

# Back-end Seminar: TypeScript & Node.js

연사: Default ( 김현수 )

대상: KAIST SPARCS 2025 봄 신입회원



# Review1

---

이런 도구들을 잘 사용하는 것 뿐만 아니라,  
이걸 왜 사용하지? 생각해보기



# React Review1

---

```
npm create vite@latest my-app --template react # 혹은 npx create-react-app my-app  
cd my-app  
npm install  
npm run dev
```

뭔가 명령어 똑딱똑딱해서 하면  
리액트 라이브러리를 써서 웹 클라이언트를 만들 수 있는 건 알겠어  
근데 어떻게 작동하는건데?



# React Review1

```
npm create vite@latest my-app --template react # 혹은 npx create-react-app my-app  
cd my-app  
npm install  
npm run dev
```

뭔가 명령어 똑딱똑딱해서 하면  
리액트 라이브러리를 써서 웹 클라이언트를 만들 수 있는 건 알겠어  
근데 어떻게 작동하는건데?



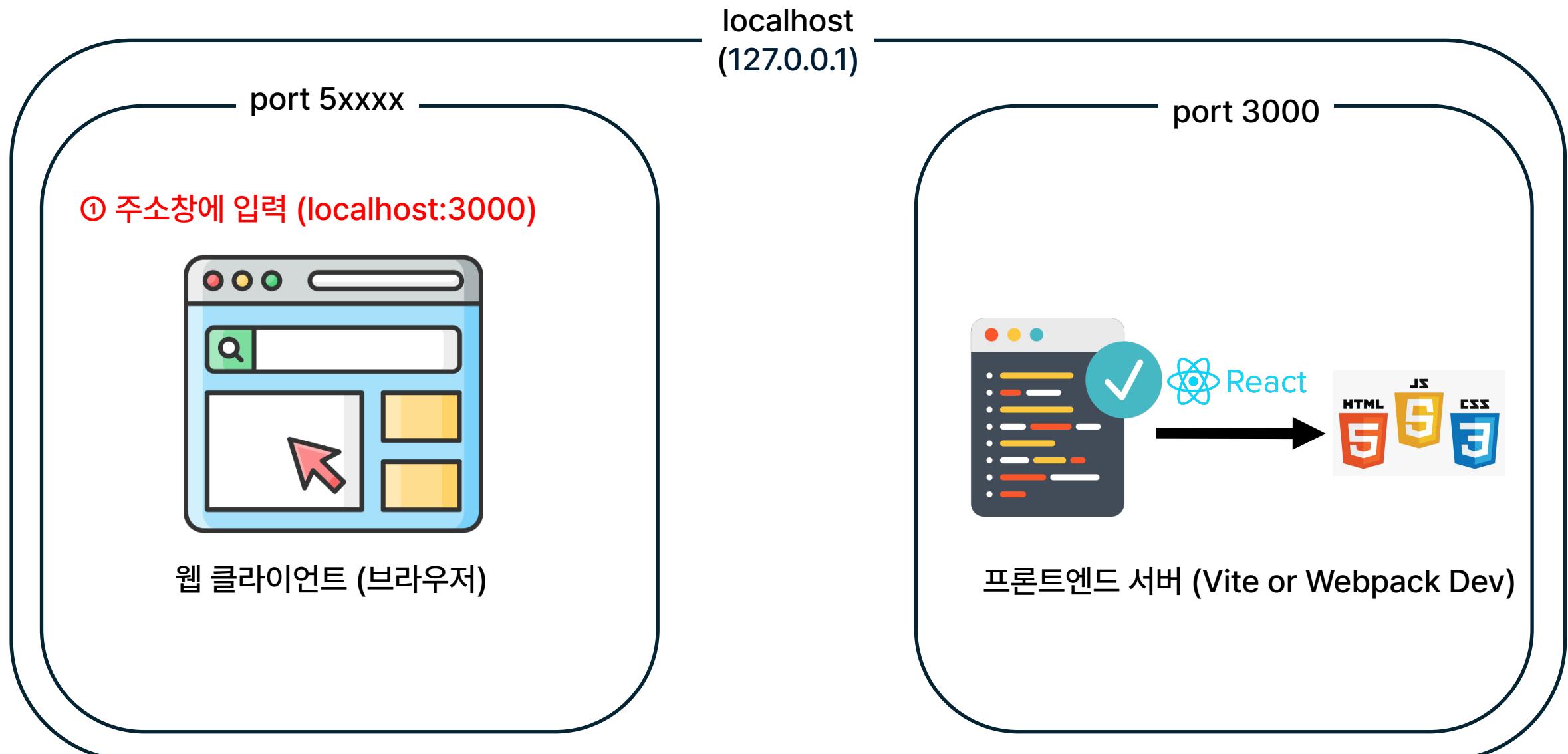
# React Review1

```
npm create vite@latest my-app --template react # 혹은 npx create-react-app my-app  
cd my-app  
npm install  
npm run dev
```

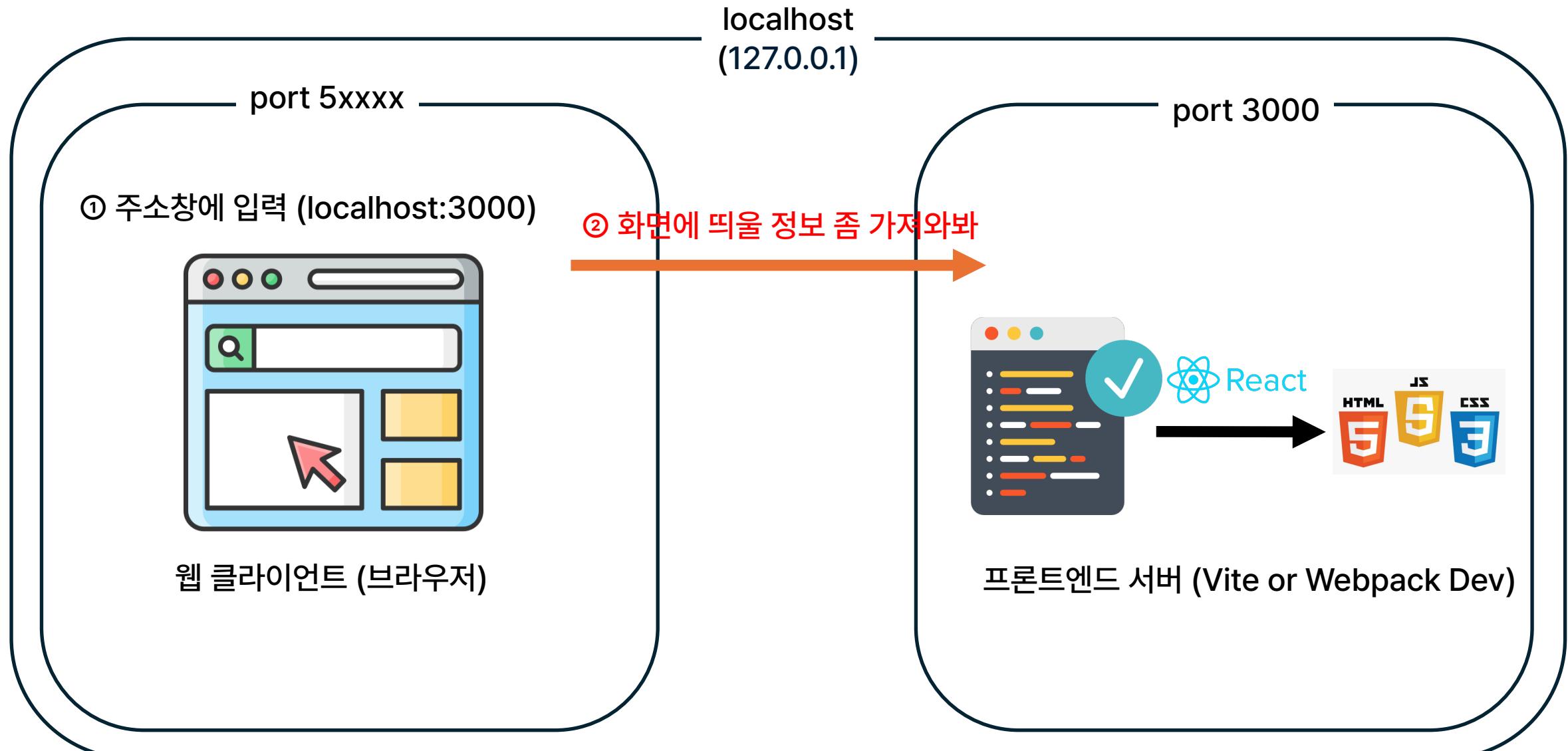
아래의 과정을 한 번에 해주는 것이 `npm run dev`라는 명령어  
물론 아래 과정을 따로따로 할 수 있는 방법도 있다.



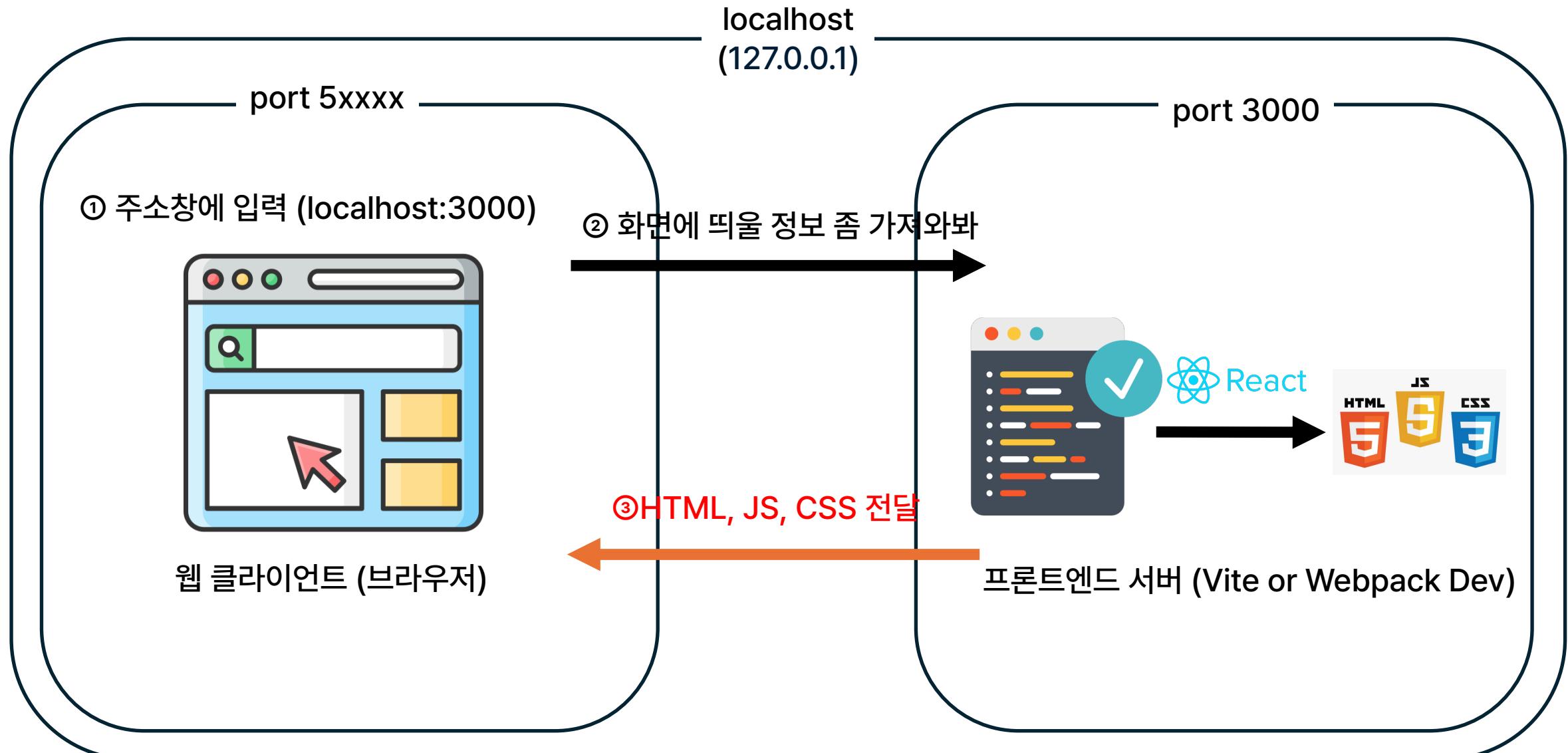
# React Review2



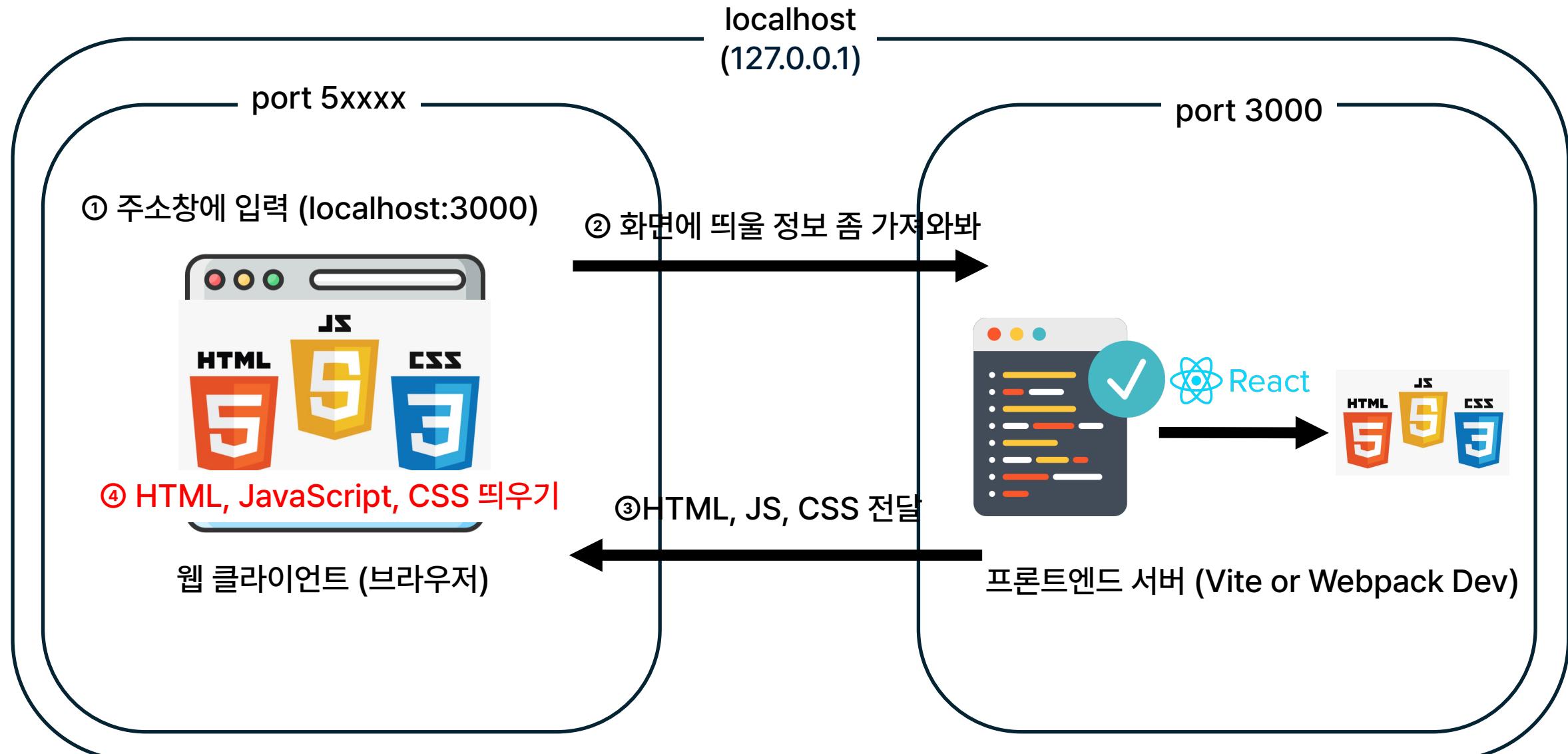
# React Review2



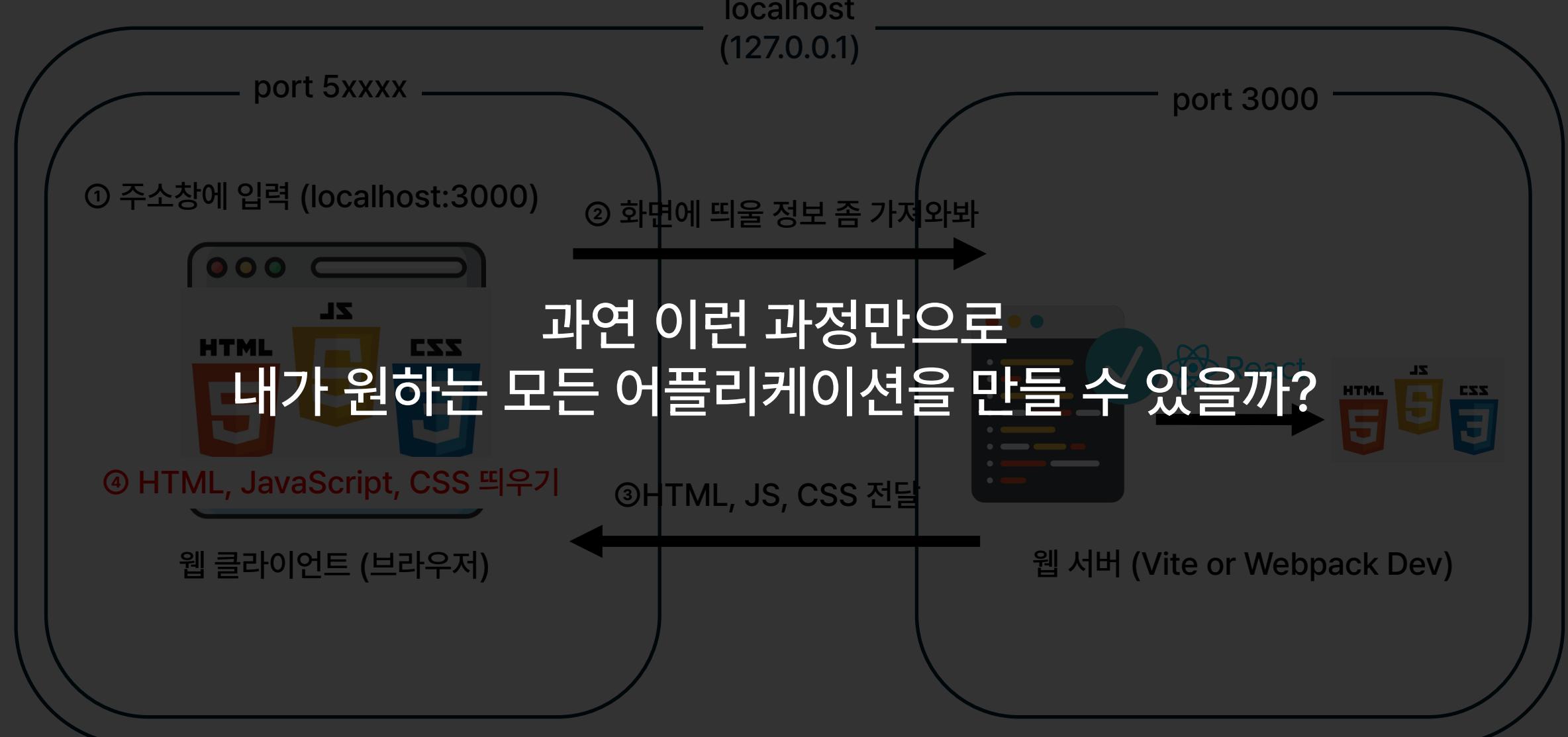
# React Review2



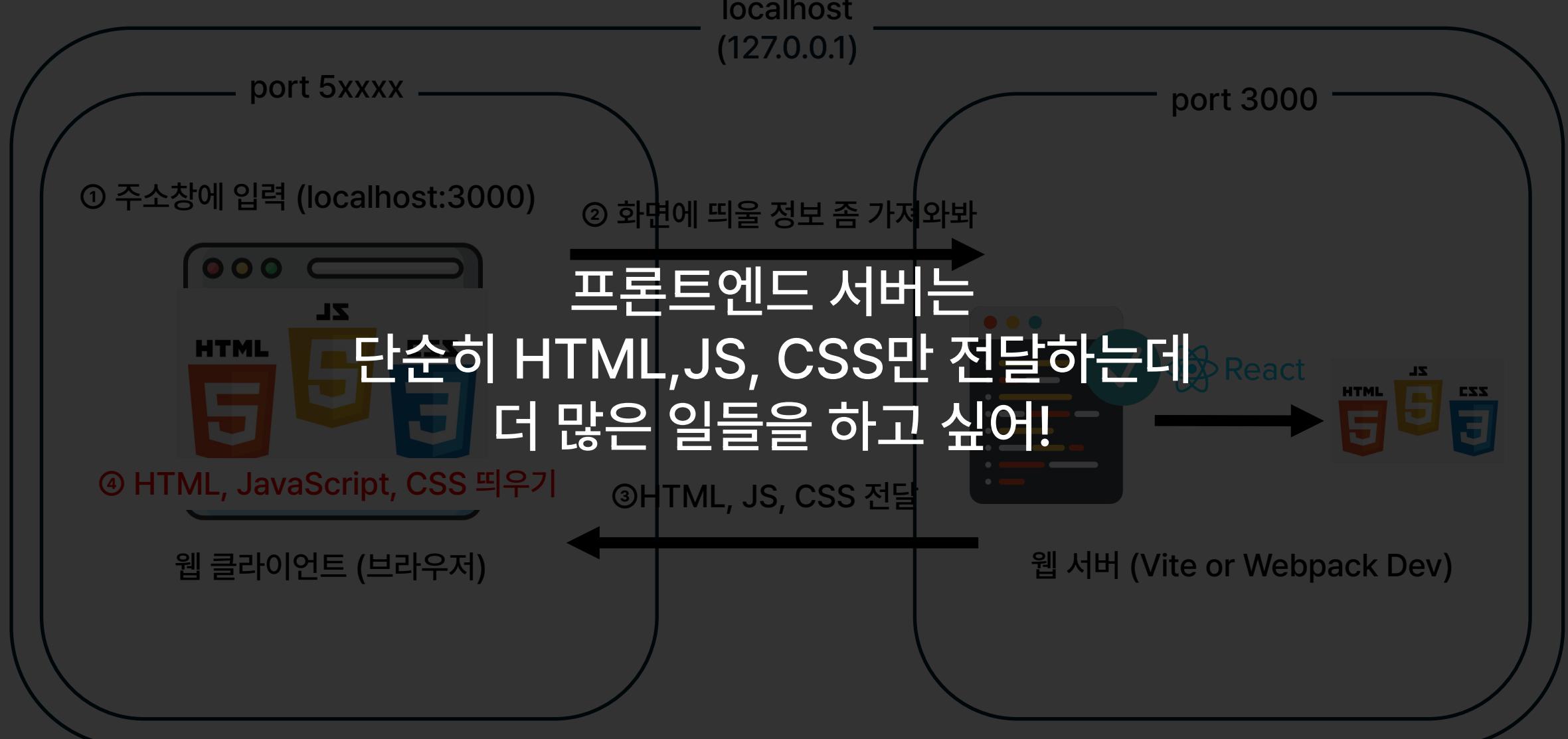
# React Review2



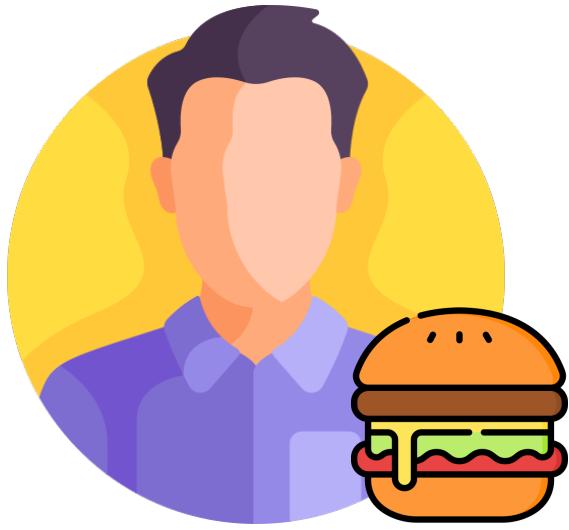
# React Review2



# React Review2



# Back-end가 필요한 이유



고객(웹 브라우저)



메뉴판(프론트엔드 서버)

# Back-end가 필요한 이유



고객(웹 브라우저)



메뉴판(프론트엔드 서버)

# Back-end가 필요한 이유



# Back-end가 필요한 이유



고객(웹 브라우저)

“음 이렇게 주문하면 배부르겠군  
근데 내 햄버거 누가 만들어줌?”

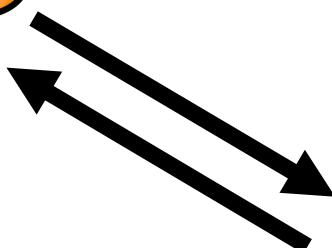


메뉴판(프론트엔드 서버)

# Back-end가 필요한 이유



고객(웹 브라우저)



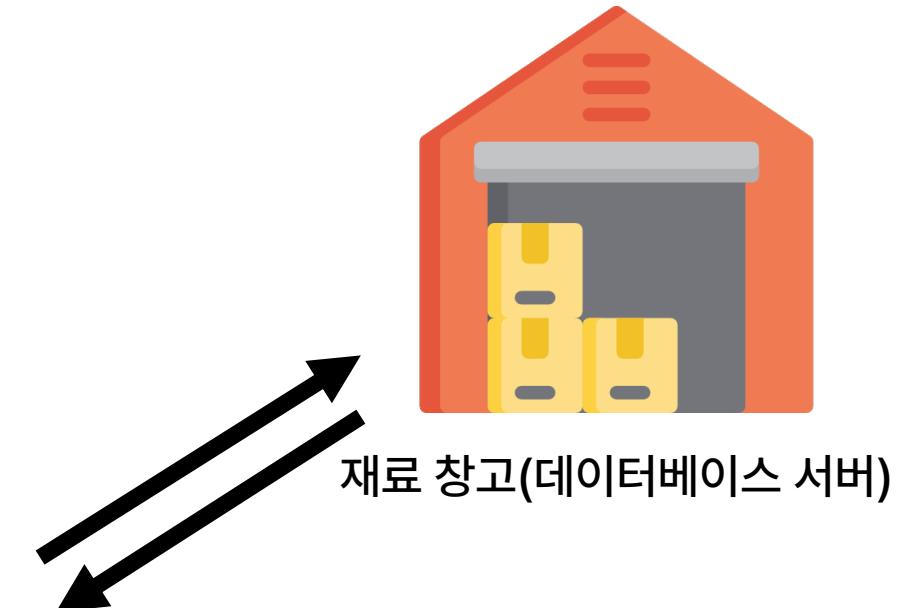
메뉴판(프론트엔드 서버)



직원(백엔드 서버)



# Back-end가 필요한 이유



# Back-end가 필요한 이유



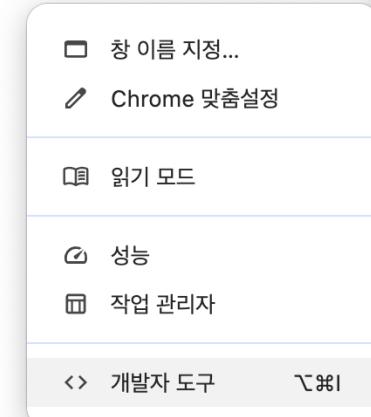


◆ Gemini Algorithm 정석기술연구소 수업 sparsc 삼성 면접 딥러닝 H 토익인강1위 해커스 챔...



Google 검색

I'm Feeling Lucky



새 탭 ⌘T

새 창 ⌘N

새 시크릿 창 ⌘⌘N

Hyeon Su 로그인됨 >

비밀번호 및 자동 완성 >

방문 기록 >

다운로드 ⌘⌘L

북마크 및 목록 >

템 그룹 >

확장 프로그램 >

인터넷 사용 기록 삭제... ⌘⌘⌫

확대/축소 - 100% + ⌘⌘[ ⌘⌘]

인쇄 ⌘P

Google 렌즈로 검색

번역...

찾기 및 수정하기 >

전송, 저장, 공유 >

도구 더보기 >

도움말 >

설정 ⌘,

Network tab in the Chrome DevTools Network panel showing the request and response details for the Google homepage. The timeline shows various requests being made, primarily scripts and images, with most requests completing within 1,000 ms.

Name	Status	Type	Initiator	Size	T...
m=dLOtP?xjs=s4	200	script	m=cdo...	629 B	1...
Ping	200	xhr	rs=AA2YrT...	64 B	1...
log?format=json&hasfast=true&authus...	200	fetch	rs=AA2YrT...	167 B	1...
m=lOO0Vd,sy8a,P6sQOc?xjs=s4	200	script	m=cdos...	758 B	1...
log?format=json&hasfast=true&authus...	200	preflight	Preflight	0 B	8...
RotateCookiesPage?og_pid=538&rot=...	200	document		324 B	1...
gen_204?atyp=i&ct=psnt&cad=&nt=rel...	204	text/html	m=cdos...	25 B	1...
log?format=json&hasfast=true&authus...	200	xhr	rs=AA2YrT...	167 B	1...
m=hfc...	200	script	RotateCookiesPage?og...	6.6 kB	1...
favicon.ico	200	x-icon	Other	1.5 kB	1...
log?format=json&hasfast=true&authus...	200	preflight	Preflight	0 B	8...
gen_204?atyp=csi&r=1&ei=YbTSZ4KqL...	204	ping	m=cdos...	25 B	1...
app?eom=1&awwd=1&origin=https%3A...	200	document	rs=AA2YrT...	15.9 kB	1...
log?format=json&hasfast=true&authus...	200	xhr	rs=AA2YrT...	167 B	1...
m=_b,_tp	200	script	app?eom=1&awwd=1&origi...	73.9 kB	3...
ACg8oclkPxL9FmZB4Xb0b30sfu39glQ...	200	png	app?eom=1&awwd=1&origi...	2.9 kB	1...
p_2x_72023649b67c.png	200	png	app?eom=1&awwd=1&origi...	157 kB	4...
4UaRrENHsxJlGDuGo1OIJfC6I_24rlCK...	200	font	app?eom=1&awwd=1&origi...	52.3 kB	3...
KFOmCnqEu92Fr1Mu4mxKKTU1Kg.woff2	200	font	app?eom=1&awwd=1&origi...	10.8 kB	2...
m=ws9Tlc,n73qwf,GkRiKb,e5qFLc,IZT6...	200	script	m=_b,_tp:392	101 kB	2...
m=p3hmRc,LvGhrf,RqjULd	200	script	m=_b,_tp:392	7.4 kB	1...
m=P6sQOc	200	script	m=_b,_tp:392	740 B	1...
m=Wt6vjf,hhhU8,FCpbqb,WhJNk	200	script	m=_b,_tp:392	1.6 kB	1...

60 requests | 1.2 MB transferred | 3.2 MB resources | Finish: 6.01 s | DOMContentLoaded: 906 ms | Load: 1.86 s

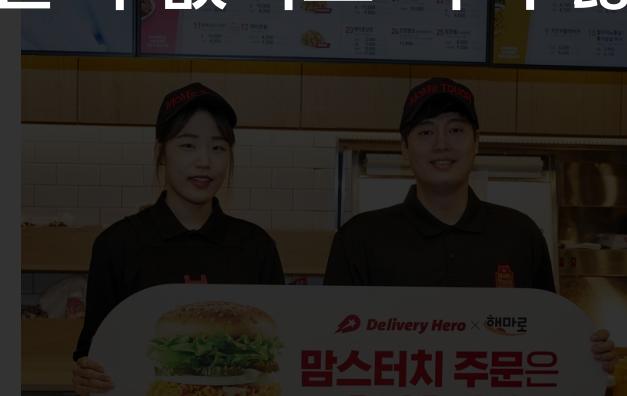
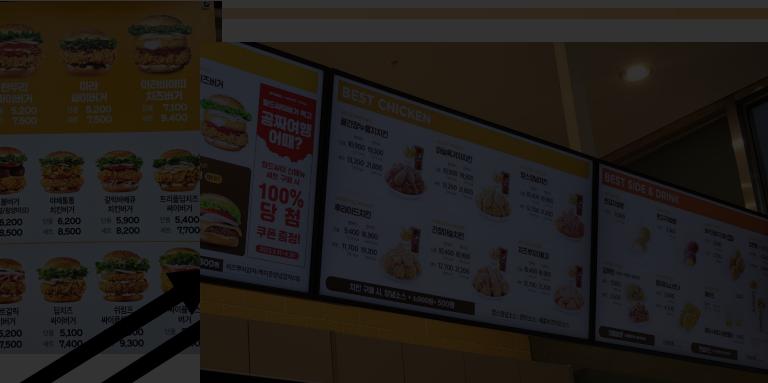
Console What's new AI assistance Issues Coverage Search

# Back-end가 필요한 이유



고객(웹 브라우저)

메뉴판(프론트엔드 서버)  
비효율적이지만  
직원이 메뉴까지 알려주면  
메뉴판이 없어도 되지 않을까?



직원(백엔드 서버)



재료 창고(데이터베이스 서버)

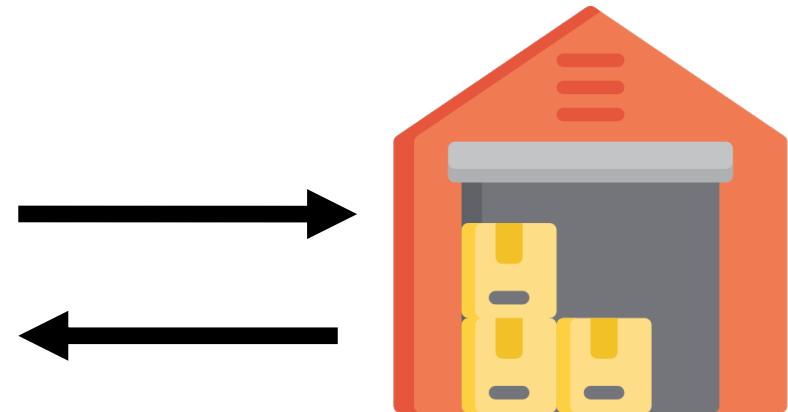
# Front-end와 Back-end의 통합



직원



고객(웹 브라우저)

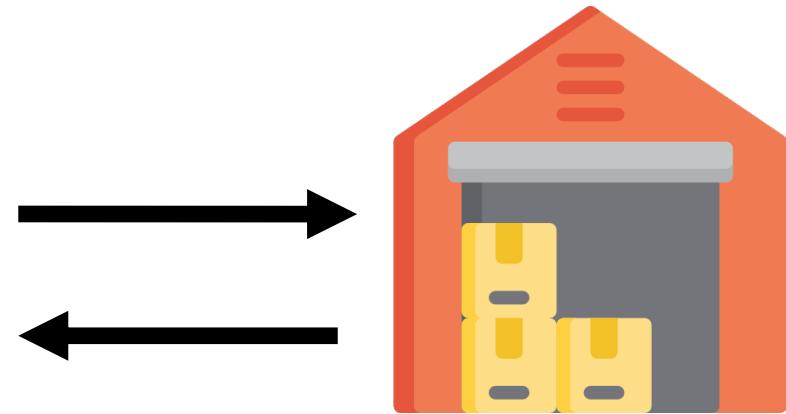


재료 창고(데이터베이스 서버)

# Front-end와 Back-end의 통합



NEXT.js

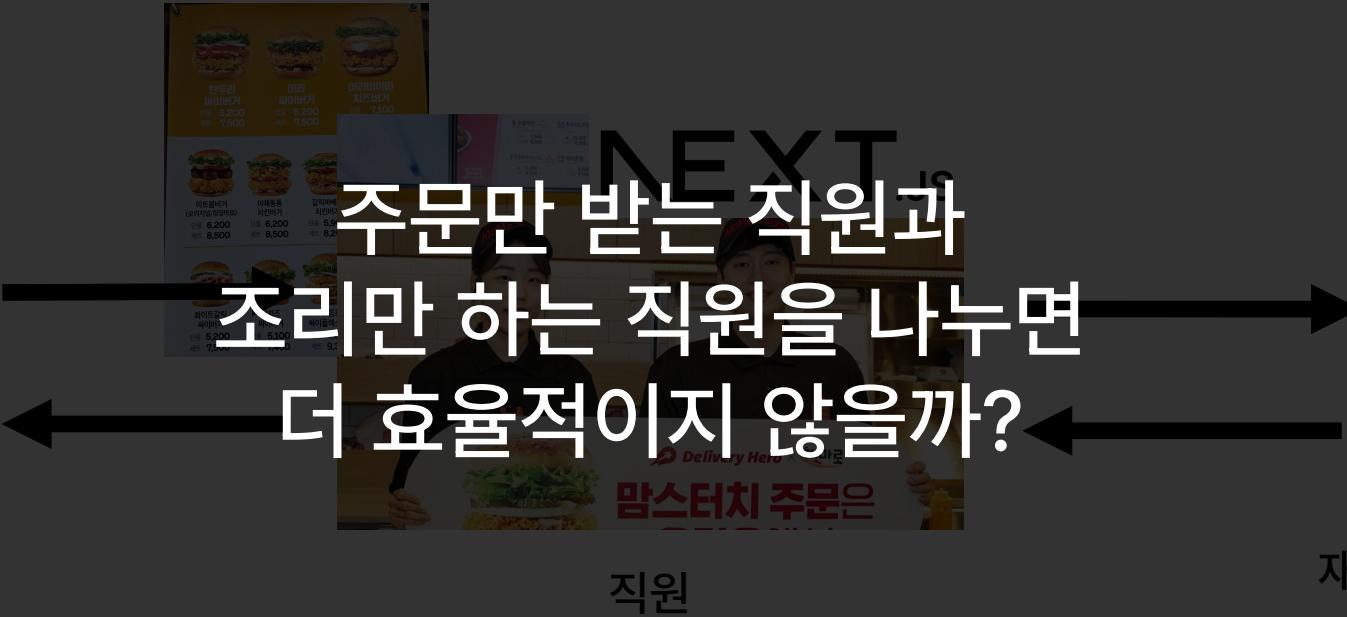


재료 창고(데이터베이스 서버)

# Front-end와 Back-end의 통합



고객(웹 브라우저)



# Front-end와 Back-end의 통합



고객(웹 브라우저)



재료 창고(데이터베이스 서버)



# Front-end와 Back-end의 통합



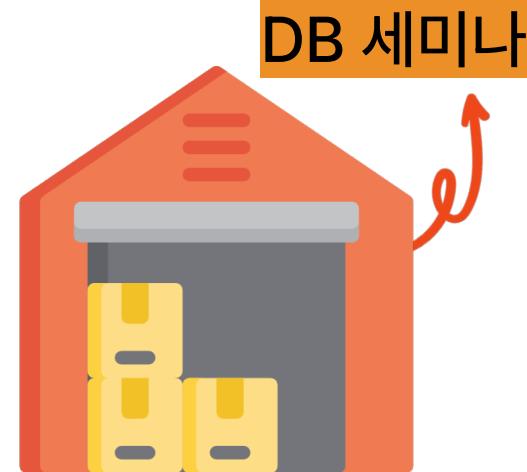
고객(웹 브라우저)



웹 세부과정



백엔드 세부과정

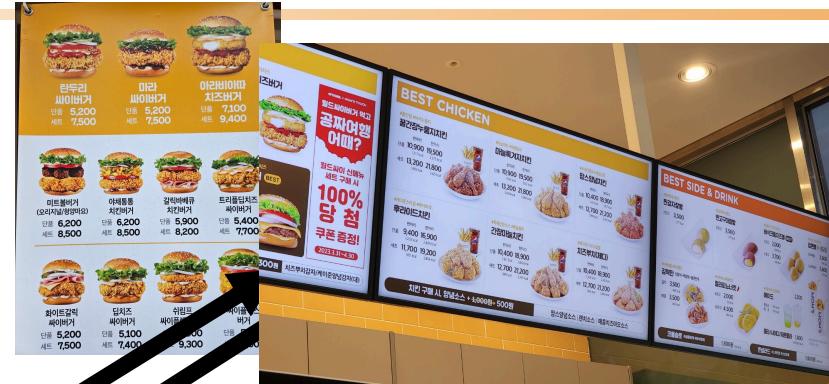


DB 세미나

# 오늘의 범위!



고객(웹 브라우저)



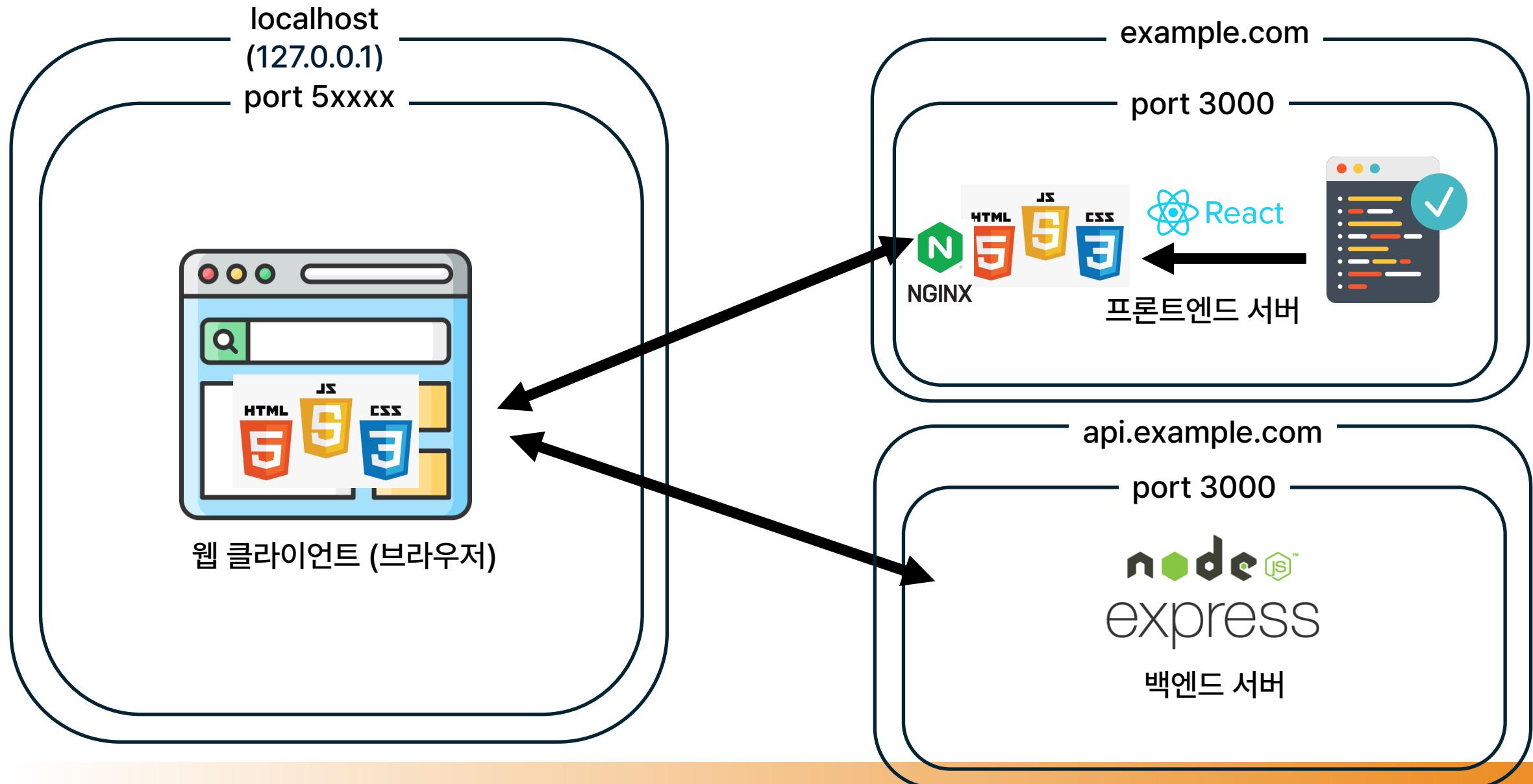
메뉴판(프론트엔드 서버)



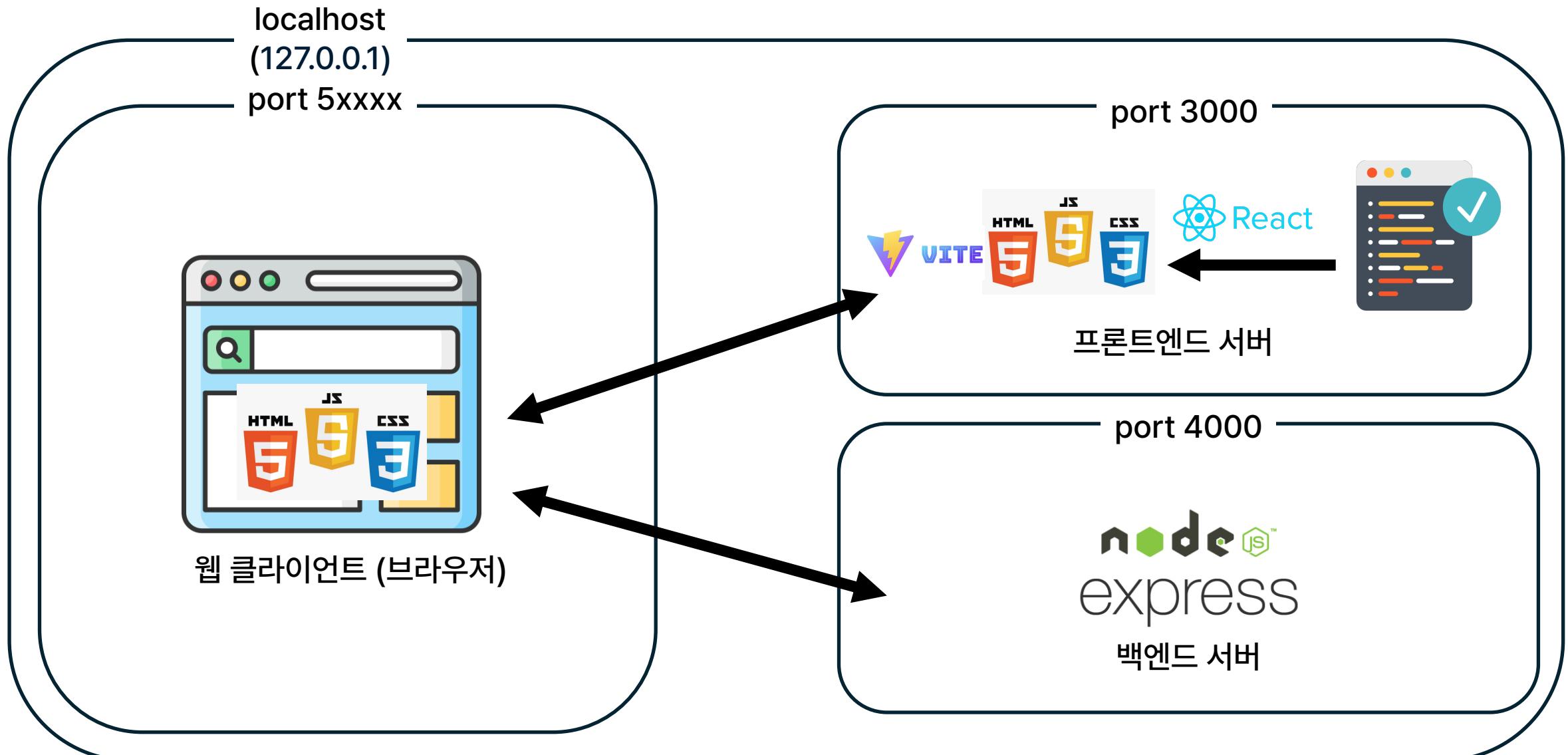
직원(백엔드 서버)

오늘 세미나 주요 내용!

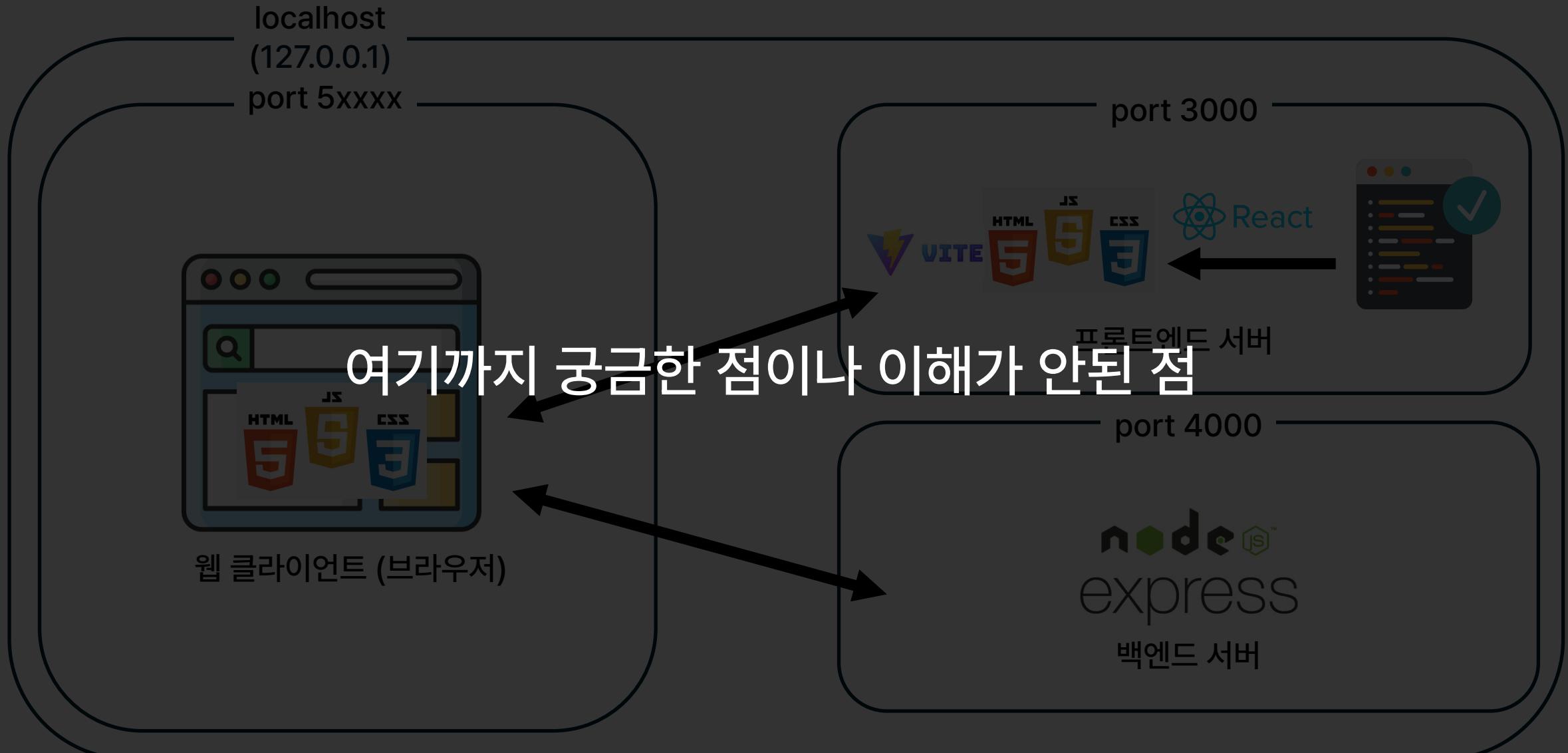
# Production Environment



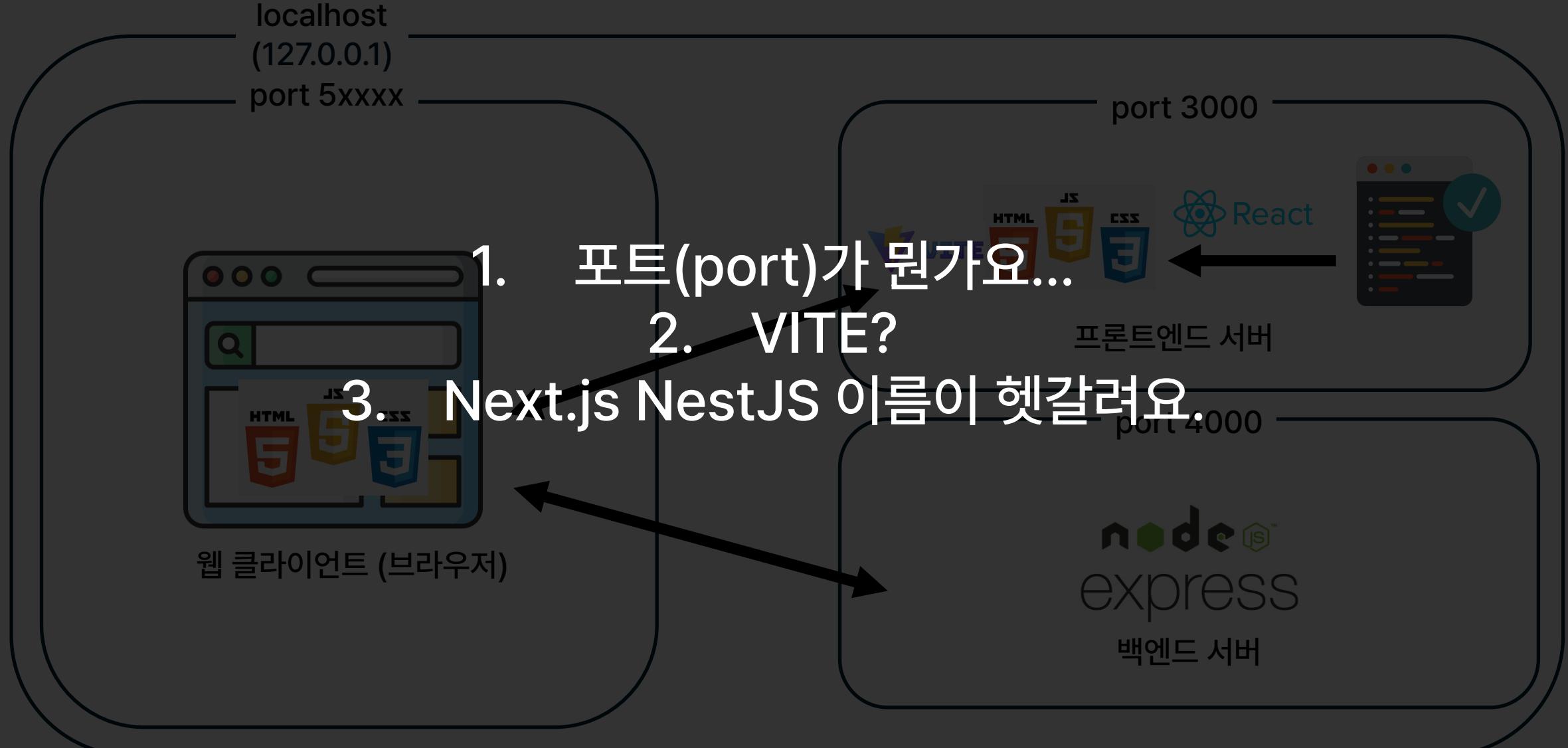
# Development Environment



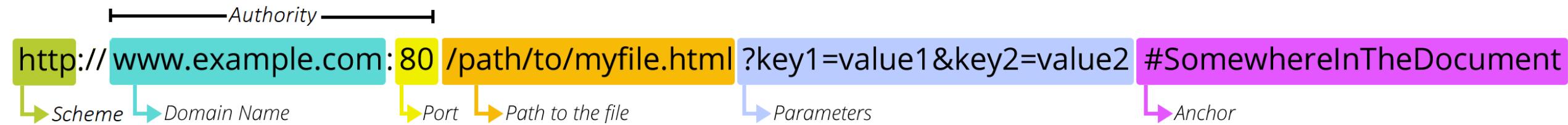
# Development Environment



# Development Environment



# URL



URL이 우편 주소라면

**Scheme** : 사용하려는 우편 서비스

**Domain Name** : 마을

**Port** : 도로명

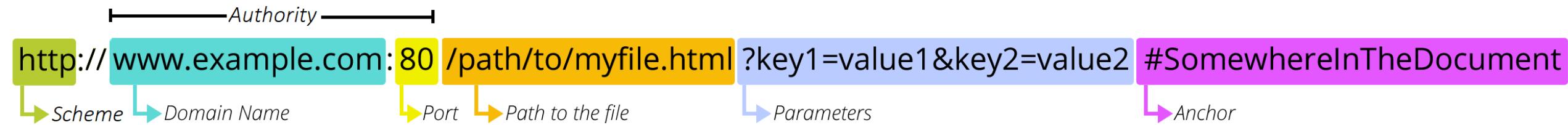
**Path to the file** : 건물

**Parameters** : 건물의 아파트 번호와 같은 추가 정보

**Anchor** : 메일을 보내는 사람



# Scheme



## HTTP (Hyper Transfer Protocol):

메시지를 빠르게 교환하기 위한 프로토콜 일종  
암호화가 안되어 있어, 통신 내용 해독 가능.

## HTTPS (Hyper Transfer Protocol Secure):

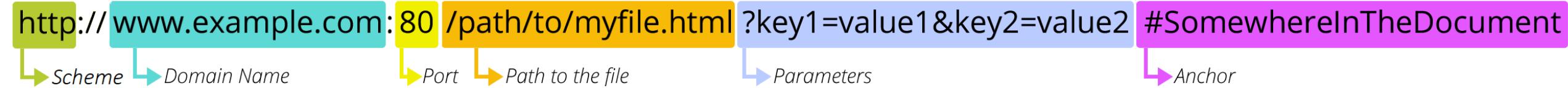
암호화가 되어 있는 HTTP  
현대화된 웹사이트들은 모두 HTTPS 프로토콜 사용.

더 자세히 알고 싶으면 네트워크 개론 수업을 들어보자.

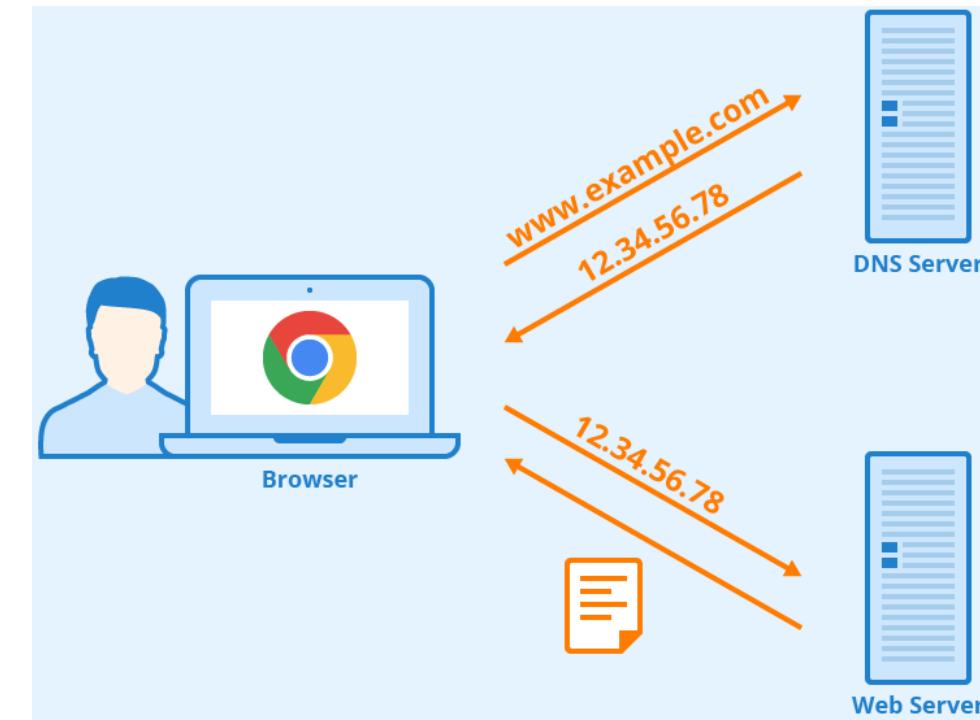


# Domain Name

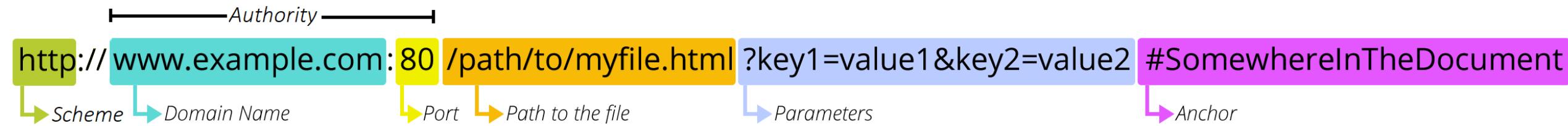
— Authority —



- 도메인 네임(Domain Name)은 IP 주소 대신 사용자가 쉽게 입력할 수 있도록 도와줌.
- 하지만 실제로 서버들은 모두 IP 주소를 기반으로 운영된다.
- ex) 브라우저 주소창에 "142.250.207.110"(IP 주소)를 입력하면, "google.com"(도메인 네임)으로 연결됨.
- "google.com"이 "142.250.207.110"이라는 정보를 DNS(Domain Name System) 서버가 저장하고 관리한다.



# Port



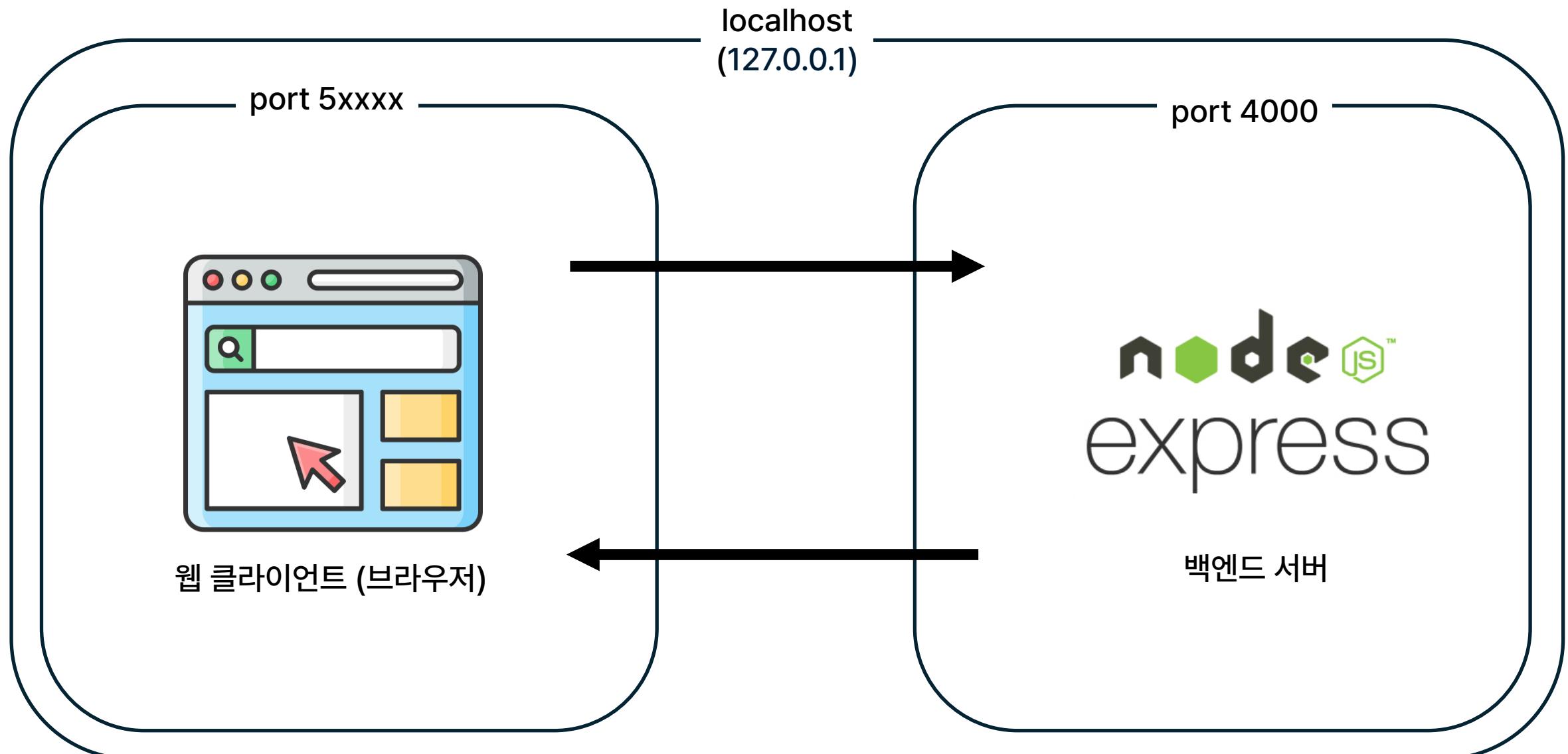
- 보통 IP 주소 하나는 네트워크에 연결된 컴퓨터 기기 하나를 지칭한다.
- Port는 컴퓨터 안에서 돌아가는 프로세스가 가지는 번호
- 즉, www.example.com의 IP 주소에 해당하는 기기의 80번 포트를 가진 프로세스를 연결해주세요라는 뜻

브라우저는 포트 미지정 시 기본적으로 443으로 연결해준다.

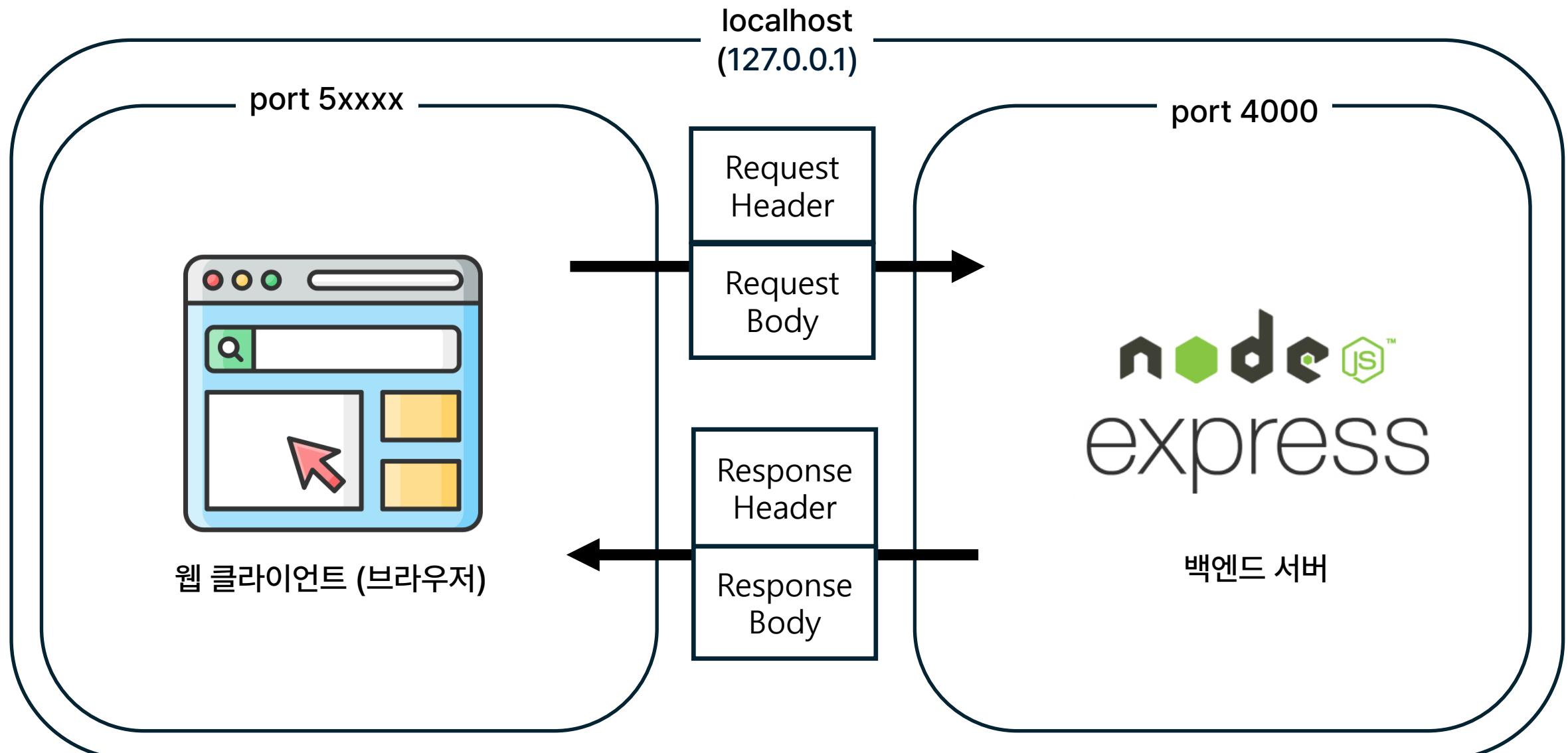
← → G 142.250.207.110  
← → G https://www.google.com  
← → G https://www.google.com:443  
**동일**



# HTTP



# HTTP



# HTTP

각 헤더에 어떤 내용을 담자 정한 규칙이 있다.

브라우저랑 node.js는 똑똑하기 때문에 특정 헤더는 자동으로 생성된다.

## Request Headers

POST /login HTTP/1.1

**Authorization:** Bearer my-jwt-token

**Content-Type:** application/json

**Accept:** application/json

**User-Agent:** Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0

**Referer:** https://example.com

**Origin:** https://example.com

**Cookie:** "mykey1=hello; mykey2=hello2"

## Request Body

```
{  
  "name": "Alice",  
  "email": "alice@example.com",  
  "password": "securepassword"  
}
```

## Response Headers

200 OK

**Access-Control-Allow-Origin:** https://example.com

**Connection:** Keep-Alive

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

**Date:** Wed, 19 Mar 2025 16:06:00 GMT

**Set-Cookie:** mykey1=myvalue; expires=Thu, 19 Mar 2026 16:06:00 GMT; Max-Age=31449600; Path=/; secure

**Set-Cookie:** mykey2=myvalue; expires=Thu, 19 Mar 2026 16:06:00 GMT; Max-Age=31449600; Path=/; secure

**Cache-Control:** no-cache, no-store, must-revalidate

**Expires:** Wed, 21 Oct 2025 07:28:00 GMT

## Request Body

```
{  
  "login": true  
}
```



# HTTP

Request Headers

POST /login HTTP/1.1

**Authorization:** Bearer my-jwt-token

**Content-Type:** application/json

**Accept:** application/json

**User-Agent:** Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0

Referer: <https://example.com>

Origin: <https://example.com>

Cookie: "mykey1=hello; mykey2=hello2"

Request Body

```
{  
  "name": "Alice",  
  "email": "alice@example.com",  
  "password": "securepassword"  
}
```

**POST:** Request Method(GET, POST, PUT, PATCH, DELETE)

/login: Request Path

HTTP/1.1: 프로토콜 종류와 버전

**Authorization:** 인증에 필요한 토큰 값.

**Content-Type:** 클라이언트가 보내는 데이터의 타입

**Accept:** Response Body를 어떤 형식으로 응답 받고 싶은지

**User-Agent:** 요청을 보낸 클라이언트의 정보

Referer: 사용자가 어떤 url에서 요청 보냈는지

**Origin:** 사용자가 어떤 Origin(프로토콜 + 도메인 + 포트)에서 요청 보냈는지

Cookie: 보내는 쿠키 내용



# HTTP

200 OK: Response Status

200: 정상

404: 자원 없음

500: 서버 내부 에러

각 숫자가 가지는 의미가 있음

Headers...

Access-Control-Allow-Origin: CORS 정책 설정

Connection: 연결 방식

Content-Type: 응답 데이터 타입

Date: 서버의 시간

Set-Cookie: 쿠키 내용을 어떻게 저장하고 만료가 언제 될 것인지.

Cache-Control: 요청자가 Response Body를 캐싱할지 여부

Response Headers

200 OK

Access-Control-Allow-Origin: https://example.com

Connection: Keep-Alive

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

Date: Wed, 19 Mar 2025 16:06:00 GMT

Set-Cookie: mykey1=myvalue; expires=Thu, 19 Mar 2026 16:06:00 GMT; Max-Age=31449600; Path=/; secure

Set-Cookie: mykey2=myvalue; expires=Thu, 19 Mar 2026 16:06:00 GMT; Max-Age=31449600; Path=/; secure

Cache-Control: no-cache, no-store, must-revalidate

Expires: Wed, 21 Oct 2025 07:28:00 GMT

Request Body

```
{  
    "login": true  
}
```

# API (Application Programming Interface)

---

- 소프트웨어 간 상호작용을 위한 기능 및 규칙

- 대표적인 API 방식:

- RESTful API
- GraphQL

**RESTful API 특징:** CRUD 형식(기본적인 데이터 조작 방식) 준수

- Create → POST (데이터 생성)
- Read → GET (데이터 조회)
- Update → PUT (데이터 전체를 요청 값으로 수정)
- Update → PATCH (데이터 일부 수정)
- Delete → DELETE (데이터 삭제)

**RESTful API 예시**

- DELETE example.com/image1  
→ image1을 삭제하는 요청



# CORS(Cross Origin Resource Sharing)

---

- 웹 브라우저와 Node.js는 기본적으로 몇몇 헤더를 자동 생성하며, 보안까지 고려함.

- CORS(Cross-Origin Resource Sharing):

- 현재 요청하는 Origin(프로토콜 + 도메인 + 포트)과 요청을 받는 Origin 간의 리소스 공유 보안 정책
- ex) 맘스터치; 브라우저가 보고 있는 메뉴판이 내 가게에서 발급한 메뉴판이 맞는지 확인, 다르면 차단

- 개발 시 발생하는 문제:

- 일반적으로 프론트엔드는 localhost:3000에서 실행됨.
- 백엔드는 localhost:4000에서 요청을 받음.
- 두 Origin이 다르므로, 백엔드에서 별도 설정이 없으면 브라우저가 요청을 차단함.

- 해결 방법:

- 서버 응답 시 Access-Control-Allow-Origin, Access-Control-Allow-Methods, Access-Control-Allow-Headers 헤더를 추가하여 localhost:3000을 허용함.
- CORS 라이브러리를 사용하면 더 쉽게 해결이 가능하다.



# CORS(Cross Origin Resource Sharing)

---

- 웹 브라우저와 Node.js는 기본적으로 몇몇 헤더를 자동 생성하며, 보안까지 고려함.
- CORS(Cross-Origin Resource Sharing):
  - 현재 요청하는 Origin(프로토콜 + 도메인 + 포트)과 요청을 받는 Origin 간의 리소스 공유 보안 정책
- 개발 시 발생하는 문제:  
**여기까지 궁금한 점이나 이해가 안된 점**
  - 일반적으로 프론트엔드는 localhost:3000에서 실행됨.
  - 백엔드는 localhost:4000에서 요청을 받음.
  - 두 Origin이 다르므로, 별도 설정이 없으면 브라우저와 Node.js가 요청을 차단함.
- 해결 방법:
  - 서버 응답 시 Access-Control-Allow-Origin, Access-Control-Allow-Methods, Access-Control-Allow-Headers 헤더를 추가하여 localhost:3000을 허용함.
  - CORS 라이브러리를 사용하면 더 쉽게 해결이 가능하다.



# Review Node.js

---



- Node.js는 자바 스크립트 코드를 브라우저 바깥에서도 실행할 수 있게 해주는 런타임 환경!
- 과거의 자바스크립트는 **브라우저에 돌리기 위한 간단한 코드 언어로 설계됨**
- ⇒브라우저 외부에서 실행할 수 없는 제한이 있었음.
- 하지만 node.js로 외부 환경에서 실행 가능해짐



# Review NVM (Node Version Manager)

다양한 Node.js 버전을 관리하기 위한 도구

## Install

github 레포의 NVM 설치 참조

```
nvm -v # nvm이 설치 되었는지 확인해보자
```

```
nvm ls-remote # 다운 받을 수 있는 모든 Node.js 버전 목록 확인
```

```
nvm install 22 # Node.js 22 버전 다운로드
```

```
nvm use 22 # Node.js 22 버전 사용
```

```
nvm list # 설치 목록
```

```
nvm alias default 22 # 새 터미널 열 때 적용되는 Node.js 버전
```



# 1. run\_nodejs

---

슬랙에 올려놓은 GitHub Repository를 본인 레포에 fork 한 후 clone 하자

```
cd 1.run_nodejs
```

```
python3 02_python.py # window는 `python python_code.py`
```

```
node 01_javascript.js
```



## 2. npm

npm는 node\_module에 있는 패키지 관리  
npx는 패키지에 관련된 특정 명령어를 실행하는 도구

```
# 프로젝트 루트 폴더로 돌아오자
```

```
cd 2.npm
```

```
node index.js # 에러 발생
```

```
npm install # package.json 정보 기반으로 node_module 설치
```

```
node index.js
```

```
npx http-server -y # 패키지 설치하지 않아도 실행 가능! 설치 시 설치한 버전 따라감.
```



# 3. functions

```
3 // Named Function
4 function greet1() {
5   console.log("안녕하세요!");
6 }
7
8 // Anonymous Function
9 > const greet2 = function () {...}
10
11
12
13 // Arrow Function
14 // 화살표 함수 사용 추천. 최근 코드에서 가장 많이 사용됨.
15 const greet3 = () => {
16   console.log("안녕하세요!");
17 }
18
19 greet1();
20
21 greet2();      You, 12 hours ago • function
22
23 greet3();
24
```

Javascript에서 가능한 다양한 함수 선언법  
Javascript도 버전에 따라 문법이 추가됨

## 1. Name Function

모든 JavaScript 버전에서 지원

## 2. Anonymous Function

ECMAScript 3 (ES3, 1999) 이상

## 3. Arrow Function

ECMAScript 6 (ES6, 2015) 이상

완전히 똑같진 않고, 조금씩 차이가 있다.



## 4.event\_loop

```
# 프로젝트 루트 폴더로 돌아오자  
cd 3.event_loop  
node 01_event.js
```

```
1   console.log("1");  
2  
3   setTimeout(() => {  
4       console.log("3");  
5   }, 1000);  
6  
7   console.log("2");  
8   |
```

Node.js는 병렬적으로 코드를 실행할 수 있는 기능을 제공한다.

실행 결과는 1 2 3

setTimeout 함수 안의 내용은 1초 후에 병렬적으로 실행된다.



## 4.event\_loop

---

node 02\_event.js

```
1  console.log("1");
2
3  setTimeout(() => {
4      |  console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

앵?

Node.js는 병렬적으로 작동할 수 있다면서요?!

근데 왜 2가 정확히 1초 후에 출력이 안되요???



## 4.event\_loop

node 02\_event.js

```
1  console.log("1");
2
3  setTimeout( () => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

앵?

Node.js는 병렬적으로 작동할 수 있다면서요?!

근데 왜 2가 정확히 1초 후에 출력이 안되요???

Node.js가 싱글 쓰레드라는 개념을 알아야 한다.

→ 사실은 싱글 스레드다.

→ Event Loop 구조로 비동기 처리를 "병렬처럼" 보이게 함



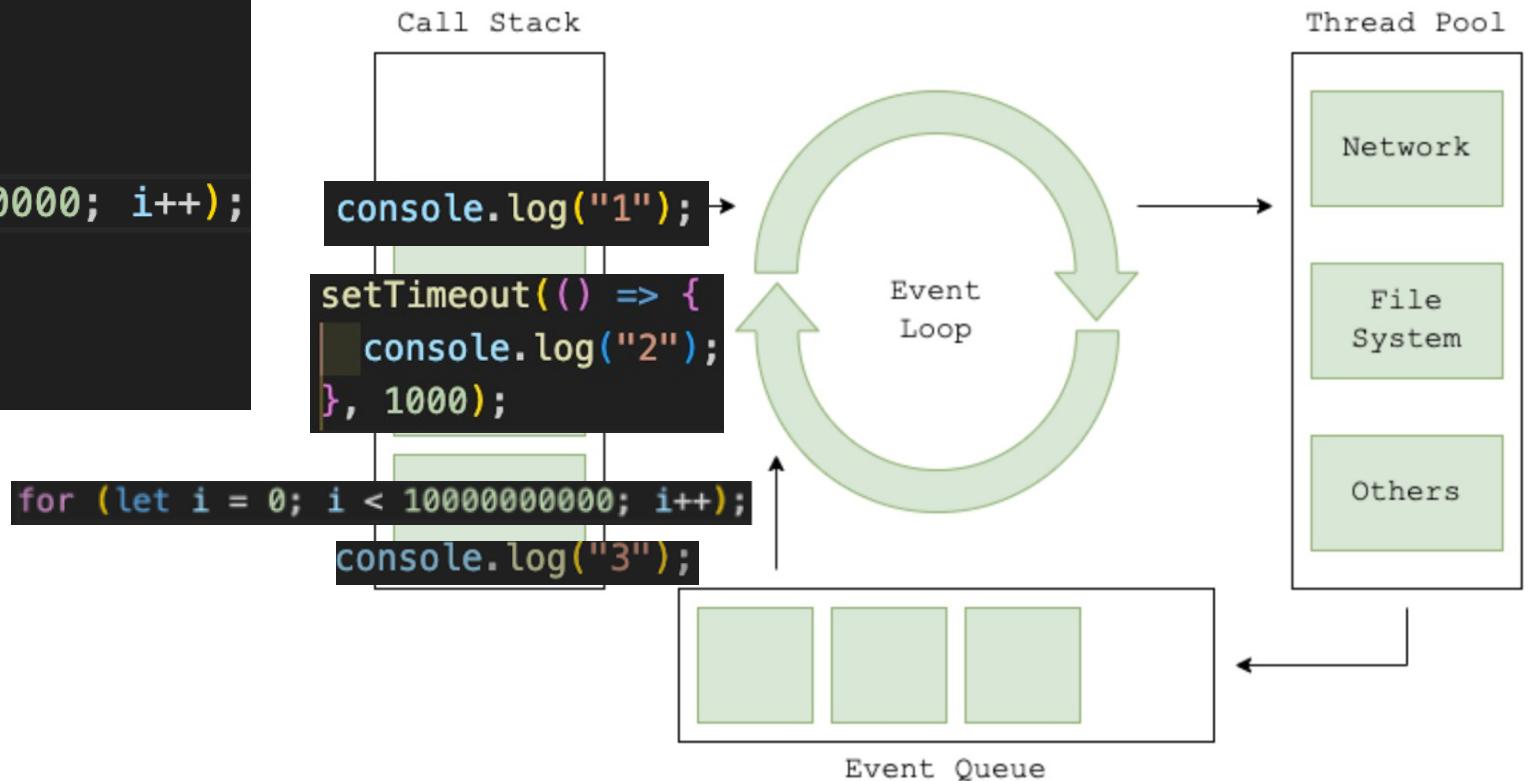
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout(() => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

## Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



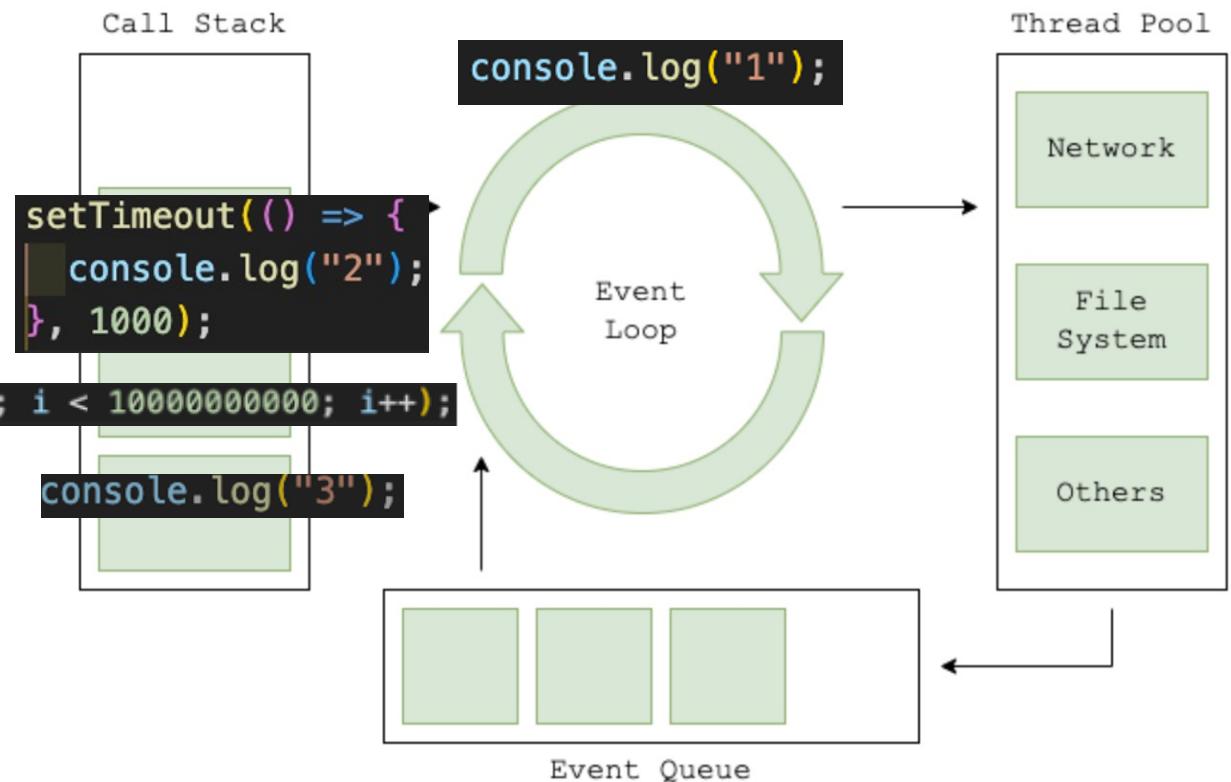
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout(() => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

## Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



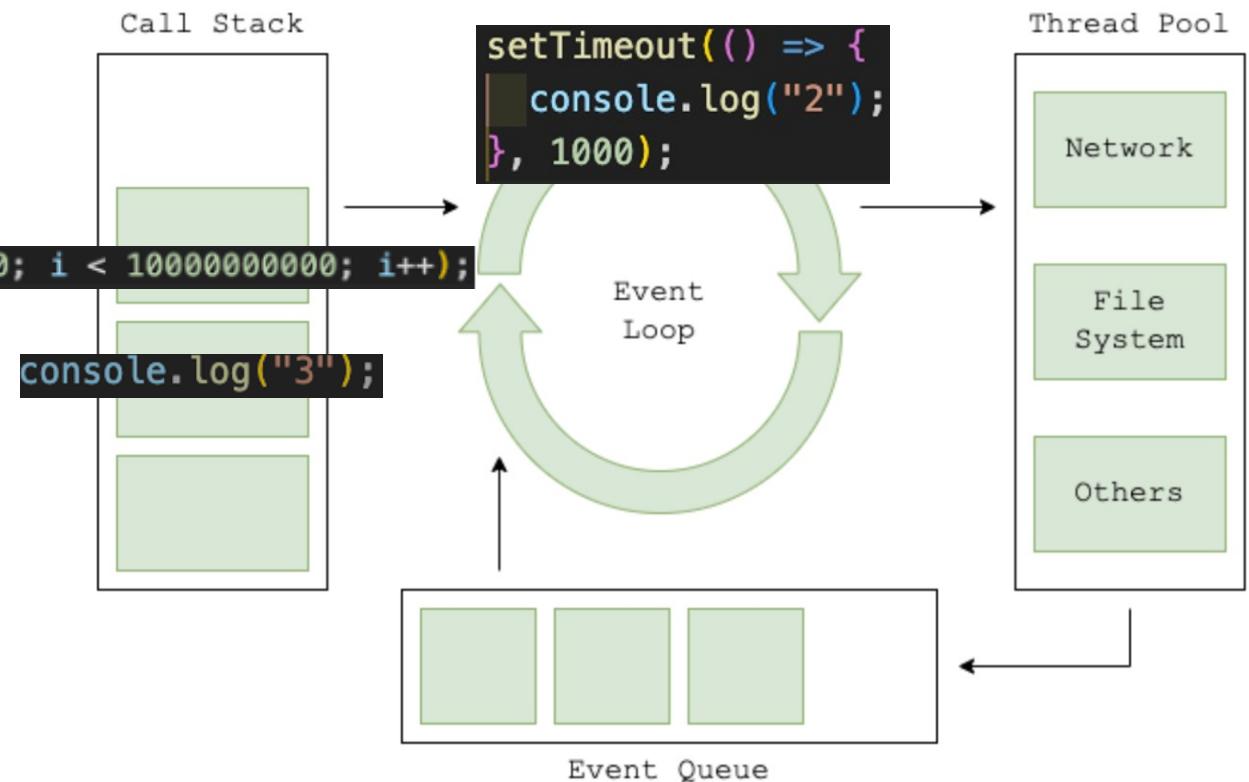
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout(() => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8          for (let i = 0; i < 10000000000; i++);
9  console.log("3");
10
```

## Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



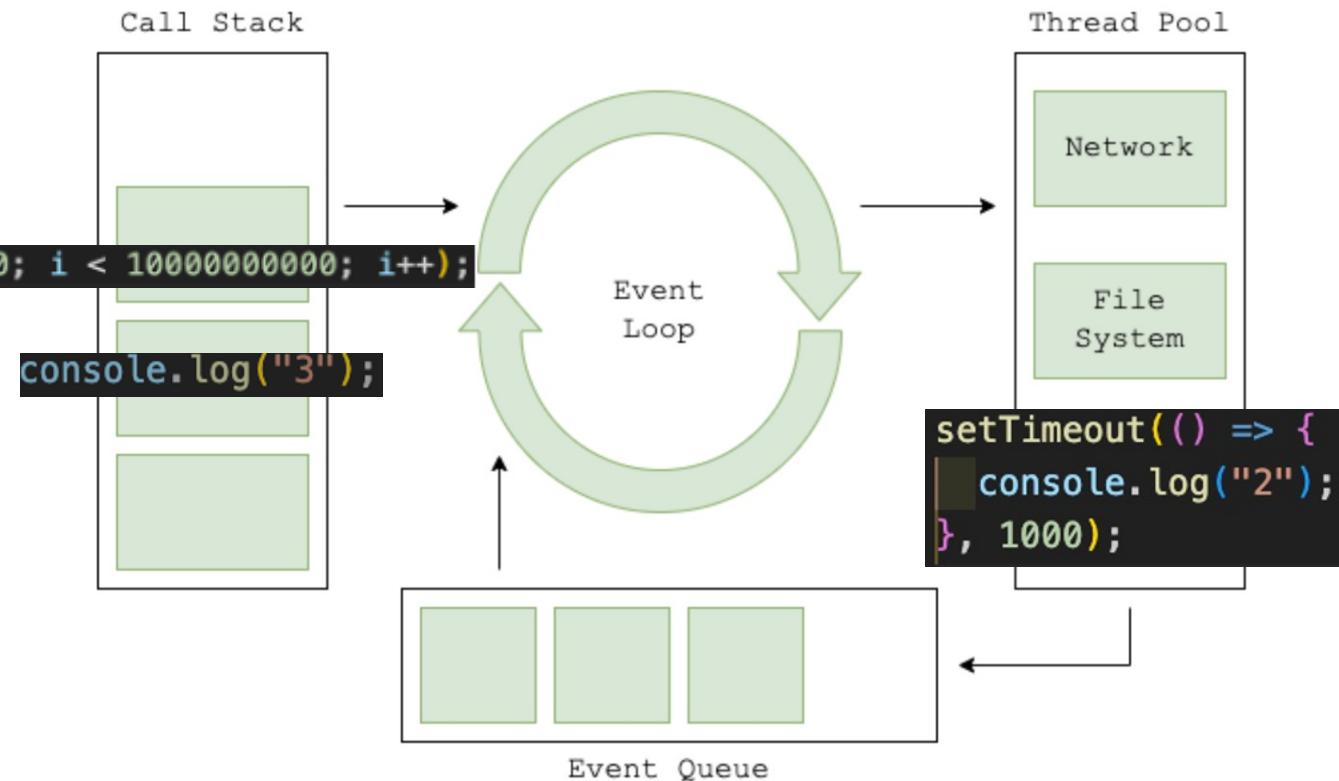
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout(() => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8          for (let i = 0; i < 10000000000; i++);
9  console.log("3");
10
```

## Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



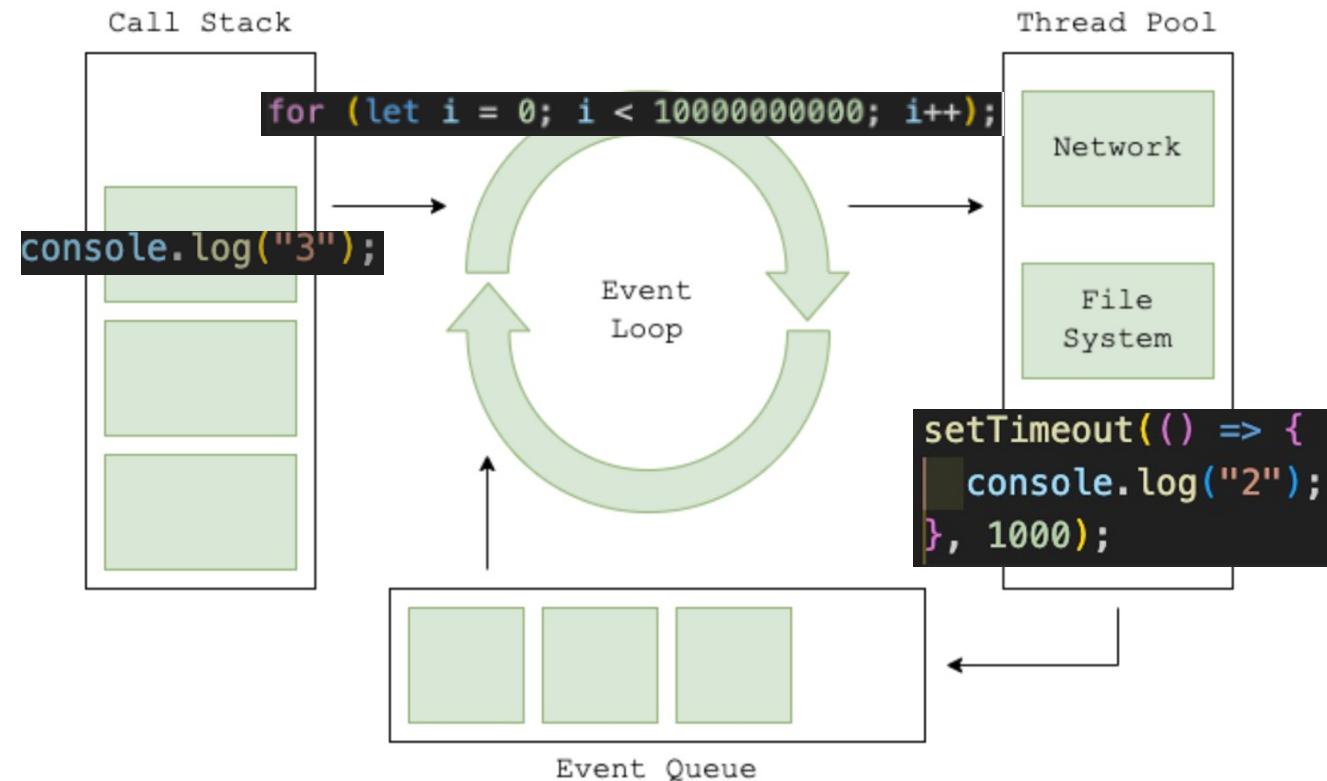
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout( () => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

## Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



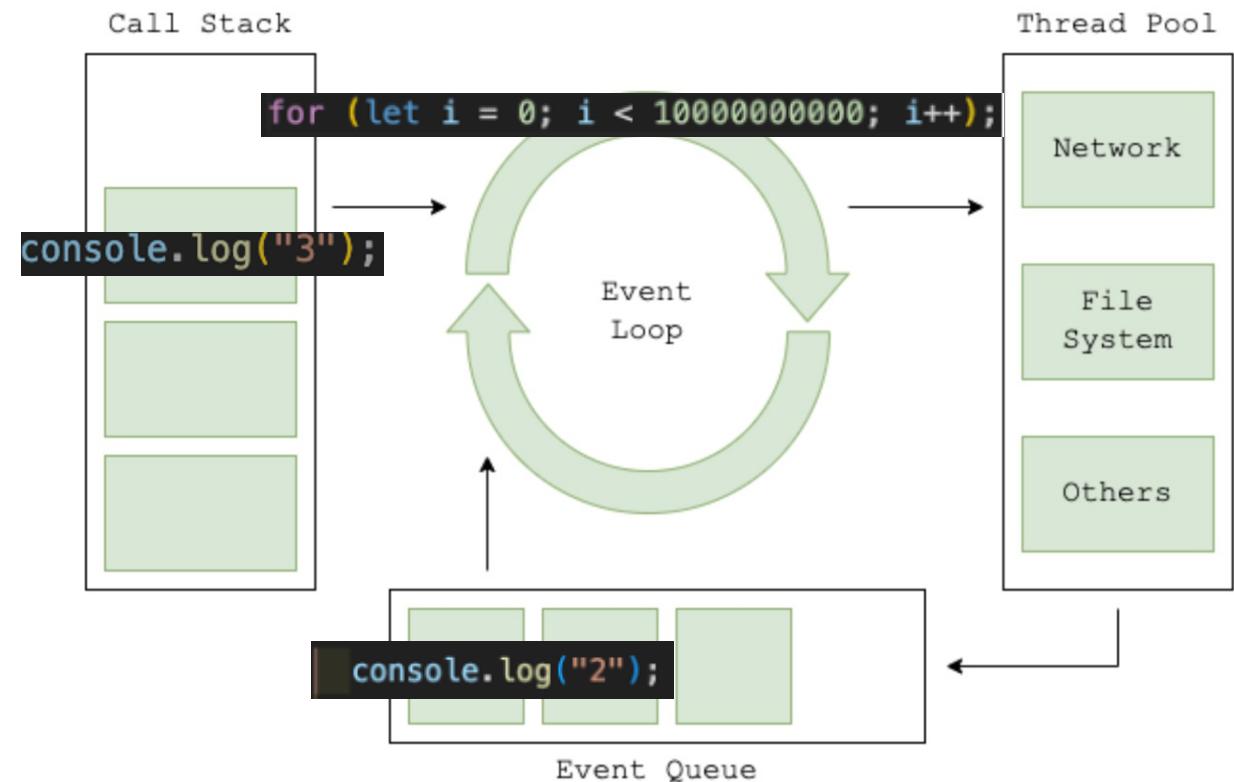
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout( () => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

## Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



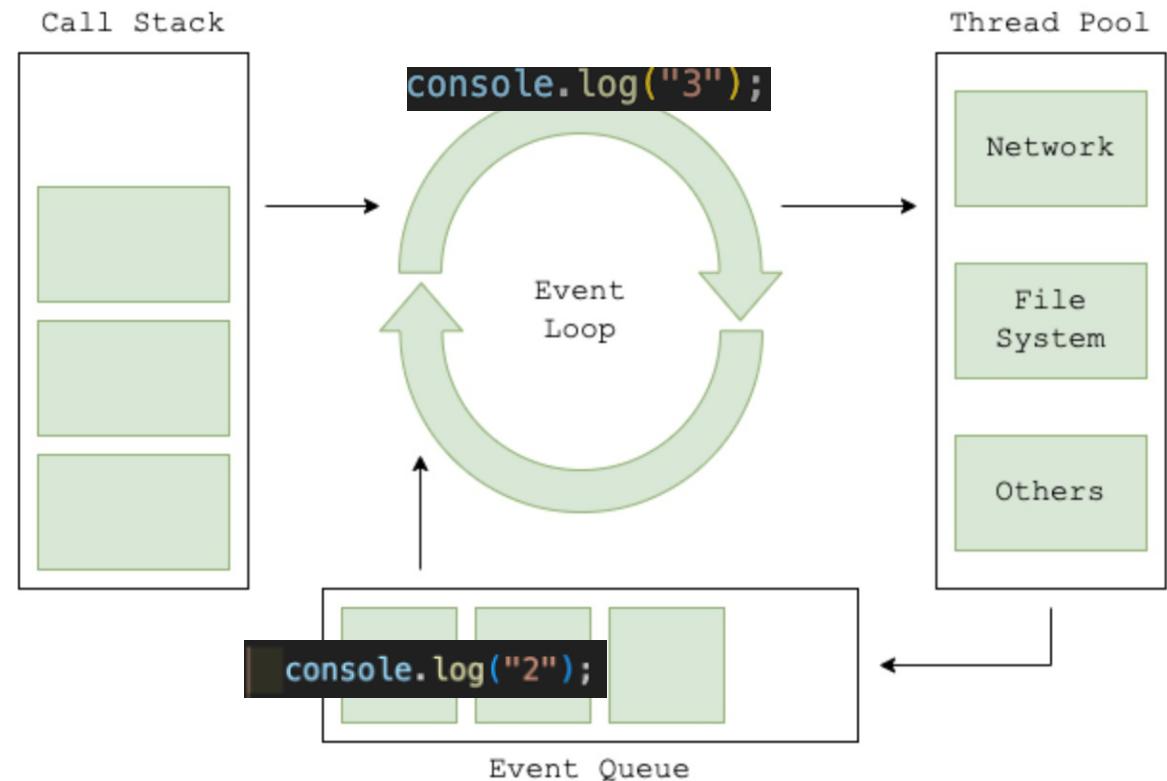
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout( () => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

### Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



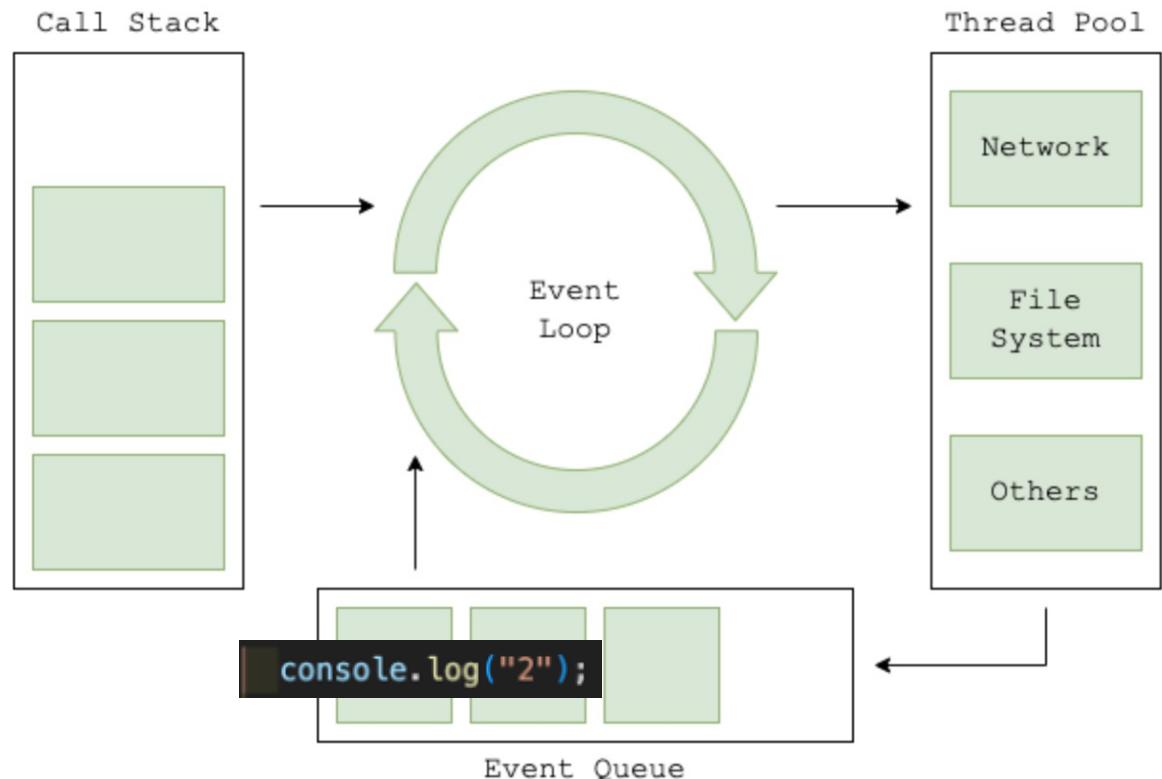
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout( () => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

### Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



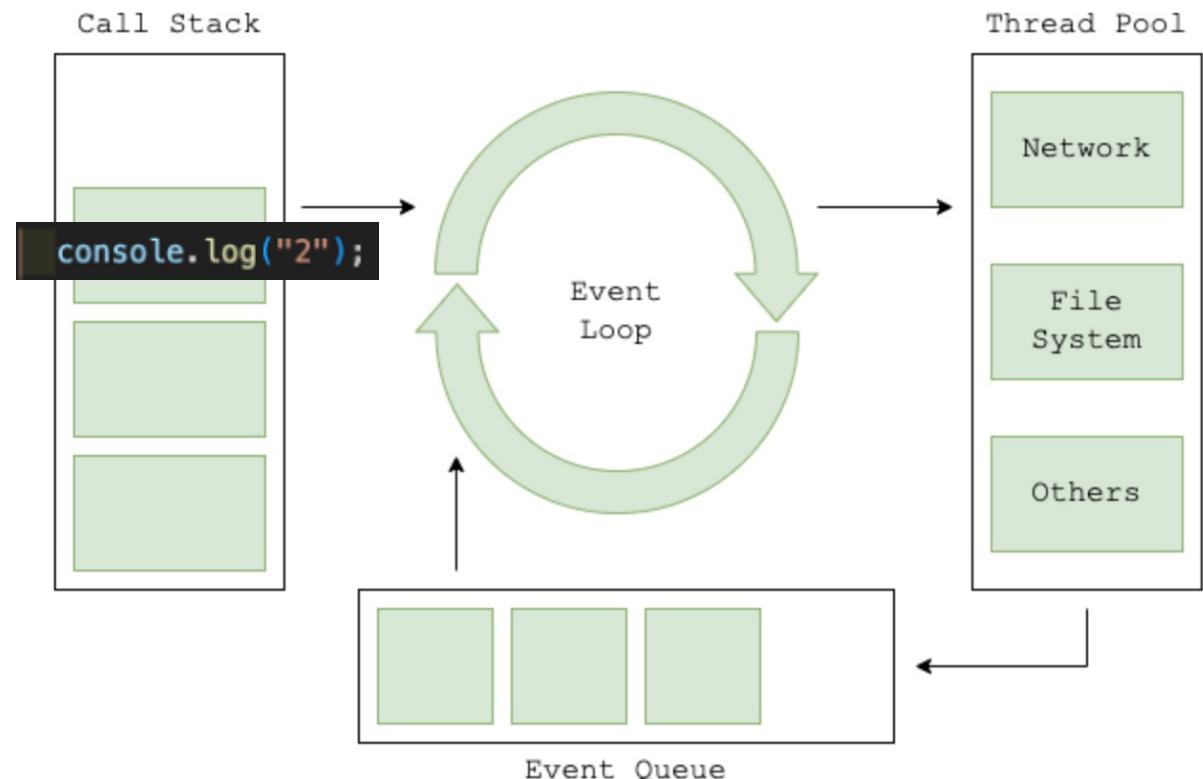
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout( () => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

### Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



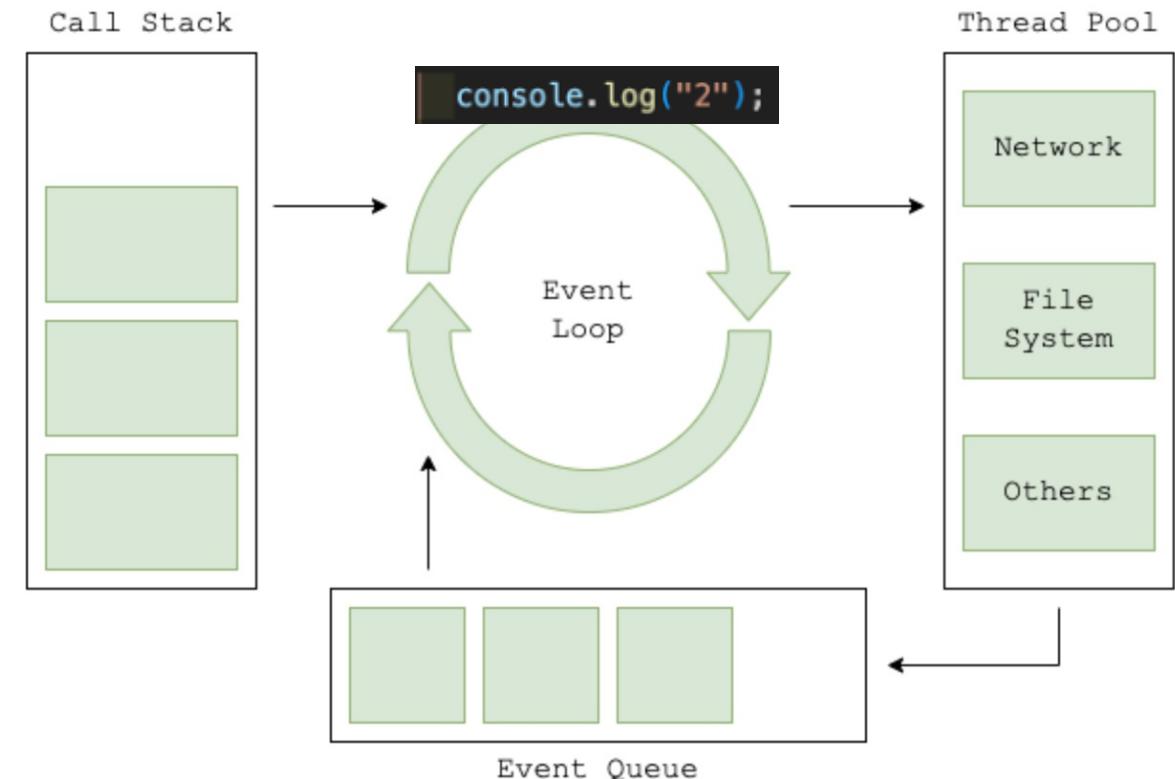
# 4.event\_loop

## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout( () => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

### Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



# 4.event\_loop

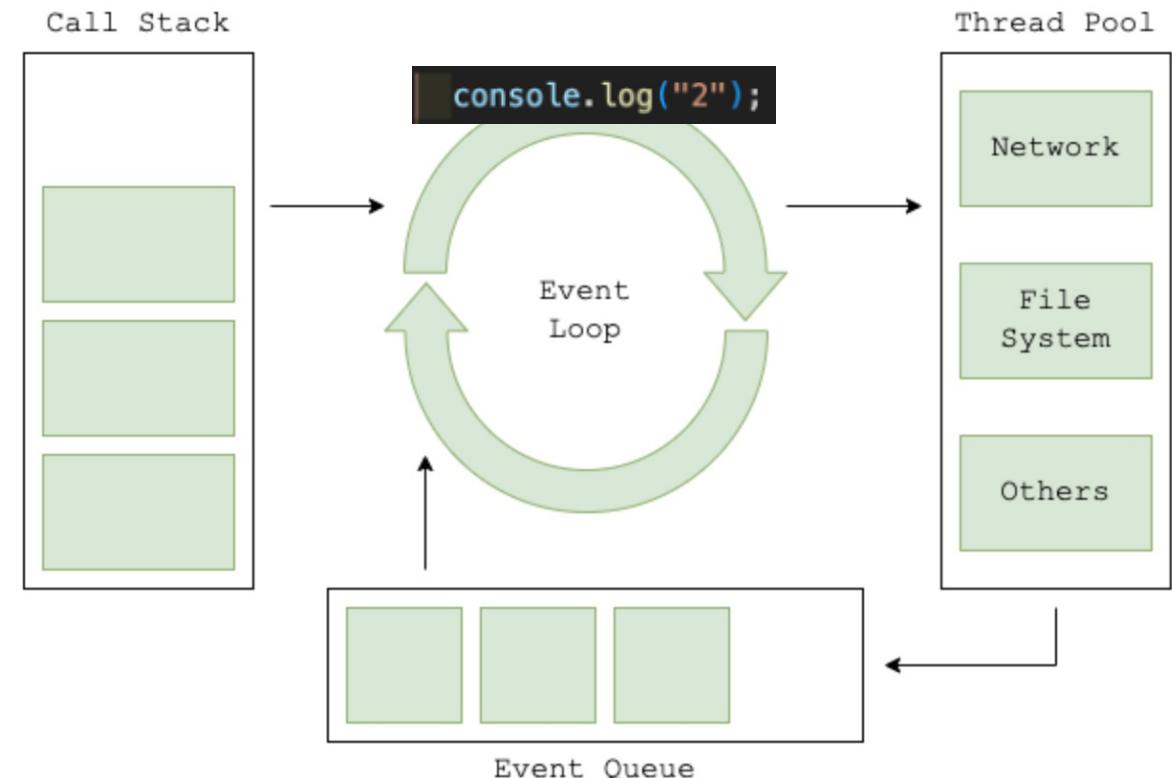
## node 02\_event.js

```
1  console.log("1");
2
3  setTimeout( () => {
4      console.log("2");
5  }, 1000);
6
7  for (let i = 0; i < 10000000000; i++);
8
9  console.log("3");
10
```

**fs.readFileSync()** 같은 시간이 오래 걸리는  
**블로킹 함수**들은 사용에 주의해야 한다.  
⇒ 해당 함수가 실행되는 동안 다른 코드가 실행이 안되  
기 때문!

## Node.js는 싱글 쓰레드

1. Call Stack의 top을 먼저 꺼내 실행한다.
2. Call Stack이 비워지면 Event Queue의 front를 Call Stack에 Push.



## 4. Synchronous/Asynchronous, Blocking/Non-Blocking

---

- Synchronous/Asynchronous는 논리적으로 함수 실행 순서(흐름)에 관한 용어
- Blocking/Non-Blocking은 실제 하드웨어(CPU) 레벨에서 메커니즘에 관한 용어

Example)

야구공 3개(A,B,C)를 순서대로 친구에게 던졌다.

거리가 멀어서 아직 친구가 받질 못하고 공 세 개가 날아가는 중.

A, B, C 순서대로 친구가 받는다면, Synchronous

A, B, C 순서에 상관 없이 친구가 받는다면, Asynchronous

A 때문에 B,C가 날아가다가 멈춘다면, Blocking

A와 상관없이 B,C가 잘 날아간다면, Non-Blocking



# 5. callback\_promise

"callback function은 함수를 파라미터로 받아서 다음 작업으로 실행하는" 코드

```
1  const fetchData = (job, callback) => {
2    setTimeout(() => {
3      callback(job + "을 마쳤습니다.");
4    }, 1000);
5  };
6
7 const main = () => {
8   console.log("1. 데이터 요청 중...");
9
10 fetchData("2. 데이터 요청", (result1) => {
11   console.log(result1);
12
13   fetchData("3. 데이터 전송", (result2) => {
14     console.log(result2);
15
16     fetchData("4. 데이터 검토", (result2) => {
17       console.log(result2);
18     });
19   });
20 }
21
22 main();
23
24 |
```

콜백 지옥

```
1  function hell(win) {
2    // for listener purpose
3    return function() {
4      loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
5        loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
6          loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
7            loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
8              loadLink(win, REMOTE_SRC+'/lib/underscore.min.js', function() {
9                loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
10               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
11                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
12                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
13                     async.eachSeries(SCRIPTS, function(src, callback) {
14                       loadScript(win, BASE_URL+src, callback);
15                     });
16                   });
17                 });
18               });
19             });
20           });
21         });
22       });
23     });
24   });
25 }
```



# 5. callback\_promise

Promise를 사용하는 것이 좀 더 현대적인 문법  
return 한 Promise는 “`await`”로 순차적 흐름 제어가 가능하다.

```
1  const fetchData = (job, callback) => {
2    setTimeout(() => {
3      callback(job + "을 마쳤습니다.");
4    }, 1000);
5  };
6
7  const main = () => {
8    console.log("1. 데이터 요청 중...");
9
10   fetchData("2. 데이터 요청", (result1) => {
11     console.log(result1);
12
13     fetchData("3. 데이터 전송", (result2) => {
14       console.log(result2);
15
16     fetchData("4. 데이터 검토", (result2) => {
17       console.log(result2);
18     });
19   });
20 };
21
22 main();
```

```
1  const fetchData = (job) => {
2    return new Promise((resolve, reject) => {
3      setTimeout(() => {
4        resolve(job + "을 마쳤습니다.");
5      }, 1000);
6    });
7
8
9  const main = async () => {
10   console.log("1. 데이터 요청 중...");
11
12   const result1 = await fetchData("2. 데이터 요청");
13   console.log(result1);
14
15   const result2 = await fetchData("3. 데이터 전송");
16   console.log(result2);
17
18   const result3 = await fetchData("3. 데이터 검토");
19   console.log(result3);
20 };
21
22 main();
```



## 6.typescript



### TypeScript란?

- 자바스크립트에 "타입(type)"을 추가한 언어. (.ts 확장자 사용)
- JavaScript의 상위 집합 (Superset)
- 결과적으로는 타입스크립트를 실행하려면 자바스크립트 파일로 변환하고 자바스크립트 파일을 실행 해야함.

### 왜 TypeScript를 사용할까?

- JavaScript는 유연하지만 실행 중 에러가 너무 늦게 터짐!
- 코드를 실행하기(런타임) 전에 에러를 찾을 수 없을까?
- 로직 측면에서의 문제는 못 찾더라도 타입적인 에러를 잡을 수 있으면 전체 에러가 감소할 것 같다.



# 6.typescript (기본 설정)

```
# 프로젝트 루트 폴더로 돌아오자
```

```
cd 6.typescript
```

```
npm init -yes # package.json 생성
```

```
npm install typescript ts-node
```

```
npx tsc --init # tsconfig.json 생성
```

- ts 파일을 js 파일로 변환하기 위해서는 “**tsconfig.json**” 파일이 필요하다.
- “**tsconfig.json**” 파일은 ts 파일을 어떻게 js 파일로 변환할 지에 대한 설정들을 담고 있다.

The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays a project structure under '9. BACKEND-SEMINAR'. The '6.typescript' folder is expanded, showing files like '01\_interface.ts', '02\_type.ts', etc., and 'tsconfig.json'. The Editor pane on the right shows the content of the 'tsconfig.json' file:

```
1  {
2   "compilerOptions": {
3     /* Visit https://aka.ms/tsconfig to read more about
4      compiler options. */
5     /* Projects */
6     // "incremental": true,
7     // compilation of projects. */
8     // "composite": true,
9     // be used with project references. */
10    // "tsBuildInfoFile": "./tsbuildinfo",
11    // compilation file. */
12    // "disableSourceOfProjectReferenceRedirect": true,
13    // files when referencing composite projects. */
14    // "disableSolutionSearching": true,
15    // when editing. */
16    // "disableReferencedProjectLoad": true,
```

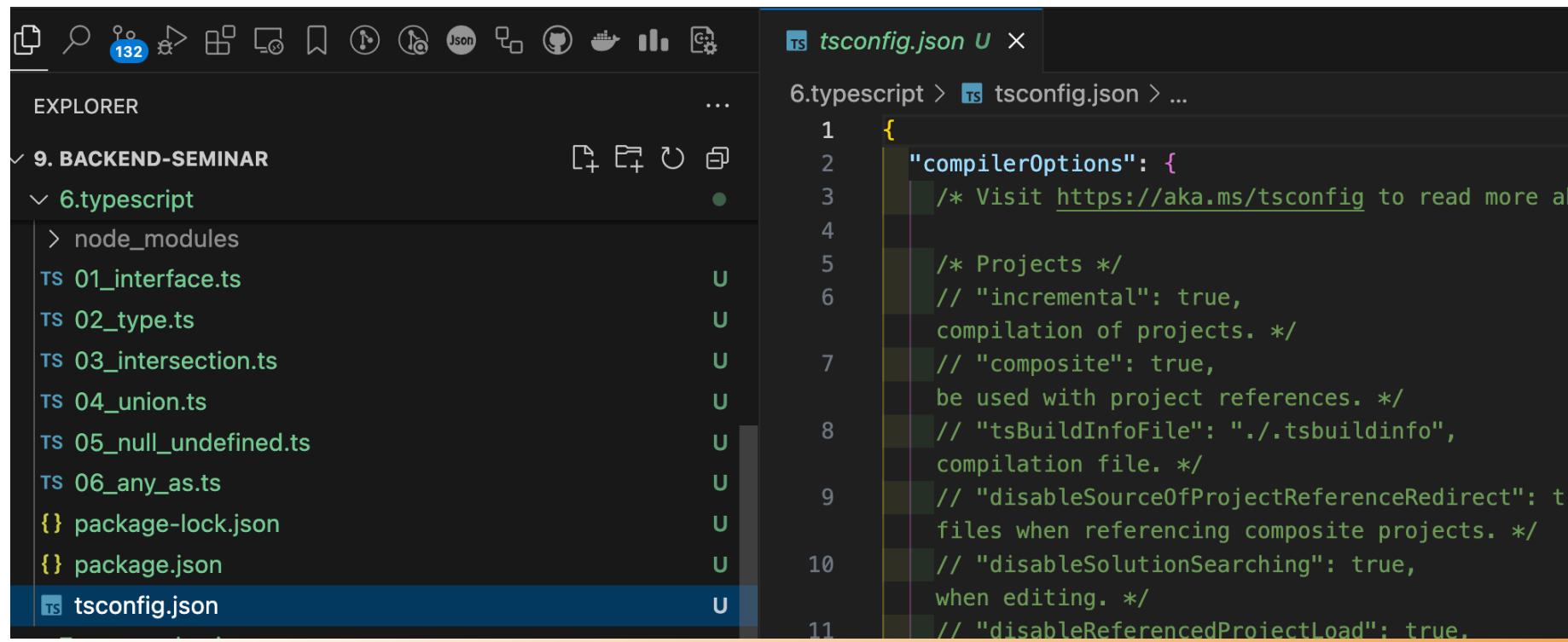
# 6.typescript (ts 실행법)

```
npx tsc # ts→js  
node 실행파일이름.js
```

# 또는

```
npx ts-node 실행파일이름.ts
```

- ts 파일을 실행하는 방법 2가지



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays a project structure under '9. BACKEND-SEMINAR'. Inside the '6.typescript' folder, there are several TypeScript files: '01\_interface.ts', '02\_type.ts', '03\_intersection.ts', '04\_union.ts', '05\_null\_undefined.ts', '06\_any\_as.ts', and 'package-lock.json', 'package.json'. The 'tsconfig.json' file is currently selected and open in the main editor area. The code in 'tsconfig.json' is as follows:

```
{  
  "compilerOptions": {  
    /* Visit https://aka.ms/tsconfig to read more about compiler options.  
     * Projects */  
    // "incremental": true,  
    // compilation of projects. */  
    // "composite": true,  
    // be used with project references. */  
    // "tsBuildInfoFile": "./tsbuildinfo",  
    // compilation file. */  
    // "disableSourceOfProjectReferenceRedirect": true,  
    // files when referencing composite projects. */  
    // "disableSolutionSearching": true,  
    // when editing. */  
    // "disableReferencedProjectLoad": true,  
  }  
}
```

# 6.typescript (type vs interface)

```
1  export {};
2
3  interface MyInterface1 {
4    a: string;
5  }
6  interface MyInterface2 {
7    b: number;
8  }
9  interface MyInterface2 extends MyInterface1 {
10   c: boolean;
11 }
12
13 const myFunction = (a: MyInterface2) => {
14   console.log(`a.a is ${a.a}, a.b is ${a.b} and a.c is ${a.c}!`);
15 };
16
17 myFunction({ a: "Hello", b: 42, c: true });
```

```
1  export {};
2
3  type MyType1 = number;
4  type MyType2 = {
5    a: string;
6    b: MyType1;
7  };
8
9  const myFunction = (a: MyType2) => {
10   console.log(`a.a is ${a.a} and a.b is ${a.b}!`);
11 };
12
13 myFunction(); // Error!
14 myFunction({ a: "Hello", b: 0 }); // OK
```

## interface

- 객체의 구조(Shape)를 정의할 때 사용
- 객체지향 스타일 개발에 익숙한 사람에게 친숙

## type

- 더 넓은 범위: 객체, 유니언, 튜플 등 모든 타입에 사용
- 타입에 별명(Alias)을 붙이는 느낌



## 6.typescript (01\_interface.ts)

```
npx ts-node 01_interface.ts
```

```
1  export {};
2
3  interface MyInterface1 {
4    a: string;
5  }
6  interface MyInterface2 {
7    b: number;
8  }
9  interface MyInterface2 extends MyInterface1 {
10   c: boolean;
11 }
12
13 const myFunction = (a: MyInterface2) => {
14   console.log(`a.a is ${a.a}, a.b is ${a.b} and a.c is ${a.c}!`);
15 };
16
17 myFunction({ a: "Hello", b: 42, c: true });
```

타입을 선언하는 방법 1



## 6.typescript (02\_type.ts)

```
npx ts-node 02_type.ts
```

```
1  export {};
2
3  type MyType1 = number;
4  type MyType2 = {
5    a: string;
6    b: MyType1;
7  };
8
9  const myFunction = (a: MyType2) => {
10    console.log(`a.a is ${a.a} and a.b is ${a.b}!`);
11  };
12
13 myFunction(0); // Error!
14 myFunction({ a: "Hello", b: 0 }); // OK
```

타입을 선언하는 방법 2



## 6.typescript (03\_declararion\_merging.ts)

**npx ts-node 03\_declarati**

```
You, 15 hours ago | 1 author (You)
1 export {};
2
3 You, 15 hours ago | 1 author (You)
4 interface MyInterface1 {
5   a: number;
6   b: string;
7 }
8 You, 15 hours ago | 1 author (You)
9 interface MyInterface1 {
10   b: string;
11   c: boolean;
12 }
13 const myFunction = (a: MyInterface1) => {
14   console.log(`a.a is ${a.a} and a.b is ${a.b}! and a.c is ${a.c}`);
15 };
16
17 myFunction({ a: 1, b: "Hello" }); // Error! This is MyType1
18 myFunction({ b: "Hello", c: true }); // Error! This is MyType2
19 myFunction({ a: 1, b: "Hello", c: true }); // OK
20 You, 5 days ago
```

- 같은 이름의 interface를 선언하면 정의가 합쳐 진다.



# 6.typescript (04\_intersection.ts)

# npx ts-node 04\_intersection.ts

```
You, 15 hours ago | 1 author (You)
1   export {};
2
3   You, 15 hours ago | 1 author (You)
4   interface MyInterface1 {
5     a: number;
6     b: string;
7   }
8
9   You, 15 hours ago | 1 author (You)
10  interface MyInterface1 {
11    b: string;
12    c: boolean;
13  }
14
15  const myFunction = (a: MyInterface1) => {
16    console.log(`a.a is ${a.a} and a.b is ${a.b}! and a.c is ${a.c}`);
17
18  myFunction({ a: 1, b: "Hello" }); // Error! This is MyType1
19  myFunction({ b: "Hello", c: true }); // Error! This is MyType2
20  myFunction({ a: 1, b: "Hello", c: true }); // OK
```

- "&" 연산자로 두 Type의 합집합을 만들 수 있다.



## 6.typescript (05\_union.ts)

```
npx ts-node 05_union.ts
```

```
1  export {};
2
3  function myFunction(a: number | string) {
4    if (typeof a === "number") {
5      return a + 1;
6    } else {
7      return a + "1";
8    }
9  }
10
11 console.log(myFunction(1)); // 2
12 console.log(myFunction("1")); // 11
13 console.log(myFunction(false)); // Error!
```

- “|” 연산자로 두 Type 중 하나에만 속해도 된다.



## 6.typescript (05\_union.ts)

```
npx ts-node 05_union.ts
```

```
1  export {};
2
3  function myFunction(a: number | string) {
4    if (typeof a === "number") {
5      return a + 1;
6    } else {
7      return a + "1";
8    }
9  }
10
11 console.log(myFunction(1)); // 2
12 console.log(myFunction("1")); // 11
13 console.log(myFunction(false)); // Error!
```

- “|” 연산자로 두 Type 중 하나에만 속해도 된다.



## 6.typescript (06\_any\_unknown.ts)

```
npx ts-node 06_any_unknown.ts
```

```
3  function myFunction1(a: any) {
4    return a + a;
5  }
6
7  function myFunction2(a: unknown) {
8    // 타입 검사를 하지 않으면 컴파일 에러 발생
9    if (typeof a == "number") {
10      return a + a;
11    } else if (typeof a == "string") {
12      return a + a;
13    }
14  }
15
16 const stringA: string = "1";
17 const numberA: number = 1;
18
19 console.log(myFunction1(stringA)); // 11
20 console.log(myFunction1(numberA)); // 2
21
22 console.log(myFunction2(stringA)); // 11
23 console.log(myFunction2(numberA)); // 2
```

- "any"로 타입 검사를 우회할 수 있다.
- "unknown"도 타입 검사를 우회하지만 사용전에 명시적 타입 확인을 강제한다.  
⇒ 좀 더 안전한 "any"



## 6.typescript (07\_null\_undefined.ts)

npx ts-node 07\_null\_undefined.ts

```
3 const a: any = {  
4   b: null,  
5   c: undefined, // 실제로는 undefined를 할당하지 않는 것이 좋다.  
6 };  
7  
8 console.log(null == undefined); // true  
9 console.log(null === undefined); // false  
10 console.log(a.b ? "Yes" : "No"); // No  
11 console.log(a.c ? "Yes" : "No"); // No  
12 console.log(a.b); // null  
13 console.log(a.c); // undefined  
14 console.log(a.d); // undefined
```

- 없음을 표현하는 두 가지 “null”, “undefined”
- “null”은 해당 속성의 값이 없음을 명시적으로 지칭
- “undefined”는 해당 속성이 정의되지 않은 것



0



null



undefined



NaN



## 6.typescript (08\_type\_casting.ts)

```
npx ts-node 08_type_casting.ts
```

```
3  const value: unknown = "hello";
4
5  // 방법 1 (angle-bracket syntax)
6  const str1 = <string>value;
7
8  // 방법 2 (as-syntax)
9  const str2 = value as string;
10
11 console.log(str1.toUpperCase());|
```

- 컴파일러가 해석한 타입을 따르지 않고,
- "내가 이 값의 타입을 더 잘 알고 있어!" 라고 컴파일러에게 알려주는 방식



## 6.typescript (09\_structural\_typing.ts)

npx ts-node 09\_structural\_typing.ts

console.log(obj)의 값이 뭘까요?

```
5.typescript > ts 09_structural_typing.ts > ...
1  export {};
2
3  interface MyInterface1 {
4    a: number;
5    b: string;
6  }
7
8  interface MyInterface2 {
9    a: number;
10 }
11
12 const obj1: MyInterface1 = {
13   a: 1,
14   b: "Hello",
15 };
16
17 const obj2: MyInterface2 = obj1; // MyInterface2는 b가 없다.
18
19 console.log(obj2);
20
```



# 6.typescript (09\_structural\_typing.ts)

npx ts-node 09\_structural\_typing.ts

```
tstypescript > ts 09_structural_typing.ts > ...
1  export {};
2
3  interface MyInterface1 {
4      a: number;
5      b: string;
6  }
7
8  interface MyInterface2 {
9      a: number;
10 }
11
12 const obj1: MyInterface1 = {
13     a: 1,
14     b: "Hello",
15 };
16
17 const obj2: MyInterface2 = obj1; // MyInterface2는 b가 없다.
18
19 console.log(obj2);
20
```

console.log(obj)의 값이 뭘까요?

- 객체가 특정 interface로 타입이 지정되었다고 해서, 해당 interface에 정의된 속성만 가지고 있다고 보장되지 않는다.
- 타입은 컴파일을 위한 것, 런타임 시 삭제
- 의도치 않게 민감한 정보(예: 비밀번호, 토큰 등) 가 포함된 객체가 클라이언트로 전송되는 보안 사고가 발생할 수 있다
- API 응답 객체를 타입만으로 검증하거나 신뢰할 경우, 보안상 큰 리스크로 이어질 수 있다. ⇒ **Zod를 사용하는 이유**



## 6.typescript (10\_zod.ts)

```
npm install zod  
npx ts-node 10_zod.ts
```

```
1 import { z } from "zod";  
2  
3 const UserSchema = z.object({  
4   name: z.string(),  
5   age: z.number(),  
6 });  
7  
8 interface User {  
9   name: string;  
10  age: number;  
11 }  
12  
13 const user: User = JSON.parse(  
14   '{ "name": "Alice", "age": 1, "extra": "Hello" }'  
15 );  
16 console.log("✓ user before zod:", user);  
17  
18 const parsed = UserSchema.safeParse(user);  
19 const safeUser = parsed.data;  
20 console.log("✓ user after zod:", safeUser);
```



Zod 라이브러리를 사용한다면, 런타임 때도 원하는 타입만 남게 할 수 있다.

<https://zod.dev/>

```
● → 6.typescript git:(main) ✘ npx ts-node 10_zod.ts  
✓ user before zod: { name: 'Alice', age: 1, extra: 'Hello' }  
✓ user after zod: { name: 'Alice', age: 1 }
```



## 6.typescript (10\_zod.ts)

```
npm install zod  
npx ts-node 10_zod.ts
```

```
1 import { z } from "zod";  
2  
3 const UserSchema = z.object({  
4   name: z.string(),  
5   age: z.number(),  
6 });  
7  
8 interface User {  
9   name: string;  
10  age: number;  
11}  
12  
13 const user: User = JSON.parse(  
14  '{ "name": "Alice", "age": 1, "extra": "Hello" }'  
15 );  
16 console.log("✓ user before zod:", user);  
17  
18 const parsed = UserSchema.safeParse(user);  
19 const safeUser = parsed.data;  
20 console.log("✓ user after zod:", safeUser);
```

여기까지 궁금한 점이나 이해가 안된 점

<https://zod.dev/>



Zod 라이브러리를 사용한다면, 런타임 때  
도 원하는 타입만 낼게 할 수 있다.

```
● → 6.typescript git:(main) ✘ npx ts-node 10_zod.ts  
✓ user before zod: { name: 'Alice', age: 1, extra: 'Hello' }  
✓ user after zod: { name: 'Alice', age: 1 }
```

# 6.typescript (10\_zod.ts)

```
npm install zod  
npx ts-node 10_zod.ts
```

```
1 import { z } from "zod";  
2  
3 const UserSchema = z.object({  
4   name: z.string(),  
5   age: z.number(),  
6 });  
7  
8 interface User {  
9   name: string;  
10  age: number;  
11}  
12  
13 const user: User = JSON.parse(  
14  '{ "name": "Alice", "age": 1, "extra": "Hello" }'  
15 );  
16 console.log("✓ user before zod:", user);  
17  
18 const parsed = UserSchema.safeParse(user);  
19 const safeUser = parsed.data;  
20 console.log("✓ user after zod:", safeUser);
```

추후에 다시 복습 추천

<https://zod.dev/>

```
● → 6.typescript git:(main) ✘ npx ts-node 10_zod.ts  
✓ user before zod: { name: 'Alice', age: 1, extra: 'Hello' }  
✓ user after zod: { name: 'Alice', age: 1 }
```



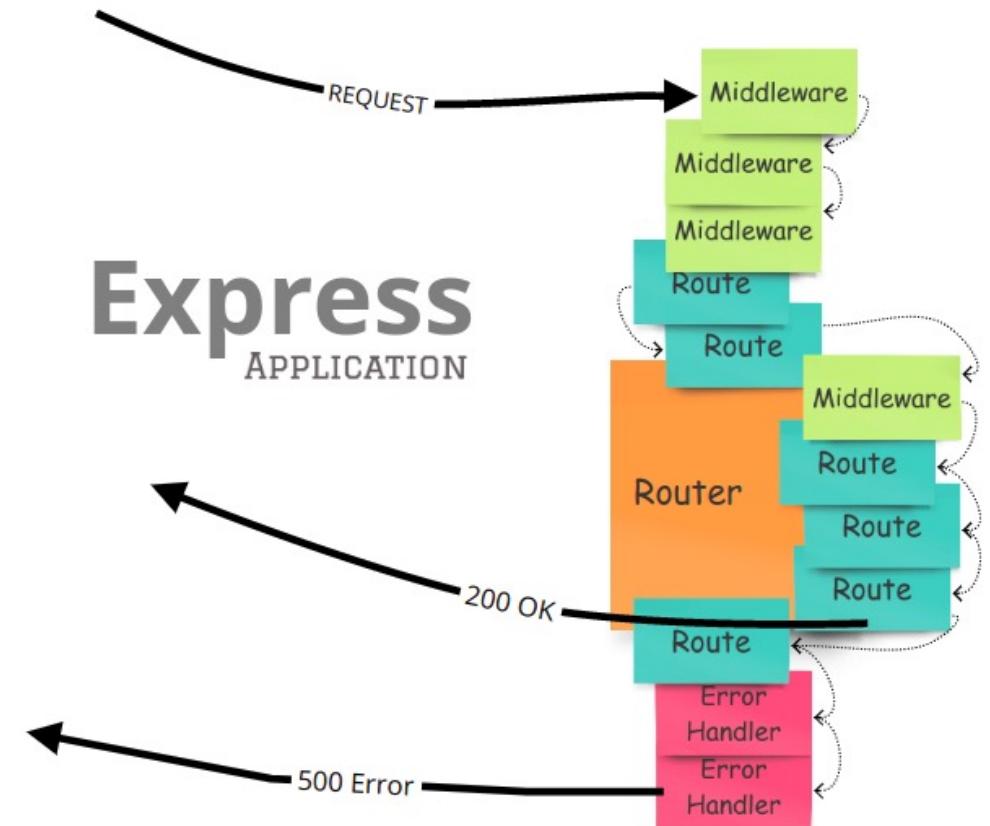
Zod 라이브러리를 사용한다면, 런타임 때  
도 원하는 타입만 남게 할 수 있다.



## 7. express\_basic



- Node.js 만으로 충분히 백엔드 서버를 구축할 수 있다.
- 하지만 Express.js 를 이용하여 좀 더 쉬운 방법으로 백엔드를 구축 가능하다.
- Node.js로 서버를 만들 때 표준 프레임워크



## 7.express\_basic

```
# 프로젝트 루트 폴더로 돌아오자  
cd 7.express_basic  
npm install express # dependencies에 추가  
npm install @types/express --save-dev # devDependencies에 추가
```

```
You, 21 hours ago | 1 author (You)  
1 {  
2   "name": "express",  
3   ▷ Debug  
4   "scripts": {  
5     "dev": "nodemon --ext ts --exec ts-node src/index.ts",  
6     "start": "node dist/index.js",  
7     "build": "tsc"  
8   },  
9   "devDependencies": {}  
10  You, 5 days ago • express  
11  "@types/node": "^22.13.10",  
12  "nodemon": "^3.1.9",  
13  "ts-node": "^10.9.2",  
14  "typescript": "^5.8.2"  
15 }  
16 }
```

{ } package.json



```
You, 4 seconds ago | 1 author (You)  
1 {  
2   "name": "express",  
3   ▷ Debug  
4   "scripts": {  
5     "dev": "nodemon --ext ts --exec ts-node src/index.ts",  
6     "start": "node dist/index.js",  
7     "build": "tsc"  
8   },  
9   "devDependencies": {}  
10  "@types/express": "^5.0.1",  
11  "@types/node": "^22.13.10",  
12  "nodemon": "^3.1.9",  
13  "ts-node": "^10.9.2",  
14  "typescript": "^5.8.2"  
15 },  
16   "dependencies": {}  
17  "express": "^4.21.2"  
18 }
```

{ } package.json

JavaScript 용으로 제  
작된 패키지는 해당하  
는 타입 패키지를 따로  
다운 받아야함.



## 7.express\_basic

```
You, 10 minutes ago | 1 author (You)          {} package.json
1  {
2    "name": "express",
3    "scripts": {
4      "dev": "nodemon --ext ts --exec ts-node src/index.ts",
5      "start": "node dist/index.js",
6      "build": "tsc"
7    },
8    "devDependencies": {
9      "@types/express": "^5.0.1",
10     "@types/node": "^22.13.10",
11     "nodemon": "^3.1.9",
12     "ts-node": "^10.9.2",
13     "typescript": "^5.8.2"
14   },
15   "dependencies": {
16     "express": "^4.21.2"
17   }
18 } You, 5 days ago • express
19
```

- npm run dev
- npm run start
- npm run build
- npm run \_\_\_\_ ← scripts 블록에서 설정 가능
- nodemon은 파일의 변화를 감지해서 실시간으로 node 명령어를 재실행하는 패키지  
⇒
- ts파일에 변화가 생길 때마다 ts-node로 "src/index.ts" 파일을 실행해라라는 뜻.

# 7.express\_basic

```
1  {
2    "compilerOptions": {
3      // JavaScript로 변환될 때 사용할 ECMAScript 버전 (ES2016은
4      // ES7)
5      // 예: async/await 등을 지원하려면 적절한 타겟이 필요
6      "target": "es2016",
7
8      // 모듈 시스템 설정 (Node.js에서는 commonjs 사용)
9      // 예: require(), module.exports를 사용하게 됨
10     "module": "commonjs",
11
12     // TypeScript 소스 코드가 위치한 루트 디렉토리
13     // 예: 소스 파일이 ./src에 있다면 컴파일 기준 경로로 사용
14     "rootDir": "./src",
15
16     // 컴파일된 JavaScript 파일이 출력될 디렉토리
17     // 예: ./dist 폴더에 결과물이 생성됨
18     "outDir": "./dist",
19
20     // CommonJS 방식 모듈(import/export)과 ES 모듈 간의 호환성 보장
21     // 예: `import fs from 'fs'`처럼 ES 스타일 import가 가능하게
22     // 함
23     "esModuleInterop": true,
24
25     // TypeScript의 모든 엄격한 타입 검사 기능을 활성화
26     // 예: null 체크, 암시적 any 방지 등 엄격한 코드 작성 유도
27     "strict": true,
28   }
}

```

- express에 맞추어 tsconfig.json 수정되어있음
- rootDir에 지정된 ts파일들을 컴파일해서 outDir에 지정된 폴더에 js파일들을 저장함
- es2016 : 2016년도 버전 javascript
- 특정 문법이 사용이 안될 때는 자바스크립트의 버전을 확인해보자



## 7.express\_basic (실행법)

#1  파일 변화 시 즉시 재실행이라 변화를 바로 볼 수 있어 편하다

`npm run dev`

#2

`nodemon --ext ts --exec ts-node src/index.ts`

#3

`npm run build && npm run start`

#4

`tsc`

`node dist/index.js`



# 7.express\_basic (실행법)

```
[nodemon] 3.1.9
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: ts
[nodemon] starting `ts-node src/index.ts`
Server is running on http://localhost:4000
```

Hello, TypeScript Express!

- app.listen: 서버 시작
- app.get: GET 요청받아서 응답하기
- app.post: POST 요청받아서 응답하기
- Express에서 app과 router의 path 매칭은 코드가 작성된 "위에서부터 아래로", 즉 등록 순서에 따라 처리
- localhost:4000/cat/12?size=123 이런 요청은 26번째 줄이 받는다.

```
You, 22 hours ago | 1 author (You)
1 import express, { Request, Response } from "express";
2
3 const app = express();
4 const PORT = 4000;
5
6 app.get("/", (req: Request, res: Response) => {
7   res.send("Hello, TypeScript Express!");
8 });
9
10 app.get("/cat", (req: Request, res: Response) => {
11   res.send("GET, It's a cute cat!");
12 });
13
14 app.post("/cat", (req: Request, res: Response) => {
15   res.send("POST, It's a cute cat!");
16 });
17
18 app.put("/cat", (req: Request, res: Response) => {
19   res.send("PUT, It's a cute cat!");
20 });
21
22 app.delete("/cat", (req: Request, res: Response) => {
23   res.send("DELETE, It's a cute cat!");
24 });
25
26 app.get("/cat/:id", (req: Request, res: Response) => [
27   const { id } = req.params;
28   const { size } = req.query;
29   res.send(`GET, It's a cute cat! ID: ${id}, Size: ${size}`);
30 ]);
31
32 app.listen(PORT, () => {
33   console.log(`Server is running on http://localhost:${PORT}`);
34 });
```



# 8. practice

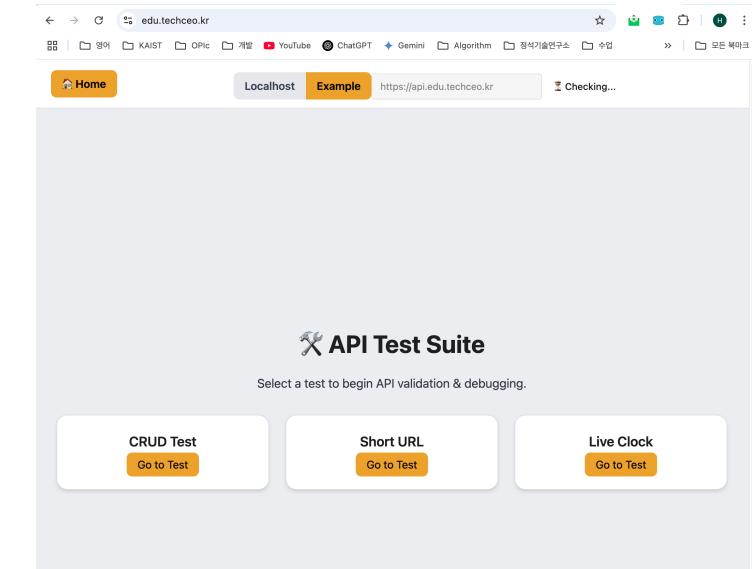
## 로컬 프론트엔드 서버 실행법

```
# 프로젝트 루트 폴더로 돌아오자  
cd 8. practice  
cd frontend  
npm install # 처음 한 번만  
npm run dev # "control + c" 를 하면 중  
단된다. 다시 실행하고 싶으면 frontend 폴  
더에서 이 명령어 실행
```

```
npm install # 처음 한 번만  
npm run dev # 다시 실행하고 싶으면  
backend 폴더에서 이 명령어 실행
```

프론트엔드를 로컬에서 실행하지 않고,  
이미 배포되어 있는 프론트엔드를 사용해도된다.

<https://edu.techceo.kr>



## 로컬 백엔드 서버 실행법

```
# 새 터미널을 열고 프로젝트 루트 폴더로  
cd 8. practice  
cd shared  
npm install # 처음 한 번만  
cd ../backend
```

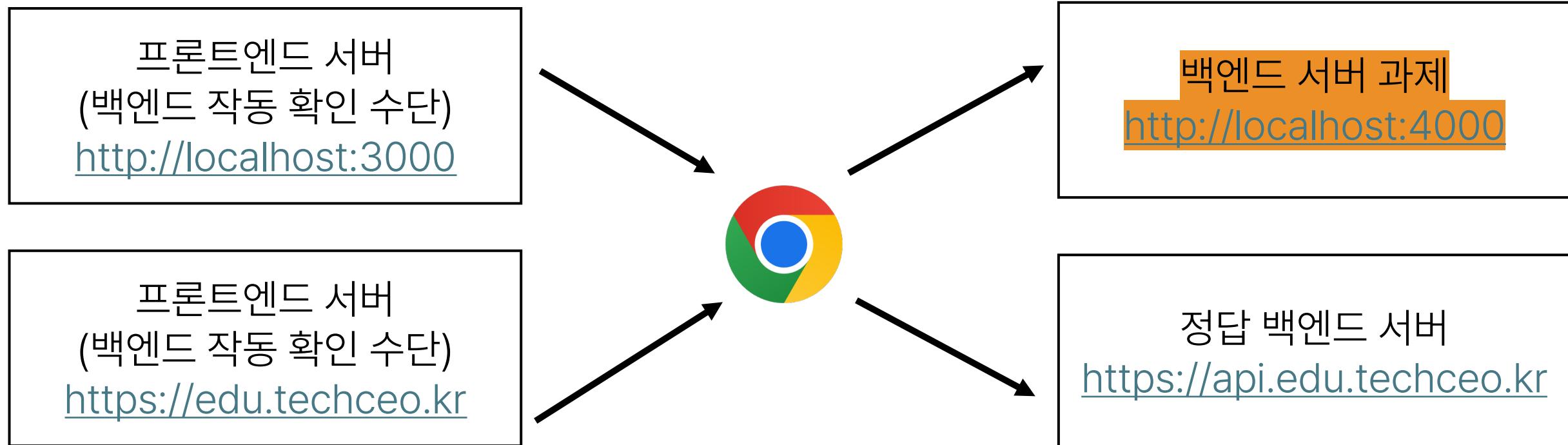


## 8. practice 과제 설명

과제를 위해 프론트엔드와 정답 백엔드 제작했음

과제

- Short URL : 링크 줄여주는 Bitly 서비스
- Live Clock : 네이비즘



## 8. practice 과제 설명

과제

- Short
- Live C

localhost

Example

http://localhost:4000

Back-end Server is running

(백엔드)  
localhost

상단의 Localhost 버튼을 눌러서 본인이 실행하는  
백엔드 서버를 테스트 할 수 있게 하자.

Example을 누르면 정상 작동하는 정답 백엔드 서버  
랑 연결 가능.

(백엔드)  
edt

☰ 과제  
st:4000

☰ 정답  
echceo.kr



# 8. 미들웨어1

```
14 //CORS 설정
15 const whitelist = [`http://localhost:3000`, "https://edu.techceo.kr"];
16 const corsOptions: CorsOptions = {
17   origin: function (origin, callback) {
18     if (!origin || whitelist.indexOf(origin) !== -1) {
19       callback(null, true);
20     } else {
21       callback(new Error("Not allowed by CORS"));
22     }
23   },
24 };
25 app.use(cors(corsOptions)); // CORS 미들웨어 추가(모든 경로에 대해 전역 적용)
26 app.use(express.json()); // JSON 파싱(모든 경로에 대해 전역 적용)
27
```

특정 경로에 대해서만 미들웨어를 적용하거나

```
51 app.use("/auth", authMiddleWare, catRouter);
```

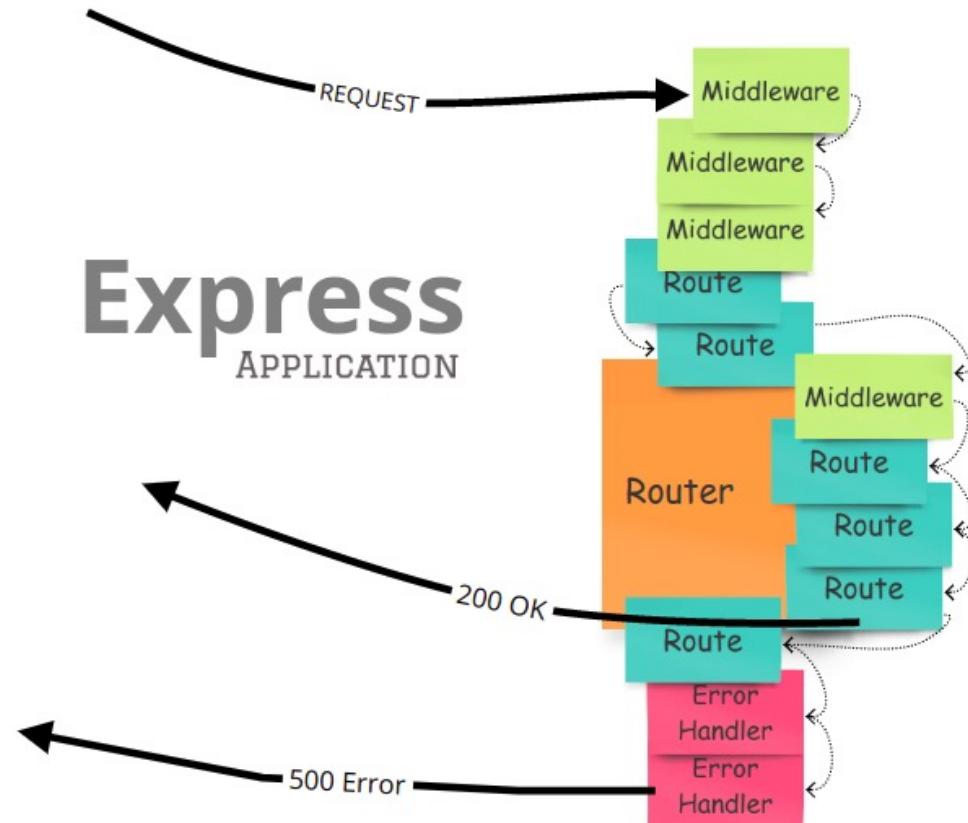
- 미들웨어(Middleware)는 요청(Request)과 응답(Response) 사이에서 실행되는 함수
- 클라이언트가 서버에 요청을 보냈을 때, 그 요청을 처리하기 전에 중간에 어떤 작업을 할 수 있게 도와줌
- cors() : 응답 전에 cors 문제 해결
- express.json() 응답 전에 요청 바디를 json형태로 파싱

응답 바로 직전에도 미들웨어 사용 가능

```
28 app.get("/cat", authMiddleWare, (req: Request, res: Response) => {
29   res.send("GET, It's a cute cat!");
30 })
```



## 8. 미들웨어가 유용한 이유



ex) 모든 요청에 대해 사용자 인증을 하고 응답을 해야한다면

app.use(middleware)를 해서 모든 응답 전에 사용자를 인증하는 미들웨어를 거치면 편하다.

ex) 만약 미들 웨어를 안쓴다면

모든 응답마다 사용자 인증하는 함수를 호출해야 한다.



# 8. 라우터

TS index.ts

```
app.get("/cat", authMiddleWare, (req: Request, res: Response) => {
  res.send("GET, It's a cute cat!");
});

app.post("/cat", (req: Request, res: Response) => {
  res.send("POST, It's a cute cat!");
});

app.put("/cat", (req: Request, res: Response) => {
  res.send("PUT, It's a cute cat!");
});

app.delete("/cat", (req: Request, res: Response) => {
  res.send("DELETE, It's a cute cat!");
});

app.get("/cat/:id", (req: Request, res: Response) => {
  const { id } = req.params;
  const { size } = req.query;
  res.send(`GET, It's a cute cat! ID: ${id}, Size: ${size}`);
});
```

- 다른 응답 처리들이 많아서 헷갈리는데
- 공통된 "/cat" path를 뺀어서 한 파일에서 볼 순 없을까?



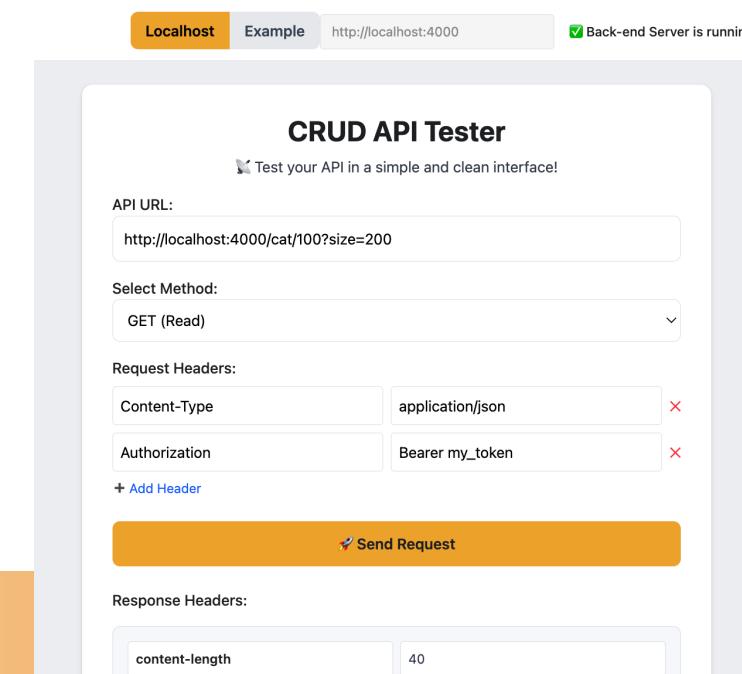
# 8. 라우터

50 | app.use("/cat", **catRouter**);

```
You, 5 hours ago | 1 author (You)
1 import express, { Request, Response } from "express";
2 import { allowCorsHeaders } from "../middlewares/cors";
3 const catRouter = express.Router();      You, 5 hours ago
4 
5 catRouter.use(allowCorsHeaders);
6
7 catRouter.get("/", (req: Request, res: Response) => {
8   res.send("GET, It's a cute cat!");
9 });
10
11 catRouter.post("/", (req: Request, res: Response) => {
12   res.send("POST, It's a cute cat!");
13 });
14
15 catRouter.put("/", (req: Request, res: Response) => {
16   res.send("PUT, It's a cute cat!");
17 });
18
19 catRouter.delete("/", (req: Request, res: Response) => {
20   res.send("DELETE, It's a cute cat!");
21 });
22
23 catRouter.get("/:id", (req: Request, res: Response) => {
24   const { id } = req.params;
25   const { size } = req.query;
26   res.send(`GET, It's a cute cat! ID: ${id}, Size: ${size}`);
27 });
28
29 export default catRouter;
```

routes  
TS cat.ts

- express.Router()을 이용하면 공통된 path를 묶어서 한 파일에 정리 가능!
- 로컬 백엔드 서버(localhost:4000)을 실행한 상태에서 프론트엔드에서 직접 요청을 보내 확인해 보자!
- 다른 post put delete 요청도 보내보자.
- ex) PUT, http://localhost:4000/cat



# 8. 미들웨어2

51 app.use("/auth", authMiddleWare, catRouter);

```
You, 5 hours ago | author (you)
1 import { Request, Response, NextFunction } from "express";
2 const authMiddleWare = (req: Request, res: Response, next: NextFunction) => {
3   const { authorization } = req.headers;
4   if (authorization && authorization === "Bearer 1234") {
5     next();
6   } else {
7     res.status(401).send("Unauthorized");
8   }
9 };
10 export { authMiddleWare };
```

- "/auth"로 시작하는 path는 응답 전에 authMiddleWare를 거치는데
- 요청 헤더의 Authorization 값이 "Bearer 1234"여야지만 통과가 된다.
- ex) GET, http://localhost:4000/auth/

The screenshot shows a REST client interface with two requests side-by-side.

**Request 1 (Left):**

- API URL: `http://localhost:4000/auth`
- Select Method: `GET (Read)`
- Request Headers:
  - Content-Type: `application/json`
  - Authorization: `Bearer my_token`
- Buttons: `+ Add Header`, `Send Request`

**Request 2 (Right):**

- API URL: `http://localhost:4000/auth`
- Select Method: `GET (Read)`
- Request Headers:
  - Content-Type: `application/json`
  - Authorization: `Bearer 1234`
- Buttons: `+ Add Header`, `Send Request`

**Responses:**

**Request 1 Response:**

- Response Headers:
  - content-length: `21`
  - content-type: `text/html; charset=utf-8`
  - date: `Sun, 23 Mar 2025 13:41:51 GMT`
  - etag: `W/"15-Bo7f9Pb1NX36drC9Sc+Iqm1PFM`
  - x-powered-by: `Express`
- Response Data:

`✗ Request failed! [401 Unauthorized]`

**Request 2 Response:**

- Response Headers:
  - content-length: `12`
  - content-type: `text/html; charset=utf-8`
- Response Data:

`HTML Preview:`  
`GET, it's a cute cat!`



# 8. 정적 파일 서빙

```
.. app.use("/static", express.static(path.join(__dirname, "..", "public")));
```

```
✓ public
  < 404.html
  [x] cute_cat.jpg
  [x] cute_cat.png
```

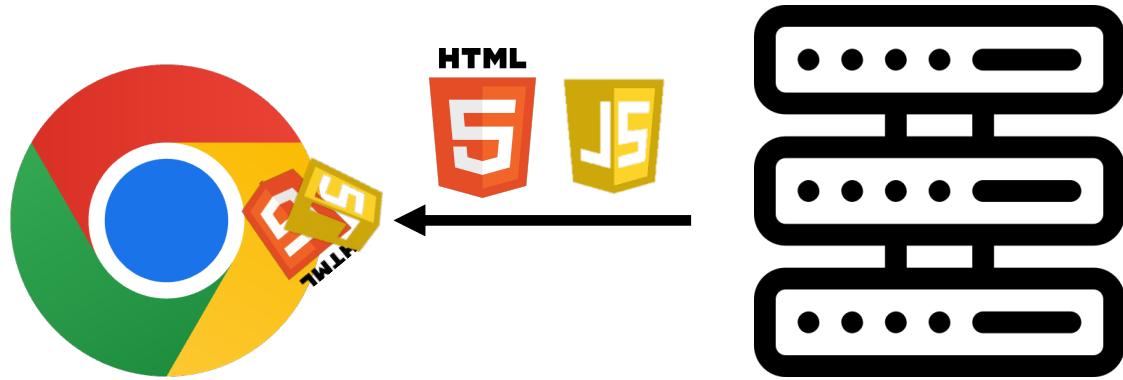
The screenshot shows a network request for a file named 'cute\_cat.png' from the URL 'http://localhost:4000/static/cute\_cat.png'. The request method is set to 'GET (Read)'. In the 'Request Headers' section, there are two entries: 'Content-Type: application/json' and 'Authorization: Bearer my\_token'. Below the request section, the 'Response Headers' are listed as follows:

cache-control	public, max-age=0
content-length	183572
content-type	image/png
last-modified	Sun, 23 Mar 2025 08:15:28 GMT

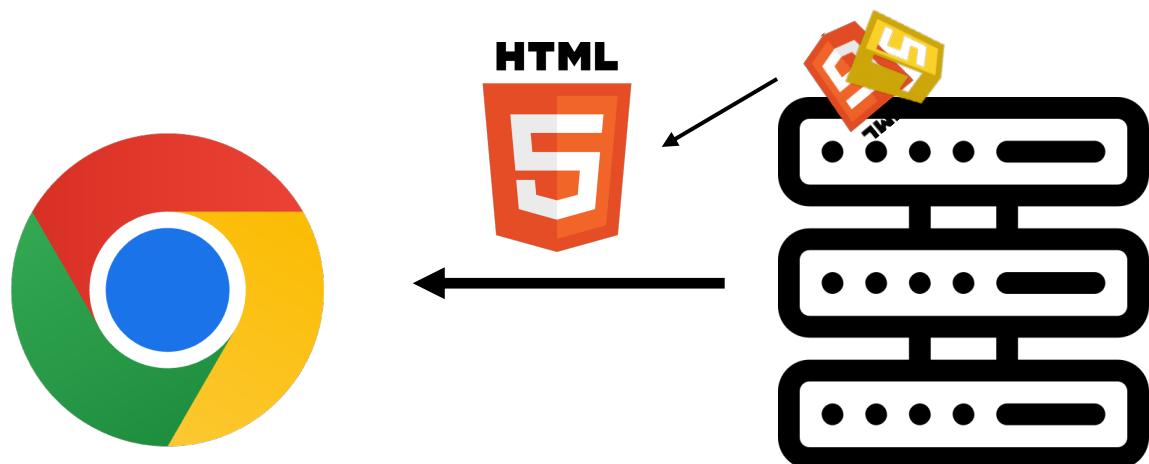
Under the 'Response Data' section, there is a heading 'Image Preview:' followed by a thumbnail image of a fluffy, light-colored kitten peeking out from a white, textured hood or hat.

- 요청이 들어오면 요청을 해석해서 응답하는 것이 아니라(동적)
- 요청 그대로 서버에 있는 파일로 응답하는 것. (정적)
- HTML, CSS, Image 등등
- ex) GET,  
http://localhost:4000/static/cute\_cat.png

# 8. SSR(Server Side Rendering)



CSR: 서버의 렌더링 부하가 없음



SSR: 검색엔진(SEO)에 최적화  
( google, Bing 같은 최신 검색엔진은 CSR도 지원)

CSR (Client-Side Rendering)

- React는 브라우저에 기본적인 HTML만 제공
- 이후 추가로 제공된 JavaScript가 실행되어 화면을 동적으로 구성
- 초기에 화면이 비어 있거나 로딩 상태일 수 있음

SSR (Server-Side Rendering)

- JavaScript를 서버에서 실행해 HTML을 완성한 뒤, 브라우저에 전달
- 브라우저는 JS 실행 없이도 완성된 화면을 바로 볼 수 있음

ex)

<http://localhost:4000/csr>

<http://localhost:4000/ssr>

결국 화면 렌더링을 브라우저에서 하냐,  
HTML을 제공하는 서버에서 하냐의 차이

# The End

## 🛠 과제 구성

- Short URL: 링크 줄여주는 Bitly 스타일 서비스 → ✓ live-clock.ts 완성
- Live Clock: 서버 실시간 시계 → ✓ short-url.ts 완성 (네이버 즘 사이트에서 시간 비교: nexon.com, google.com)

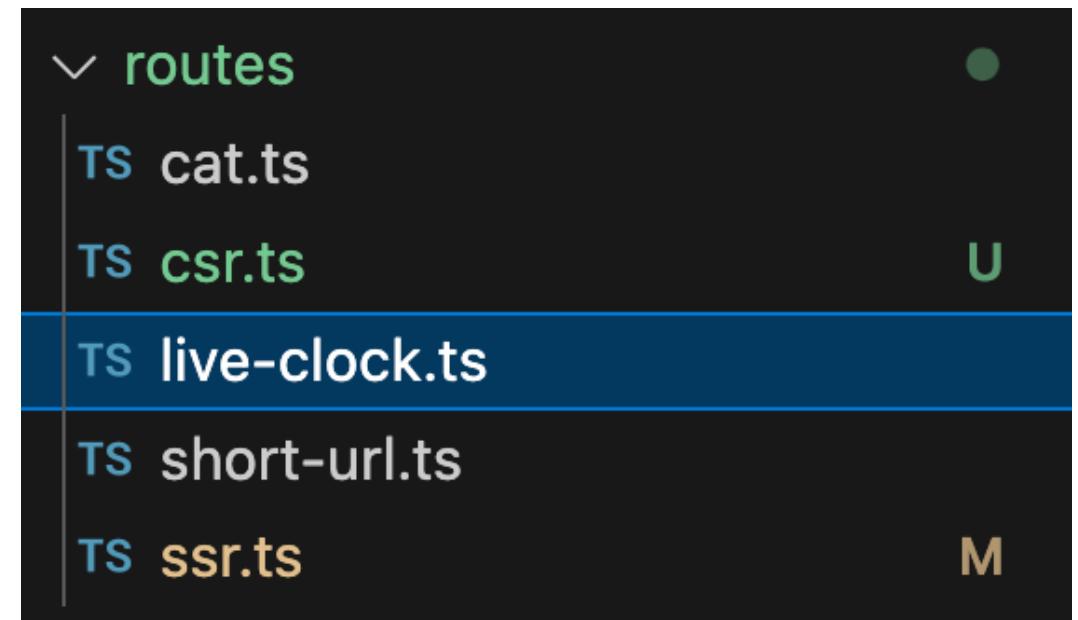
## 📌 제출 방법

- 완성한 과제는 base repository의 main 브랜치로 Pull Request(PR) 올리기
- 과제 완료 기준:  
▶ PR 작성 → 코드 리뷰 받기 → 리뷰 반영까지 완료

## ⏰ 일정

- PR 제출 기한: ~ 4월 2일
- 리뷰 반영 기한: ~ 4월 7일

빠르게 시도해보시고, 도움을 빨리 받기를 권장합니다.



참조할 수 있는 자료를 제공해 주신  
Static, Night 님 감사합니다.