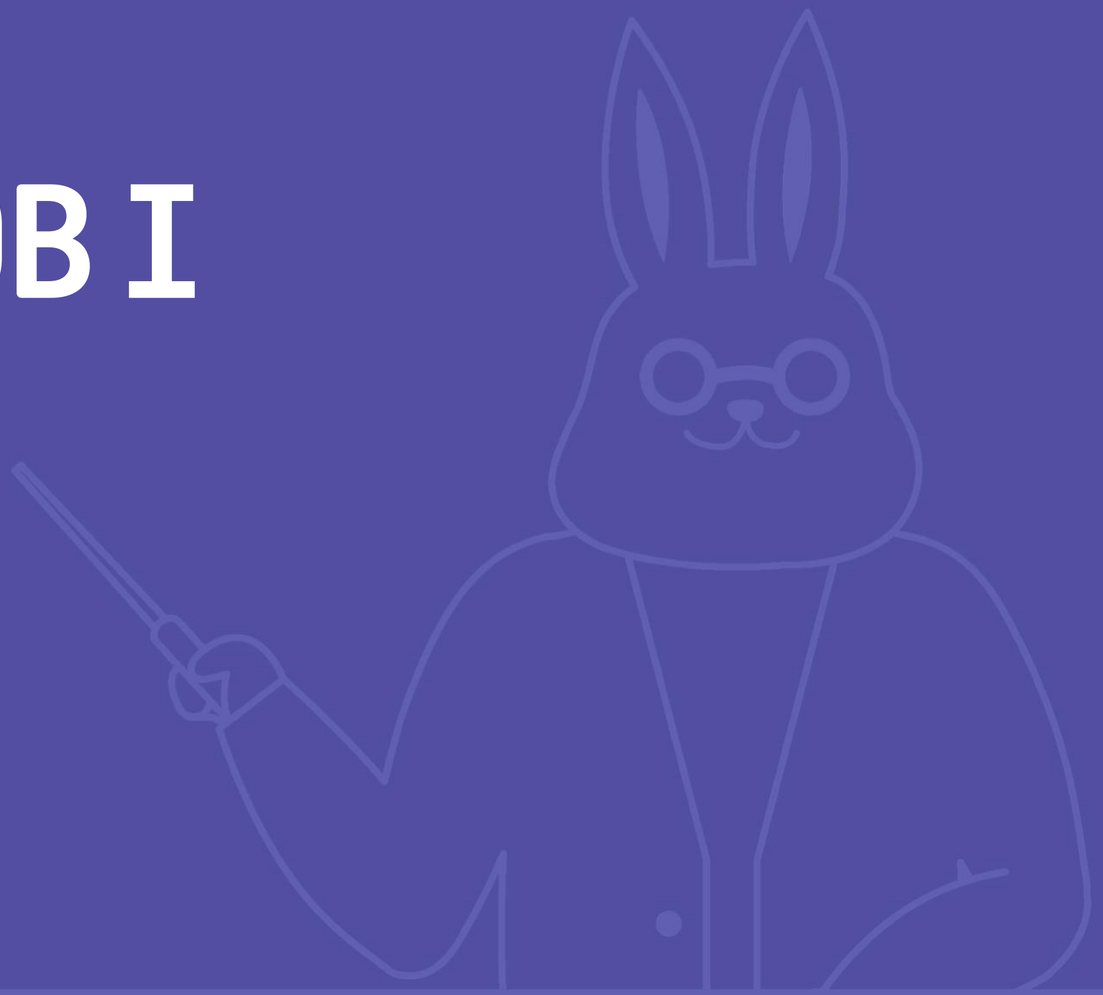
Node.js와 MongoDB I

03 웹과 Express.js



/* elice */

목차

🎯 수강목표

1. 웹 이해하기

웹에 대해 이해하고 웹 서비스의 다양한 동작 방식들을 학습합니다.

2. Express.js 사용해 보기

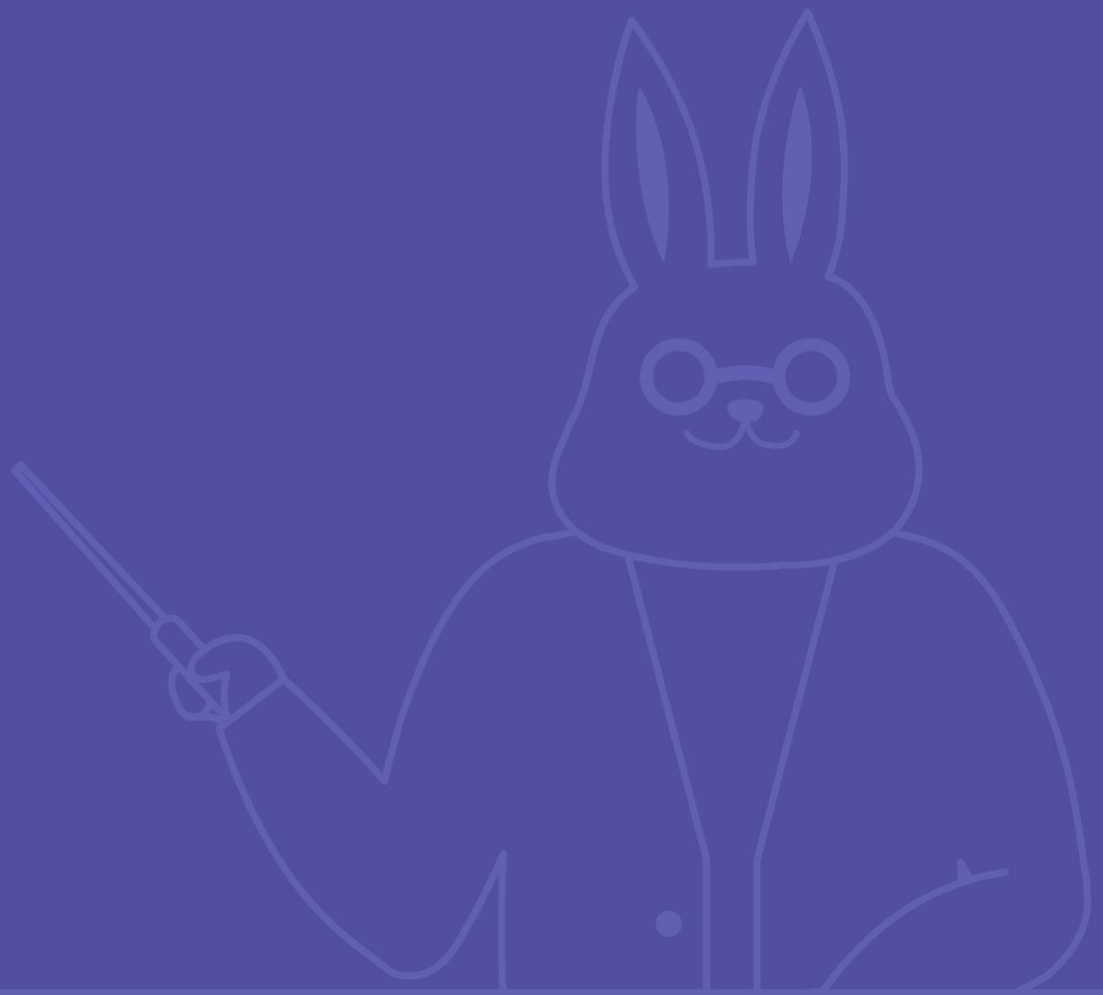
Node.js의 웹 프레임워크인 Express.js를 프로젝트에 적용하고 사용하는 방법에 대해 학습합니다.

3. Express.js 이해하기

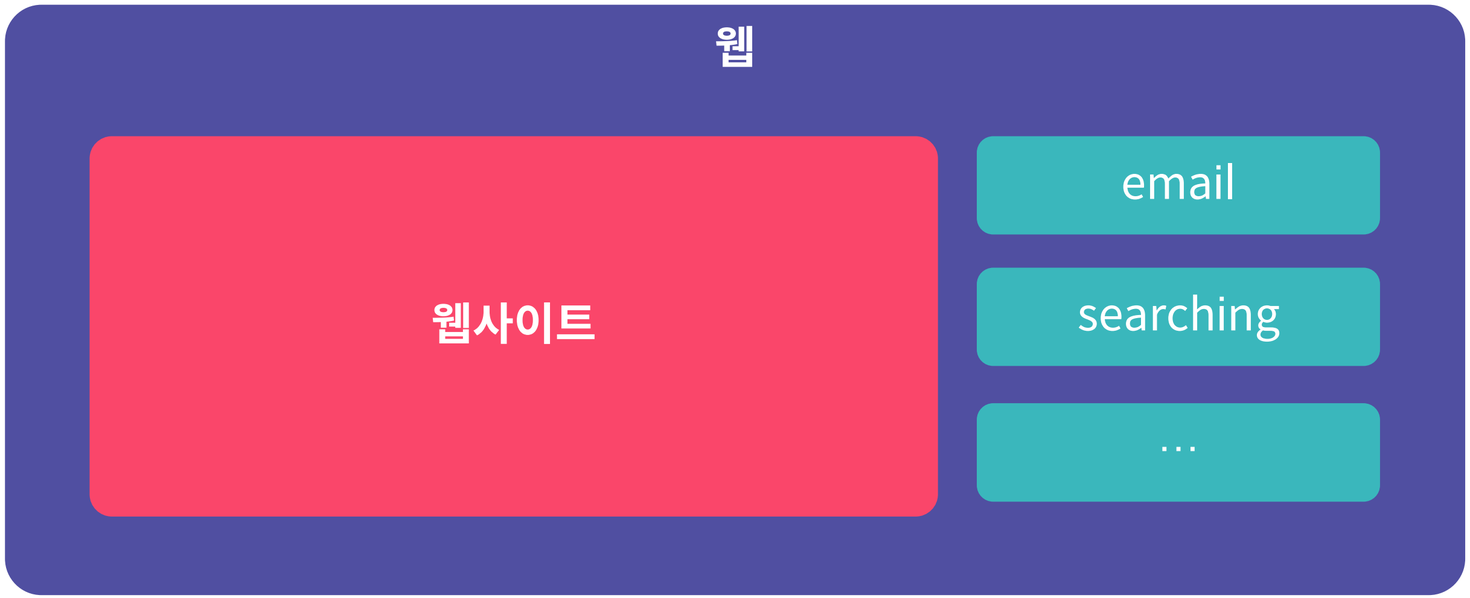
Express.js의 구조와 동작 방식에 대해 학습합니다.

01

웹의 이해



웹이란?



사전적 의미 - World Wide Web, 인터넷상에서 동작하는 **모든 서비스**

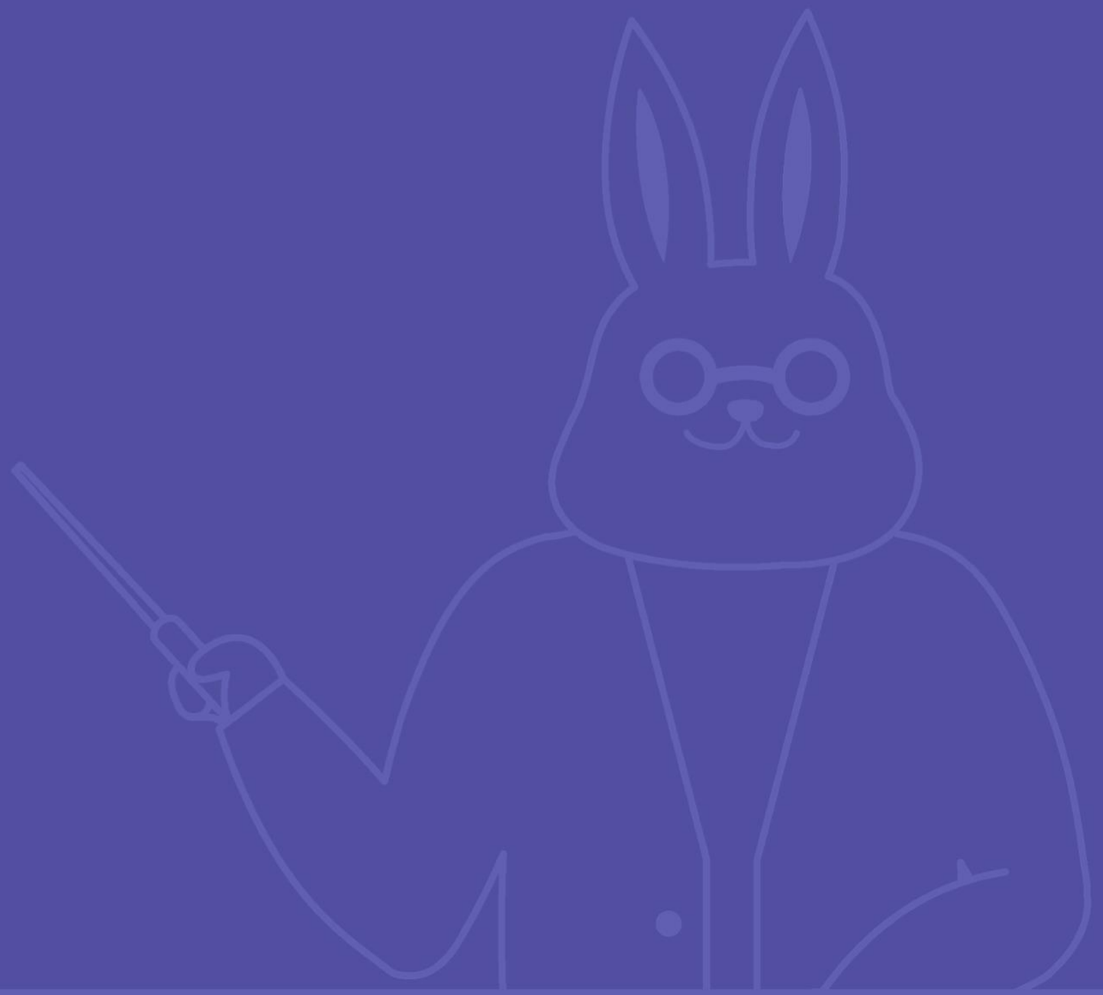
일반적 의미 - **웹 브라우저**로 접속해서 이용하는 서비스, **웹 사이트**

웹이란?

본 수업에서의 **웹 = 웹사이트**

02

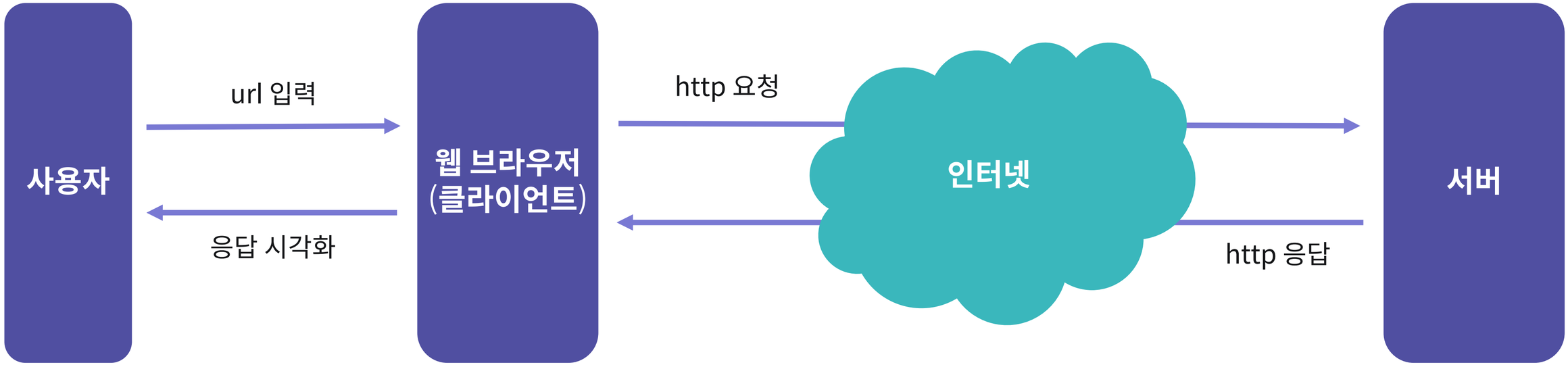
웹 서비스 동작 방식



웹 서비스 동작 방식

웹 서비스는 기본적으로 **HTTP 요청과 응답의 반복**으로 이루어짐
HTTP 요청은 사용자가 **어떤 데이터가 필요한지**를 서버에게 알리는 역할
HTTP 응답은 HTTP **요청에 해당하는 적절한 데이터**를 전달하는 역할

웹 서비스 동작 방식



1. 브라우저가 인터넷을 통해 **HTTP 요청을 서버에 전달**
2. 서버는 사용자의 **HTTP 응답을 브라우저로 전송**
3. 브라우저는 HTTP 응답을 **사용자에게 적절한 화면으로 노출**

HTTP 요청 예시

```
GET / HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:93.0)
Accept: text/html
Accept-Language: ko-KR
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

HTTP 요청은 사용자가 **어떤 사용자가**, **어떤 데이터를 필요로 하는지** 등을 담고 있음

HTTP 응답 예시

전송된 데이터

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Date: Mon, 25 Oct 2021 14:10:35 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

사용자가 요청한 데이터

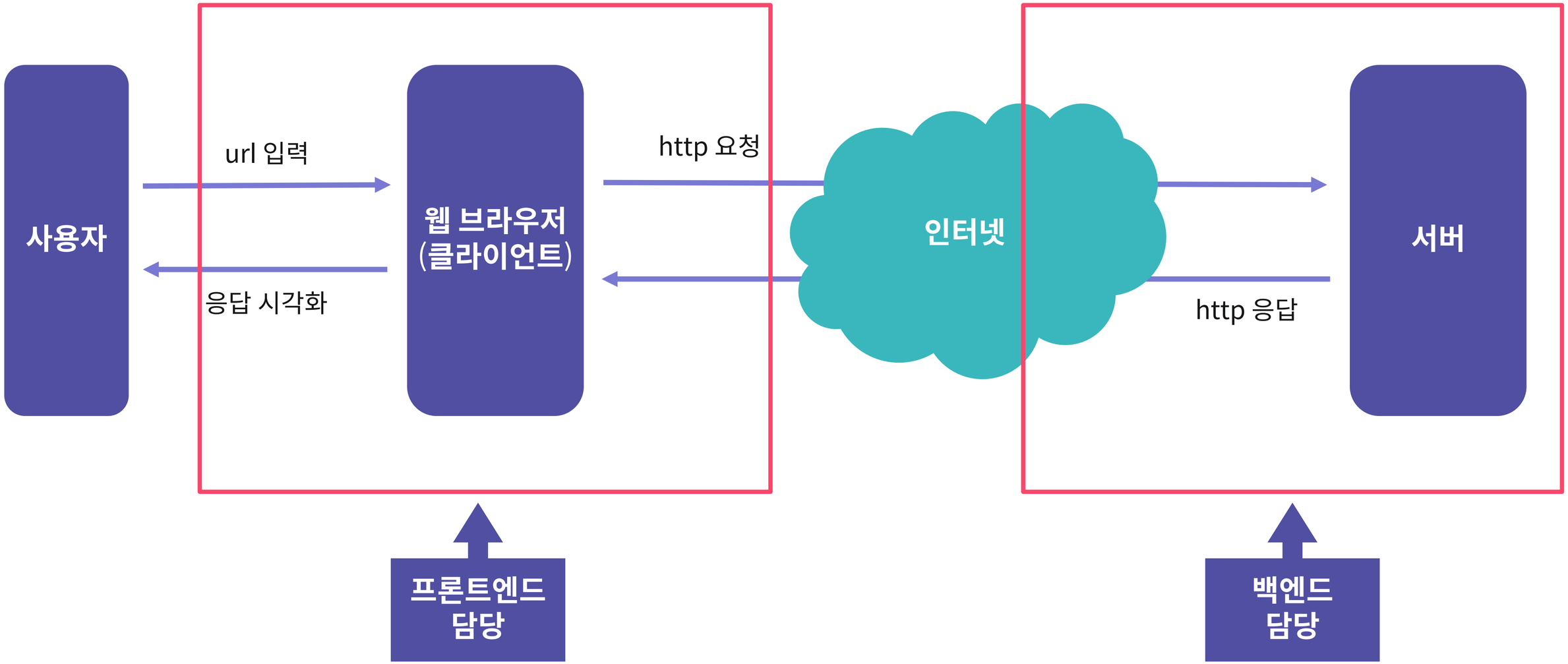
```
<html>
<body>
<h1>Hello, World!</h1>
</body>
</html>
```

HTTP 응답은 사용자가 **요청한 데이터**와, **어떤 데이터가 전송되는지** 등을 담고 있음

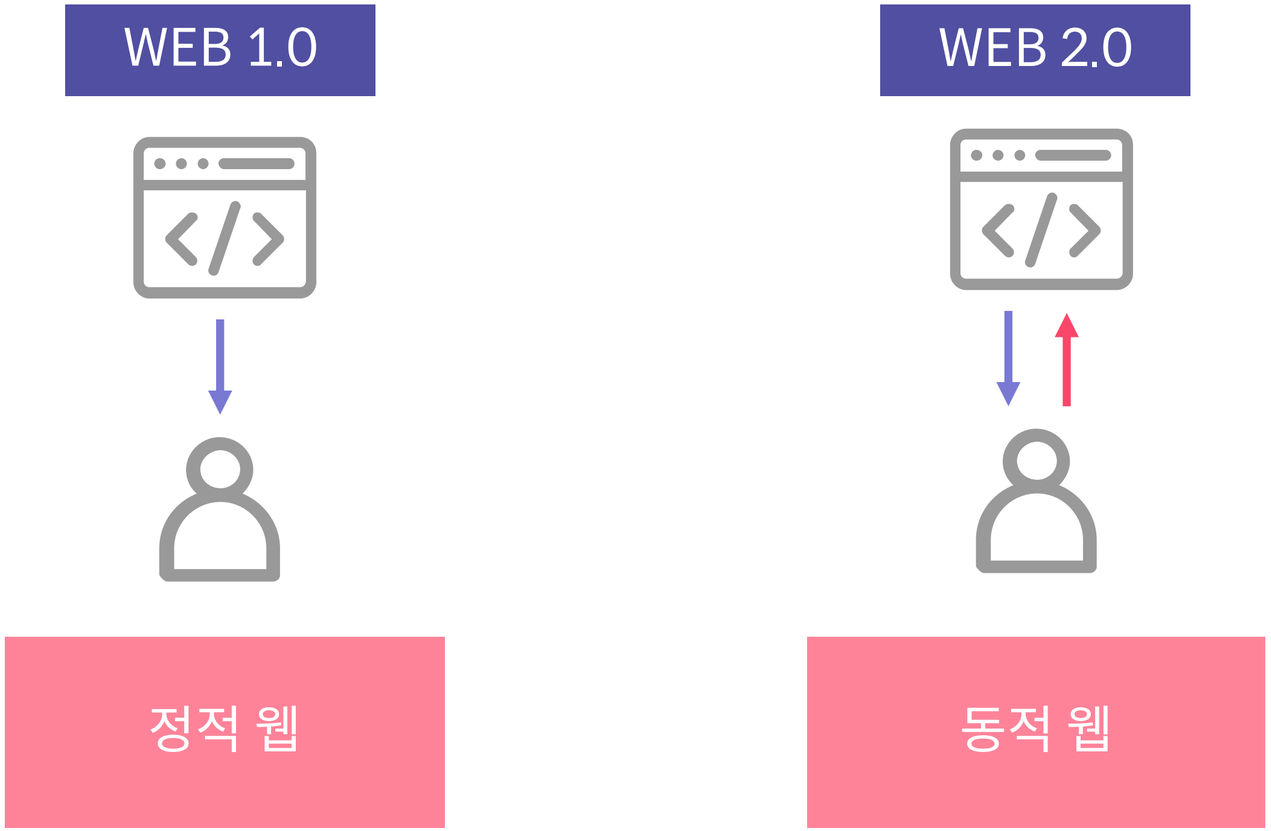
백엔드와 프론트엔드

웹 서비스 개발에서 **백엔드**와 **프론트엔드** 라는 단어를 많이 접하게 됨
프론트엔드는 **사용자가 직접 사용**하게 되는 **웹 페이지**를 주로 담당 = 클라이언트
백엔드는 **사용자에게 보이지 않는 데이터 가공 등의 기능**을 주로 담당 = 서버

백엔드와 프론트엔드


사용자
url 입력
응답 시각화
웹 브라우저
(클라이언트)
http 요청
인터넷
http 응답
서버
프론트엔드
담당
백엔드
담당

정적 웹과 동적 웹



정적 웹

Web 1.0

사용자와 상호작용하지 않는 페이지 - **단방향 통신**

Link를 통한 페이지 이동 정도만 가능

일반적으로 **변하지 않는 html 파일**로 제공

동적 웹

Web 2.0

사용자와 **상호작용**을 함 - **양방향 통신**

구글 맵, 웹 채팅, elice.io 등 **사용자가 다양한 기능을 수행**할 수 있음

프론트엔드와 백엔드가 **유기적으로 통신하며 동작**

현대적인 웹은 대부분 동적 웹

동적 웹의 두 가지 구현방법

CSR

Client-**S**ide **R**endering

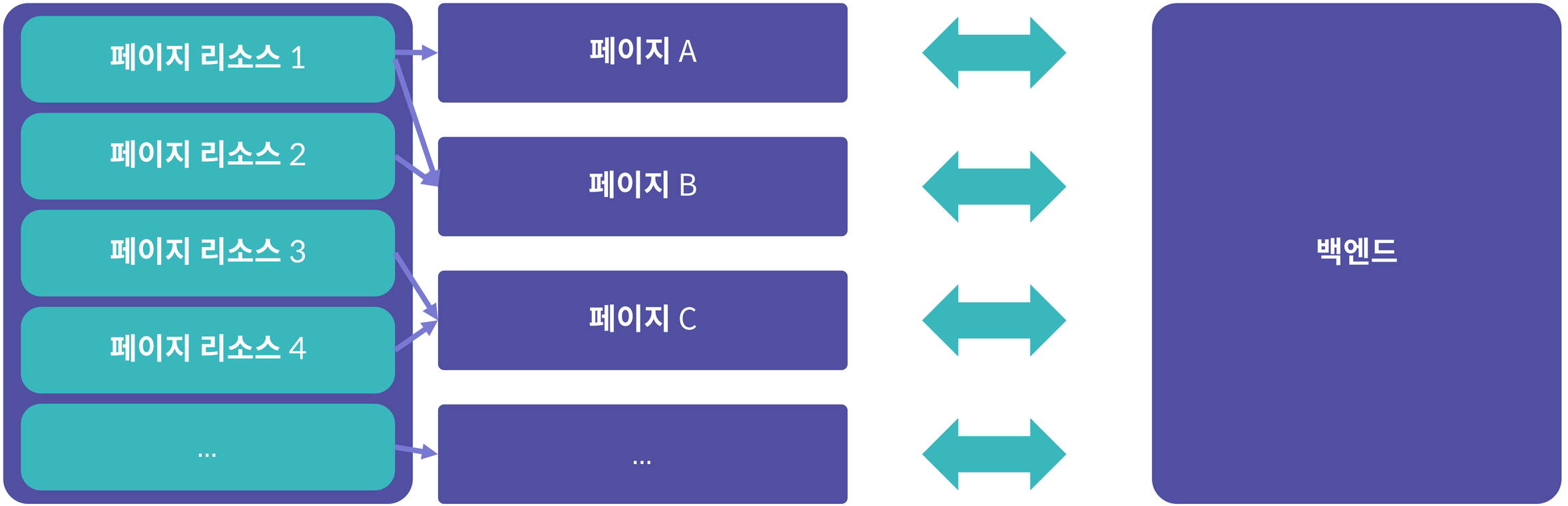
프론트엔드에서 사용자가 페이지에서 보는 **동적인 부분을 대부분 처리**하는 방식

SSR

Server-**S**ide **R**endering

백엔드에서 페이지 대부분의 영역을 처리해서 프론트엔드로 전달하는 방식

CSR



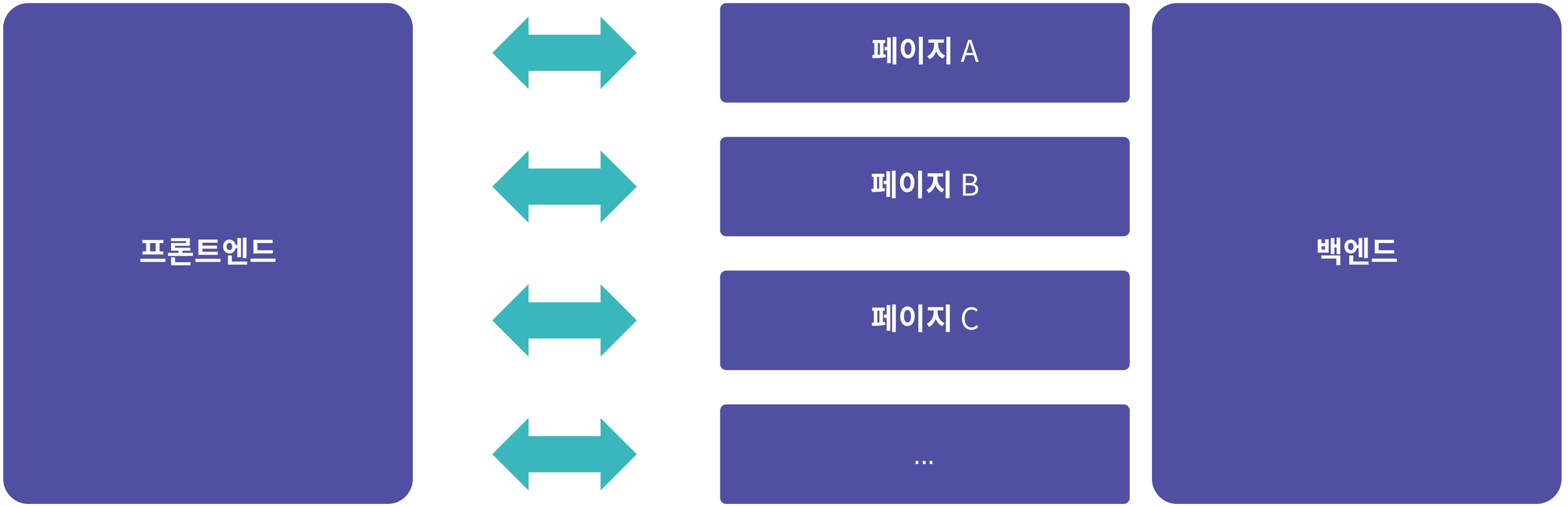
프론트엔드가 대부분의 리소스를 가지고
페이지를 표현

백엔드와 API 통신으로
데이터만 주고받음

CSR의 특징

사이트가 변하는 부분들을 **프론트엔드에서 처리**
프론트 엔드 코드에 **페이지 리소스들이 미리 정의**되어 있음
서버와의 통신은 **API 통신**을 이용
빠른 반응이지만 **페이지의 내용은 API 호출이 완료된 후**에 보여짐
복잡한 프로젝트 구성, 큰 개발 사이즈

SSR



프론트엔드는 HTTP 응답을 받아
화면에 표시

백엔드에서 필요한 데이터가 포함된
페이지를 만들어서 HTTP응답에 전달

SSR

사이트가 변하는 부분들을 **백엔드에서 처리**
백엔드에서 HTML 파일을 작성해서 프론트엔드로 전달
CSR에 비해 **쉬운 구성, 작은 개발사이즈**
로딩이 완료되면 **페이지와 데이터가 한 번에 표시**됨
상대적으로 사용자가 보기엔 **로딩이 느려 보임**
페이지 이동할 때마다 다시 로딩하기 때문에 **페이지 깜빡임**

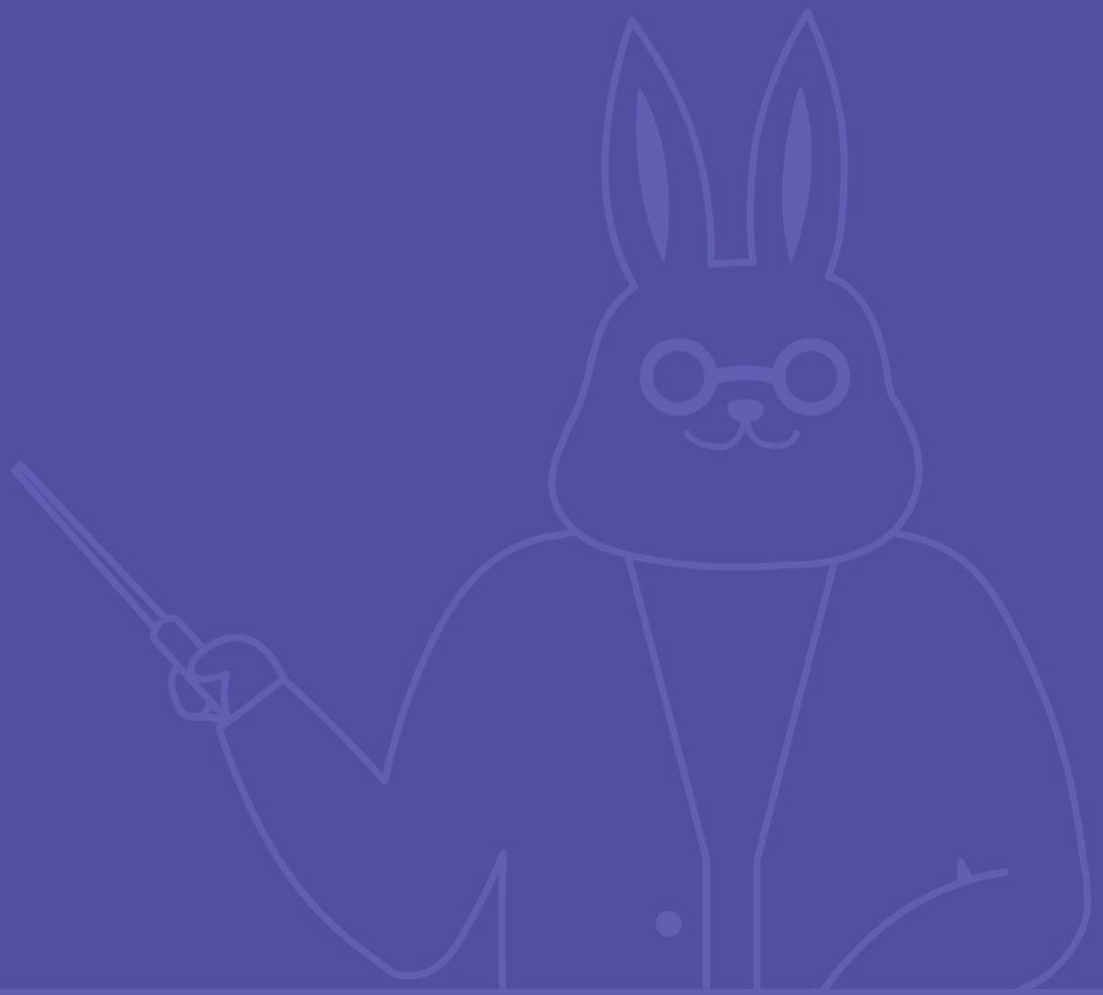
웹 서비스 동작 방식 정리

웹 서비스는 **HTTP 요청과 응답**으로 동작함
클라이언트는 서버로 HTTP 요청을, 서버는 클라이언트로 HTTP 응답을 보냄
프론트엔드는 클라이언트를 담당, **백엔드는 서버**를 담당

현대적인 많은 웹은 동적 웹으로 구현되어 있음
동적 웹을 클라이언트에서 주로 담당하는 것을 **CSR**이라고 하며,
서버에서 주로 담당하는 것을 **SSR**이라고 함.

03

웹 프레임워크



웹 프레임워크란?

웹		프레임워크
웹 서비스에 필요한 기능들을 제공해주는	+	다양한 도구들의 모음

웹 프레임워크를 사용하는 이유

웹 서비스를 구성하기 위해서는 **매우 많은 기능이 필요**함
이러한 기능들을 하나씩 직접 만드는 것에는 **너무나 큰 비용이 발생**
웹 서비스는 **많은 부분이 정형화**되어 있음
프레임워크를 사용하여 **정형화된 부분을 간단하게 구현**,
필요한 부분만 집중해서 개발 할 수 있음

웹 프레임워크의 기본 구성요소

웹 서비스의 **정형화 된 구성**을 많은 웹 프레임워크가 **기본적으로 제공**함

- ✓ HTTP 요청 처리
- ✓ HTTP 응답 처리
- ✓ 라우팅
- ✓ HTML Templating

웹 프레임워크 - HTTP 요청 처리

웹 프레임워크는 **HTTP 요청을 처리**할 수 있음
어떤 데이터를 필요로 하는지,
어떤 사용자로부터 요청이 수신되었는지 등

웹 프레임워크 - HTTP 응답 처리

웹 프레임워크는 **HTTP 응답을 처리**할 수 있음
응답 데이터가 어떤 형식인지,
응답 상태가 정상적인지 등

웹 프레임워크 - 라우팅

웹 프레임워크는 **HTTP 요청을 분기**하는 방법을 제공함
HTTP 요청 URL에 해당하는 **알맞은 응답의 경로를 미리 설정**

웹 프레임워크 - 라우팅

HTTP 요청에 따라 **알맞은 응답을 보내주는 경로**를 설정해주는 일

웹 프레임워크 - HTML Templating

웹 프레임워크는 **SSR을 구현하기 위한 방법**을 제공함
SSR에서 응답으로 보낼 HTML을 서버에서 작성하기 위해,
HTML Template를 통해 미리 **페이지의 뼈대를 작성** 가능

Node.js 의 웹 프레임워크

Node.js 에는 다양한 웹 프레임워크가 있음

Express.js - Node.js 의 가장 유명한 웹 프레임워크 (본 강의에서 사용)
Koa.js - 현대적인 JavaScript를 적극적으로 사용하는 웹 프레임워크
Nest.js - TypeScript 를 사용하며, 고정된 구조를 제공하는 웹 프레임워크
기타 - Hapi, Sails.js, Meteor.js 등

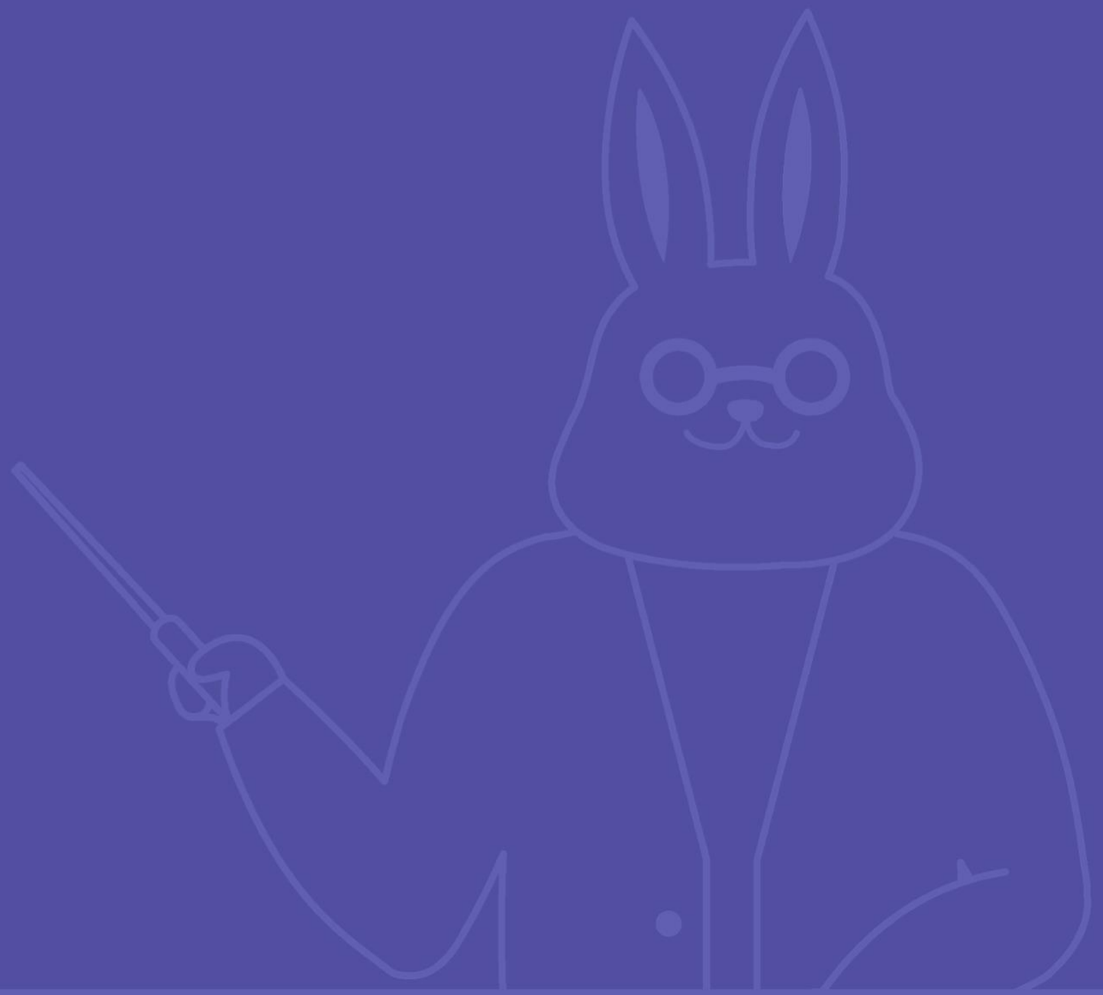
웹 프레임워크 정리

웹 서비스를 빠르게 구성하기 위해 웹 프레임워크를 사용할 수 있음
웹 프레임워크는 HTTP 요청, 응답, 라우팅, HTML Templating 등의 기능을 제공함

Node.js 에도 다양한 웹 프레임워크가 있으며,
본 강의에서는 Node.js의 **가장 유명한 웹 프레임워크인 Express.js**를 다룸

04

Express.js 시작하기



Express.js를 사용하는 이유

Express.js는 Node.js의 웹 프레임워크 중 **가장 유명한** 웹 프레임워크
필요에 따라 **유연하게 구조 설정** 가능
다양한 미들웨어를 통해 **필요한 기능을 간단하게 추가** 가능
모든 동작이 명시적으로 구성되기 때문에,
웹 프레임워크의 동작 방식을 이해하기 가장 좋은 프레임워크

npm init 으로 시작하기

npm init express

```
$mkdir my-web
$cd my-web
$npm init
$npm i express
```

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

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

app.listen(3000);
```

Express.js를 **처음부터 작성**할 수 있는 방법

직접 모든 구조를 작성해야 하기 때문에, Express.js를 **처음 접하는 사용자에겐 쉽지 않음**

express-generator 사용하기

express-generator

```
$npm i -g express-generator
$express my-web
$cd my-web
$npm i
$npm start
```

Express.js는 express-generator 라고 하는 **프로젝트 생성기**를 제공함

express-generator를 사용하면
프로젝트의 **기본구조를 자동으로 생성**해줌

빠르게 프로젝트를 시작하기 좋은 방법

생성된 프로젝트는 **npm start** 로 실행 가능

npx + express-generator 사용하기

npx + express-generator

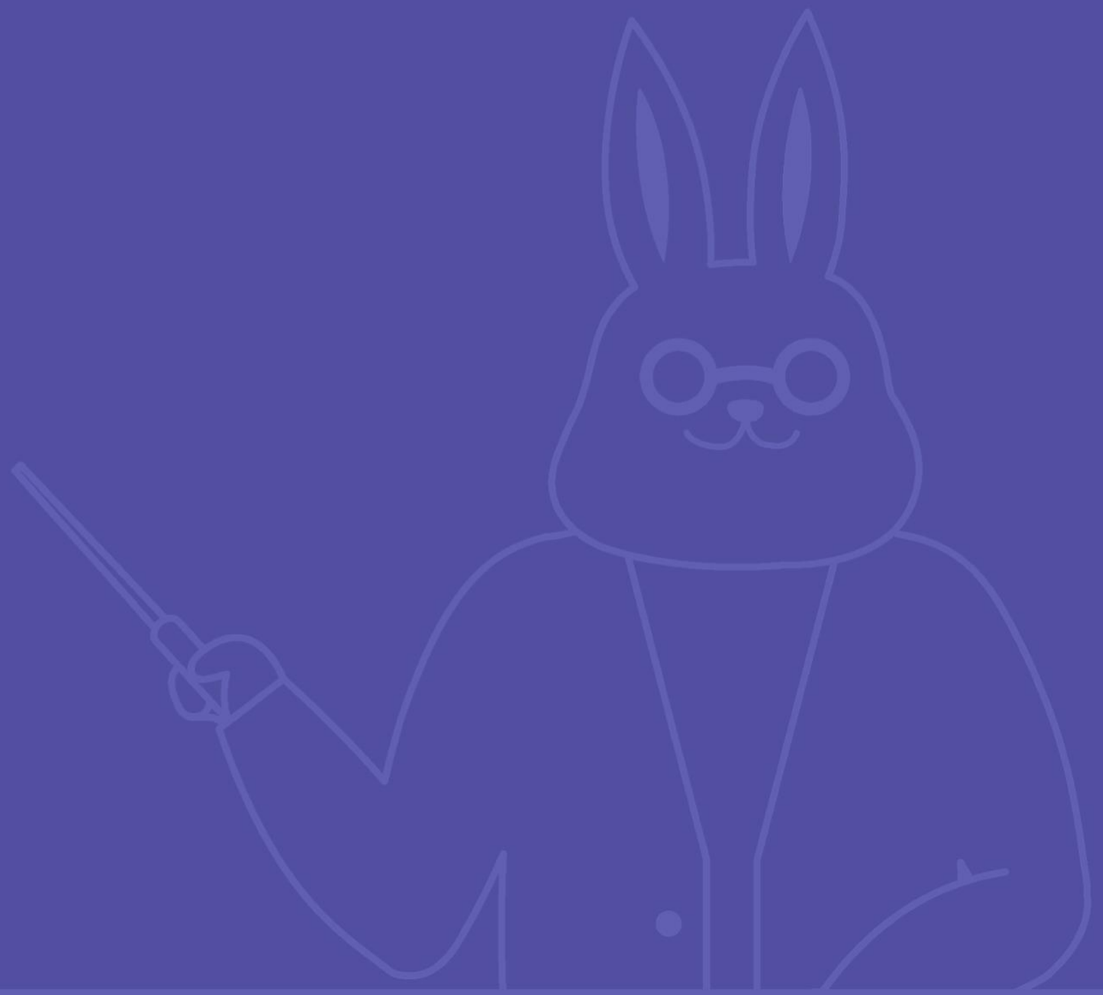
```
$npx express-generator my-web
$cd my-web
$npm i
$npm start
```

npx를 사용하여 express-generator 를 설치하지 않고, 바로 사용 가능.

express-generator는
프로젝트 생성 이후엔 사용되지 않기 때문에,
npx를 사용하는 것도 좋은 방법

05

Express.js 의 구조

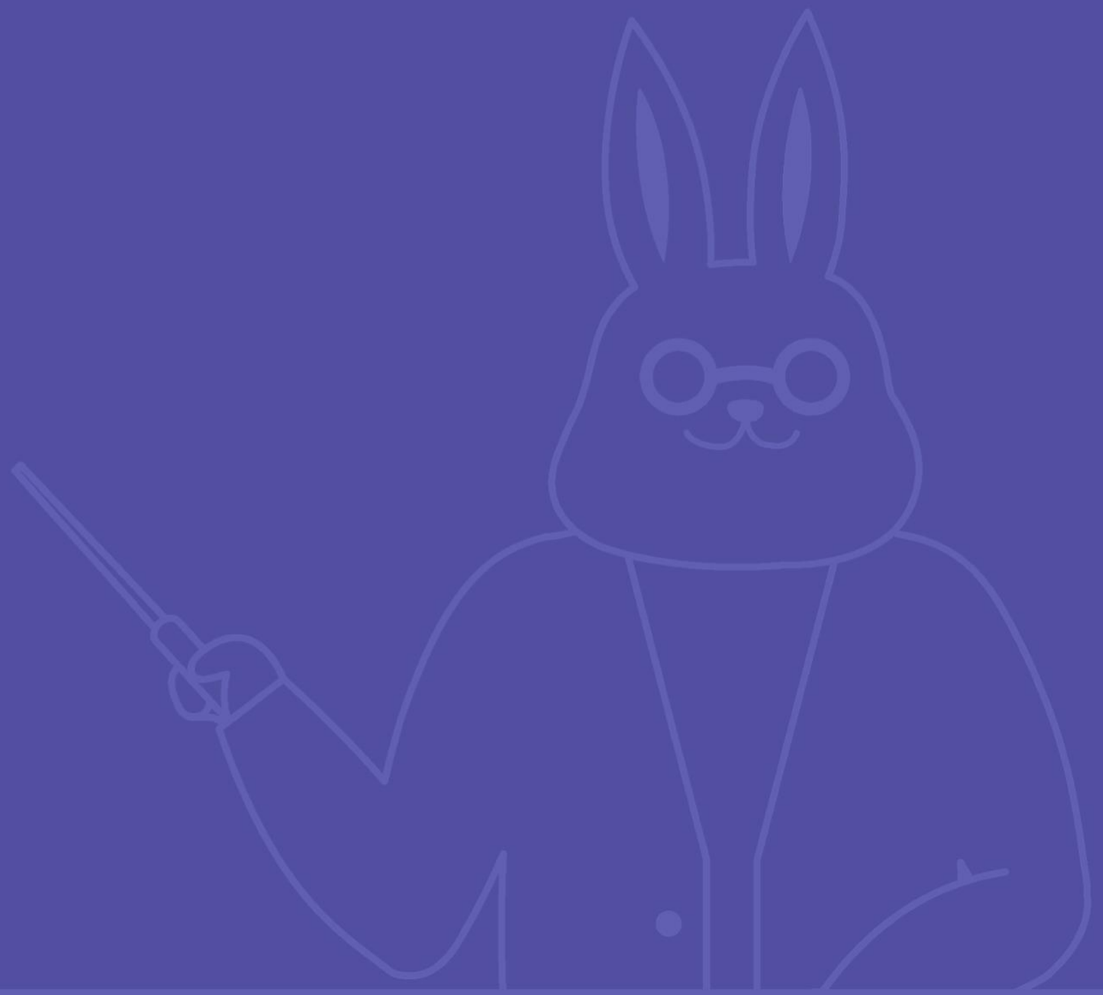


기본구조 알아보기


my-web
app.js
Express.js 의 가장 기본이 되는 파일, 웹 어플리케이션의 기능을 정의
bin/www
Express.js 실행 부분을 담당, 포트와 실행 오류 등을 정의
package.json
프로젝트 의존성 및 스크립트 정의
public
코드를 통하지 않고, 직접 제공되는 파일 디렉터리
routes
라우팅 파일 디렉터리
views
HTML Template 디렉터리

06

Express.js 동작 방식

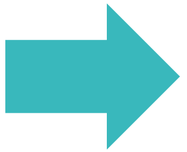


Express.js 동작 시켜 보기

express-generator로 만들어진 프로젝트 디렉터리에 접근하여,
npm start 로 Express.js 프로젝트를 실행할 수 있음
localhost:3000 에 접속하여 Welcome to Express 페이지를 확인할 수 있음

Express.js 동작 따라가기

1. 브라우저에서 localhost:3000 접속
2. app.js → app.use('/', indexRouter);
3. routes/index.js → router.get('/', ...
4. routes/index.js → res.render('index', ...
5. views/index.jade



app.js

app.js

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

app.js에서는 express()로 생성되는 app 객체를 확인할 수 있음

app 객체는 Express.js 의 기능을 담은 객체

Express.js의 모든 동작은 app 객체에 정의됨

app 객체 - 주요 기능

app.use()	middleware 를 사용하기 위한 함수 미들웨어에 대한 자세한 설명은 다음 장에서 학습함
app.listen()	http 서버를 생성해주는 함수 express-generator 를 사용하면 http.createServer를 사용하는데 app.listen 함수로 대체할 수 있음
app.locals	app에서 사용할 공통 상수 Express.js 에선 global 변수를 선언하지 않고 이 값을 사용할 수 있음

라우팅

Express.js는 다양한 라우팅 방식을 제공함
크게 **app 라우팅**과 **Express.Router**를 통한 라우팅으로 나누어짐

app 라우팅

app 라우팅

```
app.get('/', (req, res) => {
  res.send('GET /');
});
app.post('/', (req, res) => {
  res.send('POST /');
});
app.put('/', (req, res) => {
  res.send('PUT /');
});
app.delete('/', (req, res) => {
  res.send('DELETE /');
});
app.all('/all', (req, res) => {
  res.send('ANY /');
});
```

app 객체에 직접 get, post, put, delete 함수를 사용하여 HTTP method 로 라우팅 할 수 있음.

HTTP method 함수의 첫 번째 인자가 이 라우팅을 실행할 URL

마지막 인자가 이 라우팅이 실행될 때 작동하는 함수

all 함수를 사용하면 HTTP method에 상관없이 라우팅 가능

Express.Router

app 라우팅을 통해서는 라우팅의 핵심인 그룹화를 지원하지 않음
Express.Router 를 통해 라우팅을 모듈화 할 수 있음

Express.Router 모듈

Express.Router

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

router.get('/', (req, res, next) => {
  res.send('respond with a resource');
});

module.exports = router;
```

router 객체에도 app 객체처럼
get, put, post, delete 함수를 사용할 수 있음

app의 함수와 동일한 동작을 하는 함수로
첫 번째 인자가 라우팅 될 URL이고,
마지막 인자가 라우팅 시 실행될 함수

라우터는 일반적으로 모듈로 만들어서 사용함

Express.Router 사용

use Router

```
--- ./app.js
const userRouter = require('./routes/users');
const app = express();

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

--- ./routes/users.js
const petRouter = require('./pets');
const router = express.Router();

router.use('/pets', petRouter);

module.exports = router;
```

작성된 라우터 모듈을 app 에 use함수로 연결하여 사용할 수 있음

router 객체에도 하위 라우터를 use 함수로 연결하여 사용할 수 있음

라우팅 - path parameter 사용

Express.js 라우팅은 **path parameter**를 제공
path parameter를 사용하면, **주소의 일부를 변수처럼 사용**할 수 있음

Ex)
/users/**:id** - /users/123, /users/456 등으로 접속했을 때 라우팅 적용
/messages/**:from**-**:to** - /message/123-456 등으로 접속했을 때 라우팅 적용

Request Handler

라우팅에 적용되는 함수를 **Request Handler**라고 부름
HTTP 요청과 응답을 다룰 수 있는 함수로
설정된 **라우팅 경로에 해당하는 요청**이 들어오면 Request Handler **함수가 실행**됨

Request Handler

Request Handler

```
router.get('/:id', (req, res) => {
  const id = req.params.id
  res.send(`hello ${id}`);
});
```

router 나 app의 HTTP method 함수의
가장 **마지막 인자로 전달되는 함수**

설정된 라우팅 경로에 해당하는 요청이
들어오면 Request Handler 함수가 실행됨

요청을 확인하고, 응답을 보내는 역할을 함

Request Handler - Request 객체

HTTP 요청 정보를 가진 객체
HTTP 요청의 **path parameter, query parameter, body, header** 등을 확인 가능

Request Handler - Request 객체의 주요 값 및 함수

req.params	URL 표현 중 /path/:id 에서 :id 를 req.params.id 로 사용할 수 있음
req.queries	URL 표현 중 /path?page=2 에서 page 부분을 req.queries.page 로 사용할 수 있음
req.body	일반적으로 POST 요청의 요청 데이터를 담고 있음 req.body 에 요청 데이터가 저장되어 들어옴
req.get('')	HTTP Request 의 헤더 값을 가져올 수 있음 req.get('Authorization') 등으로 값을 가져옴

Request Handler – Response 객체

HTTP 응답을 처리하는 객체
HTTP 응답의 **데이터를 전송**하거나, **응답 상태 및 헤더를 설정**할 수 있음

Request Handler - Response 객체의 주요 값 및 함수

res.send()	text 형식의 HTTP 응답을 전송함
res.json()	json 형식의 HTTP 응답을 전송함
res.render()	HTML Template 을 사용하여 화면을 전송함
res.set()	HTTP 응답의 헤더를 설정함
res.status()	HTTP 응답의 상태 값을 설정함

Express.js 동작방식 정리

Express.js 는 **app 객체를 시작**으로 모든 동작이 이루어짐
app 객체나 Express.Router를 사용하여 **라우팅을 구현**할 수 있음
Request Handler 를 통해 **HTTP 요청과 응답을 처리**할 수 있음

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
최규범

강사
최규범

감수자
최규범

디자이너
김루미

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io