

# **WEB 101**

basic web concept for webhack'ng

beta@plus

# Contents

Concept

HTTP

Server

Tools

# Chapter 1.

**Concept**

**HTTP**

**Server**

**Tools**

# **두 컴퓨터를 통신하게 하고 싶다!**

두 컴퓨터가 서로 메시지나 데이터를 주고 받을 수는 없을까?



클라이언트가 서버에 데이터를 요청하면,  
서버가 데이터를 보내주는 구조!



# Static

**정적인 구조** 서버가 수동적임 - 요청이 와야만 응답하는 구조!

# Web에서 주로 주고 받는 파일



**HTML**

웹페이지의 기본 뼈대

프로그래밍 언어 아님



**JS**

웹페이지의 엔진

많은 일을 하는 핵심 PL



**CSS**

웹페이지의 장식

베타가 좋아함

# HTML

```
<html>
  <head>
    <title>beta</title>
  </head>
  <body>
    <div>I'm beta!</div>
    <button onclick="heart()">love</button>
  </body>
</html>
```

## Element

HTML을 이루는 블럭 단위

<body> ... </body>

<div> ... </div>,

## Attribute

Element의 속성

onclick, src, href, ...

# JavaScript

```
function heart() {  
    alert("love you!");  
}  
  
const hack = () => {  
    alert(document.cookies);  
}
```

## PL

브라우저의 제약 안에서,  
프로그래밍 할 수 있는 언어!

## for Web

웹과 관련된 각종 속성을  
불러오거나 이용할 수 있다!

# CSS

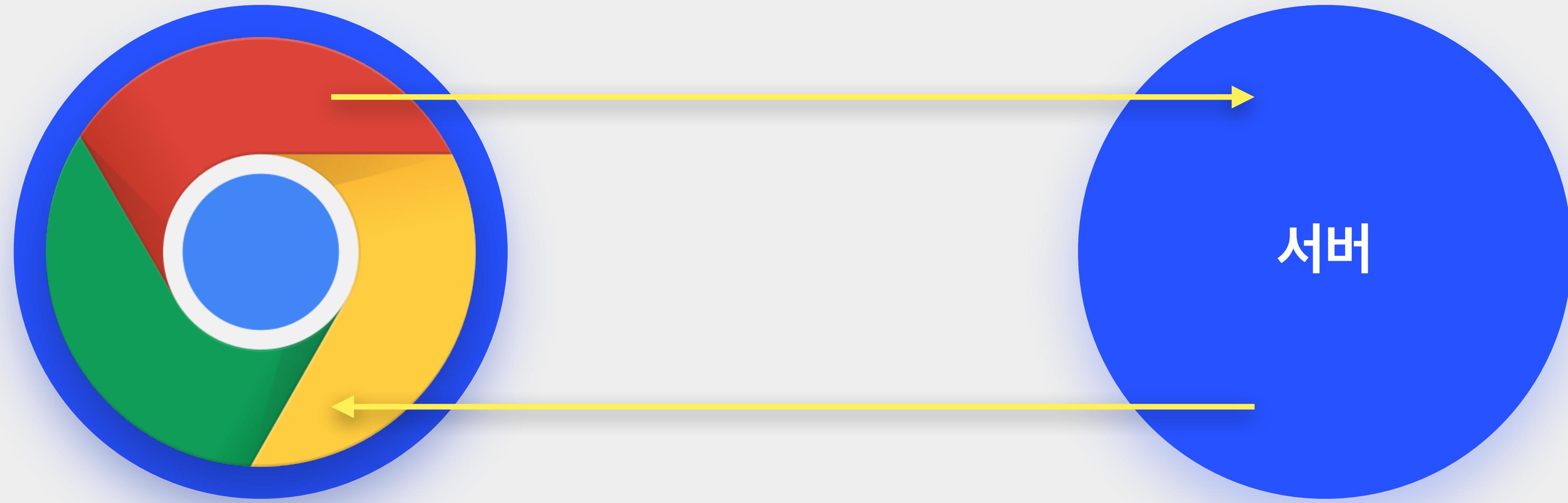
```
body {  
    background: black;  
    width: 100vw; height: 100vh;  
    margin: 0;  
    display: flex;  
    justify-content: center;  
    align-items: center;  
    flex-direction: column;  
}  
  
button {  
    background: white;  
    color: black;  
    border-radius: 20px;  
    padding: 10px; margin: 10px;  
    border: none;  
    width: 50vw;  
}  
  
div {  
    color: white;  
}
```

## Useless

정말 제한적인 상황이 아니고서야  
웹해킹에서 이용되지 않음

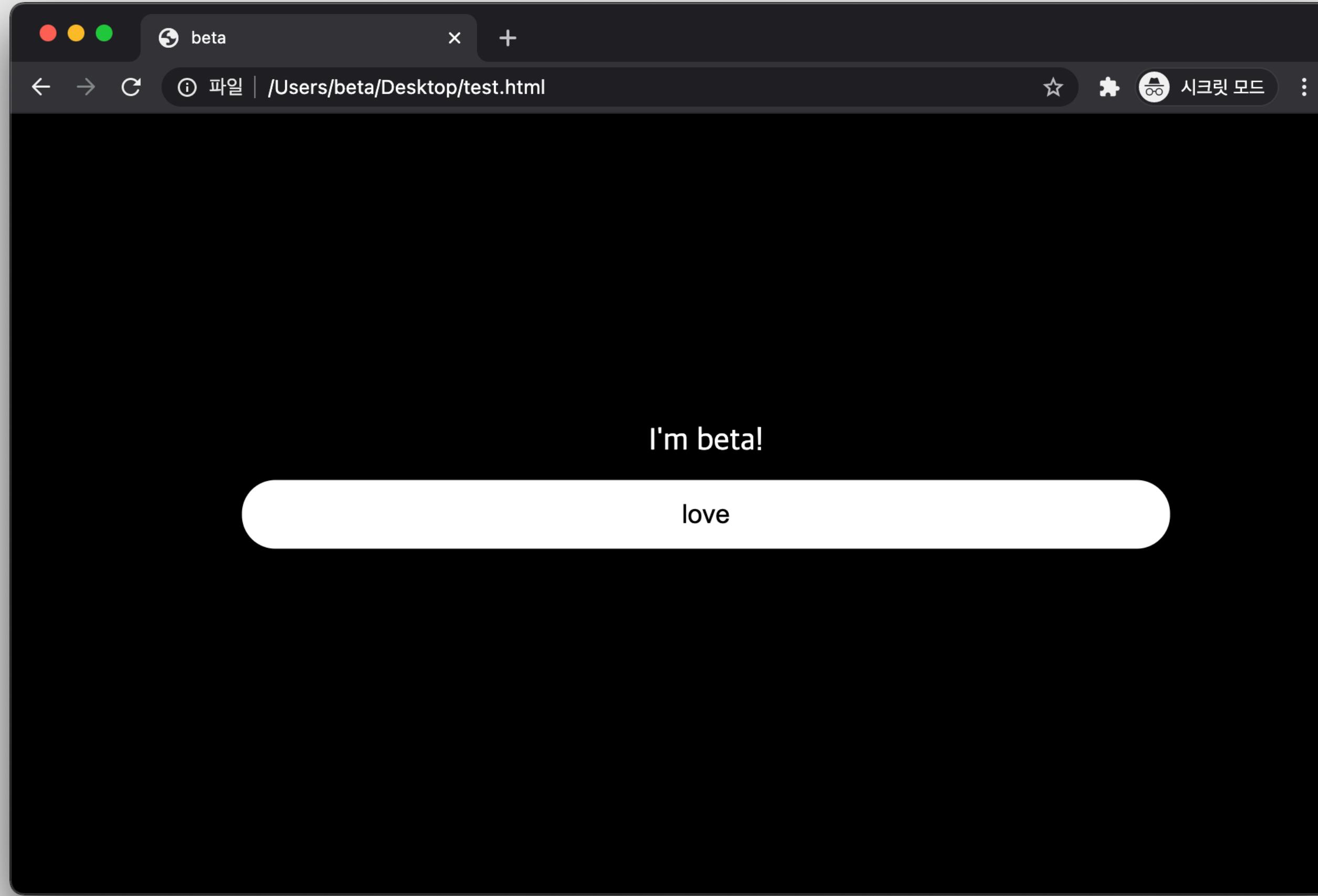
## but,

베타가 좋아함



**브라우저는 HTML을 요청하면서,  
이와 같이 필요한 JS, CSS, Image 등을 서버에 함께 요청한다!**

필요한 파일의 목록은 HTML에 명시되어 있음



받은 파일들을 **브라우저가 잘 버무려서** 보여준다!



# Static

**정적인 구조** 서버가 수동적임 - 요청이 와야만 응답하는 구조!

# 서버-클라이언트 구조로 주고 받는구나!

그런데, 서로 같은 형식으로 통신할 수 있도록 규격을 맞춰야 하지 않을까?

# Chapter 2.

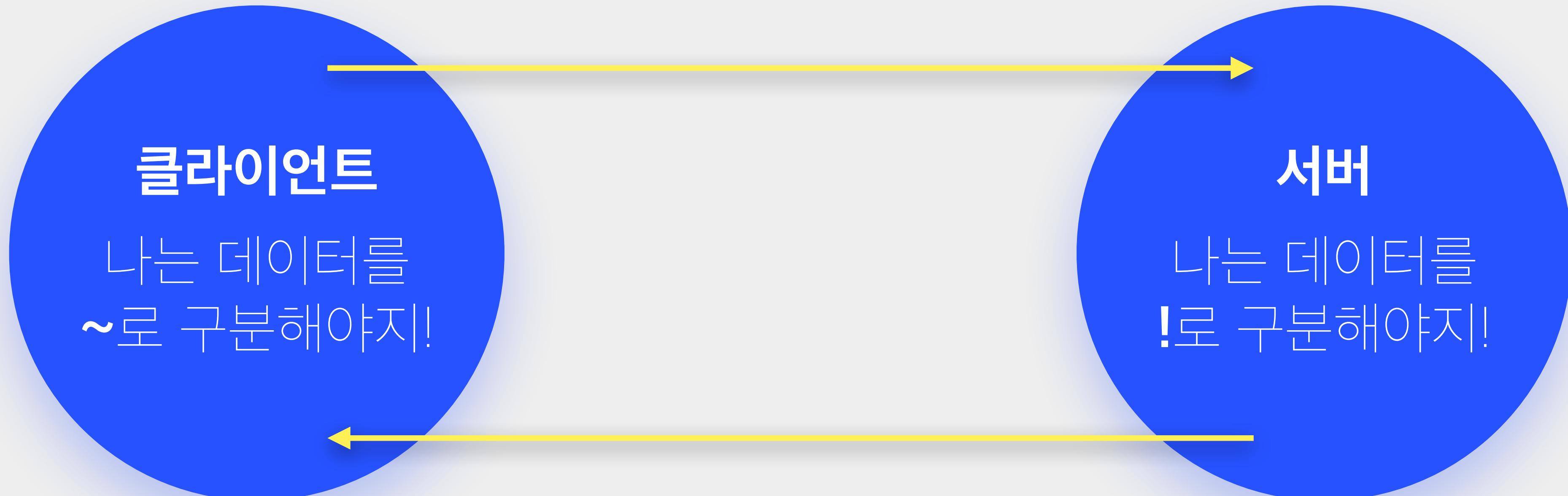
Concept

**HTTP**

Server

Tools

vanilla~ cafe~ latte~



vanilla~ cafe~ latte~

클라이언트

나는 데이터를  
~로 구분해야지!!

서버

나는 데이터를  
!로 구분해야지!!

rock! scissors! paper!

# HTTP

HyperText Transfer Protocol

**Web 상**에서 컴퓨터들이 **데이터를 주고 받는 프로토콜.**  
즉, 일종의 **약속!**

# HTTP Message

서버와 클라이언트가 주고 받는 데이터

# Request

Start Line

Headers

Body

# Response

Status Line

Headers

Body

# Request

Start Line

Headers

Body

**Start Line** HTTP Message의 시작을 나타냄

**GET /beta.html HTTP/1.1**

# Request

Start Line

Headers

Body

## HTTP method

**GET /beta.html HTTP/1.1**

서버가 클라이언트에게 무엇을 해줘야 하는지 명시!

### Example.

GET 리소스를 가져와라!

POST 데이터를 넣어라!

## Request

Start Line

Headers

Body

## Request Target

**GET /beta.html HTTP/1.1**

클라이언트가 요청하는 리소스가 무엇인지 명시!

### Example.

GET /beta.png 사진을 가져와라!

POST /beta.html 이 위치에 데이터 변경을 요청해라!

# Request

Start Line

Headers

Body

## HTTP Version

**GET /beta.html HTTP/1.1**

HTTP의 버전을 명시!

HTTP 버전마다 규격이 약간씩 달라지기 때문에 필요해요

**Example.**

HTTP/1.0, HTTP/1.1, HTTP/2.0, ...

# Request

Start Line

Headers

Body

## Header 통신에 이용될 부가적인 정보

Host: plus.or.kr:1223

User-Agent: Intel Mac OS X 11\_0\_1 Safari/537.36

Accept: text/html

Accept-Language: en-US

Accept-Encoding: gzip

Connection: keep-alive

Upgrade-Insecure-Requests: 1

Content-Type: application/json

Content-Length: 345

# Request

Start Line

Headers

Body

## Header 통신에 이용될 부가적인 정보

Host: plus.or.kr:1223

User-Agent: Intel Mac OS X 11\_0\_1 Safari/537.36

Accept: text/html

Accept-Language: en-US

Accept-Encoding: gzip

Request headers

Connection: keep-alive

General headers

Upgrade-Insecure-Requests: 1

Content-Type: application/json

Representation headers

Content-Length: 345

# Request

Start Line

Headers

Body

# Header

**Host:** plus.or.kr:8000

key

value

## Example.

**Host** 서버의 도메인과 포트를 특정!

**User-Agent** 사용자의 브라우저나 운영체제 정보!

**Cookie** 사용자 쿠키 정보!

# Request

Start Line

Headers

Body

## Body

서버에 요청할 때 함께 보내는 정보

# Request

Start Line

Headers

Body

x-www-form-encoded

id=beta&pw=htmlisnot

json

```
{  
  "id": "beta",  
  "pw": "htmlisnot"  
}
```

# Request

Start Line

Headers

Body

GET /beta.html HTTP/1.1

Host: plus.or.kr:1223

User-Agent: Intel Mac OS X 11\_0\_1 Safari/537.36

Accept: text/html

Accept-Language: en-US

Accept-Encoding: gzip

Connection: keep-alive

Upgrade-Insecure-Requests: 1

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

Content-Length: 20

id=beta&pw=htmlisnot

# Request

Start Line

Headers

Body

# Response

Status Line

Headers

Body

# Response

Status Line

Headers

Body

**Status Line** 요청에 대한 응답의 상태

**HTTP/1.1 200 OK**

# Response

Status Line

Headers

Body

## Protocol Version

**HTTP/1.1 200 OK**

HTTP의 버전을 명시!

HTTP 버전마다 규격이 약간씩 달라지기 때문에 필요해요

**Example.**

HTTP/1.0, HTTP/1.1, HTTP/2.0, ...

# Response

Status Line

Headers

Body

# Status Code

HTTP/1.1 **200** OK

요청의 성공 여부나 상태를 코드로 표현!

## Example.

200 정상!

404 당신이 찾는 페이지는 없어요!

500 웹서버 에러!

# Response

Status Line

Headers

Body

# Status Text

**HTTP/1.1 200 OK**

요청의 성공 여부나 상태를 알아보기 쉽게 표현!

## Example.

OK 정상!

Not Found 당신이 찾는 페이지는 없어요!

Internal Server Error 웹서버 에러!

# Response

Status Line

Headers

Body

## Header 통신에 이용될 부가적인 정보

**Connection:** Keep-Alive

**Date:** Wed, 20 Jan 2021 05:38:31 GMT

**Transfer-Encoding:** chunked

**Server:** nginx

**Set-Cookie:** session=thisissessiontoken

**Content-Encoding:** gzip

**Content-Type:** text/html; charset=utf-8

**Last-Modified:** Wed, 19 Jan 2019 05:16:32 GMT

# Response

Status Line

Headers

Body

## Header 통신에 이용될 부가적인 정보

**Connection:** Keep-Alive

**Date:** Wed, 20 Jan 2021 05:38:31 GMT

**Transfer-Encoding:** chunked Representation headers

**Server:** nginx Response headers

**Set-Cookie:** session=thisissessiontoken

**Content-Encoding:** gzip Representation headers

**Content-Type:** text/html; charset=utf-8

**Last-Modified:** Wed, 19 Jan 2019 05:16:32 GMT

# Response

Status Line

Headers

Body

## Body 요청에 대한 결과물

```
<html>
  <head>
    <title>beta</title>
  </head>
  <body>
    <div>I'm beta!</div>
    <button onclick="heart()">love</button>
  </body>
</html>
```

# Response

Status Line

Headers

Body

HTTP/1.1 200 OK

Connection: Keep-Alive

Date: Wed, 20 Jan 2021 05:38:31 GMT

Transfer-Encoding: chunked

Server: nginx

Set-Cookie: session=thisisessiontoken

Content-Encoding: gzip

Content-Type: text/html; charset=utf-8

Last-Modified: Wed, 19 Jan 2019 05:16:32 GMT

```
<html>
  <head>
    <title>beta</title>
  </head>
  <body>
    <div>I'm beta!</div>
    <button onclick="heart()">love</button>
  </body>
</html>
```

# Request

Start Line

Headers

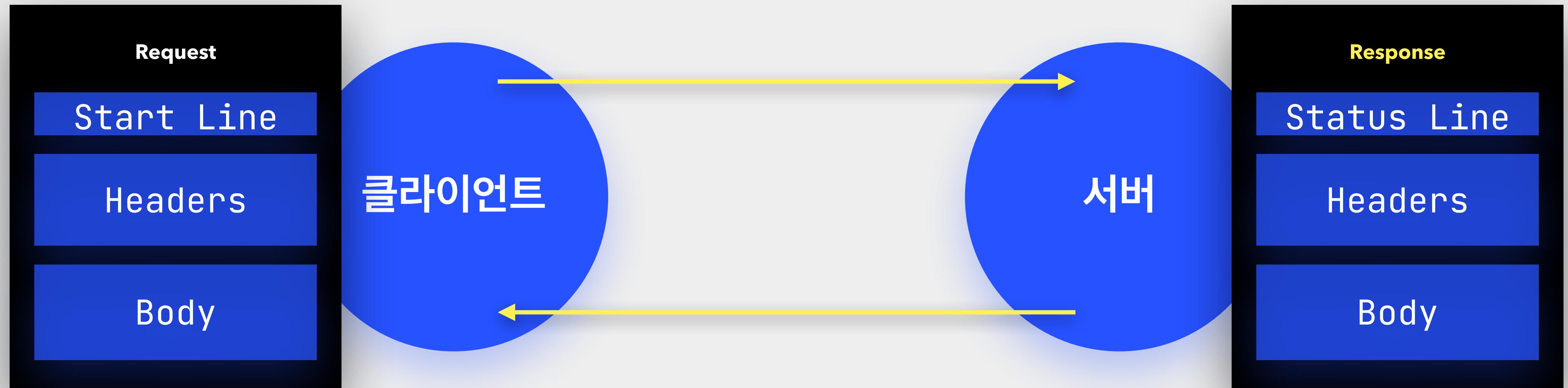
Body

# Response

Status Line

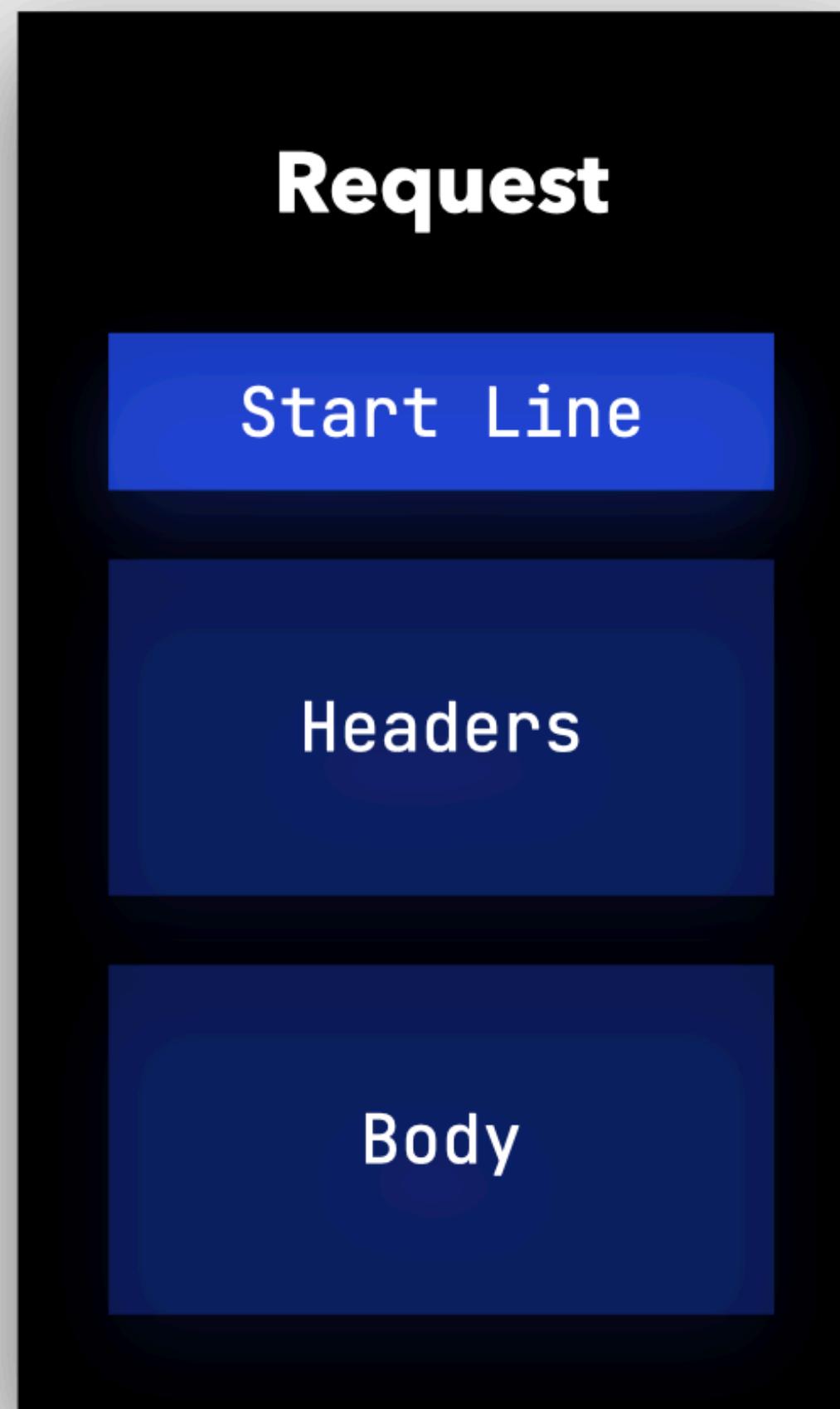
Headers

Body



**Line - Header - Body로 이어지는  
규격을 가지고 통신하는구나!**

그러면, 클라이언트는 서버에게 뭘 요청할 수 있을까?



## HTTP method

**GET /beta.html HTTP/1.1**

서버가 클라이언트에게 무엇을 해줘야 하는지 명시!

### Example.

GET 리소스를 가져와라!

POST 데이터를 넣어라!

# 대표적인 HTTP Method



**GET**

**기본적인 요청**

서버의 리소스를 요청



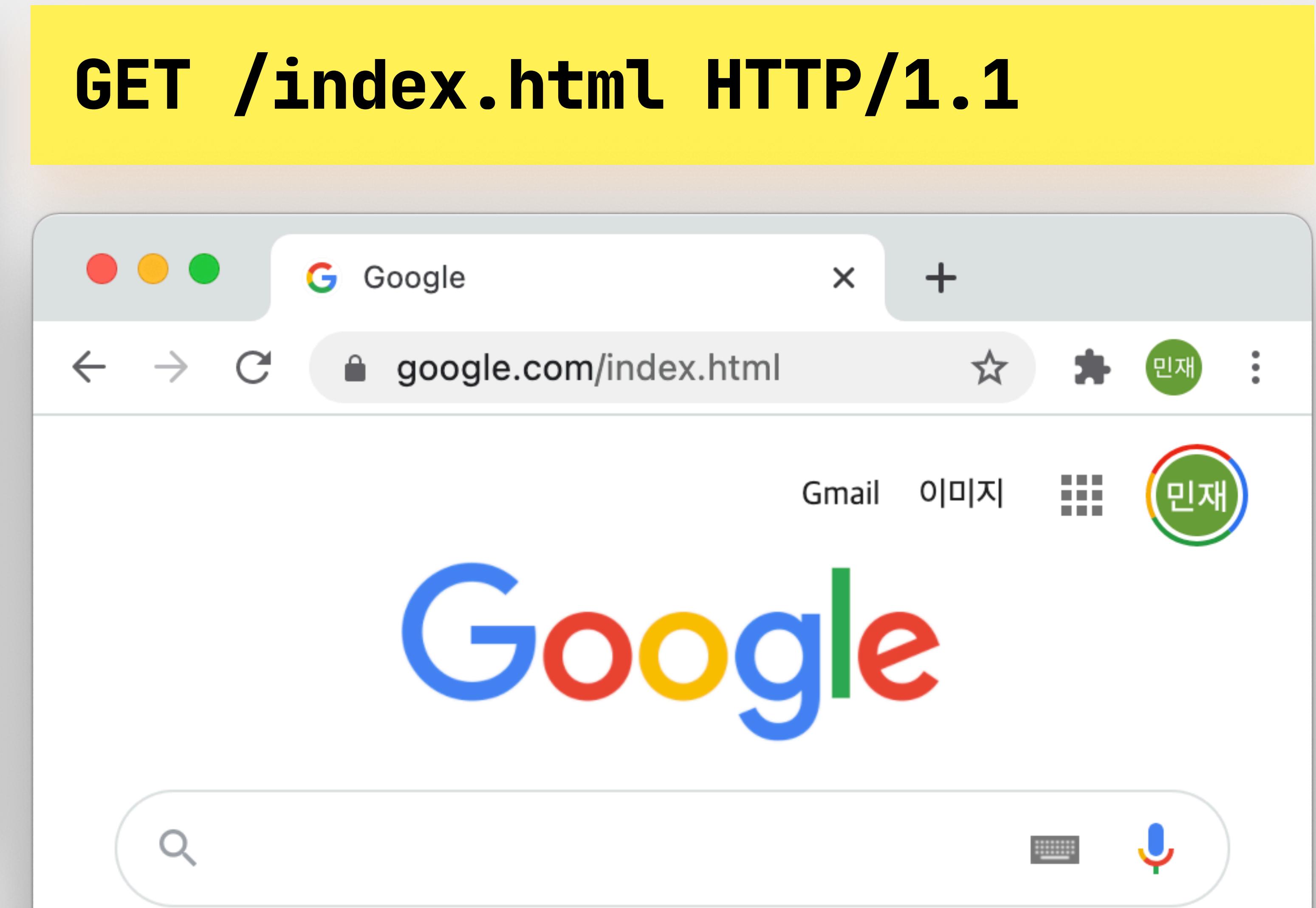
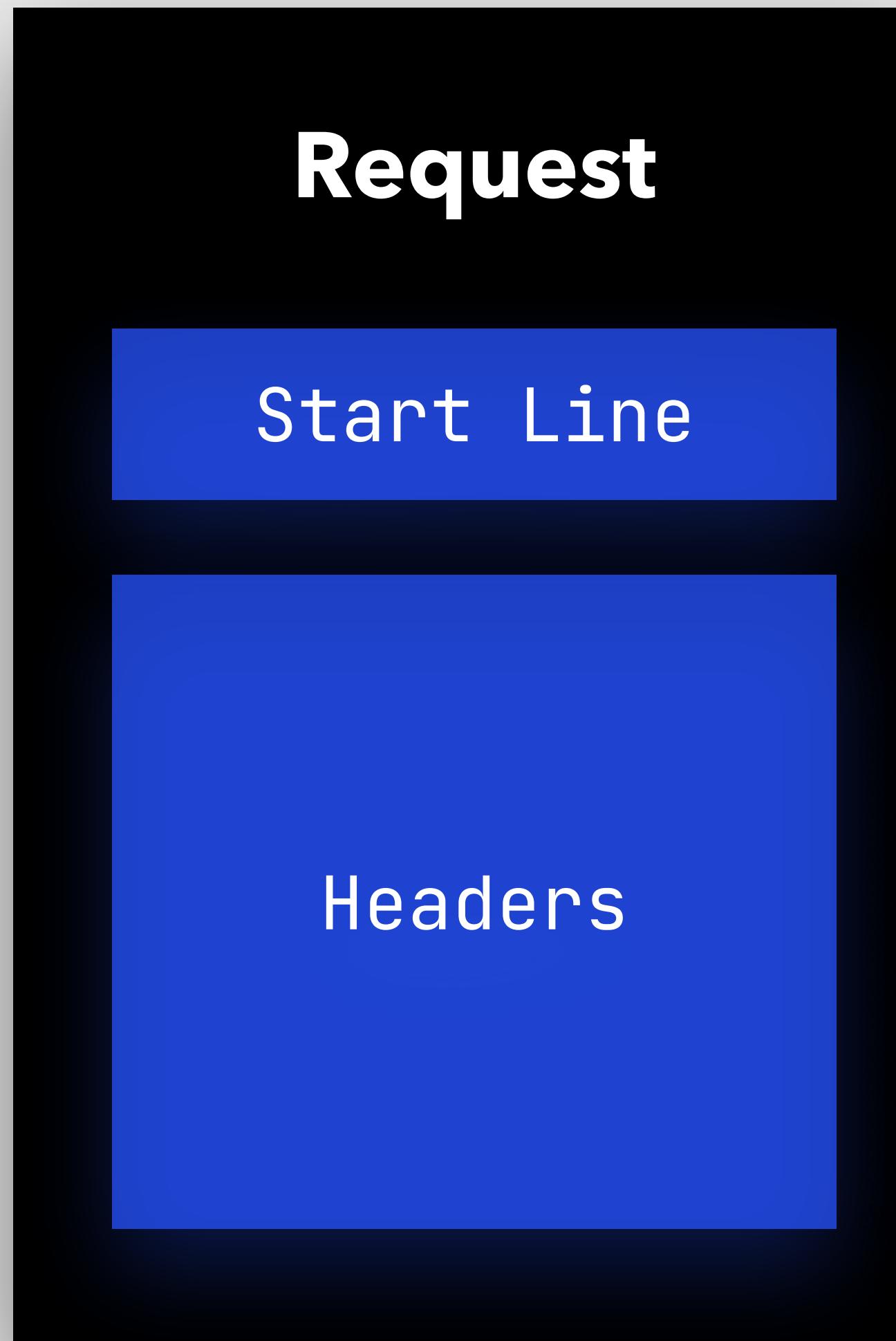
**POST**

**데이터 수정 요청**

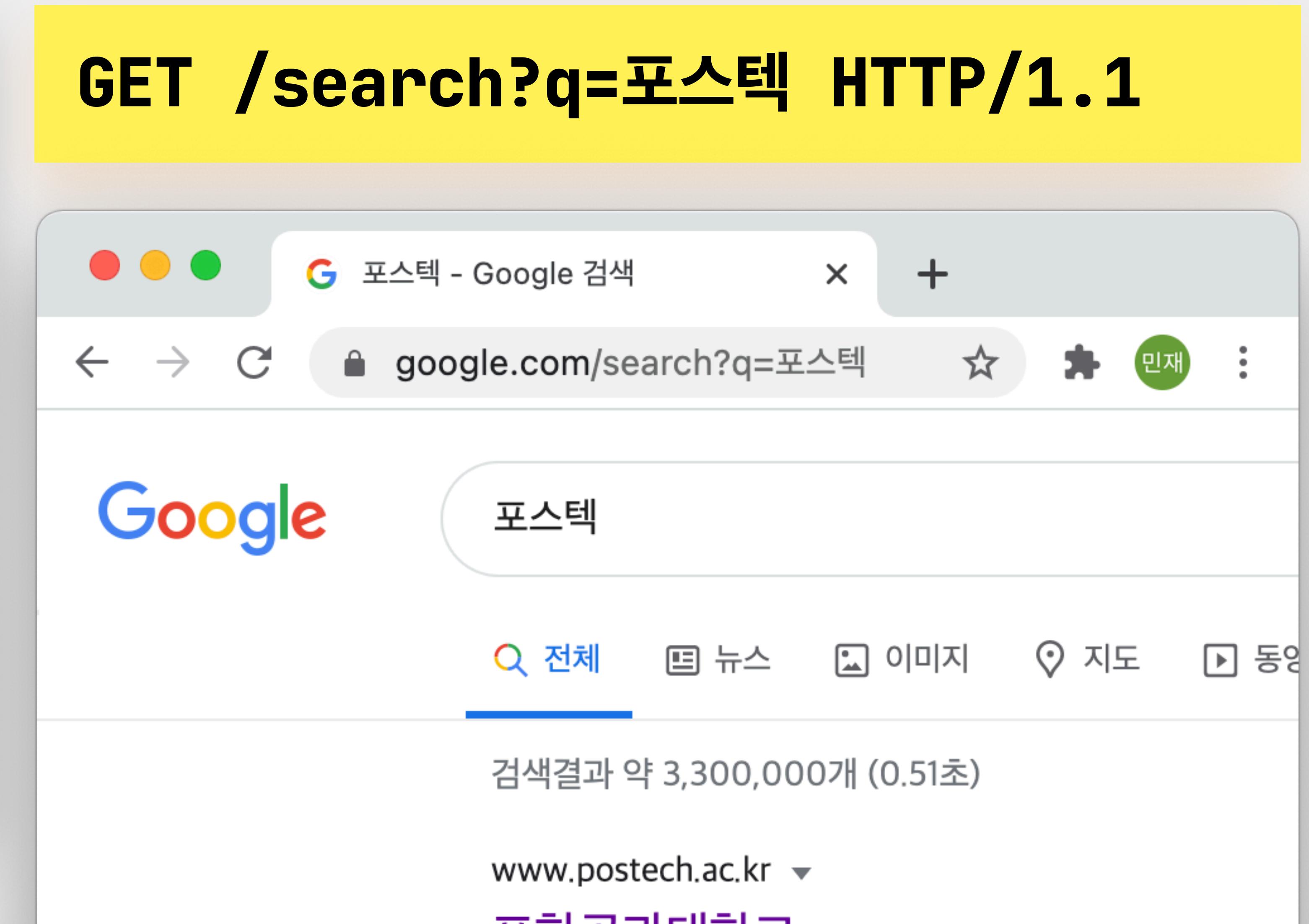
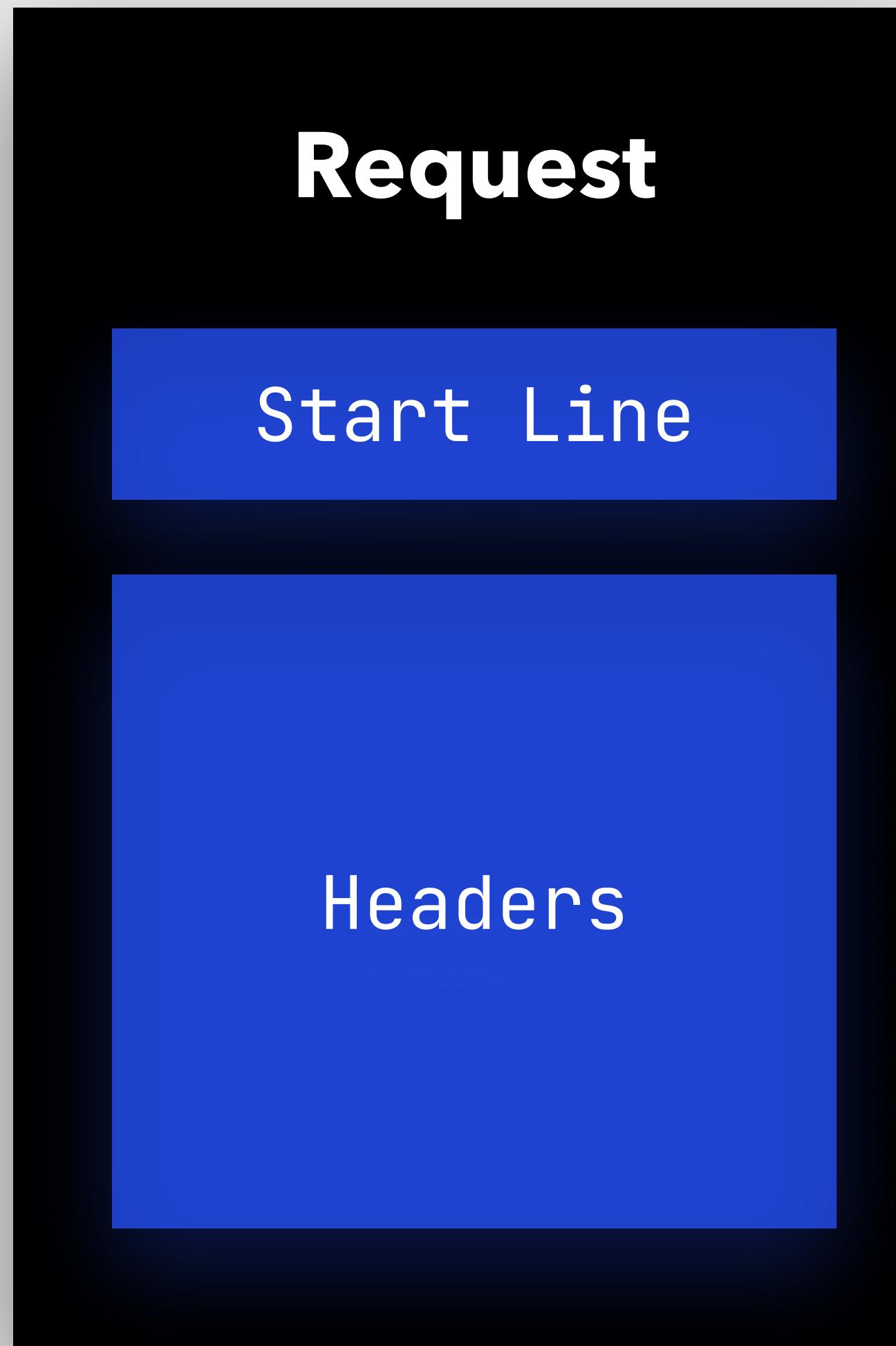
대부분의 데이터 전송에 사용

# GET

# GET

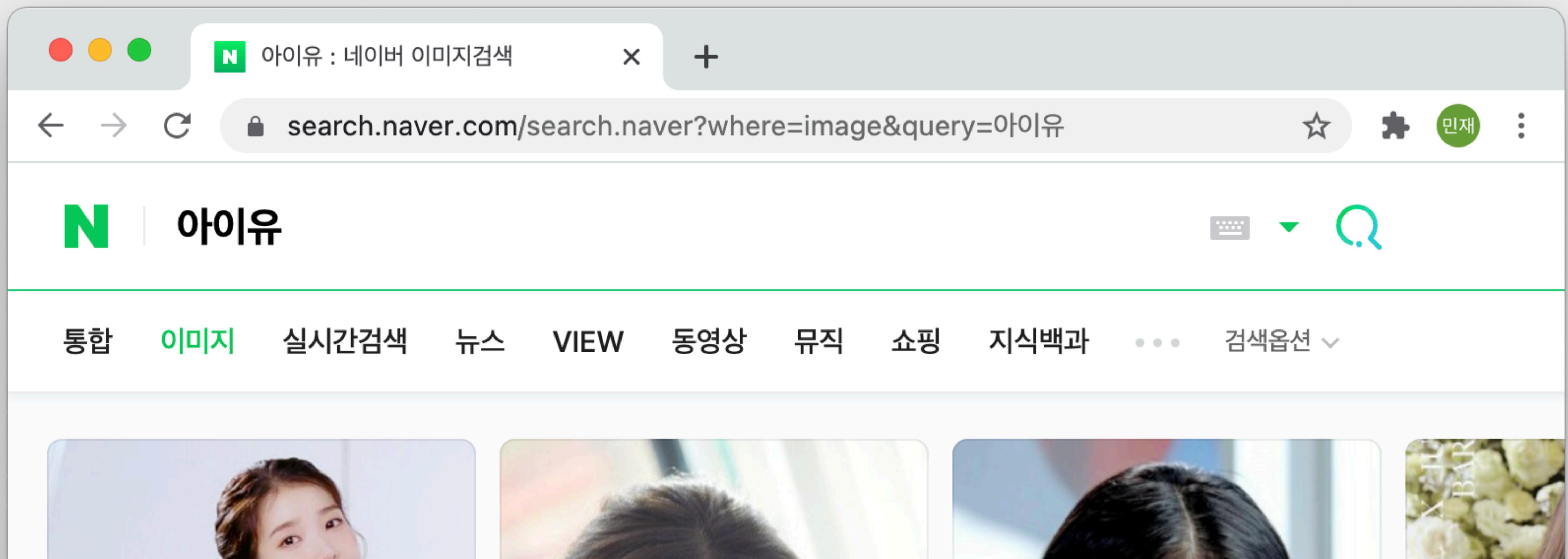


# GET



# GET

GET /search.naver?where=image&query=아이유 HTTP/1.1



# GET

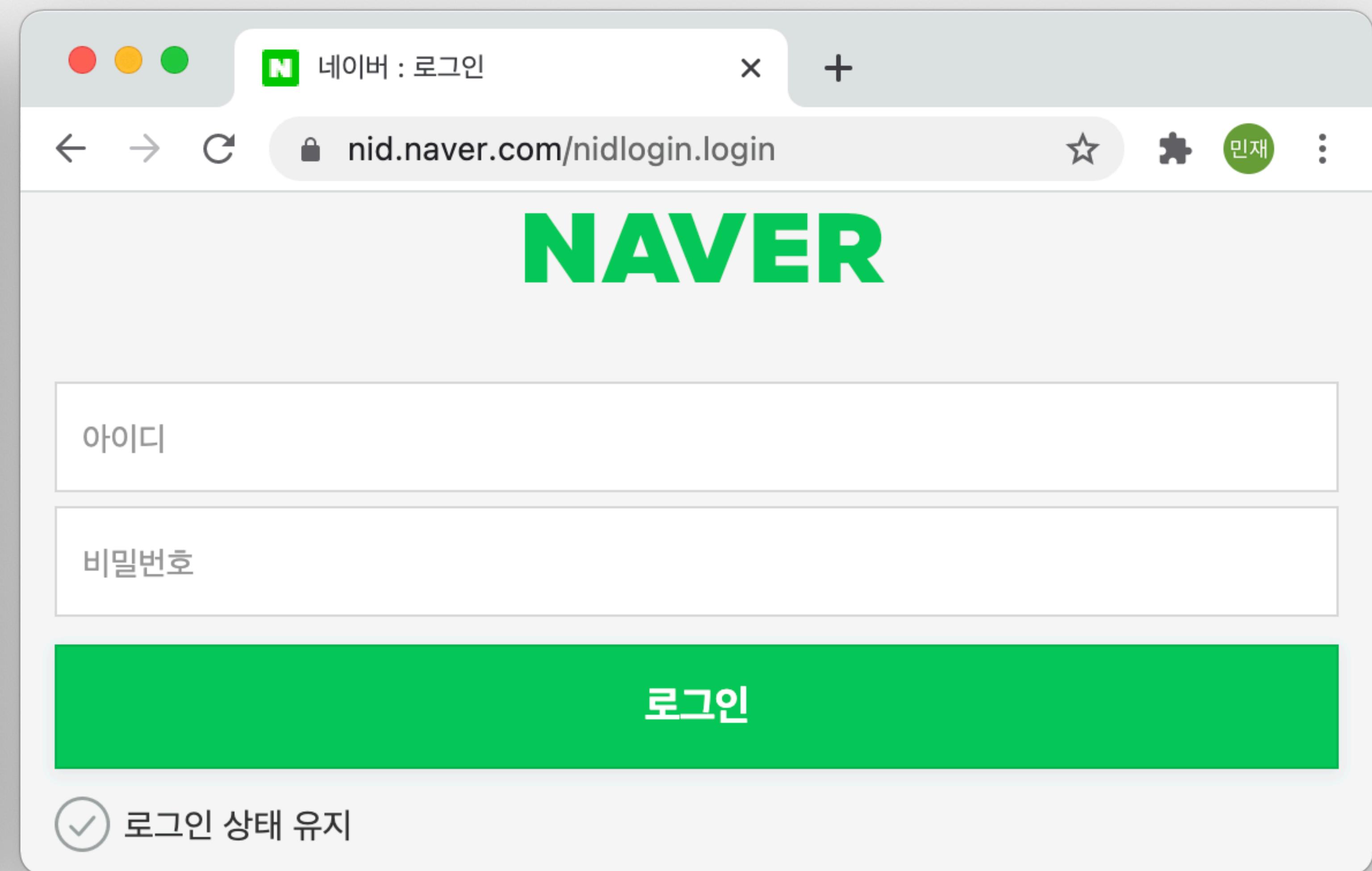
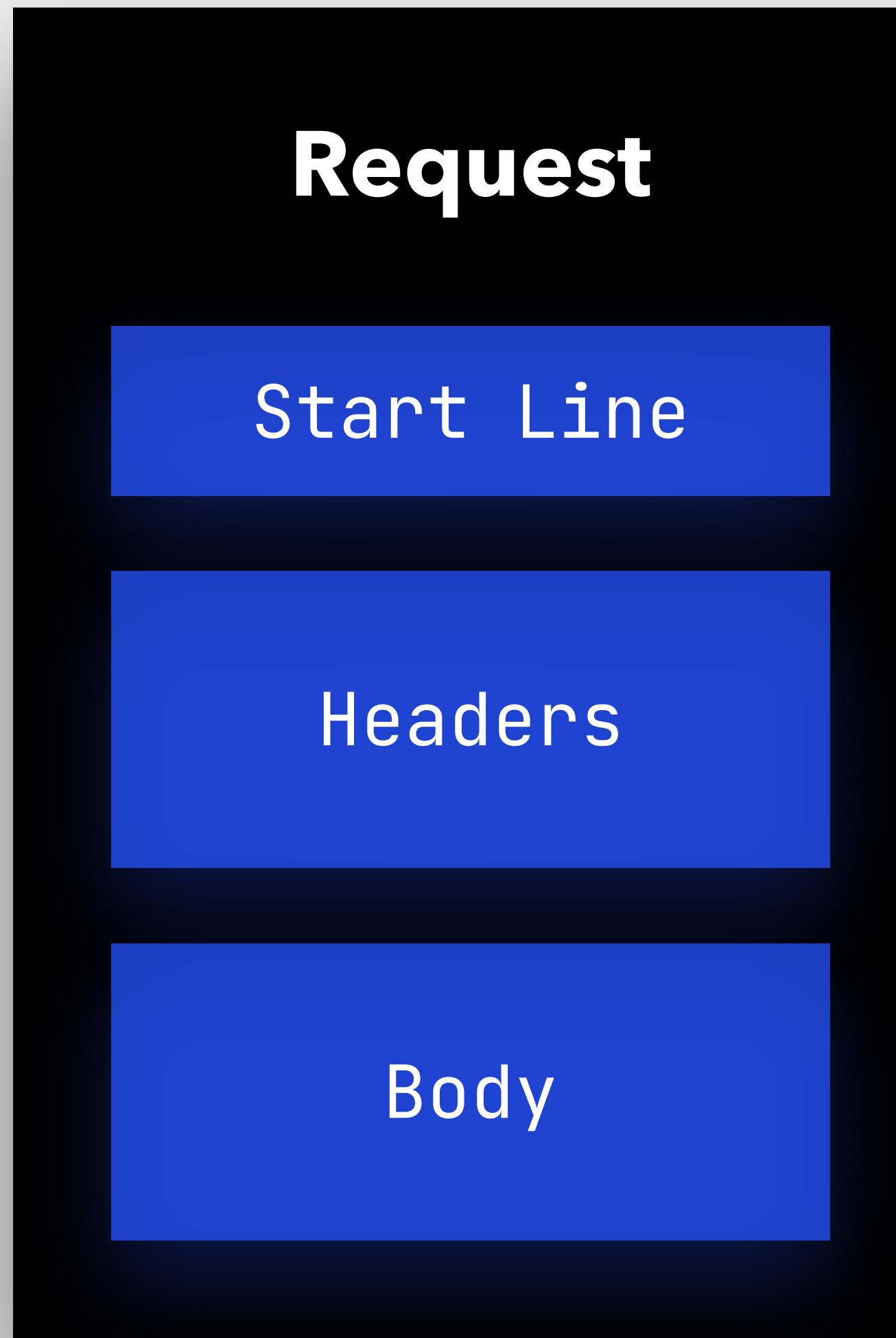
GET /**[ENDPOINT]**?**[KEY]=[VALUE]**&**[KEY]=[VALUE]** HTTP/1.1

Host: plus.or.kr

... (생략)

# POST

# POST



# POST

## Request

Start Line

Headers

Body

**POST /login HTTP/1.1**

Content-Length: 19

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

id=beta&pw=thisispw

# POST

## Request

Start Line

Headers

Body

**POST /login HTTP/1.1**

Content-Length: 32

Content-Type: application/json

{"id": "beta", "pw": "thisispw"}

# 클라이언트는 서버에게 쿼리나 데이터를 요청할 수 있구나!

그러면, 헤더에는 무엇이 들어갈 수 있을까?

# Headers.

Cookie

User-Agent

Referer

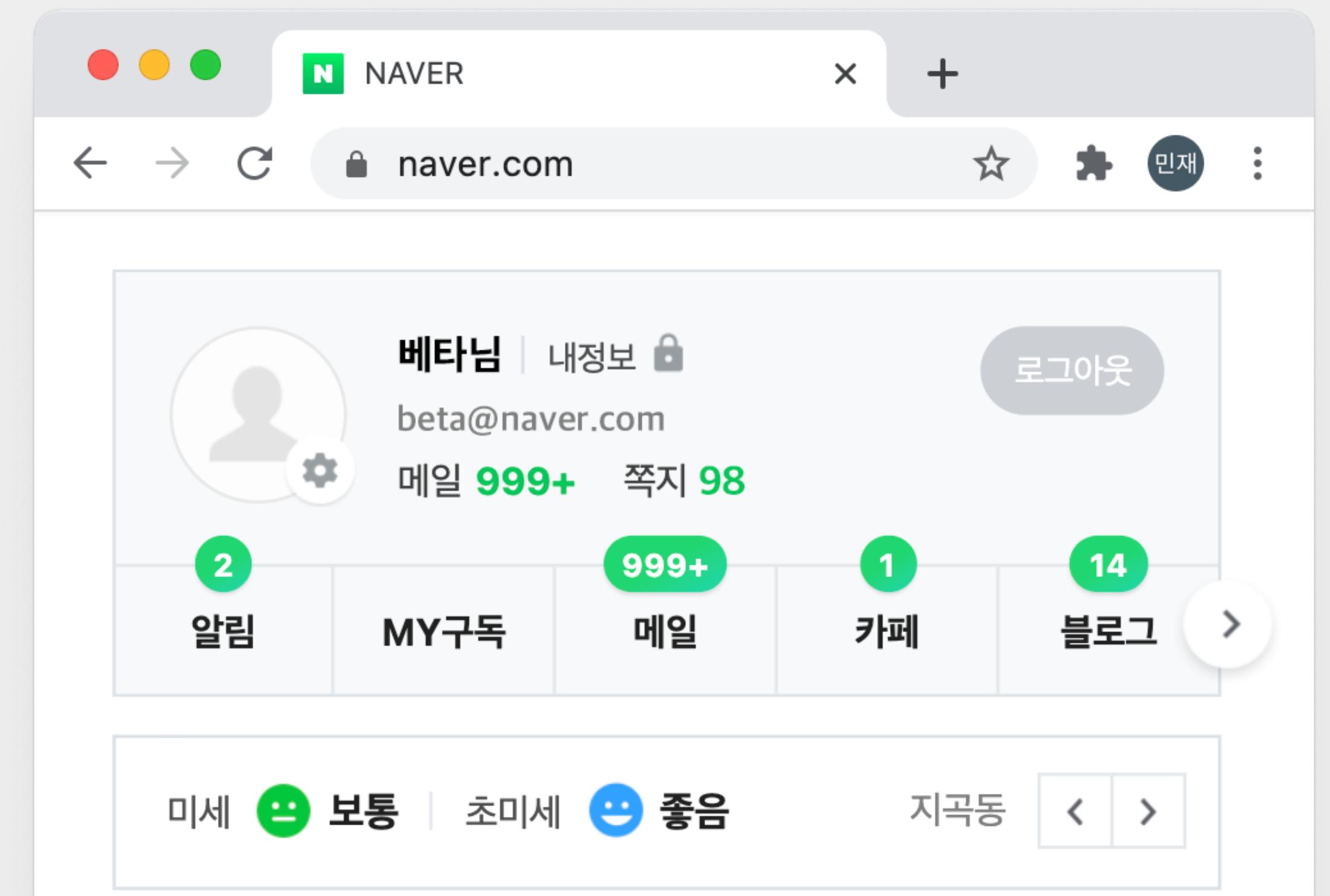
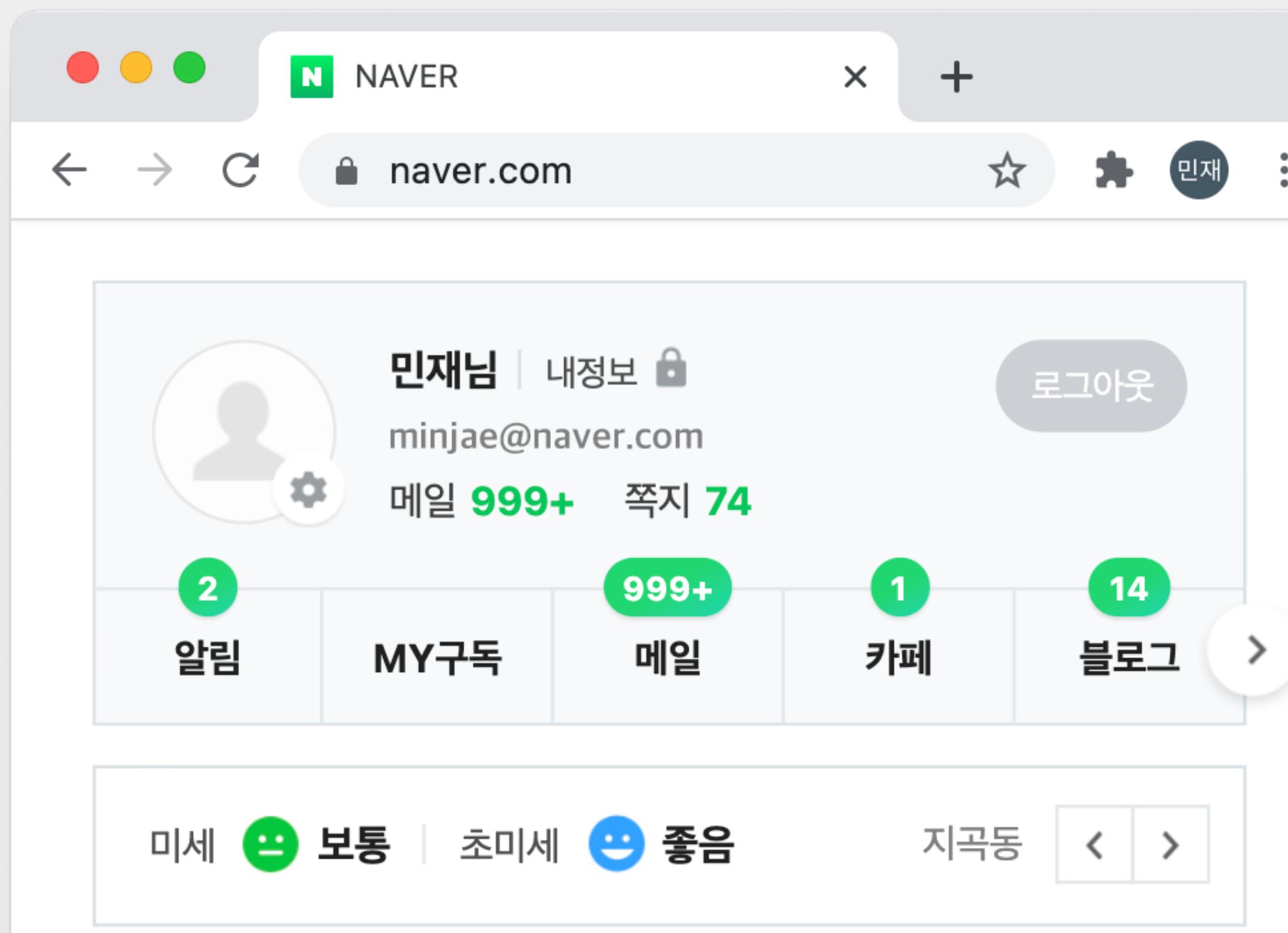
# Cookie

브라우저에 도메인 단위로 저장되는 작은 데이터!  
클라이언트가 서버로 요청을 보낼 때, 헤더에 자동으로 붙게 된다.

(브라우저에서 요청을 보낸다면)

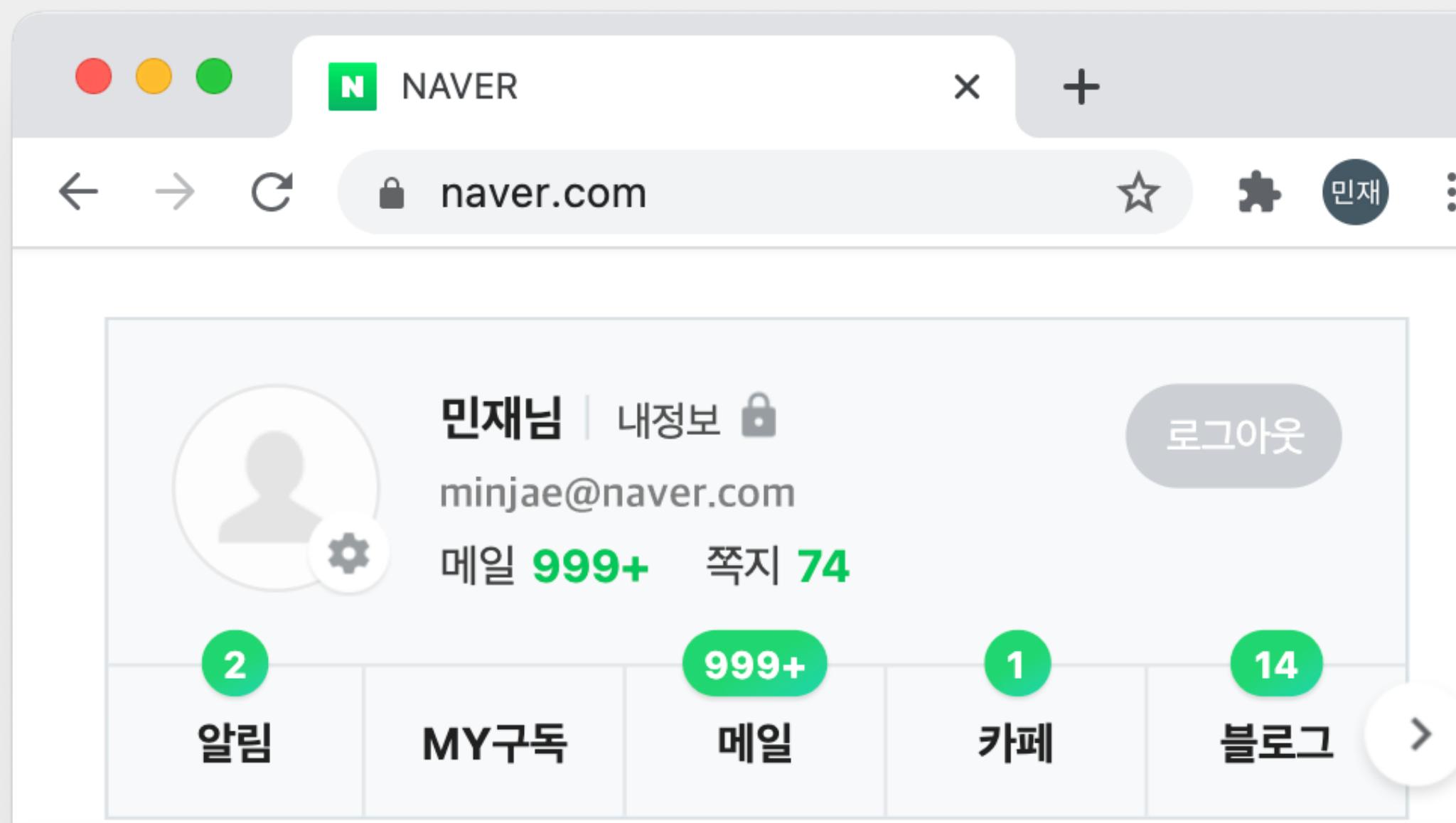
# Cookie

<https://www.naver.com/>

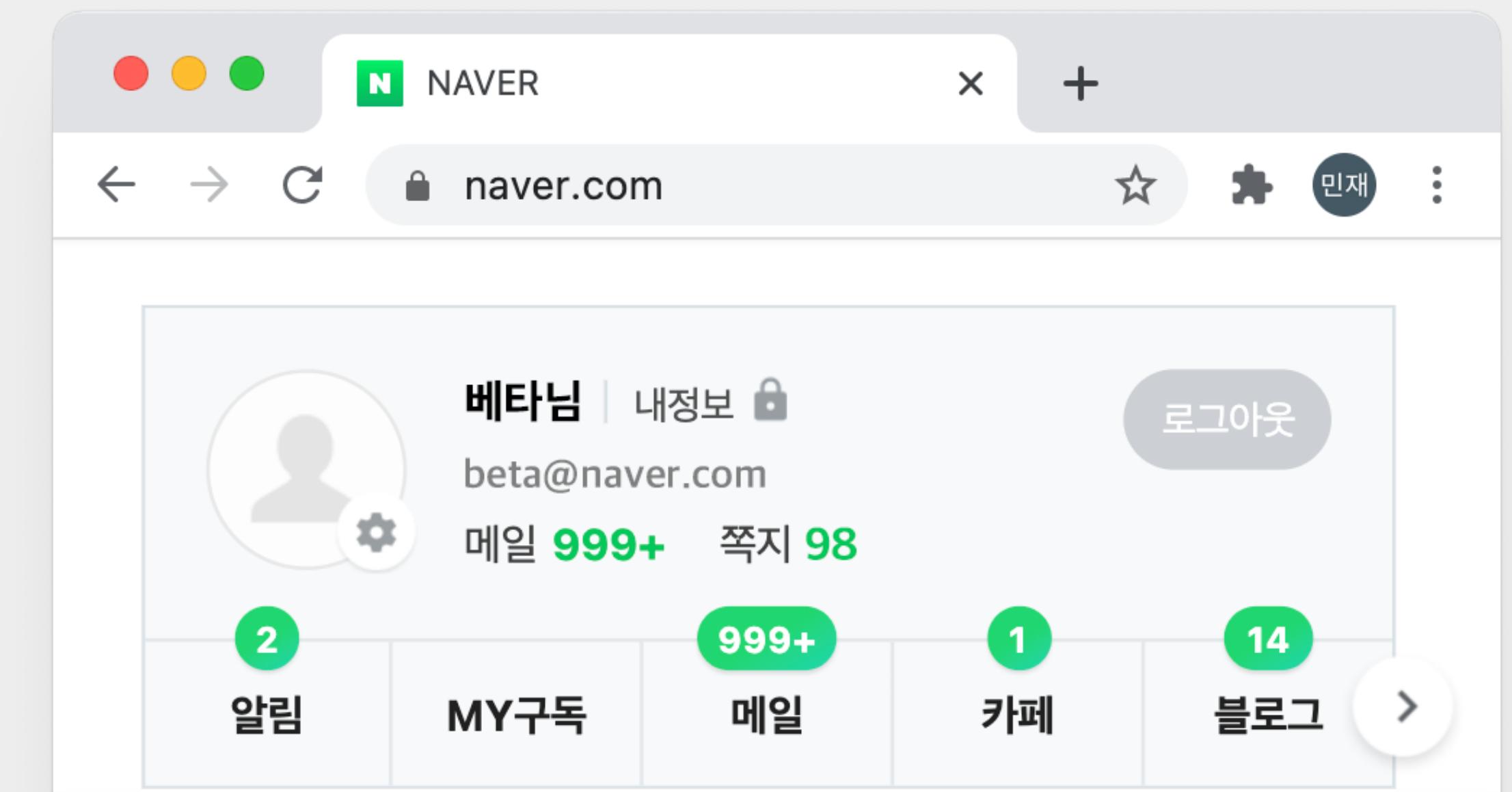


# Cookie

<https://www.naver.com/>



Session=minjae-sess-id



Session=beta-sess-id

# Cookie

`https://www.naver.com/`

Session=minjae-sess-id

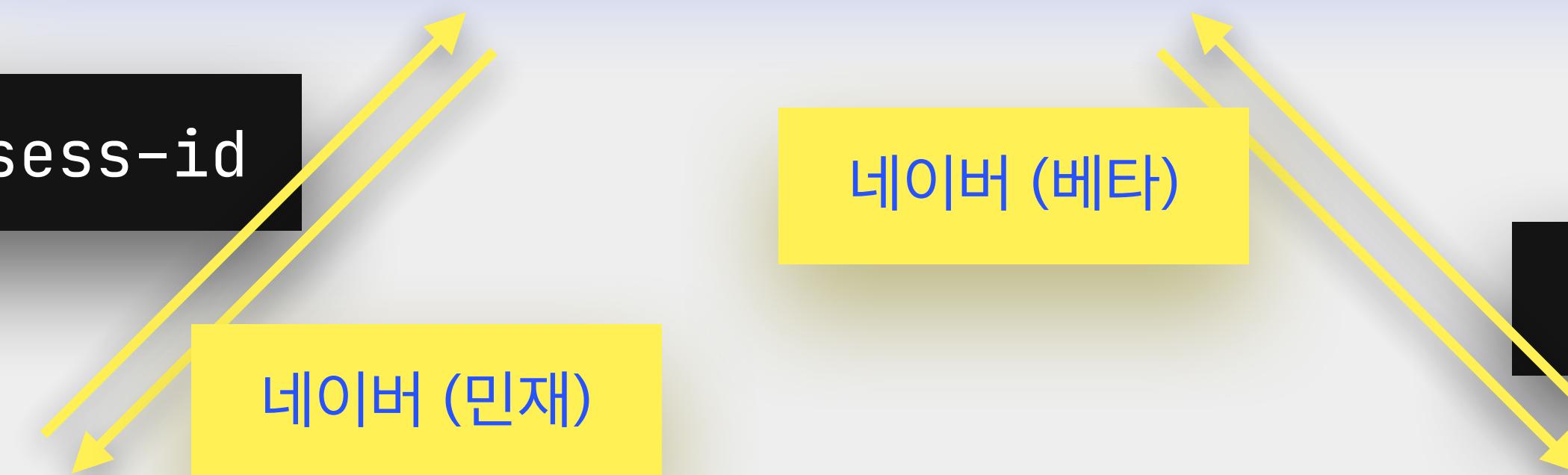
민재

네이버 (민재)

네이버 (베타)

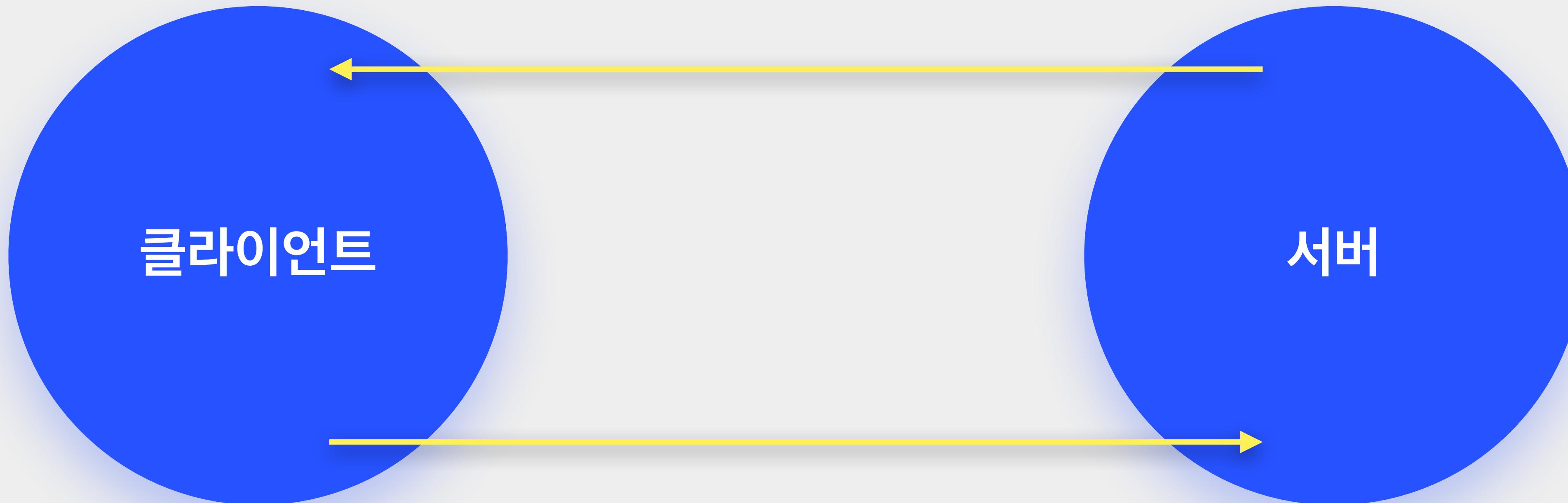
Session=beta-sess-id

베타



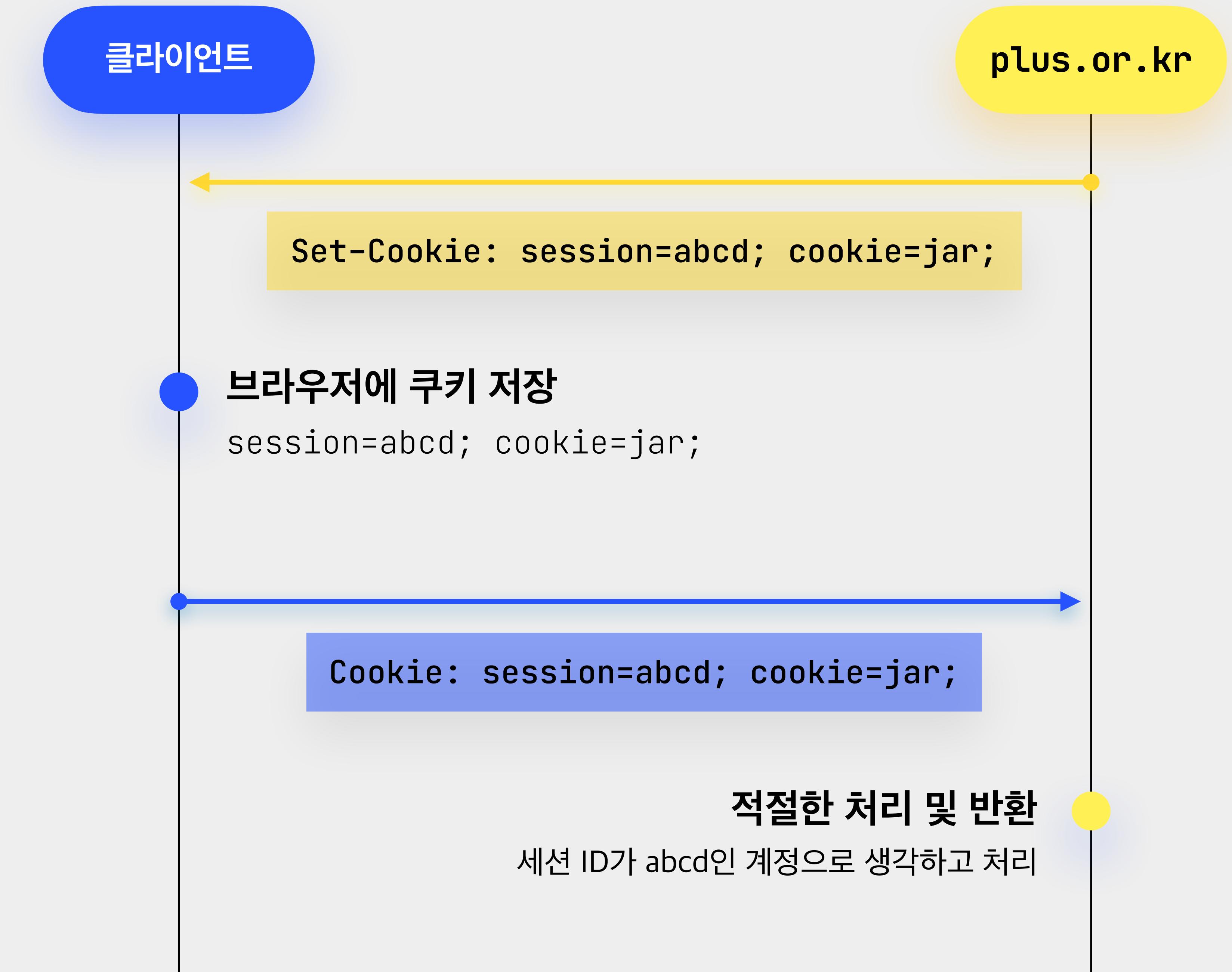
# Cookie

Set-Cookie

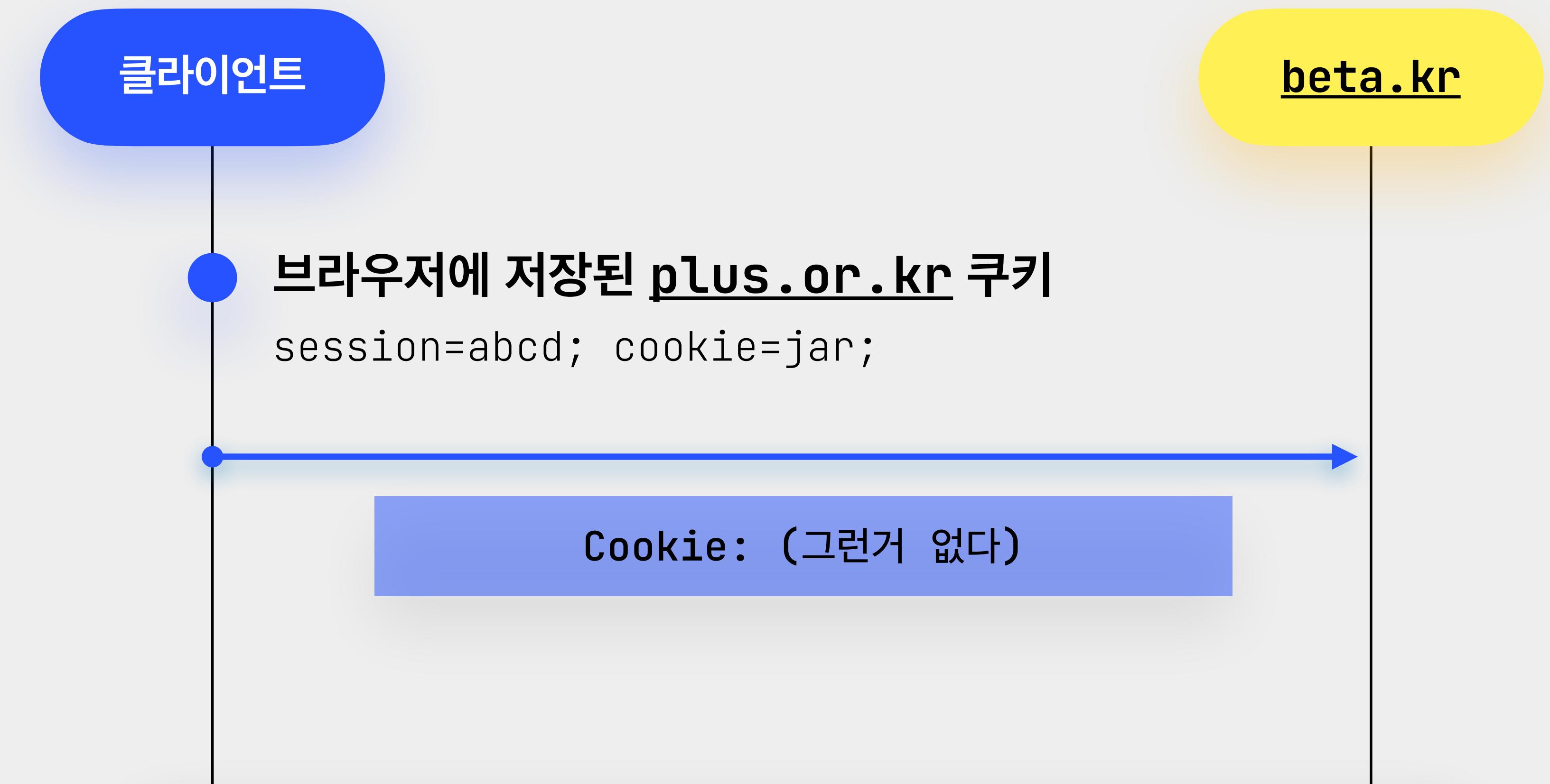


Cookie

# Cookie



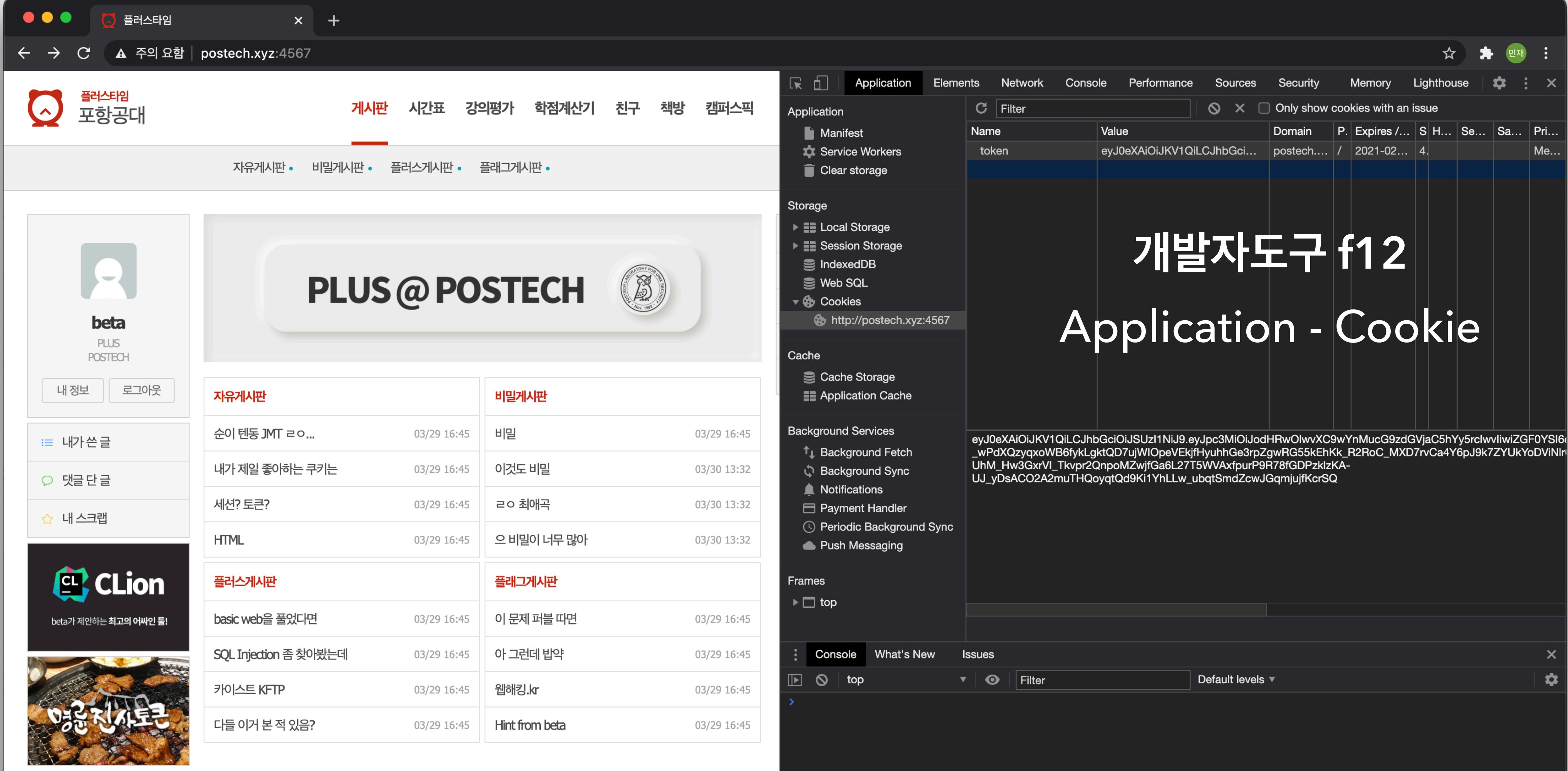
# Cookie



기본적으로, 다른 도메인의 쿠키는 전송하지 않음

Same-Origin-Policy에 대해 검색해봐요

# Cookie



# 개발자도구 f1 2

# Application - Cookie

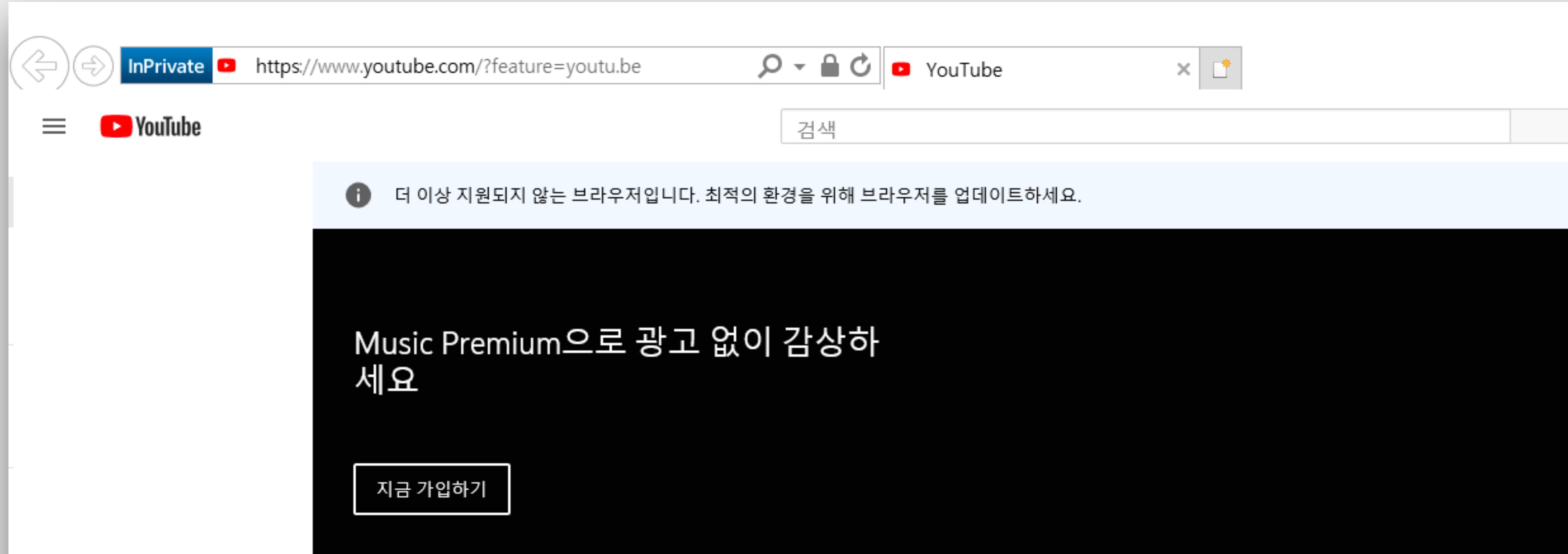
# Headers.

Cookie

User-Agent

Referer

# User-Agent



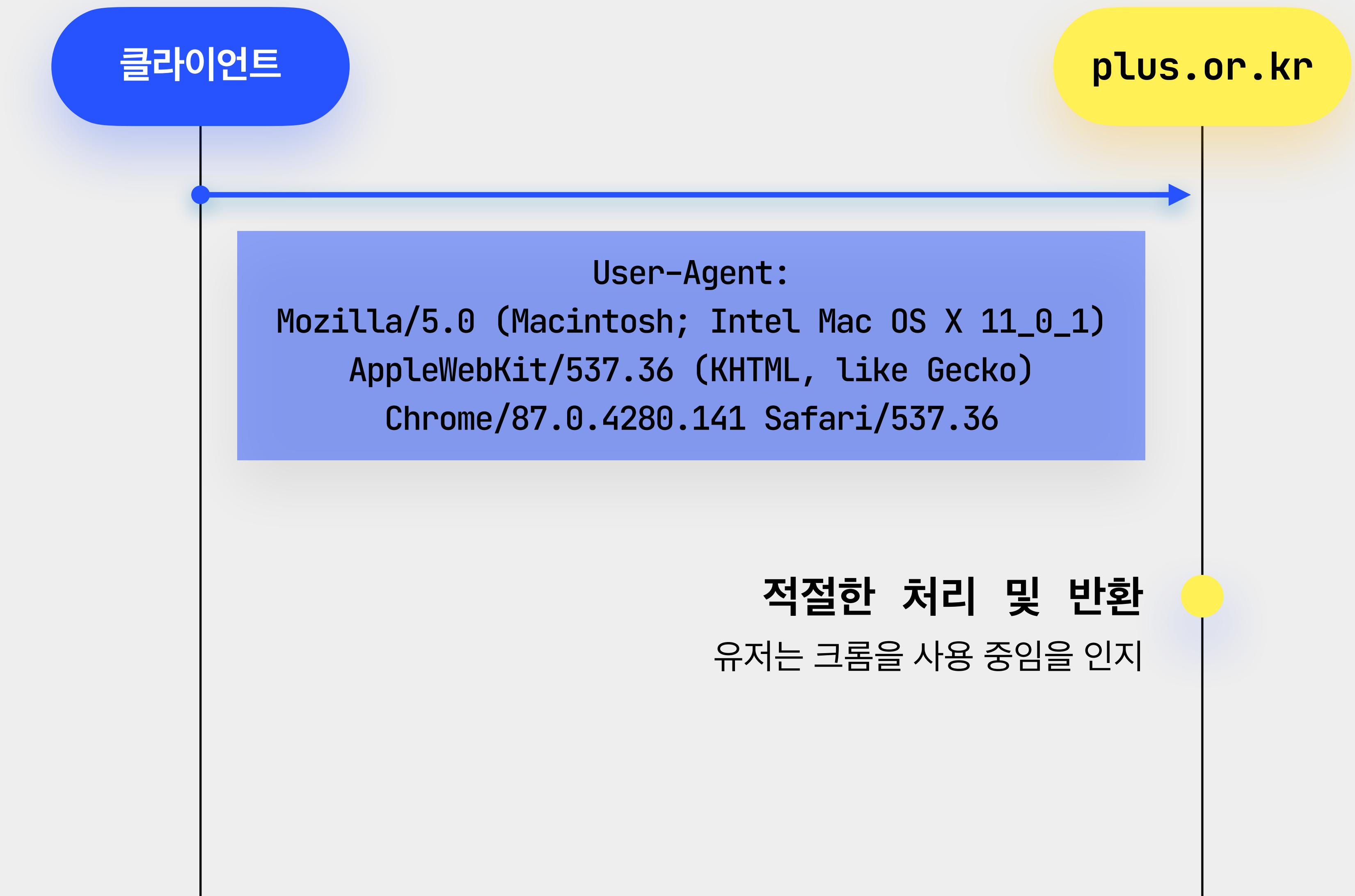
# User-Agent

클라이언트의 환경 정보가 담겨있는 헤더!

클라이언트가 서버로 요청을 보낼 때, 헤더에 자동으로 붙게 된다.

(브라우저에서 요청을 보낸다면)

# User-Agent



# Headers.

Cookie

User-Agent

Referer

# Referer

클라이언트(브라우저)가  
이전에 어떤 페이지에 있었을까?

예시는 Referer를 사용하지 않으며, 이해를 돋기 위한 것임

The screenshot shows a web browser window with the following details:

- Title Bar:** 포항공과대학교 - 나무위키
- Address Bar:** namu.wiki/w/포항공과대학교?from...
- Toolbar:** Back, Forward, Stop, Search, and other navigation icons.
- Header Bar:** 나무위키 logo, search bar, and user profile icon.
- Page Content:**
  - Page Title:** 포항공과대학교
  - Last Edit:** 최근 수정 시각: 2021-01-03 11:18:29
  - Summary:** U.S Software Developers Jobs. Work Remotely  
No Visa needed. Full time jobs. High salary.
  - Postbox Note:** 포스텍에서 넘어옴
  - Category:** 분류: 포항공과대학교
- Page Footer:** 포항공과대학교 관련 틀

# Referer

클라이언트가 이전에 어떤 페이지에 있었는지 그 정보가 담겨있는 헤더!  
클라이언트가 서버로 요청을 보낼 때, 헤더에 자동으로 붙게 된다.

(브라우저에서 요청을 보낸다면)

**TMI.** 원래 영어 단어는 Referrer 이지만, Referer가 정의된 RFC에 Referrer라고 오타가 난 것에서 'Referer'가 유래했다.

# Referer

클라이언트

plus.or.kr

postech.ac.kr 접속

Referer: https://postech.ac.kr

적절한 처리 및 반환

유저가 postech.ac.kr에서 왔음을 확인

Web 상의 컴퓨터들은  
HTTP라는 규격을 가지고 서로 대화하는구나!

그러면, 서버는 어떻게 클라이언트에게 대답할 수 있을까?

# Chapter 3.

Concept

HTTP

**Server**

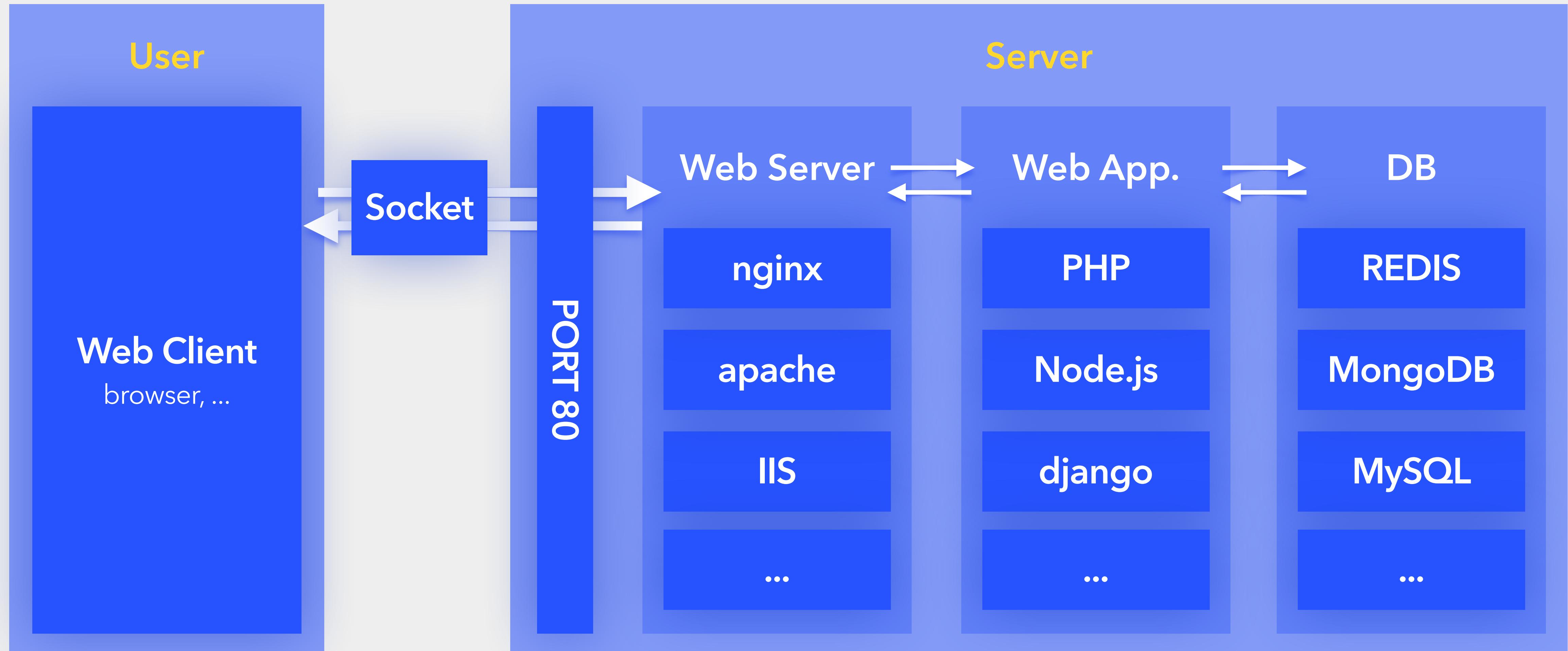
Tools



클라이언트가  
'아이유'를 검색할 때도 있고, '포스텍'을 검색할 때도 있고, (GET)  
데이터로 게시글을 보낼 때도 있고, (POST)  
클라이언트가 민재일 때도 있고, 승원일 때도 있고, (Cookie)  
크롬을 쓸 때도 있고, 파이어폭스를 쓸 때도 있고, (User-Agent)  
네이버를 방문했을 수도 있고, 구글을 방문했을 수도 있는데, (Referer)

서버는 이를 모두 처리해야한다!

도대체, 어떤 구조로?



# **Server**

**Web  
Server**

**중개업자**

요청을 적절히 토스!

**Web  
Application**

**공장**

요청한 물건을 만듦

**Database**

**농장**

원재료를 제공

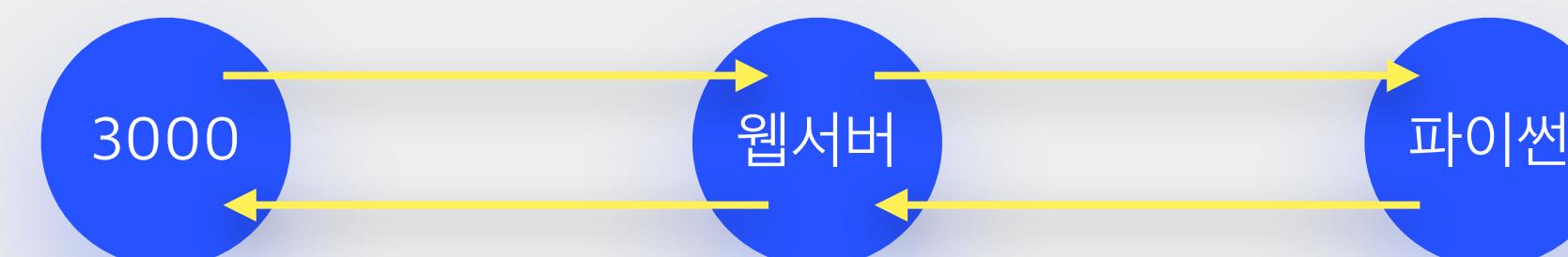
# Web Server

## "중개업자"

특정 포트, 특정 경로로 들어오는 요청에 대해  
파일을 보내주거나 Web Application에게 넘겨줌  
이후, Web Application이 보내준 응답을 클라이언트에게 보내줌.

### Example.

포트 3000에 Python이 연결되어 있는데, 해당 포트로 요청이 들어온다면,  
Python에게 해당 요청을 넘겨주고, Python으로부터 그 응답을 받아서 클라이언트에게 돌려준다.



# Web Application

"공장"

Web Server가 넘겨준 요청에 대해 적절한 응답을 만들어 줌  
응답을 만드는 과정에서 데이터가 필요하다면, 데이터베이스에 쿼리를 날림

## Example.

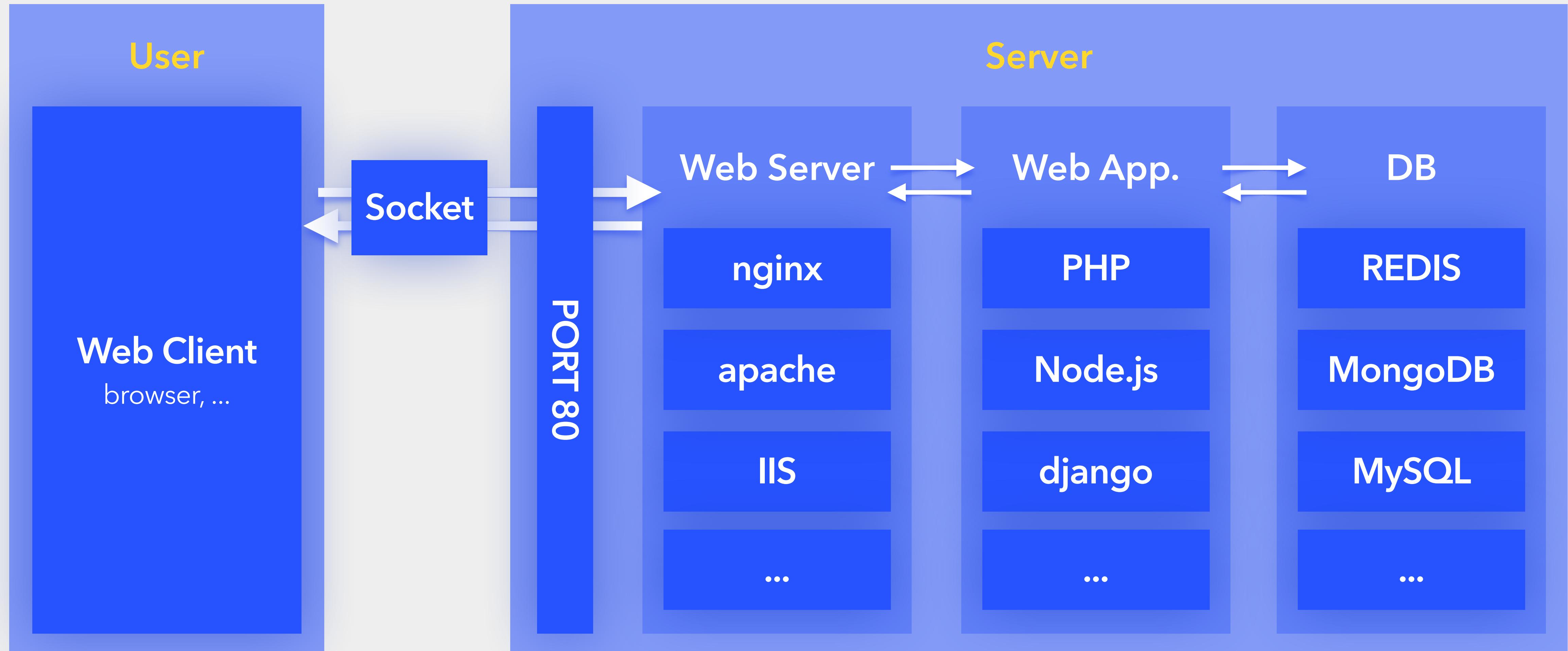
Python이 요청을 받아서 HTML 응답을 만드는데, 이 과정에서 데이터가 필요하다면 SQL에 쿼리를 보내서 데이터를 받아온다.  
응답할 HTML이 만들어지면 Web Application에 응답을 돌려준다



# Database

"농장"

Web Application의 쿼리를 바탕으로 데이터를 생성/조회/수정/삭제(CRUD)하고  
그 결과를 Web Application에게 알려줌.



Web에서 서로 다른 컴퓨터들이 HTTP로 통신하는데,  
서버는 W.Server - W.App - DB의 구조로 요청을 처리하는구나!

# Chapter 4.

Concept

HTTP

Server

Tools

# Browser

# Browser

"가장 쉽고, 빠르고, 시각적으로 Web을 탐색하는 방법"



Chrome



Firefox



Internet Exploit

# Browser

"가장 쉽고, 빠르고, 시각적으로 Web을 탐색하는 방법"

HTTP Request Message,

즉 Start Line - Header - Data를 적절하게 잘 만들어서 서버에 요청을 보내고,  
서버로부터 받은 데이터들을 잘 버무려서 사용자에게 시각적으로 보여줌

# Browser

"가장 안전하게 Web을 탐색하는 방법"

웹의 영역과 클라이언트의 영역은 구분되어야 한다!

예를 들어, 웹 상의 JS 코드가 사용자의 파일에 접근해서  
이를 마구 수정하는 것은 있어서는 절대 안된다.

# Browser

"가장 안전하게 Web을 탐색하는 방법"

웹의 영역과 클라이언트의 영역은 구분되어야 한다!

그렇기 때문에, 브라우저는 사용자와 웹을 분리하는 일종의 샌드박스로 작동해서 사용자가 웹을 안전하게 탐색할 수 있도록 하는 역할을 수행한다.

# DevTools

# Chrome DevTools

## Network Tab

브라우저가 주고 받는 모든 HTTP Message들을 볼 수 있는 탭

각 요청에 대해 Request와 Response의  
Start/Status Line, Headers, Data를 확인할 수 있다.

# Chrome DevTools

## Console Tab

현재 페이지에서 JavaScript를 실행할 수 있는 탭

HTML, JS가 서버로부터 불러와진 상태에서  
JS를 실행할 수 있다.

# Chrome DevTools

## Sources/Console Tab

HTML, JS 등 서버로부터 불러온 데이터를 확인할 수 있는 템

## Application Tab

어떤 쿠키가 있는지 확인할 수 있는 템

# Requests

# Python Requests

Python에서 HTTP Request/Response를 쉽게 처리해주는 라이브러리

파이썬의 형태로 작성한 Request를 HTTP Request로 요청하고,  
그에 따른 HTTP Response를 파이썬의 형태로 처리할 수 있도록 도와준다!

# Python Requests

```
import requests

res = requests.get("http://google.com/search", params={
    "q": "아이유"
})

res = requests.post("http://plus.or.kr/login", data={
    "id": "beta",
    "pw": "thisispassword"
})

print(res.text) # Response HTML Text
print(res.headers) # Response HTML Header
```

# Python Requests

```
import requests

res = requests.get("http://google.com/search", params={
    "q": "아이유"
}, cookies={
    "session": "this-is-session-id"
}, headers={
    "User-Agent": "Internet Exploit"
})

# https://requests.readthedocs.io/en/master/user/advanced/
s = requests.Session()
res = s.get("http://google.com")
```

# Conclusion.

서로 다른 컴퓨터들이 Web에서 통신하는데에는 약속된 프로토콜인 HTTP를 사용하며,  
HTTP Message는 Start/Status Line, Headers, Body로 구성된다.

클라이언트의 요청에 대해 서버는 클라이언트에게 해당 요청에 대한 응답을 전송하는데,  
이때 이 응답은 Web Server, Web Application, Database를 거쳐서 만들어진다.

이 과정은 브라우저, 브라우저 내장 개발자 도구, Python Requests 등을 통해 투아볼 수 있다.

# WEB 101.

beta@plus.

**Gwon Minjae**, Dept. of Computer Science & Engineering, POSTECH.

Revised 2021, Created 2020.

Unauthorized disclosure is prohibited.

Some Definitions from Mozilla Foundation.