SpringMvc1 : 섹션7. 스터디 정리 자료

💡 목 차

PRG : Post Redirect Get

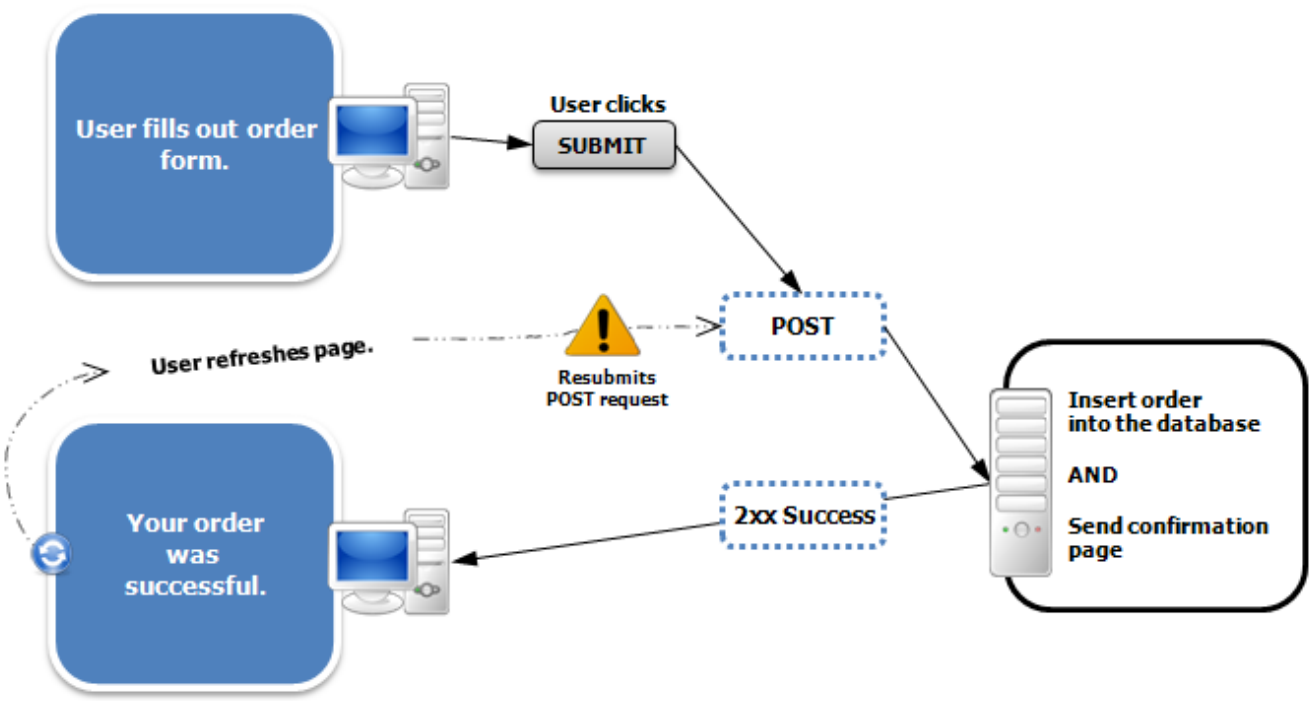
정의

- Post → Redirect → Get

> **Post/Redirect/Get** (**PRG**)은 Form 제출 후 표시되는 페이지를 나중에 양식을 제출하는 등의 부작용 없이 다시 로드, 공유 또는 북마크할 수 있도록 하는 웹 개발 디자인 패턴이다.
> (출처 :
> wikipedia)

문제점

- PRG 패턴이 적용되지 않았을 때 생기는 문제점

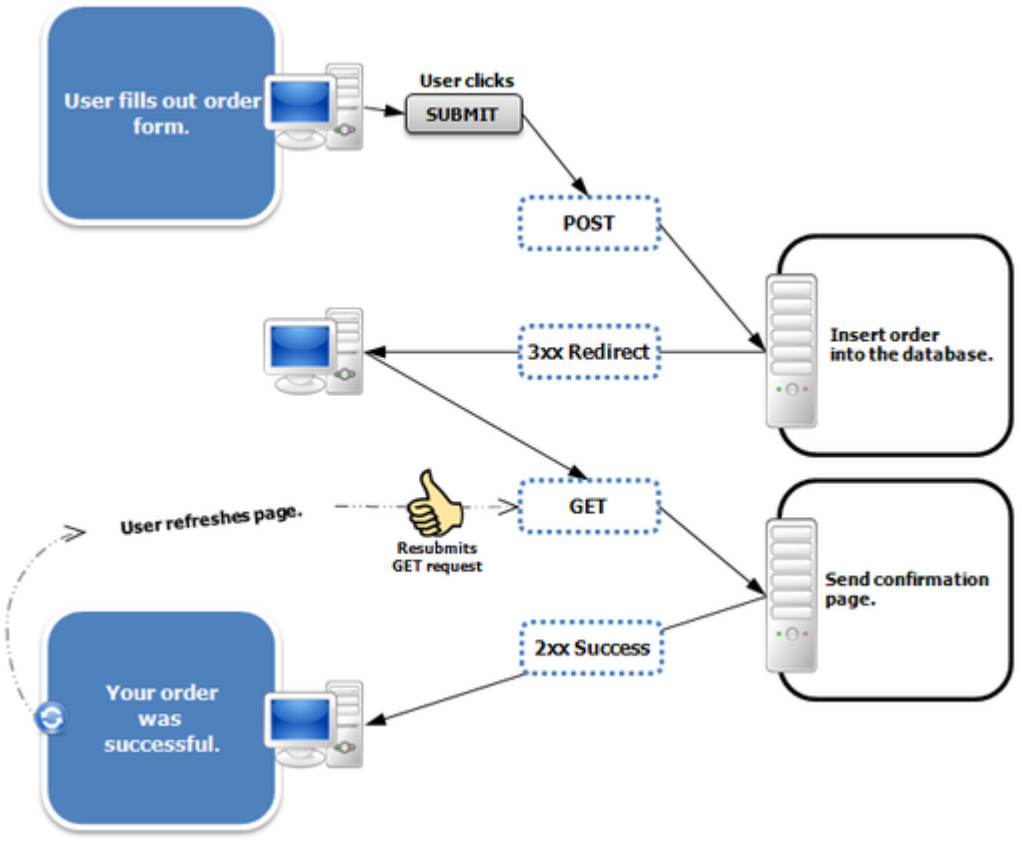


    - 클라이언트가 Form 양식으로 서버에 POST 메서드를 전송 했을 때, 서버에서는 DB에 사용자의 입력 처리를 저장하는 작업이 발생하게 된다.

- DB 저장을 발생시키는 POST 요청 → DB 저장 → 200 Success Page 클라이언트에게 반환 → **클라이언트 브라우저 Refresh → 해당 POST 재요청**
- 보통 POST 요청의 결과로 DB 쓰기 작업 (상품 주문 및 결제, 회원 가입, 게시글 작성) 등이 발생하게 된다. 결과적으로 브라우저의 Refresh가 마지막 HTTP 요청을 반복하기 때문에 PK(RID)만 변경되어 **무수한 중복 데이터를 양산**할 수 있다.

해결법

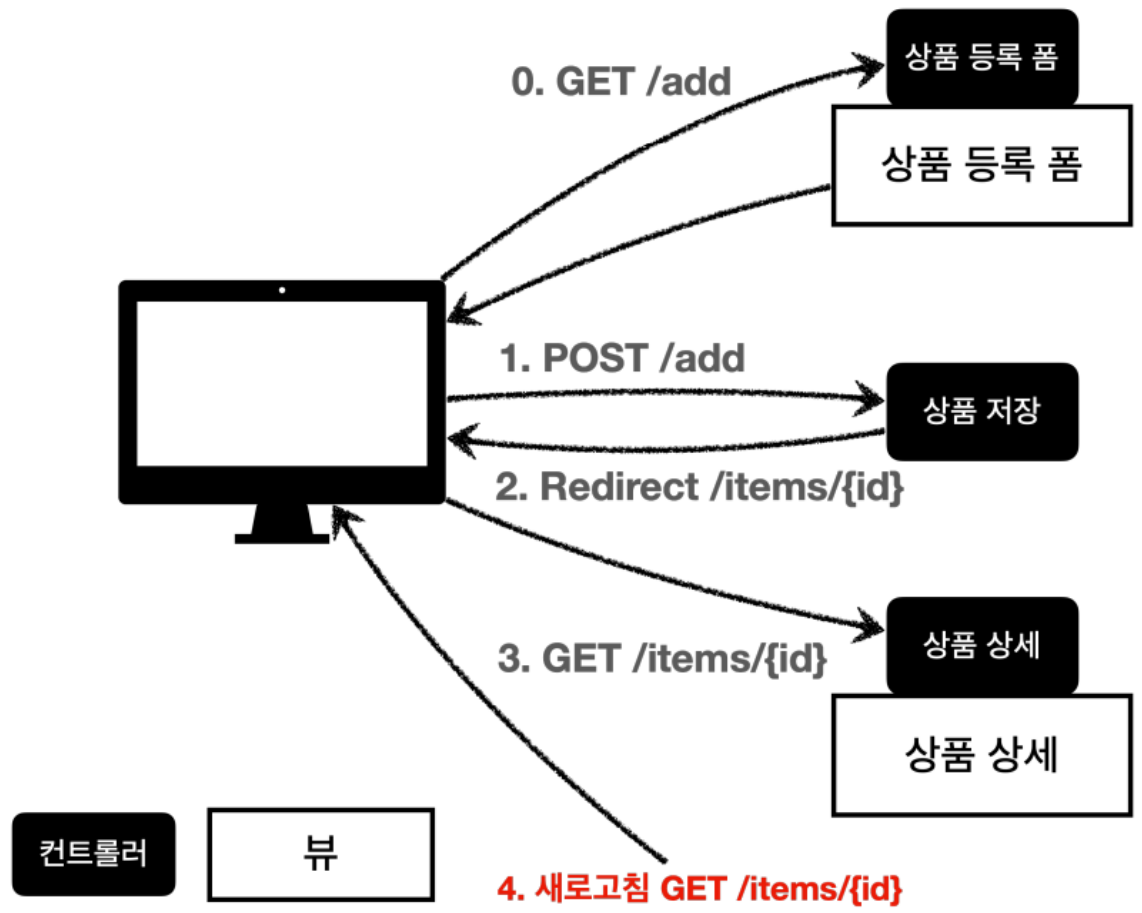
- 위와 같은 문제를 피하기 위해 많은 웹 개발자는 **콘텐츠로 응답하는 대신 서버가 클라이언트를 다른 위치로 리디렉션하여 POST 요청에 응답하는 PRG 패턴**을 사용한다.



- POST 성공의 결과로 3xx Redirect를 반환하여 다른 페이지로 GET 요청을 전송하는 것이다.
- 브라우저에서 새로고침(Refresh)을 시도해도 마지막 GET 요청만이 반복되며 DB 쓰기 작업(POST)가 발생하지 않는다.

구체적인 코드 예시

- 웹 브라우저의 새로 고침은 **마지막에 서버에 전송한 데이터를 다시 전송**한다.
- **새로 고침 문제를 해결**하려면 상품 저장 후에 뷰 템플릿으로 이동하는 것이 아니라, **상품 상세 화면으로 리다이렉트를 호출**해주면 된다.



- 기존의 상품 등록

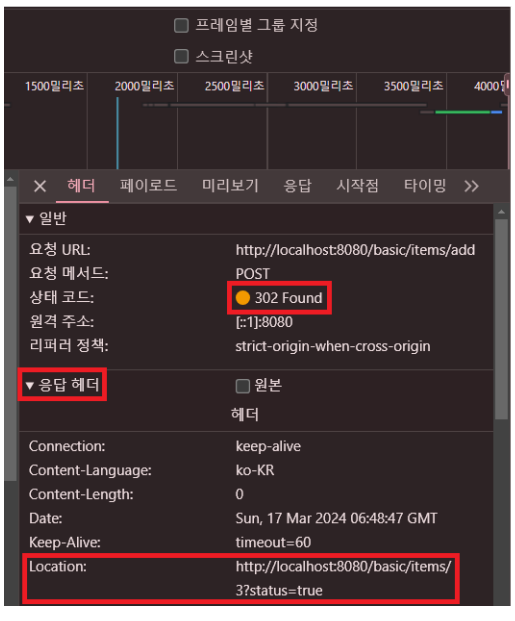
```
@PostMapping("/add")
public String addItemV4(Item item) {
    itemRepository.save(item);
    return "basic/item";
}
```

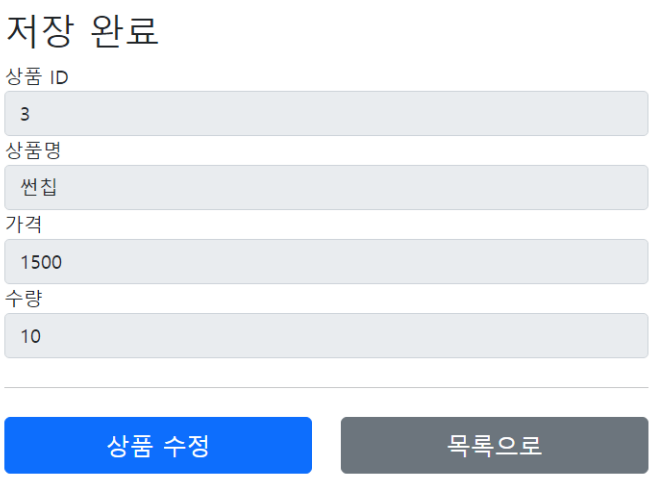
  - 상품 등록 (인메모리 DB에 저장) 이후 상품 상세 페이지로 포워딩.
- **PRG 패턴을 적용한 상품 등록**

```
/**
 * PRG - Post/Redirect/Get
 */
@PostMapping("/add")
public String addItemV5(Item item) {
    itemRepository.save(item);
    return "redirect:/basic/items/" + item.getId();
}
```

- 상품 등록 이후 뷰 템플릿이 아닌 **상품 상세 화면으로 리다이렉트 하도록** 코드를 작성
- 스프링은 `redirect:/...` 기능으로 리다이렉트를 지원한다.
- `redirect:/basic/items/{itemId}`
  - 컨트롤러에 매핑된 `@PathVariable` 의 값은 `redirect` 에도 사용 할 수 있다.
  - `redirect:/basic/items/{itemId}` → `{itemId}` 는 `@PathVariable Long itemId` 의 값을 그대로 사용한다.

- 브라우저의 응답 헤더 `location` 에 해당 상품의 상세 페이지 URI가 응답되며, 302 리다이렉션 되는 것을 확인할 수 있다.





정리

- 웹 API 호출에서 POST 요청은 **멱등성(idempotency)**을 가지지 않아 이에 대한 추가 요구사항을 필요로 한다.
  - 멱등성이란 '연산을 여러번 적용하더라도 결과가 달라지지 않는 성질'을 의미한다.
    - GET 요청은 여러 번 요청하더라도 항상 같은 결과를 조회한다.
    - POST 요청은 대부분의 경우에 새로운 연산이 새로운 자원(URI)을 생성하므로 멱등성을 가지지 못한다.
  - POST 연산이 멱등성을 가지지 못하기 때문에 이를 보상할 수 있는 디자인 패턴이 요구된다.
- PRG 패턴은 POST 요청 직후 GET 방식으로 REDIRECT 하는 것으로 중복된 쓰기 작업을 발생시키지 않는 디자인 패턴이다.
  - POST → REDIRECT → GET
- 💡**POST 방식의 처리**는 **가능하면 빨리 다른 페이지를 보도록 브라우저 화면을 이동**시키는 것이 좋다.

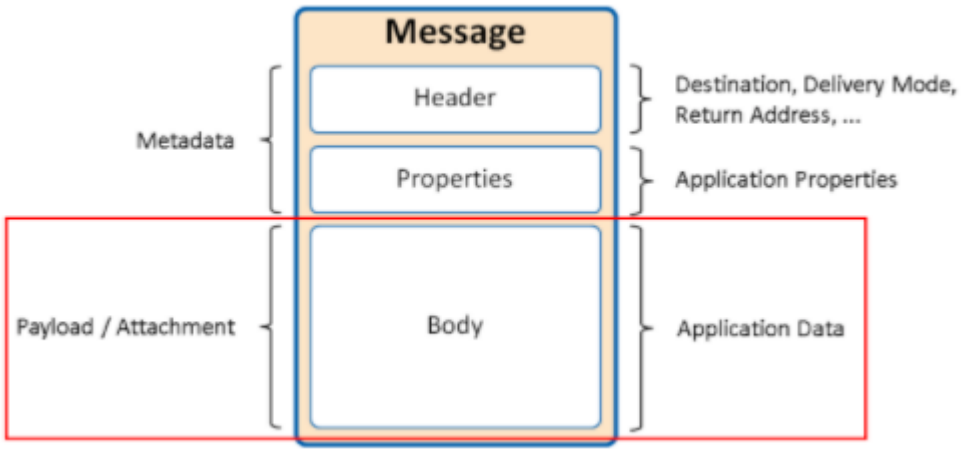
멱등성과 Http Method

HTTP 요청 메서드 유형

- `GET` **(Read)**

    - GET 메소드는 지정된 **자원의 표현을 요청**합니다. GET을 사용하는 요청은 데이터만 검색해야 합니다.
- `HEAD`
    - HEAD 메소드는 GET 요청과 동일하지만 **응답 본문이 없는 응답을 요청**합니다.
- `POST` **(Create)**
    - **POST 메서드는 엔터티를 지정된 리소스에 제출**하며, 종종 서버의 상태 변경이나 부작용을 유발합니다.
    - **멱등성(Idempotent)** 이 존재하지 않습니다.
- `PUT` **(Create / Replace)**
    - PUT 메서드는 **대상 리소스의 모든 현재 표현을 요청 페이로드(payload)로 바꿉니다.**
        - HTTP 요청을 보낼 때 **전달되는 데이터 (그 자체)**를 payload라고 합니다.

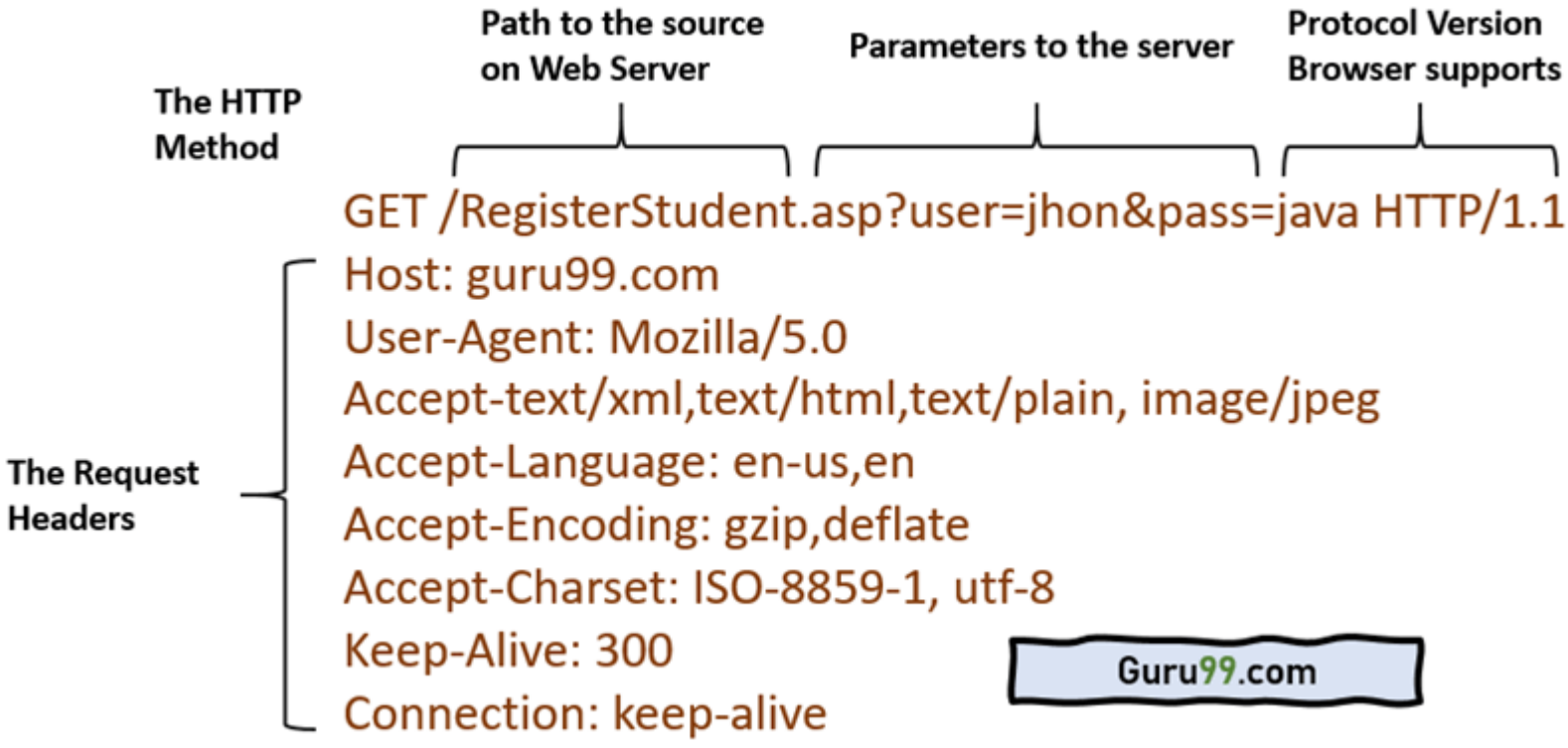
▼ 페이로드 도식화 (출처)



▼ 페이로드 코드

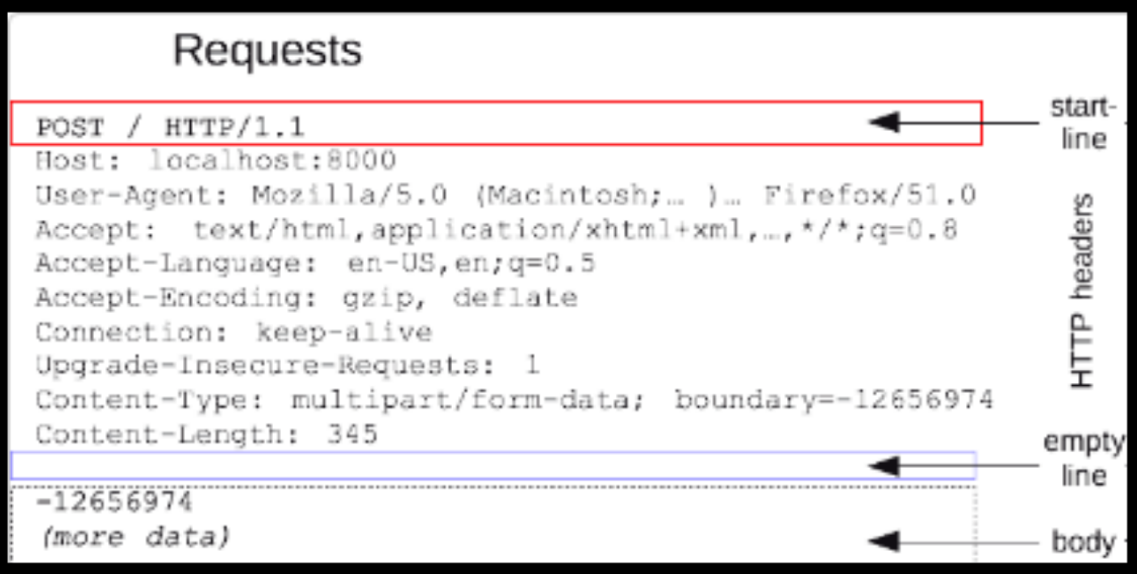
```
{
  "status" : "OK"
  "from": "localhost",
  "to": "https://hanamon.kr/users/1",
  "method": "GET",
  "data":{ "username" : "하나몬" } // data만 페이로드
}
```

▼ HTTP 메시지의 예시 (출처) - GET에는 payload가 없다.



▼ 반면 POST에는 payload가 존재한다.

HTTP 요청 메시지



출처: MDN - HTTP 메시지

- `PATCH` **(Update)**
  - PATCH 메서드는 리소스에 **부분 수정**을 적용합니다.
- `DELETE` **(Delete)**
  - DELETE 메소드는 지정된 리소스를 **삭제**합니다.

멱등성(Idempotent)이란?

- 동일한 요청을 한 번 보내는 것과 여러 번 연속으로 보내는 것이 같은 효과를 지니고, 서버의 상태도 동일하게 남을 때, 해당 HTTP 메서드가 **멱등성**을 가졌다고 말합니다.
- HTTP 메서드는 멱등성 속성과 안전 속성으로 분류할 수 있습니다.
  - 안전한 메서드는 리소스를 수정하지 않는 HTTP 메서드입니다. 예를 들어 리소스 URL에서 GET 또는 HEAD를 사용하면 절대로 리소스를 변경해서는 안 됩니다.
  - 멱등성 HTTP 메서드는 다른 결과 없이 여러 번 호출될 수 있는 HTTP 메서드입니다. 메소드가 한 번만 호출되는지, 아니면 10번 이상 호출되는지는 중요하지 않습니다.
- 다음은 멱등성과 안전성을 기준으로 HTTP 메서드를 요약한 표입니다.

HTTP Method	Idempotent	Safe
OPTIONS	yes	yes
GET	yes	yes
HEAD	yes	yes
PUT	yes	no
POST	no	no
DELETE	yes	no
PATCH	no	no

▼ **멱등성과 안전성 (HTTP Methods Properties)** (출처)

- **안전성(safety)**: 요청 시 정보 검색 이외의 작업을 실행하려는 의도가 없는 경우 메서드는 안전합니다. 메소드 처리로 인해 서버에 부작용이 없는 것으로 안전하다고 인식할 수 있습니다. 그러나 어쨌든 부작용이 발생한다면 이는 사용자의 요구가 아닌 결과일 뿐입니

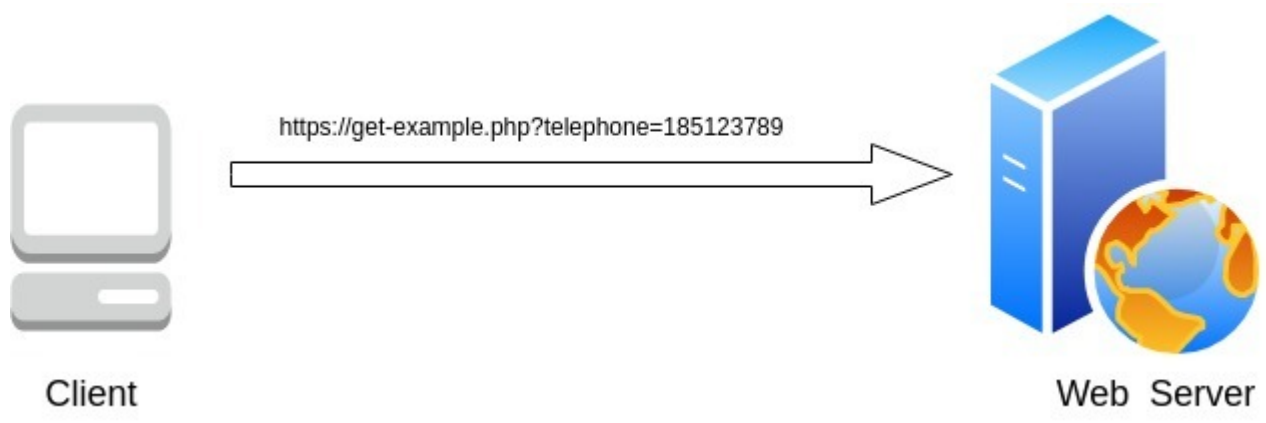
다. 관례적으로 OPTIONS, HEAD 및 GET은 안전한 방법입니다.

- **멱등성(Idempotency)**: 이 속성은 동일한 요청을 여러 번 처리할 때 서버에 미치는 부작용이 요청을 한 번 처리하는 것과 동일함을 의미합니다. 따라서 GET, HEAD, PUT, DELETE, OPTIONS 및 TRACE는 멱등성 메서드입니다.

GET과 POST 요청의 차이

GET

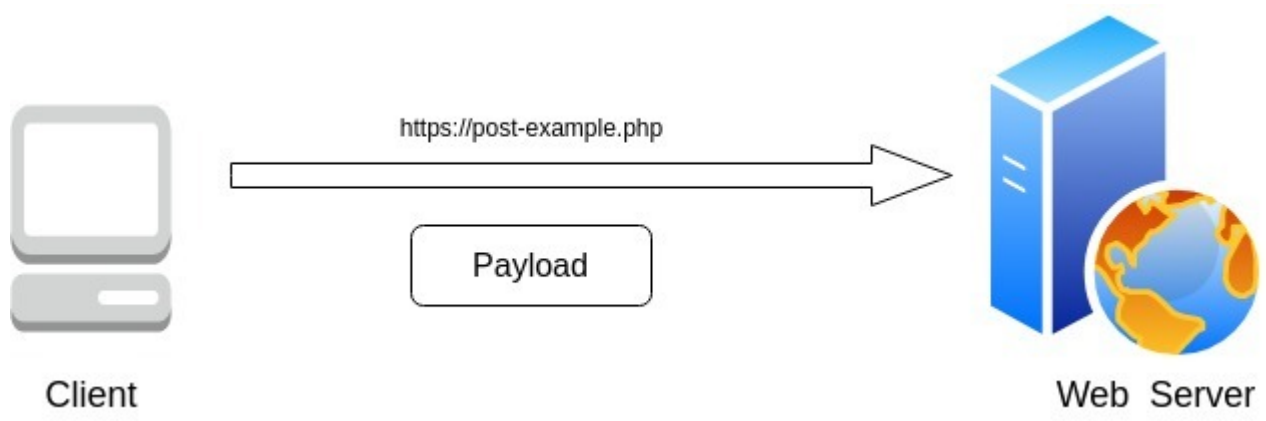
- GET은 **지정된 리소스에서 데이터를 요청**하는 데 사용됩니다. HTML 문서, 이미지, 비디오 등 클라이언트에 표시되는 모든 데이터를 검색할 수 있습니다.



GET 요청을 보내려면 클라이언트는 검색하려는 리소스의 URL을 지정해야 합니다. 그런 다음 요청은 서버로 전송되고, 서버는 요청을 처리하고 요청된 데이터를 클라이언트로 다시 보냅니다.

POST

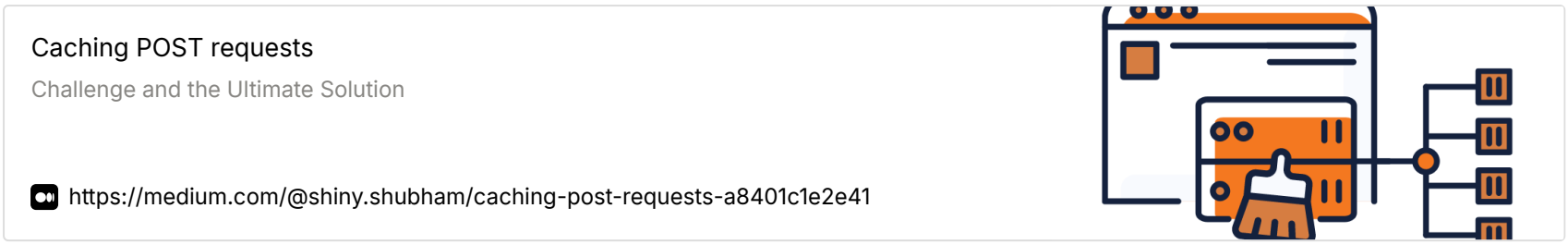
- POST는 **리소스를 생성하거나 업데이트**하기 위해 서버로 데이터를 보냅니다. 예를 들어 HTML 양식을 서버에 제출하는 데 자주 사용됩니다.

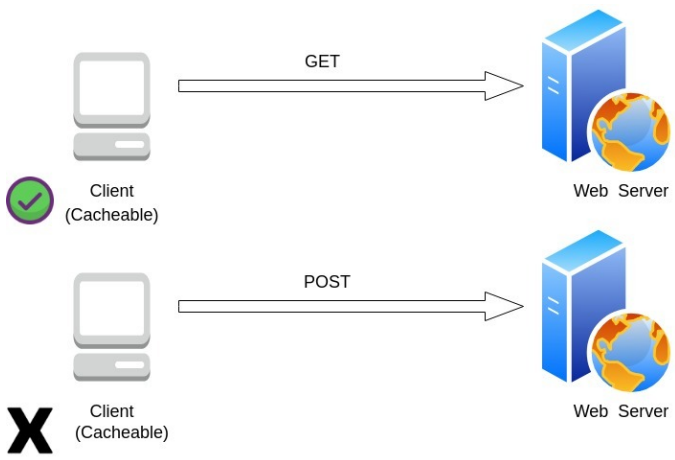


GET과 POST 요청의 차이

- **Visibility (가시성)**
  - GET을 사용하면 데이터 매개변수가 URL에 포함되어 모든 사람에게 표시됩니다. 그러나 POST를 사용하면 데이터가 URL에 표시되지 않고 HTTP 메시지 본문에 표시됩니다.
- **Security (보안)**
  - GET은 전송된 데이터의 일부가 URL에 포함되어 있기 때문에 보안 수준이 낮습니다. 반면에 POST는 매개변수가 웹 서버 로그나 브라우저 기록에 저장되지 않기 때문에 더 안전합니다.
- **Cache (캐시 가능 여부)**
  - GET 요청은 캐시되어 브라우저 기록에 남을 수 있지만 POST 요청은 캐시될 수 없습니다. 즉, GET 요청은 북마크에 추가하고, 공유하고, 다시 방문할 수 있지만 POST 요청은 다음을 수행할 수 없습니다.

    ▼ 단, 특별한 경우에 POST 요청을 캐싱하고 싶다면 Cache-Control 헤더에 적절한 지시어를 추가할 수 있다.





- **Server State (서버 상태)**
  - GET 요청은 서버에서 데이터를 검색하기 위한 것이며 서버 상태를 수정하지 않습니다. 반면에 POST 요청은 처리를 위해 서버에 데이터를 보내는 데 사용되며 서버 상태를 수정할 수 있습니다.
- **Amount of Data Transmitted (전송 데이터의 양)**
  - GET 요청은 최대 문자 수로 제한되는 반면 POST 요청에는 그러한 제한이 없습니다.
    GET 방식은 길이 제한이 있는 리소스 URL을 통해 데이터를 전송하는 반면, POST 방식은 길이 제한이 없는 HTTP 메시지 본문을 통해 데이터를 전송하기 때문입니다.
- **Data Type (데이터 형식)**
  - GET 메소드는 문자열 데이터 유형만 지원하는 반면,
    POST 메소드는 문자열, 숫자, 바이너리 등과 같은 다양한 데이터 유형을 지원합니다.
- **Idempotent (멱등성)**
  - `GET /pageX HTTP/1.1`는 안전한(읽기 전용) 메서드이기 때문에 멱등성을 가집니다. 그 동안 서버의 데이터가 업데이트된 경우, 다음에 하는 호출은 클라이언트에 다른 데이터를 반환할 수 있습니다.
  - `POST /add_row HTTP/1.1`은 멱등성을 가지지 않습니다. 여러 번 호출되면 여러 행을 추가합니다.
  - (참고) `DELETE /idX/delete HTTP/1.1`의 상태 코드는 응답마다 달라질 수 있지만, 그럼에도 멱등성을 가집니다.