




2조 발표자료

쿠키/세션, Node.js(라우터), fetch(HTTP Request)



쿠키, 세션

HTTP 프로토콜 속성

서버 자원 절약 위해,

비연결성 connectionless

클라이언트가 서버에게 요청한 후
응답 받으면 연결 끊어버림

무상태성 stateless

연결을 끊는 순간 클라이언트와 서버의
통신이 끝나며 상태 정보 유지하지 않음

문제점 예시 – 페이지 이동할 때마다 계속 로그인

“보완 위해 쿠키 등장”

쿠키

쿠키란? “클라이언트에 저장되는 작은 크기의 문자열(최대 4KB)”

쿠키의 3가지 용도

- 인증 : 로그인, 닉네임, 접속 시간, 장바구니 등
- 개인화 : 사용자마다 다르게 적절한 페이지 보여주기
- 방문자들의 상태 체크 : 사용자의 행동, 패턴 분석 & 기록

주로 서버에서 사용 : 서버가 정한 기준에 따라 유효기간 있다.

같은 도메인에서만 전송 가능

= 유튜브에서 보낸 쿠키는 유튜브에서만 사용 가능

쿠키

쿠키의 동작 방식

클라이언트가 페이지 요청 → 서버에서 쿠키 생성 → HTTP 헤더에 쿠키 포함 시켜 응답
→ 브라우저 종료되어도 쿠키 만료기간동안 클라이언트에서 보관

= 같은 요청이면 HTTP 헤더에 쿠키 함께 보냄

if 서버에서 쿠키 읽어 이전 상태 정보를 변경할 필요가 있을 때,

쿠키 업데이트해 변경된 쿠키를 HTTP 헤더에 포함시켜 응답

쿠키

쿠키 사용 예시

방문 사이트 로그인시 '아이디와 비밀번호를 저장하시겠습니까?' 묻기

쇼핑몰 장바구니 기능

자동로그인, 팝업창에서 '오늘 더이상 이 창을 보지 않음' 체크

세션

세션이란?

“쿠키를 기반으로, 클라이언트 구분하기 위해 세션 ID를 부여하며

웹 브라우저가 서버에 접속해서 브라우저 종료할 때까지 인증상태를 유지한다”

세션 ID란?

각 클라이언트에게 부여하는 고유 ID로,

클라이언트 구분해 클라이언트 요구에 맞는 서비스 제공하기 위함

세션

세션 동작 방식

클라이언트가 서버 접속 시 세션ID 발급받음

→ 클라이언트는 세션ID에 대해 쿠키를 사용하여 저장하고 갖고있음

→ 클라이언트는 서버에 요청할 때 이 쿠키의 세션ID를 같이 서버 전달해 요청

→ 서버는 세션ID를 전달받아 별다른 작업 없이 세션ID로 세션에 있는 클라이언트 정보 가져와 사용

→ 클라이언트 정보 가지고 서버 요청 처리해 클라이언트에게 응답

세션 사용 예시 로그인 같이 보안상 중요한 작업 수행할 때 사용

쿠키, 세션 차이

쿠키		세션
브라우저에 저장	※사용자 정보 파일※	서버측에서 관리
빠름	※ 요청 속도	※ 느림
		→ 세션은 서버의 처리가 필요
나쁨	※ 보안	※ 좋음
		→ 사용자에게 대한 정보를 서버에 두기 때문
		but 사용자 많아질수록 서버 메모리차지 ↑
		= 동접자 많으면 서버에 과부하 → 성능저하요인

ps. 세션도 만료시간 정할 수 있지만 브라우저 종료되면 만료시간 상관없이 삭제

Node.js 특징/ 장점

Node.js

웹 브라우저에서
여러 요소 동적으로 움직이기
& 외부와 통신하기

Node.js란? “웹 브라우저 밖에서도 자바스크립트 실행되도록 해주는 것”

Front-end
HTML
CSS
Javascript

기존 Back-end
Java
PHP
Python

Node.js가 가능하게 함



Back-end
Javascript
Java
PHP
Python

“Javascript만 알아도 웹 제작이 가능”

Node.js

작동 방법

이벤트 기반 비동기 방식으로 작동 (single-Thread)

○ 혼자서 작업이 끝나는 알림을 기다리는 것 (X 여러 일을 동시에 할 때 사람을 복제해 여러 작업 수행)

⇒ 메모리 사용량과 시스템 리소스 사용량에는 변화가 거의 없다.

Node.js

장점

- npm(node package manager)을 통한 다양한 모듈을 제공해 자신이 필요한 라이브러리와 패키지
검색해 설치 & 사용 가능
= 효율성 좋음
- 메모리 사용량과 시스템 리소스 사용량에는 변화가 거의 없어서,
대규모 네트워크 프로그램 개발이 가능하다.

Node.js

단점

single-Thread

- 알림을 기다리는 사람이 파업하는 순간 프로그램 전체에 문제를 발생시킨다.
- 해당 작업이 끝난 후 수행해야 하는 일이 여러개 일 경우 콜백함수의 높에 빠진다.
- 하나의 작업 자체에 많은 시간이 걸리는 웹서비스에는 부적합하다.
- 코드가 수행되어야 에러를 알 수 있고, 에러 날 경우 프로세스가 내려가기 때문에 테스트가 중요하다.
- C++로 개발된 서버 어플리케이션보다는 느리다.

node.js는 스크립트파일(js)
읽어서 수행하기 때문

Node.js

Node.js가 어울리는 웹 서비스

- 간단한 로직
- 대규모 네트워크 프로그램
- 빠른 응답시간 요구
- 빠른 개발 요구

Node.js가 어울리지 않는 웹 서비스

- 단일 처리가 오래 걸리는 경우
- 업무 복잡도와 난이도가 높은 경우

Node.js
get, post, route

Node.js express

Express란?

Node.js를 위한 웹 프레임워크 중 하나로 ,자유-오픈 소스 소프트웨어 입니다. 웹 어플리케이션, API 개발을 위해 설계되었습니다. Node.js의 사실상의 표준 서버 프레임워크로 불립니다.

Express의 개념

- Express.js는 Node.js를 위한 빠르고 간편한 웹 프레임워크입니다.
- 다양한 웹프레임워크가 있지만 현재까지 가장 많이 사용하는 것이 바로 익스프레스 엔진입니다.

Node.js express

왜 Express를 사용해야 하지?

Express는 프레임워크이므로 웹 애플리케이션을 만들기 위한 각종 라이브러리와 미들웨어 등이 내장돼 있어 개발하기 편하고, 수많은 개발자들에게 개발 규칙을 강제하여 코드 및 구조의 통일성을 향상시킬 수 있다.

= 프레임워크 도입의 가장 큰 장점

Node.js express

Http 내장 모듈 VS Express

```
//http 내장모듈을 사용한 웹서버 띄우기
const http = require('http');

http.createServer(function(request, response){
  response.writeHead(200, {'Content-Type':'text/html'});
  response.write('Hello http webserver!')
  response.end();
}).listen(52773, function(){
  console.log("server running http://127.0.0.1:52773/");
});
```

Http 내장 모듈로 웹 서버 띄어보기

```
//express 웹프레임워크를 이용한 서버 띄우기 실습
const express = require('express');
const app = express;
const port = 3000;

app.length('/', (req, res) => {
  res.send('Hello Express!!!!!!!')
});

app.listen(port, () => {
  console.log('Express server listen..')
});
```

Express로 웹 서버 띄워보기

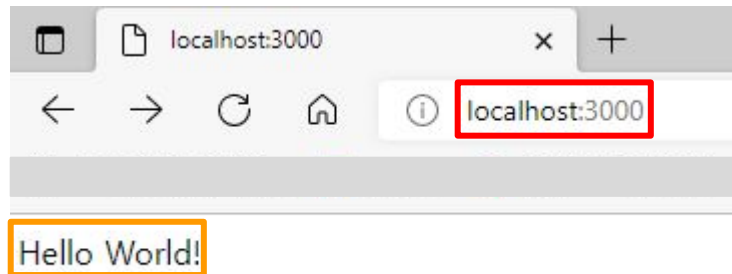
Node.js express_실행

```
JS index.js x
JS index.js > ...
1  const express = require('express');
2  const app = express();
3
4  app.get('/', function (req, res) {
5    res.send('Hello World!');
6  });
7
8  var server = app.listen(3000, function () {
9    var host = server.address().address;
10   var port = server.address().port;
11
12   console.log('Server is working : PORT - ',port);
13 });
14
```

터미널 실행

```
PS C:\Users\ > node index.js
Server is working : PORT - 3000
```

출력 화면



A screenshot of a web browser window. The address bar shows 'localhost:3000' with a red box around it. The page content displays 'Hello World!' with an orange box around it.

Node.js express_router

라우터란?

“클라이언트의 요청 경로(path)를 보고 이 요청을 처리할 수 있는 곳으로 기능을 전달해주는 역할”

라우팅

URL(경로) 및 특정한 HTTP 요청 메소드(POST, GET)의 클라이언트 요청에 응답하는 방법을 결정

각 라우트는 하나 이상의 핸들러 함수를 가질 수 있고 라우트가 일치할 때 실행

Node.js express_미들웨어

```
var express = require('express');  
var app = express();
```

```
app.get('/', function(req, res, next) {  
  next();  
})
```

```
app.listen(3000);
```

미들웨어 기능이 적용되는 HTTP 메소드

미들웨어 기능이 적용되는 PATH(경로) 메소드

HANDLER : 미들웨어 기능

next : 미들웨어 함수에 대한 콜백 인수

res : 미들웨어 함수에 대한 HTTP 응답 인수

req : 미들웨어 기능에 대한 HTTP 요청 인수

Node.js express_GET

GET이란?

“클라이언트에서 서버로 어떠한 리소스로 부터 정보를 요청하기 위해 사용되는 메소드”

```
app.get( PATH(경로), HANDLER )
```

GET 방식

URL 끝에 “?”를 붙이고 그 다음 변수명1=값1&변수명2=값2... 형식으로 작성

www.example.com/show?name1=value1&name2=value2

Node.js express_GET

GET특징

- GET 요청은 캐시가 가능하다.
- GET 요청은 브라우저 히스토리에 남는다.
- GET 요청은 북마크 될 수 있다.
- GET 요청은 길이 제한이 있다.
- GET 요청은 중요한 정보를 다루면 안된다. (보안)
- GET은 데이터를 요청할때만 사용 된다.

Node.js express_POST

POST란?

“클라이언트에서 서버로 리소스를 생성하거나 업데이트하기 위해 데이터를 보낼 때 사용되는 메소드”

```
app.post( PATH(경로), HANDLER )
```

POST 방식

전송할 데이터를 HTTP메세지 body 부분에 담아서 서버로 보낸다. 전송은 보통 HTML form을 통해 서버로 전송

(body의 타입은 Content-Type 헤더에 따라 결정)

Node.js express_POST

POST특징

- POST 요청은 캐시되지 않는다.
- POST 요청은 브라우저 히스토리에 남지 않는다.
- POST 요청은 북마크 되지 않는다.
- POST 요청은 데이터 길이에 제한이 없다.

Node.js express_GET & POST

	GET	POST
사용목적	서버의 리소스에 데이터를 요청	서버의 리소스를 새로 생성하거나 업데이트할 때 사용
요청에 body 유무	HTTP 메시지에 body 존재 안함	HTTP 메시지에 body 존재
멱등성	O	X

Node.js express_GET & POST 방식의 method

```
var express = require('express');
var app = express();

// GET method route
app.get('/', function(req, res) {
  res.send('hello world');
});

// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

Node.js express_route

```
var express = require('express');  
var app = express();
```

```
app.route('/')  
  .get(function(req, res) {  
    res.send('hi balmostory');  
  })  
  .post(function(req, res) {  
    res.send('hi balmostory');  
  })  
  
app.listen(3000, () => {  
  console.log('listen t0 3000')  
})
```

← → ↻ ⓘ localhost:3000

hello balmostory

Node.js express_Response

Method

- `res.download()` ▷ download를 실행시킵니다.
- `res.end()` ▷ response 작업을 마칩니다.
- `res.json()` ▷ Json을 보냅니다.
- `res.jsonp()` ▷ Json을 보내는데, jsonp support가 되는 json을 보냅니다.
- `res.redirect()` ▷ 페이지를 다른 곳으로 이동 시킵니다.
- `res.render()` ▷ 렌더링 엔진에 따라서 views를 해석하여 보냅니다.
- `res.send()` ▷ 다양한 타입의 내용을 보냅니다. boolean, string 등등..
- `res.sendFile()` ▷ 파일을 보냅니다.
- `res.sendStatus()` ▷ http status code를 보냅니다. 기본적으로 200은 ok, 404는 not found, 304는 not modified, 500은 internal error입니다.

Node.js express_router

라우터 사용하기

1. 라우터 객체 참조
2. 라우팅 함수 등록
3. 라우터 객체를 app 객체에 등록

```
// 라우터 객체 참조
```

```
const router = express.Router();
```

```
// 라우팅 함수 등록
```

```
router.route('/process/login').get(...);
```

```
router.route('/process/login').post(...);
```

```
...
```

```
// 라우터 객체를 app 객체에 등록
```

```
app.use('/', router);
```

Node.js express_router(express 라우터 분리)

router/ router_1.js

main.js

JS main.js X

router > express_practice > JS main.js > ...

```
1 var express = require('express');
2 var app = express();
3 var router_1 = require('./router/router_1');
4
5 app.use('/aboutrouter', router_1);
6
7 app.listen(3000, () => {
8   console.log('listen to 3000');
9 });
```

JS main.js

JS router_1.js X

router > express_practice > router > JS router_1.js > ...

```
1 var express = require('express');
2 var router = express.Router();
3
4 router.use(function(req, res, next) {
5   next();
6 });
7
8 router.get('/', function(req, res) {
9   res.send('hi1');
10 });
11
12 router.get('/about', function(req, res) {
13   res.send('hi2');
14 });
15
16 module.exports = router;
```


JS main.js X

router > express_practice > JS main.js > ...

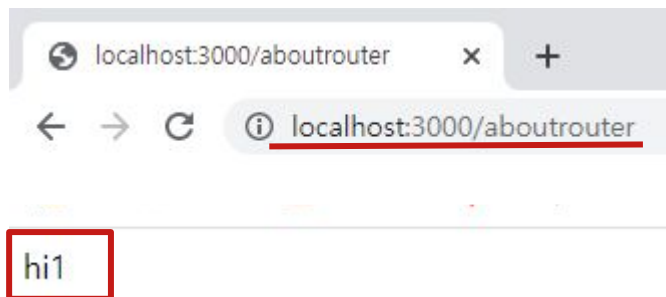
```
1 var express = require('express');
2 var app = express();
3 var router_1 = require('./router/router_1');
4
5 app.use('/aboutrouter', router_1);
6
7 app.listen(3000, () => {
8   console.log('listen to 3000');
9 });
```

JS main.js

JS router_1.js X

router > express_practice > router > JS router_1.js > ...

```
1 var express = require('express');
2 var router = express.Router();
3
4 router.use(function(req, res, next) {
5   next();
6 });
7
8 router.get('/', function(req, res) {
9   res.send('hi1');
10 });
```

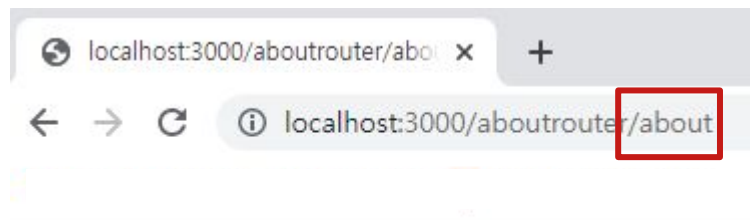


JS main.js X

router > express_practice > JS main.js > ...

```
1 var express = require('express');
2 var app = express();
3 var router_1 = require('./router/router_1');
4
5 app.use('/aboutrouter', router_1);
6
7 app.listen(3000, () => {
8   console.log('listen to 3000');
9 });
```

```
7
8 router.get('/', function(req, res) {
9   res.send('hi1');
10 });
11
12 router.get('/about', function(req, res) {
13   res.send('hi2');
14 });
15
16 module.exports = router;
```



hi2



HTTP

HTTP(Hyper Text Transfer Protocol)

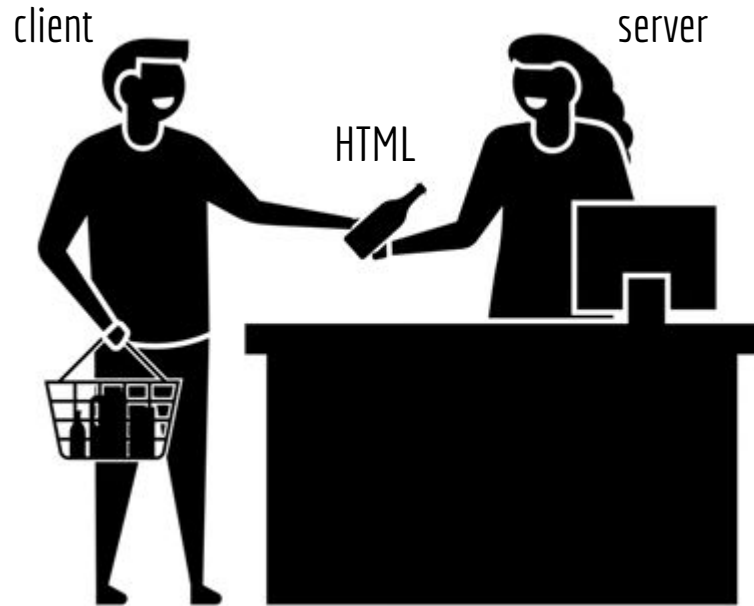


HTTP란? “웹 서버와 웹 브라우저가 통신하기 위해 사용하는 규약”

성능, 보안 안정성을 보장해준다.

HTML 뿐만 아니라 image, video 등 멀티미디어 파일을 전송할 수 있게 해주는 중요한 프로토콜.

HTTP(Hyper Text Transfer Protocol)



HTTP(Hyper Text Transfer Protocol)

▼ Request Headers view parsed

```
GET /1.html HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/67.0.3396.99 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://localhost:8080/
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: _ga=GA1.1.1634843349.1522338519; Webstorm-507ad0d5=f5a8c12c-1e2b-41c6-829d-9199fffff7a8; __tawkuuid=e::localhost::aUpdJw2SZAP7bJQJpWE4age+Vii9uw02z7Y4pohYMSn1awAbP0qmt24/u6gKYd6q::2; Tawk_57a72994c11fe69b0bd8fa90=vs23.tawk.to::0; io=NVLBXbhQuFK_kFL5AAAG
```

HTTP(Hyper Text Transfer Protocol)



HTTP(Hyper Text Transfer Protocol)

- GET** ▷ 리소스를 요청함.
- HEAD** ▷ GET처럼 응답을 요구하지만 응답 본문을 포함하지 않음.
- POST** ▷ 메시지 바디로 서버에 데이터를 전송함.
- PUT** ▷ 리소스가 있으면 대체하고 없으면 생성한다.
- PATCH** ▷ PUT처럼 리소스를 수정하나 데이터의 일부만 수정함.
- DELETE** ▷ 리소스를 삭제함.
- CONNECT** ▷ 목표로 하는 서버로 터널을 연결함.
- OPTION** ▷ 목적 리소스의 통신을 설정함.
- TRACE** ▷ 목적 리소스의 경로를 따라 메시지 loop-back 테스트를 함.

HTTP(Hyper Text Transfer Protocol)

1. 안전 — 메소드를 호출해도 리소스를 변경하지 않는다는 뜻이다.
2. 멱등 — 메소드를 계속 호출해도 결과가 똑같다는 뜻이다.
3. 캐시 가능 — 캐시 가능하다는 말은 말 그대로 캐싱을 해서 데이터를 효율적으로 가져올 수 있다는 뜻이다.

HTTP 메소드 ◆	RFC ◆	요청에 Body가 있음 ◆	응답에 Body가 있음 ◆	안전 ◆	멱등(Idempotent) ◆	캐시 가능 ◆
GET	RFC 7231 🔗	아니요	예	예	예	예
HEAD	RFC 7231 🔗	아니요	아니요	예	예	예
POST	RFC 7231 🔗	예	예	아니요	아니요	예
PUT	RFC 7231 🔗	예	예	아니요	예	아니요
DELETE	RFC 7231 🔗	아니요	예	아니요	예	아니요
CONNECT	RFC 7231 🔗	예	예	아니요	아니요	아니요
OPTIONS	RFC 7231 🔗	선택 사항	예	예	예	아니요
TRACE	RFC 7231 🔗	아니요	예	예	예	아니요
PATCH	RFC 5789 🔗	예	예	아니요	아니요	예

HTTP(Hyper Text Transfer Protocol)

상태코드

- 1xx (Informational) : 요청이 수신되어 처리중
- 2xx (Successful) : 요청 정상 처리
- 3xx (Redirection) : 요청을 완료하려면 추가 행동이 필요
- 4xx (Client Error) : 클라이언트 오류, 잘못된 문법등으로 서버가 요청을 수행할 수 없음
- 5xx (Server Error) : 서버 오류, 서버가 정상 요청을 처리하지 못함

예시)

401 Unauthorized

```
const AUTH_ERROR = { message: 'Authentication Error' };

export const isAuth = async (req, res, next) => {
  const authHeader = req.get('Authorization');
  if (!(authHeader && authHeader.startsWith('Bearer '))) {
    return res.status(401).json(AUTH_ERROR);
  }
}
```

200 Okay

```
export async function getTweets(req, res) {
  const username = req.query.username;
  const data = await (username
    ? tweetRepository.getAllByUsername(username)
    : tweetRepository.getAll());
  res.status(200).json(data);
}
```

404 not found & 500 server error

```
app.use((req, res, next) => {
  res.sendStatus(404);
});

app.use((error, req, res, next) => {
  console.error(error);
  res.sendStatus(500);
});
```

REST API & FETCH

Rest(Representational State Transfer)란?

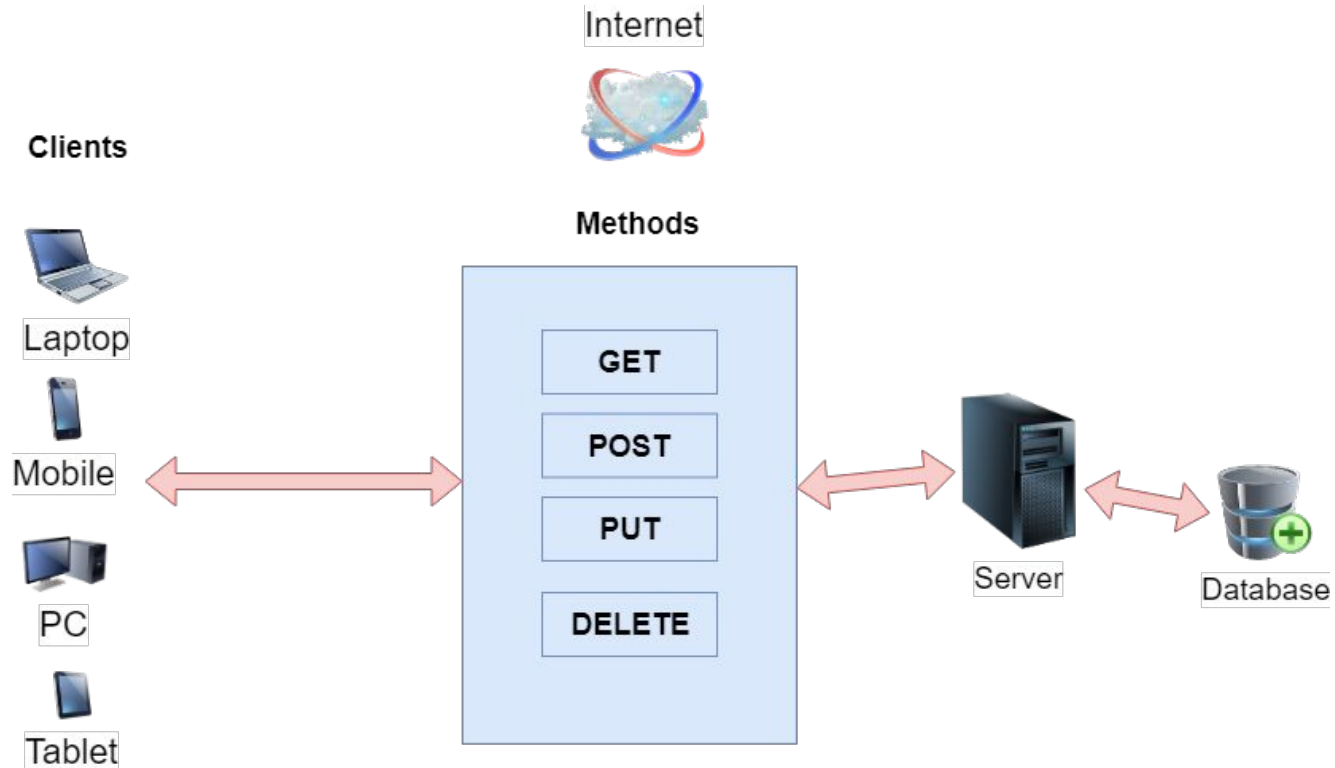
“자원의 이름을 구분하여 상태 정보에 접근하는 것”

Rest API는 4가지의 메소드를 가지고 있다. (get, post, put, delete)

Rest API = HTTP method + URL

Fetch 란? “원격 API를 간편하게 호출하는 함수”

REST API & Fetch API



Rest API (path parameter & query string)

사용자 ID가 2인 정보에 접근하려한다. 어떤 것을 사용해야 할까?

Query

/user/id=2

Path

/user/2

Rest API (path parameter & query string)

path

어떤 리소스에 대해
식별하기 위해 사용된다.

query

정렬이나 필터링할 때 주로 사용된다.

예시)

```
fetch(url, {  
  method: 'POST', // *GET, POST, PUT, DELETE, etc.  
  mode: 'cors', // no-cors, cors, *same-origin  
  cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached  
  credentials: 'same-origin', // include, *same-origin, omit  
  headers: {  
    'Content-Type': 'application/json',  
    // 'Content-Type': 'application/x-www-form-urlencoded',  
  },  
  redirect: 'follow', // manual, *follow, error  
  referrer: 'no-referrer', // no-referrer, *client  
  body: JSON.stringify(data), // body data type must match "Content-Type" header  
})
```

HTTP URL와 HTTP Method가 명시가 되었음

감사합니다