# HTTP 프로토콜 (기초)

## HTTP 프로토콜에서 사용되는 요청, 응답 메시지의 구조

- 기본적으로 요청, 응답 메시지 모두 **시작줄, 헤더(Header) 영역 및 바디(Body or Payload) 영역**으로 구분됨
- 각 줄은 **개행문자(CR+LF)**로 구분
  - CR, LF 중 하나만 쓰면 안되고 반드시 둘 다 써줘야 함
- (시작줄+헤더)와 바디 사이를 **개행으로 구분**함

### 요청(Request) 메시지 형식

- 시작줄
  - (HTTP메소드) (Path) (HTTP버전)
- 헤더 영역
- 바디 영역
  - GET 요청의 경우 **바디 내용을 포함하지 않는 것이 권장**됨
    - https://stackoverflow.com/questions/978061/http-get-with-request-body

**요청 메시지 예시**

```
GET /hello HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/72.0.3626.121 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;
q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: ko,en-US;q=0.9,en;q=0.8,ko-KR;q=0.7
```

**해설**

```
1. 시작줄 => GET 메서드로 요청을 보내고 있으며 요청 주소는 "/hello"이고 HTTP 프로토콜 버
전 1.1로 통신하려함
2. 총 5개의 헤더 정보(Host, User-Agent, Accept, Accept-Encoding, Accept-Language)를
보내고 있음
3. 또한 바디 데이터는 존재하지 않음 (개행도 없고 데이터도 없음)
```

# 응답(Response) 메시지 형식

- 시작줄
  - (HTTP버전) (상태코드) (상태메시지)
- 헤더 영역
- 바디 영역
  - 응답 메시지라면 일반적으로, 거의 대부분 **바디 내용을 포함**하지만 역시 이것도 의무사항은 아님

**응답 메시지 예시**

```
HTTP/1.1 200 OK
Content-Encoding: gzip
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html; charset=UTF-8
Date: Wed, 06 Mar 2019 08:09:37 GMT
Etag: "1541025663+ident"
Expires: Wed, 13 Mar 2019 08:09:37 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (sjc/4E45)
Vary: Accept-Encoding
X-Cache: HIT
Content-Length: 606

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />

(.. 중략 ..)

    <p><a href="http://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

**해설**

```
1. 시작줄 => HTTP 프로토콜 버전 1.1으로 통신하고 있으며 200 상태코드를 전달하고 200 상태코드와 관련된 상태 메시지는 OK임
2. 총 12개의 헤더 정보를 보내고 있음
3. 바디 데이터는 존재하며, HTML 페이지의 내용임을 유추할 수 있음. 14번 라인에 헤더와 바디를 구분하는 개행이 있는데 가독성 때문에 존재하는게 아니고 스펙상 반드시 넣어야 되는 개행임을 유의
```

# HTTP 메서드

**HTTP 메서드와 각 메서드의 기능**

| 메서드 | 기능 |
| --- | --- |
| **GET** | 리소스 조회 |
| **POST** | 리소스 생성 |
| PUT | (**리소스를 완전히 교체**한다는 의미에 더 가까운) 리소스 수정 ≒ **REPLACE** |
| PATCH | (**리소스의 일부 내용을 수정**한다는 의미에 더 가까운) 리소스 수정 ≒ **MODIFY** |
| DELETE | 리소스 삭제 |
| (*) HEAD | 기본적으로는 GET 메소드와 똑같은 역할을 하지만 **헤더 정보만 요청** (응답 메시지에 바디 정보가 포함되지 않음) |
| (*) OPTIONS | 해당 주소에 **요청 가능한 모든 메소드 정보**를 요청 (Allow 헤더 정보를 추출)<br>(단, 보안상의 이유로 서버에서 해당 메소드를 이용한 요청을 거절할 수 있음) |
| (*) TRACE | 통신 중간에 거쳐가는 라우터 정보를 추적하고 반환받기 위해서 쓰는 헤더 |

- GET, HEAD는 **조회용 메소드이므로 캐시 가능**
- GET 메소드를 이용한 **새로운 리소스 생성 및 수정 행위는 금지**됨
    - 이전의 **요청 결과가 캐시되어 실제 서버에 요청이 전달되지 않을 가능성**도 있으며, 중간 라우터가 해당 요청을 다시 보내서 리소스가 중복 생성될 수도 있음
- GET 메소드를 이용한 요청 메시지에 바디를 포함시키는 것은 **가능하지만 권장되지 않음**
- PUT은 **데이터의 완전한 교체** 작업에 PATCH는 데이터의 정보를 **일부 수정**하는 작업에 적합한 메소드

```
Several applications extending the Hypertext Transfer Protocol (HTTP) require a feature to do partial resource modification. The existing HTTP PUT method only allows a complete replacement of a document. This proposal adds a new HTTP method, PATCH, to modify an existing HTTP resource.
```

- OPTIONS 요청 후 응답 메시지 예시

```
> curl -X OPTIONS http://example.com -i

HTTP/1.1 204 No Content
Allow: OPTIONS, GET, HEAD, POST
Cache-Control: max-age=604800
...
```

- HTTP 메서드는 **HTTP 동사(Verb)**라고도 불리움
    - GET => 조회하다, POST => 생성하다, ...

## 안전한 메소드와 메소드의 멱등성(Idempotence)

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET 페이지에서 Safe, Idempotent, Cacheable 여부 확인 가능
- **안전한 메소드 (Safe method)**

    - 정보를 **조회(read-only)만 하는 메소드는 안전한(Safe) 메소드**
    - 대표적으로 GET 메소드가 안전한 메소드
- **(*) 멱등적인 메소드 (Idempotent method)**
    - 여러번 같은 내용의 요청을 보냈을 때 한 번 보냈을 때와 결과(=최종 리소스 상태)가 같다면 **멱등적인(Idempotent) 메소드**

| 메소드 | Safe | Idempotent | Cacheable |
|---|---|---|---|
| GET | Y | Y | Y |
| HEAD | Y | Y | Y |
| POST | N | N | **N (*)** |
| PUT | N | Y | N |
| PATCH | N | N | N |
| DELETE | N | Y | N |

- 단, 일반적으로 POST 요청의 캐싱은 권장되지 않음
- 안전한 메소드인지 여부 따져보는 방법 => 메서드 요청 결과로 인하여 **서버의 상태(보통 데이터베이스에 저장된 내용)가 변경**되는가?
    - GET 메소드는 **리소스 정보를 조회**하는 작업을 수행 (조회 전용 메소드)
    - POST 메소드는 **새 리소스를 생성**하는 작업을 수행
    - PUT 메소드는 **리소스를 교체**하는 작업을 수행
    - PATCH 메소드는 **리소스의 내용을 수정**하는 작업을 수행
    - DELETE 메소드는 **리소스를 삭제**하는 작업을 수행
    - 따라서 GET 메서드를 제외하고 **POST, PUT, PATCH, DELETE 모두 안전하지 않은 메소드**
- 멱등적인 메소드인지 여부 따져보기 => **같은 내용**으로 요청을 여러번 보내도 결과(**응답 메시지의 내용이 아니라, 서버의 상태**)가 같은가? => (한 번의 요청이 아닌) **연속적인 요청**으로 인하여 서버 상태의 변경을 유발하는가?
    - POST 메소드는 새 리소스를 생성하는 작업을 수행하므로 N번의 요청은 N개의 리소스를 생성함
        - 즉, 여러번 요청할 경우 서버의 상태가 계속해서 변경되므로 멱등적이지 않음
    - PUT 메소드는 리소스를 교체하는 작업을 수행하므로 여러번 같은 내용으로 교체해도 서버의 정보(상태)는 변경되지 않음
    - PATCH 메소드는 **요청의 내용에 따라 멱등적일수도 아닐수도 있음**
        - 멱등적인 PATCH 요청
            - `{"op": "set", "field": "age", "value": 20}`
            - 나이를 20세로 설정 (여러번 요청해도 계속 나이가 20인 상태로 같은 결과 => 멱등적)
        - 멱등적이지 않은 PATCH 요청
            - `{"op": "add", "field": "age", "value": 1}`
            - 나이를 1 증가 (요청할때마다 나이가 1살 많아짐 => 멱등적이지 않음)
            - **요청 내용 자체는 변경되지 않았음**을 유의 (같은 내용으로 요청을 보냄)

        - 일부 연산은 멱등적일지 몰라도 **전체 연산이 모두 멱등적이라고 확신할 수 없으므로 멱등성이 없다고 가정**해야 함
    - DELETE 메소드는 특정 리소스를 삭제하고 **이후 리소스를 삭제하는 작업은 매번 실패할 것이고 해당 리소스는 제거된 상태로 남아있으므로 서버 상태에 변화가 없음**, 따라서 멱등적임
- 수학 연산자에서 따져보는 멱등성
    - 1을 곱하는 연산 작업은 안전하지 않고 멱등적임 (내용이 바뀌지 않는다는 측면에서는 안전하지만 곱하기는 read-only 연산이 아님)
    - 증감 연산자(++, --)는 안전하지 않고 멱등적이지도 않음
    - 절대값 함수는 안전하지 않고 멱등적임

```
In mathematics, an idempotent operation is one where f(f(x)) = f(x). For example, the abs() function is idempotent because abs(abs(x)) = abs(x) for all x.
```

- 단순히 응답 메시지의 내용이 바뀌었다고 해서 멱등적이지 않다고 하는 것은 옳지 않으며 **서버의 상태가 변경되었는지 여부**가 멱등적인지 여부를 결정함
    - 가령, DELETE 메서드의 경우 처음 리소스를 삭제할 때에는 204 응답 코드 메시지가 전달되고, 이후 또 한 번 삭제 시도를 할 때에는 404 응답 코드가 담긴 다른 메시지가 전달될 수 있음, 그러나 **서버의 상태(여기서는 데이터베이스)가 바뀌지는 않았기 때문**에 DELETE 메서드는 멱등적인 메서드임
    - https://stackoverflow.com/questions/44906076/if-a-get-requests-response-changes-is-idempotency-respected

# 공통 헤더

- 요청, 응답 메시지에 **공통적으로 쓰이는 헤더**

## Content-Type

```
Content-Type: application/x-www-form-urlencoded
```

- 바디에 포함된 데이터의 **MIME 타입**을 지정할 때 사용
- MIME(Multipurpose Internet Mail Extensions) 타입
    - 메일의 첨부 파일의 파일 형식을 나타내기 위해서 도입된 개념으로 "타입/서브타입"의 형태로 파일 형식을 나타됨

**MIME 타입 예시**

```
text/plain : 텍스트, 일반 평문
text/html : 텍스트, HTML 문서
text/css : 텍스트, 스타일시트
text/* : 모든 종류의 텍스트

application/json : 텍스트, JSON
application/xml : 텍스트, XML

image/png
image/jpeg
```

```
audio/wav
audio/ogg

video/mp4

application/x-www-form-urlencoded : 파일이 포함되지 않은 폼 데이터
multipart/form-data : 파일이 포함된 폼 데이터

application/pdf
application/octet-stream : 바이트 배열 (주로 파일 전송과 관련)

*/* : 모든 종류의 콘텐츠
```

- 문자열 형태의 데이터를 전송하는 경우 **인코딩 정보(charset)에 대한 정보를 뒤에 포함**시킬수 있음 (대부분 utf-8이며 charset 정보를 생략하는 것은 권장되지 않음)

```
Content-Type: text/html; charset=euc-kr

Content-Type: application/json; charset=utf-8

Content-Type: application/xml; charset=iso-8859-1
```

## Content-Length

```
Content-Length: 348
```

- 바디 영역에 포함된 **데이터의 크기(단위는 바이트)를 지정**하기 위해서 사용됨
- 바디 영역의 내용이 압축된 경우 **압축된 상태의 데이터 크기로 지정**됨

## Content-Encoding

```
Content-Encoding: gzip
```

- 콘텐츠를 **압축할 때 사용된 압축 알고리즘을 명시**하기 위해서 사용
  - gzip, deflate 알고리즘이 자주 쓰이며 br(브로틀리)이라는 압축 방식도 새로 도입됨
  - identity로 지정시 압축을 하지 않음
- 콘텐츠를 압축하고 해제하는 과정에 CPU가 개입하여 계산을 해야되서 오버헤드가 있지만, 대역폭을 좀 덜 쓰는 것이 선호되므로 거의 대부분 데이터를 압축해서 보냄
- 보낼 내용을 압축할 경우 서버에서 내용을 압축하는데 CPU를 쓰고 클라이언트에서는 압축을 해제하는데 CPU를 씀
  - 이 시간(CPU bound 작업)이 일반적으로 통신 작업(IO bound 작업)보다 시간이 덜 걸려서 반응성이 좋아지므로 거의 대부분 내용을 압축해서 보냄
    - 특히, 텍스트 데이터의 경우 압축률이 좋은편임
  - 즉, 서버의 경우 => 압축하지 않고 그대로 데이터를 보내는 시간 >> (압축 시간 + 압축된 데이터를 보내는 시간)
  - 클라리언트의 경우 => 압축하지 않은 데이터를 그대로 받는 시간 >> (압축된 데이터를 받는 시간 + 압축을 푸는 시간)

## Location

```
Location: http://www.example.org/index.php

Location: /index.html
```

- 이동(redirect) 작업시에 이동할 주소(URL)를 설정
- 주로 **3XX로 시작하는 HTTP 상태 코드**와 함께 사용됨
- 201 응답코드와 사용될 경우 생성된 리소스 페이지로 이동할 링크 제공
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Location

## Connection

```
Connection: Keep-Alive
Keep-Alive: timeout=5, max=1000

Connection: Close

Connection: Upgrade
```

- Keep-Alive
  - Connection 값을 "Keep-Alive"로 설정하는 동시에 같이 보내지는 헤더
  - HTTP 연결을 위한 TCP 핸드쉐이크 작업을 줄이기 위해서 **연결 소켓을 계속 유지**해달라는 요청을 보내기 위한 헤더
  - 소켓을 유지하지 않을 경우 매 요청마다 TCP 핸드쉐이크 작업 진행(최소 3번의 메시지 교환)되므로 비효율적
  - timeout 값은 연결 소켓을 유지할 시간을 지정하기 위해서 max 값은 연결이 이루어진 후 연속된 요청의 최대 개수에 대한 힌트를 제공하기 위해서 전달
    - 보통 서버측에서 어느 시점에서 소켓을 닫아줘야 할 지 추측하기 어렵기 때문에 명시적으로 종료하기보다 소켓 timeout이 발생하길 기다림
      - https://stackoverflow.com/questions/140765/how-do-i-know-when-to-close-an-http-1-1-keep-alive-connection
- Close
  - 응답 메시지를 보낸 후 **소켓을 닫을 것을 명시적으로 요청** (서버, 클라이언트측에서 모두 요청 가능)
- Upgrade
  - 다른 프로토콜(ex: 웹소켓)으로 프로토콜을 변경하여 통신이 이루어지도록 요청하기 위해서 사용됨

# 주요 요청(Request) 메시지 헤더

## Host

```
Host: www.example.com
```

- 요청을 보낼 도메인 주소
- HTTP 1.1 버전 이후 **필수적으로 포함되는 헤더**
- Path 정보(쿼리 스트링을 포함한)는 **시작줄에 포함**됨

## User-Agent

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/72.0.3626.121 Safari/537.36
```

- 클라이언트 **소프트웨어 정보(OS정보, 웹 브라우저 정보)**를 보내주는 헤더
- **구버전의 브라우저 지원(ex: IE6, IE7 등)을 위해서 User-Agent를 확인**하는 경우가 있음
  - 자바스크립트에서도 navigator 객체를 통해서 확인 가능

```
console.log(navigator.userAgent);
```

## Accept

```
Accept: text/html

Accept: image/*

Accept: text/html, application/xhtml+xml, application/xml;q=0.9, */*;q=0.8
```

- 서버로부터 전달받기를 원하는 콘텐츠 형식을 **MIME 타입**으로 지정하는 헤더
- q값을 이용해서 **선호도를 조정**할 수 있으며 이러한 일련의 과정을 **내용 협상(Content Negotiation)**이라고 부름
  - **q값이 클 수록 선호**하는 데이터 타입
  - q값이 없는 경우 1.0으로 간주함
- 주로 ***/*(모든 종류의 콘텐츠)**를 마지막에 지정함
- Accept 헤더 정보가 없다면 **모든 종류의 컨텐츠 전달(*/*)로 인식**하여 처리

## Accept-Encoding

```
Accept-Encoding: gzip, deflate
```

- 클라이언트측에서 이해할 수 있는, **응답으로 받고 싶은 압축 알고리즘 방식을 명시**함
  - 구버젼 브라우저에서는 특정 압축 알고리즘으로 압축하면 못 푸는 경우도 있어서 이 때 유용하게 사용 가능함

## Accept-Charset

```
Accept-Charset: utf-8, iso-8859-1;q=0.5, *;q=0.1
```

- 문자열 데이터를 기대할 때 원하는 문자열 인코딩을 지정 (ex: utf-8)
- 와일드 카드(*)로 **모든 종류의 charset**을 지정 가능
- Accept 헤더와 마찬가지로 q 값을 통해서 선호도 조정 가능

## Accept-Language

```
Accept-Language: fr-CH, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5
```

- 다국어를 지원하는 서비스의 경우 클라이언트측에서 선호하는 언어를 설정
- 웹 브라우저의 경우 기본적으로 운영체제의 설정 언어를 보고 해당 헤더값을 설정

## Referer

```
Referer: https://www.google.com/
```

- 현재 요청된 페이지에 진입하기 **직전에 머물렀던 웹 페이지 주소** 정보를 전달하기 위한 헤더
- Referrer의 오타 (그러나 하위 호환성을 유지하기 위해서 수정하지 않고 계속 이어짐)
- 광고 배너를 클릭해서 오는 경우 리퍼러 헤더를 분석하면 어디서 유입되었는지 여부를 알 수 있음

## Cookie

```
Cookie: <cookie-list>

Cookie: name=value
Cookie: name=value; name2=value2; name3=value3
```

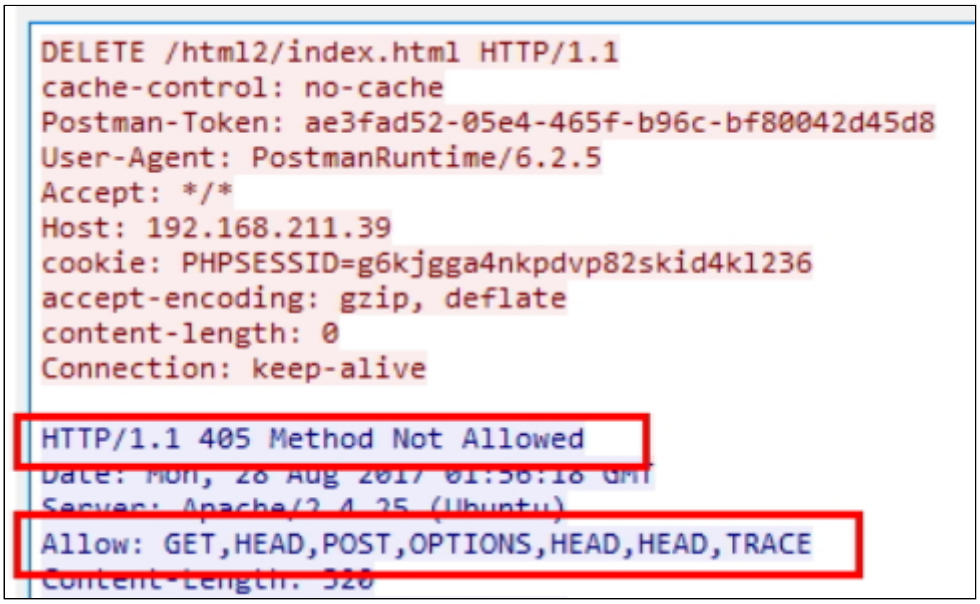
- 요청을 보낼 때 **서버로부터 전달 받아 생성한 쿠키값을 전송**하기 위해 사용
    - Set-Cookie 헤더는 **서버에서 클라이언트측**으로 생성하길 원하는 쿠키 정보를 전달하기 위해서 사용하는 헤더로 Cookie와는 다른 헤더임을 유의

# 주요 응답(Response) 메시지 헤더

## Allow

```
Allow: GET, HEAD
```

- 해당 주소에서 어떤 메소드를 허용하는지 알려주기 위해 사용
- **GET, HEAD 메소드는 필수적으로 지원**해야 함
  - GET이 필수적으로 필요한 이유
    - 요청을 보내면 비록 잘못된 요청이더라도 4XX, 5XX로 시작하는 에러 응답을 해줘야 하므로 어떤 주소에서건 지원해야 함
  - HEAD가 필수적으로 필요한 이유
    - GET과 똑같이 작동하지만 **응답에 바디가 없는 형태**로 헤더 정보만 받는 메서드이므로 지원해야 함
- 주로 **405(Method not allowed) 상태 코드**와 같이 사용됨




## Last-Modified

```
Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
```

- 해당 리소스가 **마지막으로 수정된 시간**을 표시
- 해당 정보를 이용하여 **캐시된 파일 사용 및 갱신 여부를 결정**하는데 사용함

## Set-Cookie

```
Set-Cookie: <cookie-name>=<cookie-value>

ex)
HTTP/1.0 200 OK
Content-type: text/html
Set-Cookie: yummy_cookie=choco
Set-Cookie: tasty_cookie=strawberry
```

- 특정 (이름, 값) 쌍의 쿠키값을 저장하고 다음 요청부터 서버로 보낼 수 있도록 클라이언트에게 요구하기 위해서 사용되는 헤더
- 쿠키값에는 도청당해도 상관없는 중요하지 않은 데이터를 저장해야 함
    - 장바구니 정보 O, 다크모드 정보 O, 민감한 개인 정보 X
- 단, 예외적으로 쿠키에 저장되는 세션 ID와 같은 **민감한 정보는 HttpOnly, Secure 옵션을 설정**하여 보냄
    - HttpOnly : 자바스크립트에서 접근 불가 (document.cookie 값에 접근해도 읽을 수 없음)
    - Secure : https로 연결된 상태에서만 쿠키값을 전송

```
Set-Cookie: JSESSIONID=3CB361E0BE1A9A7DE7DB926DF0772BAE; Secure; HttpOnly
```

## Transfer-Encoding

- **chunked**로 값을 설정할 경우 **스트림 형식**으로 데이터 전송
- 조각조각 나뉜 상태로 연속해서 데이터를 전달할 수 있으므로 **큰 사이즈의 데이터나 혹은 당장 크기를 알 수 없는 데이터를 계속해서 전송해야 할 경우 유용하게 사용** 가능
- HTTP/2에서는 더 이상 사용되지 않는 헤더
    - HTTP/2에서는 애초에 모든 응답 데이터를 **개선된 스트림 형식으로 전송**하므로 해당 헤더가 필요 없음
    - HTTP/2 doesn't support HTTP 1.1's chunked transfer encoding mechanism, as it provides its own, more efficient, mechanisms for data streaming.

```
Transfer-Encoding: chunked
```

**메시지 구조**

```
...
Transfer-Encoding: chunked
(CRLF)
(바이트 단위의 데이터 크기)(CRLF)
(실제 데이터)(CRLF)
(바이트 단위의 데이터 크기)(CRLF)
(실제 데이터)(CRLF)
...
0(CRLF) <= 0은 스트림의 끝을 알림
(CRLF)
```

**실제 메시지**

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

7\r\n
Mozilla\r\n
9\r\n
Developer\r\n
7\r\n
Network\r\n
0\r\n
\r\n
```

## 커스텀 헤더

- `X-`로 시작하는 **커스텀 헤더**를 정의하여 원하는대로 사용 가능
  - 본인이 필요한 임의의 헤더 정보가 있을 때 "X-"로 시작하는 임의의 헤더를 이용해서 값 전달
  - "X-"로 시작하는 특정 헤더들은 자주 사용되어 거의 표준 헤더처럼 사용됨
    - X-Powered-By : 서버 어플리케이션의 정보 전달을 위해서 사용
      - `X-Powered-By: PHP/5.2.17`
    - X-Requested-With : 주로 요청이 자바스크립트를 통해어 이루어짐을 알려주기 위해서 사용 (ajax)
      - `X-Requested-With: XMLHttpRequest`
    - X-Frame-Options : 해당 페이지가 iframe과 같은 태그를 통해서 다른 페이지 내부에 삽입되어서는 안된다고 명시하기 위해서 사용 (해당 헤더는 진짜로 문서가 존재하는 표준 헤더임)
      - ```
        // 금지
        X-Frame-Options: DENY
        // 같은 Origin이라면 허용
        X-Frame-Options: SAMEORIGIN
        ```
  - 최근에는 "X-"로 시작하는 **커스텀 헤더 사용이 권장되지 않는 편**임
    - 커스텀 헤더라고 하더라도 특별 대우 없이 적당히 알아서 센스있게 짓기를 권장함
  - 보통 외부 API 호출(네이버, 카카오 등)할 때 토큰 전달용으로 많이 사용됨

## HTTP 응답 메시지 관련 상태 코드

- https://www.webfx.com/web-development/glossary/http-status-codes
        ```

## 2XX (성공)

- **200 OK**
  - 성공적으로 요청이 수행됨
- 201 Created
  - 성공적으로 요청이 수행되었으며 해당 요청의 결과로 인해 **하나 이상의 리소스가 생성**됨
    - 리소스 생성과 관련된 응답 코드이므로 보통 POST 요청 메시지의 응답 메시지로 사용됨
  - 응답 메시지의 Location 헤더에 해당 리소스와 관련된 주소(URL)를 돌려주는 것을 권장 (그러나 **브라우저상에서의 강제 리다이렉션이 이루어지지 않으므로** Location 헤더 관련 처리는 호출측에서 처리해야 함)
- 202 Accepted
  - 성공적으로 요청이 전달되었으며 해당 **요청의 결과로 (대체적으로 일정 이상 시간이 요구되는) 비동기 작업이 진행**되고 있음 (즉, 결과값을 응답 메시지에 바로 실어서 전달할 수 없는 상황)
    - https://farazdagi.com/2014/rest-and-long-running-jobs/
  - 일반적으로 **진행 상황 or 결과값을 확인할 수 있는 상태 페이지**에 대한 링크를 같이 전달

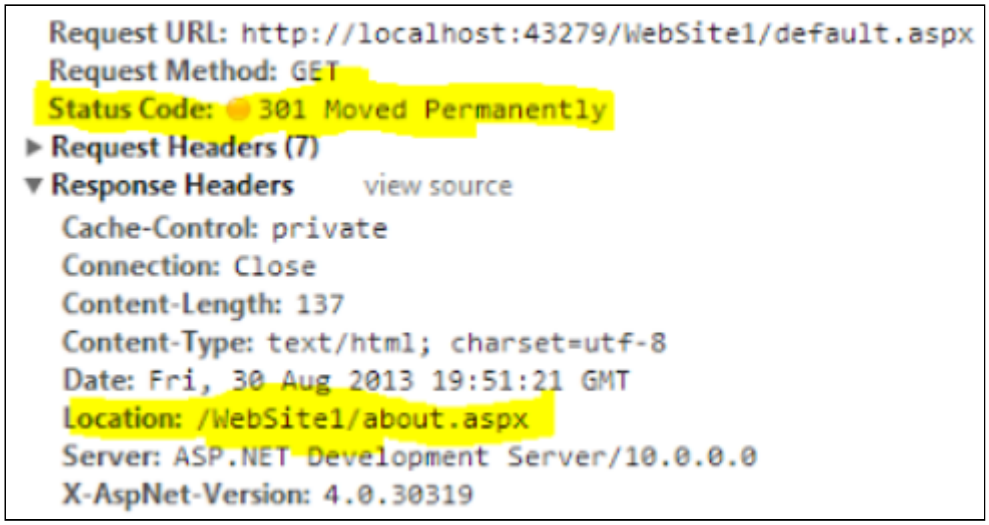
```
HTTP STATUS 202 (Accepted)

{
    "task": {
        "href": "/api/company/job-management/jobs/2130040",
        "id": "2130040"
    }
}
```

- 204 No Content
  - 말 그대로 작업이 성공하였고 바디에 포함된 정보는 없다는 의미
  - **DELETE 메소드 요청에 대한 응답**으로 적절 (성공적으로 삭제 작업이 수행되었으며 삭제 결과로 딱히 전달할 데이터가 없으므로 바디에 정보는 생략)
    - https://stackoverflow.com/questions/2342579/http-status-code-for-update-and-delete
- 206 Partial Content
  - Content-Range 헤더가 붙는 **(전체 내용이 아닌) 일부 내용을 전송**하는 응답 메시지의 코드
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/206

## 3XX (리다이렉션 및 캐시 관련)

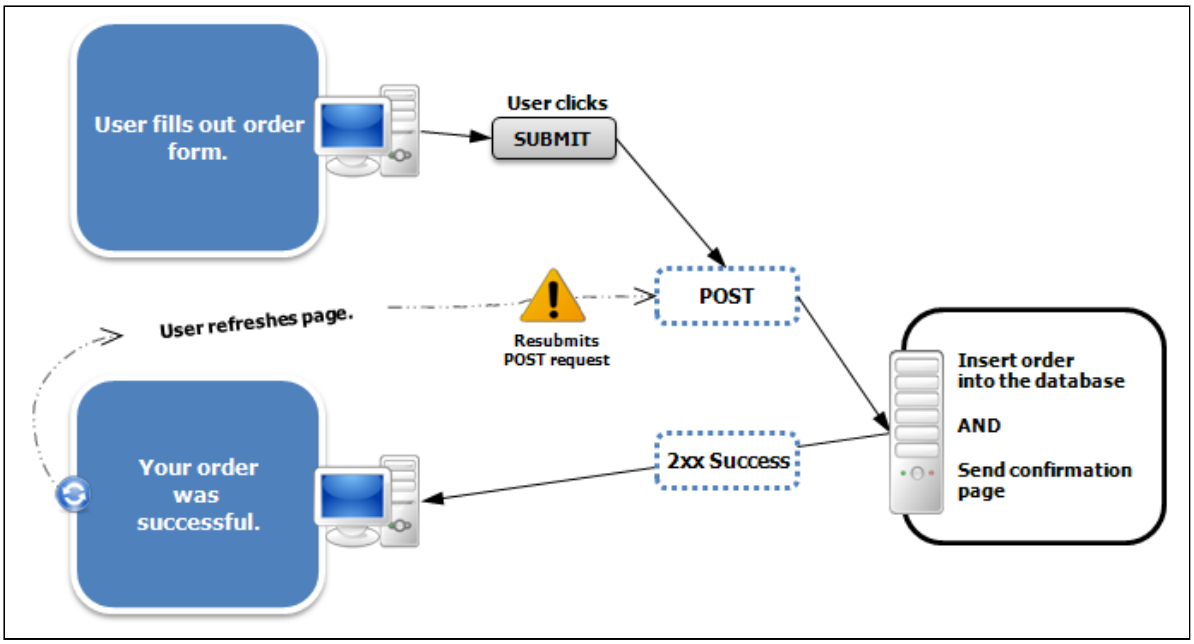
- **301 Moved Permanently**
  - 요청한 리소스 URL이 **영구적(Permanently)으로 다른 URL로 교체**됨
  - **Location 헤더**를 이용하여 교체된 URL 정보를 전달함
  - 웹 브라우저는 301 코드를 받으면 Location 헤더에 포함된 주소로 자동으로 리다이렉트를 함
    - 그리고 영구적 교체이므로 북마크에 해당 주소로 북마크된 내용이 있으면 주소를 새 주소로 변경해 줌



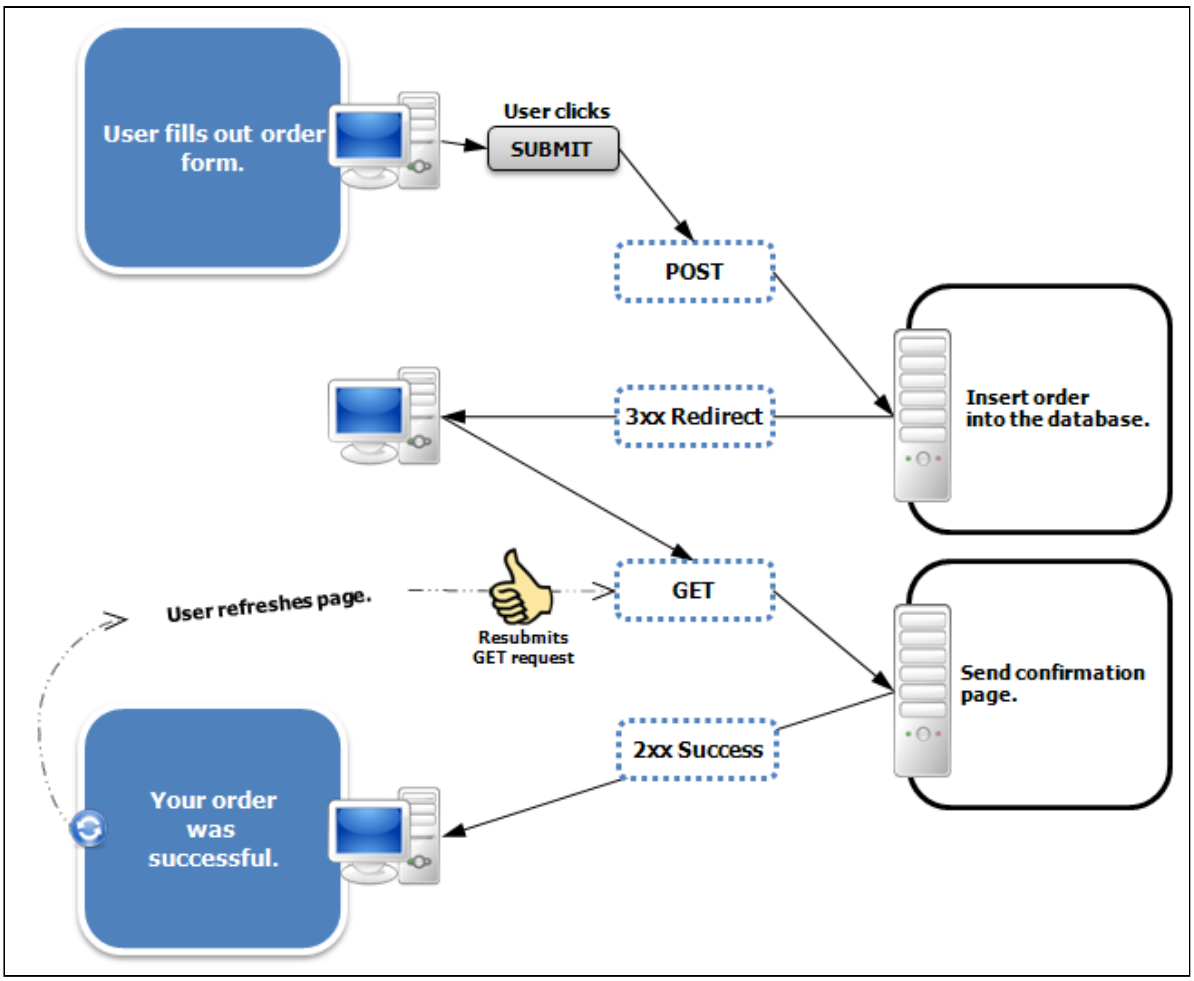


- **302 Found**
  - 요청한 리소스 URL이 **일시적(Temporary)으로 다른 URL로 교체**됨
  - Location 헤더를 사용하고 301과 같은 방식으로 동작
  - 301과 비슷하게 리다이렉트를 시도하지만, 일시적인 URL 교체이므로 따로 북마크 정보를 수정하는 작업 따위는 하지 않음
- **303 See Other**
  - 해당 주소와 **직접적 혹은 간접적인 정보를 포함**한 다른 URL 리소스로의 이동 처리를 위해서 사용됨
  - **PRG(POST/REDIRECT/GET) 패턴**
    - POST를 이용하여 리소스를 생성하고 응답 내용(ex: 결제 완료 페이지 등)을 전달받은 후, 그 페이지를 **새로 고침하면 이전의 POST 요청을 다시 보내게 되어 리소스를 중복 생성**하게 됨
    - 이 경우 POST 요청의 응답으로 303 응답코드를 보내고 응답 내용이 담긴 주소로 GET 메서드를 통해서 이동하도록 유도하면, 이후 새로고침을 해도 해당 페이지 리소스를 GET 메서드로 요청하게 되므로 리소스 중복 생성 문제가 발생할 확률이 줄어듬
      - POST 메서드로 상품 구매 주소(ex: /purchases)로 요청보내서 구매 내역 리소스 생성 => 요청 성공적으로 처리 후 304 응답 코드와 함께 Location 헤더에 해당 상품 구매와 관련된 결제 완료 주소(ex: /purchases/20) 로 이동하게 함 => 이후 GET 메서드로 해당 주소로 이동 => 새로 고침을 해도 해당 결제 완료 주소로 요청을 보내므로 문제 해결
    - 단, PRG 패턴을 적용했다고 해서 중복 리소스 생성 문제가 완전히 해결되는 것은 아니므로, 서버 로직을 통해서 똑같은 요청 내용이 안 생기도록 검증 코드가 있긴 해야 함
      - https://stackoverflow.com/questions/10827242/understanding-the-post-redirect-get-pattern

PRG 패턴 적용 이전



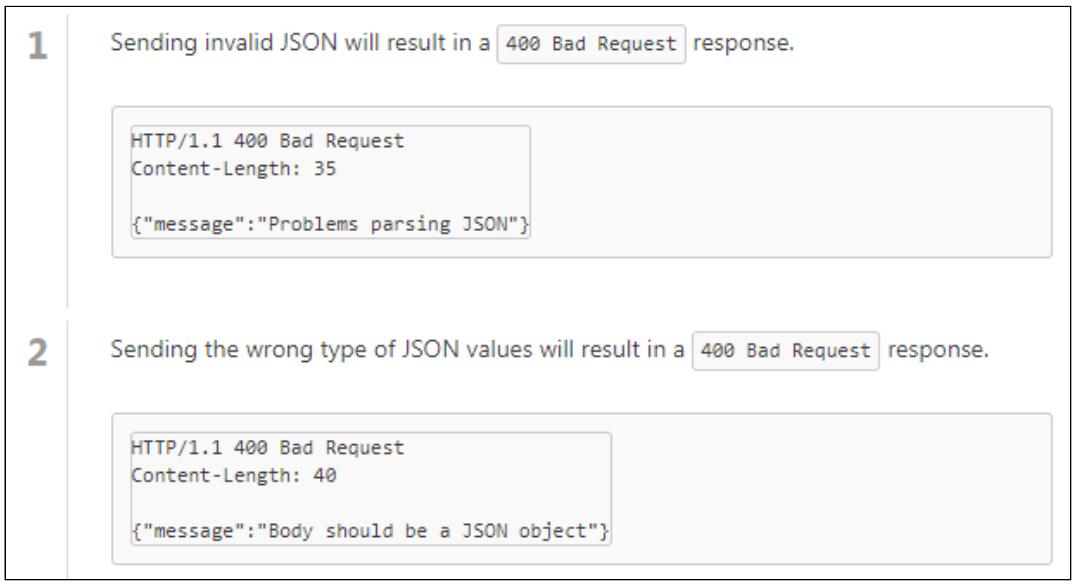

PRG 패턴 적용 이후

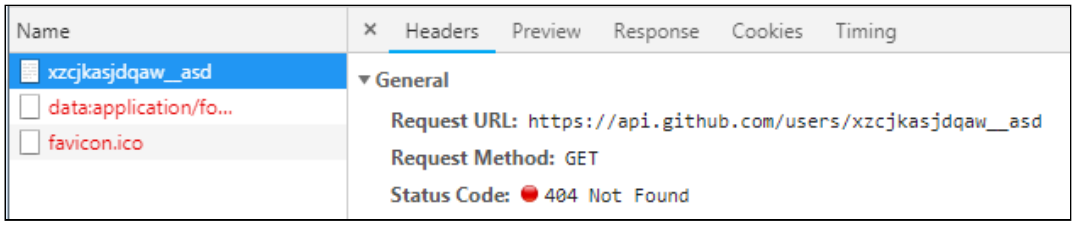



- **304 Not Modified**
  - 요청한 문서의 **수정 여부를 질의**했을 때 수정되지 않은 경우(즉, **캐시된 데이터를 써도 무방**한 경우) 해당 코드를 리턴
  - ajax 요청을 보낼때에는 304 응답 코드를 확인할 수 없음, 내부적으로 **브라우저에서 200 코드로 처리**하여 해당 응답 코드만 확인 가능

        - https://stackoverflow.com/questions/49751311/are-browsers-supposed-to-handle-304-responses-automagically

## 4XX (실패 : 클라이언트 오류)

- **400 Bad Request**
    - 잘못된 요청 (주로 **요청을 쓰는 방식**이 잘못되었을 때)
        - 요청을 처리하는데 필요한 정보를 전달하지 않았다던가, 정보는 전달했지만 형식(숫자를 써야되는데 문자열을 보낸다던지)이 잘못되었다던가 등등
    - 응답 바디에 어떻게 요청을 하면 되는지에 대한 설명 방식을 전달할수도 있음
    - ex) GitHub API
    - 

- 401 Unauthorized
    - 허가되지 않은 요청
    - **인증(authorization)**이 필요한 리소스에 접근에 인증되지 않은 상태로 접근
- 403 Forbidden
    - 숨겨진 리소스라서 제공 불가 (정보에 접근할 **권한(permission)** 필요)
    - 401과 403 응답 코드의 차이
- **404 Not Found**
    - 해당 요청에 대한 리소스(주로 웹 페이지)가 존재하지 않음
    - 꼭 웹 페이지가 존재하지 않을 때만 사용해야 하는 코드는 아니며 모든 자료(ex: json, xml)를 요청했을 때 해당 자료가 없으면 404 코드 반환 가능
    - 

- **405 Method Not Allowed**
    - **허용하지 않는 HTTP 메소드를 이용**하여 요청 (GET 요청만 받는 핸들러에 POST 요청 보내기 상황)
- 406 Not Acceptable
    - 내용 협상(Content Negotiation)에 실패하여 반환할 적당한 리소스가 없음
        - 잘 사용되지 않음 (오류 코드를 보는 것보다는 나으니 차라리 그냥 대안값을 줘버림)

    - > In practice, this error is very rarely used. Instead of responding using this error code, which would be cryptic for the end user and difficult to fix, servers ignore the relevant header and serve an actual page to the user. It is assumed that even if the user won't be completely happy, they will prefer this to an error code.

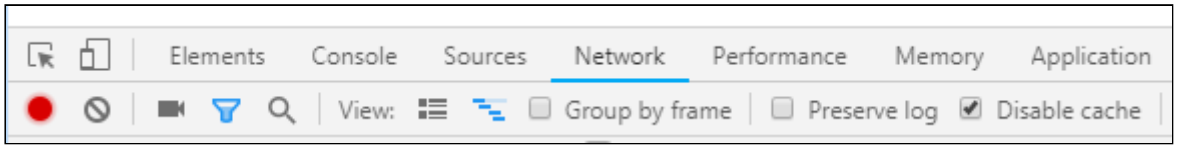
- 414 URI Too Long
    - 요청 메시지에 포함된 주소(Path)가 너무 길 경우 발생 (보통 서버는 8KB까지 주소 정보를 받을 수 있도록 설정)

## 5XX (실패 : 서버 오류)

- **500 Internal Server Error**
    - 서버 내부에서 발생한 오류(자바 예외 발생 등)로 인하여 요청 처리 불가
- 503 Service Unavailable
    - 일시적인 부하(혹은 예정된 점검) 등 예기치 않은 이유로 **일시적** 서비스 불가

# 코드 예제 살펴보기

개발자 툴에서 캐시를 사용하지 않도록 Network 탭의 Disable cache 체크




## 2XX

```
@RestController
public class Code2XX {
    private Integer percent = 0;
    private Integer id = 1;

    @RequestMapping("/c200")
    public String c200(HttpServletResponse response) {
        response.setStatus(200);

        return "OK";
    }

    @RequestMapping("/c201")
    public String c201(HttpServletResponse response) {
        response.setStatus(201);
        // 3XX 코드가 아니므로 브라우저 상 리다이렉션은 일어나지 않으나 응답 헤더에는 포함
됨
        response.setHeader("Location", "http://www.naver.com");

        return "Accepted";
    }
```

```
    @RequestMapping("/c202")
    public void c202(HttpServletResponse response) throws IOException {
        response.setStatus(202);

        // setContentType 메소드 내부적으로는  setHeader 메소드를 호출함
        //
https://alvinalexander.com/java/jwarehouse/eclipse/org.eclipse.equinox.http/src/org/eclipse/equinox/http/servlet/HttpServletResponseImpl.java.shtml
        response.setContentType("application/json");
        // response.setCharacterEncoding("utf-8");

        response.setHeader("Content-Type", "application/json; charset=utf-8");
        // response.setContentLength(1); // 이렇게 썼을 경우 달라지는 결과 확인해보기

        response.getWriter().append("{ \"url\": \"/async_job/" + id + "\" }");
        id++;

        // 뭔가 오래 걸리는 작업 진행
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    while(true) {
                        Thread.sleep(1000);
                        percent += 10;
                        System.out.println("percent : " + percent);
                        if(percent == 100) break;
                    }
                } catch (InterruptedException e) {}
            }
        }).start();

        return;
    }

    @RequestMapping("/async_job/{id}")
    public String async_job(@PathVariable Integer id) {
        String ret = percent + "%";

        if(percent == 100) {
            percent = 0;
            ret = id + " 작업 완료";
        }

        return ret;
    }
}
```

## 3XX

```
@RequestMapping("/c301")
public String c301(HttpServletResponse response) {
    response.setStatus(301);
    response.setHeader("Location", "http://www.naver.com");
```

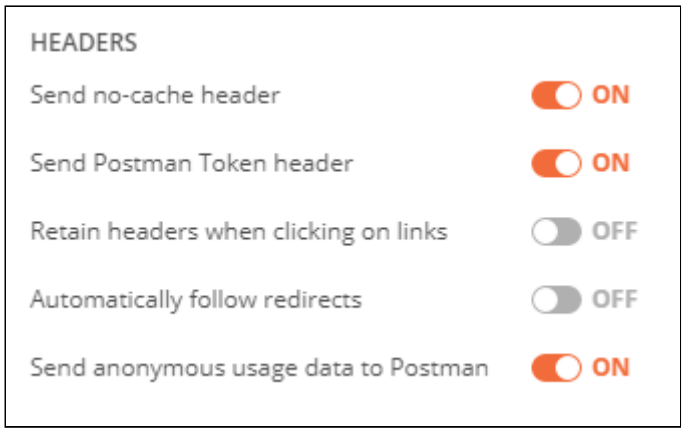
```
    // 일반적으로 redirect 응답에는 바디를 포함하지 않음 (그러나 금지는 아님)
    // https://stackoverflow.com/questions/8059662/http-302-redirect-is-a-
message-body-needed
    return "Moved Permanently";
}

// 리다이렉트를 수행하는 다양한 방법들

// sendRedirect 메소드 호출하여 리다이렉트
/*
@RequestMapping("/c302")
public void c302(HttpServletResponse response) throws IOException {
    response.sendRedirect("http://www.naver.com");
}
*/

// RedirectView 객체를 이용
/*
@RequestMapping("/c302")
public RedirectView c302() {
    return new RedirectView("http://www.naver.com");
}
*/

// ModelAndView의 뷰 이름 앞에 redirect:를 붙이고 뒤에 URL를 써줌
/*
@RequestMapping("/c302")
public ModelAndView c302() {
    return new ModelAndView("redirect:http://www.naver.com");
}
*/
```

- POSTMAN에서 테스트 시 자동으로 리다이렉트 된 곳의 요청이 응답으로 돌아오므로 **[Files-Settings]에서 옵션을 꺼주어야 리다이렉트 응답을 확인** 가능
  - 


## 4XX

```
@RestController
public class Code4XX {
    @RequestMapping("/c400")
    public void c400(HttpServletRequest request, HttpServletResponse response)
throws IOException {
        String secret = request.getParameter("secret");
```

```
        if(secret != null && secret.equals("1234")) {
            response.setStatus(200);
            response.setContentType("text/plain");
            // response.setContentType("text/html");
            response.setCharacterEncoding("utf-8");

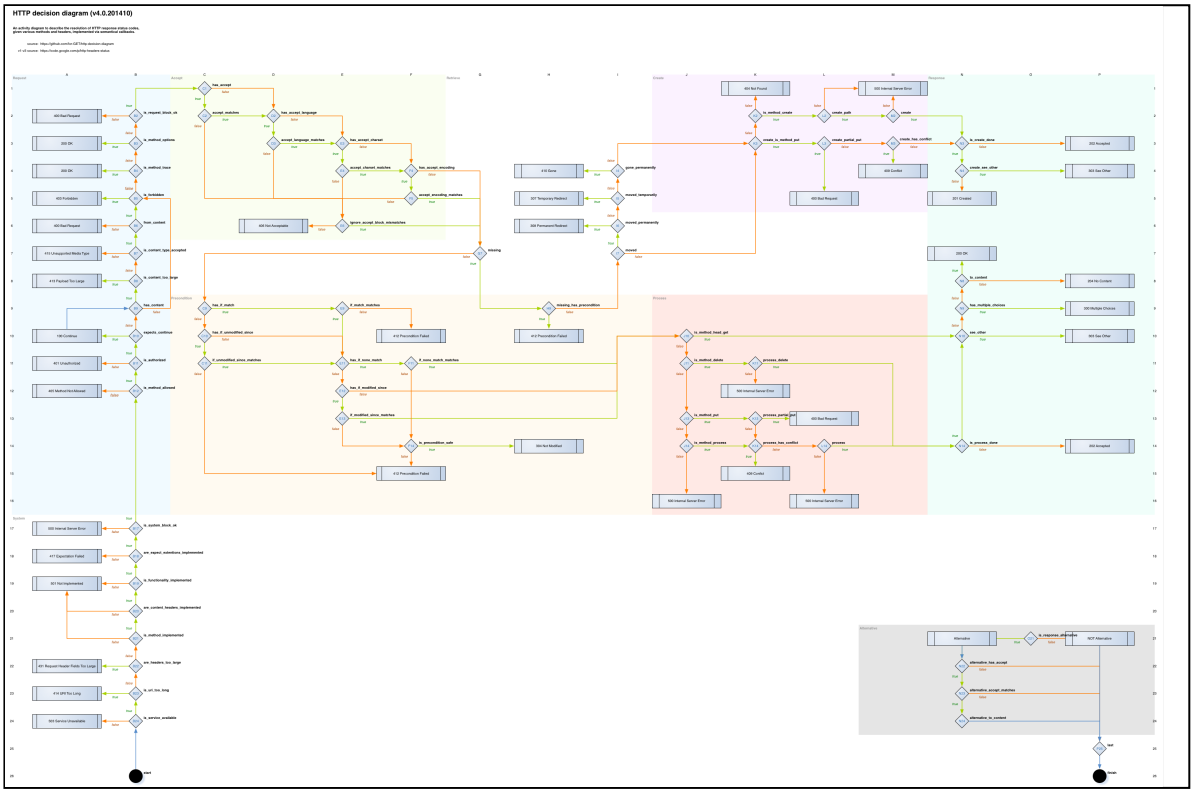
            // HTML 태그로 인식하지 않음? 왜?
            response.getWriter().append("<h1>Perfect!</h1>");
        } else {
            response.setStatus(400);
            response.setContentType("application/xml");
            response.setCharacterEncoding("utf-8");

            // 에러 해결 방법에 대한 메시지 전달
            response.getWriter().append("<?xml version=\"1.0\" encoding=\"utf-
8\" ?><cause>You missed a secret. (hint : 1234)</cause>");
        }

        return;
    }
}
```

- 404및 5XX 에러는 이후 검증, 에러 처리 및 예외 처리 부분에서 확인

### 응답 코드 설정 관련 순서도 다이어그램




# 레퍼런스

- https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview
- http://www.steves-internet-guide.com/http-basics/

- https://stackoverflow.com/questions/978061/http-get-with-request-body
- https://stackoverflow.com/questions/28459418/rest-api-put-vs-patch-with-real-life-examples
- https://restfulapi.net/idempotent-rest-apis/
- https://stackoverflow.com/questions/1077412/what-is-an-idempotent-operation
- https://stackoverflow.com/questions/45016234/what-is-idempotency-in-http-methods
- https://stackoverflow.com/questions/4088350/is-rest-delete-really-idempotent
- http://www.steves-internet-guide.com/http-headers/
- https://www.baeldung.com/spring-response-status
- https://ec.haxx.se/http/http-multipart
- https://stackoverflow.com/questions/26723467/what-is-the-difference-between-form-data-x-www-form-urlencoded-and-raw-in-the-p
- https://fideloper.com/api-etag-conditional-get
- https://www.logicbig.com/quick-info/web/etag-header.html
    - https://stackoverflow.com/questions/4533/http-generating-etag-headert
    - https://stackoverflow.com/questions/56663203/etag-weak-vs-strong-example
- https://dev.to/sidthesloth92/understanding-html-form-encoding-url-encoded-and-multipart-forms-3lpa
- https://www.adayinthelifeof.nl/2011/06/02/asynchronous-operations-in-rest/
- https://http2-explained.haxx.se/ko
- https://kinsta.com/learn/what-is-http2/
- https://developers.google.com/web/fundamentals/performance/http2
- https://stackoverflow.com/questions/3561381/custom-http-headers-naming-conventions
- https://www.smashingmagazine.com/2017/11/understanding-vary-header/