< HTTP 웹 기본지식 정리 ⇒ 1) HTTP 메서드, 2) 상태코드, 3) 헤더 > ~p.214

1) HTTP 메서드

① HTTP ⇒ 문서간의 링크로 html을 전달해주는 프로토콜로 시작..
↳ HyperText Transfer Protocol
↳ 지금은, 이미지·음성·영상·파일·JSON. XML 등등.. 모든 데이터 전송

② HTTP는 stateless다. 그렇기에 중간에 다른 서버가 투입되어도 / 대거 증설되어도 ..
(어떠한 서버를 추가해도 문제 X

client → server1
server2
server3.

↳ 필요한 정보를 모두 담아, 그때그때 보내기에, 어떤 서버 추가해도 문제X
(상품, 개수, 신용카드).

⇒ 그런데, 로그인 기능같은 경우, 클라이언트 정보를 저장해줘야 하니,
쿠키 / 세션 등으로 상태 유지.. ↳ 다음장 정리.

③ HTTP 메서드 → GET / POST / PUT - PATCH / DELETE …등등.

④ API URI ??
→ URI란? Uniform "Resource" Identifier.
ex) "회원"을 등록·수정·조회·목록·삭제
회원 ↳ resource, 등록·수정·조회·목록·삭제 ↳ method (행위)
⇒ ★ Resource와 method를 구분해, URI는 리소스만 식별하도록 설계하는 것이 좋음

↳ API URI 설계 < 리소스 식별 // URI 계층구조 활용 >.
resource ⇔ 회원
회원 | 목록 조회 ⇒ /members
| 조회 ⇒ /member / {id}
| 등록 → /members / {id}

⇒ /members / {id}
수정 ⇒ /members / {id}
삭제 ⇒ /members / {id}

⑤ HTTP 메서드

1) GET

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

- 리소스 "조회"
- 서버에 전달하고픈 데이터는 query (쿼리 파라미터, 쿼리 스트링..) 통해 전달.

⇒ 최근 get 메세지 body에 데이터를 담아 보낼 수 있다하지만, 실무 사용 X.
(지원하는 서버 많지 X)

2) POST

```
POST /members HTTP/1.1
Content-Type: application/json

{
  "username": "user1",
  "age": 20
}
```

~/mvc2/ login → LoginController..

```
→ @GetMapping("/login")
  public String loginForm
  (@ModelAttribute("loginForm") LoginForm ..)

→ @PostMapping("/login")
```

↳ 이 PostMapping에서, loginForm 뷰로부터 보낸 text/html을 요청데이터로 받아서 처리하는 코드 확인!!

- GET 메서드와 달리, 메세지 body로, 요청 데이터 전달!

"JSON만 보내는건 아니다.."

- POST 메서드는 주로, 본문으로 전달된 요청 데이터 처리!

① 새 리소스 등록 (생성)

② 요청 데이터 처리

- 단순 data 생성을 넘어, 프로세스를 처리하는 경우도 있!

ex) 주문에서 결제완료 → 배달시작 → 배달완료 .. 처럼 단순한 data 변경을 넘어, 프로세스 처리

ex) POST /orders / {orderId} / start-delivery

③ 다른 메서드로 처리하기 어려운 경우

ex) JSON으로 조회 데이터 넘겨야 하는데, GET 메서드 어려워..
⇒ POST

3) PUT

```
PUT /members/100 HTTP/1.1
Content-type : application/json
{
  "username" : "hello",
  "age" : 20
}
```

☆ 리소스를 덮어버린다!!

- 리소스가 없으면 대체
  - 있으면 변경

ex) 서버에 /members/100 리소스 내용이 {"age" : 20, "username : "user"}일 때,
클라이언트에서 {"age" : 50}으로 PUT Request 보내면,
username은 날라지고, age가 50으로 대체. ⇒ 리소스 "전체" 대체

- POST와 달리, 리소스 위치를 알고 이름 지정.

4) PATCH

```
PATCH /members/100 HTTP/1.1
Content-type: application/json
{ "age" : 50
}
```

VS

⇒ 리소스 "일부" 변경

- 리소스의 "부분" 변경

ex) /members/100 리소스 내용이 {"age" : 20, "username: "user"}일 때,
클라이언트에서 {"age : 50"}과 PATCH Request 보내면,
username 그대로 놔두고, age만 50으로 변경.

⇒ 실무 PATCH 제공 안되도 서버가 원할 때, 이때는 대부분 POST 사용!

5) DELETE

```
DELETE /members/100 HTTP/1.1
Host: localhost:8080
```

- 리소스 삭제

⑥ 참고하면 좋은, URI 설계 개념.

→ 일반적으로, 컬렉션·문서를 가지고 최대한 해결 (= 리소스 + GET·POST·PUT/PATCH·DELETE) 하다가, 해결안되면, 컨트롤 URI 사용.

1) 문서 document
- 단일 개념 (파일하나, 객체 인스턴스, 데이터베이스 row)
- /members/100 , /files/star.jpg

2) 컬렉션 collection
- 서버가 관리하는 리소스 디렉터리
- 서버가 리소스 URI를 생성하고 관리
- /members

3) 컨트롤러, 컨트롤 URI controller
- 1) + 2)만으로 해결하기 어려운, 추가 프로세스 실행
- 동사 직접 사용
- /members/{id}/delete

1) HTTP 메서드

//

2) HTTP 상태코드 (웹브라우저는, HTTP response 메시지를 보고, 화면을 렌더링해준다!)

↳ 당연한 사실이지만, 전체흐름 잡을 때 필요했음!

"클라이언트가 보낸 요청의 처리상태를, 응답에서 알려주는 기능"

1xx (Informational) : 요청이 수신되어 처리중..

2xx (Successful) : 요청 정상 처리

3xx (Redirection) : 요청이 완료되려면, 추가행동필요

4xx (Client Error) : 클라이언트 오류 - 잘못된 문법 등으로 서버가 요청 수행 못함

5xx (Server Error) : 서버 오류. 서버가 정상요청을 처리하지 못함.

1) 2xx (Successful) : 클라이언트 요청을 성공적으로 처리

200 OK

201 Created — 요청이 성공해, 새로운 리소스 생성됨.

202 Accepted — 요청 접수 △ 처리 완료 X. 배치 처리 같은 곳 (실시간으로 처리되지 않아도 되는 대용량의 데이터를, 일괄적으로 처리하는 방법)

204 No Content — 서버 요청 성공 △ 응답 payload 본문에 보낼 데이터 없을때..

2) 3XX : 요청이 완료되려면, 추가조치 필요 (Redirection)

⇒ Redirection 3종류 까지 301, 308

① 영구 리다이렉션 - 특정 리소스 URI가 영구적으로 이동.

ex) /members → /users

ex) /event → /new-event

② 일시 리다이렉션 302, 307, 303

→ PRG (Post - Redirect - Get) → velog 정리_PRG - 중복결제 막기 위해

③ 특수 리다이렉션 304

→ 결과 대신 캐시 사용.

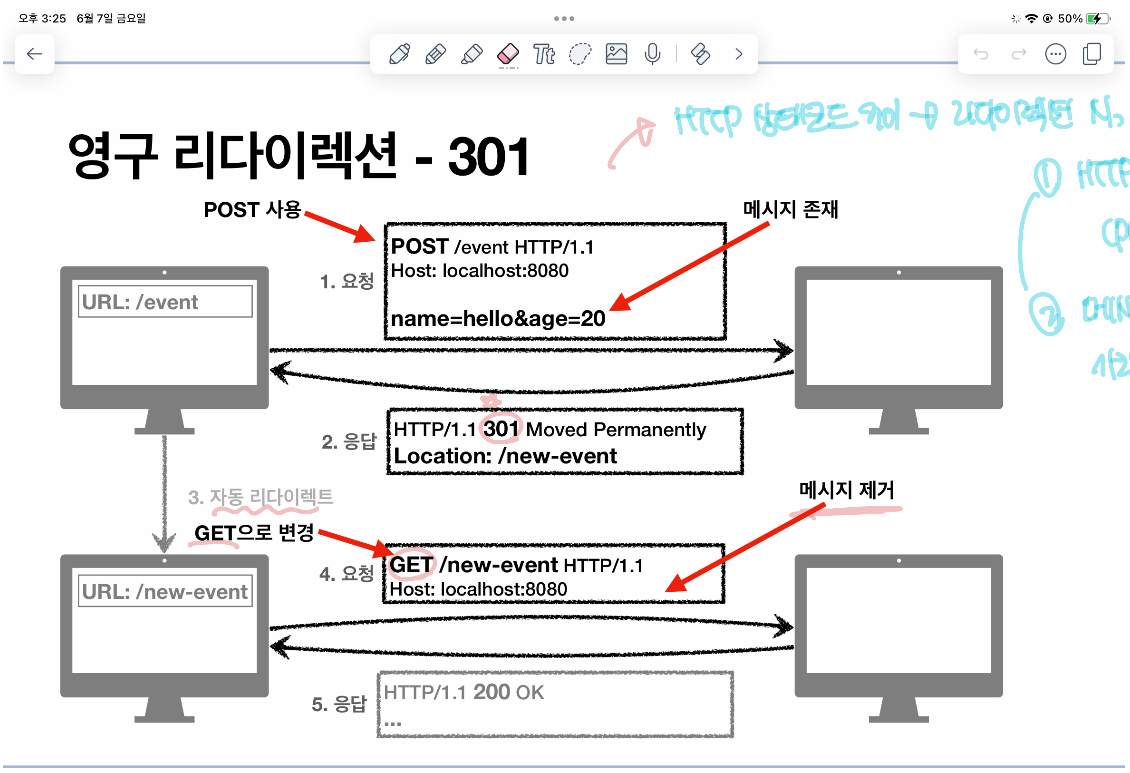
3xx (Redirection)

요청을 완료하기 위해 유저 에이전트의 추가 조치 필요

- 300 Multiple Choices
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 304 Not Modified
- 307 Temporary Redirect
- 308 Permanent Redirect

⇒ ① 영구 리다이렉션

- 301, 308
- 원래 URI 사용 X, 검색 엔진 등에서도 변경 인지
- 301 → Moved Permanently
  - 리다이렉트 시, ① 요청 메서드가 GET으로 변하고, ② 본문이 제거될 수도 있음 (May)
- 308 → Permanent Redirect
  - 301과 기능같음
  - 리다이렉트 시, ① 요청메서드와 ② 본문유지 (처음 POST보내면, 리다이렉트도 POST 유지)



→ HTTP 상태코드 301 → 리다이렉트 시,

① HTTP request 메서드 변함 (POST → GET)

② 메시지, 본문이 사라질수도 있다 (May)

⇒ 301, 308은 이런식으로 작동. △ 301·308 ⇒ 실무에서 잘 안씀!!

⇒ ② 일시 리다이렉션

302, 307, 303

→ 리소스 URI 일시적으로 변경.

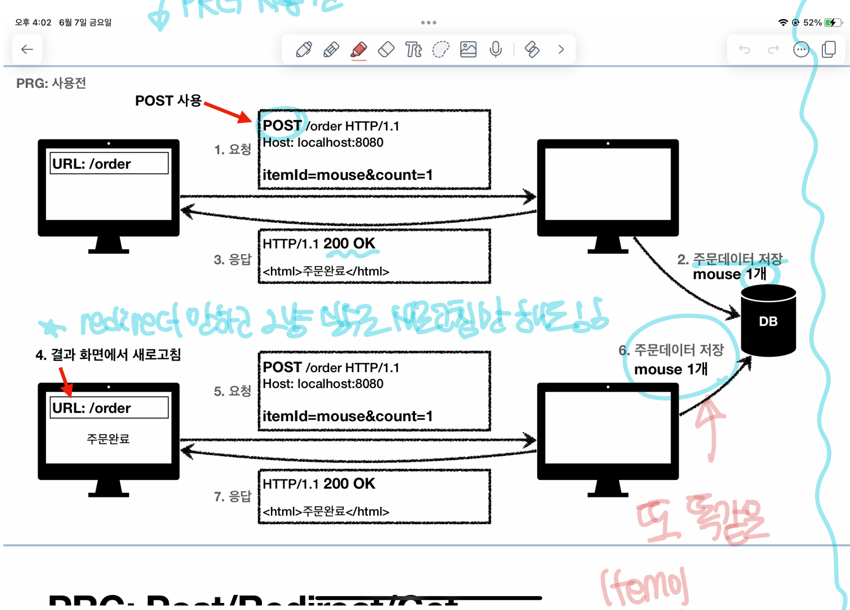
- 302 Found
  - → 리다이렉트 시, ① 요청메서드가 GET으로 변하고, ② 본문 제거될수도 (May)
- 307 Temporary Redirect
  - ⇒ 302와 기능은 같음
  - ⇒ 리다이렉트 시, ① 요청메서드, ② 본문유지 (요청메서드 변경하면 안된다!!)
- 303 See Other
  - ⇒ 302와 기능은 같음
  - ⇒ 리다이렉트 시, ① 요청메서드가 GET으로 변경 (확실)

→ 일시적인 리다이렉션 예시 (Post - Redirect - Get)

· 결론부터 말하면, PRG 쓰는 이유!
- POST로 주문 후, 새로고침으로 인한 중복 주문 방지
- POST 주문후, 자동으로 주문결과 화면을 GET 메서드로 redirect
- 새로고침해도, 결과화면 GET으로 조회하게끔!!
- 중복주문 대신, 결과화면만 GET으로 다시 요청

◦ PRG 사용 전후

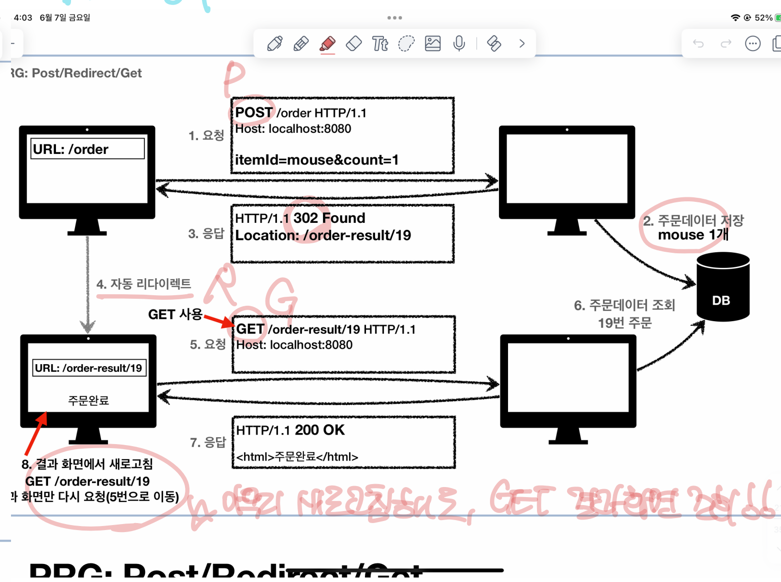
↳ PRG 사용전



* redirect 안하면 그냥 POST로 새로고침만 해도 ㅎㅎ

또 똑같은 (item이 저장됨!)

↳ PRG 사용후



☆ 아무리 새로고침해도, GET 결과화면 조회 ㅎㅎ

↳ 웹브라우저는, HTTP response 메시지를 보고, 화면을 렌더링해야함!

⇒ 그래서, HTTP 상태코드 - 3XX - ② 일시 리다이렉션에서 302·307·303 中 뭘 써야 하나?

- 정리

(
302 Found → GET으로 변할수도.. (Maybe) → 애매 maybe
307 Temporary Redirect → 메서드 변하면 안됨
303 See Other → 메서드가 GET으로 변경됨
) ⇒ 확실!

· 원래 처음에 302 등장 시, 의도는 HTTP 메서드를 유지하는 것 이었으나..
현재 대부분 웹브라우저들, GET으로 바꾸기에, 명확한 302 대신 307·303이 등장함

· 307, 303 권장하지만, 현실적으로 이미 많은 애플리케이션 라이브러리들이 302 기본값으로 사용

∴ (자동 리다이렉션 시 GET으로 변해도 되면, 그냥 302 사용해도 큰 문제 X)
PRG

⇒ ③ 기타 3XX

- 300 → 안쓴다
. 304 → Not Modified
⇒ 많이 쓰임
⇒ 요청된 자원에 대한 변경사항이 없으므로, 캐시되어 있던 자원으로 redirect 하겠다!
↳ 브라우저 최초 요청 시 200번 응답을 받지만, 이후에 자원에 변화가 없다면, 일정기간동안 304번 응답해줌.
↳ 단, 서버 재시작하면, 다시 200번 응답.

// → 이렇듯, HTTP 상태코드 3XX번대는, 요청된 자원에 대한 리다이렉트를 나타냄.

3) 4XX (Client Error)

. 오류의 원인이 클라이언트에게 있음.
※· 이미 클라이언트가 잘못된 요청·데이터를 보내고 있기에, 똑같은 재시도가 실패!

보통.. (
401 - 인증 X
403 - 권한 X
404 - 요청 리소스가, 서버에 없는 경우

4) 5XX (Server Error)

1) HTTP 메서드

- 오류의 원인이 서버에게 있음
· 서버에 문제가 있기에, 재시도하면 성공할수도 있음! (복구가 되거나.. 등등..)
↳ 서버문제만 다뤄야함!

2) HTTP 상태코드

7) HTTP 헤더

① HTTP헤더

HTTP헤더 : HTTP 전송에 필요한 모든 부가정보.

ex) 메시지 바디의 내용(크기, 압축, 인증, 요청 클라이언트, 서버정보, 캐시 관리정보 ...

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=UTF-8
Content-Length: 3423

<html>
  <body>...</body>
</html>
```

② 표현 데이터는 payload로 / 표현. 표현헤더 전달됨

→ "표현 데이터"는 payload를 통해 전달됨.

(최신)

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=UTF-8
Content-Length: 3423

<html>
  <body>...</body>
</html>
```

표현 헤더

메시지 본문

표현 데이터

; 통해 표현 데이터 전달

ıd)

→ 표현 : 요청 응답에서 전달할 실제 data

표현 헤더 : 표현 데이터를 해석할 수 있는 정보 제공

ㄴ 데이터 유형 (html, json), 데이터 길이, 압축정보..

개발자도구 - 네트워크에서 Response header 확인!

③ 표현 헤더 (Response header) → request · response 모두 쓰임!

Content-Type : 표현 데이터 형식 - text/html, charset=utf-8, application/json, image/png...

Content-Encoding : 표현 데이터 압축 방식 - gzip, deflate, identity.. // 데이터 읽는 쪽에서 인코딩 헤더 정보 읽고 압축 해제

Content-Language : 표현 데이터 자연 언어 - 표현데이터 자연언어 (ko, en, en-US)

Content-Length : 표현 데이터 길이 - 바이트 단위.

※ 표현 헤더는 request · response 모두 사용

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 16

{"data":"hello"}
```

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=UTF-8
Content-Encoding: gzip
Content-Length: 521

lkj123kljoiasudlkjaweioluywlnfdo912u34lj
ko98udjkl
```

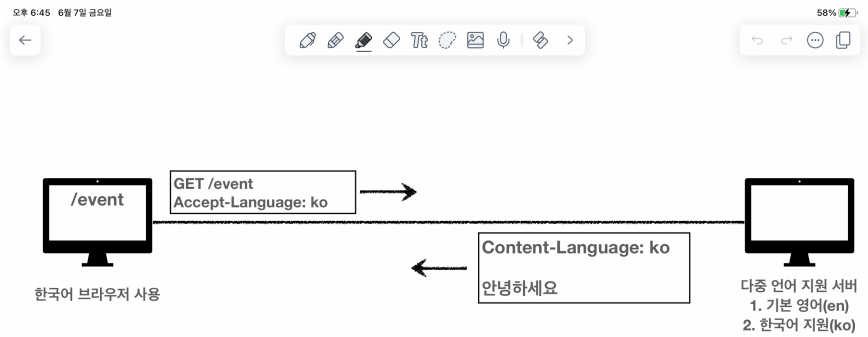
ㅓ 추가

⊕ 협상 (content negotiation)

⇒ 협상은, 클라이언트가 요청할 때만 사용되며, 앞에 Accept가 붙는다. → client의 request 때만 결재

(클라이언트가 선호하는 표현 요청)

- Accept : 클라이언트가 선호하는 미디어타입 지정.
- Accept-Charset : " " 문자 인코딩
- Accept-Encoding : " " 압축 인코딩
- Accept-Language : " " 자연 언어 ⇒ ex)



↑ 한국어 브라우저를 사용하는 클라이언트에서, 다중언어를 지원하는 서버에 Accept-Language을 ko로 보내면 답변을 ko로 받는다.

↑ 그런데, 다중 언어 지원 서버에서 ko를 지원하지 않는다면??

협상과 우선순위1

Quality Values(q)

GET /event
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7

⇒ 헤더에 설정된 Quality Values에 의해 언어가 결정된다.

- Quality Values(q) 값 사용
- 0~1, **클수록 높은 우선순위**
- 생략하면 1
- Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
  - 1. ko-KR;q=1 (q생략)
  - 2. ko;q=0.9
  - 3. en-US;q=0.8
  - 4. en;q=0.7

언어 없다면 2번.. 2번 없다면 3번 .. → 우선순위 적용

협상과 우선순위2

Quality Values(q)

GET /event
Accept: text/*, text/plain, text/plain;format=flowed, */*

- 구체적인 것이 우선한다.
- Accept: **text/***, **text/plain**, **text/plain;format=flowed**, ***/***
  1. text/plain;format=flowed
  2. text/plain
  3. text/*
  4. */*

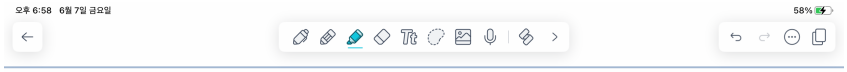
또한 Quality Values는 개발자가 강조 명시되어 있을 시 구체적인 것이 우선한다

https://tools.ietf.org/html/rfc7231#section-5.3.2

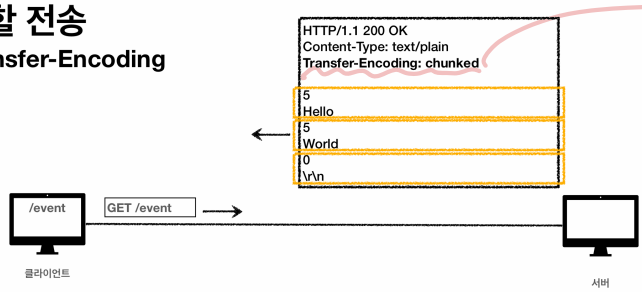
⑤ 전송방식 (p.258~)

전송방식 4가지
(서버-클라이언트)

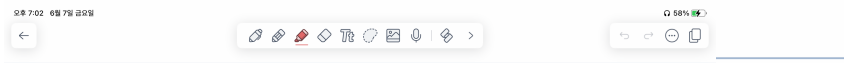
- 단순전송 - 한번에, 헤더에 Content-Length를 이용해 전송
- 압축전송 - 단순전송 + 헤더에 Content-Encoding 타입 지정해, 전송
- 분할전송 - 아래 그림 참고
- 범위전송 - 아래 그림 참고



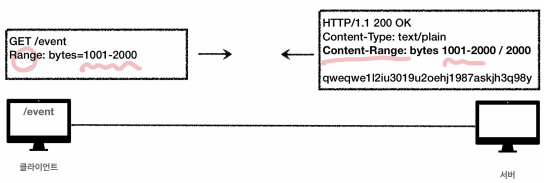
분할 전송
Transfer-Encoding



만약 클라이언트에서 헤더에 Transfer-Encoding 설정해서 서버에 전송했다면,
서버도 똑같이 각각 끊어서 보낸다.



범위 전송
Range, Content-Range



범위전송은,
클라이언트에서 Range 설정해서 보내주면,
서버에서 Content-Range를 통해, 클라이언트가 요청한 범위를 보내준다.

(Header 개념 일반정보 - p.261~) From, Referer, User-Agent, Server, Date
(Header 개념 특별한 정보 p.267~) Host, Location, Allow, Retry-After, Date

→ 알아둘 것!!

1) HTTP method
2) HTTP 상태코드
3) HTTP 헤더