



Session 9 | React 환경세팅 & 라우팅

13기 개발팀 김은성

Session 9 | React 환경세팅 & 라우팅

1. React 소개

a. React를 쓰는 이유

b. node.js

c. npm

d. JSX 문법

2. 라우팅

a. react-router-dom

b. 실습

| React란?

싱글 페이지 어플리케이션의 UI를 구축하기 위한
JavaScript 기반의 오픈 소스 라이브러리

| SPA

Single Page Application

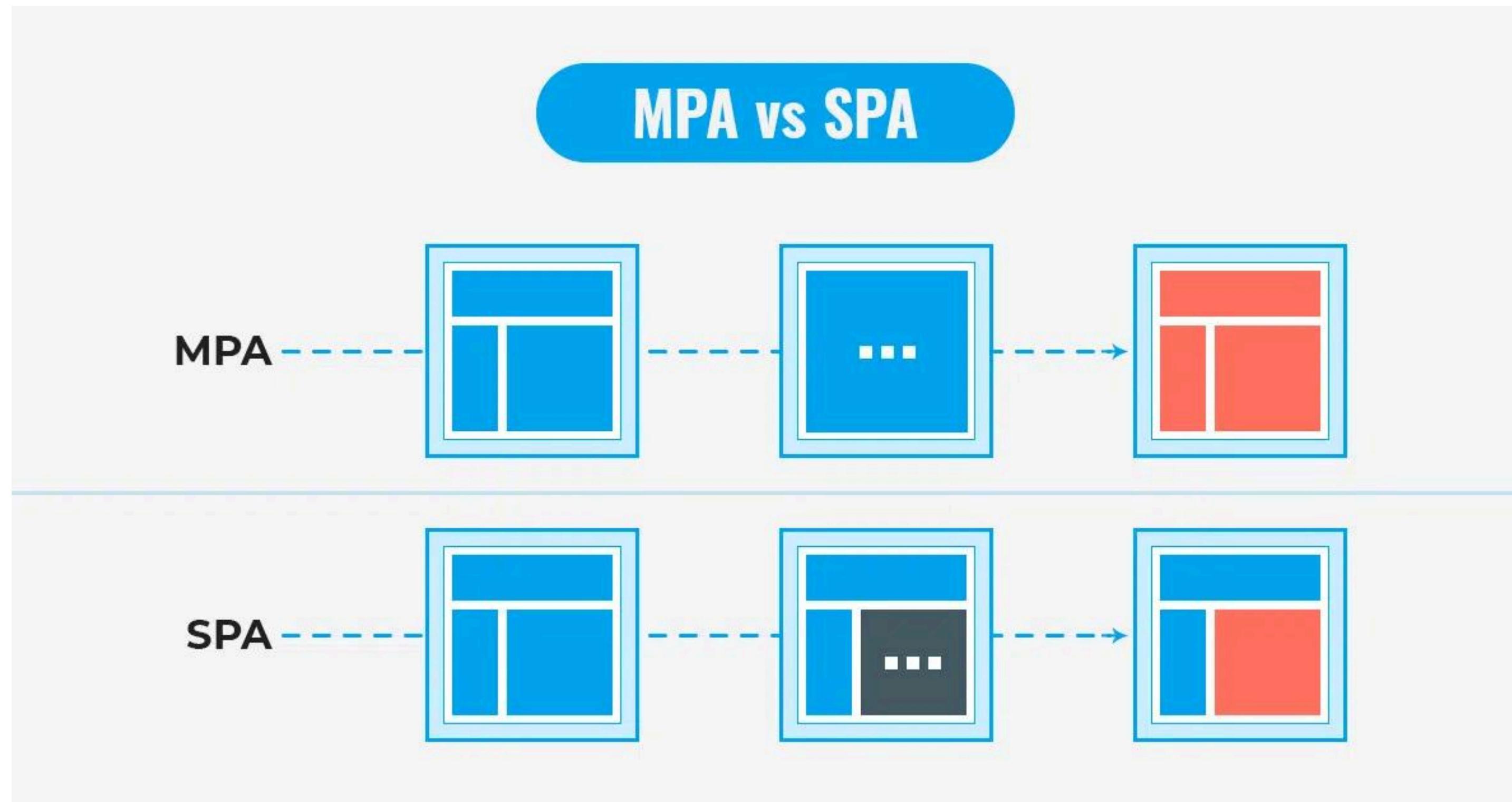
단일(Single) 페이지(Page)를 로드하고 사용자가

앱과 상호작용할 때 해당 페이지를 동적으로

업데이트하는 웹앱

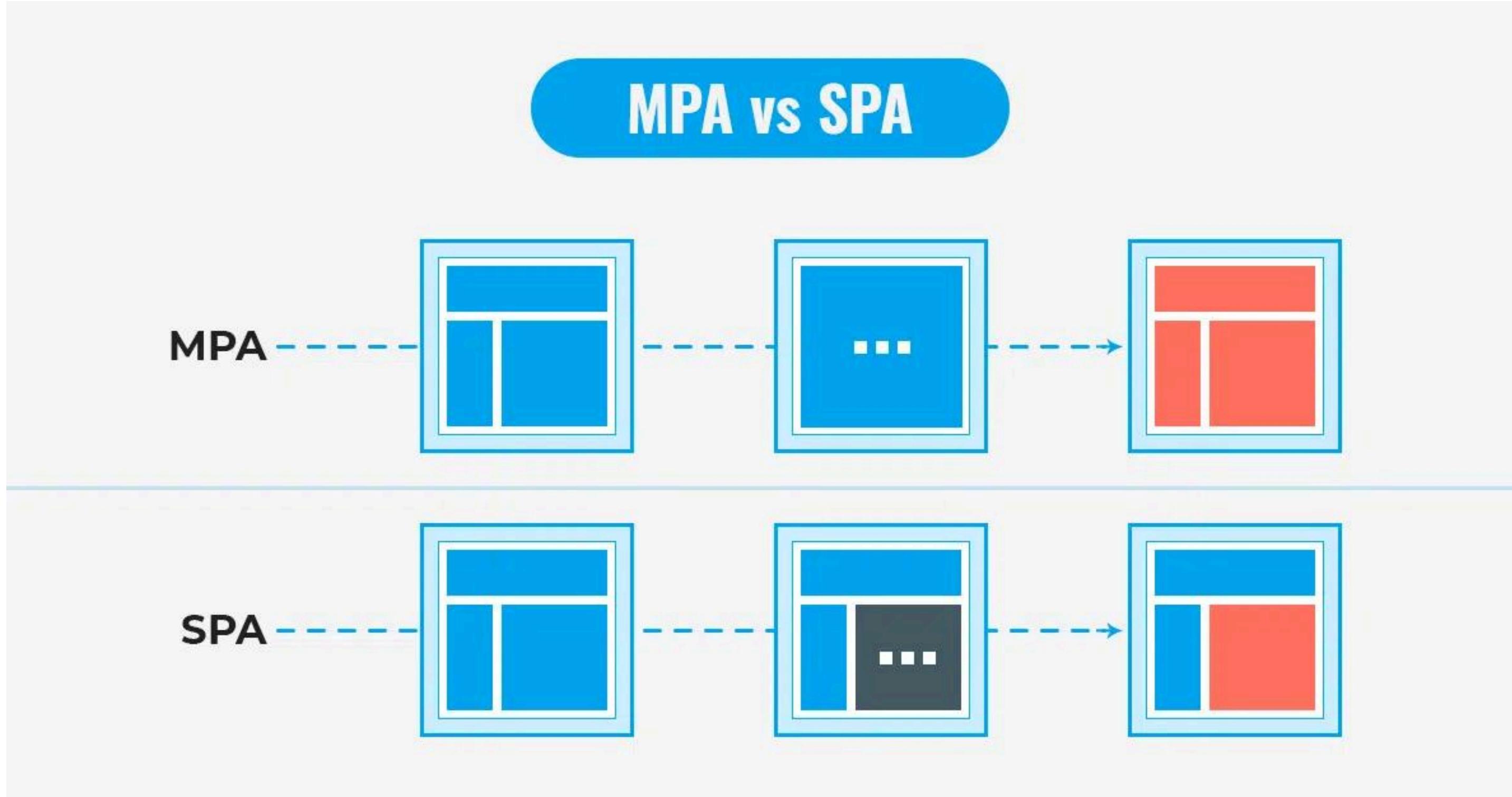
| SPA

MPA와 SPA의 차이점은?



| SPA

MPA란?

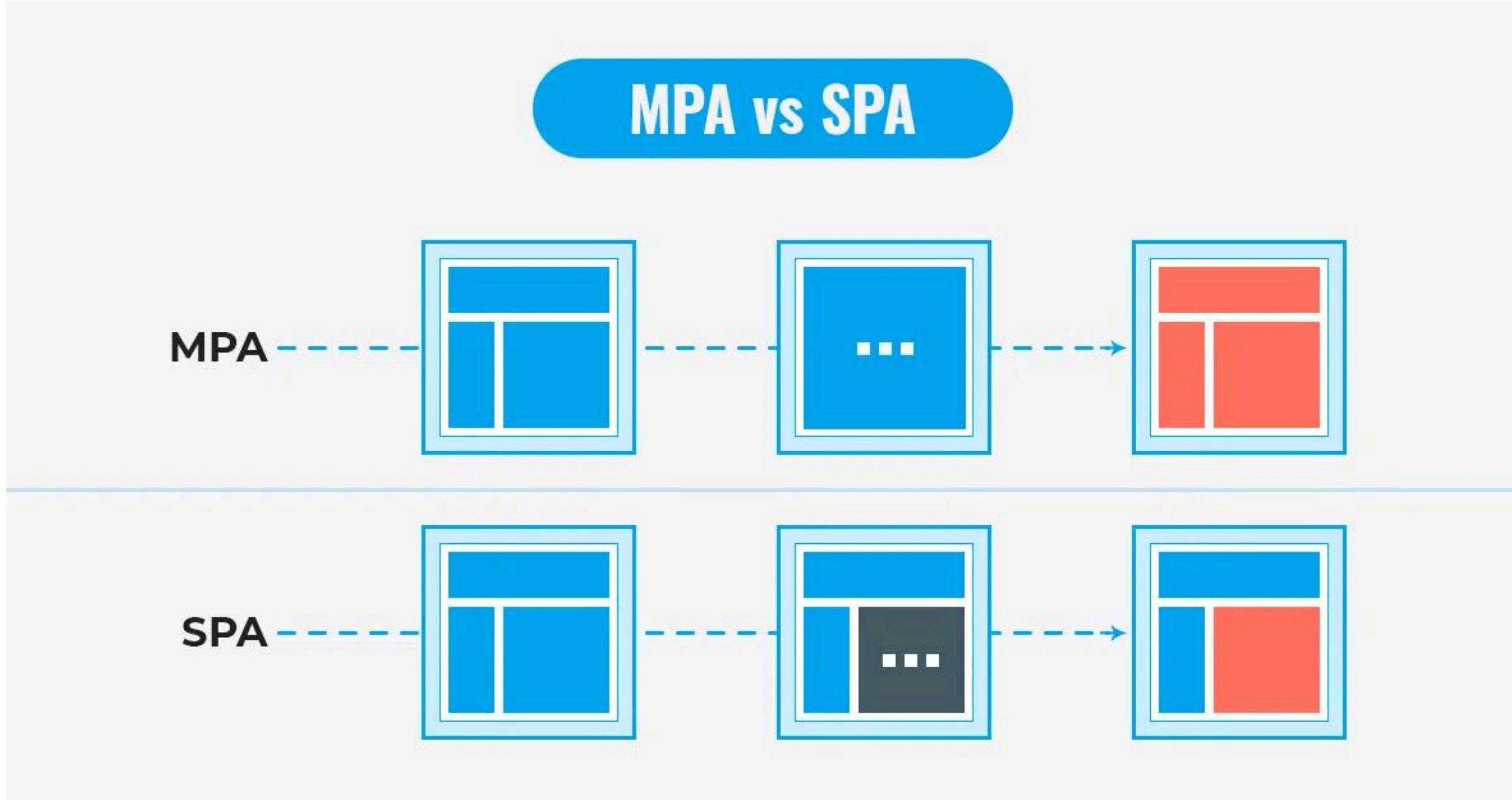


Multiple Page Application

- 새로운 페이지를 요청할 때마다 서버에서 렌더링된 정적 리소스가 다운로드
- 페이지를 이동하거나 새로고침하면 전체 페이지를 재렌더링
- 여러 페이지를 전환하는 방식이 필요한 경우에 적합
- 각 페이지가 독립적으로 SEO 최적화 가능

| SPA

SPA란?



Single Page Application

- 웹앱에 필요한 모든 정적 리소스를 최초 접근 시 한 번만 다운로드
- 페이지를 이동하거나 새로고침하면 페이지 갱신에 필요한 데이터만 수정 후 렌더링
- 사용자 인터랙션이 많은 웹 애플리케이션에 적합
- SEO 최적화에 한계

| SPA

SEO(검색 엔진 최적화)란?



Search Engine Optimization

- 웹사이트나 웹페이지가 검색 엔진에서 더 높은 순위에 노출되도록 최적화하는 과정
- 특정 키워드나 검색어를 검색했을 때, 검색 엔진 결과 페이지(SERP)에서 웹사이트가 상위에 랭크되게 하는 것
- SPA의 SEO 최적화 문제를 해결하기 위해 Next.js 사용하기도 함

| 프레임워크? 라이브러리?

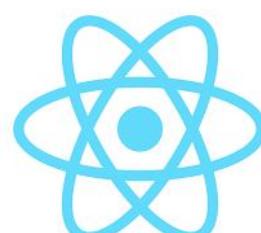
- **프레임워크
(framework)** : 웹 애플리케이션을 구축할 때,
공통적인 개발 환경을 제공해주는 구조를 짜놓고 그 위에 덧붙여 만들도록 하는 것

- ‘들어가서’ 사용한다: 프레임워크가 전체적인 흐름을 주고 있고, 사용자는 그 안에서 필요한 코드를 짜 넣는다



- **라이브러리
(library)** : 단순 활용 가능한 도구들의 집합

- ‘가져다가’ 사용한다: 사용자가 전체적인 흐름을 주고 있고, 필요한 라이브러리를 호출하여 가져다 쓴다



| React란?

싱글 페이지 어플리케이션의 UI를 구축하기 위한
프론트엔드 대표 라이브러리!!
JavaScript 기반의 오픈 소스 라이브러리

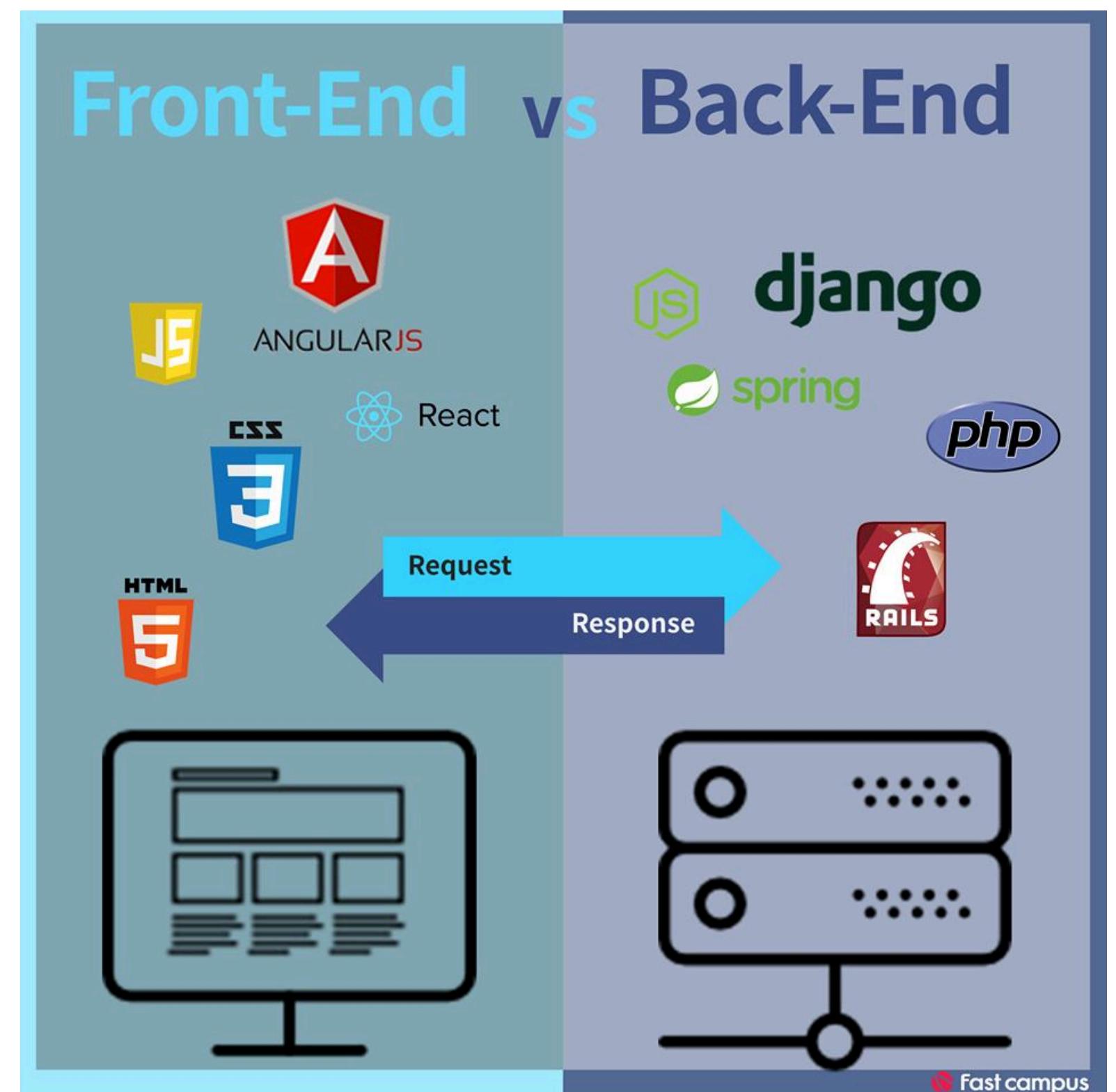
| 프론트엔드? 백엔드?

Front-end

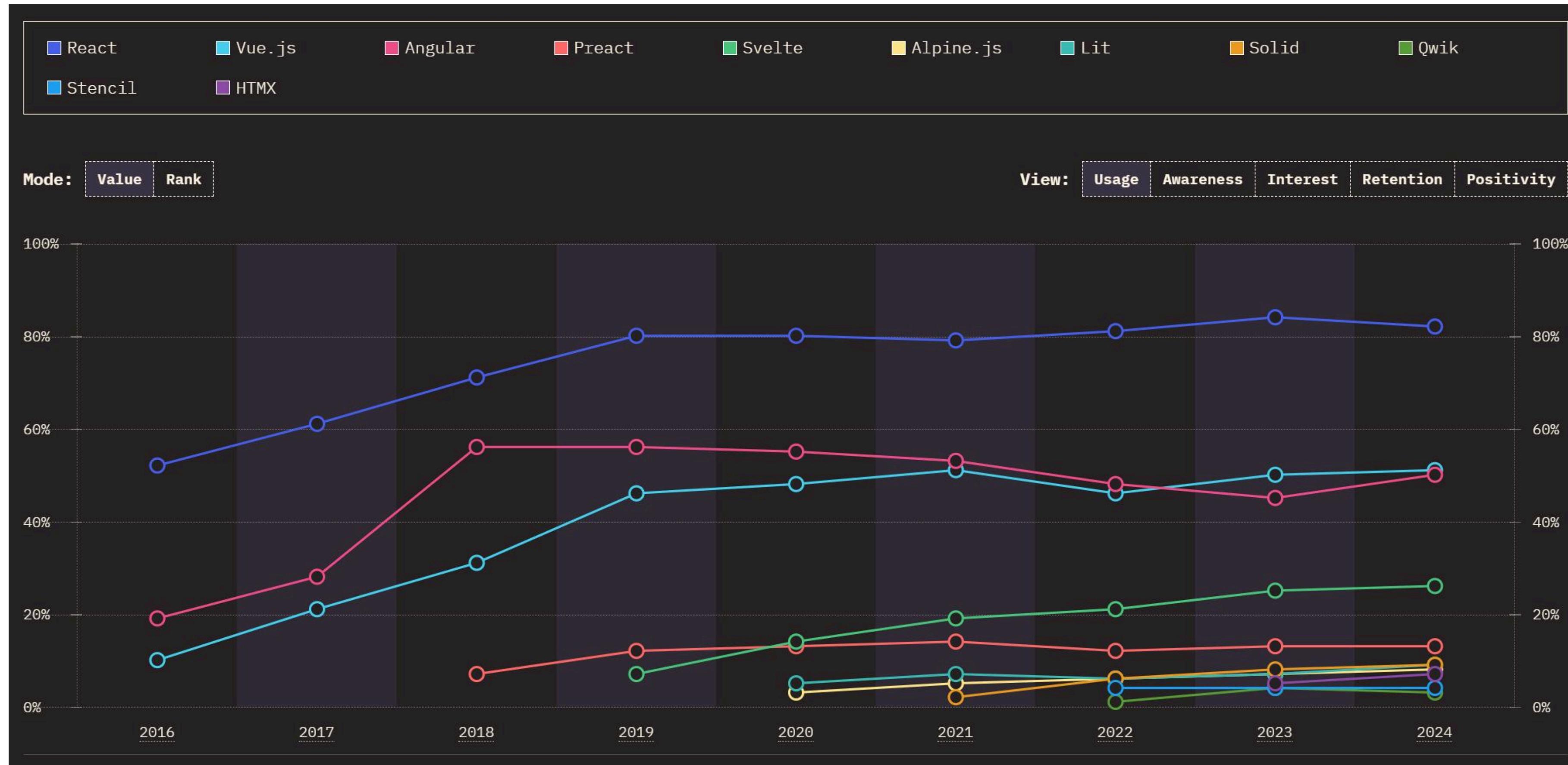
: 웹/앱에서 사용자가 직접 상호작용할 수 있는 사용자 인터페이스 (UI)를 개발

Back-end

: 웹/앱에서 사용자의 요청을 처리하고 정보를 저장, 관리, 전달하며 서버와 데이터베이스를 관리하는 역할



| React란?



그래프 출처: [State of JavaScript](#)

| React를 쓰는 이유

사람들은 왜 리액트를 쓸까?

1. 컴포넌트

- 중복 코드 최소화
- 유지보수 편리

```
<header>
  <h1>
    <a href="/">web</a>
  </h1>
</header>
<nav>
  <ol>
    <li><a href="/read/1">html</a></li>
    <li><a href="/read/2">css</a></li>
    <li><a href="/read/3">javascript</a></li>
  </ol>
</nav>
<article>
  <h2>Welcome</h2>
  Hello, WEB
</article>
```



```
<Header></Header>
<Nav></Nav>
<Article></Article>
```

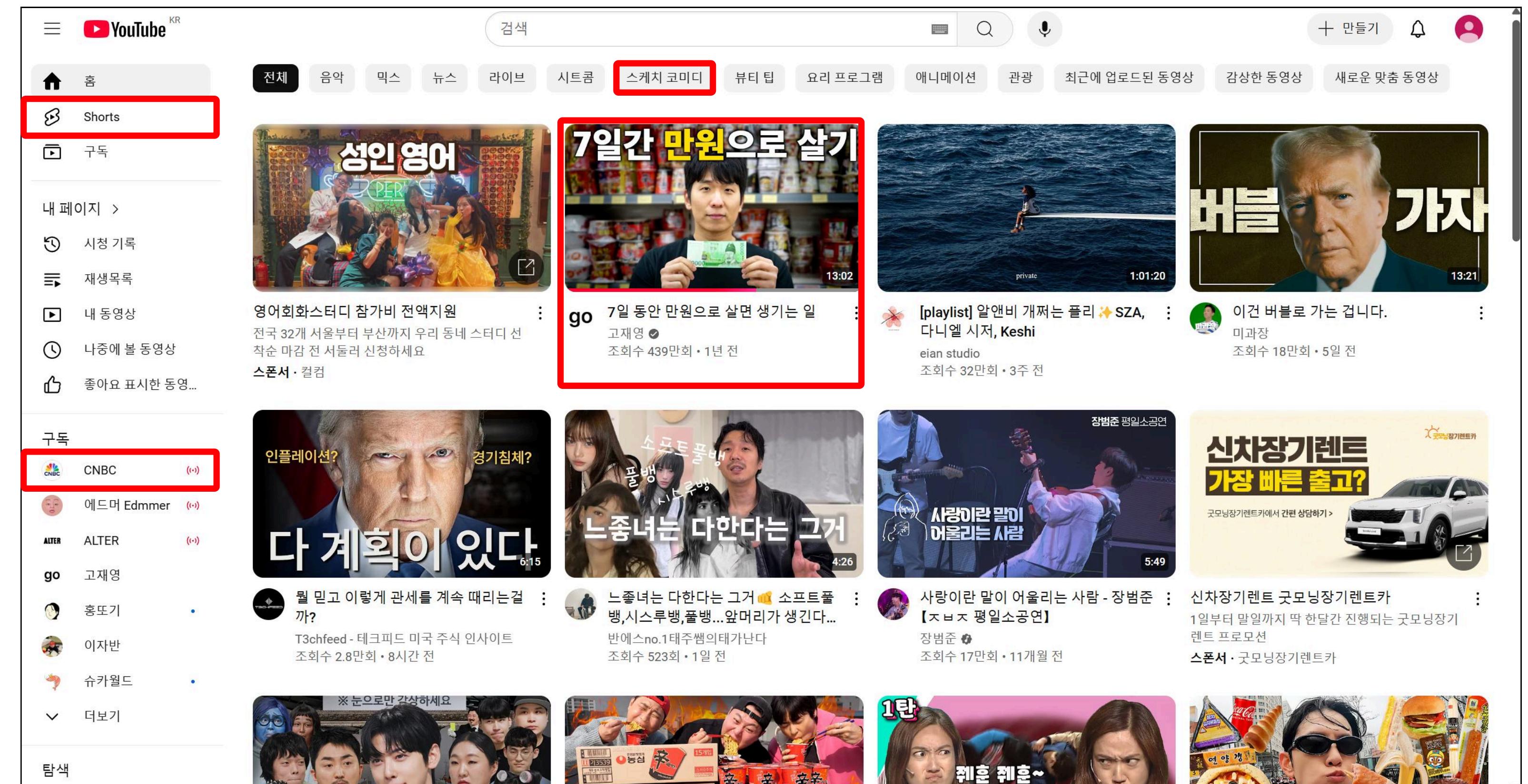
| React를 쓰는 이유

사람들은 왜 리액트를 쓸까?

컴포넌트란?

페이지를 구성하는 각각의 독립적이고 재사용 가능한 UI 요소

- 반복되는 구성 요소를 재사용할 수 있도록 만든다.
- 하나의 컴포넌트는 하나의 기능이나 역할을 담당한다.
- 여러 컴포넌트를 모아서 페이지를 완성한다.



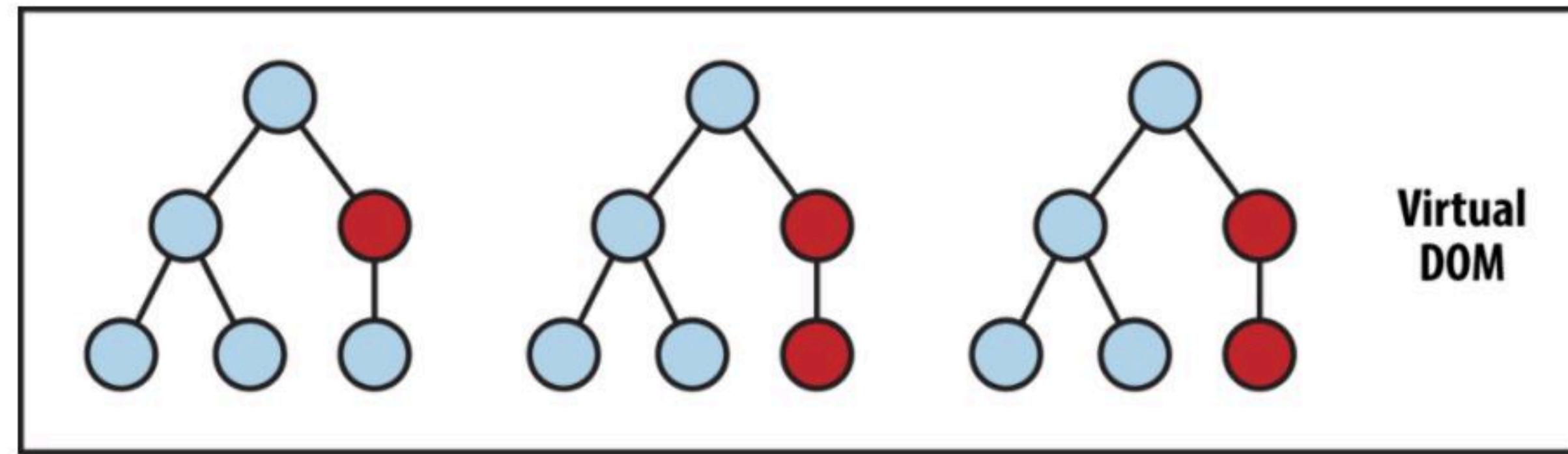
| React를 쓰는 이유

사람들은 왜 리액트를 쓸까?

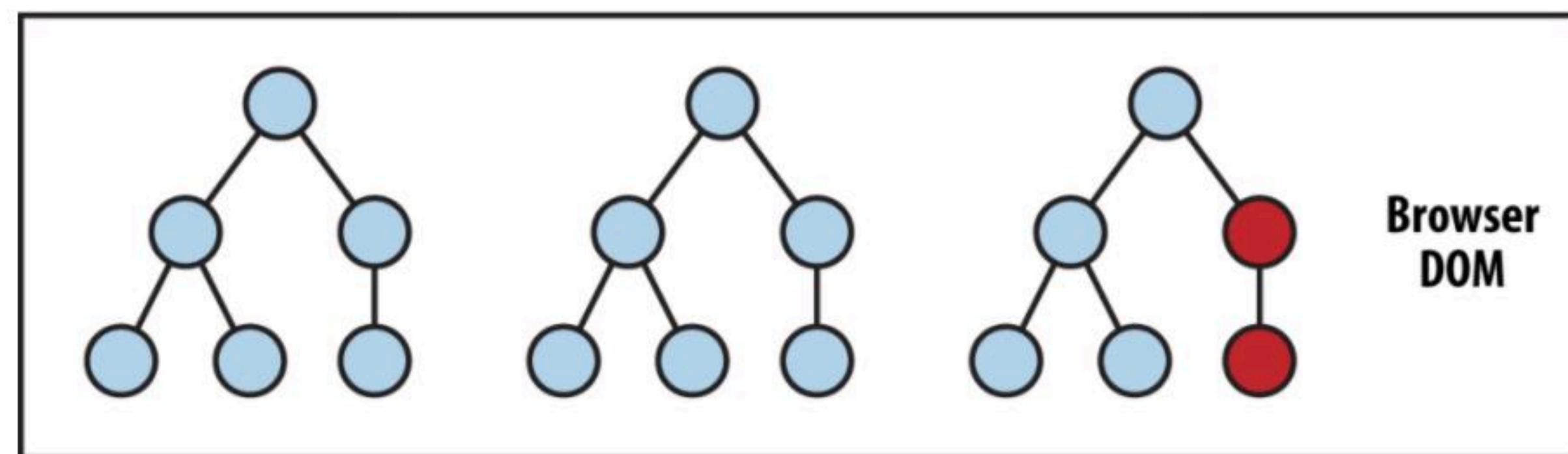
2. 가상 DOM

- 최소한의 변경사항만 반영
- 효율성 UP

<https://taedonn.com/posts/3/>



State Change → Compute Diff → Re-render



| 가상 DOM이란?

사람들은 왜 리액트를 쓸까?

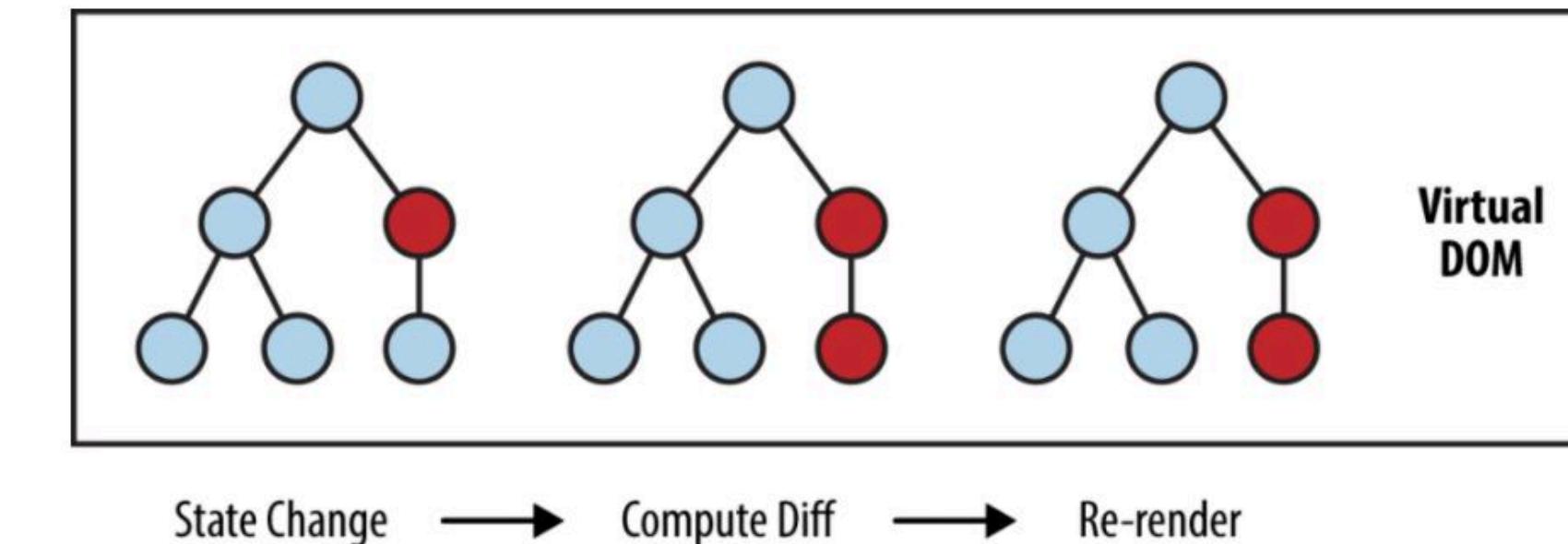
가상 DOM(Document Object Model)

실제 브라우저의 DOM이 아닌 가상의 DOM을 만들어 브라우저에 렌더링하는 방식

상태(State)가 변했을 때 브라우저 전체가 아닌 기존의 DOM과 달라진 부분만 변경
(Diffing)

→ 필요한 부분만 교체해 불필요한 업데이트를 줄이고, 렌더링 속도를 올려줌

1. 상태(State)가 바뀌면 UI를 가상 DOM에 렌더링함
2. 가상 DOM끼리 비교함
3. 바뀐 부분만 실제 DOM에 렌더링함

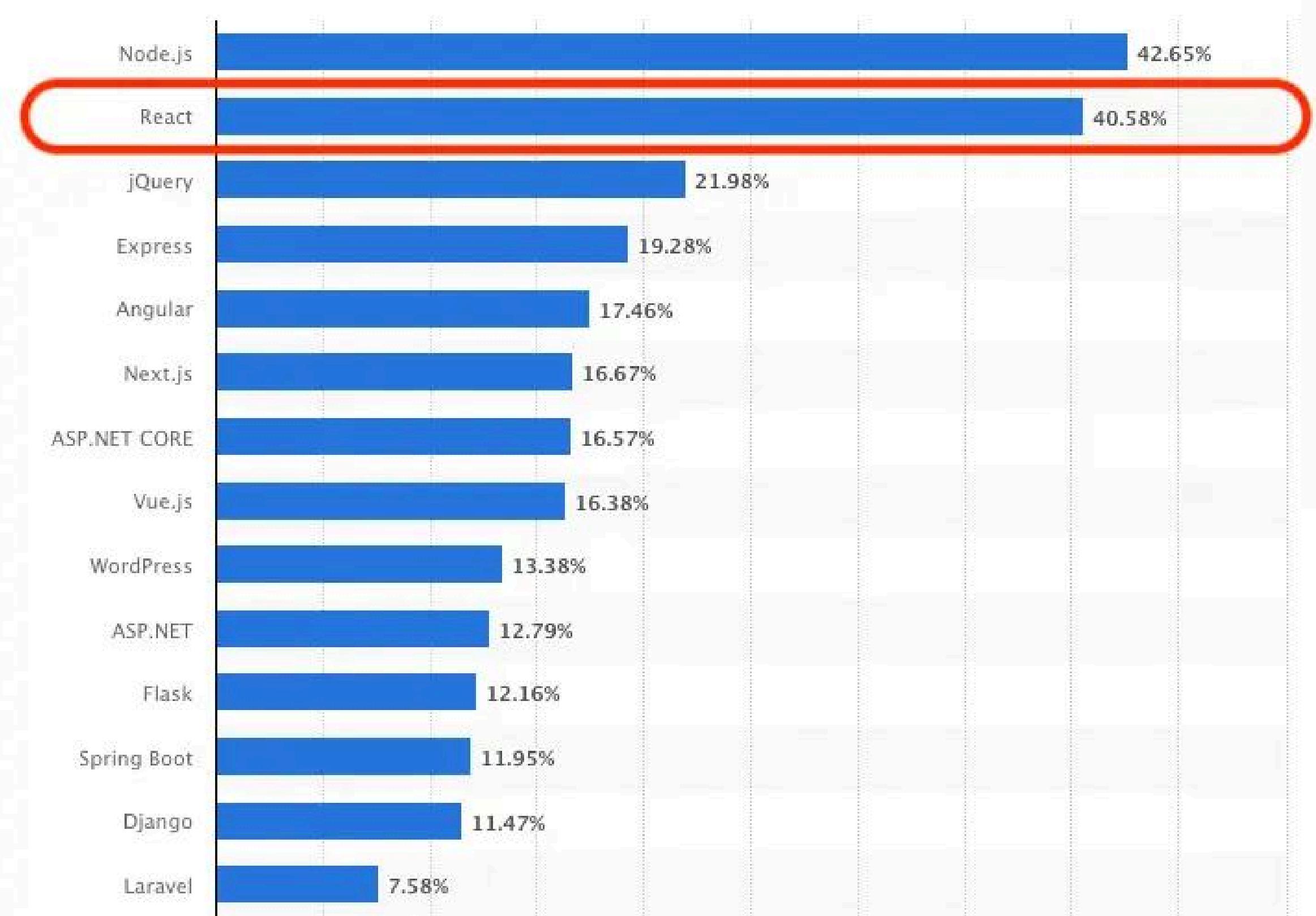


| React를 쓰는 이유

사람들은 왜 리액트를 쓸까?

3. 현업에서 사용 + 많은 레퍼런스

- 전 세계 웹 개발자의 40.58% 사용
- 국내 기업에서도 인기
- 수많은 레퍼런스 존재



| Node.js

리액트를 쓰는데 왜 node.js가 필요하지?

Node.js란?

Chrome V8 JavaScript 엔진 위에서 동작하는
자바스크립트 런타임(환경)

즉, 노드를 통해 다양한 자바스크립트 애플리케이션을 실행할 수 있다.



리액트를 쓰는데 왜 npm이 필요하지?

npm이란?

**Node Package Manager로,
node로 실행할 수 있는 패키지들을 관리하는 도구**

npm을 통해 패키지를 다운로드 받고, 업데이트 하고, 삭제할 수 있다.

| React 환경 세팅

node를 깔아봅시다

1. which node

```
~ which node  
node not found
```

node가 이미 설치된 경우, 기존 패키지 삭제

```
sudo apt remove nodejs npm -y  
sudo apt autoremove -y
```

```
~ which node  
/home/kes/.nvm/versions/node/v23.10.0/bin/node
```

| React 환경 세팅

node를 깔아봅시다

2. nvm 설치

curl -fsSL https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh | bash

```
~ curl -fsSL https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh | bash
=> Downloading nvm from git to '/home/kes/.nvm'
=> Cloning into '/home/kes/.nvm'...
remote: Enumerating objects: 381, done.
remote: Counting objects: 100% (381/381), done.
remote: Compressing objects: 100% (324/324), done.
remote: Total 381 (delta 43), reused 175 (delta 29), pack-reused 0 (from 0)
Receiving objects: 100% (381/381), 383.82 KiB | 3.96 MiB/s, done.
Resolving deltas: 100% (43/43), done.
* (HEAD detached at FETCH_HEAD)
  master
=> Compressing and cleaning up git repository

=> Appending nvm source string to /home/kes/.zshrc
=> Appending bash_completion source string to /home/kes/.zshrc
=> You currently have modules installed globally with `npm`. These will no
=> longer be linked to the active version of Node when you install a new node
=> with `nvm`; and they may (depending on how you construct your '$PATH')
=> override the binaries of modules installed with `nvm`:
```

| React 환경 세팅

node를 깔아봅시다

3. nvm 로드(명령어 또는 재실행)

```
export NVM_DIR="$HOME/.nvm"  
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/  
bash_completion"
```

```
~ export NVM_DIR="$HOME/.nvm"  
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"
```

| React 환경 세팅

node를 깔아봅시다

4. nvm 버전 확인

nvm --version

```
nvm --version  
0.39.5
```

| React 환경 세팅

node를 깔아봅시다

5. LTS 버전 설치

nvm install --lts

```
~ nvm install --lts                                     ok | 13:23:40
Installing latest LTS version.
Downloading and installing node v22.14.0...
Downloading https://nodejs.org/dist/v22.14.0/node-v22.14.0-linux-x64.tar.xz...
#####
Computing checksum with sha256sum
Checksums matched!
Now using node v22.14.0 (npm v10.9.2)
Creating default alias: default -> lts/* (-> v22.14.0)
```

| React 환경 세팅

node를 깔아봅시다

6. 최신 버전 설치

nvm install node

```
~ nvm install node                                         ok | 8s | 13:24:10
Downloading and installing node v23.8.0...
Downloading https://nodejs.org/dist/v23.8.0/node-v23.8.0-linux-x64.tar.xz...
#####
Computing checksum with sha256sum
Checksums matched!
Now using node v23.8.0 (npm v10.9.2)
```

| React 환경 세팅

node를 깔아봅시다

6. 버전 확인

node -v

npm -v

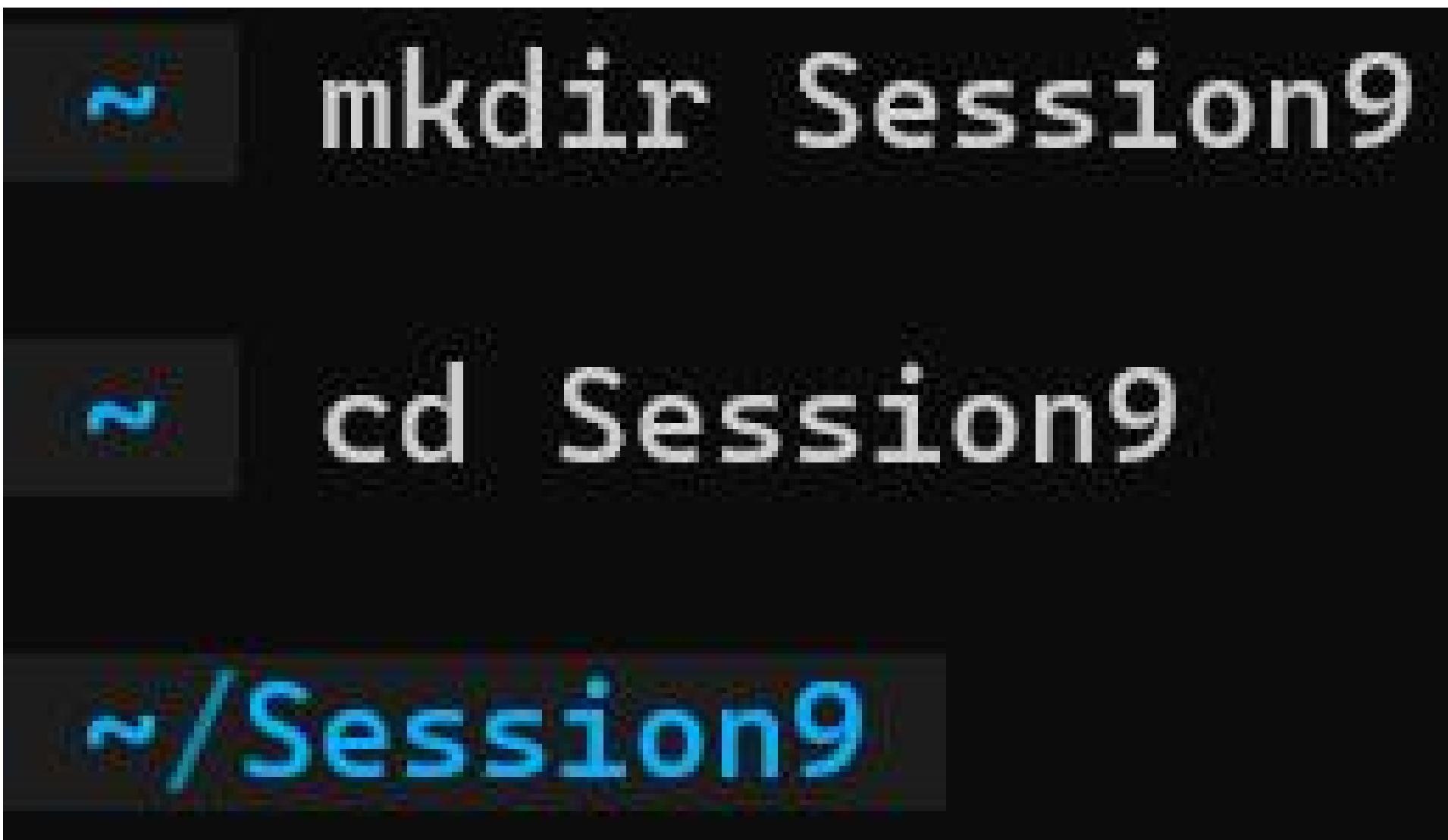
```
~ node -v
:58
v23.10.0

~ npm -v
10.9.2
```

| 프로젝트 세팅

React app을 설치해봅시다

1. 작업할 디렉토리 진입



A screenshot of a terminal window with a dark background. It shows three lines of text: 'mkdir Session9' in white, 'cd Session9' in white, and a blue cursor at the beginning of the path '~/Session9'.

```
mkdir Session9
cd Session9
~/Session9
```

| 프로젝트 세팅

React app을 설치해봅시다

2. Vite로 React 프로젝트 생성

⇒ **npm create vite@latest**

```
~/Session9$ npm create vite@latest
```

| 프로젝트 세팅

React app을 설치해봅시다

3-1. 프로젝트 명 설정

⇒ chat-project 입력하기

```
~/Session9  npm create vite@latest

> npx
> create-vite

? Project name: > chat-project|
```

| 프로젝트 세팅

React app을 설치해봅시다

3-2. 프레임워크 설정

⇒ React로 이동 후 Enter 키 누르기

```
✓ Project name: ... chat-project
? Select a framework: » - Use arrow-keys. Return to submit.
  Vanilla
  Vue
>  React
  Preact
  Lit
  Svelte
  Solid
  Qwik
  Angular
  Others
```

| 프로젝트 세팅

React app을 설치해봅시다

3-3. variant(템플릿) 설정

⇒ **JavaScript + SWC**로 이동 후 Enter 키 누르기

```
✓ Project name: ... chat-project
✓ Select a framework: » React
? Select a variant: » - Use arrow-keys. Return to submit.
    TypeScript
    TypeScript + SWC
    JavaScript
>   JavaScript + SWC
    React Router v7 ↵
```

| 프로젝트 세팅

React app을 설치해봅시다

3-4. npm 설치

⇒ 밑에 나온 세 개의 명령어를 순차적으로 입력

```
Scaffolding project in C:\Users\1209k\Session9\chat-project...
```

```
Done. Now run:
```

```
cd chat-project  
npm install  
npm run dev
```

| 프로젝트 세팅

React app을 설치해봅시다

3-5. 서버 실행

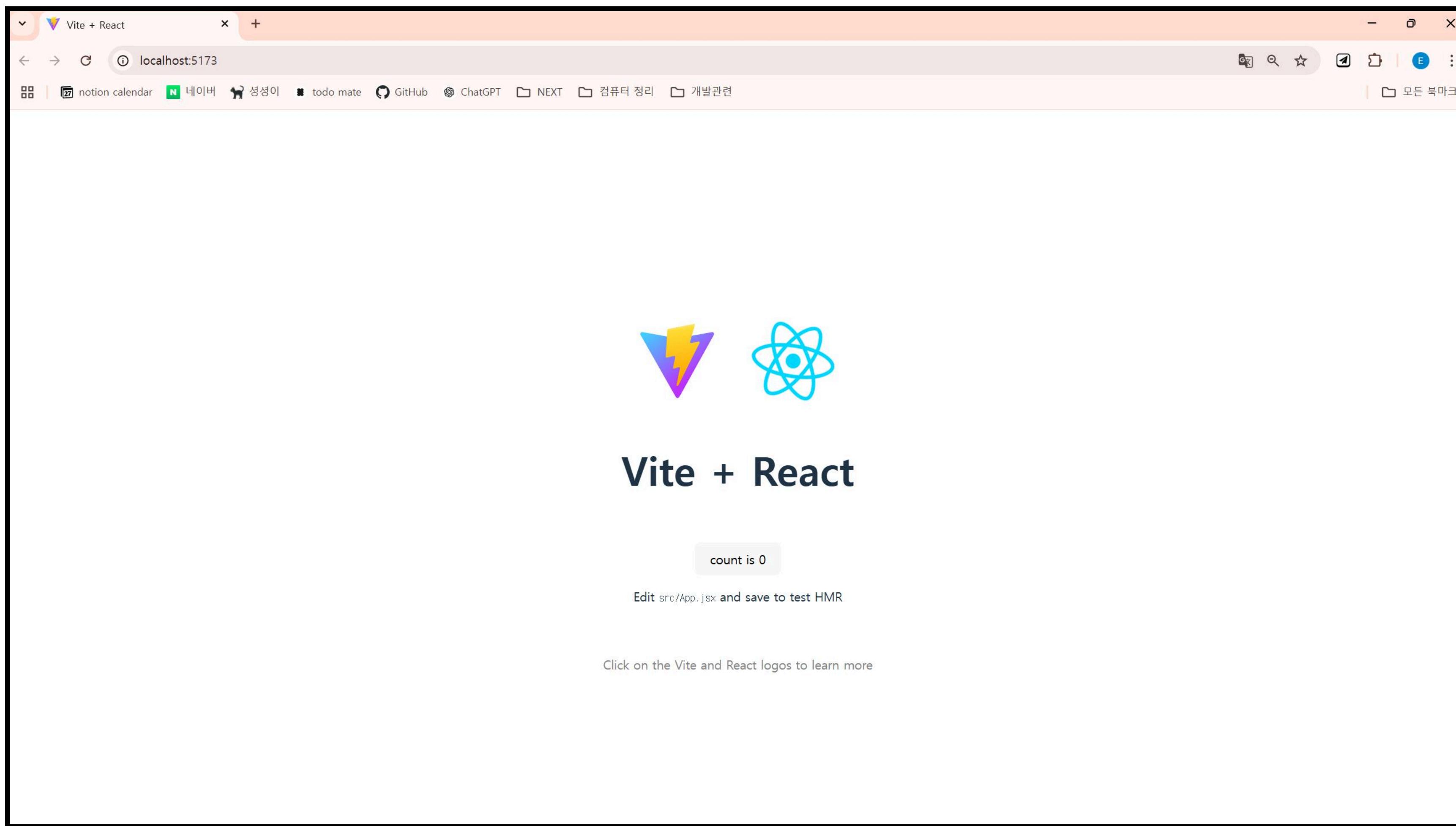
⇒ 로컬 주소를 ctrl(cmd) + 우클릭

```
VITE v6.0.11  ready in 219 ms

→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

| 프로젝트 세팅

이 화면이 뜨면 성공!!



| JSX 문법

Javascript Syntax eXtension

**JavaScript 코드 안에서 HTML과 가장 유사한
문법을 사용하게 해주는 React의 확장 문법**

| JSX 문법

1. HTML과 유사한 문법

```
const element = <h1>Hello, JSX!</h1>;
```

| JSX 문법

1. HTML과 유사한 문법

```
const element = <h1>Hello, JSX!</h1>;
```

2. JavaScript 표현식 사용 가능

```
const name = "React";
const element = <h1>Hello, {name}!</h1>;
```

| JSX 문법

1. HTML과 유사한 문법

```
const element = <h1>Hello, JSX!</h1>;
```

2. JavaScript 표현식 사용 가능

```
const name = "React";
const element = <h1>Hello, {name}!</h1>;
```

3. 가상 Dom 생성

```
const element = <h1>Hello, JSX!</h1>;
// 실제로는 아래 코드와 동일:
const element = React.createElement('h1', null, 'Hello, JSX!');
```

| JSX 규칙

1. 반드시 부모 요소가 감싸야 한다.

에러 케이스

```
function App() {
  return (
    <div>Hello</div>
    <div>Happy Session9!</div>
  )
}
```

| JSX 규칙

1. 반드시 부모 요소가 감싸야 한다.

에러 케이스

```
function App() {  
  return (  
    <div>Hello</div>  
    <div>Happy Session9!</div>  
  )  
}
```

정상 코드1. <div></div>

```
function App() {  
  return (  
    <div>  
      <div>Hello</div>  
      <div>Happy Session9!</div>  
    </div>  
  );  
}
```

| JSX 규칙

1. 반드시 부모 요소가 감싸야 한다.

에러 케이스

```
function App() {  
  return (  
    <div>Hello</div>  
    <div>Happy Session9!</div>  
  )  
}
```

정상 코드1. <div></div>

```
function App() {  
  return (  
    <div>  
      <div>Hello</div>  
      <div>Happy Session9!</div>  
    </div>  
  );  
}
```

정상 코드2. <Fragment></Fragment>

```
function App() {  
  return (  
    <Fragment>  
      <div>Hello</div>  
      <div>Happy Session9!</div>  
    </Fragment>  
  );  
}
```

| JSX 규칙

1. 반드시 부모 요소가 감싸야 한다.

에러 케이스

```
function App() {  
  return (  
    <div>Hello</div>  
    <div>Happy Session9!</div>  
  )  
}
```

정상 코드1. <div></div>

```
function App() {  
  return (  
    <div>  
      <div>Hello</div>  
      <div>Happy Session9!</div>  
    </div>  
  );  
}
```

정상 코드2. <Fragment></Fragment>

```
function App() {  
  return (  
    <Fragment>  
      <div>Hello</div>  
      <div>Happy Session9!</div>  
    </Fragment>  
  );  
}
```

정상 코드3. <></>

```
function App() {  
  return (  
    <>  
      <div>Hello</div>  
      <div>Happy Session9!</div>  
    </>  
  );  
}
```

| JSX 규칙

2. class 대신 className

```
function App() {
  return (
    <div className="testClass">Happy Session9!</div>
  );
}
```

| JSX 규칙

2. class 대신 className

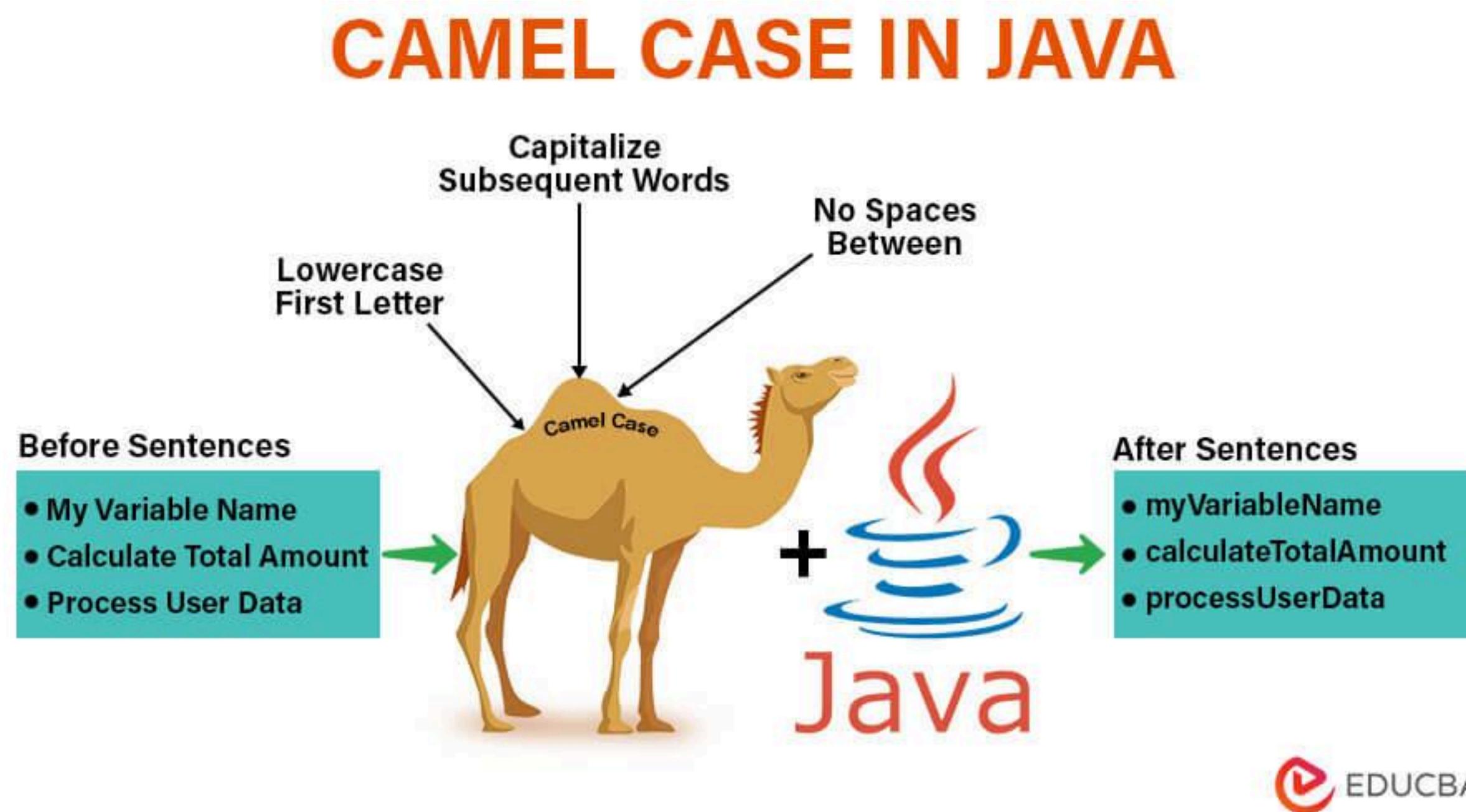
```
function App() {  
  return (  
    <div className="testClass">Happy Session9!</div>  
  );  
}
```

```
function App() {  
  return <div class="testClass">Happy Session9!</div>;  
}
```



| JSX 규칙

3. Camel Case를 사용해야 한다.



여러 단어로 구성된 경우,
첫 단어는 소문자, 다음 단어의 첫 글자는 대문자로

e.g.) `backgroundColor`, `onClick`, `fontSize`

| JSX 규칙

그 밖에 다양한 케이스(case) 스타일

케이스 스타일이란?

프로그래밍에서 변수나 클래스 명에 **공백**을 지우고 이름을 짓기 위해 사용하는 문자 표기 방식

| 스타일명 | 표기 방식 | 예제 | 주로 사용되는 곳 |
|------------|---------------------------|-----------|--------------------------------|
| camelCase | 첫 단어는 소문자, 이후 단어는 대문자로 시작 | userName | 변수명, 함수명 (JavaScript, Java) |
| PascalCase | 모든 단어의 첫 글자를 대문자로 시작 | UserName | 클래스명, 생성자 (C#, Java) |
| snake_case | 모든 단어를 소문자로 쓰고 _로 연결 | user_name | 변수명, 함수명 (Python, SQL) |
| kebab-case | 모든 단어를 소문자로 쓰고 -로 연결 | user-name | URL, CSS 클래스 |

| JSX 규칙

4. 그 외의 규칙들..

주석 달 때, `{/*...*/}`

```
function App() {
  return (
    <>
    {/* 주석은 이렇게 달아요. */}
    <div>Hello</div>
    <div>Happy Session9!</div>
  );
}
```

HTML 안에서 style 지정할 때, `style = {{style}}`

```
function App() {
  return (
    <>
    {/* style은 이렇게 먹여요. */}
    <div style={{color: 'red', fontSize: "12px"}>Hello</div>
    <div>Happy Session9!</div>
  );
}
```

| 폴더 구조

```
▽ CHAT-PROJECT
  > node_modules
  > public
  > src
  ◆ .gitignore
  ⓘ eslint.config.js
  ◁ index.html
  {} package-lock.json
  {} package.json
  ⓘ README.md
  ⚡ vite.config.js
```

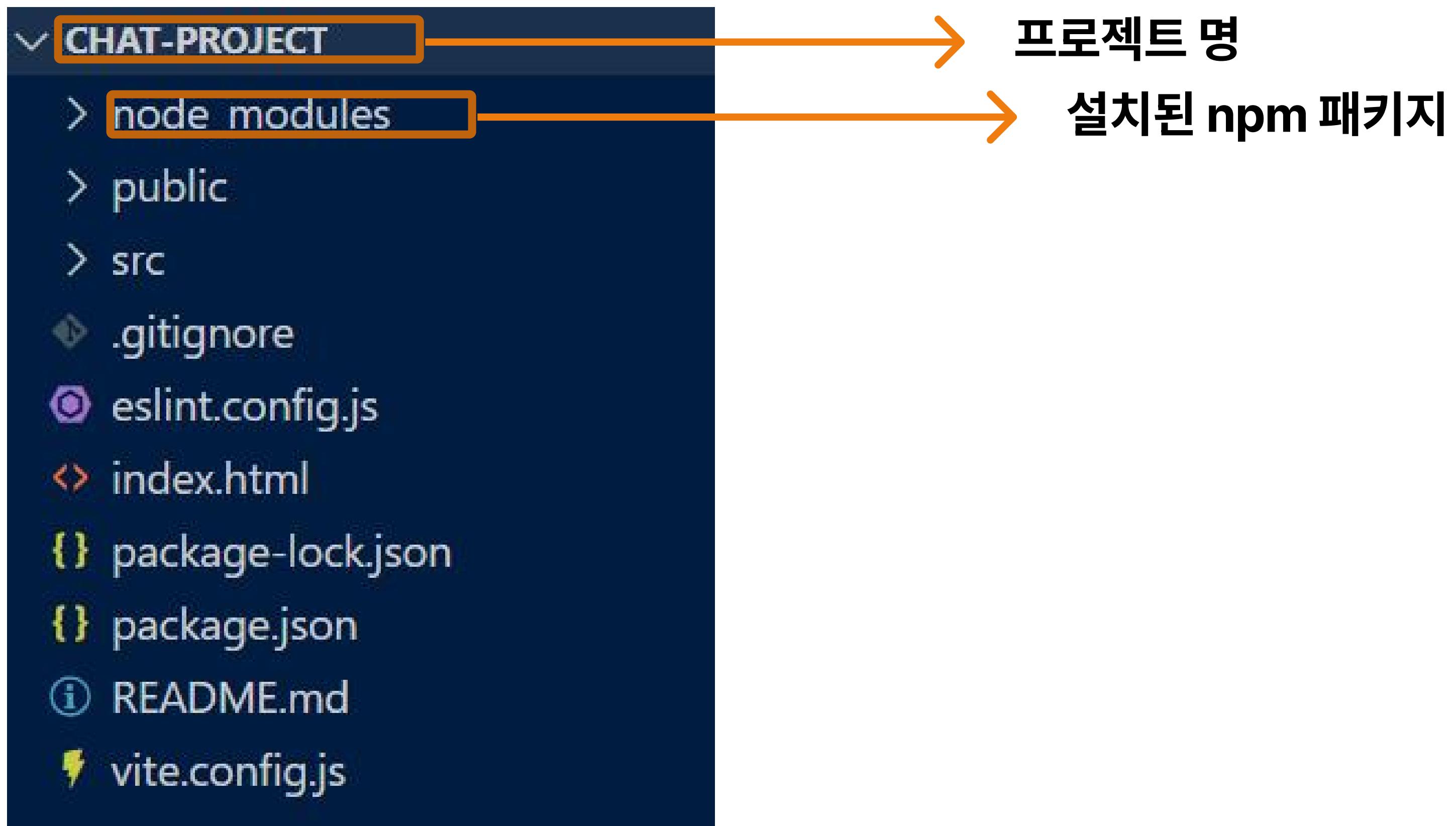
| 폴더 구조

A screenshot of a file explorer window showing the contents of a project folder named 'CHAT-PROJECT'. The folder structure is as follows:

- > node_modules
- > public
- > src
- ❖ .gitignore
- ◉ eslint.config.js
- ◇ index.html
- { package-lock.json
- { package.json
- ⓘ README.md
- ⚡ vite.config.js

프로젝트 명

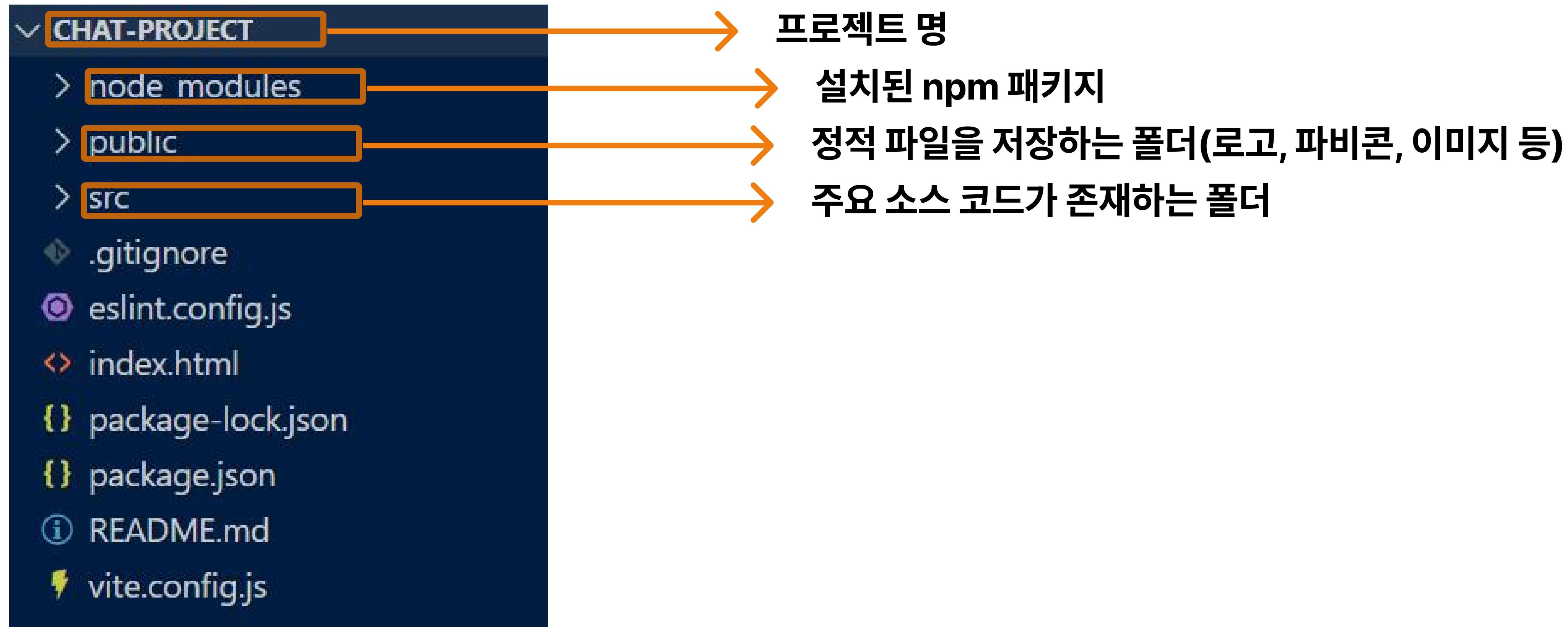
| 폴더 구조



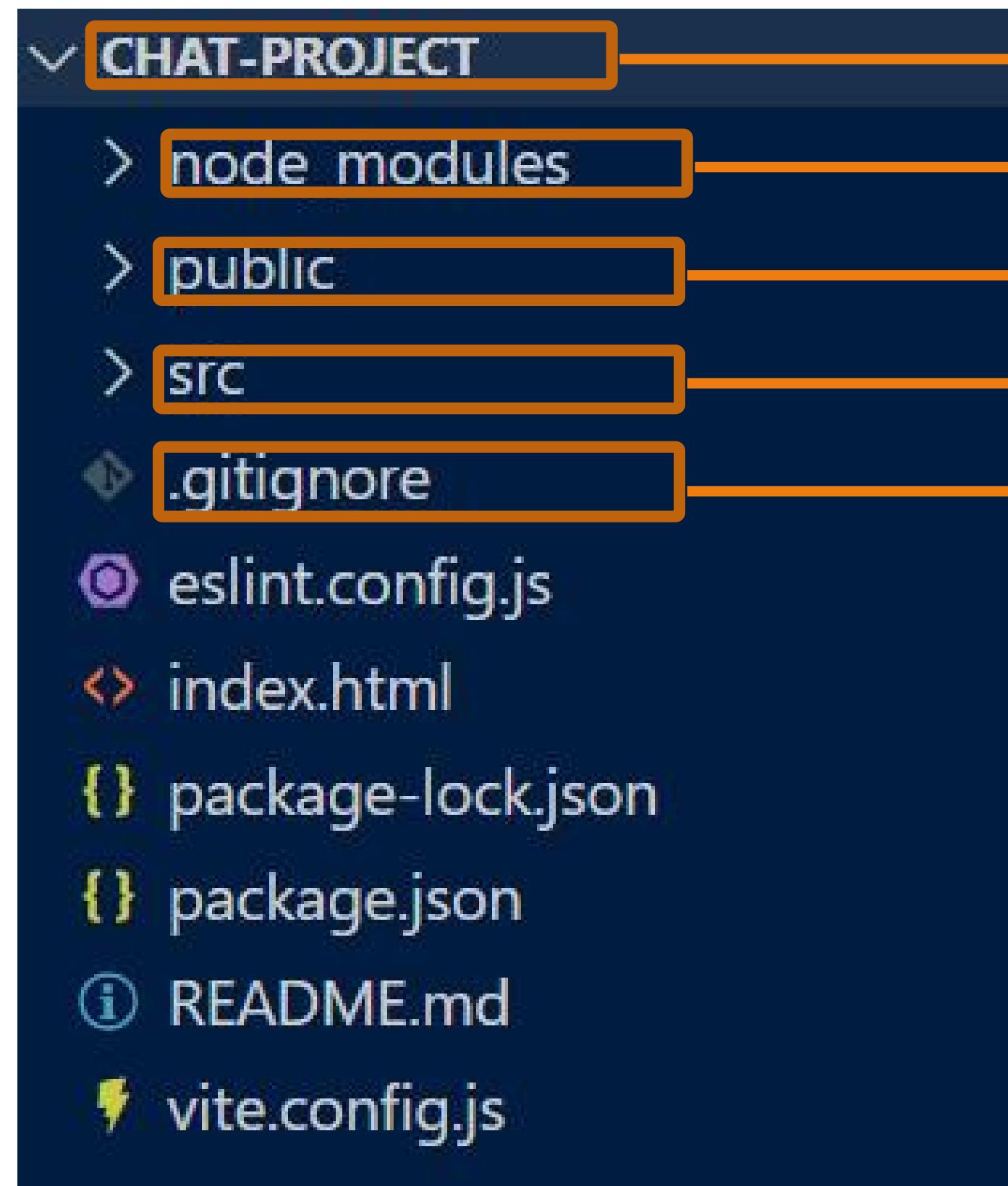
| 폴더 구조



| 폴더 구조



| 폴더 구조



프로젝트 명

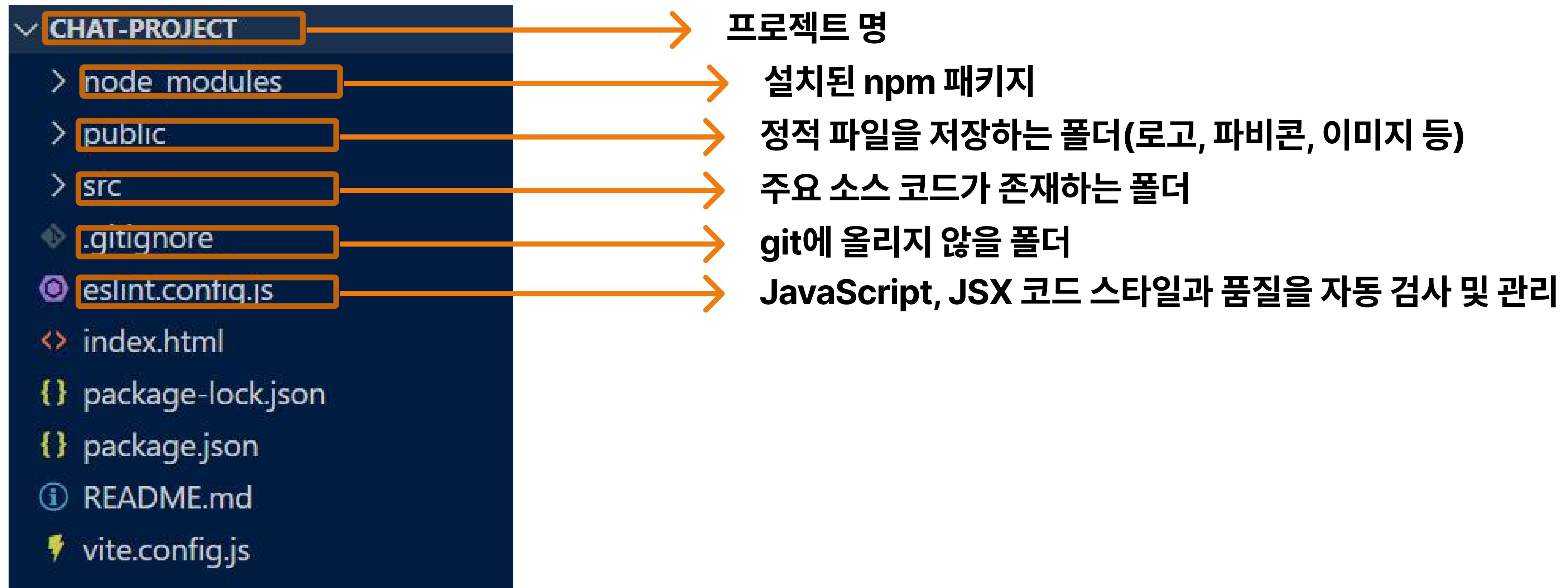
설치된 npm 패키지

정적 파일을 저장하는 폴더(로고, 파비콘, 이미지 등)

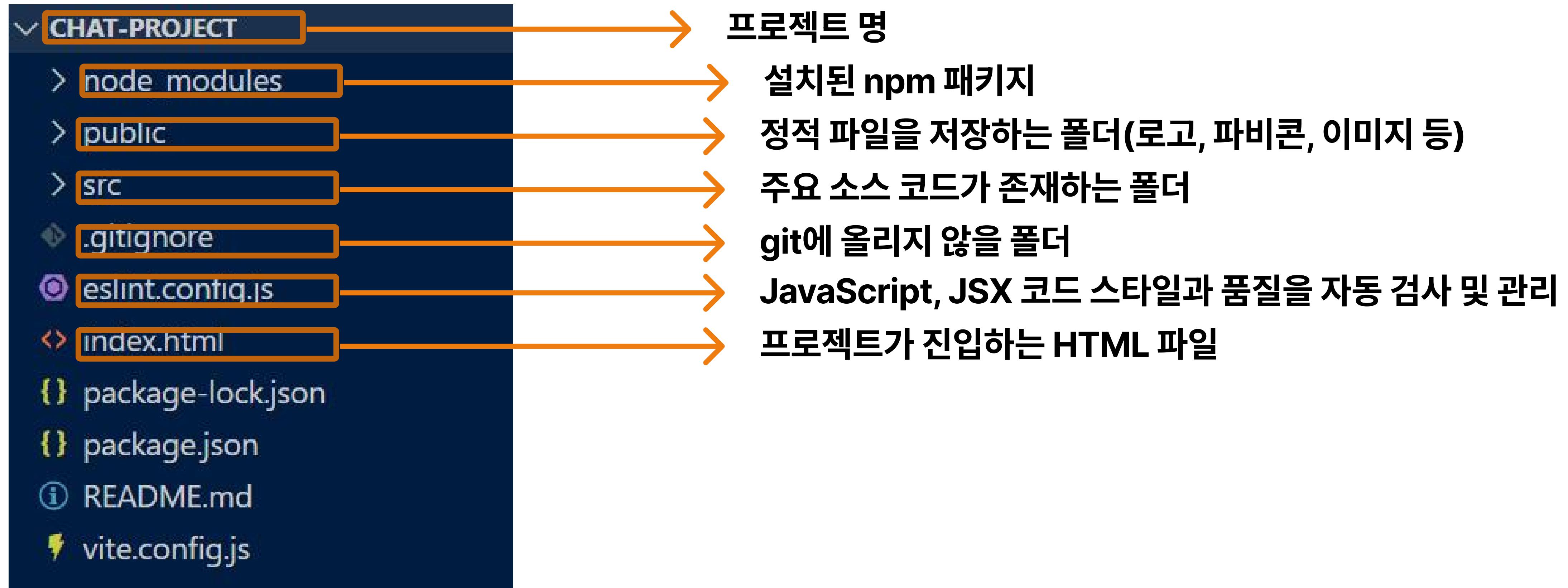
주요 소스 코드가 존재하는 폴더

git에 올리지 않을 폴더

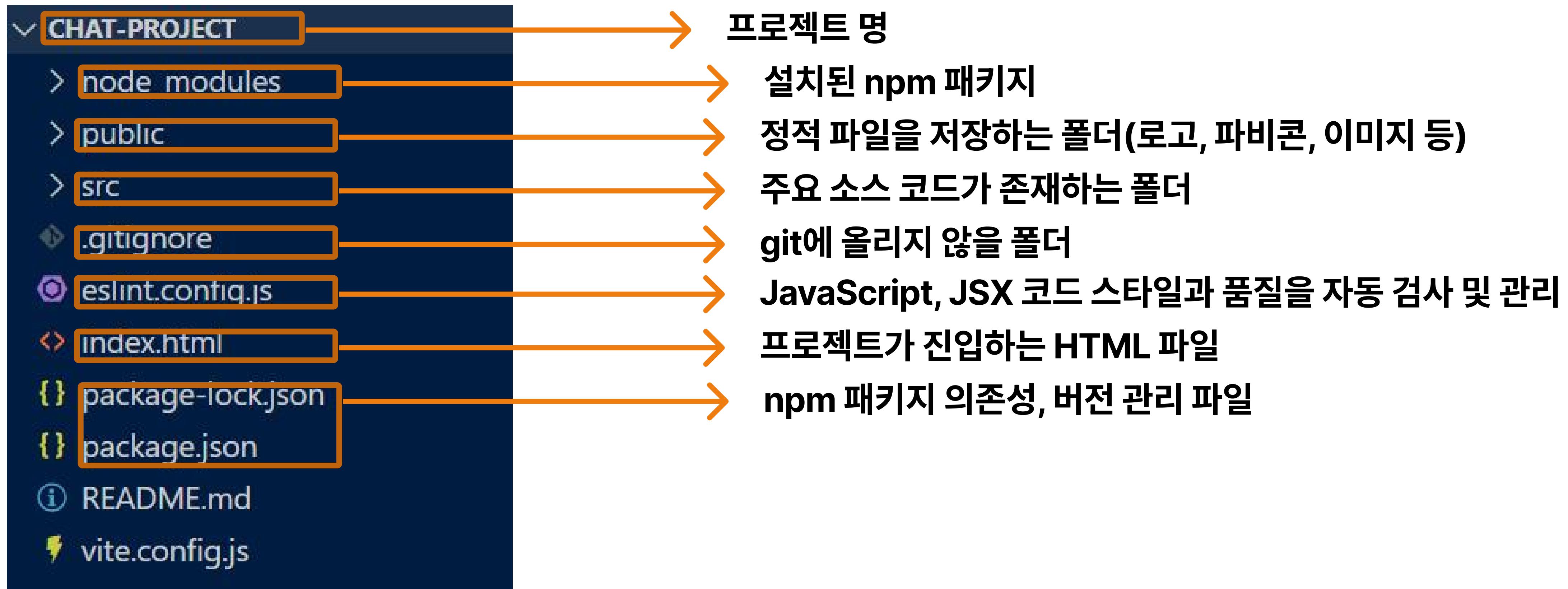
| 폴더 구조



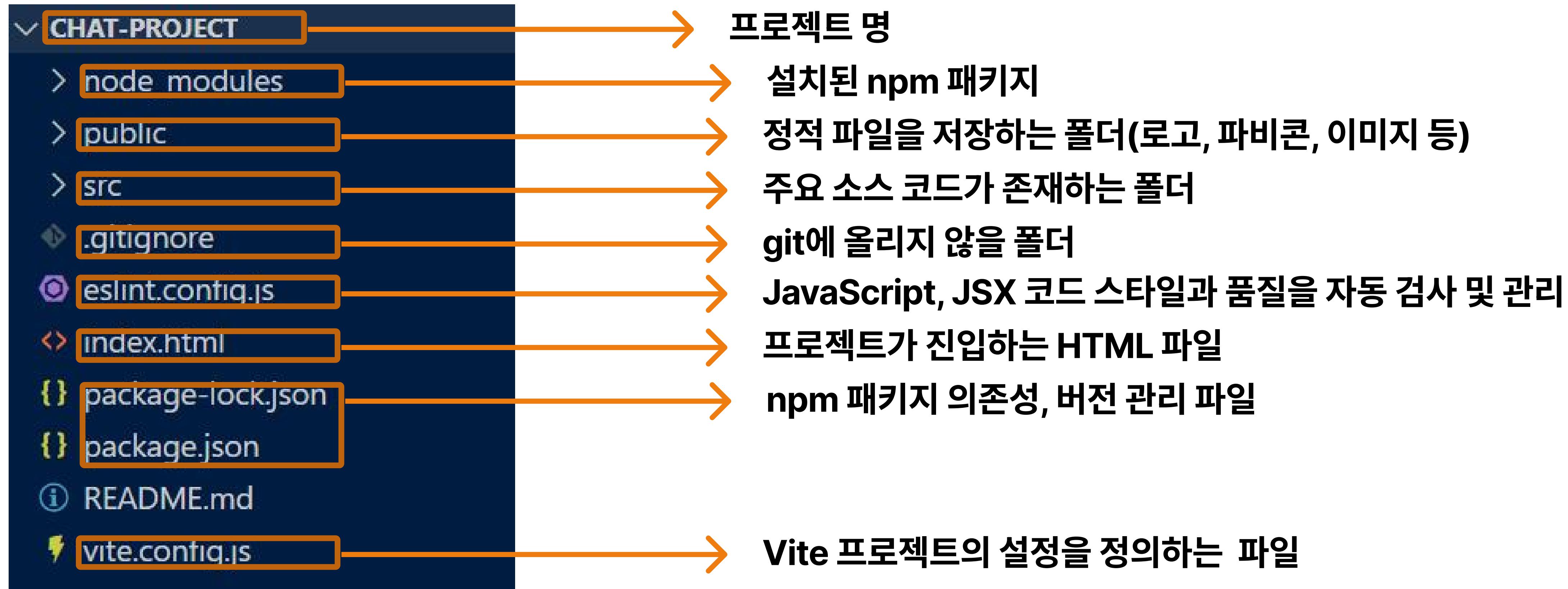
| 폴더 구조



| 폴더 구조

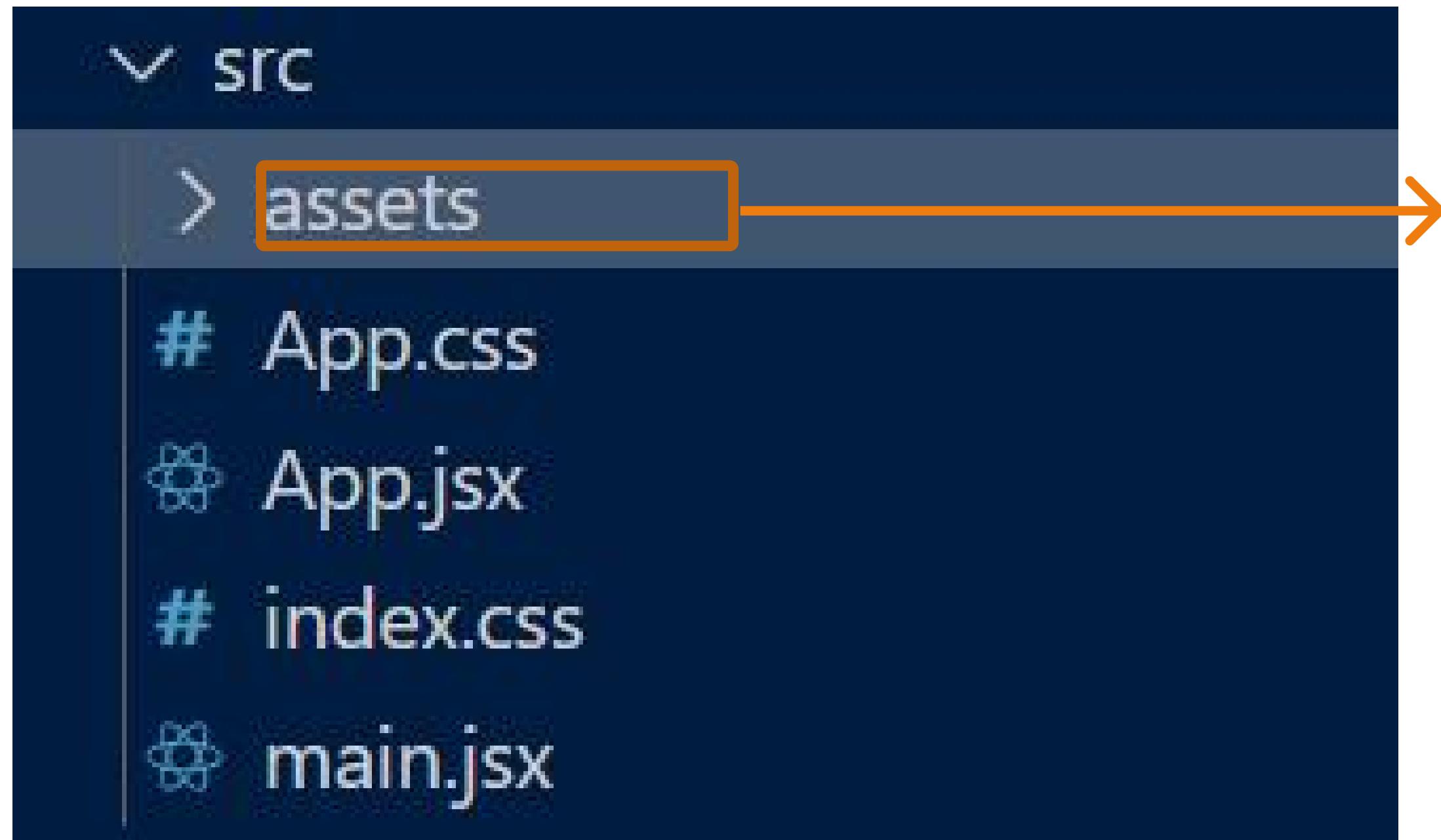


| 폴더 구조



| 폴더 구조

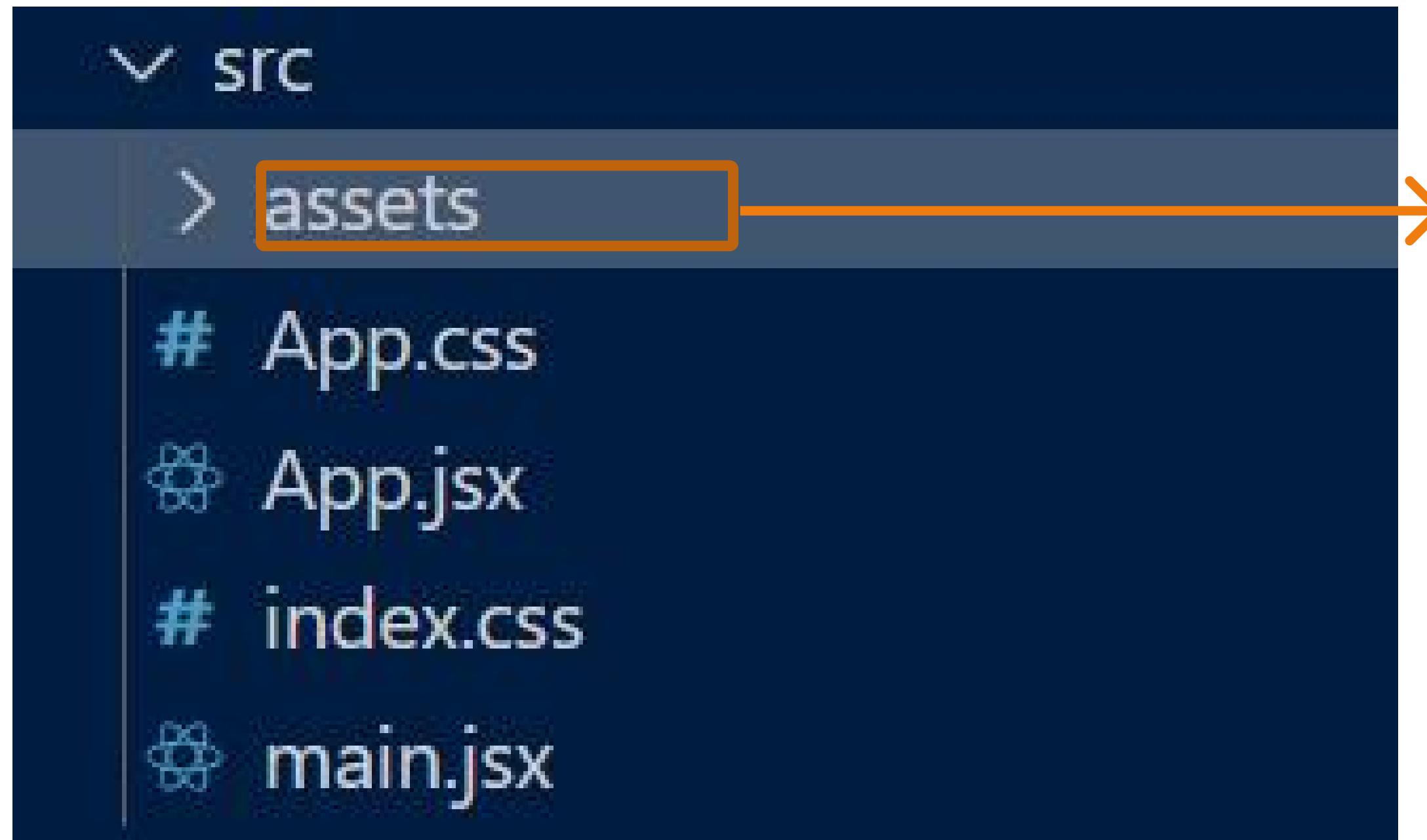
src 폴더만 집중적으로 토아봅시다



이미지, 폰트, 아이콘 등과 같은
자원 파일을 저장하는 폴더

| 폴더 구조

src 폴더만 집중적으로 토아봅시다

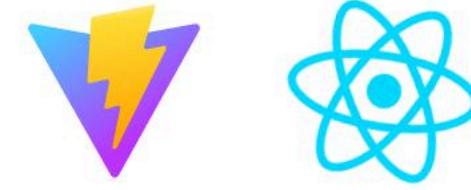


이미지, 폰트, 아이콘 등과 같은
자원 파일을 저장하는 폴더

정적 파일을 관리할 땐,
public 폴더가 권장된다!
(최상위 디렉토리로, 절대 경로
로 접근이 용이하기 때문에)

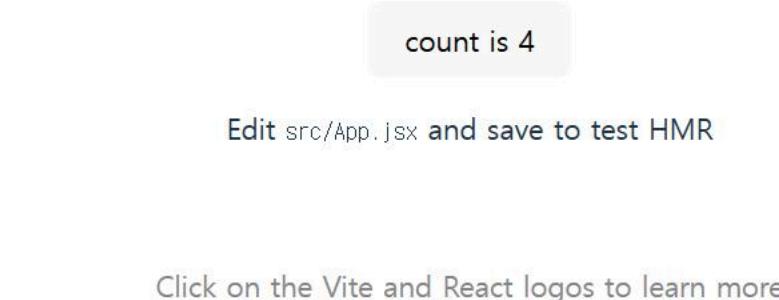
| 폴더 구조

src 폴더만 집중적으로 토아봅시다



어쩌구저쩌구

```
src
  assets
  # App.css
  App.jsx
  # index.css
  main.jsx
```

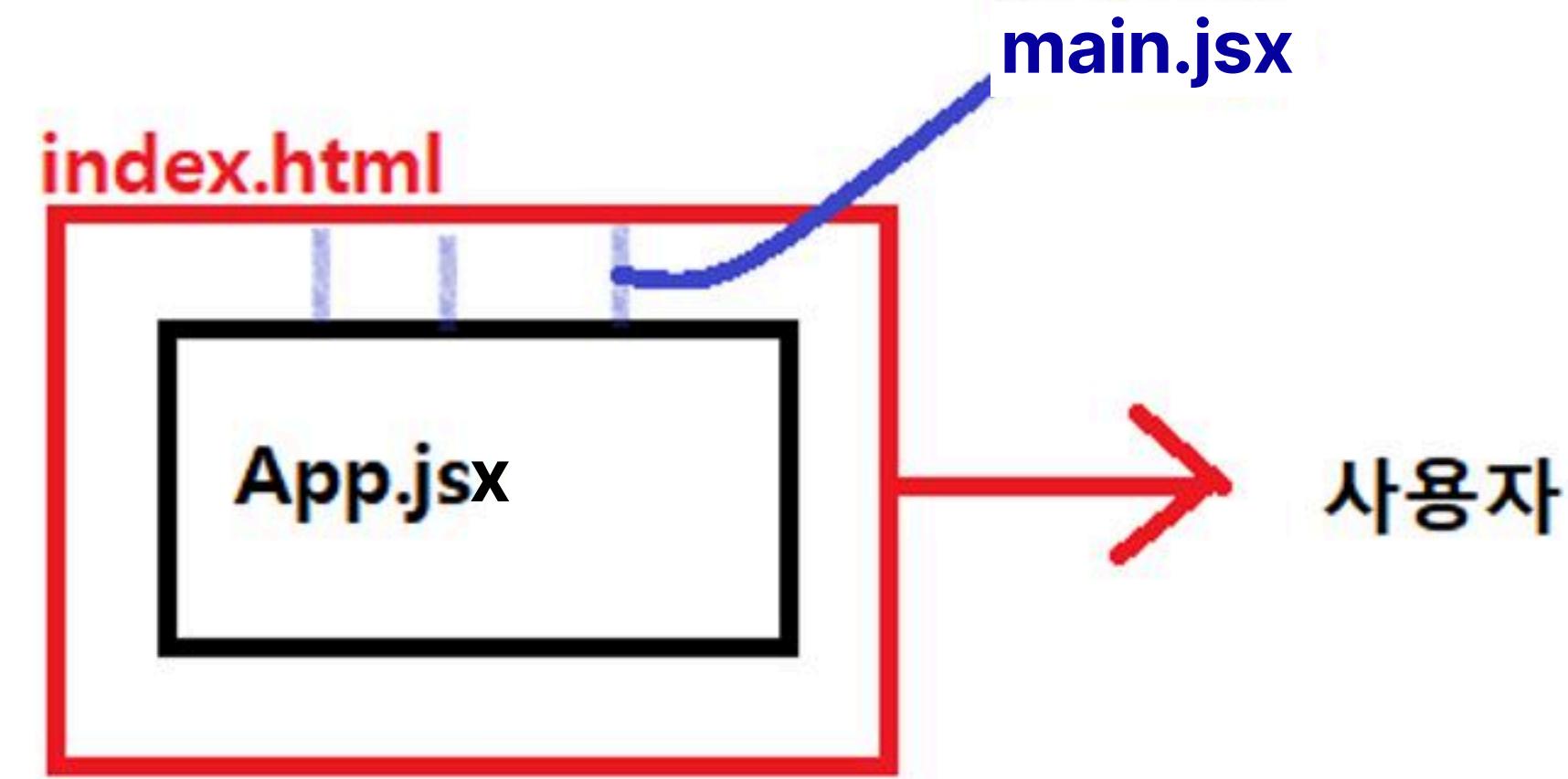
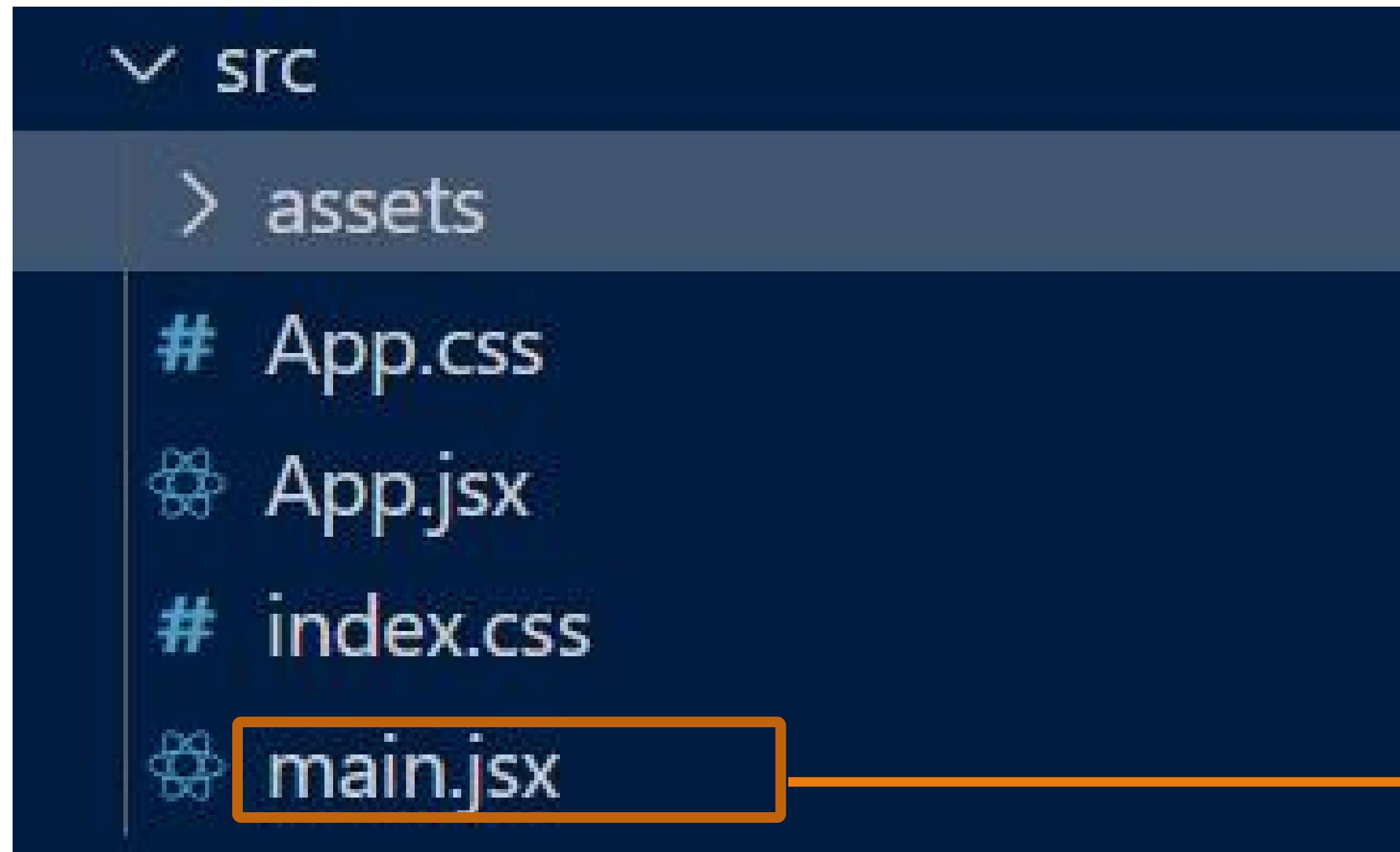


**React 애플리케이션의 최상위 컴포넌트 파일,
주요 UI 컴포넌트를 정의하고 렌더링한다**

⇒ 사용자에게 보여지는 직관적인 페이지!

폴더 구조

src 폴더만 집중적으로 토아봅시다



React 애플리케이션을 초기화하고,
React 최상위 컴포넌트인 App.jsx를
DOM에 렌더링한다!

| 렌더링 과정

그래서 index.html, main.jsx, App.jsx가 무슨 차이가 있는데요?

1. index.html

- 브라우저가 가장 먼저 읽는 파일

```
<div id="root"></div>
```

2. main.jsx

- HTML 로드 후, 실행하는 첫 번째 JSX 파일
- React, ReactDOM을 import한다.
- root div를 찾아, React 애플리케이션을 마운트한다.
- App 컴포넌트를 import 및 렌더링 한다.

```
createRoot(document.getElementById('root')).render(  
  <StrictMode>  
    <App />  
  </StrictMode>  
)
```

*마운트란? 어떠한 것을 Available 한 상태로 준비하는 것

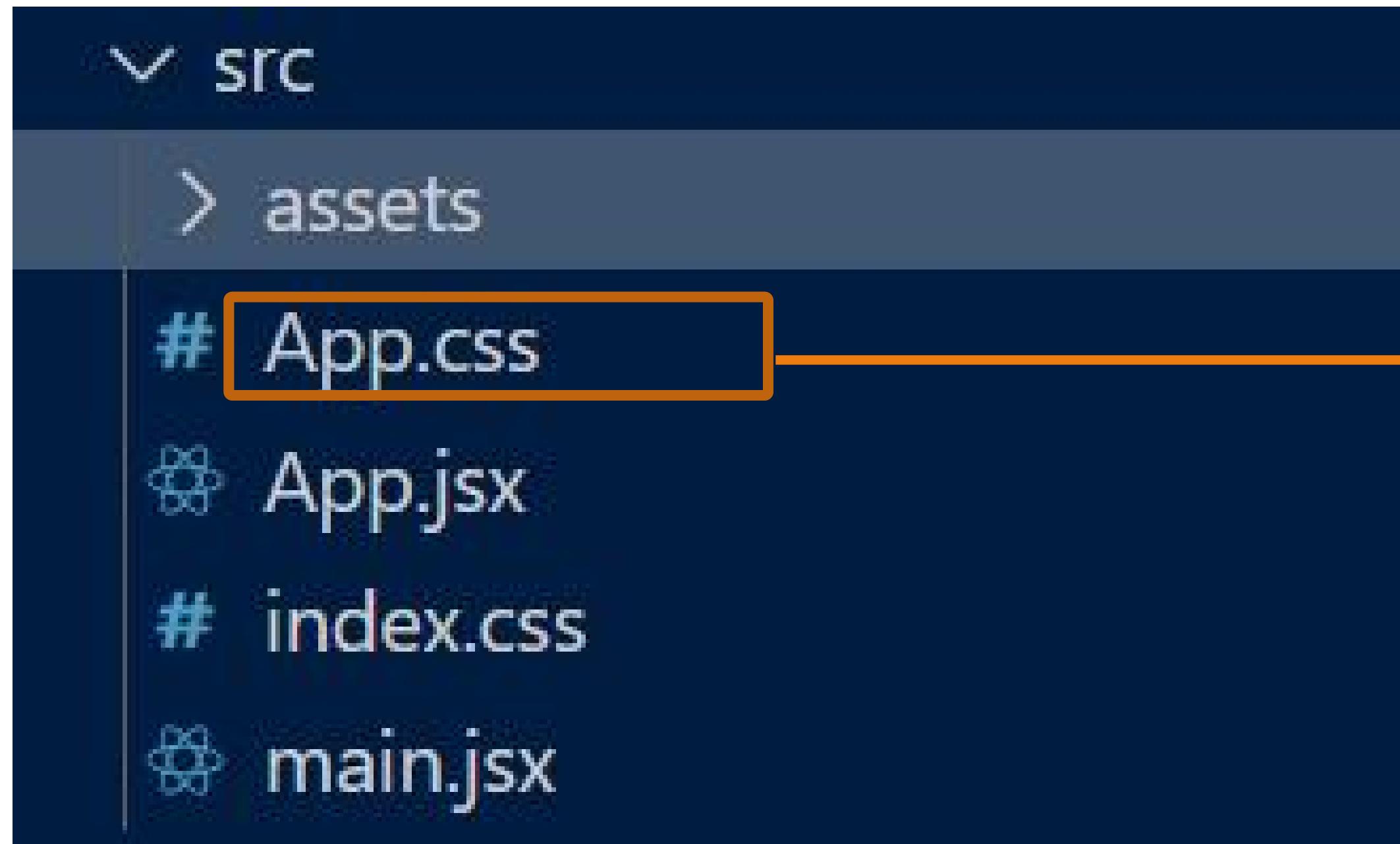
3. App.js

- 애플리케이션의 최상위 컴포넌트(<APP />)
- 이 파일 안의 모든 자식 컴포넌트들이 렌더링된다.
- 최종적으로 사용자에게 완성된 UI가 보여진다.

```
function App() {  
  const [count, setCount] = useState(0)  
  
  return (  
    <>  
    <div>  
      <a href="https://vite.dev" target="_blank">  
        <img src={viteLogo} className="logo" alt="Vite logo" />  
      </a>  
    </div>  
  )
```

| 폴더 구조

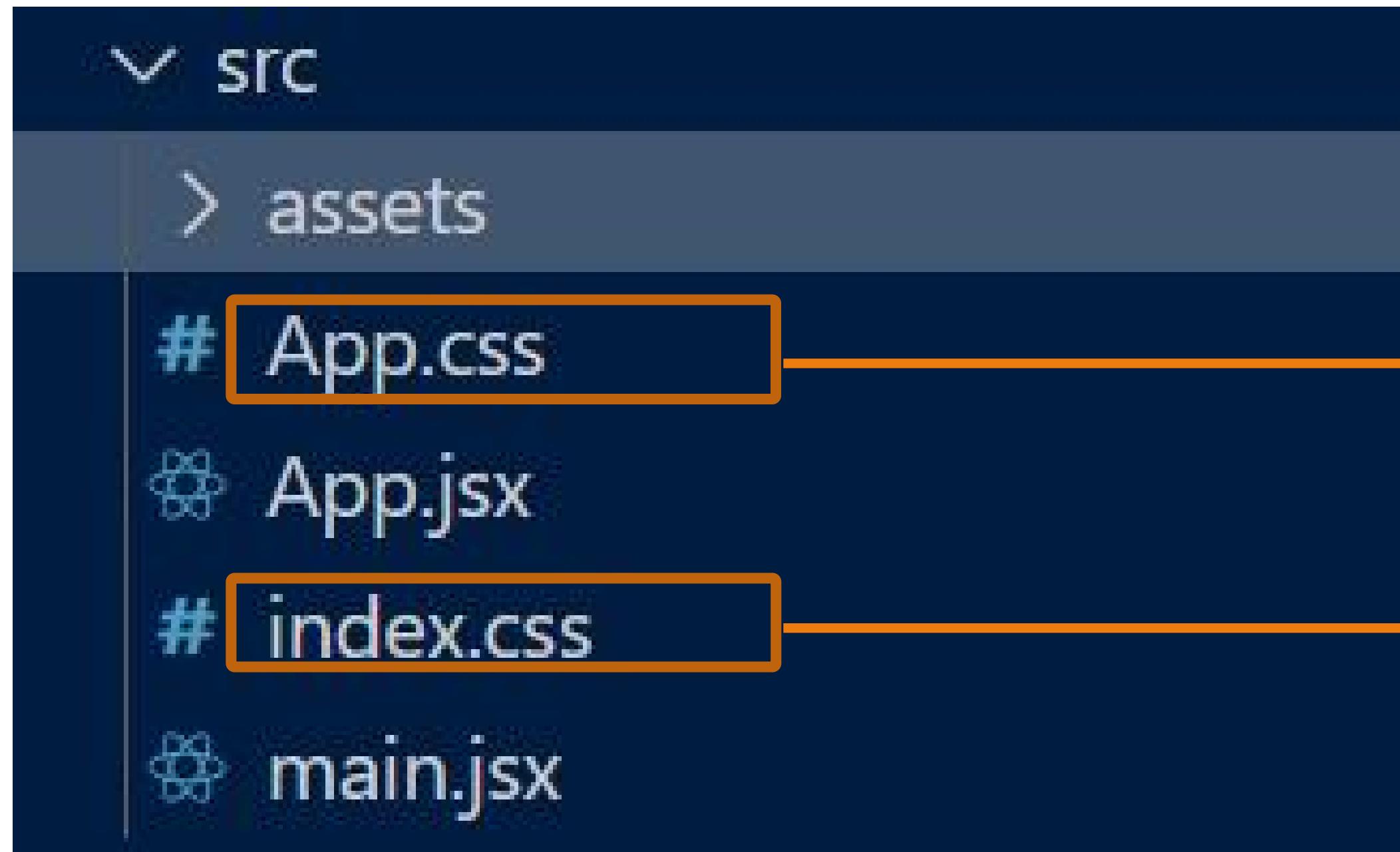
src 폴더만 집중적으로 토아봅시다



App.jsx에 import되어 App 컴포넌트에 적용되는
CSS 파일
이때, App에 있는 모든 자식 컴포넌트에 영향을 미칠 수
있다

| 폴더 구조

src 폴더만 집중적으로 토아봅시다



App.jsx에 import되어 App 컴포넌트에 적용되는 CSS 파일
이때, App에 있는 모든 자식 컴포넌트에 영향을 미칠 수 있다

전역 스타일을 정의하는 CSS 파일 (cf. global.css)
main.jsx에서 import되어 전체 애플리케이션에 적용된다

Routing

| React Routing

React Routing

URL이 서로 다른 페이지 간 이동할 수 있는 기능

| React Router

React Router

리액트에서 페이지 이동 기능(라우팅)을 구현하기 위한 패키지

| React Router Dom

React Router Dom

**React Router을 이용해 웹에서 사용하기 편한 기능을
포함해 구현된 라이브러리**

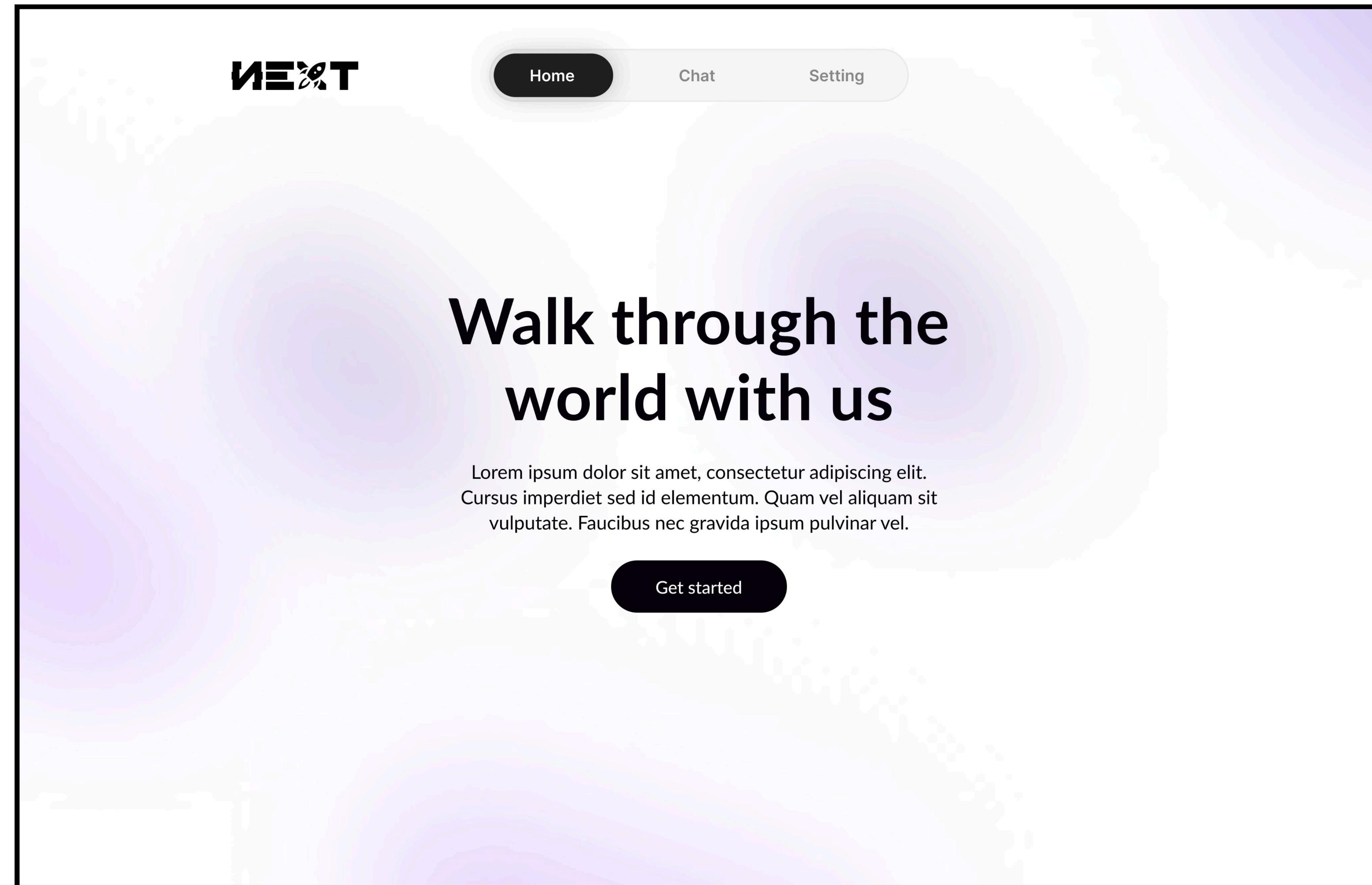
| 실습: 챗봇 프로젝트 만들기

지피티 같은 넥피티를 만들어 보아요.



미쳤습니까 휴먼?

| main 페이지

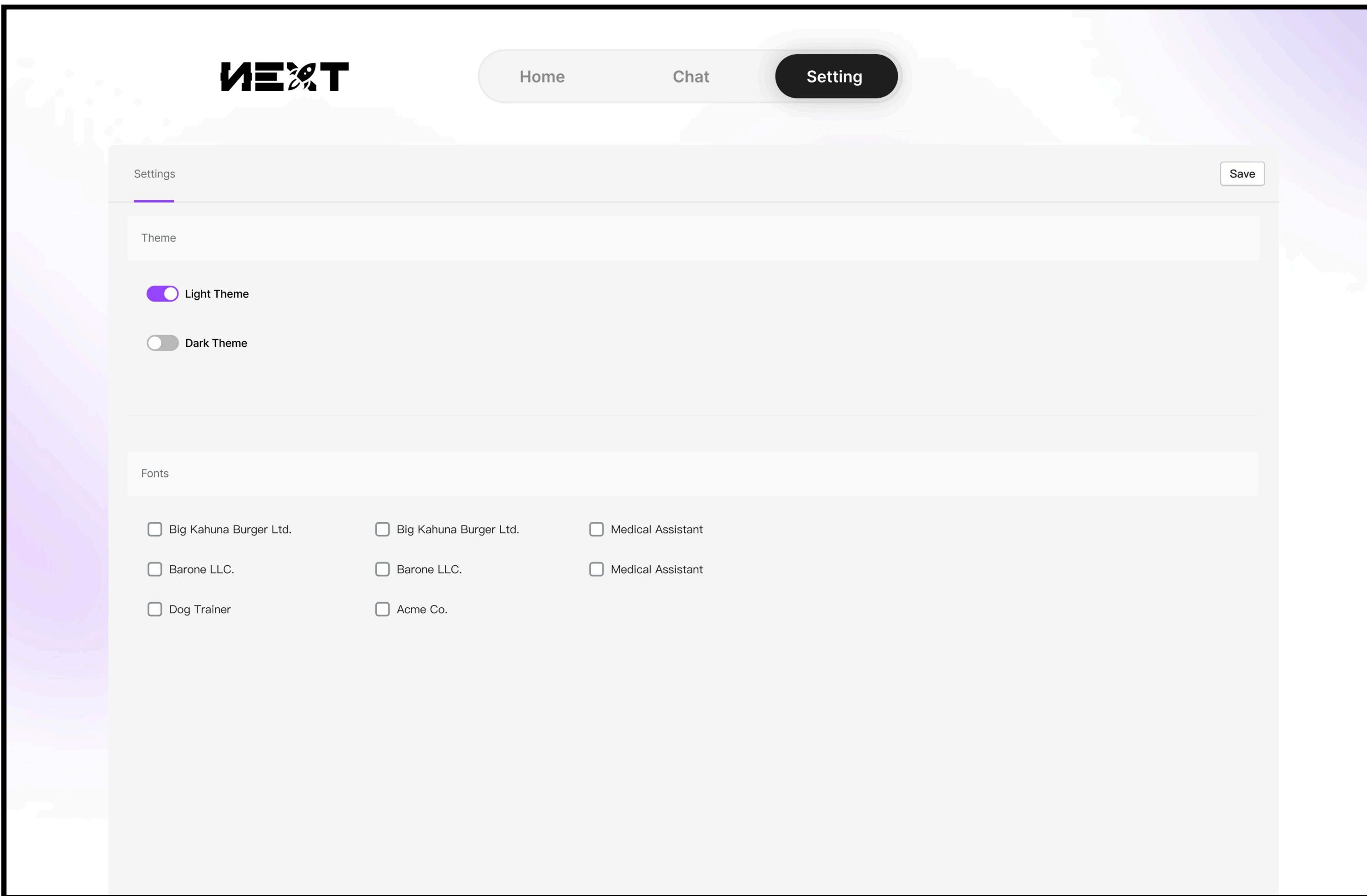


| chat 페이지

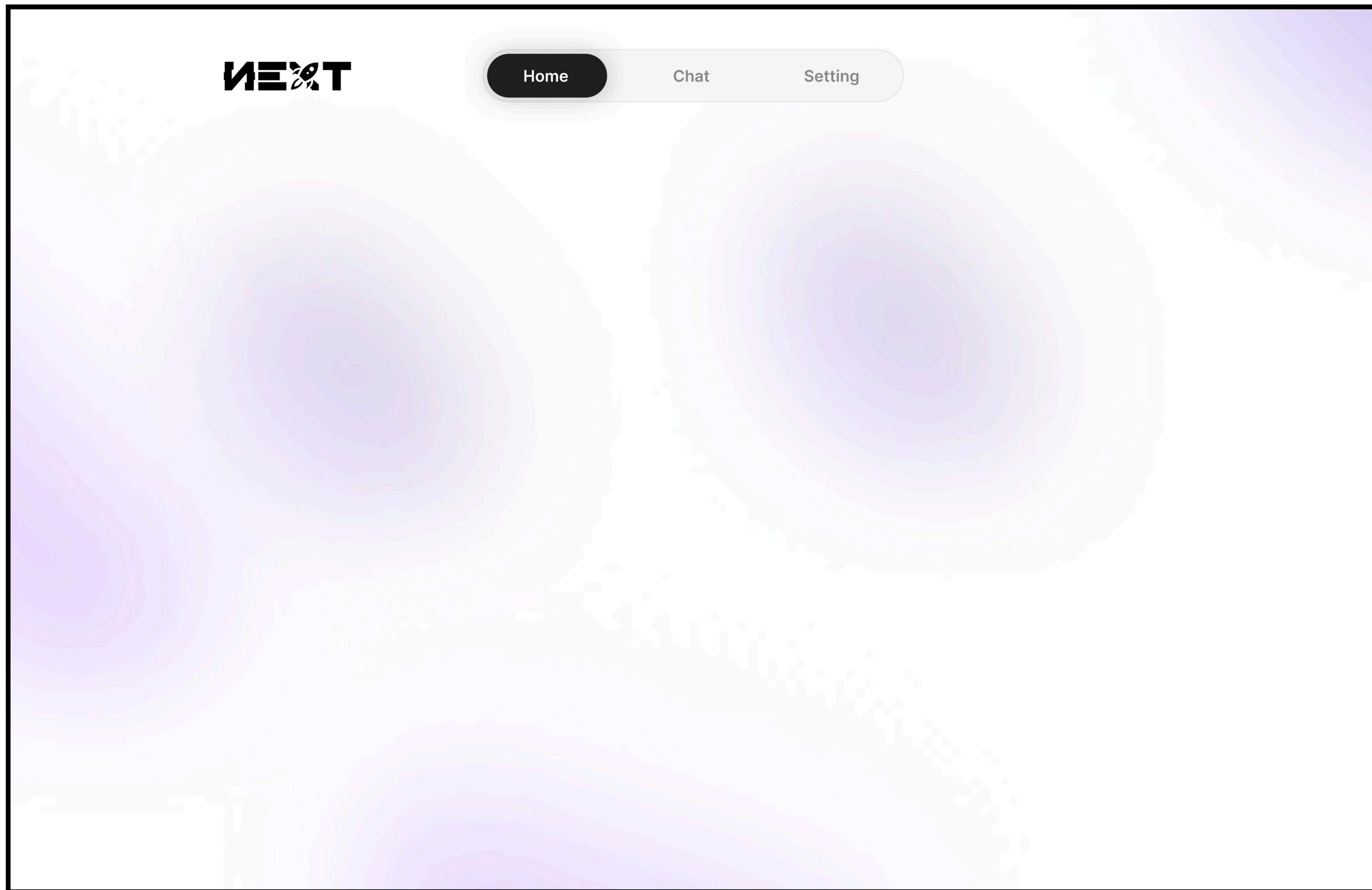
The image displays four screenshots of the NEXT AI chat application interface, illustrating its design and features.

- Home Screen:** Shows the NEXT logo at the top. Below it is a large, semi-transparent circular graphic with the text "How can we assist you today?". A list of "Recent Chats" is visible on the left, each with a small icon and a truncated message. At the bottom is a dark footer bar with "User Profile" and settings icons, and a central input field with a placeholder "Type your prompt here ...".
- Recent Chats View:** Shows a list of recent chats on the left. Each item has a checkbox icon and a truncated message. On the right, there's a large input field with a placeholder "Type your prompt here" and a purple send button with a white arrow. The top navigation bar includes "Home", "Chat", and "Setting" tabs.
- Conversation View:** Shows a list of recent chats on the left. On the right, there are two large input fields, both with a placeholder "Type your prompt here" and a purple send button with a white arrow. The top navigation bar includes "Home", "Chat", and "Setting" tabs.

| setting 페이지



| layout 페이지



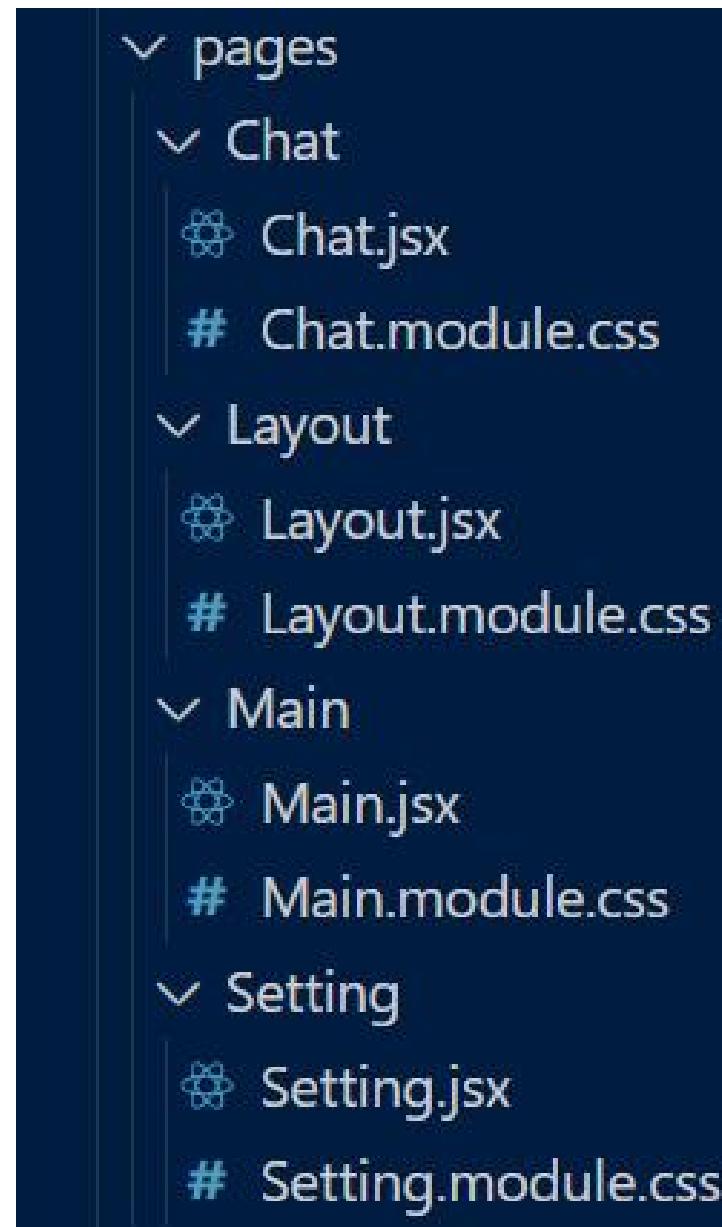
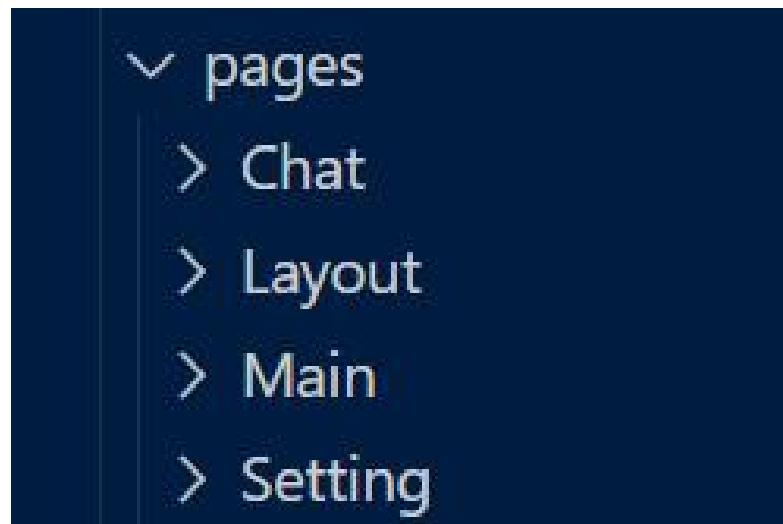
| 실습: 폴더 세팅

```
> node_modules
> public
✓ src
  > components
  > pages
  > utils
  # App.css
  ❁ App.jsx
  # index.css
  ❁ main.jsx
  ◆ .gitignore
  ○ eslint.config.js
  ◄ index.html
  { package-lock.json
  { package.json
  ⓘ README.md
  ⚡ vite.config.js
```

1. src > components 폴더 생성
→ 재사용 가능한 UI 컴포넌트들을 저장
2. src > pages 폴더 생성
→ 개별 페이지를 구성하는 컴포넌트 파일을 저장
3. src > utils 폴더 생성
→ 반복적으로 사용되는 기능이나 로직을 함수화해서 저장

| 실습: pages

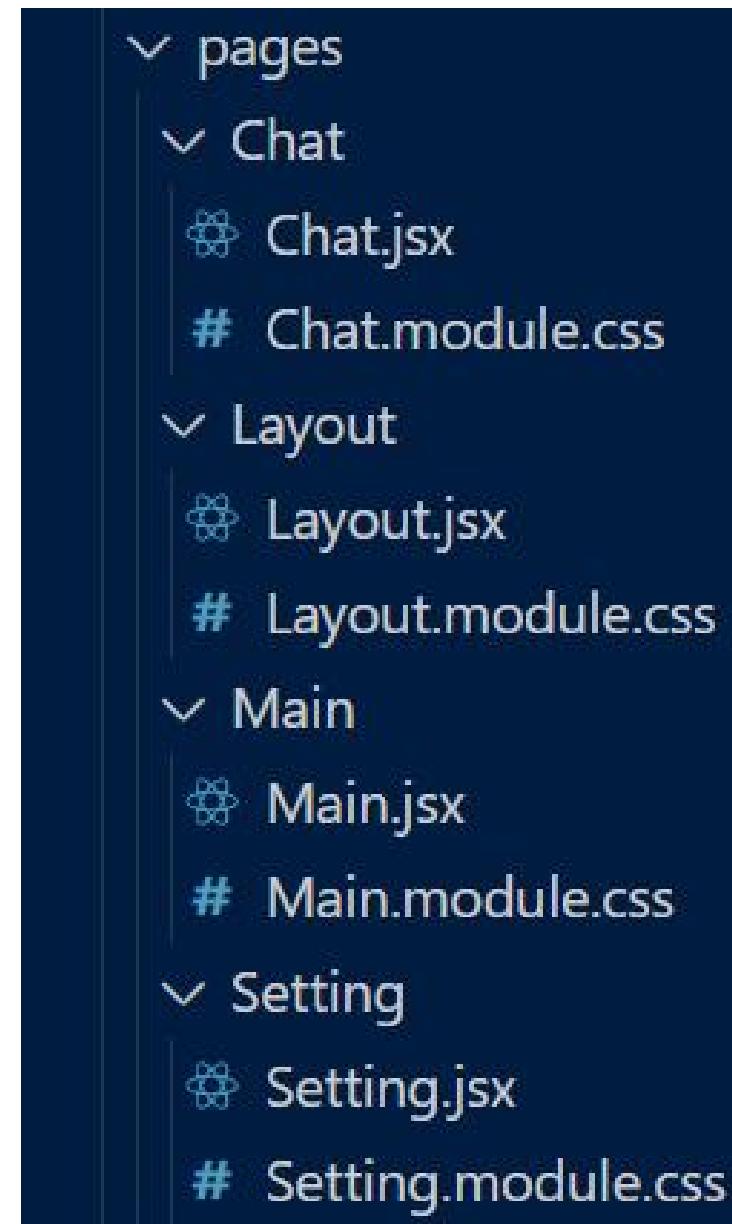
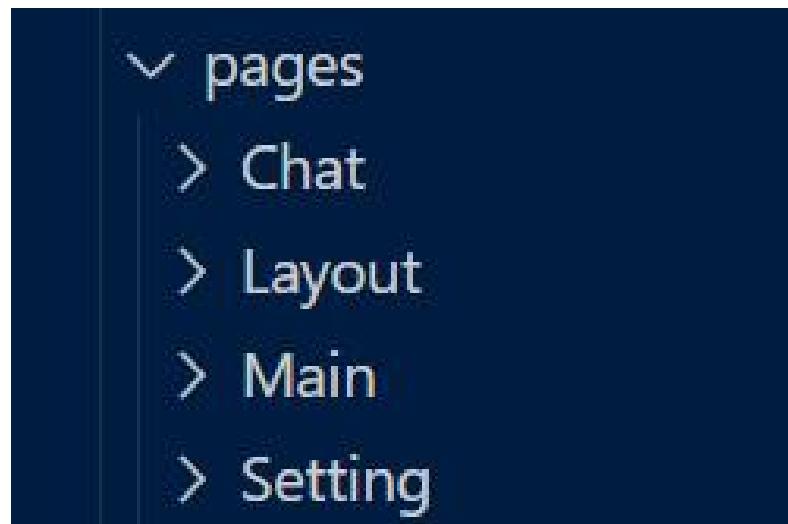
react-router-dom도 써봅시다



1. npm install react-router-dom
2. src / pages 하위에 Chat, Layout, Main, Setting 폴더 만들기
3. 각 폴더 밑에 XXX.jsx, XXX.module.css 파일 만들기

| 실습: pages

react-router-dom도 써봅시다



module.css 파일이란?

각각의 컴포넌트에 고유한 스타일을 적용

- 일반 CSS: 모든 컴포넌트가 같은 스타일을 공유해서 꼬일 수 있음
- module.css: 각 컴포넌트가 자신만의 스타일을 가져서 스타일이 섞이지 않음

| 실습: pages

react-routing도 해봅시다.

```
src > pages > Main > ❁ Main.jsx > ...
1  import React from 'react';
2  import './Main.module.css';
3
4  const Main = () => {
5    return (
6      <div>
7        <h1>Main</h1>
8      </div>
9    );
10 };
11
12 export default Main;
13
```

각 페이지에 대해서

1. React 임포트하기
2. module.css 임포트하기
3. 어떤 페이지인지 h1 태그로 명시하기

| 실습: App.jsx

react-routing도 해봅시다.

```
src > App.jsx > ...
1 import React from 'react';
2 import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
3 import Main from './pages/Main/Main';
4 import Setting from './pages/Setting/Setting';
5 import Chat from './pages/Chat/Chat';
6 import Layout from './pages/Layout/Layout';
7
8 const App = () => {
9     return (
10         <Router>
11             <Routes>
12                 <Route path="/" element={<Main />} />
13                 <Route path="/setting" element={<Setting />} />
14                 <Route path="/chat" element={<Chat />} />
15                 <Route path="/layout" element={<Layout />} />
16             </Routes>
17         </Router>
18     );
19 }
20
21 export default App;
```

1. <Router>

- URL 경로를 관리
- 라우팅을 설정하기 위한 컨테이너 역할

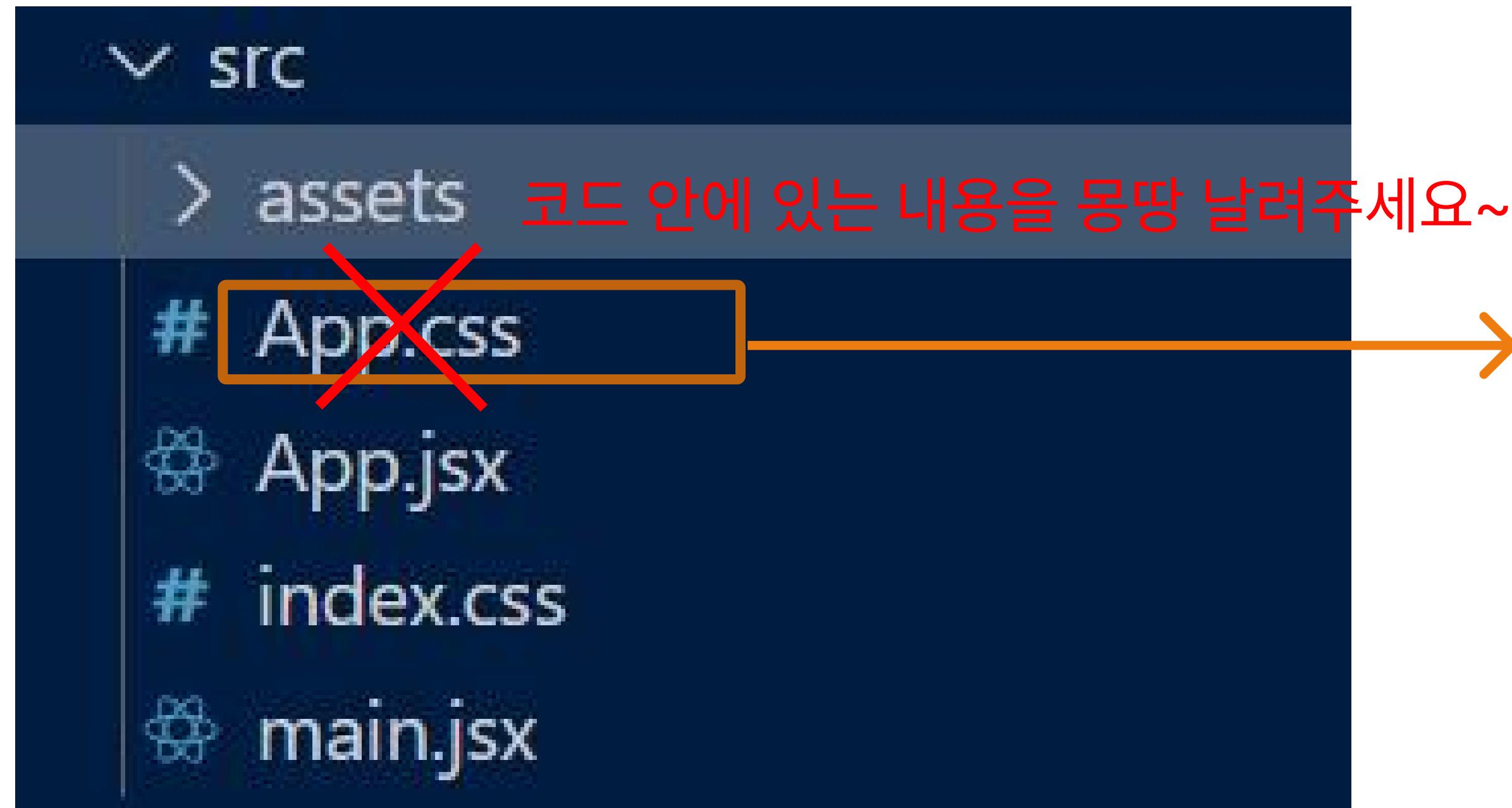
2. <Routes>

- 여러 라우트를 그룹화하는 컨테이너 역할
- Route 컴포넌트를 포함

3. <Route>

- 특정 경로(path)와 해당 경로에서 렌더링할 컴포넌트 지정

| 실습: App.css



App.jsx에 import되어 App 컴포넌트에 적용되는 CSS 파일

이때, App.jsx에 국한되지 않고, 그 하위 모든 컴포넌트에 영향을 미칠 수 있다

우리는 .module.css 파일로, 페이지마다 각각 CSS를 적용해줄 예정!!

+ 전역 스타일은 index.css에서 설정

| 실습: index.css

전역 css를 초기화 해봅시다

CSS Reset이란?

브라우저마다 다르게 적용되는 기본 스타일을 제거하여
일관된 스타일을 유지함

브라우저마다 `<h1>`, `<p>`, `` 등의 기본 스타일이 다르게 설정되어 있음

| 실습: index.css

전역 CSS를 초기화 해봅시다

CSS Reset이란?

브라우저마다 다르게 적용되는 기본 스타일을 제거하여

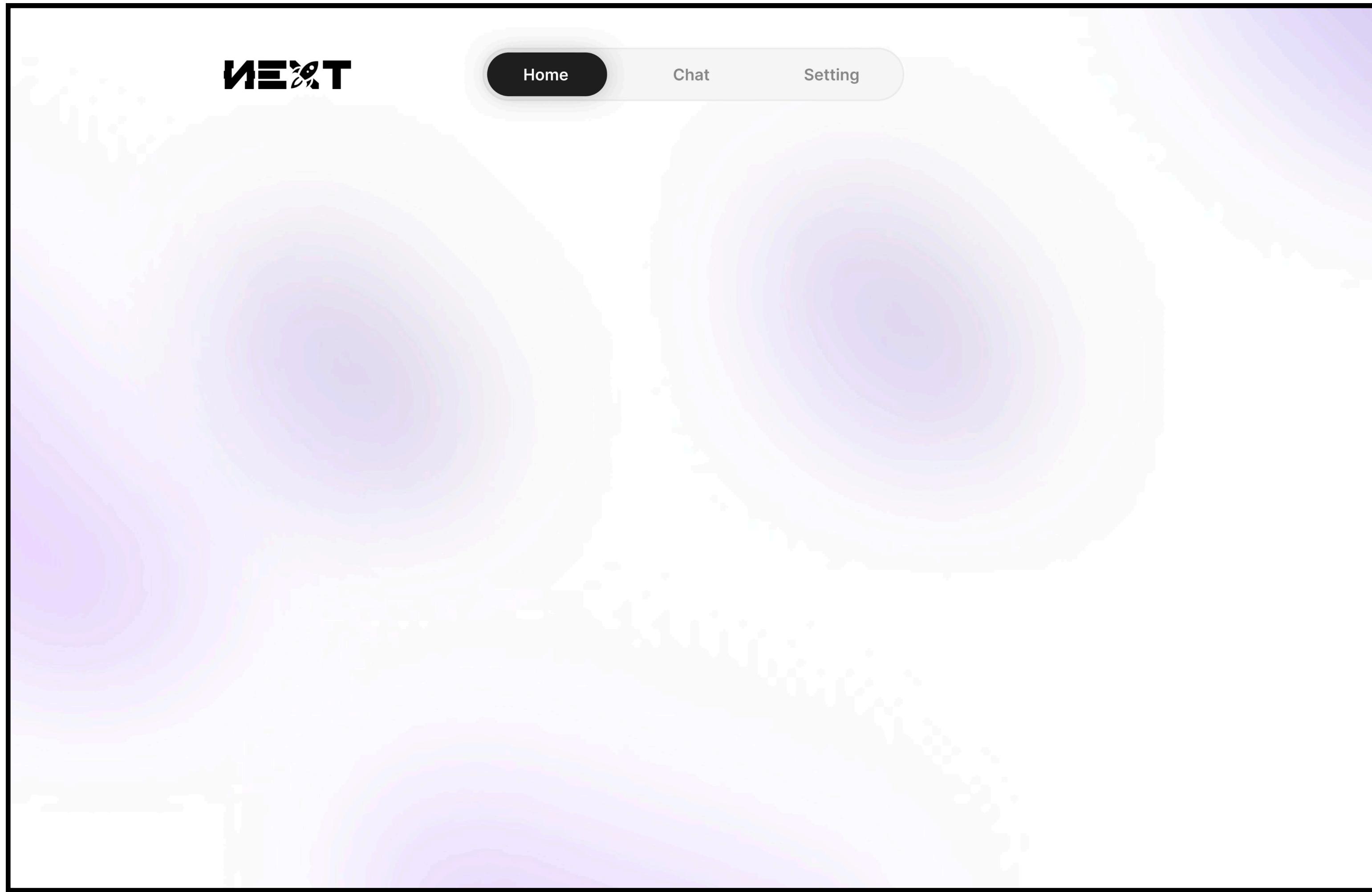
일관된 스타일을 유지함

브라우저마다 `<h1>`, `<p>`, `` 등의 기본 스타일이 다르게 설정되어 있음

src/index.css

```
src > # index.css
      You, 1 second ago | 2 authors (jhkimon and one other)
1  #root {
2   height: 100%;
3 }
4
5 /* CSS Reset */
6 *
7 *::before,
8 *::after {
9  margin: 0;
10 padding: 0;
11 box-sizing: border-box;
12 }
13
14 html,
15 body {
16 height: 100%;
17 width: 100%;
18 font-family: Arial, sans-serif;
19 line-height: 1.5;
20 background-color: #fff;
21 }
22
23 button {
24 background: none;
25 border: none;
26 padding: 0;
27 font: inherit;
28 cursor: pointer;
29 }
30
31 ul,
32 ol {
33 list-style: none;
34 }
35
36 a {
37 text-decoration: none;
38 color: inherit;
39 }
```

| 실습: layout 페이지 만들기



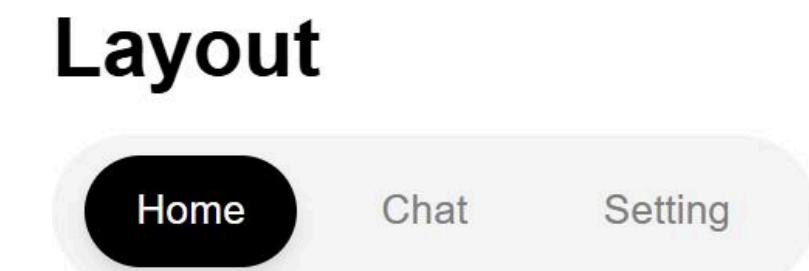
| 실습: layout 페이지 만들기

1. public / img 폴더 생성
2. img 폴더에 background, logo png 넣기 ← 노션 페이지 확인!
3. component / Header 폴더 생성
4. Header 폴더에 Header.jsx,
Header.module.css 파일 생성 후 코드 ← 노션 페이지 확인!
복사해서 넣기

| 실습: layout 페이지 만들기

Layout 페이지에서 Header 컴포넌트를 불러와보면?

```
src > pages > Layout > ✨ Layout.jsx > ...
1  import React from 'react';
2  import './Layout.module.css';
3  import Header from '../../components/Header/Header';
4
5  const Layout = () => {
6    return (
7      <div>
8        <h1>Layout</h1>
9        <Header />
10       </div>
11    );
12  };
13
14  export default Layout;
```



| 실습: layout 페이지 만들기

이제 Layout 페이지를 만들어보자!

- 로고 이미지 배치
- Header 컴포넌트 배치
- 배경이미지 추가

| 실습: layout 페이지 만들기

이제 Layout 페이지를 만들어보자!

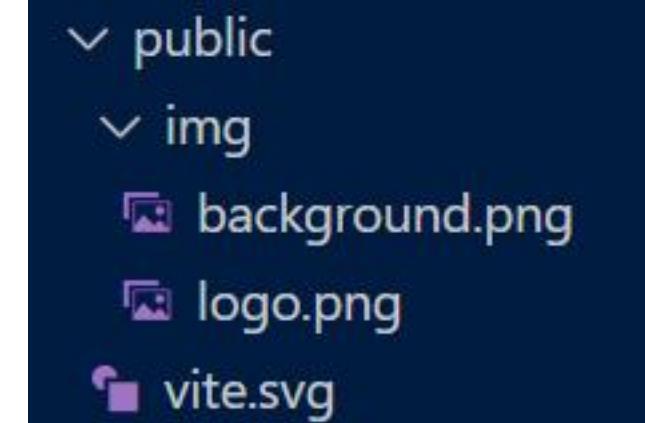
- 로고 이미지 배치
- Header 컴포넌트 배치
- 배경이미지 추가

왜 public 폴더를 쓰나요?

- React에서 public 폴더는 웹 서버의 루트로 취급
 - 루트 경로(/)로 시작하는 경로는 "절대 경로"
- ⇒ public 폴더는 경로 사용이 편리하다!

```
src > pages > Layout > Layout.jsx > ...
1  import React from 'react';
2  import styles from './Layout.module.css';
3  import Header from '../../components/Header/Header';

4
5  const Layout = () => {
6      return (
7          <div className={styles.layoutContainer}>
8              <div className={styles.Content}>
9                  
10                 <div className={styles.Header}>
11                     <Header />
12                 </div>
13             </div>
14         );
15     );
16 }
17
18 export default Layout;
```



| 실습: layout 페이지 만들기

이제 Layout 페이지를 만들어보자!

- 로고 이미지 배치
- Header 컴포넌트 배치
- 배경이미지 추가

```
src > pages > Layout > Layout.jsx ...  
1  import React from 'react';  
2  import styles from './Layout.module.css';  
3  import Header from '../../components/Header/Header';  
4  
5  const Layout = () => {  
6      return (  
7          <div className={styles.layoutContainer}>  
8              <div className={styles.Content}>  
9                    
10                 <div className={styles.Header}>  
11                     <Header />  
12                 </div>  
13             </div>  
14         </div>  
15     );  
16 };  
17  
18 export default Layout;
```

| 실습: layout 페이지 만들기

이제 Layout 페이지를 만들어보자!

- 로고 이미지 배치
- Header 컴포넌트 배치
- 배경이미지 추가

css 파일에서 배경 이미지를 지정할 수 있다!

이때, 경로에서 /public을 생략해도 이미지 로드가 가능하다.

```
src > pages > Layout > # Layout.module.css
1  .layoutContainer {
2    min-height: 100vh;
3    display: flex;
4    flex-direction: column;
5    align-items: center;
6    justify-content: center;
7    padding-top: 30px;
8    background-image: url('/img/background.png');
9    background-size: cover;
10   background-position: center;
11   background-repeat: no-repeat;
12 }
13
14 .Content {
15   width: 100%;
16   display: flex;
17   justify-content: space-between; /* 로고는 왼쪽, 헤더는 중앙 배치 */
18   align-items: center;
19   padding: 0 300px; /* 좌우 패딩 추가 */
20   position: absolute;
21   top: 40px; /* 상단 배치 */
22   left: 0;
23   right: 0;
24 }
25
26 .Header {
27   position: absolute;
28   left: 50%;
29   transform: translateX(-50%); /* 정확한 중앙 배치 */
30 }
31
32 .logo {
33   height: 30px;
34   width: auto;
35 }
```

| 실습: useNavigate

페이지 간 이동해봅시다.

useNavigate란?

React Router에서 제공하는 툐으로,
버튼 클릭, 함수 호출 등 이벤트가 발생했을 때
특정 경로로 페이지를 이동시키는 기능이다.

| 실습: useNavigate

페이지 간 이동해봅시다.

useNavigate란?

React Router에서 제공하는 툐으로,
버튼 클릭, 함수 호출 등 이벤트가 발생했을 때
특정 경로로 페이지를 이동시키는 기능이다.

Hook(훅)이란?

함수형 컴포넌트에서 다양한 기능을
사용할 수 있게 해주는 도구

| 실습: useNavigate

페이지 간 이동해봅시다.

1. useNavigate import하기
2. useNavigate 툥을 호출하여 함수 생성
3. 버튼 클릭 시 호출될 함수 작성
4. 각 버튼을 누르면 해당 페이지로 이동

| 실습: useNavigate

페이지 간 이동해봅시다.

1. useNavigate import하기
2. useNavigate 툥을 호출하여 함수 생성
3. 버튼 클릭 시 호출될 함수 작성
4. 각 버튼을 누르면 해당 페이지로 이동

```
src > components > Header > Header.jsx > ...
1 // Header.jsx
2 import React, { useState } from 'react';
3 import styles from './Header.module.css';
4 import { useNavigate } from 'react-router-dom'; // useNavigate 임포트
5
6 function Header() {
7   const [activeTab, setActiveTab] = useState('Home');
8   const navigate = useNavigate(); // useNavigate 툥 사용
9   const tabs = ['Home', 'Chat', 'Setting'];
10
11   const handleTabClick = (tab) => {
12     setActiveTab(tab);
13     // 페이지 이동
14     if (tab === 'Home') {
15       navigate('/'); // Main 페이지로 이동
16     } else if (tab === 'Chat') {
17       navigate('/chat'); // Chat 페이지로 이동
18     } else if (tab === 'Setting') {
19       navigate('/setting'); // Setting 페이지로 이동
20     }
21   };
22
23   return (
24     <div className={styles.headerContainer}>
25       {tabs.map((tab) => (
26         <button
27           key={tab}
28           className={`${styles.headerTab} ${activeTab === tab ? styles.active : ''}`}
29           onClick={() => handleTabClick(tab)} // 버튼 클릭 시 handleTabClick 실행
30         >
31           {tab}
32         </button>
33       ))}
34     </div>
35   );
36 }
```

| 실습: useNavigate

페이지 간 이동해봅시다.

useNavigate를 사용해, 로고를 눌렀을 때 Main 페이지로 이동하게 만들어보자!

| 실습: useNavigate

페이지 간 이동해봅시다.

useNavigate를 사용해, 로고를 눌렀을 때 Main 페이지로 이동하게 만들어보자!

```
src > pages > Layout > Layout.jsx > ...
1  import React from 'react';
2  import styles from './Layout.module.css';
3  import Header from '../../components/Header/Header';
4  import { useNavigate } from 'react-router-dom'; // useNavigate 임포트
5
6  const Layout = () => {
7      const navigate = useNavigate(); // useNavigate 흑 사용
8
9      const handleLogoClick = () => {
10          navigate('/'); // 로고 클릭 시 Main 페이지로 이동
11      };
12
13      return (
14          <div className={styles.layoutContainer}>
15              <div className={styles.Content}>
16                  
17                  /* 로고 클릭 시 handleLogoClick 실행 */
18                  <div className={styles.Header}>
19                      <Header />
20                  </div>
21              </div>
22          </div>
23      );
24  };
25
26  export default Layout;
```

| 과제: 클론 코딩

죽지도 않고 돌아온 CSS



| 과제: 클론 코딩

죽지도 않고 돌아온 CSS

4/2 자정까지 toss 모바일 화면을 클론 코딩하고,
github에 푸시해주세요!

