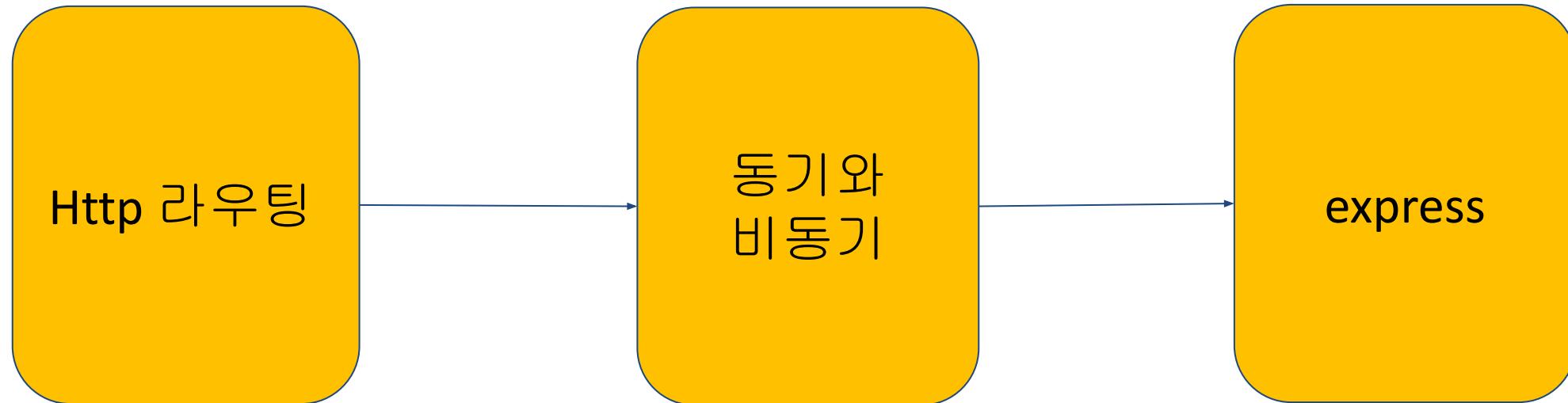




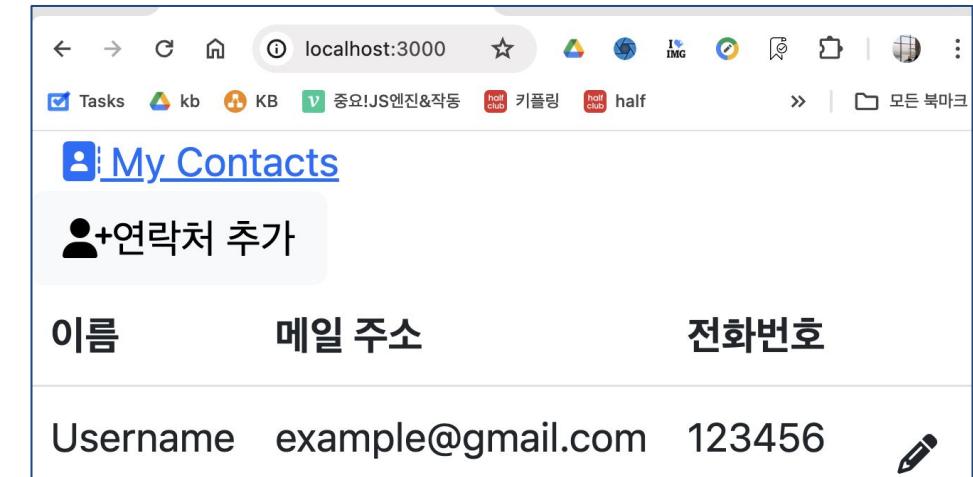
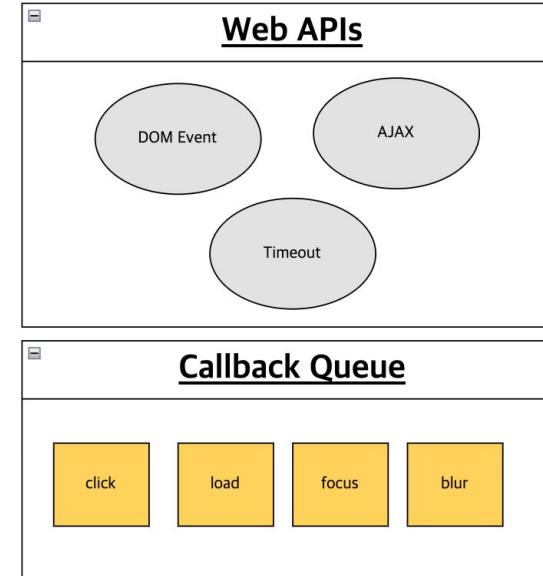
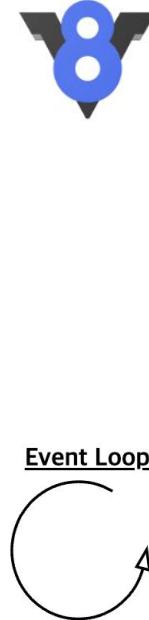
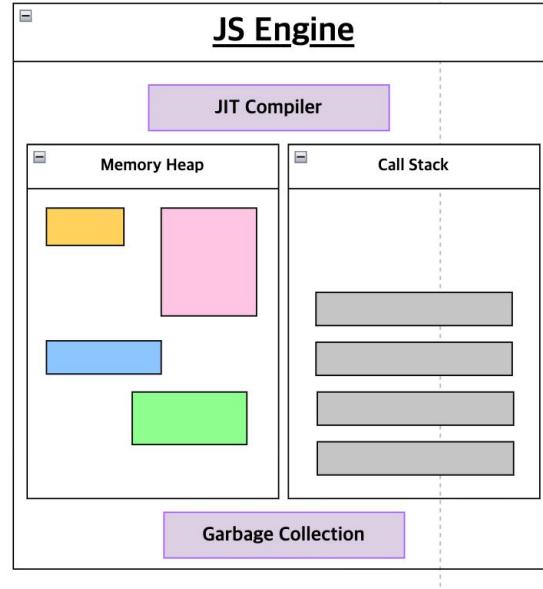
2024년 상반기 K-디지털 트레이닝

NodeJS express

[KB] IT's Your Life



오늘의 목표

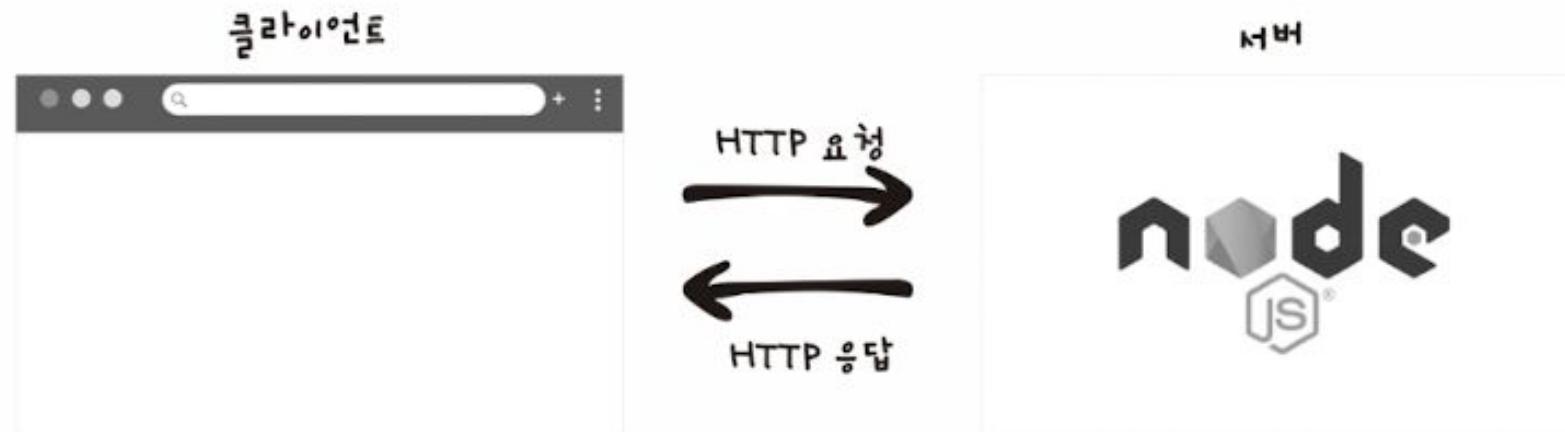


Http 라우팅

Http 라우팅

- **http 프로토콜**

- 클라이언트와 서버간 미리 약속한 규칙
- 각각 사용하는 프로그래밍 언어도, 동작하는 방식도 필요하므로 별도의 교환 규칙 필요
- header + body로 구성
- 브라우저의 네트워크 탭으로 전송되는 내용 확인 가능(F12, 우클릭 후 검사)



Http 라우팅

- http 프로토콜

The screenshot shows a browser window with the URL <https://pixabay.com/ko/>. The main content is the Pixabay homepage, featuring a large image of a bee on a flower. Below the image, there's a search bar with placeholder text "Pixabay의 모든 이미지 검색" and a button "모든 이미지". The footer includes navigation links like "집", "사진", "일러스트", "벡터", "비디오", "음악", "음향 효과", "GIF", and category filters for "봄", "꽃", "바다", "풍경", "자연", "고양이", "우주", "나무", "숲". The right side of the screen shows the Network tab of the developer tools, which is highlighted with a red box. This tab displays a timeline of network requests and a table of request details. The table columns include 이름 (Name), 상태 (Status), 유형 (Type), 시작점 (Start), 크기 (Size), 시간 (Time), and 폭포 (Throughput). Some requests listed include "recaptcha_ko.js", "messages.json", "tp2", "imgIcon.chunk.png", and "imgClose.chunk.svg". The timeline shows various requests being processed, with colors indicating different types of requests like scripts, stylesheets, and images.

이름	상태	유형	시작점	크기	시간	폭포
recaptcha_ko.js	200	script	api.js?ren...	206 ...	275...	
messages.json	(실...)	xhr	i18n.c9c4...	0 B	64밀...	
messages.json	200	xhr	i18n.c9c4...	7.5 kB	64밀...	
messages.json	200	xhr	i18n.c9c4...	6.6 kB	65밀...	
tp2	200	xhr	chunk-99...	671 B	280...	
imgIcon.chunk.png	200	png	client.1ea...	15.7...	9밀...	
imgResize.chunk.png	200	png	client.1ea...	4.3 kB	8밀...	
imgClose.chunk.svg	200	svg...	client.1ea...	329 B	7밀...	
imgArrow.chunk.svg	200	svg...	client.1ea...	469 B	7밀...	
imgGlobe.chunk.svg	200	svg...	client.1ea...	2.6 kB	5밀...	
imgDoc.chunk.png	200	png	client.1ea...	5.5 kB	5밀...	
imgSend.chunk.svg	200	svg...	client.1ea...	363 B	5밀...	
anchor?ar=1&k=6Le...	200	doc...	recaptcha...	27.7...	251...	
styles_ltr.css	200	styl...	anchor?a...	25.1...	159...	
recaptcha_ko.js	200	script	anchor?a...	206 ...	249...	
rIjZIM8ZNfOeVQTojtt...	200	script	recaptcha...	7.5 kB	41밀...	
logo_48.png	200	png	styles_ltr...	2.4 kB	34밀...	

Http 라우팅

● http 프로토콜

The screenshot shows the Pixabay homepage with a search bar for "이미지 검색". Below the search bar, there's a large image of a butterfly. At the bottom, there are filters for categories like "봄", "꽃", "바다", etc.

The screenshot shows the Network tab in developer tools with the "Fetch/XHR" tab selected. A specific request for "messages.json" is highlighted with an orange box. The details panel shows the request URL as "https://tivan.naver.com/sc2/1/" and the response status as "200 OK". The response body is partially visible, showing JSON data related to a Naver SSP Waterfall List.

```

{
  "head": {
    "version": "0.0.1",
    "description": "Naver SSP Waterfall List"
  },
  "eventTracking": {
    "ackImpressions": [
      {
        "url": "https://tivan.naver.com/sc2/1/"
      }
    ],
    "activeViewImpressions": [
      {
        "url": "https://tivan.naver.com/sc2/2/"
      }
    ],
    "clicks": [
      {
        "url": "https://tivan.naver.com/sc2/3/"
      }
    ],
    "completions": [
      {
        "url": "https://tivan.naver.com/sc2/4/"
      }
    ],
    "attached": [
      ...
    ]
  }
}
  
```

Http 라우팅

- 주요 서비스별 포트 번호

번호	기능
20,21	FTP(파일 전송 프로토콜)
25	SMTP(이메일 발송)
53	DNS 서버
80	웹(HTTP)
110	POP3(이메일 수신)
443	HTTPS

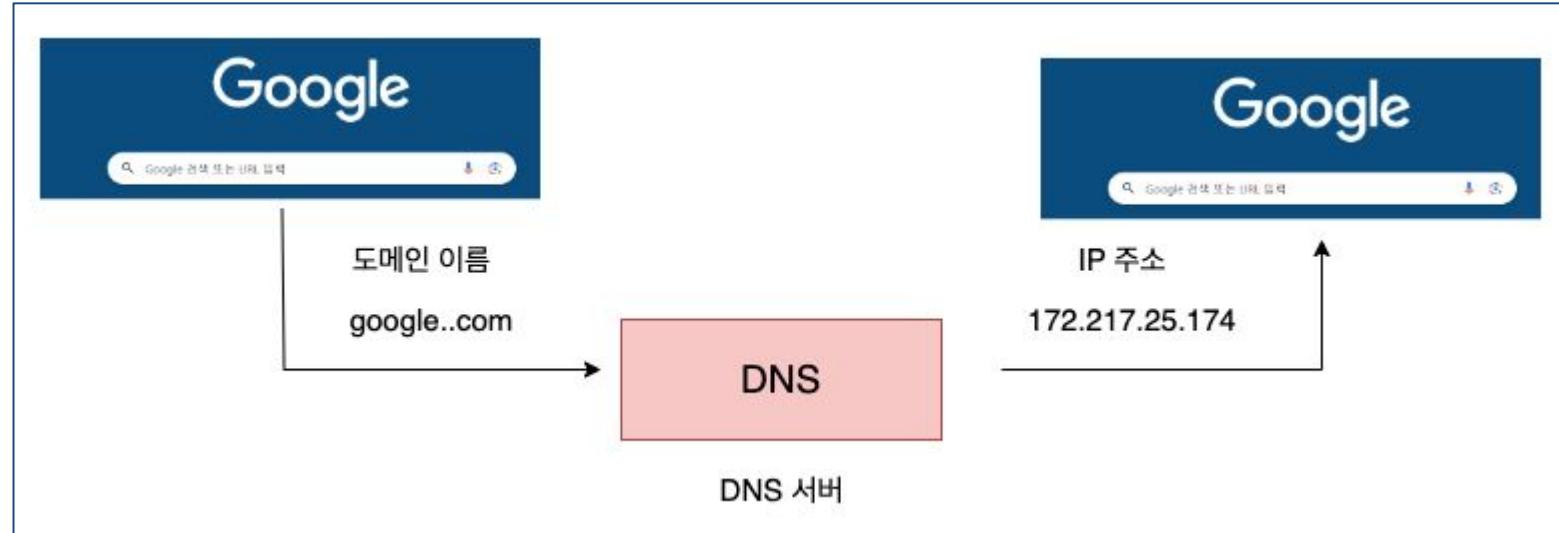


Http 라우팅

● 서버 응답 코드

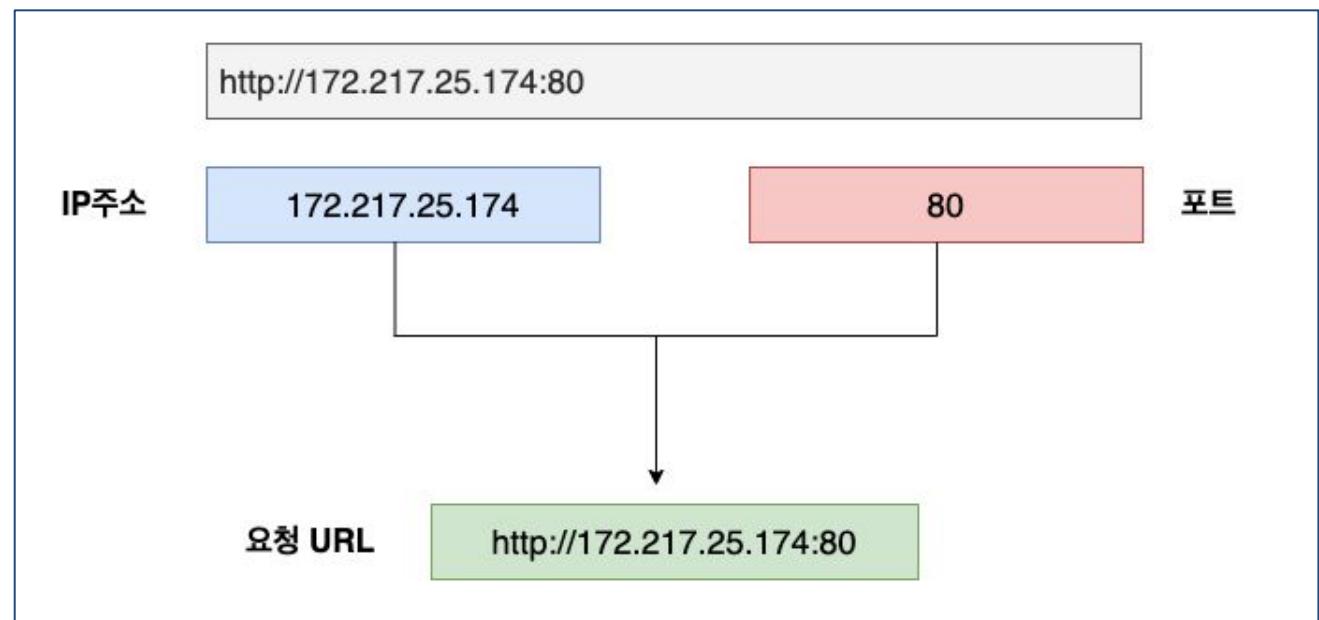
코드	메시지	설명
1xx	Informational	계속 처리 중
2xx	Successful	요청 성공
200	OK	요청이 성공적으로 처리되었습니다.
201	Created	요청이 성공적으로 처리되어 새로운 자료가 생성되었습니다.
204	No Content	요청이 성공적으로 처리되었지만 응답으로 반환할 내용이 없습니다.
3xx	Redirection	다른 위치로 이동
301	Moved Permanently	요청한 데이터가 새 URL로 옮겨졌습니다.
4xx	Client Error	클라이언트 오류
400	Bad Request	클라이언트 요청이 잘못되었거나 유효하지 않습니다.
401	Unauthorized	권한이 없어 거절되었지만 인증을 다시 시도할 수 있습니다.
403	Forbidden	권한이 없어 거절되었고 인증을 시도하면 계속 거절됩니다.
404	Not Found	해당 데이터를 찾을 수 없습니다.
5xx	Server Error	서버 오류
500	Internal Server Error	서버에 요청을 처리하는 동안 오류가 발생했습니다.
503	Service Unavailable	요청한 서비스를 이용할 수 없습니다.

Http 라우팅



DNS서버를 통해
획득한 ip주소로
요청

동일한 ip를 가지는
하나의 pc내에서는
포트로 서비스를
구분



Http 라우팅

- http 서버 만들기

기본형

```
http.createServer([옵션][, 콜백]);
```

```
const server = http.createServer((req, res) => {
  // console.log("요청 발생");
  console.log("request from client")
});
```

- http 서버 실행하기

기본형

```
server.listen(포트[, 호스트][, 콜백])
```

```
server.listen(3000, () => {
  console.log("3000번 포트에서 서버 실행 중");
});
```

Http 라우팅

```

1 // HTTP 모듈로 서버 만들고 실행하기
2
3 const http = require("http");
4
5 const server = http.createServer((req, res) => {
6   // console.log("요청 발생");
7   console.log("request from client")
8 });
9
10 server.listen(3000, () => {
11   console.log("3000번 포트에서 서버 실행 중");
12 });

```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- alicia@Aliciaui-MacBookAir results % node server-1
3000번 포트에서 서버 실행 중

alicia@Aliciaui-MacBookAir results % node server-1
3000번 포트에서 서버 실행 중
request from client
request from client

localhost:3000

서버 중지

alicia@Aliciaui-MacBookAir results %
3000번 포트에서 서버 실행 중
request from client
request from client
^C
alicia@Aliciaui-MacBookAir results %

localhost:3000



사이트에 연결할 수 없음

localhost에서 연결을 거부했습니다.

다음 방법을 시도해 보세요.

- 연결 확인
- 프록시 및 방화벽 확인

ERR_CONNECTION_REFUSED

Http 라우팅

● 요청과 응답 객체(노드에서)

- 클라이언트에서 서버로 요청하면 요청 객체 생성
- 서버에서 클라이언트로 응답할 때 응답 객체 생성



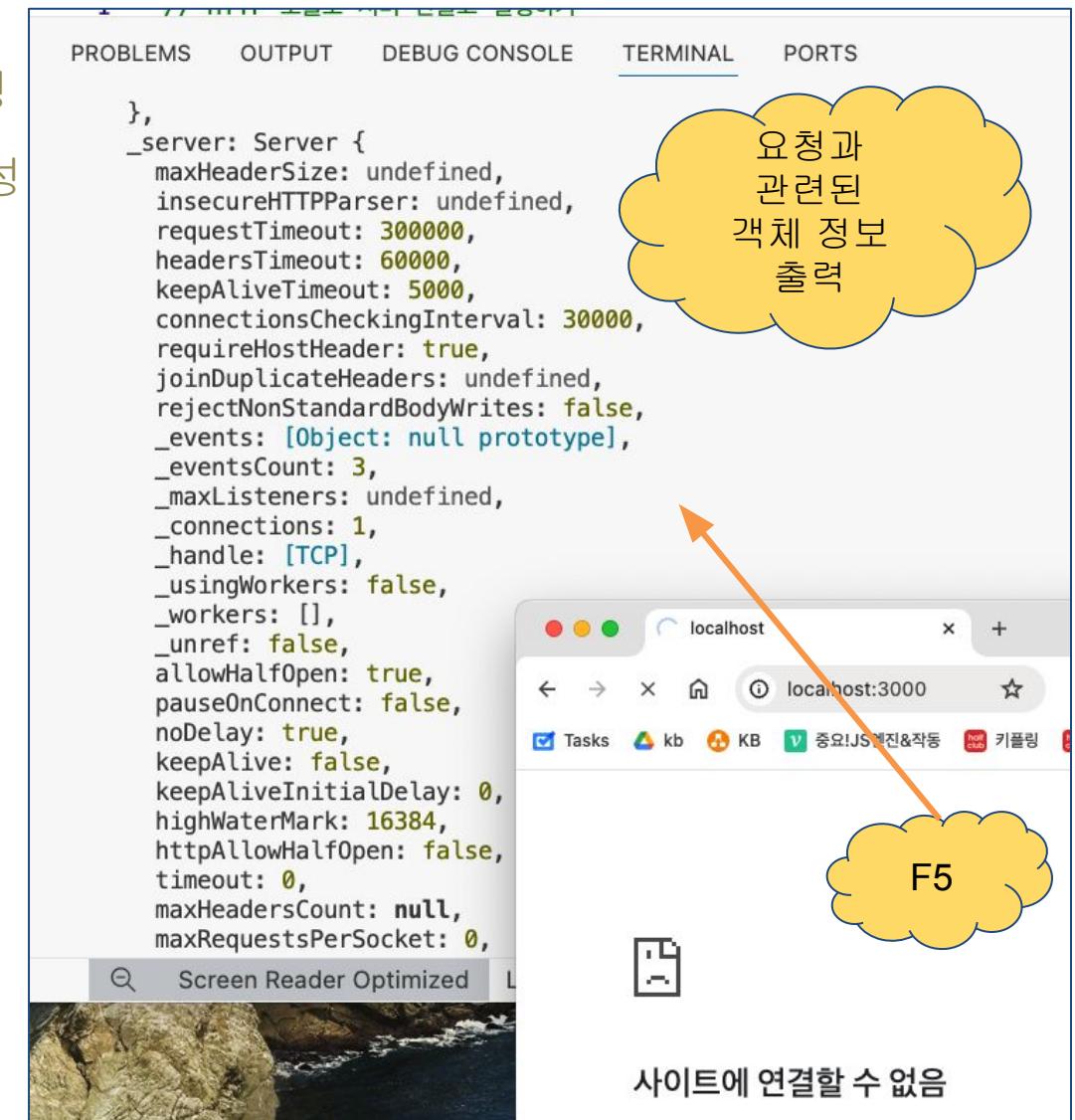
```

1 // HTTP 모듈로 서버 만들고 실행하기
2
3 const http = require("http");
4
5 const server = http.createServer((req, res) => {
6   console.log(req)
7 });
8
9 server.listen(3000, () => {
10   console.log("3000번 포트에서 서버 실행 중");
11 });

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- alicia@Aliciaui-MacBookAir results % node server-2
3000번 포트에서 서버 실행 중



Http 라우팅

- 요청과 응답 객체(노드에서)

- 응답 헤더 만들기

기본형

```
res.setHeader(이름, 값)  
res.writeHead(상태 코드[, 상태 메시지[, 헤더]])
```

- 응답 본문 만들기

기본형

```
res.write(내용[, 인코딩[, 콜백]]);
```

기본형

```
res.end(내용[, 인코딩[, 콜백]]);
```

```
res.write("내용1");  
res.write("내용2");  
res.write("내용3");  
res.end()
```

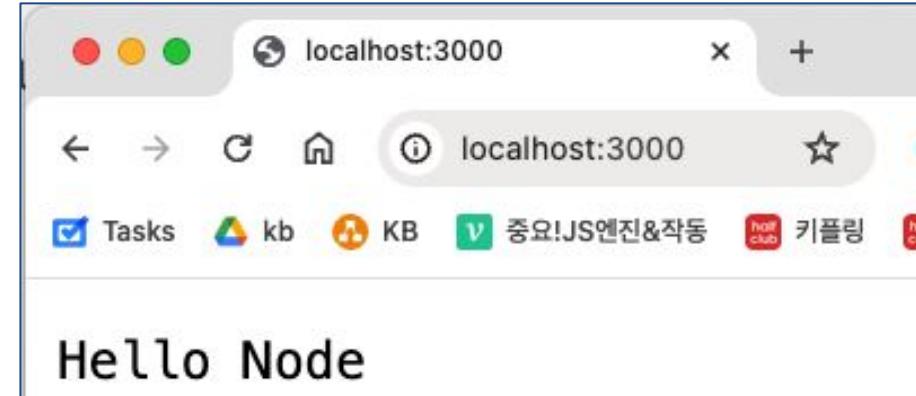
Http 라우팅

```

1 // 응답 객체 확인하기 - 응답 헤더, 응답 본문
2
3 const http = require("http");
4
5 const server = http.createServer((req, res) => {
6   console.log(req.method); // 요청 메서드 확인
7
8   res.setHeader("Content-Type", "text/plain"); // 응답 헤더
9   res.write("Hello Node"); // 응답 본문
10  res.end(); // 응답 종료
11 });
12
13 server.listen(3000, () => {
14   console.log("3000번 포트에서 서버 실행 중");
15 });
...

```

server-3.js



Hello Node

PROBLEMS OUTPUT DEBUG CONSOLE TE
alicia@Aliciaui-MacBookAir results % node
3000번 포트에서 서버 실행 중
GET
GET

GET1 → favicon.ico
** favicon을 찾기 위해 사용할 아이콘

이름	상태	유형	시작점	크기	시간	폭포
localhost	200	doc...	기타	177 B	8밀...	
hook-exec.js	200	scri...	hook.js:1	6.9 ...	28...	
detector-exec.js	200	scri...	detector.js:1	1.2 ...	2밀...	
favicon.ico	200	text...	기타	177 B	2밀...	

GET2 → localhost

네트워크

Http 라우팅

- index.html 서빙하기



```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Do it! Node.js</title>
  <style>
    * {
    }
    body {
    }
    .wrapper {
    }
    .wrapper h1 {
    }
    .wrapper p {
    }
  </style>
</head>
<body>
  <div class="wrapper">
    <h1>Welcome to Node.js</h1>
    <p>Node.js 세계에 오신 것을 환영합니다.</p>
  </div>
</body>
</html>
```

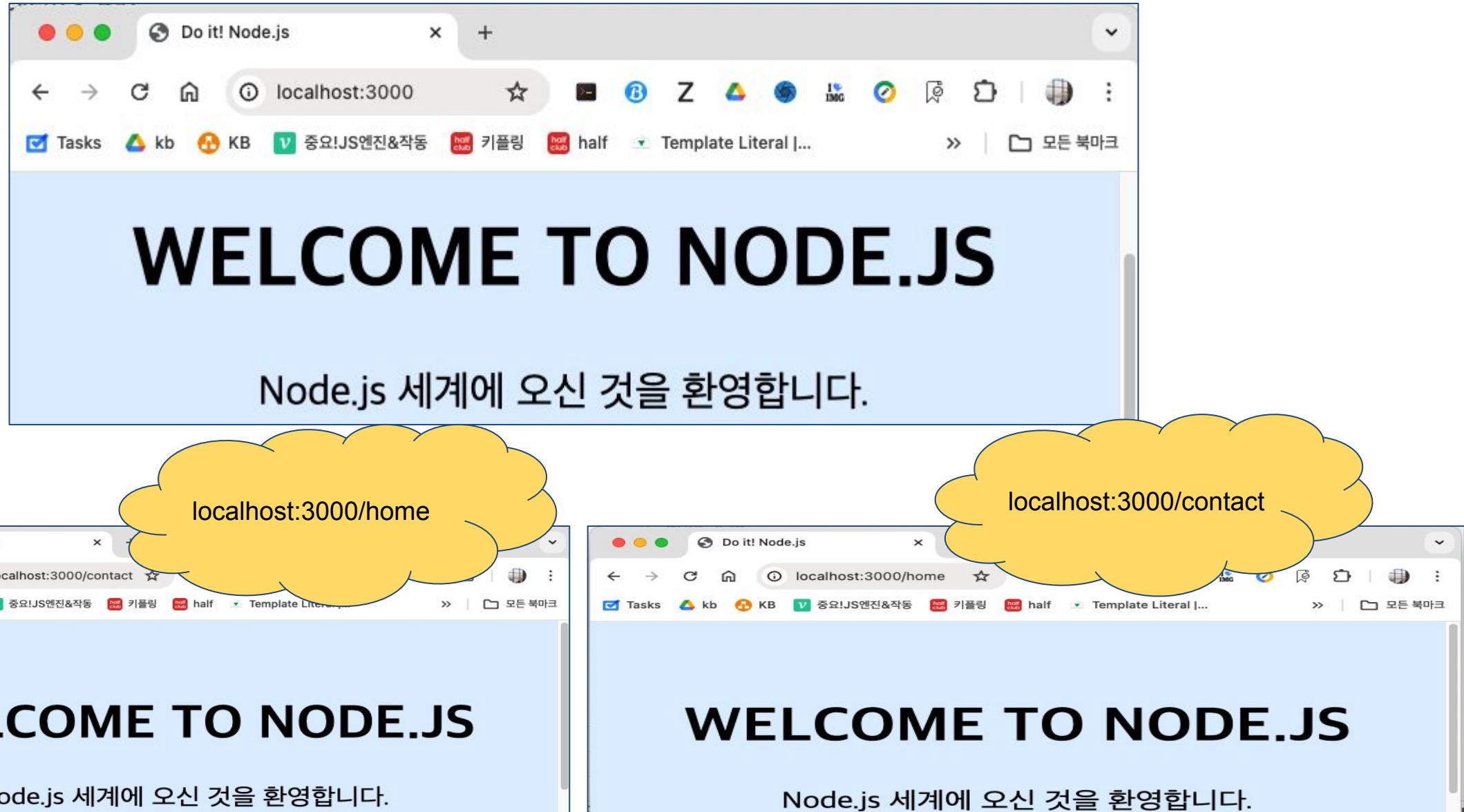


```
1 // HTML 페이지 서빙하기
2
3 const http = require("http");
4 const fs = require("fs");
5
6 const server = http.createServer((req, res) => {
7   res.setHeader("Content-Type", "text/html");
8   // 위 코드를 아래와 같이 작성해도 됩니다.
9   // res.writeHead(200, { "Content-Type": "text/html" });
10  const readStream = fs.createReadStream(__dirname + "/index.html", "utf8");
11  readStream.pipe(res);
12});
13
14 server.listen(3000, () => {
15   console.log("3000번 포트에서 서버 실행 중");
16 });
17
```

PROBLEMS OUTPUT TERMINAL ... node - results + ×

alicia@Aliciaui-MacBookAir results % node server-4
3000번 포트에서 서버 실행 중

Http 라우팅



Http 라우팅

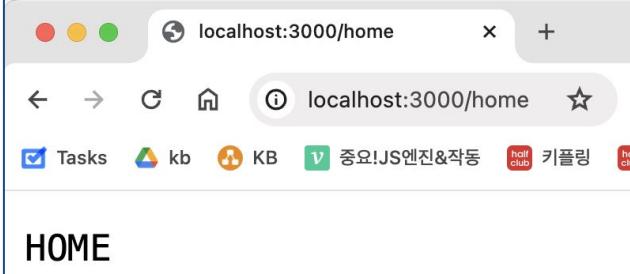
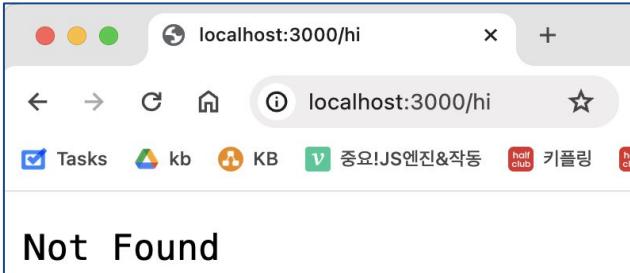
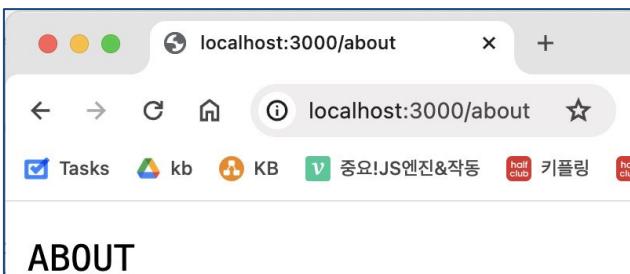
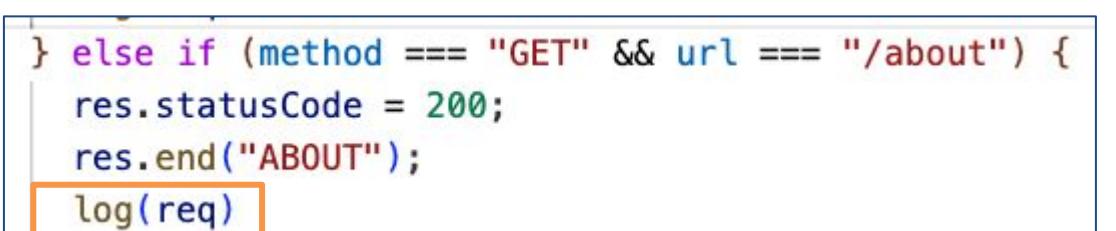
```

3 const http = require("http");
4
5 const server = http.createServer((req, res) => {
6   // 요청 메서드와 URL 가져오기
7   const { method, url } = req;
8   res.setHeader("Content-Type", "text/plain");
9
10  // URL에 따라 응답을 다르게 처리
11  if (method === "GET" && url === "/home") {
12    res.statusCode = 200;
13    res.end("HOME");
14  } else if (method === "GET" && url === "/about") {
15    res.statusCode = 200;
16    res.end("ABOUT");
17  } else {
18    res.statusCode = 404;
19    res.end("Not Found");
20  }
21);
22
23 server.listen(3000, () => {
24   console.log("3000번 포트에서 서버 실행 중");
25 });
26

```

PROBLEMS OUTPUT TERMINAL ...

alicia@Aliciaui-MacBookAir results % node server-5
3000번 포트에서 서버 실행 중

The browser screenshots show the following request headers for the 'ABOUT' page:

```

'Connection': 'keep-alive',
'Cache-Control': 'max-age=0',
'sec-ch-ua': '"Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"',
'sec-ch-ua-mobile': '?0',
'sec-ch-ua-platform': '"macOS"',
'Upgrade-Insecure-Requests': '1',
'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) fari/537.36',
'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn;q=0.8,image/apng;q=0.7,*/*;q=0.9',
'Sec-Fetch-Site': 'none',
'Sec-Fetch-Mode': 'navigate',
'Sec-Fetch-User': '?1',
'Sec-Fetch-Dest': 'document',
'Accept-Encoding': 'gzip, deflate, br, zstd',
'Accept-Language': 'ko,ko-KR;q=0.9,en-US;q=0.8,en;q=0.7',
],
rawHeaders: [],
joinDuplicateHeaders: null,
aborted: false,
upgrade: false,
url: '/about',
method: 'GET'
]

```

동기와 비동기

동기와 비동기

● 동기 처리

- 코드에 작성한 순서대로 실행
- 자바 스크립트는 싱글 스레드 언어
- 스레드 - 작업을 처리하기 위해 자원을 사용하는 단위, 하나의 작업이 실행되는 최소 단위
- 자바 스크립트에는 스레드가 하나밖에 없어서 한 번에 하나의 작업만 처리할 수 있다.
- 자바는 멀티 스레드 언어, 스레드를 여러 개 가지고 있어서 동시에 여러 작업을 실행할 수 있음.

```

3 console.log("첫번째 작업");
4 console.log(`두번째 작업`);
5 console.log(`세번째 작업`);

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

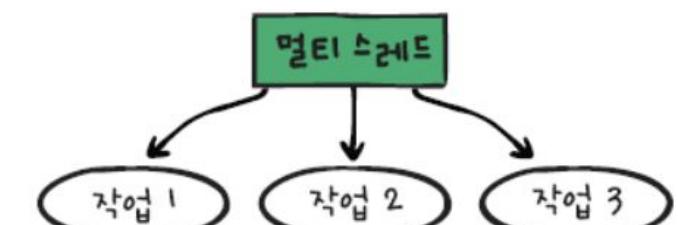
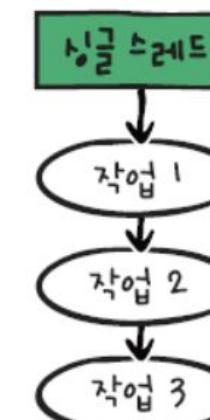
```

● alicia@Aliciaui-MacBookAir results % node sync.js

```

첫번째 작업
두번째 작업
세번째 작업

```



동기와 비동기

- 싱글 스레드가 한 번에 하나의 작업만 처리할 수 있다면 중간에 시간이 많이 걸리는 작업은??
- 작업1 → 3초후 작업2 → 작업3 순서X
- 작업1 → 작업3 → 3초후 작업2 순서O

The screenshot shows a code editor and a terminal window. The code editor has a snippet of JavaScript:

```
1 // 자바스크립트의 비동기 처리
2
3 console.log("첫번째 작업");
4 setTimeout(() => {
5   console.log("두번째 작업");
6 }, 3000);
7 console.log(`세번째 작업`);
```

Three green arrows point from the log statements in the code to their corresponding outputs in the terminal window:

Terminal Output	Order
alicia@Aliciaui-MacBookAir results % node async-1	첫 번째 작업
	세 번째 작업
	두 번째 작업

The terminal window also displays the file path and command used to run the script.

동기와 비동기

- **setTimeout**함수에서 시간을 0초로 지정하는 경우, 0초로 지정하더라도 **setTimeout**자체가 시간 지연을 포함하는 함수 → 즉시 처리할 수 있는 작업을 다 끝낸 후에 이어서 시간이

```
3 console.log("첫번째 작업");
4 setTimeout(() => {
5   console.log("두번째 작업");
6 }, 0);
7 console.log(`세번째 작업`);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- alicia@Aliciaui-MacBookAir results % node async-2

첫 번째 작업
세 번째 작업
두 번째 작업

- 
- **setTimeout()**는 비동기 처리 함수

동기와 비동기

- 사용자 정의 비동기 함수도 생성 가능

- 콜백함수
- 프라미스
- async/await

- 파일 이름을 표시하는 함수가 아무리 빨리 끝나더라도 디렉토리를 읽은 후에 실행해야 함.
- readdir함수를 실행 한 후에 이어서 실행하도록 콜백 형태로 작성

```

3 const fs = require('fs');
4
5 let files = fs.readdir("./", (err, files) => {
6   if (err) {
7     console.error(err);
8   }
9   console.log(files);
10 }
11
12 console.log("Code is done.");
13

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- alicia@Aliciaui-MacBookAir results % node async-3
Code is done.
[
 'async-1.js',
 'async-2.js',
 'async-3.js',
 'await.js',
 'blocking-1.js',
 'blocking-2.js',
 'blocking-3.js',
 'node-sync.js',
 'non-blocking.js',
 'promise.js',
 'sync.js'
]

동기와 비동기

• 논블로킹 I/O

- 노드에서는 대부분의 작업을 비동기로 처리하는데 노드가 논블로킹 I/O로 동작하기 때문
- I/O : input/output, 자료를 주고 받는다는 의미
- 블로킹 : 코드 실행을 중간에 막는 것, 코드 실행이 멈춘다는 의미
- `readFileSync()` : 동기 함수를 사용하고 있어서 파일을 다 읽어 오기 전까지는 그 다음줄 `console.log(data)`를 처리할 수 없음. “코드 끝!”을 보고 싶은 경우 파일을 다 읽을 때까지 기다려야 함.

```

3 const fs = require("fs");
4
5 const data = fs.readFileSync("example.txt"); // 블로킹
6 console.log(data); // 파일 읽기가 끝날 때까지 대기
7 console.log("코드 끝!"); // 파일을 읽고 내용을 표시할 때까지 대기
8

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● alicia@Aliciaui-MacBookAir results % node blocking-1
<Buffer 4e 6f 64 65 2e 6a 73 20 69 73 20 61 6e 20 6f 70 65 6e 2
6c 61 74 66 6f 72 6d 20 4a 61 76 61 53 63 72 69 ... 110 more b
코드 끝!

동기와 비동기

• 블로킹 I/O

```

3 const { log } = require("console");
4 const fs = require("fs");
5
6 // const data = fs.readFileSync("example.txt"); // 블로킹
7 // log(data.toString()); // 파일 읽기가 끝날 때까지 대기
8 // log("코드 끝!"); // 파일을 읽고 내용을 표시할 때까지 대기
9
10 const data2 = fs.readFile("example.txt", (error, readData) => {
11   if(error){
12     log("error");
13   }
14   log(readData.toString());
15 }
16 log("이걸 보고 싶은 거였어!! ======> 코드 끝!");
17

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

✖ alicia@Aliciaui-MacBookAir results % node blocking-1
이걸 보고 싶은 거였어 !! ======> 코드 끝!
^C

readFile()을 통해 파일을 다
읽어오는 것보다
log(readData.toString())을 먼저
실행 함.

example.txt ×

```

388784 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388785 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388786 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388787 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388788 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388789 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388790 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388791 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388792 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388793 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388794 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388795 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388796 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388797 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388798 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388799 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388800 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388801 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. Node.js is
388802 Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다. ----

```

example.txt파일의 양을 늘려서
다시 테스트해보자!

동기와 비동기

- 블로킹 I/O

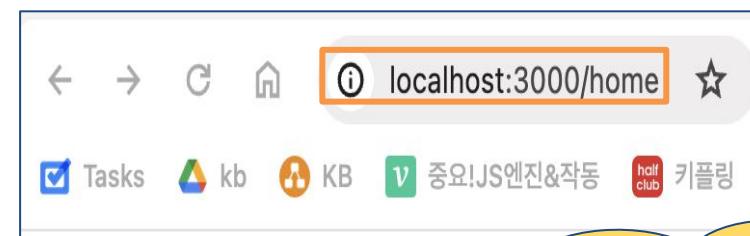
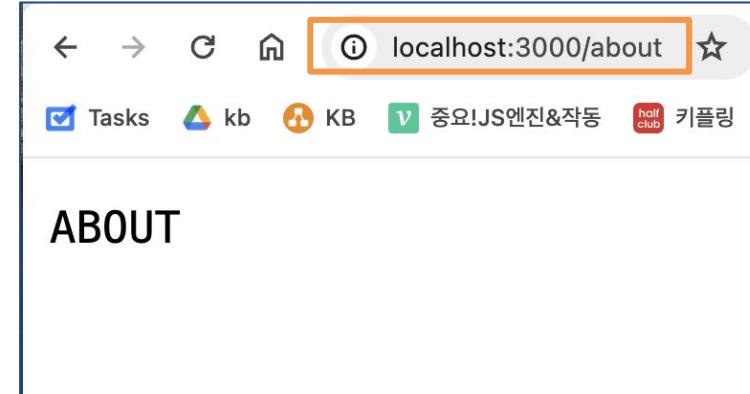
```

5  const server = http.createServer((req, res) => {
6    if (req.url === "/home") {
7      res.end("HOME");
8    } else if (req.url === "/about") {
9      res.end("ABOUT");
10   } else {
11     res.end("Not Found");
12   }
13 );
14
15 server.listen(3000, () => {
16   console.log("3000번 포트에서 서버 실행 중");
17 });

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

alicia@Aliciaui-MacBookAir results % node blocking-2
3000번 포트에서 서버 실행 중



about호출 후 → home주소로
입력하여 호출하면 바로
랜더링됨.

동기와 비동기

● 블로킹 I/O

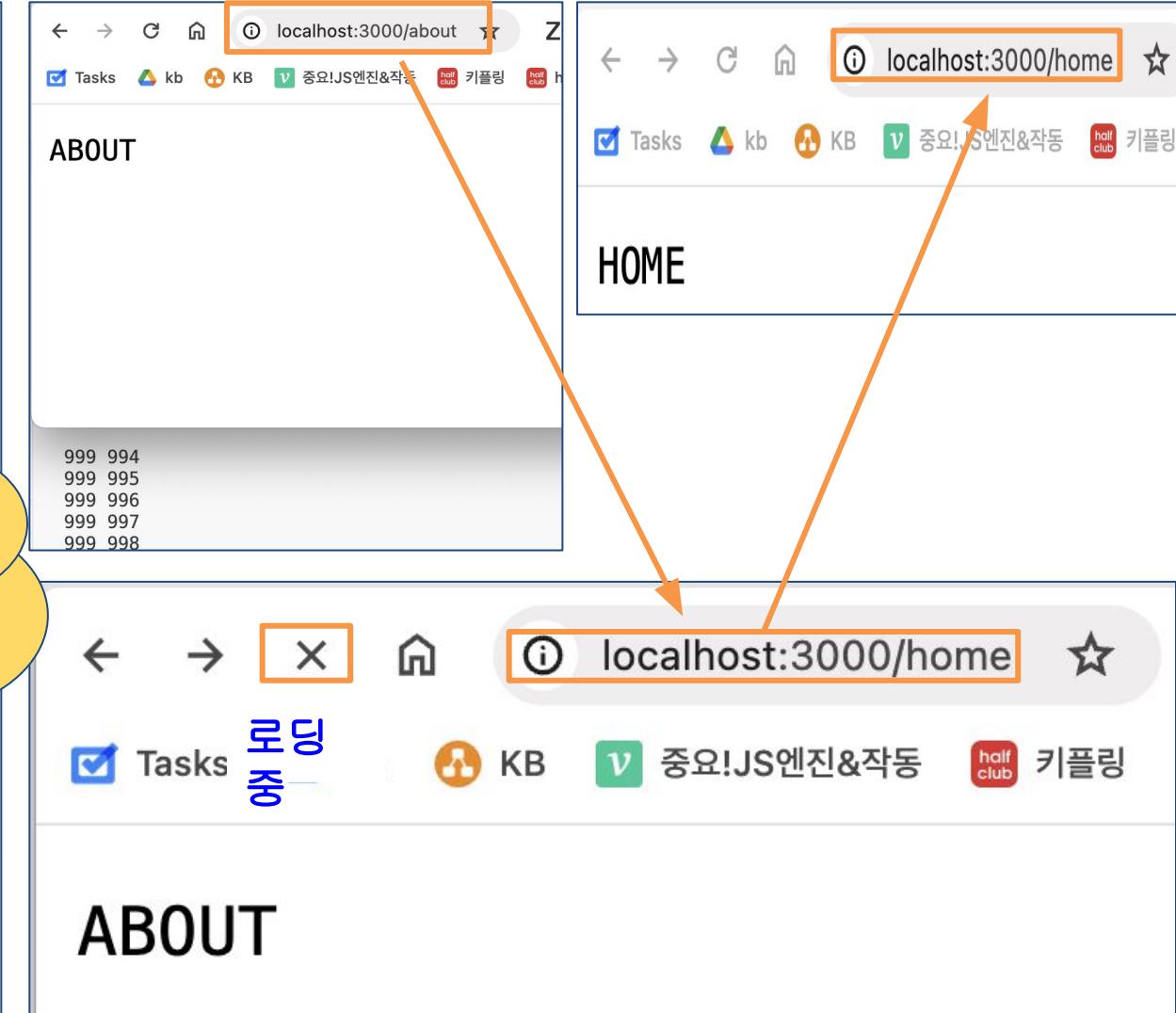
```

3  const http = ...
4
5  const server = http.createServer((req, res) => {
6    if (req.url === "/home") {
7      res.end("HOME");
8    } else if (req.url === "/about") {
9      for (let i = 0; i < 1000; i++) {
10        for (let j = 0; j < 1000; j++) {
11          console.log(`${i} ${j}`);
12        }
13      }
14      res.end();
15    } else {
16      res.end("404");
17    }
18  });
19  server.listen(3000);
20  console.log("Server is running on port 3000");
21
22
  
```

시간이 오래 걸리는 작업이 있는 about호출 후 → home주소로 입력하여 호출하면 home은 렌더링되지 않고 about이 그대로 랜더링 되어있음. about이 아직 처리되지 않았기 때문에 다음 호출인 home을 처리할 수 없음.
→ 블로킹

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

alicia@Aliciaui-MacBookAir results % node blocking-3
3000번 포트에서 서버 실행 중



동기와 비동기

• 논블로킹 I/O

- 블로킹이 생기지 않게 하려면 비동기 처리 이용해야함.
- 비동기 처리하면 코드를 실행하다가 시간이 걸리는 작업은 잠시 옆으로 빼놓고 즉시 실행해야 할 작업 먼저 처리
- 옆으로 빼놓은 비동기 작업은 이벤트 루프에서 처리

```

3 const fs = require("fs");
4
5 const data = fs.readFile("example.txt", "utf8", (err, data) => {
6   if (err) {
7     console.error(err);
8   }
9   console.log(data);
10 }
11 console.log("코드 끝!"); // 파일 읽기 전에 실행
12

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

alicia@Aliciaui-MacBookAir results % node non-blocking.js
코드 끝!

Node.js is an open-source, cross-platform JavaScript runtime environment.
Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다.

백엔드 개발을 할 때 네트워크를 통해 클라이언트와 서버 간에 자료를 주고받아야 하는 작업에서 주고받는 데이터 양도 다르고 네트워크 속도도 일정하지 않아 시간 지연을 항상 고려해야 함.

→ 백엔드 개발에서 네트워크와 관련된 작업을 할 때는 중간에 멈추는 일이 없도록 비동기 처리를 해야함.

→ 논블로킹

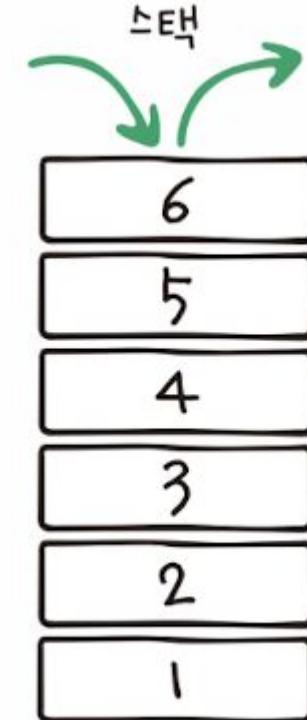
→ 노드 = 싱글 스레드(이벤트 루프 사용하여 비동기 처리) + 동시 작업을 해야 할 때는 논블로킹 방식 사용

→ 노드에서는 멀티 스레드, 병렬 처리도 가능하지만 주로 싱글 스레드를 사용함.

동기와 비동기

• 콜 스택

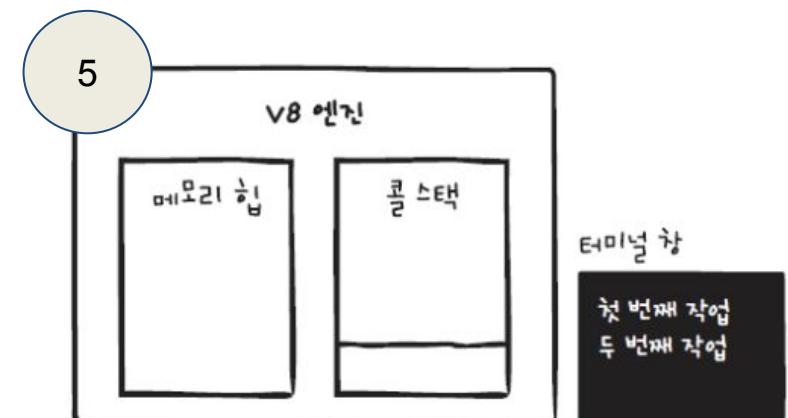
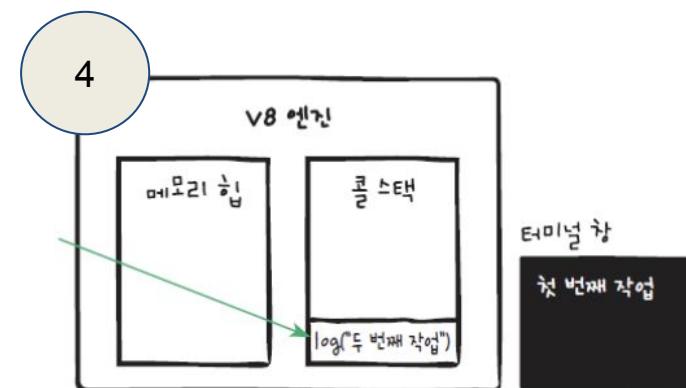
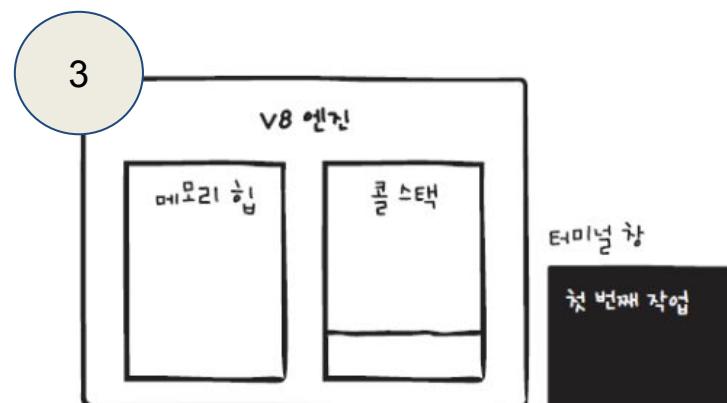
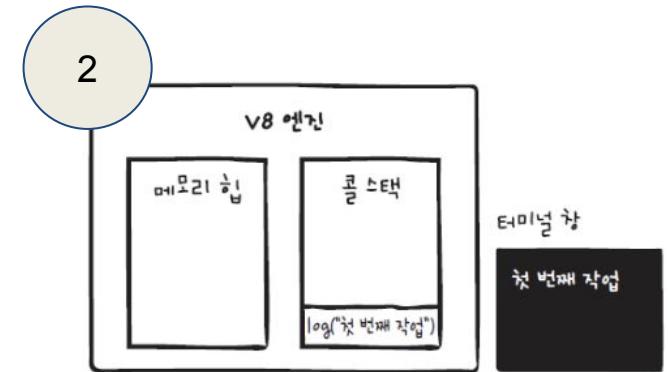
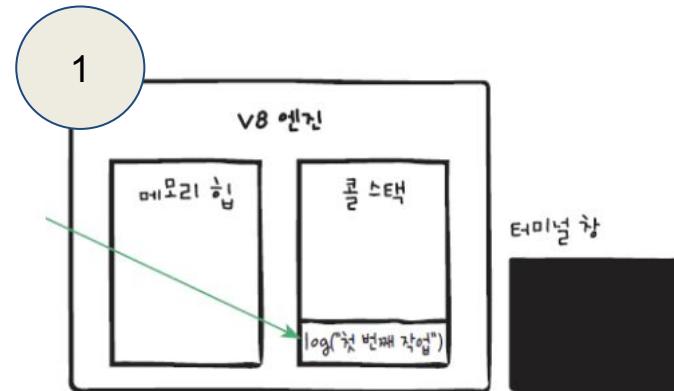
- 노드에서 비동기 처리를 할 때 사용
- 노드에서는 V8엔진 사용해서 JS코드 처리
- 콜 스택(call stack) : 순서대로 작업을 처리, 콜이 스택 형태로 모여있다라는
 - 실행할 함수들을 스택(stack, 누적, 쌓는다라는 의미)에 모아둠.
 - stack : LIFO(First Input Last Output, 후입선출)



동기와 비동기

- 콜 스택

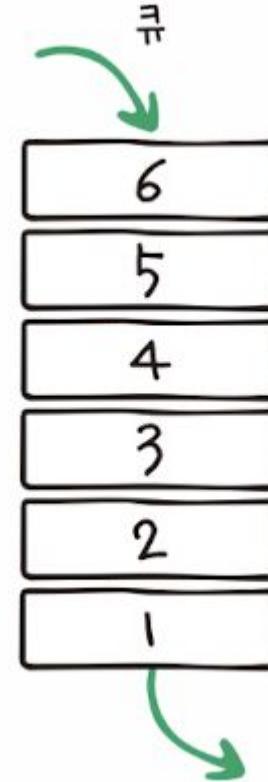
```
console.log("첫번째 작업");
setTimeout(() => {
  console.log("두번째 작업");
}, 3000);
console.log(`세번째 작업`);
```



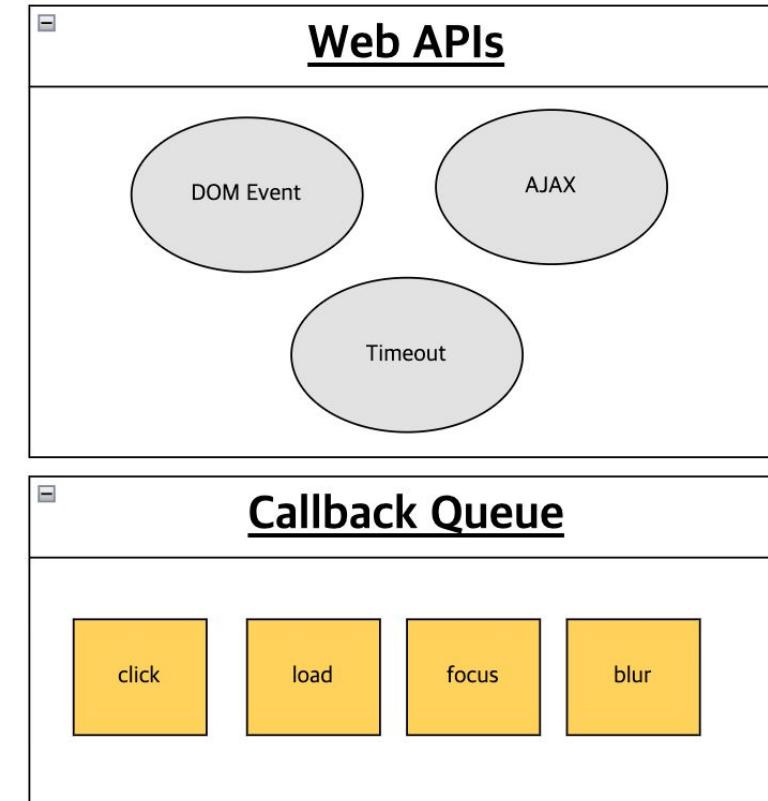
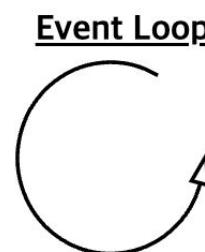
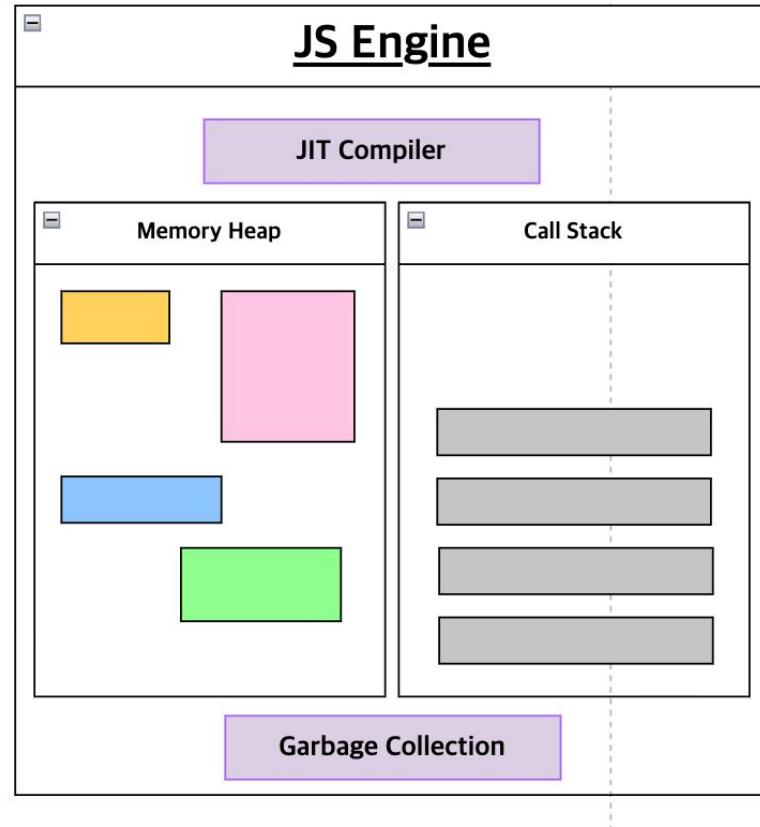
동기와 비동기

• 콜백 큐

- 콜백함수가 큐 형태로 저장되는 공간
- queue : FIFO(First Input First Output, 선입선출)
- V8엔진의 콜 스택은 실행한 순서대로 함수를 가지고 오므로 비동기 처리
- V8엔진에서 비동기 처리를 할 수 있는 라이브러리 - libuv
- 노드에서는 내부에서 libuv를 사용해 비동기 작업을 쉽게 다룰 수 있음.
- libuv = NodeAPI + 콜백큐



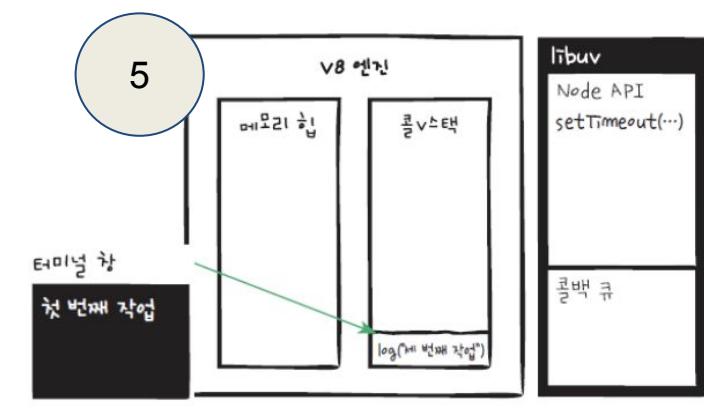
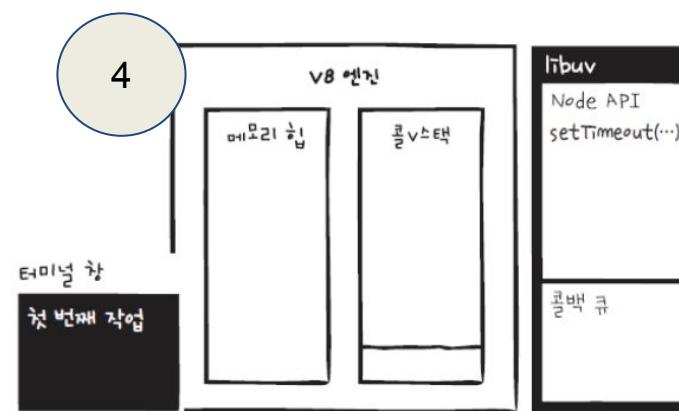
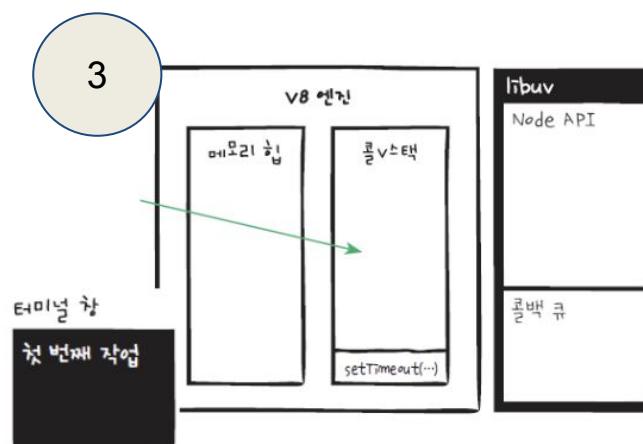
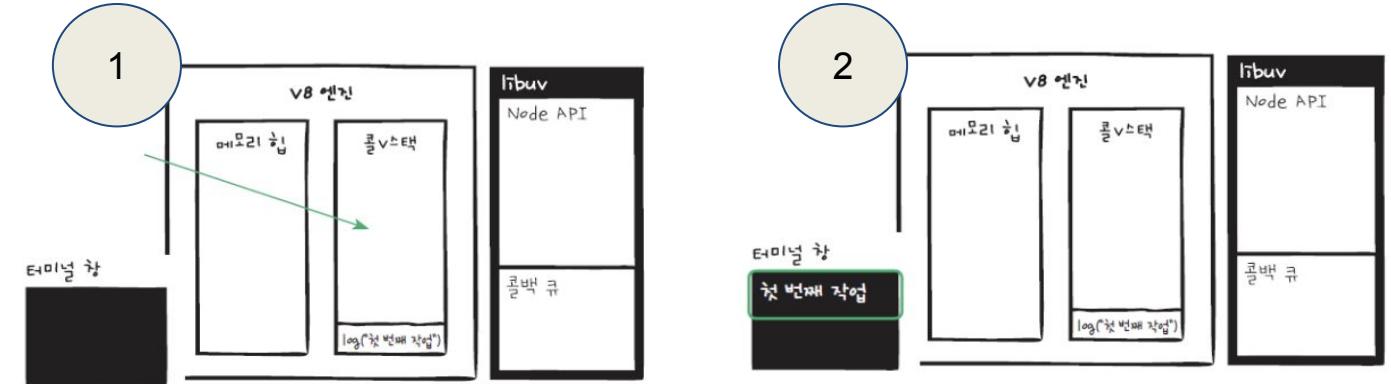
JavaScript Engine



동기와 비동기

- 콜백 큐

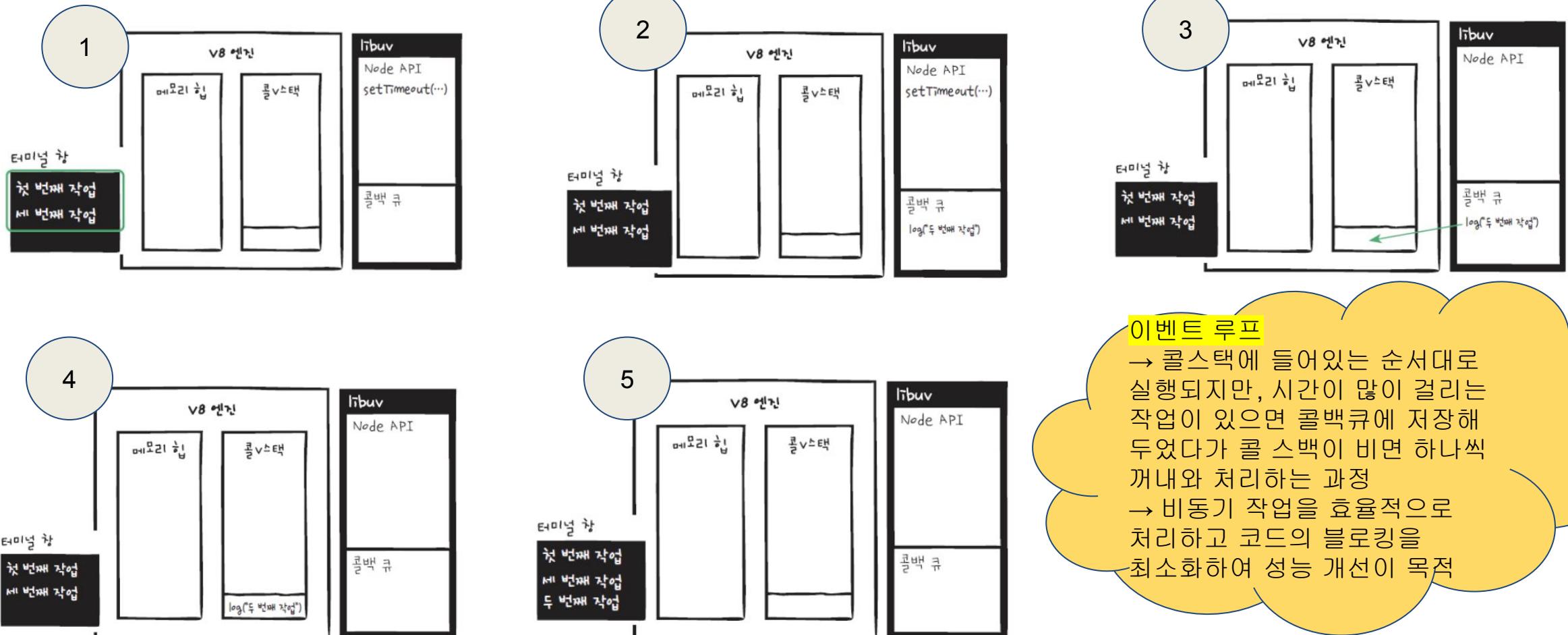
```
console.log("첫번째 작업");
setTimeout(() => {
  console.log("두번째 작업");
}, 3000);
console.log(`세번째 작업`);
```



동기와 비동기

● 이벤트 루프

○ 비동기 처리하기



이벤트 루프

→ 콜스택에 들어있는 순서대로 실행되지만, 시간이 많이 걸리는 작업이 있으면 콜백큐에 저장해 두었다가 콜스택이 비면 하나씩 꺼내와 처리하는 과정
 → 비동기 작업을 효율적으로 처리하고 코드의 블로킹을 최소화하여 성능 개선이 목적

동기와 비동기

- 노드의 비동기 패턴

- 콜백함수

- 다른 함수 안에 포함된 함수
- 다른 함수의 매개변수로 넣는 함수

```

5  const data = fs.readFile("example.txt", "utf8", (err, data) => {
6    if (err) {
7      console.error(err);
8    }
9    console.log(data);
10 });
11 console.log("코드 끝!"); // 파일 읽기 전에 실행
12

```

readFile()함수는 아직 다
실행이 완료된 것이 아니므로
어딘가에 실행하는 항목으로
저장되었다가 log("코드 끝!")이
실행되고 스택큐가 비었을 때
다시 readFile() 호출하여 실행
→ 스택큐

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- alicia@Aliciaui-MacBookAir results % node non-blocking.js
코드 끝!

Node.js is an open-source, cross-platform JavaScript runtime environment.
Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임입니다.

- 노드의 비동기 패턴

- 콜백 함수
 - 콜백 지옥

- 여러 개를 연속해서 실행하는 경우 콜백 함수 안에 또 다른 콜백 함수, 그 안에 또 다른 콜백 함수를 넣게 되어 코드가 복잡해지고, 실행되는 함수 호출도

물명확해보임

파일 읽기() {

파일 내용 프린트() +

파일 글자 수 프린트()

파일 짹수 글자수 파일명 프린트

() {

}

】

콜백 지옥

}

```
// 콜백 함수를 사용해 비동기 처리하기
1
2 
3 const fs = require("fs");
4
5 let files = fs.readdir("./", (err, files) => {
6   if (err) {
7     console.error(err);
8   }
9   console.log(files);
10 });
11
12 console.log("Code is done.");
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

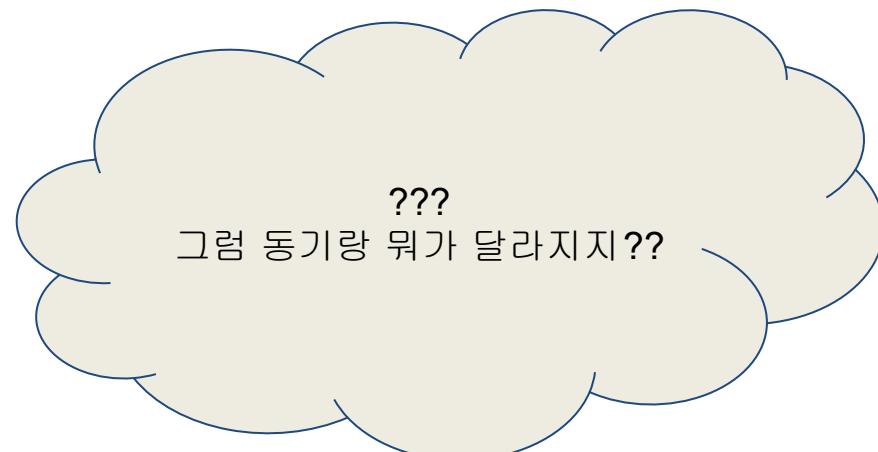
```
● alicia@Aliciaui-MacBookAir results % node async-3
Code is done.
[
  'async-1.js',
  'async-2.js',
  'async-3.js',
  'await.js',
  'blocking-1.js',
  'blocking-2.js',
  'blocking-3.js',
  'example copy.txt',
  'example.txt',
  'node-sync.js',
  'non-blocking.js',
  'promise.js',
  'sync.js'
]
```

동기와 비동기

● 노드의 비동기 패턴

○ 프로미스(Promise)

- 실행 결과의 반환값이 true일 때 then처리, false일 때 catch처리
- true, false와 상관없이 finally는 무조건 실행할 처리 내용 정의
- 노드에서 promise 사용시 가지고 올 모듈에 .promises를 붙여주어야 함.



```

3 const fs = require("fs").promises;
4
5 fs.readdir("./")
6   .then((result) => console.log(result))
7   .then((result) => console.log("----내가 그 다음 ----"))
8   .then((result) => console.log("----나도 그 다음 ----"))
9   .catch((err) => console.error(err));
10

```

PROBLEMS OUTPUT DEBUG CONSOLE

● alicia@Aliciaui-MacBookAir results % node promises

```
[
  'async-1.js',
  'async-2.js',
  'async-3.js',
  'await.js',
  'blocking-1.js',
  'blocking-2.js',
  'blocking-3.js',
  'example copy.txt',
  'example.txt',
  'node-sync.js',
  'non-blocking.js',
  'promise.js',
  'sync.js'
]
```

----내가 그 다음 ----
----나도 그 다음 ----

프로미스
체이닝

동기와 비동기

• 노드의 비동기 패턴

- **async/await**

- ECMA JS 2017(ES8)에서 정의된 방법
- 사용법이 쉬움
- 비동기 처리할 함수 앞에 **await**를 붙여줌.
await를 붙일 수 있는 프로미스를 반환하는
함수여야함.
- 정의된 함수 앞에 **async**를 붙여줌.
- **try ~ catch**와 함께 사용함. **try { 실행코드 }**
catch(e) { 에러발생시 실행 코드}

```

3 const fs = ...
4
5 async function readDirAsync() {
6   try {
7     const files = await fs.readdir("./");
8     console.log(files);
9   } catch {
10     console.error(err);
11   }
12 }
13
14 readDirAsync();

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

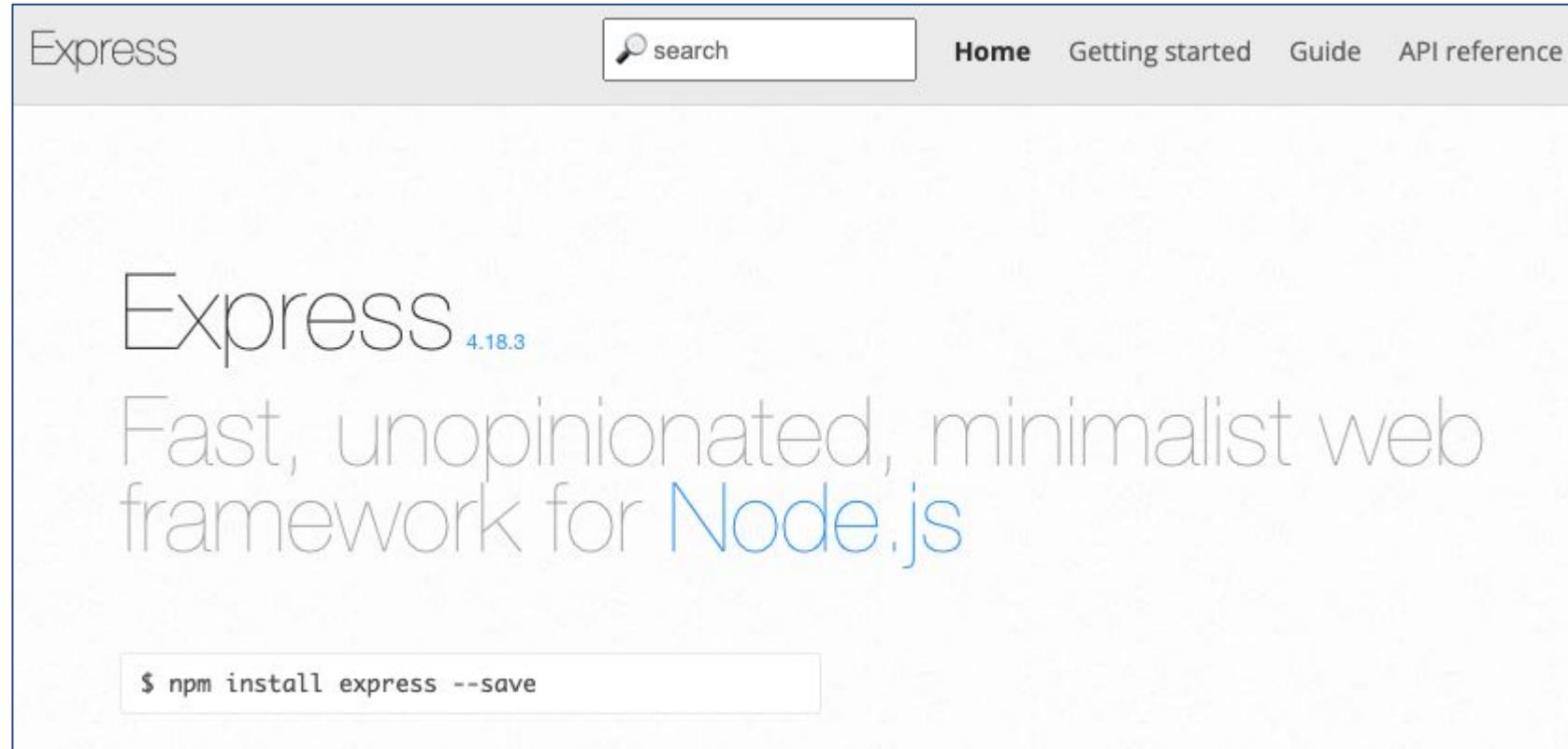
● alicia@Aliciaui-MacBookAir results % node await.js

```
[  
  'async-1.js',  
  'async-2.js',  
  'async-3.js',  
  'await.js',  
  'blocking-1.js',  
  'blocking-2.js',  
  'blocking-3.js',  
  'example copy.txt',  
  'example.txt',  
  'node-sync.js',  
  'non-blocking.js',  
  'promise.js',  
  'sync.js'  
]
```

express

express

- nodeJS에서 백엔드 웹개발을 할 수 있도록 http요청/응답/라우팅 등의 기능을 가진 프로그램 → nodeJS 프레임워크

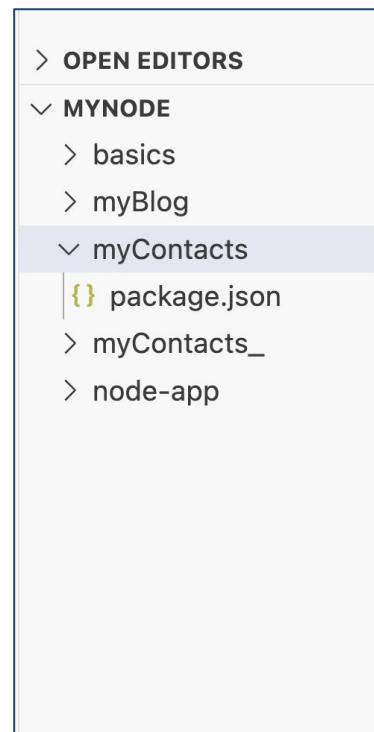
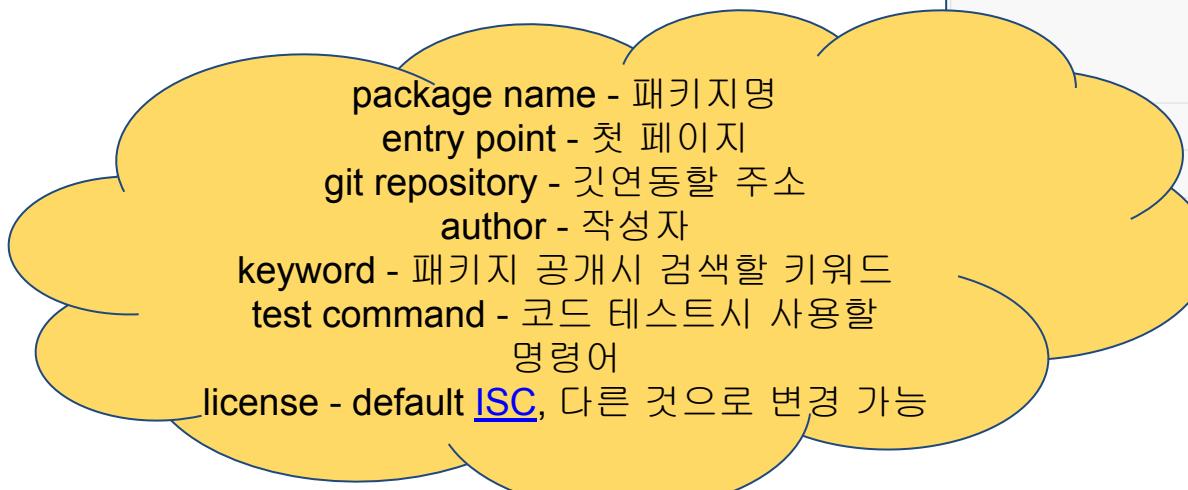
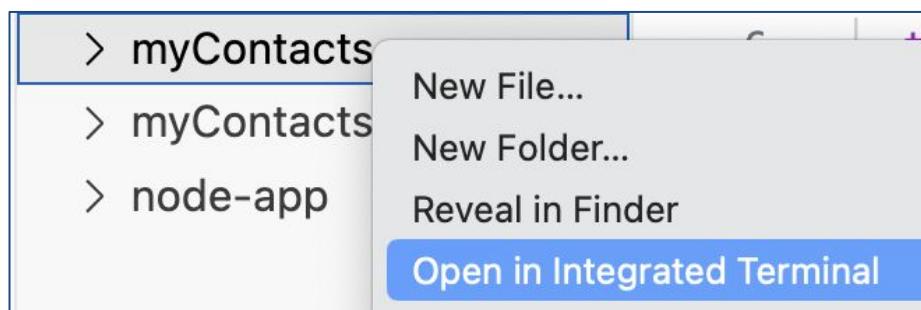
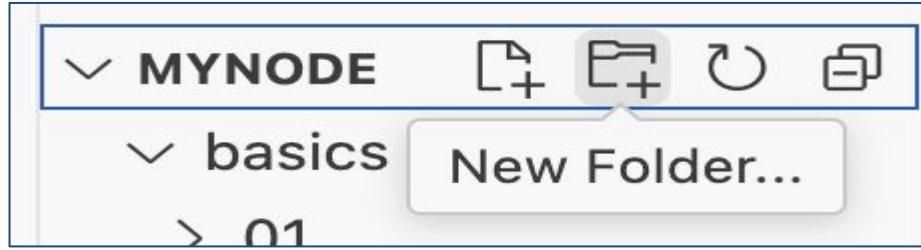


express

- 라우팅
 - http 모듈을 사용할 때는 if/switch문 사용해서 라우팅
 - express에서는 간단
- 미들웨어
 - 요청과 응답 사이에서 여러 가지 기능을 실행
 - 많은 패키지 제공
- 템플릿 엔진
 - data를 가지고 와서 html페이지를 만들 수 있음.(렌더링)
 - MVVM에서 view담당
- 정적인 파일 지원
 - CSS, JS, image 등과 같은 정적인 파일 처리 가능

express

● 프로젝트 생성



- alicia@Aliciaui-MacBookAir myContacts % npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults

See `npm help init` for definitive documentation on these fields and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and save it as a dependency in the package.json file.

Press ^C at any time to quit.

package name: (mycontacts)

version: (1.0.0)

description:

entry point: (index.js)

test command:

git repository:

keywords:

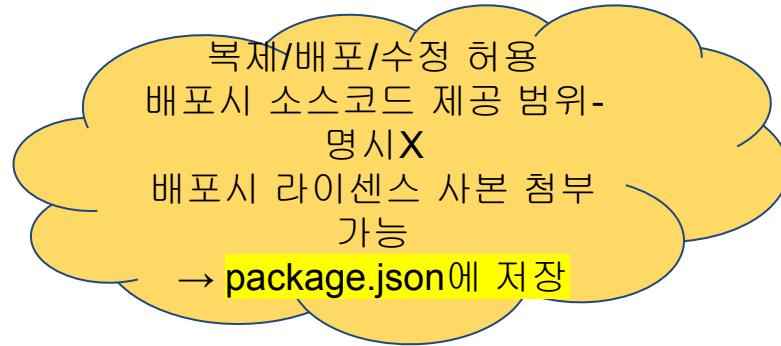
author:

license: (ISC)

About to write to /Users/alicia/Documents/myNode/myContacts/package.json:

```
{
  "name": "mycontacts",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": ""
}
```

● ISC license(Internet Systems Consortium)



{ package.json X

```

1  {
2    "name": "mycontacts",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    ▶ Debug
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "author": "",
11   "license": "ISC"
12 }
```

ISC License

ISC 라이선스는 Internet Systems Consortium(ISC)에 허용된 free Software license 로, ISC에서 개발한 OpenBSD베이스로 개발된 소프트웨어 릴리즈를 위해서 사용되는 라이선스이다.

- 버전 : 4-digit year
- 관리기관 : 4-digit year
- 라이선스 계열 : BSD
- 웹사이트 바로가기 : 4-digit year

주요내용	한글전문	영문전문	한글/영문비교	법률용어	개발템플릿	히스토리
토론						

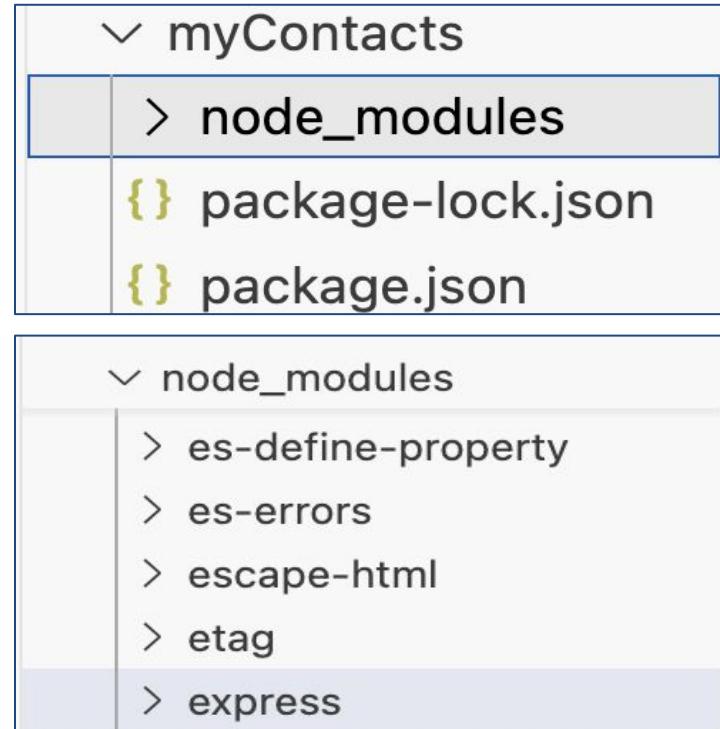
● 라이선스 주요내용

복제, 배포, 수정의 권한 허용	가능
배포시 라이센스 사본 첨부	가능
저작권 고지사항 또는 Attribution 고지사항 유지	가능
배포시 소스코드 제공의무(Reciprocity)와 범위	명시되어있지않음
조합저작물(Lager Work) 작성 및 타 라이선스 배포 허용	조건부 가능
수정 시 수정내용 고지	명시되어있지않음
명시적 특허 라이선스의 허용	가능
라이선서가 특허소송 제기 시 라이선스 종료	가능
이름, 상표, 상호에 대한 사용제한	가능
보증의 부인	가능
책임의 제한	명시되어있지않음

express

- express 설치

✖ alicia@Aliciaui-MacBookAir myContacts % npm i express



{ package.json }

```

1  {
2    "name": "mycontacts",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    ▷ Debug
7    "scripts": {
8      "test": "echo \\\"Error: no test specified\\\" && exit 1"
9    },
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "express": "^4.19.2"
14   }
  
```

express라이브러리 설치
 → node_modules/express 생성
 → package-lock.json 생성
 → package.json, package-lock.json에 dependencies 추가됨.

- **nodemon 설치**

```
④ alicia@Aliciaui-MacBookAir myContacts % npm i nodemon -g --save-dev
```

- 라이브서버 역할 - 변경사항 자동으로 서버 재시작하여 갱신
- -g : global하게 설치, 다른 폴더에서도 사용 가능하게 설치
- --save-dev : 개발코드를 배포할 때 이 라이브러리는 포함하지 않겠다고 설정, 개발할 때만 필요한 라이브러리라는 의미

express

- nodemon 설치

```
JS test.js ×
1 const http = require("node:http");
2
3 const server = http.createServer((req, res) => {
4   console.log("request received");
5 });
6
7 server.listen(3000);
8 console.log("server started");
9
10

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

○ alicia@Aliciaui-MacBookAir myContacts_ % nodemon test
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): **
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node test.js`
server started
request received



○ alicia@Aliciaui-MacBookAir myContacts_ % nodemon test
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): **
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node test.js`
server started
request received

express

- express server 생성

JS app-1.js ×

```

1  const express = require("express");
2  const app = express();
3
4  const port = 3000;
5
6  app.get("/", (req, res) => {
7    res.status(200);
8    res.send("Hello Node!");
9  });
10
11 app.listen(port, () => {
12   console.log(`#${port}번 포트에서 서버 실행 중`);
13 });

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

alicia@Aliciaui-MacBookAir results % nodemon app-1.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): **
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app-1.js`
3000번 포트에서 서버 실행 중

localhost:3000 × +

localhost:3000 ☆

Tasks kb KB 중요!JS엔진&작동 half club 키플링

Hello Node!

명령어 등록
→ “npm 등록한명령어” 실행

```

"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "nodemon app-1.js"
},

```

alicia@Aliciaui-MacBookAir results % npm start
> mycontacts_@1.0.0 start
> nodemon app-1.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): **
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app-1.js request.js`
Server is running on port 3000

express

- express 라우팅하기

기본형

```
app.METHOD(path, handler)
```

HTTP 모듈

```
if (method === "GET" && url === "/") {
```

익스프레스

```
app.get("/", ...)
```

비교

요청 방식: GET
요청 경로

```
app.get('/', (req, res) => {
  res.status(200);
  res.send("Hello Node!");
});
```

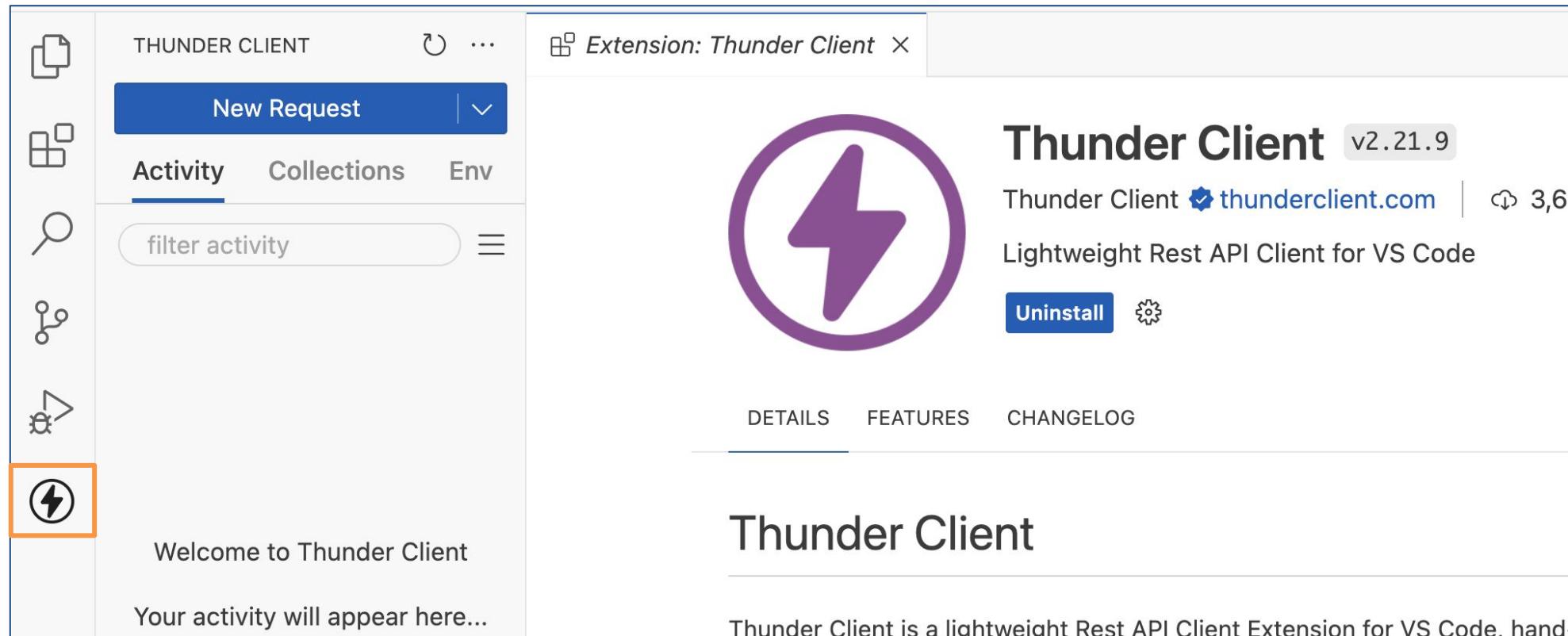
콜백 함수

메서드 체이닝

```
app.get("/", (req, res) => {
  res.status(200).send("Hello Node!");
})
```

- **Thunder Client(썬더 클라이언트)**

- VSCode에서 서버 실행 확인 가능
- 포스트맨과 유사



express

- Thunder Client(썬더 클라이언트)

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1
},
```

명령어 등록 삭제

alicia@Aliciaui-MacBookAir results % nodemon app-1

The screenshot shows the Thunder Client application interface. At the top, there are tabs for 'TC New Request' (selected), 'app-1.js', 'package.json', and '...', with a 'Send' button. Below this, a search bar contains 'localhost:3000/' with a dropdown menu showing 'GET'. The main area has tabs for 'Query' (selected), 'Headers 2', 'Auth', 'Body', 'Tests', and 'Pre Run'. Under 'Query', the status is 'Status: 200 OK', size is 'Size: 11 Bytes', and time is 'Time: 32 ms'. The response section shows the text '1 Hello Node!'. To the right, there are sections for 'Headers 2', 'Auth', 'Body', 'Tests', and 'Pre Run' with checkboxes for 'Accept' and 'User-Agent'. A 'Raw' checkbox is also present. Below these, a 'Response' section includes 'Headers', 'Cookies', 'Results', 'Docs', and 'Snippet'.

This part of the screenshot shows the 'Headers 2' tab selected in the Thunder Client interface. It lists 'Accept' and 'User-Agent' with their respective values. Below this, the 'Response' section is expanded, showing 'Headers', 'Cookies', 'Results', 'Docs', and 'Snippet'.

- Thunder Client(썬더 클라이언트)

JS app-2.js ×

```
1 const express = require("express");
2 const app = express();
3
4 const port = 3000;
5
6 app.get("/", (req, res) => {
7   res.status(200);
8   res.send("Hello Node!");
9 });
10
11 // 모든 연락처 가져오기
12 app.get("/contacts", (req, res) => {
13   res.status(200).send("Contacts Page");
14 });
15
16 // 새 연락처 추가하기
17 app.post("/contacts", (req, res) => {
18   res.status(201).send("Create Contacts");
19 });
20
21 app.listen(port, () => {
22   console.log(`#${port}번 포트에서 서버 실행 중`);
23 });
24
```

GET localhost:3000/contacts

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

Status: 200 OK Size: 13 Bytes Time: 24 ms Response

1 Contacts Page

POST localhost:3000/contacts

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

Status: 201 Created Size: 15 Bytes Time: 3 ms Response

1 Create Contacts

express

● 라우팅 파라메터

기본형 / 요청 URL/:id

/contacts/1
/contacts/10

요청 방식	요청 URL	역할
GET	/contacts/:id	id에 맞는 연락처 가져오기
PUT	/contacts/:id	id에 맞는 연락처 수정하기
DELETE	/contacts/:id	id에 맞는 연락처 삭제하기

The screenshot shows the Thunder Client interface. On the left, there's a sidebar with activity logs and a search bar. In the center, a modal window is open for a request to 'myNode' at 'localhost:3000/contacts/10'. The method dropdown shows 'DELETE' is selected. Below it, other methods like GET, POST, PUT, PATCH, HEAD, OPTIONS, PROPFIND, and CUSTOM are listed. The response section shows a status of '200 OK', a size of '25 Bytes', and a time of '5 ms'. The response body contains the message 'Delete Contact for ID: 10'.

express

● 라우팅 파라미터

```
// 연락처 상세보기
app.get("/contacts/:id", (req, res) => {
  res.status(200).send(`View Contact for ID: ${req.params.id}`);
});

// 연락처 수정하기
app.put("/contacts/:id", (req, res) => {
  res.status(200).send(`Update Contact for ID: ${req.params.id}`);
});

// 연락처 삭제하기
app.delete("/contacts/:id", (req, res) => {
  res.status(200).send(`Delete Contact for ID: ${req.params.id}`);
});
```

GET Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

<input type="checkbox"/> parameter	<input type="text"/>	<input type="text"/> value
------------------------------------	----------------------	----------------------------

Status: 200 OK Size: 23 Bytes Time: 12 ms Response

1 View Contact for ID: 10

PUT Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

<input type="checkbox"/> parameter	<input type="text"/>	<input type="text"/> value
------------------------------------	----------------------	----------------------------

Status: 200 OK Size: 25 Bytes Time: 5 ms Response

1 Update Contact for ID: 10 Copy

DELETE Send

Query Headers 2 Auth Body Tests Pre Run

Query Parameters

<input type="checkbox"/> parameter	<input type="text"/>	<input type="text"/> value
------------------------------------	----------------------	----------------------------

Status: 200 OK Size: 25 Bytes Time: 5 ms Response

1 Delete Contact for ID: 10

express

- express 요청 객체와 응답 객체

- 요청 객체의 주요 속성

속성	설명
req.body	서버로 POST 요청할 때 넘겨준 정보를 담고 있습니다. 예를 들어 로그인 버튼을 눌렀을 때 사용자의 아이디와 비밀번호의 값이 req.body에 들어 있습니다.
req.cookies	클라이언트에 저장된 쿠키 정보를 서버로 함께 넘겼을 경우 쿠키 정보를 담고 있습니다.
req.headers	서버로 요청을 보낼 때 같이 보낸 헤더 정보를 담고 있습니다.
req.params	URL 뒤에 라우트 파라미터가 포함되어 있을 경우 파라미터 정보를 담고 있습니다.
req.query	요청 URL에 포함된 질의 매개변수(쿼리, query)를 담고 있습니다. 예를 들어 검색 사이트에서 검색어를 입력하고 [검색] 버튼을 클릭했을 때 검색어와 관련된 질의 매개변수가 req.query에 담깁니다

express

- express 요청 객체와 응답 객체

- 응답 객체 주요 함수

함수	설명
res.download	파일을 내려받습니다.
res.end	응답 프로세스를 종료합니다.
res.json	JSON 응답을 전송합니다.
res.jsonp	JSONP 지원을 통해 JSON 응답을 전송합니다.
res.redirect	요청 경로를 재지정해서 강제 이동합니다.
res.render	뷰 템플릿을 화면에 렌더링합니다(10-2절에서 설명합니다.).
res.send	어떤 유형이든 res.send() 괄호 안의 내용을 전송합니다.
res.sendFile	지정한 경로의 파일을 읽어서 내용을 전송합니다.
res.sendStatus	상태 메시지와 함께 HTTP 상태 코드를 전송합니다.
res.status	응답의 상태 코드를 설정합니다.

express

- res.json()

```
app.get("/", (req, res) => {
  res.status(200);
  //res.send("Hello Node!");
  res.json({result : "ok"})
});
```

GET localhost:3000/ Send

Headers 2 Auth Body Tests Pre Run

HTTP Headers

Accept: */*

User-Agent: Thunder Client (https://www.thunderclient.com)

Status: 200 OK Size: 15 Bytes Time: 3 ms

Response

```
1  {
2    "result": "ok"
3 }
```

- res.sendFile()

```
index.html
1 <!DOCTYPE html>
2 <html lang="ko">
3   <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>연락처 관리기</title>
7     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css">
8     <link rel="stylesheet" href="css/style.css">
9     <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.css">
10   </head>
11   <body>
12     <!-- Header -->
13     <header class="border-bottom">
14       <div class="container ">
15         <nav>
16           <a href="/"><i class="fa-solid fa-address-book"></i> My Contacts</a>
17         </nav>
18       </div>
19     </header>
```

JS app-1.js

localhost:3000

```
const express = require("express");
const path = require("path")
const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.status(200);
  res.sendFile(__dirname + "/index.html");
});

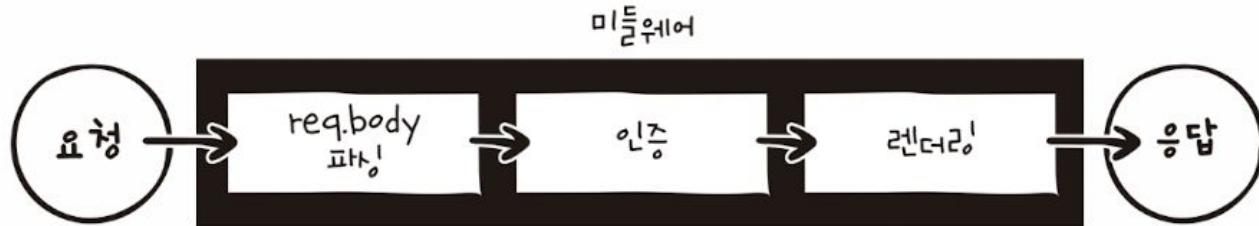
app.listen(port, () => {
  console.log(`$ {port}번 포트에서 서버 실행 중` );
});
```

이름	메일 주소	전화번호
	Username example@gmail.com	123456

Copy

● 미들웨어(middleware)

- 요청과 응답 중간에 있으면서 요청을 처리하거나 원하는 형태로 응답을 수정하는 함수
- 요청을 받아 응답하기 전까지 중간에 처리를 담당하는 함수



● 미들웨어의 주요 역할

속성	설명
요청 전처리	요청이 서버에 도착하기 전에 실행하는 작업을 담당합니다. 예를 들어 사용자 인증이나 품 내용 검증 등의 작업을 처리하죠.
라우팅 처리	지금까지 특정 URL로 들어온 요청을 미리 만들어 둔 함수(라우트 핸들러)로 연결했습니다. 이런 라우트 코드를 좀 더 읽기 쉽고 관리하기 쉽도록 모듈화하는 라우터 미들웨어도 있습니다.
응답 처리	서버에서 클라이언트로 응답을 보낼 때 자료를 적절한 형태로 변환하거나 오류를 처리하는 작업을 합니다.

express

- 미들웨어(middleware)

- 애플리케이션 레벨 미들웨어

```
app.get("/contacts", (req, res) => {...});
app.get("/contacts/:id", (req, res) => {...});
```

- 라우터 미들웨어

- 라우터 객체를 이용해서 라우트 코드를 좀더 쉽게 관리 가능
- 미들웨어는 등록해주어야 한다. → `app.use(등록할 미들웨어)`

기본형

```
app.use([경로], 미들웨어)
```

```
app.use("/", require("./routes/loginRoutes"));
```

경로도
가능

```
router
  .route("/contacts")
    .get((req, res) => {
      // 모든 연락처 가져오기
      res.status(200).send("Contacts Page");
    })
    .post((req, res) => {
      // 새 연락처 추가하기
      res.status(201).send("Create Contacts");
    });
}
```

express

● 미들웨어(middleware)

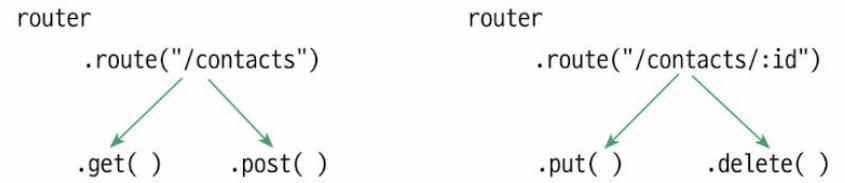
```
JS app-5.js ×
1  const express = require("express");
2  const app = express();
3  const router = express.Router();

4
5  const port = 3000;

7  app.get(req, res) => {
8    res.status(200).send("Hello Node!");
9  };

11 router
12   .route("/contacts")
13   .get(req, res) => {
14     // 모든 연락처 가져오기
15     res.status(200).send("Contacts Page");
16   }
17   .post(req, res) => {
18     // 새 연락처 추가하기
19     res.status(201).send("Create Contacts");
20   };
21
```

깔끔하고
가독성도
높다.



```

22
23 router
24   .route("/contacts/:id")
25   .get((req, res) => {
26     // 연락처 상세보기
27     res.status(200).send(`View Contact for ID: ${req.params.id}`);
28   })
29
30   .put((req, res) => {
31     // 연락처 수정하기
32     res.status(200).send(`Update Contact for ID: ${req.params.id}`);
33   })
34   .delete((req, res) => {
35     // 연락처 삭제하기
36     res.status(200).send(`Delete Contact for ID: ${req.params.id}`);
37   });
38
39 app.use(router);
40
41 app.listen(port, () => {
42   console.log(`${port}번 포트에서 서버 실행 중`);
43 });
  
```

express

● 미들웨어(middleware)

```
JS app-5.js
1 const express = require("express");
2 const app = express();
3 const router = express.Router();
4
5 // 삭제
6 const port = 3000;
7
8 app.get((req, res) => {
9   res.status(200).send("Hello Node!");
10 });
11
12 router
13 .route("/contacts")
14 .get((req, res) => {
15   // 모든 연락처 가져오기
16   res.status(200).send("Contacts Page");
17 })
18 .post((req, res) => {
19   // 새 연락처 추가하기
20   res.status(201).send("Create Contacts");
21 });
22
23 router
24 .route("/contacts/:id")
25 .get((req, res) => {
26   // 연락처 상세보기
27   res.status(200).send(`View Contact for ID: ${req.params.id}`);
28 })
29 .put((req, res) => {
30   // 연락처 수정하기
31   res.status(200).send(`Update Contact for ID: ${req.params.id}`);
32 })
33 .delete((req, res) => {
34   // 연락처 삭제하기
35   res.status(200).send(`Delete Contact for ID: ${req.params.id}`);
36 });
37
38 app.use(router);
39
40 // 삭제
41 listen(port, () => {
42   console.log(`${port}번 포트에서 서버 실행 중`);
43 });



```

JS contactRoutes-1.js

```
1 const express = require("express");
2 const router = express.Router();
3
4 router
5 .route("/")
6 // 모든 연락처 가져오기
7 .get((req, res) => {
8   res.status(200).send("Hello Node!");
9 })
10 .post((req, res) => {
11   // 새 연락처 추가하기
12   res.status(201).send("Create Contacts");
13 });
14
15 router
16 .route("/:id")
17 .get((req, res) => {
18   // 연락처 상세보기
19   res.status(200).send(`View Contact for ID: ${req.params.id}`);
20 })
21 .put((req, res) => {
22   // 연락처 수정하기
23   res.status(200).send(`Update Contact for ID: ${req.params.id}`);
24 })
25 .delete((req, res) => {
26   // 연락처 삭제하기
27   res.status(200).send(`Delete Contact for ID: ${req.params.id}`);
28 });
29
30 module.exports = router;
```

app.use("/contacts") 한 경우,
라우팅 정의 파일에서는
contacts를 제거해줌.

JS app-6.js

```
1 const express = require("express");
2 const app = express();
3
4 const port = 3000;
5
6 app.get("/", (req, res) => {
7   res.status(200).send("Hello Node!");
8 });
9
10 app.use("/contacts", require("./routes/contactRoutes"));
11
12 app.listen(port, () => {
13   console.log(`${port}번 포트에서 서버 실행 중`);
14 });
```

라우팅 파일 생성 가능
→ 외부연결해서 사용 가능

- **Http Routing**

- http 요청과 응답
- http 서버 생성
- 라우팅 - 요청에 대한 rendering

- **sync vs. async**

- 동기와 비동기 비교
- 이벤트 루프
- 노드의 비동기 패턴

- **express**

- 프레임워크
- 라우팅 설정
- 요청에 대한 렌더링 처리
- 미들웨어