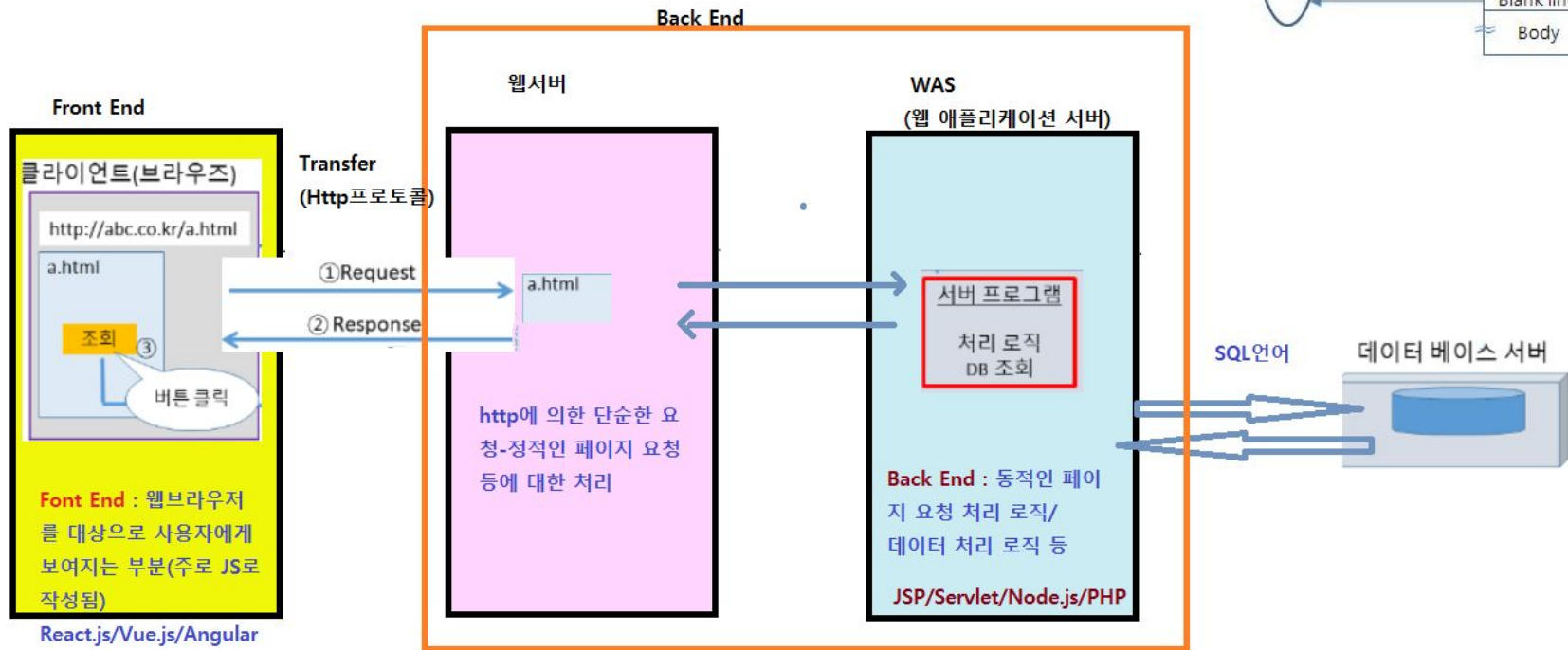


04. React.js

본 자료는 개인 학습용으로만 사용하고 웹이나 타인에게 배포를 금합니다

2025-05 백성애

웹 서비스 구조 (full stack)

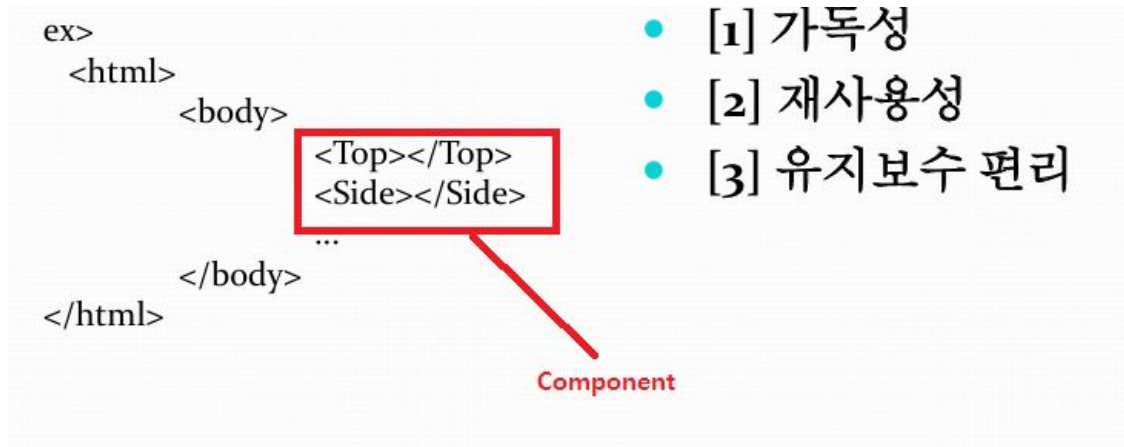


React.js란?

- 풍부한 UI를 위한 자바스크립트 라이브러리
- Facebook에서 만듦
- - 복잡한 HTML UI들을 분리하여 사용자 정의 태그로 만든 뒤,
- 이 태그들을 메인 페이지(ex)index.html)에 로드하여 붙여 사용함 (ex) Top.html/Side.html 등을 만들어 index.html에 로드해서 씬)
- 사용자 정의 태그=> 이를 **Component**라고 함
- => 가독성이 좋아짐

React.js란?

- React는 웹 프론트엔드 프레임워크의 하나로, **Single Page Application (SPA)** 개발에 주로 사용된다.
- Virtual DOM과 JSX라는 개념을 사용하여 동작한다
- 또한 Component를 활용하여 복잡한 UI를 쉽게 구성하도록 한다.
-



React 특징 -SPA

SPA란?

단일 페이지 애플리케이션 (Single Page Application)으로 서버에서 필요한 데이터만 비동기로 받아와서 동적으로 현재 화면에 다시 렌더링 하는 방식을 의미.

사용자가 애플리케이션과 상호작용할 때마다 서버에 요청하여 전체 HTML 화면을 받아오는 방식이 아니라,
=> MPA방식

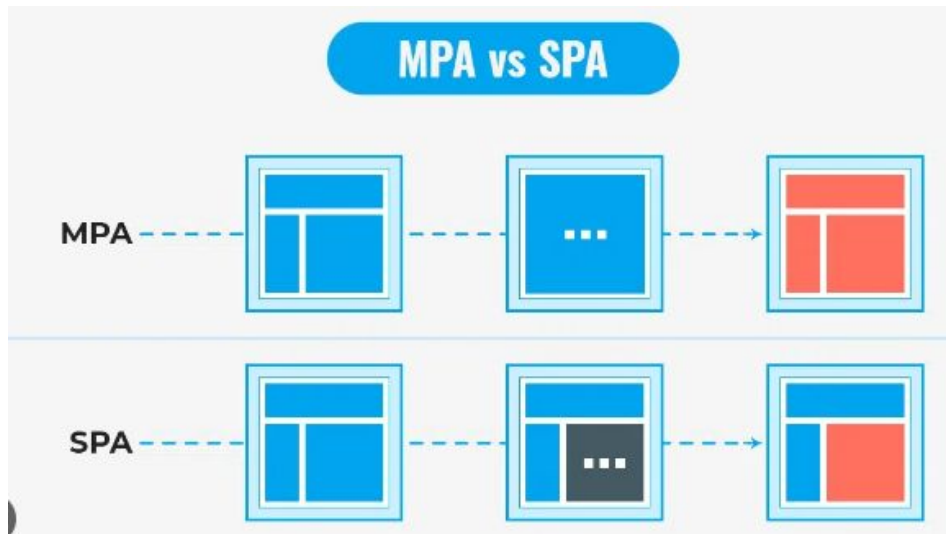
SPA 방식에서는 애플리케이션에 필요한 모든 정적 리소스를 최초 단 한 번만 다운로드하여 렌더링하고, 이후 새로운 페이지에 대한 요청이 있을 때에는 페이지 갱신에 필요한 데이터만을 내려받아 리렌더링 한다

화면 렌더링을 로컬 PC에서 즉시 생성하므로 더 빠르게 화면 전환을 처리할 수 있어서 널리 사용되고 있다

SPA가 하나의 페이지만 존재하는 애플리케이션을 의미하는 것은 아님.

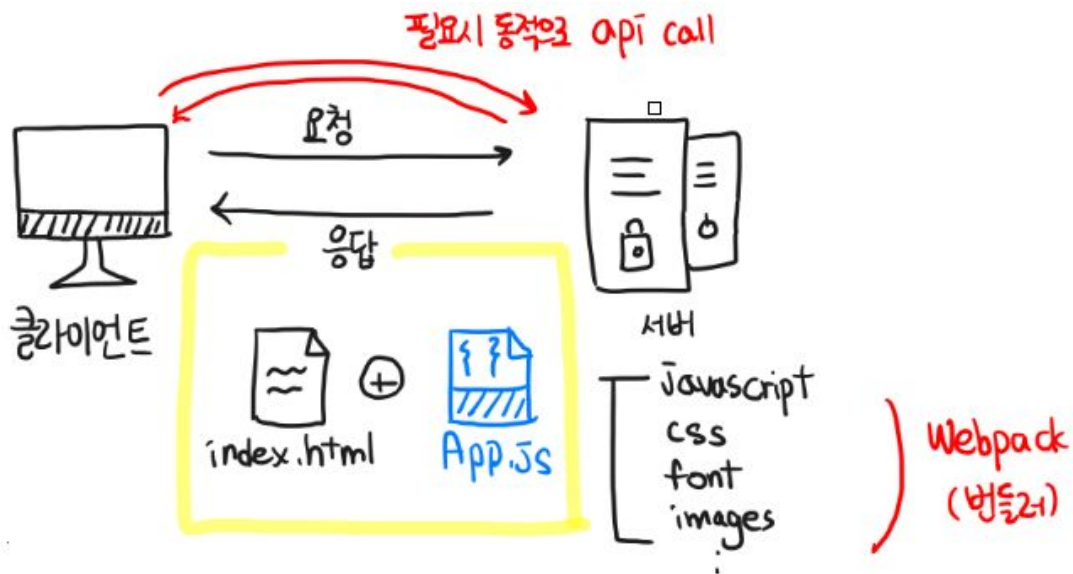
SPA(Single Page Application)도 여러 페이지가 존재하는데, 다수의 페이지를 표시하는 데 있어서 과거 전통적인 방식으로 페이지 전환을 수행하지 않고, 마치 하나의 페이지인 것처럼 처리하는 기술을 의미

MPA / SPA



- **MPA**-(Multi Page Application)는 각 페이지별로 HTML문서가 따로 존재하여, 페이지를 이동할 때마다 새로운 html 페이지를 받아와서 새로 렌더링하는 전통적인 웹페이지 구성 방식
-
- **SPA**-(Single Page Application)는 웹 애플리케이션에 필요한 모든 정적 리소스를 최초 접근 시 단 한번만 다운로드하고,
- 이후 새로운 페이지 요청 시, 페이지 갱신에 필요한 데이터만 따로 전달받아 페이지를 갱신을 합니다.
- 기존 페이지의 내부를 수정해서 보여주는 방식

React (SPA) 동작



React 특징 -SPA

SPA의 장점

- 1) **사용자 경험 향상**: 정적 리소스를 모두 다운받기 때문에 페이지 갱신시 화면 깜빡임이 없다
- 2) **높은 반응성**: 갱신에 필요한 데이터만 받기 때문에 속도와 응답시간이 빠름
- 3) **재사용성**: 컴포넌트 기반의 코드를 작성하기 때문에 재사용성이 높다.

React 특징 -SPA

SPA의 단점

- 1) 최초 단 한 번에 모든 정적 리소스를 내려받기 때문에 초기 구동 속도가 느리다
- 2) 정적 리소스를 최초 한번 로드 하고, 이후 페이지가 JS에 의해 동적으로 갱신되기 때문에 검색엔진 최적화(SEO)가 어렵다. (검색엔진 크롤러가 초기 데이터만 수집하거나 JS를 완전하게 실행하지 못하는 경우가 생김)

이를 해결하기 위한 여러 기법이 사용되고 있다.

SPA 방식은 대부분 클라이언트 측 렌더링(Client-Side Rendering, CSR) 방식 즉, 클라이언트 환경에서 자바스크립트를 사용하여 동적으로 View를 리렌더링 한다.

이렇게 동적으로 View를 리렌더링 할 때 어떻게 하면 업데이트가 필요한 DOM 요소를 좀 더 쉽게 찾을 수 있고, 업데이트 작업을 보다 효율적이고 간편하게 진행할 수 있을까를 고민하는 과정에서 Angular, Ember.js, Backbone.js 등 다양한 자바스크립트 프레임워크들이 개발되었다

React 특징

React는 오로지 어떻게 하면 사용자에게 **View**를 효율적으로 보여줄 수 있을까 라는 고민으로부터 시작되었다.

페이스북의 소프트웨어 엔지니어였던 Jordan Walke는 **Model**의 데이터가 변할 때마다 어떤 방식으로 **View**를 갱신할지 고민하는 것보다 그냥 새로운 **View**를 만들어 대체하면 어떨까 라고 생각했고, 이러한 아이디어를 바탕으로 개발된 것이 바로 **React**다

React는 **Virtual DOM**이라는 새로운 개념을 활용하여 **View**에서 변경되어야 할 부분만을 효율적으로 업데이트할 수 있도록 고안되었다.

또한, 자바스크립트 문법을 확장한 **JSX**라는 문법을 사용함으로써 불필요한 코드를 줄여 개발 생산성을 높였습니다. 그리고 **UI**를 컴포넌트로 나누어 관리함으로써 재사용성과 유지보수가 쉽도록 하였다.

1. 동적 콘텐츠 로딩:

SPA는 초기 로딩 시 최소한의 HTML을 서버에서 가져오고, 나머지 콘텐츠는 JavaScript를 통해 클라이언트 측에서 동적으로 로드합니다. 이는 검색엔진 봇이 페이지를 크롤링할 때 모든 콘텐츠를 바로 볼 수 없게 만듭니다. 검색엔진은 주로 서버에서 렌더링된 정적 HTML을 좋아하기 때문에, JavaScript가 로드되기 전의 빈 페이지나 최소한의 정보만을 볼 수 있습니다.

2. 클라이언트 사이드 렌더링 (CSR):

SPA는 주로 클라이언트 사이드 렌더링을 사용합니다. 이는 서버에서 페이지를 가져온 후 클라이언트(브라우저)에서 JavaScript를 사용해 나머지 페이지를 구성하는 방식입니다. 많은 검색엔진 봇은 JavaScript 실행을 제대로 처리하지 못하거나 제한적으로 처리하기 때문에, 전체 페이지를 인덱싱하는 데 어려움을 겪습니다.

3. URL 구조:

전통적인 다중 페이지 애플리케이션(MPA)에서는 각 페이지가 고유한 URL을 가집니다. 하지만 SPA에서는 페이지 전환이 JavaScript를 통해 이루어지며, URL 변경이 해시 기반(#)이거나 브라우저 히스토리 API를 사용해 수행됩니다. 이러한 URL 구조는 검색엔진이 각각의 페이지를 별도로 인식하고 인덱싱하는 데 문제를 일으킬 수 있습니다.

4. 초기 로딩 시간:

SPA는 초기 로딩 시 필요한 모든 JavaScript 파일을 한 번에 가져오는 경향이 있습니다. 이로 인해 초기 로딩 시간이 길어질 수 있으며, 이는 검색엔진이 페이지 로드를 포기하게 만들 수 있습니다. 이는 SEO 점수에 부정적인 영향을 미칩니다.

이러한 문제를 해결하기 위해 몇 가지 접근 방식이 사용될 수 있습니다:

• 서버 사이드 렌더링 (SSR):

서버에서 완전히 렌더링된 HTML을 클라이언트로 보내는 방식입니다. React에서는 Next.js와 같은 프레임워크를 사용하여 쉽게 SSR을 구현할 수 있습니다.

• 정적 사이트 생성 (SSG):

사전에 페이지를 정적으로 생성하여 배포하는 방식입니다. 이는 정적인 HTML 파일을 제공하기 때문에 SEO에 유리합니다. Next.js는 SSG도 지원합니다.

• 프로그레시브 웹 애플리케이션 (PWA):

PWA는 초기 로딩 시간을 줄이고 사용자 경험을 개선하는 데 도움이 됩니다. 이는 SEO에도 긍정적인 영향을 미칠 수 있습니다.

• 프리렌더링:

JavaScript를 사용해 페이지를 사전 렌더링하여 정적 HTML 파일로 생성한 후, 이를 검색엔진에 제공하는 방법입니다.

이러한 접근 방식을 통해 SPA의 SEO 문제를 어느 정도 해결할 수 있습니다.

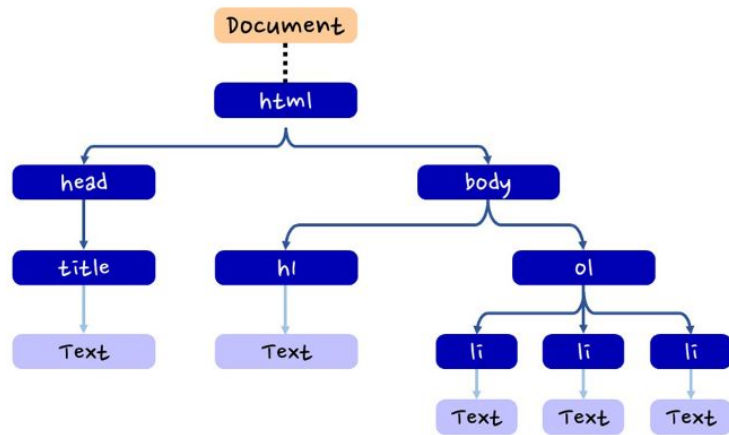
React의 장점

- - 자바스크립트와 xml을 이용해 사용자 인터페이스를 구축하는 **엔진**
- - **편리한 반응형 렌더링**. (리액트가 데이터 변경을 감지하고 인터페이스를 다시 렌더링 한다)
- - **가상DOM을 이용해 DOM을 메모리상에서 조작하므로 효율적이고 빠름** (앱의 상태가 달라지면 리액트는 UI현재상태와 원하는 상태를 비교하고 실제 DOM의 최소집합으로 계산을 수행함)

DOM 과 Virtual DOM

DOM이란?

- Document Object Model
- HTML이나 XML 문서의 구조화된 표현
- 문서 내의 모든 요소를 정의할 뿐만 아니라 각각의 요소에 접근하는 방법(인터페이스)도 같이 제공한다.
- DOM은 노드(node)와 객체(object)로 문서를 표현하며, 자바스크립트와 같은 스크립팅 언어를 사용하여 수정할 수 있다.



Virtual DOM

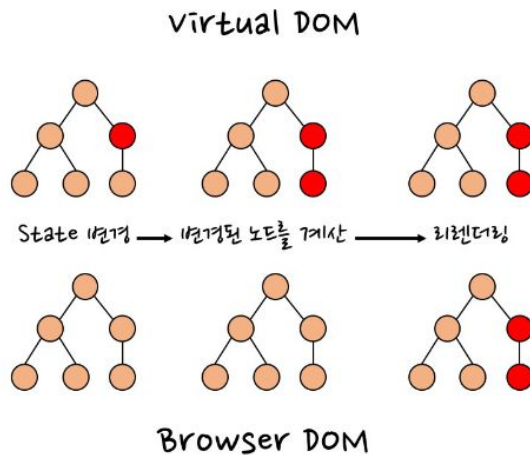
기존 방식에서는 View에 표시할 데이터가 변경되었을 때 갱신이 필요한 DOM 요소들을 직접 찾아 조작함으로써 업데이트를 진행하였다.

하지만 **React**에서는 **Virtual DOM**이라는 새로운 개념을 도입하여 View에서 변경되어야 할 부분을 효율적으로 찾아 업데이트할 수 있다.

Virtual DOM은 실제 존재하는 DOM이 아닌 **메모리 상에서만 존재하는 가상의 DOM**.

React는 상태 (state)가 업데이트되면 , 우선 변경 사항이 모두 적용된 전체 UI를 새로운 **Virtual DOM**에 렌더링한다 .

그리고 이전 버전의 **Virtual DOM**과 새로 생성된 **Virtual DOM**을 서로 비교하여 차이점이 있는 부분만 실제 **Browser DOM**에 적용하게 된다.



개발 환경 및 공식 문서

[React](#)

<https://react.dev>

[React Reference Overview – React](#)

<https://react.dev/reference/react>

node.js 다운로드

[Node.js — Run JavaScript Everywhere \(nodejs.org\)](#)

[1] node.js 설치하기

<http://nodejs.org> 사이트에서 다운로드
=> LTS (안정화 버전), Current(최신버전)

[2] VSCode 설치

<https://code.visualstudio.com/download>

[3] vscode에 확장 플러그인

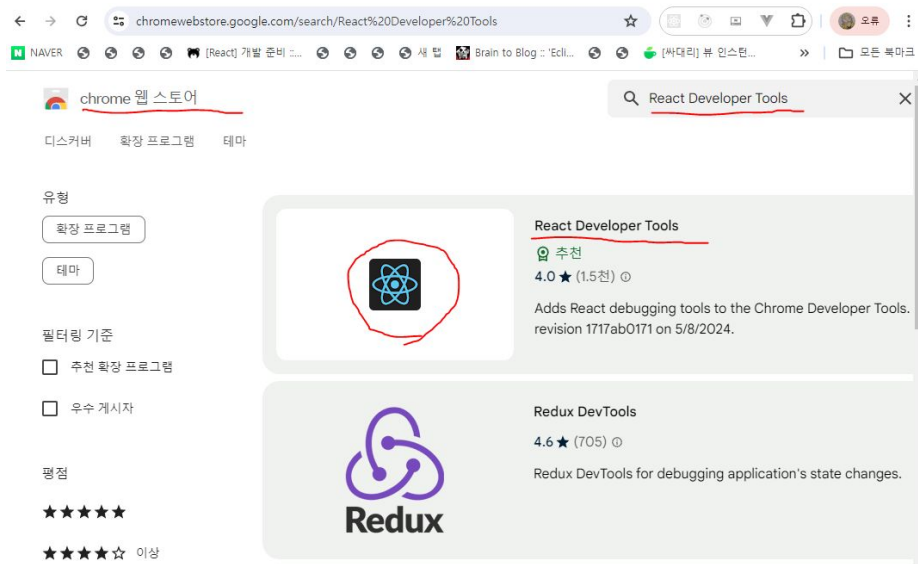
- JS JSX Snippets
- JavaScript(ES6) code snippets
- ES7 React/Redux/GraphQL/React-Native snippets
- Live Server/Prettier 등..

[5] React Developer Tool 설치

- <https://chrome.google.com/webstore>

리액트 개발자 도구 설치

- 리액트 디버깅과 개발을 돕기 위한 브라우저 확장 프로그램
- 크롬 브라우저에서 설치해서 사용한다.
- <https://chrome.google.com/webstore>



리액트 개발자 도구 설치

- 검색 결과로 나온 React Developer Tools 를 클릭하여 Chrome에 추가 버튼을 눌러준다.

chrome 웹 스토어

Q 확장 프로그램 및 테마 검색

버 확장 프로그램 테마



React Developer Tools

추천 4.0 ★ (평점 1.5천개)

확장 프로그램

개발자 도구

4,000,000 사용자

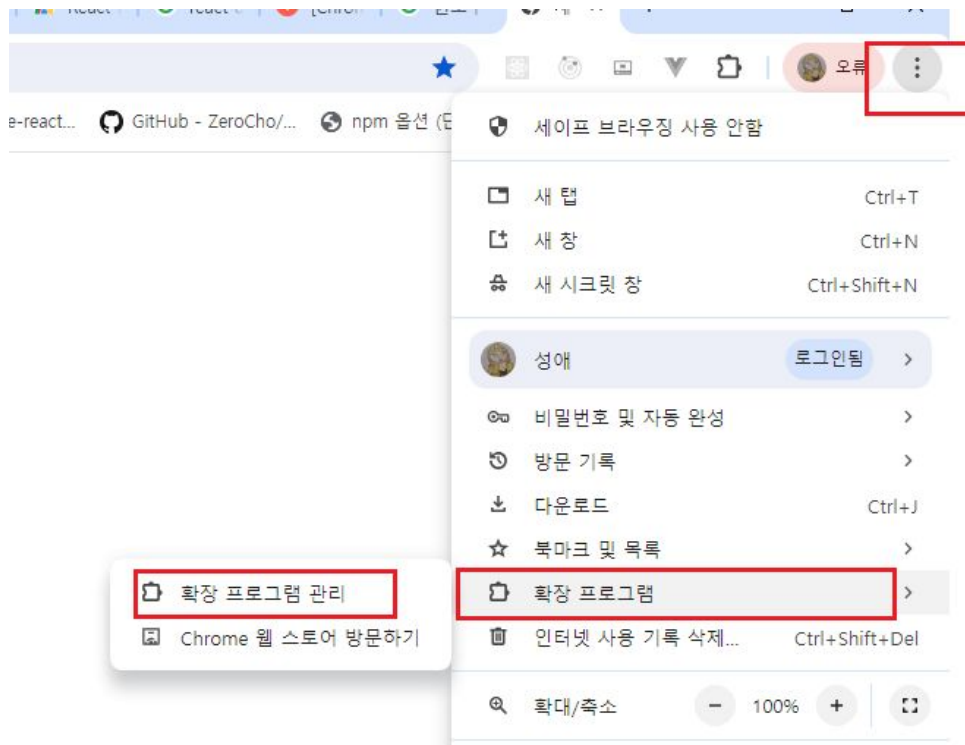
Chrome에서

추가

- 권한 관련 경고 대화상자가 나타나면 모두 허용 버튼을 클릭해서 설치한다
- 설치가 완료되면 확장 프로그램 설정을 확인해야 한다
-

리액트 개발자 도구 설정

- 브라우저 우측 상단의 점
(.)3개를 클릭> 확장
프로그램> 확장프로그램
관리로 이동

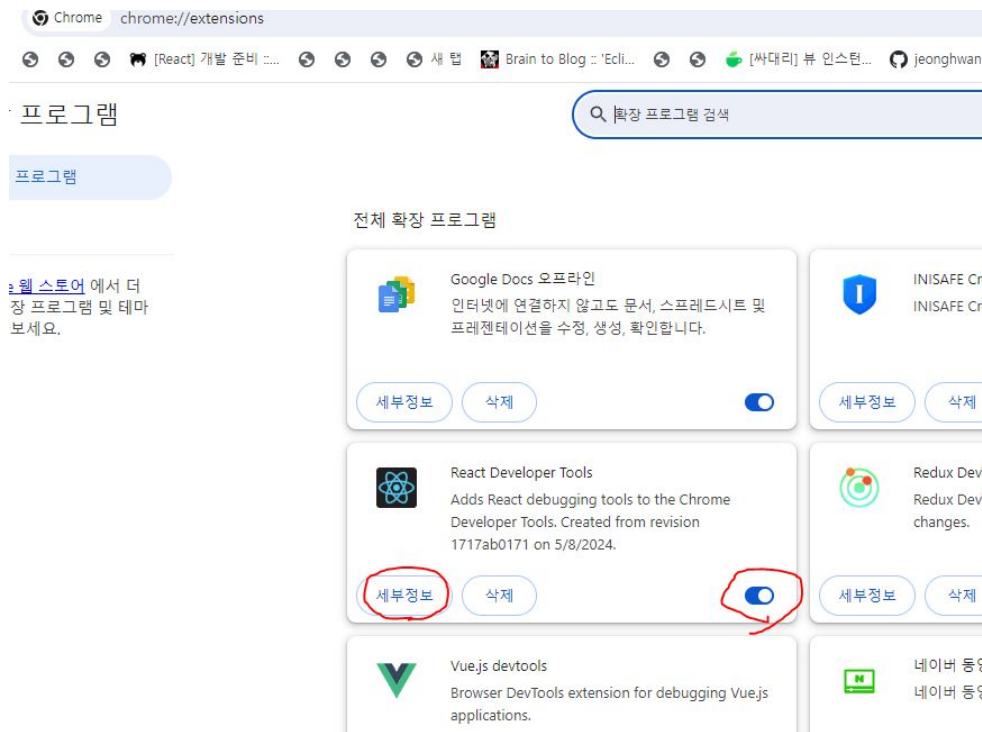


리액트 개발자 도구 설정

React Developer Tools를 찾아

스위치를 On으로 하고,

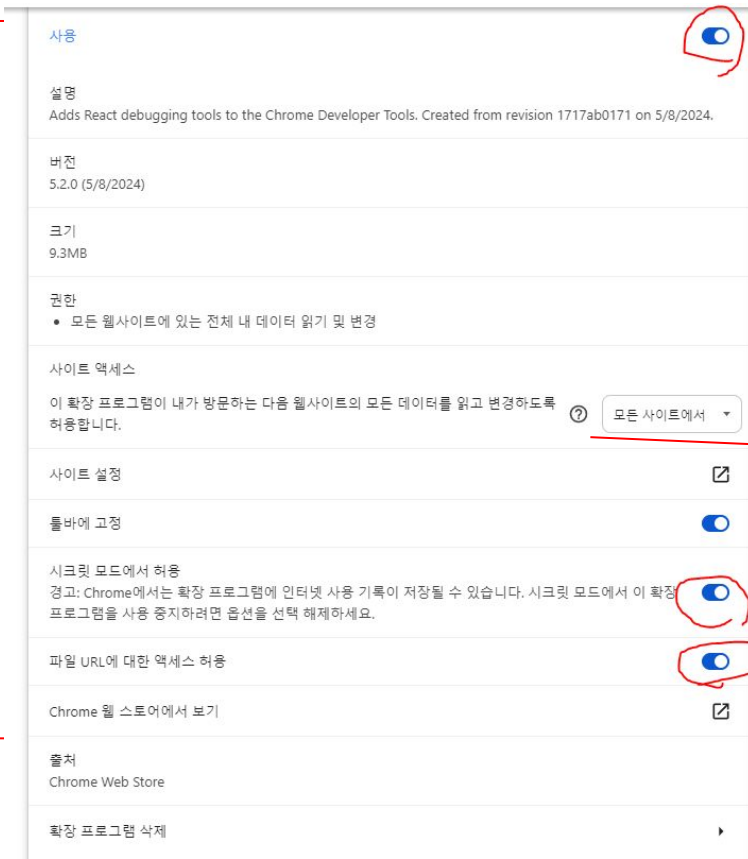
세부정보 버튼을 클릭한다



리액트 개발자 도구 설정

오른쪽 그림과 같이 스위치를 on으로 해주자

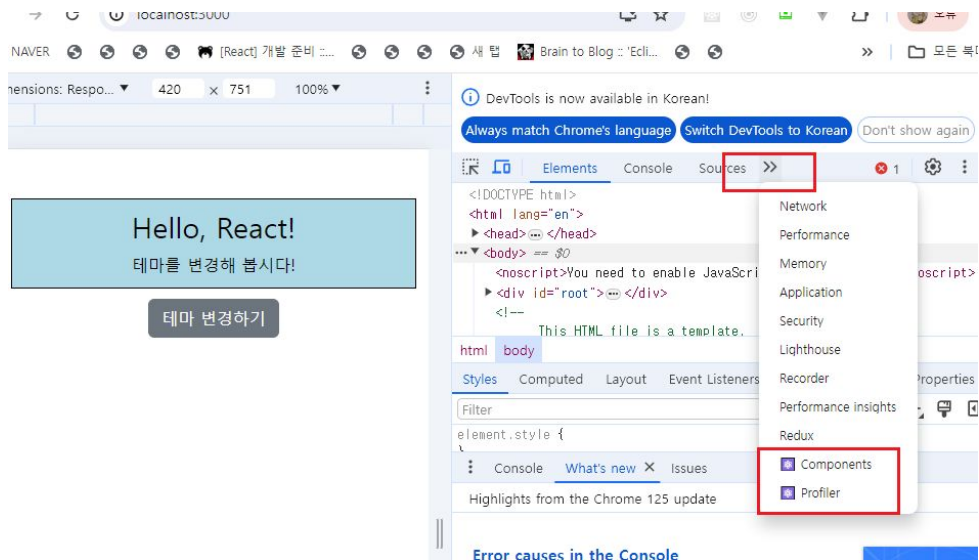
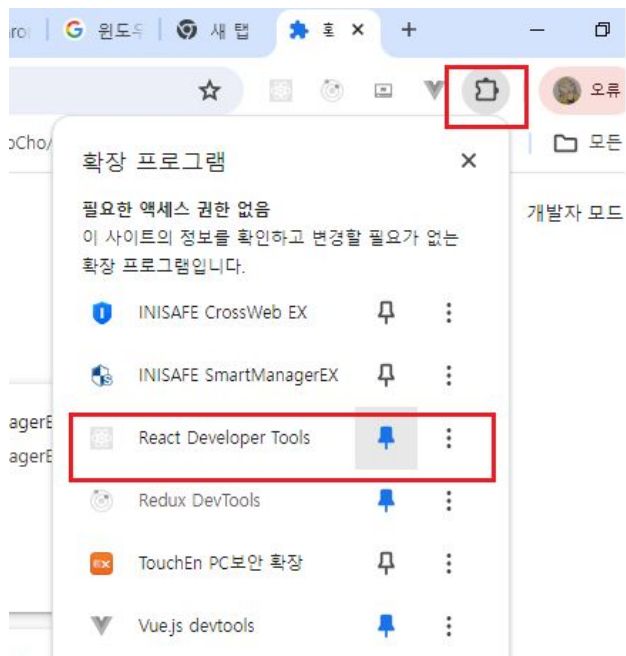
1. 사용: ON
2. 사이트 액세스: 모든 사이트에서
3. 시크릿 모드에서 허용: ON
4. 파일 URL에 대한 액세스 허용: ON



리액트 개발자 도구 설정

아래 그림과 같이 핀을 고정하자

F12 눌러 개발자도구 열고 >> 를
클릭 > Component, Profiler 확인



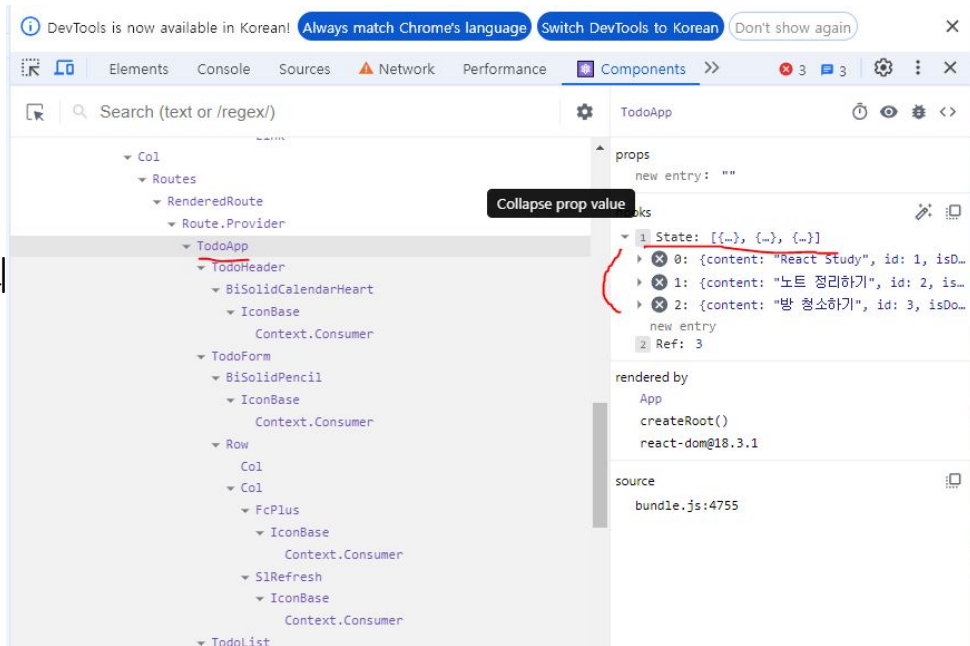
리액트 개발자 도구 설정

Components와 Profiler 탭은 리액트 개발자 도구가 제공하는 기능이다.

컴포넌트 계층구조 확인이나 성능 측정 등 개발에 필요한 유용한 기능을 갖는다

- Components 탭에서는 현재 리액트 앱의 컴포넌트 트리 **와 props, state 정보 확인 가능**
- Profiler 탭은 리액트 컴포넌트 렌더링 성능을 측정한다

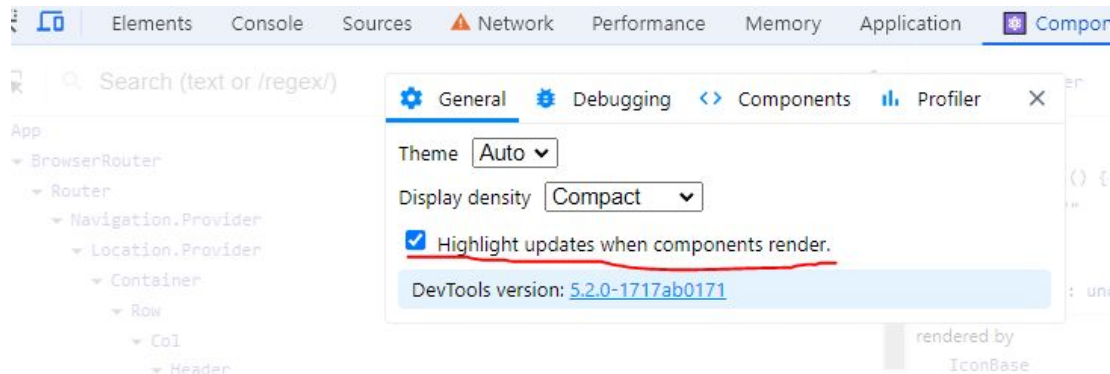
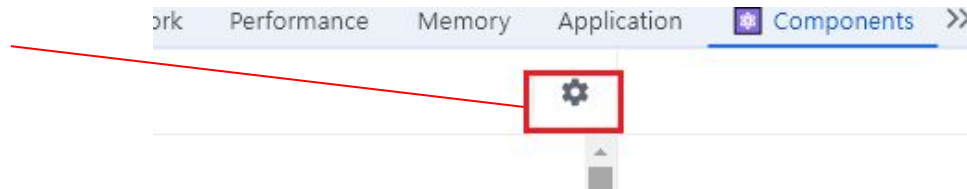
우리는 주로 Components 탭 위주로 사용할 예정



리액트 개발자 도구 설정

리렌더 하이라이트 기능 사용하기 (옵션)

- 어떤 컴포넌트가 리렌더 되는지 확인할 수 있는 기능이다.
- 톱니바퀴 View Settings
- 아이콘 클릭
- **Highlight update when**
- **...체크박스 체크**



Mac에서 설치하는 아래 사이트 참고 하세요

Mac 에서 React 개발환경

[참조 사이트]

<https://heeeming.tistory.com/entry/React-%EB%A7%A5Mac-os-VSCODE-%EB%A6%AC%EC%95%A1%ED%8A%B8-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-%EC%83%9D%EC%84%B1%ED%95%98%EB%8A%94-%EB%B0%A9%EB%B2%95%EA%B0%9C%EB%B0%9C%ED%99%98%EA%B2%BD%EC%84%B8%ED%8C%85>

<https://velog.io/@yoonezi/mac-React-%EC%84%A4%EC%B9%98-%EB%B0%8F-%EC%8B%A4%ED%96%89%EB%B2%95-%EC%83%81%EC%84%B8%EC%84%A4%EB%AA%85>

react앱 생성

Windows/Mac 의 경우 터미널 또는 도스 콘솔에서

```
npm create vite@latest 프로젝트명 --template react
```

```
swanb@DESKTOP-T5B94PA MINGW64 /d/BSA/Node/Ezen_node
$ npm create vite@latest test-app --template react
Need to install the following packages:
create-vite@6.5.0
Ok to proceed? (y)
|
◆ Select a framework:
  o Vanilla
  o Vue
  ● React
  o Preact
  o Lit
  o Svelte
  o Solid
  o Qwik
  o Angular
  o Marko
  o Others
```

npm create vite@latest test-app --template react

만약 타입스크립트를 적용한 리엑트를 하고 싶다면

npm create vite@latest test-app --template react-ts

react앱 생성

Need to install the following packages:

create-vite@6.5.0

Ok to proceed? (y)

◇ Select a framework:

React

◇ Select a variant:

TypeScript

타입스크립트를 사용

안하려면

JavaScript + SWC를 선택해도

된다

◇ Scaffolding project in D:\BSA\Node\Ezen_node\test-a

Done. Now run:

cd test-app

npm install

npm run dev

왼쪽과 같이 뜨면 가이드 대로 터미널에서
명령어를 입력한다

cd test-app
npm install

swanb@DESKTOP-T5B94PA MINGW64 /d/BSA/Node/Ezen_node

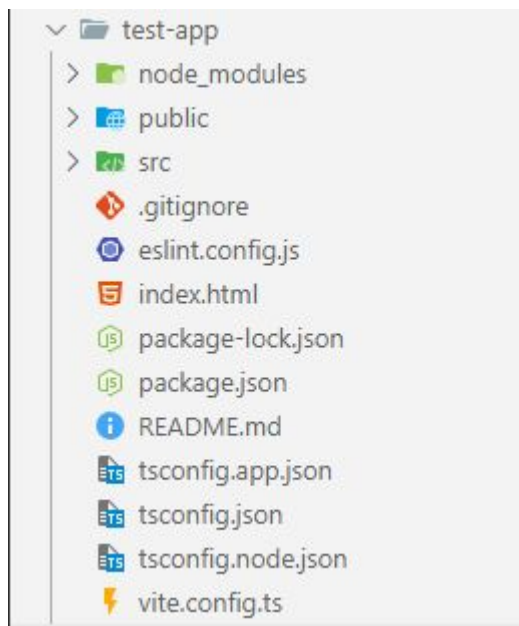
• \$ cd test-app

swanb@DESKTOP-T5B94PA MINGW64 /d/BSA/Node/Ezen_node/test-app

○ \$ npm install

[.....] / idealTree:test-app: sill idealTree buildDeps

test-app 리액트앱 생성 결과



react앱 실행

터미널 test-app 디렉토리에서 **npm run dev** 명령어를 이용해 리액트 앱을 실행시킨다

```
swanb@DESKTOP-T5B94PA MINGW64 /d/BSA/Node/Ezen_node/test-app
```

```
o $ npm run dev
```

```
> test-app@0.0.0 dev
```

```
> vite
```

```
VITE v6.3.5 ready in 354 ms
```

```
→ Local: http://localhost:5173/
```

```
→ Network: use --host to expose
```

```
→ press h + enter to show help
```

```
□
```

react앱 실행

브라우저에 <http://localhost:5173> url을 입력하면
오른쪽 화면이 뜨는 것을 확인할 수 있다.



Vite + React

count is 0

Edit src/App.tsx and save to test HMR

Click on the Vite and React logos to learn more

프로젝트 구조

main.tsx

```
test-app > src > main.tsx
1 import { StrictMode } from 'react'
2 import { createRoot } from 'react-dom/client'
3 import './index.css'
4 import App from './App.tsx'
5
6 createRoot(document.getElementById('root')!).render(
7   <StrictMode>
8     <App />
9   </StrictMode>,
10 )
11 |
```

index.html

```
test-app > index.html > ...
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8" />
5     <link rel="icon" type="image/svg+xml" href="/vite.svg" />
6     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7     <title>Vite + React + TS</title>
8   </head>
9   <body>
10     <div id="root"></div>
11     <script type="module" src="/src/main.tsx"></script>
12   </body>
13 </html>
14 |
```

test-app

src/

- **index.html** => id가 root인 엘리먼트가 있음
-> 컴포넌트들이 이안에 들어가도록 약속해 두었음
- **main.tsx** => 진입 파일. 이 안에서 App 컴포넌트를 index.html의 <div id="root"></div> 인 곳에 보여주도록 함

프로젝트 구조

리액트를 통해 컴포넌트들을 만들면 id가 root인 곳에 들어가도록 약속 되어 있다

root 안에 들어갈 컴포넌트들은 src 안에 둔다.

main.tsx 파일(또는 [main.js](#)) 이

진입 파일(entry file)이다

이 파일에 기술된 App이 컴포넌트 임

[App.js](#) 에 함수형으로 구성되어 있다

```
test-app > src > App.tsx ...
5
6 function App() {
7   const [count, setCount] = useState(0)
8
9   return (
10    <>
11      <div>
12        <a href="https://vite.dev" target="_blank">
13          <img src={viteLogo} className="logo" alt="Vite logo" />
14        </a>
15        <a href="https://react.dev" target="_blank">
16          <img src={reactLogo} className="logo react" alt="React logo" />
17        </a>
18      </div>
19      <h1>Vite + React</h1>
20      <div className="card">
21        <button onClick={() => setCount((count) => count + 1)}>
22          count is {count}
23        </button>
24        <p>
25          Edit <code>src/App.tsx</code> and save to test HMR
26        </p>
27      </div>
28      <p className="read-the-docs">
29        Click on the Vite and React logos to learn more
30      </p>
31    </>
32  )
33 }
34
35 export default App
36
```

컴포넌트 작성 규칙

클래스로 구현된 컴포넌트에는 컴포넌트의 렌더링된 출력결과를 표시할 `render()` 메서드가 필요하다.

반면 함수형 컴포넌트에서는 `render()` 메서드는 필요없다.

반환되는 함수에서는 `root element`는 딱 하나여야 함에 주의하자.

우리는 함수를 이용해 컴포넌트를 만들 예정.

리액트 컴포넌트 이름은 대문자로 시작해야 하며 각 단어를 대문자로 시작하는 파스칼 표기법 명명 규칙을 이용하는 것이 좋다.

컴포넌트 작성 규칙



React 컴포넌트 파일의 확장자는 보통 `.js` 또는 `.jsx`를 사용합니다.

- `.js`: 일반적인 JavaScript 파일로, React 컴포넌트를 포함할 수 있습니다. JSX 구문을 사용해도 Babel 등의 도구를 사용해 변환할 수 있습니다.
- `.jsx`: JSX 구문이 포함된 JavaScript 파일임을 명시적으로 나타냅니다. 이는 파일을 보는 사람에게 이 파일이 React 컴포넌트를 포함하고 있음을 더 명확히 전달할 수 있습니다.

최근에는 TypeScript를 사용하는 경우도 많기 때문에, 다음과 같은 확장자도 사용됩니다:

- `.ts`: TypeScript 파일로, 타입 검사를 통해 더 안전한 코드를 작성할 수 있습니다.
- `.tsx`: JSX 구문이 포함된 TypeScript 파일입니다. React 컴포넌트를 TypeScript로 작성할 때 사용합니다.

따라서 프로젝트의 설정과 팀의 스타일 가이드에 따라 적절한 확장자를 선택하면 됩니다. 일반적인 JavaScript 프로젝트에서는 `.js` 또는 `.jsx`를, TypeScript를 사용하는 프로젝트에서는 `.ts` 또는 `.tsx`를 사용하는 것이 좋습니다.

css / bootstrap 등 사용

- css 활용
- index.css 파일에 정의하고 index.js 에서 import './index.css'
- bootstrap 설치
- **npm intall --save bootstrap**
- 설치후
- 엔트리 파일인 index.js 에서
- **import 'bootstrap/dist/css/bootstrap.css'**

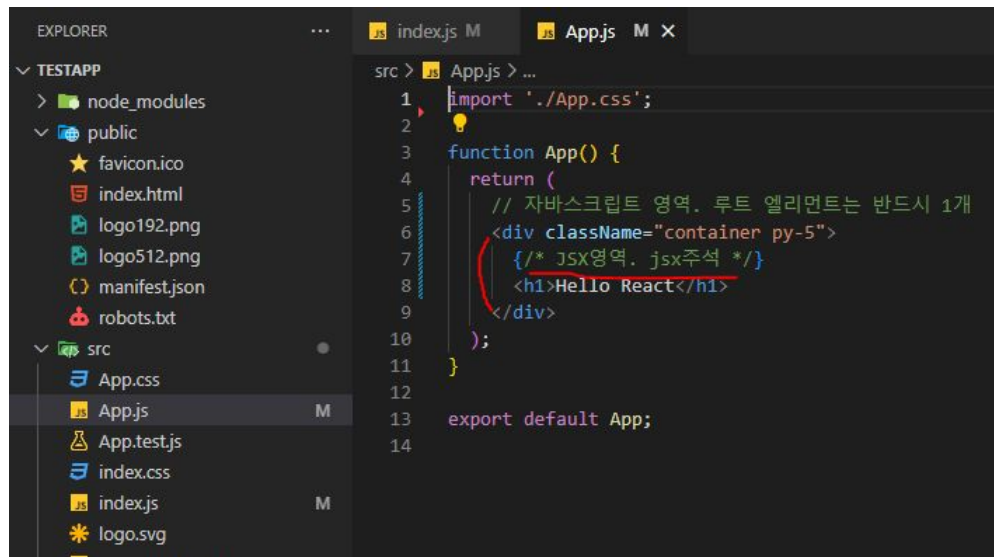
참고: [Adding Bootstrap | Create React App \(create-react-app.dev\)](https://create-react-app.dev/docs/adding-bootstrap)

```
src > index.js > ...
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6 import 'bootstrap/dist/css/bootstrap.css'
7
8 const root = ReactDOM.createRoot(document.getElementById('root'));
9 root.render(
10   // React.StrictMode 는 개발환경에서 잠재적 문제를 식별하도록 도와주는 도구이다.
11   // 하위 컴포넌트들을 엄격 모드로 감지. 오류를 검출하게 도움. 개발환경에서 사용되고 프로덕션 빌드에서는 무시된다.
12   <React.StrictMode>
13     <App />
14   </React.StrictMode>
15 );
16
17 // If you want to start measuring performance in your app, pass a function
18 // to log results (for example: reportWebVitals(console.log))
19 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
20 reportWebVitals();
21 /**함수는 React 애플리케이션의 성능을 모니터링하고 측정하기 위한 도구입니다.
22  * 이 함수는 Core Web Vitals 지표와 같은 중요한 성능 메트릭을 측정하고,
23  * 이러한 메트릭에 대한 데이터를 콘솔이나 분석 도구를 통해 보고할 수 있도록 해줍니다.
24  * Core Web Vitals는 사용자 경험에 중요한 영향을 미치는 웹 성능 지표를 의미합니다.
25  * 이러한 지표에는 로딩 속도, 반응성, 레이아웃 안정성 등이 포함될 수 있습니다.
26  * reportWebVitals() 함수를 사용하면 애플리케이션의 성능을 지속적으로 모니터링하고 개선하는 데 도움이 됩니다. */
```

index.js

JSX

- **JSX**는 자바스크립트 코드 안에 선언적인 **xml**스타일의 구문을 작성할 수 있게 해주는 자바스크립트 구문 확장이다.
- **xml**같은 문법을 자바스크립트로 변환해줌 (babel이 **jsx** 로더를 사용하여 **jsx**형태 코드를 자바스크립트 코드로 변환해줌)



```
src > App.js > ...
1 import './App.css';
2
3 function App() {
4   return (
5     // 자바스크립트 영역. 루트 엘리먼트는 반드시 1개
6     <div className="container py-5">
7       /* JSX영역. jsx주석 */
8       <h1>Hello React</h1>
9     </div>
10   );
11 }
12
13 export default App;
14
```



Hello React

JSX

- JSX를 사용하면 자바스크립트 코드 내에서도 HTML 코드처럼 UI를 바로 표현할 수 있기 때문에 UI 작업시 좀 더 직관적이고 용이한 개발이 가능
- 하지만 JSX는 표준 자바스크립트 문법이 아니기 때문에, 웹 브라우저는 JSX 코드를 그대로 이해하지는 못한다.
- 따라서 JSX 코드는 웹 브라우저에서 실행되기 전 Babel과 같은 도구를 사용하여 표준 자바스크립트 문법으로 변환되어야 한다.

```
"use strict";  
//JSX로 작성된 코드 =====> 오른쪽과 같이 js로  
변환된다  
function App() {  
  return (  
    <div>  
      <h1>Hello World!</h1>  
    </div>  
  );  
}
```

```
"use strict";  
//JSX로 작성된 React 컴포넌트를 babel에 의해 //JavaScript로  
//변환한 형태  
  
function App() {  
  return React.createElement("div", null,  
    React.createElement("h1", null, "Hello World!"));  
}  
  
//createElement(type, props, children) 함수를 이용해  
//React element가 생성되어 가상dom에 렌더링 된다
```

Babel이란?

Babel은 일종의 **자바스크립트 컴파일러**로, 최신 자바스크립트 문법(ES6)이나 실험적인 자바스크립트 문법으로 작성된 코드를 이전 버전인 ES5 문법의 자바스크립트 코드로 변환해 주는 **툴 체인**이다.

자바스크립트는 정말 다양한 종류와 버전의 웹 브라우저에서 실행된다. 대부분의 최신 웹 브라우저에서는 최신 자바스크립트 문법을 바로 실행시킬 수 있지만, 구 버전의 웹 브라우저에서는 특정 코드가 실행되지 않을 수도 있고 특정 브라우저에서만 실행되지 않는 코드들도 있을 수 있다

따라서 **Babel**과 같은 **트랜스파일러(transpiler)**을 사용하여 작성한 자바스크립트 코드가 어떤 환경에서도 정상적으로 동작할 수 있도록 이전 버전의 자바스크립트 문법(ES5)으로 변환해 주는 것.

React에서 사용하는 **JSX** 문법도 표준 자바스크립트 문법이 아니므로, **Babel**을 통해 **ES5** 문법의 자바스크립트 코드로 변환되어야만 웹 브라우저가 정상적으로 인식할 수 있습니다.

© Babel 공식 홈페이지 (<https://babeljs.io/docs/en/>)

JSX의 특징

- 모든 **jsx** 형태 코드는 **container element(root element)** 안에 포함시켜줘야 한다.
- Jsx안에서 javascript를 표현하는 형식은 **{ }** 임

```
render(){  
  let txt='Hello React'  
  return(<div>{txt}</div>)  
}
```

- Jsx안에서 style설정 할 때는 camelCase형태로 사용함

```
render() {  
  let style = {  
    color: 'aqua',  
    backgroundColor: 'black'  
  };  
  
  return (  
    <div style={style}>React CodeLab</div>  
  );  
}
```

JSX 문법

- JSX는 html과 모양새는 같아보이지만 xml규칙을 따르고 있어 jsx구문을 작성할 때도 XML표준을 엄격히 지켜야 한다

시작 태그가 있으면 반드시 종료 태그가 있어야 함

- 가령 식으로 사용해야 한다.
- 아니면
- <input type="text"/>
-
<hr/>

```
export default function App() {  
  return (  
    <div>  
      <h1>Void Elements</h1>  
      <hr />  
      <input type="text" name="name" />  
    </div>  
  );  
}
```

JSX 문법

- 컴포넌트에는 **최상위 요소가 단 하나만 존재**해야 한다
- 즉 React의 각 컴포넌트가 하나의 루트 엘리먼트를 반환해야 한다

```
jsxEx1.jsx U x App00.js U index.js M LifeCycle.jsx U
src > example > jsxEx1.jsx > jsxEx1
1 import React from 'react'
2 // root 엘리먼트는 반드시 한개 여애 한다.
3 export default function jsxEx1() {
4   return [
5     //
6     <div>
7       <h1>JSX 규칙 - root 는 1개여야 함</h1>
8       <h2>시작태그와 종료태그가 쌍으로 있어야 한다.</h2>
9       <br></br>
10      </img>
11      <br/>
12      <hr/>
13      { /* 루트 element */ }
14    </div>
15    // <div>여기에 있으면 에러남. </div>
16  ]
17 }
18
```

```
App.js
export default function App() {
  return (
    <fragment>
      <h1>Hello, World!</h1>
      <p>React도 안녕!</p>
    </fragment>
  );
}

최종 렌더링된 HTML 코드
<div id="root">
  <h1>Hello, World!</h1>
  <p>React도 안녕!</p>
</div>
```

React v16부터는
루트로
<div>요소를
불필요하게
사용하는 것을
재고할 수 있도록
Fragment라는
기능을 추가하였다
fragment를
생각하고
<></>식으로
축약하여 사용해도
된다.

JSX 문법

- 자바스크립트 표현식은 중괄호-{} 로 감싸져야 한다.

```
export default function App() {  
  const name = "홍길동";  
  return <h1>Hello, {name}</h1>;  
}
```

- jsx주석은 `{/* */}` 또는 `{// }` 식으로 사용할 수 있다.

잘못된 코드

```
export default function App() {  
  return (  
    <>  
      <h1>Hello, World!</h1>  
      {  
        // 한 줄 주석  
      }  
    </>  
  );  
}
```

// 이후는 모두 주석으로 처리하므로
{
 // 한줄 주석
}하면 닫는 중괄호도 주석
처리된다. 주의하자

{
 // 이렇게 주석을 작성하던지,
}

{/*이렇게 주석을 작성해야만 합니다.*/}

```
export default function App() {  
  return (  
    <>  
      {  
        /* 주석의 내용은 렌더링되지 않습니다. */  
        /* 중괄호로 감싸지 않으면 주석의 내용이 그대로 렌더링됩니다. */  
        <h1>Hello, World!</h1>  
        {  
          // 한 줄 주석  
        }  
        <p>React도 안녕!</p>  
        {  
          /* 여러 줄  
            주석 */  
        }  
      }  
    </>  
  );  
}
```

JSX에서의 DOM 속성

- html에서는 css클래스를 사용하기 위해 아래와 같이 사용한다

```
<div class="container"></div>
```

- 하지만 JSX에서는 **class**대신 **className**이라는 속성을 사용해야 한다. **class**가 keyword이기 때문에 사용할 수 없다

```
<div className="container"></div>
```

- label에서 사용되는 **html**의 **for**속성도 for루프문 키워드와 중복되므로 **htmlFor**로 바꿔 사용한다

```
export default function App() {  
  return (  
    <>  
      <label htmlFor="name">이름 : </label>  
      <input id="name" type="text"></input>  
    </>  
  );  
}
```

컴포넌트 (component)

- 리액트의 컴포넌트란 자바스크립트의 함수 개념과 비슷하다.
- UI를 설계할 때 사용자가 볼 수 있는 화면을 여러 개의 컴포넌트로 나누어 구성함으로써 각각의 컴포넌트를 개별적으로 관리할 수 있다.
- 컴포넌트는 단순히 재사용 가능한 템플릿 역할 뿐 아니라 데이터(props)를 입력 받아 View의 상태(state)에 따라 화면에 어떻게 표시되는지 정의하는 React 엘리먼트를 반환한다.
-
- 리액트는 선언하는 방식에 따라
 - 1. 클래스 기반 컴포넌트
 - 2. 함수 기반 컴포넌트
- 로 구분할 수 있다.

컴포넌트 (component)

클래스 기반의 App 컴포넌트

```
import React from "react";  
export default class App extends React.Component {  
  render() {  
    return <h1>Hello, World!</h1>;  
  }  
}
```

함수 기반의 App 컴포넌트

```
export default function App() {  
  return (  
    <div>  
      <h1>Hello, World!</h1>  
    </div>  
  );  
}
```

React에서 이 두 가지 유형의 컴포넌트는 기능적으로는 동일하게 동작한다. 단지 클래스 컴포넌트는 state 기능과 생명 주기 메소드를 사용할 수 있지만, 함수 컴포넌트에서는 해당 기능들을 사용할 수 없다는 차이점만 있다.

하지만 함수 컴포넌트는 클래스 컴포넌트보다 선언하는 방법이 훨씬 간결하고, 메모리의 소비 또한 적다는 장점을 가진다.

그리고 state 기능과 생명 주기 메소드의 사용이 불가능하다는 단점 또한 v16.8부터 새롭게 추가된 Hook을 사용하여 더 이상 문제가 되지 않게 되었다.

모듈 export

React에서는 import문, export문을 이용하여 모듈을 가져오고, 내보내기를 한다.

React에서 모듈 두 가지 유형

1. 하나의 컴포넌트만 선언되어 있는 모듈

2. 복수의 컴포넌트가 선언되어 있는

라이브러리 형태의 모듈

모듈이란 ?

- 모듈은 일반적으로 **재사용 가능한 코드 조각**을 가리킵니다.
- 모듈은 보통 **.js** 확장자를 가진 파일로 이루어지며, **특정 기능이나 컴포넌트를 정의하고 내보내는 역할**을 합니다

선언과 동시에 내보내기

```
export default function App() {  
  return <Greeting />;  
}
```

선언과 별도로 내보내기

```
function App() {  
  return <Greeting />;  
}  
  
export default App;
```

모듈 export

하나의 파일에 여러 개의 컴포넌트를
선언하고 내보내고 싶다면

각각 function 앞에 export를 하던지

아니면 오른쪽 과 같이

export { 컴포넌트1, 컴포넌트2 }

식으로 내보낼 수 있다.

예제(Laptop.js)

```
function GetBrand() {  
  return <h1>내 노트북은 Samsung 노트북입니다.</h1>;  
}  
  
function GetOS() {  
  return <h2>내 노트북의 OS는 Windows입니다.</h2>;  
}  
  
export { GetBrand, GetOS };
```

export default

- **default** 키워드를 이용해 ‘해당 파일에서는 하나의 컴포넌트 만을 내보낸다’는 의미를 나타낼 수 있다.
- **export default**로 내보낸 컴포넌트는 중괄호 {}를 사용하지 않고 **import**할 수 있다
- 이름 또한 원하는 이름으로 바꿔서 불러 올 수 있다.

```
export default function GetBrand() {  
  return <h1>내 노트북은 Samsung 노트북입니다.</h1>;  
}
```

```
export function GetOS() {  
  return <h2>내 노트북의 OS는 Windows입니다.</h2>;  
}
```



```
import Brand from "./Laptop";  
  
export default function App() {  
  return <Brand />;  
}
```

모듈 import

- import 키워드와 중괄호 {} 를 사용하여 다른 파일의 특정 컴포넌트를 불러올 수 있다. **import**될때 사용되는 이름은 원래 파일에서 선언된 이름을 그대로 사용해야 한다.

```
import { GetBrand, GetOS } from "../Laptop";
```

```
export default function App() {  
  return (  
    <>  
    <GetBrand />  
    <GetOS />  
  </>  
);  
}
```

예제(App.js)

```
import * as laptop from "../Laptop";
```

```
export default function App() {  
  return (  
    <>  
    <laptop.GetBrand />  
    <laptop.GetOS />  
  </>  
);  
}
```

만약 불러올 컴포넌트의 개수가 많다면 'import * as **<object>**' 구문을 사용하여 객체 형태로 원하는 컴포넌트들을 한 번에 불러올 수도 있다.

하지만 이 경우에는 사용할 때 '**laptop.**'이라는 코드가 추가되어야 하므로, 코드가 좀 더 복잡해진다.

다른 이름(별칭)으로 import하기

- as 라는 키워드를 사용하면 원래 파일에서 선언한 이름과 다른 이름으로 가져올 수 있다.

```
import { GetBrand as Brand, GetOS as OS } from "../Laptop";
```

```
export default function App() {  
  return (  
    <>  
    <Brand />  
    <OS />  
    </>  
  );  
}
```

props와 state

- props 와 state 는 컴포넌트를 렌더링하기 위한 입력 데이터다
- 컴포넌트는 props나 state가 바뀌면 다시 렌더링 된다.

React에서 props를 설정하는 방법은 HTML 요소에 속성을 설정하는 문법과 동일하다.

오른쪽을 보면 부모(App)가 자식 컴포넌트 Laptop에 데이터를 전달하고 싶다면 props를 설정하여 전달할 수 있다.

이를 자식 컴포넌트에서는 매개변수(인수) props로 받아 사용한다.

App.js

```
const Laptop = (props) => {  
  return <h1>내 노트북은 {props.brand} 노트북입니다.</h1>;  
};  
  
const App = () => {  
  return <Laptop brand="Samsung" />;  
};  
  
export default App;
```

props의 특징

props: prop의 복수형. prop은 property(속성)을 의미한다.

리액트 컴포넌트의 속성을 말한다

- props는 **readonly**
- **수정불가**. 정적 컴포넌트를 위한 것.
- props를 사용하는 쪽에서는 값을 전달할 수 있으나
- 전달받은 컴포넌트 쪽에서 props를 변경하려고
- 하면 오류 남.
- 컴포넌트 밖에서 props를 변경하는 것은 당연히 가능.
- 컴포넌트 내에서는 수정 불가

props----> 컴포넌트(state) ---->DOM
state는 내부적으로 사용됨

props의 특징

- 상위 컴포넌트가 하위 컴포넌트에 값을 전달할 때는
- **props**를 통해 전달한다.
- 하지만 하위 컴포넌트가 상위 컴포넌트의 값을 바꾸고자 할 때는 **event**를 통해 함.
- 그 이벤트가 실행되었을 때
- 상위 컴포넌트의 **state**를 변경하는 것을 통해 영향을 미침

state

- 컴포넌트의 내부 상태를 나타내는 객체
- 유동적 데이터. 쓰기 가능한 데이터
- 컴포넌트의 상태(state)는 컴포넌트가 생성될 때 초기화되고, 사용자의 상호작용에 따라 변경될 수 있다.
- React 컴포넌트는 상태(state)에 따라 동적으로 UI를 렌더링하고 업데이트할 수 있다
-

state를 사용하는 이유

state를 사용하는 이유는 다음과 같습니다:

1. 컴포넌트의 동적인 데이터 관리: 상태를 사용하면 컴포넌트 내에서 동적인 데이터를 저장하고 관리할 수 있습니다. 이를 통해 사용자 입력, 서버 응답 등에 따라 데이터를 업데이트하고 UI를 업데이트할 수 있습니다.
2. 컴포넌트의 렌더링 조건에 따른 UI 변경: 상태를 사용하여 컴포넌트의 렌더링 조건을 제어하고, 상태가 변경될 때마다 UI를 다시 렌더링할 수 있습니다.
3. React의 재렌더링 최적화: React는 상태가 변경될 때마다 자동으로 컴포넌트를 재렌더링합니다. 상태를 사용하여 필요한 경우에만 컴포넌트를 업데이트하고 재렌더링할 수 있습니다.

state는 클래스형 컴포넌트와 함수형 컴포넌트에서 모두 사용할 수 있습니다. 클래스형 컴포넌트에서는 `this.state`로 접근하고, `this.setState()` 메서드를 사용하여 상태를 업데이트합니다. 함수형 컴포넌트에서는 React Hooks를 사용하여 상태를 관리할 수 있습니다. 상태는 컴포넌트의 생명주기에 따라 변할 수 있으며, 변경된 상태는 자동으로 UI에 반영됩니다.

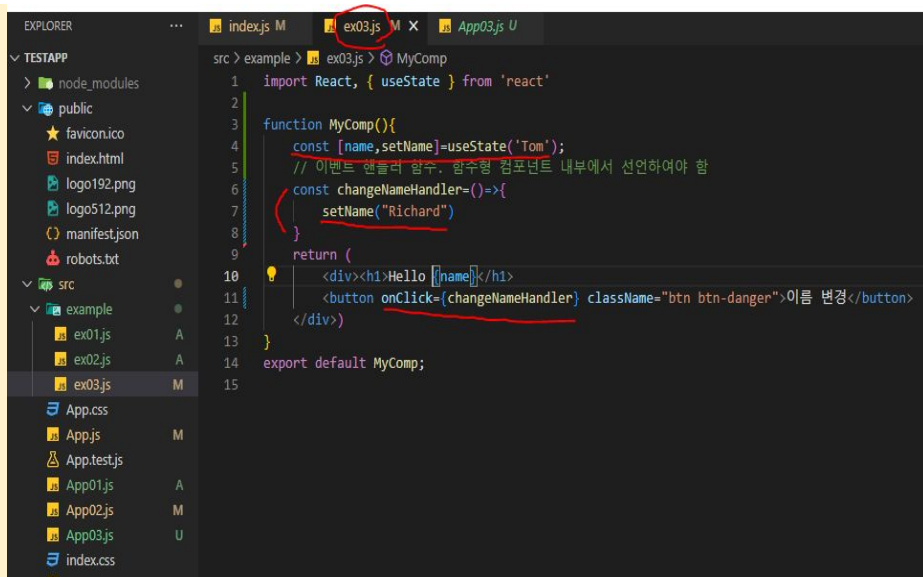
state - useState 혹은 사용

- state는 **useState()** 후크로 만든다. 이 함수는 상태의 초기값을 인수로 받고, 두 요소가 들어있는 배열을 반환한다.

```
const  
[state,setState]=React.useState(초기값);
```

```
ex)  
import {useState} from 'react';  
...중략...  
const [name,setName]=useState('홍길동');
```

이름을 변경해야 할때 setName("이영희")를 호출하여 홍길동 상태값을 수정 할 수 있다.
상태값을 업데이트하는 유일한 방법이다.

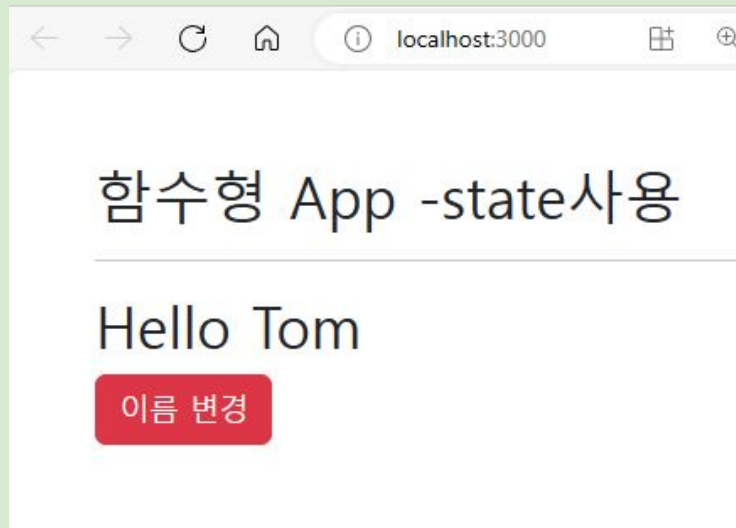


The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'src' directory containing 'example' and 'ex03.js'. The code editor shows the content of 'ex03.js', which is a React component named 'MyComp'. The component uses the 'useState' hook to manage a state variable 'name' with an initial value of 'Tom'. A button is rendered with an 'onClick' event handler 'changeNameHandler' that calls 'setName' to update the state to 'Richard'. The button's 'className' is 'btn btn-danger' and its text is '이름 변경'.

```
1 import React, { useState } from 'react'  
2  
3 function MyComp(){  
4   const [name,setName]=useState('Tom');  
5   // 이벤트 핸들러 함수. 함수형 컴포넌트 내부에서 선언하여야 함  
6   const changeNameHandler=()=>{  
7     setName("Richard")  
8   }  
9   return (  
10    <div><h1>Hello {name}</h1>  
11    <button onClick={changeNameHandler} className="btn btn-danger">이름 변경</button>  
12  </div>  
13  )  
14  export default MyComp;  
15
```

state 변경 결과

초기화면. 버튼 클릭시 ⇒



함수형 App -state사용

Hello Richard

이름 변경

클릭
27

state 사용 2 - Count 증가

다.

```
import React, { useState } from 'react';

const Counter = () => {
  // useState 혹은 사용하여 count 상태와 setCount 함수를 선언
  const [count, setCount] = useState(0);

  // 버튼 클릭 시 count 상태를 업데이트하는 함수
  const incrementCount = () => {
    setCount(count + 1); // setCount를 호출하여 count를 1 증가
  };

  return (
    <div>
      <h2>카운터</h2>
      <p>현재 카운트: {count}</p>
      { /* 버튼 클릭 시 incrementCount 함수 호출 */ }
      <button onClick={incrementCount}>증가</button>
    </div>
  );
}

export default Counter;
```

localhost:3000

함수형 App -state사용 Count 증가

카운터

현재 카운트: 2

증가

버튼을 누를때마다
count 상태값이 1씩
증가되면서 화면이
다시 렌더링된다.
상태 업데이트는
비동기적으로
수행된다.

일반 데이터와 state데이터의 차이

```
index.js M  ex03.js M  ex04.js U  순서.txt U  ex08.js U  ex05.js U  App.css

src > example > ex05.js > MyComp
1  import { useState } from 'react'
2  export default function MyComp(){
3
4  let bgcolor='orange'; //그냥 데이터
5
6  const [mycolor, setMyColor] = useState('green');//state 데이터
7  const [ flag, setFlag] = useState(false);//state 데이터
8
9  const handleClick1={()=>{
10
11    bgcolor='skyblue'
12    console.log('bgcolor: ',bgcolor)
13    //bgcolor를 변경해도 화면이 렌더링되지는 않음
14  }
15  function handleClick2(){
16
17    let cr=!flag? 'tomato':'green';
18    //setMyColor('yellow')
19    setMyColor(cr);
20    setFlag(!flag)
21    //setState계열의 함수가 호출되면 MyComp를 다시 렌더링 한다
22  }
23
24  return (
25    <div className="container py-5">
26      <button className="btn m-1" style={{backgroundColor:bgcolor}}
27        onClick={handleClick1}>좋아요1 /배경색: {bgcolor}</button>
28      <button className="btn" style={{backgroundColor:mycolor,color:'#fff'}}
29        onClick={handleClick2}>좋아요2 /배경색: {mycolor}</button>
30    </div>
31  )
32
33 }
```

useState혹 사용

클릭해도 버튼의 배경색이 변경되지 않는다

좋아요1 /배경색: orange

좋아요2 /배경색: green

bgcolor: skyblue

> |

일반 데이터와 state데이터의 차이

```
index.js M ex03.js M ex04.js U 순서.txt U ex08.js U ex05.js U App
src > example > ex05.js > MyComp
1 import { useState } from 'react'
2 export default function MyComp(){
3
4   let bgcolor='orange';//그냥 데이터
5
6   const [mycolor, setMyColor] = useState('green');//state 데이터
7   const [ flag, setFlag] = useState(false);//state 데이터
8
9   const handleClick1=()=>{
10
11     bgcolor='skyblue'
12     console.log('bgcolor: ',bgcolor)
13     //bgcolor를 변경해도 화면이 렌더링되지는 않음
14   }
15   function handleClick2(){
16
17     let cr=!flag? 'tomato':'green';
18     //setMyColor('yellow')
19     setMyColor(cr);
20     setFlag(!flag)
21     //setState개별의 함수가 호출되면 MyComp를 다시 렌더링 한다
22   }
23
24   return (
25     <div className="container py-5">
26       <button className='btn m-1' style={{backgroundColor:bgcolor}}
27         onClick={handleClick1}>좋아요1 /배경색: {bgcolor}</button>
28       <button className='btn' style={{backgroundColor:mycolor,color:'#fff'}}
29         onClick={handleClick2}>좋아요2 /배경색: {mycolor}</button>
30     </div>
31   )
32
33 }
```

useState혹 사용

좋아요1 /배경색: orange

좋아요2 /배경색: green

처음에는 green색상이다가 클릭하면
tomato색상으로 변경되고 또 클릭하면
다시 green 으로 변경(toggle기능)된다

좋아요1 /배경색: orange

좋아요2 /배경색: tomato

↑
클릭시

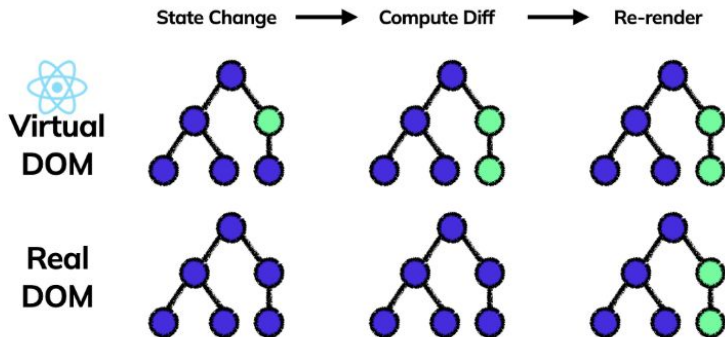
React의 DOM 관리

React 는 UI 상태를 추적하고 변화가 일어난 요소들을 빠르게 업데이트 할 수 있도록

Virtual DOM 이라는 가상의 DOM 객체를 활용한다.

이는 실제 DOM 사본 같은 개념으로, React 는 실제 DOM 객체에 접근하여 조작하는 대신 이 가상의 DOM 객체에 접근하여 변화 전과 변화 후를 비교하고 바뀐 부분을 적용한다

리액트는 diffing 알고리즘을 통해 두개의 가상 DOM 중 이전 상태와 현재 상태의 Virtual DOM 을 비교합니다.
이 비교과정에서는 React는 diffing 알고리즘을 사용하여 변경된 부분을 감지합니다.
따라서, React에서 상태를 변경하는 경우에는 diffing 알고리즘에서 이를 감지할 수 있도록 직접할당이 아닌 setState와 같은 메서드를 활용해 상태를 변경합니다.
그리고 가상 DOM과 새로운 가상DOM 을 비교하여 변경이 필요한 부분만 실제 DOM 에 반영하여 업데이트 합니다.
이 과정에서 여러개의 상태 변화가 있을 경우 이를 일일이 수행하지 않고 일괄적으로 한번에 업데이트 합니다.
이를 통해 성능을 최적화하고 불필요한 리렌더링을 최소화할 수 있습니다



결론

리엑트는 상태값 (**state**)을 기반으로 **UI**를 정의하며, **state**값이 수정되면 화면을 갱신하게 된다.

리엑트에서 UI데이터는 반드시 **state**와 **props**값으로 관리해야 한다. 그렇지 않으면 데이터가 변경 돼도 화면이 업데이트 되지 않을 수 있다.

- **state**: 쓰기 가능한 데이터 (setXXX()함수로 변경 가능). 동적으로 변경할 수 있는 데이터들을 기술한다.
- **props**: 읽기 전용 데이터 (변경 불가-readonly). 정적인 데이터들을 부모가 자식에게 전달할때 사용한다.

참고 사이트

react 17 과 18버전 차이점

<https://tech.nexr.kr/e14cd3fa-58be-4983-9e54-a1d157af08c2>

[코딩의 시작, TCP School](#)

<https://shape-coding.tistory.com/entry/SPA-vs-MPA-%EA%B0%9C%EB%85%90%EC%A0%95%EB%A6%AC-%EB%B0%8F-%EC%9E%A5%EB%8B%A8%EC%A0%90>