5주차

~~**기존 주제(NoSQL DB 사용해보기)**~~

→ 개발이 처음이신 분들두..있고.. 주제가 DB 심화인데 갑자기 백엔드 개발하는 것도 웃기고..해서..! 그리고 컨퍼 준비하면서 생각난 것도 있고.. 다른 개발하면서 느낀 것도 있는데 **API 사용은 데엔의 기본 of 기본**! 한 번도 안 해보신 분들을 위해 준비했슴다!

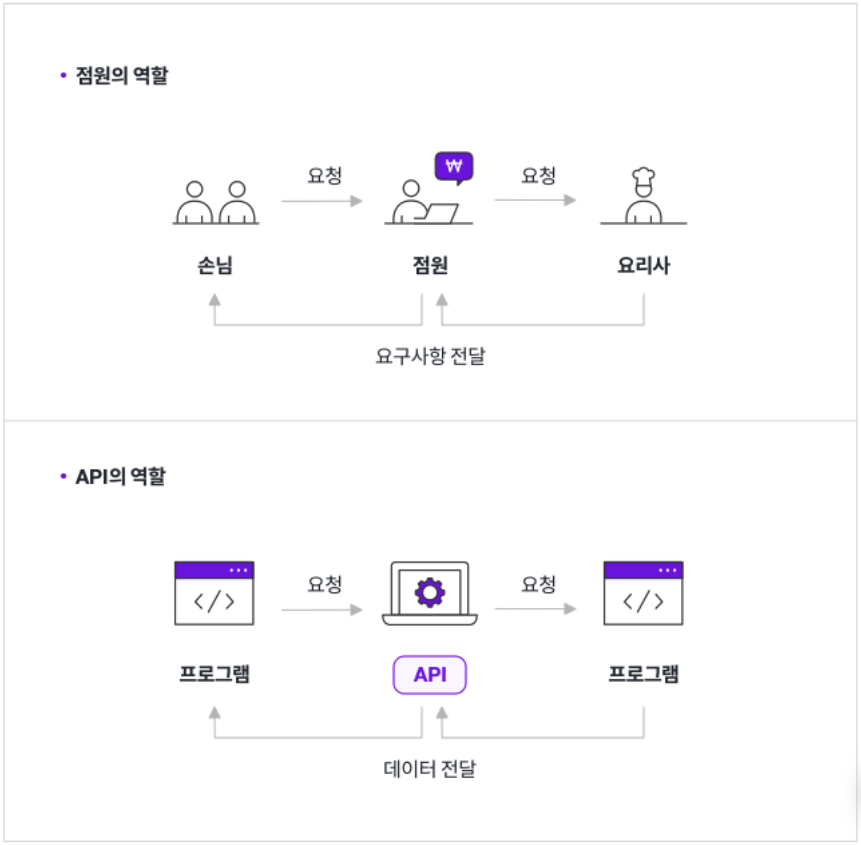
완소! API 사용법

API란?

Application **P**rogramming **I**nterface

: 이종 시스템 간에 응용 프로그래밍이 가능하도록 정의된 명세!

- 프로그램들이 서로 상호작용하는 것을 도와주는 매개체



출처) https://blog.wishket.com/api란-쉽게-설명-그린클라이언트/

API의 역할

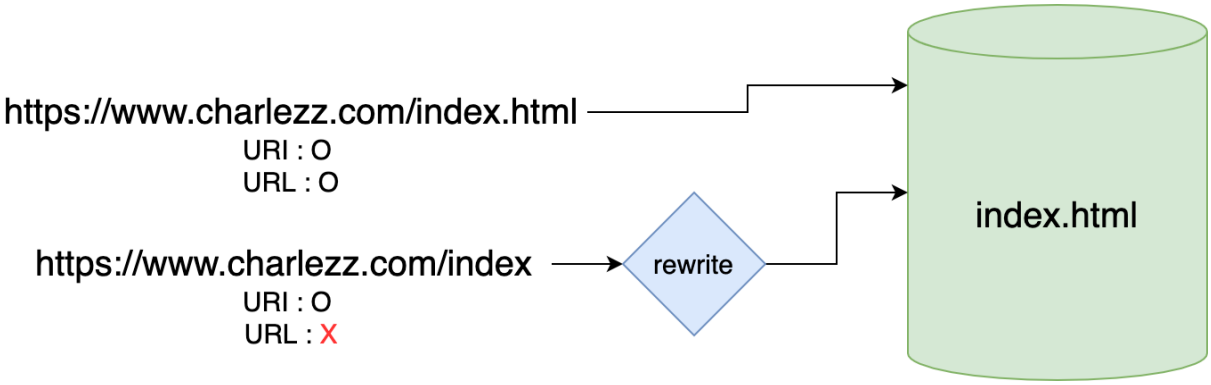
1. 서버와 DB 사이의 출입구
2. 애플리케이션과 기기 사이의 통신 수단
3. 모든 접속 표준화
    - 기계, 운영체제에 관계 없이 동일한 엑세스

URL

URL vs URI

Uniform Resource Locator: 정형화된 자원 위치(location)

Uniform Resource Identifier: 정형화된 자원 식별자(identifier) - URL은 URI에 포함된 개념



```
scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]
```

1. scheme : 사용할 프로토콜을 뜻하며 웹에서는 http 또는 https를 사용
2. user와 password : (서버에 있는) 데이터에 접근하기 위한 사용자의 이름과 비밀번호
3. host와 port : 접근할 대상(서버)의 호스트명과 포트번호
4. path : 접근할 대상(서버)의 경로에 대한 상세 정보
5. query : 접근할 대상에 전달하는 추가적인 정보 (파라미터)
6. fragment : 메인 리소스 내에 존재하는 서브 리소스에 접근할 때 이를 식별하기 위한 정보

`https://www.google.com:443/search?q=boaz&oq=boaz&sourceid=chrome&ie=UTF-8`

Protocol

컴퓨터 네트워크 환경 내 모든 기기가 서로 통신하기 위해 설정한 통신 규약

`FTP`, `SFTP`, `STP`, 등등...

`https`://www.google.com:443/search?q=boaz&oq=boaz&sourceid=chrome&ie=UTF-8

HTTP + Secure(보안) = HTTPS

Port

https://www.google.com:`443`/search?q=boaz&oq=boaz&sourceid=chrome&ie=UTF-8

- 고유 서비스별로 할당된 숫자
- 표준 포트가 지정되어 있음(생략 가능)
  - HTTP: 80
  - HTTPS: 443
  - FTP: 21
  - SSH: 22
  - MySQL: 3306
  - 그 외(사용되지 않는 포트의 경우): 지정 시 사용 가능함
  - 그러나 한 포트는 한 가지의 일만 수행할 수 있으므로 절대 겹쳐선 안 됨
    - 그리고 위와 같은 경우로 선점되어 있는 포트는 절대 사용하지 않는 것이 신상에 이로움

Host

https://`www.google.com`:443/search?q=boaz&oq=boaz&sourceid=chrome&ie=UTF-8

https://8.8.8.8:443/search?q=boaz&oq=boaz&sourceid=chrome&ie=UTF-8

- 도메인
  - ex) 건물 주소 (서울시 강남구 테헤란로~)
- IP
  - ex) 건물의 위/경도 좌표

Path

https://www.google.com:443`/search`?q=boaz&oq=boaz&sourceid=chrome&ie=UTF-8

Querystring

https://www.google.com:443/search`?q=boaz&oq=boaz&sourceid=chrome&ie=UTF-8`

- key=value 형식으로 되어 구분자&사용의 형태를 띰
  - key1=val1&key2=val2&…
- string안에 특수문자 사용시 URL Encoding/Decoding을 해야함
  - ex) 띄어쓰기의 경우 그냥 띄면 안 되고, %20이 중간에 들어가야함
    - ?club=boaz bigdata `X`
    - ?club=boaz%20bigdata `O`

JSON

JavaScript Object Notation

- 데이터를 저장하거나 전송할 때 많이 사용되는 경량의 DATA 교환 형식
- Javascript에서 객체를 만들 때 사용하는 표현식을 의미한다
- 데이터 포맷을 지칭하는 말. 통신 방법이나 프로그래밍 문법이 아님
- 비정형 Document-oriented/Graph/Key-value 데이터 포맷으로 xml과 함께 NoSQL을 표현할 때 사용함

```
{
  "students": [
    {
      "name": "Nayeon",
      "lastName": "Keum"
    },
    {
      "name": "Kyuyeon",
      "lastName": "Park"
    },
    {
      "name": "Someone",
      "lastName": "else"
    }
  ]
}
```

RESTful API

- REST 아키텍처의 제약 조건을 준수하는 애플리케이션 프로그래밍 인터페이스
- 규칙을 올바르게 지켜야 ful이 붙을 수 있음

REST

- `RE`presentational `S`tate `T`ransfer

1. HTTP URI를 통해 자원을 명시하고
2. HTTP Method(POST, GET, PUT, DELETE)를 통해
3. 해당 자원에 대한 CRUD Operation을 적용하는 것

> Create : 데이터 생성(POST)
> Read : 데이터 조회(GET)
> Update : 데이터 수정(PUT)
> Delete : 데이터 삭제(DELETE)

REST의 특징

1. Server-Client(서버-클라이언트 구조)
2. Stateless(무상태): 각각의 독립적인 프로세스

3. Cacheable(캐시 처리 가능)
4. Layered System(계층화): Path를 이용한
5. Uniform Interface(인터페이스 일관성)

REST의 장단점

장점

- HTTP 프로토콜의 인프라를 그대로 사용하므로 REST API 사용을 위한 별도의 인프라를 구출할 필요가 없다.
- HTTP 프로토콜의 표준을 최대한 활용하여 여러 추가적인 장점을 함께 가져갈 수 있게 해 준다.
- HTTP 표준 프로토콜에 따르는 모든 플랫폼에서 사용이 가능하다.
- REST API 메시지가 의도하는 바를 명확하게 나타내므로 의도하는 바를 쉽게 파악할 수 있다.
- 여러 가지 서비스 디자인에서 생길 수 있는 문제를 최소화한다.
- 서버와 클라이언트의 역할을 명확하게 분리한다.

단점

- 표준이 자체가 존재하지 않아 정의가 필요하다.
- 사용할 수 있는 메소드가 4가지밖에 없다.
- HTTP Method 형태가 제한적이다.
- 브라우저를 통해 테스트할 일이 많은 서비스라면 쉽게 고칠 수 있는 URL보다 Header 정보의 값을 처리해야 하므로 전문성이 요구된다.
- 구형 브라우저에서 호환이 되지 않아 지원해주지 못하는 동작이 많다.(익스플로어)

REST API: REST의 원리를 따르는 API

ex)

[GET]

```
curl -XGET https://boaz-web.com/member
```

결과)

```
{
  "members": [
    {
      "name": "Nayeon",
      "lastName": "Keum"
    },
    {
      "name": "Kyuyeon",
      "lastName": "Park"
    },
    {
      "name": "Someone",
      "lastName": "else"
    },
    ...
  ]
}
```

[POST]

```
curl -X POST https://boaz-web.com/member/add
   -H "Content-Type: application/json"
   -d '{"name":"Wooseok", "lastName":"Song"}'
```

결과)

커맨드에 보여지지는 않지만, member DB에 새로운 값이 더해진 형태가 될 것

```
{
  "members": [
    {
      "name": "Nayeon",
      "lastName": "Keum"
    },
    {
      "name": "Kyuyeon",
      "lastName": "Park"
    },
    {
      "name": "Someone",
      "lastName": "else"
    },
    {
      "name": "Wooseok",
      "lastName": "Song"
    },
    ...
  ]
}
```

URI 기반 vs RequestBody 기반

URI 기반: QueryString 기반

```
curl -XGET https://boaz-web.com/member/_search?q=name:Nayeon
```

RequestBody 기반: JSON 기반(Query DSL)

```
curl -XPOST -H 'Content-Type: application/json' https://boaz-web.com/member/_search?pretty -d '
{
  "query": {
    "query_string": {
      "default_field": "name",
      "query": "Nayeon"
    }
  }
}
```

참고 링크

API 실습

Open API 사용해보기

공공데이터 포털의 **기상청_단기예보 ((구)_동네예보) 조회서비스** Open API를 이용해 데이터를 조회해봅시다~

기상청_단기예보 ((구)_동네예보) 조회서비스

초단기실황, 초단기예보, 단기((구)동네)예보, 예보버전 정보를 조회하는 서비스입니다. 초단기실황정보는 예보 구역에 대한 대표 AWS 관측값을, 초단기예보는 예보시점부터 6시간까지의 예보를, 단기예보는 예보기간을 글피까지 확장 및 예보단위를 상세화(3시간→1시간)하여 시공간적으로 세분화한 예보를 제공합니다.

https://www.data.go.kr/tcs/dss/selectApiDataDetailView.do?publicDataPk=15084084

준비

1. 회원가입
2. 위 링크로 접속해 활용신청
3. 마이페이지 > 오픈API > 개발계정 API키 받기

어떻게 사용하지????

어떻게 사용하는지는 문서에 잘 나와있습니다

▼ 참고문서에 첨부된 파일을 다운받아 문서를 읽어봅시다~

> https://s3-us-west-2.amazonaws.com/secure.notion-static.com/b37e4431-02a4-454b-818c-5f627940473e/%EA%B8%B0%EC%83%81%EC%B2%AD41_%EB%8B%A8%EA%B8%B0%EC%98%88%EB%B3%B4_%EC%A1%B0%ED%9A%8C%EC%84%9C%EB%B9%84%EC%8A%A4_%EC%98%A4%ED%94%88API%ED%99%9C%EC%9A%A9%EA%B0%80%EC%9D%B4%EB%93%9C_%EC%B5%9C%EC%A2%85.docx

1. 목차
    a. 서비스 목록 확인
2. 사용하고자 하는 서비스가 명세된 페이지로 이동해서 자세한 내용을 확인!
    a. 서비스 인증 / 권한 - ServiceKey
    b. 교환데이터 표준 - XML/JSON
    c. 인터페이스 표준 - REST(GET)
    d. 메세지 교환 유형 - Request / Response
3. Call Back URL 확인
4. 각 파라미터에 대한 설명 확인
5. 브라우저에서 요청 / 응답 메세지 예시로 요청해보기

```
https://apis.data.go.kr/1360000/VilageFcstInfoService_2.0/getUltraSrtNcst?serviceKey=서비스키&numOfRows=10&pageNo=1&base_date=202208
```

▼ 혹시 안되면 서비스키 이걸로

76cGX96noaoXB18CrPFiyznVmnDcuGg%2FpDSLXmpccd1pkfUqhNsLajAz3ivPThtWgtOgPtmV2jsdBP9M9UNEww%3

▼ default XML형태로 요청

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
<response>
  <header>
    <resultCode>00</resultCode>
    <resultMsg>NORMAL_SERVICE</resultMsg>
  </header>
  <body>
    <dataType>XML</dataType>
    <items>
      <item>
        <baseDate>20220817</baseDate>
        <baseTime>0600</baseTime>
        <category>PTY</category>
        <nx>55</nx>
        <ny>127</ny>
        <obsrValue>0</obsrValue>
      </item>
      <item>
        <baseDate>20220817</baseDate>
        <baseTime>0600</baseTime>
```


▼ &dataType=JSON 붙여서 JSON형태로 요청


```
{"response":{"header":{"resultCode":"00","resultMsg":"NORMAL_SERVICE"},"body":{"dataType":"JSON","items":{"item":[{"baseDate":"20220817","baseTime":"0600","category":"PTY","nx":55,"ny":127,"obsrValue":"0"},
{"baseDate":"20220817","baseTime":"0600","category":"REH","nx":55,"ny":127,"obsrValue":"91"},{"baseDate":"20220817","baseTime":"0600","category":"RN1","nx":55,"ny":127,"obsrValue":"0"},{"baseDate":"20220817","baseTime":"0600","category":"T1H","nx":55,"ny":127,"obsrValue":"23"},
{"baseDate":"20220817","baseTime":"0600","category":"UUU","nx":55,"ny":127,"obsrValue":"0"},{"baseDate":"20220817","baseTime":"0600","category":"VEC","nx":55,"ny":127,"obsrValue":"0"},{"baseDate":"20220817","baseTime":"0600","category":"VVV","nx":55,"ny":127,"obsrValue":"0"},
{"baseDate":"20220817","baseTime":"0600","category":"WSD","nx":55,"ny":127,"obsrValue":"0.1"}]},"pageNo":1,"numOfRows":10,"totalCount":8}}
```


6. 파이썬에서 예시 코드로 요청해보기

```
import requests

url = 'http://apis.data.go.kr/1360000/VilageFcstInfoService_2.0/getUltraSrtNcst'
params ={'serviceKey' : 서비스키, 'pageNo' : '1', 'numOfRows' : '1000', 'dataType' : 'JSON', 'base_date' : '20220817', 'base_time' :

response = requests.get(url, params=params)
text = response.content.decode('utf-8')

print(text)
```

▼ XML

▼ JSON

간단한 REST API 실습

Flask와 Postman 이용해서 간단한 REST API 실습을 해봅시다!

Flask

웹 애플리케이션 개발을 위한 파이썬 프레임워크
최소한의 구성 요소와 요구 사항을 제공하기 때문에 시작하기 쉽고 필요에 따라 유연하게 사용할 수 있음

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run(debug=True)
```

default host: localhost

default port: 5000

debug=True → 코드 변경 시 변화를 인식하고 다시 실행

Route

URL을 방문 할 때 준비된 함수가 트리거되도록 바인딩 하기 위해 `route()` 데코레이터 사용

동적 URL

바인딩 할 URL을 지정할 때 `<변수>`을 명시하는 게 가능하다. 그러면 이 변수는 함수에 인수 형태로 전달이 된다.
`<int:user_id>` 변수 타입 지정도 가능

```
@app.route('/user/<user_name>/<int:user_id>')
def user(user_name, user_id):
    return f'Hello, {user_name}({user_id})!'
```

실습

준비!

1. postman 설치
2. week_5 브랜치만 깃 클론 해오고

```
git clone -b week_5 --single-branch https://github.com/BOAZ-bigdata/19Engineering_BASE
```

```
cd MentoringB
```

▼ 3. 도커 이미지 빌드 및 컨테이너 실행을 합시다

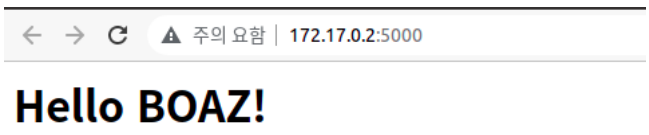
```
:~/workspace/boaz-engi-mm/05_db$ docker build -t flask-image .
```

`docker images`

```
REPOSITORY          TAG         IMAGE ID       CREATED              SIZE
flask-image         latest      5aa83f05b6ba   About a minute ago   925MB
```

```
~/workspace/boaz-engi-mm/05_db$ docker run -t -p 5000:5000 --name flask-container --rm flask-image
```

```
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.17.0.2:5000/ (Press CTRL+C to quit)
```



▼ Flask로 구현한 API 서버 app.py를 도커 컨테이너에 띄워 실행시키고, 로컬에서 해당 API서버로 접근합니다.

`docker run` 을 실행할 때 `-p` 옵션을 통해 포트 포워딩을 할 수 있습니다. `-p 1234:8888`은 호스트 기기의 1234번 포트를 도커 컨테이너의 8888번 포트와 연결한다는 의미입니다.

app.py는 5000번 포트로 서버를 열어두는데, 이를 컨테이너 환경에서 실행하기 때문에 컨테이너의 5000번 포트를 연결해야 컨테이너 내의 flask 서버에 접근할 수 있습니다.

```
# build docker image
docker build -t { image_name } .
```

```
# run docker container
docker run -t -p 8888:5000 --name { container_name } --rm { image_name }
```

간단한 REST API 코드 살피기

▼ home

```
from flask import Flask, jsonify, request
import datetime as dt

app = Flask(__name__)


@app.route("/")
def home():
    return "<h1>Hello BOAZ!</h1>"

@app.route("/<name>")
def hello(name):
    now = dt.datetime.now()
    return jsonify(
        date=now,
```

```
        text=f'Hi, {name}!',
    )

...

if __name__ == "__main__":
    # host default: localhost(127.0.0.1)
    # port default: 5000
    app.run(host="0.0.0.0", port="5000", debug=True)
```

- `jsonify` 함수를 이용해 json형태로 리턴 가능
- flask app을 실행할 때 호스트 IP를 `"0.0.0.0"`으로 명시해야 합니다.
  이는 동일한 네트워크 상에서 localhost가 아닌 서버의 IP로 접속을 가능하게 해줍니다.
- 도커 컨테이너 내에서의 localhost는 컨테이너 내에서만 접근 가능한 로컬호스트로, 호스트 기기의 localhost와는 원칙적으로 구분되어 있습니다!

▼ 전체 멤버 조회

```
# GET /members
@app.route("/members")
def get_members():
    return jsonify(members)
```

default가 `GET` 이기 때문에 methods를 따로 지정해주지 않아도 됩니다.

`GET` 방식을 이용해 데이터를 조회하는 request를 보냅니다.

▼ 새 멤버 추가

```
# POST /members
@app.route("/members", methods=['POST'])
def add_member():
    request_data = request.get_json()  # {"unit": ..., "part": ..., "name": ...}
    new = {
        "id": members[-1]['id'] + 1,
        "unit": request_data['unit'],
        "part": request_data['part'],
        "name": request_data['name'],
    }
    members.append(new)
    return jsonify(members)
```

`POST` 방식을 이용하여 데이터를 추가하는 request 보냅니다.

▼ 특정 id에 해당하는 멤버 정보 수정　　*id는 고유한 값입니다!

```
# PUT /members/<int:id>
@app.route("/members/<int:id>", methods=['PUT'])
def edit_member(id):
    request_data = request.get_json()
    new = {
        "id": int(id),
        "unit": request_data['unit'],
        "part": request_data['part'],
        "name": request_data['name'],
    }
    for i in range(len(members)):
        if members[i]['id'] == int(id):
            members[i] = new
    return jsonify(new)
```

`PUT` 방식을 이용해 데이터를 수정하는 request를 보냅니다.

▼ 특정 id에 해당하는 멤버 삭제

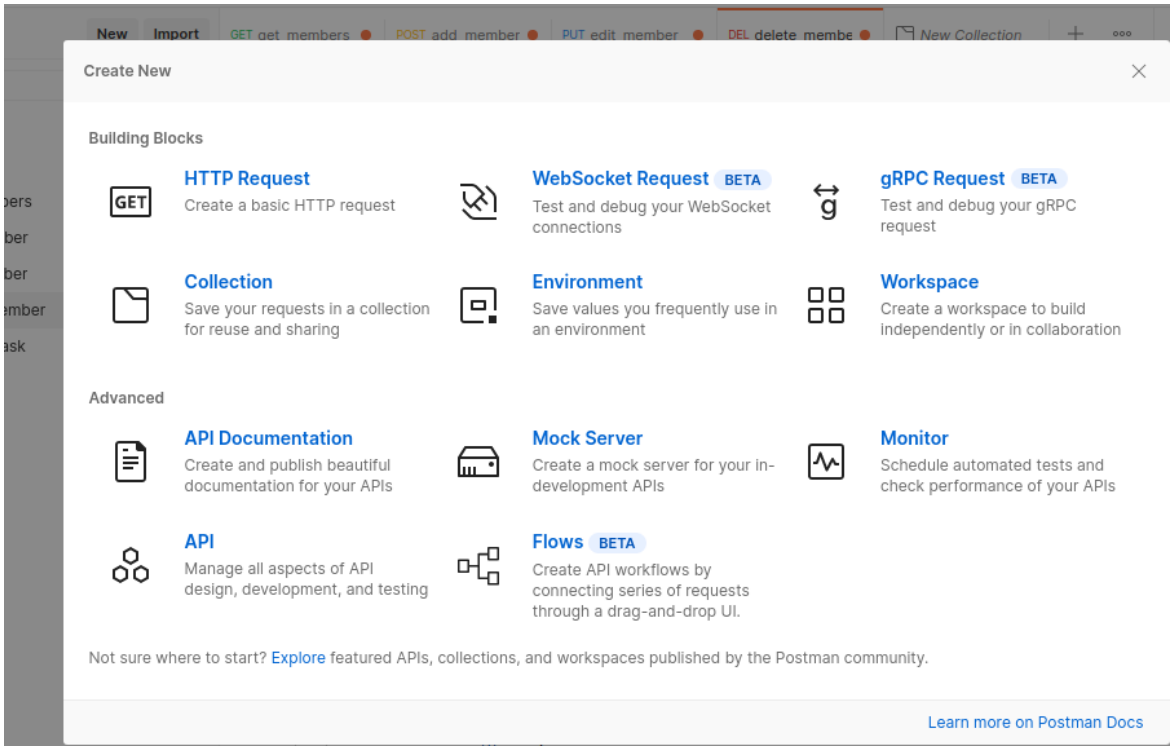
```
# DELETE /members/<int:id>
@app.route("/members/<int:id>", methods=['DELETE'])
def delete_member(id):
    for i in range(len(members)):
        if members[i]['id'] == int(id):
            members.remove(members[i])
    return jsonify(members)
```

`DELETE` 방식을 이용해 데이터를 삭제하는 request를 보냅니다.

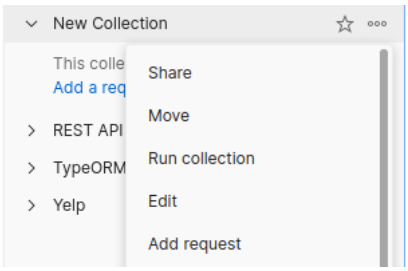
Postman

API를 테스트해볼 수 있는 플랫폼

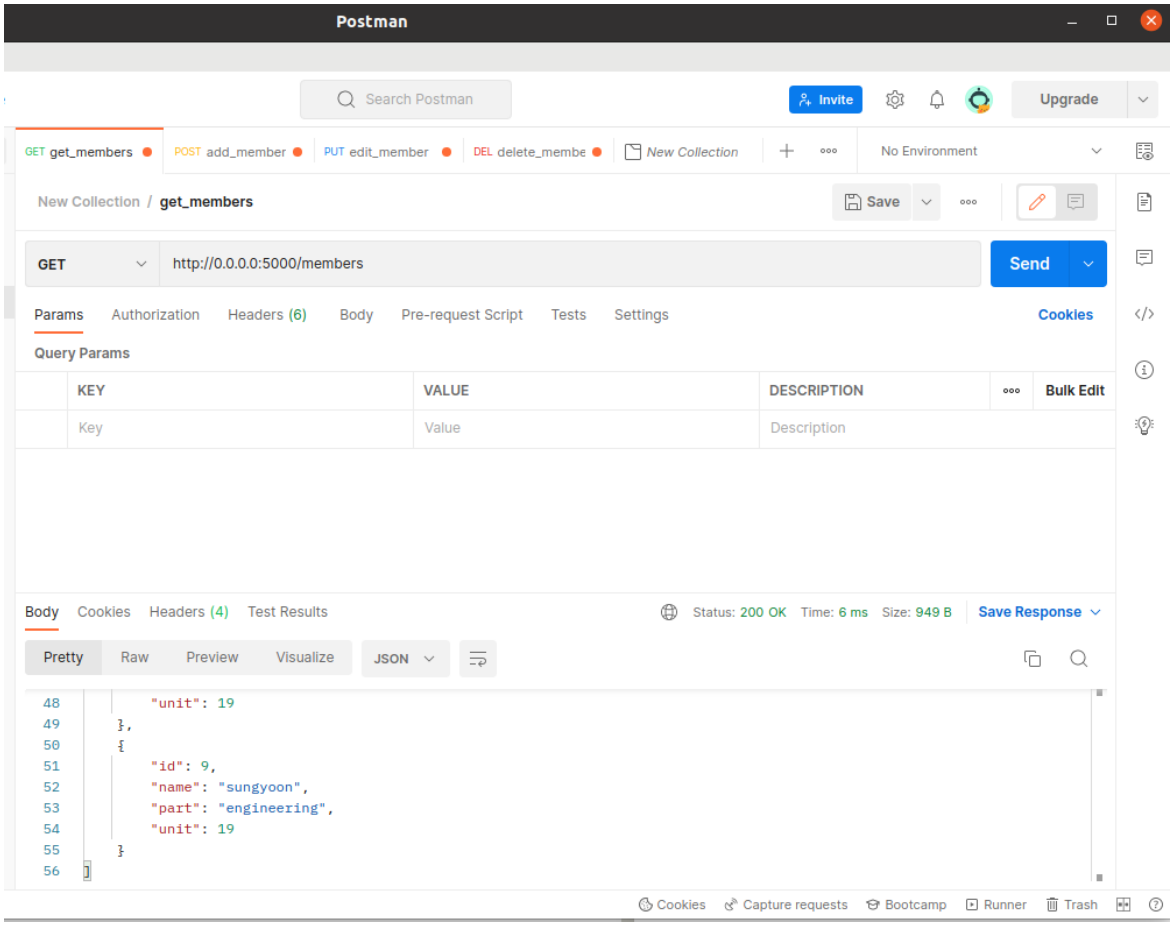
▼ new > new collection



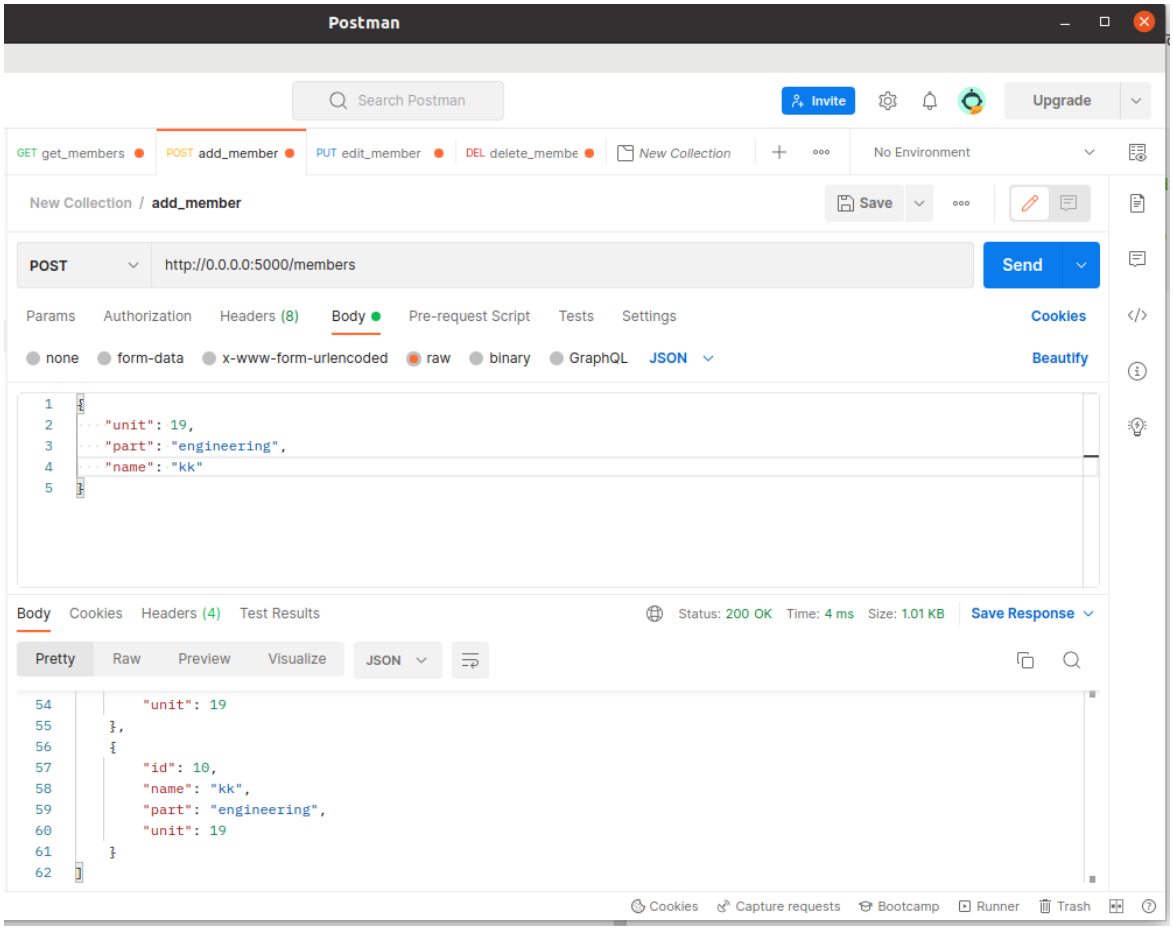
▼ add request



▼ GET

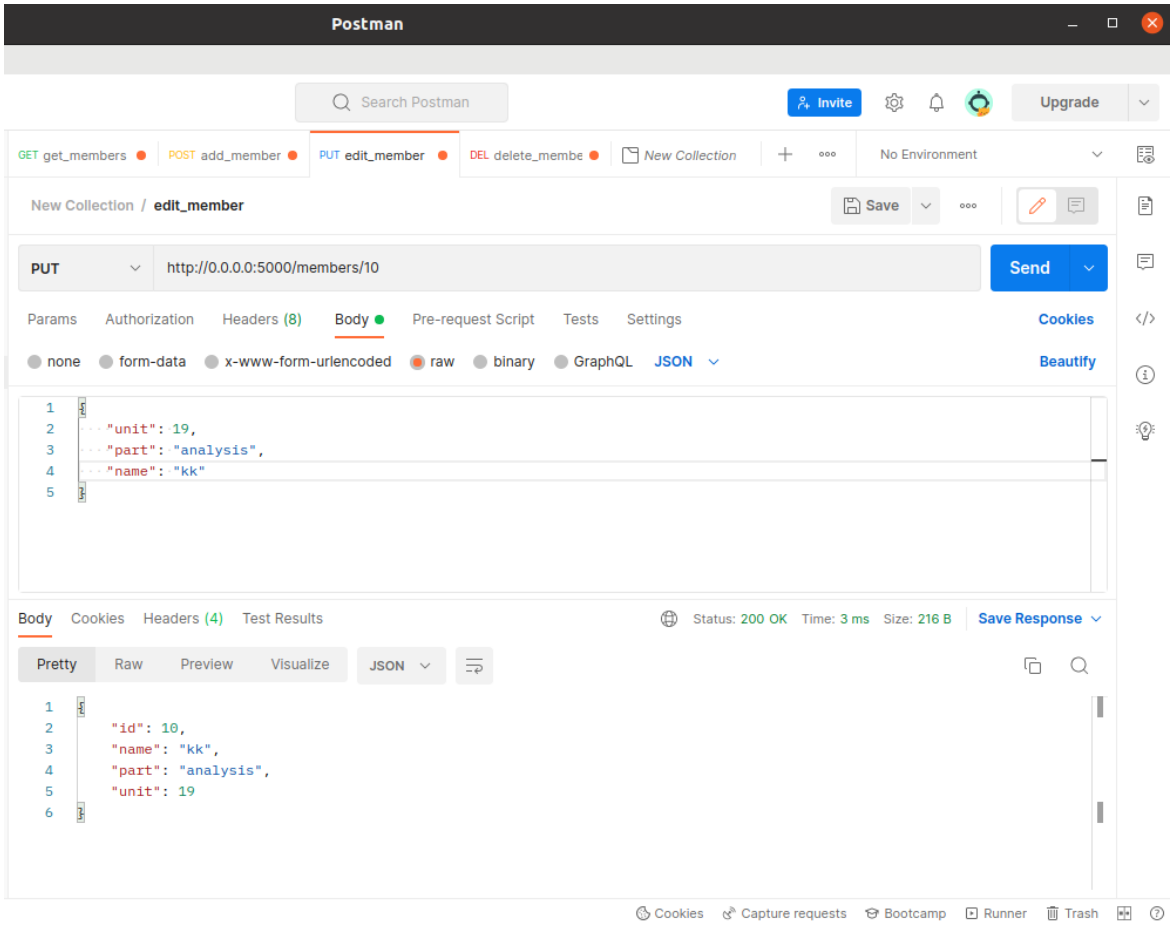

Postman
Search Postman
Invite
Upgrade
GET get_members
POST add_member
PUT edit_member
DEL delete_membe
New Collection
No Environment
New Collection / get_members
Save
GET
http://0.0.0.0:5000/members
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
KEY
VALUE
DESCRIPTION
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 6 ms
Size: 949 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
"unit": 19
},
{
"id": 9,
"name": "sungyoon",
"part": "engineering",
"unit": 19
}
]
Cookies
Capture requests
Bootcamp
Runner
Trash

▼ POST

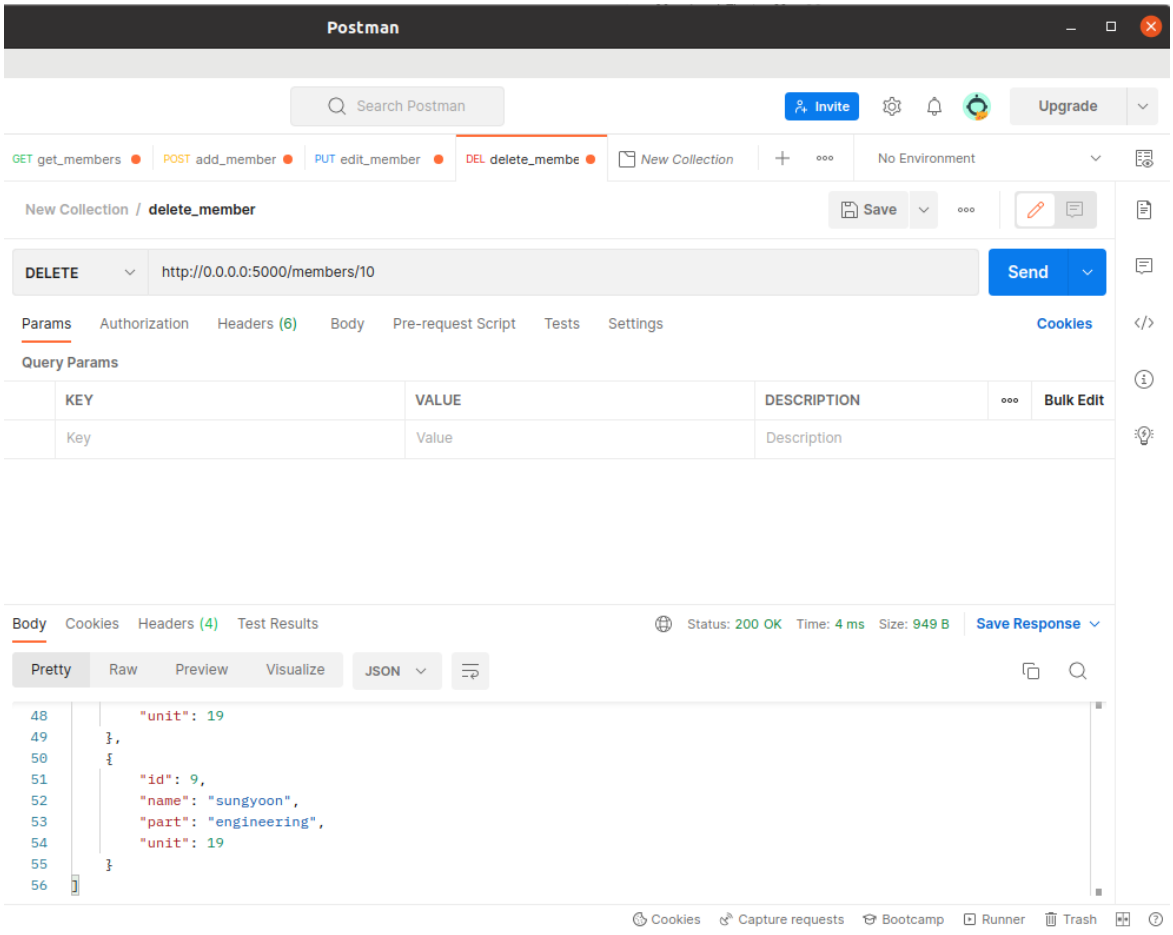



```
# Body에 작성한 data(request)를 json 형태로 받아옴
request_data = request.get_json()
```

▼ PUT


Postman
Search Postman
Invite
Upgrade
GET get_members
POST add_member
PUT edit_member
DEL delete_membe
New Collection
No Environment
New Collection / edit_member
Save
PUT
http://0.0.0.0:5000/members/10
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1 {
2 "unit": 19,
3 "part": "analysis",
4 "name": "kk"
5 }
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 3 ms
Size: 216 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "id": 10,
3 "name": "kk",
4 "part": "analysis",
5 "unit": 19
6 }
Cookies
Capture requests
Bootcamp
Runner
Trash

▼ DELETE



근데 새로고침하면 원상복구되죠?

DB 연결을 해야합니다...(열린결말)

참고 https://velog.io/@rhee519/docker-flask-hosting