Hello,

KDT 웹 개발자 양성 프로젝트

5기!

with

웹프로그래밍
팀 프로젝트

Forge the Project

최종발표

17조 박성용 김윤기 문윤기


수고하셨습니다!


곧, FE가
끝난다
FE가 끝나면
어떻게 되는데?

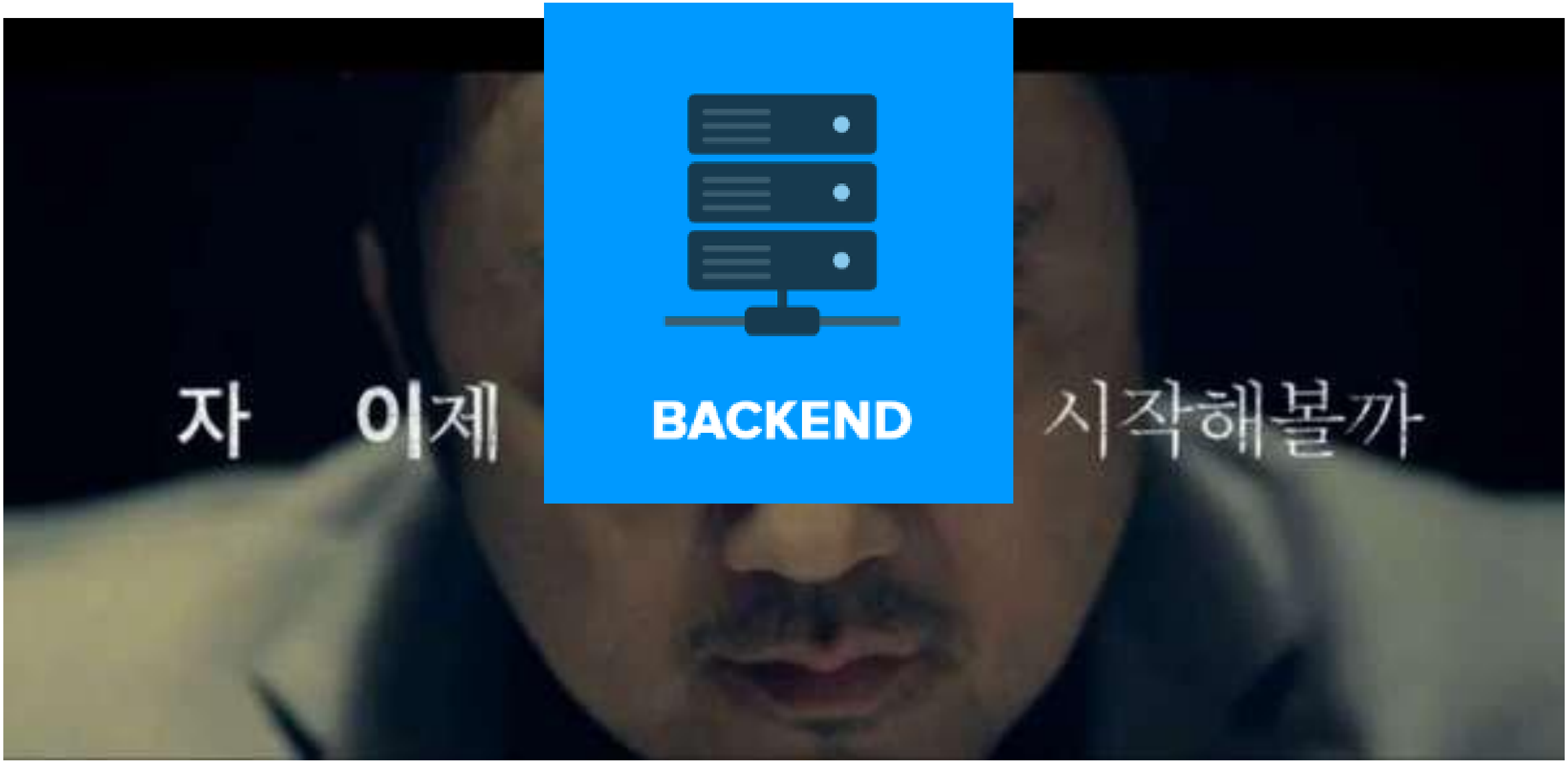

모르는가?


곧, FE가
끝난다
FE가 끝나면
어떻게 되는데?


모르는가?


Back-End가
시작되지


FE가 끝나면
어떻게 되는데?
곧, FE가
끝난다


자 이제
BACKEND
시작해볼까


그래~!
답이없네!
답이없어!
끝이없네!!
끝이없어!!
끝이없네!!!
답이없다!!!


지금 시작합니다

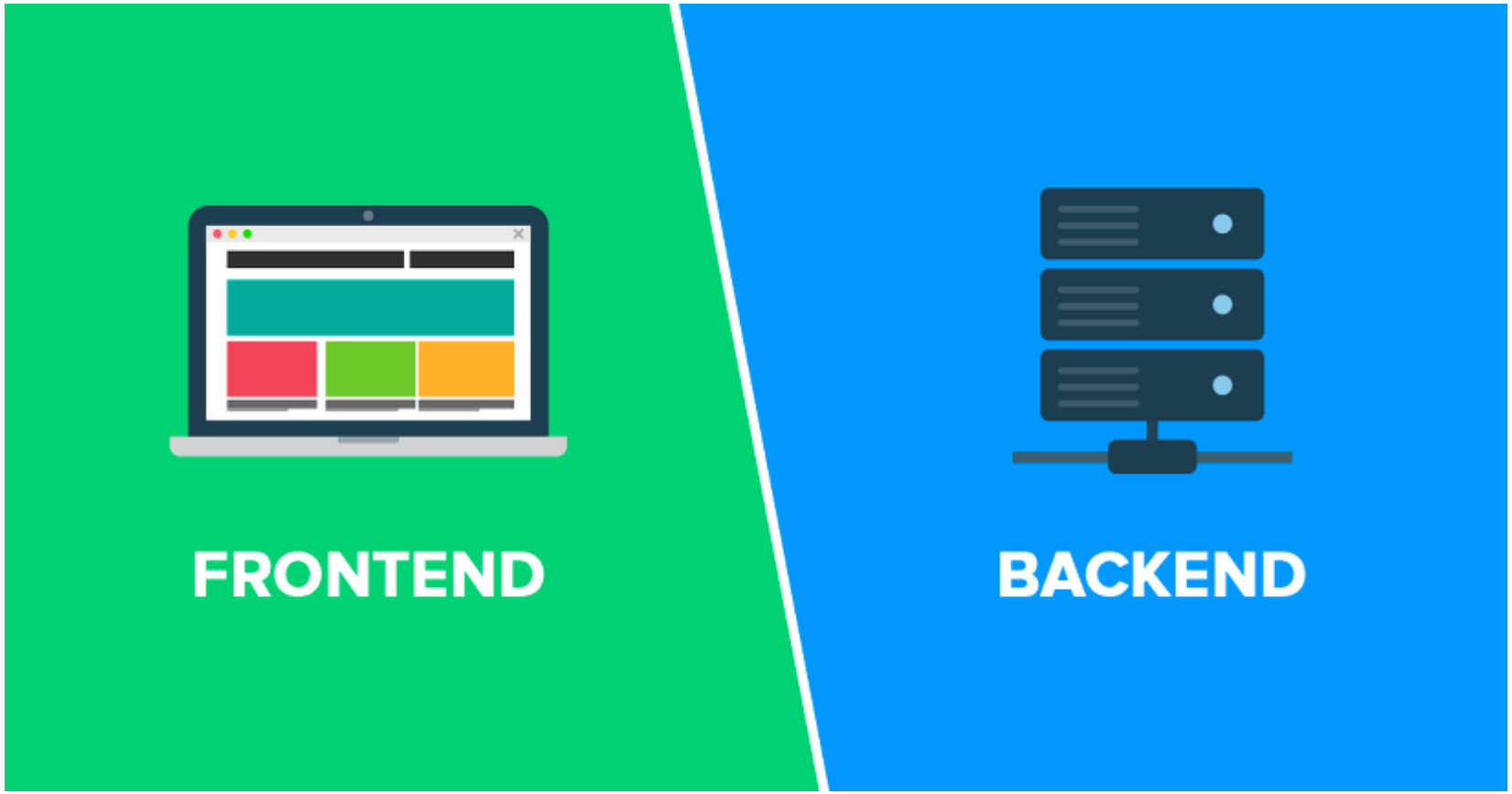

EBS 1
잠
깐

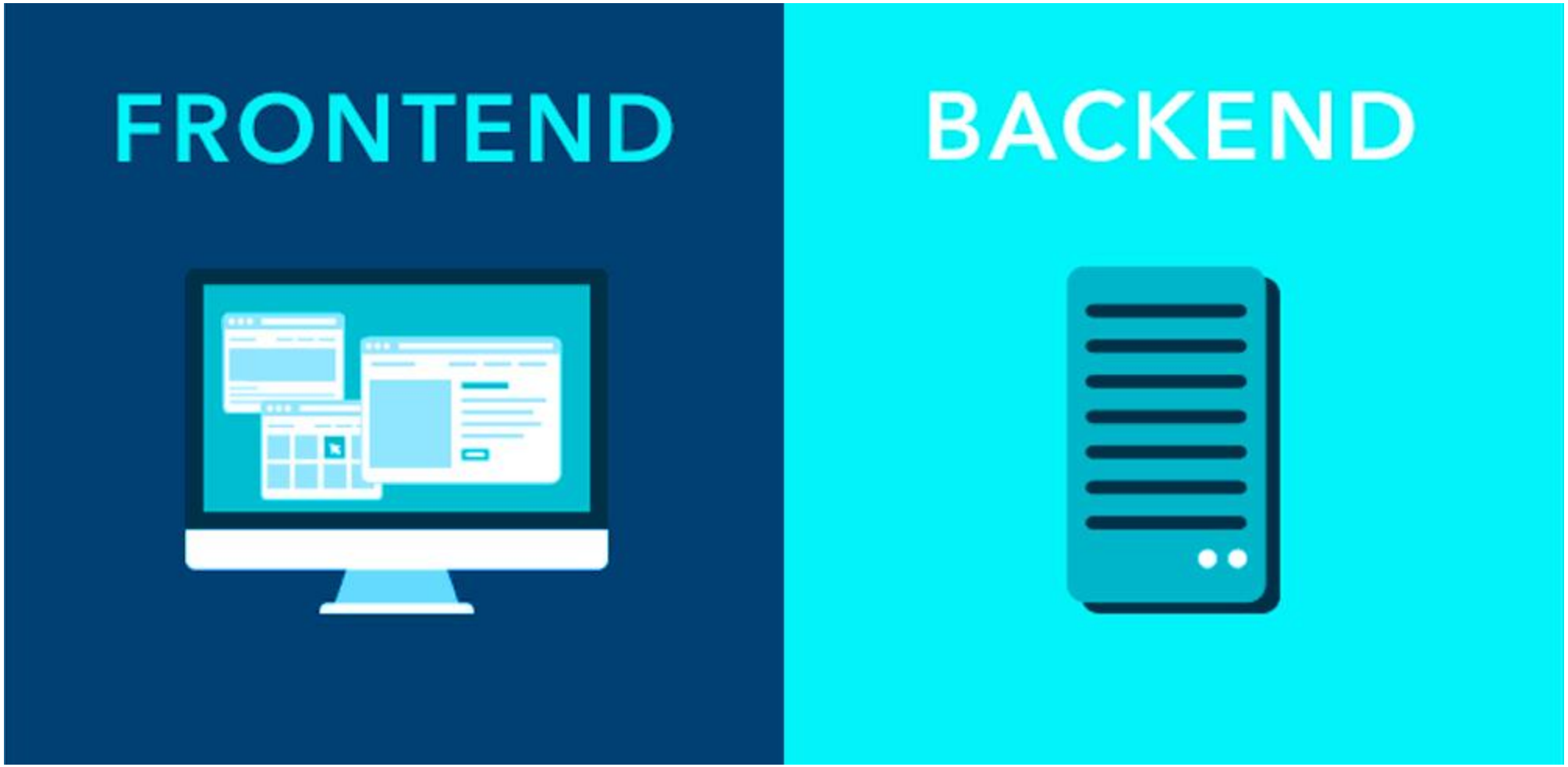

꿈은
없고요
그냥
놀고 싶습니다

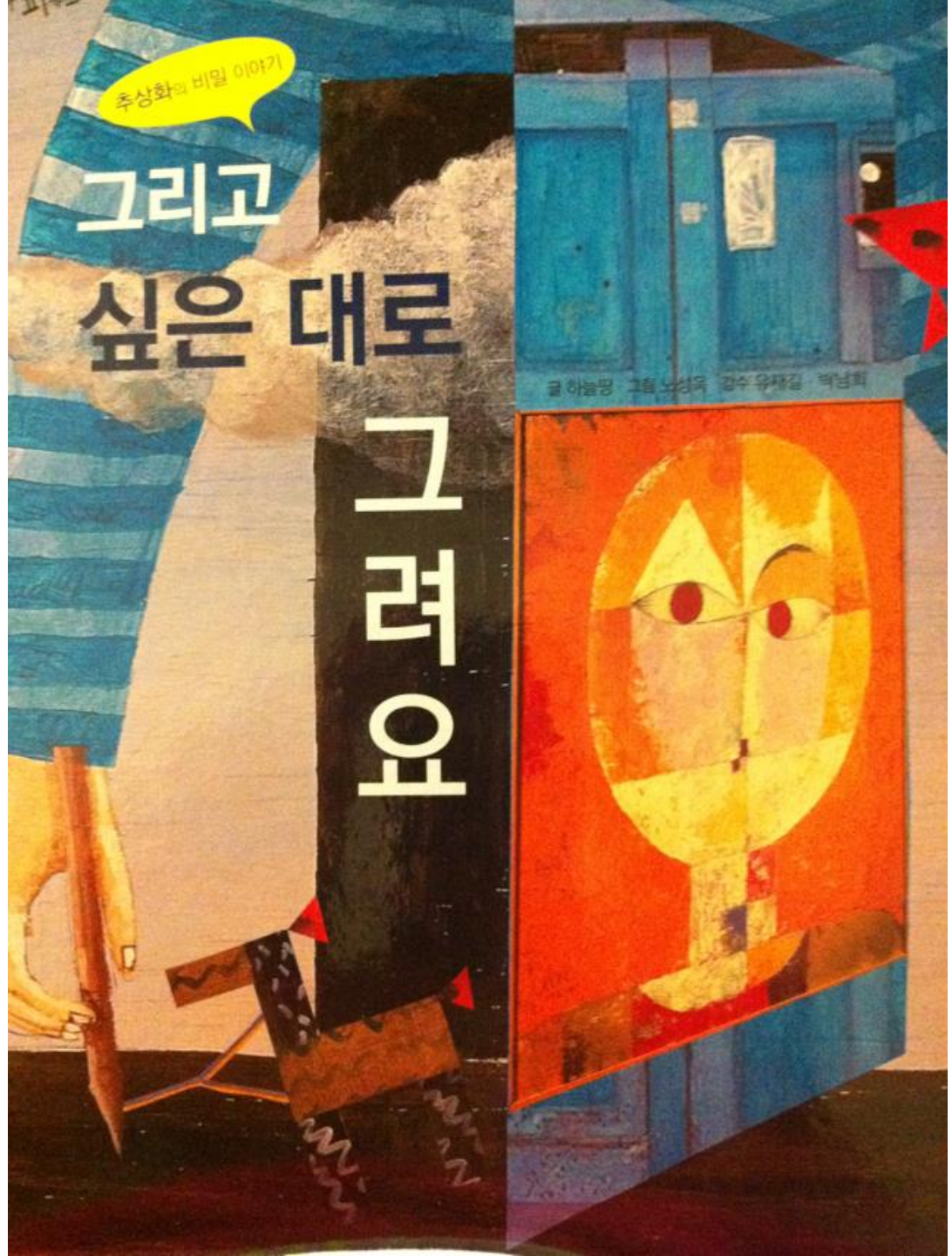

TV CHOSUN
-여러분 인생은 오르막길의 연속이죠


가 즈 아
https://coinpan.com 코인판 on 2018-01-09

Back-End


FRONTEND
BACKEND


FRONTEND
BACKEND


추상화의 비밀 이야기
그리고
싶은 대로
그려요

첼시, 바르셀로나 '초대형' 선수 영입 임박...최종 합의 완료

[포포투=이규학]프렌키 더 용을 품는 클럽은 첼시로 보인다. 더 용의 협상이 최종 합의에 이르렀다.바르셀로나 소식에 정통한 제라르 로메로 기자는 11일(한국..."

포포투 · 라리가

8년 전 방출한 FW에게 혼쭐난 맨유..."이럴 거면 다시 데려와! "

[사진] ⓒGettyimages(무단전재 및 재배포 금지)[OSEN=고성환 기자] 맨체스터 유나이티드의 전설 리오 퍼디난드(44)가 대니 웰벡(32, 브라이..."

OSEN · 프리미어리그

'아무리 생각해도 이유를 모르겠다' KIA는 왜 피렐라와 승부했나

2022 KBO리그 삼성라이온즈와 KIA타이거즈의 경기가 10일 대구삼성라이온즈파크에서 열렸다. 삼성 피렐라가 10회말 끝내기 적시타를 치고 환호하고 있다...."

스포츠조선 · KBO리그

키움, SSG 대항마였는데...한 달 만에 격차는 2. 5경기→10경기

4일 서울 구로구 고척스카이돔에서 열린 프로야구 '2022 신한은행 SOL KBO리그' SSG 랜더스와 키움 히어로즈의 경기, 키움 3루수 송성문이 10회초..."

뉴스1 · KBO리그

③ 3명 늘어난 엔트리, 벤투호에 승선할 태극전사 26명은?

FIFA 월드컵 최종 엔트리 23명→26명으로 확대손흥민·황의조·김민재 등 '붙박이'에 엄원상·정우영 합류 가능성↑손흥민의 인사(서울=연합뉴스) 이지은 기자..."

연합뉴스 · 대표팀

스포츠 창작자 영상

1 / 4

[영국방송] "손흥민 놔두면 당하지!" BBC MOTD 토...

조회 31,043 · 한준TV

역시 국대 출신! 김지영2 프로의 드라이버샷과 특별...

조회 3,047 · 스포츠W - Spor...

손흥민급 감아차기ㄷㄷ 추가시간에 터진 역대급 원...

조회 2,121 · 박축공 Football ...

뉴욕 메츠를 이끄는 공포의 슬라이더 트리오

조회 3,398 · 김형준의 야구야...

"어썸 킴!" 외친 홈팬들 앞 2루타 김하성, 판독 끝 날...

조회 16,236 · 이현우의 MLBTV

"나 잡으려면 감독이랑 단장 짤라"

조회 10,807 · 농구대학


iksuil.kr

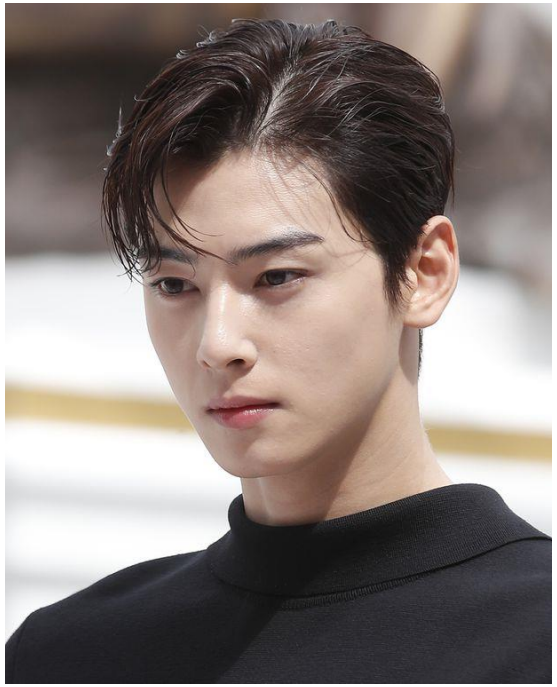
프론트-엔드

백-엔드


요청
클라이언트
서버
응답

남성 연예인

남성 연예인



Back-End


JSP


node
JS


python
TM

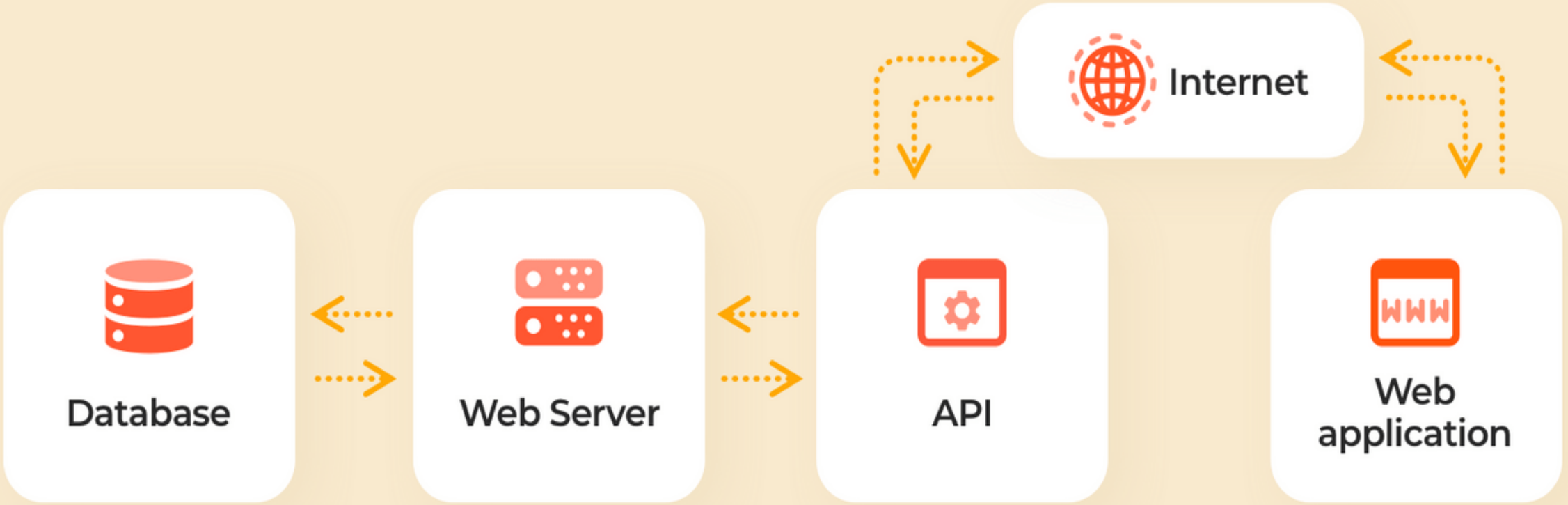
남성 연예인 목록 API

A Application

P Programming

I Interface

What is an API?



API(Application Programming Interface)

- 서로 다른 두 어플리케이션이 통신을 할 수 있도록 해주는 일종의 규약
- 예를 들어 백엔드 서버는 Java 로 되어 있고, 프론트엔드 페이지는 JS 로 되어 있다면!?
- 마치 영어만 가능한 사람과 카자흐스탄어만 가능한 사람이 대화 하는 것!
- 이때 API는 서로가 서로의 말로 통역을 해주는 것
- 영어만 가능한 사람이 카자흐말로 번역해서 데이터를 전달하는 것이죠!

API 주소

http://api.tetzkdt.com/male-stars

https://apis.data.go.kr/B551011/KorService/areaBasedList?serviceKey=rfaoGpiapHFqOcUT6bqfERRxy1WVxzOdOpEC3ChyAFPEfONdSMdRVNETTJKRhqTbPuZ2krpG2mQJMXDbyG74RA%3D%3D&numOfRows=687&pageNo=1&MobileOS=ETC&MobileApp=TripLog&_type=json&listYN=Y&arrange=B&contentTypeId=12&areaCode=1

http://api.tetzkdt.com/male-stars

```
[
  {
    name: "차은우",
    image: "http://file.tetzkdt.com/malestars/차은우.png"
  },
  {
    name: "고수",
    image: "http://file.tetzkdt.com/malestars/고수.png"
  },
  {
    name: "이동욱",
    image: "http://file.tetzkdt.com/malestars/이동욱.png"
  }
]
```

nodejs-express-middleware > JS server.js > ...

```
1   const express = require('express');
2   const app = express();
3
4   // Express is just a series of middleware calls, hmmm...
5   // Here, only middleware is registered
6   // nothing like app.get(/* ... params */) or app.post(/* ... params */)
7   // but when we open http://localhost:3000 in browser, then look at the console
8   // the output is:
9   // First middleware GET
10  // Second middleware GET
11  app.use(
12    (req, res, next) => {
13      console.log('First middleware', req.method);
14      return next();
15    },
16    (req, res, next) => {
17      console.log('Second middleware', req.method);
18      return next();
19    },
20    (req, res, next) => {
21      res.send(`Hello from ${req.method}
22      request to ${req.url}`);
23    }
24  );
25
26  app.listen(3000, () => console.log('Server started on: http://localhost:3000'));
27
```

{ REST : API }

Representational State Transfer API

https://apis.data.go.kr/B551011/KorService/areaBasedList?serviceKey=rfaoGpiapHFqOcUT6bqfERRxy1WVxzOdOpEC3ChyAFPEfONdSMdRVNETTJKRhqTbPuZ2krpG2mQJMXDbyG74RA%3D%3D&numOfRows=687&pageNo=1&MobileOS=ETC&MobileApp=TripLog&_type=json&listYN=Y&arrange=B&contentTypeId=12&areaCode=1

Front-End

http://api.tetzkdt.com/male-stars

```
[
  {
    name: "차은우",
    image: "http://file.tetzkdt.com/malestars/차은우.png"
  },
  {
    name: "고수",
    image: "http://file.tetzkdt.com/malestars/고수.png"
  },
  {
    name: "이동욱",
    image: "http://file.tetzkdt.com/malestars/이동욱.png"
  }
]
```

대표적 방법

두 가지!

프론트에서 백엔드에게 정보를 요청

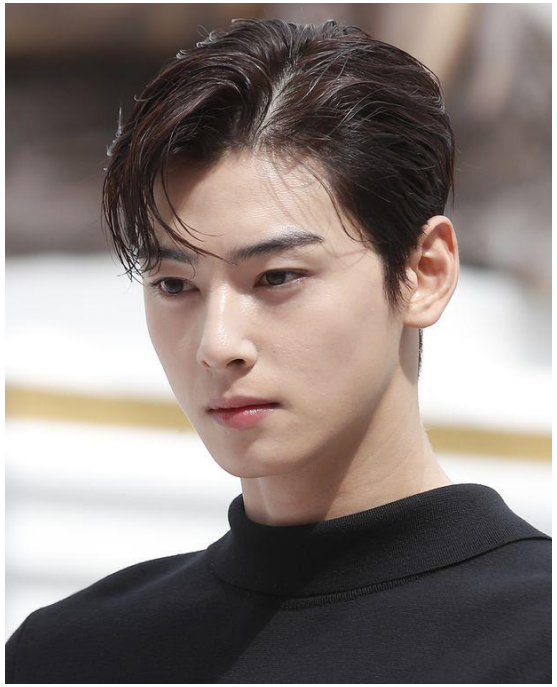
```
async function fetchData() {
      const tokenResponse = await fetch(
        `https://kauth.kakao.com/oauth/token?grant_type`,
        {
          method: 'POST',
          headers: {
            'Content-type': 'application/x-www-form-urlencoded;charset=utf-8',
          },
        }
      );

      if (tokenResponse.status === 200) {
        const tokenData = await tokenResponse.json();
        const userResponese = await fetch(`https://kapi.kakao.com/v2/user/me`, {
          method: 'POST',
          headers: {
            Authorization: `Bearer ${tokenData.access_token}`,
            'Content-type': 'application/x-www-form-urlencoded;charset=utf-8',
          },
        });
```

백엔드에서 데이터를 보내서 그려주기(ejs)

```
<body>
    <section class="male-star">
        <% for(let i=0; i < maleStar.length; i++) { %>
            <p>
                <%=maleStar[i].name%>
            </p>
            <img src="<%=maleStar[i].image%>" />
        <% } %>
    </section>
</body>
```

남성 연예인




HONEYCAM

여성 연예인

nodejs-express-middleware > JS server.js > ...

```
1   const express = require('express');
2   const app = express();
3
4   // Express is just a series of middleware calls, hmmm...
5   // Here, only middleware is registered
6   // nothing like app.get(/* ... params */) or app.post(/* ... params */)
7   // but when we open http://localhost:3000 in browser, then look at the console
8   // the output is:
9   // First middleware GET
10  // Second middleware GET
11  app.use(
12    (req, res, next) => {
13      console.log('First middleware', req.method);
14      return next();
15    },
16    (req, res, next) => {
17      console.log('Second middleware', req.method);
18      return next();
19    },
20    (req, res, next) => {
21      res.send(`Hello from ${req.method}
22      request to ${req.url}`);
23    }
24  );
25
26  app.listen(3000, () => console.log('Server started on: http://localhost:3000'));
27
```

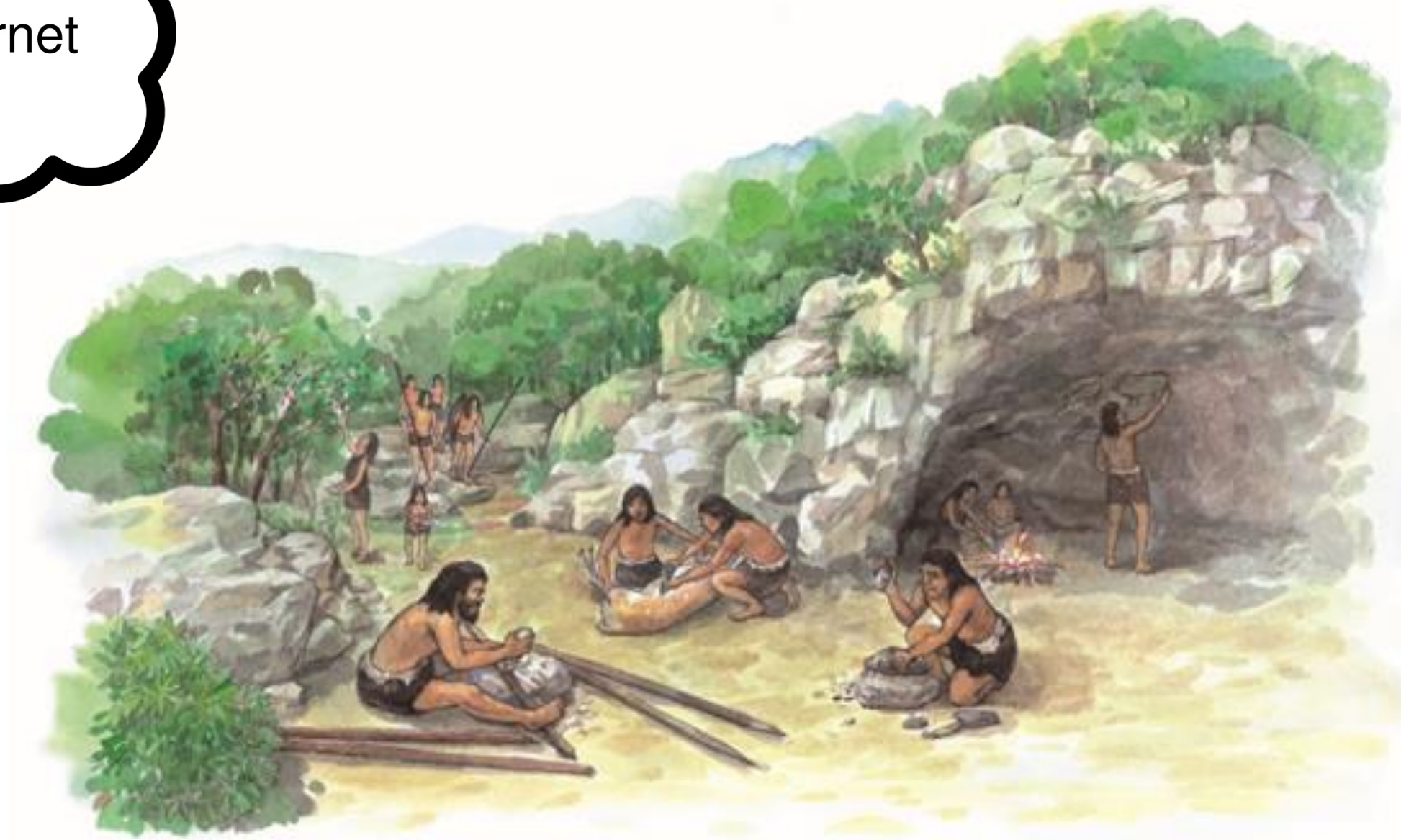
http://api.tetzkdt.com/female-stars

```
[
  {
    name: “아이린”,
    image: http://file.tetzkdt.com/malestars/아이린.png
  },
  {
    name: “카리나”,
    image: “http://file.tetzkdt.com/malestars/카리나.png”
  },
  {
    name: “김태희”,
    image: “http://file.tetzkdt.com/malestars/김태희.png”
  }
]
```


node
JS
자 이제 시작이야 Back-End

Node-JS

Internet



신문검색 | 국외검색 NAVER 서치센터 | 광고안내

맞춤정보 서비스, MyNAVER
매일매일 핫&쿨, WebToday

삼성생명 오픈 이벤트

네이버를 나의 홈페이지로...
네오넷의 부동산 급매물 서비스!

NAVER 검색

분류에서 검색

제목에서 검색 Quick Help 확장검색

NAVER 소식

네이버 포탈서비스 오픈! MyNAVER
네이버 웹문서수 3,141,838 건

NAVER 분류

- 건강/의학
  건강관리, 병원, 의학
- 교육
  대학, 시험,자격증, 유학
- 뉴스/미디어
  신문, 잡지, 텔레비젼
- 레크리에이션
  스포츠, 게임, 여행, 레저
- 사업/경제
  기업, 취업, 온라인쇼핑
- 사회과학
  경제학, 사회학, 언어학
- 사회/문화
  결혼, 기관/단체
- 연예/오락
  연예인, 영화, 음악, 유머
- 인문/예술
  디자인, 인문과학, 박물관
- 자연과학
  공학, 컴퓨터과학, 생물학
- 정부/공공기관
  정치, 한국정부, 국제기구
- 지역정보
  대한민국, 서울, 국가
- 참고자료
  도서관, 사람찾기 사전
- 컴퓨터
  인터넷, S/W, O/S, 통신

NAVER 추천

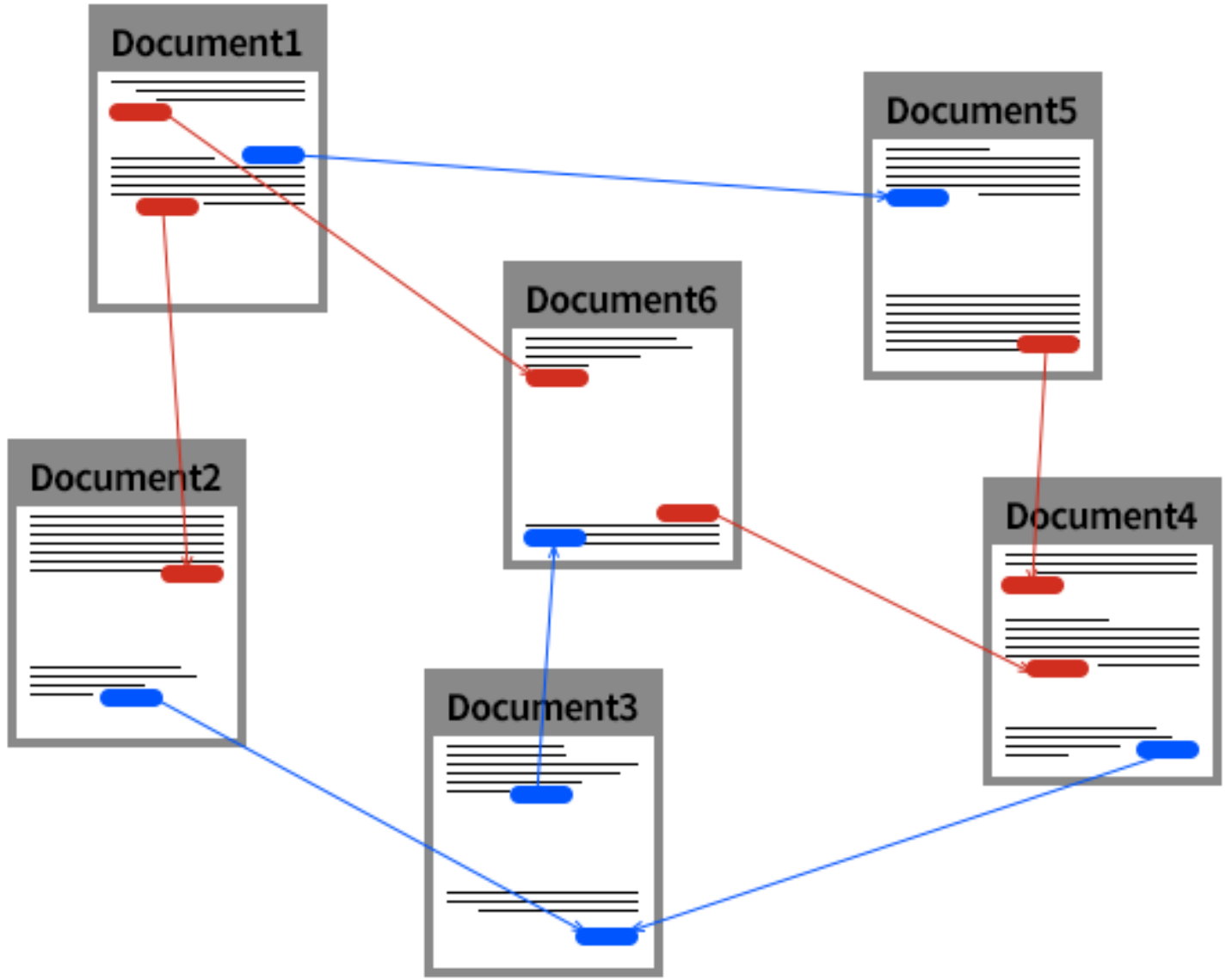
New! 크리스마스
행사,이벤트 여행 온라인쇼핑
영화,비디오 취업, 채용 투자,재테크

AT & T 전화접속서비스

도움말 | 새소식 | 홈페이지등록 | 다른검색사이트

Microsoft Internet Explorer

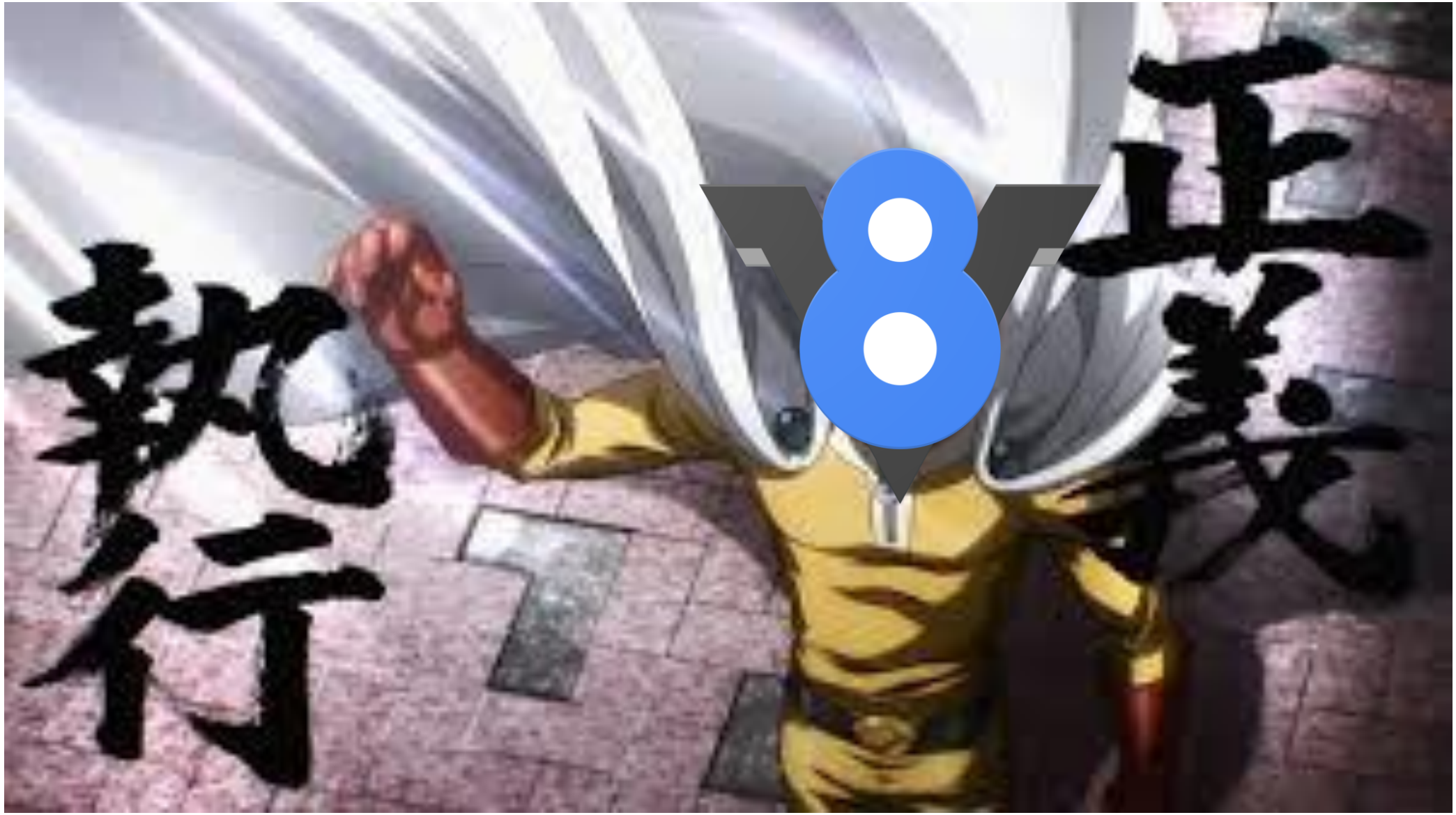

hypertext
development
internet
text
website
web
html
information
ComputerHope.com


Document1
Document5
Document6
Document2
Document4
Document3


인간의 욕심은 끝이 없고


JavaScript
<script>
if(age > 19){
alert("Adult");
} else{
alert("Teenager");
}
</script>

2008


執行
8
平美

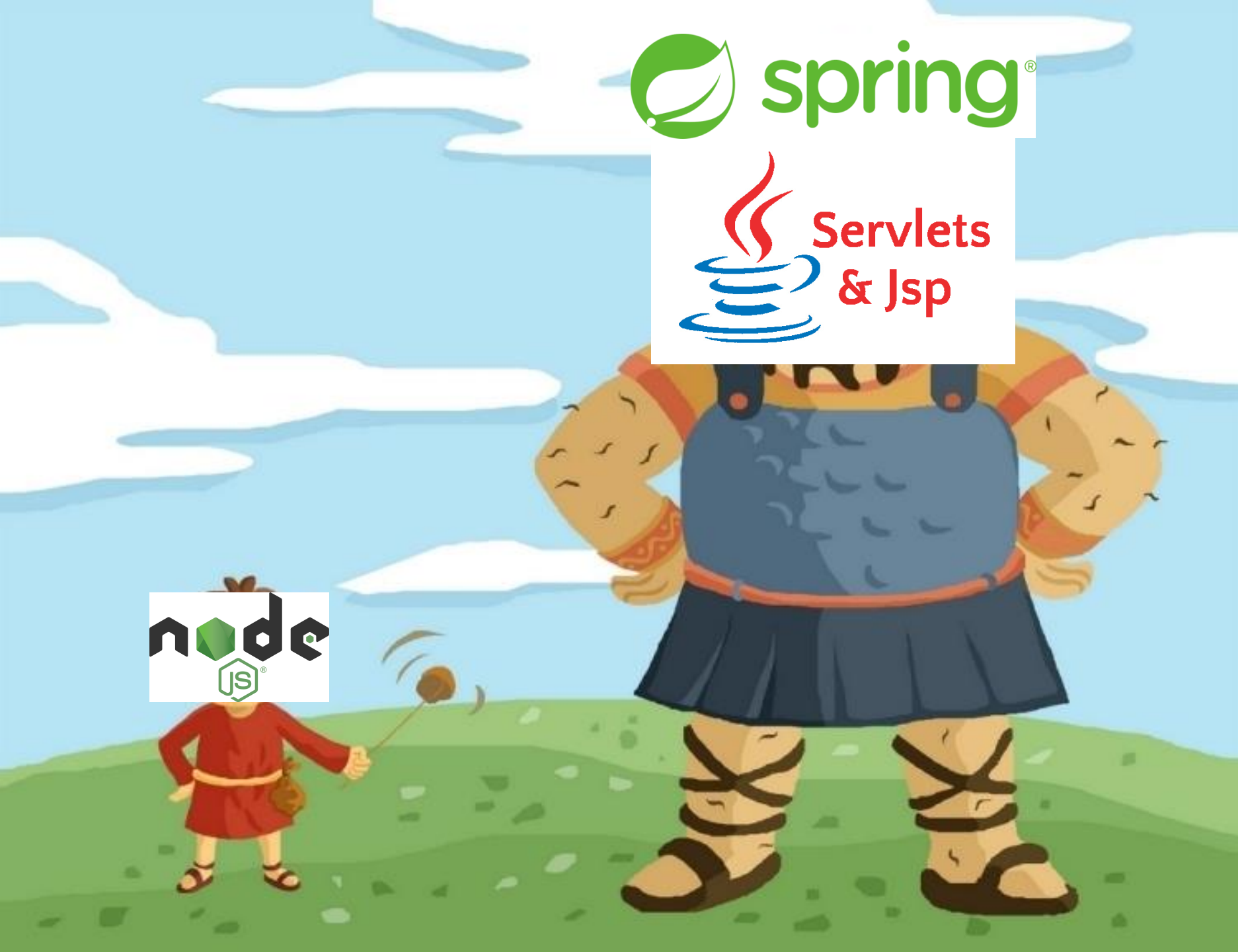
open source

1. 스크립트 언어라 어느 환경에서도 동작
2. 성능이 빠르고 좋음
3. Javascript 를 쓰는 개발자가 많음
4. Ryan Dahl이 서버용 기능을 개발!

node

JS®


JS
node
React


spring
Servlets
& Jsp
node
JS


크고 강한
spring
Servlets
& Jsp


대한민국 No.1
롯데렌터카
쉽고빠른
node
JS

NAVER

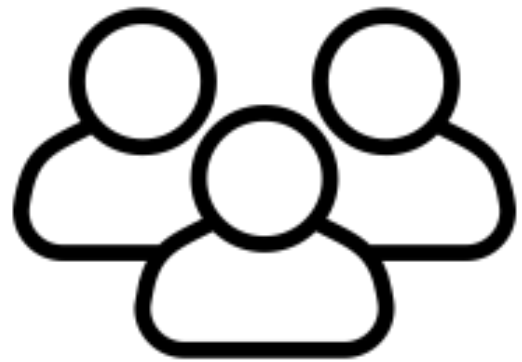
kakao

LINE

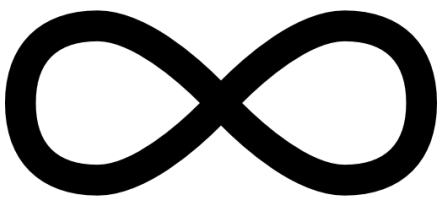
coupang

spring®

배달의민족



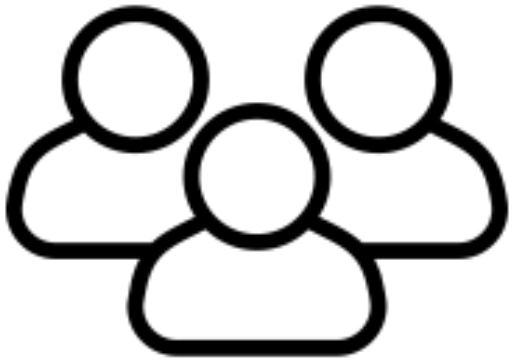
X




CJ ENM
tvN
반짝일 우리들의 이야기가 시작되는 곳
tvN 토일드라마
START-UP
스 타 트 - 업
배수지 · 남주혁 · 김선호 · 강한나 10월 17일 tvN 첫 방송 | 매주 [토일] 밤 9시 방송

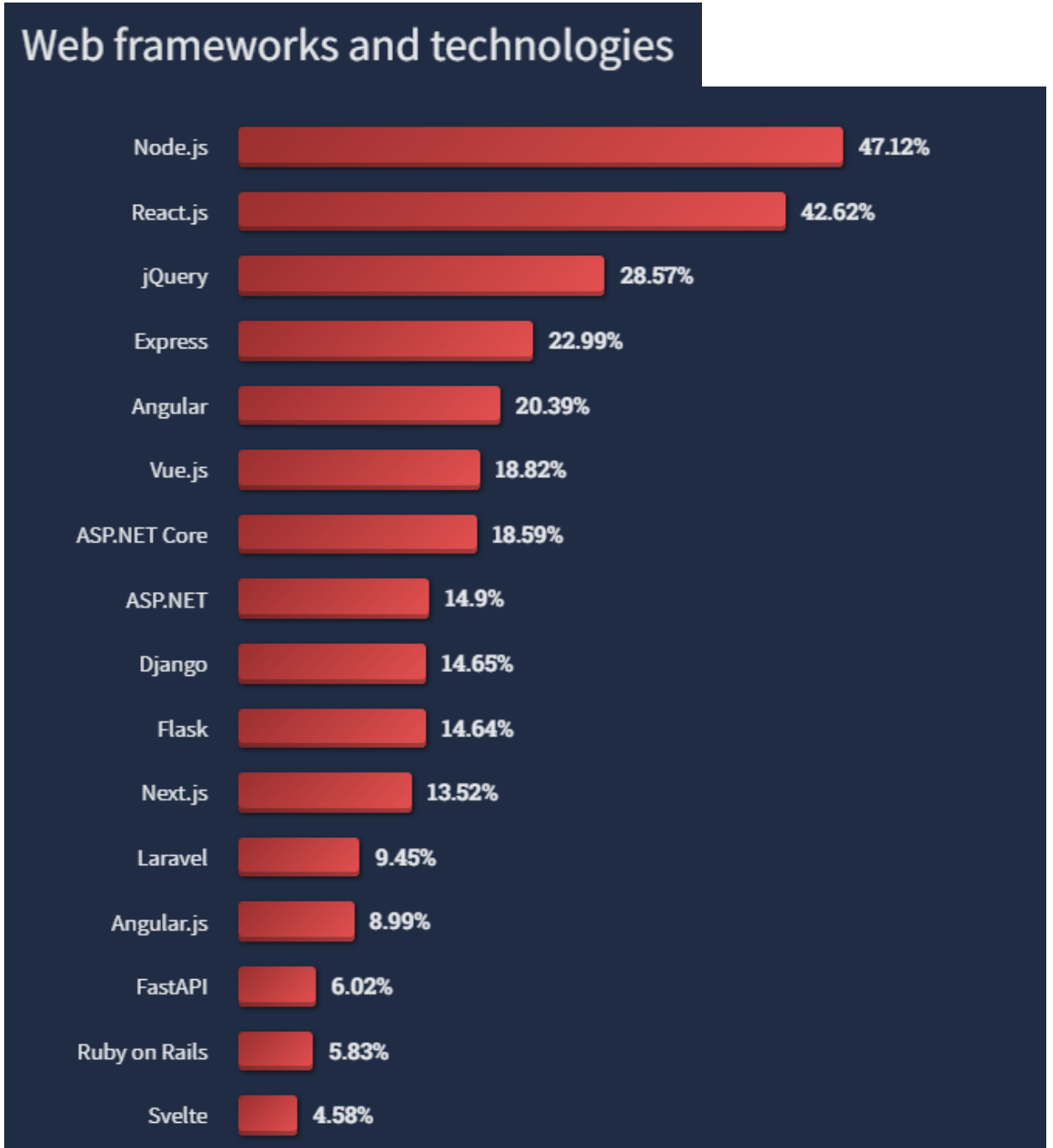

CHANGE

node JS

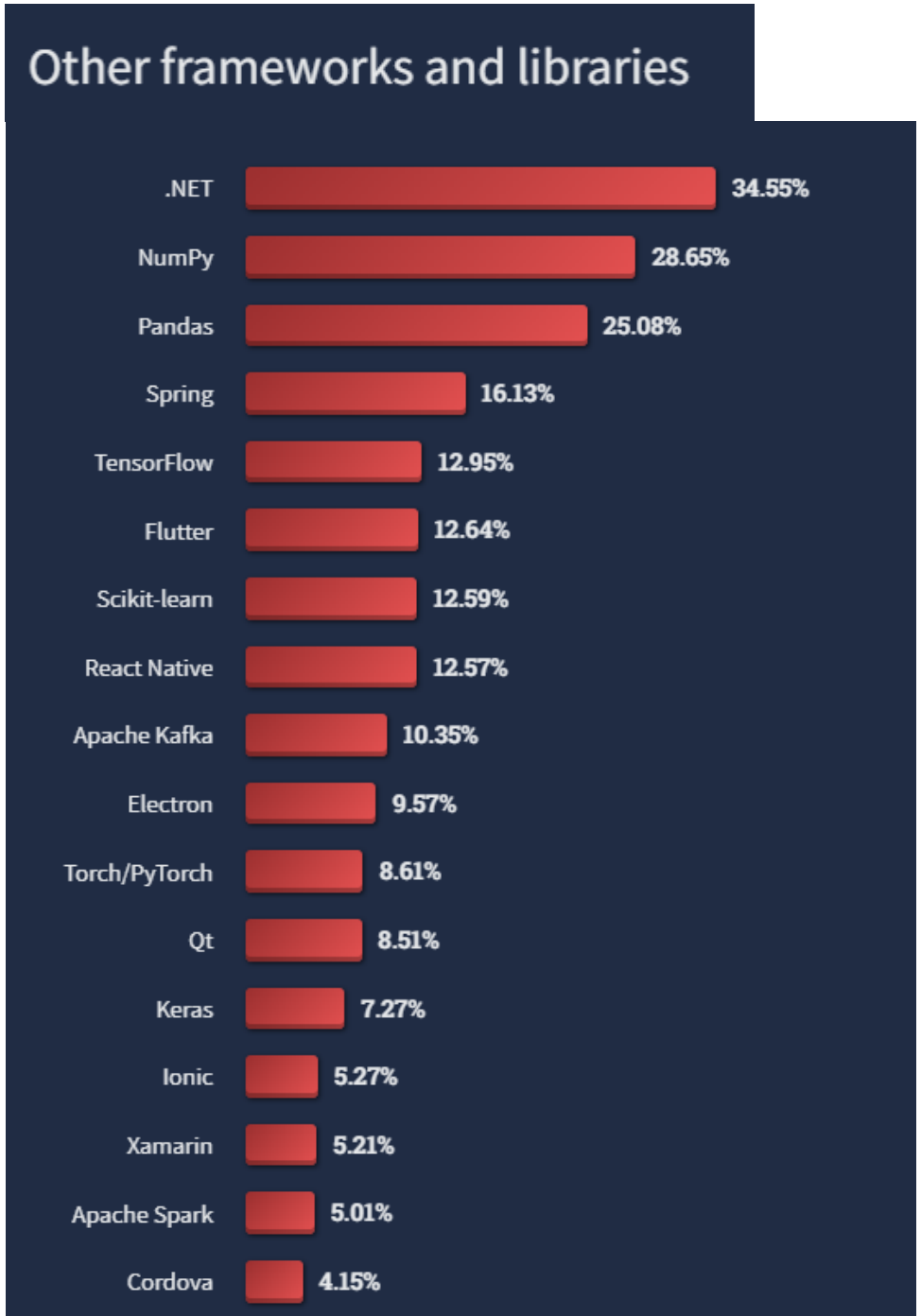


x ?

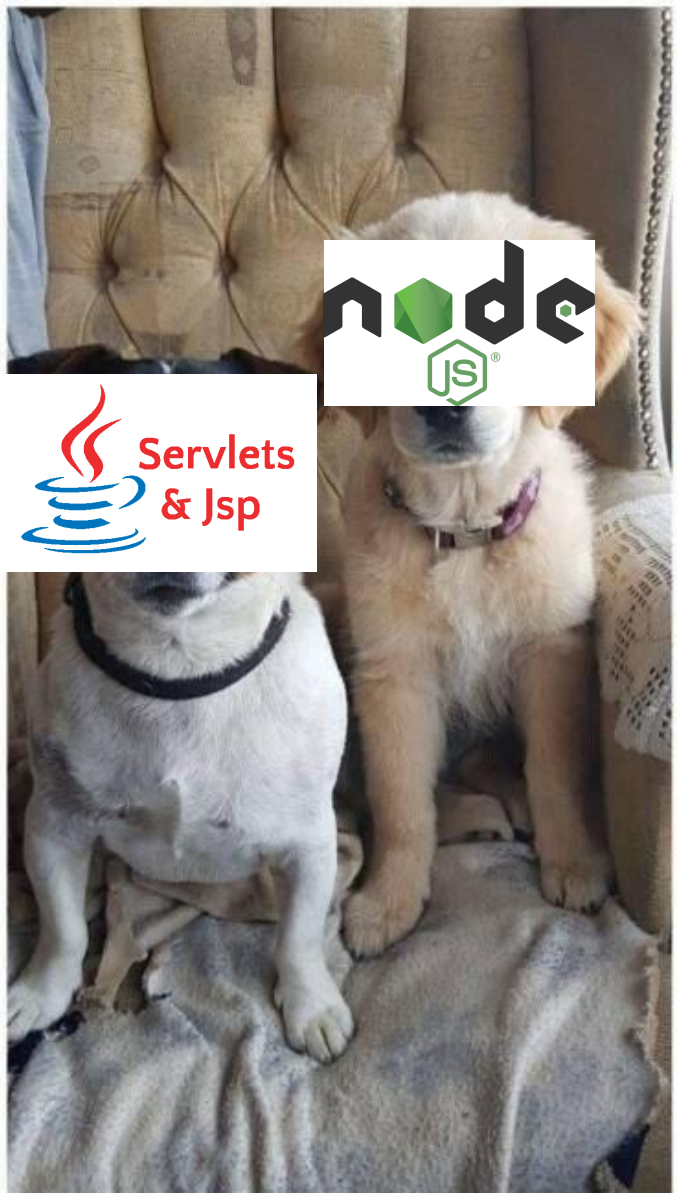
2022


Web frameworks and technologies
Node.js 47.12%
React.js 42.62%
jQuery 28.57%
Express 22.99%
Angular 20.39%
Vue.js 18.82%
ASP.NET Core 18.59%
ASP.NET 14.9%
Django 14.65%
Flask 14.64%
Next.js 13.52%
Laravel 9.45%
Angular.js 8.99%
FastAPI 6.02%
Ruby on Rails 5.83%
Svelte 4.58%

2022


Other frameworks and libraries
.NET 34.55%
NumPy 28.65%
Pandas 25.08%
Spring 16.13%
TensorFlow 12.95%
Flutter 12.64%
Scikit-learn 12.59%
React Native 12.57%
Apache Kafka 10.35%
Electron 9.57%
Torch/PyTorch 8.61%
Qt 8.51%
Keras 7.27%
Ionic 5.27%
Xamarin 5.21%
Apache Spark 5.01%
Cordova 4.15%

2009



2013

2017

2021

참고로 Node는 자바스크립트를 돌리는 런타임이고요. Node 안에서 여러 가지 패키지를 사용해서 서버를 만들 수 있습니다. 많이 쓰는것이 Express.js 이고요.

분명 추세는 Node.js + Express 로 REST API나 노드 + GraphQL 을 씁니다. 외국에선 이거 거의 정석이지만 좀 오래전부터 만들어진 사이트나 회사들의 시스템상 jsp도 많이 쓰고 있죠. 특히 우리나라 같이 익스플로어많이 썼던곳에서 jsp는 아직도 많은 기업에서 쓰입니다.

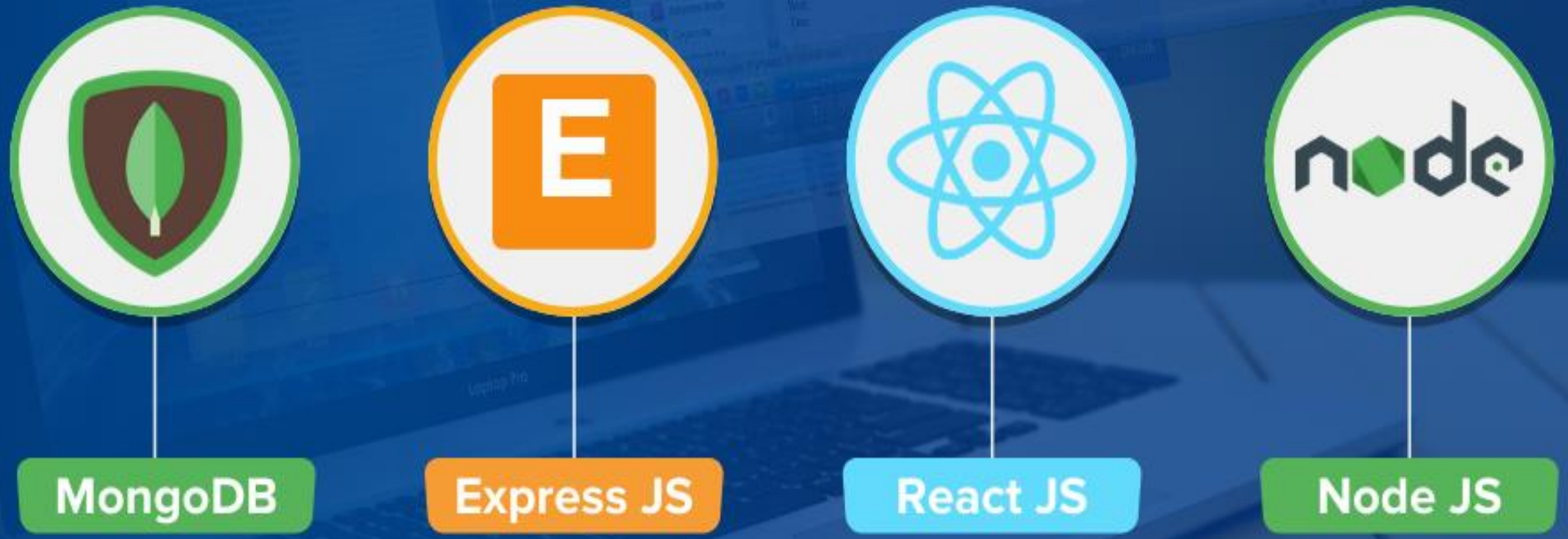
아무리 더 좋은 프레임워크가 나왔다고 해도, 인력의 숙련도, 해당언어를 할 수 있는 개발자 숫자 (가격이 싸지고 빠르게 고용해서 쓸 수 있으니까요), 이미 개발된 환경등등이 영향을 끼치죠. 특히 웹개발에 갈라파고스화 되버려서 (지금은 많이 변하고 있습니다) 아직까지 jsp가 많이 쓰이고요. 하지만 node.js도 많이 적용 되는것을 보고 있어서, 이미 시스템들이 정형화된 기업이나 그렇게 필요하지 않은 기업들을 제외한곳은 node로 많이 넘어갈 것으로 보이기 때문에, 지금부터 node를 배우시고 취업도 왠만하면 node 사용하는 곳으로 가세요.

참고로 MySQL이나 MariaDB보다 NoSQL인 MongoDB 를 Node JS와 많이 씁니다. 그래서 SPA중 하나인
React + Nodejs + Express + MongoDB를 합쳐서
MERN Stack이라고 하고 해외에서는 거의 MERN으로 웹사이트를 만든다고 보시면 됩니다.

rlogical techsoft pvt. ltd

MERN Stack

Best for Developing Web Apps



방대한 오픈 소스 생태계

By the numbers

Packages

1,614,651

Downloads · Last Week

30,687,522,522

Downloads · Last Month

126,688,929,038


JS
node
React


가 즈 아
https://coinpan.com 코인판 on 2018-01-09
K316
디지털
미디어시티
Digital Media City
K315
가 좌
Gajwa 加佐
P314
신 촌
Sinchon 新村
KORAIL
동서가구
민 성 한 의 원
373-1651

개발 환경 세팅

Node.js 설치

설치

https://nodejs.org/ko/

Node.js 버전 관리 도구 설치

버전 관리 Tool 설치

https://github.com/coreybutler/nvm-windows/releases

NVM 버전 확인

→ nvm version

```
tetz@DESKTOP-P7Q4OLL MINGW64 /d/git2/mask-ani (main)
$ nvm version
1.1.9
```

Node.js 버전 확인

→ nvm ls

```
tetz@DESKTOP-P7Q4OLL MINGW64 /d/git2/mask-ani (main)
$ nvm ls

  * 16.14.1 (Currently using 64-bit executable)
```

Node.js 다른 버전 설치

→ nvm install v16.16.0(버전 번호)

```
tetz@DESKTOP-P7Q4OLL MINGW64 /d/git2/mask-ani (main)
$ nvm install v16.16.0
Downloading node.js version 16.16.0 (64-bit)...
Extracting...
Complete


Installation complete. If you want to use this version, type

nvm use 16.16.0
```

Node.js 다른 버전 사용

→ nvm use 16.16.0

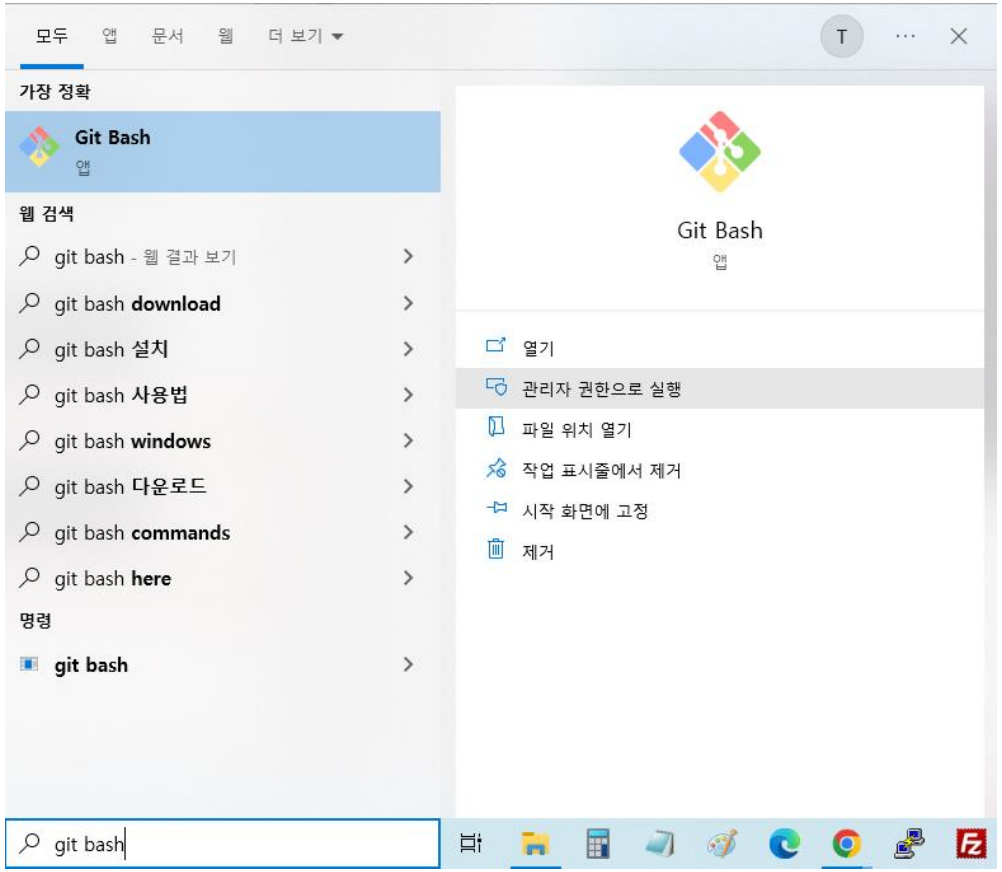
```
tetz@DESKTOP-P7Q4OLL MINGW64 /d/git2/mask-ani (main)
$ nvm use 16.16.0
exit status 5: ◆◘◆◆◆◆◆ ◆é∈Ó◆◆◆◆◍◆

exit status 1: ◆◆◆◆◆◆ ◆◆ ◆◆◆◆◆◘ ◆ ◆◆◆◆ ◆◆ ◆◆◆◆◆◍◆
```

```
tetz@DESKTOP-P7Q4OLL MINGW64 /d/git2/mask-ani (main)
$ nvm ls

    16.16.0
  * 16.14.1 (Currently using 64-bit executable)
```

Git Bash를 관리자 권한으로 실행하기!




```
tetz@DESKTOP-P7Q4OLL MINGW64 ~
$ nvm ls

    16.16.0
  * 16.14.1 (Currently using 64-bit executable)

tetz@DESKTOP-P7Q4OLL MINGW64 ~
$ nvm use 16.16.0
Now using node v16.16.0 (64-bit)
```

```
tetz@DESKTOP-P7Q4OLL MINGW64 /d/git2/mask-ani (main)
$ nvm ls

  * 16.16.0 (Currently using 64-bit executable)
    16.14.1
```

버전 관리 Tool 설치

tj/n

#641 "no version found" error on Mac with M1 chip

18 comments

shadowspawn opened on November 27, 2020

https://github.com/tj/n

N 설치

```
 ~/Desktop/node-setting npm install -g n
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 1 package, and audited 2 packages in 2s

found 0 vulnerabilities
```

Node.js 버전 별 설치

- n latest (최신 버전 설치) / n lts (안정 버전 설치)
- n 16.16.0 (원하는 버전 설치)
- 설치가 안되면 sudo 로 권한 주고 설치!

```
 ~/Desktop/node-setting  sudo n latest
Password:
  installing : node-v18.7.0
       mkdir : /usr/local/n/versions/node/18.7.0
       fetch : https://nodejs.org/dist/v18.7.0/node-v18.7.0-darwin-arm64.tar.xz
     copying : node/18.7.0
   installed : v18.7.0 to /usr/local/bin/node
      active : v16.16.0 at /Users/lhs/.nvm/versions/node/v16.16.0/bin/node
```

Node.js 버전 선택!

- n → 버전 선택하기!

```
    node/16.17.0
    node/18.7.0

Use up/down arrow keys to select a version, return key to install, d to delete, q to quit
```

NPM?

(Node Package Manager)

1,614,651 개의 패키지!

By the numbers

Packages

1,614,651

Downloads · Last Week

30,687,522,522

Downloads · Last Month

126,688,929,038


BOOTH
STREET
RAILWAYS
GRIEB
DICTIONARY
ENGLISH
GERMAN
GRIEB
DICTIONARY
GERMAN
ENGLISH
GRIEB
DICTIONARY
DER NATUR-
WISSENSCHAFTEN


string
sticker


택배 왔다!!

Package?

- 여러분이 html 에서 가져와서 사용하던 라이브러리를 생각하시면 됩니다!
- Swiper 라던가 Swiper 라던가… Swiper 라던가……
- 단, Node.js Package 는 마치 택배를 받는 것 처럼 직접 컴퓨터에 설치를 하여 사용을 합니다~!

NPM(Node Package Manager)?

NPM(Node Package Manager)?

패키지 관리가 필요하겠죠?

- 이러한 패키지를 관리하는 것이 바로
- Node Package Manager → NPM 입니다!

패키지 관리가 필요하겠죠?

- 프로젝트를 위해 어떤 패키지를 설치 해야 할까요?
- 패키지 버전에 따라 충돌이 발생할 수 있지 않을까요?
- 이러한 문제를 하나의 JSON 파일로 관리해 줍니다!

패키지 관리 시작하기!

- npm init -y

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm init -y
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
Wrote to C:\Users\tetz\Desktop\node-setting\package.json:

{
  "name": "setting",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

패키지 관리 시작하기!

- Package.json 파일이 생성 됩니다!
- Node.js 관련 package의 모든 정보가 여기에 들어 갑니다!

```
{
  "name": "setting",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Package.json 파일 알아보기!

```
{
  "name": "setting",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Package.json 파일 알아보기!

- npm run test

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm run test
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

> setting@1.0.0 test
> echo "Error: no test specified" && exit 1

"Error: no test specified"

tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$
```

Package.json 파일 변경하기!

```
{
  "scripts": {
    "test": "echo \"Hello, Node.js\""
  }
}
```

- npm run test

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm run test
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

> test
> echo "Hello, Node.js"

"Hello, Node.js"
```

Package 설치하기!

- npm install left-pad

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/kdt-5th-proj1 (main)
$ npm install left-pad
npm WARN deprecated left-pad@1.3.0: use String.prototype.padStart()

added 1 package, and audited 174 packages in 647ms

12 packages are looking for funding
  run `npm fund` for details

4 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

```
{
  "dependencies": {
    "left-pad": "^1.3.0"
  }
}
```

Package 삭제하기

- npm uninstall left-pad

```
lhs@DESKTOP-86MUCGC MINGW64 /d/git/kdt-5th-proj1 (main)
$ npm uninstall left-pad

removed 1 package, and audited 173 packages in 678ms

12 packages are looking for funding
  run `npm fund` for details

4 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

```
{
  "dependencies": {}
}
```

Package.json 만 활용하기!

- 빈 폴더에 package.json 파일만 카피를 해봅시다
- Npm install
- 실행을 하면 package.json 를 참고하여 해당 package 가 설치 됩니다!
- 따라서, 추후에 협업 시 node_modules 폴더를 보낼 필요 없이 package.json 만 공유가 되면 편리하게 관리가 가능합니다~!


그란데말입니다
그것이 알고싶다

Node.js 가 뭐였죠?

Node.js®는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다.

다운로드 - Windows (x64)

16.16.0 LTS
안정적, 신뢰도 높음

18.7.0 현재 버전
최신 기능

다른 운영 체제 | 변경사항 | API 문서

다른 운영 체제 | 변경사항 | API 문서

LTS 일정은 여기서 확인하세요

- 자바 스크립트 파일을 실행 시켜 봅시다!

Node.js 가 뭐였죠?

- main.js 파일 만들기

```
console.log("Hello, Node.js");
```

- Node main.js

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ node main.js
Hello, Node.js
```

Formatting,

Linting,

Typescript 세팅!

Formatting

Formatting?

README.md — prettier-config-example

README.md

```
# Prettier support for other languages

This folder has JavaScript, CSS, HTML, JSON and even this Markdown file all formatted using
Prettier

Note that while formatting [index.js](index.js) Prettier uses 2 spaces per tab, but in the
Markdown code blocks it uses 4 spaces.

```js
const code = true;if (code) {console.log('code is on')}
```

This is because we override options inside the .prettierrc.json file.
```

Formatting

- Code의 스타일을 통일 시켜 줍니다!
- 함수의 소괄호와 중괄호는 띄울 것인지? 세미 콜론은 찍을 것인지? 탭을 누르면 몇 칸을 띄울 것인지? 등등등
- 문법이 아닌 코드의 스타일을 통일 시켜줘서 가독성을 높이고 버그를 예방합니다!
- Prettier 를 사용!

프로젝트에 Prettier 설정하기!

- npm install --save-dev prettier

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm install --save-dev prettier
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 1 package, and audited 2 packages in 660ms

1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- --save 는 패키지 모듈에 추가한다는 의미입니다!
- --save-dev 는 개발할 때에만 사용하겠다는 의미 입니다!
  - 실제로 프로젝트를 빌드 & 배포하면 해당 패키지는 포함 X

프로젝트에 Prettier 설정하기!

```
{
  "scripts": {
    "test": "echo \"Hello, Node.js\""
  },
  "devDependencies": {
    "prettier": "^2.7.1"
  }
}
```

- Package.json 파일에 변화가 생겼죠?
- 방금 설치한 Prettier가 Package.json에 추가 되었습니다!
- 즉, 이렇게 npm 은 프로젝트의 패키지를 관리합니다!

프로젝트에 Prettier 설정하기!

- .prettierrc 파일로 prettier 세부 설정하기!

```
{
    "semi": true,
    "singleQuote": true
}
```

- Vs-code 에게 prettier 사용하라고 알려주기!
  - 프로젝트 폴더 최상단에 .vscode 폴더 만들고 settings.json 파일 만들기

```
{
    "[javascript]": {
        "editor.formatOnSave": true,
        "editor.defaultFormatter": "esbenp.prettier-vscode"
    }
}
```

프로젝트에 Prettier 설정하기!

- Prettier 확장 설치
- Main.js 파일의 코드를

```
console.log("Hello, Node.js")
```

- 로 작성하고 ctrl + s 로 저장해보기!!
- Prettier 가 작동합니다! :)

Linting

Linting?

Linting

- Formatting 에 가깝지만 더 많은 규약과 규율을 검사해주는 방법입니다!
- 웹 개발에서는 Airbnb 에서 사용하는 Linting 규율이 유명합니다!

ESLint 설치하기!

- npm install --save-dev eslint

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm install --save-dev eslint
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 111 packages, and audited 113 packages in 3s

25 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

- package-lock.json 파일을 보면 prettier에 비해 훨씬 많은 package 가 설치 되었음을 확인이 가능합니다 → 그만큼 많은 규약을 가지고 있다는 것!

ESLint 설정하기!

- .eslintrc.js 파일 생성

```
module.exports = {};
```

- 모든 Lint 관련 룰을 우리가 전부 지정 할 수는 없겠죠?
- Airbnb의 Linting Rule를 가져 옵시다!!
- npm install --save-dev eslint-config-airbnb-base eslint-plugin-import

ESLint 설정하기!

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm install --save-dev eslint-config-airbnb eslint-plugin-import
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 88 packages, and audited 201 packages in 6s

68 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
{
  "scripts": {
    "test": "echo \"Hello, Node.js\""
  },
  "devDependencies": {
    "eslint": "^8.22.0",
    "eslint-config-airbnb": "^19.0.4",
    "eslint-plugin-import": "^2.26.0",
    "prettier": "^2.7.1"
  }
}
```

ESLint 설정하기!

- .eslintrc.js 파일을 생성 후, Airbnb 모듈 추가!

```
module.exports = {
  extends: ['airbnb-base'],
};
```

- Windows 사용자라면 LF, CRLF 문제 해결을 위해 아래의 코드도 추가!

```
module.exports = {
  extends: ['airbnb-base'],
  rules: {
    'linebreak-style': 0,
    'no-console': 'off',
  },
};
```

ESLint 체험하기!

- main.js 파일로 고고고고!

```
var x = 1;
console.log(x);
```

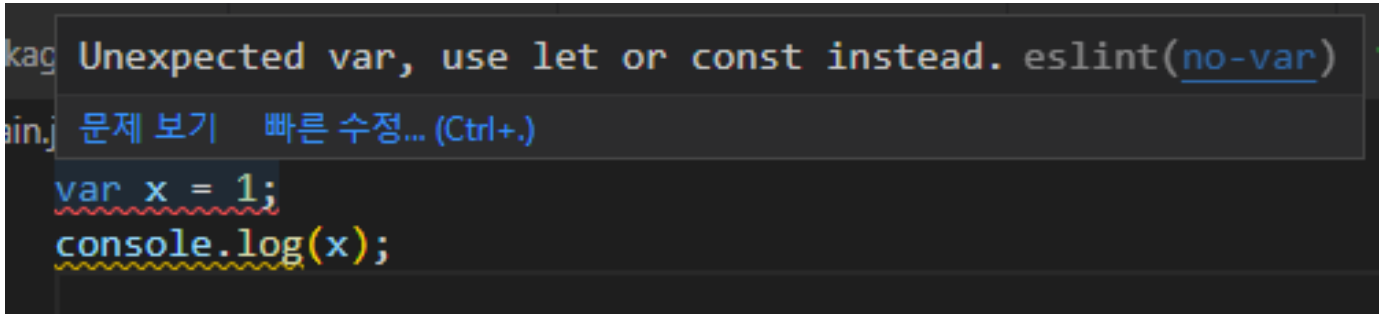
- 이 코드에 문제가 있을까요?

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ node main.js
1
```

- 실행을 해도 문제는 없습니다!

ESLint 체험하기!

- Javascript 상으로는 문제가 없지만 ESLint 의 규약에는 문제가 생깁니다!



- Var 는 위험하지 쓰지 말고 let 이나 const 를 쓰라고 하네요?

ESLint 체험하기!

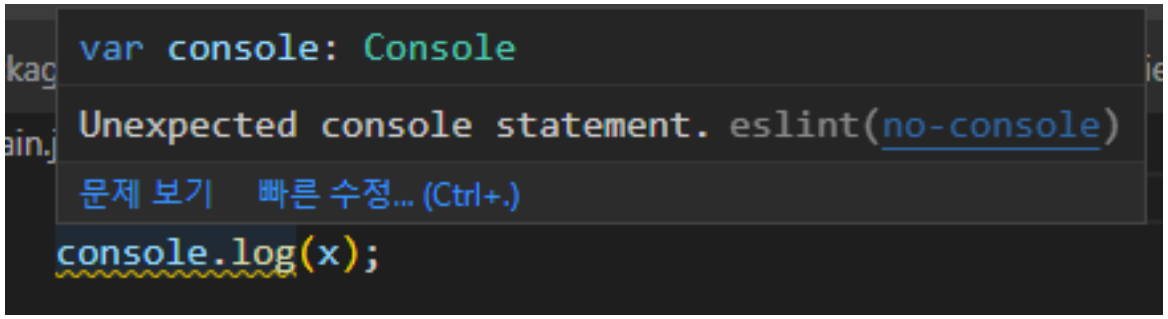
- Var 를 let 으로 바꾸면?

```
let x = 1;
cons
    let x: number
    'x' is never reassigned. Use 'const' instead. eslint(prefer-const)
    문제 보기   빠른 수정... (Ctrl+.)
```

- Let 은 변화하는 변수일 경우에만 할당하는 편이 좋으니 const 를 쓰라고 합니다!
- 즉, 이렇게 개발자가 간과할 수 있는 Rule 들을 바로바로 알려주기 때문에 나중에 발생할 예상외의 버그 또는 문제를 많이 해결해 줍니다!

ESLint 체험하기!

- Console.log 도 노란줄이 있네요?



- Console.log 는 보통 디버깅용으로 쓰다보니 사용자에게 출시 할 때는 대부분 제거를 해야만 합니다! 그렇다 보니 사용자에게 특정 목적을 알리기 위해 따로 만든 console statement 가 아닌 것은 쓰지 말라는 것이죠!
- 하지만 이건 너무 가혹하니까!! Rule 을 수정 합시다!

ESLint 체험하기!

- .eslintrc.js 로 고고고

```
module.exports = {
  extends: ['airbnb-base'],
  rules: {
    'linebreak-style': 0,
    'no-console': 'off',
  },
};
```

- No-console 옵션을 꺼줍니다!

```
JS main.js > ...
1   let x = 1;
2   console.log(x);
3
```


그분들
adidas
CHEVROLET
해축드립사전


C O
D E

니가 그렇게 코딩을 잘해!?

옥땅으로 따라와!!!

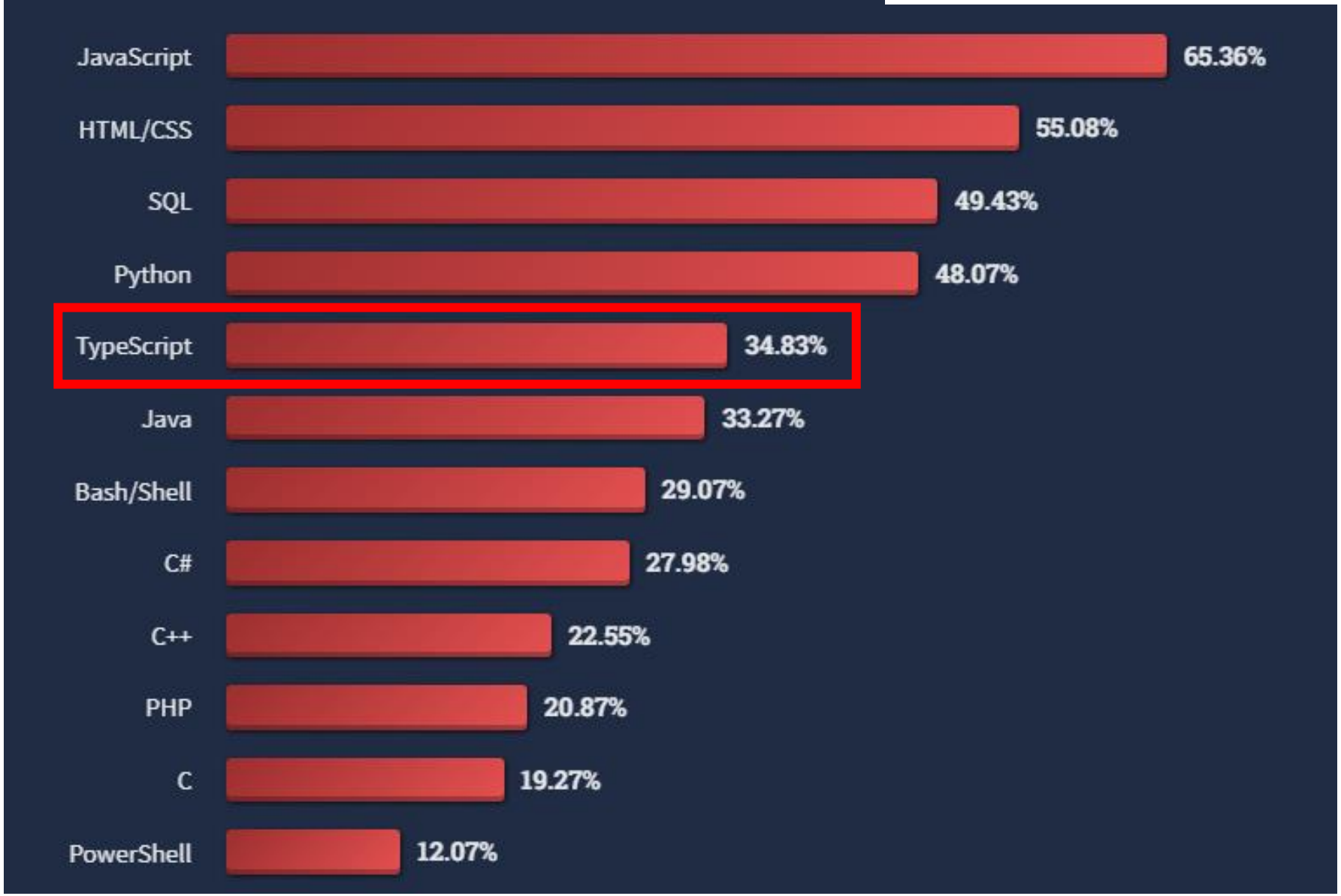
Guide

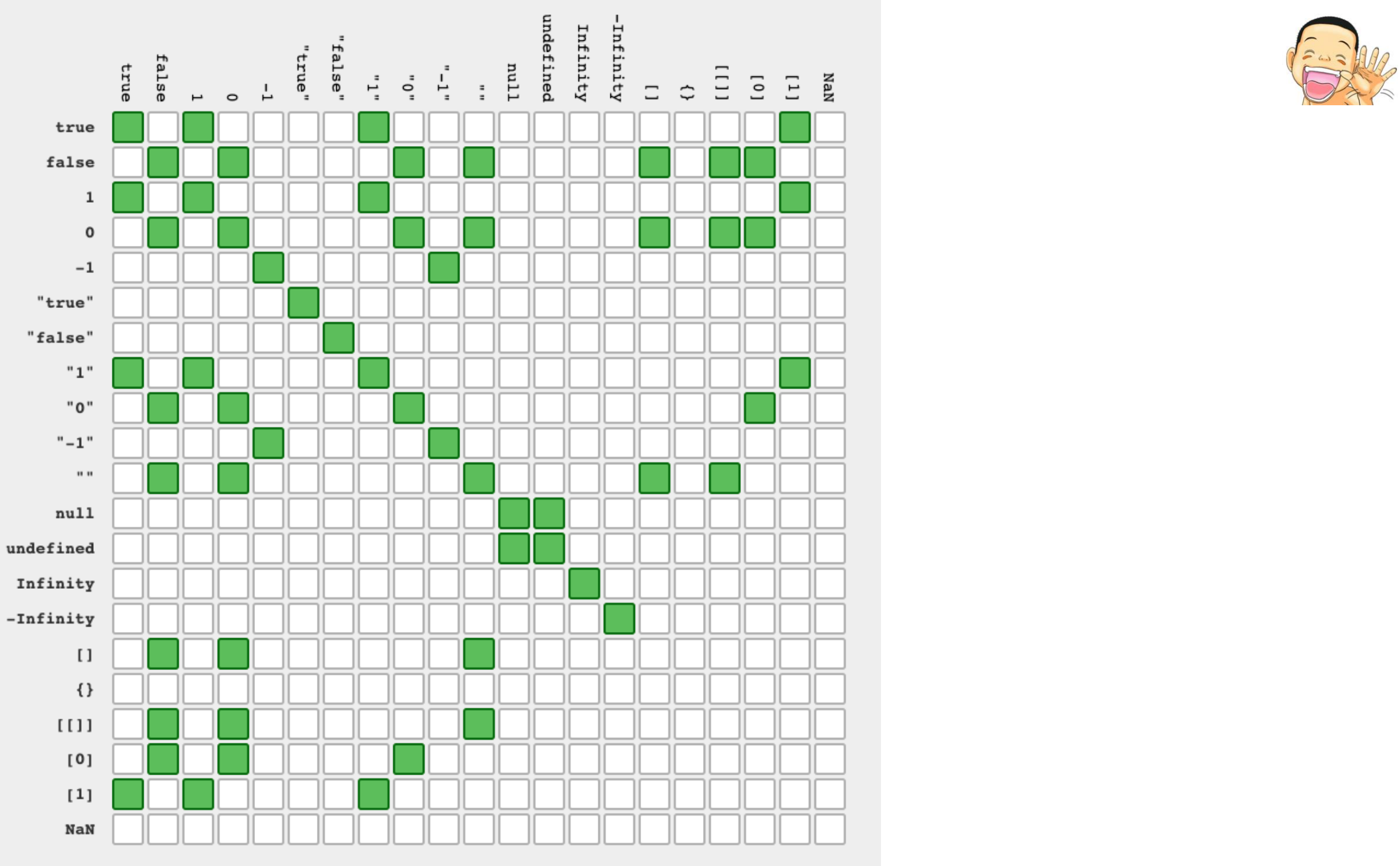
협업왕이 되어보자!

#개발자 #디자이너
#후배 #외부업체

Typescript

Programming, scripting, and markup languages




true
false
1
0
-1
"true"
"false"
"1"
"0"
"-1"
""
null
undefined
Infinity
-Infinity
[]
{}
[[]]
[0]
[1]
NaN

Bucketplace

주요업무

- 애플리케이션 컴포넌트 설계, 개발, 테스트 및 운영
- 모니터링과 경험을 통한 이용자의 기술적 문제 도출 및 제안
- 서비스 아키텍처 설계부터 출시까지 전 과정 참여
- BFF(Backend For Frontend) 설계, 개발 및 운영
- React, TypeScript 기반의 오늘의집 웹 서비스 개발

자격요건

- 만 4년 이상의 Frontend 개발 경력을 보유하신 분
- 컴퓨터공학 전공 혹은 그에 준하는 전공 및 지식을 보유하신 분
- React에 대한 이해 및 프로젝트 경험이 있는 분
- 서버 사이드 렌더링(SSR)에 대한 이해 및 처리 경험이 있는 분
- MSA, BFF(Backend For Frontend)에 대한 이해가 있는 분
- SPA Framework에 대한 이해가 있는 분

오늘의집

회사소개

[개발자 대규모 경력채용] Technical Lead & Manager, Backend

[개발자 대규모 경력채용] Senior Software Engineer, Backend

[개발자 대규모 경력채용] Software Engineer, Backend

[개발자 대규모 경력채용] Technical Lead & Manager, Frontend

[개발자 대규모 경력채용] Senior Software Engineer, Frontend

[개발자 대규모 경력채용] Software Engineer, Frontend

[개발자 대규모 경력채용] Technical Lead & Manager, iOS

[개발자 대규모 경력채용] Senior Software Engineer, iOS

[개발자 대규모 경력채용] Senior Software Engineer, Android

[개발자 대규모 경력채용] Software Engineer, Android

[개발자 대규모 경력채용] Senior Software Engineer, Data

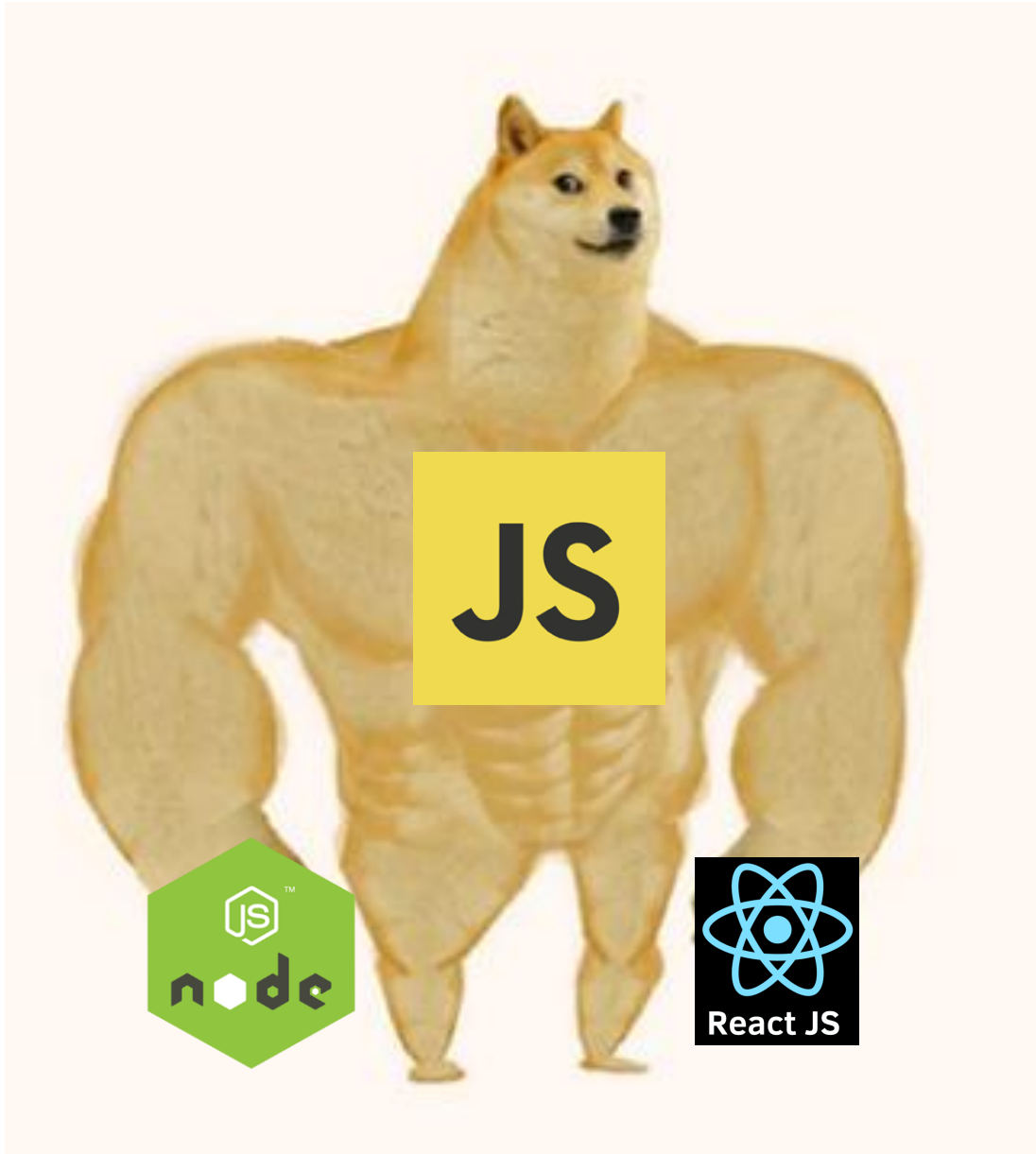
[개발자 대규모 경력채용] Senior Software Engineer, Machine Learning

[개발자 대규모 경력채용] Software Engineer, Machine Learning

[개발자 대규모 경력채용] Senior Software Engineer, Extended Reality

[개발자 대규모 경력채용] Senior Software Engineer, EngProd

[개발자 대규모 경력채용] Senior Technical Program Manager


JS
node
React JS


SATURDAY
NIGHT
LIVE
tvN
생방송
난 어디서 경력을 쌓나
면접 전쟁

자, 코드를 봅시다!

- ES Lint 상으로는 문제가 없는 코드입니다!

```
const str = 'Hello';
const num = Math.log(str);
console.log(num);
```

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ node main.js
NaN
```

- 그런데 사실 프로그래밍 적으로 NaN 은 쓸 이유가 없는 타입입니다!

Typescript 설치

- 이러한 문제를 막아 주는 것이 Typescript!
- npm install --save-dev typescript

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm install --save-dev typescript
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

changed 1 package, and audited 202 packages in 2s

68 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Typescript 적용

- Main.js 파일에 // @ts-check 주석 추가

```
// @ts-check
```

```
JS main.js > ...
1   // @ts-check
2
3   const str = 'Hello';
4   const num = Math.log(str);
5   console.log(num);
6
```

```
const str: "Hello"
'string' 형식의 인수는 'number' 형식의 매개 변수에 할당될 수 없습니다. ts(2345)
문제 보기   빠른 수정... (Ctrl+.)
```

- 이제 Type 관련 문제는 Typescript 가 알려 줍니다!

Type 관련 문제는
TS가 처리했으니 안심하너굴!

Typescript & Node

Node.js 에도 다양한 Type 이 존재 하므로!

- Node.js 에서 사용되는 수 많은 객체들에 대한 Type 정보 체크는 Typescript 만으로는 불가능 합니다!
- 그럼 해당 정보를 아는 package 를 설치하면 되겠죠?
- npm install –save-dev @types/node

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ npm install --save-dev @types/node
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.

added 1 package, and audited 203 packages in 566ms

68 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

그럼 간단한 Node.js 서버를 만들어 봅시다!

- 대부분 잘 모르시겠지만 일단 따라서 입력해 주세요!
- 이제 부터 하나하나 배울 예정입니다!

```
// @ts-check

const http = require('http');

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.end('Hello');
});

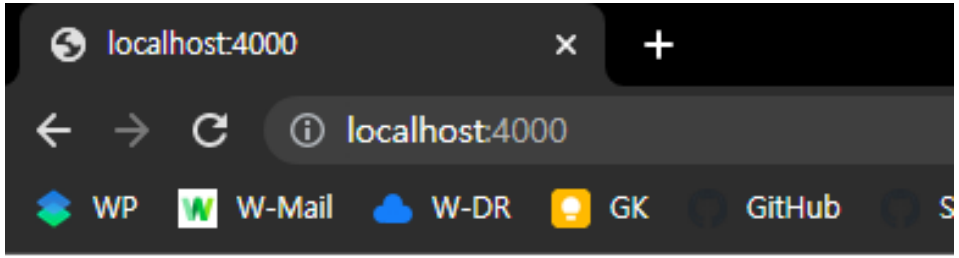
const PORT = 4000;
server.listen(PORT, () => {
  console.log(`The server is listening at port: ${PORT}`);
});
```

서버 실행 시키기!

- node main.js

```
tetz@DESKTOP-P7Q4OLL MINGW64 ~/Desktop/node-setting
$ node main.js
The server is listening at port: 4000
```

- 브라우저를 열어서 localhost:4000 입력하기



Hello

여러분은 방금 처음으로 백엔드 서버를 만드셨어요!!

Typescript & Node

Node & Typescript!

```
in.js > [@] server > http.createServer() callback
// @ts
                any
const           'statuscode' 속성이 'ServerResponse' 형식에 없습니다. 'statusCode'을(를) 사용하시겠습니까? ts(2551)
                http.d.ts(487, 9): 여기서는 'statusCode'이(가) 선언됩니다.
const           문제 보기   빠른 수정... (Ctrl+.)
  res.statuscode = 200;
  res.end('Hello');
});

const PORT = 4000;
server.listen(PORT, () => {
  console.log(`The server is listening at port: ${PORT}`);
});
```

Node & Typescript!

```
1   // @ts-check
2
3   const http = require('http');
4
5   const server = http.createServer((req, res) => {
6     res.statusCode = '200';
7
8   })
9
10  co
11  se
12
13  })
14
```

(property) ServerResponse.statusCode: number

When using implicit headers (not calling `response.writeHead()` explicitly), this property controls the status code that will be sent to t
flushed.

```
response.statusCode = 404;
```

After response header was sent to the client, this property indicates the status code which was sent out.

@since — v0.4.0

'string' 형식은 'number' 형식에 할당할 수 없습니다. ts(2322)

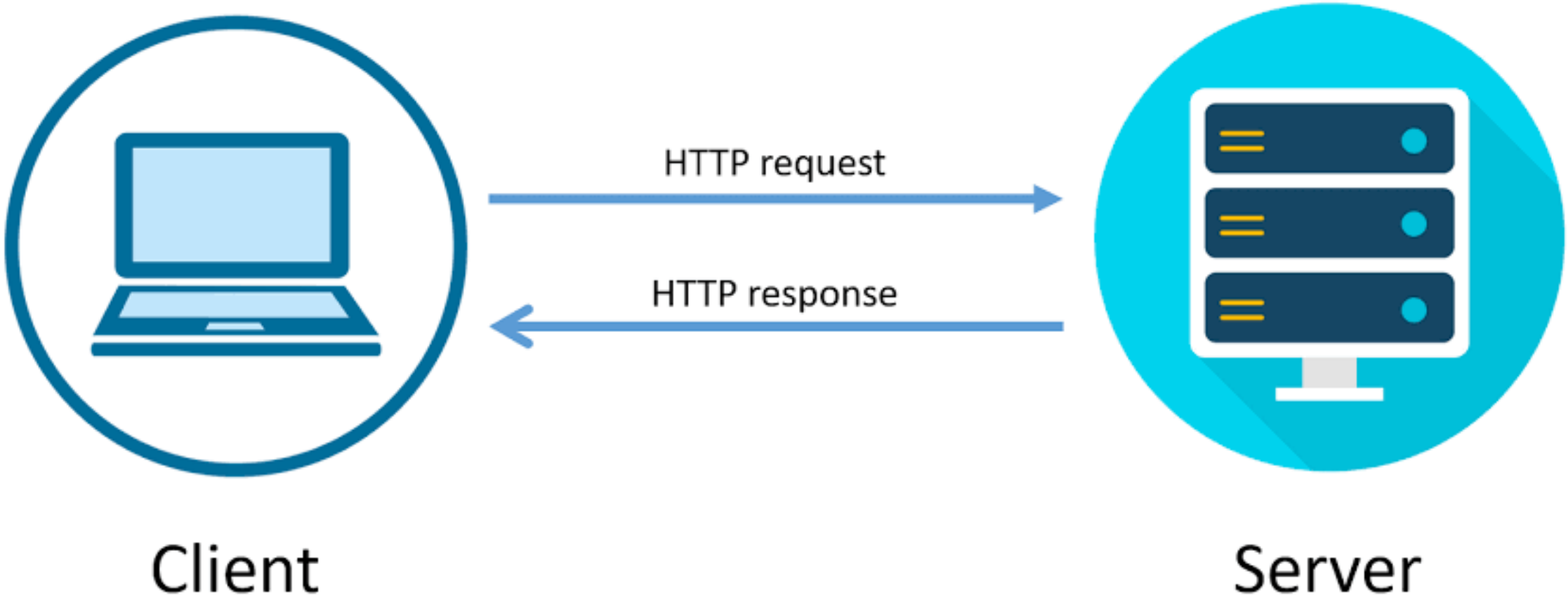
문제 보기 빠른 수정... (Ctrl+.)


node
JS
자 이제 시작이야 Back-End

간단한

통신 경험하기!


요청
클라이언트
서버
응답


HTTP request
HTTP response
Client
Server

그럼 간단한 Node.js 서버를 만들어 봅시다!

- Node.js 에서 제일 유명한 백엔드 프레임워크를 사용할 예정입니다.
- 해당 부분은 추후에 자세히 배울 예정이므로 오늘은 간단히 이렇게 되는 구나 체험만 하시면 됩니다!
- 지금부터 모든 서버는 Express 를 사용하여 구현이 될 예정입니다!

Express 서버 구축

- Express 설치
  - Npm install express
- CORS(Cross Origin Resource Sharing) 이슈를 피하기 위한 cors 모듈 설치
  - Npm install cors

Express 서버 구축

backend/server.js

```
const express = require('express');
const cors = require('cors');

const PORT = 4000;

const app = express();

app.use(cors());

app.use('/', (req, res) => {
  const str = '안녕하세요. 여기는 백엔드 입니다!';
  const json = JSON.stringify(str);
  res.send(json);
});

app.listen(PORT, () => {
  console.log(`데이터 통신 서버가 ${PORT}에서 작동 중입니다!`);
});
```

프론트에서 정보 요청하기

- 이제 백엔드에서 데이터를 보내는 기능은 만들었으니 프론트에서 백엔드로 요청을 보내 봅시다!
- 가장 기본적으로 사용되는 Javascript 의 내장 함수인 fetch 를 이용해서 통신을 요청해 보겠습니다!
- 이 부분의 코드에서 비동기/동기 통신, CallBack 등등 배우실 부분이 많습니다!
- 일단, 해당 코드가 어떤 식으로 구동이 되는지를 보시고 천천히 배워보겠습니다!

frontend/index.html

```html
<!DOCTYPE html>
<html lang="en">

<body>
  <h1 class="header">Hello, Protocol</h1>
  <button onclick="fetchData()">백엔드 통신 경험하기</button>
</body>

<script>
  const headerEl = document.querySelector('.header');

  const fetchData = () => {
    fetch('http://localhost:4000')
      .then((res) => {
        return res.json();
      })
      .then((data) => {
        headerEl.innerHTML = data;
      })
  }
</script>
</html>
```

Hello, Protocol

백엔드 통신 경험하기

안녕하세요. 여기는 백엔드 입니다!

백엔드 통신 경험하기

여러분은 방금 처음으로 백엔드 서버와 통신을 하셨어요!


그런데
말입니다
SBS

저건 어찌 저렇게
작동하는지
잘 모르겠는데?

JSON?

Callback?

JSON? → JavaScript Object Notation

```json
{
  "string": "Hi",
  "number": 2.5,
  "boolean": true,
  "null": null,
  "object": { "name": "Kyle", "age": 24 },
  "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],
  "arrayOfObjects": [
    { "name": "Jerry", "age": 28 },
    { "name": "Sally", "age": 26 }
  ]
}

```

JSON? → JavaScript Object Notation

- JS의 객체 처럼 속성과 해당 속성(Key)에 대한 값(Value)을 쌍으로 가지는 객체형 자료형 입니다!

```
const tetz = {
  name: '이효석',
  age: 38,
  isOld: true,
};
```

JS의 객체

```
{
  "name": "이효석",
  "age": "38",
  "isOld": "true"
}
```

JSON

JSON? → JavaScript Object Notation

```
const tetz = {
  name: '이효석',
  age: 38,
  isOld: true,
};
```

JS의 객체

```
{
  "name": "이효석",
  "age": "38",
  "isOld": "true"
}
```

JSON

- 대신 JSON 은 서버 통신을 위한 개방형 표준 포맷이므로 서로 다른 시스템에도 쉽게 전송이 가능해야 하며, 데이터를 최대한 압축이 가능해야 함
- 따라서, 사람 눈에는 객체 처럼 보이지만 데이터로 전달 될 때는 하나의 긴 문자열 처럼 전달이 됩니다!

```
{"name": "이효석","age": "38","isOld": "true"}
```

JSON? → JavaScript Object Notation

- 따라서 서버(백엔드)에서 다른 곳(ex. 프론트엔드)으로 데이터를 전송 할 때에는 해당 데이터를 JSON 형태로 변환해서 전달 해야 합니다!
- 그리고 해당 JSON 데이터를 받은 시스템에서는 JSON 데이터를 해당 시스템에 맞는 자료형으로 다시 변경해서 쓰면 해결이 되는 것이죠!

JSP

node JS

python™

{JSON}

JavaScript Object Notation

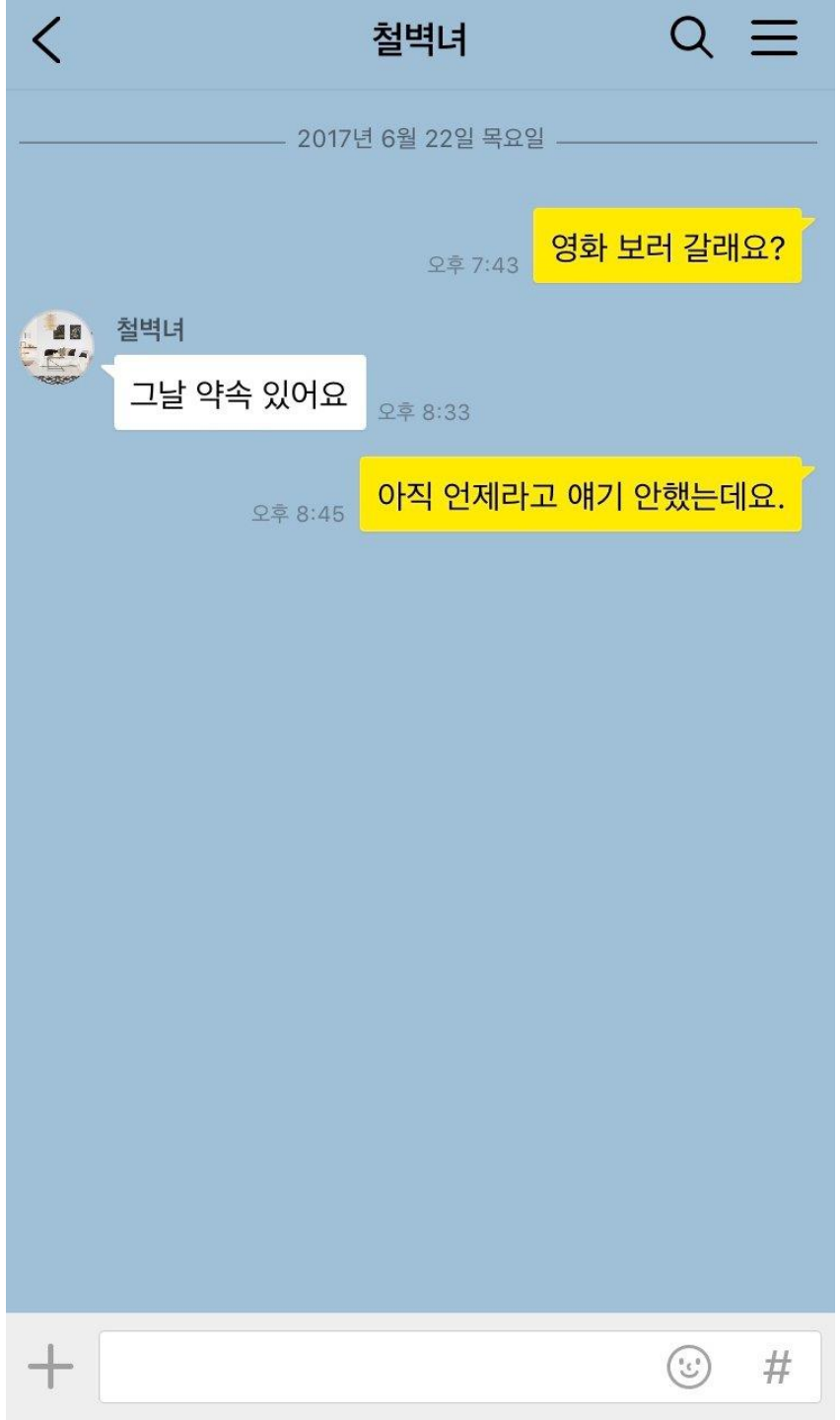
JSON 형태로 전달을 안하면!?

```
app.use('/', (req, res) => {
  const str = '안녕하세요. 여기는 백엔드 입니다!';
  res.send(str);
});
```

backend/server.js

- 이렇게 문자열을 바로 전달하면!?

```
▶Uncaught (in promise) SyntaxError: Unexpected token      VM130:1
'안', "안녕하세요. 여기는 백엔드 입니다!" is not valid JSON
```


철벽녀
2017년 6월 22일 목요일
영화 보러 갈래요?
오후 7:43
철벽녀
그날 약속 있어요
오후 8:33
아직 언제라고 얘기 안했는데요.
오후 8:45


두번봐도 그하지

Callback!?

처음 보는 형태의 함수가 등장!

frontend/index.html

```
const fetchData = () => {
    fetch('http://localhost:4000')
      .then((res) => {
        return res.json();
      })
      .then((data) => {
        headerEl.innerHTML = data;
      })
  }
```

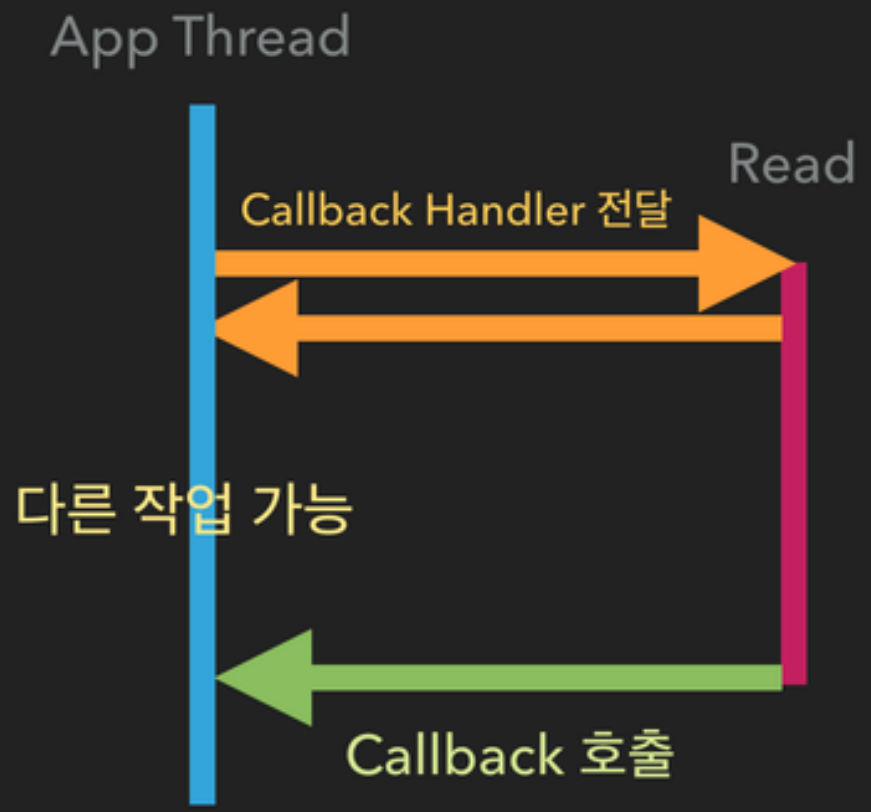
동기 / 비동기 프로그래밍

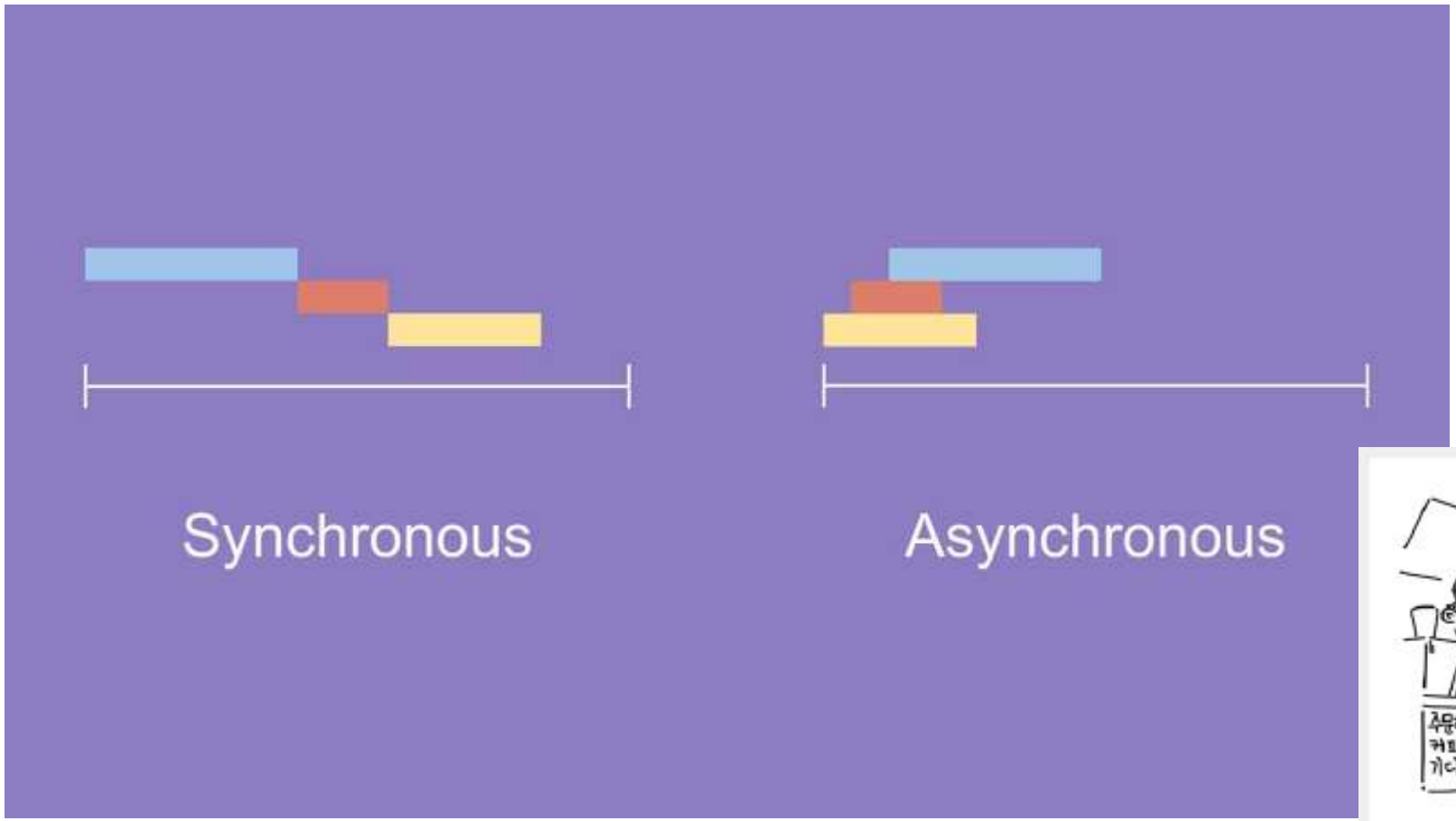
- Node.js 의 장점이었던 비동기적 프로그래밍이 기억 나시나요?
- 동기
  - 하나의 요청이 완료가 되면 그 후에 다음 요청을 실행
- 비동기
  - 하나의 요청이 완료가 되기 전에 다음 요청을 실행

SYNCHRONOUS VS ASYNCHRONOUS

Sync 작업 완료 후 리턴

Async 바로 리턴 + Callback




Synchronous
Asynchronous


주문한 후에,
커피가 나올때까지
기다려 주세요!
(a) 동기
주문한 후에,
진동벨이 울리면
커피를 가지로 오세요!
(b) 비동기

비동기 프로그래밍

- 따라서 Node.js 는 컴퓨터의 각각의 코어에 다른 일을 시킬 수 있습니다!
- 즉, 동시에 여러가지 일을 한꺼번에 수행할 수 있습니다!
- 따라서 Node.js 는 가볍고 빠른 서버를 구성할 수 있죠!
- 하지만 이러한 비동기적 특성이 항상 좋은 것은 아닙니다!

비동기 프로그래밍

- 서버(백엔드) 통신을 할 때에는 동기적(절차적)으로 문제를 해결해야합니다.
- 예를 들어서 로그인 요청을 보내고 서버에서 로그인 결과에 대한 결과 값을 받은 다음에 로그인 처리를 해야겠죠?
- 하지만 Node.js 는 비동기적 특성을 가지므로 + 아무리 인터넷이 빨라 졌어도 node.js 의 처리 속도 보다는 느리기 때문에, 기존과 동일하게 코드를 작성하면 로그인 결과를 받기도 전에 node.js 는 로그인 처리를 하려고 할 것이고 이는 문제가 될 것입니다!

구원자, Callback!

- 위와 같은 비동기 프로그래밍의 문제를 해결하기 위해서 JS 에서 지원하는 방식이 바로 Callback 방식 입니다~!
- Callback 함수는 특정 함수의 작업이 종료 되어야만 다른 함수를 호출 하기 때문에 사용자가 원하는대로 절차적 프로그래밍이 가능해 집니다!
- 그럼 일단 간단한 코드로 확인해 보시죠!

```
function buySync(item, price, quantity) {
  console.log(`${item} 상품을 ${quantity} 개 골라서 점원에게 주었습니다.`);
  console.log('계산이 필요합니다.');
  const total = price * quantity;
  return total;
}

function pay(tot) {
  console.log(`${tot} 원을 지불하였습니다.`);
}

const totalMoney = buySync('포켓몬빵', 1000, 5);

pay(totalMoney);
```

backend/callback.js

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT_4th/수
$ node callback.js
포켓몬빵 상품을 5 개 골라서 점원에게 주었습니다.
계산이 필요합니다.
5000 원을 지불하였습니다.
```

비동기 프로그래밍

- 아무리 JS가 비동기적 특성을 가지고 있다고 해도, 시간이 걸리는 요청이 들어 왔을 때에만 해당 일을 하면서 뒤의 코드를 실행합니다!
- 즉, 코드가 작성 된 순서는 지키면서 실행을 하기 때문에 전혀 문제가 없는 코드입니다!

여기서 통신 시간의 개념을 추가!

- 여기서 setTimeout 함수를 사용해서 계산원이 1초에 걸려서 계산을 한다고 만들어 봅시다!
- setTimeout(function() { 실행할 함수}, delay);

```
function buySync(item, price, quantity) {
  console.log(`${item} 상품을 ${quantity} 개 골라서 점원에게 주었습니다.`);
  setTimeout(() => {
    console.log('계산이 필요합니다.');
    const total = price * quantity;
    return total;
  }, 1000);
}

function pay(tot) {
  console.log(`${tot} 원을 지불하였습니다.`);
}

const totalMoney = buySync('포켓몬빵', 1000, 5);

pay(totalMoney);
```

backend/callback.js

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT_4th/
$ node callback.js
포켓몬빵 상품을 5 개 골라서 점원에게 주었습니다.
undefined 원을 지불하였습니다.
계산이 필요합니다.
```

- Node.js 는 이미 뒤의 pay() 함수를 실행 시키기 때문에 pay() 함수는 전체 가격을 전달 받지 못하고 undefined 를 출력하게 됩니다!
- 이때 필요한 것이 바로 Callback 이죠!

Callback 사용법!

- Callback 은 함수에 마지막 인자로 해당 함수가 종료 된 시점에서 실행하고 싶은 함수를 넣어 주면 됩니다!

```
function buySync(item, price, quantity, callback) {
  console.log(`${item} 상품을 ${quantity} 개 골라서 점원에게 주었습니다.`);
  setTimeout(() => {
    console.log('계산이 필요합니다.');
    const total = price * quantity;
    callback(total);
  }, 1000);
}

function pay(tot) {
  console.log(`${tot} 원을 지불하였습니다.`);
}

buySync('포켓몬빵', 1000, 5, pay);
```

backend/callback.js

Callback 사용법!

- 이제 계산원이 계산을 하는 1초를 기다렸다가 결과가 출력 되는 것을 볼 수 있습니다!
- Node.js 의 비동기적 특성이 아닌 동기적(절차적으로 구현이 된 것이죠!)

실습, Callback 함수 구현!

- knock.js 라는 파일을 하나 만들어 주세요!
- 여러분은 callback 함수를 포함하는 knockDoor()와 callName()는 함수를 작성 하시면 됩니다.
- Knock.js 코드 제일 하단에서 아래의 코드가 실행이 되면, 다음장의 스펙을 만족하는 기능이 구현 되면 됩니다!

```
knockDoor(3, '영식', callName);
```

실습, Callback 함수 구현!

- 실행이 되면 노크를 하고 기다립니다! 라는 문구를 띄우기
- 처음 인자로 입력 된 숫자의 초 만큼 기다렸다가 입력 된 '이름'이 문을 열고 나왔습니다를 callback 을 이용하셔서 구현 하세요!

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT_4
$ node knock.js
노크를 하고 기다립니다!
영식 이가 3초 만에 문을 열고 나왔습니다..
```

Callback

+ 익명함수

Callback 은 즉시 만들어서 전달 가능!

- Callback 함수를 꼭 만들어서 전달할 필요는 없으며, 즉시 만들어서 전달이 가능합니다!
- 이럴 때는 익명 함수를 사용하죠!
- 그리고 상당히 많이 사용이 됩니다!!

```
function buySync(item, price, quantity, callback) {
  console.log(`${item} 상품을 ${quantity} 개 골라서 점원에게 주었습니다.`);
  setTimeout(() => {
    console.log('계산이 필요합니다.');
    const total = price * quantity;
    callback(total);
  }, 1000);
}

// function pay(tot) {
//   console.log(`${tot} 원을 지불하였습니다.`);
// }

buySync('포켓몬빵', 1000, 5, function (total) {
  console.log(`${total} 원을 지불하였습니다.`);
});
```


런닝맨
SBS
나 분명히 어딘가에 생긴
사람을 본 적이 있는데...

```
function buySync(item, price, quantity, callback) {
  console.log(`${item} 상품을 ${quantity} 개 골라서 점원에게 주었습니다.`);
  setTimeout(() => {
    console.log('계산이 필요합니다.');
    const total = price * quantity;
    callback(total);
  }, 1000);
}

// function pay(tot) {
//   console.log(`${tot} 원을 지불하였습니다.`);
// }

buySync('포켓몬빵', 1000, 5, function (total) {
  console.log(`${total} 원을 지불하였습니다.`);
});
```

```ts
// @ts-check

const http = require('http');

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.end('Hello');
});

const PORT = 4000;
server.listen(PORT, () => {
  console.log(`The server is listening at port: ${PORT}`);
});
```

Callback

Hell?

콜백 지옥? 그게 뭐죠?

- 지금까지 연습한 절차적 프로그래밍은 절차가 딱 하나 였습니다!
- 즉, 하나의 수행이 끝나면 그 다음에 원하는 함수를 callback 으로 실행 시키면 끝이였죠!
- 그런데 이러한 절차가 한 4~5번이 반복이 된다면 어떻게 될까요?
- 그럼 callback 이 다시 callback 을 부르는 것이 4번이나 반복이 되어야 합니다!

callbackhell.js

```
var floppy = require('floppy');

floppy.load('disk1', function (data1) {
    floppy.prompt('Please insert disk 2', function() {
        floppy.load('disk2', function (data2) {
            floppy.prompt('Please insert disk 3', function() {
                floppy.load('disk3', function (data3) {
                    floppy.prompt('Please insert disk 4', function() {
                        floppy.load('disk4', function (data4) {
                            floppy.prompt('Please insert disk 5', function() {
                                floppy.load('disk5', function (data5) {
                                    floppy.prompt('Please insert disk 6', function() {
                                        floppy.load('disk6', function (data6) {
                                            //if node.js would have existed in 1995
                                        });
                                    });
                                });
                            });
                        });
                    });
                });
            });
        });
    });
});
```

```
foo(() => {
    bar(() => {
        baz(() => {
            qux(() => {
                quux(() => {
                    quuz(() => {
                        corge(() => {
                            grault(() => {
                                run();
                            }).bind(this);
                        }).bind(this);
                    }).bind(this);
                }).bind(this);
            }).bind(this);
        }).bind(this);
    }).bind(this);
}).bind(this);
```

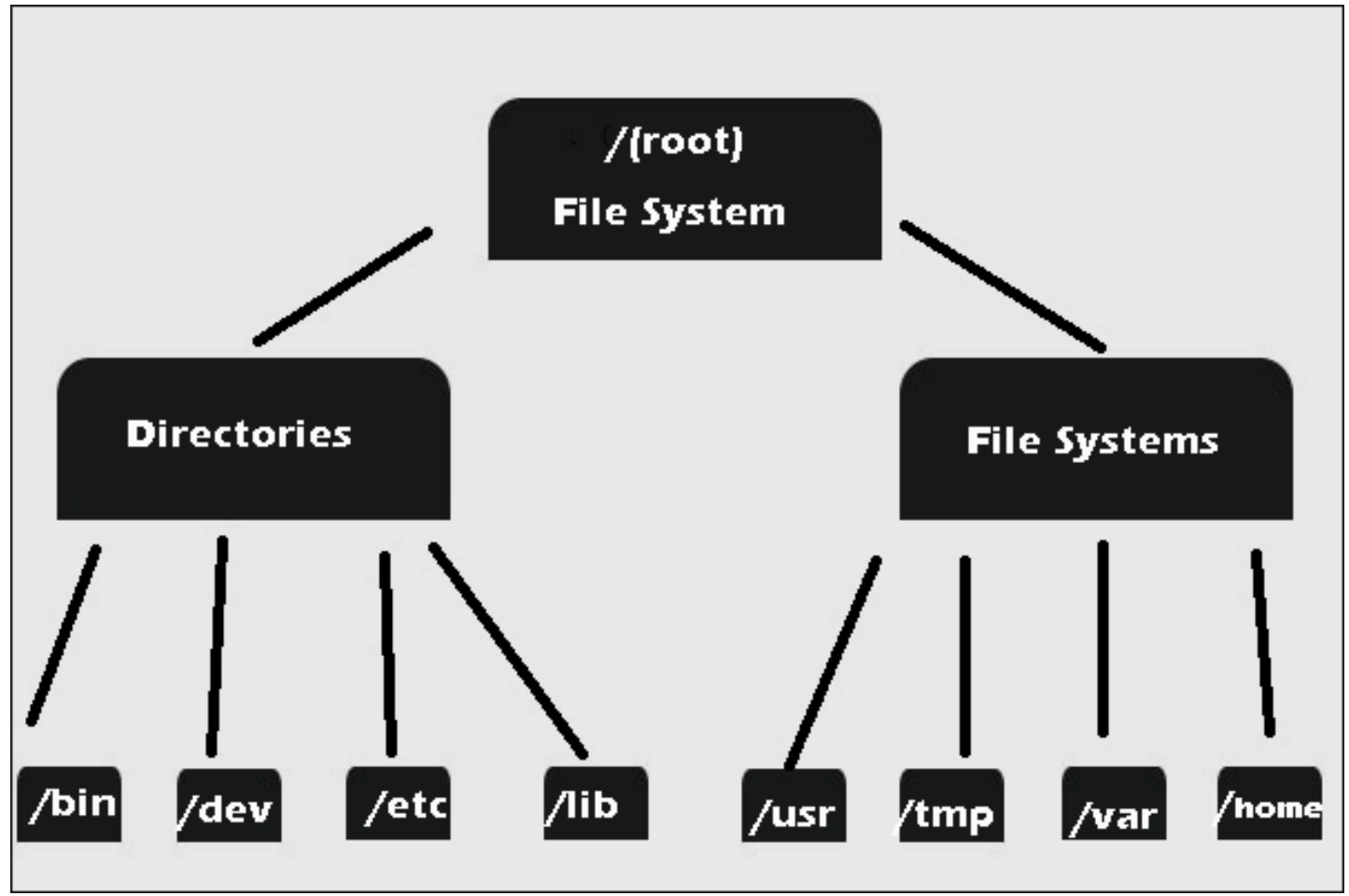
이런 사태가

발생합니다!

Callback Hell

체험 해보기!

File-System on JS


/(root)
File System
Directories
File Systems
/bin
/dev
/etc
/lib
/usr
/tmp
/var
/home

File-system 은 Node 의 기본 모듈 입니다!

```
const fs = require('fs');
```

- 파일을 읽을 때는 readFile 메소드를 사용합니다.
- 파일을 읽는 것도 시간이 필요한 작업이므로 일종의 서버 통신과 비슷합니다

  → 따라서, 기본 적으로 callback 을 사용하도록 되어있습니다.
- fs.readFile('파일위치', '유니코드포맷', callback(err, data) {})
- err 은 파일 읽기가 잘 안되었을 때, Error 코드를 반환 합니다
- data 는 파일 읽기가 잘 되엇을 때, 읽은 data 를 반환합니다.

파일 읽기!

```
const fs = require('fs');

fs.readFile('readme.txt', 'utf-8', function (err, data) {
  if (err) {
    console.log(err);
  } else {
    console.log(data);
  }
});
```

backend/file.js

```
const fs = require('fs');

fs.readFile('readme.txt', 'utf-8', (err, data) => {
  if (err) {
    console.log(err);
  } else {
    console.log(data);
  }
});
```

backend/file.js

파일 쓰기!

```
const fs = require('fs');

const str = '파일 쓰기가 정상적으로 되는지 테스트 합니다.';

fs.writeFile('readme.txt', str, 'utf-8', (err) => {
  if (err) {
    console.log(err);
  } else {
    console.log('writefile succeed');
  }
});
```

backend/file.js

File-system 과 비동기 프로그래밍

File-system 과 비동기 프로그래밍

- 파일 시스템을 이용해서 비동기 프로그래밍 코드를 짜봅시다!
- JS의 특성으로 인해 각각 readFile 메소드를 동시에 비동기적으로 실행 시켜 봅시다!
- 동시에 readme.txt 파일을 읽어서 console.log 로 출력하는 코드를 작성해 봅시다!

backend/file.js

```
const fs = require('fs');

fs.readFile('./readme.txt', (err, data) => {
  if (err) {
    throw err;
  }
  console.log('1번', data.toString());
});
fs.readFile('./readme.txt', (err, data) => {
  if (err) {
    throw err;
  }
  console.log('2번', data.toString());
});
fs.readFile('./readme.txt', (err, data) => {
  if (err) {
    throw err;
  }
  console.log('3번', data.toString());
});
fs.readFile('./readme.txt', (err, data) => {
  if (err) {
    throw err;
  }
  console.log('4번', data.toString());
});
```

File-system 과 비동기 프로그래밍

- 과연 결과는 어떻게 될까요!?

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/정규 수업/25/node_set
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다
```

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/정규 수업/25/node_set
$ node js/file.js
1번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다
2번 readme 텍스트 입니다
```

File-system 과 비동기 프로그래밍

- File 을 읽는 것은 JS가 각각의 Thread 에 실어서 처리하기 때문에 각각의 Thread 상황에 따라 file 읽는 속도가 다르게 됩니다.
- 따라서, 꼭 1, 2, 3, 4 로 실행 된다는 보장이 없죠!
- 그럼 1, 2, 3, 4 순서로 실행 시키려면 어찌하면 될까요?

Callback

Hell?!

Callback 지옥으로 구현하기!

- 앞서 배운 것 처럼 fs 는 callback 을 기본적으로 지원하므로 callback 이 호출 되면 바로 다음 fileRead 메소드를 실행 시키고, 다시 callback 에 fileRead 를 실행 시키는 콜백 지옥 형태로 코드를 구성하여 봅시다!

```js
const fs = require('fs');

fs.readFile('./readme.txt', (err, data) => {
  if (err) {
    throw err;
  }
  console.log('1번', data.toString());
  fs.readFile('./readme.txt', (err, data) => {
    if (err) {
      throw err;
    }
    console.log('2번', data.toString());
    fs.readFile('./readme.txt', (err, data) => {
      if (err) {
        throw err;
      }
      console.log('3번', data.toString());
      fs.readFile('./readme.txt', (err, data) => {
        if (err) {
          throw err;
        }
        console.log('4번', data.toString());
      });
    });
  });
});
```

backend/cbHell.js

```
lhs@DESKTOP-86MUCGC MINGW64 ~
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다

lhs@DESKTOP-86MUCGC MINGW64 ~
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다

lhs@DESKTOP-86MUCGC MINGW64 ~
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다
```

```
foo(() => {
    bar(() => {
        baz(() => {
            qux(() => {
                quux(() => {
                    quuz(() => {
                        corge(() => {
                            grault(() => {
                                run();
                            }).bind(this);
                        }).bind(this);
                    }).bind(this);
                }).bind(this);
            }).bind(this);
        }).bind(this);
    }).bind(this);
}).bind(this);
```

Callback 지옥으로 구현하기!

- 비록 코드는 좀 구리지만!? 의도했던 것 처럼 1, 2, 3, 4 가 순서대로 실행이 됩니다!
- 그럼, 이 callback 지옥을 어찌 탈출 할 수 있을까요!?

이상
콜백지옥 탈출 방법에
대해 알아 봤습니다!

Promise!

Promise!

- Promise 는 일종의 클래스 입니다! 따라서 new 로 사용하죠!

```
const promise = new Promise(function(resolve, reject) {});
```

- resolve, reject 라는 2개의 매개변수를 콜백 함수로 받아서 사용합니다.
- promise 가 할당 되면 이 promise 는 resolve 또는 reject 함수가 callback 될 때 까지 무한 대기 합니다.
- Resolve 는 promise 가 정상적으로 이행 되었을 경우 사용하며, reject 는 반대의 경우에 사용합니다.

Promise!

- resolve 는 추후에 then 으로 받으며, reject 는 catch 로 받습니다!
- resolve, reject 콜백 함수의 경우는 데이터를 매개변수로 보낼 수 있습니다.
- Resolve, reject 가 사용되지 않으면 promise 는 해당 콜백이 나올 때 까지 pending 상태가 되어 기다립니다!

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/KDT/정규 수업/25/node_set
$ node js/promise.js
Promise { <pending> }
```


Promise

```
const promise = new Promise((resolve, reject) => {
  function buySync(item, price, quantity) {
    console.log(`${item} 상품을 ${quantity} 개 골라서 점원에게 주었습니다.`);
    setTimeout(() => {
      console.log('계산이 필요합니다.');
      const total = price * quantity;
      resolve(total);
    }, 1000);
  }
  buySync('포켓몬빵', 1000, 5);
});

promise.then(function (total) {
  console.log(`${total} 원을 지불하였습니다.`);
});
```

backend/promise.js

```
const promise = new Promise(function (resolve, reject) {
  const tetz = 'old';
  if (tetz === 'old') {
    setTimeout(() => {
      resolve('tetz is old');
    }, 3000);
  } else {
    reject('tetz is getting old');
  }
});

promise
  .then(function (data) {
    console.log(data);
  })
  .catch(function (data) {
    console.log(data);
  });
```

backend/promise2.js

콜백 지옥을 Promise 로 변경 하기

- fs.promises 를 사용해서 콜백 지옥을 promise 코드로 변경해 봅시다
- fs.promises 해당 메소드의 판단 여부를 스스로 판단 후 → resolve, reject 를 알아서 알아서 반환 합니다! → 고로 편리합니다!

```
const fs = require('fs').promises;

fs.readFile('./readme.txt')
  .then((data) => {
    console.log('1번', data.toString());
    return fs.readFile('./readme.txt');
  })
  .then((data) => {
    console.log('2번', data.toString());
    return fs.readFile('./readme.txt');
  })
  .then((data) => {
    console.log('3번', data.toString());
    return fs.readFile('./readme.txt');
  })
  .then((data) => {
    console.log('4번', data.toString());
  })
  .catch((err) => {
    throw err;
  });
```

backend/promiseHell.js

```
lhs@DESKTOP-86MUCGC MINGW64 ~/
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다

lhs@DESKTOP-86MUCGC MINGW64 ~/
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다

lhs@DESKTOP-86MUCGC MINGW64 ~/
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다
```

Async / Await

최신 기술인 Async, Await 도 적용!

- Function 앞에 async 를 붙이면 해당 함수는 항상 Promise 를 반환
- 즉, async 가 붙은 함수에서 return 을 쓰면 아래와 동일한 역할을 합니다.

```
async function f1() {
  return 1;
}

async function f2() {
  return Promise.resolve(1);
}
```

- Async 가 붙은 함수 내부에는 Await 키워드 사용이 가능!
- Await 은 promise 가 결과(resolve, reject)를 가져다 줄 때 까지 기다립니다.

Async, Await 도 적용!

- 단, async 는 함수를 정의하는 상황에서 쓰이므로 함수 정의 후, 해당 함수를 외부에서 한번 사용해 줘야합니다!

Await

```
const promise = new Promise((resolve, reject) => {
  function buySync(item, price, quantity) {
    console.log(`${item} 상품을 ${quantity} 개 골라서 점원에게 주었습니다.`);
    setTimeout(() => {
      console.log('계산이 필요합니다.');
      const total = price * quantity;
      return resolve(total);
    }, 1000);
  }
  buySync('포켓몬빵', 1000, 5);
});

function pay(total) {
  console.log(`${total} 원을 지불하였습니다.`);
}

async function asyncFunc() {
  const total = await promise;
  pay(total);
}

asyncFunc();
```

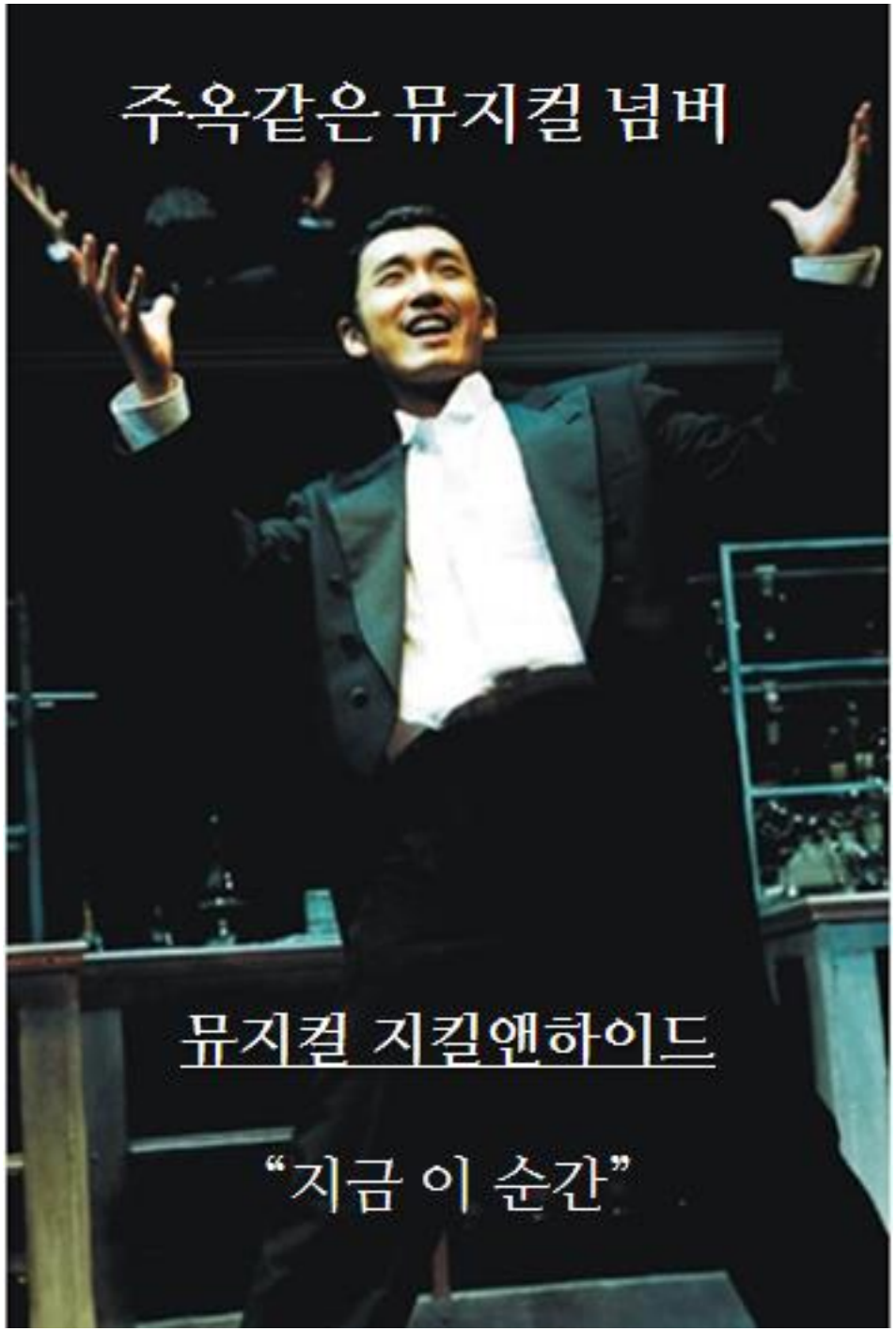
backend/async.js

Async / Await 로 콜백지옥 탈출!

```js
const fs = require('fs').promises;

async function main() {
  let data = await fs.readFile('./readme.txt');
  console.log('1번', data.toString());
  data = await fs.readFile('./readme.txt');
  console.log('2번', data.toString());
  data = await fs.readFile('./readme.txt');
  console.log('3번', data.toString());
  data = await fs.readFile('./readme.txt');
  console.log('4번', data.toString());
}

main();
```

```
lhs@DESKTOP-86MUCGC MINGW64
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다

lhs@DESKTOP-86MUCGC MINGW64
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다

lhs@DESKTOP-86MUCGC MINGW64
$ node js/file.js
1번 readme 텍스트 입니다
2번 readme 텍스트 입니다
3번 readme 텍스트 입니다
4번 readme 텍스트 입니다
```

node.js
+
express



Express

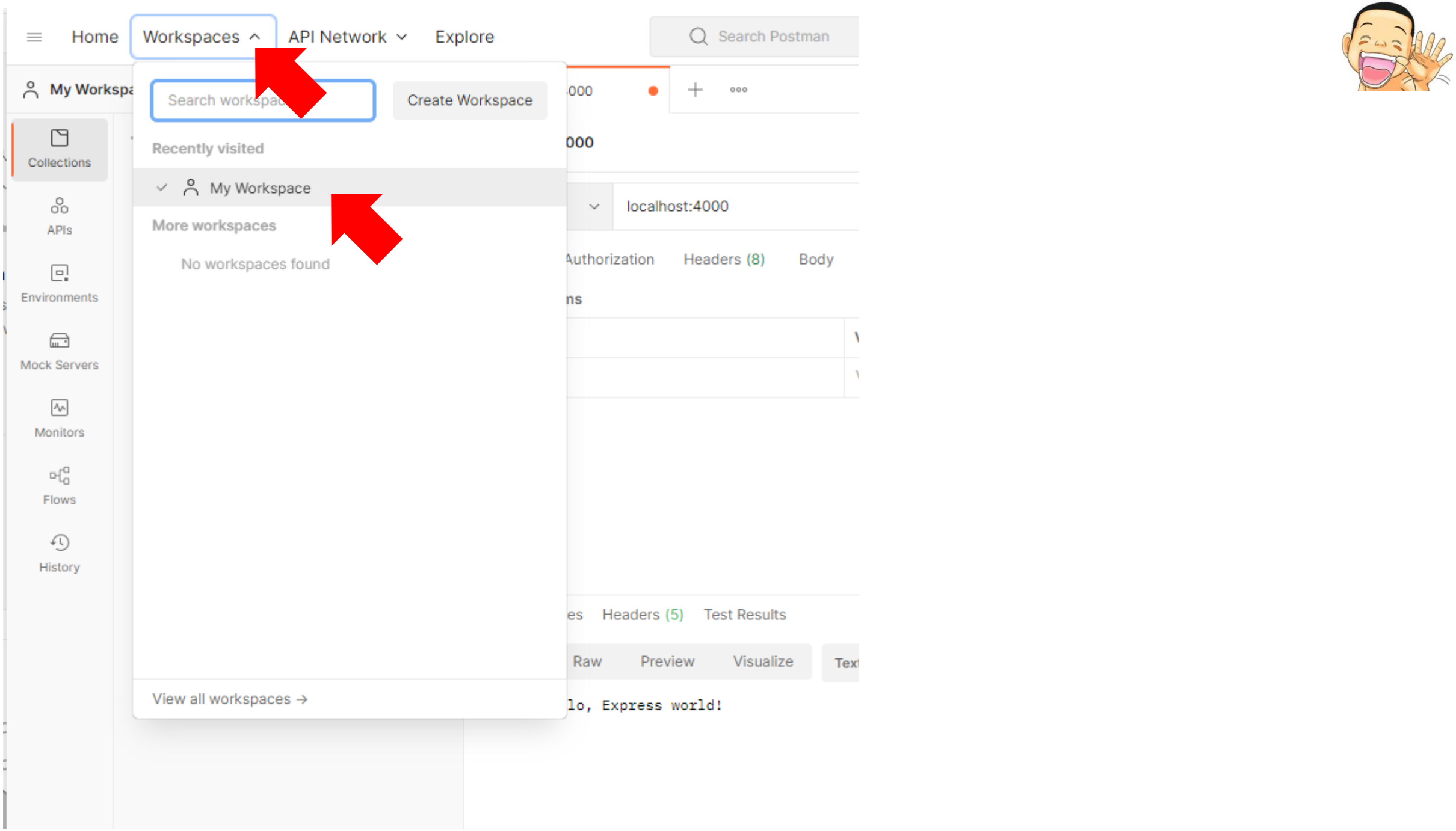
http://expressjs.com/

Postman

https://www.postman.com/downloads/

Postman

- 서버 테스트를 하는 프로그램 중, 제일 유명한 프로그램 입니다!
- 백엔드 서버 테스트는 요걸로 테스트를 할 겁니다!
- 다운로드 해주시고, 회원 가입 or 구글 로그인 하면 사용이 가능합니다!


Home
Workspaces
API Network
Explore
Search Postman
My Workspa
Search workspace
Create Workspace
Recently visited
My Workspace
More workspaces
No workspaces found
View all workspaces →
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
localhost:4000
Authorization
Headers (8)
Body
Headers (5)
Test Results
Raw
Preview
Visualize
lo, Express world!

New Collection / **localhost:4000/posts**

GET http://localhost:4000

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize HTML

```
1  "안녕하세요. 여기는 백엔드 입니다!"
```

Express Middleware

Middleware 가 뭐죠?

- 사전적인 의미로는 서로 다른 어플리케이션(프로그램)이 서로 통신을 하는 데 사용되는 소프트웨어를 뜻합니다!
- 즉, 양쪽 어플리케이션 가운데에서 역할을 하는 소프트웨어죠!
- Express 는 백엔드 서비스 구성을 위한 다양한 상황에 맞는 여러가지 서비스를 미들웨어 형태로 제공을 합니다.
- 즉, Express 에서의 미들웨어는 서버에 들어온 요청이 들어와서 응답으로 나갈 때 까지 거치는 모든 함수 또는 기능을 의미한다고 생각하시면 됩니다!

Middleware 사용하기!

- Express 에서는 app.use 또는 app.method 함수를 이용해서 미들웨어를 사용합니다.
- App.use(‘요청 주소‘, (req, res, next) => {}); 의 형태로 사용이 가능합니다!

```ts
// @ts-check

const express = require('express');

const app = express();

const PORT = 4000;

app.use('/', (req, res, next) => {
  res.send('Hello, express!');
});

app.listen(PORT, () => {
  console.log(`The express server is running at port: ${PORT}`);
});
```

```
lhs@DESKTOP-86MUCGC MINGW64 ~/Desktop/업무/K
$ http localhost:4000
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 15
Content-Type: text/html; charset=utf-8
Date: Thu, 01 Sep 2022 23:21:30 GMT
ETag: W/"f-oajqb6Lvrly+9VVWAZjgZh0HtYM"
Keep-Alive: timeout=5
X-Powered-By: Express

Hello, express!
```

Middleware 사용하기, Next(?)

- Next 는 callback 함수로서 해당 함수가 호출 되면 현재 미들웨어를 종료하고 다음 미들웨어를 실행 시킵니다!

```
const express = require('express');

const app = express();
const PORT = 4000;

app.use('/', (req, res, next) => {
  console.log('Middleware 1');
  next();
});

app.use((req, res, next) => {
  console.log('Middleware 2');
  res.send('res.send!');
  next();
});

app.use((req, res, next) => {
  console.log('Middleware 3');
});

app.listen(PORT, () => {
  console.log(`The express server is running at port: ${PORT}`);
});
```

```
The express server is running at port: 4000
Middleware 1
Middleware 2
Middleware 3
```

Middleware 사용하기, Next(?)

```
const express = require('express');

const app = express();
const PORT = 4000;

app.use((req, res, next) => {
  console.log('Middleware 2');
  res.send('res.send!');
});

app.use('/', (req, res, next) => {
  console.log('Middleware 1');
  next();
});

app.listen(PORT, () => {
  console.log(`The express server is running at port: ${PORT}`);
});
```

```
The express server is running at port: 4000
Middleware 2
```

Middleware 사용하기, Next(?)

- Next 를 상용해서 다음 미들웨어를 실행 시킬 때, 이전 미들웨어에서 처리한 값을 전달 하고 싶을 때가 생깁니다!
- 물론, global 영역에서 변수를 하나 만들어서 전달해도 되지만 global 변수는 보통 사용을 안하는 것이 좋습니다. → 변수가 언제 어디에서 변경이 되어 문제를 발생 시킬지 예측이 어렵기 때문이죠.
- 이럴 때에는 req 객체에 필드를 추가해서 전달하는 방법이 많이 사용 됩니다!
- 다만 이 방법도 정석적인 방법은 아니므로 급할 때만 사용하세요!

```
const express = require('express');

const app = express();
const PORT = 4000;

app.use('/', (req, res, next) => {
  console.log('Middleware 1');
  req.reqTime = new Date();
  next();
});

app.use((req, res, next) => {
  console.log('Middleware 2');
  res.send(`Requsted at ${req.reqTime}`);
});

app.listen(PORT, () => {
  console.log(`The express server is running at port: ${PORT}`);
});
```

Promise

- 당연히 서버 통신이므로 Promise 사용이 가능합니다!
- Fs 을 이용해서 간단하게 Promise 상황을 만들어서 봅시다!
- 사용이 직관적인 Async / Await 을 이용해서 구현합니다!
- 이 참에 promise {<pending>} 상태도 체험해 봅시다!

```
const express = require('express');
const fs = require('fs');

const app = express();
const PORT = 4000;

app.use('/', async (req, res, next) => {
  console.log('Middleware 1');
  req.reqTime = new Date();
  req.fileContent = await fs.promises.readFile('package.json', 'utf-8');
  next();
});

app.use((req, res, next) => {
  console.log('Middleware 2');
  console.log(req.fileContent);
  res.send(`Requsted at ${req.reqTime}`);
});

app.listen(PORT, () => {
  console.log(`The express server is running at port: ${PORT}`);
});
```

Back-End

서버의 기본!

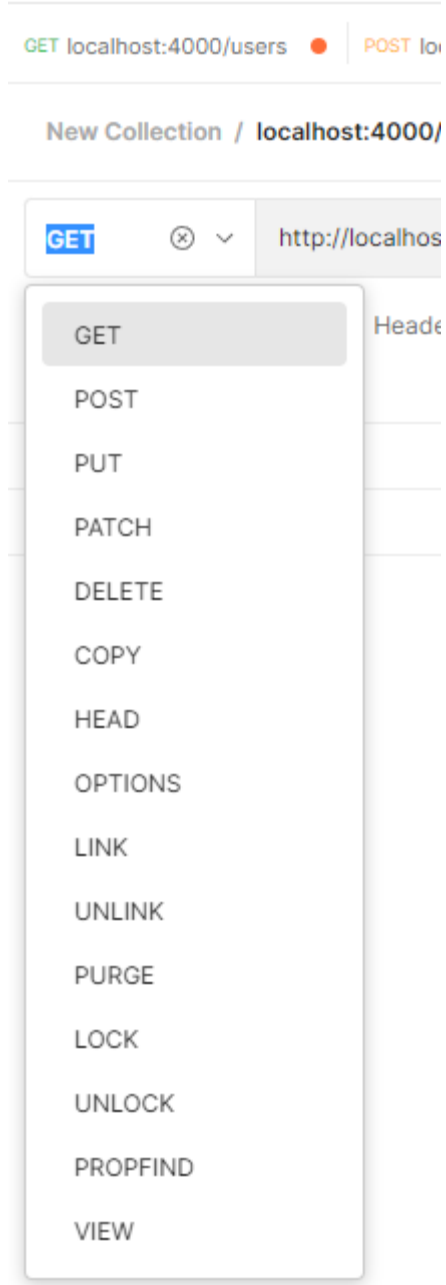
요청 메소드

Create ➡ POST

Read ➡ GET

Update ➡ PUT

Delete ➡ DELETE


GET localhost:4000/users
POST lo
New Collection / localhost:4000/
GET
http://localhos
Heade
GET
POST
PUT
PATCH
DELETE
COPY
HEAD
OPTIONS
LINK
UNLINK
PURGE
LOCK
UNLOCK
PROPFIND
VIEW

HTTP Status Codes

1XX
INFORMATIONAL

2XX
SUCCESS

3XX
REDIRECTION

4XX
CLIENT ERROR

5XX
SERVER ERROR

HTTP Status

- 100 번대 : 정보 / 리퀘스트를 받고 처리 중
- 200 번대 : 성공 / 리퀘스트를 정상 처리
- 300 번대 : 리디렉션 / 처리 완료를 위해서는 추가 동작 필요
- 400 번대 : 클라이언트 에러 / 클라이언트에서 요청을 잘못 보냄
- 500 번대 : 서버 에러 / 리퀘스트는 잘 들어 갔지만 서버에서 처리를 못함

HTTP Status Codes

Level 200 (Success)

200 : OK

201 : Created

203 : Non-Authoritative Information

204 : No Content

Level 400

400 : Bad Request

401 : Unauthorized

403 : Forbidden

404 : Not Found

409 : Conflict

Level 500

500 : Internal Server Error

503 : Service Unavailable

501 : Not Implemented

504 : Gateway Timeout

599 : Network timeout

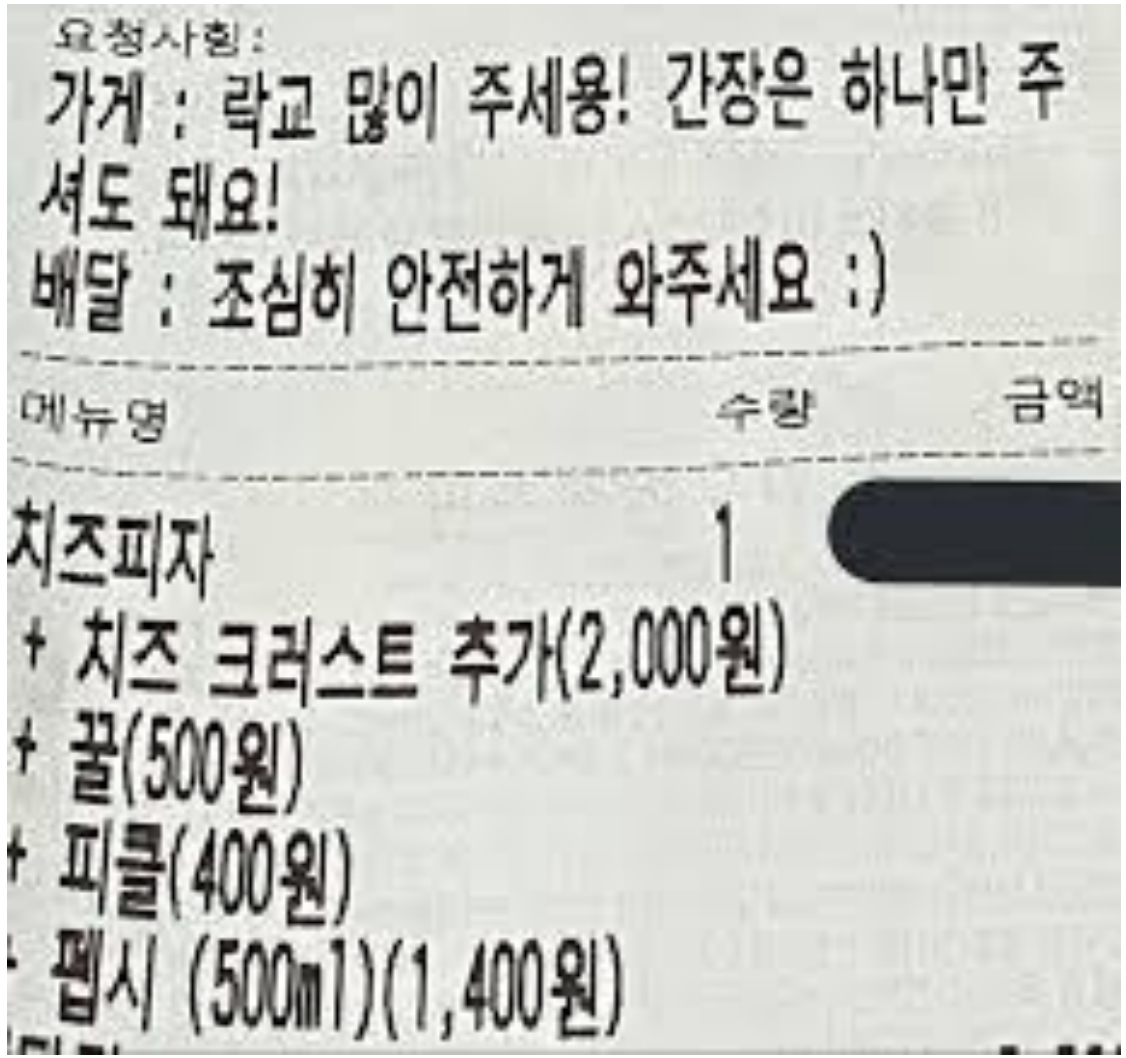
502 : Bad Gateway

백엔드로

데이터 요청!

백엔드 데이터 요청!

- 여러분 백엔드 설계를 할 때 모든 요청을 예상해서 api 를 만들어 놓을 수는 없겠죠?
- 그리고 회원 가입을 하는 과정이라면 회원 가입을 하는 유저의 ID와 Password는 프론트에서 받을 수 밖에 없겠죠?



요청사항:
가게 : 락교 많이 주세용! 간장은 하나만 주셔도 돼요!
배달 : 조심히 안전하게 와주세요 :)

메뉴명	수량	금액
치즈피자	1	[redacted]
+ 치즈 크러스트 추가(2,000원)		
+ 꿀(500원)		
+ 피클(400원)		
+ 펩시 (500ml)(1,400원)		

백엔드에서

데이터를 받는 방법

Req.Params

URL 로 Data 를 받는 방법, Params

- 보통 요청은 주소로 들어오죠? 그렇기 때문에 프론트에서 받아야 하는 정보는 주소 값에 담아서 보냅니다!

```
app.get('/:id', (req, res) => {
  console.log(req.params);
  res.end(`ID 번호가 ${req.params.id} 인 회원 정보`);
});
```

- 받을 url에서 :파라미터명 을 미리 정의해 두면 해당 내용은 req.params 에 담겨서 전달이 됩니다!

GET http://localhost:4000/1

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```
1  ID 번호가 1 인 회원 정보
```

URL 로 Data 를 받는 방법, Params

- 물론 여러 개를 받을 수도 있습니다.

```
app.get('/:id/:name/:gender', (req, res) => {
  console.log(req.params);
  res.send(req.params);
});
```

```
The express server is running at port: 4000
{ id: '1', name: 'tetz', gender: 'male' }
```

- Req.params 라는 객체에 담겨서 전달이 되어서 편리하게 사용이 가능합니다!

GET http://localhost:4000/1/tetz/male

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "id": "1",
    "name": "tetz",
    "gender": "male"
}
```

URL 로 Data 를 받는 방법, Params

- 여러 개를 받을 때, 어떤 값을 전달하는지 보내는 쪽에서 인지 시키기 위해서 이렇게 사용도 가능합니다.

```
app.get('/id/:id/name/:name/gender/:gender', (req, res) => {
  console.log(req.params);
  res.send(req.params);
});
```


```
{
   "id": "1",
   "name": "tetz",
   "gender": "male"
}
```

GET http://localhost:4000/id/1/name/tetz/gender/male

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE
Key	Value

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON


```
{
    "id": "1",
    "name": "tetz",
    "gender": "male"
}
```

Req.Query

URL 로 Data 를 받는 방법, Query

- Params 의 약점은 정의 된 형태로만 데이터를 받을 수 있습니다.
- 하지만 막 보내고 싶다면!? Query 를 쓰면 됩니다!
- Query 는 url에 ? 를 붙인 뒤, 필드명=값 으로 사용하면 됩니다!
- 여러 개를 보내고 싶으면, & 로 묶어서 여러 개를 보낼 수 있습니다!

```
app.get('/', (req, res) => {
  console.log(req.query);
  res.send(req.query);
});
```

GET | localhost:4000?test=test&id=3

Params ● | Authorization | Headers (8) | Body | Pre-request Script | Tests | Settings

KEY	VALUE
☑ test	test
☑ id	3

실습, req.params 와 req.query

- Parameter 방식과 Query 방식으로 email, password, name, gender 정보를 받아와서 출력하는 백엔드 코드 작성하기!
- 슬랙 과제 제출에 댓글로 제출해 주세요!
- 코드와, 각각 요청을 보내고 받은 postman의 스크린 샷을 보내주세요!

Express Routing

API Routing

- 프론트에서 백엔드로 요청을 보낼 때는 주소 값을 다르게 보내서 요청을 합니다!
- 따라서 백엔드에서는 각각 주소에 따라서 각기 다른 역할을 해주면 됩니다!
- 이렇게 주소에 따라서 각기 다른 역할을 하도록 나누는 방법을 Routing 이라고 합니다!
- Express 에서는 app.메소드명(); 의 형태로 요청 방식을 나누어 처리 할 수 있습니다!

메소드 별

Routing

```
const express = require('express');

const app = express();
const PORT = 4000;

app.get('/', (req, res) => {
  res.send('GET request');
});

app.post('/', (req, res) => {
  res.send('POST request');
});

app.put('/', (req, res) => {
  res.send('PUT request');
});

app.delete('/', (req, res) => {
  res.send('DELETE request');
});
```

Express Router()

Express Router

- 당연히 Express 에는 Routing 을 위한 미들웨어도 존재 합니다!
- Router() 를 사용하면 특정 url 요청에 대한 것들을 묶어서 처리가 가능합니다!

```
const express = require('express');

const app = express();
const userRouter = express.Router();
const PORT = 4000;

userRouter.get('/', (req, res) => {
  res.end('회원 목록');
});

userRouter.get('/:id', (req, res) => {
  res.end('특정 회원 정보');
});

userRouter.post('/', (req, res) => {
  res.end('회원 등록');
});

app.use('/users', userRouter);

app.use('/', (req, res) => {
  res.end('Hello, express world!');
});

app.listen(PORT, () => {
  console.log(`The express server is running at port: ${PORT}`);
});
```

간단한

데이터 처리

Express 를 이용해서 간단한 API 만들기

- 아래와 같은 API를 만들 예정입니다!
- GET Localhost:4000/users
  - 회원 목록을 보여주기
- GET Localhost:4000/users/:id
  - 특정 회원 정보 보여주기
- POST Localhost:4000/users?id=test&name=test
  - 회원 추가하기

회원 목록 보여주기 API

- GET Localhost:4000/users
- 위의 요청이 들어오면 회원 정보 Obj 를 그대로 전달하여 목록을 띄우겠습니다!

```
const USER = {
  1: {
    id: 'tetz',
    name: '이효석',
  },
};

userRouter.get('/', (req, res) => {
  res.send(USER);
});
```

특정 회원 정보 보여주기 API

- GET Localhost:4000/users/:id
- Id 정보를 params 로 받아와서 처리를 해줍니다.
- 단, 해당 id를 가지는 회원이 없으면 예외 처리를 해줍니다!
- 그리고 id 값 입력을 안하면 그 예외 처리는 Express가 해줍니다!

Express

```
userRouter.get('/:id', (req, res) => {
  const userData = USER[req.params.id];
  if (userData) {
    res.send(userData);
  } else {
    res.end('ID not found');
  }
});
```

회원 추가하기 API

- POST Localhost:4000/users?id=test&name=test
- Req.query 로 회원 가입할 정보를 받아와서 새로운 회원을 만들어 줍니다!
- 물론, id 또는 name 이 정상적으로 안들어온 경우는 예외 처리를 해야합니다!

```
userRouter.post('/', (req, res) => {
  if (req.query.id && req.query.name) {
    const newUser = {
      id: req.query.id,
      name: req.query.name,
    };

    USER[Object.keys(USER).length + 1] = newUser;

    res.send('회원 등록 완료');
  } else {
    res.end('Unexpected query');
  }
});
```

실습, 회원 수정 – 삭제 API 만들기!

- 회원 수정, 삭제 API를 만들어 주세요 ☺
- 회원 수정 테스트 URL
  - PUT Localhost:4000/users/1?id=test&name=test
- 회원 삭제 테스트 URL
  - DELETE Localhost:4000/users/1
- 슬랙 과제 제출에 댓글로 제출해 주세요!


수고하셨습니다!