이름 (최대 50자)
  - email: 예약자 이메일
  - date: 예약 날짜 (미래 날짜만 허용) -> 힌트: `datetime.now()`
- 선택적 필드
  - special_requests: 문자열 (기본값: 빈 문자열)

Pydantic 기본 개념 및 모델 정의

[실습 8] Reservation 모델 정의하기: 아래의 조건을 만족하도록 모델들을 정의해주세요.

- 요구사항
  - `/reservations/` 로 POST 요청을 받아 예약 데이터를 생성하는 API 작성
  - 아래 데이터를 검증해주세요:
    - 날짜가 과거일 경우 에러 반환
    - 이름이 50자를 초과할 경우 에러 반환

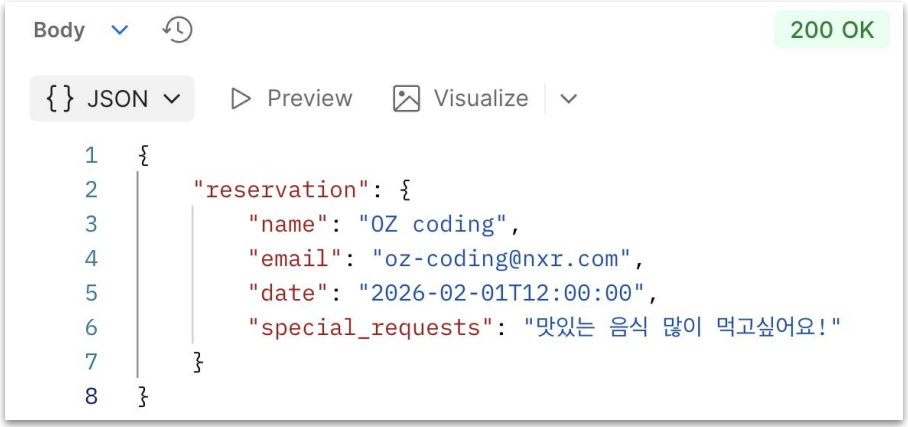
Pydantic 기본 개념 및 모델 정의

[실습 8] Reservation 모델 정의하기: 아래의 조건을 만족하도록 모델들을 정의해주세요.

요청 JSON (26년 1월 1일 오전 12시 실행)

```json
{
  "name": "OZ coding",
  "email": "oz-coding@nxr.com",
  "date": "2026-02-01T12:00:00",
  "special_requests": "맛있는 음식 많이 먹고싶어요!"
}
```

응답 JSON



Pydantic 기본 개념 및 모델 정의

[실습 8] Reservation 모델 정의하기: 아래의 조건을 만족하도록 모델들을 정의해주세요.

요청 JSON

```json
{
  "name": "OZ coding",
  "email": "oz-coding@nxr.com",
  "date": "2026-01-01T12:00:00"
}
```

응답 JSON

Body 422 Unprocessable Content 5 ms 366 B

{} JSON Preview Debug with AI


```json
{
    "detail": [
        {
            "type": "value_error",
            "loc": [
                "body",
                "date"
            ],
            "msg": "Value error, Reservation date must be in the future",
            "input": "2026-01-01T12:00:00",
            "ctx": {
                "error": {}
            },
            "url": "https://errors.pydantic.dev/2.12/v/value_error"
        }
    ]
}
```

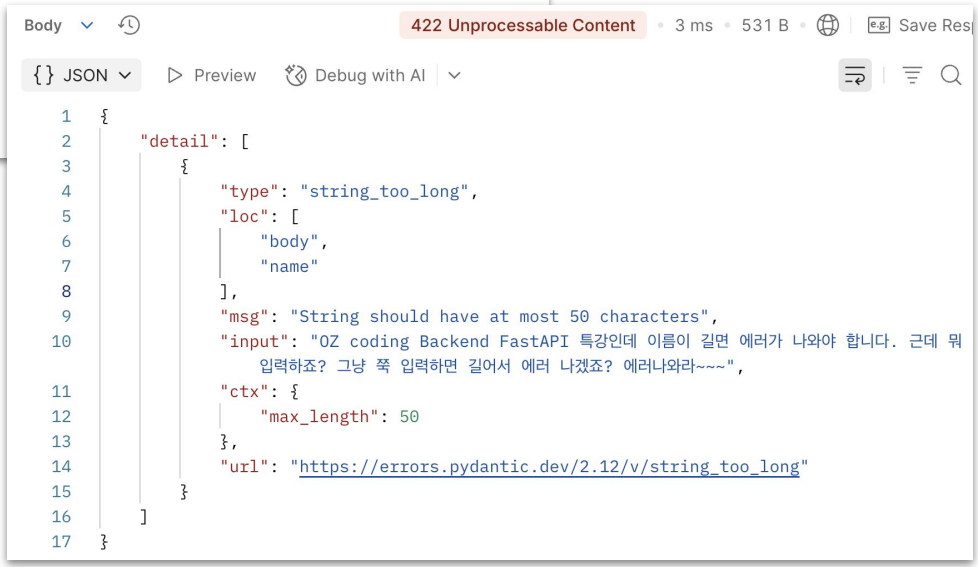
Pydantic 기본 개념 및 모델 정의

[실습 8] Reservation 모델 정의하기: 아래의 조건을 만족하도록 모델들을 정의해주세요.

요청 JSON

```
{
  "name": "OZ coding Backend FastAPI 특강인데 이름이 길면 에러가 나와야 합니다. 근데 뭐 입력하죠? 그냥 쭉
    입력하면 길어서 에러 나겠죠? 에러나와라~~~",
  "email": "oz-coding@nxr.com",
  "date": "2026-02-01T12:00:00"
}
```

응답 JSON



QnA