

GEN NX API

# MIDAS GEN NX

## API 사용자 교육

GEN NX API의 개요부터  
Postman과 Python을 활용한 기본 예제 실습까지

DATE

📅 2026. 08. 13

INSTRUCTOR

👤 김승준 프로

COURSE LEVEL

Basic



TABLE OF CONTENTS

# AGENDA

**01** API란 무엇인가?

**02** GEN NX API의 특징

**03** API 요청의 3가지 핵심 요소

**04** Postman 사용법

**05** Postman 3D-2Bay 실습

**06** Colab으로 Python 체험하기



Total 6 Sections



## 01. BASIC CONCEPT

# API란 무엇인가?

### Definition

## Application Programming Interface

애플리케이션 간 **소통**과 **기능 공유**를 위한 표준화 된 인터페이스

## 일상 속의 API

API는 이미 웹 서비스, 모바일 앱 등  
우리의 디지털 생활 전반에 깊숙이 자리 잡고 있습니다.



### OS SYSTEM

#### 운영체제 API

파일 열기/저장, 시스템 자  
원



### SOCIAL MEDIA

#### SNS 공유/로그인

간편 로그인, 게시물 공유



### NAVIGATION

#### 지도 서비스

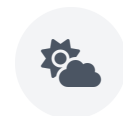
길찾기, 위치 정보 연동



### PAYMENT

#### 온라인 결제

결제 게이트웨이 연동



### WEATHER

#### 날씨 정보

실시간 기상 데이터 수신



### FINANCE

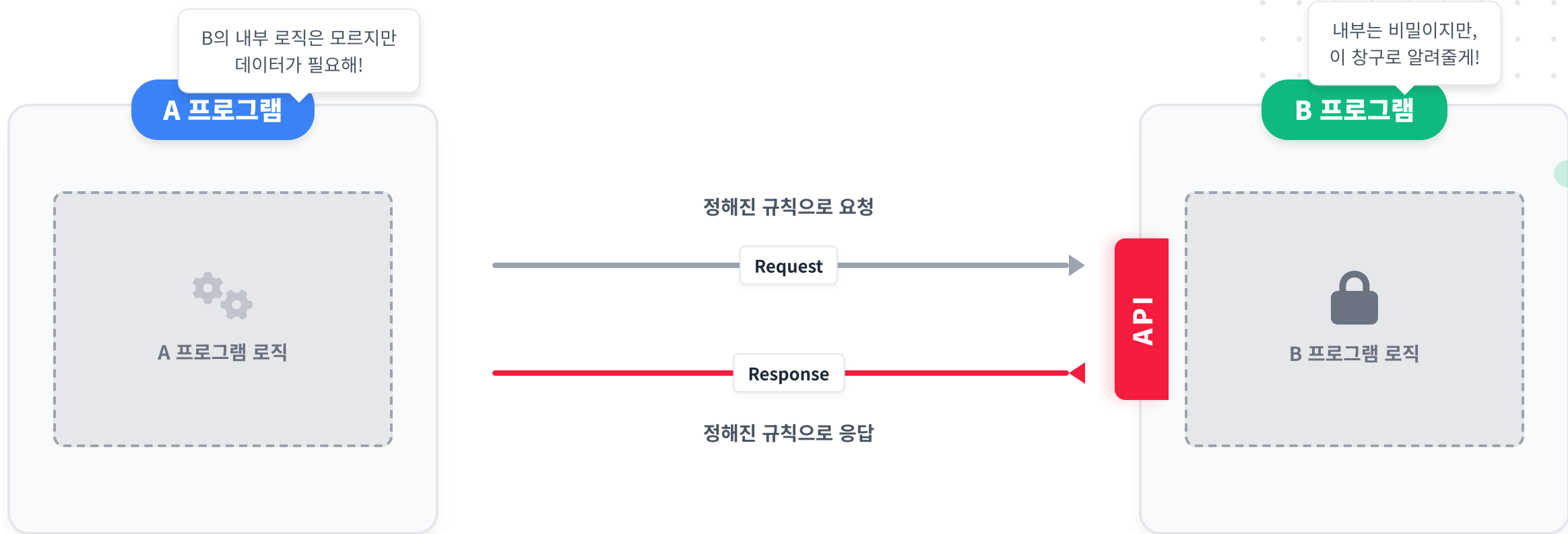
#### 금융 서비스

계좌 조회, 이체 처리

## 01-1. CONCEPT ANALOGY

# API : 프로그램 간 소통의 규칙

내부 로직을 몰라도 데이터를 주고받을 수 있는 공통된 약속

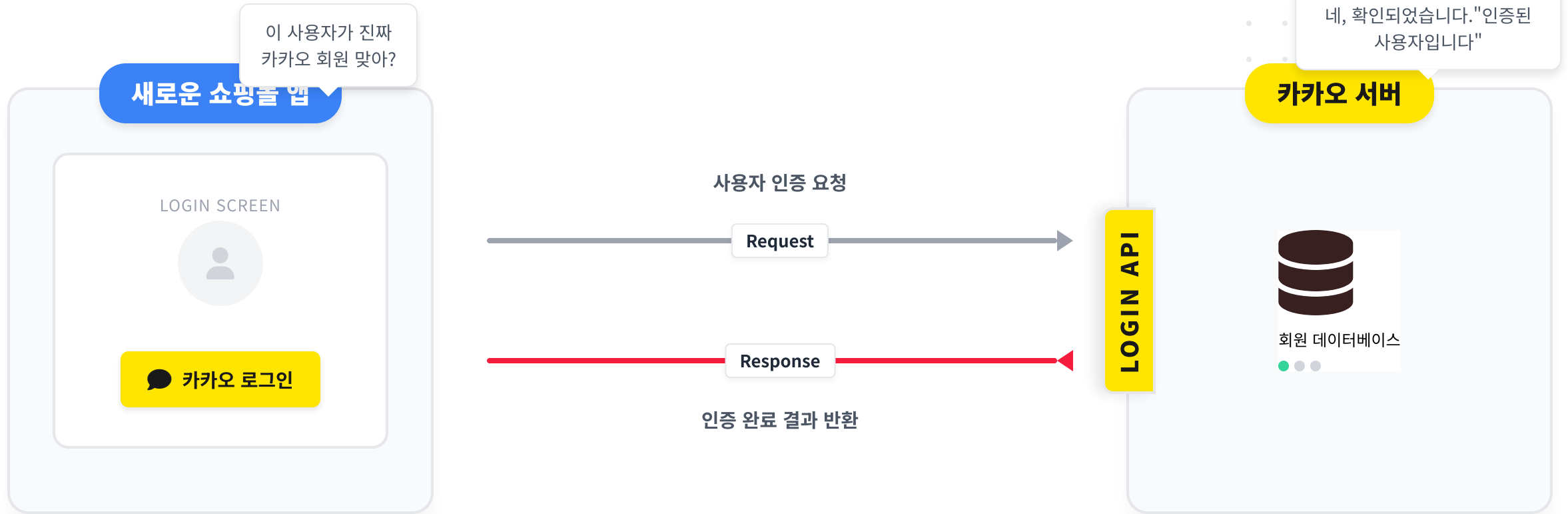


API는 서로 다른 프로그램이 **내부를 공개하지 않고도** 데이터를 주고받을 수 있게 해주는 **공통된 소통 규칙**입니다.

A는 B의 내부가 어떻게 돌아가는지 전혀 몰라도, 정해진 방식(API)으로 요청하면 원하는 결과(데이터)를 받을 수 있습니다.

# API 활용 사례 1 : 카카오 로그인

새로운 쇼핑몰 앱이 사용자 인증을 위해 카카오 서버와 통신하는 과정

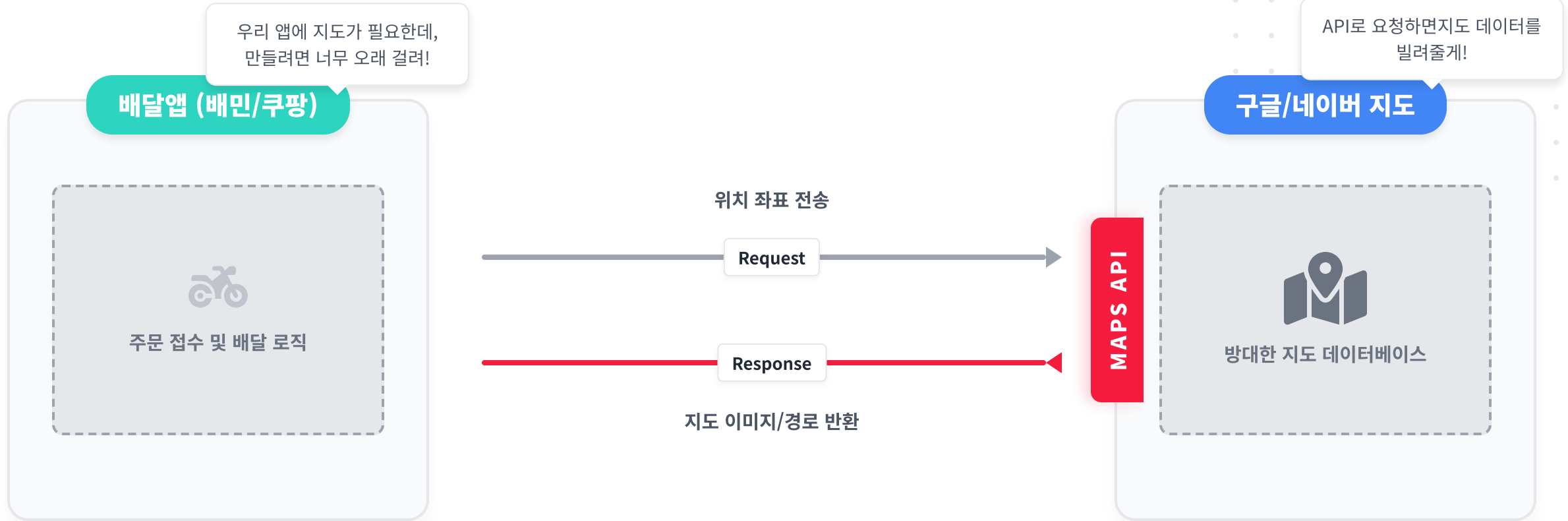


쇼핑몰 앱은 복잡한 회원가입 시스템을 직접 만들지 않고, **카카오 로그인 API** 를 통해 사용자 인증을 해결합니다.

사용자가 "카카오 로그인" 버튼을 누르면, 앱은 카카오 API 창구로 인증 요청을 보내고, 카카오 서버가 "이 사람 인증된 사용자 맞아요"라는 확인 결과를 API를 통해 전달해줍니다.

## API 활용 사례 2 : 배달 앱 지도

직접 지도를 만들지 않고도, API를 통해 지도 기능을 빌려 쓰는 방식



배달앱은 **지도 제작 기술이 없어도** 구글/네이버의 지도 API를 통해 **완성된 지도 기능**을 제공합니다.

이처럼 API를 활용하면 타사의 핵심 기능을 마치 내 앱의 일부처럼 사용하여 개발 시간과 비용을 획기적으로 줄일 수 있습니다.

# 그렇다면, **GEN NX API**는 왜 필요한가?

카카오 로그인·배달앱 지도가 그러하듯 — GEN NX의 핵심 기능을 ‘가져다’ 쓰는 개발 방식

66

**GEN NX API**라는 규칙을 활용하면,  
복잡한 시스템을 직접 만들지 않고 **GEN NX의 핵심 기능**을  
내가 만들려는 프로그램으로 그대로 가져와 사용하여,  
**개발 시간과 비용을 획기적으로 줄일 수 있습니다.**

## 02. KEY FEATURES

# GEN NX API의 특징

누구에게나 열려있는 무한한 가능성, 연결의 자유



### 언어 독립성

특정 언어에 종속되지 않고, 다양한 프로그래밍  
환경과 연동 가능

Python

Excel VBA

C#

Postman



### REST API 기반

전 세계적으로 가장 널리 쓰이는 API 규칙 준수  
(Representational State Transfer)

HTTP Method

Stateless

Standard



### JSON 데이터 표준

모든 명령어와 데이터는  
전 세계적으로 통용되는 웹 표준  
데이터 형식인 'JSON'으로 교환

{ Key : Value }

Lightweight

## 02-1. ARCHITECTURE

# REST API 이해하기



전세계적으로 가장 널리쓰이는  
API 규칙(문법)의 한 종류

\* REST: **R**epresentational **S**tate **T**ransfer

## HTTP Method

요청 보낼 때 규칙

Resource에 대한 행위를 정의합니다.

GET

조회 (Read)

정보 요청

POST

생성 (Create)

새 리소스 등록

PUT

수정 (Update)

정보 갱신

DELETE

삭제 (Delete)

리소스 제거

## Status Code

응답 받을 때 규칙

요청 처리 결과를 나타내는 상태 코드입니다.

● 200

OK

요청 성공적으로 처리됨

● 201

Created

새로운 리소스가 생성됨

● 400

Bad Request

잘못된 문법으로 요청

● 404

Not Found

요청한 리소스를 찾을 수 없음

## 02-2. DATA FORMAT

# JSON 데이터 포맷

\* JSON = JavaScript Object Notation

### KEY FEATURES



#### Readable

사람과 기계 모두가 읽기 쉽고 직관적인 텍스트 기반 포맷



#### Lightweight

데이터 용량이 적어 네트워크 전송 속도가 매우 우수



#### Language Independent

Python, C#, Java, C++ 등 거의 모든 프로그래밍 언어와 호환

## 기본 구조 및 데이터 타입

JSON 데이터는 항상 Key-Value (키-값) 쌍으로 구성됩니다.

"이름" : "김승준" (String)

"키" : 180 (Number)

"결혼여부" : false (Boolean)

#### String

문자열 데이터

#### Number

숫자 데이터

#### Boolean

논리 데이터  
(true 또는 false)

#### Null

비어있는 값  
(데이터가 없음을 명시적으로 표현)

#### Object (객체)

Key와 Value 쌍들의 집합  
중괄호 {} 로 묶여서 표현됩니다.

#### Array (배열)

Value 들의 집합  
대괄호 [] 로 묶인 리스트 데이터입니다.

## 02-2. JSON DATA FORMAT

# JSON 데이터 Node Example

### SYNTAX

### JSON 기호 설명

"Key": "Value"

Key-Value 쌍으로 구성

{ }

JSON 객체(Object)를 의미

" "

Key 이름이나 문자열 Value를 감쌈

:

Key와 Value를 구분

,

여러 Key-Value 쌍을 구분

### EX

### Node Data 해설

"1"

1번 키에 해당하는 값은 X:-1, Y:-1, Z:-1의  
**Key-Value 쌍들의 집합(Object)**입니다.

"2", "3"

나머지 키들도 동일하게 각각의 **좌표 Object**를 값으로 가집니다.

Node\_Data.json

```
{
  "Assign": {
    "1": { "X": -1, "Y": -1, "Z": -1 },
    "2": { "X": -2, "Y": -2, "Z": -2 },
    "3": { "X": -3, "Y": -3, "Z": -3 }
  }
}
```

JSON Body Example: Node Generation

# API 요청의 3가지 핵심 요소

성공적인 요청을 위해 반드시 정의해야 할 3가지 질문: 어디로, 어떻게, 무엇을?



## 주소 (Where)

어디로 보낼 것인가?

명령을 전달할 대상(리소스)의  
고유한 위치 정보

URI



## 방식 (How)

어떻게 요청할 것인가?

조회, 생성, 수정, 삭제 등  
수행할 동작의 종류

HTTP Method



## 내용 (What)

무엇을 요청할 것인가?

서버에 전달할 구체적인 데이터  
또는 파라미터 정보

JSON Data Body

KEY FORMULA

Request = URI + HTTP Method + JSON Body

# URI: 명령을 전달할 고유 주소

기본 주소

상세 기능

Base URL

+

Endpoint

=

Full URI

## Base URL

API 서버의 기본 주소입니다.  
GEN NX 사용자는 모두 동일한 주소를 사용합니다.

`https://moa-engineers.midasit.com:443/gen`

Example

## Endpoint

절점, 요소, 재료 등 각 기능(리소스)에 접근하기 위한 상세 주소입니다.  
사용 가능한 모든 Endpoint는 **API Online Manual**에 명시되어 있습니다.

[/db/NODE](#) (절점 정보)  
[/db/MATL](#) (재료 정보)  
[/doc/ANAL](#) (해석 수행)

## MIDAS API Online Manual - JSON Manual

Node/Element

| No. | End Point | Details                             |
|-----|-----------|-------------------------------------|
| 1   | /db/NODE  | <a href="#">Node ↗</a>              |
| 2   | /db/ELEM  | <a href="#">Element ↗</a>           |
| 3   | /db/SKEW  | <a href="#">Node Local Axis ↗</a>   |
| 4   | /db/MADO  | <a href="#">Define Domain ↗</a>     |
| 5   | /db/SBDO  | <a href="#">Define Sub-Domain ↗</a> |
| 6   | /db/DOEL  | <a href="#">Domain-Element ↗</a>    |

# API 인증의 핵심

## MAPI-Key



### MAPI-Key

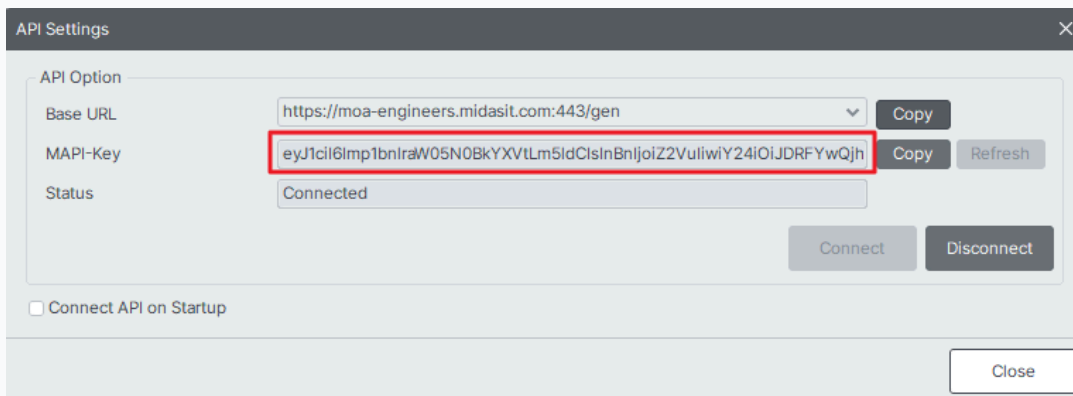
특정 제품을 식별하고 인증하기 위한

#### 임시 보안 키 (Temporary Security Key)

사용자의 PC나 네트워크 환경이 아닌, 연결된 '제품' 자체를 식별하여 권한을 부여합니다.

##### ✓ 핵심 역할

- 서버가 연결 대상(GEN NX)을 식별
- REST API 요청 시 **Header**에 필수 포함



#### 보안 주의사항 (Security Notice)

MAPI-Key는 타인에게 **노출되어서는 안 됩니다.**

유출 시 해당 키를 통해 제품이 원격 제어될 위험이 있습니다.

🔄 필요 시 언제든지 키를 재생성(Regenerate) 가능

# HTTP Methods: 4가지 요청 방식

요청의 종류를 정의하는 HTTP 메서드(Methods)와 GEN NX API 활용 예시



| </> Method | 🔑 기능        | 📄 상세설명                     | 🔗 GEN NX API 활용 예시        |
|------------|-------------|----------------------------|---------------------------|
| POST       | 생성 (CREATE) | 새로운 데이터를 서버에 추가해달라고 요청     | ➕ 새로운 절점, 요소, 재료, 하중 등 생성 |
| GET        | 조회 (READ)   | 서버에 있는 데이터를 보내달라고 요청       | 🔍 생성된 모델의 정보, 해석 결과 조회    |
| PUT        | 수정 (UPDATE) | 기존 데이터를 새로운 데이터로 변경해달라고 요청 | ✎ 특정 절점의 좌표, 재료의 물성치 변경   |
| DELETE     | 삭제 (DELETE) | 특정 데이터를 삭제해달라고 요청          | 🗑 불필요한 하중 케이스, 경계 조건 삭제   |

# GEN NX가 알아들을 수 있는 **JSON** 규격으로 전달해야 합니다

## 일상의 비유 : 설문조사 양식

### 신규 회원 등록 (예시)

이름 (Key)      홍길동 (Value)

나이 (Key)      30 (Value)

사는 곳 (Key)      서울 (Value)

 정해진 문항(Key)에 맞는 답(Value)만 작성해야  
시스템이 내용을 정확히 인식하고 처리할 수 있습니다.



## GEN NX API : JSON 규격

### Endpoint별 고유 규격

**/db/NODE**      →      점점과 관련된 Key-Value

**/db/MATL**      →      재료와 관련된 Key-Value

**/db/SECT**      →      단면과 관련된 Key-Value

 각 Endpoint마다 정해진 **Key-Value** 구조가 존재합니다.  
정해진 규격에 맞춰야만 GEN NX가 명령을 수행합니다.

## JSON **Key-Value**는 GEN NX 입력창, 선택창의 값과 1:1로 대응됩니다

assign\_section.json

```
{
  "Assign": {
    "5": {
      "SECTTYPE": "DBUSER",
      "SECT_NAME": "H-Section_DB",
      "SECT_BEFORE": {
        "USE_SHEAR_DEFORM": true,
        "SHAPE": "H",
        "DATATYPE": 1,
        "SECT_I": {
          "DB_NAME": "KS21",
          "SECT_NAME": "H100×100×6/8"
        }
      }
    }
  }
}
```

(1)

(2)

(3)

(4)

(5)

(6)

(7)



1 : 1 대응

Section Data

DB/User Value SRC Combined Tapered Composite

Section ID 5

H-Section

H-Section\_DB

DB

KS21

Sect. Name H 100×100×6/8

Built-Up Section

Get Data from Single Angle

DB Name KS21

Sect. Name

H 100 mm

B1 100 mm

tw 6 mm

tf1 8 mm

B2 0 mm

tf2 0 mm

r1 10 mm

r2 0 mm

Offset: Center-Center ☒ Consider Shear Deformation.

Change Offset ...

Show Calculation Results... OK Cancel Apply

# JSON Body는 두 가지 형식으로 나뉩니다

데이터를 저장/조회하느냐, 프로그램의 기능을 실행하느냐에 따라 규격이 달라집니다.

## TABLE FORMAT

### DB 형식 데이터

(DB Table Format)

#### 1 용도

재료·단면·절점·요소 등 모델링의 기본 정보를 다룰 때 사용

EX)

`/db/MATL` 재료    ·    `/db/SECT` 단면  
`/db/NODE` 절점    ·    `/db/ELEM` 요소

#### 2 지원 메서드

POST GET PUT DELETE

CRUD 4가지 메서드를 모두 사용합니다.

#### 3 최상위 Key

"Assign"

{ "Assign": { ... } }

VS

## OPERATION FUNCTION FORMAT

### 기능 실행 데이터

(Operation Function Format)

#### 1 용도

프로그램이 내부 계산을 수행하고 결과를 테이블에 저장하는 형식

EX)

`/ope/DIVIDEELEM` Divide Elements  
`/doc/SAVEAS` 저장    ·    `/doc/ANAL` 해석 수행

#### 2 지원 메서드

POST GET PUT DELETE

'기능 실행'의 의미만 있어 **POST만 사용** 가능합니다.

#### 3 최상위 Key

"Argument"

{ "Argument": { ... } }

# API 통신의 3가지 핵심 요소

API를 통해 데이터를 주고받기 위해 반드시 필요한 3가지 요소(주소, 방식, 내용)입니다.



## 1. 주소 (Where)

Base URL + Endpoint

데이터가 위치한  
서버의 경로(Path)입니다.



## 2. 방식 (How)

HTTP Method

데이터를 다루는 행위입니다.  
(GET, POST, PUT, DELETE)



## 3. 내용 (What)

JSON Data Body

실제 주고받는  
구체적인 데이터 내용입니다.

KEY FORMULA

Request = URI + Method + JSON Body



## 05. POSTMAN 사용법



# Postman

## 도구 소개

### DEFINITION

API 개발, 테스트, 문서화를 위한 통합 플랫폼입니다.

복잡한 코딩 없이 **직관적인 GUI 환경**을 통해  
API 요청을 손쉽게 보내고 응답 데이터를 즉시 확인할 수 있습니다.

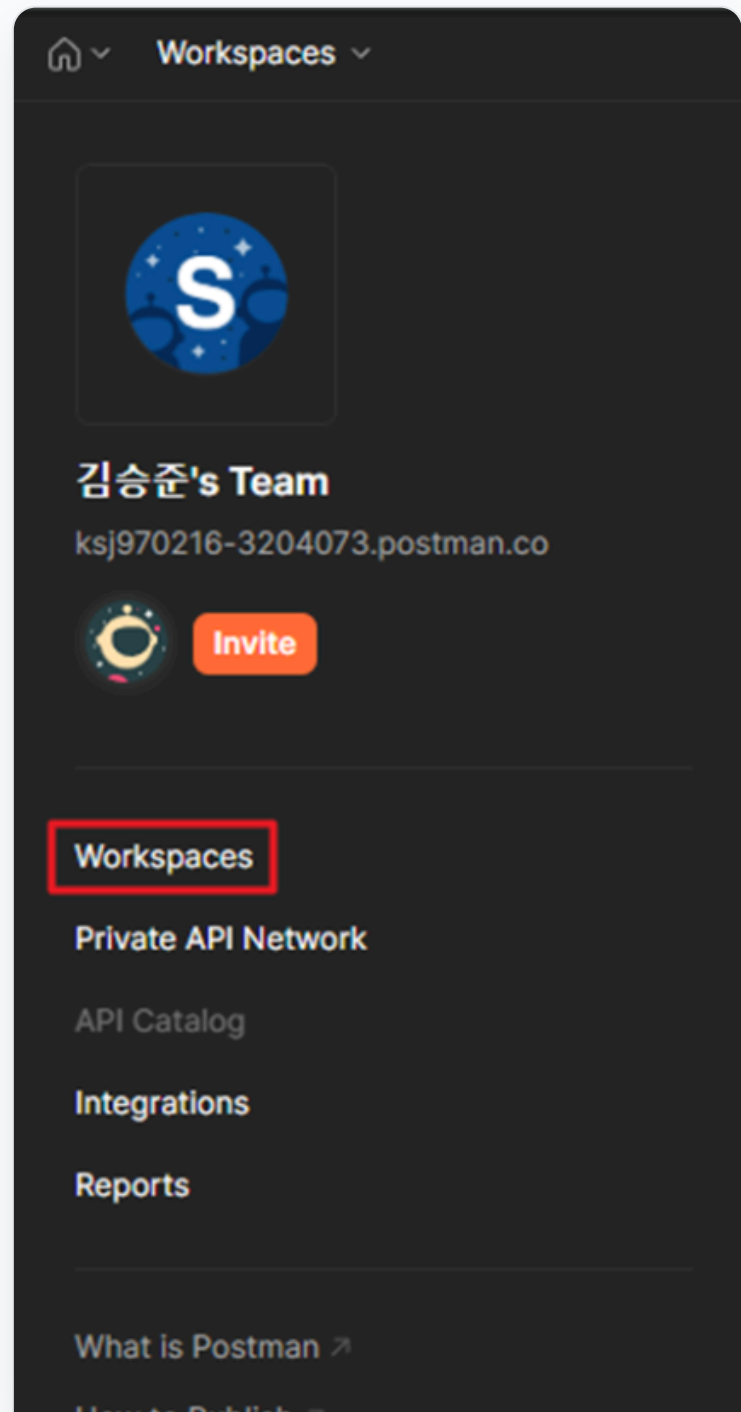
## 05. POSTMAN 사용법

# Workspaces 메뉴 접근

1

### 좌측 사이드바 메뉴 확인

Postman 실행 후,  
화면 좌측에 위치한 **Workspaces** 메뉴를 클릭합니다.



## Create workspace 버튼 클릭

1

### All workspaces 화면 확인

이전 단계에서 Workspaces 메뉴를 클릭하면, 현재 내가 접근 가능한 모든 워크스페이스 목록이 표시되는 'All workspaces' 페이지가 열립니다.

2

### 생성 버튼 클릭

화면 우측 상단에 위치한 초록색 **Create workspace** 버튼을 클릭하여 새로운 작업 공간 생성 프로세스를 시작합니다.

< Go back

## All workspaces

A directory of all workspaces in 김승준's Team you can access.

Create workspace

# Blank workspace 선택

1

## 옵션 선택

'Create your workspace' 화면에서 목록의 첫 번째에 있는 **Blank workspace** 옵션을 선택합니다.

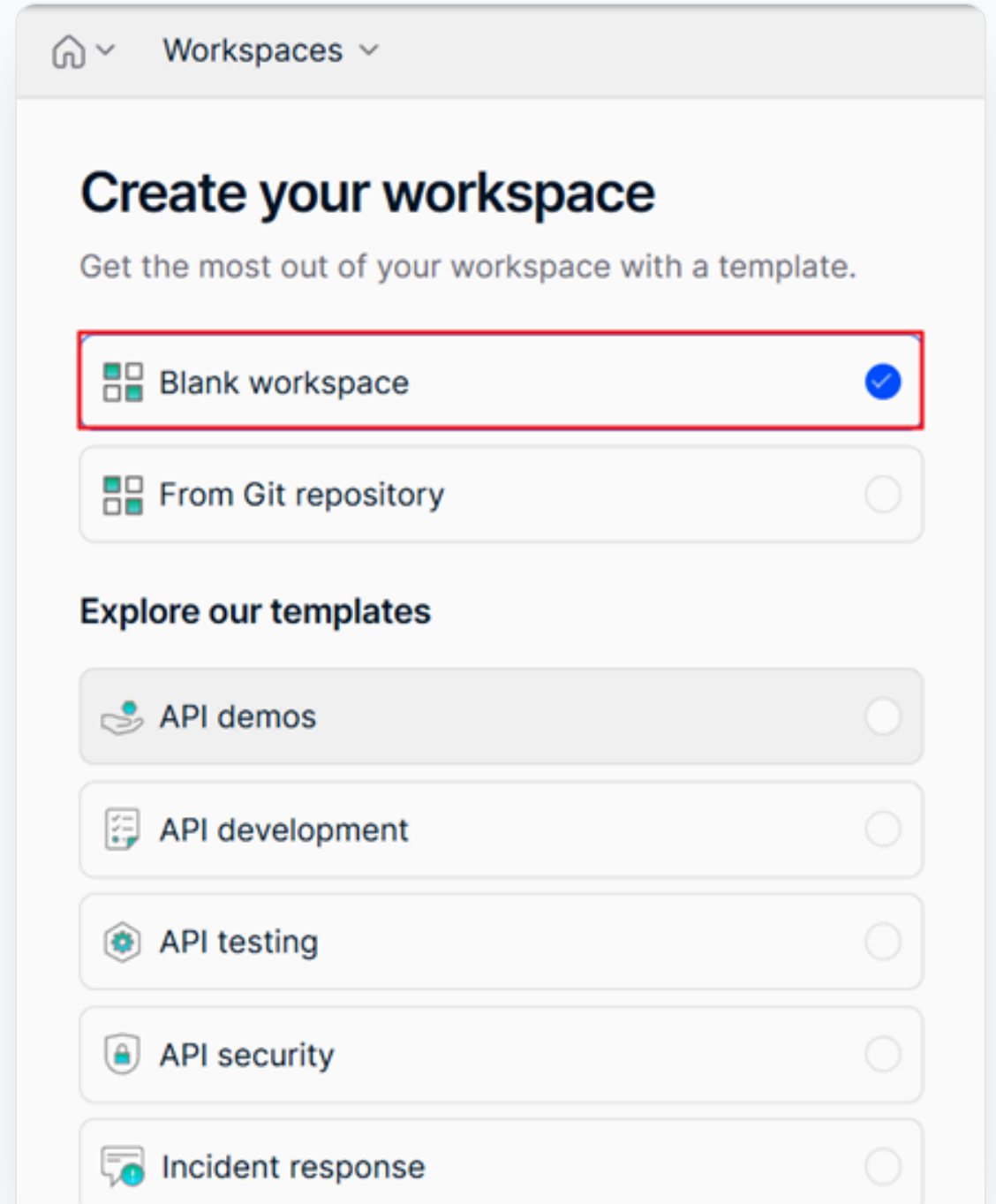
2

## 다음 단계 진행

우측 하단의 [Next] 버튼을 클릭하여 설정 화면으로 이동합니다.

### Why Blank Workspace?

기존 템플릿(API demos 등)을 사용하지 않고,  
처음부터 직접 API 환경을 구성하며 학습하기 위해 빈 작업 공간을 생성합니다.



## 05. POSTMAN 사용법

# Workspace 이름 및 설정

### Workspace 세부 정보 입력

빈 Workspace에 이름을 부여하고, 팀원들과의 공유 범위를 설정하는 단계입니다.

 Workspace가 생성되면 즉시 API 작업을 시작할 수 있습니다.

Create Workspace Settings

## Create your workspace

Name

GEN NX API 교육

Select workspace type

 Internal 

Build and test APIs within your team.

 Partner [Upgrade plan](#)

Share APIs securely with trusted partners.

 Public ☐

Make APIs accessible to the world.

Manage access

☒ Everyone in 김승준's Team  
1 member

☐ Only you and invited people

[Back](#) [Create](#)

# Postman 컬렉션 및 요청 생성

1

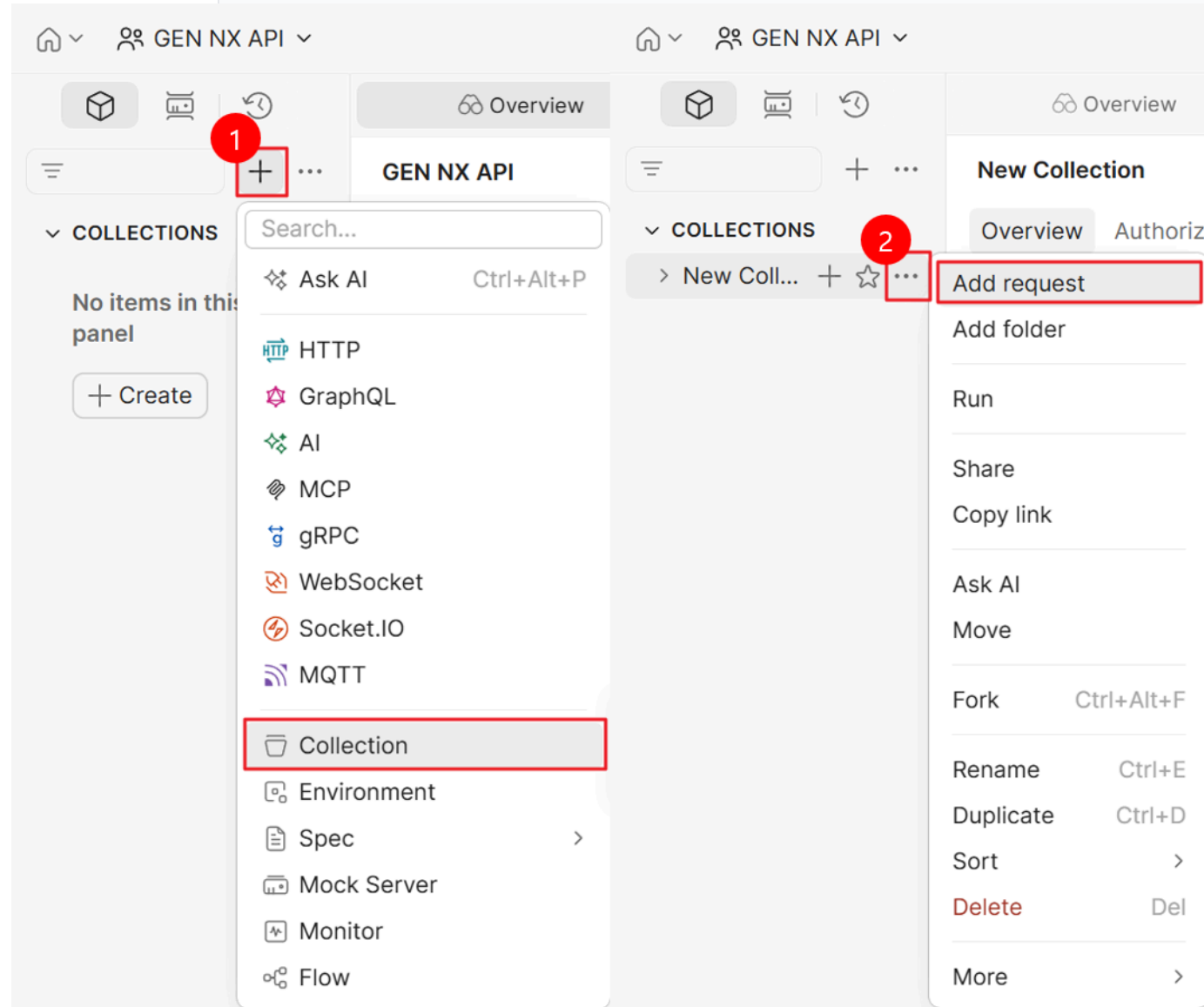
## 컬렉션(Collection) 생성

API 요청들을 관리하기 위한 폴더 개념.  
좌측 Workspace 메뉴에서 [+] 모양 버튼 클릭.  
-> **Collection** 버튼 클릭.

2

## 요청(Request) 생성

생성된 New Collection 우측 점 세 개(...) 메뉴 클릭 후 **Add Request** 클릭  
또는 생성된 New Collection 우측 [+] 모양 클릭



## 05. POSTMAN 사용법

# GEN NX & Postman 연결 설정

### 1 GEN NX API 설정 확인

GEN NX 상단 탭 [Apps] ▶ [API] ▶ [API Settings]로 이동하여  
Base URL 과 MAPI-Key 를 복사

### 2 Base URL 입력

Postman URL 창에 Base URL을 입력합니다.  
<https://moa-engineers.midasit.com:443/gen>

### 3 Headers 탭 선택

URL 입력창 아래의 [Headers] 탭을 클릭합니다.

### 4 Key 입력

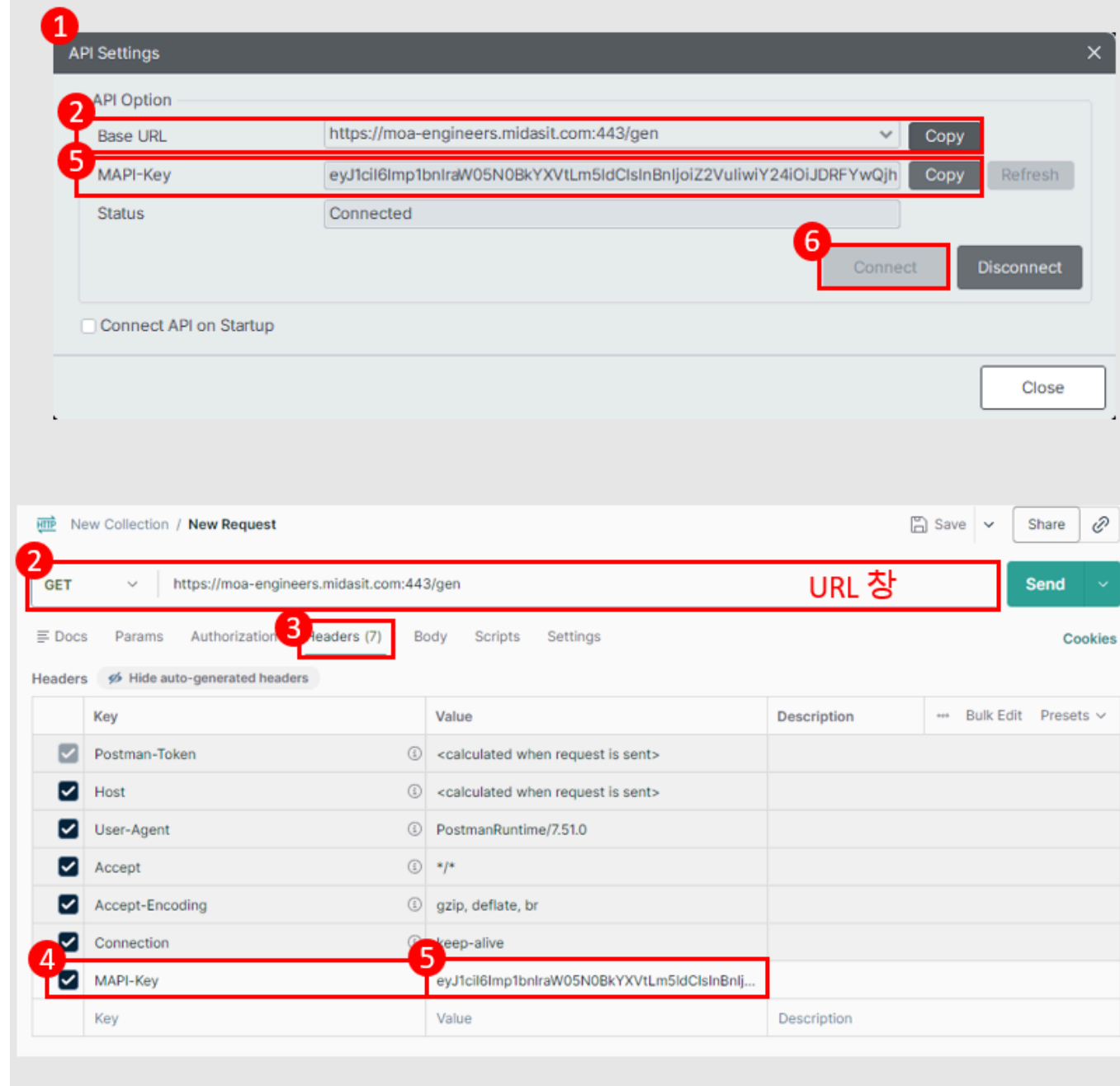
Headers 테이블의 Key 컬럼에 MAPI-Key 를 입력합니다.

### 5 Value 입력

Value 컬럼에 GEN NX API Settings에서 복사한 MAPI-Key 값을 붙여넣습니다.

### 6 연결 (Connect)

GEN NX API Settings 창의 [Connect] 버튼을 클릭합니다.  
\* 연결 성공 시 Status가 Connected로 표시됩니다.



## 05. POSTMAN 사용법

# 실습: GEN NX 단위계(UNIT) 정보 '읽기'

### 1 HTTP 메서드 및 URL 설정

Method: **GET** 선택

URL: **Base URL** + **/db/UNIT** 입력

### 2 요청 전송

우측 상단의 **Send** 버튼을 클릭하여 서버로 요청을 보냅니다.

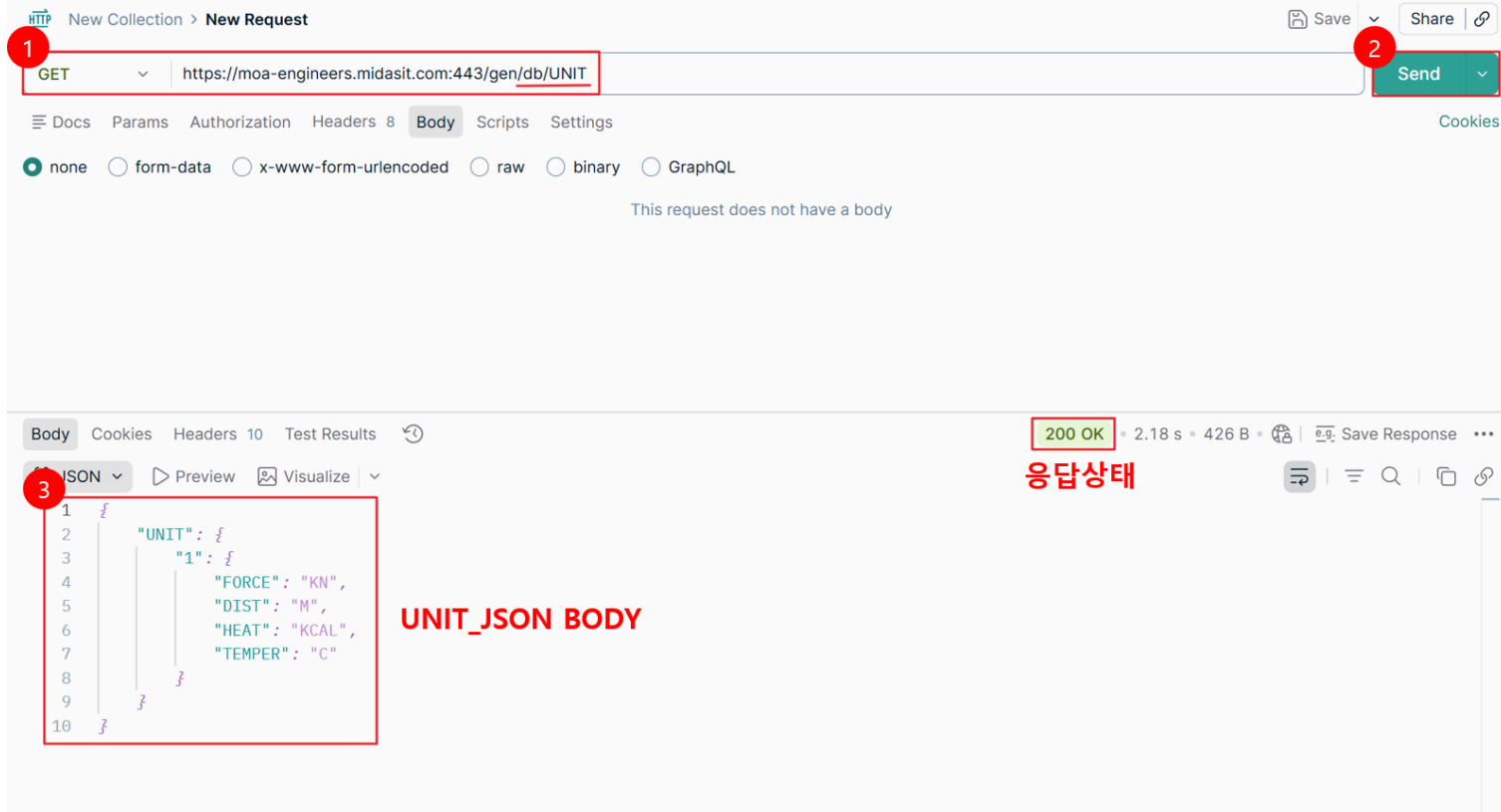
### 3 응답(Response) 확인

하단 Body 탭의 JSON 데이터 확인

상태 코드 **200 OK** 및 응답 시간 체크

#### ✓ Check Point

- 응답받은 JSON의 UNIT(FORCE, DIST 등) 값이 실제 GEN NX 프로그램의 설정과 일치하는지 검증하세요.



## 05. POSTMAN 사용법

# 실습: GEN NX 단위계(UNIT)정보 '수정하기'

### 1 HTTP 메서드 및 URL 설정

PUT 메서드를 선택합니다. (생성 또는 수정)  
URL 및 Header는 이전 GET 요청과 동일하게 유지합니다.

### 2 Body 데이터 설정

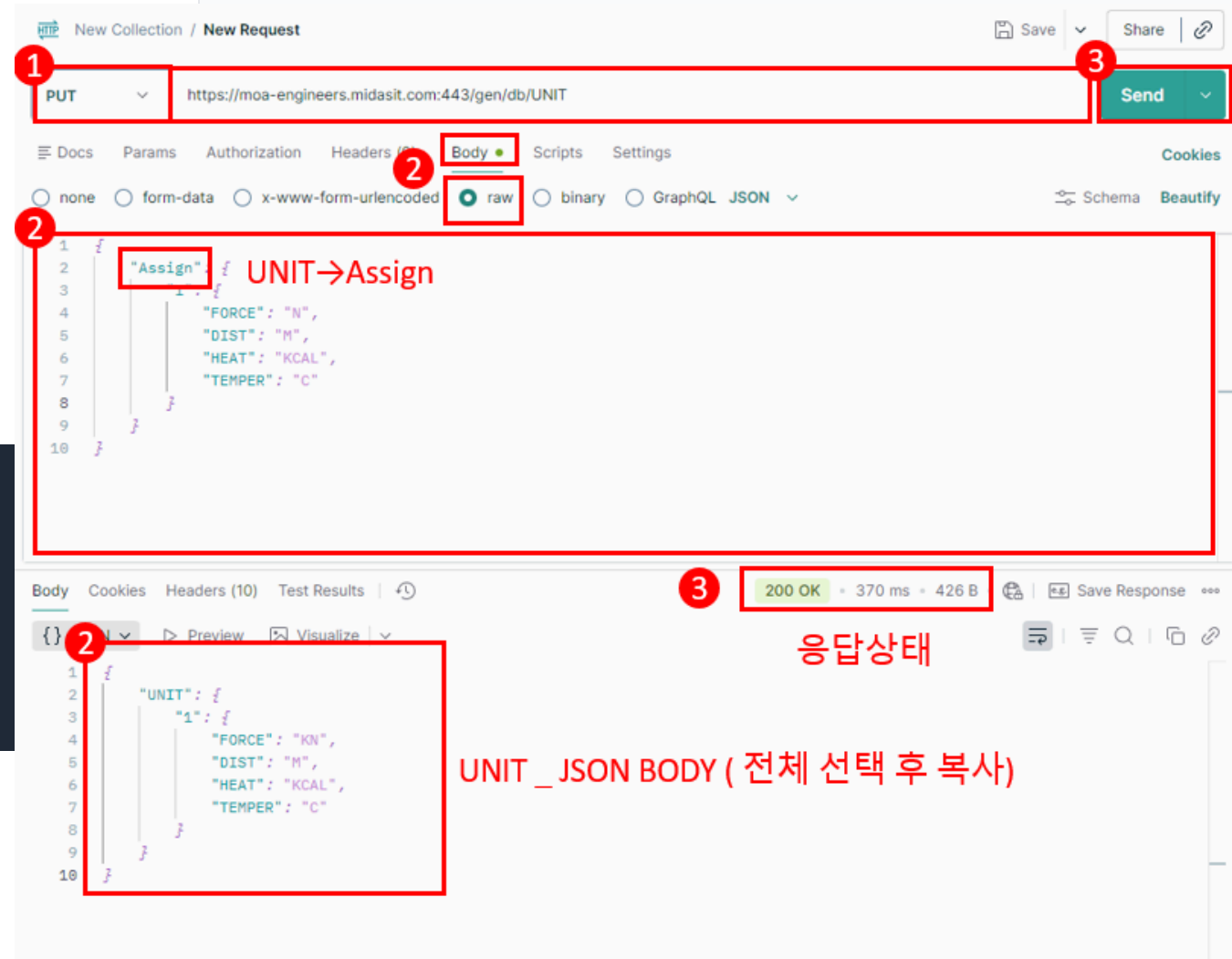
Body 탭 → raw → JSON을 선택합니다.  
기존 조회된 데이터에서 UNIT을 Assign으로 수정하여 입력합니다.

```
{
  "Assign": {
    "1": {
      "FORCE": "kN", "DIST": "M",
      "HEAT": "J", "TEMPER": "C"
    }
  }
}
```

### 3 요청 전송 및 결과 확인

Send 버튼을 클릭합니다.

- Postman 상태 코드 **200 OK** 확인
- 실제 GEN NX 프로그램에서 단위계가 변경되었는지 확인



# Postman 'Environments' 활용



## Environments란?

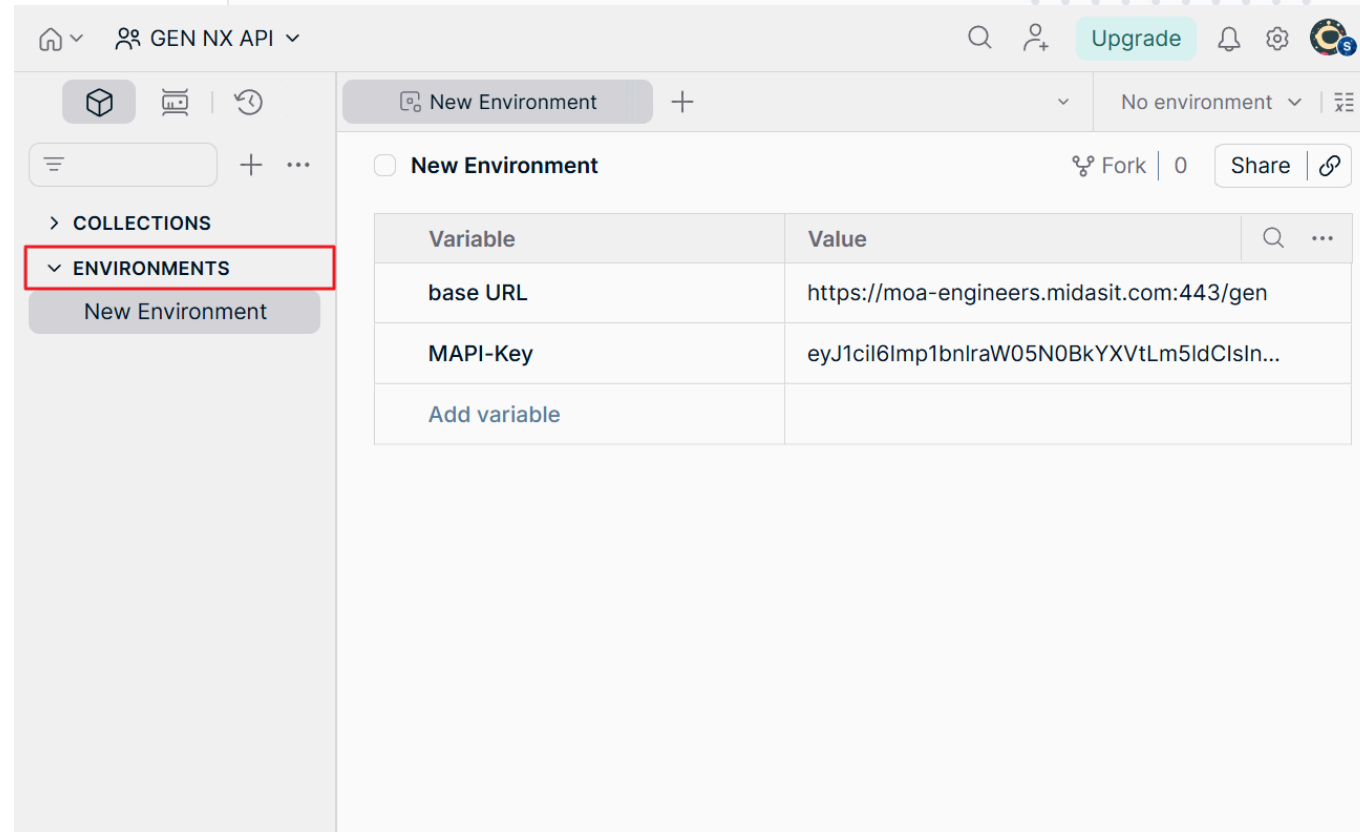
자주 변경되거나 여러 요청에서 공통으로 사용되는 값을 '변수(Variable)'로 저장하고 관리하는 독립적인 공간입니다.



## 핵심 기능

**재사용성** baseURL, MAPI-Key와 같은 값들을 등록하여 손쉽게 재사용

**일괄 적용** 변수 값만 한 번 변경하면, 해당 변수를 사용하는 모든 요청에 일괄적으로 자동 적용되어 효율적입니다.



## 새로운 Environment 생성하기

1

### Environments 생성 시작

좌측 사이드바 메뉴에서 **Environments** 탭을 선택한 후,  
+ **Create** 버튼을 클릭합니다.

2

### 환경 이름 설정

상단 입력창에 환경 이름을 입력합니다. (예: **MIDAS Open API**)

3

### 변수(Variable) 추가

다음 두 개의 핵심 변수를 추가합니다.

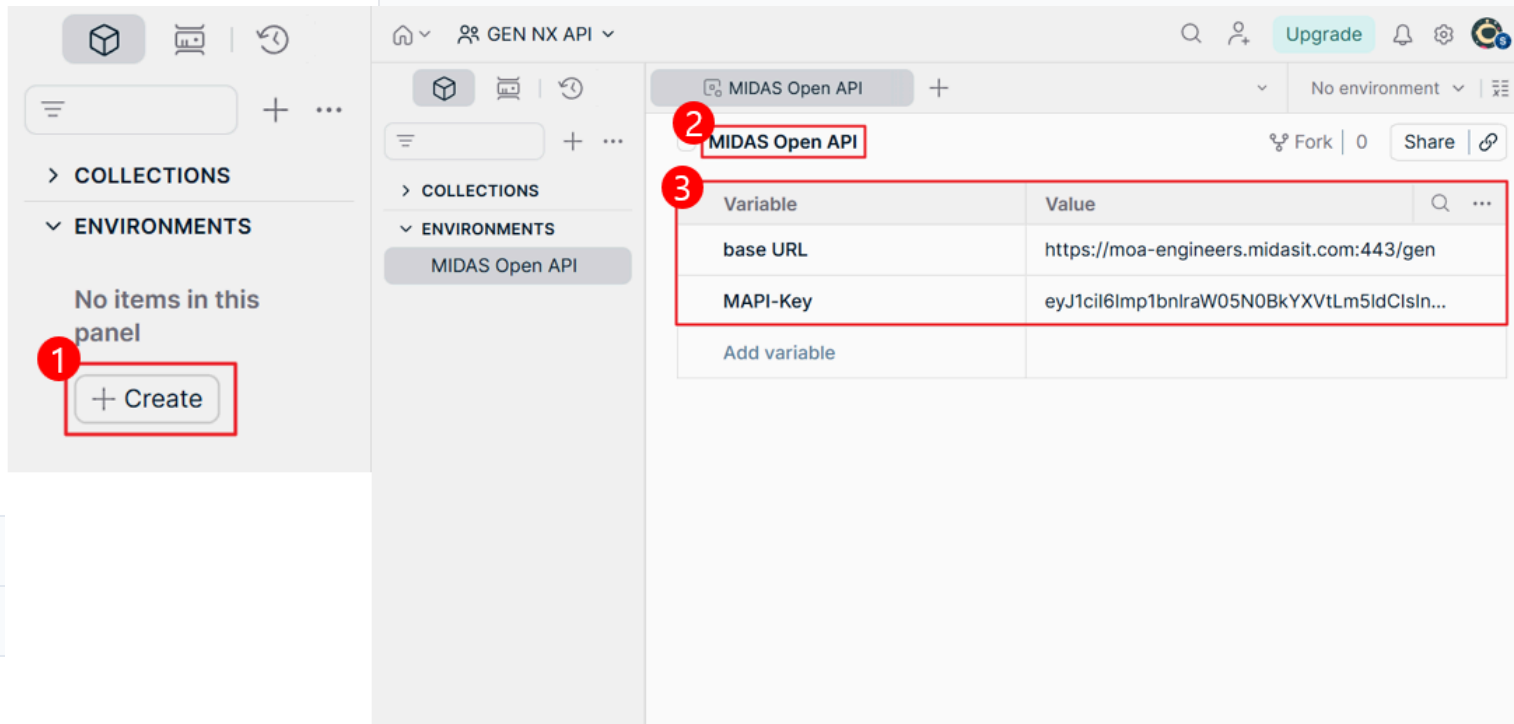
**base URL** `https://moa-engineers.midasit.com:443/gen`

**MAPI-Key** [GEN NX에서 복사한 MAPI-Key 값]

4

### 저장 (Save)

단축키 **Ctrl+S** 를 눌러 환경을 저장합니다.



# Environment 적용 및 활용하기

## 1 환경(Environment) 활성화

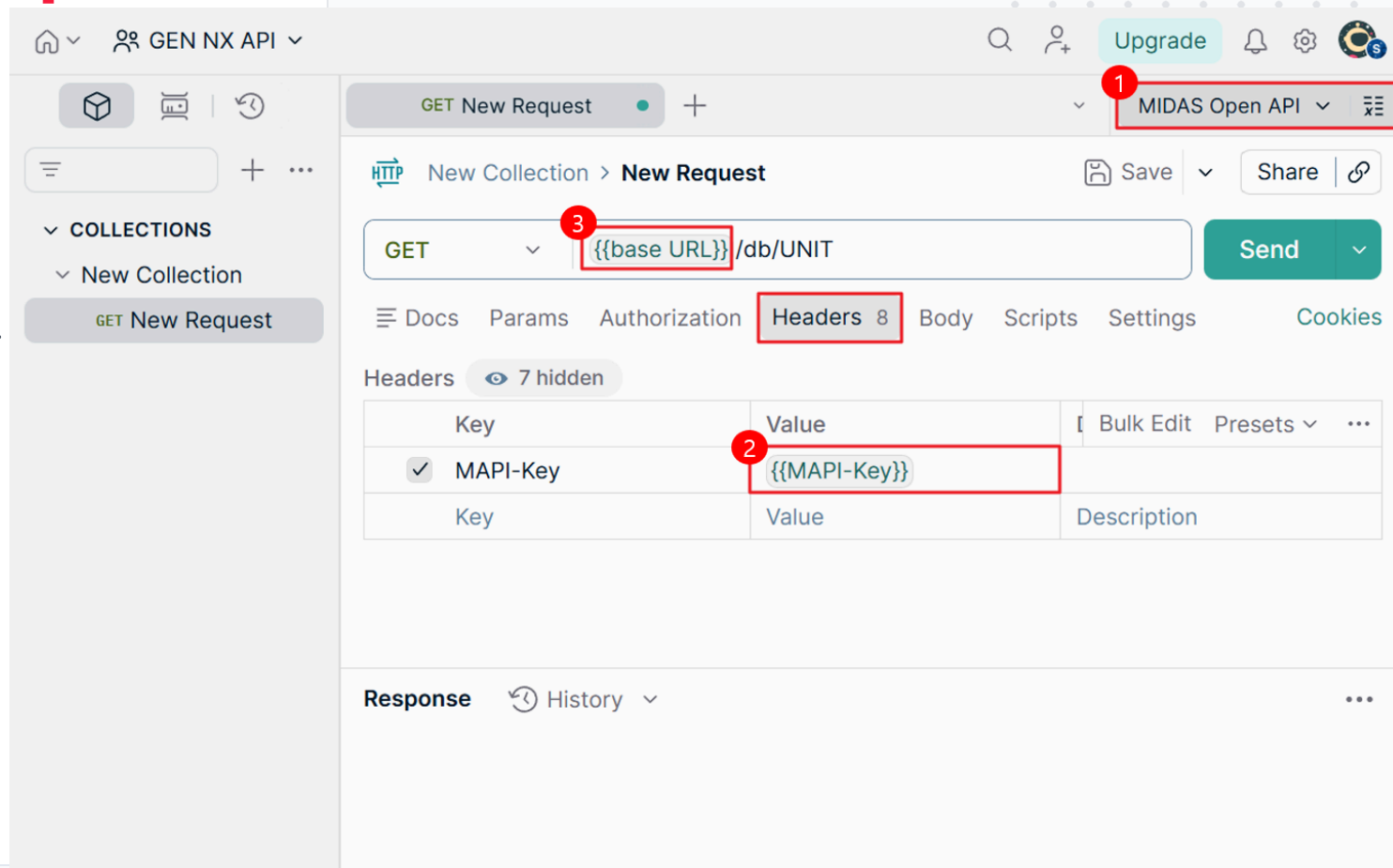
Postman 우측 상단 드롭다운 메뉴에서 방금 생성한 환경 [ MIDAS Open API ]를 선택하여 활성화합니다.

## 2 Headers 탭 설정

Collections 탭에서 Request를 선택 후, **Headers** 탭으로 이동합니다. MAPI-Key의 Value 부분을 `{{MAPI-Key}}` 변수로 수정합니다.

## 3 요청(Request) URL 수정

URL의 고정된 Base URL 부분을 `{{base URL}}` 변수로 대체합니다.  
예: `https://moa-engineers.midasit.com:443/gen/db/UNIT`  
→ `{{base URL}}/db/UNIT`



### ! 주의사항

- ✓ 변수 문법: 변수 이름 앞뒤로 중괄호 2개 `{{ }}` 를 반드시 감싸야 합니다.
- ✓ 환경이 선택되지 않은 상태(No Environment)에서는 변수가 작동하지 않습니다.

## 05. POSTMAN 사용법

# 변수 값 신속하게 수정하기

1

### 아이콘 클릭

Environments 탭으로 매번 이동할 필요 없이, Postman 우측 상단의 아이콘(Variables in request)을 클릭합니다.

2

### 변수 목록 확인

현재 활성화된 환경(Environment)에 저장된 변수 목록이 팝업으로 나타납니다.

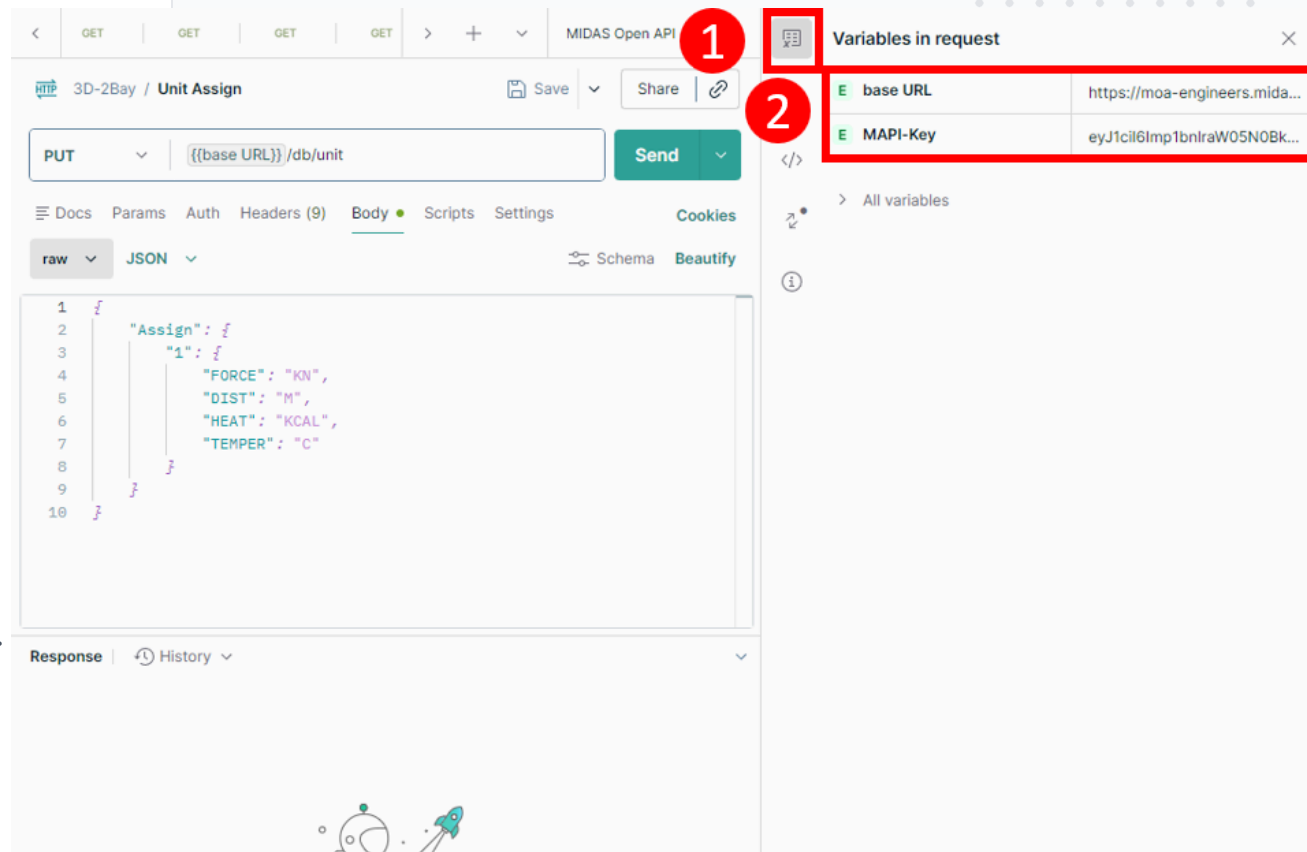
3

### 값 수정 및 저장

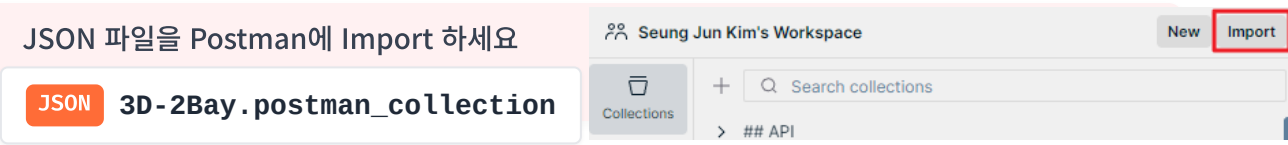
이 창에서 즉시 base URL과 MAPI-Key 값을 수정하고 저장할 수 있습니다.

#### ⚡ Tip: 신속한 대응 가능

연결 오류 발생 시, 이 기능을 활용해 변수 값을 빠르게 점검하고 수정하세요.



# 3D-2Bay 모델링 및 해석 종합 실습



|                                |                                            |                                           |
|--------------------------------|--------------------------------------------|-------------------------------------------|
| 1 doc/SAVEAS<br>프로젝트 저장        | 8 ope/DIVIDEELEM<br>요소 분할 (Divide)         | 15 db/CNLD<br>Nodal Load                  |
| 2 db/UNIT<br>단위계 설정            | 9 db/FRLS<br>Beam End Release              | 16 db/BMLD<br>Beam Load                   |
| 3 db/STYP<br>Structure Type 설정 | 10 db/CONS<br>구속 조건 (Supports)             | 17 doc/ANAL<br>Perform Analysis           |
| 4 db/MATL<br>재료(Material) 정의   | 11 db/STLD<br>정적 하중 케이스 (Static Load Case) | 18 db/LCOM-STEEL<br>하중 조합 (Steel Design)  |
| 5 db/SECT<br>단면(Section) 정의    | 12 db/BODF<br>Self Weight                  | 19 view/ANGLE<br>뷰 각도 조정                  |
| 6 db/NODE<br>절점(Node) 생성       | 13 db/FBLD<br>Define Floor Load Type       | 20 view/RESULTGRAPHIC<br>결과 그래픽 확인        |
| 7 db/ELEM<br>요소(Element) 생성    | 14 db/FBLA<br>Assign Floor Loads           | 21 view/CAPTURE<br>화면 캡처 (Screen Capture) |



doc/SAVEAS

```
{
  "Argument": "C:\\\\Users\\ksj0216\\Desktop\\3D2BAY_API\\3D-2BAY.MGBX"
}
```



db/UNIT

```
{
  "Assign": {
    "1": {
      "FORCE": "KN",
      "DIST": "M",
      "HEAT": "KCAL",
      "TEMPER": "C"
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY

## Save As + Unit

DOC

### SAVEAS

Argument

모델을 저장할 **절대 경로**를 입력합니다.

⚠ 경로의 폴더가 반드시 존재해야 하며, 확장자(.MGBX)를 포함해야 합니다.

DB

### UNIT

1

단위 설정 세트의 **ID**입니다. (일반적으로 1을 사용)

FORCE

힘(Force) 단위 (예: KN, N, KIPS 등)

DIST

길이(Length) 단위 (예: M, CM, MM, IN, FT 등)

HEAT

열량(Heat) 단위 (예: KCAL, BTU, J 등)

TEMPER

온도(Temperature) 단위 (예: C, F)

# JSON BODY

## Structure Type

DB

STYP

```
{
  "Assign": {
    "1": {
      "STYP": 0,
      "MASS": 2,
      "bSELFWEIGHT": true,
      "SMASS": 1,
      "GRAV": 9.806,
      "TEMP": 0,
      "bALIGNBEAM": true,
      "bALIGNSLAB": true
    }
  }
}
```

STYP

구조 타입을 설정합니다. (0: 3-D, 1: X-Z Plane, 2: Y-Z Plane 등)

MASS

Mass Control 타입입니다. (1: Lumped, 2: Consistent)

bSELFWEIGHT

Convert Self-weight into Masses 적용 여부입니다. (true/false)

SMASS

Structure Mass 타입 입니다. (1: Convert to X, Y, Z)

GRAV

중력 가속도 값입니다. (예: 9.806 m/s<sup>2</sup>)

TEMP

초기 기본 온도입니다. (Default: 0)

bALIGNBEAM

Align Top of Beam Section with Center Line 적용 여부입니다.

bALIGNSLAB

Align Top of Slab Section with Center Line 적용 여부입니다.



db/MATL

```
{
  "Assign": {
    "101": {
      "TYPE": "STEEL",
      "NAME": "SS275",
      "HE_SPEC": 0,
      "HE_COND": 0,
      "bMASS_DENS": false,
      "DAMP_RAT": 0.02,
      "PARAM": [
        {
          "P_TYPE": 1,
          "STANDARD": "KS22(S)",
          "DB": "SS275"
        }
      ]
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY

## Material Property

DB

### MATL Assign

**101** 재료 ID 번호입니다.

**TYPE** 재료 유형입니다. (예: "STEEL", "CONC", "USER" 등)

**NAME** 재료의 명칭입니다. (원하는 이름으로 설정)

**DAMP\_RAT** 감쇠비(Damping Ratio)입니다. (0.02)

**bMASS\_DENS** Use Mass Density 적용 여부입니다.  
false: 표준값 사용 / true: 사용자 정의

**PARAM[ ]** 재료의 상세 파라미터 입니다.

#### INSIDE PARAM OBJECT:

**P\_TYPE** Material Type (1: Standard, 2: Isotropic, 3: Orthotropic)

**STANDARD** 표준 규격 이름 (예: "KS22(S)")

**DB** DB 상의 재료 이름 (예: "SS275")



db/SECT

```
{
  "Assign": {
    "101": {
      "SECTTYPE": "DBUSER",
      "SECT_NAME": "H200×200×8/12",
      "SECT_BEFORE": {
        "USE_SHEAR_DEFORM": true,
        "SHAPE": "H",
        "DATATYPE": 1,
        "SECT_I": {
          "DB_NAME": "KS21",
          "SECT_NAME": "H200×200×8/12"
        }
      }
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY Section Property (101)

DB

## SECT (단면 속성)

101

**단면 ID** (Section ID). 각 단면을 식별하는 고유 번호입니다.

SECTTYPE

**DBUSER**: 표준 DB를 기반으로 사용자가 정의한 단면 형식을 의미합니다.

SECT\_NAME

사용자가 지정한 **단면 이름**입니다. (원하는 이름으로 설정)

USE\_SHEAR ..

**true**: 전단 변형(Shear Deformation) 효과를 고려합니다.

SHAPE

**"H"**: H- Section을 정의합니다.

DATATYPE

**1**: DB 기반 정의를 나타냅니다. (2: User Defined)

SECT\_I

실제 DB 정보를 정의하는 객체입니다.

- DB\_NAME: 표준 규격 이름 (예: "KS21")
- SECT\_NAME: 규격 내 단면 호칭 (예: "H200x200x8/12")

```
{
  "Assign": {
    "1": {
      "X": 0,
      "Y": 0,
      "Z": 0
    },
    "2": {
      "X": 5,
      "Y": 0,
      "Z": 0
    },
    "..."
  },
  "12": {
    "X": 10,
    "Y": 6,
    "Z": 3
  }
}
```

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY Node (절점 정의)

### DB NODE

구조물의 기하학적 형상을 정의하는 가장 기본적인 요소인 **절점(Node)**을 생성합니다. 3D 공간 상의 X, Y, Z 좌표를 입력하여 위치를 지정합니다.

**Assign** 데이터 생성을 위한 최상위 키워드입니다. (생성/수정 시 사용)

**"1" ... "12"** 절점 번호(Node ID)입니다. 고유한 정수 값을 가집니다.

**X** 절점의 **X 좌표** (Global Coordinate System 기준)

**Y** 절점의 **Y 좌표** (Global Coordinate System 기준)

**Z** 절점의 **Z 좌표** (Global Coordinate System 기준)

### 💡 Tip

여러 개의 절점을 한 번에 생성할 때, JSON 객체 내에 ID를 키로 하여 연속적으로 데이터를 나열하면 됩니다.



db/ELEM

```
{
  "Assign": {
    "1": {
      "TYPE": "BEAM",
      "MATL": 101,
      "SECT": 101,
      "NODE": [
        1,
        7
      ],
      "ANGLE": 0
    },
    "2": {
      "TYPE": "BEAM",
      "MATL": 101,
      "SECT": 101,
      "NODE": [
        2,
        8
      ],
      "ANGLE": 0
    },
    " ... ": { " ... " },
  },
}
```

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY Element (요소 정의)

DB

ELEM

구조물의 요소(Element)를 생성합니다. 예제에서는 1번부터 6번까지 총 6개의 BEAM 요소가 정의되며, 각 요소는 동일한 재료(101)와 단면(101)을 공유합니다.

1, 2 ...

생성할 요소의 **고유 ID(번호)**입니다. 중복될 수 없습니다.

TYPE

요소의 종류를 지정합니다.

BEAM, TRUSS, PLATE, WALL 등

MATL

적용할 재료(Material)의 ID입니다. (예: 101번 재료)

SECT

적용할 단면(Section)의 ID입니다. (예: 101번 단면)

NODE

요소를 구성하는 **절점 연결 정보**입니다.

BEAM 요소의 경우 **[시작 절점(I), 끝 절점(J)]** 순서로 입력합니다.

ANGLE

요소의 로컬 축 회전 각도(Beta Angle)입니다. (단위: 도)

### Tip: 연결성 확인

NODE 배열에 입력되는 절점 번호(예: 1, 7)는 db/NODE에서 정의된 번호와 반드시 일치해야 합니다. 존재하지 않는 절점을 참조하면 오류가 발생합니다.

ope/DIVIDEELEM

```
{
  "Argument": {
    "TARGETS": [
      101,
      102,
      103,
      104
    ],
    "DIVIDE": {
      "ELEM_TYPE": "Frame",
      "DIV_METHOD": "Equal",
      "OPTION": {
        "EQUAL_OPTION": {
          "NUM_X": 2
        }
      }
    }
  }
}
```

Operation Function Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY Divide Element

OPE

## DIVIDEELEM

기존에 생성된 요소를 분할하는 **Operation** 명령입니다. DB 형식이 아닌 Argument 형식을 사용합니다.

### TARGETS

분할을 수행할 대상 요소의 **ID 목록**입니다.

예: [101, 102, 103, 104] (4개의 요소를 동시에 분할)

### ELEM\_TYPE

분할할 요소의 **유형**을 지정합니다.

"Frame" (보/기동 요소)

### DIV\_METHOD

분할 **방법**을 지정합니다.

"Equal" (등간격 분할)

### NUM\_X

등간격 분할 시 나눌 **개수**입니다.

2 입력 시 요소를 절반으로 나누어 중간에 절점 1개가 생성됩니다.



### POST 메서드 전용

Operation(OPE) 명령은 데이터를 저장하는 것이 아니라 동작을 수행하므로 반드시 **POST** 메서드만 사용해야 합니다.



db/FRLS (Beam End Release)

```
{
  "Assign": {
    "301": {
      "ITEMS": [
        {
          "ID": 1,
          "bVALUE": false,
          "FLAG_I": "0000110",
          "VALUE_I": [
            0, 0, 0, 0, 0, 0, 0
          ],
          "FLAG_J": "0000110",
          "VALUE_J": [
            0, 0, 0, 0, 0, 0, 0
          ]
        }
      ]
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY

## Beam End Release

DB

### FRLS

보 요소 양단의 연결 조건(접합/핀)을 정의합니다. 강접합(Rigid)이 아닌 경우 특정 자유도를 해제(Release)합니다.

**301** 해당 Beam End Release 조건이 적용될 **요소(Element) 번호**입니다. (여러 요소에 적용 시 각각의 키로 정의)

**FLAG\_I** **I단(시점)**의 자유도 해제 여부를 7자리 비트 문자열로 표현합니다.

순서: **Fx Fy Fz Mx My Mz Rw**

"0": 고정(Fixed), "1": 해제(Released)

예시 "0000110": My, Mz 모멘트 해제 (핀 접합)

**FLAG\_J** **J단(종점)**의 자유도 해제 여부 (FLAG\_I와 동일한 규칙)

**VALUE\_I/J** 부분 고정(Partial Fixity) 등 값이 필요한 경우 입력합니다.  
완전 해제(Pinned)인 경우 일반적으로 모두 0을 사용합니다.

#### **i** Pin 접합 (Moment Release)

보의 양단을 핀 접합(단순보)으로 만들려면 My, Mz (5, 6번째 비트)를 "1"로 설정합니다. ("0000110")

```

{
  "Assign": {
    "1": {
      "ITEMS": [
        {
          "CONSTRAINT": "1111111"
        }
      ]
    },
    "2": {
      "ITEMS": [
        {
          "CONSTRAINT": "1111111"
        }
      ]
    }
  }
}
// ... (노드 3~6 생략)
}

```

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY

## Supports (CONS)

### DB CONS (Constraint)

구조물의 지점 조건(Supports)을 정의합니다. 각 자유도(DOF)에 대한 구속 여부를 문자열로 설정합니다.

**Assign Key** 구속 조건을 적용할 **절점 번호(Node ID)**입니다.  
예: "1", "2" 등 절점별로 정의

**ITEMS** 해당 절점에 적용될 구속 조건 항목들의 배열입니다.

**CONSTRAINT** **7자리 문자열 플래그**로 구속 여부를 결정합니다.  
( '1': 구속, '0': 자유 )

플래그 순서 (Dx, Dy, Dz, Rx, Ry, Rz, Rw)

| Dx | Dy | Dz | Rx | Ry | Rz | Rw |
|----|----|----|----|----|----|----|
| 1  | 1  | 1  | 1  | 1  | 1  | 1  |

\* Rw: 워핑(Warping) 자유도 (7번째)

#### i 자주 쓰이는 구속 조건

- "1111111": 고정단 (Fixed)
- "1110000": 힌지 (Pinned)
- "0010000": 롤러 (Roller - Z방향만 구속 시)

```
{
  "Assign": {
    "1": {
      "NO": 1,
      "NAME": "DL",
      "TYPE": "D",
      "DESC": "Floor Dead Load"
    },
    "2": {
      "NO": 2,
      "NAME": "LL",
      "TYPE": "L",
      "DESC": "Floor Live Load"
    },
    "3": {
      "NO": 3,
      "NAME": "WX",
      "TYPE": "W",
      "DESC": "Wind Load X-dir"
    },
    "4": {
      "NO": 4,
      "NAME": "WY",
      "TYPE": "W",
      "DESC": "Wind Load Y-dir"
    }
  }
}
```

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY

## Static Load Case

**DB** STLD (Static Load)

정적 하중 케이스(Static Load Case)를 정의합니다. 하중의 이름, 타입, 설명을 설정하며, 이후 하중 조합(Load Combination)에서 이 이름을 참조하여 사용합니다.

**Assign** 하중 케이스 생성을 위한 최상위 키입니다. 하위 항목으로 각 케이스의 ID("1", "2" 등)를 가집니다.

**NO** 하중 케이스 번호 (ID)  
GEN NX 내부에서 식별하는 고유 번호입니다.

**NAME** 하중 케이스 이름  
예: "DL" (Dead Load), "LL" (Live Load).

**TYPE** 하중 타입 코드

|     |                      |
|-----|----------------------|
| "D" | Dead Load (고정 하중)    |
| "L" | Live Load (활 하중)     |
| "W" | Wind Load (풍 하중)     |
| "E" | Earthquake (지진 하중) 등 |

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY Self Weight

### DB BODF

구조물의 자중(Self Weight)을 하중으로 정의합니다.

**Assign** 데이터를 생성하거나 수정하는 명령어입니다.

**1** 자중 하중 데이터의 고유 ID입니다.

**LCNAME** 하중이 포함될 **Load Case Name**입니다.  
(예: "DL" - Dead Load에 할당)

**FV** 자중 계수 (**Factor Value**) 배열입니다.

[X방향, Y방향, Z방향]

일반적으로 중력 방향(Z축 아래)을 고려하여 [0, 0, -1]로 설정합니다.

#### 참고 사항

**LCNAME**에 지정된 하중 케이스("DL")가 db/STLD 엔드포인트를 통해 미리 정의되어 있어야 합니다.

```
{
  "Assign": {
    "1": {
      "LCNAME": "DL",
      "FV": [
        0,
        0,
        -1
      ]
    }
  }
}
```



db/FBLD

```
{
  "Assign": {
    "1": {
      "NAME": "Office Room",
      "DESC": "2F Floor",
      "ITEM": [
        {
          "LCNAME": "DL",
          "FLOOR_LOAD": -5,
          "OPT_SUB_BEAM_WEIGHT": true
        },
        {
          "LCNAME": "LL",
          "FLOOR_LOAD": -2.5,
          "OPT_SUB_BEAM_WEIGHT": false
        }
      ]
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY

## Define Floor Load Type

### DB FBLD (Define Floor Load Type)

건축물 구조 해석 시 자주 사용되는 바닥 하중(Floor Load)의 **유형(Type)**을 정의하는 엔드포인트입니다. 여기서 정의한 하중 타입은 추후 영역(Area)에 할당(Assign)할 때 사용됩니다.

**1** 바닥 하중 타입의 ID입니다.

**NAME** 하중 타입의 **이름**입니다. (예: "Office Room", "Roof")

**ITEM** 이 타입에 포함될 하중 항목들의 배열(Array)입니다. 하나의 타입에 여러 하중 케이스(Dead Load, Live Load 등)를 묶어서 정의할 수 있습니다.

**LCNAME** 하중이 적용될 **Load Case 이름**입니다. (db/STLD에서 정의한 이름과 일치해야 함)

**FLOOR\_LOAD** 단위 면적당 하중 크기입니다. 중력 방향일 경우 **음수(-)** 값을 입력합니다. (예: -5 KN/m<sup>2</sup>)

**OPT\_SUB\_BEAM\_WEIGHT** Sub Beam의 자중을 포함할지 여부를 설정합니다. (true/false)



db/FBLA

```
{
  "Assign": {
    "1": {
      "FLOOR_LOAD_TYPE_NAME": "Office Room",
      "FLOOR_DIST_TYPE": 1,
      "SUB_BEAM_NUM": 0,
      "SUB_BEAM_ANGLE": 0,
      "UNIT_SELF_WEIGHT": 0,
      "DIR": "GZ",
      "OPT_PROJECTION": false,
      "DESC": "",
      "NODES": [
        7,
        9,
        12,
        10
      ]
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY

## Assign Floor Loads

DB

### FBLA (Floor Load Assign)

**1** 할당된 바닥 하중 객체의 ID입니다.

**FLOOR\_LOAD\_TYPE\_NAME** db/FBLD에서 정의한 **Floor Load Type**의 이름입니다.  
(예: "Office Room")

**FLOOR\_DIST\_TYPE** 하중 분배 방식입니다.

1 : One Way (1방향)

2 : Two Way (2방향)

**DIR** 하중 작용 방향입니다.  
(예: "GZ" - Global Z축 방향)

**NODES** 하중이 작용할 바닥 영역을 정의하는 **절점 ID 배열**입니다.

❗ 순차적으로 폐곡선을 그리도록 입력해야 합니다.



db/CNLD

```
{
  "Assign": {
    "7": {
      "ITEMS": [
        {
          "LCNAME": "WX",
          "GROUP_NAME": "",
          "FX": 100,
          "FY": 0,
          "FZ": 0,
          "MX": 0,
          "MY": 0,
          "MZ": 0
        }
      ]
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY Nodal Load

DB

CNLD

**Assign**

절점 하중을 할당하기 위한 최상위 키입니다.

**"7"**

하중을 적용할 대상 **절점 번호**입니다. (여기서는 7번 절점)

**LCNAME**

적용할 **하중 케이스 이름**입니다.

[db/STLD](#)에서 정의한 하중 케이스 이름(예: "WX")과 일치해야 합니다.

**FX, FY, FZ**

각 방향별 **하중(Force) 크기**입니다.

예시에서는 FX 방향으로 100KN의 하중이 재하됩니다.

**MX, MY, MZ**

각 축에 대한 **모멘트(Moment) 크기**입니다.

**GROUP\_NAME**

하중 그룹의 이름입니다. (비워둘 경우 기본값 사용)



Tip

여러 절점에 동일한 하중을 적용하려면, Assign 객체 내에 절점 번호를 키로 추가하여 반복 작성하면 됩니다.



db/BMLD (Beam Load)

```
{
  "Assign": {
    "1": {
      "ITEMS": [
        {
          "LCNAME": "WY",
          "CMD": "BEAM",
          "TYPE": "UNILOAD",
          "DIRECTION": "GY",
          "D": [0, 1, 0, 0],
          "P": [10, 10, 0, 0]
        }
      ]
    },
    "2": {
      "ITEMS": [
        {
          "LCNAME": "WY",
          "CMD": "BEAM",
          "TYPE": "UNILOAD",
          "DIRECTION": "GY",
          "P": [10, 10, 0, 0]
        }
      ]
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY: Beam Load

Beam

DB

BMLD

보(Beam) 요소에 하중을 할당합니다.

Assign Key

하중을 할당할 **요소 번호(Element ID)**를 최상위 키로 사용합니다. (예: "1", "2")

LCNAME

하중이 포함될 **하중 케이스 이름** (Load Case Name)입니다. (예: "WY")

CMD

하중 적용 대상입니다. ("BEAM" or "LINE" or "TYPICAL")

TYPE

하중 유형입니다.

UNILOAD : 등분포 하중

CONLOAD : 집중 하중

DIRECTION

하중 작용 방향입니다.

GY : 전체 좌표계 Y방향

GZ : 전체 좌표계 Z방향 (중력방향)

D

하중 작용 위치 거리 비율 [x1, x2, x3, x4] (0~1 사이 값)입니다. [0, 1, 0, 0]은 요소 전체 길이에 작용함을 의미합니다.

P

하중 크기 [w1, w2, w3, w4]입니다.

[10, 10, 0, 0]은 시점 10kN/m, 종점 10kN/m의 등분포 하중을 의미합니다.

# JSON BODY

## Perform Analysis

DOC

ANAL

### 해석 수행 (Perform Analysis)

모델링이 완료된 후 구조 해석을 수행하기 위한 명령입니다. 별도의 인자 (Argument)나 할당(Assign) 데이터 없이 빈 JSON 객체 `{}`를 전송하면 GEN NX가 해석을 시작합니다.

#### 💡 핵심 포인트

- **Empty Body:** Body에 내용을 채우지 않고 `{}`만 전송합니다.
- **Method:** **POST** 메서드를 사용합니다.
- **전제 조건:** 모델링 데이터(재료, 단면, 노드, 요소, 하중, 경계조건 등)가 모두 입력되어 있어야 정상적으로 해석이 수행됩니다.
- **해석 시간:** 해석 시간이 길어질 수 있으므로 응답 대기 시간에 유의해야 합니다.



db/LCOM-STEEL

```
{
  "Assign": {
    "1": {
      "NO": 1,
      "NAME": "LCB1",
      "ACTIVE": "STRENGTH",
      "iTYPE": 0,
      "DESC": "1.2D+1.6L",
      "vCOMB": [
        {
          "ANAL": "ST",
          "LCNAME": "DL",
          "FACTOR": 1.2
        },
        {
          "ANAL": "ST",
          "LCNAME": "LL",
          "FACTOR": 1.6
        }
      ]
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY

## Load Combination

DB

### LCOM-STEEL

**NAME** Combination Name (하중 조합 이름)

예: "LCB1", "LCB2"

**ACTIVE** Active Type (활성 타입)

• STRENGTH, SERVICE, INACTIVE (비활성)

**iTYPE** Summation Method (합산 방식)

• 0: Add, 1: Envelope, 3: SRSS

**DESC** Description (설명)

예: "1.2D+1.6L"

**vCOMB** Combination List (개별 하중 케이스 목록)

**ANAL** : Analysis Type (ST: Static, CS: Construction Stage 등)

**LCNAME** : Load Case Name (적용할 하중 케이스 이름)

**FACTOR** : Factor (적용할 하중 계수)

#### **i** Note

Load Combination은 여러 하중 케이스(DL, LL 등)에 설계 기준에 따른 계수(Factor)를 곱하여 합산하는 단계입니다.

# JSON BODY

## View Angle Setting

### VIEW ANGLE

모델 뷰의 시점(View Point) 각도를 설정합니다. 3D 모델을 원하는 방향에서 바라보도록 조정하여 결과 확인이나 이미지 캡처 시 활용됩니다.

- HORIZONTAL** 수평 회전 각도 (Horizontal Angle)  
기준 평면(XY)을 중심으로 모델을 좌우로 회전시킵니다. (단위: 도 Degree)
- VERTICAL** 수직 회전 각도 (Vertical Angle)  
수평면에서 위/아래로 모델을 회전시킵니다. (단위: 도 Degree)

#### 활용 팁

이 설정은 **view/CAPTURE** 명령을 사용하기 전에 모델의 뷰를 적절한 위치로 조정하는 데 필수적입니다. ISO 뷰와 유사한 시점을 얻으려면 일반적으로 Horizontal 30~45도, Vertical 20~30도 정도를 사용합니다.

```
{  
  "Argument": {  
    "HORIZONTAL": 30,  
    "VERTICAL": 25  
  }  
}
```



view/RESULTGRAPHIC

```
{
  "Argument": {
    "CURRENT_MODE": "ReactionForces/Moments",
    "LOAD_CASE_COMB": {
      "TYPE": "CBS",
      "MINMAX": "Max",
      "NAME": "LCB1"
    },
    "COMPONENTS": {
      "COMP": "FZ",
      "OPT_LOCAL_CHECK": false
    },
    "TYPE_OF_DISPLAY": {
      "VALUES": {
        "OPT_CHECK": true
      },
      "LEGEND": {
        "OPT_CHECK": true,
        "POSITION": "right"
      },
      "ARROW_SCALE_FACTOR": 1.0
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY Result Graphic

VIEW

## RESULTGRAPHIC

해석 결과를 시각화하여 확인하기 위한 설정을 정의합니다. 반력, 변위, 부재력 등 다양한 결과를 화면에 표시할 수 있습니다.

### CURRENT\_MODE

표시할 결과의 종류를 설정합니다.

(예: ReactionForces/Moments, Displacement)

### LOAD\_CASE

확인할 하중 조건 또는 하중 조합을 지정합니다.

└ **TYPE**: 하중 타입 (CBS: 조합, ST: 정적)

└ **NAME**: 하중 조건/조합의 이름 (예: LCB1)

### COMPONENTS

확인할 성분을 지정합니다.

└ **COMP**: 성분명 (예: FZ - 수직 반력)

### TYPE\_OF\_DISPLAY

화면 표시 옵션 (TYPE\_OF\_DISPLAY)

└ **VALUES**: 수치값 표시 여부 (OPT\_CHECK)

└ **LEGEND**: 범례 표시 및 위치 설정



view/CAPTURE

```
{
  "Argument": {
    "SET_MODE": "post",
    "SET_HIDDEN": true,
    "EXPORT_PATH": "C:\\Users\\...\\Reaction.jpg",
    "HEIGHT": 768,
    "WIDTH": 1280,
    "ACTIVE": { "ACTIVE_MODE": "All" },
    "ANGLE": {
      "HORIZONTAL": 30,
      "VERTICAL": 25
    },
    "RESULT_GRAPHIC": {
      "CURRENT_MODE": "ReactionForces/Moments",
      "LOAD_CASE_COMB": {
        "TYPE": "CBS",
        "NAME": "LCB1"
      },
      "COMPONENTS": { "COMP": "FZ" }
    }
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY Screen Capture

VIEW

CAPTURE

**SET\_MODE** 캡처 모드를 설정합니다. ("pre": 전처리, "post": 후처리)

**SET\_HIDDEN** Hidden 적용 여부입니다. (Hidden 적용 시 true)

**EXPORT\_PATH** 이미지가 저장될 **절대 경로**를 입력합니다.

예: "C:\\Users\\...\\Reaction.jpg"

**WIDTH/HEIGHT** 저장할 이미지의 해상도(너비/높이)를 픽셀 단위로 설정합니다.

**ANGLE** 캡처 시점의 뷰 각도를 설정합니다. (HORIZONTAL / VERTICAL)

**RESULT\_GRAPHIC** 캡처할 결과 항목을 상세 정의합니다.

✓ 결과 모드, 하중 조합, 성분 등 설정 가능

### i API 활용 팁

반복적인 보고서 작성 업무 시, **view/CAPTURE**를 루프(Loop)로 실행하여 수십 장의 결과 이미지를 자동으로 저장하고 엑셀/워드에 붙여넣을 수 있습니다.

```
POST post/TABLE

{
  "Argument": {
    "TABLE_NAME": "Reaction(Global)",
    "TABLE_TYPE": "REACTIONG",
    "EXPORT_PATH":
      "C:\\Users\\ksj0216\\Desktop\\3D2BAY_API\\Reaction.JSON",
    "UNIT": {
      "FORCE": "kN",
      "DIST": "m"
    },
    "STYLES": {
      "FORMAT": "Fixed",
      "PLACE": 6
    },
    "COMPONENTS": [
      "Node", "Load", "FX", "FY", "FZ",
      "MX", "MY", "MZ", "Mb"
    ],
    "LOAD_CASE_NAMES": [
      "DL(ST)"
    ]
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY / Result Table Export

### POST TABLE

- TABLE\_NAME** 결과 테이블의 이름을 지정합니다. (예: Reaction(Global))
- TABLE\_TYPE** Export할 결과 테이블 종류. (REACTIONG = 전체좌표계 반력)
- EXPORT\_PATH** 📁 결과 JSON 파일이 저장될 절대 경로.
- UNIT** 결과에 사용될 단위계 (FORCE / DIST). 예: kN, m
- STYLES** 숫자 표현 방식. FORMAT(Fixed/Exp), PLACE(소수점 자릿수).
- COMPONENTS** Export할 성분 목록 (Node, Load, FX~MZ, Mb 등).
- LOAD\_CASE\_NAMES** Export할 하중조합/하중케이스 이름 배열. (예: DL(ST))

#### 💡 API 활용 팁

post/TABLE을 활용하면 해석 결과를 JSON 파일로 즉시 저장할 수 있어, 후속 리포트 자동화나 데이터 분석에 바로 활용할 수 있습니다.

```
POST DESIGN/STEEL/KDS-41-30-2022/CODE-ANAL

{
  "Argument": {
    "PERFORM_TYPE": "ALL"
  }
}
```

JSON Body Structure

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY Steel Code Check

### POST CODE-ANAL

#### PERFORM\_TYPE

강재 설계 검토를 수행할 대상 범위 지정.  
"ALL"은 모든 강재 부재에 대해 KDS 41 30 2022 코드체크를 수행함. 다른 옵션은 매뉴얼 참고.

#### 📌 추가 설명

- 적용 코드: KDS 41 30 2022 (강구조 설계기준)
- 실행 조건: 사전에 doc/ANAL로 정적/해석이 완료되어 있어야 함
- 결과 활용: CODE-REPORT API로 검토 리포트 이미지 생성 가능

#### 💡 API 활용 팁

코드체크를 API로 자동화하면 다수의 모델 파일이나 반복적인 설계 검토 업무를 스크립트 한 번으로 처리할 수 있습니다.

## 06. POSTMAN 3D-2BAY 실습

# JSON BODY / Steel Code Report

### POST CODE-REPORT

- REPORT\_TYPE** 리포트 유형. "**MEMB**" = 부재별(Member) 리포트.
- CURRENT\_MODE** 리포트 표현 모드. "**Graphic**"은 그래픽 리포트로 출력.
- EXPORT\_PATH** 📁 리포트 이미지가 저장될 절대 경로(폴더).
- OUTPUT\_NAME** 저장될 파일명. 확장자 포함 (예: STEELCODEREPORT.jpg).
- ELEMS.KEYS** 리포트 생성 대상 부재(요소) ID 배열. 예: [1,2,3,4,5].

#### 💡 API 활용 팁

CODE-ANAL 실행 후 CODE-REPORT를 이어서 호출하면 강재 부재별 검토 결과 이미지를 자동 저장할 수 있어, 설계 보고서 작성 시간을 대폭 단축할 수 있습니다.

JSON Body Structure

# Python | API 활용 확장

## CONCEPT

단순한 API 요청을 넘어 **로직과 자동화**를 결합하는 단계입니다.

파이썬의 강력한 생태계와 GEN NX API를 연동하여 **엔지니어링 업무 효율**을 극대화합니다.

## 👍 주요 장점



### 함수화 & 모듈화

자주 사용하는 API 호출 패턴을 함수(Function)로 만들어 레고 블록처럼 재사용합니다. 코드의 중복을 줄이고 유지보수성을 높입니다.



### 반복 작업 자동화

수백 개의 Load Case나 모델을 반복문(Loop) 하나로 일괄 처리하고 실행합니다. 단순 반복 업무에서 해방되어 핵심 설계에 집중할 수 있습니다.



### 데이터 가공 & 분석

API 응답(JSON)을 받아 엑셀 저장, 그래프 시각화 등 즉시 리포팅으로 연결합니다. Pandas, Matplotlib 등 강력한 라이브러리를 활용할 수 있습니다.



### 안정적 파이프라인

조건문(If)과 예외 처리(Try)로 오류 상황에도 멈추지 않는 워크플로우를 구축합니다. 복잡한 설계 시나리오도 안정적으로 자동화할 수 있습니다.



GOOGLE COLAB 실습 환경 접속

<https://colab.research.google.com>

