HTTP 메서드

- HTTP API를 만들어보자
- HTTP 메서드 - GET, POST
- HTTP 메서드 - PUT, PATCH, DELETE
- HTTP 메서드의 속성

HTTP API를 만들어보자

요구사항

⇒ 회원 정보 관리 API

- 회원 목록 조회
- 회원 조회
- 회원 등록
- 회원 수정
- 회원 삭제

API URI 설계

User Resource Identifier

- 회원 목록 조회
  ⇒ /read-member-list
- 회원 조회
  ⇒ /read-member-by-id
- 회원 등록
  ⇒ /create-member
- 회원 수정
  ⇒ /update-member
- 회원 삭제
  ⇒ /delete-member

이것은 좋은 URI 설계일까?

⇓

가장 중요한 건 "리소스 식별"

API URI 고민

- 리소스의 의미?

  ⇒ "회원"이라는 개념 자체가 리소스!

- 리소스 식별 어떻게??

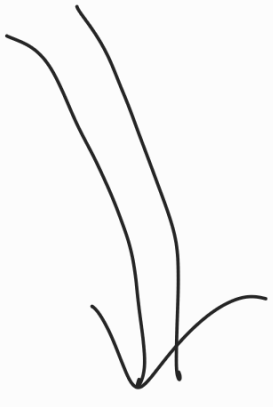
  - 회원을 (등록, 수정, 조회, ⋮) 하는 것은 모두 배제!

  - "회원"이라는 리소스만 식별하면 OK

    ⇒ 회원 리소스를 URI에 맵핑!

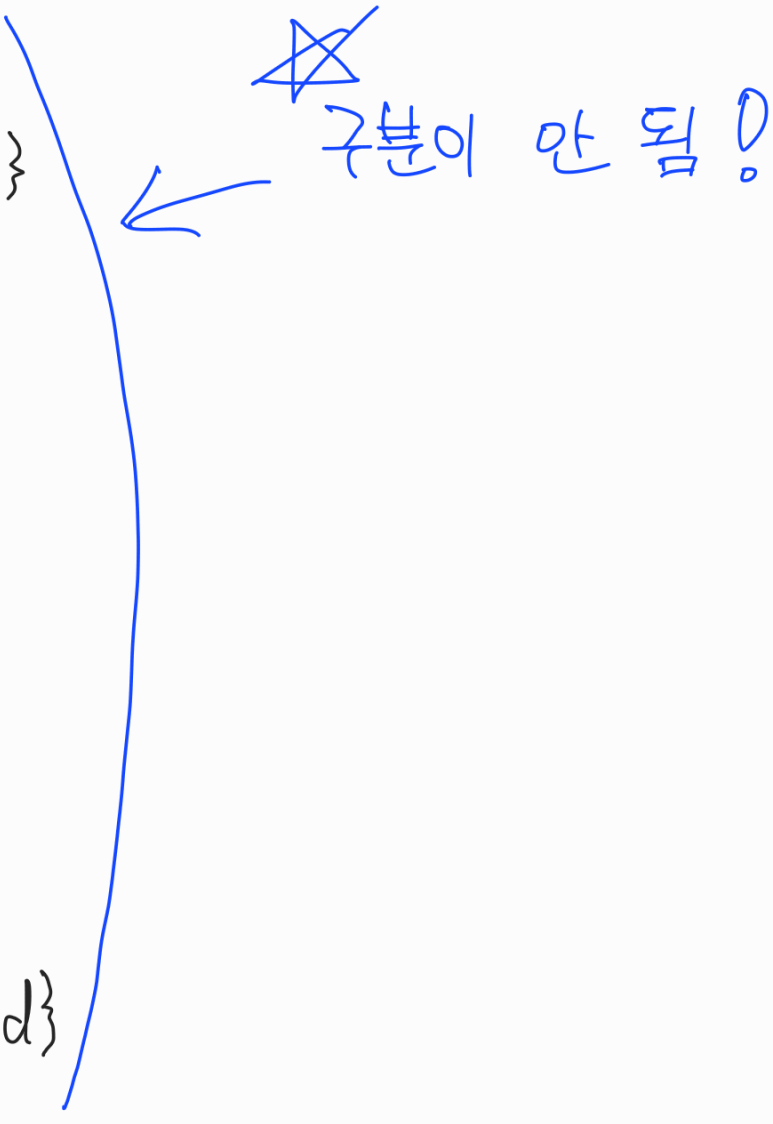
API URI 설계

- 회원 목록 조회
- 회원 조회
- 회원 등록
- 회원 수정
- 회원 삭제



리소스 식별, URI 계층 구조 활용

- 회원 목록 조회
  ⇒ /members
- 회원 조회
  ⇒ /members/{id}
- 회원 등록
  ⇒ /members/{id}
- 회원 수정
  ⇒ /members/{id}
- 회원 삭제
  ⇒ /members/{id}



리소스와 행위를 분리
(결국 "리소스 식별")

- URI는 리소스만 식별!
- 리소스: 회원
  행위: 조회, 등록, 삭제, 변경

HTTP 메서드 종류
(주요 메서드)

최근엔 representation으로 바뀜

- GET
  : 리소스 조회
- POST
  : 요청 데이터 처리, 주로 등록에 사용
- PUT
  : 리소스를 대체. 해당 리소스 없을 시 생성
- PATCH
  : 리소스 부분 변경
- DELETE
  : 리소스 삭제

- HEAD
  : GET과 동일 But 메시지 부분 제외, (상태 줄 / 헤더)만 반환
- OPTIONS ← 주로 CORS에서
  : 대상 리소스에 대한 통신 가능 옵션(메서드)을 설명
- CONNECT
  : 대상 자원으로 식별되는 서버에 대한 터널 설정
- TRACE
  : 대상 리소스 대한 경로 따라 메시지 루프백 테스트 수행

HTTP 메서드 - GET, POST

GET

- 리소스 조회
- 서버에 전달하려는 data는
  query 통해 전달
  (쿼리 패러미터, 쿼리 스트링)
- messag body 사용해서 data 전달 가능

But, 지원 X인 곳 많음 ⇒ 권장 X

ex)

```
GET /search?q=hello&hl=ko HTTP/1.1
Host: www.google.com
```

GET

리소스 조회 ① - 메시지 전달
② - 서버 도착
③ - 응답 데이터

POST

- 요청 데이터 처리
- message body 통해
  서버로 request data 전달
- 서버는 req. data 를 처리
  ↳ msg. body 통해 들어온 data 처리하는
  모든 기능 수행
- 전달된 data로
  주로 신규 리소스 등록, 프로세스 처리에 사용

리소스 등록 ① - 메시지 전달
② - 신규 리소스 생성 (신규 리소스 식별자 생성)
③ - 응답 데이터
↳ location
(자원이 신규 생성된 URI)

정리

- 새 리소스 생성(등록)
- req. data 처리 ← cf) 컨트롤 URI
  (동사 행위가 URI로)
- 다른 메서드로 처리 애매한 경우
  ex) JSON으로 data 넘기기
  but GET 쓰기 어려울 때

PUT

- 리소스를 대체 (완전히)
  - 리소스 있으면 ⇒ 대체
  - 없으면 ⇒ 생성
  - ⇒ 덮는다.

- client가 resource 식별
  - client가 리소스 위치 알고 URI 식별
  - POST와의 차이점

(i) 리소스가 있는 경우

①

```
{
  "username": "old",
  "age": 50
}
```

```
{
  "username": "young",
  "age": 20
}
```

②

대체됨!

```
{
  "username": "old",
  "age": 50
}
```

✡ 리소스를 완전히 대체한다.

(ii) 리소스 없는 경우

⟹ 신규 리소스 생성

PATCH

- 리소스 부분 변경

DELETE

- 리소스 제거

HTTP 메서드의 속성

- 안전 (Safe)
- 멱등 (Idempotent)
- 캐시 가능

안전 (Safe)

- 호출해도 리소스를 변경 X
  (해당 리소스에 대해서만 안전 이라는 의미)

멱등 (Idempotent)

- 한 번 호출, 두 번 호출, ..., 100번 호출
  ⇒ 다 결과가 똑같다.

- 멱등인 메서드
  - GET
  - PUT
  - DELETE
  - ~~POST~~

- 활용
  - 자동 복구 메커니즘
  - 서버가 정상 req. 못 주었을 때 client가 같은 req. 다시 해도 되는가?
    판단근거

✡ 멱등은

외부 요인으로 중간에 resource 변경되는 건 고려 X

⇓

동일 사용자가 동일 요청을 여러 번

캐시 가능 (Cacheable)

- req. result 리소스를 캐시해서 사용해도 되는가?
- GET, HEAD, POST, PATCH 가능
- 실제론 GET, HEAD 정도만 캐시로 사용

↳ POST, PATCH는 본문 내용까지 cache key로 고려